

Python® for R Users

Python® for R Users

A Data Science Approach

Ajay Ohri

WILEY

This edition first published 2018
© 2018 John Wiley & Sons, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by law. Advice on how to obtain permission to reuse material from this title is available at <http://www.wiley.com/go/permissions>.

The right of Ajay Ohri to be identified as the author of this work has been asserted in accordance with law.

“Python” and the Python Logo are trademarks of the Python Software Foundation.

Registered Office

John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, USA

Editorial Office

111 River Street, Hoboken, NJ 07030, USA

For details of our global editorial offices, customer services, and more information about Wiley products visit us at www.wiley.com.

Wiley also publishes its books in a variety of electronic formats and by print-on-demand. Some content that appears in standard print versions of this book may not be available in other formats.

Limit of Liability/Disclaimer of Warranty

The publisher and the authors make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties; including without limitation any implied warranties of fitness for a particular purpose. This work is sold with the understanding that the publisher is not engaged in rendering professional services. The advice and strategies contained herein may not be suitable for every situation. In view of on-going research, equipment modifications, changes in governmental regulations, and the constant flow of information relating to the use of experimental reagents, equipment, and devices, the reader is urged to review and evaluate the information provided in the package insert or instructions for each chemical, piece of equipment, reagent, or device for, among other things, any changes in the instructions or indication of usage and for added warnings and precautions. The fact that an organization or website is referred to in this work as a citation and/or potential source of further information does not mean that the author or the publisher endorses the information the organization or website may provide or recommendations it may make. Further, readers should be aware that websites listed in this work may have changed or disappeared between when this work was written and when it is read. No warranty may be created or extended by any promotional statements for this work. Neither the publisher nor the author shall be liable for any damages arising here from.

Library of Congress Cataloguing-in-Publication Data

Name: Ohri, A. (Ajay), author.

Title: Python® for R users : a data science approach / Ajay Ohri.

Description: Hoboken, NJ : John Wiley & Sons, 2018. | Includes bibliographical references and index. |

Identifiers: LCCN 2017022045 (print) | LCCN 2017036415 (ebook) |

ISBN 9781119126775 (pdf) | ISBN 9781119126782 (epub) |

ISBN 9781119126768 (pbk.)

Subjects: LCSH: Python (Computer program language) | R (Computer program language)

Classification: LCC QA76.73.P98 (ebook) | LCC QA76.73.P98 O37 2017 (print) |

DDC 005.13/3—dc23

LC record available at <https://lcn.loc.gov/2017022045>

Cover design: Wiley

Cover images: (Background) © Duncan Walker/iStockphoto

Set in 10/12pt Warnock by SPi Global, Pondicherry, India

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Dedicated to my family in Delhi, Mumbai, and the United States

and

Kush Ohri (my son whom I love very much)

and

Jesus Christ (my personal savior)

Contents

Preface	<i>xi</i>
Acknowledgments	<i>xv</i>
Scope	<i>xvii</i>
Purpose	<i>xix</i>
Plan	<i>xxi</i>
The Zen of Python	<i>xxiii</i>

1	Introduction to Python R and Data Science	1
1.1	What Is Python?	1
1.2	What Is R?	2
1.3	What Is Data Science?	3
1.4	The Future for Data Scientists	3
1.5	What Is Big Data?	4
1.6	Business Analytics Versus Data Science	6
1.6.1	Defining Analytics	6
1.7	Tools Available to Data Scientists	7
1.7.1	Guide to Data Science Cheat Sheets	7
1.8	Packages in Python for Data Science	8
1.9	Similarities and Differences between Python and R	9
1.9.1	Why Should R Users Learn More about Python?	10
1.9.2	Why Should Python Users Learn More about R?	10
1.10	Tutorials	10
1.11	Using R and Python Together	11
1.11.1	Using R Code for Regression and Passing to Python	11
1.12	Other Software and Python	15
1.13	Using SAS with Jupyter	15
1.14	How Can You Use Python and R for Big Data Analytics?	15
1.15	What Is Cloud Computing?	16
1.16	How Can You Use Python and R on the Cloud?	17

1.17	Commercial Enterprise and Alternative Versions of Python and R	18
1.17.1	Commonly Used Linux Commands for Data Scientists	20
1.17.2	Learning Git	20
1.18	Data-Driven Decision Making: A Note	38
1.18.1	Strategy Frameworks in Business Management: A Refresher for Non-MBAs and MBAs Who Have to Make Data-Driven Decisions	39
1.18.2	Additional Frameworks for Business Analysis	45
	Bibliography	49
2	Data Input	51
2.1	Data Input in Pandas	51
2.2	Web Scraping Data Input	54
2.2.1	Request Data from URL	55
2.3	Data Input from RDBMS	60
2.3.1	Windows Tutorial	62
2.3.2	137 Mb Installer	63
2.3.3	Configuring ODBC	65
3	Data Inspection and Data Quality	77
3.1	Data Formats	77
3.1.1	Converting Strings to Date Time in Python	78
3.1.2	Converting Data Frame to NumPy Arrays and Back in Python	81
3.2	Data Quality	84
3.3	Data Inspection	88
3.3.1	Missing Value Treatment	91
3.4	Data Selection	92
3.4.1	Random Selection of Data	94
3.4.2	Conditional Selection	95
3.5	Data Inspection in R	98
3.5.1	Diamond Dataset from ggplot2 Package in R	106
3.5.2	Modifying Date Formats and Strings in R	113
3.5.3	Managing Strings in R	116
	Bibliography	118
4	Exploratory Data Analysis	119
4.1	Group by Analysis	119
4.2	Numerical Data	119
4.3	Categorical Data	121

5	Statistical Modeling	139
5.1	Concepts in Regression	139
5.1.1	OLS	140
5.1.2	R-Squared	141
5.1.3	p-Value	141
5.1.4	Outliers	141
5.1.5	Multicollinearity and Heteroscedascity	142
5.2	Correlation Is Not Causation	142
5.2.1	A Note on Statistics for Data Scientists	143
5.2.2	Measures of Central Tendency	145
5.2.3	Measures of Dispersion	145
5.2.4	Probability Distribution	147
5.3	Linear Regression in R and Python	154
5.4	Logistic Regression in R and Python	187
5.4.1	Additional Concepts	194
5.4.2	ROC Curve and AUC	194
5.4.3	Bias Versus Variance	194
	References	196
6	Data Visualization	197
6.1	Concepts on Data Visualization	197
6.1.1	History of Data Visualization	197
6.1.2	Anscombe Case Study	200
6.1.3	Importing Packages	201
6.1.4	Taking Means and Standard Deviations	202
6.1.5	Conclusion	204
6.1.6	Data Visualization	204
6.1.7	Conclusion	207
6.2	Tufte's Work on Data Visualization	207
6.3	Stephen Few on Dashboard Design	208
6.3.1	Maeda on Design	209
6.4	Basic Plots	210
6.5	Advanced Plots	219
6.6	Interactive Plots	223
6.7	Spatial Analytics	223
6.8	Data Visualization in R	224
6.8.1	A Note of Sharing Your R Code by RStudio IDE	232
6.8.2	A Note on Sharing Your Jupyter Notebook	233
	Bibliography	235
6.8.3	Special Note: A Complete Wing to Wing Tutorial on Python	236

- 7 Machine Learning Made Easier 251**
 - 7.1 Deleting Columns We Dont Need in the Final
Decision Tree Model 259
 - 7.1.1 Decision Trees in R 276
 - 7.2 Time Series 294
 - 7.3 Association Analysis 301
 - 7.4 Cleaning Corpus and Making Bag of Words 316
 - 7.4.1 Cluster Analysis 319
 - 7.4.2 Cluster Analysis in Python 319

- 8 Conclusion and Summary 331**

- Index 333**

Preface

I started my career with selling cars in 2003. That was my first job after 2 years of MBA and 4 years of engineering. In addition, I took off 2 years to enter a military academy as an officer cadet (dropped out in 1 year) and as a physicist (dropped out after 1 year). Much later, I dropped out of my PhD Track (MS Stats) after 1 year in Knoxville. I did not do very well in statistics theory in my engineering, my MBA, or even my grad school. I was only interested in statistical software and fortunately I was not very bad at using it. So in 2004, I dropped out of selling cars and entered into writing statistical software for General Electric's then India-based offshore company.

I used a language called SAS for a software called Base SAS. The help provided by the software company called SAS for this software and language was quite nice, so it was nice to play with data and code all day and be paid to have fun. After a few years of job changes, I came across open-source software when I started building my own start-up. I really like SAS as a language and a company, but as a start-up guy I could not afford it, and the SAS University Edition was not there in 2007. Since I needed money to pay for diapers of my baby Kush, and analysis was the only gift God had given me, I turned to R.

R, Open Office, and Ubuntu Linux were my first introduction to open-source statistical computing, and I persevered in it. In 2007 I started my own start-up in business analytics writing and consulting, Decisionstats.com. In 2009 I entered the University of Tennessee for a funded assistantship, I interned in Silicon Valley for a few weeks in the winter, and I dropped out on medical reasons after taking courses across multiple departments from graphics design and genetic algorithms from Computer Science Department, apart from Statistics Department. Cross-domain training helped me a lot to think in various ways to give simple solutions, and I will always be thankful to the kind folks in Statistics and Computer Science Department of the University of Tennessee.

Once I mastered my brain around the vagaries of troubleshooting in Linux and of object-oriented programming on R, I was good to go to give consulting projects for data analysis. Those days we used to call it business analytics, but today of course we call it data science.

Since I often forget things including where I kept my code, I started blogging on things that I felt were useful and might be useful to others. After a few years I discovered that in the real world it was not what I knew, but who I knew that really helped my career. So I began interviewing people in Analytics and R and my blog viewership took off. My blog philosophy continues to be—a blog post should be useful, it should be unique, and it should be interesting. In 2016, I had amassed 1,000,000 views on DecisionStats.com—again a surprising turn of events for me. I am most grateful to the 100 plus people who agreed to be interviewed by me.

2007 and 2008 were early days for analytics blogging for sure. After a few years I had enough material to put together a book and enough credibility to publish with a publisher. In 2012 I came up with my first book and in 2014 I came up with my second book. In 2016, the Chinese translation of my first book was realized. Surprisingly for me, a review of my second book appeared in the *Journal of Statistical Software*.

After publishing two books on R, mentoring many start-ups by consulting and training, engaging consulting clients in real-world problems, and making an established name in social media, I still felt I needed to learn more.

Data was getting bigger and bigger. It was not enough to know how to write small data analytics using a single machine in serialized code; perhaps it was time to write parallel code in multiple machines on big data analytics. Then there was the divide between statisticians and computer science that fascinated me since I see data as data, a problem to be solved. As Eric S. Raymond wrote in the Hacker's attitude, "The world is full of interesting problems."

Then there was temptation and intellectual appeal of an alternative to R, called Python, which came with batteries attached (allegedly).

Once my scientific curiosity was piqued, I started learning Python. I found Python was both very good and very bad compared with R. Its community has different sets of rules and behavior (which are always turbulent in the passionate world of open-source developer ninjas). But the language itself was very different. I don't care about the language. I love science. But if a person like me who at least knows how to code a wee bit in R found it so tough to redo the same thing in Python, I thought maybe others were facing this transitioning problem too. For big data and for some specific use cases, Python was better in terms of speed. Speed matters, no matter how much Moore's law conspires with the either to make it easier for you to write code. R also seemed to turn into a language where all I did was import a package and run a function with tweaked parameters. As R became the scientific mainstream replacing SAS

language, and SAS remained the enterprise statistical language, Python and how to write code in it became the thing for anonymous red hat hackers like me to venture delve and explore into.

As the Internet of people expands to Internet of things, I feel that budding data scientists should know at least two languages in analytics so they can be secure on career. This also gives enterprises an open choice on which software to prototype models and which software to deploy in production environments.

Acknowledgments

The author is grateful to many people working in both the Python and R community for making this book possible. He would especially like to thank Dr. Eric Siegel of Predictive Analytics Conference and John Sall of JMP. He would like to thank all his students in 2012–2016.

This book would not be done without the support from Madhur Batra for mentoring and logistical support. On a technical side, inputs and hard work from his interns Yashika and Chandan Routray (IIT Kharagpur) and his DecisionStats team helped him. His coresearcher F. Xavier provided invaluable help with case studies.

Scope

The scope of the book is to introduce Python as a platform for data science practitioners including aspiring budding data scientists. The book is aimed at people who know R coding at various levels of expertise, but even those who know no coding in no language may find some value in it. It is not aimed at members of research communities and research departments. The focus is on simple tutorials and actionable analytics, not theory. I have also tried to incorporate R code to give a compare and contrast approach to learners.

Chapter 1

Introduction deals with Python and comparison with R. It also lists the functions and packages used in both languages. It also lists some managerial models that the author feels data scientists should be aware of. It introduces the reader to basics of Python and R language.

Chapter 2

“Data Input” deals with an approach for people to get data of various volume variety and velocity in Python. This includes web scraping, databases, noSQL data, and spreadsheet like data.

Chapter 3

“Data Inspection and Data Quality”—Data Inspection deals with choices in verifying data quality in Python.

Chapter 4

“Exploratory Data Analysis” deals with basic data exploration and data summarization with rolling up data with group by criterion.

Chapter 5

“Statistical Modeling” deals with creating models based on statistical analysis including OLS regression that are useful for industry to build propensity models.

Chapter 6

“Data Visualization” deals with visual methods to inspect raw and rolled-up data.

Chapter 7

“Machine Learning Made Easier” deals with commonly used data mining methods for model building. This is done with an emphasis on both supervised and unsupervised methods and further emphasis on regression and clustering techniques. Time series forecasting helps the user with time series forecasting. Text mining deals with text mining methods and natural language processing. Web analytics looks at using Python for analyzing web data. Advanced data science looks at methods and techniques for newer age use cases including cloud computing-enabled big data analysis, social network analysis, Internet of things, etc.

Chapter 8

Conclusion and Summary—We list down what we learned and tried to achieve in this book, and our perspective for future growth of R and Python as well as statistical computing to grow, and render data science a credible foothold for the future.

Purpose

The book has been written from a practical use case perspective for helping people navigate multiple open-source languages in the pursuit of data science excellence. The author believes that there is no one software or language that can solve all kinds of data problems all the time. An optimized approach to learning is better than an ideological approach to learning statistical software. Past habits of thinking must be confronted to enhance speed of future knowledge enhancement.

Plan

I will continue to use screenshots as a tutorial device and I will draw upon my experience in data science consulting to highlight practical data parsing problems. This is because choosing the right tool and technique and even package is not so time consuming but the sheer variety of data and business problems can suck up the data scientist's time that can later affect quality of his judgment and solution.

Intended Audience

This is a book for budding data scientists and existing data scientists married to other languages like SPSS or R or Julia. I am trying to be practical about solving problems in data. Thus there will be very little theory.

Afterthoughts

I am focused on practical solutions. I will therefore proceed on the assumption that the user wants to do data science or analytics at the lowest cost and greatest accuracy, robustness, and ease possible. A true scientist always keeps his mind open to data and options regardless of who made whom. The author finds that information asymmetry and brand clutter have managed to confuse audiences of the true benefits of R versus Python versus other languages. The instructions and tutorials within this book have no warranty and you are doing so at your own risk.

As a special note on formatting of this manuscript, the author mostly writes on Google Docs, but here he is writing using the GUI LyX for the typesetting software LaTeX, and he confesses he is not very good at it. We do hope the book is read by business users, technical users, CTOs keen to know more on R and Python and when to use open-source analytics, and students wishing to enter a very nice career as data scientists. R is well known for excellent graphics but

not so suitable for bigger datasets in its native straight to use open-source version. Python is well known for being great with big datasets and flexibility but has always played catch-up to the number of good statistical libraries as available in R.

The enterprise CTO can reduce costs incredibly by using open-source software and hardware via blended cloud and blended open-source software.

The Zen of Python

Tim Peters

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently. Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one—and preferably only one—obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never. Although never is often better than right now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea—let's do more of those!

Source: <https://www.python.org/dev/peps/pep-0020/>

1

Introduction to Python R and Data Science

1.1 What Is Python?

Python is a programming language that lets you work more quickly and integrate your systems more effectively. It was created by Guido van Rossum. You can read Guido's history of Python at the History of Python blog at <http://python-history.blogspot.in/2009/01/introduction-and-overview.html>.

It is worth reading for beginners and even experienced people in Python. The following is just an extract:

many of Python's keywords (if, else, while, for, etc.) are the same as in C, Python identifiers have the same naming rules as C, and most of the standard operators have the same meaning as C. Of course, Python is obviously not C and one major area where it differs is that instead of using braces for statement grouping, it uses indentation. For example, instead of writing statements in C like this

```
if (a < b) {  
    max = b;  
} else {  
    max = a;  
}
```

Python just dispenses with the braces altogether (along with the trailing semicolons for good measure) and uses the following structure:

```
if a < b:  
    max = b  
else:  
    max = a
```

The other major area where Python differs from C-like languages is in its use of dynamic typing. In C, variables must always be explicitly declared and given a specific type such as `int` or `double`. This information is then used to perform static compile-time checks of the program as well as for allocating memory locations used for storing the variable's value. In Python, variables are simply names that refer to objects.

The Python Package Index (PyPI) <https://pypi.python.org/pypi> hosts third-party modules for Python. There are currently **91 625** packages there. You can browse Python packages by topic at <https://pypi.python.org/pypi?%3Aaction=browse>

1.2 What Is R?

The official definition of what is R is given on the main website at <http://www.r-project.org/about.html>

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. It includes an effective data handling and storage facility, a suite of operators for calculations on arrays, in particular matrices, a large, coherent, integrated collection of intermediate tools for data analysis, graphical facilities for data analysis and display either on-screen or on hardcopy, and a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

The term 'environment' is intended to characterize it as a fully planned and coherent system, rather than an incremental accretion of very specific and inflexible tools, as is frequently the case with other data analysis software.

The Comprehensive R Archive Network (CRAN) hosts thousands of packages for R at <https://cran.r-project.org/web/packages/>, so does GitHub (see <https://github.com/search?utf8=%E2%9C%93&q=stars%3A%3E1+language%3AR>) as well as Bioconductor as package repositories. You can see all the packages from these repositories for R at <http://www.rdocumentation.org/> (11 885 packages as of 2016).

As per the author, R is both a language in statistics as well as computer science and an analytics software with great usefulness in analyzing business data and applying data science to it. In particular the appeal of R remains: it is a free open source and has a huge number of packages particularly dealing with analysis of data.

Disadvantages of R remain memory handling in production environments, lack of incentives for R developers, and a sometimes turgid documentation that is mildly academic oriented rather than enterprise user oriented.

1.3 What Is Data Science?

Data science lies at the intersection of programming, statistics, and business analysis. It is the use of programming tools with statistical techniques to analyze data in a systematic and scientific way. A famous diagram by Drew Conway put data science as the intersection of the three. It is given at <http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram>

The author defines a data scientist as follows:

A data scientist is simply a person who can **write code** (in languages like R, Python, Java, SQL, Hadoop (Pig, HQL, MR) etc.) **for data** (storage, querying, summarization, visualization) **efficiently** and **quickly on hardware** (local machines, on databases, on cloud, on servers) and **understand enough statistics** to derive **insights** from **data** so business can make **decisions**.

1.4 The Future for Data Scientists

The respectable *Harvard Business Review* defines data scientist to be the sexiest job of the twenty-first century (<https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century/>).

Surveys on salaries point out to both rising demand and salaries for data scientists and a big shortage for trained professionals (see <http://www.forbes.com/sites/gilpress/2015/10/09/the-hunt-for-unicorn-data-scientists-lifts-salaries-for-all-data-analytics-professionals/>). Indeed this has coined a new term unicorn data scientists. A unicorn data scientist is rare to find for he has all the skills in programming, statistics, and business aptitude. A modification of the Data Science Venn Diagram in Figure 1.1 is available at <http://www.anlytcs.com/2014/01/data-science-venn-diagram-v20.html>, which the author found more updated.

In addition, unicorn is a term in the investment industry, and in particular the venture capital industry, which denotes a start-up company whose valuation has exceeded \$1 billion. The term has been popularized by Aileen Lee of Cowboy Ventures. They can be seen at <http://graphics.wsj.com/billion-dollar-club/> and <http://fortune.com/unicorns/>

Not surprisingly data science offers a critical edge to these start-ups as well. So we can have both rising demand and short supply of data scientists, leading to a more secure work environment. A list of start-ups can be seen at Y Combinator at <http://yclist.com/> including data science related start-ups. You can see a survey here on data scientist salaries at <http://www.burtchworks.com/2015/07/14/compensation-of-data-scientists-insights-from-the-past-year>. The annual Rexer Analytics survey helps gauge skills and usage by data miners. You can read an interview at <http://decisionstats.com/2013/12/25/>

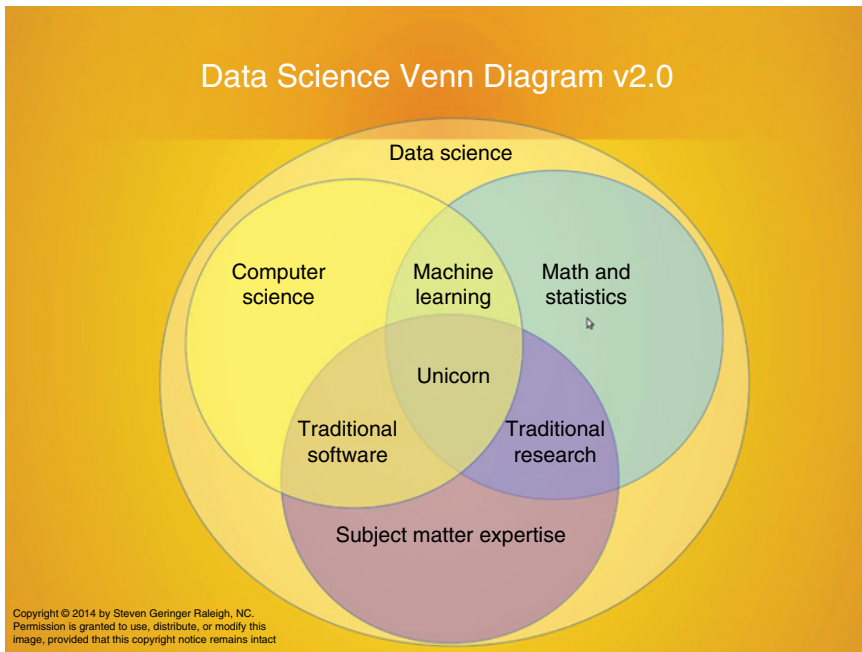


Figure 1.1 Data Science Venn diagram. *Source:* Copyright © 2014 Steven Geringer Raleigh, NC.

karl-rexer-interview-on-the-state-of-analytics/ or read the report at www.rexeranalytics.com. We can thus sum up and say that data scientists who have the right skills have a great future ahead professionally.

A note of caution is that skills need to be updated by data scientists very quickly and they need to be responsive to business needs to frame the data science solutions. So the risk of being obsolete remains an encouragement for data scientists to get multiple skills. An interesting fellowship program for data scientists is run by Insight at <http://insightdatascience.com/>, and a repository for data science is available for free at <https://github.com/okulbilisim/awesome-datascience>

Closer home, the NY-based Byte academy offers a Python-based program for data science at <http://byteacademy.co/>

1.5 What Is Big Data?

Big data is a broad term for datasets so large or complex that traditional data processing applications are inadequate. The 3Vs model helps with understanding big data.

These are:

- 1) Volume (size and scale of data)
- 2) Velocity (streaming or data refresh rate)
- 3) Variety (type: structured or unstructured) of data

The fourth V is veracity.

Typical approaches to deal with big data are hardware based, and use distributed computing, parallel processing, cloud computing, and specialized software like Hadoop stack. An interesting viewpoint to big data is given at <https://peadarcoyle.wordpress.com/2015/08/02/interview-with-a-data-scientist-hadley-wickham/> by Dr. Hadley Wickham, a noted R scientist:

There are two particularly important transition points:

- * From in-memory to disk. If your data fits in memory, it's small data. And these days you can get 1 TB of ram, so even small data is big! Moving from in-memory to on-disk is an important transition because access speeds are so different. You can do quite naive computations on in-memory data and it'll be fast enough. You need to plan (and index) much more with on-disk data
- * From one computer to many computers. The next important threshold occurs when you data no longer fits on one disk on one computer. Moving to a distributed environment makes computation much more challenging because you don't have all the data needed for a computation in one place. Designing distributed algorithms is much harder, and you're fundamentally limited by the way the data is split up between computers.

Wes McKinney, the author of pandas, the primary Python package for data science, has this to offer on <http://wesmckinney.com/blog/the-problem-with-the-data-science-language-wars/>

“any data processing engine that allows you to extend it with user-defined code written in a “foreign language” like Python or R has to solve at least these 3 essential problems:

- Data movement or access: making runtime data accessible in a form consumable by Python, say. Unfortunately, this often requires expensive serialization or deserialization and may dominate the system runtime. Serialization costs can be avoided by carefully creating shared byte-level memory layouts, but doing this requires a lot of experienced and well-compensated people to agree to make major engineering investments for the greater good.
- Vectorized computation: enabling interpreted languages like Python or R to amortize overhead and calling into fast compiled code that is

array-oriented (e.g. NumPy or pandas operations). Most libraries in these languages also expect to work with array / vector values rather than scalar values. So if you want to use your favorite Python or R packages, you need this feature.

- IPC overhead: the low-level mechanics of invoking an external function. This might involve sending a brief message with a few curt instructions over a UNIX socket.”

The author defines big data as data that requires more hardware (Cloud et al.) or more complicated programming or specialized software (Hadoop) than small data.

1.6 Business Analytics Versus Data Science

The author found the historical evolution from statistical computing to business analytics (BA) to data science both fascinating and amusing in the various claims of hegemonic superiority. This is how he explains it to his students and readers.

1.6.1 Defining Analytics

Analytics is the systematic computational analysis of data or statistics. It is the discovery and communication of meaningful patterns in data. Especially valuable in areas rich with recorded information, analytics relies on the simultaneous application of statistics, computer programming, and operations research to quantify performance.

The **information ladder** was created by education professor Norman Longworth to describe the stages in human learning. According to the ladder, a learner moves through the following progression to construct “wisdom” from “data”:

Data → Information → Knowledge → Understanding → Insight → Wisdom

BA refers to the skills, technologies, and practices for continuous iterative exploration and investigation of past business performance to gain insight and drive business planning.

Data analytics (DA) is the science of examining raw data with the purpose of drawing conclusions about that information.

Citation from <http://www.gartner.com/it-glossary/analytics>

Data science is a more recent term and implies much more programming complexity:

Data Science = programming + statistics + business knowledge

from <http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram>

Business intelligence (BI) is an umbrella term that includes the applications, infrastructure and tools, and best practices that enable access to and analysis of information to improve and optimize decisions and performance.

Overall the most important thing should be assistance to decision-making rendered not just the science of data analysis.

1.7 Tools Available to Data Scientists

Some (and not all) of the widely used tools available to data scientists are the following:

- Data storage—MySQL, Oracle, SQL Server, HBase, MongoDB, and Redis
- Data querying—SQL, Python, Java, and R
- Data analysis—SAS, R, and Python
- Data visualization—JavaScript, R, and Python
- Data mining—Clojure, R, and Python
- Cloud—Amazon AWS, Microsoft Azure, and Google Cloud
- Hadoop Big Data—Spark, HDFS MapReduce (Java), Pig, Hive, and Sqoop

A cheat sheet is a piece of paper bearing written notes intended to aid one's memory. It can also be defined as a compilation of mostly used commands to help you learn that language's syntax at a faster rate. To help with remembering syntax for many tools, cheat sheets can be useful for data scientists.

The author has written an article on KDnuggets on cheat sheets for data science at <http://www.kdnuggets.com/2014/05/guide-to-data-science-cheat-sheets.html> where he elaborates on his philosophy of what is a data scientist or not.

1.7.1 Guide to Data Science Cheat Sheets

Selection of the most useful Data Science cheat sheets, covering SQL, Python (including NumPy, SciPy, and Pandas), R (including Regression, Time Series, Data Mining), MATLAB, and more. By Ajay Ohri, May 2014.

Over the past few years, as the buzz and apparently the demand for data scientists has continued to grow, people are eager to learn how to join, learn, advance, and thrive in this seemingly lucrative profession. As someone who writes on analytics and occasionally teaches it, I am often asked—How do I become a data scientist?

Adding to the complexity of my answer is data science seems to be a multi-disciplinary field, while the university departments of statistics, computer science, and management deal with data quite differently.

But to cut the marketing created jargon aside, a data scientist is simply a person who can write code in a few languages (primarily R, Python, and SQL)

for data querying, manipulation, aggregation, and visualization using enough statistical knowledge to give back actionable insights to the business for making decisions.

Since this rather practical definition of a data scientist is reinforced by the accompanying words on a job website for “data scientists,” ergo, here are some tools for learning the primary languages in data science—Python, R, and SQL.

A cheat sheet or reference card is a compilation of mostly used commands to help you learn that language’s syntax at a faster rate. The inclusion of SQL may lead to some to feel surprised (isn’t this the NoSQL era?), but it is there for a logical reason. Both PIG and Hive Query Language are closely associated with SQL—the original Structured Query Language. In addition one can solely use the `sqldf` package within R (and the less widely used `python-sql` or `python-sql-parse` libraries for Pythonic data scientists) or even the Proc SQL commands within the old champion language SAS and do most of what a data scientist is expected to do (at least in data munging).

Python Cheat Sheets is a rather partial list given the fact that Python, the most general-purpose language within the data scientist quiver, can be used for many things. But for the data scientist, the packages of NumPy, SciPy, pandas, and scikit-learn seem the most pertinent.

Do all the thousands of R packages have useful interest to the aspiring data scientist? No.

Accordingly we chose the appropriate cheat sheets for you. Note that this is a curated list of lists. If there is anything that can be assumed in the field of data science, it should be that the null hypothesis is that the data scientist is intelligent enough to make his own decisions based on data and its context. Three printouts are all it takes to speed up the aspiring data scientist’s journey.

You can also view the presentation on SlideShare at <http://www.slideshare.net/ajayohri/cheat-sheets-for-data-scientists> that has more than 8000 views.

1.8 Packages in Python for Data Science

Some useful packages for data scientists in Python are as follows:

- **pandas**—A software library written for data structures, data manipulation, and analysis in Python.
- **NumPy**—Adds Python support for large, multidimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.
- **IPython Notebook(s)**—Demonstrates Python functionality geared toward data analysis.
- **SciPy**—A fundamental library for scientific computing.

- **Matplotlib**—A comprehensive 2D plotting for graphs and data visualization.
- **Seaborn**—A Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics.
- **scikit-learn**—A machine learning library.
- **statsmodels**—For building statistical models.
- **Beautiful Soup**—For web scraping.
- **Tweepy**—For Twitter scraping.
- **Bokeh** (<http://bokeh.pydata.org/en/latest/>)—A Python interactive visualization library that targets modern web browsers for presentation. Its goal is to not only provide elegant, concise construction of novel graphics in the style of D3.js but also deliver this capability with high-performance interactivity over very large or streaming datasets. It has interfaces in Python, Scala, Julia, and now R.
- **ggplot** (<http://ggplot.yhathq.com/>)—A plotting system for Python based on R's ggplot2 and the Grammar of Graphics. It is built for making professional-looking plots quickly with minimal code.

For R the best way to look at packages is see CRAN Task Views (<https://cran.r-project.org/web/views/>) where the packages are aggregated by usage type. For example, the CRAN Task View on High Performance Computing is available at <https://cran.r-project.org/web/views/HighPerformanceComputing.html>.

1.9 Similarities and Differences between Python and R

- Python is used in a wide variety of use cases unlike R that is mostly a language for statistics.
- Python has two versions: Python 2 (or 2.7) and Python 3 (3.4). This is not true in R that has one major release.
- R has very good packages in data visualization and data mining and so does Python. R however has a large number of packages that can do the same thing, while Python generally focuses on adding functions to same package. This is both a benefit in terms of options available and a disadvantage in terms of confusing the beginner. Python has comparatively fewer packages (like statsmodels and scikit-learn for data mining).
- Communities differ in terms of communication and interaction. The R community uses the #rstats on Twitter (see <https://twitter.com/hashtag/rstats>) to communicate.
- R has an R Journal at <https://journal.r-project.org/>, and Python has a journal at *Python Papers* (<http://ojs.pythonpapers.org/>). In addition there is a *Journal of Statistical Software* (<http://www.jstatsoft.org/index>).

1.9.1 Why Should R Users Learn More about Python?

A professional data scientist should hedge his career by not depending on just one statistical computing language. The ease at which a person can learn a new language does decrease with age, and it's best to base your career on more than R. SAS language did lead the world for four decades, but in a fast-changing world, it is best not to bet your mortgage that R skills are all you need for statistical computing in a multi-decade career.

1.9.2 Why Should Python Users Learn More about R?

R will continue to have the maximum number of packages in statistics data science and visualization. Since R is also open source and free, it is best to prototype your solution in R than use Python for scaling up in production environment.

An interesting viewpoint is given at <http://www.kdnuggets.com/2015/05/r-vs-python-data-science.html> by a founder of DataCamp and at <http://multithreaded.stitchfix.com/blog/2015/03/17/grammar-of-data-science/>

1.10 Tutorials

A notebook by Radim Rehurek on data science with Python with code and output is available at http://radimrehurek.com/data_science_python/.

A good list of notebooks in data science for Python is also available at <https://github.com/donnemartin/data-science-ipython-notebooks>.

More general knowledge on data science-related activities in Python can be found at <https://github.com/okulbilisim/awesome-datascience>.

For more learning on data science, see <http://datasciencespecialization.github.io/>. It has all nine courses in the Coursera Data Science Specialization from Johns Hopkins University.

It has the following courses:

- The Data Scientist's Toolbox
- R Programming
- Getting and Cleaning Data
- Exploratory Data Analysis
- Reproducible Research
- Statistical Inference
- Regression Models
- Practical Machine
- Learning Developing Data Products

1.11 Using R and Python Together

The author has helped create a SlideShare ppt on a side-by-side comparison of R and Python syntax for data science at <http://www.slideshare.net/ajayohri/python-for-r-users> (35 000+ views). However a guide for using Python and R for quantitative finance is also found at <http://www.slideshare.net/lbbarde/python-and-r-for-quantitative-finance-2409526>

Additionally the following methods help to use both R and Python and leverage their tremendous strengths:

- 1) RPy2 RPy2 helps in using R and Python together. The official documentation is given at <http://rpy.sourceforge.net/rpy2/doc-dev/html/introduction.html>. The object `r` in `rpy2.robjects` represents the running embedded R process. If familiar with R and the R console, `r` is a little like a communication channel from Python to R.

A lucid example of using RPy2 is given here at A Slug's Guide to Python (<https://sites.google.com/site/aslugsguidetopython/data-analysis/pandas/calling-r-from-python>):

```
from pandas import*
from rpy2.robjects.packages import importr
import rpy2.robjects as ro
import pandas.rpy.common as com
```

We can pass commands to the R session by putting the R syntax within the `ro.r()` method as strings, and we can read the R data.frame into pandas data frame with `com.load_data` method. We can then pass the pandas data frame back to the R instance by first converting `pydf` to an R data frame by using `com.convert_to_r_dataframe` method.

A truncated screenshot of the website is given in Figure 1.2 to help the reader understand and refer back to <https://sites.google.com/site/aslugsguidetopython/data-analysis/pandas/calling-r-from-python>

1.11.1 Using R Code for Regression and Passing to Python

An example of using `rpy2` and `caret` package in R is given for kaggle at <https://www.kaggle.com/c/bike-sharing-demand/forums/t/12923/rpy2-caret-example>

```
a caret use from python environment:
import pandas.rpy.common as com
import rpy2
```

```

from rpy2.robjects.packages import importr
graphics = importr('graphics')
grdevices = importr('grDevices')
base = importr('base')
stats = importr('stats')

import array

x = array.array('i', range(10))
y = stats.rnorm(10)

grdevices.X11()

graphics.par(mfrow = array.array('i', [2,2]))
graphics.plot(x, y, ylab = "foo/bar", col = "red")

kwargs = {'ylab':"foo/bar", 'type':"b", 'col':"blue", 'log':"x"}
graphics.plot(x, y, **kwargs)

m = base.matrix(stats.rnorm(100), ncol=5)
pca = stats.princomp(m)
graphics.plot(pca, main="Eigen values")
stats.biplot(pca, main="biplot")

```

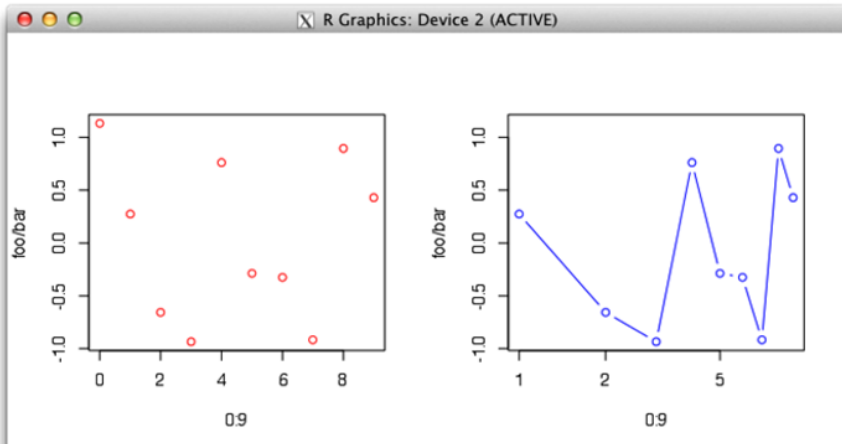


Figure 1.2 Using R code for regression and passing to Python.

```

import rpy2.robjects as ro
from rpy2.robjects import Formula
from rpy2.robjects.packages import importr
caret = importr("caret")
data_trainr = com.convert_to_r_dataframe(data_train)
param1 = {'method' : 'repeatedcv', 'number' : 3,
          'repeats' : 5}
ctrl = caret.trainControl(**param1)
param2 = {'method' : 'rf', 'trControl' : ctrl}
rf_for = Formula("log(casual + 1) ~ dm + t + wd + tp +
hum + ws")

```

```
rfmod = caretr.train(rf_for, data = data_trainr,
**param2)
print(rfmod)
```

A better but slightly old demo of using R and Python together in rpy2 is given at <http://www.bytemining.com/wp-content/uploads/2010/10/rpy2.pdf>

Another good example is given by Laurent Gautier in her talk “Polyglot applications with R and Python [BARUG Meeting]” at http://files.meetup.com/1225993/Laurent%20Gautier_R_toPython_bridge_to_R.pdf#!

A minimal example of rpy2 regression using pandas data frame is given at Stack Overflow at <http://stackoverflow.com/questions/30922213/minimal-example-of-rpy2-regression-using-pandas-data-frame>

```
from rpy2.robjects import pandas2ri
pandas2ri.activate()
robjects.globalenv['dataframe'] = dataframe
M = stats.lm('y~x', data=base.as_symbol('dataframe'))
```

The result is:

```
>>> print(base.summary(M).rx2('coefficients'))
      Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.6   1.1489125  0.522233 0.6376181
x           0.8   0.3464102  2.309401 0.1040880
```

CONDA—Conda is an open-source package management system and environment management system for installing multiple versions of software packages and their dependencies and switching easily between them. It works on Linux, OS X, and Windows and was created for Python programs but can package and distribute any software. Using conda we can use Python and R together. We can then use the familiar interface of Jupyter/IPython Notebook. You can refer to <https://www.continuum.io/conda-for-r>.

You can see the demo for R within Jupyter at <https://try.jupyter.org/>. A good blog post on using Jupyter to R is found at <https://www.continuum.io/blog/developer/jupyter-and-conda-r>.

The Anaconda team has created an “R Essentials” bundle with the IRkernel and over 80 of the most used R packages for data science, including dplyr, shiny, ggplot2, tidyr, caret, and nnet.

Once you have conda, you may install “R Essentials” into the current environment:

```
conda install -c r r-essentials
Bash
```

or create a new environment just for “R essentials”:

```
conda create -n my-r-env -c r r-essentials
```

(<https://www.continuum.io/content/preliminary-support-r-conda>)

conda create -c r -n r r will download R from our official R channel on Anaconda.org:

- Revolution Analytics—A Microsoft company that is one of the leading vendors of R has a blog post on this at <http://blog.revolutionanalytics.com/2015/09/using-r-with-jupyter-notebooks.html>
- Official documentation is also at <http://conda.pydata.org/docs/r-with-conda.html>

DOCKER—Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud.

- You can use Docker to run Jupyter. This is available at <https://hub.docker.com/r/jupyter/datascience-notebook/> and <https://github.com/jupyter/docker-stacks>

A good discussion for Docker is given at <http://stackoverflow.com/questions/16047306/how-is-docker-different-from-a-normal-virtual-machine>. I am reproducing a part of the technical answer in the following text.

Docker was using Linux Containers (LXC) earlier but switched to runC (formerly known as libcontainer) that runs in the same operating system as its host. **This allows it to share a lot of the host operating system resources.** It also uses layered file systems like AuFS. It also manages the networking for you as well.

AuFS is a layered file system, so you can have a read-only part and a write part and merge those together. So you could have the common parts of the operating system as read only, which are shared among all of your containers, and then give each container its own mount for writing.

So let's say you have a container image that is 1GB in size. If you wanted to use a full VM, you would need to have 1GB times $\times$ number of VMs you want. With LXC and AuFS you can share the bulk of the 1GB, and if you have 1000 containers, you still might only have a little over 1GB of space for the container OS, assuming they are all running the same OS image.

A full virtualized system gets its own set of resources allocated to it and does minimal sharing. You get more isolation, but it is much heavier (requires more resources).

With LXC you get less isolation, but they are more lightweight and require fewer resources. So you could easily run 1000's on a host.

You can build your own docker environment for data science.

For Jupyter and Docker, see “A data science environment in minutes using Docker and Jupyter” at <https://www.dataquest.io/blog/data-science-quickstart-with-docker/>, and for Docker and R, you can use the instructions and file at <https://hub.docker.com/r/library/r-base/>. The official R base is available at <https://store.docker.com/images/f2e50720-cada-432f-85a5-1ade438d537b?tab=description>, and you can just copy and paste to pull the image

docker pull r-base

- Python and R together using Beaker—You can use Python and R together using Jupyter, Rpy2, or Beaker. While Jupyter and rpy2 have been covered before, we can also use Beaker. You can use R Python and JavaScript within the same notebook in Beaker (and other languages too).

You can see examples here <http://beakernotebook.com/examples> and read about it at <http://blog.dominodatalab.com/interactive-data-science/> and <https://github.com/twosigma/beaker-notebook>

1.12 Other Software and Python

- SAS and Python—You can use the SAS language to talk to both Python and R. This is done using Java (passed to the Java class SASJavaExec using the Base SAS Java Object). More specifically you can see the instructions at <https://github.com/sassoftware/enlighten-integration/> and <https://communities.sas.com/docs/DOC-10746>

1.13 Using SAS with Jupyter

You can also use SAS from within Jupyter (Figure 1.3; see <http://blogs.sas.com/content/sasdummy/2016/04/24/how-to-run-sas-programs-in-jupyter-notebook/>).

1.14 How Can You Use Python and R for Big Data Analytics?

Big data is synonymous with Hadoop. For using Python with Hadoop, you can use the following packages:

- 1) Hadoop Streaming
- 2) mrjob
- 3) dumbo
- 4) hadooppy
- 5) pydoop

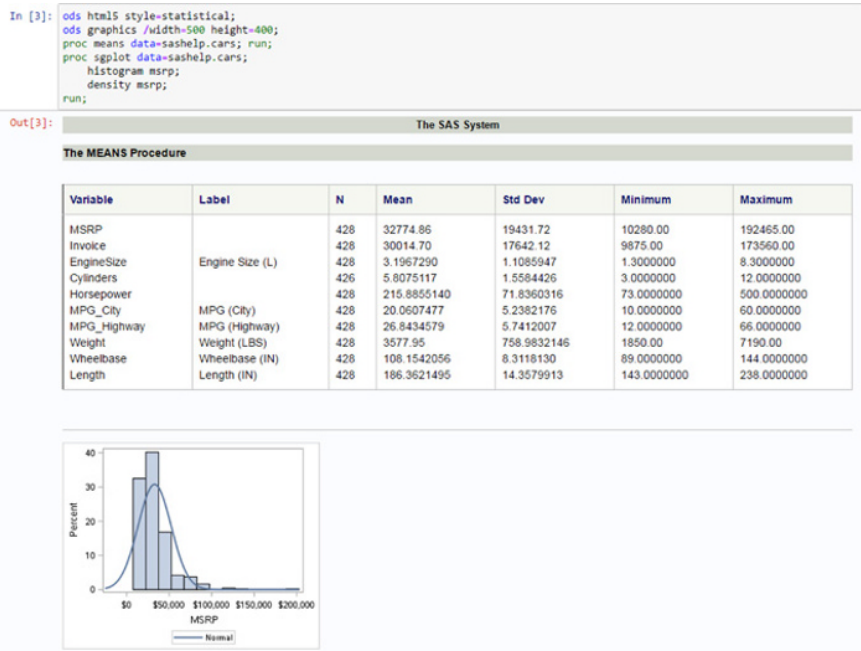


Figure 1.3 Using SAS from within Jupyter Notebook. *Source:* Chris Hemedinger on The SAS Dummy, SAS Institute. Reproduced with the permission of SAS Institute Inc.

An example is given at <https://blog.cloudera.com/blog/2013/01/a-guide-to-python-frameworks-for-hadoop/>

A recent innovation is Apache Arrow (see <https://blog.cloudera.com/blog/2016/02/introducing-apache-arrow-a-fast-interoperable-in-memory-columnar-data-structure-standard/>). As per the article, “For the Python and R communities, Arrow is extremely important, as data interoperability has been one of the biggest roadblocks to tighter integration with big data systems (which largely run on the JVM).”

The next innovation is Feather (see <https://blog.rstudio.org/2016/03/29/feather/>). Feather is a fast, lightweight, and easy-to-use binary file format for storing data frames, and Feather files are the same whether written by Python or R code. The Python interface uses Cython to expose Feather’s C++11 core to users, while the R interface uses Rcpp for the same task.

1.15 What Is Cloud Computing?

The official definition of cloud computing is given at <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>

Separation of responsibilities

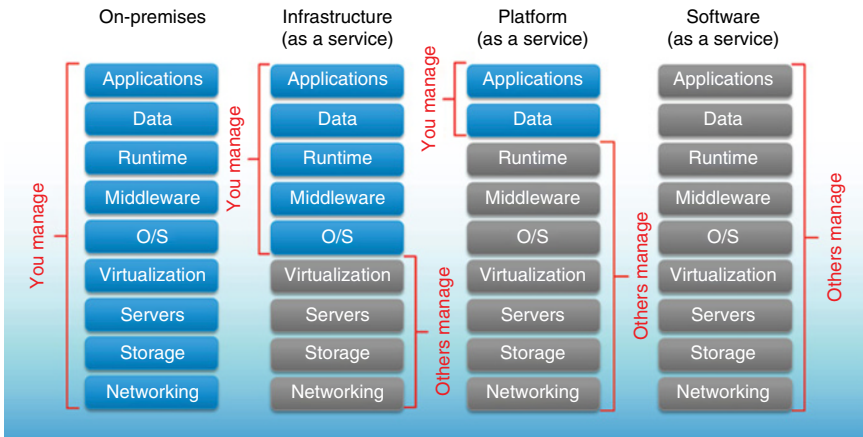


Figure 1.4 The difference between infrastructure as service, platform as a service, and software as a service. *Source:* <https://blogs.technet.microsoft.com/kevinremde/2011/04/03/saas-paas-and-iaas-oh-my-cloudy-april-part-3/>. © Microsoft.

Cloud computing is a model for enabling:

- 1) Ubiquitous, convenient on-demand network access
- 2) A shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction

Amazon (EC2), Google, Oracle, IBM, and Microsoft Azure are some examples of cloud providers. For a data scientist, it is important to know the difference between Infrastructure as a Service, Platform as a Service, and Software as a Service (Figure 1.4).

1.16 How Can You Use Python and R on the Cloud?

If you want to host and run Python in the cloud, these implementations may be right for you: PythonAnywhere (freemium hosted Python installation that lets you run Python in the browser, e.g., for tutorials, showcases, etc.). It has an additional use case for education.

From <https://www.pythonanywhere.com/details/education>, Python is a great language for teaching, but getting it installed and set up on all your students' computers can be less than easy. PythonAnywhere provides an

environment that's ready to go—including a syntax-highlighting, error-checking editor and Python 2 and 3 consoles.

You can use web scraping from Python on the cloud through <http://scrapinghub.com/scrapy-cloud/>. Scrapy is the most popular and advanced web crawling framework for Python. It makes writing web crawlers fast, easy, and fun. However, you still need to deploy and run your crawler periodically, manage servers, monitor performance, review scraped data, and get notified when spiders break. This is where Scrapy Cloud comes in.

Additionally, you can run RStudio Server on the cloud if you prefer the RStudio interface using the instructions at http://www.louisaslett.com/RStudio_AMI/. As of May 2016 there is experimental support for Julia (and Python).

1.17 Commercial Enterprise and Alternative Versions of Python and R

Two principal commercial distributions of Python for data scientists are as follows:

- Anaconda from Continuum Analytics (<https://www.continuum.io/downloads>)
Anaconda is a completely free Python distribution (including for commercial use and redistribution). It includes more than 300 of the most popular Python packages for science, math, engineering, and data analysis.

- Enthought Canopy (<https://www.enthought.com/products/canopy/>)
Enthought Canopy is a Python analysis environment that provides easy installation of the core scientific analytic and scientific Python packages.

A number of alternative implementations are also available (see <https://www.python.org/download/alternatives/>):

- IronPython (Python running on .NET).
- Jython (Python running on the Java virtual machine).
- PyPy (<http://pypy.org/>). PyPy is a fast, compliant alternative implementation of the Python language (2.7.10 and 3.2.5). It has several advantages in terms of speed and distinct features but is currently trying to port NumPy package (NumPy is the basic package for many numerical operations in Python).
- Stackless Python (branch of CPython supporting microthreads).

Some repackagings of Python are the following:

- ActiveState ActivePython (commercial and community versions, including scientific computing modules)

- pythonxy (scientific-oriented Python Distribution based on Qt and Spyder)
- winpython (WinPython is a portable scientific Python distribution for Windows)
- Conceptive Python SDK (targets business, desktop, and database applications)
- PyIMSL Studio (a commercial distribution for numerical analysis—free for noncommercial use)
- eGenix PyRun (a portable Python runtime, complete with stdlib, frozen into a single 3.5–13 MB executable file)

In addition there is Cython (<http://cython.org/>), an optimizing static compiler for both the Python programming language and the extended Cython programming language (based on Pyrex). It makes writing C extensions for Python easier:

- For R the commercial versions are by Revolution Analytics, a Microsoft-acquired subsidiary (www.revolutionanalytics.com). Revolution Analytics makes RevoScaleR package that helps scale up to bigger datasets. RStudio also makes software around R (including Shiny Package and a widely used IDE at www.rstudio.com).
- Renjin is a JVM-based interpreter for the R language for statistical computing (<http://www.renjin.org/>).
- pqR, a pretty quick version of R (<http://www.pqr-project.org/>), is a new version of the R interpreter. It is based on R-2.15.0 later versions distributed by the R Core Team (at r-project.org).
- Oracle R Enterprise (ORE) (see <https://blogs.oracle.com/R/>) (<http://www.oracle.com/technetwork/database/database-technologies/r/r-enterprise/overview/index.html>). Oracle R Enterprise, a component of the Oracle Advanced Analytics Option, makes the open-source R statistical programming language and environment ready for the enterprise and big data. Designed for problems involving large volumes of data, it integrates R with Oracle Database. R users can run R commands and scripts for statistical and graphical analyses on data stored in Oracle Database.
- **TIBCO Enterprise Runtime for R (TERR)** (<http://spotfire.tibco.com/discover-spotfire/what-does-spotfire-do/predictive-analytics/tibco-enterprise-runtime-for-r-terr>). TERR, a key component of Spotfire Predictive Analytics, is an enterprise-grade analytic engine that TIBCO has built from the ground up to be fully compatible with the R language, leveraging our long-time expertise in the closely related S+ analytic engine. This allows customers not only to continue to develop in open source R but also to then integrate and deploy their R code on a commercially supported and robust platform.

1.17.1 Commonly Used Linux Commands for Data Scientists

It is important for the budding data scientist to learn the right operating system before a language; hence here are some Linux tips:

- **ls**—Directory listing
- **cd dir**—Change directory to dir
- **mkdir dirname**—Makes a directory named dirname
- **cd**—Change to home
- **sudo**—Gives superuser or admin rights
- **sudo bash**—Changes to root
- **pwd**—Shows present working directory
- **rm filename**—Removes file named filename
- **cat > filename**—Puts standard output in a file
- **cp filename1 filename2**—Copies filename1 to filename2
- **mv filename1 filename2**—Moves filename1 to filename2

Refer to <http://www.linuxstall.com/linux-command-line-tips-that-every-linux-user-should-know/> and <http://i0.wp.com/www.linuxstall.com/wp-content/uploads/2012/01/linux-command-line-cheat-sheet.png> (Figure 1.5).

1.17.2 Learning Git

Git is a version control system that enables teams to work together on projects as well as share code. GitHub is a popular website for sharing packages and libraries under development in R.

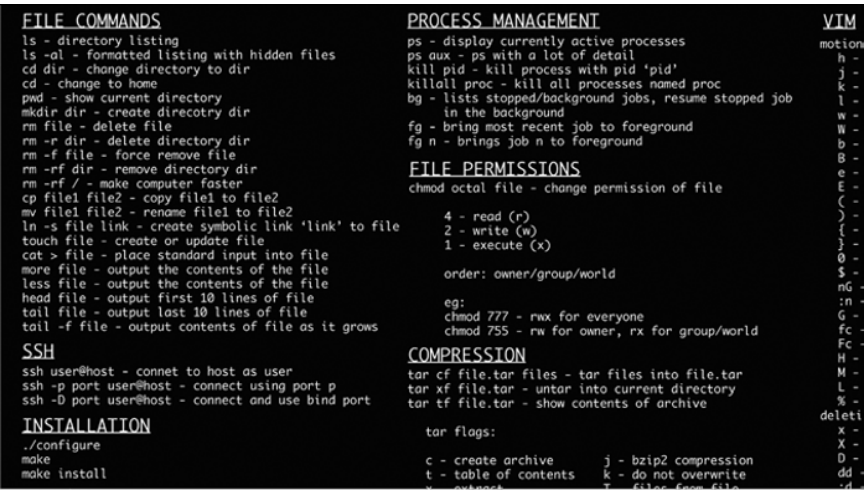


Figure 1.5 Linux cheat sheet.

The following cheat sheet will help you get started in Git (<http://overapi.com/static/cs/git-cheat-sheet.pdf>) (Figures 1.6 and 1.7). You can test your knowledge by a tutorial at <https://try.github.io/levels/1/challenges/1>. Lastly the author believes the best way to learn Git is to start contributing to a project.

Git Cheat Sheet

<http://git.or.cz/>

Remember: `git command --help`

Global Git configuration is stored in `$HOME/.gitconfig` (`git config --help`)

Create	Concepts
From existing data <pre>cd ~/projects/myproject git init git add .</pre> From existing repo <pre>git clone ~/existing/repo ~/new/repo git clone git://host.org/project.git git clone ssh://you@host.org/proj.git</pre> Show Files changed in working directory <pre>git status</pre> Changes to tracked files <pre>git diff</pre> What changed between \$ID1 and \$ID2 <pre>git diff \$id1 \$id2</pre> History of changes <pre>git log</pre> History of changes for file with diffs <pre>git log -p \$file \$dir/ec/ory/</pre> Who changed what and when in a file <pre>git blame \$file</pre>	Git Basics <pre>master : default development branch origin : default upstream repository HEAD : current branch HEAD^ : parent of HEAD HEAD~4 : the great-great grandparent of HEAD</pre> Revert Return to the last committed state <pre>git reset --hard</pre>  you cannot undo a hard reset Revert the last commit <pre>git revert HEAD</pre> Creates a new commit Revert specific commit <pre>git revert \$id</pre> Creates a new commit Fix the last commit <pre>git commit -a --amend</pre> (after editing the broken files) Checkout the \$id version of a file <pre>git checkout \$id \$file</pre> Branch Switch to the \$id branch

Figure 1.6 Git cheat sheet. Source: © Github.

Commands Sequence

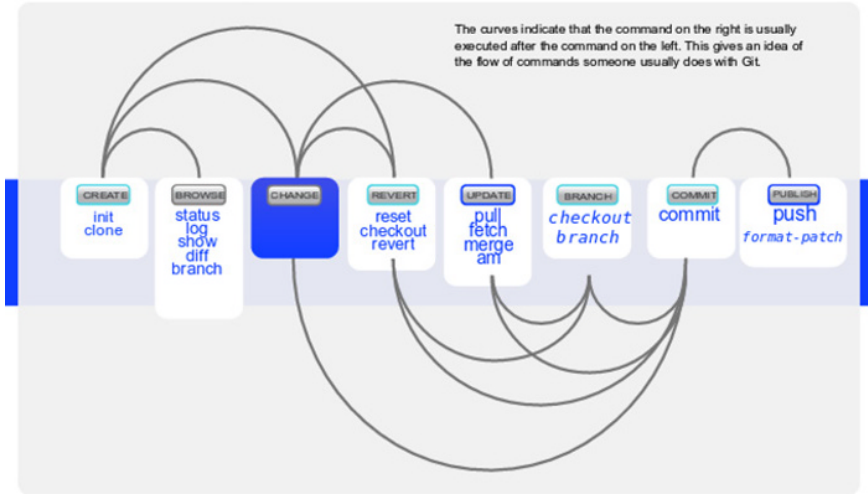


Figure 1.7 Git command sequence. *Source:* © Github.

Let's begin learning the basics of Python (<https://nbviewer.jupyter.org/gist/decisionstats/ce2c16ee98abcf328177>).

Bold font is code; normal font is output.

Numerical Operations

2+3+5

10

66-3-(-4)

67

32*3

96

23 #2 raised to power of 3**

8

43/3

14.333333333333334

32//3 #Gives quotient

10

44%3 #Gives remainder

2

For R it is almost the same except for the last two. There the syntax is:

```
32 % / % 3 #Gives quotient
44 % % 3 #Gives remainder
```

For Loops

See <https://docs.python.org/3/tutorial/controlflow.html>

```
#numbers from 0 to 30 increment 6
for x in range(0, 30, 6):
    print (x)

0
6
12
18
24
```

For Loops can be slightly different in R. Note the + sign denotes a new line in R code:

```
for(i in seq(0,30,6)){
+   print(i)
+ }
[1] 0
[1] 6
[1] 12
[1] 18
[1] 24
[1] 30
```

Functions

```
def myfirstfunction(x):
    y=x**3+3*x+20
    print(y)

myfirstfunction(20)

8080
```

In R creating a function would be different. You would need to use a function like `function(x)` as follows, then write the function within brackets, and print out the value. This is because R like most computer languages does not use

space indentation. In **R**, you can view a **function's** code by typing the **function** name without the **()**. This is especially useful to see the algorithms in an existing package and to tweak it if possible:

```
myfirstfunction =function(x) {
  y=x**3+3*x+20
  print(y) }
```

```
myfirstfunction(20)
```

```
for x in range(0,30,6):
    myfirstfunction (x)
```

```
20
254
1784
5906
13916
```

```
def mynewfunction(x,y):
    z=x**3+3*x*y+20*y
    print(z)
```

```
mynewfunction(1,3)
```

```
70
```

```
mynewfunction(10,3)
```

```
1150
```

See <http://rpubs.com/ajaydecis/forfunctions>, <https://docs.python.org/2/library/functions.html>, and <http://stackoverflow.com/questions/7969949/whats-the-difference-between-globals-locales-and-vars>

- `globals()`—*Always* returns the dictionary of the *module* namespace
- `locals()`—*Always* returns a dictionary of the *current* namespace
- `vars()`—Returns *either* a dictionary of the current namespace (if called with no argument) or *the* dictionary of the argument

locals() #gives objects in local space

```
{ 'In': [' ',
        '2+3+5 ',
        '66-3-(-4) ',
```

```

'2^3',
'3^3',
'44%%3',
'44%3',
'43/3',
'32*3',
'2**3',
..... truncated by author
globals()

{'In': ['',
  '2+3+5',
  '66-3-(-4)',
  '2^3',
  '3^3',
  '44%%3',
  '44%3',
  '43/3',
  '32*3',
  '2**3',
  '32//3',
  'for i in 1:30\n    print i',
  'for i in 1:30:\n    print i',
  'for i in 1:30:\n    print i',
  'for i in range(1,30):\n    print i',
  'for i in range(1,30):\n    print i',
  'for i in range(1,30):\n    print %i',
  'for i in range(1,30):\n    print %(i)',
  'for i in range(1,30):\n    print % (i)',
  'for i in range(1,30):\n    print % (i)',
  'for x in range(0,30):\n    print % (x)',
  -----truncated by author

```

More Numerical Operations

```

import math

math.exp(2)

7.38905609893065
math.log(2)

0.6931471805599453
math.log(2,10)

```

```
0.30102999566398114
```

```
math.sqrt(10)
```

```
3.1622776601683795
```

```
dir(math) #dir gives all the identifiers a module defines
```

```
['__doc__',  
 '__file__',  
 '__loader__',  
 '__name__',  
 '__package__',  
 '__spec__',  
 'acos',  
 'acosh',  
 'asin',  
 'asinh',  
 'atan',  
 'atan2',  
 'atanh',  
 'ceil',  
 'copysign',  
 'cos',  
 'cosh',  
 'degrees',  
 'e',  
 'erf',  
 'erfc',  
 'exp',  
 'expm1',  
 'fabs',  
 'factorial',  
 'floor',  
 'fmod',  
 'frexp',  
 'fsum',  
 'gamma',  
 'hypot',  
 'isfinite',  
 'isinf',  
 'isnan',  
 'ldexp',  
 'lgamma',  
 'log',
```

```

'log10',
'log1p',
'log2',
'modf',
'pi',
'pow',
'radians',
'sin',
'sinh',
'sqrt',
'tan',
'tanh',
'trunc']
a=[23,45,78,97,89]

type(a)

list
len(a)

5
max(a)

97
min(a)

23
sum(a)

332
import numpy

numpy.mean(a)

66.4000000000000006

numpy.std(a)

28.011426240018555
numpy.var(a)

784.639999999999999
#Example of Help (note the ? is almost the same as R)
numpy.random?
```

```
from random import randint,randrange
print(randint(0,9))
```

```
5
```

```
randrange(10)
```

```
4
```

```
for x in range(0,5):
    print(randrange(10))
```

```
8
```

```
3
```

```
7
```

```
8
```

```
5
```

Strings, Lists, Tuples, and Dicts

```
newstring='Hello World'
```

```
newstring
```

```
'Hello World'
```

```
print(newstring)
```

```
Hello World
```

```
newstring2='Hello World's'
```

```
File "<ipython-input-56-8c5b85561ed9>", line 1
    newstring2='Hello World's'
                ^
```

```
SyntaxError: invalid syntax
```

#Double Quotes and Single Quotes

```
newstring2="Hello World's"
```

```
print(newstring2)
```

```
Hello World's
```

#Escape character \

```
newstring3="Hello, World\'s"
```

```
print(newstring3)
```

Hello, World's

10*newstring3

"Hello, World'sHello, World'sHello, World'sHello,
World'sHello, World'sHello, World'sHello,
World'sHello, World'sHello, World'sHello World's "

Passing Variables in Strings in Python

```
myname1= 'Ajay'
myname2= 'John'
message =" Hi I am %s. How do you do"
message %myname1
' Hi I am Ajay. How do you do'
```

```
message %myname2
' Hi I am John. How do you do'
```

```
new1= "Why did the %s cross the %s"
print(new1%('chicken','road'))
```

Why did the chicken cross the road

```
print(new1%(10,40))
Why did the 10 cross the 40
```

```
new2= "Why did the %d cross the %d"
```

```
print(new2%('chicken','road'))
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-11-b2f398d16f9c> in <module>()
----> 1 print(new2%('chicken','road'))
```

TypeError: %d format: a number is required, not str

Note the error caused by %d and %s

Lists

```
newnames='ajay,vijay,john,donald,hillary,bill,ashok'
```

```
type(newnames)
```

```
str
```

```
newnames[0:9]
```

```
'ajay,vija'
newnames2=['ajay','vijay','john','donald','hillary',
'bill','ashok']
```

```
type(newnames2)
```

```
list
```

In R, a list would be created like this:

```
newnames2=c('ajay','vijay','john','donald','hillary',
'bill','ashok')
```

```
newnames2[0]
```

```
'ajay'
```

So in R the index starts from 1, while in Python the index starts with 0.

```
newnames2[0]='micky mouse' #substituting members in a list
```

```
newnames2
```

```
['micky mouse', 'vijay', 'john', 'donald', 'hillary',
'bill', 'ashok']
```

```
newnames2[2]
```

```
'john'
```

```
newnames2.append('daisy')
```

```
newnames2
```

```
['micky mouse', 'vijay', 'john', 'donald', 'hillary',
'bill', 'ashok', 'daisy']
```

.append to add and del to delete members in a list

```
del newnames2[2]
```

```
newnames2
```

```
['micky mouse', 'vijay', 'donald', 'hillary', 'bill',
'ashok', 'daisy']
```

```
newlist=[1,2,4,7]
```

```
newnames2+newlist
```

```
['micky mouse',
 'vijay',
 'donald',
 'hillary',
 'bill',
 'ashok',
 'daisy',
 1,
 2,
 4,
 7]
```

```
newlist*3
```

```
[1, 2, 4, 7, 1, 2, 4, 7, 1, 2, 4, 7]
```

a tuple is a list that uses parentheses () not square brackets [] and it CANNOT be modified at all once created

```
scores=(23,46,69,7,5)
```

```
type(scores)
```

```
tuple
scores[3]
```

```
7
```

dir(scores) #dir command gives various operations that can be done to that object

```
['_add_',
 '_class_',
 '_contains_',
 '_delattr_',
 '_dir_',
 '_doc_',
 '_eq_',
 '_format_',
 '_ge_',
 '_getattribute_',
 '_getitem_',
 '_getnewargs_',
 '_gt_',
```

```

'__hash__',
'__init__',
'__iter__',
'__le__',
'__len__',
'__lt__',
'__mul__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rmul__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'count',
'Index']

```

```

favourite_movie=['micky mouse,steamboat willie',
'vijay,slumdog millionaire', 'john,passion of christ',
'donald,arthur']

```

```

type(favourite_movie)

```

```

list

```

```

favourite_movie2={'micky mouse':'steamboat
willie','vijay':'slumdog millionaire','john':'passion
of christ','donald':'arthur'}

```

```

type(favourite_movie2)

```

```

dict

```

```

favourite_movie2['micky mouse']

```

```

'steamboat willie'

```

```

favourite_movie2['vijay']

```

```

'slumdog millionaire'

```

Refer to <https://nbviewer.jupyter.org/gist/decisionstats/752ff727101cf6fc13225bd94eef358a> for the code in this example.

Strings—We use `str` function to convert data to string data (we use `int` to convert data to integer values). We can use *slicing* on index to create substrings and concatenate strings using `+` sign. The following example shows some of the things that can be done with string data:

```
names=['Ajay','Vijay','Ra Jay','Jayesh']
```

```
type(names)
```

```
list
```

```
names[1]
```

```
'Vijay'
```

```
type(names[1])
```

```
str
```

```
names[0][1:3]
```

```
'ja'
```

```
names[2][2:]
```

```
' Jay'
```

```
names[2][2:] + names[3][2:]
```

```
' Jayyesh'
```

```
names[1].lower()
```

```
'vijay'
```

```
names[2].replace(" ", "")
```

```
'RaJay'
```

Let's try to do the same thing in R (<http://rpubs.com/ajaydecis/strings4>). There are important differences we want to highlight:

```
names=c('Ajay','Vijay','Ra Jay','Jayesh')
```

R uses `c` to make a list. Python does not—but uses square brackets. Python uses `type` while R uses `class` to find out the object's type:

```
class(names)
```

```
## [1] "character"
```

Python starts the index from 0 while begins the index of a list from 1:

```
names[1]
## [1] "Ajay"
class(names[1])
## [1] "character"
```

You have to use `substr` in R to find part of a string. In Python you simply can look this from within square brackets:

```
substr(names[1], 2, 3)
## [1] "ja"
substr(names[3], 3, nchar(names[3]))
## [1] " Jay"
```

While Python simple combined strings using `+`, R used `paste`:

```
paste(substr(names[3], 3, nchar(names[3])), substr(names
[2], 3, nchar(names[2])))
## [1] " Jay jay"
```

R uses **tolower** while Python uses `lower()`:

```
tolower(names[1])
## [1] "ajay"
```

Python used `replace` while R used `gsub`:

```
gsub(" ", "", names[3])
## [1] "RaJay"
```

The biggest difference is R mostly uses `function(object)`, while Python uses `object.function()` to get things done. This is an important difference

File and Folder Operations

In Python we use the `os` package for file operations to refer and read the file from a particular directory. We also use `!pip freeze` to get the list of packages (versions). We use `print(IPython.sys_info())` and `version_information` package (`%load_ext version_information`

`%version_information`) to get System Information (see Python code at <https://nbviewer.jupyter.org/gist/decisionstats/29f3adfb6980db52a61130aa8c8f9166>).

