Python Data Science Essentials

The book starts by introducing you to setting up your essential data science toolbox. Then it will guide you across all the data munging and preprocessing phases. This will be done in a manner that explains all the core data science activities related to loading data, transforming and fixing it for analysis, as well as exploring and processing it. Finally, it will complete the overview by presenting you with the main machine learning algorithms, the graph analysis technicalities, and all the visualization instruments that can make your life easier in presenting your results.

In this walkthrough, structured as a data science project, you will always be accompanied by clear code and simplified examples to help you understand the underlying mechanics and real-world datasets.

Who this book is written for

If you are an aspiring data scientist and you have at least a working knowledge of data analysis and Python, this book will get you started in data science. Data analysts with experience of R or MATLAB will also find the book to be a comprehensive reference to enhance their data manipulation and machine learning skills.

What you will learn from this book

- Set up your data science toolbox using a Python scientific environment on Windows, Mac, and Linux
- Get data ready for your data science project
- Manipulate, fix, and explore data in order to solve data science problems
- Set up an experimental pipeline to test your data science hypothesis
- Choose the most effective and scalable learning algorithm for your data science tasks
- Optimize your machine learning models to get the best performance
- Explore and cluster graphs, taking advantage of interconnections and links in your data


Alberto Boschetti
Luca Massaron
In this package, you will find:

- The authors biography
- A preview chapter from the book, Chapter 1 'First Steps'
- A synopsis of the book’s content
- More information on Python Data Science Essentials

About the Authors

Alberto Boschetti is a data scientist with expertise in signal processing and statistics. He holds a PhD in telecommunication engineering and currently lives and works in London. In his work projects, he faces challenges involving natural language processing (NLP), machine learning, and probabilistic graph models everyday. He is very passionate about his job and he always tries to stay updated on the latest developments in data science technologies by attending meetups, conferences, and other events.

Luca Massaron is a data scientist and marketing research director who specializes in multivariate statistical analysis, machine learning, and customer insight, with over a decade of experience in solving real-world problems and generating value for stakeholders by applying reasoning, statistics, data mining, and algorithms. From being a pioneer of web audience analysis in Italy to achieving the rank of a top 10 Kaggler, he has always been passionate about everything regarding data and analysis and about demonstrating the potentiality of data-driven knowledge discovery to both experts and nonexperts. Favoring simplicity over unnecessary sophistication, he believes that a lot can be achieved in data science by understanding its essentials.
Python Data Science Essentials

"A journey of a thousand miles begins with a single step."

– Laozi (604 BC - 531 BC)

Data science is a relatively new knowledge domain that requires the successful integration of linear algebra, statistical modelling, visualization, computational linguistics, graph analysis, machine learning, business intelligence, and data storage and retrieval.

The Python programming language, having conquered the scientific community during the last decade, is now an indispensable tool for the data science practitioner and a must-have tool for every aspiring data scientist. Python will offer you a fast, reliable, cross-platform, mature environment for data analysis, machine learning, and algorithmic problem solving. Whatever stopped you before from mastering Python for data science applications will be easily overcome by our easy step-by-step and example-oriented approach that will help you apply the most straightforward and effective Python tools to both demonstrative and real-world datasets.

Leveraging your existing knowledge of Python syntax and constructs (but don't worry, we have some Python tutorials if you need to acquire more knowledge on the language), this book will start by introducing you to the process of setting up your essential data science toolbox. Then, it will guide you through all the data munging and preprocessing phases. A necessary amount of time will be spent in explaining the core activities related to transforming, fixing, exploring, and processing data. Then, we will demonstrate advanced data science operations in order to enhance critical information, set up an experimental pipeline for variable and hypothesis selection, optimize hyper-parameters, and use cross-validation and testing in an effective way.

Finally, we will complete the overview by presenting you with the main machine learning algorithms, graph analysis technicalities, and all the visualization instruments that can make your life easier when it comes to presenting your results.

In this walkthrough, which is structured as a data science project, you will always be accompanied by clear code and simplified examples to help you understand the underlying mechanics and real-world datasets. It will also give you hints dictated by experience to help you immediately operate on your current projects. Are you ready to start? We are sure that you are ready to take the first step towards a long and incredibly rewarding journey.
What This Book Covers

Chapter 1, First Steps, introduces you to all the basic tools (command shell for interactive computing, libraries, and datasets) necessary to immediately start on data science using Python.

Chapter 2, Data Munging, explains how to upload the data to be analyzed by applying alternative techniques when the data is too big for the computer to handle. It introduces all the key data manipulation and transformation techniques.

Chapter 3, The Data Science Pipeline, offers advanced explorative and manipulative techniques, enabling sophisticated data operations to create and reduce predictive features, spot anomalous cases and apply validation techniques.

Chapter 4, Machine Learning, guides you through the most important learning algorithms that are available in the Scikit-learn library, which demonstrates the practical applications and points out the key values to be checked and the parameters to be tuned in order to get the best out of each machine learning technique.

Chapter 5, Social Network Analysis, elaborates the practical and effective skills that are required to handle data that represents social relations or interactions.

Chapter 6, Visualization, completes the data science overview with basic and intermediate graphical representations. They are indispensable if you want to visually represent complex data structures and machine learning processes and results.

Chapter 7, Strengthen Your Python Foundations, covers a few Python examples and tutorials focused on the key features of the language that it is indispensable to know in order to work on data science projects.

This chapter is not part of the book, but it has to be downloaded from Packt Publishing website at https://www.packtpub.com/sites/default/files/downloads/04290S_Chapter-07.pdf.
Whether you are an eager learner of data science or a well-grounded data science practitioner, you can take advantage of this essential introduction to Python for data science. You can use it to the fullest if you already have at least some previous experience in basic coding, writing general-purpose computer programs in Python, or some other data analysis-specific language, such as MATLAB or R.

