MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Autonomous Institution - UGC, Govt. of India

APPROVED BY AICTE, ACCREDITED BY NBA & NAAC – 'A' GRADE ISO 9001:2015 CERTIFIED

B.TECH	**II – I**	**R20**	**2021-2022**

Introduction to Data Science

[R20DS501]

DIGITAL NOTES

DEPARTMENT OF CSE (DATA SCIENCE, CYBER SECURITY, INTERNET OF THINGS)

Introduction to Datascience

(R20DS501)

LECTURE NOTES

B.TECH II YEAR – I SEM (R20)

(2021-2022)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
(DATA SCIENCE,CYBER SECURITY,INTERNET OF THINGS)

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, INDIA.

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

II Year B.Tech CSE(DS) – I Sem

L/T/P/C
3 -/-/-3

(R20DS501) Introduction to Datascience

COURSE OBJECTIVES:

1. An understanding of the data operations
2. An overview of simple statistical models and the basics of machine learning techniques of regression.
3. An understanding good practices of data science
4. Skills in the use of tools such as python, IDE
5. Understanding of the basics of the Supervised learning

UNIT-1
Introduction, Toolboxes: Python, fundamental libraries for data Scientists. Integrated development environment (IDE). Data operations: Reading, selecting, filtering, manipulating, sorting, grouping, rearranging, ranking, and plotting.

UNIT-2
Descriptive statistics, data preparation. Exploratory Data Analysis data summarization, data distribution, measuring asymmetry. Sample and estimated mean, variance and standard score. Statistical Inference frequency approach, variability of estimates, hypothesis testing using confidence intervals, using p-values

UNIT-3
Supervised Learning: First step, learning curves, training-validation and test. Learning models generalities, support vector machines, random forest. Examples

UNIT-4
Regression analysis, Regression: linear regression simple linear regression, multiple & Polynomial regression, Sparse model. Unsupervised learning, clustering, similarity and distances, quality measures of clustering, case study.

UNIT-5
Network Analysis, Graphs, Social Networks, centrality, drawing centrality of Graphs, PageRank, Ego-Networks, community Detection

TEXT/REFERENCES BOOK:

1. Introduction to Data Science a Python approach to concepts, Techniques and Applications, Igual, L;Seghi', S. Springer, ISBN:978-3-319-50016-4
2. Data Analysis with Python A Modern Approach, David Taieb, Packt Publishing, ISBN-9781789950069
3. Python Data Analysis, Second Ed., Armando Fandango, Packt Publishing, ISBN: 9781787127487

COURSE OUTCOMES:

1. Describe what Data Science is and the skill sets needed to be a data scientist
2. Explain the significance of exploratory data analysis (EDA) in data science
3. Ability to learn the supervised learning, SVM
4. Apply basic machine learning algorithms (Linear Regression)
5. Explore the Networks, PageRank

UNIT-1

Introduction, Toolboxes: Python, fundamental libraries for data Scientists. Integrated development environment (IDE). Data operations: Reading, selecting, filtering, manipulating, sorting, grouping, rearranging, ranking, and plotting.

Introduction to Data Science

1.1 What is Data Science?

You have, no doubt, already experienced data science in several forms. When you are looking for information on the web by using a search engine or asking your mobile phone for directions, you are interacting with data science products. Data science has been behind resolving some of our most common daily tasks for several years.

Most of the scientific methods that power data science are not new and they havebeen out there, waiting for applications to be developed, for a long time. Statistics is an old science that stands on the shoulders of eighteenth-century giants such as Pierre Simon Laplace (1749–1827) and Thomas Bayes (1701–1761). Machine learning isyounger, but it has already moved beyond its infancy and can be considered a well- established discipline. Computer science changed our lives several decades ago andcontinues to do so; but it cannot be considered new.

So, why is data science seen as a novel trend within business reviews, in technologyblogs, and at academic conferences?

The novelty of data science is not rooted in the latest scientific knowledge, but in a disruptive change in our society that has been caused by the evolution of technology: datification. Datification is the process of rendering into data aspects of the world that have never been quantified before. At the personal level, the list of datified conceptsis very long and still growing: business networks, the lists of books we are reading,the films we enjoy, the food we eat, our physical activity, our purchases, our drivingbehavior, and so on. Even our thoughts are datified when we publish them on our favorite social network; and in a not so distant future, your gaze could be datified by wearable vision registering devices. At the business level, companies are datifying semi-structured data that were previously discarded: web activity logs, computer network activity, machinery signals, etc. Nonstructured data, such as written reports,e-mails, or voice recordings, are now being stored not only for archive purposes butalso to be analyzed.

However, datification is not the only ingredient of the data science revolution. The other ingredient is the democratization of data analysis. Large companies such as Google, Yahoo, IBM, or SAS were the only players in this field when data science had no name. At the beginning of the century, the huge computational resources of those companies allowed them to take advantage of datification by using analytical techniques to develop innovative products and even to take decisions about their own business. Today, the analytical gap between those companies and the rest of the world (companies and people) is shrinking. Access to cloud computing allows any individual to analyze huge amounts of data in short periods of time. Analytical knowledge is free and most of the crucial algorithms that are needed to create a solution can be found, because open-source development is the norm in this field. As a result, the possibility of using rich data to take evidence-based decisions is open to virtually any person or company.

Data science is commonly defined as a methodology by which actionable insights can be inferred from data. This is a subtle but important difference with respect to previous approaches to data analysis, such as business intelligence or exploratory statistics. Performing data science is a task with an ambitious objective: the produc-tion of beliefs informed by data and to be used as the basis of decision-making. In the absence of data, beliefs are uninformed and decisions, in the best of cases, are based on best practices or intuition. The representation of complex environments by rich data opens up the possibility of applying all the scientific knowledge we have regarding how to infer knowledge from data.

In general, data science allows us to adopt four different strategies to explore theworld using data:

1. *Probing reality*. Data can be gathered by passive or by active methods. In the latter case, data represents the response of the world to our actions. Analysis of those responses can be extremely valuable when it comes to taking decisions about our subsequent actions. One of the best examples of this strategy is the use of A/B testing for web development: What is the best button size and color? The best answer can only be found by probing the world.
2. *Pattern discovery*. Divide and conquer is an old heuristic used to solve complex problems; but it is not always easy to decide how to apply this common sense to problems. Datified problems can be analyzed automatically to discover useful patterns and natural clusters that can greatly simplify their solutions. The use of this technique to profile users is a critical ingredient today in such importantfields as programmatic advertising or digital marketing.
3. *Predicting future events*. Since the early days of statistics, one of the most impor-tant scientific questions has been how to build robust data models that are capa- ble of predicting future data samples. Predictive analytics allows decisions to be taken in response to future events, not only reactively. Of course, it is not possible to predict the future in any environment and there will always be unpre- dictable events; but the identification of predictable events represents valuableknowledge. For example, predictive analytics can be used

to optimize the tasks

planned for retail store staff during the following week, by analyzing data suchas weather, historic sales, traffic conditions, etc.

4. *Understanding people and the world*. This is an objective that at the moment is beyond the scope of most companies and people, but large companies and governments are investing considerable amounts of money in research areas such as understanding natural language, computer vision, psychology and neu- roscience. Scientific understanding of these areas is important for data science because in the end, in order to take optimal decisions, it is necessary to know the real processes that drive people's decisions and behavior. The development of deep learning methods for natural language understanding and for visual object recognition is a good example of this kind of research.

Toolboxes for Data Scientists

Introduction

In this chapter, first we introduce some of the tools that data scientists use. The toolbox of any data scientist, as for any kind of programmer, is an essential ingredient for success and enhanced performance. Choosing the right tools can save a lot of time and thereby allow us to focus on data analysis.

The most basic tool to decide on is which programming language we will use. Many people use only one programming language in their entire life: the first and only one they learn. For many, learning a new language is an enormous task that, if at all possible, should be undertaken only once. The problem is that some languages are intended for developing high-performance or production code, such as C, C++, or Java, while others are more focused on prototyping code, among these the best known are the so-called scripting languages: Ruby, Perl, and Python. So, depending on the first language you learned, certain tasks will, at the very least, be rather tedious. The main problem of being stuck with a single language is that many basic tools simply will not be available in it, and eventually you will have either to reimplementthem or to create a bridge to use some other language just for a specific task.

In conclusion, you either have to be ready to change to the best language for each task and then glue the results together, or choose a very flexible language with a rich ecosystem (e.g., third-party open-source libraries). In this book we have selected Python as the programming language.

Why Python?

Python[1] is a mature programming language but it also has excellent properties for newbie programmers, making it ideal for people who have never programmed before. Some of the most remarkable of those properties are easy to read code, suppression of non-mandatory delimiters, dynamic typing, and dynamic memory usage. Pythonis an interpreted language, so the code is executed immediately in the Python con- sole without needing the compilation step to machine language. Besides the Python console (which comes included with any Python installation) you can find other in-teractive consoles, such as IPython,[2] which give you a richer environment in whichto execute your Python code.

Currently, Python is one of the most flexible programming languages. One of its main characteristics that makes it so flexible is that it can be seen as a multiparadigm language. This is especially useful for people who already know how to program with other languages, as they can rapidly start programming with Python in the same way. For example, Java programmers will feel comfortable using Python as it supports the object-oriented paradigm, or C programmers could mix Python and C code using *cython*. Furthermore, for anyone who is used to programming in functional languages such as Haskell or Lisp, Python also has basic statements for functional programming in its own core library.

In this book, we have decided to use Python language because, as explained before, it is a mature language programming, easy for the newbies, and can be used as a specific platform for data scientists, thanks to its large ecosystem of scientific libraries and its high and vibrant community. Other popular alternatives to Python for data scientists are R and MATLAB/Octave.

Fundamental Python Libraries for Data Scientists

The Python community is one of the most active programming communities with a huge number of developed toolboxes. The most popular Python toolboxes for any data scientist are NumPy, SciPy, Pandas, and Scikit-Learn.

Numeric and Scientific Computation: NumPy and SciPy

NumPy[3] is the cornerstone toolbox for scientific computing with Python. NumPy provides, among other things, support for multidimensional arrays with basic oper-ations on them and useful linear algebra functions. Many toolboxes use the NumPyarray representations as an efficient basic data structure. Meanwhile, *SciPy* provides a collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimization, statistics, and much more. Another core toolbox in SciPyis the plotting library *Matplotlib*. This toolbox has many tools for data visualization.

SCIKIT-Learn: Machine Learning in Python

Scikit-learn[4] is a machine learning library built from NumPy, SciPy, and Matplotlib. Scikit-learn offers simple and efficient tools for common tasks in data analysis such as classification, regression, clustering, dimensionality reduction, model selection, and preprocessing.

PANDAS: Python Data Analysis Library

Pandas[5] provides high-performance data structures and data analysis tools. The keyfeature of Pandas is a fast and efficient DataFrame object for data manipulation withintegrated indexing. The DataFrame structure can be seen as a spreadsheet which offers very flexible ways of working with it. You can easily transform any dataset in the way you want, by reshaping it and adding or removing columns or rows. It alsoprovides high-performance functions for aggregating, merging, and joining dataset-
s. Pandas also has tools for importing and exporting data from different formats: comma-separated value (CSV), text files, Microsoft Excel, SQL databases, and the fast HDF5 format. In many situations, the data you have in such formats will not be complete or totally structured. For such cases, Pandas offers handling of miss-ing data and intelligent data alignment. Furthermore, Pandas provides a convenientMatplotlib interface.

Data Science Ecosystem Installation

Before we can get started on solving our own data-oriented problems, we will need toset up our programming environment. The first question we need to answer concerns

Python language itself. There are currently two different versions of Python: Python 2.X and Python 3.X. The differences between the versions are important, so there isno compatibility between the codes, i.e., code written in Python 2.X does not workin Python 3.X and vice versa. Python 3.X was introduced in late 2008; by then, a lotof code and many toolboxes were already deployed using Python 2.X (Python 2.0 was initially introduced in 2000). Therefore, much of the scientific community didnot change to Python 3.0 immediately and they were stuck with Python 2.7. By now, almost all libraries have been ported to Python 3.0; but Python 2.7 is still maintained, so one or another version can be chosen. However, those who already have a large amount of code in 2.X rarely change to Python 3.X. In our examples throughout thisbook we will use Python 2.7.

Once we have chosen one of the Python versions, the next thing to decide is whether we want to install the data scientist Python ecosystem by individual tool- boxes, or to perform a bundle installation with all the needed toolboxes (and a lot more). For newbies, the second option is recommended. If the first option is chosen,then it is only necessary to install all the mentioned toolboxes in the previous section, in exactly that order.

However, if a bundle installation is chosen, the Anaconda Python distribution[6] is then a good option. The Anaconda distribution provides integration of all the Python toolboxes and applications needed for data scientists into a single directory without mixing it with other Python toolboxes installed on the machine. It contain- s, of course, the core toolboxes and applications such as NumPy, Pandas, SciPy, Matplotlib, Scikit-learn, IPython, Spyder, etc., but also more specific tools for other related tasks such as data visualization, code optimization, and big data processing.

Integrated Development Environments (IDE)

For any programmer, and by extension, for any data scientist, the integrated de- velopment environment (IDE) is an essential tool. IDEs are designed to maximize programmer productivity. Thus, over the years this software has evolved in order tomake the coding task less complicated. Choosing the right IDE for each person is crucial and, unfortunately, there is no “one-size-fits-all” programming environment. The best solution is to try the most popular IDEs among the community and keep whichever fits better in each case.

In general, the basic pieces of any IDE are three: the editor, the compiler, (or interpreter) and the debugger. Some IDEs can be used in multiple programming languages, provided by language-specific plugins, such as Netbeans[7] or Eclipse.[8] Others are only specific for one language or even a specific programming task. In

the case of Python, there are a large number of specific IDEs, both commercial (PyCharm,[9] WingIDE[10] …) and open-source. The open-source community helps IDEs to spring up, thus anyone can customize their own environment and share it with the rest of the community. For example, Spyder[11] (Scientific Python Development EnviRonment) is an IDE customized with the task of the data scientist in mind.

Web Integrated Development Environment (WIDE): Jupyter

With the advent of web applications, a new generation of IDEs for interactive lan- guages such as Python has been developed. Starting in the academia and e- learningcommunities, web-based IDEs were developed considering how not only your codebut also all your environment and executions can be stored in a server. One of the first applications of this kind of WIDE was developed by William Stein in early 2005 using Python 2.3 as part of his SageMath mathematical software. In

SageMath, a server can be set up in a center, such as a university or school, and then students can work on their homework either in the classroom or at home, starting from exactly the same point they left off. Moreover, students can execute all the previous steps overand over again, and then change some particular *code cell* (a segment of the docu- ment that may content source code that can be executed) and execute the operation again. Teachers can also have access to student sessions and review the progress orresults of their pupils.

Nowadays, such sessions are called notebooks and they are not only used in classrooms but also used to show results in presentations or on business dashboards. The recent spread of such notebooks is mainly due to IPython. Since December 2011, IPython has been issued as a browser version of its interactive console, called IPython notebook, which shows the Python execution results very clearly and concisely by means of cells. Cells can contain content other than code. For example, markdown (a wiki text language) cells can be added to introduce algorithms. It is also possible toinsert Matplotlib graphics to illustrate examples or even web pages. Recently, some scientific journals have started to accept notebooks in order to show experimental results, complete with their code and data sources. In this way, experiments can become completely and absolutely replicable.

Since the project has grown so much, IPython notebook has been separated fromIPython software and now it has become a part of a larger project: Jupyter[12]. Jupyter (for Julia, Python and R) aims to reuse the same WIDE for all these interpreted languages and not just Python. All old IPython notebooks are automatically imported to the new version when they are opened with the Jupyter platform; but once they

Get Started with Python for Data Scientists

Throughout this book, we will come across many practical examples. In this chapter, we will see a very basic example to help get started with a data science ecosystem from scratch. To execute our examples, we will use Jupyter notebook, although anyother console or IDE can be used.

The Jupyter Notebook Environment

Once all the ecosystem is fully installed, we can start by launching the Jupyter notebook platform. This can be done directly by typing the following command onyour terminal or command line: $ jupyter notebook

If we chose the bundle installation, we can start the Jupyter notebook platform by clicking on the Jupyter Notebook icon installed by Anaconda in the start menu or onthe desktop.

The browser will immediately be launched displaying the Jupyter notebook home- page, whose URL is http://localhost:8888/tree. Note that a special port is used; by default it is 8888. As can be seen in Fig. 2.1, this initial page displays a tree view of a directory. If we use the command line, the root directory is the same directory where we launched the Jupyter notebook. Otherwise, if we use the Anaconda launcher, the root directory is the current user directory. Now, to start a new

New » Notebooks » Python 2

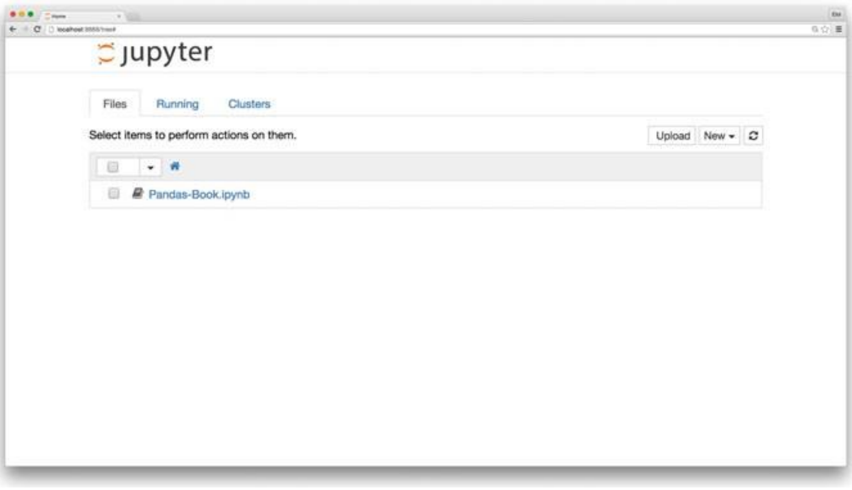
notebook, we only
need to press button at the top on the right of the
thehome page.

