

Hands-On Data Science with Anaconda

Utilize the right mix of tools to create high-performance
data science applications



Packt

www.packt.com

By Dr. Yuxing Yan and James Yan

Hands-On Data Science with Anaconda

Utilize the right mix of tools to create high-performance data science applications



Dr. Yuxing Yan
James Yan



BIRMINGHAM - MUMBAI

Hands-On Data Science with Anaconda

Copyright © 2018 Packt Publishing All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Commissioning Editor: Pravin Dhandre

Acquisition Editor: Divya Poojari

Content Development Editor: Dattatraya More

Technical Editor: Nirbhaya Shaji

Copy Editor: Safis Editing

Project Coordinator: Shweta H Birwatkar

Proofreader: Safis Editing

Indexer: Tejal Daruwale Soni

Graphics: Jisha Chirayil

Production Coordinator: Shantanu Zagade

First published: May 2018

Production reference: 1300518

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-78883-119-2

www.packtpub.com



To my bosses: Mark Zaporowski (Canisius College), K.G. Viswanathan (Hofstra University), Lisa Fairchild (Loyola University), John Doe (Wharton School), David Ding (Nanyang Technological University), and Ben Amoako-Adu (Wilfrid Laurier University).

– Yuxing Yan

To my dad, mom, and sister.

–James Yan



mapt.io

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

PacktPub.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributors

About the authors

Dr. Yuxing Yan graduated from McGill University with a PhD in finance. He has taught various finance courses at eight universities in Canada, Singapore, and the U.S. He has published 23 research and teaching-related papers, and is the author of 6 books. Two of his recent publications are *Python for Finance* and *Financial Modelling using R*. He is well-versed in R, Python, SAS, MATLAB, Octave, and C. In addition, he is an expert on financial data analytics.

I thank Ben Amoako-Adu, Brian Smith, Jin-Chun Duan, Jerome Detemple, Lawrence Kryzanowski, Chris Schull, Mark Keintz, Dong Xu, Eric Zhu, Paul Ratnaraj, Premal Vora, Shuguang Zhang, Mireia Gine, Shaojun Zhang, Qian Sun, Shaobo Ji, Xing Zhang, Changwen Miao, Karyl Leggio, K. G. Viswanathan, Mark Lennon, Qiyu Zhang, Xiaoning (my wife), Jing (my daughter) and James (my son) for their help and support.

James Yan is an undergraduate student at the **University of Toronto (UofT)**, currently double-majoring in computer science and statistics. He has hands-on knowledge of Python, R, Java, MATLAB, and SQL. During his study at UofT, he has taken many related courses, such as *Methods of Data Analysis I and II*, *Methods of Applied Statistics*, *Introduction to Databases*, *Introduction to Artificial Intelligence*, and *Numerical Methods*, including a capstone course on AI in clinical medicine.

About the reviewer

Justin (Byung Uk) Lee completed his BA and master's in computer science at KAIST. He developed Korean Windows CE 1.0 and 2.0 at Microsoft while working for LG Electronics. Later, he ran his own business for more than 7 years, which proposed custom-tailored financial portfolios derived from data analysis. He then worked for several life and non-life insurers, including Samsung Life as a CMO and CSMO conducting CRM-based marketing. Currently, he intensively researches machine learning based big data finance analysis and financial applications using blockchain.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit authors.packtpub.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Title Page
Copyright and Credits
Hands-On Data Science with Anaconda
Dedication
Packt Upsell
Why subscribe?
PacktPub.com
Contributors
About the authors
About the reviewer
Packt is searching for authors like you
Preface
Who this book is for
What this book covers
To get the most out of this book
Download the example code files
Download the color images
Conventions used
Get in touch
Reviews
1. Ecosystem of Anaconda
Introduction
Reasons for using Jupyter via Anaconda

[Using Jupyter without pre-installation](#)

[Miniconda](#)

[Anaconda Cloud](#)

[Finding help](#)

[Summary](#)

[Review questions and exercises](#)

2. [Anaconda Installation](#)

[Installing Anaconda](#)

[Anaconda for Windows](#)

[Testing Python](#)

[Using IPython](#)

[Using Python via Jupyter](#)

[Introducing Spyder](#)

[Installing R via Conda](#)

[Installing Julia and linking it to Jupyter](#)

[Installing Octave and linking it to Jupyter](#)

[Finding help](#)

[Summary](#)

[Review questions and exercises](#)

3. [Data Basics](#)

[Sources of data](#)

[UCI machine learning](#)

[Introduction to the Python pandas package](#)

[Several ways to input data](#)

[Inputting data using R](#)

[Inputting data using Python](#)

[Introduction to the Quandl data delivery platform](#)

[Dealing with missing data](#)

[Data sorting](#)

[Slicing and dicing datasets](#)

[Merging different datasets](#)

[Data output](#)

[Introduction to the cbsodata Python package](#)

[Introduction to the datadotworld Python package](#)

[Introduction to the haven and foreign R packages](#)

[Introduction to the dslabs R package](#)

[Generating Python datasets](#)

[Generating R datasets](#)

[Summary](#)

[Review questions and exercises](#)

4. Data Visualization

[Importance of data visualization](#)

[Data visualization in R](#)

[Data visualization in Python](#)

[Data visualization in Julia](#)

[Drawing simple graphs](#)

[Various bar charts, pie charts, and histograms](#)

[Adding a trend](#)

[Adding legends and other explanations](#)

[Visualization packages for R](#)

[Visualization packages for Python](#)

[Visualization packages for Julia](#)

[Dynamic visualization](#)

[Saving pictures as pdf](#)

[Saving dynamic visualization as HTML file](#)

[Summary](#)

[Review questions and exercises](#)

5. Statistical Modeling in Anaconda

[Introduction to linear models](#)

[Running a linear regression in R, Python, Julia, and Octave](#)

[Critical value and the decision rule](#)

[F-test, critical value, and the decision rule](#)

[An application of a linear regression in finance](#)

[Dealing with missing data](#)

[Removing missing data](#)

[Replacing missing data with another value](#)

[Detecting outliers and treatments](#)

[Several multivariate linear models](#)

[Collinearity and its solution](#)

[A model's performance measure](#)

[Summary](#)

[Review questions and exercises](#)

6. Managing Packages

[Introduction to packages, modules, or toolboxes](#)

[Two examples of using packages](#)

[Finding all R packages](#)

[Finding all Python packages](#)

[Finding all Julia packages](#)

[Finding all Octave packages](#)

[Task views for R](#)

[Finding manuals](#)

[Package dependencies](#)

[Package management in R](#)

[Package management in Python](#)

[Package management in Julia](#)

[Package management in Octave](#)

[Conda – the package manager](#)

[Creating a set of programs in R and Python](#)

[Finding environmental variables](#)

[Summary](#)

[Review questions and exercises](#)

7. Optimization in Anaconda

[Why optimization is important](#)

[General issues for optimization problems](#)

[Expressing various kinds of optimization problems as LPP](#)

[Quadratic optimization](#)

[Optimization in R](#)

[Optimization in Python](#)

[Optimization in Julia](#)

[Optimization in Octave](#)

[Example #1 – stock portfolio optimization](#)

[Example #2 – optimal tax policy](#)

[Packages for optimization in R](#)

[Packages for optimization in Python](#)

Packages for optimization in Octave

Packages for optimization in Julia

Summary

Review questions and exercises

8. Unsupervised Learning in Anaconda

Introduction to unsupervised learning

Hierarchical clustering

k-means clustering

Introduction to Python packages – scipy

Introduction to Python packages – contrastive

Introduction to Python packages – sklearn (scikit-learn)

Introduction to R packages – rattle

Introduction to R packages – randomUniformForest

Introduction to R packages – Rmixmod

Implementation using Julia

Task view for Cluster Analysis

Summary

Review questions and exercises

9. Supervised Learning in Anaconda

A glance at supervised learning

Classification

The k-nearest neighbors algorithm

Bayes classifiers

Reinforcement learning

Implementation of supervised learning via R

Introduction to RTextTools

[Implementation via Python](#)

[Using the scikit-learn \(sklearn\) module](#)

[Implementation via Octave](#)

[Implementation via Julia](#)

[Task view for machine learning in R](#)

[Summary](#)

[Review questions and exercises](#)

10. Predictive Data Analytics – Modeling and Validation

[Understanding predictive data analytics](#)

[Useful datasets](#)

[The AppliedPredictiveModeling R package](#)

[Time series analytics](#)

[Predicting future events](#)

[Seasonality](#)

[Visualizing components](#)

[R package – LiblineaR](#)

[R package – dataRobot](#)

[R package – ecluster](#)

[Model selection](#)

[Python package – model-catwalk](#)

[Python package – sklearn](#)

[Julia package – QuantEcon](#)

[Octave package – ltfat](#)

[Granger causality test](#)

[Summary](#)

[Review questions and exercises](#)

11. Anaconda Cloud

Introduction to Anaconda Cloud

Jupyter Notebook in depth

Formats of Jupyter Notebook

Sharing of notebooks

Sharing of projects

Sharing of environments

Replicating others' environments locally

Downloading a package from Anaconda

Summary

Review questions and exercises

12. Distributed Computing, Parallel Computing, and HPCC

Introduction to distributed versus parallel computing

Task view for parallel processing

Sample programs in Python

Understanding MPI

R package Rmpi

R package plyr

R package parallel

R package snow

Parallel processing in Python

Parallel processing for word frequency

Parallel Monte-Carlo options pricing

Compute nodes

Anaconda add-on

Introduction to HPCC

[Summary](#)

[Review questions and exercises](#)

[References](#)

[Chapter 01: Ecosystem of Anaconda](#)

[Chapter 02: Anaconda Installation](#)

[Chapter 03: Data Basics](#)

[Chapter 04: Data Visualization](#)

[Chapter 05: Statistical Modeling in Anaconda](#)

[Chapter 06: Managing Packages](#)

[Chapter 07: Optimization in Anaconda](#)

[Chapter 08: Unsupervised Learning in Anaconda](#)

[Chapter 09: Supervised Learning in Anaconda](#)

[Chapter 10: Predictive Data Analytics – Modelling and Validation](#)

[Chapter 11: Anaconda Cloud](#)

[Chapter 12: Distributed Computing, Parallel Computing, and HPCC](#)

[Other Books You May Enjoy](#)

[Leave a review - let other readers know what you think](#)

Preface

Anaconda is an open source data science platform that brings the best tools for data science together. It is a data science stack that includes more than 100 popular packages based on Python, Scala, and R. With the help of its package manager, `conda`, users can work with hundreds of packages in different languages and perform data preprocessing, modeling, clustering, classification, and validation with ease.

This book will get you started with Anaconda and how you can use it to perform data science operations in the real world. You will start of setting up the environment for the Anaconda platform, Jupyter, and installing the relevant packages. You will then cover the basics of data science and linear algebra for performing data science tasks. Once you are ready to go, you will start with data science operations such as cleaning, sorting, and data classification. You will then learn how to perform tasks such as clustering, regression, prediction, building machine learning models, and optimizing them. You will also learn how to visualize data and share the projects.

During this course, you will learn how to use different packages, using Anaconda to get the best results. You will learn how to efficiently use `conda` — the package, dependency, and environment manager for Anaconda. You will also be introduced to several powerful features of Anaconda, such as additional projects, project add-ons, shared project drives, and powerful compute nodes that are available in the paid version for accomplishing advanced data handling processes. You will learn how to build scalable and functionally efficient packages, and how to perform heterogeneous data exploration, distributed computing, and more. You will learn to discover and share packages, notebooks, and environments to increase productivity. You will also learn about Anaconda Accelerate, a feature that can help you to achieve SLAs easily and optimize computational power.

In this book, we introduce four programming languages: R, Python, Octave, and Julia. There are several reasons for doing so. Firstly, all four are open source, which is one of the future trends. Secondly, one of the most obvious advantages

to using the Anaconda platform is that it allows you to where we could implement many programs written in different languages. However, for many new readers, learning four languages at the same time would be quite challenging. The best strategy is to focus on R and Python first. After a while, or after finishing the whole book, learn Octave or Julia on the second reading.

- **R:** This is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, such as Windows and macOS. We think that R might be the easiest of many good computer languages, especially those that offer free software. The author has published a book entitled *Financial Modeling using R*; you can refer to its Amazon link at <http://canisius.edu/~yany/webs/amazon2018R.shtml>.
- **Python:** This is an interpreted high-level programming language for general-purpose programming. For business analytics/data science, Python is probably the number 1 choice out of many promising computer languages. In 2017, the author published a book entitled *Python for Finance* (second edition); you can refer to its Amazon link at <http://canisius.edu/~yany/webs/amazonP4F2.shtml>.
- **Octave:** This is a piece of software featuring a high-level programming language, primarily intended for numerical computations. Octave helps with solving linear and nonlinear problems numerically, as well as performing other numerical experiments. Octave is also free. Its syntax is largely compatible with MATLAB, which is quite popular on Wall Street and in other industries.
- **Julia:** This is a high-level, high-performance dynamic programming language for numerical computing. It provides a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library. Julia's base library, largely written in Julia itself, also integrates mature, best-of-breed, open source C and Fortran libraries for linear algebra, random number generation, signal processing, and string processing.

Happy reading!

Who this book is for

Hands-On Data Science with Anaconda is for you if you are a developer who is looking for the best tools on the market to perform data science operations. It's also ideal for data analysts and data science professionals who want to improve the efficiency of their data science applications using the best libraries in multiple languages. Basic programming knowledge with R or Python and basic knowledge of linear algebra is expected.

What this book covers

[Chapter 1](#), *Ecosystem of Anaconda*, introduces some basic concepts such as the reasons why we use Anaconda and the advantages of using a full-fledged Anaconda and/or its baby version, Miniconda. Then, it covers the use of Anaconda online, without installation. We also test a few simple programs, written in R, Python, Julia, and Octave.

[Chapter 2](#), *Anaconda Installation*, shows how to install Anaconda, test whether the installation is successful, how to launch Jupyter and use it to launch Python, how to launch Spyder and R, and how to find help. Most of these concepts or procedures are quite basic, so users who are quite confident with them can skip this chapter and go directly to the next chapter.

[Chapter 3](#), *Data Basics*, discusses sources of open data, which include the Bureau of Labor Statistics, the Census Bureau, Professor French's Data Library, the Federal Reserve's Data Library, and the UCI (University of California at Irvin) Machine Learning Repository. After that, it explains how to input data; how to deal with missing data; how to sort, slice, and dice datasets; how to merge different datasets and data output. For different languages, such as Python, R, Julia and Octave, several relevant packages for data manipulation are introduced and discussed.

[Chapter 4](#), *Data Visualization*, discusses various types of visual presentations, which include simple graphs, bar charts, pie charts, and histograms, written in different languages such as R, Python, and Julia. Visual presentations can help our audience understand our data better. For many complex concepts or theories, we could use visual presentations to help explain their logic and complexity. A typical example is the so-called bisection method or bisection search.

[Chapter 5](#), *Statistical Modeling in Anaconda*, explains many important issues related to statistics, such as T-distribution, F-distribution, T-test, and F-test. We also discuss linear regression, how to deal with missing data, how to treat outliers, collinearity and its treatments, and how to run a multi-variable linear regression.

[Chapter 6](#), *Managing Packages*, explains the importance of managing packages, how to find out all packages available for R, Python, and Julia, and how to find the manual for each package. In addition, we discuss the issue of package dependency and how to make our programming a little easier when dealing with packages.

[Chapter 7](#), *Optimization in Anaconda*, discusses several optimization topics, including general optimization problems, expressing various kinds of optimization problems as LPPs, and quadratic optimization. Several examples are offered to make our discussion more practice-oriented, such as how to choose an optimal stock portfolio, how to optimize wealth and resources to promote sustainable development, and how much the government should really tax people. In addition, we introduce several packages for optimization in R, Python, Julia, and Octave.

[Chapter 8](#), *Unsupervised Learning in Anaconda*, covers unsupervised learning. In particular, hierarchical clustering and k-means clustering are covered. As for R and Python, several related packages are looked at in details. For R: `rattle`, `Rmixmod`, and `randomUniformForest`; For Python: `Scipy.cluster`, `Contrastive`, and `sklearn`.

[Chapter 9](#), *Supervised Learning in Anaconda*, discusses supervised learning, including classification, k-nearest neighbors algorithm, Bayes' classifiers, reinforcement learning, and specific R and Python-related modules, such as `RTextTools` and `sklearn`. In addition, you will see their implementation in R, Python, Julia, and Octave.

[Chapter 10](#), *Predictive Data Analytics – Modelling and Validation*, covers predictive data analytics, modeling and validation, some useful datasets, time series analytics, how to predict future events, seasonality, and how to visualize our data. We mention `prsklearn` and `catwalk` for Python, `datarobot`, `LiblineaR`, and `eclust` for R, `QuantEcon` for Julia and `ltfat` for Octave.

[Chapter 11](#), *Anaconda Cloud*, discusses Anaconda Cloud. Some topics include Jupyter Notebook in depth, different formats of Jupyter notebooks, how to share notebooks with your partners, how to share different projects over different platforms, how to share your working environments, and how to replicate other's environments locally.

[Chapter 12](#), *Distributed Computing, Parallel Computing, and HPCC*, covers distributed computing and Anaconda Accelerate. When our data or tasks become more complex, we need a good system or a set of tools to process data and run complex algorithms. For this purpose, distributed computing is one solution. In particular, we will explain compute nodes, project add-ons, parallel processing, and advanced Python for data parallelism.

To get the most out of this book

The chapters in this book require a PC or Mac with 8GB or 16GB of RAM (the higher, the better). Your machine should have at least a 2.2 GHz Core i3/i5 processor or an AMD equivalent.

Download the example code files

You can download the example code files for this book from your account at www.packtpub.com. If you purchased this book elsewhere, you can visit www.packtpub.com/support and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at www.packtpub.com.
2. Select the SUPPORT tab.
3. Click on Code Downloads & Errata.
4. Enter the name of the book in the Search box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Ziipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at <https://github.com/PacktPublishing/Hands-On-Data-Science-with-Anaconda>. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: https://www.packtpub.com/sites/default/files/downloads/HandsOnDataSciencewithAnaconda_ColorImages.pdf.

Conventions used

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, path names, dummy URLs, user input, and Twitter handles are shown as follows: "The most widely used Python package for graphs and images is called `matplotlib`."

A block of code is set as follows:

```
| import matplotlib.pyplot as plt  
| plt.plot([2,3,8,12])  
| plt.show()
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
| import matplotlib.pyplot as plt  
| plt.plot([2,3,8,12])  
| plt.show()
```

Any command-line input or output is written as follows:

```
| install.packages("rattle")
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "For the sources of data, we choose from seven potential formats, such as File, ARFF, ODBC, R Dataset, RData File, and we can load our data from there."



Warnings or important notes appear like this.



Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: Email feedback@packtpub.com and mention the book title in the subject of your message. If you have questions about any aspect of this book, please email us at questions@packtpub.com.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit www.packtpub.com/submit-errata, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at copyright@packtpub.com with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit authors.packtpub.com.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit packtpub.com.

Ecosystem of Anaconda

In the preface, we mentioned that this book is designed for readers who are looking for tools in the area of data science. Existing data analysts and data science professionals who wish to improve the efficiency of their data science applications by using the best libraries with multiple languages will find this book quite useful. The platform discussed in detail across various chapters is **Anaconda** and the computational tools could be **Python**, **R**, **Julia**, or **Octave**. The beauty of using these programming languages is that they are all open source, as in free to download. In this chapter, we start from the very beginning: a simple introduction. For this book, we assume that readers have some basic knowledge related to several programming languages, such as R and Python. There are many books available, such as *Python for Data Analysis* by McKinney (2013) and *Python for Finance* by Yan (2017).

In this chapter, the following topics will be covered:

- Introduction
- Miniconda
- Anaconda Cloud
- Finding help

Introduction

Nowadays, we are overwhelmed by large amounts of information—see Shi, Zhang, and Khan (2017), or Fang and Zhang (2016)—the catchphrase being big data. However, defining it is still controversial, since many explanations are available. Davenport and Patil (2012) suggest that if your organization stores multiple petabytes of data, if the information most critical to your business resides in forms other than rows and columns of numbers, or if answering your biggest question would involve a mashup of several analytical efforts, you've got a big data opportunity.

Many users of data science or data analytics are learning several programming languages such as R and Python, but how can they use both of them at the same time? If John is using R while his teammate is using Python, how do they communicate with each other? How do team members share their packages, programs, and even their working environments? In this book, we try our best to offer a solution to all of these challenging tasks by introducing Anaconda, since it possesses several wonderful properties.

Generally speaking, *R* is a programming language for statistical computing and graphics that is supported by the R Foundation for statistical computing. *Python* is an interpreted, object-oriented programming language similar to Perl that has gained popularity because of its clear syntax and readability. *Julia* is for numerical computing and extensive mathematical function and is designed for parallelism and cloud computing, while *Octave* is for numerical computation and mathematics-oriented and batch-oriented language. All those four languages, R, Python, Julia, and Octave, are free.

Reasons for using Jupyter via Anaconda

In data science or data analytics, we usually work in a team. This means that each developer, researcher, or team member, might have his/her favorite programming language, such as Python, R, Octave, or Julia. If we could have a platform to run all of those languages, it would be great. Fortunately, Jupyter is such a platform, since this platform can accommodate over 40 languages, including Python, R, Julia, Octave, and Scala.

In [Chapter 2, Anaconda Installation](#), we will show you how to run those four languages via Jupyter. Of course, there are other benefits of using Anaconda: we might worry less about the dependency of installed packages, manage packages more efficiently, and share our programs, projects, and working environments. In addition, Jupyter Notebooks can be shared with others using email, Dropbox, GitHub, and the Jupyter Notebook Viewer.

Using Jupyter without pre-installation

In [Chapter 2](#), *Anaconda Installation*, we will discuss how to install Jupyter via Anaconda installation. However, we could launch Jupyter occasionally without pre-installation by going to the web page at <https://jupyter.org/try>:

1. The welcome screen will be presented with various options for trying out different languages.
2. For example, by clicking the Try Jupyter with Julia image, we would see the following screen:

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

Julia 0.6.0



This is a simple Julia 0.6 notebook. Below we'll show off the Julia `PyPlot` module.

```
In [1]: using PyPlot  
ioff() # Interactive plotting OFF, necessary for inline plotting in IJulia
```

```
In [2]: x = 100 * rand(500)  
y = 100 * rand(500)  
areas = 800 * rand(500);
```

```
In [3]: # Scatterplot  
fig = figure("scatterplot", figsize=(10,10))  
ax = axes()  
scatter(x,y,s=areas,alpha=0.5, c=areas)  
  
title("Scatter Plot")  
xlabel("X")  
ylabel("Y")  
grid("on")  
gcf();
```

3. To save space, the screenshot shows only the first part of the demo. Any readers could try the previous two steps to view the whole demo. In addition, if we click the Try Jupyter with R image, the following screen would show:

File Edit View Insert Cell Kernel Widgets Help

Not Trusted

R O

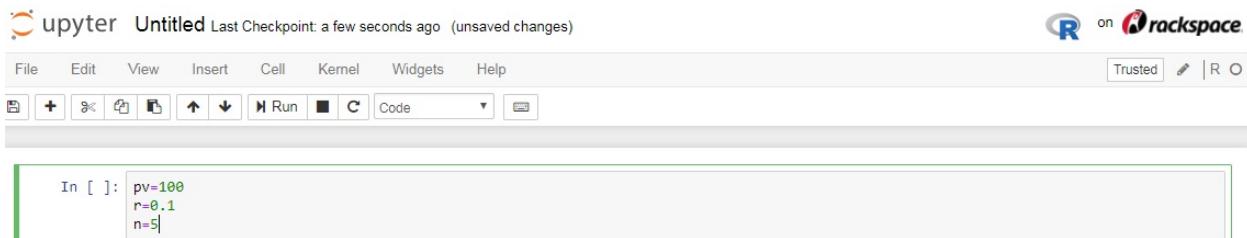
R Code pulled from <https://www.statmethods.net/advgraphs/ggplot2.html>

```
In [1]: # ggplot2 examples  
library(ggplot2)
```

```
In [2]: # create factors with value labels  
mtcars$gear <- factor(mtcars$gear,levels=c(3,4,5),  
                         labels=c("3gears","4gears","5gears"))  
mtcars$am <- factor(mtcars$am,levels=c(0,1),  
                         labels=c("Automatic","Manual"))  
mtcars$cyl <- factor(mtcars$cyl,levels=c(4,6,8),  
                         labels=c("4cyl","6cyl","8cyl"))
```

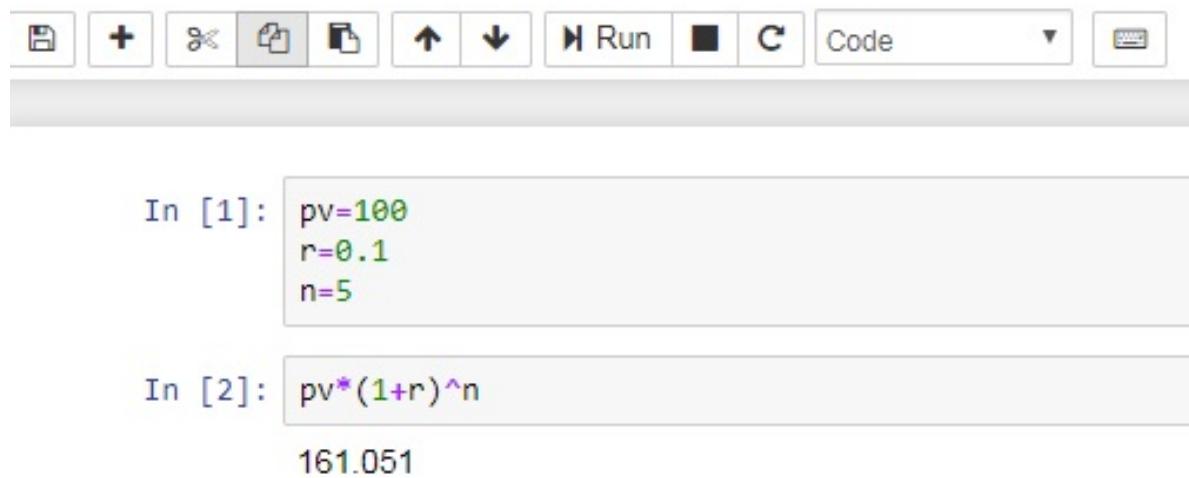
```
In [3]: # Kernel density plots for mpg  
# grouped by number of gears (indicated by color)  
qplot(mpg, data=mtcars, geom="density", fill=gear, alpha=I(.5),  
      main="Distribution of Gas Milage", xlab="Miles Per Gallon",  
      ylab="Density")
```

- After selecting Try Jupyter with Python, you will be presented with the welcome screen for the same.
- Next, we will show you how to execute a few simple commands in R, Python, and Julia. For example, we could use R to use the platform to run a few simple command lines. In the following example, we enter `pv=100`, `r=0.1`, and `n=5`:



In []: `pv=100`
`r=0.1`
`n=5`

- After clicking the Run button on the menu bar, we assign those values to the three variables. Then we can estimate the future value of this present value, as illustrated here:

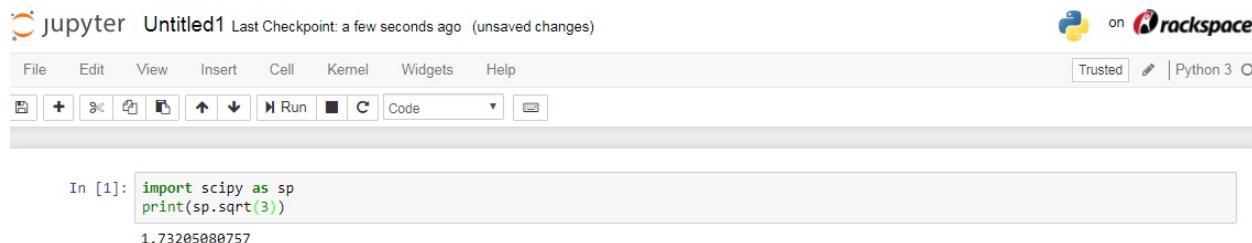


In [1]: `pv=100`
`r=0.1`
`n=5`

In [2]: `pv*(1+r)^n`

161.051

- Similarly, we could try to use Python, as shown here:

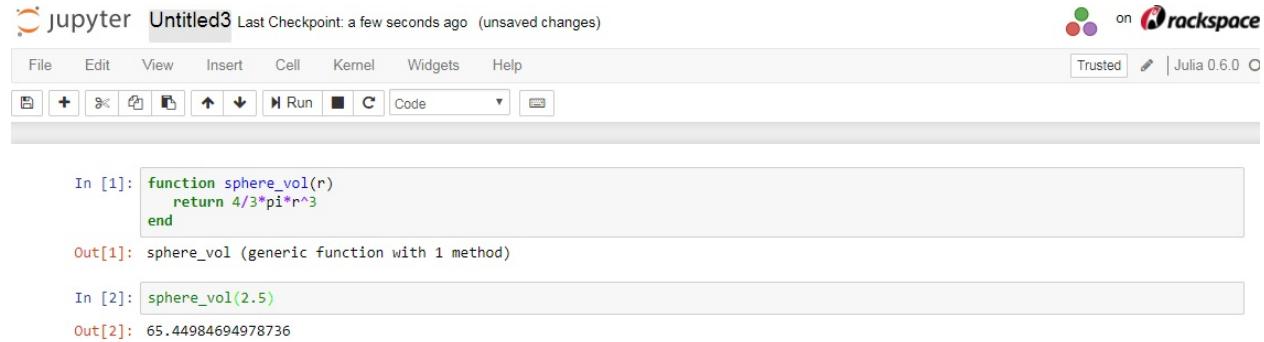


In [1]: `import scipy as sp`
`print(sp.sqrt(3))`

1.73205080757

In the preceding example, we import the Python package called `scipy` and give it a short name, `sp`. Although other short names could be used to represent the `scipy` package, it is a convention to use `sp`. Then, we use the `sqrt()` function included in the Python package.

For Julia, we could try the following code (shown in the following screenshot). Again, after going to File|New on the menu, we choose Julia 0.6.0. As of May 09, 2018, 0.6.0 is the current version for Julia. Note that your current version for Julia could be different:



The screenshot shows a Jupyter Notebook interface running on Rackspace. The top bar includes the Jupyter logo, the notebook title "Untitled3", a last checkpoint timestamp, and a "Trusted" button. The toolbar below has icons for file operations like New, Open, Save, and Run, along with Kernel and Widgets options. The main area is divided into two code cells. Cell [1] contains the definition of a function `sphere_vol(r)` which returns $\frac{4}{3}\pi r^3$. Cell [2] shows the result of running this function with input `2.5`, outputting the value `65.44984694978736`.

```
In [1]: function sphere_vol(r)
    return 4/3*pi*r^3
end

Out[1]: sphere_vol (generic function with 1 method)

In [2]: sphere_vol(2.5)
Out[2]: 65.44984694978736
```

In the code, we define a function called `sphere_vol` with just one input value of `r` (in radians). The answer is 64.45 for an input value of 2.5.

Miniconda

Anaconda is a full distribution of Python and comes with over 1,000 open source packages after installation. Because of this, the total size is over 3 GB. Anaconda is good if we intend to have many packages downloaded and pre-installed. On the other hand, Miniconda contains only Python and other necessary libraries needed to run conda itself. The size for the Miniconda is about 400 MB, much smaller than the full version of Anaconda, so extra packages have to be downloaded and installed as requested.

There are many reasons why a new user might prefer a watered-down version of Anaconda. For example, they might not need so many packages. Another reason is that users might not have enough space. Those users could download Miniconda at <https://conda.io/miniconda.html>. Again, in *Chapter 2, Anaconda Installation*, we will discuss in detail how to install Anaconda and run programs written in different languages, such as Python, R, Julia, and Octave.

Anaconda Cloud

In [Chapter 2](#), *Anaconda Installation*, we'll explain this in more detail. This function is used to collaborate with different users or group members. For example, we have a small group of ten developers working on the same project. For this reason, we have to share our programs, command datasets, and working environments, and we could use Anaconda Cloud to do so. After going to <https://anaconda.org/>, we will be directed to the Anaconda home page.

Note that users have to register with Anaconda before they can use this function. For example, one of the authors has the link <https://anaconda.org/paulyan/dashboard>. After we register, we can see the following:

My Anaconda Landscape

? Packages[View all \(0\)](#)

Get more information on how to [upload a Package](#).

? Notebooks[View all \(0\)](#)

Get more information on how to [upload a Notebook](#).

? Environments[View all \(0\)](#)

Get more information on how to [upload an Environment](#).

? Projects[View all \(0\)](#)

No projects yet, [upload one here](#).

★ Favorites[View all \(0\)](#)

Favorite some packages, notebooks, and environments to get started!

! Activity Feed[View more](#)

Welcome to Anaconda Cloud! 1 day and 17 hours ago

Anaconda Cloud allows you to create or distribute software packages.

Getting started: [Installing your first package](#)

Getting started: [Distributing your first package](#)

Later in the book, we devote a whole chapter to this.

Finding help

There are many websites we can visit to get help. The first allows us to find the user guide, shown at the following link: <https://docs.anaconda.com/anaconda/user-guide/>. After we click the link, we will see four entries:

≡  > Anaconda Distribution > User guide

User guide

- Getting started with Anaconda
- Tasks
- Cheatsheet
- Troubleshooting

We can find lots of information by choosing one of the previous four entries. For example, after clicking Getting started with Anaconda, we will see the following information:

Getting started with Anaconda

Anaconda Distribution contains **conda** and **Anaconda Navigator**, as well as Python and hundreds of scientific **packages**. When you installed Anaconda, you installed all these too. You can try both conda and Navigator to see which is right for you to manage your packages and environments. You can even switch between them, and the work you do with one can be viewed in the other.

Now, try this simple programming exercise two ways, with Navigator and a terminal, to help you decide which approach is right for you.

Your first Python program: Hello, Anaconda!

Write and run a Python program using Anaconda Navigator.

1. Open Navigator

Choose the instructions for your operating system.

- Windows
- macOS
- Linux

From the Navigator bar in the left-hand side, we could go to Tasks, and then we have the following entries:

Tasks

- [Installing licenses](#)
- [Installing conda packages](#)
- [Using Jupyter Notebook extensions](#)
- [Using R language with Anaconda](#)
- [Switching between Python 2 and Python 3 environments](#)
- [Moving Anaconda from one directory to another](#)
- [Integrations](#)
- [Using default repositories](#)

For example, after clicking on [Installing conda packages](#), we can see the following information (only the first part of the document is shown to save space):

Installing conda packages

For more information about using the conda package manager in Anaconda Prompt (Terminal on Linux or macOS), see the [conda documentation](#).

You can also use the graphical interface [Anaconda Navigator](#) to install conda packages with just a few clicks.

Open an Anaconda Prompt (Terminal on Linux or macOS) and follow these instructions.

Installing a conda package

Enter the command:

```
conda install package-name
```

NOTE: Replace `package-name` with your package name.

Installing specific versions of conda packages

Include the desired version number or its prefix after the package name:

```
conda install package-name=2.3.4
```

For developers, we have the following links:

- Getting started: <https://conda.io/docs/user-guide/getting-started.html>
- Online documents: <http://conda.io/docs>
- Developer blog: <https://www.anaconda.com/developer-blog/>
- Anaconda cheat sheet: <https://docs.anaconda.com/anaconda/user-guide/cheatsheet>

Summary

In this chapter, we have introduced some basic concepts, such as the reasons why we use Anaconda, and the advantages of using full-fledged Anaconda and/or its baby version, Miniconda. Then, it was shown that without installing Anaconda, we could still use it by visiting a designated website. We could also test a few simple programs there, writing in R, Python, Julia, or Octave.

In [Chapter 2, Anaconda Installation](#), we will show you how to install Anaconda and test if the installation is successful. We will look at how to launch Jupyter, how to launch Python, Spyder, and R, and how to find related help. Most of those concepts or procedures are quite basic, so readers who are confident with those basic concepts can skip this chapter, [Chapter 2, Anaconda Installation](#), and go to [Chapter 3, Data Basics](#), directly.

Review questions and exercises

1. What is Anaconda and how do we use its platform?
2. How many open source packages are accompanied with Anaconda?
3. What is the home page for Anaconda?
4. How do we install Anaconda? After Anaconda is installed, should we install Python separately? What about R?
5. What is the size of a full Anaconda installation?
6. Why should we care about Miniconda?
7. What is Jupyter? How do we launch it without installation?
8. What are the advantages and disadvantages of using <https://jupyter.org/try>?
9. Where could a new learner find more useful information about Anaconda?
10. Get more information about the Julia programming language.

11. How do we write a simple program in Julia via Jupyter?
12. How do we write a simple program in R via Jupyter?
13. How do we find help for Jupyter?
14. What is the conda Cheat Sheet and where can we download it?
15. Could we run a simple R program without installing Anaconda?
16. Could we run Anaconda without pre-installing it?
17. Try the following two lines of Python code:

```
| import numpy as np  
| print(np.sqrt(2))
```

18. Try the following simple code for R:

```
| x<-1:500  
| mean(x)  
| sd(x)
```

19. Try the following code for Julia:

```
| x=1:500  
| mean(x)
```

20. Try the following code for R:

```
| dd<-Sys.Date()  
| dd+40
```


Anaconda Installation

In this chapter, we will discuss how to install Anaconda and use its various components, such as Python, IPython, Jupyter, and Spyder. In this book, we will not teach R, Python, Julia, or Octave. Thus, we assume that readers have some basic knowledge related to those programming languages, especially Python and R. There are many books available, such as *Python for Finance* by Yan (2017, 2nd edition) and *Financial Modeling Using R* by Yan (2016).

In this chapter, the following topics will be covered:

- Installing Anaconda
- Testing Python
- Using IPython
- Using Python via Jupyter
- Introducing Spyder
- Installing R via Conda
- Installing Julia and linking it to Jupyter
- Installing Octave and linking it to Jupyter
- Finding help

Installing Anaconda

First, go to <http://anaconda.org/>. We can see Download Anaconda the top-right part of the page. Based on the individual operating system, users should download the appropriate package.

Depending on which version you prefer, you could download Python 3.6 or Python 2.7. Generally speaking, for new learners it is a good idea to download the latest version (Python 3.6 in this case). This applies for both Mac and Linux-based operating systems.

Anaconda for Windows

Refer to the following steps:

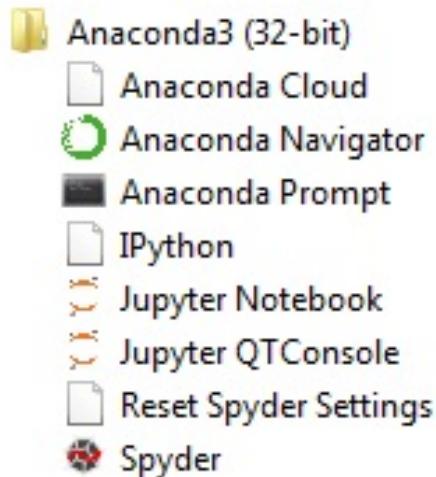
1. For PC users, after clicking on Windows, you will be presented with different versions to download.

As of May 09, 2018, two versions of Python accompanying Anaconda, version 3.6 and version 2.7, are available. If there are no particular requirements or restrictions, it is a good idea to download the latest version, version 3.6. In addition, there are two different downloads for 64-bit and 32-bit systems.



If your system is 32-bit and you try to install a 64-bit version, you would get an error message.

2. During the installation process, we can choose the default setting. After this is completed, we can see the following information (for a Windows version) after clicking All Programs and `Anaconda3 (32-bit)`. However, different readers/users might have slightly different settings:



3. After clicking Anaconda Prompt, we could type a command of `conda info`, shown here:

```
[C:\Users\yany\AppData\Local\Continuum\anaconda3] C:\Users\yany>conda info
Current conda install:

      platform : win-32
      conda version : 4.3.30
      conda is private : False
      conda-env version : 4.3.30
      conda-build version : 3.0.27
      python version : 3.6.3.final.0
      requests version : 2.18.4
      root environment : C:\Users\yany\AppData\Local\Continuum\anaconda3 (writable)
                        default environment : C:\Users\yany\AppData\Local\Continuum\anaconda3
                        envs directories : C:\Users\yany\AppData\Local\Continuum\anaconda3\envs
                                         C:\Users\yany\AppData\Local\conda\conda\envs
                                         C:\Users\yany\.conda\envs
      package cache : C:\Users\yany\AppData\Local\Continuum\anaconda3\pkgs
                                         C:\Users\yany\AppData\Local\conda\conda\pkgs
      channel URLs : https://repo.continuum.io/pkgs/main/win-32
                     https://repo.continuum.io/pkgs/main/noarch
                     https://repo.continuum.io/pkgs/free/win-32
                     https://repo.continuum.io/pkgs/free/noarch
                     https://repo.continuum.io/pkgs/r/win-32
                     https://repo.continuum.io/pkgs/r/noarch
                     https://repo.continuum.io/pkgs/pro/win-32
                     https://repo.continuum.io/pkgs/pro/noarch
                     https://repo.continuum.io/pkgs/msys2/win-32
                     https://repo.continuum.io/pkgs/msys2/noarch
      config file : None
      netrc file : None
      offline mode : False
      user-agent : conda/4.3.30 requests/2.18.4 CPython/3.6.3 Windows/7 Windows/6.1.7601
      administrator : False
```

Again, note that readers/users would have quite different results, depending on their system and installed packages.

Testing Python

The steps for testing Python are as follows:

1. For PC users, after clicking All Programs, Anaconda3.6, and then Anaconda Prompt, we would see the following.

Note that different users would probably get a different path:

```
(C:\Users\yany\AppData\Local\Continuum\anaconda3) C:\Users\yany>
```

2. Then, just type `python`, and we can launch it, as shown here:

```
(C:\Users\yany\AppData\Local\Continuum\anaconda3) C:\Users\yany>
(C:\Users\yany\AppData\Local\Continuum\anaconda3) C:\Users\yany>python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 15 2017, 07:29:16) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

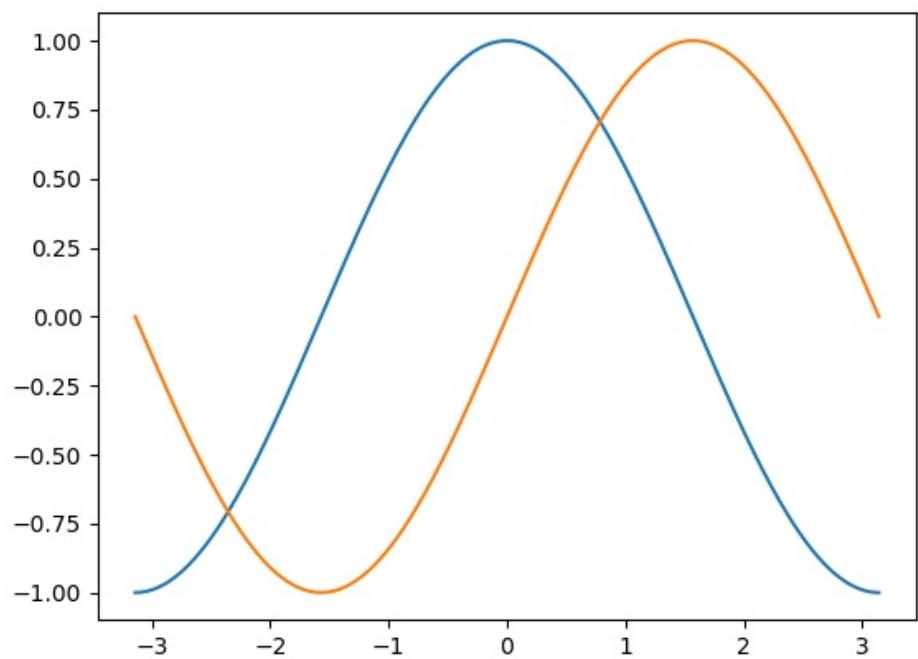
3. It tells us that `Python 3.6.3` was operational. We could also try `import scipy as sp` to see if it is preinstalled:

```
>>> import scipy as sp
>>> sp.sqrt(3)
1.7320508075688772
>>>
```

4. After we type `import scipy as sp`, no error message appears, which indicates that the package was preinstalled. The command of `sp.sqrt(3)` would offer us the square root of 3. Another example with the related graph is shown here:

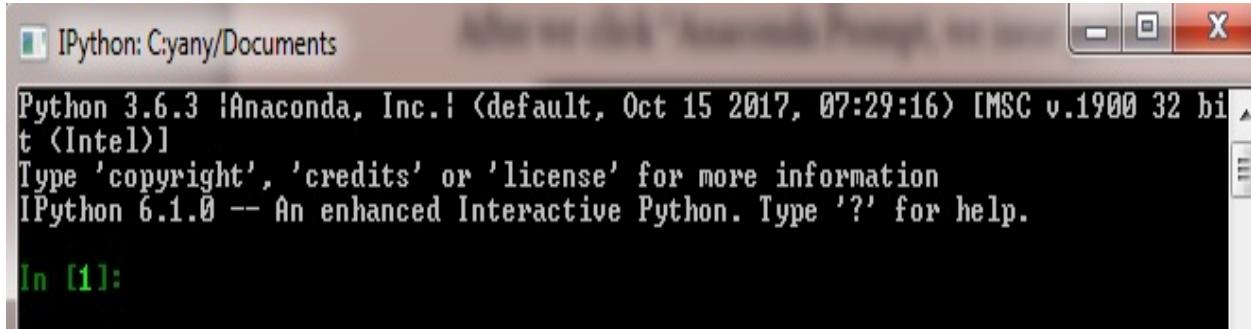
```
import scipy as np
from pylab import *
x=np.linspace(-np.pi,np.pi,256,endpoint=True)
c,s=np.cos(x),np.sin(x)
plot(x,c),plot(x,s)
show()
```

The previous code will give an output such as the following:



Using IPython

For a Windows version, after navigating to Anaconda3 from All Programs and then clicking on IPython, we can see the following:



The screenshot shows a terminal window titled "IPython: C:\any\Documents". The window displays the following text:
Python 3.6.3 |Anaconda, Inc.| (default, Oct 15 2017, 07:29:16) [MSC v.1900 32 bit (Intel)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.
In [1]:



You can also do the same by launching Anaconda Prompt and then typing `ipython`.

This tells us that the version of `Python` is `3.6.3` and that the version of `IPython` is `6.1.0`. We could type a few commands to test it. For example, if we invest \$100 today for an investment horizon of 5 years, and the annual return is 10%, what is our expected future value? The future value formula for a given present value is provided here:

$$FV = PV(1 + R)^n \quad (1)$$

where FV is the future value, PV is the present value, R is the period return, and n is the number of periods:

```
In [1]: pv=100
In [2]: pv
Out[2]: 100
In [3]: r=0.1
In [4]: n=5
In [5]: pv*(1+r)**n
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-5-e52348b6e68d> in <module>()
      1 pv*(1+r)**n
TypeError: unsupported operand type(s) for ^: 'float' and 'int'
In [6]: pv*(1+r)**n
Out[6]: 161.05100000000004
```

The first line of command of `pv=100` assigns a value of `100` to `pv`. For the fifth line, we typed a command line of `pv*(1+r)**n`. However, we received an error message of `TypeError: unsupported operand type(s) for ^: 'float' and 'int'`. The error message tells us that Python does not recognize the operand of `^` for a power. The corresponding operation for a power is double multiplication, as in `**`. There is also a `power()` function similar to `**`, shown here:

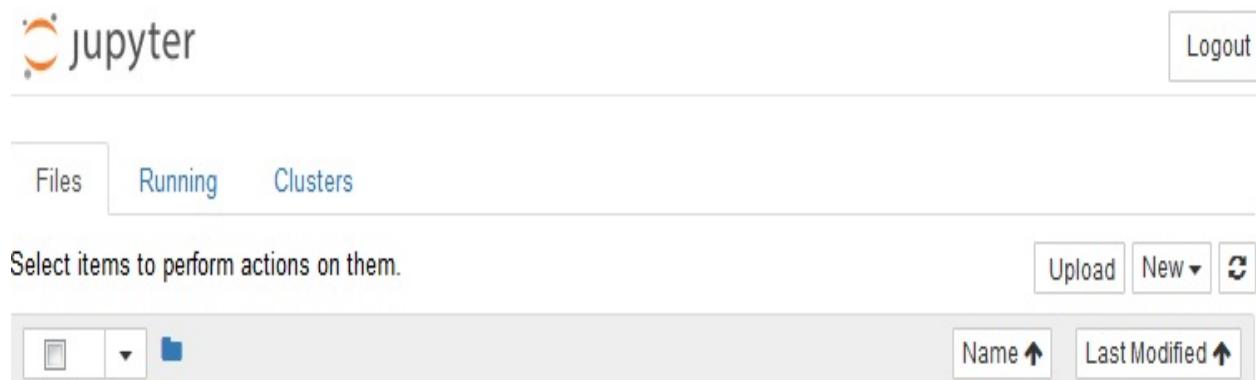
```
[IPython: C:\any\Documents] IPython 3.6.3 |Anaconda, Inc.| (default, Oct 15 2017, 07:29:16) [MSC v.1900 32 bit (Intel)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import scipy as sp
In [2]: sp.power(2,3)
Out[2]: 8
In [3]: 2**3
Out[3]: 8
In [4]:
```

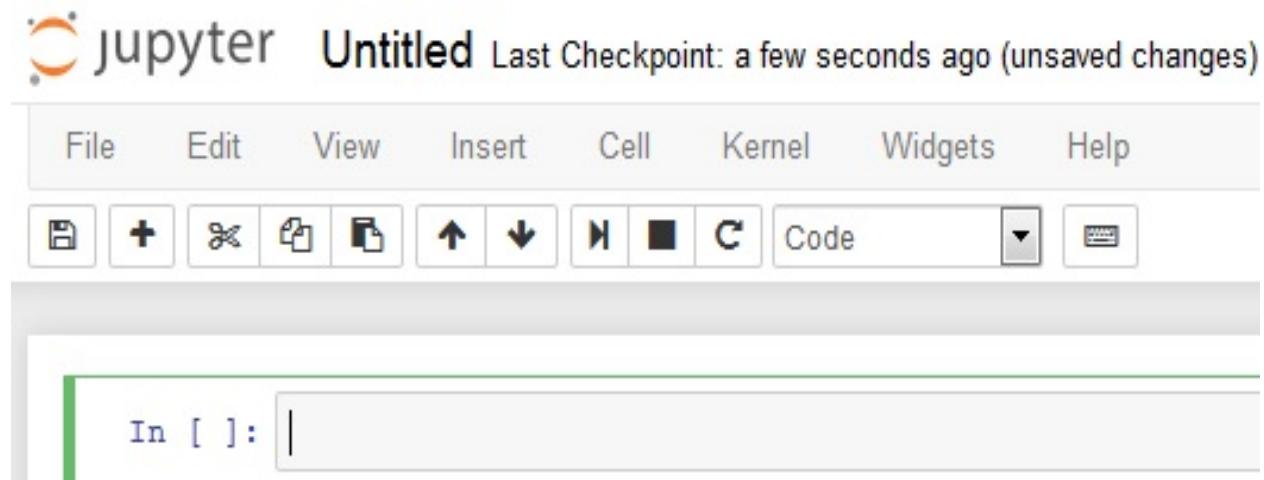
Using Python via Jupyter

Here, we show you how to write simple Python code via Jupyter:

1. From the menu, after clicking Anaconda, we can find an entry called Jupyter Notebook. After clicking on that entry, we can see the following:



2. If we click New on the right-hand side, we can find several choices. After we choose Python 3, we end up with the following starting file:



3. We can now type our Python commands in the box. Remember to press *Shift + Enter* if you want to execute the command (shown here):

```
In [1]: pv=10
```

```
In [2]: pv
```

```
Out[2]: 10
```

4. We can also type multiple commands and execute them, as shown here:

```
In [4]: import scipy as sp  
sp.sqrt(3)
```

```
Out[4]: 1.7320508075688772
```

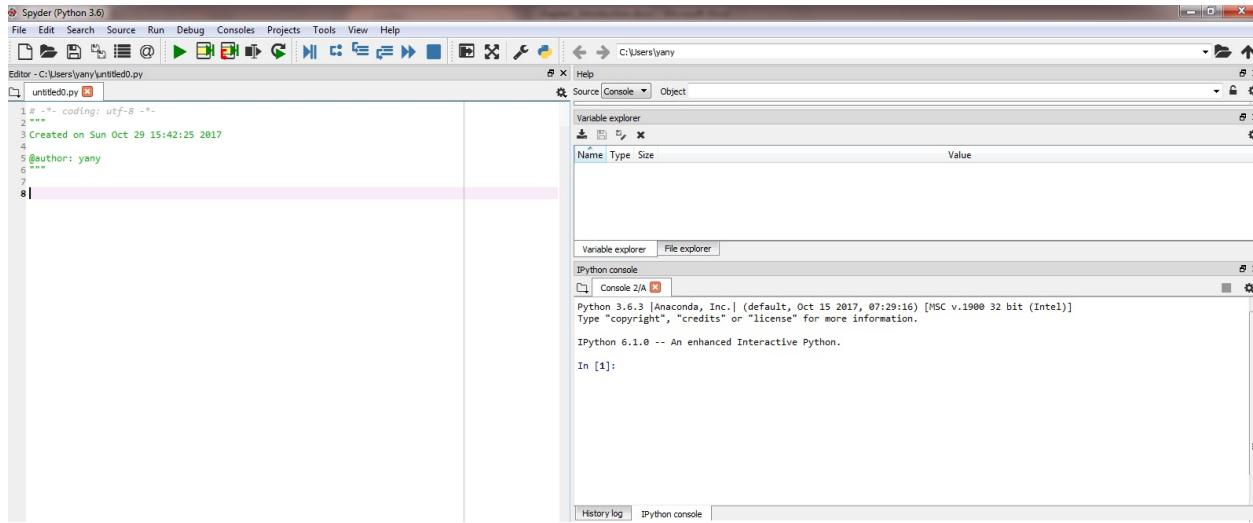
The colorful and distinct treatment of keywords, parentheses, and values makes our programming a little easier.



We could save our programs by choosing File|Save and Checkpoint on the menu bar. Similarly, we could load our presaved programs by choosing File|Revert to Checkpoint directly from the menu bar or finding it under Files in the Jupyter home page.

Introducing Spyder

In the Anaconda3 menu, the last entry is Spyder. After clicking it, we can launch Spyder, shown here:



The preceding screenshot shows three panels. The left panel is for writing and editing our programs, the bottom-right panel is for command lines (we could type simple commands there), and the top-right panel is for our variables. For example, after we type `pv=100`, it would show the variable name, type, size, and value, as shown here:

The screenshot shows the Jupyter Notebook interface. At the top is the 'Variable explorer' panel, which contains a table with one row:

Name	Type	Size	Value
pv	int	1	100

Below the Variable explorer is the 'File explorer' tab. The main area is the 'IPython console', which displays the following text:

```
Python 3.6.3 |Anaconda, Inc.| (default, Oct 15 2017, 07:29:16) [MSC v.1900 32 bit (Intel)]
Type "copyright", "credits" or "license" for more information.

IPython 6.1.0 -- An enhanced Interactive Python.

In [1]: pv=100
In [2]:
```

We could also write our Python program, and debug and run them via the top-left panel. For example, we could run a program with `pv_f()` function to estimate the present value of one future cash flow, shown here:

The screenshot shows the Spyder Python IDE interface. The title bar reads "Spyder (Python 3.6)". The menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. Below the menu is a toolbar with various icons for file operations like Open, Save, and Print, as well as run and debug tools. The main area is a code editor titled "Editor - C:\yan\01_DataScience_Anaconda\codes\c1_01_pv_function.py". The file contains the following Python code:

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Oct 29 15:42:25 2017
4
5 @author: yany
6 """
7 def pv_f(pv,r,n):
8     return pv/(1+r)**n
9 #
10 pv=pv_f(100,0.1,2)
11 print(pv)
```

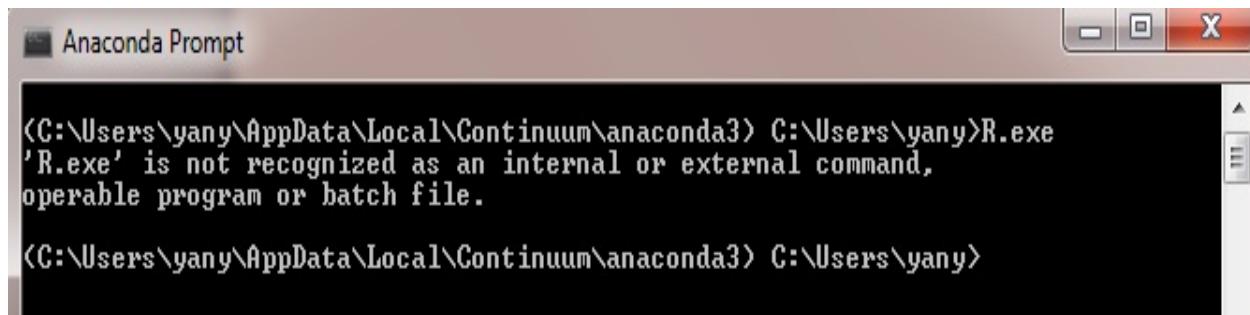
In the preceding screenshot, the green play button is for running the whole Python program, while the second green-yellow one is for partially running it. This feature makes our debugging efforts a little easier, which is especially true for a long and complex program.

Assume that we plan to debug such a program. First, we divide it into a few logic blocks. Then, we highlight the first block and run it. After it is bug free, we go to the next block. We continue this way until we debug the whole program.

Installing R via Conda

The following steps are for R package installation:

1. Note that the R package is not preinstalled when installing Anaconda, as shown here:



A screenshot of the Anaconda Prompt window. The title bar says "Anaconda Prompt". The main area shows the following command and its output:
(C:\Users\yany\AppData\Local\Continuum\anaconda3) C:\Users\yany>R.exe
'R.exe' is not recognized as an internal or external command,
operable program or batch file.
(C:\Users\yany\AppData\Local\Continuum\anaconda3) C:\Users\yany>

2. To install R software, we have to issue the following command after we click Anaconda Prompt:

```
| conda install r-essentials
```

3. After installation, we can see the following:



4. We could try a few commands. For example, we first generate an R function called `pv_f()` with the keyword function. For this function, we have three input values: `fv` for future value, `r` for interest rate, and `n` for the

number of periods:

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** jupyter Untitled4 Last Checkpoint: a minute ago (unsaved changes)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, R, O
- Toolbar Icons:** Includes icons for file operations like Open, Save, Copy, Paste, and Cell.
- In [1]:** `pv_f<-function(fv,r,n) fv/(1+r)^n`
- In [2]:** `pv_f(100,0.1,1)`
- Output:** 90.9090909090909

Installing Julia and linking it to Jupyter

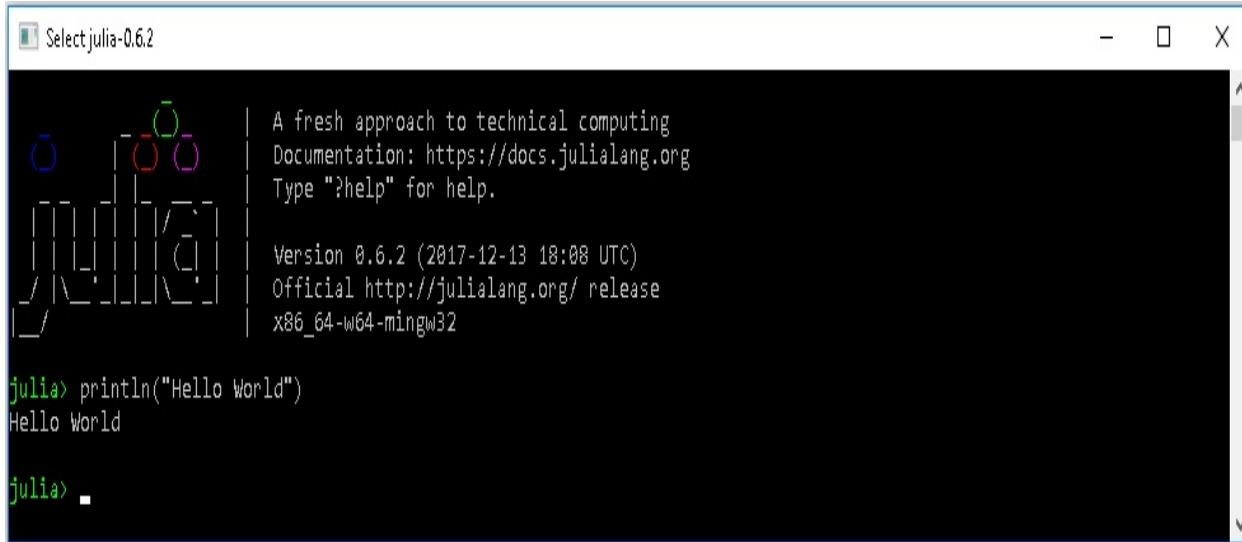
The steps for installing Julia are as follows:

1. First, we go to the following web page to download Julia: <https://julialang.org/downloads/>.
2. As when downloading Anaconda, we have different versions depending on our system, as shown here:

Julia (command line version)

Windows Self-Extracting Archive (.exe) [help]	32-bit	64-bit	
macOS Package (.dmg) [help]	10.8+ 64-bit		
Generic Linux Binaries for x86 [help]	32-bit (GPG)	64-bit (GPG)	
Generic Linux Binaries for ARM	32-bit (armv7-a hard float) (GPG)	64-bit (armv8-a) (GPG)	
Source	Tarball (GPG)	Tarball with dependencies (GPG)	GitHub

3. Again, based on your operating system (Windows, Mac, or Linux), choose the appropriate version and download the executable package.
4. Clicking the executable, we can install the package and, in order to launch Julia, we simply click on `julia.exe`.
5. In it, we can type `println("Hello World")`, as shown:



A screenshot of a Windows command prompt window titled "Selectjulia-0.6.2". The window shows the Julia 0.6.2 welcome screen with the text: "A fresh approach to technical computing", "Documentation: https://docs.julialang.org", "Type "?help" for help.", "Version 0.6.2 (2017-12-13 18:08 UTC)", "Official http://julialang.org/ release", and "x86_64-w64-mingw32". Below this, a Julia session starts with "julia> println("Hello World")" followed by "Hello World".

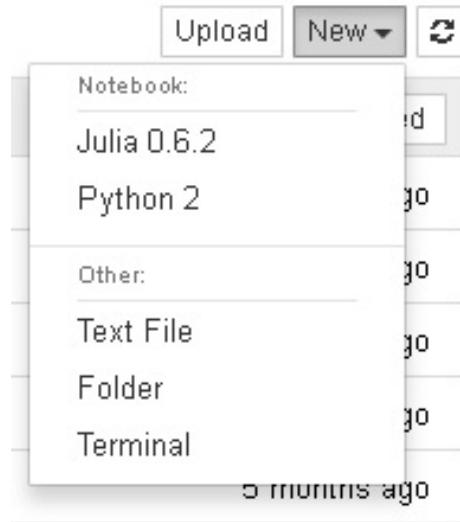
Alternatively, for Windows, we can click All Programs | Julia. We can also copy and paste Julia to our desktop.

6. In addition, another way to launch Julia is through a DOS window. However, before we can do so, we have to add the path to the directory that contains `julia.exe` to our path. For example, in order to use Julia via Jupyter, we have to issue the following command:



```
julia> Pkg.add("IJulia")
INFO: Initializing package repository C:\Users\yany\.julia\v0.6
INFO: Cloning METADATA from https://github.com/JuliaLang/METADATA.jl
INFO: Cloning cache of BinDeps from https://github.com/JuliaLang/BinDeps.jl.git
INFO: Cloning cache of BufferedStreams from https://github.com/BioJulia/BufferedStreams.jl.git
```

7. It takes about 10 minutes to install the package called `IJulia`. After that, we launch Jupyter, then click New, and we can see the following:



8. Then we can launch Julia and type our command line of `println("This is great!")` there, as shown:

```
In [1]: println("This is great!")

This is great!
```

In terms of Julia language, when discussing optimization in [chapter 7, Optimization in Anaconda](#), we used one or two optimization packages in Julia to solve a few problems.

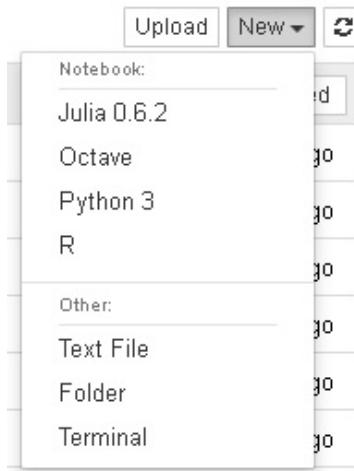
Installing Octave and linking it to Jupyter

Follow these steps to install Octave:

1. To install the Octave kernel linked to Anaconda, at Anaconda Prompt, we log in by issuing the Anaconda login command. Note that in order to log in, users have to register first at <https://anaconda.org>. After logging in to Anaconda, we could issue the following command:

```
| pip install octave_kernel
```

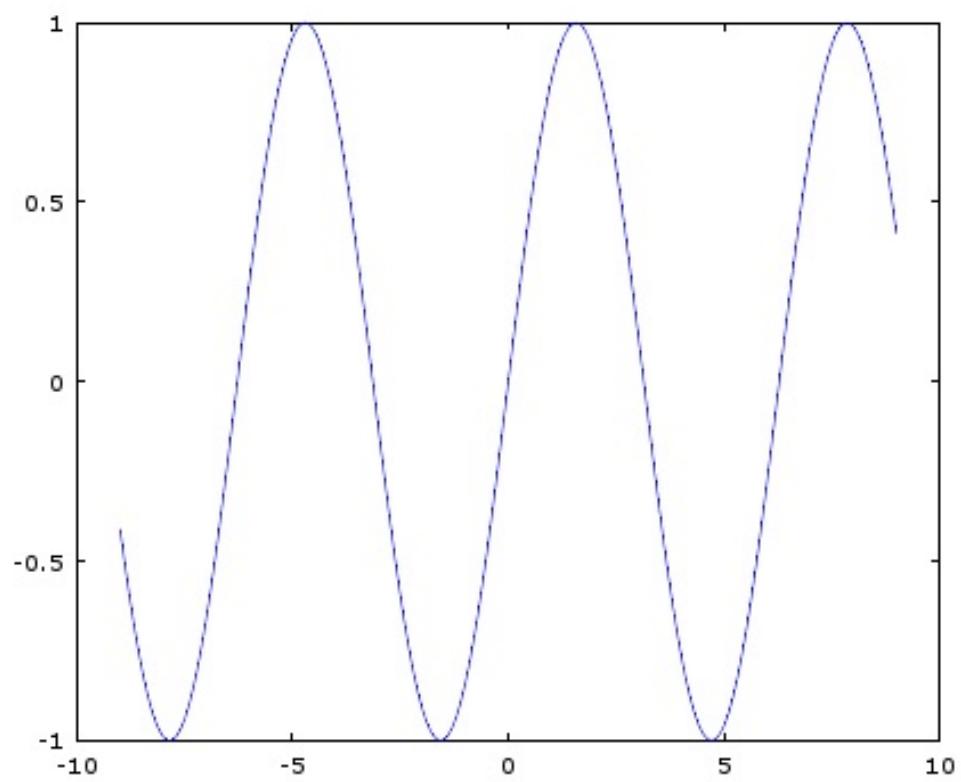
2. After we launch Jupyter and click New on the right-hand side, we can see Octave, among other languages:



3. Now we can test a simple program in Octave:

```
x=-9:0.1:9;  
  
y=sin(x);  
  
plot(x,y)
```

4. The related output is shown here:



Finding help

1. In [Chapter 1](#), *Ecosystem of Anaconda*, we showed that we could go to the link <https://docs.anaconda.com/anaconda/user-guide/>.
2. After clicking the link, we would see four entries, shown here:

≡  > Anaconda Distribution > User guide

User guide

- [Getting started](#)
- [Tasks](#)
- [Cheatsheet](#)
- [Troubleshooting](#)

3. We could type `conda help` to find information about the usages of Conda. In Windows, click All Programs | Anaconda | Anaconda Prompt. In the Prompt, type `conda help`, as shown here:

Anaconda Prompt

```
(C:\Users\yany\AppData\Local\Continuum\anaconda3) C:\Users\yany>conda help
usage: conda [-h] [-V] command ...

conda is a tool for managing and deploying applications, environments and packages.

Options:

positional arguments:
  command
    info          Display information about current conda install.
    help          Displays a list of available conda commands and their help
                  strings.
    list          List linked packages in a conda environment.
    search        Search for packages and display their information. The input
                  is a Python regular expression. To perform a search with a
                  search string that starts with a -, separate the search from
                  the options with --, like 'conda search -- -h'. A * in the
                  results means that package is installed in the current
                  environment. A . means that package is not installed but is
                  cached in the pkgs directory.
    create        Create a new conda environment from a list of specified
                  packages.
    install       Installs a list of packages into a specified conda
                  environment.
    update        Updates conda packages to the latest compatible version. This
                  command accepts a list of package names and updates them to
                  the latest versions that are compatible with all other
                  packages in the environment. Conda attempts to install the
                  newest versions of the requested packages. To accomplish
                  this, it may update some packages that are already installed,
                  or install additional packages. To prevent existing packages
                  from updating, use the --no-update-deps option. This may
                  force conda to install older versions of the requested
                  packages, and it does not prevent additional dependency
                  packages from being installed. If you wish to skip dependency
                  checking altogether, use the '--force' option. This may
                  result in an environment with incompatible packages, so this
                  option must be used with great caution.
    upgrade       Alias for conda update. See conda update --help.
    remove        Remove a list of packages from a specified conda environment.
    uninstall    Alias for conda remove. See conda remove --help.
    config        Modify configuration values in .condarc. This is modeled
                  after the git config command. Writes to the user .condarc
                  file (C:\Users\yany\.condarc) by default.
    clean         Remove unused packages and caches.
    package       Low-level conda package utility. <EXPERIMENTAL>

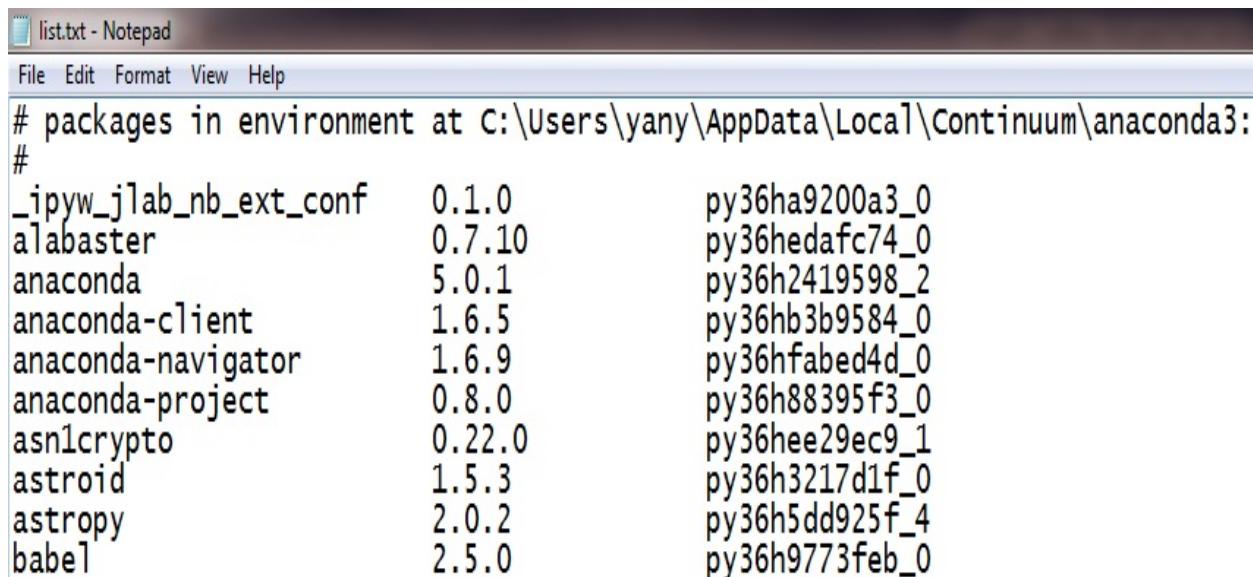
optional arguments:
  -h, --help      Show this help message and exit.
  -V, --version   Show the conda version number and exit.

other commands, such as "conda build", are available when additional conda
packages (e.g. conda-build) are installed
```

4. To find all packages associated with a Conda environment, we could issue the `conda list` command.
5. Since the number of packages is quite large, a better solution is to generate a text file. We could issue `conda list >c:/temp/list.txt`:

```
(C:\Users\yany\AppData\Local\Continuum\anaconda3) C:\Users\yany>
(C:\Users\yany\AppData\Local\Continuum\anaconda3) C:\Users\yany>conda list >c:/temp/list.txt
```

6. The first several lines from the output file called `list.txt` are shown:



```
list.txt - Notepad
File Edit Format View Help
# packages in environment at C:\Users\yany\AppData\Local\Continuum\anaconda3:
#
_ipyw_jlab_nb_ext_conf    0.1.0          py36ha9200a3_0
alabaster                  0.7.10         py36hedafc74_0
anaconda                   5.0.1          py36h2419598_2
anaconda-client              1.6.5          py36hb3b9584_0
anaconda-navigator           1.6.9          py36hfabed4d_0
anaconda-project              0.8.0          py36h88395f3_0
asn1crypto                  0.22.0         py36hee29ec9_1
astroid                     1.5.3          py36h3217d1f_0
astropy                     2.0.2          py36h5dd925f_4
babel                       2.5.0          py36h9773feb_0
```

7. We could write an R program to read this text file:

```
> x<-read.csv('c:/temp/list.txt',skip=2) > head(x)

      X_ipyw_jlab_nb_ext_conf....0.1.0.....py36ha9200a3_0

  1 alabaster          0.7.10         py36hedafc74_0

  2 anaconda           5.0.1          py36h2419598_2

  3 anaconda-client     1.6.5          py36hb3b9584_0

  4 anaconda-navigator  1.6.9          py36hfabed4d_0
```

```
5 anaconda-project      0.8.0      py36h88395f3_0
6 asn1crypto            0.22.0     py36hee29ec9_1
> dim(x)
[1] 380    1
```

8. To learn how to use Jupyter, we can click Help on the menu bar after we launch Jupyter.
9. From the list of entries, we can find the menu about Python and IPython and information about the Python packages called NumPy, SciPy, Matplotlib, and so on.
10. After clicking on Keyboard Shortcuts, we would see the following:

Help

User Interface Tour
Keyboard Shortcuts
Edit Keyboard Shortcuts

Notebook Help

Markdown

Python
IPython
NumPy
SciPy
Matplotlib
SymPy
pandas

About

Keyboard shortcuts

The Jupyter Notebook has two different keyboard input modes. Edit mode allows you to type code/text into a cell and is indicated by a green cell border. Command mode binds the keyboard to notebook level commands and is indicated by a grey cell border with a blue left margin.

Command Mode (press `Esc` to enable)

[Edit Shortcuts](#)

`F`: find and replace

`Ctrl-Shift-F`: open the command palette

`Ctrl-Shift-P`: open the command palette

`Enter`: enter edit mode

`P`: open the command palette

`Shift-Enter`: run cell, select below

`Ctrl-Enter`: run selected cells

`Alt-Enter`: run cell, insert below

`Y`: to code

`M`: to markdown

`R`: to raw

`1`: to heading 1

`2`: to heading 2

`3`: to heading 3

`4`: to heading 4

`5`: to heading 5

`6`: to heading 6

`X`: select cell above

`Shift-J`: extend selected cells below

`A`: insert cell above

`B`: insert cell below

`X`: cut selected cells

`C`: copy selected cells

`Shift-V`: paste cells above

`V`: paste cells below

`Z`: undo cell deletion

`D`, `D`: delete selected cells

`Shift-M`: merge selected cells, or current cell with cell below if only one cell selected

`Ctrl-S`: Save and Checkpoint

`S`: Save and Checkpoint

`L`: toggle line numbers

`O`: toggle output of selected cells

`Shift-O`: toggle output scrolling of selected cells

[Close](#)

To access the free support group, go to the link at <https://groups.google.com/a/anaconda.com/forum/#!forum/anaconda>.

Summary

In this chapter, we showed you how to install Anaconda and test if the installation was successful. We looked at how to launch Jupyter and how to use it to launch Python, how to launch Spyder and R, and how to find help. Most of these concepts or procedures are quite basic, so users who are quite confident with them can skip this chapter and go to [Chapter 3, Data Basics](#), directly.

In [Chapter 3, Data Basics](#), we first discuss open data sources such as the UCI (University of California at Irvin) machine learning depository and the bureau of labor statistics. Then, we introduce the Python Pandas package. Many issues, such as how to deal with missing data, sorting, how to slice and dice datasets, merging different datasets, and data input and output, will be discussed in detail. Several relevant packages for data manipulation will be also introduced and discussed.

Review questions and exercises

1. What is Anaconda? How do we use its platform?
2. What is the home page for Anaconda?
3. How do we install Anaconda?
4. After Anaconda is installed, should we install Python separately? What about R?
5. What is Jupyter? How do we launch it?
6. How do we launch Python from Jupyter?
7. How do we use Python via Jupyter?
8. What is Spyder and how do we launch it?
9. Write a Python program to estimate the present value of the annuity, as shown in the following formula:

$$pv(\text{annuity}) = \frac{C}{R} \left[1 - \frac{1}{(1 + R)^n} \right]$$

$pv(\text{annuity})$ is the present value of the annuity, C is the periodic payment, R is the discount rate, and n is the number of periods.

10. Find out how many packages are available after you have installed Anaconda. (Hint: check the link at <https://docs.anaconda.com/anaconda/packages/pkg-docs>.)
11. What are the advantages and disadvantages of using <https://jupyter.org/try>?
12. Where can new users find more information about Anaconda?
13. What is the Julia programming language?
14. How do we install Julia and how do we call Julia from Jupyter?
15. How do we install Octave and link it to Jupyter?
16. How do we find help for Jupyter?
17. Where can we find lots of open data?
18. For Python and R, calculate the value of 2^3 (2 raised to a power of 3).
19. How do we launch R independently?
20. How do we use Spyder to load a Python program and run it?

Data Basics

In this chapter, we'll first discuss sources of open data, which includes the **University of California at Irvine (UCI)** Machine Learning Depository, the Bureau of Labor Statistics, the Census Bureau, Professor French's Data Library, and the Federal Reserve's Data Library. Then, we will show you several ways of inputting data, how to deal with missing values, sorting, choosing a subset, merging different datasets, and data output. For different languages, such as Python, R, and Julia, several relevant packages for data manipulation will be introduced as well. In particular, the Python `pandas` package will be discussed.

In this chapter, the following topics will be covered:

- Sources of data
- Introduction to the Python `pandas` package
- Several ways to inputting packages
- Introduction to the Quandl data delivery platform
- Dealing with missing data
- Sorting data, as well as how to slice, dice, and merge various datasets
- Introduction to Python packages: `cbsodata` and `datadotword`
- Introduction to R packages: `dslabs`, `haven`, and `foreign`
- Generating Python datasets
- Generating R datasets

Sources of data

For users in the area of data science and business analytics, one important issue is the source of data, or simply where to get data. When working at a company, the obvious source of data is one's own company, such as sales, cost of raw materials, the salary of managers and other employees, the related information of suppliers and clients, estimations of future sales, the cost of raw materials, and so on. It is a good idea to find some data for learning purposes, and this is especially true for full-time students.

Generally speaking, there are two types of data: **public** and **private**. Private or proprietary databases are quite expensive. A typical example is the **Center for Research in Security Prices (CRSP)** database, a financial database generated and maintained by the University of Chicago. This database has daily, weekly, monthly, and annual trading data for all stocks listed on stock exchanges in the US from 1926 onward.

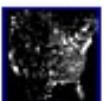
The second type of data is public or free data. For users in various data science or business analytics programs, this type of data is more than enough. For example, the UCI offers many useful datasets for machine learning that can be used for testing and learning purposes. This offers great benefits to new learners in the area of data science. Later in the chapter, several lists of free data will be offered for learners in data science, economics, and finance and accounting.

UCI machine learning

The UCI maintains 413 datasets, as of 1/10/2018, for machine learning: <http://archive.ics.uci.edu/ml/index.php>. The following screenshot shows the top three downloaded datasets:

Most Popular Data Sets (hits since 2007):

1631325:  [Iris](#)

1052490:  [Adult](#)

801975:  [Wine](#)

For the number one downloaded dataset called **Iris**, we have the following information:

Iris Data Set

Download: [Data Folder](#), [Data Set Description](#)

Abstract: Famous database; from Fisher, 1936



Data Set Characteristics:	Multivariate	Number of Instances:	150	Area:	Life
Attribute Characteristics:	Real	Number of Attributes:	4	Date Donated	1988-07-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	1631334

Source:

Creator:

R.A. Fisher

Donor:

Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)

The beauty of these datasets is that they give quite detailed information such as the source, the creator or donator, a description, and even citations.

The following table shows several potential public data sources for users in the area of data science and business analytics:

Name	Web page	Data types
UCI	http://archive.ics.uci.edu/ml/index.php	Data for machine learning

World Health Organization	http://www.who.int/en/	Healthcare data
Amazon Web Services	https://aws.amazon.com/cn/assets/?nc1=h_ls	Web usage
Data.gov (US Government Open Data)	https://www.data.gov/	Agriculture, climate, consumer, and so on
GitHub	https://github.com/caesar0301/awesome-public-datasets	Many datasets offered by individuals
Open data network	https://www.opendatanetwork.com/	Many useful datasets
Government health data	https://www.healthdata.gov/	Healthcare-related datasets
Google public data	https://www.google.com/public-data/directory	World development indicators

Table 3.1: Potential sources of open data for data science and business analytics

After we go to <https://www.data.gov/>, we can see the following choices related to **Agriculture, Climate, Consumer, Ecosystems, Education**, and the like:

Health Care Provider Charge Data



BROWSE TOPICS



[Agriculture](#)



Climate



Consumer



Ecosystems



Education



Energy



Finance



Health



Local
Government



Manufacturing



Maritime



Ocean



Public Safety



Science &
Research

The next table shows the potential sources of open data for users in the area of economics:

Name	Web page	Data types
Federal reserve economic data	https://www.federalreserve.gov/datadownload/Choose.aspx?rel=H15	Interest rates and rates for AAA/AA rated bonds

Census bureau	http://www.census.gov/ https://www.census.gov/about/index.html	Census data
US. dept. treasure	http://www.treas.gov	US treasure yield
Bureau of labor statistics	http://www.bls.gov/ http://download.bls.gov/	Inflation, employment, unemployment, pay and benefits, and so on
Bureau of economic analysis	http://www.bea.gov/	GDP, personal income outlays, corporate profit, and so on
National bureau of economic research	http://www.nber.org/	Business cycles, vital statistics, reports of presidents, and so on

Table 3.2: Potential sources of open data for economics

After going to the Federal Reserve economic data and clicking Data on the menu, we can see the following entries:

Monetary Policy	Supervision & Regulation	Payment Systems	Economic Research	Data	Consumers & Communities
Bank Assets and Liabilities	Bank Structure Data		Exchange Rates and International Data		Household Finance
Aggregate Reserves of Depository Institutions and the Monetary Base - H.3	Large Commercial Banks		Foreign Exchange Rates - H.10/G.5		Consumer Credit - G.19
Assets and Liabilities of Commercial Banks in the U.S. - H.8	Minority-Owned Depository Institutions		International Summary Statistics		Household Debt Service and Financial Obligations Ratios
Assets and Liabilities of U.S. Branches and Agencies of Foreign Banks	Structure and Share Data for the U.S. Offices of Foreign Banks		Securities Holdings and Transactions		Mortgage Debt Outstanding Survey of Consumer Finances (SCF)
Charge-Off and Delinquency Rates on Loans and Leases at Commercial Banks	Business Finance		Statistics Reported by Banks and Other Financial Firms in the United States		Industrial Activity
Senior Loan Officer Opinion Survey on Bank Lending Practices	Commercial Paper		New Security Issues, State and Local Governments		Industrial Production and Capacity Utilization - G.17
Survey of Terms of Business Lending - E.2	Finance Companies - G.20		New Security Issues, U.S. Corporations		Interest Rates
	New Security Issues, State and Local Governments		Structure and Share Data for U.S. Offices of Foreign Banks		Selected Interest Rates - H.15
	New Security Issues, U.S. Corporations		Financial Accounts		Money Stock and Reserve Balances
	Dealer Financing Terms		Financial Accounts of the United States - Z.1		Factors Affecting Reserve Balances - H.4.1
	Senior Credit Officer Opinion Survey on Dealer Financing Terms				Money Stock Measures - H.6

The following table offers free data for users in the areas of finance and accounting:

Name	Web page	Data types
Yahoo!Finance	http://finance.yahoo.com	Historical pricing data, options, balance sheets, and income statements

Google Finance	http://www.google.com/finance	Current and historical trading prices
Marketwatch	http://www.marketwatch.com	Financial statements
SEC filing	http://www.sec.gov/edgar.shtml	Balance sheets, income statements, holdings, and so on
Professor French's data library	http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html	Fama/French factors, market index, risk-free rate, industry classification, and so on

Table 3.3: Potential sources of open data for finance and accounting

From Professor French's data library, we could download the famous Fama/French's three-factor time series:

```
> infile<- "http://canisius.edu/~yany/data/ff3monthly.csv"
> x<-read.csv(infile,skip=3)
> head(x,2)
  Date Mkt.RF  SMB   HML   RF
1 192607    2.96 -2.3 -2.87 0.22
2 192608    2.64 -1.4  4.19 0.25
> tail(x,2)
  Date Mkt.RF  SMB   HML   RF
1095 201709    2.51  4.53  3.02 0.09
1096 201710    2.25 -1.94 -0.09 0.09
```

In the previous code, the input file called `ff3monthly.csv` is a modified copy of `F-F_Research_Data_Factor.csv` made by removing the second part of the annual data and adding a `Date` as the header. Note that `F-F_Research_Data_Factor.csv` is from the ZIP file called `F-F_Research_Data_Factor.zip`.

Introduction to the Python pandas package

The Python pandas package is very useful when dealing with data. The pandas package is a wonderful tool for data preprocessing, which is essential for data analysis. There is a humorous way of describing the importance of data cleaning: "*A data scientist spends 80% of their time cleaning the data and the other 20% complaining about cleaning the data*". To test if the package is preinstalled, we can type `import pandas as pd` after we launch Python. If we don't see any error messages, it means that the package was preinstalled. If we do, then we can use `conda install pandas` to install the package. To find all available functions, we could use the following three lines of Python code:

```
In [1]: import pandas as pd
```

```
In [2]: x=dir(pd)
```

```
In [3]: print(x)
```

```
['Categorical', 'CategoricalIndex', 'DataFrame', 'DateOffset', 'DatetimeIndex', 'ExcelFile', 'ExcelWriter', 'Expr',
'Float64Index', 'Grouper', 'HDFStore', 'Index', 'IndexSlice', 'Int64Index', 'MultiIndex', 'NaT', 'Panel', 'Panel4D',
'Period', 'PeriodIndex', 'RangeIndex', 'Series', 'SparseArray', 'SparseDataFrame', 'SparseList', 'SparseSeries',
'SparseTimeSeries', 'Term', 'TimeGrouper', 'TimeSeries', 'Timedelta', 'TimedeltaIndex', 'Timestamp', 'WidePanel',
'__builtins__', '__doc__', '__docformat__', '__file__', '__name__', '__package__', '__path__', '__version__', '__join__',
'__np_version_under1p10', '__np_version_under1p11', '__np_version_under1p12', '__np_version_under1p8', '__np_version_under1p9',
'_period', '_sparse', '_testing', '_version', '_window', 'algos', 'api', 'bdate_range', 'compat', 'computation', 'concat',
'core', 'crosstab', 'cut', 'date_range', 'datetime', 'datetools', 'describe_option', 'eval', 'ewma', 'ewmcorr', 'ewmcov',
'ewmstd', 'ewmvar', 'ewmvol', 'expanding_apply', 'expanding_corr', 'expanding_count', 'expanding_cov', 'expanding_kurt',
'expanding_max', 'expanding_mean', 'expanding_median', 'expanding_min', 'expanding_quantile', 'expanding_skew',
'expanding_std', 'expanding_sum', 'expanding_var', 'factorize', 'fama_macbeth', 'formats', 'get_dummies', 'get_option',
'get_store', 'groupby', 'hashtable', 'index', 'indexes', 'infer_freq', 'info', 'io', 'isnull', 'json', 'lib', 'lreshape',
'match', 'melt', 'merge', 'merge_asof', 'merge_ordered', 'msgpack', 'notnull', 'np', 'offsets', 'ols', 'option_context',
'options', 'ordered_merge', 'pandas', 'parser', 'period_range', 'pivot', 'pivot_table', 'plot_params', 'pnow', 'qcut',
'read_clipboard', 'read_csv', 'read_excel', 'read_fwf', 'read_gbq', 'read_hdf', 'read_html', 'read_json', 'read_msgpack',
'read_pickle', 'read_sas', 'read_sql', 'read_sql_query', 'read_sql_table', 'read_stata', 'read_table', 'reset_option',
'rolling_apply', 'rolling_corr', 'rolling_count', 'rolling_cov', 'rolling_kurt', 'rolling_max', 'rolling_mean',
'rolling_median', 'rolling_min', 'rolling_quantile', 'rolling_skew', 'rolling_std', 'rolling_sum', 'rolling_var',
'rolling_window', 'scatter_matrix', 'set_eng_float_format', 'set_option', 'show_versions', 'sparse', 'stats', 'test',
'timedelta_range', 'to_datetime', 'to_msgpack', 'to_numeric', 'to_pickle', 'to_timedelta', 'tools', 'tseries', 'tslib',
'types', 'unique', 'util', 'value_counts', 'wide_to_long']
```

To find out about the usage or examples of individual functions, the `help()` function can be used. For example, for the `to_pickle` functionality shown at the bottom of the previous screenshot, we type `help(pd.to_pickle)`, shown in the first line here:

```
In [78]: help(pd.to_pickle)
Help on function to_pickle in module pandas.io.pickle:

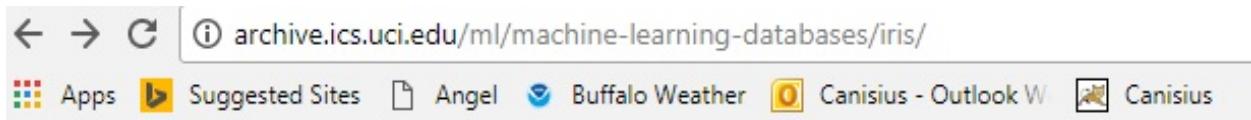
to_pickle(obj, path, compression='infer')
    Pickle (serialize) object to input file path

Parameters
-----
obj : any object
path : string
    File path
compression : {'infer', 'gzip', 'bz2', 'xz', None}, default 'infer'
    a string representing the compression to use in the output file

.. versionadded:: 0.20.0
```

Several ways to input data

First, let's look at how to input a **Comma Separated Value (CSV)** file. The input dataset is the most popular one from UCI Machine Learning Data Depository. The location is <http://archive.ics.uci.edu/ml/machine-learning-databases/iris/bezdekIris.data>; you can refer to the following screenshot as well:



Index of /ml/machine-learning-databases/iris

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
Parent Directory		-	
Index	03-Dec-1996 04:01	105	
bezdekIris.data	14-Dec-1999 12:12	4.4K	
iris.data	08-Mar-1993 16:27	4.4K	
iris.names	11-Jul-2000 21:30	2.9K	

```
> path<- "http://archive.ics.uci.edu/ml/machine-learning-databases/"

> dataSet<- "iris/bezdekIris.data"

> a<-paste(path,dataSet,sep="") > x<-read.csv(a,header=F)

> colnames(x)<-
c("sepalLength","sepalWidth","petalLength","petalWidth","Class")

> dim(x)

<strong> [1] 150 5</strong> > head(x,2)

<strong> sepalLength sepalWidth petalLength petalWidth Class 1 5.1 3.5 1.4 0.2
Iris-setosa

2 4.9 3.0 1.4 0.2 Iris-setosa</strong> > tail(x,2)

<strong> sepalLength sepalWidth petalLength petalWidth Class 149 6.2 3.4 5.4
2.3 Iris-virginica 150 5.9 3.0 5.1 1.8 Iris-virginica</strong>

> apropos("read.")