The book will delve directly into Python for data science, providing you with a straight and fast route to solve various data science problems using Python and its powerful data analysis and machine learning packages. The code examples that are provided in this book don't require you to master Python. However, they will assume that you at least know the basics of Python scripting, data structures such as lists and dictionaries, and the working of class objects. If you don't feel confident about this subject or have minimal knowledge of the Python language, we suggest that before you read this book, you should take an online tutorial, such as the Code Academy course at http://www.codecademy.com/en/tracks/python or Google's Python class at https://developers.google.com/edu/python/. Both the courses are free, and in a matter of a few hours of study, they should provide you with all the building blocks that will ensure that you enjoy this book to the fullest. We have also prepared a tutorial of our own, which you can download from the Packt Publishing website, in order to provide an integration of the two aforementioned free courses.

In any case, don't be intimidated by our starting requirements; mastering Python for data science applications isn't as arduous as you may think. It's just that we have to assume some basic knowledge on the reader's part because our intention is to go straight to the point of using data science without having to explain too much about the general aspects of the language that we will be using.

Are you ready, then? Let's start!
In this short introductory chapter, we will work out the basics to set off in full swing and go through the following topics:

- How to set up a Python **Data Science Toolbox**
- Using IPython
- An overview of the data that we are going to study in this book

## Introducing data science and Python

Data science is a relatively new knowledge domain, though its core components have been studied and researched for many years by the computer science community. These components include linear algebra, statistical modelling, visualization, computational linguistics, graph analysis, machine learning, business intelligence, and data storage and retrieval.

Being a new domain, you have to take into consideration that currently the frontier of data science is still somewhat blurred and dynamic. Because of its various constituent set of disciplines, please keep in mind that there are different profiles of data scientists, depending on their competencies and areas of expertise.

In such a situation, what can be the best tool of the trade that you can learn and effectively use in your career as a data scientist? We believe that the best tool is Python, and we intend to provide you with all the essential information that you will need for a fast start.

Also, other tools such as R and MATLAB provide data scientists with specialized tools to solve specific problems in statistical analysis and matrix manipulation in data science. However, only Python completes your data scientist skill set. This multipurpose language is suitable for both development and production alike and is easy to learn and grasp, no matter what your background or experience is.

Created in 1991 as a general-purpose, interpreted, object-oriented language, Python has slowly and steadily conquered the scientific community and grown into a mature ecosystem of specialized packages for data processing and analysis. It allows you to have uncountable and fast experimentations, easy theory developments, and prompt deployments of scientific applications.

At present, the Python characteristics that render it an indispensable data science tool are as follows:

- Python can easily integrate different tools and offer a truly unifying ground for different languages (Java, C, Fortran, and even language primitives), data strategies, and learning algorithms that can be easily fitted together and which can concretely help data scientists forge new powerful solutions.
• It offers a large, mature system of packages for data analysis and machine learning. It guarantees that you will get all that you may need in the course of a data analysis, and sometimes even more.

• It is very versatile. No matter what your programming background or style is (object-oriented or procedural), you will enjoy programming with Python.

• It is cross-platform; your solutions will work perfectly and smoothly on Windows, Linux, and Mac OS systems. You won't have to worry about portability.

• Although interpreted, it is undoubtedly fast compared to other mainstream data analysis languages such as R and MATLAB (though it is not comparable to C, Java, and the newly emerged Julia language). It can be even faster, thanks to some easy tricks that we are going to explain in this book.

• It can work with in-memory big data because of its minimal memory footprint and excellent memory management. The memory garbage collector will often save the day when you load, transform, dice, slice, save, or discard data using the various iterations and reiterations of data wrangling.

• It is very simple to learn and use. After you grasp the basics, there's no other better way to learn more than by immediately starting with the coding.

Installing Python
First of all, let's proceed to introduce all the settings you need in order to create a fully working data science environment to test the examples and experiment with the code that we are going to provide you with.

Python is an open source, object-oriented, cross-platform programming language that, compared to its direct competitors (for instance, C++ and Java), is very concise. It allows you to build a working software prototype in a very short time. Did it become the most used language in the data scientist's toolbox just because of this? Well, no. It's also a general-purpose language, and it is very flexible indeed due to a large variety of available packages that solve a wide spectrum of problems and necessities.

Python 2 or Python 3?
There are two main branches of Python: 2 and 3. Although the third version is the newest, the older one is still the most used version in the scientific area, since a few libraries (see http://py3readiness.org for a compatibility overview) won't run otherwise. In fact, if you try to run some code developed for Python 2 with a Python 3 interpreter, it won't work. Major changes have been made to the newest version, and this has impacted past compatibility. So, please remember that there is no backward compatibility between Python 3 and 2.
In this book, in order to address a larger audience of readers and practitioners, we're going to adopt the Python 2 syntax for all our examples (at the time of writing this book, the latest release is 2.7.8). Since the differences amount to really minor changes, advanced users of Python 3 are encouraged to adapt and optimize the code to suit their favored version.

### Step-by-step installation

Novice data scientists who have never used Python (so, we figured out that they don't have it readily installed on their machines) need to first download the installer from the main website of the project, https://www.python.org/downloads/, and then install it on their local machine.

This section provides you with full control over what can be installed on your machine. This is very useful when you have to set up single machines to deal with different tasks in data science. Anyway, please be warned that a step-by-step installation really takes time and effort. Instead, installing a ready-made scientific distribution will lessen the burden of installation procedures and it may be well suited for first starting and learning because it saves you time and sometimes even trouble, though it will put a large number of packages (and we won't use most of them) on your computer all at once. Therefore, if you want to start immediately with an easy installation procedure, just skip this part and proceed to the next section, Scientific distributions.

Being a multiplatform programming language, you'll find installers for machines that either run on Windows or Unix-like operating systems. Please remember that some Linux distributions (such as Ubuntu) have Python 2 packeted in the repository, which makes the installation process even easier.

