

Python FOR BEGINNERS

Comprehensive Guide to
the Basics of Programming,
Machine Learning,
Data Science and Analysis
with Python.



Alex Campbell

Python for Beginners

**Comprehensive Guide to the Basics of
Programming, Machine Learning, Data Science
and Analysis with Python.**

Alex Campbell

© Copyright 2021 by Alex Campbell - All rights reserved.

The contents of this book may not be reproduced, duplicated, or transmitted without direct written permission from the author.

Under no circumstances will any legal responsibility or blame be held against the publisher for any reparation, damages, or monetary loss due to the information herein, either directly or indirectly.

Legal Notice:

You cannot amend, distribute, sell, use, quote, or paraphrase any part of the content within this book without the consent of the author.

Disclaimer Notice:

Please note the information contained within this document is for educational and entertainment purposes only. No warranties of any kind are expressed or implied. Readers acknowledge that the author is not engaging in the rendering of legal, financial, medical, or professional advice. Please consult a licensed professional before attempting any techniques outlined in this book.

By reading this document, the reader agrees that under no circumstances are the author responsible for any losses, direct or indirect, which are incurred as a result of the use of the information contained within this document, including, but not limited to, —errors, omissions, or inaccuracies.

Table of Contents

Python for Beginners

BOOK 1- Python Computer Programming: Simple Step-By-Step Introduction to the Python Object-Oriented Programming. Quick Start Guide for Beginners.

[INTRODUCTION](#)

[WHAT IS PYTHON?](#)

[INSTALLING PYTHON 3 - WINDOWS](#)

[INSTALLING PYTHON - MAC OSX](#)

[INSTALLING PYTHON - LINUX](#)

[RUNNING PYTHON PROGRAMS](#)

[DATA TYPES AND VARIABLES](#)

[Assign a Variable With a Value](#)

[Using Comments](#)

[Simultaneous Assignment](#)

[Data Types](#)

[Receiving An Input from The Console](#)

[Importing a Module](#)

[PYTHON NUMBERS](#)

[Determining Types](#)

[Python Operators](#)

[Operator Precedence](#)

[Augmented Assignment Operator](#)

[PYTHON STRINGS](#)

[Creating Strings](#)

[String Operations](#)

[Slicing Strings](#)

[ord\(\) and chr\(\) Functions](#)

[Python String Functions](#)

[In and Not In Operators](#)

[String Comparison](#)

[String Iteration Using a For Loop](#)

[Testing Strings](#)

[Searching for Substrings](#)

[Converting Strings](#)

[PYTHON LISTS](#)

[Creating Lists](#)

[Accessing List Elements](#)

[Common List Operations](#)

[List Slicing](#)

[+ and * Operators in List](#)

[in or not in Operator](#)

[Using a For Loop to Traverse a List](#)

[List Comprehension](#)

[PYTHON DICTIONARIES](#)

[Creating a Dictionary](#)

[Retrieving, Modifying And Adding Elements](#)

[Deleting Items](#)

[Looping Items](#)

[Find the Dictionary Length](#)

[in or not in Operators](#)

[Equality Tests](#)

[Dictionary Methods](#)

[PYTHON TUPLES](#)

[Creating A Tuple](#)

[Tuples Functions](#)

[Iterating Through Tuples](#)

[Slicing Tuples](#)

[In And Not In Operator](#)

[DATATYPE CONVERSION](#)

[Converting an int to a Float](#)

[Converting a Float to an int](#)

[Converting a String to an int](#)

[Converting a Number to a String](#)

[Rounding Numbers](#)

[PYTHON CONTROL STATEMENTS](#)

[Nested if Statements](#)

[PYTHON FUNCTIONS](#)

[Creating Functions](#)

[Function With A Return Value](#)

[Global Variables Vs. Local Variables](#)

[Arguments With Default Values](#)

[Keyword Arguments](#)

[Combining Keyword and Positional Arguments](#)

[Multiple Values Returned From Function](#)

[PYTHON LOOPS](#)

[The for Loop](#)

[range\(a, b\) Function](#)

[The while Loop](#)

[The break Statement](#)

[The continue Statement](#)

[MATHEMATICAL FUNCTIONS](#)

[GENERATING RANDOM NUMBERS](#)

[FILE HANDLING](#)

[Open a File](#)

[Close a File](#)

[Append Data](#)

[Using a For Loop](#)

[Reading and Writing - Binary](#)

[OBJECTS AND CLASSES](#)

[Define a Class](#)

[Self](#)

[Objects Created from Classes](#)

[How to Hide Data fields](#)

[OPERATOR OVERLOADING](#)

[Inheritance](#)

[Multiple Inheritance](#)

[Method Overriding](#)

[isinstance\(\) Function](#)

EXCEPTION HANDLING

[Raising Exceptions](#)

[Exception Objects](#)

[Create A Custom Exception Class](#)

[Using A Custom Exception Class](#)

MODULES

[Creating a Module](#)

[Using the From Statement With Import](#)

[Using the dir\(\) Method](#)

BEGINNER TIPS FOR LEARNING PYTHON

CONCLUSION

REFERENCES

BOOK 2 - Python Machine Learning: Complete and Clear Introduction to the Basics of Machine Learning with Python. Comprehensive Guide to Data Science and Analytics.

INTRODUCTION

CHAPTER ONE: AN OVERVIEW OF MACHINE LEARNING

[Machine Learning Categories](#)

[Examples of Machine Learning Applications](#)

[Classification: Predicting Discrete Labels](#)

[Regression: Predicting Continuous Labels](#)

[Clustering: Inferring the Labels on Unlabeled Data](#)

[Dimensionality Reduction: Inferring the Structure of Unlabeled Data](#)

CHAPTER TWO: REGRESSION MACHINE LEARNING MODELS

[When is Regression Required?](#)

[Different Types of Regression Techniques](#)

[Linear Regression](#)

[Logistic Regression](#)

[Polynomial Regression](#)

[Stepwise Regression](#)

[Ridge Regression](#)

[Lasso Regression](#)

[ElasticNet Regression](#)

CHAPTER THREE: CLASSIFICATION MACHINE LEARNING MODELS

[Different Classification Algorithms for Python](#)

[Logistic Regression](#)

[Naïve Bayes](#)

[Stochastic Gradient Descent](#)

[K-Nearest Neighbors](#)

[Decision Tree](#)

[Random Forest](#)

[Support Vector Machine](#)

[Accuracy](#)

CHAPTER FOUR: UNSUPERVISED MACHINE LEARNING ALGORITHMS

[Why Choose Unsupervised Learning?](#)

[Different Types of Unsupervised Machine Learning](#)

[Clustering](#)

[Types of Clustering](#)

[Supervised Machine Learning vs. Unsupervised Machine Learning](#)

[Unsupervised Machine Learning Applications](#)

[The Disadvantages of Using Unsupervised Learning](#)

CHAPTER FIVE: YOUR FIRST MACHINE LEARNING PROJECT

[The Hello World of Python Machine Learning](#)

CHAPTER SIX: AN INTRODUCTION TO DATA SCIENCE

[What is Data Science?](#)

[Using Pandas for Exploratory Analysis](#)

[Using Pandas for Data Wrangling](#)

[Building the Model](#)

[How to Learn Python For Data Science](#)

[Step 1: Learn The Fundamentals of the Python Language](#)

[Step 2: Do Some Small Python Projects](#)

[Step 3: Learn the Python Data Science Libraries](#)

[Step 4: As You Learn Python, Build Up a Data Science Portfolio](#)

[Step 5: Apply Some Advanced Techniques in Data Science](#)

CHAPTER SEVEN – TEN THINGS YOU SHOULD KNOW ABOUT MACHINE LEARNING

CONCLUSION

BOOK 3 - Python Data Analysis: Comprehensive Guide to Data Science, Analytics and Metrics with Python.

INTRODUCTION

CHAPTER 1: BASICS OF DATA SCIENCE

[What is Data Science?](#)

[Career Scope and Impact of Data Science Using Python](#)

CHAPTER 2: VARIOUS ASPECTS OF DATA SCIENCE

[Steps Involved in a Data Science Project](#)

[Defining the Problem](#)

[Collecting Data from Sources](#)

[Data Processing](#)

[Feature Engineering](#)

[Algorithm Selection](#)

[Hyperparameter Tuning](#)

[Data Visualization](#)

[Interpretation of Results](#)

[How to Solve the Problems with Python](#)

CHAPTER 3: PYTHON EXPLORATORY ANALYSIS WITH PANDAS

Understanding DataFrames and Series

Data Set For Practice – A Loan Prediction Problem

[Beginning with Exploration](#)

[How to Import the Data Set and Libraries](#)

[Quick Data Exploration](#)

[Distribution Analysis](#)

[Categorical Variable Analysis](#)

Using Pandas for Data Munging in Python

[Checking for Missing Values](#)

[Treating Extreme Values in a Distribution](#)

CHAPTER 4: BASICS OF PYTHON FOR DATA ANALYSIS

Why Do We Use Python v.3 And Not V.2|?

Data Structures of Python

Data Analysis in Python Using Pandas

Data Science Using Python: Start Instantly

Read the Tutorial Carefully

[Anaconda](#)

[Jupyter Notebook](#)

[Open New Notebook](#)

[Math Calculations](#)

[Data Importing](#)

[Importing Dataset](#)

[Exploration](#)

[Clean the Dataset](#)

[Features](#)

[Develop an Easy Model](#)

Using Matplotlib

CHAPTER 5: METRICS IN PYTHON ALONG WITH DEMO

Changes when a System shows unusual Behavior

What Are Metrics And How Many Types Of Metrics Are There?

Counters

Gauges

Histograms or Timers

Demo 1

[Mean](#)

[Median](#)

[Percentile](#)

[Histogram and Cumulative Histogram](#)

Demo 2

[Network Applications](#)

[Long Processes](#)

[How to Monitor in a Python Application](#)

CHAPTER 6: HOW TO BUILD A PREDICTIVE MODEL IN PYTHON

[Logistic Regression](#)

[Decision Tree](#)

[Data Prediction and Analysis](#)

CHAPTER 7: INCOME INCREMENT USING DATA SCIENCE WITH PYTHON

[Search Options](#)

[Churn Prediction](#)

[Churn Categories](#)

[Data Science and Python: The Essential Relationship](#)

[Learning Python for Data Science](#)

CONCLUSION

BOOK 4 - Python for Data Science: Comprehensive Guide to Data Science with Python.

INTRODUCTION

CHAPTER 1: DATA ANALYSIS? DATA SCIENCE? OR MACHINE LEARNING?

[Machine Learning and Data Analysis Limitations](#)

[The Potential and the Implications](#)

[CHAPTER 2: GET AND PROCESS YOUR DATA](#)

[CSV Files](#)

[Internal Data](#)

[CHAPTER 3: DATA VISUALIZATION](#)

[Importing and Using Matplotlib](#)

[Supervised and Unsupervised Learning](#)

[CHAPTER 4: A DEEPER LOOK AT REGRESSION](#)

[Multiple Linear Regression](#)

[Decision Tree Regression](#)

[Random Forest Regression](#)

[CHAPTER 5: DIGGING INTO CLASSIFICATION](#)

[Logistic Regression](#)

[K-Nearest Neighbors](#)

[Decision Tree Classification](#)

[Random Forest Classification](#)

[CHAPTER 6: A LOOK AT CLUSTERING](#)

[Clustering Goals and Uses](#)

[K-Means Clustering](#)

[Anomaly Detection](#)

[CHAPTER 7: WHAT IS REINFORCEMENT LEARNING?](#)

[Reinforcement Learning Compared with Supervised and Unsupervised Learning](#)

[How to Apply Reinforcement Learning](#)

CHAPTER 8: THE ARTIFICIAL NEURAL NETWORK

Imitating the Human Brain

CONCLUSION

REFERENCES

Python Computer Programming

**Simple Step-By-Step Introduction to the
Python Object-Oriented Programming. Quick
Start Guide for Beginners.**

Alex Campbell

Introduction

Python is one of the most powerful computer programming languages of all time, for several reasons we'll discuss in the first section. The syntax is simple to learn and use and, compared to other programming languages, you often don't need to write so much code. The sheer simplicity of the language helps programmers write more and develop programs that are more complex in less time.

This guide provides all you need to master the basics of programming with Python. I have kept it deliberately simple – it is a quick-start guide, after all. I have provided plenty of coding examples to show you how the syntax works, too, along with a guide on installing Python on Windows, Mac, and Linux systems. I finish with some useful tips on helping you to code better.

By the end of the guide, you will have a deeper understanding of the Python language, a stepping stone from which to take your learning further.

Please note that we are using Python 3 in this guide, not Python 2, as many similar guides do.

Are you ready to become a computer programmer? Let's get started on this wonderful journey.

What is Python?

Python is the brainchild of Guido Van Rossum and is a simple, all-purpose programming language. It is known for its easy syntax, easy-to-read code, and, for complete beginners to computer programming, it is one of the best to start with. Learning Python allows you to do many things, and there are several advantages to using it.

- **Interpreted Language**

That means an interpreter is used to parse any program you write, one line at a time. With compiled languages, such as the C languages, a compiler is used to compile the code before it is executed or run.

So, what is the difference?

Interpreted computer languages are a little slower than compiled languages and don't offer the performance benefits of compiled languages. However, the latter are too difficult and complicated for complete beginners to learn, and they require you to write every function, even the most basic ones. Advanced programming tasks sometimes require you to encapsulate data by creating data structures too. In a compiled language, therefore, all the small details must be taken care of before you can even begin to solve the problem the code is written for. With interpreted languages, you don't define data structures, and you don't define any of the smaller functions because Python does it all for you.

Plus, you can make use of the multitude of libraries included with Python without having to write reams of code.

- **Dynamically Typed**

Unlike compiled languages, with Python, there is no need to define the data type of a variable before you need it. Python can infer it automatically based on the value type in the variable.

For example:

```
myvar = "Good Morning Python."
```

In the above code line, we have a variable called myvar, and a string of “Good Morning Python” is assigned to it. That means myvar has a data type of a string.

Also, note that Python does not require a semicolon (;) at the end of a statement. Compiled languages do require this, and it is one of the most significant areas where errors occur – forgetting to add semicolons can throw up all kinds of errors in the code.

Let’s say that, a bit later, we wanted a value of 1 assigned to myvar:

```
myvar = 1
```

What does that do?

It makes the variable into an int (integer) type.

- **Strongly Typed**

If you have written any programs in JavaScript or PHP, you probably spotted that both will automatically convert the data in one data type into other types.

For example:

JavaScript

```
1 + "2"
```

Would be treated as '12.'

1 gets converted into a string type and is concatenated (joined) to 2, thus the result of '12' – that is a string. Python does not allow automatic conversions so the same code:

```
1 + "2."
```

Would result in an error like this:

Traceback (most recent call last):

```
File "main.py", line 2, in <module>
```

```
1 + "2 "
```

- **Less Code = More Achievement**

Python uses far less code to achieve what other programs do; for example, it uses around one-third or one-fifth less code than Java.

You need only two lines of code to read Python files:

```
with open("myfile.txt") as f:
```

```
    print(f.read())
```

For example:

```
# these lines create a "myfile.txt" file containing the data "Let's Learn Python"
```

```
with open("myfile.txt", "w") as f:
```

```
    f.write("Let's Learn Python")
```

```
# Both lines will read the data from myfile.txt
```

```
with open("myfile.txt") as f:
```

```
print(f.read())
```

And the output would be:

Let's Learn Python

Don't worry too much about the commands used for reading and writing a file; I will cover it all as the book progresses.

Who Uses Python?

Many of the largest names use Python, such as NASA, Google, HortonWorks, Quora, and many others. Do you want to know what you can build with it? Just about anything, including:

- Web applications
- GUI applications
- Data scraping from a website
- Data analysis
- Developing games
- Building utilities for system administration
- And much more

Now you know what Python is and what you can use it for, we'll turn to setting up your environment and installing it on Windows, Mac, and Linux systems.

Installing Python 3 - Windows

The first step is to download Python onto your Windows machine, so head to <https://www.python.org/downloads> and download the latest version for Windows. During the download, you will see a Customize Python window – make sure the option to “Add Python.exe to Path” is checked.

Once you have installed Python, open a command prompt on your Windows PC – either open the search box in the Start menu and type in ‘cmd’ or open the Start menu, click on Windows System>Command Prompt.

At the command prompt, type in

```
python
```

The Python shell opens; type the following to check it all works:

```
print("Good Morning World")
```

You will see the following on your screen:

```
Good Morning World
```

Installing a Text Editor

You require a text editor to write programs in Python, and you can use any one you want. Notepad is already included with Windows, but a better one is Notepad++ or even Sublime Text. Make your choice and download it via the internet.

After we've looked at the other platforms, we'll move on to running Python programs.

A quick aside; while I am showing you how to install on Mac and Linux, the book is based on Python for Windows – the code is the same for all platforms, though, so you shouldn't have any trouble following along.

Installing Python - Mac OSX

The first step is to install XCode, the IDE (integrated development environment) provided by Apple. It comes with a few tools that we will need to use later too. Head to the app store on your mac to download XCode – it is a large download but is the best way of getting what you need.

Once that is installed, we need the command line tools. Open the XCode menu and choose Preferences. A window will open, click the tab for Downloads and then click on Command Line Tools>Install.

You will need somewhere to store your projects, so open a Terminal – go to your Applications folder and click on Utilities>Terminal. It should open in the home directory – test it by typing the following – do not type the \$ sign:

```
$cd
```

The \$ sign signifies the start of a command-line instruction and cd stands for change directory, If you don't say where you want to go to, you will automatically be sent to your home directory. What we want is a folder named Code to store our projects, so type in:

```
$mkdir Code
```

You might have guessed that mkdir stands for make directory, and these are similar to folders in Finder. By making a new directory using the command line, you will place another folder into Finder. The word Code is what is known as a command argument – mkdir requires a string specified as the new directory name – we want one called Code, so that is the argument we pass to it. If you do not provide a name, you will get an error, similar to the following:

```
$ mkdir
```

```
usage: mkdir [-pv] [-m mode] directory ...
```

Now you need to install an OS X package manager called Homebrew, a series of code files that all work together. To install them, we need to run a script – a piece of code – that places specific files into the right directories. Many of the packages you require will also have dependencies, and Homebrew will find those and install them for you; it also keeps them in one place and let you know when there are available updates.

Type the following code to install

```
Homebrew:$ruby-e"$(curl-fsSL  
https://raw.github.com/mxcl/homebrew/go)"
```

The last part of that code is a URL, and if you opened it, you would see nothing but code. This script is written in Ruby, telling your computer how to install Homebrew. The curl bit is one of the command-line tools, this one transferring files by using their URL and -fsSL is four flags, options that tell curl how the file from the URL should be handled. Because the script needs executing, we use ruby as the starting command, and -e is a flag that executes strings as lines of code. Homebrew will provide you with all the instructions you need from here to finish the installation.

Now we are finally ready to install Python, but, usually, OS X comes with Python pre-installed. Go to your command prompt and type in

```
$python—version
```

You should see one of two things – an error message, in which case Python is not installed, or a message telling you what version of Python you have. If you haven't got Python or you have an older version and want to upgrade, use Homebrew. All Homebrew commands start with brew and the command you want. Type the following in to install Python:

```
$ brew install python
```

And to upgrade to the latest version, type

```
$ brew install python 3
```

Python has a few specific package managers, and the one you want is called pip, standing for pip installs packages. It has a single dependency, which is distribute, but you can't use Homebrew to get either. Instead, use readily available Python scripts. Type the following at the command prompt:

```
$ curl -O http://python-distribute.org/distribute_setup.py
```

```
$ python distribute_setup.py
```

```
$ curl -O https://raw.github.com/pypa/pip/master/contrib/get-pip.py
```

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

Each script is obtained and executed using two scripts – last time, we did everything with one.

The last step is to install a virtual environment. These are useful because you might be working with projects with conflicting or different dependencies, and you don't want these on your machine. Instead, install virtualenv using this code:

```
$ pip install virtualenv
```

That completes the Python setup for Mac OS X.

Installing Python - Linux

Some Linux versions already have Python, but the version will vary, and not all of them include IDLE, the interactive development environment application. If you have an old Python version, you should install a newer one so you can access IDLE.

If you need to install Python, follow these steps:

Standard Installation

This works on all Linux systems, but you do need to type commands into the terminal. Again, depending on your Linux version, those commands may vary.

First, go to <http://python.org/download> and download the latest version for Linux

Save the file and then double-click on it to open the Archive Manager window. Here, you will see Python when all the files have successfully extracted.

Double click on that folder and the files get extracted and saved to a home folder subfolder called Python 3.x.x

Open a Terminal and, at the command prompt type the following to install everything you need – ignore this step and Python won't work. Type all these commands, pressing enter in between each one:

sudo apt-get install build-essential

sudo apt-get install libsqlite3-dev

sudo apt-get install libbz2-dev

CD Python 3.x.x

./configure and press Enter.

Now, the script will check the type of system build and carry out a few tasks based on your system – be patient as this may take a few minutes.

Next, type in

make

and press enter.

The script is executed, creating the application software – again, this can take a minute or so.

Type

```
sudo make altinstall
```

and press Enter.

You might be asked to input the administrator password – type it in and press Enter. Python is now installed in your system.

Running Python Programs

There are two ways to run Python programs, first using commands in the Python shell and second, by running a program already stored in a file. The latter is the most common method, so let's make a file we'll name morning.py. You need to use your text editor for this so open it and type

```
print("Good Morning World")
```

Save it as morning.py in your documents directory.

The print function displays strings onto the console, and it can take one or more arguments. When you pass two or more arguments, each is displayed separately by the print() function, with a space between each one.

For example, typing this:

```
print("Good," "Morning," "World")
```

would result in this:

Good Morning World

Open your terminal and use the cd command to change the current directory to C:\Users\YourUserName\Documents:

```
cd Documents
```

and to run it, type:

```
python hello.py
```

You should, if all goes well, see this as the output:

```
Good Morning World
```

That takes care of setting up Python, time to start coding! We'll begin by looking at data types and variables.

Data Types and Variables

A variable is a location in memory, given a specific name, used for storing object references. The names given to both functions and variables are also called identifiers and there are specific rules we must work to:

1. Identifiers may only begin with an underscore (_) or a letter – no numbers. So, my_var is valid, 1var is not.
2. Identifiers may be made up of a combination of numbers letters and underscores, no other characters. For example, you could have python_101 or _python, but you could not have \$python\$ - the \$ sign is not valid.
3. There is no limit on the length of an identifier name, although keeping them short and to the point is recommended.
4. You cannot use any Python keywords as an identifier name. These are reserved for use by Python for specific purposes.

Python Keywords

These are the keywords:

and

as

assert

break

class

continue

def

del
elif
else
except
False
finally
for
from
global
if
import
in
is
lambda
None
nonlocal
not
or
pass
raise
return
True
try
while

with

yield

Assign a Variable With a Value

Every program needs values to work with, i.e. 1, 10, 3.14, “Good Morning” are values. In computer coding, values are also called literals and these may be of different types. For example, 1 and 10 are ints, 9.15 are floats, and “Good Morning” is type string. Everything in Python is an object, including the data types such as string, int, and float – we’ll go into this in more detail later.

Unlike other programming languages, you don’t have to declare variable types before you need them. The interpreter will know what the variable type is using the data in it.

We use the = sign to assign variables with values, also called the ‘assignment operator.’

Here are a few examples:

x = 100

pi = 3.14

empname – “Python is fantastic”

a = b = c = 150

Try these statements in your interpreter – copy the following code:

x = 100 # x is an integer

pi = 3.14 # pi is a float

empname = "python is fantastic" # empname is a string

a = b = c = 150 # this assigns a, b, and c with 150

```
print(x) # print the value of x  
8  
print(pi) # print the value of pi  
9  
print(empname) # print the value of empname  
10  
print(a, b, c) # print the value of a, b, and c, at the same time  
You should see this on your screen:  
150  
3.14  
python is fantastic
```

Note:

Values assigned to variables are not stored in the variable. Instead, only a reference, an address if you like, of the object's location in memory is stored. So, in the code above, the variable named x is storing a reference to 100, which is the int object, and not 100 as a number.

Using Comments

A comment is nothing more than a note added by the coder to explain parts of the code. They are not program statements, and the interpreter will not execute them. Python comments always start with the hash sign (#).

For example:

```
# This program will print “Good Morning World.”  
print(“Good Morning World”)
```

Try it for yourself by typing the above into the interpreter, and you should see “Good Morning World” displayed on the screen.

The first line of the program is a comment so that Python will ignore it. You can also add comments to the end of a line too. Try this example:

```
#This program will print “Good Morning World.”
```

```
print(“Good Morning World”) # display “Good Morning World.”
```

These are known as end-of-line comments.

Simultaneous Assignment

Simultaneous assignment, also called multiple assignment, lets us give values to more than one variable at a time. The syntax for this is:

```
var1, var2, ..., varn = exp1, exp2, ..., expn
```

That statement tells Python that all expressions to the right should be evaluated and assigned to the correct variable to the left.

Try this example:

```
a, b = 10, 20
```

```
print(a)
```

```
print(b)
```

The output should be:

```
10, 20
```

This is useful when we need the values from two variables swapped.

For example:

```
x = 1 # x has a starting value of 1
```

```
y = 2 # y has a starting value of 2
```

```
y, x = x, y # swap the x and y values
```

```
print(x) # x has a final value of 2
```

```
print(y) # y has a final value of 1
```

The output should be

1, 2

Data Types

There are five main data types in Python:

- Number
- List
- String
- Dictionary
- Tuple

But there is a sixth, the Boolean type, which has two values or literals – True and False – but other values are false too:

- 0 – zero
- 0.0
- [] – an empty list
- () – an empty tuple
- {} – an empty dictionary
- None

Receiving An Input from The Console

We use the `input()` function for this, and the syntax is:

```
input([prompt]) -> string
```

The function will take an optional argument of string type; the argument is called prompt, and the output is a string.

For example:

```
name = input("Input your name: ")
```

```
print(name)
```

And the output will be whatever you typed in as your name

It is worth noting that, even if you input a number, `input()` will only return a string. If you wanted a number output, you would need to use either the `eval()` or `int()` functions.

For example:

```
age = int(input("Enter your age: "))
```

```
print(age)
```

```
print(type(age))
```

Again, the output is whatever you typed in.

Importing a Module

Code organization in Python is done using modules, and there are quite a few already built-in. These include:

- The math module for all math-related functions
- The re module for regular expressions
- The os module for functions related to the operating system

And many more.

Using a module is simple enough, but it must be imported. We do this with the import statement, like this:

```
import module_name
```

And we use slightly different syntax to import multiple modules at once:

```
import module_name_1, module_name_2
```

Try this example:

```
>>> import math, os
```

```
>>>
```

```
>>> math.pi
```

```
3.141592653589793
```

```
>>>
```

```
>>> math.e  
2.718281828459045  
>>>  
>>>  
>>> os.getcwd() # print the current working directory  
>>> '/home/user'  
>>>
```

And the output should be:

```
3.141592653589793  
2.718281828459045  
/box
```

The first line of the code is importing all the classes, functions, constants, and variables, the os and math modules define. Accessing an object in a module requires the module name, a dot, and the object name – it could be a class, a constant, a function, or a variable. In our example, we want to access pi and e – these are common constants from the math module. Next, we call a function called getcwd() from the os module and this prints our current working directory.

Got all that?

Then we'll move onto numbers.

Python Numbers

The number data type supports numerical values only, such as 3.14, -100, 0.000123, 99999999, etc.

There is support for three type of numerical value:

- int – integer values, which are whole numbers – 1, 100, -1000000, 987654321, for example
- float – floating point numbers, those with decimal points – 1.1, 3.14, -55.5, for example
- complex – complex numbers such as 1+8j, -1+8j, 25j, 5.5+3.14j, for example

Integers

All Python integer literals are part of the int class, for example:

```
>>> i = 150
```

```
>>> i
```

The output is:

```
150
```

Floats

A floating point number is a literal containing a decimal point – here's an example:

```
>>> f = 10.5
```

```
>>> f
```

The output is:

10.5

Worth noting is that when an operand for a numeric operator is a floating value, the result will also be a floating value.

```
>>> 3 * 1.5
```

The output is:

4.5

Complex Number

Complex numbers are made up of two parts – imaginary and real – and j is used to denote them. Complex numbers may be defined in this way:

```
>>> x = 2 + 3j # 2 is real and 3 is imaginary
```

Determining Types

Python has a built-in function called `type()`, used for determining a variable's type. Here's an example:

```
>>> x = 14  
>>> type(x)  
<class 'int'>
```

Python Operators

Python has several built-in number operators used for carrying out calculations in code. The operators are:

Operator	Name	Example	Result
+	Addition	<code>20+20</code>	40
-	Subtraction	<code>20-10</code>	10.0
*	Multiplication	<code>5*10</code>	50
/	Float division	<code>4/5</code>	0.8
//	Integer division	<code>4//5</code>	0
**	Exponentiation	<code>4**2</code>	16
%	Remainder	<code>27%4</code>	3

The first three are self-explanatory – the rest you'll need a bit of help with.

Float Division

The division operator (`/`) divides the given numbers and returns a floating-point number as the result. This means that the result will always return a fractional part, for example:

```
>>>3/2
```

Returns

1.5

Integer Division

Integer division (//) does the division operation but truncates the result, so the decimal or fractional part is not shown, only the integer. For example:

```
>>>3/2
```

Returns

1

Exponential

The exponential operator (**) will compute a raised to the power of b – a^b. For example:

```
>>>2 ** 3
```

is the same as

```
>>>2 * 2 * 2
```

And the result in both cases is

8

Remainder Operator

The remainder operator (%) is also called the modulus operator, and it returns the remainder after a division operation. For example:

```
>>> 7 % 2
```

Will return

1

To explain this – 2 goes into 7 three times with 1 left over.

Operator Precedence

All Python expressions are evaluated using something called operator precedence. This means that the operators are evaluated in a certain order. Here's an example:

```
>>> 5 * 5 + 2
```

Which one is evaluated first - + or *? This should be relatively easy to work out if you remember your math lessons.

To help you out, the list below is the Python operators in order of precedence – this is all the operators, not just the mathematical operators so don't worry if you don't understand them all right now:

Operator Description

) parentheses or grouping

f(args...) function calls

x[index:index] slicing

x[index] subscription

attribute attribute reference

** exponentiation

~x bitwise not

+x, -x positive and negative

* , /, % multiplication, division, remainder
+, - addition, subtraction
<<, >> bitwise shifts
& bitwise AND
^ bitwise XOR
| bitwise OR
in, not in, is, not is
<, <=, >, =>, !=, == comparisons and membership identity
not x Boolean NOT
and Boolean AND
or Boolean OR
lambda lambda expressions

So, going back to our example, * comes before +, so the multiplication is evaluated first, and then the addition. So the output for the example is 27.

Here's another example:

5 + 5 – 3

What happens first - + or -? Both the addition and subtraction operators are the same in terms of precedence so that Python will evaluate this statement from left to right – addition first and then the subtraction – and the output is 7.

There is an exception to this, though – when the assignment operator is used, evaluation goes right to left. The only way to change the order of precedence here is to use parentheses. For example:

```
>>> 5 * (5 + 2)
```

And this returns

35

Anything in the parentheses comes first in the evaluation, so (5 + 2) is evaluated first and then the multiplication.

Augmented Assignment Operator

Augmented assignment operators let you use shortcuts for assignment statements. For example:

```
>>> count =  
>>> count = count +  
>>> count  
2
```

Could be written as

```
>>> count = 1  
>>> count += 1  
>>> count  
2
```

And the other operators may be combined with the assignment operator to give us augmented assignment operators, like this:

Operator	Name	Example	Equivalent
<code>+=</code>	addition assignment	<code>x += 3</code>	<code>x = x + 3</code>
<code>-=</code>	subtraction assignment	<code>x -= 2</code>	<code>x = x - 2</code>
<code>*=</code>	multiplication assignment	<code>x *= 6</code>	<code>x = x * 6</code>
<code>/*=</code>	division assignment	<code>x /= 6</code>	<code>x = x / 6</code>
<code>//*=</code>	integer division assignment	<code>x //= 6</code>	<code>x = x // 6</code>
<code>%*=</code>	remainder assignment	<code>x %= 6</code>	<code>x = x % 6</code>
<code>**=</code>	exponent assignment	<code>x **= 6</code>	<code>x = x ** 6</code>

Once you have digested that, take a short break and we'll move onto strings.

Python Strings

Python strings are a series of adjacent characters with single or double quotation marks as delimiters. There is no specific data type in Python for characters; instead, a character string represents them.

Creating Strings

Here's how to create a string in Python:

```
>>> name = "thomas" # a string
```

```
>>> mychar = 'a' # a character
```

This syntax may be used for string creation too:

```
>>> name1 = str() # this creates empty string objects
```

```
>>> name2 = str("newstring") # string object with 'newstring'
```

Try this example:

```
name = "thomas" # a string
```

```
mychar = 'a' # a character
```

```
print(name)
```

```
print(mychar)
```

```
name1 = str() # this creates empty string objects
```

```
name2 = str("newstring") # string object with 'newstring'
```

```
print(name1)
```

```
print(name2)
```

And you should see the following on your screen:

Python Strings are Immutable

What does this mean? It means that you cannot modify a string once you have created it. Here's an example:

```
>>> str1 = "good day"
```

```
>>> str2 = "good day"
```

In this example, str1 and str2 are both referring to one string object of "good day," stored somewhere in memory. An easy way to test if both are referring to one object is by using a function called `id()`. We can use this function to get the memory address or location of any object stored in memory.

For example:

```
>>> id(str1)
```

```
78965432
```

```
>>> id(str2)
```

```
78965432
```

Both objects point to the same location, so both point to the same object.

What about if we tried modifying the str1 object? Let's try to add another string to it:

```
>>> str1 += " michael"
```

```
>>> str1
```

```
good day michael
```

```
>>> id(str1)
```

```
>>> 78964597
```

Now, str1 is pointing to a different location in memory, proving that concatenation does not modify existing string objects; instead, a new string object gets created. Another immutable type is int.

Try this example:

```
str1 = "good day"  
str2 = "good day"  
print(id(str1), id(str2))
```

```
str1 += " michael"
```

```
print(str1)
```

```
print(id(str1))
```

The result is:

```
140425297247904 140425297247904
```

```
Good day michael
```

```
140425297288304
```

String Operations

In Python, indexing begins at 0. Accessing the first character in any string is this:

```
>>> name[0] #
```

t

Try it for yourself:

```
name = "thomas"
```

```
print(name[0])
```

```
print(name[1])
```

You should see this on your screen:

t

h

Two more operations you can do on a string is concatenation (+) and repetition (*), Here's an example:

The + operator is used to concatenate string and * operator is a repetition operator for string.

```
>>> s = "thomas and " + "friends"
```

```
>>> print(s)
```

The result is:

thomas and friends

And another example:

```
>>> s = "pork is delicious " * 3
```

```
>>> print(s)
```

The result is:

'pork is delicious pork is delicious pork is delicious.'

Try it for yourself:

```
s = "thomas and " + "friends"
```

```
print(s)
```

```
s = "pork is delicious " * 3
```

```
print(s)
```

The result is

thomas and friends

pork is delicious pork is delicious pork is delicious

Slicing Strings

Slicing is an operation that takes part of a string from another one, and the operator is []. The syntax is

```
s[start:end]
```

The result will be the section of the string starting from the index indicated by 'start' and ending with the indicated end index – 1. Here's an example:

```
>>> s = "Good day"
```

```
>>> s[1:3]
```

oo

And a few more:

```
>>> s = "Good day"
```

```
>>>
```

```
>>> s[:6]
```

```
'Good d'
```

```
>>>
```

```
>>> s[4:]
```

```
'ing'
```

```
>>>
```

```
>>> s[1:-1]
```

```
'ood da'
```

Note

Start and end indexes are optional; if you leave them out, the start index is a default of 0, and the end index is a default of the last string index.

ord() and chr() Functions

The `ord()` function returns the given character's ASCII code, and the `chr()` function returns the character that corresponds to a given ASCII number. For example:

```
>>> ch = 'b'
```

```
>>> ord(ch)
```

```
98
```

```
>>> chr(97)
```

```
'a'
```

```
>>> ord('A')
```

```
65
```

Try it for yourself:

```
ch = 'b'
```

```
print(ord(ch))
```

```
print(chr(97))
```

```
print(ord('A'))
```

The result is

98

a

65

Python String Functions

Python has three built-in string functions, as you can see below.

Function Name Description

`len()` returns string length

`max()` returns the character with the highest ASCII value

`min()` returns the character with the lowest ASCII value

Here are examples of all three:

```
>>> len("hello")
```

5

```
>>> max("abc")
```

'c'

```
>>> min("abc")
```

'a'

Try it for yourself:

```
print(len("hello"))
```

```
print(max("abc"))
```

```
print(min("abc"))
```

The result is:

5

c

a

In and Not In Operators

The in and not in operators, also called membership operators, are used for checking if one string exists in another string. For example:

```
>>> s1 = "Good day"
```

```
>>> "come" in s1
```

True

```
>>> "come" not in s1
```

False

```
>>>
```

Try it for yourself:

```
s1 = "Good day"
```

```
print("come" in s1)
```

```
print("come" not in s1)
```

And the result is:

True

False

String Comparison

There are several comparison operators, all used for comparing two strings.

The operators are:

- < - less than
- > - greater than
- <= - less than or equal to
- => - equal to or greater than
- == - equal to – checks two values, returns True if both have the same value, False if they are different
- != - not equal to – checks two values, returns True if they are not the same as one another, False if they are

Python string comparisons are lexicographical – that means using the character's ASCII values for the comparisons.

Let's say that str1 has a value of "Michael," and str2 has a value of "Milly." The first character of each string, M and M, are compared; they are the same. Then the second characters are compared, I and I. Again, they are the same.

The third set of two characters, C and L, are then compared. L has a higher ASCII value than C, so str 2 is greater than str1.

Here's a few more examples:

```
>>> "tie" == "tim"
```

False

```
>>> "free" != "freed"
```

True

```
>>> "arrow" > "aron"
```

```
True
```

```
>>> "right" >= "left"
```

```
True
```

```
>>> "abc" > ""
```

```
True
```

```
>>>
```

String Iteration Using a For Loop

A string is a type of sequence and we can iterate over it using a for loop. We'll be discussing loops in more detail later.

Here's an example:

```
>>> s = "hi"  
>>> for i in s:  
...     print(i, end="")  
hi
```

Note:

The `print()` function will, by default, print a string with a newline character. To change that, we pass a keyword argument called `end`, like this:

```
print("my string", end="\n") #this is the default  
print("my string", end="") # this prints the string with no newline  
print("my string", end="foo") # print() will now print every string with foo  
at the end
```

Try it for yourself:

```
s = "hi"
```

```
for i in s:
```

```
    print(i, end="")
```

The result is:

```
Hi
```

Testing Strings

There are several built-in methods in the Python string class that allow us to check for different string types:

Method Name Method Description

isalnum() if string is alphanumeric, returns True
isalpha() if string only contains alphabetic characters, returns True
isdigit() if string only contains digits, returns True
isidentifier() if string is a valid identifier, returns True
islower() if string is lowercase, returns True
isupper() if string is uppercase, returns True
isspace() if string has only whitespace, returns True

Here's some examples:

```
>>> s = "good day to python"
```

```
>>> s.isalnum()
```

False

```
>>> "Good day".isalpha()
```

True

```
>>> "2019".isdigit()
```

True

```
>>> "first Number".isidentifier()
```

False

```
>>> s.islower()
```

True

```
>>> "GOOD DAY".isupper()
```

True

```
>>> " \t".isspace()
```

True

Try it for yourself:

```
s = "good day to python"
```

```
print(s.isalnum())
```

```
print("Good day".isalpha())
```

```
print("2019".isdigit())
```

```
print("first Number".isidentifier())
```

```
print(s.islower())
```

```
print("GOOD DAY".isupper())
```

```
print(" \t".isspace())
```

And the result should be:

False

True

True

False

True

True

True

Searching for Substrings

The methods used to search for substrings in a string are:

Method Name Method Description

endswith(s1: str): bool if the string ends with the s1 substring returns True

startswith(s1: str): bool if the string starts with the s1 substring, returns True

count(substring): int counts and returns number of times the given substring appears in the string

find(s1): int finds and returns the lowest index at which s1 starts; if it isn't found, -1 is returned

rfind(s1): int finds and returns the highest index at which s1 starts or -1 if it isn't found

's an example:

```
>>> s = "good day to python"
```

```
>>> s.endswith("thon")
```

True

```
>>> s.startswith("good")
```

False

```
>>> s.find("day")
```

3

```
>>> s.find("become")
```

-1

```
>>> s.rfind("o")
```

15

```
>>> s.count("o")
```

1

>>>

Try it for yourself:

```
s = "good day to python."
```

```
print(s.endswith("thon"))
```

```
print(s.startswith("good"))
```

```
print(s.find("day"))
```

```
print(s.find("become"))
```

```
print(s.rfind("o"))
```

```
print(s.count("o"))
```

And the result is:

True

False

1

Converting Strings

There are also several string conversion methods in Python:

Method Name Method Description

`capitalize()`: str a copy of the string is returned with a capitalized first character

) : str string returned with all characters converted to lowercase
): str string returned with all characters converted to uppercase
str string returned with the first character in every word capitalized
case(): str string returned with the cases swapped over
e(old\, new): str old string replaced by a new string and new string returned
some examples:
"thon string"

= s.capitalize()

'a string'

' = s.title()

'

'a String'

= "This is a test"

' = s.lower()

'

'a test'

' = s.upper()

'

'IS A TEST'

```
s = s.swapcase()
```

```
;
```

```
S A tEST'
```

```
s = s.replace("Is", "Was")
```

```
;
```

```
'Was A Test'
```

```
's a test'
```

Try it for yourself:

```
s = "python string"
```

```
s1 = s.capitalize()
```

```
print(s1)
```

```
s2 = s.title()
```

```
print(s2)
```

```
s = "This is a test."
```

```
s3 = s.lower()
```

```
print(s3)
```

```
s4 = s.upper()
```

```
print(s4)
```

```
s5 = s.swapcase()
```

```
print(s5)
```

```
s6 = s.replace("Is", "Was")
```

```
print(s6)
```

```
print(s)
```

And the result is:

Python string

Python String

this is a test

THIS IS A TEST

tHIS iS A tEST

This Was A Test

This is a test

In the next section, we will look at Python lists.

Python Lists

Another type of sequence the list class defines the list, which lets you add elements, delete them, or process them very easily.

Creating Lists

The syntax for creating a list is:

```
>>> l = [1, 2, 3, 4]
```

The entire list is enclosed in square brackets, and a comma separates every element. The elements can be the same or different types. For example:

```
l2 = ["a string", 12]
```

You can also create lists in other ways:

```
list1 = list() # Creates empty list
```

```
list2 = list([21, 30, 52]) # Creates list containing elements 21, 30, 52
```

```
list3 = list(["thomas", "friends", "spike"]) # Creates list containing strings
```

```
list5 = list("python") # Creates list containing the characters p, y, t, h, o, n
```

Note:

Lists are mutable, meaning you can modify them after you have created them.

Accessing List Elements

The index operator ([]) may be used for accessing elements in a list. Don't forget, indexing begins at 0.

Here's an example:

```
>>> l = [1,2,3,4,5]
```

```
>>> l[1] # access element 2 in the list  
2
```

```
>>> l[0] # access element 1 in the list 1
```

Common List Operations

The list operations in Python are:

Operation Name Description

f element x appears in sequence s, returns True, otherwise False

n s if element x is not in sequence s, returns True, otherwise False

! concatenated or joins sequence s1 to s2

* s n number of copies of the sequence s, all concatenated

: ith element from sequence s

: the length or number of elements in sequence s

: the smallest element from s

: the largest element from s

: the sum of all the numbers in s

p in a for loop, the elements are traversed right to left

re some examples of lists using functions:

```
>>> list1 = [2, 3, 4, 1, 32]
```

```
>>> 2 in list1
```

```
True
```

```
>>> 33 not in list1
```

```
True
```

```
>>> len(list1) # find number of elements in list
```

```
5
```

```
>>> max(list1) # find largest element in list
```

```
32
```

```
>>> min(list1) # find smallest element in list
```

```
1
```

```
>>> sum(list1) # sum of elements in list
```

```
42
```

List Slicing

You can also use the slice operator)[start:end]) to fetch sub lists from a list in a similar way to how it works with strings:

```
>>> list = [11,33,44,66,788,1]
```

```
>>> list[0:5] # returns list beginning from index 0 to index 4
```

```
[11,33,44,66,788]
```

```
>>> list[:3]
```

```
[11,33,44]
```

And, the same as it is for strings, the start index is optional; if left out, by default it is 0.

+ and * Operators in List

The + operator is used for joining lists:

```
>>> list1 = [11, 33]  
>>> list2 = [1, 9]  
>>> list3 = list1 + list2  
>>> list3  
[11, 33, 1, 9]
```

While the * operator is used for replicating the list elements:

```
>>> list4 = [4, 3, 2, 1]  
>>> list5 = list4 * 3  
>>> list5  
[4, 3, 2, 1, 4, 3, 2, 1, 4, 3, 2, 1]
```

in or not in Operator

We use the in operator to find out if specified elements are in the list. If they are, True is returned; if not, it is False.

The in operator is used to determine whether the elements exists in the list. On success it returns True on failure it returns False .

```
>>> list1 = [10, 20, 40, 15, 79, 99]
```

```
>>> 20 in list1
```

True

And not in is the opposite:

```
>>> 42 not in list1
```

False

Using a For Loop to Traverse a List

You know that a list is a sequence and you know it is iterable which means a for loop may be used for looping through the list's elements. Here's an example:

```
>>> list = [1,2,3,4,5]
```

```
>>> for i in list:
```

```
... print(i, end=" ")
```

```
1 2 3 4 5
```

The most common list methods are:

Method Name Description

x: object): none adds a new element, x, at the end of the list, returning None

object): int counts how often x is in the list and returns the number

: list): None appends every element in l to the given list, returns None

x: object): int returns the index where x first appears in the list

index: int, x: object): None inserts x at a specified index, returns None

(: object): None removes x from the first place it appears, returns None

reverse(): None reverses the list order, returns None

lone sorts the list elements in ascending order, returns None

bject removes and returns the object at a specified index

```
>>> list1 = [1, 3, 5, 2, 25, 6]
```

```
>>> list1.append(19)
```

```
>>> list1
```

```
[1, 3, 5, 2, 25, 6, 19]
```

```
>>> list1.count(5) # Returns count for number 5
```

```
1
```

```
>>> list2 = [96, 59]
```

```
>>> list1.extend(list2)
```

```
>>> list1
```

```
[ 1, 3, 5, 2, 25, 6, 19, 96, 59 ]
```

```
>>> list1.index(2) # Returns index of number 2
```

```
2
```

```
>>> list1.insert(1, 22) # Inserts 22 at index 1
```

```
>>> list1
```

```
[ 1, 22, 3, 5, 2, 25, 6, 19, 96, 59 ]  
>>>  
>>> list1 = [1, 22, 3, 5, 2, 25, 6, 19, 96, 59]  
>>> list1.pop(2)  
3  
>>> list1  
[1, 22, 3, 5, 2, 25, 6, 19, 96, 59]  
>>> list1.pop()  
96  
>>> list1  
[ 1, 22, 3, 5, 2, 25, 6, 19, 59 ]  
>>> list1.remove() # Remove number 25  
>>> list1  
[1, 22, 3, 5, 2, 6, 19, 59]  
>>> list1.reverse() # Reverse the list  
>>> list1  
[99, 19, 6, 25, 2, 5, 3, 22, 1]  
>>> list1.sort() # Sort the list  
>>> list1  
[1, 2, 3, 5, 6, 19, 22, 25, 99]  
>>>
```

List Comprehension

Although you need knowledge of for loops to understand this properly, we are discussing that later; run through this for now and come back to it after you've done loops.

List comprehension is a neat way of creating a list. Inside a set of square brackets is an expression, a for clause, and then for or if clauses – 0 or more.

Have a look at these examples:

```
>>> list1 = [ x for x in range(10) ]  
>>> list1  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
  
>>>  
  
>>>  
>>> list2 = [ x + 1 for x in range(10) ]  
>>> list2  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
  
>>>  
  
>>>  
>>> list3 = [ x for x in range(10) if x % 2 == 0 ]  
>>> list3  
[0, 2, 4, 6, 8]  
  
>>>  
  
>>>  
>>> list4 = [ x *2 for x in range(10) if x % 2 == 0 ]  
[0, 4, 8, 12, 16]
```

Next, we will look at Python dictionaries.

Python Dictionaries

A dictionary is a data type in Python used for storing key-value pairs. The idea of the dictionary is to let you easily retrieve information, add new, remove, or even modify the values by using a key. If you are familiar with other languages, you probably know this as a ‘hash.’ Because you can do so much with a dictionary, it follows that they are mutable.

Creating a Dictionary

Dictionaries are easy to create; all you need is the information and a set of curly braces {}. Every dictionary item is a key and a value, separated by a colon. And each key-value pair is separated by a comma. Here’s an example:

```
friends = {
```

Like this:

```
'thomas' : '234-567-890',
```

```
'friends' : '444-66-2221'
```

```
}
```

In this code, we have a dictionary called friends containing two items. Something to note here is that a key is a hashable type, but you can have any type for the value, and all keys should be unique.

```
>>> dict_emp = {} # creates an empty dictionary
```

Retrieving, Modifying And Adding Elements

Retrieving items from a dictionary is done with this syntax:

```
>>> dictionary_name['key']
```

Like this:

```
>>> friends['thomas']
```

```
'234-567-890'
```

The value is only returned if the key is in the dictionary; if not, you will get a KeyError exception or error message.