<strong> [1] ".read.xls" ".readRDS" "read.csv"
[4] "read.csv2" "read.dcf" "read.delim"
[7] "read.delim2" "read.DIF" "read.fortran"
[10] "read.ftable" "read.fwf" "read.lines"
[13] "read.links" "read.lynx" "read.socket"
[16] "read.table" "read.w3m" "read.xls"
[19] "readBin" "readChar" "readCitationFile"
[22] "readClipboard" "readline" "readLines"
[25] "readRDS" "readRegistry" "readRenviron"
```

```
[28] "readSeries" "spreads" "spreadSeries"
```

```
[31] "Sys.readlink" </strong>
```

To find out the exact usage of individual functions, we use the `help()` function. For example, after issuing `help(read.fwf)`, we can find information about this specific function, shown in the screenshot in the first several lines:

```
read.fwf {utils}
```

Read Fixed Width Format Files

Description

Read a table of fixed width formatted data into a [data.frame](#).

Usage

```
read.fwf(file, widths, header = FALSE, sep = "\t",
         skip = 0, row.names, col.names, n = -1,
         bufsize = 2000, fileEncoding = "", ...)
```

Inputting data using Python

Similarly, we can use Python to retrieve the data, as shown in the code here:

```
import pandas as pd path="http://archive.ics.uci.edu/ml/machine-learning-databases/" dataset="iris/bezdekIris.data" inFile=path+dataset  
data=pd.read_csv(inFile,header=None) data.columns=[  
    "sepalLength","sepalWidth","petalLength","petalWidth","Class"]
```

After retrieving data, the `print(data.head(2))` function can be used to see the first two instances: > `print(data.head(2))`

sepalLength sepalWidth petalLength petalWidth Class 0 5.1 3.5 1.4 0.2 Iris-setosa 1 4.9 3.0 1.4 0.2 Iris-setosa

When typing `pd.read.csv()`, we can find the definitions of all input variables, shown in the following screenshot. Again, to save space, only the first several input variables are shown:

```
1 import pandas as pd  
2 pd.read_csv()  
3  
4 Arguments  
read_csv(filepath_or_buffer, sep=',',  
        delimiter=None, header='infer',  
        names=None, index_col=None, usecols=None,  
        squeeze=False, prefix=None,  
        mangle_dupe_cols=True, dtype=None,  
        engine=None, converters=None,
```

To prevent a future potential change in terms of a dataset link, we have a backup dataset located at the author's website, shown in the following Python code:

```
inFile="http://canisius.edu/~yany/data/bezdekIris.data.txt" import pandas as pd  
d=pd.read_csv(inFile,header=None)
```

The following table shows several functions included in the pandas package that we could use to retrieve data:

Function	Description
----------	-------------

<code>read_clipboard</code>	Read text from clipboard and pass to <code>read_table</code>
<code>read_csv</code>	Read a CSV file into <code>DataFrame</code>
<code>read_excel</code>	Read an Excel table into a pandas <code>DataFrame</code>
<code>read_feather</code>	Load a feather-format object from the file path
<code>read_fwf</code>	Read a table of formatted lines with fixed-width into <code>DataFrame</code>
<code>read_gbq</code>	Load data from Google BigQuery
<code>read_hdf</code>	Retrieve a pandas object stored in a file
<code>read_html</code>	Read HTML tables into a list of <code>DataFrame</code> objects
<code>read_json</code>	Convert a JSON string into a pandas object
<code>read_msgpack</code>	Load a <code>msgpack</code> pandas object from the specified file path
<code>read_pickle</code>	Load a <code>pickled</code> pandas object (or any other <code>pickled</code> object) from the specified file path

read_sas	Read SAS files stored as either XPORT or SAS7BDAT format files
read_sql	Read an SQL query or database table into a DataFrame
read_sql_query	Read an SQL query into a DataFrame
read_sql_table	Read an SQL database table into a DataFrame
read_stata	Read a Stata file into a DataFrame
read_table	Read a general delimited file into a DataFrame

Table 3.4 Functions included in the Python pandas module for inputting data To find out detailed information on each of the preceding functions, we use the `help()` function. For example, if we want to get more information about the `read_sas()` function, we issue the following commands: `import pandas as pd help(pd.read_sas)`

The corresponding output, the top part only, is shown here:

```
In [4]: help(pd.read_sas)
Help on function read_sas in module pandas.io.sas.sasreader:

read_sas(filepath_or_buffer, format=None, index=None, encoding=None, chunksize=None, iterator=False)
    Read SAS files stored as either XPORT or SAS7BDAT format files.
```

Introduction to the Quandl data delivery platform

Quandl is a data delivery platform that includes many free datasets. Its website is <https://www.quandl.com>. The following are a few programs written in R or Python to download data from the platform. The following program retrieves the latest 50 trading days' data for **International Business Machine (IBM)**:

```
> library(Quandl)
> x<- Quandl.dataset.get("WIKI/ibm", list(rows=50))
> head(x,2)
  Date   Open   High   Low Close Volume Ex-Dividend
1 2018-01-09 163.90 164.530 163.060 163.83 4333418      0
2 2018-01-08 162.66 163.905 161.701 163.47 5101023      0
  Split Ratio Adj. Open Adj. High Adj. Low Adj. Close Adj. Volume
1          1    163.90 164.530 163.060    163.83    4333418
2          1    162.66 163.905 161.701    163.47    5101023
```

Note that we just need to issue the first two lines in the code once. The following code would get the data for a Chinese company with a ticker symbol of 601628. Obviously, DY4 is for Chinese data:

```
import Quandl as qd
y<- Quandl.dataset.get("DY4/601628", list(rows=50))
> head(y,2)
```

The corresponding output is shown here:

```
> head(y,2)
  Date Pre_Close Open High Low Close Adj_Pre_Close Adj_Open
1 2018-01-10     30.25 30.18 30.21 29.78 30.12           30.25    30.18
2 2018-01-09     29.65 29.70 30.30 29.55 30.25           29.65    29.70
  Adj_High Adj_Low Adj_Close Turnover_Volume Turnover_Value Deal_Amount
1     30.21   29.78     30.12       16383506     491497957      21220
2     30.30   29.55     30.25       27625443     830430487      33053
  Circulation_Market_Value Market_Value Turnover_Rate Adj_Factor Adj_Reason
1             627204723600 851332914600        0.0008        NA        NA
2             629911782500 855007326250        0.0013        NA        NA
```

For the following R program, we assume that the Quandl application key is contained in the `myQuandlKey` variable. Later in the chapter, we explain how to get a Quandl application key:

```

|> library(Quandl)
|> Quandl.api_key(myQuandlKey)
|> x<- Quandl.dataset.get("WIKI/wmt", list(rows=500))

```

For Python, we have the following code to download the **Gross Domestic Product (GDP)** from Quandl:

```

import quandl
x= quandl.get("FRED/GDP")
x.head()
      Value
Date
1947-01-01  243.080
1947-04-01  246.267
1947-07-01  250.115
1947-10-01  260.309
1948-01-01  266.173

```

The format of the output dataset is a pandas data frame. `quandl` can also return data as a `numpy` array, shown here:

```

import quandl
x=quandl.get("FRED/GDP", returns="pandas")
y=quandl.get("FRED/GDP", returns="numpy")

```

The following table shows patterns for different sources of data from the Quandl data platform:

Pattern	Example	Explanation
WIKI	WIKI/IBM	End-of-day US stock prices
DY4	DYE/60618	Chinese data
FRED	FRED/GDP	Federal Reserve economic data
EIA	EIA/PET_RWTC_D	US Department of Energy
NSE	NSE/OIL	Oil prices

ZEP	ZEP/ibm	Zacks equity prices, dividend history (US)
CBOE	CBOE/VXU2018	Chicago Board Options Exchange

Table 3.5: Patterns for different types/courses of data on the Quandl platform

As of 1/10/2017, in terms of the limitation set by Quandl, see the following table. For free subscribers, users need to register with Quandl to get a Quandl application key:

Type of user	Max calls (every 10 min.)	Max calls every day	Concurrent Limit
Anonymous	20	50	1 call
Free subscribers	2,000	50,000	1 + 1 in the queue
Premium	5,000	720,000	No limit

Table 3.6: Max number of calls every 10 min and every day for Quandl users

The next program shows how to retrieve Walmart data for free subscribers, as in, those with a Quandl application key. Being a free subscriber means that you register with Quandl without paying any fees:

```
import quandl as qd
quandl.ApiConfig.api_key = 'YOUR_API_KEY'
y=qd.get("WIKI/WMT")
y.head(2)
y.tail(1)
```

The output is shown here:

```
In [36]: y.head(2)
```

```
Out[36]:
```

Date	Open	High	Low	Close	Volume	Ex-Dividend	Split Ratio
1972-08-25	32.5	33.13	32.50	33.00	4900.0	0.0	1.0
1972-08-28	33.0	33.25	32.88	32.88	1900.0	0.0	1.0

Date	Adj. Open	Adj. High	Adj. Low	Adj. Close	Adj. Volume
1972-08-25	0.043162	0.043999	0.043162	0.043826	2508800.0
1972-08-28	0.043826	0.044158	0.043667	0.043667	972800.0

```
In [37]: y.tail(1)
```

```
Out[37]:
```

Date	Open	High	Low	Close	Volume	Ex-Dividend	\
2018-01-09	102.03	102.35	100.31	100.39	7296463.0	0.0	

Date	Split Ratio	Adj. Open	Adj. High	Adj. Low	Adj. Close	\
2018-01-09	1.0	102.03	102.35	100.31	100.39	

Date	Adj. Volume
2018-01-09	7296463.0

The second line of `quandl.ApiConfig.api_key = 'YOUR_API_KEY'` indicates that a user has to enter his/her Quandl application key.

Dealing with missing data

First, let's look at the missing codes for different languages:

Languages	Missing code	Explanation or examples
R	NA	NA stands for Not Available
Python	nan	import scipy as sp missingCode=sp.nan
Julia	missing	julia> missing + 5 missing
Octave	Nan	Same for MATLAB as well

Table 3.7: Missing codes for R, Python, Julia, and Octave

For R, the missing code is `NA`. Here are several functions we could use to remove those missing observations, shown in an example:

```
> head(na_example,20)
[1]  2  1  3  2  1  3  1  4  3  2  2 NA  2  2  1  4 NA  1  1  2
> length(na_example)
[1] 1000
> x<-na.exclude(na_example)
> length(x)
[1] 855
> head(x,20)
[1] 2 1 3 2 1 3 1 4 3 2 2 2 2 1 4 1 1 2 1 2
```

In the previous example, we removed 145 missing values by using the R function called `na.exclude()`. We could also use the `apropos()` function to find more

functions dealing with missing code in R, as shown here:

```
> apropos("na.")
 [1] "na.action"           "na.contiguous"
 [3] "na.exclude"          "na.fail"
 [5] "na.omit"             "na.pass"
 [7] "na_example"          "names"
 [9] "names.POSIXlt"       "names<-"
[11] "names<- POSIXlt"    "namespaceExport"
[13] "namespaceImport"     "namespaceImportClasses"
[15] "namespaceImportFrom" "namespaceImportMethods"
[17] "napredict"           "naprint"
[19] "naresid"              "nargs"
```

For Python, we have the following example, First, let's generate a dataset called `z.csv`, see the R code given next. For the program, we generate 100 zeros as our missing values:

```
set.seed(123)
n=500
x<-rnorm(n)
x2<-x
m=100
y<-as.integer(runif(m)*n)
x[y]<-0
z<-matrix(x,n/5,5)
outFile<-"c:/temp/z.csv"
write.table(z,file=outFile,quote=F, row.names=F, col.names=F, sep=', ')
```

The following Python program checks missing values for 5 columns, replace them with `NaN` or with the averages of each columns:

```
import scipy as sp
import pandas as pd
path="https://canisius.edu/~yany/data/"
dataSet="z.csv"
infile=path+dataset
#infile="c:/temp/z.csv"
x=pd.read_csv(infile,header=None)
print(x.head())
print((x[[1,2,3,4,5]] ==0).sum())
```

The related output is shown here:

```

          0           1           2           3           4           5           6   \
0 -0.560476  0.253319 -0.710407  0.787739  2.198810 -0.375603 -0.715242
1 -0.230177 -0.028547  0.256884  0.769042  1.312413 -0.561876 -0.752689
2  1.558708 -0.042870 -0.246692  0.000000 -0.265145  0.000000 -0.938539
3  0.070508  1.368602 -0.347543 -1.008377  0.543194  0.090497 -1.052513
4  0.129288 -0.225771 -0.951619 -0.119453 -0.414340  1.598509 -0.437160

          7           8           9
0  1.014943 -0.073556  1.430402
1 -1.992748 -1.168651  1.046629
2 -0.427279  0.000000  0.435289
3  0.000000 -0.028842  0.715178
4 -0.893208  0.000000  0.917175
1    10
1    10
2    9
3    8
4    12
5    12
dtype: int64

```

At this stage, we just know that for the first five columns, zero represents a missing value. The code of `print((x[[1,2,3,4,5]] == 0).sum())` shows the number of zeros for five columns. For instance, there are five zeros for the first column. We could use `scipy.NaN` to replace those zeros, as shown here:

```

x2=x
x2[[1,2,3,4,5]] = x2[[1,2,3,4,5]].replace(0,  sp.NaN)
print(x2.head())

```

The output with zeros is replaced with `sp.NaN`, as shown here:

```
In [14]: x2=x
....: x2[[1,2,3,4,5]] = x2[[1,2,3,4,5]].replace(0, np.NaN)
....: print(x2.head())
      0         1         2         3         4         5         6 \
0 -0.560476  0.253319 -0.710407  0.787739  2.198810 -0.375603 -0.715242
1 -0.230177 -0.028547  0.256884  0.769042  1.312413 -0.561876 -0.752689
2  1.558708 -0.042870 -0.246692      NaN -0.265145      NaN -0.938539
3  0.070508  1.368602 -0.347543 -1.008377  0.543194  0.090497 -1.052513
4  0.129288 -0.225771 -0.951619 -0.119453 -0.414340  1.598509 -0.437160

      7         8         9
0  1.014943 -0.073556  1.430402
1 -1.992748 -1.168651  1.046629
2 -0.427279  0.000000  0.435289
3  0.000000 -0.028842  0.715178
4 -0.893208  0.000000  0.917175
```

If we plan to use the mean to replace those `NaN`s, we have the following code:

```
|x3=x2
|x3.fillna(x3.mean(), inplace=True)
|print(x3.head())
```

The output is shown here:

```
In [15]: x3=x2
....: x3.fillna(x3.mean(), inplace=True)
....: print(x3.head())
      0         1         2         3         4         5         6 \
0 -0.560476  0.253319 -0.710407  0.787739  2.198810 -0.375603 -0.715242
1 -0.230177 -0.028547  0.256884  0.769042  1.312413 -0.561876 -0.752689
2  1.558708 -0.042870 -0.246692 -0.004695 -0.265145  0.274805 -0.938539
3  0.070508  1.368602 -0.347543 -1.008377  0.543194  0.090497 -1.052513
4  0.129288 -0.225771 -0.951619 -0.119453 -0.414340  1.598509 -0.437160

      7         8         9
0  1.014943 -0.073556  1.430402
1 -1.992748 -1.168651  1.046629
2 -0.427279  0.000000  0.435289
3  0.000000 -0.028842  0.715178
4 -0.893208  0.000000  0.917175
```

Data sorting

In R, we have several ways to sort data. The easiest way is to use the `sort()` function (see the code for the simplest one-dimensional data):

```
> set.seed(123)
> x<-rnorm(100)
> head(x) [1] -0.56047565 -0.23017749 1.55870831 0.07050839
  0.12928774 1.71506499
> y<-sort(x)
> head(y) [1] -2.309169 -1.966617
  -1.686693 -1.548753 -1.265396 -1.265061
```

Let's look at another way to sort data. The dataset used is called `nyseListing`, which is included in the R package called `fImport`, shown here:

```
library(fImport)
data(nyseListing)
dim(nyseListing)
head(nyseListing)
```

The output is shown here:

```
> dim(nyseListing)
[1] 3387      4
> head(nyseListing)
  Symbol                      Name MarketCap Exchange
1     A    Agilent Technologies, Inc. $12,852.3    NYSE
2    AA          Alcoa Inc.        $28,234.5    NYSE
3   AAI      AirTran Holdings, Inc.     $156.9    NYSE
4   AAP      Advance Auto Parts Inc. $3,507.4    NYSE
5   AAR          AMR CORPORATION      $81.7    NYSE
6   AAV ADVANTAGE ENERGY INCOME FUND $1,674.4    NYSE
```

In total, we have 3,387 observations, each with 4 variables. The dataset is sorted by `Symbol`, as in the tickers of individual stocks. Assume that we want to sort them by `Name`, as shown here:

```
> x<-nyseListing[order(nyseListing$Name),]
> head(x)
```

The output shows that the dataset is indeed sorted by company `Name`:

```

> x<-nyseListing[order(nyseListing$Name), ]
> head(x)
  Symbol          Name MarketCap Exchange
2017    MMM      3M Company $48,398.7   NYSE
557     CFD 40/86 Strategic Income Fund   $56.8   NYSE
1721    KDE 4Kids Entertainment, Inc.   $99.7   NYSE
2164    NDN 99 CENTS ONLY STORES   $432.3   NYSE
87      AHC A.H. Belo Corporation   $107.4   NYSE
1242    GFW AAG Holding Company Inc. <NA>   NYSE

```

In the following example, we sort by `ID` first, then by `RET`:

```

> x<-c(1,3,1, 0.1,0.3,-0.4,100,300,30)
> y<-data.frame(matrix(x,3,3))
> colnames(y)<-c("ID","RET","Data1")
> y

```

Our simple output dataset is shown here:

```

> y
  ID  RET Data1
1  1  0.1   100
2  3  0.3   300
3  1 -0.4    30

```

To sort the data according to `ID` and `RET`, we could use `order(ID,RET)`, shown here: > `z<-y[order(yID,yRET),]` > `z`

The following screenshot shows that the output dataset was sorted correctly:

```

> z
  ID  RET Data1
3  1 -0.4    30
1  1  0.1   100
2  3  0.3   300

```

If we want to sort according to decreasing order, we could add `decreasing=TRUE`:

```

> z2<-y[order(y$ID,decreasing = TRUE,y$RET),]
> z2
  ID  RET Data1
2  3  0.3   300
1  1  0.1   100
3  1 -0.4    30

```

To sort data in Python, see the following code:

```
import pandas as pd
a = pd.DataFrame([[8,3],[8,2],[1,-1]],columns=['X','Y'])
print(a)
# sort by A ascending, then B descending
b= a.sort_values(['X', 'Y'], ascending=[1, 0])
print(b)
# sort by A and B, both ascending
c= a.sort_values(['X', 'Y'], ascending=[1, 1])
print(c)
```

The output is shown here:

	X	Y
0	8	3
1	8	2
2	1	-1
	X	Y
2	1	-1
0	8	3
1	8	2
	X	Y
2	1	-1
1	8	2
0	8	3

```
> con<-url("http://canisius.edu/~yany/RData/marketCap.RData") > load(con)
```

```
> head(.marketCap)
```

```
> head(.marketCap)
```

 Symbol Name MarketCap Exchange 1 A Agilent Technologies, Inc.
\$12,852.3 NYSE

2 AA Alcoa Inc. \$28,234.5 NYSE

3 AA-P Alcoa Inc. \$43.6 AMEX
4 AAC
Ableauctions.Com Inc \$4.3 AMEX

5 AAI AirTran Holdings, Inc. \$156.9 NYSE

6 AAP Advance Auto Parts Inc \$3,507.4 NYSE

```
a<- .marketCap[1] # choose the 1st column b<- .marketCap$SYMBOL # another  
way to choose the 1st column c<- .marketCap[,1:2] # choose the first two  
columns d<- subset(.marketCap,.marketCap$EXCHANGE=="NYSE") e<-  
subset(head(.marketCap))
```

```
f<- subset(.marketCap,.marketCap$MARKET>200 &  
.marketCap$MARKETCAP<=3000)
```

A Python dataset is downloadable at
<http://canisius.edu/~yany/python/marketCap.pkl>.

```

> set.seed(123)

> nStocks<-4

> nPeriods<-24

> x<-runif(nStocks*nPeriods,min=-0.1,max=0.20) > a<-
matrix(x,nPeriods,nStocks) > d1<-as.Date("2000-01-01") > d2<-as.Date("2001-
12-01") > dd<-seq(d1,d2,"months")

> stocks<-data.frame(dd,a) > colnames(stocks)<-
c("DATE",paste('stock',1:nStocks,sep=""))

> dim(stocks)

<strong>[1] 24 5</strong> > head(stocks)

<strong> DATE stock1 stock2 stock3 stock4

1 2000-01-01 -0.01372674 0.09671174 -0.02020821 0.11305472

2 2000-02-01 0.13649154 0.11255914 0.15734831 -0.09981257

3 2000-03-01 0.02269308 0.06321981 -0.08625065 0.04259497

4 2000-04-01 0.16490522 0.07824261 0.03266002 -0.03396433

5 2000-05-01 0.18214019 -0.01325208 0.13967745 0.01394496

6 2000-06-01 -0.08633305 -0.05586591 -0.06343022 0.08383130 </strong>

> d3<-as.Date("1999-01-01") > d4<-as.Date("2010-12-01") > dd2<-
seq(d3,d4,"months") > y<-runif(length(dd2),min=-0.05,max=0.1) > market<-
data.frame(dd2,y) > colnames(market)<-c("DATE","MKT")

> dim(market)

<strong>[1] 144 2</strong> > head(market,2)

<strong> DATE MKT

```

1 1999-01-01 0.047184022

2 1999-02-01 -0.002026907

> final<-merge(stocks,market) > dim(final)

[1] 24 6 > head(final,2)

 DATE stock1 stock2 stock3 stock4 MKT

1 2000-01-01 -0.01372674 0.09671174 -0.02020821 0.11305472 0.05094986

2 2000-02-01 0.13649154 0.11255914 0.15734831 -0.09981257 0.06056166


```
import pandas as pd
```

```
import scipy as sp

x= pd.DataFrame({'YEAR': [2010,2011, 2012, 2013], 'FirmA': [0.2, -0.3, 0.13, -0.2], 'FirmB': [0.1, 0, 0.05, 0.23]}) y = pd.DataFrame({'YEAR': [2011,2013,2014, 2015], 'FirmC': [0.12, 0.23, 0.11, -0.1], 'SP500': [0.1,0.17, -0.05, 0.13]})

print("n inner merge ")

print(pd.merge(x,y, on='YEAR'))

print(" n outer merge ")

print(pd.merge(x,y, on='YEAR',how='outer')) print("n left merge ")

print(pd.merge(x,y, on='YEAR',how='left')) print("n right merge ")

print(pd.merge(x,y, on='YEAR',how='right'))
```

The related output is shown here:

```
inner merge
FirmA FirmB YEAR FirmC SP500
0 -0.3 0.00 2011 0.12 0.10
1 -0.2 0.23 2013 0.23 0.17

outer merge
FirmA FirmB YEAR FirmC SP500
0 0.20 0.10 2010 NaN NaN
1 -0.30 0.00 2011 0.12 0.10
2 0.13 0.05 2012 NaN NaN
3 -0.20 0.23 2013 0.23 0.17
4 NaN NaN 2014 0.11 -0.05
5 NaN NaN 2015 -0.10 0.13

left merge
FirmA FirmB YEAR FirmC SP500
0 0.20 0.10 2010 NaN NaN
1 -0.30 0.00 2011 0.12 0.10
2 0.13 0.05 2012 NaN NaN
3 -0.20 0.23 2013 0.23 0.17

right merge
FirmA FirmB YEAR FirmC SP500
0 -0.3 0.00 2011 0.12 0.10
1 -0.2 0.23 2013 0.23 0.17
2 NaN NaN 2014 0.11 -0.05
3 NaN NaN 2015 -0.10 0.13
```

```
> write.csv(final,file="c:/temp/t.csv") >
write.table(final,file="c:/temp/t2.txt",sep=";")

> apropos("^write")

<strong> [1] "write" "write.csv" "write.csv2"
[4] "write.dcf" "write.ftable" "write.socket"
[7] "write.table" "writeBin" "writeChar"
[10] "writeClipboard" "writeLines" </strong>
```

```
import pandas as pd

infile="http://canisius.edu/~yany/data/ff3monthly.csv"
ff3=pd.read_csv(infile,skiprows=3) print(ff3.head(2))

# output to pickle

ff3.to_pickle("c:/temp/ff3.pkl") # output to a csv file

outfile=open("c:/temp/ff3.csv","w") ff3.to_csv(outfile,index=None)
outfile.close()

# output to text file

outfile2=open("c:/temp/ff3.txt","w") ff3.to_csv(outfile2, header=True,
index=None, sep=' ', mode='a') outfile2.close()
```

Introduction to the cbsodata Python package

To install the `cbsodata` Python package, perform the following steps:

1. We can use one of the following commands:

```
conda install cbsodata  
  
pip install cbsodata
```

For more detailed instructions about how to install the Python package, please see [Chapter 6, Managing Packages](#):

2. The next program shows one example of using the package:

```
import pandas as pd  
  
import cbsodata as cb  
  
name='82070ENG'  
  
data = pd.DataFrame(cb.get_data(name))  
  
print(data.head())  
  
info=cb.get_info(name)  
  
print(info['Title'])
```

3. The corresponding output is shown in the following screenshot:

```
Retrieving data from table '82070ENG'
```

```
Done!
```

```
CaribbeanNetherlands EmployedLabourForceInternatDef_1 \
0           Bonaire                8837.0
1      St. Eustatius              2099.0
2           Saba                1045.0
3           Bonaire              6344.0
4      St. Eustatius              1443.0
```

```
EmployedLabourForceNationalDef_2          Gender ID Periods
0                      8559.0 Total male and female 0 2012
1                      1940.0 Total male and female 1 2012
2                      971.0 Total male and female 2 2012
3                      6241.0 Total male and female 3 2012
4                      1362.0 Total male and female 4 2012
```

```
PersonalCharacteristics
0 Total personal characteristics
1 Total personal characteristics
2 Total personal characteristics
3 Employment status: permanent contract
4 Employment status: permanent contract
Caribbean Netherlands; employed labour force characteristics 2012
```

The last line in the screenshot gives the name of the dataset. In the previous example, we used the dataset with the name 82070ENG.

4. To find out all the names of lists, we use the `get_table_list()` function; see the following code:

```
import cbsodata as cb

list=cb.get_table_list()

print(list)
```

Introduction to the datadotworld Python package

To install the `datadotworld` Python package, follow these steps:

1. First, we have to install the package. To do so, we could try one of the following lines:

```
| conda install datadotworld  
|  
| pip install datadotworld
```

2. After the package is successfully installed, we can use the `dir()` function to list all its functions, as shown in this screenshot:

In [188]: `import datadotworld as dw`

In [189]: `x=dir(dw)`

In [190]: `print(x)`
['ChainedConfig', 'DataDotWorld', 'EnvConfig', 'FileConfig', 'InlineConfig', 'UriParam', '__builtins__', '__cached__', '__doc__', '__file__', '__instances__', '__loader__', '__name__', '__package__', '__path__', '__spec__', '__version__', '__get_instance', 'absolute_import', 'api_client', 'client', 'config', 'datadotworld', 'files', 'load_dataset', 'models', 'open_remote_file', 'query', 'util', 'weakref']

3. A user has to get an API token at <https://data.world/settings/advanced> in order to run a Python program. Without such a token, we might get the following error message if we run a `datadotworld` function:

`RuntimeError: Configuration file not found at C:\Users\yany/.dw/config. To fix this issue, run dw configure`

4. According to the error message, we must run the following `configure` command:

```
(C:\Users\yany\AppData\Local\Continuum\anaconda3) C:\Users\yany>dw configure  
API token (obtained at: https://data.world/settings/advanced): c:/Users/yany/.dw  
/
```

5. Now we can use the Python package as shown in the following Python program:

```
import datadotworld as dw

name='jonloyens/an-intro-to-dataworld-dataset'

results = dw.query(name,

    'SELECT * FROM `DataDotWorldBBallStats`, `DataDotWorldBBallTeam` '

    'WHERE DataDotWorldBBallTeam.Name = DataDotWorldBBallStats.Name')

df = results.dataframe

print(df)
```

6. The related output is shown here:

```
DataDotWorldBBallStats.name  pointspergame  assistspergame \
0                      Jon          20.4        1.3
1                      Rob          15.5        8.0
2                     Sharon         30.1       11.2
3                     Alex          8.2        0.5
4                    Rebecca        12.3       17.0
5                     Ariane        18.1        3.0
6                     Bryon         16.0        8.5
7                     Matt          13.0        2.1
```

```
DataDotWorldBBallTeam.name  height handedness
0                      Jon      6'5"    Right
1                      Rob     6'7.5"   Left
2                     Sharon    6'3"    Right
3                     Alex     6'2"    Right
4                    Rebecca    7'     Right
5                     Ariane    5'8"    Left
6                     Bryon     7'     Right |
7                     Matt      5'5"    Right
```

```
library(haven)<br/>x<-1:100<br/>y<-matrix(x,50,2)<br/>z<-data.frame(y)
<br/>colnames(z)<-c("a","b")<br/>write_sas(z,"c:/temp/tt.sas7bdat")
<br/>write_spss(z,"c:/temp/tt.sav")<br/>write_stata(z,"c:/temp/tt.dta")
```

```
library(foreign)
```

```
x<-read.spss("c:/temp/airline_passengers.sav", to.data.frame=TRUE)
```

```
dim(x)
```

```
<strong>[1] 144 1</strong>
```

Introduction to the dslabs R package

The `dslabs` R package is short for **Data Science Labs**. The package includes several datasets, such as the dataset called `murders` for US gun murders by state for 2010:

```
> library(dslabs) > data(murders) > head(murders)
```

state	abb	region	population	total	1
Alabama	AL	South	4779736	135	2
Alaska	AK	West	710231	19	3
Arizona	AZ	West	6392017	232	4
Arkansas	AR	South	2915918	93	5
California	CA	West	37253956	1257	6
Colorado	CO	West	5029196	65	

The following table shows the datasets included in the package:

Name of dataset	Description
<code>admissions</code>	Gender bias among graduate school admissions to UC Berkeley
<code>divorce_margarine</code>	Divorce rate and margarine consumption data
<code>ds_theme_set</code>	dslabs theme set
<code>gapminder</code>	Gapminder data
<code>heights</code>	Self-Reported heights
<code>murders</code>	US gun murders by state for 2010
<code>na_example</code>	Count data with some missing values

<code>oecd</code>	Gapminder data
<code>opec</code>	Gapminder data
<code>outlier_example</code>	Adult male heights in feet with outliers
<code>polls_us_election_2016</code>	Fivethirtyeight 2016 poll data
<code>raw_data_research_funding_rates</code>	Gender bias in research funding in the Netherlands
<code>reported_heights</code>	Self-reported Heights
<code>research_funding_rates</code>	Gender bias in research funding in the Netherlands
<code>results_us_election_2016</code>	Fivethirtyeight 2016 poll data
<code>rfalling_object</code>	Simulated falling object data
<code>take_poll</code>	Models results from taking a poll
<code>us_contagious_diseases</code>	Contagious disease data for US states

Table 3.8 Datasets included in the R package called dslabs

Generating Python datasets

To generate a Python dataset, we use the `Pandas to_pickle` functionality. The dataset we plan to use is called `adult.pkl`, as shown in the following screenshot:

The screenshot shows a web browser window with the URL `archive.ics.uci.edu/ml/machine-learning-databases/adult/` in the address bar. The page title is "Index of /ml/machine-learning-databases/adult". The page content lists several files and a parent directory:

Name	Last modified	Size	Description
Parent Directory		-	
Index	03-Dec-1996 04:06	140	
adult.data	10-Aug-1996 11:14	3.8M	
adult.names	31-Jan-2001 08:53	5.1K	
adult.test	10-Aug-1996 11:14	1.9M	
old.adult.names	10-Aug-1996 11:14	4.2K	

Index of /ml/machine-learning-databases/adult

Name	Last modified	Size	Description
Parent Directory		-	
Index	03-Dec-1996 04:06	140	
adult.data	10-Aug-1996 11:14	3.8M	
adult.names	31-Jan-2001 08:53	5.1K	
adult.test	10-Aug-1996 11:14	1.9M	
old.adult.names	10-Aug-1996 11:14	4.2K	

The related Python code is given here:

```
import pandas as pd
path="http://archive.ics.uci.edu/ml/machine-learning-databases/"
dataSet="adult/adult.data"
inFile=path+dataSet
x=pd.read_csv(inFile,header=None)
adult=pd.DataFrame(x,index=None)
adult=adult.rename(columns={0:'age',1: 'workclass',
2:'fnlwgt',3:'education',4:'education-num',
5:'marital-status',6:'occupation',7:'relationship',
8:'race',9:'sex',10:'capital-gain',11:'capital-loss',
12:'hours-per-week',13:'native-country',14:'class'})
adult.to_pickle("c:/temp/adult.pkl")
```

To show the first several lines of observations, we use the `x.head()` functionality, shown in the following screenshot:

```
In [66]: adult.head()
Out[66]:
   age      workclass fnlwgt education education-num \
0   39      State-gov  77516  Bachelors          13
1   50  Self-emp-not-inc  83311  Bachelors          13
2   38        Private 215646    HS-grad           9
3   53        Private 234721      11th            7
4   28        Private 338409  Bachelors          13

   marital-status      occupation relationship    race    sex
0  Never-married  Adm-clerical  Not-in-family  White  Male
1 Married-civ-spouse  Exec-managerial     Husband  White  Male
2        Divorced  Handlers-cleaners  Not-in-family  White  Male
3 Married-civ-spouse  Handlers-cleaners     Husband  Black  Male
4 Married-civ-spouse       Prof-specialty       Wife  Black Female

   capital-gain  capital-loss hours-per-week native-country class
0      2174             0            40  United-States <=50K
1        0             0            13  United-States <=50K
2        0             0            40  United-States <=50K
3        0             0            40  United-States <=50K
4        0             0            40        Cuba <=50K
```

Note that the backup dataset is available at the author's website, downloadable at <http://canisius.edu/~yany/data/adult.data.txt>.

```

path<- "http://archive.ics.uci.edu/ml/machine-learning-databases/"
dataSet<- "iris/bezdekIris.data"
a<-paste(path,dataSet,sep="")
.iris<-read.csv(a,header=F)
colnames(.iris)<-
c("sepalLength","sepalWidth","petalLength","petalWidth","Class")
save(iris,file="c:/temp/iris.RData")

>load("c:/temp/iris.RData")
> head(.iris)

<strong> sepalLength sepalWidth petalLength petalWidth Class 1 5.1 3.5 1.4 0.2
Iris-setosa
2 4.9 3.0 1.4 0.2 Iris-setosa
3 4.7 3.2 1.3 0.2 Iris-setosa
4 4.6 3.1 1.5 0.2 Iris-setosa
5 5.0 3.6 1.4 0.2 Iris-setosa
6 5.4 3.9 1.7 0.4 Iris-setosa</strong>

inFile<- "http://canisius.edu/~yany/data/ff3monthly.csv" <br/>ff3monthly<-
read.csv(inFile,skip=3) saveRDS(ff3monthly,file="c:/temp/ff3monthly.rds")

>abc<-readRDS("c:/temp/ff3monthly.rds") >head(abc,3)

<strong> DATE MKT_RF SMB HML RF
1 1926-07-01 0.0296 -0.0230 -0.0287 0.0022
2 1926-08-01 0.0264 -0.0140 0.0419 0.0025

```

3 1926-09-01 0.0036 -0.0132 0.0001 0.0023 >head(ff3monthly,3)

 DATE MKT_RF SMB HML RF

1 1926-07-01 0.0296 -0.0230 -0.0287 0.0022

2 1926-08-01 0.0264 -0.0140 0.0419 0.0025

3 1926-09-01 0.0036 -0.0132 0.0001 0.0023

Summary

In this chapter, we first discussed sources of open data, which included The Bureau of Labor Statistics, the Census Bureau, Professor French's data library, the Federal Reserve's data library, and the UCI Machine Learning Depository. After that, we showed you how to input data; how to deal with missing data; how to sort, slice, and dice the datasets; and how to merge different datasets. Data output was discussed in detail. For different languages, such as Python, R, and Julia, several relevant packages for data manipulation were introduced and discussed.

In [Chapter 4](#), *Data Visualization*, we will discuss data visualization in R, Python, and Julia separately. To make our visual presentation more eye catching, we will show how you to generate simple graphs and bar charts, as well as how to add trend lines and legends. Other explanations will include how to save graphs to an external file, and we will discuss several packages related to data visualization in R, Python, and Julia.

Review questions and exercises

1. What is the difference between open data and proprietary databases?
2. Is it enough for learners in the area of data science to use open data?
3. Where can we access open public data?
4. From The UCI Data Depository, <http://archive.ics.uci.edu/ml/index.php>, download a dataset called `wine`. Write a program in R to import it.
5. From the UCI Data Depository, download a dataset called `Forest Fire`. Write a program in Python to import it.
6. From the UCI Data Depository, download a dataset called `Bank Marketing`. Write a program in Octave to import it. Answer the following questions: 1) How many banks? and 2) What is the cost?
7. How can we find all R functions with `read.` as their leading letters? (Note that there is a dot after `read.`)
8. How can we find more information on an R function called `read.xls()`?
9. Explain the differences between two R functions: `save()` and `saveRDS()`.
10. Find more information about the `read_clipboard()` function included in the Python pandas package.

11. What is the Quandl platform? What kinds of data could we download from Quandl?
12. Write both R and Python programs to download GDP (Gross Domestic Product) data from the Quandl platform.
13. When loading an R dataset, what is the difference between using the `load()` function and the `readRDS()` function?
14. After uploading the Python pandas package, explain why we have the following error message:

```
In [21]: import pandas as pd

In [22]: help(to_numeric)
Traceback (most recent call last):

  File "<ipython-input-22-a1c2c20cd180>", line 1, in <module>
    help(to_numeric)

NameError: name 'to_numeric' is not defined
```

15. First, download a ZIP file called `bank-fall.zip` at <http://archive.ics.uci.edu/ml/datasets/Bank+Marketing>. Unzip the file to get a CSV file; see the related code that follows:

```
> x<-read.table("c:/temp/bank-full.csv",sep=";",header=T)
> head(x,3)
  age      job marital education default balance housing loan contact day month
1 58 management married tertiary no 2143 yes no unknown 5 may
2 44 technician single secondary no 29 yes no unknown 5 may
3 33 entrepreneur married secondary no 2 yes yes unknown 5 may
  duration campaign pdays previous poutcome y
1     261        1     -1       0 unknown no
2     151        1     -1       0 unknown no
3      76        1     -1       0 unknown no
>
```

Generate an R dataset called `bank.Rata` and `bank.rds` and answer the following questions: a) What is the average age? b) What percentage of people are married? c) Is the default probability of those who are married higher than those who are single?

16. How do we merge two datasets in R?
17. Write a Python program to download IBM's daily data from Quandl and merge it with Fama-French three-factor. To get a Fama-French daily factor time series we could go to http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html or download a dataset at <http://canisius.edu/~yany/python/data/fDaily.pkl>.
18. Generate both R and Python datasets for monthly Fama-French-Charhart four factors. Both time series, can be downloaded from Professor French's

- data library.
19. Write a Python program to merge FRED/GDP data with market index data.

Data Visualization

It is said that *a picture is worth a thousand words*. Through various pictures and graphical presentations, we can express many abstract concepts, theories, data patterns, or certain ideas much clearer. In this chapter, we first explain why we should care about data visualization. After that, we will discuss several techniques often used for data visualization in R, Python, and Julia. Several special topics will be introduced, such as how to generate a graph, pie chart, and bar chart, how to add a title, trend line, Greek letters, and how to output our graphs. An optional topic at the end of the chapter will discuss dynamic presentations and how to save them as HTML files.

In this chapter, the following topics will be covered:

- Importance of data visualization
- Data visualization in R
- Data visualization in Python
- Data visualization in Julia
- Drawing simple graphs
- Visualization packages for R, Python, and Julia
- Dynamic visualization

Importance of data visualization

For learners, users, or researchers in the areas of data science and business analytics, using various types of graphs, pie charts, bar charts, and other visual means to show some underlying trend or pattern implied by data is vital to understand data and to help researchers present their data to their audience or clients better. There are several reasons for this. First, it is sometimes difficult to describe our findings, especially when we have several patterns or influencing factors. With several separate graphs and a joint one, complex relationships can be understood or explained better.

We can use graphs or pictures to explain certain algorithms, such as the Bisection method (see the section related to dynamic visual presentation, *Dynamic visualization*).

We can also use relative sizes to represent different meanings. In finance, a basic concept is called **time value of money (TVM)**. It means that *a bird in the hand is worth two in the bush*. Today's \$100 is more valuable than a future cash flow of the same amount. With different sizes of different circles representing the present value of cash flow occurring at different points in time in the future, learners can understand this concept much more clearly.

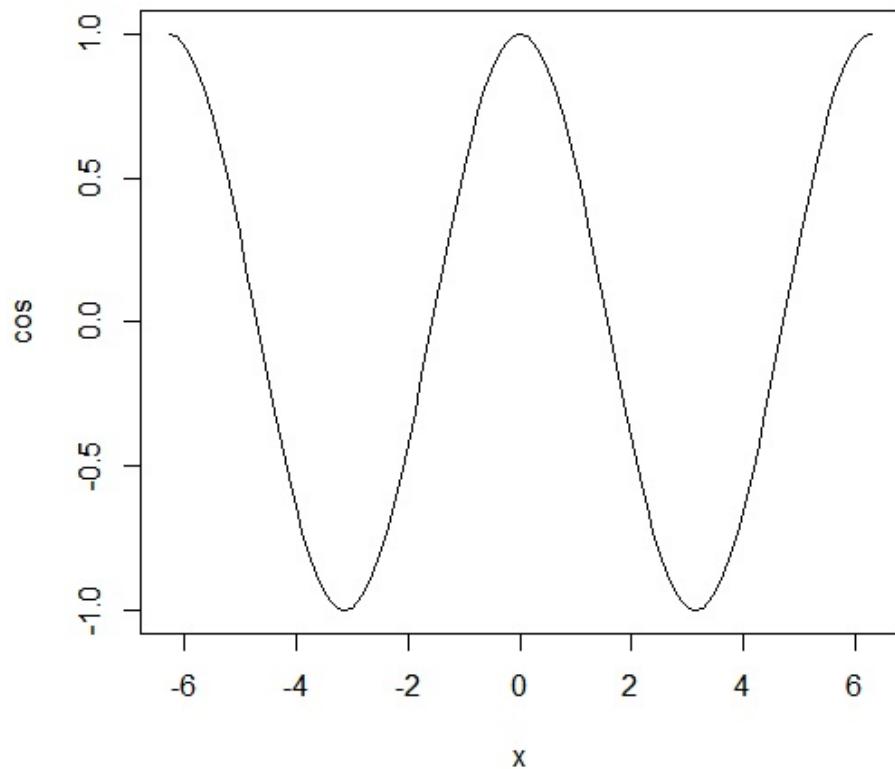
Lastly, our data might be quite messy, and simply showing the data points would confuse audiences further. If we could have a simple graph to show its main characteristics, properties, or patterns, it would help greatly.

Data visualization in R

Firstly, let's see the simplest graph for R. With the following one-line R code, we draw a cosine function from -2π to 2π :

```
|> plot(cos, -2*pi, 2*pi)
```

The related graph is shown here:



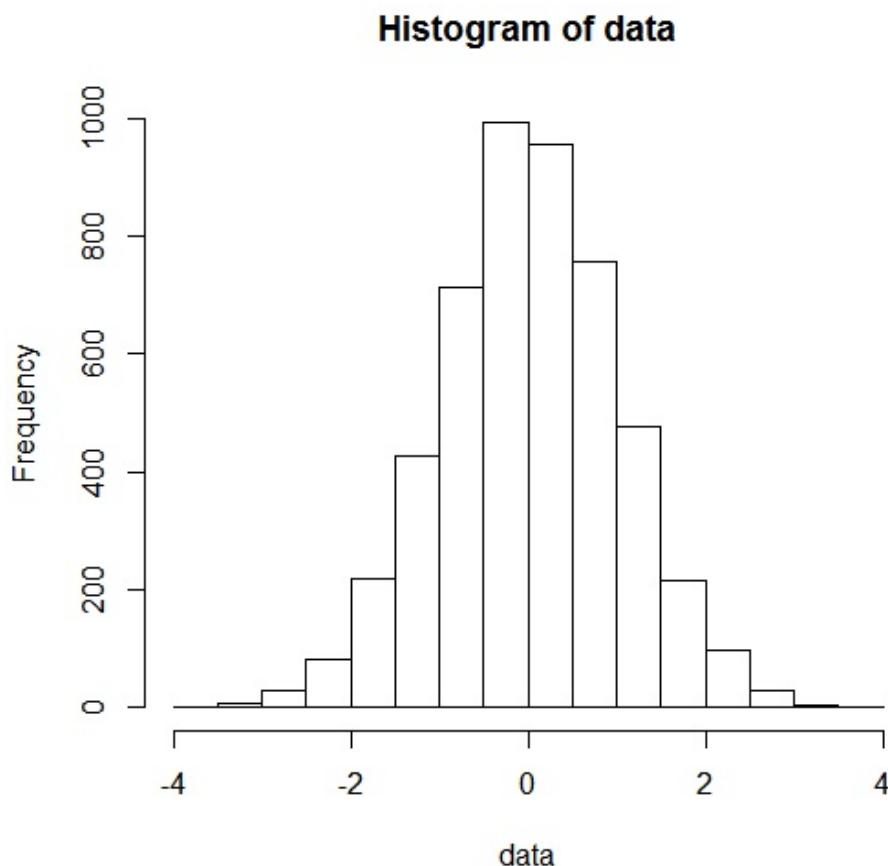
Histograms could also help us understand the distribution of data points. The previous graph is a simple example of this. First, we generate a set of random numbers drawn from a standard normal distribution. For the purposes of illustration, the first line of `set.seed()` is actually redundant. Its existence would guarantee that all users would get the same set of random numbers if the same seed was used (333 in this case).

In other words, with the same set of input values, our histogram would look the

same. In the next line, the `rnorm(n)` function draws n random numbers from a standard normal distribution. The last line then has the `hist()` function to generate a histogram:

```
|> set.seed(333)
|> data<-rnorm(5000)
|> hist(data)
```

The associated histogram is shown here:

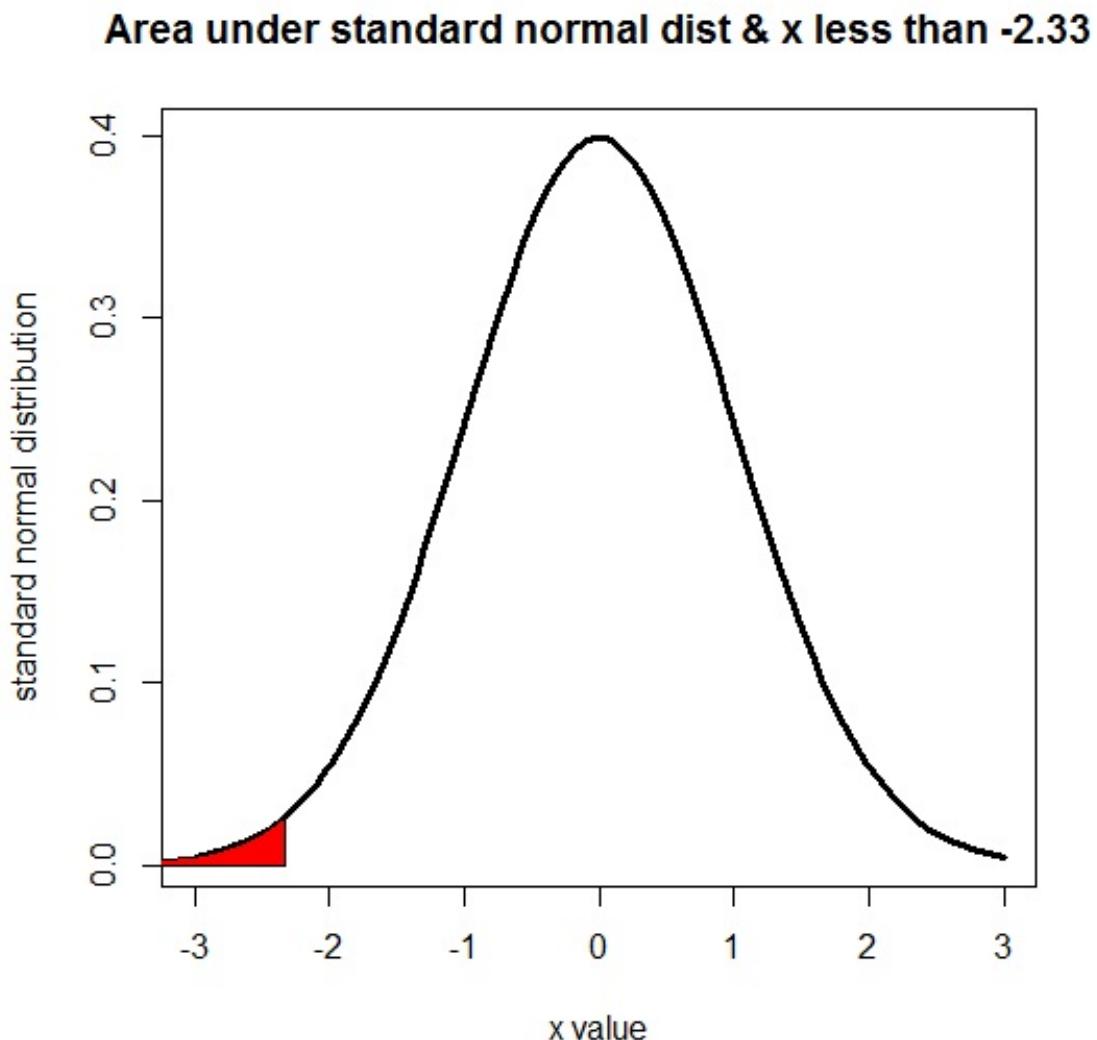


Note that the code of `rnorm(5000)` is the same as `rnorm(5000, mean=0, sd=1)`, which implies that the default value of the `mean` is 0 and the default value for `sd` is 1. The next R program would shade the left-tail for a standard normal distribution:

```
|x<-seq(-3,3,length=100)
y<-dnorm(x,mean=0,sd=1)
title<-"Area under standard normal dist & x less than -2.33"
yLabel<-"standard normal distribution"
xLabel<-"x value"
plot(x,y,type="l",lwd=3,col="black",main=title,xlab=xLabel,ylab=yLabel)
x<-seq(-3,-2.33,length=100)
y<-dnorm(x,mean=0,sd=1)
```

```
| polygon(c(-4,x,-2.33),c(0,y,0),col="red")
```

The related graph is shown here:



Note that according to the last line in the preceding graph, the shaded area is red.

In terms of exploring the properties of various datasets, the R package called `rattle` is quite useful. If the `rattle` package is not preinstalled, we could run the following code to install it:

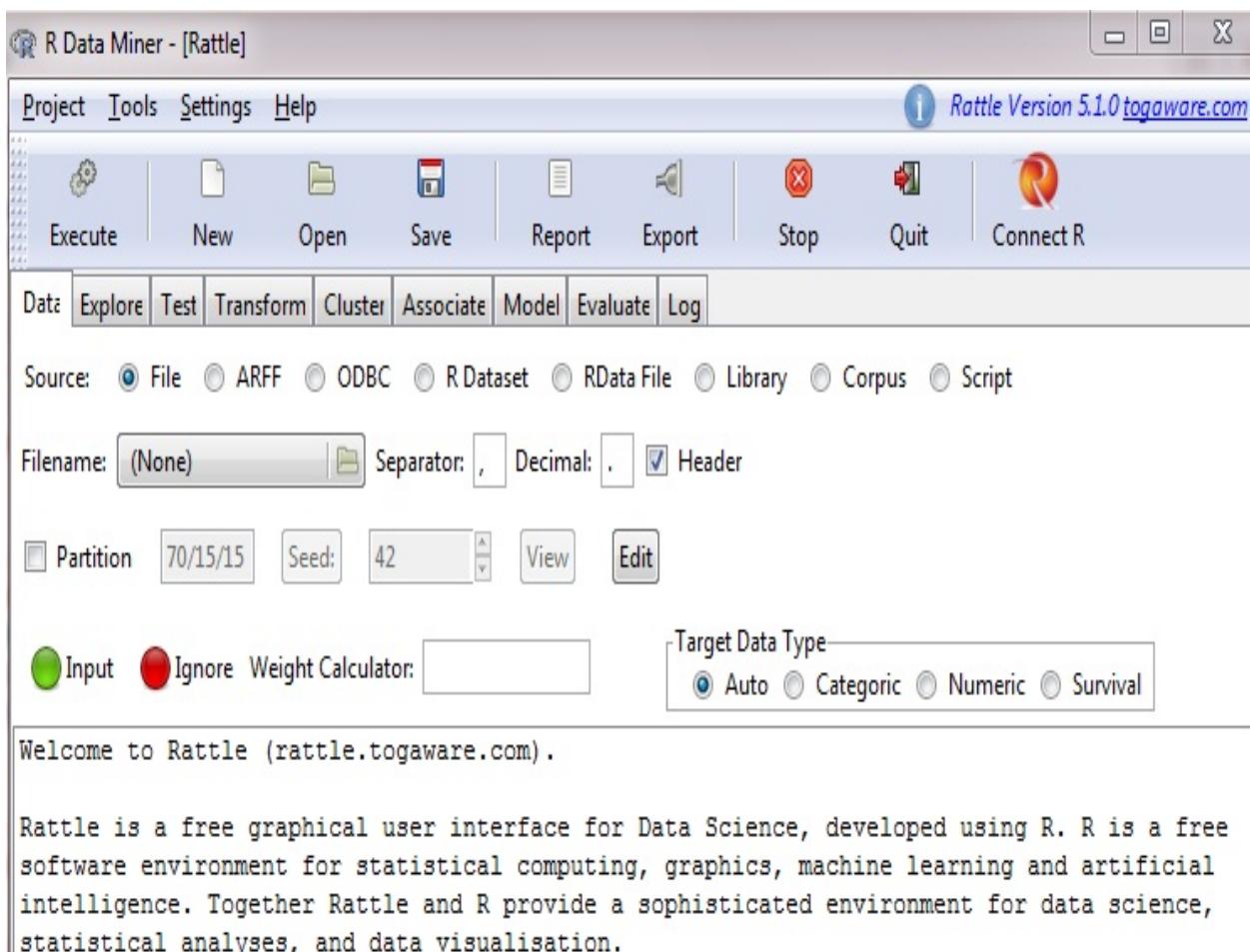
```
| > install.packages("rattle")
```

Then, we run the following code to launch it;

```
| > library(rattle)
```

```
|> rattle()
```

After hitting the *Enter* key, we can see the following:



As our first step, we need to import certain datasets. For the sources of data, we choose from seven potential formats, such as File, ARFF, ODBC, R Dataset, and RData File, and we can load our data from there.

The simplest way is using the `library` option, which would list all the embedded datasets in the `rattle` package. After clicking Library, we can see a list of embedded datasets. Assume that we choose `acme:boot:Monthly Excess Returns` after clicking Execute in the top left. We would then see the following:

The screenshot shows the Rattle software interface. At the top is a menu bar with Project, Tools, Settings, Help, and a status bar indicating "Rattle Version 5.1.0". Below the menu is a toolbar with icons for Execute, New, Open, Save, Report, Export, Stop, Quit, and Connect R. A sub-menu bar below the toolbar has tabs for Data, Explore, Test, Transform, Cluster, Associate, Model, Evaluate, and Log. The "Data" tab is selected. Below this is a "Source:" dropdown menu with options: File, ARFF, ODBC, R Dataset, RData File, Library, Corpus, and Script. A "Data Name:" field contains "acme:boot:Monthly Excess Returns". Below the data name are buttons for "Partition" (checkbox checked), "Seed:" (set to 42), "View", and "Edit". To the right is a "Target Data Type" section with radio buttons for Auto (selected), Categorical, Numeric, and Survival. A "Weight Calculator:" input field is also present. The main area displays a table of variables:

No.	Variable	Data Type	Input	Target	Risk	Ident	Ignore	Weight	Comment
1	month	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	Unique: 60				
2	market	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	Unique: 60				
3	acme	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	Unique: 60				

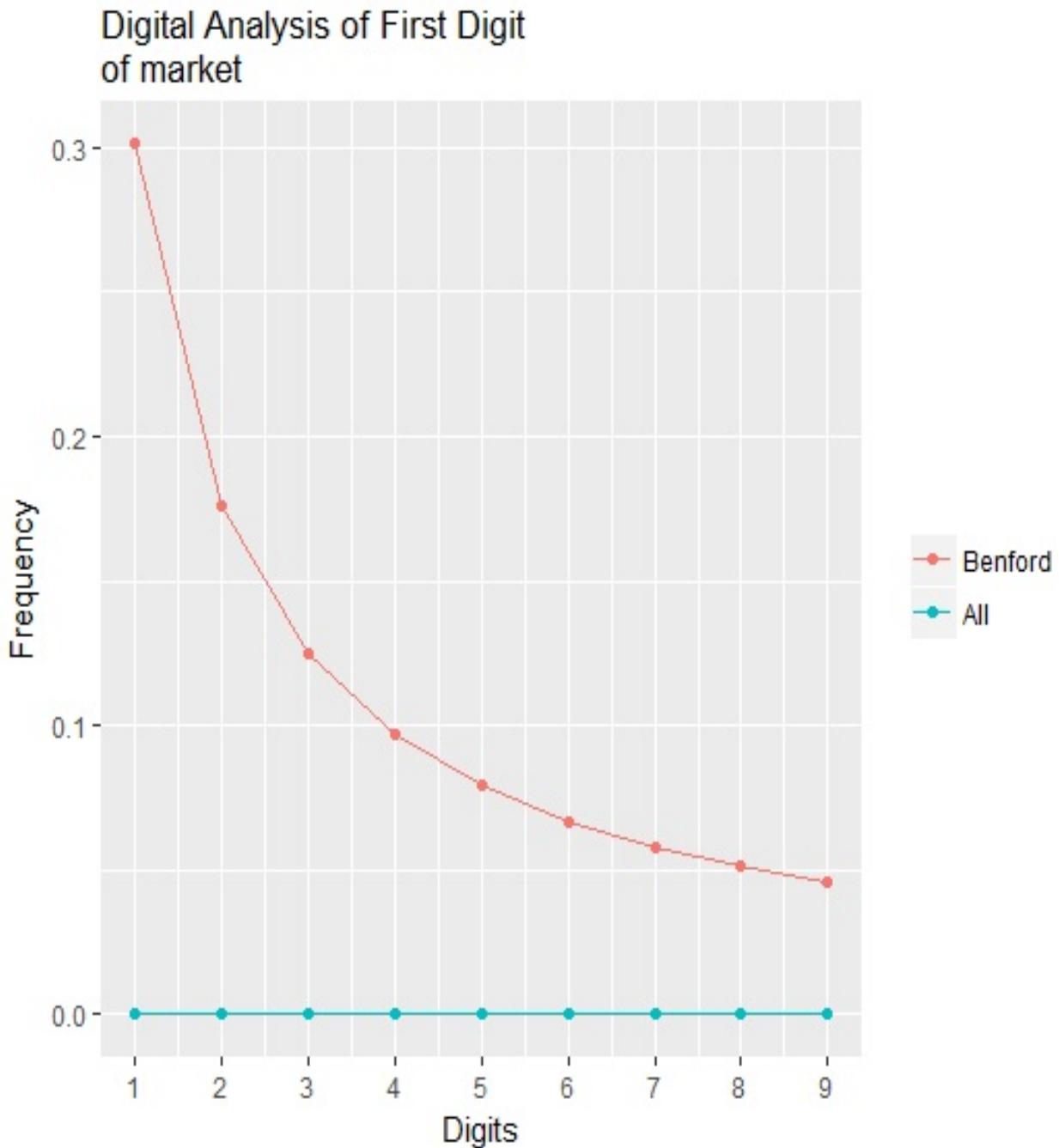
Now, we can study the properties of the dataset. After clicking Explore, we can use various graphs to view our dataset. Assume that we choose Distribution and select the Benford check box. We can then refer to the following screenshot for more details:

The screenshot shows the "Explore" interface. At the top, there are radio buttons for Type: Summary, Distributions (selected), Correlation, Principal Components, and Interactive. Below this are sections for Numeric: Annotate and Group By: (with a dropdown menu). Under Benfords:, there are checkboxes for Bars and Starting Digit: 1, Digits: 1, and options for abs, +ve, and -ve. The main area displays a table of variables with distribution analysis results:

No.	Variable	Box Plot	Histogram	Cumulative Benford	Pairs	Min; Median/Mean; Max	
2	market				<input checked="" type="checkbox"/>		-0.26; -0.04/-0.05; 0.07
3	acme						-0.28; -0.09/-0.07; 0.24

After clicking Execute, the following would pop up. The top red line shows the frequencies for the **Benford** Law for each digits of 1 to 9, while the blue line at the bottom shows the properties of our data set. Note that if you don't have the

reshape package already installed in your system, then this either won't run or will ask for permission to install the package to your computer:



The dramatic difference between those two lines indicates that our data does not follow a distribution suggested by the Benford Law. In our real world, we know that many people, events, and economic activities are interconnected, and it would be a great idea to use various graphs to show such a multi-node,

interconnected picture. If the `qgraph` package is not preinstalled, users have to run the following to install it:

```
|> install.packages("qgraph")
```

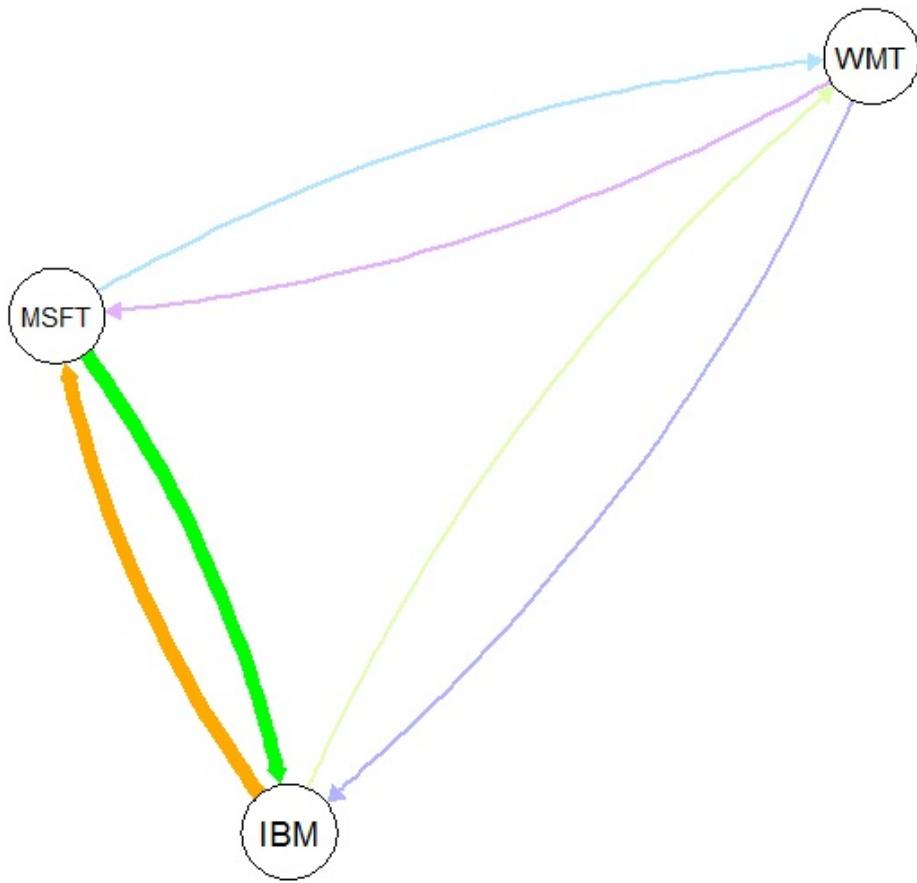
The next program shows the connection from a to b, a to c, and the like:

```
library(qgraph)
stocks<-c("IBM", "MSFT", "WMT")
x<-rep(stocks, each = 3)
y<-rep(stocks, 3)
correlation<-c(0,10,3,10,0,3,3,3,0)
data <- as.matrix(data.frame(from =x, to =y, width =correlation))
qgraph(data, mode = "direct", edge.color = rainbow(9))
```

If the data is shown, the meaning of the program will be much clearer. The correlation shows how strongly those stocks are connected. Note that all those values are randomly chosen with no real-world meanings:

```
|> data
      from    to    width
[1,] "IBM"  "IBM"  " 0"
[2,] "IBM"  "MSFT" "10"
[3,] "IBM"  "WMT"  " 3"
[4,] "MSFT" "IBM"  "10"
[5,] "MSFT" "MSFT" " 0"
[6,] "MSFT" "WMT"  " 3"
[7,] "WMT"  "IBM"  " 3"
[8,] "WMT"  "MSFT" " 3"
[9,] "WMT"  "WMT"  " 0"
```

A high value for the third variable suggests a stronger correlation. For example, `IBM` is more strongly correlated with `MSFT`, with a value of `10`, than its correlation with `WMT`, with a value of `3`. The following graph shows how strongly those three stocks are correlated:



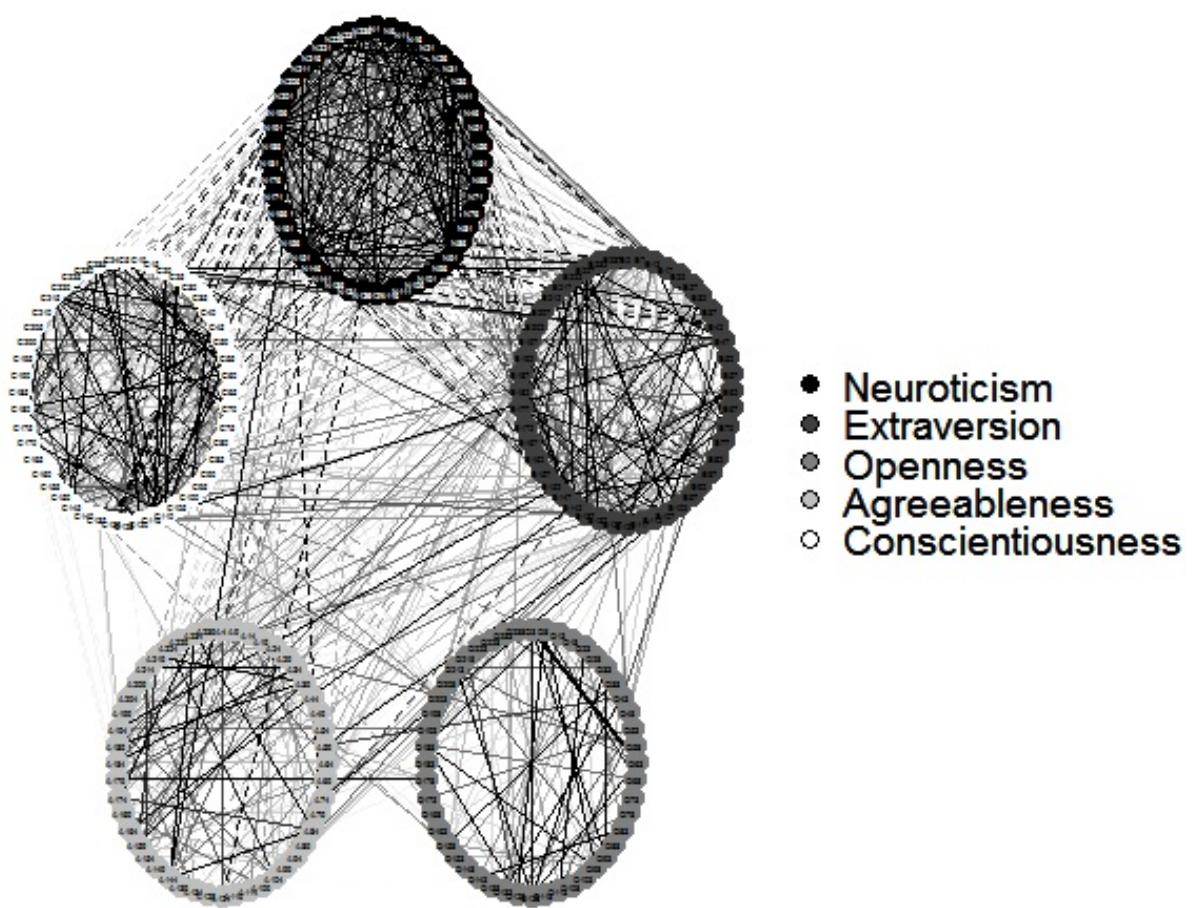
The following program shows the relationship or interconnection between five factors:

```

library(qgraph)
data(big5)
data(big5groups)
title("Correlations among 5 factors",line = 2.5)
qgraph(cor(big5),minimum = 0.25,cut = 0.4,vsize = 1.5,
      groups = big5groups,legend = TRUE, borders = FALSE,theme = 'gray')

```

The related graph is shown here:

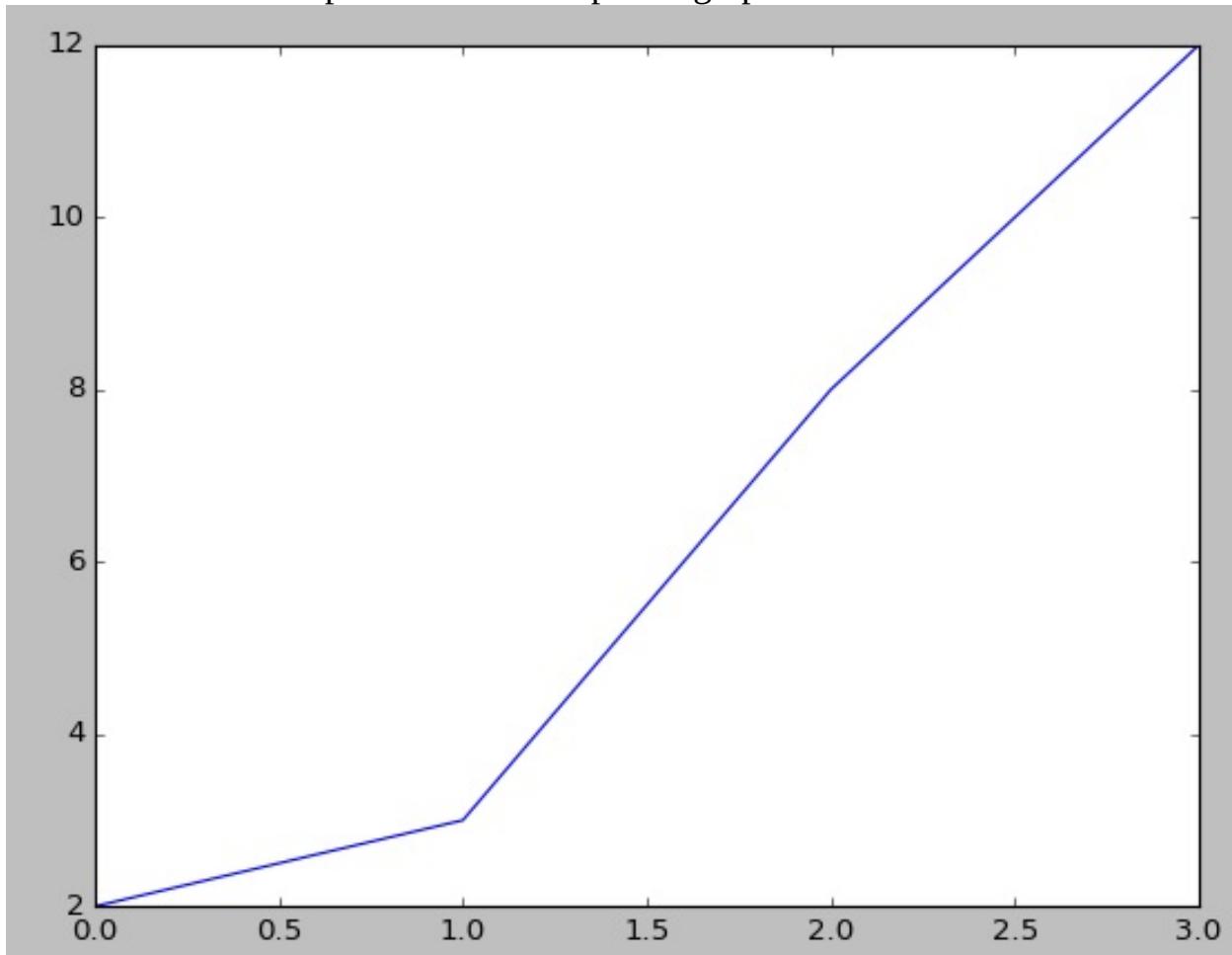


Data visualization in Python

The most widely used Python package for graphs and images is called `matplotlib`. The following program can be viewed as the simplest Python program to generate a graph since it has just three lines: `import matplotlib.pyplot as plt`
`plt.plot([2,3,8,12])` `plt.show()`

The first command line would upload a Python package called `matplotlib.pyplot` and rename it to `plt`.

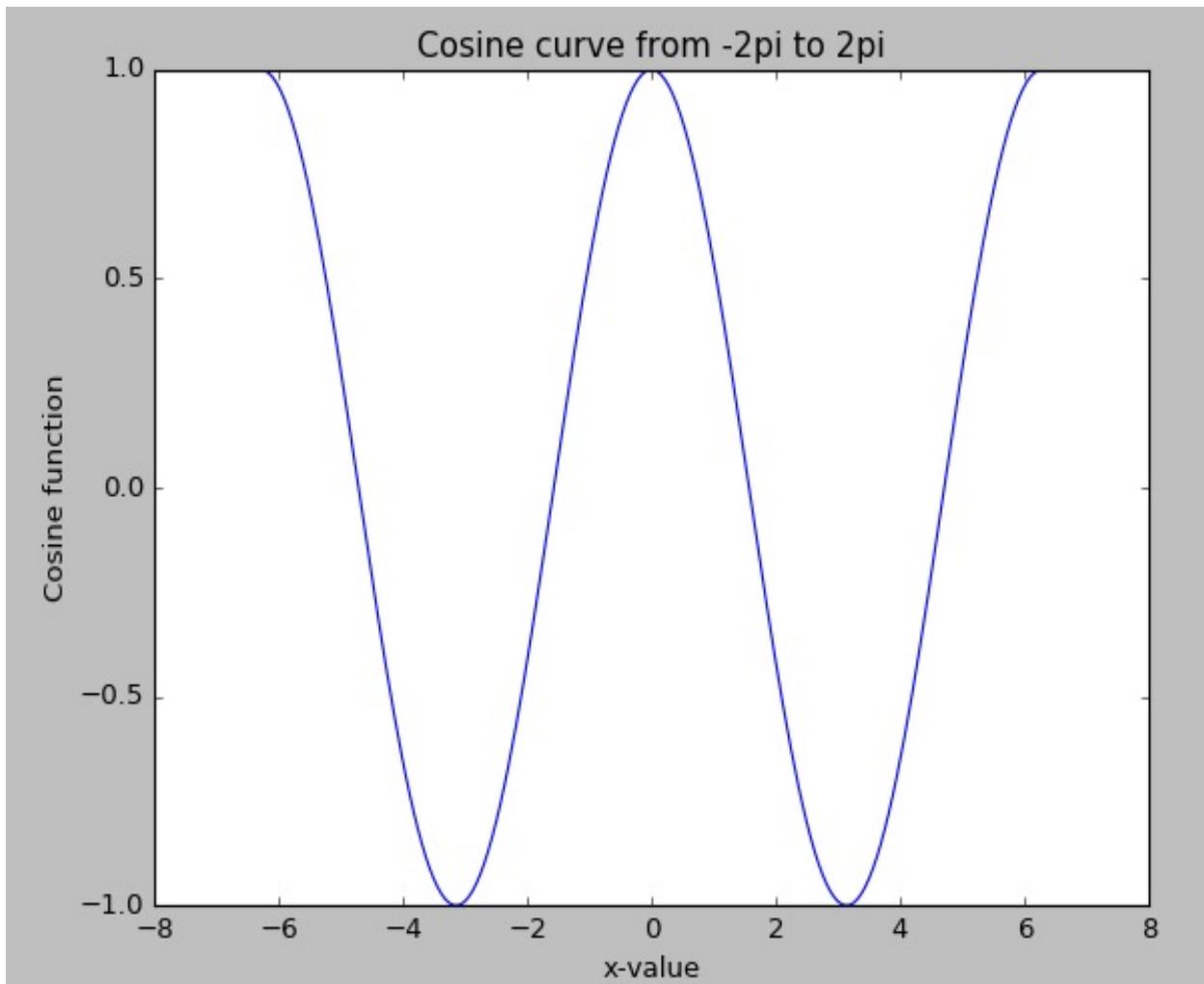
Note that we could even use other short names, but it is conventional to use `plt` for the `matplotlib` package. The second line plots four points, while the last one concludes the whole process. The completed graph is shown here:



For the next example, we add labels for both x and y , and a title. The function is the cosine function with an input value varying from -2π to 2π :

```
import scipy as sp
import matplotlib.pyplot as plt
x=sp.linspace(-2*sp.pi,2*sp.pi,200,endpoint=True)
y=sp.cos(x)
plt.plot(x,y)
plt.xlabel("x-value")
plt.ylabel("Cosine function")
plt.title("Cosine curve from -2pi to 2pi")
plt.show()
```

The nice-looking cosine graph is shown here:



If we received \$100 today, it would be more valuable than what would be received in two years. This concept is called the time value of money, since we could deposit \$100 today in a bank to earn interest. The following Python