1. To open a python shell, type python in the terminal or click on the Python icon.

2. Then, to test the installation, run the following code in the Python interactive shell or REPL:

   ```
   >>> import sys
   >>> print sys.version_info
   ```

3. If a syntax error is raised, it means that you are running Python 3 instead of Python 2. Otherwise, if you don't experience an error and you can read that your Python version has the attribute `major=2`, then congratulations for running the right version of Python. You're now ready to move forward.
To clarify, when a command is given in the terminal command line, we prefix the command with `>`. Otherwise, if it's for the Python REPL, it's preceded by `>>>`.

### A glance at the essential Python packages

We mentioned that the two most relevant Python characteristics are its ability to integrate with other languages and its mature package system that is well embodied by PyPI (the Python Package Index; [https://pypi.python.org/pypi](https://pypi.python.org/pypi)), a common repository for a majority of Python packages.

The packages that we are now going to introduce are strongly analytical and will offer a complete Data Science Toolbox made up of highly optimized functions for working, optimal memory configuration, ready to achieve scripting operations with optimal performance. A walkthrough on how to install them is given in the following section.

Partially inspired by similar tools present in R and MATLAB environments, we will together explore how a few selected Python commands can allow you to efficiently handle data and then explore, transform, experiment, and learn from the same without having to write too much code or reinvent the wheel.

### NumPy

NumPy, which is Travis Oliphant's creation, is the true analytical workhorse of the Python language. It provides the user with multidimensional arrays, along with a large set of functions to operate a multiplicity of mathematical operations on these arrays. Arrays are blocks of data arranged along multiple dimensions, which implement mathematical vectors and matrices. Arrays are useful not just for storing data, but also for fast matrix operations (vectorization), which are indispensable when you wish to solve ad hoc data science problems.

- **Website**: [http://www.numpy.org/](http://www.numpy.org/)
- **Version at the time of print**: 1.9.1
- **Suggested install command**: `pip install numpy`

As a convention largely adopted by the Python community, when importing NumPy, it is suggested that you alias it as `np`:

```python
import numpy as np
```

We will be doing this throughout the course of this book.
SciPy
An original project by Travis Oliphant, Pearu Peterson, and Eric Jones, SciPy completes NumPy's functionalities, offering a larger variety of scientific algorithms for linear algebra, sparse matrices, signal and image processing, optimization, fast Fourier transformation, and much more.

- **Website:** [http://www.scipy.org/](http://www.scipy.org/)
- **Version at time of print:** 0.14.0
- **Suggested install command:** `pip install scipy`

pandas
The pandas package deals with everything that NumPy and SciPy cannot do. Thanks to its specific object data structures, DataFrames and Series, pandas allows you to handle complex tables of data of different types (which is something that NumPy's arrays cannot do) and time series. Thanks to Wes McKinney's creation, you will be able to easily and smoothly load data from a variety of sources. You can then slice, dice, handle missing elements, add, rename, aggregate, reshape, and finally visualize this data at your will.

- **Website:** [http://pandas.pydata.org/](http://pandas.pydata.org/)
- **Version at the time of print:** 0.15.2
- **Suggested install command:** `pip install pandas`

Conventionally, pandas is imported as `pd`:

```python
import pandas as pd
```

Scikit-learn
Started as part of the SciKits (SciPy Toolkits), Scikit-learn is the core of data science operations on Python. It offers all that you may need in terms of data preprocessing, supervised and unsupervised learning, model selection, validation, and error metrics. Expect us to talk at length about this package throughout this book. Scikit-learn started in 2007 as a Google Summer of Code project by David Cournapeau. Since 2013, it has been taken over by the researchers at INRA (French Institute for Research in Computer Science and Automation).

- **Website:** [http://scikit-learn.org/stable/](http://scikit-learn.org/stable/)
- **Version at the time of print:** 0.15.2
- **Suggested install command:** `pip install scikit-learn`
Note that the imported module is named `sklearn`.

**IPython**

A scientific approach requires the fast experimentation of different hypotheses in a reproducible fashion. IPython was created by Fernando Perez in order to address the need for an interactive Python command shell (which is based on shell, web browser, and the application interface), with graphical integration, customizable commands, rich history (in the JSON format), and computational parallelism for an enhanced performance. IPython is our favored choice throughout this book, and it is used to clearly and effectively illustrate operations with scripts and data and the consequent results.

- **Website**: [http://ipython.org/](http://ipython.org/)
- **Version at the time of print**: 2.3
- **Suggested install command**: `pip install "ipython[notebook]"

**Matplotlib**

Originally developed by John Hunter, matplotlib is the library that contains all the building blocks that are required to create quality plots from arrays and to visualize them interactively.

You can find all the MATLAB-like plotting frameworks inside the `pylab` module.

- **Website**: [http://matplotlib.org/](http://matplotlib.org/)
- **Version at the time of print**: 1.4.2
- **Suggested install command**: `pip install matplotlib`

You can simply import what you need for your visualization purposes with the following command:

```python
import matplotlib.pyplot as plt
```

**Downloading the example code**

You can download the example code files from your account at [http://www.packtpub.com](http://www.packtpub.com) for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit [http://www.packtpub.com/support](http://www.packtpub.com/support) and register to have the files e-mailed directly to you.
Statsmodels
Previously part of SciKits, statsmodels was thought to be a complement to SciPy statistical functions. It features generalized linear models, discrete choice models, time series analysis, and a series of descriptive statistics as well as parametric and nonparametric tests.

- **Website**: http://statsmodels.sourceforge.net/
- **Version at the time of print**: 0.6.0
- **Suggested install command**: pip install statsmodels

Beautiful Soup
Beautiful Soup, a creation of Leonard Richardson, is a great tool to scrap out data from HTML and XML files retrieved from the Internet. It works incredibly well, even in the case of tag soups (hence the name), which are collections of malformed, contradictory, and incorrect tags. After choosing your parser (basically, the HTML parser included in Python's standard library works fine), thanks to Beautiful Soup, you can navigate through the objects in the page and extract text, tables, and any other information that you may find useful.

- **Website**: http://www.crummy.com/software/BeautifulSoup/
- **Version at the time of print**: 4.3.2
- **Suggested install command**: pip install beautifulsoup4

Note that the imported module is named bs4.