If you want to add an item or modify an existing one use this syntax:

```
>>> dictionary_name['newkey'] = 'newvalue'
```

Like this:

```
>>> friends['billy'] = '888-999-666'
```

```
>>> friends
```

```
{'thomas': '234-567-890', 'billy': '888-999-666', 'friends': '444-66-2221'}
```

Deleting Items

To delete an item from a dictionary, use this syntax

```
>>> del dictionary_name['key']
```

Like this:

```
>>> del friends['billy']
```

```
>>> friends
```

```
{'thomas': '234-567-890', 'friends': '444-66-2221'}
```

The item is deleted if the key is located, if not, you get a KeyError exception

Looping Items

To traverse the items in your dictionary, you can use a for loop:

```
>>> friends = {  
... 'thomas' : '234-567-890',  
... 'friends' : '444-66-2221'  
...}  
>>>
```

```
>>> for key in friends:  
...     print(key, ":", friends[key])  
...  
thomas : 234-567-890  
friends : 444-66-2221  
>>>  
>>>
```

Find the Dictionary Length

The len() function will return the dictionary length:

```
>>> len(friends)
```

```
2
```

in or not in Operators

The in or not in operators are used for checking the existence of a specific key:

```
>>> 'thomas' in friends
```

```
True
```

```
>>> 'thomas' not in friends
```

```
False
```

Equality Tests

Equality tests using the == and the != operators tell us if the dictionary has or hasn't got the same items:

```
>>> d1 = {"michael":41, "billy":3}  
>>> d2 = {"billy":3, "michael":41}
```

```
>>> d1 == d2
```

True

```
>>> d1 != d2
```

False

```
>>>
```

Note:

It is not possible to use any other relational operator, such as `>`, `<`, `=>`, or `<=` for dictionary comparisons.

Dictionary Methods

Python comes with a number of methods that help you work with dictionaries:

Method Description

`popitem()` removes specified item and returns a random dictionary item

`clear()` deletes all items from the dictionary

`keys()` returns all the keys as tuples

`values()` returns all the values as tuples

`get(key)` returns the specifies key value; if not found, `None` is returned. A `KeyError` exception is NOT thrown.

`del friends[key]` remove the specified item; if the key isn't found, a `KeyError` exception is thrown

are some examples:

```
friends = {'thomas': '234-567-890', 'billy': '888-999-666', 'friends': '444-66-2221'}
```

```
friends.popitem()
```

```
nas', '234-567-890')
```

```
friends.clear()
```

```
friends
```

```
friends = {'thomas': '234-567-890', 'billy': '888-999-666', 'friends': '444-66-2221'}
```

```
friends.keys()
```

```
keys(['thomas', 'billy', 'friends'])
```

```
friends.values()
```

```
values(['234-567-890', '888-999-666', '444-66-2221'])
```

```
friends.get('thomas')
```

```
567-890'
```

```
friends.get('michael', Doesn't exist')
```

```
Exists'
```

```
friends.pop('billy')
```

```
999-666'
```

```
friends  
mas': '234-567-890', 'friends': '444-66-2221'}  
, we turn our attention to tuples.
```

Python Tuples

Tuples are much like lists, but with one difference – they are immutable. Once you have created a tuple, you cannot add to it, delete anything from it, modify,

replace or even change the order of the items.

Creating A Tuple

Creating a tuple is done like this:

```
>>> t1 = () # create empty tuple without data  
>>>  
>>> t2 = (44, 55, 66)  
>>>  
>>> t3 = tuple([5, 6, 7, 8, 8]) # tuple created from array  
>>>  
>>> t4 = tuple("abc") # tuple created from string
```

Tuples Functions

You can also use other functions with tuples, such as `len()`, `min()`, `max()`, and `sum()`:

```
>>> t1 = (2, 15, 66, 15, 90)  
>>> min(t1)
```

```
1  
>>> max(t1)
```

```
81  
>>> sum(t1)  
161  
>>> len(t1)
```

```
5
```

Iterating Through Tuples

You can iterate over tuples with a for loop – more about these later:

```
>>> t = (44, 55, 66, 77, 88)  
>>> for i in t:  
...     print(i, end=" ")  
>>> 44 55 66 77
```

Slicing Tuples

The slicing operators work the same on a tuple as they do on a string or a list:

```
>>> t = (44, 55, 66, 77, 88)  
>>> t[0:2]  
(44, 55)
```

In And Not In Operator

The in and the not in operators may be used to check if items exist in tuples:

```
>>> t = (44, 55, 66, 77, 88)  
>>> 55 in t  
True  
>>> 55 not in t
```

False

In the next section, we will look at Python data conversions

Datatype Conversion

Now and again, you will find that you need to change a data type to a different one. This is done through conversion or typecasting.

Converting an int to a Float

The float() function is used to convert an int to a float:

```
>>> i = 15
```

```
>>> float(i)
```

```
15.0
```

Converting a Float to an int

The int() function is used to convert a float to an int:

```
>>> f = 25.85
```

```
>>> int(f)
```

```
25
```

Converting a String to an int

The int() function is used to convert a string to an int:

```
>>> s = "456"
```

```
>>> int(s)
```

```
456
```

Note

If there are any non-numeric characters in a string, the int() function will cause a ValueError exception to be thrown.

Converting a Number to a String

The str() function is used to convert a number to a string:

```
>>> i = 150
```

```
>>> str(i)
```

```
"150"
```

```
>>> f = 1.5
```

```
str(f)
```

```
'1.5'
```

Rounding Numbers

The round() function is used to round numbers and the syntax is

```
round(number[, ndigits])
```

Here's an example:

```
>>> i = 24.86213
```

```
>>> round(i, 2)
```

```
24.86
```

In the next section, we turn to control statements.

Python Control Statements

Python statements are commonly executed based on one or more conditions. We're going to look at the if-else statement but, before we do that, we need to know what the relational operators are, used for comparisons between two objects:

Operator Description

`<=` less than or equal to

`<` less than

`>` more than

`>=` more than or equal to

`==` equal to

`!=` not equal to

The comparison result is and always will be a Boolean of True or False.
Let's look at a few examples:

```
>>> 2 == 4
```

False

```
>>> 10 > 2
```

True

```
>>> 10 == 10
```

True

```
>>> 55 != 15
```

True

Now we can look at if statements; the syntax for an if statement is this:

if boolean-expression:

#statements

else:

#statements

Note:

Every statement inside an if block should be indented to the same number of spaces; if the indentation differs, a syntax error will occur. In other languages, such as C, C#, and Java, curly braces are used instead.

Here's an example of an if statement:

```
i = 16
```

```
if i % 2 == 0:
```

```
print("Number is even")
else:
    print("Number is odd")
```

You can see from this that “Number is even” is printed if you have an even number; if it is odd, then “Number is odd” gets printed.

Note:

You do not need to use an else clause – it is optional – but if you choose to use one you use it like this:

```
if today == "wedding":
    print("congratulations!")
```

If today has a value of ‘wedding,’ then the result will be “congratulations.” If not, nothing is printed.

If you have multiple conditions to check, you would use an if-elif-else statement:

```
if boolean-expression:
    #statements
elif boolean-expression:
    #statements
elif boolean-expression:
    #statements
elif boolean-expression:
    #statements
else:
    #statements
```

There is no limit to how many elif conditions you use; it will depend on what the program requires. Here’s a proper example of an if-elif-else

statement:

```
today = "sunday"  
if today == "sunday":  
    print("this is sunday")  
elif today == "monday":  
    print("this is monday")  
elif today == "tuesday":  
    print("this is tuesday")  
elif today == "wednesday":  
    print("this is wednesday")  
elif today == "thursday":  
    print("this is thursday")  
elif today == "friday":  
    print("this is friday")  
elif today == "saturday":  
    print("this is saturday")  
else:  
    print("something else")
```

Nested if Statements

It is possible to nest one if statement inside another, like this:

```
today = "thanksgiving"  
bank_balance = 35000  
if today == "thanksgiving":
```

```
if bank_balance > 25000:  
    print("go shopping")  
else:  
    print("Do the gardening")  
else:  
    print("standard working day")
```

Next, we will delve into Python functions

Python Functions

A function is a piece of code that you can reuse; they are useful for proper organization of code. Functions are created and used to run one statement set several times in one program, so we don't have to keep repeating ourselves and writing repetitive code over and again.

Creating Functions

The def keyword is used to begin a function, with this syntax:

```
def function_name(arg1, arg2, arg3, .... argN):  
    #a statement inside a function
```

Note:

Every statement inside a function must be indented with an equal number of spaces. Functions can accept none or more parameters (arguments), inside a set of parentheses. And you can use the pass keyword to leave out the function body, like this:

```
def myfunc():  
    pass
```

Here's an example of how it works:

```
def sum(start, end):
```

```
result = 0  
for i in range(start, end + 1):  
    result += i  
print(result)
```

sum(20, 60)

Here, we have defined a function named sum(); it has two parameters, called start and end. It calculates the sum of every number between start and end.

Function With A Return Value

The function we wrote above just prints or displays the result on your console. But what if we wanted to go further and assign the result of the function to a variable so that we could do some more processing on it? To do that, we would use a return statement, which, in turn, will send the result to the caller and leave the function. Here's how:

```
def sum(start, end):  
    result = 0  
    for i in range(start, end + 1):  
        result += i  
    return result
```

```
s = sum(20, 60)
```

```
print(s)
```

The return statement is used to return a result of the sum of all the numbers and then assign that result to a variable called s.

The return statement may also be used with no return value:

```
def sum(start, end):  
    if(start > end):  
        print("start must not be more than end")  
    return      # no value is returned so we return a special value of None  
instead  
    result = 0  
    for i in range(start, end + 1):  
        result += i  
    return result
```

```
s = sum(120, 60)
```

```
print(s)
```

The output of this will be:

Start must not be more than end

None

If a function does not explicitly return a value, None will always be returned instead:

```
def test(): #a test function that has one statement
```

```
i = 100
```

```
print(test())
```

You can see from this that there isn't any value explicitly returned by the function so that None will be returned instead.

Global Variables Vs. Local Variables

- **Global variables** – these are not bound to any specific function; they are accessible inside the function and outside of it.
- **Local variables** – these are declared in a specific function and can only be accessed within that function.

Here are some examples:

Example 1 :

```
global_var = 12      # global variable
```

```
def func():
```

```
    local_var = 100    # local variable
```

```
    print(global_var) # global variables may be accessed inside a function
```

```
func()      # calling the function func()
```

```
#print(local_var)      # local_var cannot be accessed from outside the  
function – once the function has ended, local_var will be destroyed
```

Example 2:

```
xy = 150
```

```
def cool():
```

```
    xy = 250  # xy in the function is not the same as xy outside the function
```

```
    print(xy)  # this prints local xy variable i.e. 250
```

```
cool()
```

```
print(xy)  # this prints global xy variable i.e. 150
```

Local variables can be bound in the global scope, and we do this with the global keyword and the variable names separated with commas.

```
t = 1
```

```
def increment():
```

```
    global t  # t in the function is now the same as t outside the function
```

```
    t = t + 1
```

```
    print(t) # Displays 2
```

```
increment()
```

```
print(t) # Displays 2
```

Be aware that you cannot assign any values to variables that you declare as global:

```
t = 1
```

```
def increment():
    #global t = 1 # this is an error
    global t
    t = 101      # this is good
    t = t + 1
    print(t) # Displays 102
```

```
increment()
```

```
print(t) # Displays 101
```

In all truthfulness, you do not need to declare any global variable outside of the function; they can be globally declared inside it:

```
def foo():
    global x  # x is declared global so that it can be accessed outside of the
              # function
    x = 150
foo()
print(x)
```

Arguments With Default Values

Default argument values can be specified by using the assignment operator to assign values:

```
def func(i, j = 150):
    print(i, j)
```

This function has got two parameters named i and j. Because the default value of j is 150, when we call the function, j's value can be left out:

`func(2)` # no value has been passed to `j`, so the default is used

We'll call the function called `func()` again, but we'll give the `j` parameter a value this time:

`func(4, 600)` #we pass 600 as a value of `j`, so we don't need to use the default value

Keyword Arguments

Arguments can be passed to a method in two ways – using positional or keyword arguments. We saw how positional arguments worked earlier, so now we'll look at keyword arguments.

A keyword argument lets you use name-value pairs, such as `name=value`, to pass the arguments. Here's an example:

```
def named_args(name, greeting):
```

```
    print(greeting + " " + name )
```

```
named_args(name='james', greeting='Hello')
```

```
named_args(greeting='Hello', name='james') # you can pass arguments this way too
```

The output will be:

```
Hello james
```

```
Hello james
```

Combining Keyword and Positional Arguments

Combining keyword and positional arguments is possible, but positional arguments must always come before keyword arguments, like this:

```
def my_func(a, b, c):
```

```
    print(a, b, c)
```

Now, this function can be called in these ways:

```
# only positional arguments
```

```
my_func(15, 16, 17)
```

```
# the first argument is a positional argument, and the other two are keyword  
arguments
```

```
my_func(15, b=16, c=17)
```

```
# the same as the above
```

```
my_func(15, c=16, b=17)
```

```
# this is not correct; the positional argument must come before the keyword  
argument
```

```
# my_func(115, b=16, 17)
```

Multiple Values Returned From Function

The return statement is used to return more than one value from a function, with a comma used to separate each value. The result is a tuple:

```
def bigger(a, b):
```

```
    if a > b:
```

```
        return a, b
```

```
    else:
```

```
        return b, a
```

```
s = bigger(15, 150)
```

```
print(s)
```

```
print(type(s))
```

Next we will look at Python loops

Python Loops

There are only two loops in Python – the for loop and the while loop – both of which are used to control the flow of the program and help in making decisions.

The for Loop

The syntax for a for loop is

```
for i in iterable_object:
```

```
    # do something
```

Note:

Every statement within a for or a while loop must have the same indentation spaces; otherwise, you will get a `SyntaxError`. Here's an example:

```
my_list = [5, 6, 7, 8]
```

```
for i in my_list:
```

```
    print(i)
```

The output you would expect from this is

5

6

7

8

Let's discuss how a for loop works. In the first loop iteration, we assign `i` with a value of 1, and then we execute our `print` statement. In the second loop iteration, we assign `i` with a value of 2 and then execute the `print` statement once more. This will continue until all the list elements have been iterated over and the for loop exits.

range(a, b) Function

The function called `range(a, b)` will return a sequence. That sequence is made up of integers ranging from `a`, $a + 1$, $a + 2$, ..., $b - 2$, $b - 1$.

Here's an example:

```
for i in range(11, 20):
```

```
    print(i)
```

The output we expect from this is:

```
11
```

```
12
```

```
13
```

```
14
```

```
15
```

```
16
```

```
17
```

```
18
```

```
19
```

The range() function can also be used with a single argument, such as:

```
>>> for i in range(10):
```

```
...     print(i)
```

And the result should be:

```
0
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

7

8

9

The range for loop has printed or displayed the numbers from 0 to 9.

There is a third parameter, an optional one, for the range(a, b) function, once that specifies the size of the step. For example:

1

2 for i in range(1, 20, 2):

 print(i)

The output we would expect to see from this is:

1

3

5

7

9

11

13

15

17

19

The while Loop

The syntax for the while loop is:

while condition:

```
# do something
```

So, how does a while loop work?

A while loop will execute the statements it contains until a specified condition is no longer true. The condition is checked at the end of every loop iteration – if it is true, the statements will be executed. When the condition becomes false, the loop stops.

Here's an example:

```
count = 0
```

```
while count < 8:
```

```
    print(count)
```

```
    count += 1
```

We would expect to see the following output:

0

1

2

3

4

5

6

7

The loop continues to print until count is lower than 8.

The break Statement

The break statement is used to let us get out, or break out, from the loop; for example:

```
count = 0
```

```
while count < 8:
```

```
    count += 1
```

```
    if count == 4:
```

```
        break
```

```
        print("inside loop", count)
```

```
    print("out of while loop")
```

So, when count is equal to 4, if the condition is true, the break keyword will allow us to break out from our loop.

We would expect to see this output:

inside loop 1

inside loop 2

inside loop 3

out of while loop

The continue Statement

Whenever the program comes up against a continue statement, the current iteration ends, and control of the program will go straight to the end of the body of the loop. For example:

```
count = 0
```

```
while count < 8:
```

```
    count += 1
```

```
    if count % 2 == 0:
```

```
        continue
```

```
        print(count)
```

The output we expect to see from this is:

1

3

5

7

In this code, as soon as `count % 2 == 0`, execution of the continue statement occurs; the current iteration will end and control goes onto the next iteration.

Now we move onto mathematical functions in Python

Mathematical Functions

There are several mathematical functions built into Python:

Method Description

`i(number[, ndigits])` rounds the specified number. Precision may be specified in argument two

a, b) a raised to the power of b is returned

:) the absolute value of x is returned

x1, x2, ... xn) the biggest value from the arguments supplied is returned

«1, x2,... xn) the smallest value is returned from the given arguments

And the functions below are found in the math module; if you want to use these, you need to import the module, like this:

```
import math
```

Method Description

ceil(x) rounds up the given number; the nearest integer is returned

floor() rounds up the down and the nearest integer is returned

sqrt(x) the square root of the given number is returned

sin(x) the sin of x is returned where x is in the radian

cos(x) the cosine of x is returned where x is in the radian

tan(x) the tangent of x is returned where x is in the radian

Let's try to understand this with some examples:

```
>>> abs(-44)      # the absolute value is returned
```

```
44
```

```
>>>
```

```
>>> max(10, 4, 15, 92)    # the maximum number is returned
```

```
92
```

```
>>>
```

```
>>> min(65, 95, 14, 25)  # the minimum number is returned
```

```
14
```

```
>>>
```

```
>>> pow(9, 3)      # may also be seen as 9 ** 3
```

```
729
```

```
>>>  
>>> pow(3.5, 5.2)      # may also be seen as 3.5 ** 5.2  
674.766993305  
>>>  
>>> round(6.98)       # the number is rounded to the nearest integer  
7  
>>>  
>>> round(3.12498775614, 3) # the number is returned decimalized to  
three places  
3.124
```

And another example:

```
>>> import math  
>>> math.ceil(3.1234)  
4  
>>> math.floor(25.78912)  
25
```

Hopefully that explains the mathematical functions in Python, now its time to look at how random numbers are generated.

Generating Random Numbers

To generate random numbers in Python, you need to use the `random()` function, and that is found inside the `random` module. Using it requires you to import the module first:

```
import random
```

`random()` Function

When you use the `random()` function, random numbers are generated. The random number is `r` in that $0 \leq r < 1.0$. Here's an example:

```
>>> import random  
>>> for i in range(0, 10):  
...     print(random.random())  
...
```

We would expect to see an output of something similar to this:

```
0.7346397596936707  
0.8314503234887478  
0.9987325602052806  
0.43625298553512726  
0.4471960266125812  
0.5467030001314136  
0.22524111130982516  
0.5541548738565403  
0.8856297454931572  
0.08108953103391203
```

Another way of generating random numbers is to use randint(a, b) – this will produce random numbers that all between a and b, inclusive. Here's an example:

```
>>> import random  
>>> for i in range(0, 12):  
...     print(random.randint(1, 12))  
...
```

You will see an output similar to this:

12

2

12

5

7

3

1

1

10

5

In the next part, we'll take a look at techniques for file handling.

File Handling

Python file handling is used to read data to a file and to write from it.

Open a File

Before you can even think about reading to or writing from a file, you must open it, and the syntax for that is:

```
f = open(filename, mode)
```

We can give the `open()` function two arguments, called `mode` and `filename`. The latter is classed as a string argument; it specifies the filename and its file path while the former, also a string argument and that one specifies how the file is going to be used, i.e., writing or reading. The file handler object, `f`, is also called the file pointer.

Close a File

Once you are done reading and/or writing to a file, the next step is to close it. For this, you need the `close()` method:

```
f.close() # f is a file pointer
```

There are several ways of opening a file:

Method Description

r for reading a file

or writing to a file. If the file is already in existence, all its data is removed before it is opened. If not, a new file gets created.

w opens in ‘append’ mode – data is written to the end of it

for writing in binary mode

or reading in binary mode

Here's a few examples:

Writing Data

```
>>> f = open('myfile.txt', 'w') # file opened for writing  
>>> f.write('the first line\n') # a line gets written to the file  
>>> f.write('the second line\n') # another line gets written  
>>> f.close() # file is closed
```

Note:

Using the write() method doesn't add (\n) a new line the way the print() function does; instead, you must tell it by adding n to the method.

Reading Data

There are three methods you can use for writing data back from files:

Method Description

l[number] returns the number of characters specified; if not given the whole file content will be read

`ilne()` returns the following line

`ilnes()` reads every line as a string list

Reading all data:

```
>>> f = open('myfile.txt', 'r')
>>> f.read() # read all the file at the same time
"the first line\nthe second line\n"
>>> f.close()
```

Reading every line as an array:

```
>>> f = open('myfile.txt', 'r')
>>> f.readlines() # read all the file at the same time
["the first line\n", "the second line\n"]
>>> f.close()
```

Reading a single line:

```
>>> f = open('myfile.txt', 'r')
>>> f.readline() # read first line
"the first line\n"
>>> f.close()
```

Append Data

The file must be opened using ‘a’ to do this:

```
>>> f = open('myfile.txt', 'a')
```

```
>>> f.write("the third line\n")
```

19

```
>>> f.close()
```

Using a For Loop

The file pointer is required to iterate through files:

```
>>> f = open('myfile.txt', 'r')
```

```
>>> for line in f:
```

```
...   print(line)
```

...

the first line

the second line

the third line

```
>>> f.close()
```

Reading and Writing - Binary

Binary input/output is done using the pickle module. This is what lets you read data using the load method and write it using the dump method

Writing:

```
>> import pickle
```

```
>>> f = open('pick.dat', 'wb')
```

```
>>> pickle.dump(12, f)
```

```
>>> pickle.dump("a line", f)
```

```
>>> pickle.dump([5, 6, 7, 8], f)
```

```
>>> f.close()
```

Reading:

```
>> import pickle
```

```
>>> f = open('pick.dat', 'rb')
>>> pickle.load(f)
12
>>> pickle.load(f)
"a line"
>>> pickle.load(f)
[5, 6, 7, 8]
>>> f.close()
```

If the file doesn't have any more data, an `EOFError` (end of file) is thrown by `pickle.load()`.

Next, we'll look at objects and classes.

Objects and Classes

Because Python is object-oriented, everything in it is an object, even functions, and modules. Objects are used for creating programs and for storing both behavior and data.

Define a Class

Every class name consists of the `class` keyword, the class name, and ends with a colon (`:`). Usually, they have a `data` field where the data is stored and methods that define behavior. Not only that, all Python classes have an initializer, a special kind of method that is sometimes called a constructor. These are automatically invoked whenever an object is created.

Here's an example:

```
class Person:
```

```
    # initializer or constructor
    def __init__(self, name):
```

```
    self.name = name # a data field that is often called an instance  
variable
```

```
# method returning a string  
  
def whoami(self):  
  
    return "You are " + self.name
```

A new class has been created, named Person. It has one data field, name, and a method called whoami().

Self

Every Python method has an initial parameter of self, and that includes initializers and other special methods. This parameter is referring to an object used to invoke a method. When a new object is created, the self parameter, found inside the method called `__init__`, automatically references the newly created object.

Objects Created from Classes

```
p1 = Person('thomas') # we now have a new person object, named p1  
  
print(p1.whoami())  
  
print(p1.name)
```

From this, we would expect to see the following output:

You are Thomas

thomas

Note:

When a method is called, there is no need for anything to be passed to self; it is automatically done for you by Python.

You can also make changes to name data:

```
p1.name = 'jeremy'
```

```
print(p1.name)
```

The output from this should be:

Jeremy

How to Hide Data fields

Hiding data fields requires that private data fields are defined. Python allows you to do this quite easily with a pair of leading underscores. Private methods can be created in the same way.

Here's an example:

class BankAccount:

```
# initializer or constructor
```

```
def __init__(self, name, money):
```

```
    self.__name = name
```

self.__balance = money # __balance has been made private – it can only be accessed from within the class

```
def deposit(self, money):
```

```
    self.__balance += money
```

```
def withdraw(self, money):
```

```
    if self.__balance > money :
```

```
        self.__balance -= money
```

```
        return money
```

```
    else:
```

```
        return "Not Enough funds"
```

```
def checkbalance(self):  
    return self.__balance  
  
b1 = BankAccount('timothy', 500)  
print(b1.withdraw(600))  
b1.deposit(700)  
print(b1.checkbalance())  
print(b1.withdraw(800))  
print(b1.checkbalance())
```

Here's how you could try to get external access to `__balance`:

```
print(b1.__balance)
```

If you did this, you would get this back:

```
AttributeError: 'BankAccount' object has no attribute '__balance'
```

Because you have made `__balance` private, you can only access it inside the class.

In the next section, we will discuss operator overloading.

Operator Overloading

Python operators all do something different, but some can do more than one thing. Take the `+` operator, for example. We can use it for both addition and string concatenation. We can do this because both the `str` and the `int` class overload the `+` operator.

Operators are methods, each defined in the classes they are used in. When you define a method for an operator, it is called operator overloading. For example, when you have custom objects that you want to use the `+` operator with, a method named `__add__` must be defined.

Let's see an example to get a better understanding of this:

```
import math
```

```
class Circle:
```

```
    def __init__(self, radius):
```

```
        self.__radius = radius
```

```
    def setRadius(self, radius):
```

```
        self.__radius = radius
```

```
    def getRadius(self):
```

```
        return self.__radius
```

```
    def area(self):
```

```
        return math.pi * self.__radius ** 2
```

```
    def __add__(self, another_circle):
```

```
        return Circle( self.__radius + another_circle.__radius )
```

```
c1 = Circle(7)
```

```
print(c1.getRadius())
```

```
c2 = Circle(7)
```

```
print(c2.getRadius())
```

```
c3 = c1 + c2 # We can do this because we added the __add__ method, thus  
overloading the + operator
```

```
print(c3.getRadius())
```

From this, we would expect to see:

7

7

14

What we did here was included the method called `__add__()`. This lets us make use of the `+` operator for adding a pair of objects – circles, in our case. A new object has been created in the `__add__()` method, and that object is returned to the caller.

Python has quite a few built-in special methods:

Operator	Function	Method Description
<code>+</code>	<code>__add__(self, other)</code>	Addition
<code>*</code>	<code>__mul__(self, other)</code>	Multiplication
<code>-</code>	<code>__sub__(self, other)</code>	Subtraction
<code>%</code>	<code>__mod__(self, other)</code>	Remainder
<code>/</code>	<code>__truediv__(self, other)</code>	Division
<code><</code>	<code>__lt__(self, other)</code>	Less than
<code><=</code>	<code>__le__(self, other)</code> ,	Less than or equal to
<code>==</code>	<code>__eq__(self, other)</code> ,	Equal to
<code>!=</code>	<code>__ne__(self, other)</code> ,	Not equal to
<code>></code>	<code>__gt__(self, other)</code> ,	Greater than
<code>>=,</code>	<code>__ge__(self, other)</code> ,	Greater than or equal
to [index].	<code>__getitem__(self, index)</code> ,	
Index operator in ,	<code>__contains__(self, value)</code> ,	
Check membership len ,	<code>__len__(self)</code> ,	
Number of elements str ,	<code>__str__(self)</code> ,	string representation

You can see some of these functions in use below:

```
import math
```

```
class Circle:
```

```
    def __init__(self, radius):
```

```
        self.__radius = radius
```

```
    def setRadius(self, radius):
```

```
        self.__radius = radius
```

```
    def getRadius(self):
```

```
        return self.__radius
```

```
    def area(self):
```

```
        return math.pi * self.__radius ** 2
```

```
    def __add__(self, another_circle):
```

```
        return Circle( self.__radius + another_circle.__radius )
```

```
    def __gt__(self, another_circle):
```

```
        return self.__radius > another_circle.__radius
```

```
    def __lt__(self, another_circle):
```

```
return self.__radius < another_circle.__radius
```

```
def __str__(self):  
    return "Circle with radius " + str(self.__radius)
```

```
c1 = Circle(7)  
print(c1.getRadius())
```

```
c2 = Circle(7)  
print(c2.getRadius())
```

```
c3 = c1 + c2  
print(c3.getRadius())
```

```
print( c3 > c2) # possible because we added __gt__ method
```

```
print( c1 < c2) # possible because we added __lt__ method
```

```
print(c3) # possible because we added __str__ method
```

And we would expect to see the following output:

7

7

14

True

True

Circle with a radius 14

And, next, we look at inheritance and polymorphism in Python

Inheritance And Polymorphism

Inheritance and polymorphism are two of the most important concepts in Python.

Inheritance

Inheritance is one of the ways that a programmer can write much better code. It lets you make a general class before extending it, later in the code, to a specialized class.

With inheritance, all methods and data fields can be accessed, along with the ability to add fields and method of your own. In this way, inheritance allows us to organize our code better, rather than having to write it all again.

As an example, if class X were to extend class Y, Y would be the base or superclass, otherwise known as the parent class, while X would be the derived or subclass, otherwise known as the child class. A child class may only access public methods and data fields; the private ones can only be accessed from within the class.

Creating a subclass uses this syntax:

```
class SubClass(SuperClass):
```

```
# the data fields
```

```
# the instance methods
```

Here's an example to see how it works:

```
class Vehicle:
```

```
def __init__(self, name, color):
```

```
self.__name = name    # __name is private to the Vehicle class
self.__color = color

def getColor(self):      # getColor() function can be accessed by the Car
    class

        return self.__color

def setColor(self, color): # setColor can be accessed outside the class
    self.__color = color

def getName(self):       # getName() can be accessed outside the class
    return self.__name

class Car(Vehicle):

    def __init__(self, name, color, model):
        # set the name and color by calling the parent constructor
        super().__init__(name, color)
        self.__model = model

    def getDescription(self):
        return self.getName() + self.__model + " in " + self.getColor() + "
color"

# in the getDescription() method, getColor() and getName() can be called
because
```

```
# inheritance makes it so the child class can access them
```

```
c = Car("Toyota Camry", "blue", "XLE")
```

```
print(c.getDescription())
```

```
print(c.getName()) # car does not have a getName() method but it can be  
accessed via the Vehicle class
```

We would expect to see this output:

```
Toyota Camry XLE in blue
```

```
Toyota Camry
```

What we did here was create a base class named `Vehicle` and a subclass called `Car`. We did not define the method called `getName()` in our `Car` class, but we still have access; this is because the `Car` class has inherited `getName()` from `Vehicle`. We used the `super()` method to call the base class method.

How does `super()` work?

Let's say that you have a method named `getInformation()`. It is in the base class, and you want to call it from the child class. That is possible with this code:

```
super().get_information()
```

Alternatively, the base class constructor could be called from the constructor in the child class, with this code:

```
super().__init__()
```

Multiple Inheritance

Unlike other languages, multiple inheritance is allowed in Python. This means that multiple classes may be inherited from simultaneously. Here's how:

```
class Subclass(SuperClass1, SuperClass2, ...):
```

```
    # initializer
```

```
    # methods
```

Here's an example of multiple inheritance in action:

```
class MySuperClass1():
```

```
    def method_super1(self):
```

```
        print("method_super1 method called")
```

```
class MySuperClass2():
```

```
    def method_super2(self):
```

```
        print("method_super2 method called")
```

```
class ChildClass(MySuperClass1, MySuperClass2):
```

```
    def child_method(self):
```

```
        print("child method")
```

```
c = ChildClass()
```

```
c.method_super1()
```

```
c.method_super2()
```

As an output, you should see:

```
method_super1 method called
```

```
method_super2 method called
```

Here you can see that MySuperClass1, MysuperClass2 are inherited by ChildClass, the ChildClass object can now have access to method_super1() and method_super2().

Method Overriding

Overriding a base class method requires the subclass to define another method that has an identical signature – it has the same name and parameter number as the base class method. Here's an example:

```
class A():
```

```
    def __init__(self):
```

```
        self.__x = 1
```

```
    def m2(self):
```

```
        print("m2 from B")
```

```
class C(B):
```

```
    def __init__(self):
```

```
        self.__y = 1
```

```
    def m1(self):
```

```
        print("m2 from C")
```

```
c = C()
```

```
c.m2()
```

The output should be

m2 from C

isinstance() Function

We use this function to work out if an object is a class instance or not and the syntax is:

```
isinstance(object, class_type)
```

Here's an example:

```
>>> isinstance(5, int)
```

True

```
>>> isinstance(5.3, int)
```

False

```
>>> isinstance([5, 6, 7, 8], list)
```

True

In the next chapter, we'll turn to exception handling

Exception Handling

Exception handling in Python is all about handling errors in the right way, perhaps having a message display on the screen if a specified file cannot be found. Errors are handled using a try, except code block, and the syntax is:

try:

```
# write a little code  
# that could throw an exception  
except <ExceptionType>:  
    # Exception handler, alert the user
```

In the try block, a piece of code needs to be written, that might potentially throw an exception. When that exception happens, the code inside the try block is ignored. If, in the except clause to the block, there is an exception type that matches, the clause handler will be executed. Here's an example:

```
try:  
    f = open('somefile.txt', 'r')  
    print(f.read())  
    f.close()  
  
except IOError:  
    print('file not found')
```

Here's how this works:

First, the initial statement written between the try block and the except block is executed.

If there is no exception thrown, the code written in the except clause is ignored.

If the specified file doesn't exist, an exception is raised, and the remainder of the try block code will be ignored.

When an exception is thrown, the code inside the except clause will be executed, provided the type matches with the exception name listed after the except keyword.,

Note:

The code above can only handle an IOError exception. If you want to handle other types of error or exception, you must add other except clauses.

Try statements can have multiple except clauses, along with an else and/or a finally statement as optional. Here's an example:

try:

 <body>

 except <ExceptionType1>:

 <handler1>

 except <ExceptionTypeN>:

 <handlerN>

 except:

 <handlerExcept>

 else:

 <process_else>

 finally:

 <process_finally>

The except clause is very much like the elif statement. When an exception happens, it will be checked to see if the type matches the type in the except clause. If it does, the matching handler will be executed. Note that ExceptionType has been omitted from the final except clause. If there is no match to the exception type, the handler in the final except clause will be executed.

Note:

Statements listed under the else clause will only run when there is no exception raised, whereas those listed in the finally clause run whether or not the exception is raised.

Have a look at an example:

try:

```
    num1, num2 = eval(input("Enter a pair of numbers with a comma separating them : "))
```

```
    result = num1 / num2
```

```
    print("Result is", result)
```

except ZeroDivisionError:

```
    print("Division by zero cannot be done and throws an error !!")
```

except SyntaxError:

```
    print("you have missed a comma; make sure you add a comma between the numbers, for example, 1, 2")
```

except:

```
    print("Incorrect input")
```

else:

```
    print("No exceptions")
```

finally:

```
    print("This executes, whatever happens")
```

Note:

Using the eval() function allows the program to run the code inside itself – eval() will expect to have a string argument.

Raising Exceptions

It is possible to raise an exception from one of your methods by using the raise keyword, like this:

```
raise ExceptionClass("the argument")
```

Here's an example:

```
def enterage(age):
```

```
    if age < 0:
```

```
        raise ValueError("You can only have positive integers")
```

```
    if age % 2 == 0:
```

```
        print("age is an even number")
```

```
    else:
```

```
        print("age is an odd number")
```

```
try:
```

```
    num = int(input("Input your age: "))
```

```
    enterage(num)
```

```
except ValueError:
```

```
    print("You can only have positive integers")
```

```
except:
```

```
print("something isn't right")
```

Try running this program, using a positive integer and see what the output is and then rerun it using a negative number.

Exception Objects

Learning how to handle and raise exceptions is one thing; you also need to know how you can access the exception object in the code. Assigning the exception object to a variable is done like this:

try:

```
# We expect an exception to be thrown from this code
```

```
except ExceptionType as ex:
```

```
# this code will handle the exception
```

You can see from this that we stored the exception object inside a variable named ex. This object may now be used in your handler code. Let's see an example:

try:

```
number = eval(input("Input a number: "))
```

```
print("The number you input is," number)
```

```
except NameError as ex:
```

```
print("Exception:", ex)
```

Run that program and input any number – you should see its output on your screen.

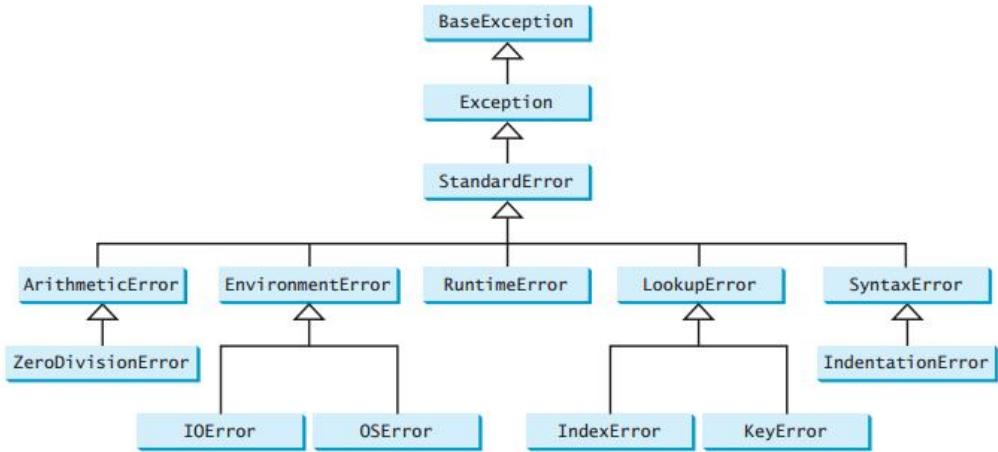
Run the program and enter a number.

Run it again, and this time, enter a string.

This time, you should see a ‘not defined’ exception on the screen.

Create A Custom Exception Class

It is also possible to create custom exception classes and we do this by extending our BaseException class or a BaseException subclass.



As you see in the above diagram, most Python classes are extended from the `BaseException` class. It is possible to derive an exception class of your own from the `BaseException` class or one of the subclasses.

In Python, make a file and name it `NegativeAgeException.py`. Add this code to it:

```

class NegativeAgeException(RuntimeError):

    def __init__(self, age):
        super().__init__()
        self.age = age
  
```

What this code does is creates another exception class and names it `NegativeAgeException`. It only has a single constructor, which will call the constructor from the parent class. It uses `super().__init__()` to do this, setting the age.

Using A Custom Exception Class

Here's how to use your custom exception classes:

```

def enterage(age):

    if age < 0:
        raise NegativeAgeException("You can only have positive integers")
  
```

```
if age % 2 == 0:  
    print("age is an even number")  
  
else:  
    print("age is an odd number")  
  
  
try:  
    num = int(input("Input your age: "))  
    enterage(num)  
  
except NegativeAgeException:  
    print("You can only have positive integers")  
  
except:  
    print("something isn't right. ")
```

For our penultimate section, we will look at Python modules.

Modules

Modules in the Python language are standard files with the ability to store classes, variables, constants, functions, and more. We use them to help us keep related code more organized and easier to read. For example, the math module contains all the functions related to mathematics.

Creating a Module

This is pretty easy to do. First, make a new file, saving it as mymodule.py. Then add this code to it:

```
foo = 150
```

```
def hello():  
    print("I come from the mymodule.py")
```

What we did here was to define a global variable called foo and a function called hello() inside the module. What if we want to use this module in a program? Like I showed you earlier, the first thing you must do is import the module and we use the import statement to do that:

```
import mymodule
```

Now, it is possible to use the function and the variable you defined in the module, and you do that like this:

```
import mymodule
```

```
print(mymodule.foo)
```

```
print(mymodule.hello())
```

Try it, and you should see the following on your screen:

```
150
```

```
i come from the mymodule.py.
```

Don't forget; to get access to any functions or variables in a module, you must make sure the module name is specified first; if you don't do this, you will get an error.

Using the From Statement With Import

When you use an import statement, everything inside the module gets imported. But what if you want a specific variable or function?

That is where we use the from statement, like this:

```
from mymodule import foo # statement only imports variable called foo  
from mymodule
```

```
print(foo)
```

The output should be

```
150
```

Note:

When you use the `from` statement, there is no need to specify the name of the module to get access to the functions and variables.

Using the `dir()` Method

This is one of the built-in methods, used for finding the object attributes – the available constants, functions, classes, and variables. You already know that everything is an object in Python, so we use the following syntax to find a module's attributes:

```
dir(module_name)
```

The `dir()` method will return a string that has a list of all the attribute names:

```
>>> dir(mymodule)
['__builtins__', '__cached__', '__doc__', '__file__',
 '__loader__', '__name__', '__package__', '__spec__', 'foo', 'hello']
```

You can see from this that, apart from `hello` and `foo`, there are some other attributes inside `mymodule`. These attributes are built-in by Python and appear in every module, regardless of what it is for.

Congratulations!

You have almost reached the end and have learned the basics of Python. To finish off, I am going to provide you with a series of tips to help you code better.

Beginner Tips for Learning Python

It is so exciting that you have chosen Python as your preferred computer programming language and, while I could tell you to go through this book over and again to learn the concepts, there are other ways to learn Python and become proficient in it.

The first step is in understanding how you should learn and is arguably the most crucial skill. Why do you need to understand this? Because as Python continues to evolve, more libraries get created, and the Python tools receive

upgrades. If you don't know how to learn, you can't keep up with all the changes and, if you can't do that, you have no chance of being successful.

That's why I want to give you these beginner tips, strategies that help you get your priorities straight and put you on the right path:

1. Write Code Every Single Day

The most important thing when you are learning to write code is consistency, and that is why you should commit to writing some code every day. You might not believe it, but there is one thing that has a significant effect on computer programming – muscle memory – and committing yourself to write a set amount of code each day will help that memory to develop. Try to start off writing for 20-30 minutes per day, even if it is the same thing over and again, and gradually build your time up.

2. Take Notes and Write Your Code Out

As with anything you are learning, it is crucial to take notes on your programming journey. It might interest you to know that writing notes by hand also makes it more likely that you will remember things, ideal for those who are planning a career as a developer – note – many developer interviews require you to write code in longhand on a whiteboard!

As you start to work on programs and projects, you will find it helpful to write your code out longhand. Not only will that help you to remember it and understand it, but it also helps you plan the code before you input it. It can save you a lot of time and heartache if you write out the classes and the functions you need and their interactions with one another.

3. Time to Get Interactive

It doesn't matter what part of Python you are learning, be it basic structures or how to debug a program,

One of the best tools you can use to learn is the interactive shell. You will sometimes see this as the REPL and, to use it, open a terminal and run Python.

Have a look at a few ways you can use it to learn the Python language:

Learn how to use the `dir()` method to perform operations on elements:

```
>>>  
>>> my_string = 'This is a string'  
>>> dir(my_string)  
['__add__', ..., 'upper', 'zfill'] # Truncated to make it easier to read
```

The `dir()` method will return all methods that can be applied to the specified element. As an example:

```
>>>  
>>> my_string.upper()  
>>> 'THIS IS A STRING.'
```

Here, we called a method named `upper()`, and it's clear to see how it works – all the string letters are turned into UPPERCASE letters.

Learn how to determine an element's type:

```
>>>  
>>> type(my_string)  
>>> str
```

Learn how to get documentation using the help system built-in:

```
>>>  
>>> help(str)
```

Learn how to import different libraries and get to know them:

```
>>>  
>>> from datetime import datetime  
>>> dir(datetime)  
['__add__', ..., 'weekday', 'year'] # Truncated to make it easier to read  
>>> datetime.now()  
datetime.datetime(2018, 3, 14, 23, 44, 50, 851904)
```

Learn how to run a shell command and what they do :

```
>>>  
>>> import os  
>>> os.system('ls')  
python_hw1.py python_hw2.py README.txt
```

4. Take plenty of breaks

This is very important; don't try to learn too much at once. Your brain can only soak in so much information at any one time, and you need to be able to step away and take in what you have learned. One of the best techniques to use is Pomodoro – work for 25 minutes and then take a break. Repeat.

Taking a break is especially useful when you debug a program because it can help you work out what is wrong. Step back; go for a short walk, do

something, and then, when you go back, you will probably find you can see the error. Even omitting one simple comma or quotation mark can cause an error in your programming, and a set of fresh eyes can help make a difference.

5. Debug Methodically

Bugs are an inevitable part of writing code, especially complex code – it happens to the most experienced of programmers. The one thing you shouldn't do is let the bugs get to you.

You must have a method to debug code, and the best way to do it is to go through it in the same order Python executes it – that way, you can make sure that every part works. When you think you have figured out where the problem lies, write this into the program script:

```
import pdb; pdb.set_trace()
```

Run the code to start the debugger, and you will go into interactive mode. You can also run it directly from the command line, using this command:

```
python -m pdb <my_file.py>
```

6. Get Together With Other Learners

One common misconception about coding is that it is something you do alone, but, in fact, the opposite is true. Coding works much better when you work with others and when you are learning, it is very important to get together with other Python learners. That way, you can share knowledge, tips, and brainstorm problems.

If you don't know anyone else who is learning, don't worry; search the internet and your local area for meetups, events, and so on. Failing that, there are plenty of Python forums that are very active – it's the next best thing.

7. Teach Python

One of the very best ways to learn is to teach, and the same goes with Python. There are a few ways to do this – get together with other learners and use a whiteboard to discuss concepts, write articles or blog posts talking

about concepts; you can set up a video and record yourself explaining something you learned or even chat to yourself while you are learning. Any of these methods will highlight gaps in your knowledge and help to cement your learning.

8. Pair Programming

This technique is where a pair of developers work together, at a single workstation, on a task. They will each take turns at being the ‘driver,’ writing the code, and the ‘navigator,’ helping to guide the process of problem-solving and reviewing the written code.

The switch between the two should be as frequent as possible. There are some benefits to this – first, you get somebody to look at the code you write and, second, you get to see another point of view in terms of solving problems. And this can all help you when you go back to working by yourself.

That brings us to the end of this quick start guide; you are well on your way to learning the Python programming language.

Conclusion

First of all, thanks for purchasing my guide and well done for reaching the end. I hope that you found it useful and learned the basic skills and knowledge you need to grasp the Python programming languages.

The next step is to further that knowledge by delving deeper into Python, seeing just what it can do for you, and how powerful it truly is. If you are considering a career in data science, you will need extensive knowledge of Python; you need it for designing games, tools, web apps, and much more besides.

You may have noticed that this is not a complex guide. There is nothing worse for a beginner than being bamboozled with long, complicated code examples and explanations of the features – short, concise, and simple is far better.

I hope that you gained some useful knowledge from this and are ready to take the next step. Thank you once again for purchasing it, and good luck on your journey.

References

- <https://www.programiz.com/>
 - <https://thepythonguru.com/>
 - <http://www.pyladies.com/>
 - <https://www.dummies.com/>
 - <https://www.python.org/>
 - <https://www.pythonforbeginners.com/>
 - <https://realpython.com/>
 - <https://www.guru99.com/>
-

Python Machine Learning

**Complete and Clear Introduction to the
Basics of Machine Learning with Python.
Comprehensive Guide to Data Science and
Analytics.**

Alex Campbell

Introduction

Machine learning has long been in the spotlight, and it's not likely to disappear. It is one of the most prominent areas of data science and artificial intelligence, and it offers some great employment opportunities; it certainly isn't as difficult to get into as you might think. Even if you aren't experienced in programming or even math, you can still get involved in machine learning because the most important things are your motivation and interest.

And that's why you are here.

In this guide to machine learning with Python, I'm going to give you an overview of all the most important topics.

- Chapter One – an overview of machine learning, where you learn exactly what it is and what it's all about
- Chapter Two – we'll delve into the different types of regression
- Chapter Three – we'll look at classification algorithms
- Chapter Four – an overview of unsupervised learning, including some algorithms
- Chapter Five – a machine learning project for you to try
- Chapter Six – an introduction to data science with Python, why you want to learn Python and the five steps to becoming a data scientist
- Chapter Seven – a list of ten things you should all know about machine learning

What I won't be doing is teaching you the Python programming language. You do need to know the basics, but you can find those in my book, titled "Python Computer Programming - Simple Step-By-Step Introduction to the Python Object-Oriented Programming: Quick Start Guide for Beginners."

Why Python?

Python is the very best computer programming language for beginners to cut their teeth on, especially in machine learning and data science. It is an easy language to get to grips with, and it offers all the libraries and

frameworks you need, significantly cutting the time it will take to get results.

Are you ready to learn machine learning with Python?

Then let's dive in .

Chapter One: An Overview of Machine Learning

Before we dive into the machine learning models, we should take a step back and learn what machine learning is and isn't. It is often termed as an Artificial Intelligence subfield, but that's a bit misleading. Sure, machine learning came from AI research, but, as far as data science goes, it's better to consider machine learning as a way of building models filled with data.

Simply put, machine learning is all about mathematical models, built to help us understand data. The 'learning' part of it comes in when we provide the models with parameters that are adaptable to observed data. First, the models are shown a certain set of data, and, from that, they can learn to identify, classify, and make predictions on previously unseen data. That is a simple way of explaining it, but by the time you get to the end of this chapter, you will have a better understanding. Basically, the machine is learning in a similar way to how the human brain learns. Knowing what it's all about is one thing; putting it into perspective is another, and one of the critical points to using machine learning models correctly is understanding the types of problems they can be used on. So, let's kick off with a few categorizations to explain things.

Machine Learning Categories

At its most basic, we can split machine learning into two primary categories – supervised and unsupervised learning.

Supervised learning revolves around measured data features and the relationship with associated labels. When a supervised learning model has been built and trained, it can then be used to label new data that it hasn't seen before. We can divide supervised learning further into regression and classification tasks. You'll be learning about these in more depth later, but, for now, classification labels are discrete categories, and, in regression, the labels are continuous quantities. You'll see examples of these shortly.

Unsupervised learning works with unlabeled data and includes tasks like dimensionality reduction and clustering. Dimensionality reduction

algorithms look for succeeding data representations, while clustering algorithms identify distinct data groups. Again, you will see examples of these shortly.

There are also semi-supervised models which, as you can guess, are somewhere between the two. These tend to be used when the data labels are incomplete.

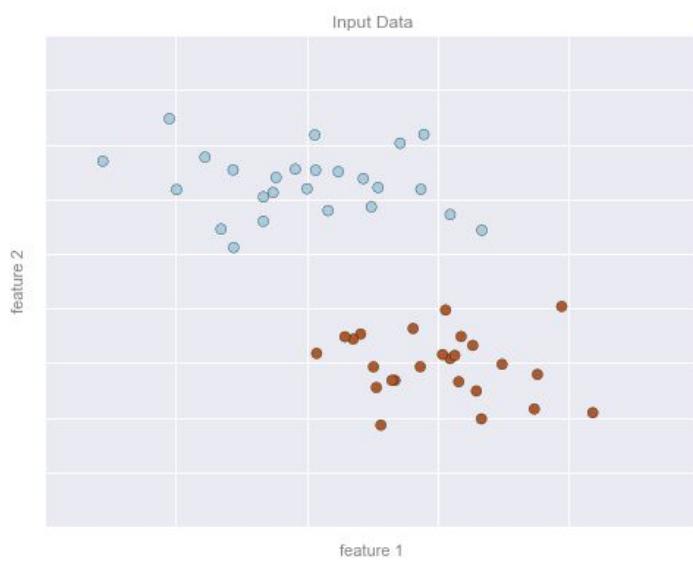
Examples of Machine Learning Applications

Let's give these ideas some flesh by looking at some examples of a task. These examples are designed to give you a basic overview of the different types of tasks we'll be looking at; later, we'll go more in-depth.

All images attributed to <https://jakevdp.github.io/>

Classification: Predicting Discrete Labels

First, a classification task. Here, you have some labeled points that we want to use for classifying unlabeled points. Let's assume that the data we have is this:

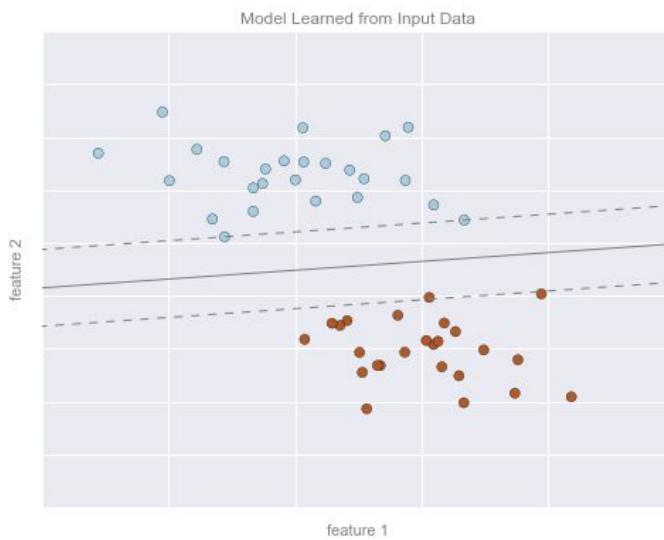


The data is two-dimensional, which means each point has two features. The x, y positions of those points represent the features on the plane. Each point also has one of two possible class labels, indicated by the colors for the points. We want to use those labels and features to build a model that predicts if new points should be given a 'blue' or 'red' label.

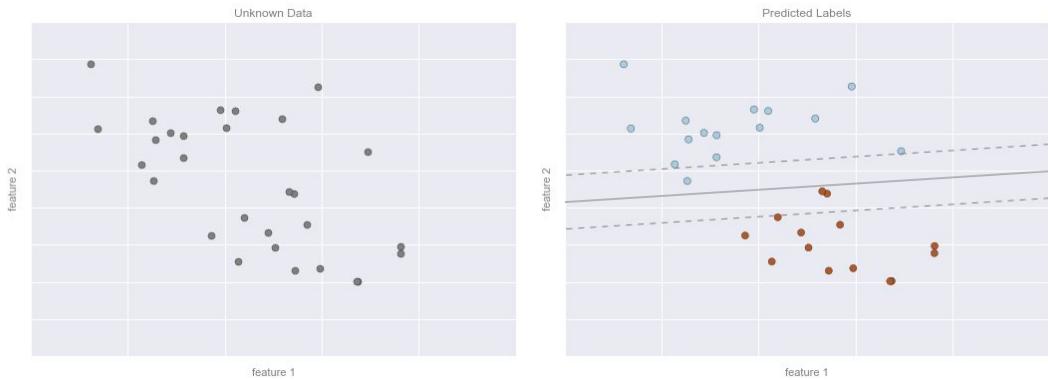
Classification tasks like this can be done using several different models, but we're going to use a simple one. We will assume that we can separate the groups with a simple straight line drawn between them on the plane; in this way,

The points on either side belong to the same group. The parameters are the numbers that indicate the location of the line and its orientation. The model will learn the parameter's optimal values from the data in a process known as 'training.'

In the next image, you can see what the trained model will look like:



Once the model is trained, we can generalize it to new data that has no labels. In simple terms, we draw the model line through a new set of data, assigning the new points with labels based on that model. This is known as 'prediction.'



That gives you some idea of how classification tasks work, with 'classification' an indication that the class labels on the data are discrete. This looks pretty trivial at first look; you would find it easy to look at the data and classify it by drawing a discriminatory line, but machine learning offers a simple benefit – we can generalize it to huge datasets in multiple dimensions.

As an example, think about a machine learning task to detect spam email automatically. We could have these labels and features:

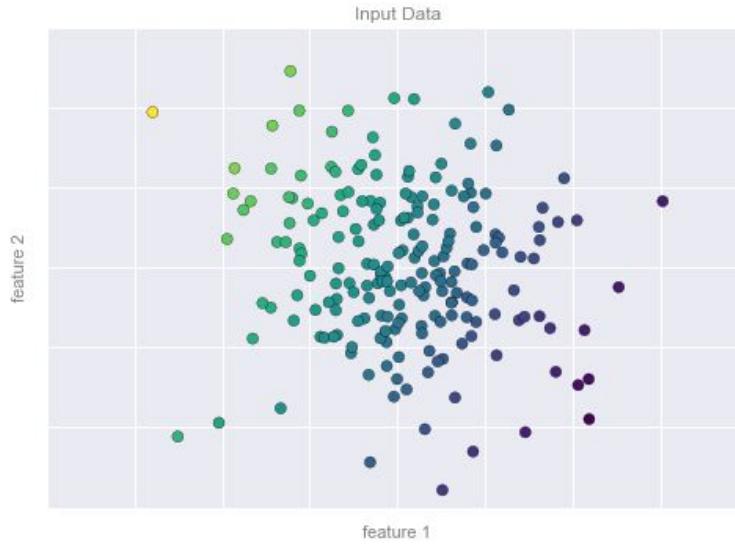
- Feature 1, feature 2, etc. – normalized counts of words and phrases that are important in identifying spam email, i.e., Nigerian Prince, Viagra, etc.
- Label – Spam or Not Spam

On the training dataset, we could determine the labels by inspecting a sample set of emails individually, and, for the rest of the emails, the models would determine which label to apply. If a classification algorithm is properly trained with well-constructed and defined features, this can be an effective approach.

Regression: Predicting Continuous Labels

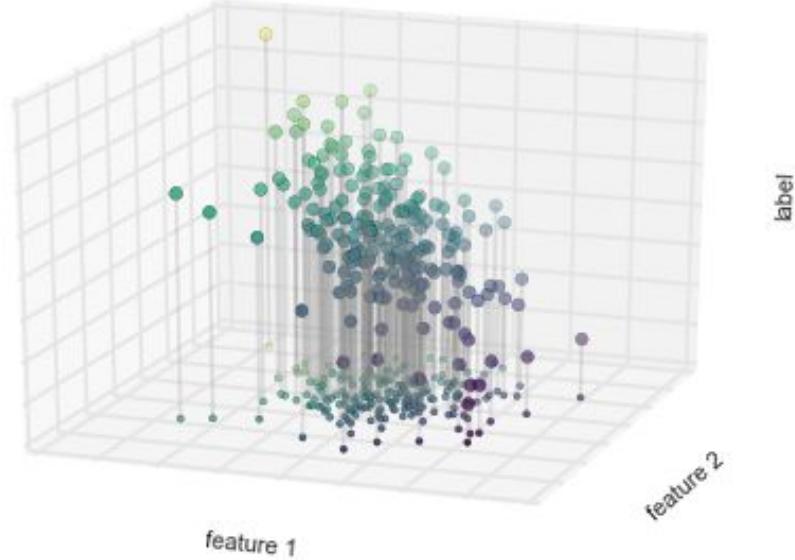
Regression tasks, by contrast, uses labels that are continuous quantities. Have a look at the next image,

Showing a set of points, each having a continuous label.



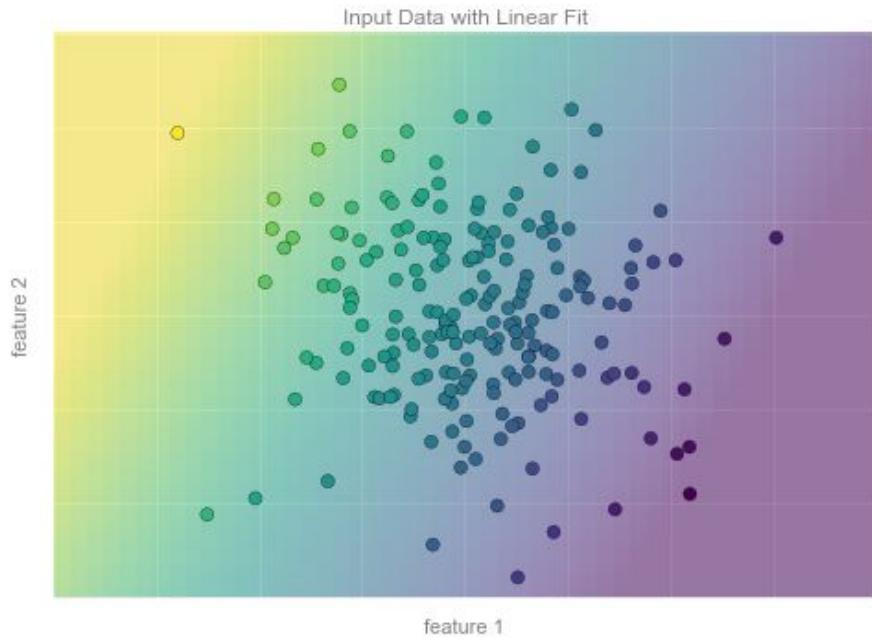
As you saw with the classification, the data is two-dimensional, each data point having two features. The colors indicate the point's continuous label. We could choose from any number of regression models for this, but we're going to make the points predictions using linear regression. The model is assuming that, if the label is treated as a third spatial dimension, the plane can be fitted to the data.

This can be visualized as follows:

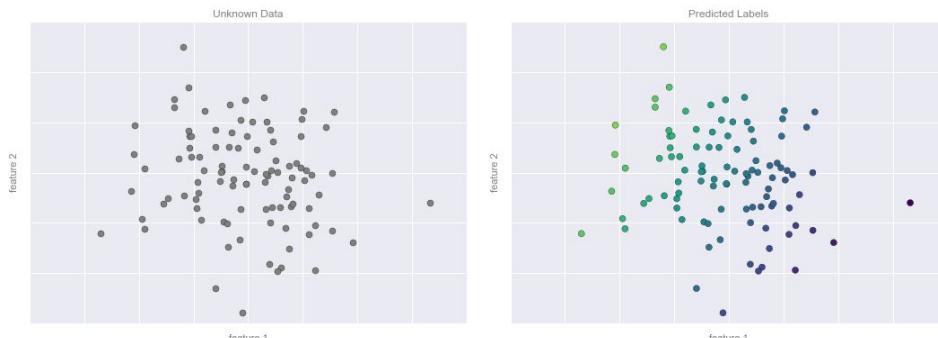


The plane for feature 1-feature 2 is the same as it was in the plot from earlier, but this time, the labels are represented using a three-dimensional axis position and color. It doesn't seem unreasonable that we could predict labels for any input parameters by fitting the plane through the data. Using

the two-dimensional data, though, we would get the following result.



This plane provides exactly what is needed for predicting labels on new points. In visual terms, the results would look like this:



Again, this may appear trivial when the number of dimensions is low, but these methods are powerful, in that we can apply them in a straightforward manner, and evaluate them on data with multiple features.

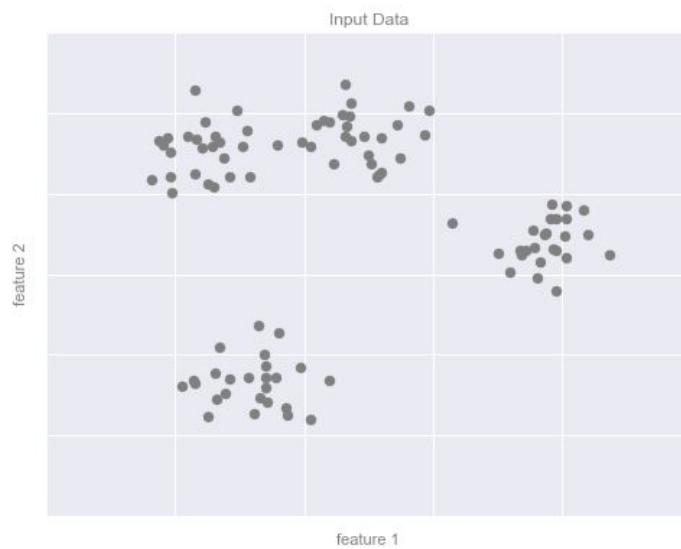
We could liken this to a task that computes how far away galaxies are when we look at them through a telescope, and the following labels and features can be used:

- Feature 1-feature 2 – how bright each galaxy is at a given color or wavelength
- Label – the galaxy's redshift or distance

We could use a more expensive independent observation set to determine the distance for a few galaxies and then estimate the remaining distances using a regression model. Doing this, we would not need to use the original expensive observation over the dataset.

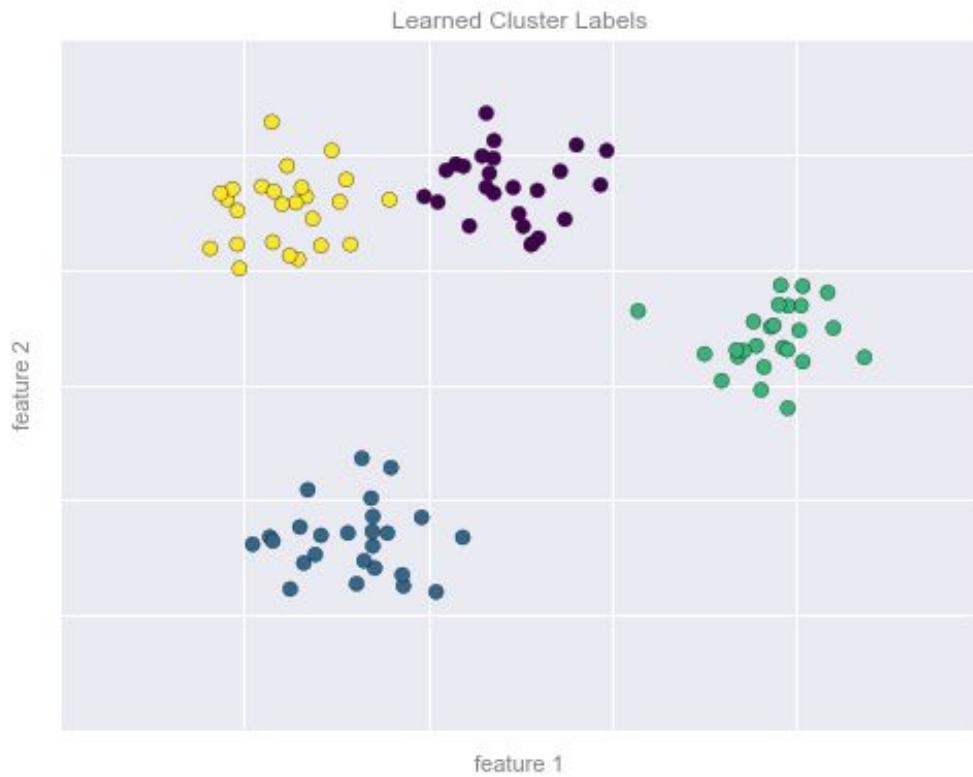
Clustering: Inferring the Labels on Unlabeled Data

We've just looked at two supervised learning examples; regression and classification. We tried to build models to predict the labels on new, unseen data, But unsupervised learning requires us to build models that don't have known labels to learn from and are, in effect, making blind predictions. Clustering is the most common method, whereby a specific number of discrete groups are assigned with data automatically. The next image shows two-dimensional data:



Just by looking, you can see the distinct groups of points, and, using that input, clustering models will use the data structure to work out the related points.

The next figure shows the result of using the k-means algorithm:



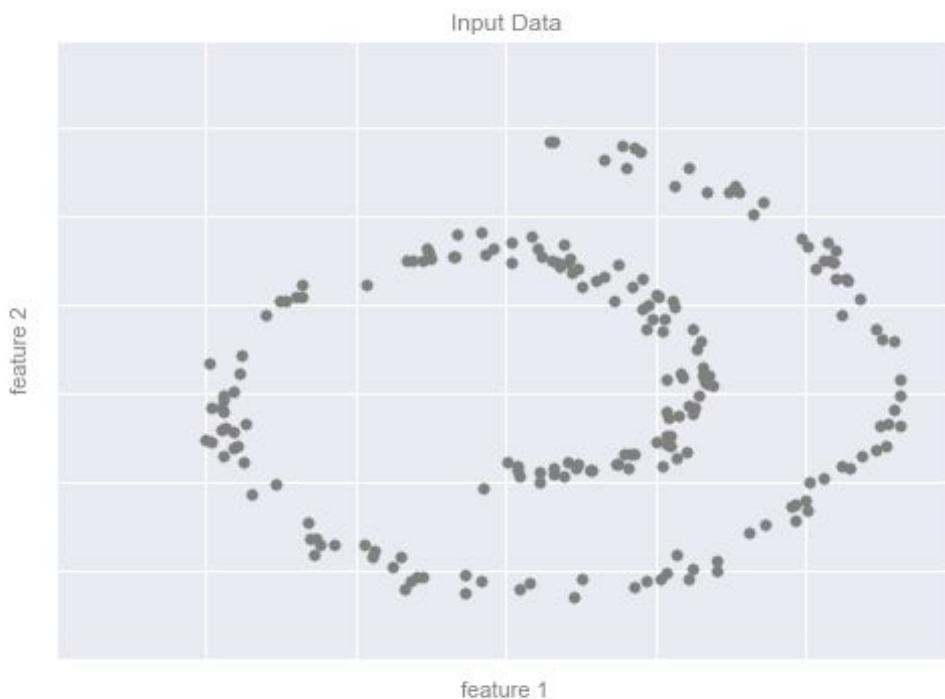
The k-means algorithm is used for fitting models that have k cluster centers. It is assumed that the optimal centers are those that minimize how far each point is from the center assigned to it. Again, with two-dimensional data, this probably seems trivial, and, as we get larger data, more complex, clustering algorithms become useful for extracting information from the data.

Dimensionality Reduction: Inferring the Structure of Unlabeled Data

Our final example is another unsupervised algorithm called dimensionality reduction. This algorithm infers information, such as labels, from the dataset structure.

It is somewhat abstract, compared to the other examples, but it is useful for pulling out low-dimensional data representations that, in some way, maintain the entire dataset's qualities.

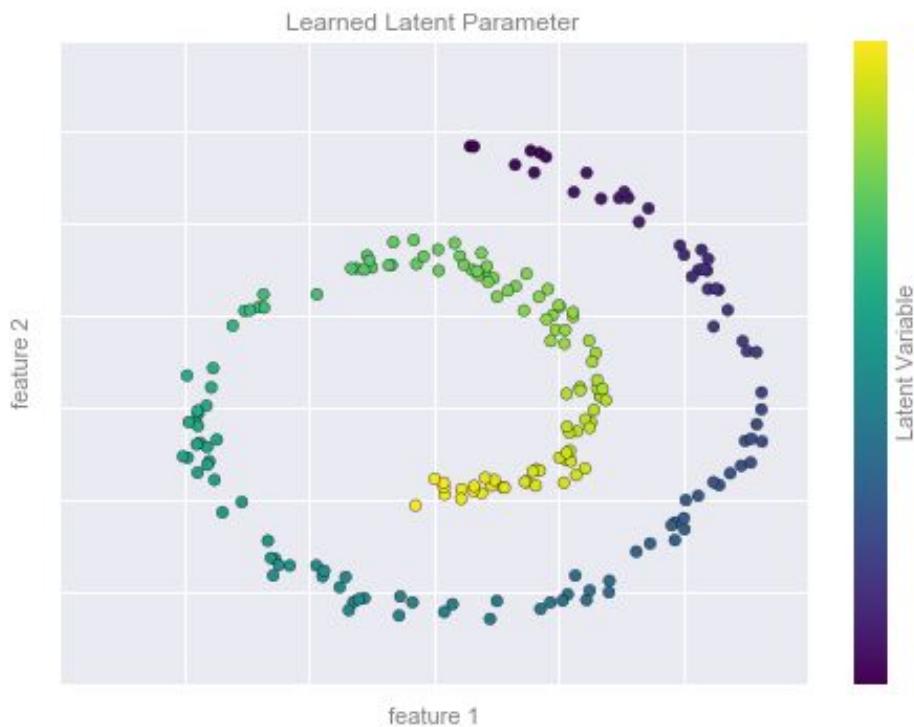
Have a look at the example below:



You can see that the data has a kind of structure; a one-dimensional line in a spiral has been drawn in the two-dimensional space.

In a way, we could say that we have one-dimensional data, even though it is higher-dimension space. In this case, the right model would have a

sensitivity to this embedded structure, which is nonlinear, and would easily pull the representation out. You can see the results from a manifold learning algorithm called Isomap below:



The colors are representing the one-dimensional latent variable that was extracted; the change along the spiral is uniform, indicating the algorithm detected the structure we saw above. As with the other examples, the higher the dimension, the more powerful the algorithm. If, for example, we may want important relationships in a dataset with 100 or 1000 features visualized.

It becomes challenging when we work with $-$ -dimensional data, and using dimensionality reduction is one way of reducing the data down by several dimensions and making it easier to manage.

That ends this brief discussion on machine learning, with a few examples to make it clearer, and now it's time to go in-depth. We'll start with Regression.

Chapter Two: Regression Machine Learning Models

We looked at regression briefly in the first chapter, but what exactly is? Regression is a technique that examines variables, looking for relationships between them. As an example, you could look at a handful of employees from a large company and see if you could work out how their salaries were dependent on certain features, such as education, experience, where they work, their role, and so on.

That is a regression problem; individual pieces of data relating to each employee are one observation, and there is a presumption that the independent features are the education, experience, role, and where they work (city), while the salary is the dependent feature.

In a similar fashion, you could try to determine how house prices were mathematically dependent on where they are, how many bedrooms, how close to the city center they are, and so on.

Typically, when we do regression analysis, we are looking at a specific point of interest together with multiple observations. Every observation has at least two features and, if we assume that one or more of those features is dependent on the other, we can try to find a relationship between them.

In simpler terms, we need a function that can sufficiently map some variables or features to others.

Dependent features are also known as outputs, dependent variables, or responses, while the independent features are known as inputs, independent variables, or predictors.

Usually, a regression problem consists of one independent variable that is unbounded and continuous. The dependent variables may be discrete, continuous, or categorical, such as nationality, gender, brand, etc.

Common practice dictates that the outputs are denoted with a y , and the inputs are denoted with an x . If there are two independent variables (or more), the vector, $x = (\text{? } \text{? }_1, \dots, \text{? } \text{? }_r)$, with r being the number of inputs, is used to represent them.

When is Regression Required?

Regression is typically used to answer how or if a phenomenon influences another one or if there is a relationship between multiple variables. As an example, it could be used for determining if gender or experience have an impact on salary and, if yes, to what extent.

Regression is also good for forecasting responses based on a set of new predictors, such as predicting a household's electricity consumption for the hour ahead, given the time of day, the external temperature, and how many people live there.

Regression is used in several fields, including computer science, economy, and social sciences, to name a few. As each day passes, it grows in importance as the amount of available data grows, and we have a better awareness of the value of that data in practical terms.

Different Types of Regression Techniques

There are a few regression techniques that can be used for making predictions and are driven by three primary metrics. They are:

- How many independent variables there are
- What type the independent variables
- What shape the regression line is

We'll look at some of those techniques now.

Linear Regression

This is the commonest regression technique and is typically the first one learned for predictive modeling. With linear regression, we have a continuous dependent variable, continuous or discrete independent variables, and a linear regression line.

The equation used to represent linear regression is $Y=a+b*X + e$. To break this down:

- a is the intercept
- b is the line slope
- e is the error term.

We can use the equation for predicting a target variable's value, based on supplied predictor variables.

Linear regression comes in two flavors – simple and multiple. The only difference between the two is in the number of independent variables – simple has just one, while multiple linear regression has more than one.

Now, how do you get the best fit line, which is the value of $a+b$?

We can do this using a technique called Least Square Method, or LSM, the commonest way to fit regression lines. LSM works out the fit line for the data by "minimizing the sum of the squares of the vertical deviations from individual data points to the line." When the deviations are added, they are squared first, thus avoiding any cancellations between negative and positive values.

Logistic Regression

We use logistic regression to find what the probability of event=Failure and event=Success is, i.e., classification problems. In theory, logistic regression should be used when there is a binary value for the dependent variable, i.e., 0 or 1, Yes or No, True or False.

In logistic regression, the Y value is from 0 to 1 and the equation below represents it:

$\text{odds} = p / (1-p) = \text{probability of event occurrence} / \text{probability of not event occurrence}$

$$\ln(\text{odds}) = \ln(p/(1-p))$$

$$\text{logit}(p) = \ln(p/(1-p)) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \dots + b_kX_k$$

In this equation, p represents the “probability of presence of the characteristic of interest.” What you should be asking yourself is, why did we use logit?

Dependent variables are binomial distributions, and they require that the link function used suits the distribution; in this case, that's the logit function.

The parameters in the equation above ensure the likelihood of observing sample values is maximized, rather than the sum of the squared errors being minimized, as it would be in standard regression.

Things to keep in mind with logistic regression are that a linear relationship between dependent and independent variables is not required, as it is with linear regression. Logistic regression has the ability to handle multiple relationship types as a nonlinear log transformation is applied to the odds ratio predicted.

To ensure that the model neither under nor overfits, all the most significant variables should be included. However, larger sample sizes are needed because the estimates for maximum likelihood are not as powerful at lower samples than OLS, or Ordinary Least Square is.

There also shouldn't be any correlation between the independent variables (multicollinearity).

Polynomial Regression

Regression equations are classed as polynomial, provided the independent variable has a power of more than 1. The following equation is representative of polynomial equations – $y=a+b*x^2$.

In polynomial regression, a straight line is not considered a best-fit line; instead, a curve is the best fit, fitting into the individual data points.

There is always a temptation to try to fit a high-degree polynomial as a way of lowering the error, but the result will almost always be over fitting. The relationships should always be plotted so you can see the fit, and you should focus on ensuring the curve fits the problem.

Stepwise Regression

Stepwise regression is typically used when we have several independent variables. With stepwise, an automatic process is used to choose the independent variables; no human intervention is required at all. This is done through the observation of certain statistical values, such as t-stats, R-square, and the AIC metric, used for the discernment of the significant variables.

Stepwise regression fits the model by adding co-variates or dropping them, one at a time, based on set criteria. Below are the commonest stepwise methods:

- **Standard** – this method of stepwise regression adds the predictors and removes them as required for every step.
- **Forward Selection** – this begins at the model predictor that is most significant and adds variables for every step
- **Backward Elimination** – this begins with every model prediction, and, at each step, the least significant one is removed.

The idea of stepwise regression is to ensure the prediction power is maximized using the least number of variables; it is typically used with high-dimensionality data.

Ridge Regression

This technique is used in cases of multicollinearity or when there is a high correlation between the independent variables. With multicollinearity, although OLS (ordinary least squares) is unbiased, it has large variances, deviating the observed from the true value by quite a long way.

When you add a degree of bias to the estimates, the standard errors are reduced by ridge regression.

Going back to the linear regression equation, which is $y=a+b*x$, there is also an error term built-in. The entire equation will be:

$y=a+b*x+e$ (error term), [error term is the value needed to correct for a prediction error between the observed and predicted value]

=> $y=a+y= a+ b_1x_1+ b_2x_2+....+e$, for multiple independent variables.

With linear equations, we can decompose the prediction errors down into sub-components – the first for biased and the second for variance. A prediction error can happen because of one or both of these, and we're going to look at the one caused by variance.

With ridge regression, a shrinkage parameter of λ (lambda) solves the problem of multicollinearity. Have a look at the equation:

$$= \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \underbrace{\|y - X\beta\|_2^2}_{\text{Loss}} + \lambda \underbrace{\|\beta\|_2^2}_{\text{Penalty}}$$

Here, there are two components. The first is the least-square term, while the second is the lambda of the β^2 or beta-square summation (in which the coefficient is represented by β).

The second is added to the first as a way of shrinking the parameter for low variance. A couple of things to remember about ridge regression are, firstly, that the ridge regression and least-square term assumptions are the same, except we cannot assume normality. Second, when you use ridge regression, the coefficient values are shrunk, but they will never get to zero, suggesting that there isn't a feature for feature selection. Lastly, as ridge regression is one of the regularization methods, it uses l2 regularization.

Lasso Regression

Much like the previous regression technique, Lasso, which stands for Least Absolute Shrinkage and Selection Operator, penalizes the regression coefficient absolute size. It can also reduce variability and make linear regression more accurate. Look at the following equation:

$$= \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \underbrace{\|y - X\beta\|_2^2}_{\text{Loss}} + \lambda \underbrace{\|\beta\|_1}_{\text{Penalty}}$$

Where lasso and ridge regression differ is in the way absolute values, rather than squares, are used in the penalty functions. The result of this is "the constraint of the sum of the absolute values of the estimates," or penalizing the values. This will make some parameter estimates come out as zero. The bigger the applied penalty, the more the estimates shrink towards absolute zero, and the result is a variable selection from the supplied n variables.

Important points to remember are, first, similar to ridge regression, that the assumptions match those of least squared regression, but normality cannot be assumed.

Second, when you use lasso regression, the coefficients are shrunk to exactly zero, providing a helping hand for the feature selection.

Being another regularization method, it uses l1 regularization and, where there is a high correlation between predictor groups, only one is chosen, and the rest shrunk to zero.

ElasticNet Regression

Our final technique is a hybrid of the ridge and lasso regression techniques. Training is done with l1 and l2 to regularize them. This is a useful technique when there are many correlated features. Where lasso picks one randomly, ElasticNet picks both.

One of the advantages of the trade-off between ridge and lasso regression techniques is that ElasticNet will inherit stability under rotation from the ridge technique.

L1 and L2 Regularization

Overfitting is when the analysis result corresponds very closely with the data set and may result in problems fitting more data or reliably predicting observations. Regularization is a technique to help improve the training and prevent overfitting. Two of the most common regularization techniques are l1 and l2.

L1

The lasso regression techniques use l1 regularization. To see how it works, we'll define a linear regression model, Y, that has a single independent variable.

W is representing Weight, while b is representing bias.

$$W = w_1, w_2, \dots, w_n \\ X = x_1, x_2, \dots, x_n \\ W = w_1, w_2, \dots, w_n \\ X = x_1, x_2, \dots, x_n$$

The result that is predicted is

$$\hat{Y} = Y$$

$$\hat{Y} = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

The formula below will calculate what the error is without using a regularization function:

$$\text{Loss} = \text{Error}(Y, \hat{Y})$$

And this one uses the l1 function:

$$\text{Loss} = \text{Error}(Y - \hat{Y}) + \lambda \sum |w_i|$$

Note

If the lambda value were zero, the loss function above would become OLS (ordinary least square); having oversized values shrinks the weights (coefficients) to zero, causing underfitting.

One thing you do need to note is $|w|$ becomes differentiable when $w \neq 0$:

$$\frac{d|w|}{dw} = \begin{cases} 1 & w > 0 \\ 0 & w = 0 \\ -1 & w < 0 \end{cases}$$

To help you understand this, we can use the gradient descent optimizer to find new weights:

$$w_{\text{new}} = w - \eta \frac{\partial L}{\partial w}$$

When l1 is applied to this, it will become:

$$\begin{aligned} w_{\text{new}} &= w - \eta \cdot (\text{Error}(Y, Y^{\wedge}) + \lambda d|w|dw) = \{w - \eta \cdot (\text{Error}(Y, Y^{\wedge}) + \lambda)w - \eta \cdot \\ &(\text{Error}(Y, Y^{\wedge}) - \lambda)w > 0 \\ w < 0 & \quad w_{\text{new}} = w - \eta \cdot (\text{Error}(Y, Y^{\wedge}) + \lambda d|w|dw) = \{w - \eta \cdot \\ &(\text{Error}(Y, Y^{\wedge}) + \lambda)w > 0 \\ w - \eta \cdot (\text{Error}(Y, Y^{\wedge}) - \lambda)w < 0 & \end{aligned}$$

From this formula:

Where w is positive, the $\lambda > 0$ regularization parameter pushes w , so it isn't so positive. It does through subtraction of λ from w .

Where w is negative, the $\lambda < 0$ regularization parameter pushed w , so it isn't so negative. It does this through the addition of λ to w , and this will push w nearer to 0.

Simple Python Implementation

```
def update_weights_with_l1_regularization(features, targets, weights,  
lr, lambda):
```

```
"
```

```
    Features:(200, 3)
```

```
    Targets: (200, 1)
```

```
    Weights:(3, 1)
```

```
""
```

```
    predictions = predict(features, weights)
```

```
#Extract our features
```

```
x1 = features[:,0]
```

```
x2 = features[:,1]
```

```
x3 = features[:,2]
```

```
# Use matrix cross product (*) to simultaneously
```

```
# calculate the derivative for each weight
```

```
d_w1 = -x1*(targets - predictions)
```

```
d_w2 = -x2*(targets - predictions)
```

```
d_w3 = -x3*(targets - predictions)
```

```
# Multiply mean derivative by the learning rate
```

```
# and subtract from weights (gradient points in the direction of steepest ASCENT)
```

```
weights[0][0] = (weights[0][0] - lr * np.mean(d_w1) - lambda) if  
weights[0][0] > 0 else (weights[0][0] - lr * np.mean(d_w1) + lambda)
```

```
weights[1][0] = (weights[1][0] - lr * np.mean(d_w2) - lambda) if  
weights[1][0] > 0 else (weights[1][0] - lr * np.mean(d_w2) + lambda)
```

```
    weights[2][0] = (weights[2][0] - lr * np.mean(d_w3) - lambda) if  
    weights[2][0] > 0 else (weights[2][0] - lr * np.mean(d_w3) + lambda)
```

```
return weights
```

L1

regularization and variations if it is typically chosen when there are multiple features because the solutions provided by it are sparse. Because the features that have zero weights can be ignored, it provides an advantage in computational terms.

L2

Ridge regression uses the l2 technique, and the primary difference between l1 and l2 is that the latter makes use of the coefficient's squared magnitude, as a way of penalizing the loss function.

We'll define a linear regression model, called Y, with a single independent variable to see how this works.

Again, W is representing Weight while b is representing bias.

$$W=w_1, w_2, \dots, w_n \quad X=x_1, x_2, \dots, x_n \quad W=w_1, w_2, \dots, w_n \quad X=x_1, x_2, \dots, x_n$$

The result that is predicted is $\hat{Y} = Y^T Y$

$$\hat{Y} = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b \quad Y = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$$

The formula below will calculate the error without using a regularization function:

$$\text{Loss} = \text{Error}(Y, Y^{\wedge})$$

While this one uses the l2 function to calculate it:

$$\text{Loss} = \text{Error}(Y - Y^{\wedge}) + \lambda \sum 1n w_2 i$$

Note

As you can imagine, where lambda is zero, we get OLS back. However, where the lambda is large, too much weight is added, resulting in underfitting.

To help you understand this, we can substitute our formula and use gradient descent to find the new weights.

$$w_{\text{new}} = w - \eta \partial L_2 \partial w$$

When l2 is applies to this, we get:

$$\begin{aligned} w_{\text{new}} &= w - \eta \cdot (\text{Error}(Y, Y^{\wedge}) + \lambda \partial L_2 \partial w) \\ &= w - \eta \cdot (\text{Error}(Y, Y^{\wedge}) + 2\lambda w) \end{aligned}$$

Simple Python Implementation

```
def update_weights_with_l2_regularization(features, targets, weights, lr, lambda):
```

""

Features:(200, 3)

Targets: (200, 1)

Weights:(3, 1)

""

predictions = predict(features, weights)

#Extract our features

x1 = features[:,0]

x2 = features[:,1]

x3 = features[:,2]

Use matrix cross product (*) to simultaneously

calculate the derivative for each weight

d_w1 = -x1*(targets - predictions)

d_w2 = -x2*(targets - predictions)

d_w3 = -x3*(targets - predictions)

```

# Multiply mean derivative by the learning rate

    # and subtract from weights (remember gradient points in the direction
    # of steepest ASCENT)

    weights[0][0] = weights[0][0] - lr * np.mean(d_w1) - 2 * lambda *
    weights[0][0]

    weights[1][0] = weights[1][0] - lr * np.mean(d_w2) - 2 * lambda *
    weights[1][0]

    weights[2][0] = weights[2][0] - lr * np.mean(d_w3) - 2 * lambda *
    weights[2][0]

return weights

```

L2 regularization is used in cases of multicollinearity; it can reduce this through the constraint of the coefficient norm and retaining every variable. L2 regression is good for estimating the importance of predictors, penalizing those that have the least importance. One of the biggest problems with co-linearity is the huge parameter estimate variance. Where there are more features than observations, the OLS matrix is typically not invertible but, by using ridge regression, it can be inverted.

Chapter Three: Classification Machine Learning Models

Classification models are used on both unstructured and structured data. The technique categorizes data into a specified number of classes, and the primary goal is to identify the class or category that new data will fall into.

Some of the terminology you will come across includes:

- **Classifier** – algorithm for mapping input data to categories
- **Classification Model** – a machine learning model that takes input values for training purposes and attempts to draw conclusions from it. These models predict categories and labels for new data
- **Features** – an individual property of an observed phenomenon; these properties are measurable
- **Binary Classification** – a classification task that has two potential outcomes, for example, a gender classification task, Male/Female
- **Multi-class Classification** – a classification that has two or more classes; every sample is assigned to just one target label, for example, an insect may be a fly or a spider, but it cannot be both at the same time
- **Multi-label Classification** – classification, where the samples are individually mapped to sets of target labels, i.e., two or more classes. For example, news articles can be about people, locations, and activities at the same time.

To build a classification model:

The classifier you want to use must be initialized

- The classifier must then be trained.
- The classifiers included in Scikit-learn use a fit method (X , y) for training or fitting the model. The training data is X , and the training label is y .
- The target is predicted. With an observation, X , that has no label, predict(X) will return y , the expected and predicted label
- The model is evaluated
- **Initialize** the classifier to be used.
- **Train the classifier:** All classifiers in scikit-learn uses a fit(X , y) method to fit the model(training) for the given train data X and train label y .
- **Predict the target:** Given an unlabeled observation X , the predict(X) returns the predicted label y .
- **Evaluate** the classifier model

Different Classification Algorithms for Python

There are several major classification algorithms, and the one you use will depend entirely on the problem to be solved.

Logistic Regression

Logistic regression is one of the primary classification algorithms. The algorithm uses a logistic function to model the probabilities that describe the potential outcomes.

It is specifically designed for classification and is one of the best models to help understand how multiple independent variables affect the outcome variable.

However, it will only work with binary predicted variables; it assumes that every predictor is independent of the others and that the data has no missing values.

Naïve Bayes

Naïve Bayes is based on the Bayes' Theorem, and it has the assumption that there is independence between feature pairs.

These classifiers are well-suited to real-world problems, such as email spam filtering and classifying documents. For the right parameters to be estimated, a small amount of training is required, but the Naïve Bayes classifiers are much faster than other sophisticated classification methods.

However, the downside is, it is not a good estimator.

Stochastic Gradient Descent

This is one of the simpler approaches to fitting linear models, not to mention incredibly efficient. It is a very useful algorithm to use when you have large numbers of samples, and it has support for multiple classification penalties and loss functions.

It is very easy to implement, but several hyper-parameters are required, and it has a high sensitivity to feature scaling.

K-Nearest Neighbors

This is one of the lazier methods of learning as the model makes no attempt to build an internal model; it just stores the training data instances. The classification is done via a computation taking the majority vote of each point's k-nearest neighbors.

It is a simple algorithm to implement, stands up to training data with a lot of noise in it, and, if you have a large training data set, is very effective. However, the k value must be determined first, and, in computation terms, it

is an expensive algorithm – each instance's distance to all training samples must be computed.

Decision Tree

Decision trees take attribute data and corresponding classes, producing a rule sequence for classifying data. It is an easy algorithm to understand, and even easier to visualize. The data doesn't need much preparation, and it can take categorical and numerical data.

However, some trees are complex, and generalization is not very good. Decision trees also react to data variations; one small variation could end up with the generation of a totally different tree.

Random Forest

This is known as a 'meta estimator.' The random forest classifier is used for fitting different decision trees on different dataset sub-samples. It improved the model's predictive accuracy using average and has over-fitting under control. The size of the sub-sample will always be the same size as the input sample size.

The advantages to the random forest classifier are that it can reduce overfitting and, typically, they are far more accurate than standard decision trees. However, the downsides are that implementation isn't easy, the algorithm itself is complex, and real-time prediction is very slow.

Support Vector Machine

This algorithm represents the training data as points in space, all in categories separated clearly by a wide gap. New data examples get mapped to the space, too, and, depending on what side of the gap they land, a category is predicted.

While the SVM is very effective for high-dimensional spaces, and while a training points subset is used in the decision function to ensure an efficient

memory, you don't get probability estimates directly; instead, five-fold cross-validation is used, but this is computationally expensive.

Accuracy

Classification Algorithms Accuracy F1-Score

Logistic Regression	84.60%	0.6337
Naïve Bayes	80.11%	0.6005

Stochastic Gradient Descent	82.20%	0.5780
--------------------------------	--------	--------

K-Nearest Neighbors	83.56%	0.5924
---------------------	--------	--------

Decision Tree	84.23%	0.6308
---------------	--------	--------

Random Forest	84.33%	0.6275
---------------	--------	--------

Support Machine	Vector	84.09%	0.6145
--------------------	--------	--------	--------

In the next chapter, we'll take a look at unsupervised learning algorithms.

Chapter Four: Unsupervised Machine Learning Algorithms

In terms of machine learning, unsupervised learning is where the model does not require any supervision. What that means is, the machine learning model works alone, usually with unlabeled data, discovering information. With unsupervised learning algorithms, you can move on to complex tasks, more complex than those you can process with supervised learning. However, it is a more unpredictable way of learning, compared to other methods.

Let me give you an example of unsupervised learning.

Let's say that we have a baby girl and a family pet, a cute dog. She knows this dog; she can identify it. A couple of weeks later, her mom's friend comes over, bringing her own dog with her.

Now, although the baby girl has never seen this dog, she does recognize a few features. She can see it has two eyes and four legs, like the family dog. So, she can identify this new animal as a dog.

That is unsupervised learning, where, rather than being taught something, you learn it from the data available to you. In the case of supervised learning, the baby girl would have been told that the new animal was a dog.

Why Choose Unsupervised Learning?

These are the primary reasons why you would use unsupervised learning:

- Unsupervised learning is used for finding patterns in data that you wouldn't otherwise see
- Unsupervised learning helps find useful features for categorization
- Unsupervised learning is in real-time; all the input data analysis and labeling is done with the learners
- Unlabeled data is easier to obtain from computers than labeled data is because labeled data requires human intervention.

Different Types of Unsupervised Machine Learning

Unsupervised learning typically falls into two groups – association and clustering problems.

Clustering

Clustering is incredibly important in unsupervised learning because it primarily looks at finding patterns or structures in a series of data that hasn't been categorized.

These algorithms process data, finding groups, or clusters of data (if they exist). It is possible to modify the number of clusters the algorithm should find.

There are a few types of clustering algorithms:

- **Exclusive (partitioning)** - Data is grouped in a way that each data may only be a part of a single group; it cannot belong to two or more groups. An example of this is the k-means algorithm.
- **Agglomerative** – Each data in this type of algorithm is a cluster. The cluster numbers are reduced by way of iterative unions between two close clusters. An example of this is hierarchical clustering.
- **Overlapping** – This technique used fuzzy sets for clustering the data. Each data point may belong to multiple clusters, but each will have its own degree of membership. The data is then associated with the correct membership value. An example of this is the Fuzzy C-Means algorithm.
- **Probabilistic** – The probabilistic technique creates clusters using a probability distribution. For example, keywords such as "man's hat," "woman's hat," "man's scarf" and "woman's scarf" would be clustered into two separate categories, either "man" and "woman" or "hat" and "scarf."

Types of Clustering

There are several different types of clustering, and we're going to discuss six of them here briefly.

Hierarchical Clustering

This algorithm builds up a cluster hierarchy. This starts with the data that has already been assigned to its own cluster; two clusters close together will be within the same cluster, and the algorithm ends when only a single cluster remains.

K-Means Clustering

This is an iterative algorithm that looks for the largest value in each iteration. To start with, you select the required number of clusters. The data points must be clustered into k groups – the larger the k, the smaller the groups with more granularity, and the smaller the k, the larger the groups with less granularity.

The algorithm outputs a group of labels, assigning a data point to every k group. The k groups are defined by a centroid in each group; these are seen as the heart of each cluster, capturing the nearest points and adding them to the cluster.

K-Means clustering can be broken down into two more subgroups – dendrogram and agglomerative.

- **Agglomerative Clustering**

The k-means clustering subgroup begins with a set number of clusters. All the data is allocated to that number. The technique doesn't need to know the k number of clusters as the input, and the agglomeration begins when each data is formed into one cluster. Distance measures are used in agglomerative clustering, one measure for each iteration, for reducing the number of clusters by merging the processes. To finish, there is one large cluster that has all the objects or data in it.

- **Dendrogram Clustering**

In this technique, every level is representing a potential cluster. The dendrogram's height indicates how much similarity there is between two of the join clusters; the nearer the process bottom they are, the more similar the clusters are. Dendograms tend to be unnatural and subjective techniques and are not used that often.

K-Nearest Neighbors

This is, by far, the simplest classifier. It is different from other techniques in that a model isn't produced. A simple algorithm, it stores the cases already available, and new instances are then classified using a similarity measure.

It works best when the examples have some distance between them, and, on large training sets, it is a slow learner.

Principal Components Analysis (PCA)

PCA is typically used for higher-dimensional space. A basis must be chosen for the space, along with the top 200 important scores. That basis is the principal component, and the subset you choose is a new, small space (smaller than the original). The technique will keep as much data complexity as it can.

Association

This technique lets you find associations between objects in big databases, basically, relationships between variables. As an example, if a person buys a house, they are highly likely to purchase new furniture.

Supervised Machine Learning vs. Unsupervised Machine Learning

Parameters	Supervised Machine Learning Technique	Unsupervised Machine Learning Technique
Input Data	Labeled data is used to train the algorithms.	Unlabeled data is used for training the algorithm
Computational Complexity	Simple method	Computationally complex
Accuracy	An accurate method you can trust	Not so accurate and not so easy to trust

Unsupervised Machine Learning Applications

Some of the more common applications for unsupervised machine learning:

- **Clustering** – the dataset is split automatically into groups, based on similarities
- **Anomaly Detection** – it can find strange data points and is typically used for detecting fraudulent bank transactions
- **Association Mining** – sets of items that often happen together are identified in the dataset
- **Latent Variable Models** – these are typically used for preprocessing data, i.e., reducing the features, decomposing datasets into several components, etc.

The Disadvantages of Using Unsupervised Learning

- In terms of data sorting, it isn't possible to obtain precise information.
- The data output is labeled and unknown
- The results are not so accurate because the input data is unknown and hasn't been labeled by a human beforehand. As such, the machine has to do the labeling itself
- Spectral and informational classes do not always correspond with one another.
- A user must take the time to interpret the classes and label them following classification
- Class spectral properties can change, which means the same class information cannot be used when you move between images.

In the next chapter, quite a long one, we're going to finally get down to some practical work with a complete machine learning project.

Chapter Five: Your First Machine Learning Project

There really is only one way you can learn machine learning, and that is to get on with it. Design your own projects, complete them, see where you went wrong and put it right.

Python is one of the most popular of all the interpreted computer languages, not to mention the most powerful. It is a complete platform and language you can use for research, development, and the development of a production system.

Python also has many libraries and modules built-in, giving you plenty of ways to complete every task. It can, to a complete beginner, feel quite overwhelming, so, as I said earlier, the best thing to do is to dive in and get your hands dirty – that is the best way to learn what works and what doesn't.

This does the following:

- It makes you install the Python interpreter and start it
- It lets you see how to work your way through a project

Rather than pushing you right in at the deep end and expecting you to create a project (you couldn't, not at this stage), I'm providing you with a complete project to follow.

Sure, you can buy books or sign up to course, but rarely will they give you anything but the bare bones and a few snippets of code. I could have done that, but I preferred to walk you through a project with all the code needed from start to finish. That way, you can see how it all fits, how it all works together, and you may even be able to see where you can make improvements or change things slightly to get a better result.

No machine learning project is linear, but they all have one thing in common – the same basic steps:

1. Define the problem
2. Prepare your data
3. Evaluate your algorithms

4. Improve the results
5. Present the results

The project below will walk you through everything, from defining your problem, loading your data, summarizing it, evaluating your algorithms, and making a few predictions.

If you can successfully work through this and come out the other end understanding it, you have a template for future use. Later, you can add more details, such as preparing the data further and improving your results later, when you have more confidence.

The Hello World of Python Machine Learning

When you learn how to code in Python, you typically start with the Hello World program. It's a simple one, and it gets you started. In the same way, with machine learning, we typically start by using the iris dataset to classify iris flowers.

This is one of the easiest projects because it is simple to understand, and here's why:

1. It has numeric attributes, meaning you need to work out how to load the data and then handle it
2. It lets you get your feet wet with one of the simpler supervised machine learning algorithms, classification
3. It is a multi-nominal classification problem, which means it has multiple classes; that means it may have to be handled in a special way
4. It has just four attributes and only 150 rows; that makes it small enough to handle easily, and it will fit into memory, on an A4 page, even on one screen
5. The numeric attributes are all the same scale and the same unit, so there is no need for any specialized scaling or any transforms.

Step-by-Step Tutorial

What we're going to do now is go, one step at a time, through a machine learning project. Here's what you will learn:

1. How to install Python and SciPy
2. How to load the dataset
3. How to summarize the dataset
4. How to visualize the dataset
5. How to evaluate a few algorithms
6. How to make some predictions

The important thing is that you take your time. Work through each step and don't move onto the next one until you understand how to do it and what it all means.

You can copy and paste the commands and code but, where you can try to type it in yourself – it helps you understand the Python language better.

Step One – Download, Install and Start Python SciPy

The first step is to install Python and then SciPy onto your computer – I did cover how to do this in my previous book, and it is really well covered elsewhere, too, so I don't want to waste time on it. You should already have Python on your machine. Installing SciPy is pretty simple too.

Install the SciPy Libraries

I am going to assume that you already have Python 2.7 or 3.6+ on your computer, so you need to look at five libraries that you will need to complete this tutorial:

- SciPy
- NumPy
- Matplotlib
- Pandas
- Sklearn

There are loads of ways to get these libraries installed; pick one and stick with it to install every library you need. If you go to the SciPy installation page, it will tell you how to install the libraries on different platforms.

For example, Mac users can install Python 3.6 plus all these libraries using macports. Linux users can use the package manager that comes with your distribution, i.e., if you use Fedora, your package manager is called yum. And if you are a Windows user and not confident about installing these libraries, your best bet is to install a Python distribution called Anaconda, as it contains everything you will need.

Please make sure you have version 0.2 or above of Scikit-learn installed.

Start Python and Check the Version

The next step is to make sure you have the right Python environment installed and that it all works as it should. The script you will see in this

step will test your environment, and it will import the libraries you need and tells you what version of each one you have

Open your command line and type the following command to start the interpreter:

Python

It's best if you work in the interpreter or use the command line to write and run your scripts, rather than messing about with IDEs and editors.

That way, it keeps everything neat and tidy, allowing you to focus your attention on machine learning and not on what tools you need.

Type in the following:

```
# Check library versions  
# Python version  
import sys  
print('Python: {}'.format(sys.version))  
    # scipy  
        import scipy  
print('scipy: {}'.format(scipy.__version__))  
# numpy  
  
    import numpy  
  
print('numpy: {}'.format(numpy.__version__))  
  
# matplotlib  
    import matplotlib
```

```
print('matplotlib: {}'.format(matplotlib.__version__))

# pandas

import pandas

print('pandas: {}'.format(pandas.__version__))

# scikit-learn

import sklearn

print('sklearn: {}'.format(sklearn.__version__))

You should get an output similar to this (this came from a Mac):

Python: 3.6.9 (default, June 20, 2020 05:21:45)

[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.2)]

scipy: 1.3.1

numpy: 1.17.3

matplotlib: 3.1.1

pandas: 0.25.1
```

sklearn: 0.21.3

Compare this to what you get. In an ideal world, you should have something similar; if anything, the versions may be later. As far as the API's go, as they change very little, there's no need to worry if your versions are a little earlier as everything will still work. If an error occurs, stop; now is the best time to resolve it. If the script doesn't run cleanly, you will not be able to go any further with the tutorial, so run a Google search on your error and find out how to fix it.

Step Two - Load Your Data

As explained, we'll be using the popular iris flowers dataset. This is one of the most famous as it used the world over to teach people machine learning, the "Hello World" if you like.

The dataset has 150 iris flower observations. It has four columns, each containing the flower measurements in centimeters. A fifth column displays what species each flower is; every iris flower in the dataset will belong to one of three different species.

The first step will be to import the dataset.

Import the Libraries

First, we need to import a lot of stuff – the modules, the objects, and the functions we need for the tutorial, so type or copy/paste this to the command line:

```
# Load libraries
```

```
from pandas import read_csv  
from pandas.plotting import scatter_matrix
```

```
from matplotlib import pyplot
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import StratifiedKFold
```

```
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import confusion_matrix  
from sklearn.metrics import accuracy_score  
  
from sklearn.linear_model import LogisticRegression  
  
from sklearn.tree import DecisionTreeClassifier  
  
from sklearn.neighbors import KNeighborsClassifier  
  
from sklearn.discriminant_analysis import
```

LinearDiscriminantAnalysis

```
from sklearn.naive_bayes import GaussianNB  
  
from sklearn.svm import SVC
```

You should find it all loads quite easily, without any errors. Again, if an error occurs, stop and find out what has happened. You cannot proceed until your SciPy environment is fully working.

Load the Dataset

We don't need to do anything special here; the dataset can be directly loaded from the source, the UCI Machine Learning Repository.

To load the dataset, we need pandas, and we'll also need it for exploring our data, with data visualization and descriptive statistics. Note – when we load the data, the name of each column is

specified; you will find this very helpful later when it comes to exploring the data.

Type or copy/paste the following:

```
# Load dataset
```

```
url=
```

```
https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv
```

```
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',  
'class']
```

```
dataset = read_csv(url, names=names)
```

Again, there shouldn't be any problems with loading the dataset. If you find you have network issues, find and download the iris.csv file, save it in your working directory, and then use the same way to load it – make sure the URL is changed to the local file name.

Step Three - Summarizing the Dataset

Time to actually look at what data we have and we can do that in a few ways:

- The dataset dimensions
- Peeking
- A statistical summary of the attributes
- Breaking the data down based on class variables

Don't panic; each of these requires just one command, and these are incredibly useful commands that you will use time and again in your projects.

Dataset Dimensions

We can see the number of rows (instances) and columns (attributes) the dataset contains using a property called shape and the following command:

```
# shape
```

```
print(dataset.shape)
```

you should see the following, indicating the number of instances (150) and the number of attributes (5):

(150, 5)

Peeking at the Data

It's never a bad idea to see the data so use the following

command:

```
# head
```

```
print(dataset.head(20))
```

You should get shown the top 20 data rows, like this:

sepal-length sepal-width petal-length petal-width class

Sepal-length sepal-width petal-length petal-width class

5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5.0	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5.0	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa
4.8	3.0	1.4	0.1	Iris-setosa
4.3	3.0	1.1	0.1	Iris-setosa
5.8	4.0	1.2	0.2	Iris-setosa
5.7	4.4	1.5	0.4	Iris-setosa
5.4	3.9	1.3	0.4	Iris-setosa
5.1	3.5	1.4	0.3	Iris-setosa
5.7	3.8	1.7	0.3	Iris-setosa
5.1	3.8	1.5	0.3	Iris-setosa

Statistical Summary of the Attributes

Next we can look at a statistical summary, including mean, count, min values, max values, and a few percentiles. Again, use this command:

```
# descriptions
```

```
print(dataset.describe())
```

As you can see from the result, the numerical values are all in centimeters and are all within a similar range – 0 to 8 centimeters.

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

The Class Distribution

Now we want to see how many rows or instances should eb with each class and this can be seen as an absolute count, using the following command:

```
# class distribution
```

```
print(dataset.groupby('class').size())
```

Each class shares the same amount of instances – 33% or 50 instances.

```
class
```

```
Iris-setosa    50
```

```
Iris-versicolor 50
```

```
Iris-virginica 50
```

A Complete Example

To make it nice and easy for you, we can put all of this together in one simple script:

```
# summarize the data
from pandas import read_csv
# Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)
# shape
print(dataset.shape)
# head
print(dataset.head(20))
# descriptions
print(dataset.describe())
# class distribution
print(dataset.groupby('class').size())
```

Step Four - Visualizing the Data

So, we have some ideas about our data; the next step is to visualize it. We'll look at two different plots:

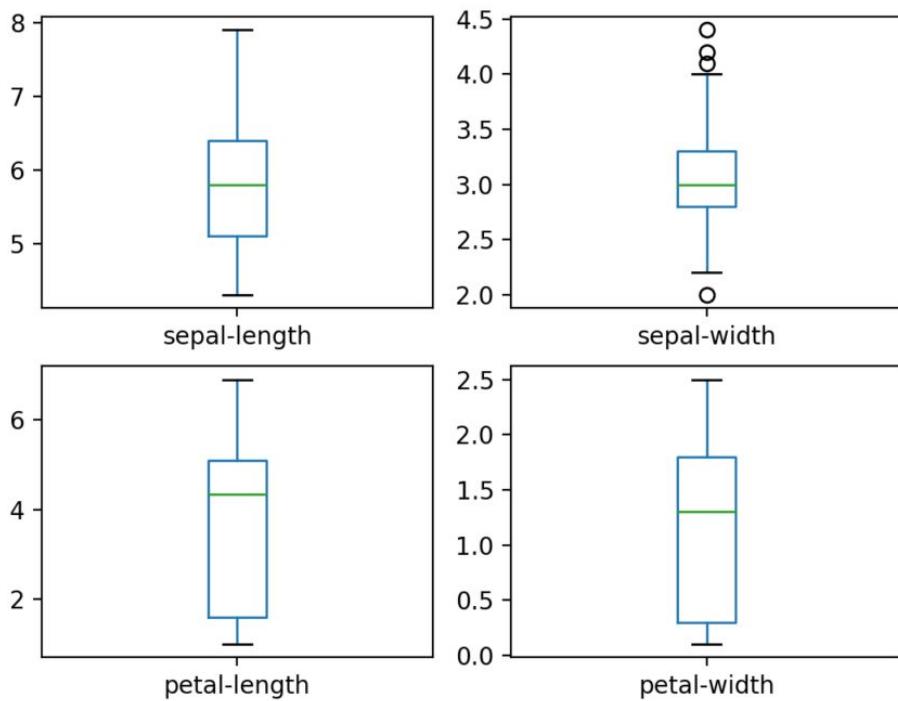
1. Univariate Plots – to get a better understanding of the attributes
2. Multivariate Plots – to get a better understanding of the relationship that exists between the attributes.

The Univariate Plot

A univariate plot will plot each variable individually. Because the variables (input) are numeric, we can use a box and whiskers plot for each variable. Here's how you do that:

```
# box and whisker plots  
dataset.plot(kind='box',      subplots=True,      layout=(2,2),      sharex=False,  
sharey=False)  
pyplot.show()
```

This will show you clearly how the input attributes are distributed



Box

and Whisker Plots for the Individual Input Variable in the Iris Flowers Dataset (www.machinelearningmastery.com)

Another way to see the distribution would be with a histogram:

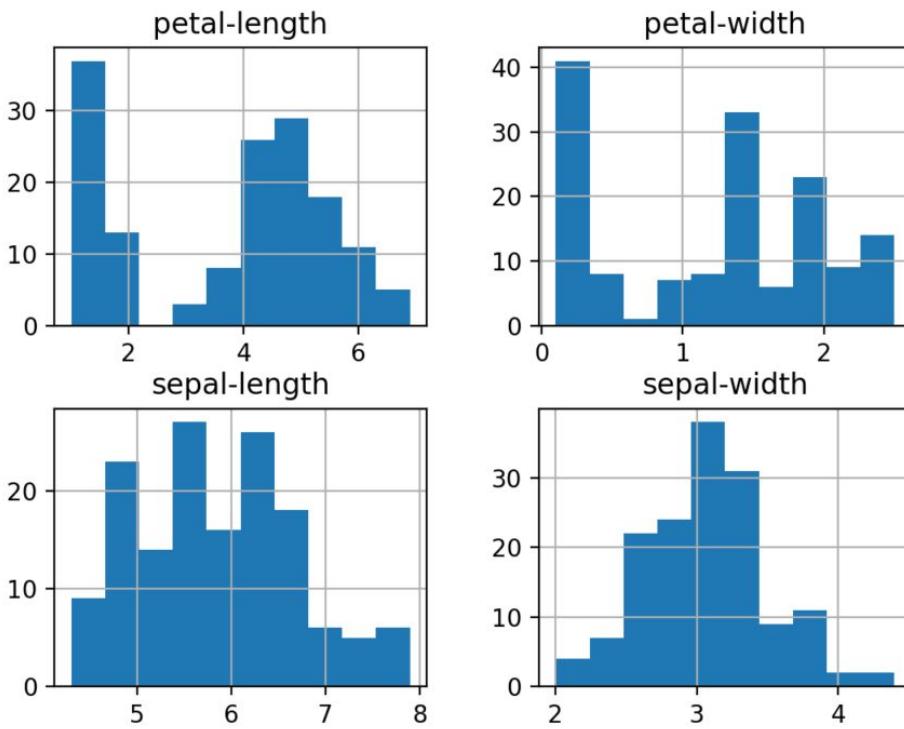
```
# histograms
```

```
dataset.hist()
```

```
pyplot.show()
```

It looks, from the results, like a couple of the variables have what is known as a Gaussian distribution.

This is quite useful information as there are some algorithms that can be used for exploiting that:



Histogram Plots for the Individual Input Variables In the Iris Flowers Dataset (www.machinelearningmastery.com)

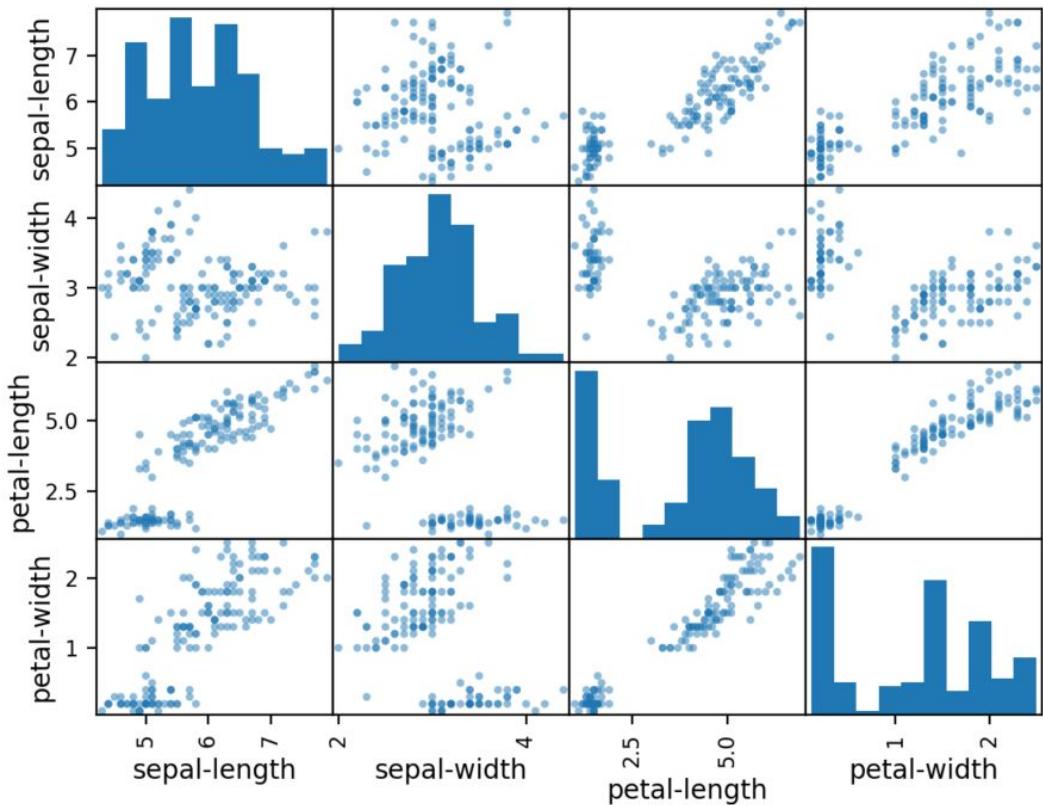
The Multivariate Plots

Next, we need to see what interactions exist between the variables. First, we'll create scatterplots showing all the variable pairs. This is useful for spotting what if any structured relationships exist between the input variables.

Here's the command:

```
# scatter plot matrix
scatter_matrix(dataset)
pyplot.show()
```

Note from the result that some attribute pairs are grouped diagonally, suggesting a predictable relationship and a high degree of correlation.



Scatter Matrix Plot for the Individual Input Variables in the Iris Flowers Dataset
www.machinelearningmastery.com

A Complete Example

Again, to make things easy, here's all of that tied up in one script:

```
# visualize the data
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
# Load dataset
url="https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)
```

```
# box and whisker plots  
dataset.plot(kind='box',      subplots=True,      layout=(2,2),      sharex=False,  
sharey=False)  
pyplot.show()  
  
# histograms  
dataset.hist()  
pyplot.show()  
  
# scatter plot matrix  
scatter_matrix(dataset)  
pyplot.show()
```

Step Five – Evaluating Some Algorithms

The next step is to create some data models, estimating how accurate they are when provided with unseen data.

We'll cover:

1. How to separate a dataset for validation

2. How to set up a test-harness that uses a technique called 10-fold cross-validation

3. How to build several models that use flower measurements to predict the species

4. How to choose the best model

Creating the Validation Dataset

It's important to know that our model is working well. Later, we'll be using some statistical methods as a way of estimating how accurate our models are on unseen data and we'll also want a strong estimate of how accurate the best model is on the unseen data – we can only do that by giving it real unseen data and evaluating how accurate it is.

What that means is we need to hold a section of our dataset back, not letting the algorithms see it,

and then we can use it as the unseen data that will independently confirm whether the best model is accurate or not.

To do this, our dataset needs to be separated into two subsets – 80% of the data for training, evaluation and selection, and the remaining 20% to validate the model. Here's the command to do that:

We need to know that the model we created is good.

```
# Split-out validation dataset  
array = dataset.values  
X = array[:,0:4]  
y = array[:,4]  
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,  
test_size=0.20, random_state=1)
```

We have now got two sets – the training data is split into X_train and Y_train for model preparation and we also have X_validation and Y_validation for use later on.

You might have noticed that a Python slice was used for choosing the NumPy array columns. If you have a basic understanding of Python, which you should have for this book, then you will have come across slicing before.

Creating the Test-Harness

We want to estimate how accurate the model is, and, to do that, we will make use of stratified 10-fold cross-validation.

This will result in the dataset being split down into ten parts. We'll use nine of them for training and one for testing; then, we'll repeat for every different combination there is for training and testing splits.

By stratified, I mean that each split (fold) of the original dataset will try to have an identical distribution class examples that the original, whole dataset

has.

The random seed is set to a fixed number using the argument of random state. This will make sure that the algorithms are all evaluated using the same training dataset split.

It doesn't matter what the specific random seed is, so long as they all use it.

We are also going to evaluate our models use the 'accuracy' metric,

which is a ration of how many instances are predicted correctly divided the how many instances are in the dataset in total, and then multiplied by one hundred to provide a percentage of accuracy, i.e., 97% accurate. The scoring variable will be used next when the models are run, built, and evaluated.

Build the Models

At this stage, we have no idea which would be the best algorithms to use for this or the specific configurations to use. We can get a reasonable idea by looking at our plots, that we have partly linearly separable classes in some of the dimensions, so the results should be pretty good.

We're going to test six algorithms:

Let's test 6 different algorithms:

- Logistic Regression
- Linear Discriminant Analysis
- K-Nearest Neighbors
- Classification and Regression Trees
- Gaussian Naive Bayes
- Support Vector Machines

This is a pretty good combination of simple linear algorithms and nonlinear algorithms.

Here's how we build our models and evaluate them:

```

# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='liblinear',
                                         multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))

# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
                                 scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

```

Select the Best Model

So, we have our models and we have an accuracy estimate for each one. The next step is to compare all the models to one another and choose the one that is most accurate. If we run the above example, we would get something like this:

LR: 0.960897 (0.052113)

LDA: 0.973974 (0.040110)

KNN: 0.957191 (0.043263)

CART: 0.957191 (0.043263)

NB: 0.948858 (0.056322)

SVM: 0.983974 (0.032083)

What we can see here is that the SVM (Support Vector Machine) shows the biggest accuracy score (estimated) of 98% or 0.98.

A plot can be created showing the evaluation results for each model, and showing each model's spread and mean accuracy. Each algorithm has an accuracy measure population because we evaluated each one ten times, using 10-fold cross-validation.

One of the best ways of comparing the result's samples for the individual algorithms is with a box and whisker plot for the individual distributions;

those distributions are then compared:

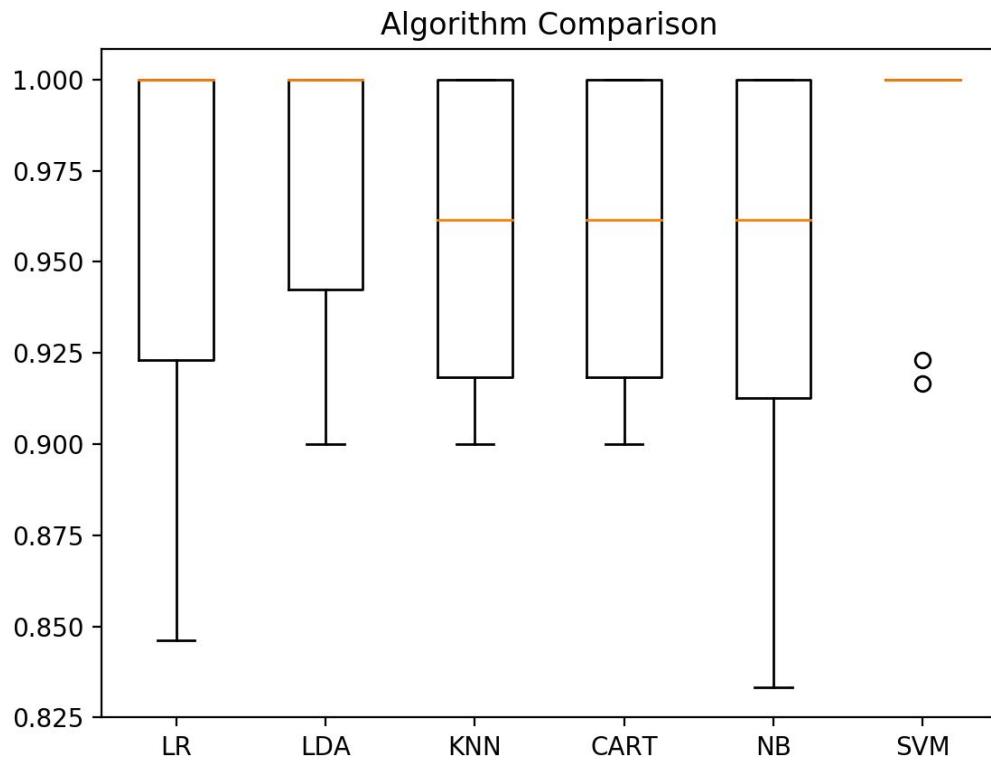
```
# Compare Algorithms
```

```
pyplot.boxplot(results, labels=names)
```

```
pyplot.title('Algorithm Comparison')
```

```
pyplot.show()
```

It's easy to see the plots squashed in at the range's top, with lots of the evaluations coming in at 100% accuracy. Some do push down to around the high 80's as well.



Box and Whisker Plot Comparing the Machine Learning Algorithms on the Iris Flowers Dataset (www.machinelearningmastery.com)

A Complete Example

Again, for you reference, here it is all tied together:

```
# compare algorithms
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

```

from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

# Load dataset
url="https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)

# Split-out validation dataset
array = dataset.values
X = array[:,0:4]
y = array[:,4]

X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
test_size=0.20, random_state=1, shuffle=True)

# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='liblinear',
multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))

# evaluate each model in turn
results = []

```

```

names = []

for name, model in models:

    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)

    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
                                 scoring='accuracy')

    results.append(cv_results)

    names.append(name)

    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

# Compare Algorithms

pyplot.boxplot(results, labels=names)

pyplot.title('Algorithm Comparison')

pyplot.show()

```

Step Six - Make Some Predictions

Next, we want to make some predictions and we can only do that by choosing the right algorithm to use. We got an idea in Step five that we Support Vector Machine (SVM) algorithm, is probably the most accurate one for this problem so this will be the algorithm we use for the final model.

So, now we need to get an idea of how accurate the model is on the validation dataset. This is a final, independent check on how accurate the best model is. It's always a good idea to split your dataset into training and validation sets, just in case something goes wrong during training, like a data leak or overfitting. If either of those happens, the results will be somewhat over-optimistic, and that won't give you an accurate representation.

So to make our predictions, the model needs to be fit onto the whole training dataset,

and the predictions are then made on the validation set. Here's how:

```
# Make predictions on the validation dataset  
model = SVC(gamma='auto')  
model.fit(X_train, Y_train)  
predictions = model.predict(X_validation)
```

Evaluate the Predictions

To evaluate our predictions, we need to compare them with the results expected from the validation dataset. Then the classification accuracy must be calculated, along with a classification report and confusion matrix.

```
# Evaluate predictions  
print(accuracy_score(Y_validation, predictions))  
print(confusion_matrix(Y_validation, predictions))  
print(classification_report(Y_validation, predictions))
```

As you can see, we have an accuracy of around 96% or 0.966 on our hold-out dataset and the confusion matrix gives us an indication of three errors.

Lastly, in the classification report, we can see each class broken down by recall, precision, the f-1 score, and support and, although we did only use a small validation dataset, we get excellent results.

0.9666666666666667

[[11 0 0]

[0 12 1]

[0 0 6]]

precision-recall f1-score support

Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.92	0.96	13
Iris-virginica	0.86	1.00	0.92	6

accuracy		0.97	30
macro avg	0.95	0.97	0.96
weighted avg	0.97	0.97	0.97

A Complete Example

And here it all is tied up together:

```
# make predictions
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC

# Load dataset
url="https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)

# Split-out validation dataset
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
test_size=0.20, random_state=1)

# Make predictions on the validation dataset
model = SVC(gamma='auto')
```

```
model.fit(X_train, Y_train)

predictions = model.predict(X_validation)

# Evaluate predictions

print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

Work through this tutorial, one step at a time and please, do not move on until you have got to grips with the step and all it entails. You don't have to understand it all, not right from the start. Your goal is to get a result, and if you run through this tutorial properly, that is what you will get. Write down any questions you may have as you work through.

Do use Python's built-in `help("FunctionName")` syntax to understand the functions in use.

You don't even need to know how those algorithms work right now, either. What you do need to understand is what their limitations are and how you should configure them; learning about them comes later on. Build up your knowledge of algorithms slowly and surely, starting by getting to grips with the platform.

Although I asked that you be comfortable with programming in Python, you do not need to be a programmer. Python language syntax is quite intuitive once you get the hang of it and, like any other computer language, you should focus, to start with, on assignments (`a = "b"`) or function calls (`function()`, for example; this will get you a long way. You can pick up the rest of the language through practice.

You also don't need to be an expert in machine learning. Indeed, that's not what this book is about. All I've done here is give you an introduction to machine learning, but it is an incredibly complex subject that requires real justice to be done to it. Again, you can learn more about algorithms at a later date; just get through this project and understand what it is all about.

Because this is your very first machine learning project, we didn't go over everything that you can do;

we needed to focus attention on the main steps – loading the data, looking at it, evaluating a few algorithms, and making predictions. Other steps, more complex ones, including preparing the data and improving your results.

In the next chapter, I will give you a brief introduction to data science with Python.

Chapter Six: An Introduction to Data Science

In the last few decades, the world has seen an explosion in big data, and that has led to a need for more efficient storage – a huge challenge to all businesses. The focus on making the best use of this data was to create frameworks capable of storing vast amounts of it, leading to Hadoop and other similar frameworks being built to help.

With the storage problem solved, businesses then moved their focus to ways of processing that stored data, and that is where data science enters the picture, a way of processing data and analyzing it. These days, data science is built-in to all businesses that process vast data amounts, and data scientists are hired to turn that data into something meaningful.

So, what is data science, and how is using it with Python so beneficial?

What is Data Science?

Let's start by understanding what data science is. As a simple explanation, it is about finding data and exploring it in the real world, and then using what we learned to solve problems. A couple of examples of how businesses use data science are:

- **Customer Prediction** – based on current and past customer behavior, a system can be trained to predict how likely they are to purchase a certain item
- **Service Planning** – restaurants can predict the number of customers visiting their establishment on the weekend and ensure they have sufficient food to handle the numbers

Before we delve deeper into data science with Python, let's talk a little more about why you should use Python.

Why Use Python?

Data science requires a computer programming language to work and, although you could choose to use SAS or R, Python is far superior and better-equipped to handle it. As a programming language, Python has grown in popularity and is now used, not just in data science, but in

artificial intelligence, IoT, and lots of other technologies, boosting its popularity even more.

We use Python for data science, primarily because it contains some of the best and costliest tools from the statistical and mathematical perspectives. Other reasons include:

- **Speed** – Python is much faster than other comparable computer programming languages
- **Availability** – it contains lots of different packages, developed by other users, that are easily reusable
- **Design Goal** – Python has intuitive syntax, easy to understand, that helps us to build applications containing an easily readable codebase.

You learned how to install Python in my last book, "Python Computer Programming - Simple Step-By-Step Introduction to the Python Object-Oriented Programming: Quick Start Guide for Beginners."

So, now, we can look at some of the Python libraries designed for data science.

Python Data Analysis Libraries

Python is one of the easiest languages to learn, and there is quite a lot of basic stuff, such as printing statements, that you can already do with it off the bat. But, if you want to use Python for data science, you need to use different libraries, which must be imported into Python. Some of those libraries are:

- **Pandas** – useful for data operations that need to be structured
- **NumPy** – an incredibly powerful Python library for creating n-dimensional arrays
- **SciPy** – gives us the scientific capabilities needed for data analysis, such as the Fourier transform and linear algebra
- **Matplotlib** – used mainly for data visualization
- **Scikit-learn** – used for all activities in machine learning

Those are the main libraries, but you may, at some point in your data science and machine learning journey, use these as well:

- TensorFlow
- Networks & I graph
- OS
- BeautifulSoup

Let's look at some of the primary libraries in more detail.

SciPy

The name kind of gives this one away, as a scientific library for Python, and it has a few special functions:

- Special functions, such as integration, gradient optimization, ordinary differential equation (ODE) solvers, and more are included
- It offers fully featured modules for linear algebra
- It is built on NumPy

NumPy

NumPy is the primary data science package for Python, containing:

- Tools to help you integrate Fortran, C, and C++ code
- Some of the most powerful N-dimensional array objects
- Useful capabilities for random numbers, Fourier transform and linear algebra

Pandas

Pandas is a useful library for the manipulation and operations on structured data. It is:

- The single most useful library in Python for data analysis

Instrumental in ensuring Python is widely used in data science

- Extensively used for data preparation and data munging

Let's delve a little deeper into using Pandas for exploratory analysis:

Using Pandas for Exploratory Analysis

This approach is used for analyzing large datasets, finding their main characteristics, and summarizing them. Visualization is used to help get valuable insights from the data.

Two of the most common terms you will hear in conjunction with Pandas are:

- Series – a one-dimensional object, capable of taking any type of data, including floats, integers, and strings

Here's an example:

```
import pandas as pd  
  
x = pd.Series([6, 3, 4, 6])
```

```
0      6  
1      3  
2      4  
3      6
```

dtype: int64

- DataFrame – a two-dimensional object whose columns can each have a different data type.

For example, the following is a DataFrame that has four rows and three columns:

```
import pandas as pd  
  
import numpy as np  
  
df = pd.DataFrame(np.random.randn(4, 3))  
  
df
```

	0	1	2
0	-0.568892	0.428537	2.125661
1	0.357808	-1.087910	-0.956698
2	0.06709	-0.232456	-1.006084
3	-1.822809	1.255754	0.441918

Next, we'll look at using Pandas for predicting if a loan application for a specific customer is likely to be approved or not.

1. First, import the libraries and then use the `read_csv()` function to read the dataset:

```
import pandas as pd  
import numpy as np  
import matplotlib as plt  
%matplotlib inline
```

```
df = pd.read_csv("../Downloads/loanP_train.csv") # using Pandas to read  
the dataset in the dataframe
```

2. Next, use the describe() function to look at a summary of the dataset:

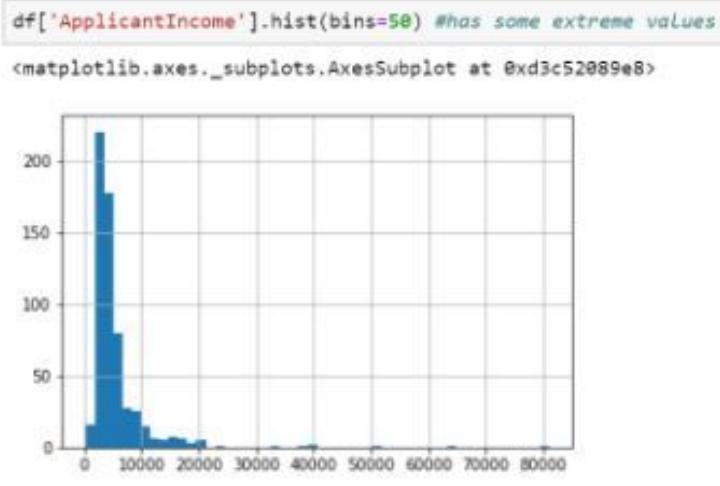
df.describe() #Look at summary					
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	599.000000	599.000000	578.000000	585.000000	550.00000
mean	5420.687813	1563.913055	145.434256	342.461538	0.84000
std	6169.168633	2456.618360	84.596463	64.868352	0.36694
min	150.000000	0.000000	9.000000	12.000000	0.00000
25%	2874.000000	0.000000	100.000000	360.000000	1.00000
50%	3812.000000	1210.000000	126.500000	360.000000	1.00000
75%	5807.500000	2279.000000	165.000000	360.000000	1.00000
max	81000.000000	33837.000000	700.000000	480.000000	1.00000

<https://www.simplilearn.com/>

3. The loan amount distribution can be visualized in this way:

<https://www.simplilearn.com/>

4. The applicant's income can be visualized in this way:



<https://www.simplilearn.com/>

5. And the categorical value distribution can be visualized

$$dist = Intercept + (\beta * speed)$$

$$\Rightarrow dist = -17.579 + 3.932 \cdot speed$$

Using Pandas for Data Wrangling

Data wrangling is the term used to describe the process of cleaning up and unifying complicated, somewhat messy datasets. These are the main benefits of data wrangling:

- It can tell you quite a bit about your data
- It helps you to make better business decisions
- Helps to gather precise data that has meaning to the business

In the real world, almost all of the data generated by a business is going to be messy, and there will be missing values. Take the loan dataset for example, some columns are missing some values.

To see if there are missing values in your data:

```
df.apply(lambda x: sum(x.isnull()),axis=0) # number of missing values in each column
```

Loan_ID	0
Gender	13
Marital_Status	3
Dependents	15
Graduate	0
Self_Employed	30
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	21
Loan_Amount_Term	14
Credit_History	49
Property_Area	0
Loan_Status	0
dtype:	int64

<https://www.simplilearn.com/>

We can fill in missing values in several different ways, and the business scenario will determine which parameters you use to fill them.

For example, you could use the mean of a specified column to replace the values:

```
df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)
#filling missing values
```

Loan_ID	0
Gender	13
Marital_Status	3
Dependents	15
Graduate	0
Self_Employed	30
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	14
Credit_History	49
Property_Area	0
Loan_Status	0
dtype:	int64


```
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean(), inplace=True)
df['Credit_History'].fillna(df['Credit_History'].mean(), inplace=True) #filling missing values
```

df.mean() #basic math operations ex df.median, cumulative summation etc	
ApplicantIncome	5420.687813
CoapplicantIncome	1563.913055
LoanAmount	145.434256
Loan_Amount_Term	342.461538
Credit_History	0.840000
dtype:	float64

<https://www.simplilearn.com/>

Or you could use dtypes to look at each column's data type:

df.dtypes	
Loan_ID	object
Gender	object
Marital_Status	object
Dependents	object
Graduate	object
Self_Employed	object
ApplicantIncome	int64
CoapplicantIncome	float64
LoanAmount	float64
Loan_Amount_Term	float64
Credit_History	float64
Property_Area	object
Loan_Status	object
dtype:	object

<https://www.simplilearn.com/>

You can even combine data frames and merge them using merge and concatenation methods.

It's time to build a model, and, for this, we'll use Scikit-learn.

Building the Model

Note that we have a mixture of written code and images in this section (courtesy of <https://www.simplilearn.com/>). Please treat them both the same; if you are following this and typing the commands in on Python, just type in what you see on the images.

1. First, we need several models imported from scikit-learn:

```
#Import the models from the scikit-learn module:
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import metrics
```

1. Next, the dependent and the independent variables need to be extracted from the dataset:

```
# Extract just the independent variables
```

```
X = df.iloc[:, [8, 10]].values # the credit history and amount of loan
```

```
# Extract just the dependent variables
```

```
y = df.iloc[:, 12].values #status of the loan
```

2. Now we can split the dataset into two – a training and a testing set. 75% goes to training and 25% for testing:

```
# Split the dataset into training and testing sets
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

We're going to build our model using logistic regression, the ideal algorithm for when we have a binary dependent variable.

3. We standardize the independent features from a fixed range in the data using feature scaling:

```
# feature scaling  
from sklearn.preprocessing import StandardScalar  
sc_X = StandardScalar()  
X_train = sc_X.fit_transform(X_train)  
X_test = sc_X.transform(X_test)
```

4. Then we fit the data to the model

```
# Fitting Logistics regression to training dataset  
from sklearn.linear_model import LogisticRegression  
classifier = LogisticRegression(random_state = 0)  
classifier.fit(X_train, y_train)  
  
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,  
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,  
penalty='l2', random_state=0, solver='liblinear', tol=0.0001,  
verbose=0, warm_start=False)
```

<https://www.simplilearn.com/>

5. Then the test set values are predicted

```

# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred

array(['Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y'],
      dtype=object)

```

<https://www.simplilearn.com/>

6. A confusion matrix is built to evaluate how the model performs

```

install.packages("caret")
library(caret)
install.packages("e1071")
library(e1071)

```

```

> confusionMatrix(iris_ran[101:150,], reference = model_pred)
Confusion Matrix and Statistics

          Reference
Prediction    setosa versicolor virginica
setosa         16        0        0
versicolor      0       13        2
virginica       0        2       17

Overall Statistics

    Accuracy : 0.92
    95% CI : (0.8077, 0.9778)
    No Information Rate: 0.38
    P-Value [Acc > NIR] : 1.678e-15

    Kappa : 0.8794
    Mcnemar's Test P-Value : NA

Statistics by Class:

           Class: setosa Class: versicolor Class: virginica
Sensitivity            1.00            0.8667            0.8947
Specificity             1.00            0.9429            0.9355
Pos Pred Value          1.00            0.8667            0.8947
Neg Pred Value          1.00            0.9429            0.9355
Prevalence              0.32            0.3000            0.3800
Detection Rate          0.32            0.2600            0.3400
Detection Prevalence    0.32            0.3000            0.3800
Balanced Accuracy       1.00            0.9048            0.9151
>

```

<https://www.simplilearn.com/>

How does a confusion matrix decide how accurate the model is? With the following calculation:

(True Positive (TP) + True Negative (TN)) / Total

(103+18)/150 = **0.80**

The precision is when yes is predicted and the number of times the predictions are correct:

True Positive / Predicted Yes = 103/130 = **0.79**

7. Lastly, we find the model accuracy

```
> actuals_preds <- data.frame(cbind(actuals=testData$dist, predicteds=distPred))
>
> correlation_accuracy <- cor(actuals_preds)
>
> head(actuals_preds)
  actuals predicteds
1       2     -5.392776
4      22      7.555787
8      26     20.504349
20     26     37.769100
26     54     42.085287
31     50     50.717663
>
```

<https://www.simplilearn.com/>

That completes your logistic regression model; you can use this in the future and build on it to produce different results.

How to Learn Python For Data Science

Before we dig into that subject, there's another question that needs answering – why you should even start with Python. Well, we answered that earlier, when we said that it is the most popular language for data science, although it never used to be. Kaggle is the leading data science competition platform, and, in 2016, Python stormed past R. In 2017, it carried on storming, passing R on the KD Nuggets annual poll asking data scientists which tools they used the most. And in 2018, two-thirds of data scientists said they used the language every day, pushing it firmly into the number one slot as the language for data science and analytics.

This trend is expected to continue as the Python ecosystem continues to develop and, even though this may be just the start of your machine

learning and data science journey, you'll be pleased to learn that there are plenty of employment opportunities.

The average salary for data scientists is just under \$130,000, and that is set to increase. According to IBM experts, a rise in the demand for data scientists is expected to see that figure rise by around 28% this year.

So, it's looking like a bright future for those who are serious about getting into data science, and Python is only one of the puzzle pieces. Fortunately, it is an easy part of the puzzle to learn, and I'm going to show you the five steps needed to learn how to be a data scientist.

Do keep one thing in mind – these may be simple steps, but you still have to put the effort and the work in. Apply yourself, dedicate the time needed, and you put yourself firmly in the running to take your career to a whole new level.

So, the future is bright for data science, and Python is just one piece of the proverbial pie. Fortunately, learning Python and other programming fundamentals is as attainable as ever. We'll show you how in five simple steps.

But remember – just because the steps are simple doesn't mean you won't have to put in the work. If you apply yourself and dedicate meaningful time to learning Python, you have the potential not only to pick up a new skill but potentially bring your career to a new level.

Step 1: Learn The Fundamentals of the Python Language

We all have to start somewhere, and this is the first step on the ladder to data science. Not only do you learn the basics of the Python language, but you also get an introduction to data science. One of the most important tools you need is the Jupyter Notebook, built-in to the Python libraries, and ideal to help you learn everything you need.

Kickstart your Learning

Join a community. When you do that, you put yourself with other like-minded people; not only does that give you a helping hand you need, you

also increase your opportunities for future employment. The Society for Human Resource Management says that around 30% of hires come via employee referrals. Create an account with Kaggle, take part in the Dataquest Slack discussions for members only, and join local Meetup groups in your area.

Related Skills

The command-line interface, or CLI, allows quicker running of scripts, which means you can test your programs quicker and works with a whole lot more data.

Step 2: Do Some Small Python Projects

Hands-on learning is absolutely the best way to learn; the sooner you start, the sooner you will be ready to start building your own Python projects.

Try small things first, perhaps a calculator for a game played online, or a program that pulls the weather data from Google for your area. When you build small projects like this, you learn Python much quicker. Small projects like these are pretty standard for most programming languages and are one of the best ways to ensure your learning.

Build up your experience using APIs and start doing some web scraping. As well as helping you learn how to program this helps you to understand what's involved in gathering data.

Kickstart your Learning

By reading blog posts, guidebooks, open-source code, anything, you can get your hands on. Pick up new ideas, cement your own learning, and see how other people do it.

Related Skills

So some work with SQL databases. SQL talks to databases, altering them, editing them, and reorganizing the data. It is one of the hard and fast data science staples, and more than 40% of data scientists use it regularly.

Step 3: Learn the Python Data Science Libraries

Not all programming languages provide you with a 'best way' to do something, but Python does, in the form of data science libraries. Three of the best are Matplotlib, NumPy and Pandas. The latter two are a fantastic way to explore data and play around with it, while Matplotlib helps you create visualizations of your data, in the form of graphs.

Kickstart your Learning

Ask questions. Lots of questions. Python is backed up by a huge community of developers and experts, and all are ready and willing to answer your questions and help you learn.

There are also plenty of other useful resources, like Dataquest's Slack, Quora, Stack Overflow, and many more, all filled with people just waiting to help you and share their own hard-earned knowledge.

Related Skills

Learn how to use Git for controlling code versions. It is one of the most popular tools to track your code changes; that means you can easily correct mistakes, do some experimenting, and collaborate with other like-minded people.

Step 4: As You Learn Python, Build Up a Data Science Portfolio

This is an absolute must for anyone who wants to become a data scientist. Many data science projects have multiple datasets and should give you lots of interesting insights. You don't need a theme of any sort for your portfolio, simply choose the datasets you have an interest in and work out ways of bringing them together.

If you display such projects, you are providing others with something they can collaborate on, and it shows any potential employer that you've put the time and effort in to learn Python and some other skills in programming.

Plus, your portfolio can act as a kind of resume, and it highlights all the skills you learned in your journey.

Kickstart your Learning

Communicate with others, collaborate, and keep your focus firmly on technical competence. Make sure you use the time to cultivate all the soft skills you need to work with other people and, importantly, make sure you understand how the tools you use work.

Related Skills

Learn beginner statistics and move on to intermediate. It's a good idea to get a decent background on statistics when you are learning Python for data science. It will give you the right mindset to ensure you focus on the right stuff, ensuring you will find real solutions and good insights instead of just repetitively executing scripts.

Step 5: Apply Some Advanced Techniques in Data Science

The last step is to sharpen your skill. Your entire journey is going to be packed with learning, but, once you have the basics down, look into taking an advanced course, just to make sure you covered everything. There are certain things you need to be comfortable with – classification, regression, k-means, and many other models. You can also look at learning how to

bootstrap a model or how to use scikit learn to build a neural network. At this point in your learning, you can take on projects like creating a model to use a live data feed, or something as complex as that.

REMEMBER

Keep on learning. The data science field never stops growing, and it covers hundreds of industries. Demand is increasing at such a rate that there are plenty of learning opportunities. Don't stop reading, don't stop collaborating and definitely don't stop talking to others. That way, you keep your hand in, and you keep up with an ever-changing field.

Chapter Seven – Ten Things You Should Know About Machine Learning

Machine learning is a complex subject, not one you can learn in five minutes. It takes dedication, effort, no small amount of passion for what you are doing, and a lot of hard work and time. It is incredibly rewarding, though, and if you move onto data science and further your learning more, not only are you getting to grips with the Python computer programming language, you are also giving yourself every chance to get into data science as a career.

To help you out, these are ten things that you must understand about machine learning.

- 1. Machine Learning is All About Learning From Data.** AI or artificial intelligence is a huge buzzword these days and, if there's one thing for sure, machine learning defiantly lives up to the hype.

You can use machine learning to solve an untold number of problems, so long as you put the correct training data to the correct machine learning algorithms, of course. Some people call machine learning AI and, if that's what makes you happy, go for it. Just be aware that AI is just a buzzword, albeit a huge one, and it can mean whatever you want it to mean.

- 2. Machine Learning is All About Data and Partly About Algorithms.** There is definitely an increase in excitement levels about the huge leaps and bounds being made in machine learning, specifically in the algorithms and deep learning. But the key thing to remember with machine learning is that it's all about data; that is the primary ingredient, and, without it, you simply can't do any machine learning. You don't need sophisticated or complex algorithms to learn, but you do need quality data.

- 3. Don't Use Complicated Models Unless You Have Vast Amounts of Data.** Machine learning finds patterns that exist in data and then uses them to train the machine learning models, using specified parameters. If you have too large a parameter space, your training

data will be over fitted, and the model you are training will not have the ability to generalize beyond that. In general, keep things a simple as you can, and that means matching your models to the level and complexity of data you are working with.

4. **Machine Learning is Only Ever as Good as the Data Used for Training.** Those who have experience with computer programming will already be aware of the phrase, "garbage in, garbage out." This is way older than machine learning, but it is a very apt phrase that describes one of the primary limitations machine learning has. A machine learning model will only find the patterns if they are in your training data. If the data is rubbish, don't expect your model to perform as you expect . With classification, and other supervised learning tasks, your data must be rich in features, and it must have the correct labels.
5. **Your Training Data Needs to be Representative if Machine Learning is to Work.** To draw a parallel to this, think about a fund prospectus; it will tell you that past performance cannot be used as a guarantee of results in the future. With machine learning, the unspoken warning is that it can only work if the data generated for the testing model comes from the same distribution used to generate the training model. You must be fully aware of any skews between production and training data and make sure your models don't get stale by retraining them on a regular basis.
6. **Data Transformation Represents the Largest Part of the Hard Work.** If you get sucked into the hype written about up and coming and brand-new techniques in machine learning, it's perfectly possible that you will come to see machine learning as being mostly about choosing algorithms and tuning them. The reality is somewhat different since the bulk of your time will actually go into cleansing the data and doing the feature engineering – taking the raw features and transforming them into features that represent your data signal much better.
7. **Deep Learning May be One of the Most Revolutionary Advances, But it isn't the "Magic Bullet."** Over time, deep

learning has undoubtedly lived up to and properly earned its hype; after all, It offers huge advantages to many different applications in machine learning. And it's being widely used to automate work that was once done in feature engineering, particularly where video and image data is concerned. However, it isn't a magic bullet; you can't just decide to use it, not without serious preparation, and a whole lot of time and effort in cleansing your data and transforming it.