In R we get System Information using `sessionInfo()`. (For R Code see <http://rpubs.com/newajay/systeminfo>)

```
import IPython
print (IPython.sys_info())
{'commit_hash': 'c963f6b',
 'commit_source': 'installation',
 'default_encoding': 'UTF-8',
 'ipython_path': '/home/ajayohri/anaconda3/lib/
python3.5/site-packages/IPython',
 'ipython_version': '4.2.0',
 'os_name': 'posix',
 'platform': 'Linux-4.4.0-53-generic-x86_64-with-
debian-stretch-sid',
 'sys_executable': '/home/ajayohri/anaconda3/bin/
python',
 'sys_platform': 'linux',
 'sys_version': '3.5.2 |Anaconda 4.1.1 (64-bit)|
(default, Jul  2 2016, '
    '17:53:06) \n'
    '[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]'}
```

!pip install version_information

The directory “/home/ajayohri/.cache/pip/http” or its parent directory is not owned by the current user, and the cache has been disabled. Please check the permissions and owner of that directory. If executing pip with sudo, you may want sudo’s -H flag.

The directory “/home/ajayohri/.cache/pip” or its parent directory is not owned by the current user, and caching wheels has been disabled. Check the permissions and owner of that directory. If executing pip with sudo, you may want sudo’s -H flag.

Collecting version_information

Downloading version_information-1.0.3.tar.gz

Installing collected packages: version-information

Running setup.py install for version-information ... - \ | done

Successfully installed version-information-1.0.3

You are using pip version 8.1.2; however version 9.0.1 is available.

You should consider upgrading via the “pip install --upgrade pip” command.

```
%load_ext version_information
```

```
%version_information
```

Software	Version
Python	3.5.2 64bit [GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
IPython	4.2.0
OS	Linux 4.4.0 53 generic x86_64 with Debian stretch sid
Sat Dec 24 19:47:41 2016 IST	

!pip freeze

```
alabaster==0.7.8
anaconda-client==1.4.0
anaconda-navigator==1.2.1
argcomplete==1.0.0
astropy==1.2.1
Babel==2.3.3
backports.shutil-get-terminal-size==1.0.0
beautifulsoup4==4.4.1
-----list truncated by author
SQLAlchemy==1.0.13
statsmodels==0.6.1
sympy==1.0
tables==3.2.2
terminado==0.6
toolz==0.8.0
tornado==4.3
traitlets==4.2.1
unicodcsv==0.14.1
version-information==1.0.3
Werkzeug==0.11.10
xlrd==1.0.0
XlsxWriter==0.9.2
xlwt==1.1.2
```

The directory “/home/ajayohri/.cache/pip/http” or its parent directory is not owned by the current user, and the cache has been disabled. Please check the permissions and owner of that directory. If executing pip with sudo, you may want sudo’s -H flag.

You are using pip version 8.1.2; however version 9.0.1 is available.

You should consider upgrading via the “pip install --upgrade pip” command.
(Authors note-warning message by system)

```
import os as os
os.getcwd()
'/home/ajayohri/Desktop'
os.chdir('/home/ajayohri/Desktop')
```

```

os.getcwd()
'/home/ajayohri/Desktop'
os.listdir()

['Data Analytics Course: Master Data Analytics Using
Python in 2.5 Months_files',
 'dump 4 nov 2016',
 'Hadoop Tutorial | All you need to know about Hadoop
| Edureka_files',
 'Data Analytics Course: Master Data Analytics Using
Python in 2.5 Months.html',
 'Hadoop Tutorial | All you need to know about Hadoop
| Edureka.html',
 'Note to R Users – Data Analysis in Python 0.1
documentation_files',
 'Note to R Users – Data Analysis in Python 0.1
documentation.html',
 'Jupyter Notebook Viewer.html',
 'py4r.jpg',
 'test',
 'hackerearth',
 'Jupyter Notebook Viewer_files',
 'logo-ds.png']

```

In R this would be slightly different:

```

sessionInfo()
## R version 3.3.1 (2016-06-21)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.1 LTS
##
## locale:
##  [1] LC_CTYPE=en_IN.UTF-8    LC_NUMERIC=C
##  [3] LC_TIME=en_IN.UTF-8     LC_COLLATE=en_IN.UTF-8
##  [5] LC_MONETARY=en_IN.UTF-8 LC_MESSAGES=en_IN.UTF-8
##  [7] LC_PAPER=en_IN.UTF-8    LC_NAME=C
##  [9] LC_ADDRESS=C            LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_IN.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats graphics grDevices utils datasets
##      methods base
##

```

```
## loaded via a namespace (and not attached):
## [1] magrittr_1.5 tools_3.3.1 htmltools_0.3.5
##      Rcpp_0.12.8
## [5] stringi_1.1.1 rmarkdown_1.0 knitr_1.13
##      stringr_1.0.0
## [9] digest_0.6.9 evaluate_0.9

getwd()
## [1] "/home/ajayohri"

setwd("~/home/ajayohri/Desktop/")

dir()
## [1] "Data Analytics Course: Master Data Analytics
##      Using Python in 2.5 Months_files"
## [2] "Data Analytics Course: Master Data Analytics
##      Using Python in 2.5 Months.html"
## [3] "dump 4 nov 2016"
## [4] "hackerearth"
## [5] "Hadoop Tutorial | All you need to know about
##      Hadoop | Edureka_files"
## [6] "Hadoop Tutorial | All you need to know about
##      Hadoop | Edureka.html"
## [7] "Jupyter Notebook Viewer_files"
## [8] "Jupyter Notebook Viewer.html"
## [9] "logo-ds.png"
## [10] "Note to R Users – Data Analysis in Python 0.1
##      documentation_files"
## [11] "Note to R Users – Data Analysis in Python 0.1
##      documentation.html"
## [12] "py4r.jpg"
## [13] "test"
```

The following deals with the business part (or domain expertise part) of the decision science triad (programming, statistics, and domain expertise).

1.18 Data-Driven Decision Making: A Note

A fundamental principle of data-driven decision making is a famous quote: If you can't measure it, you can't manage it—Peter Drucker.

As per <http://whatis.techtarget.com/definition/data-driven-decision-management-DDDM>, data-driven decision management (DDDM) is an approach to business governance that values decisions that can be backed

up with verifiable data. The success of the data-driven approach is reliant upon the quality of the data gathered and the effectiveness of its analysis and interpretation.

As per author, the following constitutes data-driven decision making:

- Using past data and trending historical data
- Validating assumptions if any after listing all assumptions
- Using champion challenger scenarios to test scenarios
- Using experiments for various tests
- Use baselines for continuous improvement in customer experiences, costs, and revenues
- Taking decisions based on the previous process

As per HBR.org, the more frequent the correlation in a company's data and the lower the risk of being wrong, the more it makes sense to act based on that correlation (Citation: <https://hbr.org/2014/05/an-introduction-to-data-driven-decisions-for-managers-who-dont-like-math>).

1.18.1 Strategy Frameworks in Business Management: A Refresher for Non-MBAs and MBAs Who Have to Make Data-Driven Decisions

Some frameworks are used for business strategy—to come up with decisions after analyzing the huge reams of qualitative and uncertain data that business generates. This is also part of the substantive expertise circle in Conway's Venn diagram definition of data science at <http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram> (Figure 1.8).

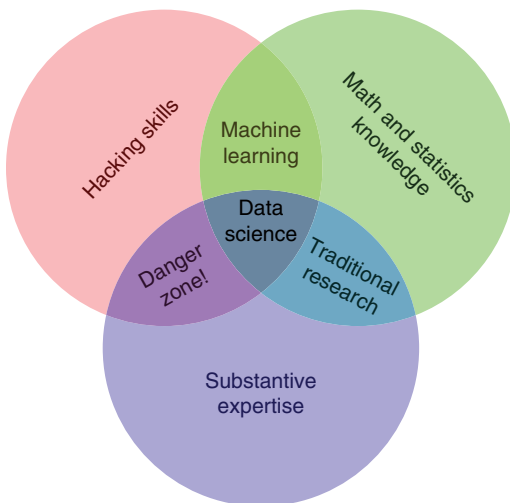


Figure 1.8 Conway's Venn diagram. Source: © Drew Conway Data Consulting, LLC.

- **Porter's five forces model**—To analyze industries. Porter's famous model is used to derive five forces that determine the competitive intensity and therefore attractiveness of a market. Attractiveness in this context refers to the overall industry profitability. An “unattractive” industry is one in which the combination of these five forces acts to drive down overall profitability. A very unattractive industry would be one approaching “pure competition” (Figure 1.9).
- **Business canvas**—The business model canvas is used for developing new or documenting existing business models. It describes a firm's value proposition, infrastructure, customers, and finances and thus assists firms by illustrating potential trade-offs in various activities. The business model canvas was initially proposed by Alexander Osterwalder. A bigger graphic can be obtained at https://en.wikipedia.org/wiki/File:Business_Model_Canvas.png (Figure 1.10).
- **BCG matrix**—To analyze product portfolios. BCG Matrix is best used to analyze your own or target organization's product portfolio—applicable for companies with multiple products. This helps corporations allocate resources by analyzing their business units or product lines (Figure 1.11).
- **Porter's diamond model**—To analyze locations. An economical model developed by Michael Porter in his book *The Competitive Advantage of Nations*, where he published his theory of why particular industries become competitive in particular locations. This helps to analyze countries, states, or locations for both customers and vendors (Figure 1.12).
- **McKinsey 7S model**—To analyze teams. To check which teams work and which teams are done (within an organization), we can use the 7S model. It is a strategic vision for groups to include businesses, business units, and teams. The 7S are structure, strategy, systems, skills, style, staff, and shared values. The model is most often used as a tool to assess and monitor changes in the internal situation of an organization (Figure 1.13).

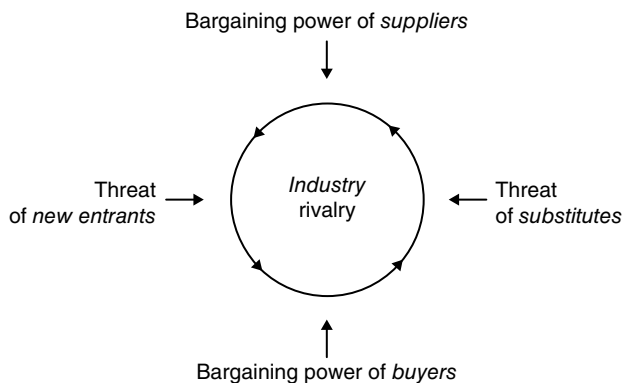


Figure 1.9 Porter five forces for competitive strategy. Source: © Wikipedia.

Version:

CC BY SA | **DESIGNED BY: Business Model Foundry AG**
The makers of Business Model Generation and Strategizer

This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, visit: <http://creativecommons.org/licenses/by-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA

Figure 1.10 Business model canvas. *Source:* © Strategyzer AG Services.

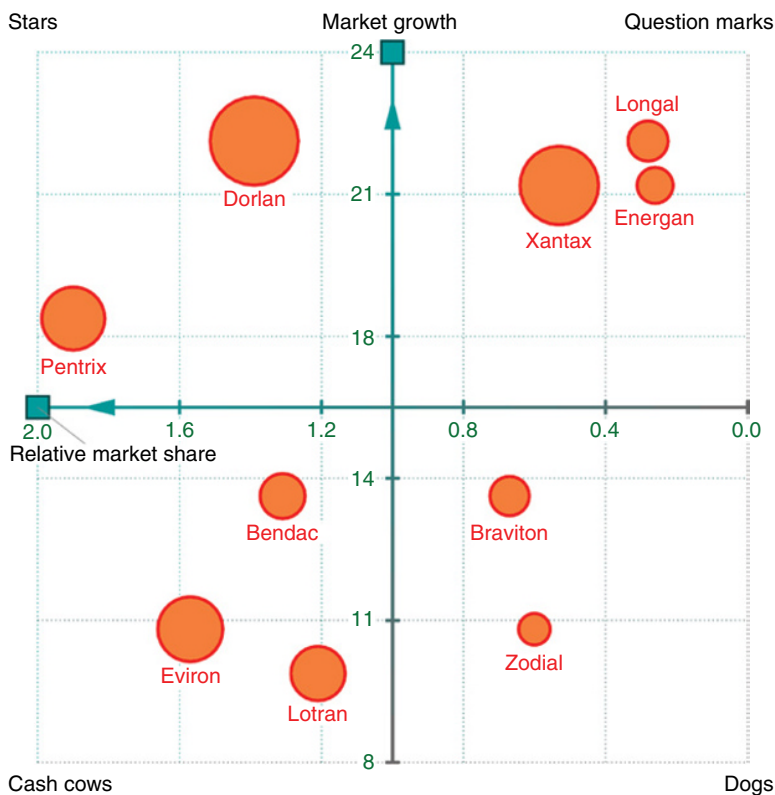


Figure 1.11 BCG matrix. Source: http://en.wikipedia.org/wiki/Growth-share_matrix. © Wikipedia.

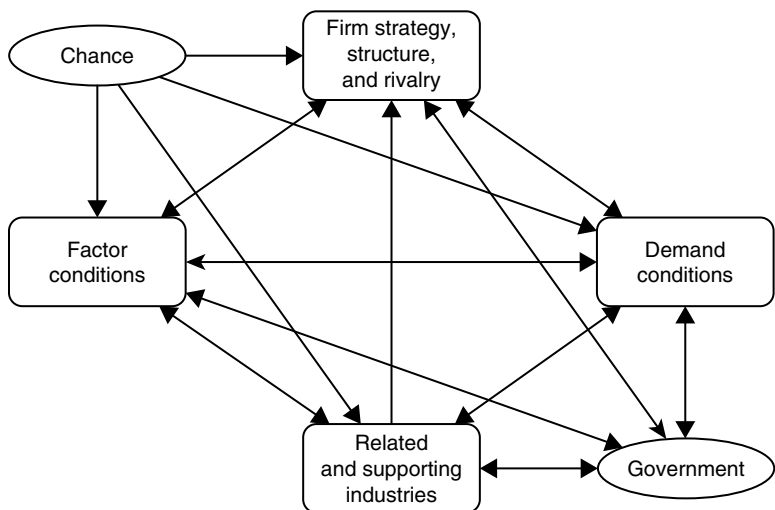


Figure 1.12 Porter's diamond model for competitiveness in locations. Source: http://en.wikipedia.org/wiki/Diamond_model. © Wikipedia.

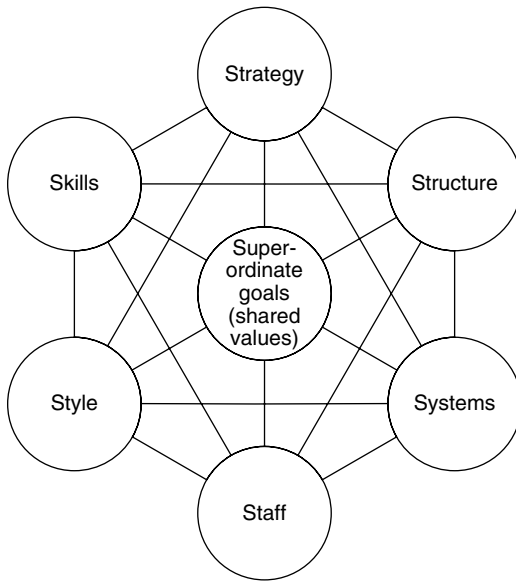


Figure 1.13 McKinsey 7S model. Source: http://en.wikipedia.org/wiki/McKinsey_7S_Framework. © Wikipedia.

- Grenier's theory—To analyze growth of organization. It was developed by Larry E. Greiner and is helpful when examining the problems associated with growth on organizations and the impact of change on employees. It can be argued that growing organizations move through five relatively calm periods of evolution, each of which ends with a period of crisis and revolution. Each evolutionary period is characterized by the dominant management style used to achieve growth, while each revolutionary period is characterized by the dominant management problem that must be solved before growth will continue (Figure 1.14).
- Herzberg's hygiene theory—To analyze soft aspects of individuals.

The following table presents the top seven factors causing dissatisfaction and the top six factors causing satisfaction, listed in the order of higher to lower importance.

Leading to satisfaction

- Achievement
- Recognition
- Work itself
- Responsibility
- Advancement
- Growth

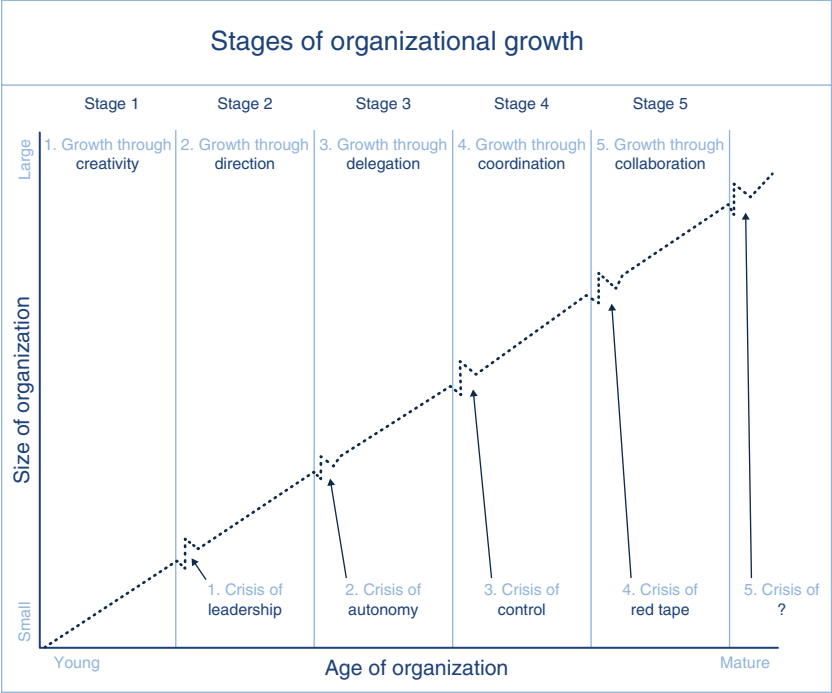


Figure 1.14 Grenier theory. *Source:* Adapted from Greiner (1998). *Evolution and Revolution as Organizations Grow.* © Harvard Business Publishing.

Leading to dissatisfaction

- Company policy
- Supervision
- Relationship with boss
- Work conditions
- Salary
- Relationship with peers
- Security

This framework helps to explain what motivates people to contribute (or fail to contribute) to teams, products, organizations, and nations. Alternative motivational models are Maslow’s hierarchy of needs (shown here) and McGregor Theory X and Theory Y. McGregor terms the two models as “Theory X,” which stresses the importance of strict supervision and external rewards and penalties, and “Theory Y,” which highlights the motivating role of job satisfaction and allows scope for workers to approach tasks creatively.

- Marketing mix modeling—To analyze marketing mix for determining a product or a brand's offer. It has the four P's: price, product, promotion, and place. This can also be shown by four C's model: consumer, cost, communication, and convenience.

1.18.2 Additional Frameworks for Business Analysis

Pareto Principle

The Pareto principle (also known as the 80/20 rule, the law of the vital few, and the principle of factor sparsity) is a heuristic or a thumb rule that tells analysts to prioritize their analysis. It helps states that, for many events, roughly 80% of the effects come from 20% of the following causes:

- 80% of a company's profits come from 20% of its customers.
- 80% of a company's complaints come from 20% of its customers.
- 80% of a company's profits come from 20% of the time its staff spend.
- 80% of a company's sales come from 20% of its products.
- 80% of a company's sales are made by 20% of its sales staff.

Thus a business analyst should look at the top and bottom 20% of the products, orders, customers, and staff when doing an analysis to determine the cause and effect relationships that can be then modified for positive value creation.

An additional framework is root cause analysis where one can ask five successive why's to determine the root cause of an effect. The process is to ask "why" and identify the causes associated with each sequential step toward the event. "Why" here stands for "What were the factors that directly resulted in the effect?"

LTV Analysis

Lifetime value (LTV) analysis is often a widely used technique within BA to help businesses which customers to retain and which to churn. It also helps with promotions and customer acquisition strategy. LTV is the cumulative revenue a customer will generate for a business over his active lifetime when associated with the products and brands of that business—from acquisition to churn. LTV helps us answer three fundamental questions:

- 1) Did the business pay enough to acquire customers from each marketing channel (cost of acquisition)?
- 2) Did the business acquire the best kind of customers (profitability analysis)?
- 3) How much could the business spend on keeping or retaining them as your customers (prevent churn by offers, calls, email, and social media)?

You can calculate LTV analysis using the methods given at <https://blog.kissmetrics.com/how-to-calculate-lifetime-value/> and at <http://www.kaushik.net/avinash/analytics-tip-calculate-ltv-customer-lifetime-value/>

For LTV analysis in R, you can see this R package at <https://cran.r-project.org/web/packages/BTYD/vignettes/BTYD-walkthrough.pdf>. The BTYD package contains models to capture noncontractual purchasing behavior of customers—or, more simply, models that tell the story of people buying until they die (become inactive as customers). The main models presented in the package are the Pareto/NBD, BG/NBD, and BG/BB models.

For Python, LTV can be calculated at <http://srepho.github.io/CLV/CLV> or by the python package `lifetimes` (<https://pypi.python.org/pypi/Lifetimes>) or see home page <https://github.com/CamDavidsonPilon/lifetimes>

RFM Analysis

RFM stands for recency, frequency, and monetization. RFM is thus a method used for analyzing customer value of current customers:

Recency—How recently did the customer purchase?

Frequency—How often do they purchase?

Monetary value—How much do they spend?

This can be quantified in the following way:

Recency = 10 - The number of months that have passed since the customer last purchased

Frequency = Number of purchases in the last 12 months (maximum of 10)

Monetary = Value of the highest order from a given customer (benchmarked against a standard, say, 1000\$ or something relevant)

Alternatively, one can create categories for each metric.

For instance, the recency attribute might be broken into three categories: customers with purchases within the last 90 days, between 91 and 365 days, and longer than 365 days. Such categories may be arrived at by applying business rules, or using a data mining technique, to find meaningful breaks (like CHAID). A commonly used shortcut is to use deciles. One is advised to look at distribution of data before choosing breaks.

Practice

You can see RFM analysis in action at <https://decisionstats.com/2010/10/03/ibm-spss-19-marketing-analytics-and-rfm/> and some R code for it here at <https://github.com/hoxo-m/easyRFM>. You should also see <http://www.dataapple.net/?p=133>. For doing the RFM Analysis in Python, you can see <http://www.marketingdistillery.com/2014/11/02/rfm-customer-segmentation-in-r-pandas-and-apache-spark/>

Biases in Decision Making

Though not often taught in a standard BA or data science course, the author feels biases in decision making should be useful for a data scientist since the data scientist influences decisions. Even though decision making driven by

data should be objective, it is not as it is driven by humans, not machines, and humans make errors due to multiple reasons.

The author would like to point out these resources.

Logical Fallacies—These would help the data scientist in recognizing the erroneous arguments used by various stakeholders in decision making. A fallacy is an incorrect argument in logic and rhetoric that undermines an argument's logical validity. The following refers to <https://yourlogicalfallacyis.com/>, which is created by Jesse Richardson, Andy Smith, Som Meaden, and Flip Creative.

Some of the top logical fallacies are as follows:

- **Ad hominem**—You attacked your opponent's character or personal traits in an attempt to undermine their argument.
- **Slippery slope**—You said that if we allow A to happen, then Z will eventually happen too; therefore A should not happen. The problem with this reasoning is that it avoids engaging with the issue at hand and instead shifts attention to extreme hypotheticals.
- **Straw man**—You misrepresented someone's argument to make it easier to attack. By exaggerating, misrepresenting, or just completely fabricating someone's argument, it's much easier to present your own position as being reasonable.

Cognitive Biases—These impact decisions based on the own psychology of the decision maker. A cognitive bias refers to a systematic pattern of deviation from norm or rationality in judgment, whereby inferences about other people and situations may be drawn in an illogical fashion. Individuals create their own “subjective social reality” from their perception of the input.

Some prominent cognitive biases are as follows:

Confirmation bias—In this the individual only selects data or analysis that supports his preconception and tries to discredit, ignore, or trivialize information that is against the preconceived views. This is a very common confirmation bias in practice. One common reason for doing so is agency-owner conflict in which decision makers in an organization take decisions to maximize their own self-interests (like their annual bonuses) rather than team or organizational goals.

Some other common biases are the following:

Self-serving bias	The tendency to claim more responsibility for successes than failures
Belief bias	Evaluating the strength of an argument by your own belief in the truth or falsity of the conclusion
Framing	Using a narrow approach and scope of the problem to avoid difficult to solve issues
Hindsight bias	The inclination to see past events as being predictable

An excellent article on this is also available at Hilbert (2012).

Statistical Bias Versus Variance

This is a more realistic and statistical description of the kind of error a statistical modeler or a data scientist faces when confronted with data. The following is taken from Fortmann Roe (2012)

Error due to bias: The error due to statistical bias is taken as the difference between the expected (or average) prediction of our model and the correct value that we are trying to predict. Of course you only have one model, so talking about expected or average prediction values might seem a little strange. However, imagine you could repeat the whole model building process more than once: each time you gather new data and run a new analysis, creating a new model. Due to randomness in the underlying datasets, the resulting models will have a range of predictions. Bias measures how far off in general these models' predictions are from the correct value.

Error due to variance: The error due to variance is taken as the variability of a model prediction for a given data point. Again, imagine you can repeat the entire model building process multiple times. The variance is how much the predictions for a given point vary between different realizations of the model (Figure 1.15).

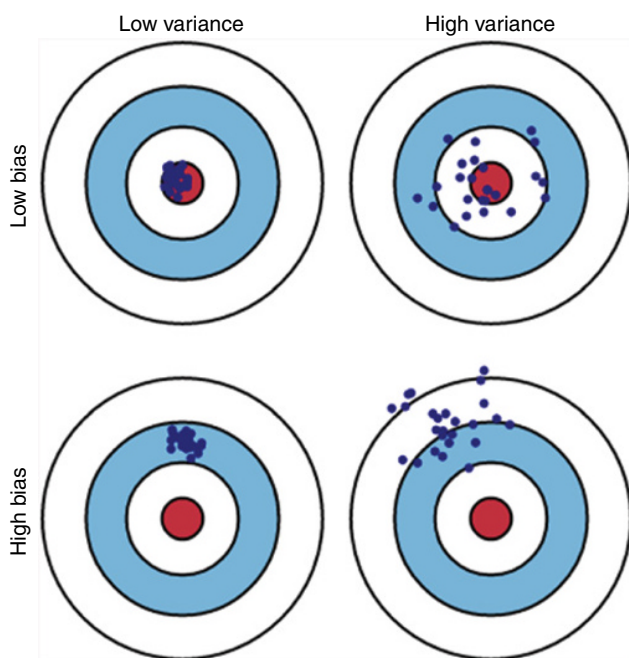


Figure 1.15 Graphical illustration of bias and variance. Source: Scott Fortmann-Roe.
© CSS from Substance.io.

Bibliography

- Jason R. Briggs. *A Playful Introduction to Programming*. No Starch Press, 2012, 344 pp, 978-1-59327-407-8.
- Scott Fortmann Roe (June 2012). Understanding the Bias-Variance Tradeoff. <http://scott.fortmann-roe.com/> and <http://scott.fortmann-roe.com/docs/BiasVariance.html> (accessed April 29, 2017).
- Larry E. Greiner (1998). Evolution and Revolution as Organizations Grow. <https://hbr.org/1998/05/evolution-and-revolution-as-organizations-grow>. May–June 1998 issue of *Harvard Business Review* (accessed May 22, 2017).
- Martin Hilbert (2012). Toward a Synthesis of Cognitive Biases: How Noisy Information Processing Can Bias Human Decision Making. *Psychological Bulletin*, 138(2), 211–237. Available at martinhilbert.net/HilbertPsychBull.pdf. 10.1037/a0025940, <http://dx.doi.org/10.1037/a0025940>, <http://supp.apa.org/psycarticles/supplemental/a0025940/160972-2010-0470-RRAppendix.pdf> (accessed April 29, 2017).
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, 2013, <http://www.R-project.org/> (accessed May 9, 2017).

2

Data Input

2.1 Data Input in Pandas

The pandas library offers many flexible formats for reading in data.

The most commonly used is `read_csv` to read in comma-separated values (from the Internet URL). That is,

```
anscombe=pd.read_csv("https://vincentarelbundock.github.io/Rdatasets/csv/datasets/anscombe.csv")
```

See the top few lines at <http://nbviewer.jupyter.org/gist/decisionstats/3737642751895f470d5c07194302f53e>. © GitHub repository.

```
: anscombe=pd.read_csv("https://vincentarelbundock.github.io/Rdatasets/csv/datasets/anscombe.csv")
```

```
: anscombe
```

	Unnamed: 0	x1	x2	x3	x4	y1	y2	y3	y4
0	1	10	10	10	8	8.04	9.14	7.46	6.58
1	2	8	8	8	8	6.95	8.14	6.77	5.76
2	3	13	13	13	8	7.58	8.74	12.74	7.71
3	4	9	9	9	8	8.81	8.77	7.11	8.84
4	5	11	11	11	8	8.33	9.26	7.81	8.47
5	6	14	14	14	8	9.96	8.10	8.84	7.04
6	7	6	6	6	8	7.24	6.13	6.08	5.25
7	8	4	4	4	19	4.26	3.10	5.39	12.50
8	9	12	12	12	8	10.84	9.13	8.15	5.56
9	10	7	7	7	8	4.82	7.26	6.42	7.91
10	11	5	5	5	8	5.68	4.74	5.73	6.89

Dropping the column

```
: anscombe=anscombe.drop('Unnamed: 0', 1)
```

Or read in csv data from a local file.

See <http://nbviewer.jupyter.org/gist/decisionstats/4142e98375445c5e4174>

```
import pandas as pd #importing packages
import os as os
```

In [2]:

```
#pd.describe_option() #describe options
for customizing
```

In [3]:

```
#pd.get_option("display.memory_usage")
#setting some options
```

In [4]:

```
os.getcwd() #current working directory
```

Out [4]:

```
'/home/ajay'
```

In [5]:

```
os.chdir('/home/ajay/Desktop')
```

In [6]:

```
os.getcwd()
```

Out [6]:

```
'/home/ajay/Desktop'
```

In [7]:

```
a=os.getcwd()
os.listdir(a)
```

Out [7]:

```
['adult.data']
```

In [8]:

```
names2=["age","workclass","fnlwgt","education",
"education-num","marital-status","occupation",
"relationship","race","sex","capital-gain","capital-
loss","hours-per-week","native-country","income"]
```

In [9]:

```
len(names2)
```

Out [9]:

15

```

adult=pd.read_csv("adult.data",header=None)
len(adult)
32562
adult.columns
Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 14], dtype='int64')
adult.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 32562 entries, 0 to 32561
Data columns (total 15 columns):
0      32561 non-null float64
1      32561 non-null object
2      32561 non-null float64
3      32561 non-null object
4      32561 non-null float64
5      32561 non-null object
6      32561 non-null object
7      32561 non-null object
8      32561 non-null object
9      32561 non-null object
10     32561 non-null float64
11     32561 non-null float64
12     32561 non-null float64
13     32561 non-null object
14     32561 non-null object
dtypes: float64(6), object(9)

adult.columns= names2

```

In [10]:

In [11]:

Out [11]:

In [12]:

Out [12]:

In [13]:

In [15]:

7	52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	45	United-States	>50K
---	----	------------------	--------	---------	---	--------------------	-----------------	---------	-------	------	---	---	----	---------------	------

```
In [15]: adult.columns= names2
```

```
In [16]: adult.head(30)
```

out[16]:	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	per-week	native-country
0	39	State-gov	77616	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United States
1	50	Self-emp-not-inc	63311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United States
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United States
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United States
4	26	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba
5	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United States
6	49	Private	160187	9th	5	Married-spouse-absent	Other-service	Not-in-family	Black	Female	0	0	16	Jamaica
7	52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	45	United States
8	31	Private	45761	Masters	14	Never-	Prof-	Not-in-family	White	Female	14064	0	60	United States

We can see the entire list of data input in pandas at <http://pandas.pydata.org/pandas-docs/stable/io.html>.

Source: © pandas 0.19.2 documentation.

The pandas I/O API is a set of top-level reader functions accessed like `pd.read_csv()` that generally return a pandas object.

- read_csv
- read_excel
- read_hdf
- read_sql
- read_json
- read_msgpack (experimental)
- read_html
- read_gbq (experimental)
- read_stata
- read_sas
- read_clipboard
- read_pickle

2.2 Web Scraping Data Input

We can use the beautiful Python library Beautiful Soup to scrape the web for data. The following code scrapes Yelp for comments (see <http://nbviewer.jupyter.org/gist/decisionstats/3385dc84c39109f49b83>).

```
# pip install beautifulsoup4.
from bs4 import BeautifulSoup
```

In [5]:

```
#pip install urllib3
#This library helps in downloading data
import urllib.request
```

2.2.1 Request Data from URL

In [6]:

```
r = urllib.request.urlopen('http://www.
yelp.ca/search?find_loc=Calgary,+AB&cflt=
homeservices').read()
```

In [28]:

```
#Using Beautiful Soup Library to parse
the data
soup = BeautifulSoup(r, "lxml")
type(soup)
```

Out [28]:

```
bs4.BeautifulSoup
```

In [52]:

```
#We find the number of characters
in data downloaded
len(str(soup.prettify()))
```

Out [52]:

```
440689
```

In [53]:

```
#We convert the data to a string format
using str.
#Note in R we use str for structure, but in
Python we use str to convert to character
(like as.character or paste command would do in R)
a=str(soup.prettify())
```

In [57]:

```
# We try and find location of a particular
tag we are interested in.
#Note we are using triple quotes to escape
special characters
a.find(''class="snippet"')
```

Out [57]:

```
352138
```

In [58]:

```
a[352000:358000]
```

Out [58]:

```
'dth="30"/>\n    </a>\n  </div>\n  </div>\n  <div class="media-  
story">\n    <p class="snippet">\n      We\'re the best of bank and broker. We have  
locations so that you know where we are. We\'re connected with all banks, not just one. And we  
pass along our volume discount to get your mortgage...\n    </p>\n  </div>\n</div>\n  </div>\n  </div>\n  </li>\n  <li class="regular-search-result">\n<div class="search-result natural-search-result" data-key="1">\n  <div class="biz-listing-  
large">\n    <div class="main-attributes">\n      <div class="media-block media-block--  
12">\n        <div class="media-avatar">\n          <div class="photo-box pb-90s">\n<a href="/biz/always-affordable-always-available-locksmiths-calgary?search_key=36031">\n\n</a>\n        </div>\n        </div>\n        <div class="media-story">\n          <h3  
class="search-result-title">\n            <span class="indexed-biz-name">\n              1.\n            <a class="biz-name" data-hovercard-id="8QwuvWymqegNxbMgegZ1kg" href="/biz/always-  
affordable-always-available-locksmiths-calgary?search_key=36031">\n              <span>\n                Always Affordable Always Available Locksmiths\n              </span>\n            </a>\n          </span>\n          </h3>\n          <div class="biz-rating biz-rating-large clearfix">\n            <div class="rating-large">\n              <i class="star-img stars_5" title="5.0 star rating">\n                \n              \n            </div>\n            <span class="review-count rating-  
qualifier">\n              7 reviews\n            </span>\n          </div>\n          <div  
class="price-category">\n            <span class="category-str-list">\n              <a  
href="/search?find_loc=Calgary%2C+AB&cf=locksmiths">\n                Keys &\n                Locksmiths\n              </a>\n            </span>\n          </div>\n          <ul  
class="search-result_tags">\n            </ul>\n          </div>\n          </div>\n          </div>\n          <div class="secondary-attributes">\n            <address>\n              1437 Kensington Road NW\n            <br/>\n              Calgary, AB T2N 3R1\n            </address>\n            <span class="offscreen">\n              Phone number\n            </span>\n            <span class="biz-phone">\n              (403) 272-
```

8923\n \n </div>\n </div>\n <div class="snippet-block
review-snippet">\n <div class="media-block">\n <div class="media-avatar">\n <div class="photo-box pb-30s" data-hovercard-id="6G17PcLIXZHTsRUqLgo44A">\n \n \n \n </div>\n </div>\n <div
class="media-story">\n <p class="snippet">\n We were very pleased with the
quick, professional, quality service we got from this company. \xa0When booking the appointment,
the person on the phone was efficient and helpful, and although I... \n </p>\n </div>\n </div>\n </div>\n </div>\n \n <li class="regular-
search-result">\n <div class="search-result natural-search-result" data-key="2">\n <div class="biz-listing-large">\n <div class="main-attributes">\n <div class="media-
block media-block--12">\n <div class="media-avatar">\n <div class="photo-box
pb-90s">\n \n \n \n </div>\n </div>\n <div class="media-story">\n <h3 class="search-result-title">\n \n 2.\n <a class="biz-name" data-hovercard-id="DG-pdTKaegi87Df9xQvp2A" href="/biz/golden-acre-
garden-sentres-calgary?search_key=36031">\n \n Golden Acre
Garden Sentres\n \n \n \n </h3>\n <div class="biz-rating biz-rating-large clearfix">\n <div class="rating-large">\n <i class="star-img stars_4" title="4.0 star rating">\n \n \n </div>\n <span class="review-count rating-
qualifier">\n 13 reviews\n \n </div>\n <div class="price-category">\n \n <span
class="business-attribute price-range">\n '

In [21]:

```
#Lets try and find the list of phone numbers.
We note both the HTML tag and the class for it.
# We use the find_all function
letters = soup.find_all("span", class_="biz-phone")
letters[1:100]
```

Out [21]:

```
[<span class="biz-phone">
  (403) 272-8923
</span>, <span class="biz-phone">
  (403) 274-4286
</span>, <span class="biz-phone">
  (403) 918-4475
</span>, <span class="biz-phone">
  (403) 681-4376
</span>, <span class="biz-phone">
  (403) 454-0243
</span>, <span class="biz-phone">
  (403) 457-6333
</span>, <span class="biz-phone">
  (403) 899-0599
</span>, <span class="biz-phone">
  (403) 452-2881
</span>, <span class="biz-phone">
  (587) 229-0673
</span>, <span class="biz-phone">
  (403) 770-4700
</span>]
```

In [22]:

```
#Lets try and see the feedback given by users.
letters2 = soup.find_all("p", class_="snippet")
letters2[1:100]
```

Out [22]:

```
[<p class="snippet">
  We were very pleased with the quick,
professional, quality service we got from this company.
When booking the appointment, the person on the phone
was efficient and helpful, and although I...
</p>, <p class="snippet">
  Yesterday I was at Golden Acres and
carelessly had let myself become dehydrated, but hadn't
realized what was going on. An employee, Rachel,
recognized I was in trouble, made suggestions,...
</p>, <p class="snippet">
```

Holy crap, I believe I have died and gone to heaven... I can't believe that I just discovered that there is actually a store that sells mid century modern furniture and accessories in town. I...

</p>, <p class="snippet">

Really appreciate the help I've received from Mark at Mortgage Alliance. On two occasions he sent me back to my bank with some advice to get what I was looking for and saved me a lot of grief...

</p>, <p class="snippet">

Such a wicked venue, place, space, I'm not even sure what the term is. I've been here on a couple occasions, the first time was a random Saturday in Inglewood and popped in. We got to meet the...

</p>, <p class="snippet">

I called Carol mid-afternoon on Monday for a move-out clean. She showed up bright and early the next morning with her supplies, and (dare I say) insanely beautiful and outgoing colleague, Liz....

</p>, <p class="snippet">

...Did not think I'd be writing a review on a furnace company but here I am. Right now in the middle of troubleshooting a heating issue. Thanks to Flash Furnace I am identifying the issue...

</p>, <p class="snippet">

F2 Furnishings is a great place to shop for furniture and other home decor. The company really supports local artists and designers. A lot of their pieces are originals from local crafts...

</p>, <p class="snippet">

Brandon was prompt in answering any questions we had prior to the move. On the day of the move they were on time, efficient, and professional. Brandon and Jesse took especial care of our...

</p>, <p class="snippet">

I am a huge fan of what the Niklas Group has done to my community. I live just a block away from the Casel Marche building on 17th ave and I'm really impressed with the sense of community this...

</p>]

In [23]:
type(letters2)

Out [23]:

```
bs4.element.ResultSet
```

```
In [24]:
```

```
str(letters2)[1:1000]
```

```
Out [24]:
```

```
'<p class="snippet">We\'re the best of bank and broker.
We have locations so that you know where we are. We\'re
connected with all banks, not just one. And we pass
along our volume discount to get your mortgage...</p>,
<p class="snippet">\n    We were very pleased with the
quick, professional, quality service we got from this
company. \xa0When booking the appointment, the person
on the phone was efficient and helpful, and although
I...\n    </p>, <p class="snippet">\n    Yesterday I was at
Golden Acres and carelessly had let myself become
dehydrated, but hadn\'t realized what was going on. \xa0An
employee, Rachel, recognized I was in trouble, made
suggestions,...\n    </p>, <p class="snippet">\n Holy crap,
I believe I have died and gone to heaven... I can\'t
believe that I just discovered that there is actually
a store that sells mid century modern furniture and
accessories in town. I...\n</p>, <p class="snippet">\n    '
```

```
In [25]:
```

```
str(letters2).count("service")
```

```
Out [25]:
```

```
1
```

2.3 Data Input from RDBMS

After csv files and web scraping, the last type of data input we consider is from relational database management system (RDBMS) databases. Here is a brief note on RDBMS first to understand them. SQL is a domain-specific language used in programming and designed for managing data held in an RDBMS or for stream processing in a relational data stream management system (RDSMS). SQL has been designed for managing data in RDBMSs like Oracle, MySQL, MS SQL Server, and IBM DB2 besides PostgreSQL:

- SQL is one of the first commercial languages used for Edgar F. Codd's relational model, also described in his influential 1970 paper, "A Relational Model of Data for Large Shared Data Banks."

Below is a quote from Edgar F. Codd's 1970 paper, "A Relational Model of Data for Large Shared Data Banks."

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a

satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information. Existing non-inferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on n -ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

Properties of Databases:

A database transaction, however, must be ACID compliant. ACID stands for atomic, consistent, isolated, and durable.

- **Atomic:** A transaction must be either completed with all of its data modifications or may not.
- **Consistent:** At the end of the transaction, all data must be left consistent.
- **Isolated:** Data modifications performed by a transaction must be independent of other transactions.
- **Durable:** At the end of transaction, effects of modifications performed by the transaction must be permanent in system.

To counter ACID, the consistent services provide *basically available, soft state, eventual consistency* (BASE) features.

Earlier, SQL was a *de facto* language for the generation of information technology professionals due to the fact that data warehouses consisted of one RDBMS or RDSMS. The simplicity and beauty of the language enabled data warehousing professionals to query data and provide it to business analysts.

RDBMS are often suitable only for structured information. For unstructured information, newer databases like MongoDB, CouchDB, and HBase (from Hadoop) prove to be a better fit. Part of this is a trade-off in databases, which is due to the CAP theorem (Figure 2.1).

CAP theorem states that at best we can aim for two of the following three properties:

- **Consistency**—This means that data in the database remains consistent after the execution of an operation.
- **Availability**—This means that the database system is always on to ensure availability.
- **Partition Tolerance**—This means that the system continues to function even if the transfer of information between the servers is unreliable.

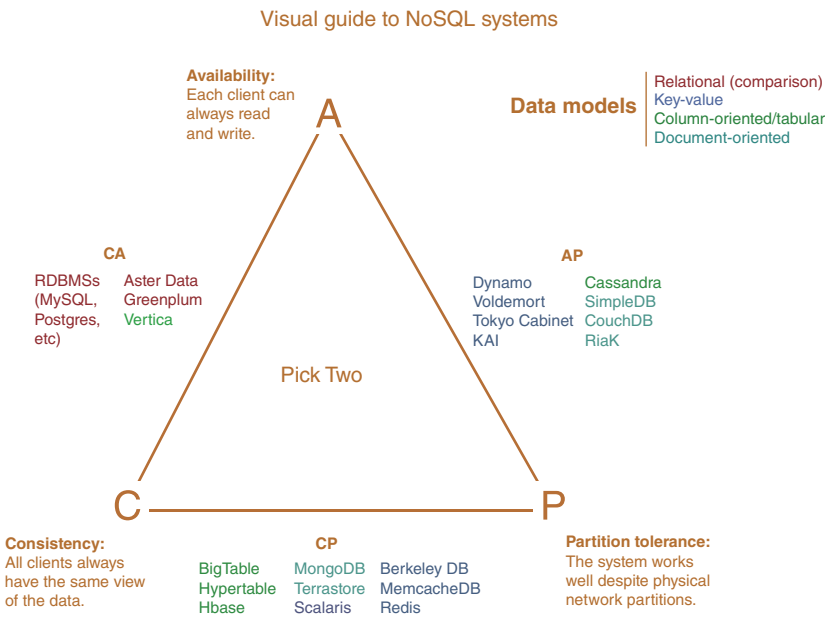


Figure 2.1 <http://blog.nahurst.com/visual-guide-to-nosql-systems>. *Source:* Courtesy: Nathan Hurst.

2.3.1 Windows Tutorial

<https://www.postgresql.org/>. PostgreSQL is a powerful, open-source object-relational database system (ORDBMS). It has more than 15 years of active development and a proven architecture.

PostgreSQL, often simply **Postgres**, is an ORDBMS—that is, an RDBMS with additional (optional use) “object” features—with an emphasis on extensibility and standards compliance.

PostgreSQL is developed by the PostgreSQL Global Development Group. Some general PostgreSQL limits are included in the table as follows.

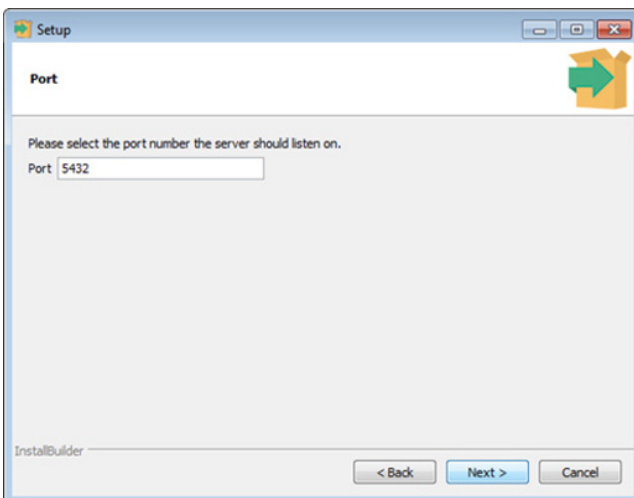
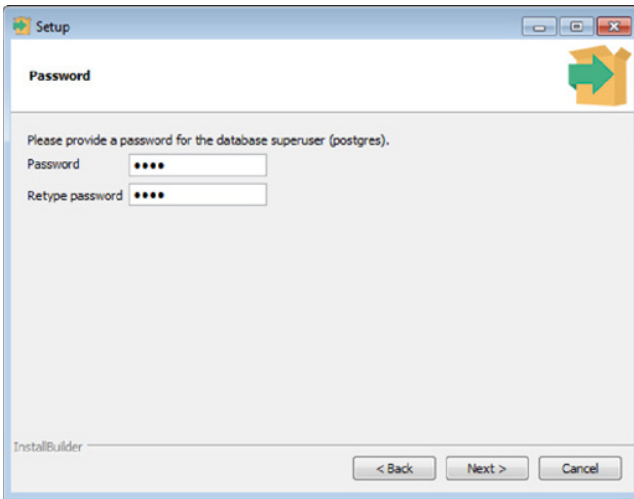
Limit	Value
Maximum database size	Unlimited
Maximum table size	32 TB
Maximum row size	1.6 TB
Maximum field size	1 GB
Maximum rows per table	Unlimited
Maximum columns per table	250–1600 depending on column types
Maximum indexes per table	Unlimited

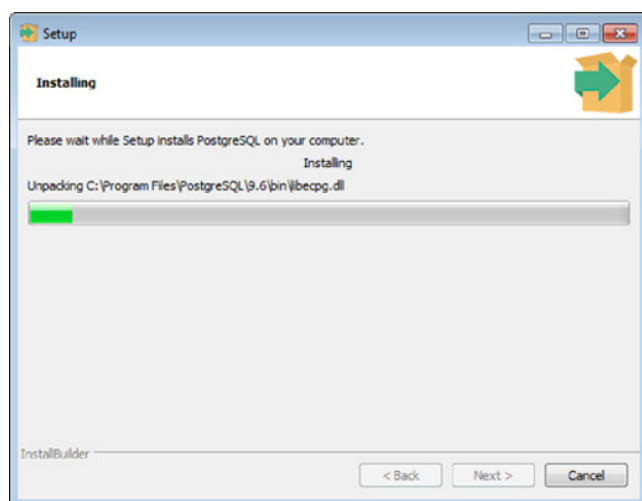
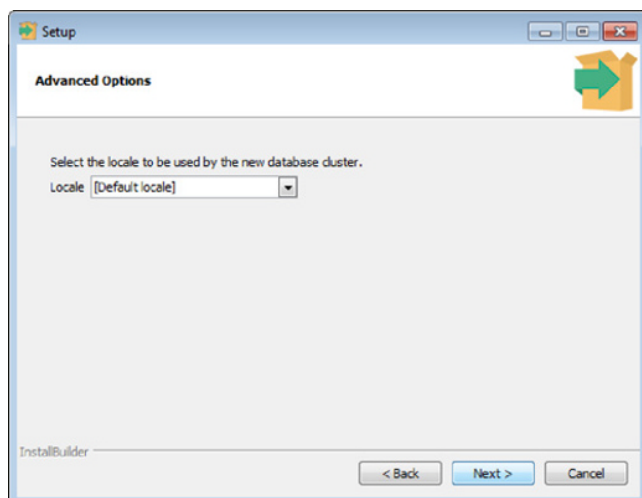
Download the Database <https://www.postgresql.org/download/> and <https://www.postgresql.org/download/windows/>.

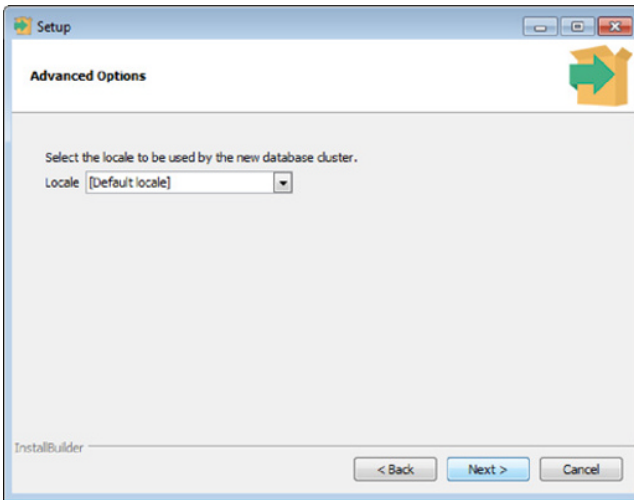
Two choices—we go for enterprise db.

<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads#windows>.

2.3.2 137 Mb Installer



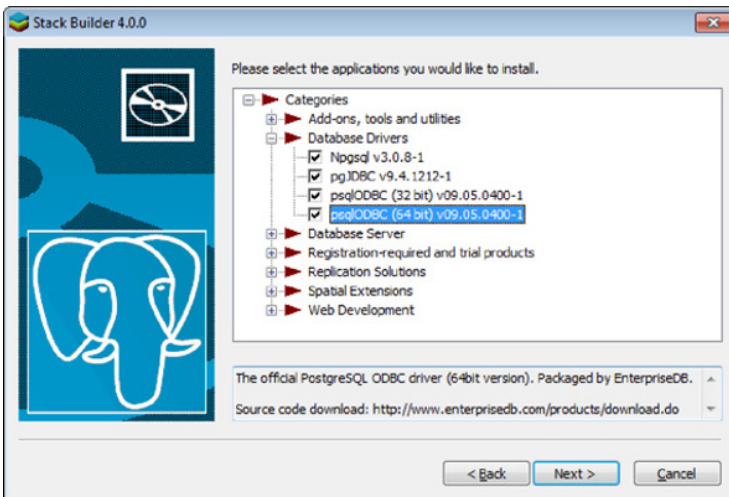


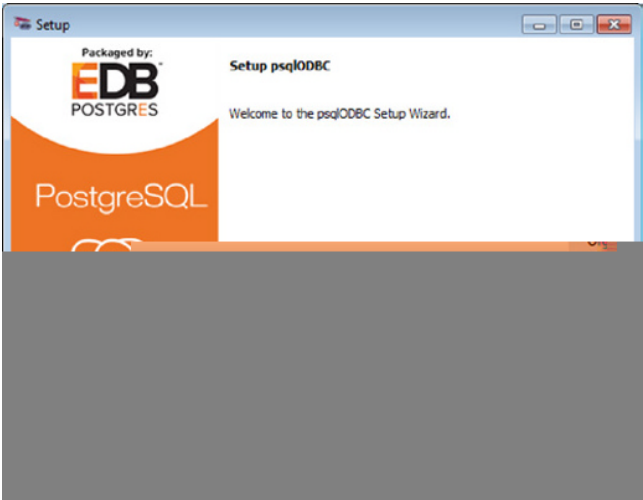


2.3.3 Configuring ODBC

Download and Install ODBC Driver

Use the stack builder to check option ODBC Driver. Open Database Connectivity (**ODBC**) is an open standard application programming interface (API) for accessing a database. By using **ODBC** statements in a program, you can access files in a number of different databases, including Access, dBase, DB2, Excel, and Text.

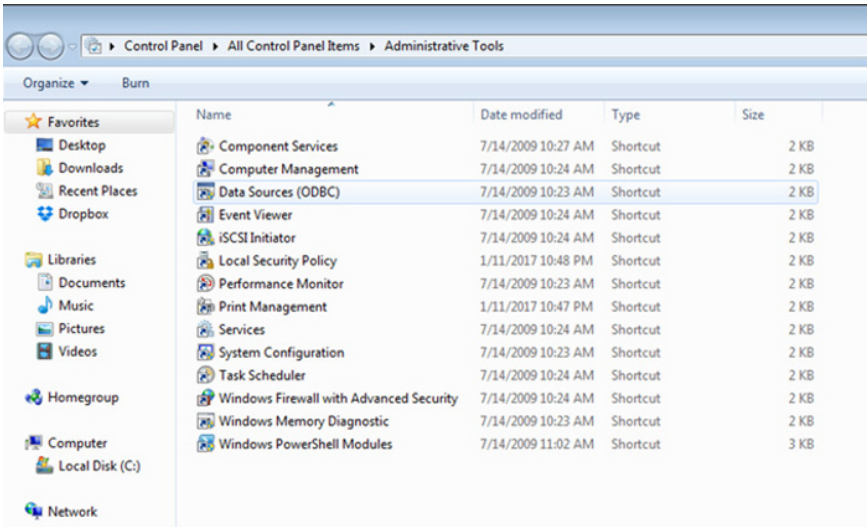




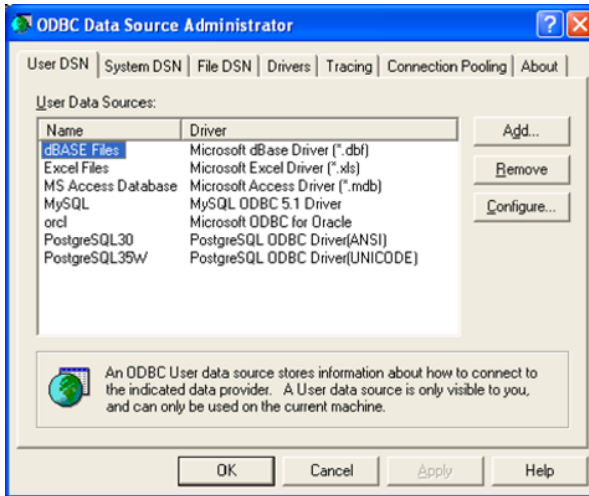
After installing the ODBC Driver, now you need to make sure your OS knows. It is time for connections. Connect it to a data source name (DSN) (using Control Panel) in Windows.

Go to Control Panel> Administrative Tools> ODBC Connections.

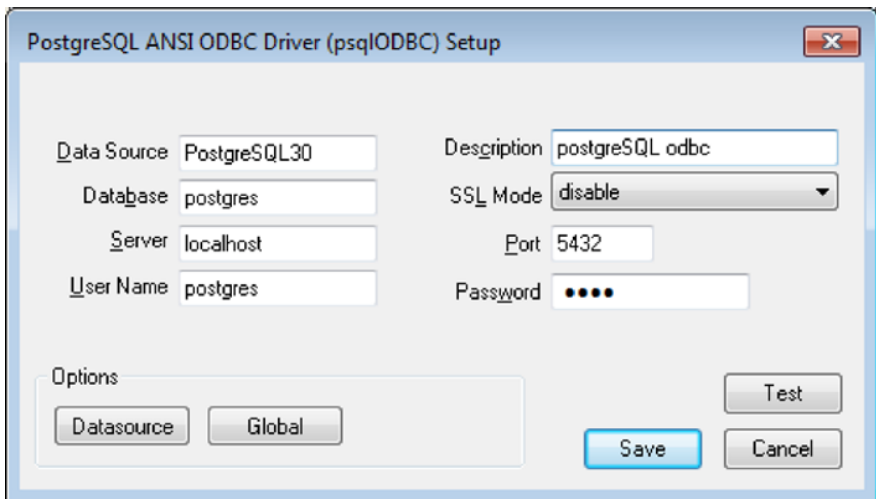
A **DSN** is a data structure that contains the information about a specific database that an ODBC driver needs in order to connect to it.



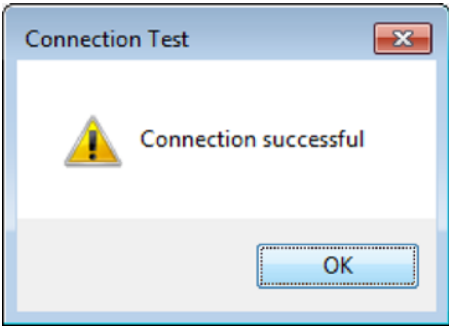
Click Add User DSN.



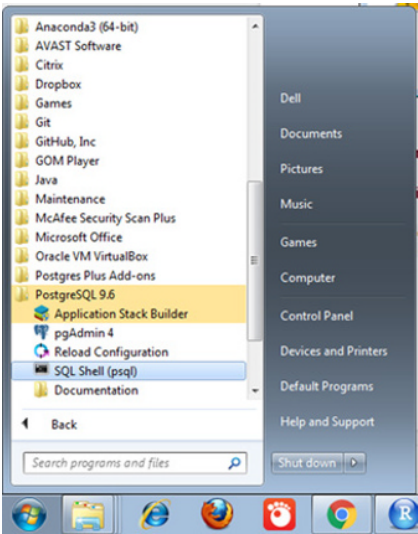
Put the options as below. Remember to put the same port and the same password as we did in the steps mentioned earlier.



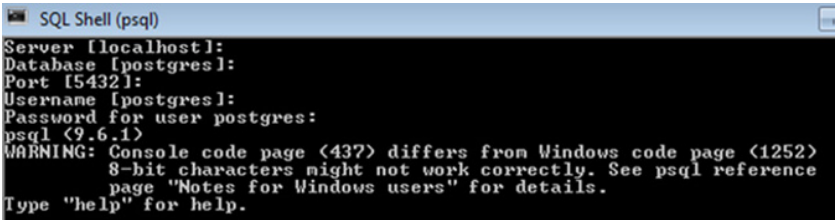
Click on the Test box (above Cancel and below Password). If the connection is successful, you should see this, or else you need to go back and find what you got wrong (mostly password or port).



Now open your Postgres using command line (see in Programs).

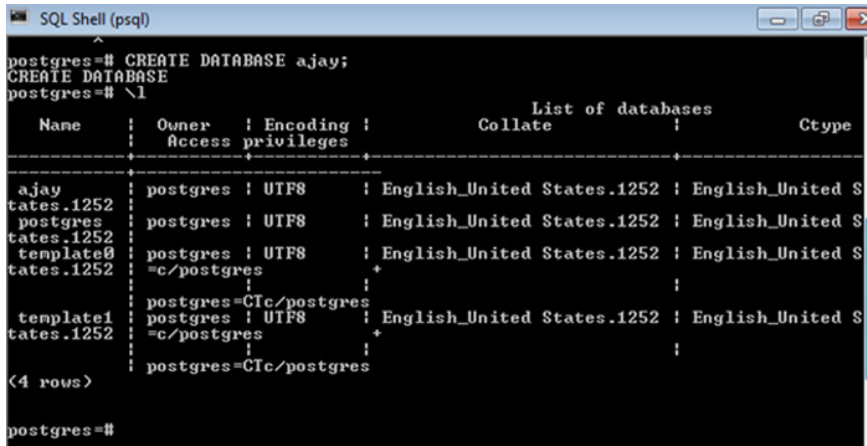


Log on to Postgres using the password.



Then create a database using the syntax below (note the smaller case of the database name and the;) to complete the command. Then type \l to see all available databases.

```
postgres=# CREATE DATABASE ajay;
postgres=# \l
```



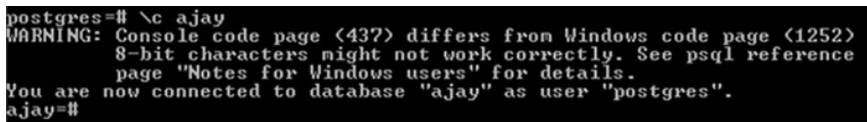
```
SQL Shell (psql)
postgres=# CREATE DATABASE ajay;
CREATE DATABASE
postgres=# \l
```

Name	Owner	Access privileges	Encoding	Collate	Ctype
ajay	postgres		UTF8	English_United States.1252	English_United S
postgres	postgres		UTF8	English_United States.1252	English_United S
template0	postgres		UTF8	English_United States.1252	English_United S
template1	postgres		UTF8	English_United States.1252	English_United S

```
<4 rows>
postgres=#
```

Now we use \c databasename to connect to the database.

```
\c ajay;
```



```
postgres=# \c ajay
WARNING: Console code page (437) differs from Windows code page (1252)
8-bit characters might not work correctly. See psql reference
page "Notes for Windows users" for details.
You are now connected to database "ajay" as user "postgres".
ajay=#
```

Now we create tables inside the database. This is done using the CREATE TABLE command. Following are examples of CREATE TABLE command.

```
CREATE TABLE table_name(
    column1 datatype,
    column2 datatype,
    .....
    columnN datatype,
    PRIMARY KEY( one or more columns )
);
```

Example

```
CREATE TABLE weather (
    city          varchar(80),
    temp_lo       int,          -- low temperature
    temp_hi       int,          -- high temperature
    prcp          real,         -- precipitation
    date          date
);
```

PostgreSQL supports the standard SQL types `int`, `smallint`, `real`, `double precision`, `char(N)`, `varchar(N)`, `date`, `time`, `timestamp`, and `interval`, as well as other types of general utility and a rich set of geometric types. PostgreSQL can be customized with an arbitrary number of user-defined data types. Consequently, type names are not syntactical key words, except where required to support special cases in the SQL standard.

From <https://www.postgresql.org/docs/8.1/static/tutorial-table.html>

```
CREATE TABLE cities (
    name          varchar(80),
    location      point
);
```

For our use case let's make a table suited for sales and business:

```
CREATE TABLE SALES (
    CUSTOMER_ID INT PRIMARY KEY      NOT NULL, --unique
id of customers
    SALES        int NOT NULL,       --sales in rupees
date           date,               --date of sale
    PRODUCT_ID  INT      NOT NULL
);
```

I then check the table created using `\d`.

```
ajay=#
ajay=# CREATE TABLE SALES(
ajay=#   CUSTOMER_ID INT PRIMARY KEY      NOT NULL, --unique id of customers
ajay=#   SALES        int NOT NULL,       --sales in rupees
ajay=#   date         date,               --date of sale
ajay=#   PRODUCT_ID  INT      NOT NULL
ajay=# );
ajay=# CREATE TABLE
ajay=# \d
List of relations
Schema | Name  | Type  | Owner
-----+-----+-----+-----
public | sales | table | postgres
(1 row)

ajay=# _
```

Now try \d tablename to get details of the table.
Here \d sales.

```

ajay=# \d
          List of relations
 Schema | Name  | Type  | Owner
-----+-----+-----+-----
 public | sales | table | postgres
(1 row)

ajay=# \d ajay
Did not find any relation named "ajay".
ajay=# \d sales
          Table "public.sales"
   Column   | Type   | Modifiers
-----+-----+-----
 customer_id | integer | not null
 sales       | integer | not null
 date        | date   |
 product_id  | integer | not null
Indexes:
    "sales_pkey" PRIMARY KEY, btree (customer_id)

```

Quit using \q
I can delete a table using drop table tablename.

```

ajay=# \d
          List of relations
 Schema | Name  | Type  | Owner
-----+-----+-----+-----

```

Now let's copy some data (see <http://bit.ly/2postgres>) into my database table.
First of all my data is in the same format.

```

ajay=# \d
          List of relations
Schema | Name  | Type  | Owner
-----+-----+-----+-----
public | sales | table | postgres
(1 row)

ajay=# \d ajay
Did not find any relation named "ajay".
ajay=# \d sales
          Table "public.sales"
   Column   | Type   | Modifiers
-----+-----+-----
customer_id | integer | not null
sales       | integer | not null
date       | date   |
product_id | integer | not null
Indexes:
    "sales_pkey" PRIMARY KEY, btree (customer_id)

ajay=#
ajay=# \copy sales from C:\Users\Dell\Downloads\data1.csv DELIMITER ',' CSV;
Invalid command \. Try \? for help.
ajay=# \copy sales from C:\Users\Dell\Downloads\data1.csv DELIMITER ',' CSV;
ERROR:  duplicate key value violates unique constraint "sales_pkey"
DETAIL:  Key (customer_id)=(10024) already exists.
CONTEXT: COPY sales, line 4
ajay=# \copy sales from C:\Users\Dell\Downloads\data1.csv DELIMITER ',' CSV;
COPY 500
ajay=#

```

We use the Copy command to load the data.

```

\copy sales from C:/Users/Dell/Downloads/data1.csv
DELIMITER ',' CSV;

```

Note an error value when we try and import data with a duplicate primary key. We rectify our data and then do the import again. COPY 500 shows 500 records imported successfully.

And then we can make a connection in R to do analysis in R on the Postgres data (see <http://rpubs.com/newajay/RODBC>).

```

install.packages("RODBC") #installing the package RODBC
library(RODBC) #loading the package RODBC
odbcDataSources() # to check all available ODBC data
sources

```

```

#creating a Database connection
# for username,password,database name and DSN name

chan=odbcConnect ("PostgreSQL30", "postgres; Password=root;
Database=ajay")

```

```
#to list all table names
sqlTables(chan)

#and fetch some data
sqlFetch(chan, "sales", max=10)
```

```
#creating a Database connection
# for username,password,database name and DSN name

chan=odbcConnect("PostgreSQL30","postgres;Password=root;Database=ajay")

#to list all table names
sqlTables(chan)
```

```
##  TABLE_CAT TABLE_SCHEM TABLE_NAME TABLE_TYPE REMARKS
##  1          ajay        public      sales      TABLE
```

```
sqlFetch(chan, "sales", max=10)
```

```
##  customer_id sales      date product_id
##  1          10001  5230  2017-02-07      524
##  2          10002  2781  2017-05-12      469
##  3          10003  2083  2016-12-18      917
##  4          10004   214  2015-01-19      354
##  5          10005  9407  2016-09-26      292
##  6          10006  4705  2015-10-17      380
##  7          10007  4729  2016-01-02      469
##  8          10008  7715  2015-09-12      480
##  9          10009  9898  2015-04-05      611
## 10          10010  5797  2015-08-13      959
```

In Python the package to do the same is SQLAlchemy as the code in the following text shows.

```
import psycopg2
import pandas as pd
import sqlalchemy as sa
import time
import seaborn as sns
import re
```

In [17]:

```
parameters = {
    'username': 'postgres',
    'password': 'root',
    'server': 'localhost',
    'database': 'ajay'
}
```

In [19]:

```
connection= 'postgresql://{username}:
{password}@{server}:5432/{database}'.
format(**parameters)
```

In [20]:

```
# The database connection
print (connection)
```

```
postgresql://postgres:root@localhost:
5432/ajay
```

In [21]:

```
engine = sa.create_engine(connection_
string, encoding="utf-8")
```

In [31]:

```
insp = sa.inspect(engine)
db_list = insp.get_schema_names()
print(db_list)
['information_schema', 'public']
```

In [37]:

```
dir(engine)
```

Out [37]:

```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattr__',
 '__gt__',
 '__hash__',
 '__init__',
 '__le__',
 '__lt__',
 '__module__',
```

```

'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__weakref__',
'_connection_cls',
'_echo',
'_execute_clauseelement',
'_execute_compiled',
'_execute_default',
'_execution_options',
'_has_events',
'_optional_conn_ctx_manager',
'_run_visitor',
'_should_log_debug',
'_should_log_info',
'_trans_ctx',
'_wrap_pool_connect',
'begin',
'connect',
'contextual_connect',
'create',
'dialect',
'dispatch',
'dispose',
'driver',
'drop',
'echo',
'engine',
'execute',
'execution_options',
'has_table',
'logger',
'logging_name',
'name',
'pool',

```

```
'raw_connection',
'run_callable',
'scalar',
'table_names',
'transaction',
'update_execution_options',
'url']
```

In [36]:

```
engine.table_names()
```

Out [36]:

```
['sales']
```

In [39]:

```
data3= pd.read_sql_query('select * from
"sales" limit 10',con=engine)
```

In [40]:

```
data3.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
customer_id    10 non-null int64
sales          10 non-null int64
date           10 non-null object
product_id     10 non-null int64
dtypes: int64(3), object(1)
memory usage: 400.0+ bytes
```

In [41]:

```
data3.head()
```

Out [41]:

	customer_id	sales	date	product_id
0	10001	5230	2017-02-07	524
1	10002	2781	2017-05-12	469
2	10003	2083	2016-12-18	917
3	10004	214	2015-01-19	354
4	10005	9407	2016-09-26	292

In []:

Finally in PostgreSQL to delete database, use Drop Database.

3

Data Inspection and Data Quality

To dos: how to delete values and how to convert pandas to numpy array and back, data, table and dplyr in r, and hmisc in R.

3.1 Data Formats

In R we can use the `as` operator to change from one data format to another.

In Python we can use `str` and `int` to convert to string and integer formats. We can use `split` to convert string to list.