NetworkX
Developed by the Los Alamos National Laboratory, NetworkX is a package specialized in the creation, manipulation, analysis, and graphical representation of real-life network data (it can easily operate with graphs made up of a million nodes and edges). Besides specialized data structures for graphs and fine visualization methods (2D and 3D), it provides the user with many standard graph measures and algorithms, such as the shortest path, centrality, components, communities, clustering, and PageRank. We will frequently use this package in Chapter 5, *Social Network Analysis*.

- **Website**: https://networkx.github.io/
- **Version at the time of print**: 1.9.1
- **Suggested install command**: pip install networkx
Conventionally, NetworkX is imported as `nx`:

```python
import networkx as nx
```

**NLTK**

The Natural Language Toolkit (NLTK) provides access to corpora and lexical resources and to a complete suit of functions for statistical Natural Language Processing (NLP), ranging from tokenizers to part-of-speech taggers and from tree models to named-entity recognition. Initially, the package was created by Steven Bird and Edward Loper as an NLP teaching infrastructure for CIS-530 at the University of Pennsylvania. It is a fantastic tool that you can use to prototype and build NLP systems.

- **Website**: [http://www.nltk.org/](http://www.nltk.org/)
- **Version at the time of print**: 3.0
- **Suggested install command**: `pip install nltk`

**Gensim**

Gensim, programmed by Radim Řehůřek, is an open source package that is suitable for the analysis of large textual collections with the help of parallel distributable online algorithms. Among advanced functionalities, it implements Latent Semantic Analysis (LSA), topic modeling by Latent Dirichlet Allocation (LDA), and Google's `word2vec`, a powerful algorithm that transforms text into vector features that can be used in supervised and unsupervised machine learning.

- **Website**: [http://radimrehurek.com/gensim/](http://radimrehurek.com/gensim/)
- **Version at the time of print**: 0.10.3
- **Suggested install command**: `pip install gensim`

**PyPy**

PyPy is not a package; it is an alternative implementation of Python 2.7.8 that supports most of the commonly used Python standard packages (unfortunately, NumPy is currently not fully supported). As an advantage, it offers enhanced speed and memory handling. Thus, it is very useful for heavy duty operations on large chunks of data and it should be part of your big data handling strategies.

- **Website**: [http://pypy.org/](http://pypy.org/)
- **Version at time of print**: 2.4.0
- **Download page**: [http://pypy.org/download.html](http://pypy.org/download.html)
The installation of packages

Python won't come bundled with all you need, unless you take a specific premade distribution. Therefore, to install the packages you need, you can either use pip or easy_install. These are the two tools that run in the command line and make the process of installation, upgrade, and removal of Python packages a breeze. To check which tools have been installed on your local machine, run the following command:

```bash
$> pip
```

Alternatively, you can also run the following command:

```bash
$> easy_install
```

If both these commands end with an error, you need to install any one of them. We recommend that you use pip because it is thought of as an improvement over easy_install. By the way, packages installed by pip can be uninstalled and if, by chance, your package installation fails, pip will leave your system clean.


The most recent versions of Python should already have pip installed by default. So, you may have it already installed on your system. If not, the safest way is to download the get-pip.py script from https://bootstrap.pypa.io/get-pip.py and then run it using the following:

```bash
$> python get-pip.py
```

The script will also install the setup tool from https://pypi.python.org/pypi/setuptools, which also contains easy_install.

You're now ready to install the packages you need in order to run the examples provided in this book. To install the generic package `<pk>`, you just need to run the following command:

```bash
$> pip install <pk>
```

Alternatively, you can also run the following command:

```bash
$> easy_install <pk>
```

After this, the package `<pk>` and all its dependencies will be downloaded and installed. If you're not sure whether a library has been installed or not, just try to import a module inside it. If the Python interpreter raises an `ImportError` error, it can be concluded that the package has not been installed.
This is what happens when the NumPy library has been installed:

```python
>>> import numpy
```

This is what happens if it's not installed:

```python
>>> import numpy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named numpy
```

In the latter case, you'll need to first install it through `pip` or `easy_install`.

```
Take care that you don't confuse packages with modules. With pip, you install a package; in Python, you import a module. Sometimes, the package and the module have the same name, but in many cases, they don't match. For example, the sklearn module is included in the package named Scikit-learn.
```

Finally, to search and browse the Python packages available for Python, take a look at [https://pypi.python.org](https://pypi.python.org).

### Package upgrades

More often than not, you will find yourself in a situation where you have to upgrade a package because the new version is either required by a dependency or has additional features that you would like to use. First, check the version of the library you have installed by glancing at the `__version__` attribute, as shown in the following example, `numpy`:

```python
>>> import numpy
>>> numpy.__version__ # 2 underscores before and after '1.9.0'
```

Now, if you want to update it to a newer release, say the 1.9.1 version, you can run the following command from the command line:

```
$> pip install -U numpy==1.9.1
```

Alternatively, you can also use the following command:

```
$> easy_install --upgrade numpy==1.9.1
```
Finally, if you're interested in upgrading it to the latest available version, simply run the following command:

```
$> pip install -U numpy
```

You can alternatively also run the following command:

```
$> easy_install --upgrade numpy
```

**Scientific distributions**

As you've read so far, creating a working environment is a time-consuming operation for a data scientist. You first need to install Python and then, one by one, you can install all the libraries that you will need (sometimes, the installation procedures may not go as smoothly as you'd hoped for earlier).

If you want to save time and effort and want to ensure that you have a fully working Python environment that is ready to use, you can just download, install, and use the scientific Python distribution. Apart from Python, they also include a variety of preinstalled packages, and sometimes, they even have additional tools and an IDE. A few of them are very well known among data scientists, and in the sections that follow, you will find some of the key features of each of these packages.

We suggest that you first promptly download and install a scientific distribution, such as Anaconda (which is the most complete one), and after practicing the examples in the book, decide to fully uninstall the distribution and set up Python alone, which can be accompanied by just the packages you need for your projects.