As can be seen in Fig. 2.2, a blank notebook is created called Untitled. First of all, we are going to change the name of the notebook to something more appropriate. To do this, just click on the notebook name and rename it: DataScience-GetStartedExample.

Let us begin by importing those toolboxes that we will need for our program. In the first cell we put the code to import the *Pandas* library as pd. This is for convenience; every time we need to use some functionality from the Pandas library, we will write pd instead of pandas. We will also import the two core libraries mentioned above: the numpy library as npand the matplotlib library as plt.

In []:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```



Select items to perform actions on them.

Fig. 2.1 IPython notebook home page, displaying a home tree directory

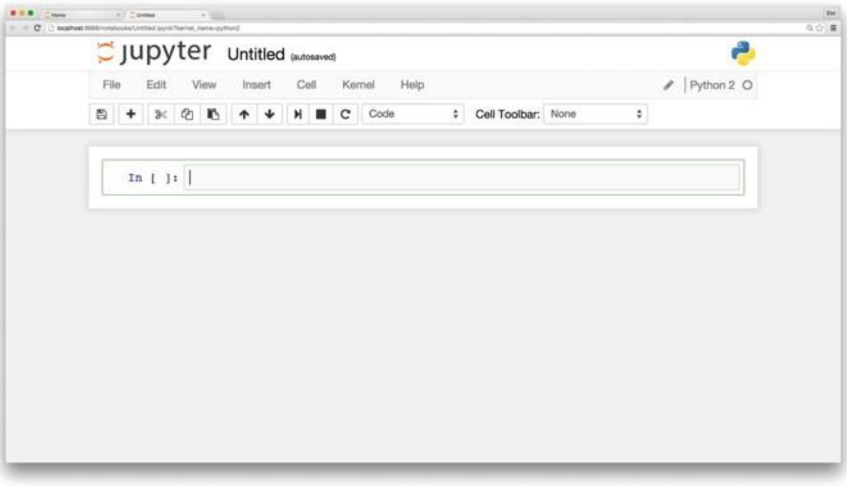


Fig. 2.2 An empty new notebook

To execute just one cell, we press the ¸ button or click on Cell》Run or press the keys Ctrl + Enter. While execution is underway, the header of the cell shows the * mark:

In [*]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

While a cell is being executed, no other cell can be executed. If you try to executeanother cell, its execution will not start until the first cell has finished its execution.Once the execution is finished, the header of the cell will be replaced by the next number of execution. Since this will be the first cell executed, the number shown will be 1. If the process of importing the libraries is correct, no output cell is produced.

In[1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

For simplicity, other chapters in this book will avoid writing these imports.

The DataFrame Data Structure

The key data structure in Pandas is the DataFrame object. A DataFrame is basically a tabular data structure, with rows and columns. Rows have a specific index to access them, which can be any name or value. In Pandas, the columns are called Series,a special type of data, which in essence consists of a list of several values, where each value has an index. Therefore, the DataFrame data structure can be seen as a spreadsheet, but it is much more flexible. To understand how it works, let us see how to create a DataFrame from a common Python dictionary of lists. First, we will create a new cell by clicking Insert › Insert Cell Below or pressing the keys Ctrl + B. Then, we write in the following code:

In[2]:

```
data = {'year': [
        2010, 2011, 2012,
        2010, 2011, 2012,
        2010, 2011, 2012
    ],
    'team': [
        'FCBarcelona', 'FCBarcelona',
        'FCBarcelona', 'RMadrid',
        'RMadrid', 'RMadrid',
        'ValenciaCF', 'ValenciaCF',
        'ValenciaCF'
    ],
    'wins':   [30, 28, 32, 29, 32, 26, 21, 17, 19],
    'draws':  [6, 7, 4, 5, 4, 7, 8, 10, 8],
    'losses': [2, 3, 2, 4, 2, 5, 9, 11, 11]
    }
football = pd.DataFrame(data, columns = [
    'year', 'team', 'wins', 'draws', 'losses'
    ]
)
```

In this example, we use the pandas DataFrame object constructor with a dictionary of lists as argument. The value of each entry in the dictionary is the name of the column, and the lists are their values.

The DataFrame columns can be arranged at construction time by entering a key-word columnswith a list of the names of the columns ordered as we want. If the column keyword is not present in the constructor, the columns will be arranged in alphabetical order. Now, if we execute this cell, the result will be a table like this:

Out[2]:

	year	team	wins	draws	losses
0	2010	FCBarcelona	30	6	2
1	2011	FCBarcelona	28	7	3
2	2012	FCBarcelona	32	4	2
3	2010	RMadrid	29	5	4
4	2011	RMadrid	32	4	2
5	2012	RMadrid	26	7	5
6	2010	ValenciaCF	21	8	9
7	2011	ValenciaCF	17	10	11
8	2012	ValenciaCF	19	8	11

where each entry in the dictionary is a column. The index of each row is created automatically taking the position of its elements inside the entry lists, starting from 0. Although it is very easy to create DataFrames from scratch, most of the time what we will need to do is import chunks of data into a DataFrame structure, and we willsee how to do this in later examples.

Apart from DataFrame data structure creation, Panda offers a lot of functions to manipulate them. Among other things, it offers us functions for aggregation, manipulation, and transformation of the data. In the following sections, we will introduce some of these functions.

Open Government Data Analysis Example Using Pandas

To illustrate how we can use Pandas in a simple real problem, we will start doing some basic analysis of government data. For the sake of transparency, data producedby government entities must be open, meaning that they can be freely used, reused,and distributed by anyone. An example of this is the Eurostat, which is the home ofEuropean Commission data. Eurostat's main role is to process and publish compa- rable statistical information at the European level. The data in Eurostat are providedby each member state and it is free to reuse them, for both noncommercial and commercial purposes (with some minor exceptions).

Since the amount of data in the Eurostat database is huge, in our first study we are only going to focus on data relative to indicators of educational funding by the member states. Thus, the first thing to do is to retrieve such data from Eurostat. Since open data have to be delivered in a plain text format, CSV (or any other delimiter-separated value) formats are commonly used to store tabular data. In a delimiter-separated value file, each line is a data record and each record consist- s of one or more fields, separated by the delimiter character (usually a comma). Therefore, the data we will use can be found already processed at book's Github repository as educ_figdp_1_Data.csv file. Of course, it can also be download- ed as unprocessed tabular data from the Eurostat database site[13] following the path:

Tables by themes 》 Population and social conditions 》 Education and training 》 Education 》 Indicators on education finance 》 Public expenditure on education .

Reading

Let us start reading the data we downloaded. First of all, we have to create a new notebook called Open Government Data Analysis and open it. Then, after ensuring that the educ_figdp_1_Data.csvfile is stored in the same directoryas our notebook directory, we will write the following code to read and show the content:

In[1]:

```
edu = pd.read_csv('files/ch02/educ_figdp_1_Data.csv',
                  na_values = ':',
                  usecols = ["TIME","GEO","Value"])
edu
```

Out[1]:

	TIME	GEO	Value
0	2000	European Union ...	NaN
1	2001	European Union ...	NaN
2	2002	European Union ...	5.00
3	2003	European Union ...	5.03
...	...	...	...
382	2010	Finland	6.85
383	2011	Finland	6.76

384 rows × 5 columns

The way to read CSV (or any other separated value, providing the separator character) files in Pandas is by calling the read_csvmethod. Besides the nameof the file, we add the na_values key argument to this method along with the character that represents "non available data" in the file. Normally, CSV files have aheader with the names of the columns. If this is the case, we can use the usecols parameter to select which columns in the file will be used.

In this case, the DataFrame resulting from reading our data is stored in edu. The output of the execution shows that the edu DataFrame size is 384 rows× 3 columns. Since the DataFrame is too large to be fully displayed, three dots appear in the middle of each row.

Beside this, Pandas also has functions for reading files with formats such as Excel, HDF5, tabulated files, or even the content from the clipboard (read_excel(), read_hdf(), read_table(), read_clipboard()). Whichever function we use, the result of reading a file is stored as a DataFrame structure.

To see how the data looks, we can use the head() method, which shows just the first five rows. If we use a number as an argument to this method, this will be the number of rows that will be listed:

.

In [2]:
```
edu . head ()
```

Out[2]:

	TIME	GEO	Value
0	2000	European Union ...	NaN
1	2001	European Union ...	NaN
2	2002	European Union ...	5.00
3	2003	European Union ...	5.03
4	2004	European Union ...	4.95

Similarly, it exists the tail()method, which returns the last five rows by default.

In [3]:
```
edu . tail ()
```

Out[3]:

379	2007	Finland	5.90
380	2008	Finland	6.10
381	2009	Finland	6.81
382	2010	Finland	6.85
383	2011	Finland	6.76

If we want to know the names of the columns or the names of the indexes, we can use the DataFrame attributes columns and index respectively. The names of the columns or indexes can be changed by assigning a new list of the same length to these attributes. The values of any DataFrame can be retrieved as a Python array bycalling its valuesattribute.

If we just want quick statistical information on all the numeric columns in a DataFrame, we can use the function describe(). The result shows the count, the mean, the standard deviation, the minimum and maximum, and the percentiles, by default, the 25th, 50th, and 75th, for all the values in each column or series.

In[4]:
```
edu . describe ()
```

Out[4]:

	TIME	Value
count	384.000000	361.000000
mean	2005.500000	5.203989
std	3.456556	1.021694
min	2000.000000	2.880000
25%	2002.750000	4.620000
50%	2005.500000	5.060000
75%	2008.250000	5.660000
max	2011.000000	8.810000

Name: Value, dtype: float64

Selecting Data

If we want to select a subset of data from a DataFrame, it is necessary to indicate this subset using square brackets ([]) after the DataFrame. The subset can be specified in several ways. If we want to select only one column from a DataFrame, we only need to put its name between the square brackets. The result will be a Series data structure, not a DataFrame, because only one column is retrieved.

In [5]:

```
edu['Value']
```

Out[5]:

```
0       NaN
1       NaN
2      5.00
3      5.03
4      4.95
... ...380 6.10
381 6.81
382 6.85
383 6.76
Name: Value, dtype: float64
```

If we want to select a subset of rows from a DataFrame, we can do so by indicating a range of rows separated by a colon (:) inside the square brackets. This is commonly known as a *slice* of rows:

In [6]:

```
edu[10:14]
```

Out[6]:

	TIME	GEO	Value
10	2010	European Union (28 countries)	5.41
11	2011	European Union (28 countries)	5.25
12	2000	European Union (27 countries)	4.91
13	2001	European Union (27 countries)	4.99

This instruction returns the slice of rows from the 10th to the 13th position. Notethat the slice does not use the index labels as references, but the position. In this case, the labels of the rows simply coincide with the position of the rows.

If we want to select a subset of columns and rows using the labels as our references instead of the positions, we can use ixindexing:

In [7]:

```
edu.ix[90:94, ['TIME','GEO']]
```

Out[7]:

	TIME	GEO
90	2006	Belgium
91	2007	Belgium
92	2008	Belgium
93	2009	Belgium
94	2010	Belgium

This returns all the rows between the indexes specified in the slice before the comma, and the columns specified as a list after the comma. In this case, ixreferences the index labels, which means that ix does not return the 90th to 94th rows, but it returns all the rows between the row labeled 90 and the row labeled 94; thus if the index 100 is placed between the rows labeled as 90 and 94, this row would also bereturned.

Filtering Data

Another way to select a subset of data is by applying Boolean indexing. This indexing is commonly known as a *filter*. For instance, if we want to filter those values less than or equal to 6.5, we can do it like this:

In [8]:

```
edu[edu['Value'] > 6.5].tail()
```

Out[8]:

	TIME	GEO	Value
218	2002	Cyprus	6.60
281	2005	Malta	6.58
94	2010	Belgium	6.58
93	2009	Belgium	6.57
95	2011	Belgium	6.55

Boolean indexing uses the result of a Boolean operation over the data, returninga mask with True or False for each row. The rows marked True in the mask will be selected. In the previous example, the Boolean operation edu['Value'] >produces a Boolean mask. When an element in the "Value" column is greaterthan 6.5, the corresponding value in the mask is set to True, otherwise it is set to False. Then, when this mask is applied as an index in edu[edu['Value'] > 6.5], the result is a filtered DataFrame containing only rows with values higher than 6.5. Of course, any of the usual Boolean operators can be used for filtering: <(less than),<= (less than or equal to), > (greater than), >= (greater than or equal to), = (equal to), and != (not equal to).

Filtering Missing Values

Pandasuses the special value NaN(not a number) to represent missing values. InPython, NaNis

Table 2.1 List of most common aggregation functions

Function	Description
count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of values
median()	Arithmetic median of values
min()	Minimum
max()	Maximum
prod()	Product of values
std()	Unbiased standard deviation
var()	Unbiased variance

one of their results ends in an undefined value. A subtle feature of NaN values is that two NaN are never equal. Because of this, the only safe way to tell whether a value is missing in a DataFrame is by using the isnull() function. Indeed, this function can be used to filter rows with missing values:

In [9]:

```
edu [ edu [" Value "]. isnull () ]. head ()
```

Out[9]:

	TIME	GEO	Value
0	2000	European Union (28 countries)	NaN
1	2001	European Union (28 countries)	NaN
36	2000	Euro area (18 countries)	NaN
37	2001	Euro area (18 countries)	NaN
48	2000	Euro area (17 countries)	NaN

Manipulating Data

Once we know how to select the desired data, the next thing we need to know is howto manipulate data. One of the most straightforward things we can do is to operate with columns or rows using aggregation functions. Table 2.1 shows a list of the most common aggregation functions. The result of all these functions applied to a row orcolumn is always a number. Meanwhile, if a function is applied to a DataFrame or aselection of rows and columns, then you can specify if the function should be appliedto the rows for each column (setting the axis=0keyword on the invocation of thefunction), or it should be applied on the columns for each row (setting the axis=1keyword on the invocation of the function).

In [10]:

```
edu . max ( axis  =  0)
```

Out[10]:
```
TIME          2011
GEO          Spain
Value         8.81
dtype: object
```

Note that these are functions specific to Pandas, not the generic Python functions. There are differences in their implementation. In Python, NaN values propagate through all operations without raising an exception. In contrast, Pandas operations exclude NaN values representing missing data. For example, the pandas max function excludes NaN values, thus they are interpreted as missing values, while the standard Python max function will take the mathematical interpretation of NaN and return it as the maximum:

In [11]:
```
print "Pandas max function:", edu['Value'].max()
print "Python max function:", max(edu['Value'])
```

Out[11]:
```
Pandas max function: 8.81Python max function:
nan
```

Beside these aggregation functions, we can apply operations over all the values in rows, columns or a selection of both. The rule of thumb is that an operation between columns means that it is applied to each row in that column and an operation between rows means that it is applied to each column in that row. For example we can applyany binary arithmetical operation (+,-,*,/) to an entire row:

In [12]:
```
s = edu["Value"]/100
s.head()
```

Out[12]:
```
0        NaN
1        NaN
2     0.0500
3     0.0503
4     0.0495
Name: Value, dtype: float64
```

However, we can apply any function to a DataFrame or Series just setting its name as argument of the apply method. For example, in the following code, we apply the sqrt function from the NumPy library to perform the square root of each value in the Value column.

In [13]:
```
s = edu["Value"].apply(np.sqrt)
s.head()
```

Out[13]:
```
0          NaN
1          NaN
2     2.236068
3     2.242766
4     2.224860
Name: Value, dtype: float64
```

If we need to design a specific function to apply it, we can write an in-line function, commonly known as a λ-function. A λ-function is a function without a name. It is only necessary to specify the parameters it receives, between the lambda keyword and the colon (:). In the next example, only one parameter is needed, which will bethe value of each element in the Value column. The value the function returns will be the square of that value.

In [14]:
```
s = edu["Value"].apply(lambda d: d**2)
s.head()
```

Out[14]:
```
0              NaN
1              NaN
2          25.0000
3          25.3009
4          24.5025
Name: Value, dtype: float64
```

Another basic manipulation operation is to set new values in our DataFrame. This can be done directly using the assign operator (=) over a DataFrame. For example, to add a new column to a DataFrame, we can assign a Series to a selection of a column that does not exist. This will produce a new column in the DataFrame after all the others. You must be aware that if a column with the same name already exists, the previous values will be overwritten. In the following example, we assign the Seriesthat results from dividing the Value column by the maximum value in the same column to a new column named ValueNorm.

In [15]:
```
edu['ValueNorm'] = edu['Value']/edu['Value'].max()
edu.tail()
```

Out[15]:

	TIME	GEO	Value	ValueNorm
379	2007	Finland	5.90	0.669694
380	2008	Finland	6.10	0.692395
381	2009	Finland	6.81	0.772985
382	2010	Finland	6.85	0.777526
383	2011	Finland	6.76	0.767310

Now, if we want to remove this column from the DataFrame, we can use the drop function; this removes the indicated rows if axis=0, or the indicated columns if axis=1. In Pandas, all the functions that change the contents of a DataFrame, such as the drop function, will normally return a copy of the modified data, instead of overwriting the DataFrame. Therefore, the original DataFrame is kept. If you do notwant to keep the old values, you can set the keyword inplaceto True. By default, this keyword is set to False, meaning that a copy of the data is returned.

