AI Sciences

Machine Learning from Scratch

AI SCIENCES

Alain Kaufmann

MACHINE LEARNING FROM SCRATCH

Alain Kaufmann

AI SCIENCES

Preface

"In God we trust. All others must bring data."

~ W. Edwards Deming, statistician

The overall aim of this book is to give you an application of machine learning techniques with python.

Machine learning is a field of Artificial Intelligence that uses algorithms to learn from data and make predictions. This means that we can feed data into an algorithm, and use it to make predictions about what might happen in the future.

This book is a practical guide through the basic principles of machine learning, and how to get started with machine learning using Python based on libraries that make machine learning easy to get started with.

Book Objectives

If you are interested in learning more about machine learning with practical examples and application with python, then this book is exactly what you need.

This book will help you:

- Have an appreciation for machine learning and an understanding of their fundamental principles.
- Have an elementary grasp of machine learning concepts and algorithms.
- Have achieve a technical background in machine learning and also deep learning

Who Should Read This?

- Anyone curious about machine learning but with zero programming knowledge
- People who want to demystify machine learning (it's not magic and it's probably not the end of the world)
- Technical people who want to quickly gain knowledge in machine learning

Is this book for me?

If you want to smash machine learning algorithms with Python, this book is for you. Little programming experience is required. If you already wrote a few lines of code and recognize basic programming statements, you'll be OK.

Edited by Davies Company
Ebook Converted and Cover by Pixels Studio
Publised by AI Sciences LLC

ISBN-13: 978-1643160733

To my father and mother: Jim and Leticia, I love you both so much and God bless you.

AI SCIENCES

Author Biography

Alain is a Data Scientist and a member of the R Foundation. He built data science's packages and participated in the development of some software. He also have great interest to other data science languages such as Python and SAS. Now, he is a writer and educator. He also promotes some data sciences software applications.

Table of Contents

Introduction

To better understand Machine Learning requires *a bit of knowledge* about statistics, linear algebra, programming and computer science. We'll explain the most important concepts as we go along. We'll also dive into Python programming so you'll be familiar with the *behind-the-scenes* of machine learning. We'll introduce you to the most popular machine learning tools you can start using today.

You'll learn how to analyze and visualize data besides using Microsoft Excel. You'll also understand what the following block of code means:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

dataset = pd.read_csv( 'your_data_file.csv' )

dataset.describe()
```

We'll take it slow and explain the intuition behind the code and the algorithms.

After reading this book, you'll be in a better position to create informed opinions about the opportunities (and threats) machine learning and artificial intelligence will bring. You'll also be able to apply basic machine learning algorithms you can apply to your professional or personal projects.

What is Machine Learning & Why Big Companies are investing

Let's first define machine learning. Basically, it's about allowing computer systems to "learn" from data. The data can be in the form of text, numbers, images, games, audio, or video. After applying machine learning techniques and algorithms, these are the most common results:

Machine Learning on Text

- Determine the sentiment and attitude (positive, negative, or neutral) of the social media users based on what they wrote (e.g. useful in movie ratings)
- Categorize news articles based on their theme or topic
- Generate new headlines based on news content

Machine Learning on Numbers

- Better forecast future demand so companies can promptly stock more goods
- Predict a loan applicant's probability to pay the full loan based from data

Images & machine learning

- Optical Character Recognition (OCR) to identify numbers and text embedded on an image
- Face detection & Face ID

Beating world-class players with machine learning

- AlphaGo (a computer program) was able to defeat Lee Sedol (South Korean professional player) in the game of Go

Audio & video

- Virtual assistants you can make enquiries or give commands to
- "Learning to see" what's in the videos
- Language translation

The possibilities could be endless, which is why huge companies (especially Google, Amazon, Microsoft, Facebook, Baidu, Alibaba) are investing billions in the technology. They are also busy buying and acquiring companies left and right to strengthen their team and get a competitive advantage.

Machine learning is now optimizing business operations and profits in various industries. More importantly, machine learning is now also being used to change people's lives especially in the areas of healthcare (prediction of disease progression, more accurate diagnosis).

Real-World Examples You Might Not Be Aware Of

We've already mentioned several examples above. But just today you might have already taken advantage of machine learning without being aware of it.

Spam or not spam? Computer systems learn from past labeling of users which emails are spam. Machine learning allows professionals to know why an email is a spam (e.g. does it contain words such as "free" or "money" or does the email contain ALL CAPS or a questionable attachment?).

Amazon and other ecommerce sites provide product recommendations through machine learning. The recommendations you see might be based from your browsing history, past purchases and/or purchases by users similar to you.

Google uses a machine learning system to sort their search results. The system "learns" from how users interact with the search results and after they click. For example, if many users are staying for a while in a page, the search ranking for that page might improve as a result.

Facebook and other social media channels continuously tweak their algorithms so users will see the most relevant content in their feed. Social media channels are also able to recommend people who are most likely to be connected to you.

Why it's Only Becoming Popular Recently

Given the dramatic result and huge potential of machine learning, why it has only been recently when it became really popular?

First, there were serious constraints in computational and processing power. Second, back then lack of data (or access difficulty) was also a problem. But now we have massive computational power coupled with billions of gigabytes of data.

That's why companies are hiring data scientists and machine learning engineers to tap into that potential. In addition, more and more people are studying machine learning to take advantage of the opportunity. In fact, due to lack of available talent companies are getting from a pool of scientists and mathematicians (with PhDs in Physics or Statistics).

Do You Really Need to Know Statistics & Python?

Speaking of mathematics, do you really have to master statistics, linear algebra and calculus to fully understand machine learning?

The short answer is no. Although it's not essential to have a solid mathematical background, it will still give you an edge. All things being equal, the one with strong mathematical skills will likely be hired.

Thankfully, you can study all the math you want after reading this book. This resource is already enough to give you a solid understanding on machine learning and apply it to fundamental problems.

What about Python and programming? Knowledge and skills on these will allow you to better understand the limitations and possibilities of machine learning. Programming is also a way of thinking which could give you new perspectives about the problems around you.

To start applying machine learning techniques and algorithms, you don't have to master Python and programming. Here in this book we'll discuss the blocks of code most commonly used in the machine learning field. In addition, there are off-the-shelf tools you can already use to make the job a lot easier (shorter time period and fewer lines of code so you can focus on the actual analysis).

In the next chapter we'll introduce those tools and discuss how Python is used for machine learning. We'll also explain the intuition behind the code and how the algorithms are being used in real-world examples.

Let's start.

Using Python for Machine Learning

Why Choose Python?

Python is often said to be a simplified, intuitive and clear programming language. As an example, let's compare how to print "Hello world!" on a screen:
For C programming language:

```
#include <stdio.h>
int main(void)
{
  printf("Hello world!¥n");
  return 0;
}
```

In contrast, this is Python:

```
print( "Hello world!" )
```

It's an extreme example just to illustrate the possible differences (just one line of Python code vs 6 lines in C language). Also see in the above example that most of the lines in C are far from intuitive.
Python's simplicity allows beginners to quickly create something and understand the syntax. The minimal structure also allows the scientific, research communities to quickly experiment, and produce a prototype.
After all, we want to quickly translate our ideas into something that the computer will understand and execute (the essence of programming). Python makes that job easier because of its simplicity and clarity.
In addition, Python has an active online community and more schools are using it as introductory course in computer science. This means there are many online tutorials available. Whenever you're stuck on a problem relating to Python, it's very likely someone else has already solved it for you.

In the **Bonus Chapter,** you can use it as quick reference if you're a complete beginner in Python and programming. We discuss there the required setup and some basics. This is to make you more comfortable as you read along the rest of this book.

Overview & Tools

Aside from simplicity and information availability, many machine learning tools are using Python (perhaps so more users will use that particular tool). Also, there are enough reliable libraries (precompiled routines) you can use to make the job easier.

Those libraries can also be thought of as “pre-written codes” so that you won’t have to write everything from scratch. That’s because many procedures are actually repetitive. Experienced software engineers and contributors instead compiled them so anyone can use the library and avoid too much repetition. This means lesser time programming and a lot fewer lines of code.
Overall, this makes the learning curve a bit smoother. It’s far from perfect and being too easy. Good thing is it would save you a lot of time upfront and in the long term. You won’t be an expert in Python within a few days but you could still gain reasonable abilities within that same timeframe. With Python you can also quickly start applying machine learning. You can always dig deeper into Python if you’re aiming for expertise.

Install Anaconda

So how do you get started with Python? One of the most recommended ways is to download and install Anaconda (https://www.anaconda.com/download/).

Why? Almost all the data science and machine learning packages you’ll need are already there. With a few lines of code you can already apply basic machine learning algorithms and analyze some data.
If you require some advanced stuff, you can also dive into TensorFlow (which we’ll also discuss later on). For now, let’s focus on quickly building the foundations and learning the principles of machine learning. We’ll first give an overview about the most popular tools relating to Python.

Scikit-learn, numpy, pandas, matplotlib

These are some of the most common Python libraries being used by data scientists and even machine learning engineers.

Scikit-learn is free and you can use it to do the following:

- Classification (example: spam or not spam?)
- Regression (predicting values)
- Clustering (group similar objects into sets)
- Dimensionality Reduction (reduce number of variables for easier visualization)
- Model Selection (comparing and validating models)
- Data Preprocessing (initially use scikit-learn to make data “workable” and then apply advanced machine learning techniques)

Numpy is mostly used for scientific computing for Python. It makes computation more efficient and more effective (in contrast with using regular Python which can take several lines of code). Aside from numpy, pandas are also used for computation and data manipulation. Pandas is derived from the term “panel data” which is associated with time-series and cross-sectional data.

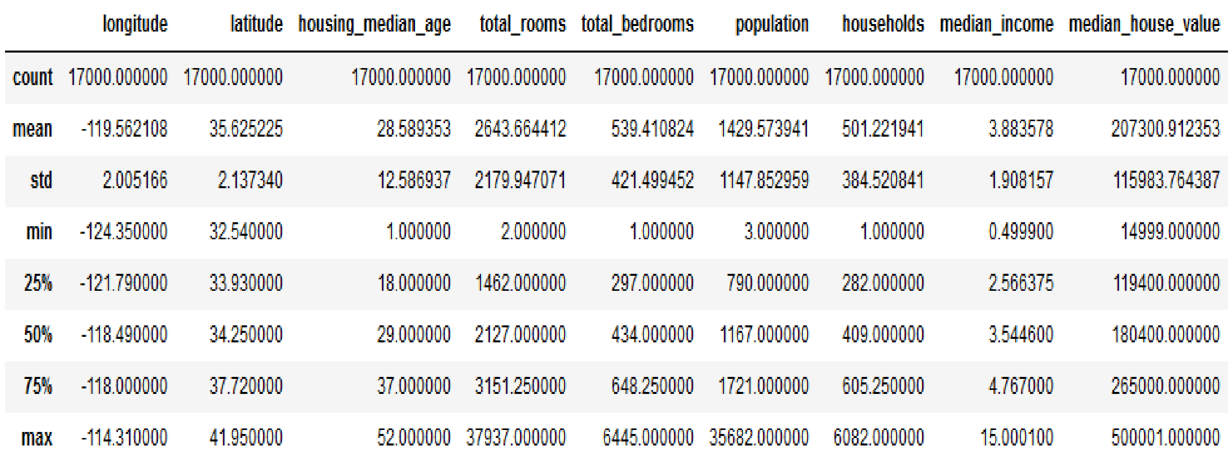
Anyway, pandas are useful in data preprocessing and manipulation. Most commonly, pandas are used to read csv (comma-separated values which is a convenient data file format) files which goes something like this:

```
import pandas as pd

dataset = pd.read_csv( ‘your_file.csv’ )
```

First, we import the package or module so our system understands that we’ll use it. Then we instruct the system to read the file. This is often required so you can further manipulate and analyze the data.

Next is we can then perform additional things such as seeing the mean, standard deviation, and minimum and maximum values in a given dataset. We can accomplish this by:

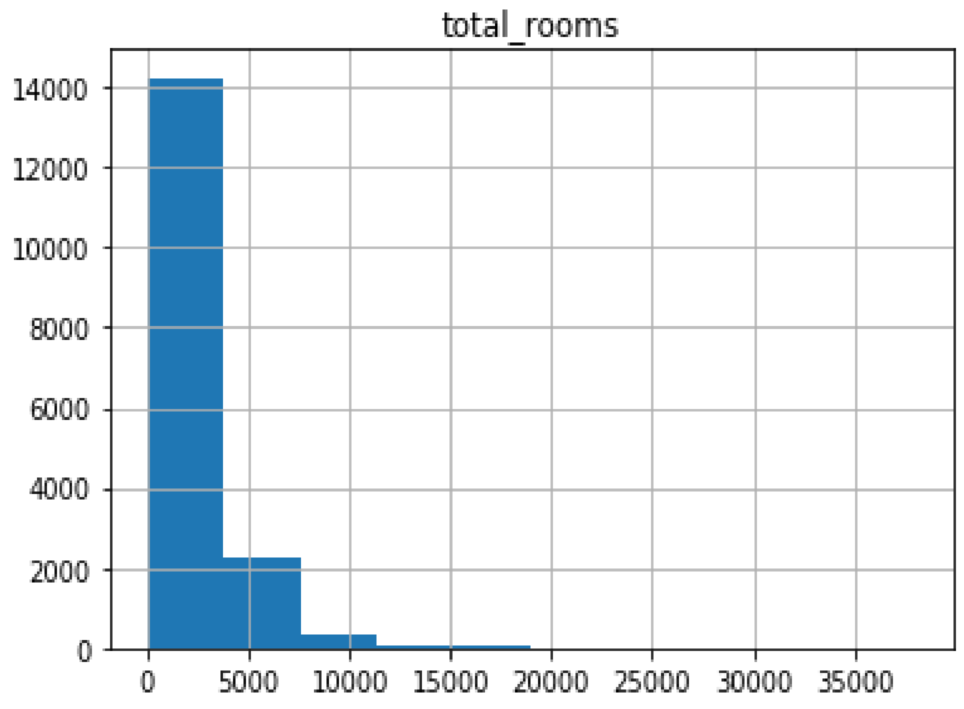
```
dataset.describe()
```



	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000
mean	-119.562108	35.625225	28.589353	2643.664412	539.410824	1429.573941	501.221941	3.883578	207300.912353
std	2.005166	2.137340	12.586937	2179.947071	421.499452	1147.852959	384.520841	1.908157	115983.764387
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.790000	33.930000	18.000000	1462.000000	297.000000	790.000000	282.000000	2.566375	119400.000000
50%	-118.490000	34.250000	29.000000	2127.000000	434.000000	1167.000000	409.000000	3.544600	180400.000000
75%	-118.000000	37.720000	37.000000	3151.250000	648.250000	1721.000000	605.250000	4.767000	265000.000000
max	-114.310000	41.950000	52.000000	37937.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

We can also create a histogram to view the distribution of values:

```
dataset.hist( ‘sample_feature’ )
```



Pandas is also useful for choosing which part of the dataset we'll analyze and apply the machine learning algorithm to. We'll discuss more about this as we explore specific examples on how to use Python libraries for machine learning.

Aside from scikit-learn, numpy and pandas, many data scientists and machine learning engineers also use matplotlib. A few lines of code will allow you to create scatterplots and bar charts. We can even include some animation and movement to make the plot or chart more engaging.

When & How to Use Python Libraries

Main reasons for using libraries are for speed and convenience. It's like using "off-the-shelf" techniques and tools so you can better focus on the machine learning itself (instead of bothering yourself with hundreds of lines of code). These also allow professionals to get started quickly without diving too deep into mathematics and programming.

With all that speed and convenience though, it's still up to you how to use them effectively (or whether to use them or not in the first place). New tools and technologies will also appear that would claim to make machine learning 2x faster and easier.

That's why in the next chapter we'll briefly discuss how to initially approach machine learning problems no matter which tools you use. We'll use the libraries mentioned in this chapter to better illustrate the thought process.

Steps to Solving Machine Learning Problems

Many beginners make the mistake of immersing early on scikit-learn, pandas, numpy, and TensorFlow without first learning how to approach common machine learning problems.
That's why in this short chapter we'll discuss how to effectively approach such problems and perhaps save you a lot of time in the long run (especially if you plan to be a machine learning engineer).

Identify the Goal & Problem

This is the first step. Before we waste any resources and use sophisticated tools, what's our goal and what's the problem we're trying to solve?

Machine learning is not magic. It still requires a clear objective.
"Oh, perhaps the pattern will reveal itself if we spend enough time on it."
"It's organic and spontaneous. Something will surely happen."

In most cases not having a clear goal will waste a lot of time and resources. That's why at the start set a goal and determine the problem. This will give you a direction and lets you know when to stop. This also sets clear expectations among the team.

Getting & Accessing the Data

In Chapter 1 we mentioned that machine learning became popular recently because of the "explosion" of data.

That's why it's good to make sure that you have access to relevant data (and it's in sufficient amounts). In the later chapters we'll discuss how significantly more data can improve our machine learning model.

Visualizing the Data

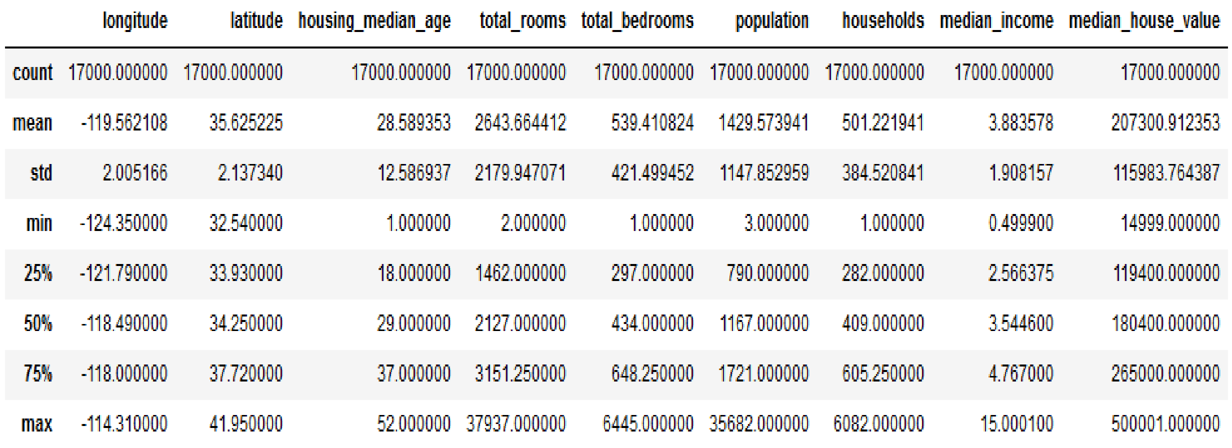
Once we have access to relevant data, one of the immediate next steps is to visualize the dataset (using matplotlib or other data visualization tools).

Probably nothing revolutionary will come out from viewing the histogram or the scatterplot. But at least it will give you some idea about the data (how the values are distributed and is there an obvious pattern?). Data visualization also allows us to quickly spot outliers (important in engineering and manufacturing). This data visualization step works well when you're dealing with numbers or words (e.g. analyzing the frequency of words in a book or comments feed).

Describing & Summarizing the Dataset

Does the data make sense? We might find an answer by doing the same thing as in Chapter 2:

```
import pandas as pd

dataset = pd.read_csv( 'your_file.csv' )

dataset.describe()
```



	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000
mean	-119.562108	35.625225	28.589353	2643.664412	539.410824	1429.573941	501.221941	3.883578	207300.912353
std	2.005166	2.137340	12.586937	2179.947071	421.499452	1147.852959	384.520841	1.908157	115983.764387
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.790000	33.930000	18.000000	1462.000000	297.000000	790.000000	282.000000	2.566375	119400.000000
50%	-118.490000	34.250000	29.000000	2127.000000	434.000000	1167.000000	409.000000	3.544600	180400.000000
75%	-118.000000	37.720000	37.000000	3151.250000	648.250000	1721.000000	605.250000	4.767000	265000.000000
max	-114.310000	41.950000	52.000000	37937.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

At the very top you'll see the **count**, which shows the number of instances for each feature or attribute. Any inconsistency in the row will reveal that there's missing data (or data in the wrong format).

The `dataset.describe()` will also show you if the data really makes sense. You can look at the mean and see if it falls within reasonable values. You can also see the standard deviation and get a quick idea of variability within the data (higher standard deviation means data points are more dispersed).

Cleaning the Data

As mentioned earlier, visualizing and summarizing the data will allow us to spot outliers or anything that seems off about the data.

The next step then is to determine the cause of those and possibly correct or eliminate them from our analysis. That's because erroneous data can skew or make our analysis far from correct.

This is where your domain knowledge comes in (e.g. specialized knowledge in chemistry, social behavior, traffic optimization, or markets). Correcting or eliminating a set of data requires judgment and that judgment may only be meaningful if there's some context to it.

Aside from erroneous and missing data, messy data (including incorrect and inconsistent file formats) can make the life of a data scientist and machine learning engineer a lot harder. For many projects dealing with messy data may take the bulk of the time.

After all, the whole machine learning project is only as good as the data's quality (Garbage In, Garbage Out as they say). The results and accuracy greatly depend on what you feed into the machine learning algorithm.

Summary of steps & beyond

1. Identify the goal and problem.
2. Get and access the data.
3. Visualize the data.
4. Describe or summarize the data.
5. Clean the data (Garbage in, garbage out).

To better understand these initial steps, let us look at an example and how Python is used in the initial steps of a machine learning project.

A Machine Learning Example: Predicting Housing Prices

Let us say we have a dataset about median housing prices and different features (e.g. total number of rooms).

First thought is to find out how the total number of rooms affect the house price. Second thought is given the total number of rooms, what would likely be the house price?

It's a general example of a Regression problem wherein you determine the relationship between a variable's value to that of the output. This is a common data science & machine learning problem in many businesses and research communities. That's because it allows us to predict the value of the output if we're given a value of a variable.

Let's explore an example to better illustrate the idea (using the steps outlined in the previous chapter).

Dataset: California Housing Dataset

Source: https://storage.googleapis.com/mledu-datasets/california_housing_train.csv)

Let's review the initial steps in solving machine learning problems:

1. Identify the problem.
2. Access the data (so our system can work on it).
3. Visualize the data.
4. Describe or summarize the data.
5. Clean the data (for easy manipulation & correcting errors)

Identify the Problem: Visualize relationship between number of rooms and housing price. Also predict the housing price given the number of rooms.

Access the data: The following lines of code allow us to do it,

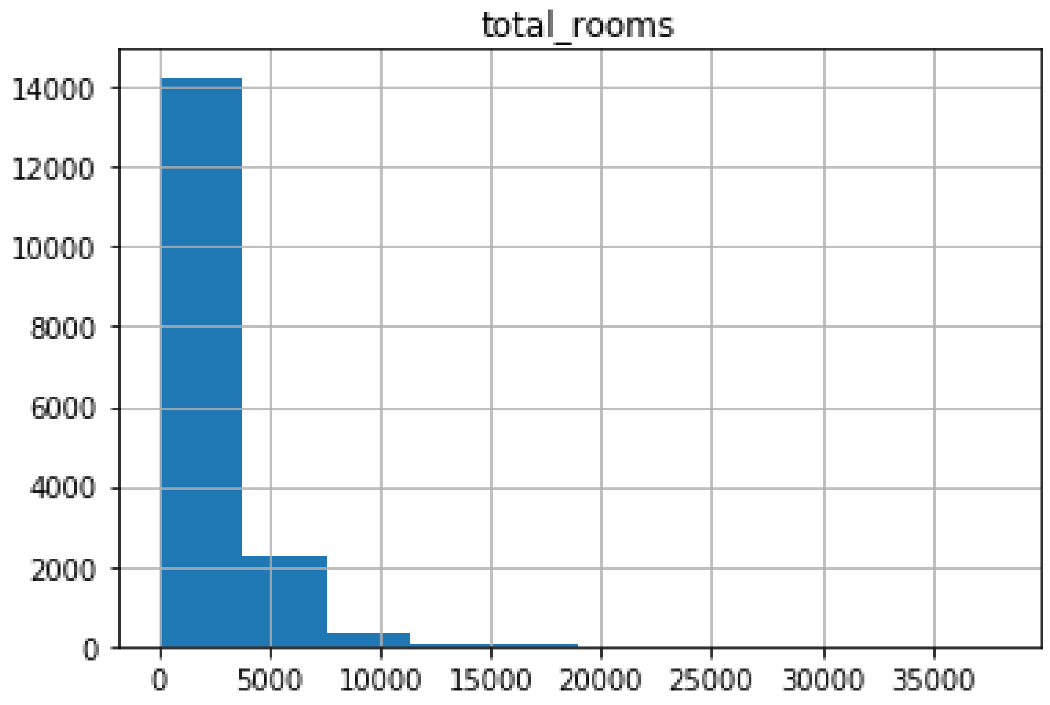
```
import numpy as np

import pandas as pd

dataset = pd.read_csv( ‘california_housing_train.csv’ )
```

Visualize, describe and summarize the data: Get a feel of the problem and data by viewing the summary and the plot.

```
dataset.describe() # This shows the summary of the data (mean, standard deviation, etc.)
```

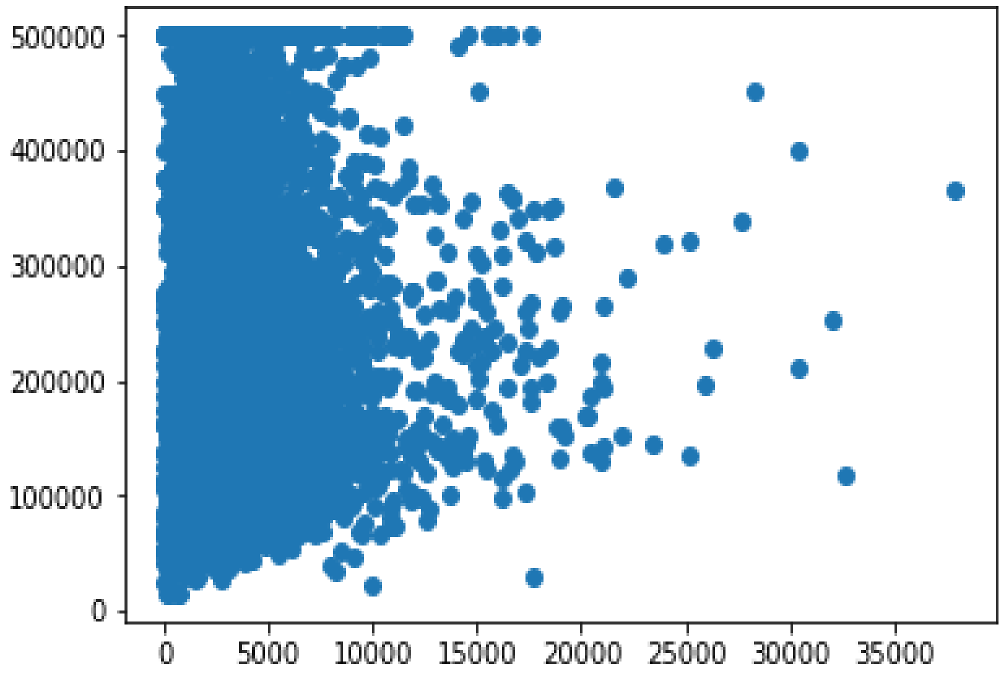
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000
mean	-119.562108	35.625225	28.589353	2643.664412	539.410824	1429.573941	501.221941	3.883578	207300.912353
std	2.005166	2.137340	12.586937	2179.947071	421.499452	1147.852959	384.520841	1.908157	115983.764387
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.790000	33.930000	18.000000	1462.000000	297.000000	790.000000	282.000000	2.566375	119400.000000
50%	-118.490000	34.250000	29.000000	2127.000000	434.000000	1167.000000	409.000000	3.544600	180400.000000
75%	-118.000000	37.720000	37.000000	3151.250000	648.250000	1721.000000	605.250000	4.767000	265000.000000
max	-114.310000	41.950000	52.000000	37937.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

```
dataset.hist( 'total_rooms' ) # View the frequency of values
```



Next is we use matplotlib to plot total_rooms against median_house_value:

```
import matplotlib.pyplot as plt

X = dataset[[ ‘total_rooms’ ]].values

y = dataset[[ ‘median_house_value’ ]].values

plt.scatter(X, y)
```



This gives us a quick idea about the relationship of the variable (total_rooms) and the output (median_house_value).
Perhaps you expected something a lot more sophisticated than this. However, this seemingly “simple” Regression example forms the foundation of many advanced Predictive Analytics procedures.
First we have to determine the relationship between a variable (or several features) and an output. Then we visualize the data using a scatter plot to better get an idea. Whether it’s a small dataset or a file with billions of data points (e.g. social media), the general steps are the same.

Don’t skip those fundamental steps and dive straight to “sophisticated machine learning algorithms.” After all, the goal is to **work and use the data** (not necessarily apply machine learning to it).
After we visualize the data, we can then look for outliers or anything out of ordinary in the data. These outliers can greatly skew our model. In other words, it can make our model perform poorly when new data comes in.

In other applications, outliers might be more important than the bulk of the data. That's because they signal serious issues such as fraud (financial transactions), overloading (computer servers fail) and potential workplace accidents (temperature & pressure levels are spiking).
Data visualization can quickly reveal those outliers. Software engineers and machine learning developers may also create a "trigger" to stop the harmful effects of outliers. For example, detecting an outlier early on may prevent a catastrophe in a nuclear power plant.

This is just the beginning

It was a simple and generalized example of machine learning. Best thing about this is you gained a few insights about how machine learning is implemented in the real world. It's about learning from data and then drawing conclusions, solving problems, preventing serious issues, and optimizing positive results.

In the next chapter, we'll take it a step further. This will require more lines of code (that seem more abstract and reserved for experienced programmers). As mentioned early in this book, the goal is to introduce you to the most common things that machine learning developers and data scientists do. This is to give you a solid foundation if you'll strive to be an expert.

Here's Where Real Machine Learning Starts

In the previous chapter you're introduced to a primer in "machine learning." Mostly it's only about visualizing the data and getting a feel about the relationship between a certain variable and the output.

That doesn't make it less important though. It's actually more important to get those basic things right early on before writing more code and applying complex machine learning algorithms.

Once you got the fundamental things really right, it's time to really apply machine learning according to its true meaning.

Dividing the dataset into training and test set

How do we know if the model we've built is working or good enough? As with all research and scientific endeavors, the best way is to test it.

We can accomplish that by dividing the dataset into training and test set. First, the system "learns" from the training set and then it "tests" that model against the test set. This is a good way to validate the model.

In Python and scikit-learn, this is often accomplished by the following lines of code:

```
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)
```

The input feature (X) is divided into train and test set. Same case with the output (y). The `test_size` is ⅓ which means ⅓ is for the test set and the rest (⅔) is for the training set. You can set the `random_state` to any value you like (purpose of a set `random_state` is to be consistent with producing random values.) Once the train and test sets are ready, it's now time to find the best "linear fit." In the next section we'll discuss what this exactly means.

Linear regression, training, and loss function

In the previous chapter you've gained a feel about the relationship between the total_rooms and the median_house_value just by looking at the scatterplot. But what if there's a new set of data about the total_rooms (or any other relevant variable) and you want to make a useful prediction about the corresponding median_house_value?

This is where you need to make a model which clearly shows the relationship between the variable and output. This can be in the form of a straight line which approximates that relationship.

As mentioned, it's only an approximation because many values won't fall exactly into that straight line. However, we have to get the best approximation to make the model useful.
Remember that the formula for a straight line is determined by:

$$\mathbf{y = mx + b}$$

Where y is the output, m is the slope, x is the value of an input feature, and b is the bias (the y-intercept).

We have to find the best value of m and b so that the resulting line yields the "best fit." Different values of m and b might increase or decrease the "loss" (how far off the line is with respect to the true value). This is where the loss function comes in.

The goal is to minimize the "loss" (how bad the model's prediction was on a single example). One popular approach is by summing up all the squared losses for individual examples and next is divide that number by the number of examples. It's often called the Root Mean Square Error (RMSE).

Another approach, which is perhaps more practical especially in large-scale applications, is reducing loss by gradient descent. It's an iterative approach wherein you pick a starting value for m or b and then perform "steps" to make the initial values approach the minimum loss.
In TensorFlow this might look something like this:

```
my_optimizer=tf.train.GradientDescentOptimizer(learning_rate=0.0000001)

my_optimizer = tf.contrib.estimator.clip_gradients_by_norm(my_optimizer, 5.0)

linear_regressor = tf.estimator.LinearRegressor(

    feature_columns=feature_columns,

    optimizer=my_optimizer

)
```

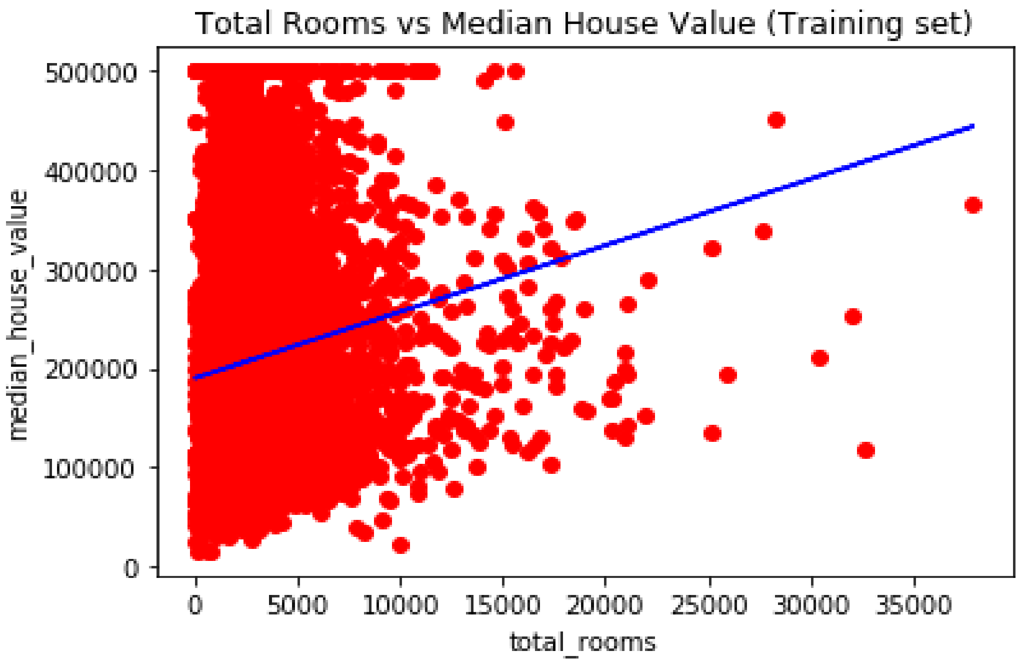
On the other hand, applying simple linear regression in scikit-learn may require the following code:

```
from sklearn import linear_model

regressor = linear_model.LinearRegression()

regressor.fit(X_train, y_train)
```

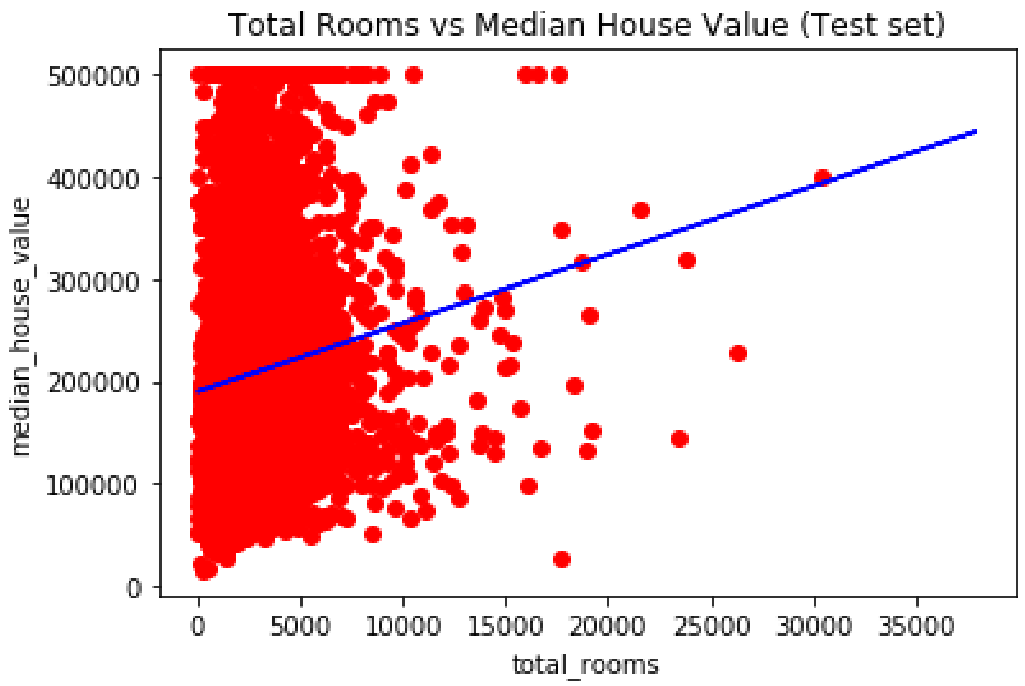
```
# This results to creating a linear fit. We can then make a prediction using the Test Set by:

y_pred = regressor.predict(X_test)
```

Let's put them all together and make a visualization:

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


dataset = pd.read_csv('california_housing_train.csv')

dataset.describe()

dataset.hist('total_rooms')


X = dataset[['total_rooms']].values

y = dataset[['median_house_value']].values

plt.scatter(X, y)


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)


from sklearn.linear_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X_train, y_train)


y_pred = regressor.predict(X_test)


# Create the linear fit using the train set

plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')

plt.title('Total Rooms vs Median House Value (Training set)')

plt.xlabel('total_rooms')

plt.ylabel('median_house_value')

plt.show()
```



```
# Apply that linear fit to the test set

plt.scatter(X_test, y_test, color = 'red')

plt.plot(X_train, regressor.predict(X_train), color = 'blue')

plt.title('Total Rooms vs Median House Value (Test set)')

plt.xlabel('total_rooms')

plt.ylabel('median_house_value')

plt.show()
```



Notice that we're applying the same line we got from the test set into the training set. After all, we're building the best fitting line to the training set and seeing if it's also working well with the test set.

Also notice that this seems far from being the best example (this is only for illustration purposes). No matter the specifics, the goal in linear regression is to find the best fitting line to approximate the relationship between an input and output & also be able to use that line in predicting values for "unseen" data (e.g. test set and new incoming data).

What If Regression Doesn’t Apply?

Spam or not spam? Approve the loan or not? Reject or accept the batch?

These problems are entirely different from linear regression. As a result, they require a different approach. After all, you’re not predicting a continuous value. Rather, you’re predicting whether it’s just 0 or 1 (spam vs not spam, approve vs reject).

These are called classification problems. The common goal is to determine whether a value falls either to 0 or 1. It could be very useful especially in the medical and healthcare field.
For example, an image of a lung x-ray may be classified as either having a tumor or not. This could possibly automate some parts of the diagnosis (and perhaps make the diagnosis more accurate and more consistent).

As with the Linear Regression example from the previous chapters, the machine “learns” from previous data. In this case of Classification, the data is in the form whether an image is classified as having a tumor OR not having a tumor. Once the machine “learns” from the data, the model created can then be used to classify future or unseen data.

This means each of the training data should have a label (0 or 1, spam or not spam). Both the input and output are provided (examples are labelled). Similar to how the human brain works, the model learns through the examples given.

A famous example: The Iris Dataset

Data source: https://archive.ics.uci.edu/ml/datasets/iris

This is one of the best examples on how to apply and learn Classification in machine learning. The goal here is to classify flowers as Iris setosa, Iris versicolour, or Iris virginica based on the following given features/attributes:

- Sepal length in cm
- Sepal width in cm
- Petal length in cm
- Petal width in cm

Here’s a general example. Given the lengths and widths of the sepals and petals of an Iris flower, what is its specific class?

Perhaps there’s a range in sepal length and width for an Iris setosa or a certain combination of petal length and sepal length for it to be an Iris versicolour.

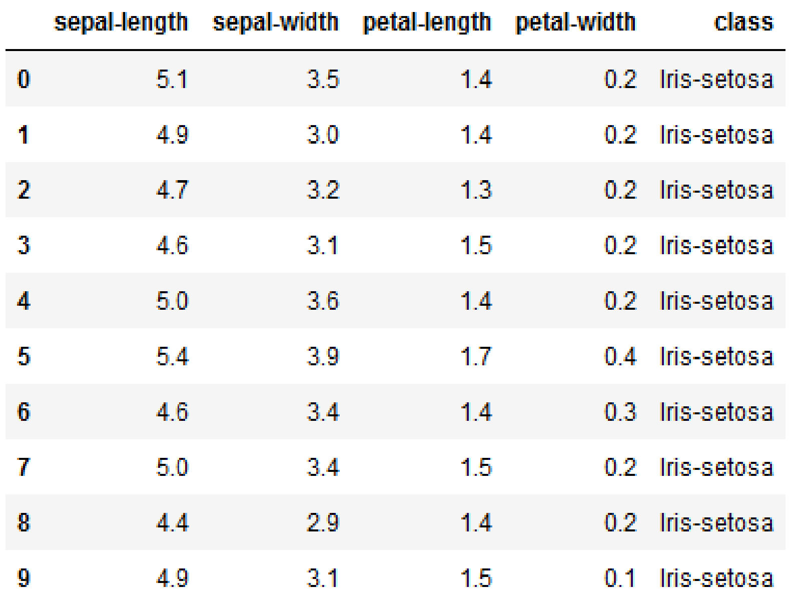
This seemingly simple problem could be the similar basis for image recognition and face detection apps. The system learns from the given labels and examples. Then it will be able to apply that learning into new data with different measurements of attributes.