Numeric—We use `int` and `float` functions to convert data to numeric types integer and float, respectively.

This is demonstrated in the following code. Note in R the index starts from 1 and in Python it starts from 0.

```
import re
import numpy as np
import pandas as pd
numlist=["$10000","$20,000","30,000",40000,"50000    "]
for i,value in enumerate(numlist):
    numlist[i]=re.sub(r"([\$,])","",str(value))

numlist
['10000', '20000', '30000', '40000', '50000    ']
int(numlist[1])
20000
for i,value in enumerate(numlist):
    numlist[i]=int(value)
numlist
[10000, 20000, 30000, 40000, 50000]
```

```

np.mean(numlist)
30000.0
numlist2=str(numlist)
numlist2.split(None,0)
['[10000, 20000, 30000, 40000, 50000]']
numlist2.split(None,0)[0]
'[10000, 20000, 30000, 40000, 50000]'

```

3.1.1 Converting Strings to Date Time in Python

```

from datetime import datetime
datetime_object = datetime.strptime('Jun 7 2016 1:33PM',
'%b %d %Y %I:%M%p')

```

R has lubridate package (<https://cran.r-project.org/web/packages/lubridate/lubridate.pdf>) for easy conversion of strings of data to date and time, but Python has the date-time package. See examples of lubridate at <http://rpubs.com/newajay/datesquality> and <http://rpubs.com/ajaydecis/lubridate>

```

adob="7 June 1977 19:20"
library(lubridate)
adob2=dmy_hm(adob)
adob2

## [1] "1977-06-07 19:20:00 UTC"

```

```

mydob="1 June 1981"
mydob2=dmy(mydob)
mydob2

## [1] "1981-06-01 UTC"

```

```

#give me your age in secs
#my dob = 1 june 1981

```

```

pd=Sys.Date() #Date right now
pt=Sys.time() # Date Time Right Now
#give me how old you are from me
# hint I was born on 7 june 1977 at 1920 hours

```

```

difftime(adob2,pt,units="secs")
## Time difference of -1203358055 secs
difftime(adob2,mydob2,units="days")
## Time difference of -1454.194 days

```

Lubridate Example 2

```
library(lubridate)
##
## Attaching package: 'lubridate'
## The following object is masked from 'package:base':
##
##      date
classdata=c("1-April-1977",
            "April/2/2014",
            "1Jun2016")

lapply(classdata,dmy)
## Warning: All formats failed to parse. No formats found.
## [[1]]
## [1] "1977-04-01"
##
## [[2]]
## [1] NA
##
## [[3]]
## [1] "2016-06-01"
is.na(lapply(classdata,dmy))

## Warning: All formats failed to parse. No formats found.
## [1] FALSE  TRUE FALSE
ifelse(!is.na(lapply(classdata,dmy))
      ,lapply(classdata,dmy),lapply(classdata,mdy))
## Warning: All formats failed to parse. No formats found.
## Warning: All formats failed to parse. No formats found.
## Warning: All formats failed to parse. No formats found.

## Warning: All formats failed to parse. No formats found.
## [[1]]
## [1] "1977-04-01"
##
## [[2]]
## [1] "2014-04-02"
##
## [[3]]
## [1] "2016-06-01"
```

Date Time—We use the `datetime` module to convert string data to date-time format and then do numeric operations on it. The following example shows using the `strptime` function to parse the date. We can then use the “now” function to find the difference in days from current date. This creates a `datetime` delta object. (<http://nbviewer.jupyter.org/gist/decisionstats/246c835576a9537a037768ab30a45f4a>)

```
from datetime import datetime

date_object=datetime.strptime("7nov-2007", "%d%-Y")

date_object

datetime.datetime(2007, 11, 7, 0, 0)

print(format(date_object.year))
print(format(date_object.month))
print(format(date_object.day))
print(format(date_object.hour))

2007
11
7
0

datetime.now()
datetime.datetime(2016, 10, 30, 16, 38, 6, 260123)
datetime.now()-date_object

datetime.timedelta(3280, 59947, 411736)
a=datetime.now()-date_object

a.days

3280
a.seconds

59968
```

We can use `timeit` package in Python for finding time of execution of code snippets (<https://docs.python.org/2/library/timeit.html>). This is done by `system.time()` in R.

For converting dates into strings, use `strftime` function (the help is at <https://docs.python.org/2/library/time.html#time.strftime>).

Directive	Meaning	Notes
%a	Locale's abbreviated weekday name	
%A	Locale's full weekday name	
%b	Locale's abbreviated month name	
%B	Locale's full month name	
%c	Locale's appropriate date and time representation	
%d	Day of the month as a decimal number [01,31]	
%H	Hour (24-hour clock) as a decimal number [00,23]	
%I	Hour (12-hour clock) as a decimal number [01,12]	
%j	Day of the year as a decimal number [001,366]	
%m	Month as a decimal number [01,12]	
%M	Minute as a decimal number [00,59]	
%p	Locale's equivalent of either AM or PM	(1)
%S	Second as a decimal number [00,61]	(2)
%U	Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0	(3)
%w	Weekday as a decimal number [0(Sunday),6]	
%W	Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0	(3)
%x	Locale's appropriate date representation	
%X	Locale's appropriate time representation	
%y	Year without century as a decimal number [00,99]	
%Y	Year with century as a decimal number	
%Z	Time zone name (no characters if no time zone exists)	
%%	A literal "%" character	

3.1.2 Converting Data Frame to NumPy Arrays and Back in Python

To convert pandas data frame `df` to NumPy, use the `values` command:

```
a=df.iloc[:,1:]
b=df.iloc[:,1:].values
```

```
print (type(df))
print (type(a))
print (type(b))
```

To convert to a pandas data frame, use DataFrame with values for index and column. (See code at <https://nbviewer.jupyter.org/gist/decisionstats/b818917b37807fa0ded41522928f26af>).

```
titanic=pd.read_csv("https://vincentarelbundock.github.io/Rdatasets/csv/datasets/Titanic.csv")
```

```
titanic=titanic.drop('Unnamed: 0', 1)
```

```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1313 entries, 0 to 1312
Data columns (total 6 columns):
Name          1313 non-null object
PClass        1313 non-null object
Age           756 non-null float64
Sex           1313 non-null object
Survived      1313 non-null int64
SexCode       1313 non-null int64
dtypes: float64(1), int64(2), object(3)
memory usage: 61.6+ KB
```

```
titanic.head()
```

	Name	PClass	Age	Sex	Survived	SexCode
0	Allen, Miss Elisabeth Walton	1st	29.00	female	1	1
1	Allison, Miss Helen Loraine	1st	2.00	female	0	1
2	Allison, Mr Hudson Joshua Creighton	1st	30.00	male	0	0
3	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.00	female	0	1
4	Allison, Master Hudson Trevor	1st	0.92	male	1	0

```
a=titanic.iloc[:,1:]
b=titanic.iloc[:,1:].values
```

```
print (type(titanic))
print (type(a))
print (type(b))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'numpy.ndarray'>
```

a

	PClass	Age	Sex	Survived	SexCode
0	1st	29.00	female	1	1
1	1st	2.00	female	0	1
2	1st	30.00	male	0	0
3	1st	25.00	female	0	1
4	1st	0.92	male	1	0
5	1st	47.00	male	1	0
6	1st	63.00	female	1	1
7	1st	39.00	male	0	0
8	1st	58.00	female	1	1
9	1st	71.00	male	0	0
...	...	...	...	...	...
1306	3rd	NaN	female	0	1
1307	3rd	NaN	female	0	1
1308	3rd	27.00	male	0	0
1309	3rd	26.00	male	0	0
1310	3rd	22.00	male	0	0
1311	3rd	24.00	male	0	0
1312	3rd	29.00	male	0	0

1313 rows × 5 columns

b

```
array([[ '1st', 29.0, 'female', 1, 1],
       [ '1st', 2.0, 'female', 0, 1],
       [ '1st', 30.0, 'male', 0, 0],
       ...,
       [ '3rd', 22.0, 'male', 0, 0],
       [ '3rd', 24.0, 'male', 0, 0],
       [ '3rd', 29.0, 'male', 0, 0]], dtype=object)
```

```
titanic.columns[1:]
```

```
Index(['PClass', 'Age', 'Sex', 'Survived', 'SexCode'],
      dtype='object')
```

```
titanic.as_matrix(columns=titanic.columns[1:])
```

```
array([[ '1st', 29.0, 'female', 1, 1],
       [ '1st', 2.0, 'female', 0, 1],
       [ '1st', 30.0, 'male', 0, 0],
       ...,
       [ '3rd', 22.0, 'male', 0, 0],
       [ '3rd', 24.0, 'male', 0, 0],
       [ '3rd', 29.0, 'male', 0, 0]])
```

```
        ['3rd', 22.0, 'male', 0, 0],
        ['3rd', 24.0, 'male', 0, 0],
        ['3rd', 29.0, 'male', 0, 0]], dtype=object)

data=titanic.as_matrix(columns=titanic.columns[1:])

len(data)
1313

range(0,len(data))
range(0, 1313)

g=pd.DataFrame(data=data[0:,0:],      # values
               index=range(0,len(data)), # 1st column as index
               columns=titanic.columns[1:]) # 1st row as the
               column names

g.head()
```

	PClass	Age	Sex	Survived	SexCode
0	1st	29	female	1	1
1	1st	2	female	0	1
2	1st	30	male	0	0
3	1st	25	female	0	1
4	1st	0.92	male	1	0

3.2 Data Quality

We can use the re package for regular expressions. The following example shows how to replace non-numeric values in data to clean it up for numerical analysis. We use re.sub to replace the values of \$, command, and whitespace.

```
import re
import numpy as np

numlist=["$10000", "$20,000", "30,000", 40000, "50000  "]

help(re.sub)
```

Help on function sub in module re:

`sub(pattern, repl, string, count=0, flags=0)`
 Return the string obtained by replacing the leftmost non-overlapping occurrences of the pattern in string by the replacement `repl`. `repl` can be either a string or a callable;
 if a string, backslash escapes in it are processed. If it is a callable, it's passed the match object and must return a replacement string to be used.

```
for i,value in enumerate(numlist):
```

```
    numlist[i]=re.sub(r"([$,])","",str(value))  

    numlist[i]=int(numlist[i])
```

Here `re.sub` replaces the `$` and patterns with nothing (`""`) from each value of the list just like `gsub` does in R. In python `str` converts an object to string just like `paste` does in R. In the next step, `int` converts the object to numeric values (integer) just as `as.numeric` does in R:

```
print(numlist)
```

```
[10000, 20000, 30000, 40000, 50000]
```

```
np.mean(numlist)
```

```
30000.0
```

```
help(enumerate)
```

Help on class `enumerate` in module `builtins`:

```
class enumerate(object)
|   enumerate(iterable[, start]) -> iterator for index,
|       value of iterable
|
|   Return an enumerate object. iterable must be another
|       object that supports
|       iteration. The enumerate object yields pairs
|       containing a count (from
|       start, which defaults to zero) and a value yielded
|       by the iterable argument.
```

```

    enumerate is useful for obtaining an indexed list:
        (0, seq[0]), (1, seq[1]), (2, seq[2]), ...

```

```

Methods defined here:

```

```

__getattr__(self, name, /)
    Return getattr(self, name).

__iter__(self, /)
    Implement iter(self).

__new__(*args, **kwargs) from builtins.type
    Create and return a new object.  See help(type)
    for accurate signature.

__next__(self, /)
    Implement next(self).

__reduce__(...)
    Return state information for pickling.

```

Here we used `enumerate` to replace and convert each value of the list one by one. This is a powerful method unique to Python as loops are not computationally efficient in R. The code of example earlier is at <https://nbviewer.jupyter.org/gist/decisionstats/42b3fc90ae6fa537a19a08017e0336cb>

Now let us see how we would do this in R. The following is R code (also available at <http://rpubs.com/newajay/dataquality2>):

```

numlistr=c("$10000", "$20,000", "30,000", 40000, "50000  ")
mean(numlistr)

```

```

## Warning in mean.default(numlistr): argument is not
## numeric or logical:
## returning NA
## [1] NA

```

```

numlistr=gsub(",", "", numlistr)
numlistr

```

```

## [1] "$10000"    "$20000"    "30000"     "40000"
"50000"     "

```

```

numlistr=gsub("\\$", "", numlistr)
Numlistr

```

```
## [1] "10000"      "20000"      "30000"      "40000"
      "50000"      "
```

```
numlistr=as.numeric(numlistr)
numlistr
```

```
## [1] 10000 20000 30000 40000 50000
```

```
mean(numlistr)
## [1] 30000
```

For searching on character strings, we can use `re.search` and use `bool` to return True or False. The `bool` function returns True when the argument for which it is passed on is true; otherwise it returns false. The following code is also available at <https://nbviewer.jupyter.org/gist/decisionstats/612116b1b8147cfb3808f5ac3c791eba>

```
import re
```

```
names=["Ajay","V ijay","Ra jay ", " Jayesh"]
```

```
for name in names:
    print (re.search(r' (jay)',name))
```

```
<_sre.SRE_Match object; span=(1, 4), match='jay'>
<_sre.SRE_Match object; span=(3, 6), match='jay'>
<_sre.SRE_Match object; span=(3, 6), match='jay'>
None
```

Using `re.search` we got the positions of the string (jay).

```
for name in names:
    print (bool(re.search(r' (jay)',name)))
```

```
True
True
True
False
```

Using `bool` we got whether the string (jay) was present or not.

In R, we use `grep`, `grepl` functions for searching by string pattern. The following code is also available at <http://rpubs.com/newajay/grepinr>

```
names=c("Ajay","V ijay","Ra jay ", " Jayesh")
grepl("jay",names)
```

```
## [1] TRUE TRUE TRUE FALSE
gregexpr(pattern = 'jay', names)
## [[1]]
## [1] 2
## attr(,"match.length")
## [1] 3
## attr(,"useBytes")
## [1] TRUE
##
## [[2]]
## [1] 4
## attr(,"match.length")
## [1] 3
## attr(,"useBytes")
## [1] TRUE
##
## [[3]]
## [1] 4
## attr(,"match.length")
## [1] 3
## attr(,"useBytes")
## [1] TRUE
##
## [[4]]
## [1] -1
## attr(,"match.length")
## [1] -1
## attr(,"useBytes")
## [1] TRUE
grep("jay", names)
## [1] 1 2 3
grep("jay", names, value = T)
## [1] "Ajay" "V ijay" "Ra jay "
```

3.3 Data Inspection

We need to inspect data after import to see whether we correctly imported the right size as well as the format of data columns (from <http://nbviewer.jupyter.org/gist/decisionstats/43e332cdf2d5a7599f4e61205e8788f> and <http://nbviewer.jupyter.org/gist/decisionstats/c1684daaecf62dd4bf4>).

```
'''Lets get some information on the object.
In R we would get this by str command (for structure).
```

In Python `str` turns the object to string so we use `info`. This was a multiple-line comment using three single quote marks.'''

diamonds.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53940 entries, 0 to 53939
Data columns (total 11 columns):
Unnamed: 0      53940 non-null int64
carat           53940 non-null float64
cut             53940 non-null object
color           53940 non-null object
clarity         53940 non-null object
depth           53940 non-null float64
table           53940 non-null float64
price           53940 non-null int64
x               53940 non-null float64
y               53940 non-null float64
z               53940 non-null float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.3+ MB
```

`diamonds.head(10)` #we check the first 10 rows in the dataset

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
5	6	0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48
6	7	0.24	Very Good	I	VVS1	62.3	57	336	3.95	3.98	2.47
7	8	0.26	Very Good	H	SI1	61.9	55	337	4.07	4.11	2.53
8	9	0.22	Fair	E	VS2	65.1	61	337	3.87	3.78	2.49
9	10	0.23	Very Good	H	VS1	59.4	61	338	4.00	4.05	2.39

- To refer to a particular row in Python, I can use index.
- In R I refer to the object in *i*th row and *j*th column by `OBJECTNAME[i,j]`.
- In R I refer to the column name by `OBJECTNAME$ColumnName`.
- Note in Python Index starts with 0 while in R it starts with 1.

You can use the `info` command to look at imported objects.

Dropping variables is easily done by the `drop` command followed by column name.

Table 3.1 R and Python are quite easily comparable.

	R	Python (using pandas package*)
Getting the names of rows and columns of data frame “df”	rownames(df)	df.index
	<i>returns the name of the rows</i>	<i>returns the name of the rows</i>
	colnames(df)	df.columns
	<i>returns the name of the columns</i>	<i>returns the name of the columns</i>
Seeing the top and bottom “x” rows of the data frame “df”	head(df,x)	df.head(x)
	<i>returns top x rows of data frame</i>	<i>returns top x rows of data frame</i>
	tail(df,x)	df.tail(x)
	<i>returns bottom x rows of data frame</i>	<i>returns bottom x rows of data frame</i>
Getting dimensions of data frame “df”	dim(df)	df.shape
	<i>returns in this format: rows, columns</i>	<i>returns in this format: (rows, columns)</i>
Length of data frame “df”	length(df)	len(df)
	<i>returns no. of columns in data frames</i>	<i>returns no. of columns in data frames</i>

Using the head function allows you to look at the first few rows.

We get Table 3.1 from <http://www.slideshare.net/ajayohri/python-for-r-users> that has got 37 000 views till December 2016.

```
diamonds=diamonds.drop("Unnamed: 0",1)
```

```
diamonds.columns
```

```
Index(['carat', 'cut', 'color', 'clarity', 'depth',  
      'table', 'price', 'x', 'y',  
      'z'],  
      dtype='object')
```

```
diamonds.index
```

```
RangeIndex(start=0, stop=53940, step=1)
```

Sorting data is quite easy too

```
diamonds3=diamonds.sort(["price"])  
diamonds3.head()
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

3.3.1 Missing Value Treatment

Missing values can be dropped by dropna command. Python uses NaN to denote missing values, while R denotes them with NA. For more information on missing values, please do read the documentation at http://pandas.pydata.org/pandas-docs/stable/missing_data.html

```
diamonds4 = diamonds.dropna(how = 'any')
```

In R missing value treatment is done by the functions na.rm = T (which ignores NA or missing values) and na.omit (which deletes the missing values). Note in R missing values are denoted by NA. In addition—using gsub, one can replace a pattern by another (or delete it), and—using the as operator, one can convert data from one format to another. We use the is.na function to find if a value is a missing value (is.na = TRUE) or not.

For missing values and other issues, let us take this small caselet from <http://rpubs.com/ajaydecis/dataman3>. We have a small list of mixed formats of money and we need to find the mean money. We first use gsub to remove the comma, then gsub with an escape character \\ to remove the \$ sign, and then the as.numeric operator to make it numeric. Finally, we use the na.rm operator to find mean of non-missing values and use na.omit to remove missing values altogether.

```
money=c("$50000", "$50,000", "50,000", 50000, "50000", NA)
money
## [1] 50000 50000 50000 50000 50000      NA
mean(money)
## [1] NA
mean(money, na.rm=T)
## [1] 50000
money=na.omit(money)
money
```

```
## [1] 50000 50000 50000 50000 50000
## attr(,"na.action")
## [1] 6
## attr(,"class")
## [1] "omit"

mean(money)

## [1] 50000
```

3.4 Data Selection

To refer to data by row number, we can use the `.ix` command to refer to it by index value (in Python index starts from 0, while in R it starts from 1 for the first row).

Unlike R, there is no `$` command to select columns for a data frame (i.e., `diamonds$color`). So we can use the notation `dataframename.columnname` or `dataframename[["columnname"]]`. In python chaining the commands is easy by just adding a dot with command to previous object.

For choosing both row number and column name, I put these values within the double square brackets.

```
diamonds.ix[20:30]
```

	carat	cut	color	clarity	depth	table	price	x	y	z
20	0.30	Good	I	SI2	63.3	56.0	351	4.26	4.30	2.71
21	0.23	Very Good	E	VS2	63.8	55.0	352	3.85	3.92	2.48
22	0.23	Very Good	H	VS1	61.0	57.0	353	3.94	3.96	2.41
23	0.31	Very Good	J	SI1	59.4	62.0	353	4.39	4.43	2.62
24	0.31	Very Good	J	SI1	58.1	62.0	353	4.44	4.47	2.59
25	0.23	Very Good	G	VVS2	60.4	58.0	354	3.97	4.01	2.41
26	0.24	Premium	I	VS1	62.5	57.0	355	3.97	3.94	2.47
27	0.30	Very Good	J	VS2	62.2	57.0	357	4.28	4.30	2.67
28	0.23	Very Good	D	VS2	60.5	61.0	357	3.96	3.97	2.40
29	0.23	Very Good	F	VS1	60.9	57.0	357	3.96	3.99	2.42
30	0.23	Very Good	F	VS1	60.0	57.0	402	4.00	4.03	2.41

```
diamonds.ix[20:30].cut
20          Good
21    Very Good
```

```

22    Very Good
23    Very Good
24    Very Good
25    Very Good
26    Premium
27    Very Good
28    Very Good
29    Very Good
30    Very Good

```

Name: cut, dtype: object

diamonds.ix[20:30] ["color"]

```

20    I
21    E
22    H
23    J
24    J
25    G
26    I
27    J
28    D
29    F
30    F

```

Name: color, dtype: object

diamonds[["cut", "color", "clarity"]][20:30]

	cut	color	clarity
20	Good	I	SI2
21	Very Good	E	VS2
22	Very Good	H	VS1
23	Very Good	J	SI1
24	Very Good	J	SI1
25	Very Good	G	VVS2
26	Premium	I	VS1
27	Very Good	J	VS2
28	Very Good	D	VS2
29	Very Good	F	VS1

```
diamonds[["cut", "color", "clarity"]].head()
```

	cut	color	clarity
0	Ideal	E	SI2
1	Premium	E	SI1
2	Good	E	VS1
3	Premium	I	VS2
4	Good	J	SI2

```
diamonds.ix[20:30, ["cut", "color", "clarity"]]
```

	cut	color	clarity
20	Good	I	SI2
21	Very Good	E	VS2
22	Very Good	H	VS1
23	Very Good	J	SI1
24	Very Good	J	SI1
25	Very Good	G	VVS2
26	Premium	I	VS1
27	Very Good	J	VS2
28	Very Good	D	VS2
29	Very Good	F	VS1
30	Very Good	F	VS1

3.4.1 Random Selection of Data

Using the .ix method we can do random selection of a data frame that can be useful for large amounts of data:

```
import numpy as np

len(diamonds)

53940
0.0001*len(diamonds)

5.394
round(0.0001*len(diamonds))
```

```

5
diamonds.index.values

array([    0,     1,     2, ..., 53937, 53938, 53939])

rows=np.random.choice(diamonds.index.values,round
(0.0001*len(diamonds)))
print(rows)

[26766 43621 3614 35052 51042]

```

```
diamonds.ix[rows]
```

	carat	cut	color	clarity	depth	table	price	x	y	z
26766	2.45	Ideal	F	SI2	62.0	55.0	16589	8.67	8.64	5.36
43621	0.46	Premium	F	VS1	60.5	58.0	1432	5.02	4.97	3.02
3614	1.05	Ideal	I	VS2	62.2	56.0	3428	6.52	6.50	4.05
35052	0.31	Ideal	F	VVS1	61.8	56.0	884	4.33	4.37	2.69
51042	0.70	Good	H	VS2	64.2	58.0	2330	5.58	5.61	3.59

3.4.2 Conditional Selection

Let us try selecting data by conditions. We can again use the double square brackets.

We can use the query function for easier conditional selection and using multiple conditions including (&) or (|) operators. Note we use the parenthesis in query here, not the square brackets.

```
diamonds[diamonds['carat'] > 3.7]
```

	carat	cut	color	clarity	depth	table	price	x	y	z
25998	4.01	Premium	I	I1	61.0	61.0	15223	10.14	10.10	6.17
25999	4.01	Premium	J	I1	62.5	62.0	15223	10.02	9.94	6.24
26444	4.00	Very Good	I	I1	63.3	58.0	15984	10.01	9.94	6.31
27130	4.13	Fair	H	I1	64.8	61.0	17329	10.00	9.85	6.43
27415	5.01	Fair	J	I1	65.5	59.0	18018	10.74	10.54	6.98
27630	4.50	Fair	J	I1	65.8	58.0	18531	10.23	10.16	6.72

```
diamonds.query('carat >3.5 and color == "J"')
```

	carat	cut	color	clarity	depth	table	price	x	y	z
25999	4.01	Premium	J	I1	62.5	62.0	15223	10.02	9.94	6.24
27415	5.01	Fair	J	I1	65.5	59.0	18018	10.74	10.54	6.98
27630	4.50	Fair	J	I1	65.8	58.0	18531	10.23	10.16	6.72
27679	3.51	Premium	J	VS2	62.5	59.0	18701	9.66	9.63	6.03

We can also use pandasql package to use SQL to query data conditionally (from <https://nbviewer.jupyter.org/gist/decisionstats/284a86d0541d06489e92>). In R we use sqldf package for the same.

```
from pandasql import sqldf
pysqldf = lambda q: sqldf(q, globals())

import pandas as pd

mycars=pd.read_csv("http://vincentarelbundock.github.io/
Rdatasets/csv/datasets/mtcars.csv")

mycars.head()
```

	Unnamed: 0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2

```
mycars.columns
Index(['Unnamed: 0', 'mpg', 'cyl', 'disp', 'hp',
      'drat', 'wt', 'qsec', 'vs',
      'am', 'gear', 'carb'], dtype='object')
mycars.columns= ['brand','mpg', 'cyl', 'disp',
                 'hp', 'drat', 'wt', 'qsec', 'vs', 'am', 'gear',
                 'carb']
```

```
pysql>df("SELECT * FROM mycars LIMIT 10;")
```

	brand	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4

```
pysql>df("SELECT * FROM mycars WHERE gear > 3 ;")
```

	brand	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
4	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
5	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
6	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
7	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
8	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
9	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
10	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
11	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
12	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
13	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
14	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
15	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
16	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

```
pysqldf("SELECT * FROM mycars WHERE gear > 3 and carb > 4 ;")
```

	brand	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Ferrari Dino	19.7	6	145	175	3.62	2.77	15.5	0	1	5	6
1	Maserati Bora	15.0	8	301	335	3.54	3.57	14.6	0	1	5	8

```
pysqldf("SELECT AVG(mpg),gear FROM mycars GROUP by gear;")
```

	AVG(mpg)	gear
0	16.106667	3
1	24.533333	4
2	21.380000	5

```
pysqldf("SELECT AVG(mpg),gear,cyl FROM mycars GROUP by gear,cyl;")
```

	AVG(mpg)	gear	cyl
0	21.500	3	4
1	19.750	3	6
2	15.050	3	8
3	26.925	4	4
4	19.750	4	6
5	28.200	5	4
6	19.700	5	6
7	15.400	5	8

3.5 Data Inspection in R

R is quite simple on how we can inspect data. We can use head and tail to look at first few and last few records, and we can use str and names to look at structure and column names of a data frame. We can use the \$ notation to look at a particular column name and use the [] square bracket (row,column) notation to look at a particular value.

You can see the code at <http://rpubs.com/ajaydecis/mtcars1> and <http://rpubs.com/ajaydecis/mtcars> or at the following code to understand how easy it is. Conditional selection is thus quite easy in R. The data in I row and J column for DataFrameX is shown by DataFrameX[I,J] and alternatively the data in J column can be DataFrameX\$J_Column_Name or DataFrameX[,J].

The actual data mtcars can also be seen at <http://vincentarelbundock.github.io/Rdatasets/csv/datasets/mtcars.csv>

```
data("mtcars")
head(mtcars,10)
##      mpg   cyl  disp    hp  drat    wt    qsec    vs  am  gear  carb
## Mazda RX4      21.0   6  160.0  110   3.90  2.620  16.46   0   1    4    4
## Mazda RX4 Wag  21.0   6  160.0  110   3.90  2.875  17.02   0   1    4    4
## Datsun 710      22.8   4  108.0   93   3.85  2.320  18.61   1   1    4    1
## Hornet 4 Drive  21.4   6  258.0  110   3.08  3.215  19.44   1   0    3    1
## Hornet Sportabout 18.7   8  360.0  175   3.15  3.440  17.02   0   0    3    2
## Valiant        18.1   6  225.0  105   2.76  3.460  20.22   1   0    3    1
## Duster 360     14.3   8  360.0  245   3.21  3.570  15.84   0   0    3    4
## Merc 240D      24.4   4  146.7   62   3.69  3.190  20.00   1   0    4    2
## Merc 230       22.8   4  140.8   95   3.92  3.150  22.90   1   0    4    2
## Merc 280       19.2   6  167.6  123   3.92  3.440  18.30   1   0    4    4
tail(mtcars,5)
##      mpg   cyl  disp    hp  drat    wt    qsec    vs  am  gear  carb
## Lotus Europa   30.4   4   95.1  113   3.77  1.513  16.9    1   1    5    2
## Ford Pantera L  15.8   8  351.0  264   4.22  3.170  14.5    0   1    5    4
## Ferrari Dino    19.7   6  145.0  175   3.62  2.770  15.5    0   1    5    6
## Maserati Bora   15.0   8  301.0  335   3.54  3.570  14.6    0   1    5    8
## Volvo 142E      21.4   4  121.0  109   4.11  2.780  18.6    1   1    4    2
names(mtcars)
## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
## [11] "carb"
```

```

str(mtcars)
## 'data.frame': 32 obs. of 11 variables:
## $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num 160 160 108 258 360 ...
## $ hp : num 110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
## $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num 16.5 17 18.6 19.4 17 ...
## $ vs : num 0 0 1 1 0 1 0 1 1 1 ...
## $ am : num 1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num 4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num 4 4 1 1 2 1 4 2 2 4 ...

mtcars[1,]
## mpg cyl disp hp drat wt qsec vs am gear carb
## Mazda RX4 21 6 160 110 3.9 2.62 16.46 0 1 4 4

mtcars[,2]
##[1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 4 8 6 8 4
mtcars[2,3]
##[1] 160

mtcars$cyl
##[1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 4 8 6 8 4

```

Now for conditional selection we use the following

```

mtcars[2,3]
## [1] 160

```

```
mtcars[mtcars$cyl>4,]
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
## Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
## Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
## Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
## Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
## Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
## Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
## Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
## Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
## Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
## Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
## Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
## Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
## Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
## Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
## AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
## Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
## Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
## Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0		5	4
## Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
## Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8

Using attach function we no longer have to write mtcars\$ every time but can refer to the column name directly.

```
attach(mtcars)
mtcars[cyl>4,]
##          mpg   cyl  disp    hp  drat    wt    qsec    vs  am  gear  carb
## Mazda RX4      21.0   6  160.0  110   3.90  2.620  16.46   0   1     4     4
## Mazda RX4 Wag  21.0   6  160.0  110   3.90  2.875  17.02   0   1     4     4
## Hornet 4 Drive  21.4   6  258.0  110   3.08  3.215  19.44   1   0     3     1
## Hornet Sportabout 18.7   8  360.0  175   3.15  3.440  17.02   0   0     3     2
## Valiant        18.1   6  225.0  105   2.76  3.460  20.22   1   0     3     1
## Duster 360     14.3   8  360.0  245   3.21  3.570  15.84   0   0     3     4
## Merc 280       19.2   6  167.6  123   3.92  3.440  18.30   1   0     4     4
## Merc 280C      17.8   6  167.6  123   3.92  3.440  18.90   1   0     4     4
## Merc 450SE     16.4   8  275.8  180   3.07  4.070  17.40   0   0     3     3
## Merc 450SL     17.3   8  275.8  180   3.07  3.730  17.60   0   0     3     3
## Merc 450SLC    15.2   8  275.8  180   3.07  3.780  18.00   0   0     3     3
## Cadillac Fleetwood 10.4   8  472.0  205   2.93  5.250  17.98   0   0     3     4
## Lincoln Continental 10.4   8  460.0  215   3.00  5.424  17.82   0   0     3     4
## Chrysler Imperial 14.7   8  440.0  230   3.23  5.345  17.42   0   0     3     4
## Dodge Challenger 15.5   8  318.0  150   2.76  3.520  16.87   0   0     3     2
## AMC Javelin    15.2   8  304.0  150   3.15  3.435  17.30   0   0     3     2
## Camaro Z28     13.3   8  350.0  245   3.73  3.840  15.41   0   0     3     4
## Pontiac Firebird 19.2   8  400.0  175   3.08  3.845  17.05   0   0     3     2
## Ford Pantera L  15.8   8  351.0  264   4.22  3.170  14.50   0   1     5     4
## Ferrari Dino   19.7   6  145.0  175   3.62  2.770  15.50   0   1     5     6
## Maserati Bora  15.0   8  301.0  335   3.54  3.570  14.60   0   1     5     8
```

```
mtcars[cyl>4 &gear >4,] #AND &
##      mpg   cyl  disp    hp  drat    wt    qsec    vs  am  gear  carb
## Ford Pantera L   15.8   8   351   264  4.22  3.17   14.5    0   1     5     4
## Ferrari Dino     19.7   6   145   175  3.62  2.77   15.5    0   1     5     6
## Maserati Bora     15.0   8   301   335  3.54  3.57   14.6    0   1     5     8

mtcars[cyl>4 &gear ==4,] # EQUALITY ==
##      mpg   cyl  disp    hp  drat    wt    qsec    vs  am  gear  carb
## Mazda RX4       21.0   6   160.0  110  3.90  2.620  16.46    0   1     4     4
## Mazda RX4 Wag   21.0   6   160.0  110  3.90  2.875  17.02    0   1     4     4
## Merc 280         19.2   6   167.6  123  3.92  3.440  18.30    1   0     4     4
## Merc 280C        17.8   6   167.6  123  3.92  3.440  18.90    1   0     4     4

mtcars[cyl>4 |gear ==4,] #OR |
##      mpg   cyl  disp    hp  drat    wt    qsec    vs  am  gear  carb
## Mazda RX4       21.0   6   160.0  110  3.90  2.620  16.46    0   1     4     4
## Mazda RX4 Wag   21.0   6   160.0  110  3.90  2.875  17.02    0   1     4     4
## Datsun 710       22.8   4   108.0   93  3.85  2.320  18.61    1   1     4     1
## Hornet 4 Drive   21.4   6   258.0  110  3.08  3.215  19.44    1   0     3     1
## Hornet Sportabout 18.7   8   360.0  175  3.15  3.440  17.02    0   0     3     2
## Valiant          18.1   6   225.0  105  2.76  3.460  20.22    1   0     3     1
## Duster 360       14.3   8   360.0  245  3.21  3.570  15.84    0   0     3     4
## Merc 240D        24.4   4   146.7   62  3.69  3.190  20.00    1   0     4     2
## Merc 230         22.8   4   140.8   95  3.92  3.150  22.90    1   0     4     2
## Merc 280         19.2   6   167.6  123  3.92  3.440  18.30    1   0     4     4
## Merc 280C        17.8   6   167.6  123  3.92  3.440  18.90    1   0     4     4
## Merc 450SE       16.4   8   275.8  180  3.07  4.070  17.40    0   0     3     3
```

```

## Merc 450SL      17.3  8   275.8 180  3.07  3.730 17.60  0  0   3   3
## Merc 450SLC     15.2  8   275.8 180  3.07  3.780 18.00  0  0   3   3
## Cadillac Fleetwood 10.4  8   472.0 205  2.93  5.250 17.98  0  0   3   4
## Lincoln Continental 10.4  8   460.0 215  3.00  5.424 17.82  0  0   3   4
## Chrysler Imperial 14.7  8   440.0 230  3.23  5.345 17.42  0  0   3   4
## Fiat 128        32.4  4    78.7  66  4.08  2.200 19.47  1  1   4   1
## Honda Civic     30.4  4    75.7  52  4.93  1.615 18.52  1  1   4   2
## Toyota Corolla  33.9  4    71.1  65  4.22  1.835 19.90  1  1   4   1
## Dodge Challenger 15.5  8   318.0 150  2.76  3.520 16.87  0  0   3   2
## AMC Javelin     15.2  8   304.0 150  3.15  3.435 17.30  0  0   3   2
## Camaro Z28      13.3  8   350.0 245  3.73  3.840 15.41  0  0   3   4
## Pontiac Firebird 19.2  8   400.0 175  3.08  3.845 17.05  0  0   3   2
## Fiat X1-9       27.3  4    79.0  66  4.08  1.935 18.90  1  1   4   1
## Ford Pantera L  15.8  8   351.0 264  4.22  3.170 14.50  0  1   5   4
## Ferrari Dino    19.7  6   145.0 175  3.62  2.770 15.50  0  1   5   6
## Maserati Bora   15.0  8   301.0 335  3.54  3.570 14.60  0  1   5   8
## Volvo 142E     21.4  4   121.0 109  4.11  2.780 18.60  1  1   4   2

mtcars[cyl>4 &gear !=4,] #NOT !=
##      mpg  cyl  disp  hp  drat   wt   qsec  vs  am  gear  carb
## Hornet 4 Drive  21.4   6   258.0 110  3.08  3.215 19.44  1   0    3    1
## Hornet Sportabout 18.7   8   360.0 175  3.15  3.440 17.02  0   0    3    2
## Valiant        18.1   6   225.0 105  2.76  3.460 20.22  1   0    3    1
## Duster 360     14.3   8   360.0 245  3.21  3.570 15.84  0   0    3    4
## Merc 450SE     16.4   8   275.8 180  3.07  4.070 17.40  0   0    3    3
## Merc 450SL     17.3   8   275.8 180  3.07  3.730 17.60  0   0    3    3
## Merc 450SLC    15.2   8   275.8 180  3.07  3.780 18.00  0   0    3    3

```

## Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
## Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
## Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
## Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
## AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
## Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
## Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
## Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
## Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
## Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8

We can use the sqldf package to use SQL to query data in R.

<http://rpubs.com/ajaydecis/dataman>

```
data("mtcars")
library(sqldf)
head(mtcars)
```

##	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
## Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
## Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
## Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
## Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
## Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

#Give us average mpg for every carb and every cyl

```

sqldf("select avg(mpg)  from mtcars ")

##      avg(mpg)
## 1 20.09062

sqldf("select avg(mpg),cyl from mtcars group by cyl")

##      avg(mpg)  cyl
## 1 26.66364    4
## 2 19.74286    6
## 3 15.10000    8

sqldf("select avg(mpg),cyl,gear from mtcars group by cyl,gear")

##      avg(mpg)  cyl  gear
## 1 21.500    4    3
## 2 26.925    4    4
## 3 28.200    4    5
## 4 19.750    6    3
## 5 19.750    6    4
## 6 19.700    6    5
## 7 15.050    8    3
## 8 15.400    8    5

```

3.5.1 Diamond Dataset from ggplot2 Package in R

Let us do some more data munging on the diamonds dataset in R (see <http://rpubs.com/ajaydecis/basicR>). We see random selection, multiple conditional selection, and other ways in R to manipulate data.

ls() lists all objects in Memory. rm("objectname") removes a particular object, while rm(list=ls()) removes all objects. gc() does garbage collection to free up memory particularly if a large object has been deleted.

```

ls()
## character(0)
rm(list=ls())
gc()

##          used (Mb) gc trigger (Mb) max used (Mb)
## Ncells 291320   7.8      592000 15.9    391619 10.5
## Vcells 333507   2.6      786432  6.0    692009  5.3

#memory.size() windows specific-this gives memory occupied
#memory.limit() windows specific-this gives total memory available
# install.packages(ggplot2)
library(ggplot2)
data(diamonds)
names(diamonds)

##[1] "carat"    "cut"       "color"     "clarity"  "depth"    "table"    "price"
##[8] "x"        "y"        "z"

class(diamonds) #What type of object is this?
##[1] "data.frame"

dim(diamonds) #Dimensions - rows and columns
##[1] 53940      10

nrow(diamonds) #Number of Rows
##[1] 53940

```

```
ncol(diamonds) #Number of Columns
## [1] 10

str(diamonds) #Structure - same as info in Python

## 'data.frame':    53940 obs. of  10 variables:
## $ carat   : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
## $ cut     : Ord.factor w/ 5 levels "Fair"<"Good"<...: 5 4 2 4 2 3 3 1 3 ...
## $ color   : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<...: 2 2 2 6 7 7 6 5 2 5 ...
## $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<...: 2 3 5 4 2 6 7 3 4 5 ...
## $ depth   : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
## $ table   : num  55 61 65 58 58 57 57 55 61 61 ...
## $ price   : int  326 326 327 334 335 336 336 337 337 338 ...
## $ x       : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
## $ y       : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
## $ z       : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...

#data inspection
head(diamonds)

##      carat   cut      color clarity  depth  table  price    x    y    z
## 1  0.23    Ideal      E      SI2    61.5    55    326   3.95  3.98  2.43
## 2  0.21   Premium      E      SI1    59.8    61    326   3.89  3.84  2.31
## 3  0.23     Good      E     VS1    56.9    65    327   4.05  4.07  2.31
## 4  0.29   Premium      I     VS2    62.4    58    334   4.20  4.23  2.63
## 5  0.31     Good      J      SI2    63.3    58    335   4.34  4.35  2.75
## 6  0.24   Very Good  J     VVS2    62.8    57    336   3.94  3.96  2.48
head(diamonds$carat) #Using the $operator in take in one column
```

[1] 0.23 0.21 0.23 0.29 0.31 0.24

Diamonds[3,] #looking at the third row

##	carat	cut	color	clarity	depth	table	price	x	y	z
## 3	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31

head(diamonds[,3],10) #Taking first 10 values of 3rd column

[1] E E E I J J I H E H
Levels: D < E < F < G < H < I < J

tail(diamonds) #Last six values by default

##		carat	cut	color	clarity	depth	table	price	x	y	z
##	53935	0.72	Premium	D	SI1	62.7	59	2757	5.69	5.73	3.58
##	53936	0.72	Ideal	D	SI1	60.8	57	2757	5.75	5.76	3.50
##	53937	0.72	Good	D	SI1	63.1	55	2757	5.69	5.75	3.61
##	53938	0.70	Very Good	D	SI1	62.8	60	2757	5.66	5.68	3.56
##	53939	0.86	Premium	H	SI2	61.0	58	2757	6.15	6.12	3.74
##	53940	0.75	Ideal	D	SI2	62.2	55	2757	5.83	5.87	3.64

#missing value treatment

head(na.omit(diamonds))

##	carat	cut	color	clarity	depth	table	price	x	y	z
## 1	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
## 2	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
## 3	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31

```
## 4  0.29  Premium    I        VS2        62.4    58        334    4.20    4.23    2.63
## 5  0.31   Good      J        SI2        63.3    58        335    4.34    4.35    2.75
## 6  0.24  Very Good J        VVS2        62.8    57        336    3.94    3.96    2.48
```

```
head(mean(diamonds$price,na.rm=T))
```

```
## [1] 3932.8
```

```
head(is.na(diamonds$price))
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
naa=is.na(diamonds$price)
```

```
table(naa) #table gives frequency values and can be used for data inspection
```

```
## naa
```

```
## FALSE
```

```
## 53940
```

```
#random sample
```

```
sample(10,3,T) #This is random sample similar to numpy example earlier
```

```
## [1] 10  2  3
```

```
sample(10,5,F) #Out of ten numbers choose 5 values, with substitution =False
```

```
## [1]  4  6 10  9  7
```

```

rnorm(10,5,9)
#Random numbers by normal distribution. 10 numbers with mean 5
#and standard deviation 9
## [1] -5.323364 10.778377 9.562595 25.202856 14.740653 6.146948 4.712989
## [8] 10.033625 13.576019 -2.904901

sample(53940,54,F) #Choosing Random Row Numbers

## [1] 16015 3295 30810 6128 13020 17596 26258 13724 18290 49344 28803
## [12] 19296 30969 18564 47637 19556 36676 31809 27470 40427 13827 19285
## [23] 34027 19775 3914 42903 38505 22783 22571 24783 53796 49667 35219
## [34] 43201 41791 47455 22991 900 32144 27631 28891 40676 12270 9432
## [45] 25694 4886 20734 8846 28651 6460 33818 7642 21005 15168

sample(nrow(diamonds),0.001*(nrow(diamonds)),F) #0.1 % Random Sample

## [1] 40131 24733 29152 4190 4079 31014 23678 20268 36029 37565 26792
## [12] 52564 31527 22044 4255 39092 40824 35166 4566 20044 8307 21015
## [23] 42075 36046 41057 20671 28080 5624 31169 49728 48181 17372 26373
## [34] 11403 37404 8279 25680 15130 23026 43130 43979 10054 43876 16751
## [45] 8193 25554 42141 3124 29700 45469 53186 25642 33776

a=nrow(diamonds)
sample(a,0.0001*a,F) #To explain the code above

## [1] 40676 32758 43476 47130 38664
randomrows=sample(a,0.0001*a,F)
Diamonds[randomrows,]

```

```
##          carat cut      color clarity depth table price  x    y    z
## 29067    0.40  Good      F      SI1   63.1   58    687 4.66 4.69 2.95
## 28304    0.32  Very Good I      SI1   62.8   58    432 4.34 4.39 2.74
## 25301    1.58  Very Good G      VS1   62.8   57   13963 7.34 7.40 4.63
## 15670    1.00  Fair      E      VS2   57.3   64   6285 6.59 6.46 3.79
## 5267     1.03  Good      J      VS2   63.7   56   3795 6.42 6.35 4.07
```

```
cut2=diamonds[diamonds$cut=="Ideal",]
head(cut2)
```

```
##      carat cut  color clarity depth table price  x    y    z
## 1    0.23  Ideal  E      SI2   61.5   55    326 3.95 3.98 2.43
## 12   0.23  Ideal  J      VS1   62.8   56    340 3.93 3.90 2.46
## 14   0.31  Ideal  J      SI2   62.2   54    344 4.35 4.37 2.71
## 17   0.30  Ideal  I      SI2   62.0   54    348 4.31 4.34 2.68
## 40   0.33  Ideal  I      SI2   61.8   55    403 4.49 4.51 2.78
## 41   0.33  Ideal  I      SI2   61.2   56    403 4.49 4.50 2.75
```

```
cut3=diamonds[diamonds$cut=="Ideal" & diamonds$color=="D",]
head(cut3)
```

```
##      carat cut  color clarity depth table price  x    y    z
## 63    0.30  Ideal  D      SI1   62.5   57    552 4.29 4.32 2.69
## 64    0.30  Ideal  D      SI1   62.1   56    552 4.30 4.33 2.68
## 121   0.71  Ideal  D      SI2   62.3   56   2762 5.73 5.69 3.56
## 133   0.71  Ideal  D      SI1   61.9   59   2764 5.69 5.72 3.53
## 145   0.71  Ideal  D      SI2   61.6   55   2767 5.74 5.76 3.54
## 156   0.76  Ideal  D      SI2   62.4   57   2770 5.78 5.83 3.62
```

```
cut4=diamonds[diamonds$cut=="Ideal" | diamonds$color=="D",]  
head(cut4)
```

##	carat	cut	color	clarity	depth	table	price	x	y	z
## 1	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
## 12	0.23	Ideal	J	VS1	62.8	56	340	3.93	3.90	2.46
## 14	0.31	Ideal	J	SI2	62.2	54	344	4.35	4.37	2.71
## 17	0.30	Ideal	I	SI2	62.0	54	348	4.31	4.34	2.68
## 29	0.23	Very Good	D	VS2	60.5	61	357	3.96	3.97	2.40
## 35	0.23	Very Good	D	VS1	61.9	58	402	3.92	3.96	2.44

```
cut5=ifelse(diamonds$price>9000,"Expensive","Not So Expensive")  
table(cut5)  
## cut5  
## Expensive Not So Expensive  
## 6298 47642
```

If else helps with conditional variable creation. Here for condition 1 (price>9000), if true, the value is second parameter (Expensive), or else third parameter (Not Expensive). Then we do a table (frequency analysis to find the values).

3.5.2 Modifying Date Formats and Strings in R

The following code will help us see how we modify date formats easily (using strptime function and lubridate packages) for date formats and use nchar and substr functions on character data.

```
#CHARACTER TO DATES
```

```
dobofclass=c("1April2007",  
             "28th july 1984",  
             "05 May 1988",  
             "29nov-2008")
```

```
strptime("29nov-2008", "%d%b-%Y")
```

```
## [1] "2008-11-29 IST"
```

```
strptime("05 May 1988", "%d%b-%Y")
```

```
## [1] NA
```

```
strptime("05 May 1988", "%d %B %Y") #Strptime needs exact format
```

```
## [1] "1988-05-05 IST"
```

```
library(lubridate) #lubridate is better and easier in guessing date format
```

```
dmy(dobofclass)
```

```
## [1] "2007-04-01 UTC" "1984-07-28 UTC" "1988-05-05 UTC" "2008-11-29 UTC"
```

```
Sys.Date()
```

```
## [1] "2015-12-12"
```

Differences in dates is given by difftime

```
difftime(Sys.Date(), dmy(dobofclass))
```

```
## Time differences in days
```

```
## [1] 3177 11459 10082 2569
```

?strptime will give you an insight on all the date formats.

Converting to Character

```
x=c(23,56,78,89)
as.character(x)
## [1] "23" "56" "78" "89"
```

```
paste(x)
## [1] "23" "56" "78" "89"
```

In R, modifying format is as simple as using the as operator. For character variables we can also just use paste.

```
paste("ajay",dobofclass[1])
```

```
## [1] "ajay 1April2007"
```

```
paste("student1",dobofclass[2])
```

```
## [1] "student1 28th july 1984"
```

Substr is a command that helps extract part of the string. Here the first value (2) is the beginning of the substring, while the second value (3) is the ending part of it. The command thus tells to begin from 2 character of ajay and end at the third value (included).

```
substr("ajay",2,3)
```

```
## [1] "ja"
```

Let us create a list

```
namclass=c("Ajay","Ajith","Sudeeptha","Yogisha")
```

Let us take first initial of every member of nclass, that is, A,A,S,Y.

```
substr(namclass,1,1)
## [1] "A" "A" "S" "Y"
```

The number of characters in a string is given by nchar:

```
nchar(namclass)
## [1] 4 5 9 7
```

To get the last character of every member of nclass list,

```
substr(namclass,nchar(namclass),nchar(namclass))
## [1] "y" "h" "a" "a"
```

3.5.3 Managing Strings in R

Let us take a small list. We use the c operator to make a list in R. We use grepl to find out if a certain pattern is present (here “jay”). We use ifelse for a conditional substitute.

Ifelse works like this in R, if the condition (first input **grepl(“jay”;names)**) is satisfied, it will replace it by (second input **“Yay its Jay”**), or else it would replace by (third input **“Oh no where is Jay”**).

```
names=c("Ajay","Vijay","Rajay","Jayesh")
grepl("jay",names)
## [1] TRUE TRUE TRUE FALSE
ifelse(grepl("jay",names),"Yay its Jay", " Oh no where is Jay")
## [1] "Yay its Jay" "Yay its Jay" "Yay its Jay"
## [4] " Oh no where is Jay"
```

We can also use stringr package to manage strings:

```
library(stringr)

str_dup(names,3)
## [1] "AjayAjayAjay"      "VijayVijayVijay"    "RajayRajayRajay"
## [4] "JayeshJayeshJayesh"

namq=c("Ajay  ", "Vijay  ", "Rajay", "Jay  esh  ")

str_trim(namq)
## [1] "Ajay"      "Vijay"     "Rajay"     "Jay  esh"

str_pad(namq,width=20,side="left")
## [1] "          Ajay  " "          Vijay  " "          Rajay" "          Jay  esh  "
## [4] "          Jay  esh  " "          Rajay"
```

Bibliography

Hadley Wickham (2015). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.0.0. <https://CRAN.R-project.org/package=stringr> (accessed May 2, 2017).

Hadley Wickham (2016). *tidyr: Easily Tidy Data with “spread()” and “gather()” Functions*. R package version 0.6.0. <https://CRAN.R-project.org/package=tidyr> (accessed May 2, 2017).

4

Exploratory Data Analysis

- Definition of EDA
- Box Plot and Five Numbers

4.1 Group by Analysis

When a numeric quantity is summarized across various levels of a factor or categorical variable, that is known as a group by Analysis Numerical.

Summaries of numerical variables can be done by describe, group by commands.

Categorical

Summaries are best done by cross tab, group by operations.

Datetime

Datetime data is best handled by datetime library.

4.2 Numerical Data

Let's take some data in <http://nbviewer.jupyter.org/gist/decisionstats/4142e98375445c5e4174> (Figure 4.1).

For numerical data **Describe** command in pandas acts the same as summary command in R for numerical data. Describe in Python Pandas gives you count, mean std min 25% 50% 75% max. **Summary** in R gives you mean, median, 25th and 75th quartiles, min, max.

There is another function in R called **fivenum**, and it gives you Tukey's five numbers for exploratory data analysis (min, lower-hinge, median, upper-hinge, max).

R has a better function in the Hmisc package called **describe** (yes it can be confusing to go back and forth between pandas and R). Hmisc::Describe

```
In [17]: adult.describe() #numerical summaries
```

```
Out[17]:
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	189778.366512	10.080679	1077.648844	87.303830	40.437456
std	13.640433	105549.977697	2.572720	7385.292085	402.960219	12.347429
min	17.000000	12285.000000	1.000000	0.000000	0.000000	1.000000
25%	28.000000	117827.000000	9.000000	0.000000	0.000000	40.000000
50%	37.000000	178356.000000	10.000000	0.000000	0.000000	40.000000
75%	48.000000	237051.000000	12.000000	0.000000	0.000000	45.000000
max	90.000000	1484705.000000	16.000000	99999.000000	4356.000000	59.000000

Figure 4.1 Describe function.

```
adult.quantile([.1,.5])
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
0.1	22.0	65716.0	7.0	0.0	0.0	24.0
0.5	37.0	178356.0	10.0	0.0	0.0	40.0

```
adult.quantile([.1,.5,.10,.25,.50,.75,.90,.95,.99])
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
0.10	22.0	65716.0	7.0	0.0	0.0	24.0
0.50	37.0	178356.0	10.0	0.0	0.0	40.0
0.10	22.0	65716.0	7.0	0.0	0.0	24.0
0.25	28.0	117827.0	9.0	0.0	0.0	40.0
0.50	37.0	178356.0	10.0	0.0	0.0	40.0
0.75	48.0	237051.0	12.0	0.0	0.0	45.0
0.90	58.0	329054.0	13.0	0.0	0.0	55.0
0.95	63.0	379682.0	14.0	5013.0	0.0	60.0
0.99	74.0	510072.0	16.0	15024.0	1980.0	80.0

Figure 4.2 Quantile function for percentiles and quartiles.

gives you a more elaborate numerical exploration (n,missing unique, Mean, .05,.10,.25,.50,.75,.90,.95 and 5 lowest and 5 highest scores). In Python we can do it using quantiles for percentiles (Figure 4.2).

We can look at correlations between numerical data using `corr` function (Figure 4.3).

```
adult.corr(method='pearson', min_periods=1)
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
age	1.000000	-0.076646	0.036527	0.077674	0.057775	0.068756
fnlwgt	-0.076646	1.000000	-0.043195	0.000432	-0.010252	-0.018768
education-num	0.036527	-0.043195	1.000000	0.122630	0.079923	0.148123
capital-gain	0.077674	0.000432	0.122630	1.000000	-0.031615	0.078409
capital-loss	0.057775	-0.010252	0.079923	-0.031615	1.000000	0.054256
hours-per-week	0.068756	-0.018768	0.148123	0.078409	0.054256	1.000000

Figure 4.3 `Corr` function for correlation.

4.3 Categorical Data

In Python `value_counts()` acts the same way as `table()` does in R for frequency tabulations (Figure 4.4).

```
: adult.race.value_counts()
:
:  White          27816
:   Black          3124
: Asian-Pac-Islander  1039
: Amer-Indian-Eskimo   311
:   Other           271
: Name: race, dtype: int64

: adult.sex.value_counts()
:
:  Male      21790
:  Female   10771
: Name: sex, dtype: int64
```

Figure 4.4 Frequency tabulation using `value_counts` function.

A cross tabulation between two variables in pandas is given by `crosstab`, while in R you can just do `table(var1,var2)` (Figure 4.5).

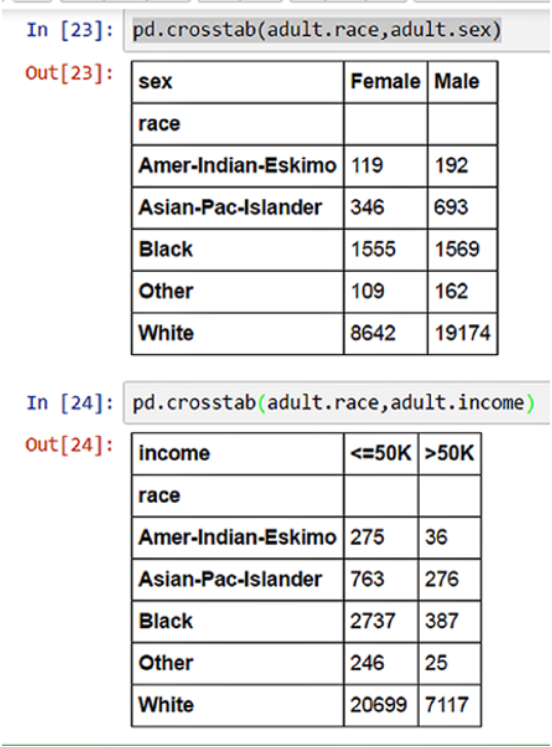


Figure 4.5 Cross tabulation using Cross Tab function.

Group by operations is best done by `groupby()` and then a numerical function applied to it (Figures 4.6 and 4.7).

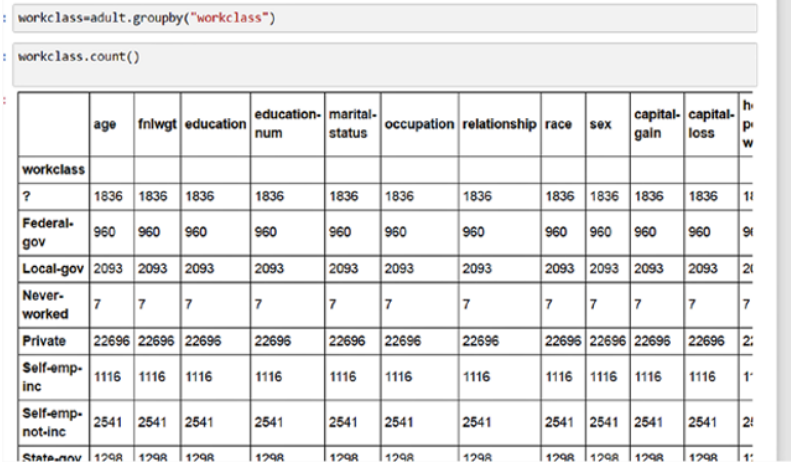


Figure 4.6 Grouping by a variable using `groupby`.

workclass.mean()

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
workclass						
?	40.960240	188516.338235	9.260349	606.795752	60.760349	31.919390
Federal-gov	42.590625	185221.243750	10.973958	833.232292	112.268750	41.379167
Local-gov	41.751075	188639.712852	11.042045	880.202580	109.854276	40.982800
Never-worked	20.571429	225989.571429	7.428571	0.000000	0.000000	28.428571
Private	36.797585	192764.114734	9.879714	889.217792	80.008724	40.267096
Self-emp-inc	46.017025	175981.344086	11.137097	4875.693548	155.138889	48.818100
Self-emp-not-inc	44.969697	175608.641480	10.226289	1886.061787	116.631641	44.421881
State-gov	39.436055	184136.613251	11.375963	701.699538	83.256549	39.031587
Without-pay	47.785714	174267.500000	9.071429	487.857143	0.000000	32.714286

Figure 4.7 Calculating mean (or a summary function) of Group by.

Note: To transpose the data from columns to rows and vice versa, we can use the transpose function (Figure 4.8).

File Edit View Insert Cell Kernel Widgets Help | Python [conda root]

```
In [28]: adult.transpose()
Out[28]:
```

	0	1	2	3	4	5	6	7	8	9	...	32551	32552	32553
age	39	50	38	53	28	37	49	52	31	42	...	32	43	32
workclass	State-gov	Self-emp-not-inc	Private	Private	Private	Private	Private	Self-emp-not-inc	Private	Private	...	Private	Private	Private
fnlwgt	77516	83311	215646	234721	338409	284582	160187	209642	45781	159449	...	34066	84651	1161
education	Bachelors	Bachelors	HS-grad	11th	Bachelors	Masters	9th	HS-grad	Masters	Bachelors	...	10th	Assoc-voc	Mas
education-num	13	13	9	7	13	14	5	9	14	13	...	6	11	14
marital-status	Never-married	Married-civ-spouse	Divorced	Married-civ-spouse	Married-civ-spouse	Married-civ-spouse	Married-civ-spouse	Married-civ-spouse	Never-married	Married-civ-spouse	...	Married-civ-spouse	Married-civ-spouse	Never-married
occupation	Adm-clerical	Exec-managerial	Handlers-cleaners	Handlers-cleaners	Prof-specialty	Exec-managerial	Other-service	Exec-managerial	Prof-specialty	Exec-managerial	...	Handlers-cleaners	Sales	Tech-support
relationship	Not-in-family	Husband	Not-in-family	Husband	Wife	Wife	Not-in-family	Husband	Not-in-family	Husband	...	Husband	Husband	Not-in-family
race	White	White	White	Black	Black	White	Black	White	White	White	...	Amer-Indian-Eskimo	White	Asia-Pac-Islander
sex	Male	Male	Male	Male	Female	Female	Female	Male	Female	Male	...	Male	Male	Male
capital-gain	2174	0	0	0	0	0	0	0	14084	5178	...	0	0	0
capital-loss	0	0	0	0	0	0	0	0	0	0	...	0	0	0
hours-per-week	40	13	40	40	40	40	16	45	50	40	...	40	45	11
native-country	United-States	United-States	United-States	United-States	Cuba	United-States	Jamaica	United-States	United-States	United-States	...	United-States	United-States	Taiwan
income	<=50K	<=50K	<=50K	<=50K	<=50K	<=50K	<=50K	<=50K	>50K	>50K	...	<=50K	<=50K	<=50K

15 rows x 32561 columns

Figure 4.8 Transpose function.

You can use the pivot command to present data in a pivot table format (Figure 4.9).

The above shows median age for different sex and races.

```
In [29]: e=adult.groupby(['sex', "race"]).age.median().reset_index()
e.pivot(index='sex', columns='race', values='age')
```

Out[29]:

race	Amer-Indian-Eskimo	Asian-Pac-Islander	Black	Other	White
sex					
Female	36	33	37	29	35
Male	35	37	36	32	38

Figure 4.9 Pivot function.

We can use the pandasql package and use SQL syntax to do selection as well as groupby operations on data. You can see this from <https://nbviewer.jupyter.org/gist/decisionstats/284a86d0541d06489e92> (or all the code from <https://github.com/decisionstats/pythonfordatascience>).

The SQL syntax makes it easy for existing SQL users to quickly manipulate and select data in a pandas DataFrame. Note this functionality is available in R in the sqldf package (Figure 4.10).

From <http://rpubs.com/ajaydecis/basicR> let's look at the way to explore data in R in multiple ways using summary, table functions, and Hmisc, and other packages. ls()

```
] : mycars.columns
]: Index(['Unnamed: 0', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs',
        'am', 'gear', 'carb'],
        dtype='object')

]: mycars.columns= ['brand', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs',
                  'am', 'gear', 'carb']

]: pysqldf("SELECT * FROM mycars LIMIT 10;")

]:
```

	brand	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
5	Valliant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4

Figure 4.10 Using SQL in Python using PandaSQL package.