In [16]:
```
edu.drop('ValueNorm', axis = 1, inplace = True)
edu.head()
```

Out[16]:

	TIME	GEO	Value
0	2000	European Union (28 countries)	NaN
1	2001	European Union (28 countries)	NaN
2	2002	European Union (28 countries)	5
3	2003	European Union (28 countries)	5.03
4	2004	European Union (28 countries)	4.95

Instead, if what we want to do is to insert a new row at the bottom of the DataFrame,we can use the Pandas appendfunction. This function receives as argumentthe new row, which is represented as a dictionary where the keys are the nameof the columns and the values are the associated value. You must be aware to setting the ignore_index flag in the append method to True, otherwise the index 0is given to this new row, which will produce an error if it already

In [17]:

```
edu = edu.append({"TIME": 2000,"Value": 5.00,"GEO": 'a'},
                 ignore_index = True)
edu.tail()
```

Out[17]:

	TIME	GEO	Value
380	2008	Finland	6.1
381	2009	Finland	6.81
382	2010	Finland	6.85
383	2011	Finland	6.76
384	2000	a	5

Finally, if we want to remove this row, we need to use the drop function again. Now we have to set the axis to 0, and specify the index of the row we want to remove. Since we want to remove the last row, we can use the max function over the indexesto determine which row is.

In [18]:

```
edu.drop(max(edu.index), axis = 0, inplace = True)
edu.tail()
```

Out[18]:

	TIME	GEO	Value
379	2007	Finland	5.9
380	2008	Finland	6.1
381	2009	Finland	6.81
382	2010	Finland	6.85
383	2011	Finland	6.76

The drop()function is also used to remove missing values by applying it overthe result of the isnull() function. This has a similar effect to filtering the NaN values, as we explained above, but here the difference is that a copy of the DataFramewithout the NaNvalues is returned, instead of a view.

In [19]:

```
eduDrop = edu.drop(edu["Value"].isnull(), axis = 0)
eduDrop.head()
```

Out[19]:

	TIME	GEO	Value
2	2002	European Union (28 countries)	5.00
3	2003	European Union (28 countries)	5.03
4	2004	European Union (28 countries)	4.95
5	2005	European Union (28 countries)	4.92
6	2006	European Union (28 countries)	4.91

To remove NaN values, instead of the generic drop function, we can use the specificdropna() function. If we want to erase any row that contains an NaN value, we have to set the how keyword to any. To restrict it to a subset of columns, we can specify it using the subset keyword. As we can see below, the result will be the same as using the dropfunction:

In [20]:

```
eduDrop = edu.dropna(how = 'any', subset = ["Value"])
eduDrop.head()
```

Out[20]:

	TIME	GEO	Value
2	2002	European Union (28 countries)	5.00
3	2003	European Union (28 countries)	5.03
4	2004	European Union (28 countries)	4.95
5	2005	European Union (28 countries)	4.92
6	2006	European Union (28 countries)	4.91

If, instead of removing the rows containing NaN, we want to fill them with another value, then we can use the fillna() method, specifying which value has to be used. If we want to fill only some specific columns, we have to set as argument to the fillna() function a dictionary with the name of the columns as the key and which character to be used for filling as the value.

In [21]:

```
eduFilled = edu.fillna(value = {"Value": 0})
eduFilled.head()
```

Out[21]:

	TIME	GEO	Value
0	2000	European Union (28 countries)	0.00
1	2001	European Union (28 countries)	0.00
2	2002	European Union (28 countries)	5.00
3	2003	European Union (28 countries)	4.95
4	2004	European Union (28 countries)	4.95

Sorting

Another important functionality we will need when inspecting our data is to sort bycolumns. We can sort a DataFrame using any column, using the sortfunction. If we want to see the first five rows of data sorted in descending order (i.e., from the largest to the smallest values) and using the Value column, then we just need to do this:

In [22]:

```
edu.sort_values(by = 'Value', ascending = False,
                inplace = True)
edu.head()
```

Out[22]:

	TIME	GEO	Value
130	2010	Denmark	8.81
131	2011	Denmark	8.75
129	2009	Denmark	8.74
121	2001	Denmark	8.44
122	2002	Denmark	8.44

Note that the inplace keyword means that the DataFrame will be overwritten, and hence no new DataFrame is returned. If instead of ascending = False we use ascending = True, the values are sorted in ascending order (i.e., from thesmallest to the largest values).

If we want to return to the original order, we can sort by an index using the sort_indexfunction and specifying axis=0:

In [23]:

```
edu.sort_index(axis = 0, ascending = True, inplace = True)
edu.head()
```

Out[23]:

	TIME	GEO	Value
0	2000	European Union ...	NaN
1	2001	European Union ...	NaN
2	2002	European Union ...	5.00
3	2003	European Union ...	5.03
4	2004	European Union ...	4.95

Grouping Data

Another very useful way to inspect data is to group it according to some criteria. For instance, in our example it would be nice to group all the data by country, regardlessof the year. Pandas has the groupby function that allows us to do exactly this. The value returned by this function is a special grouped DataFrame. To have a proper DataFrame as a result, it is necessary to apply an aggregation function. Thus, this function will be applied to all the values in the same group.

For example, in our case, if we want a DataFrame showing the mean of the valuesfor each country over all the years, we can obtain it by grouping according to country and using the mean function as the aggregation method for each group. The result would be a DataFrame with countries as indexes and the mean values as

In [24]:

```
group = edu[["GEO", "Value"]].groupby('GEO').mean()
group.head()
```

Out[24]:

	Value
GEO	
Austria	5.618333
Belgium	6.189091
Bulgaria	4.093333
Cyprus	7.023333
Czech Republic	4.16833

Rearranging Data

Up until now, our indexes have been just a numeration of rows without much meaning. We can transform the arrangement of our data, redistributing the indexes and columns for better manipulation of our data, which normally leads to better performance. We can rearrange our data using the pivot_table function. Here, we can specify which columns will be the new indexes, the new values, and the new columns.

For example, imagine that we want to transform our DataFrame to a spreadsheet- like structure with the country names as the index, while the columns will be the years starting from 2006 and the values will be the previous Value column. To do this, first we need to filter out the data and then pivot it in

In [25]:

```
filtered_data = edu[edu["TIME"] > 2005]
pivedu = pd.pivot_table(filtered_data, values = 'Value',
                        index = ['GEO'],
                        columns = ['TIME'])
pivedu.head()
```

Out[25]:

TIME	2006	2007	2008	2009	2010	2011
GEO						
Austria	5.40	5.33	5.47	5.98	5.91	5.80
Belgium	5.98	6.00	6.43	6.57	6.58	6.55
Bulgaria	4.04	3.88	4.44	4.58	4.10	3.82
Cyprus	7.02	6.95	7.45	7.98	7.92	7.87
Czech Republic	4.42	4.05	3.92	4.36	4.25	4.51

Now we can use the new index to select specific rows by label, using the ix operator:

In [26]:

```
pivedu.ix[['Spain','Portugal'], [2006,2011]]
```

Out[26]:

TIME	2006	2011
GEO		
Spain	4.26	4.82
Portugal	5.07	5.27

Pivot also offers the option of providing an argument aggr_function thatallows us to perform an aggregation function between the values if there is more than one value for the given row and column after the transformation. As usual, you can design any custom function you want, just giving its name or using a λ-function.

Ranking Data

Another useful visualization feature is to rank data. For example, we would like to know how each country is ranked by year. To see this, we will use the pandas rank function. But first, we need to clean up our previous pivoted table a bit so that it only has real countries with real data. To do this, first we drop the Euro area entries andshorten the Germany name entry, using the renamefunction and then we drop allthe rows containing any NaN, using the dropnafunction.

Now we can perform the ranking using the rank function. Note here that the parameter ascending=False makes the ranking go from the highest values tothe lowest values. The Pandas rank function supports different tie-breaking methods, specified with the method parameter. In our case, we use the first method, in which ranks are assigned in the order they appear in the array, avoiding gaps between ranking.

In [27]:

```
pivedu = pivedu.drop([
                              'Euro area (13 countries)','Euro area (
                              15 countries)','Euro area (17 countries
                              )','Euro area (18 countries)',
                              'European Union (25 countries)','European
                              Union (27 countries)','European Union (28
                              countries)'
                             ],
                             axis = 0)
pivedu = pivedu.rename(index ={'Germany (until 1990 former territoryof the FRG)': 'Germany'})
pivedu = pivedu.dropna()
pivedu.rank(ascending = False, method = 'first').head()
```

Out[27]:

TIME	2006	2007	2008	2009	2010	2011
GEO						
Austria	10	7	11	7	8	8
Belgium	5	4	3	4	5	5
Bulgaria	21	21	20	20	22	21
Cyprus	2	2	2	2	2	3
Czech Republic	19	20	21	21	20	18

If we want to make a global ranking taking into account all the years, we can sum up all the columns and rank the result. Then we can sort the resulting values toretrieve the top five countries for the last 6 years, in this way:

In [28]:

```
totalSum = pivedu.sum(axis = 1)
totalSum.rank(ascending = False, method = 'dense')
        .sort_values().head()
```

Out[28]:
```
GEO
Denmark          1
Cyprus           2
Finland          3
Malta            4
Belgium          5
dtype: float64
```

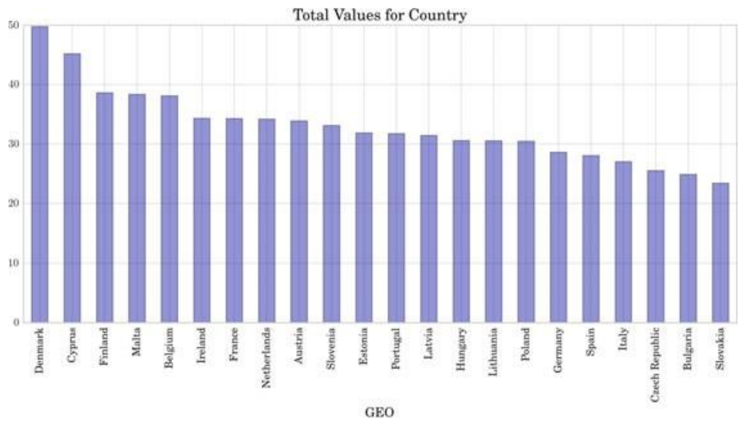
Notice that the method keyword argument in the in the rank function specifies how items that compare equals receive ranking. In the case of dense, items that compare equals receive the same ranking number, and the next not equal item receives the immediately following ranking number.

Plotting

Pandas DataFrames and Series can be plotted using the plot function, which uses the library for graphics Matplotlib. For example, if we want to plot the accumulated values for each country over the last 6 years, we can take the Series obtained in thepprevious example and plot it directly by calling the plot function as shown in the next cell:

In [29]:
```
totalSum = pivedu.sum(axis = 1)
                .sort_values(ascending = False)
totalSum.plot(kind = 'bar', style = 'b', alpha = 0.4,
             title = "Total Values for Country")
```

Out[29]:



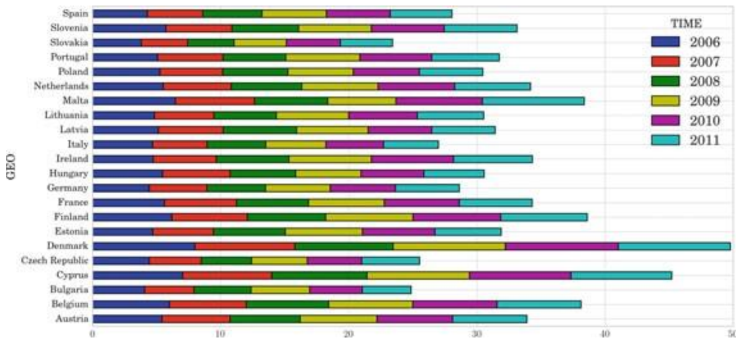
Note that if we want the bars ordered from the highest to the lowest value, we need to sort the values in the Series first. The parameter kind used in the plot function defines which kind of graphic will be used. In our case, a bar graph. The parameter stylerefers to the style properties of the graphic, in our case, the color of bars is set to b (blue). The alpha channel can be modified adding a keyword parameter alpha with a percentage, producing a more translucent plot. Finally,using the titlekeyword the name of the graphic can be set.

It is also possible to plot a DataFrame directly. In this case, each column is treated as a separated Series. For example, instead of printing the accumulated value over the years, we can plot the value for each year.

In [30]:

```
my_colors = ['b', 'r', 'g', 'y', 'm', 'c']
ax = pivedu.plot(kind = 'barh',
                 stacked = True,
                 color = my_colors)
ax.legend(loc = 'center left', bbox_to_anchor = (1, .5))
```

Out[30]:



In this case, we have used a horizontal bar graph (kind='barh') stacking all the years in the same country bar. This can be done by setting the parameter stacked to True. The number of default colors in a plot is only 5, thus if you have more than 5 Series to show, you need to specify more colors or otherwise the same set ofcolors will be used again. We can set a new set of colors using the keyword color with a list of colors. Basic colors have a single-character code assigned to each, for example, “b” is for blue, “r” for red, “g” for green, “y” for yellow, “m” for magenta, and “c” for cyan. When several Series are shown in a plot, a legend is created for identifying each one. The name for each Series is the name of the column in the DataFrame. By default, the legend goes inside the plot area. If we want to change this, we can use the legend function of the axis object (this is the object returned when the plot function is called). By using the loc keyword, we can set the relative position of the legend with respect to the plot. It can be a combination of right or left and upper, lower, or center. With bbox_to_anchor we can set an absolute position with respect to the plot, allowing us to put the legend outside the graph.

UNIT-2

Descriptive statistics, data preparation. Exploratory Data Analysis data summarization, data distribution, measuring asymmetry. Sample and estimated mean, variance and standard score. Statistical Inference frequency approach, variability of estimates, hypothesis testing using confidence intervals, using p-values

DescriptiveStatistics

Descriptive statistics helps to simplify large amounts of data in a sensible way. In contrast to inferential statistics, which will be introduced in a later chapter, in descriptive statistics we do not draw conclusions beyond the data we are analyzing;neither do we reach any conclusions regarding hypotheses we may make. We do nottry to infer characteristics of the "population" (see below) of the data, but claim to present quantitative descriptions of it in a manageable form. It is simply a way to describe the data.

Statistics, and in particular descriptive statistics, is based on two main concepts:

- a *population* is a collection of objects, items ("units") about which information issought;
- a *sample* is a part of the population that is observed.

Descriptive statistics applies the concepts, measures, and terms that are used to describe the basic features of the samples in a study. These procedures are essential to provide summaries about the samples as an approximation of the population. Together with simple graphics, they form the basis of every quantitative analysis ofdata. In order to describe the sample data and to be able to infer any conclusion, weshould go through several steps:

1. *Data preparation*: Given a specific example, we need to prepare the data forgenerating statistically valid descriptions.
2. *Descriptive statistics*: This generates different statistics to describe and summa-rize the data concisely and evaluate different ways to visualize them.

Data Preparation

One of the first tasks when analyzing data is to collect and prepare the data in a format appropriate for analysis of the samples. The most common steps for data preparationinvolve the following operations.

1. *Obtaining the data:* Data can be read directly from a file or they might be obtained by scraping the web.
2. *Parsing the data:* The right parsing procedure depends on what format the dataare in: plain text, fixed columns, CSV, XML, HTML, etc.
3. *Cleaning the data:* Survey responses and other data files are almost always in-complete. Sometimes, there are multiple codes for things such as, not asked, didnot know, and declined to answer. And there are almost always errors. A simplestrategy is to remove or ignore incomplete records.
4. *Building data structures:* Once you read the data, it is necessary to store them ina data structure that lends itself to the analysis we are interested in. If the data fitinto the memory, building a data structure is usually the way to go. If not, usuallya database is built, which is an out-of-memory data structure. Most databases provide a mapping from keys to values, so they serve as dictionaries.

The Adult Example

Let us consider a public database called the "Adult" dataset, hosted on the UCI's Machine Learning Repository.[1] It contains approximately 32,000 observations con-cerning different financial parameters related to the US population: age, sex, marital(marital status of the individual), country, income (Boolean variable: whether the per- son makes more than $50,000 per annum), education (the highest level of educationachieved by the individual), occupation, capital gain, etc.

We will show that we can explore the data by asking questions like: "Are men more likely to become high-income professionals than women, i.e., to receive an income of over $50,000 per annum?"

First, let us read the data:

In [1]:

```
file = open('files/ch03/adult.data', 'r')
def chr_int(a):
    if a.isdigit(): return int(a)
    else: return 0

data = []
for line in file:
     data1 = line.split(', ')
     if len(data1) == 15:
        data.append([chr_int(data1[0]), data1[1],
                     chr_int(data1[2]), data1[3],
                     chr_int(data1[4]), data1[5],
                     data1[6], data1[7], data1[8],
                     data1[9], chr_int(data1[10]),
                     chr_int(data1[11]),
                     chr_int(data1[12]),
                     data1[13], data1[14]
                    ])
```

Checking the data, we obtain:

In [2]:

```
print data[1:2]
```

Out[2]: [[50, ’Self-emp-not-inc’, 83311, ’Bachelors’, 13,
’Married-civ-spouse’, ’Exec-managerial’, ’Husband’, ’White’,’Male’, 0, 0, 13, ’United-States’, ′ <= 50*K*’]]

One of the easiest ways to manage data in Python is by using the DataFrame structure, defined in the *Pandas* library, which is a two-dimensional, size-mutable,potentially heterogeneous tabular data structure with labeled axes:

In [3]:

```
df = pd.DataFrame(data)
df.columns = [
    'age', 'type_employer', 'fnlwgt',
    'education', 'education_num', 'marital',
    'occupation',' relationship', 'race',
    'sex', 'capital_gain', 'capital_loss',
    'hr_per_week', 'country', 'income'
    ]
```

The command shapegives exactly the number of data samples (in rows, in this case) and features (in columns):

In [4]:

```
df.shape
```

Out[4]: (32561, 15)

Thus, we can see that our dataset contains 32,561 data records with 15 featureseach. Let us count the number of items per country:

In[5]:

```
counts = df.groupby('country').size()
print counts.head()
```

Out[5]:

```
country
? 583
Cambodia 19
Vietnam 67
Yugoslavia 16
```

The first row shows the number of samples with unknown country, followed bythe number of samples corresponding to the first countries in the dataset.