8. **Machine Learning Systems Are Reliant on Operator Input.** And that makes them incredibly vulnerable to operator errors. When a machine learning system goes wrong, very rarely is that because the algorithm had some kind of problem. It's more than likely that human error has made its way into the training data. It could have caused bias or any other kind of error that causes incorrect results. Machine learning requires no small amount of skepticism and must be approached with the same discipline that is required with software engineering.
9. **Inadvertently, Machine Learning May End up Creating a Self-Fulfilling Prophecy.** In lots of machine learning applications, when you make a decision today, it could have a significant effect on the training data for tomorrow. Once bias has been embedded into your machine learning model, that model will just carry on generating the data that reinforces them, and that can create chaos. Be aware and be responsible.
10. **AI Isn't Going to Suddenly Rise and Destroy Humanity as we know it.** It's surprising how many people get all their artificial intelligence "information" from sci-fi books and movies. Yes, these should give us some inspiration, but we should never, ever mistake them for reality. We have more than enough real dangers to occupy us; we don't need to convince ourselves that the world will soon be run by AI robots!

Conclusion

This book is by no means a comprehensive guide, but I have gone over the most important features of machine learning. You now know the difference between supervised and unsupervised learning, what regression and classification are all about, and the best algorithms to use. I hope that you managed to work through your first machine learning project, too and that it shone a light on what the subject is all about.

It is a complex subject, and you should use this guide as a stepping stone to learning more. There are plenty of online courses and books on the subject that you can use to further your knowledge but, don't forget; the best way to learn is practice.

Machine learning is not something you can forget about and then hope to pick up where you left off. New algorithms are being created regularly and new projects and uses coming to light. Machine learning is the future, and learning it will put you in good stead and on a solid footing to succeed.

References

<https://towardsdatascience.com/>

<https://machinelearningmastery.com/>

<https://jakevdp.github.io/>

<https://realpython.com/>

<https://www.analyticsvidhya.com/>

<https://analyticsindiamag.com/>

<https://financetrain.com/>

<https://builtin.com/>

<https://medium.com/>

<https://www.simplilearn.com/>

<https://www.dataquest.io/>

Computer code attributed to GitHub is open-source, license free code.

Python Data Analysis

**Comprehensive Guide to Data Science, Analytics
and Metrics with Python.**

Alex Campbell

Introduction

People often consider data science to be one of the toughest and most challenging subjects of all time. Its name has more weight than its inner materials. The myth regarding data science makes most students stay away from it.

You would silly to believe in myths rather than verifying it all yourself. You are most likely to fall in love with the subject as it hides many surprises in its deepest folds. Data science does not require anything more than logical mathematic knowledge and little programming skills.

You might have to be aware of terms like discrete mathematics, linear algebra, calculus, vector algebra, etc. Your programming skills on Python, basic, R, and Tableau must be strong. But you can succeed! This book will help you learn the basic concepts and the various sectors in data science. Let's discover the deepest secrets of data science with Python.

Chapter 1: Basics of Data Science

Stepping into the world of data science will enlighten your path of knowledge. Pay close attention to the following chapter to understand the basics of Data Science.

What is Data Science?

If you are new to data science, then all the above sentences might seem like Hebrew. But in short, data science is nothing more complicated than scientists extracting data and information from big data and analyzing it. That data comes from multiple sources – search engines, social media sites, surveys, e-commerce sites and many more places.

Every day, our access to data increases. This is mostly down to how technology and techniques in collecting it have advanced and continue to advance. These days, just about anything can be monitored, including buying patterns, customer behavior, and so on, and many companies use this to make predictions about future purchases. These decisions can help them plan better.

However, as fast as this data grows, the more unstructured it becomes and in that format, data is no good and it must be parsed to get any real information from us. That is where data science came into the picture, using machine learning and big data to interpret the data, enabling effective decision-making.

The term, 'data science' has been around for around 30 years. In 1960 it was used instead of the term 'computer science' and, around 15 years later, it was used to define the methods available for processing data; in 2001, data science became an independent discipline.

So, how does data science work? It uses tools from various disciplines to gather in the data, process it, and gain insights from that data, extracting the meaningful data and using it to make the right decisions. It's that simple. Data science comprises several fields, including programming, machine learning, analytics, statistics and data mining.

With data mining, algorithms are applied to the data set, revealing patterns used to extract meaningful data. Predictive analytics or statistical measures use the data extracted to predict what might happen in the future based on what happened previously.

Machine learning is part of the artificial intelligence family. It is used for processing huge amounts of data, quantities that humans simply couldn't even comprehend, let alone process in their entire lifetime. When predictive analytics presents a model, machine learning takes it and makes it even better. It does this by matching event likelihoods to real events that happened at a predicted time.

Analytics is about the data analyst collecting the structured data from the machine learning model and processing it using algorithms. The data is interpreted, converted and summarized into a language easily understood by the decision-making team. Data science is applied to multiple contexts. As the role of the data scientist evolves, so the field will evolve with it, encompassing data engineering, data architecture, and data administration.

Until recently, implementing data science skills into practical life was next to impossible, but in the last decade, new upcoming scientists did their best to implement the statistical knowledge into practical life.

Scientists are still working day and night to enhance the concept of data science.

Career Scope and Impact of Data Science Using Python

Coming to the first part of the topic, the career scope in data science is more than most other job scopes globally. Data science is a field of knowledge that just keeps on growing, with no boundaries. People have considered data science in Python to be the best job for the past three years, but at present, its rank has gone down to third place. Data science is a wise career choice, not just for the salary but the many benefits it offers.

Although data science is relatively new, thanks to the positivity that surrounds it, it is now in much greater demand than ever before. There are few vacancies in the data science field but these are set to increase year on year as the amount of unstructured data grows and data science jobs are created to take care of it.

A career in data science just requires a little knowledge of quantitative studies and programming skills. This is what you have to do to open career scopes in data science:

- The efficiency with quantitative streams and programming language like that of Python widens your chances of making a data science career.
- You will have to complete the basic bachelor's degree from a university in one of the quantitative streams. Learning computing languages like Python during your college days is a requirement.

Python is one of the highest-level languages of all time. It is used to code and decodes other programs without much effort in no time. The higher the level of the computing language, the easier it is for the machine to understand.

- Some world scientists in data sciences even prefer completing their Ph.D. in one of the quantitative fields before getting into data sciences.
- The next best thing you can do is get into an essential career-making line in data sciences. The value of job experience in data sciences is excellent. So you can quickly gain some knowledge and some experience from the entry-level job, which will help apply for high-level jobs in data sciences.

By this time, you already know how to get started on a career in data science. Here is a list of positions in data science jobs that might be useful to you.

- **Machine learning scientists:** one of the most critical career options in data analysis or data science is machine learning. As you already know, machine level language and information continuously modify and a highly skilled machine learning scientist helps by discovering new data and creating algorithms to get new ideas easily.
- **Data Engineer:** you already know that the engineers are the gods of creation. They build for the people. Similarly, data engineers give an appropriate structure to the machine learning scientist's raw data. Later on, the software engineer uses the data frame from the data engineer and completes the package.

- **Data Analyst:** data analysts also play a pivotal role in the respective companies. They analyze the data from external sources and filter out the only necessary information for the company's needs.
- **Data Consultant:** the data consultants and data analysts are dependent on each other. The data consultant looks into the data analyst's information and uses it to make the best possible business decisions.
- **Data Architect:** to bring out the best from a company, the data architects apply useful data and information from the raw source and optimize them to apply in the best way possible for the company.
- **Applications Architects:** these people track down the various information used in the company. The higher authorities decide on how to deal with the information based on the application architects' report.

The impact of the above aspects and positions equally matters as none of them will work well without one another. The usable information from data scientists helps bring out the best of a company, while the algorithms and new software from the data engineers work in new ways and open up new career-making aspects. With such valuable information, you can do a lot more with the perks of data sciences. Let's move on to the next chapter!

Chapter 2: Various Aspects of Data Science

In this chapter, you will learn to utilize the knowledge from the previous chapter. The various aspects and more in-depth knowledge from the basics of data science will help you understand it more and in a more appropriate way. Take a look!

Steps Involved in a Data Science Project

Previously you learned what data science is and the basic concept and ideas related to it. The next step is to learn how to utilize that information and how you can apply it in working with data science. The following steps on the data science project will help you to understand.

Defining the Problem

Here, the problem is referred to as the data you are seeking. The first step involved in data science or cracking a data code is understanding and defining the type of data you are dealing with. If you fail to understand the first step, you will get the whole process wrong. It is one of the most critical points where the data scientists have to show their skills. After understanding the problem once, you will be able to look for the exact data required.

Collecting Data from Sources

Defining the problem will take you to the very next step of the data science project. It involves collecting the required data from a sensitive platform to increase your company's standing and reputation.

Millions of data are available in a data platform; the data scientists apply their brains and outstanding knowledge to find the right data to make the right decisions for the company.

Data Processing

The third most important thing you have to do in the data science project is to process the data. Data processing involves the data analysts and data consultants going through the source data thoroughly to make sure there are no errors that can cause trouble for the company.

Feature Engineering

The first step to forming any real meaning from data in data science is feature engineering. In feature engineering, the data engineers take a step forward to look into the data that came from the data consultant and the data analyst. They sensitively observe the data and attempt to give it a shape or simple frame. Data engineers play a vital role in the process. As you know, if the base of a building is not strong enough, it would easily fall with the slightest pressure. Similarly, the data engineer's work has to be solid so the software engineer can do their job properly.

Algorithm Selection

The most crucial section in any data science project is the algorithm selection. After the data engineer has passed on the data frame, it is the data architect's turn to go through the information and data frame. He or she has to ensure all the data is sorted correctly and divided into the right sections.

Once this is done, the right algorithms can be selected, choosing the most important and useful data for the company and rejecting the rest.

Hyperparameter Tuning

In this step, the information from the data architect is verified once again. Data science is not an easy project, so all the data has to be right. To achieve this, the date must be filtered several times, through a process of hyperparameter tuning and, when the data finally only has very low-level errors, if any at all, it is ready for the next step.

Data Visualization

The second last step in the project of data science is data visualization, which is where the application architect steps in. They will evaluate the result of all the previous steps and determine the exact location where the data will be used. The application architect visualizes the final data frame and fills in the data.

Interpretation of Results

Moving to the final step of the data science project, the company will judge if the information and data are correct and have been useful depending on the problem the company faces. Using the company's outcome wherever required, the higher authorities try to interpret the final result.

How to Solve the Problems with Python

Python helps in solving the problems to some great extent and is the best computer programming language to use with data science.

The first thing you must consider is your knowledge of the Python programming language. If you want to work in data science, then you must have experience with Python and know at least the fundamentals of coding with it. You must learn how to identify which libraries to import and how to run exploratory data analysis on the data to understand better the problem you are trying to solve.

Python can help you identify missing data and how to fill in those gaps, perhaps using an average of the available data. Once that has been done, you can verify the results with some simple Python coding; if the method works, you can move onto the next step. If it doesn't, the only thing to do is start over.

The final step is the data cleaning with Python. This process helps you find the best data with the least requirements and lets you cut out data that doesn't affect the result.

Chapter 3: Python Exploratory Analysis with Pandas

For further exploitation of the data and a smoother understanding, let us now take into consideration a popular Python library: Pandas!

Pandas is the most important and widely used library that you will find in Python. It has made the use and implementation of Python more diverse and easier to use for data science. We're going to use it later to read a data set, analyze it and build our first categorization algorithm. It won't be complex, but it does give you a good idea of how it all works.

Before we start working with the data at all, we must understand the two main data structures that are included in Pandas, which are DataFrames and Series.

Understanding DataFrames and Series

We can understand Series as a one-dimensional indexed array or a one-dimensional labeled array. With these labels, it becomes effortless for you to access each of these elements in the complete array. A DataFrame, on the other hand, is very similar to an Excel workbook. Here you will see column names and row numbers that let you easily access the specific data you want. In terms of the DataFrame, the numbers are names are termed as row and column indexes. Basically, DataFrames and Series form the basic data model in Pandas that can be used for Python programming. All data sets get integrated first and read on these data frames and then forwarded to other operations, for example, group or aggregation, which can very quickly be used from the columns themselves.

Data Set For Practice – A Loan Prediction Problem

You'll be able to find the data set [here](#).

<https://datahack.analyticsvidhya.com/contest/practice-problem-loan-prediction-iii/>

The following description of the various variables:

Description of the Variables

- Loan_ID: It is used to denote the applicant's loan id details. It is unique and different for every customer.
- Gender: Either male or female.
- Married: It denotes if the applicant is single or married. (Y means married, N means single).
- Dependents: It denotes the total number of family persons dependent on the primary loan holder.
- Education: It denotes the education qualifications of the applicant. These can be a Graduate, Post Graduate, or Ph.D. holder.
- Self_Employed: It denotes if the applicant is a salaried person or having his own business. (Y means own business and N means salaried person). ApplicantIncome: It denotes the yearly income of the loan applicant.
- Co-applicantIncome: If there is a co-applicant too, it denotes his yearly income.
- LoanAmount: It denotes the total loan amount requested by the applicant.
- Loan_Amount_Term: It denotes the duration of the loan in terms of months.
- Credit_History: It denotes if the applicant has taken any previous loans.
- Property_Area: It denotes the residential area in which the user lives.
- Loan_Status: It denotes if the loan application is approved or not. Y means approved, N means not approved.

Beginning with Exploration

Start the iPython interface in the Inline Pylab mode by typing the following in your Windows terminal command prompt:

ipython notebook --pylab=inline

The command will help you to open the iPython notebook within the Pylab environment. This environment already has some important libraries pre-imported and the following location is where the data is being stored.

/home/mary/project/Loan_Prediction/train.csv

How to Import the Data Set and Libraries

The libraries that will be used are as follows:

- pandas
- matplotlib
- NumPy

One thing that has to be noted here is that NumPy and matplotlib don't need to be imported because they are already included in the Pylab environment. However, we are still showing you the code needed to import them should you need them for any other project.

After you import the library, now you will have to start reading the dataset. This has to be done by using the function `read_csv()`. The following is the code to import the libraries and read the dataset:

```
import matplotlib as plt  
import numpy as np  
import pandas as PD  
%matplotlib inline  
df = pd.read_csv("/home/mary/project/Loan_Prediction/train.csv") # This is  
how to read the dataset with the help of pandas library
```

Quick Data Exploration

Once you have taken the data set into consideration, you will now be able to read some of the top rows with the help of the function `head()`

df.head(15)

This command will print 15 rows. But you can request to see more just by changing the number in the brackets.

After this, you can look at the summary of numerical fields with the describe() method.

df.describe()

The describe() function is very important as it will provide you with mean, quartile, count, min, max, standard deviations (std), as well as their outputs. This will help you to understand the necessary statistics to understand population distribution.

The following are a few inferences that can quickly be drawn by examining the output at the describe() function:

1. There are 50 missing values in the Credit_History.
2. There are 14 missing values in the Loan_Amount_Term.
3. There are 22 missing values in the LoanAmount.

Also note that you can understand possible skews in the data by comparing the mean with the median, which is 50% of the total figure.

For the various non-numerical values which include Credit_History, Property_Area, and more, the frequency distribution can be taken into consideration to see whether it makes sense at all. Print the frequency table using the below command:

```
df['Property_Area'].value_counts()
```

Similarly, we can also check the unique values related to the applicant's credit history. You need to understand that the code `dfname['column_name']` is a very preliminary indexing technique used to access a particular column present in the data frame. You can also use it for conjuring lists of columns.

Distribution Analysis

Since we have familiarized ourselves with the necessary data characteristics, we can understand the various distributions of the multiple variables. It is best to start with the numeric variables, which include `LoanAmount` and `ApplicantIncome`.

We can start this process by plotting a histogram of the `ApplicantIncome`, which can be done with the following command:

```
df['ApplicantIncome'].hist(bins = 100)
```

From the histogram, it can be observed that there are some extreme values present in the data. This is the reason why 100 bins are required to represent the data accurately.

Next, we can further analyze the box plots that will help us to understand the distribution. The box plot for fares is plotted with the help of the following command:

```
df.boxplot(column = 'ApplicantIncome')
```

This will help us to understand the presence of extreme values, and we can understand the disparity through the differences in income throughout society. Some of this can be attributed to how we can examine the different educational levels of the applicants too. For better analysis, we should segregate them based on educational qualifications. This can be done with the help of the following command:

```
df.boxplot(column = 'ApplicantIncome', by = 'Education')
```

Now we can further see that there is not much discrepancy between the mean of income between graduates and non-graduates. However, there are

more people in the high-income group in the graduates segregation and these form the outlier cases. Next, we can see the box plot as well as the histogram for the loan amount with the help of the following commands:

```
df['LoanAmount'].hist(bins = 100)  
df.boxplot(column = 'LoanAmount')
```

We can again see some extreme values, which only indicate that both the loan amount and applicant income require some data munging. We can also see that the loan amount section has some missing data and influential figures, while the presence of powerful statistics can also be noted for the applicant income section. This necessitates the further analysis of the data so we can understand it better.

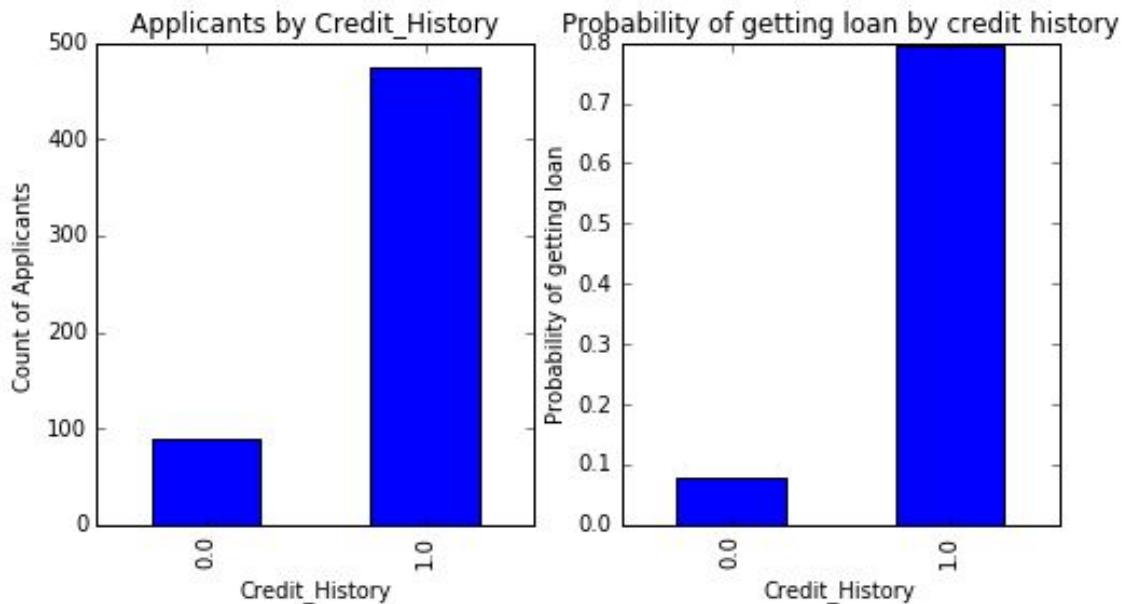
Categorical Variable Analysis

Once we have understood the distribution of data in the Applicant Income and the Loan Amount section, we can now perform categorical variable analysis, which will help us to understand the data even better.

For this step, we will be using cross-tabulation as well as an Excel-style pivot table. For instance, let us first consider the probability of getting a loan based on credit history. This can be easily done on MS Excel by using a pivot table.

It is important to note that 1 has been quoted as Yes for the loan status and 0 represents no in terms of the probability of getting a loan based on credit history.

Now we will have to look at the steps required to get similar insights by using Python. You can read online articles that will help you to get the hang of the various data manipulation techniques that are used in Pandas. What we do here will simply help us to get a pivot table like the one we got using Excel. This can also be plotted as a bar graph by using the matplotlib library.



The bar graph and shows that a person with a better credit history is 8 times more likely to get a loan. Similar graphs can be plotted, taking into consideration other variables such as self-employed, married, etc.

On the other hand, these two can also be plotted in a manner that will help you to visualize them in a combined form by using a stack chart. In the same way, we can also add gender just like it was present in the Excel sheet.

If you have not understood yet, we have actually created two classification algorithms, one based on credit history and the other is two categorical variables that include gender.

By using the given codes and techniques, you will be able to use Pandas for exploratory analysis in Python. By now you should be able to see useful Pandas is, keeping in mind how simple it makes various operations and the fact that it is so easy to use and generate. This library can help you to analyze the data with ease.

After this, you can further analyze the two variables, LoanStatus and ApplicantIncome by performing data munging. Once you perform data munging on the data set, you will create a data set that can be used for applying various modeling techniques. To help you understand better, find examples on the internet that use these steps.

Pandas is a powerful and simple data analysis framework for Python. It's tightly incorporated with NumPy and matplotlib packages. In this section, we will show you some examples to demonstrate its power.

DataFrame

It's a 2-dimensional data structure with columns of different types, similar to an excel spreadsheet or SQL table.

When using pandas, we must import the following:

```
import pandas as pd
```

Creating a DataFrame

```
import pandas as pddf = pd.DataFrame({'A' : [21, 22, 23, 42],  
                                     'B' : [24, 23, 22, 13],  
                                     'C' : [67, 14, 12, 18],  
                                     'D' : [24, 53, 52, 41]})  
  
print df
```

The below outcome shows the DataFrame creation with the different columns.

	A	B	C	D
0	21	24	67	24
1	22	23	14	53
2	23	22	12	52
3	42	13	18	41

Column Selection, Addition, Deletion

Users can select, add, and delete the columns in DataFrame like an SQL table operation.

Selection: We can select specific or all columns from DataFrame as shown below.

Example - Selection: import pandas as pddf = pd.DataFrame({'A' : [21, 22, 23, 42],

```
'B' : [24, 23, 22, 13],  
'C' : [67, 14, 12, 18],  
'D' : [24, 53, 52, 41]})print df['A'] Output: 0    21  
1    22  
2    23  
3    42
```

Name: A, dtype: int64 **Addition:**

We can add a specific column to DataFrame as shown below.

Example - Addition: import pandas as pddf = pd.DataFrame({'A' : [21, 22, 23, 42],

```
'B' : [24, 23, 22, 13],  
'C' : [67, 14, 12, 18],  
'D' : [24, 53, 52, 41]})df['E'] = df['A'] * df['B']
```

print df *Output:*

	A	B	C	D	E
0	21	24	67	24	504
1	22	23	14	53	506
2	23	22	12	52	506
3	42	13	18	41	546

Deletion

Columns can be deleted or popped from DataFrame.

Example:

```
import pandas as pddf = pd.DataFrame({'A' : [21, 22, 23, 42],  
'B' : [24, 23, 22, 13],  
'C' : [67, 14, 12, 18],  
'D' : [24, 53, 52, 41]})# Delete the column
```

```
del df['A']# Pop the column  
df.pop('B')print df Output:
```

	C	D
0	67	24
1	14	53
2	12	52
3	18	41

DataFrame - Insert Column

It's used to insert data at a specific location in the columns. By default, the columns get inserted at the end.

Example:

```
import pandas as pddf = pd.DataFrame({'A' : [21, 22, 23, 42],  
'B' : [24, 23, 22, 13],'C' : [67, 14, 12, 18],  
'D' : [24, 53, 52, 41]})df.insert(1, 'E', df['A'])print df Output:
```

	A	E	B	C	D
0	21	21	24	67	24
1	22	22	23	14	53
2	23	23	22	12	52
3	42	42	13	18	41

DataFrame - Indexing, Selection

Row selection returns a ***Series*** result whose index is the columns of the DataFrame.

Example: import pandas as pddf = pd.DataFrame({'A' : [21, 22, 23, 42],

```
'B' : [24, 23, 22, 13],  
'C' : [67, 14, 12, 18],  
'D' : [24, 53, 52, 41]})
```

```
print "\nActual DataFrame: \n", df  
print "\nZero(0) Location Index Result: \n", df.loc[0]  
print "\nZero(0) Location Index Result: \n", df.iloc[0]
```

Output:

Actual DataFrame:

	A	B	C	D
0	21	24	67	24
1	22	23	14	53
2	23	22	12	52
3	42	13	18	41

Zero (0) Location Index Result:

A 21
B 24
C 67

D 24

Zero (0) Location Index Result:
A 21
B 24
C 67

Using Pandas for Data Munging in Python

If you have lost track of where the last lesson went, the following recap will help you get back on track. When the data was being explored, it was found that there were problems with the data set. These problems need to be solved before we could use it as a good model. This process is known as data munging. The following were the troubles that were encountered:

- Some values are missing on some variables. We need to estimate the cost of these missing numbers based on the variable's position and importance.
- Although it makes intuitive sense to have extreme values on both the variables, they need to be treated for better distribution.

Along with these problems present in the numerical field, we must also consider the difficulties in the non-numerical areas, including Married, Property_Area, gender, and more, to examine usage information.

Before you start working on this problem, you should find information on the internet discussing Pandas' data manipulation techniques.

Checking for Missing Values

First and foremost, it is important to look for missing values in the variable as most programs do not run without all the values, and even if they do, having all the values yields better results.

This command provides the number of values that are missing in each column. Even if the number of missing values is not very high, it should still be considered as it may be spread through many variables. These can be estimated and added to the data. You can learn imputation techniques from various sources on the internet.

Filling Missing Values in LoanAmount

There are many ways in which you can provide missing values for the loan amount. The easiest way to do so is by replacing them with the mean.

The other method includes building a supervised model for learning that will predict the loan amount based on the value of other variables. This can then be used with other variables as well.

The primary purpose at the moment is to bring about the various steps in data munging. The most rational method that lies in between these two extremes, whether a person is self-employed or educated, is combining them to give us an estimate of the loan amount.

This can quickly be done with the help of a box plot to check for the validity of the trend. When we find some variation in the value of the median loan amount in each group's case, this can be used to impute the value.

Following a few more steps, we can now create a pivot table that will provide all the median values for each group. This helps us to understand the values of the Education and Self-employed features. We can then define the function that will ultimately help us to return the values on the cells and help us to apply it to the missing value in the loan amount.

Treating Extreme Values in a Distribution

First, let us analyze the Loan Amount category as it is possible for people to apply for higher loan amounts depending on their needs. So we should treat them as log transformation that will help us to nullify their effects rather than using extreme cases.

This will help us to make the distribution a lot more familiar and curb the effects of the extreme values to a considerable extent.

Now consider the Applicant Income category. One action that can be made is that some applicants may have a low income but strong support from a co-applicant. So both the incomes can be combined and then you can perform log transformation on it.

Doing this, we will be able to make the distribution much better and balanced than previously.

Chapter 4: Basics of Python for Data Analysis

Why We Learn Python for Data Science

Recently Python became the number one programming language for most data scientists and there are several reasons behind its popularity:

- It is a versatile platform
- It is easy to learn
- It is open-source so anyone can download it
- In data science, Python is becoming the most common language

Why Do We Use Python v.3 And Not V.2 |?

Python V.2

- This was first introduced in 2000 and remained until it was updated in 2015. This version of Python is fantastic for beginners.
- Today you may have found many third-party libraries in Python V.3, but there are still some modules that only work in Python V.2.
- There are some features in Python V.3 which have backward compatibility and work fine in Python V.2.

Python V.3

- Python V.3 is much cleaner than Python V.2, which makes it more user-friendly.

Most of the previous versions' glitches and bugs are fixed here, which is why the coder will be able to build a strong foundation for the future.

- It's undeniable that Python V.3 is the future. It is a human tendency to leave previous things behind and move forward with new versions. Since the last release of Python V.2, most data scientists now use the 3x versions of Python.

There is no clear answer to the above question, but shifting between the programming language versions is just a factor of time. What matters is learning the language; other than that, you can choose any version you want.

Data Structures of Python

- **Lists-** This one is the most versatile data structure of Python. Lists can be easily denoted with any list written within the square brackets and each item separated with a comma. It can contain different types of content.
- **Tuples-** Tuples defines multiple values in a comma-separated list. Lists are mutable (which means they can be changed), but tuples are not, which is why in processing, tuples are faster.
- **Strings-** In Python, quoted statements are strings. These can be easily defined with single, double, or triple inverted commas. \ is used as an escape character of the string.
- **Dictionary-** Dictionary is a bunch of unordered key:value pairs. Within one dictionary, every key pair is unique and can be used as needed. The pair of empty braces denote an empty dictionary.

Data Analysis in Python Using Pandas

There are several useful libraries stored in Python and the process of importing them is straightforward. NumPy, Matplotlib, SciPy, Scikit Learn, Statsmodels, Pandas, Seaborn, Blaze, Bokeh, SymPy, Scrapy, and Requests are some of the libraries that are quite famous among data scientists.

Pandas is the most useful ones among the others. Among many other reasons, Pandas is one reason for the increased use of Python in data science. This library provides extensive means of data analysis in Python. While working with Pandas, load, process, etc., tasks become very convenient. One of the Pandas high performances is the blackened source code, which is solely written in Python or C.

Another advantage of using Pandas is, it has several methods for data filtering. The list of Pandas' utilities is also quite extensive so that the input or output operations can be easily performed.

In short, Pandas is a powerful, fast, flexible, and user-friendly data analysis and manipulation tool. And the most crucial part is, it is built on the Python Programming language.

Data Science Using Python: Start Instantly

Data science uses methods that provide some processes and techniques to summarize knowledge from raw data and solve problems. The algorithms provided by Data science are significantly beneficial to increase company profit. The companies use most of the data to gain insights into business and enhance their processing ability.

With the help of the following tutorial, you will learn coding in Python. The coding will work as a strong foundation to help you learn more languages in the future.

Read the Tutorial Carefully

Anaconda

By installing the Anaconda distribution, you will also get significant libraries like Python, Jupyter Notebook, etc. While downloading Anaconda, just keep it in mind that you will need Python3.

Jupyter Notebook

Jupyter notebook is a lightweight, integrated development environment that is used by most Data Scientists. You can also call it an advanced text editor.

When you downloaded Anaconda, the Jupyter notebook was downloaded automatically. To open it, you can either right click the command Jupyter notebook in the Command Prompt, or launch it from Anaconda's Navigator application.

Open New Notebook

First, create a folder where you want to save your notebook. Remember, all of your datasets must be kept in that single folder for your benefit during any coding.

When your Jupyter notebook has opened, navigate to the "New" button at the top right corner. Click on it, and a new notebook will be opened. Now you will see a blank canvas in your default web browser.

Math Calculations

It's time to write some codes. You can start coding with some math calculations. One thing that is extremely good about Python is, it is a versatile platform that can easily be used as a math calculator.

To activate the code cell, click inside the cell in front of you and press Shift+Enter.

- Start by importing math module
- You must know if your math functions all math constants, like math.sqrt() or math.pi, calculate specific python problems.
- To get answers to questions like three to the power five, write `3**5`.
- Write texts or comments followed by a hashtag (#)
- To print more than one calculation in one output, use the `print(...)` function.
- Store things in objects (variables)

Write a message surrounded by quote marks to display any sentence. There are many more critical points about Python, which you will learn as your coding days will pass. Additionally, you will find bundles of codes stored into libraries that you can import as per your requirement.

Data Importing

Anaconda contains several libraries from where you can easily import specific coding, which is why, at first, we recommended installing Anaconda.

In the new cell write,

```
import matplotlib.pyplot as plt  
import pandas As pd  
%matplotlib inline  
from sklearn.linear_model import LinearRegression
```

- Here we chose to import the panda's library, followed by the alias of pd. With the help of this pd, the library can be evoked.
- Then you can see we used the import command again to import matplotlib.pyplot. Here the pyplot module was chosen from the matplotlib library. The Matplotlib library plays a massive role in Python; this is the prime plotting library here.

Now in this tutorial, we have only selected a part of the library with the plt module.

- Now you must have lots of confusion with the %matplotlib inline command. It's nothing but a specified command for the Jupyter notebook, giving the notebook command to display our plots. Another reason behind this command is, it shows the plots inside the notebook rather than displaying it in another window.
- Next, you can see, the basic linear regression algorithm is imported from the scikit-learn. Scikit-learn contains a lot of algorithms from which you can easily choose one.

You will find lots of excellent libraries here that are quite famous and beneficial in data science.

Importing Dataset

With Python, you can work with databases like SQL, CSV, and Excel. Here we are going to give you an example of an excel dataset file.

First, download a dataset and save it to the same folder where you saved the Anaconda distribution first.

When your downloading has completed, write the code followed by an object named df (short form of dataframe). The code is,

```
df = pd.read_excel('mydata.xlsx')
```

Now your data set has been imported. To take a sneak peek of what's inside the database, simply run the code.

```
df.head()
```

You will be able to see five observations from the dataframe. If you feel like practicing more and more with datasets, then you can easily download as many as you want.

Exploration

Now it's time to explore the plots. But first, we are not going through the whole exploratory analysis phase. We will just run our eyes over the distributions of our variables.

Here we have started with the X1 variable. Write plt.hist(df.X1), and it will show you a histogram.

Generally, functions like plt.hist() have several parameters that you can pass to them. With these parameters, you can control things like the color scheme, the axes, the number of bins, etc. You don't have to remember these parameters because you can easily predict things and complete the coding automatically as you gain experience.

Clean the Dataset

Though the datasets you will download are very clean, you may still find some missing values. You can check with just a little coding, type:

```
df.isnull().sum()
```

In detail:

- df is the short form of dataframe and this object stores data in Python
- .isnull() is called a function in Python. This function searches cells with missing values, and if there are any missing values, it shows True.
- .sum() adds up the missing values together. With this method, you can generally sum up any numbers.

Features

Now we are going to learn about engineer features. Most data scientists are quite in love with this part because they can apply their domain knowledge and create completely new features for input or variables.

Before going into complicated variables, let's see how dummy variables are created here.

You can choose to create either numerical variables or categorical. If you take education as an example, then numerical refers to the numbers of academic years, and categorical refers to primary school, high school or college or undergraduate, postgraduate, etc.

If you talk about Glazing Area Distribution and Orientation, they are categorical.

```
2 == 'north',  
3 == 'east',  
4 == 'south',  
5 == 'west'
```

The above encoding is basically four integers.

To simplify the whole thing, you can create temporary dummy variables. That's how you can directly refer to the four separate classes of X6. Write:

```
df = pd.get_dummies( df, columns = ['X6', 'X8'] )
```

Run the code and repeat the same with X8.

Develop an Easy Model

Yeah, you read right. You are all caught up and ready to make the model. But always keep it in mind that this tutorial is written in the simplest possible way. So you can use it to train yourself and practice. Solve simple problems with this coding and start your journey of learning coding and training models.

First, split the dataset into two separate objects. Just predict simple things with the model in the target variable.

Separate the input features and target variables in Python.

```
# Target variable
```

```
y = df.Y1
```

```
# Input features
```

```
X = df.drop( ['Y1', 'Y2' ], axis = 1
```

- .drop() is a dataframe method.

Now you are all set, let's get a model ready.

At first, compute the model instance. If you aim to train another model and also compare the other one with this, you can get the job done with a separate instance;

```
model_2 = LinearRegression()
```

After that, use a fit () method, write input features and the target variable and pass the features as parameters.

This is most of the part of the basics you need to know before creating a model. And now you can call yourself a data scientist. In the end, we can confidently say that the above tutorial is a great place to start.

Using Matplotlib

One of the most popular uses of Python is for data analysis. Data scientists want to visualize data to convey their analysis results.

Matplotlib is one of the most popular data visualization modules and yet very simple to implement visualization.

We will cover three basic plots here – Line, Bar chart, and Scatter plot.

Installation and Getting Started

To get Matplotlib, you can download the version compatible with your Python from Matplotlib.org. The easiest way to install Matplotlib is by installing it through a package like Anaconda Distribution (a platform for data analysis and scientific computing).

You can find more information on the below link.

https://matplotlib.org/faq/installing_faq.html

We can code our program in different IDEs like Jupyter, Spyder which comes with the Anaconda package.

We can use matplotlib with other libraries like NumPy, pandas, SKlearn for data analysis and visualization purposes.

Note: All the coding (screenshots attached) is performed in Jupyter Notebook using Python 3 kernel .

Import pandas, matplotlib, and NumPy libraries

Pyplot is a sub-library of matplotlib and is used to plot the data.

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

With the code above, we just imported pyplot from matplotlib, to plot the data.

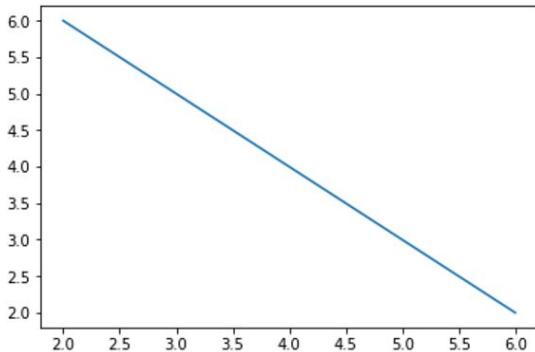
Line Graph

We need data points for plotting. Let's take a simple one

```
x=[2,4,6]
y=[6,4,2]
plt.plot(x,y)
plt.show()
```

plot takes various variables. As of now, for plotting purposes only, we have taken x and y.

show method displays the plot like below.

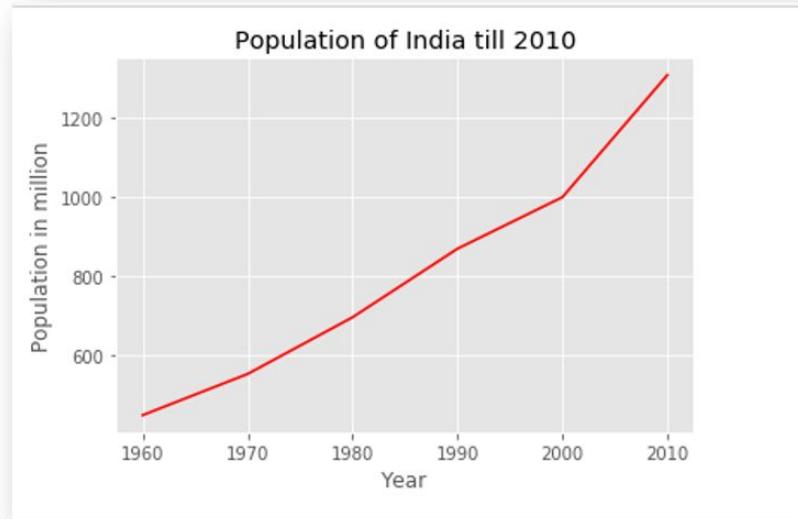


The above graph is very basic and does not provide any useful information.

Let's see at one more example where we can decorate the plot.

```
style.use('ggplot')
year = [1960, 1970, 1980, 1990, 2000, 2010]
population = [449.48, 553.57, 696.783, 870.133, 1000.4, 1309.1]
plt.plot(year,population,color='red')
plt.xlabel('Year')
plt.ylabel('Population in million')
plt.title('Population of India till 2010')
plt.show()
```

After running the above code, we get a plot like below.



The style package adds support for easy-to-switch plotting "styles". Here we have used the ggplot style. You can import the style package from the matplotlib library as below.

```
import matplotlib.style as style
```

The color of the line is set to red using the color parameter.

The X and Y axis are labeled as 'Year' and 'Population in millions' respectively using `xlabel` and `ylabel`.

The title of the plot is set using `plt.title`

Plotting Multiple Lines in the Same Graph

Multiple lines can be plotted in a single graph as shown below.

The point to note here is, we have called `plt.show()` at the end. If one wants to plot multiple lines in a single graph, you have to call `plt.show()` only once. We can specify which line represents what type of data by using the `label` parameter of `plt.plot()` and using `plt.legend()` after `plt.plot()`

```

x1=[3,5,9,2]
y1=[1,9,4,7]

x2=[6,3,7,2]
y2=[9,8,2,1]

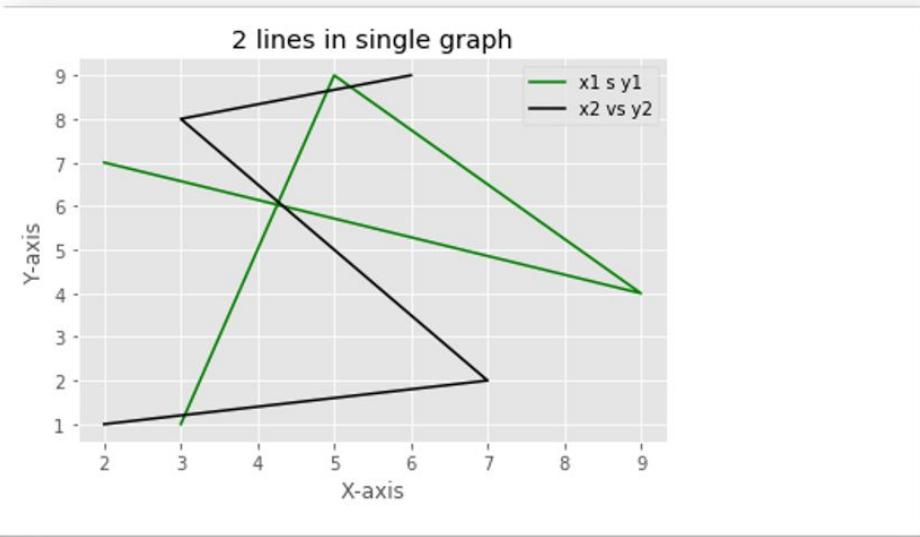
plt.plot(x1,y1,color='Green',label='x1 vs y1')
plt.legend()
plt.plot(x2,y2,color='Black',label='x2 vs y2')
plt.legend()

plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('2 lines in single graph')
plt.show()

```

After

running the code, we get a plot like:



Bar Chart

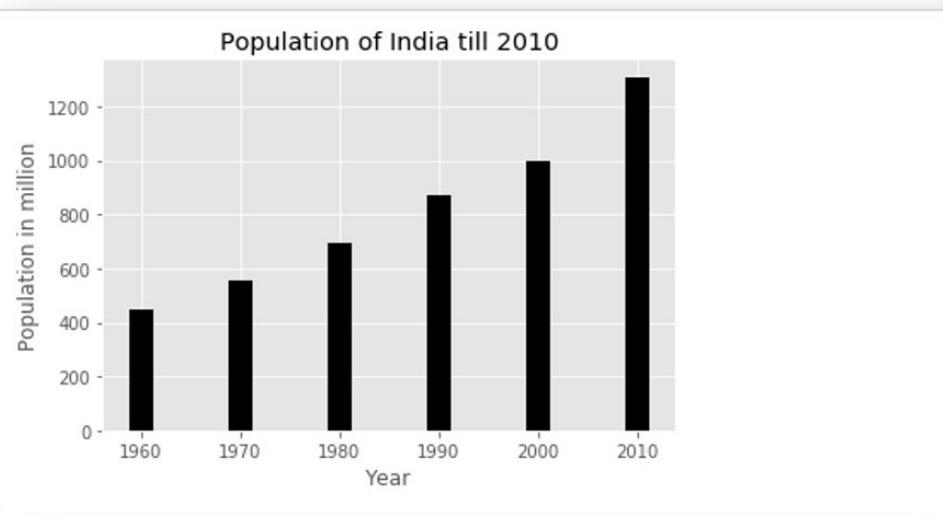
Bar charts are useful for visualization of counts or summary statistics. We will use the same example of the population of India used in the bar chart.

```

bar_width= 2.5
year = [1960, 1970, 1980, 1990, 2000, 2010]
population = [449.48, 553.57, 696.783, 870.133, 1000.4, 1309.1]
plt.bar(year,population,bar_width,color='black')
plt.xlabel('Year')
plt.ylabel('Population in million')
plt.title('Population of India till 2010')
plt.show()

```

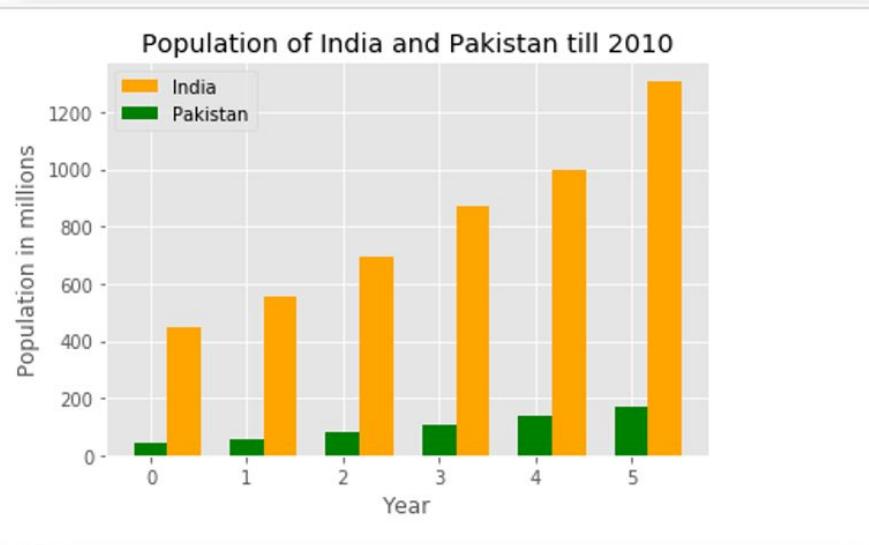
We can specify the width of bars using `bar_width`. This is a positional parameter. All other parameters are the same as line graphs. After running the above code, we will get a bar chart like below:



We can also plot 2 bars side-by-side for comparison purposes.

For the second bar, we have added `bar_width` to index. By doing this we will get two bars side by side. You have to choose appropriate `bar_width` so adjacent bars do not get overlap.

After running the above code, we will get the following bar chart.



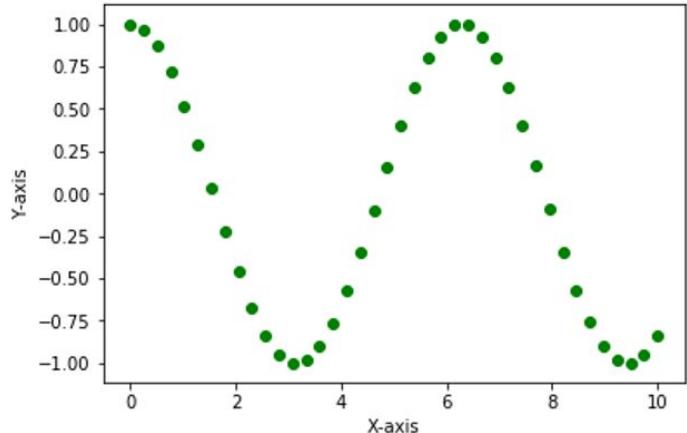
Scatter Plot

The Scatter plot is very close to the line plot. The only difference between line and scatter is that the dot points are not joined by a line in scatter plot.

Below is a simple example of a scatter plot used to plot the cosine function.

```
x=np.linspace(0,10,40)
y=np.cos(x)
plt.scatter(x,y,marker='o',color='green')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

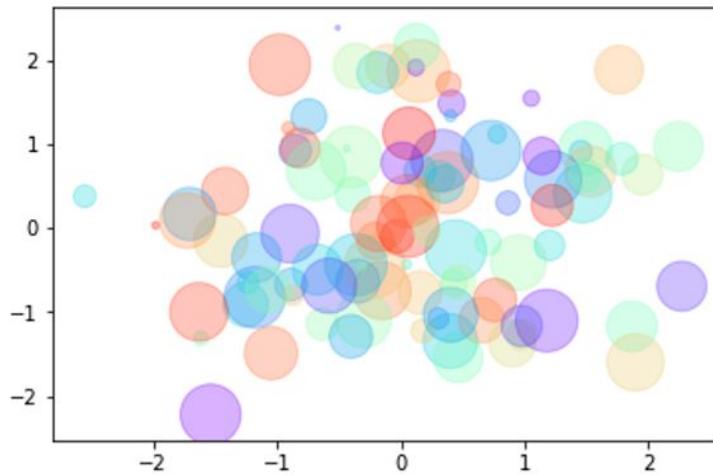
After running the above code, we will get the below scatter plot.



For the below example, we have randomly generated values and used them as data points.