Let's learn how to accomplish that using Python and scikit-learn:

```
import pandas as pd

import matplotlib.pyplot as plt

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class'] # column labels

dataset = pd.read_csv( 'iris.data' )
```

Let's take a peek of the data:

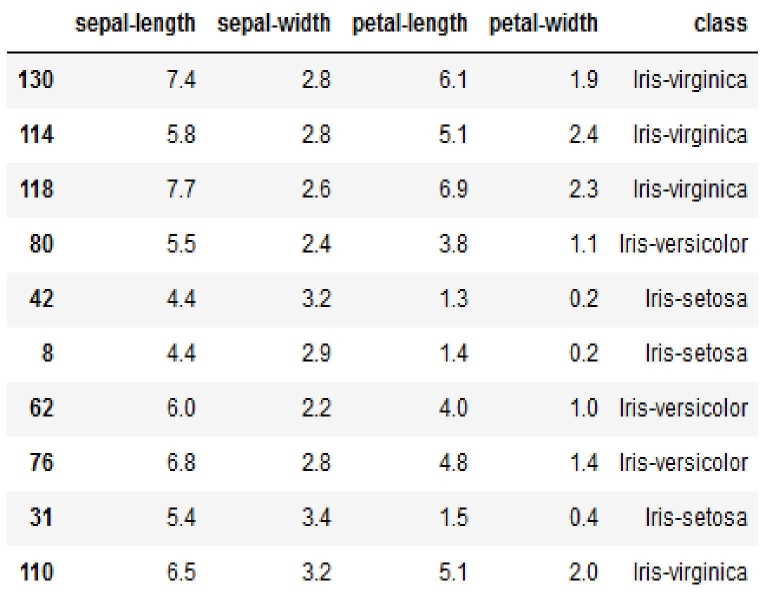
```
dataset.head(10) # Shows first 10 entries of the data, indexing starts at 0 in Python
```



	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

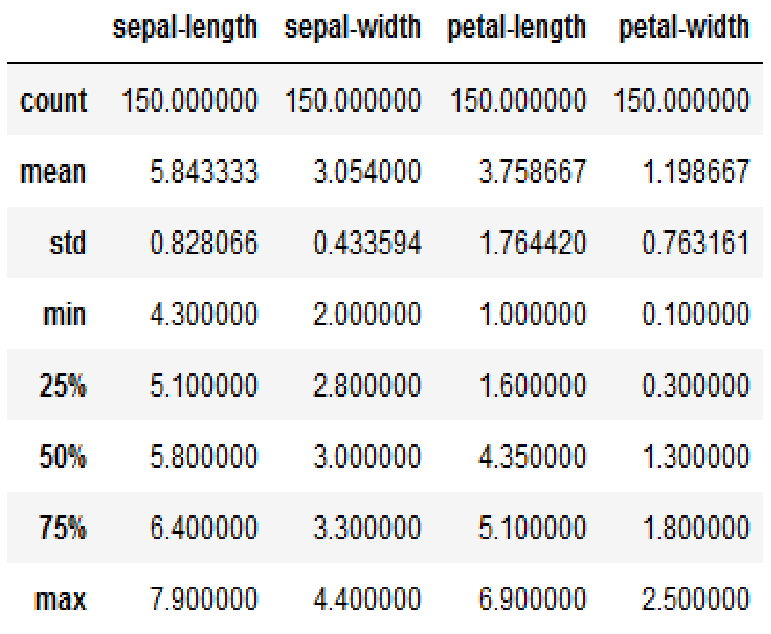
We can also look at 10 random entries:

dataset.sample(10)



	sepal-length	sepal-width	petal-length	petal-width	class
130	7.4	2.8	6.1	1.9	Iris-virginica
114	5.8	2.8	5.1	2.4	Iris-virginica
118	7.7	2.6	6.9	2.3	Iris-virginica
80	5.5	2.4	3.8	1.1	Iris-versicolor
42	4.4	3.2	1.3	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
62	6.0	2.2	4.0	1.0	Iris-versicolor
76	6.8	2.8	4.8	1.4	Iris-versicolor
31	5.4	3.4	1.5	0.4	Iris-setosa
110	6.5	3.2	5.1	2.0	Iris-virginica

dataset.describe() # Show mean, standard deviation, summary of data

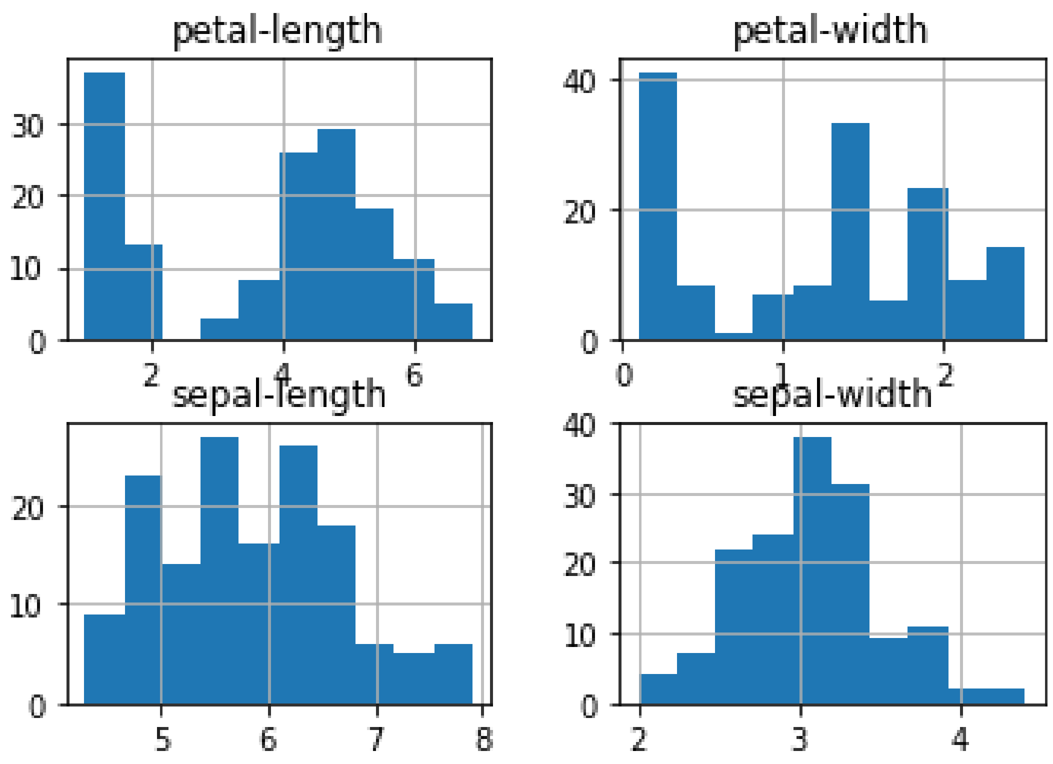


	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

From the summary, there seems to be nothing suspicious. The values are in similar scale and the data is complete (count is 150 for all columns).

We can then visualize the data by creating histograms for each feature (get a better idea about the distribution of values):

```
dataset.hist()
```



As with the Linear Regression example, we could also divide the dataset into training and test set (to validate our model):

First, set where to get the X and y values:

```
X = dataset[['sepal-length', 'sepal-width']].values # for simplicity we only use the first 2 features

y = dataset[['class']].values
```

Then, split the dataset:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/4, random_state = 0)
```

The size of the training set is 75% of the original (25% is for the train set). This value is actually arbitrary and can be changed anytime. Keep in mind though that the goal is for the model to learn from enough data.

What's next? We then use Logistic Regression that takes advantage of maximum likelihood. It's just one of the basic approaches to Classification.

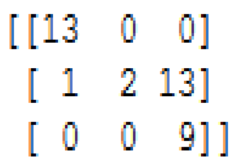
This is different from the Linear Regression discussed earlier. Linear Regression is often about minimizing the errors (or loss) to come up with the best fitting line. In contrast, Logistic Regression is more about probabilities (the probability of a given value as 0 or 1, True or False).
There's a straightforward way to implement this on Python and scikit-learn:

```
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

Again, our system builds a model based on the training set. Then, we can apply that same model to the test set and get the corresponding predictions. An easy way to do this is by showing a confusion matrix:

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
```

Output:

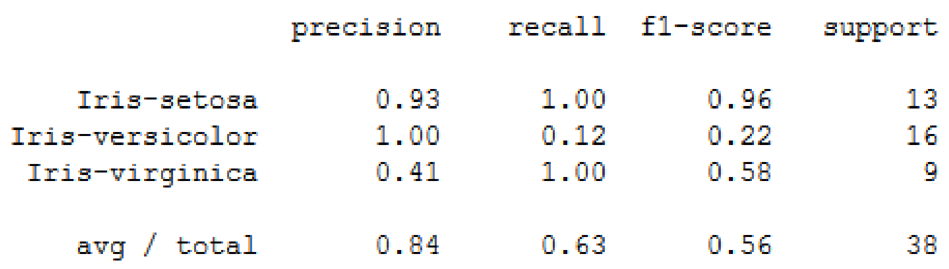


```
[[13  0  0]
 [ 1  2 13]
 [ 0  0  9]]
```

A confusion matrix describes the classifier's performance. With the confusion matrix you can get a quick idea of the Predicted vs Actual values. This also shows how many are the True Positives, True Negatives, False Positives, and False Negatives.

Yes, this could be a bit hard to interpret and master which is why many data scientists use references all the time. But if you want to quickly assess the model's performance, you can read the Precision and Recall instead:

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```



```
                 precision    recall  f1-score   support

    Iris-setosa       0.93      1.00      0.96        13
Iris-versicolor       1.00      0.12      0.22        16
 Iris-virginica       0.41      1.00      0.58         9

    avg / total       0.84      0.63      0.56        38
```

The most important thing to remember is to get both **high values for Precision and Recall**. This somehow means the model captures a lot of the True Positives while keeping away the False Positives.

Other methods besides Logistic Regression

There are other classifiers you can use such as:

- Naive Bayes classifier
- K-nearest neighbor (or knn)
- Random forests
- Support vector machines (SVMs)
- Decision trees
- Random forest

The major difference is how each of them works in the background (the mathematical basis for their probabilities and classification). In actual implementation, the differences could be very minor.

For instance, you might just need to replace 2 lines of code:

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)
```

For Random Forest classifier

```
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
```

Many data scientists and machine learning developers actually use ensemble methods (a combination of methods) that average the results of different methods. This approach is said to be more robust (reduces the effects of too much variability). They also choose a method that best suits their purpose and application (which brings the best results?).

Clustering

Both Regression and Classification require an input/s and an output. What if there's no output at all? What, if you just want to reveal the natural aggregation of data points?

This is where Clustering comes in. The goal is to find groups in data and assign each data point to one group. For example, you have a grocery store and thankfully, it has many customers each day. Most of them have a loyalty card, which they use to gather points whenever they buy from you.

When they signed up for the loyalty card they willingly gave their estimated income range. Aside from that information, you're also able to track their purchases through the loyalty card. As a result, you have 2 sets of data: their income range and purchase score (higher score means they're a frequent buyer).

How will that be useful? We can apply Clustering to group the customers according to their Income & Purchase Score. This will reveal the naturally formed groups/aggregates, which can guide us in planning our marketing campaign and business strategy.

Let us quickly apply Clustering so you'll see what we're talking about. We start with importing the required libraries:

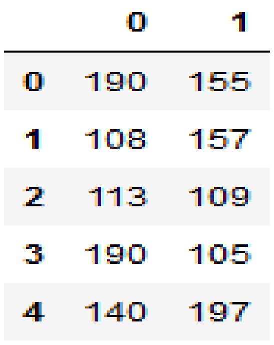
```
import numpy as np

import pandas as pd
```

Next is we generate random data (this is only for illustration purposes). This is very useful for testing and practicing Python & data science.

```
dataset = pd.DataFrame(np.random.randint(100, 200,(100, 2)))
```

This will generate 2 columns (1st column is Income, 2nd column is Purchase Score) with 100 rows. The range of the values is from 100 to 200. Let's take a peek:

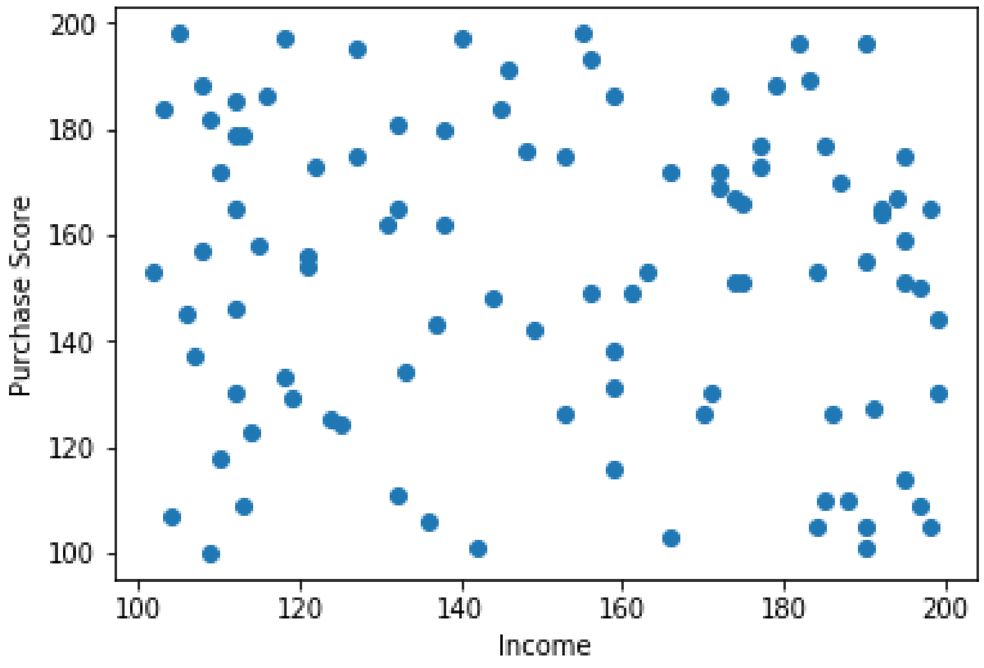
```
dataset.head(5) # Shows the first 5 entries
```



	0	1
0	190	155
1	108	157
2	113	109
3	190	105
4	140	197

Let’s then visualize the data to get an overview:

```
X_vis = dataset[0].values

y_vis = dataset[1].values

plt.xlabel('Income')

plt.ylabel('Purchase Score')

plt.scatter(X_vis, y_vis)

plt.show()
```



At first, it can be hard to see the Clusters because it all seems random. However, given enough data points, some points tend to aggregate and naturally form groups. One way to better see those groups is by doing Clustering.

In marketing (and in life in general), we can't please everybody no matter how hard we try. That's why we have to somehow create different campaigns for different customer groups. This way, the marketing message will better appeal to a certain group.

First, we have to create groups for those customers. How many groups should we create? Are the formed groups meaningful?

This is where our objectives and judgment come in. Too many groups and our resulting marketing strategy won't be targeted enough. Too few groups could make our strategy generalized.

How to reveal the clusters

We have to apply a method to form and reveal those clusters. One of the simplest and most popular methods is k-means. It's an iterative approach wherein the goal is to partition the data points into k clusters. The end result is minimizing the within-cluster sum of squares (WCSS).

The full explanation seems complex but its application is easy and straightforward. The method is also good enough for most applications.

Let's start applying k-means clustering and continue with our work earlier:

First is we set the values to be included in our Clustering goal

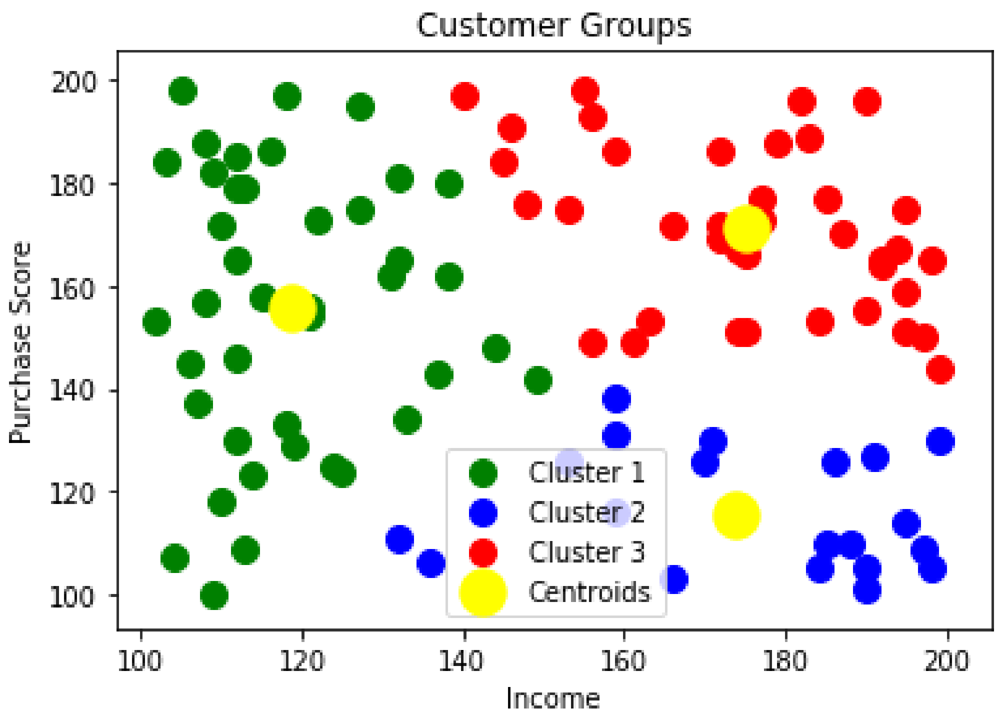
```
X = dataset[:].values # this means all of data points are included
```

Then, we select the number of clusters (n_clusters) and implement the k-means algorithm

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 3, init = 'k-means++', random_state = 0)

y_kmeans = kmeans.fit_predict(X)
```

Finally, we can again plot the points. This time though the Clusters are more visible because of the color differences.

```
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'green', 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 = 'red', label = 'Cluster 3')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label
= 'Centroids')

plt.title('Customer Groups')

plt.xlabel('Income')

plt.ylabel('Purchase Score')

plt.legend()

plt.show()
```

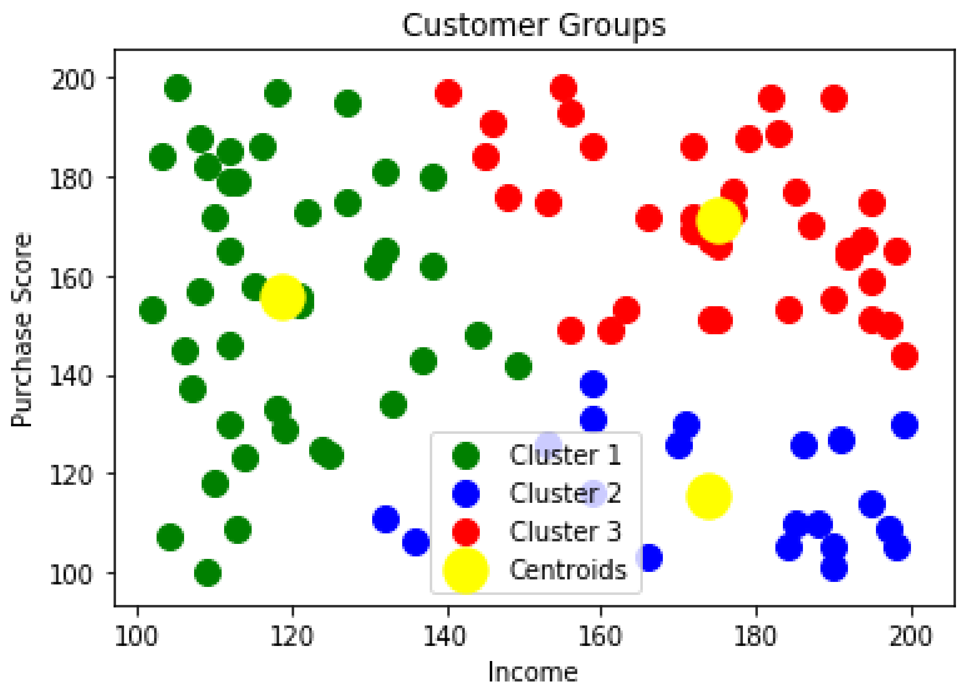


The Yellow points are the Centroids and the Green, Red, and Blue Points are in their respective clusters. Notice that the Centroids seem to be at the "center" of each cluster (minimizing the sum of squares of the distances).

The number of clusters may be arbitrary. We can set it to 2, 4, 5, or any other number. The choice actually depends on our goal and preferences. There are mathematical and recommended ways to determine the supposed number of clusters. Nevertheless, it's always up to us if the revealed clusters suit our goals.

What can we do with those Clusters?

As an example, pay attention to the Blue data points.



They have High Incomes but Low Purchasing Scores. In other words, the customers in this group have a lot of money but they might be spending too little of it in your profitable grocery store.

The goal then is to encourage them to buy more from your store. Here are a few possible ways to accomplish this:

- Sell more of what high-income people buy
- Sell products related to what this group already buys (e.g. if they always buy shaving products, they might also be interested in luxury grooming kits)
- List their brand preferences and offer more products under those brands

Although it's still possible to arrive at those recommendations without applying Clustering, our method and the results could be reproducible and consistent. Instead of randomly formulating strategies, we have a documented way of revealing those clusters. This could be a part of making your organization more data-driven.

How to Improve Your Model's Performance

In the previous chapters we explored Linear Regression, Classification, and Clustering. We applied the appropriate algorithms to make sense of the data.

In this chapter we'll focus on improving the accuracy of our machine learning model. In the real world, data science and machine learning actually goes beyond applying sophisticated algorithms. It's also about making the model optimized or useful enough for our purposes.

For example, even a 1% improvement in a model's prediction accuracy could translate to millions of dollars of additional profits. Small accuracy improvements may also help prevent massive fraud and catastrophes in various industries especially in healthcare.

So how can we improve our model and achieve a higher accuracy? How can we accomplish that without sacrificing the accuracy on unseen data? What are the pros and cons of striving for a higher accuracy?

Let's discuss it now.

Learning from more data & examples

More data points may allow our model to "learn" more. It's similar to how we personally learn. If we've seen many examples, our view and learning may become more accurate and realistic. But if we're only exposed to very few examples, the tendency is to become biased or we're getting incomplete information.

That's why companies are busy gathering more data from users and other sources. More data could mean better prediction accuracy (e.g. better product recommendations, customized offers, fewer diagnostic errors).

However, gathering more data may not be worth it in some cases. Often there are financial and time constraints in many machine learning projects. Further data collection could be expensive or by the time it's been accomplished, a competitor might have already come up with a good enough solution.

Choose ensemble learning

This is another method of improving accuracy. Simply, it's using a combination of machine learning models ("the ensemble") then averaging the results. It's similar to applying an integrated approach (often a combination of several methods) so we can make better decisions.

For instance, Charlie Munger (Warren Buffett's long-time business partner) has several models he uses when making major business and investment decisions. These mental models may come from different subject areas such as psychology, physics, and perhaps even evolutionary biology.

This combination of different approaches may or may not improve our decision (because luck still plays a huge part). However, the use of several approaches may help correct for the individual errors coming from each model.

Let's say we're deciding whether to invest in a business or not. Should we just consider the company's present profitability and ignore all the other factors? What if we could make a better decision by also considering its past financial performance, the current economic climate, the recent relevant technological advances, and the competency of the new CEO?

Considering all those other factors may greatly affect your decision. In addition, they could help improve your chances of getting the best results.

In the case of data science and machine learning, an ensemble approach could lead to better results. For example, many practitioners choose Random Forest classification (an ensemble approach) instead of Decision Tree. That's because Random Forest is actually made up of a collection of many decision trees.

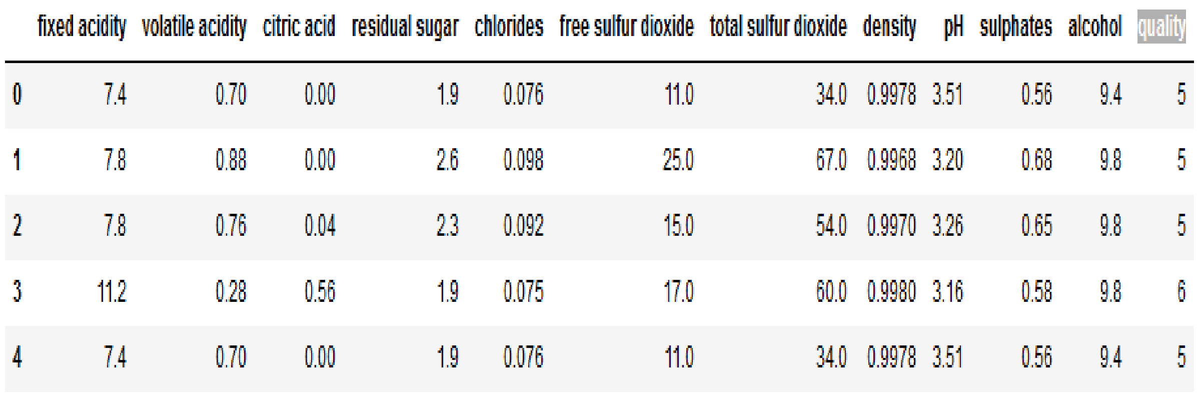
The Random Forest algorithm builds multiple decision trees from different sets of random data points. The "randomness factor" may help correct the errors and biases in individual decision trees.

Let's compare the accuracies of Decision Tree and Random Forest using Python and scikit-learn:

Data source: https://archive.ics.uci.edu/ml/datasets/wine+quality

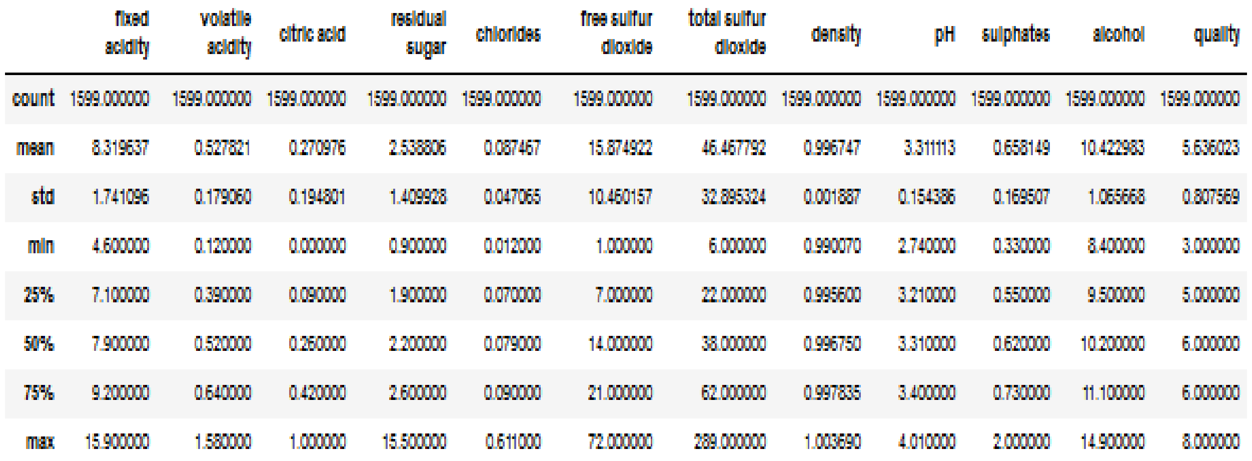
# Decision Tree Classifier

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read_csv('winequality-red.csv', sep=';')

dataset.head(5) # shows first 5 entries
```