Let us split people according to their gender into two groups: men and women.

In[6]:

```
ml = df[(df.sex == 'Male')]
```

If we focus on high-income professionals separated by sex, we can do:

In[7]:

```
ml1 = df[(df.sex == 'Male') & (df.income=='>50K\n')
    ]
fm = df[(df.sex == 'Female')]
fm1 = df[(df.sex == 'Female') & (df.income=='>50K\n
    ')]
```

Exploratory Data Analysis

The data that come from performing a particular measurement on all the subjects in a sample represent our observations for a single characteristic like country, age, education, etc. These measurements and categories represent a *sample distribution* of the variable, which in turn approximately represents the *population distribution* of the variable. One of the main goals of exploratory data analysis is to visualize and summarize the sample distribution, thereby allowing us to make tentative assumptions about the population distribution.

Summarizing the Data

The data in general can be categorical or quantitative. For categorical data, a simple tabulation of the frequency of each category is the best non-graphical exploration for data analysis. For example, we can ask ourselves what is the proportion of high-income professionals in our database:

In[8]:

```
df1 = df[(df.income=='>50K\n')]
print 'The rate of people with high income is: ',
      int(len(df1)/float(len(df))*100), '%.'
print 'The rate of men with high income is: ',
      int(len(ml1)/float(len(ml))*100), '%.'
print 'The rate of women with high income is: ',
      int(len(fm1)/float(len(fm))*100), '%.'
```

Out[8]:

```
The rate of people with high income is: 24 %.
The rate of men with high income is: 30 %. The rate of women
with high income is: 10 %.
```

Given a quantitative variable, exploratory data analysis is a way to make prelim-inary assessments about the population distribution of the variable using the data of the observed samples. The characteristics of the population distribution of a quanti- tative variable are its *mean*, *deviation*, *histograms*, *outliers*, etc. Our observed datarepresent just a finite set of samples of an often infinite number of possible samples.The characteristics of our randomly observed samples are interesting only to the degree that they represent the population of the data they came from.

Mean

One of the first measurements we use to have a look at the data is to obtain *samplestatistics* from the data, such as the sample mean [1]. Given a sample of n values, $\{x_i\}, i = 1, \ldots, n$, the *mean*, μ, is the sum of the values divided by the number of values,[2] in other words:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i . \tag{3.1}$$

The terms mean and *average* are often used interchangeably. In fact, the maindistinction between them is that the mean of a sample is the summary statistic com-puted by Eq. (3.1), while an average is not strictly defined and could be one of manysummary statistics that can be chosen to describe the central tendency of a sample.

In our case, we can consider what the average age of men and women samples inour dataset would be in terms of their mean:

Descriptive Statistics

In[9]:

```
print 'The average age of men is: ',
      ml['age'].mean()
print ' The average age  of women is: ',
      fm['age'].mean()

print ' The average age  of high-income men is: ',
      ml1['age'].mean()
print ' The average age of high - income women is : ',
      fm1['age'].mean()
```

Out[9]:

```
The average age of men is: 39.4335474989 The average age of
women is: 36.8582304336
The average age of high-income men is: 44.6257880516
The average age of high-income women is: 42.1255301103
```

This difference in the sample means can be considered initial evidence that thereare differences between men and women with high income!

Comment: Later, we will work with both concepts: the population mean and thesample mean. We should not confuse them! The first is the mean of samples takenfrom the population; the second, the mean of the whole population.

Sample Variance

The mean is not usually a sufficient descriptor of the data. We can go further by knowing two numbers: mean and *variance*. The variance σ^2 describes the spread ofthe data and it is defined as follows:

$$\sigma^2 = \frac{1}{n}\sum_{i} (x_i - \mu)^2. \tag{3.2}$$

The term $(x_i \ \mu)$ is called the *deviation* from the mean, so the variance is the mean squared deviation. The square root of the variance, σ, is called the *standard deviation*. We consider the standard deviation, because the variance is hard to interpret (e.g., ifthe units are grams, the variance is in grams squared).

Let us compute the mean and the variance of hours per week men and women inour dataset work:

In[10]:

```
ml_mu  =  ml['age'].mean()
fm_mu  =  fm['age'].mean()
ml_var  =  ml['age'].var()
fm_var  =  fm['age'].var()
ml_std  =  ml['age'].std()
fm_std  =  fm['age'].std()
print 'Statistics of age for men: mu:',
```

Out[10]:
```
Statistics of age for men: mu: 39.4335474989 var: 178.773751745std: 13.3706301925
Statistics of age for women: mu: 36.8582304336 var:196.383706395 std:
14.0136970994
```

We can see that the mean number of hours worked per week by women is significantly lesser than that worked by men, but with much higher variance and standarddeviation.

Sample Median

The mean of the samples is a good descriptor, but it has an important drawback: what will happen if in the sample set there is an error with a value very different from the rest? For example, considering hours worked per week, it would normally be in a range between 20 and 80; but what would happen if by mistake there was a value of 1000? An item of data that is significantly different from the rest of the data is called an *outlier*. In this case, the mean, μ, will be drastically changed towards the outlier. One solution to this drawback is offered by the statistical *median*, μ_{12}, which is an order statistic giving the middle value of a sample. In this case, all the values are ordered by their magnitude and the median is defined as the value that is in themiddle of the ordered list. Hence, it is a value that is much more robust in the face of outliers.

Let us see, the median age of working men and women in our dataset and the median age of high-income men and women:

In[11]:
```
ml_median = ml['age'].median()
fm_median = fm['age'].median()
print "Median age per men and women: ",
      ml_median, fm_median

ml_median_age = ml1['age'].median()
fm_median_age = fm1['age'].median()
print "Median age per men and women with high-
    income: ",
      ml_median_age, fm_median_age
```

Out[11]:
```
Median age per men and women: 38.0 35.0
Median age per men and women with high-income: 44.0 41.0
```

As expected, the median age of high-income people is higher than the whole setof working people, although the difference between men and women in both sets isthe same.

Quantiles and Percentiles

Sometimes we are interested in observing how sample data are distributed in general. In this case, we can order the samples $\{x_i\}$, then find the x_p so that it divides the datainto two parts, where:

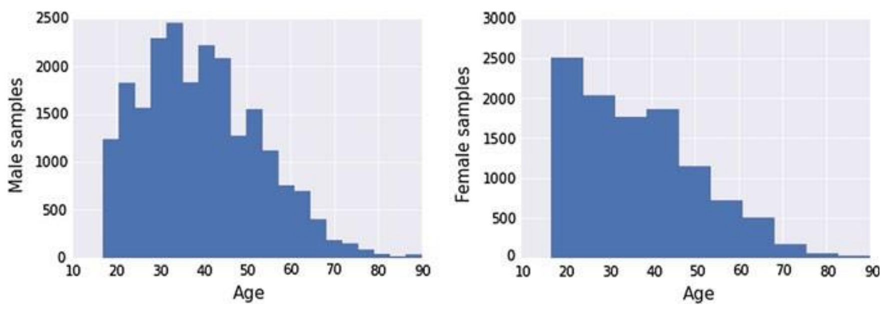


Fig. 3.1 Histogram of the age of working men (*left*) and women (*right*)

- a fraction p of the data values is less than or equal to x_p and
- the remaining fraction $(1 - p)$ is greater than x_p.

That value, x_p, is the *p-th* quantile, or the 100 *p*✗*-th* percentile. For example, a 5-number summary is defined by the values x_{min}, Q_1, Q_2, Q_3, x_{max} , where Q_1 is the 25 *p-th*✗percentile, Q_2 is the 50 *p-th* pe✗rcentile and Q_3 is the 75 *p-th* perce✗ntile.

Data Distributions

Summarizing data by just looking at their mean, median, and variance can be dangerous: very different data can be described by the same statistics. The best thing to do is to validate the data by inspecting them. We can have a look at the data distribution, which describes how often each value appears (i.e., what is its frequency).

The most common representation of a distribution is a *histogram*, which is a graph that shows the frequency of each value. Let us show the age of working men and women separately.

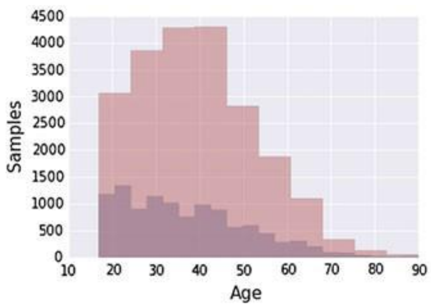
In[12]:

```
ml_age = ml['age']
ml_age.hist(normed = 0, histtype = 'stepfilled',
    bins = 20)
```

In[13]:

```
fm_age = fm['age']
fm_age.hist(normed = 0, histtype = 'stepfilled',
    bins = 10)
```

The output can be seen in Fig. 3.1. If we want to compare the histograms, we canplot them overlapping in the same graphic as follows:



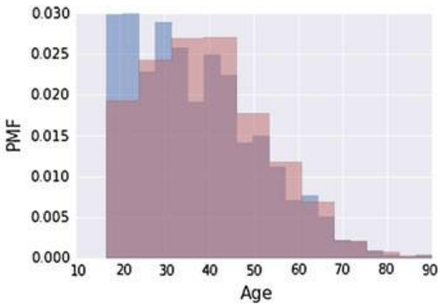


Fig. 3.2 Histogram of the age of working men (in *ochre*) and women (in *violet*) (*left*). Histogram of the age of working men (in *ochre*), women (in *blue*), and their intersection (in *violet*) after samples normalization (*right*)

In[14]:
```
import seaborn as sns
fm_age.hist(normed = 0, histtype = 'stepfilled',
            alpha = .5, bins = 20)
ml_age.hist(normed = 0, histtype = 'stepfilled',
            alpha = .5,
            color = sns.desaturate("indianred",
                .75),
            bins = 10)
```

The output can be seen in Fig. 3.2 (left). Note that we are visualizing the absolute values of the number of people in our dataset according to their age (the abscissa ofthe histogram). As a side effect, we can see that there are many more men in these conditions than women.

We can normalize the frequencies of the histogram by dividing/normalizing by *n*, the number of samples. The normalized histogram is called the *Probability MassFunction* (PMF).

In[15]:
```
fm_age.hist(normed = 1, histtype = 'stepfilled',
            alpha = .5, bins = 20)
ml_age.hist(normed = 1, histtype = 'stepfilled',
            alpha = .5, bins = 10,
            color = sns.desaturate("indianred",
                .75))
```

This outputs Fig. 3.2 (right), where we can observe a comparable range of individuals (men and women).

The *Cumulative Distribution Function* (CDF), or just distribution function, describes the probability that a real-valued random variable X with a given probability distribution will be found to have a value less than or equal to x. Let us show the CDF of age distribution for both men and women.

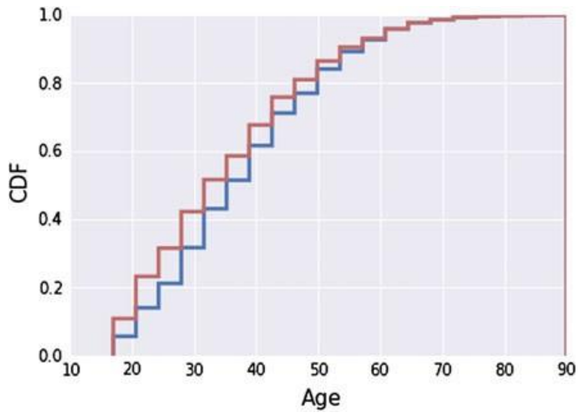


Fig. 3.3 The CDF of the ageof working male (in *blue*) and female (in *red*) samples

In[16]:

```
ml_age.hist(normed = 1, histtype = 'step',
            cumulative = True,  linewidth = 3.5,
            bins = 20)
fm_age.hist(normed = 1, histtype='step',
            cumulative = True,  linewidth = 3.5,
            bins = 20,
            color = sns.desaturate("indianred",
                .75))
```

The output can be seen in Fig. 3.3, which illustrates the CDF of the age distributions for both men and women.

Outlier Treatment

As mentioned before, outliers are data samples with a value that is far from the centraltendency. Different rules can be defined to detect outliers, as follows:

- Computing samples that are far from the median.
- Computing samples whose values exceed the mean by 2 or 3 standard deviations.

For example, in our case, we are interested in the age statistics of men versus women with high incomes and we can see that in our dataset, the minimum age is 17years and the maximum is 90 years. We can consider that some of these samples are due to errors or are not representable. Applying the domain knowledge, we focus onthe median age (37, in our case) up to 72 and down to 22 years old, and we considerthe rest as outliers.

In[17]:
```
df2 = df.drop(df.index[
      (df.income == '>50K\n') &
      (df['age'] > df['age'].median() + 35) &
      (df['age'] > df['age'].median() -15)
      ])
ml1_age = ml1['age']
fm1_age = fm1['age']

ml2_age = ml1_age.drop(ml1_age.index[
      (ml1_age > df['age'].median() + 35) &
      (ml1_age > df['age'].median() - 15)
      ])
fm2_age = fm1_age.drop(fm1_age.index[
      (fm1_age > df['age'].median() + 35) &
      (fm1_age > df['age'].median() - 15)
      ])
```

We can check how the mean and the median changed once the data were cleaned:

In[18]:
```
mu2ml = ml2_age.mean()
std2ml = ml2_age.std()
md2ml = ml2_age.median()
mu2fm = fm2_age.mean()
std2fm = fm2_age.std()
md2fm = fm2_age.median()

print "Men statistics:"
print "Mean:", mu2ml, "Std:", std2ml
print "Median:", md2ml
print "Min:", ml2_age.min(), "Max:", ml2_age.max()

print "Women statistics:"
print "Mean:", mu2fm, "Std:", std2fm
print "Median:", md2fm
print "Min:", fm2_age.min(), "Max:", fm2_age.max()
```

Out[18]:
```
Men statistics: Mean: 44.3179821239 Std: 10.0197498572 Median:
44.0 Min: 19 Max: 72
Women statistics: Mean: 41.877028181 Std: 10.0364418073 Median:
41.0 Min: 19 Max: 72
```

Let us visualize how many outliers are removed from the whole data by:

In[19]:
```
plt.figure(figsize = (13.4, 5))
df.age[(df.income == '>50K\n')]
      .plot(alpha = .25, color = 'blue')
df2.age[(df2.income == '>50K\n')]
      .plot(alpha = .45, color = 'red')
```

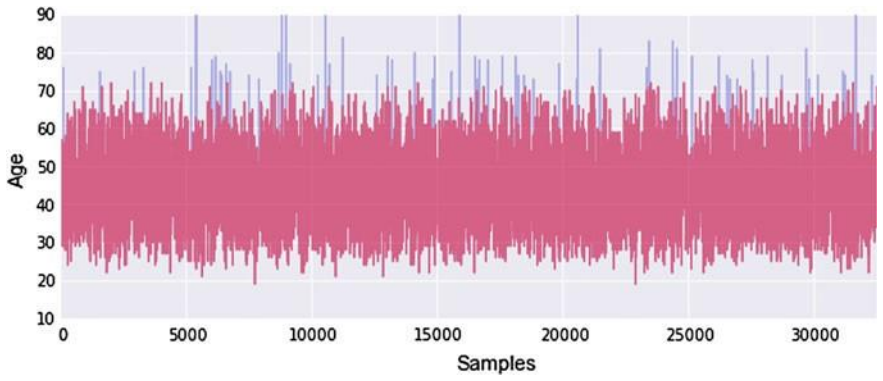


Fig. 3.4 The *red* shows the cleaned data without the considered outliers (in *blue*)

Figure 3.4 shows the outliers in blue and the rest of the data in red. Visually, wecan confirm that we removed mainly outliers from the dataset.

Next we can see that by removing the outliers, the difference between the popula-tions (men and women) actually decreased. In our case, there were more outliers inmen than women. If the difference in the mean values before removing the outliersis 2.5, after removing them it slightly decreased to 2.44:

In[20]:
```
print 'The mean difference with outliers is: %4.2f.
    '
       % (ml_age.mean() - fm_age.mean())
print 'The mean difference without outliers is:
    %4.2f.'
       % (ml2_age.mean() - fm2_age.mean())
```

Out[20]:
```
The mean difference with outliers is: 2.58.
The mean difference without outliers is: 2.44.
```

Let us observe the difference of men and women incomes in the cleaned subsetwith some more details.

In[21]:
```
countx, divisionx = np.histogram(ml2_age, normed =
    True)
county, divisiony = np.histogram(fm2_age, normed =
    True)

val = [(divisionx[i] + divisionx[i+1])/2
        for i in range(len(divisionx) - 1)]
plt.plot(val, countx - county, 'o-')
```

The results are shown in Fig. 3.5. One can see that the differences between maleand female values are slightly negative before age 42 and positive after it. Hence, women tend to be promoted (receive more than 50 K) earlier than men.