```
rnge = np.random.RandomState(0)
x = rnge.randn(100)
y = rnge.randn(100)
colors = rng.rand(100)
sizes = 1000 * rng.rand(100)
plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,cmap='rainbow')
```

After running the above code, we will get the below graph.



Here we have to note that the color parameter is automatically mapped to a color scale.

Chapter 5: Metrics in Python along with Demo

The terminology like 'gauge,' 'mean,' 'upper90' and 'counter' are integral parts of Python metrics. These are the parts of network applications that one needs to learn to use metrics in their programming. The three crucial parts to comprehending monitoring include the following:

- What is it?
- What is the use of no metrics at all?
- How can we refrain from using metrics the wrong way?

Before we get a head start into this chapter's main topic, let us get to know the prerequisites as far as the software is considered. You will be required to have docker-compose and docker installed to play with the demos that we talk about here. All the demos are available on GitHub. Before we understand the topic, we need to understand why the need for monitoring arises and where it arises from. One needs monitoring for the following reasons:

- To realize the difference between usual and unusual system behavior and services
- To perform the planning of capacities, and to scale down or up
- To work out troubleshooting methods
- To realize the changes and the way the software or hardware influences the working of the system
- Altered behavior of the system as a reaction to a particular measurement

Changes when a System shows unusual Behavior

What Are Metrics And How Many Types Of Metrics Are There?

In Python's language and for our use of the language, metrics define a value that has been observed to represent a particular quantity at any particular point in time. Some examples of metrics include the following:

- The total number of attendees at an event

- The total number of likes on a post
- The total number of times an item was searched
- The total number of users that have signed into a system
- The total number of people reading an article online
- The total number of time you did not find the data you were looking for in the system.

These counts represent a specific value, and these values are broadly called metrics under Python's language. The metrics can be classified into three categories based on their characteristics. They are:

Counters

Suppose you have made a post on your account on a social media platform or published an article.

You wish to keep a tab on the number of views, likes or hits the link receives. This number of views, hits, or likes can only show an increment in the number. This metric is known as a counter.

A counter determines a number that starts at 0 and shows an increment over the time the post or the blog article exists on your social media handle. If you represent a counter graphically, it will be a constant upward rising curve where the metric is always shown to increase.

Gauges

Let us consider the same blog article or the social media post you have put up on your social media account. In the place of the total number of views, likes, or hits, you wish to keep a count of the number of hits your blog article or post receives per day. This number could either increase or decrease. Thus, it is a variable and is commonly known as a gauge in the Python language. Graphically representing a gauge, it has no fixed path concerning the time axis. However, the gauge metric has a ceiling as well as a floor in a particular time frame.

Histograms or Timers

Histograms or timers are metrics that keep track of the observations which have been sampled over time. It is nothing like the gauge or the counter as it does not showcase a net upward or downward trend of data. The fundamental difference between a gauge and a histogram is the purpose for which you require the data. We will understand this more clearly if we visualize a histogram curve that shows latency data over the two axes.

This metric is called histogram by Prometheus and timer by StatsD. To understand the use of the metrics mentioned above, we need to understand the demos in Python.

Calculation and the Report of Metrics in Python

Demo 1

Before we understand the calculation and report of metrics, what is Demo 1? Demo 1 is nothing but a standard web-based program that has been coded using the Flask framework, and it is used to represent the calculation and the report of the metrics. We use the Demo 1 program or code to make a calculation and report particular report latency.

The app.py program or application can be found in the src directory with the pathname src/helpers/middleware.py. When the programmer calls the setup_metrics() from the application, the function named start_timer() must be configured. This must be called before a request is processed; the function named stop_timer() is called when the request has already been processed.

However, in this function, we use the timestamp to note down the operation's time for the request to be carried out where time is measured in milliseconds. After this, the function named docker_compose in the directory for demo1 begins the web application. Following this, the client part of the application makes several requests meant for the web application. Following this, a src/metrics.csv file is created along with two columns - request_latency and timestamp. There are precisely two things we can learn from this file.

They are: There is a lot of data that has been produced. There has been no observation as far as the metric is considered to have typical traits associated with it. This typical trait is a characteristic of the metric. If no

characteristic is related to metric observation, it will be difficult to tell which endpoint of HTTP the metric is attached to. Alternatively, it will be difficult to tell which computer or access node the application had generated in the first place.

Use of Statistical Tools in Python

The metric system in Python uses statistical tools we had used in high school to analyze various data and situations so the application may carry out the task successfully. The tools include the following:

Mean

Mean is also known as the average total of numbers. The sum of the numbers is divided by the number of observations. For example, the mean of 4, 5, and 6 is the sum of the three numbers, 15, divided by the number of observations, 3. Thus, $15/3 = 5$.

The mean of 4, 5, and 6 is 5.

Median

Another statistical tool is the median, which is slightly different from the mean. The median calculation involves the central number among the total observations, arranged from the least to the highest number in ascending order. The order can be the opposite as well. This means that out of three numbers 4, 7, and 11, the median is 7.

The calculation of the median is a longer process than that of the mean.

Percentile

Percentile is used to denote the percentage of observations lying beneath it. It is a relative parameter that measures one constant's performance compared to the other variables or constants. On a general basis, it is supposed to tell you how the number concerns the other numbers. The percentile can range between 0 to 100.

Putting things more interestingly, the median which is 7, has a percentile of 50%.

A few monitoring systems denote the percentile measure as upper_Xin, in which case X is the percentile, and the upper 90 is the data at the 90th percentile. Quantile

A q-quantile is a metric allocating position to the ranks qN where N is a set of numbers. In this case, the value of q ranges from 0 to 1, where both the numbers are included. The median value in terms of quantile is 0.5.

What is the difference between quantile and percentile? The quantile q is corresponding to the measure of the $100q$ percentile.

Histogram and Cumulative Histogram

The histogram is a model that showcases the observed data. It functions based on the implementation of specific monitoring systems. Statistically, histogram categorizes data and pushed them into categories, which are known as buckets.

Let us take the example of your blog where you wish to find out the age group of the people going through your blog. Having a fuller idea of your user's data, you can plot the histogram points, which shows you a cleaner outcome graphically.

However, a cumulative histogram is where the count of every 'bucket' is inclusive of the count registered in the classes of previously recorded. This assembling of total count shows on a histogram where the set of data looks neater graphically. In Demo 1, we understood the need for statistical tools in the data that is being generated. Along with the metrics, another essential part of the game is statistics, as it helps one record the overall experience of a person. In the Demo 1 procedure that we saw above in the working of the latency report calculation and report, only a few characteristics have been identified concerning the request at hand. These characteristics include:

- The endpoint for the HTTP
- The method of the HTTP

The node or host of the identifier on which the request is running. On assigning the characteristics to the observation made using metrics, we can gather some context on the metric and its working. In Demo 2, we can learn about the addition of characteristics to the metrics.

Demo 2

The file `src/helpers/middleware.py` is used to write several columns to the CSV file. During the writing of this particular demo, we have used random and unrelated IP addresses as the node IDs during the metrics report.

Furthermore, on running the `docker-compose up` for `demo2`, the CSV file shows as a result of many options in columns.

CSV files can be analyzed with the use of pandas. When the user runs the `docker-compose up` the file, it takes a print of the web address or URL, which is, in turn, used to open a Jupyter file session. The user can upload the notebook for analysis.ipynb in the Jupyter session and can access the CSV file in the Pandas dataframe. In this code, the `index_col` shows that we wish to use the `timestamp` call to action as the index.

The user can also perform the aggregation and regrouping based on the columns representing each character that we add.

What be Monitored?

In the software system or a code, there are quite a few variables as well as constants. The variables change their value throughout their life in the code. The code that runs in a framework or system changes the software system variables as well. Thus, this is a classic example of something going wrong when you have more data to handle.

Some essential operating system metrics have been given below:

- Usage of System memory
- Usage of CPU
- Usage of Disk Usage
- Usage of File Descriptor.

Apart from this, the other metrics depend on the software application.

Network Applications

In case your code has been written for a network-based application, it will have the purpose to offer help to clients and chat with them. The essential metrics here are:

- The total requests that are registered – counter
- The number of errors yet to be handled – counter
- The latency of request – histogram/timer
- Time queued in case the application has such a system – histogram/timer
- Queued size in case the application has such a system – gauge
- Usage of threads/Worker processes – gauge

These metrics are used to track the behavior of the services and communications with the client. The most important ones are the total number of requests, response time, response status, and request latency.

HTTP Backends of the Web-Based Application

The HTTP applications are ones that should use monitoring above everything else. Additionally, they should also use and keep granular data on the total amount of non-200 HTTP bunch of statuses gathered using all added HTTP status codes. In case your web-based application has a log-in or sign-up availability function, it should also have metrics for those.

Long Processes

The things that fall under long-running processes are task-queue workers or consumers such as RabbitMQ consumers. These are not technically network servers, but they work in such a manner that they pick up a particular task and work on it to process it. Thus, it is vital to monitor the total number of requests processed and the request latency for each process. It does not matter what the type of application may be; every metric should follow with its suitable metadata that comes attached to it.

How to Monitor in a Python Application

There are two processes to integrate monitoring into a Python application. They are:

Update the calculation to process and report the metrics

Make an infrastructure for monitoring, which keeps the metrics used in the application and allows the complaints or queries to be made against them. Users often use context managers, decorators, as well as middleware for the use of network applications to calculate and report the metrics. In both the Demo1 and Demo 2, the use of the Flask application has been made. This will help you understand what a metric type could be in a monitoring system. Furthermore, you can also monitor some of the essential components of the application using these metrics.

Chapter 6: How to Build a Predictive Model in Python

After the data has finally been made useful for modeling, we will now look at the various Python codes that will help us develop a predictive model based on our data set. The most common library used in Python to solve this function is Scikit-Learn (sklearn).

All the inputs and numeric must be converted to a categorical variable and the missing values need to be filled in the data set. Then we will have to import the required modules. After the test, the generic classification function is defined, which takes a particular model as an input, determining the cross-validation and accuracy scores. The details of the coding are not required in an introductory class like this. For a detailed analysis of algorithms and Python codes, you can check online articles.

Logistic Regression

The first model that will be made is a Logistic Regression model. One way to do so will be to write all the variables into the model itself; frequency results in overfitting.

If you take all the variables at once, it may lead to the model understanding the various complex relations only specific to the data set itself. It will not generalize it in other circumstances. On the other hand, we can provide a crucial hypothesis and increase accuracy by adding more variables.

Decision Tree

This is the next method that is employed for making a predictive model on a data set. This method generally provides us with higher accuracy than the Logistic regression model. We can also increase the result by adding various variables, and the cross-validation error may go down, which leads to a model where the data is overfitted.

Random Forest

Random forest is a critical algorithm that will help you to solve the classification problem when it comes to making a model with a data set. One of the main advantages of this algorithm is that it can work with most of the features, and it also provides a feature importance matrix used to select various features.

Make a Model with the Data Set

Now that you understand the various steps required to start with the problem in Python using the Pandas library. Now, you finally make a model with the data set after smoothening extreme values and munging missing data, and you are now ready to work on some problems that will help you further sharpen your knowledge.

So if you are ready to take some exciting challenges, here are some projects:

- Find data on food demand forecasting and then predict the demand for various means for a particular mail delivery company.
- Find data on HR analytics and then identify those employees were most likely to get promotions and those whose job position means to be terminated.
- Find data on the number of upvotes and check the number of upvotes for a particular question asked during an online question and answer sequence in a virtual platform.

These problems will allow you to expand your knowledge on Python and use it in daily life scenarios.

Python is one of the best tools available for coding and has now become one of the most popular coding languages for data scientists. The reason why it is so popular is that not only is it easy to learn, but it can also be well integrated into existing tools and databases.

Learning Python will help you perform data science projects and even help you achieve the full life cycle of these projects. It is much more than coding and requires your full attention in reading, visualizing, analyzing, and finally making fruitful predictions with the data set you are working on.

Data Science and Metrics with Python

When you want to calculate your company's exact metrics, you would need something more than a calculator, specifically a programming language, to keep the records and predict the data in your system. The first and foremost step you need to take is to calculate the exact revenue, income, and expenditure every month. The following chapter will help you know the exact way of using Python in places where necessary.

- **Monthly Revenue:** Various Python programs would help you know the exact monthly revenue you have to work with. Here, you can deal with the exact information on customer IDs, Unit prices, quantity sold, and the invoice dates on which the items were sold. You can use Python to keep track of the above points without much effort at all. It would also help you form a line graph to understand the company's revenue system without even going through the details.
- **Active Customers:** Every business deals with some permanent customers or those who buy often. A significant part of income and revenue comes from these active customers monthly. You can predict the percentage of income in the upcoming months or calculate the details of income in one particular month by counting on these active customers. Python programs would help you take out the list of these sources of fixed incomes and form a graph.
- **Quantity Field:** One of the essential points you should keep in mind while dealing with the metrics is to keep a monthly record of the monthly orders. Some months your business might do well, and it might suffer from a loss in the others. If you use Python to find

out the quantity field or keep a record of every day orders month-wise, it will help you take out the profit or loss metrics at ease.

- **Per Order Revenue:** When you are taking out the metric in your business or company, you should record the monthly Revenue per order. Python would help you track the Revenue per order and predict or calculate the amount of profit per order in no time. You can form a bar graph at ease with Python to find out the revenue details per order monthly without going through the details.
- **New Customers:** If you want to find out the metrics on company progress, then nothing would help you more than details of the new customer gains every day. A line graph with Python to keep a record would help you know your business or company's credentials. The new customer base of your company also marks the increment in income and eventually gives you an idea or estimate to predict the monthly revenue with Python.
- **Retention Rate:** One of the most critical and essential parts of company metrics is the monthly retention rate. The retention rate of customers in a company indicates the improvement or downfall of the company. With its metric bars and programming language, Python would help you know the points of further improvement or investment for betterment.

Data Prediction and Analysis

Only dealing with the metrics is not enough. You also have to take some action based on data science metrics. This part of data science is considered to be the analytical part. In the following section, you would know how to take action based on analytics.

- **Arrangement of Customers:** You already know how to keep a record of your company's customers from the previous section. Now you would have to divide the customers according to their investment in your company. There are mainly three types of customer divisions: the high, mid, and low-level customers.
- **Judging Monthly Data Analysis:** From the above part, you have already taken out your company's monthly expenditure and income. At times, when you suffer a loss, Python's record and graphical

representation of the records would quickly help you understand the flaws in your company. Such a system would help you push your limits from the next month. It would also help you fix your company's flaws and improve the ratio from the previous month's records on Python.

- **Customer Judgment:** After bringing about a positive change in your company's atmosphere or product, you can ask for honest opinions and reviews from the customers. Bring about every possible change in your company to make it reach the peak of success in no time. It would help if you were open to suggestions from the customers as well. Python programs can help you determine the outcome of the recent changes in your company by forming graphs.
- **Use Python:** The most crucial role which Python plays in your company is by keeping records, retention rates, new customer gain rates, etc. Feature engineering based on customer needs is necessary too. The relatable features for the company based on the Python graphs from previous months and the current month would help you understand the basics at ease. You must train yourself and the company employees before bringing about a change; if you feel it is positive for the company, you can continue with it without any hesitation.

You will be able to handle your online business or site properly on knowing the analytics and metrics using data science. In this book, you will come across ideas on data science analytics and metrics using Python.

Chapter 7: Income Increment using Data Science with Python

Have you ever imagined doubling your income using data science? Yes, you heard it right! Now you can boost your income rate at ease without much effort using data science. Data science provides an array of streams and options to find various aspects of your source of income. The streams do not focus on one point but several points equally. You will learn about the details of different ways to boost your income in the following chapter. Let's take a deep dive into the chapter now!

Search Options

When it comes to boosting your income in less time, the first and foremost thing which you should do is to perform some searches based on your needs. You can search for various job options related to data science projects. Such searches would not only provide you with job options but also widen your knowledge of data science projects to boost your income. You would know different facts on data science streams and automatically push your limits with data science. Companies working with data science analytics would offer you job opportunities because of your knowledge on the subject. The more you search, the more you get. So doing searches to look for better job opportunities and increasing your knowledge on the subject can easily increase your experience and knowledge on data science that would opt the companies to seek for you and your work.

Churn Prediction

One of the most important points in increasing growth in the company and increasing income is predicting churn beforehand.

When you have a good customer base for your company, then you should focus on retention soon. The customers in every company keep on changing with time. You cannot expect the same customer base to stay intact forever. Some will come, and some will leave. But your target should be on how to keep the customer base intact. You need to think first if the customers get what they desire from your company, then they would not leave, but if they do not get their needs, then they would leave anyway.

You have to find out the data science for the customers and fulfill their demands before they leave. The attrition rate in a company cannot be nil even after giving your best, but you should at least try to keep it low. The first step towards decreasing the attrition rate is to find out the needs of the customers. According to the needs of the clients, you can easily match what your company can offer them with Python. You have to find out the flaws in your company and work on them for improvement.

You would not be able to decrease the churn if you do not work on the above points. Python can be of great help to predict the churn rate beforehand, and using the prediction graph with Python can help you understand the Churn prediction with ease. Churn prediction models with Python can also be of great help.

Churn Categories

The churn features are of two types, mainly categorical and numerical. The categorical features consist of various categories like gender, streaming, payment modes, etc. At the same time, the numerical features in churn consist of tenure, monthly charges, and total charges.

- Genders in churn categories are one of the most important points. Your company does not stay focused on only one gender. It is for both males and females. It is clear that females or males would not churn alone. But according to the statistics data science, the female customers tend to churn more than the males. So the focus of your company should be to impress the females more. Python can help you predict the churn graph with gender churning at ease.
- Internet service is another factor. People use different internet portals and you can easily use Python to determine which internet service users tend to churn out more usually. When you find it, you can divert some focus on those customers to decrease the churn rate from those internet service portals.
- Contracts play a pivotal role in churn prediction in data science companies. You should be careful and wise with the contracts of your company while offering goods and services to the customers. If the customers find your contracts satisfactory and interesting, they will not churn out irrespective of gender and internet services. Your company should have enough contracts to attract customers. The lesser number of contracts leads to loss of customers and clients of the company.

The contracts should be such that the customers find them quite reasonable. Such tips can be of great use to decrease the churn prediction rate with Python's graph.

- A lot of churn prediction depends on the payment method. The payment method decides whether your company would have a strong customer base or not. If the customers do not get any profit from staying or using data science from your company, then what is the point of staying and sticking with your company. You have to keep your payment method such that the customers find it quite

affordable. The payment method can make a difference of about 30% in your company's churn prediction.

Python's graphs can be of great help to the churn prediction for a numerical feature like tenure. The higher the tenure graph goes, the lower is the churn prediction or company loss rate. If your company can arrange for some good tenure for the customers, then it is obvious that they would stay intact in your company without much variation. Monthly and total charges of a company work similar as that of tenures. Even the monthly and total charges decide a lot about the customer base and spread rate of the information of data science at ease. The more the monthly and total charges of your company, the lesser the churn prediction of the company. The above facts can easily help you increase the profit of your company and decrease the churning rate of your company.

Data Science and Python: The Essential Relationship

Many studies have repeatedly proven that Python is the favorite coding language among data scientists. It is an easy to use language that has good availability of a library and stellar participation from the community. It has often been seen that projects that have an inactive organization often fizzle out because of the lack of updates and maintenance. However, this is not true at all for Python. At the same time, Python is straightforward to learn the syntax angle. It has an extensive library, a very active community that results in the interaction of data science and Python. Python is hence the best programming platform that is available to data scientists as it is adaptable with emerging technologies like machine learning and data science itself. Those who work in the data science department do not want to be slowed down by programming languages that are too complicated. This is where Python becomes handy.

Learning Python for Data Science

When you want to learn Python that can be applied in data science, you will need to begin with a course that will allow you to not only learn at your speed but at the same time, encourage you to write your codes and become an active member of the Python community. However, merely learning Python as a part of a course is not enough; you also need to develop a range of soft skills that will allow you to become successful as a data scientist.

Other than these, there are also some technical skills that you can learn along with Data science which is complementary to one another.

Step 1: Learning the fundamentals of Python

When you begin your journey in Python and data science, you need to start learning the fundamentals of not only Python but also data science. This will include fundamental coding using the Python language as well as an introduction to data science itself. In this regard, there is a particular tool that you must begin using in your initial days of Python, which is known as the Jupyter Notebook. This tool comes with Python libraries, and will help you to learn both the programming language as well as data science.

It is also imperative for you to join our community when learning Python. Some people do not understand the importance of getting associated with the community and its impact on learning Python. However, what people often seem to miss out on is the fact that when you join a community, you surround yourself with those who have specialized or are at a similar level learning Python. This increases the opportunity of learning as well as creates a chance to get notified for any job openings that are available in the area. 30% of employment that takes place in the data science department are from human resource referrals. Hence, you need to consider joining a community.

Another important skill that you can develop along with Python that will help you in learning data science is Command Line Interface. This guide will enable you to run various scripts more effectively as well as quickly. This, in turn, allows you to test programs at a faster rate and hence work with more volume of data.

Step 2: Get practical knowledge by performing short Python projects

Programming depends vastly on hands-on training, and you need to start building your own mini-projects in Python. Although this may seem like a daunting task, it is relatively easy for even those at a beginner level to start working on small Python projects. Here are a few ideas that you can take as guidance to create your Python projects:

Analyze and track your spending habits on Amazon or other online shopping portals : This is a fun little project that will help you to practice not only Python but also Pandas while at the same time providing you with relevant data on your personal finances which is always motivating in nature.

Try a guided project: No matter which Python community or learning group you join, you will always find a host of interactive Python projects that you can work on. These projects are available for every skill level and use actual data. There is no problem if you get stuck here and there because you will always have guidance in completing the project. These projects have versatile challenges that will help you to develop and use your skill in the best manner possible.

Analysis of survey data: You can find a vast majority of data from various surveys online; you can even use your own survey data from a beginner project.

There are many more ways in which you can enjoy programming in Python, let using the calculators and an online game or even getting weather data from Google in your city, which you can then further analyze.

What's more? You may also try your hand at building elementary applications and games that will help you to get adjusted with the nitty-gritty of Python. If you are genuinely motivated, there is no shortage of data or opportunities that you can use to have a career in Python. These mini projects will help you to learn to program and build a solid idea of the various basics that will only enable you to learn further in your career. You can also try building your experiential data with the help of web scraping. Not only will it help you learn Python programming but also is useful for getting data in the years to come.

Step 3: Familiarize with Python data science library

One of the biggest advantages of using Python in data science is that some libraries will help you to get the job done in no time at all. Three of the most important libraries that are used in Python include NumPy, Matplotlib, and Pandas. Some of the most important Python libraries that are useful in data science and their description are as follows:

Matplotlib: This is basically a visualization library that will allow you to generate charts quickly and easily from your data set.

NumPy: This is a library that makes the use of several mathematical and statistical operations easier. It also provides a basis for various features in the Pandas library.

Pandas: This library has been specifically created to facilitate working with various data sets. This is one of the most important libraries for several python data science workers.

Scikit-learn: This is the most critical library available in Python for machine learning.

In short, NumPy and Pandas while Matplotlib is essential for visualization of these data and turning them into graphs that you can easily find in Google sheets for Excel.

At this stage, you need to ask relevant questions where ever you get stuck. If you find that you are not able to understand something, you must make use of the affluent community of experts who are influential in the field of Python and are experts. These experts will provide you with guaranteed answers no matter what your query is. You can make use of platforms such as Stack Overflow, Quora, or any other learning group that you are a part of. These groups are generally formed with very knowledgeable and experienced personnel who will be able to provide you with all the answers you need concerning Python.

Step 4: Build a portfolio

Building a portfolio is imperative for those who are aspiring to become a data scientist. Your portfolio must include projects that come from a variety of data sets and explore various facets of data science. The readers must find your portfolio to be insightful while also getting a look at the level of expertise you show. Some projects that you can definitely include are as follows:

Data Visualization Project: You need to show your skill in developing easy to read as well as attractive visualizations. This is not only a programming challenge but also a designing aspect that needs to be taken care of.

However, if you can strike the right balance, you will be able to create a significant impact on your readers.

Your portfolio will definitely stand out if you can provide visual inputs that are impactful and easy to understand.

Data cleaning projects: If you can clean up unstructured or dirty projects, this will automatically grab the attention of potential employers. This is an essential skill that needs to be employed more often than not. If you can add this to the team, you will definitely be an important candidate.

Various machine learning projects: If you want to work as a data scientist, you need to show your skills in the machine learning department as they are connected. You might have to provide various machine learning projects and there are many of these, requiring different algorithms. It is hence best to showcase a few of these projects that shows your diversity.

A very important aspect that you need to keep in mind here is that your analysis must be provided visually and in an unambiguous format. It is best to offer them in a Jupyter Notebook format which will help technical people to understand your codes. At the same time, enable those from a non-technical background to follow the charts and the written explanation you have provided.

One thing that you must always bear in mind is that your portfolio does not need to maintain a particular theme at all times. It is more vital for you to show your diversity and a vast area of expertise. Hence always choose data sets that interest you, and then you can find a way to put them together. However, if you have a particular company or industry in mind, you should showcase such projects that are relevant to the industry in particular.

When you display such projects, it allows potential employers to collaborate to full term and unleashes all your potentials. At the same time, it shows your future employers that you are passionate about Python and hence have developed some of the most important skills.

You need to communicate and collaborate based on your technical competence at this stage. The more real-life experience you gather, the better your portfolio becomes. Your portfolio actually doubles as a resume

itself and hence has to be perfect. You must also start gathering information about beginner and intermediate level statistics, as it is vital to have a solid understanding of statistics when learning Python for data science.

Step 5: Application of advanced techniques in data science

Once you have covered the basics, you must develop your skills and make them even better. Data science is all about learning and getting accustomed to the new surge of material. You need to learn the advanced forces that will help you to get ahead in your game.

You need to learn regression, k-means clustering, and classification models. Along with this, there are also some machine learning modules that you might find interesting. These include bootstrapping models as well as creating various neural networks with the help of scikit learning. You can also try programming projects that need you to create models with the use of live data feeds. Other machine learning models may also interest you.

Conclusion

Data science is a vital part of our lives today; it exists everywhere, and most of the time you do not see it. Financial institutions, websites, major online marketplaces, social media sites, even email providers use data science every single day to improve your user experience.

Learning Python is one of the best ways to get the hang of data science. It is the preferred language, and it is your starting point. Do not rush this stage – the more you understand Python and how to code with it, the easier you will find data science.

Python for Data Science

**Comprehensive Guide to Data Science with
Python.**

Alex Campbell

Introduction

"Data is the new oil." That's a phrase you'll see a lot in your data science journey, and it's also true to say that being a data scientist is considered incredibly sexy these days. As each day passes, more and more data is produced, and it becomes more imperative to find ways of dealing with all of it. Somehow, we have to make sense of it all, and the only way to do that is through the methods and tools that data science provides us.

This is a short but comprehensive guide on how to use Python for data science and analysis. You will learn how to apply machine learning and Python to real-world applications – technology, retail, science, business, manufacturing, market research, and financial, to name a few. You will see many practical and helpful examples of how modern data analysis works with modern-day problems and problem-solving.

This is a very important thing to earn because the sheer amount of data we are faced with today gives us so many opportunities to gain meaningful insight and positively impact virtually every field. This comes with its own challenges, requiring new approaches and technologies, new mindsets, and skills.

Why Python?

Because it is the easiest and most intuitive programming language, it's as simple as that. Python is widely used by data scientists the world over. Its simplicity ensures that scientists don't have to spend hours and hours learning a complex language before even getting into their data science tasks.

Python provides us with tons of useful tools for data science – Scikit-learn, TensorFlow, NumPy, Pandas, and more – making our jobs much easier and faster to do. Python is the most taught computer programming language globally, and it has one of the largest communities – whatever you want to know, whatever you are stuck on, somebody will know the answer and be happy to share it with you.

Prerequisites

There are some things you need before you dive into this guide:

- Knowledge of Computer Programming and Python

This guide is not a guide into how to use Python – you should already know the basics before you even start on this. You need to understand Boolean operators, comparison operators, variables, lists, loops, functions, and more so. If you don't, go learn them before you start learning data science. I will not be giving you an overview of the Python basics.

You don't need to be an expert, but even the most basic knowledge can help things go much smoother. It's not even that complicated – computer programming is about nothing more than giving a computer a series of commands, telling it what you want it to do. Once your computer understands what you are telling it, it can do what it's told – execute the instructions.

And another reason why Python is used for data science is that much of what you need is already there, built-in to the programming language or included as libraries. Many times, all you may need to do is copy some code, modify it slightly and then execute it. But that's no excuse for laziness. It doesn't mean you shouldn't put your all into learning Python – otherwise, how you spot any problems? How will you troubleshoot things if error messages appear? Learning the programming language will boost confidence because you will know how it all works.

- Install and Set Up Python

If you want to take part in this guide and try all the code examples for yourself, you will need to have Anaconda installed on your system. It's completely free and works on macOS, Windows, and Linus – follow the link to download and install it on your system – full instructions are given at the link:

<https://www.anaconda.com/download/>

Included in Anaconda is Jupyter Notebook, and this is the tool we will use the most. Jupyter is a notebook where your code is typed and executed, where you can add notes and text. Provided Anaconda is installed properly,

you can open the Anaconda prompt and type jupyter notebook at the cursor. The notebook opens in your default browser, and you can create a new one or open an existing one.

Python also includes tools to help you study and analyze much faster, ensuring you know where something has gone wrong and what is needed to fix it.

No Math Needed!

Often, when you do data analysis, you are working with numbers and pulling insights from them. However, you do not need to be an expert in mathematics! Yes, you need decent math skills and programming, and you need some knowledge of the domain you end up working on. But you don't need to be at an expert level for any of them.

You'll come across many people who claim to be experts who can teach you all they know – be aware that many of them are nothing but fakes or know a fraction of what they claim to. Successfully finishing your project means knowing what comes next. This book will not make you an expert, nor does it claim to, but it will give you a decent basis from which to work.

In terms of mathematical expertise, you probably already know most of the common statistics terms – standard deviation, mean, etc. – and you are likely to encounter linear algebra and calculus on your data science journey. If you have time, you can always make it a side-study to learn more about these if you are really interested. It may give you a slight edge over others on your particular project, it may not.

Data science is about problem-solving at the end of the day, and your focus should be on facing challenges and working through them. Focus your attention on the core data science subjects, and you'll not go far wrong.

Tips to Learn Faster

The best tip I can give you to learn faster is to put more time into learning. Learning to program and think as a programmer does takes time. You can also make use of cheat sheets, which you can find all over the internet. Even the most experienced programmers are still learning – nobody knows

everything, and this is one field that is constantly changing. Keeping up with the latest concepts and practicing constantly is all you can do.

With all that said, let's start finding out what data science with Python is all about.

Sources of image and code in this guide:

<https://matplotlib.org/2.0.2/gallery.html>

http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-glr-auto-examples-linear-model-plot-ols-py

http://chem-eng.utoronto.ca/~datamining/dmc/decision_tree_reg.htm

Chapter 1: Data Analysis? Data Science? Or Machine Learning?

Those three terms are commonly heard and often used interchangeably. But, should they be, or are they something completely different?

Data science and data analysis are pretty much the same because they both work towards the same goal – deriving useful insight from data and making decisions based on those insights.

Data analysis is mostly associated with data summarization tools and those finding patterns, such as Microsoft Excel. Data science is associated with coding to manage huge datasets. Data science became more popular as more data came online, from search engines, retail sectors, e-commerce businesses, social media, and more.

Tell someone you are a data scientist, and it sounds much cooler than saying you are a data analyst. The jobs are similar, and their functions overlap to a certain extent – they both generate insights and find patterns in data, both asking the right intelligent questions about the data.

So, where does machine learning fit into the picture?

Again, machine learning and data science are two terms interchangeably used because machine learning is about learning from the data. When we apply a machine-learning algorithm to data, it will find patterns and then use its learned abilities on new data.

Let's say that we want to know if a person is likely to pay their debts. We have a very large database containing details about different people and whether they paid their debts. We also have access to lots of other data from customer profiles, such as age, location, income, occupation, and so on. By applying the right machine learning algorithm, the model can learn from that data. When we add new, previously unseen data, i.e., information about a new credit applicant, the computer will apply what it learned before to that data.

Then, we could create a small program that used information, such as location, age, income, etc., to determine if a person is likely to pay their debts. That is a simple example of a prediction model.

Possibilities and Potentials

Allowing a machine to learn from data opens a whole new world of possibilities and potentials, especially predicting and optimizing data. The availability of huge amounts of data has made this possible, not to mention massive amounts of computer processing power, allowing us to use the cloud or a computer to process ginormous amounts of data daily.

Machine learning and data science algorithms are constantly being perfected, but they are already in use in search engine rankings, rooting out spam email, image recognitions, even in medical diagnoses.

Machine Learning and Data Analysis Limitations

No doubt you have seen the news about how machine learning and data analysis can change the way we work – think AI taking over, job losses, automation, and so on. That won't happen in the future because it is happening right now. Machine learning is already changing how you do things, gathering data from social media sites, search engines, and more. With machine learning, we can do the following much faster:

- Determine whether images contain human faces – image recognition
- Predict whether an ad is appealing or personal enough for a user to click on it - predictions
- Create accurate YouTube video captions – speech recognition, speech-to-text translation
- Whether an engine or engine component will fail – preventative maintenance
- Whether a transaction is fraudulent
- Whether an email is spam

All of this is only possible because we have the processing power and the data we need at our fingertips. However, using Python and machine

learning to analyze all this data is not magic – we can't just wave a magic wand, and it's all done! And it certainly can't be used to solve every problem.

Why not?

It's simple - the performance and the accuracy of the tools we use are almost entirely dependent on the data, not to mention our own judgment and skill.

Sure, algorithms and computers can certainly be used to provide the answers we want, but only if we ask the right questions. Those questions can only come from a human, and it all comes down to whether we use the answers our computers provide.

Performance and Accuracy

Data analysis is commonly used to make predictions and optimize data and results. Some of the questions it can answer are:

Will product demand increase over the next five or ten years?

What are the best delivery routes to save on costs and optimize business?

This is why improving accuracy by just 1% can result in millions of dollars in extra revenue and why a decrease of just 1% can result in huge losses. For example, a major store can use the prediction results to stock up on specific products, should the prediction say that demand will increase.

Logistics and shipping companies can plan their routes better, lowering fuel costs and speeding up delivery times.

Besides an improvement in accuracy, data analysis also ensures better, more reliable performance. How does our analysis work on unseen data? Are there other factors we should consider when we analyze data and make some predictions? No matter what work we do, the results should always be accurate and consistent. Otherwise, the results cannot be reproducible, and it isn't scientific – it's just guesswork.

Aside from optimization and predictions, data analysis also helps us find new opportunities, and we can use our work in different fields and projects to produce results. Dig deeper into the data, and we can also see patterns

and outliers. Let's say we are analyzing customer data, and the data aggregates into large clusters we can explore. We can tap into that information and see whether there are high concentrations in specific spending levels or income ranges.

Those are just a few examples of how data analysis can be used. Next, we'll look at a quick example of how it all works.

The Iris Dataset

We'll use a real-world example to show you how it all works and the potentials that machine learning and Python have on interesting, real-world problems. The computer will use the code we provide to learn and find patterns. It will then use what it learned and apply it to new data.

Here's the code:

```
#import the necessary libraries from sklearn.datasets import load_iris  
from sklearn import tree  
  
from sklearn.metrics import accuracy_score  
  
import numpy as np  
  
#loading the iris dataset  
  
iris = load_iris()  
  
x = iris.data #array of the data  
  
y = iris.target #array of labels (i.e. answers) of each data entry  
  
#getting label names i.e. the three flower species  
  
y_names = iris.target_names  
  
#use random indices to split the dataset into train and test test_ids =  
np.random.permutation(len(x))  
  
#split data and labels into train and test  
  
#keep last 10 entries for testing, rest for training  
  
x_train = x[test_ids[:-10]]  
  
x_test = x[test_ids[-10:]]
```

```
y_train = y[test_ids[:-10]]  
y_test = y[test_ids[-10:]]  
#classify using decision tree  
clf = tree.DecisionTreeClassifier()  
#train (fit) the classifier with the training set  
clf.fit(x_train, y_train)  
#predictions on the test dataset  
pred = clf.predict(x_test)  
print(pred) #predicted labels i.e. flower species  
print(y_test) #actual labels  
print((accuracy_score(pred, y_test)) * 100) #prediction accuracy
```

Run that code and you should see this, or something similar:

[0 1 1 1 0 2 0 2 2 2]

[0 1 1 1 0 2 0 2 2 2]

100.0

In the first line of the output, we see the predictions. For reference:

0 refers to Iris setosa

1 refers to Iris versicolor

2 refers to Iris virginica

In the second line, we have the actual species, as indicated above and from the dataset. Note that we have a prediction accuracy of 100%, meaning we got a correct prediction on every species.

If this seems confusing, don't worry. All you really need to understand is that the model we created predicts the species. We do that by splitting our dataset into two – a training set and a test set. The algorithm is run on the training dataset, and then we use it on the testing dataset to see how accurate it is. We can predict the species in the test set based on what was learned in the training set.

The Potential and the Implications

That was just a small example, but it has huge potential, and the implications are widespread. With a modification or two, that workflow can be applied to many different problems and tasks.

For example, we could use it on other species of flowers, animals, and plants. We could use it in classification tasks, which we'll see more of later, like determining whether cancer is malignant or benign, whether a person is likely to be a customer or not, and whether a photo shows human faces.

The real challenge lies in getting sufficient quality data to train the computer on – the real essence of machine learning lies in training a computer on one set of data and testing it on another, potentially using the same model on future datasets.

You can easily see why there is so much hype around machine learning and data analysis. With sufficient data, automated systems can be created to classify objects and predict events. Computers can learn to classify unlabeled images, especially in medical diagnoses and x-rays.

Data analysis used to look at past events and prepare reports, but now, we can use it to make real-time and future predictions.

In the chapters ahead, we'll take you deeper into a data analyst's mindset and look at different approaches to doing certain things. You will learn that a data analysis workflow and the process typically follows this pattern:

- Asking questions to identify the problem
- Get the data and process it
- Visualize the data
- Choose the approach you want to follow and the algorithm
- Evaluate the output
- Try different approaches
- Compare the results
- Know if you can stop – are the result good enough?

You should always determine the project objectives first. That way, you can set clear boundaries and expectations. Second, the data needs to be gathered or accessed, which we will do in the next chapter.

Chapter 2: Get and Process Your Data

GIGO. Garbage In, Garbage Out is a term you've probably come across already in your programming journey and is commonly used in data analysis. Our data analysis accuracy is heavily dependent on how good the data is – if garbage goes in, we only get garbage out.

As a data scientist or a machine learning engineer, you will spend a lot of time getting quality data and processing it. The only way you can do this successfully is to ensure the data is in a format that data analysts can use. In terms of processing it, it must be done so that algorithms can be applied to it, which will ensure we are doing our job rights.

CSV Files

One of the most common formats is the humble CSV file, especially when using Python for data analysis. CSV stands for Comma-Separated-Values, which means commas separate all the values in the columns.

Here's an example:

cauliflower,6.8

kale,7.2

cucumber,4.2

This is just a simple example, containing two columns but, in most modern data science projects, your data is likely to look more like this:

RowNumber,CustomerId,Surname,CreditScore,Geography,Gender,Age,
Tenure....

1,15634602,Hargrave,619,Italy,Female,42,2,0,1,1,1,101348.88,1

2,15647311,Hill,608,Greece,Female,41,1,83807.86,1,0,1,112542.58,0

3,15619304,Onio,502,Italy,Female,42,8,159660.8,3,1,0,113931.57,1

4,15701354,Boni,699,Italy,Female,39,1,0,2,0,0,93826.63,0

5,15737888,Mitchell,850,Greece,Female,43,2,125510.82,1,1,1,79084.1,
0

6,15574012,Chu,645,Greece,Male,44,8,113755.78,2,1,0,149756.71,1

7,15592531,Bartlett,822,Italy,Male,50,7,0,2,1,1,10062.8,0
8,15656148,Obinna,376,Portugal,Female,29,4,115046.74,4,1,0,119346.
88,1
9,15792365,He,501,Italy,Male,44,4,142051.07,2,0,1,74940.5,0
10,15592389,H?,684,Italy,Male,27,2,134603.88,1,1,1,71725.73,0
11,15767821,Bearce,528,Italy,Male,31,6,102016.72,2,0,0,80181.12,0
12,15737173,Andrews,497,Greece,Male,24,3,0,2,1,0,76390.01,0
13,15632264,Kay,476,Italy,Female,34,10,0,2,1,0,26260.98,0
14,15691483,Chin,549,Italy,Female,25,5,0,2,0,0,190857.79,0
15,15600882,Scott,635,Greece,Female,35,7,0,2,1,1,65951.65,0
16,15643966,Goforth,616,Portugal,Male,45,3,143129.41,2,0,1,64327.2
6,0
17,15737452,Romeo,653,Portugal,Male,58,1,132602.88,1,1,0,5097.67,
1
18,15788218,Henderson,549,Greece,Female,24,9,0,2,1,1,14406.41,0
19,15661507,Muldrow,587,Greece,Male,45,6,0,1,0,0,158684.81,0
20,15568982,Hao,726,Italy,Female,24,6,0,2,1,1,54724.03,0
21,15577657,McDonald,732,Italy,Male,41,8,0,2,1,1,170886.17,0
22,15597945,Dellucci,636,Greece,Female,32,8,0,2,1,0,138555.46,0
23,15699309,Gerasimov,510,Greece,Female,38,4,0,1,1,0,118913.53,1
24,15725737,Mosman,669,Italy,Male,46,3,0,2,0,1,8487.75,0
25,15625047,Yen,846,Italy,Female,38,5,0,1,1,1,187616.16,0
26,15738191,Maclean,577,Italy,Male,25,3,0,2,0,1,124508.29,0
27,15736816,Young,756,Portugal,Male,36,2,136815.64,1,1,1,170041.9
5,0
28,15700772,Nebechi,571,Italy,Male,44,9,0,2,0,0,38433.35,0

29,15728693,McWilliams,574,Portugal,Female,43,3,141349.43,1,1,1,1
00187.43,0

30,15656300,Lucciano,411,Italy,Male,29,0,59697.17,2,1,1,53483.21,0

31,15589475,Azikiwe,591,Greece,Female,39,3,0,3,1,0,140469.38,1

....

In the real world, particularly where social media, E-commerce, online ads, etc., are concerned, data could run to thousands of columns and millions of rows.

CSVs are a convenient format, and there are thousands of them online. They are structured and, with just a couple of lines of easy code, Python makes processing such data simple. The code you need is nothing more than:

```
import pandas as pd
dataset = pd.read_csv('Data.csv')
```

This is a necessary step before your computer and Python can do anything with the data so, whenever your data source is a CSV file and you are using Python, make sure those code lines are there at the top of your project.

Next, the input values are set (X), along with the output files (y). More often than not, the output values are the target outputs. For example, one of the most common data science goals is to work out how certain input values affect their corresponding output values. Later, we can apply that to new input values and see if it is useful in predicting previously unknown output values.

Once the data is readable and we can use it, the next step is often to make sure there isn't too much variance in the magnitude and scale of the values. Why do we need to do this? Because values in some columns may be in a completely different league to others. For example, customer ages can run from 18 to 70, but income ranges from 90,000 to 950,000. These two columns show massive gaps in the values. These would significantly affect the model – the income ranges may contribute more to the predictions that the model pushes out, rather than both columns being treated equally.

Scaling similar or same magnitude values is known as feature scaling and one way we can do this is to use these code lines:

```
from sklearn.preprocessing import StandardScaler  
sc_X = StandardScaler()  
X_train = sc_X.fit_transform(X_train)  
X_test = sc_X.transform(X_test)  
# sc_y = StandardScaler()  
# y_train = sc_y.fit_transform(y_train )
```

What we are trying to do here is scale all our values to be of the same magnitude. That way, all the values, no matter what columns or rows they come from, can contribute to the predictions and resulting outputs.

Machine learning and data analysis often require that the dataset is split into two – a training set and a smaller test set. We need the training dataset, which is much larger, to allow the computer to learn from the data. That learning is then applied to the test set to see how it performs and how accurate the predictions are.

The code below is one of the commonest ways we can achieve this:

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,  
random_state = 0)
```

We imported `train_test_split` from Scikit-Learn, a free Python library for machine learning, and then used it to split the dataset. The most common split is 80% for training and 20% for testing. To do that, we set the test set as:

`test_size = 0.2`

The `random_state` may be whatever value you want so long as you use the same value consistently throughout the project.

While 80-20 is the most common split, you can use whatever ratios you want. Some programmers choose 70-30 while others choose 60-40 – it all depends on what your project requires. Just bear in mind that your training dataset must be large enough to provide the computer with sufficient meaningful data to learn from.

This is why the more data you can use, the better, as it tends to ensure the learning is more accurate. With little data, your system may not be able to spot patterns, and the algorithm we apply to the date may overgeneralize. This will result in the algorithm not working on any new data. In other words, on the existing data, the algorithm will provide excellent results, but it is likely to fail in a big way on new data.

You may also come across a case where you have enough data to ensure the system learns meaningfully. Obtaining new data won't do any good because the effect could be as small as an improvement in accuracy of 0.0000001%. Not only that, gathering more data could result in wasting time, money, and effort, and it's much better to work with what you have than waste time looking for new data.

Feature Selection

While you have a ton of data, how do you know if all of it is relevant? Useful? Which columns in those data are likely to contribute significantly to your results?

Often, some of the data you have is irrelevant. For example, does the name of a startup business affect how successful, or otherwise it can get funding? Does a person's favorite color bear any relationship to how intelligent they are?

Another critical task in data processing is picking the relevant features to analyze. There is little point in wasting time and resources on analyzing irrelevant columns, and features and those features may even result in your analysis being skewed.

Go back to the first line of this chapter – GIGO – Garbage in, Garbage Out.

If irrelevant features are included in the analysis, your results are also likely to be irrelevant and inaccurate. Your computer and the algorithm would only be learning from poor examples, leading to poor results.

Feature selection helps us remove the "Garbage" and ensure our analysis is accurate and relevant. The term is self-explanatory – we select the features with the most relevance to our project, which contribute the most, ensuring the predictive model is much easier to understand.

Let's say we have more than 20 features describing customers. Those could include age, gender, location, income, whether they have children, own a house, their recent purchases, and much more. Not all of these will be relevant to the analysis we are doing or the model we are building. Although it is perfectly possible that all of them will have some effect, the resulting analysis is likely to be too complex to be of any use.

Feature selection allows us to focus on relevance and simplify our analysis. The big question is – how do you know if a feature has relevance? Remember – in the introduction, I told you that data scientists need domain knowledge. That, together with expertise, are important factors in telling the relevance of a feature. For example, taking our example of a customer database for the retail sector, the data team needs retail knowledge. That way, they can choose the right features that will positively affect the model and the analysis.

Different fields will sometimes offer features with differing levels of relevance. For example, retail data must be analyzed differently from wine quality data. With retail, the focus is on features that might influence the purchases a customer makes. At the same time, wine data tends to focus more on the chemical constituents and how they affect preferences.

Domain knowledge is also needed to know what features depend on others and to what level. For example, in the wine quality data, substances present in wine may negatively react with one another, which could affect how much of each substance there is. When the level of one substance is changed, it could affect the levels of other substances.

This applies to the analysis of business data, too. The more customers you have, the more sales you are likely to have. And customers with a higher income are also more likely to spend more money.

All of these features depend on one another, and if we excluded some, the analysis could become more simplified.

When you have a very large dataset, potentially containing hundreds of thousands of columns, it can take time to choose the right features. Often, professionals will try combinations to see what gives the best results or look for what makes sense regarding their project.

In some circles, domain expertise is seen as being more important than skills in data analysis. If we start by asking the right questions rather than applying elaborate, complex algorithms to the data, we get better results. And the only way to know what the right questions are, or the most important ones, is to have someone on your team with domain expertise.

Online Data

We looked at data processing and feature selection, but we haven't looked at where the data comes from. How do we know our data is credible? And for beginners, where do you get your data from so you can practice data analysis?

The best place to start is the UCI Machine Learning Repository:

(<https://archive.ics.uci.edu/ml/datasets.html>)

Here, you gain access to loads of databases relating to social sciences, business, physical sciences, engineering, and life sciences. You can work on data about bank marketing, handwritten characters, social media, sensorless drive diagnostics, El Nino, and much more.

Another place to look is Kaggle:

(<https://www.kaggle.com/datasets>)

Here, you will find datasets on grocery shopping, the popular Titanic Survival dataset, Amazon reviews, housing prices, crimes, statistics, and historical air quality.

Start with both of those places, and you will have plenty to work on. In fact, go ahead, and start browsing their datasets to give you a few ideas on how you could analyze the data. Keep in mind that data analysis is all about exploration and problem solving, and that's why you should take the time to explore the data available.

Internal Data

If you aim to work in a University, a company, or a research institution, you will likely be working with internal data. For example, if you get a job in a large e-commerce business, you will probably be working on that business data, both gathered and generated by the company.

Large companies can generate many megabytes of data per second, and these data are usually stored in or processed in databases. Your job would be to make some sense of the never-ending data streams, using the insights you derive to ensure better profitability and/or efficiency.

First, the data should be relevant to the business operations. This could be the time of purchase, what was purchased, whether it was discounted product, and so on. All this information goes in the database, backed up, allowing the team to analyze it later.

That data can be stored in all manner of file types and formats, like SQLite, CSV, BigQuery, and JSON. The specific file type will depend on what the company finds convenient and its existing infrastructure. The most important thing you would need to know is how to work with all the file types to ensure you get the best insights from the data.

Chapter 3: Data Visualization

Data visualization ensures that our meaningful data analysis is faster and easier to do, and, in many cases, it should be one of the first steps in analyzing data.

The data is accessed, processed, and then visualized to show quick insights, such as looking for outliers, patterns, etc.

Visualization Goals

The two main aims of data visualization are exploring data and communicating it. When data is visualized in histograms, bar charts, or another visualization format, you should be able to see any patterns immediately. If you use a line graph, you will see straight away if a trend is rising. If you use a pie chart, you can see the magnitude of a factor compared to other factors. Data visualization is used because it makes data clearer than long lists of numbers and other data.

For example, look at the graph at the link – it shows the search trend across the world on the word "bitcoin":

<https://trends.google.com/trends/explore?q=bitcoin>

Straightaway you should see a massive, albeit temporary, increase in searches for "bitcoin," but it decreases steadily after that huge peak. It could be that, during the peak, there is a great deal of hype about bitcoin, but that will die down over time because that's what happens with hypes or because people have become familiar with it.

Whatever the case, data visualization showed us those patterns quickly – imagine that data presented in a list of numbers. You know it would take much longer to analyze, but a simple graph showed us at a glance what was going on.

This is why data visualization is important when presenting data to the public or a panel. Some people prefer an overview that doesn't go into vast amounts of details, and the last thing you want to do is present them with a whole heap of boring numbers and text. When you present the data so that

people can immediately see what's important and what isn't, you make a much bigger impact.

Data visualization can be done in a few ways. You can use Microsoft Excel to create graphs and plots or use matplotlib, Bokeh, D3, and Seaborn to create a visualization. We'll be using Matplotlib throughout the rest of this guide, as it is one of the most popular data visualization libraries for Python.