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

```
dataset.describe() # shows data summary (mean, standard deviation, etc.)
```

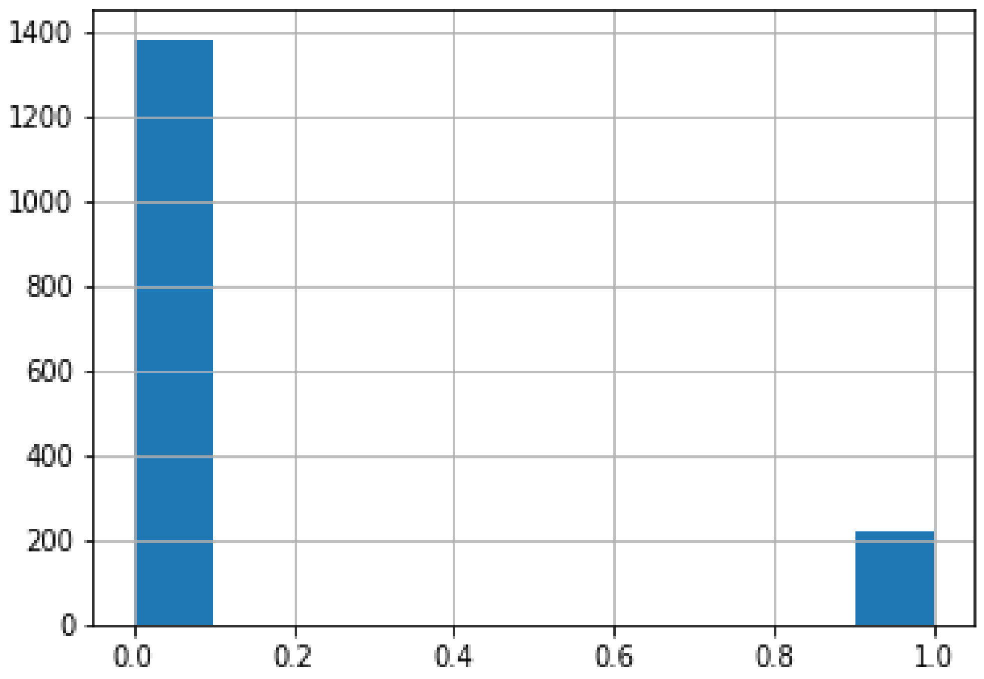


	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	5.636023
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0.807569
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	5.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000	6.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000	6.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.000000

Remember that the quality score ranges from 0 to 10. This means the score could be between 11 different values which can be complicated for our classification task.

To simplify the task, we can divide the quality scores into just 2 categories. Wines with quality scores of equal or higher than 7 will be considered Good (1). While the rest will be considered Bad (0).

```
def isGood(quality):
    if quality >= 7:
        return 1
    else:
        return 0
X = dataset.iloc[:, 0:11].values
y = dataset['quality'].apply(isGood)
y.hist()
```



```
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)
```

Before we apply the Decision Tree classification algorithm, it's recommended to perform Feature Scaling to standardize the range of values from among the different features. This way, no feature will dominate. Each feature will somehow contribute equally to the model.

For example, features with a broad range of values may dominate. It's ideal to put all the values of the features under the "same scale or magnitude." We can accomplish this through the following lines of code.

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train) # apply standard scaling to X_train

X_test = sc.transform(X_test)
```

After Feature Scaling, we can then now apply the Decision Tree classifier and get the accuracy_score.

```
from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)
```

Result is **0.90625**

The accuracy score is around 90.6% when we use the Decision Tree Classifier. Let's then use Random Forest Classifier and compare the results. Most of the lines of code are just the copy of the code for Decision Tree.

# Random Forest Classifier

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read_csv('winequality-red.csv', sep=';')

dataset.head(5)

dataset.describe()


def isGood(quality):

    if quality >= 7:

        return 1

    else:

        return 0


X = dataset.iloc[:, 0:11].values

y = dataset['quality'].apply(isGood)

y.hist()
```

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)


from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)

classifier.fit(X_train, y_train)


y_pred = classifier.predict(X_test)

from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)
```

Result is **0.928125**

The accuracy score from applying Random Forest (~0.92) is a bit higher than Decision Tree (~0.90).

This approach is actually similar to the so-called Wisdom of Crowds wherein the collective opinion of a group is often better than the opinion of a single individual (or even an expert). In other words, large groups of people might be collectively smarter than individual experts.

It's also the case with ensemble methods in machine learning. The use of multiple methods and averaging the results could be more accurate. For example, practitioners might use all the following algorithms for Classification:

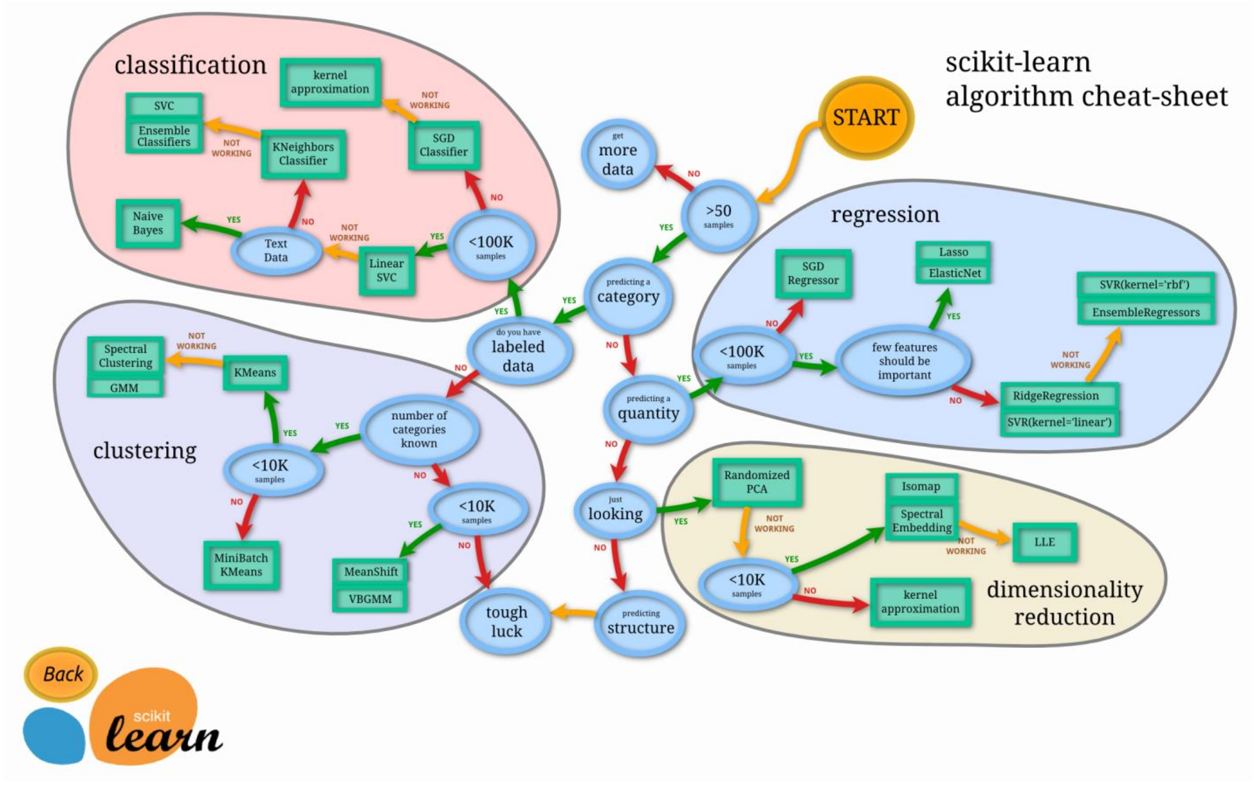
- Logistic Regression
- Naive Bayes
- KNeighbors
- Support Vector Machines (SVMs, ensemble learning)*

Use a cheat sheet

Aside from using ensemble learning, another way to improve results accuracy is by referring to a cheat sheet before, during, and after applying the data preprocessing and actual machine learning steps.

A cheat sheet is actually another term for reference, short guide, and summary. It's your quick source of information whenever you're starting something. It can also be used as a checklist so you're not missing on anything.

Here's an amazing example of a "cheat sheet" in machine learning:



Source: http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

Just by following the arrows you increase your chances of getting accurate results and reducing the possible mistakes and errors. It doesn't include all the information you'll need, but it's a great quick reference you can use anytime.

Other reasons to use a cheat sheet:

- We can't remember everything
- We can trace the steps where we went wrong
- Focus on asking the more important questions instead of focusing on the minutiae
- Easier documentation so we can study again our model later

There are various cheat sheets available about data science, machine learning, numpy, pandas, scikit-learn, and Python. A few were made by expert practitioners who want to make things easier for fellow professionals (especially for beginners).

Before using a cheat sheet though, it's best to study and review the fundamentals first. This way, you're not just applying a formula without gaining real understanding. Formulas come and go (even the tools and technologies get replaced every few years).

But with strong fundamentals, you'll gain a strong advantage through the years. You'll know the limitations of each algorithm. This approach demands more time though. But almost any worthwhile goal truly requires significant amounts of time and effort.

Choose only the most relevant features/attributes

Which are the most important factors that affect success in any field? Hard work is a sure one. What about the special morning routines? Do those really contribute to a person's success?

There are many other factors that people say are crucial to success such as diet, where the people is born, if she had a summer job, if the person learned to be independent while young, and so on.

But let's face it, only a few of those factors actually contribute to the positive results. It's similar to the 80/20 principle popular in business and management. Only 20% of the input is responsible for 80% of the output (not exactly but you get the point).

It's a similar case with machine learning. People collect massive amounts of data that result to thousands of feature columns. It's a huge task that even computers will struggle to crunch the numbers.

Let us say we're able to accomplish that gargantuan task. Gathering more data may not significantly increase the model's accuracy. We might also use ensemble learning methods but there's a limit to how much it can help.

The 80/20 principle might also help again here. We only select the most relevant features that are likely to have the largest effect on the output.

However, the pressing question then is this: How do we figure out which are the most important features? These are the common approaches to find out and make the right selection:

- Use expertise in the subject (domain knowledge)
- Test the features one by one or in combination and find out which produce the best results

Let us start with domain knowledge. Many companies prefer hiring data scientists and machine learning developers who are also experts in certain fields (e.g. agriculture, manufacturing, logistics, social media, physics). That's because a strong background knowledge is often essential in making sense of the data (and taking advantage of it).

For example, the following factors are often attributed to startup success:

- Timing (or just plain luck)
- Potential market evolution (how the market will grow through years)
- Proportion of investment to research & development
- The founders (can they execute and sustain the plan?)
- The marketing (starting niche then go broad, or go broad in the first place)
- Competitors
- Current economic, social, & political climate (will the society embrace the idea & startup?)

At first, it seems each of those factors can have a huge effect to a startup's success. However, a few experienced investors might only consider 2 or 3 factors. Even fewer investors will only consider just one factor (timing or plain luck). With that in mind, they'll spread their investments in different startups and wait for a winner.

Experience and domain knowledge play a role in that decision (determining which are the most relevant factors and deciding where to invest the bulk of their money). Without those, they'll be spreading themselves too thin and perhaps get results which are far from optimal.

Use dimensionality reduction

Dimensionality reduction simply means we'll work only on fewer features as a result of removing redundancies. It's similar to mapping data from a higher dimensional space into a lower dimensional space.

For example, we have 2 features and one output. This may require plotting the points into a 3-dimensional space. But if we apply dimensionality reduction, the plotting may only require 2d.

This technique may or may not affect the accuracy score of our model. One sure thing though is the whole thing becomes easier to visualize and interpret.

Even with Dimensionality Reduction, the analysis and model creation could still be complex and challenging because of multiple features (e.g. number of features > 5). One common way to deal with this is through Principal Component Analysis (PCA).

Here's an extensive example to illustrate PCA: Modeling Wine Quality According to Physicochemical Properties (data source: https://archive.ics.uci.edu/ml/datasets/wine+quality). Researchers and analysts performed chemical tests on different wine samples as well as gather the quality scores as given by raters.

The goal is to model wine quality given the results of testing the wine samples for their physicochemical properties. These properties (or features/attributes) are the following:

1. Fixed acidity

2. Volatile acidity
3. Citric acid
4. Residual sugar
5. Chlorides
6. Free sulfur dioxide
7. Total sulfur dioxide
8. Density
9. pH
10. Sulphates
11. Alcohol

The quality score ranges from 0 to 10 (0 is poor rating, 10 is excellent).

We have 11 feature columns with 1 target output (predicting quality score based on different measurements of physicochemical properties). The goal then is to determine or reveal the "principal components."

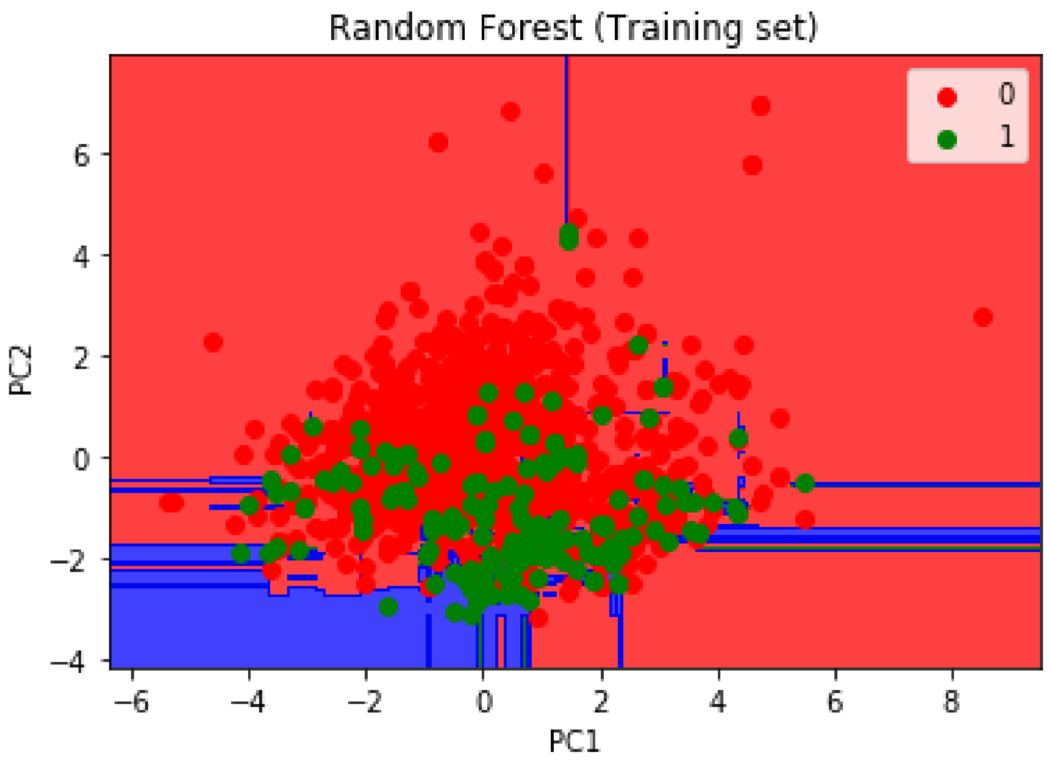
What happens is the data gets summarized so there will be "fewer" properties to think of. The PCA may accomplish this by searching for properties that show the most variation among wines. It's similar to getting a list of new features (but this time, fewer features). Two or more of the features might be combined into one (constructing new features using the old ones).

The concept might be better explained by using Python and scikit-learn & using again the example earlier about Modeling Wine Quality with Physicochemical Properties:

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read_csv('winequality-red.csv', sep=';')

dataset.head(5)

dataset.describe()

def isGood(quality):

    if quality >= 7:

        return 1

    else:
        return 0

X = dataset.iloc[:, 0:11].values

y = dataset['quality'].apply(isGood)

y.hist()

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)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)
```

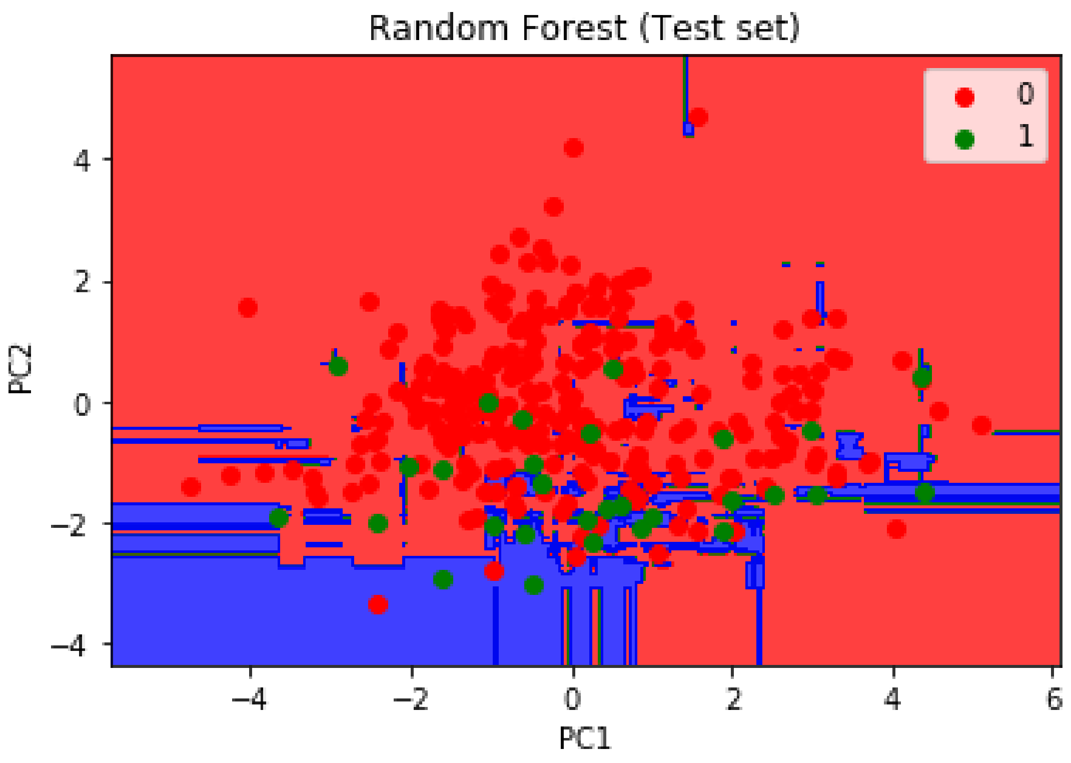
Now here's where it becomes different. We apply Principal Component Analysis (PCA) and "reduce" the 11 features into just 2 components (n_components = 2).