**Anaconda**

Anaconda ([https://store.continuum.io/cshop/anaconda](https://store.continuum.io/cshop/anaconda)) is a Python distribution offered by Continuum Analytics that includes nearly 200 packages, which include NumPy, SciPy, pandas, IPython, Matplotlib, Scikit-learn, and NLTK. It's a cross-platform distribution that can be installed on machines with other existing Python distributions and versions, and its base version is free. Additional add-ons that contain advanced features are charged separately. Anaconda introduces conda, a binary package manager, as a command-line tool to manage your package installations. As stated on the website, Anaconda's goal is to provide enterprise-ready Python distribution for large-scale processing, predictive analytics and scientific computing.
Enthought Canopy

Enthought Canopy (https://www.enthought.com/products/canopy/) is a Python distribution by Enthought, Inc. It includes more than 70 preinstalled packages, which include NumPy, SciPy, Matplotlib, IPython, and pandas. This distribution is targeted at engineers, data scientists, quantitative and data analysts, and enterprises. Its base version is free (which is named Canopy Express), but if you need advanced features, you have to buy a front version. It's a multiplatform distribution and its command-line install tool is canopy_cli.

PythonXY

PythonXY (https://code.google.com/p/pythonxy/) is a free, open source Python distribution maintained by the community. It includes a number of packages, which include NumPy, SciPy, NetworkX, IPython, and Scikit-learn. It also includes Spyder, an interactive development environment inspired by the MATLAB IDE. The distribution is free. It works only on Microsoft Windows, and its command-line installation tool is pip.

WinPython

WinPython (http://winpython.sourceforge.net) is also a free, open-source Python distribution maintained by the community. It is designed for scientists, and includes many packages such as NumPy, SciPy, Matplotlib, and IPython. It also includes Spyder as an IDE. It is free and portable (you can put it in any directory, or even in a USB flash drive). It works only on Microsoft Windows, and its command-line tool is the WinPython Package Manager (WPPM).

Introducing IPython

IPython is a special tool for interactive tasks, which contains special commands that help the developer better understand the code that they are currently writing. These are the commands:

- `<object>??`: This prints a detailed description (with ?? being even more verbose) of the `<object>`
- `%<function>`: This uses the special `<magic function>`
Let's demonstrate the usage of these commands with an example. We first start the interactive console with the `ipython` command that is used to run IPython, as shown here:

```
$> ipython
Python 2.7.6 (default, Sep 9 2014, 15:04:36)
Type "copyright", "credits" or "license" for more information.
IPython 2.3.1 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
In [1]: obj1 = range(10)
```

Then, in the first line of code, which is marked by IPython as [1], we create a list of 10 numbers (from 0 to 9), assigning the output to an object named `obj1`:

```
In [2]: obj1?
Type:        list
String form: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Length:      10
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items
In [3]: %timeit x=100
10000000 loops, best of 3: 23.4 ns per loop
In [4]: %quickref
```

In line [3], we use the magic function `timeit` to a Python assignment (`x=100`). The `timeit` function runs this instruction many times and stores the computational time needed to execute it. Finally, it prints the average time that was taken to run the Python function.

We complete the overview with a list of all the possible IPython special functions by running the helper function `quickref`, as shown in line [4].
As you noticed, each time we use IPython, we have an input cell and optionally, an output cell, if there is something that has to be printed on stdout. Each input is numbered, so it can be referenced inside the IPython environment itself. For our purposes, we don't need to provide such references in the code of the book. Therefore, we will just report inputs and outputs without their numbers. However, we'll use the generic In: and Out: notations to point out the input and output cells. Just copy the commands after In: to your own IPython cell and expect an output that will be reported on the following Out:

Therefore, the basic notations will be:

- The In: command
- The Out: output (wherever it is present and useful to be reported in the book)

Otherwise, if we expect you to operate directly on the Python console, we will use the following form:

```python
>>> command
```

Wherever necessary, the command-line input and output will be written as follows:

```bash
$> command
```

Moreover, to run the bash command in the IPython console, prefix it with a "!":

```bash
In: !ls
Applications    Google Drive    Public          Desktop
Develop
Pictures        env             temp
...
In: !pwd
/Users/mycomputer
```

### The IPython Notebook

The main goal of the IPython Notebook is easy storytelling. Storytelling is essential in data science because you must have the power to do the following:

- See intermediate (debugging) results for each step of the algorithm you're developing
- Run only some sections (or cells) of the code
- Store intermediate results and have the ability to version them
- Present your work (this will be a combination of text, code, and images)
Here comes IPython; it actually implements all the preceding actions.

1. To launch the IPython Notebook, run the following command:
   
   $> ipython notebook

2. A web browser window will pop up on your desktop, backed by an
   IPython server instance. This is the how the main window looks:

   ![IPython Notebook Interface]

3. Then, click on **New Notebook**. A new window will open, as shown in the
   following screenshot:

   ![New Notebook Window]

   This is the web app that you’ll use to compose your story. It's very similar to a Python
   IDE, with the bottom section (where you can write the code) composed of cells.

   A cell can be either a piece of text (eventually formatted with a markup language)
   or a piece of code. In the second case, you have the ability to run the code, and any
   eventual output (the standard output) will be placed under the cell. The following
   is a very simple example of the same:
In: import random
    a = random.randint(0, 100)
    a
Out: 16
In: a*2
Out: 32

In the first cell, which is denoted by \texttt{In:}, we import the random module, assign a random value between 0 and 100 to the variable \texttt{a}, and print the value. When this cell is run, the output, which is denoted as \texttt{Out:}, is the random number. Then, in the next cell, we will just print the double of the value of the variable \texttt{a}.