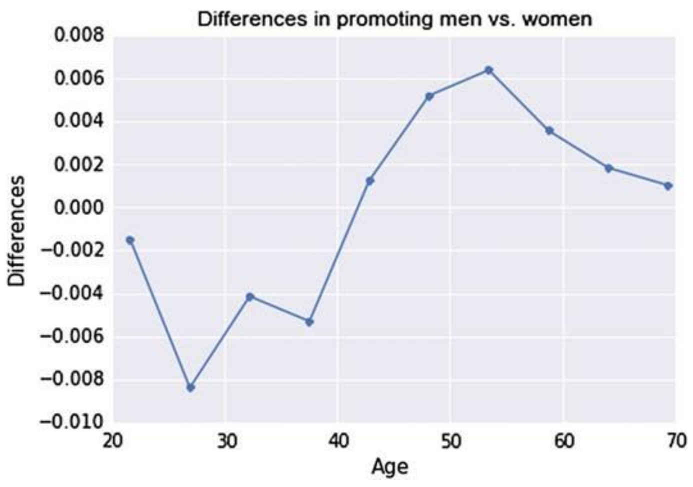


Fig. 3.5 Differences in high-income earner men versus women as a function of age

Measuring Asymmetry: Skewness and Pearson's Median Skewness Coefficient

For univariate data, the formula for *skewness* is a statistic that measures the asym-metry of the set of *n* data samples, x_i:

$$g_1 = \frac{1}{n}\frac{\sum_i (x_i - \mu^3)}{\sigma^3}, \tag{3.3}$$

where μ is the mean, σ is the standard deviation, and n is the number of data points.

Negative deviation indicates that the distribution "skews left" (it extends further to the left than to the right). One can easily see that the skewness for a normal distribution is zero, and any symmetric data must have a skewness of zero. Note that skewness can be affected by outliers! A simpler alternative is to look at the relationship between the mean μ and the median μ_{12}.

In[22]:

```
def skewness(x):
    res = 0
    m = x.mean()
    s = x.std()
    for i in x:
        res += (i-m) * (i-m) * (i-m)
    res /= (len(x) * s * s * s)
    return res

print "Skewness of the male population = ",
      skewness(ml2_age)
print "Skewness of the female population is = ",
      skewness(fm2_age)
```

Out[22]:
```
Skewness of the male population = 0.266444383843 Skewness of the female
population = 0.386333524913
```

That is, the female population is more skewed than the male, probably since mencould be most prone to retire later than women.

The **Pearson's median skewness coefficient** is a more robust alternative to the skewness coefficient and is defined as follows:

$$g_p = 3(\mu - \mu_{12})\sigma.$$

There are many other definitions for skewness that will not be discussed here. In our case, if we check the Pearson's skewness coefficient for both men and women,we can see that the difference between them actually increases:

In[23]:
```
def pearson(x):
    return 3*(x.mean() - x.median())*x.std()

print "Pearson's coefficient of the male population
    = ",
      pearson(ml2_age)
print "Pearson's coefficient of the female
    population = ",
      pearson(fm2_age)
```

Out[23]:
```
Pearson's coefficient of the male population = 9.55830402221 Pearson's coefficient of the
female population = 26.4067269073
```

Continuous Distribution

The distributions we have considered up to now are based on empirical observationsand thus are called *empirical distributions*. As an alternative, we may be interested in considering distributions that are defined by a continuous function and are called*continuous distributions* [2]. Remember that we defined the PMF, $f_X(x)$, of a discreterandom variable X as $f_X(x) = P(X = x)$ for all x. In the case of a continuous random variable X, we speak of the *Probability Density Function* (PDF), which

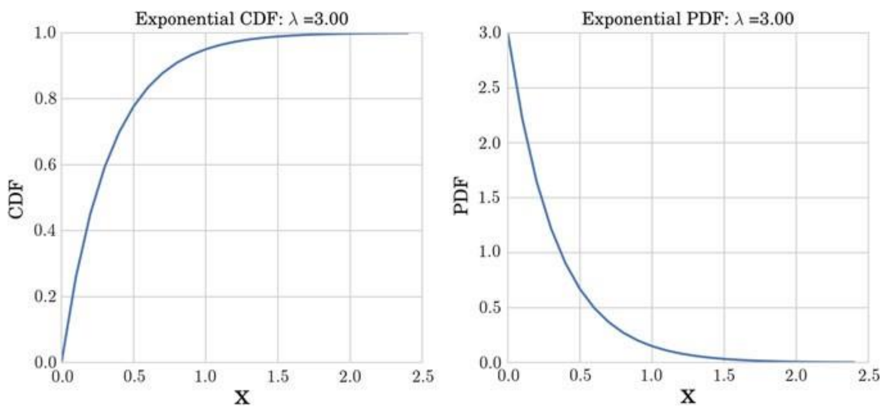


Fig. 3.6 Exponential CDF (*left*) and PDF (*right*) with λ = 3.00

is defined as $F_X(x)$ where this satisfies: $F_X(x) = \int_{\infty}^{x} f_X(t)\delta t$ for all x. There are many continuous distributions; here, we will consider the most common ones: the exponential and the normal distributions.

The Exponential Distribution

Exponential distributions are well known since they describe the inter-arrival time between events. When the events are equally likely to occur at any time, the distri-bution of the inter-arrival time tends to an exponential distribution. The CDF and the PDF of the exponential distribution are defined by the following equations:

$$CDF(x) = 1 - e^{-\lambda x}, \qquad PDF(x) = \lambda e^{-\lambda x}.$$

The parameter λ defines the shape of the distribution. An example is given inFig. 3.6. It is easy to show that the mean of the distribution is ^{1}, the variance is$\frac{}{\lambda^2}$ and the median is $\frac{\ln}{\lambda}$

Note that for a small number of samples, it is difficult to see that the exact empiricaldistribution fits a continuous distribution. The best way to observe this match is to generate samples from the continuous distribution and see if these samples match the data. As an exercise, you can consider the birthdays of a large enough group of people, sorting them and computing the inter-arrival time in days. If you plot the CDF of the inter-arrival times, you will observe the exponential distribution.

There are a lot of real-world events that can be described with this distribution, including the time until a radioactive particle decays; the time it takes before your next telephone call; and the time until default (on payment to company debt holders)in reduced-form credit risk modeling. The random variable X of the lifetime of some
batteries is associated with a probability density function of the form: $PDF(x) = \frac{1}{4}e^{-\frac{x}{4}}e^{-\frac{(x-\mu)^2}{2}}$

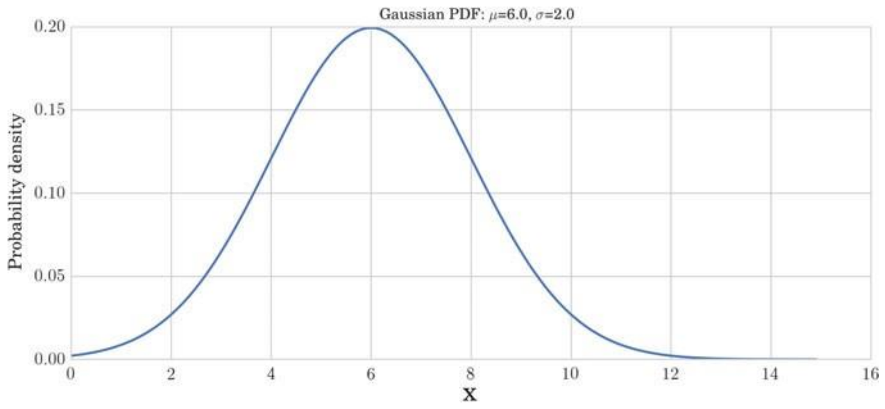


Fig. 3.7 Normal PDF with $\mu = 6$ and $\sigma = 2$

The Normal Distribution

The *normal distribution*, also called the *Gaussian distribution*, is the most common since it represents many real phenomena: economic, natural, social, and others. Some well-known examples of real phenomena with a normal distribution are as follows:

- The size of living tissue (length, height, weight).
- The length of inert appendages (hair, nails, teeth) of biological specimens.
- Different physiological measurements (e.g., blood pressure), etc.

The normal CDF has no closed-form expression and its most common represen-tation is the PDF:

$$\mathrm{PDF}(x) = \sqrt{\frac{1}{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

The parameter σ defines the shape of the distribution. An example of the PDF ofa normal distribution with $\mu = 6$ and $\sigma = 2$ is given in Fig. 3.7.

Kernel Density

In many real problems, we may not be interested in the parameters of a particular distribution of data, but just a continuous representation of the data. In this case, we should estimate the distribution non-parametrically (i.e., making no assumptionsabout the form of the underlying distribution) using kernel density estimation. Let us imagine that we have a set of data measurements without knowing their distribution and we need to estimate the continuous representation of their distribution. In this case, we can consider a Gaussian kernel to generate the density around the data. Letus consider a set of random data generated by a bimodal normal distribution. If we consider a Gaussian kernel around the data, the sum of those kernels can give us

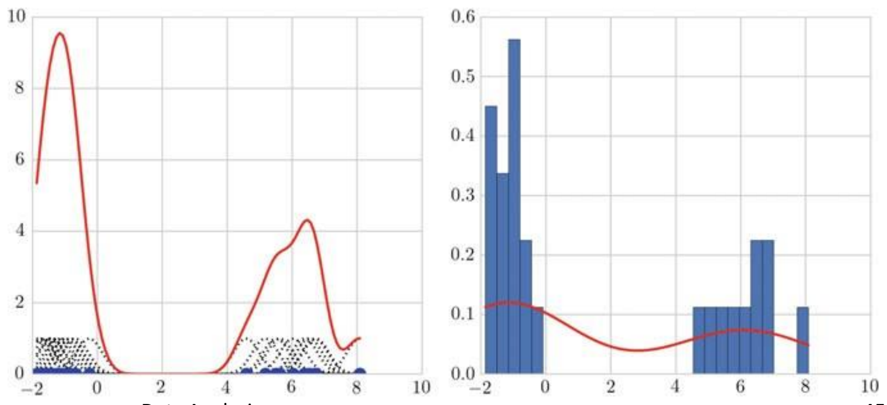


Fig. 3.8 Summed kernel functions around a random set of points (*left*) and the kernel density estimate with the optimal bandwidth (*right*) for our dataset. Random data shown in *blue*, kernel shown in *black* and summed function shown in *red*

a continuous function that when normalized would approximate the density of the distribution:

In[24]:

```
x1 = np.random.normal(-1, 0.5, 15)
x2 = np.random.normal(6, 1, 10)
y = np.r_[x1, x2] # r_ translates slice objects to
    concatenation along the first axis.
x = np.linspace(min(y), max(y), 100)

s = 0.4 # Smoothing parameter

# Calculate the kernels
kernels = np.transpose([norm.pdf(x, yi, s) for yi
    in y])
plt.plot(x, kernels, 'k:')
plt.plot(x, kernels.sum(1), 'r')
plt.plot(y, np.zeros(len(y)), 'bo', ms = 10)
```

Figure 3.8 (left) shows the result of the construction of the continuous functionfrom the kernel summarization.

In fact, the library SciPy[3] implements a Gaussian kernel density estimation that automatically chooses the appropriate bandwidth parameter for the kernel. Thus, thefinal construction of the density estimate will be obtained by:

.

In [25]:
```
from scipy.stats import kde
density = kde.gaussian_kde(y)
xgrid = np.linspace(x.min(), x.max(), 200)
plt.hist(y, bins = 28, normed = True)
plt.plot(xgrid, density(xgrid), 'r-')
```

Figure 3.8 (right) shows the result of the kernel density estimate for our example.

Estimation

An important aspect when working with statistical data is being able to use estimatesto approximate the values of unknown parameters of the dataset. In this section, wewill review different kinds of estimators (estimated mean, variance, standard score,etc.).

Sample and Estimated Mean, Variance and Standard Scores

In continuation, we will deal with point estimators that are single numerical estimates of parameters of a population.

Mean

Let us assume that we know that our data are coming from a normal distribution andthe random samples drawn are as follows:

$$\{0.33, -1.76, 2.34, 0.56, 0.89\}.$$

The question is can we guess the mean μ of the distribution? One approximation isgiven by the sample mean,$\bar{x}$. This process is called *estimation* and the statistic (e.g., the sample mean) is called an *estimator*. In our case, the sample mean is 0.472, and it seems a logical choice to represent the mean of the distribution. It is not so evident ifwe add a sample with ~~a~~ value of 465. In this case, the sample mea~~n~~ will be 77.11, which does not look like the mean of the distribution. The reason is due to the fact that the last value seems to be an outlier compared to the rest of the sample. In orderto avoid this effect, we can try first to remove outliers and then to estimate the mean; or we can use the sample median as an estimator of the mean of the distribution. If there are no outliers, the sample mean x minimizes the following *mean squared error*:

$$MSE = \frac{1}{n} \sum (x - \mu)^2,$$

where n is the number of times we estimate the mean.Let us compute the MSE of a set of random data:

In[26]:
```
NTs = 200
mu = 0.0
var = 1.0
err = 0.0
NPs = 1000
for i in range(NTs):
    x = np.random.normal(mu, var, NPs)
    err += (x.mean()-mu)**2
print 'MSE: ', err/NTests
```

Out[26]:
```
MSE: 0.00019879541147
```

Variance

If we ask ourselves what is the variance, σ^2, of the distribution of X, analogously we can use the sample variance as an estimator. Let us denote by $\bar{\sigma}^2$ the sample varianceestimator:

$$\bar{\sigma}^2 = \frac{1}{n} \sum_i (x_i - \bar{x})^2.$$

For large samples, this estimator works well, but for a small number of samplesit is biased. In those cases, a better estimator is given by:

$$\bar{\sigma}^2 = \frac{1}{n-1} \sum_i (x_i - \bar{x})^2.$$

Standard Score

In many real problems, when we want to compare data, or estimate their correlations or some other kind of relations, we must avoid data that come in different units. For example, weight can come in kilograms or grams. Even data that come in the same units can still belong to different distributions. We need to normalize them tostandard scores. Given a dataset as a series of values, $\{x_i\}$, we convert the data to standard scores by subtracting the mean and dividing them by the standard deviation:

$$z_i = \frac{(x_i - \mu)}{\sigma}.$$

Note that this measure is dimensionless and its distribution has a mean of 0 and variance of 1. It inherits the "shape" of the dataset: if X is normally distributed, so is Z; if X is skewed, so is Z.

Covariance, and Pearson's and Spearman's Rank Correlation

Variables of data can express relations. For example, countries that tend to invest in research also tend to invest more in education and health. This kind of relationshipis captured by the covariance.

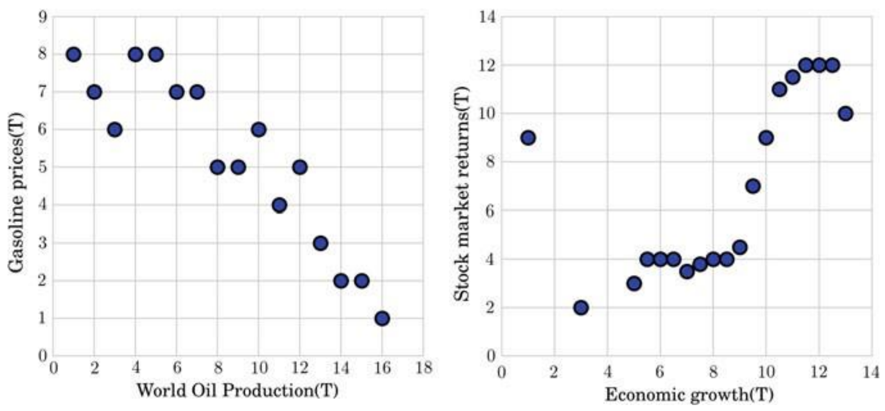


Fig. 3.9 Positive correlation between economic growth and stock market returns worldwide (*left*). Negative correlation between the world oil production and gasoline prices worldwide (*right*)

Covariance

When two variables share the same tendency, we speak about *covariance*. Let us consider two series, $\{x_i\}$ and $\{y_i\}$. Let us center the data with respect to their mean: $dx_i = x_i - \mu_X$ and $dy_i = y_i - \mu_Y$. It is easy to show that when $\{x_i\}$ and $\{y_i\}$ vary together, their deviations tend to have the same sign. The covariance is defined as the mean of the following products:

$$Cov(X, Y) = \frac{1}{n} \sum_{i=1}^{n} dx_i \, dy_i ,$$

where n is the length of both sets. Still, the covariance itself is hard to interpret.

Correlation and the Pearson’s Correlation

If we normalize the data with respect to their deviation, that leads to the standardscores; and then multiplying them, we get:

$$\rho_i = \frac{x_i - \mu_X}{\sigma_X} \frac{y_i - \mu_Y}{\sigma_Y} .$$

The mean of this product is $\rho = \frac{1}{n} \sum_{i=1}^{n} \rho_i$. Equivalently, we can rewrite ρ in terms of the covariance, and thus obtain the *Pearson’s correlation*:

$$\rho = \frac{Cov(X, Y)}{\sigma_X \sigma_Y} .$$

Note that the Pearson’s correlation is always between -1 and $+1$, where themagnitude depends on the degree of correlation. If the Pearson’s correlation is -1 (or1), it means that the variables are perfectly correlated (positively or negatively) (see Fig. 3.9). This means that one variable can predict the other very well. However,

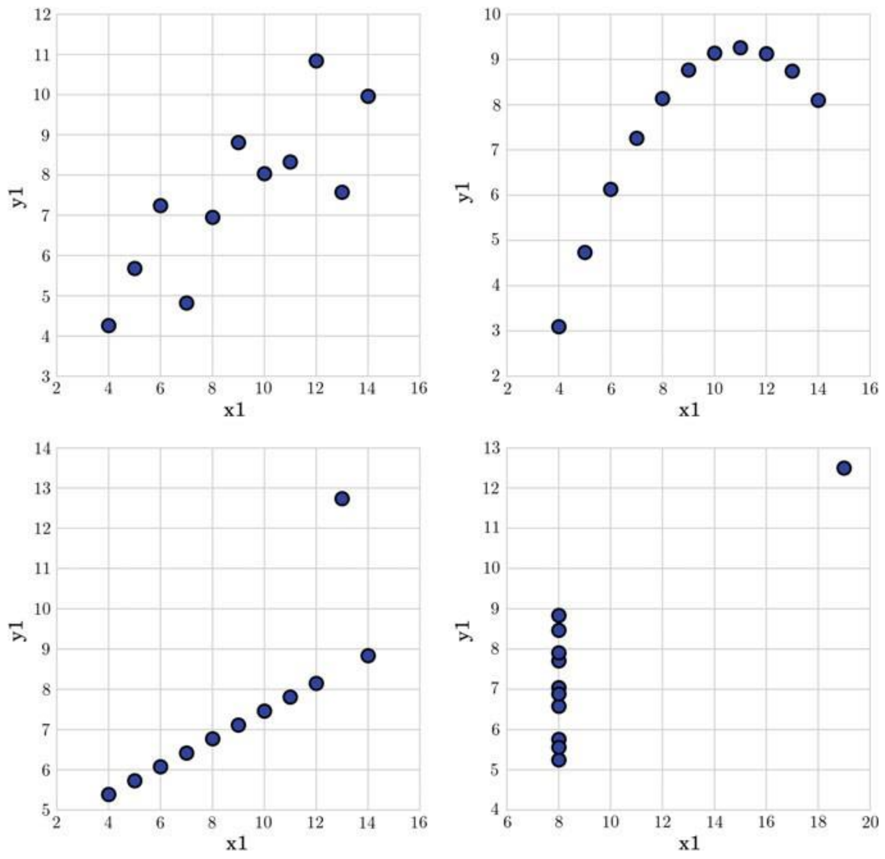


Fig. 3.10 Anscombe configurations

having ρ 0, does not necessarily mean that the variables are not correlated! Pearson's correlation captures correlations of first order, but not nonlinear correlations.Moreover, it does not work well in the presence of outliers.