Importing and Using Matplotlib

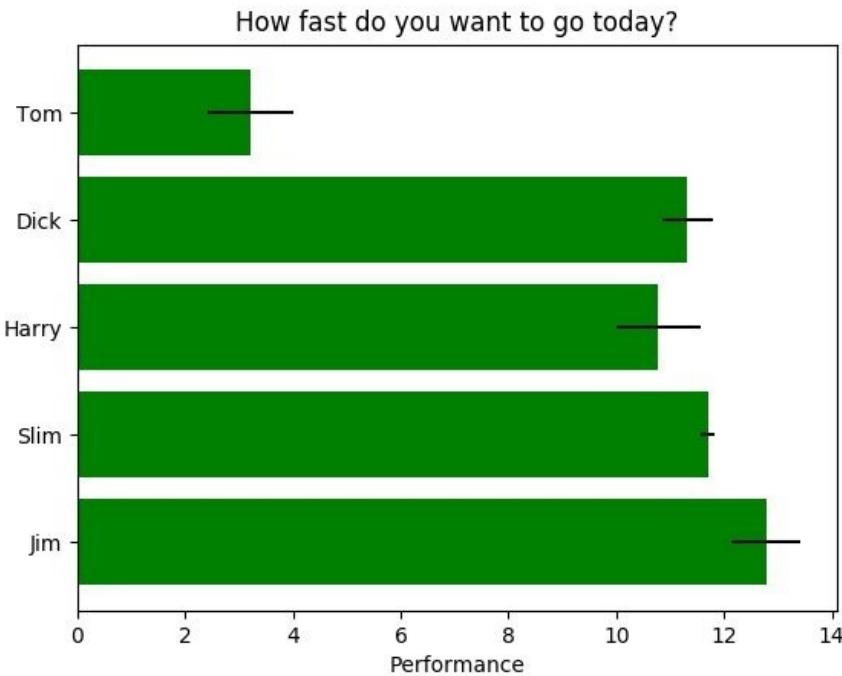
According to the Matplotlib home page:

"Matplotlib is a Python 2D plotting library that produces publication quality figures in various hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits."

This means that using Python and some simple code, you can generate scatterplots, bar charts, plots, and many other visualization types very easily. Rather than wasting all that time and effort on working things out, you can place your focus firmly on faster data exploration and analysis.

Okay, so that sounds like a bit of a mouthful - all you need to remember is that it is about nothing more than exploring the data and communicating the insights.

We can make this clearer by looking at a simple example. Look at the bar chart below:

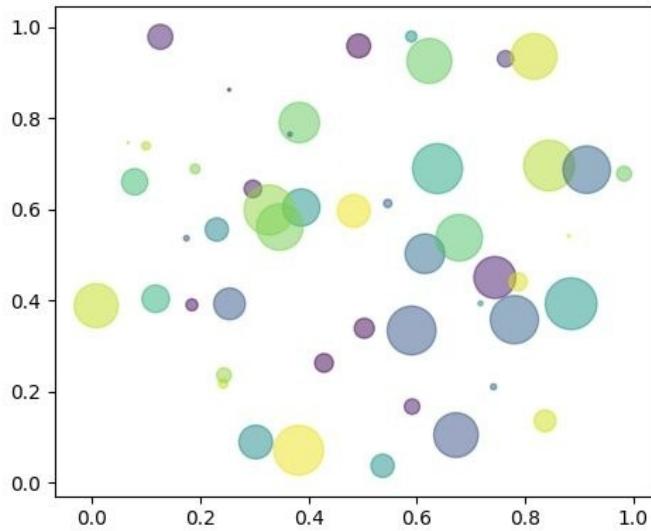


You only need to use the following code block to create that bar chart:

```
import matplotlib.pyplot as plt
plt.rcParams()
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams()
fig, ax = plt.subplots()
# Example data
people = ('Tom', 'Dick', 'Harry', 'Slim', 'Jim')
y_pos = np.arange(len(people))
performance = 3 + 10 * np.random.rand(len(people))
error = np.random.rand(len(people))
ax.barh(y_pos, performance, xerr=error, align='center',
color='green', ecolor='black')
ax.set_yticks(y_pos)
```

```
ax.set_yticklabels(people)
ax.invert_yaxis() # labels read top-to-bottom
ax.set_xlabel('Performance')
ax.set_title('How fast do you want to go today?')
plt.show()
```

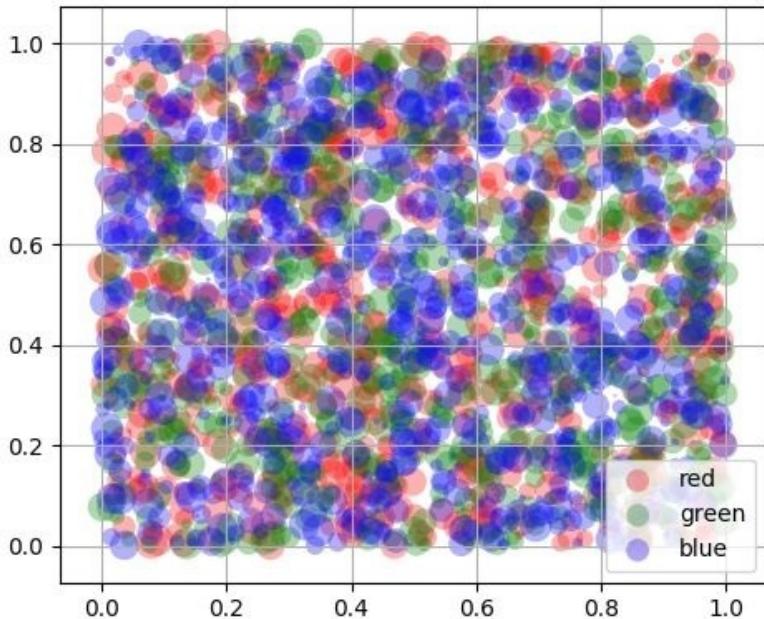
This might look complex, but it really isn't. First, we imported the libraries we needed. Then we set our data and described how we wanted to see it. If you had to write the entire code from scratch, you might struggle a bit, especially as a beginner, but we don't need to do this. All we need to do is copy the code and modify it as per our needs. Matplotlib isn't just useful for creating horizontal bar charts. You can also use it to create boxplots, scatterplots, and lots of other data representations.



.....
Simple demo of a scatter plot.

.....
import numpy as np
import matplotlib.pyplot as plt
N = 50

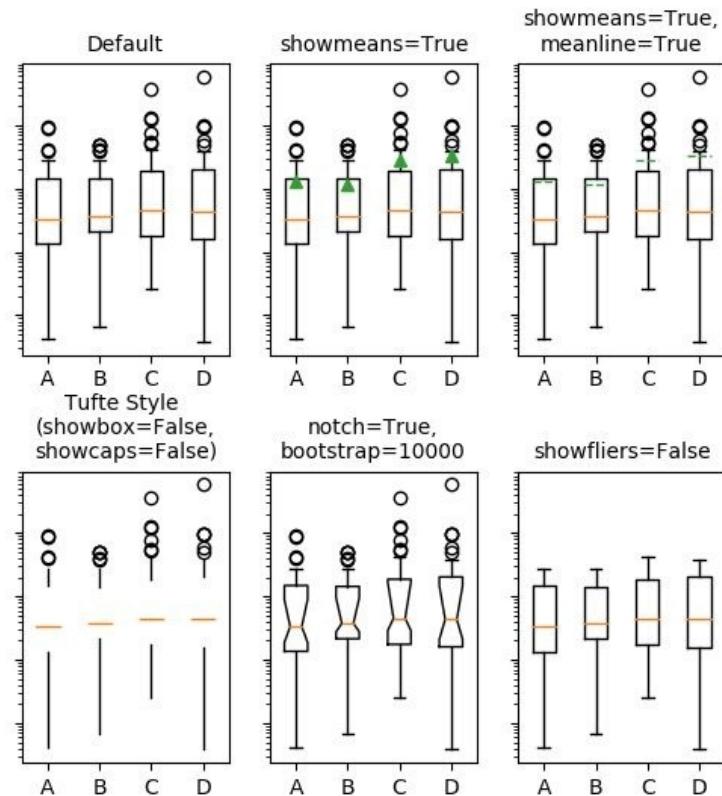
```
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi * (15 * np.random.rand(N))**2 # 0 to 15 point radii
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
```



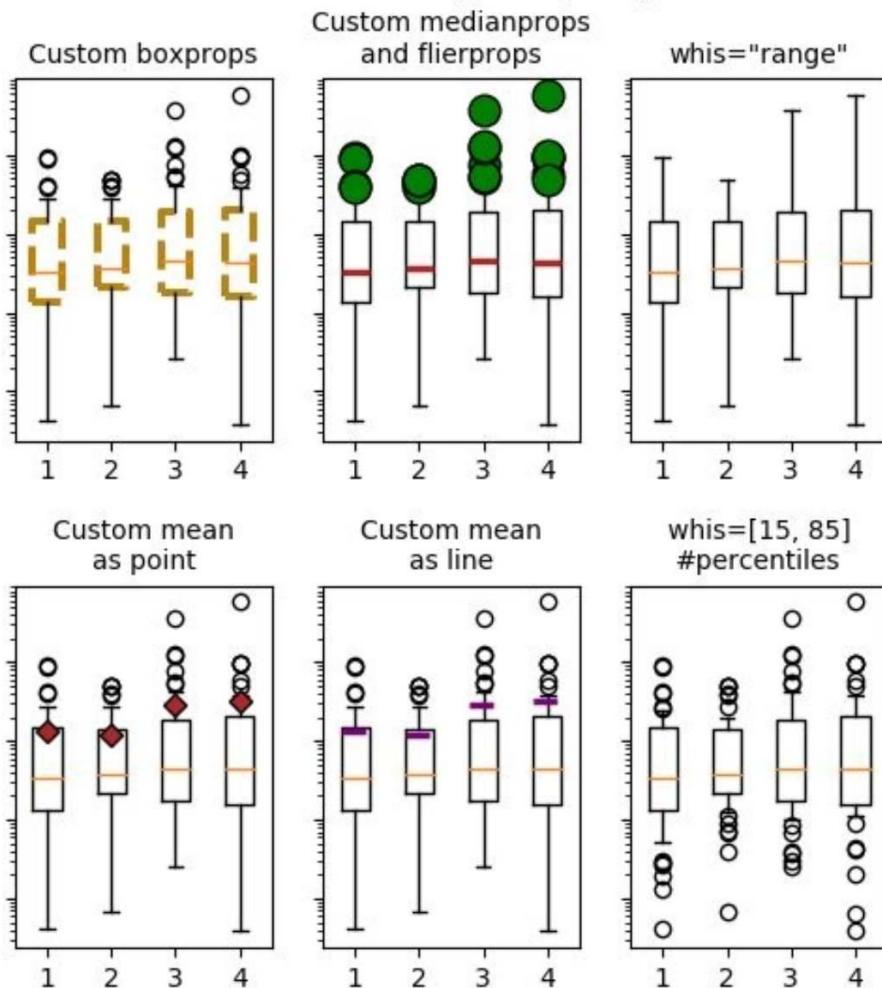
```
plt.show()
```

```
import matplotlib.pyplot as plt
from numpy.random import rand
fig, ax = plt.subplots()
for color in ['red', 'green', 'blue']:
    n = 750
    x, y = rand(2, n)
    scale = 200.0 * rand(n)
    ax.scatter(x, y, c=color, s=scale, label=color,
```

```
alpha=0.3, edgecolors='none')  
ax.legend()  
ax.grid(True)  
plt.show()
```



I never said they'd be pretty



These are just a few simple examples to show you what you can do with Matplotlib.

You can use this cool library to make data visualizations that are suited to publication. You can modify the code as you need it to suit your project and your requirements.

Reinventing the wheel is no longer necessary – simply copy the code you need and adapt it.

Maybe, in the near future, things will get even easier for creating data visualizations, especially where we are dealing with massive datasets. It's

even possible to create animated visualizations, those that change with data changes through time.

We've discussed some general topics relating to data analysis, so now it's time to go a bit more in-depth and look at advanced topics specific to data analysis and machine learning. This book aims to familiarize you with the terms and concepts related to data analysis and data science, and we'll start by looking at supervised and unsupervised learning.

Supervised and Unsupervised Learning

These are two of the most common terms you will encounter in data science and machine learning because they are the two most used machine learning categories used by data scientists.

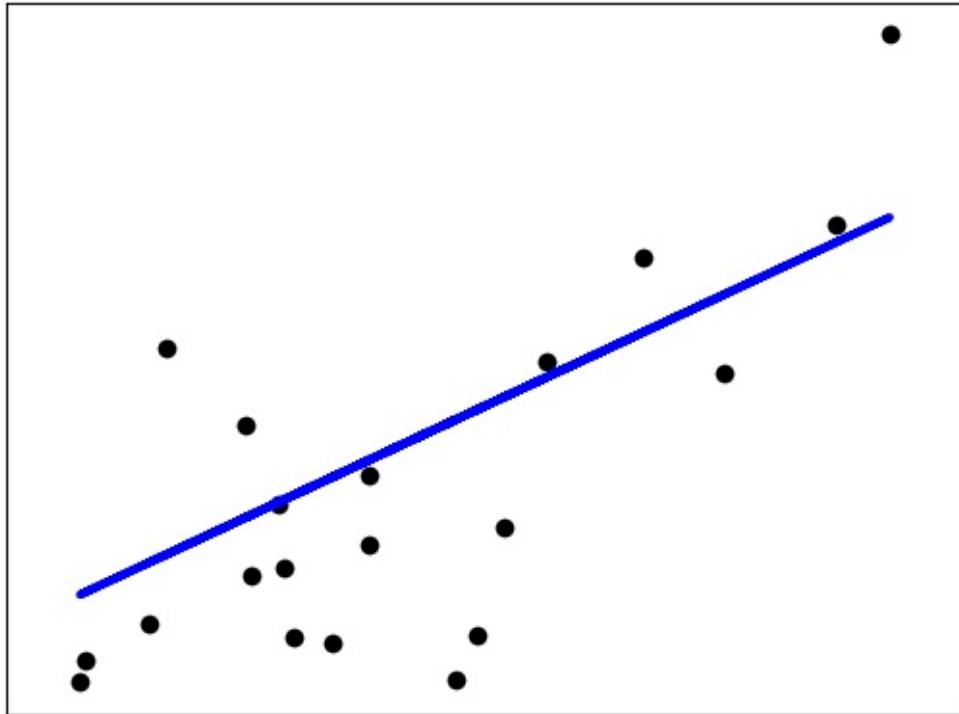
Supervised Learning

Supervised learning is very much like learning by looking at examples. For example, let's say we have a large collection of images of dogs and cats. All the images are correctly labeled as "cat" or "dog." When we give that collection to our computer, it will learn, by looking at them, how to correctly classify a dog or cat picture. As it does, the computer will find similarities and patterns in the images.

When we introduce a new set of images to our computer model, it will correctly identify those images with cats or dogs.

It's much like you sitting down and learning something while being supervised. There are right answers and wrong ones; our computer model needs to put what it learned on the first dataset into practice on the second one, so it can correctly predict the right answers in the majority of cases – it is virtually impossible to reach 100%.

Linear regression is one of the best-known examples of supervised learning. Linear regression is all about predicting an output value (y) for a given input (X). The first thing that must be done is to find patterns in the data. Then, a line must be "fit" to describe the X/y relationship and predict the y values.



```
print(__doc__)

# Code source: Jaques Grobler
# License: BSD 3 clause

import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
# Load the diabetes dataset
diabetes = datasets.load_diabetes()
# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
```

```

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
# Create linear regression object
regr = linear_model.LinearRegression()
# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(diabetes_y_test,
                                         diabetes_y_pred))
# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()

```

This looks relatively simple, but the resulting line came from an iterative minimization of the sum of squares between the values (true) and the predictions. That means the model aimed to come up with the right prediction by using what it learned from the data it looked at before.

Classification is another supervised learning task with the goal of classifying data correctly into one of two categories.

For example, we want to determine if an email is spam or not. The model will learn on a training set of correctly labeled data – spam/not spam – and then use what it learned to create a model to predict labels on new data.

Unsupervised Learning

Unsupervised learning is the opposite – the machine does not have any guidance or supervision to help it. This type of learning is often considered as having acceptable answers but not necessarily correct ones.

Clustering is an unsupervised learning task that tries to determine an aggregation for data points – where the data points cluster naturally. The data points do not have labels, so there are no examples for the machine model to learn from. Instead, it must learn to identify patterns on its own. That is the real essence of AI (Artificial Intelligence), where a computer can do what it has to without any human intervention. It's all about using the data to learn and trying to work out relationships between the inputs. Unlike classification and regression, no output is expected with clustering or unsupervised learning. The focus is entirely on the inputs and looking for relationships and patterns. There may be clusters. The inputs may show clear relationships or associations. Or it could be that the model doesn't find any kind of relationship at all.

Approaching Problems

The common approach to problems among many data scientists is the binary method – is the task covered by supervised learning or unsupervised learning? The easiest way to determine this is to work out what output you expect. Do you want outputs predicted on new inputs? In that case, it would be regression under supervised learning. Or does a new input fall under category A or B, based on already labeled data? That would fall under classification as a supervised learning task. Do you want to see if or how data points aggregate? Are there any natural clusters? That would be clustering under unsupervised learning. Also, under unsupervised learning would be if you were trying to see any interesting relationships between inputs. Most advanced problems in data analysis come under these

questions as the objective is always going to be the same – something needs to be predicted, be it based on training examples or data exploration.

In the next chapter, we will look at regression in more depth.

Chapter 4: A Deeper Look at Regression

In the last chapter, we looked at the differences between supervised and unsupervised learning, and we also talked a bit about linear regression. This chapter will focus more deeply on regression, particularly predicting outputs based on learning and new inputs.

Regression is all about allowing us to see if relationships exist between independent and dependent variables. For example, we could use simple linear regression to tell us if a relationship exists between x and y. This is useful for making predictions on where trends are heading and doing time series modeling, i.e., yearly temperature levels and whether they prove global warming or not.

Simple Linear Regression

In this section, we will use one dependent and one independent variable. Later, we'll look at using multiple variables and use them to predict outputs, similar to how we talked about using multiple attributes or features to make a prediction.

Let's start with a simple linear regression example. We will analyze data about salaries from a file called `Salary_Data.csv`. The dataset is as follows, commas separate all the values and with three columns – years, experience, and salary.

YearsExperience,Salary 1.1,39343.00

1.3,46205.00

1.5,37731.00

2.0,43525.00

2.2,39891.00

2.9,56642.00

3.0,60150.00

3.2,54445.00

3.2,64445.00

3.7,57189.00
3.9,63218.00
4.0,55794.00
4.0,56957.00
4.1,57081.00
4.5,61111.00
4.9,67938.00
5.1,66029.00
5.3,83088.00
5.9,81363.00
6.0,93940.00
6.8,91738.00
7.1,98273.00
7.9,101302.00
8.2,113812.00
8.7,109431.00
9.0,105582.00
9.5,116969.00
9.6,112635.00
10.3,122391.00
10.5,121872.00

The following code is used to fit the training set with a simple linear regression :

```
# Import the libraries
import matplotlib.pyplot as plt
import pandas as pd
```

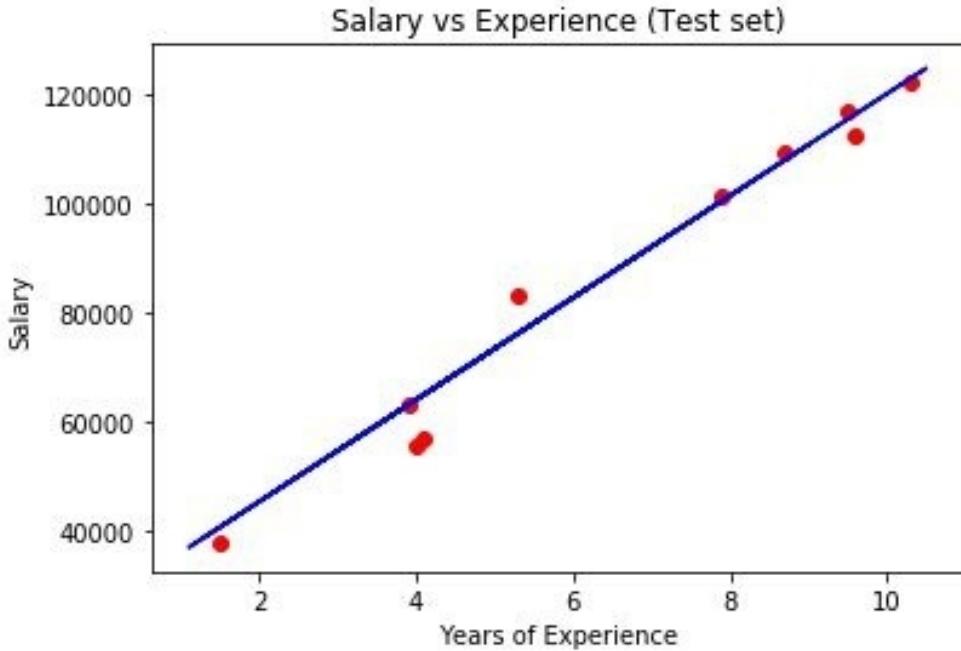
```
# Import the dataset
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
# Split the dataset into the Training set and Test set from
sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3,
random_state = 0)
# Fit Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
# Predict the Test set results
y_pred = regressor.predict(X_test)
# Visualiz e the Training set results
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
# Visual ize the Test set results
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
```

```
plt.show()
```

The goal we are aiming for is a model that uses years of experience to predict salary. First, we use the training set to create the model with 70% of the dataset. It then fits a line as near as it can be to the majority of data points.



Once the line is created, it is applied to our test set, which is 30% of the original dataset.



You can see from the images that our line performed well on the training and the test datasets, and that means the line or the model will do well with new data.

Let's recap what happened. First, the libraries were imported – we need pandas to process the data and matplotlib to do the visualization. Then the dataset was imported. The independent variable, X, was assigned to Years of Experience, and the dependent variable, y, was assigned to Salary. The dataset was split, 70% to training and 30% to testing.

The linear regression model was applied, and Scikit-learn was used to fit a line. We did that with the following code:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

After the model has learned from the training set – X_train and y-train – the regressor is applied to X_test, the test set, and matplotlib is used to create the visualization to compare the results.

It's a pretty straightforward way of doing things – the model learns from the training data, applies what it learned to the testing data, and determines if

the model is good enough. That is the principle of simple linear regression.

Multiple Linear Regression

The same is also similar to multiple linear regression. We still have the same goal, to fit a line showing the relationship between independent and dependent variables. The difference between this and simple linear regression is that we have more than one independent variable.

Let's look at an example of a dataset called 50_Startups.csv, containing details of 50 startup companies.

R&D Spend,Administration,Marketing Spend,State,Profit

165349.2,136897.8,471784.1,New York,192261.83

162597.7,151377.59,443898.53,California,191792.06

153441.51,101145.55,407934.54,Florida,191050.39

144372.41,118671.85,383199.62,New York,182901.99

142107.34,91391.77,366168.42,Florida,166187.94

131876.9,99814.71,362861.36,New York,156991.12

134615.46,147198.87,127716.82,California,156122.51

130298.13,145530.06,323876.68,Florida,155752.6

120542.52,148718.95,311613.29,New York,152211.77

123334.88,108679.17,304981.62,California,149759.96

101913.08,110594.11,229160.95,Florida,146121.95

100671.96,91790.61,249744.55,California,144259.4

93863.75,127320.38,249839.44,Florida,141585.52

91992.39,135495.07,252664.93,California,134307.35

119943.24,156547.42,256512.92,Florida,132602.65

114523.61,122616.84,261776.23,New York,129917.04

78013.11,121597.55,264346.06,California,126992.93

94657.16,145077.58,282574.31,New York,125370.37

91749.16,114175.79,294919.57,Florida,124266.9
86419.7,153514.11,0,New York,122776.86
76253.86,113867.3,298664.47,California,118474.03
78389.47,153773.43,299737.29,New York,111313.02
73994.56,122782.75,303319.26,Florida,110352.25
67532.53,105751.03,304768.73,Florida,108733.99
77044.01,99281.34,140574.81,New York,108552.04
64664.71,139553.16,137962.62,California,107404.34
75328.87,144135.98,134050.07,Florida,105733.54
72107.6,127864.55,353183.81,New York,105008.31
66051.52,182645.56,118148.2,Florida,103282.38
65605.48,153032.06,107138.38,New York,101004.64
61994.48,115641.28,91131.24,Florida,99937.59
61136.38,152701.92,88218.23,New York,97483.56
63408.86,129219.61,46085.25,California,97427.84
55493.95,103057.49,214634.81,Florida,96778.92
46426.07,157693.92,210797.67,California,96712.8
46014.02,85047.44,205517.64,New York,96479.51
28663.76,127056.21,201126.82,Florida,90708.19
44069.95,51283.14,197029.42,California,89949.14
20229.59,65947.93,185265.1,New York,81229.06
38558.51,82982.09,174999.3,California,81005.76
28754.33,118546.05,172795.67,California,78239.91
27892.92,84710.77,164470.71,Florida,77798.83
23640.93,96189.63,148001.11,California,71498.49
15505.73,127382.3,35534.17,New York,69758.98

22177.74,154806.14,28334.72,California,65200.33
1000.23,124153.04,1903.93,New York,64926.08
1315.46,115816.21,297114.46,Florida,49490.75
0,135426.92,0,California,42559.73
542.05,51743.15,0,New York,35673.41
0,116983.8,45173.06,California,14681.4

You can see that there are several features – R&D Spend, Marketing Spend, Administration, etc. We aim to see if there are any relationships between the independent variables and the dependent variable of Profit. You should also see that the data is in text format under the State column rather than in number format. That text is New York, California, and Florida. So, how do we deal with this?

A good way is to transform the categorical data – the text – into numerical and we do this with the code below:

```
from sklearn.preprocessing import LabelEncoder , OneHotEncoder  
labelencoder = LabelEncoder()  
X[:, 3] = labelencoder.fit_transform(X[:, 3]) #Note this  
onehotencoder = OneHotEncoder(categorical_features = [3]) X =  
onehotencoder.fit_transform(X).toarray()
```

Pay attention to

```
X[:, 3] =  
labelencoder.fit_transform(X[:, 3])
```

We transformed the data from the State column, which is the fourth column – note that it is numbered 3 because indexing begins at zero in Python. We wanted to transform this data into something we could work with, so we created dummy variables, taking a value of 0 or 1. That means they indicate that something is there or it isn't.

For example, we have this data with categorical variables:

3.5, New York 2.0, California 6.7, Florida

If dummy variables were instead, that data would look like this:

3.5, 1, 0, 0

2.0, 0, 1, 0

6.7, 0, 0, 1

The State column is now the equivalent of three columns:

	New York	California	Florida
3.5	1	0	0
2.0	0	1	0
6.7	0	0	1

As we said earlier, a dummy variable is used to indicate whether something is there or not. Commonly, they are used as substitutes, allowing us to take qualitative data and do quantitative analysis. From that table, we can easily see that New York is 3.5, split as follows:

- New York – 1
- California – 0
- Florida – 0

This is a very easy way of representing text categories into numeric values.

However, you need to watch out for the dummy variable trap. In this trap, an extra variable exists – this could have been eliminated because the other variables could predict it. In the above example, when the New York and California columns are zero, you automatically know it is Florida. Even with two variables, you can know the State.

Going back to our Startups example, we can get around this trap by having the following in the code:

X = X[:, 1:]

Here's what we have so far:

```

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('50_Startups.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values

```

Have a look at the data:

dataset.head()

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

Next, the categorical variables are transformed into numeric or dummy variables:

```

# Encode categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder = LabelEncoder()
X[:, 3] = labelencoder.fit_transform(X[:, 3])
onehotencoder = OneHotEncoder(categorical_features = [3]) X =
onehotencoder.fit_transform(X).toarray() # Avoiding the Dummy Variable Trap
X = X[:, 1:]

```

This is the result of all your data preprocessing: Note that the categorical variables are gone, and the redundant variable is eliminated, so we don't fall

into the dummy variable trap.

Now we can get on with dividing the dataset and we use the code below to do this:

0	1	165349	136898	471784
0	0	162598	151378	443899
1	0	153442	101146	407935
0	1	144372	118672	383200
1	0	142107	91391.6	366168
0	1	131877	99814.7	362861
0	0	134615	147199	127717
1	0	138298	145538	323877
0	1	128543	148719	311613
0	0	123335	108679	304982
1	0	101913	110594	229161
0	0	100672	91798.6	249745

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,  
random_state = 0)
```

We've gone for an 80/20 split so our next step is creating a regressor, fitting the line and then using it on our testing data:

```
from sklearn.linear_model import LinearRegression  
regressor = LinearRegression()  
regressor.fit(X_train, y_train)  
# Predicting the Test set results  
y_pred = regressor.predict(X_test )
```

The predicted Profit values for X_test look like this:

```
array([103015.20159796, 132582.27760815, 132447.73845175, 71976.09851258,  
178537.48221056, 116161.24230166, 67851.69209676, 98791.73374687,  
113969.43533013, 167921.06569551])
```

Surely, there is more to it than that? Did all of the variables contribute to the target?

Many data analysts will add in extra steps in the hope of creating better predictors and models. They could include Backward Elimination, which is where variables are eliminated one at a time until there are just a couple left. In that case, we know which variable contributes the most to the result, providing accurate predictions.

There are many ways to make your model give better results; it all comes down to your objectives and resources.

Decision Tree Regression

The regression method is great if a linear relationship exists between the independent and dependent variables. But what if that relationship doesn't exist, even though the target can still be predicted using the dependent variables?

That's where methods like decision tree regression can be used. In this method, there is no linearity; it works by breaking the dataset down into ever-smaller subsets. Have a look at the following illustration to explain it:



Rather than lines being plotted and fit, decision nodes and leaf nodes are used.

Here's an example of how it all works – we're using a dataset called Position_Salaries.csv and it looks like this:

```
Position,Level,Salary Business
Analyst,1,45000
Junior Consultant,2,50000
Senior Consultant,3,60000
```

Manager,4,80000
Country Manager,5,110000
Region Manager,6,150000
Partner,7,200000
Senior Partner,8,300000
C-level,9,500000
CEO,10,1000000

```
# Decision Tree Regression
# Import the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values
y = dataset.iloc[:, 2].values
# Split the dataset into the Training set and Test set
"""from sklearn.cross_validation import train_test_split
X_train, X_test,
y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)"""
# Fit Decision Tree Regression to the dataset
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X, y)
# Predicting a new result
y_pred = regressor.predict(6.5)
```

```

# Visualize the Decision Tree Regression results (higher resolution)
X_grid = np.arange(min(X), max(X), 0.01)

X_grid = X_grid.reshape((len(X_grid), 1))

plt.scatter(X, y, color = 'red')

plt.plot(X_grid, regressor.predict(X_grid), color = 'blue') plt.title('Truth or Bluff (Decision Tree Regression)')

plt.xlabel('Position level')

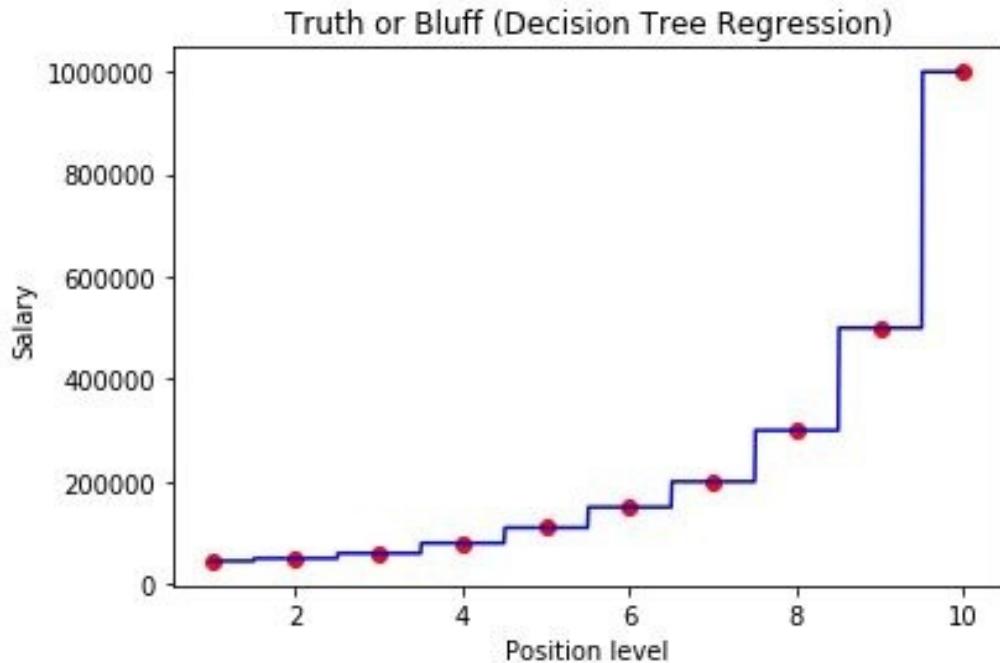
plt.ylabel('Salary')

plt.show()

```

Run that code in Jupyter notebook, and you should see something like the following:

When you run the previous code, you should see the following in the Jupyter Notebook:



As you can see, a linear relationship doesn't exist between Position Level and Salary. Instead, the result is a step-wise one. There is a relationship between the two, and we can see it, but it is expressed differently, in a less straightforward approach.

Random Forest Regression

Decision tree regression works when there is little linearity between independent and dependent variables. However, in this approach, the dataset is only used once to produce the results. Why? Because most of the time, it is better to see what results other approaches produce, such as using multiple decision trees and getting an average of the results.

This can be solved using random forest regression, nothing more than a collection of decision trees, using different subsets, where the results are averaged. It's similar to creating decision trees over and over and getting a result from each one.

In code form, it would look like this:

```
# Random Forest Regression

# Import the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline

# Import the dataset
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values
y = dataset.iloc[:, 2].values

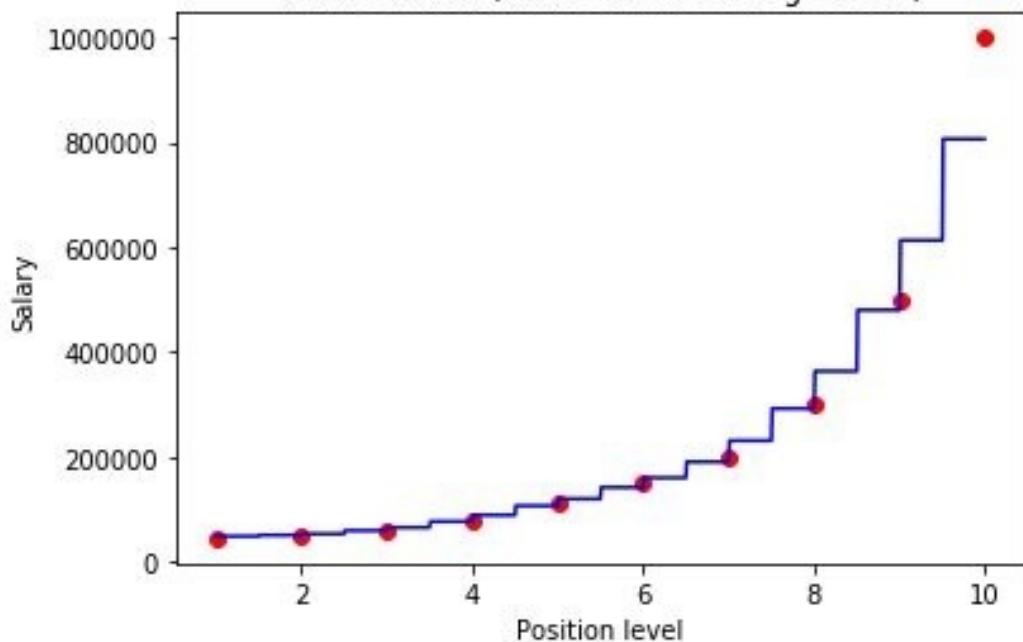
# Split the dataset into the Training set and Test set
"""from sklearn.cross_validation import train_test_split X_train, X_test,
y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
0)"""

# Feature Scaling
"""from sklearn.preprocessing import StandardScaler sc_X =
StandardScaler()

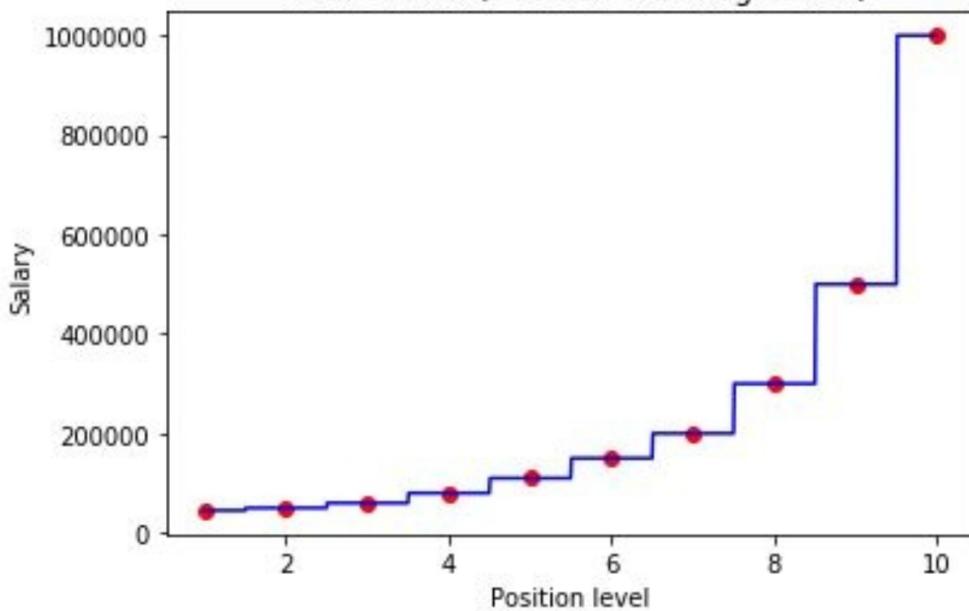
X_train = sc_X.fit_transform(X_train)
```

```
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train)"""
# Fit Random Forest Regression to the dataset
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 300, random_state
= 0) regressor.fit(X, y)
# Predict a new result
y_pred = regressor.predict(6.5)
# Visualize the Random Forest Regression results (higher resolution)
X_grid = np.arange(min(X), max(X), 0.01)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue') plt.title('Truth
or Bluff (Random Forest Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show( )
```

Truth or Bluff (Random Forest Regression)



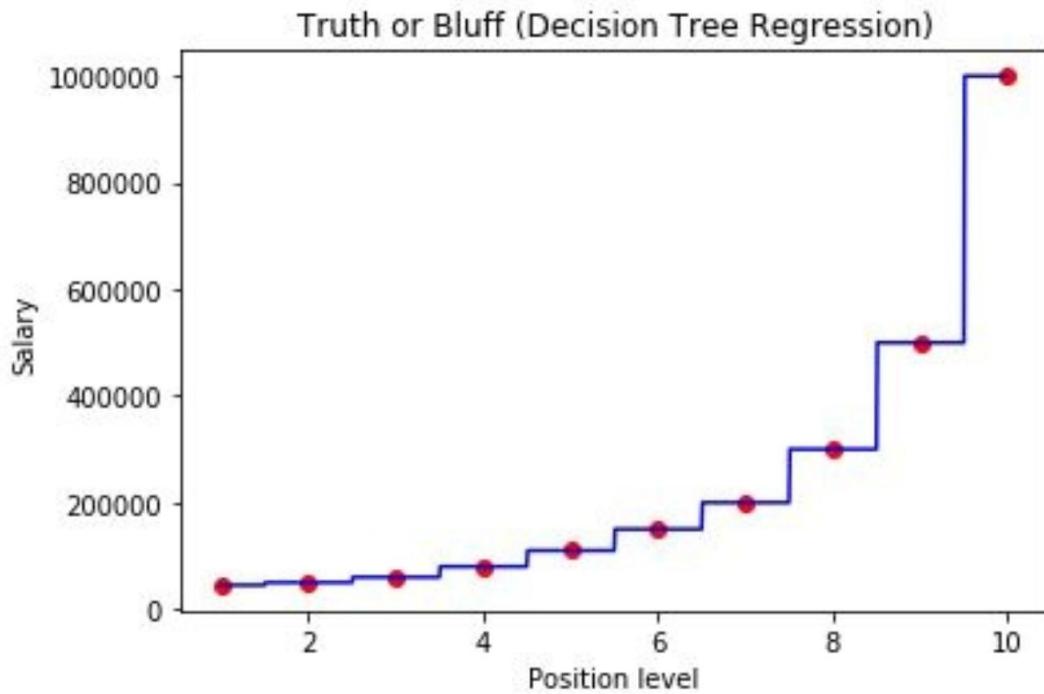
Truth or Bluff (Decision Tree Regression)



As you can see, it is much like the decision tree regression – it is a collection of trees, anyway. If the dataset doesn't show much deviation, the results should look similar. Here's a visualization to see what they would

look

like:



Most data scientists prefer using random forest regression because the averaged results are good at reducing errors. When you look at the code, it seems fairly simple, but it's a different matter behind the scenes, where complex algorithms are doing their work. Think of it as being akin to a black box – you have an input, a black box, and a result. We don't know much about what goes on in the black box, but we could find out if we were prepared to dig deep into the mathematics.

We'll come across this quite a lot throughout this guide, and, in the next chapter, we go on to discuss classification.

Chapter 5: Digging into Classification

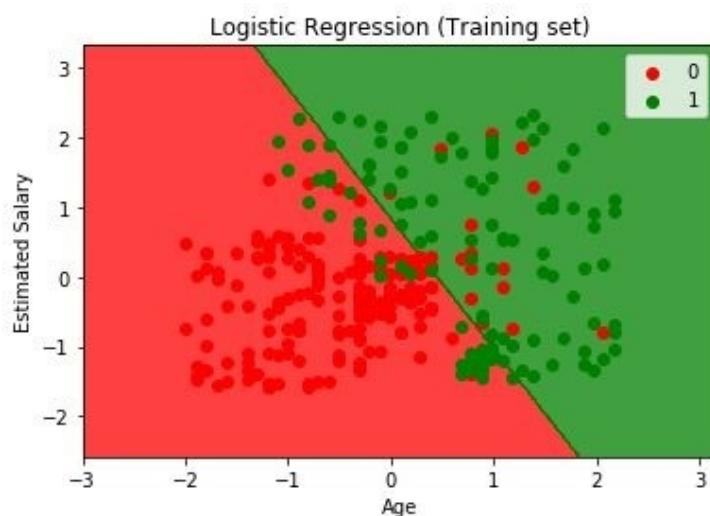
One of the most popular uses for classification is determining whether an email is spam or not. It falls under supervised learning, the same as regression – the model uses labeled data to learn from and then applies what it learned to a new dataset. Let's say we have a dataset containing lots of different email messages. Each one has a label of Pam or Not Spam. The model looks for patterns in the messages marked Spam and, when the model performs the prediction, the model looks for the same patterns in new messages.

There are several approaches to classification, and we're going to discuss a few of them now.

Logistic Regression

Many classification tasks use two independent variables to decide if the result is 0 or 1. Let's say, with variables of Estimated Salary and Age, we want to decide an outcome, like if and when a person made a purchase. How would we do this? We need a model that will show relationships between variables and make predictions.

The illustration below explains this better:



We have our two variables, Age and Estimated Salary. The data points are classified – 0 indicates a purchase was not made, and 1 indicates a purchase was made. As you can see, a line separates the two, and colors are used to

make it easy to visualize. This is logistic regression, and it's based on the probability of a data point being 1 or 0.

Remember that black box we talked about in the last chapter? This is much the same, and there is a lot of complex stuff going on behind the scenes. There is good news – implementing it is straightforward, particularly when Scikit-learn and Python are used.

Have a look at the dataset = Social_Network.csv:

	User ID	Gender	Age	Estimated Salary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0

```
# Logistic Regression  
# Import the libraries  
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd
```

```
%matplotlib inline

# Import the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Split the dataset into the Training set and Test set from
sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fit Logistic Regression to the Training set from sklearn.linear_model
import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

# Predict the Test set results
y_pred = classifier.predict(X_test)

# Make the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Visualize the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
```

```

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step =
0.01),
np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
step = 0.01)) plt.contourf(X1, X2,
classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap =
ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualize the Test set results

from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test

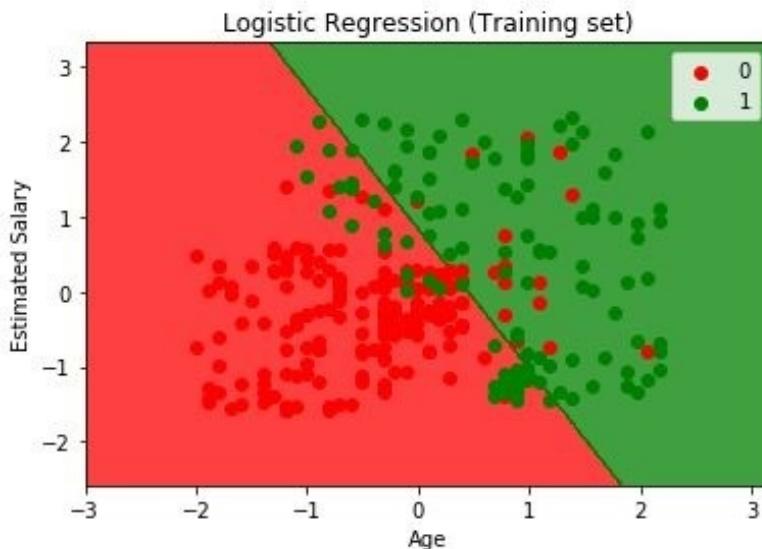
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step =
0.01),
np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
step = 0.01)) plt.contourf(X1, X2,
classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap =
ListedColormap(('red', 'green')))
```

```

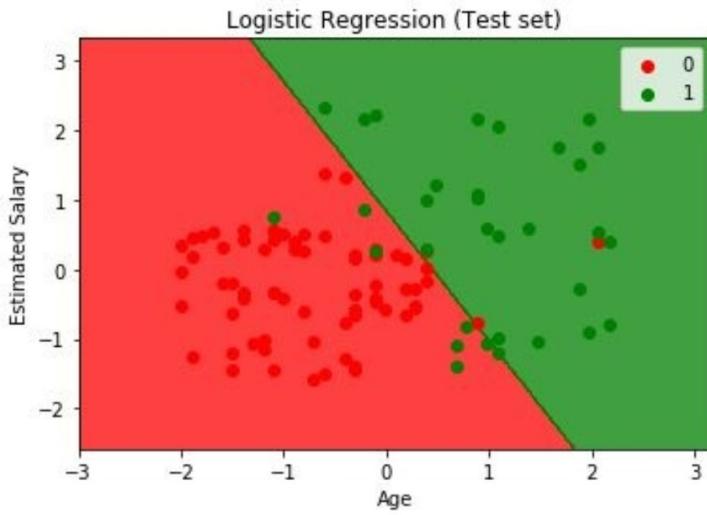
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

```

Run that, and this will show up in your Jupyter notebook:



As you already know, supervised learning uses training data, which the machine learns on, and applies it to the test set to see if the model is accurate at predicting results on new data.



In the test set visualization, you can see that most green dots are in the green region, with just a few red ones. 100% accuracy is almost impossible to achieve with logistic regression. So, the model could be good at predicting whether people of a specific age and with a specific estimated salary would make a purchase or not.

Make sure you look carefully at this code block:

```
# Feature Scaling from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

The data was transformed to get it in the same scale, thus avoiding reliance on one variable or skewing. The Estimated Salary in our dataset has been expressed in thousands but age is a much smaller scale. We need them all in the same range to make our model more reasonable.

Logistic regression isn't the only way to do classification.

K-Nearest Neighbors

You've already seen that there is a linear boundary between 0 and 1 in logistic regression. The result of that is that some data points are missed, some that should have been placed on the other side.

Some non-linear models more accurately capture more data points, and one of those is using K-Nearest Neighbors.

This provides a new data point and then counts the number of neighbors that belong in each category. If there are more in category A than B, the new point goes into category A.

In this case, when a new point is classified, it is based on a majority of nearest neighbors. We can do this using the code below:

```
from sklearn.neighbors import KNeighborsClassifier classifier =  
KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p  
= 2)  
  
classifier.fit(X_train, y_train)
```

Again, rather than writing everything from scratch, we can import code already written and modify it for our purposes. This makes everything much faster. Sure, you could learn everything that goes on behind the scenes, but, in most cases, pre-built code is more convenient and enough to give us a decent model.