```

## character(0)
rm(list=ls())
gc()
##          used (Mb) gc trigger (Mb) max used (Mb)
## Ncells 291320  7.8      592000 15.9      391619 10.5
## Vcells 333507  2.6      786432  6.0      692009  5.3
#memory.size() windows specific
#memory.limit() windows specific
# install.packages(ggplot2)
library(ggplot2)
data(diamonds)
names(diamonds)
## [1] "carat"  "cut"    "color"  "clarity" "depth"  "table"  "price"
## [8] "x"      "y"      "z"
class(diamonds)
## [1] "data.frame"
dim(diamonds)
## [1] 53940 10
nrow(diamonds)
## [1] 53940
ncol(diamonds)
## [1] 10
str(diamonds)
## 'data.frame':  53940 obs. of  10 variables:
## $ carat   : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
## $ cut     : Ord.factor w/ 5 levels "Fair"<"Good"<...: 5 4 2 4 2 3 3 3 1 3 ...
## $ color   : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<...: 2 2 2 6 7 7 6 5 2 5 ...
## $ clarity : Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<...: 2 3 5 4 2 6 7 3 4 5 ...

```

```
## $ depth : num 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
## $ table : num 55 61 65 58 58 57 57 55 61 61 ...
## $ price : int 326 326 327 334 335 336 336 337 337 338 ...
## $ x : num 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
## $ y : num 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
## $ z : num 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
#data inspection
head(diamonds)
##      carat cut      color clarity depth table price x      y      z
## 1 0.23 Ideal      E      SI2    61.5  55    326  3.95  3.98  2.43
## 2 0.21 Premium    E      SI1    59.8  61    326  3.89  3.84  2.31
## 3 0.23 Good      E      VS1    56.9  65    327  4.05  4.07  2.31
## 4 0.29 Premium    I      VS2    62.4  58    334  4.20  4.23  2.63
## 5 0.31 Good      J      SI2    63.3  58    335  4.34  4.35  2.75
## 6 0.24 Very Good J      VVS2    62.8  57    336  3.94  3.96  2.48
head(diamonds$carat)
## [1] 0.23 0.21 0.23 0.29 0.31 0.24
diamonds[3,]
##      carat cut      color clarity depth table price x      y      z
## 3 0.23 Good      E      VS1    56.9  65    327  4.05  4.07  2.31
head(diamonds[,3],10)
##      [1] E E E I J J I H E H
## Levels: D < E < F < G < H < I < J
tail(diamonds)
##      carat cut      color clarity depth table price x      y      z
## 53935 0.72 Premium    D      SI1    62.7  59    2757  5.69  5.73  3.58
## 53936 0.72 Ideal      D      SI1    60.8  57    2757  5.75  5.76  3.50
```

```

## 53937 0.72    Good      D      SI1      63.1    55      2757    5.69    5.75    3.61
## 53938 0.70   Very Good D      SI1      62.8    60      2757    5.66    5.68    3.56
## 53939 0.86   Premium   H      SI2      61.0    58      2757    6.15    6.12    3.74
## 53940 0.75   Ideal     D      SI2      62.2    55      2757    5.83    5.87    3.64
#missing value treatment
head(na.omit(diamonds))
##      carat cut      color clarity depth table price x      y      z
## 1  0.23   Ideal      E      SI2    61.5   55    326   3.95  3.98  2.43
## 2  0.21   Premium    E      SI1    59.8   61    326   3.89  3.84  2.31
## 3  0.23    Good      E      VS1    56.9   65    327   4.05  4.07  2.31
## 4  0.29   Premium    I      VS2    62.4   58    334   4.20  4.23  2.63
## 5  0.31    Good      J      SI2    63.3   58    335   4.34  4.35  2.75
## 6  0.24   Very Good J      VVS2    62.8   57    336   3.94  3.96  2.48
head(mean(diamonds$price,na.rm=T))
## [1] 3932.8
head(is.na(diamonds$price))
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
naa=is.na(diamonds$price)
table(naa)
## naa
## FALSE
## 53940
#random sample
sample(10,3,T)
## [1] 10 2 3
sample(10,5,F)
## [1] 4 6 10 9 7

```

```

rnorm(10,5,9)
## [1] -5.323364 10.778377 9.562595 25.202856 14.740653 6.146948 4.712989
## [8] 10.033625 13.576019 -2.904901
sample(53940,54,F)
## [1] 16015 3295 30810 6128 13020 17596 26258 13724 18290 49344 28803
## [12] 19296 30969 18564 47637 19556 36676 31809 27470 40427 13827 19285
## [23] 34027 19775 3914 42903 38505 22783 22571 24783 53796 49667 35219
## [34] 43201 41791 47455 22991 900 32144 27631 28891 40676 12270 9432
## [45] 25694 4886 20734 8846 28651 6460 33818 7642 21005 15168
sample(nrow(diamonds),0.001*(nrow(diamonds)),F)
## [1] 40131 24733 29152 4190 4079 31014 23678 20268 36029 37565 26792
## [12] 52564 31527 22044 4255 39092 40824 35166 4566 20044 8307 21015
## [23] 42075 36046 41057 20671 28080 5624 31169 49728 48181 17372 26373
## [34] 11403 37404 8279 25680 15130 23026 43130 43979 10054 43876 16751
## [45] 8193 25554 42141 3124 29700 45469 53186 25642 33776
a=nrow(diamonds)
sample(a,0.0001*a,F)
## [1] 40676 32758 43476 47130 38664
randomrows=sample(a,0.0001*a,F)
diamonds[randomrows,]
##          carat  cut          color  clarity  depth  table  price    x      y      z
## 29067    0.40   Good          F      SI1     63.1   58     687    4.66  4.69  2.95
## 28304    0.32  Very Good    I      SI1     62.8   58     432    4.34  4.39  2.74
## 25301    1.58  Very Good    G      VS1     62.8   57    13963    7.34  7.40  4.63
## 15670    1.00   Fair          E      VS2     57.3   64     6285    6.59  6.46  3.79
## 5267     1.03   Good          J      VS2     63.7   56     3795    6.42  6.35  4.07
#Descriptive Stats
summary(diamonds)
```

```

##          carat          cut          color          clarity
## Min.      :0.2000    Fair      :1610    D: 6775    SI1      :13065
## 1st Qu.   :0.4000    Good      :4906    E: 9797    VS2      :12258
## Median    :0.7000    Very Good:12082  F: 9542    SI2      : 9194
## Mean      :0.7979    Premium  :13791  G: 11292  VS1      : 8171
## 3rd Qu.   :1.0400    Ideal    :21551  H: 8304    VVS2     : 5066
## Max.      :5.0100                    I: 5422    VVS1     : 3655
##                                         J: 2808    (Other)  :2531
##          depth          table          price          x
## Min.      :43.00    Min.      :43.00    Min.      :326    Min.      :0.000
## 1st Qu.   :61.00    1st Qu.   :56.00    1st Qu.   :950    1st Qu.   :4.710
## Median    :61.80    Median    :57.00    Median    :2401    Median    :5.700
## Mean      :61.75    Mean      :57.46    Mean      :3933    Mean      :5.731
## 3rd Qu.   :62.50    3rd Qu.   :59.00    3rd Qu.   :5324    3rd Qu.   :6.540
## Max.      :79.00    Max.      :95.00    Max.      :18823    Max.      :10.740
##
##          y          z
## Min.      : 0.000    Min.      : 0.000
## 1st Qu.   : 4.720    1st Qu.   : 2.910
## Median    : 5.710    Median    : 3.530
## Mean      : 5.735    Mean      : 3.539
## 3rd Qu.   : 6.540    3rd Qu.   : 4.040
## Max.      :58.900    Max.      :31.800
##

```

```

table(diamonds$cut)
##
## Fair      Good      Very Good      Premium      Ideal
## 1610      4906      12082           13791      21551
table(diamonds$cut,diamonds$color)
##
##           D      E      F      G      H      I      J
## Fair           163    224    312    314    303    175    119
## Good           662    933    909    871    702    522    307
## Very Good    1513   2400   2164   2299   1824   1204   678
## Premium      1603   2337   2331   2924   2360   1428   808
## Ideal        2834   3903   3826   4884   3115   2093   896
table(diamonds$cut,diamonds$color,diamonds$clarity)
## , , =I1
##
##
##           D      E      F      G      H      I      J
## Fair           4      9     35     53     52     34     23
## Good           8     23     19     19     14      9      4
## Very Good      5     22     13     16     12      8      8
## Premium        12     30     34     46     46     24     13
## Ideal          13     18     42     16     38     17      2
##
## , , =SI2
##
##
##           D      E      F      G      H      I      J
## Fair          56     78     89     80     91     45     27
## Good          223    202    201    163    158     81     53
## Very Good     314    445    343    327    343    200    128
## Premium        421    519    523    492    521    312    161
## Ideal          356    469    453    486    450    274    110
##
## , , =SI1
##
##
##           D      E      F      G      H      I      J
## Fair          58     65     83     69     75     30     28
## Good          237    355    273    207    235    165     88
## Very Good     494    626    559    474    547    358    182
## Premium        556    614    608    566    655    367    209
## Ideal          738    766    608    660    763    504    243
##

```

```
## , , =VS2
##
##
##      D      E      F      G      H      I      J
## Fair      25     42     53     45     41     32     23
## Good     104    160    184    192    138    110     90
## Very Good 309    503    466    479    376    274    184
## Premium   339    629    619    721    532    315    202
## Ideal     920   1136    879    910    556    438    232
##
## , , =VS1
##
##
##      D      E      F      G      H      I      J
## Fair       5     14     33     45     32     25     16
## Good      43     89    132    152     77    103     52
## Very Good 175    293    293    432    257    205    120
## Premium   131    292    290    566    336    221    153
## Ideal     351    593    616    953    467    408    201
##
## , , =VVS2
##
##
##      D      E      F      G      H      I      J
## Fair       9     13     10     17     11      8      1
## Good      25     52     50     75     45     26     13
## Very Good 141    298    249    302    145     71     29
## Premium    94    121    146    275    118     82     34
## Ideal     284    507    520    774    289    178     54
##
## , , =VVS1
##
##
##      D      E      F      G      H      I      J
## Fair       3      3      5      3      1      1      1
## Good      13     43     35     41     31     22      1
## Very Good  52    170    174    190    115     69     19
## Premium    40    105     80    171    112     84     24
## Ideal     144    335    440    594    326    179     29
##
## , , =IF
##
##
```

```

##           D      E      F      G      H      I      J
## Fair      3      0      4      2      0      0      0
## Good      9      9     15     22      4      6      6
## Very Good 23     43     67     79     29     19      8
## Premium   10     27     31     87     40     23     12
## Ideal     28     79    268    491    226     95     25
mean(diamonds$price)
## [1] 3932.8
#using Hmisc
library(Hmisc)
## Loading required package: grid
## Loading required package: lattice
## Loading required package: survival
## Loading required package: Formula
##
## Attaching package: 'Hmisc'
##
## The following objects are masked from 'package:base':
##
##   format.pval, round.POSIXt, trunc.POSIXt, units
describe(diamonds$price)
## diamonds$price
##      n missing unique Info Mean .05 .10 .25 .50
## 53940      0  11602     1 3933 544 646 950 2401
##   .75   .90   .95
##  5324   9821 13107
##
## lowest :   326   327   334   335   336
## highest: 18803 18804 18806 18818 18823
summarize(diamonds$price, diamonds$color, mean)
## diamonds$color diamonds$price
## 1      D      3169.954
## 2      E      3076.752
## 3      F      3724.886
## 4      G      3999.136
## 5      H      4486.669
## 6      I      5091.875
## 7      J      5323.818
summarize(diamonds$price, diamonds$color, max)
## diamonds$color diamonds$price
## 1      D      18693
## 2      E      18731
## 3      F      18791

```

```
## 4      G      18818
## 5      H      18803
## 6      I      18823
## 7      J      18710
summarize(diamonds$price, llist(diamonds$color, diamonds
$cut), mean)
##  diamonds$color diamonds$cut diamonds$price
## 1      D      Fair      4291.061
## 2      D      Good      3405.382
## 5      D      Very Good  3470.467
## 4      D      Premium   3631.293
## 3      D      Ideal     2629.095
## 6      E      Fair      3682.312
## 7      E      Good      3423.644
## 10     E      Very Good  3214.652
## 9      E      Premium   3538.914
## 8      E      Ideal     2597.550
## 11     F      Fair      3827.003
## 12     F      Good      3495.750
## 15     F      Very Good  3778.820
## 14     F      Premium   4324.890
## 13     F      Ideal     3374.939
## 16     G      Fair      4239.255
## 17     G      Good      4123.482
## 20     G      Very Good  3872.754
## 19     G      Premium   4500.742
## 18     G      Ideal     3720.706
## 21     H      Fair      5135.683
## 22     H      Good      4276.255
## 25     H      Very Good  4535.390
## 24     H      Premium   5216.707
## 23     H      Ideal     3889.335
## 26     I      Fair      4685.446
## 27     I      Good      5078.533
## 30     I      Very Good  5255.880
## 29     I      Premium   5946.181
## 28     I      Ideal     4451.970
## 31     J      Fair      4975.655
## 32     J      Good      4574.173
## 35     J      Very Good  5103.513
## 34     J      Premium   6294.592
## 33     J      Ideal     4918.186
#reshape
```

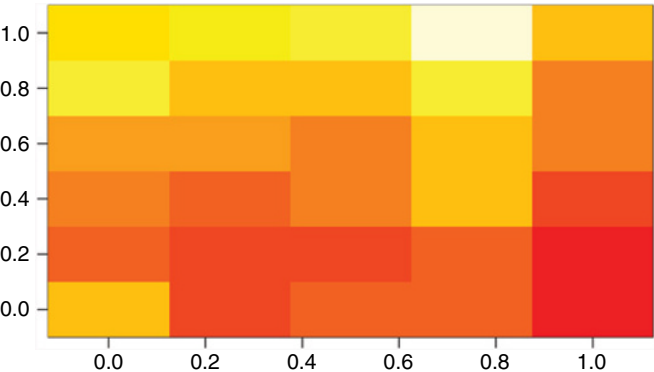
```
library(reshape2)
acast(diamonds, cut~color, value.var='price', mean)
##           D           E           F           G           H           I           J
## Fair      4291.061  3682.312  3827.003  4239.255  5135.683  4685.446  4975.655
## Good      3405.382  3423.644  3495.750  4123.482  4276.255  5078.533  4574.173
## Very Good 3470.467  3214.652  3778.820  3872.754  4535.390  5255.880  5103.513
## Premium   3631.293  3538.914  4324.890  4500.742  5216.707  5946.181  6294.592
## Ideal     2629.095  2597.550  3374.939  3720.706  3889.335  4451.970  4918.186
with(diamonds, tapply(price, list(cut,color), FUN= mean))
##           D           E           F           G           H           I           J
## Fair      4291.061  3682.312  3827.003  4239.255  5135.683  4685.446  4975.655
## Good      3405.382  3423.644  3495.750  4123.482  4276.255  5078.533  4574.173
## Very Good 3470.467  3214.652  3778.820  3872.754  4535.390  5255.880  5103.513
## Premium   3631.293  3538.914  4324.890  4500.742  5216.707  5946.181  6294.592
## Ideal     2629.095  2597.550  3374.939  3720.706  3889.335  4451.970  4918.186
xtabs(price ~ cut + color, diamonds)/table(diamonds[c('cut', 'color')])
##           color
## cut           D           E           F           G           H           I           J
## Fair      4291.061  3682.312  3827.003  4239.255  5135.683  4685.446  4975.655
## Good      3405.382  3423.644  3495.750  4123.482  4276.255  5078.533  4574.173
## Very Good 3470.467  3214.652  3778.820  3872.754  4535.390  5255.880  5103.513
## Premium   3631.293  3538.914  4324.890  4500.742  5216.707  5946.181  6294.592
## Ideal     2629.095  2597.550  3374.939  3720.706  3889.335  4451.970  4918.186
library(data.table)
```

```
dcast(as.data.table(diamonds), cut~color, value.var='price', mean)

##      cut          D          E          F          G          H          I          J
## 1 Fair      4291.061  3682.312  3827.003  4239.255  5135.683  4685.446  4975.655
## 2 Good      3405.382  3423.644  3495.750  4123.482  4276.255  5078.533  4574.173
## 3 Very Good 3470.467  3214.652  3778.820  3872.754  4535.390  5255.880  5103.513
## 4 Premium   3631.293  3538.914  4324.890  4500.742  5216.707  5946.181  6294.592
## 5 Ideal     2629.095  2597.550  3374.939  3720.706  3889.335  4451.970  4918.186
library(dplyr)
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:data.table':
##
##   between, last
##
## The following objects are masked from 'package:Hmisc':
##
##   combine, src, summarize
##
## The following object is masked from 'package:stats':
##
##   filter
##
## The following objects are masked from 'package:base':
##
##   intersect, setdiff, setequal, union
library(tidyr)
```

```
b=diamonds %>%
  group_by(cut, color) %>%
  summarise(price = mean(price)) %>%
  spread(color, price)
b
## Source: local data frame [5 x 8]
##
##   cut          D          E          F          G          H          I          J
## 1 Fair      4291.061  3682.312  3827.003  4239.255  5135.683  4685.446  4975.655
## 2 Good      3405.382  3423.644  3495.750  4123.482  4276.255  5078.533  4574.173
## 3 Very Good 3470.467  3214.652  3778.820  3872.754  4535.390  5255.880  5103.513
## 4 Premium   3631.293  3538.914  4324.890  4500.742  5216.707  5946.181  6294.592
## 5 Ideal     2629.095  2597.550  3374.939  3720.706  3889.335  4451.970  4918.186
str(b)
## Classes 'tbl_df', 'tbl' and 'data.frame': 5 obs. of 8 variables:
## $ cut: Ord.factor w/ 5 levels "Fair"<"Good"<...:
##   1 2 3 4 5
## $ D : num 4291 3405 3470 3631 2629
## $ E : num 3682 3424 3215 3539 2598
## $ F : num 3827 3496 3779 4325 3375
## $ G : num 4239 4123 3873 4501 3721
## $ H : num 5136 4276 4535 5217 3889
## $ I : num 4685 5079 5256 5946 4452
## $ J : num 4976 4574 5104 6295 4918
```

```
image(as.matrix(b[2:7]))
```



```
#subset
cut2=diamonds[diamonds$cut=="Ideal",]
head(cut2)
##      carat cut    color clarity depth  table  price  x      y      z
## 1  0.23  Ideal  E      SI2     61.5   55    326  3.95  3.98  2.43
## 12 0.23  Ideal  J      VS1     62.8   56    340  3.93  3.90  2.46
## 14 0.31  Ideal  J      SI2     62.2   54    344  4.35  4.37  2.71
## 17 0.30  Ideal  I      SI2     62.0   54    348  4.31  4.34  2.68
## 40 0.33  Ideal  I      SI2     61.8   55    403  4.49  4.51  2.78
## 41 0.33  Ideal  I      SI2     61.2   56    403  4.49  4.50  2.75
cut3=diamonds[diamonds$cut=="Ideal" & diamonds$color=="D",]
```

```
head(cut3)
##      carat cut      color clarity depth table price x      y      z
## 63  0.30  Ideal  D        SI1      62.5  57     552  4.29 4.32 2.69
## 64  0.30  Ideal  D        SI1      62.1  56     552  4.30 4.33 2.68
## 121 0.71  Ideal  D        SI2      62.3  56     2762  5.73 5.69 3.56
## 133 0.71  Ideal  D        SI1      61.9  59     2764  5.69 5.72 3.53
## 145 0.71  Ideal  D        SI2      61.6  55     2767  5.74 5.76 3.54
## 156 0.76  Ideal  D        SI2      62.4  57     2770  5.78 5.83 3.62
cut4=diamonds[diamonds$cut=="Ideal" | diamonds$color=="D",]
head(cut4)
##      carat cut      color clarity depth table price x      y      z
## 1  0.23  Ideal      E        SI2      61.5  55     326  3.95 3.98 2.43
## 12 0.23  Ideal      J        VS1      62.8  56     340  3.93 3.90 2.46
## 14 0.31  Ideal      J        SI2      62.2  54     344  4.35 4.37 2.71
## 17 0.30  Ideal      I        SI2      62.0  54     348  4.31 4.34 2.68
## 29 0.23  Very Good D        VS2      60.5  61     357  3.96 3.97 2.40
## 35 0.23  Very Good D        VS1      61.9  58     402  3.92 3.96 2.44
cut5=ifelse(diamonds$price>9000,"Expensive","Not So Expensive")
table(cut5)
## cut5
##      Expensive Not So Expensive
##      6298      47642
```

5

Statistical Modeling

5.1 Concepts in Regression

What is statistical modeling?

- It is a formalization of relationships between variables in the form of mathematical equations.
- It describes how one or more random variables are related to one or more other variables.
- The variables are not deterministically but stochastically related.

Reading Statistical Modeling: The Two Cultures http://projecteuclid.org/download/pdf_1/euclid.ss/1009213726

Example

- Height and age are probabilistically distributed among humans.
- They are stochastically related; when you know that a person is of age 30 years, this influences the chance of this person of being 4-feet tall. When you know that a person is of age 13 years, this influences the chance of this person of being 6 feet tall.
- Model 1
 - $\text{height}_i = b_0 + b_1 \text{age}_i + \epsilon_i$, where b_0 is the intercept, b_1 is a parameter that age is multiplied by to get a prediction of height, ϵ is the error term, and i is the subject.
- Model 2
 - $\text{height}_i = b_0 + b_1 \text{age}_i + b_2 \text{sex}_i + \epsilon_i$, where the variable sex is dichotomous.

Regression models involve the following variables:

- The **unknown parameters**
- The **independent variables**, X
- The **dependent variable**, Y

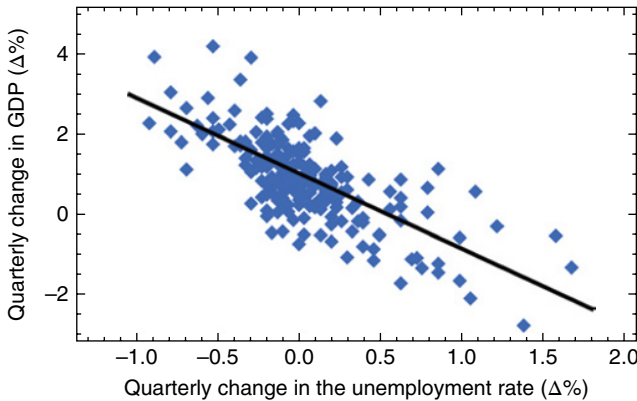


Figure 5.1 Okun's law.

$Y = a + BX$ is the simplest form of regression

Linear regression $Y = a + Bx + (E)$

Multivariate regression $Y = a + bx + cy + (E)$

Logistic regression $\ln(p/1 - p) = a + bX$

Example

Okun's Law

The relationship between an economy's unemployment rate and its gross national product (GNP). Economist Arthur Okun developed this idea, which states that when unemployment falls by 1%, GNP rises by 3% (Figure 5.1).

5.1.1 OLS

Ordinary least squares (OLS) or **linear least squares** is a method for estimating the unknown parameters in a linear regression model, with the goal of minimizing the differences between the observed responses in some arbitrary dataset and the responses predicted by the linear approximation of the data (visually this is seen as the sum of the vertical distances between each data point in the set and the corresponding point on the regression line—the smaller the differences, the better the model fits the data) (https://en.wikipedia.org/wiki/Ordinary_least_squares). The **primary assumption of OLS is that there are zero or negligible errors in the independent variable, since this method only attempts to minimize the mean squared error in the dependent variable**. The method of **least squares** is a standard approach in regression analysis to the approximate solution of overdetermined systems, that is, sets of equations in which there are more equations than unknowns. “Least squares” means that the overall solution minimizes the sum of the squares of the errors made in the results of every single equation.

The most important application is in data fitting. The best fit in the least-squares sense minimizes the sum of squared residuals, a residual being the difference between an observed value and the fitted value provided by a model.

https://en.wikipedia.org/wiki/Least_squares.

5.1.2 R-Squared

In statistics, the **coefficient of determination**, denoted R^2 or r^2 and pronounced “R-squared,” is a number that indicates the proportion of the variance in the dependent variable that is predictable from the independent variable.

It is a statistic used in the context of statistical models whose main purpose is either the prediction of future outcomes or the testing of hypotheses, on the basis of other related information.

The use of an adjusted R^2 (often written as $\bar{R}^2$ $\{\displaystyle \{\bar{R}\}^{\{2\}}\}$ and pronounced “R-bar squared”) is an attempt to take account of the phenomenon of the R^2 automatically and spuriously increasing when extra explanatory variables are added to the model.

https://en.wikipedia.org/wiki/Coefficient_of_determination#Adjusted_R2.

To summarize, R-squared is the percentage of the response variable variation that is explained by a linear model. Or

$$\text{R-squared} = \frac{\text{Explained variation}}{\text{Total variation}}$$

Adjusted R-squared adjusts the statistic based on the number of independent variables in the model.

5.1.3 p-Value

The p-value for each term tests the null hypothesis that the coefficient is equal to zero (no effect). A low p-value (**<0.05**) indicates that you can reject the null hypothesis.

In other words, a predictor that has a low p-value is likely to be a meaningful addition to your model because changes in the predictor’s value are related to changes in the response variable.

<http://blog.minitab.com/blog/adventures-in-statistics/how-to-interpret-regression-analysis-results-p-values-and-coefficients>

5.1.4 Outliers

Sometimes outliers are bad data and should be excluded, such as typos. Sometimes they are Wayne Gretzky or Michael Jordan and should be kept.

Statistical distance measures are specifically catered to detecting outliers and then consider whether such outliers should be removed from your linear regression.

The first one is Cook's distance. You can find a pretty good explanation of it at Wikipedia (http://en.wikipedia.org/wiki/Cook%27s_distance).

The higher the Cook's distance is, the more influential (impact on regression coefficient) the observation is. The typical cutoff point to consider removing the observation is a Cook's distance = $4/n$ (n is sample size).

<http://stats.stackexchange.com/questions/175/how-should-outliers-be-dealt-with-in-linear-regression-analysis>

The second way is to use `outlierTest` function from `car` package in R.

5.1.5 Multicollinearity and Heteroscedasticity

Multicollinearity is a statistical phenomenon in which two or more predictor variables in a multiple regression model are highly correlated, meaning that one can be linearly predicted from the others with a nontrivial degree of accuracy. In this situation the coefficient estimates may change erratically in response to small changes in the model or the data.

vif from car package

In statistics, a collection of random variables is **heteroscedastic** (often spelled **heteroskedastic** and commonly pronounced with a hard *k* sound regardless of spelling) if there are subpopulations that have different variabilities from others. Here "variability" could be quantified by the variance or any other measure of statistical dispersion.

gvlma package

5.2 Correlation Is Not Causation

Correlation does not imply causation is a phrase used in statistics to emphasize that a correlation between two variables does not imply that one causes the other (Figures 5.2 and 5.3).

Both the aforementioned charts show the absurdity that occurs when we suppose correlation is the same as a causal relation.

"Causes" is an asymmetric relation (X causes Y is different from Y causes X), whereas "is correlated with" is a symmetric relation.

For instance, homeless population and crime rate might be correlated, in that both tend to be high or low in the same locations. It is equally valid to say that homeless population is correlated with crime rate, or crime rate is correlated with homeless population. For example, crime causes homelessness and homeless populations cause crime are different statements. And correlation does not imply that either is true. For instance, the underlying cause could be a third variable such as drug abuse or unemployment.

The mathematics of statistics is not good at identifying underlying causes, which requires some other form of judgment (Figure 5.4).

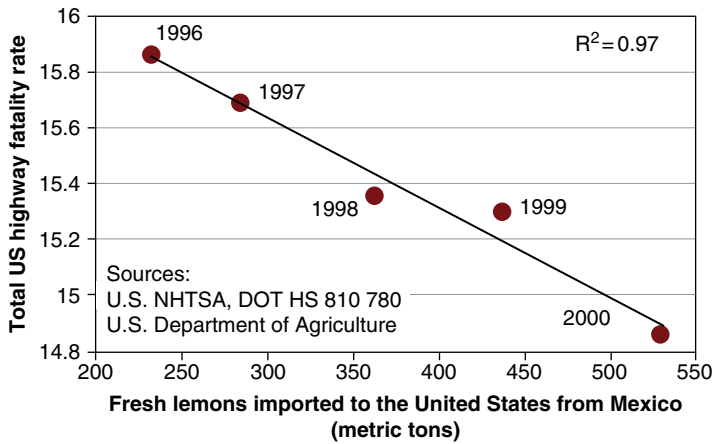


Figure 5.2 <http://pubs.acs.org/doi/abs/10.1021/ci700332k>. Source: Johnson (2008). Reproduced with the permission of American Chemical Society.

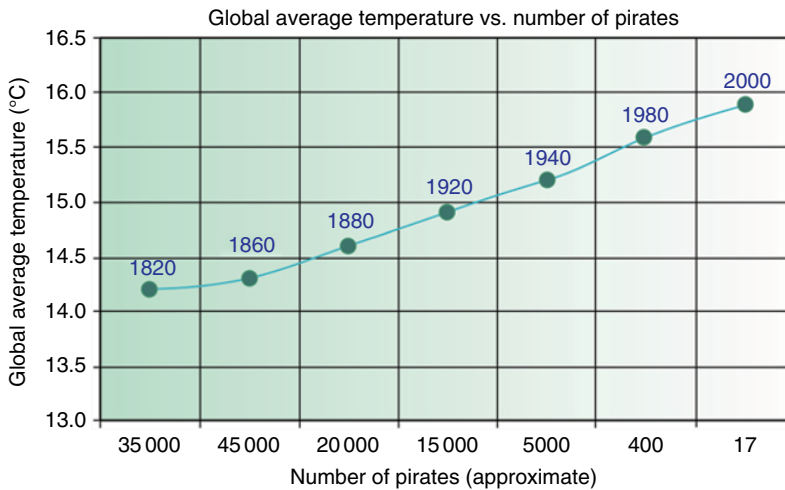


Figure 5.3 <https://www.forbes.com/sites/erikaandersen/2012/03/23/true-fact-the-lack-of-pirates-is-causing-global-warming/>. Source: © Forbes.com.

5.2.1 A Note on Statistics for Data Scientists

Data scientists tend to be either computer science leaning or statistics leaning. In languages, R is preferred by those who are from statistical background and Python often by computer science background. Both programming and statistics are needed for a balanced skill set in data analysis.

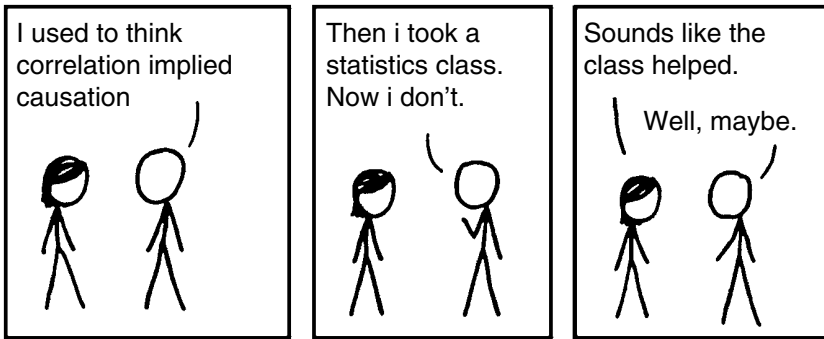


Figure 5.4 XKCD.com cartoon correlation is not causation. <http://stats.stackexchange.com/questions/36/examples-for-teaching-correlation-does-not-mean-causation>. Source: © Stack Exchange Inc.

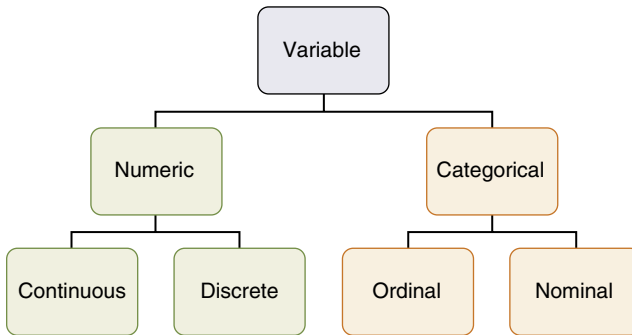


Figure 5.5 Types of variables. With reference to <https://statistics.laerd.com/statistical-guides/types-of-variable.php>. Source: © Lund Research Ltd.

A brief summary of statistics needed for data scientists is at <https://www.slideshare.net/ajayohri/statistics-for-data-scientists>.

Here is a brief extract:

Data—Facts and statistics collected together for reference or analysis

Variable—Something that varies (Figure 5.5)

Ordinal variables are variables that have two or more categories just like nominal variables, only the categories can also be ordered or ranked, for example, excellent–horrible. **Dichotomous** variables are nominal variables that have only two categories or levels. **Nominal** variables are variables that have two or more categories, but do not have an intrinsic order.

Interval variables are variables whose central characteristic is that they can be measured along a continuum and have a numerical value (e.g., temperature measured in degrees Celsius or Fahrenheit).

Ratio variables are interval variables but with the added condition that 0 (zero) of the measurement indicates that there is none of that variable. A distance of 10 m is twice the distance of 5 m.

5.2.2 Measures of Central Tendency

Mean

Arithmetic mean is the sum of the values divided by the number of values.

The geometric mean is an average that is useful for sets of positive numbers that are interpreted according to their product and not their sum (as is the case with the arithmetic mean), for example, rates of growth.

Median

The **median** is the number separating the higher half of a data sample, a population, or a probability distribution from the lower half.

Mode

The “mode” is the value that occurs most often.

5.2.3 Measures of Dispersion

Range

The **range** of a set of data is the difference between the largest and smallest values.

Variance

Mean of squares of differences of values from mean

Standard Deviation (sd)

Square root of its variance

Frequency

A **frequency distribution** is a table that displays the **frequency** of various outcomes in a sample.

What Is a Distribution?

The **distribution** of a **statistical** dataset (or a population) is a listing or function showing all the possible values (or intervals) of the data and how often they occur. When a **distribution** of categorical data is organized, you see the number or percentage of individuals in each group (<http://www.dummies.com/education/math/statistics/what-the-distribution-tells-you-about-a-statistical-data-set/>).

The simplest case of a normal distribution is known as the *standard normal distribution* (Figure 5.6). This is a special case where mean $\mu = 0$ and standard deviation $\sigma = 1$.

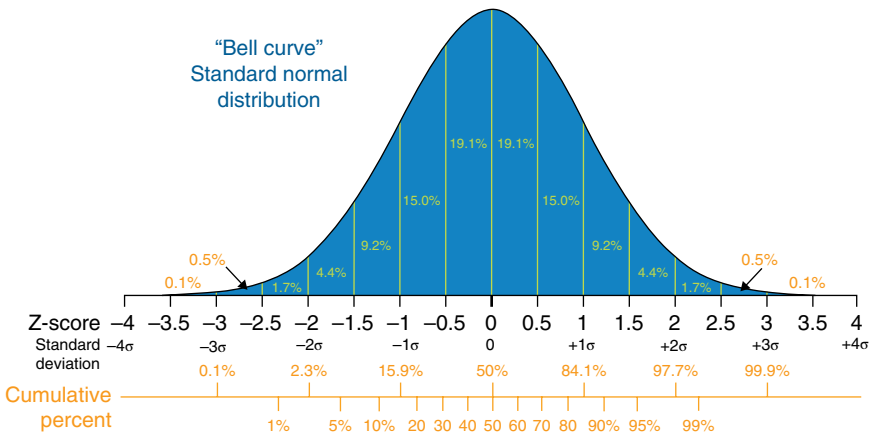


Figure 5.6 Standard normal distribution—a most useful distribution curve for a data scientist. Source: Rumsey (2016). Reproduced with the permission of John Wiley & Sons, Inc.

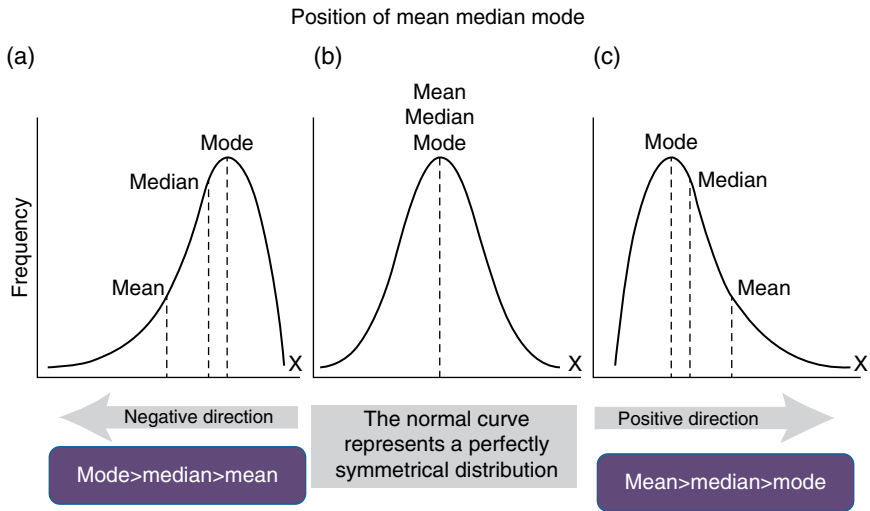


Figure 5.7 Skewed curves. (a) Negatively skewed, (b) normal (no skew), and (c) positively skewed.

Clearly we won't get a normal distribution all the time for skewed or tilted distribution when the following measures, skewness, and kurtosis are used (Figure 5.7).

Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. The skewness value can be positive or negative or even undefined (Figure 5.8).

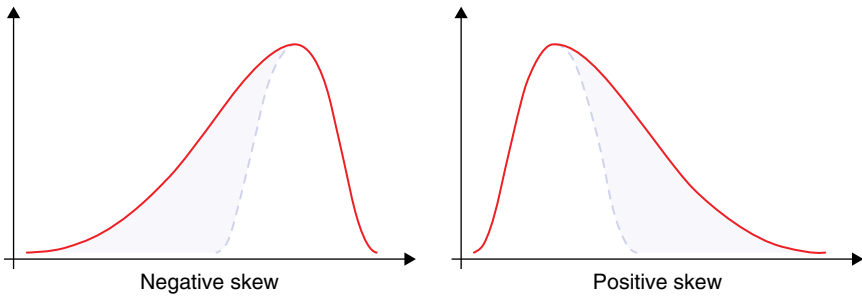


Figure 5.8 Skewness. [https://en.wikipedia.org/wiki/File:Negative_and_positive_skew_diagrams_\(English\).svg](https://en.wikipedia.org/wiki/File:Negative_and_positive_skew_diagrams_(English).svg). Source: © Wikipedia.

Kurtosis is a measure of the “tailedness” of the probability distribution of a real-valued random variable. *Kurtosis* is a descriptor of the shape of a probability distribution (Figure 5.9).

Some useful distributions apart from normal distribution are the following:

Bernoulli—Distribution of a random variable that takes value 1 with success probability and value 0 with failure probability. It can be used, for example, to represent the toss of a coin.

Chi-square—The distribution of a sum of the squares of k independent standard normal random variables (Figure 5.10).

Poisson—A discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time and/or space if these events occur with a known average rate and independently of the time since the last event.

You can see others at https://en.wikipedia.org/wiki/Probability_distribution#Discrete_probability_distribution (Figure 5.11).

5.2.4 Probability Distribution

The probability density function (pdf) (http://en.wikipedia.org/wiki/Probability_density_function) of the normal distribution, also called Gaussian or “bell curve,” is the most important continuous random distribution (Figure 5.12). As notated on the figure, the probabilities of intervals of values correspond to the area under the curve.

In probability theory, the **central limit theorem (CLT)** states that, given certain conditions, the arithmetic mean of a sufficiently large number of iterates of independent random variables, each with a well-defined expected value and well-defined variance, will be approximately normally distributed, regardless of the underlying distribution.

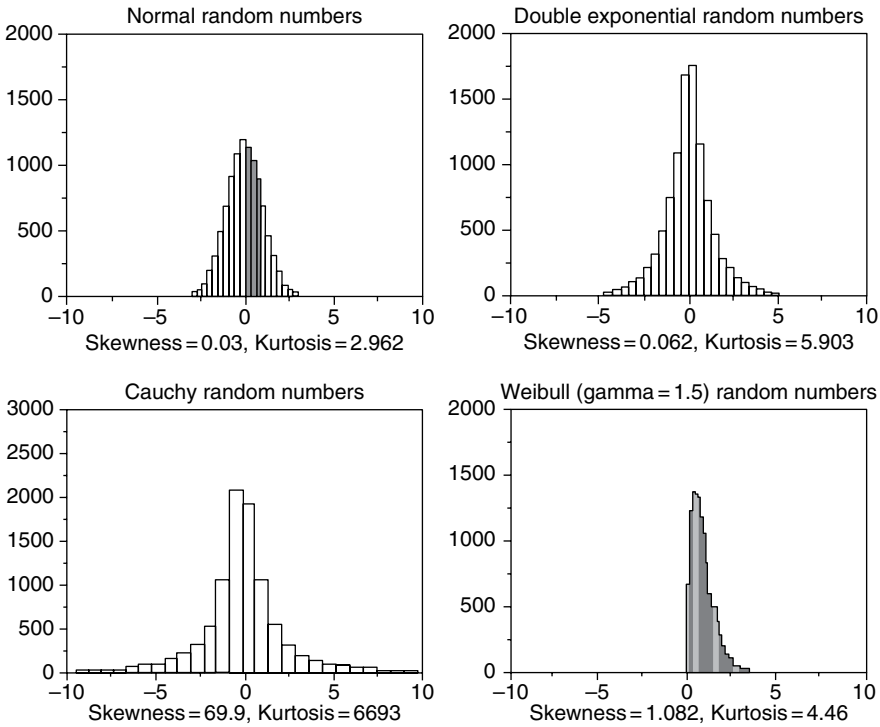


Figure 5.9 Kurtosis. <http://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm>.
Source: © U.S. Department of Commerce.

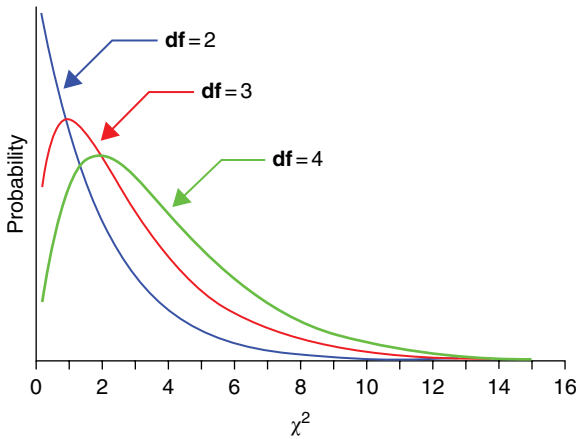


Figure 5.10 Chi square curve.

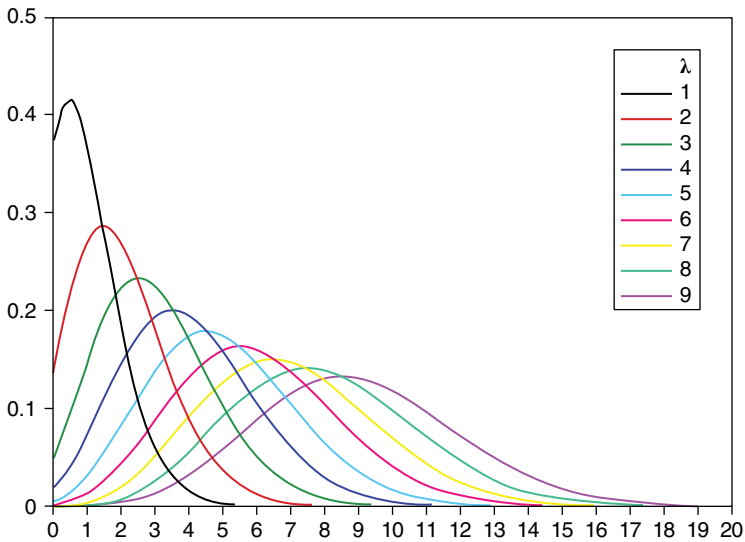


Figure 5.11 Poisson curve. Source: © Wikipedia.

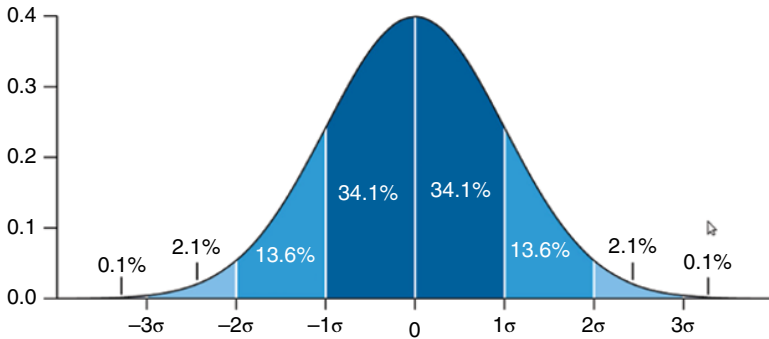


Figure 5.12 Normal distribution curve.

A widely underused technique by computer scientists is hypothesis testing. What is hypothesis testing? Hypothesis testing is the use of statistics to determine the probability that a given hypothesis is true (<http://mathworld.wolfram.com/HypothesisTesting.html>). The usual process of hypothesis testing consists of four steps:

- 1) Formulate the null hypothesis (commonly, that the observations are the result of pure chance) and the alternative hypothesis (commonly, that the observations show a real effect combined with a component of chance variation).

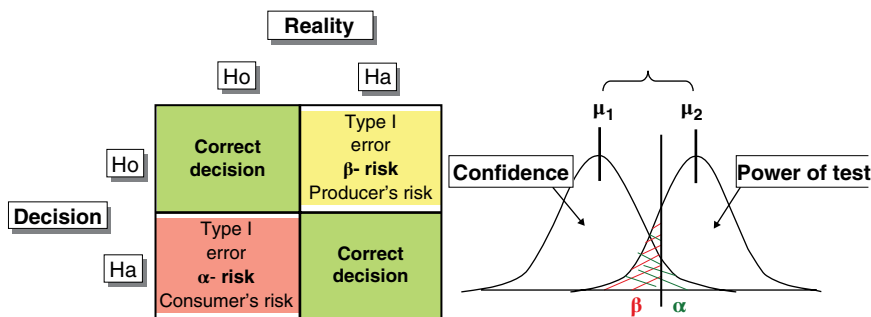


Figure 5.13 Type 1 and Type 2 errors.

- 2) Identify a test statistic that can be used to assess the truth of the null hypothesis.
- 3) Compute the p-value, which is the probability that a test statistic at least as significant as the one observed would be obtained assuming that the null hypothesis were true. The smaller the p-value, the stronger the evidence against the null hypothesis.
- 4) Compare the p-value with an acceptable significance value (sometimes called an alpha value). If the observed effect is statistically significant, the null hypothesis is ruled out, and the alternative hypothesis is valid.

This can be represented by (Figure 5.13)

The researcher's decision	The truth (unknown to the researcher)	
	The null hypothesis is true	The null hypothesis is false
Reject the null hypothesis	Type I error	Correct decision
Fail to reject the null hypothesis	Correct decision	Type II error

What are various kinds of tests? (Figure 5.14)

A slightly easier way to understand which among various tests to use is the RATTLE GUI in R. Some R code for Z tests can be found at <http://rpubs.com/newajay/stats4>. Here is a Z test to reject or accept if sample mean is >10000.

```
#null hypothesis umean >=10000
xbar=9900 (sample mean)
umean=10000 (population mean)
sd=120 (standard deviation)
n=30 number of observations
```

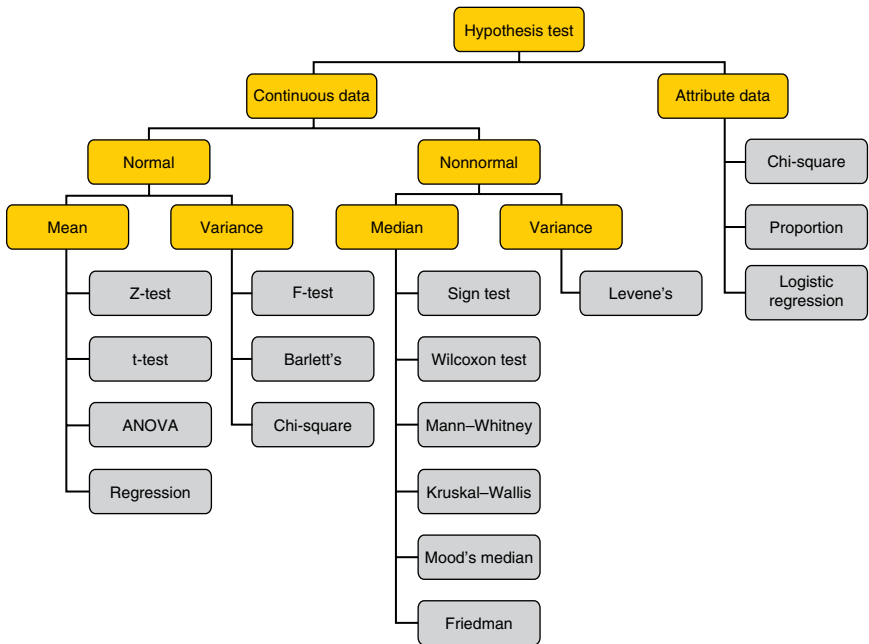


Figure 5.14 Types of hypothesis tests.

```
z = (xbar - umean) / (sd / sqrt(n))
```

```
z
```

```
## [1] -4.564355
```

```
alpha = 0.05
```

```
z.alpha = qnorm(1 - alpha)
```

```
-z.alpha
```

```
## [1] -1.644854
```

```
#NULL hypothesis is rejected
```

This may seem difficult for the non-statistician to understand unless they visualize on the normal distribution where these values are occurring (in acceptance zone or rejection zone). Here the value -4.5 is much less than -1.65 so it is clearly in the rejection zone. For upper tail, the following shows a rejection case since it is greater than 1.645 (please read from <http://www.stat.wmich.edu/s216/htests/htests.html#ztest4mu> (Figure 5.15) if interested more in this).

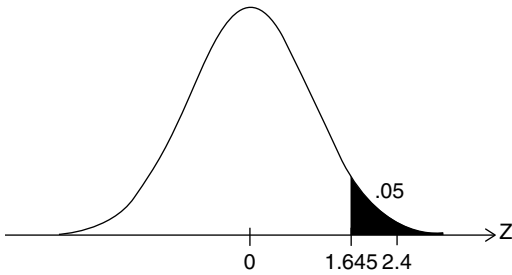


Figure 5.15 P-value and rejection zone.

Statistical Tests			
These tests apply to two samples. The paired two sample tests assume that we have two samples or observations, and that we are testing for a change, usually from one time period to another.			
Distribution of the Data			
* Kolomogorov-Smirnov	Non-parametric	Are the distributions the same?	
* Wilcoxon Signed Rank	Non-parametric	Do paired samples have the same distribution?	
Location of the Average			
* T-test	Parametric	Are the means the same?	
* Wilcoxon Rank-Sum	Non-parametric	Are the medians the same?	
Variation in the Data			
* F-test	Parametric	Are the variances the same?	
Correlation			
* Correlation	Pearsons	Are the values from the paired samples correlated?	

Figure 5.16 An easy way to explain hypothesis tests using Rattle GUI in R.

In Python we can also look at statsmodels for tests than SciPy (see <http://statsmodels.sourceforge.net/stable/generated/statsmodels.stats.weightstats.ztest.html#statsmodels.stats.weightstats.ztest> (Figure 5.16)).

Here is an example in R of chi-square test to see if exercising affects smoking. <http://rpubs.com/newajay/chisquaretest>.

```
library(MASS)
tbl = table(survey$Smoke, survey$Exer)
tbl
##
##           Freq None Some
## Heavy      7      1    3
## Never     87     18   84
## Occas     12      3    4
## Regul      9      1    7
```

```

table(survey$Smoke)

##
## Heavy Never Occas Regul
##      11      189      19      17

dim(survey)

## [1] 237  12

#Test the hypothesis whether the students
#smoking habit is independent of
#their exercise level at .05 significance level.
chisq.test(tbl)

## Warning in chisq.test(tbl): Chi-squared approximation
## may be incorrect
##
## Pearson's Chi-squared test
##
## data:  tbl
## X-squared = 5.4885, df = 6, p-value = 0.4828

#As the p-value 0.4828 is greater than the .05
significance level, we do not reject the null
hypothesis that the smoking habit is
#independent of the exercise level of the students.
ctbl = cbind(tbl[, "Freq"], tbl[, "None"] + tbl[, "Some"])
ctbl

##           [,1] [,2]
## Heavy         7    4
## Never        87   102
## Occas        12    7
## Regul         9    8

chisq.test(ctbl)

##
## Pearson's Chi-squared test
##
## data:  ctbl

```

```
## X-squared = 3.2328, df = 3, p-value = 0.3571
#As the p-value 0.3571 is greater than the .05
significance level, we do not reject the null
hypothesis that the smoking habit is independent
of the exercise level of the students. The warning
message found in the solution above is due to the
small cell values in the contingency table
```

We redo the same in Python using scipy and numpy very easily.

```
In [11]:
from scipy.stats import chi2_contingency
import numpy as np

In [13]:
obs = np.array([[7, 87, 12,9], [4, 102, 7,8]])

In [15]:
chi2, p, dof, expected = chi2_contingency(obs)

In [16]:
print (p)

0.357103080041
```

<https://github.com/decisionstats/pythonfordatascience/blob/master/chi%2Bsquare%2Btest.ipynb>

5.3 Linear Regression in R and Python

In Python, statsmodels can be used for linear regression. Here is an example for iris dataset from <https://nbviewer.jupyter.org/gist/decisionstats/8ac83dbe4dd08808af3d9c0869259cf6>

```
import pandas as pd

In [3]:
import statsmodels.formula.api as sm

In [4]:
iris=pd.read_csv("http://vincentarelbundock.github.io/
Rdatasets/csv/datasets/iris.csv")

In [6]:
iris =iris.drop('Unnamed: 0', 1)

In [7]:
iris.head()

Out [7]:
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
In [15]:
iris.columns=['Sepal_Length', 'Sepal_Width', 'Petal_
Length', 'Petal_Width',
'Species']
```

```
In [16]:
iris.columns
```

```
Out [16]:
Index(['Sepal_Length', 'Sepal_Width', 'Petal_Length',
'Petal_Width',
'Species'],
dtype='object')
```

```
In [17]:
result = sm.ols(formula="Sepal_Length ~ Petal_
Length + Sepal_Width + Petal_Width + Species",
data=iris)
```

```
In [18]:
result.fit()
```

```
Out [18]:
<statsmodels.regression.linear_model.
RegressionResultsWrapper at 0x9bafef0>
```

```
In [19]:
result.fit().summary()
```

```
Out [19]:
```

Dep. Variable:	Sepal_Length	R-squared:	0.867
Model:	OLS	Adj. R-squared:	0.863
Method:	Least Squares	F-statistic:	188.3
Date:	Mon, 13 Mar 2017	Prob (F-statistic):	2.67e-61
Time:	17:56:48	Log-Likelihood:	-32.558
No. Observations:	150	AIC:	77.12
Df Residuals:	144	BIC:	95.18
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[95.0% Conf. Int.]
Intercept	2.1713	0.280	7.760	0.000	1.618 2.724
Species[T.versicolor]	-0.7236	0.240	-3.013	0.003	-1.198-0.249
Species[T.virginica]	-1.0235	0.334	-3.067	0.003	-1.683-0.364
Petal_Length	0.8292	0.069	12.101	0.000	0.694 0.965
Sepal_Width	0.4959	0.086	5.761	0.000	0.326 0.666
Petal_Width	-0.3152	0.151	-2.084	0.039	-0.614-0.016

Omnibus:	0.418	Durbin-Watson:	1.966
Prob(Omnibus):	0.811	Jarque-Bera (JB):	0.572
Skew:	-0.060	Prob(JB):	0.751
Kurtosis:	2.722	Cond. No.	94.0

```
result.fit().params
```

In [20]:

Out [20]:

```
Intercept                2.171266
Species [T.versicolor]   -0.723562
Species [T.virginica]    -1.023498
Petal_Length             0.829244
Sepal_Width              0.495889
Petal_Width              -0.315155
dtype: float64
```

In R regression is done by the `lm` function (for linear models) and `glm` for logistic regression. Let us try some regression basics <http://rpubs.com/newajay/regbasics>.

```
ls()
## character(0)
getwd()
## [1] "C:/Users/dell/Desktop/regression"
dir()
## [1] "reg1.R"      "reg1.spin.R"  "reg1.spin.Rmd"
```

```
## [4] "regression.Rproj"
data("iris")
names(iris)

## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length"
##      "Petal.Width"
## [5] "Species"

lm(Sepal.Length~Sepal.Width,data = iris)

##
## Call:
## lm(formula = Sepal.Length ~ Sepal.Width, data = iris)
##
## Coefficients:
## (Intercept)  Sepal.Width
##      6.5262      -0.2234

a=lm(Sepal.Length~Sepal.Width,data = iris)
names(a)
## [1] "coefficients" "residuals"  "effects"      "rank"
## [5] "fitted.values" "assign"      "qr"          "df.residual"
## [9] "xlevels"      "call"        "terms"        "model"

class(a)

## [1] "lm"

a$coefficients

## (Intercept) Sepal.Width
##      6.5262226  -0.2233611

summary(a)

##
## Call:
## lm(formula = Sepal.Length ~ Sepal.Width, data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.5561 -0.6333 -0.1120  0.5579  2.2226
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    6.5262    0.4789   13.63  <2e-16 ***
```

```
## Sepal.Width  -0.2234      0.1551   -1.44    0.152
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8251 on 148 degrees of freedom
## Multiple R-squared:  0.01382,    Adjusted R-squared:  0.007159
## F-statistic: 2.074 on 1 and 148 DF,  p-value: 0.1519

b=lm(Sepal.Length~Sepal.Width + Petal.Length + Petal.Width,data = iris)
names(b)
## [1] "coefficients" "residuals"      "effects"        "rank"
## [5] "fitted.values" "assign"          "qr"             "df.residual"
## [9] "xlevels"       "call"           "terms"          "model"

class(b)

## [1] "lm"

b$coefficients

## (Intercept) Sepal.Width Petal.Length Petal.Width
## 1.8559975 0.6508372 0.7091320 -0.5564827

summary(b)

##
## Call:
## lm(formula = Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width,
## data = iris)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.82816 -0.21989  0.01875  0.19709  0.84570
##
## Coefficients:
##      Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.85600    0.25078    7.401 9.85e-12 ***
## Sepal.Width    0.65084    0.06665    9.765  < 2e-16 ***
## Petal.Length   0.70913    0.05672   12.502  < 2e-16 ***
## Petal.Width   -0.55648    0.12755   -4.363 2.41e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3145 on 146 degrees of freedom
## Multiple R-squared:  0.8586, Adjusted R-squared:  0.8557
## F-statistic: 295.5 on 3 and 146 DF,  p-value: < 2.2e-16

c=lm(Sepal.Length~Sepal.Width + Petal.Length + Petal.Width+Species,data = iris)
names(c)

## [1] "coefficients" "residuals"      "effects"        "rank"
## [5] "fitted.values" "assign"          "qr"             "df.residual"
## [9] "contrasts"     "xlevels"        "call"           "terms"
## [13] "model"

class(c)

## [1] "lm"
```

c\$coefficients

##	(Intercept)	Sepal.Width	Petal.Length	Petal.Width
##	2.1712663	0.4958889	0.8292439	-0.3151552
##	Speciesversicolor	Speciesvirginica		
##	-0.7235620	-1.0234978		

summary(c)

##				
##	Call:			
##	lm(formula = Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width +			
##	Species, data = iris)			
##				
##	Residuals:			
##	Min	1Q	Median	3Q
##	-0.79424	-0.21874	0.00899	0.20255
##				0.73103
##				
##	Coefficients:			
##	Estimate	Std. Error	t value	Pr(> t)
##	(Intercept)	2.17127	0.27979	7.760 1.43e-12 ***
##	Sepal.Width	0.49589	0.08607	5.761 4.87e-08 ***
##	Petal.Length	0.82924	0.06853	12.101 < 2e-16 ***
##	Petal.Width	-0.31516	0.15120	-2.084 0.03889 *
##	Speciesversicolor	-0.72356	0.24017	-3.013 0.00306 **
##	Speciesvirginica	-1.02350	0.33373	-3.067 0.00258 **
##	---			

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3068 on 144 degrees of freedom
## Multiple R-squared:  0.8673, Adjusted R-squared:  0.8627
## F-statistic: 188.3 on 5 and 144 DF,  p-value: < 2.2e-16
```

```
#mtcars
```

```
data("mtcars")
```

```
names(mtcars)
```

```
## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
## [11] "carb"
```

```
str(mtcars)
```

```
## 'data.frame': 32 obs. of 11 variables:
## $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num 160 160 108 258 360 ...
## $ hp : num 110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
## $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num 16.5 17 18.6 19.4 17 ...
## $ vs : num 0 0 1 1 0 1 0 1 1 1 ...
## $ am : num 1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num 4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num 4 4 1 1 2 1 4 2 2 4 ...
```

```
d=lm(mpg~cyl+disp+hp+drat+wt+qsec+vs+am+gear+carb,data = mtcars)
summary(d)
```

```
##
## Call:
## lm(formula = mpg ~ cyl + disp + hp + drat + wt + qsec + vs +
##      am + gear + carb, data = mtcars)
##
## Residuals:
```

##	Min	1Q	Median	3Q	Max
##	-3.4506	-1.6044	-0.1196	1.2193	4.6271

```
##
## Coefficients:
```

##	Estimate	Std. Error	t value	Pr(> t)
## (Intercept)	12.30337	18.71788	0.657	0.5181
## cyl	-0.11144	1.04502	-0.107	0.9161
## disp	0.01334	0.01786	0.747	0.4635
## hp	-0.02148	0.02177	-0.987	0.3350
## drat	0.78711	1.63537	0.481	0.6353
## wt	-3.71530	1.89441	-1.961	0.0633 .
## qsec	0.82104	0.73084	1.123	0.2739
## vs	0.31776	2.10451	0.151	0.8814
## am	2.52023	2.05665	1.225	0.2340
## gear	0.65541	1.49326	0.439	0.6652
## carb	-0.19942	0.82875	-0.241	0.8122

```
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.65 on 21 degrees of freedom
## Multiple R-squared:  0.869,    Adjusted R-squared:  0.8066
## F-statistic: 13.93 on 10 and 21 DF,  p-value: 3.793e-07
```

```
#diamonds
```

```
library(ggplot2)
data(diamonds)
str(diamonds)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':   53940 obs. of  10 variables:
## $ carat : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
## $ cut : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
## $ color : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
## $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
## $ depth : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
## $ table : num  55 61 65 58 58 57 57 55 61 61 ...
## $ price : int  326 326 327 334 335 336 336 337 337 338 ...
## $ x : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
## $ y : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
## $ z : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

```
diamonds$unitprice=with(diamonds,price/carat)
head(diamonds)
```

```
## # A tibble: 6 × 11
##   carat      cut color clarity depth table price      x      y      z
##   <dbl>    <ord> <ord>    <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23    Ideal     E      SI2   61.5   55   326   3.95   3.98   2.43
## 2  0.21   Premium     E      SI1   59.8   61   326   3.89   3.84   2.31
## 3  0.23     Good     E      VS1   56.9   65   327   4.05   4.07   2.31
## 4  0.29   Premium     I      VS2   62.4   58   334   4.20   4.23   2.63
## 5  0.31     Good     J      SI2   63.3   58   335   4.34   4.35   2.75
## 6  0.24 Very Good     J     VVS2   62.8   57   336   3.94   3.96   2.48
## # ... with 1 more variables: unitprice <dbl>
h=lm(unitprice~table+color+clarity+cut+x+y+z+depth,
      data=diamonds)
summary(h)

##
## Call:
## lm(formula = unitprice ~ table + color + clarity + cut + x +
##     y + z + depth, data = diamonds)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -5166.4  -463.9   -92.6   355.8 17851.4
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -11688.890   260.693   -44.838  < 2e-16 ***
## table         1.875     2.037     0.920   0.35732
## color.L      -1885.098   12.044  -156.515  < 2e-16 ***
```

```

## color.Q      -456.386    11.015    -41.434    < 2e-16 ***
## color.C      -78.133    10.305     -7.582    3.45e-14 ***
## color^4       78.474     9.464     8.292    < 2e-16 ***
## color^5      -56.637     8.941    -6.335    2.40e-10 ***
## color^6      -10.800     8.128    -1.329     0.18396
## clarity.L    3794.856    21.175    179.211    < 2e-16 ***
## clarity.Q   -1074.105    19.694    -54.540    < 2e-16 ***
## clarity.C     507.278    16.878     30.055    < 2e-16 ***
## clarity^4    -169.564    13.496    -12.564    < 2e-16 ***
## clarity^5     117.177    11.023     10.630    < 2e-16 ***
## clarity^6      47.190     9.598     4.917    8.83e-07 ***
## clarity^7     124.830     8.466     14.745    < 2e-16 ***
## cut.L        487.537    15.727     31.000    < 2e-16 ***
## cut.Q       -217.290    12.590    -17.259    < 2e-16 ***
## cut.C        127.423    10.836     11.759    < 2e-16 ***
## cut^4        16.718     8.661     1.930     0.05359 .
## x           1761.626    18.702     94.195    < 2e-16 ***
## y            67.623    13.523      5.001    5.74e-07 ***
## z            67.878    23.435      2.896     0.00378 **
## depth        73.620     3.110     23.671    < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 790.9 on 53917 degrees of freedom
## Multiple R-squared:  0.8456, Adjusted R-squared:  0.8456
## F-statistic: 1.343e+04 on 22 and 53917 DF,  p-value: < 2.2e-16

```

Note the three stars *** point to a low p-value in R's regression summary.

A slightly more elaborate way to see R's regression uses the car package from <http://rpubs.com/newajay/modelsinR>

```
getwd()

## [1] "C:/Users/dell/Desktop"

setwd("C:/Users/dell/Desktop")
#dir(pattern = ".csv")
memory.limit()

## [1] 1535

memory.size()

## [1] 18.04

rm(list = ls())
gc()

##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells 365542   9.8      592000 15.9    460000 12.3
## Vcells 372990   2.9     1023718  7.9     752284  5.8

library(car)
library(MASS)
data(Boston, package="MASS")
#?Boston
# crim
# per capita crime rate by town.
#
# zn
# proportion of residential land zoned for lots
# over 25,000 sq.ft.
#
# indus
# proportion of non-retail business acres per town.
#
# chas
# Charles River dummy variable (= 1 if tract bounds
# river; 0 otherwise).
#
```

```

# nox
# nitrogen oxides concentration (parts per 10 million).
#
# rm
# average number of rooms per dwelling.
#
# age
# proportion of owner-occupied units built prior
  to 1940.
#
# dis
# weighted mean of distances to five Boston employment
  centres.
#
# rad
# index of accessibility to radial highways.
#
# tax
# full-value property-tax rate per \$10,000.
#
# ptratio
# pupil-teacher ratio by town.
#
# black
#  $1000(B_k - 0.63)^2$  where  $B_k$  is the proportion of
  blacks by town.
#
# lstat
# lower status of the population (percent).
#
# medv
# median value of owner-occupied homes in \$1000s.
#
# Source
#

```

#Harrison, D. and Rubinfeld, D.L. (1978) Hedonic prices and the demand for clean air. *J. Environ. Economics and Management* 5, 81-102.

cor(Boston)

##	crim	zn	indus	chas	nox
## crim	1.00000000	-0.20046922	0.40658341	-0.055891582	0.42097171
## zn	-0.20046922	1.00000000	-0.53382819	-0.042696719	-0.51660371
## indus	0.40658341	-0.53382819	1.00000000	0.062938027	0.76365145
## chas	-0.05589158	-0.04269672	0.06293803	1.000000000	0.09120281
## nox	0.42097171	-0.51660371	0.76365145	0.091202807	1.00000000
## rm	-0.21924670	0.31199059	-0.39167585	0.091251225	-0.30218819
## age	0.35273425	-0.56953734	0.64477851	0.086517774	0.73147010
## dis	-0.37967009	0.66440822	-0.70802699	-0.099175780	-0.76923011
## rad	0.62550515	-0.31194783	0.59512927	-0.007368241	0.61144056
## tax	0.58276431	-0.31456332	0.72076018	-0.035586518	0.66802320
## ptratio	0.28994558	-0.39167855	0.38324756	-0.121515174	0.18893268
## black	-0.38506394	0.17552032	-0.35697654	0.048788485	-0.38005064
## lstat	0.45562148	-0.41299457	0.60379972	-0.053929298	0.59087892
## medv	-0.38830461	0.36044534	-0.48372516	0.175260177	-0.42732077
##	rm	age	dis	rad	tax
## crim	-0.21924670	0.35273425	-0.37967009	0.625505145	0.58276431
## zn	0.31199059	-0.56953734	0.66440822	-0.311947826	-0.31456332
## indus	-0.39167585	0.64477851	-0.70802699	0.595129275	0.72076018
## chas	0.09125123	0.08651777	-0.09917578	-0.007368241	-0.03558652
## nox	-0.30218819	0.73147010	-0.76923011	0.611440563	0.66802320
## rm	1.00000000	-0.24026493	0.20524621	-0.209846668	-0.29204783
## age	-0.24026493	1.00000000	-0.74788054	0.456022452	0.50645559
## dis	0.20524621	-0.74788054	1.00000000	-0.494587930	-0.53443158
## rad	-0.20984667	0.45602245	-0.49458793	1.000000000	0.91022819
## tax	-0.29204783	0.50645559	-0.53443158	0.910228189	1.00000000

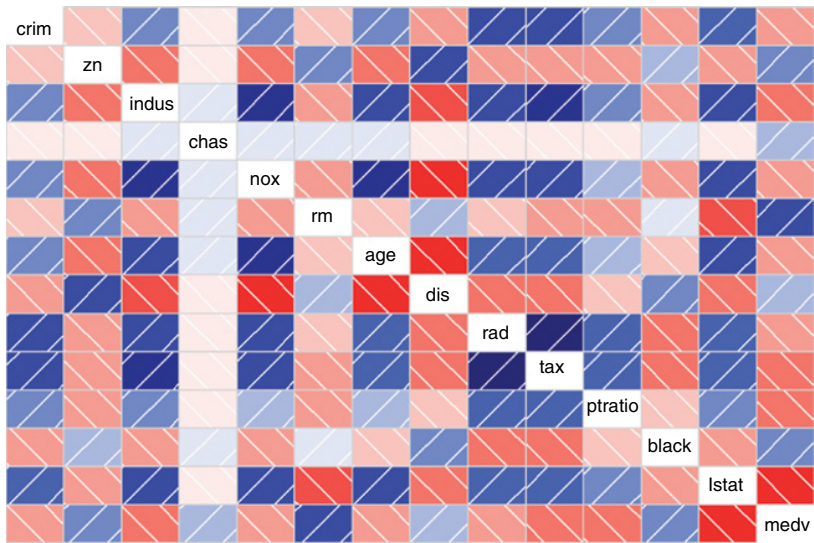
```
## ptratio -0.35550149  0.26151501 -0.23247054  0.464741179  0.46085304
## black    0.12806864 -0.27353398  0.29151167 -0.444412816 -0.44180801
## lstat    -0.61380827  0.60233853 -0.49699583  0.488676335  0.54399341
## medv     0.69535995 -0.37695457  0.24992873 -0.381626231 -0.46853593
##          ptratio      black      lstat      medv
## crim      0.2899456 -0.38506394  0.4556215 -0.3883046
## zn        -0.3916785  0.17552032 -0.4129946  0.3604453
## indus     0.3832476 -0.35697654  0.6037997 -0.4837252
## chas      -0.1215152  0.04878848 -0.0539293  0.1752602
## nox       0.1889327 -0.38005064  0.5908789 -0.4273208
## rm        -0.3555015  0.12806864 -0.6138083  0.6953599
## age       0.2615150 -0.27353398  0.6023385 -0.3769546
## dis       -0.2324705  0.29151167 -0.4969958  0.2499287
## rad       0.4647412 -0.44441282  0.4886763 -0.3816262
## tax       0.4608530 -0.44180801  0.5439934 -0.4685359
## ptratio   1.0000000 -0.17738330  0.3740443 -0.5077867
## black     -0.1773833  1.00000000 -0.3660869  0.3334608
## lstat     0.3740443 -0.36608690  1.0000000 -0.7376627
## medv      -0.5077867  0.33346082 -0.7376627  1.0000000
```

summary(Boston)

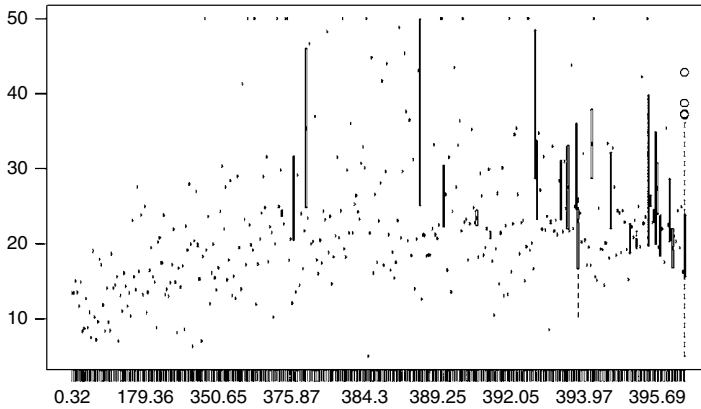
```
##          crim              zn              indus              chas
## Min.      : 0.00632   Min.      : 0.00   Min.      : 0.46   Min.      :0.00000
## 1st Qu.: 0.08204   1st Qu.: 0.00   1st Qu.: 5.19   1st Qu.:0.00000
## Median : 0.25651   Median : 0.00   Median : 9.69   Median :0.00000
## Mean      : 3.61352   Mean      : 11.36   Mean      :11.14   Mean      :0.06917
## 3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000
## Max.      :88.97620   Max.      :100.00   Max.      :27.74   Max.      :1.00000
```

##	nox	rm	age	dis
##	Min. :0.3850	Min. :3.561	Min. : 2.90	Min. : 1.130
##	1st Qu.:0.4490	1st Qu.:5.886	1st Qu.: 45.02	1st Qu.: 2.100
##	Median :0.5380	Median :6.208	Median : 77.50	Median : 3.207
##	Mean :0.5547	Mean :6.285	Mean : 68.57	Mean : 3.795
##	3rd Qu.:0.6240	3rd Qu.:6.623	3rd Qu.: 94.08	3rd Qu.: 5.188
##	Max. :0.8710	Max. :8.780	Max. :100.00	Max. :12.127
##	rad	tax	ptratio	black
##	Min. : 1.000	Min. :187.0	Min. :12.60	Min. : 0.32
##	1st Qu.: 4.000	1st Qu.:279.0	1st Qu.:17.40	1st Qu.:375.38
##	Median : 5.000	Median :330.0	Median :19.05	Median :391.44
##	Mean : 9.549	Mean :408.2	Mean :18.46	Mean :356.67
##	3rd Qu.:24.000	3rd Qu.:666.0	3rd Qu.:20.20	3rd Qu.:396.23
##	Max. :24.000	Max. :711.0	Max. :22.00	Max. :396.90
##	lstat	medv		
##	Min. : 1.73	Min. : 5.00		
##	1st Qu.: 6.95	1st Qu.:17.02		
##	Median :11.36	Median :21.20		
##	Mean :12.65	Mean :22.53		
##	3rd Qu.:16.95	3rd Qu.:25.00		
##	Max. :37.97	Max. :50.00		

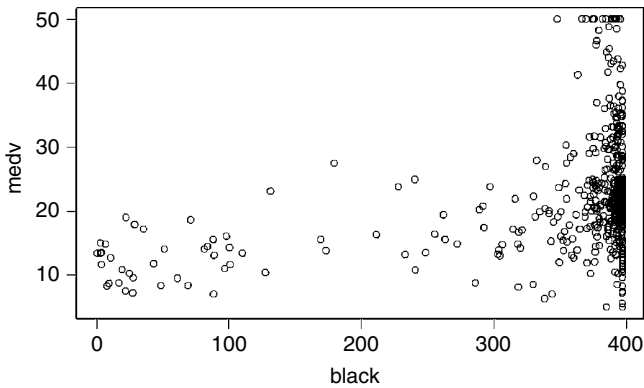
```
library(corrgram)
corrgram(Boston)
```



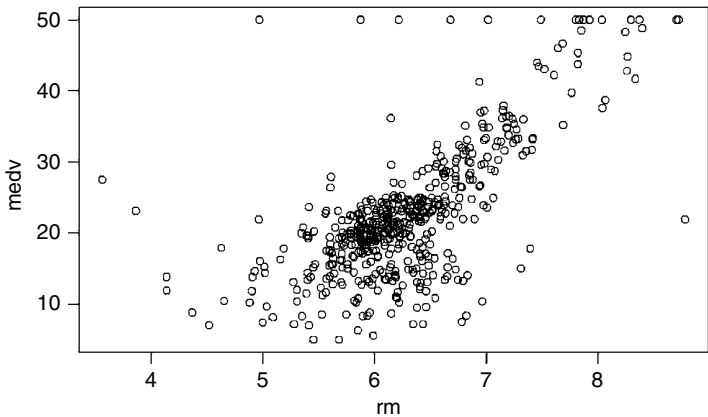
```
attach(Boston)
boxplot(medv~black)
```