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)

X_train = pca.fit_transform(X_train)

X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)

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', 'blue')))
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', 'blue'))(i), label = j)
plt.title('Random Forest (Training set)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```



Notice that in the visualization above, there's only the PC1 and PC2 (only 2 components). Our Random Forest Classifier only used 2 "features" to classify the wines into 0 (bad, ratings below 7) and 1 (good, ratings equal or higher than 7).

```
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', 'blue')))

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', 'blue'))(i), label = j)

plt.title('Random Forest (Test set)')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.legend()

plt.show()
```



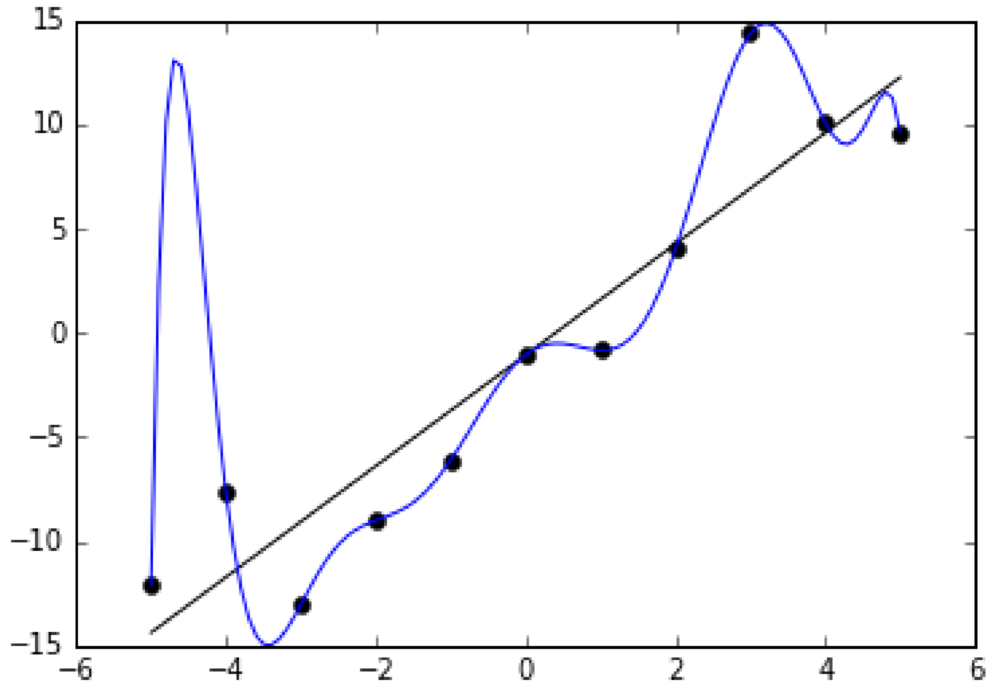
As with earlier examples, we applied the same classifier from Training Set into the Test Set and see if it performs well enough.

Using Principal Component Analysis and other Dimensionality Reduction techniques will just add a few lines of code to our analysis. The main use of PCA and similar techniques is that we're trying to include all features in our model, visualization, and analysis.

Instead of choosing just 1 or 2 input features based on instinct or domain knowledge, with PCA we create 2 "new" features that try to include all of the original features. This may or may not affect the accuracy of our model. Nevertheless, the good thing is there's lower loss of data (in contrast to ignoring all the other features).

Beware of overfitting

Overfitting is that the data points are overly fitting with our model. Here's a good example:



Instead of a straight line that misses most of the data points, the model literally followed the points to include them all in our model. The result is the model appears complicated and with a lot of twists and turns.

The accuracy score for the training set (or the current available data) might be excellent (~100%). However, when new data comes, the model will fail spectacularly. In addition, the model seems complicated. The equation for that model is likely to be a high degree polynomial equation (e.g. $y = x^2 - 2x^3 + 5x^5$) instead of just the simple $y = mx + b$.

How do we prevent overfitting and still maintain a good accuracy score? One way is by wise feature selection. Using our domain knowledge (or consulting with an expert in the subject area), we can select 1 or 2 features that may have the biggest impact to the output. For example, the weather in a particular day might not have anything to do with the result of a basketball game (but other data such as home court advantage might have).

As hinted earlier though, feature selection may also mean loss of data and therefore some tradeoff on avoiding overfitting. That's why experts may use Dimensionality Reduction techniques instead to avoid ignoring features that might have an actual impact to the output.

Another way to prevent or solve overfitting is by getting more data (more examples for our model to train with). It's somehow similar to the Law of Large Numbers wherein as the number of experiments increases, the actual and expected ratio of outcomes converge (nearing the values of each other). For instance, out of 10 coin tosses you might get 8 tails (0.8 ratio). But if you toss the coin maybe a million times, the result you get might be near the expected value which is 0.5.

With more training data you might get a model that generalizes well and may perform well enough with unseen data. However, getting more data may be more expensive and time-consuming.

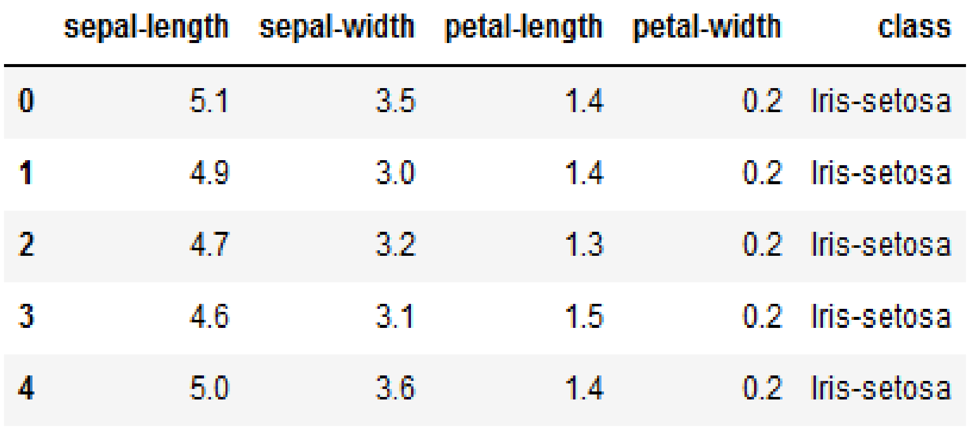
There's a way though to train on more data (while still using the same dataset). It's by applying Cross-Validation (specifically k-fold cross validation). Here's a simple example of how it works.

You divide the dataset into 10 subsets (Subset 1, Subset 2, Subset 3, and so on). You'll use Subsets 2 to 10 as your Training Data (while Subset 1 is your Test Set). You do this procedure again but this time you use Subset 2 as your Test Set while the rest is your Training Data. You do this until the "rotation" is complete.

The result is you've used different combinations of Training Data and Test Set. This is a good way to prevent overfitting because the model has a lot "more chances of learning." It's similar to what we've discussed about Ensemble Learning (use of different or combination of algorithms then averaging the results).

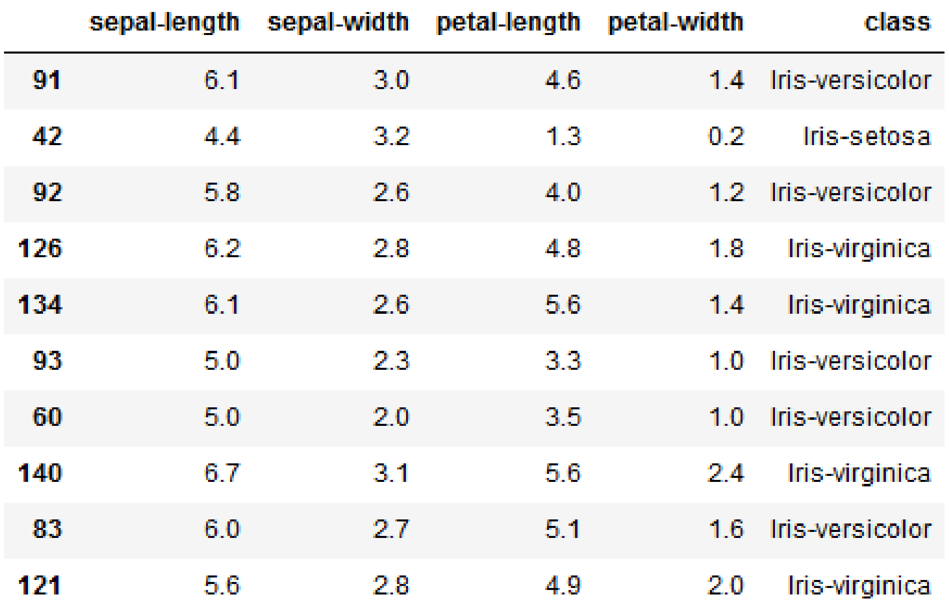
Let us revisit our Classification example earlier (classifying flowers into Iris setosa, Iris virginica, and Iris versicolor according to their sepal length and sepal width). This time though, let us apply Cross Validation (CV) and compare the accuracies from with and without CV.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

dataset = pd.read_csv('iris.data', names=names)

dataset.head(5)
```



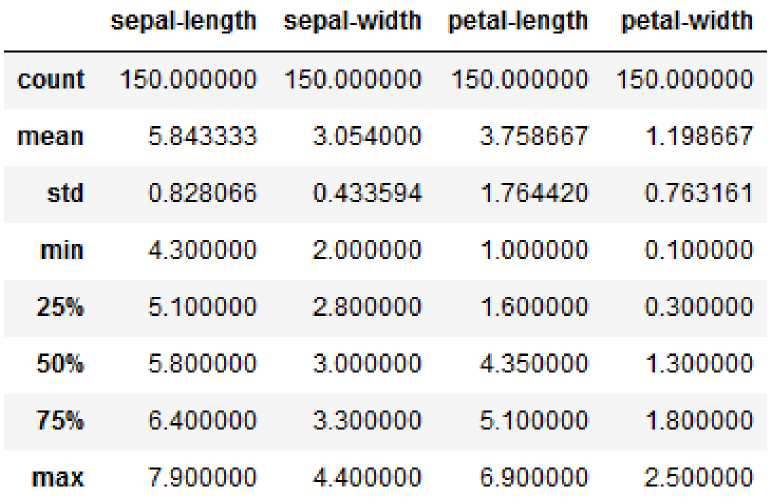
	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
dataset.sample(10)
```



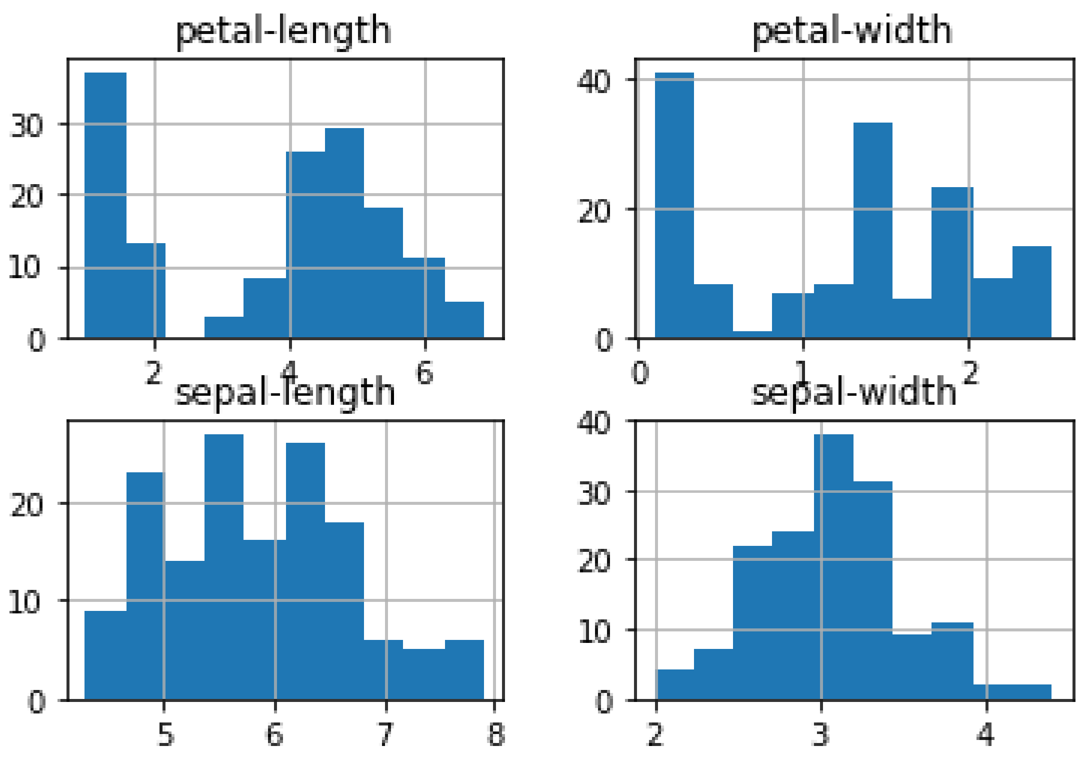
	sepal-length	sepal-width	petal-length	petal-width	class
91	6.1	3.0	4.6	1.4	Iris-versicolor
42	4.4	3.2	1.3	0.2	Iris-setosa
92	5.8	2.6	4.0	1.2	Iris-versicolor
126	6.2	2.8	4.8	1.8	Iris-virginica
134	6.1	2.6	5.6	1.4	Iris-virginica
93	5.0	2.3	3.3	1.0	Iris-versicolor
60	5.0	2.0	3.5	1.0	Iris-versicolor
140	6.7	3.1	5.6	2.4	Iris-virginica
83	6.0	2.7	5.1	1.6	Iris-versicolor
121	5.6	2.8	4.9	2.0	Iris-virginica

```
dataset.describe()
```



	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

```
dataset.hist()
```



```
X = dataset[['sepal-length', 'sepal-width']].values

y = dataset[['class']].values

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)

from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)

classifier.fit(X_train, y_train.ravel())

y_pred = classifier.predict(X_test)

from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)
```

Result: **0.63** (no Cross-Validation applied)

Now, let's apply Cross-Validation and compare the results:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

dataset = pd.read_csv('iris.data', names=names)

X = dataset[['sepal-length', 'sepal-width']].values

y = dataset[['class']].values

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)

from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)

classifier.fit(X_train, y_train.ravel())

y_pred = classifier.predict(X_test)
```

Now here's where it gets different. We'll apply Cross-Validation using scikit-learn and get the average or mean of the accuracies from the rotation of the training and test sets.

```
from sklearn.model_selection import cross_val_score

accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)