Using the Social_Network.csv database again, here's how we would implement this code:

```
'Social_Network_Ads.csv':  
  
# K-Nearest Neighbors (K-NN)  
  
# Import the libraries  
  
import numpy as np  
  
import matplotlib.pyplot as plt  
  
import pandas as pd  
  
%matplotlib inline
```

```
# Import the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
# Split the dataset into the Training set and Test set from
sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Fit K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p
= 2)
classifier.fit(X_train, y_train)
# Predict the Test set results
y_pred = classifier.predict(X_test)
# Make the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
# Visualize the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
```

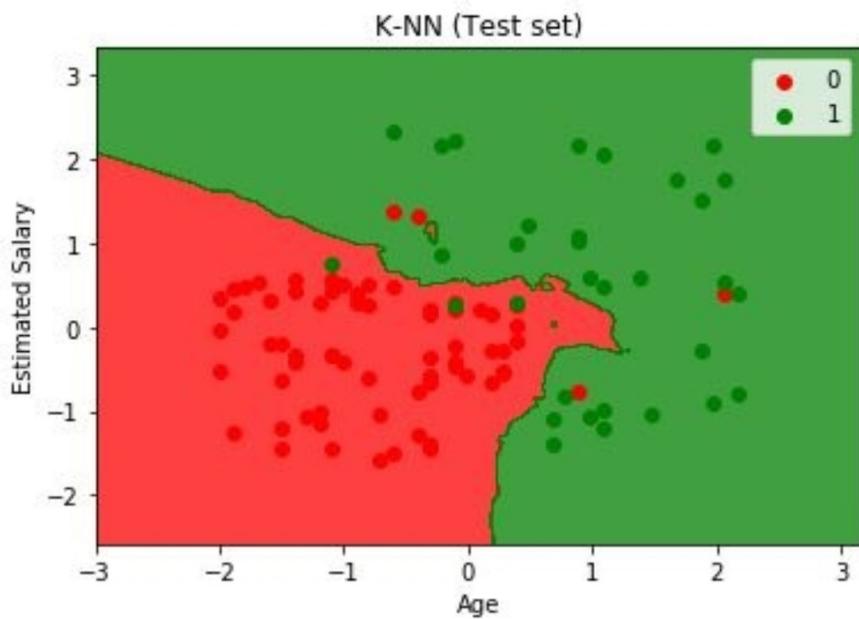
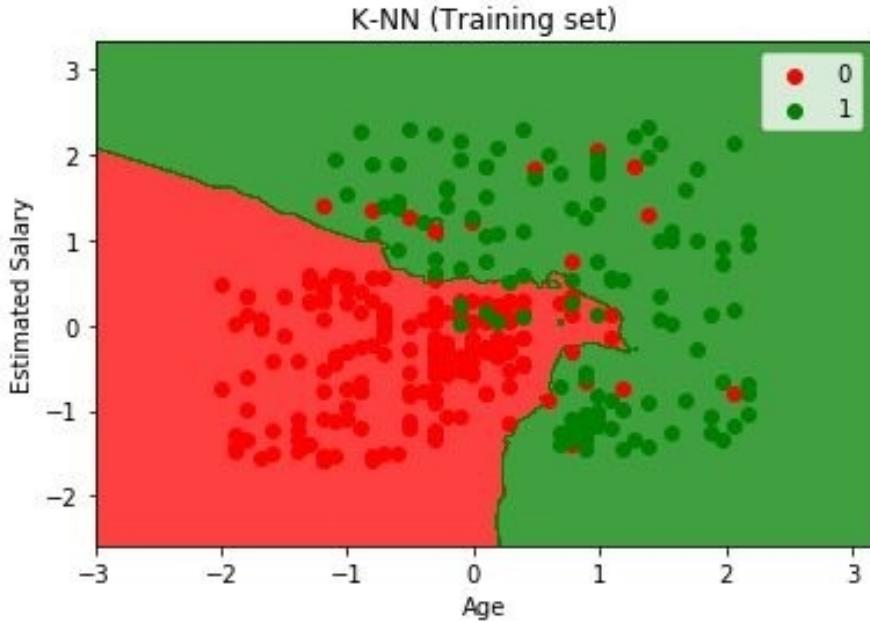
```

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
step =
0.01))
plt.contourf(X1,
X2,
classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
# Visualize the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
step =

```

```
0.01))  
plt.contourf(X1,  
X2,  
classifier.predict(np.array([X1.ravel(),  
X2.ravel()]).T).reshape(X1.shape),  
alpha = 0.75, cmap = ListedColormap(('red', 'green')))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i, j in enumerate(np.unique(y_set)):  
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],  
                c = ListedColormap(('red', 'green'))(i), label = j)  
plt.title('K-NN (Test set)')  
plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()  
plt.show()
```

Run it in Jupyter notebook, and here's what you would see:



As you can see, we have a non-linear boundary. This is because K-Nearest Neighbors takes a different approach. You can also see that some data points have still been missed, i.e., some red dots still show up in the green region. If you wanted to capture all the red dots, you would need to consider using a much larger dataset or a different method. Alternatively, you might need to consider that there is no way of capturing 100% because the model and data are not, and never will be, perfect.

Decision Tree Classification

Decision trees are often implemented in classification, and, as you learned earlier, decision trees are all about breaking datasets down into smaller subsets. We can also create an associated decision tree by branching the subsets out.

Here's an illustration to show you what we mean:



The result of breaking the dataset down into small subsets are branches and leaves. This can be applied in classification using the code below with the Social_Network.csv dataset:

```
# Decision Tree Classification

# Import the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline

# Import the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Split the dataset into the Training set and Test set from
sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fit Decision Tree Classification to the Training set from sklearn.tree
import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)

# Predict the Test set results
y_pred = classifier.predict(X_test)

# Make the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Visualize the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
step =
0.01))
plt.contourf(X1,
X2,
classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
alpha = 0.75)

plt.scatter(X_set[y_set == 0, 0], X_set[y_set == 0, 1], color = 'red', edgecolor = 'black', s = 50)
plt.scatter(X_set[y_set == 1, 0], X_set[y_set == 1, 1], color = 'blue', edgecolor = 'black', s = 50)

plt.title('Decision Tree Classification')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```

```

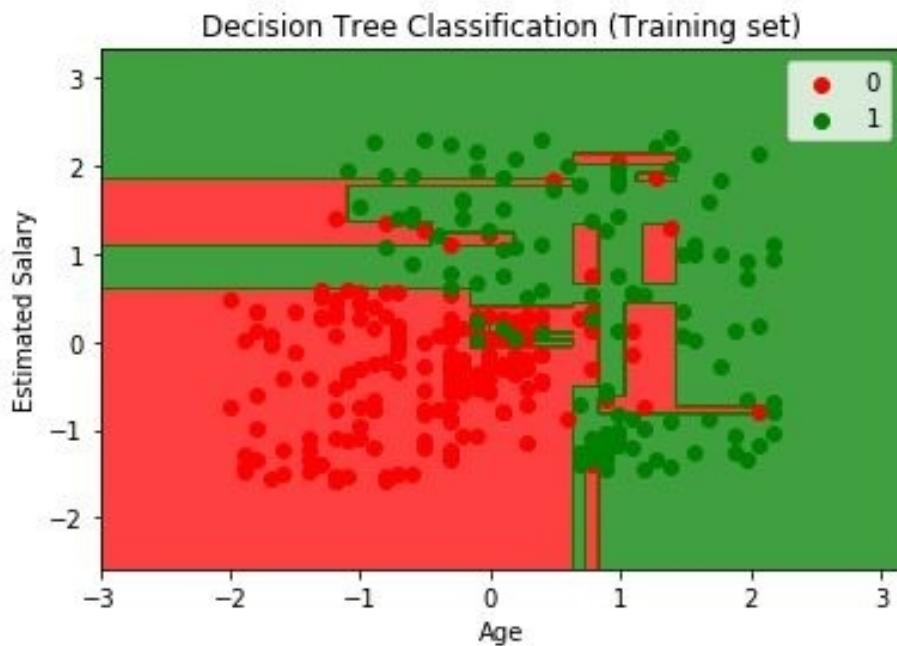
X2.ravel()].T).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
    c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
# Visualize the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),
np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
step =
0.01))
plt.contourf(X1,
X2,
classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
```

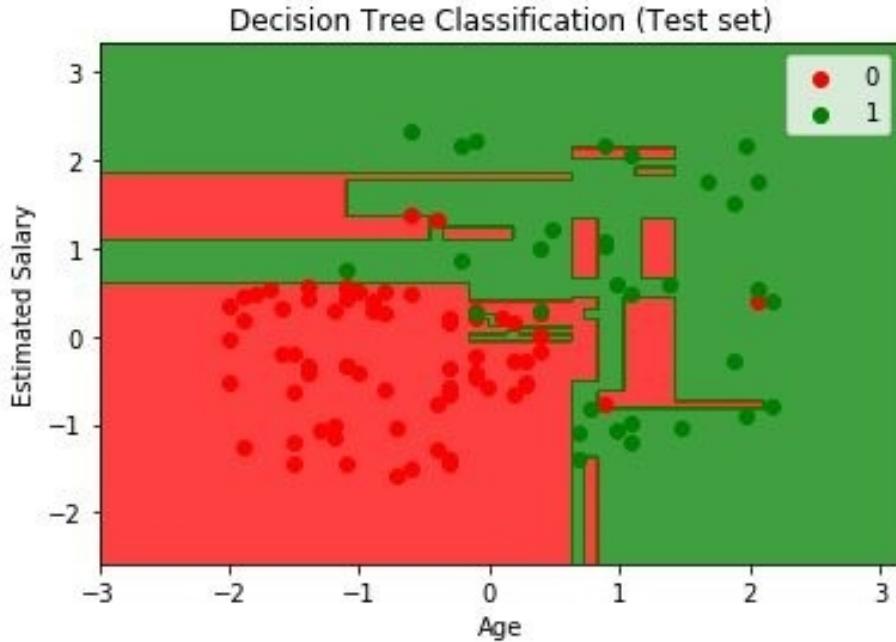
```

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show() The most important difference is in this block of code: from
sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)

```

The following will appear in your Jupyter notebook when you run this code:





As you can see, there is a huge difference between this and K-Nearest Neighbors and logistic regression. The latter ones show two boundaries, but the decision tree shows points that are not in the main red region but are inside smaller red regions. Because of this, the model could capture otherwise impossible data points.

Random Forest Classification

If you recall, a random forest in regression is made up of multiple decision trees. The same applies to a random forest in classification, with multiple trees being used and the results being averaged:

```
# Random Forest Classification

# Import the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline

# Import the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
```

```

X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Split the dataset into the Training set and Test set from
sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fit Random Forest Classification to the Training set from
sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators = 10, criterion =
'entropy', random_state = 0) classifier.fit(X_train, y_train)

# Predict the Test set results
y_pred = classifier.predict(X_test)

# Make the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Visualize the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step =
0.01),
np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
step = 0.01)) plt.contourf(X1, X2,

```

```

classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap =
ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):

    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
c = ListedColormap(('red', 'green'))(i), label = j)

    plt.title('Random Forest Classification (Training set)')
    plt.xlabel('Age')
    plt.ylabel('Estimated Salary')
    plt.legend()
    plt.show()

# Visualize the Test set results

from matplotlib.colors import ListedColormap

X_set, y_set = X_test, y_test

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step =
0.01),

np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
step = 0.01)) plt.contourf(X1, X2,
classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap =
ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

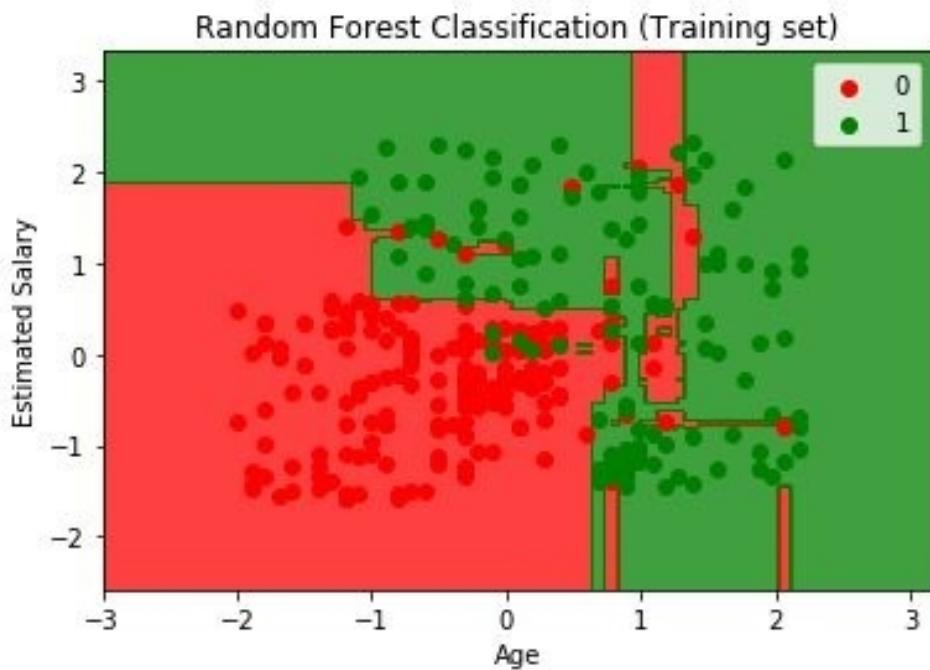
for i, j in enumerate(np.unique(y_set)):

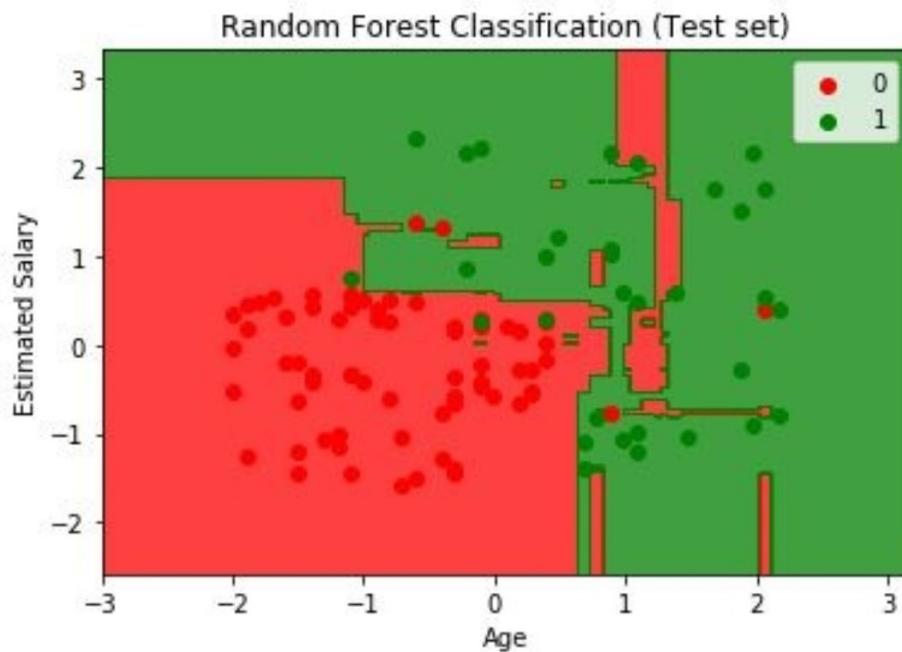
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],

```

```
c = ListedColormap(['red', 'green'))(i), label = j)
plt.title('Random Forest Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

Run the code and you should see this:





Note how similar the decision tree and random forest are – they both take an approach of breaking the dataset into smaller subsets.

The difference lies in the random forest using randomness and averaging the results of the decision trees to create a model that produces more accurate results.

In the next chapter, we will look at clustering.

Chapter 6: A Look at Clustering

Previously, we discussed regression and classification, two supervised learning methods, and we also talked about labeled data and how a machine can learn from it. In the early days of machine learning and data science, we had the right answers, and our job was to find out how to get to the answers and then apply that to new data.

This chapter will be a bit different because we are going to start with unsupervised learning. Here, we have no right answers or labels. We have all the input data, but we don't have an output, and the machine is not supervised when it is learning from the data. The algorithm is left to learn what it can from the data.

This is particularly the case with clustering, where we want to see clusters or aggregates in the data.

Clustering Goals and Uses

Clustering falls under unsupervised learning, where no data is labeled and, in many cases, there may not be a right answer. Why? Because we didn't start with any correct answers as we do in supervised learning. We have our dataset and know what we want to achieve – to see organic data clusters.

Clustering isn't about predicting outcomes. It's about looking for structures that may be visible in the data. That means the dataset is being split into groups where the members of each group share similarities.

For example, e-commerce customers may belong to a group depending on their income or their spending. If sufficient data points are there, we may see aggregates.

To start with, the data points won't show much of a pattern. But, once the clustering algorithm is applied, the data will begin to make some sense because we can easily see clusters. Besides finding natural clusters, the clustering algorithm may also show outliers for anomaly detection – we'll discuss this a bit later.

Clustering is constantly being applied in certain fields, such as earthquake studies, biology, categorizing products, manufacturing, etc. However, there are no hard and fast rules regarding how many clusters there are or which cluster a data point should be in. It's down to our project goal to determine this, and this is also where domain expertise comes into play.

With a solid knowledge of a domain, we can analyze the data properly – it doesn't matter how advanced your techniques or tools are. Objective and context are still two of the most important factors in making sense of the data.

K-Means Clustering

The K-Means algorithm is one way you can use clustering to analyze data. It's simple and works by splitting the objects into the number of clusters we want – k-clusters. You can set any number you want but try to use a number just enough to ensure your work means something. Here's an example.

We have a dataset called `Mall_Customers.csv`, containing details of mall customers, including age, income, spending, gender, etc. The higher the spending score is, the higher their spending level.

First, we need the right libraries imported:

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

Then the data is imported and we take a look:

dataset =

```
pd.read_csv('Mall_Customers.csv') dataset.head(10)
```

	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72

Here, we want the customers grouped by annual income and spending score.

X = dataset.iloc[:, [3, 4]].values

We want to show the clusters, enabling the marketing department to build their strategies. We could divide the customers into five groups:

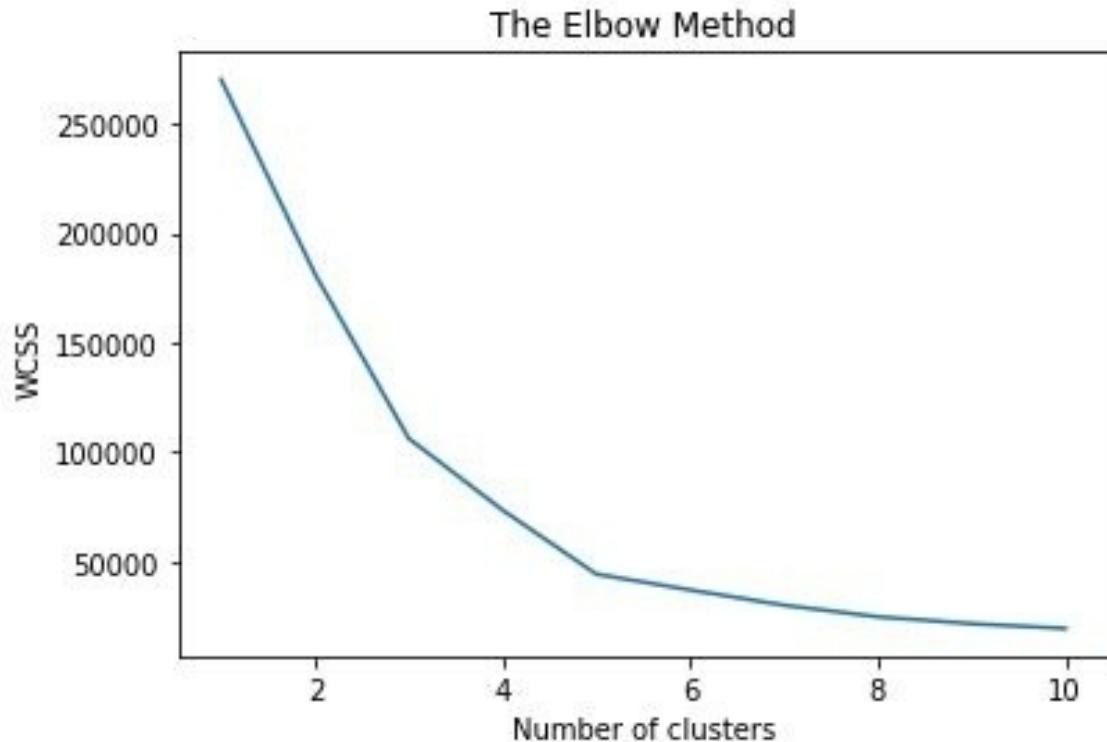
1. Medium Annual Income, Medium Spending Score

2. High Annual Income, Low Spending Score
3. Low Annual Income, Low Spending Score
4. Low Annual Income, High Spending Score
5. High Annual Income, High Spending Score

Pay close attention to group two – if many customers are in this group, the mall can potentially have a massive opportunity. These customers may have a large income, but they aren't spending it at the mall. With enough numbers in that group, the marketing department could devise a plan to get them to spend their cash at the mall and not elsewhere.

While you can pick the number of clusters you want, there is a way to determine the optimal number – we use the Elbow method with WCSS (within-cluster sums of squares. This is the code:

```
fro m sklearn.cluster import KMeans  
wcss = []  
for i in range(1, 11):  
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)  
    kmeans.fit(X)  
    wcss.append(kmeans.inertia_)  
plt.plot(range(1, 11), wcss)  
plt.title('The Elbow Method')  
plt.xlabel('Number of clusters')  
plt.ylabel('WCSS')  
plt.show()
```



The elbow is pointing at five as the optimal cluster number, and this was also the number of groups we specified earlier.

Once the optimal number is determined, K-Means can be applied to our dataset and the data visualization performed.

```

kmeans
= KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)

plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

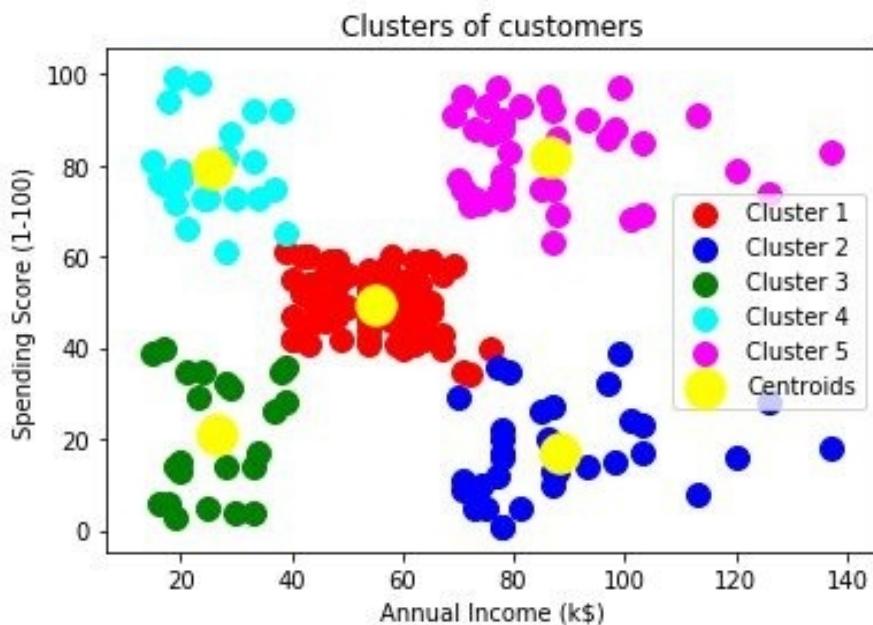
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

```

```

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
           s = 300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()

```



As you can see, we have five clusters. Cluster or group two has enough points for the marketing department to focus their attention on it.

Note the yellow points. These are called centroids, and they are integral to how K-Means works. Random points are initially placed until they converge to a minimum – this means the sum of distances has been minimized.

This all depends on what the results will be applied to. We don't have to go with five – n-clusters can be set to anything. The elbow method helps us decide the optimal number, but it's all down to the project's goal and whether the results are good enough.

Anomaly Detection

Clustering isn't just about showing the natural clusters. We can also use it to show points that don't belong, and this is where anomaly detection comes in. Anomaly detection is important because large deviations can cause trouble. Is there suspicion surrounding a login? Is a credit card transaction genuine or fraudulent? Are a tank's pressure and temperature levels consistent?

Data visualization allows us to see the outliers immediately and evaluate if they present a threat. We can also use the mean and standard deviation to assess the outliers – if data points deviate from the mean by a standard deviation, it may be an anomaly.

Again, domain expertise comes into play. Are the consequences of an anomaly serious? For example, thousands of transactions may be happening in an online store daily. If our anomaly detection is tight, quite a few transactions could be rejected, resulting in a loss of sales. Conversely, if too much freedom is given, more transactions would be approved. This could lead to complaints and potential loss of customers.

Let's move on to reinforcement learning.

Chapter 7: What Is Reinforcement Learning?

In the last few chapters, we have focused on deriving insights from previous information. However, if machine learning and data science are to be successful, our algorithms need to work on real-time problems. For example, we need systems that learn on current data and adjust as the data does to ensure the maximum reward.

That's where reinforcement learning steps in. It is about reinforcing behavior over time, rewarding correct behavior, and punishing incorrect behavior. In the recent past, reinforcement learning was used to beat the world Go game champions and play Atari video games. Because reinforcement is used to train the system, it could reach its goals or maximize its rewards.

A simple example of this online ad CTR (click-through rate.) Let's say you have ten ads, all saying much the same thing but in different words and/or designs. You want to know which one gets the highest CTR – the more clicks an ad gets, the more potential customers for the business.

To maximize the CTR, you could make the adjustments while the ads are running. You don't need to run through your entire ad budget to know which ones work and which don't. Instead, monitor them while they run and make early adjustments so that, later, prospective customers will only see the high-performing ads.

This is much like a probability theory problem about a multi-armed bandit. Let's assume that you have a limited advertising budget and ten variants of your ad.

How would your resources be allocated among the ads to ensure the best CTR?

First, you would try each ad on its own. If the first one performed well, you would use it for the rest of your ad campaign, rather than wasting money on others. However, there is a catch to this. The first ad might perform so well at the start that you keep on using it. But what if the second ad starts performing really well and overtakes the first one in results? You wouldn't know because you already exploited the first ad's performance.

Many projects contain tradeoffs which is why you should set your performance targets at the start rather than wallowing in a load of what-ifs later down the line. Even the sophisticated algorithms and techniques have constraints and tradeoffs - it's all about finding what works for your given project.

Reinforcement Learning Compared with Supervised and Unsupervised Learning

By definition, reinforcement learning doesn't fall under supervised or unsupervised learning. Supervised learning uses structured and labeled data to learn from, while unsupervised uses unstructured and unlabeled data to derive insights. The biggest differences are that reinforcement learning can update itself in real-time, maximize its reward, and learn from user interaction. By contrast, supervised and unsupervised learning use historic data to learn from.

The line between the three types of learning is very fine because they all focus on optimization, and all three are useful in business and scientific settings.

How to Apply Reinforcement Learning

Reinforcement learning is useful for businesses because it can be used to optimize CTR. How do we maximize the CTR for a specific headline? Often, a news story will be limited in its lifespan in terms of popularity and relevance.

Given that time is a limited resource, how can we show the headline that performs the best straight away?

This works the same for online ad CTR. Our ad budget is limited, and we want the most out of it. Let's have a look at an example using the ads_CTR_Optimization.csv dataset. First, we need to import our libraries for working on the data and doing the data visualization:

```
import matplotlib.pyplot as plt  
import pandas as pd  
%matplotlib inline #so plots can show in our Jupyter Notebook
```

We then import the dataset and take a peek

```
dataset = pd.read_csv('Ads_CTR_Optimisation.csv') dataset.head(10)
```

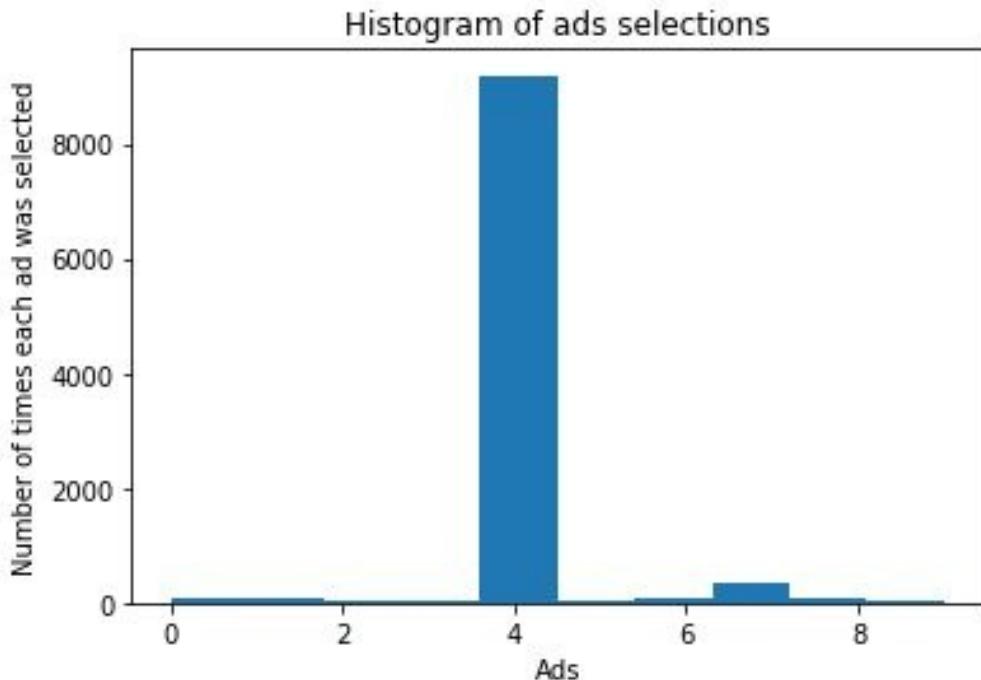
	Ad 1	Ad 2	Ad 3	Ad 4	Ad 5	Ad 6	Ad 7	Ad 8	Ad 9	Ad 10
0	1	0	0	0	1	0	0	0	1	0
1	0	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	0	0
5	1	1	0	0	0	0	0	0	0	0
6	0	0	0	1	0	0	0	0	0	0
7	1	1	0	0	1	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	0	0

In every round, we see the ads displayed. We can also see an indication of which were clicked – 0 indicates no, 1 indicates yes. As we said earlier, we want to explore these first, choose the best one and exploit it.

A good way of doing this is to use Thompson sampling. This is used to address the dilemma of exploration-exploitation, i.e., finding that balance. It does it by sampling the potentially promising actions while discarding those that look like underperformers. This algorithm uses probabilities, and we can use the following code to express this:

```
import random  
N = 10000  
d = 10  
ads_selected = []  
numbers_of_rewards_1 = [0] * d  
numbers_of_rewards_0 = [0] * d  
total_reward = 0
```

```
for n in range(0, N):
    ad = 0
    max_random = 0
    for i in range(0, d):
        random_beta
        =
        random.betavariate(numbers_of_rewards_1[i]
                            +
                            1,
                            numbers_of_rewards_0[i] + 1)
        if random_beta > max_random:
            max_random = random_beta
            ad = i
        ads_selected.append(ad)
        reward = dataset.values[n, ad]
        if reward == 1:
            numbers_of_rewards_1[ad] = numbers_of_rewards_1[ad] + 1
        else:
            numbers_of_rewards_0[ad] = numbers_of_rewards_0[ad] + 1
        total_reward = total_reward + reward
    When we run and the code and visualize:
    plt.hist(ads_selected)
    plt.title('Histogram of ads selections')
    plt.xlabel('Ads')
    plt.ylabel('Number of times each ad was selected')
    plt.show()
```



Implementing Thompson sampling is not always easy. It is one of the more interesting algorithms, widely used in optimizing online ads, sorting products, recommending news articles, and other applications.

There are a lot of other interesting heuristics and algorithms, like the Upper Confidence Bound. The goal is the same – earn at the same time as learning. Instead of making the analysis later, the algorithm performs in real-time, making the necessary adjustments as needed. We want to maximize the reward while, at the same time, balancing the exploration-exploitation tradeoff. This means we want the immediate performance maximized to ensure future performance is improved. Rather than going deep into that here, I would point you in the direction of the following tutorial, which I would urge you to take the time to look at.

https://web.stanford.edu/~bvr/pubs/TS_Tutorial.pdf

Chapter 8: The Artificial Neural Network

Humans find it easy to recognize digits or objects. We can also determine what a piece of text or a sentence means with little effort. However, computers are a whole new ball game. What we find trivial, what we do automatically, could be a huge task for an algorithm or a computer.

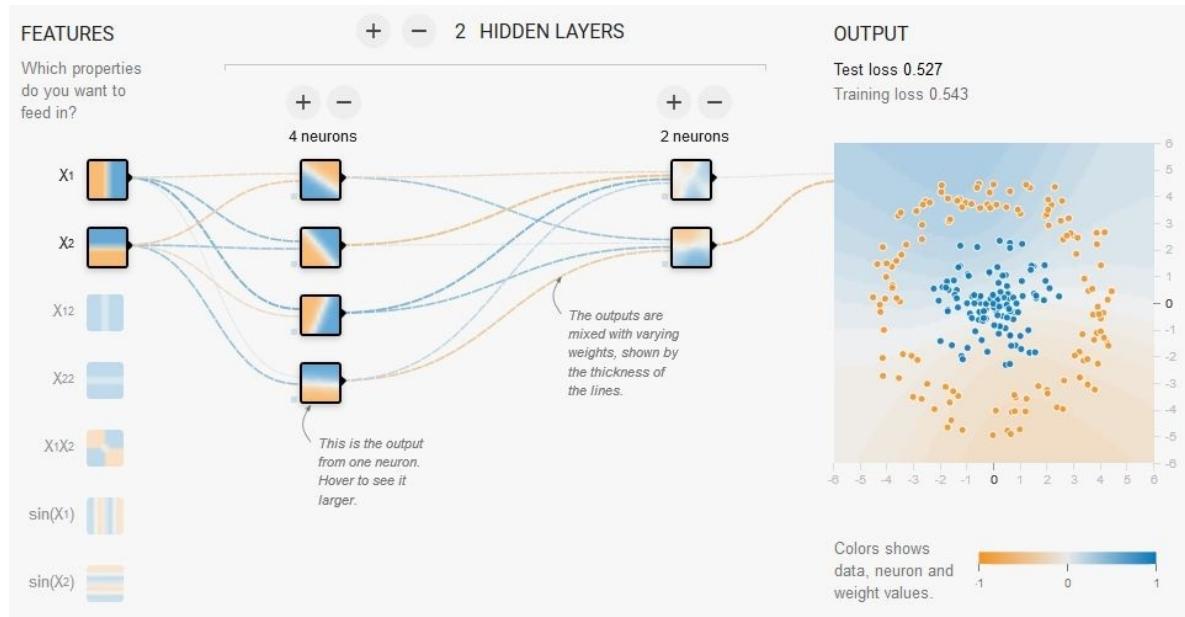
By contrast, where humans struggle to do complex or long calculations, a computer can do it with ease. By that, you can see that human and computer capabilities are complementary, opposite to one another.

The next natural step for computers is to imitate human capabilities, even to surpass them. At first glance, it would seem that the goal is to set computers to do what humans do, and, in the future, you may struggle to determine if you are talking to a human or a computer.

Imitating the Human Brain

One of the best ways to accomplish this is using artificial neural networks, or ANNs as they are popularly known. These are loosely based on how the human brain and its neurons work. The basic model, the one we use now, shows the brain working by neurons receiving signals, processing them, and sending new signals. In the process, they may connect with another neuron, get their inputs from senses or provide an output. It isn't a 100% accurate picture, but this kind of model is useful for many different applications.

In an artificial neural network, neurons are placed in one or more layers and used to send and receive signals. Here's an illustration from the TensorFlow Playground to make this a little clearer:



Here, you can see that the inputs or features are the starting point. These are then connected to two layers (hidden) of neurons. Lastly, we get that output, where the data was iteratively processed to provide a generalization or useful model.

In many ways, artificial neural networks are used in much the same way that supervised learning works. ANNs are often fed huge amounts of training data and create a model or system that lets them learn from the sample data. During the learning phase, the ANN will infer some rules that allow it to recognize data, such as images, audio, or text.

As you already know, recognition accuracy is heavily dependent on how much data there is and its quality. Remember – Garbage In, Garbage Out.

An artificial neural network will learn from the data fed to it. We can still improve the performance and accuracy, and we can do it in ways that don't always revolve around improving the data quantity and quality. Those ways include changes to the learning rate, feature selection, and regularization.

Constraints and Potentials

The theory behind the ANN is not new but has recently re-emerged very strongly, causing people to talk about it, whether they understand it or not. This came about because there is so much more data available now, not to mention an exponential increase in computing power. It wasn't always

considered practical to build and implement artificial neural networks in the early days, especially in terms of resources and time.

Because of the increase in data and computing power, ANNs are easier than ever to implement, and they can be done just about anywhere, even on your everyday laptop or desktop computer.

ANNs are already hard at work behind the scenes, providing you the right search results, products you might be interested in purchasing, ads you are likely to click on, etc. They can also be used to recognize video, image, and audio content.

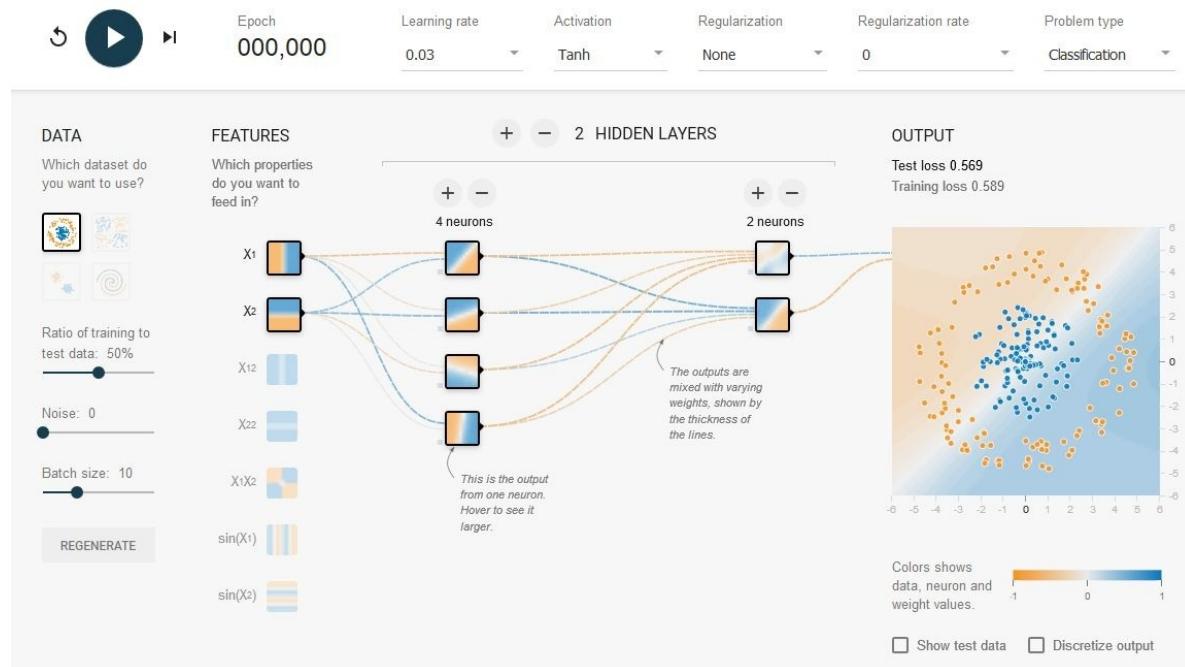
Right now, we are literally scratching the surface of what ANNs can do for us, and there is still so much potential. Think about Michael Faraday; when he performed an experiment with electricity, nobody knew where that would lead. At the time, Faraday told the UK prime minister he would eventually be able to tax electricity. Today, virtually our whole lives revolve around or depend on electricity, be it directly or indirectly.

The same thing may apply to ANNs and deep learning, which is an ANN-focused subfield of machine learning.

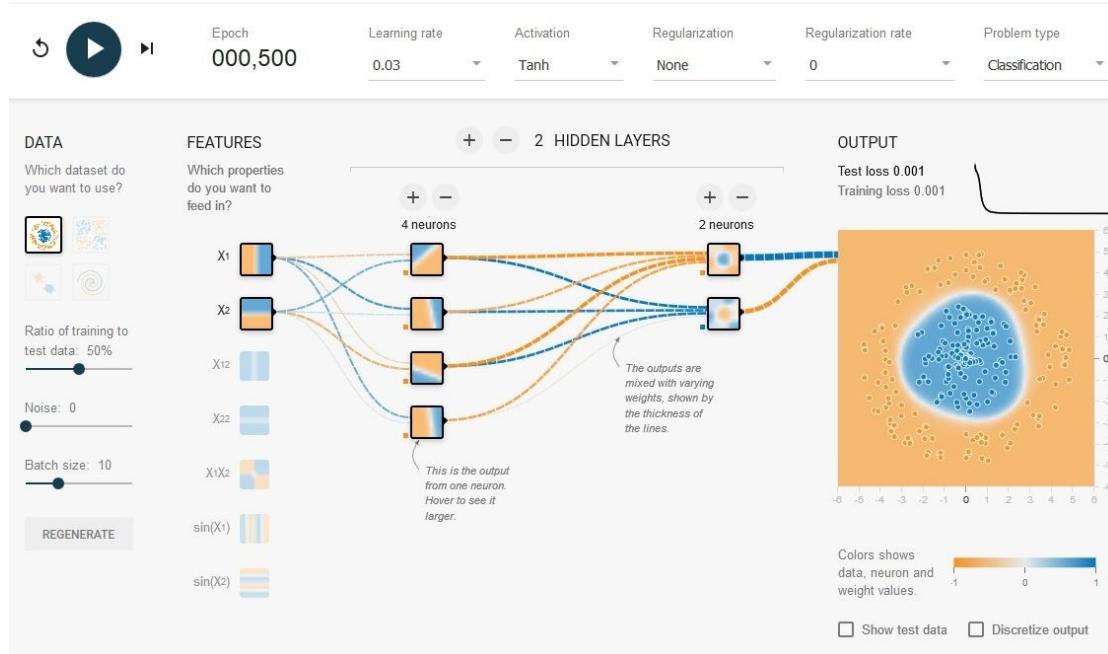
Have a look at the following example:

TensorFlow Playground lets us see how all this works, so have a look on their website, <https://playground.tensorflow.org>, at all the different terms you see – Regularization, Features, Activation, Learning Rate, Hidden Layers, and so on.

To start with, it will look like the following:



If you then click the Play button at the top left corner, you will see an animation. Make sure you look at the far right to see the output. After a while, it will look like this:

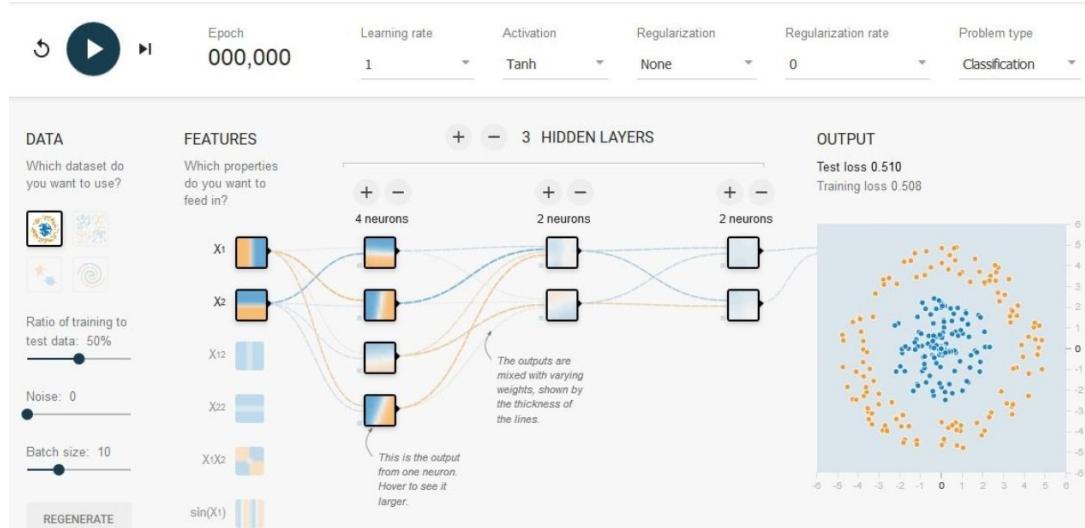


You can clearly see the connections between the inputs (features), the hidden layers, and the output. You can also see the blue region on the output. This may well be a classification task, where the orange is in group B while the blue is in group A.

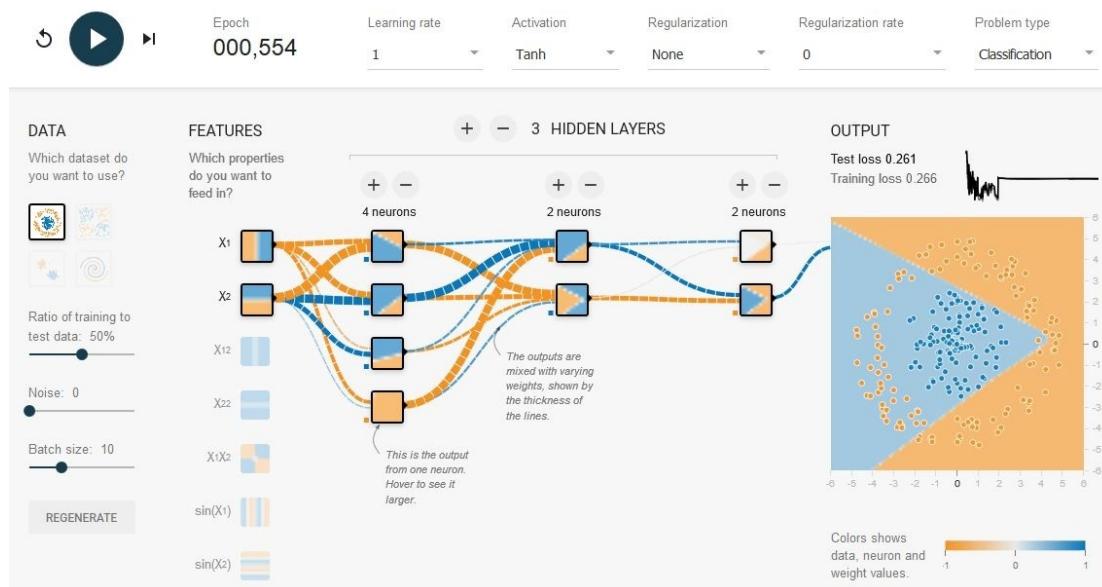
Watch the ANN run, and you will see the division between the two groups becomes clearer. This is because of the system's continuous learning from the training data – the more solid that learning becomes, the more accurate the classification.

The TensorFlow Playground gives you an easy way of learning how neural networks operate, showing you a visualization of the features, the hidden layers, and the output. It even allows you to tweak things a little, such as altering the learning rate, the training to test data ratio, and how many hidden layers there are.

For example, the number of hidden layers could be set to 3 and the learning ratio to 1 – that would look like this:



Click the Play button and leave it to run – for some reason, the image will stay like this:



Again, look at the output. As you can see, the classification looks worse. Rather than most of the orange points being enclosed in the orange area, we can see many misses, with orange points showing in the blue area instead. This is because of those changes we made.

Pay attention to the Output. Notice that the Classification seems worse. Instead of enclosing most of the yellow points under the Yellow region, there are many misses (many yellow points fall under the Blue region instead). This occurred because of the change in parameters we've done.

For example, changing the learning rate will significantly affect the accuracy and convergence. Too low, and convergence will take much longer while too high, and convergence may be overshot altogether.

Convergence can be achieved in good time in a few ways, for example, the right learning rate, the right number of hidden layers, a change in the number of features, regularization being applied, etc. But it doesn't always make sense to over-optimize everything. Right at the start, a clear objective should be set, and you should stick to it. If an interesting opportunity arises along the way, you can also tune the parameters a little and improve your model's performance.

The sample code below will give you a simple idea of how ANNs look in Python:

```
X=np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ] ) y = np.array([[0,1,1,0]]).T  
syn0 = 2*np.random.random((3,4)) - 1  
syn1 = 2*np.random.random((4,1)) - 1  
for j in xrange(60000):  
    l1 = 1/(1+np.exp(-(np.dot(X,syn0))))  
    l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))  
    l2_delta = (y - l2)*(l2*(1-l2))  
    l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))  
    syn1 += l1.T.dot(l2_delta)  
    syn0 += X.T.dot(l1_delta)
```

It is a simple example – real-world ANNs are a lot longer and more complex, but it is becoming far easier to work with them – even those not well-versed in the technical stuff can use them.

Conclusion

Thank you for taking the time to read my guide. It is by no means an in-depth guide into using Python for data science but it does cover the most important subjects, providing you with practical examples on how they work.

Data science is a complex subject and, to understand how it works, you need to have prior knowledge of programming in the right language – Python, in this case. I have not given you the basics of Python here because you already need to know that to follow this guide, and, to be honest, it's a whole new book.

Data Science is an up-and-coming area, fast-growing and providing tons of opportunities for people with the right skills. Today, there are more jobs than ever in this exciting field and the future holds plenty of promise.

It isn't going anywhere and, the more data the world produces, the more in-depth data science will become, not to mention more important so take the right steps now and join a revolution that shows no signs of slowing down.

References

- Carter, Daniel Smilkov and Shan. “Tensorflow — Neural Network Playground.” *Playground.tensorflow.org* , playground.tensorflow.org/. Accessed 2 June 2021.
- “Datasets - Keras Documentation.” *Keras.io* , keras.io/datasets/.
- “Datasets | Kaggle.” *Kaggle.com* , 2019, www.kaggle.com/datasets.
- “Deep Learning | Coursera.” *Coursera* , 2019, www.coursera.org/specializations/deep-learning.
- <https://www.facebook.com/data36>. “Learn Python for Data Science - from Scratch (14 Articles).” *Data36* , 2017, data36.com/learn-python-for-data-science-from-scratch/. Accessed 1 Dec. 2019.
- “Learn Python - Free Interactive Python Tutorial.” *Learnpython.org* , 2019, www.learnpython.org/.
- Matplotlib. “Matplotlib: Python Plotting — Matplotlib 3.1.1 Documentation.” *Matplotlib.org* , 2012, matplotlib.org/.
- “NumPy — NumPy.” *Numpy.org* , 2019, www.numpy.org/.
- Pandas. “Python Data Analysis Library — Pandas: Python Data Analysis Library.” *Pydata.org* , 2018, pandas.pydata.org/.
- “Python for Beginners.” *Python.org* , Python.org, 2019, www.python.org/about/gettingstarted/.
- “PyTorch.” *Pytorch.org* , 2019, pytorch.org/.
- “Virtualenv — Virtualenv 20.4.7 Documentation.” *Virtualenv.pypa.io* , virtualenv.pypa.io/en/stable/. Accessed 2 June 2021.

Sources of image and code in this guide:

<https://matplotlib.org/2.0.2/gallery.html>

http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-glr-auto-examples-linear-model-plot-ols-py

http://chem-eng.utoronto.ca/~datamining/dmc/decision_tree_reg.htm