```

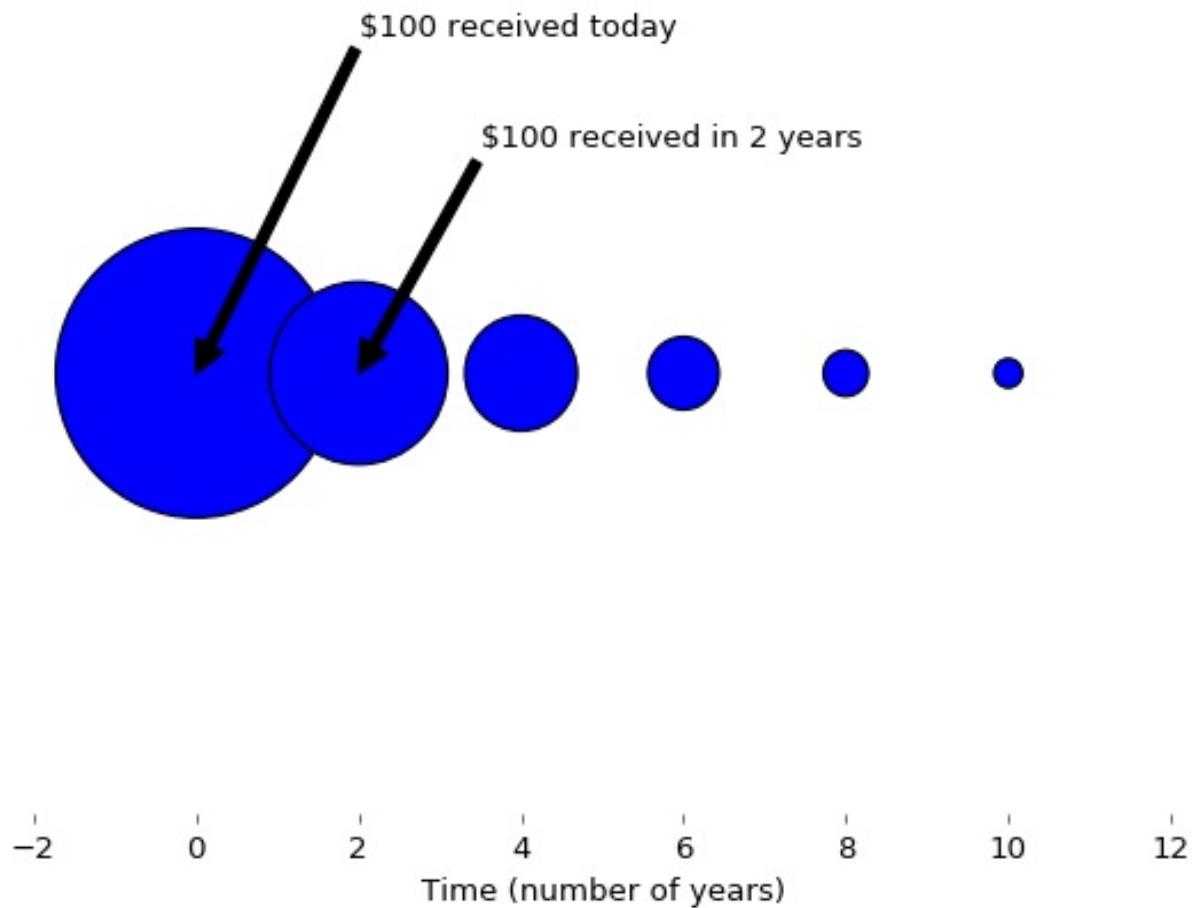
program uses size to illustrate this concept: import matplotlib.pyplot as plt fig =
plt.figure(facecolor='white') dd = plt.axes(frameon=False)
dd.set_frame_on(False) dd.get_xaxis().tick_bottom()
dd.axes.get_yaxis().set_visible(False) x=range(0,11,2) x1=range(len(x),0,-1) y =
[0]*len(x); plt.annotate("$100 received today",xy=(0,0),xytext=
(2,0.15),arrowprops=dict(facecolor='black',shrink=2)) plt.annotate("$100
received in 2 years",xy=(2,0),xytext=
(3.5,0.10),arrowprops=dict(facecolor='black',shrink=2)) s = [50*2.5**n for n in
x1];

|plt.title("Time value of money ")
|plt.xlabel("Time (number of years)")
|plt.scatter(x,y,s=s);
|plt.show()

```

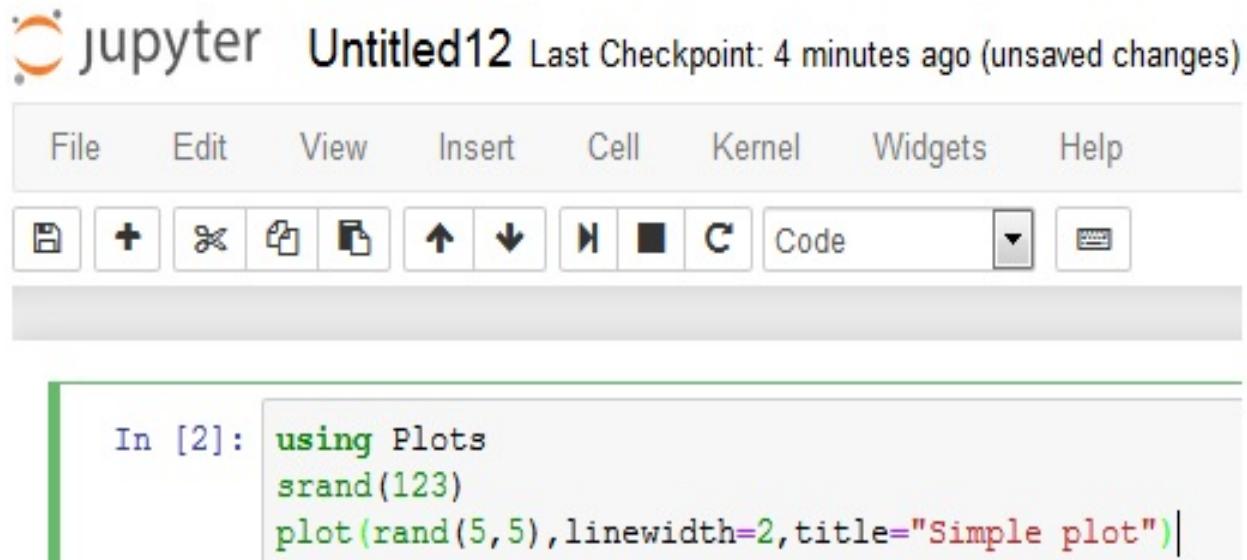
The associated graph is shown here. Again, the different sizes show their present values in relative terms:

Time value of money



Data visualization in Julia

For the following Julia program, we use a package called `Plots`. The command used to install the package is `Pkg.add("Plots")`. Here, we run Julia programs via a Jupyter notebook. The Julia program is presented in the following screenshot:

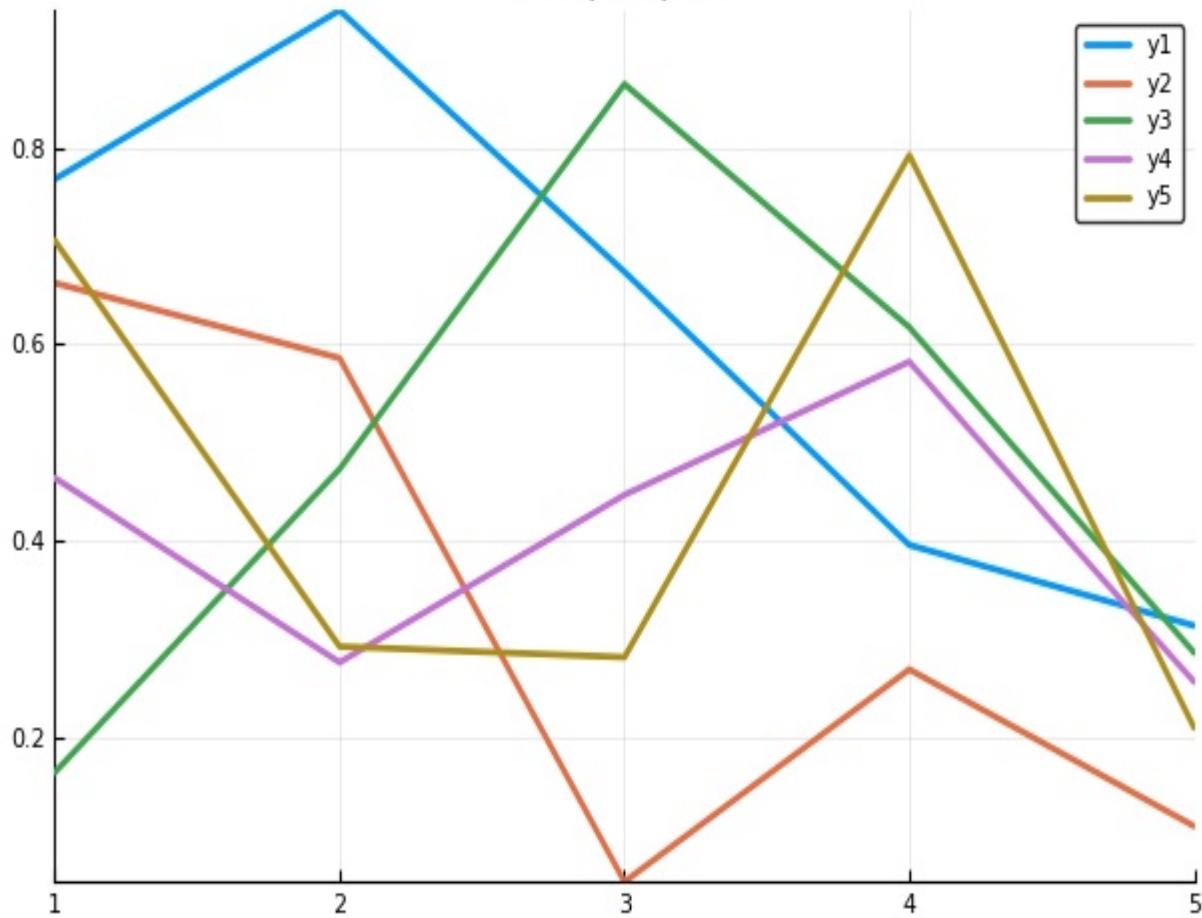


The screenshot shows a Jupyter Notebook interface. At the top, there's a toolbar with various icons for file operations like opening, saving, and running cells. Below the toolbar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The main area contains a code cell labeled "In [2]". The code in the cell is:

```
using Plots
srand(123)
plot(rand(5,5), linewidth=2, title="Simple plot")|
```

After clicking Kernel on the menu bar, and then Restart and Run All, we get the following:

Simple plot

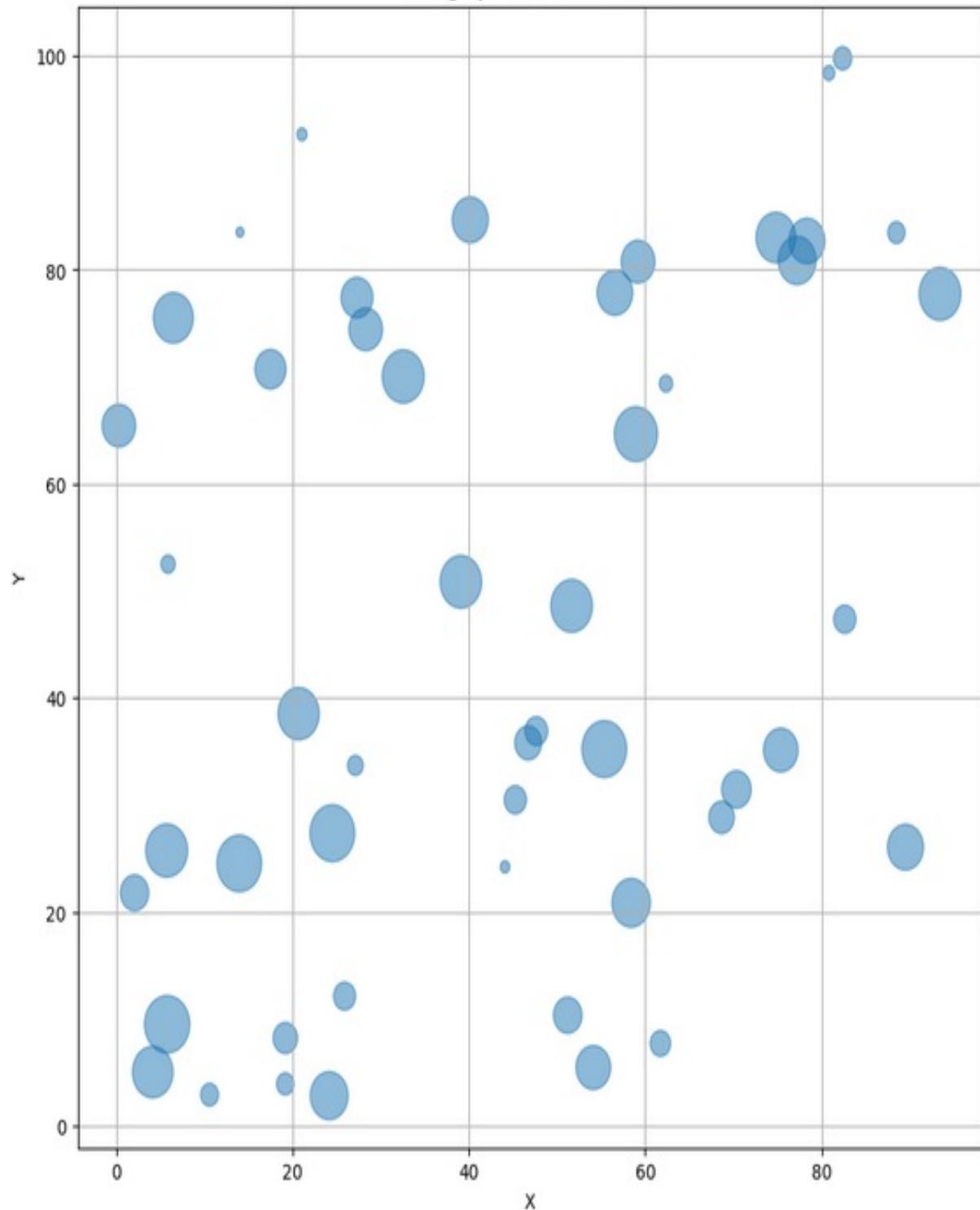


Again, the `srand(123)` command guarantees that any user who applies the same random seed will get the same set of random numbers. Because of this, he/she would get the same graph shown previously. The next example is a scatter plot using a Julia package called `PyPlot`:

```
using PyPlot
n=50
srand(333)
x = 100*rand(n)
y = 100*rand(n)
areas = 800*rand(n)
fig = figure("pyplot_scatterplot", figsize=(10,10))
ax = axes()
scatter(x,y,s=areas,alpha=0.5)
title("using PyPlot: Scatter Plot")
xlabel("X")
ylabel("Y")
grid("on")
```

The related graph is shown here:

using PyPlot: Scatter Plot



The next Julia program is borrowed from *Sargent and Stachurski (2017)*:

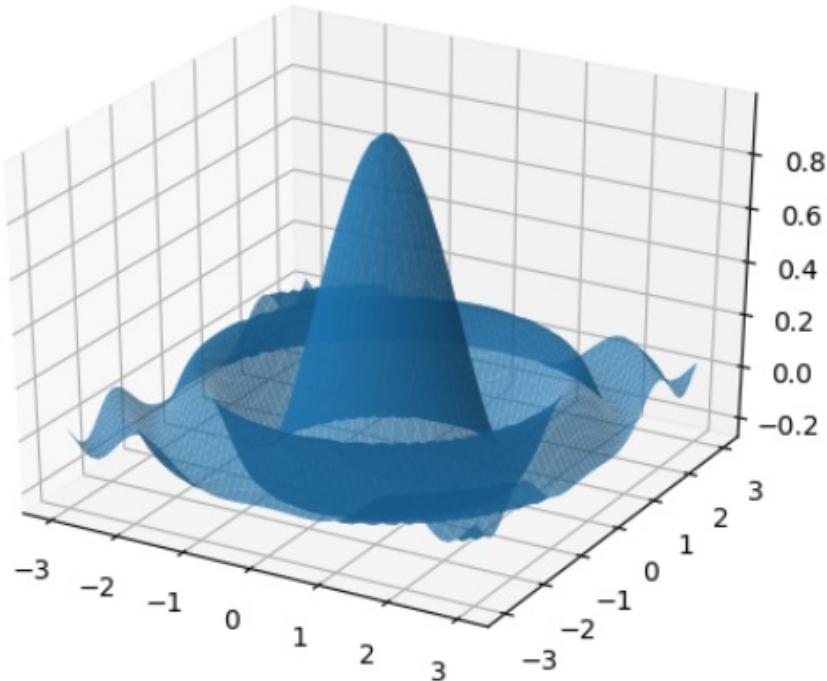
```
| using QuantEcon: meshgrid
```

```

using PyPlot:surf
using Plots
n = 50
x = linspace(-3, 3, n)
y = x
z = Array{Float64}(n, n)
f(x, y) = cos(x^2 + y^2) / (1 + x^2 + y^2)
for i in 1:n
    for j in 1:n
        z[j, i] = f(x[i], y[j])
    end
end
xgrid, ygrid = meshgrid(x, y)
surf(xgrid, ygrid, z', alpha=0.7)

```

The impressive graph is shown here:

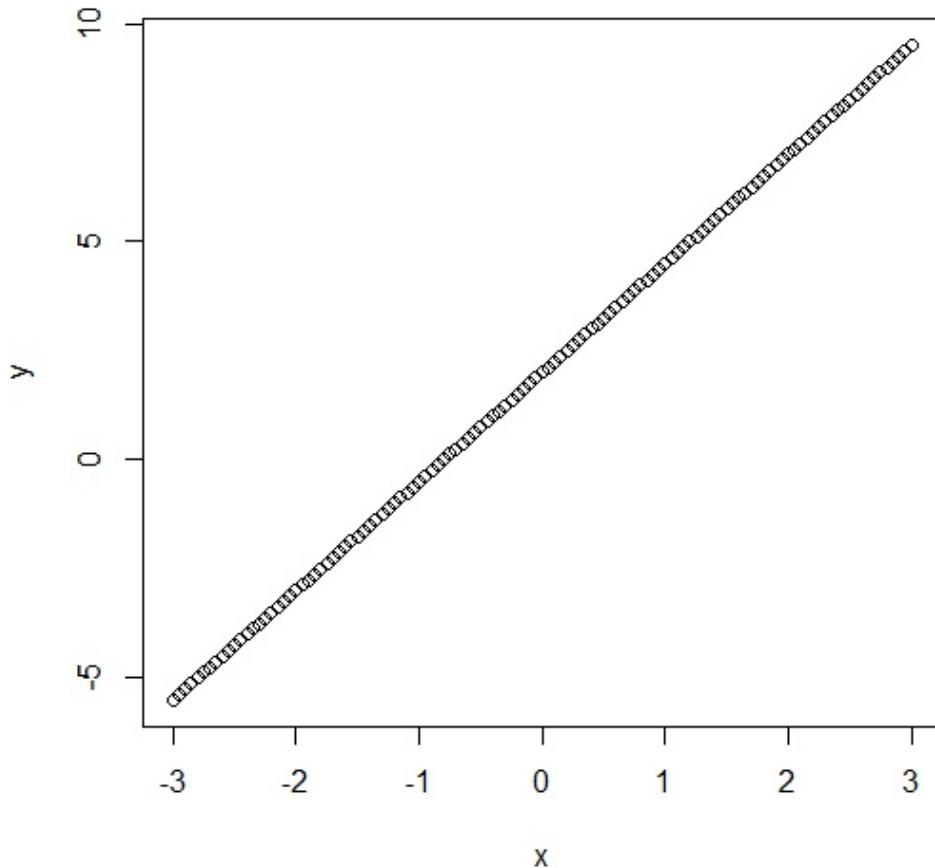


Drawing simple graphs

The simplest graph will be a straight line. For R, we have the following example:

```
| x<-seq(-3,3,by=0.05)
| y<-2+2.5*x
| plot(x,y,type="b")
```

In this simple program, type would specify the format of the line, and b would specify both the spot and the line. The corresponding graph is shown here:



The possible values for type are given in the following table:

Value	Description

p	for points
l	for lines
b	for both
c	for the lines part, alone of b
o	for both overplotted
h	for histogram-like (or high-density) vertical lines
s	for stair steps
s	for other steps (<i>see the following details</i>)
n	for no plotting

Table 4.1 Possible values for type in the R function plot()

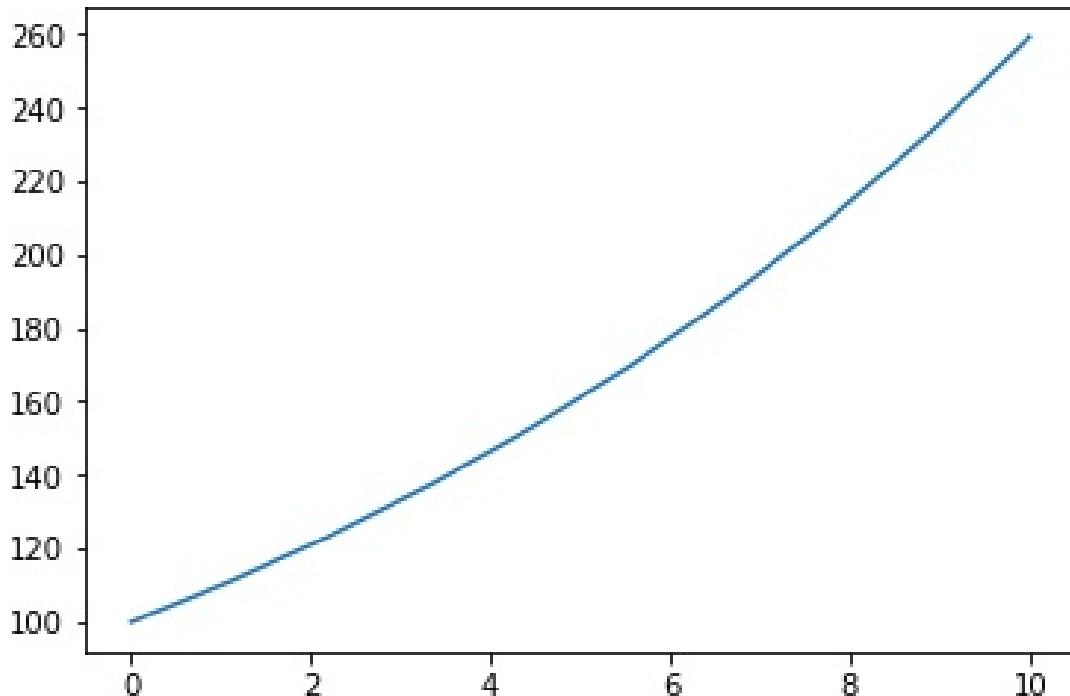
For Python, we have the following simple example for a future value, given the present value and interest rate:

```
import numpy as np
import matplotlib.pyplot as plt
n=np.linspace(0,10,10)
pv=100
R=0.1
fv=pv*(1+R)**n
plt.plot(n,fv)
plt.show()
```

The formula used is given here, where FV is the future value, PV is the present value, R is the interest rate, and n is the number of periods:

$$FV = PV(1 + R)^n$$

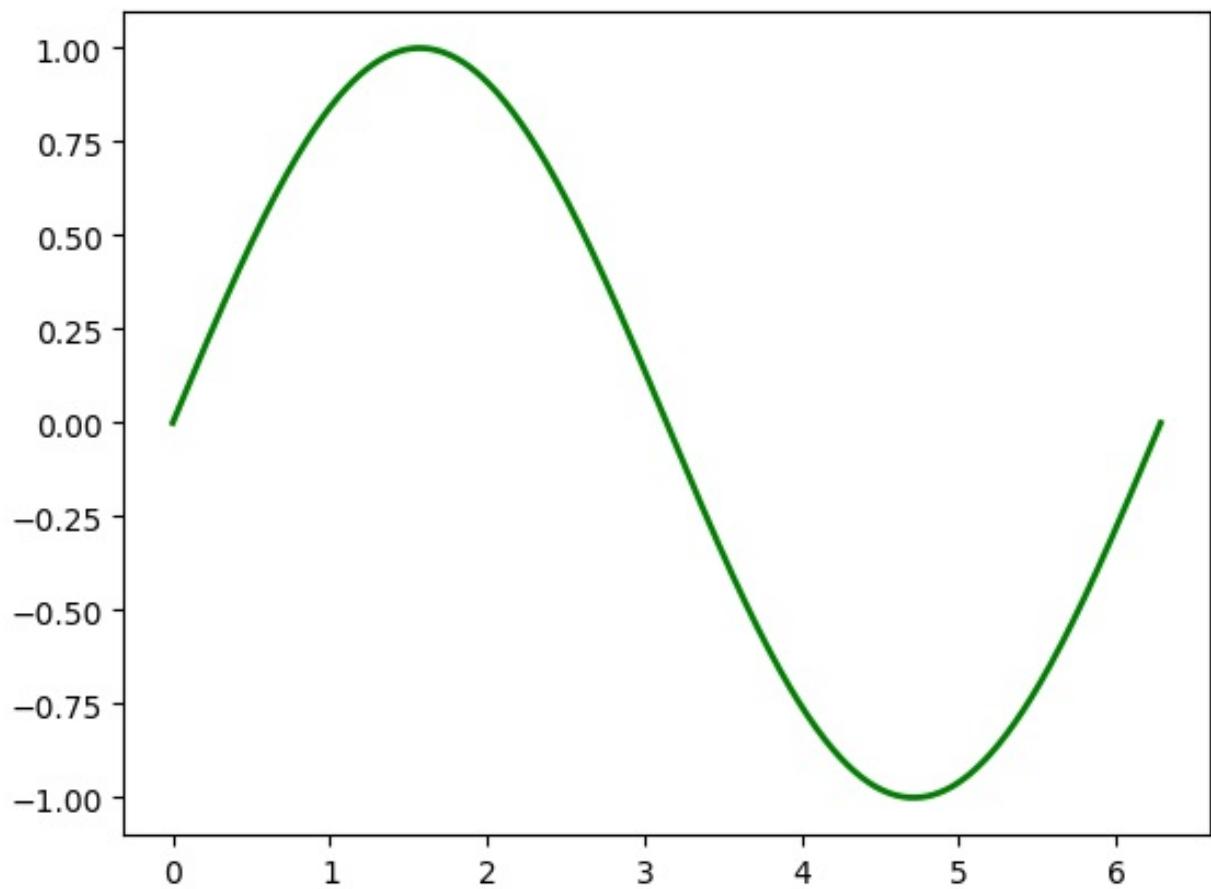
The associated line is shown here:



For Julia, one simple example is drawing the sine function. The input variable for the sine function varies from 0 to 2π :

```
using PyPlot
x = linspace(0, 2*pi, 1000)
y = sin.(x)
plot(x, y, color="green", linewidth=2.0)
```

The corresponding graph is shown here:

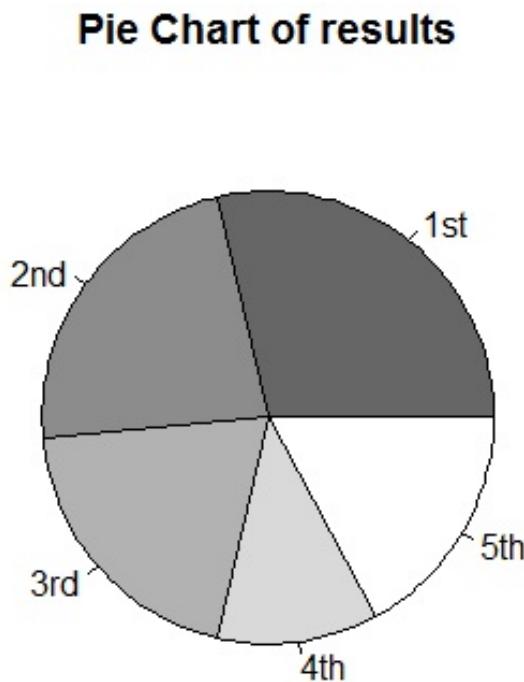


Various bar charts, pie charts, and histograms

For the next R program, we show a pie chart based on the percentage of each input value:

```
results <- c(10,8,7,4,6)
names <- c("1st","2nd","3rd","4th","5th")
pct <- round(results/sum(results)*100)
pct2 <- rev(sort(pct))
name <- "Pie Chart of results"
colors <- seq(0.4, 1.0, length = length(results))
pie(results, labels = names, col = gray(colors), main=name)
```

The related graph is shown here:

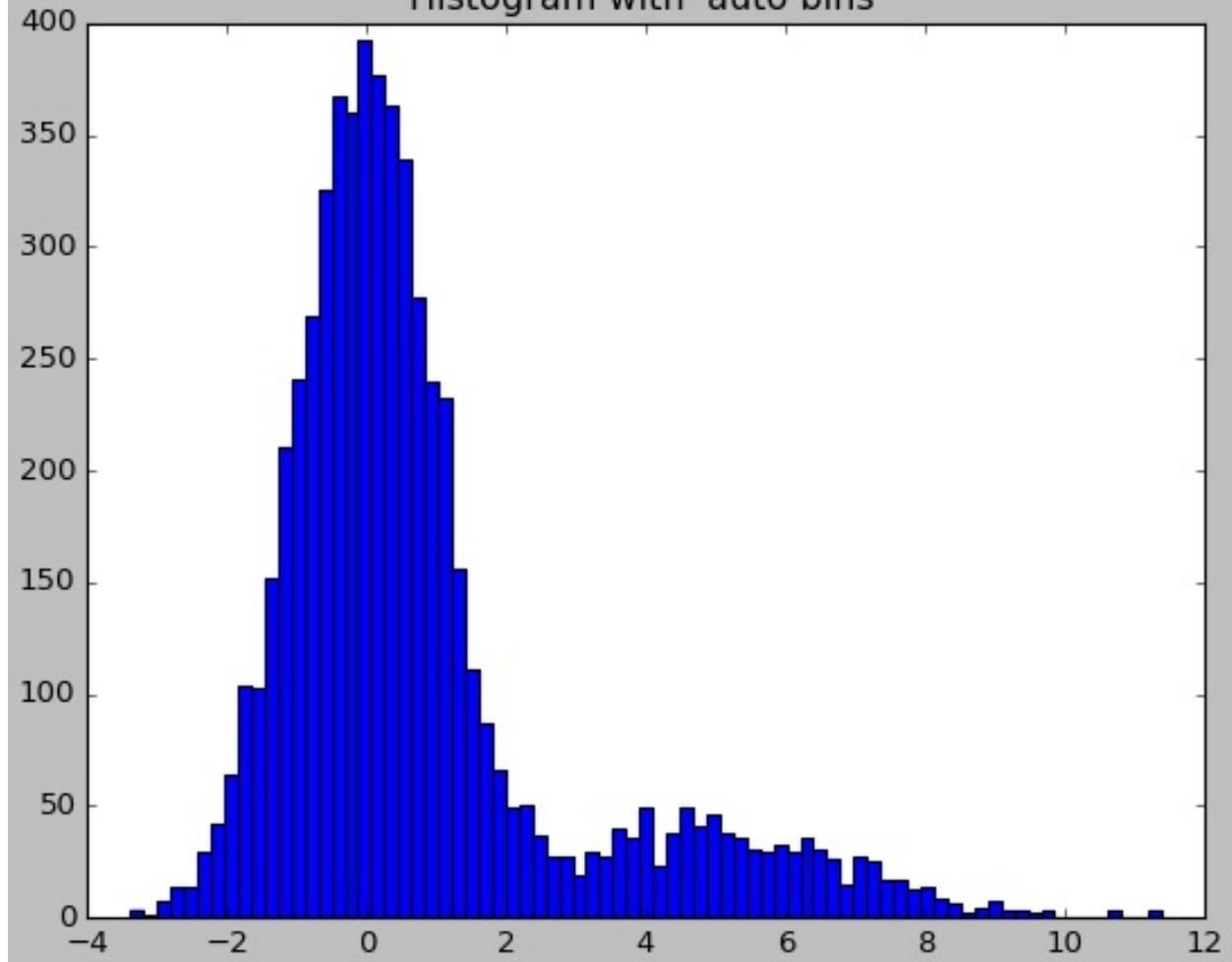


The next Python program shows the histogram for a standard normal distribution:

```
import numpy as np
import matplotlib.pyplot as plt
data = np.random.RandomState(10)
n=5000
n2=1000
x=data.normal(size=n)
y=data.normal(loc=5, scale=2, size=n2)
a=(x,y)
b = np.hstack(a)
plt.hist(b, bins='auto')
plt.title("Histogram with 'auto bins'")
plt.show()
```

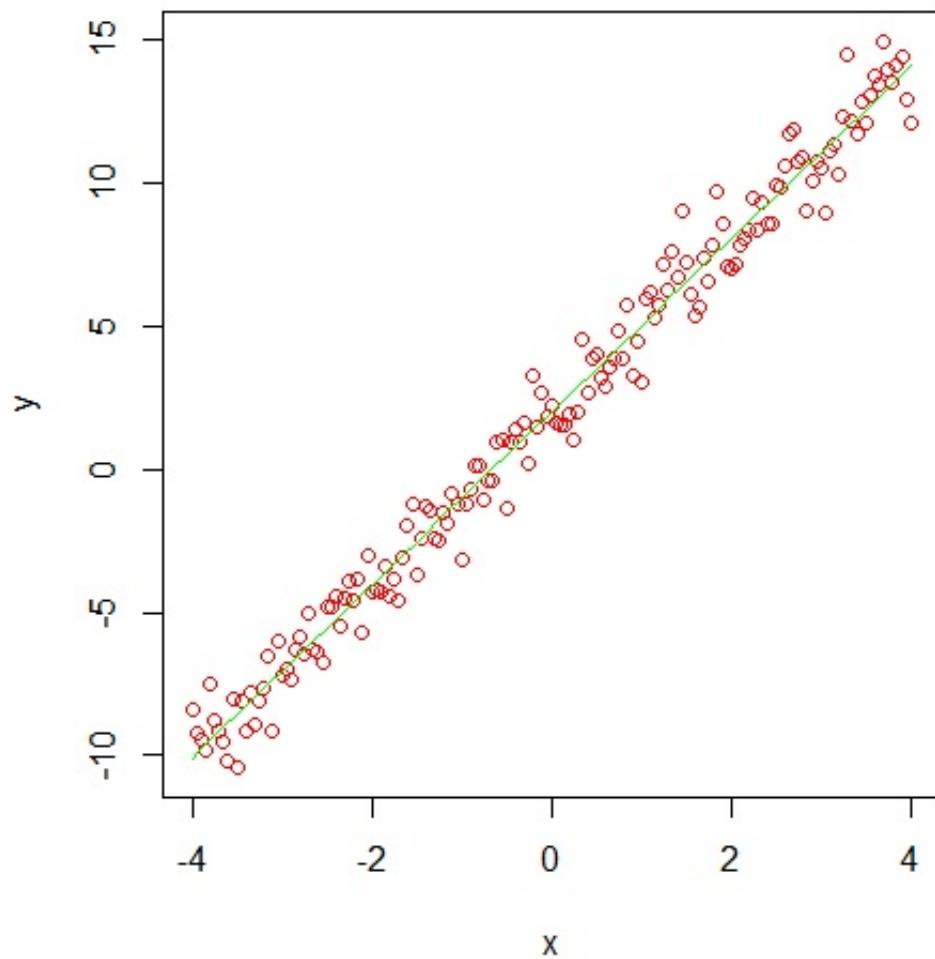
The associated graph is shown here:

Histogram with 'auto bins'



```
x=seq(-4,4,by=0.05)
n<-length(x)
y=2+3*x + rnorm(n)
k<-coef(lm(y~x))
intercept<-k[1]
slope<-k[2]
y2<-intercept+slope*x <br/>#
plot(x,y,type="p",col="red")
lines(x,y2,col="green")
```

The graph associated with the previous program is shown here:

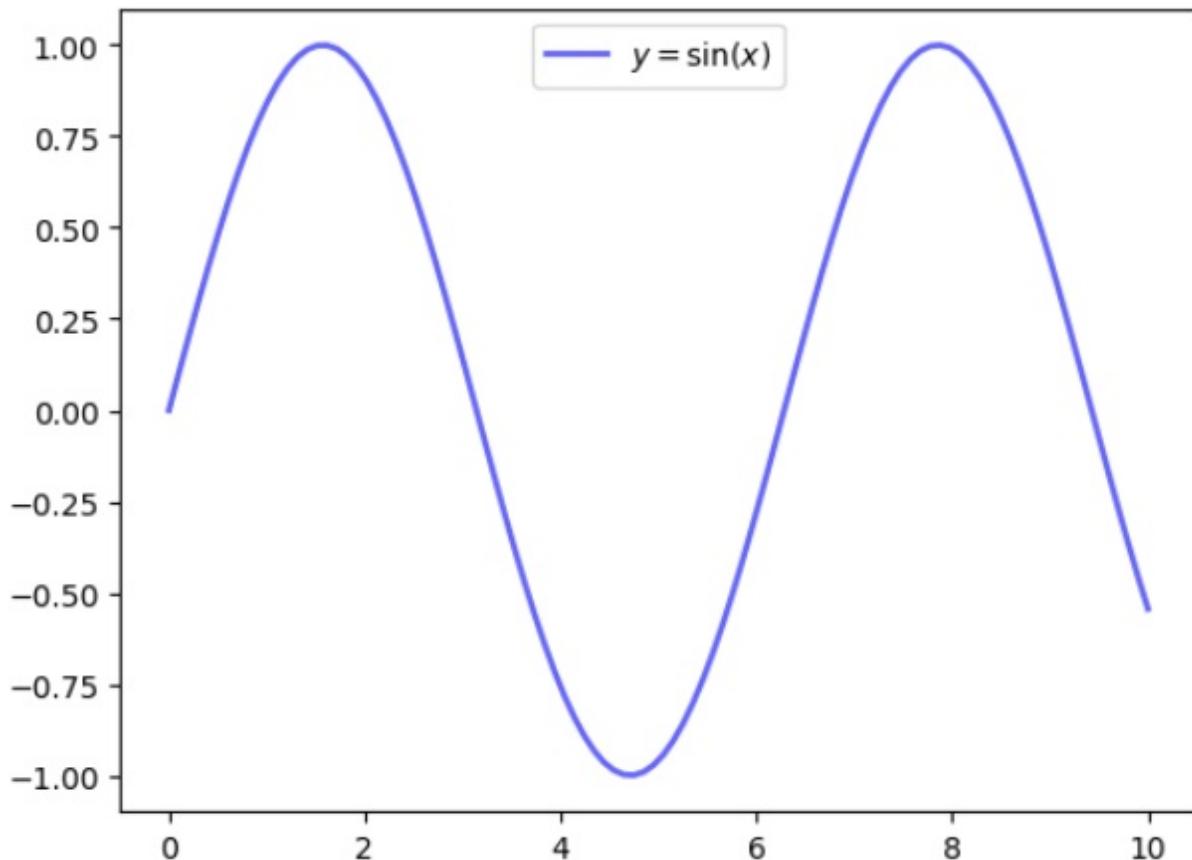


Adding legends and other explanations

In the following Julia function, we add the legend $y=\sin(x)$ to the output graph:

```
using PyPlot
x = linspace(0, 10, 100)
y = sin.(x)
name=L"$y = \sin(x)$"
fig, ax = subplots()
ax[:plot](x, y, "b-", linewidth=2, label=name, alpha=0.6)
ax[:legend](loc="upper center")
```

The associated graph is shown here:



In the R program, we add the following formula to the graph (the formula for the normal distribution):

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

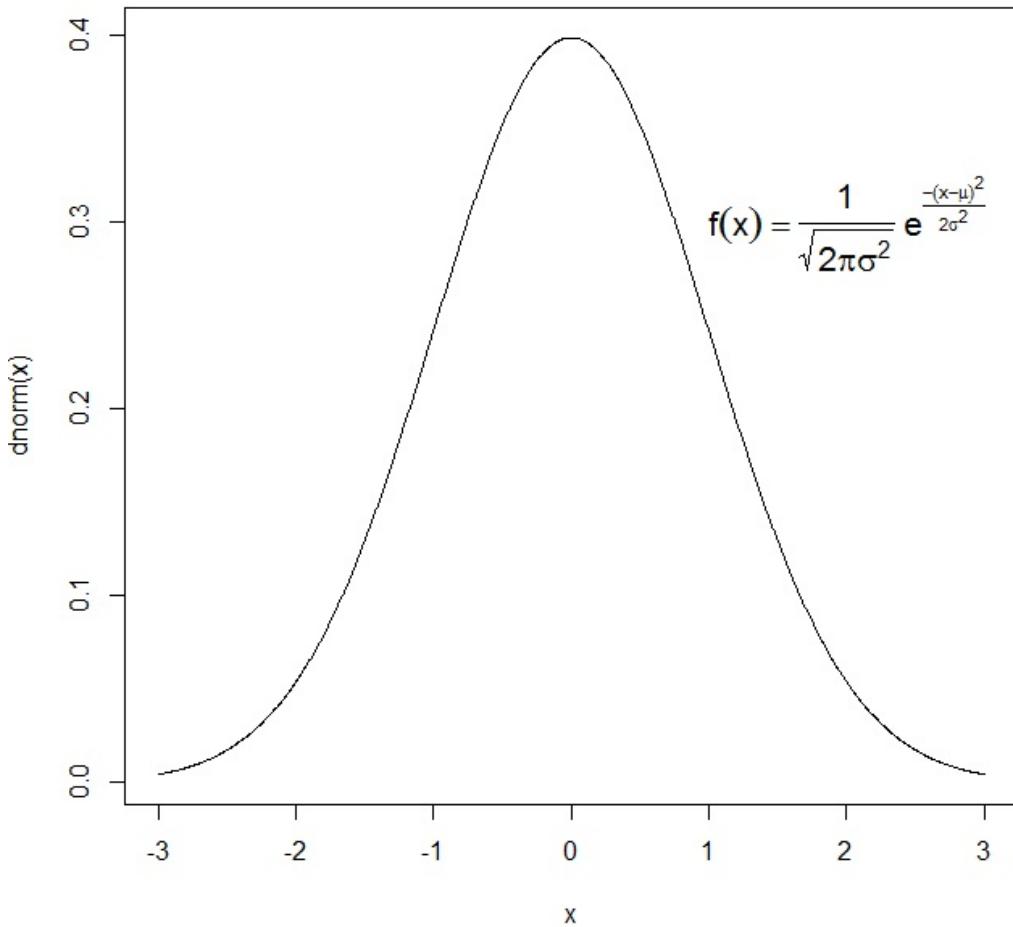
The R code is given here:

```
| set.seed(12345)
| mu=4
| std=2
| nRandom=2000
| x <- rnorm(mean =mu, sd =std, n =nRandom)
| name<- "Normal Probability Density Function"
| curve(dnorm, from = -3, to = 3, n = 1000, main = name)
| formula<-expression(f(x) ==paste(frac(1, sqrt(2 * pi * sigma^2)), " ", e^{frac(-(x - mu)^2, 2)}))
| text(2, 0.3, formula, cex = 1.3)
```

$$\frac{1}{2}$$

Note that the command of `frac(1,2)` would generate $\frac{1}{2}$. When included in the expression function, `sigma` would generate σ and `mu` would generate μ . In a sense, they are reserved words when included in the expression function. The related graph is given here:

Normal Probability Density Function



The next Python program generates the image for the famous **Black-Scholes** formula:

```
import matplotlib
import numpy as np
import matplotlib.mathtext as mathtext
import matplotlib.pyplot as plt
matplotlib.rc('image', origin='upper')
parser = mathtext.MathTextParser("Bitmap")
#parser = mathtext.MathTextParser("Black")
#
x1=r'$d\_2=\frac{\ln(S\_0/K)+(r-\sigma^2/2)T}{\sigma\sqrt{T}}=d\_1-\sigma\sqrt{T}$'
x2=r'$d\_1=\frac{\ln(S\_0/K)+(r+\sigma^2/2)T}{\sigma\sqrt{T}}$'
x3=r' $c=S\_0N(d\_1)- Ke^{-rT}N(d\_2)$'
rgba1, depth1 = parser.to_rgba(x1, color='blue', fontsize=12, dpi=200)
rgba2, depth2 = parser.to_rgba(x2, color='blue', fontsize=12, dpi=200)
rgba3, depth3 = parser.to_rgba(x3, color='red', fontsize=14, dpi=200)
fig = plt.figure()

fig.figimage(rgba1.astype(float)/255., 100, 100)
fig.figimage(rgba2.astype(float)/255., 100, 200)
fig.figimage(rgba3.astype(float)/255., 100, 300)
plt.show()
```

The associated image is shown here:

$$c = S_0 N(d_1) - K e^{-rT} N(d_2)$$

$$d_1 = \frac{\ln(S_0/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{\ln(S_0/K) + (r - \sigma^2/2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$$

In the preceding program, the backslash of \ is used many times. For example, \frac{}{} indicates that this is a ratio, as shown in the following table:

Symbol	Meanings (example)
\frac{a}{b}	$\frac{a}{b}$
\sigma	σ
\sqrt{T}	\sqrt{T}
a_1	a1
s_0	s0

Table 4.2 Meanings of a few notations

Visualization packages for R

To find out about some packages written in R, you can go to the home page for the R software at <http://r-project.org> and click CRAN on the left-hand side. Then, you can choose a mirror location (close to you). After clicking Packages on the left-hand side, you will find two lists. One is sorted by date while the second one is sorted alphabetically. To search for useful packages, we can use both lists.

There are many R packages associated with data visualization (see the partial list that follows):

Name of R package	Description
showtext	Use fonts more easily in R graphs
shinyaframe	WebVR Data Visualizations with RStudio Shiny and Mozilla A-Frame
moveVis	Movement Data Visualization
sjPlot	Data Visualization for Statistics in Social Science
qgraph	Graph Plotting methods, Psychometric Data Visualization, and Graphical Model Estimation
statquotes	Quotes on Statistics, Data Visualization, and Science

HistData	Datasets from the History of Statistics and Data Visualization
loon	Interactive Statistical Data Visualization
Bdvis	Biodiversity Data Visualizations
visdat	Preliminary Data Visualization
Ggfortify	Data Visualization Tools for Statistical Analysis Results
ggplot2	Create Elegant Data Visualizations Using the Grammar of Graphics
RcmdrPlugin.KMggplot2	R Commander Plugin for Data Visualization with ggplot2
hierarchicalSets	Set Data Visualization Using Hierarchies
Dashboard	Interactive Data Visualization with D3.js
Mapplots	Data Visualization on Maps
zoom	A spatial data visualization tool

Table 4.3 A partial list of R packages Here, we use `ggplot2` as an example to show you how to get information about this specific package. Note that when uploading an R package, if we want to eliminate package startup messages, we could use `suppressPackageStartupMessages()`. To get more information about this package, we could issue the following two lines: >
library(ggplot2) > help(package=ggplot2)

The first R command line would upload the package while the second one would go to a specific website related to this package (see the first part in the following screenshot):

Documentation for package 'ggplot2'

- [DESCRIPTION file](#).
- [User guides, package vignettes and other documentation](#).

Help Pages

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [L](#) [M](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [X](#) [Y](#) [misc](#)

-- A --

aes	Construct aesthetic mappings
aes_	Define aesthetic mappings programatically
aes_colour_fill_alpha	Colour related aesthetics: colour, fill and alpha

Visualization packages for Python

The most widely used Python package is `matplotlib`. To find all its functions, we could issue the following three lines: `import matplotlib as mat` `x=dir(mat)` `print(x)`

The output is shown here:

```
In [7]: print(x)
['MutableMapping', 'RcParams', 'URL_REGEX', 'Verbose', '_DATA_DOC_APPENDIX', '__bibtex__', '__builtins__', '__cached__',
 '__doc__', '__file__', '__loader__', '__name__', '__package__', '__path__', '__spec__', '__version__', '__version_numpy__',
 '__all_DEPRECATED', '__backports__', '__cm__', '__cm_listed__', '__cntr__', '__color_data__', '__contour__', '__create_tmp_config_dir',
 '__decode_filesystem_path', '__deprecated_ignore_map', '__deprecated_map', '__deprecated_set', '__error_details_fmt',
 '__get_cachedir', '__get_config_or_cache_dir', '__get_configdir', '__get_data_path', '__get_data_path_cached', '__get_home',
 '__get_xdg_cache_dir', '__get_xdg_config_dir', '__image__', '__init_tests', '__is_writable_dir', '__mathtext_data', '__obsolete_set',
 '__open_file_or_url', '__path__', '__png__', '__preprocess_data', '__pylab_helpers', '__python27', '__python34', '__qhull',
 '__rc_params_in_file', '__replacer', '__tri__', '__url_lines', '__use_error_msg', '__version', 'absolute_import', 'afm', 'artist',
 'axes', 'axis', 'backend_bases', 'backend_tools', 'backends', 'bezier', 'blocking_input', 'category', 'cbook',
 'checkdep_dvipng', 'checkdep_ghostscript', 'checkdep_inkscape', 'checkdep_pdftops', 'checkdep_ps_distiller', 'checkdep_tex',
 'checkdep_usetex', 'checkdep_xmllint', 'cm', 'collections', 'colorbar', 'colors', 'compare_versions', 'compat', 'container',
 'contextlib', 'contour', 'cycler', 'dates', 'dateutil', 'dedent', 'defaultParams', 'default_test_modules', 'distutils',
 'division', 'docstring', 'dviread', 'figure', 'font_manager', 'fontconfig_pattern', 'ft2font', 'functools', 'get_backend',
 'get_cachedir', 'get_configdir', 'get_data_path', 'get_home', 'get_label', 'get_py2exe_datafiles', 'gridspec', 'image',
 'inspect', 'interactive', 'io', 'is_interactive', 'is_url', 'itertools', 'legend', 'legend_handler', 'lines', 'locale',
 'markers', 'mathtext', 'matplotlib_fname', 'mlab', 'mplDeprecation', 'numpy', 'offsetbox', 'os', 'patches', 'path',
 'print_function', 'projections', 'pyparsing', 'pyplot', 'quiver', 'rc', 'rcParams', 'rcParamsDefault', 'rcParamsOrig',
 'rc_context', 'rc_file', 'rc_file_defaults', 'rc_params', 'rc_params_from_file', 'rcdefaults', 'rcsetup', 're', 'reload',
 'sanitize_sequence', 'scale', 'six', 'spines', 'stackplot', 'streamplot', 'style', 'subprocess', 'sys', 'table', 'tempfile',
 'test', 'texmanager', 'text', 'textpath', 'ticker', 'tight_bbox', 'tk_window_focus', 'transforms', 'tri', 'unicode_literals',
 'units', 'urlopen', 'use', 'validate_backend', 'verbose', 'warnings', 'widgets']
```

To find out about all Python packages associated with data visualization, we can go to <https://pypi.python.org>. The following table shows the first several packages when we search `data visualization`, where `W` is the weight (in terms of popularity):

Package	W	Description
PyQtDataVisualization 5.9.2	11	Python bindings for the Qt Data Visualization library
data-utilities 1.2.8	10	A data analysis and visualization helper module
DataEnvironment 0.0.0	10	Data environment is for data analysis, visualization, and management
dataswim 0.4.10	10	Utilities to swim in a data lake
datashader 0.4	9	Data visualization tool chain based on aggregating data into a grid
scikit-data 0.1.3	9	The purpose of this library is to allow the data analysis process to be more easy and automatic
DataSpyre 0.2.7	8	Spyre makes it easy to build interactive web applications and requires no knowledge of HTML, CSS, or JavaScript
Khooshe 0.2	8	Big GeoSptial Data Points Visualization Tool
lolviz 1.3.1	8	A simple Python data structure visualization tool for lists of lists, lists, and dictionaries

bitjet 0.2.1	7	Binary visualization using IPython widgets
ckanext-wirecloud_view 1.0.0	7	Provides visualization dashboards on CKAN using WireCloud
Data-viz 0.13	7	Represents your high-dimensional data in a 2D space and plays with it
datapusher 0.0.12	7	A standalone web service that parses the contents of a CKAN site's data files and pushes them into its DataStore
datatank_py 0.71	7	Module for reading and writing DataTank files
gmane 0.1.dev25	7	Utilities for the analysis of the GMANE email list database
gviz_data_table 1.0.2	7	Python API for Google visualization
lightning-pthon 1.2.1	7	A Python client library for the Lightning data visualization server
Orange3-Netwerk 1.3.0	7	Network add-on for the Orange 3 data mining software package

phovea_data_hdf 0.1.0	7	Data provider plugin for loading data stored in hierarchical data format (HDF)
phovea_data_mongo 0.1.0	7	Data provider plugin for loading graph data stored in a MongoDB
phovea_data_redis 0.1.0	7	Data provider plugin for loading data stored in a Redis in-memory database
phovea_data_sql 0.1.0	7	Data provider plugin for loading tabular data stored in an SQLite database
psyplot 1.0.0 .post1	7	Python package for interactive data visualization
scikit-dataaccess 1.9.9	7	Scikit Data Access Package for accessing scientific datasets

Table 4.4 A partial list of Python packages associated with data visualization

Visualization packages for Julia

Similarly, we can go to <https://pkg.julialang.org> to find out about all the packages written in Julia for data visualization:

Package	Description
<code>ExperimentalAnalysis</code>	A package for creating scatter matrix plots and visualizing linear models and <code>DataFrames</code>
<code>GLBooks</code>	<code>GLVisualize</code> used as a notebook
<code>GLVisualize</code>	Visualization library written in Julia and OpenGL
<code>GR</code>	Plotting for Julia based on GR, a framework for visualization applications
<code>GraphPlot</code>	Graph visualization for Julia
<code>NetworkViz</code>	Julia interface to visualize graphs
<code>Plots</code>	Powerful convenience for Julia visualizations and data analysis
<code>ProfileView</code>	Visualization of Julia profiling data

D3Trees

Flexible interactive visualization for large trees in Julia
using D3.js

Table 4.5 A partial list of packages written in Julia for data visualization

Dynamic visualization

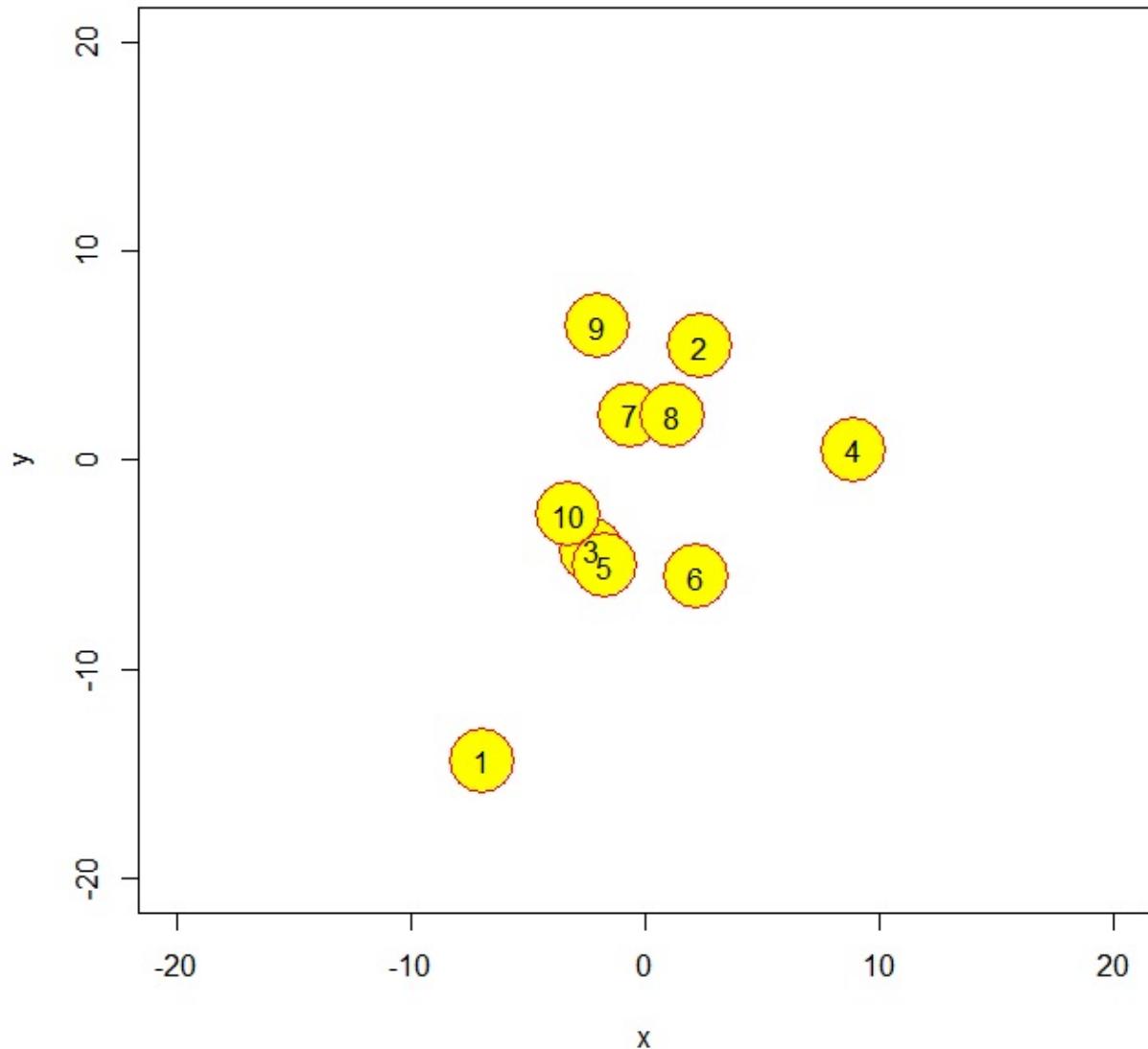
Dynamic visualization means that we can see the concept or theory vividly, just like a video or movie. For example, in the Monte Carlo simulation, we could generate a set of random numbers from certain distributions, such as a uniform or a normal distribution. If we could show how particles moved within a range, learners would understand the concept of randomness better. This is like dropping red ink into a basin of water and observing how the ink diffuses. Another example is related to the so-called bisection method or bisection search. Assume that we have a database that contains daily trading data for over 5,000 stocks. There are many ways to retrieve one specific stock's information. One is called sequential search. In a sequential search, we pick up the first stock and compare it with our target. If it doesn't match, we go to the next stock. Since it uses too much computational power, it is not efficient. The bisection method is much more efficient.

In this section, we illustrate several examples based on the R package called `animation`. The first example is to dynamically show the random movement called Brownian motion, shown in this code:

```
library(animation)
ani.options(interval = 0.05, nmax = 30)
a<- c(3,3,2,0.5)
b<-c(2,0.5,0)
name<-"Brownian Motion"
par(mar=a,mgp=b,tcl=-0.3,cex.axis=0.8,cex.lab=0.8,cex.main=1)
brownian.motion(pch=21,cex=5,col="red",bg="yellow",main=name)
```

One static picture is shown here. Again, the preceding code would show the Brownian moving dynamically (just like watching a short movie):

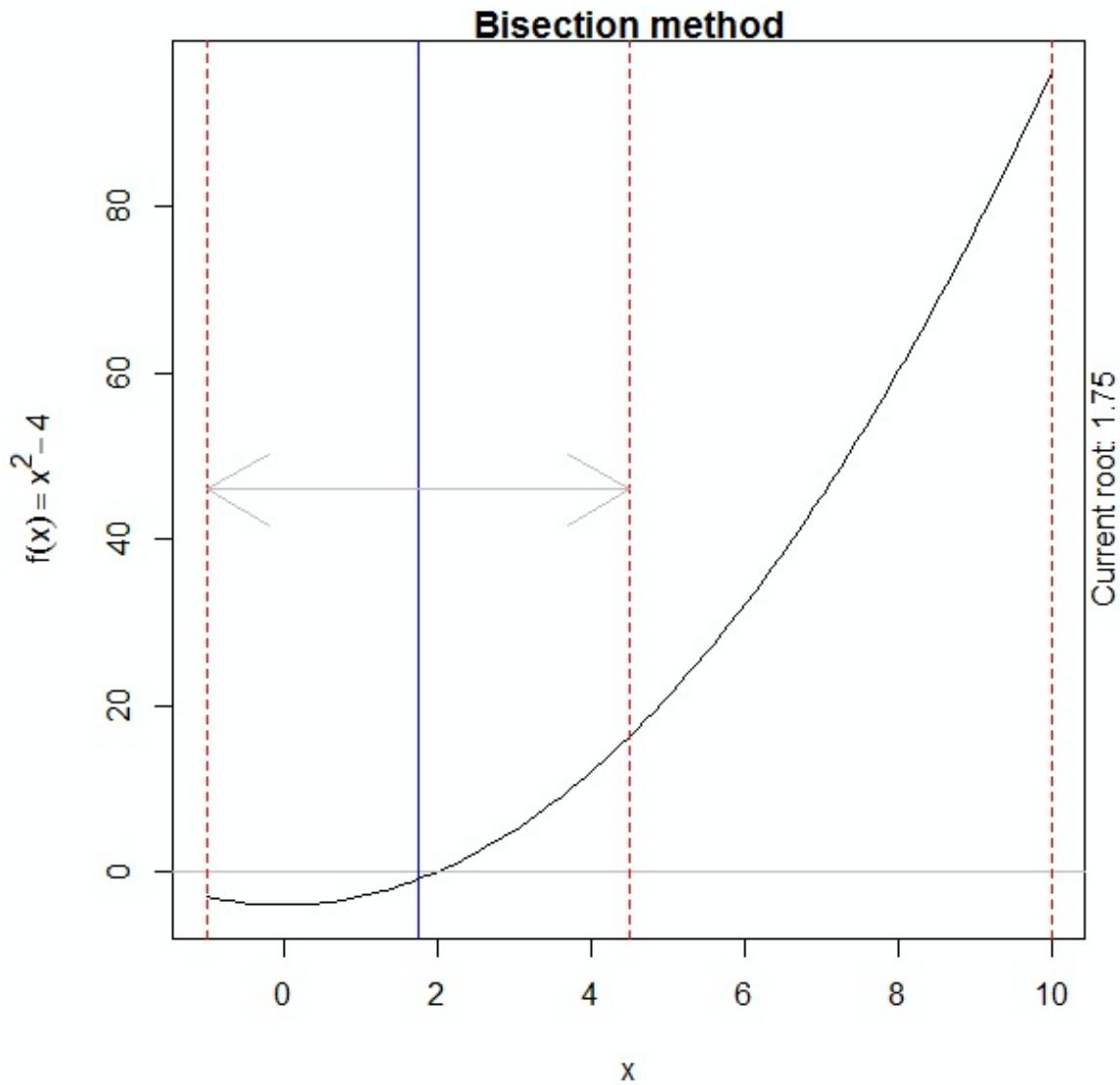
Brownian Motion



The next R program shows the logic behind the `Bisection method` dynamically:

```
library(animation)
par(mar = c(4, 4, 1, 2))
myFunction<-function(x) x^2 - 4
title<-"Bisection method"
bisection.method(FUN=myFunction,main =title)
```

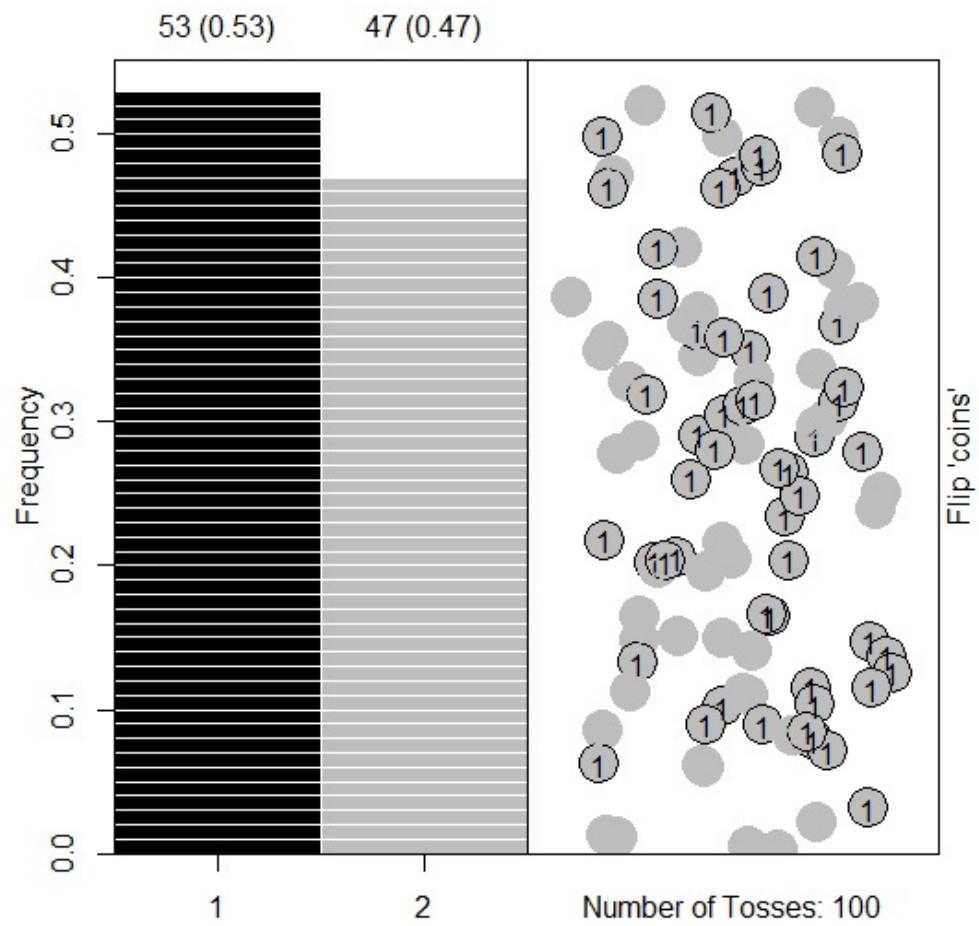
Since this is a dynamic picture, we can just show you a diagram. Note that we use *Ctrl + Shift + PrintScrn* to capture it:



We know that if we toss a coin, we could end up with either a head or a tail. The following R program shows the process:

```
library(animation)
myProb<-c(0.5,0.5)
ani.options(interval=0.2,nmax = ifelse(interactive(), 100, 2))
flip.coin(bg = "grey",col=c("black","grey"))
```

A diagram is shown here:



```
pdf("c:/temp/myGraph.pdf") plot(cos,-2*pi,2*pi) dev.off()
```

```
library(animation)

a<- c(3, 3, 1, 0.5)

b<- c(2, 0.5, 0)

part1<-"Random walk on the 2D plane: for each point"

part2<-"(x, y), x = x + rnorm(1) and y = y + rnorm(1)."

des<-c(part1,part2)

titleName<-"Demonstration of Brownian Motion"

#
```

saveHTML({ par(mar =a, mgp =b, tcl = -0.3,cex.axis = 0.8, cex.lab = 0.8, cex.main = 1) ani.options(interval = 0.05, nmax = ifelse(interactive(),150, 10)) brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow") },description =des,title = titleName)

```
library(animation)

a<- c(4, 4, 1, 2)
```

```
part1<-"The bisection method is a root-finding algorithm"

part2<-"which works by repeatedly dividing an interval in half and then"

part3<-"selecting the subinterval in which a root exists."

des<-c(part1,part2,part3)

titleName<-"The Bisection Method for Root-finding on an Interval"
```

```
#
```

saveHTML({ par(mar = a)

```
bisection.method(main = ""),img.name = "bisection.method",htmlfile =  
"bisection.method.html", ani.height = 400, ani.width = 600, interval = 1, title =  
titleName, description =des)
```

The website <http://canisius.edu/~yany/visual/bisection.shtml> is related to the preceding program, which generated a moving image.

Summary

In this chapter, we have discussed various types of visual presentation, which included simple graphs, bar charts, pie charts, and histograms written in different languages such as R, Python, and Julia. Visual presentations can help our audience understand our data better. For many complex concepts or theories, we can use visual presentations to help explain their logic and complexity. A typical example is the so-called bisection method or bisection search.

In the next chapter, we will explain many important issues related to statistics, such as the T-distribution, F-distribution, T-test, F-test, and other hypothesis tests. We will also discuss how to run a linear regression, how to deal with missing data, how to treat outliers, how to detect collinearity and its treatments, and how to run a multi-variable linear regression.

Review questions and exercises

1. Why should we care about data visualization?
2. Where can we find lists of R, Python, and Julia packages associated with data visualization?
3. Draw a graph using R and Python for the following formula:

$$y = 2 + 2.5x^{0.5}$$

4. Based on R programming, put the following two graphs together:

$$\begin{cases} y = 0.12 + 2\sqrt{x} \\ y = 2 + 5x \end{cases}$$

5. Download the R dataset related to the Fama-French monthly factor time series at <http://canisius.edu/~yany/RData/ff3monthly.RData>. Then, draw the histograms for these three factors: Market, SMB, and HML.
6. Write an R program to generate 1,000 random numbers from a uniform distribution. Then, estimate their mean and standard deviation. Finally, draw a histogram. Note that the R function for drawing n random numbers from a uniform distribution is `runif(n)`.
7. Repeat the previous exercise using Python and Julia.
8. Use Python to draw both the sine and cosine functions together.
9. From a beta distribution, generate 1,000 random numbers, estimate their mean and standard deviation, and draw a histogram. Note that the related R function is `rbeta(n)`.
10. From Yahoo!Finance (<http://finance.yahoo.com>), download the latest 5 years of monthly data for Walmart, with a ticker symbol of WMT. Generate a histogram from the monthly returns.
11. Find more information about the `HistData` R package. Show a few typical examples by using embedded functions.
12. Find all the embedded functions of the `dataswim` Python package. Show a few examples from the package.

13. Do the same things as in the previous two exercises for the `GraphPlot` Julia module.
14. Write an R program to show the following formula for the present value of annuity, where $PV(annuity)$ is the present value of annuity, R is the period rate, and n is the number of periods:

$$PV(annuity) = \frac{1}{R} \left[1 - \frac{1}{(1 + R)^n} \right]$$

15. Write a Python program to print the following formula, where $FV(growing annuity)$ is the future value of a growing annuity, R is the period rate, n is the number of periods, and g is the growth rate per period:

$$FV(growing annuity) = \frac{1}{R - g} [(1 + R)^n - (1 + g)^n]$$

16. For the famous Black-Scholes call option model, we have the following formula, where s is the underlying security price today, x is the exercise price, T is maturity day (in years), r is a continuously compounded risk-free rate, and σ is the volatility:

```
bsCall_<-function(s,x,T,r,sigma){

  d1 = (log(s/x)+(r+sigma*sigma/2.)*T)/(sigma*sqrt(T))

  d2 = d1-sigma*sqrt(T)

  call<-s*pnorm(d1)-x*exp(-r*T)*pnorm(d2)

  return(call)

}
```

For example, in the following set of values, the call option is 2.28, from `bsCall(40,42,0.5,0.1,0.2)`. Answer the following two questions:

- Graphically show that the call option price is an increasing function of volatility
- Dynamically show how to use the Bisection method to find the implied volatility

Note that the implied volatility is the volatility implied by the price of a call with the same set of input values, except the last one of sigma. For example, for $s=40$, $x=42$, $T=0.5$, $r=0.1$, and $c=3$, where c is the market call value we observed, what is the implied volatility?

17. Download the IBM, WMT, and MSFT historical monthly stock data from Yahoo!Finance at <http://finance.yahoo.com>. Estimate their correlation and show how strongly or weakly they are interconnected.
18. Find a way to save the following R program as HTML:

```
library(animation)

myProb<-c(0.45, 0.1, 0.45)

ani.options(interval=0.2, nmax = ifelse(interactive(), 100, 2))

flip.coin(bg = "yellow")
```

19. Find out how to save a JPG file in the R package called `animation`.
20. Find out how to run a stock price simulation in the R package called `animation`.

Statistical Modeling in Anaconda

In this chapter, we will first present the simplest statistical model: the one-factor linear model. To make the learning process more interesting, we will discuss an application of such a model: the famous financial model called the **Capital Asset Pricing Model (CAPM)**. In terms of processing data, we will show you how to detect and remove missing values, and how to replace missing values with means or other values in R, Python, or Julia. Also, **outliers** would distort our statistical results. Thus, we need to know how to detect and deal with them. After that, we talk about multi-factor linear models. Again, to make our discussion more meaningful, we will discuss the famous **Fama-French** 3-factor and 5-factor linear models, and the **Fama-French-Carhart** 4-factor linear model. Then, we will discuss how to rank those different models, that is, how to measure the performance of different models. In this chapter, the following topics will be covered:

- Introduction to linear models
- Running a linear regression in R, Python, Julia, and Octave
- Critical value and the decision rule
- F-test, critical value, and the decision rule
- Dealing with missing data
- Detecting outliers and treatments
- Several multivariate linear models
- Collinearity and its solution
- A model's performance measure

Introduction to linear models

The one-factor linear model is the simplest way to show a relationship between two variables: y and x . In other words, we try to use x to explain y . The general form for a one-factor linear model is given here, where y_t is the dependent variable at time t , α is the intercept, β is the slope, x_t is the value of an independent variable at time t , and ε_t is a random term:

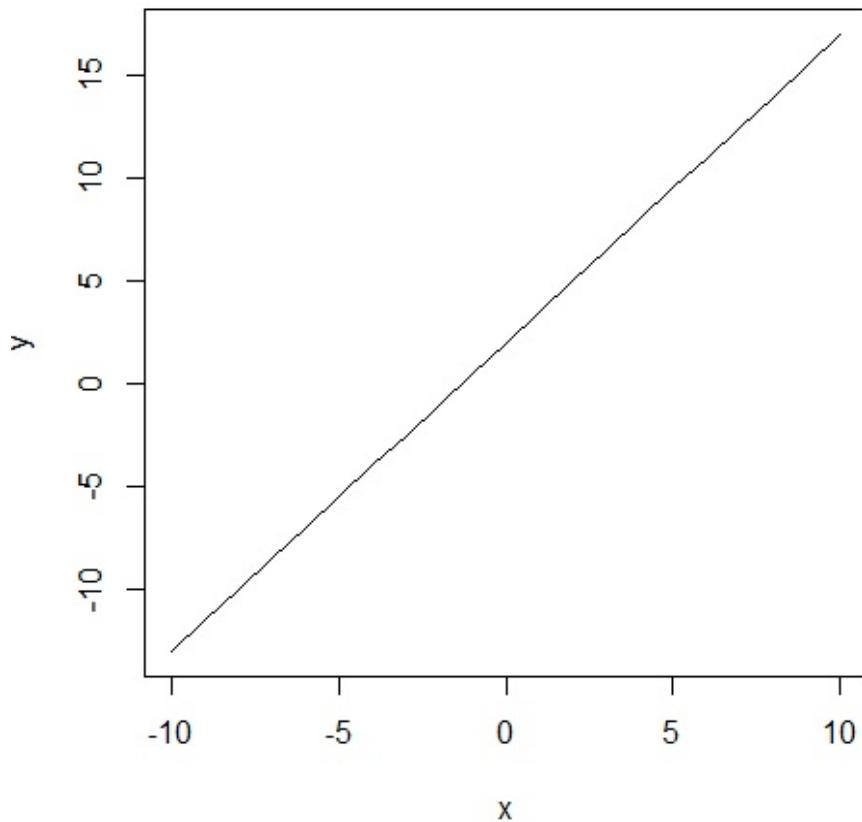
$$y_t = \alpha + \beta x_t + \varepsilon_t \quad (1)$$

To run a linear regression, we intend to estimate the intercept (α) and the slope (β). One-factor means that the model has just one explanatory variable, that is, one independent variable of x , and linear means that when drawing a graph based on the equation (1), we would have a straight line. With the following R program, we could get a linear line:

```
> x<-10:10
> y<-2+1.5*x
> title<-"A straight line"
> plot(x,y,type='l',main=title)
```

The related graph is shown here:

A straight line



Another explanation for a linear model is that the power of the independent variable of x is 1. There are many applications for such a simple one-factor linear model. Later in the chapter, we present the famous CAPM, which is a one-factor linear model.

Running a linear regression in R, Python, Julia, and Octave

The following code block shows how to run such a one-factor linear regression in R:

```
> set.seed(12345)
> x<-1:100
> a<-4
> beta<-5
> errorTerm<-rnorm(100)
> y<-a+beta*x+errorTerm
> lm(y~x)
```

The first line of `set.seed(12345)` guarantees that different users will get the same random numbers when the same `seed()` is applied, that is, `12345` in this case. The R function `rnorm(n)` is used to generate n random numbers from a standard normal distribution. Also, the two letters of the `lm()` function stand for linear model. The result is shown here:

```
Call: lm(formula = y ~ x)
Coefficients:
(Intercept) x
4.114 5.003
```

The estimated intercept is `4.11`, while the estimated slope is `5.00`. To get more information about the function, we can use the `summary()` function, shown in the following code:

```
> summary(lm(y~x))
Call:
lm(formula = y ~ x)

Residuals:
    Min 1Q Median 3Q Max
-2.6168 -0.8699 0.2071 0.6326 2.1451

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 4.114297 0.225258 18.27 <2e-16 ***
x           5.002592 0.003873 1291.81 <2e-16 ***
---
Signif. codes: 0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1

Residual standard error: 1.118 on 98 degrees of freedom
Multiple R-squared: 0.9999, Adjusted R-squared: 0.9999
F-statistic: 1.669e+06 on 1 and 98 DF, p-value: < 2.2e-16
```

The alpha, that is, the intercept, is 4.11 and it is statistically significant. The reason is that its T-value is 18.27, which is higher than 2 if we choose a 5% significance value. However, it is also significant at any significance level, indicated by the three stars: ***. Alternatively, we could look at its P-value to determine significance. With the same logic, the slope value is 5, which is statistically significant. Usually, we have the following two critical values for the T-value:

$$\begin{cases} \text{if } |T| > 2 \rightarrow \text{it is not a zero} \\ \text{if } |T| < 2 \rightarrow \text{it is the same as zero} \end{cases} \quad (2)$$

Alternatively, we have the following decision rule for *pValue*:

$$\begin{cases} \text{if } pValue < 0.05 \rightarrow \text{it is not a zero} \\ \text{if } pValue > 0.05 \rightarrow \text{it is a zero} \end{cases} \quad (3)$$

The following Python code would run a linear regression:

```
from scipy import stats
import scipy as sp
sp.random.seed(31233)
alpha=2.0
beta=3.8
n=1000
x=sp.arange(n)
y=alpha+beta*x+sp.random.rand(n)
(beta, alpha, r_value, p_value, std_err) = stats.linregress(y,x)

print("Alpha , Beta")
print(alpha,beta)
print("R-squared=", r_value**2)
print("p-value =", p_value)
```

Similarly, in the following code, for a set of *x* and *y* values, we could run a linear regression in Julia:

```
using GLM, DataFrames
data = DataFrame(X=[1,2,3], Y=[2,4,7])
OLS = glm(@formula(Y ~ X), data, Normal(), IdentityLink())
```

The related output is shown here:

```
julia> using GLM,DataFrames

julia> data = DataFrame(X=[1,2,3], Y=[2,4,7])
3x2 DataFrames.DataFrame
| Row | X | Y |
| 1   | 1 | 2 |
| 2   | 2 | 4 |
| 3   | 3 | 7 |

julia> OLS = glm(@formula(Y ~ X), data, Normal(), IdentityLink())
DataFrames.DataFrameRegressionModel(GLM.GeneralizedLinearModel(GLM.GlmResp{Array{Float64,1}},Distributions.Normal{Float64},GLM.IdentityLink),GLM.DensePredChol{Float64,Base.LinAlg.Cholesky{Float64,Array{Float64,2}}},Array{Float64,2})

Formula: Y ~ 1 + X

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.666667  0.62361 -1.06904  0.2850
X            2.5      0.288675  8.66025  <1e-17
```

Note that there is no meaning in x and y since their values are arbitrarily chosen. Based on the z values for both intercept and β , we can conclude that the intercept is statistically the same as zero since its z value is -1.09 and the slope is statistically different from zero since its z value is 8.66 . In the previous Julia program, we assume that the `GLM` package is installed. If not, see the following screenshot to install the package:



```
A fresh approach to technical computing
Documentation: https://docs.julialang.org
Type "?help" for help.