accuracies.mean()
```

Result is **0.69** (with Cross-Validation applied)

Without Cross-Validation, our accuracy (a single value) is only 0.63 or 63%. But **with Cross-Validation**, the average of the accuracies (from rotating the training and test sets) is higher, 0.69 or 69%.

Our model has an improved accuracy just by using Cross-Validation or rotating the training and test sets. It's rigorous and requires some computational time and power. However, the improvement in accuracy can be worth it especially when talking about millions of data points to be analyzed or the field's sensitivity and importance (e.g. healthcare & medical diagnosis, scientific research).

Beware of underfitting

Aside from overfitting (model is complicated and it may fail to generalize to new data), underfitting could also be a problem. Underfitting is when the model has poor performance.

To better illustrate this, let's compare the results of Underfitting, Balanced (just right), and Overfitting:

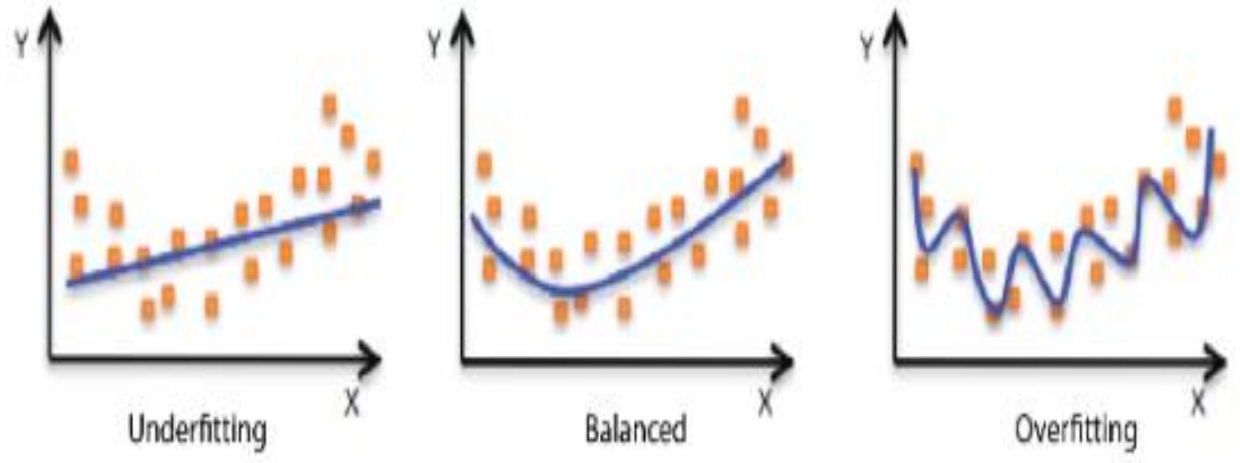


Image source: https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

The first example seems it's a good enough model. However, we can still improve the model (improve its accuracy) by applying Cross-Validation, wise feature selection, and other methods. If we do these, we can achieve a better accuracy which may result to the second example (Balanced).

Another way to accomplish that is by adding domain-specific features, which is somehow related to wise feature selection. That's because when there's underfitting, the model failed to capture the relationship between the input features and output. Perhaps there's no relationship between them at all in the first place, which is why the model has poor performance.

That's why **domain knowledge** is also essential in successfully creating a machine learning model. In a machine learning team, one or more members should have an expertise in the subject area. For instance, if the project is about predicting the properties of molecules using machine learning, the team should have members that are experts in chemistry and physics.

In other words, data availability and machine learning algorithms are not sufficient to creating an excellent model and analysis. What works best is to make domain knowledge equally important as the two. This way, the project is in a better context and the model will perform well.

Summary of improving your model’s performance

Here are the ways to improve your model’s performance:

1. Get more data (more data within a feature or more relevant features to include in the model)
2. Apply ensemble learning (use combinations of algorithms then averaging the results, e.g. Decision Tree vs Random Forest)
3. Use a cheat sheet for quick reference (and so you won’t miss any critical steps)
4. Wise feature selection (to solve underfitting and give the project a better context)
5. Use dimensionality reduction (given that relevant features were included first, then “summarize” the features for easier processing and visualization)
6. Apply Cross-Validation (rotation of training and test sets)
7. Use domain knowledge and make it as important as getting more data and applying complex machine learning algorithms)

It seems a simple checklist because the advice is straightforward. Nevertheless, in practice, it might take several weeks and months before a team can arrive at a satisfactory machine learning model and results. For instance, gathering more data could be expensive and time consuming. Moreover, even with all that efforts, the accuracy might only improve by 0.00001%.

During the project, it’s common to encounter and make numerous errors. It’s totally a repetitive and iterative process. It’s likely machine learning developers and data scientists try different approaches before finalizing and closing the project. In the near future, they might revisit the project and evaluate its performance (and see where are the potential improvements).

In other words, expect a long and complicated journey especially when you’re tackling massive datasets and complicated subject matter. You may even dive into advanced approaches, which we’ll discuss in the next chapter.

We’ll give an overview about those “advanced approaches” because actually, they are already practiced widely in many companies and organizations. Let us talk about neural networks and deep learning.

Neural Networks & Deep Learning

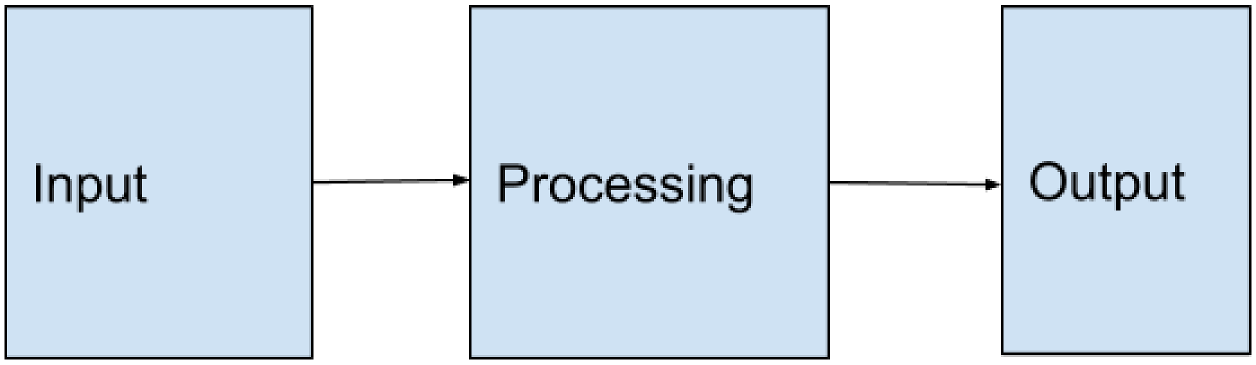
In the previous chapters we've focused on machine learning using Python, numpy, pandas, and scikit-learn. We can take it a step further by somehow emulating how the brain works. After all, the human brain is one of the best examples of how "learning" actually happens.

Although how the brain works *exactly* is still a huge mystery (our minds may not be able to understand how our brains work), we can still get an inspiration from what we know so far about it.

Basically we learn from experience and seeing examples. We also derive conclusions and recognize patterns from our examples and experiences. It's the core of machine learning even when we're using the machine learning algorithms we've already discussed.

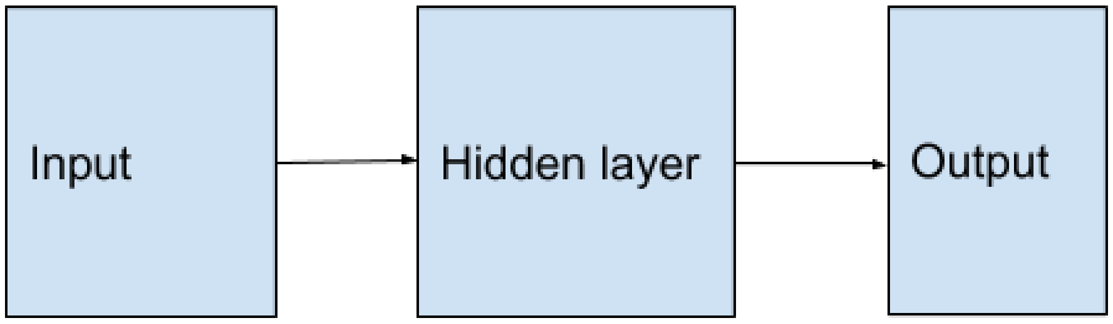
About artificial neural networks

Artificial neural networks (ANNs) are no exceptions. We also require enough data and examples for a meaningful learning. What makes ANNs different is that the structure could be different. That's because it's heavily inspired by how the neurons and our brain work.



That's a simple explanation of how the neurons work. Our neurons receive inputs and processing occurs. Then, the output comes out with a final result or that output can be used by other neurons (the output becomes the input in this case).

Artificial neural networks (ANNs) are loosely inspired from this. Generally here's how it works:



It’s the same with our basic model how the brain neurons work. The difference is that instead of Processing, we have the so-called Hidden Layer. It sounds mysterious but what it really means is just it’s neither an Input or Output.

In practice though, here’s how ANNs actually look like:

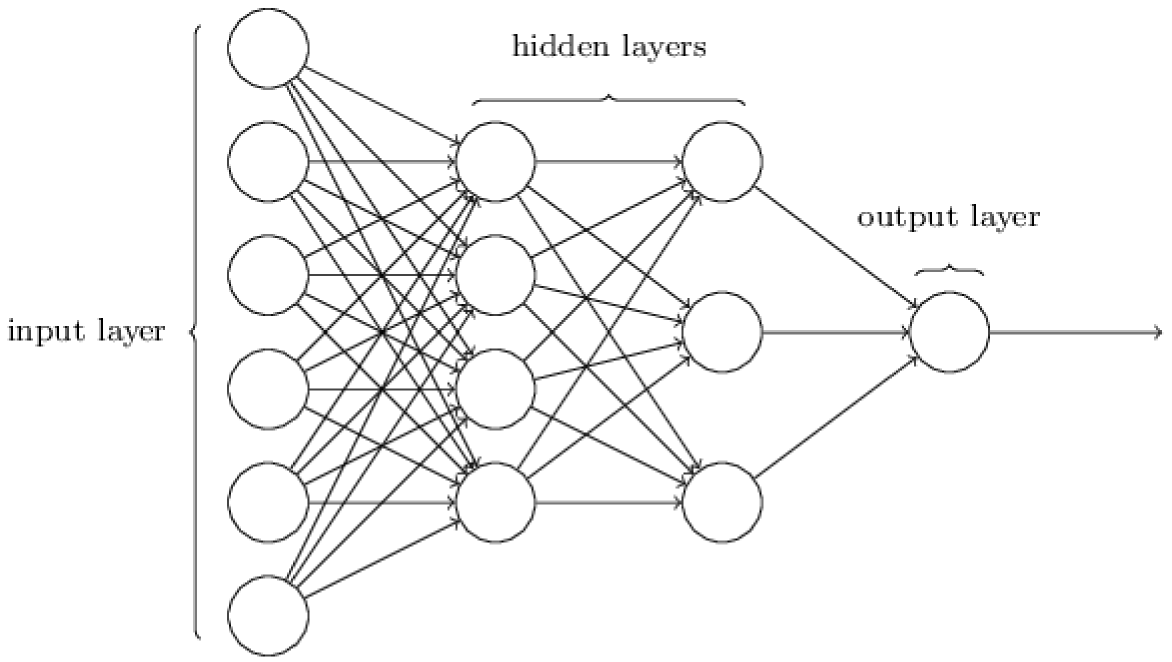


Image source: http://neuralnetworksanddeeplearning.com/chap1.html

Notice there is 1 column of Input, 2 columns of Hidden Layers, and finally just 1 Output. Also, notice the connections and networks between the layers. It’s also loosely based on how the brain works (we also have neural networks but they’re far more complicated than what computers have now).

In other words, the ANNs are our attempt to recreate the brain. Scientists and researchers are still far from it, but the ANNs now could be enough for some purposes.

This particular use of ANNs is called Deep Learning. It’s a subfield of machine learning that specializes in applying algorithms inspired by brain neurons.

Creating an artificial neural network in Python

Without using more libraries and sophisticated tools, we can already create a 2-layer artificial neural network with just Python and numpy. Here's how it could look like in Python code:

```
import numpy as npdef nonlin(x,deriv=False):

    if(deriv==True):

        return x*(1-x)

    return 1/(1+np.exp(-x))


X = np.array([  [0,0,1],

                [0,1,1],

                [1,0,1],

                [1,1,1] ])


y = np.array([[0,0,1,1]]).T


np.random.seed(1)


syn0 = 2*np.random.random((3,1)) - 1

for iter in xrange(10000):

    l0 = X

    l1 = nonlin(np.dot(l0,syn0))

    l1_error = y - l1

    l1_delta = l1_error * nonlin(l1,True)

    syn0 += np.dot(l0.T,l1_delta)

print("Output After Training:")

print(l1)
```

It looks complex and abstract. We require advanced Python knowledge to recreate an ANN from scratch.

Thankfully, we don't have to do all that. As with our use of numpy, pandas, and scikit-learn, there are also libraries for building and applying ANNs. We can also import a particular library and use that so we can save ourselves from tons of lines of code.

One of the most popular libraries is TensorFlow by Google. From its website:

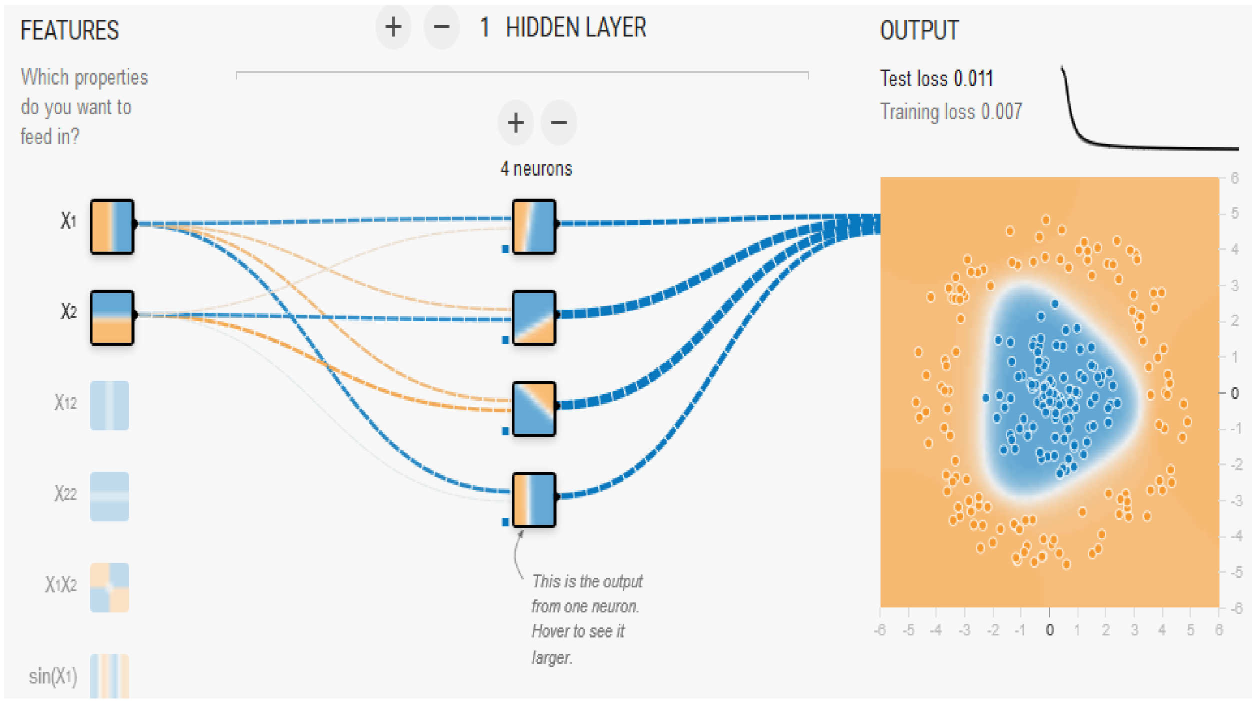
"TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them…"

It's free and we can use it anytime just like scikit-learn and other Python libraries. Aside from TensorFlow, there are other libraries such as the following:

- Caffe
- Theano
- Keras
- MXNet

Why TensorFlow

It's from Google where you can expect it will be here for a long time. This means learning how to use TensorFlow could be very worth your time and effort. Also, in the future you might easily find it easier to implement especially in large projects (there's Google Cloud Platform which fully supports TensorFlow).
For this chapter let's focus on TensorFlow and how to use it. Let's start with a simple illustration so you know the context before diving deep into the code.



Source: https://playground.tensorflow.org/

The first column is the Inputs (2 Features), the second column is the Hidden Layer, and finally the third layer is the Output.

It's a Classification task wherein you "feed" the two features into the Hidden Layer with 4 neurons. The Output is that the data points are now already classified into two groups (Blue points under the Blue area and the rest in Orange).

On the surface it all seems easy because all work is done in the background. However, the background processes and mathematics could be long and rigorous.

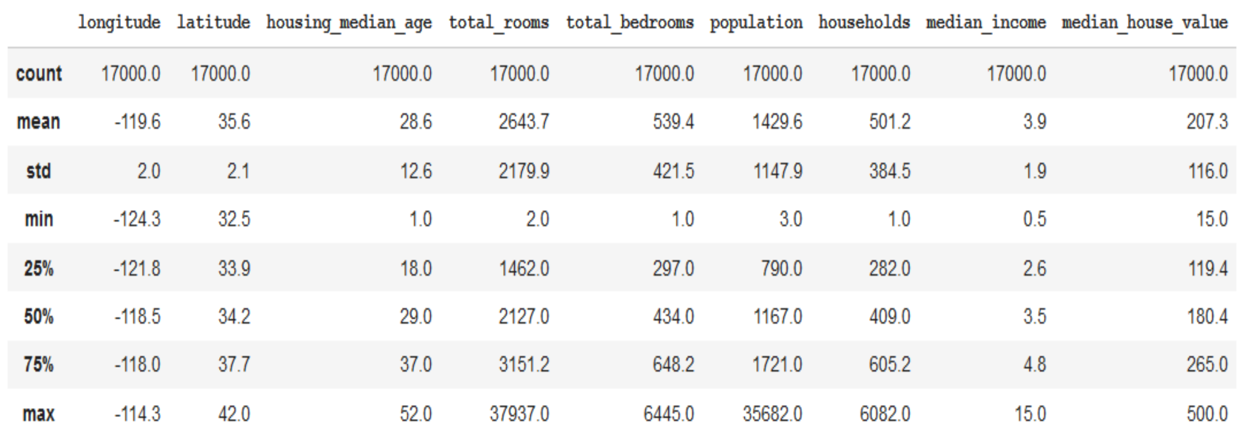
For instance, there's also the minimization of the "loss" until we get an optimal result or convergence. What this means is it's also an iterative approach wherein the Output also "gives back" a report about the loss and error. The neurons in the Hidden Layer will then be recalibrated to get a lower loss value. This process goes on and on until there's convergence or minimal change in the loss value while the training still progresses.

In short, both forward and backward processes occur between Hidden Layer and Output Layer until an "equilibrium" is accomplished. All of these are happening in the background. These processes will take time and computational power especially if you're dealing with massive datasets.

An example using TensorFlow

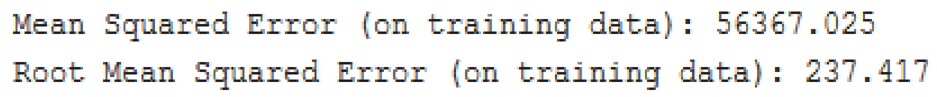
Let us get an idea how to use TensorFlow. We will use the California Housing Dataset (with features such as total _rooms and output of median_house_value). This is an example from the Machine Learning Crash Course by Google:

```
import math



from IPython import display

from matplotlib import cm

from matplotlib import gridspec

from matplotlib import pyplot as plt

import numpy as np

import pandas as pd

from sklearn import metrics

import tensorflow as tf

from tensorflow.python.data import Dataset



tf.logging.set_verbosity(tf.logging.ERROR)

pd.options.display.max_rows = 10

pd.options.display.float_format = '{:.1f}'.format

dataset   =   pd.read_csv("https://storage.googleapis.com/mledu-datasets/california_housing_train.csv", 
sep=",")

dataset = dataset.reindex(np.random.permutation(california_housing_dataframe.index))

dataset[ "median_house_value" ] /= 1000.0

dataset.describe()
```



	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	17000.0	17000.0	17000.0	17000.0	17000.0	17000.0	17000.0	17000.0	17000.0
mean	-119.6	35.6	28.6	2643.7	539.4	1429.6	501.2	3.9	207.3
std	2.0	2.1	12.6	2179.9	421.5	1147.9	384.5	1.9	116.0
min	-124.3	32.5	1.0	2.0	1.0	3.0	1.0	0.5	15.0
25%	-121.8	33.9	18.0	1462.0	297.0	790.0	282.0	2.6	119.4
50%	-118.5	34.2	29.0	2127.0	434.0	1167.0	409.0	3.5	180.4
75%	-118.0	37.7	37.0	3151.2	648.2	1721.0	605.2	4.8	265.0
max	-114.3	42.0	52.0	37937.0	6445.0	35682.0	6082.0	15.0	500.0

```
my_feature = dataset[[ “total_rooms]]

feature_columns = [tf.feature_column.numeric_column(”total_rooms”)]

targets = dataset[”median_house_value”]

my_optimizer=tf.train.GradientDescentOptimizer(learning_rate=0.0000001)

my_optimizer = tf.contrib.estimator.clip_gradients_by_norm(my_optimizer, 5.0)

linear_regressor = tf.estimator.LinearRegressor(

    feature_columns=feature_columns,

    optimizer=my_optimizer

)

def my_input_fn(features, targets, batch_size=1, shuffle=True, num_epochs=None):

    features = {key:np.array(value) for key,value in dict(features).items()}

    ds = Dataset.from_tensor_slices((features,targets))

    ds = ds.batch(batch_size).repeat(num_epochs)

    if shuffle:

      ds = ds.shuffle(buffer_size=10000)

    features, labels = ds.make_one_shot_iterator().get_next()

    return features, labels

_ = linear_regressor.train(

    input_fn = lambda:my_input_fn(my_feature, targets),

    steps=100
)

prediction_input_fn =lambda: my_input_fn(my_feature, targets, num_epochs=1, shuffle=False)


predictions = linear_regressor.predict(input_fn=prediction_input_fn)


predictions = np.array([item['predictions'][0] for item in predictions])

mean_squared_error = metrics.mean_squared_error(predictions, targets)

root_mean_squared_error = math.sqrt(mean_squared_error)

print("Mean Squared Error (on training data): %0.3f" % mean_squared_error)

print("Root Mean Squared Error (on training data): %0.3f" % root_mean_squared_error)
```



```
Mean Squared Error (on training data): 56367.025
Root Mean Squared Error (on training data): 237.417
```

```
def train_model(learning_rate, steps, batch_size, input_feature="total_rooms"):

  periods = 10

  steps_per_period = steps / periods


  my_feature = input_feature

  my_feature_data = dataset[[my_feature]]

  my_label = "median_house_value"

  targets = dataset[my_label]


  feature_columns = [tf.feature_column.numeric_column(my_feature)]

  training_input_fn = lambda:my_input_fn(my_feature_data, targets, batch_size=batch_size)

  prediction_input_fn = lambda: my_input_fn(my_feature_data, targets, num_epochs=1, shuffle=False)


  my_optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)

  my_optimizer = tf.contrib.estimator.clip_gradients_by_norm(my_optimizer, 5.0)
```

```
linear_regressor = tf.estimator.LinearRegressor(
    feature_columns=feature_columns,
    optimizer=my_optimizer
)
plt.figure(figsize=(15, 6))
plt.subplot(1, 2, 1)
plt.title("Learned Line by Period")
plt.ylabel(my_label)
plt.xlabel(my_feature)
sample = dataset.sample(n=300)
plt.scatter(sample[my_feature], sample[my_label])
colors = [cm.coolwarm(x) for x in np.linspace(-1, 1, periods)]

print("Training model...")
print("RMSE (on training data):")
root_mean_squared_errors = []
for period in range (0, periods):
  linear_regressor.train(
      input_fn=training_input_fn,
      steps=steps_per_period
  )
  predictions = linear_regressor.predict(input_fn=prediction_input_fn)
  predictions = np.array([item['predictions'][0] for item in predictions])

  root_mean_squared_error = math.sqrt(
      metrics.mean_squared_error(predictions, targets))

  print("  period %02d : %0.2f" % (period, root_mean_squared_error))
```

```
    root_mean_squared_errors.append(root_mean_squared_error)
    y_extents = np.array([0, sample[my_label].max()])

    weight = linear_regressor.get_variable_value('linear/linear_model/%s/weights' % input_feature)[0]
    bias = linear_regressor.get_variable_value('linear/linear_model/bias_weights')

    x_extents = (y_extents - bias) / weight
    x_extents = np.maximum(np.minimum(x_extents,
                                      sample[my_feature].max()),
                           sample[my_feature].min())
    y_extents = weight * x_extents + bias
    plt.plot(x_extents, y_extents, color=colors[period])
  print("Model training finished.")

  plt.subplot(1, 2, 2)
  plt.ylabel('RMSE')
  plt.xlabel('Periods')
  plt.title("Root Mean Squared Error vs. Periods")
  plt.tight_layout()
  plt.plot(root_mean_squared_errors)
  calibration_data = pd.DataFrame()
  calibration_data["predictions"] = pd.Series(predictions)
  calibration_data["targets"] = pd.Series(targets)
  display.display(calibration_data.describe())

  print("Final RMSE (on training data): %0.2f" % root_mean_squared_error)
```

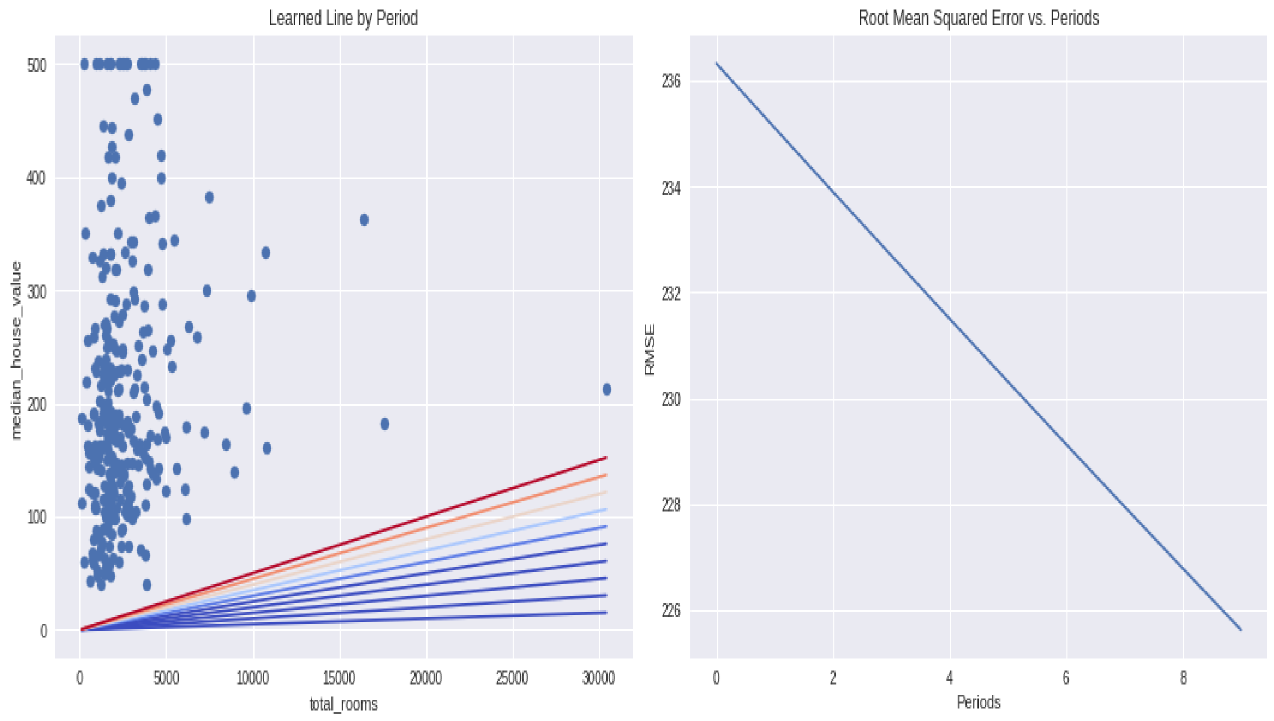
Now let's call on our function (train_model) and pass it the learning rate, steps, and batch size.

```
train_model(
    learning_rate=0.00001,
    steps=100,
    batch_size=1
)
```

```
Training model...
RMSE (on training data):
  period 00 : 236.32
  period 01 : 235.11
  period 02 : 233.90
  period 03 : 232.70
  period 04 : 231.50
  period 05 : 230.31
  period 06 : 229.13
  period 07 : 227.96
  period 08 : 226.79
  period 09 : 225.63
Model training finished.
```

	predictions	targets
count	17000.0	17000.0
mean	13.2	207.3
std	10.9	116.0
min	0.0	15.0
25%	7.3	119.4
50%	10.6	180.4
75%	15.8	265.0
max	189.7	500.0

```
Final RMSE (on training data): 225.63
```



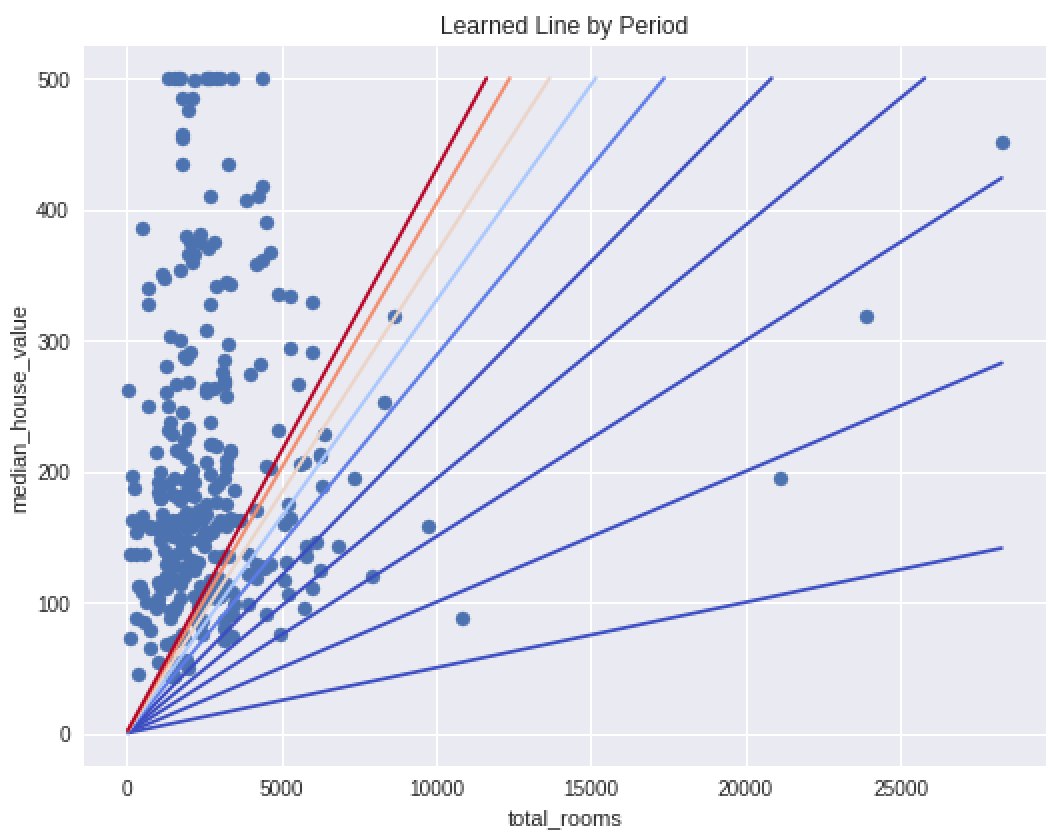
Our model performs poorly. The lines are far off from the data points. Let's try again by changing the parameters (learning rate, steps, batch size):

```
train_model(
    learning_rate=0.00002,
    steps=500,
    batch_size=5
)
```

```
Training model...
RMSE (on training data):
  period 00 : 225.63
  period 01 : 214.42
  period 02 : 204.04
  period 03 : 195.69
  period 04 : 187.86
  period 05 : 180.80
  period 06 : 175.66
  period 07 : 172.08
  period 08 : 169.21
  period 09 : 167.79
Model training finished.
```

	predictions	targets
count	17000.0	17000.0
mean	113.7	207.3
std	93.7	116.0
min	0.1	15.0
25%	62.9	119.4
50%	91.5	180.4
75%	135.5	265.0
max	1631.3	500.0

```
Final RMSE (on training data): 167.79
```



We got a lot better result just by just changing the parameters. This is called **Hyperparameter Tuning** which is about adjusting the parameters until we get a good model or accuracy.

Notice that the code using TensorFlow looks a lot more complicated than with Python and scikit-learn. However, remember, the whole process was already simplified because we're using a library. The code will get more abstract and more complex if we do it all from scratch.

The important thing to remember here is to start with something simple first (Python, scikit-learn, small dataset) then scale it up and use sophisticated tools when required. Also, choose libraries with extensive documentations and active communities. This way, whenever you encounter a problem, the process becomes easier because you can just search and adapt the solution to your own case.

Conclusion

Machine learning has earned a great deal of importance over the last few years. People from different fields have begun to research how they can incorporate machine learning in their field of study. Therefore, it is of utmost importance to understand what machine learning is and how it is linked to different fields of study.

This book provides you with all the information you would need on machine learning. You will gather an idea on the different subjects that are linked to machine learning and some facts about machine learning that make it an interesting subject to learn. Machine learning has been linked to artificial intelligence and data mining since the beginning of time. Therefore, it is important to gather some information about these fields of study too.

Thank you for purchasing the book. I hope you have gathered all the information necessary for machine learning.

Sources & References

Software, libraries, & programming language

- Python (https://www.python.org/)
- Anaconda (https://anaconda.org/)
- Numpy (http://www.numpy.org/)
- Pandas (https://pandas.pydata.org/)
- Matplotlib (https://matplotlib.org/)
- Scikit-learn (http://scikit-learn.org/)
- TensorFlow (https://www.tensorflow.org/)

Datasets

- Kaggle (https://www.kaggle.com/datasets)
- California Housing Dataset (https://www.kaggle.com/camnugent/california-housing-prices/data)
- UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.html)
- Wine Quality Dataset (https://archive.ics.uci.edu/ml/datasets/Wine+Quality)
- Iris Dataset (https://archive.ics.uci.edu/ml/datasets/Iris)

Online books, tutorials, & other references

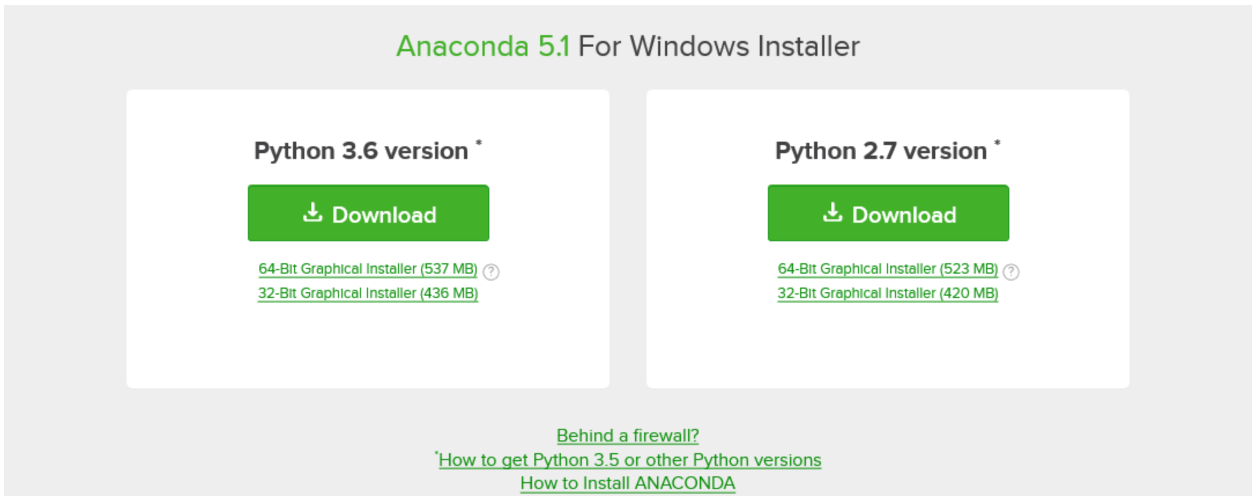
- Neural Networks and Deep Learning (http://neuralnetworksanddeeplearning.com/)
- Model Fit: Underfitting vs Overfitting (https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html)
- Overfitting (https://en.wikipedia.org/wiki/Overfitting)
- A Neural Network Program (https://playground.tensorflow.org/)
- Machine Learning Crash Course by Google (https://playground.tensorflow.org/)
- Choosing the Right Estimator (http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)
- Cross-validation: evaluating estimator performance (http://scikit-learn.org/stable/modules/cross_validation.html)

Bonus Chapter: Anaconda Setup & Python Crash Course

How to Download & Install Anaconda

Go to Anaconda Download page: https://www.anaconda.com/download/

The site will automatically detect your operating system (mine is Windows) and show the most appropriate installer for you. For example, if you scroll down you might see a page that looks like this:


Anaconda 5.1 For Windows Installer
Python 3.6 version *
Download
64-Bit Graphical Installer (537 MB)
32-Bit Graphical Installer (436 MB)
Python 2.7 version *
Download
64-Bit Graphical Installer (523 MB)
32-Bit Graphical Installer (420 MB)
Behind a firewall?
*How to get Python 3.5 or other Python versions
How to Install ANACONDA

Choose the Python 3 version. Python 2 is an older version. It's highly recommended to **choose Python 3** because all the developments are going there.

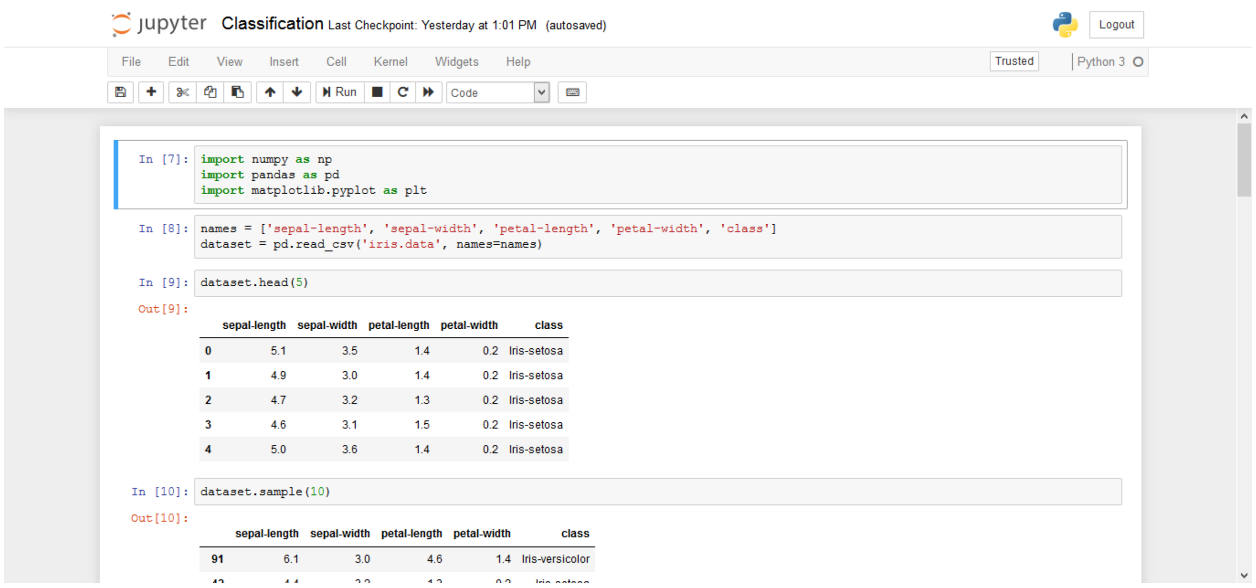
Once the Download is completed (could be 500+ MB file), you can do a standard installation and accept the default. You can also follow a more detailed instruction by visiting this page: https://docs.anaconda.com/anaconda/install/windows

The installation will already include the most important things you'll need for data science and machine learning. Later on if you're doing advanced work you might need to install other packages. Anaconda has a comprehensive documentation for that. You can also search for "[package name] + anaconda" and likely you'll see relevant search results.

Using Jupyter Notebook

Usually this is already included when you install Anaconda. Jupyter Notebook is an awesome tool that allows us to really create a virtual notebook. With that we can include code, explanations, tables, figures, and notes using just our browser.

This is what a "notebook" would look like:


jupyter
Classification Last Checkpoint: Yesterday at 1:01 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [7]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt

In [8]: names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
        dataset = pd.read_csv('iris.data', names=names)

In [9]: dataset.head(5)
```

Out[9]:

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

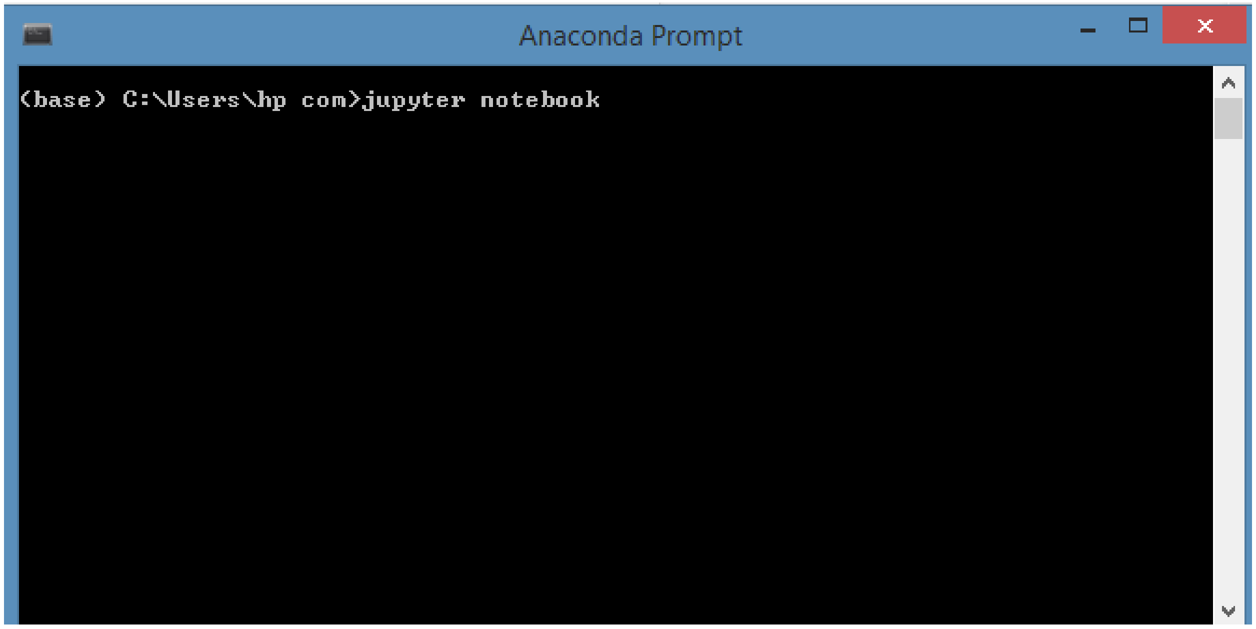
```
In [10]: dataset.sample(10)
```

Out[10]:

	sepal-length	sepal-width	petal-length	petal-width	class
91	6.1	3.0	4.6	1.4	Iris-versicolor

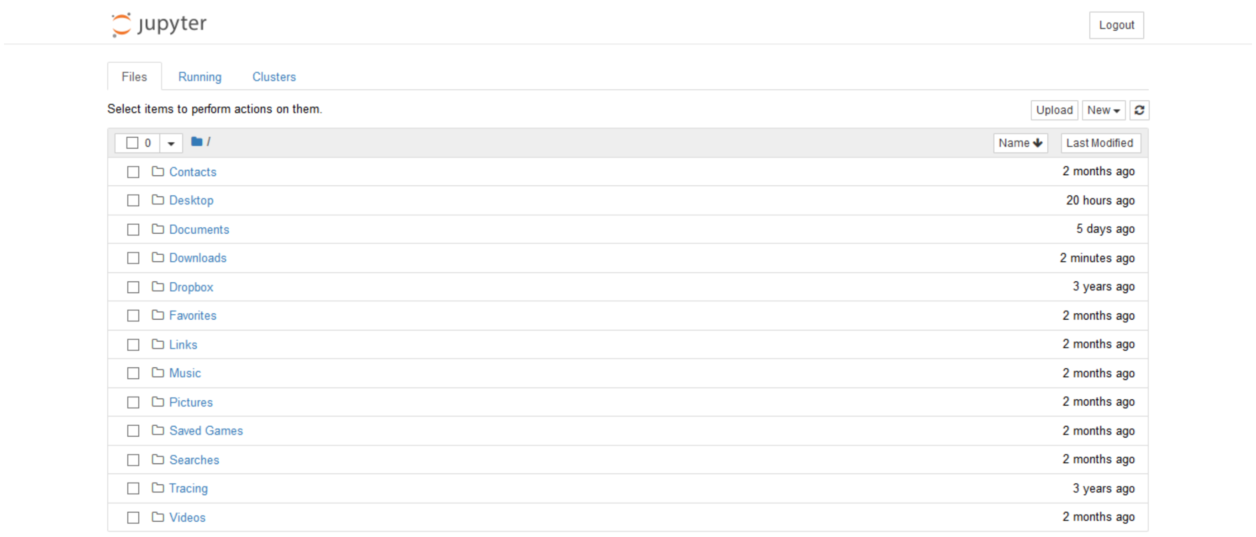
In just one page you can perform a full data analysis with all the code, figures, and notes included. Many online instructors actually use Jupyter Notebook for convenience and accessibility to learners.

To use Jupyter Notebook, one way is to open the Anaconda Prompt (type and search through your installed apps). Then, type **jupyter notebook**. Here's an example how to do it:



```
(base) C:\Users\hp com>jupyter notebook
```

Press enter and a new browser tab will appear. It will look something like this:



Select items to perform actions on them.

```
Contacts        2 months ago
Desktop         20 hours ago
Documents       5 days ago
Downloads       2 minutes ago
Dropbox         3 years ago
Favorites       2 months ago
Links           2 months ago
Music           2 months ago
Pictures        2 months ago
Saved Games     2 months ago
Searches        2 months ago
Tracing         3 years ago
Videos          2 months ago
```

You can click the folders and files just like how you normally use your computer. You can then choose a folder (where you want to put the file) and then click Python 3 to create a new notebook.

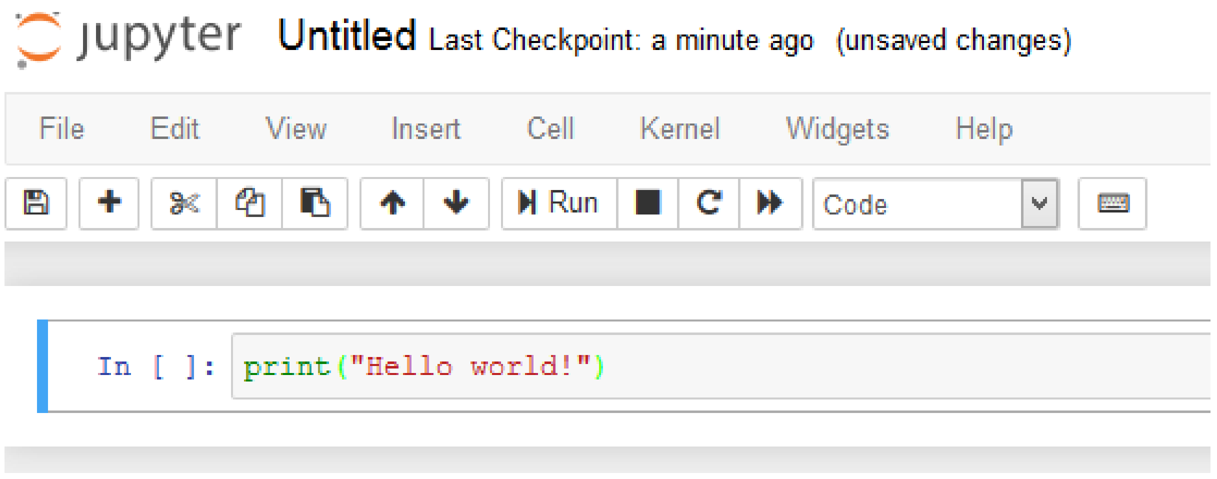
For more tutorials, you can always refer to their documentation at https://jupyter-notebook.readthedocs.io/en/stable/

Another tool that works like Jupyter Notebook is the Google Colab (https://colab.research.google.com/). It's getting popular because of the minimal setup and online collaboration. It's like having a Google Docs for coding.

Python Crash Course

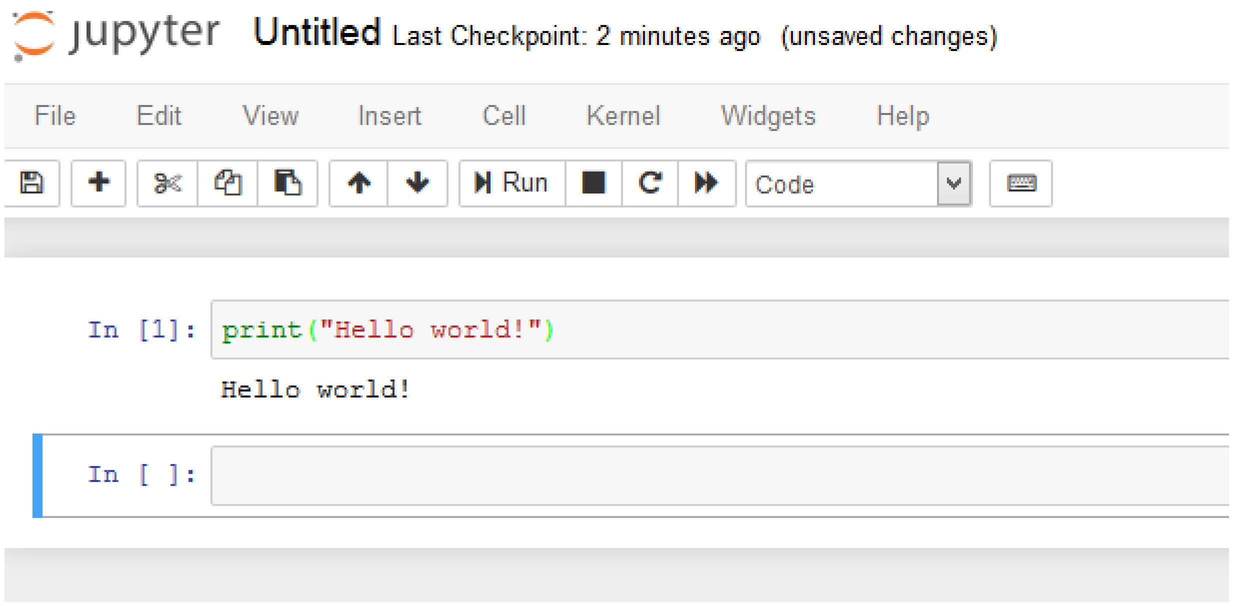
You won't be an instant Python expert after reading this. But this might be a good introduction for those with zero programming background.

If you follow the steps above for using Jupyter Notebook, you can already type Python code in the cells. For example, we can do `print( "Hello world!" )`


jupyter
Untitled Last Checkpoint: a minute ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
In [ ]: print("Hello world!")
```

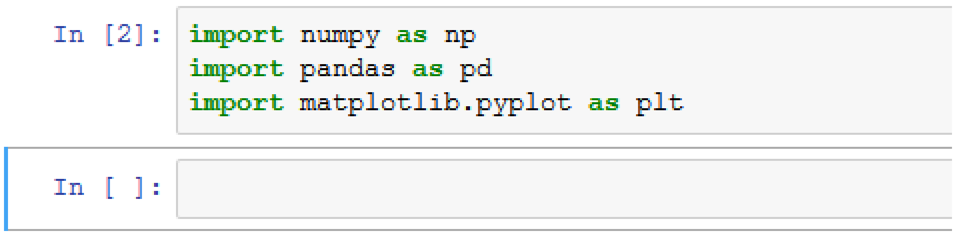
While the cursor is inside the cell, press SHIFT + ENTER. The result is it will print out **Hello world!** and create a new cell.


jupyter
Untitled Last Checkpoint: 2 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
In [1]: print("Hello world!")

        Hello world!

In [ ]:
```

We're just telling the computer to print **Hello world!** in a way that the computer understands (and in the Python way, other programming languages may have different ways).

Now let's focus on writing Python code for use in machine learning. Many projects start with importing libraries.



```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt

In [ ]:
```

We're just telling it to import the libraries so we can use them later when we write additional code. These libraries make life easier because there will be less code (they're not preloaded though so we have to import them first).

Initially, importing the numpy, pandas, and matplotlib are enough for data preprocessing and data visualization. In this book, we will import other libraries so we can do other specific things such as actual analysis.

After importing the libraries, the next step is to load and read the data. You can do this by typing:

```
dataset = pd.read_csv( “your_data.csv” )
```

Then pressing Shift + Enter to make it run in our cell.

What this does is we use pandas (we imported it earlier) to read the data (also getting access to it). After that, we can do further data processing and manipulation. For example we can do:

`dataset.describe()` to show summary of the data such as mean and standard deviation

`dataset.head(5)` to show the first 5 entries of the dataset

`dataset.hist( “feature_1” )` to create a histogram of a particular column/feature

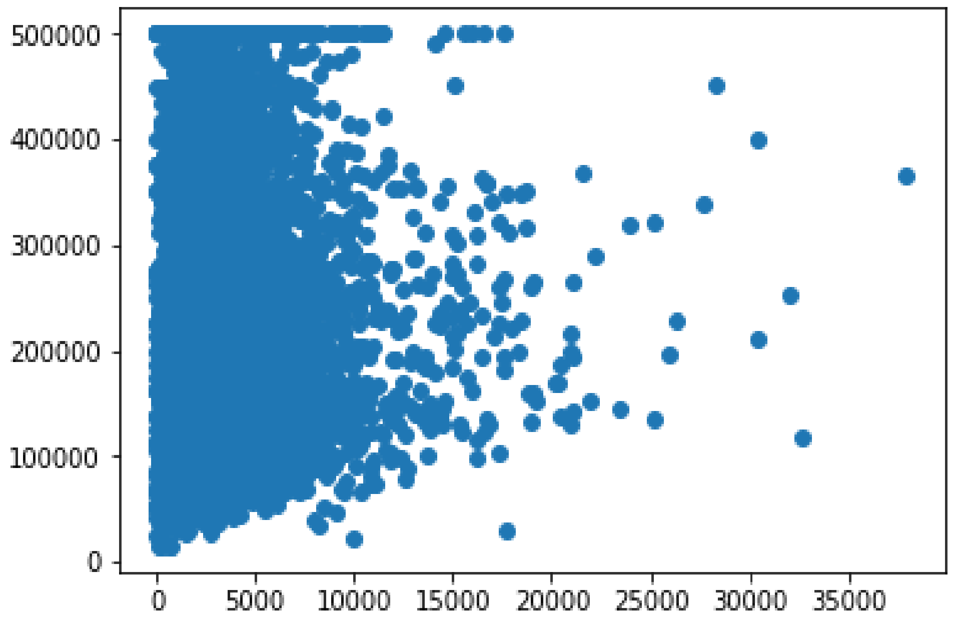
`X = dataset[[ “feature_1” ]].values` to assign X into the first feature

`y = dataset[[ “output]].values` to assign y as the output or target

We can also use the data to create scatterplots:

```
plt.scatter(X,y)
```

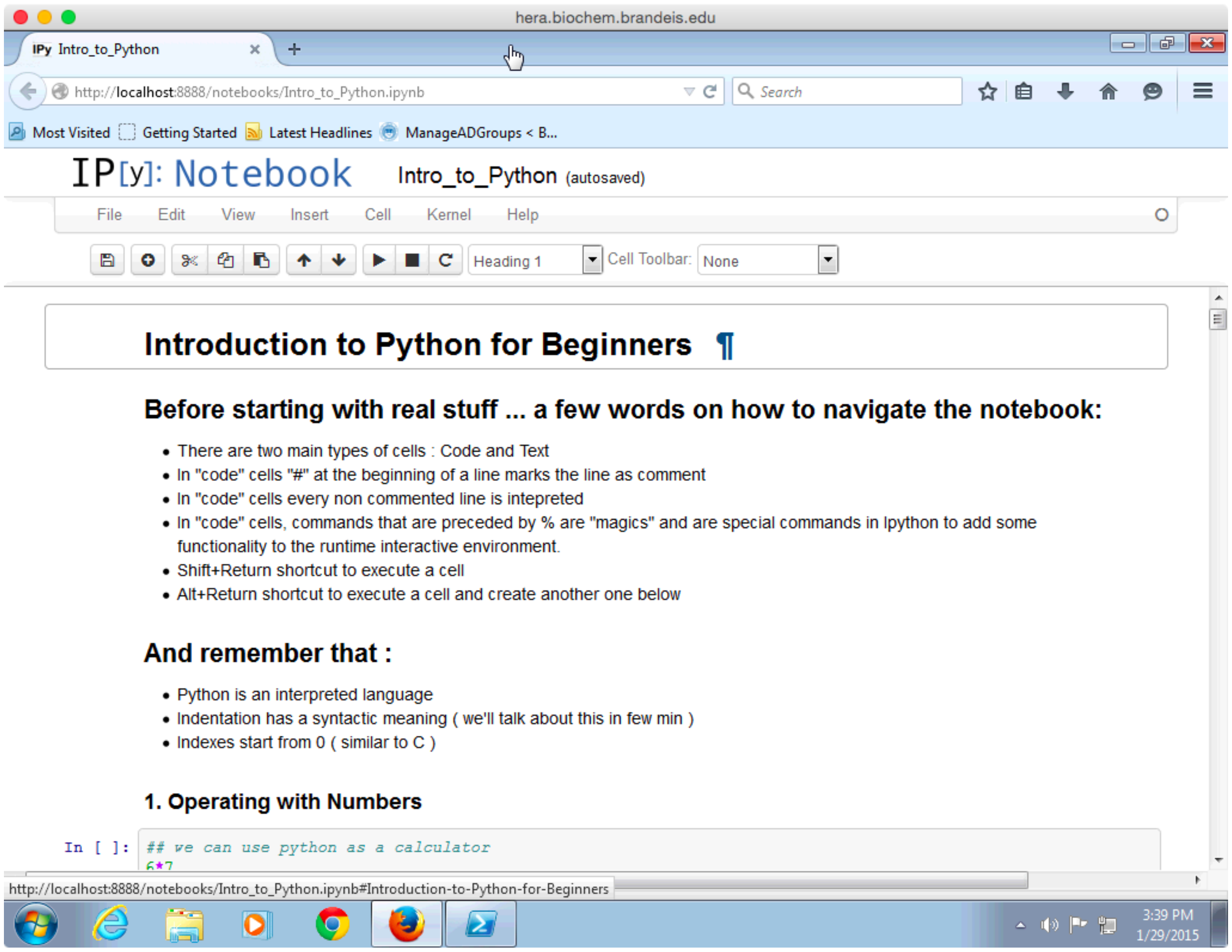
which will show something like this:



Just by reading Python code you can already get an idea of what it does. That's one of the core strengths of Python. The syntax is intuitive and you can learn it in a reasonable amount of time.

Go Forth and Do Machine Learning With IPython

IPython Notebook is a web-based interactive computational environment for creating **IPython notebooks**. An **IPython notebook** is a JSON document containing an ordered list of input/output cells which can contain code, text, mathematics, plots and rich media.


hera.biochem.brandeis.edu
IPy Intro_to_Python
http://localhost:8888/notebooks/Intro_to_Python.ipynb
Search
Most Visited
Getting Started
Latest Headlines
ManageADGroups < B...
IP[y]: Notebook
Intro_to_Python (autosaved)
File
Edit
View
Insert
Cell
Kernel
Help
Heading 1
Cell Toolbar:
None
http://localhost:8888/notebooks/Intro_to_Python.ipynb#Introduction-to-Python-for-Beginners
3:39 PM
1/29/2015

Introduction to Python for Beginners ¶

Before starting with real stuff ... a few words on how to navigate the notebook:

- There are two main types of cells : Code and Text
- In "code" cells "#" at the beginning of a line marks the line as comment
- In "code" cells every non commented line is intepreted
- In "code" cells, commands that are preceded by % are "magics" and are special commands in Ipython to add some functionality to the runtime interactive environment.
- Shift+Return shortcut to execute a cell
- Alt+Return shortcut to execute a cell and create another one below

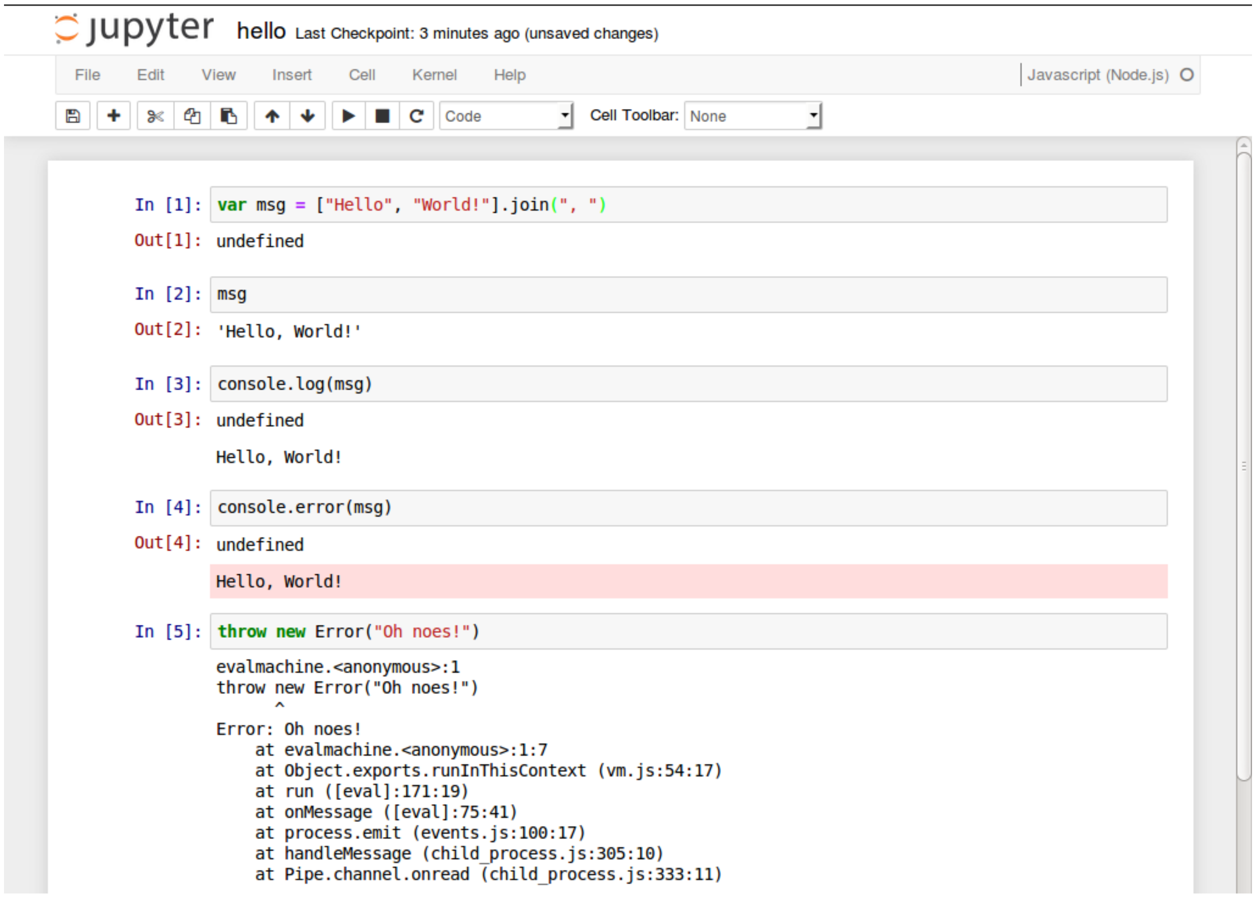
And remember that :

- Python is an interpreted language
- Indentation has a syntactic meaning (we'll talk about this in few min)
- Indexes start from 0 (similar to C)

1. Operating with Numbers

```
In [ ]: ## we can use python as a calculator
        6*7
```

It provides a shell with far more functionality than the standard Python shell, and it adds "magic functions" that allow you to (among other things) easily copy and paste code (which is normally complicated by the combination of blank lines and whitespace formatting) and run scripts from within the shell. Mastering IPython will make your life far easier. (Even learning just a little bit of IPython will make your life a lot easier.) Additionally, it allows you to create "notebooks" combining text, live Python code, and visualizations that you can share with other people, or just keep around as a journal of what you did.


jupyter
hello Last Checkpoint: 3 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Help
Javascript (Node.js)
Code
Cell Toolbar: None

```
In [1]: var msg = ["Hello", "World!"].join(", ")
Out[1]: undefined

In [2]: msg
Out[2]: 'Hello, World!'

In [3]: console.log(msg)
Out[3]: undefined
        Hello, World!

In [4]: console.error(msg)
Out[4]: undefined
        Hello, World!

In [5]: throw new Error("Oh noes!")
        evalmachine.<anonymous>:1
        throw new Error("Oh noes!")
              ^
        Error: Oh noes!
            at evalmachine.<anonymous>:1:7
            at Object.exports.runInThisContext (vm.js:54:17)
            at run ([eval]:171:19)
            at onMessage ([eval]:75:41)
            at process.emit (events.js:100:17)
            at handleMessage (child_process.js:305:10)
            at Pipe.channel.onread (child_process.js:333:11)
```

Not for Beginners

Implementing things “for beginners” is great for understanding how they work. But it’s generally not great for performance (unless you’re implementing them specifically with performance in mind), ease of use, rapid prototyping, or error handling.

In practice, you’ll want to use well-designed libraries that solidly implement the fundamentals.

Find Data

If you’re doing data science as part of your job, you’ll most likely get the data as part of your job (although not necessarily). What if you’re doing data science for fun? Data is everywhere, but here are some starting points:

Data sets for data Visualization Projects

1. FiveThirtyEight (Makes data available on GitHub)
2. BuzzFeed (Makes data available on GitHub)
3. Socrata Open Data

For larger data sets we can use

4. AWS public data sets

5. Google Public Data Sets
6. Wikipedia Data sets

For Machine Learning projects

7. Kaggle Data Sets
8. UCI Machine Learning Repository
9. Quandl

Data sets for Data Cleaning Projects

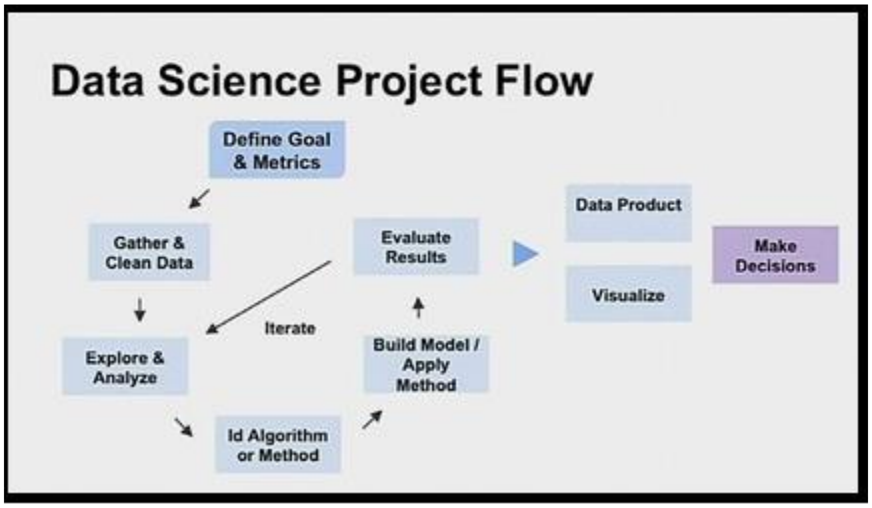
10. data.world
11. Data.gov
12. The World Bank Data Sets
13. The reddit /r/datasets
14. Academic torrents

For Streaming Data

15. Twitter
16. GitHub
17. Quantopian
18. Wunderground

Practicing Data Science

Python is very easy language to put interest in and among the most renowned programming languages. To get a hold of actual data science it will always be a good idea to work on actual interesting projects. A Data Science Project Flow can be illustrated in the following way:-



Some of the ideas, which you can go through, are

1. **Learning to mine on Twitter**

This is a simple project for beginners and is useful for understanding the importance of data science. When doing this project you will come to know what is trending. It can be a viral news being discussed or politics or some new movie. It will also teach you to integrate API in scripts for accessing any information on social media. It also exposes the challenges faced in mining social media.

2. **Identify your digits data set**

This is a type of training your program to recognize different digits. This problem is known as digit recognition problem. It is similar to camera detecting faces. It has as many as 7000 images with 28 X 28 size making it 31MB sizing.

3. **Loan Prediction data set**

The biggest user of data science among industries is insurance. It puts to use lots of analytics and data science methods. In this problem we are provided with enough information to work on data sets of insurance companies, the challenges to be faced, strategies that are to be used, the variables that influences the outcome etc. It has around classification problem with 615 rows and 13 columns.

4. **Credit Card Fraud Detection**

It is a classification problem where we are supposed to classify whether the transactions taking place on a card are legal or illegal. This does not have a huge data set since banks do not reveal their customer data due to privacy constraints.

5. **Titanic dataset from Kaggle:**

This dataset provides a good understanding of what a typical data science project will involve. The starters can work on the dataset in excel and the professionals can work on advanced tools to extract hidden information and algorithms to substitute some of the missing values in the dataset.