As you can see, it's a great tool to debug and decide which parameter is best for a given operation. Now, what happens if we run the code in the first cell? Will the output of the second cell be modified since \texttt{a} is different? Actually, no. Each cell is independent and autonomous. In fact, after we run the code in the first cell, we fall in this inconsistent status:

In: import random
    a = random.randint(0, 100)
    a
Out: 56
In: a*2
Out: 32

Also note that the number in the squared parenthesis has changed (from 1 to 3) since it's the third executed command (and its output) from the time the notebook started. Since each cell is autonomous, by looking at these numbers, you can understand their order of execution.

IPython is a simple, flexible, and powerful tool. However, as seen in the preceding example, you must note that when you update a variable that is going to be used later on in your Notebook, remember to run all the cells following the updated code so that you have a consistent state.

When you save an IPython notebook, the resulting .ipynb file is JSON formatted, and it contains all the cells and their content, plus the output. This makes things easier because you don’t need to run the code to see the notebook (actually, you also don’t need to have Python and its set of toolkits installed). This is very handy, especially when you have pictures featured in the output and some very time-consuming routines in the code. A downside of using the IPython Notebook is that its file format, which is JSON structured, cannot be easily read by humans. In fact, it contains images, code, text, and so on.
Now, let's discuss a data science related example (don't worry about understanding it completely):

In:
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor

In the following cell, some Python modules are imported:

In:
boston_dataset = datasets.load_boston()
X_full = boston_dataset.data
Y = boston_dataset.target
print X_full.shape
print Y.shape
Out:
(506, 13)
(506,)

Then, in cell [2], the dataset is loaded and an indication of its shape is shown. The dataset contains 506 house values that were sold in the suburbs of Boston, along with their respective data arranged in columns. Each column of the data represents a feature. A feature is a characteristic property of the observation. Machine learning uses features to establish models that can turn them into predictions. If you are from a statistical background, you can add features that can be intended as variables (values that vary with respect to the observations).

To see a complete description of the dataset, print boston_dataset.DESCR.

After loading the observations and their features, in order to provide a demonstration of how IPython can effectively support the development of data science solutions, we will perform some transformations and analysis on the dataset. We will use classes, such as SelectKBest, and methods, such as .get_support() or .fit(). Don't worry if these are not clear to you now; they will all be covered extensively later in this book. Try to run the following code:

In:
selector = SelectKBest(f_regression, k=1)
In our example, as $X$ increases, $Y$ decreases. However, this does not happen at a constant rate, because the rate of change is intense up to a certain $X$ value but then it decreases and becomes constant. This is a condition of nonlinearity, and we can furthermore visualize it using a regression model. This model hypothesizes that the relationship between $X$ and $Y$ is linear in the form of $y = a + bX$. Its $a$ and $b$ parameters are estimated according to a certain criteria.
First Steps

In the fourth cell, we scatter the input and output values for this problem:

In:

```python
regressor = LinearRegression(normalize=True)
regressor.fit(X, Y)
plt.scatter(X, Y, color='black')
plt.plot(X, regressor.predict(X), color='blue', linewidth=3)
plt.show()
```

In the next cell, we create a regressor (a simple linear regression with feature normalization), train the regressor, and finally plot the best linear relation (that's the linear model of the regressor) between the input and output. Clearly, the linear model is an approximation that is not working well. We have two possible roads that we can follow at this point. We can transform the variables in order to make their relationship linear, or we can use a nonlinear model. **Support Vector Machine (SVM)** is a class of models that can easily solve nonlinearities. Also, **Random Forests** is another model for the automatic solving of similar problems. Let's see them in action in IPython:

In:

```python
regressor = SVR()
regressor.fit(X, Y)
plt.scatter(X, Y, color='black')
plt.scatter(X, regressor.predict(X), color='blue', linewidth=3)
plt.show()
```
In:
```python
regressor = RandomForestRegressor()
regressor.fit(X, Y)
plt.scatter(X, Y, color='black');
plt.scatter(X, regressor.predict(X), color='blue', linewidth=3)
plt.show()
```

Finally, in the last two cells, we will repeat the same procedure. This time we will use two nonlinear approaches: an SVM and a Random Forest based regressor.
Having been written down on the IPython interface, this demonstrative code solves the nonlinearity problem. At this point, it is very easy to change the selected feature, regressor, the number of features we use to train the model, and so on, by simply modifying the cells where the script is. Everything can be done interactively, and according to the results we see, we can decide both what should be kept or changed and what is to be done next.

Datasets and code used in the book
As we progress through the concepts presented in this book, in order to facilitate the reader's understanding, learning, and memorizing processes, we will illustrate practical and effective data science Python applications on various explicative datasets. The reader will always be able to immediately replicate, modify, and experiment with the proposed instructions and scripts on the data that we will use in this book.

As for the code that you are going to find in this book, we will limit our discussions to the most essential commands in order to inspire you from the beginning of your data science journey with Python to do more with less by leveraging key functions from the packages we presented beforehand.

Given our previous introduction, we will present the code to be run interactively as it appears on an IPython console or Notebook.

All the presented code will be offered in Notebooks, which is available on the Packt Publishing website (as pointed out in the Preface). As for the data, we will provide different examples of datasets.

Scikit-learn toy datasets
The Scikit-learn toy dataset is embedded in the Scikit-learn package. Such datasets can easily be directly loaded into Python by the import command, and they don't require any download from any external Internet repository. Some examples of this type of dataset are the Iris, Boston, and Digits datasets, to name the principal ones mentioned in uncountable publications and books, and a few other classic ones for classification and regression.

Structured in a dictionary-like object, besides the features and target variables, they offer complete descriptions and contextualization of the data itself.

For instance, to load the Iris dataset, enter the following commands:

```python
In: from sklearn import datasets
In: iris = datasets.load_iris()
```
After loading, we can explore the data description and understand how the features and targets are stored. Basically, all Scikit-learn datasets present the following methods:

- `.DESCR`: This provides a general description of the dataset
- `.data`: This contains all the features
- `.feature_names`: This reports the names of the features
- `.target`: This contains the target values expressed as values or numbered classes
- `.target_names`: This reports the names of the classes in the target
- `.shape`: This is a method that you can apply to both `.data` and `.target`; it reports the number of observations (the first value) and features (the second value, if present) that are present

Now, let's just try to implement them (no output is reported, but the print commands will provide you with plenty of information):

```python
In: print iris.DESCR
In: print iris.data
In: print iris.data.shape
In: print iris.feature_names
In: print iris.target
In: print iris.target.shape
In: print iris.target_names
```

Now, you should know something more about the dataset—about how many examples and variables are present and what their names are.