Version 0.6.1 (2017-10-24 22:15 UTC)
Official http://julialang.org/ release
i686-w64-mingw32

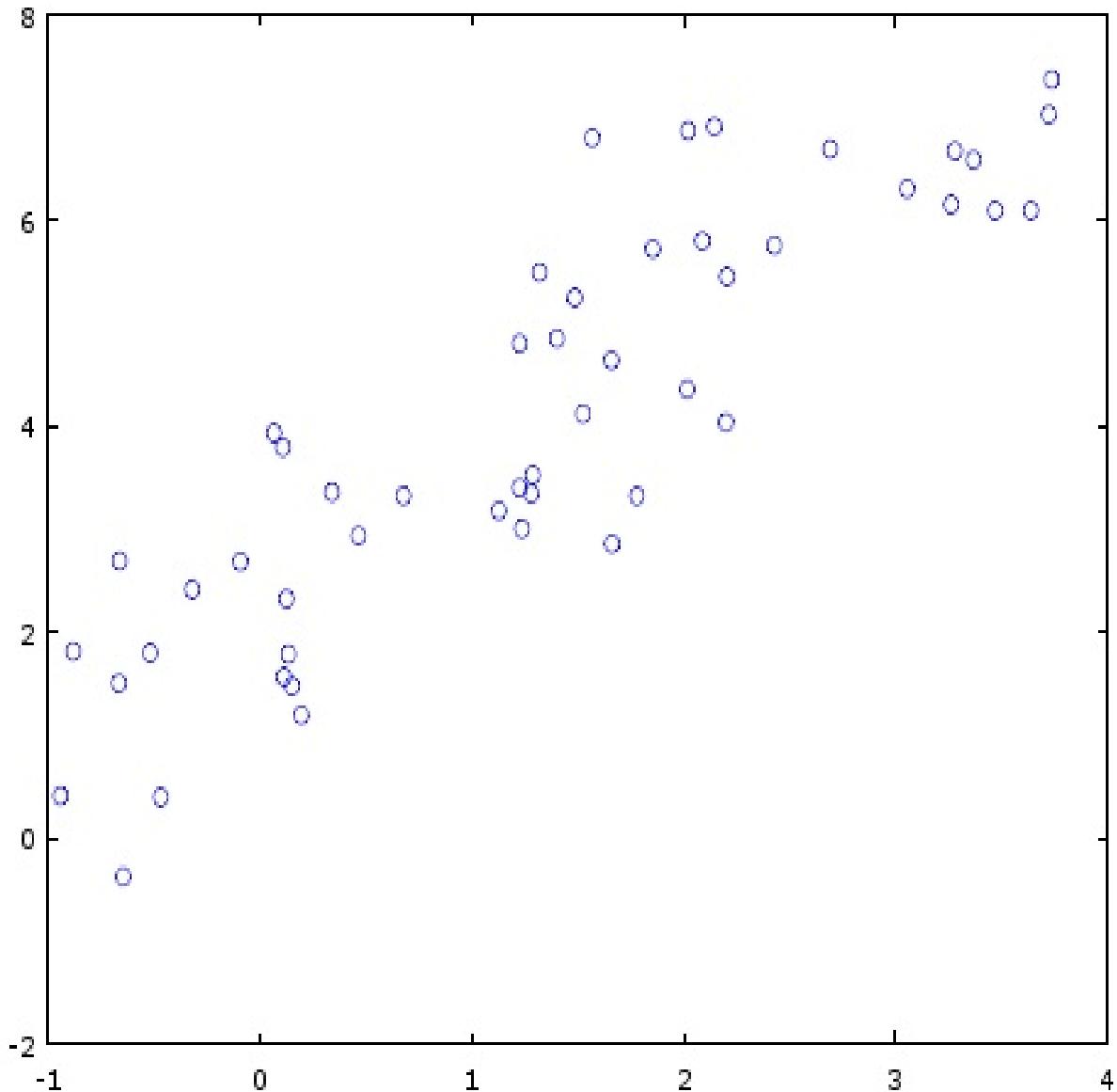
julia> Pkg.add("GLM")
INFO: Cloning cache of GLM from https://github.com/JuliaStats/GLM.jl.git
INFO: Installing GLM v0.8.1
INFO: Building SpecialFunctions
INFO: Building Rmath
INFO: Package database updated
INFO: METADATA is out-of-date - you may not have the latest version of GLM
INFO: Use 'Pkg.update()' to get the latest versions of your packages

julia> Pkg.update()
INFO: Updating METADATA...
```

In other words, if the `DataFrames` Julia package is not installed, we need to issue a command of `Pkg.add("DataFrames")`. For more information on handling various packages for R, Python, and Julia, see [chapter 6, *Managing Packages*](#). To run a linear regression for Octave, we generate `x` and `y` first:

```
rand('seed',123)
n = 50;
x = sort(rand(n,1)*5-1);
y = 2+1.5*x + randn(size(x));
figure % open a new figure window
plot(x, y, 'o');
```

The related graph is shown here:



Based on the previously generated x and y values, we have the following Octave program to run a linear regression:

```
n = length(y);
x2= [ones(n, 1), x]; % Add a column of ones to x
b = inv(x2'*x2)*x2'*y;
R = y - (x2 * b);      # residuals
v = (R'*R)/(4 - 3);   # residual variance
sigma = v * inv(x2'*x2); # variance covariance matrix
se = sqrt(diag(sigma)); # std errors of parameters
The related estimated parameters are shown below.
>> b
b =
    2.5259
    1.3585
```

```
>> sigma
sigma =
 1.01374 -0.37919
 -0.37919  0.29591

>> se
se =
 1.00685
 0.54398

>>
```

Critical value and the decision rule

A T-test is any statistical hypothesis test in which the test statistic follows a student's T-distribution under the null hypothesis. It can be used to determine whether two sets of data are significantly different from each other. For a one-sample T -test and for the null hypothesis that the mean is equal to a specified value μ_0 , we use the following statistic, where t is the T-value, \bar{R} is the sample mean, μ_0 is the our assumed return mean, σ is the sample standard deviation of the sample, n is the sample size, and $S.E.$ is the standard error:

$$\begin{cases} t = \frac{\bar{R} - \mu_0}{S.E} \\ S.E = \frac{\sigma}{\sqrt{n}} \end{cases} \quad (4)$$

The degrees of freedom used in this test are $n - 1$. The critical T -value is used to accept or reject the null hypothesis. The decision rule is given here:

$$\begin{cases} T > T_{critical} \text{ reject the null} \\ T < T_{critical} \text{ accept the null} \end{cases} \quad (5)$$

In the previous section, we mentioned two critical values: z for a T-value and 0.05 for a P-value. Actually, those two critical values are correlated. For a 5% critical P-value, its related critical T-value will be about z . The following function, `qt()`, shows this relationship:

```
|> qt(0.05/2, 50)
|[1] -2.008559
|> qt(1-0.05/2, 50)
|[1] 2.008559
```

The general function will be:

$$qt(1 - \frac{\alpha}{2}, degree_freedom) \quad (6)$$

where `qt()` is the student's T-distribution, α is the significance level, such as 1%, 5%, or 10%, *degree_freedom* is the degree of freedom ($n-p$) where n is the number of data points, and p is the number of repressors, that is, the number of independent variables. When we want a much stronger result, we can choose `0.01` as our α . Assume that from the code that follows, our degree of freedom is `50`; the corresponding critical T-value will be `2.68`:

```
|> alpha<-0.01  
|> degreeFreedom<-50  
|> qt(1-alpha/2,degreeFreedom)  
[1] 2.677793
```

The following Python code would give us the critical value for a two-sided test:

```
|> from scipy import stats  
|> alpha=0.05  
|> print(stats.t.ppf(1-alpha/2, 100))  
1.98397151845
```

F-test, critical value, and the decision rule

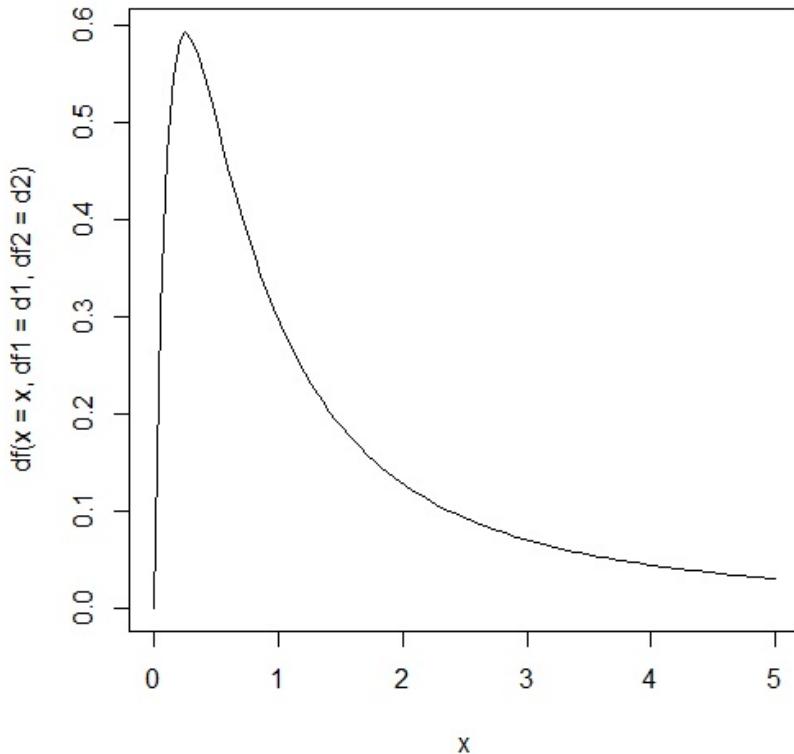
In the previous examples, we saw the F-value for the performance of the whole model. Now, let's look at the F-distribution. Assume that x_1 and x_2 are two independent random variables with the [chi-Square distribution](#) with df_1 and df_2 degrees of freedom, respectively. The ratio of x_1/df_1 divided by x_2/df_2 would follow an F-distribution:

$$F(n_1, n_2) = \frac{x_1/df_1}{x_2/df_2}$$

An R program to draw a graph for the F distribution with (10, 2) degrees of freedom is shown here:

```
> d1<-4
> d2<-2
> n<-100
> x = seq(0, 5, length = n)
> plot(x, df(x = x, df1 = d1, df2 = d2), type='l')
```

The related plot is shown here:



The following R program shows the critical value for a given α of 0.1 and (1, 2) degrees of freedom:

```
> alpha<-0.1
> d1<-1
> d2<-1
> qf(1-alpha, df1=d1, df2=d2)
[1] 39.86346
```

The following Python program estimates the critical F-value:

```
import scipy as sp
alpha=0.10
d1=1
d2=1
critical=sp.stats.f.ppf(q=1-alpha, dfn=d1, dfd=d2)
prob=sp.stats.f.cdf(critical, dfn=d1, dfd=d2)
print("alpha, d1, d2, critical value, prob")
print(alpha, d1, d2, critical, prob)
alpha, d1, d2, critical value, prob
(0.1, 1, 1, 39.86345818906144, 0.9000000000000002)
```

The related decision rule is that if our F-value is higher than the critical value, we would conclude that our overall model is significant.

An application of a linear regression in finance

In finance, a famous application of a one-factor linear model is associated with CAPM, which was developed in the 1960s; see *Sharpe* (1964), *Litner* (1965), and *Mossin* (1966). Up until today, CAPM has been discussed in almost all finance textbooks because of its simplicity and usefulness. Its general formula is shown here:

$$E(R_i) = R_f + \beta_i(E(R_{mkt}) - R_f) \quad (8)$$

where $E(\cdot)$ is the expectation, R_i is return for stock i , R_f is the risk-free interest rate, and $E(R_{mkt})$ is the expected market return. Usually, we could use the S&P500 index as our market index. The slope of the equation, that is β , is a measure of the market risk. For individual stock, β represents the sensitivity of a stock's expected returns in response to the market risk premium, that is, $E(R_{mkt}) - R_f$. A risk premium represents the required extra return for bearing extra risk when compared with the risk-free rate. The following formula could be used to estimate β , where β_i is the beta for stock i , $cov(R_i, R_m)$ is the covariance between the returns of stock i , and the returns of a market index, ρ , is the correlation between stock i and the market index. σ_i is the standard deviation of stock i and σ_m is the standard deviation of the market index:

$$\beta_i = \frac{cov(R_i - R_m)}{\sigma_m^2} = \frac{\rho * \sigma_i}{\sigma_m} \quad (9)$$

For the beta value, we could run a linear regression for CAPM by using historical data. There is another variable of the previous formula shown next, where $R_{i,t}$ is stock i 's return at time t , $R_{f,t}$ is the risk-free interest rate at time t , and $R_{mkt,t}$ is the market return at time t while ϵ_t is a random factor:

$$R_{i,t} - R_{f,t} = \alpha + \beta_i(R_{mkt,t} - R_{f,t}) + \varepsilon_t \quad (10)$$

The previous equation associated with CAPM can be applied to estimate the expected cost of equity, that is, the expected stock return, after we estimate the beta successfully. In the preceding variation, one added advantage is that the logic of CAPM is much clearer. On the left-hand side, we have an individual stock's risk premium while on the right-hand side, we have the market risk premium, shown in the following:

$$\underbrace{R_{i,t} - R_{f,t}}_{\substack{\text{individual stock} \\ \text{risk premium}}} = \alpha + \underbrace{\beta_i}_{\substack{\text{individual stock} \\ \text{market risk}}} * \underbrace{(R_{mkt,t} - R_{f,t})}_{\substack{\text{market risk} \\ \text{premium}}} + \epsilon_t \quad (11)$$

Later in the chapter, we will show you how to run CAPM by using historical monthly stock data and the *S&P500* data as the market index; programs written in R, Python, and Julia will be available. Using R, we could estimate a stock's market risk, that is, the beta value. First, let's write a function called `ret_f()` to estimate returns for a given input price dataset:

```
ret_f<-function(data){
  ddate<-as.Date(data[,1])
  n<-nrow(data)
  p<-data[,6]
  ret<-p[2:n]/p[1:(n-1)]-1
  final<-data.frame(ddate[2:n],ret,stringsAsFactors=F)
  colnames(final)<-c("DATE","RET")
  return(final)
}
#
x<-read.csv("http://canisius.edu/~yany/data/ibmMonthly.csv")
stock<-ret_f(x)
y<-read.csv("http://canisius.edu/~yany/data/^gspcMonthly.csv")
mkt<-ret_f(y)
colnames(mkt)<-c("DATE","MKTRET")
final<-merge(stock,mkt)
```

In the previous R program, we define a function called `ret_f()`. Then, we call this function twice to estimate returns for the download IBM's price data and S&P500 monthly data. Again, these two time series can be manually downloaded from Yahoo!Finance at <http://finance.yahoo.com>. The first several lines of the final dataset are shown here:

```
> head(final)
  DATE RET MKTRET
1 1962-02-01 -0.009225374 0.016269655
2 1962-03-01 -0.007779475 -0.005860435
3 1962-04-01 -0.147816831 -0.061969875
4 1962-05-01 -0.135463769 -0.085990147
5 1962-06-01 -0.135531661 -0.081838016
```

```
| 6 1962-07-01 0.140751225 0.063561644
```

Now, we can apply the `lm()` function, shown as follows, with the corresponding output:

```
> summary(lm(final$RET~final$MKTRET))

Call:
lm(formula = final$RET ~ final$MKTRET)

Residuals:
    Min      1Q  Median      3Q     Max 
-0.261101 -0.031148 -0.002537  0.029451  0.281473 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.00277   0.00217   1.277   0.202    
final$MKTRET  0.96442   0.05060  19.061  <2e-16 ***  
---
Signif. codes:  0 '****' 0.001 '***' 0.01 '**' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0555 on 667 degrees of freedom
Multiple R-squared:  0.3526,    Adjusted R-squared:  0.3517 
F-statistic: 363.3 on 1 and 667 DF,  p-value: < 2.2e-16
```

Based on the result, the intercept is `0.00277` and it is not statistically significant. The slope (or the beta value) is `0.96`, which is statically significant. R^2 is 35%, which means that the model could explain 35% of the variation in the dependent variable. The F-statistic is a measure of goodness of fit of the whole model, which is statistically significant since its P-value is extremely small. Note that for the previous beta estimation, we use the whole input data set, that is, all years, while in real-world market risk (β) estimation, researchers or professionals working on Wall Street usually use only a few years' data, such as a 3-year window. Before writing a Python program to estimate a stock's beta, first we have to retrieve stock data from `quandl`, shown in the program here:

```
import quandl as qd
x=qd.get("WIKI/ibm")
print(x.head(2))
print(x.tail(2))
```

The related output is shown in the following screenshot:

```
In [1]: %run "c:\users\yany\appdata\local\temp\tmphevjxh.py"
          Open   High    Low  Close   Volume Ex-Dividend Split Ratio \
Date
1962-01-02  578.5  578.5  572.0  572.0  19360.0            0.0      1.0
1962-01-03  572.0  577.0  572.0  577.0  14400.0            0.0      1.0

          Adj. Open  Adj. High  Adj. Low  Adj. Close  Adj. Volume
Date
1962-01-02  15.270839 15.270839 15.099257 15.099257 387200.0
1962-01-03  15.099257 15.231243 15.099257 15.231243 288000.0
          Open   High    Low  Close   Volume Ex-Dividend \
Date
2018-02-06  150.29  155.49  149.11  155.34  9823824.0            0.0
2018-02-07  154.17  155.34  153.28  153.85  6069258.0            0.0

          Split Ratio  Adj. Open  Adj. High  Adj. Low  Adj. Close \
Date
2018-02-06           1.0     150.29     155.49     149.11     155.34
2018-02-07           1.0     154.17     155.34     153.28     153.85 |

          Adj. Volume
Date
2018-02-06    9823824.0
2018-02-07    6069258.0
```

From the previous output, we know that the frequency of data is daily. For monthly data, we have the following Python code:

```
| import quandl as qd
| x=qd.get("WIKI/ibm",collapse='monthly')
```

quandl is a data delivery platform which includes many free datasets, but users might need to register to get specific datasets (for a more detailed discussion, see [Chapter 3, Data Basics](#)). The following Python program downloads IBM's monthly data from quandl and the monthly S&P500 market index data from the author's website, and runs a linear regression to get the market risk, that is, β for IBM:

```
| import quandl as qd
| import pandas as pd
| from scipy import stats
| x=qd.get("WIKI/ibm",collapse='monthly')
|
| p=x[['Adj. Close']]
```

```

ret=p.diff()/p.shift(1)
stockRet=ret.dropna()
stockRet.columns=['stockRet']
#stockRet.assign(yyyymm=stockRet.index.strftime("%Y%m"))
#
inFile="http://canisius.edu/~yany/data/sp500monthlyEndOfMonthDate.csv"
y=pd.read_csv(inFile,index_col=0)
d=y[['Adj.Close']]
ret2=d.diff()/d.shift(1)
mktRet=ret2.dropna()
mktRet.columns=['mktRet']
df= stockRet.merge(mktRet, how='inner', left_index=True, right_index=True)
(beta,alpha,r_value,p_value,std_err)=stats.linregress(df.stockRet,df.mktRet)
alpha=round(alpha,8)
beta=round(beta,3)
r_value=round(r_value,3)
p_value=round(p_value,3)
print("alpha, beta, R2 and P-value")
print(alpha,beta,r_value,p_value)

```

The related output is shown here:

```

| alpha, beta, R2 and P-value
| (0.00408539, 0.322, 0.561, 3.8213963635856179e-38)

```

By using daily data, the next Python program estimates the annual betas for Walmart, with a ticker symbol of WMT using quandl to download data:

```

import scipy as sp
import pandas as pd
import quandl as qd
from scipy import stats
#
ticker="wmt"
x=qd.get("WIKI/"+ticker)
p=x[['Adj. Close']]
ret=p.diff()/p.shift(1)
stockRet=ret.dropna()
stockRet.columns=['stockRet']
#
inFile="http://canisius.edu/~yany/data/^gspcDaily.csv"
y=pd.read_csv(inFile,index_col=0)
d=y[['Adj Close']]
ret2=d.diff()/d.shift(1)
mktRet=ret2.dropna()
mktRet.columns=['mktRet']

final= stockRet.merge(mktRet, how='inner', left_index=True, right_index=True)
years=pd.unique(final.index.strftime("%Y"))
print(" year, alpha, beta,R_value, P_value")
for i in sp.arange(0,5):
#for i in sp.arange(1,len(years)):
#print(years[i])
d=final[final.index.strftime("%Y")==years[i]]                                (beta,alpha,r_
alpha=round(alpha,8)
beta=round(beta,3)
r_value=round(r_value,3)
p_value=round(p_value,3)
print(years[i],alpha,beta,r_value,p_value)

```

The related output is shown here:

```
year,      alpha,    beta,R_value, P_value
(u'1972',  0.00073605, 0.054, 0.19,  0.081888902299272023)
(u'1973', -0.00040998, 0.092, 0.3,   1.2421671282048366e-06)
(u'1974', -0.00120897, 0.11,  0.261,  2.5687411810107753e-05)
(u'1975',  0.00080998, 0.072, 0.22,  0.00042128685673863711)
(u'1976',  0.00064678, 0.076, 0.253, 4.5885040919840052e-05)
```

In the following Julia program, we assume that two files are downloaded from Yahoo!Finance at <http://finance.yahoo.com>. Those input files are `ibmMonthly5years.csv` and `sp500Monthly5years.csv`, and they are the latest 5-year monthly historical data. The Julia program is given here:

```
using DataFrames, StatsBase, StatsModels, GLM
x = readtable("c:/temp/ibmMonthly5years.txt")
#x = CSV.read("c:/temp/ibmMonthly5years.csv")
p=x[:Adj_Close]
n=length(p)
stockRet=p[2:n]./p[1:(n-1)]-1
y = readtable("c:/temp/sp500Monthly5years.txt")
#y = CSV.read("c:/temp/sp500Monthly5years.csv")
p2=y[:Adj_Close]
n2=length(p2)
mktRet=p2[2:n2]./p2[1:(n2-1)]-1
n3=min(length(stockRet),length(mktRet))
data = DataFrame(X=mktRet[1:n3], Y=stockRet[1:n3])
OLS = glm(@formula(Y ~ X), data, Normal(), IdentityLink())
```

Note that we manually removed the first observation since it contained a non-numerical value. The related output is shown here:

```
julia> OLS = glm(@formula(Y ~ X), data, Normal(), IdentityLink())
DataFrames.DataFrameRegressionModel{GLM.GeneralizedLinearModel{GLM.GlmResp{Array{Float64,1}}, Distributions.Normal{Float64}, GLM.IdentityLink}, GLM.DensePredChol{Float64, Base.LinAlg.Cholesky{Float64, Array{Float64,2}}}}, Array{Float64,2}>

Formula: Y ~ 1 + X

Coefficients:
              Estimate Std.Error z value Pr(>|z|)
(Intercept)  0.00617165 0.00863561 0.714674  0.4748
X            0.145727  0.282945  0.515036  0.6065
```

Dealing with missing data

There are many ways to deal with missing records. The simplest one is to delete them. This is especially true when we have a relative large dataset. One potential issue is that our final dataset should not be changed in any fundamental way after we delete the missing data. In other words, if the missing records happened in a random way, then simply deleting them would not generate a biased result.

```
> x<-c(NA,1,2,50,NA) > y<-na.omit(x) > mean(x)
```

```
<strong>[1] NA</strong> > mean(y)
```

```
<strong>[1] 17.66667</strong>
```

```
import scipy as sp x={2,4,3,sp.nan,6,sp.nan,7}
```

```
print(x)
```

```
x.remove(sp.nan) print(x)
```

For brevity, we omitted the output.

```
> x<-c(NA,2,3,4,NA)  
> y<-na.omit(x)  
> m<-mean(y)  
> m  
[1] 3
```

```
<strong>> x[is.na(x)]<-m</strong>  
> x
```

```
<strong>[1] 3 2 3 4 3</strong>
```

```
>
```

```
import scipy as sp
```

```
import pandas as pd
```

```
df = pd.DataFrame({'A' : [2,sp.nan,3,4]})
```

```
print(df)
```

```
df.fillna(df.mean(), inplace=True)
```

```
print(df)
```

```
<strong> A
```

```
0 2.0
```

1 NaN

```
2 3.0
```

```
3 4.0
```

A

0 2.0

1 3.0

2 3.0

3 4.0

Detecting outliers and treatments

First, a word of caution: one person's waste might be another person's treasure, and this is true for outliers. For example, for the week of 2/5/2018 to 2/15/2018, the **Dow Jones Industrial Average (DJIA)** suffers a huge loss. Cheng and Hum (2018) show that the index travels more than 22,000 points, as shown in the following table:

Weekday	Points
Monday	5,113
Tuesday	5,460
Wednesday	2,886
Thursday	3,369
Friday	5,425
Total	22,253

Table 5.1 Dow Jones industrial average points traveled

If we want to study the relationship between a stock and the DJIA index, the observations might be treated as outliers. However, when studying the topic related to the impact of the market on individual stocks, we should pay special attention to those observations. In other words, those observations should not be treated as outliers.

There are many different definitions of an outlier:

- First, for a given dataset, an outlier is a data point, or an observation, that is located an abnormal distance from other observations
- Second, if removing an observation results in a metrical change for a regression, then this observation will be an outlier
- Third, the distance between an outlier and the mean is at least three standard deviations

Assume that we have download the weekly S&P500 historical data from Yahoo!Finance at <https://finance.yahoo.com/>. The ticker symbol for the S&P500 market index is `^GSPC`. Assume further that the dataset is saved under `c:/temp` with a name of `^GSPCweekly.csv`. The following R program shows the number of cases satisfying the following condition: n standard deviations from their mean. In the program, we assign `n` a value of 3:

```
> distance<-3
> x<-read.csv("c:/temp/^GSPCweekly.csv")
> p<-x$Adj.Close
> ret<-p[2:n]/p[1:(n-1)]-1
> m<-mean(ret)
> std<-sd(ret)
> ret2<-subset(ret,((ret-m)/std)>distance)
> n2<-length(ret2)
```

It is a good idea to show a few output results:

```
> head(x,2)
  Date Open High Low Close Adj.Close Volume
1 1950-01-02 16.66 17.09 16.66 17.09 17.09 9040000
2 1950-01-09 17.08 17.09 16.65 16.65 16.65 14790000
> m
[1] 0.001628357
> std
[1] 0.02051384
> length(ret)
[1] 3554
> n2
[1] 15
```

Among 3554 weekly returns, 15 of them could be treated as outliers if defined as at least three standard deviations from the mean. Of course, users could use other ways to define an outlier. How to treat those outliers depends on the research topic. One way is to delete them, but the most important reminder is that researchers should detail their methods of treating outliers.

Several multivariate linear models

As we mentioned at the beginning of the chapter, we could show several applications of multivariable linear models. The first one is a three-factor linear model. The general formula is quite similar to the one-factor linear model, shown here:

$$y_t = \alpha + \beta_1 x_{1,t} + \beta_2 x_{2,t} + \beta_3 x_{3,t} + \varepsilon_3 \quad (12)$$

The definitions are the same as before. The only difference is that we have three independent variables instead of one. Our objective is to estimate four parameters, one intercept plus three coefficients:

$$R_i = R_f + \beta_{mkt}(R_m - R_f) + \beta_{SMB} * SMB + \beta_{HML} * HML \quad (13)$$

For example, the equation of the famous Fama-French 3-factor model is given, where R_i is the stock i's return and R_m is the market return. SMB (Small Minus Big) is defined as the returns of the small portfolios minus the returns of the big portfolios and HML (High Minus Low) is the difference of returns of high book-to-market portfolios minus the returns of low book-to-market portfolios. (See the Ken French data library at http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html for more detailed definitions.) To download these three factors, we could go to Prof. French's data library to download them, as shown in the following steps:

1. Go to the previous link.
2. Choose CSV for the monthly data (the first line in the following screenshot):

Fama/French 3 Factors [TXT](#) [CSV](#) [Details](#)
Fama/French 3 Factors [Weekly] [TXT](#) [CSV](#) [Details](#)
Fama/French 3 Factors [Daily] [TXT](#) [CSV](#) [Details](#)

The first several lines of the unzipped csv file are shown here:

| This file was created by CMPT_ME_BEME_RET using the 201712 CRSP database.

```
The 1-month TBill return is from Ibbotson and Associates, Inc.
```

```
,Mkt-RF,SMB,HML,RF  
192607, 2.96, -2.30, -2.87, 0.22  
192608, 2.64, -1.40, 4.19, 0.25  
192609, 0.36, -1.32, 0.01, 0.23  
192610, -3.24, 0.04, 0.51, 0.32  
192611, 2.53, -0.20, -0.35, 0.31  
192612, 2.62, -0.04, -0.02, 0.28  
192701, -0.06, -0.56, 4.83, 0.25  
192702, 4.18, -0.10, 3.17, 0.26  
192703, 0.13, -1.60, -2.67, 0.30  
192704, 0.46, 0.43, 0.60, 0.25
```

This file was created by CMPT_ME_BEME_RETs using the 201712 CRSP database:

```
The 1-month TBill return is from Ibbotson and Associates, Inc.
```

```
,Mkt-RF,SMB,HML,RF  
192607, 2.96, -2.30, -2.87, 0.22  
192608, 2.64, -1.40, 4.19, 0.25  
192609, 0.36, -1.32, 0.01, 0.23  
192610, -3.24, 0.04, 0.51, 0.32  
192611, 2.53, -0.20, -0.35, 0.31  
192612, 2.62, -0.04, -0.02, 0.28  
192701, -0.06, -0.56, 4.83, 0.25  
192702, 4.18, -0.10, 3.17, 0.26  
192703, 0.13, -1.60, -2.67, 0.30  
192704, 0.46, 0.43, 0.60, 0.25
```

In R, we could issue the following code to download it from the author's website:

```
> con<-url("http://canisius.edu/~yany/RData/ff3monthly.RData")  
> load(con)  
> head(.ff3monthly)  
    DATE MKT_RF SMB HML RF  
1 1926-07-01 0.0296 -0.0230 -0.0287 0.0022  
2 1926-08-01 0.0264 -0.0140 0.0419 0.0025  
3 1926-09-01 0.0036 -0.0132 0.0001 0.0023  
4 1926-10-01 -0.0324 0.0004 0.0051 0.0032  
5 1926-11-01 0.0253 -0.0020 -0.0035 0.0031  
6 1926-12-01 0.0262 -0.0004 -0.0002 0.0028
```

For the following R program, we can run the Fama-French 3-factor model for IBM:

```
con<-url("http://canisius.edu/~yany/RData/ff3monthly.RData")  
load(con)  
head(.ff3monthly)  
x<-read.csv("http://canisius.edu/~yany/data/ibmMonthly.csv")  
stock<-ret_f(x)  
final<-merge(stock,.ff3monthly)  
y<-final$RET  
x<-as.matrix(data.frame(final[,3:5]))  
summary(lm(y~x))
```

In the previous code, we assume that the function called `ret_f()` is available (see the previous discussion about this function). The output is shown here:

```
> summary(lm(y~x))

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max 
-0.268710 -0.032789 -0.001212  0.029722  0.282335 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.005664   0.002216   2.557 0.010794 *  
xMKT_RF      0.888114   0.052521  16.910 < 2e-16 *** 
xSMB        -0.179172   0.074995  -2.389 0.017168 *  
xHML        -0.278756   0.081042  -3.440 0.000619 *** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 

Residual standard error: 0.05589 on 660 degrees of freedom
Multiple R-squared:  0.3469,    Adjusted R-squared:  0.344 
F-statistic: 116.9 on 3 and 660 DF,  p-value: < 2.2e-16
```

Here is the Fama-French-Carhart's 4-factor model:

$$R_i = R_f + \beta_{mkt}(R_m - R_f) + \beta_{SMB} * SMB + \beta_{HML} * HML + \beta_{MOM} * MOM \quad (14)$$

To download these four factors (Market, SMB, HML, and Momentum), we go to Professor Ken French's data library to download two zipped files:

```
| con<-url("http://canisius.edu/~yany/RData/fffc4monthly.RData")
| load(con)
| head(.fffc4monthly)
```

The related output is shown here:

	DATE	MKT_RF	SMB	HML	MOM	RF
1	1927-01-31	-0.0006	-0.0056	0.0483	0.0044	0.0025
2	1927-02-28	0.0418	-0.0010	0.0317	-0.0201	0.0026
3	1927-03-31	0.0013	-0.0160	-0.0267	0.0359	0.0030

4	1927-04-30	0.0046	0.0043	0.0060	0.0419	0.0025
5	1927-05-31	0.0544	0.0141	0.0493	0.0301	0.0030
6	1927-06-30	-0.0234	0.0047	-0.0153	0.0051	0.0026

To save space, we would not run a Fama-French-Carhart 4-factor model since it is quite similar to running the CAPM and Fama-French 3-factor model.

In 2014, Fama and French developed their 5-factor model, which has the following form:

$$R_i - R_{t,t} = \alpha_i + \beta_1(R_{m,t} - R_{f,t}) + \beta_2SMB_t + \beta_3HML_t + \beta_4RMW_t + \beta_5CMA_t \quad (15)$$

In the equation, RMW_t is the return difference between portfolios of stocks with robust and weak profitability, and CMA_t is the return difference between portfolios of low and high investment stocks. Fama and French call the low and high investment stocks conservative and aggressive, respectively. If the sensitivities to the five factors, β_i ($i=1,2,\dots,5$), capture all the variations in the expected returns, the intercept α_i will be zero for all securities and portfolios i .

The following R program downloads the `ff5Monthly.RData` set from the author's website:

```
> con<-url("http://canisius.edu/~yany/RData/ff5Monthly.RData")
> load(con)
> head(.ff5Monthly)
```

The related output is shown here:

	DATE	MKT_RF	SMB	HML	RMW	CMA	RF
1	1963-07-01	-0.0039	-0.0046	-0.0082	0.0072	-0.0116	0.0027
2	1963-08-01	0.0507	-0.0081	0.0163	0.0042	-0.0040	0.0025
3	1963-09-01	-0.0157	-0.0048	0.0019	-0.0080	0.0023	0.0027
4	1963-10-01	0.0253	-0.0129	-0.0011	0.0275	-0.0226	0.0029
5	1963-11-01	-0.0085	-0.0084	0.0166	-0.0034	0.0222	0.0027
6	1963-12-01	0.0183	-0.0187	-0.0011	0.0018	-0.0031	0.0029

Collinearity and its solution

In statistics, multicollinearity, or collinearity, is a phenomenon in which one independent variable (predictor variable) in a multiple regression model can be linearly predicted from the others with a substantial degree of accuracy.

Collinearity tends to inflate the variance of at least one estimated regression coefficient. This could cause some regression coefficients to have the wrong sign. Those issues would make our regression results unreliable. Therefore, how can we detect the potential problem? One way is that we could simply look at the correlation between each pair of independent variables. If their correlation is close to ± 1 , then we might have such an issue:

```
>con<-url("http://canisius.edu/~yany/RData/ff3monthly.RData")
>load(con)
> head(.ff3monthly)
  DATE  MKT_RF     SMB     HML      RF
1 1926-07-01  0.0296 -0.0230 -0.0287 0.0022
2 1926-08-01  0.0264 -0.0140  0.0419 0.0025
3 1926-09-01  0.0036 -0.0132  0.0001 0.0023
4 1926-10-01 -0.0324  0.0004  0.0051 0.0032
5 1926-11-01  0.0253 -0.0020 -0.0035 0.0031
6 1926-12-01  0.0262 -0.0004 -0.0002 0.0028
```

To find out the correlation between each pair of portfolios, we could use the R function called `cor()` to generate a correlation matrix, shown here:

```
> cor(.ff3monthly[,2:5])
      MKT_RF        SMB        HML        RF
MKT_RF  1.00000000  0.31850074  0.23995229 -0.06543067
SMB     0.31850074  1.00000000  0.12240744 -0.05063652
HML     0.23995229  0.12240744  1.00000000  0.02117111
RF      -0.06543067 -0.05063652  0.02117111  1.00000000
```

According to *Martz* (2013), we have the following warning signs for a collinearity problem:

#	Description
1	A coefficient is not significant, even though it should be

2	After adding/deleting an x variable, the regression coefficients change dramatically
3	A negative coefficient when the opposite should be true
4	A positive coefficient when the opposite should be true
5	Input variables have high pairwise correlations

Table 5.2 Warning signs of collinearity

There are many ways to treat the issue. The simplest one is to delete one of the input variables that is highly correlated with another input variable. For example, if two input variables are strongly correlated, we could drop one of them. The second way is to use the residuals after running two variables against each other.

A model's performance measure

In this chapter, we looked at several applications of linear models, including CAPM, the Fama-French 3-factor linear model, the Fama-French-Carhart 4-factor linear model, and the Fama-French 5-factor linear model. Obviously, CAPM is the simplest one since it only involves a market index as the explanatory variable. One question remains though: which model is the best? In other words, how do we rank these models and how is their performance measured? When running linear regressions, the output will show both the R^2 and adjusted R^2 . When comparing models with different numbers of independent variables, the adjusted R^2 is a better measure since it is adjusted by the number of input variables. However, note we should not depend only on the adjusted R^2 since this is in the sample measure. In other words, a higher adjusted R^2 simply means that based on this sample dataset, model A is better than model B. Hence, a much better measure is to use the performance of the out-of-sample prediction.

Summary

In this chapter, we have explained many important issues related to statistics, such as T-distribution, F-distribution, T-tests, F-tests, and other hypothesis tests. We have also discussed linear regression, how to deal with missing data, how to treat outliers, collinearity and its treatments, and how to run a multi-variable linear regression.

In [Chapter 6, Managing Packages](#), we will discuss the importance of managing package; how to find out about all packages available for R, Python, and Julia; and how to find the manual for each package. In addition, we will discuss the issue of package dependency and how to make our programming a little easier when dealing with packages.

Review questions and exercises

1. What is the definition of a single-factor linear model?
2. How many independent variables are there in a single-factor model?
3. What does it mean for something to be statically different from zero?
4. What are the critical T-values and P-values to tell whether an estimate is statistically significant?
5. When the significant level is 1%, what is the critical T-value when there are 30 degrees of freedom?

6. What is the difference between a one-sided test and a two-sided test?
7. What are the corresponding missing codes for missing data items in R, Python, and Julia?
8. How do we treat missing variables if our sample is big? How about if our sample is small?
9. How do we generally detect outliers and deal with them?
10. How do we generate a correlated return series? For example, write an R program to generate 5-year monthly returns for two stocks with a fixed correlation of 0.5.
11. In terms of the critical T-value, is this the same for a one-sided test and a two-sided test when the significance level is fixed?
12. What is collinearity? What might be damaged if our dependents variables are strongly correlated?
13. How do we detect the multicollinearity? What are the functions to estimate the correlation matrix in R, Python, and Julia?
14. Write an R program to estimate Microsoft's annual β s over the last 10 years by using daily data. (Source of data: <http://finance.yahoo.com>.)
15. Based on the following code, we could download monthly data. Find out whether the last day of each week is a trading day or not (hint: you could use an R dataset downloadable at <http://canisius.edu/~yany/RData/tradingDaysM.RData>):

```
| import quandl as qd  
| x=qd.get("WIKI/ibm",collapse='monthly')
```

16. Download the monthly price data for Walmart from the Quandl platform and S&P500 data from Yahoo!Finance. Estimate Walmart's β by using monthly data from the last 5 years. A few lines of sample code are given here:

```
import pandas as pd
import quandl as qd
x=qd.get("WIKI/ibm",collapse='monthly')
inFile="http://canisius.edu/~yany/data/^gspcMonthly.csv.csv"
y=pd.read_csv(inFile,index_col=0)
```

17. Write an R program by using the monthly data over the last 10 years for IBM and run a Fama-French-Carhart 4-factor model. Then, compare it with CAPM. Which model is better?
18. Generate a pickle dataset similar to the following R dataset:

```
>con<-url("http://canisius.edu/~yany/RData/ff3monthly.RData")
>load(con)
> head(.ff3monthly)
  DATE  MKT_RF      SMB      HML      RF
1 1926-07-01  0.0296 -0.0230 -0.0287  0.0022
2 1926-08-01  0.0264 -0.0140  0.0419  0.0025
3 1926-09-01  0.0036 -0.0132  0.0001  0.0023
4 1926-10-01 -0.0324  0.0004  0.0051  0.0032
5 1926-11-01  0.0253 -0.0020 -0.0035  0.0031
6 1926-12-01  0.0262 -0.0004 -0.0002  0.0028
```

19. When estimating β , we could have the following formula, where β_i is the beta for stock i , R_i is the return for stock i , R_m is the return for a market index, and σ_m^2 is the variance for the market:

$$\beta_i = \frac{\text{cov}(R_i, R_m)}{\sigma_m^2}$$

20. Write an R program to estimate the correlation matrix for the following five stocks:

#	Name	Ticker
1	International Business	IBM

	Machines	
2	Walmart	WMT
3	Citigroup	CG
4	Microsoft	MSFT
5	Johnson and Johnson	JNJ

21. Use the 10-year monthly price data to estimate the correlation matrix based on the previous five stocks (assume the returns of those five stocks follow a normal distribution).
22. Based on the correlation matrix, generate a return matrix to simulate their future monthly returns over the next 10 years.
23. Write programs in R or Python to generate similar F-tables (see the link at http://www.socr.ucla.edu/applets.dir/f_table.html).

24. In this chapter, we have the following Julia program to run CAPM (one-factor linear model). Simplify the program by writing a subroutine to estimate a return:

```

using DataFrames;
using GLM, StatsModels
#
x = readtable("c:/temp/ibmMonthly5years.csv")
p=x[:Adj_Close]
n=length(p)
stockRet=p[2:n]./p[1:(n-1)]-1
#
y = readtable("c:/temp/sp500Monthly5years.csv")
p2=y[:Adj_Close]
n2=length(p2)
mktRet=p2[2:n2]./p2[1:(n2-1)]-1
#
n3=min(length(stockRet),length(mktRet))

```

```
| data = DataFrame(X=mktRet[1:n3], Y=stockRet[1:n3])
| OLS = glm(@formula(Y ~ X), data, Normal(), IdentityLink())
```

Managing Packages

In the preface, we mentioned that this book is for readers who are looking for tools in the area of data science. For the researchers or practitioners working in the area of data science, there are several important issues. First, they need to understand their raw data, such as its purpose, structure, how reliable and complex it is, and how it is collected. Second, researchers and practitioners should have a good method of processing that data. In other words, they should master at least one computer language, such as R, Python, or Julia. After learning a language's basics, they should turn to some related packages, since understanding these packages might determine how far they can go in the area of data science. In this chapter, the following topics will be covered:

- Introduction to packages, modules, or toolboxes
- Two examples of using packages
- Finding all packages in R, Python, Julia, and Octave
- Task views for R
- Finding manuals
- Package dependencies
- Package management in R, Python, Julia, and Octave
- Conda-the package manager
- Creating a set of programs in R and Python
- Finding environmental variables

Introduction to packages, modules, or toolboxes

Over the years, researchers or users have generated many packages around different specific tasks for various programming languages. For this book, we treat *module* or *toolbox* as a synonym for *package*. For the analyses in the area of data science, it is very important to use various packages to achieve our goals. There are several advantages in using various packages. First, we don't have to write our code from scratch if we can find some relevant programs contained in certain packages. This would save us a huge amount of time. In other words, we don't have to reinvent the wheel, and this is especially true for developers. Second, packages are usually developed by people who have certain expertise in relevant areas. Because of this, the quality of a package is usually higher than the programs written by a, relatively speaking, new user. Third, for beginners, the knowledge of packaging and related skill sets would help them speed up their learning process.

Two examples of using packages

It is always a good idea to use examples to illustrate how useful or important it is to understand some closely related packages. The first example is extremely simple: generate a QR code for the CNN website. It has just two lines. Note that you need to run `install.packages("qrcode")` if the package is not preinstalled:

```
> library(qrcode) > qrcode_gen("https://www.cnn.com")
```

The generated QR code is shown here. Users can use the QR scanner installed



on their cell phone to go to the CNN website:

For the second example, we believe the best example for the researchers and users in the area of data science is an R package called `rattle`. If a user hasn't preinstalled the package, they can type the following line of R code:

```
>install.packages("rattle")
```

To launch the package, type the following two lines of R code:

```
>library(rattle)
```

Rattle: A free graphical interface for data science with R

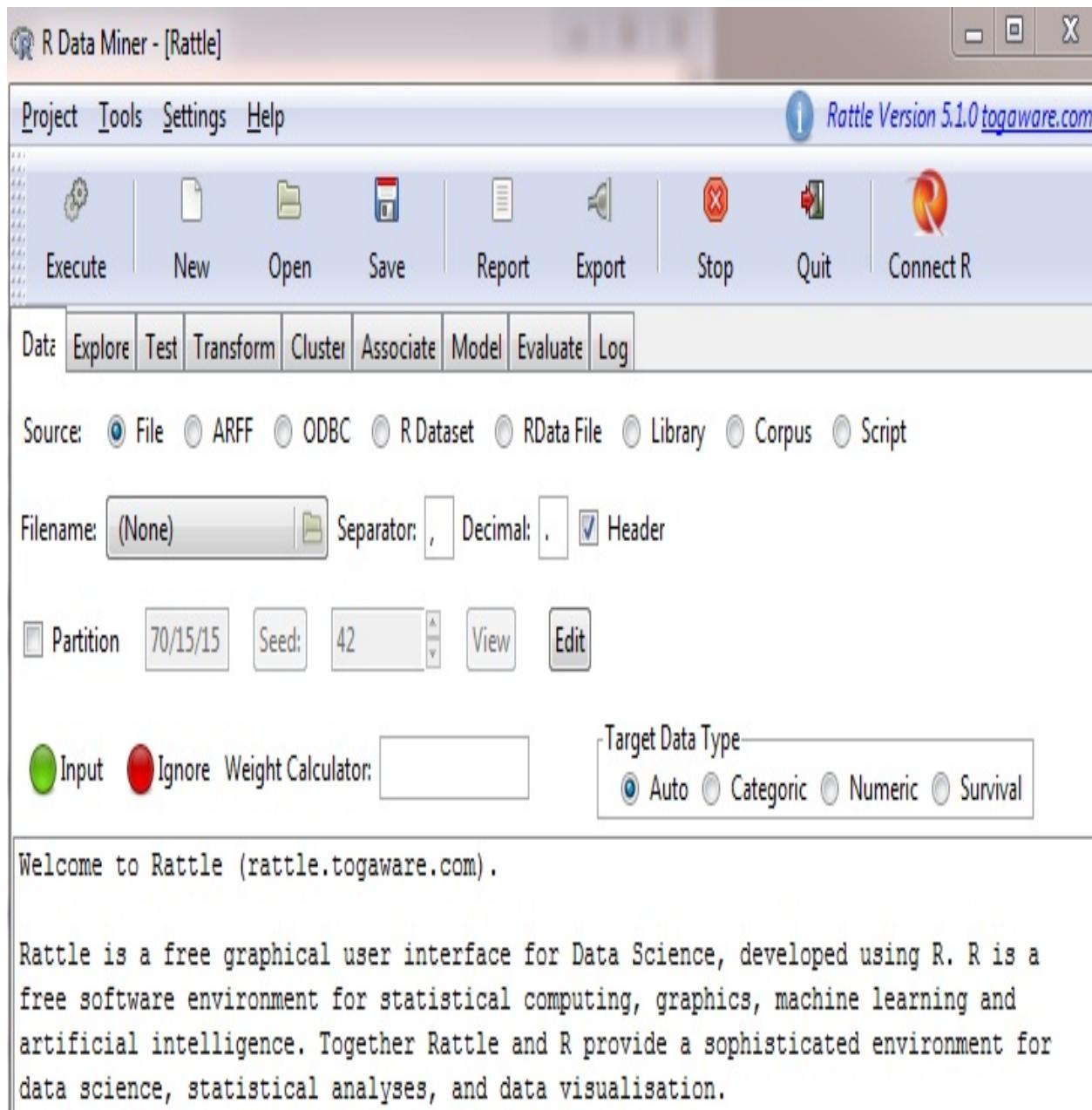
Version 5.1.0 Copyright (c) 2006-2017 Togaware Pty Ltd

Type rattle() to shake, rattle, and roll your data

```
>rattle()
```

Once the application is opened follow these steps:

1. After you hit the *Enter* key, the screen displays the following:



This is a powerful package for users in the area of data science. If we take the Project, Tools, Settings, and Help tabs as the first row, then the default location after we launch the package is on the Data tab (see the third row). Under Data, we can see that there are eight ways to retrieve data. Assume that we choose R Dataset. The advantage of this choice is that we can generate our own dataset with a data frame format. The following line of R code will retrieve the historical data for IBM. Users can also go to Yahoo! finance at <https://finance.yahoo.com> to download a more updated dataset: > x<-

```
read.csv("http://canisius.edu/~yany/data/ibmDaily.csv")
```

- After clicking R Dataset, followed by the down arrow key to the right of Data Name, we will see x. Select it and click Execute – we will see the following result:

Rattle Version 5

Project Tools Settings Help

Execute New Open Save Report Export Stop Quit Connect R

Data: Explore Test Transform Cluster Associate Model Evaluate Log

Source: File ARFF ODBC R Dataset RData File Library Corpus Script

Data Name: x

Partition 70/15/15 Seed: 42 View Edit

Input Ignore Weight Calculator:

Target Data Type
 Auto Categoric Numeric

No.	Variable	Data Type	Input	Target	Risk	Ident	Ignore	Weight	Comment
1	Date	Ident	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 13,990
2	Open	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 6,518
3	High	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 6,686
4	Low	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 6,688
5	Close	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 6,858
6	Adj.Close	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 11,758
7	Volume	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 9,814

- After clicking Explore, we can see several options on how to look at this specific dataset. For the Summary of the dataset, we have the following result:

Type: Summary Distributions Correlation Principal Components Interactive

Summary Describe Basics Kurtosis Skewness Show Missing Cross Tab

The data is limited to the training dataset.

Data frame:crs\$dataset[crs\$sample, c(crs\$input, crs\$risk, crs\$target)] 9793 observations and 6 variables Maximum # NAs:0

```
Storage
Open    double
High    double
Low     double
Close   double
Adj.Close double
Volume  integer
```

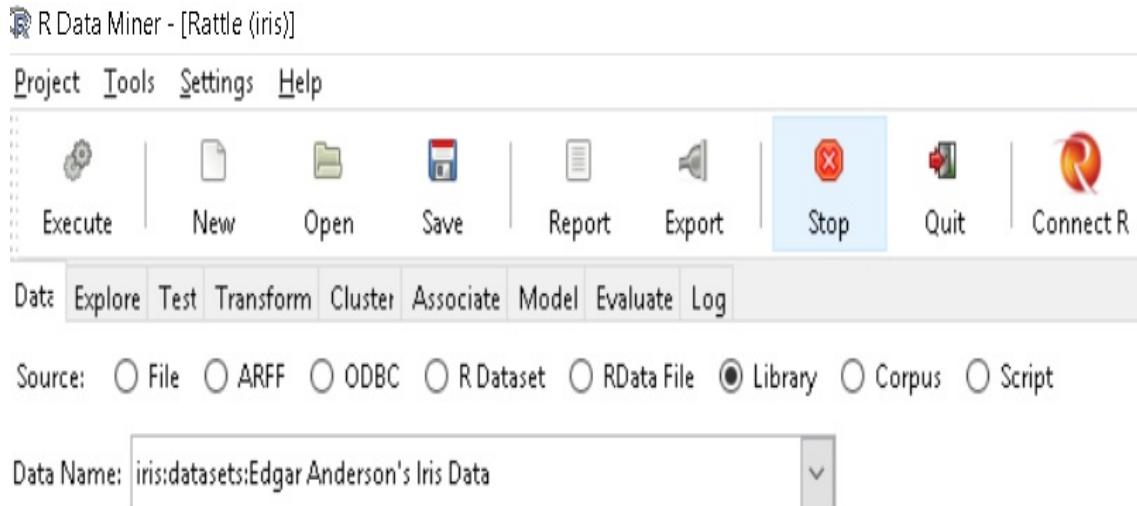
For the simple distribution tables below the 1st and 3rd Qu. refer to the first and third quartiles, indicating that 25% of the observations have values of that variable which are less than or greater than (respectively) the value listed.

	Open	High	Low	Close
Min. :	4.08	Min. : 4.287	Min. : 4.00	Min. : 4.08
1st Qu.:	15.84	1st Qu.: 15.963	1st Qu.: 15.73	1st Qu.: 15.84
Median :	27.56	Median : 27.812	Median : 27.34	Median : 27.56
Mean :	56.43	Mean : 56.961	Mean : 55.93	Mean : 56.45
3rd Qu.:	92.22	3rd Qu.: 93.000	3rd Qu.: 91.25	3rd Qu.: 92.27
Max. :	215.38	Max. :215.900	Max. :214.30	Max. :215.80

	Adj.Close	Volume
Min. :	1.189	Min. : 0
1st Qu.:	5.797	1st Qu.: 1214400
Median :	15.834	Median : 4127800
Mean :	43.126	Mean : 4866490
3rd Qu.:	71.146	3rd Qu.: 6947600
Max. :	190.226	Max. :69444700

Rattle timestamp: 2018-02-15 15:41:35 yany

The following is our third example. Click Data | Library, and then choose the Titanic dataset, as shown:



4. After clicking Execute, the screen will display the following:

No.	Variable	Data Type	Input	Target	Risk	Ident	Ignore	Weight	Comment
1	Sepal.Length	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 35
2	Sepal.Width	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 23
3	Petal.Length	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 43
4	Petal.Width	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 22
5	Species	Categoric	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 3

Now, let's try to use a decision tree to find out the impacts of **Sepal.Length**, **Sepal.width**, **Petal.Length**, and **Petal.width** on the classification of **Species**.

5. Choose **Species** as our target variable as shown in the previous screenshot.
 6. Click Model | Tree | Execute. From that, we will get the following result. To make our presentation better, the final output is divided into two parts. The top part is shown here:

Data | Explore | Test | Transform | Cluster | Associate | Model | Evaluate | Log

Type: Tree Forest Boost SVM Linear Neural Net Survival All

Target: Species Algorithm: Traditional Conditional Model Builder: rpart

Min Split:	20	Max Depth:	3	Priors:	<input type="text"/>	<input type="checkbox"/> Include Missing
Min Bucket:	7	Complexity:	0.0100	Loss Matrix:	<input type="text"/>	<input type="button" value="Rules"/> <input type="button" value="Draw"/>

Summary of the Decision Tree model for Classification (built using 'rpart'):

- The second part is shown next. On the right-hand side of the previous screenshot, we can see Draw, which can be used to draw a decision tree:

Summary of the Decision Tree model for Classification (built using 'rpart'):

```
n= 105
```

```

node), split, n, loss, yval, (yprob)
      * denotes terminal node

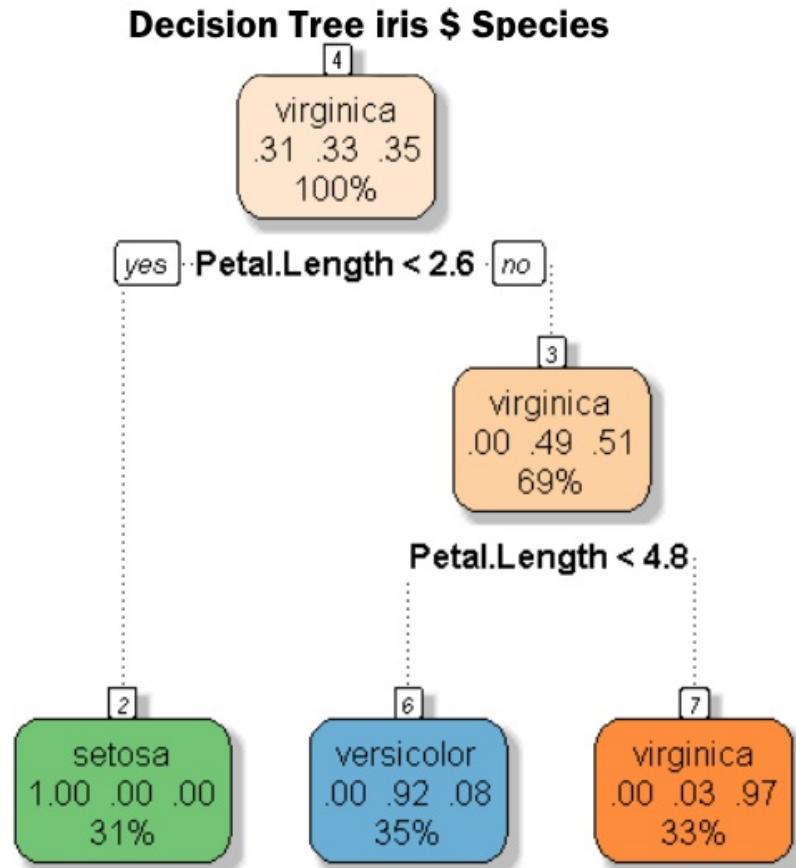
1) root 105 68 virginica (0.31428571 0.33333333 0.35238095)
  2) Petal.Length< 2.6 33  0 setosa (1.00000000 0.00000000 0.00000000) *
  3) Petal.Length>=2.6 72 35 virginica (0.00000000 0.48611111 0.51388889)
    6) Petal.Length< 4.85 37  3 versicolor (0.00000000 0.91891892 0.08108108) *
    7) Petal.Length>=4.85 35  1 virginica (0.00000000 0.02857143 0.97142857) *

```

Classification tree:

```
rpart(formula = Species ~ ., data = crs$dataset[crs$train, c(crs$input,
  crs$target)], method = "class", parms = list(split = "information"),
  control = rpart.control(usesurrogate = 0, maxsurrogate = 0))
```

- On the other hand, we can draw a decision tree based on the previous result by clicking on Draw (see the following diagram):



From the previous tree, the first layer or question is whether the **Petal.Length** is smaller than **2.6** or not. From the green box, we know that there were 100% chance that it is **setosa**. If the answer is **no** (that is, the **Petal.Length** is bigger than **2.6**), we have another question: is the **Petal.Length** longer than **4.8**.

After clicking on Rules, we will see several layers of rules, as shown: **Tree as rules:**

Rule number: 2 [Species=setosa cover=33 (31%) prob=1.00]
Petal.Length< 2.6

Rule number: 7 [Species=virginica cover=35 (33%) prob=0.00]
Petal.Length>=2.6
Petal.Length>=4.85

Rule number: 6 [Species=versicolor cover=37 (35%) prob=0.00]
Petal.Length>=2.6

Petal.Length< 4.85

[1] 2 1 5 4 3

Another great thing is that we can recover the R program or R scripts to run the previous decision tree model, after clicking on Log. To save space, the related graph and script will not be shown here. The related code can be found at the bottom of the log file, where users can save it or copy and paste it to generate their own program. Readers will find it at <http://canisius.edu/~yany/R/iris02.txt> (This link will be active from June 2018).

Finding all R packages

For R-related packages, go to <http://r-project.org> first. Click on CRAN and choose a mirror location, then click Packages on the left-hand side. We can see two lists, as shown here:

Available Packages

Currently, the CRAN package repository features 12173 available packages.

[Table of available packages, sorted by date of publication](#)

[Table of available packages, sorted by name](#)

On February 22, 2018, there are 12,173 R packages available. The first list contains all available packages sorted by their publication dates (that is, the dates they updated, or published if they were never updated). The second list is sorted by their names. If we just want to find relevant packages, either list will be fine. For example, for the first list, here is a snapshot of a few lines:

Date	Package	Title
2018-02-22	anipaths	Animation of Observed Trajectories Using Spline-Based Interpolation
2018-02-22	BayesXsrc	R Package Distribution of the BayesX C++ Sources
2018-02-22	profvis	Interactive Visualizations for Profiling R Code
2018-02-22	suncalc	Compute Sun Position, Sunlight Phases, Moon Position and Lunar Phase
2018-02-22	wbsts	Multiple Change-Point Detection for Nonstationary Time Series

The first column shows when the packages were last updated, or published if no updates were available. The second column shows the names of the packages, while the last column offers a short description of the usage for each package. We can use keywords to find the packages we want. Later in the chapter, we will discuss **task views**, which is a summary of all the related R packages around a topic.

Finding all Python packages

To find all Python packages, we can go to <https://pypi.python.org/>. The following screenshot shows the top part of the website. As of February 22, 2018, there are 130,230 packages available:



» Package Index

PACKAGE INDEX

[Browse packages](#)

[List trove classifiers](#)

[RSS \(latest 40 updates\)](#)

[RSS \(newest 40 packages\)](#)

[Terms of Service](#)

[PyPI Tutorial](#)

[PyPI Security](#)

[PyPI Support](#)

[PyPI Bug Reports](#)

[PyPI Discussion](#)

[PyPI Developer Info](#)

ABOUT

NEWS

DOCUMENTATION

DOWNLOAD

COMMUNITY

PyPI - the Python Package Index

The Python Package Index is a repository of software for the Python programming language. There are currently **130230** packages here. To contact the PyPI admins, please use the [Support](#) or [Bug reports](#) links.

Get Packages

To use a package from this index either "`pip install package`" ([get pip](#)) or download, unpack and "`python setup.py install`" it.

Package Authors

Submit packages with "[python setup.py upload](#)". You can also use [twine!](#) The index hosts [package docs](#). You must [register](#). Testing? Use [testpypi](#).

To find the packages we want, just click Browse packages and use keywords. For example, after entering `Data Science`, we will see the following results:

Index of Packages Matching 'data science'

Not Logged In
Login
Register
Lost Login?
Login with OpenID 
Login with Google 
Status
Spam on PyPI: monitoring

Package	Weight*	Description
jpl.mcl.site.science data 0.0.5	17	Science data for the MCL site
data-science 0.1	16	Full Data Science lib
datascience 0.10.4	14	A Jupyter notebook Python library for introductory data science
dataone.scimeta 2.4.2	13	Utilities for processing Science Metadata
jr-data-science 0.1.1	13	Quelques fonctions pour la data science.
data-science-library 1.0-dev	12	Mnubo Data Science Library
datapack 0.0	12	Data Packages for Science
datascience_tools 0.0.3	12	Tools for data science
datasciencebox 0.3	12	Data Science Box
datafuzz 0.1.0a0	11	A data-science library built for testing cleaning, schema validation and model robustness. It messes up your data so you can test your data engineering and data science code (before it breaks in production).
datakit-core 0.3.0	11	A pluggable command-line tool for custom data science workflows.
data-pack 0.0	10	Data packages
data-utilities 1.2.9	10	A data analysis and visualization helper module.

From the previous screenshot, we can see three columns: the first one gives the name of the package; the second one is called Weight, which can be viewed as the popularity index; and the last one offers a short description. The related URL is <https://pypi.python.org/pypi?%3Aaction=search&term=data+science&submit=search>.

Finding all Julia packages

For the packages written in Julia, we can go to <https://pkg.julialang.org/>. As of February 22, 2018, there are 1,725 packages available, as shown here:

Listing all 1725 registered packages for the Julia programming language.

Made with [PackageEvaluator.jl](#).

Last updated 2018-02-21 – Package ecosystem pulse

Packages tested on [Julia versions](#):

v0.5.2 (unmaintained release) – v0.6.2 (current release) – v0.7.0-DEV (unstable)

ACME

ACME.jl - Analog Circuit Modeling and Emulation for Julia

[MIT](#) license / Owner: HSU-ANT / [permalink](#)

26 ★ / 4 ↓ / 0 ↑

Julia v0.6: 0.6.2 (2 days ago) / Tests fail.

Julia v0.7: 0.6.2 (2 days ago) / Tests fail.

Again, we can use keywords to search this list. For example, if we use `data` as our keyword, we will find 94 locations – the first one is shown in the following screenshot:

AbaqusReader

AbaqusReader.jl is a parse for ABAQUS FEM models. It's capable of parsing the geometry accurately, including surface sets, node sets, and other relevant geometrical data used in FEM calculations. Other option is to parse whole model, including boundary conditions, material data and load steps.

[MIT license](#) / Owner: [JuliaFEM](#) / [permalink](#)

3 ★ / 1 ↓ / 1 ↑

Julia v0.6: [0.1.0](#) (6 months ago) /  Tests pass.

Finding all Octave packages

At <https://octave.sourceforge.io/packages.php>, we can find a list of all available packages for Octave:

Octave-Forge - Extra packages for GNU Octave

Home · Packages · Developers · Documentation · FAQ · Bugs · Mailing Lists · Links · Code

Packages

These packages are meant for current versions of Octave. See the [unmaintained](#) section for information on older versions.



bim

external

Package for solving Diffusion Advection Reaction (DAR) Partial Differential Equations



[details](#)



bsrtl

external

The BSRTL package is a free collection of OCTAVE/MATLAB routines for working with the biospeckle laser technique



[details](#)



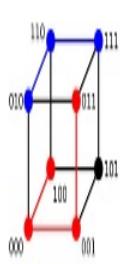
cgi

community

Common Gateway Interface for Octave



[details](#)



communications

community

Digital Communications, Error Correcting Codes (Channel Code), Source Code functions, Modulation and Galois Fields

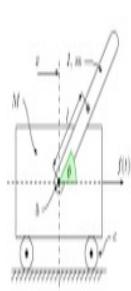


[details](#)



Again, we can search for keywords. If the word `data` is used, we will find 10

locations – the first few are shown here:

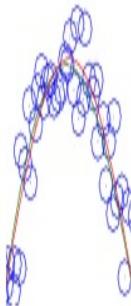


control

community

Computer-Aided Control System Design
(CACSD) Tools for GNU Octave, based on the
proven SLICOT Library

details download repository



data-smoothing

community

Algorithms for smoothing noisy data

details download repository



database

community

Interface to SQL databases, currently only
postgresql using libpq

details download repository

```
x_18145          C
double    double
1.0272e-11 7.0219e-13
1.1990e-11 6.9007e-13
1.3789e-11 6.9048e-13
1.4420e-11 6.8517e-13
1.2910e-11 6.7369e-13
1.2610e-11 6.7444e-13
1.4190e-11 6.7011e-13
1.0489e-11 6.6410e-13
               N 6.5358e-13
1.0610e-11 6.5355e-13
```

dataframe

community

Data manipulation toolbox similar to R data.frame

details download repository

Task views for R

A task view is a set of R packages grouped by one or more experts around a specific topic. For example, for data visualization, we could choose the task view called Graphics. For a text analysis, we could choose the NaturalLanguageAnalysis task view. To find a list of all these task views, we can go to the R home page at <http://r-project.org>. After clicking CRAN, choose a mirror server, then click Task Views on the left-hand side. The following screen will be displayed:

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
MetaAnalysis	Meta-Analysis
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
NumericalMathematics	Numerical Mathematics
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods
SocialSciences	Statistics for the Social Sciences
Spatial	Analysis of Spatial Data
SpatioTemporal	Handling and Analyzing Spatio-Temporal Data
Survival	Survival Analysis
TimeSeries	Time Series Analysis
WebTechnologies	Web Technologies and Services
gR	gRaphical Models in R

If we are interested in data visualization, then we can click on **Graphics** (see the following screenshot):

CRAN Task View: Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization

Maintainer: Nicholas Lewin-Koh

Contact: nikko at hailmail.net

Version: 2015-01-07

URL: <https://CRAN.R-project.org/view=Graphics>

R is rich with facilities for creating and developing interesting graphics. Base R contains functionality for many plot types including coplots, mosaic plots, biplots, and the list goes on. There are devices such as postscript, png, jpeg and pdf for outputting graphics as well as device drivers for all platforms running R. [lattice](#) and grid are supplied with R's recommended packages and are included in every binary distribution. [lattice](#) is an R implementation of William Cleveland's trellis graphics, while grid defines a much more flexible graphics environment than the base R graphics.

To save space, only the top part is shown. The task view gives many R-related packages around the topics of **Graphic Display & Visualization**. Another great benefit is installing all related packages by issuing just three lines of R code. Assume that we are interested in the task view related to finance. We can install all R packages listed with the task view by issuing the following commands:

```
>install.packages("ctv") >library("ctv") >install.views("Finance")
```

To update a specific view, we use the R function `update.views()`:

```
|>install.packages("ctv")
|>library("ctv")
|>update.views("Finance")
```

Finding manuals

For an R package, the best way to find the manual is to find the location of the installed R package. In the following example, we use the R package called `rattle` as an example:

```
> library(rattle) > path.package('rattle') [1]
```

```
"C:/Users/yany/Documents/R/win-library/3.3/rattle"
```

Note that different readers will definitely get different paths. Our result is shown in the following screenshot:

	csv	10/25/2017 3:41 PM File folder
	doc	10/25/2017 3:41 PM File folder
	etc	10/25/2017 3:41 PM File folder
	help	10/25/2017 3:41 PM File folder
	html	10/25/2017 3:41 PM File folder
	Meta	10/25/2017 3:41 PM File folder
	odt	10/25/2017 3:41 PM File folder
	po	10/25/2017 3:41 PM File folder
	R	10/25/2017 3:41 PM File folder
	CITATION	10/25/2017 3:41 PM File 1 KB
	DESCRIPTION	10/25/2017 3:41 PM File 4 KB
	INDEX	10/25/2017 3:41 PM File 3 KB
	MD5	10/25/2017 3:41 PM File 3 KB
	NAMESPACE	10/25/2017 3:41 PM File 2 KB
	NEWS	10/25/2017 3:41 PM File 48 KB

The PDF manual and HTML manuals are located under the `doc` subdirectory. It is a great idea to explore these subdirectories. To save space, we will not show the detailed files contained under the subdirectories. The second best way is to go to <http://r-project.org>, click CRAN, choose a nearby mirror location and click packages on the left-hand side. Then, from one of the two lists, search for the package. After clicking on the package, we will find the PDF file. For Python documentation, we can visit two websites <https://www.python.org/doc/> and <https://packaging.python.org/>. The second one is shown in the following screenshot:

Python Packaging User Guide

Welcome to the *Python Packaging User Guide*, a collection of tutorials and references to help you distribute and install Python packages with modern tools.

This guide is maintained on [GitHub](#) by the [Python Packaging Authority](#). We happily accept any contributions and feedback. ☺

Note: Looking for guidance on migrating from legacy PyPI to [pypi.org](#)? Please see [Migrating to PyPI.org](#).

To find related manuals for Julia and Octave, we can go to the website of the related package and find more information. Again, the two related websites are <https://pkg.julialang.org/> and <https://octave.sourceforge.io/packages.php>.

Package dependencies

There are two types of package dependencies. The first one is the package depending on the version of underlying software. For example, take the Octave package called `statistics`, available at <https://octave.sourceforge.io/statistics>. On February 22, 2018, it has a version of **1.3.0** and it requires an underlying Octave with a version of at least **4.0.0**, as shown in the last line of the following screenshot:

statistics

The screenshot shows the Octave package statistics page. On the left, a yellow box displays package details: Package Version: 1.3.0, Last Release Date: 2016-10-09, Package Author: various authors, Package Maintainer: Arno Onken <asnelt@asnelt.org>, and License: GPLv3+, public domain. To the right, there are two green and blue boxes. The green box contains 'Download Package' with a downward arrow icon, 'Repository' with a database icon, and 'Older versions'. The blue box contains 'Function Reference' with a question mark icon and 'NEWS' with a clipboard icon.

Description

Additional statistics functions for Octave.

Details

Dependencies: `Octave >= 4.0.0` `io >= 1.0.18`

The second type of dependency is between packages. Developers of various packages use many functions embedded in other developed packages. Not only does this save time, but it also means they don't have to reinvent the wheel. From the last line of the previous screenshot, we know that this package depends on another Octave package called `io`.

In the following, we show the process of installation. First, we download the ZIP file from <https://octave.sourceforge.io/statistics/>. After using the correct path directing us to the ZIP file, we issue the following command:

```
|>> pkg install statistics-1.3.0.tar.gz  
| error: the following dependencies were unsatisfied:  
|   statistics needs io >= 1.0.18
```

The `error` message indicates that we have to install another package first. After installing the `io` package, the problem is solved, and here is the result after issuing the previous command the second time:

```
|>> pkg install statistics-1.3.0.tar.gz
```

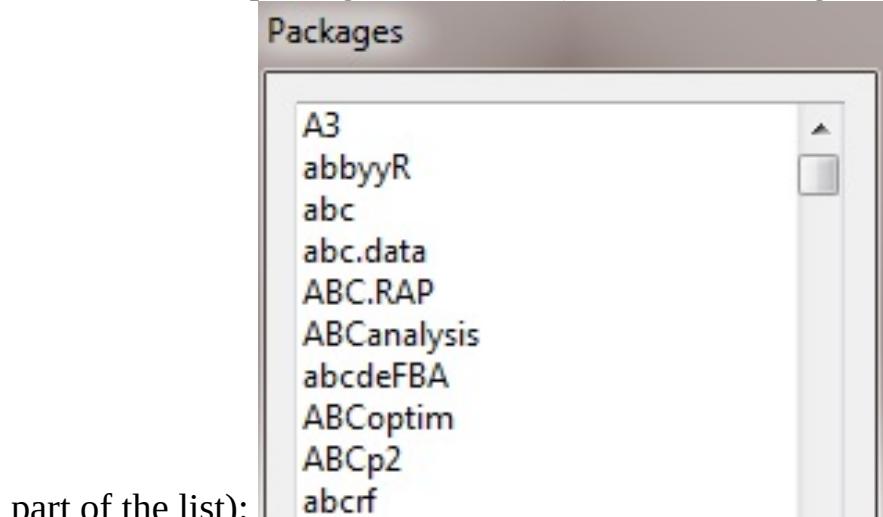
For information about changes from previous versions of the `statistics` package, run `news statistics`. Most of the time, users should not worry about package dependencies since many packages will update those packages automatically when we install them.

Package management in R

There are three ways to install an R package. The first way is to use the `install.packages()` function. For example, assume that we plan to install an R package called `rattle`. We can use the following code to do this:

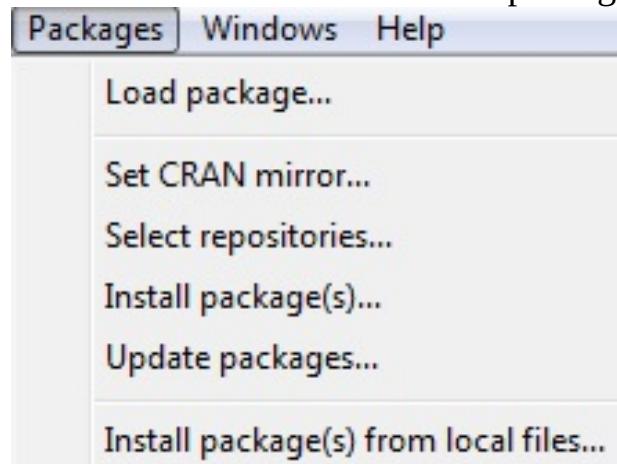
```
>install.packages("rattle")
```

The second way is to click Packages on the menu bar, choose a mirror location, then find the R package from a list (see the following screenshot showing the top



part of the list):

The third way to install an R package is to install it from a local ZIP file. To do so, first, manually download a ZIP file to your computer. Then click Packages on the menu bar and choose Install package(s) from local files..., as shown here:



To update a package, click Packages on the menu bar, then choose Update packages... from the drop-down menu (that is, the fifth entry in the previous screenshot). Another way to update is to use the `update.packages()` function, shown here: `>update.packages("rattle")`

The following table lists the most frequently used commands related to handling R packages:

Action	R command
Load an R package	<code>>library(rattle)</code> <code>>require(rattle)</code>
Install a package	<code>>install.packages("rattle")</code>
Update a package	<code>>update.packages("rattle")</code>
List all loaded packages	<code>>search()</code>
List all preinstall packages	<code>>library()</code>
XML package loaded or not	<code>>"package:XML" %in% search()</code>
XML package installed or not	<code>>"package:XML" %in% .packages(all.available=T)</code>
Unload a package	<code>detach(package:rattle)</code> <code>>detach("package:rattle", unload=TRUE)</code>

Find more information

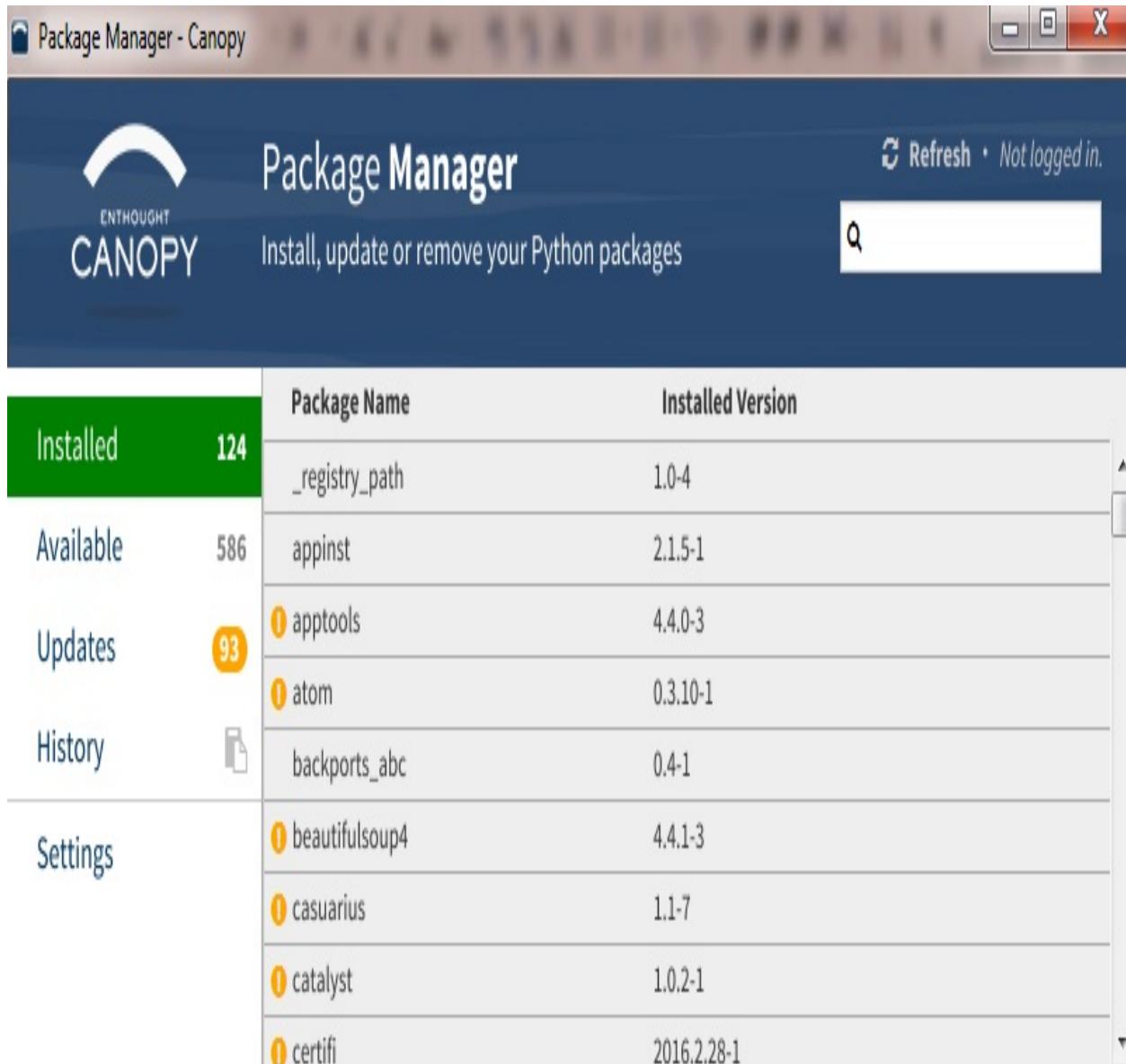
```
help(package=rattle)  
>library(help="rattle")
```

Table 6.1 R commands for handling packages As mentioned before, we could install all R packages listed within a specific task view by issuing just three lines of code. Since machine learning is one of the most important areas for data science, the related task view will be used as an example: >install.packages("ctv") >library("ctv") >install.views("MachineLearning")

As of February 22, 2018, there are about 168 R packages (see a list at <http://canisius.edu/~yany/doc/taskViewMachineLearning.txt>). We can also update its specific view: >install.packages("ctv") >library("ctv") >update.views("MachineLearning")

Package management in Python

We can use conda to install Python-related packages (see the related section later in the chapter). If we have various Python compilers, we can install Python packages easily. For example, if we use the Canopy compiler by Enthought, we can use Package Manager, as shown in the following screenshot:



The screenshot shows the Canopy Package Manager interface. The title bar reads "Package Manager - Canopy". The main header features the Enthought logo and the word "CANOPY". Below the header, there's a search bar with a magnifying glass icon and a "Refresh" button followed by the text "Not logged in.". The central area is titled "Package Manager" and contains the sub-instruction "Install, update or remove your Python packages". On the left, a sidebar lists navigation options: "Installed" (124), "Available" (586), "Updates" (93), "History", and "Settings". The "Installed" section is currently selected. The main table displays the following package information:

	Package Name	Installed Version
Installed	_registry_path	1.0-4
Available	appinst	2.1.5-1
Updates	apptools	4.4.0-3
	atom	0.3.10-1
History	backports_abc	0.4-1
Settings	beautifulsoup4	4.4.1-3
	casuarus	1.1-7
	catalyst	1.0.2-1
	certifi	2016.2.28-1

From this, we can find out how many packages were installed and how many are

available.

It's quite simple to install or update a package. For example, to install a package, we simply choose one from the list. The same logic applies when we update one. To find all embedded functions within a package, we can use the following command:

```
|>import matplotlib as mat  
|>x=dir(mat)  
|>print(x)
```

The related screenshot is shown here:

```
In [4]: import matplotlib as mat
```

```
In [5]: x=dir(mat)
```

```
In [6]: print(x)
```

```
['RcParams', 'URL_REGEX', 'Verbose', '_DATA_DOC_APPENDIX', '__builtins__', '__doc__', '__file__', '__name__',
 '__package__', '__path__', '__version__', '__version__numpy__', '_all_DEPRECATED', '_cm', '_cm_listed',
 '_cntr', '_contour', '_create_tmp_config_dir', '_decode_filesystem_path', '_deprecated_ignore_map',
 '_deprecated_map', '_error_details_fmt', '_get_cachedir', '_get_config_or_cache_dir', '_get_configdir',
 '_get_data_path', '_get_data_path_cached', '_get_home', '_get_xdg_cache_dir', '_get_xdg_config_dir',
 '_image', '_is_writable_dir', '_mathtext_data', '_obsolete_set', '_open_file_or_url', '_path', '_png',
 '_pylab_helpers', '_python26', '_qhull', '_rc_params_in_file', '_replacer', '_tri', '_url_lines',
 '_use_error_msg', '_version', 'absolute_import', 'afm', 'artist', 'axes', 'axis', 'backend_bases',
 'backend_tools', 'backends', 'bad_pyparsing', 'bezier', 'blocking_input', 'cbook', 'chain',
 'checkdep_dvipng', 'checkdep_ghostscript', 'checkdep_inkscape', 'checkdep_pdftops', 'checkdep_ps_distiller',
 'checkdep_tex', 'checkdep_usetex', 'checkdep_xmllint', 'cm', 'collections', 'colorbar', 'colors',
 'compare_versions', 'compat', 'container', 'contextlib', 'contour', 'converter', 'cycler', 'dates',
 'dateutil', 'dedent', 'default', 'defaultParams', 'default_test_modules', 'distutils', 'division',
 'docstring', 'dviread', 'externals', 'f', 'figure', 'finance', 'font_manager', 'fontconfig_pattern',
 'ft2font', 'functools', 'get_backend', 'get_cachedir', 'get_configdir', 'get_data_path', 'get_example_data',
 'get_home', 'get_label', 'get_py2exe_datafiles', 'gridspec', 'image', 'inspect', 'interactive', 'io',
 'is_interactive', 'is_string_like', 'is_url', 'key', 'legend', 'legend_handler', 'lines', 'locale', 'major',
 'markers', 'mathtext', 'matplotlib_fname', 'minor1', 'minor2', 'mlab', 'mplDeprecation', 'numpy',
 'offsetbox', 'os', 'patches', 'path', 'print_function', 'projections', 'pylab', 'pyparsing', 'pyplot',
 'quiver', 'rc', 'rcParams', 'rcParamsDefault', 'rcParamsOrig', 'rc_context', 'rc_file', 'rc_file_defaults',
 'rc_params', 'rc_params_from_file', 'rcdefaults', 'rcsetup', 're', 'reload', 's', 'scale', 'six', 'spines',
 'stackplot', 'streamplot', 'style', 'subprocess', 'sys', 'table', 'tempfile', 'test', 'texmanager', 'text',
 'textpath', 'ticker', 'tight_bbox', 'tk_window_focus', 'tmp', 'transforms', 'tri', 'unicode_literals',
 'units', 'unpack_labeled_data', 'urlopen', 'use', 'validate_backend', 'verbose', 'verify_test_dependencies',
 'warnings', 'widgets']
```

Package management in Julia

To see a list of installed packages, we use the `Pkg.status()` function, as shown:

```
julia> Pkg.status()
11 required packages:
- AbstractTables          0.0.1
- Calculus                 0.2.2
- Clustering                0.9.1
- DataFrames                  0.10.1
- GLM                         0.8.1
- Gadfly                      0.6.4
- IJulia                      1.7.0
- Plots                        0.15.0
- PyPlot                       2.3.2
- QuantEcon                   0.14.1
```

To save space, only the first several lines are shown. Alternatively, we can issue `Pkg.installed()`, which returns a dictionary, mapping installed package names to the versions of those which are installed, as shown here:

```
julia> Pkg.installed()
Dict{String,VersionNumber} with 91 entries:
"CodecZlib"           => v"0.4.2"
"MacroTools"          => v"0.4.0"
"Reexport"            => v"0.1.0"
"CoupledFields"       => v"0.0.1"
"DualNumbers"         => v"0.3.0"
"Measures"            => v"0.1.0"
"Plots"               => v"0.15.0"
"NLOpt"                => v"0.3.6"
"CommonSubexpressions" => v"0.0.1"
"KernelDensity"        => v"0.4.1"
"IJulia"                => v"1.7.0"
"Compat"                => v"0.53.0"
"PlotThemes"           => v"0.2.0"
"FixedPointNumbers"    => v"0.4.6"
"DiffRules"             => v"0.0.3"
"SpecialFunctions"     => v"0.3.8"
"PositiveFactorizations" => v"0.1.0"
"StatsBase"             => v"0.19.5"
"Primes"                => v"0.2.0"
"AbstractFFTs"          => v"0.2.1"
?                      => ?
```

To add or remove a package, we apply the `Pkg.add()` and `Pkg.rm()` functions, as shown in this example: Julia>`Pkg.add("AbstractTable")`
Julia>`Pkg.rm("AbstractTable")`

To get all the latest versions, we issue the following command:

```
| Julia>Pkg.update()
```

Package management in Octave

We will use the Octave package called `statistics` as an example. First, we find the ZIP file for the package at <https://octave.sourceforge.io/statistics/>, as shown in the following screenshot:

statistics



Description

Additional statistics functions for Octave.

Details

Dependencies: `Octave` >= 4.0.0 `io` >= 1.0.18

Second, we set up our path to the directory containing the previously downloaded ZIP file. Third, we issue `pkg install package_name`, as shown:

```
| > pkg install statistics-1.3.0.tar.gz
```

For information about changes from previous versions of the `statistics` package, run `news statistics`. To get more information about the new version, we type `news statistics` as mentioned previously:

```
>> news statistics
Summary of important user-visible changes for statistics 1.3.0:
-----
** The following functions are new:
bbscdf bbsinv bbspdf bbsrnd
binotest
burrcdf burrinv burrpdf burrrnd
gpcdf gpinv gppdf gprnd
grp2idx
mahal
mvtpdf
nakacdf nakainv nakapdf nakarnd
pdf
tricdf triinv tripdf trirnd
violin
```

To load and unload a package, we have the following code:

```
|>pkg load statistics
|>pkg unload statistics
```

As for all the other functions included in the `statistics` package, see <https://octave.sourceforge.io/statistics/overview.html>.

Conda – the package manager

After we launch Anaconda Prompt and issue `conda help`, we will see the following output:



The screenshot shows the Anaconda Prompt window with the title bar "Anaconda Prompt". The main area displays the output of the `conda help` command. The output includes the usage information, a brief description of what Conda is, and detailed descriptions of various commands and their options.

```
(base) C:\Users\yany>conda help
usage: conda [-h] [-V] command ...

conda is a tool for managing and deploying applications, environments and packages.