```
plot (medv~black)
```



```
plot (medv~rm)
```



```
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
## $ crim      : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
## $ zn        : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
## $ indus     : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
## $ chas      : int   0 0 0 0 0 0 0 0 0 0 ...
## $ nox       : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
## $ rm        : num  6.58 6.42 7.18 7 7.15 ...
## $ age       : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
## $ dis       : num  4.09 4.97 4.97 6.06 6.06 ...
## $ rad       : int   1 2 2 3 3 3 5 5 5 5 ...
## $ tax       : num  296 242 242 222 222 222 311 311 311 311 ...
## $ ptratio   : num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
## $ black     : num  397 397 393 395 397 ...
## $ lstat     : num  4.98 9.14 4.03 2.94 5.33 ...
## $ medv      : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
RegModel.1 <-
```

```
  lm(medv~age+black+chas+crim+dis+indus+lstat+nox+ptratio+rad+rm+tax+zn,
     data=Boston)
```

```
summary(RegModel.1)
```

```
##
```

```
## Call:
```

```
## lm(formula = medv ~ age + black + chas + crim + dis + indus +
```

```
##      lstat + nox + ptratio + rad + rm + tax + zn, data = Boston)
```

```
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.595  -2.730  -0.518   1.777   26.199
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
## age          6.922e-04  1.321e-02   0.052 0.958229
## black        9.312e-03  2.686e-03   3.467 0.000573 ***
## chas         2.687e+00  8.616e-01   3.118 0.001925 **
## crim        -1.080e-01  3.286e-02  -3.287 0.001087 **
## dis         -1.476e+00  1.995e-01  -7.398 6.01e-13 ***
## indus        2.056e-02  6.150e-02   0.334 0.738288
## lstat       -5.248e-01  5.072e-02 -10.347 < 2e-16 ***
## nox         -1.777e+01  3.820e+00  -4.651 4.25e-06 ***
## ptratio     -9.527e-01  1.308e-01  -7.283 1.31e-12 ***
## rad          3.060e-01  6.635e-02   4.613 5.07e-06 ***
## rm          3.810e+00  4.179e-01   9.116 < 2e-16 ***
## tax         -1.233e-02  3.760e-03  -3.280 0.001112 **
## zn           4.642e-02  1.373e-02   3.382 0.000778 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.745 on 492 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7338
## F-statistic: 108.1 on 13 and 492 DF,  p-value: < 2.2e-16
```

```
vif(RegModel.1)
```

```
##      age      black      chas      crim      dis      indus      lstat      nox
## 3.100826 1.348521 1.073995 1.792192 3.955945 3.991596 2.941491 4.393720
## ptratio      rad      rm      tax      zn
## 1.799084 7.484496 1.933744 9.008554 2.298758
```

```
library(zoo, pos=15)
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##      as.Date, as.Date.numeric
```

```
library(lmtest, pos=15)
bptest(RegModel.1)
```

```
##
## studentized Breusch-Pagan test
##
## data: RegModel.1
## BP = 65.122, df = 13, p-value = 6.265e-09
```

```
RegModel.2 <- lm(medv~black+chas+crim+dis+lstat+nox+ptratio+rm+zn,
                 data=Boston)
summary(RegModel.2)
```

```
##
## Call:
## lm(formula = medv ~ black + chas + crim + dis + lstat + nox +
##      ptratio + rm + zn, data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.803  -2.832  -0.625   1.454   27.766
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  29.507997   4.872538   6.056 2.76e-09 ***
## black         0.008292   0.002688   3.084 0.002153 **
## chas         3.029924   0.868349   3.489 0.000527 ***
## crim        -0.061174   0.030377  -2.014 0.044567 *
## dis         -1.431665   0.188603  -7.591 1.59e-13 ***
## lstat       -0.525004   0.048351 -10.858 < 2e-16 ***
## nox        -16.088513   3.232702  -4.977 8.93e-07 ***
## ptratio     -0.838640   0.117342  -7.147 3.19e-12 ***
## rm          4.149667   0.407685  10.179 < 2e-16 ***
## zn          0.042032   0.013422   3.131 0.001842 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.833 on 496 degrees of freedom
## Multiple R-squared:  0.7288, Adjusted R-squared:  0.7239
## F-statistic: 148.1 on 9 and 496 DF, p-value: < 2.2e-16
```

```

vif(RegModel.2)
##      black      chas      crim      dis      lstat      nox      ptratio      rm
## 1.302455 1.051879 1.476281 3.410535 2.577927 3.034316 1.395503 1.774261
##      zn
## 2.119038

bptest(medv ~ black + chas + crim + dis + lstat + nox + ptratio + rm + zn,
       varformula = ~ fitted.values(RegModel.2), studentize=FALSE, data=Boston)
##
## Breusch-Pagan test
##
## data:  medv ~ black + chas + crim + dis + lstat + nox + ptratio + rm + zn
## BP = 8.817, df = 1, p-value = 0.002984

outlierTest(RegModel.2)

##      rstudent unadjusted p-value Bonferonni p
## 369 6.093117      2.2275e-09    1.1271e-06
## 372 5.574335      4.0893e-08    2.0692e-05
## 373 5.360117      1.2776e-07    6.4644e-05

Boston <- Boston[-c(369,372,373),]
RegModel.3 <- lm(medv~black+chas+crim+lstat+nox+ptratio+rm+zn, data=Boston)
summary(RegModel.3)

```

```
##
## Call:
## lm(formula = medv ~ black + chas + crim + lstat + nox + ptratio +
##      rm + zn, data = Boston)
##
## Residuals:
##      Min        1Q    Median        3Q        Max
## -16.6944  -2.6544  -0.6449   1.6392  21.4155
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  11.386058   4.150172   2.744  0.006300 **
## black         0.008849   0.002545   3.476  0.000553 ***
## chas         2.670557   0.833048   3.206  0.001434 **
## crim        -0.046149   0.028565  -1.616  0.106827
## lstat        -0.392872   0.046578  -8.435  3.68e-16 ***
## nox         -5.144940   2.539786  -2.026  0.043329 *
## ptratio      -0.958587   0.111396  -8.605  <2e-16 ***
## rm           5.318514   0.384910  13.818  <2e-16 ***
## zn          -0.010100   0.011040  -0.915  0.360736
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.578 on 494 degrees of freedom
## Multiple R-squared:  0.744, Adjusted R-squared:  0.7399
## F-statistic: 179.5 on 8 and 494 DF, p-value: < 2.2e-16
```

```
RegModel.4 <- lm(medv~black+lstat+pترatio+rm+zn, data=Boston)
summary(RegModel.4)

##
## Call:
## lm(formula = medv ~ black + lstat + pترatio + rm + zn, data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.2574  -2.8802  -0.6129   1.8640  22.9620
##
## Coefficients:
##      Estimate Std. Error t value Pr(>|t|)
## (Intercept)  9.253649   3.879508   2.385  0.0174 *
## black        0.011319   0.002462   4.598 5.43e-06 ***
## lstat       -0.453250   0.041643 -10.884 <2e-16 ***
## pترatio     -0.991397   0.108946  -9.100 <2e-16 ***
## rm          5.274984   0.386849  13.636 <2e-16 ***
## zn         -0.004863   0.010154  -0.479  0.6322
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.644 on 497 degrees of freedom
## Multiple R-squared:  0.735, Adjusted R-squared:  0.7323
## F-statistic: 275.7 on 5 and 497 DF,  p-value: < 2.2e-16

RegModel.5 <- lm(medv~black+lstat+pترatio+rm, data=Boston)
summary(RegModel.5)
```

```
##
## Call:
## lm(formula = medv ~ black + lstat + ptratio + rm, data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.1780  -2.8640  -0.6212   1.8545  23.0366
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   8.92877    3.81678   2.339   0.0197 *
## black          0.01130    0.00246   4.592 5.55e-06 ***
## lstat        -0.44870    0.04051 -11.075 <2e-16 ***
## ptratio      -0.97746    0.10491  -9.317 <2e-16 ***
## rm           5.26910    0.38635  13.638 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.64 on 498 degrees of freedom
## Multiple R-squared:  0.7348, Adjusted R-squared:  0.7327
## F-statistic: 345 on 4 and 498 DF, p-value: < 2.2e-16

vif(RegModel.5)

##      black      lstat  ptratio      rm
## 1.182491 1.954482 1.205162 1.715115
```

```

bptest(medv ~ black + lstat + ptratio + rm, varformula = ~
      fitted.values(RegModel.5), studentize=FALSE, data=Boston)

##
## Breusch-Pagan test
##
## data: medv ~ black + lstat + ptratio + rm
## BP = 0.21772, df = 1, p-value = 0.6408

#install.packages("gvlma")
#library(gvlma)
#Boston$medvbc=boxcox(Boston$medv)
#http://rstatistics.net/how-to-test-a-regression-model-for-heteroscedasticity-and-if-
#present-how-to-correct-it/

#Overfitting
a=nrow(Boston)
a

## [1] 503

b=round(0.7*a)
b

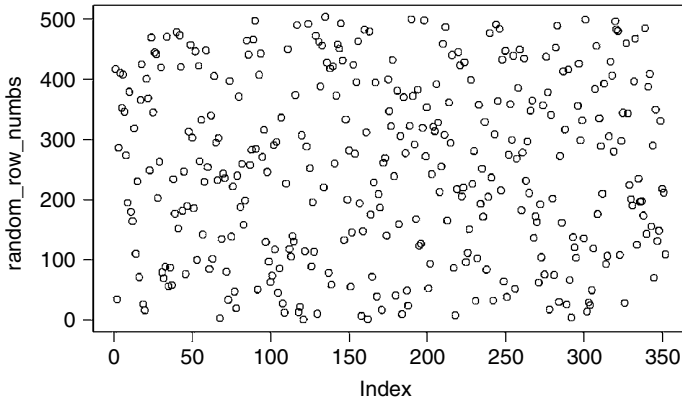
## [1] 352

random_row_numbs=sample(a,b,F)
random_row_numbs

```

##	[1]	417	35	286	410	352	408	346	274	195	379	180	165	318	110	231	72	366
##	[18]	425	27	16	401	368	249	469	345	444	442	203	263	419	80	70	89	470
##	[35]	57	88	58	234	177	478	152	473	420	182	247	77	190	313	457	303	186
##	[52]	446	100	422	264	333	142	230	448	254	85	340	102	405	295	233	302	4
##	[69]	135	243	236	81	34	397	139	222	48	20	240	371	188	259	158	199	464
##	[86]	441	258	283	466	497	284	51	407	443	271	316	130	246	98	64	74	291
##	[103]	117	296	46	86	336	28	13	227	450	118	106	140	131	374	490	14	23
##	[120]	307	1	115	288	492	252	90	196	114	472	11	462	388	456	221	503	427
##	[137]	79	418	59	421	260	373	458	451	493	431	133	334	200	282	56	146	424
##	[154]	276	395	463	194	7	148	482	312	2	479	175	73	229	394	40	209	187
##	[171]	17	261	269	141	400	347	322	432	239	41	381	159	306	10	370	277	49
##	[188]	24	323	500	372	292	168	383	124	127	319	498	272	353	53	94	242	321
##	[205]	314	392	328	213	255	459	308	486	166	361	294	440	87	8	217	445	423
##	[222]	206	220	428	97	112	151	399	226	281	32	103	358	193	251	172	329	84
##	[239]	205	477	237	33	309	491	364	484	216	433	65	447	39	275	359	300	439
##	[256]	52	268	385	449	183	278	435	232	297	211	348	365	137	173	163	63	192
##	[273]	105	357	76	437	378	18	341	202	75	452	489	31	273	162	413	317	26
##	[290]	416	67	5	138	121	104	356	426	299	332	136	499	15	30	25	50	119
##	[307]	384	455	176	335	289	210	393	93	107	305	429	406	280	496	483	480	108
##	[324]	298	344	29	460	343	225	201	191	396	468	125	235	197	198	174	485	143
##	[341]	387	409	156	290	71	350	132	149	331	218	212	109					

```
plot(random_row_nums)
```



```
Boston_train=Boston[random_row_nums,]
Boston_test=Boston[-random_row_nums,]
```

```
RegModel.6 <- lm(medv~black+lstat+ptratio+rm,
  data=Boston_train)
```

```
summary(RegModel.6)
```

```
##
## Call:
## lm(formula = medv ~ black + lstat + ptratio + rm,
## data = Boston_train)
##
## Residuals:
```

	Min	1Q	Median	3Q	Max
	-14.6989	-2.8976	-0.6286	1.8782	20.9866

```
##
## Coefficients:
```

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	12.082977	4.685939	2.579	0.010333	*
black	0.010551	0.002876	3.668	0.000283	***
lstat	-0.459176	0.049661	-9.246	< 2e-16	***
ptratio	-1.081483	0.130743	-8.272	2.86e-15	***
rm	5.130985	0.465497	11.023	< 2e-16	***

```
## ---
```

```

## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.731 on 347 degrees of freedom
## Multiple R-squared:  0.7336, Adjusted R-squared:  0.7305
## F-statistic: 238.8 on 4 and 347 DF,  p-value: < 2.2e-16

vif(RegModel.6)

##      black      lstat   ptratio      rm
## 1.181865 1.909594 1.266947 1.670012

outlierTest(RegModel.6)

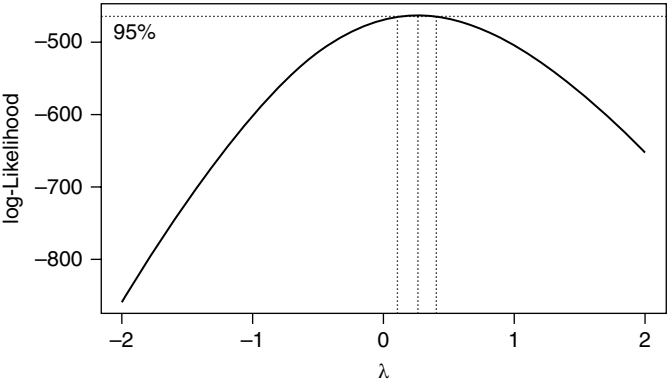
##      rstudent unadjusted p-value Bonferonni p
## 371 4.597485          6.0040e-06    0.0021134
## 413 4.320507          2.0369e-05    0.0071698
## 366 4.216016          3.1779e-05    0.0111860
## 368 4.012047          7.3806e-05    0.0259800

bptest(RegModel.6)

##
## studentized Breusch-Pagan test
##
## data:  RegModel.6
## BP = 1.9743, df = 4, p-value = 0.7405

```

```
#?  
dbc=boxcox (RegModel.6)
```



```
dbc  
## $x  
## [1] -2.00000000 -1.95959596 -1.91919192 -1.87878788 -1.83838384  
## [6] -1.79797980 -1.75757576 -1.71717172 -1.67676768 -1.63636364  
## [11] -1.59595960 -1.55555556 -1.51515152 -1.47474747 -1.43434343  
## [16] -1.39393939 -1.35353535 -1.31313131 -1.27272727 -1.23232323  
## [21] -1.19191919 -1.15151515 -1.11111111 -1.07070707 -1.03030303  
## [26] -0.98989899 -0.94949495 -0.90909091 -0.86868687 -0.82828283  
## [31] -0.78787879 -0.74747475 -0.70707071 -0.66666667 -0.62626263  
## [36] -0.58585859 -0.54545455 -0.50505051 -0.46464646 -0.42424242  
## [41] -0.38383838 -0.34343434 -0.30303030 -0.26262626 -0.22222222
```

```
## [46] -0.18181818 -0.14141414 -0.10101010 -0.06060606 -0.02020202
## [51] 0.02020202 0.06060606 0.10101010 0.14141414 0.18181818
## [56] 0.22222222 0.26262626 0.30303030 0.34343434 0.38383838
## [61] 0.42424242 0.46464646 0.50505051 0.54545455 0.58585859
## [66] 0.62626263 0.66666667 0.70707071 0.74747475 0.78787879
## [71] 0.82828283 0.86868687 0.90909091 0.94949495 0.98989899
## [76] 1.03030303 1.07070707 1.11111111 1.15151515 1.19191919
## [81] 1.23232323 1.27272727 1.31313131 1.35353535 1.39393939
## [86] 1.43434343 1.47474747 1.51515152 1.55555556 1.59595960
## [91] 1.63636364 1.67676768 1.71717172 1.75757576 1.79797980
## [96] 1.83838384 1.87878788 1.91919192 1.95959596 2.00000000
##
## $y
## [1] -859.1799 -847.1039 -835.1407 -823.2931 -811.5635 -799.9545 -788.4687
## [8] -777.1090 -765.8782 -754.7796 -743.8162 -732.9914 -722.3087 -711.7718
## [15] -701.3844 -691.1505 -681.0742 -671.1596 -661.4114 -651.8339 -642.4319
## [22] -633.2102 -624.1740 -615.3282 -606.6782 -598.2292 -589.9869 -581.9566
## [29] -574.1441 -566.5549 -559.1948 -552.0694 -545.1843 -538.5451 -532.1573
## [36] -526.0261 -520.1569 -514.5544 -509.2236 -504.1689 -499.3943 -494.9038
## [43] -490.7007 -486.7882 -483.1687 -479.8445 -476.8171 -474.0877 -471.6570
## [50] -469.5250 -467.6913 -466.1549 -464.9143 -463.9677 -463.3123 -462.9453
## [57] -462.8632 -463.0621 -463.5379 -464.2858 -465.3009 -466.5779 -468.1114
## [64] -469.8955 -471.9244 -474.1919 -476.6919 -479.4181 -482.3641 -485.5236
## [71] -488.8903 -492.4578 -496.2198 -500.1703 -504.3031 -508.6121 -513.0917
## [78] -517.7360 -522.5394 -527.4966 -532.6022 -537.8512 -543.2387 -548.7598
## [85] -554.4099 -560.1847 -566.0799 -572.0912 -578.2149 -584.4471 -590.7842
## [92] -597.2227 -603.7593 -610.3907 -617.1139 -623.9259 -630.8240 -637.8056
## [99] -644.8680 -652.0084
```

So to summarize what we should do for regression,

- First know about data and variables.
- Do Descriptive Statistics (summary) and a correlation matrix.
- Then run initial model.
- Remove outliers.
- Remove variables due to VIF (multicollinearity).
- Remove Heteroscedascity (advanced).
- Reduce variables and rerun, to maximize R^2 .
- Keep an eye on p-value for removing variables.

5.4 Logistic Regression in R and Python

Let us read some basics on logistic regression first. Logistic regression is used for predicting binary variables, and it is used a lot—whether a customer will default or not (FINANCIAL SERVICES DEFAULT), whether they will click on an internet ad or not (ECOMMERCE WEB ANALYTICS), whether they will buy a product or not (PROPENSITY), or whether they will leave a company for another one (CHURN) (<http://www.statmethods.net/advstats/glm.html>).

We use logit function from statsmodel for logistic regression.

```
import pandas as pd
import statsmodels.api as sm
import pylab as pl
import numpy as np

df = pd.read_csv("http://www.ats.ucla.edu/stat/data/
  binary.csv")
df.columns = ["admit", "gre", "gpa", "new1"]
df.head()
```

	admit	gre	gpa	new1
0	0	380	3.61	3
1	1	660	3.67	3
2	1	800	4.00	1
3	1	640	3.19	4
4	0	520	2.93	4

In [31]:

```
#create dummy variables
```

Note this step creates dummy numeric variables from a categoric variable

```
dummy_ranks = pd.get_dummies(df['new1'], prefix='new')
cols_to_keep = ['admit', 'gre', 'gpa']
print (dummy_ranks.head())
```

	new_1	new_2	new_3	new_4
0	0.0	0.0	1.0	0.0
1	0.0	0.0	1.0	0.0
2	1.0	0.0	0.0	0.0
3	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	1.0

```
cols_to_keep = ['admit', 'gre', 'gpa']
data = df[cols_to_keep].join(dummy_ranks.ix[:, 'new_2':])
data['intercept'] = 1.0
train_cols = data.columns[1:]

# Index([gre, gpa, prestige_2, prestige_3,
#        prestige_4], dtype=object)

logit = sm.Logit(data['admit'], data[train_cols])

# fit the model

result = logit.fit()

Optimization terminated successfully.
Current function value: 0.573147
Iterations 6
```

In [40]:

```
print (result.summary())
```

Logit Regression Results						
=====						
Dep. Variable:	admit		No. Observations:		400	
Model:	Logit		Df Residuals:		394	
Method:	MLE		Df Model:		5	
Date:	Mon, 13 Mar 2017		Pseudo R-squ.:		0.08292	
Time:	18:31:59		Log-Likelihood:		-229.26	
converged:	True		LL-Null:		-249.99	
			LLR p-value:		7.578e-08	
=====						
	coef	std err	z	P> z	[95.0% Conf. Int.]	

gre	0.0023	0.001	2.070	0.038	0.000	0.004
gpa	0.8040	0.332	2.423	0.015	0.154	1.454
new_2	-0.6754	0.316	-2.134	0.033	-1.296	-0.055
new_3	-1.3402	0.345	-3.881	0.000	-2.017	-0.663
new_4	-1.5515	0.418	-3.713	0.000	-2.370	-0.733
intercept	-3.9900	1.140	-3.500	0.000	-6.224	-1.756
=====						

The aforementioned code was referred from <http://blog.yhat.com/posts/logistic-regression-python-rodeo.html>
Let's do the same data in R (<http://rpubs.com/newajay/uclaglm>).

```
library(aod)

## Warning: package 'aod' was built under R version 3.3.3

library(ggplot2)
library(Rcpp)
mydata <- read.csv("http://www.ats.ucla.edu/stat/data/binary.csv")
head(mydata)

##   admit gre  gpa rank
## 1     0 380 3.61    3
## 2     1 660 3.67    3
## 3     1 800 4.00    1
## 4     1 640 3.19    4
## 5     0 520 2.93    4
## 6     1 760 3.00    2

summary(mydata)

##      admit      gre      gpa      rank
## Min.   :0.0000  Min.   :220.0  Min.   :2.260  Min.   :1.000
## 1st Qu.:0.0000  1st Qu.:520.0  1st Qu.:3.130  1st Qu.:2.000
## Median :0.0000  Median :580.0  Median :3.395  Median :2.000
## Mean   :0.3175  Mean   :587.7  Mean   :3.390  Mean   :2.485
## 3rd Qu.:1.0000  3rd Qu.:660.0  3rd Qu.:3.670  3rd Qu.:3.000
## Max.   :1.0000  Max.   :800.0  Max.   :4.000  Max.   :4.000
sapply(mydata, sd)
```

```
##          admit          gre          gpa          rank
##    0.4660867 115.5165364    0.3805668    0.9444602
```

```
xtabs(~ admit + rank, data = mydata)
```

```
##          rank
## admit  1  2  3  4
##      0 28 97 93 55
##      1 33 54 28 12
```

```
mydata$rank <- factor(mydata$rank)
mylogit <- glm(admit ~ gre + gpa + rank, data = mydata, family = "binomial")
summary(mylogit)
```

```
##
## Call:
## glm(formula = admit ~ gre + gpa + rank, family = "binomial",
##      data = mydata)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.6268   -0.8662   -0.6388    1.1490    2.0790
##
```

```

## Coefficients:
##           Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.989979   1.139951  -3.500 0.000465 ***
## gre          0.002264   0.001094   2.070 0.038465 *
## gpa          0.804038   0.331819   2.423 0.015388 *
## rank2        -0.675443   0.316490  -2.134 0.032829 *
## rank3        -1.340204   0.345306  -3.881 0.000104 ***
## rank4        -1.551464   0.417832  -3.713 0.000205 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 499.98  on 399  degrees of freedom
## Residual deviance: 458.52  on 394  degrees of freedom
## AIC: 470.52
##
## Number of Fisher Scoring iterations: 4

```

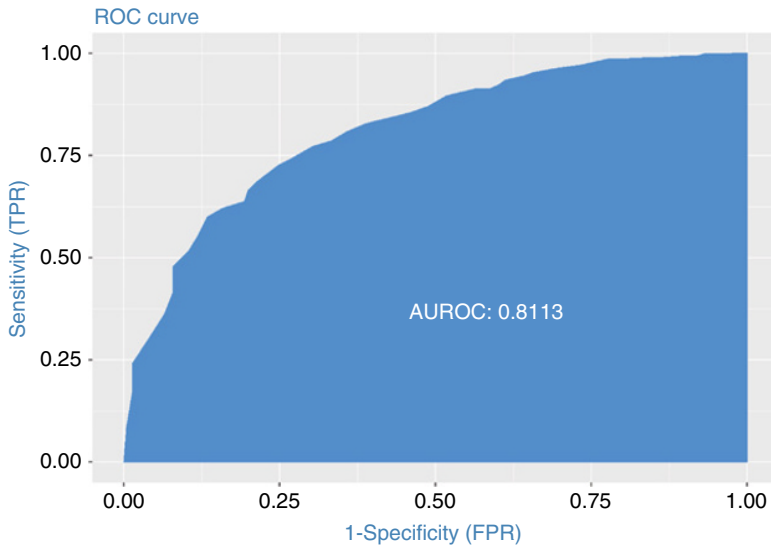
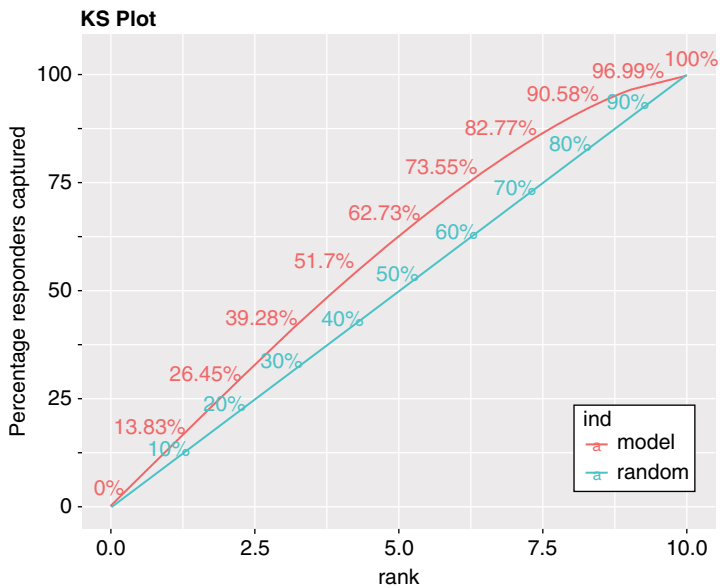


Figure 5.17 The ROC curve.

In R we can do it using `glm` (see <http://rpubs.com/newajay/logisticregression>, Figure 5.17). Some terms that are introduced are area under a curve, confusion matrix, and KS distance.

```
ks_plot(actuals=Training$Class, predictedScores=as.numeric(fitted(fitD)))
```



```
ks_stat(actuals=Training$Class,predictedScores=as.numeric(fitted(fitD)))
## [1] 0.4718
```

5.4.1 Additional Concepts

Odds ratio = $p/1 - p$ where p is probability of success

Logit = $\log(\text{odds ratio})$

Overfitting—It occurs when the model is closer to sample data than real data and due to excessive noise. It is avoided by splitting the data into test and training and then building the model on one part of data.

5.4.2 ROC Curve and AUC

The ROC is a curve generated by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings, while the AUC is the area under the ROC curve. As a rule of thumb, a model with good predictive ability should have an AUC closer to 1 (1 is ideal) than to 0.5.

Confusion matrix helps determine classifier. It is a matrix of predicted versus actual.

	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

A **confusion matrix**, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm.

Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class (or vice versa) (Figure 5.18).

An additional example of R based modeling is at http://rpubs.com/newajay/titanic_kaggle and <http://rpubs.com/ajaydecis/logisticmodels>

5.4.3 Bias Versus Variance

Lastly a modeler should be careful of errors due to bias or variance.

Error Due to Bias

The error due to bias is taken as the difference between the expected (or average) prediction of our model and the correct value, which we are trying to predict. Of course you only have one model so talking about expected or average

		Predicted condition			
Total population		Predicted condition positive	Predicted condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	
True condition	Condition positive	True positive	False negative (type II error)	True positive rate (TPR), sensitivity, recall = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False negative rate (FNR), miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$
	Condition negative	False positive (type I error)	True negative	False positive rate (FPR), fall-out = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	True negative rate (TNR), specificity (SPC) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$
Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$		Positive predictive value (PPV), precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR-}}$
		False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Test outcome positive}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Test outcome negative}}$	Negative likelihood ratio (LR-) = $\frac{\text{FNR}}{\text{TNR}}$	

Figure 5.18 Confusion matrix.

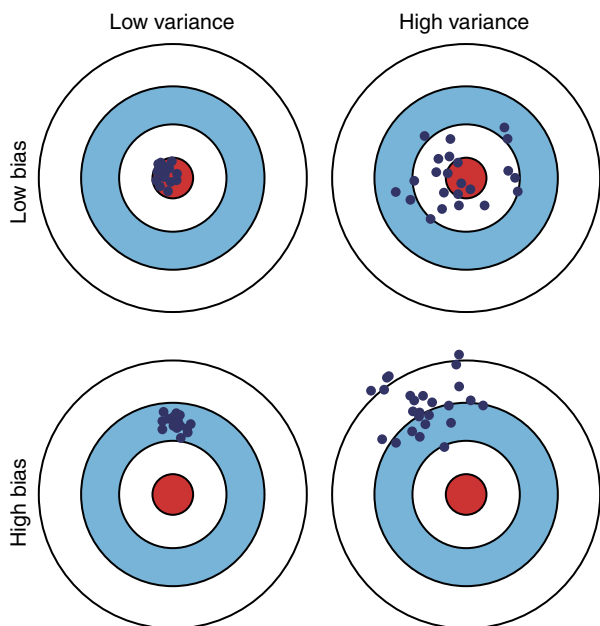


Figure 5.19 Bias and variance. <http://scott.fortmann-roe.com/docs/BiasVariance.html>.
Source: Scott Fortmann-Roe. © CSS from Substance.io.

prediction values might seem a little strange. However, imagine you could repeat the whole model building process more than once: each time you gather new data and run a new analysis creating a new model. Due to randomness in the underlying datasets, the resulting models will have a range of predictions. Bias measures how far off in general these models' predictions are from the correct value.

Error Due to Variance

The error due to variance is taken as the variability of a model prediction for a given data point. Again, imagine you can repeat the entire model building process multiple times. The variance is how much the predictions for a given point vary between different realizations of the model (Figure 5.19).

References

- Stephen R. Johnson (2008). The Trouble with QSAR (or How I Learned To Stop Worrying and Embrace Fallacy). *Journal of Chemical Information and Modeling*, 48(1), 25–26. 10.1021/ci700332k
- Deborah J. Rumsey. *Statistics for Dummies*. Standard Normal Distribution. John Wiley & Sons, Inc., Hoboken, 2016.

6

Data Visualization

To dos: plot.ly, bokeh, Shiny, Googlevis

6.1 Concepts on Data Visualization

Data visualization is the presentation of data in a pictorial or graphical format to understand information more easily and quickly. Effective visualization helps users in analyzing and reasoning about data and evidence. It makes complex data more accessible, understandable, and usable. It is more than just impressive-looking graphs because it helps to understand data much better than a tabular nonvisual form would do. A good course to learn data visualization would be at <https://www.coursera.org/learn/datavisualization>

6.1.1 History of Data Visualization

William Playfair is credited with inventing many of the graphs associated with modern data visualization such as the line, area, and bar charts of economic data, pie chart, and circle graph.

The work of Charles Minard is said to have greatly influenced the field of data visualization. His famous chart, Napoleon's march shows the death and decline of the French Grande Armée in the war against Russia. The graphic is notable for its representation in two dimensions of six types of data: the number of Napoleon's troops, distance, temperature, latitude and longitude, direction of travel, and location relative to specific dates. It is thus an early example of an information graphic (Figure 6.1).

Florence Nightingale did similar pioneering work in data visualization in representing deaths due to various diseases during the Crimean War with her coxcomb graphs (Figure 6.2). The following is cited in <http://understandinguncertainty.org/coxcombs>

An early example of how spatial visualization can greatly aid decision-making is by Jon Snow (not from the Game of Thrones!) whose cholera

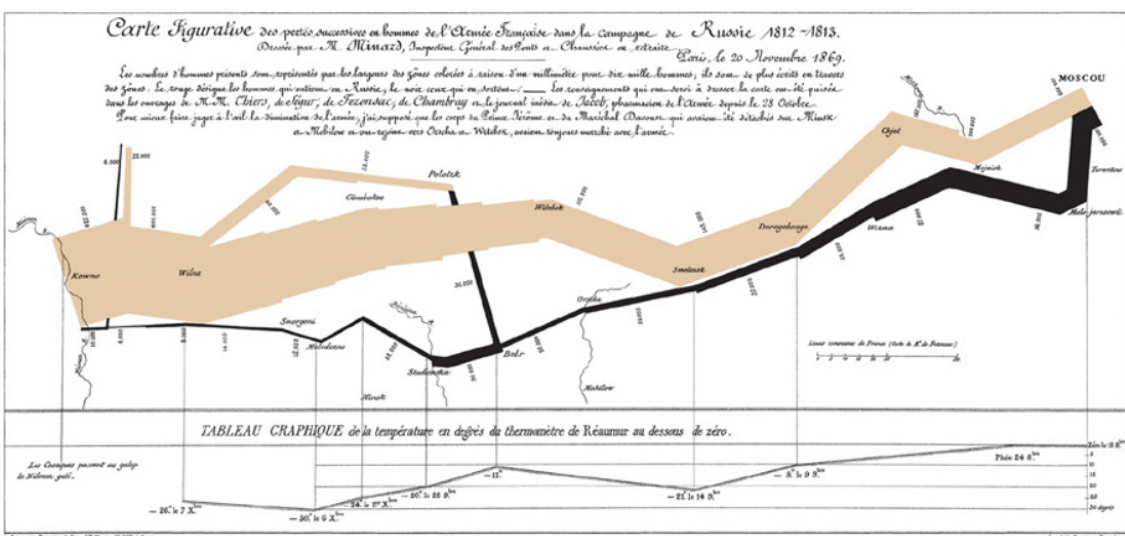


Figure 6.1 Minard's graph for Napoleon's march. *Source:* © University of Cambridge.

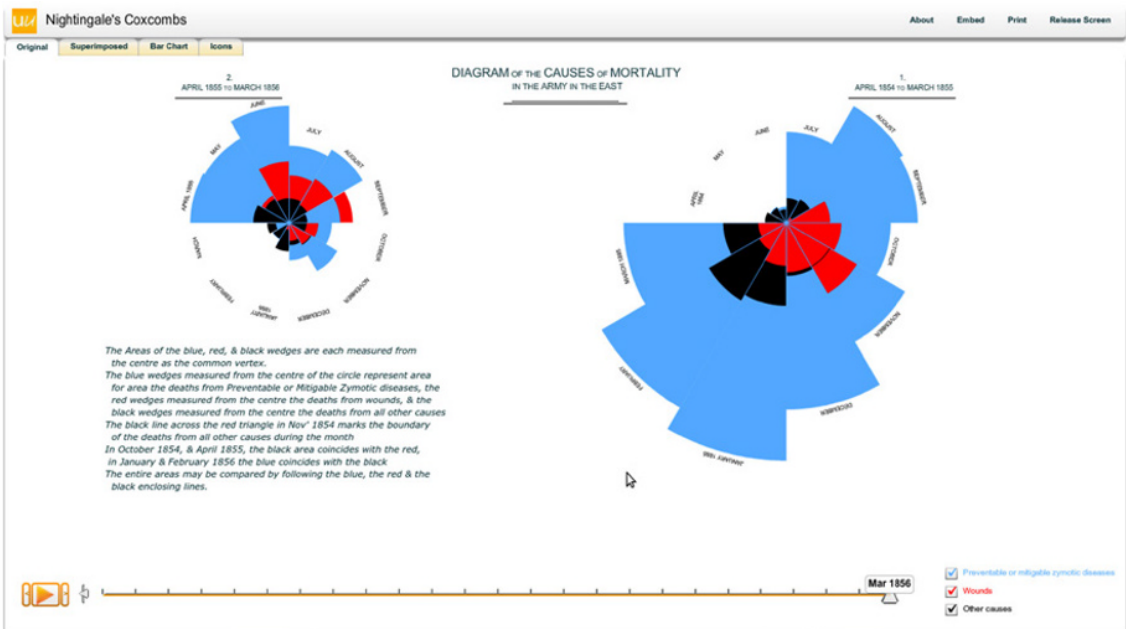


Figure 6.2 Florence Nightangle Coxcomb charts.



Figure 6.3 Jon Snow Cholera Outbreak Map.

outbreak graph helped pinpoint the cause to a single water pump. It is regarded as the founding event of the science of epidemiology (Figure 6.3).

6.1.2 Anscombe Case Study

“Lies, damned lies, and statistics” is a phrase describing the sometimes misleading but powerful power of numbers, particularly the use of statistics to bolster a weak or untenable argument.

The Anscombe case study shows how misleading conclusions from identical summary statistics (e.g., mean, standard deviation, and correlation) can be corrected only when we visualize data (revealing dissimilar data graphics (scatterplots)). Source: Anscombe (1973) <http://www.sjsu.edu/faculty/gerstman/StatPrimer/anscombe1973.pdf>

We recreate the case study in Python using the R Dataset. A copy of the code is at the author’s github (<https://github.com/decisionstats/pythonfordata-science>) and at <http://nbviewer.jupyter.org/gist/decisionstats/3737642751895f470d5c07194302f53e>

We read the data using pandas, find the mean and standard deviations through numpy, use regression using statsmodel package, and finally visualize using the ggplot package.

6.1.3 Importing Packages

```
import pandas as pd
import statsmodels.formula.api as sm
import numpy as np
import ggplot as gg
```

Reading the Dataset

```
anscombe=pd.read_csv("https://vincentarelbundock.
github.io/Rdatasets/csv/datasets/anscombe.csv")
anscombe
```

	Unnamed: 0	x1	x2	x3	x4	y1	y2	y3	y4
0	1	10	10	10	8	8.04	9.14	7.46	6.58
1	2	8	8	8	8	6.95	8.14	6.77	5.76
2	3	13	13	13	8	7.58	8.74	12.74	7.71
3	4	9	9	9	8	8.81	8.77	7.11	8.84
4	5	11	11	11	8	8.33	9.26	7.81	8.47
5	6	14	14	14	8	9.96	8.10	8.84	7.04
6	7	6	6	6	8	7.24	6.13	6.08	5.25
7	8	4	4	4	19	4.26	3.10	5.39	12.50
8	9	12	12	12	8	10.84	9.13	8.15	5.56
9	10	7	7	7	8	4.82	7.26	6.42	7.91
10	11	5	5	5	8	5.68	4.74	5.73	6.89

Dropping the Column

```
anscombe=anscombe.drop('Unnamed: 0', 1)
```

The Anscombe Quartet

```
anscombe
```

	x1	x2	x3	x4	y1	y2	y3	y4
0	10	10	10	8	8.04	9.14	7.46	6.58
1	8	8	8	8	6.95	8.14	6.77	5.76
2	13	13	13	8	7.58	8.74	12.74	7.71
3	9	9	9	8	8.81	8.77	7.11	8.84
4	11	11	11	8	8.33	9.26	7.81	8.47
5	14	14	14	8	9.96	8.10	8.84	7.04
6	6	6	6	8	7.24	6.13	6.08	5.25
7	4	4	4	19	4.26	3.10	5.39	12.50
8	12	12	12	8	10.84	9.13	8.15	5.56
9	7	7	7	8	4.82	7.26	6.42	7.91
10	5	5	5	8	5.68	4.74	5.73	6.89

6.1.4 Taking Means and Standard Deviations

```
np.mean(anscombe)
x1      9.000000
x2      9.000000
x3      9.000000
x4      9.000000
y1      7.500909
y2      7.500909
y3      7.500000
y4      7.500909
dtype: float64
```

```
np.std(anscombe)
x1      3.162278
x2      3.162278
x3      3.162278
x4      3.162278
y1      1.937024
y2      1.937109
y3      1.935933
y4      1.936081
dtype: float64
```

Fitting Regression Line between Respective X and Y

```
result1 = sm.ols(formula="y1 ~ x1 ", data=anscombe).fit()
result1.summary()
```

```
/home/ajay/anaconda3/lib/python3.4/site-packages/
scipy/stats/stats.py:1285: UserWarning: kurtosistest
only valid for n>=20 ... continuing anyway, n=11
"anyway, n=%i" % int(n))
```

Dep. Variable:	y1	R-squared:	0.667
Model:	OLS	Adj. R-squared:	0.629
Method:	Least Squares	F-statistic:	17.99
Date:	Thu, 07 Jul 2016	Prob (F-statistic):	0.00217
Time:	04:32:15	Log-Likelihood:	-16.841
No. Observations:	11	AIC:	37.68
Df Residuals:	9	BIC:	38.48
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[95.0% Conf. Int.]
Intercept	3.0001	1.125	2.667	0.026	0.456 5.544
x1	0.5001	0.118	4.241	0.002	0.233 0.767

Omnibus:	0.082	Durbin-Watson:	3.212
Prob(Omnibus):	0.960	Jarque-Bera (JB):	0.289
Skew:	-0.122	Prob(JB):	0.865
Kurtosis:	2.244	Cond. No.	29.1

```

result1.params
Intercept  3.000091
x1         0.500091
dtype: float64
result1.rsquared

0.66654245950877489
result2 = sm.ols(formula="y2 ~ x2 ", data=anscombe).fit()
result3 = sm.ols(formula="y3 ~ x3 ", data=anscombe).fit()
result4 = sm.ols(formula="y4 ~ x4 ", data=anscombe).fit()

print(result1.params)
print(result2.params)
print(result3.params)
print(result4.params)

Intercept  3.000091
x1         0.500091
dtype: float64
Intercept  3.000909
x2         0.500000
dtype: float64
Intercept  3.002455
x3         0.499727
dtype: float64
Intercept  3.001727
x4         0.499909
dtype: float64

print(result1.rsquared)
print(result2.rsquared)
print(result3.rsquared)
print(result4.rsquared)

0.666542459509
0.666242033727
0.666324041067
0.666707256898

```

```
print(np.mean(anscombe))
```

```
x1      9.000000
x2      9.000000
x3      9.000000
x4      9.000000
y1      7.500909
y2      7.500909
y3      7.500000
y4      7.500909
dtype: float64
```

```
print(np.std(anscombe))
```

```
x1      3.162278
x2      3.162278
x3      3.162278
x4      3.162278
y1      1.937024
y2      1.937109
y3      1.935933
y4      1.936081
dtype: float64
```

6.1.5 Conclusion

It seems that X and Y have the same means, same standard deviations, same regression parameters, and same R-squared value (up to two decimal places). So as per summary statistics, the data between all four quartets (X1 Y1, X2 Y2, X3 Y3, and X4 Y4) is the same. A copy of this tutorial is available at <http://nbviewer.jupyter.org/gist/decisionstats/3737642751895f470d5c07194302f53e> #Data-Visualization

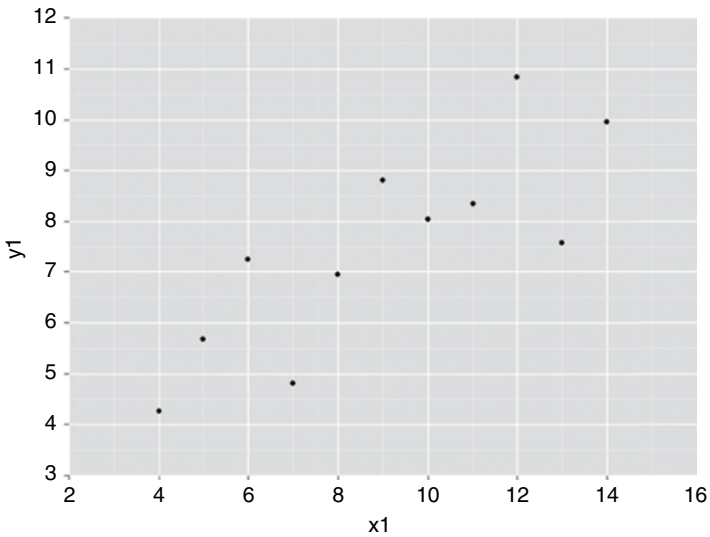
6.1.6 Data Visualization

```
%matplotlib inline
```

```
p = gg.ggplot(gg.aes(x='x1', y='y1'), data=anscombe)
p + gg.geom_point()
```

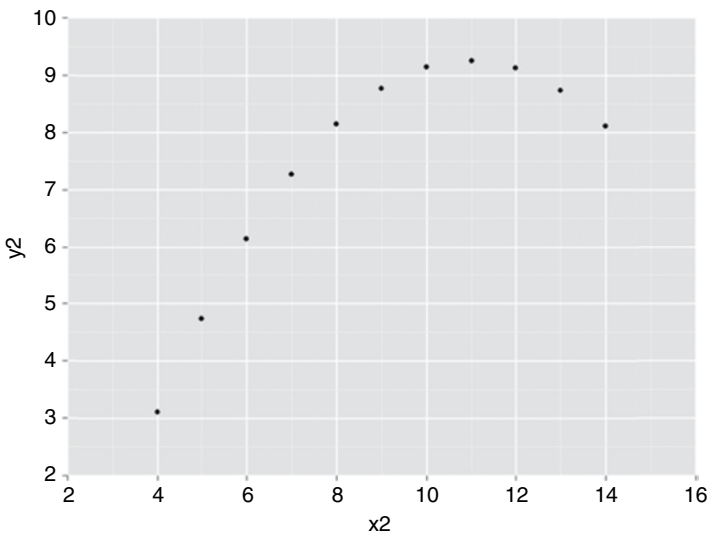
```
/home/ajay/anaconda3/lib/python3.4/site-packages/
matplotlib/__init__.py:872: UserWarning: axes.color_
cycle is deprecated and replaced with axes.prop_cycle;
please use the latter.
```

```
warnings.warn(self.msg_depr % (key, alt_key))
```



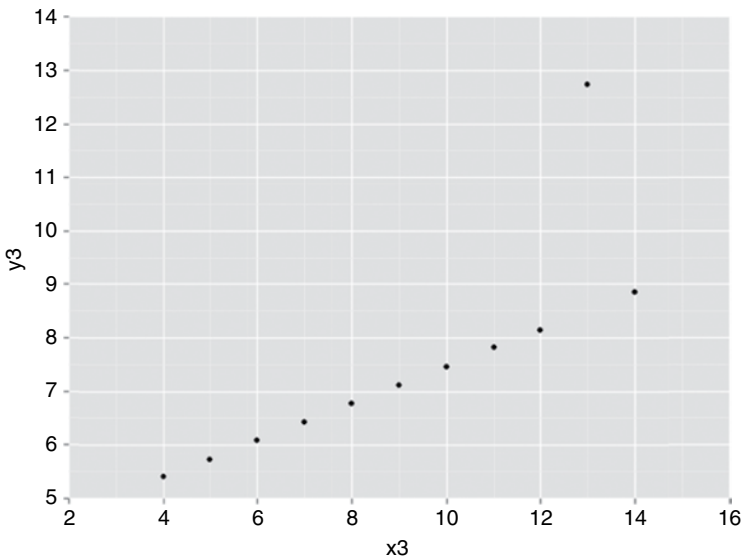
```
<ggplot: (-901764730)>
p2 = gg.ggplot(gg.aes(x='x2', y='y2'), data=anscombe)
p2 + gg.geom_point()

/home/ajay/anaconda3/lib/python3.4/site-packages/
matplotlib/__init__.py:872: UserWarning: axes.color_
cycle is deprecated and replaced with axes.prop_cycle;
please use the latter.
  warnings.warn(self.msg_depr % (key, alt_key))
```



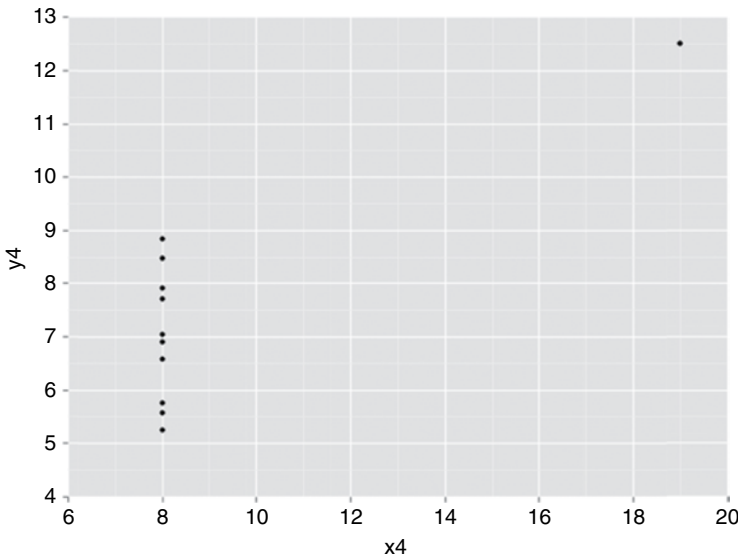
```
<ggplot: (-901793152)>
p3 = gg.ggplot(gg.aes(x='x3', y='y3'), data=anscombe)
p3 + gg.geom_point()

/home/ajay/anaconda3/lib/python3.4/site-packages/
matplotlib/__init__.py:872: UserWarning: axes.color_
cycle is deprecated and replaced with axes.prop_cycle;
please use the latter.
  warnings.warn(self.msg_depr % (key, alt_key))
```



```
<ggplot: (-901915866)>
p4=gg.ggplot(gg.aes(x='x4', y='y4'), data=anscombe)
p4+gg.geom_point()

/home/ajay/anaconda3/lib/python3.4/site-packages/
matplotlib/__init__.py:872: UserWarning: axes.color_
cycle is deprecated and replaced with axes.prop_cycle;
please use the latter.
  warnings.warn(self.msg_depr % (key, alt_key))
<ggplot: (-901651556)>
```



6.1.7 Conclusion

The graphs show that the four quartets are completely different even though summary statistics (means, deviations, regression) was showing identical result.

The first quartet (x_1, y_1) shows a scattered relationship.

The second quartet shows a curved polynomial relationship.

The third shows a straight line with one outlier.

The fourth shows a constant value of x and one outlier.

So we are better off relying on data visualization as an additional step to verify summary statistics or exploratory data analysis (but note we should rely on both, not just data visualization alone as many dashboards tend to do).

6.2 Tufte's Work on Data Visualization

Edward Tufte is known in some circles as the father of modern data visualization. Some of his seminal principles for data visualization are the following:

- 1) The representation of numbers, as physically measured on the surface of the graph itself, should be directly proportional to the numerical quantities represented.

- 2) Clear, detailed, and thorough labeling should be used to defeat graphical distortion and ambiguity. Write out explanations of the data on the graph itself. Label important events in the data.
- 3) Show **data variation**, not **design variation**.
- 4) In time-series displays of money, deflated and standardized units of monetary measurement are nearly always better than nominal units.
- 5) The number of information carrying (variable) dimensions depicted should not exceed the number of dimensions in the data. Graphics must not quote data out of context.

To the data scientist, Tufte shows a set of simple and easy to follow directives:

- 1) Above all else show data.
- 2) Maximize the data-ink ratio.
- 3) Erase non-data-ink.
- 4) Erase redundant data-ink.
- 5) Revise and edit.

6.3 Stephen Few on Dashboard Design

Stephen Few is the acknowledged master for designing better dashboards that show how enterprises visualize their business data. There are three key questions for a dashboard:

- 1) Who is my audience?
- 2) What value will the dashboard add?
- 3) What type of dashboard am I creating?

In his paper, “Common Pitfalls on Dashboard Design,” Few lists the common mistakes when designers build dashboards. These can and should be used as a checklist for data scientists or designers of new dashboards. For the full paper the reader is advised to read it at http://www.perceptualedge.com/articles/Whitepapers/Common_Pitfalls.pdf

- Exceeding the boundaries of a single screen
- Supplying inadequate context for the data
- Displaying excessive detail or precision
- Expressing measures indirectly
- Choosing inappropriate media of display
- Introducing meaningless variety
- Using poorly designed display media

- Encoding quantitative data inaccurately
- Arranging the data poorly
- Ineffectively highlighting what's important
- Cluttering the screen with useless decoration
- Misusing or overusing color
- Designing an unappealing visual display

Stephen Few also gives advice that is very practical to people building data science teams.

The advice is as follows:

When you need more computing power, there are three potential choices:

- 1) Replace your computer with one that's more powerful.
- 2) Add more computers.
- 3) Upgrade the computer that you have to make it more powerful.

When you need more human power, what are your choices?

- Replace the employee with one who's more productive.
- Add more people.
- Help your employee upgrade skills to make him more productive.

6.3.1 Maeda on Design

John Maeda created the laws of simplicity to help designers create better interfaces. While a data scientist is typically analyzing data created by different interfaces (e.g., an experiment on web sites), a knowledge of design can help them better and provide them more useful advice to design counterparts.

The laws of simplicity are taken from a small 100-page book called *The Laws of Simplicity*:

- Reduce—the simplest way to achieve simplicity is through thoughtful reduction.
- Organize—organization makes a system of many appear fewer.
- Time—savings in time feel like simplicity.
- Learn—knowledge makes everything simpler.
- Differences—simplicity and complexity need each other.
- Context—what lies in the periphery of simplicity is definitely not peripheral.
- Emotion—more emotions are better than less.
- Trust—in simplicity we trust.
- Failure—some things can never be made simple.
- The one—simplicity is about subtracting the obvious and adding the meaningful.

Three keys:

- 1) Away—more appears like less by simply moving it far, far away.
- 2) Open—openness simplifies complexity.
- 3) Power—use less, gain more.

6.4 Basic Plots

These are some of the basic plots in Python.

The basic packages for data visualization in Python Data Science (PyData) are matplotlib, seaborn, ggplot, and bokeh. We import the packages as

```
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
```

An online version of this tutorial is available at <http://nbviewer.jupyter.org/gist/decisionstats/e9fd40890553b24acda5e07654bceaa8>

To make sure graphs remain in same window of our Jupyter notebook, we use the following line.

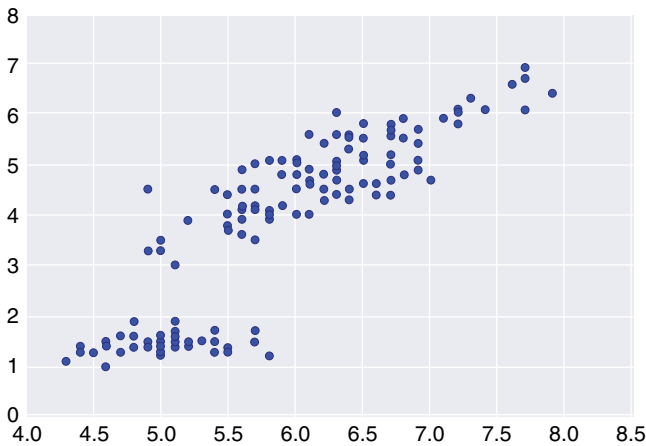
```
%matplotlib inline
```

Let's take the Iris Dataset from R using the code below. The following will plot a scatterplot. Simply put—a scatterplot plots the data in points

```
iris =pd.read_csv("https://vincentarelbundock.github.io/Rdatasets/csv/datasets/iris.csv ")
iris=iris.drop('Unnamed: 0', 1)
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
Sepal.Length    150 non-null float64
Sepal.Width     150 non-null float64
Petal.Length    150 non-null float64
Petal.Width     150 non-null float64
Species         150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

```
plt.scatter(x="Sepal.Length",y="Petal.Length",
data=iris);
```

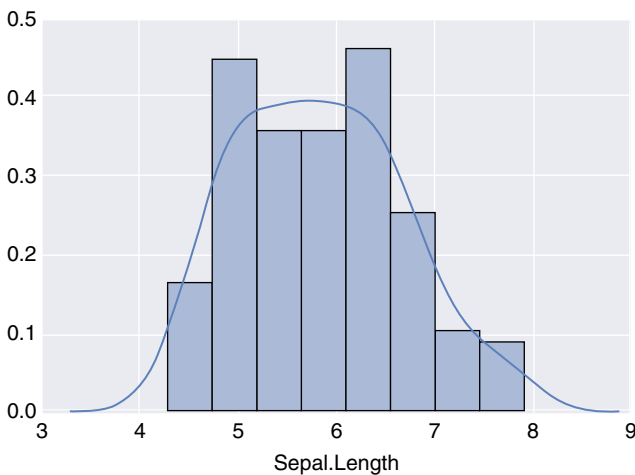


Distribution plot—To plot the distribution, I can use a `distplot` from `seaborn`, while to add a regression line I can use `regplot`.

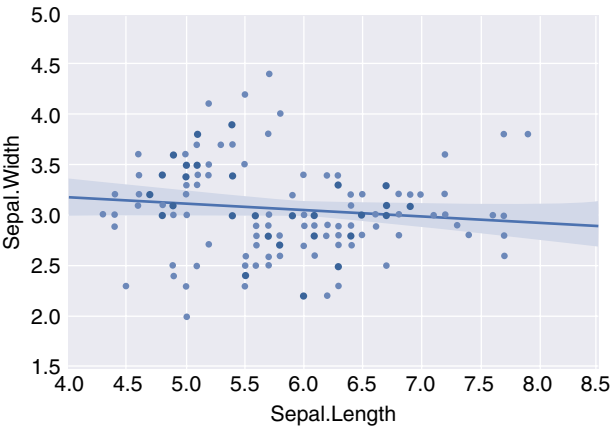
```
sns.distplot(iris["Sepal.Length"])
```

```
/home/ajayohri/anaconda3/lib/python3.5/site-packages/
statsmodels/nonparametric/kdetools.py:20: Visible
DeprecationWarning: using a non-integer number instead of
an integer will result in an error in the future
```

```
y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
<matplotlib.axes._subplots.AxesSubplot at
0x7f0302e4e278>
```



```
sns.regplot(x="Sepal.Length", y="Sepal.Width",
data=iris);
```



You can use swarmplot from seaborn to do a scatterplot for multiple categories.

```
plt.figure(figsize=(8, 6))
sns.swarmplot(x="Sepal.Length", y="Sepal.Width",
hue="Species",data=iris)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f0302d57438>

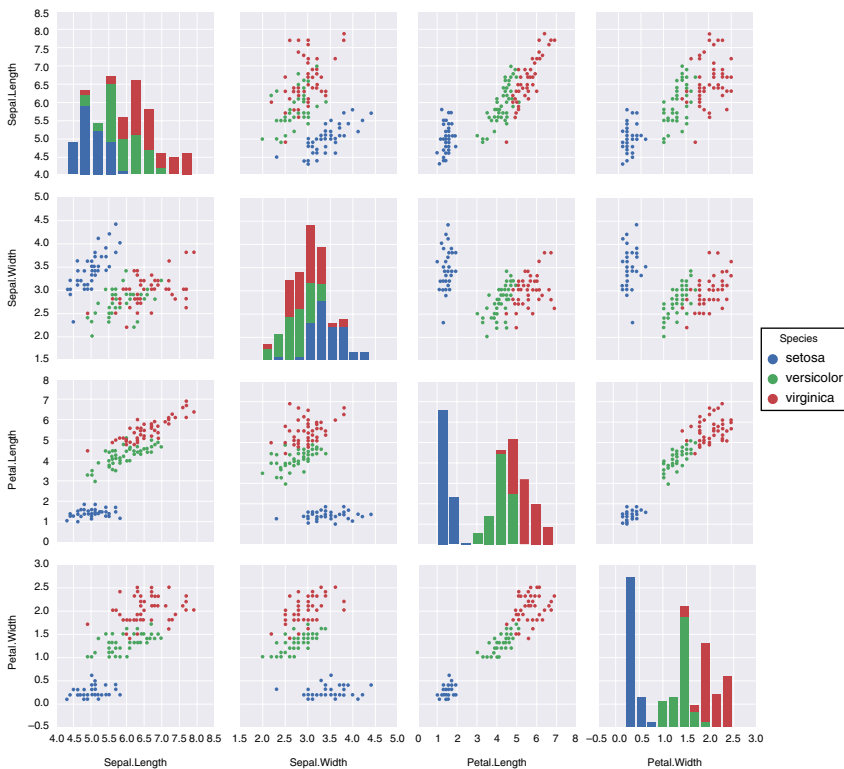


A **swarmplot** can help us visualize multiple categories of scatterplots (it draws a categorical scatterplot with non-overlapping points), while a **pairplot** can help us with plotting entire data frame (by plotting the pairwise relationships in the entire dataset).

NOTE: We can modify the size of the figure by the parameter **plt.figure(figsize=(A,B))** with the hue parameter as a step to modify color and make the graphic more coherent or easy to understand.

```
sns.pairplot(iris, hue="Species")
```

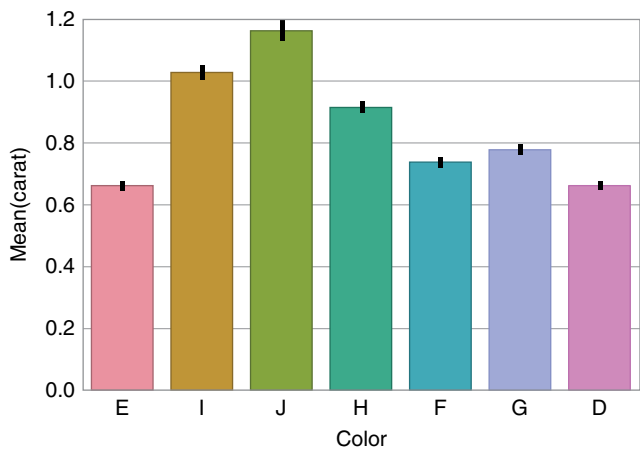
```
<seaborn.axisgrid.PairGrid at 0x7f0302824550>
```



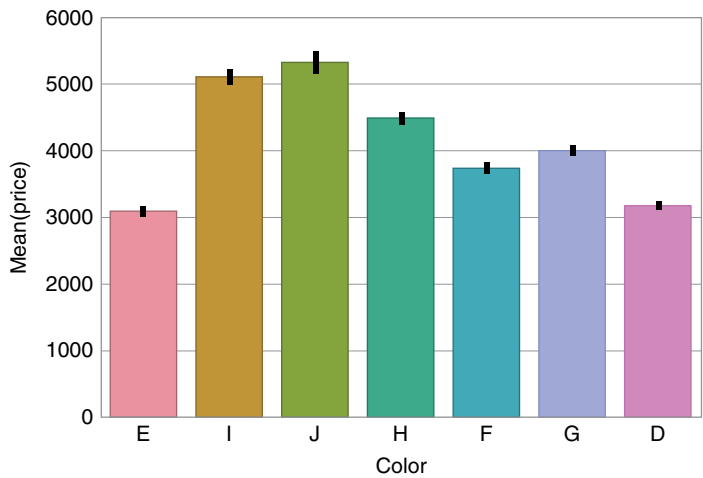
Barplot—We can use barplot as well quite easily in Python using seaborn. Let us take the diamonds dataset from the original ggplot2 package in R.

```
diamonds = pd.read_csv("https://vincentarelbundock.github.io/Rdatasets/csv/ggplot2/diamonds.csv")
sns.barplot(x="color", y="carat", data=diamonds)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0300092c18>
```



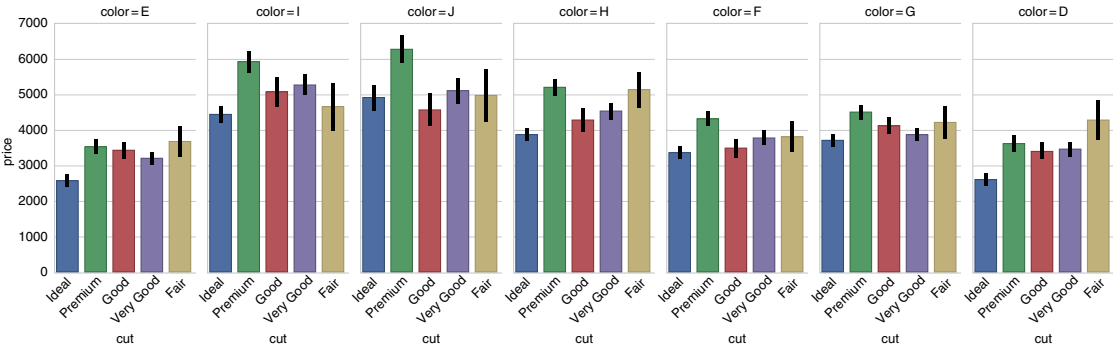
```
sns.barplot(x="color", y="price", data=diamonds)
<matplotlib.axes._subplots.AxesSubplot at
0x7f030003ce80>
```



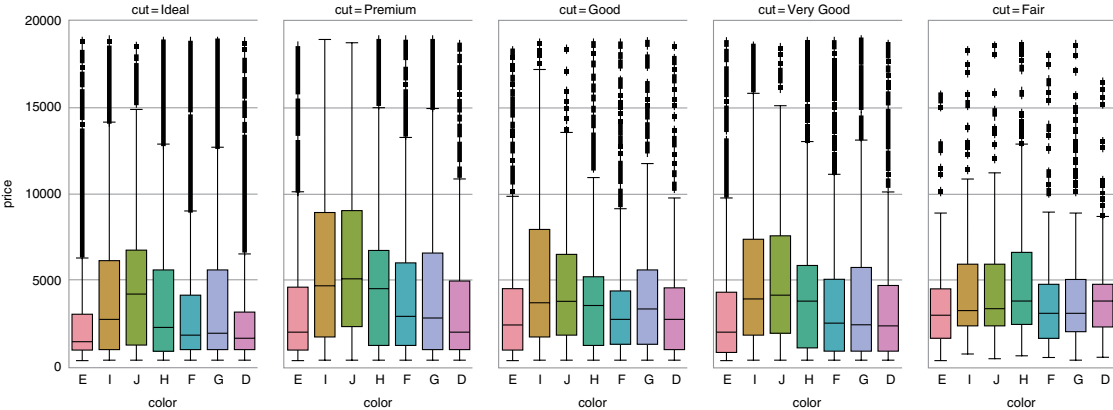
Factorplot—We use factorplot from seaborn library and find in the diamonds dataset that colors I,J have maximum price while cut Premium has maximum price compared with others. Factorplot draws a categorical plot onto a FacetGrid (see <https://stanford.edu/~mwaskom/software/seaborn/generated/seaborn.factorplot.html>)

By changing the **x-axis** and the **col** (color) variable, we get the following graphs, and by changing the **kind** parameter of factorplot from **box** to **bar** or **point**, we get the following graphs. The change in graphs helps with exploratory analysis.

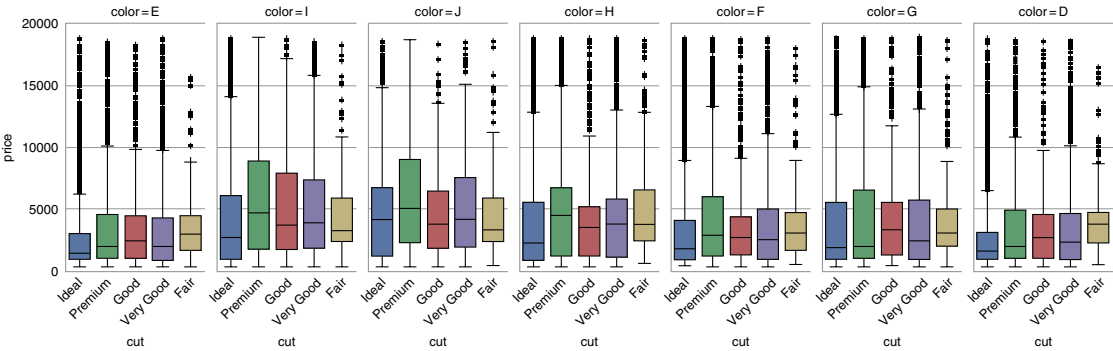
```
sns.factorplot(x="cut", y="price",col="color", data=diamonds, kind="bar", size=4,
aspect=.5);
```



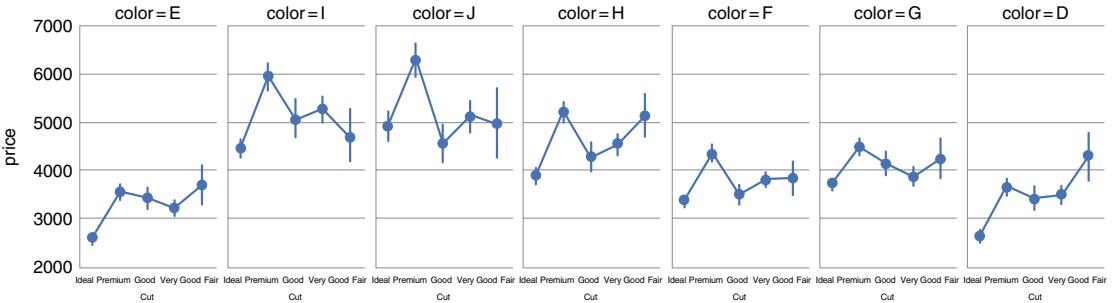
```
sns.factorplot(x="color", y="price",col="cut", data=diamonds, kind="box", size=4, aspect=.5);
```



```
sns.factorplot(x="cut", y="price",col="color", data=diamonds, kind="box", size=4,
aspect=.5);
```



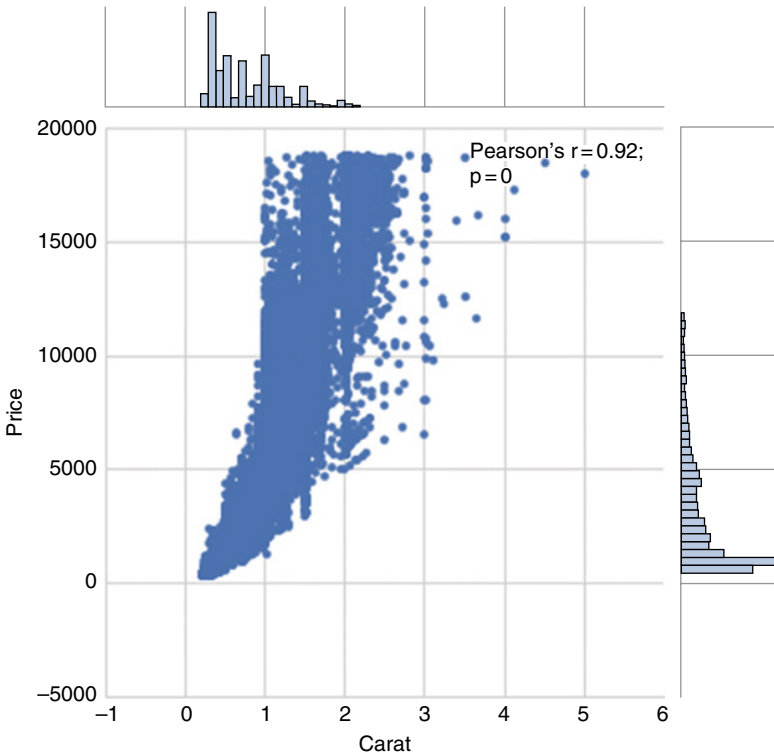
```
sns.factorplot(x="cut", y="price", col="color", data=diamonds, kind="point", size=4, aspect=.5);
```



We can use **jointplots** for combined plots.

They can be of the form kde (for density) or scatter (for points) or hexbins (for overplotting).