Notice that the main data structures that are enclosed in the iris object are the two arrays, data and target:

```python
In: print iris.data
Out: <type 'numpy.ndarray'>
```

Iris.data offers the numeric values of the variables named `sepal length`, `sepal width`, `petal length`, and `petal width` arranged in a matrix form (150,4), where 150 is the number of observations and 4 is the number of features. The order of the variables is the order presented in `iris.feature_names`.

Iris.target is a vector of integer values, where each number represents a distinct class (refer to the content of `target_names`; each class name is related to its index number and `setosa`, which is the zero element of the list, is represented as 0 in the target vector).
The Iris flower dataset was first used in 1936 by Ronald Fisher, who was one of the fathers of modern statistical analysis, in order to demonstrate the functionality of linear discriminant analysis on a small set of empirically verifiable examples (each of the 150 data points represented iris flowers). These examples were arranged into tree balanced species classes (each class consisted of one-third of the examples) and were provided with four metric descriptive variables that, when combined, were able to separate the classes.

The advantage of using such a dataset is that it is very easy to load, handle, and explore for different purposes, from supervised learning to graphical representation. Modeling activities take almost no time on any computer, no matter what its specifications are. Moreover, the relationship between the classes and the role of the explicative variables are well known. So, the task is challenging, but it is not arduous.

For example, let’s just observe how classes can be easily separated when you wish to combine at least two of the four available variables by using a scatterplot matrix.

Scatterplot matrices are arranged in a matrix format, whose columns and rows are the dataset variables. The elements of the matrix contain single scatterplots whose x values are determined by the row variable of the matrix and y values by the column variable. The diagonal elements of the matrix may contain a distribution histogram or some other univariate representation of the variable at the same time in its row and column.

The pandas library offers an off-the-shelf function to quickly make up scatterplot matrices and start exploring relationship and distributions between the quantitative variables in a dataset.

In:
import pandas as pd
import numpy as np
In: colors = list()
In: palette = {0: "red", 1: "green", 2: "blue"}
In:
for c in np.nditer(iris.target): colors.append(palette[int(c)])
    # using the palette dictionary, we convert
    # each numeric class into a color string
In: dataframe = pd.DataFrame(iris.data,
columns=iris.feature_names)
In: scatterplot = pd.scatter_matrix(dataframe, alpha=0.3,
figsize=(10, 10), diagonal='hist', color=colors, marker='o',
grid=True)
We encourage you to experiment a lot with this dataset and with similar ones before you work on other complex real data, because the advantage of focusing on an accessible, non-trivial data problem is that it can help you to quickly build your foundations on data science.

After a while, anyway, though useful and interesting for your learning activities, toy datasets will start limiting the variety of different experimentations that you can achieve. In spite of the insight provided, in order to progress, you’ll need to gain access to complex and realistic data science topics. We will, therefore, have to resort to some external data.
The MLdata.org public repository

The second type of example dataset that we will present can be downloaded directly from the machine learning dataset repository, or from the LIBSVM data website. Contrary to the previous dataset, in this case, you will need to have access to the Internet.

First of all, mldata.org is a public repository for machine learning datasets that is hosted by the TU Berlin University and supported by Pattern Analysis, Statistical Modelling, and Computational Learning (PASCAL), a network funded by the European Union.

For example, if you need to download all the data related to earthquakes since 1972 as reported by the United States Geological Survey, in order to analyze the data to search for predictive patterns you will find the data repository at http://mldata.org/repository/data/viewslug/global-earthquakes/ (here, you will find a detailed description of the data).

Note that the directory that contains the dataset is global-earthquakes; you can directly obtain the data using the following commands:

In: from sklearn.datasets import fetch_mldata
In: earthquakes = fetch_mldata('global-earthquakes')
In: print earthquakes.data
In: print earthquakes.data.shape
Out: (59209L, 4L)

As in the case of the Scikit-learn package toy dataset, the obtained object is a complex dictionary-like structure, where your predictive variables are earthquakes.data and your target to be predicted is earthquakes.target. This being the real data, in this case, you will have quite a lot of examples and just a few variables available.

LIBSVM data examples

LIBSVM Data (http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/) is a page-gathering data from many other collections. It offers different regression, binary, and multilabel classification datasets stored in the LIBSVM format. This repository is quite interesting if you wish to experiment with the support vector machine's algorithm.

If you want to load a dataset, first go to the page where you wish to visualize the data. In this case, visit http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/a1a and take down the address. Then, you can proceed by performing a direct download:
In: import urllib2
In: target_page = 'http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/a1a'
In: a2a = urllib2.urlopen(target_page)
In: from sklearn.datasets import load_svmlight_file
In: X_train, y_train = load_svmlight_file(a2a)
In: print X_train.shape, y_train.shape
Out: (2265, 119) (2265L,)

In return, you will get two single objects: a set of training examples in a sparse matrix format and an array of responses.

**Loading data directly from CSV or text files**

Sometimes, you may have to download the datasets directly from their repository using a web browser or a `wget` command.

If you have already downloaded and unpacked the data (if necessary) into your working directory, the simplest way to load your data and start working is offered by the NumPy and the pandas library with their respective `loadtxt` and `read_csv` functions.

For instance, if you intend to analyze the Boston housing data and use the version present at http://mldata.org/repository/data/viewslug/regression-datasets-housing/, you first have to download the regression-datasets-housing.csv file in your local directory.

Since the variables in the dataset are all numeric (13 continuous and one binary), the fastest way to load and start using it is by trying out the NumPy function `loadtxt` and directly loading all the data into an array.