Spearman's Rank Correlation

The *Spearman's rank correlation* comes as a solution to the robustness problem ofPearson's correlation when the data contain outliers. The main idea is to use the ranks of the sorted sample data, instead of the values themselves. For example, in the list [4, 3, 7, 5], the rank of 4 is 2, since it will appear second in the ordered list ([3, 4, 5, 7]). Spearman's correlation computes the correlation between the ranks
of the data. For example, considering the data: X= [10, 20, 30, 40, 1000], and Y = [70, 1000, 50, 10, 20], where we have an outlier in each one set. If we compute the ranks, they are [1.0, 2.0, 3.0, 4.0, 5.0] and [2.0, 1.0, 3.0, 5.0, 4.0]. As value of the Pearson's coefficient, we get 0.28, which does not show much

between the sets. However, the Spearman's rank coefficient, capturing the correlation between the ranks, gives as a final value of 0.80, confirming the correlation betweenthe sets. As an exercise, you can compute the Pearson's and the Spearman's rank correlations for the different Anscombe configurations given in Fig. 3.10. Observe if linear and nonlinear correlations can be captured by the Pearson's and the Spearman's rank correlations.

Statistical Inference

Introduction

There is not only one way to address the problem of statistical inference. In fact, there are two main approaches to statistical inference: the frequentist and Bayesianapproaches. Their differences are subtle but fundamental:

- In the case of the *frequentist approach*, the main assumption is that there is a population, which can be represented by several parameters, from which we can obtain numerous random samples. Population parameters are fixed but they are not accessible to the observer. The only way to derive information about these parameters is to take a sample of the population, to compute the parameters of thesample, and to use statistical inference techniques to make probable propositionsregarding population parameters.
- The *Bayesian approach* is based on a consideration that data are fixed, not the result of a repeatable sampling process, but parameters describing data can be described probabilistically. To this end, Bayesian inference methods focus on producing parameter distributions that represent all the knowledge we can extract from the sample and from prior information about the problem.

A deep understanding of the differences between these approaches is far beyondthe scope of this chapter, but there are many interesting references that will enable you to learn about it [1]. What is really important is to realize that the approaches are based on different assumptions which determine the validity of their inferences.The assumptions are related in the first case to a sampling process; and to a statistical model in the second case. Correct inference requires these assumptions to be correct. The fulfillment of this requirement is not part of the method, but it is the responsibility of the data scientist.

In this chapter, to keep things simple, we will only deal with the first approach, but we suggest the reader also explores the second approach as it is well worth it!

Statistical Inference: The Frequentist Approach

As we have said, the ultimate objective of statistical inference, if we adopt the fre- quentist approach, is to produce probable propositions concerning population param- eters from analysis of a sample. The most important classes of propositions are as follows:

- Propositions about *point estimates*. A point estimate is a particular value that bestapproximates some parameter of interest. For example, the mean or the varianceof the sample.
- Propositions about *confidence intervals* or *set estimates*. A confidence interval isa range of values that best represents some parameter of interest.
- Propositions about the acceptance or rejection of a *hypothesis*.

In all these cases, the production of propositions is based on a simple assumption: we can estimate the probability that the result represented by the proposition has been caused by chance. The estimation of this probability by sound methods is oneof the main topics of statistics.

The development of traditional statistics was limited by the scarcity of computa- tional resources. In fact, the only computational resources were mechanical devices and human computers, teams of people devoted to undertaking long and tedious calculations. Given these conditions, the main results of classical statistics are theo- retical approximations, based on idealized models and assumptions, to measure the effect of chance on the statistic of interest. Thus, concepts such as the *Central Limit Theorem*, the *empirical sample distribution* or the *t-test* are central to understandingthis approach.

The development of modern computers has opened an alternative strategy for measuring chance that is based on simulation; producing computationally inten- sive methods including resampling methods (such as bootstrapping), Markov chain Monte Carlo methods, etc. The most interesting characteristic of these methods is that they allow us to treat more realistic models.

Measuring the Variability in Estimates

Estimates produced by descriptive statistics are not equal to the *truth* but they are better as more data become available. So, it makes sense to use them as central elements of our propositions and to measure its variability with respect to the sample size.

Point Estimates

Let us consider a dataset of accidents in Barcelona in 2013. This dataset can be downloaded from the OpenDataBCN website,[1] Barcelona City Hall's open data service. Each register in the dataset represents an accident via a series of features: weekday, hour, address, number of dead and injured people, etc. This dataset will represent our population: the set of all reported traffic accidents in Barcelona during2013.

In[1]:

Sampling Distribution of Point Estimates

Let us suppose that we are interested in describing the daily number of traffic acci- dents in the streets of Barcelona in 2013. If we have access to the *population*, the computation of this parameter is a simple operation: the total number of accidents divided by 365.

```
data = pd.read_csv ("files/ch04/ACCIDENTS_GU_BCN_2013.csv")
data['Date'] = data[u'Dia de mes'].apply(lambda x: str(x))
                + '-' +
                data[u'Mes de any'].apply(lambda x: str(x))
data['Date'] = pd.to_datetime(data['Date'])
```

```
Out[1]:
Mean:
25.9095
```

But now, for illustrative purposes

suppose that we only have access to a limited part of the data (the *sample*): the number of accidents during *some* days of 2013. Can we still give an approximation of the population mean?

The most intuitive way to go about providing such a mean is simply to take the *sample mean*. The sample mean is a point estimate of the population mean. If we can only choose one value to estimate the population mean, then this is our best guess.

The problem we face is that estimates generally vary from one sample to another, and this sampling variation suggests our estimate may be close, but it will not be exactly equal to our parameter of interest. How can we measure this variability?

In our example, because we have access to the population, we can empirically buildthe *sampling distribution of the sample mean*[2] for a given number of observations.Then, we can use the sampling distribution to compute a measure of the variability.In Fig. 4.~~1~~, we can see t~~he~~ empirical sample distribution of the mean for *s* 10.000 samp 200 observations from our dataset. This empirical distribution has been built in the following way:

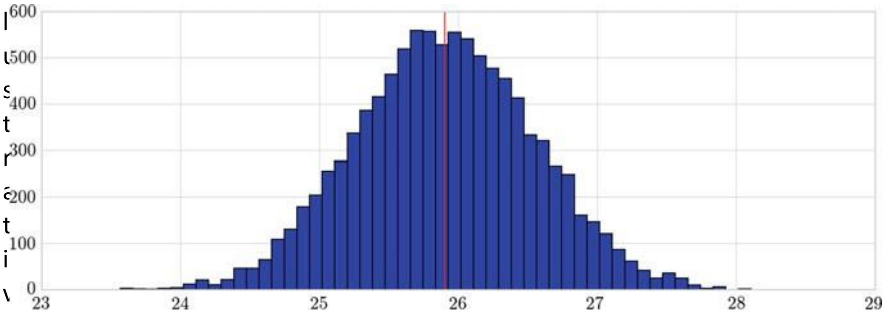


Fig. 4.1 Empirical distribution of the sample mean. In *red*, the mean value of this distribution

1. Draw *s* (a large number) independent samples $\{\mathbf{x}^1, \ldots, \mathbf{x}^s\}$ from the populationwhere each element $\mathbf{x}^j$ is composed of $\{x_i^j\}_{i=1,\ldots,n}$.
2. Evaluate the sample mean $\hat{\mu}^j = \frac{1}{n}\sum_{i=1}^{n} x_i^j$ of each sample.
3. Estimate the sampling distribution of μ by the empirical distribution of the sample replications.

```
# population
df = accidents.to_frame()
N_test = 10000
elements = 200
# mean array of samples
means = [0] * N_test
# sample generation
for i in range(N_test):
    rows = np.random.choice(df.index.values, elements)
    sampled_df = df.ix[rows]
    means[i] = sampled_df.mean()
```

In general, given a point estimate from a sample of size *n*, we define its *sampling distribution* as the distribution of the point estimate based on samples of size *n* from its population. This definition is valid for point estimates of other population parameters, such as the population median or population standard deviation, but we will focus on the analysis of the sample mean.

The sampling distribution of an estimate plays an important role in understanding the real meaning of propositions concerning point estimates. It is very useful to think of a particular point estimate as being drawn from such a distribution.

The Traditional Approach

In real problems, we do not have access to the real population and so estimation of the sampling distribution of the estimate from the empirical distribution of the sample replications is not an option. But this problem can be solved by making use of some theoretical results from traditional statistics.

It can be mathematically shown that given *n* independent observations $\{x_i\}_{i=1,..,n}$ of a population with a standard deviation σ_x, the standard deviation of the samplemean $\sigma_{\bar{x}}$, or *standard error*, can be approximated by this formula:

$$SE = \frac{\sigma_x}{\sqrt{n}}$$

The demonstration of this result is based on the Central Limit Theorem: an oldtheorem with a history that starts in 1810 when Laplace released his first paper on it.This formula uses the standard deviation of the population σ_x, which is not known, but it can be shown that if it is substituted by its empirical estimate $\hat{\sigma}_x$, the estimationis sufficiently good if $n > 30$ and the population distribution is not skewed. Thisallows us to estimate the standard error of the sample mean even if we do not have access to the population.

So, how can we give a measure of the variability of the sample mean? The answeris simple: by giving the **empirical standard error of the mean distribution**.

```
rows = np.random.choice(df.index.values, 200)
sampled_df = df.ix[rows]
est_sigma_mean = sampled_df.std()/math.sqrt(200)

print 'Direct estimation of SE from one sample of
      200 elements:', est_sigma_mean[0]
print ' Estimation of the SE by simulating 10000 samples of
      200 elements:', np.array(means).std()
```

Out[3]:
```
Direct estimation of SE from one sample of 200 elements: 0.6536Estimation of the SE by
simulating 10000 samples of 200
elements: 0.6362
```

Unlike the case of the sample mean, there is no formula for the standard error ofother interesting sample estimates, such as the median.

The Computationally Intensive Approach

Let us consider from now that our full dataset is a sample from a hypothetical population (this is the most common situation when analyzing real data!).

A modern alternative to the traditional approach to statistical inference is the bootstrapping method [2]. In the bootstrap, we draw n observations *with replacement* from the original data to create a bootstrap sample or resample. Then, we can calculate the mean for this resample. By repeating this process a large number of times, we can build a good approximation of the mean sampling distribution (see Fig. 4.2).

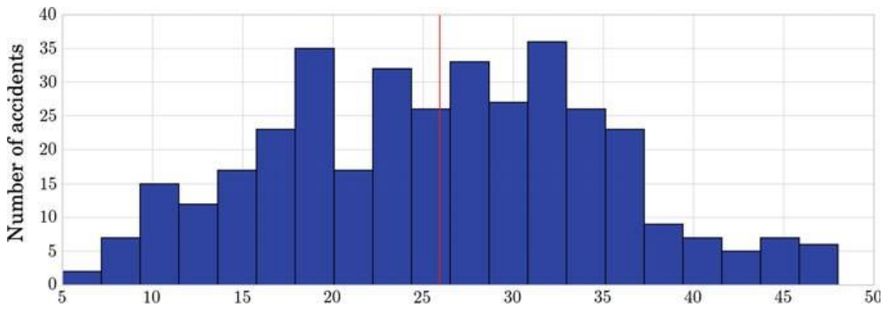


Fig. 4.2 Mean sampling distribution by bootstrapping. In *red*, the mean value of this distribution

In [4]:

```
def meanBootstrap(X, numberb):
    x = [0]*numberb
    for i in range(numberb):
        sample = [X[j]
                  for j
                  in np.random.randint(len(X), size=len(X))
                 ]
        x[i] = np.mean(sample)
    return x
m = meanBootstrap(accidents, 10000)
print "Mean estimate:", np.mean(m)
```

Out[4]:

```
Mean estimate: 25.9094
```

The basic idea of the bootstrapping method is that the observed sample contains sufficient information about the underlying distribution. So, the information we canextract from resampling the sample is a good approximation of what can be expected from resampling the population.

The bootstrapping method can be applied to other simple estimates such as the median or the variance and also to more complex operations such as estimates of censored data.[3]

Confidence Intervals

A point estimate Θ, such as the sample mean, provides a *single plausible value fora parameter*. However, as we have seen, a point estimate is rarely perfect; usually there is some error in the estimate. That is why we have suggested using the standard error as a measure of its variability.

Instead of that, a next logical step would be to provide *a plausible range of values*for the parameter. A plausible range of values for the sample parameter is called a *confidence interval*.

We will base the definition of *confidence interval* on two ideas:

1. Our point estimate is the most plausible value of the parameter, so it makes senseto build the confidence interval around the point estimate.
2. The *plausibility* of a range of values can be defined from the sampling distributionof the estimate.

For the case of the mean, the Central Limit Theorem states that its samplingdistribution is normal:

Theorem 4.1 *Given a population with a finite mean μ and a finite non-zero variance σ^2, the sampling distribution of the mean approaches a normal distribution with a mean of μ and a variance of σ^2/n as n, the sample size, increases.*

In this case, and in order to define an interval, we can make use of a well-knownresult from probability that applies to normal distributions: roughly 95% of the timeour estimate will be within 1.96 standard errors of the true mean of the distribution. If the interval spreads out 1.96 standard errors from a normally distributed point estimate, intuitively we can say that we are *roughly 95% confident that we have captured the true parameter*.

$$CI = [\Theta - 1.96 \times SE, \Theta + 1.96 \times SE]$$

In[5]:
```
m = accidents.mean()
se = accidents.std()/math.sqrt(len(accidents))
ci = [m - se*1.96, m + se*1.96]
print "Confidence interval:", ci
```

Out[5]:
```
Confidence interval: [24.975, 26.8440]
```

Suppose we want to consider confidence intervals where the confidence level issomewhat higher than 95%: perhaps we would like a confidence level of 99%. To create a 99% confidence interval, change 1.96 in the 95% confidence interval formula to be 2.58 (it can be shown that 99% of the time a normal random variable will be within 2.58 standard deviations of the mean).

In general, if the point estimate follows the normal model with standard error *SE* , then a confidence interval for the population parameter is

$$\Theta \pm z \times SE$$

where *z* corresponds to the confidence level selected:

Confidence Level	90%	95%	99%	99.9%
z Value	1.65	1.96	2.58	3.291

This is how we would compute a 95% confidence interval of the sample mean using bootstrapping:

1. Repeat the following steps for a large number, *s*, of times:

   a. Draw *n* observations with replacement from the original data to create abootstrap sample or resample.
   b. Calculate the mean for the resample.

2. Calculate the mean of your *s* values of the sample statistic. This process givesyou a "bootstrapped" estimate of the sample statistic.
3. Calculate the standard deviation of your *s* values of the sample statistic. Thisprocess gives you a "bootstrapped" estimate of the SE of the sample statistic.
4. Obtain the 2.5th and 97.5th percentiles of your *s* values of the sample statistic.

In [6]:

```
m = meanBootstrap(accidents, 10000)
sample_mean = np.mean(m)
sample_se = np.std(m)

print "Mean estimate:", sample_mean
print "SE of the estimate:", sample_se

ci = [np.percentile(m, 2.5), np.percentile(m, 97.5)]
print "Confidence interval:", ci
```

Out[6]:

```
Mean estimate: 25.9039
SE of the estimate: 0.4705
Confidence interval: [24.9834, 26.8219]
```

But What Does "95% Confident" Mean?

The real meaning of "confidence" is not evident and it must be understood from thepoint of view of the generating process.

Suppose we took many (infinite) samples from a population and built a 95% confidence interval from each sample. Then about 95% of those intervals would contain the actual parameter. In Fig. 4.3 we show how many confidence intervals computed from 100 different samples of 100 elements from our dataset contain the real population mean. If this simulation could be done with infinite different samples, 5% of those intervals would not contain the true mean.

So, when faced with a sample, the correct interpretation of a confidence intervalis as follows:

> In 95% of the cases, when I compute the 95% confidence interval from this sample, the true mean of the population will fall within the interval defined by these bounds: $\pm 1.96 \times SE$.

We cannot say either that our specific sample contains the true parameter or that the interval has a 95% chance of containing the true parameter. That interpretation would not be correct under the assumptions of traditional statistics.