Options:
positional arguments:
command
  clean      Remove unused packages and caches.
  config     Modify configuration values in .condarc. This is modeled
             after the git config command. Writes to the user .condarc
             file (C:\Users\yany\.condarc) by default.
  create     Create a new conda environment from a list of specified
             packages.
  help       Displays a list of available conda commands and their help
             strings.
  info       Display information about current conda install.
  install    Installs a list of packages into a specified conda
             environment.
  list       List linked packages in a conda environment.
  package   Low-level conda package utility. <EXPERIMENTAL>
  remove    Remove a list of packages from a specified conda environment.
  uninstall Alias for conda remove. See conda remove --help.
  search    Search for packages and display associated information. The
             input is a MatchSpec, a query language for conda packages.
             See examples below.
  update    Updates conda packages to the latest compatible version. This
             command accepts a list of package names and updates them to
             the latest versions that are compatible with all other
             packages in the environment. Conda attempts to install the
             newest versions of the requested packages. To accomplish
             this, it may update some packages that are already installed,
             or install additional packages. To prevent existing packages
             from updating, use the --no-update-deps option. This may
             force conda to install older versions of the requested
             packages, and it does not prevent additional dependency
             packages from being installed. If you wish to skip dependency
             checking altogether, use the '--force' option. This may
             result in an environment with incompatible packages, so this
             option must be used with great caution.
  upgrade   Alias for conda update. See conda update --help.

optional arguments:
-h, --help      Show this help message and exit.
-V, --version   Show the conda version number and exit.

conda commands available from other packages:
env
```

From the previous help menu, we know that we can install, update, and uninstall

a package. Usually, we can use `conda` to install a package. However, we may receive an error message (see the following example):

```
C:\Users\yany\AppData\Local\Continuum\anaconda3> C:\Users\yany>conda install cb
sodata
Fetching package metadata .....
PackageNotFoundError: Packages missing in current channels:
- cbsodata

We have searched for the packages in the following channels:
- https://repo.continuum.io/pkgs/main/win-32
- https://repo.continuum.io/pkgs/main/noarch
- https://repo.continuum.io/pkgs/free/win-32
- https://repo.continuum.io/pkgs/free/noarch
- https://repo.continuum.io/pkgs/r/win-32
- https://repo.continuum.io/pkgs/r/noarch
- https://repo.continuum.io/pkgs/pro/win-32
- https://repo.continuum.io/pkgs/pro/noarch
- https://repo.continuum.io/pkgs/msys2/win-32
- https://repo.continuum.io/pkgs/msys2/noarch
```

To update `conda` itself, we use `conda update -n base conda` command, as shown here:

```
(base)> C:\Users\yany>
(base)> C:\Users\yany>conda install numpy
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.4.7
  latest version: 4.4.10

Please update conda by running
$ conda update -n base conda
```

We could find more information about specific Python packages by using the search function, as shown here:

```
(base)> C:\Users\yany>conda search matplotlib
Loading channels: done
Name          Version      Build  Channel
matplotlib    1.2.0       np16py27_0  defaults
matplotlib    1.2.0       np16py27_1  defaults
matplotlib    1.2.0       np17py27_1  defaults
matplotlib    1.2.0       np17py33_1  defaults
matplotlib    1.2.1       np17py27_0  defaults
matplotlib    1.2.1       np17py27_1  defaults
matplotlib    1.2.1       np17py33_0  defaults
matplotlib    1.2.1       np17py33_1  defaults
```

The following table lists several of the most used commands:

Command	Explanation
Conda help Conda info	Get help about the usages of conda
conda update -n base conda	Get information related to conda, such as the current version, base environment and the related websites
conda search matplotlib	Find all versions about this specific Python package
conda install matplotlib	Install the Python package called matplotlib
conda update matplotlib	Update the Python packaged called matplotlib
conda remove matplotlib	Remove the Python package called matplotlib

Table 6.2 Most used Conda commands

Creating a set of programs in R and Python

On numerous occasions, for a specific research topic, researchers will collect many datasets and write many programs. Why not write a big program? There are several reasons for not doing so. First, we might need several steps to finish the project. Second, the project might be too complex, so we have divided the whole project into several small portions. Each researcher will be responsible for one or a few portions. Third, according to the flow of the whole process, we might want to have several parts, such as devoting a part to processing data, a part to running various regressions, and a part to summarizing the results. Because of this, we need a way of putting all the programs together. In the following example, we will show you how to achieve this in both R and Python. For R, assume that we have the following functions:

```
| pv_f<-function(fv,r,n) fv/(1+r)^n
| fv_f<-function(pv,r,n)pv*(1+r)^n
| pv_annuity<-function(c,r,n)c/r*(1-1/(1+r)^n)
| fv_annuity<-function(c,r,n)c/r*((1+r)^n-1)
| pv_perpetuity<-function(c,r)c/r
| pv_perpetuityDue<-function(c,r)c/r*(1+r)
```

To make our task easy, we deliberately chose these one-line functions. Assume that the file is called `financialCalculator.R` under `c:/temp`. We use the `source()` command to activate them and use the `ls()` function to list all contained functions:

```
| > source("c:/temp/financialCalculator.R")
| > ls()
| [1] "fv_annuity" "fv_f" "pv_annuity"
| [4] "pv_f" "pv_perpetuity" "pv_perpetuityDue" mp/fincalCalculator.R")
```

For Python, we can generate a file containing all related functions. Then we can use an import function to load them all. First, download a Python program at <http://canisius.edu/~yany/python/myPackage.py>. Assume that it is saved under `c:/temp/`. We type the following two lines:

```
| import py_compile
| py_compile.compile('c:/temp/myPackage.py')
```

After entering these two lines, we have generated a file with an extension of `pyc`, as shown here:

Name	Date modified	Type	Size
 myPackage.pyc	2/27/2018 2:07 PM	Compiled Python ...	19 KB
 myPackage.py	2/27/2018 2:03 PM	PY File	21 KB

Now, we can use the `import` function to import the contained functions, as shown in the following command:

```
>>>sys.path.append("c:/temp/")
>>>import myPackage as my
>>>x=dir(my)
>>>print(x)
```

Note that the first command adds `c:/temp/` to the path, while the second one imports our package. The following screenshot shows the output:

```
In [19]: sys.path.append("c:/temp/")
In [20]: import myPackage as my
In [21]: x=dir(my)
In [22]: print(x)
['CND', 'EAR_f', 'IRR_f', 'IRRs_f', 'NPER', 'PMT', 'Rc_f', 'Rm_f', '__builtins__', '__doc__',
 '__file__', '__name__', '__package__', 'binomial_grid', 'bond_price', 'bs_call', 'bs_call_old',
 'bs_put', 'delta_call', 'delta_put', 'durationBond', 'fv_annuity', 'fv_f', 'mean', 'n_annuity',
 'npv_f', 'payback_', 'payback_period', 'pvValueNperiodModel', 'pv_annuity',
 'pv_annuity_k_period_from_today', 'pv_excel', 'pv_f', 'pv_grow_perpetuity', 'pv_growing_annuity',
 'pv_perpetuity', 'pv_perpetuity_due', 'r_continuous', 'sign']
```

Finding environmental variables

For R, we can use the `sys.getenv()` function to find all environmental variables:

```
> Sys.getenv()
$ALLUSERSPROFILE          C:\ProgramData
$APPDATA                   C:\Users\yany\AppData\Roaming
$CommonProgramFiles         C:\Program Files\Common Files
$COMPUTERNAME               IC-22866
$ComSpec                    C:\windows\system32\cmd.exe
$CP_USER_NAME                yany
$EMC_AUTOPLAY              C:\Program Files\Common Files\Roxio Shared\
$f95include                 C:\Program Files\Silverfrost\FTN95\include
$FP_NO_HOST_CHECK           NO
$GFORTRAN_STDERR_UNIT       -1
$GFORTRAN_STDOUT_UNIT       -1
$HOME                       C:\Users\yany\Documents
```

To save space, only the top part is shown. Again, different users will get different results. For Python, we use the following commands: import sys
sys.path

The top part of the output is shown here:

```
In [8]: sys.path
Out[8]:
['',
'C:\\\\Users\\\\yany\\\\AppData\\\\Local\\\\Enthought\\\\Canopy32\\\\User\\\\Scripts\\\\python27.zip',
'C:\\\\Users\\\\yany\\\\AppData\\\\Local\\\\Enthought\\\\Canopy32\\\\App\\\\appdata\\\\canopy-1.7.4.3348.win-x86\\\\DLLs',
'C:\\\\Users\\\\yany\\\\AppData\\\\Local\\\\Enthought\\\\Canopy32\\\\App\\\\appdata\\\\canopy-1.7.4.3348.win-x86\\\\lib',
'C:\\\\Users\\\\yany\\\\AppData\\\\Local\\\\Enthought\\\\Canopy32\\\\App\\\\appdata\\\\canopy-1.7.4.3348.win-x86\\\\lib\\\\plat-win',
'C:\\\\Users\\\\yany\\\\AppData\\\\Local\\\\Enthought\\\\Canopy32\\\\App\\\\appdata\\\\canopy-1.7.4.3348.win-x86\\\\lib\\\\lib-tk',
'C:\\\\Users\\\\yany\\\\AppData\\\\Local\\\\Enthought\\\\Canopy32\\\\App\\\\appdata\\\\canopy-1.7.4.3348.win-x86',
'C:\\\\Users\\\\yany\\\\AppData\\\\Local\\\\Enthought\\\\Canopy32\\\\User',
```

For Julia, we use the `ENV` function, as in the following:

```
julia> ENV
Base.EnvHash with 47 entries:
"ALLUSERSPROFILE" => "C:\\\\ProgramData"
"APPDATA" => "C:\\\\Users\\\\yany\\\\AppData\\\\Roaming"
"CommonProgramFiles" => "C:\\\\Program Files\\\\Common Files"
"COMPUTERNAME" => "IC-22866"
"ComSpec" => "C:\\\\windows\\\\system32\\\\cmd.exe"
"CP_USER_NAME" => "yany"
"EMC_AUTOPLAY" => "C:\\\\Program Files\\\\Common Files\\\\Roxio Shared\\\\"
"f95include" => "C:\\\\Program Files\\\\Silverfrost\\\\FTN95\\\\include"
"FP_NO_HOST_CHECK" => "NO"
"HOMEDRIVE" => "H:"
"HOME PATH" => "\\"
"HOME SHARE" => "\\\\filer\\\\faculty\\\\yany"
"JULIA_EDITOR" => "vim"
"LOCALAPPDATA" => "C:\\\\Users\\\\yany\\\\AppData\\\\Local"
"LOGONSERVER" => "\\\\AD-CANISIUS0"
"mod_path" => "C:\\\\Program Files\\\\Silverfrost\\\\FTN95\\\\include"
"MOZ_PLUGIN_PATH" => "C:\\\\Program Files\\\\Foxit Software\\\\Foxit Reader\\\\p.

"NUMBER_OF_PROCESSORS" => "4"
"OPENBLAS_MAIN_FREE" => "1"
"OS" => "Windows_NT"
? => ?
```

For Octave, we can use the `getenv()` function, as shown:

```
>> getenv('path')
ans = C:OctaveOctave-4.0.0bin;C:Program FilesSilverfrostFTN95;C:Perlsitebin;C:Perlbin;C:
windowsSystem32Wbem;C:windowsSystem32WindowsPowerShellv1.0;C:Program FilesIntelOpenCL SI
Common FilesRoxio SharedDLLShared;C:Program FilesCommon FilesRoxio Shared10.0DLLShared;C
;C:Anaconda;C:AnacondaScripts;C:Program FilesWindows Kits8.1Windows Performance Toolkit;
OctaveOctave-4.0.0bin;C:OctaveOctave-4.0.0notepad++;C:OctaveOctave-4.0.0libexecoctave4.0
OctaveOctave-4.0.0libexecoctaveapi-v50+siteexeci686-w64-mingw32;C:OctaveOctave-4.0.0libe
ingw32;C:OctaveOctave-4.0.0libexecoctave4.0.0execi686-w64-mingw32;C:OctaveOctave-4.0.0bi
>>
```

Summary

In this chapter, we have first discussed the importance of managing packages. Then, we have shown how to find all the available packages for R, Python, Julia, and Octave, how to install and update individual packages, and how to find the manual for teaching the packages. In addition, we have explained the issue of package dependencies and how to make our programming a little easier when dealing with packages. The topic of systematic environment was touched on as well.

In [Chapter 7](#), *Optimization in Anaconda*, we will discuss several topics around optimization, such as general issues for optimization problems and expressing various kinds of optimization problems (for example, LP and quadratic optimization). Several examples are offered to make our discussion more practice-oriented, such as how to choose an optimal stock portfolio and how to optimize wealth and resources to promote sustainable development and how much the government should tax. In addition, we will introduce several packages for optimization in R, Python, Julia, and Octave, such as `optimization`, `M1BayesOpt`, `rgenoud`, `colf`, and `mize`.

Review questions and exercises

1. Why is understanding various packages important?
2. What are package dependencies?
3. For R, Python, Julia, and Octave, find out how many packages are available for each of them, today.
4. How do we install a package in R, Python, and Julia?
5. How do we update a package in R, Python, and Julia?
6. What is the task view for R?
7. How do we install all R packages included in a task view?
8. After an R package is installed, how do you find its related directory? What is the usage to find its related directory? You could use the R package called `healthcare` as an example. Note that the package is about tools for healthcare machine learning.
9. Find out more details about the task views related to the subject of Econometrics. Then install all related R packages. How many are there?
10. How do we update one R package? How would we do it for Octave?
11. How do we find the manual for a specific package written in R, Python, Julia, or Octave?
12. Choose a subject, such as graph, data visualization, find 10 packages for each language of R, Python, Julia, and Octave. Use machine learning as an example.
13. Write an R program to test if a specific R package is loaded or not. If it is not loaded but preinstalled, then load the R package. Also, if the R package is not installed, then remind users to install it.
14. How do we use conda to install a Python package called `matplotlib`?
15. Where can we find the manual related to a Python package called `matplotlib`? How do we find all the embedded functions?
16. How do we find the manual for an R package called Stats?
17. For exchanging data between Excel and R, we can use an R package called `XLConnect`. Install the package and issue the following two lines. Comment on your result:

```
>library(XLConnect)  
  
>vignette("XLConnect")
```

Try about a dozen R packages to see if the function is valid for all of them.

18. How do we find the so-called environment variables in R, Python, Julia, and Octave?
19. Generate two dozen R programs around a topic, such as text mining. Save them within one file. Activate them and use them.
20. Repeat the previous exercise for the Python language.

Optimization in Anaconda

Optimization plays a very important role in the area of data science. For example, in finance, investors are constantly seeking a trade-off between risk and return. To diversify their investment, they would like to invest in different industries or buy several stocks. Thus, for an expected portfolio return, how do they choose appropriate stocks to minimize their portfolio risk? For this objective, we could apply some kind of portfolio optimization technique.

Another application is linked to the government's tax policy. We know that cutting the corporate tax rate would encourage companies that are considering more capital investment (that is, long-term investment), such as in equipment. However, at the same time, the government's revenue might fall. If this is true, the government would have to cut many programs intended to help unfortunate people. Because of this, the government has a strong incentive to find an optimal tax rate. In this chapter, the following topics will be covered:

- Why optimization is important
- General issues for optimization problems
- Quadratic optimization
- Example #1 – stock portfolio optimization
- Example #2 – optimal tax policy
- Packages for optimization in R, Python, Octave, and Julia

Why optimization is important

In our lives, people face all kinds of choices. In a sense, we implement, either consciously or subconsciously, various kinds of *implied* optimization procedures. For example, when a high school junior or senior is looking for a college, they might have many choices, such as good schools, local schools, public schools, or private schools. When making a decision with a couple of offers, these high school students usually have some objectives in their mind. These objectives might include the ranking and cost of attending a school, scholarships, the reputation, and name recognition of the program, or even the fame of the football team. For corporations, they have to make all kinds of optimal or reasonable decisions. For instance, what kinds of products they should produce, what are the quantities and at what prices, and to whom they should target. Since most companies have limited resources, they have to make an optimal decision with certain constraints. For example, in employee scheduling, the target function is the total cost. At the same time, we have certain constraints such as the number of employees, their seniority, preferences, and the like.

Here is a general form for the optimization problem, where $\text{Min}()$ is a minimizing function and $f(x)$ is our objective function:

$$\underbrace{\text{Min } f}_{\text{subject to constraints}} = f(x) \quad (1)$$

We intend to minimize f by changing x , which is a set of input values, by subjecting it to certain constraints. Note that if we plan to maximize f , we simply could have minimized its negative value. Later in the chapter, we introduce a utility function for investors, which could be viewed as the trade-off between benefits (expected portfolio return) and cost (the portfolio risk). For investors, they want to maximize the utility. It could be equivalent to minimizing its negative objective function; see the following formula:

$$\underbrace{\text{Min } U}_{\text{subject to constraints}} = -U(x)$$

Where $U()$ is the utility function and x will be a set of input variables. The

following table shows a partial list of applications for optimization:

Area	Name	Description
Economics	Utility function	For many variables in the real world, it is difficult to quantify them, such as happiness. For this, economists have developed a utility function. To find an optimal solution, users could maximize their utility function.
	Quantity, price, and production	A company might consider the best strategy to produce, price, and sell its products.
	Welfare transfer	Governments might consider many input variables such as incentives to work, crime rate, and other factors.
	Optimal tax policy	Maximize the GDP growth, subject to controllable inflation rate.
Finance	Portfolio optimization	For a given portfolio of expected returns, choose a set of stocks to minimize the portfolio risk.
	Optimal investment strategy	When facing several possible investment opportunities and many constraints, how to choose the best combination to achieve the best result.
	Profit optimization	For companies, they want to maximize their profit by optimally choosing their input variables.
	High-frequency trading	Balance the signalling effect with various bid-and-ask offers, and long-term and short-term profits.
	Scheduling	Minimize the total cost with certain constraints.
	Minimize total cost	Consider various trade-offs between certain benefits and costs.

Others	Flying issues	Minimize the total distance by visiting certain cities.
	Task arrangements	With many tasks and potential labor and input variables available, how to achieve an optimal solution.

Application of optimization

General issues for optimization problems

There are several issues in optimization. The most important one is how to choose an appropriate objective function. For some cases, the objective function is obvious. Unfortunately, for other cases, it is not that crystal clear. Since choosing a good objective depends on the specific situation, we will discuss it further, but remember that an appropriate objective function might make our task much easier.

In many cases, an inappropriate objective function might cause the following problems:

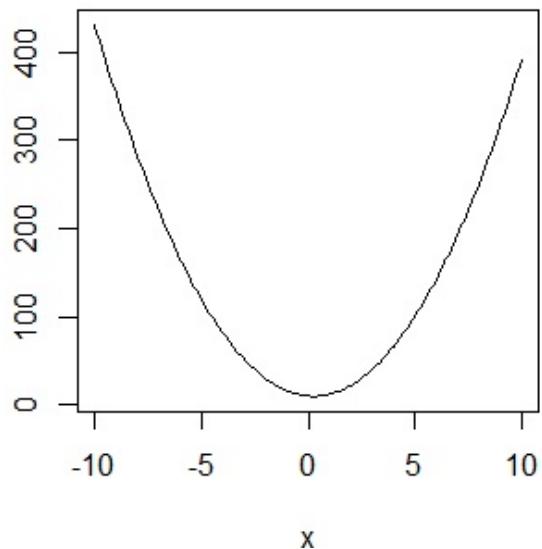
- It is difficult to find a feasible solution
- We might end up with a local solution
- We might have a corner solution
- It takes a long time to converge (that is, too much computation time to find a good solution)

Let's look at a convex function; the code and corresponding graph are given here:

```
| x<-seq(-10,10,0.1)
| a<-4
| b<- -2
| c<-10
| y<-a*x^2+b*x+c
| plot(x,y,type='l')
```

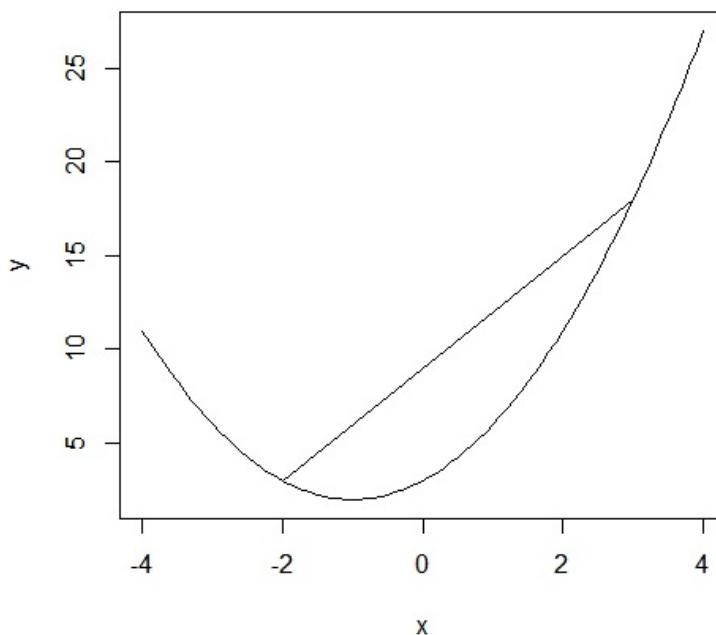
This code will provide an output shown here:

A convex function



Geometrically, a function is convex if a line segment drawn from any point (x_1, y_1) to another point (x_2, y_2) – called the chord from x to y – lies on or above the graph of f , as shown in the diagram here:

For a convex function: chord is above



The code for this graph and one of its chords is given here:

```
| x<-seq(-4,4,0.1)
```

```

a<-1
b<-2
c<-3
myFunction<-function(x)a*x^2+b*x+c
y<-myFunction(x)
name<- "For a convex function:chord is above"
plot(x,y,type='l',main=name)
x1<--2
y1<-myFunction(x1)
x2<-3
y2<-myFunction(x2)
segments(x1, y1, x2, y2,col = par("fg"), lty = par("lty"), xpd = FALSE)

```

For a convex function, it is quite easy to find its optimal solution. However, it is not that easy to find an optimal solution for a non-convex function. In that case, we would have to face a few issues: a global solution versus a local solution, unconstrained versus constrained optimization, initial input values, and conversion criteria.

A global solution is a solution that achieves our objective function by using all the values or possible input values. However, for some reason, our program could fail to find a global solution, but finds a local solution instead. A local solution is an optimal solution for just a small area. Let's look at the following function:

$$f(x, y) = (x^2 - 1)^2 + (x^2 y - x - 1)^2 \quad (2)$$

Since both items on the right-hand side are positive, $f(x,y)$ could have multiple minima. When $x=-1$ and $y=0$, $f(x,y)$ is zero, so $(-1,0)$ is a minimum. In addition, when $x=1$ and $y=2$, $f(x,y)$ is zero as well, so $(1,2)$ is also a minimum. Note that users need to run `install.packages('scatterplot3d')` if that library is not installed yet:

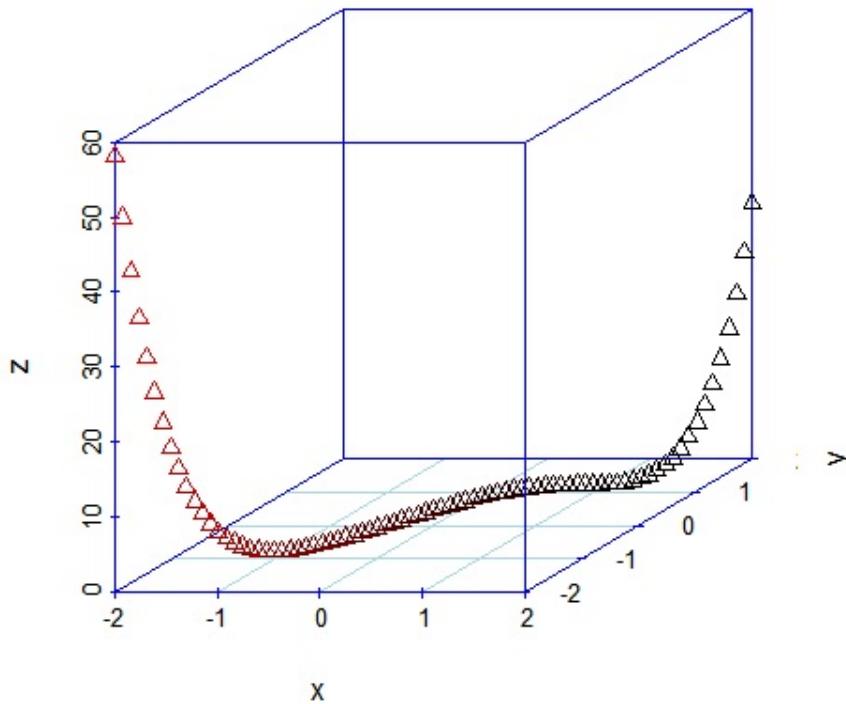
```

library(scatterplot3d)
x<-seq(-2,2,0.05)
y<-seq(-2,2,0.05)
z<-(x^2-1)^2+(x^2*y-x-1)^2
name<-"3 dimensional graph"
scatterplot3d(x, y, z, highlight.3d = TRUE, col.axis = "blue",
col.grid = "lightblue", main =name, pch = 2)

```

The related three-dimensional graph is shown here:

3 dimensional graph



A corner solution is a special solution to a maximization or minimization problem in which the quantity of one of the input variables is zero or with an extreme value. In non-technical terms, a corner solution is when the chooser is either unwilling or unable to make a trade-off between benefit and cost. Most of the time, corner solutions are not optional, let alone a global optimal solution. To avoid a local or corner solution, we sometimes have to find our input spaces for various input variables first. Then, we choose some evenly spaced intervals for our initial input values. Here is one example.

The PIN measure of information asymmetry is derived from the market microstructure model proposed in *Easley and O'Hara* (1992) and *Easley et al.* (1997). Mathematically, the model specifies that on any day, the likelihood of observing the number of buy trades (ε_b) and the number of sell trades (ε_s) is represented by the following joint Poisson distribution, L :

$$L(\theta|B_i, S_i) = \alpha(1 - \delta)X + \alpha\delta Y + (1 - \alpha)Z \quad (3)$$

The values of X , Y , and Z are defined as:

$$\begin{aligned}
X &= e^{-(\mu+\epsilon_b)} \frac{(\mu+\epsilon_b)^{B_i}}{B_i!} e^{-\epsilon_s} \frac{(\mu+\epsilon_b)^{B_i}}{B_i!} \\
Y &= e^{-\epsilon_b} \frac{\epsilon_b^{B_i}}{B_i!} e^{-\epsilon_s} \frac{(\mu+\epsilon_s)^{S_i}}{S_i!} \quad (4) \\
Z &= e^{-\epsilon_b} \frac{\epsilon_b^{B_i}}{B_i!} e^{-\epsilon_s} \frac{\epsilon_s^{S_i}}{S_i!}
\end{aligned}$$

Where $\theta = [\alpha, \delta, \mu, \epsilon_b, \epsilon_s]$ represents five structural parameters that describe the trading process in each day. Specifically, α denotes the probability that an information event occurs, δ is the probability of a bad event happening, μ is the Poisson parameter for when no information event happens, ϵ_b is the Poisson parameter for when a good event happens (then investors would buy), ϵ_s is the Poisson parameter for when a bad event happens (then investors would sell), B_i is the number of buyer-initiated trades on day i , while S_i is the number of seller-initiated trades.

Since this likelihood objective is not well-behaved, it is sometimes difficult to converge or end up at a corner solution. For many scenarios, optimal solutions depend on the initial input values. *Yan and Zhang (2014)* first found the ranges of those five input values and then designed an even distribution of initial values. Their results showed that such optimally designed input values would reduce corner solutions dramatically.

Expressing various kinds of optimization problems as LPP

LPP stands for **Linear Programming Problem** or linear optimization problem. It is a method of reaching the best (optimal) outcome, such as the maximum expected portfolio return for a given risk level, in a mathematical model whose requirements are represented by linear relationships. LPP has the following

$$\begin{cases} \text{maximize} & C^T X \\ \text{subject to} & Ax \leq b \\ \text{constraints} & x \geq 0 \end{cases} \quad (5)$$

Where $C^T X$ is our objective function, $Ax \leq b$ is a set of constraints, and $x \geq 0$ is another set of constraints. For the constraints, we can have many different formats.

Quadratic optimization

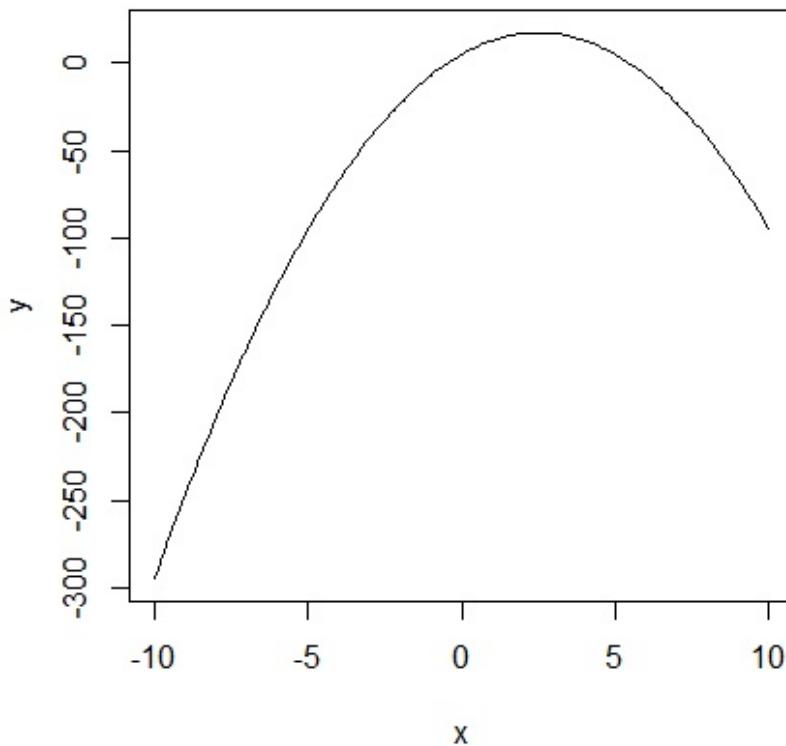
If the highest power is 1, then we call it a linear model. On the other hand, if the highest power is 2, we call it a quadratic function. The R `optim()` function can be used to find a solution for a minimization problem. For example, we have the following objective function:

$$y(x) = ax^2 + bx + c \quad (6)$$

Since there is only one variable, we can solve it manually. Take the first-order derivative and set it to:

```
x<-seq(-10,10,0.1)
a<-2
b<-10
c<-5
y<-a*x^2+b*x+c
plot(x,y,type='l')
```

The related graph is shown here:



From the graph, we know that we could get a maximum y value when x is zero:

```
| y<-20-3.5*x^2
| a<-2
| b<-10
| c<-5
| f<-function(x)-(a*x^2+b*x+c)
```

In the preceding formula, we use a negative function since the R `optim()` function would get a minimum value instead of a maximum one:

```
> optim(0.3,f)
$par
[1] 2.500078
$value
[1] -17.5
$counts
function gradient
      36      NA
$convergence
[1] 0
$message
NULL
```

Optimization in R

In this section, we show one example of optimization for R. The first one is quite simple, which is the maximum of a user's utility, expressed here:

$$U = E(R_p) - \frac{1}{2}A\sigma_p^2 \quad (7)$$

Where U is the utility function, $E(x)$ is the expectation function, R_p is the portfolio return, A is a parameter representing risk preference, and σ_p^2 is the portfolio variance. Obviously, the utility is positively correlated with the expected portfolio return and negatively correlated with the portfolio risk (that is, variance). Note that this is a quadratic function.

Next, we try to use the value-weighted five industries supplied by Fama and French to see how to use the optimization procedure. First, we retrieve the data by using the following code:

```
| path<- "http://canisius.edu/~yany/RData/ff5industries.RData"
| load(url(path))
| head(.ff5industries,3)
```

The output is shown here:

```
> head(.ff5industries,3)
  date VW_Cnsmr VW_Manuf VW_HiTec VW_Hlth VW_Other EW_Cnsmr EW_Manuf EW_HiTec EW_Hlth EW_Other
1 192607    5.43    2.73    1.83    1.77    2.16    1.69    1.42    2.01    2.45    0.21
2 192608    2.76    2.33    2.41    4.25    4.38    2.87    2.27    2.27    6.11    4.53
3 192609    2.16   -0.44    1.06    0.69    0.29   -0.90   -0.33    0.72    0.80   -0.53
> |
```

The first column is the date, while the next five columns are returns for the value-weighted industry portfolios, and the last five columns are for the equal-weighted industry portfolios. `Cnsmr` is for consumer, `Manuf` is for manufacture, `Hlth` is for health care, and `Other` is for other industries. For a more detailed definition, see the link at http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/Data_Library/det_5_ind_port.html. In the following function, we use the value-weighted five industries

as an example:

```
retMatrix<-as.matrix(.ff5industries[,2:6]/100)
n1<-ncol(retMatrix)
w<-rep(1/n1,n1)
A<-1.5
bigValue=100
#
utilityFunction<-function(w){
  portfolioRet<-retMatrix%*%w
  x<-portfolioRet
  loss<-(sum(w)-1)^2*bigValue
  u $\equiv$ (mean(x)-0.5*A*var(x))+loss
  return(u)
}
```

In the preceding function, a negative sign was added to the utility function since the `optim()` function is to minimize. In addition, a so-called loss was added to the objective function. Its value is associated with the deviation of summation of all weights equal to 1. In other words, if the summation of weights is 1, this loss variable has a value of zero. This is one convenient way of adding a constraint. In the following code, we call the R `optim()` function:

```
>optim(w,utilityFunction)
$par
[1] 1.0274904 -0.1901590 0.1474859 0.8867341 -0.8715205
$value
[1] -0.009195987
$counts
function gradient
      502 NA
$convergence
[1] 1
$message
NULL
```

The weights for these five industries are 103%, -19%, 14.7%, 88.7%, and -87.2%. A negative weight indicates a short position. If we have a lower bound and an upper bound, we could specify this with the following code. Note that the related method is `method="L-BFGS-B"`:

```
>optim(w,utilityFunction,lower=-0.1,upper=0.5,method="L-BFGS-B")
$par
[1] 0.50000000 0.01584360 0.08418691 0.50000000 -0.10000000
$value
[1] -0.008587635
$counts
function gradient
      21 21
$convergence
[1] 0
$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

The output shows that the weights of this five-industry portfolio should be (0.5, 0.016, 0.084, 0.5, -0.1). To get more information about this function, we could type `help(optim);` see here:

```
| optim(par, fn, gr = NULL, ...,
|       method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN",
|                  "Brent"),
|       lower = -Inf, upper = Inf,
|       control = list(), hessian = FALSE)
```

Optimization in Python

For optimization, the most frequently used Python function is:

```
| import scipy.optimize as sp
| x=dir(sp.optimize)
| print(x)
```

The output is shown here:

```
In [4]: import scipy as sp
...: x=dir(sp.optimize)
...: print(x)
...
['LbfgsInvHessProduct', 'OptimizeResult', 'OptimizeWarning', 'Tester', '__all__', '__builtins__', '__doc__',
 '__file__', '__name__', '__package__', '__path__', '_basinhopping', '_cobyla', '_differentialevolution',
 '_group_columns', '_hungarian', '_lbfgsb', '_linprog', '_lsq', '_minimize', '_minpack', '_nlls', '_numdiff',
 '_root', '_slsqp', '_spectral', '_trustregion', '_trustregion_dogleg', '_trustregion_ncg', '_zeros',
 'absolute_import', 'anderson', 'approx_fprime', 'basinhopping', 'bench', 'bisect', 'bracket', 'brent',
 'brent', 'brentq', 'broyden1', 'broyden2', 'brute', 'check_grad', 'cobyla', 'curve_fit', 'diagbroyden',
 'differential_evolution', 'division', 'excitingmixing', 'fixed_point', 'fmin', 'fmin_bfgs', 'fmin_cg',
 'fmin_cobyla', 'fmin_l_bfgs_b', 'fmin_ncg', 'fmin_powell', 'fmin_slsqp', 'fmin_tnc', 'fminbound', 'fsolve',
 'golden', 'lbfgsb', 'least_squares', 'leastsq', 'line_search', 'linear_sum_assignment', 'linearmixing',
 'linesearch', 'linprog', 'linprog_verbose_callback', 'lsq_linear', 'minimize', 'minimize_scalar', 'minpack',
 'minpack2', 'moduleTNC', 'newton', 'newton_krylov', 'nlls', 'nonlin', 'optimize', 'print_function', 'ridder',
 '_root', 'rosen', 'rosen_der', 'rosen_hess', 'rosen_hess_prod', 's', 'show_options', 'slsqp', 'test', 'tnc',
 '_zeros']
```

To find the usage of a function called `minimize`, we have the following code:

```
| from scipy.optimize import minimize
| help(minimize)
```

The output is shown here. To save space, only the top part is presented:

Help on function `minimize` in module `scipy.optimize._minimize`:

```
minimize(fun, x0, args=(), method=None, jac=None, hess=None, hessp=None,
bounds=None, constraints=(), tol=None, callback=None, options=None)
    Minimization of scalar function of one or more variables.
```

In general, the optimization problems are of the form::

minimize $f(x)$ subject to

$g_i(x) \geq 0, \quad i = 1, \dots, m$
 $h_j(x) = 0, \quad j = 1, \dots, p$

where x is a vector of one or more variables.

`` $g_i(x)$ '' are the inequality constraints.

`` $h_j(x)$ '' are the equality constraints.

Optionally, the lower and upper bounds for each element in x can also be specified using the `bounds` argument.

Parameters

The next example is from <https://scipy.org/>. The objective function is shown here:

$$f(x) = \sum_{i=2}^5 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2 \quad (8)$$

It is the Rosenbrock function with five input variables. Since each item on the right-hand side is non-negative, we would like to find a set of input variables $(x_1, x_2, x_3, x_4, x_5)$ to minimize the objective function:

```
import numpy as np
from scipy.optimize import minimize
def rosen(x):
    return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)
#
x0 = np.array([1.3, 0.7, 0.8, 1.9, 1.2])
solution= minimize(rosen, x0, method='nelder-mead',options={'xtol': 1e-8, 'disp': True})
print(solution.x)
```

The code of $x[1:]$ means the values of x from the second item to the last one, while the code of $x[:-1]$ is the values included in x except the last one. The output is shown here:

```
Optimization terminated successfully.  
    Current function value: 0.000000  
    Iterations: 339  
    Function evaluations: 571  
[ 1.  1.  1.  1.  1.]
```

Optimization in Julia

The next example is from the web page <https://jump.readthedocs.io/en/release-0.2/jump.html>. The code is given here:

```
using JuMP
using ECOS
m= Model(solver =ECOSSolver())
@variable(m, 0 <= x <= 2 )
@variable(m, 0 <= y <= 30 )
@setObjective(m, Max, 5x + 3*y )
@addConstraint(m, 1x + 5y <= 3.0 )
print(m)
status = solve(m)
println("Objective value: ", getObjectiveValue(m))
println("x = ", getValue(x))
println("y = ", getValue(y))
```

First, the objective function is shown here:

```
julia> print(m)
Max 5 x + 3 y
Subject to
    x + 5 y <= 3
    0 <= x <= 2
    0 <= y <= 30
```

The output is given in the following screenshot:

```
julia> status = solve(m)

ECOS 2.0.2 - (C) embotech GmbH, Zurich Switzerland, 2012-15. Web: www.embotech.com/ECOS

It    pcost      dcost      gap    pres   dres    k/t    mu    step   sigma    IR    |    BT
0   -1.252e+01  -2.013e+02  +5e+02  3e-01  3e-01  1e+00  8e+01    ---    ---    1  1  -  |  -  -
1   -9.764e+00  -1.482e+01  +1e+01  1e-02  8e-03  2e-01  2e+00  0.9703  2e-03  0  0  0  |  0  0
2   -1.080e+01  -1.123e+01  +1e+00  7e-04  7e-04  2e-02  2e-01  0.9005  4e-03  0  0  0  |  0  0
3   -1.060e+01  -1.061e+01  +3e-02  2e-05  2e-05  1e-03  6e-03  0.9800  5e-03  0  0  0  |  0  0
4   -1.060e+01  -1.060e+01  +4e-04  2e-07  2e-07  1e-05  7e-05  0.9890  1e-04  1  0  0  |  0  0
5   -1.060e+01  -1.060e+01  +4e-06  2e-09  2e-09  1e-07  8e-07  0.9890  1e-04  1  0  0  |  0  0
6   -1.060e+01  -1.060e+01  +5e-08  2e-11  2e-11  1e-09  8e-09  0.9890  1e-04  1  0  0  |  0  0

OPTIMAL (within feastol=2.1e-11, reltol=4.4e-09, abstol=4.7e-08).
Runtime: 0.045317 seconds.

:Optimal

julia> println("Objective value: ", getObjectiveValue(m))
Objective value: 10.600000000335104

julia> println("x = ", getValue(x))
x = 2.00000000720887

julia> println("y = ", getValue(y))
y = 0.19999999891022338
```

For the code of `m=Model(solver=Ecossolver())` in the preceding program, the solver contained in the package called `ecos` was used. The following table shows various solver functions embedded in different Julia packages:

Solver	Julia Package	solver=	License	LP	SOCP	MILP	NLP	MINLP	SDP
Artelys Knitro	KNITRO.jl	KnitroSolver()	Comm.				X	X	
BARON	BARON.jl	BaronSolver()	Comm.				X	X	
Bonmin	AmplNLWriter.jl	BonminNLSolver() *	EPL	X		X	X	X	
CoinOptServices.jl	OsilBonminSolver()								
Cbc	Cbc.jl	CbcSolver()	EPL			X			
Clp	Clp.jl	ClpSolver()	EPL	X					
Couenne	AmplNLWriter.jl	CouenneNLSolver() *	EPL	X		X	X	X	
CoinOptServices.jl	OsilCouenneSolver()								
CPLEX	CPLEX.jl	CplexSolver()	Comm.	X	X	X			
ECOS	ECOS.jl	ECOSSolver()	GPL	X	X				
FICO Xpress	Xpress.jl	XpressSolver()	Comm.	X	X	X			
GLPK	GLPKMath...	GLPKSolver[LP MIP]()	GPL	X		X			
Gurobi	Gurobi.jl	GurobiSolver()	Comm.	X	X	X			
Ipopt	Ipopt.jl	IpoptSolver()	EPL	X			X		
MOSEK	Mosek.jl	MosekSolver()	Comm.	X	X	X	X		X
NLopt	NLopt.jl	NLoptSolver()	LGPL				X		
SCS	SCS.jl	SCSSolver()	MIT	X	X				X

Julia packages with solver

The notations in the table are defined here:

- LP: Linear programming
- SOCP: Second-order conic programming (including problems with convex quadratic constraints and/or objectives)
- MILP: Mixed-integer linear programming
- NLP: Nonlinear programming
- MINLP: Mixed-integer nonlinear programming
- SDP: Semidefinite programming

The source of the previous table is here : <http://jump.readthedocs.io/en/latest/installation.html>.

Now, consider the following equation:

$$f(x) = (1 - x_1)^2 + 200(x_2 - x_1^2)^2 \quad (9)$$

The following Julia program is to find a solution for the preceding function:

```
| using Optim
| f(x) = (1.0 - x[1])^2 + 200.0 * (x[2] - x[1]^2)^2
| optimize(f, [0.0, 0.0])
```

The output is shown in the following screenshot:

```
julia> using Optim
julia> f(x) = (1.0 - x[1])^2 + 200.0 * (x[2] - x[1]^2)^2
f (generic function with 1 method)

julia> optimize(f, [0.0, 0.0])
Results of Optimization Algorithm
  * Algorithm: Nelder-Mead
  * Starting Point: [0.0,0.0]
  * Minimizer: [0.9999855177658813,0.9999718307694201]
  * Minimum: 3.361490e-10
  * Iterations: 66
  * Convergence: true
    * √(Σ(yᵢ - yᵢ̂)²)/n < 1.0e-08: true
    * Reached Maximum Number of Iterations: false
  * Objective Calls: 126
```

The next example is from Julia's website, <http://juliansolvers.github.io/Optim.jl/test/user/minimization/>. The code is given here:

```
| using Optim
function g!(s, x)
  s[1] = -2.0*(1.0-x[1])-400.0*(x[2]-x[1]^2)*x[1]
  s[2] = 200.0*(x[2]-x[1]^2)
end
lower = [1.25, -2.1]
upper = [Inf, Inf]
initial_x = [2.0, 2.0]
od = OnceDifferentiable(f, g!, initial_x)
results = optimize(od, initial_x, lower, upper, Fminbox{GradientDescent}())
```

The output is shown here:

```
Results of Optimization Algorithm
* Algorithm: Fminbox with Gradient Descent
* Starting Point: [2.0,2.0]
* Minimizer: [1.6557274658535348,2.7427019545260074]
* Minimum: 4.303003e-01
* Iterations: 3
* Convergence: true
  * |x - x'| ≤ 1.0e-32: true
    |x - x'| = 0.00e+00
  * |f(x) - f(x')| ≤ 1.0e-32 |f(x)|: true
    |f(x) - f(x')| = 0.00e+00 |f(x)|
  * |g(x)| ≤ 1.0e-08: false
    |g(x)| = 4.64e-01
* Stopped by an increasing objective: true
* Reached Maximum Number of Iterations: false
* Objective Calls: 322
* Gradient Calls: 322
```

Optimization in Octave

The Octave package called `optim` can be downloaded easily from <https://octave.sourceforge.io/optim/>. The following screenshot indicates the version of the package. Note this was the version on March 1, 2018:

community	
Package Version:	1.5.2
Last Release Date:	2016-09-18
Package Author:	various authors
Package Maintainer:	Olaf Till <i7tiol@t-online.de>
License:	<u>GPLv3+, modified BSD, public domain</u>

After downloading and installing the package (see [Chapter 6, Managing Packages](#), for more detail on how to install an Octave package), we could use the following command lines to see some functions:

```
| pkg load optim  
| pkg describe -verbose optim
```

The following screenshot shows the output. To save space, only the top part is shown:

```
Package name:  
    optim  
Version:  
    1.5.2  
Short description:  
    Non-linear optimization toolkit.  
Status:  
    Loaded  
---  
Provides:  
Optimization  
    nelder_mead_min  
    nrm  
    line_min  
    powell  
    fmins  
    adsmax  
    mdsmax  
    nmsmax  
    bfgsmin  
    samin  
    battery  
    cg_min  
    de_min  
    nonlin_min  
    brent_line_min  
Data fitting  
    lsqlin  
    expfit
```

To find the usage of a specific function, we could use the `help()` function:

```
| help fminsearch
```

The related output is shown in the following screenshot:

```

>> help fminsearch
'fminsearch' is a function from the file C:\Octave\Octave-4.0.0\share\oc

-- Function File: X = fminsearch (FUN, X0)
-- Function File: X = fminsearch (FUN, X0, OPTIONS)
-- Function File: [X, FVAL] = fminsearch (...)
    Find a value of X which minimizes the function FUN.

The search begins at the point X0 and iterates using the Nelder &
Mead Simplex algorithm (a derivative-free method). This algorithm
is better-suited to functions which have discontinuities or for
which a gradient-based search such as 'fminunc' fails.

Options for the search are provided in the parameter OPTIONS using
the function 'optimset'. Currently, 'fminsearch' accepts the
options: "TolX", "MaxFunEvals", "MaxIter", "Display". For a
description of these options, see 'optimset'.

On exit, the function returns X, the minimum point, and FVAL, the
function value thereof.

```

Example usages:

```

fminsearch (@(x) (x(1)-5).^2+(x(2)-8).^4, [0;0])
fminsearch (inline ("(x(1)-5).^2+(x(2)-8).^4", "x"), [0;0])

```

See also: fminbnd, fminunc, optimset.

From the preceding information, we know that the `fminsearch()` function would minimize our objective function with a given set of initial values with a set of options. Next, we have two examples. For the first example, the objective function is given here:

$$f(x) = 50(x_i^2 - x_2)^2 + (x_i - 3)^2 \quad (10)$$

The code and related output is shown here:

```

|>> fun = @(x)50*(x(1)^2-x(2))^2 + (x(1)-3)^2;
|>> x0 = [0,0];
|>> x = fminsearch(fun,x0)
|x
| 3.0000 9.0000

```

For the next example, its objective function is given here:

$$f(x) = \sum_{i=-5}^5 e^{-(x_1-x_2)^2} - 2x_1^2 \cos(x_2) \sin(2x_2) \quad (11)$$

To see each step, we have the following code:

```

OPTIONS = optimset('Display','iter');
function f = fun2(x)
    f = 0;
    for k = -5:5
        f = f + exp(-(x(1)-x(2))^2 - 2*x(1)^2)*cos(x(2))*sin(2*x(2));
    end
endfunction
x0 = [0.5,-0.5];
[x,fval] = fminsearch(@fun2,x0,OPTIONS)

```

The first line is for the settings for the optimization procedure. The word `Display` means to display all the intermediate steps while `iter` means integration. The output is shown here. To save space, we show the top and bottom parts. The top part is given first:

```

>> [x,fval] = fminsearch(@fun2,x0,OPTIONS)
f(x0) = 1.8125e+000
Iter. 1, how = initial nf = 3, f = 1.8125e+000 (0.0%)
Iter. 2, how = contract, nf = 5, f = 1.8125e+000 (0.0%)
Iter. 3, how = reflect, nf = 6, f = 1.8125e+000 (0.0%)
Iter. 4, how = reflect, nf = 8, f = 4.7390e+000 (161.5%)

```

Here is the bottom part:

```

Iter. 33, how = contract, nf = 64, f = 6.8781e+000 (0.0%)
Iter. 34, how = contract, nf = 66, f = 6.8781e+000 (0.0%)
Simplex size 9.4335e-005 <= 1.0000e-004...quitting
x =
-0.16954 -0.50866

fval = -6.8781

```

Example #1 – stock portfolio optimization

Sometimes we refer to single-period portfolio optimization as Markowitz portfolio optimization. Our input datasets include the expected returns, the standard deviations, and the correlation matrix between financial assets, and our output will be an efficient frontier formed by those assets. In the rest of the chapter, we will use historical returns to represent expected returns and use the historical correlation in place of expected correlation.

In the following examples, we use an R package called `fPortfolio`. We use the following code to install the package:

```
| install.packages("fPortfolio")
```

To load various embedded datasets, we use the `data()` function (see the following example code):

```
| library(fPortfolio)
| data(GCCINDEX.RET)
| dim(GCCINDEX.RET)
| [1] 824 11
```

The following table lists the embedded datasets:

#	Name	Dimension	#	Name	Dimension
1	ECON85	[1] 304 12	7	SMALLCAP	[1] 60 22
2	ECON85LONG	[1] 304 19	8	GCCINDEX.RET	[1] 824 11
3	GCCINDEX	[1] 825 11	9	SPISECTOR.RET	[1] 2198 10

4	SPISECTOR	[1] 2216 10	10	SWX.RET	[1] 1916 6
5	SWX	[1] 1917 6	11	LPP2005.RET	[1] 377 9
6	LPP2005	[1] 377 9	12	SMALLCAP.RET	[1] 60 22

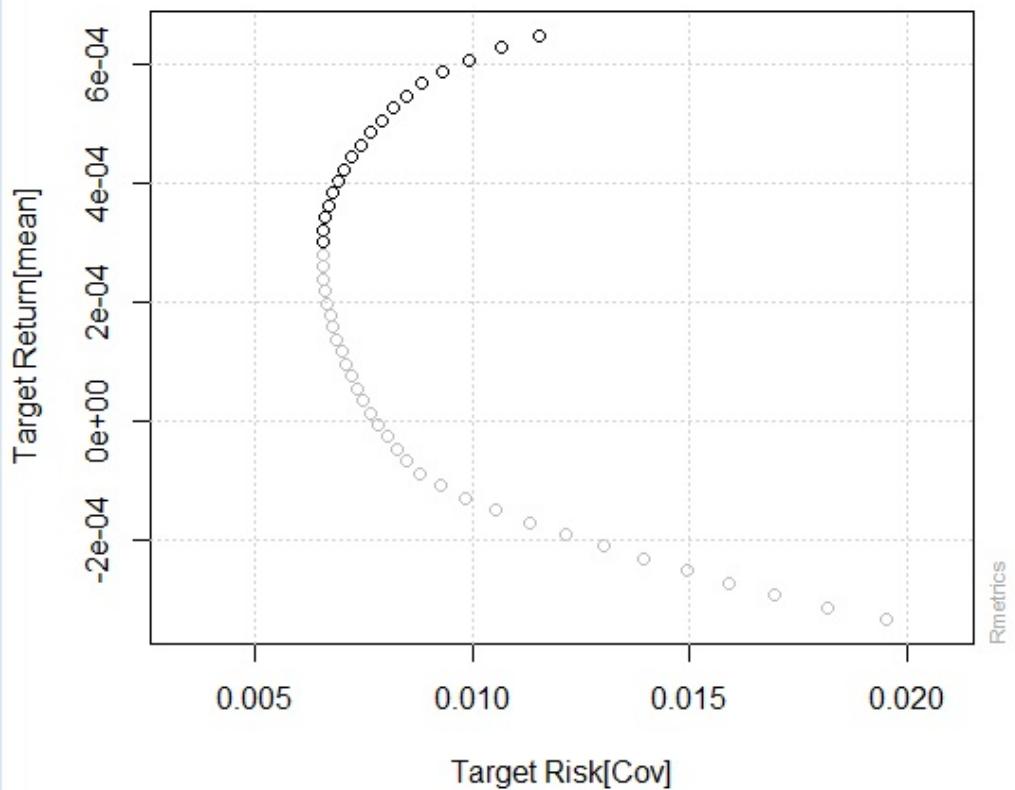
A list of datasets in the fPortfolio R package

To get the efficient frontier, the connections of all efficient portfolios, we have the following code:

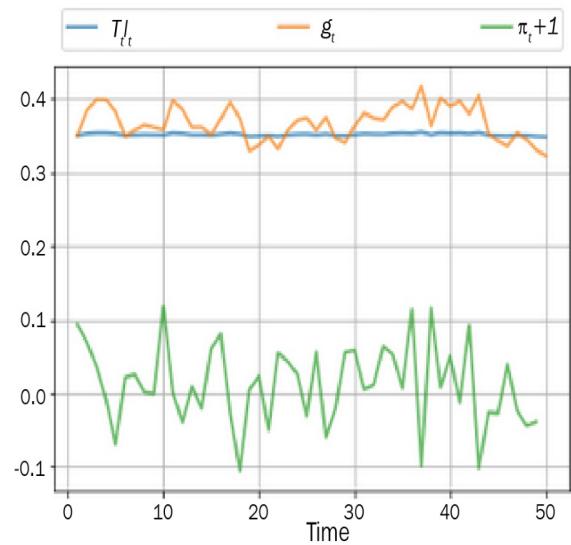
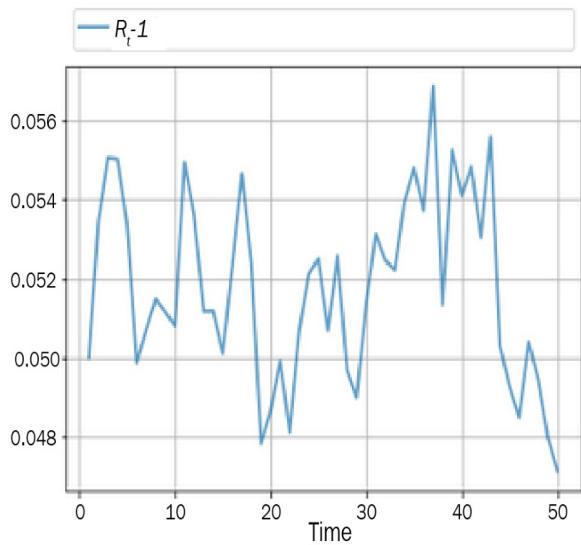
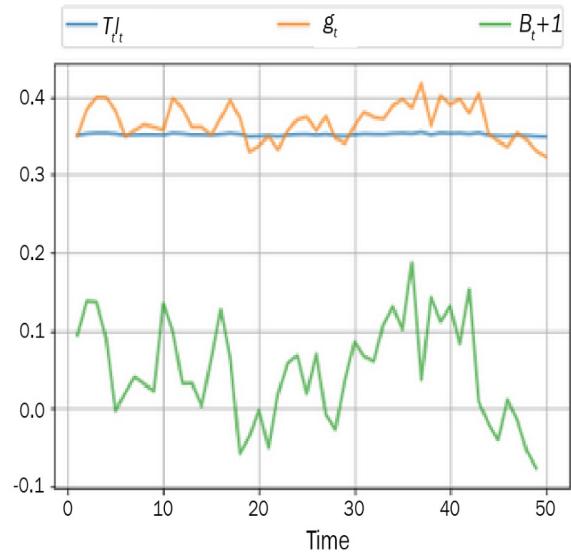
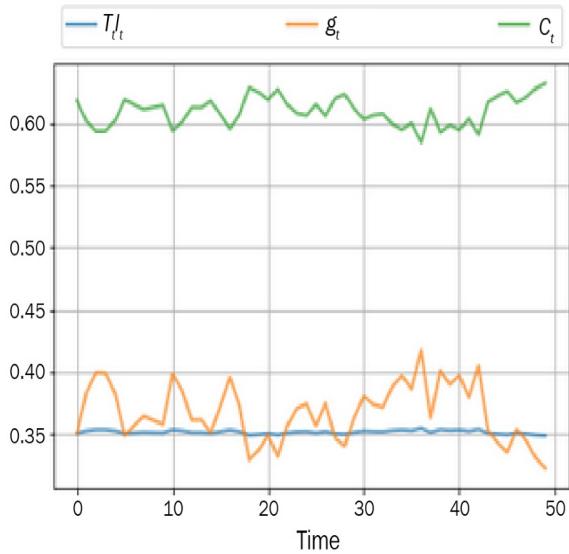
```
library(fPortfolio)
data(GCCINDEX.RET)
retMatrix<-GCCINDEX.RET
frontier=portfolioFrontier(as.timeSeries(retMatrix))
frontierPlot(frontier)
grid()
```

In this code, we first load the R package called `fPortfolio`. Then, `data(GCCINDEX.RET)` loads the dataset. The related plot is shown here:

Efficient Frontier



```
import numpy as np from numpy import array beta= 1 / 1.05  
rho, mg = .7, .35  
A = np.identity(2) A[0, :] = rho, mg * (1-rho) C = np.zeros((2, 1)) C[0,0] =  
np.sqrt(1 - rho**2) * mg / 10.  
Sg = array((1, 0)).reshape(1, 2) Sd = array((0, 0)).reshape(1, 2) Sb = array((0,  
2.135)).reshape(1, 2) Ss = array((0, 0)).reshape(1, 2)  
economy=Economy(beta=beta,Sg=Sg,Sd=Sd,Sb=Sb,Ss=Ss,discrete=False,proc=  
(A, C)) T = 50  
path = compute_paths(T, economy) gen_fig_1(path)
```



Packages for optimization in R

There are many R packages available for various types of optimization, such as `optimization`, `M1BayesOpt`, `rgenoud`, `colf`, and `mize`. The following table offers a partial list:

#	Name	Description
1	dtwclust	Time Series Clustering Along with Optimizations for the Dynamic Time Warping Distance
2	CVXR	Disciplined Convex Optimization
3	IROmiss	Imputation Regularized Optimization Algorithm
4	subplex	Unconstrained Optimization using the Subplex Algorithm
5	GPareto	Gaussian Processes for Pareto Front Estimation and Optimization
6	OOB	Optimistic Optimization in R
7	ROI	R Optimization Infrastructure
8	lbreg	Log-Binomial Regression with Constrained Optimization

9	<code>PEIP</code>	Geophysical Inverse Theory and Optimization
10	<code>dfoptim</code>	Derivative-Free Optimization
11	<code>SPOT</code>	Sequential Parameter Optimization Toolbox

A partial list of R packages for optimization

In addition, we can check the task view related to optimization. To do this, follow these steps:

1. Go to the R home page at <https://r-project.org>
2. Click Download under CRAN
3. Choose a mirror location close to your location
4. Click Task View on the left-hand side

5. Click Optimization on the list, see the top part here:

CRAN Task View: Optimization and Mathematical Programming

Maintainer: Stefan Theussl and Hans W. Borchers

Contact: R-optimization at mailbox.org

Version: 2018-03-02

URL: <https://CRAN.R-project.org/view=Optimization>

This CRAN task view contains a list of packages which offer facilities for solving optimization problems. Although every regression model in statistics solves an optimization problem they are not part of this view. If you are looking for regression methods, the following views will contain useful starting points:

[Multivariate](#), [SocialSciences](#), [Robust](#) among others. The focus of this task view is on [Optimization Infrastructure Packages](#), [General Purpose Continuous Solvers](#), [Mathematical Programming Solvers](#), and [Specific Applications in Optimization](#). Packages are categorized in these four sections.

Many packages provide functionality for more than one of the subjects listed at the end of this task view. E.g., mixed integer linear programming solvers typically offer standard linear programming routines like the simplex algorithm. Therefore following each package description a list of abbreviations describes the typical features of the optimizer (i.e., the problems which can be solved). The full names of the abbreviations given in square brackets can be found at the end of this task view under [Classification According to Subject](#).

If you think that some package is missing from the list, please let us know.

[Optimization Infrastructure Packages](#)

There are over 100 R packages included in the preceding task view. To install all at once, we can issue the following commands:

```
| install.packages("ctv")
| library("ctv")
| install.views("Optimization")
```

Packages for optimization in Python

From [Chapter 6, Managing Packages](#), we know that to find all Python packages, we go to the website at <https://pypi.python.org/>. The following table shows a list of Python packages/models related to optimization after we type `optimization` as the keyword:

Package	Wt*	Description
heuristic-optimization 0.4.3	7	Heuristics for derivative-free optimization
streams-optimization 1.0.0.dev3	7	A library for LHCb trigger/stripping streams optimization
adjointShapeOptimizationFlux 1.0	6	Python frontend of the adjointShapeOptimizationFoam
bayesian-optimization 0.6.0	6	Bayesian Optimization package
emzed_optimizations 0.6.0	6	particular optimizations for speeding up emzed
scikits.optimization 0.3	6	A python module for numerical optimization

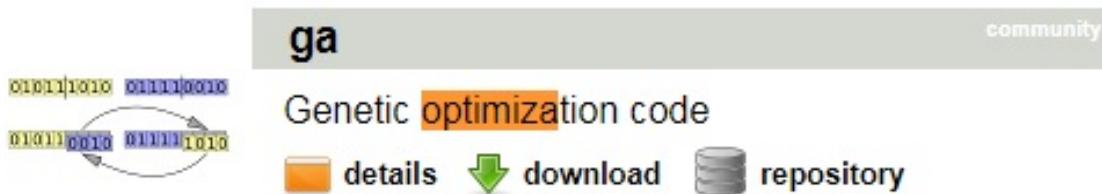
asprin 3.0.2	5	Qualitative and quantitative optimization in answer set programming
bolib 0.19.11	5	Python library for Bayesian Optimization
brent-search 1.0.32	5	Brent's method for univariate function optimization
clientMOE 0.1	5	Metric Optimization Engine
cma 2.5.3	5	CMA-ES, Covariance Matrix Adaptation Evolution Strategy for nonlinear numerical optimization in Python
coopr.opt 2.12.2	5	Coopr generic optimization interfaces
coopr.os 1.0.4	5	Coopr interfaces to the COIN-OR Optimization Services project
cro 0.0.5.0	5	Coral Reef Optimization (CRO) Algorithm
dfoalgos 0.5	5	Derivative-free optimization algorithms
dillinger 1.0.0.dev1	5	Bayesian optimization for iterated multi-armed bandit experiments

downhill 0.4.0	5	Stochastic optimization routines for Theano
dy 0.0.1	5	Dynamic Optimization/Dynamic Programming
fst-pso 1.3.2	5	Fuzzy Self-Tuning PSO global optimization library

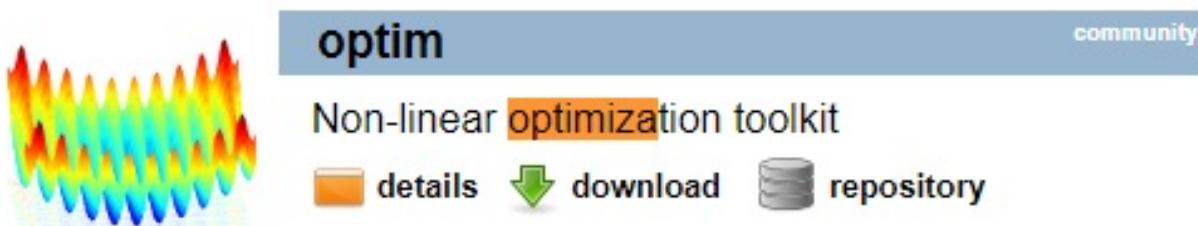
A partial list of the Python packages for optimization

Packages for optimization in Octave

To find packages for optimization in Octave, we go to the web page at <https://octave.sourceforge.io/packages.php>. Then, we can search these packages by using the keyword `optimization`; see the first package called `ga` in this screenshot:



The second one is the `optim` package:



Packages for optimization in Julia

Similarly, for packages for optimization in Julia, we go to the web page at <https://pkg.julialang.org/>. Then, we can search these packages by using the keyword `optimization`; see one package called `JuMP` in this screenshot:

The screenshot shows the JuMP package page on the Julia Packages website. The package name "JuMP" is displayed prominently in blue. Below it, a brief description reads: "Modeling language for Mathematical Optimization (linear, mixed-integer, conic, semidefinite, nonlinear)". To the right, there is detailed information: "MPLv2 license / Owner: JuliaOpt / permalink", "456 ★ / 15 J / 40 ↑", "Julia v0.6: 0.18.0 (7 months ago) / Tests pass.", and "Julia v0.7: 0.18.0 (7 months ago) / Tests fail.". A small green checkmark icon is next to the "Tests pass" status.

There are about 45 matches for the keyword `optimization`. To save space, we won't show the other packages. This web page, titled *Optimization packages for Julia language*, might be quite useful: <http://www.juliaopt.org/>.

Summary

In this chapter, we have discussed several topics around optimization, such as general issues for optimization problems, expressing various kinds of optimization problems as LPPs, and quadratic optimization. Several examples were offered to make our discussion more practice-oriented, such as how to choose an optimal stock portfolio, optimize wealth and resources to promote sustainable development, and how much the government really should tax. In addition, we introduced several packages for optimization in R, Python, Julia, and Octave.

In the next chapter, we will discuss unsupervised learning. In particular, we will explain hierarchical clustering and k-means clustering. For R and Python, we will explain in detail several related packages. For R, we will discuss `Rattle`, `randomUniformForest`, and `Rmixmod`. For Python, we will cover `SciPy`, `Contrastive`, `milk`, `scikit-learn`, and `skleam`. Several real-world examples will be used to illustrate the application of these R and Python packages.

Review questions and exercises

1. What does optimization mean?
2. What is an LPP? What are its uses?
3. What is the difference between a global solution and a local solution?
4. In what situations would our LPP program not converge? Give a few simple examples and possible solutions.
5. Explain why we have the following weird result:

```
> f<-function(x)-2*x^2+3*x+1
```

```
> optim(13,f)
```

```
$par
```

```
[1] 2.352027e+75
```

```
$value
```

```
[1] -1.106406e+151
```

```
$counts
```

```
function gradient
```

```
502      NA
```

```
$convergence
```

```
[1] 1
```

```
$message
```

```
NULL
```

6. What does quadratic equation mean?
7. From where could we search all the R packages targeting optimization issues?
8. What is the usage of the task view related to optimization?
9. According to the related task view, how many R packages are associated with optimization, and how do we install them all at once?
10. From the Prof. French Data Library, download the return data for 10 industries. Then, choose an optimal portfolio by applying the following utility function for five types of investors with $A=1, 3, 5, 20$, and 100:

$$U = E(R_p) - \frac{1}{2} A \sigma_p^2$$

Where U is the utility function, $E()$ is the expectation function, R_p is the portfolio return, A is a parameter representing risk preference, and σ_p^2 is the portfolio variance. Note the source of data is at http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html.

11. For Octave, find two examples of using the `leasqr` (least square) function. You can use the following code to get information about this function:

```
>> pkg describe -verbose optim

>> help leasqr
```

12. How many functions could we find for the Octave package called `optim`?
13. For Octave, download a package called `ga`. Find out the uses of several functions included in the package.
14. How many R packages are installed after you issue the following three lines:

```
>install.packages("ctv")

>library("ctv")

>install.views("Optimization")
```

15. Install a Julia package called `JUMP` and find a few examples its embedded functions.

Unsupervised Learning in Anaconda

Before discussing unsupervised learning, it might be a good idea to introduce supervised learning since most of us will be familiar with functions discussed in the previous chapters. For a function of $y=f(x)$, usually we have values for independent variables of x_1, x_2, \dots, x_n and a set of corresponding values for a dependent variable of y . In previous chapters, we have discussed various types of functions, such as the single-factor linear model. Our task is to figure out the format of the function, given a set of input values. For supervised learning, we have two datasets: the **training data** and **test data**. For the training dataset, it has a set of input variables and related output values (also called a **supervisory signal**). A supervised learning algorithm analyzes the training data and produces an inferred function. Then, we apply this inferred function to map the given test dataset.

Unlike supervised learning, we don't have y for unsupervised learning. In other words, unsupervised learning is used to draw inferences from datasets consisting of input data without labelled responses. In a sense, for supervised learning, we have both x and y , while for unsupervised learning, we have x only. In other words, for one algorithm, we have both input and output, while for the other, we have inputs only. The most common unsupervised learning method is Cluster Analysis, which is used for exploratory data analysis to find hidden patterns or groupings in data.

In this chapter, the following topics will be covered:

- Introduction to unsupervised learning
- Hierarchical clustering
- k-means clustering
- Introduction to Python packages: `scipy`, `contrastive`, and `sklearn` (`scikit-learn`)
- Introduction to R packages: `rattle`, `randomUniformForest`, and `Rmixmod`
- Implementation using Julia
- Task view for Cluster Analysis

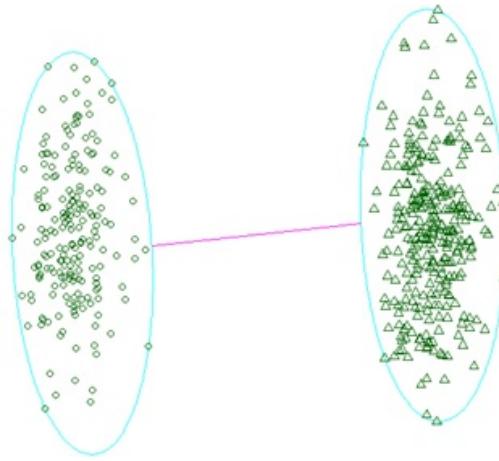
Introduction to unsupervised learning

For unsupervised learning, we try to reorganize data or classify it into different groups based on certain traits or characteristics. For this purpose, we can use certain rules to categorize our dataset. For example, we could classify them into different groups based on investors' characteristics, such as age, education level, background, job types, living city, profession, salary level, and house ownership. For instance, they could be classified into four types of investors: aggressive, risk averse, risk neutral, and extremely risk averse. After that, financial institutions could design and market specific financial products targeting different groups.

To plan an equitable income tax policy, governments could classify potential taxpayers based on various criteria, such as income level and whether a person has a certain disability. Then, they could design distinct tax plans to target various groups with varying social welfare status. Such a tax policy could be more equitable than a simple progressive tax scheme.

Another example is that investors could have over 5,000 stocks available to buy. It might be a good idea to group them according to various benchmarks, such as riskiness, profitability, familiarity, reputation, local/national/international, social responsibility, and transaction cost. After that, financial institutions could form different mutual funds which could be sold to different investors.

The following diagram is used to summarize the idea: grouping data according to certain criteria. The data is drawn from two normal distributions with the same standard deviation, but with different means. Later in the chapter, we will show the related code:



The logic for clustering or grouping is that the distance between group members is less than that between groups. In other words, the similarity among members within each group is higher than that between groups. For instance, if we have ZIP codes of many houses, we could estimate the distance between each pair of houses and classify them into different communities. For numerical values, such as (x_1, y_1) , (x_2, y_2) , the difference could be defined by the following formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (1)$$

The following code generates a graph that shows the distance between two points. This is also called Euclidean distance:

```

> a<-c(2,5)
> b<-c(3,7)
> z<-rbind(a,b)
> dist(z, method = "euclidean")
      a
b 2.236068
> plot(z,lwd=20)

```

In the previous code, `method="euclidean"` could be omitted since it is the default setting. The distance is `2.236` and the graph is ignored for brevity. The potential phrases for the method variable are `euclidean`, `maximum`, `manhattan`, `canberra`, `binary`, and `minkowski`. The general form is called the Minkowski distance, as shown here:

$$d = [(x_2 - x_1)^n + (y_2 - y_1)^n]^{1/n} \quad (2)$$

Obviously, when n takes a value of 2, Equation (2) is the same as Equation (1).

On the other hand, when $p = 1$, this is equivalent to the Manhattan distance, as shown in the following:

$$d = |x_2 - x_1| + |y_2 - y_1| \quad (3)$$

When x and y differ in terms of sizes, we could scale them by using the following formulae:

$$\begin{cases} x_i^{new} = \frac{x_i - \bar{x}}{\sigma_x} \\ y_i^{new} = \frac{y_i - \bar{y}}{\sigma_y} \end{cases} \quad (4)$$

The previous scaling method is based on the assumption that x and y are independent, where x_i^{new} (y_i^{new}) is the new data point for x (y), \bar{x} (\bar{y}) is the mean for x (y), and σ_x (σ_y) is the standard deviation of x (y). If researchers put more weight on the direction of the data rather than the magnitude, then the cosine distance could be used. The cosine of two non-zero vectors can be derived by using the Euclidean dot product formula:

$$a * b = |a| * |b| \cos(\theta) \quad (5)$$

where θ is the angle between those two vectors. Also, we could view $\cos(\theta)$ as a scaling factor, shown here:

```
> angles<-seq(0,380,30)
> y<-cos(angles)
> y2<-round(y,2)
> z<-cbind(angles,y2)
> colnames(z)<-c("ANGLE", "cos(angle)")
> print(z)
      ANGLE cos(angle)
[1,]    0     1.00
[2,]   30     0.15
[3,]   60    -0.95
[4,]   90    -0.45
[5,]  120     0.81
[6,]  150     0.70
[7,]  180    -0.60
[8,]  210    -0.88
[9,]  240     0.33
[10,] 270     0.98
[11,] 300    -0.02
[12,] 330    -0.99
[13,] 360    -0.28
```

Obviously, when the x and y vector has the exact same direction, $\cos(\theta)$ has the

highest value of 1. When they have opposite directions, $\cos(\theta)$ has a negative value.

Given two vectors A and B, the cosine similarity, $\cos(\theta)$, is represented using a dot product and magnitude in the following:

$$\text{similarity} = \cos(\theta) = \frac{a * b}{|a| * |b|} = \frac{\sum_{i=1}^n a_i b_i}{\sqrt{\sum_{i=1}^n a_i} * \sqrt{\sum_{i=1}^n b_i}} \quad (6)$$

For categorical data points, they could not be ranked because of non-numeric values. For these cases, we can calculate the following similarity index:

$$\text{similarity_Index} = \frac{n_{matched}}{n_{total}} \quad (7)$$

Where $n_{matched}$ is the number of matched treats and n_{total} is the number of total treats. When we have both categorical and numeric data, we can estimate both types of distances first when choosing a weighted final value:

$$d = w_1 d_1 + w_2 d_2$$

Assuming d_{num} is the distance based on the numerical data and d_{cat} is the distance based on the categorical data, we have the following equivalent:

$$d = w_{num} * d_{num} + (1 - w_{num}) * d_{cat} \quad (8)$$

where w_{num} is the weight of the numerical value.

Assume that we have two sets of data, X and Y. For the X set, we have x_1, x_2, \dots, x_n , while for the Y set, we have y_1, y_2, \dots, y_m . In other words, for X, there are n observations, while for Y, there are m observations. For a pair of clusters, we have several ways to define their linkages (see the following table):

Linkage	Math formula	Description
		Shortest distance between the

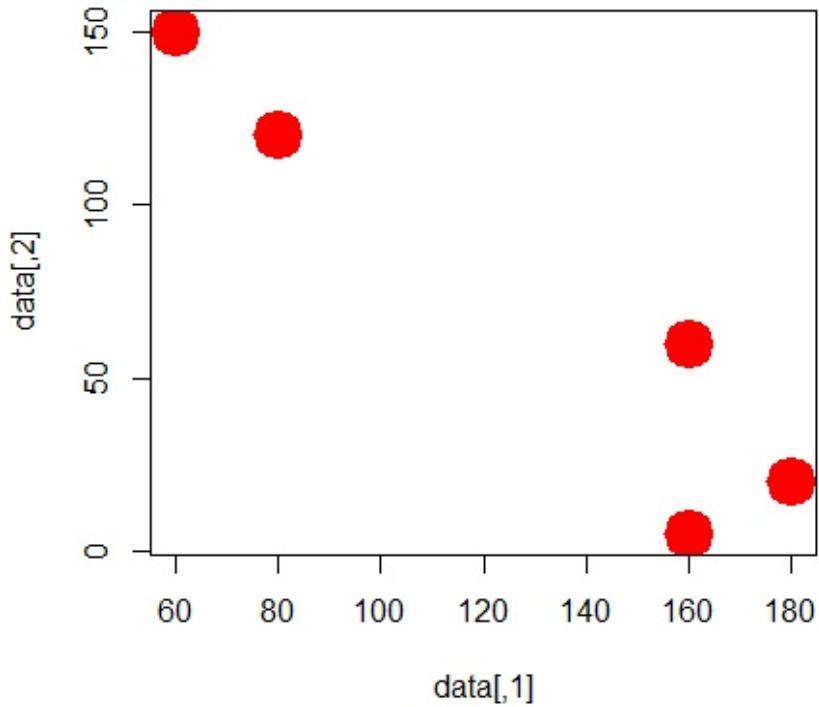
Single	$d = \underbrace{\text{Min}}_{i,j} d(x_i, y_j)$	closest members of the two clusters
Complete	$d = \underbrace{\text{Max}}_{i,j} d(x_i, y_j)$	Longest distance between the members that are farthest apart (most dissimilar)
Average	$d = \frac{1}{n * m} \sum_{i=1}^n \sum_{j=1}^m d(x_i, y_j)$	Step 1: Get distances between all pairs Step 2: Calculate the average
Centroid Method	$d = d(\bar{x}, \bar{y})$	Step 1: Find the mean vector locations Step 2: Get the distance between the two locations

Table 8.1 Four types of linkages between two clusters

Let's use some hypothetical values to see how to get those linkages. To make our estimate easier to understand, we have just five values (see the following code):

```
|> data <- rbind(c(180,20), c(160,5), c(60, 150), c(160,60), c(80,120))
|> plot(data, col = "red", lwd = 20)
```

Refer to the following output:



For the centers of these two clusters, we could guess that the first one should be about (65,140), while the second one should be around (170, 40) (see the following output code):

```
> library(cluster)
> data <- rbind(c(180,20), c(160,5), c(60, 150), c(160,60), c(80,120))
> output<-clara(data,2)
> output$clustering
[1] 1 1 2 1 2
> output$medoids
[,1] [,2]
[1,] 180   20
[2,]  80   120
```

Hierarchical clustering

In hierarchical clustering, the two most similar clusters are combined and continue to combine until all objects are in the same cluster. Hierarchical clustering produces a tree called a **dendrogram** that shows the hierarchy of the clusters.