Even in real-life datasets, you will often find mixed types of variables, which can be addressed by pandas.read_table or pandas.read_csv. Data can then be extracted by the `values` method; `loadtxt` can save a lot of memory if your data is already numeric since it does not require any in-memory duplication.

In: housing = np.loadtxt('regression-datasets-housing.csv',delimiter=',')
In: print type(housing)
Out: <type 'numpy.ndarray'>
In: print housing.shape
Out: (506L, 14L)
The `loadtxt` function expects, by default, tabulation as a separator between the values on a file. If the separator is a colon (:) or a semi-colon (;), you have to explicit it using the parameter delimiter.

```python
>>> import numpy as np
>>> type(np.loadtxt)
<type 'function'>
>>> help(np.loadtxt)
Help on function loadtxt in module numpy.lib.npyio.
```

Another important default parameter is `dtype`, which is set to `float`. This means that `loadtxt` will force all the loaded data to be converted into a floating point number.

If you need to determinate a different type (for example, an `int`), you have to declare it beforehand.

For instance, if you want to convert numeric data to `int`, use the following code:

```python
In: housing_int = np.loadtxt('regression-datasets-housing.csv', delimiter=',', dtype=int)
```

Printing the first three elements of the row of the `housing` and `housing_int` arrays can help you understand the difference:

```python
In: print housing[0,:3], '
', housing_int[0,:3]
Out: [  6.32000000e-03   1.80000000e+01   2.31000000e+00] [ 0 18  2]
```

Frequently, though not always the case in our example, the data on files feature in the first line a textual header that contains the name of the variables. In this situation, the parameter that is skipped will point out the row in the `loadtxt` file from where it will start reading the data. Being the header on row 0 (in Python, counting always starts from 0), parameter `skip=1` will save the day and allow you to avoid an error and fail to load your data.

The situation would be slightly different if you were to download the Iris dataset, which is present at `http://mldata.org/repository/data/viewslug/datasets-uci-iris/`. In fact, this dataset presents a qualitative target variable, `class`, which is a string that expresses the iris species. Specifically, it’s a categorical variable with four levels.
Therefore, if you were to use the `loadtxt` function, you will get a value error due to the fact that an array must have all its elements of the same type. The variable class is string, whereas the other variables are constituted of floating point values.

How to proceed? The pandas library offers the solution, thanks to its DataFrame data structure that can easily handle datasets in a matrix form (row per columns) that is made up of different types of variables.

First of all, just download the `datasets-uci-iris.csv` file and have it saved in your local directory.

At this point, using pandas' `read_csv` is quite straightforward:

```python
In: iris_filename = 'datasets-uci-iris.csv'
In: iris = pd.read_csv(iris_filename, sep=',', decimal='.', header=None, names=['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'target'])
In: print type(iris)
Out: <class 'pandas.core.frame.DataFrame'>
```

Apart from the filename, you can specify the separator (`sep`), the way the decimal points are expressed (`decimal`), whether there is a header (in this case, `header=None`; normally, if you have a header, then `header=0`), and the name of the variable — where there is one (you can use a list; otherwise, pandas will provide some automatic naming).

Also, we have defined names that use single words (instead of spaces, we used underscores). Thus, we can later directly extract single variables by calling them as we do for methods; for instance, `iris.sepal_length` will extract the sepal length data.

If, at this point, you need to convert the pandas DataFrame into a couple of NumPy arrays that contain the data and target values, this can be easily done in a couple of commands:

```python
In: iris_data = iris.values[:, :4]
In: iris_target, iris_target_labels = pd.factorize(iris.target)
In: print iris_data.shape, iris_target.shape
Out: (150L, 4L) (150L,)
```
Scikit-learn sample generators

As a last learning resource, Scikit-learn also offers the possibility to quickly create synthetic datasets for regression, binary and multilabel classification, cluster analysis, and dimensionality reduction.

The main advantage of recurring to synthetic data lies in its instantaneous creation in the working memory of your Python console. It is, therefore, possible to create bigger data examples without having to engage in long downloading sessions from the Internet (and saving a lot of stuff on your disk).

For example, you may need to work on a million example classification problem:

In: from sklearn import datasets # We just import the "datasets" module
In: X,y = datasets.make_classification(n_samples=10**6, n_features=10, random_state=101)
In: print X.shape, y.shape
Out: (1000000L, 10L) (1000000L,)

After importing just the datasets module, we ask, using the make_classification command, for 1 million examples (the n_samples parameter) and 10 useful features (n_features). The random_state should be 101, so we can be assured that we can replicate the same datasets at a different time and in a different machine.

For instance, you can type the following command:

$> datasets.make_classification(1, n_features=4, random_state=101)

This will always give you the following output:

(array([[3.1994186, 2.39469384, 2.35882002, 1.40145585]], array([0]))

No matter what the computer and the specific situation is, random_state assures deterministic results that make your experimentations perfectly replicable.

Defining the random_state parameter using a specific integer number (in this case 101, but it may be any number that you prefer or find useful) allows the easy replication of the same dataset on your machine, the way it is set up, on different operating systems, and on different machines.

By the way, did it take too long?
On an i3-2330M CPU @ 2.20GHz machine, it takes:

```python
In: %timeit X, y = datasets.make_classification(n_samples=10**6,
    n_features=10, random_state=101)
Out: 1 loops, best of 3: 2.17 s per loop
```

If it doesn't seem so also on your machine and if you are ready, having set up and tested everything up to this point, we can start our data science journey.

**Summary**

In this short introductory chapter, we installed everything that we will be using throughout this book, even the examples, which were installed either directly or by using a scientific distribution. We also introduced you to IPython and demonstrated how you can have access to the data run in the tutorials.

In the next chapter, *Data Munging*, we will have an overview of the data science pipeline and explore all the key tools to handle and prepare data before you apply any learning algorithm and set up your hypothesis experimentation schedule.
Where to buy this book

You can buy Python Data Science Essentials from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.