```
sns.jointplot(x="carat", y="price", data=diamonds)
<seaborn.axisgrid.JointGrid at 0x7f02f3d37908>
```



You can also view this tutorial online at <http://nbviewer.jupyter.org/gist/decisionstats/e9fd40890553b24acda5e07654bceaa8>

6.5 Advanced Plots

Grammar of graphics created by Wilkinson and implemented by Wickham in R has revolutionized data visualization in recent years. To summarize, when creating a plot we start with data. We can create many different types of plots using this same basic specification. (Bars, lines, and points are all examples of geometric objects.) We can scale the axes and statistically transform the data (bins, aggregates).

The concept of layers:

Plot = data 1 + scales and coordinate system 2 + plot annotations 3

- 1) data plot type
- 2) Axes and legends
- 3) background and plot title

The layered grammar defines the components of a plot as (Figures 6.4 and 6.5):

- A default dataset and set of mappings from variables to aesthetics
- One or more layers, with each layer having one geometric object, one statistical transformation, one position adjustment, and optionally, one dataset and a set of aesthetic mappings
- One scale for each aesthetic mapping used
- A coordinate system
- The facet specification

We can use the ggplot library created by Yhat to recreate ggplot style diagrams in Python without even changing the code. This is an example from <http://nbviewer.jupyter.org/gist/decisionstats/df98ff9df42e7764d600>

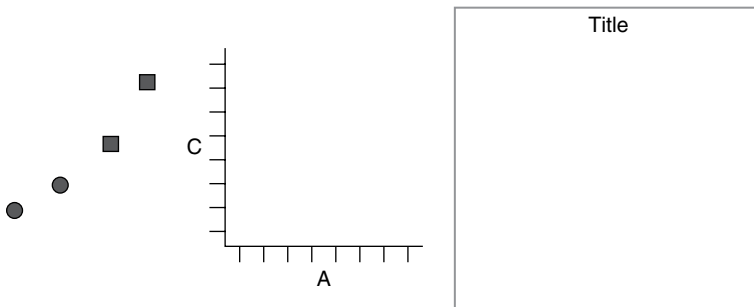


Figure 6.4 Graphics objects produced by (from left to right) geometric objects, scales and coordinate system, plot annotations. *Source:* <http://vita.had.co.nz/papers/layered-grammar.pdf>. © University of Cambridge.

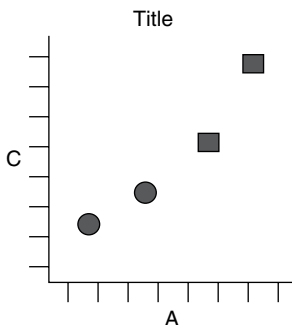
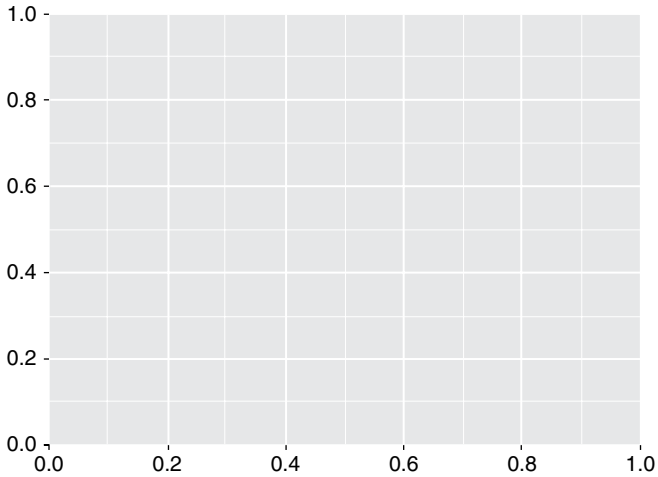


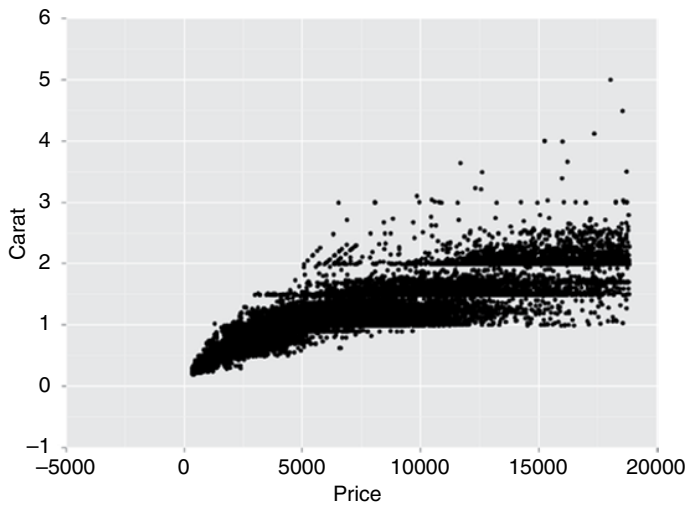
Figure 6.5 The final graphic, produced by combining the pieces in Figure 6.4. *Source:* <http://vita.had.co.nz/papers/layered-grammar.pdf>. © University of Cambridge.

```
import matplotlib as mt
%matplotlib inline #this line makes sure plots are in
same notebook
```

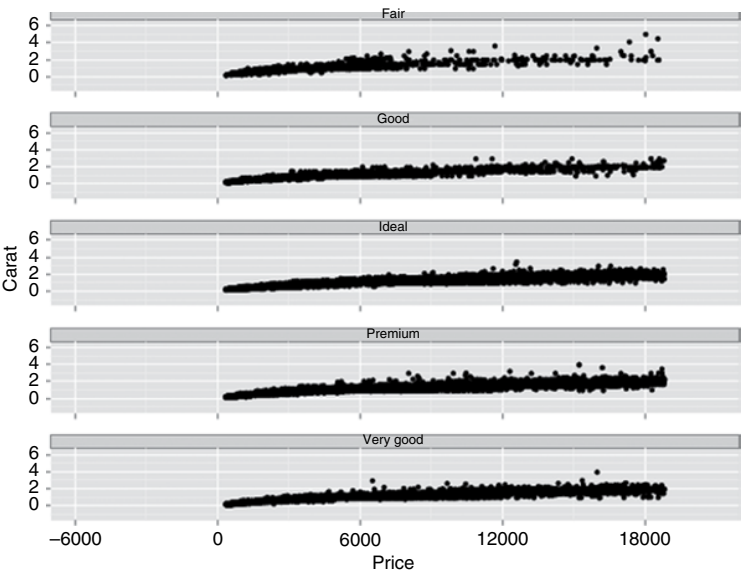
```
from ggplot import *
p = ggplot(aes(x='price', y='carat'), data=diamonds)
p
```



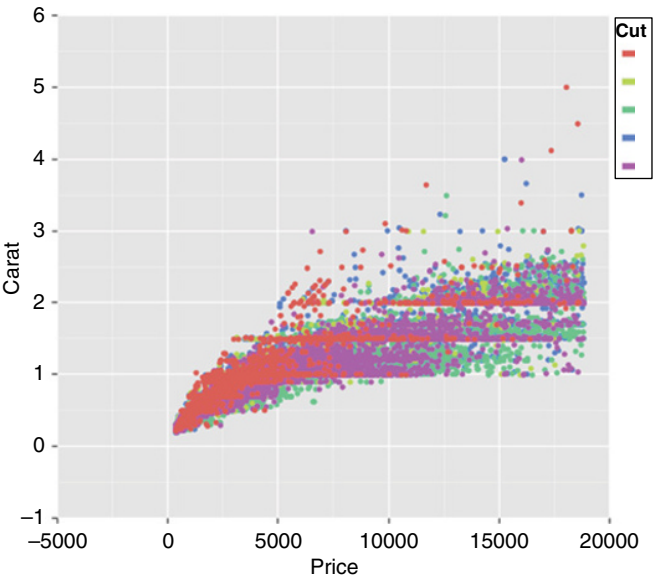
```
<ggplot: (-1059997756)>
p + geom_point()
```



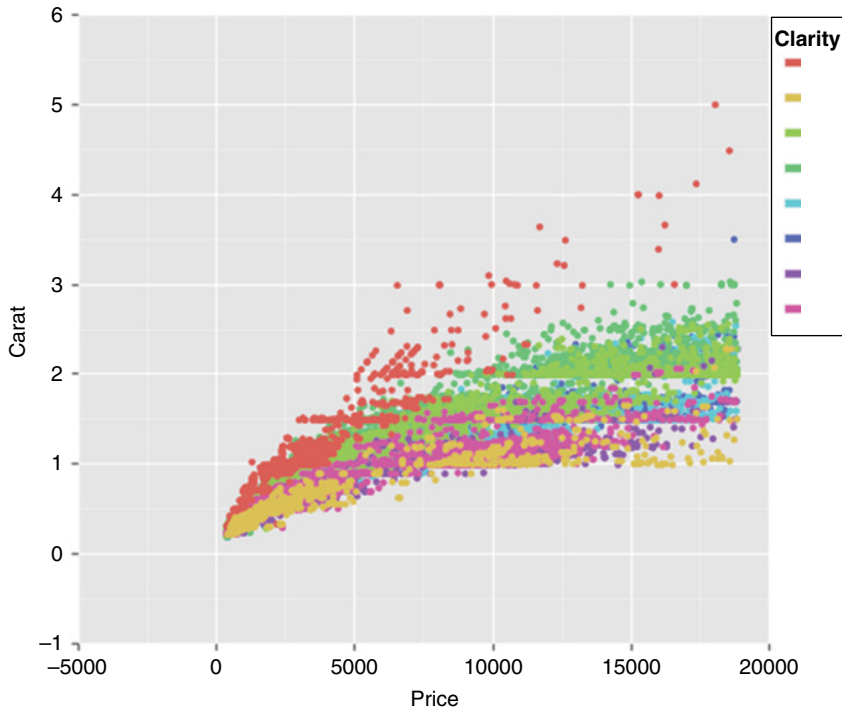
```
<ggplot: (-1059338452)>  
p + geom_point() +facet_grid('cut')
```



```
<ggplot: (-1057884332)>  
p = ggplot(aes(x='price', y='carat',color="cut"),  
data=diamonds)  
p + geom_point()
```



```
<ggplot: (-1059249386)>
p = ggplot(aes(x='price', y='carat', color="clarity"),
data=diamonds)
p + geom_point()
```



```
<ggplot: (-1060618628)>
```

6.6 Interactive Plots

Interactive plots can be done by bokeh in Python and by shiny package in R. You can also use plot.ly for both.

6.7 Spatial Analytics

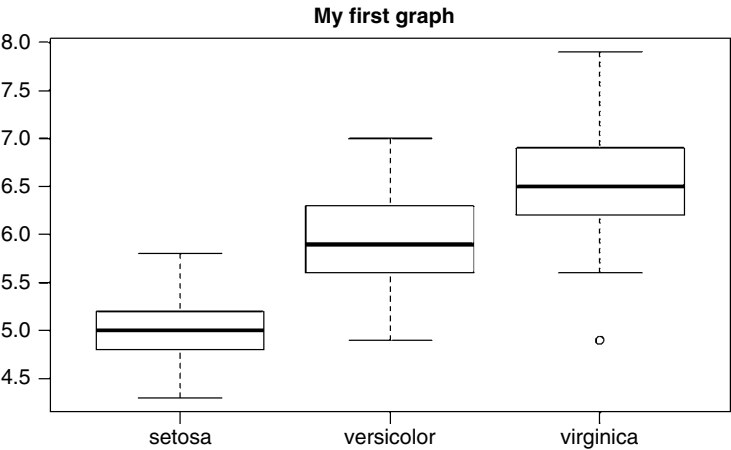
Spatial analytics can be done by leaflet package and by ggmap package in R. In R a special section for spatial packages is at <https://cran.r-project.org/web/views/Spatial.html>. In Python you can refer to <http://pysal.readthedocs.io/en/latest/PySAL> and packages at <https://pythongisresources.wordpress.com/packages/>

6.8 Data Visualization in R

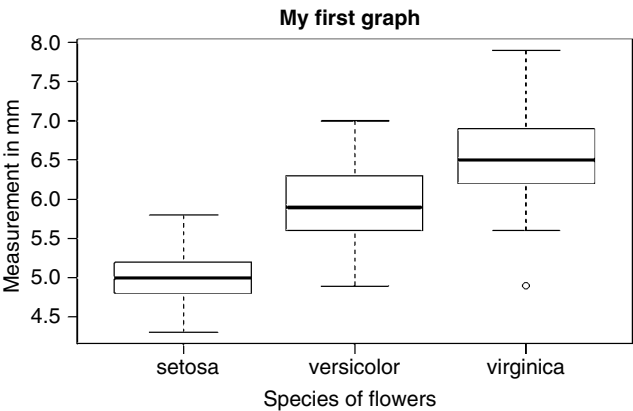
- Basic graphs

Some basic graphs in R are barplot and histogram. They are given by `barplot` and `hist` functions. Boxplot is given by `boxplot` function and is used for exploratory data analysis (EDA). The following is taken from <http://rpubs.com/ajaydecis/basicRdataviz2> and <http://rpubs.com/ajaydecis/dataviz2>

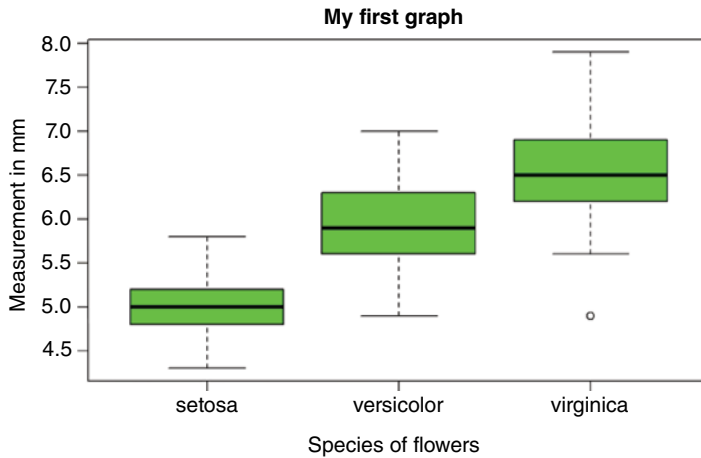
```
par (bg="yellow")
boxplot (Sepal.Length~Species,
        main="My First Graph")
```



```
boxplot (Sepal.Length~Species,
        main="My First Graph",
        xlab="Species of Flowers",
        ylab=" Measurement in mm")
```



```
boxplot(Sepal.Length~Species,
        main="My First Graph",
        xlab="Species of Flowers",
        ylab=" Measurement in mm",
        col="green")
```



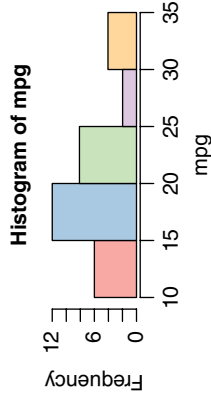
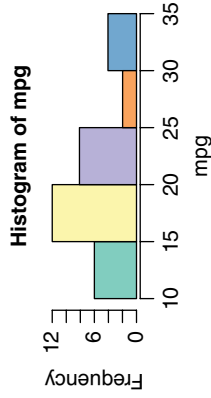
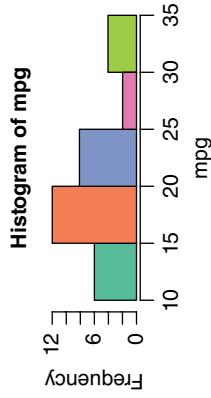
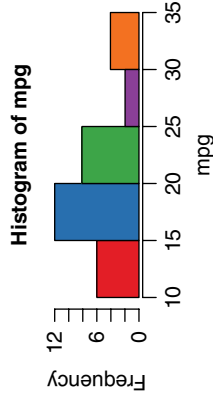
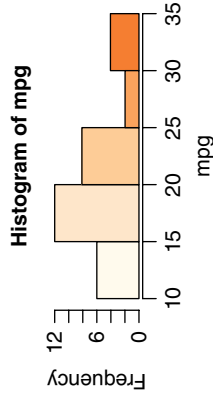
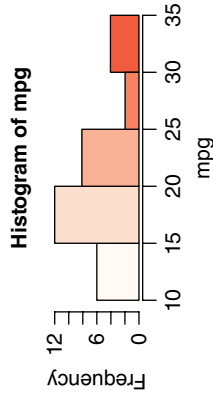
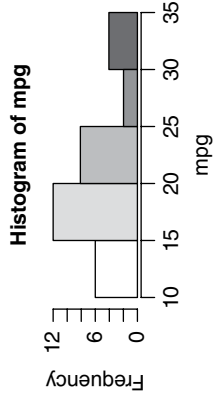
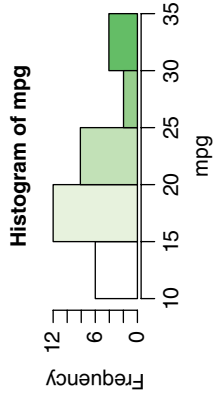
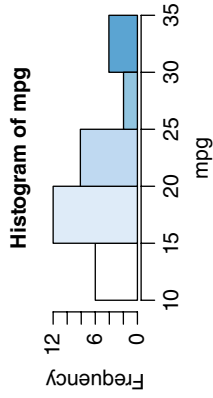
The package RColorBrewer adds in special color palettes or combinations of colors to R.

```
library(RColorBrewer)
par(mfrow=c(3,3))

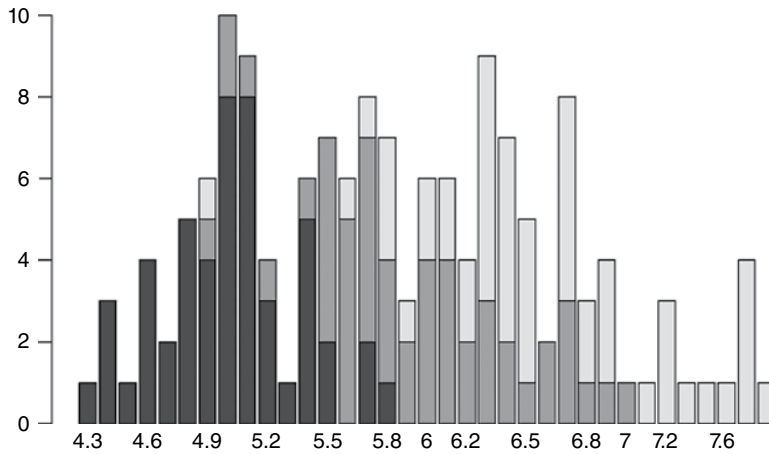
hist(mpg,col=brewer.pal(8,"Blues"))
hist(mpg,col=brewer.pal(8,"Greens"))
hist(mpg,col=brewer.pal(8,"Greys"))

hist(mpg,col=brewer.pal(8,"Reds"))
hist(mpg,col=brewer.pal(8,"Oranges"))

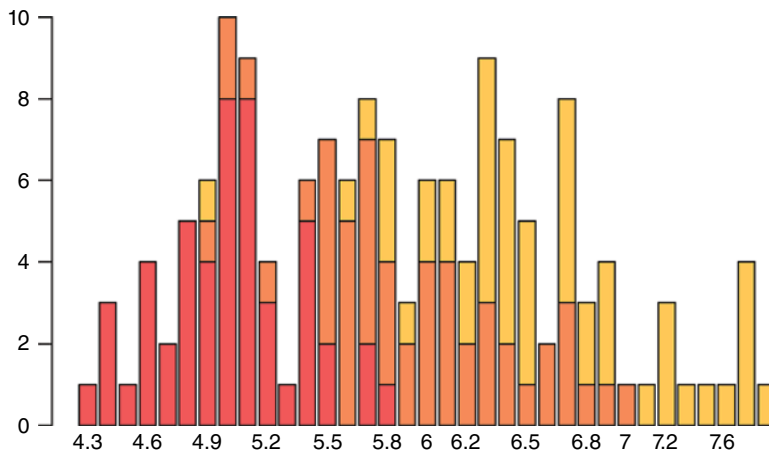
hist(mpg,col=brewer.pal(8,"Set1"))
hist(mpg,col=brewer.pal(8,"Set2"))
hist(mpg,col=brewer.pal(8,"Set3"))
hist(mpg,col=brewer.pal(8,"Pastell1"))
```



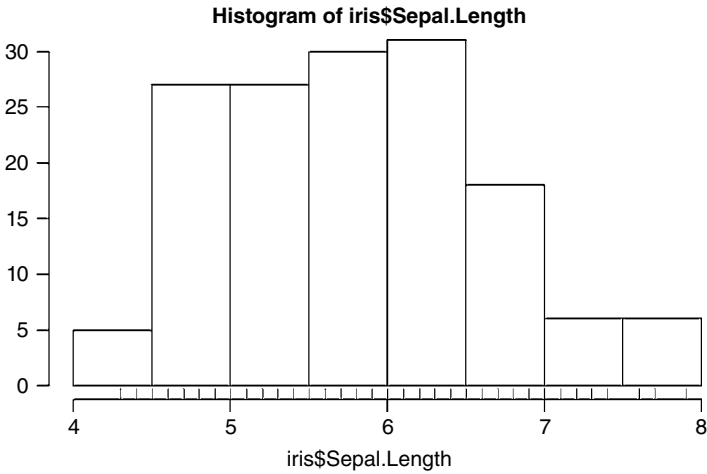
```
#barplot
barplot(table(iris$Species,iris$Sepal.Length))
```



```
barplot(table(iris$Species,iris$Sepal.Length),col=heat.colors(5,0.6))
```



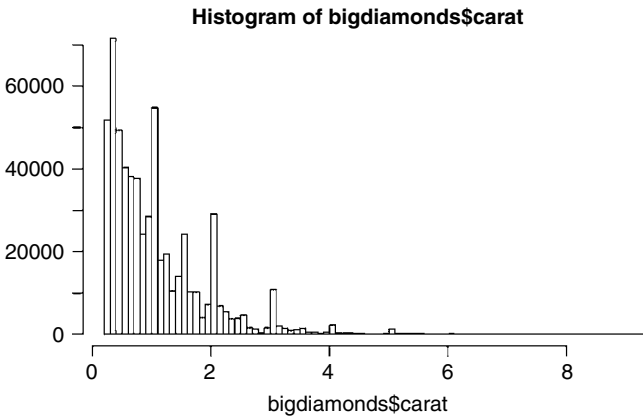
```
#rug plot
hist(iris$Sepal.Length,breaks=10)
rug(iris$Sepal.Length)
```



You can read and manipulate data quite fast using the data.table package in R:

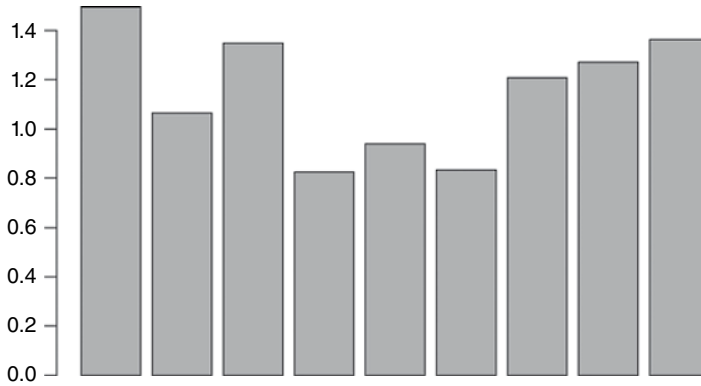
```
setwd("C:/Users/dell/Desktop")
library(data.table)

bigdiamonds=fread("BigDiamonds.csv")
##
Read 23.4% of 598024 rows
Read 50.2% of 598024 rows
Read 75.2% of 598024 rows
Read 598024 rows and 13 (of 13) columns from 0.049 GB
file in 00:00:06
hist(bigdiamonds$carat,breaks=100)
```



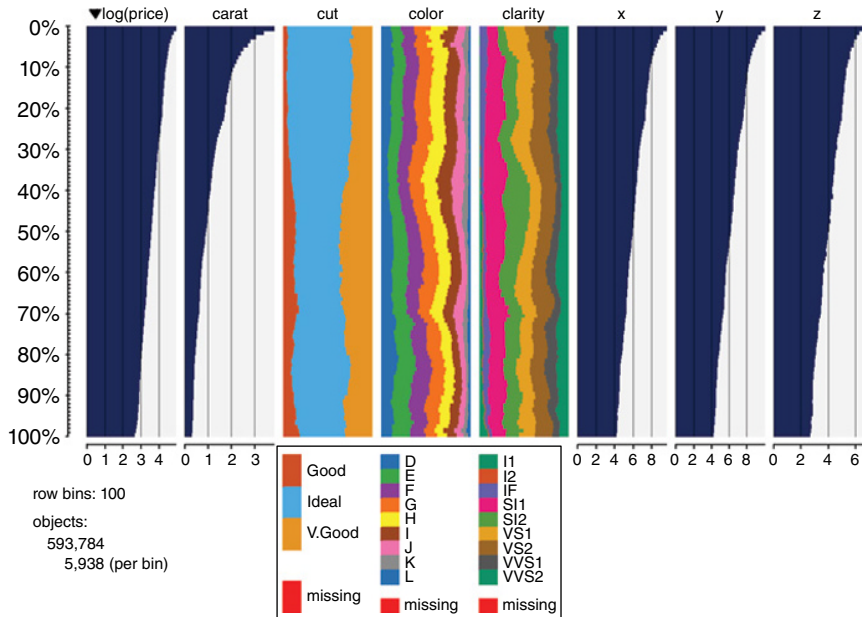
```
#rug(diamonds$carat)
```

```
barplot(bigdiamonds[,mean(carat),color]$V1)
```



We also have specialized packages/functions like `tableplot`:

```
tableplot(diamonds3)
```

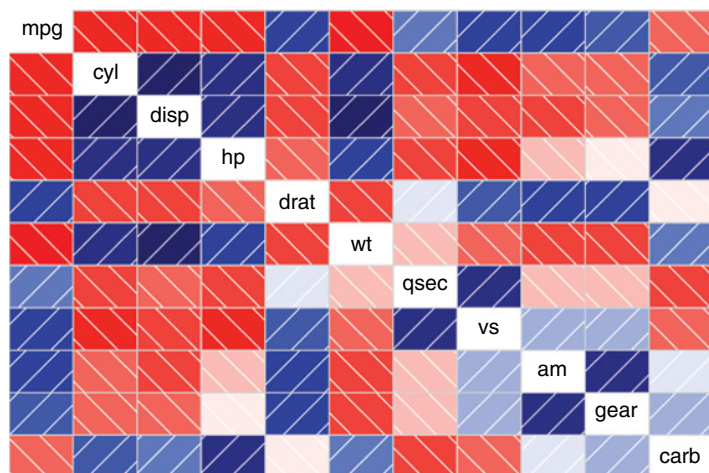


From <http://rpubs.com/ajaydecis/corrmosaic>

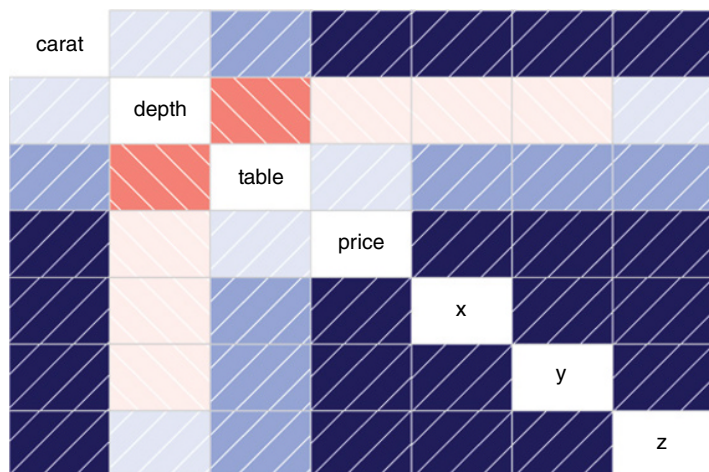
In a corrgram, negative correlation is shown in red, while the positive in blue with the intensity of colors showing the magnitude of correlation (for color

version refer online). In a mosaic plot, the area of the boxes shows the numbers for various subcategories.

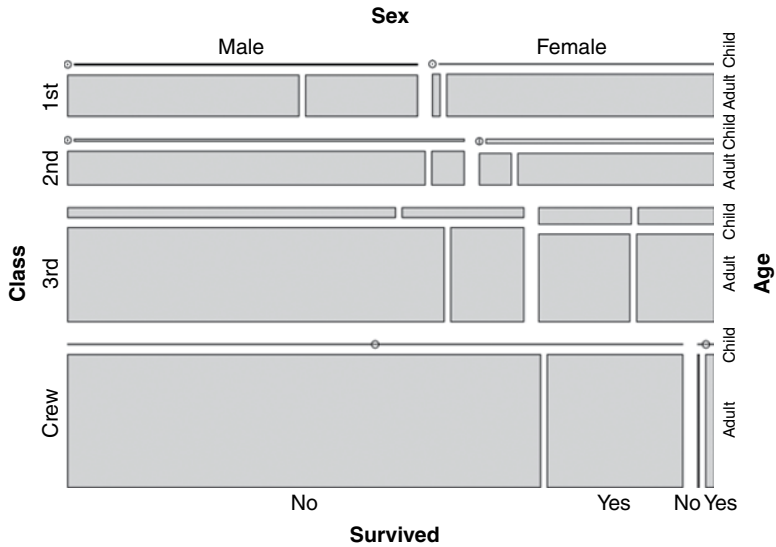
```
#install.packages("corrgram")
library(corrgram)
corrgram(mtcars)
```



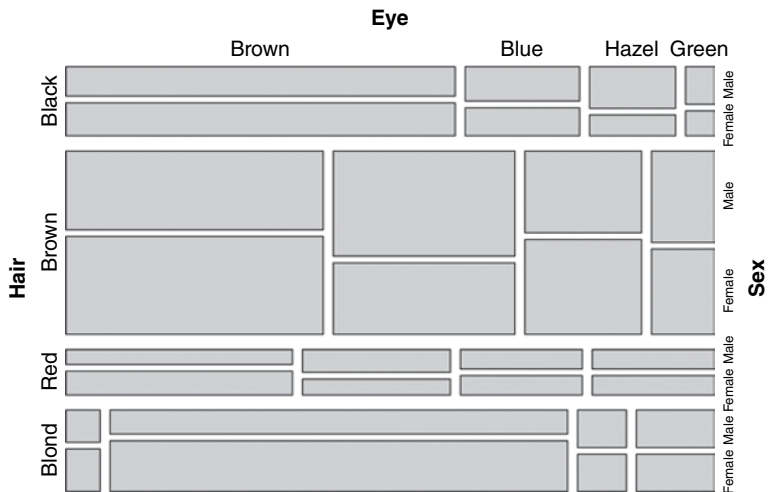
```
data(diamonds, package = "ggplot2")
corrgram(diamonds)
```



```
library(vcd)
## Loading required package: grid
mosaic(Titanic)
```



```
mosaic(HairEyeColor)
```

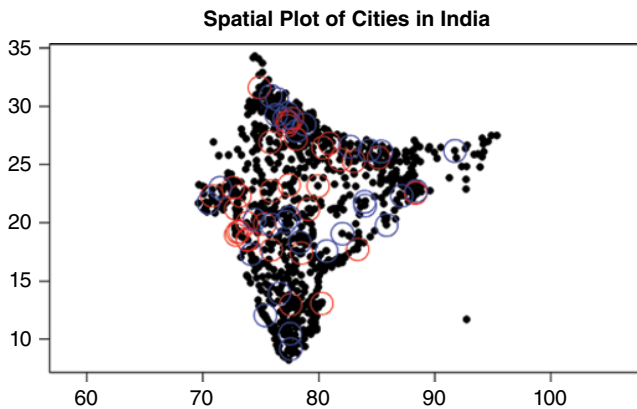


We can build spatial visualization using maps and ggmap packages in R. Example from <http://rpubs.com/ajaydecis/basicspatial>

```

par(mfrow=c(1,1))
plot(citiesIND, axes=T, asp=1, pch=16, main="Spatial
Plot of Cities in India")
## Highlight big cities
plot(citiesIND[citiesIND@data$pop > 1000000, ], pch=1,
col="red", cex=3, add=TRUE)
## Highlight cities with bigger dengue deaths
plot(citiesIND[citiesIND@data$samp > 960, ], pch=1,
col="blue", cex=3, add=TRUE)

```



A gallery of R graphs is available at http://scs.math.yorku.ca/index.php/R_Graphs_Gallery. © Wikipedia.

6.8.1 A Note of Sharing Your R Code by RStudio IDE

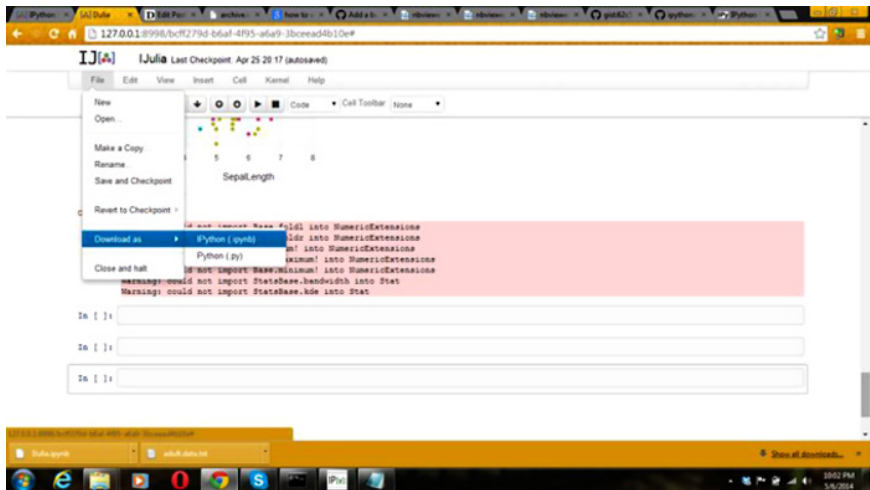
From <https://rpubs.com/about/getting-started>

- 1) In RStudio, create a new R Markdown document by choosing File. | New. | R Markdown.
- 2) Click the Knit HTML button in the doc toolbar to preview your document.
- 3) In the preview window, click the Publish button.

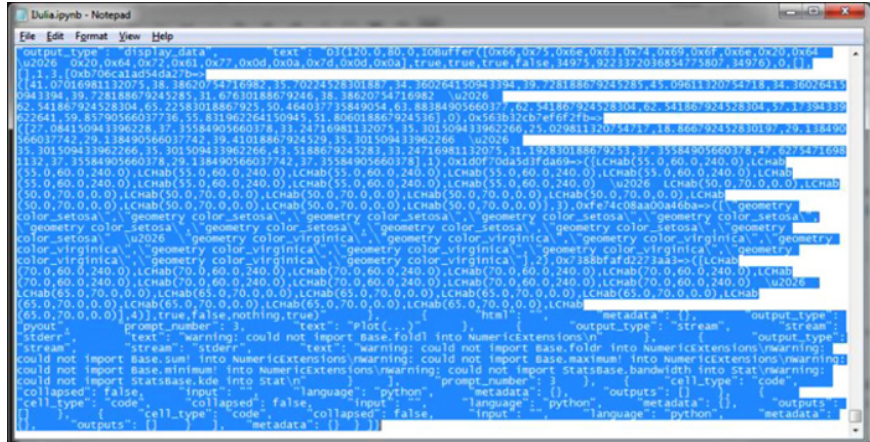
You will need a RPubs.com account to publish.

6.8.2 A Note on Sharing Your Jupyter Notebook

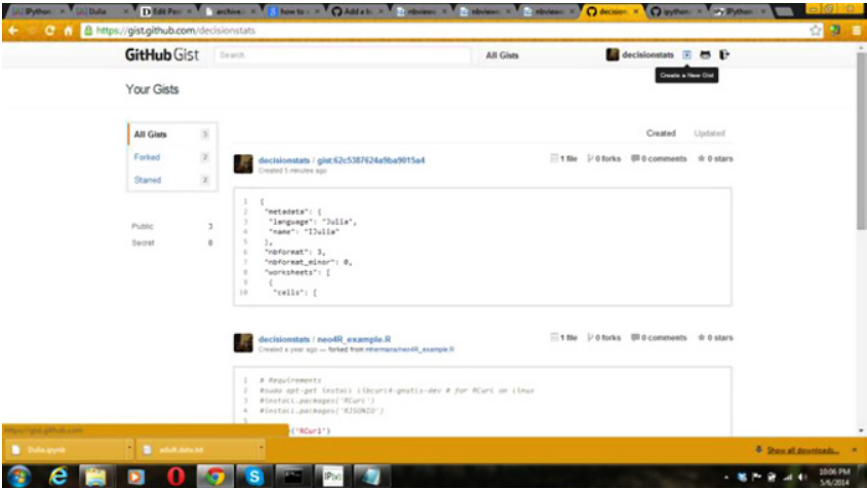
1) Download as IPython file from the file option.



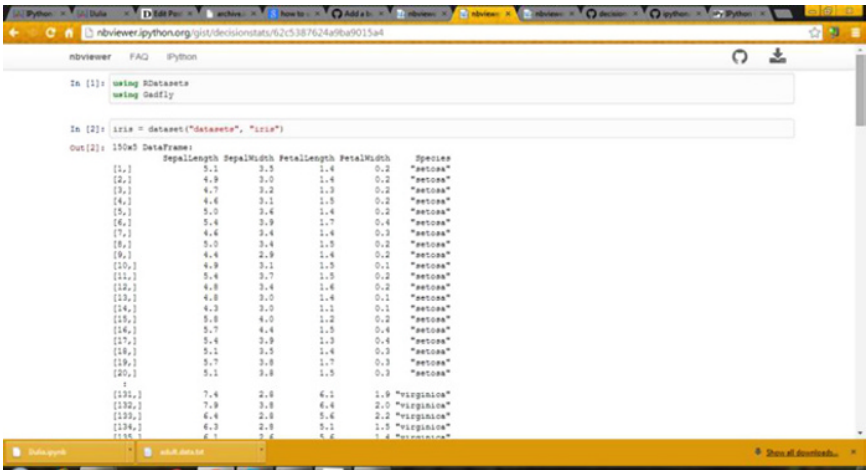
2) Use notepad to open the file downloaded. Copy the text contents.



3) Create a new gist at by pasting the text from step 2 here <https://gist.github.com/> (assuming you have a github account).



- 4) Paste the URL of the Gist at <http://nbviewer.ipython.org/> to get your iNotebook URL for sharing.
- 5) To update your notebook, simply copy and paste the new IPython code by editing the gist again.
- 6) An example here is <http://nbviewer.ipython.org/gist/decisionstats/62c5387624a9ba9015a4>



Bibliography

- F. J. Anscombe (1973). Graphs in Statistical Analysis. *The American Statistician*, 27(1), 17–21. <http://links.jstor.org/sici?sici=0003-1305> (accessed May 6, 2017).
- Vincent Arel-Bundock, Université de Montréal, Science politique, <https://vincentarelbundock.github.io/Rdatasets/> (accessed May 6, 2017). Rdatasets is a collection of 1039 datasets that were originally distributed alongside the statistical software environment R and some of its add-on packages. The goal is to make these data more broadly accessible for teaching and statistical software development.
- Coxcomb graphs. <http://understandinguncertainty.org/coxcombs> (accessed May 6, 2017).
- Stephen Few (2006). Common Pitfalls in Dashboard Design. http://www.perceptualedge.com/articles/Whitepapers/Common_Pitfalls.pdf (accessed May 6, 2017).
- Michael Friendly (2008). The Golden Age of Statistical Graphics. *Statistical Science*, 23(4), 502–535. http://projecteuclid.org/download/pdfview_1/euclid.ss/1242049392 (accessed May 6, 2017).
- Charles Joseph (Spring 2002). Visions and Re-Visions. *Journal of Educational and Behavioral Statistics*, 27(1), 31–52. <http://www.datavis.ca/papers/jebs.pdf> (accessed May 6, 2017).
- Ajay Ohri (2014). Decisionstats. <https://decisionstats.com/2014/05/08/how-to-share-your-ipython-or-ijulia-code/> (accessed May 6, 2017).
- Ian Spence (Winter 2005). No Humble Pie: The Origins and Usage of a Statistical Chart. *Journal of Educational and Behavioral Statistics*, 30(4), 353–368. <http://www.psych.utoronto.ca/users/spence/Spence%202005.pdf> (accessed May 6, 2017).
- Edward Tufte. *The Visual Display of Quantitative Information*, Second Edition, 1983. <http://thedoublethink.com/2009/08/tufte%E2%80%99s-principles-for-visualizing-quantitative-information/> (accessed May 6, 2017).
- Eric W. Weisstein. Hypothesis Testing. From *MathWorld*—A Wolfram Web Resource. <http://mathworld.wolfram.com/HypothesisTesting.html> (accessed May 6, 2017).
- Hadley Wickham (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, New York.
- Graham J. Williams (2011). *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*. Springer-Verlag, New York.
- Chi Yau. R Tutor. <http://www.r-tutor.com/elementary-statistics/goodness-fit/chi-squared-test-independence> (accessed May 6, 2017).
- Mark Zachry and Charlotte Thralls (2004). An Interview with Edward R. Tufte. *Technical Communication Quarterly*, 13(4), 447–462. https://www.edwardtufte.com/tufte/s15427625tcq1304_5.pdf (accessed May 6, 2017).

6.8.3 Special Note: A Complete Wing to Wing Tutorial on Python

Python is a very widely used programming language. Written by Guido Von Rossum in 1989, it is now one of the most widely used programming languages. In data science, Python has increasingly made strides, thanks to the pandas package as well as the efforts of PyData community. Companies like Continuum Analytics, Enthought, and Civis Analytics are creating both tools as well as actually utilizing Python for data science. Companies like Datakind, CodeAcademy, and Dataquest offer online education on Python for free. Unlike R language, Python has two major versions, Python 2 and Python 3, but just like R it is free and open source.

Core design parameters for Python remain crisp lines of code, using white space as an input, emphasis for indentation, and sparse grammar. People interested in knowing more on Python can go to the home page at <https://www.python.org/>

Data science lies at the intersection of programming, statistics, and business analysis. It is the use of programming tools with statistical techniques to analyze data in a systematic and scientific way. Accordingly this tutorial will try to focus at least on the statistical and programming parts of data science. Data scientists would also be interested in the PyData community at <http://pydata.org/>. Why use Python for data science? Python has surprising capabilities in data analysis and data visualization, thanks to the new generation of packages being created (pairplot on famous iris dataset using seaborn package is shown below) (Figure 6.6).

Here is a brief tutorial in Pythonic data science. Some prerequisites are given as follows:

Installations:

- 1) Download and install Anaconda from <https://www.continuum.io/downloads> (alternatives could be Canopy Express from <https://store.enthought.com/> or just the core implementation from <https://www.python.org/downloads/>).
- 2) Download and install the Jupyter Notebook Interface from <http://jupyter.readthedocs.org/en/latest/install.html>
- 3) You can use pip or easy_install to install packages. There are more than 72000 Python packages available at <https://pypi.python.org/pypi> and you can browse Python packages by topic at <https://pypi.python.org/pypi?%3Aaction=browse>

Packages for data science

Some important packages for data scientists to use in Python are as follows:

- 1) Pandas (<http://pandas.pydata.org/>)—Pandas allows users the familiar data frame format in which rows are observations and columns are variables and a wide variety of useful data analysis features.

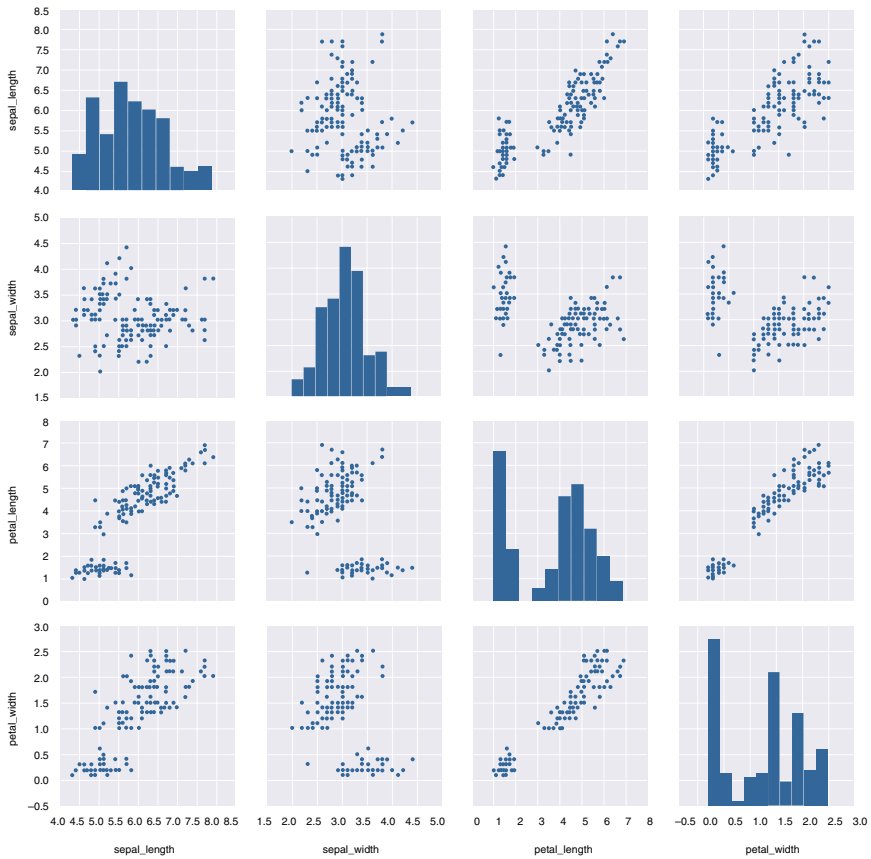


Figure 6.6 Pairplot on Iris Dataset using seaborn package.

- 2) Scikit-learn (<http://scikit-learn.org/>)—Scikit-learn allows you a widely used machine learning package for data mining and modeling.
- 3) Statsmodels (<http://statsmodels.sourceforge.net/>)—Statsmodels brings statistical tests and models available in Python.
- 4) Seaborn (<http://stanford.edu/~mwaskom/software/seaborn/>)—Seaborn brings statistical data visualization to Python.
- 5) Pandasql (<https://pypi.python.org/pypi/pandasql>)—This package allows SQL syntax and is thus similar to sqldf package in R.
- 6) ggplot (<http://ggplot.yhathq.com/>)—This is the implementation of grammar of graphics in Python. You can practically reuse same ggplot2 code from R to this package in Python.
- 7) SQLAlchemy (<http://www.sqlalchemy.org/>)—This tool allows you to connect and query with databases.

Tutorial Overview

- 1) You can write markdown within Jupyter notebook by changing the code cell type from code to Markdown. You can also install and work with R using the IR Kernel. This makes the code more readable as well as very easy to switch between kernels.
- 2) Install packages from within the Jupyter notebook using a `!` sign in the beginning.
- 3) Import (or load) packages using the following syntax `Import Package`, or `Import Package as Pkg` or `Import Function from Package`. This is similar to library function in R.
- 4) Read in data using the `read_csv` or similar Input functions from Pandas (<http://pandas.pydata.org/pandas-docs/stable/io.html>).
- 5) Inspect data using the `info` and `head` methods.
- 6) Slice data using the `query` function or `index` or the column name.
- 7) Summarize data using the `describe`, `group_by`, and `value_counts` functions.
- 8) Use `dir` on the object to find out what all can be done on it.
- 9) Visualize using various plots from `seaborn` and `ggplot` package.
- 10) Build a regression model using `statsmodel` using the familiar formula method (`dependent_var ~ independent_var 1 + independent_var2 +`).
- 11) Learn about additional tools useful for data scientists.

Detailed Tutorial

- 1) Install packages from within Jupyter notebook. Use the `--upgrade` flag to upgrade existing packages.

```
In [1]: ! sudo pip install pandas --upgrade
```

- 2) Load the package. You can load a Python package using the following ways: `import PACKAGE` or `import PACKAGE as PK` or `from PACKAGE import FUN`. You can then invoke the function using `PACKAGE.FUN`, `PK.FUN`, and `FUN`, respectively.

```
In [2]: import pandas as pd
```

- 3) Import Data. We use `read_csv` from `pandas` to import a csv file. Note that Jupyter automatically applies color to the code to ensure code, functions, comments are easily readable. In case the file is stored locally, we can use the `os` Python library.

```
In [3]: import os as os
os.getcwd() #current working directory Out[3]:
'/home/ajay/Dropbox/PYTHON BOOK WILEY/FINAL'
```

```
In [4]:os.chdir('/home/ajay/Desktop/test') #change
current working directory
```

```
In [5]:os.listdir(os.getcwd()) #list files in
directory Out [5]: ['adult.data.txt']
```

```
In [6]:adult=pd.read_csv("adult.data.
txt",header=None) #read data
```

```
'''Lets get some information on the object. This
was a multiple line comment using three single quote
marks'''
```

- 4) Let's use a dataset from within R's dataset for familiarity. We will use diamond dataset bundled with R language from <https://vincentarelbundock.github.io/Rdatasets/datasets.html>

```
In [12]:
diamonds =pd.read_csv("https://vincentarelbundock.
github.io/Rdatasets/csv/ggplot2/diamonds.csv")
```

- 5) We can use len to find out number of observations or length, and type to find out class of object type. Using **info** we can combine all these to get the information on object.

```
In [7]:diamonds.info()
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 53940 entries, 0 to 53939
```

```
Data columns (total 11 columns):
```

```
Unnamed: 0    53940 non-null int64
carat         53940 non-null float64
cut           53940 non-null object
color         53940 non-null object
clarity       53940 non-null object
depth         53940 non-null float64
table         53940 non-null float64
price         53940 non-null int64
x             53940 non-null float64
y             53940 non-null float64
z             53940 non-null float64
```

```
dtypes: float64(6), int64(2), object(3)
```

```
memory usage: 4.3+ MB
```

- 6) To find out what all functions can do, we can just use the **dir** command on the object, that is, `dir(diamonds)`. We can use **head** to inspect first few rows, **.ix** to select rows by index number, and **double square brackets** with **column names in quotes** to select by column name. Note that we can **chain multiple commands** in Python very easily.

```
In [8]:diamonds2=diamonds.drop('Unnamed: 0', 1)
#Dropping a particular variable
diamonds2.head()
```

Out [8] :

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75

In [9]:diamonds.ix[20:28] *#refers to the 21st to 29th row since index starts from 0.*

Out [9] :

	carat	cut	color	clarity	depth	table	price	x	y	z
20	0.30	Good	I	SI2	63.3	56	351	4.26	4.30	2.71
21	0.23	Very Good	E	VS2	63.8	55	352	3.85	3.92	2.48
22	0.23	Very Good	H	VS1	61.0	57	353	3.94	3.96	2.41
23	0.31	Very Good	J	SI1	59.4	62	353	4.39	4.43	2.62
24	0.31	Very Good	J	SI1	58.1	62	353	4.44	4.47	2.59
25	0.23	Very Good	G	VVS2	60.4	58	354	3.97	4.01	2.41
26	0.24	Premium	I	VS1	62.5	57	355	3.97	3.94	2.47
27	0.30	Very Good	J	VS2	62.2	57	357	4.28	4.30	2.67
28	0.23	Very Good	D	VS2	60.5	61	357	3.96	3.97	2.40

```
In [10]:diamonds.ix[20:25].cut
```

Out [10] :

```
20    Good
21  Very Good
22  Very Good
23  Very Good
24  Very Good
25  Very Good
Name: cut, dtype: object
```

```
In [11]: diamonds[['color', 'cut', 'price']].head()
#Note the double square brackets [[]]
Out [8]:
```

	color	cut	price
0	E	Ideal	326
1	E	Premium	326
2	E	Good	327
3	I	Premium	334
4	J	Good	335

- 7) Conditional selection—We can use the **query** command for conditional selection of data.

```
In [12]: diamonds.query('carat >3 and color == "J"')
Out [12]:
```

	carat	cut	color	clarity	depth	table	price	x	y	z
21758	3.11	Fair	J	I1	65.9	57	9823	9.15	9.02	5.98
25999	4.01	Premium	J	I1	62.5	62	15223	10.02	9.94	6.24
26467	3.01	Ideal	J	SI2	61.7	58	16037	9.25	9.20	5.69
26744	3.01	Ideal	J	I1	65.4	60	16538	8.99	8.93	5.86
27415	5.01	Fair	J	I1	65.5	59	18018	10.74	10.54	6.98
27630	4.50	Fair	J	I1	65.8	58	18531	10.23	10.16	6.72
27679	3.51	Premium	J	VS2	62.5	59	18701	9.66	9.63	6.03
27684	3.01	Premium	J	SI2	60.7	59	18710	9.35	9.22	5.64
27685	3.01	Premium	J	SI2	59.7	58	18710	9.41	9.32	5.59

- 8) Data summary is done in Pandas by **describe** for numerical variables and by **value_counts** for categorical variables. Numerical correlation can be done by **corr** command. Unique values are given by **unique** command.

```
In [13]: diamonds.price.describe()
Out [13]:
count    53940.000000
mean         3932.799722
std         3989.439738
min          326.000000
25%         950.000000
50%        2401.000000
75%        5324.250000
max       18823.000000
Name: price, dtype: float64
```

```
In [14]:diamonds.corr() #Numerical Correlations
Out [14]:
```

	carat	depth	table	price	x	y	z
carat	1.000000	0.028224	0.181618	0.921591	0.975094	0.951722	0.953387
depth	0.028224	1.000000	-0.295779	-0.010647	-0.025289	-0.029341	0.094924
table	0.181618	-0.295779	1.000000	0.127134	0.195344	0.183760	0.150929
price	0.921591	-0.010647	0.127134	1.000000	0.884435	0.865421	0.861249
x	0.975094	-0.025289	0.195344	0.884435	1.000000	0.974701	0.970772
y	0.951722	-0.029341	0.183760	0.865421	0.974701	1.000000	0.952006
z	0.953387	0.094924	0.150929	0.861249	0.970772	0.952006	1.000000

```
In [15]:diamonds['cut'].unique()
Out [15]:array(['Ideal', 'Premium', 'Good', 'Very
Good', 'Fair'], dtype=object)
```

```
In [16]:pd.value_counts(diamonds.cut)
Out [16]:
Ideal          21551
Premium        13791
Very Good     12082
Good           4906
Fair           1610
Name: cut, dtype: int64
```

Note: To run a command on a particular column instead of entire data frame, I can just use the dot notation and its name (i.e., diamonds.price instead of diamonds. This is analogous to R’s \$ notation).

9) Group by summary is done by **group_by** command and cross tabulation can be done by **crosstab**.

```
In [17]:cutgroup=pd.groupby(diamonds,diamonds.cut)
In [18]:type(cutgroup)
Out [18]:
pandas.core.groupby.DataFrameGroupBy
In [19]:cutgroup.price.median()
Out [19]:
cut
Fair          3282.0
Good          3050.5
Ideal          1810.0
Premium        3185.0
Very Good      2648.0
Name: price, dtype: float64
```

```
In [20]:pd.crosstab(diamonds.cut,diamonds.
color,margins='TRUE')
Out [20]:
```

color	D	E	F	G	H	I	J	All
cut								
Fair	163	224	312	314	303	175	119	1610
Good	662	933	909	871	702	522	307	4906
Ideal	2834	3903	3826	4884	3115	2093	896	21551
Premium	1603	2337	2331	2924	2360	1428	808	13791
Very Good	1513	2400	2164	2299	1824	1204	678	12082
All	6775	9797	9542	11292	8304	5422	2808	53940

Note: We can use `dropna` to remove missing values in Python, that is,
`diamonds= diamonds.dropna(how='any')`

- 10) We can also pivot data like a pivot table using **pivot** command.

```
In [21]:e=diamonds.groupby(['cut', "color"]).price.
median().reset_index()
e.pivot(index='cut', columns='color',
values='price')
Out [21]:
```

color	D	E	F	G	H	I	J
cut							
Fair	3730.0	2956.0	3035	3057.0	3816.0	3246.0	3302
Good	2728.5	2420.0	2647	3340.0	3468.5	3639.5	3733
Ideal	1576.0	1437.0	1775	1857.5	2278.0	2659.0	4096
Premium	2009.0	1928.0	2841	2745.0	4511.0	4640.0	5063
Very Good	2310.0	1989.5	2471	2437.0	3734.0	3888.0	4113

- 11) Using SQL—Python does have the `pandasql` package, thanks to the team at YHat (who also made the Rodeo IDE). It is similar to the `sqldf` package in R that allows the user to write sql queries to the data frame object. Note that you need to ensure table names are consistent with SQLite tablename conventions (thus it makes sense to drop or rename any column name with any special characters).

```
In [22]:from pandasql import sqldf
pysqldf = lambda q: sqldf(q, globals())
In [23]:pysqldf("SELECT * FROM diamonds2 LIMIT 5 ; ")
```

Out [23] :

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
5	0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48

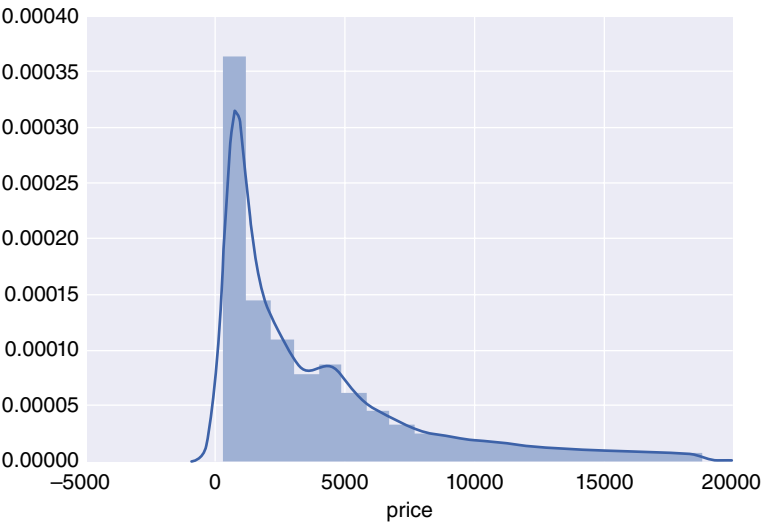
In [24]:`pysqldf("SELECT * FROM diamonds2 WHERE carat >4 ;")`

Out [24] :

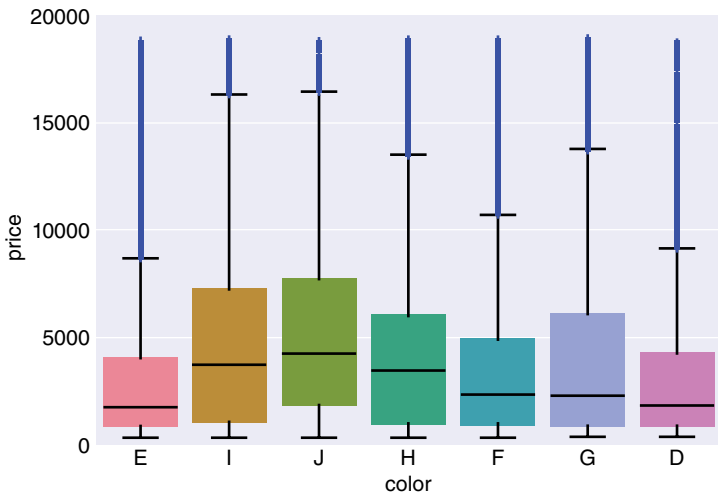
	carat	cut	color	clarity	depth	table	price	x	y	z
0	4.01	Premium	I	I1	61.0	61	15223	10.14	10.10	6.17
1	4.01	Premium	J	I1	62.5	62	15223	10.02	9.94	6.24
2	4.13	Fair	H	I1	64.8	61	17329	10.00	9.85	6.43
3	5.01	Fair	J	I1	65.5	59	18018	10.74	10.54	6.98
4	4.50	Fair	J	I1	65.8	58	18531	10.23	10.16	6.72

- 12) For data visualization I am going to first use the excellent seaborn package from <http://stanford.edu/~mwaskom/software/seaborn/index.html>. Histograms, boxplots, scatterplots, and jointplots are very easily plotted using seaborn.

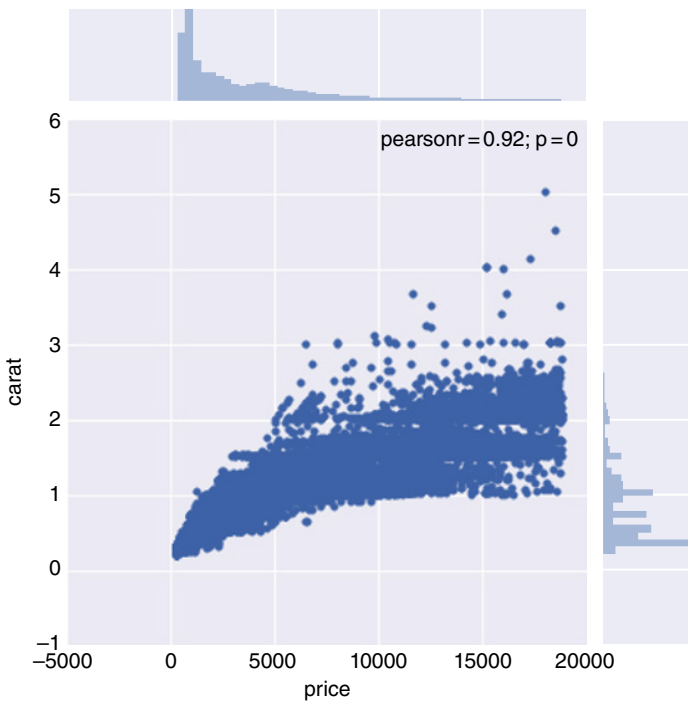
In [25]:`sns.distplot(diamonds.price, bins=20, kde=True, rug=False);`



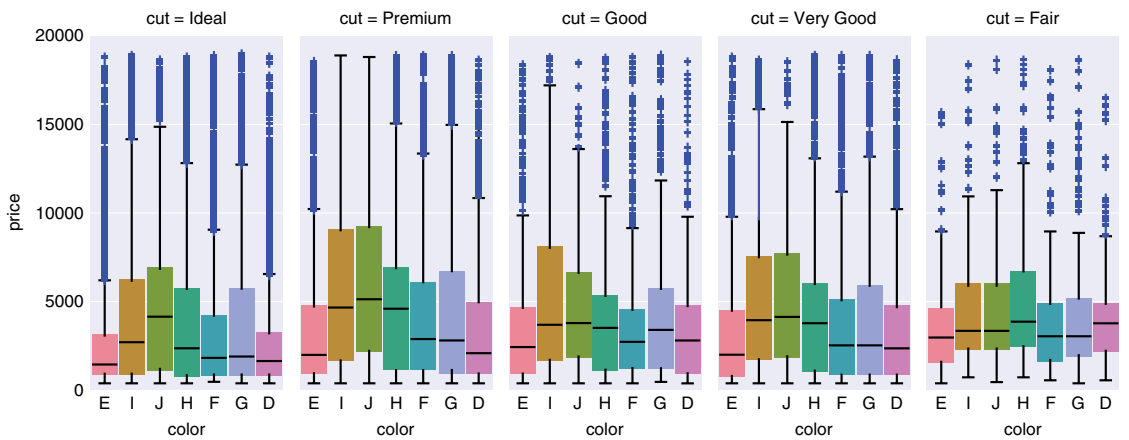
```
In [25]: ax = sns.boxplot(x="color", y="price",
data=diamonds)
```



```
In [26]: sns.jointplot('price', 'carat', data=diamonds2)
Out [26]: <seaborn.axisgrid.JointGrid at 0x9717fd8c>
```

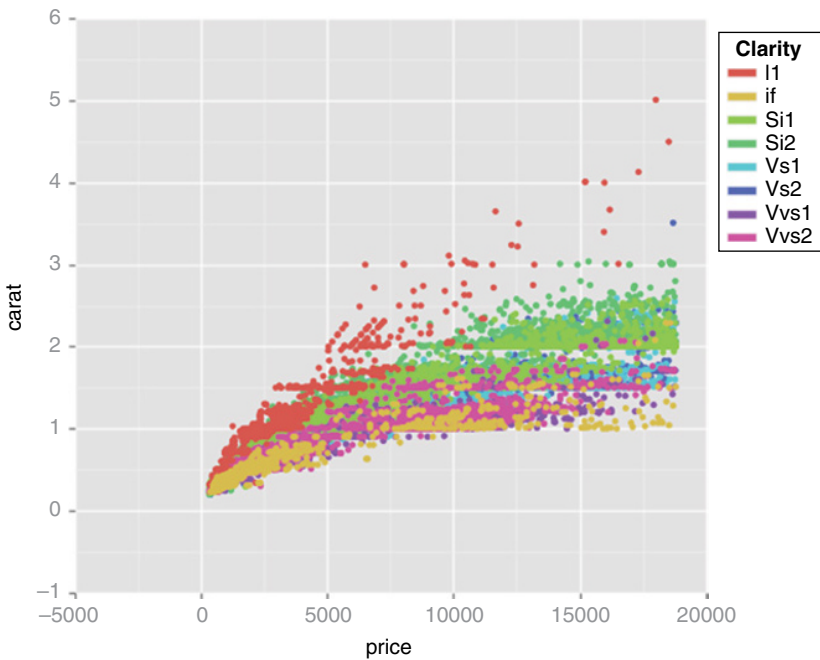


```
In [27]:sns.factorplot(x="color", y="price",
                      col="cut", data=diamonds, kind="box", size=4, aspect=.5);
```



- 13) For data visualization, I can also use the ggplot package created by Yhat (who also created pandasql and rodeo—a RStudio style editor for Python). It uses the grammar of graphics as created by Wilkinson and popularized by Hadley Wickham.

```
In [28]: p = ggplot(aes(x='price',
y='carat',color="clarity"), data=diamonds)
p + geom_point()
```



```
Out [28]: <ggplot: (-917530690)>
```

- 14) For regression models, a widely used data science technique for business, I can also use the statsmodel package.