To show this concept, let's start by looking at the dataset called `animals` embedded in the R package called `cluster`:

```
> library(cluster)
> data(animals)
> dim(animals)
[1] 20  6
> head(animals)
   war fly ver end gro hai
ant  1   1   1   1   2   1
bee  1   2   1   1   2   2
cat   2   1   2   1   1   2
cpl   1   1   1   1   1   2
chi   2   1   2   2   2   2
cow   2   1   2   1   2   2
> colnames(animals)
[1] "war" "fly" "ver" "end" "gro" "hai"
> apply(animals,2, table) # simple overview
   war fly ver end gro hai
1  10  16   6  12   6  11
2   10    4  14   6  11    9
```

In total, we have 20 observations with 6 characteristics: `war`, `fly`, `ver`, `end`, `gro`, and `hai`, illustrated by their column names. The following table shows their definitions:

#	Name	Description
1	war	warm-blooded
2	fly	can fly
3	ver	vertebrate
4	end	endangered
5	gro	live in groups

6	hai	have hair
---	-----	-----------

Table 8.2 Definitions of column names for animals

Again, we have 20 observations with 6 characters to describe each of them:

```
| ma <- mona(animals)
| ma
```

The `mona()` function is **MONothetic Analysis Clustering of Binary Variables**, and returns a list representing a divisive hierarchical clustering of a dataset with only binary variables. The related output is shown here:

```

> ma
Revised data:
   war fly ver end gro hai
ant  0   0   0   0   1   0
bee  0   1   0   0   1   1
cat  1   0   1   0   0   1
cpl  0   0   0   0   0   1
chi  1   0   1   1   1   1
cow  1   0   1   0   1   1
duc  1   1   1   0   1   0
eag  1   1   1   1   0   0
ele  1   0   1   1   1   0
fly  0   1   0   0   0   0
fro  0   0   1   1   0   0
her  0   0   1   0   1   0
lio  1   0   1   1   1   1
liz  0   0   1   0   0   0
lob  0   0   0   0   0   0
man  1   0   1   1   1   1
rab  1   0   1   0   1   1
sal  0   0   1   0   0   0
spi  0   0   0   0   0   1
wha  1   0   1   1   1   0

Order of objects:
[1] ant cpl spi lob bee fly fro her liz sal cat cow rab chi lio man
[17] ele wha duc eag
Variable used:
[1] gro NULL hai fly gro ver end gro NULL war gro NULL end
[14] NULL NULL hai NULL fly end
Separation step:
[1] 4 0 5 3 4 2 3 4 0 1 4 0 3 0 0 4 0 2 3

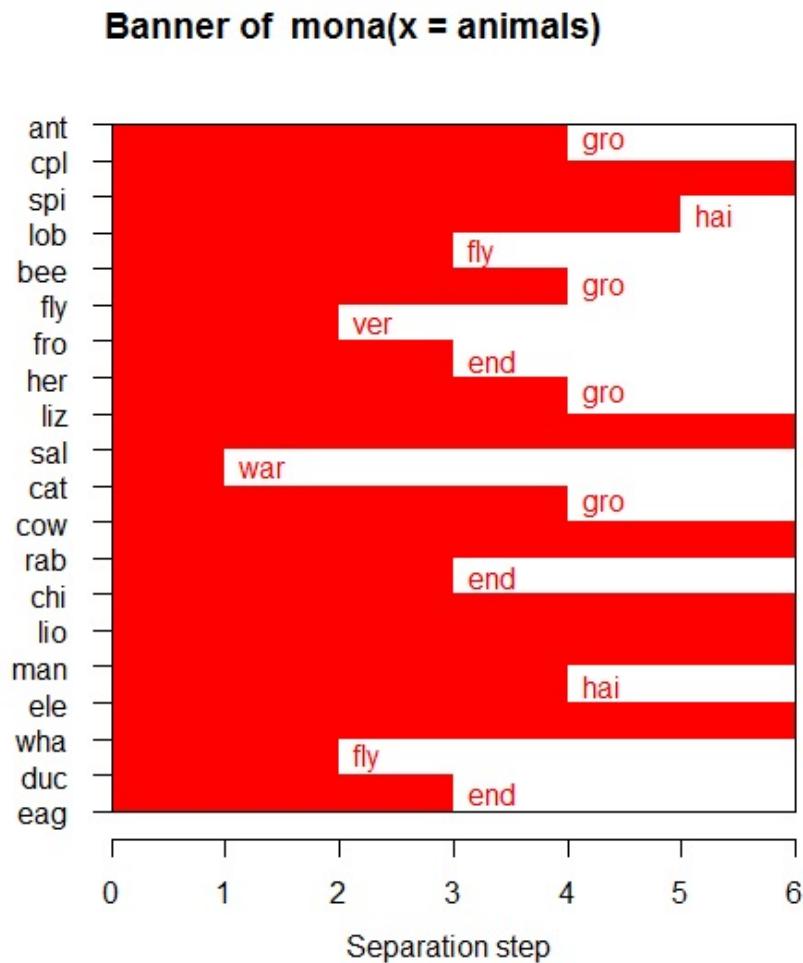
Available components:
[1] "data"      "order"      "variable"    "step"       "call"
[6] "order.lab"

```

From the previous output, we can see that there are five groups. For example, the first observation called `ant` belongs to group 4. Its vector is `[0, 0, 0, 0, 1, 0]`. On the other hand, for the second observation called `bee`, it does not belong to any of those five groups (that is, its group is zero). Its six-value binary vector is `[0, 1, 0, 0, 1, 1]`. To visually present the outcome, we have the following code:

```
| plot(ma)
```

The output is shown in the following screenshot:

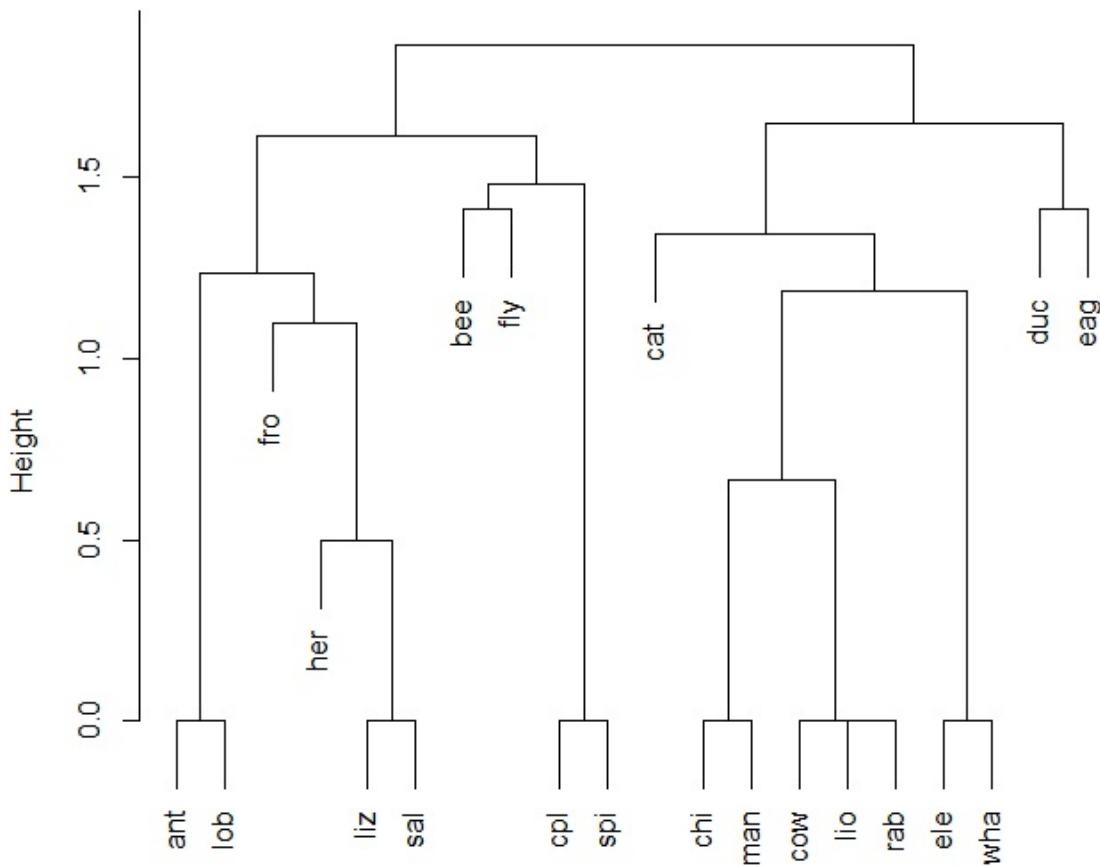


From the previous graph, it can be seen that the first observation belongs to group 4, while the second one does not belong to any of those five groups. To draw a dendrogram for the dataset called `animals`, we have the following code:

```
> require(cluster)
> data(animals)
> x<-agnes(animals)
> pltree(x)
```

Refer to the following graph:

Dendrogram of agnes(x = animals)



animals
agnes (*, "average")

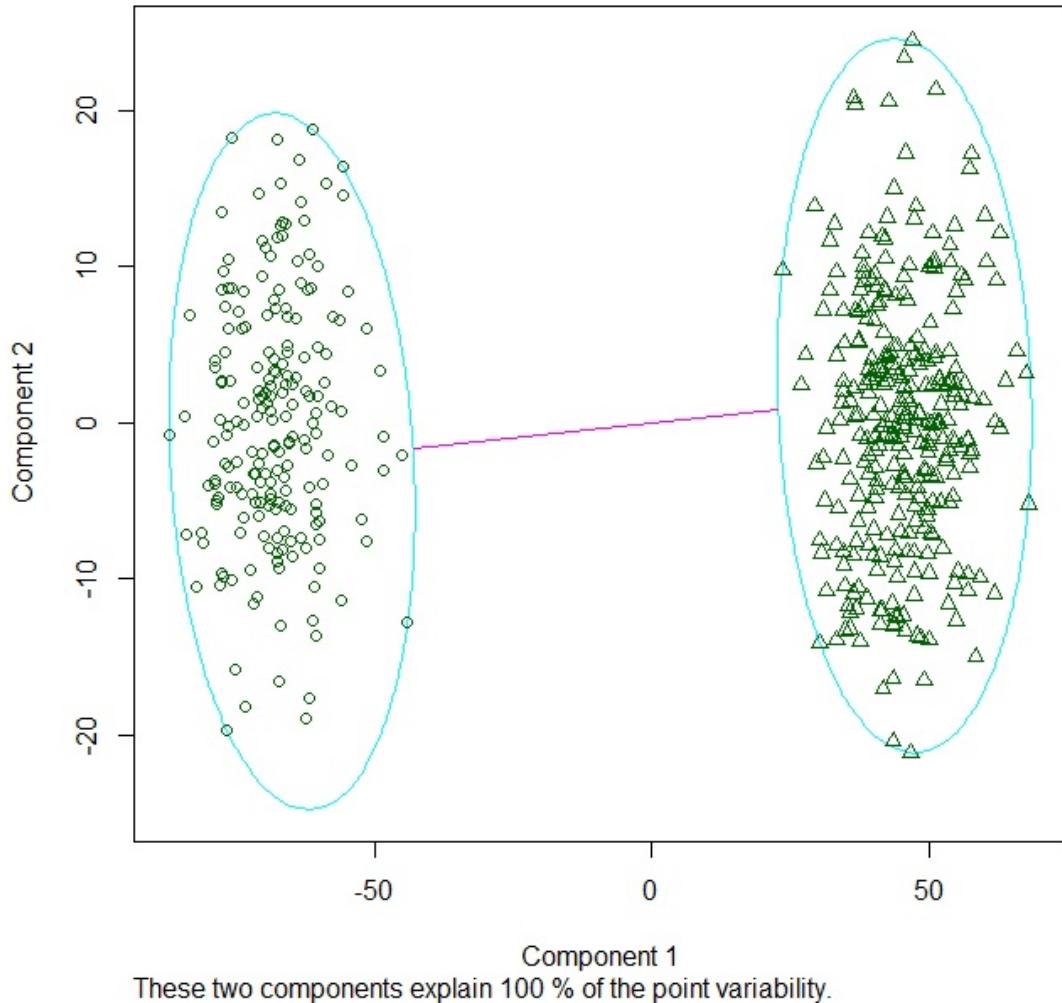
For the next example, we generate two sets of random numbers drawn from a normal distribution with different means, θ_1 and θ_2 , with the same standard deviation of 8:

```
library(cluster)
set.seed(123)
n1<-200; mean1<-0; std1<-8
n2<-300; mean2<-80; std2<-8
set1<-cbind(rnorm(n1,mean1,std1), rnorm(n1,mean1,std1))
set2<-cbind(rnorm(n2,mean2,std2), rnorm(n2,mean2,std2))
x <- rbind(set1, set2)
data <- clara(x, 2, samples=50)
plot(data)
```

The output graph is shown in the following diagram. From it, we can see that

those random values belong to two distinct groups:

```
clusplot(clara(x = x, k = 2, samples = 50))
```



Also note that the `clara()` function from the program has the following structure:

```
| clara(data, nCluster, metric = "euclidean", stand = FALSE, samples = 5, sampsize = min(r
```

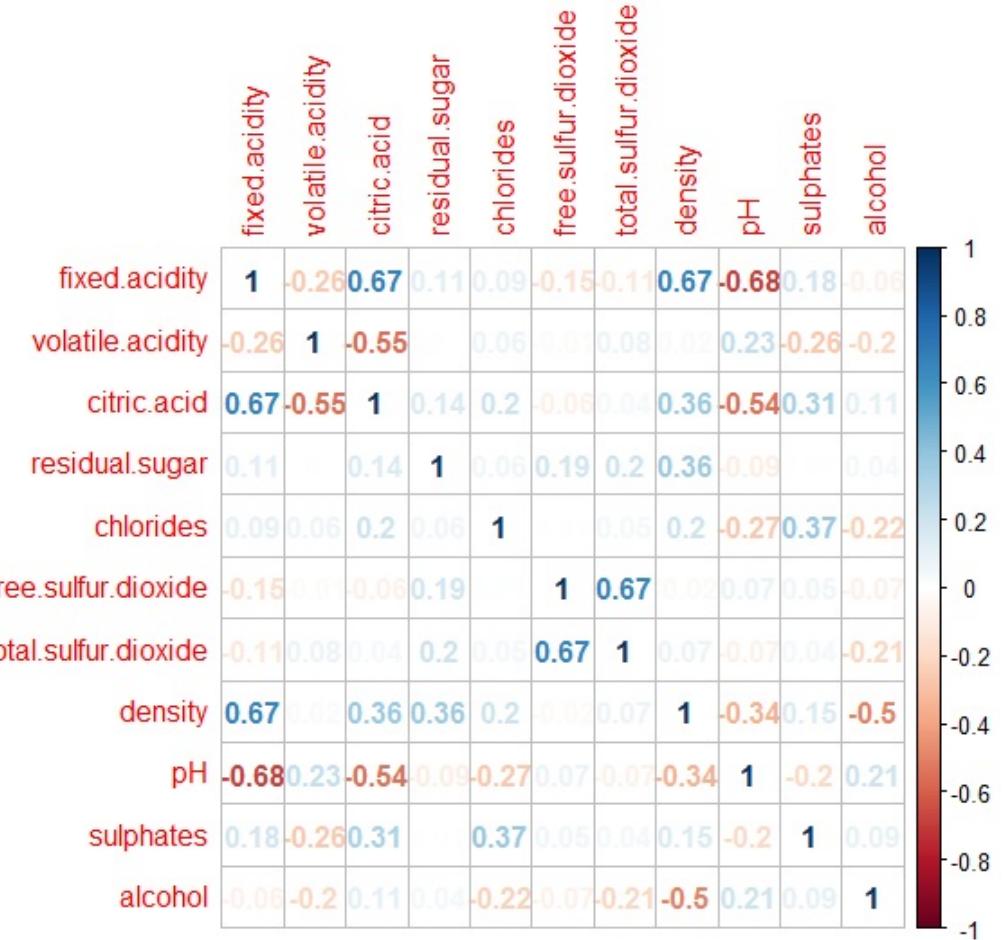
k-means clustering

The purpose of k-means clustering is to partition n observations into k clusters, where each observation belongs to the cluster with the nearest mean. This results in a partitioning of the data space into a **Voronoi diagram**. In mathematics, a Voronoi diagram is a partitioning of a plane into regions based on the distance to points in a specific subset of the plane.

Borrowed from Priyadarsini (<https://www.kaggle.com/maitree/kmeans-unsupervised-learning-using-wine-dataset/data>), the slightly modified code is given here:

```
library(readr)
library(corrplot)
library(ggplot2)
#
path<- "http://canisius.edu/~yany/RData/wine.RData"
load(url(path))
red2<-red
red2$quality<-NULL
white2<-white
white2$quality<-NULL
red_cor<-cor(red2)
white_cor<-cor(white2)
class(red_cor)
class(white_cor)
#
corrplot(red_cor,method="number")
```

The output is shown here:



This square matrix shows the similarity between each combination of treatments. Blue values are all positive, representing similarity, while red values are negative, representing dissimilarity.

```
import scipy.cluster as cluster
```

```
x=dir(cluster)
```

```
print(x)
```

```

import numpy as np

import matplotlib.pyplot as plt

import scipy.cluster.hierarchy as hac

a = np.array([[0.11, 2.5],[1.45, 0.40], [0.3, 1.1],[0.9 , 0.8], [0.5, 0.01],[0.1 , 0.5],
[0.6, 0.5],[2.6, 2.1], [2.3, 3.2],[3.1, 2.2], [3.2, 1.3]])

name='possiblen<- knee point'

yName='{}ncluster distance'

title='Screeplot{}'

t1=['#2200CC','#D9007E','#FF6600','#FFCC00']

t2=['#ACE600','#0099CC','#8900CC','#FF0000']

t3=['#FF9900' , '#FFFF00','#00CC01' , '#0055CC']

fig, axes23 = plt.subplots(2, 3)

for method, axes in zip(['single','complete'], axes23): z = hac.linkage(a,
method=method) axes[0].plot(range(1, len(z)+1), z[::-1, 2]) knee = np.diff(z[::-1,
2], 2) axes[0].plot(range(2, len(z)), knee) for method, axes in
zip(['single','complete'], axes23): z = hac.linkage(a, method=method)
axes[0].plot(range(1, len(z)+1), z[::-1, 2]) knee = np.diff(z[::-1, 2], 2)
axes[0].plot(range(2, len(z)), knee) num_clust1 = knee.argmax() + 2

knee[knee.argmax()] = 0

num_clust2 = knee.argmax() + 2

axes[0].text(num_clust1, z[::-1, 2][num_clust1-1], name) part1 =
hac.fcluster(z, num_clust1, 'maxclust') part2 = hac.fcluster(z, num_clust2,
'maxclust') clr = t1+t2+t3

for part,ax in zip([part1, part2], axes[1:]): for cluster in set(part):
ax.scatter(a[part==cluster,0],a[part==cluster,1], color=clr[cluster]) m =

```

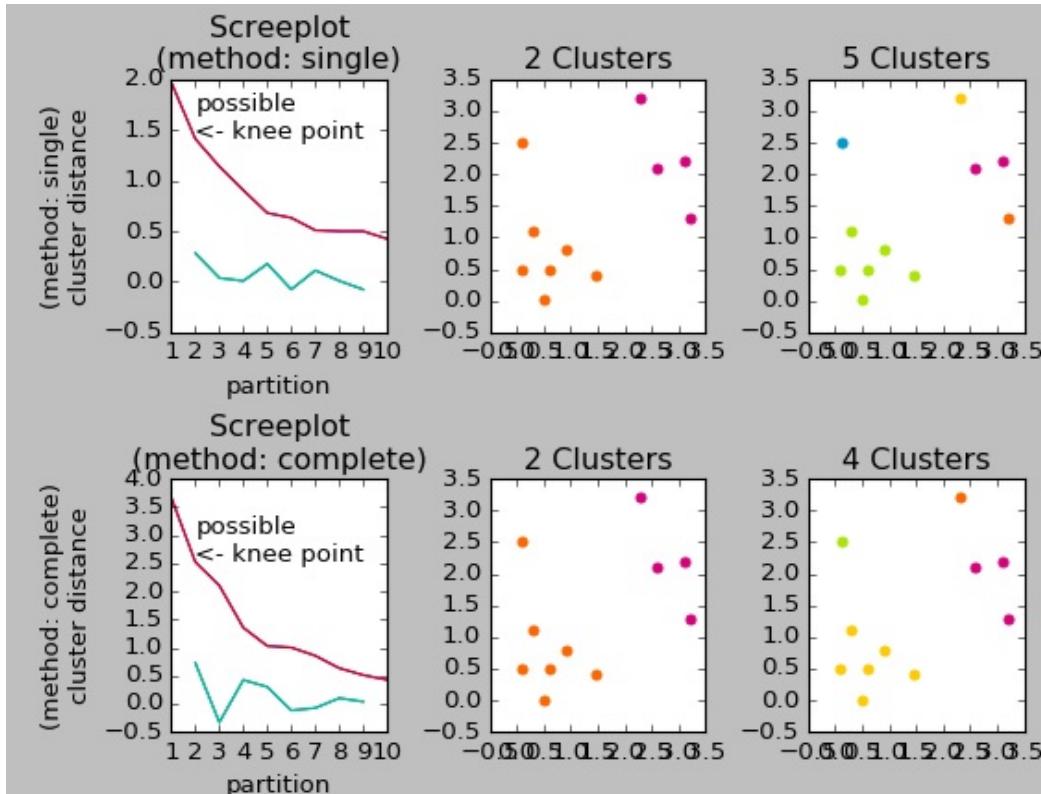
```

'n(method: {})'.format(method) plt.setp(axes[0]
],title=title.format(m), xlabel='partition', ylabel=yName.format(m))
plt.setp(axes[1], title='{} Clusters'.format(num_clust1)) plt.setp(axes[2], title='{} Clusters'.format(num_clust2)) plt.tight_layout()

plt.show()

```

The related graphs are shown here:



The methods used are single and complete in terms of estimating the distance between two points. The top middle graph shows the clustering with two groups, while the top right one shows the clustering for five groups.

Introduction to Python packages – contrastive

To install the Python package called `contrastive`, we issue the following command after launching the Anaconda prompt: **pip install contrastive**

The following screenshot shows the top part of the installation:

```
<base> C:\Users\yany>pip install contrastive
Collecting contrastive
  Downloading contrastive-0.1.6.tar.gz
Requirement already satisfied: numpy in c:\users\yany\.julia\v0.6\conda\deps\usr
\lib\site-packages (from contrastive)
Collecting sklearn (from contrastive)
  Downloading sklearn-0.0.tar.gz
```

Using the `dir()` and `print()` functions, we could find more information about a function embedded in the Python package: from `contrastive import cluster`
`x=dir(cluster)` `print(x)`

The output is shown here:

```
In [3]: from contrastive import cluster
x=dir(cluster)
print(x)

['AffinityPropagation', 'AgglomerativeClustering', 'Birch',
 'DBSCAN', 'FeatureAgglomeration', 'KMeans', 'MeanShift', 'M
iniBatchKMeans', 'SpectralBiclustering', 'SpectralClusterin
g', 'SpectralCoclustering', '__all__', '__builtins__', '__d
oc__', '__file__', '__name__', '__package__', '__path__',
 '_dbSCAN_inner', '_feature_agglomeration', '_hierarchical',
 '_k_means', '_k_means_elkan', 'affinity_propagation', 'affi
nity_propagation_', 'bicluster', 'birch', 'dbSCAN', 'dbSCAN '
, 'estimate_bandwidth', 'get_bin_seeds', 'hierarchical',
 'k_means', 'k_means_', 'linkage_tree', 'mean_shift', 'mean_
shift_', 'spectral', 'spectral_clustering', 'ward_tree']
```

Introduction to Python packages – sklearn (scikit-learn)

First, let's look at the functions contained in the Python package called `sklearn`. The code has just three lines:

```
import sklearn as sk  
x=dir(sk)  
print(x)
```

The related output is shown here:

```
In [6]: import sklearn as sk  
x=dir(sk)  
print(x)  
  
['__ASSUMEFINITE', '__SKLEARN_SETUP__', '__all__', '__builtins__', '__check_build', '__doc__', '__file__', '__name__', '__package__', '__path__', '__version__', '__contextmanager', '__isotonic', 'base', 'clone', 'cluster', 'config_context', 'decomposition', 'exceptions', 'externals', 'feature_selection', 'get_config', 'isotonic', 'linear_model', 'logger', 'logging', 'manifold', 'metrics', 'model_selection', 'neighbors', 'os', 'preprocessing', 'random_projection', 're', 'set_config', 'setup_module', 'svm', 'sys', 'tree', 'utils', 'warnings']
```

For one specific submodule, it is called `sklearn.cluster`, as shown:

```
from sklearn import cluster  
x=dir(cluster)  
print(x)
```

The output is shown here:

```
In [5]: from sklearn import cluster
x=dir(cluster)
print(x)

['AffinityPropagation', 'AgglomerativeClustering', 'Birch',
 'DBSCAN', 'FeatureAgglomeration', 'KMeans', 'MeanShift', 'M
iniBatchKMeans', 'SpectralBiclustering', 'SpectralClusterin
g', 'SpectralCoclustering', '__all__', '__builtins__', '__d
oc__', '__file__', '__name__', '__package__', '__path__', '__
_dbSCAN_inner', '__feature_agglomeration', '__hierarchical',
 '__k_means', '__k_means_elkan', 'affinity_propagation', 'affi
nity_propagation_', 'bicluster', 'birch', 'dbSCAN', 'dbSCAN_
', 'estimate_bandwidth', 'get_bin_seeds', 'hierarchical',
 'k_means', 'k_means_', 'linkage_tree', 'mean_shift', 'mean_
shift_', 'spectral', 'spectral_clustering', 'ward_tree']
```

In addition, we can show many embedded datasets by using the following three lines of Python code:

```
import sklearn.datasets as datasets
x=dir(datasets)
print(x)
```

The output is shown here:

```
print(x)

['__all__', '__builtins__', '__doc__', '__file__', '__name__', '__package
__', '__path__', '__svmlight_format', 'base', 'california_housing', 'clear
_data_home', 'covtype', 'dump_svmlight_file', 'fetch_20newsgroups', 'fetc
h_20newsgroups_vectorized', 'fetch_california_housing', 'fetch_covtype',
'fetch_kddcup99', 'fetch_lfw_pairs', 'fetch_lfw_people', 'fetch_mldata',
'fetch_olivetti_faces', 'fetch_rcv1', 'fetch_species_distributions', 'get
_data_home', 'kddcup99', 'lfw', 'load_boston', 'load_breast_cancer', 'loa
d_diabetes', 'load_digits', 'load_files', 'load_iris', 'load_linnerud', 'l
oad_mlcomp', 'load_sample_image', 'load_sample_images', 'load_svmlight_f
ile', 'load_svmlight_files', 'load_wine', 'make_biclusters', 'make_blobs'
, 'make_checkerboard', 'make_circles', 'make_classification', 'make_fried
man1', 'make_friedman2', 'make_friedman3', 'make_gaussian_quantiles', 'ma
ke_hastie_10_2', 'make_low_rank_matrix', 'make_moons', 'make_multilabel_c
lassification', 'make_regression', 'make_s_curve', 'make_sparse_coded_sig
nal', 'make_sparse_spd_matrix', 'make_sparse_uncorrelated', 'make_spd_mat
rix', 'make_swiss_roll', 'mlcomp', 'mldata', 'mldata_filename', 'olivetti
_faces', 'rcv1', 'samples_generator', 'species_distributions', 'svmlight_
format', 'twenty_newsgroups']
```

For example, if we want to use a dataset called `iris`, we can apply the `load_iris()` function, as shown:

```
|from sklearn import datasets
```

```
| data= datasets.load_iris()  
| print(data)
```

The first few lines are shown here:

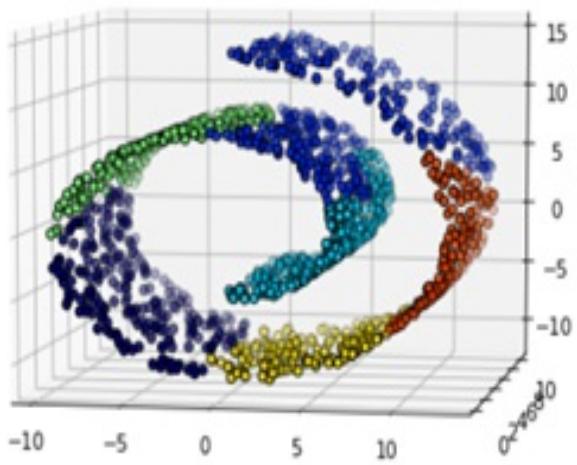
```
import numpy as np  
from sklearn import datasets  
x= datasets.load_iris()  
print(x)  
  
{'target_names': array(['setosa', 'versicolor', 'virginica']),  
 'data': array([[5.1, 3.5, 1.4,  
 , 0.2],  
 [4.9, 3. , 1.4, 0.2],  
 [4.7, 3.2, 1.3, 0.2],  
 [4.6, 3.1, 1.5, 0.2],  
 [5. , 3.6, 1.4, 0.2],  
 [5.4, 3.9, 1.7, 0.4],  
 [4.6, 3.4, 1.4, 0.3],
```

The following code is one example of using the module:

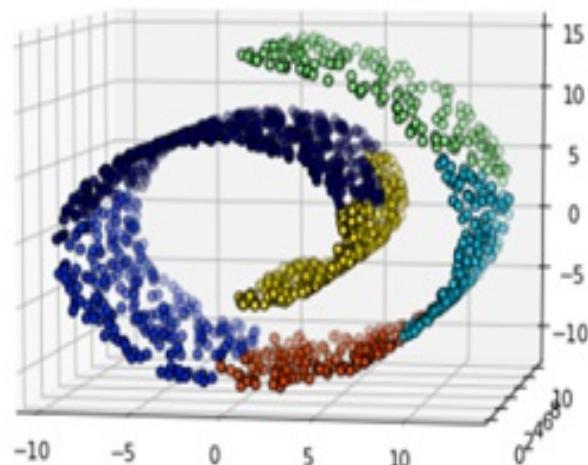
```
from sklearn import cluster  
from sklearn import datasets  
x=datasets.load_iris()  
iris = datasets.load_iris()  
x = iris.data  
k_means = cluster.KMeans(n_clusters=3)  
k_means.fit(x)  
print(k_means.labels_[:10])  
[1 1 1 1 1 0 0 0 0 2 2 2 2]
```

The last line of the previous code shows the potential groups they belong to. The next example is by *Michel, Gramfort and Varoquaux* (2010), http://scikit-learn.org/stable/auto_examples/cluster/plot_ward_structured_vs_unstructured.html#sphx-glr-auto-examples-cluster-plot-ward-structured-vs-unstructured-py. The output is shown in the following screenshot:

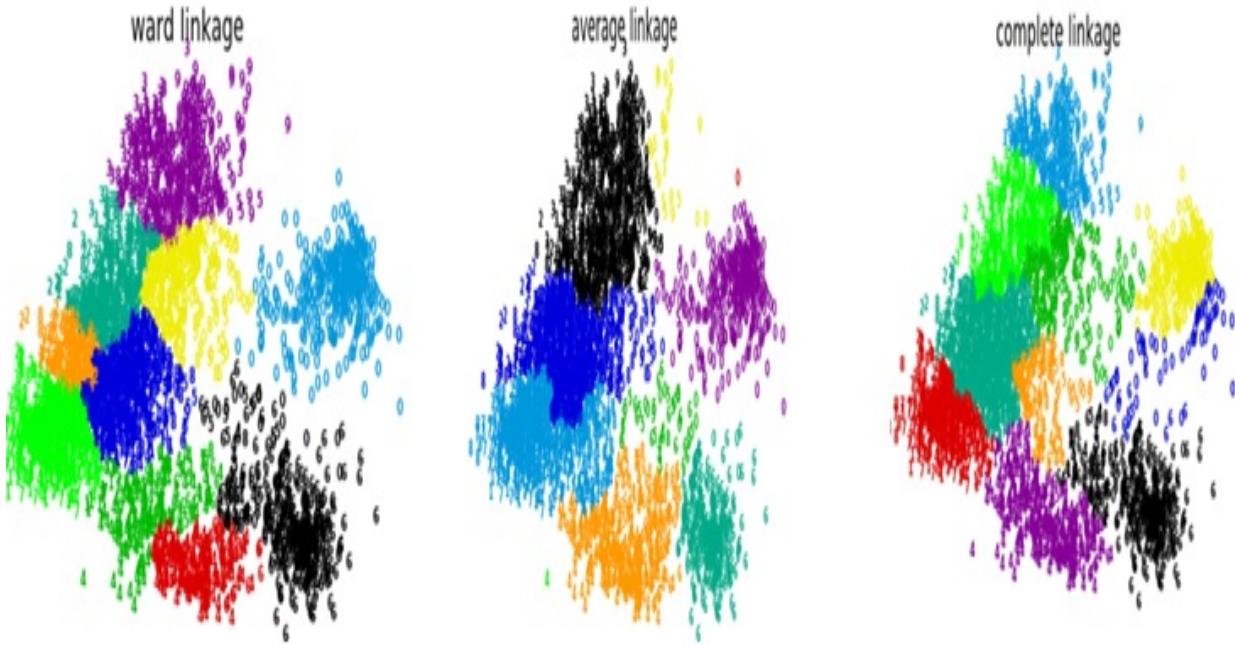
Without connectivity constraints (time 0.19s)



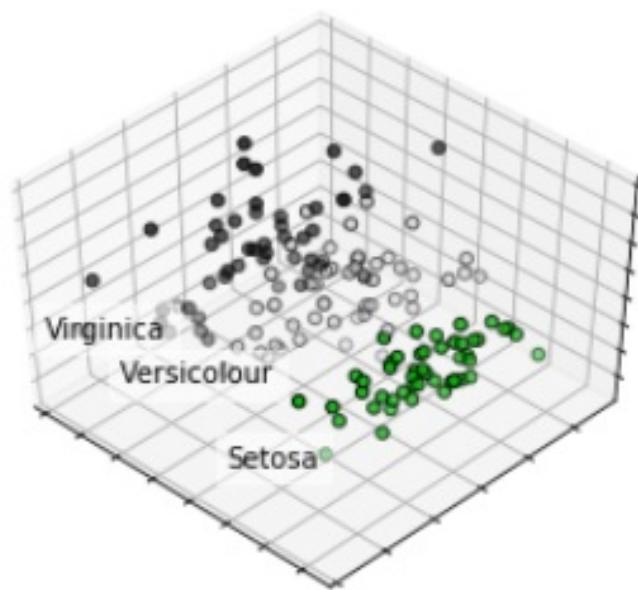
With connectivity constraints (time 0.11s)



From the previous graphs, we visually view groups by their colors. For example, the left graph shows us there are six groups. The next example is about Agglomerative Clustering on a 2D embedding of digits. Since the program is too long, we would not include it here. You can find the program at http://scikit-learn.org/stable/auto_examples/cluster/plot_digits_linkage.html#sphx-glr-auto-examples-cluster-plot-digits-linkage-py. The related graph output is shown in the following screenshot. The graph shows several groups based on a few different definitions of distance (linkage):



Based on the definitions of distance of **ward**, **average**, or **complete**, we have three ways to generate clusters. For each method, various colors represent different clusters. The last example is looking at the `iris` dataset, which is associated with **Principal Component Analysis (PCA)**. PCA uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called **principal components** (http://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_iris.html). The related 3D picture is shown here. The graph shows the groups for these three types:



One useful link is <http://scikit-learn.org/stable/modules/clustering.html>.

Introduction to R packages – rattle

Before discussing one or two examples of using `rattle`, it might be a good idea to discuss an R package called `rattle.data`. As its name suggests, we could guess that it contains data used by `rattle`. It is a good idea to use a small dataset to generate a dendrogram. For the next case, we use the first 20 observations from a dataset called `wine`:

```
| library(rattle.data)
| data(wine)
| x<-head(wine,20)
```

To launch `rattle`, we have the following two lines:

```
| library(rattle)
| rattle()
```

For data, we choose R Dataset, then choose `x`, as shown in the following screenshot. To save space, only the top part is presented here:

The screenshot shows the Rattle software interface. At the top, there is a menu bar with 'File', 'Edit', 'View', 'Help', and 'About'. Below the menu, there is a toolbar with icons for 'New', 'Open', 'Save', 'Print', 'Exit', 'Copy', 'Paste', 'Find', 'Replace', 'Select All', 'Clear', 'Zoom In', 'Zoom Out', and 'Zoom Fit'. The main area has several sections:

- Source:** A radio button group with 'File' selected, and other options like 'ARFF', 'ODBC', 'R Dataset' (which is selected), 'RData File', 'Library', 'Corpus', and 'Script'.
- Data Name:** A text input field containing 'x'.
- Partition:** A checked checkbox with a date field '70/15/15' and a seed field '42' with up/down arrows, along with 'View' and 'Edit' buttons.
- Input:** A green radio button is selected.
- Ignore:** A red radio button is selected.
- Weight Calculator:** An empty text input field.
- Target Data Type:** A radio button group with 'Auto' selected, and other options like 'Categoric', 'Numeric', and 'Survival'.
- Table:** A grid showing variable details:

No.	Variable	Data Type	Input	Target	Risk	Ident	Ignore	Weight	Comment
1	Type	Categoric	<input type="radio"/>	Unique: 1					
2	Alcohol	Numeric	<input type="radio"/>	Unique: 20					

The following screenshot shows our choice:

Type: KMeans Ewkm Hierarchical BiCluster

Build Options: Distance: euclidean Agglomerate: ward Processors: 1

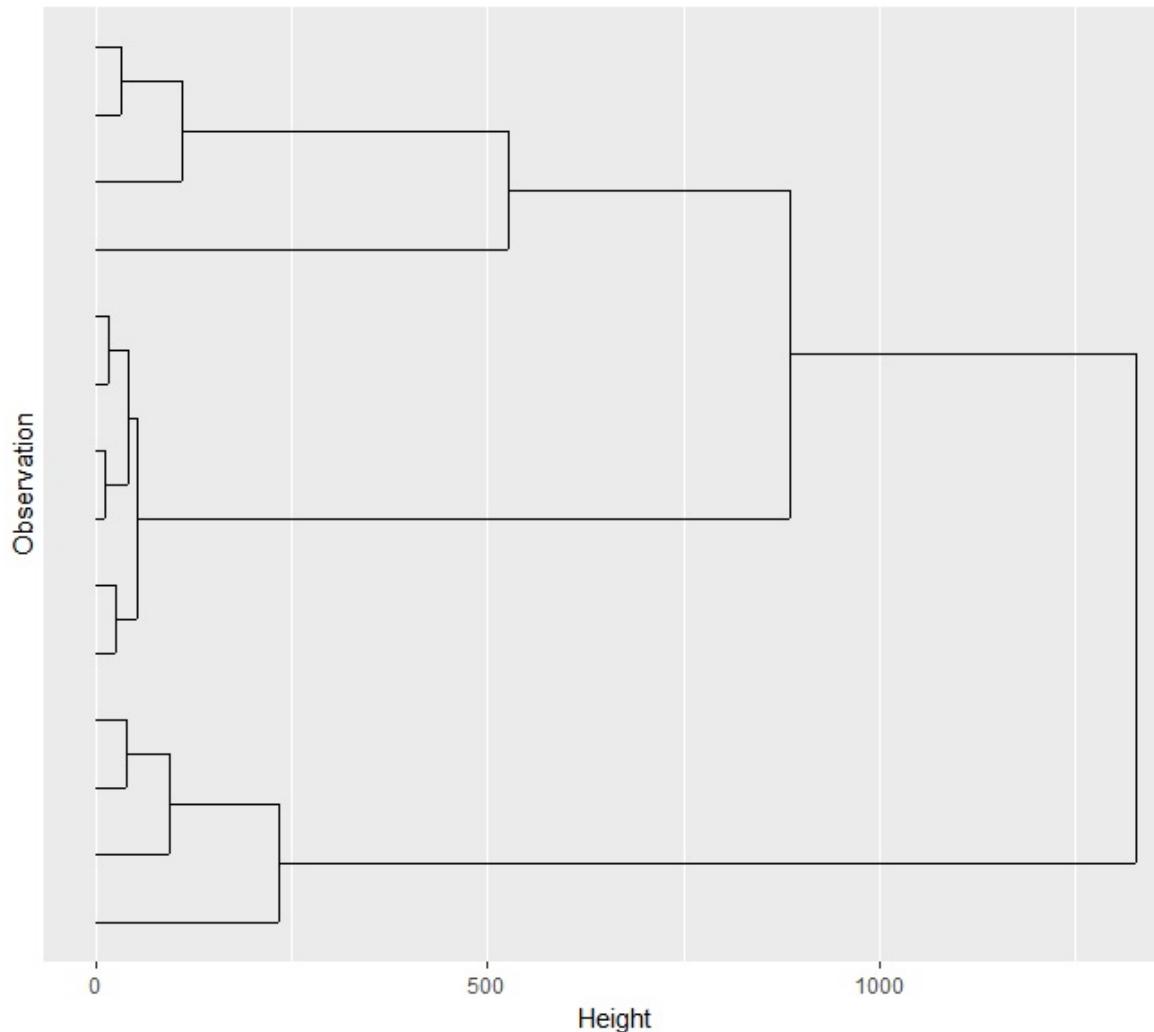
Cluster Options: Clusters: 10

Hierachical Cluster

```
Call:  
amap::hclusterpar(x = ., method = "euclidean", link = "ward",      nbproc = 1)  
  
Cluster method    : ward  
Distance          : euclidean  
Number of objects: 14  
  
Time taken: 0.00 secs  
  
Rattle timestamp: 2018-03-22 14:57:17 yany
```

From the previous screenshot, we see 14 observations. Click Clusters, with a default of 10 clusters, and Dendrogram. See the result in the following screenshot:

Cluster Dendrogram x
Rattle 2018-Mar-25 10:58:07 yany



The previous dendrogram is a tree diagram that is used to show the relationship for a clustering analysis. Based on the definition of distance or relationship, we have a link between the closed associated one pair. Then, we move one step up to link this pair with another adjacent point or observation. The procedure continues until we reach the final step of just one group.

Introduction to R packages – randomUniformForest

The R package, `randomUniformForest` is for classification, regression, and unsupervised learning. The basic block for a random forest is the decision tree model. To make our classification, researchers add some random effect. Because of this, the random forest method performs better than the decision tree model. We can use the following code to install, load, and get help about this specific R package:

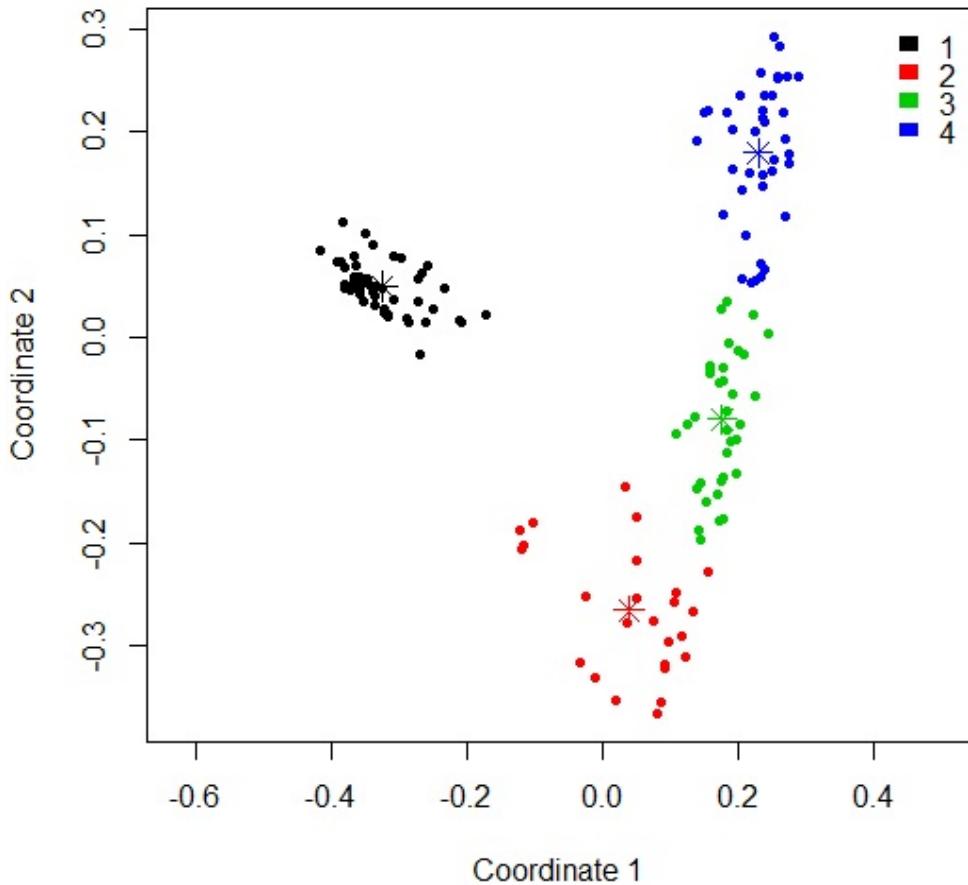
```
|> install.packages("randomUniformForest")
|> library(randomUniformForest)
|> help(package=randomUniformForest)
```

First, let's look at a relatively simple program that tries to classify different plants into four groups by using the random forest method:

```
library(randomUniformForest)
data(iris)
shortName<-unsupervised.randomUniformForest
x =shortName(iris[,-5],mtry=1,nodesize = 2)
plot(x)
```

It is obvious that the function name of `unsupervised.randomUniformForest` is quite long. To make our program more readable, we generate a short name called `shortName` instead. The related output is shown in the following screenshot, where it illustrates four groups by color:

Multidimensional scaling and Clusters representation

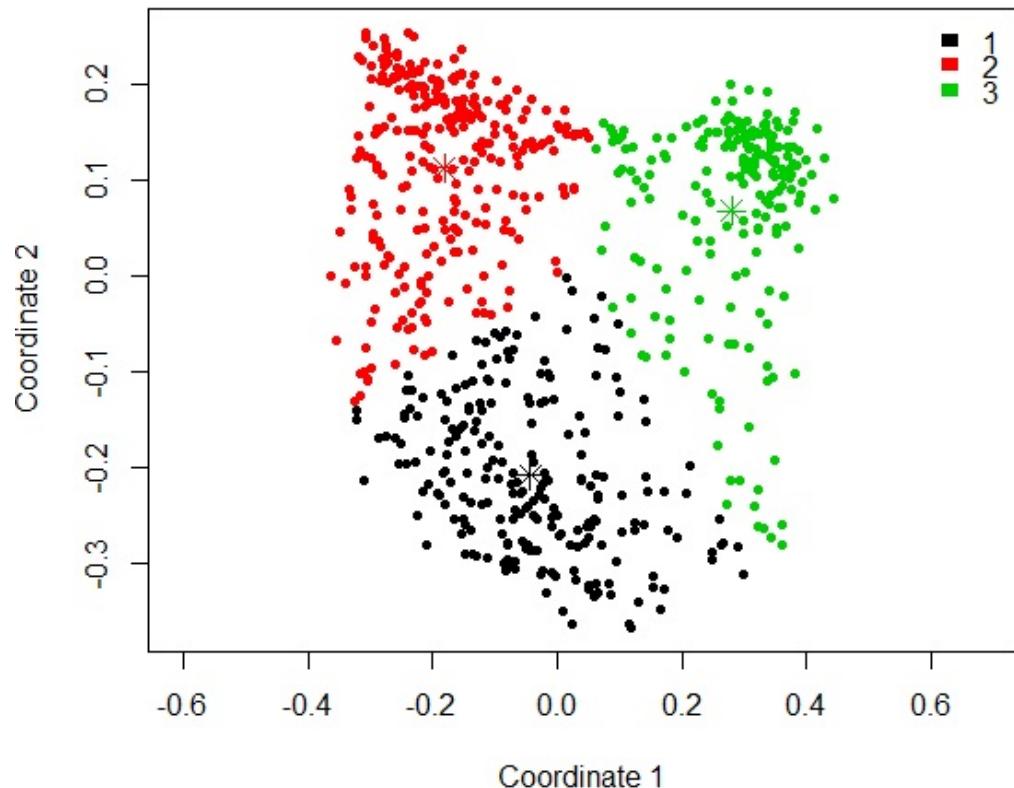


The second example presents several groups based on the wine quality, such as alcohol concentration, color, and dilution:

```
library(randomUniformForest)
data(wineQualityRed)
x = wineQualityRed[, -ncol(wineQualityRed)]
# run unsupervised analysis on the first half of dataset
data1 = 1:floor(nrow(x)/2)
sf<-unsupervised.randomUniformForest
model1 =sf(x,subset =data1,depth = 5)
plot(model1)
```

The graph is shown here:

Multidimensional scaling and Clusters representation



```
> library(Rmixmod)

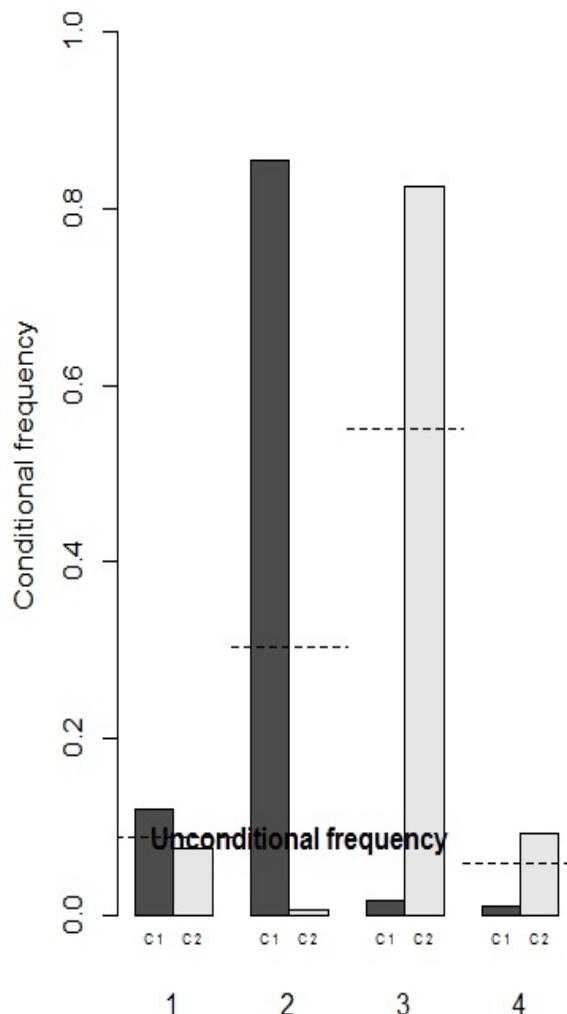
> data(birds)<br/>> head(birds) <strong> gender eyebrow collar sub-caudal
border 1 male poor pronounced dotted white few 2 female none dotted black
none 3 female pronounced none white none 4 male pronounced dotted white
none
```

5 male pronounced dotted white none

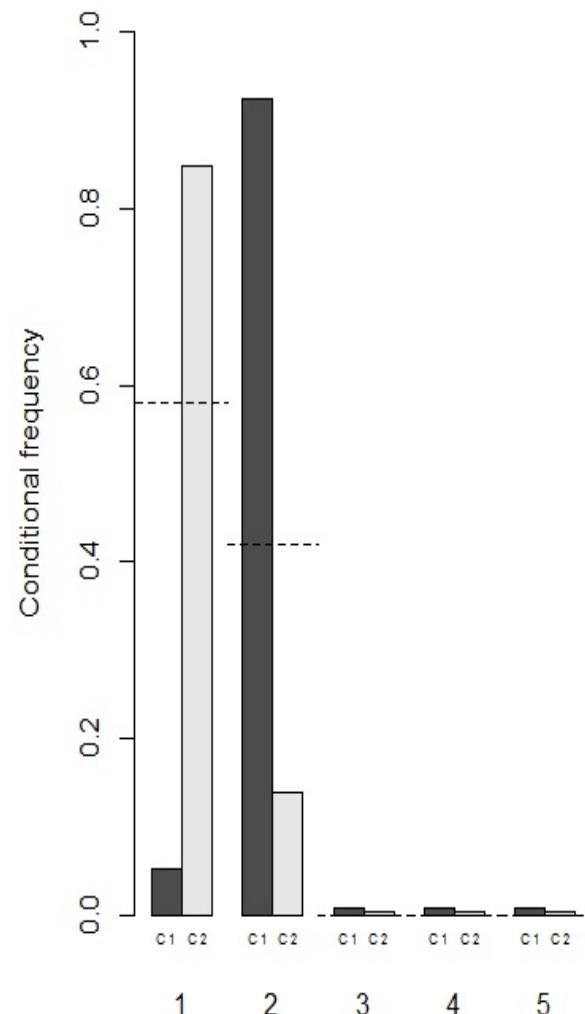
```
6 male pronounced dotted white none</strong> > dim(birds)
<strong>[1] 69 5</strong>
x <- mixmodCluster(birds,2) bb<-barplotCluster
bb(x["bestResult"], birds)
bb(x["bestResult"], birds, variables=c(2,3,4)) bb(x["bestResult"], birds,
variables=c("eyebrow","collar"))
```

The graphs are shown here:

Barplot of eyebrow



Barplot of collar



```
using Base.Test
```

```
using Distances
```

```
using Clustering
```

```
#
```

```
srand(12345)
```

```
d = 10
```

```
n = 500
```

```
x = rand(d, n)
```

```
S = -pairwise(Euclidean(), x, x)
```

```
# set diagonal value to median value
```

```
S = S - diagm(diag(S)) + median(S)*eye(size(S,1))
```

```
R = affinityprop(S)
```

The output is shown here:

```
julia> R = affinityprop(S)
Clustering.AffinityPropResult([11, 26, 55, 56, 102, 111, 114, 123, 125, 126,
401, 402, 406, 429, 434, 448, 451, 452, 486, 495], [50, 15, 29, 44, 38, 12, 51,
53, 1, 52, 39, 34, 36, 41, 56, 20, 48, 30, 28, 35], [10, 19, 10, 8, 8, 10, 6,
16, 8, 7, 7, 15, 6, 5, 9, 11, 6, 8, 10, 7], 68, true)
```

Based on the output, we see a few groups and which numbers belong to which group.

Task view for Cluster Analysis

In the previous chapters, we have discussed various kinds of task views. A task view is a set of R packages organized by one or a few experts around a topic. For the **Cluster Analysis**, <https://CRAN.R-project.org/view=Cluster>, the top part is shown in the following screenshot:

CRAN Task View: Cluster Analysis & Finite Mixture Models

Maintainer: Friedrich Leisch and Bettina Gruen
Contact: Bettina.Gruen at jku.at
Version: 2018-03-09
URL: <https://CRAN.R-project.org/view=Cluster>

This CRAN Task View contains a list of packages that can be used for finding groups in data and modeling unobserved cross-sectional heterogeneity. Many packages provide functionality for more than one of the topics listed below, the section headings are mainly meant as quick starting points rather than an ultimate categorization. Except for packages stats and cluster (which ship with base R and hence are part of every R installation), each package is listed only once.

Most of the packages listed in this CRAN Task View, but not all are distributed under the GPL. Please have a look at the DESCRIPTION file of each package to check under which license it is distributed.

Hierarchical Clustering:

- Functions `hclust()` from package stats and `agnes()` from `cluster` are the primary functions for agglomerative hierarchical clustering, function `diana()` can be used for divisive hierarchical clustering. Faster alternatives to `hclust()` are provided by the packages `fastcluster` and `flashClust`.

As discussed in [chapter 6, Managing Packages](#), we can use just three lines of R code to install these cluster-related packages:

```
>install.packages("ctv")
>library("ctv")
>install.views("Cluster")
```

In total, there are about 154 R packages included in the previous task view called **Cluster**, as of March 21, 2018.

Summary

In this chapter, we have discussed unsupervised learning. In particular, we have explained hierarchical clustering and k-means clustering. As for R and Python, we have explained several related packages:

- R: `rattle`, `Rmixmod`, and `randomUniformForest`
- Python: `scipy.cluster`, `contrastive`, and `sklearn`

Several real-world examples have also been used to illustrate the applications of these packages in detail.

For the next chapter, we will discuss supervised learning, such as classification, the k-nearest neighbors algorithm, Bayes' classifiers, reinforcement learning, and specific R and Python-related modules, such as `RTextTools` and `sklearn`. In addition, we will discuss implementation via R, Python, Julia, and Octave.

Review questions and exercises

1. What does *unsupervised learning* mean?
2. What is the major difference between unsupervised learning and supervised learning?
3. How do we install the Python package `sklearn`?
4. Discuss the relationship between distance and clustering classification.
5. How do we define the distance between two objects?
6. For non-numeric values, how do we define a distance between two members?
7. For R, we could find a set of related packages related to unsupervised learning called `cluster`. Is there any task view, or similar super package, for Python?
8. First, generate the following set of random numbers:

```
>set.seed(12345)

>n=30

>nGroup=4

>x <- matrix(rnorm(n*nGroup), nrow =nGroup)
```

Then, based on the various definitions of distance, estimate the distances between those four groups.

9. For the following set of data, estimate the minimum, maximum, and average distances:

```
|       > data <- rbind(c(180,20), c(160,5), c(60, 150), c(160,60), c(80,120))
```

10. What is the usage of a dendrogram?
11. Draw a dendrogram by using all wine data.
12. Generate 20 random numbers with a mean of 1.2 and standard deviation of

2 from a normal distribution. Then draw a dendrogram.

13. Using a five-year monthly historical price data for 30 stocks, estimate their annualized standard deviations and means. Classify them into different groups. The source of data is Yahoo!Finance (<http://finance.yahoo.com>). Note that the following formulae are used to calculate an annualized standard deviation:

$$\begin{cases} \sigma_{annual} = \sqrt{12} \sigma_{monthly} \\ \bar{R}_{annual} = (1 + \bar{R}_{monthly})^{12} - 1 \end{cases}$$

Where σ_{annual} is the annualized standard deviation, $\sigma_{monthly}$ is the standard deviation based on monthly returns, \bar{R}_{annual} is the annualized mean return, and $\bar{R}_{monthly}$ is the monthly mean return.

14. For the R package called `cluster`, what is the meaning of the `votes.repub` dataset? Using that dataset, conduct a Cluster Analysis and draw a dendrogram tree.
15. Find more information about the `linkage_tree()` function contained in the `sklearn.cluster` submodule. (Python)
16. For the `rattle` package, how do we save an R script?

Supervised Learning in Anaconda

Since most of us understand the format of the function $y=f(x)$, it is a good idea to use it to explain supervised learning. When having both y and x , we could run various regressions to identify the correct function forms. This is the spirit of supervised learning. For supervised learning, we have two datasets: the **training data** and **test data**. Usually, the training set has a set of input variables, such as x , and a related output value such as y (that is, the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function form. Then, we apply this inferred function to map our test dataset.

In this chapter, the following topics will be covered:

- A glance at supervised learning
- Classification
- Implementation of supervised learning via R, Python, Julia, and Octave
- Task view for machine learning in R

A glance at supervised learning

In the previous chapter, we discussed unsupervised learning where we have input data only. In terms of the function $y=f(x)$, for unsupervised learning we have only inputs x . Unlike unsupervised learning, we have both inputs x and the corresponding output y for supervised learning. Our task is to find the best function, linking x with y , based on our training dataset. In supervised learning, our training dataset consists of an input object, typically a vector, and a desired output value, where it could be either binary, categorical, discrete, or continuous. A supervised learning algorithm examines a given training dataset and produces an inferred best-fit function. To verify the accuracy of this inferred function, we use the second dataset, the test set.

In an ideal world, we would want to have a large sample size. However, for many occasions, this is not true. In those cases, we could apply the bootstrap method for estimating statistical quantities from samples. The **Bootstrap Aggregation** algorithm is used to create multiple different models from a single training dataset. Actually, random forest is a sort of nickname for *decision tree ensemble through bagging*. Because of this, it ends up with a very powerful classifier.

Let's use the famous Titanic tragedy as an example. In 1912, on its maiden voyage, the Titanic sank after striking an iceberg. Of its passengers and crew, more than 1,500 died. First, let's look at the simple dataset. The following R code can be downloaded from the author's website at <http://canisius.edu/~yany/RData/titanic.RData>:

```
> path<- "http://canisius.edu/~yany/RData/"
> dataSet<- "titanic"
> link<-paste(path,dataSet,".RData",sep='')
> con<-url(link)
> load(con)
> dim(.titanic)
[1] 2201    4
> head(.titanic)
  CLASS AGE GENDER SURVIVED
1 First  Adult   Male    Yes
2 First  Adult   Male    Yes
3 First  Adult   Male    Yes
4 First  Adult   Male    Yes
5 First  Adult   Male    Yes
6 First  Adult   Male    Yes
```

```

> summary(.titanic)
  CLASS      AGE      GENDER    SURVIVED
  Crew :885   Adult:2092   Female: 470   No :1490
  First :325   Child: 109   Male   :1731   Yes: 711
  Second:285
  Third :706

```

From the preceding output, it can be seen that the dataset has 2,201 observations with just 4 variables. The `CLASS` is for cabin or economic status, `GENDER` is for gender, and `SURVIVED` indicates whether the passenger survived or not. The `unique()` function can be used to show their unique values, shown here:

```

> unique(.titanic$CLASS)
[1] First  Second Third  Crew
Levels: Crew First Second Third

> unique(.titanic$GENDER)
[1] Male   Female
Levels: Female Male
> unique(.titanic$SURVIVED)
[1] Yes  No
Levels: No  Yes

```

Our task is to use the decision tree to find the contributions of those three input variables to the survival rate. Let's use the R package called `Rattle` to run a simple decision tree model by using the embedded dataset. To launch the `Rattle` package, we have the following two lines:

```

>library(rattle)
>rattle()

```

The next screenshot shows how to choose a library and a related dataset. Note that, depending on the version, the dataset might not be available. An alternative way is to download from the author's web page at <http://canisius.edu/~yany/RData/titanic.RData>. The default setting for the partition is 70% (training), 15% (verification), and 15% (testing):

Data [Explore](#) [Test](#) [Transform](#) [Cluster](#) [Associate](#) [Model](#) [Evaluate](#) [Log](#)

Source: File ARFF ODBC R Dataset RData File Library Corpus Script

Data Name: titanic:Rmixmod:Qualitative data: Survival of passengers on the Titanic ▾

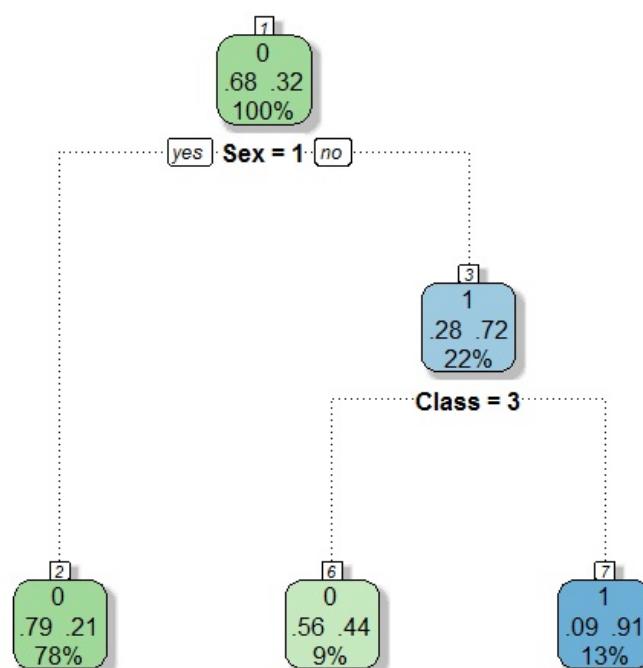
Partition Seed:

Input Ignore Weight Calculator: Target Data Type Auto Categoric Numeric

No.	Variable	Data Type	Input	Target	Risk	Ident	Ignore	Weight	Comment
1	Class	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 4
2	Age	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2
3	Sex	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2
4	Survived	Categoric	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2

If we choose Model | Tree, and then Execute in the upper left corner, we will get the following result:

Decision Tree titanic \$ Survived



The top ratio of **68%** is for the rate of those who did not survive while the **32%** is for the survival rate. The first question is whether the passenger is a male or not. If the answer is a yes, then the person would have a **21%** chance of survival, shown in the lower-left leaf. For a female passenger who booked a cabin of **Class 3**, she would have a **44%** chance of survival. For female passengers, when they occupy a first- or second-class cabin, she would have a **91%** chance of survival. Since the age does not appear in our final result, the model considers it useless.

Note that if you plan to use the loaded dataset called `.titanic`, you could use `x<-.titanic`, and then choose it by selecting R Dataset after Rattle is launched. For different associated datasets with more variables, it is not a surprise that users could reach different conclusions. You can save the log, that is, the code, for a later usage. The simplified code is given here. The log is on the Rattle menu bar:

```
library(rpart, quietly=TRUE)
con<-url("http://canisius.edu/~yany/RData/titanic.RData")
load(con)
x<-.titanic
scoring <- FALSE
set.seed(42)
risk<-ident<-ignore<-weights<-numeric<-NULL
str(dataset)
n<- nrow(dataset)
train <- sample <- sample(n,0.7*n)
validate<- sample(setdiff(seq_len(n),train),0.15*n)
test<- setdiff(setdiff(seq_len(n), train), validate)
inputVars<-categorical<-c("CLASS", "AGE", "GENDER")
target<-"SURVIVED"
output<-rpart(SURVIVED~.,data=x[train, c(inputVars, target)],
  method="class", parms=list(split="information"),control=
    rpart.control(usesurrogate=0,maxsurrogate=0))
fancyRpartPlot(output, main="Decision Tree for Titanic")
```

The second example is associated with the dataset called `iris`. This time, the language used is Python. First, let's look at the data itself. The code and related output is shown here:

```
import sklearn as sk
from sklearn import datasets
iris = datasets.load_iris()
print("data:n",iris.data[0:4,:])
print("target",iris.target[0:2,:])
mylist=list(iris.target)
used = []
[x for x in mylist if x not in used and used.append(x)]
print("unique values for targetsn",used)
```

The output would look like follows:

```

data:
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.  1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]]
target [0 0]
unique values for targets
[0, 1, 2]

```

The dataset has five variables: sepal length, sepal width, petal length, petal width (all in cm), and class. The last one has three unique values for Setosa, Versicolour, or Virginica; see <https://archive.ics.uci.edu/ml/datasets/Iris> for more information. The first several lines of the original dataset are shown here:

```

5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa

```

The dataset can be downloaded from the UCI Machine Learning Repository: <http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data>. First, let's cut the dataset into a training and testing set. Then we use supervised learning to figure out the mapping function, that is, an inferred function:

```

import numpy as np
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier as KNC
iris = datasets.load_iris()
x= iris.data
y= iris.target
np.unique(y)
np.random.seed(123)
indices = np.random.permutation(len(x))
iris_x_train = x[indices[:-10]]
iris_y_train = y[indices[:-10]]
iris_x_test = x[indices[-10:]]
iris_y_test = y[indices[-10:]]
knn = KNC()
knn.fit(iris_x_train, iris_y_train)
KNC(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform')
knn.predict(iris_x_test)
out=knn.predict(iris_x_test)
print("predicted:",out)
print("True :",iris_y_test)

```

The output is shown here:

```
| predicted: [1 1 2 1 2 0 1 1 2 2]  
| True      : [1 1 2 2 1 0 1 1 2 2]
```

Classification

Supervised learning problems can be further divided into two groups:

classification and **regression**. For the classification problem, the output variable, such as y , could be a binary variable, that is, 0 or 1, or several categories. For a regression, variables or values could be discrete or continuous. In the Titanic example, we have 1 for survived and 0 for not survived. For a regression problem, the output could be a value, such as, 2.5 or 0.234. In the previous chapter, we discussed the concept of distance between group members within the same group and between groups.

The logic for classification is that the distance between (among) group members is shorter than the distance between different groups. Alternatively speaking, the similarity between (among) group members is higher than the similarity between (among) different groups or categories. Since categorical data cannot be ranked, we could use the following method to group them:

$$\text{similarity_Index} = \frac{n_{matched}}{n_{total}} \quad (1)$$

Here, $n_{matched}$ is the number of matched treatments and n_{total} is the number of total treatments. When both categorical data and numeric data are available, we could estimate both types of distances first and then choose appropriate weights to combine them.

Assume d_{num} is for the distance based on the numerical data and d_{cat} is for the distance based on the categorical data. Then, we have the following combined distance:

$$d = w_{num} * d_{num} + (1 - w_{num}) * d_{cat} \quad (2)$$

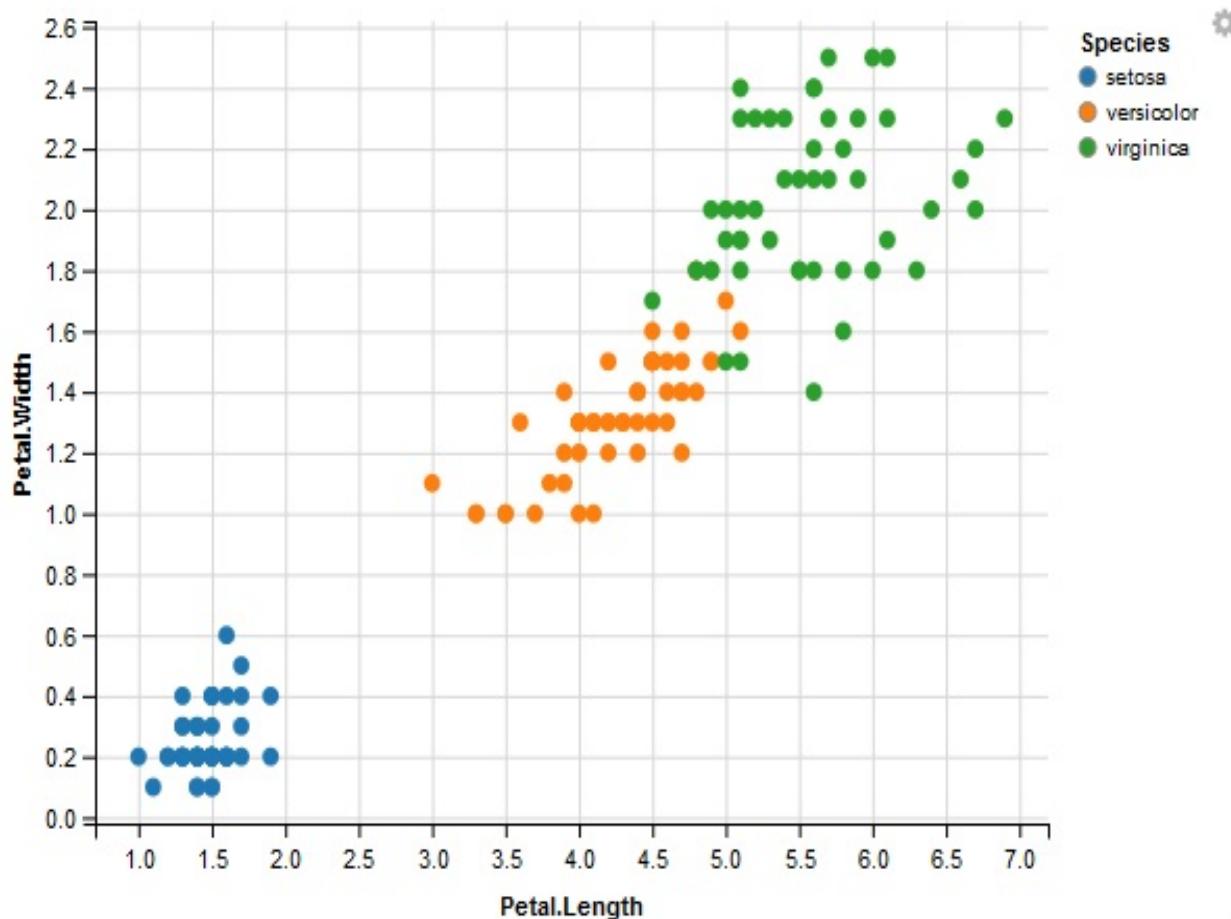
Here, w_{num} is the weight for the numerical value.

The k-nearest neighbors algorithm

In pattern recognition or grouping, the **k-nearest neighbors (k-NN)** algorithm is a non-parametric method implemented for classification and regression. For those two cases, the input consists of the k-closest training examples in the feature space. The following four lines of R code tries to separate plants into k-groups by using a dataset called `iris`:

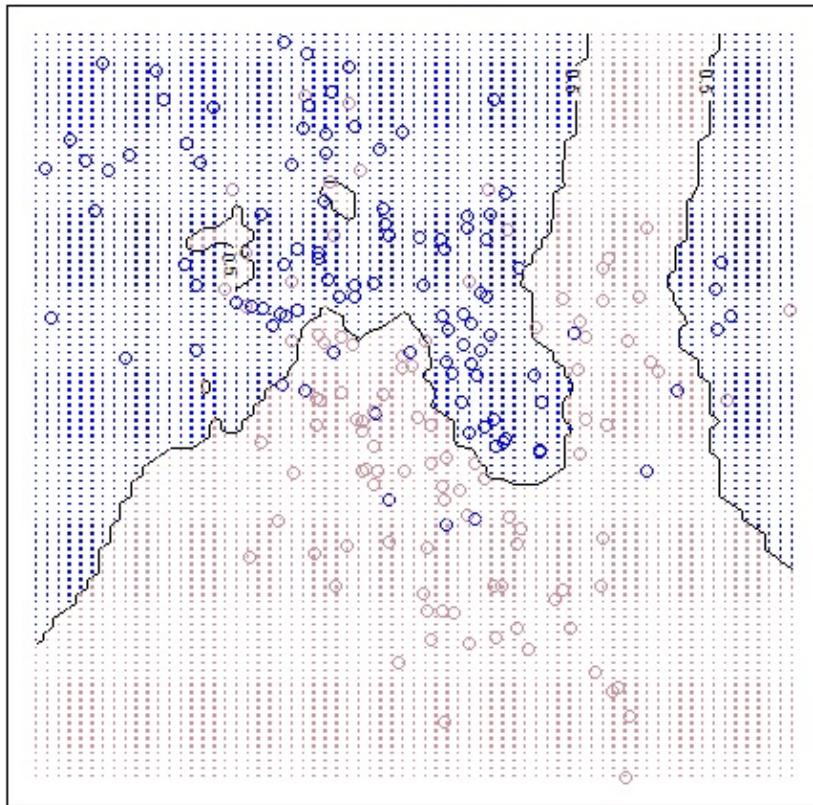
```
library(ggvis)
x<-ggvis
y<-layer_points
iris %>% x(~Petal.Length,~Petal.Width,fill=-Species) %>% y()
```

The graph is shown here:



The following diagram shows the five nearest neighbors:

5 -nearest neighbour



The code generated by the five nearest neighbors is given here. The code is slightly modified from the code offered by others at <https://stats.stackexchange.com/questions/21572/how-to-plot-decision-boundary-of-a-k-nearest-neighbor-classifier-from-elements-o/21602#21602>:

```
library(ElemStatLearn)
require(class)
x <- mixture.example$x
y <- mixture.example$y
xnew <- mixture.example$xnew
px1 <- mixture.example$px1
px2 <- mixture.example$px2
#
color1<-"blue"
color2<-"pink3"
kNearest<-5
model<- knn(x, xnew, y, k=kNearest, prob=TRUE)
title<-paste(kNearest,"-nearest neighbour")
prob <- attr(model,"prob")
prob <- ifelse(model=="1",prob,1-prob)
prob15 <- matrix(prob,length(px1),length(px2))
par(mar=rep(2,4))
contour(px1,px2,prob15,levels=0.5,main=title,axes=FALSE)
points(x, col=ifelse(g==1,color1,color2))
gd <- expand.grid(x=px1, y=px2)
points(gd,pch=".",cex=1.5,col=ifelse(prob15>0.5,color1,color2))
box()
```

For more details about each function, readers can find the related manual regarding the `ElemStatLearn` package.

Bayes classifiers

This is a classification technique based on Bayes theorem with an assumption of independence among predictors. In layman's terms, a naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. First, let's look at the `HouseVotes84` dataset:

```
|> library(mlbench)
|> data(HouseVotes84)
|> head(HouseVotes84)
```

The output is shown in the following screenshot:

```
> head(HouseVotes84)
   Class V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16
1 republican n y n y y y n n n y <NA> y y y n y
2 republican n y n y y y n n n n n y y y n <NA>
3 democrat <NA> y y <NA> y y n n n n y n y y n n
4 democrat n y y n <NA> y n n n n y n y n n y
5 democrat y y y n y y n n n n y <NA> y y y y y
6 democrat n y y n y y n n n n n n y y y y y
> |
```

The first variable, `Class`, is a binary one: `republican` OR `democrat`. In addition, we have 16 traits associated with each individual. The following program calls the `naiveBayes()` function which computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule:

```
library(e1071)
data(HouseVotes84, package = "mlbench")
model<-naiveBayes(Class ~ ., data = HouseVotes84)
#
predict(model, HouseVotes84[1:10,])
predict(model, HouseVotes84[1:10,], type = "raw")
pred <- predict(model, HouseVotes84)
table(pred, HouseVotes84$Class)
```

The final output is given here:

pred	democrat	republican
democrat	238	13
republican	29	155

The next example is to predict the survival rate by applying the naive Bayes methodology to the Titanic dataset:

```
|> library(e1071)
|> data(Titanic)
|> m <- naiveBayes(Survived ~ ., data = Titanic)
|> output<-predict(m, as.data.frame(Titanic))
```

The output is shown here:

```
> print(m)
Naive Bayes Classifier for Discrete Predictors
Call:
naiveBayes.formula(formula = Survived ~ ., data = Titanic)
A-priori probabilities:
Survived
  No      Yes
0.676965 0.323035
Conditional probabilities:
  Class
Survived      1st      2nd      3rd      Crew
  No 0.08187919 0.11208054 0.35436242 0.45167785
  Yes 0.28551336 0.16596343 0.25035162 0.29817159
  Sex
Survived      Male      Female
  No 0.91543624 0.08456376
  Yes 0.51617440 0.48382560
  Age
Survived      Child      Adult
  No 0.03489933 0.96510067
  Yes 0.08016878 0.91983122
> print(output)
[1] Yes No  No  Yes Yes Yes Yes No  No  No  No  Yes Yes
[15] Yes Yes Yes No  No  No  Yes Yes Yes Yes No  No  No  No
[29] Yes Yes Yes Yes
Levels: No Yes
```

From the previous output, we could find the final prediction in terms of 32 individuals.

```
> library("ReinforcementLearning") > set.seed(123)

> data <- sampleGridSequence(1000) > dim(data)

<strong>[1] 1000 4</strong> > head(data)

<strong> State Action Reward NextState 1 s2 left -1 s2
2 s4 right -1 s4
3 s2 down -1 s2
4 s4 up -1 s4
```

5 s4 up -1 s4

```
6 s1 left -1 s1</strong> > unique(data$State) <strong>[1] "s2" "s4" "s1" "s3"
</strong> > unique(data$Action) <strong>[1] "left" "right" "down" "up"
</strong> > unique(data$NextState) <strong>[1] "s2" "s4" "s1" "s3"</strong> >
unique(data$Reward) <strong>[1] -1 10</strong>

> x<-subset(data,data$State==data$NextState) > head(x)

<strong> State Action Reward NextState 1 s2 left -1 s2
2 s4 right -1 s4
3 s2 down -1 s2
4 s4 up -1 s4
```

5 s4 up -1 s4

```
6 s1 left -1 s1</strong> > unique(x$Reward)

<strong>[1] -1</strong>

library(ReinforcementLearning) set.seed(123)
```

```
data <- sampleGridSequence(1000) control <- list(alpha = 0.1, gamma = 0.1,  
epsilon = 0.1) model <-  
ReinforcementLearning(data,s="State",a="Action",r="Reward",s_new="NextState",  
print(model)
```

```
> print(model)
```

State-Action function Q

```
right up down left s1 -1.09768561 -1.098833 -1.00284548 -1.098910  
s2 -0.03076799 -1.099211 -1.00501562 -1.005837  
s3 -0.02826295 9.808764 -0.02869875 -1.003904  
s4 -1.10177002 -1.106688 -1.10776585 -1.106276
```

Policy

s1 s2 s3 s4

"down" "right" "up" "right"

Reward (last iteration) [1] -505

Implementation of supervised learning via R

As we have discussed in the previous chapter, the best choice to conduct various tests for supervised learning is applying an R package called `Rattle`. Here, we show two more examples. Let's first look at the `iris` dataset:

```
|> library(rattle)  
|> rattle()
```

The next example is using the `diabetes` dataset, shown in the screenshot here:

The screenshot shows the Rattle software interface. At the top, there is a menu bar with options like File, ARFF, ODBC, R Dataset, RData File, Library, Corpus, and Script. Below the menu is a toolbar with buttons for Partition (checkbox checked), Seed (set to 42), View, and Edit. A dropdown menu is open over the 'Data Name' field, which contains the value 'diabetes:ddalpha:Data for Classification'. To the right of the dropdown is a small downward arrow icon. Below the toolbar, there are two buttons: 'Input' (green circle) and 'Ignore' (red circle). Next to them is a 'Weight Calculator' input field. To the right of the calculator is a 'Target Data Type' section with radio buttons for Auto (selected), Categorical, Numeric, and Survival. The main area is a table with the following data:

No.	Variable	Data Type	Input	Target	Risk	Ident	Ignore	Weight	Comment
1	X1	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 17
2	X2	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 136
3	X3	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 47
4	X4	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 51
5	X5	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 186
6	X6	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 248
7	X7	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 517
8	X8	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 52
9	C	Numeric	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2

For example, we could choose the logistic model after clicking Model on the menu bar. After clicking on Execute, we would have the following output:

Data Explore Test Transform Cluster Associate Model Evaluate Log

Type: Tree Forest Boost SVM Linear Neural Net Survival All

Numeric Generalized Poisson Logistic Probit Multinomial

Plot

Summary of the Logistic Regression model (built using glm):

Call:

```
glm(formula = C ~ ., family = binomial(link = "logit"), data = crs$dataset[crs$train,
  c(crs$input, crs$target)])
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.2855	-0.6672	-0.3782	0.6813	3.0690

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	-8.973429	0.884379	-10.147	< 2e-16 ***
X1	0.119559	0.038208	3.129	0.00175 **
X2	0.039366	0.004632	8.499	< 2e-16 ***
X3	-0.014184	0.006245	-2.271	0.02312 *
X4	0.005091	0.008585	0.593	0.55315
X5	-0.001426	0.001037	-1.375	0.16908
X6	0.089911	0.018733	4.800	0.00000159 ***
X7	0.477225	0.385543	1.238	0.21579
X8	0.021502	0.011310	1.901	0.05728 .

Signif. codes: 0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

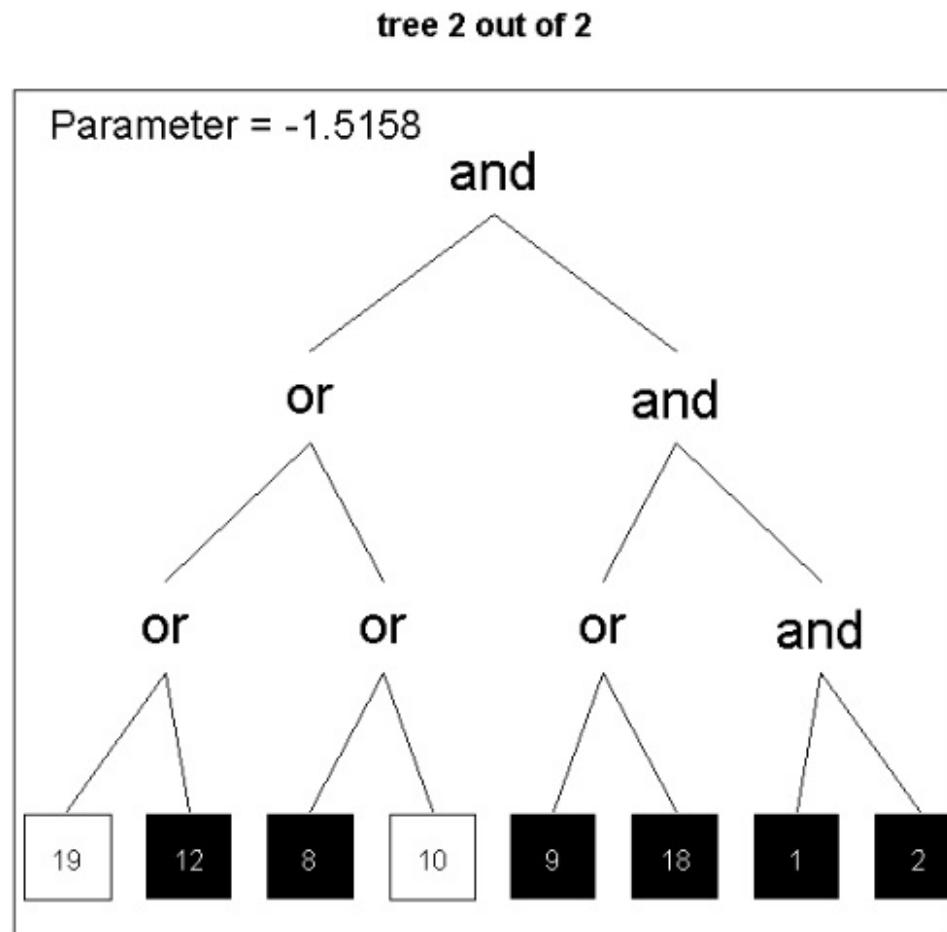
(Dispersion parameter for binomial family taken to be 1)

Based on the significant level of p-values, we could see that in addition to the intercept, x₁, x₂, x₃, and x₆ are statistically significant.

The next example is from the R package called `LogicReg`. The code is given here:

```
library(LogicReg)
data(logreg.testdat)
y<-logreg.testdat[,1]
x<-logreg.testdat[, 2:21]
n=1000
n2=25000
set.seed(123)
myanneal<-logreg.anneal.control(start=-1,end=-4,iter=n2,update=n)
output<-logreg(resp=y,bin=x,type=2,select = 1,ntrees=2,anneal.control=myanneal)
plot(output)
```

The related graph is shown here. Again, the `set.seed()` function is used for an easy replication. If the user omits the function or chooses a different seed, they would get quite a different result:



Introduction to RTextTools

This R package is about *Automatic Text Classification via Supervised Learning*. It is a machine learning package for automatic text classification that makes it simple for novice users to get started with machine learning, while allowing experienced users to easily experiment with different settings and algorithm combinations. The package includes nine algorithms for ensemble classification (svm, slda, boosting, bagging, random forests, glmnet, decision trees, neural networks, and maximum entropy), comprehensive analytics, and thorough documentation. Here, we use the New York Times Times article as an example. First, let's look at the data:

```
|>library(RTextTools)
|>data(NYTimes)
|>set.seed(123) # guarantees the same result
|>data <- NYTimes[sample(1:3100, size=100, replace=FALSE), ]
|>head(data)
```

When running the program, users should move this line. The output is shown here:

```
> head(data)
   Article_ID      Date                      Title
892     33741 14-Jan-99 Removal of Healthy Breasts Is Found to Cut Cancer Risk
2443     7711 17-Apr-04 In Afghanistan, U.S. Envoy Sits In Seat of Power
1268    19020 25-Apr-00 Iran Reformers Feeling Pressed By Hard-Liners
2735    27021 25-Jun-05 Arbitration Points to 10% Raise for City Police
2912    37154 25-Mar-06 New York City Meets Stranger From the West: the Brush Fire
141     42842 22-Jun-96 Serb Gangs Rule in Last-Chance Fief of U.N.
                                         Subject Topic.Code
892          At risk women take drastic measures to prevent breast cancer      3
2443 Key role of U.S. Ambassador to Afghanistan in the new government's policy.  16
1268          Iranian reform politics                                19
2735          Salary disputes of NYPD                               12
2912          brush fires in NYC                                27
141          UN involvement in Eastern Slavonia                  19
```

Next, we could look at the types of existing algorithms by using the `print_algorithms()` function, shown in the code and output here:

```
|>library(RTextTools)
```

```
|>print_algorithms()
|[1] "BAGGING"   "BOOSTING"  "GLMNET"    "MAXENT"    "NNET"
|[6] "RF"        "SLDA"      "SVM"       "TREE"
```

The `BAGGING` is for zbc, `BOOSTING` is the bbb, `GLMNET` is for the general linear model, `MAXENT` is for maximum entropy model, `NNET` is for neural network, `RF` is for random forest, `SLDA` is for supervised machine learning algorithms, `SVM` is for support vector machine, and `TREE` is for decision trees. The code is shown here:

```
library(RTextTools)
data(NYTimes)
data <- NYTimes[sample(1:3100, size=100, replace=FALSE), ]
matrix <- create_matrix(cbind(data["Title"], data["Subject"]), language="english",
removeNumbers=TRUE, stemWords=FALSE, weighting=tm::weightTfIdf)
container <- create_container(matrix, data$Topic.Code, trainSize=1:75, testSize=76:100,
virgin=TRUE)
models <- train_models(container, algorithms=c("MAXENT", "SVM"))
results <- classify_models(container, models)
analytics <- create_analytics(container, results)
summary(analytics)
```

Implementation via Python

In the previous chapter related to unsupervised learning, we have learnt about several Python packages. Fortunately, these packages can be applied to supervised learning algorithms as well. The following example is for a linear regression by using a few Python datasets. The Python dataset can be downloaded from the author's website at <http://www.canisius.edu/~yany/python/ffcMonthly.pkl>. Assume that the data is saved under `c:/temp/`:

```
import pandas as pd
x=pd.read_pickle("c:/temp/ffcMonthly.pkl")
print(x.head())
print(x.tail())
```

The output is shown here:

	MKT_RF	SMB	HML	Rf	MOM
1927-01-01	-0.0006	-0.0056	0.0483	0.0025	0.0044
1927-02-01	0.0418	-0.0010	0.0317	0.0026	-0.0201
1927-03-01	0.0013	-0.0160	-0.0267	0.0030	0.0359
1927-04-01	0.0046	0.0043	0.0060	0.0025	0.0419
1927-05-01	0.0544	0.0141	0.0493	0.0030	0.0301
	MKT_RF	SMB	HML	Rf	MOM
2016-06-01	-0.0005	0.0061	-0.0149	0.0002	0.0428
2016-07-01	0.0395	0.0290	-0.0098	0.0002	-0.0317
2016-08-01	0.0050	0.0094	0.0318	0.0002	-0.0316
2016-09-01	0.0025	0.0200	-0.0134	0.0002	-0.0052
2016-10-01	-0.0202	-0.0440	0.0415	0.0002	0.0058

We plan to run a linear regression; see the formula here:

$$R_i = \alpha + \beta_1(R_{mkt} - R_f) + \beta_2 R_{SMB} + \beta_3 R_{HML} \quad (3)$$

Here, R_i is stock i's returns, R_{mkt} is the *market returns*, R_{SMB} is the *portfolio returns of small stocks* minus the *portfolio returns of big stocks*, R_{HML} is the *portfolio returns with high book-to-market ratio* (of equity) minus the *portfolio returns of stocks with low book-to-market ratio*. The Python program is given next. Note that the Python dataset called `ffDaily.pkl` is downloadable at <http://canisius.edu/~yany/python/data/ffDaily.pkl>:

```

import scipy as sp
import pandas as pd
import quandl as qd
import statsmodels.api as sm
#quandl.ApiConfig.api_key = 'YOUR_API_KEY'
a=qd.get("WIKI/IBM")
p=a['Adj. Close']
n=len(p)
ret=[]
#
for i in range(n-1):
    ret.append(p[i+1]/p[i]-1)
#
c=pd.DataFrame(ret,a.index[1:n],columns=['RET'])
ff=pd.read_pickle('c:/temp/ffDaily.pkl')
final=pd.merge(c,ff, left_index=True, right_index=True)
y=final['RET']
x=final[['MKT_RF', 'SMB', 'HML']]
#x=final[['MKT_RF']]
x=sm.add_constant(x)
results=sm.OLS(y,x).fit()
print(results.summary())

```

The output is shown here:

OLS Regression Results						
=====						
Dep. Variable:	RET	R-squared:	0.373			
Model:	OLS	Adj. R-squared:	0.373			
Method:	Least Squares	F-statistic:	2805.			
Date:	Wed, 18 Apr 2018	Prob (F-statistic):	0.00			
Time:	10:15:11	Log-Likelihood:	41169.			
No. Observations:	14135	AIC:	-8.233e+04			
Df Residuals:	14131	BIC:	-8.230e+04			
Df Model:	3					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[95.0% Conf. Int.]	
=====						
const	0.0002	0.000	1.642	0.101	-3.52e-05	0.000
MKT_RF	0.9217	0.012	79.182	0.000	0.899	0.945
SMB	-0.4164	0.022	-19.361	0.000	-0.459	-0.374
HML	-0.4337	0.023	-18.962	0.000	-0.479	-0.389
=====						
Omnibus:	16154.987	Durbin-Watson:	1.988			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	19838982.354			
Skew:	-5.038	Prob(JB):	0.00			
Kurtosis:	186.258	Cond. No.	214.			
=====						

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

The next example predicts `iris` categories. The code is given first:

```

from sklearn import metrics
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
x=datasets.load_iris()
model=DecisionTreeClassifier()
model.fit(x.data, x.target)
print(model)
true=x.target
predicted=model.predict(x.data)
print("----- output below -----")
print(metrics.classification_report(true, predicted))
print(metrics.confusion_matrix(true, predicted))

```

The nice output is shown here:

```

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                      splitter='best')
----- output below -------

          precision    recall   f1-score   support
          0         1.00     1.00     1.00      50
          1         1.00     1.00     1.00      50
          2         1.00     1.00     1.00      50

avg / total       1.00     1.00     1.00     150

[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]

```

```

import matplotlib.pyplot as plt

from sklearn import datasets, svm, metrics from sklearn.metrics import
classification_report as report

#



format1="Classification report for classifier %s:n%sn"
format2="Confusion matrix:n%os"

digits = datasets.load_digits()

imageLabels = list(zip(digits.images, digits.target)) for index,(image,label) in
enumerate(imageLabels[:4]): plt.subplot(2, 4, index + 1) plt.axis('off')

plt.imshow(image,cmap=plt.cm.gray_r,interpolation='nearest')
plt.title('Training: %i' % label) n=len(digits.images)

data = digits.images.reshape((n,-1))

classifier = svm.SVC(gamma=0.001)

classifier.fit(data[:n//2],digits.target[:n//2]) expected = digits.target[n//2:]

predicted = classifier.predict(data[n//2:]) print(format1 %
(classifier,report(expected, predicted))) print(format2 %
metrics.confusion_matrix(expected,predicted))

imageAndPrediction=list(zip(digits.images[n//2:],predicted)) for index,
(image,prediction) in enumerate(imageAndPrediction[:4]):
plt.subplot(2,4,index+5) plt.axis('off')

plt.imshow(image,cmap=plt.cm.gray_r,interpolation='nearest')
plt.title('Prediction: %i' % prediction) plt.show()

```

Part of the output is shown here:

```
Classification report for classifier SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
max_iter=-1, probability=False, random_state=None, shrinking=True,
tol=0.001, verbose=False):
```

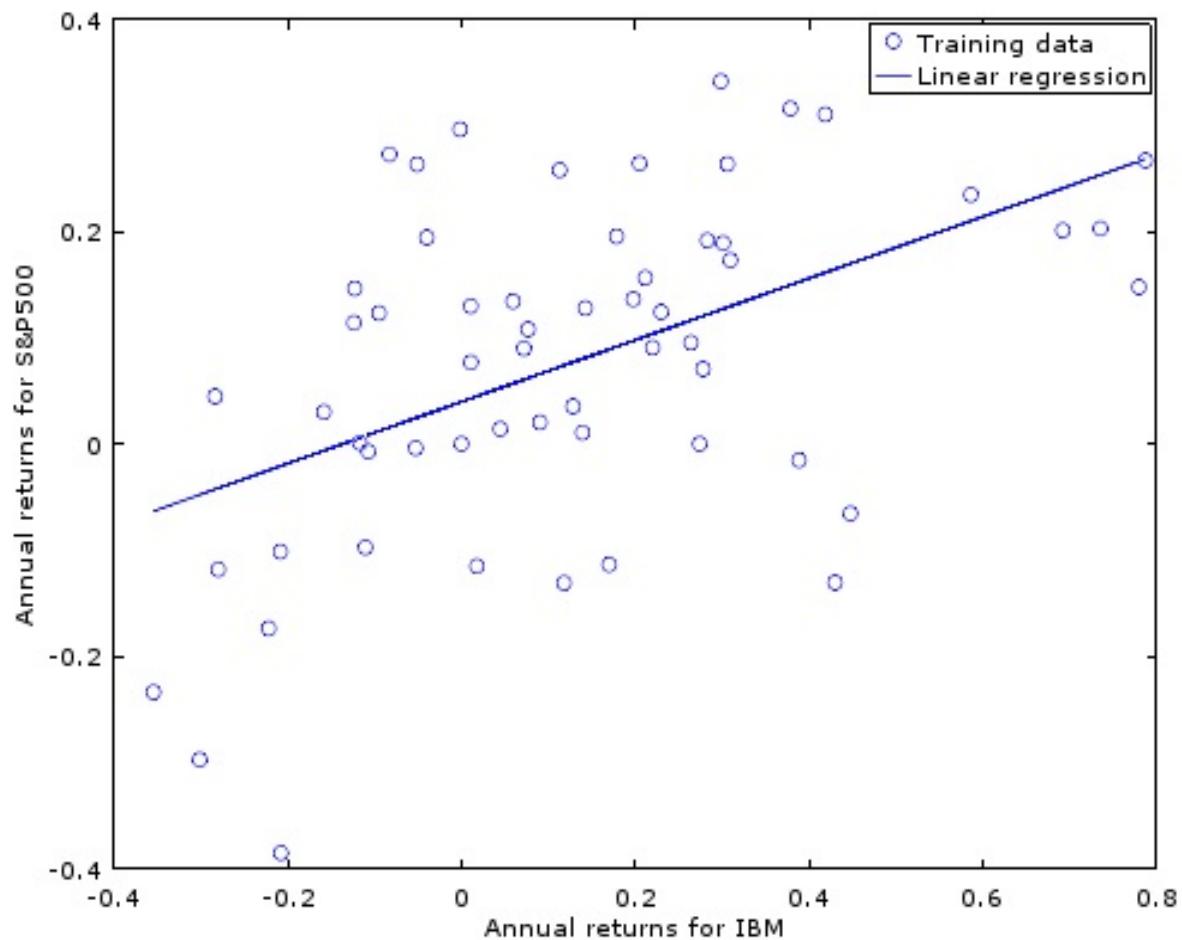
	precision	recall	f1-score	support
0	1.00	0.99	0.99	88
1	0.99	0.97	0.98	91
2	0.99	0.99	0.99	86
3	0.98	0.87	0.92	91
4	0.99	0.96	0.97	92
5	0.95	0.97	0.96	91
6	0.99	0.99	0.99	91
7	0.96	0.99	0.97	89
8	0.94	1.00	0.97	88
9	0.93	0.98	0.95	92
avg / total	0.97	0.97	0.97	899

Implementation via Octave

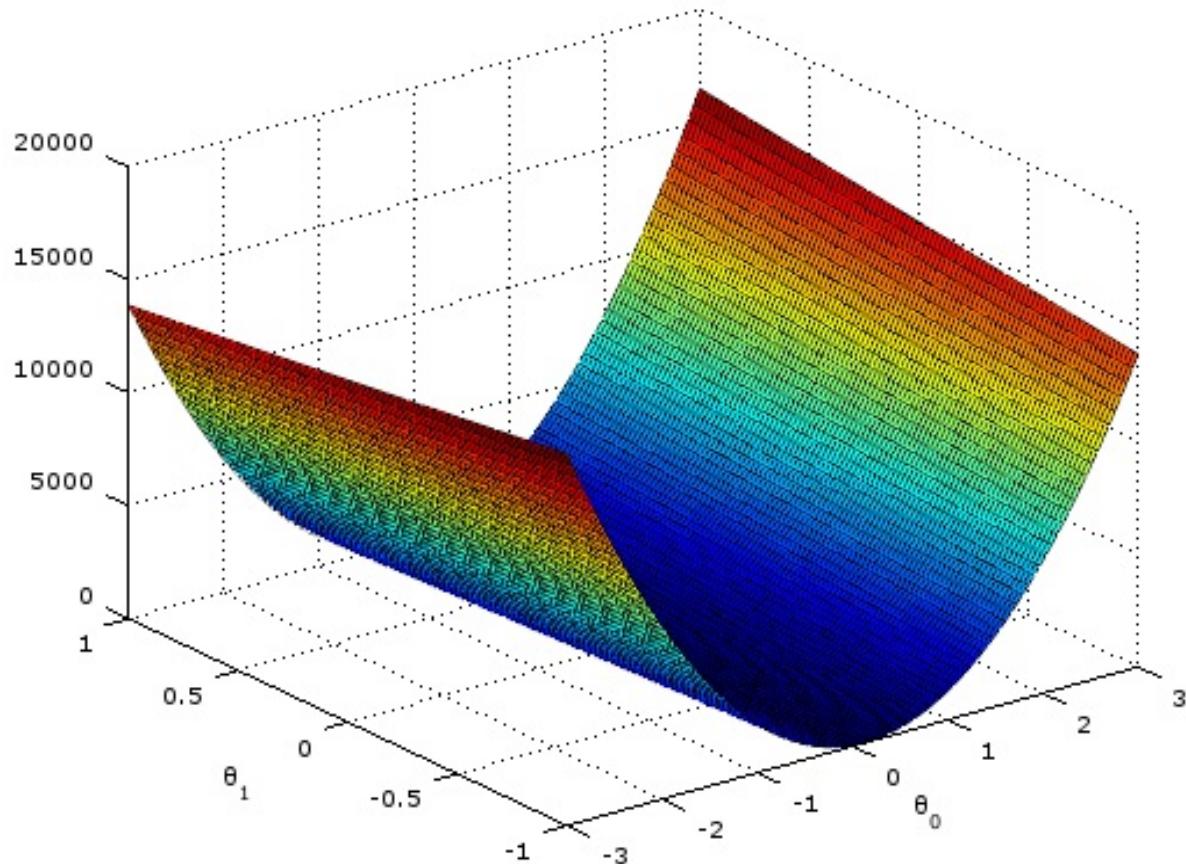
The next example of running a linear regression and the related datasets could be downloaded at http://canisius.edu/~yany/data/c9_input.csv. In the following program, the input data set is assumed to be under c:/temp:

```
a=csvread("c:/temp/c9_input.csv");
x=a(:,2);
y=a(:,3);
figure % open a window for graph
plot(x, y, 'o');
ylabel('Annual returns for S&P500')
xlabel('Annual returns for IBM')
```

The first graph is shown here:



To save space, the long program will not be shown here. Interested readers can use the previous link. However, its output graph is shown instead:



For the next example, we download an Octave machine library at <https://github.com/partharamanujam/octave-ml>. Assume that the location of the directory is `C:\Octave\octave-ml-master` and the related `octavelib` is `C:\Octave\octave-ml-master\octavelib`. Then we add its path to our Octave program with the following one-liner:

```
| addpath(genpath('C:\Octave\octave-ml-master\octavelib'));
```

There are many useful programs included under the subdirectory. Note that the `.m` extensions are all removed in the following table for brevity. Some useful programs included in the directory are presented in the table here:

addBiasTerm	kMeansClosestCentroids
choosePolynomialForLinearGradDesc	kMeansComputeCentroids
chooseRBParamsForSVM	kMeansCostFunction
chooseRegularizationForLinearGradDesc	kMeansInitCentroids
chooseRegularizationForLogisticGradDesc	linearRegressionCostFunction
collaborativeFilteringCostFunction	logisticRegressionCostFunction
computeCoFiParamsByGradDescFmincg	logisticRegressionOneVsAllError
computeGaussianParams	logisticRegressionOneVsAllThe
computeMultivarGaussianDistribution	normalizeRatings
computePCA	porterStemmer
computeScalingParams	predictByLinearGradDesc
computeThetaByLinearGradDescFminunc	predictByLogisticGradDescOne

computeThetaByLogisticGradDescFminunc	predictByNormalEquation
computeThetaByNormalEquation	projectPCAData
computeThresholdForMultivarGaussian	recoverPCAData
fmincg	scaleFeatures
generateFeaturesPolynomial	sigmoid
generateKMeansClusters	
generateKMeansClustersMinCost	

Table 9.1 Supporting Octave programs under the octavelib directory

In addition, the sample programs and their related input datasets are included under the subdirectory called `examples` (see the following table):

Programs	Dataset(s)
cofi.m	movieList.mat
extractEmailFeatures.m	email-sample-1.txt email-sample-2.txt email-sample-3.txt

gaussian_m.m	anomaly.dat
initEnv.m	Note: setup program
k_means.m	bird_small.png
linear_gd.m	damlevels.mat
logistic_gd.m	numbers.mat
normal_eq.m	prices.csv
pca.m	faces.mat
svm.m	spam-vocab.txt

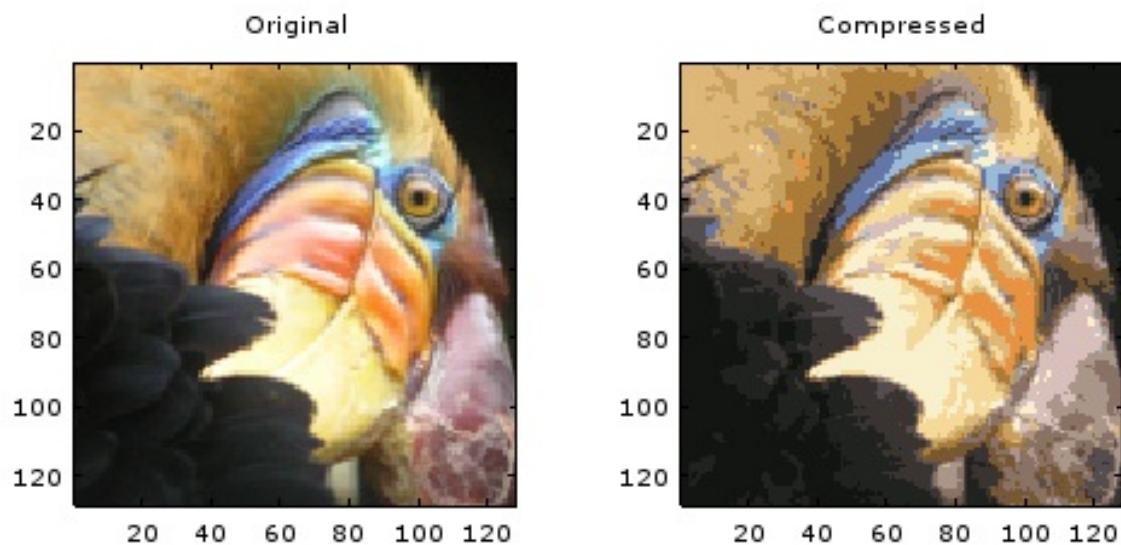
Table 9.2 Examples and their related datasets

Let's look at the following program for k-means. The original program is called `k_means.m` and its input photo is `bird_small.png`. The program tries to save space for a given picture by using the k-means method:

```
A = double(imread('bird_small.png'));
A = A / 255; % Divide by 255, values in the range 0 - 1
imgSize = size(A);
X = reshape(A, imgSize(1) * imgSize(2), 3);
k = 16; % using 4-bit (16) colors, minimize cost
[Centroids, idx, cost]=generateKMeansClustersMinCost(X,k,10,10);
fprintf('Cost/Distortion of computed clusters:%.3fn', cost);
% regenerate colors & image
NewX = Centroids(idx, :);
NewA = reshape(NewX, imgSize(1), imgSize(2), 3);
% compare both the images
```

```
fprintf('Comparing original & compressed imagesn');
subplot(1, 2, 1);
imagesc(A);
axis("square");
title('Original');
subplot(1, 2, 2);
imagesc(NewA);
axis("square");
title('Compressed');
```

The related two photos are shown here:

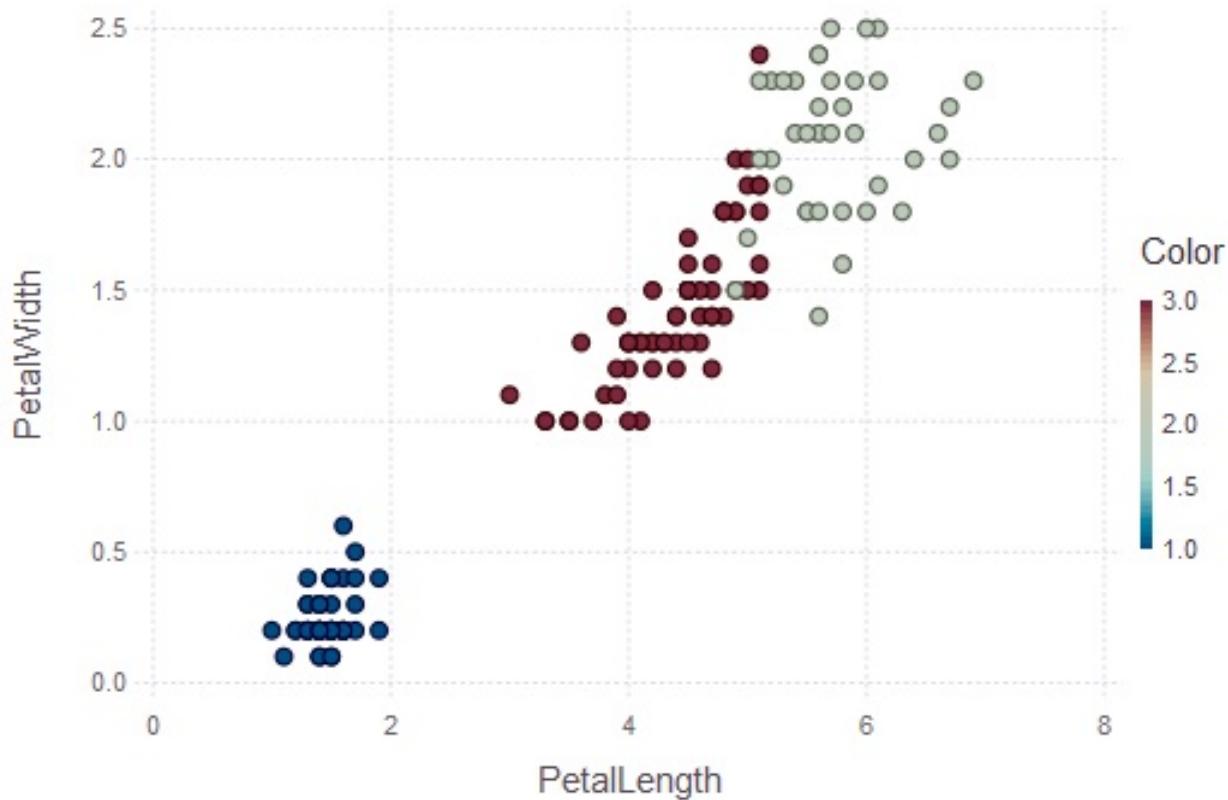


Implementation via Julia

The first example uses the familiar dataset called `iris` again. Using the `kmeans()` function, the program tries to group these plants:

```
using Clustering
iris = dataset("datasets", "iris")
head(iris)
features=permutedims(convert(Array, iris[:,1:4]),[2, 1])
result=kmeans(features,3)
nameX="PetalLength"
nameY="PetalWidth"
assignments=result.assignments
plot(iris,
x=nameX,y=nameY,color=assignments,Geom.point)
```

The related output is shown here:



For the next example, we try to sort a set of random numbers into 20 clusters. The code is shown here:

```
using Clustering
srand(1234)
nRow=5
```

```

nCol=1000
x = rand(nRow,nCol)
maxInter=200 #max interation
nCluster=20
R = kmeans(x,nCluster;maxiter=maxInter,display=:iter)
@assert nclusters(R) ==nCluster
c = counts(R)
clusters=R.centers

```

To show the value of `x` and `clusters`, we simply type them on the command line, shown here:

```

julia> x
5x1000 Array{Float64,2}:
 0.590845  0.854147  0.648882  0.112486   .  0.853632  0.325751  0.707301
 0.766797  0.200586  0.0109059 0.276021    .  0.151614  0.798595  0.608648
 0.566237  0.298614  0.066423  0.651664    .  0.862541  0.115767  0.0761294
 0.460085  0.246837  0.956753  0.0566425   .  0.52499  0.401829  0.901916
 0.794026  0.579672  0.646691  0.842714    .  0.932708  0.994582  0.348597

julia> clusters
5x20 Array{Float64,2}:
 0.731216  0.80513  0.306281  0.322454   .  0.745429  0.727124  0.278213
 0.799427  0.212387 0.191713  0.280681   .  0.708182  0.709777  0.216027
 0.808221  0.293112 0.733413  0.200435   .  0.663792  0.192699  0.654714
 0.309802  0.713249  0.799372  0.181764   .  0.755552  0.792906  0.329358
 0.595867  0.33937  0.266841  0.450164   .  0.719329  0.395103  0.259146

```

Task view for machine learning in R

From the previous chapters, we know that there are about three dozen task views for different topics. A task view is a set of R packages around a specific topic such as finance, econometrics, and the like. In the previous chapter, we briefly discussed the task view for clustering. There is no task view with a name of supervised learning. Instead, the closest one will be the machine learning task view; see the screenshot here for the top part:

CRAN Task View: Machine Learning & Statistical Learning

Maintainer: Torsten Hothorn

Contact: Torsten.Hothorn at R-project.org

Version: 2018-02-28

URL: <https://CRAN.R-project.org/view=MachineLearning>

Several add-on packages implement ideas and methods developed at the borderline between computer science and statistics - this field of research is usually referred to as machine learning. The packages can be roughly structured into the following topics:

- *Neural Networks and Deep Learning* : Single-hidden-layer neural network are implemented in package [nnet](#) (shipped with base R). Package [RSNNS](#) offers an interface to the Stuttgart Neural Network Simulator (SNNS). An interface to the FCNN library allows user-extensible artificial neural networks in package [FCNN4R](#). [rnn](#) implements recurrent neural networks. Packages implementing deep learning flavours of neural networks include [deepnet](#) (feed-forward neural network, restricted Boltzmann machine, deep belief network, stacked autoencoders), [RcppDL](#) (denoising autoencoder, stacked denoising autoencoder, restricted Boltzmann machine, deep belief network) and [h2o](#) (feed-forward neural network, deep autoencoders). An interface to [tensorflow](#) is available in [tensorflow](#).

The URL is given in the preceding screenshot. Again, we could issue just three lines of R code to install the R packages included in the task view:

```
>install.packages("ctv") >library("ctv") >install.views("MachineLearning")
```

About 105 related R packages were installed as of April 3, 2018.

Summary

In this chapter, we have discussed supervised learning, such as classification, the k-nearest neighbors algorithm, Bayes classifiers, reinforcement learning, and the `RTextTools` and `sklearn` modules in R. In addition, we discussed implementations of supervised learning via R, Python, Julia, and Octave.

For the next chapter, we will discuss predictive data analytics, modeling and validation, some useful datasets, time-series analytics, how to predict the future, seasonality, and how to visualize our data. For Python packages, we will mention `predictives-models-building`, `model-catwalk`, and `easyML`. For R packages, we will discuss `datarobot`, `LiblineaR`, `eclust`, and `AppliedPredictiveModeling`. For Julia packages, we will explain `EmpiricalRisks` and `ValidatedNumerics`.

Review questions and exercises

1. What does *unsupervised learning* mean?
2. What is the major difference between unsupervised learning and supervised learning?
3. How do we install the Python package `sklearn`?
4. Discuss the relationship between distance and clustering classification.
5. What does Bayes classification mean?
6. Find out the related functions for Bayes classification in R, Python, Octave, and Julia.

7. How many R packages are installed after you run the following three lines of R code?

```
| >install.packages("ctv")
| >library("ctv")
| >install.views("MachineLearning")
```

8. Download the IBM monthly data from Yahoo!Finance, <https://finance.yahoo.com>. Then run a Fama-French-Carhart four factor model by using Python. The 4-factor model is given here:

$$R_i = R_f + \beta_{mkt}(R_{mkt} - R_f) + \beta_{SMB}SMB + \beta_{HML}HML + \beta_{MOM}MOM + \varepsilon$$

The Python dataset related to those 4-factors can be downloaded from the author's website at <http://www.canisius.edu/~yany/python/ffcMonthly.pkl>.

9. Download the famous iris data set from the UCI Machine Learning Repository. The first several lines are given here:

```

5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa

```

Then generate data sets for both R and Python.

10. To implement machine learning using Octave, how is the usage of the `octavelib` library introduced and used in the chapter? How many programs are under it?
11. From the provided Octave programs discussed in this chapter, run the program called `logistic_gd.m` with an input Octave data set called `numbers.mat`. For more details, see Table 9.1 and 9.2 and the related explanations.
12. Discuss the `spa` R package and show a few related examples.
13. Download Fama-French-Carhart 4-factor model, see here:

$$R_i = \alpha + \beta_1(R_{mkt} - R_f) + \beta_2 R_{SMB} + \beta_3 R_{HML} + \beta_4 R_{MOM}$$

where R_i is stock i 's returns, R_{mkt} is the market returns, R_{SMB} is the portfolio returns of small stocks minus the portfolio returns of big stocks, R_{HML} is the portfolio returns with high book-to-market ratio (of equity) minus the portfolio returns of stocks with low book-to-market ratio, R_{MOM} is the momentum factor. Write R/Python/Octave/Julia programs to run linear regressions for IBM stocks. Source of Data: 1) Prof. French's Data Library, http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html, and Yahoo!Finance, <https://finance.yahoo.com>.

14. Download the Bank Marketing Data Set from the UCI Machine Learning Repository, <https://archive.ics.uci.edu/ml/datasets/Bank+Marketing>. The data is related to direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. Use supervised

learning to identify the most influential factors.

15. Download a data set called *Census Income*, <https://archive.ics.uci.edu/ml/datasets/Census+Income>. Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the following conditions: $((\text{AGE} > 16) \ \&\& (\text{AGI} > 100) \ \&\& (\text{AFNLWGT} > 1) \ \&\& (\text{HRSWK} > 0))$. Write both R and Python programs to predict whether a person makes over \$50,000 a year.
16. The utility function for an investor is shown here:

$$U = E(R) - \frac{1}{2}A * \sigma^2$$

where U is the utility function, E(R) is the expected portfolio return and we could use its mean to approximate, A is the risk-averse coefficient and σ^2 is the variance of the portfolio. Go to Professor French's Data Library to download 10 industry portfolio returns. Assume a risk-averse variable of A varies from 1 to 100. Use the k-means method to group investors based on E(R), A, and risk.

17. Download a data set called Bike Sharing Data Set from the UCI Machine Learning Repository, <https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset>. The dataset contains the hourly and daily count of rental bikes between years 2011 and 2012 in Capital bike share system with the corresponding weather and seasonal information. Write an R program to group individual riders into k-groups.
18. Download a data set call Bag of Words, <https://archive.ics.uci.edu/ml/datasets/Bag+of+Words>. Write both R and Python to conduct a cluster analysis.
19. Download a data set called Sonar Dataset. The Sonar Dataset involves the prediction of whether or not an object is a mine or a rock given the strength of sonar returns at different angles.
20. Use the data set called Boston to run a regression (Python). The first several lines are shown here:

```
|     from sklearn import datasets  
|     boston = datasets.load_boston()  
|     print(boston.data.shape)  
|     (506, 13)
```


Predictive Data Analytics – Modeling and Validation

Our utmost objective in conducting various data analyses is trying to find patterns in order to predict what might happen in the future. For the stock market, researchers and professionals are conducting various tests to understand market mechanisms. In this case, many questions could be asked. What will the market index level be in the next five years? What will IBM's price range be next year? Will the market volatility increase or decrease in the future? What might be the impact if governments change their tax policies? What is the potential gain and loss if one country launches a trade war with another one? How do we predict a consumer's behavior by analyzing some related variables? Could we predict the probability that an undergraduate student will successfully graduate? Could we find an association between certain behaviors of one specific disease?

In this chapter, the following topics will be covered:

- Understanding predictive data analytics
- Useful dataset
- Predicting future events
- Model selection
- Granger causality test

Understanding predictive data analytics

In terms of future events, people could have many questions. For an investor, if he/she could predict the future movement of a stock price, he/she could make more profit. For a company, if they could forecast the trend of their products, they could increase their stock price and products' market shares. For governments, if they could predict the impact of an aging population on society and the economy, they would have more incentive to design a better policy in terms of government budget and other related strategic decisions.

For universities, if they could have a good grasp of the market demand in terms of quality and skill sets for their graduates, they could design a set of better programs or launch new programs to satisfy the future needs in terms of a labor force.

For a better prediction or forecast, researchers have to consider many issues. For example, is the sample too small? How do we remove missing variables? Is this dataset biased in terms of data collection procedures? How do we treat extreme values or outliers? What is the seasonality and how do we deal with it? What kinds of models should we apply? In this chapter, some of these issues will be touched upon. We start with the useful dataset.

Useful datasets

One of the best data sources is the **UCI Machine Learning Repository**. When we go to the web page at <https://archive.ics.uci.edu/ml/datasets.html>, we see the following list:

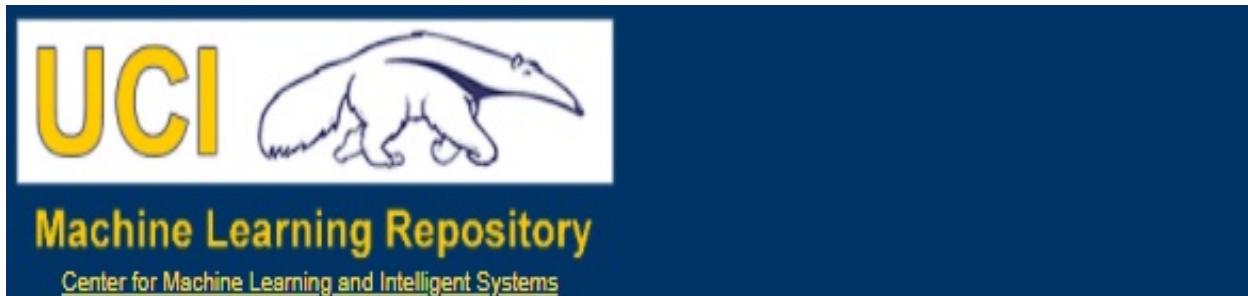


Machine Learning Repository
Center for Machine Learning and Intelligent Systems

Browse Through: 427 Data Sets

Default Task	Name	Data Types
Classification (314)		
Regression (82)		
Clustering (74)		
Other (54)		
Attribute Type		
Categorical (37)		
Numerical (274)		
Mixed (55)		
Data Type		
Multivariate (325)		
Univariate (19)		
Sequential (45)		
Time-Series (79)		
Text (44)		
Domain-Theory (23)		
Other (21)		
Area		
Life Sciences (97)		
Physical Sciences (47)		
CS / Engineering (150)		
Social Sciences (24)		
Business (26)		
Game (10)		
Other (69)		
# Attributes		
Less than 10 (98)		
10 to 100 (195)		
Greater than 100 (78)		
# Instances		
Less than 100 (25)		
100 to 1000 (148)		
Greater than 1000 (222)		
Format Type		
Matrix (293)		
Non-Matrix (134)		
Abalone	Multivariate	
Adult	Multivariate	
Annealing	Multivariate	
Anonymous Microsoft Web Data		
Arrhythmia	Multivariate	
Artificial Characters	Multivariate	
Audiology_(Original)	Multivariate	
Audiology_(Standardized)	Multivariate	
Auto MPG	Multivariate	
Automobile	Multivariate	

For example, if we click the first dataset (Abalone), we see the following. To save space, only the top part is shown:



Abalone Data Set

[Download](#) [Data Folder](#) [Data Set Description](#)

Abstract: Predict the age of abalone from physical measurements



Data Set Characteristics:	Multivariate	Number of Instances:	4177	Area:	Life
Attribute Characteristics:	Categorical, Integer, Real	Number of Attributes:	8	Date Donated	1995-12-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	577549

From the web page, users can download the dataset and find definitions of variables and even citations. The code that follows can be used to download a related R dataset:

```

dataSet<- "UCIdatasets"
path<- "http://canisius.edu/~yany/RData/"
con<- paste(path,dataSet,".RData",sep='')
load(url(con))
dim(.UCIdatasets)
head(.UCIdatasets)

```

The related output is shown here:

```

> dim(.UCIdatasets)
[1] 427   7
> head(.UCIdatasets)
      Name Data.Types Default_Task Attribute.Types N_Instances N_Attributes Year
1 Abalone Multivariate Classification Categorical, Integer, Real    4177           8 1995
2 Adult Multivariate Classification Categorical, Integer    48842          14 1996
3 Annealing Multivariate Classification Categorical, Integer, Real     798           38 NA
4 Anonymous Microsoft Web Data Recommender-Systems Categorical    37711          294 1998
5 Arrhythmia Multivariate Classification Categorical, Integer, Real     452           279 1998
6 Artificial Characters Multivariate Classification Categorical, Integer, Real    6000           7 1992

```

From the preceding output, we know that the dataset has 427 observations

(dataset). For each dataset, we have 7 related features, such as `Name`, `Data_Types`, `Default_Task`, `Attribute_Types`, `N_Instances` (number of instances), `N_Attributes` (number of attributes), and `Year`. The variable called `Default_Task` could be interpreted as the basic usage of each dataset. For example, the first dataset, called `Abalone`, could be used for `classification` problems. The `unique()` function could be used to find out all possible `Default_Task`, shown here:

```
> unique(.UCIdatasets$Default_Task)
[1] Classification          Recommender-Systems
[3] Regression              Classification, Regression
[5] 
[7] Relational-Learning     Function-Learning
[9] Regression, Description Clustering
[11] Causal-Discovery       Classification, Causal-Discovery
[13] Regression, Clustering, Causal-Discovery Classification, Regression, Clustering
[15] Regression, Clustering Regression, Classification
[17] Classification, Regression, Clustering, Causa Classification, Clustering, Causal-Discovery
[19] Clustering, Causal-Discovery
```

The AppliedPredictiveModeling R package

This package includes many useful datasets that can be used for this chapter and others. The easiest way to find those datasets is with the `help()` function, shown here:

```
>library(AppliedPredictiveModeling)  
>help(package=AppliedPredictiveModeling)
```

The following table shows the datasets included in this package:

Data (name)	Names of dataset(s)	Explanation
abalone	abalone	Abalone data
bio	bio	Hepatic injury data
bookTheme	bookTheme	Lattice themes
cars2010, cars2011, and cars2012	cars2010, cars2011, cars2012	Fuel economy data
chem	chem	Hepatic injury data
ChemicalManufacturingProcess	ChemicalManufacturingProcess	Chemical

		manufactur process dat
classes	classes	Two class example da
concrete	concrete	Compressiv strength of concrete
diagnosis	diagnosis	Alzheimer's disease CS1 data
easyBoundaryFunc	easyBoundaryFunc	Functions f simulating data
fingerprints	fingerprints	Permeabilit data
getPackages	getPackages	Install packages fo each chapter
injury	injury	Hepatic inj data

logisticCreditPredictions	logisticCreditPredictions	Credit data
mixtures	mixtures	Compressive strength of concrete
permeability	permeability	Permeability data
permuteRelief	permuteRelief	Permutation statistics for the relief algorithm
predictors	predictors	Alzheimer's disease CSI data
quadBoundaryFunc	quadBoundaryFunc	Functions for simulating data
schedulingData	schedulingData	HPC job scheduling data
scriptLocation	scriptLocation	Find chapter script files

segmentationOriginal	segmentationOriginal	Cell body segmentation
solubility	solTestX, solTestXtrans, solTestY, solTrainX, solTrainXtrans, solTrainY, and trainX	Solubility data
transparentTheme	transparentTheme	Lattice themes
twoClassData	twoClassData	Two class example data

Table 10.1 datasets embedded in the R package AppliedPredictiveModeling. Here, we show you a few examples of how to load these datasets. To load one set of data, we use the `data()` function. For the first dataset, called `abalone`, we have the following code:

```
library(AppliedPredictiveModeling) > data(abalone) > dim(abalone) > head(abalone)
```

```
> dim(abalone)
[1] 4177    9
> head(abalone)
   Type LongestShell Diameter Height WholeWeight ShuckedWeight VisceraWeight ShellWeight Rings
1   M      0.455     0.365  0.095      0.5140      0.2245      0.1010     0.150    15
2   M      0.350     0.265  0.090      0.2255      0.0995      0.0485     0.070     7
3   F      0.530     0.420  0.135      0.6770      0.2565      0.1415     0.210     9
4   M      0.440     0.365  0.125      0.5160      0.2155      0.1140     0.155    10
5   I      0.330     0.255  0.080      0.2050      0.0895      0.0395     0.055     7
6   I      0.425     0.300  0.095      0.3515      0.1410      0.0775     0.120     8
```

The output is as follows:

```
> |
```

For some, the big dataset includes a few sub-datasets: > library(AppliedPredictiveModeling) > data(solubility) > ls(pattern="sol") [1] "solTestX" "solTestXtrans" "solTestY" [4] "solTrainX" "solTrainXtrans" "solTrainY"

To load each dataset, we could use the `dim()`, `head()`, `tail()`, and `summary()` functions.

```
> path<- "http://canisius.edu/~yany/RData/"  
  
> dataSet<- "usGDPannual"  
  
> con<-paste(path,dataSet,".RData",sep="") > load(url(con))  
  
> head(.usGDPannual) <strong> YEAR GDP  
1 1930 92.2  
2 1931 77.4  
3 1932 59.5  
4 1933 57.2  
5 1934 66.8  
6 1935 74.3</strong>  
> dataSet<- "usGDPquarterly"  
  
> con<-paste(path,dataSet,".RData",sep="") > load(url(con))  
  
> head(.usGDPquarterly) <strong> DATE GDP_CURRENT GDP2009DOLLAR  
1 1947Q1 243.1 1934.5  
2 1947Q2 246.3 1932.3  
3 1947Q3 250.1 1930.3  
4 1947Q4 260.3 1960.7  
5 1948Q1 266.2 1989.5  
6 1948Q2 272.9 2021.9</strong>  
  
> library(timeSeries)<br/>> data(MSFT)<br/>> x <- MSFT<br/>> by <-  
timeSequence(from = start(x), to = end(x), by = "week")<br/>> y<-  
aggregate(x,by,mean)
```

> head(x)
GMT
 Open High Low Close
Volume
2000-09-27 63.4375 63.5625 59.8125 60.6250
53077800
2000-09-28 60.8125 61.8750 60.6250 61.3125
26180200
2000-09-29 61.0000 61.3125 58.6250 60.3125
37026800
2000-10-02 60.5000 60.8125 58.2500 59.1250
29281200
2000-10-03 59.5625 59.8125 56.5000 56.5625
42687000
2000-10-04 56.3750 56.5625 54.5000 55.4375
68226700
> head(y)
GMT
 Open High Low Close Volume
2000-09-27 63.4375
63.5625 59.8125 60.6250 53077800
2000-10-04 59.6500
60.0750 57.7000 58.5500 40680380
2000-10-11 54.9750
56.4500 54.1625 55.0875 36448900
2000-10-18 53.0375
54.2500 50.8375 52.1375 50631280
2000-10-25 61.7875
64.1875 60.0875 62.3875 86457340
2000-11-01 66.1375
68.7875 65.8500 67.9375 53496000

Predicting future events

There are many techniques we could employ when trying to predict the future, such as **moving average (MA)**, regression, auto-regression, and the like. First, let's start with the simplest one for a moving average:

```
movingAverageFunction<- function(data,n=10){  
  out= data  
  for(i in n:length(data)){  
    out[i] = mean(data[(i-n+1):i])  
  }  
  return(out)  
}
```

In the preceding program, the default value for the number of periods is `10`. We could use the dataset called `MSFT` included in the R package called `timeSeries` (see the code that follows):

```
> library(timeSeries)  
> data(MSFT)  
> p<-MSFT$Close  
> #  
> ma<-movingAverageFunction(p,3)  
> head(p)  
[1] 60.6250 61.3125 60.3125 59.1250 56.5625 55.4375  
> head(ma)  
[1] 60.62500 61.31250 60.75000 60.25000 58.66667 57.04167  
> mean(p[1:3])  
[1] 60.75  
> mean(p[2:4])  
[1] 60.25
```

Manually, we find that the average of the first three values of `x` is the same as the third value of `y`. In a sense, we could use the moving average to predict the future.

In the next example, we will show you how to estimate the expected next year's market return. Here, we use the S&P500 index and the historical annual average as our expected values. First, we could go to Yahoo!Finance to download the data. The symbol for the S&P500 index is `^GSPC`:

```
library(plyr)  
year<-c(2000,2000,2001,2001,2004)  
values<-c(2, 3, 3, 5, 6)  
df <- data.frame(DATE=year,B =values )  
dfsum <- ddply(df, c("DATE"),summarize,B=sum(B))
```

The output is shown here:

```
> df
  DATE B
1 2000 2
2 2000 3
3 2001 3
4 2001 5
5 2004 6
> dfsum
  DATE B
1 2000 5
2 2001 8
3 2004 6
```

Now, we download the S&P500 index's historical monthly price data from Yahoo!Finance. Alternatively, we could use the following code to download the R dataset. The first several commands are used to download the related dataset called `.sp500monthly`. The objective of the program is to estimate annual mean and its 90% confidence range:

```
> library(data.table)
> path<- 'http://canisius.edu/~yany/RData/'
> dataSet<- 'sp500monthly.RData'
> link<-paste(path,dataSet,sep=' ')
> load(url(link))
> #head(.sp500monthly,2)
> p<- .sp500monthly$ADJ.CLOSE
> n<-length(p)
> logRet<-log(p[2:n]/p[1:(n-1)])
> years<-format(.sp500monthly$DATE[2:n], "%Y")
> y<-data.frame(.sp500monthly$DATE[2:n],years,logRet)
> colnames(y)<-c("DATE","YEAR","LOGRET")
> y2<- data.table(y)
> z<-y2[,sum(LOGRET),by=YEAR]
> z2<-na.omit(z)
> annualRet<-data.frame(z2$YEAR,exp(z2[,2])-1)
> n<-nrow(annualRet)
> std<-sd(annualRet[,2])
> stdErr<-std/sqrt(n)
> ourMean<-mean(annualRet[,2])
> min2<-ourMean-2*stdErr
> max2<-ourMean+2*stdErr
> cat("[min mean max ]\n")
[min mean max ]
> cat(min2,ourMean,max2," \n")
0.05032956 0.09022369 0.1301178
```

From the result, the historical mean annual returns for S&P500 is 9%. If we claimed that next year's index return would be 9% as well, it could be between 5% and 13%, a potential huge swing.

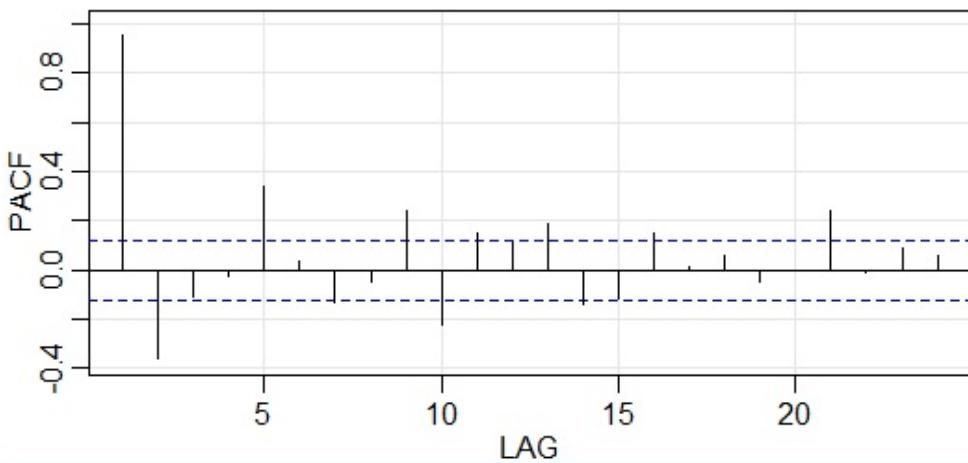
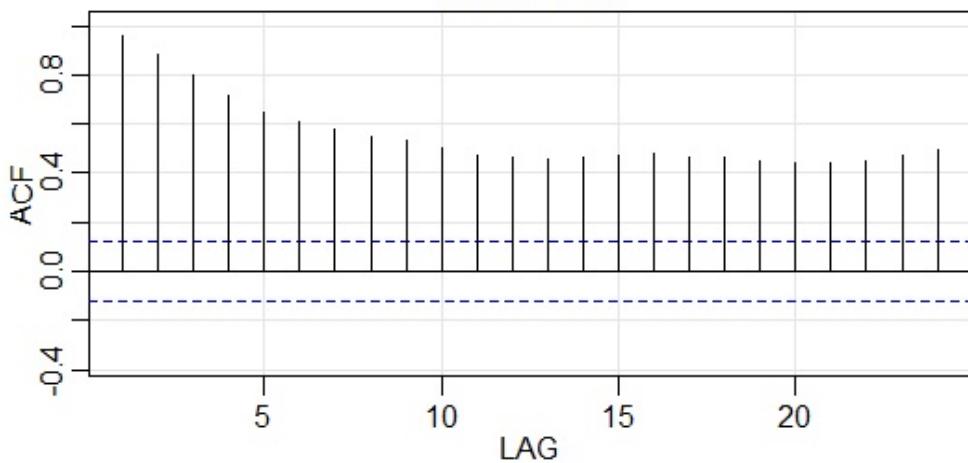
Seasonality

In the next example, we will show you auto-correlation. First, we upload an R package called **astsa**, which stands for **Applied Statistical Time Series Analysis**. Then, we upload the US GDP with quarterly frequency:

```
library(astsa)
path<- "http://canisius.edu/~yany/RData/"
dataSet<- "usGDPquarterly"
con<- paste(path,dataSet,".RData",sep=' ')
load(url(con))
x<- .usGDPquarterly$DATE
y<- .usGDPquarterly$GDP_CURRENT
plot(x,y)
diff4 = diff(y,4)
acf2(diff4,24)
```

In the preceding program, the `diff()` function takes a difference, such as the current value minus the previous value. The second input value indicates the lag. The function called `acf2()` is used to plot and print the ACF and PACF of a time series. ACF stands for **auto-covariance function**, while PACF stands for the **partial auto-correlation function**. The related graphs are shown here:

Series: diff4

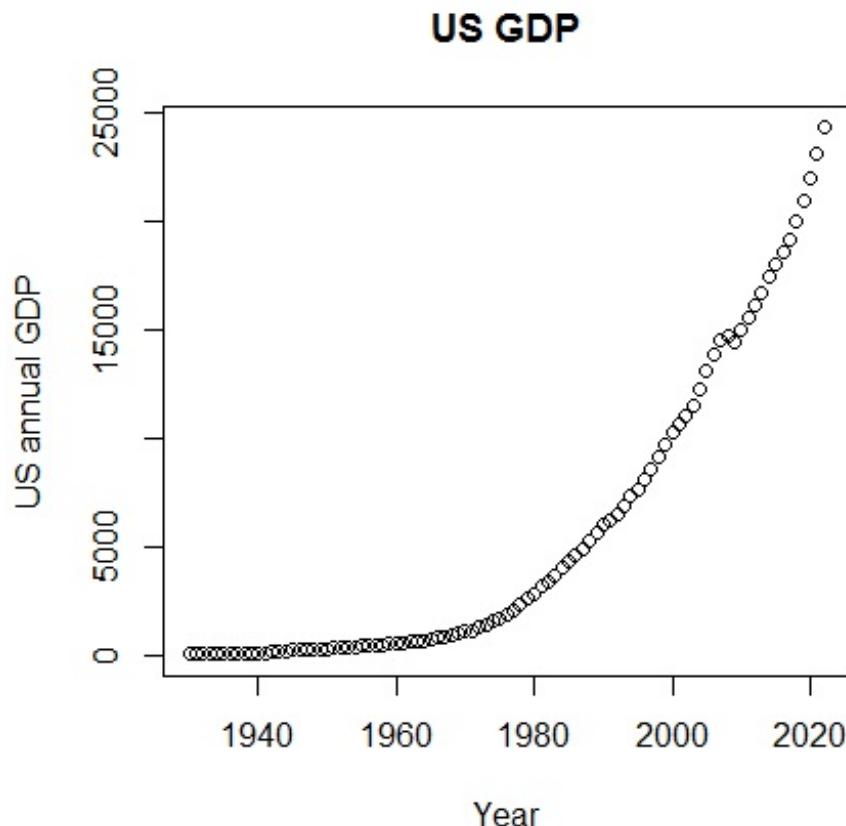


Visualizing components

It is understandable that the concepts and datasets would be much more clear if we could use graphs. The first example shows the fluctuation of the US GDP over the last five decades:

```
> path<-http://canisius.edu/~yany/RData/"  
> dataSet<-"usGDPannual"  
> con<-paste(path,dataSet,".RData",sep=".")  
> load(url(con))  
> title<-"US GDP"  
> xTitle<-"Year"  
> yTitle<-"US annual GDP"  
> x<-.usGDPannual$YEAR  
> y<-.usGDPannual$GDP  
> plot(x,y,main=title,xlab=xTitle,ylab=yTitle)
```

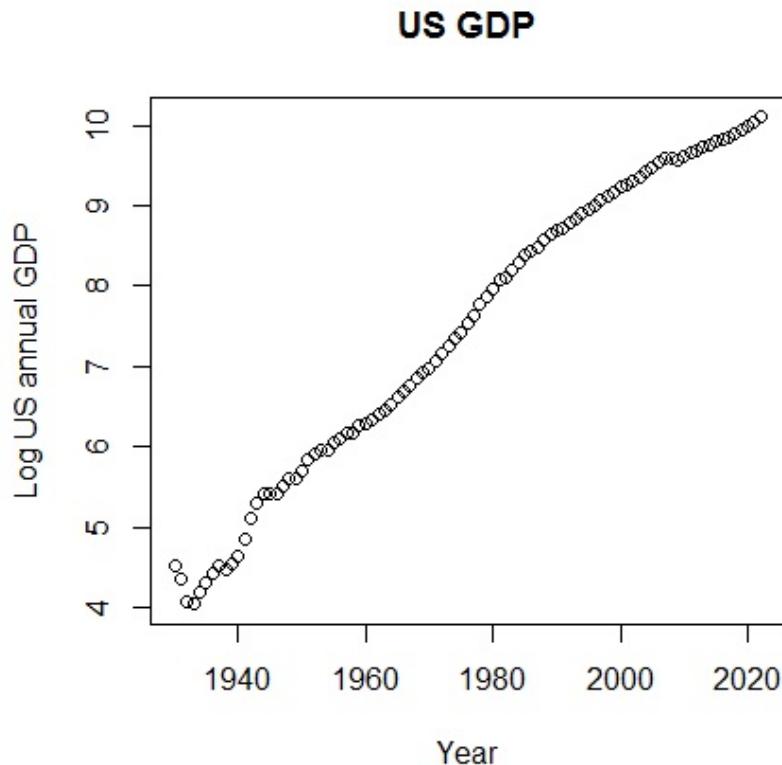
The related graph is shown here:



If we used log scale for the GDP, we would have the following code and graph:

```
> yTitle<- "Log US annual GDP"  
> plot(x,log(y),main=title,xlab=xTitle,ylab=yTitle)
```

The following graph is close to a straight line:



```

library(LiblineaR)<br/>data(iris)<br/>attach(iris)<br/>x=iris[,1:4]
<br/>y=factor(iris[,5])<br/>train=sample(1:dim(iris)[1],100)
<br/>xTrain=x[train,];xTest=x[-train,]<br/>yTrain=y[train]; yTest=y[-train]
<br/>s=scale(xTrain,center=TRUE,scale=TRUE)<br/>#<br/>tryTypes=c(0:7)
<br/>tryCosts=c(1000,1,0.001)
<br/>bestCost=NA<br/>bestAcc=0<br/>bestType=NA<br/>#<br/>for(ty in
tryTypes){<br/> for(co in tryCosts){<br/>
acc=LiblineaR(data=s,target=yTrain,type=ty,cost=co,bias=1,cross=5,verbose=FA
<br/> cat("Results for C=",co,": ",acc," accuracy.\n",sep="")<br/>
if(acc>bestAcc){<br/> bestCost=co<br/> bestAcc=acc<br/> bestType=ty<br/> }
<br/> }<br/>}<br/>cat("Best model type is:",bestType,"\n")<br/>cat("Best cost
is:",bestCost,"\n")<br/>cat("Best accuracy is:",bestAcc,"\n")<br/># Re-train best
model with best cost value.
<br/>m=LiblineaR(data=s,target=yTrain,type=bestType,cost=bestCost,bias=1,vei
<br/># Scale the test
data<br/>s2=scale(xTest,attr(s,"scaled:center"),attr(s,"scaled:scale"))
<br/>pr=FALSE; # Make prediction<br/>if(bestType==0 || bestType==7)
pr=TRUE<br/>p=predict(m,s2,proba=pr,decisionValues=TRUE)
<br/>res=table(p$predictions,yTest) # Display confusion matrix<br/>print(res)
<br/># Compute Balanced Classification
Rate<br/>BCR=mean(c(res[1,1]/sum(res[,1]),res[2,2]/sum(res[,2]),res[3,3]/sum(i
<br/>print(BCR)

> cat("Best model type is:",bestType,"\n")<br/><strong>Best model type is:
4</strong> <br/>> cat("Best cost is:",bestCost,"\n")<br/><strong>Best cost is:
1</strong> <br/>> cat("Best accuracy is:",bestAcc,"\n")<br/><strong>Best
accuracy is: 0.98</strong> <br/>> print(res)<br/> <strong> yTest</strong><br/>
<strong> setosa versicolor virginica</strong><br/><strong> setosa 16 0
0</strong><br/><strong> versicolor 0 17 0</strong><br/><strong> virginica 0 3
14</strong><br/>> print(BCR)<br/><strong>[1] 0.95</strong>
```

R package – datarobot

The name DataRobot refers to three things: a Boston-based software company, the massively parallel modelling engine developed by the DataRobot company, and an open source R package that allows interactive R users to connect to this modelling engine. This vignette provides a brief introduction to the `datarobot` R package, highlighting the following key details of its use:

- Connecting to the DataRobot modeling engine from an interactive R session
- Creating a new modeling project in the DataRobot modeling engine
- Retrieving the results from a DataRobot modeling project
- Generating predictions from any DataRobot model

To launch the package, we use the `library()` function:

```
|> library(datarobot)
```

After the previous code, there is a good chance a new user will get an error message, something like the following:

```
|> library(datarobot)
Did not connect to DataRobot on package startup. Use `ConnectToDataRobot`.
To connect by default on startup, you can put a config file at: C:\Users\yany\Documents\
```

This means that we have to register with the company to get a verified token key. The final one will have the following format:

```
| loc<- "YOUR-ENDPOINT-HERE"
| myToken<- "YOUR-API_TOKEN-HERE"
| ConnectToDataRobot(endpoint=loc, token=myToken)
```

We could also use the `help()` function to find users of the package:

```
|> help(package=datarobot)
```

R package – eclust

This package is the **Environment-Based Clustering for Interpretable Predictive Models in High Dimensional Data**. First, let's look at a dataset called `simdata`, which contains simulated data for the package:

```
> library(eclust)
> data("simdata")
> dim(simdata)
[1] 100 502
> simdata[1:5, 1:6]
      Y E Gene1 Gene2 Gene3 Gene4
[1,] -94.131497 0 -0.4821629 0.1298527 0.4228393 0.36643188
[2,] 7.134990 0 -1.5216289 -0.3304428 -0.4384459 1.57602830
[3,] 1.974194 0 0.7590055 -0.3600983 1.9006443 -1.47250061
[4,] -44.855010 0 0.6833635 1.8051352 0.1527713 -0.06442029
[5,] 23.547378 0 0.4587626 -0.3996984 -0.5727255 -1.75716775
> table(simdata[, "E"])
  0 1
50 50
>
```

The preceding output shows that the dimension of the data is 100 by 502. `Y` is a continuous response vector, and `E` is a binary environment variable for the ECLUST method. $E = 0$ for unexposed ($n=50$) and $E = 1$ for exposed ($n=50$). The following R program estimates the Fisher z-transformation of correlations. The definition of Fisher's Z transformation is shown in the following code:

```
> library(eclust)
> data("simdata")
> X = simdata[,c(-1,-2)]
> firstCorr<-cor(X[1:50,])
> secondCorr<-cor(X[51:100,])
> score<-u_fisherZ(n0=100,cor0=firstCorr,n1=100,cor1=secondCorr)
> dim(score)
[1] 500 500
> score[1:5,1:5]
      Gene1 Gene2 Gene3 Gene4 Gene5
Gene1 1.000000 -8.062020 6.260050 -8.133437 -7.825391
Gene2 -8.062020 1.000000 9.162208 -7.431822 -7.814067
Gene3 6.260050 9.162208 1.000000 8.072412 6.529433
Gene4 -8.133437 -7.431822 8.072412 1.000000 -5.099261
Gene5 -7.825391 -7.814067 6.529433 -5.099261 1.000000
>
```

The Fisher's Z-transformation is defined here. Assuming that we have a set of n pairs of x_i and y_i , we could estimate their correlation by applying the following formula:

$$p = \frac{cov(x, y)}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (1)$$

Here, ρ is the correlation between two variables, while \bar{x} and \bar{y} are the means of x and y . The Fisher's z value is defined as :

$$Z = \frac{1}{2} \ln\left(\frac{1+p}{1-p}\right) = \text{arctanh}(p) \quad (2)$$

In is the natural logarithm function and `arctanh()` is the inverse hyperbolic tangent function.

Model selection

When finding a good model, sometimes we face under fitting and over fitting. The first example is borrowed; you can download the program at http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html#sphx-glr-auto-examples-model-selection-plot-underfitting-overfitting-py. It demonstrates the problems of under fitting and over fitting and how we can use linear regression with polynomial features to approximate nonlinear functions. The true function is given here:

$$f(x) = \cos(1.5\pi x) \quad (3)$$

In the following program, we try to use linear and polynomial models to approximate the equation. The slightly modified code is shown here. The program tries to show the impact of different models in terms of under-fitting and over-fitting:

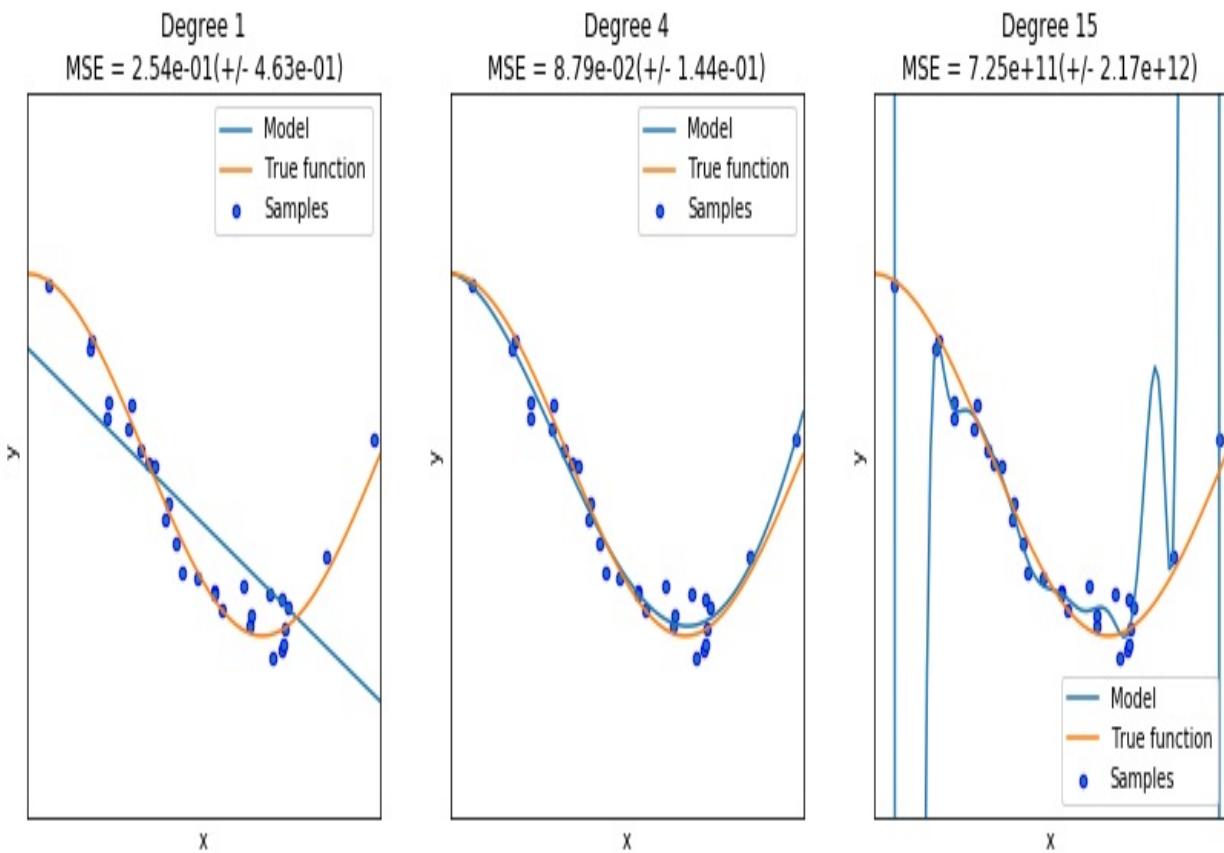
```
import sklearn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
#
np.random.seed(123)
n= 30 # number of samples
degrees = [1, 4, 15]
def true_fun(x):
    return np.cos(1.5*np.pi*x)
x = np.sort(np.random.rand(n))
y = true_fun(x) + np.random.randn(n) * 0.1
plt.figure(figsize=(14, 5))
title="Degree {}\\nMSE = {:.2e}(+/- {:.2e})"
name1="polynomial_features"
name2="linear_regression"
name3="neg_mean_squared_error"
#
for i in range(len(degrees)):
    ax=plt.subplot(1,len(degrees),i+1)
    plt.setp(ax, xticks=(), yticks=())
    pFeatures=PolynomialFeatures(degree=degrees[i], include_bias=False)
    linear_regression = LinearRegression()
    pipeline=Pipeline([(name1,pFeatures),(name2,linear_regression)])
    pipeline.fit(x[:,np.newaxis],y)
    scores=cross_val_score(pipeline,x[:,np.newaxis],y,scoring=name3,cv=10)
    xTest = np.linspace(0, 1, 100)
    plt.plot(xTest,pipeline.predict(xTest[:,np.newaxis]),label="Model")
    plt.plot(xTest,true_fun(xTest),label="True function")
```

```

plt.scatter(x,y,edgecolor='b',s=20,label="Samples")
plt.xlabel("x")
plt.ylabel("y")
plt.xlim((0,1))
plt.ylim((-2,2))
plt.legend(loc="best")
plt.title(title.format(degrees[i], -scores.mean(), scores.std()))
plt.show()

```

The related graphs are shown here:



Note that, on the top of each graph, **MSE** stands for the **Mean Squared Error**. For the left-hand graph, the program tries to use a line to fit the true model based on the input dataset. Since it is linear, the **Degree** is **1**. Compared with the second model with a **Degree** of **4**, the **MSE** of this learned model is larger, 0.54 versus 0.0879. This indicates that the linear model might under-fit the model while the second model might over-fit the model.

```
import datetime<br/>import pandas<br/>from sqlalchemy import  
create_engine<br/>from metta import metta_io as metta<br/>from  
catwalk.storage import FSModelStorageEngine, CSVMatrixStore<br/>from  
catwalk.model_trainers import ModelTrainer<br/>from catwalk.predictors  
import Predictor<br/>from catwalk.evaluation import ModelEvaluator<br/>from  
catwalk.utils import  
save_experiment_and_get_hash<br/>help(FSModelStorageEngine)
```

Help on class FSModelStorageEngine in module catwalk.storage:

class FSModelStorageEngine(ModelStorageEngine)
 | Method
resolution order:
 | FSModelStorageEngine
 |
ModelStorageEngine
 | builtins.object
 |
 | Methods defined here:
 |
 | __init__(self, *args, **kwargs)
 | Initialize self. See
help(type(self)) for accurate signature.
 |
 | get_store(self, model_hash)
 |
 | -----

| Data descriptors inherited from ModelStorageEngine:
 |
 |
__dict__
 | dictionary for instance variables (if defined)
 |
 |
__weakref__
 | list of weak references to the object (if defined)

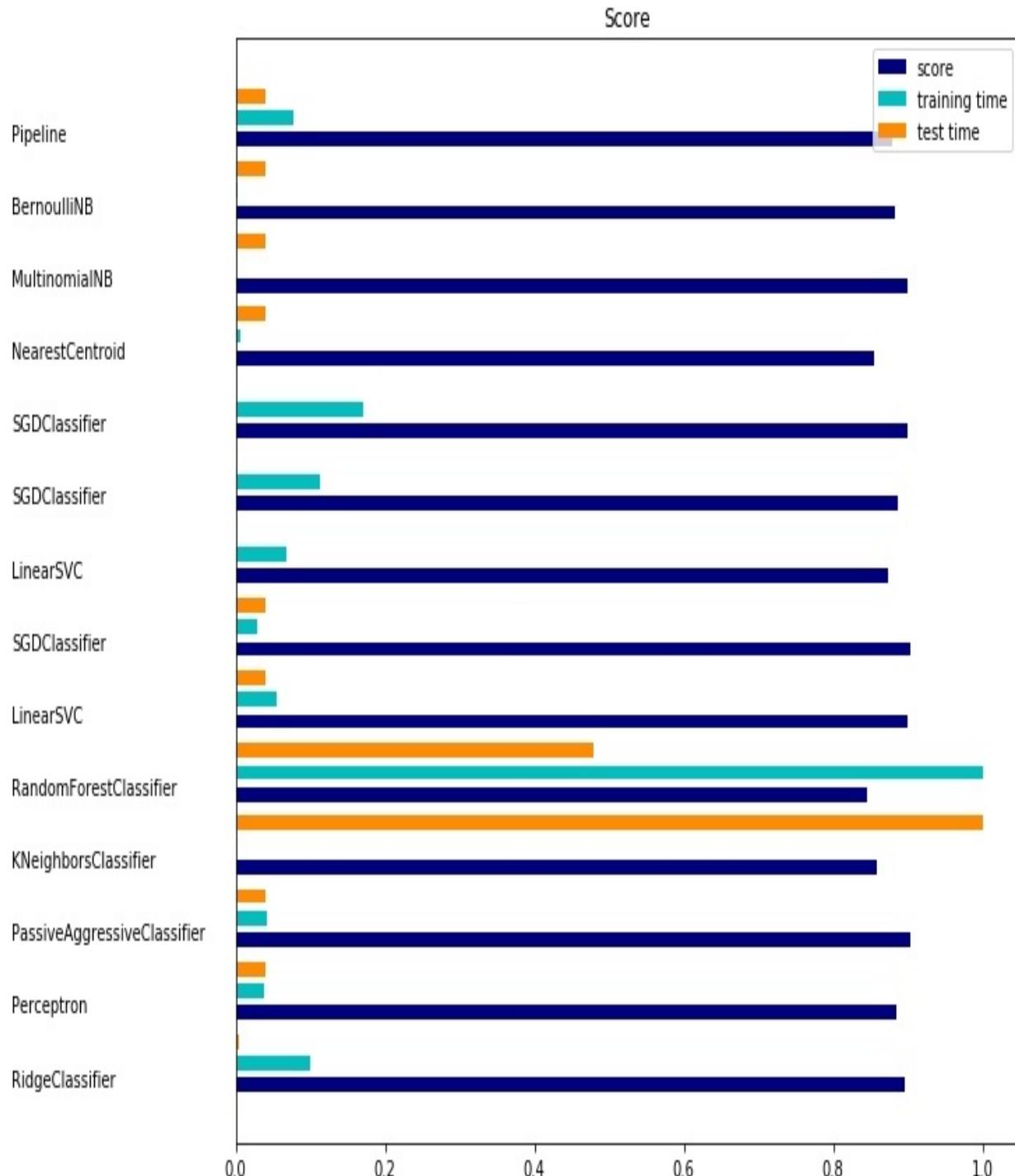
Python package – sklearn

Since Python `sklearn` is a very useful package, it is worthwhile to show you more examples of using this package. The example cited here is how to use the package to classify documents by topics using a bag-of-words approach.

This example uses a `scipy.sparse` matrix to store the features and demonstrates various classifiers that can efficiently handle sparse matrices. The dataset used in this example is the 20 newsgroups dataset. It will be automatically downloaded, then cached. The ZIP file contains the input files and can be downloaded at <http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz>. It has a size of about 14 MB. The code is available at the following web link: http://scikit-learn.org/stable/auto_examples/text/document_classification_20newsgroups.html#sphx-glr-auto-examples-text-document-classification-20newsgroups-py. To save space, only the first several lines are shown here:

```
import logging
import numpy as np
from optparse import OptionParser
import sys
from time import time
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.feature_selection import SelectFromModel
```

The related output is shown here:



For each method, there are three measures: **score**, **training time**, and **testing time**. For example, for the `RandomForestClassifier` method, it uses lots of time for training and testing; see the longest three bars. It is understandable since this method uses lots of simulations.

Julia package – QuantEcon

For predicting a future possible event, one of the tools is the Monte Carlo simulation. For this purpose, we could use a Julia package called `QuantEcon`. This package is **Quantitative Economics with Julia**. The first example is the Markov simulation: using `QuantEcon`

```
P = [0.4 0.6; 0.2 0.8];
mc = MarkovChain(P)
x = simulate(mc, 100000);
mean(x .== 1)
#
mc2 = MarkovChain(P, ["employed", "unemployed"])
simulate(mc2, 4)
```

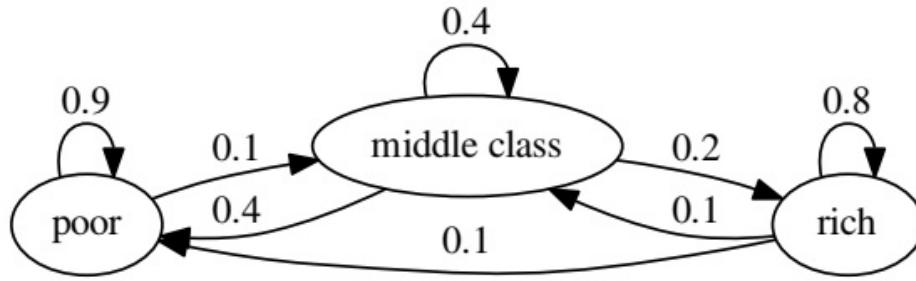
The first part of the program simulates 100,000 times of P matrix while the second part simulates two statuses: `employed` and `unemployed`. See the following output:

```
julia> mean(x .== 1)
0.24978

julia> #
julia> mc2 = MarkovChain(P, ["employed", "unemployed"])
Discrete Markov Chain
stochastic matrix of type Array{Float64,2}:
[0.4 0.6; 0.2 0.8]

julia> simulate(mc2, 4)
4-element Array{String,1}:
 "unemployed"
 "unemployed"
 "unemployed"
 "employed"
```

The next example is also borrowed from the manual. The objective is to see how a person from one economic status transforms to another one in the future. First, let's see the following graph:



Let's look at the leftmost oval with **poor** inside. It means for a **poor** person he/she has 90% chance to remain **poor** and 10% moves to the **middle class**. It could be represented by the following matrix, putting zeros where there's no edge between nodes:

$$p = \begin{pmatrix} 0.9 & 0.1 & 0 \\ 0.4 & 0.4 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{pmatrix} \quad (4)$$

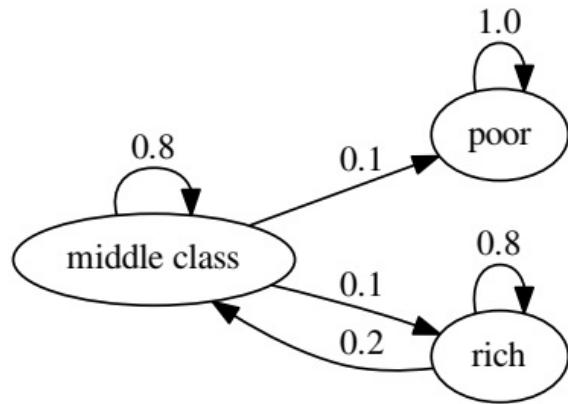
Two states, x and y , are said to communicate with each other if there exists positive integers j and k , such as:

$$\begin{aligned} p^j(x, y) &> 0 \\ p^k(y, x) &> 0 \end{aligned}$$

The stochastic matrix P is called irreducible if all states communicate; that is, if x and y communicate for every (x, y) . It's clear from the graph that this stochastic matrix is irreducible: we can reach any state from any other state eventually. The following code would confirm this: using QuantEcon

```
P = [0.9 0.1 0.0; 0.4 0.4 0.2; 0.1 0.1 0.8];
mc = MarkovChain(P)
is_irreducible(mc)
```

The following graph would represent an extreme case for irreducible since the future status for a **poor** person will be 100% **poor**:



The following code would confirm this as well, as the result will be `false`:

```

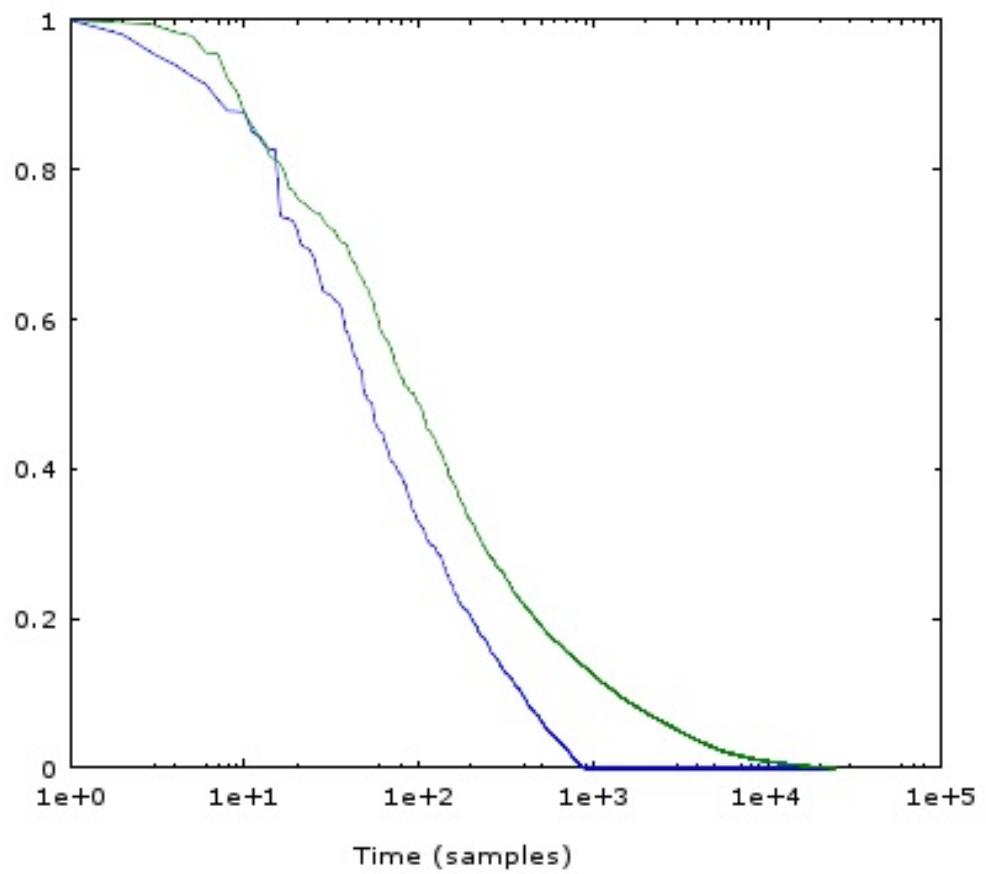
using QuantEcon
P2 = [1.0 0.0 0.0; 0.1 0.8 0.1; 0.0 0.2 0.8];
mc2 = MarkovChain(P2)
is_irreducible(mc2)
  
```

Octave package – ltfat

The `ltfat` package stands for **Large Time/Frequency Analysis Toolbox** and is a Matlab/Octave toolbox for working with time frequency analysis, wavelets, and signal processing. It is intended both as an educational and a computational tool. The toolbox provides a large number of linear transforms including Gabor and wavelet transforms, along with routines for constructing windows (filter prototypes) and manipulating coefficients. The following example is borrowed from their manual, which is available at <http://ltfat.github.io/doc/ltfat.pdf>. The following example shows how the function `franalasso()` produces a sparse representation of a test signal `greasy`:

```
pkg load ltfat
f = greasy;
name1="sparsified coefficients"
name2="dual system coefficients"
F = frame('dgtreal','gauss',64,512);
lambda = 0.1;
% Solve the basis pursuit problem
[c,~,~,frec,cd] = franalasso(F,f,lambda);
figure(1); % Plot sparse coefficients
plotframe(F,c,'dynrange',50);
figure(2); % Plot coefficients
plotframe(F,cd,'dynrange',50);
norm(f-frec)
figure(3);
semilogx([sort(abs(c),'descend')/max(abs(c)),...
sort(abs(cd),'descend')/max(abs(cd))]);
legend({name1,name2});
```

The program outputs three graphs, and the last one is shown here:



Granger causality test

When saying that A causes B, this means that A is the reason that B happens. This is the common definition of causality: which one causes the next one. The Granger causality test is used to determine whether one time series is a factor and offers useful information in forecasting the second one. In the following code, a dataset called `ChickEgg` is used as an illustration. The dataset has two columns, number of chicks and number of eggs, with a timestamp:

```
> library(lmtest)
> data(ChickEgg)
> dim(ChickEgg)
[1] 54 2
> ChickEgg[1:5, ]
  chicken egg
[1,] 468491 3581
[2,] 449743 3532
[3,] 436815 3327
[4,] 444523 3255
[5,] 433937 3156
```

The question is: could we use this year's egg numbers to predict the next year's chicken numbers? If this is true, our statement will be *the number of chicks Granger causes the number of eggs*. If this is not true, we say *the number of chicks does not Granger cause the number of eggs*. Here is the related code:

```
> library(lmtest)
> data(ChickEgg)
> grangertest(chicken~egg, order = 3, data = ChickEgg)
Granger causality test
Model 1: chicken ~ Lags(chicken, 1:3) + Lags(egg, 1:3)
Model 2: chicken ~ Lags(chicken, 1:3)
  Res.Df Df F Pr(>F)
1 44
2 47 -3 5.405 0.002966 **
---
Signif. codes: 0 '****' 0.001 '***' 0.01 '**' 0.05 '.' 0.1 ' ' 1
```

In model 1, we try to use the lags of chicken plus the lags of the lags of egg to explain the chicken numbers. Since the P-value is quite small, it is significant at 0.01. Because of this, we say that *egg Granger causes chicken*. The following test shows that chicken could not be used to forecast the next period's error:

```
> grangertest(egg~chicken, order = 3, data = ChickEgg)
Granger causality test
```

```

Model 1: egg ~ Lags(egg, 1:3) + Lags(chicken, 1:3)
Model 2: egg ~ Lags(egg, 1:3)
  Res.Df Df F Pr(>F)
1 44
2 47 -3 0.5916 0.6238

```

In the next example, we test which one (IBM's return or S&P500 return) could Granger cause the other. First, we define a return function:

```

ret_f<-function(x,ticker=""){
  n<-nrow(x)
  p<-x[,6]
  ret<-p[2:n]/p[1:(n-1)]-1
  output<-data.frame(x[2:n,1],ret)
  name<-paste("RET_",toupper(ticker),sep=' ')
  colnames(output)<-c("DATE",name)
  return(output)
}

```

Note that we use the return plus the ticker as the column name:

```

> x<-read.csv("http://canisius.edu/~yany/data/ibmDaily.csv", header=T)
> ibmRet<-ret_f(x,"ibm")
> x<-read.csv("http://canisius.edu/~yany/data/^gspcDaily.csv", header=T)
> mktRet<-ret_f(x,"mkt")
> final<-merge(ibmRet,mktRet)
> head(final)
  DATE RET_IBM RET_MKT
1 1962-01-03 0.008742545 0.0023956877
2 1962-01-04 -0.009965497 -0.0068887673
3 1962-01-05 -0.019694350 -0.0138730891
4 1962-01-08 -0.018750380 -0.0077519519
5 1962-01-09 0.011829467 0.0004340133
6 1962-01-10 0.001798526 -0.0027476933

```

Now, the function could be called with the input values. The objective of the program is to test if we could use the lagged market returns to explain IBM's returns. Similarly, we test whether the lagged IBM's returns explain the market's returns:

```

> library(lmtest)
> grangertest(RET_IBM ~ RET_MKT, order = 1, data =final)
Granger causality test
Model 1: RET_IBM ~ Lags(RET_IBM, 1:1) + Lags(RET_MKT, 1:1)
Model 2: RET_IBM ~ Lags(RET_IBM, 1:1)
  Res.Df Df F Pr(>F)
1 14149
2 14150 -1 24.002 9.729e-07 ***
---
Signif. codes: 0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1

```

The preceding results suggest that the S&P500 index could be used to explain IBM's next period's return, since it is statistically significant at 0.1%. The next

line will test if the IBM lags of returns would explain S&P500 index returns:

```
> grangertest(RET_MKT ~ RET_IBM, order = 1, data =final)
Granger causality test
Model 1: RET_MKT ~ Lags(RET_MKT, 1:1) + Lags(RET_IBM, 1:1)
Model 2: RET_MKT ~ Lags(RET_MKT, 1:1)
  Res.Df Df F Pr(>F)
1 14149
2 14150 -1 7.5378 0.006049 **
---
Signif. codes: 0 '****' 0.001 '***' 0.01 '**' 0.05 '.' 0.1 ' ' 1
>
```

The result suggests that, over this period, IMB's returns could be used to explain the S&P500 index in the next period's returns as well.

Summary

In this chapter, we have discussed predictive data analytics, modeling and validation, some useful datasets, time series analytics, how to predict future events, seasonality, and how to visualize our data. For Python packages, we have mentioned `prsklearn` and `catwalk`. For R packages, we have discussed `datarobot`, `LiblineaR`, and `eclust`. For Julia packages, we explained `EQuantEcon`. For Octave, we have explained `1tfat`.

In the next chapter, we will discuss Anaconda Cloud. Some topics include the Jupyter Notebook in depth, different formats of the Jupyter Notebooks, how to share notebooks with your partners, how to share different projects over different platforms, how to share your working environments, and how to replicate others' environments locally.

Review questions and exercises

1. Why do we care about predicting the future?
2. What does *seasonality* mean? How could it impact our predictions?
3. How does one measure the impact of seasonality?
4. Write an R program to use the moving average of the last five years to predict the next year's expected return. The source of the data is <http://financ.e.yahoo.com>. You can test a few stocks such as IBM, C, and WMT. In addition, apply the same method to the S&P500 index. What is your conclusion?
5. Assume that we have the following true model:

$$f(x) = \text{sinc}(2.5\pi + 3x)$$

Write a Python program to use linear and polynomial models to approximate the previous function and show the related graphs.

6. Download a market index monthly data and estimate its next year's annual return. The S&P500 could be used as the index and Yahoo!Finance at financ.e.yahoo.com could be used as the source of data. Source of data: <https://finance.yahoo.com>
7. Download several countries' index data, such as the US, Canada, Hong Kong, and China. Estimate the returns. Conduct the Granger causality test to see which country's stock market index is a dominant force.
8. From Professor French's Data Library, download the Fama-French three-factor time series. Estimate the next year's **Small Minus Big (SMB)** factor and **High Minus Low (HML)** factor. The web page is http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html. Alternatively, a Python dataset can be downloaded at <http://canisius.edu/~yany/python/data/ffMonthly.pkl>. A few lines of Python code are shown here:

```
| import pandas as pd  
| ff=pd.read_pickle("c:/temp/ffMonthly.pkl")  
| print(ff.head())
```

The related output is shown here:

```

print(ff.head())
      MKT_RF SMB HML RF
1926-07-01 0.0296 -0.0230 -0.0287 0.0022
1926-08-01 0.0264 -0.0140 0.0419 0.0025
1926-09-01 0.0036 -0.0132 0.0001 0.0023
1926-10-01 -0.0324 0.0004 0.0051 0.0032
1926-11-01 0.0253 -0.0020 -0.0035 0.003

```

9. What is IBM's next expected annual return?
10. What is the next three years' S&P500 index level?
11. Could we add the business cycle to our regression models? The dataset can be downloaded by using the following R code:

```

> path<- "http://canisius.edu/~yany/RData/"
> dataSet<- "businesscycle"
> link<-paste(path,dataSet,".RData",sep=' ')
> load(url(link))
> head(.businessCycle)
      DATE CYCLE
1 19261001 1.000
2 19261101 0.846
3 19261201 0.692
4 19270101 0.538
5 19270201 0.385
6 19270301 0.231

```

A peak is defined as 1 while a trough is defined as -1. Any months between a peak or trough or between a trough and peak are linearly interpolated.

Note that the following four questions are based on Octave machine learning examples. The dataset and Octave sample programs can be downloaded at <https://github.com/partharamanujam/octave-ml>.

12. Run the sample Octave machine learning program called `linear_gd.m`, the dataset containing historical records on the change in the water level.
13. Run the sample Octave machine learning program called `svm.m`. The program reads `spam-vocab` list into a struct `words` which occur at least 100 times in the spam corpus. Three input files are `spam-vocab.txt`, `spamTrain.mat`, and `spamTest.mat`.
14. Run the sample Octave machine learning program called `logistic_gd.m`.
15. Run the sample Octave machine learning program called `pca.m`.
16. Install a Julia package called `EmpiricalRisks` and show a few examples by using this package.

Anaconda Cloud

Nowadays, one of the big buzzwords in computing is *cloud*. The simplest example is that for many users, Google Drive is a kind of cloud where users can save their files, photos, and programs and share them with their colleagues, family members, or others. Another example is so-called cloud computing. According to the National Institute of Standards and Technology, cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (for example, networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. In Old English, *mass of rock, hill*, was related to cloud. The *metaphoric extension to raincloud, mass of evaporated water in the sky*, was attested by c. 1200, based on the similarity of cumulus clouds and rock masses. The usual *Old English* word for cloud was *weolcan*. In Middle English, *skie* also originally meant cloud. In this chapter, the following topics will be covered:

- Introduction to Anaconda Cloud
- Jupyter Notebook in depth
- Replicating others' environments locally
- Downloading a package from Anaconda

Introduction to Anaconda Cloud

Nowadays, we have used or at least heard of the word cloud. Usually, it means that we can save our data on other people's servers. For Anaconda Cloud, users can use the platform to save and share packages, notebooks, projects, and environments. The process, platform, and the methodology make it much easier for package management and collaboration, including sharing of their packages and projects. The public projects and notebooks are free. At the moment, private plans start at \$7 per month. Anaconda Cloud allows users to create or distribute software packages.

For a Windows version of Anaconda, click All Programs | Anaconda, and then choose Anaconda Cloud.

After double-clicking on Cloud, the welcome screen will appear.

Based on the information presented by the welcome screen, we know that we need an account with Anaconda before we can use it. After login, we will see the following screen:

0 Packages [View all \(0\)](#)

Get more information on how to [upload a Package](#).

0 Notebooks [View all \(0\)](#)

Get more information on how to [upload a Notebook](#).

0 Environments [View all \(0\)](#)

Get more information on how to [upload an Environment](#).

0 Projects [View all \(0\)](#)

No projects yet, [upload one here](#).

★ Favorites [View all \(0\)](#)

Favorite some packages, notebooks, and environments to get started!

i Activity Feed [View more](#)

 **Welcome to Anaconda Cloud!** 6 months and 5 days ago

Anaconda Cloud allows you to create or distribute software packages.

Getting started: [Installing your first package](#)

Getting started: [Distributing your first package](#)

For example, if you double-click on [Installing your first package](#), you will get more information on **Anaconda Cloud**. We do not need to be logged in, or even need a cloud account, to search for public packages, download, and install them. We need an account only to access private packages without a token or to share your packages with others.

Jupyter Notebook in depth

In this section, many examples are borrowed from the following web page: <https://github.com/ipython/ipython-in-depth>. If you are interested, you can explore more by going to the web page in order to download a ZIP file. The following screenshot shows the content of examples:

Name	Date modified	Type
binder	5/3/2018 11:10 AM	File folder
examples	5/3/2018 11:10 AM	File folder
exercises	5/3/2018 11:10 AM	File folder
tools	5/3/2018 11:10 AM	File folder
.gitignore	2/28/2018 4:30 AM	GITIGNORE File
fabfile.py	2/28/2018 4:30 AM	PY File
Index.ipynb	2/28/2018 4:30 AM	IPython Notebook
LICENSE	2/28/2018 4:30 AM	File
monitor.sh	2/28/2018 4:30 AM	SH File
pycon-2015-abstract.md	2/28/2018 4:30 AM	MD File
pycon-submission.md	2/28/2018 4:30 AM	MD File
README.md	2/28/2018 4:30 AM	MD File

After launching Jupyter Notebook, we can search the `example` subdirectory. For example, we can upload a notebook called `factoring.ipynb` under the `interactive widgets` subdirectory(see the following screenshot):

Factoring Polynomials with SymPy

Here is an example that uses [SymPy](#) to factor polynomials.

```
In [ ]: 1 from ipywidgets import interact  
        2 from IPython.display import display  
  
In [ ]: 1 from sympy import Symbol, Eq, factor, init_printing  
        2 init_printing(use_latex='mathjax')  
  
In [ ]: 1 x = Symbol('x')  
  
In [ ]: 1 def factorit(n):  
        2     display(Eq(x**n-1, factor(x**n-1)))
```

Notice how the output of the `factorit` function is properly formatted LaTeX.

```
In [ ]: 1 factorit(12)  
  
In [ ]: 1 interact(factorit, n=(2,40));
```

After clicking Run, we can change the value of `n` (see the following result when we choose `8` for the variable):

```
In [6]: 1 interact(factorit, n=(2,40));
```


$$x^8 - 1 = (x - 1)(x + 1)(x^2 + 1)(x^4 + 1)$$

After increasing `n` to `20`, we have the corresponding output:

```
In [6]: 1 interact(factorit, n=(2,40));
```

n  20

$$x^{20} - 1 = (x - 1)(x + 1)(x^2 + 1)(x^4 - x^3 + x^2 - x + 1)(x^4 + x^3 + x^2 + x + 1)(x^8 - x^6 + x^4 - x^2 + 1)$$

Sometimes, after running a Jupyter Notebook and logging out, we need a token or password to login again. We can run the following code to locate our token:

```
| Jupyter notebook list
```

Alternatively, we can run the following code to save our token to a text file:

```
| Jupyter notebook list > t.txt
```

The next example shows how to activate Jupyter QtConsole, that is, how to connect to an existing IPython kernel using the QtConsole. Again, the Jupyter Notebook is also included in the preceding download. First, we upload the notebook called `Connecting with the Qt Console.ipynb`. To save space, only the first several lines are shown here:

```
In [1]: 1 %connect_info
```

```
{  
    "shell_port": 55225,  
    "iopub_port": 55226,  
    "stdin_port": 55227,  
    "control_port": 55228,  
    "hb_port": 55229,  
    "ip": "127.0.0.1",  
    "key": "5a2f3863-38c1ceb64da3b92b9077cc67",  
    "transport": "tcp",  
    "signature_scheme": "hmac-sha256",  
    "kernel_name": ""  
}
```

After the program runs without any issues, we will see the following window:

Jupyter QtConsole

File Edit View Kernel Window Help

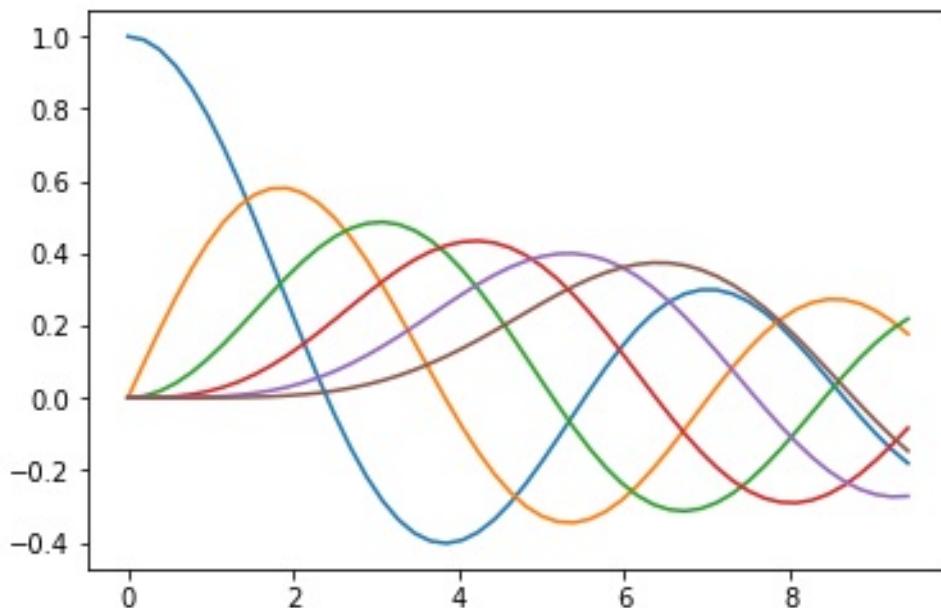
Jupyter QtConsole 4.3.1
Python 3.6.3 |Anaconda custom (32-bit)| (default, Oct 15 2017, 07:29:16) [MSC v. 1900 32 bit (Intel)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [4]:

Then, we type our commands here. The **QtConsole** is a lightweight application that feels like a Terminal. However, it provides a number of enhancements only possible in a GUI, such as inline figures, proper multi-line editing with syntax highlighting, and graphical calltips. The QtConsole can use any Jupyter kernel. Here is one example:

```
import numpy as np
from scipy.special import jn
import matplotlib.pyplot as plt
from matplotlib.pyplot import plot
#
x=np.linspace(0,3*np.pi)
for i in range(6):
    plot(x,jn(i,x))
#
plt.show()
```

The associated diagram is shown here:



```
{  
  "cells": [],  
  "metadata": {},  
  "nbformat": 4,  
  "nbformat_minor": 0  
}  
  
{  
  "cells": [  
    {  
      "cell_type": "code", "execution_count": null, "metadata": {  
        "collapsed": true }, "outputs": [], "source": [  
          "# import numpy as npn", "x=10n", "print(np.sqrt(x))"  
        ]  
    }  
  ],  
  "metadata": {  
    "kernelspec": {  
      "display_name": "Python 3", "language": "python", "name": "python3"  
    }, "language_info": {  
      "codemirror_mode": {
```

```
"name": "ipython", "version": 3  
}, "file_extension": ".py", "mimetype": "text/x-python", "name": "python",  
"nbconvert_exporter": "python", "pygments_lexer": "ipython3", "version":  
"3.6.3"  
}  
,  
"nbformat": 4,  
"nbformat_minor": 2  
}
```

Sharing of notebooks

To share a notebook or project, follow these steps:

1. Save your notebook.
2. Log in by running the Anaconda login command.
3. To upload your notebook to the cloud, open the Anaconda prompt or Terminal and enter the following command:

```
| anaconda upload my-notebook.ipynb
```

4. To check, we can try the link at <http://notebooks.anaconda.org/<USERNAME>/my-notebook>, where USERNAME would be your username. The following shows an example where we generate a simple Jupyter notebook, using the earlier code:

```
1 import numpy as np
2 x=10
3 print(np.sqrt(x))
```

5. Assume that we download it as c11_01.ipynb. First, we launch the Anaconda prompt.
6. Move to the correct subdirectory containing our Jupyter notebook. Then issue the following command:

```
| Anaconda upload c11_01.ipynb
```

7. The corresponding output is shown here:

```
<base> C:\Users\yany\Downloads>anaconda upload c11_01.ipynb
Using Anaconda API: https://api.anaconda.org
detecting package type ...
ipynb
extracting package attributes for upload ...
done

Uploading file paulyan/c11_01/2018.05.03.1447/c11_01.ipynb ...
uploaded 4 of 4Kb: 100.00% ETA: 0.0 minutes

Upload(s) Complete

Package located at:
https://anaconda.org/paulyan/c11_01
```

8. The last instruction shown in the preceding screenshot tells us that we can locate it at https://anaconda.org/paulyan/c11_01 (see the following screenshot):

The screenshot shows a web browser window with the URL https://anaconda.org/paulyan/c11_01/notebook in the address bar. The browser's navigation bar includes links for 'Sites', 'Angel', 'Buffalo Weather', 'Canisius - Outlook', 'Canisius College', and 'Google Scholar'. The main content area features the 'ANACONDA CLOUD' logo and a search bar. The page title is 'paulyan / notebooks / c11_01'. Below the title, there are three tabs: 'Notebook' (which is selected), 'Files', and 'Lab'. A dark blue banner displays the message 'You are viewing 2018.05.03.1447 ▾'. At the bottom, a code cell shows the following Python code:

```
In [ ]: # import numpy as np
          x=10
          print(np.sqrt(x))
```

Obviously, after that, we can share the link with our partners or coauthors.

Sharing of projects

First, let's look at the definition of a project. A project is a folder that contains an `anaconda-project.yml` configuration file together with scripts (code), notebooks, datasets, and other files. We can add a folder into a project by adding a configuration file named `anaconda-project.yml` to the folder. The configuration file can include the following sections: commands, variables, services, downloads, packages, channels, and environment specifications. Data scientists can use projects to encapsulate data science projects and make them easily portable. A project is usually compressed into a `.tar.bz2` file for sharing and storing.

Anaconda Project automates setup steps, so that people with whom you share projects can run your projects with the following single command:

```
| anaconda-project run
```

To install Anaconda Project, type the following:

```
| conda install anaconda-project
```

Anaconda Project encapsulates data science projects and makes them easily portable. It automates setup steps such as installing the right packages, downloading files, setting environment variables, and running commands. Project makes it easy to reproduce your work, share projects with others, and run them on different platforms. It also simplifies deployment to servers. Anaconda projects run the same way on your machine, on another user's machine, or when deployed to a server.

Traditional build scripts such as `setup.py` automate building of the project – going from source code to something runnable – while Project automates running the project, taking build artifacts, and doing any necessary setup before executing them.

We can use Project on Windows, macOS, and Linux. Project is supported and offered by Anaconda Inc® and contributors under a three-clause BSD license. Project sharing will save us a great deal of time, since other developers will not spend too much time on the work done already. Here is the procedure:

1. Build up your project
2. Log in to Anaconda
3. From the project's directory on your computer, type the following command:

```
| anaconda-project upload
```

Alternatively, from Anaconda Navigator, in the Projects tab, upload via the bottom-right Upload to Anaconda Cloud.

Projects can be any directory of code and assets. Often, projects will contain notebooks or Bokeh applications, for example. Here, we show how to generate a project called `project01`. First, we want to move to the correct location. Assume that we choose `c:/temp/`. The key command is given here:

```
| anaconda-project init --directory project01
```

Next, both commands are shown side by side as well:

```
$ cd c:/temp/  
$ anaconda-project init --directory project01  
Create directory 'c:temp\project01'? y  
Project configuration is in c:temp\project01\iris\anaconda-project.yml
```

The corresponding output is shown here:

```
<base> C:\Users\yany\Downloads>cd c:/temp  
  
<base> c:\temp>anaconda-project init --directory project01  
Create directory 'c:\temp\project01'? y  
Project configuration is in c:\temp\project01\anaconda-project.yml
```

We can also turn any existing directory into a project by switching to the directory and then running `anaconda-project init` without options or arguments. We can use MS Word to open `anaconda-project.yml` (see the first couple of lines shown here):

```
# This is an Anaconda project file.  
#  
# Here you can describe your project and how to run it.  
# Use `anaconda-project run` to run the project.  
# The file is in YAML format, please see http://www.yaml.org/start.html for more.  
#  
# Set the 'name' key to name your project  
#  
name: project01  
#
```

```
# Set the 'icon' key to give your project an icon
#
icon:
#
# Set a one-sentence-or-so 'description' key with project details
#
description:
#
# In the commands section, list your runnable scripts, notebooks, and other code.
# Use `anaconda-project add-command` to add commands.
#
```

There are two ways to share our projects with others. First, we archive the project by issuing the following command:

```
| anaconda-project archive project01.zip
```

Then, we email the ZIP file to our colleague or others. The second way of sharing a project is to use Anaconda Cloud. Log in to Anaconda Cloud first. From the project's directory on our computer, type `anaconda-project upload`, or, from Anaconda Navigator, in the Projects tab, upload via the bottom-right Upload to Anaconda Cloud.

Sharing of environments

In terms of computer software, an operating environment or integrated applications environment is the environment in which users can execute software. Usually, such an environment consists of a user interface and an API. To a certain degree, the term *platform* could be viewed as its synonym. There are many reasons why we want to share our environment with someone else. For example, they can re-create a test that we have done. To allow them to quickly reproduce our environment with all of its packages and versions, give them a copy of your `environment.yml` file. Depending on the operating system, we have the following methods to export our environment file. Note that if we already have an `environment.yml` file in our current directory, it will be overwritten during this task.

There are different ways to activate the `myenv` environment file depending on the systems used. For Windows users, in our Anaconda prompt, type the following command: **activate myenv**

On macOS and Linux, in our Terminal window, issue the following command:

```
| source activate myenv
```

Note that we replace `myenv` with the name of the environment. To export our active environment to a new file, type the following: **conda env export > environment.yml**

To share, we can simply email or copy the exported `environment.yml` file to the other person. On the other hand, in order to remove an environment, run the following code in our Terminal window or at an Anaconda prompt: **conda remove --name myenv --all**

Alternatively, we can specify the name, as shown here:

```
| conda env remove --name myenv
```

To verify that the environment was removed, run the following command line:

```
| conda info --envs
```

conda env export > myEnv.yml

name: base

channels:

- anaconda

- defaults

dependencies:

- bzip2=1.0.6=vc14h31cfb99_1

- curl=7.55.1=vc14h8488bd0_3

- freetype=2.8=vc14hfa46bb4_0

- hdf5=1.10.1=vc14h7c25517_0

- icu=58.2=vc14hbe8e9fa_0

- jpeg=9b=vc14h39a59fb_1

- libiconv=1.15=vc14h61b1384_5

- libpng=1.6.32=vc14hf267d31_3

- libssh2=1.8.0=vc14h91c0462_2

- xlrd=1.1.0=py36he7b893d_1

- xlsxwriter=1.0.2=py36hde92fec_0

- xlwings=0.11.4=py36h89f095e_0

- xlwt=1.3.0=py36hb79324b_0

- zict=0.1.3=py36hf291af1_0

prefix: C:\Users\yany\AppData\Local\Continuum\anaconda3

conda env create -f myEnv.yml

Downloading a package from Anaconda

Sometimes, users might generate their own packages and upload them to Anaconda Cloud. For a group project, group members can do so as well. Thus, we need to know how to download a package from Anaconda Cloud. Here is the general procedure:

1. Go to <http://anaconda.org>
2. Search packages by using a keyword or keywords, as in the example here:

The screenshot shows the Anaconda Cloud search interface. At the top, there is a search bar with the word "data" and a green search button. To the right of the search bar are "View" and "Help" dropdowns and a user profile icon for "paulyan". Below the search bar, there is a "Filters" section with dropdowns for "Type: All", "Access: All", and "Platform: All". At the bottom of the interface, there is a navigation bar with links for "Favorites", "Downloads", "Package (owner / package)", and "Platforms". The main content area displays two package results:

Rank	Downloads	Owner / Package	Description	Platform
0	163091	r / r-data.table	Fast aggregation of large data (e.g. 100GB in RAM), fast ordered joins, fast add/modify/delete of columns by group using no copies at all, list columns, a fast friendly file reader and parallel file writer. Offers a natural and flexible syntax, for faster development.	linux-32 linux-64 osx-64 win-32 win-64 conda
0	83482	conda-forge / poppler-data	Encoding data for the Poppler PDF manipulation library.	linux-64 osx-64 conda

Assume that we are interested in the r-data.table package.

3. Issue the following command:

```
| conda install -c r-data.table
```

Sometimes, we can generate a set of functions and share with others. Here is one simple example. Assume that we generate the following program with several functions:

```
def pv_f(r,n,fv):  
    """  
        Objective: estimate present value  
        r : period rate  
        n : number of periods  
        fv : future value  
        fv  
    formula used : pv = -----  
                    (1+r)**n  
    Example 1: >>>pv_f(0.1,1,100)    # meanings of input variables  
                    90.9090909090909    # based on their input order  
    Example #2 >>>pv_f(r=0.1,fv=100,n=1) # on keywords  
                    90.9090909090909  
    """  
    return fv/(1+r)**n  
#  
def pvGrowingPerpetuity(c,r,q):  
    return(c/(r-q))  
#  
def fv_f(pv,r,n):  
    return pv*(1+r)**n  
def fvAnnuity(r,n,c):  
    return c/r*((1+r)**n-1)  
#  
def fvAnnuityDue(r,n,c):  
    return c/r*((1+r)**n-1)*(1+r)
```

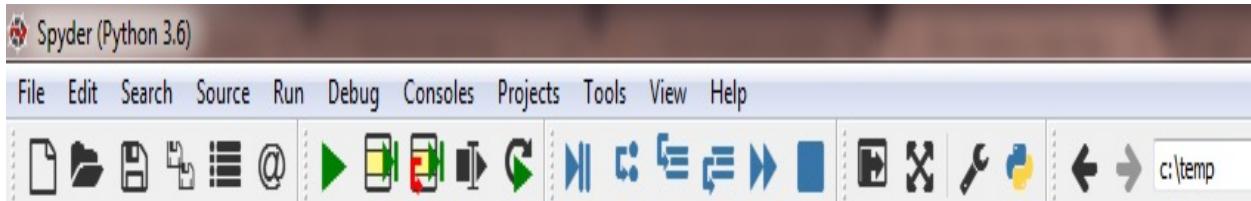
To save space, there is help for only the first function. Assume the preceding function is called `myfincal.py` and is located under `c:/temp`. We can use the following command to generate a `myfincal.pyc` and import `py_compile`.

```
| py_compile.compile('c:/temp/myfincal.py')
```

We have generated a set of functions contained in `fincal.pyc`. If you are interested, you can use the following procedure to download and import it:

1. Download it at <http://canisius.edu/~yany/python/fincal.pyc>.
2. Launch Python and try to import it. There are several ways to make `fincal.pyc` accessible:

- **Method 1:** Assume that the file is saved under `c:/temp/`. Change the path, as in the right-hand side of this screenshot:



- **Method 2:** Copy `fincal.pyc` to one of the accessible subdirectories. To see all paths, issue the following code:

```
| import sys
| sys.path
```

The output is shown here. Different users will have different output:

```
In [4]: import sys
...: sys.path
...:
Out[4]:
['',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\User\Scripts\python27.zip',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata\canopy-1.7.4.3348.win-x86\DLLs',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata\canopy-1.7.4.3348.win-x86\lib',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata\canopy-1.7.4.3348.win-x86\lib\plat-win',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata\canopy-1.7.4.3348.win-x86\lib\lib-tk',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata\canopy-1.7.4.3348.win-x86',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\User',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\User\lib\site-packages',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\User\lib\site-packages\win32',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\User\lib\site-packages\win32\lib',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\User\lib\site-packages\Pythonwin',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata\canopy-1.7.4.3348.win-x86\lib\site-packages\win32',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata\canopy-1.7.4.3348.win-x86\lib\site-packages\win32\lib',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata\canopy-1.7.4.3348.win-x86\lib\site-packages\Pythonwin',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata\canopy-1.7.4.3348.win-x86\lib\site-packages\IPython\extensions',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata\canopy-1.7.4.3348.win-x86',
'C:\Users\yany\AppData\Local\Enthought\Canopy32\App\appdata\canopy-1.7.4.3348.win-x86\lib\site-packages',
'C:\Users\yany\.ipython']
```

Now, you simply copy `fincal.pyc` to one of the preceding subdirectories.

- **Method 3:** Add the path instead, see the code here:

```
| import sys  
| sys.path.append('c:\\temp')
```

We can issue `sys.path` to check, see the following screenshot:

```
In [13]: sys.path  
Out[13]:  
['',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\User\\Scripts\\python27.zip',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata\\canopy-1.7.4.3348.win-x86\\DLLs',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata\\canopy-1.7.4.3348.win-x86\\lib',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata\\canopy-1.7.4.3348.win-x86\\lib\\plat-win',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata\\canopy-1.7.4.3348.win-x86\\lib\\lib-tk',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata\\canopy-1.7.4.3348.win-x86',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\User',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\User\\lib\\site-packages',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\User\\lib\\site-packages\\win32',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\User\\lib\\site-packages\\win32\\lib',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\User\\lib\\site-packages\\Pythonwin',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata\\canopy-1.7.4.3348.win-x86\\lib\\site-packages\\win32',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata\\canopy-1.7.4.3348.win-x86\\lib\\site-packages\\win32\\lib',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata\\canopy-1.7.4.3348.win-x86\\lib\\site-packages\\Pythonwin',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata\\canopy-1.7.4.3348.win-x86\\lib\\site-packages\\IPython\\extensions',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata\\canopy-1.7.4.3348.win-x86',  
 'C:\\Users\\yany\\AppData\\Local\\Enthought\\Canopy32\\App\\appdata\\canopy-1.7.4.3348.win-x86\\lib\\site-packages',  
 'C:\\Users\\yany\\.ipython',  
 'c:\\temp']
```

Note that if we want to remove a path, we can use the following code:

```
| import sys  
| sys.path.remove('c:\\temp')
```

3. Now, we could import that module, see the following code:

```
| import fincal  
| x=dir(fincal)  
| print(x)
```

The related output is shown here:

```
In [17]: import fincal
...: x=dir(fincal)
...: print(x)
...
['CND', 'EBITDA_value', 'IRR_f', 'IRRs_f', 'NPER', 'PMT', 'Rc_f', 'Rm_f', '__builtins__', '__doc__', '__file__', '__name__',
 '__package__', '__request', 'bondPrice', 'bsCall', 'convert_B_M', 'duration', 'exp', 'fincalHelp', 'fvAnnuity', 'fvAnnuityDue', 'fv_f',
 'get_200day_moving_avg', 'get_50day_moving_avg', 'get_52week_high', 'get_52week_low', 'get_EBITDA', 'get_all', 'get_avg_daily_volume',
 'get_book_value', 'get_change', 'get_dividend_per_share', 'get_dividend_yield', 'get_earnings_per_share', 'get_historical_prices',
 'get_market_cap', 'get_price', 'get_price_book_ratio', 'get_price_earnings_growth_ratio', 'get_price_earnings_ratio',
 'get_price_sales_ratio', 'get_short_ratio', 'get_stock_exchange', 'get_volume', 'log', 'market_cap', 'mean', 'modified_duration',
 'n_annuity', 'npv_f', 'payback_', 'payback_period', 'pi', 'pvAnnuity', 'pvAnnuityDue', 'pvAnnuity_k_period_from_today',
 'pvGrowingAnnuity', 'pvGrowingPerpetuity', 'pvPerpetuity', 'pvPerpetuityDue', 'pv_excel', 'pv_f', 'r_continuous', 'rateYan', 're', 'sign',
 'sqrt', 'urllib']
```

```
In [18]: |
```

Note that since Yahoo!Finance and Google Finance have changed their data structures, many functions related to downloading data from their websites will not work.

Summary

In this chapter, we have discussed Anaconda Cloud. Some topics included the Jupyter Notebook in depth, different formats of the Jupyter Notebook, how to share notebooks with your partner, how to share different projects over different platforms, how to share your working environments, and how to replicate others' environments locally.

For the next chapter, we will discuss distributed computing and Anaconda Accelerate. When our data or tasks become more complex, we'll need a good system or a set of tools to process data and run a complex algorithm. For this purpose, distributed computing is one solution. In particular, we will explain many concepts, such as compute nodes, project add-ons, parallel processing advanced Python for data parallelism. In addition, we will give several examples showing how to use distributed computing.

Review questions and exercises

1. What does *cloud* mean?
2. What does *cloud computing* mean?
3. Why do we care about sharing our working environments with others?
4. How do we register on Anaconda Cloud?
5. Do users need an account to use the Anaconda Cloud platform?
6. How do we upload our notebook to Anaconda Cloud?
7. Why is it important to share our projects?
8. How do you share your project? Build a project and share it with someone else.
9. How do you recreate your partner's Anaconda environment?
10. What is the meaning of the following two command lines:

```
conda remove --name myenv -all
```

```
conda info -envs
```

11. How can we launch a Jupyter QtConsole?
12. Run a Jupyter notebook called `octave Maric.ipynb` under the subdirectory of `examples/Builtin Extension`. Note that you have to find a way to install a package called `octavemagic` first.
13. Share your environment with another person to see the effect.
14. Generate a set of functions around a topic and produce a corresponding `.pyc` file and share it with your colleagues or co-workers.

Distributed Computing, Parallel Computing, and HPCC

Since our society has entered a data-intensive era (that is, a big data era), we face larger and larger datasets. For this reason, companies and users are considering what kinds of tools they could use to speed up the process when dealing with data. One obvious solution is to increase their data storage capacity. Unfortunately, there is a huge cost associated with this. The other solutions include distributed computing and some ways to accelerate our process.

In this chapter, we'll cover the following topics:

- Introduction to distributed versus parallel computing
- Understanding MPI
- Parallel processing in Python
- Compute nodes
- Anaconda add-ons
- Introduction to HPCC

Introduction to distributed versus parallel computing

Distributed computing is a subfield of computer science that studies distributed systems and models in which components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal.

It is worthwhile to discuss another phrase: parallel computing. Parallel computing is more tightly coupled to multi-threading, or how to make full use of a single CPU, while distributed computing refers to the notion of divide and conquer, executing subtasks on different machines, and then merging the results.

Since we have entered a so-called big data era, it seems that the distinction is melting. In fact, nowadays, many systems use a combination of parallel and distributed computing.

Task view for parallel processing

For R, there is a task view called **High-Performance and Parallel Computing with R**. Recall that a task view is a set of R programs around a specific topic. To find the task view for **High-Performance and parallel computing using R**, we go to <http://r-project.org>, click on CRAN on the left-hand side, choose a nearby mirror location, and click on Packages and Task Views.

After double-clicking on Task Views, we can see the following list—to save space, only the top task views are shown:

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning

After double-clicking on the related task view (**HighPerformanceComputing**), we can see the following screenshot—to save space, only the top few lines are shown:

CRAN Task View: High-Performance and Parallel Computing with R

Maintainer: Dirk Eddelbuettel

Contact: [Dirk.Eddelbuettel at R-project.org](mailto:Dirk.Eddelbuettel@R-project.org)

Version: 2016-01-28

This CRAN task view contains a list of packages, grouped by topic, that are useful for high-performance computing (HPC) with R. In this context, we are defining 'high-performance computing' rather loosely as just about anything related to pushing R a little further: using compiled code, parallel computing (in both explicit and implicit modes), working with large objects as well as profiling.

Unless otherwise mentioned, all packages presented with hyperlinks are available from CRAN, the Comprehensive R Archive Network.

Several of the areas discussed in this Task View are undergoing rapid change. Please send suggestions for additions and extensions for this task view to the [task view maintainer](#).

Suggestions and corrections by Achim Zeileis, Markus Schmidberger, Martin Morgan, Max Kuhn, Tomas Radovoyevitch, Jochen Knaus, Tobias Verbeke, Hao Yu, David Rosenberg, Marco Enea, Ivo Welch, Jay Emerson, Wei-Chen Chen, Bill Cleveland, Ross Boylan, Ramon Diaz-Uriarte, and Mark Zeligman (as well as others I may have forgotten to add here) are gratefully acknowledged.

The preceding task view contains a list of packages, grouped by various topics that could be useful for **high-performance computing (HPC)** with R. In this context, the task view maintainers define high-performance computing quite loosely as just about anything related to pushing R a little further: using compiled code, parallel computing (in both explicit and implicit modes), working with large objects as well as profiling. The aforementioned task view can be downloaded at <https://CRAN.R-project.org/view=HighPerformanceComputing>. As we have discussed in previous chapters, we could use the following command to install all related R packages:

```
| install.packages("ctv")
| library("ctv")
| install.views("HighPerformanceComputing")
```

On May 14, 2018, by running the previous code, 217 related R packages were downloaded and installed.

Sample programs in Python

For Python programs in parallel computing, we can visit the *IPython in-depth Tutorial* at <https://github.com/ipython/ipython-in-depth>. After downloading the ZIP file and unzipping it, we can find programs related to parallel computing (refer to the following two screenshot):

 daVinci Word Count	5/11/2018 1:41 PM	File folder
 interengine	5/3/2018 11:10 AM	File folder
 pi	5/3/2018 11:10 AM	File folder
 rmt	5/3/2018 11:10 AM	File folder
 wave2D	5/3/2018 11:10 AM	File folder
 workflow	5/3/2018 11:10 AM	File folder

The preceding screenshot shows a few subdirectories, while the following one shows 19 programs including both Jupyter Notebook and Python programs:

 customresults.py	2/28/2018 4:30 AM	PY File	2 KB
 dagdeps.py	2/28/2018 4:30 AM	PY File	4 KB
 Data Publication API.ipynb	2/28/2018 4:30 AM	IPYNB File	187 KB
 dependencies.py	2/28/2018 4:30 AM	PY File	4 KB
 fetchparse.py	2/28/2018 4:30 AM	PY File	3 KB
 Index.ipynb	2/28/2018 4:30 AM	IPYNB File	10 KB
 iopubwatcher.py	2/28/2018 4:30 AM	PY File	3 KB
 itermapresult.py	2/28/2018 4:30 AM	PY File	3 KB
 Monitoring an MPI Simulation - 1.ipynb	2/28/2018 4:30 AM	IPYNB File	42 KB
 Monitoring an MPI Simulation - 2.ipynb	2/28/2018 4:30 AM	IPYNB File	23 KB
 Monte Carlo Options.ipynb	2/28/2018 4:30 AM	IPYNB File	163 KB
 nwmerge.py	2/28/2018 4:30 AM	PY File	4 KB
 Parallel Decorator and map.ipynb	2/28/2018 4:30 AM	IPYNB File	3 KB
 Parallel Magics.ipynb	2/28/2018 4:30 AM	IPYNB File	11 KB
 phistogram.py	2/28/2018 4:30 AM	PY File	2 KB
 task_profiler.py	2/28/2018 4:30 AM	PY File	3 KB
 throughput.py	2/28/2018 4:30 AM	PY File	2 KB
 Using Dill.ipynb	2/28/2018 4:30 AM	IPYNB File	26 KB
 Using MPI with IPython Parallel.ipynb	2/28/2018 4:30 AM	IPYNB File	4 KB

Understanding MPI

Usually, a parallel algorithm needs to move data between different engines. One way to do so is by doing a pull and then a push using the direct view. However, this method is quite slow since all the data has to go through the controller to the client and then back through the controller, to its final destination. A much better way of moving data between engines is to use a message passing library, such as the **Message Passing Interface (MPI)**. IPython's parallel computing architecture has been designed to integrate with MPI. To download and install Windows MPI, readers can refer to <https://msdn.microsoft.com/en-us/library/bb524831%28v=vs.85%29.aspx>.

In addition, you could install the `mpi4py` package.

R package Rmpi

To find a demo associated with the `Rmpi` package, we could issue the following two lines:

```
| library(Rmpi)  
| demo()
```

After hitting *Enter*, we will see the following output:

```
R demos  
  
cslavePI          R function to use external C program cslavePI. Copy cslavePI.c in  
                  Rmpi library directory to your working directory and compile as  
                  "mpicc -o cslavePI cslavePI.c"  
masterslavePI     R script to compute PI=4*\int_0^1 1/(1+x^2) dx.  
simPI             Use Monto Carlo simulation to compute PI (with plot). Use simPI to  
                  run the program. Run mpi.setup.rngstream to setup a parallel RNG.  
simplePI          R function to compute PI using slavedaemon.R.  
slave1PI          R functions to compute PI using slavedaemon.R (load balancing with a  
                  server and workers). Use master1PI to run the program.  
slave2PI          R functions to compute PI using slavedaemon.R (load balancing with  
                  workers only). Use master2PI to run the program.  
  
Demos in package 'stats':  
  
glm.vr            Some glm() examples from V&R with several predictors  
lm.glm             Some linear and generalized linear modelling examples from 'An  
                  Introduction to Statistical Modelling' by Annette Dobson  
nlm                Nonlinear least-squares using nlm()  
smooth             'Visualize' steps in Tukey's smoothers
```

For the first function, called `cslavePI`, we know that we should copy `cslavePI.c` in `Rmpi` library directory to your working directory and compile it as `mpicc -o cslavePI cslavePI.c`. To find the path of `Rmpi`, we issue the `find.package()` function (refer to the code and the result in the following code):

```
| > find.package("Rmpi")  
| [1] "C:/Users/yany/Documents/R/win-library/3.4/Rmpi"
```

Note that different readers will get quite different paths. We can use `demo(cslavePI)` to find out its function. Refer to the code and the related output, as follows:

```
> demo(cslavePI)

demo(cslavePI)
-----
Type <Return> to start :

> cslavePI <- function (n)
+ {
+   if (mpi.comm.size(1) > 1)
+     stop ("It seems some slaves running on comm 1.")
+   mpi.comm.spawn("cslavePI")
+   mpi.intercomm.merge(2,0,1)
+   mpi.bcast(as.integer(n),1)
+   out <- mpi.reduce(0)
+   mpi.comm.free()
+   out
+ }
>
```

The mpicc software is used to compile and link MPI programs written in C.

```
library(plyr)

d1<-c(rep('A', 8), rep('B', 15), rep('C', 6)) d2<-sample(c("M", "F"), size = 29,
replace = TRUE) d3<-runif(n = 29, min = 18, max = 54)

#
# dfx <- data.frame(group =d1,sex=d2,age=d3)

#
ddply(dfx, .(group, sex), summarize, mean = round(mean(age), 2),
sd = round(sd(age), 2))

#
ddply(baseball[1:100,], ~ year, nrow) ddply(baseball, .(lg), c("nrow", "ncol"))
rbi<-ddply(baseball, .(year), summarise, mean_rbi=mean(rbi, na.rm = TRUE))
plot(mean_rbi~year,type="l",data = rbi) base2<-ddply(baseball,.(id),mutate,
career_year = year - min(year) + 1

)

library(datasets)

library(plyr)

data(mtcars)

#
d1<-mtcars[with(mtcars, order(cyl, disp)), ]

d2<-arrange(mtcars, cyl, disp)

myCars = cbind(vehicle=row.names(mtcars), mtcars) d3<-arrange(myCars, cyl,
```

disp)

```
d4<-arrange(myCars, cyl, desc(disp))
```

R package parallel

First, let's look at the simple usage of an R function called `lapply()` (refer to the following code):

```
|> lapply(1:3,function(x) c(sin(x),x^2))
|[1]
|[1] 0.841471 1.000000
|[2]
|[1] 0.9092974 4.0000000
|[3]
|[1] 0.14112 9.00000
```

The meaning is clear: we have an input size of 1, 2, and 3 and we assign them to three functions. The following example is a slightly more complex one:

```
|myFunctions<-c(sin(x),x^2+2,4*x^2-x^3-2)
|inputValue<-1:10
|output<-lapply(inputValue,function(x) myFunctions)
```

The first couple of lines are shown here:

```
> output
[[1]]
[1] 0.8414710 0.9092974 0.1411200 -0.7568025 -0.9589243
[6] -0.2794155 0.6569866 3.0000000 6.0000000 11.0000000
[11] 18.0000000 27.0000000 38.0000000 51.0000000 1.0000000
[16] 6.0000000 7.0000000 -2.0000000 -27.0000000 -74.0000000
[21] -149.0000000
```

The following example is borrowed from *Gordon (2015)*:

```
|library(parallel)
|n_cores <- detectCores() - 1
|cl <- makeCluster(n_cores)
|parLapply(cl, 2:4,function(exponent) 2^exponent)
|stopCluster(cl)
```

In the preceding code, the `makecluster()` function will set up the cluster. The `parLapply()` function calls the parallel version of `lapply()` or `parLapply()` functions. The output is shown here:

```
| [[1]]
```

```

[1] 4
[[2]]
[1] 8
[[3]]
[1] 16

```

For the following code, we will see an error message:

```

c2<-makeCluster(n_cores)
base <- 2
parLapply(c2, 2:4, function(exponent) base^exponent)
stopCluster(c2)

```

The error message is as follows:

```

Error in checkForRemoteErrors(val) :
  3 nodes produced errors; first error: object 'base' not found

```

To correct it, the base variable will be added (refer to the following code).

```

c3<-makeCluster(n_cores)
base <- 2
clusterExport(c3, "base")
parLapply(c3, 2:4, function(exponent) base^exponent)
stopCluster(c3)

```

To save space, the output will not be shown here. The following is another example to see the difference between calling the `lapply()` and `mclapply()` functions:

```

library(parallel)
detectCores()
myFunction<- function(iter=1,n=5){
  x<- rnorm(n, mean=0, sd=1 )
  eps <- runif(n, -2, 2)
  y <- 1 + 2*x + eps
  result<-lm( y ~ x )
  final<-cbind(result$coef,confint(result))
  return(final)
}
#
m<-5000
n2<-5000
system.time(lapply(1:m,myFunction,n=n2))
system.time(mclapply(1:m,myFunction,n=n2))

```

The output is shown here:

```

> system.time(lapply(1:n,myFunction,n=n2))
  user  system elapsed
 63.97   3.26  22.49
> system.time(mclapply(1:n,myFunction,n=n2))
  user  system elapsed

```

| 63.33 3.28 22.26

In the preceding code, the `lappy()` and `mclapply()` functions are used. The `mclapply()` function is a parallelized version of the `lapply()` function. It returns a list of the same length as `x`, each element of which is the result of applying `FUN` to the corresponding element of `x`. The following program is borrowed from <http://www.s mart-stats.org/wiki/parallel-computing-cluster-using-r> with minor modifications. Note that the program is run on a UNIX instead of a PC:

```
library(snow)
library(parallel)
#library(Rmpi)
myFunction<-function(n) {
  a<-rnorm(n)
  final<-log(abs(a))+a^3+2*a;
  return(final)
}
nCores=11;
#Using multicore
system.time(mclapply(rep(5E6,11),myFunction,mc.cores=nCores))
#Using snow via MPI
system.time(sapply(rep(5E6,11),myFunction))
#cl <- getMPIcluster()
cl <- makeCluster(c("localhost","localhost"), type = "SOCK")
system.time(parSapply(cl,rep(5E6,11),myFunction))
```

The related output is shown here:

```
> system.time(mclapply(rep(5E6,11),myFunction,mc.cores=nCores))
  user  system elapsed
 4.440   1.075   1.926
> system.time(sapply(rep(5E6,11),myFunction))
  user  system elapsed
10.294   0.992  11.286
> system.time(parSapply(cl,rep(5E6,11),myFunction))
  user  system elapsed
 0.655   0.626   7.328
> proc.time()
  user  system elapsed
15.621   2.936  22.134
```

```
library(snow) cl <- makeSOCKcluster(c("localhost","localhost"))
clusterApply(cl, 1:2, get("+"), 3) clusterEvalQ(cl, library(boot)) x<-1
clusterExport(cl, "x") clusterCall(cl, function(y) x + y, 2)
```

The `makeSOCKcluster()` function is used to start and stop a snow cluster and to set default cluster options. The `clusterApply()` calls the function on the first cluster node with arguments `seq[[1]]` and ..., on the second node with `seq[[2]]` and ..., and so on. If the length of `seq` is greater than the number of nodes in the cluster, then cluster nodes are recycled. A list of the results is returned; the length of the result list will be equal to the length of `seq`. The `clusterCall()` calls a function with identical arguments on each node in the cluster `cl` and returns a list of the results. The `clusterEvalQ()` function evaluates a literal expression on each cluster node. It's a cluster version of `evalq`, and is a convenience function defined in terms of `clusterCall()`.

Parallel processing in Python

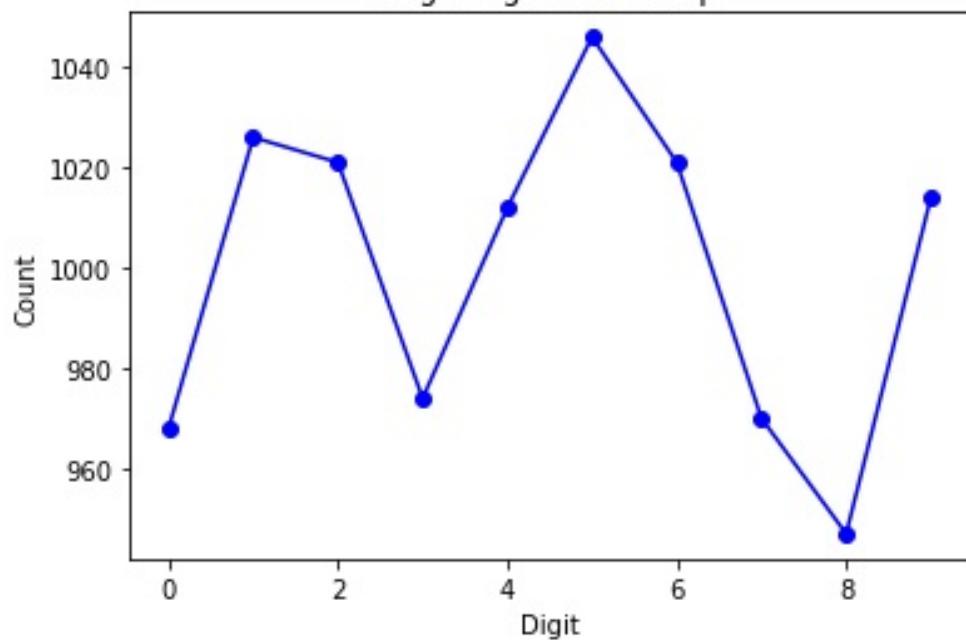
The following example is about computing π digits and is borrowed from the website <http://ipyparallel.readthedocs.io/en/latest/demos.html#parallel-examples>. Since the first part needs a program called `one_digit_freqs()` function, we could run a Python program called `pidigits.py` contained at `.../ipython-ipython-in-depth-4d98937\examples\Parallel Computing\pi`, and this path depends on where the reader downloaded and saved his/her files.

To complete our part, we simply include it in the first part of the program, as shown here:

```
import matplotlib.pyplot as plt
import sympy
import numpy as np
#
def plot_one_digit_freqs(f1):
    """
    Plot one digit frequency counts using matplotlib.
    """
    ax = plt.plot(f1, 'bo-')
    plt.title('Single digit counts in pi')
    plt.xlabel('Digit')
    plt.ylabel('Count')
    return ax
#
def one_digit_freqs(digits, normalize=False):
    """
    Consume digits of pi and compute 1 digit freq. counts.
    """
    freqs = np.zeros(10, dtype='i4')
    for d in digits:
        freqs[int(d)] += 1
    if normalize:
        freqs = freqs/freqs.sum()
    return freqs
#
pi = sympy.pi.evalf(40)
pi
pi = sympy.pi.evalf(10000)
digits = (d for d in str(pi)[2:]) # create a sequence of digits
freqs = one_digit_freqs(digits)
plot_one_digit_freqs(freqs)
plt.show()
```

The related graph is shown here:

Single digit counts in pi



```
text = open("c:/temp/daVinci.txt",'r').read().lower()
def byFreq(pair):
    return pair[1]

for ch in '!"#$%&()*+,-./;:<=>?@[\]^_`{|}~':
    text = text.replace(ch, ' ')
words = text.split()
counts = {}

for w in words:
    counts[w] = counts.get(w, 0) + 1

n = 10 # for the first n most frequently used words
words = list(counts.items())
words.sort()

words.sort(key=byFreq, reverse=True)
for i in range(n):
    word, count = words[i]
    print("{0:<15}{1:>5}".format(word, count))

<strong>the 22989
of 11232
and 8434
in 5750
to 5313
a 4725
is 4273
it 3071
that 2902
which 2545</strong>
```

Parallel Monte-Carlo options pricing

This example is from the sample programs in Python discussed earlier. The notebook shows how to use the `ipyparallel` package to do Monte-Carlo options pricing in parallel. The notebook computes the price of a large number of options for different strike prices and volatilities.

To save space, only the first several lines of code are given here:

```
%matplotlib inline
import matplotlib.pyplot as plt
import sys
import time
from ipyparallel import Client
import numpy as np
price = 100.0 # Initial price
rate = 0.05 # Interest rate
days = 260 # Days to expiration
paths = 10000 # Number of MC paths
n_strikes = 6 # Number of strike values
min_strike = 90.0 # Min strike price
max_strike = 110.0 # Max strike price
n_sigmas = 5 # Number of volatility values
min_sigma = 0.1 # Min volatility
max_sigma = 0.4 # Max volatility
# (more ....)
```

To run it, click on IPython Clusters. After clicking IPython Clusters on the right-hand side, we will see the following screenshot:

The screenshot shows the Jupyter Notebook interface. At the top, there is a navigation bar with 'jupyter' on the left and 'Logout' on the right. Below the navigation bar, there are three tabs: 'Files', 'Running', and 'IPython Clusters'. The 'IPython Clusters' tab is currently selected. Underneath the tabs, there is a section titled 'IPython parallel computing clusters'. A table lists two profiles: 'default' and 'home'. Both profiles are currently 'stopped'. Each profile row has an 'action' button labeled 'Start' on the right. The table has columns for 'profile', 'status', '# of engines', and 'action'.

profile	status	# of engines	action
default	stopped	<input type="button" value="▲\n▼"/>	<input type="button" value="Start"/>
home	stopped	<input type="button" value="▲\n▼"/>	<input type="button" value="Start"/>

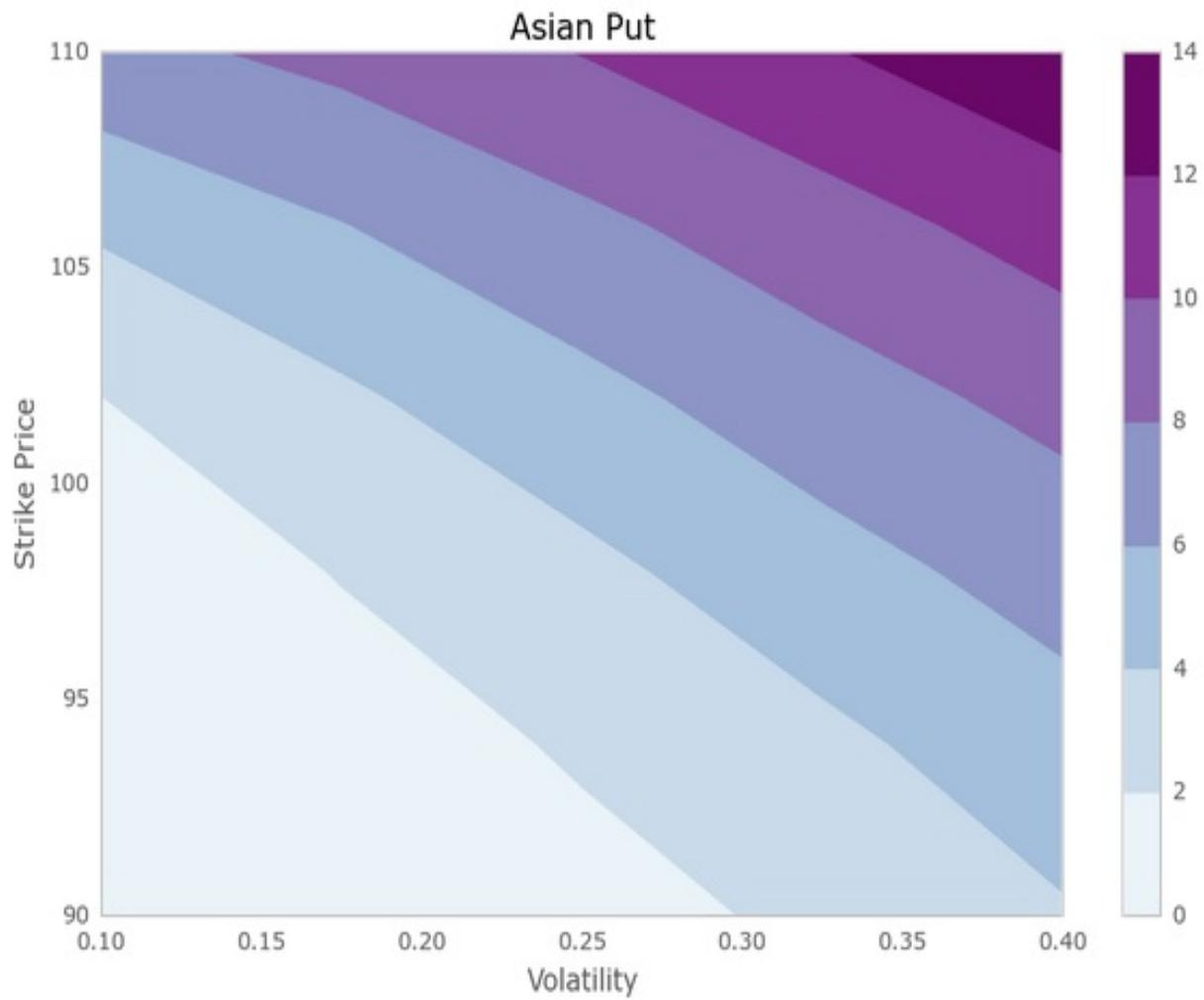
We can click on Start for the default. The number of engines will show a value

of 4, as follows:

The screenshot shows the Jupyter IPython Clusters interface. At the top, there is a navigation bar with tabs for 'Files', 'Running' (which is selected), and 'IPython Clusters'. On the right side of the header, there is a 'Logout' button. Below the header, the title 'IPython parallel computing clusters' is displayed, followed by a small refresh icon. The main content area is a table listing two profiles:

profile	status	# of engines	action
default	running	4	<button>Stop</button>
home	stopped	<button>Start</button>	<button>▲</button> <button>▼</button>

Now, we go back to our uploaded Jupyter Notebook for the parallel monte-carlo options pricing. In total, there are four output images. To save space, only the first one is shown here:



Note that the payoff function for an Asian put using stock average price as the final terminal price is given here, where $\text{Put}(\text{Asian})$ is the Asian put option, K is the exercise price, and \bar{S} is the average price over the path:

$$\text{Put}(\text{Asian}) = \text{Max}(K - \bar{S}, 0) \quad (1)$$

Compute nodes

A compute node provides the ephemeral storage, networking, memory, and processing resources that can be consumed by virtual machine instances. The cloud system supports two types of compute nodes: **ESX clusters**, where clusters are created in VMware vCenter Server, and **KVM compute nodes**, where KVM compute nodes are created manually. In the previous chapter, we mentioned the concept of the cloud.

Within a cloud environment, which is quite useful for a more complex project, compute nodes form the core of resources. Typically, these notes supply the processing, memory, network, and storage that virtual machine instances need. When an instance is created, it is matched to a compute node with the available resources. A compute node can host multiple instances until all of its resources are consumed.

Anaconda add-on

The following information is from the Anaconda Addon Development Guide.

An Anaconda add-on is a Python package containing a directory with an `__init__.py` file and other source directories (sub packages) inside. Because Python allows importing each package name only once, the package top-level directory name must be unique. At the same time, the name can be arbitrary, because add-ons are loaded regardless of their name; the only requirement is that they must be placed in a specific directory.

The suggested naming convention for add-ons is therefore similar to that of Java packages or D-Bus service names: prefix the add-on name with the reversed domain name of your organization, using underscores (`_`) instead of dots so that the directory name is a valid identifier for a Python package. An example add-on name following these suggestions would therefore be, for example, `org_fedora_hello_world`. This convention follows the recommended naming scheme for Python package and module names. Interested readers can find lots of information about Anaconda add-ons at <https://rhinstaller.github.io/anaconda-addon-development-guide/index.html#sect-anaconda-introduction-addons>.

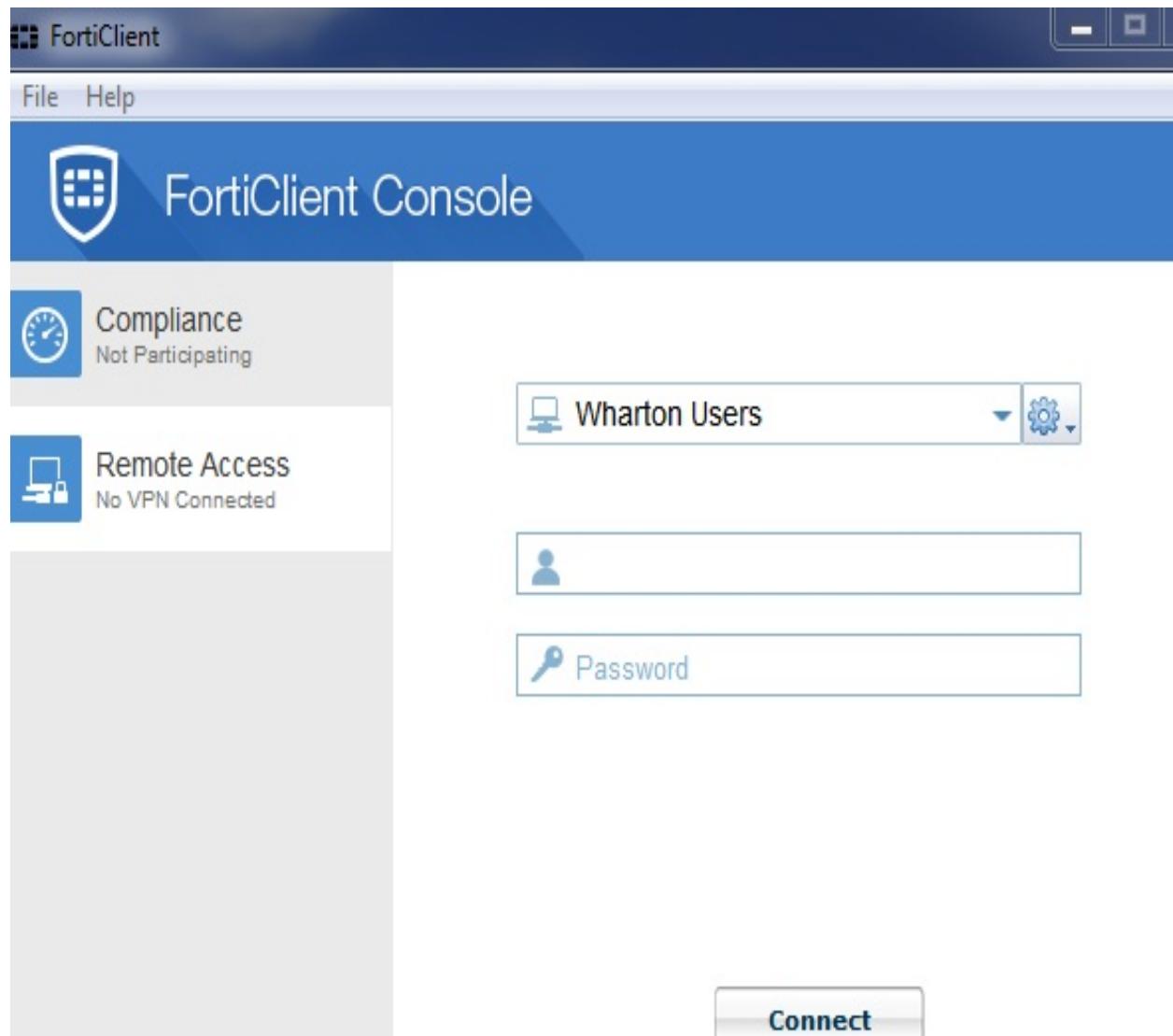
Introduction to HPCC

HPCC stands for **High-Performance Computing Cluster**. It is also known as **Data Analytics Supercomputer (DAS)**, an open source, data-intensive computing system platform developed by LexisNexis Risk Solutions. The HPCC platform incorporates a software architecture implemented on computing clusters to provide high-performance, data-parallel processing design for various applications using big data. The HPCC platform includes system configurations to support both parallel batch data processing (Thor) and high-performance online query applications using indexed data files (Roxie). The HPCC platform also includes a data-centric declarative programming language for parallel data processing called ECL.

You can see a simple example of using Wharton's HPCC system at <https://research-it.wharton.upenn.edu/documentation/>. Wharton's HPC Cluster (HPCC) provides access to advanced computational research hardware and software for Wharton faculty, faculty collaborators and research assistants, and Wharton doctoral candidates. It is designed for simple and parallel processing across a large set of tightly integrated hardware with dedicated networking and storage.

For more information about the hardware, refer to the Hardware page. HPCC users have access to a number of scientific, mathematics, and analytic software, including MATLAB, Mathematica, R, Stata, and SAS. MySQL server access can be provided as well. The HPCC also has Fortran, C, and C++ compilers in GNU and Intel versions. The following is a simple procedure to link to their HPCC platform:

1. First, download and install FortClient and MobaXterm software
2. Connect to the Wharton VPN via FortClient (as shown in the following screenshot):



3. After connecting, the following screen will appear:



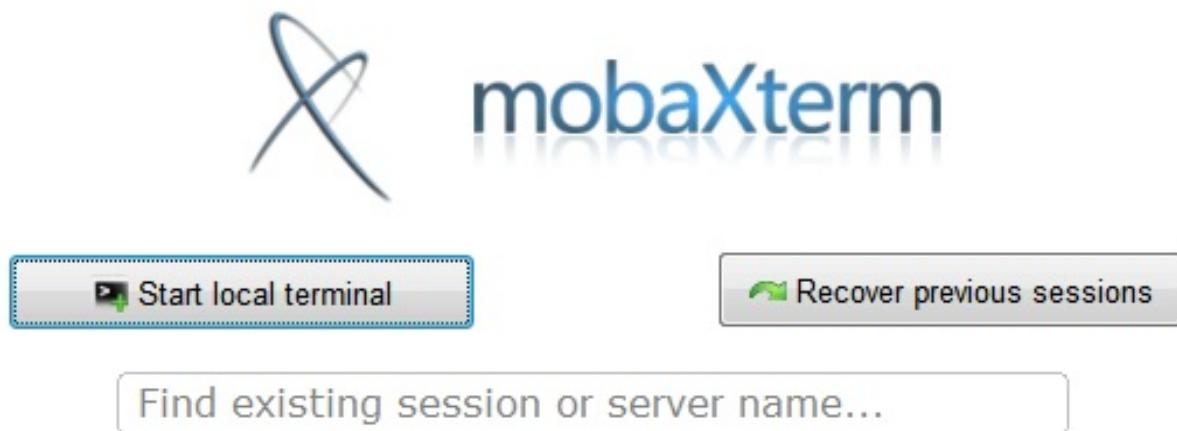
Wharton Users

10.150.16.27

⌚ Duration	00:00:17
➡ Bytes Received	2.31 KB
⬅ Bytes Sent	8.12 KB

[Disconnect](#)

4. Use MobaXterm software to connect to Wharton's HPCC platform. Here, we assume that you have an account with Wharton (refer to the following screenshot):



5. Now, we can connect, as shown in the following screenshot:

```
2. hpcc.wharton.upenn.edu (yxyan) X +  
• SSH-browser : ✓  
• X11-forwarding : ✓ (remote display is forwarded through SSH)  
• DISPLAY : ✓ (automatically set on remote server)  
► For more info, ctrl+click on help or visit our website  
  
Last login: Mon Jan 22 16:49:12 2018 from 10.150.16.18  
  
NOTICE:: Please do NOT run any computationally-intensive programs on the  
hpcc.wharton.upenn.edu login nodes. If you need assistance figuring out how  
to run jobs on the compute nodes with sample code, see the documentation at:  
https://research-it.wharton.upenn.edu/  
-----  
Welcome to [UPenn Logo] @ Wharton  
  
PLEASE type 'menu<Enter>' for current available applications  
PLEASE type 'hpccstatus<Enter>' for current cluster status  
  
UPDATES::  
  
2016-09-23 (Friday):  
MATLAB was upgraded to 2016b. If you use parallel pools, you will need to  
run 'setup-parMatlab' one time.  
  
2017-06-12 (Monday):  
Stata was upgraded from v14 to v15. Enjoy!  
  
* Interactive Matlab Users should use 'qrsh -pe openmp 4 matlab'.  
  
* If you are having troubles with 'qrsh' or 'qlogin', try the '-now no' option.  
  
NOTE: If you are not actively using an interactive session ('qrsh stata', etc.)  
please log out. Unused but locked sessions like that are a waste of Wharton's  
resources, and if noticed we will terminate those sessions. If the job will be  
running for more than a few minutes, you must use 'qsub'.  
  
CURRENT PER-PROJECT LIMITS (last updated 2017-05-18):  
Max TTL Cores: 64  
Max TTL RAM: 1024 GB  
Default RAM per job: 5 GB  
  
[yxyan@hpcc-login1 ~]$ █
```

6. Now, the users will write their own programs to take advantage of HPCC to

speed up their research and experiments. For more information, refer to the documentation at <https://research-it.wharton.upenn.edu/documentation/>.

Summary

In this chapter, we have discussed several R packages such as `plyr`, `snow`, `Rmpi`, and `parallel`, and the Python package `ipyparallel`. In addition, we mentioned compute nodes, project add-ons, parallel processing, and HPCC.

Now we've arrived at the end of our journey. We wish you good luck for the amazing endeavors you'll be taking up with the knowledge you've got from this book.

Review questions and exercises

1. What is distributed computing? Why it is useful?
2. From where could we get a task view for parallel computing?
3. From the task view related to parallel computing, we can find many R packages. Identify a few of them. Install two and find a few examples of using these two packages.
4. Conduct a word frequency analysis using: The Count of Monte Cristo by Alexandre Dumas (input file is at <http://www.gutenberg.org/files/1184/1184-0.txt>).
5. From where could we find more information about Anaconda add-ons?
6. What is HPCC and how does it work?
7. How do we find the path of an installed R package?
8. In the sample Jupyter notebook about parallel Monte-Carlo options pricing, the related Asian options are defined here, where `call(Asian)` is the Asian put option, `Put(Asian)`, K is the exercise price, and \bar{S} is the average price over the path:

$$\begin{aligned} Call(Asian) &= \text{Max}(\bar{S} - K, 0) \\ Put(Asian) &= \text{Max}(K - \bar{S}, 0) \end{aligned}$$

Write a Jupyter notebook to use the following definitions of Asian options:

$$\begin{aligned} Call(Asian) &= \text{Max}(S_T - \bar{S}, 0) \\ Put(Asian) &= \text{Max}(\bar{S} - S_T, 0) \end{aligned}$$

9. In this chapter, we mentioned that the following three lines could be used to download all R packages related to high-performance computing:

```
| install.packages("ctv")
| library("ctv")
| install.views("HighPerformanceComputing")
```

Try this and report how many R packages are downloaded.

10. Find more examples associated with the R package called `Rmpi`.
11. Run the sample Jupyter notebook called `using MPI` with `IPython Parallel.ipynb`.

Explain its meaning.

12. The R `doRNG` package provides functions to perform reproducible parallel `foreach` loops, using independent random streams as generated by the package `rstream`, suitable for the different `foreach` back ends. Download and install this and other related packages. Show a few examples.

References

Chapter 01: Ecosystem of Anaconda

- Anaconda's home page, <https://www.anaconda.com/>
- Anaconda Enterprise 5, The Enterprise-Ready Data Science Platform video (10 minutes and 54 seconds), <https://www.youtube.com/watch?v=r8DCh6Lv4TU>
- Anaconda Committee, <https://www.anaconda.com/community/>
- Anaconda forum, <https://groups.google.com/a/continuum.io/forum/#!forum/anaconda>
- Conda Cheat Sheet, https://conda.io/docs/_downloads/conda-cheatsheet.pdf
- Davenport, Thomas H. and D.J. Patil, 2012, Data Scientist: The Sexiest Job of the 21st Century, <https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century>
- Documents: Introduction to Julia, <https://docs.julialang.org/en/stable/>
- Fang, Bing and Peng Zhang, 2016, Big Data in Finance, in Big Data Concepts, Theories, and Applications, S. Yu and S. Guo, Eds., ed Cham: Springer, pp. 391-412
- Jupyter home page, <http://jupyter.org/>
- Python Tutorial: Anaconda - Installation and Using Conda (11 minutes and 24 seconds), <https://www.youtube.com/watch?v=YJC6ldI3hWk>
- Shi, Xiang, Peng Zhang, and Samee U. Khan, 2017, Quantitative Data Analysis in Finance, in Handbook of Big Data Technologies, A. Y. Zomaya and S. Sakr, Eds., Springer
- YouTube: Python Tutorial: Anaconda - Installation and Using Conda, (11 minutes 24 seconds) <https://www.youtube.com/watch?v=YJC6ldI3hWk>
- YouTube: [APMonitor.com](#), Install Julia on Windows and Linux with Jupyter Notebook (11 minutes 48 seconds), <https://www.youtube.com/watch?v=uRIQXJXRtqq>
- YouTube, Fisseha Berhane, Using Python, R, Julia and Octave with Jupyter Notebook (27 minutes, 40 seconds), <https://www.youtube.com/watch?v=psfSxvMOXyo>

Chapter 02: Anaconda Installation

- Anaconda's home page, <https://www.anaconda.com/>
- Anaconda Enterprise 5, The Enterprise-Ready Data Science Platform video (10 minutes and 54 seconds), <https://www.youtube.com/watch?v=r8DCh6Lv4TU>
- Anaconda Committee, <https://www.anaconda.com/community/>
- Anaconda forum, <https://groups.google.com/a/continuum.io/forum/#!forum/anaconda>
- Conda Cheat Sheet, https://conda.io/docs/_downloads/conda-cheatsheet.pdf
- Documents: Introduction to Julia, <https://docs.julialang.org/en/stable/>
- Getting started with Julia, <https://docs.julialang.org/en/latest/manual/getting-started/>
- Jupyter home page, <http://jupyter.org/>
- Octave installation, <https://anaconda.org/conda-forge/octave>
- Octave-kernel installation, https://anaconda.org/pypi/octave_kernel
- Python Tutorial: Anaconda - Installation and Using Conda (11 minutes and 24 seconds), <https://www.youtube.com/watch?v=YJC6ldI3hwk>
- Yan, Yuxing, 2017, Python for Finance, 2017, 2nd ed., *Packt Publishing*, ISBN: 978-1-78588-553-2
- Yan, Yuxing, 2016, Financial Modeling Using R, 2017, *Legaia Publishing (forthcoming)*
- YouTube: Python Tutorial: Anaconda - Installation and Using Conda, (11 minutes 24 seconds) <https://www.youtube.com/watch?v=YJC6ldI3hwk>
- YouTube: [APMonitor.com](#), Install Julia on Windows and Linux with Jupyter Notebook (11 minutes 48 seconds), <https://www.youtube.com/watch?v=uRIQXJXRtqq>
- YouTube, Fisseha Berhane, Using Python, R, Julia and Octave with Jupyter Notebook (27 minutes, 40 seconds), <https://www.youtube.com/watch?v=psfSxvMOXyo>

Chapter 03: Data Basics

- Black, Paul, 2017, *ig Data: 50 Fascinating and Free Data Sources for Data Visualization*, <https://www.bigdatanews.datasciencentral.com/profiles/blogs/big-data-50-fascinating-and-free-data-sources-for-data>
- Brownlee, Jason, 2017, How to Handle Missing Data with Python, <https://machinelearningmastery.com/handle-missing-data-python/>
- Datadotworld website, <https://data.world/>
- Julia packages, <https://docs.julialang.org/en/release-0.4/manual/packages/>
- McKinney, Wes, 2012, *Python for Data Analysis*, O'Reilly.
- Missing code for Julia, <https://docs.julialang.org/en/latest/manual/missing/>
- Python package datadotworld, <https://github.com/datadotworld/data.world-py>
- Quandl data delivery platform, <https://www.quandl.com>
- SPSS datasets, http://calcnet.mth.cmich.edu/org/spss/Prjs_DataSets.htm
- UCI Machine Learning Data Depository, <http://archive.ics.uci.edu/ml/index.php>
- Website for the Python Package of cbsodata, <https://pypi.python.org/pypi/cbsodata/0.1.3>

Chapter 04: Data Visualization

- Julia packages, <https://pkg.julialang.org>
- Python packages, <https://pypi.python.org/>
- R packages, <http://r-project.org>
- R package `pgraph` examples, <http://sachaepskamp.com/qgraph/examples>
- Sargent, T. J. and J. Stachurski (2017), https://lectures.quantecon.org/jl/julia_p_lots.html
- Writing mathematical expressions, <https://matplotlib.org/users/mathtext.html>

Chapter 05: Statistical Modeling in Anaconda

- Fama and French, 2013, A five-factor model, <https://www8.gsb.columbia.edu/programs/sites/programs/files/finance/Finance%20Seminar/spring%202014/ken%20french.pdf>
- Cheng, Evelyn and Robert Hum, 2018, Dow travels more than 22,000 points in one of the wildest weeks since the financial crisis, <https://www.cnbc.com/2018/02/09/dow-travels-20000-points-in-wild-week.html>
- Julia 0.7 documentation, <https://docs.julialang.org/en/latest/>
- Litner, J, 1965, The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets. Review of financial study, 1, 41-66
- Martz, Estonl, 2013, <http://blog.minitab.com/blog/understanding-statistics/handling-multicollinearity-in-regression-analysis>
- Mossin, J, 1966, equilibrium in a capital asset market, econometrica, 34, 768-783
- Python packages, <https://pypi.python.org/>
- Sharpe, W, 1964, capital asset prices: a theory of market equilibrium under conditions of risk, journal of finance, 19, 425-442

Chapter 06: Managing Packages

- Julia, package management, <https://docs.julialang.org/en/stable/manual/packages/>
- Julia packages, <https://pkg.julialang.org/>
- myPackage.py, <http://canisius.edu/~yany/python/myPackage.py>.
- Octave package, <https://octave.sourceforge.io/packages.php>
- Octave, overview for the package called statistics, <https://octave.sourceforge.io/statistics/overview.html>
- Python packages, <https://pypi.python.org/>
- Python documents, <https://www.python.org/doc/> and <https://packaging.python.org/>
- R home page, <http://r-project.org>
- Quant source, <https://github.com/wilsonfreitas/awesome-quant>
- Task View for Machine Learning, <http://canisius.edu/~yany/doc/taskViewMachineLearning.txt>
- Titatic text file , <http://canisius.edu/~yany/R/titanic02.txt>
- UCI Machine Learning Depository, <https://archive.ics.uci.edu/ml/index.php>

Chapter 07: Optimization in Anaconda

- Chandraprabhu, A. and R. Anantharaman, 2016, An Introduction to Machine Learning in Julia, <https://juliacomputing.com/blog/2016/09/28/knn-char-recognition.html>
- Fama and French, 2013, A five-factor model, <https://www8.gsb.columbia.edu/programs/sites/programs/files/finance/Finance%20Seminar/spring%202014/ken%20french.pdf>
- ff5industries.RData, <http://canisius.edu/~yany/RData/ff5industries.RData>
- Fama-French definition of 5 industries, http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/Data_Library/det_5_ind_port.html
- Frech's data library, http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html
- Johnson, Steven G, 2008,A Brief Overview of Optimization Problems, <https://math.mit.edu/~stevenj/18.335/optimization-handout.pdf>
- Julia, Optimization, <http://julianlsolvers.github.io/Optim.jl/latest/user/minimization/>
- JulioOpt, <http://www.juliaopt.org/>
- JuMP — Julia for Mathematical Programming , <https://jump.readthedocs.io/en/release-0.2/jump.html>
- JuMp installation web page, <http://jump.readthedocs.io/en/latest/installation.html>
- Machine-learning data depository, <https://archive.ics.uci.edu/ml/index.php>
- Octave package, <https://octave.sourceforge.io/packages.php>
- Octave package optim download page, <https://octave.sourceforge.io/optim/>
- Optimization Problems in Economics, <https://www.math24.net/optimization-problems-economics/>
- Optimization (scipy.optimize) , <https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html>
- Python package, <https://pypi.python.org/>
- quantecon Python package, <https://pypi.python.org/pypi/quantecon/>
- R home page, <https://r-project.org>
- Sargent, Thomas J, and John Stachurski, Optimal Taxation in an LQ Economy, <https://lectures.quantecon.org/py/lqramsey.html>
- Sargent, Thomas and John, Stachurski, <https://raw.githubusercontent.com/QuantE>

`con/QuantEcon.lectures.code/master/lqramsey/lqramsey.py`

- Unconstrained Optimization, http://julianlsolvers.github.io/Optim.jl/latest/use_r/minimization/

Chapter 08: Unsupervised Learning in Anaconda

- Arslan,Alex, Clustering.jl, <https://github.com/JuliaStats/Clustering.jl/blob/master/test/affprop.jl>
- Chandraprabhu, A. and R. Anantharaman, 2016, An Introduction to Machine Learning in Julia,<https://juliacomputing.com/blog/2016/09/28/knn-char-recognition.html>
- Clustering, <http://scikit-learn.org/stable/modules/clustering.html>
- CRAN Task View: Cluster Analysis & Finite Mixture Models, <https://CRAN.R-project.org/view=Cluster>
- Data sets that could be used for the fields of machine learning, https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research
- ffcMonthly.pkl, <http://www.canisius.edu/~yany/python/ffcMonthly.pkl>
- Julia packages, <https://pkg.julialang.org/>
- Hierarchical clustering (scipy.cluster.hierarchy), <https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html>
- Hierarchical clustering: structured vs unstructured ward , http://scikit-learn.org/stable/auto_examples/cluster/plot_ward_structured_vs_unstructured.html#sphx-glr-auto-examples-cluster-plot-ward-structured-vs-unstructured-py
- Julia 0.7 Documentation, <https://docs.julialang.org/en/latest/>
- Julia, clustering analysis manual, <http://clusteringjl.readthedocs.io/en/latest/>
- Julia, clustering analysis, <https://github.com/JuliaStats/Clustering.jl>
- Kaggle Wine data, <https://www.kaggle.com/maitree/kmeans-unsupervised-learning-using-wine-dataset/data>
- Matlab, unsupervised learning, <https://www.mathworks.com/discovery/unsupervised-learning.html>
- Tutorial for scipy.cluster.hierarchy, <https://stackoverflow.com/questions/21638130/tutorial-for-scipy-cluster-hierarchy>
- PCA example with Iris Data-set, http://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_iris.html
- Various Agglomerative Clustering on a 2D embedding of digits, http://scikit-learn.org/stable/auto_examples/cluster/plot_digits_linkage.html#sphx-glr-auto-examples-cluster-plot-digits-linkage-py
- Wind R data set, <http://canisius.edu/~yany/RData/wine.RData>

Chapter 09: Supervised Learning in Anaconda

- Chandraprabhu, A. and R. Anantharaman, 2016, An Introduction to Machine Learning in Julia,<https://juliacomputing.com/blog/2016/09/28/knn-char-recognition.html>
- Cross validated, <https://stats.stackexchange.com/questions/21572/how-to-plot-decision-boundary-of-a-k-nearest-neighbor-classifier-from-elements-of-21602#21602>
- French's Data Library, http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html
- Ng, Andrew, Linear Regression, openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex2/ex2.html
- Octave machine learning library at <https://github.com/partharamanujam/octave-ml>
- Sklearn package (Python) , <http://scikit-learn.org/stable/modules/clustering.html>
- Proellocos, Nicolas , 2017, Reinforcement Learning in R, <https://cran.r-project.org/web/packages/ReinforcementLearning/vignettes/ReinforcementLearning.html>
- UCI data set Iris at <https://archive.ics.uci.edu/ml/datasets/Iris>
- UCI Machine-learning depository, <https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data>
- UCI machine learning Bank Marketing Data Set , <https://archive.ics.uci.edu/ml/datasets/Bank+Marketing>
- UCI machine learning Census Income Data Set , <https://archive.ics.uci.edu/ml/datasets/Census+Income>
- UCI machine learning Bike share Data Set , <https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset>
- UCI machine learning Bag of Words Data Set , <https://archive.ics.uci.edu/ml/datasets/Bag+of+Words>
- Recognizing hand-written digits , http://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html#sphx-glr-auto-examples-classification-plot-digits-classification-py
- Yahoo!Finance, <http://finance.yahoo.com>

Chapter 10: Predictive Data Analytics – Modelling and Validation

- IPython in-depth Tutorial, first presented at PyCon 2012, <https://github.com/ipython/ipython-in-depth>
- Clustering for Python package called `sklearn`, <http://scikit-learn.org/stable/modules/clustering.html>
- Clustering analysis for Julia, <http://clusteringjl.readthedocs.io/en/latest/>
- Clustering analysis for Julia, <https://github.com/JuliaStats/Clustering.jl>
- Code, http://scikit-learn.org/stable/auto_examples/text/document_classification_20newsgroups.html#sphx-glr-auto-examples-text-document-classification-20newsgroups-py
- Comparing Python Clustering Algorithms, http://hdbSCAN.readthedocs.io/en/latest/comparing_clustering_algorithms.html
- `ffMonthly.pkl` at <http://canisius.edu/~yany/python/data/ffMonthly.pkl>
- French Data Library, http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html
- Ligges, Uwe and Martin Machler, Scatterplot3d –an R package for Visualizing Multivariate Data, <https://cran.r-project.org/web/packages/scatterplot3d/vignettes/s3d.pdf>
- Litner, J, 1965, The valuation of risk asset and the selection of risky investments in stock portfolios and capital budgets. Review of Financial Study, 1, 41-66.
- Manual for Octave package called `ltfat`, <http://ltfat.github.io/doc/lutfat.pdf>
- Mossin, J. 1966. Equilibrium in a Capital Asset Market, *Econometrica* 34, 768-783.
- Sklearn package (Python) , <http://scikit-learn.org/stable/modules/clustering.html>
- Skylearn download, <http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz>
- Under-fitting vs. Overfitting at http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html#sphx-glr-auto-examples-model-selection-plot-underfitting-overfitting-py
- Yan, Yuxing and Zhang, 2014, Quality of PIN Estimates and PIN-return Relationship, *Journal of Banking and Finance*
- Yahoo!finance, <http://fiannce.yahoo.com>

Chapter 11: Anaconda Cloud

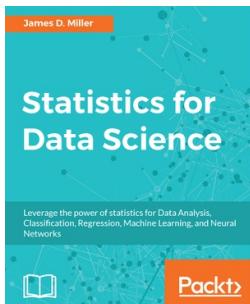
- Anaconda home page, <http://anaconda.org>
- Financial calculator download page, <http://canisius.edu/~yany/python/fincal.pyc>
- Jupyter notebook in depth, <https://github.com/ipython/ipython-in-depth>
- Jupyter Notebook Format, <http://nbformat.readthedocs.io/en/latest/>
- YAML, <http://www.yaml.org/start.html>

Chapter 12: Distributed Computing, Parallel Computing, and HPCC

- Da Vinci Code at <http://www.gutenberg.org/files/5000/5000-8.txt>
- Dumas, Alexandre, Count of Monte Cristo, <http://www.gutenberg.org/files/1184/1184-0.txt>
- Gordon, Max, 2015, How-to go parallel in R – basics + tips, <http://gforge.se/2015/02/how-to-go-parallel-in-r-basics-tips/>
- IPython in-depth Tutorial at <https://github.com/ipython/ipython-in-depth>.
- Parallel Computing on the Cluster using R, <http://www.smart-stats.org/wiki/parallel-computing-cluster-using-r>
- Parallel examples, <http://ipyparallel.readthedocs.io/en/latest/demos.html#parallel-examples>
- Python add-ons, <https://rhinstaller.github.io/anaconda-addon-development-guide/index.html#sect-anaconda-introduction-addons>
- Task View for High-Performance Computing is at <https://CRAN.R-project.org/view=HighPerformanceComputing>
- Wharton's HPCC system, <https://research-it.wharton.upenn.edu/documentation/>
- Wharton research, <https://research-it.wharton.upenn.edu/documentation/>
- Windows MPI download page, <https://msdn.microsoft.com/en-us/library/bb524831%28v=vs.85%29.aspx>

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

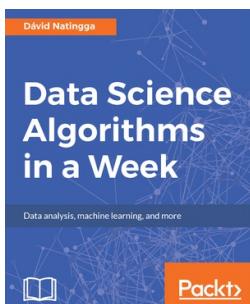


Statistics for Data Science

James D. Miller

ISBN: 978-1-78829-067-8

- Analyze the transition from a data developer to a data scientist mindset
- Get acquainted with the R programs and the logic used for statistical computations
- Understand mathematical concepts such as variance, standard deviation, probability, matrix calculations, and more
- Learn to implement statistics in data science tasks such as data cleaning, mining, and analysis
- Learn the statistical techniques required to perform tasks such as linear regression, regularization, model assessment, boosting, SVMs, and working with neural networks
- Get comfortable with performing various statistical computations for data science programmatically



Data Science Algorithms in a Week

Dávid Natingga

ISBN: 978-1-78728-458-6

- Find out how to classify using Naive Bayes, Decision Trees, and Random Forest to achieve accuracy to solve complex problems
- Identify a data science problem correctly and devise an appropriate prediction solution using Regression and Time-series
- See how to cluster data using the k-Means algorithm
- Get to know how to implement the algorithms efficiently in the Python and R languages

Leave a review - let other readers know what you think

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!