Hypothesis Testing

Giving a measure of the variability of our estimates is one way of producing a statistical proposition about the population, but not the only one. R.A. Fisher (1890–1962) proposed an alternative, known as *hypothesis testing*, that is based on the concept of *statistical significance*.

Let us suppose that a deeper analysis of traffic accidents in Barcelona results in a difference between 2010 and 2013. Of course, the difference could be caused only by chance, because of the variability of both estimates. But it could also be the case that traffic conditions were very different in Barcelona during the two periods and, because of that, data from the two periods can be considered as belonging to two different populations. Then, the relevant question is: Are the observed effects real ornot?

Technically, the question is usually translated to: *Were the observed effects statistically significant*?

The process of determining the statistical significance of an effect is called *hypothesis testing*.

This process starts by simplifying the options into two competing hypotheses:

- H_0: The mean number of daily traffic accidents is the same in 2010 and 2013 (there is only one population, one true mean, and 2010 and 2013 are just differentsamples from the same population).
- H_A: The mean number of daily traffic accidents in 2010 and 2013 is different (2010 and 2013 are two samples from two different populations).

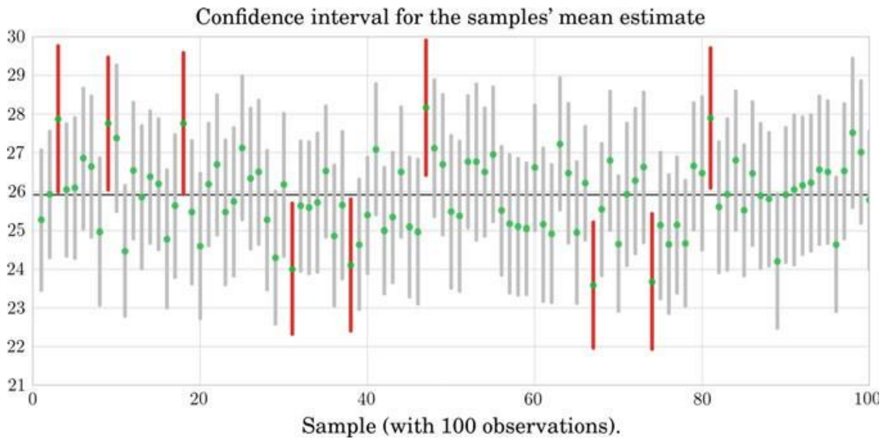


Fig. 4.3 This graph shows 100 sample means (*green points*) and its corresponding confidence intervals, computed from 100 different samples of 100 elements from our dataset. It can be observed that a few of them (those in *red*) do not contain the mean of the population (*black horizontal line*)

We call H_0 the *null hypothesis* and it represents a *skeptical* point of view: the effect we have observed is due to chance (due to the specific sample bias). H_A is the*alternative hypothesis* and it represents the other point of view: the effect is real.

The general rule of frequentist hypothesis testing: we will not *discard* H_0 (and hence we will not consider H_A) unless the observed effect is *implausible* under H_0.

Testing Hypotheses Using Confidence Intervals

We can use the concept represented by *confidence intervals* to measure the *plausi-bility* of a hypothesis.

We can illustrate the evaluation of the hypothesis setup by comparing the meanrate of traffic accidents in Barcelona during 2010 and 2013:

In [7]:
```
data  =  pd.read_csv("files/ch04/ACCIDENTS_GU_BCN_2010.csv",
                     encoding='latin-1')

# Create a new column which is the date
data['Date'] = data['Dia de mes'].apply(lambda x: str(x))
                + '-' +
                data['Mes de any'].apply(lambda x: str(x))
data2 = data['Date']
counts2010 = data['Date'].value_counts()
print '2010: Mean', counts2010.mean()

data  =  pd.read_csv("files/ch04/ACCIDENTS_GU_BCN_2013.csv",
                     encoding='latin-1')

# Create a new column which is the date
data['Date'] = data['Dia de mes'].apply(lambda x: str(x))
                + '-' +
                data['Mes de any'].apply(lambda x: str(x))
data2 = data['Date']
counts2013 = data['Date'].value_counts()
print '2013: Mean', counts2013.mean()
```

Out[7]:
```
2010: Mean 24.8109
2013: Mean 25.9095
```

This estimate suggests that in 2013 the mean rate of traffic accidents in Barcelonawas higher than it was in 2010. But is this effect statistically significant?

Based on our sample, the 95% confidence interval for the mean rate of trafficaccidents in Barcelona during 2013 can be calculated as follows:

In [8]:
```
n = len(counts2013)
mean = counts2013.mean()
s = counts2013.std()
ci = [mean - s*1.96/np.sqrt(n),  mean + s*1.96/np.sqrt(n)]
print '2010 accident rate estimate:', counts2010.mean()
print '2013 accident rate estimate:', counts2013.mean()
print 'CI for 2013:',ci
```

```
Out[8]: 2010 accident rate estimate: 24.8109
        2013 accident rate estimate: 25.9095
CI for 2013: [24.9751, 26.8440]
```

Because the 2010 accident rate estimate does not fall in the range of plausible values of 2013, we say the alternative hypothesis cannot be discarded. That is, it cannot be ruled out that in 2013 the mean rate of traffic accidents in Barcelona washigher than in 2010.

Interpreting CI Tests

Hypothesis testing is built around rejecting or failing to reject the null hypothesis. That is, we do not reject H_0 unless we have strong evidence against it. But what precisely does strong evidence mean? As a general rule of thumb, for those cases where the null hypothesis is actually true, we do not want to incorrectly reject H_0 more than 5% of the time. This corresponds to a *significance level* of $\alpha = 0.05$. In this case, the correct interpretation of our test is as follows:

If we use a 95% confidence interval to test a problem where the null hypothesis is true, wewill make an error whenever the point estimate is at least 1.96 standard errors away from thepopulation parameter. This happens about 5% of the time (2.5% in each tail).

Testing Hypotheses Using *p*-Values

A more advanced notion of *statistical significance* was developed by R.A. Fisher in the 1920s when he was looking for a test to decide whether variation in crop yields was due to some specific intervention or merely random factors beyond experimental control.

Fisher first assumed that fertilizer caused no difference (*null hypothesis*) and thencalculated P, the probability that an observed yield in a fertilized field would occurif fertilizer had no real effect. This probability is called the *p-value*.

The *p*-value is the probability of observing data at least as favorable to the alter-native hypothesis as our current dataset, if the null hypothesis is true. We typically use a summary statistic of the data to help compute the *p*-value and evaluate the hypotheses.

Usually, if P is less than 0.05 (the chance of a fluke is less than 5%) the result is declared *statistically significant*.

It must be pointed out that this choice is rather arbitrary and should not be takenas a scientific truth.

The goal of classical hypothesis testing is to answer the question, "*Given a sample and an apparent effect, what is the probability of seeing such an effect by chance?*"Here is how we answer that question:

- The first step is to quantify the size of the apparent effect by choosing a test statistic. In our case, the apparent effect is a difference in accident rates, so a natural choicefor the test statistic is the **difference in means between the two periods**.

- The second step is to define a *null hypothesis*, which is a model of the system based on the assumption that the apparent effect is not real. In our case, the null hypothesis is that there is no difference between the two periods.
- The third step is to compute a *p-value*, which is the probability of seeing the apparent effect if the null hypothesis is true. In our case, we would compute the difference in means, then compute the probability of seeing a difference as big, orbigger, under the null hypothesis.
- The last step is to *interpret the result*. If the *p*-value is low, the effect is said to be *statistically significant*, which means that it is unlikely to have occurred by chance. In this case we infer that the effect is more likely to appear in the larger population.

In our case, the test statistic can be easily computed:

In [9]:
```
m= len(counts2010)
n= len(counts2013)
p = (counts2013.mean() - counts2010.mean())
print 'm:', m, 'n:', n
print 'mean difference: ', p
```

Out[9]:
```
m: 365 n: 365
mean difference: 1.0986
```

To approximate the *p*-value , we can follow the following procedure:

1. Pool the distributions, generate samples with size n and compute the differencein the mean.
2. Generate samples with size n and compute the difference in the mean.
3. Count how many differences are larger than the observed one.

In [10]:
```
# pooling distributions
x = counts2010
y = counts2013
pool = np.concatenate([x, y])
np.random.shuffle(pool)

#sample generation
import random
N = 10000 # number of samples
diff = range(N)
for i in range(N):
    p1 = [random.choice(pool) for _ in xrange(n)]
    p2 = [random.choice(pool) for _ in xrange(n)]
    diff[i] = (np.mean(p1) - np.mean(p2))
```

In [11]:

```
# counting differences larger than the observed one
diff2 = np.array(diff)
w1 = np.where(diff2 > p)[0]

print 'p-value (Simulation)=', len(w1)/float(N),
      '(', len(w1)/float(N)*100 ,'%)', 'Difference =', p
if (len(w1)/float(N)) < 0.05:
    print 'The effect is likely'
else:
    print 'The effect is not likely'
```

Out[11]:

```
p-value (Simulation)= 0.0485 ( 4.85%) Difference = 1.098The effect is likely
```

Interpreting *P*-Values

A *p*-value is the probability of an observed (or more extreme) result arising only from chance.

If *P* is less than 0.05, there are two possible conclusions: there is a real effect or the result is an improbable fluke. *Fisher's method offers no way of knowing which isthe case*.

We must not confuse the odds of getting a result (if a hypothesis is true) with the odds of favoring the hypothesis if you observe that result. If *P* is less than 0.05,we cannot say that this means that it is 95% certain that the observed effect is real and could not have arisen by chance. Given an observation E and a hypothesis H, $P(E|H)$ and $P(H|E)$ are not the same!

Another common error equates *statistical significance* to *practical importance/ relevance*. When working with large datasets, we can detect statistical significancefor small effects that are meaningless in practical terms.

We have defined the effect as *a difference in mean as large or larger than δ, considering the sign*. A test like this is called *one sided*.

If the relevant question is whether *accident rates are different*, then it makes senseto test the absolute difference in means. This kind of test is called two sided becauseit counts both sides of the distribution of differences.

Direct Approach

The formula for the standard error of the absolute difference in two means is similarto the formula for other standard errors. Recall that the standard error of a single mean can be approximated by:

$$SE_{\bar{x}_1} = \frac{\sigma_1}{\sqrt{n_1}}$$

The standard error of the difference of two sample means can be constructed fromthe standard errors of the separate sample means:

$$SE_{\bar{x}_1-\bar{x}_2} = \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$$

This would allow us to define a direct test with the 95% confidence interval.

But Is the Effect E Real?

We do not yet have an answer for this question! We have defined a null hypothesisH_0 (the effect is not real) and we have computed the probability of the observed effect under the null hypothesis, which is $P(E|H_0)$, where E is an effect as big as or bigger than the apparent effect and a *p*-value .

We have stated that from the frequentist point of view, we cannot consider H_A unless $P(E|H_0)$ is less than an arbitrary value. But the real answer to this question must be based on comparing $P(H_0|E)$ to $P(H_A|E)$, not on $P(E|H_0)$! One possi- ble solution to these problems is to use *Bayesian reasoning*; an alternative to the frequentist approach.

No matter how many data you have, you will still depend on intuition to decide how to interpret, explain, and use that data. Data cannot speak by themselves. Data scientists are interpreters, offering one interpretation of what the useful narrative story derived from the data is, if there is one at all.

UNIT-3

Supervised Learning: First step, learning curves, training-validation and test. Learning models generalities, support vector machines, random forest. Examples

Supervised Learning

Machine learning involves coding programs that automatically adjust their perfor- mance in accordance with their exposure to information in data. This learning is achieved via a parameterized model with tunable parameters that are automatically adjusted according to different performance criteria. Machine learning can be con- sidered a subfield of artificial intelligence (AI) and we can roughly divide the fieldinto the following three major classes.

1. Supervised learning: Algorithms which learn from a training set of labeled examples (exemplars) to generalize to the set of all possible inputs. Examples of techniques in supervised learning: logistic regression, support vector machines,decision trees, random forest, etc.
2. Unsupervised learning: Algorithms that learn from a training set of unlabeled examples. Used to explore data according to some statistical, geometric or sim- ilarity criterion. Examples of unsupervised learning include k-means clustering and kernel density estimation. We will see more on this kind of techniques in Chap. 7.
3. Reinforcement learning: Algorithms that learn via reinforcement from criticismthat provides information on the quality of a solution, but not on how to improve it. Improved solutions are achieved by iteratively exploring the solution space.

This chapter focuses on a particular class of supervised machine learning: *clas- sification*. As a data scientist, the first step you apply given a certain problem is to identify the question to be answered. According to the type of answer we are seeking, we are directly aiming for a certain set of techniques.

- If our question is answered by YES/NO, we are facing a classification problem. Classifiers are also the tools to use if our question admits only a discrete set of answers, i.e., we want to select from a finite number of choices.

  – Given the results of a clinical test, e.g., does this patient suffer from diabetes?
  – Given a magnetic resonance image, is it a tumor shown in the image?
  – Given the past activity associated with a credit card, is the current operationfraudulent?

- If our question is a prediction of a real-valued quantity, we are faced with a *regres-sion* problem. We will go into details of regression in Chap. 6.

– Given the description of an apartment, what is the expected market value of theflat? What will the value be if the apartment has an elevator?
– Given the past records of user activity on Apps, how long will a certain client be connected to our App?
– Given my skills and marks in computer science and maths, what mark will I achieve in a data science course?

Observe that some problems can be solved using both regression and classification. As we will see later, many classification algorithms are thresholded regressors. There is a certain skill involved in designing the correct question and this dramatically affects the solution we obtain.

The Problem

In this chapter we use data from the Lending Club[1] to develop our understanding of machine learning concepts. The Lending Club is a peer-to-peer lending company. It offers loans which are funded by other people. In this sense, the Lending Club acts as a hub connecting borrowers with investors. The client applies for a loan of acertain amount, and the company assesses the risk of the operation. If the applicationis accepted, it may or may not be fully covered. We will focus on the predictionof whether the loan will be fully funded, based on the scoring of and information related to the application.

We will use the partial dataset of period 2007–2011. Framing the problem a little bit more, based on the information supplied by the customer asking for a loan, we want to predict whether it will be granted up to a certain threshold *thr* . The attributes we use in this problem are related to some of the details of the loan application, such as amount of the loan applied for the borrower, monthly payment to be made by the borrower if the loan is accepted, the borrower's annual income, the number of incidences of delinquency in the borrower's credit file, and interest rate of the loan,among others.

In this case we would like to predict unsuccessful accepted loans. A loan applica-tion is unsuccessful if the funded amount (funded_amnt) or the amount funded by investors (funded_amnt_inv) falls far short of the requested loan amount (loan_amnt). That is,

$$\frac{loan - funded}{loan} \geq 0.95.$$

First Steps

Note that in this problem we are predicting a binary value: either the loan is fully funded or not. Classification is the natural choice of machine learning tools for prediction with discrete known outcomes. According to the cardinality of the target set, one usually distinguishes between *binary* classifiers when the target output onlytakes two values, i.e., the classifier answers questions with a yes or a no; or *multiclass* classifiers, for a larger number of classes. This issue is important in that not all methods can naturally handle the multiclass setting.[2]

In a formal way, classification is regarded as the problem of finding a function $h(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathcal{K}$ that maps an input space in $\mathbb{R}^d$ onto a discrete set of k target outputs or classes $\mathcal{K} = \{1, \ldots, k\}$. In this setting, the features are arranged as a vector $\mathbf{x}$ of d real-valued numbers.[3]

We can encode both target states in a numerical variable, e.g., a successful loan target can take value $+1$; and it is -1, otherwise.

Let us check the dataset,[4]

```
import pickle
ofname = open('./files/ch05/dataset_small.pkl','rb')
# x stores input data and y target values
(x,y) = pickle.load(ofname)
```

In[1]:

[2]Several well-known techniques such as support vector machines or adaptive boosting (adaboost) are originally defined in the binary case. Any binary classifier can be extended to the multiclass case in two different ways. We may either change the formulation of the learning/optimization process. This requires the derivation of a new learning algorithm capable of handling the new modeling. Alternatively, we may adopt ensemble techniques. The idea behind this latter approach is that we may divide the multiclass problem into several binary problems; solve them; and then aggregate the results. If the reader is interested in these techniques, it is a good idea to look for: one-versus-all, one-versus-one, or error correcting output codes methods.

[3]Many problems are described using categorical data. In these cases either we need classifiers that are capable of coping with this kind of data or we need to change the representation of those variables into numerical values.

[4]The notebook companion shows the preprocessing steps, from reading the dataset, cleaning and imputing data, up to saving a subsampled clean version of the original dataset.

A problem in Scikit-learn is modeled as follows:

- Input data is structured in Numpy arrays. The size of the array is expected to be [n_samples, n_features]:
  - n_samples: The number of samples (*n*). Each sample is an item to process (e.g., classify). A sample can be a document, a picture, an audio file, a video, an astronomical object, a row in a database or CSV file, or whatever you can describe with a fixed set of quantitative traits.
  - n_features: The number of features (*d*) or distinct traits that can be used to describe each item in a quantitative manner. Features are generally real-valued,but may be Boolean, discrete-valued or even categorical.

$$\text{feature matrix :} \quad \mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ x_{31} & x_{32} & \cdots & x_{3d} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix}$$

$$\text{label vector :} \quad \mathbf{y}^T = [y_1, y_2, y_3, \cdots y_n]$$

The number of features must be fixed in advance. However, it can be very great(e.g., millions of features).

In[2]:
```
dims = x.shape[1]
N = x.shape[0]
print 'dims: ' + str(dims) + ', samples: ' + str(N)
```

Out[2]:
```
dims: 15, samples: 4140
```

Considering data arranged as in the previous matrices we refer to:

- the columns as features, attributes, dimensions, regressors, covariates, predictors,or independent variables;
- the rows as instances, examples, or samples;
- the target as the label, outcome, response, or dependent variable.