```
In [80]: import statsmodels.formula.api as sm
In [81]: boston=pd.read_csv("http://
vincentarelbundock.github.io/Rdatasets/csv/MASS/
Boston.csv")
In [82]: boston =boston.drop('Unnamed: 0', 1)
In [83]: boston.head()
```

Out [83] :

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

```
In [87]:import statsmodels.formula.api as sm
result = sm.ols(formula="medv ~ crim + zn + nox +
ptratio + black + rm ", data=boston).fit()
result.summary()
Out [87] :
```

Dep. Variable:	medv	R-squared:	0.631
Model:	OLS	Adj. R-squared:	0.626
Method:	Least Squares	F-statistic:	142.0
Date:	Fri, 22 Jan 2016	Prob (F-statistic):	1.49e-104
Time:	13:22:42	Log-Likelihood:	-1588.2
No. Observations:	506	AIC:	3190.
Df Residuals:	499	BIC:	3220.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[95.0% Conf. Int.]
Intercept	-0.3594	4.863	-0.074	0.941	-9.915 9.196
crim	-0.0991	0.034	-2.890	0.004	-0.167 -0.032
zn	-0.0064	0.014	-0.470	0.638	-0.033 0.020
nox	-10.8653	2.865	-3.793	0.000	-16.494 -5.237
ptratio	-1.0519	0.135	-7.796	0.000	-1.317 -0.787
black	0.0137	0.003	4.453	0.000	0.008 0.020
rm	6.9796	0.396	17.612	0.000	6.201 7.758

Omnibus:	298.859	Durbin-Watson:	0.808
Prob(Omnibus):	0.000	Jarque-Bera (JB):	3305.426
Skew:	2.385	Prob(JB):	0.00
Kurtosis:	14.577	Cond. No.	7.66e+03

```
In [88]:result.params
Out [88] :
Intercept  -0.359432
crim       -0.099122
zn         -0.006364
nox        -10.865295
```

```
ptratio    -1.051937
black      0.013737
rm         6.979587
dtype: float64
```

- 15) One more thing. For data mining we have the wonderful scikit-learn package. For example, see decision trees from <http://scikit-learn.org/stable/modules/tree.html>
- 16) For using both R and Python together, you can use <http://beakernotebook.com/> as it allows you to select kernel specific to each code block, not just the whole notebook like Jupyter does and makes passing of objects very easy between languages.

A 50-page elaborate version of this tutorial is available at <http://www.slideshare.net/ajayohri/a-data-science-tutorial-in-python>. This tutorial was first published on the Wiley web site Statisticsviews.com www.statisticsviews.com/details/feature/8868901/A-Tutorial-on-Python.html

For data scientists working with huge amounts of data, Python is an increasingly credible option to R to try out in production systems.

7

Machine Learning Made Easier

Machine learning is the buzzword of the decade as students and companies vie to get this skill for business applications. However many parts of machine learning are quite easy. In supervised learning, we know what we are trying to predict (a group to class in classification and a number/equation to predict in regression), whereas in unsupervised learning we do not know what is to be predicted (no given tag is there), so we do association analysis and cluster analysis. Text mining on the other hand looks at frequency of words for pattern analysis. Social network analysis looks at relationships between nodes, edges, and actors to see how networks behave. Deep learning is an even more recent case of such advances in techniques.

One of the most widely used techniques is decision trees.

Decision trees in Python (weather dataset)

<https://nbviewer.jupyter.org/gist/decisionstats/47a2324b14ebfd22657b40ec1ae5b480>

```
#rattle package in R has weather dataset
#(see help at http://artax.karlin.mff.cuni.cz/r-help/library/rattle/html/weather.html)
```

```
import os as os                                     In [259]:

import pandas as pd                                 In [260]:

os.getcwd()                                         In [261]:

'/home/ajayohri'                                   Out [261]:

os.listdir()                                         In [262]:
```

Out [262] :

```
[ '.hplip',
  '.xsession-errors.old',
  'VirtualBox VMs',
  'filename.pkl_04.npy',
  '.thunderbird',
  'SVM.R',
  'R',
  'Desktop',
  'filename.pkl_07.npy',
  '.cache',
  '.webex',
  'file.R',
  '.ipython',
  'unique_ids_for_list.html',
  'filename.pkl_11.npy',
  '.Xauthority',
  'Dropbox',
  'examples.desktop',
  'machine learning-plot and bagged pima indians.ipynb',
  'date time.ipynb',
  'Untitled.ipynb',
  '.rstudio-desktop',
  'filename.pkl_01.npy',
  'anaconda3',
  '.dropbox',
  'Music',
  '.pki',
  'rsconnect',
  'GoodReads.ipynb',
  '.config',
  'diamsum.html',
  'filename.pkl_06.npy',
  'data inspection .ipynb',
  '.sudo_as_admin_successful',
  '.continuum',
  '.java',
  'unique ids for list.R',
  '.bashrc-anaconda3.bak',
  '.texmf-var',
  'numpy scipy pandas.ipynb',
  'mozilla.pdf',
  '.dropbox-dist',
```

```

'.bash_logout',
'.jupyter',
'.ecryptfs',
'.dbus',
'.local',
'.lyx',
'.xsession-errors',
'hebrew',
'RCommanderMarkdown.Rmd',
'.bash_history',
'SAS',
'nbr2mp4.sh',
'.adobe',
'.Skype',
'filename.pkl_05.npy',
'.wajig',
'ajay ohri.odt',
'.macromedia',
'.gphoto',
'.oracle_jre_usage',
'machine learning-rattle dataset from R.ipynb',
'.profile',
'file operations.ipynb',
'Documents',
'filename.pkl_09.npy',
'Videos',
'RCommander.R',
'filename.pkl_08.npy',
'.gstreamer-0.10',
'SVM.html',
'.Private',
'RCommander.txt',
're for searching strings.ipynb',
'.Rhistory',
'filename.pkl_02.npy',
'RcmdrMarkdown.Rmd',
'Scikit Tutorial',
'machine learning.ipynb',
'.ivy2',
'assignment2.R',
'assignment2.html',
'filename.pkl_03.npy',
'Public',

```

```
'nbr2mp4.tar',
'RcmdrMarkdown.md',
'.bashrc',
'.mozilla',
'Pictures',
'Data Viz Tutorial.ipynb',
'filename.pkl_10.npy',
'.RData',
'.gconf',
'data transformations.ipynb',
'RcmdrMarkdown.html',
'file.html',
'Scikit Tutorial.ipynb',
'Strings, Lists and Maps.ipynb',
'filename.pkl',
'weather.csv',
'Downloads',
'.gnupg',
'.nano',
'variables in strings.ipynb',
'Templates',
'.ICEauthority',
'.ipynb_checkpoints']
```

In [263]:

```
#Finding only csv files in a directory using os and
glob packages
import glob
```

```
path = os.getcwd()
extension = 'csv'
os.chdir(path)
result = [i for i in glob.glob('*.{}'.format(extension))]
print(result)
```

```
['weather.csv']
```

In [264]:

```
dataframe=pd.read_csv("weather.csv")
```

In [265]:

```
dataframe.head()
```

Out [265] :

Un-named: 0		Date	Location	Min	Max	Rain	Evapo-	Sun-	Wind	Wind	Humidity	Pressure	Pressure	Cloud	Cloud	Temp	Temp	Rain	RISK_	Rain	
				Temp	Temp	fall	ration	shine	Gust	Gust		9 am	3 pm	9 am	3 pm	9 am	3pm	Today	MM	Tomorrow	
0	1	2007-11-01	Canberra	8.0	24.3	0.0	3.4	6.3	NW	30.0	...	29	1019.7	1015.0	7	7	14.4	23.6	No	3.6	Yes
1	2	2007-11-02	Canberra	14.0	26.9	3.6	4.4	9.7	ENE	39.0	...	36	1012.4	1008.4	5	3	17.5	25.7	Yes	3.6	Yes
2	3	2007-11-03	Canberra	13.7	23.4	3.6	5.8	3.3	NW	85.0	...	69	1009.5	1007.2	8	7	15.4	20.2	Yes	39.8	Yes
3	4	2007-11-04	Canberra	13.3	15.5	39.8	7.2	9.1	NW	54.0	...	56	1005.5	1007.0	2	7	13.5	14.1	Yes	2.8	Yes
4	5	2007-11-05	Canberra	7.6	16.1	2.8	5.6	10.6	SSE	50.0	...	49	1018.3	1018.5	7	7	11.1	15.4	Yes	0.0	No

5 rows x 25 columns

In [266]:

```
dataframe.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 366 entries, 0 to 365
Data columns (total 25 columns):
Unnamed: 0      366 non-null int64
Date            366 non-null object
Location        366 non-null object
MinTemp        366 non-null float64
MaxTemp        366 non-null float64
Rainfall       366 non-null float64
Evaporation    366 non-null float64
Sunshine       363 non-null float64
WindGustDir    363 non-null object
WindGustSpeed  364 non-null float64
WindDir9am    335 non-null object
WindDir3pm    365 non-null object
WindSpeed9am   359 non-null float64
WindSpeed3pm   366 non-null int64
Humidity9am    366 non-null int64
Humidity3pm    366 non-null int64
```

	Min Temp	MaxTemp	Rainfall	Evaporation	Sunshine	Wind Gust Speed	Wind Speed 9 am	Wind Speed 3 pm
count	366.000000	366.000000	366.000000	366.000000	363.000000	364.000000	359.000000	366.000000
mean	7.265574	20.550273	1.428415	4.521858	7.909366	39.840659	9.651811	17.986339
std	6.025800	6.690516	4.225800	2.669383	3.481517	13.059807	7.951929	8.856997
min	-5.300000	7.600000	0.000000	0.200000	0.000000	13.000000	0.000000	0.000000
25%	2.300000	15.025000	0.000000	2.200000	NaN	NaN	NaN	11.000000
50%	7.450000	19.650000	0.000000	4.200000	NaN	NaN	NaN	17.000000
75%	12.500000	25.500000	0.200000	6.400000	NaN	NaN	NaN	24.000000
max	20.900000	35.800000	39.800000	13.800000	13.600000	98.000000	41.000000	52.000000

```

Pressure9am      366 non-null float64
Pressure3pm      366 non-null float64
Cloud9am         366 non-null int64
Cloud3pm         366 non-null int64
Temp9am          366 non-null float64
Temp3pm          366 non-null float64
RainToday        366 non-null object
RISK_MM          366 non-null float64
RainTomorrow     366 non-null object
dtypes: float64(12), int64(6), object(7)
memory usage: 71.6+ KB

```

In [267]:

```
dataframe=dataframe.drop('Unnamed: 0', 1)
```

In [268]:

```

dataframe.describe()
/home/ajayohri/anaconda3/lib/python3.5/site-packages/
numpy/lib/function_base.py:3834: RuntimeWarning:
Invalid value encountered in percentile
RuntimeWarning)

```

Out [268]:

Humidity 9 am	Humidity 3 pm	Pressure 9 am	Pressure 3 pm	Cloud 9 am	Cloud 3 pm	Temp 9 am	Temp 3 pm	RISK_MM
366.000000	366.000000	366.000000	366.000000	366.000000	366.000000	366.000000	366.000000	366.000000
72.035519	44.519126	1019.709016	1016.810383	3.890710	4.024590	12.358470	19.230874	1.428415
13.137058	16.850947	6.686212	6.469422	2.956131	2.666268	5.630832	6.640346	4.225800
36.000000	13.000000	996.500000	996.800000	0.000000	0.000000	0.100000	5.100000	0.000000
64.000000	32.250000	1015.350000	1012.800000	1.000000	1.000000	7.625000	14.150000	0.000000
72.000000	43.000000	1020.150000	1017.400000	3.500000	4.000000	12.550000	18.550000	0.000000
81.000000	55.000000	1024.475000	1021.475000	7.000000	7.000000	17.000000	24.000000	0.200000
99.000000	96.000000	1035.700000	1033.200000	8.000000	8.000000	24.700000	34.500000	39.800000

In [269]:

```
dataframe['RainTomorrow'].unique()
```

Out [269]:

```
array(['Yes', 'No'], dtype=object)
```

In [270]:

```
dataframe['RainToday'].unique()
```

Out [270]:

```
array(['No', 'Yes'], dtype=object)
```

In [271]:

```
dataframe['Location'].unique()
```

Out [271]:

```
array(['Canberra'], dtype=object)
```

In [272]:

```
dataframe['Date'].unique()
```

Out [272]:

```
array(['2007-11-01', '2007-11-02', '2007-11-03',
      '2007-11-04',
```

Output truncated by author for publication purposes

```
      '2008-01-04', '2008-01-05', '2008-01-06',
      '2008-01-07',
      '2008-10-30', '2008-10-31'], dtype=object)
```

In [273]:

```
# Bagged Decision Trees for Classification
from sklearn import cross_validation
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

In [274]:

```
dataframe.columns
```

Out [274]:

```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp',
      'Rainfall', 'Evaporation',
```

```

        'Sunshine', 'WindGustDir', 'WindGustSpeed',
        'WindDir9am', 'WindDir3pm',
        'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am',
        'Humidity3pm',
        'Pressure9am', 'Pressure3pm', 'Cloud9am',
        'Cloud3pm', 'Temp9am',
        'Temp3pm', 'RainToday', 'RISK_MM',
        'RainTomorrow'],
        dtype='object')

```

7.1 Deleting Columns We Dont Need in the Final Decision Tree Model

```

In [275]:
del dataframe['Date']

In [276]:
del dataframe['Location']

In [277]:
del dataframe['WindDir9am']

In [278]:
del dataframe['WindSpeed3pm']

In [279]:
del dataframe['WindGustDir']
del dataframe['WindDir3pm']
del dataframe['RISK_MM']

In [280]:
dataframe=dataframe.replace(['Yes', 'No'], [1, 0])
#using replace to change string to numeric values

In [281]:
dataframe=dataframe.dropna()

In [282]:
dataframe.head()

Out [282]:

```

	Min Temp	Max Temp	Rain- fall	Evapo- ration	Sun- shine	Wind Gust Speed	Wind Speed 9 am	Humi- dity 9 am	Humi- dity 3 pm	Pressure 9 am	Pressure 3 pm	Cloud 9 am	Cloud 3 pm	Temp 9 am	Temp 3 pm	Rain Today	Rain Tomor- row
0	8.0	24.3	0.0	3.4	6.3	30.0	6.0	68	29	1019.7	1015.0	7	7	14.4	23.6	0	1
1	14.0	26.9	3.6	4.4	9.7	39.0	4.0	80	36	1012.4	1008.4	5	3	17.5	25.7	1	1
2	13.7	23.4	3.6	5.8	3.3	85.0	6.0	82	69	1009.5	1007.2	8	7	15.4	20.2	1	1
3	13.3	15.5	39.8	7.2	9.1	54.0	30.0	62	56	1005.5	1007.0	2	7	13.5	14.1	1	1
4	7.6	16.1	2.8	5.6	10.6	50.0	20.0	68	49	1018.3	1018.5	7	7	11.1	15.4	1	0

```
In [283]:
len(dataframe)

Out [283]:
354

In [284]:
len(dataframe.columns)

Out [284]:
17

In [285]:
names=dataframe.columns
names

Out [285]:
Index(['MinTemp', 'MaxTemp', 'Rainfall',
      'Evaporation', 'Sunshine',
      'WindGustSpeed', 'WindSpeed9am', 'Humidity9am',
      'Humidity3pm',
      'Pressure9am', 'Pressure3pm', 'Cloud9am',
      'Cloud3pm', 'Temp9am',
      'Temp3pm', 'RainToday', 'RainTomorrow'],
      dtype='object')

In [286]:
dataframe.describe()

Out [286]:
```

	Min Temp	Max Temp	Rain fall	Evaporation	Sunshine	Wind Gust Speed	Wind Speed 9 am	Humidity 9 am
count	354.000000	354.000000	354.000000	354.000000	354.000000	354.000000	354.000000	354.000000
mean	7.362429	20.601412	1.420904	4.558192	7.925424	40.011299	9.666667	71.875706
std	6.010927	6.708966	4.235358	2.667877	3.510039	13.034488	7.978489	13.161939
min	-5.300000	7.600000	0.000000	0.200000	0.000000	13.000000	0.000000	36.000000
25%	2.400000	15.100000	0.000000	2.400000	5.925000	31.000000	6.000000	64.000000
50%	7.500000	19.750000	0.000000	4.200000	8.650000	39.000000	7.000000	72.000000
75%	12.500000	25.500000	0.200000	6.400000	10.600000	46.000000	13.000000	80.000000
max	20.900000	35.800000	39.800000	13.800000	13.600000	98.000000	41.000000	99.000000

```

In [287]:
type(dataframe)

Out[287]:
pandas.core.frame.DataFrame

In [288]:
array = dataframe.values

In [289]:
pd.value_counts(dataframe["RainTomorrow"])

Out[289]:
0      290
1       64
Name: RainTomorrow, dtype: int64

In [290]:
array
Out[290]:
array([[ 8. , 24.3,  0. , ..., 23.6,  0. ,  1. ],
       [14. , 26.9,  3.6, ..., 25.7,  1. ,  1. ],
       [13.7, 23.4,  3.6, ..., 20.2,  1. ,  1. ],
       ...,
       [12.5, 19.9,  0. , ..., 18.3,  0. ,  0. ],
       [12.5, 26.9,  0. , ..., 25.9,  0. ,  0. ],
       [12.3, 30.2,  0. , ..., 28.6,  0. ,  0. ]])

In [291]:
X = array[:,0:16]
Y = array[:,16]
num_folds = 10
num_instances = len(X)
seed = 7

In [292]:
type(X)

Out[292]:

```

Humidity 3 pm	Pressure 9 am	Pressure 3 pm	Cloud 9 am	Cloud 3 pm	Temp 9 am	Temp 3 pm	Rain Today	Rain Tomorrow
354.000000	354.000000	354.000000	354.000000	354.000000	354.000000	354.000000	354.000000	354.000000
44.454802	1019.562147	1016.692090	3.920904	4.019774	12.438701	19.271469	0.180791	0.180791
16.944316	6.602685	6.373679	2.962363	2.672312	5.630160	6.663681	0.385390	0.385390
13.000000	996.500000	996.800000	0.000000	0.000000	0.100000	5.100000	0.000000	0.000000
32.000000	1015.225000	1012.725000	1.000000	1.000000	7.725000	14.300000	0.000000	0.000000
43.000000	1020.000000	1017.200000	4.000000	4.000000	12.600000	18.600000	0.000000	0.000000
54.750000	1024.400000	1021.350000	7.000000	7.000000	17.000000	24.000000	0.000000	0.000000
96.000000	1035.700000	1033.200000	8.000000	8.000000	24.700000	34.500000	1.000000	1.000000

```
numpy.ndarray
```

In [293]:

```
X
```

Out[293]:

```
array([[ 8. , 24.3,  0. , ..., 14.4, 23.6,  0. ],
       [ 14. , 26.9,  3.6, ..., 17.5, 25.7,  1. ],
       [ 13.7, 23.4,  3.6, ..., 15.4, 20.2,  1. ],
       ...,
       [ 12.5, 19.9,  0. , ..., 14.5, 18.3,  0. ],
       [ 12.5, 26.9,  0. , ..., 15.8, 25.9,  0. ],
       [ 12.3, 30.2,  0. , ..., 23.8, 28.6,  0. ]])
```

In [294]:

```
#Y[Y == "Yes"] = 1 An alternative way to make a NumPy array change values
#Y[Y == "No"] = 0
Y
```

Out[294]:

```
array([ 1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,
        --output truncated by author
        0.,  0.,  0.]])
```

In [295]:

```
dtr = tree.DecisionTreeRegressor(max_depth=3)
dtr.fit(X, Y)
```

Out [295]:

```
DecisionTreeRegressor(criterion='mse', max_depth=3, max_features=None,  
                      max_leaf_nodes=None, min_samples_leaf=1, min_samples_split=2,  
                      min_weight_fraction_leaf=0.0, presort=False, random_state=None,  
                      splitter='best')
```

In [296]:

```
# from sklearn.metrics import roc_curve, auc
```

In [297]:

```
#!sudo pip install pydotplus  
# http://scikit-learn.org/stable/auto\_examples/model\_selection/plot\_roc.html  
# http://machinelearningmastery.com/ensemble-machine-learning-algorithms-python-scikit-learn/  
# http://machinelearningmastery.com/compare-machine-learning-algorithms-python-scikit-learn/
```

In [298]:

```
#!pip freeze  
#checking if we have the right packages
```

In [299]:

```
#!pip install --upgrade pip
```

In [300]:

```
#!pip install pydotplus
```

In [301]:

```
import pydotplus as pydot
from IPython.display import Image
from sklearn.externals.six import StringIO
```

In [302]:

```
# Graphviz
#sudo add-apt-repository ppa:gviz-adm/graphviz-dev
# sudo apt-get update
# http://www.graphviz.org/Download\_linux\_ubuntu.php
```

In [303]:

```
dot_data = StringIO()
```

In [304]:

```
tree.export_graphviz(dtr, out_file=dot_data, feature_names=names[:-1])
```

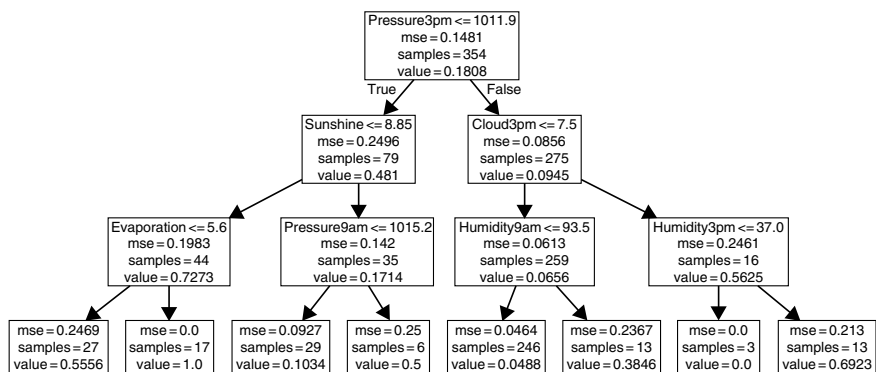
In [305]:

```
graph = pydot.graph_from_dot_data(dot_data.getvalue())
```

In [306]:

```
Image(graph.create_png())
```

Out [306] :



In [307] :

```
kfold = cross_validation.KFold(n=num_instances,
    n_folds=num_folds, random_state=seed)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart,
    n_estimators=num_trees, random_state=seed)
```

In [308] :

```
model
```

Out [308]:

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None,
    criterion='gini', max_depth=None,
    max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
    min_samples_split=2, min_weight_fraction_leaf=0.0,
    presort=False, random_state=None, splitter='best'),
    bootstrap=True, bootstrap_features=False, max_features=1.0,
    max_samples=1.0, n_estimators=100, n_jobs=1, oob_score=False,
    random_state=7, verbose=0, warm_start=False)
```

In [309]:

```
kfold
```

Out [309]:

```
sklearn.cross_validation.KFold(n=354, n_folds=10, shuffle=False, random_state=7)
```

In [310]:

```
results = cross_validation.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
0.850873015873
```

In [311]:

```
results
```

Out [311]:

```
array([ 0.75      ,  0.86111111,  0.69444444,  0.88888889,  0.88571429,
        0.82857143,  0.91428571,  0.85714286,  0.94285714,  0.88571429])
```

Decision trees in Python (2)

<https://nbviewer.jupyter.org/gist/decisionstats/8b762caa7b7deebb68e3f275daf02a9d>

```
# Bagged Decision Trees for Classification
import pandas
from sklearn import cross_validation
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

```
In [2]:
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-
diabetes/pima-indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
```

```
In [3]:
names[:-1]
```

```
Out[3]:
['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age']
```

```
In [4]:
dataframe.head()
```

Out [4] :

	preg	plas	pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [5] :

```
pandas.value_counts(dataframe["class"])
```

Out [5] :

```
0      500
1      268
Name: class, dtype: int64
```

In [6] :

```
array
```

Out [6] :

```
array([[ 6. , 148. , 72. , ..., 0.627, 50. , 1. ],
       [ 1. , 85. , 66. , ..., 0.351, 31. , 0. ],
       [ 8. , 183. , 64. , ..., 0.672, 32. , 1. ],
       ...,
       [ 5. , 121. , 72. , ..., 0.245, 30. , 0. ],
       [ 1. , 126. , 60. , ..., 0.349, 47. , 1. ],
       [ 1. , 93. , 70. , ..., 0.315, 23. , 0. ]])
```

```
X = array[:,0:8]
Y = array[:,8]
num_folds = 10
num_instances = len(X)
seed = 7
```

In [7]:

```
type(X)
```

In [8]:

```
numpy.ndarray
```

Out [8]:

```
X
```

In [9]:

```
array([[ 6.   , 148.   , 72.   , ..., 33.6 , 0.627, 50.   ],
       [ 1.   , 85.   , 66.   , ..., 26.6 , 0.351, 31.   ],
       [ 8.   , 183.   , 64.   , ..., 23.3 , 0.672, 32.   ],
       ...,
       [ 5.   , 121.   , 72.   , ..., 26.2 , 0.245, 30.   ],
       [ 1.   , 126.   , 60.   , ..., 30.1 , 0.349, 47.   ],
       [ 1.   , 93.   , 70.   , ..., 30.4 , 0.315, 23.   ]])
```

Out [9]:

```
Y
```

In [10]:

```
array([ 1.,  0.,  1.,  0.,  1.,  0.,  1.,  0.,  1.,  1.,  0.,  1.,  0.,
        1.,  1.,  1.,  1.,  1.,  0.,  1.,  0.,  0.,  1.,  1.,  1.,  1.,
        1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,
        1.,  0.,  0.,  0.,  1.,  0.,  1.,  0.,  0.,  1.,  0.,  0.,  0.,
        0.,  1.,  0.,  0.,  0.,  1.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,
        1.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,
        0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,  1.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  1.,  1.,  1.,  0.,  0.,  1.,  1.,  1.,
        0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  1.,
        1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,
        0.,  1.,  1.,  0.,  0.,  0.,  1.,  0.,  1.,  0.,  1.,  0.,  0.,
        0.,  0.,  0.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,  1.,  1.,  0.,
        1.,  0.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,
        0.,  1.,  0.,  0.,  0.,  1.,  1.,  1.,  1.,  0.,  1.,  1.,  1.,
        1.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  1.,  1.,  0.,  0.,
        0.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  1.,  1.,  0.,  1.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,
        0.,  1.,  0.,  0.,  1.,  0.,  1.,  0.,  0.,  1.,  1.,  1.,  0.,
        0.,  1.,  0.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,
```

Out [10] :

1.,	1.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	1.,	0.,
0.,	0.,	0.,	0.,	1.,	1.,	1.,	0.,	1.,	1.,	0.,	0.,	0.,	1.,
0.,	0.,	1.,	0.,	0.,	1.,	1.,	0.,	0.,	0.,	0.,	0.,	1.,	0.,
0.,	1.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	1.,	1.,	1.,	1.,	0.,
0.,	1.,	0.,	0.,	1.,	0.,	0.,	1.,	0.,	1.,	1.,	0.,	0.,	1.,
0.,	1.,	0.,	1.,	0.,	1.,	1.,	0.,	0.,	0.,	0.,	0.,	1.,	1.,
0.,	1.,	0.,	1.,	0.,	0.,	0.,	0.,	1.,	1.,	0.,	1.,	0.,	0.,
1.,	0.,	0.,	0.,	0.,	0.,	1.,	0.,	0.,	0.,	0.,	0.,	1.,	0.,
0.,	1.,	1.,	1.,	0.,	0.,	1.,	0.,	0.,	1.,	0.,	0.,	0.,	0.,
1.,	0.,	0.,	1.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,
1.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	1.,	0.,	0.,	0.,	0.,	1.,
0.,	0.,	0.,	1.,	1.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	1.,
0.,	0.,	0.,	0.,	1.,	0.,	0.,	0.,	1.,	0.,	0.,	0.,	0.,	1.,
0.,	0.,	0.,	1.,	0.,	0.,	0.,	0.,	1.,	1.,	0.,	0.,	0.,	0.,
0.,	0.,	0.,	1.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,
0.,	0.,	1.,	0.,	0.,	0.,	1.,	1.,	1.,	1.,	0.,	0.,	0.,	1.,
1.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,
0.,	1.,	1.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	1.,	0.,	0.,	0.,
0.,	0.,	0.,	0.,	0.,	1.,	0.,	1.,	1.,	0.,	0.,	0.,	0.,	1.,
0.,	1.,	0.,	1.,	0.,	1.,	0.,	1.,	0.,	0.,	1.,	0.,	0.,	0.,
1.,	0.,	0.,	0.,	0.,	1.,	1.,	0.,	1.,	0.,	0.,	0.,	0.,	0.,
1.,	1.,	0.,	1.,	0.,	0.,	0.,	1.,	1.,	0.,	0.,	0.,	0.,	0.,
0.,	0.,	0.,	0.,	0.,	0.,	1.,	0.,	0.,	0.,	0.,	1.,	0.,	0.,
0.,	1.,	0.,	0.,	0.,	1.,	0.,	0.,	0.,	1.,	1.,	1.,	1.,	0.,
0.,	0.,	0.,	0.,	0.,	1.,	0.,	0.,	0.,	1.,	0.,	1.,	1.,	1.,
1.,	1.,	0.,	1.,	1.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	0.,	1.,
1.,	0.,	1.,	0.,	0.,	1.,	0.,	1.,	0.,	0.,	0.,	0.,	0.,	0.,

```

1., 0., 1., 0., 1., 0., 1., 1., 0., 0., 0., 0., 1.,
1., 0., 0., 0., 1., 0., 1., 1., 0., 0., 1., 0., 0.,
1., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0.,
0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 1., 1.,
0., 0., 1., 0., 0., 1., 0., 1., 1., 1., 0., 0., 1.,
1., 1., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 1., 0.]

```

In [11]:

```

dtr = tree.DecisionTreeRegressor(max_depth=3)
dtr.fit(X, Y)

```

Out[11]:

```

DecisionTreeRegressor(criterion='mse', max_depth=3, max_features=None,
                      max_leaf_nodes=None, min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                      splitter='best')

```

In [12]:

```

# from sklearn.metrics import roc_curve, auc

```

In [13]:

```

#!sudo pip install pydotplus
# http://scikit-learn.org/stable/auto\_examples/model\_selection/plot\_roc.html
# http://machinelearningmastery.com/ensemble-machine-learning-algorithms-python-scikit-learn/
# http://machinelearningmastery.com/compare-machine-learning-algorithms-python-scikit-learn/

```

```

In [14]:
#!pip freeze
#checking if we have the right packages

In [15]:
#!pip install --upgrade pip

In [16]:
#!pip install pydotplus

In [17]:
import pydotplus as pydot
from IPython.display import Image
from sklearn.externals.six import StringIO

In [18]:
# Graphviz
#sudo add-apt-repository ppa:gviz-adm/graphviz-dev
# sudo apt-get update
# http://www.graphviz.org/Download\_linux\_ubuntu.php

In [19]:
dot_data = StringIO()

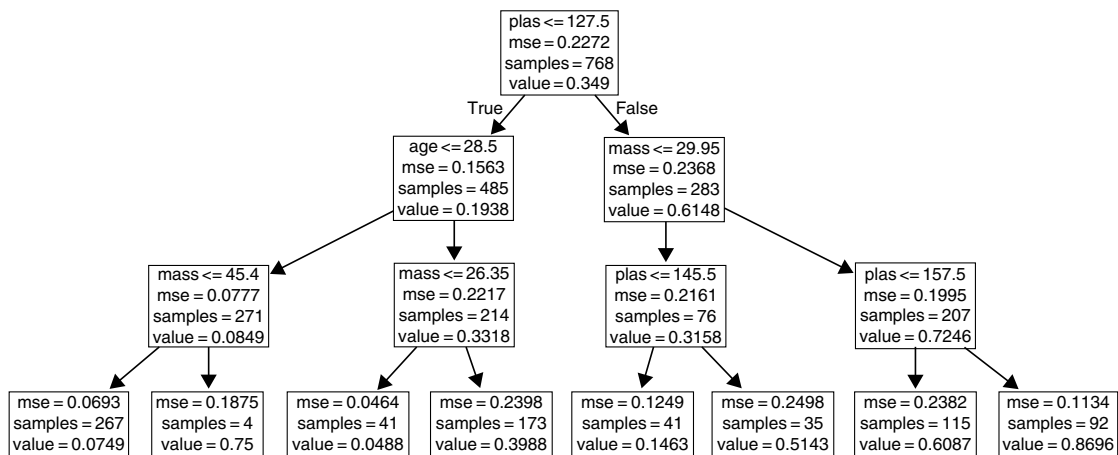
In [20]:
tree.export_graphviz(dtr, out_file=dot_data,
    feature_names=names[:-1])

In [21]:
graph = pydot.graph_from_dot_data(dot_data.getvalue())

In [22]:
Image(graph.create_png())

```

Out [22] :



In [23] :

```
kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, random_state=seed)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees,
    random_state=seed)
```

In [24]:

Out [24]:

In [25]:

Out [25]:

In [26]:

In [27]:

Out [27]:

```
model

BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None,
    criterion='gini', max_depth=None,
    max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
    min_samples_split=2, min_weight_fraction_leaf=0.0,
    presort=False, random_state=None, splitter='best'),
    bootstrap=True, bootstrap_features=False, max_features=1.0,
    max_samples=1.0, n_estimators=100, n_jobs=1, oob_score=False,
    random_state=7, verbose=0, warm_start=False)

kfold

sklearn.cross_validation.KFold(n=768, n_folds=10, shuffle=False, random_state=7)

results = cross_validation.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
0.770745044429

results

array([ 0.67532468,  0.81818182,  0.75324675,  0.63636364,  0.81818182,
        0.81818182,  0.85714286,  0.85714286,  0.69736842,  0.77631579])
```

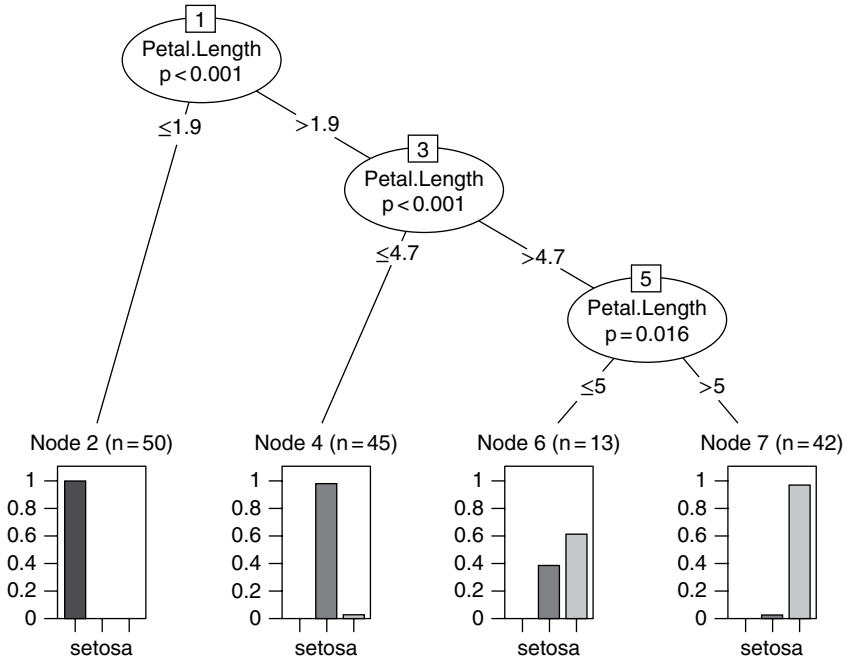
7.1.1 Decision Trees in R

Decision trees in R can be done through multiple packages. Primary are conditional, traditional, and CHAID.

See <http://rpubs.com/newajay/classification> and <http://rpubs.com/newajay/partyR>

```
#install.packages("party")
library(party)
## Loading required package: grid
## Loading required package: mvtnorm
## Loading required package: modeltools
## Loading required package: stats4
## Loading required package: strucchange
## Loading required package: zoo
##
## Attaching package: 'zoo'
## The following objects are masked from
  'package:base':
##
##      as.Date, as.Date.numeric
## Loading required package: sandwich
data("iris")
names(iris)
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length"
  "Petal.Width"
## [5] "Species"
fit2 <- ctree(Species ~ Sepal.Length + Petal.Length +
  Sepal.Width ,
              data=iris)
```

```
plot(fit2)
```



```
print(fit2)
##
##   Conditional inference tree with 4 terminal nodes
##
## Response:  Species
## Inputs:  Sepal.Length, Petal.Length, Sepal.Width
## Number of observations:  150
##
## 1) Petal.Length <= 1.9; criterion = 1, statistic =
   140.264
##   2)*  weights = 50
## 1) Petal.Length > 1.9
##   3) Petal.Length <= 4.7; criterion = 1, statistic
   = 61.228
##   4)*  weights = 45
##   3) Petal.Length > 4.7
##   5) Petal.Length <= 5; criterion = 0.984,
   statistic = 7.701
##   6)*  weights = 13
```

```

##      5) Petal.Length > 5
##      7)* weights = 42

nodes(fit2,1)
## [[1]]
## 1) Petal.Length <= 1.9; criterion = 1, statistic =
  140.264
## 2)* weights = 50
## 1) Petal.Length > 1.9
## 3) Petal.Length <= 4.7; criterion = 1, statistic
  = 61.228
## 4)* weights = 45
## 3) Petal.Length > 4.7
## 5) Petal.Length <= 5; criterion = 0.984,
  statistic = 7.701
## 6)* weights = 13
## 5) Petal.Length > 5
## 7)* weights = 42

nodes(fit2,3)
## [[1]]
## 3) Petal.Length <= 4.7; criterion = 1, statistic =
  61.228
## 4)* weights = 45
## 3) Petal.Length > 4.7
## 5) Petal.Length <= 5; criterion = 0.984, statistic
  = 7.701
## 6)* weights = 13
## 5) Petal.Length > 5
## 7)* weights = 42

table(Predict(fit2), iris$Species)
##
##          setosa versicolor virginica
## setosa      50          0          0
## versicolor   0         44          1
## virginica    0          6         49

#install.packages("randomForest")
library(randomForest)
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug
  fixes.

```

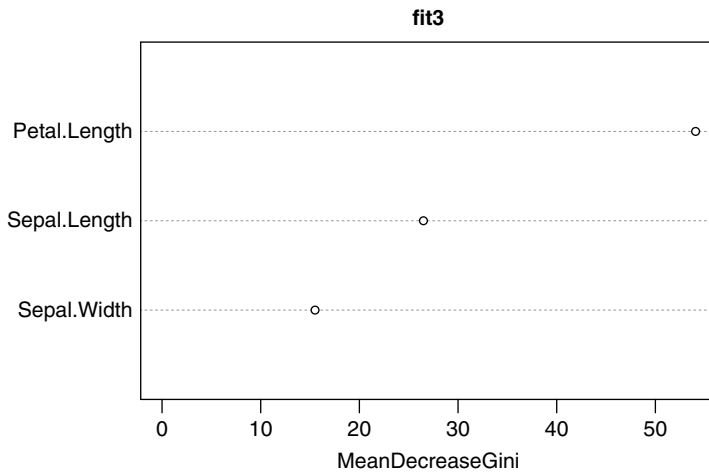
```

fit3 <- randomForest(Species ~ Sepal.Length + Petal.
  Length + Sepal.Width ,
                    data=iris)

print(fit3)
##
## Call:
## randomForest(formula = Species ~ Sepal.Length +
  Petal.Length + Sepal.Width, data = iris)
##
## Type of random forest: classification
##
## Number of trees: 500
## No. of variables tried at each split: 1
##
## OOB estimate of error rate: 7.33%
## Confusion matrix:
##
##      setosa versicolor virginica class.error
## setosa      50         0         0         0.00
## versicolor   0        44         6         0.12
## virginica    0         5        45         0.10

importance(fit3)
##              MeanDecreaseGini
## Sepal.Length      26.45171
## Petal.Length      54.09109
## Sepal.Width       15.53006
# plot(fit3)
varImpPlot(fit3)

```



```

iris$predicted.response <- predict(fit3 ,iris)
library(e1071)
#install.packages("caret")
library(caret)
## Loading required package: lattice
## Loading required package: ggplot2
##
## Attaching package: 'ggplot2'
## The following object is masked from 'package:
  random Forest':
##
##      margin
confusionMatrix(data=iris$predicted.response,
                 reference=iris$Species,
                 positive='yes')
## Confusion Matrix and Statistics
##
##              Reference
## Prediction  setosa versicolor virginica
##   setosa      50         0         0
##   versicolor   0         50         0
##   virginica    0         0         50
##
## Overall Statistics
##
##              Accuracy : 1
##              95% CI   : (0.9757, 1)
##   No Information Rate : 0.3333
##   P-Value [Acc > NIR] : < 2.2e-16
##
##              Kappa : 1
##   McNemar's Test P-Value : NA
##
## Statistics by Class:
##

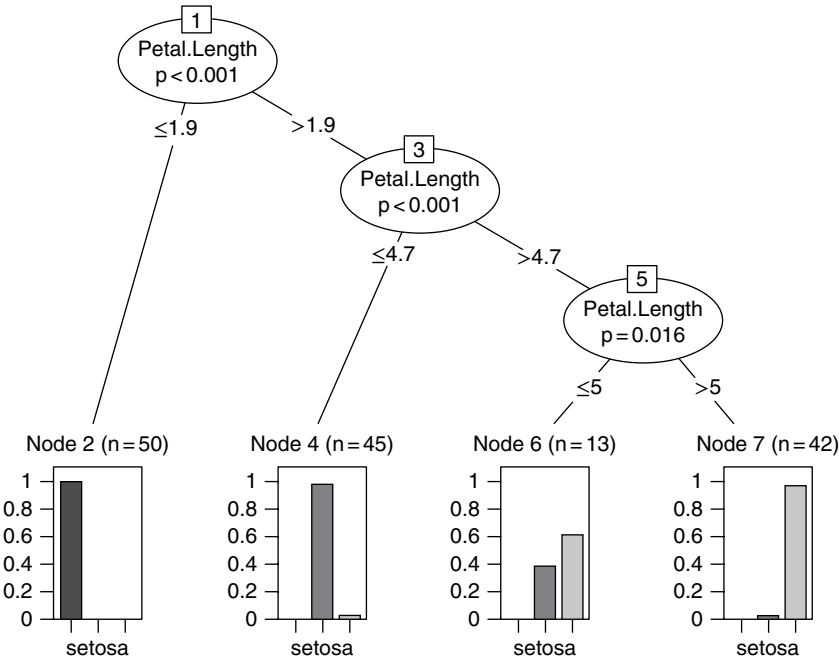
```

	Class: setosa	Class: versicolor	Class: virginica
## Sensitivity	1.0000	1.0000	1.0000
## Specificity	1.0000	1.0000	1.0000
## Pos Pred Value	1.0000	1.0000	1.0000
## Neg Pred Value	1.0000	1.0000	1.0000
## Prevalence	0.3333	0.3333	0.3333
## Detection Rate	0.3333	0.3333	0.3333
## Detection Prevalence	0.3333	0.3333	0.3333
## Balanced Accuracy	1.0000	1.0000	1.0000

And

```
library(party)
## Loading required package: grid
## Loading required package: mvtnorm
## Loading required package: modeltools
## Loading required package: stats4
## Loading required package: strucchange
## Loading required package: zoo
##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##      as.Date, as.Date.numeric
## Loading required package: sandwich
data(iris)
```

```
fit2 <- ctree(Species ~ Sepal.Length + Petal.Length +
  Sepal.Width ,
  data=iris)
plot(fit2)
```



```
table(Predict(fit2), iris$Species)
##
##          setosa versicolor virginica
##  setosa      50          0          0
##  versicolor   0         44          1
##  virginica    0          6         49
library(randomForest)
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
fit3 <- randomForest(Species ~ Sepal.Length + Petal.
  Length + Sepal.Width ,
  data=iris)
print(fit3)
##
## Call:
##  randomForest(formula = Species ~ Sepal.Length +
  Petal.Length +      Sepal.Width, data = iris)
```

```

##                               Type of random forest:
      classification
##                               Number of trees: 500
## No. of variables tried at each split: 1
##
##               OOB estimate of error rate: 6.67%
## Confusion matrix:
##               setosa versicolor virginica class.error
## setosa         50           0           0           0.00
## versicolor     0           44           6           0.12
## virginica      0           4           46           0.08
library(e1071)
#install.packages("caret")
library(caret)
## Loading required package: lattice
## Loading required package: ggplot2
##
## Attaching package: 'ggplot2'
## The following object is masked from
  'package:randomForest':
##
##      margin
iris$predicted.response <- predict(fit3 ,iris)
confusionMatrix(data=iris$predicted.response,
                 reference=iris$Species,
                 positive='yes')
## Confusion Matrix and Statistics
##
##               Reference
## Prediction   setosa versicolor virginica
##   setosa      50           0           0
##   versicolor  0           49           0
##   virginica   0           1           50
##
## Overall Statistics
##
##               Accuracy : 0.9933
##               95% CI : (0.9634, 0.9998)
##   No Information Rate : 0.3333
##   P-Value [Acc > NIR] : < 2.2e-16
##
##               Kappa : 0.99
##   McNemar's Test P-Value : NA
##

```

```
## Statistics by Class:
##
##
##          Class: setosa Class: versicolor Class: virginica
## Sensitivity          1.0000          0.9800          1.0000
## Specificity          1.0000          1.0000          0.9900
## Pos Pred Value       1.0000          1.0000          0.9804
## Neg Pred Value       1.0000          0.9901          1.0000
## Prevalence           0.3333          0.3333          0.3333
## Detection Rate       0.3333          0.3267          0.3333
## Detection Prevalence 0.3333          0.3267          0.3400
## Balanced Accuracy     1.0000          0.9900          0.9950
x = iris[,-5]
y = iris$Species

model = train(x,y,'nb',trControl=trainControl(method='cv',number=10))
## Loading required package: klaR
## Loading required package: MASS
model
## Naive Bayes
##
## 150 samples
##   5 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...
```

```
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy  Kappa
##   FALSE      0.9866667  0.98
##   TRUE       0.9800000  0.97
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##   parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = FALSE
##   and adjust = 1.
predict(model$finalModel,x)
## $class
##   [1] setosa      setosa      setosa      setosa      setosa      setosa
##   [7] setosa      setosa      setosa      setosa      setosa      setosa
##  [13] setosa      setosa      setosa      setosa      setosa      setosa
##  [19] setosa      setosa      setosa      setosa      setosa      setosa
##  [25] setosa      setosa      setosa      setosa      setosa      setosa
##  [31] setosa      setosa      setosa      setosa      setosa      setosa
##  [37] setosa      setosa      setosa      setosa      setosa      setosa
##  [43] setosa      setosa      setosa      setosa      setosa      setosa
##  [49] setosa      setosa      versicolor versicolor versicolor versicolor
##  [55] versicolor versicolor versicolor versicolor versicolor versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor versicolor
##  [67] versicolor versicolor versicolor versicolor versicolor versicolor
##  [73] versicolor versicolor versicolor versicolor versicolor versicolor
```

```
## [79] versicolor versicolor versicolor versicolor versicolor virginica
## [85] versicolor versicolor versicolor versicolor versicolor versicolor
## [91] versicolor versicolor versicolor versicolor versicolor versicolor
## [97] versicolor versicolor versicolor versicolor virginica virginica
## [103] virginica virginica virginica virginica virginica virginica
## [109] virginica virginica virginica virginica virginica virginica
## [115] virginica virginica virginica virginica virginica virginica
## [121] virginica virginica virginica virginica virginica virginica
## [127] virginica virginica virginica virginica virginica virginica
## [133] virginica virginica virginica virginica virginica virginica
## [139] virginica virginica virginica virginica virginica virginica
## [145] virginica virginica virginica virginica virginica virginica
## Levels: setosa versicolor virginica
##
## $posterior
##          setosa  versicolor   virginica
## [1,] 1.000000e+00 2.981309e-21 2.152373e-28
## [2,] 1.000000e+00 3.169312e-20 6.938030e-28
## [3,] 1.000000e+00 2.367113e-21 7.240956e-29
## [4,] 1.000000e+00 3.069606e-20 8.690636e-28
## [5,] 1.000000e+00 1.017337e-21 8.885794e-29
## [6,] 1.000000e+00 2.717732e-17 4.344285e-24
## [7,] 1.000000e+00 2.321639e-20 7.988271e-28
## [8,] 1.000000e+00 1.390751e-20 8.166995e-28
## [9,] 1.000000e+00 1.990156e-20 3.606469e-28
## [10,] 1.000000e+00 7.378931e-21 3.615492e-28
## [11,] 1.000000e+00 9.396089e-21 1.474623e-27
## [12,] 1.000000e+00 3.461964e-20 2.093627e-27
## [13,] 1.000000e+00 2.804520e-21 1.010192e-28
```

```

## [14,] 1.000000e+00 1.799033e-22 6.060578e-30
## [15,] 1.000000e+00 5.533879e-22 2.485033e-28
## [16,] 1.000000e+00 6.273863e-20 4.509864e-26
## [17,] 1.000000e+00 1.106658e-19 1.282419e-26
## [18,] 1.000000e+00 4.841773e-20 2.350011e-27
## [19,] 1.000000e+00 1.126175e-17 2.567180e-24
## [20,] 1.000000e+00 1.808513e-20 1.963924e-27
## [21,] 1.000000e+00 2.178382e-18 2.013989e-25
## [22,] 1.000000e+00 1.210057e-18 7.788592e-26
## [23,] 1.000000e+00 4.535220e-23 3.130074e-30
## [24,] 1.000000e+00 3.147327e-14 8.175305e-22
## [25,] 1.000000e+00 1.838507e-17 1.553757e-24
## [26,] 1.000000e+00 6.873990e-19 1.830374e-26
## [27,] 1.000000e+00 3.192598e-17 1.045146e-24
## [28,] 1.000000e+00 1.542562e-20 1.274394e-27
## [29,] 1.000000e+00 8.833285e-21 5.368077e-28
## [30,] 1.000000e+00 9.557935e-20 3.652571e-27
## [31,] 1.000000e+00 2.166837e-19 6.730536e-27
## [32,] 1.000000e+00 3.940500e-17 1.546678e-24
## [33,] 1.000000e+00 1.609092e-23 1.013278e-29
## [34,] 1.000000e+00 7.222217e-23 4.261853e-29
## [35,] 1.000000e+00 6.289348e-20 1.831694e-27
## [36,] 1.000000e+00 2.850926e-21 8.874002e-29
## [37,] 1.000000e+00 7.746279e-21 7.235628e-28
## [38,] 1.000000e+00 8.623934e-23 1.223633e-29
## [39,] 1.000000e+00 4.612936e-21 9.655450e-29
## [40,] 1.000000e+00 2.009325e-20 1.237755e-27
## [41,] 1.000000e+00 1.300634e-20 5.657689e-28
## [42,] 1.000000e+00 1.577617e-18 5.717219e-27
## [43,] 1.000000e+00 1.494911e-21 4.800333e-29
## [44,] 1.000000e+00 1.076475e-13 3.721344e-21
## [45,] 1.000000e+00 1.357569e-15 1.708326e-22
## [46,] 1.000000e+00 3.882113e-19 5.587814e-27
## [47,] 1.000000e+00 5.086735e-21 8.960156e-28
## [48,] 1.000000e+00 5.012793e-21 1.636566e-28
## [49,] 1.000000e+00 5.717245e-21 8.231337e-28
## [50,] 1.000000e+00 7.713456e-21 3.349997e-28
## [51,] 6.225045e-110 9.997479e-01 2.520714e-04
## [52,] 8.570847e-103 9.999382e-01 6.175742e-05
## [53,] 1.215697e-123 9.988066e-01 1.193433e-03
## [54,] 1.152529e-72 1.000000e+00 3.867457e-08
## [55,] 1.581871e-108 9.999467e-01 5.331762e-05
## [56,] 1.972563e-92 9.999990e-01 1.020052e-06
## [57,] 7.022043e-116 9.994729e-01 5.271422e-04

```

```

## [58,] 2.601790e-37 1.000000e+00 3.183182e-10
## [59,] 6.767475e-100 9.999892e-01 1.078777e-05
## [60,] 5.102801e-72 9.999999e-01 1.093625e-07
## [61,] 7.504643e-44 1.000000e+00 3.208078e-10
## [62,] 4.917431e-89 9.999958e-01 4.245756e-06
## [63,] 4.725838e-63 1.000000e+00 7.695236e-09
## [64,] 1.089964e-106 9.999846e-01 1.544774e-05
## [65,] 4.887696e-58 1.000000e+00 3.060696e-08
## [66,] 1.579126e-95 9.999779e-01 2.214465e-05
## [67,] 1.379538e-100 9.999896e-01 1.037882e-05
## [68,] 2.067506e-65 1.000000e+00 2.048320e-08
## [69,] 6.720035e-104 9.999940e-01 5.953299e-06
## [70,] 3.077859e-61 1.000000e+00 8.918403e-09
## [71,] 6.643323e-130 9.947081e-01 5.291896e-03
## [72,] 1.273962e-73 9.999998e-01 2.303272e-07
## [73,] 3.635930e-122 9.999168e-01 8.322850e-05
## [74,] 1.343761e-98 9.999979e-01 2.106614e-06
## [75,] 3.069700e-86 9.999982e-01 1.764714e-06
## [76,] 2.623805e-95 9.999861e-01 1.392484e-05
## [77,] 1.747438e-114 9.998994e-01 1.006309e-04
## [78,] 8.854376e-138 9.883852e-01 1.161480e-02
## [79,] 5.212805e-102 9.999850e-01 1.501794e-05
## [80,] 1.468423e-44 1.000000e+00 1.634262e-09
## [81,] 1.277115e-57 1.000000e+00 4.592000e-09
## [82,] 8.948524e-51 1.000000e+00 1.778126e-09
## [83,] 3.517650e-65 1.000000e+00 3.430714e-08
## [84,] 2.726206e-135 3.076150e-02 9.692385e-01
## [85,] 4.238525e-100 9.999916e-01 8.405606e-06
## [86,] 1.332644e-105 9.998521e-01 1.478728e-04
## [87,] 2.875899e-113 9.997405e-01 2.595475e-04
## [88,] 4.973519e-91 9.999993e-01 7.170422e-07
## [89,] 2.070566e-75 9.999998e-01 2.429045e-07
## [90,] 2.273490e-72 9.999999e-01 5.500821e-08
## [91,] 5.215785e-84 9.999998e-01 1.520450e-07
## [92,] 5.960938e-102 9.999882e-01 1.182936e-05
## [93,] 5.251986e-69 1.000000e+00 4.173171e-08
## [94,] 1.360017e-37 1.000000e+00 2.771698e-10
## [95,] 6.219736e-80 9.999998e-01 2.006854e-07
## [96,] 1.453599e-75 9.999998e-01 1.800789e-07
## [97,] 8.474883e-80 9.999997e-01 3.376164e-07
## [98,] 1.875115e-85 9.999988e-01 1.164505e-06
## [99,] 5.826890e-33 1.000000e+00 3.157898e-10
## [100,] 4.078752e-76 9.999998e-01 1.832703e-07

```

```

## [101,] 3.993755e-252 2.062063e-12 1.000000e+00
## [102,] 1.262363e-152 5.598282e-04 9.994402e-01
## [103,] 2.460661e-219 4.654977e-09 1.000000e+00
## [104,] 2.871277e-176 4.592219e-05 9.999541e-01
## [105,] 8.299887e-217 6.350771e-09 1.000000e+00
## [106,] 1.371182e-270 7.614910e-12 1.000000e+00
## [107,] 7.258155e-109 4.096782e-01 5.903218e-01
## [108,] 3.741935e-227 3.564099e-08 1.000000e+00
## [109,] 5.567821e-191 1.165303e-05 9.999883e-01
## [110,] 2.052443e-263 4.923323e-14 1.000000e+00
## [111,] 8.673566e-162 9.795170e-06 9.999902e-01
## [112,] 4.233346e-166 6.357027e-05 9.999364e-01
## [113,] 4.360086e-193 1.246172e-07 9.999999e-01
## [114,] 6.229150e-154 2.887102e-04 9.997113e-01
## [115,] 2.201429e-189 2.786497e-08 1.000000e+00
## [116,] 2.949946e-194 1.225678e-08 1.000000e+00
## [117,] 2.915226e-171 4.314790e-05 9.999569e-01
## [118,] 1.347608e-284 5.745992e-14 1.000000e+00
## [119,] 2.786402e-309 2.302938e-14 1.000000e+00
## [120,] 3.307637e-125 3.037334e-01 6.962666e-01
## [121,] 2.194169e-220 3.424331e-10 1.000000e+00
## [122,] 3.376038e-148 3.083326e-04 9.996917e-01
## [123,] 6.251357e-272 2.341745e-11 1.000000e+00
## [124,] 5.094321e-138 3.139418e-03 9.968606e-01
## [125,] 6.315724e-204 2.601027e-08 1.000000e+00
## [126,] 5.257396e-206 1.901615e-07 9.999998e-01
## [127,] 1.851587e-132 5.186340e-03 9.948137e-01
## [128,] 9.865968e-137 2.542930e-03 9.974571e-01
## [129,] 5.230872e-197 2.791477e-07 9.999997e-01
## [130,] 7.020556e-182 1.647833e-05 9.999835e-01
## [131,] 6.306827e-221 2.428996e-08 1.000000e+00
## [132,] 2.539020e-250 9.337782e-12 1.000000e+00
## [133,] 2.210816e-204 4.000640e-08 1.000000e+00
## [134,] 3.732889e-131 4.709186e-02 9.529081e-01
## [135,] 1.561444e-153 1.886075e-02 9.811392e-01
## [136,] 7.419068e-252 2.896100e-12 1.000000e+00
## [137,] 1.004503e-218 1.948671e-10 1.000000e+00
## [138,] 1.349608e-170 4.383372e-05 9.999562e-01
## [139,] 2.480958e-131 4.974797e-03 9.950252e-01
## [140,] 8.440522e-188 1.353834e-07 9.999999e-01
## [141,] 2.334365e-221 1.491244e-10 1.000000e+00
## [142,] 2.179140e-186 1.270533e-08 1.000000e+00
## [143,] 1.262363e-152 5.598282e-04 9.994402e-01

```

```

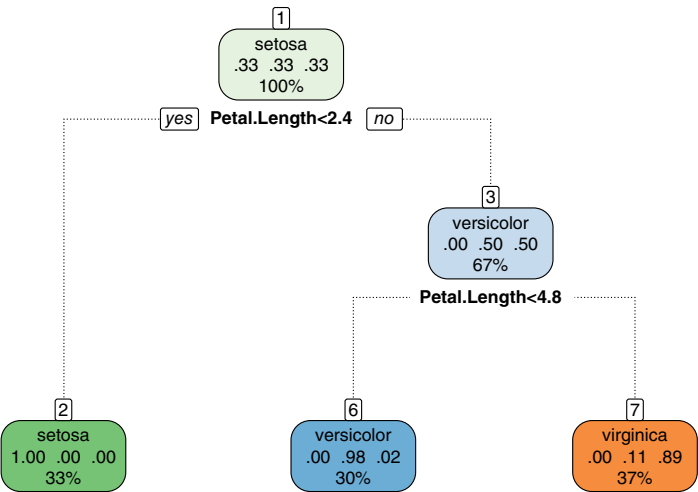
## [144,] 3.426814e-232 1.319403e-10 1.000000e+00
## [145,] 2.011574e-235 5.241271e-12 1.000000e+00
## [146,] 1.078519e-190 1.583110e-08 1.000000e+00
## [147,] 1.091014e-149 5.695800e-04 9.994304e-01
## [148,] 1.847697e-167 8.800598e-06 9.999912e-01
## [149,] 1.439996e-198 6.768314e-09 1.000000e+00
## [150,] 2.944253e-146 1.272237e-03 9.987278e-01
table(predict(model$finalModel,x)$class,y)
##           y
##           setosa versicolor virginica
## setosa           50             0         0
## versicolor        0             49         0
## virginica          0             1        50
naive_iris <- NaiveBayes(iris$Species ~ ., data = iris)
#plot(naive_iris)
library(rpart)
fit4 <- rpart(Species ~ Sepal.Length + Petal.Length + Sepal.Width ,
              data=iris,method = "class")

print(fit4)
## n= 150
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##
## 1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
##   2) Petal.Length< 2.45 50   0 setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.45 100  50 versicolor (0.00000000 0.50000000 0.50000000)

```

```
##      6) Petal.Length< 4.75 45    1 versicolor (0.00000000 0.97777778 0.02222222) *
##      7) Petal.Length>=4.75 55    6 virginica (0.00000000 0.10909091 0.89090909) *
library(rattle)
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
fancyRpartPlot(fit4)
```



```

#rest models are in http://rpubs.com/newajay/chaid

#SOURCE https://sites.google.com/site/kittipat/rtechniques/usingchaiddecisiontreeinr

#install.packages("CHAID", repos="http://R-Forge.
  R-project.org")

library("CHAID")
## Loading required package: partykit
##
## Attaching package: 'partykit'
## The following objects are masked from 'package:party':
##
##      cforest, ctree, ctree_control, edge_simple, mob, mob_control,
##      node_barplot, node_bivplot, node_boxplot, node_inner,
##      node_surv, node_terminal
data(iris)
str(iris)
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1
...
ctrl <- chaid:control(minsplit = 20, minbucket = 5, minprob = 0)

```

```

iris=lapply(iris,as.factor)
chaidiris <- chaid(Species ~ Sepal.Length + Petal.Length + Sepal.Width + Petal.Width ,
                  data=iris, control = ctrl)

print(chaidiris)
##
## Model formula:
## Species ~ Sepal.Length + Petal.Length + Sepal.Width + Petal.Width
##
## Fitted party:
## [1] root
## |   [2] Petal.Width in 0.1, 0.2, 0.3, 0.4, 0.5, 0.6: setosa (n = 50, err = 0.0%)
## |   [3] Petal.Width in 1, 1.1, 1.2, 1.3: versicolor (n = 28, err = 0.0%)
## |   [4] Petal.Width in 1.4, 1.5, 1.6, 1.7: versicolor (n = 26, err = 19.2%)
## |   [5] Petal.Width in 1.8, 1.9, 2, 2.1, 2.2, 2.3, 2.4, 2.5: virginica (n = 46,
    err = 2.2%)
##
## Number of inner nodes:    1
## Number of terminal nodes: 4

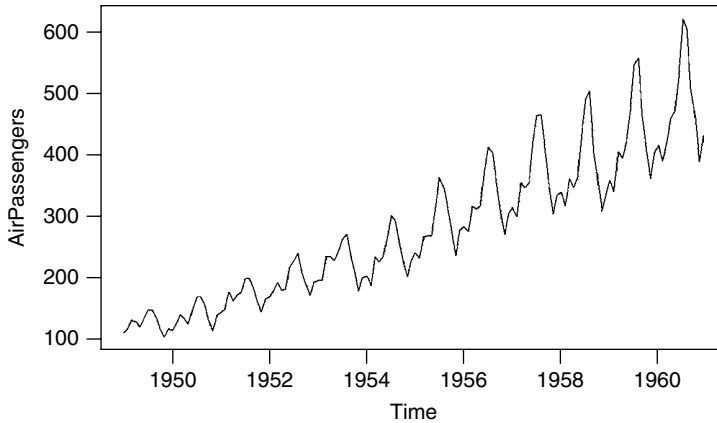
```

7.2 Time Series

Time series forecasting is very easily done in R thanks to `auto.arima` in `forecast` package. In Python it is not automated so easily though `statsmodels` has libraries for it. The reader is thus advised to forecast in R and then apply model in Python.

R Code from <http://rpubs.com/newajay/ts>

```
data("AirPassengers")
library(forecast)
## Loading required package: zoo
##
## Attaching package: 'zoo'
## The following objects are masked from
##   'package:base':
##
##     as.Date, as.Date.numeric
## Loading required package: timeDate
## This is forecast 7.3
ts.plot(AirPassengers)
```



decompose(AirPassengers)

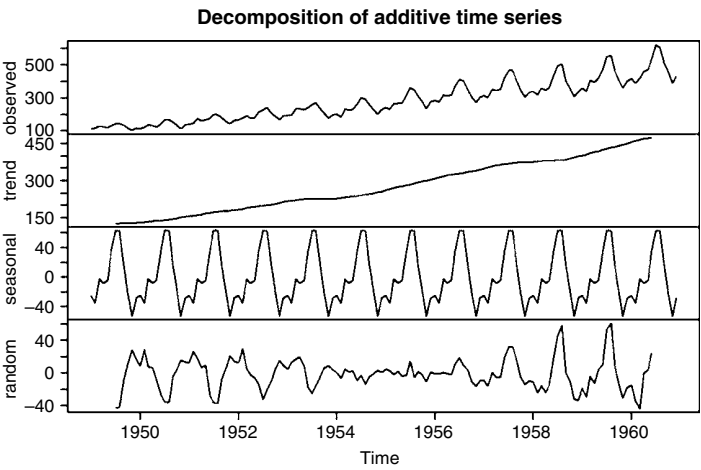
## \$x													
##		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
##	1949	112	118	132	129	121	135	148	148	136	119	104	118
##	1950	115	126	141	135	125	149	170	170	158	133	114	140
##	1951	145	150	178	163	172	178	199	199	184	162	146	166
##	1952	171	180	193	181	183	218	230	242	209	191	172	194
##	1953	196	196	236	235	229	243	264	272	237	211	180	201
##	1954	204	188	235	227	234	264	302	293	259	229	203	229
##	1955	242	233	267	269	270	315	364	347	312	274	237	278
##	1956	284	277	317	313	318	374	413	405	355	306	271	306
##	1957	315	301	356	348	355	422	465	467	404	347	305	336
##	1958	340	318	362	348	363	435	491	505	404	359	310	337
##	1959	360	342	406	396	420	472	548	559	463	407	362	405
##	1960	417	391	419	461	472	535	622	606	508	461	390	432
##													
## \$seasonal													
##		Jan		Feb		Mar		Apr		May		Jun	
##	1949	-24.748737		-36.188131		-2.241162		-8.036616		-4.506313		35.402778	
##	1950	-24.748737		-36.188131		-2.241162		-8.036616		-4.506313		35.402778	
##	1951	-24.748737		-36.188131		-2.241162		-8.036616		-4.506313		35.402778	
##	1952	-24.748737		-36.188131		-2.241162		-8.036616		-4.506313		35.402778	
##	1953	-24.748737		-36.188131		-2.241162		-8.036616		-4.506313		35.402778	
##	1954	-24.748737		-36.188131		-2.241162		-8.036616		-4.506313		35.402778	
##	1955	-24.748737		-36.188131		-2.241162		-8.036616		-4.506313		35.402778	
##	1956	-24.748737		-36.188131		-2.241162		-8.036616		-4.506313		35.402778	
##	1957	-24.748737		-36.188131		-2.241162		-8.036616		-4.506313		35.402778	
##	1958	-24.748737		-36.188131		-2.241162		-8.036616		-4.506313		35.402778	

##	1959	-24.748737	-36.188131	-2.241162	-8.036616	-4.506313	35.402778	
##	1960	-24.748737	-36.188131	-2.241162	-8.036616	-4.506313	35.402778	
##		Jul	Aug	Sep	Oct	Nov	Dec	
##	1949	63.830808	62.823232	16.520202	-20.642677	-53.593434	-28.619949	
##	1950	63.830808	62.823232	16.520202	-20.642677	-53.593434	-28.619949	
##	1951	63.830808	62.823232	16.520202	-20.642677	-53.593434	-28.619949	
##	1952	63.830808	62.823232	16.520202	-20.642677	-53.593434	-28.619949	
##	1953	63.830808	62.823232	16.520202	-20.642677	-53.593434	-28.619949	
##	1954	63.830808	62.823232	16.520202	-20.642677	-53.593434	-28.619949	
##	1955	63.830808	62.823232	16.520202	-20.642677	-53.593434	-28.619949	
##	1956	63.830808	62.823232	16.520202	-20.642677	-53.593434	-28.619949	
##	1957	63.830808	62.823232	16.520202	-20.642677	-53.593434	-28.619949	
##	1958	63.830808	62.823232	16.520202	-20.642677	-53.593434	-28.619949	
##	1959	63.830808	62.823232	16.520202	-20.642677	-53.593434	-28.619949	
##	1960	63.830808	62.823232	16.520202	-20.642677	-53.593434	-28.619949	
##								
##	\$trend							
##		Jan	Feb	Mar	Apr	May	Jun	Jul
##	1949	NA	NA	NA	NA	NA	NA	126.7917
##	1950	131.2500	133.0833	134.9167	136.4167	137.4167	138.7500	140.9167
##	1951	157.1250	159.5417	161.8333	164.1250	166.6667	169.0833	171.2500
##	1952	183.1250	186.2083	189.0417	191.2917	193.5833	195.8333	198.0417
##	1953	215.8333	218.5000	220.9167	222.9167	224.0833	224.7083	225.3333
##	1954	228.0000	230.4583	232.2500	233.9167	235.6250	237.7500	240.5000
##	1955	261.8333	266.6667	271.1250	275.2083	278.5000	281.9583	285.7500
##	1956	309.9583	314.4167	318.6250	321.7500	324.5000	327.0833	329.5417
##	1957	348.2500	353.0000	357.6250	361.3750	364.5000	367.1667	369.4583
##	1958	375.2500	377.9167	379.5000	380.0000	380.7083	380.9583	381.8333
##	1959	402.5417	407.1667	411.8750	416.3333	420.5000	425.5000	430.7083

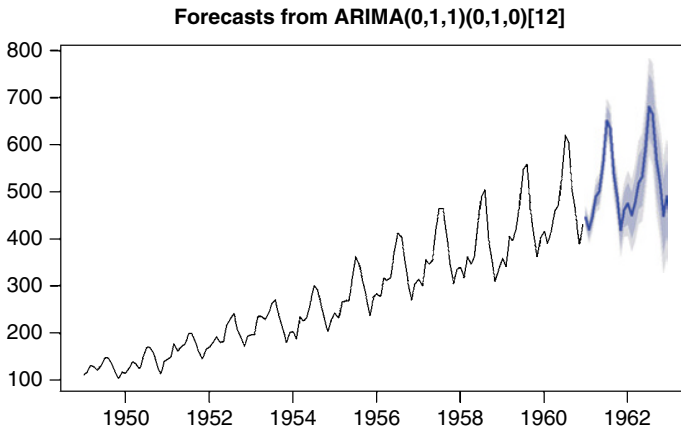
```
## 1960 456.3333 461.3750 465.2083 469.3333 472.7500 475.0417 NA
##           Aug       Sep       Oct       Nov       Dec
## 1949 127.2500 127.9583 128.5833 129.0000 129.7500
## 1950 143.1667 145.7083 148.4167 151.5417 154.7083
## 1951 173.5833 175.4583 176.8333 178.0417 180.1667
## 1952 199.7500 202.2083 206.2500 210.4167 213.3750
## 1953 225.3333 224.9583 224.5833 224.4583 225.5417
## 1954 243.9583 247.1667 250.2500 253.5000 257.1250
## 1955 289.3333 293.2500 297.1667 301.0000 305.4583
## 1956 331.8333 334.4583 337.5417 340.5417 344.0833
## 1957 371.2083 372.1667 372.4167 372.7500 373.6250
## 1958 383.6667 386.5000 390.3333 394.7083 398.6250
## 1959 435.1250 437.7083 440.9583 445.8333 450.6250
## 1960          NA          NA          NA          NA          NA
##
## $random
##           Jan       Feb       Mar       Apr       May
## 1949          NA          NA          NA          NA          NA
## 1950  8.4987374 29.1047980  8.3244949  6.6199495 -7.9103535
## 1951 12.6237374 26.6464646 18.4078283  6.9116162  9.8396465
## 1952 12.6237374 29.9797980  6.1994949 -2.2550505 -6.0770202
## 1953  4.9154040 13.6881313 17.3244949 20.1199495  9.4229798
## 1954  0.7487374 -6.2702020  4.9911616  1.1199495  2.8813131
## 1955  4.9154040  2.5214646 -1.8838384  1.8282828 -3.9936869
## 1956 -1.2095960 -1.2285354  0.6161616 -0.7133838 -1.9936869
## 1957 -8.5012626 -15.8118687  0.6161616 -5.3383838 -4.9936869
## 1958 -10.5012626 -23.7285354 -15.2588384 -23.9633838 -13.2020202
## 1959 -17.7929293 -28.9785354 -3.6338384 -12.2967172  4.0063131
## 1960 -14.5845960 -34.1868687 -43.9671717 -0.2967172  3.7563131
```

##	Jun	Jul	Aug	Sep	Oct
## 1949	NA	-42.6224747	-42.0732323	-8.4785354	11.0593434
## 1950	-25.1527778	-34.7474747	-35.9898990	-4.2285354	5.2260101
## 1951	-26.4861111	-36.0808081	-37.4065657	-7.9785354	5.8093434
## 1952	-13.2361111	-31.8724747	-20.5732323	-9.7285354	5.3926768
## 1953	-17.1111111	-25.1641414	-16.1565657	-4.4785354	7.0593434
## 1954	-9.1527778	-2.3308081	-13.7815657	-4.6868687	-0.6073232
## 1955	-2.3611111	14.4191919	-5.1565657	2.2297980	-2.5239899
## 1956	11.5138889	19.6275253	10.3434343	4.0214646	-10.8989899
## 1957	19.4305556	31.7108586	32.9684343	15.3131313	-4.7739899
## 1958	18.6388889	45.3358586	58.5101010	0.9797980	-10.6906566
## 1959	11.0972222	53.4608586	61.0517677	8.7714646	-13.3156566
## 1960	24.5555556	NA	NA	NA	NA
##	Nov	Dec			
## 1949	28.5934343	16.8699495			
## 1950	16.0517677	13.9116162			
## 1951	21.5517677	14.4532828			
## 1952	15.1767677	9.2449495			
## 1953	9.1351010	4.0782828			
## 1954	3.0934343	0.4949495			
## 1955	-10.4065657	1.1616162			
## 1956	-15.9482323	-9.4633838			
## 1957	-14.1565657	-9.0050505			
## 1958	-31.1148990	-33.0050505			
## 1959	-30.2398990	-17.0050505			
## 1960	NA	NA			
##					

```
## $figure
## [1] -24.748737 -36.188131 -2.241162 -8.036616 -4.506313 35.402778
## [7] 63.830808 62.823232 16.520202 -20.642677 -53.593434 -28.619949
##
## $type
## [1] "additive"
##
## attr(,"class")
## [1] "decomposed.ts"
plot(decompose(AirPassengers))
```



```
b=auto.arima(AirPassengers)
forecast(b,24)
##          Point Forecast      Lo 80      Hi 80      Lo 95      Hi 95
## Jan 1961      446.7582 431.6858 461.8306 423.7070 469.8094
## Feb 1961      420.7582 402.5180 438.9984 392.8622 448.6542
## Mar 1961      448.7582 427.8241 469.6923 416.7423 480.7741
## Apr 1961      490.7582 467.4394 514.0770 455.0952 526.4212
## May 1961      501.7582 476.2770 527.2395 462.7880 540.7284
## Jun 1961      564.7582 537.2842 592.2323 522.7403 606.7761
## Jul 1961      651.7582 622.4264 681.0900 606.8991 696.6173
## Aug 1961      635.7582 604.6796 666.8368 588.2275 683.2889
## Sep 1961      537.7582 505.0258 570.4906 487.6983 587.8181
## Oct 1961      490.7582 456.4516 525.0648 438.2908 543.2256
## Nov 1961      419.7582 383.9466 455.5698 364.9891 474.5273
## Dec 1961      461.7582 424.5023 499.0141 404.7803 518.7361
## Jan 1962      476.5164 431.4567 521.5761 407.6036 545.4292
## Feb 1962      450.5164 400.9938 500.0390 374.7781 526.2547
## Mar 1962      478.5164 424.9010 532.1318 396.5188 560.5141
## Apr 1962      520.5164 463.0993 577.9335 432.7045 608.3283
## May 1962      531.5164 470.5341 592.4987 438.2520 624.7808
## Jun 1962      594.5164 530.1661 658.8667 496.1011 692.9317
## Jul 1962      681.5164 613.9659 749.0670 578.2068 784.8261
## Aug 1962      665.5164 594.9105 736.1223 557.5340 773.4988
## Sep 1962      567.5164 493.9820 641.0508 455.0552 679.9776
## Oct 1962      520.5164 444.1657 596.8671 403.7481 637.2847
## Nov 1962      449.5164 370.4497 528.5831 328.5943 570.4385
## Dec 1962      491.5164 409.8239 573.2089 366.5785 616.4543
plot(forecast(b,24))
```



7.3 Association Analysis

Association analysis is widely used in e-commerce websites (which products sell well together), as well as areas like retail (keeping products that sell well together placed together), healthcare, telecom (which are value-added services to bundle), and many others.

Example database with four items and five transactions

Transaction ID	Milk	Bread	Butter	Beer
1	1	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	1	1	0
5	0	1	0	0

- The item set (milk,bread- $\rightarrow$ butter) has a support of 20% since it occurs in 20% of all transactions (1 out of 5 transactions). Support is an indication of how frequently the item set appears.
- The item set (milk,bread- $\rightarrow$ butter) has a confidence of 50% since it occurs in 50% of all such transactions (1 out of 2 transactions). Confidence is an indication of how often the rule has been found to be true.
- Lift would be $= 0.2/0.4*0.4 = 1.25$. Lift considers both the confidence of the rule and the overall data.

The basic theoretical framework came from this paper: Fast Algorithms for

Mining Association Rules at <http://rakesh.agrawal-family.com/papers/vldb94apriori.pdf>

You can see some datasets for association analysis at **Frequent Itemset Mining Dataset Repository**: <http://fimi.ua.ac.be/data/>.

Again association analysis is very easy in R due to a rules package and difficult to find in Python package landscape. (See <https://github.com/scikit-learn/scikit-learn/issues/2872> and <https://github.com/scikit-learn/scikit-learn/issues/2662> for the reasons scikit-learn won't be able to accept it.) One possible solution is PyFIM (Frequent Item Sets) available at <http://www.borgelt.net/pyfim.html>.

In R here is some code to show how easy it is: <http://rpubs.com/newajay/associationanalysis>.

```
library(rattle)
## Rattle: A free graphical interface for data mining
## with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty
## Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
#rattle()
#install.packages("arulesViz")
library(arulesViz)
## Loading required package: arules
## Loading required package: Matrix
##
## Attaching package: 'arules'
## The following objects are masked from
## 'package:base':
##
##     abbreviate, write
## Loading required package: grid
## Warning: failed to assign NativeSymbolInfo for lhs
## since lhs is already
## defined in the 'lazyeval' namespace
## Warning: failed to assign NativeSymbolInfo for rhs
## since rhs is already
## defined in the 'lazyeval' namespace
data(Groceries)
str(Groceries)
```

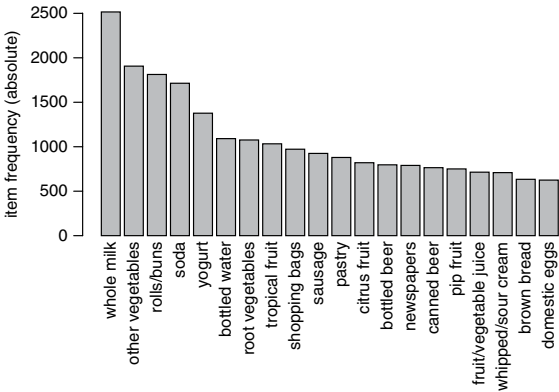
```

## Formal class 'transactions' [package "arules"] with 3 slots
##   ..@ data          :Formal class 'ngCMatrix' [package "Matrix"] with 5 slots
##   .. .. ..@ i       : int [1:43367] 13 60 69 78 14 29 98 24 15 29 ...
##   .. .. ..@ p       : int [1:9836] 0 4 7 8 12 16 21 22 27 28 ...
##   .. .. ..@ Dim      : int [1:2] 169 9835
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : NULL
##   .. .. ..@ factors : list()
##   ..@ itemInfo      :'data.frame': 169 obs. of 3 variables:
##   .. ..$ labels: chr [1:169] "frankfurter" "sausage" "liver loaf" "ham" ...
##   .. ..$ level2: Factor w/ 55 levels "baby food","bags",...: 44 44 44 44 44 44 44
42 42 41 ...
##   .. ..$ level1: Factor w/ 10 levels "canned food",...: 6 6 6 6 6 6 6 6 6 6 ...
##   ..@ itemsetInfo:'data.frame': 0 obs. of 0 variables
summary(Groceries)
## transactions as itemMatrix in sparse format with
## 9835 rows (elements/itemsets/transactions) and
## 169 columns (items) and a density of 0.02609146
##
## most frequent items:
##      whole milk other vegetables      rolls/buns      soda
##      2513      1903      1809      1715
##      yogurt      (Other)
##      1372      34055
##
## element (itemset/transaction) length distribution:
## sizes

```

##	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
##	2159	1643	1299	1005	855	645	545	438	350	246	182	117	78	77	55
##	16	17	18	19	20	21	22	23	24	26	27	28	29	32	
##	46	29	14	14	9	11	4	6	1	1	1	1	3	1	
##															
##	Min.	1st Qu.	Median		Mean		3rd Qu.	Max.							
##	1.000	2.000	3.000		4.409		6.000	32.000							

```
## includes extended item information - examples:
##      labels  level2      level1
## 1 frankfurter sausage meat and sausage
## 2      sausage sausage meat and sausage
## 3  liver loaf sausage meat and sausage
itemFrequencyPlot(Groceries,topN=20,type="absolute")
```



```

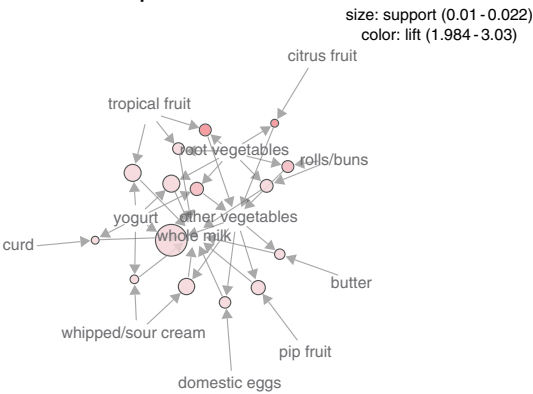
rules <- apriori(Groceries, parameter=list(support=0.01, confidence=0.5))
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##          0.5    0.1    1 none FALSE                TRUE         5    0.01    1
## maxlen target  ext
##      10  rules FALSE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##    0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
summary(rules)
## set of 15 rules
##
## rule length distribution (lhs + rhs):sizes
##    3

```

```
## 15
##
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      3      3      3      3      3      3
##
## summary of quality measures:
##      support      confidence      lift
## Min.      :0.01007    Min.      :0.5000    Min.      :1.984
## 1st Qu.:0.01174    1st Qu.:0.5151    1st Qu.:2.036
## Median :0.01230    Median :0.5245    Median :2.203
## Mean      :0.01316    Mean      :0.5411    Mean      :2.299
## 3rd Qu.:0.01403    3rd Qu.:0.5718    3rd Qu.:2.432
## Max.      :0.02227    Max.      :0.5862    Max.      :3.030
##
## mining info:
##      data ntransactions support confidence
## Groceries      9835      0.01      0.5
inspect(head(sort(rules, by = "lift"),10))
##      lhs      rhs      support confidence      lift
## [1] {citrus fruit,
##      root vegetables} => {other vegetables} 0.01037112 0.5862069 3.029608
## [2] {tropical fruit,
##      root vegetables} => {other vegetables} 0.01230300 0.5845411 3.020999
## [3] {root vegetables,
##      rolls/buns}      => {other vegetables} 0.01220132 0.5020921 2.594890
## [4] {root vegetables,
##      yogurt}          => {other vegetables} 0.01291307 0.5000000 2.584078
```

```
## [5] {curd,  
##      yogurt}          => {whole milk}      0.01006609  0.5823529  2.279125  
## [6] {other vegetables,  
##      butter}          => {whole milk}      0.01148958  0.5736041  2.244885  
## [7] {tropical fruit,  
##      root vegetables} => {whole milk}      0.01199797  0.5700483  2.230969  
## [8] {root vegetables,  
##      yogurt}          => {whole milk}      0.01453991  0.5629921  2.203354  
## [9] {other vegetables,  
##      domestic eggs}   => {whole milk}      0.01230300  0.5525114  2.162336  
## [10] {yogurt,  
##       whipped/sour cream} => {whole milk}  0.01087951  0.5245098  2.052747  
plot(rules, method="graph")
```

Graph for 15 rules



```
library(arulesViz)
data(Groceries)
summary(Groceries)
## transactions as itemMatrix in sparse format with
## 9835 rows (elements/itemsets/transactions) and
## 169 columns (items) and a density of 0.02609146
##
## most frequent items:
##      whole milk other vegetables      rolls/buns      soda
##      2513      1903      1809      1715
##      yogurt      (Other)
##      1372      34055
##
## element (itemset/transaction) length distribution:
## sizes
##      1      2      3      4      5      6      7      8      9     10     11     12     13     14     15
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117  78  77  55
##      16     17     18     19     20     21     22     23     24     26     27     28     29     32
##      46     29     14     14      9     11      4      6      1      1      1      1      3      1
##
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##      1.000   2.000   3.000   4.409   6.000  32.000
##
## includes extended item information - examples:
##      labels level2      level1
## 1 frankfurter sausage meat and sausage
## 2      sausage sausage meat and sausage
## 3  liver loaf sausage meat and sausage
```

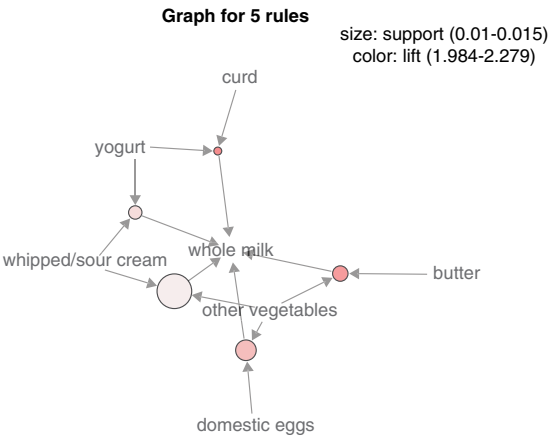
```

rules <- apriori(Groceries, parameter=list(support=0.01, confidence=0.5))
## Apriori
##
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
##           0.5    0.1    1 none FALSE                TRUE         5     0.01      1
## maxlen target   ext
##          10 rules FALSE
##
## Algorithmic control:
## filter tree heap memopt load sort verbose
##      0.1 TRUE TRUE  FALSE TRUE     2     TRUE
##
## Absolute minimum support count: 98
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [88 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
summary(rules)
## set of 15 rules
##
## rule length distribution (lhs + rhs):sizes
##    3
## 15

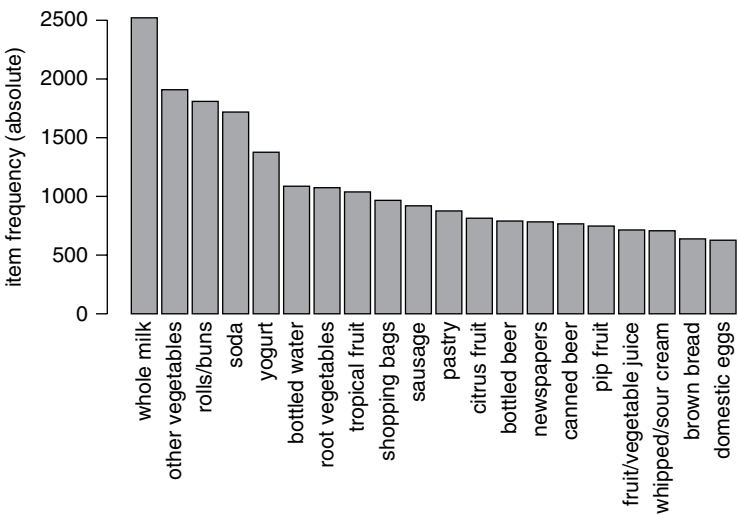
```

```
##
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##      3      3      3      3      3      3
##
## summary of quality measures:
##      support      confidence      lift
## Min.      :0.01007   Min.      :0.5000   Min.      :1.984
## 1st Qu.:0.01174   1st Qu.:0.5151   1st Qu.:2.036
## Median :0.01230   Median :0.5245   Median :2.203
## Mean    :0.01316   Mean    :0.5411   Mean    :2.299
## 3rd Qu.:0.01403   3rd Qu.:0.5718   3rd Qu.:2.432
## Max.    :0.02227   Max.    :0.5862   Max.    :3.030
##
## mining info:
##      data ntransactions support confidence
## Groceries      9835      0.01      0.5
inspect(head(sort(rules, by = "lift"),10))
##      lhs      rhs      support confidence      lift
## [1] {citrus fruit,
##      root vegetables} => {other vegetables} 0.01037112 0.5862069 3.029608
## [2] {tropical fruit,
##      root vegetables} => {other vegetables} 0.01230300 0.5845411 3.020999
## [3] {root vegetables,
##      rolls/buns}      => {other vegetables} 0.01220132 0.5020921 2.594890
## [4] {root vegetables,
##      yogurt}          => {other vegetables} 0.01291307 0.5000000 2.584078
## [5] {curd,
##      yogurt}          => {whole milk}      0.01006609 0.5823529 2.279125
```

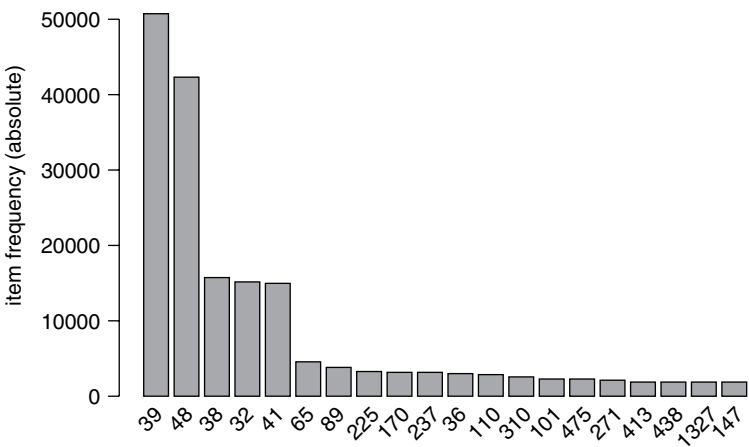
## [6]	{other vegetables,					
##	butter}	=>	{whole milk}	0.01148958	0.5736041	2.244885
## [7]	{tropical fruit,					
##	root vegetables}	=>	{whole milk}	0.01199797	0.5700483	2.230969
## [8]	{root vegetables,					
##	yogurt}	=>	{whole milk}	0.01453991	0.5629921	2.203354
## [9]	{other vegetables,					
##	domestic eggs}	=>	{whole milk}	0.01230300	0.5525114	2.162336
## [10]	{yogurt,					
##	whipped/sour cream}	=>	{whole milk}	0.01087951	0.5245098	2.052747
plot(rules[1:5],method="graph",interactive = F)						



```
plot(rules[1:15],method="graph",interactive = T)
itemFrequencyPlot(Groceries,topN=20,type="absolute")
```



```
#http://fimi.ua.ac.be/data/retail.pdf
library(arules)
a=read.transactions("http://fimi.ua.ac.be/data/retail.
  dat")
itemFrequencyPlot(a,topN=20,type="absolute")
```



basket

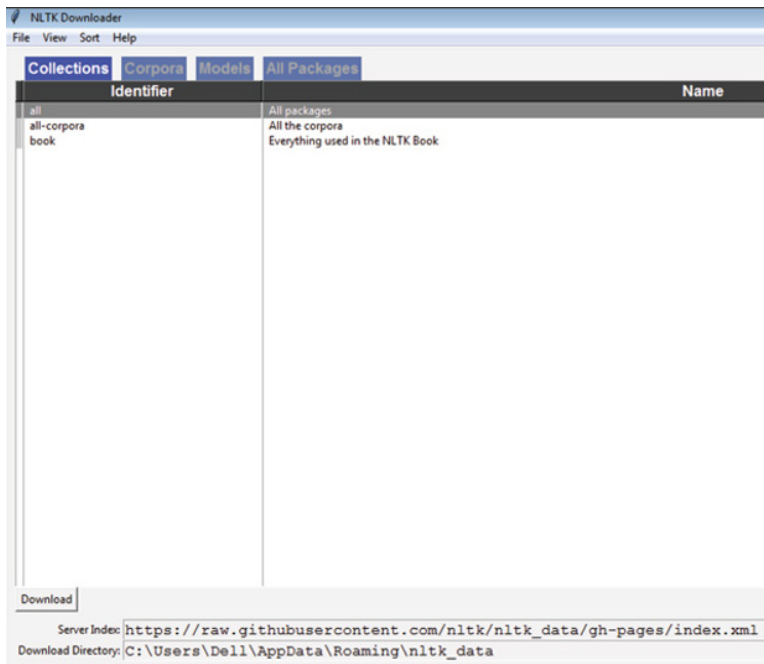
Text mining is much more elaborate and powerful, and the `nltk` package in Python does match up to the `tm` package (and its sub-packages) in R. From <https://github.com/decisionstats/pythonfordatascience/blob/master/nltk.ipynb> we see

```
import nltk
```

In [*] :

```
nltk.download()
```

showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml



In [4] :

```
import nltk
```

In [5] :

```
nltk.download()
```

showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml

Out [5] :

```
True
```

In [6]:

```

from nltk.book import *

*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G. K. Chesterton
      1908

```

In [7]:

```

text1.similar("great")

good whale long vast sea whole living small other
  large dead mighty
same such last more much sperm noble old

```

In [8]:

```

from urllib import request
url = "http://www.gutenberg.org/files/2554/2554.txt"
response = request.urlopen(url)
raw = response.read().decode('utf8')
type(raw)

```

Out [8]:

```
str
```

In [9]:

```
len(raw)
```

Out [9]:

```
1176896
```

In [10]:

```
raw[:75]
```

Out [10]:

```

'The Project Gutenberg EBook of Crime and Punishment,
  by Fyodor Dostoevsky\r\n'

```

```

tokens = nltk.word_tokenize(raw)
type(tokens)

list

len(tokens)

254352

tokens[:10]

['The',
 'Project',
 'Gutenberg',
 'EBook',
 'of',
 'Crime',
 'and',
 'Punishment',
 ',',
 'By']

```

Citation-

<http://www.cs.duke.edu/courses/spring14/compsci290/assignments/lab02.html>

From <http://rpubs.com/newajay/textmining> we see basic text mining in R

```

memory.size()
## [1] 17.11
memory.limit()
## [1] 1535
#install.packages("tm")
library(tm)
## Loading required package: NLP
getReaders()
## [1] "readDOC" "readPDF"
## [3] "readPlain" "readRCV1"
## [5] "readRCV1asPlain" "readReut21578XML"
## [7] "readReut21578XMLasPlain" "readTabular"

```

```
## [9] "readTagged" "readXML"
getSources()
## [1] "DataframeSource" "DirSource" "URISource"
## "VectorSource"
## [5] "XMLSource" "ZipSource"
warp = "http://www.gutenberg.org/files/2600/2600-0.txt"

Corpus1=Corpus(URISource(warp), readerControl =
  list(language = "eng"))

inspect(Corpus1)
## <<VCorpus>>
## Metadata: corpus specific: 0, document level
## (indexed): 0
## Content: documents: 1
##
## [[1]]
## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 3170017
summary(Corpus1)
## Length Class Mode
## 2600-0.txt 2 PlainTextDocument list
```

7.4 Cleaning Corpus and Making Bag of Words

```
Corpus1 <- tm_map(Corpus1, removePunctuation)
Corpus1 <- tm_map(Corpus1, removeNumbers)
Corpus1 <- tm_map(Corpus1, tolower)
Corpus1 <- tm_map(Corpus1, removeWords,
  stopwords("english"))
#install.packages("SnowballC")
library(SnowballC)
Corpus1 <- tm_map(Corpus1, stemDocument)
Corpus1 <- tm_map(Corpus1, stripWhitespace)
Corpus1 <- tm_map(Corpus1, PlainTextDocument)

dtm <- DocumentTermMatrix(Corpus1)

tdm <- TermDocumentMatrix(Corpus1)
tdm
## <<TermDocumentMatrix (terms: 21354, documents: 1)>>
## Non-/sparse entries: 21354/0
## Sparsity : 0%
## Maximal term length: 29
```

```
## Weighting          : term frequency (tf)
inspect(tdm[1:30,1])
## <<TermDocumentMatrix (terms: 30, documents: 1)>>
## Non-/sparse entries: 30/0
## Sparsity           : 0%
## Maximal term length: 13
## Weighting          : term frequency (tf)
##
##                      Docs
## Terms               character(0)
##  "â€"                9
##  "â€" "â€"            1
##  "â€" annett         1
##  "â€" precede        1
##  "â€" salut          1
##  "â€" st              1
##  "â€" though         1
##  "aâ€"              1
##  aah                 1
##  aback               3
##  abacus              1
##  abandon             31
##  abandoned           51
##  abandoning          24
##  abandonment         13
##  abandons            1
##  abas                1
##  abash               1
##  abashed             11
##  abate               1
##  abbã                18
##  abbãs               1
##  abbreviations       1
##  abc                 1
##  abdicate            1
##  abdomen             2
##  abdomens            2
##  abduction           3
##  abductor            1
##  abhorrence          1
matx1=as.matrix(tdm)
matx1[1:10]
##  [1] 9 1 1 1 1 1 1 1 1 3
sort1=sort(rowSums(matx1),decreasing=T)
```

```
sort1[1:10]
##      said      one  prince  pierre      now
  natãsha      man  andrew
##      2834      1882      1725      1561      1304
  1103      1077      1045
##      will princess
##      997      916
di=data.frame(Word=names(sort1),Frequency=sort1)
di[1:10,]
##              Word Frequency
## said          said      2834
## one           one      1882
## prince        prince     1725
## pierre        pierre     1561
## now           now      1304
## natãsha       natãsha    1103
## man           man      1077
## andrew        andrew     1045
## will          will       997
## princess      princess    916
#install.packages("wordcloud")
library(wordcloud)
## Loading required package: RColorBrewer
wordcloud(di$Word, di$Frequency, max.
  words=100,colors=brewer.pal(6, "Reds"))
```




```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm

import pandas as pd
import numpy as np
```

In [2]:

```
wine=pd.read_csv("http://archive.ics.uci.edu/
    ml/machine-learning-databases/wine/wine.
    data",header=None)
```

In [3]:

```
wine.head()
```

Out [3]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

From <http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.names> we get the column names

In [4]:

```
wine.columns=['winetype', 'Alcohol', 'Malic
    acid', 'Ash', 'Alcalinity of ash', 'Magnesium', 'Total
    phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proan
    thocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of
    diluted wines', 'Proline']
```

In [5]:

```
wine.head()
```

	winetype	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	1.938202	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112
std	0.775035	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000
25%	1.000000	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500
50%	2.000000	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000
75%	3.000000	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000

Out [5] :

	wine type	Alcohol	Malic acid	Ash	Alcalinity of ash	Magne- sium	Total phenols	Flava- noids	Nonfla- vanoid phenols	Proantho- cyanins	Color intensity	Hue	OD280/ OD315 of diluted wines	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

In [6] :

```
wine.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
winetype                    178 non-null int64
Alcohol                     178 non-null float64
Malic acid                  178 non-null float64
Ash                         178 non-null float64
Alcalinity of ash           178 non-null float64
Magnesium                   178 non-null int64
Total phenols                178 non-null float64
Flavanoids                  178 non-null float64
Nonflavanoid phenols        178 non-null float64
Proanthocyanins             178 non-null float64
Color intensity             178 non-null float64
Hue                         178 non-null float64
OD280/OD315 of diluted wines 178 non-null float64
Proline                     178 non-null int64
dtypes: float64(11), int64(3)
memory usage: 19.5 KB
```

In [7] :

```
wine.describe()
```

Out [7] :

	Flavanoids	Nonflavanoid phenols	Proantho- cyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
2.029270	0.361854	1.590899	5.058090	0.957449	2.611685	746.893258	
0.998859	0.124453	0.572359	2.318286	0.228572	0.709990	314.907474	
0.340000	0.130000	0.410000	1.280000	0.480000	1.270000	278.000000	
1.205000	0.270000	1.250000	3.220000	0.782500	1.937500	500.500000	
2.135000	0.340000	1.555000	4.690000	0.965000	2.780000	673.500000	
2.875000	0.437500	1.950000	6.200000	1.120000	3.170000	985.000000	
5.080000	0.660000	3.580000	13.000000	1.710000	4.000000	1680.000000	

```
pd.value_counts(wine['winetype'])
```

In [8]:

```
2      71
1      59
3      48
Name: winetype, dtype: int64
```

Out [8]:

The R solution is https://rstudio-pubs-static.s3.amazonaws.com/33876_1d7794d9a86647ca90c4f182df93f0e8.html

The clustering optimization problem is solved with the function *kmeans* in R.

```
wine.stand <- scale(wine[-1]) # To standarize the
  variables

# K-Means
k.means.fit <- kmeans(wine.stand, 3) # k = 3
In k.means.fit are contained all the elements of the
  cluster output:
attributes(k.means.fit)
## $names
## [1] "cluster"      "centers"      "totss"
  "withinss"
## [5] "tot.withinss" "betweenss"    "size"
  "iter"
## [9] "ifault"
##
## $class
## [1] "kmeans"
# Centroids:
k.means.fit$centers
##   Alcohol   Malic      Ash Alkalinity Magnesium
  Phenols Flavanoids
```

```
## 1  0.1644  0.8691  0.1864      0.5229  -0.07526 -0.97658  -1.21183
## 2  0.8329 -0.3030  0.3637     -0.6085   0.57596  0.88275   0.97507
## 3 -0.9235 -0.3929 -0.4931      0.1701  -0.49033 -0.07577   0.02075
##   Nonflavanoids Proanthocyanins   Color      Hue Dilution Proline
## 1           0.72402             -0.7775  0.9389 -1.1615  -1.2888 -0.4059
## 2          -0.56051             0.5787  0.1706  0.4727   0.7771  1.1220
## 3          -0.03344             0.0581 -0.8994  0.4605   0.2700 -0.7517
```

For Python it is a bit similar to do kmeans clusterinnng

```
In [9]:
x=wine.ix[:,1:14]
y=wine.ix[:,1]

In [10]:
x.columns

Out[10]:
Index(['Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium',
      'Total phenols', 'Flavanoids', 'Nonflavanoid phenols',
      'Proanthocyanins', 'Color intensity', 'Hue',
      'OD280/OD315 of diluted wines', 'Proline'],
      dtype='object')

In [33]:
x.ix[:,1].head()
```

Out [33] :

	Alcohol
0	14.23
1	13.20
2	13.16
3	14.37
4	13.24

In [12] :

y.columns

Out [12] :

Index(['winetype'], dtype='object')

In [13] :

x.head()

Out [13] :

	Alcohol	Malic acid	Ash	Alcali- nity of ash	Magne- sium	Total phenols	Flava- noids	Non- flava- noid phenols	Proan- thocy- anins	Color inten- sity	Hue	OD280/ OD315 of diluted wines	Proline
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

In [14] :

y.head()

Out [14] :

	winetype
0	1
1	1
2	1
3	1
4	1

In [15] :

y.info

Out [15] :

<bound method DataFrame.info of winetype

0	1	148	3
1	1	149	3
2	1	150	3
3	1	151	3
4	1	152	3
5	1	153	3
6	1	154	3
7	1	155	3
8	1	156	3
9	1	157	3
10	1	158	3
11	1	159	3
12	1	160	3
13	1	161	3
14	1	162	3
15	1	163	3
16	1	164	3
17	1	165	3
18	1	166	3
19	1	167	3
20	1	168	3
21	1	169	3
22	1	170	3
23	1	171	3
24	1	172	3
25	1	173	3
26	1	174	3
27	1	175	3
28	1	176	3
29	1	177	3

.. ..

[178 rows x 1 columns]>

In [16] :

```
# K Means Cluster
model = KMeans(n_clusters=3)
model.fit(x)
```

```
Out [16]:
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=3, n_init=10,
        n_jobs=1, precompute_distances='auto', random_state=None, tol=0.0001,
        verbose=0)

In [17]:
model.labels_

Out [17]:
array([0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0,
        0, 2, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 2, 2, 0, 0, 2, 2, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 2, 1, 1, 2, 1, 1, 2,
        2, 2, 1, 1, 0, 2, 1, 1, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1,
        1, 1, 1, 2, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1,
        1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1,
        1, 1, 2, 2, 1, 1, 2, 2, 1, 2, 2, 1, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 1,
        2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 1], dtype=int32)

In [18]:
pd.value_counts(model.labels_)

Out [18]:
1      69
2      62
0      47
dtype: int64

In [19]:
pd.value_counts(y['winetype'])

Out [19]:
2      71
1      59
3      48
Name: winetype, dtype: int64
```

In [20]:

```
# We convert all the 1s to 0s and 0s to 1s.
predY = np.choose(model.labels_, [1, 2, 3]).astype(np.
    int64)
```

In [21]:

```
print (y['winetype'])
print (model.labels_)
print (predY)
```

0	1	148	3
1	1	149	3
2	1	150	3
3	1	151	3
4	1	152	3
5	1	153	3
6	1	154	3
7	1	155	3
8	1	156	3
9	1	157	3
10	1	158	3
11	1	159	3
12	1	160	3
13	1	161	3
14	1	162	3
15	1	163	3
16	1	164	3
17	1	165	3
18	1	166	3
19	1	167	3
20	1	168	3
21	1	169	3
22	1	170	3
23	1	171	3
24	1	172	3
25	1	173	3
26	1	174	3
27	1	175	3
28	1	176	3
29	1	177	3
..			

```
Name: winetype, dtype: int64
[0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 0 0 2 2 0 0 2 0 0 0 0 0 0 2 2
 0 0 2 2 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 1 2 1 1 2 1 1 2 2 2 1 1 0
 2 1 1 1 2 1 1 2 2 1 1 1 1 1 2 2 1 1 1 1 1 2 2 1 2 1 2 1 1 1 2 1 1 1 2 1
 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 2 1 1 1 2 2 1 1 2 2 1 2
 2 1 1 1 1 2 2 2 1 2 2 2 1 2 1 2 2 1 2 2 2 2 1 1 2 2 2 2 2 1]
[1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 1 3 3 1 1 3 1 1 1 1 1 3 3
 1 1 3 3 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 2 3 2 2 3 2 2 3 3 3 2 2 1
 3 2 2 2 3 2 2 3 3 2 2 2 2 2 3 3 2 2 2 2 3 3 2 3 2 3 2 2 2 3 2 2 2 2 3 2
 2 3 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 3 2 2 3 3 3 2 2 2 3 3 2 2 3 3 2 3
 3 2 2 2 2 3 3 2 3 3 2 3 2 3 2 3 3 3 2 2 3 3 3 3 3 2]
```

In [22]:

```
# Performance Metrics
sm.accuracy_score(y, predY)

0.702247191011236
```

Out [22]:

```
# Confusion Matrix
sm.confusion_matrix(y, predY)
```

In [23]:

```
array([[46,  0, 13],
       [ 1, 50, 20],
       [ 0, 19, 29]])
```

Out [23]:

```
pd.unique(y.winetype)
```

In [24]:

```
array([1, 2, 3])
```

Out [24]:

```
#!sudo pip install ggplot
```

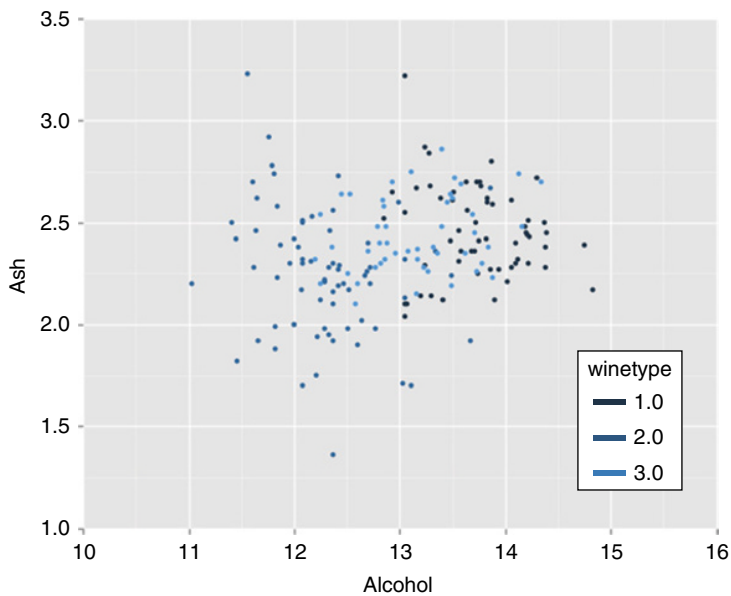
In [25]:

```
from ggplot import *
%matplotlib inline
```

In [30]:

```
p = ggplot(aes(x='Alcohol', y='Ash',color="winetype"),
            data=wine)
p + geom_point()
```

In [31]:

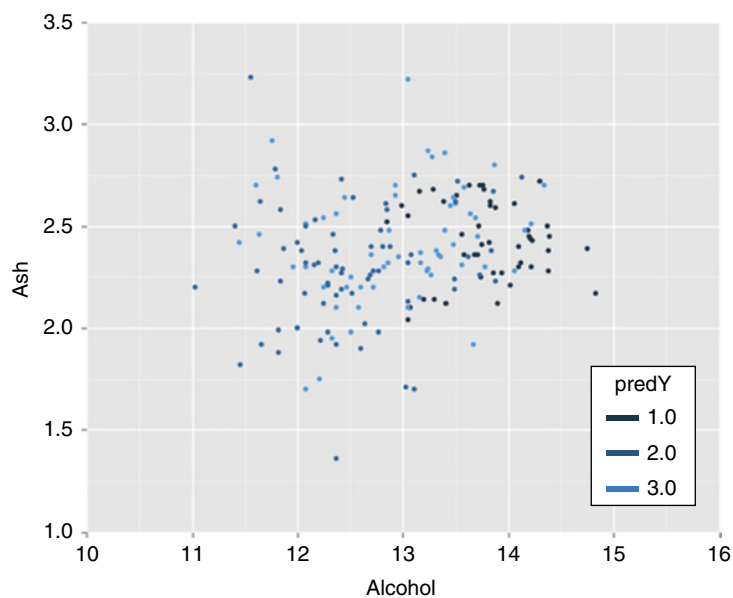


```
<ggplot: (-9223363292162990364)>
```

Out [31]:

```
p2 = ggplot(aes(x='Alcohol', y='Ash',color="predY"),
             data=wine)
p2 + geom_point()
```

In [32]:



Out [32] :

<ggplot: (8744691751337)>

8

Conclusion and Summary

Some conclusions to draw from comparing R and Python functionality as well as the author's own experience with the SAS language are as follows:

- 1) There is no one software or language that is good for each and every use case or situation.
- 2) For the student, researcher, job seeker, and professional, having skills in two languages is better than having skills in one language.
- 3) R ecosystem has learnt from Python (like Beautiful Soup or bokeh or R Essentials for Jupyter), while Python ecosystem has learnt from R (like ggplot and pandas). This cross-language learning should be encouraged especially in academia and industry.
- 4) As feature requests, Python statsmodels can be more user-friendly (like car), time series can have more tools like auto.arima (in forecast), and scikit-learn can have spin-off smaller packages (like arules for association analysis) and easier to read syntax (like party, rpart, and RandomForest) instead of having a very big scikit-learn package. Some of these machine learning packages should be made panda ready rather than numpy specific to help make them more popular.
- 5) Python can make or port GUIs like R Commander and rattle as that will help in teaching.

Index

a

abs 167
 absolute 328, 329, 333, 336
 academia 355
 academic 26
 academy 11, 28
 acast 158
 account 165, 256, 257
 accuracy 4, 21, 166, 219, 304, 305,
 307–309, 352
 acid 85, 344, 345, 347, 348
 acknowledgments 7, 15
 acos 50
 acosh 50
 acquisition 70
 across 11, 143, 309
 ActivePython 42
 adjusted 165, 182, 183, 185, 187, 189,
 198, 200, 202–204, 208
 advstats 211
 AIC 179, 216, 226, 272
 AirPassengers 318, 319, 323, 324
 ajaydecis 48, 57, 102, 115, 122, 129,
 130, 148, 218, 248, 253, 255
 ajayohri 32, 35, 59–62, 114, 168, 235,
 273, 275, 281
 alcohol 344–348, 353, 354
 algorithms 11, 29, 48, 287, 296,
 326, 343
 Anaconda 37, 38, 42, 59, 260

anaconda3 59, 226, 228–230, 235,
 276, 281
 analysis 8, 10–12, 18, 26, 27, 30–32,
 34, 35, 42, 43, 61–63, 69–72, 96, 108,
 137, 143, 164–168, 220, 231, 238,
 248, 259, 260, 275, 325, 326, 343, 355
 analytics 7, 9, 11–13, 15, 17, 18, 21,
 26–28, 30, 31, 38, 39, 42, 43, 61, 62,
 70, 71, 211, 247, 260
 ANOVA 175
 anscombe 9, 75, 224–230, 259
 api 78, 89, 178, 211, 225, 271, 272
 apriori 329, 333
 arima 318, 324, 325, 355
 array 30, 101, 107, 119, 178, 266, 282,
 285, 286, 290–294, 299, 350, 352, 353
 arules 326, 327, 336, 355
 arulesViz 326, 332
 aslugsguidetopython 35
 association 10, 275, 325, 326, 355
 auc 9, 218, 287, 296
 AUROC 217
 average 72, 129, 167, 169, 171, 191, 218
 avg 122, 130
 Azure 31, 41

b

BaggingClassifier 282, 289–291,
 298, 299
 barplot 237, 238, 248, 251, 253, 316

basicR 130, 148
 Bayes 308
 BeautifulSoup 79
 beautifulsoup4 60, 79
 bias 9, 68, 72, 73, 218, 220
 BIC 179, 226, 272
 bokeh 33, 221, 234, 247, 355
 Bonferonni 201, 208
 boston 190–195, 197, 199–205, 207,
 271, 272
 boxcox 205, 209
 boxplot 195, 248, 249, 269, 316
 bptest 199, 201, 205, 208
 Breusch-Pagan 199, 201, 205, 208
 byteacademy 28

C

calgary 79–81
 CAP 85
 caret 35–37, 304, 307
 caretr 36, 37
 Cassandra 86
 categorical 8, 143, 145, 168, 169, 237,
 238, 265
 Cauchy 172
 chaid 71, 300, 316, 317
 chisq 177
 chi-square 175
 chi-squared 177
 chisquaretest 176
 classification 275, 282, 291, 300,
 303, 307
 Clojure 31
 cloudera 40
 coefficients 37, 165, 181–184, 186,
 188, 198, 200, 202–204, 207, 216
 cognitive 72, 73
 colnames 114
 colors 238, 249, 251, 253, 343
 column-oriented 86
 confusionMatrix 304, 307
 contingency 178
 continuum 37, 38, 42, 168, 260, 276

Conway, Drew 27, 63
 core 40, 42, 43, 73, 77, 100, 106,
 107, 113, 234, 260, 263, 266, 280,
 285, 345
 corr 144, 145, 265, 266
 correlation 9, 63, 145, 166, 168, 211,
 224, 253, 265
 corrgram 194, 253, 254
 CouchDB 85, 86
 counter 85
 counterparts 233
 countries 64
 counts 23, 145, 262, 265, 266, 285,
 292, 346, 350
 couple 83
 coursera 34, 221
 courses 11, 34, 339
 Courtesy 86
 Covariance 179, 226, 272
 coxcomb 221, 223, 259
 cp 44, 86
 CPython 42
 cran 26, 33, 70, 102, 142, 247
 crawling 42
 Creighton, H.J. 106
 crim 190, 192, 193, 195, 197–202, 272
 Crimean 221
 criterion 18, 287, 290, 296, 299, 301,
 302
 critical 27
 crosstab 145, 266, 267
 Cross-Validated 308
 CSS 68
 csv 75–78, 84, 96, 106, 120, 122, 178,
 190, 211, 214, 225, 234, 237, 252,
 262, 263, 271, 278, 291, 344
 CTOs 21
 ctree 300, 306, 316
 customers 43, 64, 69–71, 94, 343
 cutgroup 266
 cutoff 166
 cv 290, 299, 308
 cx 81

cy 164
 cycle 228–230
 cython 40, 43

d

dashboard 9, 232, 259
 data 3, 4, 7–9, 11–13, 17–19, 21,
 25–44, 57, 61–63, 71–73, 75–81,
 84–86, 90, 94–97, 100–102, 104–
 106, 108, 112–116, 118–120,
 122–124, 129–132, 134, 137,
 142–145, 147–150, 158–160,
 164–170, 175, 177, 179, 181–188,
 190, 197, 199–205, 207, 208,
 211–215, 218, 220, 221, 224,
 226–234, 236–248, 252, 254, 256,
 259–263, 265–273, 276, 278, 280,
 288, 291, 297, 300, 303–307,
 314–318, 325–327, 330, 332, 334,
 336, 337, 342–345, 353
 database(s) 17, 27, 43, 84–87, 89,
 90, 93, 95, 96, 98, 100, 261, 291,
 325, 344
 DataCamp 34
 dataframe 35–37, 77, 100, 106–108,
 113, 148, 234, 263, 278, 280–285,
 291, 292, 345, 349
 datascience 28, 34, 38
 datavisualization 221
 date 8, 94, 100, 102–105, 137–139,
 179, 199, 213, 226, 272, 276, 279,
 280, 282, 283, 300, 305, 318
 datetime 102, 104, 143
 DB2 84, 89
 dBase 89
 dcast 159
 Debian 60
 decision 8, 10, 31, 62, 63, 71–73, 174,
 221, 273, 275, 282, 283, 291, 300
 decisionstats 11, 12, 15, 27, 45, 56, 58,
 71, 75, 76, 78, 104, 106, 110–112,
 120, 143, 147, 178, 224, 228, 234,
 243, 244, 258, 259, 275, 291, 337, 343

DecisionTreeClassifier 282, 289–291,
 298, 299
 DecisionTreeRegressor 286, 287, 296
 definition 26, 32, 40, 62, 63, 143
 degrees 50, 168, 182, 183, 185, 187,
 189, 198, 200, 202–204, 208, 216
 delete 54, 95, 100, 101, 115
 DELIMITER 96
 density 171, 243, 327, 332
 deploy 13, 42, 43
 depth 113–116, 119, 120, 131–133,
 136, 137, 149–153, 161, 162,
 187–189, 254, 263–266, 268, 286,
 287, 290, 296, 299
 Descriptive 152, 211
 desktop 43, 60–62, 76, 180, 190, 252,
 263, 276
 developers 26, 38
 deviation 72, 135, 169, 170, 174, 224
 diamonds 113–120, 130–137,
 149–152, 154, 156–162, 187, 188,
 237–243, 245–247, 253, 254,
 263–271
 diapers 11
 dimensions 114, 131, 221, 232
 Dimnames 327
 DirSource 340
 dispersion 9, 166, 169, 216
 distance 165, 166, 169, 217, 221
 distplot 235, 268
 distribute 28, 37
 distributed 29, 38, 43, 163, 171, 259
 distribution 9, 42, 43, 71, 135,
 169–171, 173, 175, 220, 235, 327,
 329, 332, 333, 343
 D3.js 33
 dmy 102, 103, 138
 DocumentTermMatrix 340
 Dostoevsky 338
 download 38, 42, 87, 89, 163, 257,
 259, 260, 288, 297, 337
 dplyr 37, 101, 145, 159
 drewconway 30, 63

dropbox 262, 276
 dropbox-dist 276
 dropna 115, 267, 283
 Drucker, Peter 62
 dtypes 77, 100, 106–108, 113, 114,
 117, 120, 179, 180, 212, 226–228,
 234, 263–266, 273, 281–285, 292,
 345–348, 350, 352

e

e1071 304, 307
 easyRFM 71
 EBook 338, 339
 EC2 41
 Edureka 61, 62
 ensemble 282, 287, 291, 296
 enterprisedb 87
 enthought 42, 260
 erikaandersen 167
 error 37, 42, 53, 72, 73, 96, 163, 164,
 174, 181–189, 198, 200, 202–204,
 207, 208, 216, 218–220, 235, 303, 307
 estimate 37, 181, 183, 184, 186, 188,
 198, 200, 202–204, 207, 216, 303, 307
 euclid 163, 259
 excel 78, 89
 exploratory 8, 18, 34, 143, 231,
 238, 248
 exponential 172

f

FacetGrid 238
 factor 67, 69, 132, 143, 149, 160, 187,
 215, 316, 317, 327
 factorial 50
 factorplot 238–242, 270
 false 103, 111, 112, 134, 140, 151, 174,
 201, 205, 218, 219, 268, 287, 289,
 290, 296, 298, 299, 309, 329, 333
 fancyRpartPlot 315
 feather 40
 fivenum 143
 forecast 318, 324, 355

format 40, 53, 55, 79, 95, 98, 101, 104,
 112, 114, 115, 138, 139, 145, 156,
 221, 260, 278, 327, 332
 formula 36, 156, 178, 179, 181, 182,
 184, 186, 188, 197, 200, 202–204,
 207, 215, 225–227, 262, 271, 272,
 303, 306, 317
 frame 8, 28, 35, 37, 77, 100, 105–107,
 113, 114, 116, 118, 122, 124, 131,
 132, 149, 160, 185, 187, 197, 234,
 237, 260, 263, 266, 267, 280, 285,
 316, 327, 342, 345
 frequency 70, 71, 134, 137, 145, 169,
 170, 275, 336, 341–343
 F-statistic 182, 183, 185, 187, 189,
 198, 200, 202–204, 208
 F-test 175
 function 12, 30, 47, 48, 57, 58, 82, 85,
 104, 105, 108, 111, 114, 115, 119,
 125, 137, 143–148, 166, 169, 171,
 180, 211, 212, 248, 262, 281, 346

g

gain 30, 234
 garbage 130
 gartner 30
 GDP 164
 Genesis 338
 genetic 11
 geom 228–230, 245–247, 271, 353
 geometric 94, 169, 243, 244
 Geringer, Steven 28
 gerstman 224
 getcwd 60, 61, 76, 262, 263, 275, 278
 getSources 340
 getwd 62, 180, 190
 ggplot 33, 224, 225, 228–230, 234,
 244–247, 261, 262, 271, 353–355
 ggplot2 8, 33, 37, 130, 131, 149, 187,
 214, 237, 254, 259, 261, 263, 304, 307
 gini 290, 299
 gist 45, 56, 58, 75, 76, 78, 104, 106,
 110–112, 120, 143, 147, 178, 224,

228, 234, 243, 244, 257, 258, 275,
291, 343
git 8, 44–46
github 26, 28, 34, 38, 39, 44–46, 70,
71, 75, 106, 120, 122, 147, 178, 224,
225, 234, 237, 257, 259, 263, 271,
326, 337
githubusercontent 337
glm 180, 211, 215, 217
globals 48, 49, 120, 267
glossary 30
GNP *see* gross national product (GNP)
God 11
Googlevis 221
grammar 33, 34, 243, 244, 260,
261, 271
graph 221, 222, 224, 231, 232, 248,
249, 288, 297, 331, 335, 336
graphviz 288, 297
Greenplum 86
Greiner, L.E. 66, 68, 73
grep 111, 112
grepinr 111
grepl 111, 140
Gretzky, Wayne 165
grid 156, 246, 255, 300, 305, 326
Groceries 326–330, 332–334, 336
gross national product (GNP) 164
groupby 145, 147, 266, 267
gsub 58, 109, 110, 115
GUIs 21, 174, 176, 355
guttenberg 338–340
gviz 288, 297
gvlma 166, 205

h

hackerearth 61, 62
Hacking 63
hadley 29, 142, 259, 271
hadoop 27, 29–31, 39, 40, 61, 62, 85
HairEyeColor 255
Harrison, D. 191
Harvard 27, 68, 73

hashtag 33
Hbase 31, 85, 86
HDFS 31
hebrew 277
Hemedinger 40
Heteroscedascity 9, 166, 211
hexbins 243
HighPerformanceComputing 33
Hilbert, M. 72, 73
histogram 248, 250, 252
Hive 31, 32
hmisc 101, 143, 148, 156, 159
hominem 71
HQL 27
href 80, 81
html 25, 26, 31, 33–35, 38, 43, 47, 48,
61, 62, 73, 78, 94, 104, 105, 115, 173,
175, 176, 211, 213, 220, 238, 247,
256, 259, 260, 262, 263, 268, 273,
275–278, 287, 296, 326
htmltools 62
https 23, 26–29, 33–35, 37–43, 45,
47, 48, 56, 58, 63, 64, 70, 71, 73, 75,
86, 87, 94, 102, 104–106, 110, 111,
120, 142, 147, 164, 165, 167, 168,
171, 178, 221, 224, 225, 234, 237,
238, 247, 256, 257, 259–261, 263,
275, 291, 316, 326, 337, 343, 346
HypothesisTesting 173, 259

i

IDE 9, 43, 256, 267
ifelse 103, 137, 140, 162
ijulia 259
import 12, 35–37, 49, 51, 52, 59, 60,
76, 79, 96, 97, 101, 102, 104, 108,
111, 112, 118, 120, 178, 211, 225,
234, 245, 262, 267, 271, 272, 275,
278, 282, 287, 288, 291, 296, 297,
337, 338, 344, 353
important 29, 31, 40, 41, 44, 57, 58,
165, 171, 232, 233, 260
indentation 25, 48, 260

index 10, 26, 29, 33, 43, 54, 56–58, 101, 106–109, 113, 114, 116, 119, 120, 179, 191, 207, 212, 256, 262, 264, 267, 268, 282, 284, 285, 337, 347, 348

India 4, 11, 256

info 58, 59, 77, 99, 106, 113, 132, 156, 234, 262, 263, 280, 330, 334, 337, 345, 348, 349

information 4, 21, 26, 30, 31, 58, 59, 72, 73, 84, 85, 90, 98, 110, 112, 115, 165, 220, 221, 232, 259, 263, 304, 307, 328, 332

Inglewood 83

install 37, 59, 60, 79, 89, 96, 131, 149, 205, 254, 260, 262, 287, 296, 297, 300, 302, 304, 307, 316, 326, 339, 340, 342, 353

int 26, 57, 94, 101, 109, 132, 150, 180, 187, 188, 197, 213, 226, 272, 327

intercept 37, 163, 180–184, 186, 188, 198, 200, 202–204, 207, 212, 213, 216, 226, 227, 272

io 34, 37–39, 42, 45, 68, 70, 75, 78, 106, 120, 122, 178, 225, 234, 237, 247, 259, 260, 262, 263, 271

ipynb 276–278, 337

ipython 32, 34, 37, 58–60, 257–259, 276, 288, 297

iris 178, 179, 181–184, 234–237, 251, 252, 260, 261, 300, 302–308, 314, 316, 317

IRkernel 37

IronPython 42

item 325, 326, 328, 329, 332, 333, 336

itemFrequencyPlot 328, 336

itemMatrix 327, 332

iter 56, 110, 346, 350

iterable 109

iterator 109

j

Jason, R. 73

java 27, 31, 39, 42, 276

JavaScript 31, 39

Javelin 125, 126, 128, 129

Jesus 5

JMP 15

jobs 290, 299, 350

John 4, 15, 53, 54, 56, 170, 220, 233

Johns 34

Johnson, S.R. 167, 220

JointGrid 243, 269

jointplot 243, 269

jre 277

json 78

jstatsoft 33

jstor 259

Julia 21, 33, 42

jupyter 7, 9, 37–40, 45, 56, 58, 61, 62, 75, 76, 78, 104, 106, 110–112, 120, 143, 147, 178, 224, 228, 234, 243, 244, 257, 260, 262, 273, 275, 277, 291, 343, 355

JVM 40, 43

Jython 42

k

kaggle 35, 218

kdnuggets 31, 34

kfold 289, 290, 298, 299

Kharagpur 15

kmeans 344, 346, 347, 349, 350

Kruskal–Wallis 175

kurtosis 170–172, 180, 227, 272

Kush 5, 11

l

lambda 120, 267

lapply 103, 317

LaTeX 21

lattice 156, 304, 307

lazyeval 326

leaflet 247

len 51, 56, 76, 77, 79, 108, 114, 118, 119, 263, 284, 285, 293, 338, 339

levels 17, 45, 132, 133, 143, 149, 150, 160, 168, 187, 310, 316, 327

- library 4, 32, 33, 39, 48, 75, 78, 79, 96, 102–105, 129, 131, 138, 141, 143, 149, 156, 158, 159, 176, 187, 190, 194, 199, 205, 214, 238, 244, 249, 252, 254, 255, 262, 275, 300, 302, 304–307, 314–316, 318, 326, 332, 336, 339, 340, 342
- linux 8, 11, 12, 37, 38, 44, 59, 60, 288, 297
- logistic 9, 164, 175, 180, 211, 213
- logisticmodels 218
- logisticregression 217
- logit 211–213, 218
- Log-Likelihood 213
- loops 26, 47, 110
- lubridate 102, 103, 137, 138
- m**
- machinelearningmastery 287, 296
- magrittr 62
- Mann–Whitney 175
- MapReduce 31
- maps 255, 278
- markdown 256, 262
- Maslow's 69
- math 28, 42, 49, 50, 63, 169, 256
- MATLAB 31
- matplotlib 33, 228–230, 234–238, 245, 344, 353
- matrix 64, 66, 107, 108, 161, 211, 217–219, 303, 304, 307, 326, 327, 341, 343, 352
- max 25, 51, 97, 131, 143, 149, 153, 156, 181, 183, 184, 186, 188, 190, 193, 194, 198, 200, 202–204, 207, 214, 215, 265, 279–281, 284–287, 290, 296, 299, 328, 330, 332, 334, 342–345, 350
- McKinney 29
- Mckinsey 64, 67
- MeanDecreaseGini 303
- means 4, 9, 85, 164, 226, 228, 231, 343, 346, 349
- median 143, 147, 153, 169, 170, 175, 181, 183, 184, 186, 188, 191, 193, 194, 198, 200, 202–204, 207, 214, 215, 266, 267, 328, 330, 332, 334
- medv 191–197, 199–205, 207, 272
- MemcacheDB 86
- memory 26, 29, 31, 40, 76, 100, 106, 113, 130, 131, 149, 190, 234, 263, 281, 339, 345
- methods 18, 35, 61, 70, 110, 262
- microsoft 31, 38, 41, 43
- min 51, 143, 153, 181, 183, 184, 186, 188, 193, 194, 198, 200, 202–204, 207, 214, 215, 265, 279–281, 284, 285, 287, 290, 296, 299, 328, 330, 332, 334, 344, 345
- Minard, Charles 221
- mining 18, 31, 33, 71, 259, 261, 273, 275, 315, 326, 330, 334, 337, 339
- mkdir 44
- model 10, 18, 28, 41, 64, 65, 67, 69, 72, 73, 84, 85, 163–166, 179, 181–183, 205, 211–213, 217, 218, 220, 226, 262, 272, 283, 287, 289, 290, 296, 298, 299, 308, 309, 314, 317, 318, 349–351
- modeling 9, 18, 69, 163, 218, 220, 261
- MongoDB 31, 85, 86
- mtcars 120, 122–130, 185, 186, 254
- multicollinearity 9, 166, 211
- munging 32, 130
- MySQL 31, 84, 86
- n**
- na 103, 110, 115, 116, 133, 134, 138, 151, 304, 307, 320–322
- NaiveBayes 314
- names 26, 57, 58, 94, 97, 98, 100, 108, 111, 112, 114, 122, 123, 131, 140, 141, 149, 181–183, 185, 264, 267, 284, 285, 288, 291, 297, 300, 342, 344, 346
- namespace 48, 62, 326

NaN 107, 115, 280, 281
 Napoleon's 221, 222
 nbviewer 45, 56, 58, 75, 76, 78, 104,
 106, 110–112, 120, 143, 147, 178,
 224, 228, 234, 243, 244, 258, 275,
 291, 343
 ncol 132, 149
 ndarray 107, 286, 293
 nltk 337–339
 nnet 37
 norm 72
 nosql 17, 32, 86
 notebook 9, 32, 34, 37–40, 61, 62,
 234, 245, 257, 258, 260, 262, 273
 nrow 131, 135, 149, 152, 205
 ntransactions 330, 334
 null 32, 94, 165, 173–175, 177, 178,
 216, 327
 numeric 61, 101, 104, 108–111, 115,
 143, 168, 199, 212, 217, 218, 283,
 300, 305, 318
 numpy 8, 30–32, 42, 51, 101, 105,
 107, 108, 118, 134, 178, 211, 224,
 225, 276, 281, 286, 293, 344, 355

O

ODBC *see* Open Database
 Connectivity (ODBC)
 Ohri, Ajay 3–5, 31, 259, 277
 Okun's 164
 Open Database Connectivity (ODBC)
 8, 89, 90, 96
 operator 101, 115, 132, 139, 140
 oracle 31, 41, 43, 84, 277
 Oracle R Enterprise (ORE) 43
 outlier 231
 outlierTest 166, 201, 208
 Overfitting 205, 218

P

package 4, 8, 12, 21, 26, 29, 32, 33, 35,
 37, 42, 43, 48, 50, 58, 70, 96, 97,
 102–104, 108, 114, 120, 129, 130,

141–143, 145, 147, 148, 156, 159,
 166, 190, 199, 214, 224, 237, 247,
 249, 252, 254, 255, 260–262, 267,
 268, 271, 273, 275, 300, 304, 305,
 307, 308, 316, 318, 326, 327, 337,
 339, 342, 355
 pairplot 237, 260, 261
 pandas 8, 29–32, 35, 37, 71, 75–78,
 97, 100, 101, 105–107, 113–115, 120,
 143, 145, 148, 178, 211, 224, 225,
 234, 260, 262, 263, 265, 266, 275,
 276, 280, 285, 291, 292, 344, 345, 355
 Pareto 69, 70
 partyR 300
 perceptualedge 232, 259
 pie 221, 259
 Pig 27, 31, 32
 pima 276, 291
 pip 58–60, 79, 260, 262, 287, 296,
 297, 331, 336, 353
 pivot 145, 148, 267
 Playfair, William 221
 plot 143, 196, 207, 217, 221, 234, 235,
 238, 243, 244, 247, 251, 254, 256,
 287, 296, 301, 303, 306, 314, 318,
 323, 324, 331, 335, 336
 Poisson 171, 173
 Polyglot 37
 POSIXt 156
 posterior 310
 postgresql 84, 86, 87, 94, 98, 100
 prediction 72, 73, 163, 165, 218, 220,
 304, 307
 price 69, 80, 81, 113–116, 119, 120,
 131–134, 136, 137, 149–153,
 156–162, 187, 188, 238–243,
 245–247, 253, 254, 263–271
 probability 9, 169–174, 218
 programming 12, 25–27, 30, 34, 43,
 62, 73, 84, 89, 167, 260
 projecteuclid 163, 259
 propensity 18, 211
 psycpg2 97

p-value 177, 178, 182, 183, 185, 187,
189, 198–205, 208, 213, 304, 307

pydoop 39

pypi 26, 70, 260, 261

pyplot 234, 344

pypy 42

Pyrex 43

PyRun 43

pysal 247

pysqldf 120–122, 267, 268

pythonanywhere 41

q

qnorm 175

quantiles 144

quartiles 143, 144

query 32, 85, 100, 119, 120, 129, 261,
262, 265

r

radimrehurek 34

Raleigh 28

random 8, 51, 52, 83, 118, 119, 130,
134, 135, 151, 163, 166, 170–172,
205, 207, 217, 287, 289, 290, 296,
298, 299, 303, 304, 307, 321, 350

randomforest 302, 303, 306,
307, 355

rattle 174, 176, 259, 275, 315,
326, 355

Rcolorbrewer 249, 342

RCommander 277

Rcpp 40, 62, 214

Rdatasets 75, 106, 120, 122, 178, 225,
234, 237, 259, 263, 271

RDBMS *see* relational database
management system (RDBMS)

re 80, 84, 97, 101, 108, 109, 111,
259, 277

readPDF 339

Redis 31, 86

RegModel 197, 199, 201, 203–205,
207–209

regression 7, 9, 18, 31, 34–37, 163–
166, 175, 178–181, 190, 205, 211, 213,
224, 226, 228, 231, 235, 262, 271, 275

relational 84, 86

relational database management
system (RDBMS) 8, 84–86

required 53, 94, 156, 255, 300, 304,
305, 307, 308, 316, 318, 326, 339, 342

Resampling 308, 309

reshape2 158

residual 165, 181–183, 185, 187, 189,
198, 200, 202–204, 208, 216

revolutionanalytics 38, 43

RevoScaleR 43

rexeranalytics 28

rfm 70, 71

rm 44, 115, 130, 131, 134, 149, 151,
190–205, 207, 208, 272, 273

RODBC 96

rodeo 213, 267, 271

rows 86, 107, 113, 114, 119, 131, 145,
252, 260, 264, 279, 327, 332, 349

rowSums 341

rpubs 48, 57, 59, 96, 102, 110, 111,
115, 122, 129, 130, 148, 174, 176,
180, 190, 213, 217, 218, 248, 253,
255, 256, 300, 316, 318, 326, 339

rpy2 35, 37, 39

R-squared 165, 182, 183, 185, 187,
189, 198, 200, 202–204, 208

rstudio 9, 40, 42, 43, 256, 271, 346

s

sample 134, 135, 151, 152, 166, 169,
174, 205, 218, 308

Scala 33

scatterplot 234, 236, 237

scientists 7–9, 13, 17, 21, 27, 28, 31, 32,
42, 44, 167, 168, 173, 232, 260, 262, 273

scikit 32, 33, 261, 273, 277, 278, 287,
296, 326, 355

scipy 31, 32, 176, 178, 226, 276

Scoring 216

- scrape 78
 - scraping 8, 17, 33, 42, 78, 84
 - scrapinghub 42
 - scrapy 42
 - SDK 43
 - seaborn 33, 97, 234–238, 243, 260–262, 268, 269
 - selection 8, 31, 116, 118, 119, 122, 124, 130, 147, 265, 287, 296
 - sensitivity 217, 219, 305, 308
 - sessionInfo 59, 61
 - shiny 37, 43, 221, 247
 - Siegel, Eric 15
 - SimpleDB 86
 - skewness 170–172
 - sm 178, 179, 211, 212, 225–227, 271, 272, 344, 352
 - SnowballC 340
 - sns 97, 234–243, 268–270
 - source 4, 11, 12, 19, 21–23, 26, 28, 34, 37, 40, 41, 43, 45, 46, 59, 63–68, 78, 86, 90, 160, 167, 168, 170–173, 191, 222, 224, 244, 260, 316
 - spark 31, 71
 - sparse 23, 260, 327, 332, 340, 341
 - spatial 9, 221, 247, 255, 256
 - specificity 219, 305, 308
 - spotfire 43
 - Springer 259
 - spss 21, 71
 - Spyder 43
 - sql 27, 31, 32, 78, 84, 85, 94, 100, 120, 129, 147, 148, 261, 267
 - sqlalchemy 60, 97, 261
 - sqldf 32, 120, 129, 130, 148, 261, 267
 - SQLite 267
 - Sqoop 31
 - stackexchange 166, 168
 - stackoverflow 37, 38, 48
 - standard 4, 9, 25, 40, 44, 71, 89, 94, 135, 164, 169–171, 174, 182, 183, 185, 187, 189, 198, 200, 202–204, 208, 220, 224, 226, 228
 - stata 78
 - statistical 9, 11–13, 18, 19, 22, 27, 30, 32–34, 43, 72, 73, 163, 165–169, 244, 259–261
 - statistics 9, 11, 26–28, 30, 31, 33, 34, 62, 63, 165–169, 173, 211, 220, 224, 228, 231, 259, 260, 304, 307, 308
 - statisticsviews 273
 - statsmodels 33, 60, 176, 178, 179, 211, 225, 235, 261, 271, 272, 318, 355
 - status 191
 - std 37, 51, 143, 180, 181, 183, 184, 186, 188, 198, 200, 202–204, 207, 213, 216, 226, 228, 265, 272, 280, 281, 284, 285, 344, 345
 - stochastically 163
 - str 53, 56, 57, 79, 80, 84, 99, 101, 102, 109, 112, 113, 122, 124, 132, 141, 149, 160, 185, 187, 197, 316, 326, 338
 - stringi 62
 - StringIO 288, 297
 - stringr 62, 141, 142
 - substr 58, 137, 139, 140
 - sudo 44, 59, 60, 262, 276, 287, 288, 296, 297, 353
 - summarise 160
 - summary 10, 18, 37, 143, 147, 148, 152, 168, 179, 181, 182, 184, 186, 188, 190, 193, 197, 199, 201, 203, 207, 211, 213–215, 224, 228, 231, 265, 266, 272, 308, 327, 329, 330, 332–334, 340, 355
 - SVM 276, 277
 - swarmplot 236, 237
- t**
- tableplot 253
 - tables 60, 93
 - TermDocumentMatrix 340, 341
 - textmining 339
 - tibble 188
 - tibco 43
 - Tidy 142

tidyr 37, 142, 159

Tufte's 9, 231

Tukey's 143

tuple 55

u

ubuntu 11, 61, 288, 297

ucla 211, 214

unicorn 27, 28

UNIX 30

v

van Rossum, Guido 25, 260

VCorpus 340

vincentarelbundock 75, 106, 120,
122, 178, 225, 234, 237, 259,
263, 271

visualization 9, 18, 27, 31–34, 218,
221, 228, 231, 234, 243, 248, 255,
259–261, 263, 265, 267–269, 271,
273, 343

Voldemort 86

w

wajig 277

Weibull 172

wesmckinney 29

wickham 29, 142, 243, 259, 271

wikipedia 64, 66, 67, 164–166, 171,
173, 256

Wilcoxon 175

Wilkinson 243, 271

Williams, G.J. 259

wolfram 173, 259

x

xbar 174

XlsxWriter 60

XMLSource 340

X-squared 177, 178

xtabs 158, 215

y

yhat 213, 244, 267, 271

yhathq 33, 261

yourlogicalfallacyis 71

z

Zen 7, 23

zia 27, 30, 63

Z-score 170

ztest 176