All objects in Scikit-learn share a uniform and limited API consisting of three complementary interfaces:

- an estimator interface for building and fitting models (fit());
- a predictor interface for making predictions (predict());
- a transformer interface for converting data (transform()).

Let us apply a classifier using Python’s Scikit-learn libraries,

In [3]:

```
from sklearn import neighbors
from sklearn import datasets
# Create an instance of K-nearest neighbor classifier
knn = neighbors.KNeighborsClassifier(n_neighbors = 11)
# Train the classifier
knn.fit(x, y)
# Compute the prediction according to the model
yhat = knn.predict(x)
# Check the result on the last example
print 'Predicted value: ' + str(yhat[-1]),
      ', real target: ' + str(y[-1])
```

Out[3]: Predicted value: -1.0 , real target: -1.0

The basic measure of performance of a classifier is its *accuracy*. This is defined as the number of correctly predicted examples divided by the total amount of examples. Accuracy is related to the error as follows: $acc = 1 - err$.

$$acc = \frac{\text{Number of correct predictions}}{n}$$

Each estimator has a score()method that invokes the default scoring metric. In the case of k-nearest neighbors, this is the classification accuracy.

In [4]:

```
knn.score(x,y)
```

Out[4]: 0.83164251207729467

It looks like a really good result. But how good is it? Let us first understand a little bit more about the problem by checking the distribution of the labels.

Let us load the dataset and check the distribution of labels:

In [5]:

```
plt.pie(np.c_[np.sum(np.where(y == 1, 1, 0)),
        np.sum(np.where(y == -1, 1, 0))][0],
        labels = ['Not fully funded','Full amount'],
        colors = ['r', 'g'],shadow = False,
        autopct = '%.2f' )
plt.gcf().set_size_inches((7, 7))
```

with the result observed in Fig. 5.1.

Note that there are far more positive labels than negative ones. In this case, the dataset is referred to as *unbalanced*.[5] This has important consequences for a classifier as we will see later on. In particular, a very simple rule such as always predict the

[5]The term unbalanced describes the condition of data where the ratio between positives and negatives is a small value. In these scenarios, always predicting the majority class usually yields accurate performance, though it is not very informative. This kind of problems is very common when we want to model unusual events such as rare diseases, the occurrence of a failure in machinery, fraudulent credit card operations, etc. In these scenarios, gathering data from usual events is very easy but collecting data from unusual events is difficult and results in a comparatively small dataset.

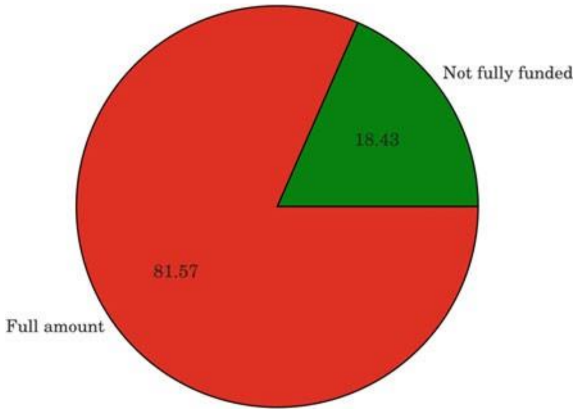


Fig. 5.1 Pie chart showing the distribution of labels inthe dataset

majority class, will give us good performance. In our problem, always predicting that the loan will be fully funded correctly predicts 81.57% of the samples. Observethat this value is very close to that obtained using the classifier.

Although accuracy is the most normal metric for evaluating classifiers, there arecases when the business value of correctly predicting elements from one class is different from the value for the prediction of elements of another class. In those cases, accuracy is not a good performance metric and more detailed analysis is needed. The *confusion matrix* enables us to define different metrics considering such scenarios. The confusion matrix considers the concepts of the classifier outcome and the actual ground truth or gold standard. In a binary problem, there are four possiblecases:

- *True positives (TP)*: When the classifier predicts a sample as positive and it really is positive.
- *False positives (FP)*: When the classifier predicts a sample as positive but in fact it is negative.
- *True negatives (TN)*: When the classifier predicts a sample as negative and it really is negative.
- *False negatives (FN)*: When the classifier predicts a sample as negative but in fact it is positive.

We can summarize this information in a matrix, namely the confusion matrix, asfollows:

		Gold Standard		
		Positive	Negative	
Prediction	Positive	TP	FP	→ Precision
	Negative	FN	TN	→ Negative Predictive Value

↓
Sens itivity
↓
Spec ificity (Recall)

The combination of these elements allows us to define several performance metrics:

- *Accuracy:*

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- Column-wise we find these two partial performance metrics:

  – *Sensitivity or Recall:*

$$\text{sensitivity} = \frac{\text{TP}}{\text{Real Positives}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

  – *Specificity:*

$$\text{specificity} = \frac{\text{TN}}{\text{Real Negatives}} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

- Row-wise we find these two partial performance metrics:

  – *Precision or Positive Predictive Value:*

$$\text{precision} = \frac{\text{TP}}{\text{Predicted Positives}} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

  – *Negative predictive value:*

$$\text{NPV} = \frac{\text{TN}}{\text{Predicted Negative}} = \frac{\text{TN}}{\text{TN} + \text{FN}}$$

These partial performance metrics allow us to answer questions concerning how often a classifier predicts a particular class, e.g., what is the rate of predictions for not fully funded loans that have actually not been fully funded? This question is answered by recall. In contrast, we could ask: Of all the fully funded loans predicted by the classifier, how many have been fully funded? This is answered by the precision metric.

Let us compute these metrics for our problem.

In [6]:
```
yhat  =  knn.predict(x)
TP  =  np.sum(np.logical_and(yhat  ==  -1,  y  == -1))
TN  =  np.sum(np.logical_and(yhat  ==  1,  y  ==  1))
FP  =  np.sum(np.logical_and(yhat  ==  -1,  y  ==  1))
FN  =  np.sum(np.logical_and(yhat  ==  1,  y  ==  -1))
print  'TP:  '+  str(TP),  ',  FP:  '+  str(FP)
print  'FN:  '+  str(FN),  ',  TN:  '+  str(TN)
```

Out[6]:
```
TP: 3370 ,FP: 690
     FN: 7 ,        TN: 73
```

Scikit-learn provides us with the confusion matrix,

In [7]:
```
from  sklearn  import  metrics
metrics.confusion_matrix(yhat,  y)
#  sklearn  uses  a  transposed  convention  for  the  confusion
#  matrix  thus  I  change  targets  and  predictions
```

Out[7]:
```
3370, 690
     7,    73
```

Let us check the following example. Let us select a nearest neighbor classifier with the number of neighbors equal to one instead of eleven, as we did before, andcheck the training error.

In [8]:
```
#  Train  a  classifier  using  .fit()
knn = neighbors.KNeighborsClassifier(n_neighbors  =  1)
knn.fit(x,  y)
yhat  =  knn.predict(x)

print  "classification  accuracy:"  +
       str(metrics.accuracy_score(yhat,  y))
print  "confusion  matrix:  \n"  +
       str(metrics.confusion_matrix(yhat,  y))
```

Out[8]:
```
classification accuracy: 1.0 confusion matrix:
     3377  0
       0   763
```

The performance measure is perfect! 100% accuracy and a diagonal confusion matrix! This looks good. However, up to this point we have checked the classifier performance on the same data it has been trained with. During exploitation, in real applications, we will use the classifier on data not previously seen. Let us simulate this effect by splitting the data into two sets: one will be used for learning (*trainingset*) and the other for testing the accuracy (*test set*).

In [9]:
```
# Simulate a real case: Randomize and split data into
# two subsets PRC*100\% for training and the rest
# (1-PRC)*100\% for testing
perm = np.random.permutation(y.size)
PRC = 0.7
split_point = int(np.ceil(y.shape[0]*PRC))

X_train = x[perm[:split_point].ravel(),:]
y_train = y[perm[:split_point].ravel()]

X_test = x[perm[split_point:].ravel(),:]
y_test = y[perm[split_point:].ravel()]
```

If we check the shapes of the training and test sets we obtain,

Out[9]:
```
Training shape: (2898, 15), training targets shape: (2898,)
Testing shape: (1242, 15), testing targets shape: (1242,)
```

With this new partition, let us train the model

In [10]:
```
#Train a classifier on training data
knn = neighbors.KNeighborsClassifier(n_neighbors = 1)
knn.fit(X_train, y_train)
yhat = knn.predict(X_train)

print "\n TRAINING STATS:"
print "classification accuracy:" +
      str(metrics.accuracy_score(yhat, y_train))
print "confusion matrix: \n" +
      str(metrics.confusion_matrix(y_train, yhat))
```

Out[10]:
```
TRAINING STATS:
classification accuracy: 1.0
confusion matrix:
 2355  0
    0  543
```

As expected from the former experiment, we achieve a perfect score. Now let ussee what happens in the simulation with previously unseen data.

In [11]:
```
#Check on the test set
yhat = knn.predict(X_test)
print "TESTING STATS:"
print "classification accuracy:",
      metrics.accuracy_score(yhat, y_test)
print "confusion matrix: \n" +
      str(metrics.confusion_matrix(yhat, y_test))
```

Out[11]:
```
TESTING STATS:
classification accuracy: 0.754428341385
confusion matrix:
 865 148
 157 72
```

Observe that each time we run the process of randomly splitting the dataset and train a classifier we obtain a different performance. A good simulation for approxi-mating the test error is to run this process many times and average the performances.Let us do this![6]

In [12]:

```
# Spitting done by using the tools provided by sklearn:
from sklearn.cross_validation import train_test_split

PRC = 0.3
acc = np.zeros((10,))
for i in xrange(10):
    X_train, X_test, y_train, y_test =
        train_test_split(x, y, test_size = PRC)
    knn = neighbors.KNeighborsClassifier(n_neighbors = 1)
    knn.fit(X_train, y_train)
    yhat = knn.predict(X_test)
    acc[i] = metrics.accuracy_score(yhat, y_test)
acc.shape = (1, 10)
print "Mean expected error:" + str(np.mean(acc[0]))
```

Out[12]:

```
Mean expected error: 0.754669887279
```

As we can see, the resulting error is below 81%, which was the result of the mostnaive decision process. What is wrong with this result?

Let us introduce the nomenclature for the quantities we have just computed anddefine the following terms.

- *In-sample error* E_{in}: The in-sample error or training error is the error measuredover all the observed data samples in the training set, i.e.,

$$E_{\text{in}} = \frac{1}{N}\sum_{i=1}^{N} e(x_i, y_i)$$

- *Out-of-sample error* E_{out}: The out-of-sample error or generalization error measures the expected error on unseen data. We can approximate/simulate this quantity by holding back some training data for testing purposes.

$$E_{\text{out}} = \mathbb{E}_{x,y}(e(x, y))$$

Note that the definition of the instantaneous error $e(x_i, y_i)$ is still missing. For example, in classification we could use the indicator function to account for a cor- rectly classified sample as follows:

$$e(x_i, y_i) = I[h(x_i) = y_i] = \begin{cases} 1, \text{ if } h(x_i) = y_i \\ 0 \text{ otherwise.} \end{cases}$$

[6]*sklearn* allows us to easily automate the train/test splitting using the function train_test_split(...).

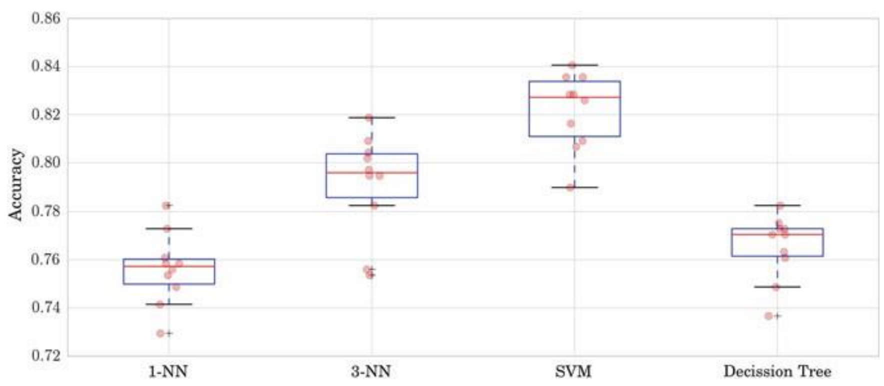


Fig. 5.2 Comparison of the methods using the accuracy metric

Observe that:

$$E_{\text{out}} \geq E_{\text{in}}$$

Using the expected error on the test set, we can select the best classifier for our application. This is called model selection. In this example we cover the most simplistic setting. Suppose we have a set of different classifiers and want to select the "best" one. We may use the one that yields the lowest error rate.

In [13]:

```
from sklearn import tree
from sklearn import svm
PRC = 0.1
acc_r = np.zeros((10, 4))
for i in xrange(10):
    X_train, X_test, y_train, y_test =
        train_test_split(x, y, test_size = PRC)
    nn1 = neighbors.KNeighborsClassifier(n_neighbors = 1)
    nn3 = neighbors.KNeighborsClassifier(n_neighbors = 3)
    svc = svm.SVC()
    dt = tree.DecisionTreeClassifier()

    nn1.fit(X_train, y_train)
    nn3.fit(X_train, y_train)
    svc.fit(X_train, y_train)
    dt.fit(X_train, y_train)

    yhat_nn1 = nn1.predict(X_test)
    yhat_nn3 = nn3.predict(X_test)
    yhat_svc = svc.predict(X_test)
    yhat_dt = dt.predict(X_test)

    acc_r[i][0] = metrics.accuracy_score(yhat_nn1, y_test)
    acc_r[i][1] = metrics.accuracy_score(yhat_nn3, y_test)
    acc_r[i][2] = metrics.accuracy_score(yhat_svc, y_test)
    acc_r[i][3] = metrics.accuracy_score(yhat_dt, y_test)
```

Figure 5.2 shows the results of applying the code.

This process is one particular form of a general model selection technique called *cross-validation*. There are other kinds of cross-validation, such as *leave-one-out* or*K-fold cross-validation*.

- In leave-one-out, given N samples, the model is trained with $N-1$ samples and tested with the remaining one. This is repeated N times, once per training sampleand the result is averaged.
- In K-fold cross-validation, the training set is divided into K nonoverlapping splits. K-1 splits are used for training and the remaining one used to assess the mean. This process is repeated K times leaving one split out each time. The results are then averaged.

What Is Learning?

Let us recall the two basic values defined in the last section. We talk of *training error* or in-sample error, E_{in}, which refers to the error measured over all the observed datasamples in the training set. We also talk of *test error* or *generalization error*, E_{out}, as the error expected on unseen data.

We can empirically estimate the generalization error by means of cross-validationtechniques and observe that:

$$E_{\text{out}} \geq E_{\text{in}}.$$

The goal of learning is to minimize the generalization error; but how can we guarantee this minimization using only training data?

From the above inequality it is easy to derive a couple of very intuitive ideas.

- Because E_{out} is greater than or equal to E_{in}, it is desirable to have
$$E_{\text{in}} \to 0.$$
- Additionally, we also want the training error behavior to track the generalization error so that if one minimizes the in-sample error the out-of-sample error follows,i.e.,
$$E_{\text{out}} \approx E_{\text{in}}.$$

We can rewrite the second condition as

$$E_{\text{in}} \leq E_{\text{out}} \leq E_{\text{in}} + \Omega,$$

with $\Omega \to 0$.

We would like to characterize Ω in terms of our problem parameters, i.e., thenumber of samples (N), dimensionality of the problem (d), etc.

Statistical analysis offers an interesting characterization of this quantity[7]

[7]The reader should note that there are several bounds in machine learning to characterize the generalization error. Most of them come from variations of Hoeffding's inequality.

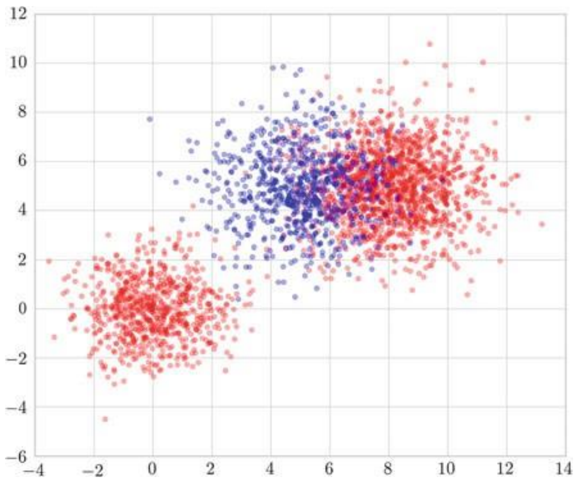


Fig. 5.3 Toy problem data

$$E_{out} \le E_{in}(C) + O\left(\sqrt{\frac{\log C}{N}}\right),$$

where C is a measure of the complexity of the model class we are using. Technically, we may also refer to this model class as the hypothesis space.

Learning Curves

Let us simulate the effect of the number of examples on the training and test errorsfor a given complexity. This curve is called the *learning curve*. We will focus for amoment in a more simple case. Consider the toy problem in Fig. 5.3.

Let us take a classifier and vary the number of examples we feed it for training purposes, then check the behavior of the training and test accuracies as the numberof examples grows. In this particular case, we will be using a decision tree with fixedmaximum depth.

Observing the plot in Fig. 5.4, we can see that:

- As the number of training samples increases, both errors tend to the same
- value. When we have few training data, the training error is very small but the test erroris very large.

Now check the learning curve when the degree of complexity is greater in Fig. 5.5. We simulate this effect by increasing the maximum depth of the tree.

And if we put both curves together, we have the results shown in Fig. 5.6.

Although both show similar behavior, we can note several differences: