Data Science Nigeria

AI Invasion 2021

June 21 -25 | FREE 5-day Machine Learning Immersion across Nigeria

About Us

DSN
Data Science Nigeria

Data Science Nigeria is registered non-profit organization with a vision to build a world-class artificial intelligence knowledge, research and innovation ecosystem that delivers high impact & transformational research, business use applications, AI-first start-ups, employability and social good use cases; such that in 10years, Nigeria will become one of the top 10 AI talent/knowledge destinations with 20 GDP multiplier impact.

#1million_AITalents_in_10years

Our Achievements

AI Learning videos - publicly available contents from top international universities- curated in external drive and distributed free across the country to address the challenge of of poor internet

17,000+

Artificial intelligence for Starters free e-book download -99 use cases and expert views

1million+

Online participants in Data Science Courses from registration to at least one class

22,446

YouTube view in 18 online classes ran between March - May 2020 during the COVID-19 lockdown

95.2k

271,807

YouTube views of the 100Days of machine learning from Beginner to expert level classes

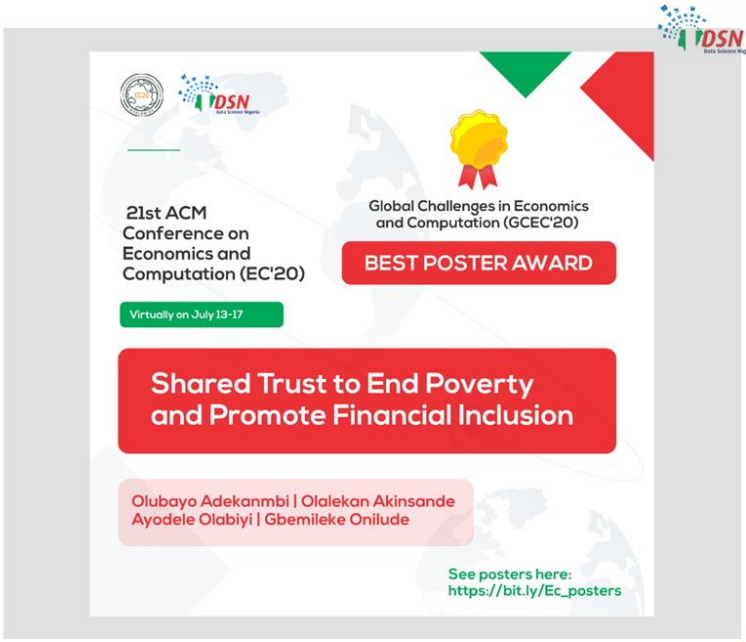
>3,000

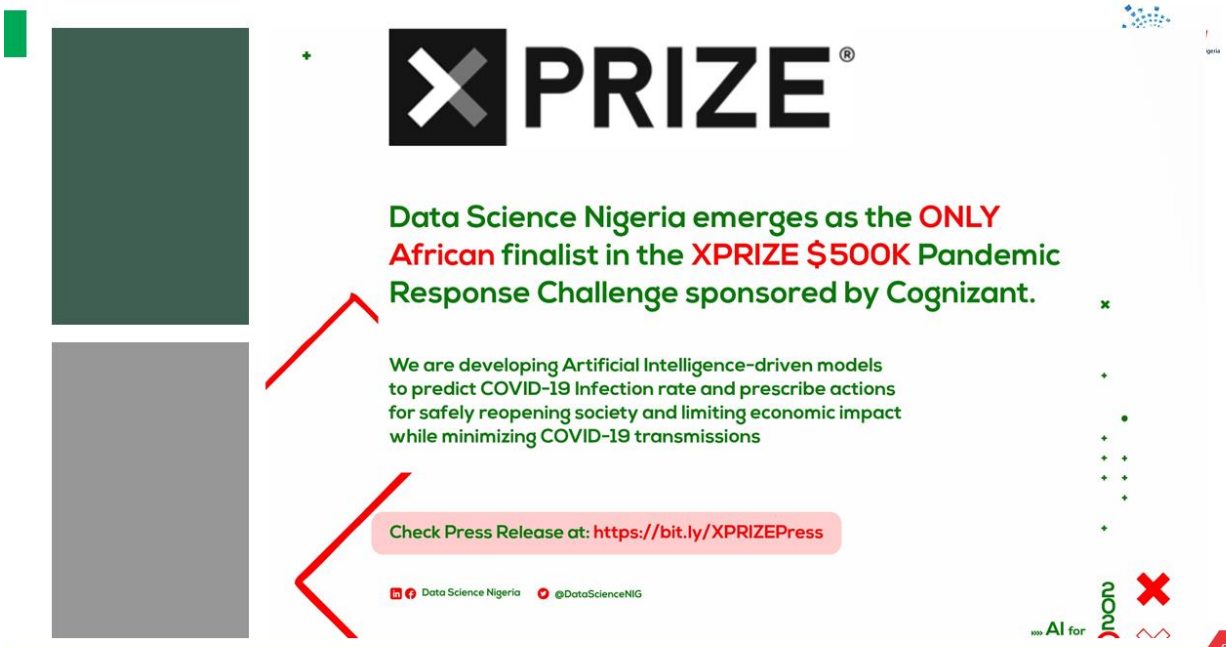
Hours of face-to-face free classes in Data Science and Machine learning applications organised in different states of Nigeria

No 1 Artificial Intelligence Learning Community and Solution Delivery network in Africa

Our work has been showcased, won awards and celebrated in Africa and in the US
Hundreds of thousands trained FREE via online, offline, face-to-face, content development and job placement

Best Academic Poster at the 21st ACM Conference on Economics Computer (EC' 20)- *the world's premier conference on the interface of economics and computer science*





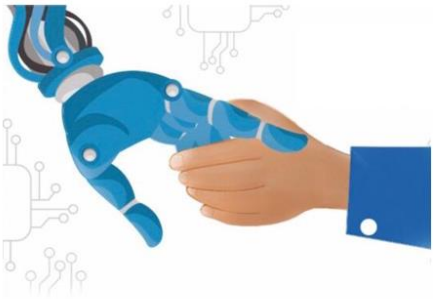
6

We are humbled to run the biggest AI learning programme in Africa with 30 unique platforms face to face, online, offline and hands-on and with impact on hundreds of thousands

Segments	Face to Face	Online	Offline Content	Hands-on
Beginners	• AI Wednesdays (Intro Python/ML) • AI Invasion • AI Everywhere		• AI for Starters ebook	• Industry Use Case Discussion sessions
Kids/Teens	• Pre-University Intro Python/ML • AI Summer School for Kids	• Weekly AI Slack Discussion Channels • Online Masterclass	• Python and Machine Learning for Grades 5-9	
Intermediates to Experts	• Deep Learning Nigeria • AI Bootcamp	• 100 Days of Machine/Deep Learning via YouTube • AI Mentorship • AI Dial-in Masterclass	• AI Knowledge Box	• DataHack • Data Scientists On Demand • On-project Internship
Higher Institution of Learning Students	• AI+ Clubs • Intercampus Machine Learning Competition • AI+ Researchers Forum			• AI MiniLabs and Tools
Professionals/C-level Executives/ Government agencies	• Business Analytics for Professionals • AI for Executives • AI Summit		• Ai+ Professional meet-up • Ladies in AI	• AI Project participation

8

Focus on the future by releasing first AI book for kids in Africa



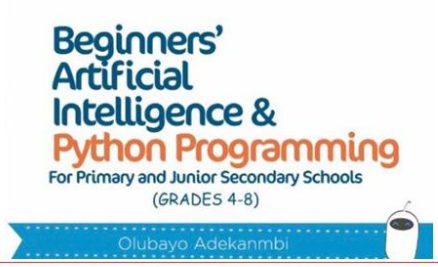
Amazon Kindle: https://amzn.to/3blD9b7

Amazon Paperback: https://amzn.to/33PfptG

Apple iBook: https://apple.co/3aeANL6

Google Play: https://bit.ly/AIBook_Play

Jumia: http://bit.ly/AI_BookJumia



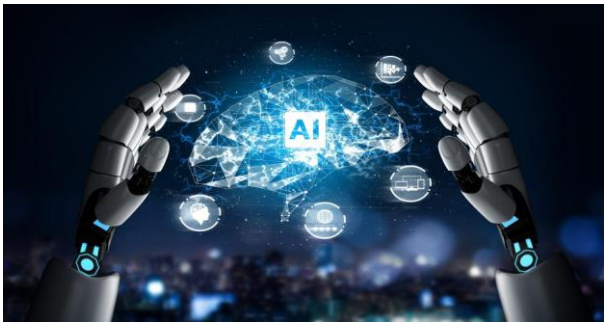
AI Book secured multiple global endorsement

Learn more about Data Science Nigeria

1. Official Website: https://www.datasciencenigeria.org/
2. AI+ Communities: Click here
3. AI+ for Kids and Teens Community: Click here
4. AI Book Purchase: Click here
5. Paid professional courses, trainings and certifications: Click here
6. Social Media
    - Twitter: https://twitter.com/DataScienceNIG
    - LinkedIn: https://ng.linkedin.com/in/datasciencenigeria
    - Instagram: https://www.instagram.com/datasciencenigeria/
    - Facebook: https://facebook.com/datasciencenig/
7. Annual Report: Click here

DAY 1

Artificial Intelligence, Machine Learning and Data Science



Introduction to Artificial Intelligence

- Introduction
- What is Artificial Intelligence?
- What is Machine Learning?
- What is Data Science?
- Artificial Intelligence Applications
- Machine Learning Use Cases/Applications

Introduction

Artificial intelligence (AI) is undoubtedly the hottest buzzword today and for all the good reasons. Over the past decade AI truly became a factor of production, with the potential to introduce new sources of growth and change the way work is done across industries.
The function and popularity of Artificial Intelligence are soaring by the day. Artificial intelligence is the ability of a system or a program to think and learn from the experience. AI has significantly evolved over the past few years and has found its applications in almost every sector.

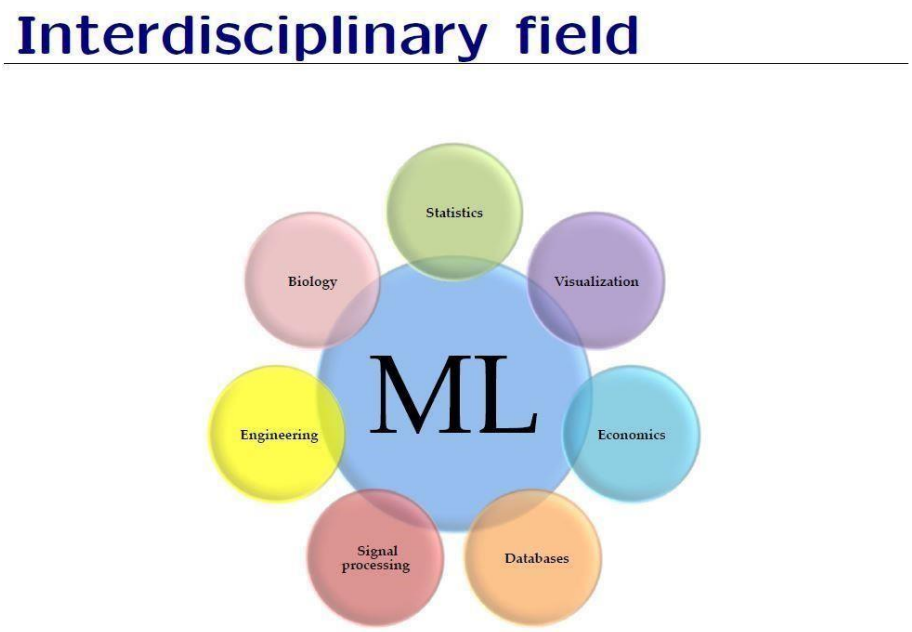
What is Artificial Intelligence?

Artificial Intelligence (AI) is the machine-displayed intelligence that simulates human behavior or thinking and can be trained to solve specific problems. AI is a combination of Machine Learning techniques and Deep Learning. AI models that are trained using vast volumes of data can make intelligent decisions. We will consider how AI is used in different domains.

What is machine learning?

Machine learning is the subset of artificial intelligence that involves the study and use of algorithms and statistical models for computer systems to perform specific tasks without human interaction. Machine learning models rely on

patterns and inference instead of manual human instruction. Most any task that can be completed with a data-defined pattern or set of rules can be done with machine learning.



Spam Detection: When Google Mail detects your spam mail automatically, it is as a result of applying machine learning techniques.

Link for more Reading

Credit Card Frauds: Identifying 'unusual' activities on a credit card is often a machine-learning problem involving anomaly detection.

Link for more Reading

Product Recommendation: Whether Netflix was suggesting you watch House of Cards or Amazon was insisting you finally buy those Bose headphones, it's not magic, but machine learning at work.

Link for more Reading

Medical Diagnosis: Using a database of symptoms and treatments of patients, a popular machine learning problem is to predict if a patient has a particular illness.

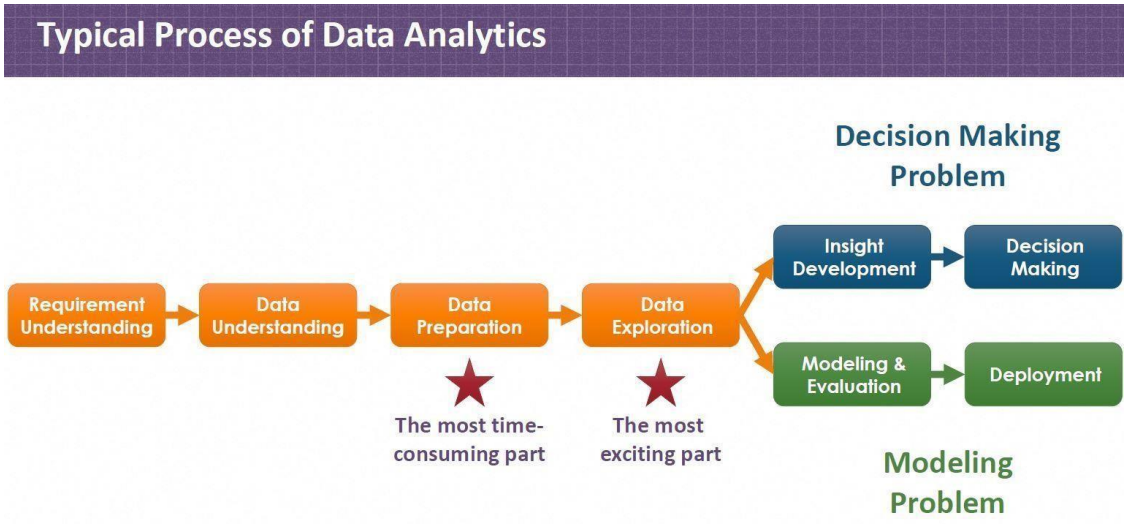
Link for more Reading

Face Detection: When Facebook automatically recognizes faces of your friends in a photo, a machine learning process is what's running in the background.

Link for more Reading

Customer Segmentation: Using the data for usage during a trial period of a product, identifying the users who will go onto subscribe for the paid version is a learning problem.

Link for more Reading



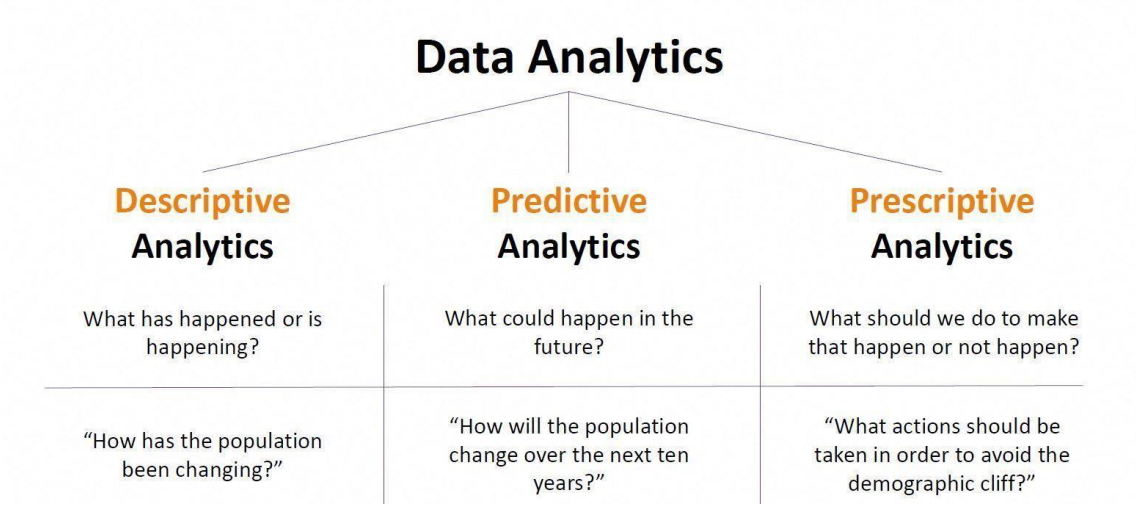
Types of Machine Learning algorithms

All of the applications of Machine Learning mentioned above differ from each other. For example, the customer segmentation problem *clusters* the customers into two segments - customers who will pay or the others who won't. Another example like, face recognition problem aims to *classify* a face. So, we can say that, there are broadly two types of machine learning algorithms under which a number of algorithms are present respectively.

- **Supervised Learning**: It consists of Classification and Regression Algorithms. The target variable is known and so it is called supervised learning. For example, when you rate a movie after you view it on Netflix, the suggestion which follows is predicted using a database of your ratings (known as the training data). When the problem is based on continuous variables, (maybe predicting the stock price) then it falls under *regression*. With class labels (such as in the Netflix problem), the learning problem is called a *classification* problem.

- **Unsupervised Learning**: It consists of Clustering Algorithms. The target variable is unknown and so it is called un-supervised learning. In the case of unsupervised learning, there is no labelled data set which can be used for making further predictions. The learning algorithm tries to find patterns or associations in the given data set. Identifying clusters in the data (as in the customer segmentation problem) and reducing dimensions of the data falls in the unsupervised learning category.

Types of Data Analytics



So, that was a quick introduction to Machine Learning and different types of Machine Learning algorithms and a few applications as well. In this course, we will be exploring all that about Machine Learning and Artificial Intelligence. Python and its open source technologies will be used as the primary programming language within this course.

Python is the world largest growing programming language and until before 2007, there existed no built-in machine learning package in Python when David Cournapeau (Developer of Numpy/Scipy) developed **Scikit-learn** as a part of a Google Summer of Code project. The project now has 30 contributors and many sponsors including Google and Python Software Foundation. Scikit-learn provides a range of supervised and unsupervised learning algorithms via a Python interface. In the same way TensorFlow also built by google provides machines the power to handle deep learning stuff.

What is Data Science?

Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of application domains. Data science is related to data mining, machine learning and big data. - Wikipedia

Artificial Intelligence Applications

The integration of AI tools with various software and devices is digitizing the world.
Data Science Nigeria has a module dedicated to various AI uses cases, read here.

References

Mitchell, Tom (1997). Machine Learning. New York: McGraw Hill. ISBN 0-07-042807-7. OCLC 36417892.
Machine Learning Use Cases: https://algorithmia.com/blog/machine-learning-use-cases
"1. Introduction: What Is Data Science? - Doing Data Science [Book]". www.oreilly.com. Retrieved 3 April 2020.
Top Applications of Artificial Intelligence: Top Applications of Artificial Intelligence(AI) in 2021 (intellipaat.com)
Artificial Intelligence Tutorial for Beginners: Artificial Intelligence Tutorial for Beginners [Updated 2020] (simplilearn.com)

INTRODUCTION TO PLATFORMS

Anaconda: https://www.anaconda.com/products/individual
Pandas: https://pandas.pydata.org/
Jupyter Notebook Installation without Anaconda: https://jupyter.org/
Google Colab: https://colab.research.google.com

Jupyter Notebooks

Here is a video link, explaining the use and importance of Jupyter Notebook. To install Jupyter Notebook, go to the link above and download Anaconda, the one which matches your OS requirements (Win/Mac/Linux). We will be using Python 3.6 version. Jupyter Notebooks have a lot of advantages which are the following:

1) **High-Performance Distribution:** Easily install 1,000+ data science packages.
2) **Package Management:** Manage packages, dependencies and environments with Conda.
3) **Portal to Data Science:** Uncover insights in your data and create interactive visualizations. 4) **Interactive:** Coding and Visualization

Getting Started With Jupyter Notebook for Python

In the following tutorial, you will be guided through the process of installing Jupyter Notebook. Furthermore, we'll explore the basic functionality of Jupyter Notebook and you'll be able to try out first examples.
This is at the same time the beginning of a series of Python-related tutorial on CodingTheSmartWay.com. From the very beginning you'll learn everything to need to know to use Python for scientific computing and machine learning use cases.

Jupyter Notebook is a web application that allows you to create and share documents that contain:

- live code (e.g. Python code)
- visualizations
- explanatory text (written in markdown syntax)

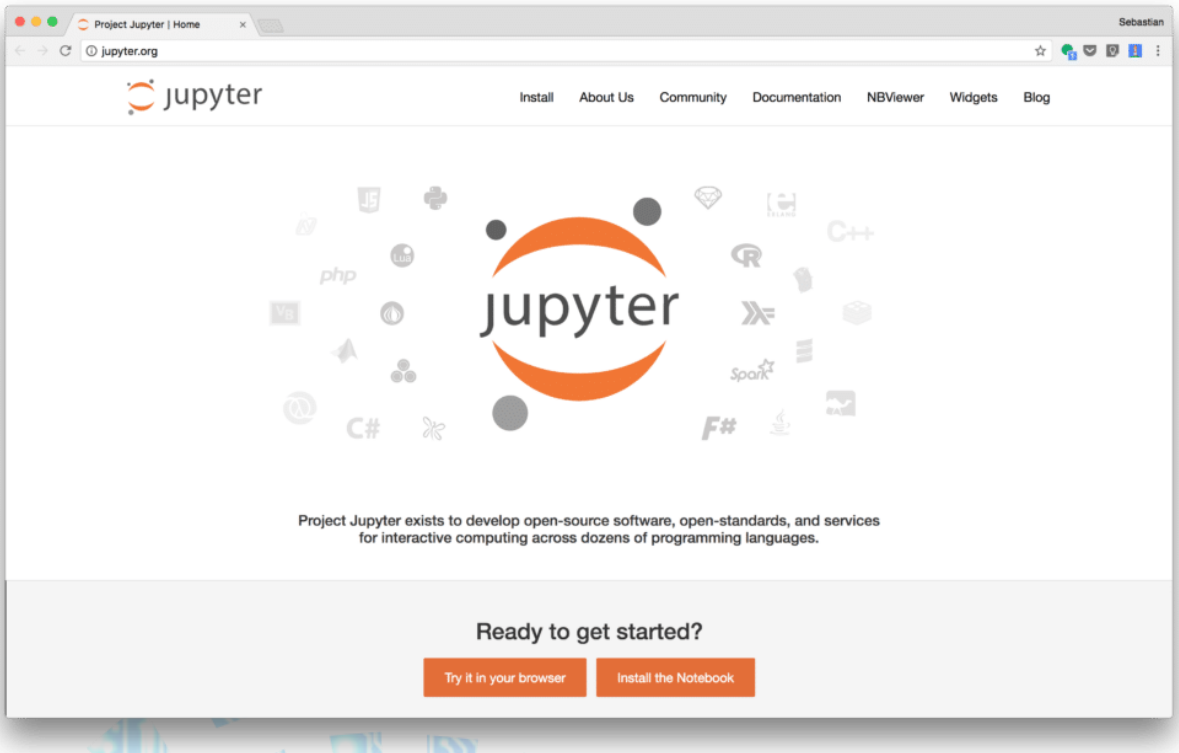
Jupyter Notebook is great for the following use cases:

- learn and try out Python
- data processing / transformation
- numeric simulation
- statistical modeling
- machine learning

Let's get started and install Jupyter Notebook on your computer …

Setting up Jupyter Notebook

The first step to get started is to visit the project's website at http://www.jupyter.org:



Here you'll find two options:

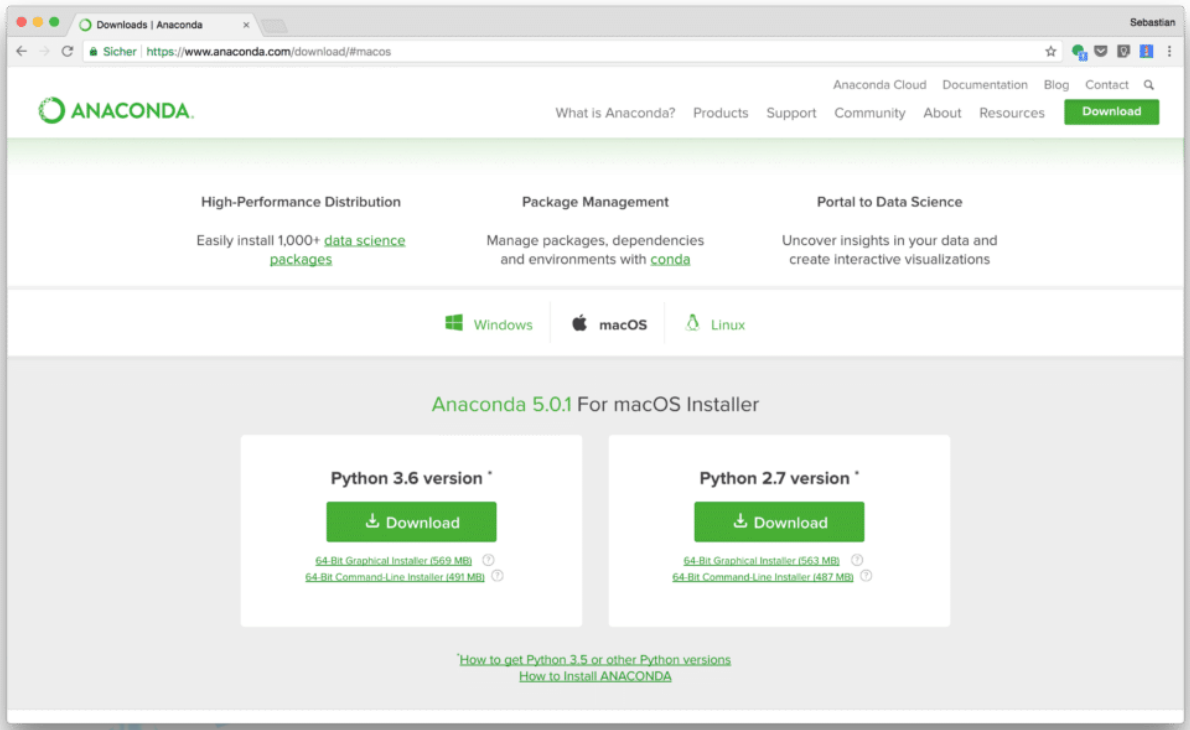
- Try it in your browser Install the Notebook

With the first option *Try it in your browser* you can access a hosted version of Jupyter Notebook. This will get you direct access without needing to install it on your computer.

The second option *Install the Notebook* will take you to another page which gives you detailed instruction for the installation. There are two different ways:

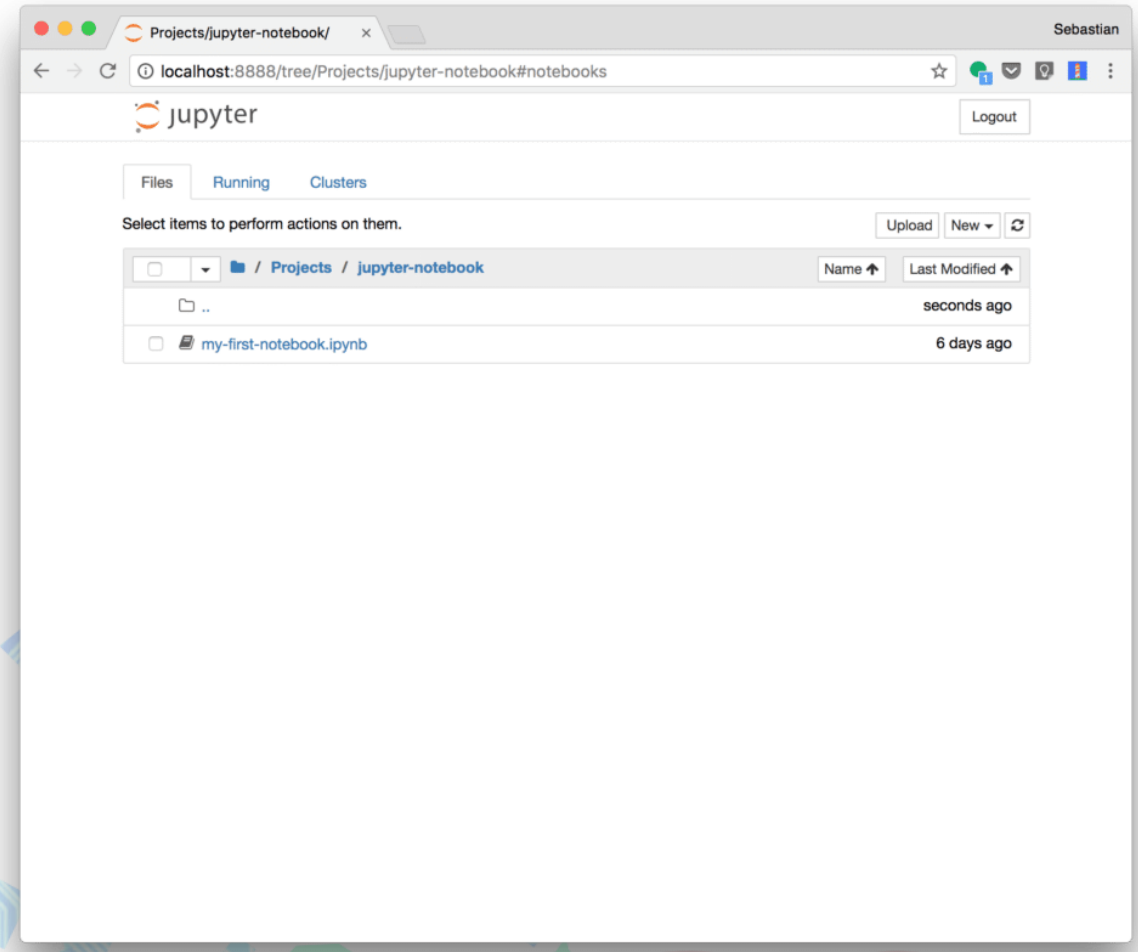
- Installing Jupyter Notebook by using the Python's package manager *pip*
- Installing Jupyter Notebook by installing the Anaconda distribution

Especially if you're new to Python and would like to set up your development environment from scratch using the Anaconda distribution is a great choice. If you follow the link (https://www.anaconda.com/download/) to the Anaconda download page you can choose between installers for Windows, macOS, and Linux:



Download and execute the installer of your choice. Having installed the Anaconda distribution we can now start Jupyter Notebook by clicking the Jupyter n

The web server is started and the Jupyter Notebook application is opened in your default browser automatically. You should be able to see a browser output, which is similar to the following screenshot:



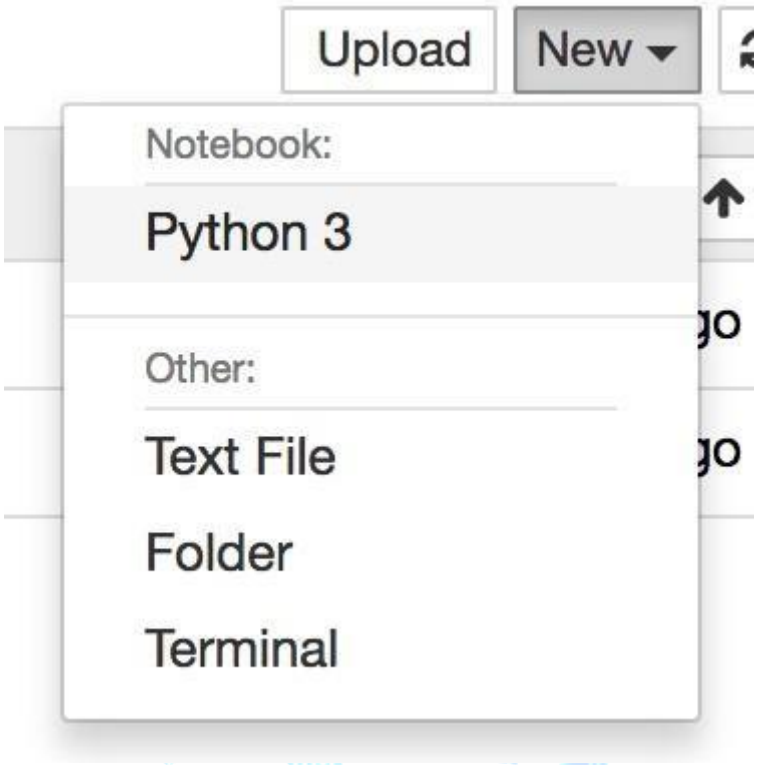
As you can see the user interface of Jupyter Notebook is split up into three sections (tabs):

- Files
- Running
- Clusters

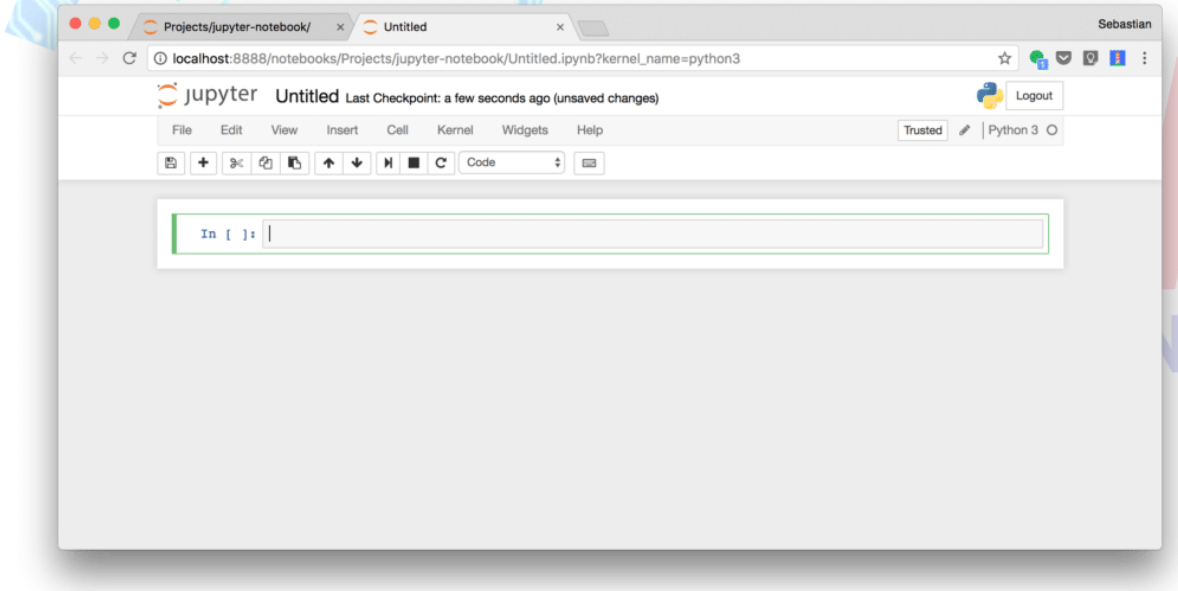
The default view is the *Files* tab from where you can open or create notebooks.

Creating A New Notebook

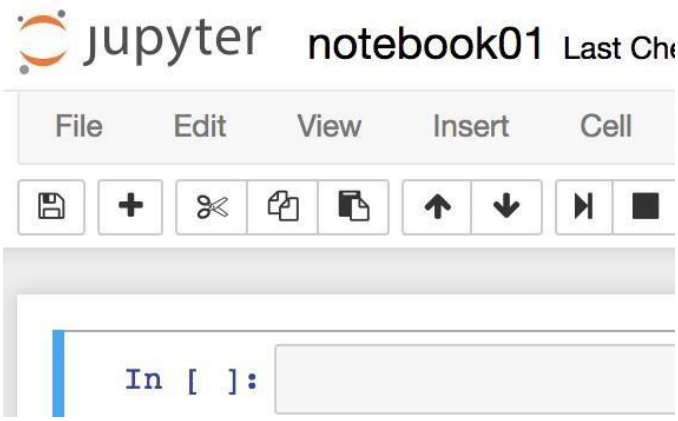
Creating a new Jupyter Notebook is easy. Just use the *New* dropdown menu and you'll see the following options:



Select option *Python 3* to open a new Jupyter Notebook for Python. The notebook is created and you should be able to see something similar to:



The notebook is created but still untitled. By clicking into the text “Untitled” on the top, you can give it a name. By giving it a name the notebook will also be saved as a file of the same name with extension *.ipynb*. E.g. name the notebook *notebook01*:



Switching back to the Files tab you'll be able to see a new file *notebook01.ipynb*:

Because this notebook file is opened right now the file is marked with status *Running*. From here you can decided to shutdown this notebook by clicking on button *Shutdown*.

However before shutting down the notebook let's switch back to the notebook view and try out a few things to get familiar with the notebook concept.

Working with the Notebook

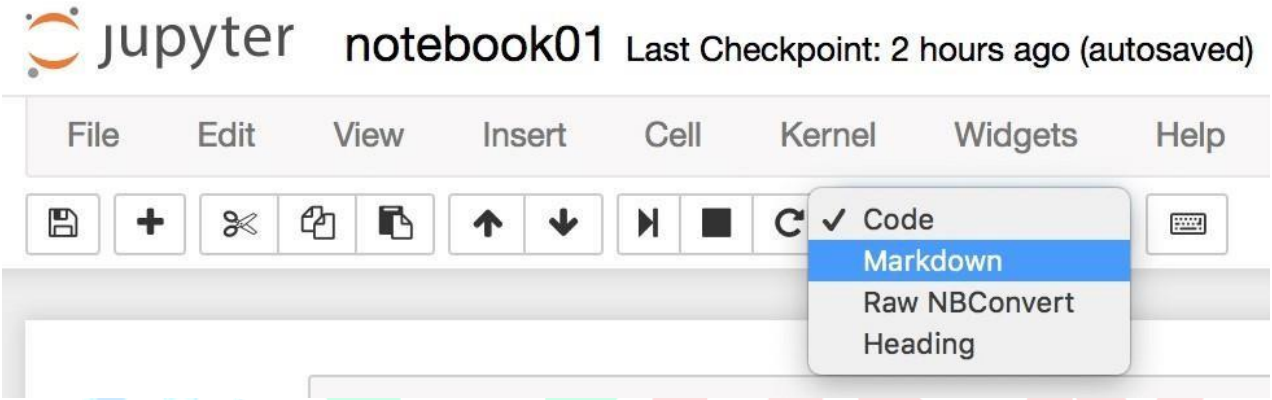
The notebook itself consists of cells. A first empty cell is already available after having created the new notebook:

This cell is of type "Code" and you can start typing in Python code directly. Executing code in this cell can be done by either clicking on the *run cell* button or hitting Shift + Return keys:

The resulting output becomes visible right underneath the cell.

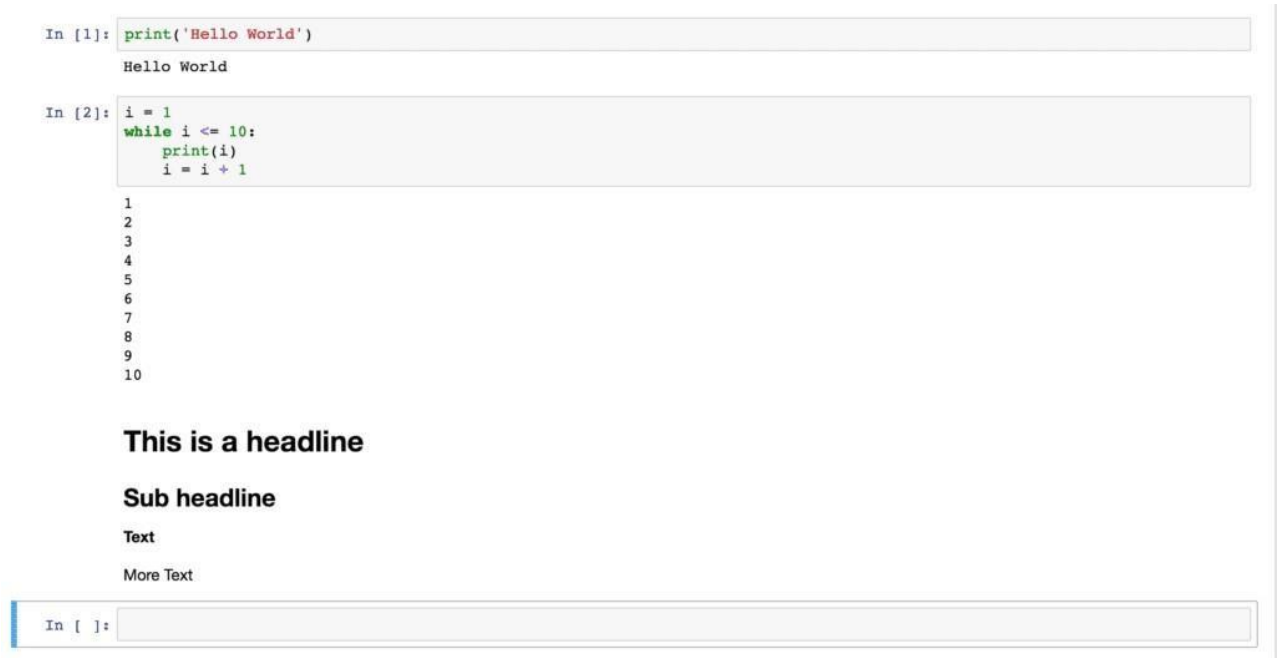
The next empty code cell is created automatically and you can continue to add further code to that cell. Just another example:

You can change the cell type from *Code* to *Markdown* to include explanatory text in your notebook. To change the type you can use the dropdown input control:



Once switched the type to *Markdown* you can start typing in markdown code:

After having entered the markdown code you can compile the cell by hitting Shift + Return once again. The markdown editor cell is then replaced with the output:



If you want to change the markdown code again you can simply click into the compiled result and the editor mode opens again.

Edit And Command Mode

If a cell is active, two modes distinguished:

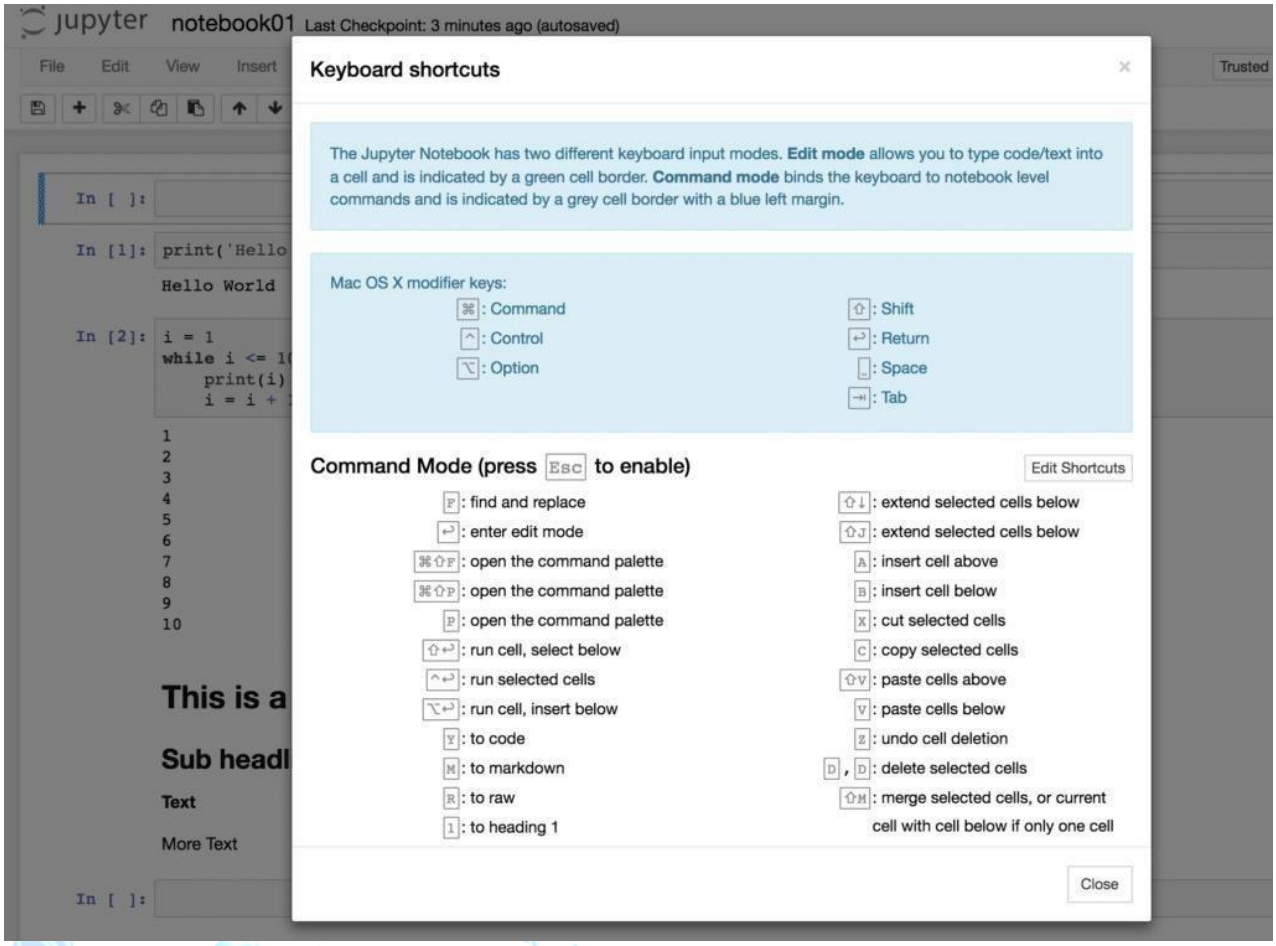
- edit mode
- command mode

If you just click in one cell the cell is opened in command mode which is indicated by a blue border on the left:

The edit mode is entered if you click into the code area of that cell. This mode is indicated by a green border on the left side of the cell:

If you'd like to leave edit mode and return to command mode again you just need to hit ESC.

To get an overview of functions which are available in command and in edit mode you can open up the overview of key shortcuts by using menu entry *Help → Keyboard Shortcuts*:



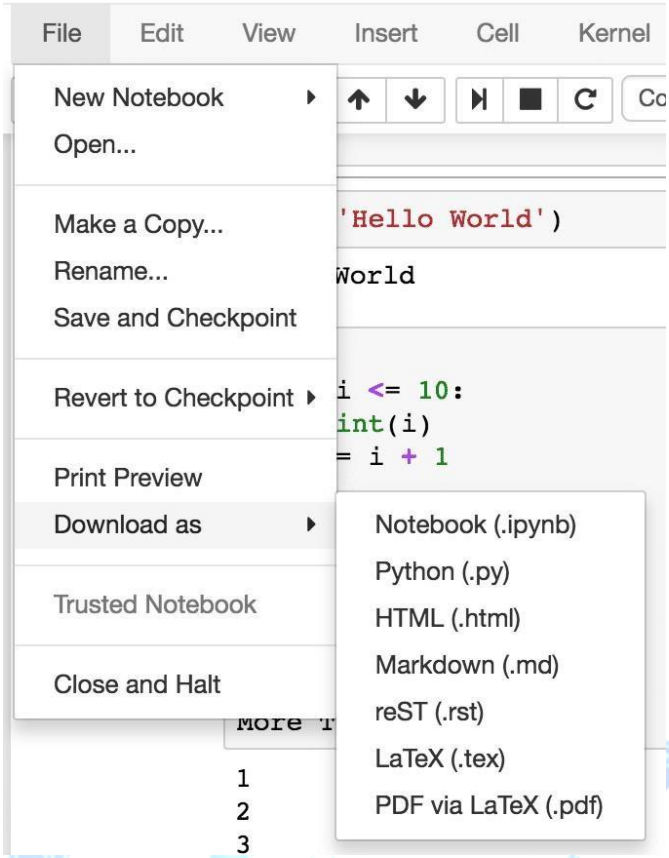
Checkpoints

Another cool function of Jupyter Notebook is the ability to create checkpoint. By creating a checkpoint you're storing the current state of the notebook so that you can later on go back to this checkpoint and revert changes which have been made to the notebook in the meantime.

To create a new checkpoint for your notebook select menu item *Save and Checkpoint* from the *File* menu. The checkpoint is created and the notebook file is saved. If you want to go back to that checkpoint at a later point in time you need to select the corresponding checkpoint entry from menu *File → Revert to Checkpoint*.

Exporting The Notebook

Jupyter Notebook gives you several options to export your notebook. Those options can be found in menu *File* → *Download as*:



Maths: To understand the machine learning algorithms and to **choose the right model**, one needs to understand the maths behind. But,

you don't need all the maths but only some sub-branches:

- Linear algebra
- Probability theory
- Optimization
- Calculus
- Information theory and decision theory

Programming: Programming is needed for the following tasks.

1. Using ML models
2. Build new one
3. Get the data from various sources
4. Clean the data
5. Choose the right features and to validate

Some programming languages are preferred for doing ML than others because they have large number of libraries with most of the ML models already implemented.

- Python
- R (good but slow run time)
- MATLAB (good but costly and slow)
- JULIA (Future best! very fast, good, limited libraries as it is new)

As mentioned, we will be using Python throughout this course. Some good books in ML are the following:

Books:

- Elements of Statistical Learning
- Pattern Recognition and Machine Learning Courses:
- Machine Learning in Coursera • Deep Learning in Coursera Practice:
- Kaggle
- Analytics Vidhya Driven Data

If you read and implement lot of good papers (say 100) you will become an expert in ML/ DL. After this point you can create your own algorithms and start publishing your work.

Popular Python Data Analytics Libraries

Library	Usage
numpy, scipy	Scientific & technical computing
pandas	Data manipulation & aggregation
mlpy, scikit-learn	Machine learning
theano, tensorflow, keras	Deep learning
statsmodels	Statistical analysis
nltk, gensim	Text processing
networkx	Network analysis & visualization
bokeh, matplotlib, seaborn, plotly	Visualization
beautifulsoup, scrapy	Web scraping

Datasets for Machine Learning

Another important consideration while getting into Machine Learning and Deep Learning is the dataset that you are going to use.

Below is a list of free datasets for data science and machine learning, organized by their use cases.

Datasets for Exploratory Analysis

- Game of Thrones
- World University Rankings
- IMDB 5000 Movie Dataset
- Kaggle Datasets

Datasets for General Machine Learning

- Wine Quality (Regression)
- Credit Card Default (Classification)
- US Census Data (Clustering)
- UCI Machine Learning Repository Datasets for Deep Learning
- MNIST CIFAR
- ImageNet
- YouTube 8M
- Deeplearning.net
- DeepLearning4J.org
- Analytics Vidhya

Datasets for Natural Language Processing

- Enron Dataset
- Amazon Reviews
- Newsgroup Classification
- NLP-datasets (Git-Hub)
- Quora Answer

Datasets for Cloud Machine Learning

- AWS Public Datasets
  - Google Cloud Public Datasets
  - Microsoft Azure Public Datasets Datasets for Time Series Analysis
  - EOD Stock Prices
  - Zillow Real Estate Research
  - Global Education Statistics

Datasets for Recommender Systems

- MovieLens
- Jester
- Million Song Dataset

Some more Machine Learning Resources

From the next lesson, we will be studying algorithms specifically such as for regression, classification, clustering etc. In this lesson, we provided you resources to all the common machine-learning algorithms as a whole to explore and also links to a few highly rated and renowned courses from top professionals in Machine Learning as well as Deep Learning. All of the resources are available free online.

Machine Learning Theory

- Machine Learning, Stanford University
- Machine Learning, Carnegie Mellon University
- Machine Learning, MIT
- Machine Learning, California Institute of Technology
- Machine Learning, Oxford University
  - Machine Learning, Data School

A Mathematical Introduction to Machine Learning

Machine Learning theory is a field that intersects statistical, probabilistic, computer science and algorithmic aspects arising from learning iteratively from data and finding hidden insights which can be used to build intelligent applications (Akinfaderin, 2017). Machine learning could also be seen as the subfield of computer science concerned with creating machines that can improve from experience and interaction. These machines rely upon mathematical optimization, statistics, and algorithm design (Arora, 2018).

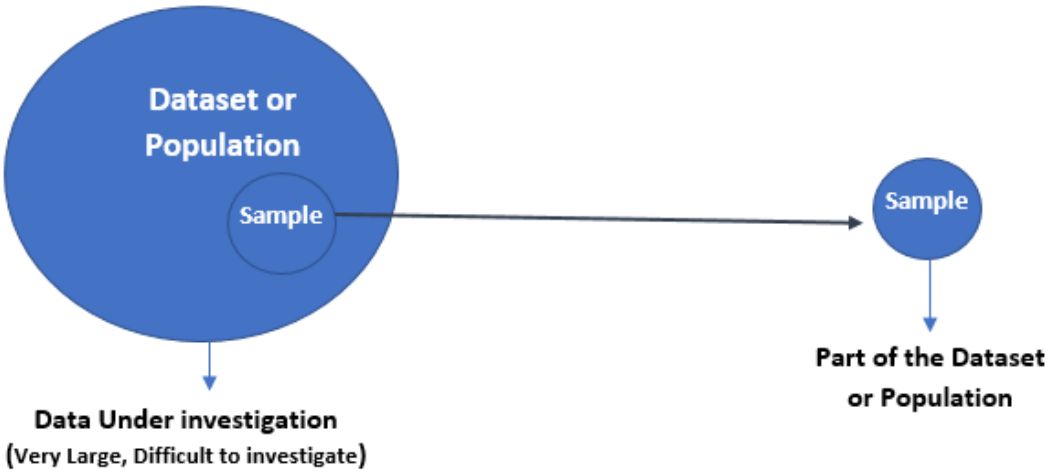
There are some basic important concepts or minimum level of mathematics needed to be a Machine Learning Scientist/Engineer . These includes ; Linear Algebra, Probability Theory and Statistics, Calculus, Algorithms and Complex Optimizations

Sampling techniques

Analyzing and building machine learning model for relatively big dataset could be a cumbersome task on computationally limited machines. This made it useful to pick a subset of the data and analyze that in such a way that

the subset could be a good representation of the entire dataset. The method used to get such subset of data is called Sampling

Sampling is a method that helps researchers to infer information about a dataset/population based on results from a subset of the dataset/population, without having to investigate every individual.



The above diagram perfectly illustrates what sampling is. Let us understand this at a more intuitive level through an example.

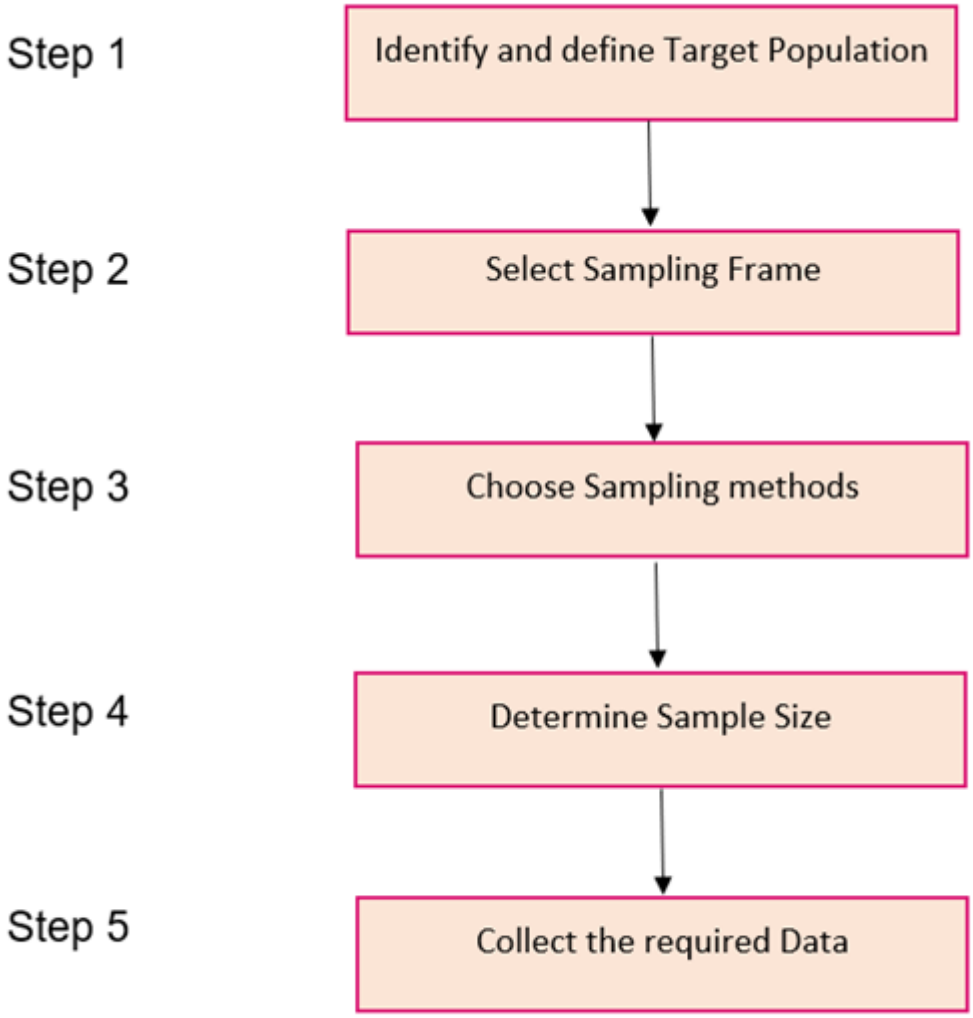
We want to find the average height of all adult males in Yobe State. The population of Yobe State is around 3 million and males would be roughly around 1.5 million (these are general assumptions for this example so don't take them at face value!). As you can imagine, it is nearly impossible to find the average height of all males in Yobe State.

It's also not possible to reach every male so we can't really analyze the entire population. So, what *can* we do instead? We can take multiple samples and calculate the average height of individuals in the selected samples.

Here's a potential solution – find random people in random situations where our sample would not be skewed based on heights.

Steps involved in Sampling

In order to best make sampling out of a population. The following step-by-step process show in a flowchart can be Adopted.



(Gangwal, 2019)

Let's take an interesting case study and apply these steps to perform sampling. We recently conducted General Elections in India a few months back. You must have seen the public opinion polls every news channel was running at the time:

Were these results concluded by considering the views of all 900 million voters of the country or a fraction of these voters? Let us see how it was done.

Step 1

The first stage in the sampling process is to clearly define the target population. So, to carry out opinion polls, polling agencies consider only the people who are above 18 years of age and are eligible to vote in the population.

Step 2

Sampling Frame – It is a list of items or people forming a population from which the sample is taken.

So, the sampling frame would be the list of all the people whose names appear on the voter list of a constituency.

Step 3

Generally, probability sampling methods are used because every vote has equal value and any person can be included in the sample irrespective of his caste, community, or religion. Different samples are taken from different regions all over the country.

Step 4

Sample Size – It is the number of individuals or items to be taken in a sample that would be enough to make inferences about the population with the desired level of accuracy and precision.

Larger the sample size, more accurate our inference about the population would be.

For the polls, agencies try to get as many people as possible of diverse backgrounds to be included in the sample as it would help in predicting the number of seats a political party can win.

Step 5

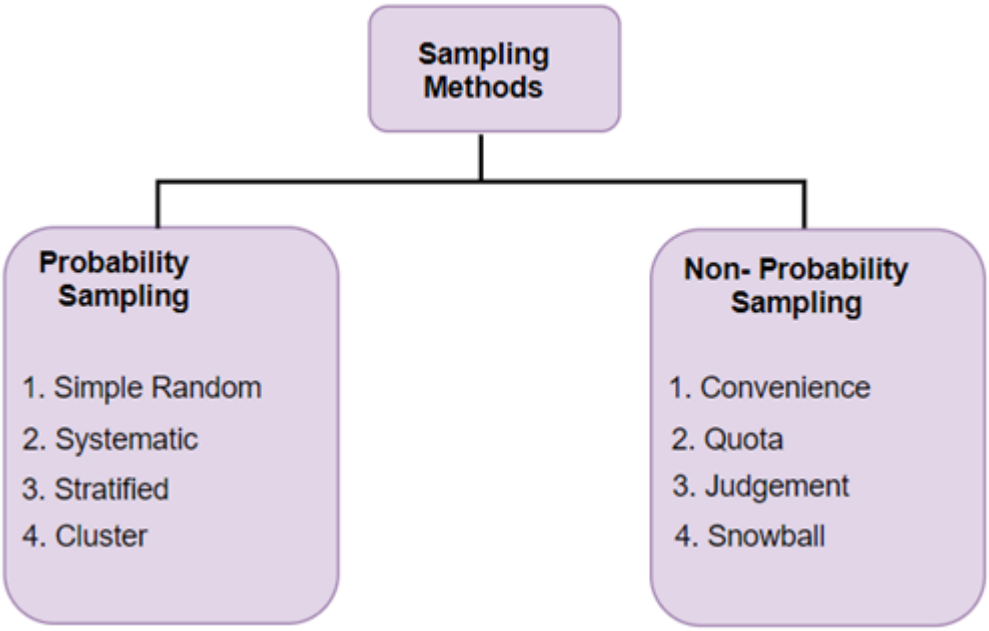
Once the target population, sampling frame, sampling technique, and sample size have been established, the next step is to **collect data from the sample**.

In opinion polls, agencies generally put questions to the people, like which political party are they going to vote for or has the previous party done any work, etc.

Based on the answers, agencies try to interpret who the people of a constituency are going to vote for and approximately how many seats is a political party going to win.

Different Types of Sampling Techniques

Here comes another diagrammatic illustration! This one talks about the different types of sampling techniques available to us:



- **Probability Sampling:** In probability sampling, every element of the population has an equal chance of being selected. Probability sampling gives us the best chance to create a sample that is truly representative of the population
- **Non-Probability Sampling:** In non-probability sampling, all elements do not have an equal chance of being selected. Consequently, there is a significant risk of ending up with a non-representative sample which does not produce generalizable results

For example, let's say our population consists of 20 individuals. Each individual is numbered from 1 to 20 and is represented by a specific color (red, blue, green, or yellow). Each person would have odds of 1 out of 20 of being chosen in probability sampling.

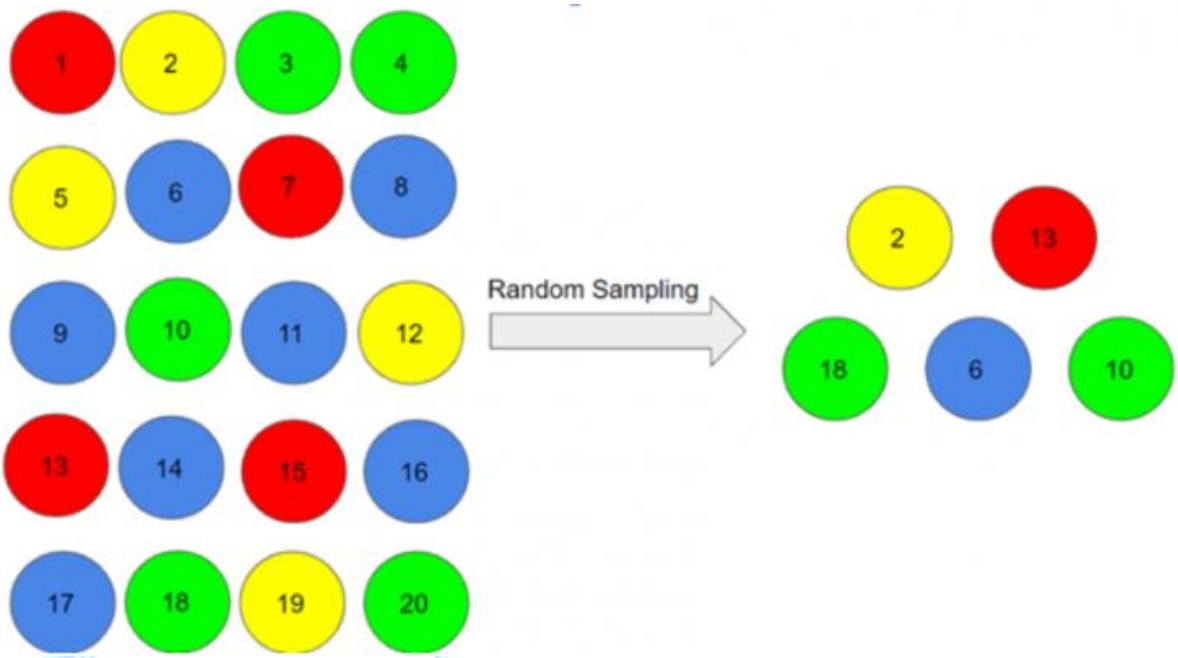
With non-probability sampling, these odds are not equal. A person might have a better chance of being chosen than others. So now that we have an idea of these two sampling types, let's dive into each and understand the different types of sampling under each section.

Types of Probability Sampling

Simple Random Sampling

This is a type of sampling technique you must have come across at some point. Here, every individual is chosen entirely by chance and each member of the population has an equal chance of being selected.

Simple random sampling reduces selection bias.

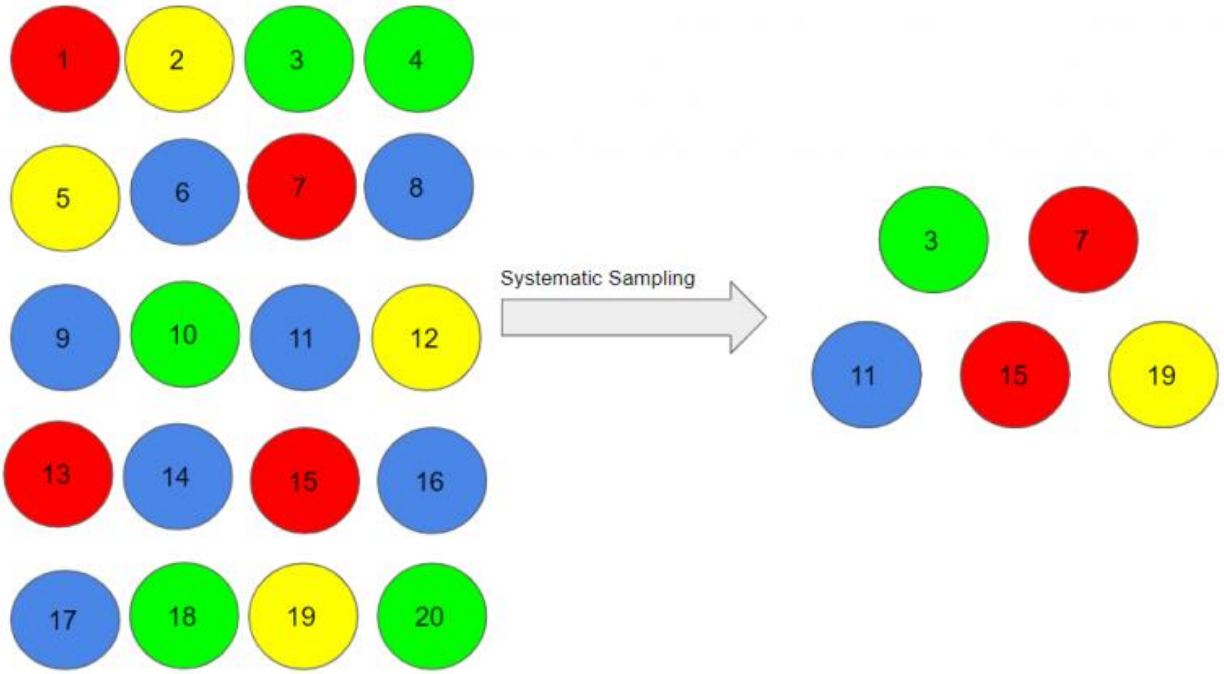


One big advantage of this technique is that it is the most direct method of probability sampling. But it comes with a caveat – it may not select enough individuals with our characteristics of interest. *Monte Carlo methods use repeated random sampling for the estimation of unknown parameters.*

Systematic Sampling

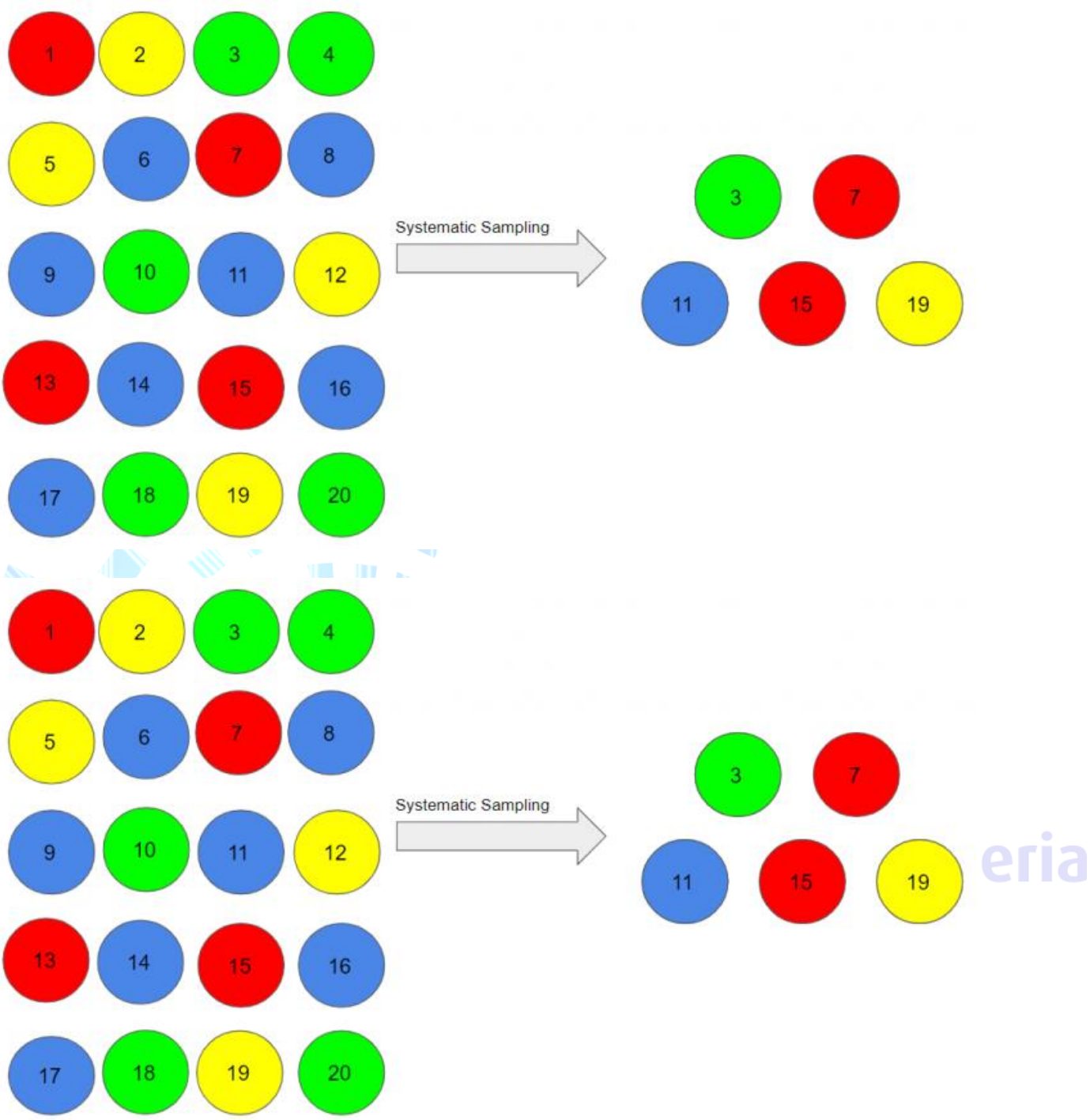
In this type of sampling, the first individual is selected randomly and others are selected using a fixed 'sampling interval'. Let's take a simple example to understand this.

Say our population size is x and we have to select a sample size of n. Then, the next individual that we will select would be x/nth intervals away from the first individual. We can select the rest in the same way.



Suppose, we began with person number 3, and we want a sample size of 5. So, the next individual that we will select would be at an interval of (20/5) = 4 from the 3rd person, i.e. 7 (3+4), and so on.

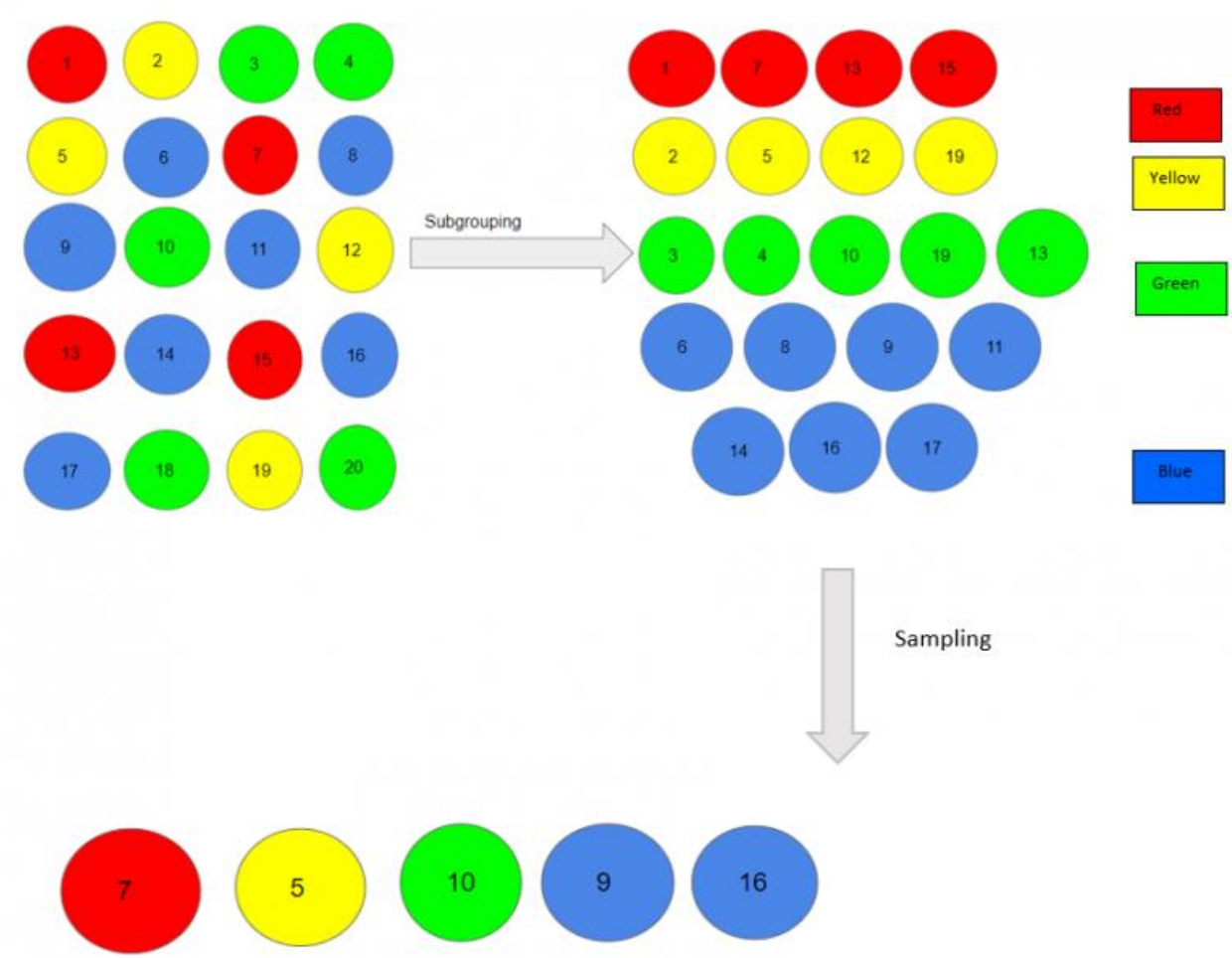
3, 3+4=7, 7+4=11, 11+4=15, 15+4=19 = **3, 7, 11, 15, 19**



Systematic sampling is more convenient than simple random sampling. However, it might also lead to bias if there is an underlying pattern in which we are selecting items from the population (though the chances of that happening are quite rare).

Stratified Sampling

In this type of sampling, we divide the population into subgroups (called strata) based on different traits like gender, category, etc. And then we select the sample(s) from these subgroups:

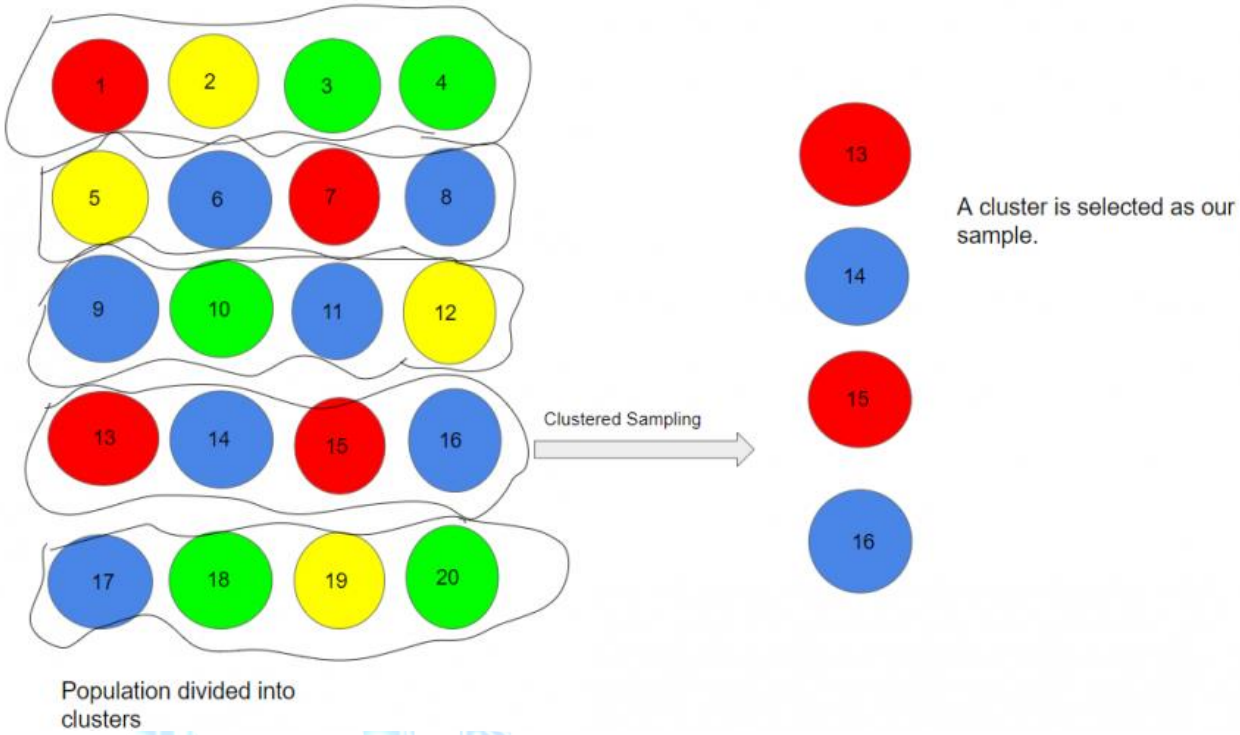


Here, we first divided our population into subgroups based on different colors of red, yellow, green and blue. Then, from each color, we selected an individual in the proportion of their numbers in the population.

We use this type of sampling when we want representation from all the subgroups of the population. However, stratified sampling requires proper knowledge of the characteristics of the population.

Cluster Sampling

In a clustered sample, we use the subgroups of the population as the sampling unit rather than individuals. The population is divided into subgroups, known as clusters, and a whole cluster is randomly selected to be included in the study:



In the above example, we have divided our population into 5 clusters. Each cluster consists of 4 individuals and we have taken the 4th cluster in our sample. We can include more clusters as per our sample size.

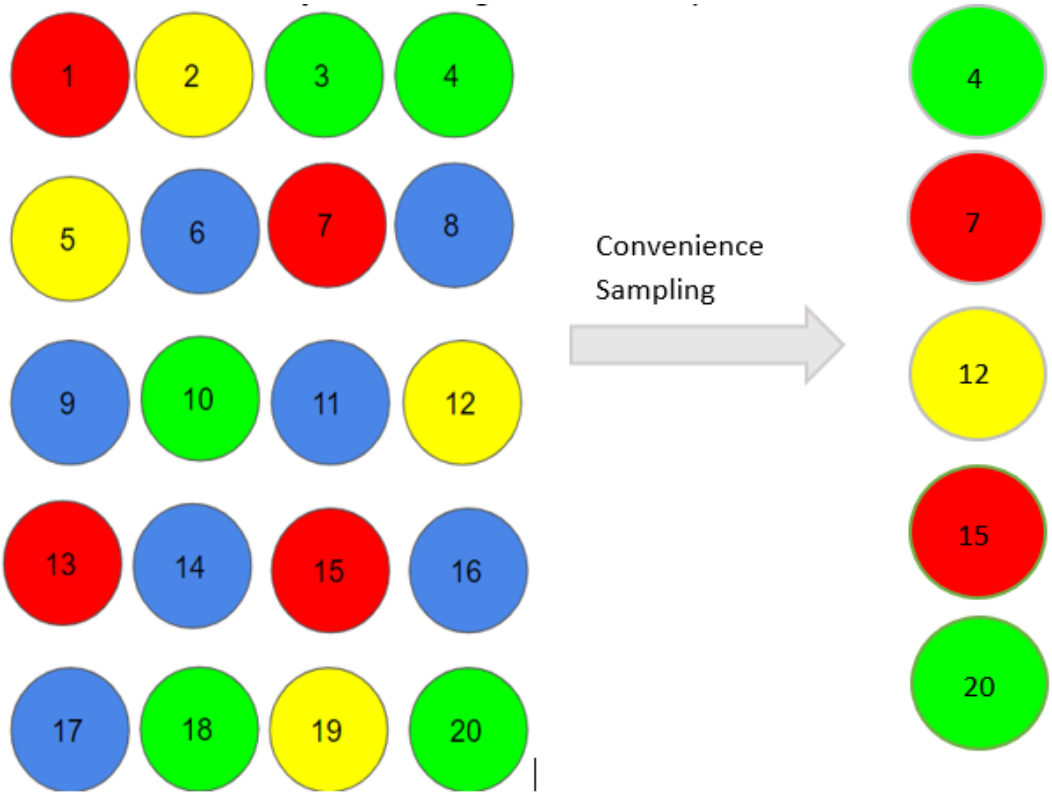
This type of sampling is used when we focus on a specific region or area.

Types of Non-Probability Sampling

Convenience Sampling

This is perhaps the easiest method of sampling because individuals are selected based on their availability and willingness to take part.

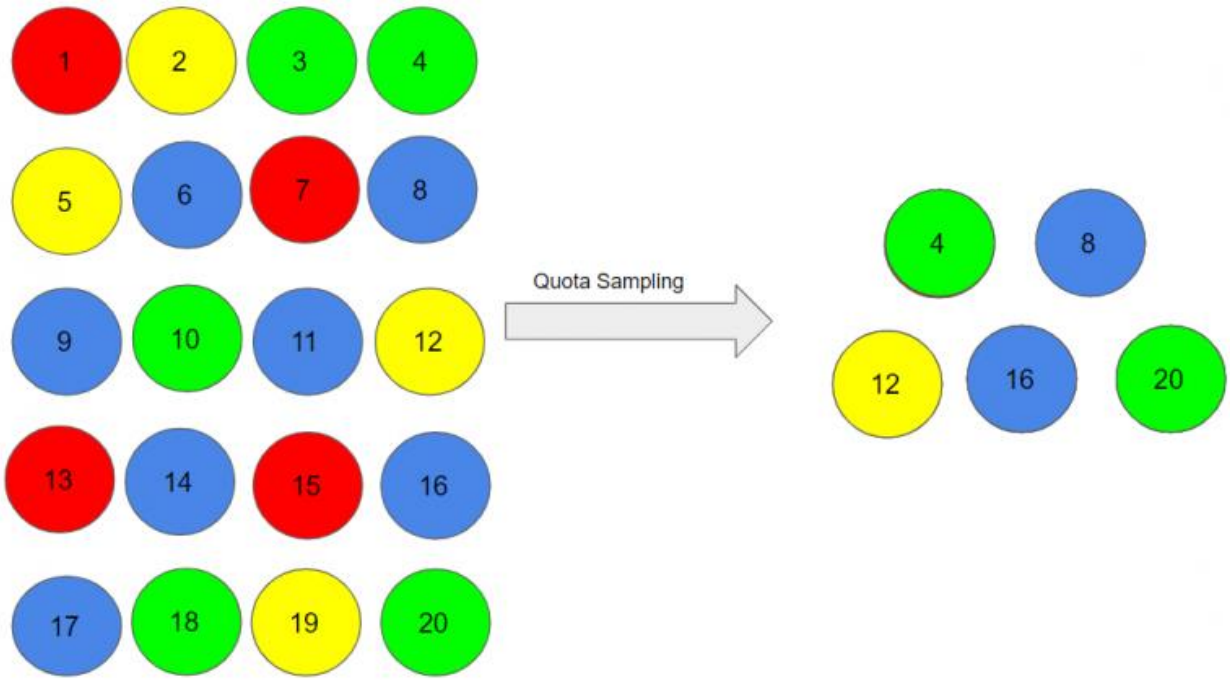
Here, let's say individuals numbered 4, 7, 12, 15 and 20 want to be part of our sample, and hence, we will include them in the sample.



Convenience sampling is prone to significant bias, because the sample may not be the representation of the specific characteristics such as religion or, say the gender, of the population.

Quota Sampling

In this type of sampling, we choose items based on predetermined characteristics of the population. Consider that we have to select individuals having a number in multiples of four for our sample:

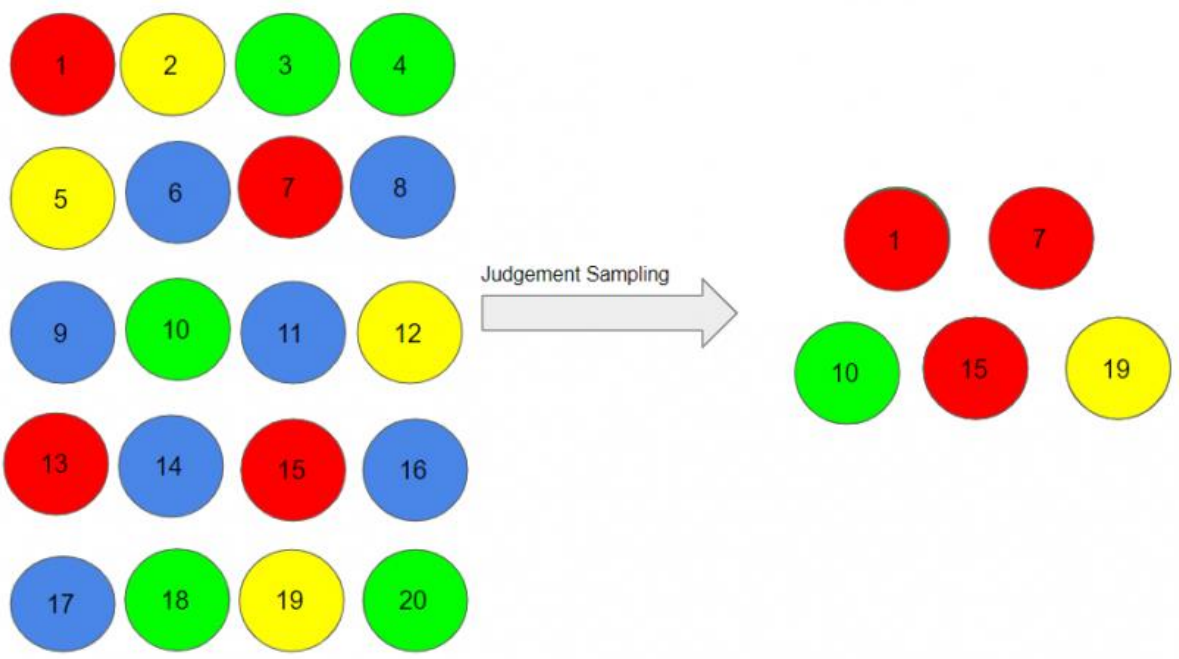


Therefore, the individuals numbered 4, 8, 12, 16, and 20 are already reserved for our sample.

In quota sampling, the chosen sample might not be the best representation of the characteristics of the population that weren't considered.

Judgment Sampling

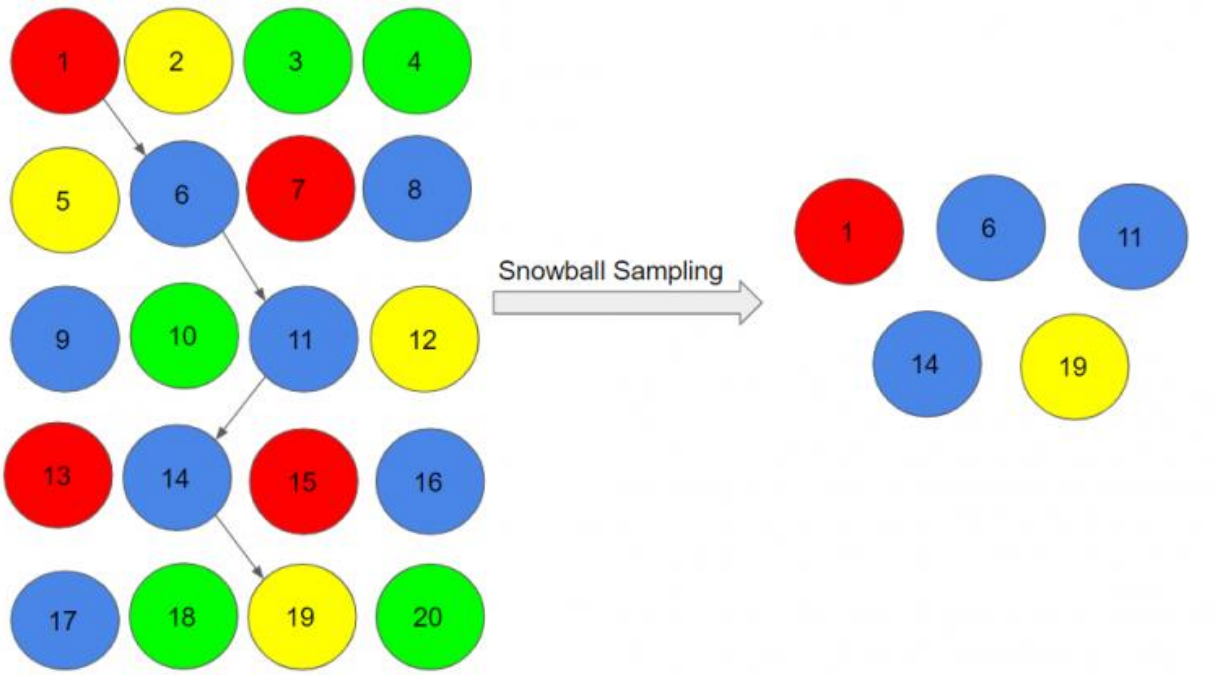
It is also known as selective sampling. It depends on the judgment of the experts when choosing whom to ask to participate.



Suppose, our experts believe that people numbered 1, 7, 10, 15, and 19 should be considered for our sample as they may help us to infer the population in a better way. As you can imagine, quota sampling is also prone to bias by the experts and may not necessarily be representative.

Snowball Sampling

I quite like this sampling technique. **Existing people are asked to nominate further people known to them so that the sample increases in size like a rolling snowball.** This method of sampling is effective when a sampling frame is difficult to identify.



Here, we had randomly chosen person 1 for our sample, and then he/she recommended person 6, and person 6 recommended person 11, and so on.

1->6->11->14->19

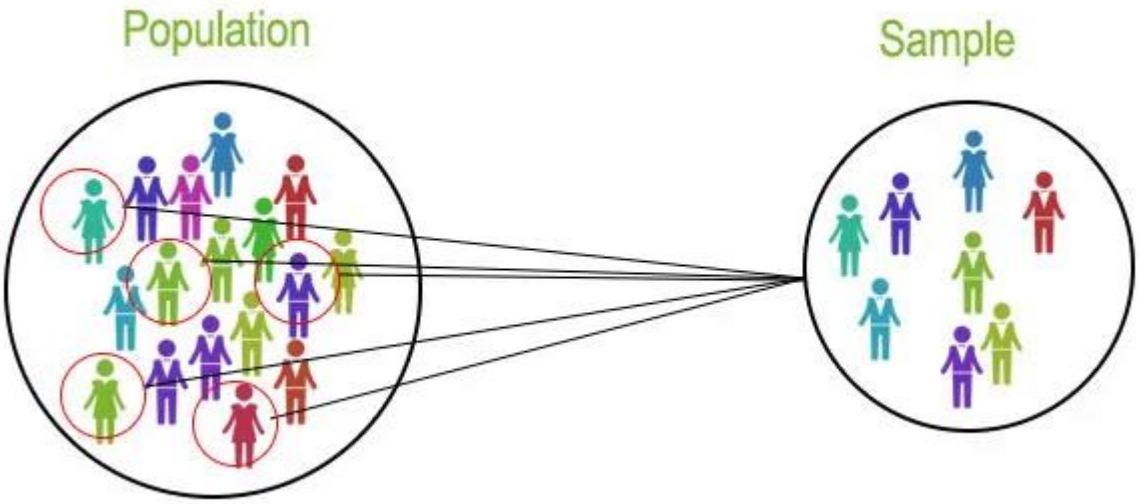
There is a significant risk of selection bias in snowball sampling, as the referenced individuals will share common traits with the person who recommends them.

Types of Statistics for Machine Learning

Below are the points that explains the types of statistics:

1. Population

It refers to the collection that includes all the data from a defined group being studied. The size of the population may be either finite or infinite.



2. Sample

The study of the entire population is always not feasible, instead, a portion of data is selected from a given population to apply the statistical methods. This portion is called a Sample. The size of the sample is always finite

3. Mean

More often termed as “average”, the meaning is the number obtained by computing the sum of all observed values divided by the total number of values present in the data

4. Median

Median is the middle value when the given data are ordered from smallest to largest. In case of even observations, the median is an average value of 2 middle numbers

5. Mode

The mode is the most frequent number present in the given data. There can be more than one mode or none depending on the occurrence of numbers.

6. Variance

Variance is the averaged squared difference from the Mean. The difference is squared to not cancel out the positive and negative values.

7. Standard Deviation

Standard Deviation measures how spread out the numerical values are. It is the square root of variance. A higher number of Standard Deviation indicates that data is more spread.

8. Range

Difference between the highest and lowest observations within the given data points. With extreme high and low values, the range can be misleading, in such cases interquartile range or std is used

9. Inter Quartile Range (IQR)

Quartiles are the numbers that divide the given data points into quarters and are defined as below

- **Q1:** middle value in the first half of the ordered data points
- **Q2:** median of the data points
- **Q3:** middle value in the second half of the ordered data points
- **IQR:** given by Q3-Q1

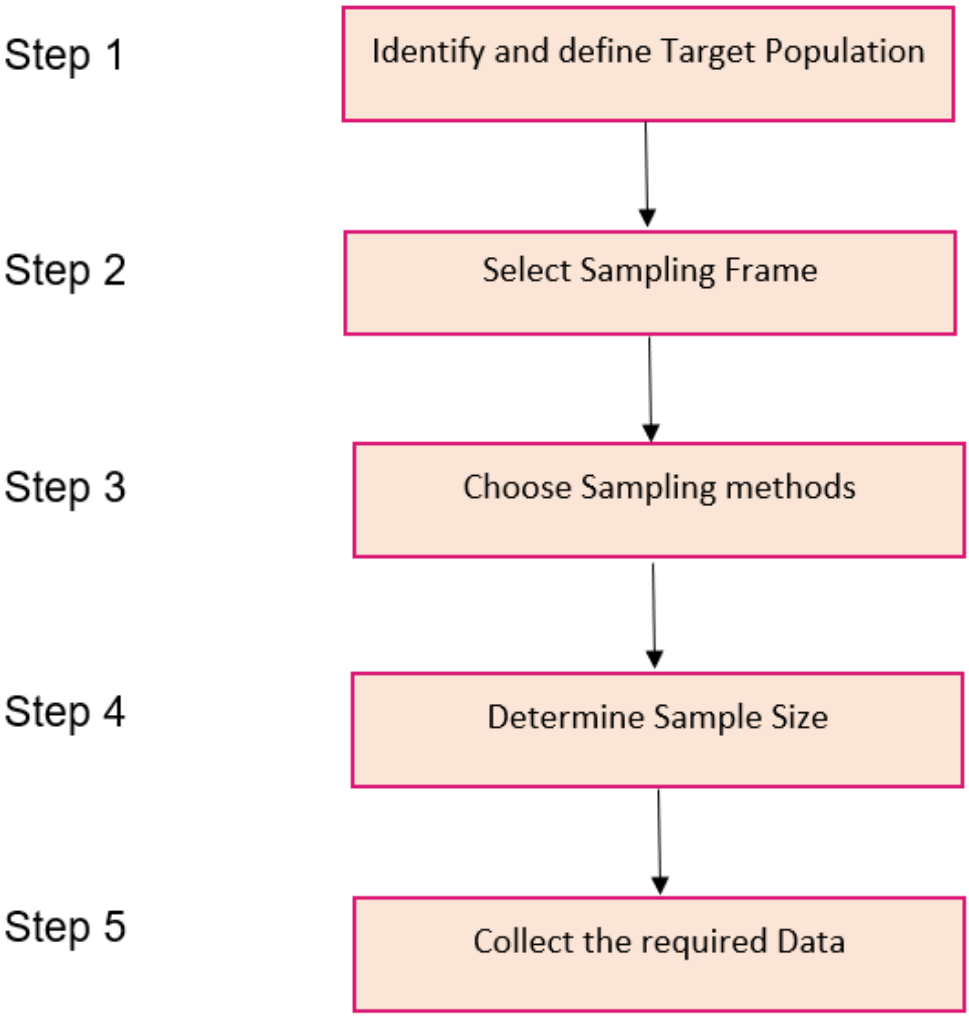
IQR gives us an idea where most of the data points lie contrary to the range that only provides the difference between the extreme points. Due to this IQR can also be used to detect outliers in the given data set

Probability

Probability is the branch of mathematics concerning numerical descriptions of how likely an event is to occur, or how likely it is that a proposition is true. The probability of an event is a number between 0 and 1, where, roughly speaking, 0 indicates impossibility of the event and 1 indicates certainty.

In probability theory, an event is a set of outcomes of an experiment to which a probability is assigned. If ***E*** represents an event, then P(***E***) represents the probability that ***E*** will occur. A situation where ***E*** might happen (success) or might not happen (failure) is called a ***trial***.

This event can be anything like tossing a coin, rolling a die or pulling a colored ball out of a bag. In these examples the outcome of the event is random, so the variable that represents the outcome of these events is called a ***Random Variable*** (Nabi, 2019) . Random variables may be discrete or continuous. A discrete random variable is one that has a finite or countably infinite number of states. Note that these states are not necessarily the integers; they can also just be named states that are not considered to have any numerical value. A continuous random variable is associated with a real value.



References

Akinfaderin, W. (2017, March 24). *The Mathematics of Machine Learning.* Retrieved from towardsdatascience: https://towardsdatascience.com/the-mathematics-of-machine-learning-894f046c568

Arora, S. (2018). *MATHEMATICS OF MACHINE LEARNING: AN INTRODUCTION.* Retrieved from RIO De Janeiro: https://eta.impa.br/dl/PL019.pdf

Nabi, J. (2019, January 7). *Machine Learning — Probability & Statistics*. Retrieved from towardsdatascience: https://towardsdatascience.com/machine-learning-probability-statistics-f830f8c09326

DSN
Data Science Nigeria

Day 2

Introduction to Pandas and Numpy

- Installation and Environment Setup
- Libraries
  - Numpy, Pandas, Matplotlib basics
  - Data Preprocessing
    - Handling Missing Data
    - Encoding Categorical Variable
- Introduction to Regression
- Homework

Installation and Environment Setup

For this training, we will be using one of the most popular framework used for data science and machine learning known as **Anaconda**. You can download the correct version of anaconda here (https://www.anaconda.com/distribution/)as it relates to your op erating system – Windows, Mac or Linux.

- Why Python?

Python is a general-purpose, beginner friendly language, which can be used to build virtually anything. It can be used for web development, desktop app development, gaming, data analysis, Artificial intellingence, scientific computing etc. Increasing community of python users is another reason for choosing PYTHON

Remember,

> The larger a community, the more likely you'd get help and the more people will be building useful tools to ease the process of development.

Required Libraries

- Numpy
- Pandas
- Matplotlib
- Scikit Learn

NUMPY

This is a fundamental Package for scientific computing for manipulation of multi-dimensional arrays and matrices. It is particularly useful for linear algebra, Fourier transform, random number simulation etc

Matrices are rectangular array of numbers, symbols and expressions arranged in rows and columns. The numbers, symbols or expressions in the matrix are called its entries or its elements. The horizontal and vertical lines of entries in a matrix are called rows and columns, respectively. Its operations inclue addition, subtraction, multiplication

The first step is to import numpy library into the active notebook

In [1]:

```
import numpy
```

To shorten the length of any library, a better alternative is to instantiate the library with a shorter name, as in, In [2]:

```
import numpy as np
```

With this, each time numpy is required on this active notebook, **np** will be used instead

In [3]:

```
#creating a 1 dimensional array

x = np.array([1, 2, 3, 4, 5])
y = np.array([9, 10])
print(x)
print('The shape of X is' , x.shape)

print(y)
print('The shape of Y is' , y.shape)
```

```
[1 2 3 4 5]
The shape of X is (5,)
[ 9 10]
The shape of Y is (2,)
```

In [4]:

```
# Creating a 2D arrays
z = np.array([[1, 2], [3, 4]])

print(z)
print('The shape of Z is' , z.shape)
```

```
[[1 2]
 [3 4]]
The shape of Z is (2, 2)
```

In [5]:

```
# creating a multidimensional array

w = np.array([[[1,2,3],[4,5,6],[7,8,9]],[[10,11,12],[13,14,15],[16,17,18]],[[19,20,21],[
print(w, '\n')
print('The shape of W is' , w.shape)
```

```
[[[ 1  2  3]
  [ 4  5  6]
  [ 7  8  9]]

 [[10 11 12]
  [13 14 15]
  [16 17 18]]

 [[19 20 21]
  [22 23 24]
  [26 27 28]]]

The shape of W is (3, 3, 3)
```

Numpy Functions

Numpy has built-in functions for creating arrays. These includes:

arrange:

reshape

zeros ones

full

linspace random

> The dimensions (no of rows and column) are passed as parameters to the function.

In [6]:

```
#arrange is Used to create arrays with values in a specified range.

A10 = np.arange(10)
A10
print(A10.shape)
```

(10,)

In [7]:

```
#To change a scalar matrix to vextor

B10 = A10.reshape(-1,1)

print ( A10, '\n', B10)
print ("The shape of 1D array X = " , B10.shape)
```

```
[0 1 2 3 4 5 6 7 8 9]
 [[0]
 [1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
The shape of 1D array X =  (10, 1)
```

In [8]:

```
A10 = B10.reshape(2,5)

print ( A10)
print ("The shape of 1D array X = " , A10.shape)
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
The shape of 1D array X =  (2, 5)
```

> Note: The new dimension must be compatible with the old one

In [9]:

```
#zeros is used to create an array filled with zeros.

np_Zeros = np.zeros((2,3))

np_Zeros
```

Out[9]:

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [10]:

```
#ones is used to create an array filled with ones
np_Ones = np.ones((2,3))

np_Ones
```

Out[10]:

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

In [11]:

```
 #function creates a n * n array filled with a specified given value.
np_full = np.full((2,3), 4)

np_full
```

Out[11]:

```
array([[4, 4, 4],
       [4, 4, 4]])
```

In [12]:

```
#The eye function lets you create a n * n diagonal matrix
np_eye = np.eye(3,6)

np_eye
```

Out[12]:

```
array([[1., 0., 0., 0., 0., 0.],       [0., 1., 0., 0., 0., 0.],
    [0., 0., 1., 0., 0., 0.]])
```

In [13]:

```
#linspace returns evenly spaced numbers over a specified interval
np_linspace = np.linspace(0, 10, num = 6)

np_linspace
```

Out[13]:

```
array([ 0.,  2.,  4.,  6.,  8., 10.])
```

In [14]:

```
#This creates an array filled with random values between 0 and 1
np_rand = np.random.random_sample((2,3))

np_rand1 = np.random.rand(2,3)

X = np.random.randint(10, size=(5,3))

print(np_rand)

print(np_rand1)
X
```

```
[[0.28608689 0.58308977 0.96682239]
 [0.29172351 0.82030534 0.41377409]] [[0.12963402
0.59496657 0.17656489]  [0.76165033 0.14181669
0.17240588]]
```

Out[14]:

```
array([[3, 9, 7],       [3, 2, 3],
    [7, 4, 6],
    [1, 1, 8],
    [4, 9, 9]])
```

In

Accessing elements of Numpy array

> To access an element in a two-dimensional array, you need to specify an index for both the row and the column.

[15]:

```
#Row 1, column 0 gives a scalar
z[1,0]
```

Out[15]:

3

In [16]:

```
#or

p = z[1][0]

p
```

Out[16]:

3

In [17]:

```
p = (z[0:1, 0])
p
```

Out[17]: array([1])

Numpy Attributes

Array attributes reflect information that is intrinsic to the array itself. Generally, accessing an array through its attributes allows you to get and sometimes set intrinsic properties of the array without creating a new array. The exposed attributes are the core parts of an array and only some of them can be reset meaningfully without creating a new array

Some commonly used attributes are:

- Shape: indicates the size of an array
- Size: returns the total number of elements in the NumPy array
- Dtype: returns the type of elements in the array, i.e., int64, character

In [18]:

```
print ("The Dtype of elements in array X= "  , x.dtype)

print ("The shape of ND array W= " , w.dtype)
```

In

The Dtype of elements in array X= int32
The shape of ND array W= int32

[19]:

```
print ("The shape of 1D array X = ", x.shape) print ("The shape of 2D
array Z = ", z.shape) print ("The shape of ND array W = ", w.shape)
print ("The shape of arange A10 = ", A10.shape)
```

The shape of 1D array X = (5,)
The shape of 2D array Z = (2, 2)
The shape of ND array W = (3, 3, 3)
The shape of arange A10 = (2, 5)

In [20]:

```
print ("The shape of ND array W = ", w.size) print ("The shape of
arange A10 = ", A10.size)
```

The shape of ND array W = 27
The shape of arange A10 = 10

Numpy array math operations

In [21]:

```
x = np.array([[1,2,3],[4,5,6]])
y = np.array([[2,2,2],[3,3,3]])
z = np.array([1,2,3])
```

In [22]:

```
#Transpose a matrix

x.T
```

Out[22]:

```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [23]:

```
#Elementwise addittion

print (x+y)
print (np.add(x,y))
```

```
[[3 4 5]
 [7 8 9]] [[3 4 5]
 [7 8 9]]
```

In [24]:

```
#Elementwise Subtraction

print (x-y)
print (np.subtract(x,y))
```

```
[[-1  0  1]
 [1 2 3]] [[-1 0 1]
 [1 2 3]]
```

In [25]:

```
#Elementwise Multiplication

print (x*z)
print (np.multiply(x,z))
```

```
[[ 1  4  9]
 [4 10 18]] [[ 1  4  9]
 [4 10 18]]
```

In [26]:

```
##Elementwise Division

print (x/y)
print (np.divide(x,y))
```

```
[[0.5        1.         1.5       ]
 [1.33333333 1.66666667 2.        ]] [[0.5      1.       1.5      ]
 [1.33333333 1.66666667 2.       ]]
```

In [27]:

```
# Inner product of vectors
print(np.dot(x, z), "\n")
```

```
[14 32]
```

PYTHON PANDAS

This is a multidimensional data structures and analysis tool for manipulating numerical

> Note: Rows represent **observations** while columns represent **input features**

Pandas Data Type

Recognised pandas data type includes:

In

- **object:** To represent text **int64:**
- Integer values

- **float64:** Floating point numbers **Category:** List of
- text values **bool:** True or false values **datetime64:**
- Date and time values **timedelta:** Difference
- between two datetimes
-

In [29]:

```
import pandas as pd
```

Ways to create pandas dataframe

In [30]:

```
# initialize list of lists
data = [['Ayo', 10], ['Imran', 15], ['Chucks', 14]]

# Create the pandas DataFrame from the list and adding column headers
df = pd.DataFrame(data, columns = ['Name', 'Age'])

# print dataframe.
df
```

Out[30]:

	Name	Age
0	Ayo	10
1	Imran	15
2	Chucks	14

In [31]:

```
# Create the pandas DataFrame from the dictionary of narray list
#Example 1:
# initialize list of lists
data = {'Name': ['Ayo', 'Imran','Chucks'] ,'Age':[10, 15, 14]}

# Create the pandas DataFrame from the list and adding column headers
df = pd.DataFrame(data)

# print dataframe.
df
```

Out[31]:

	Age	Name

In [34]:

0 10 Ayo

1 15 Imran

2 14 Chucks

[32]:

```
#Example 2:

#Population and area (km/square) of some states in Nigeria and their capital

dict_data = {"State": ["Abia", "Adamawa", "Lagos", "Osun", "Rivers"],
       "Capital": ["Umuahia", "Yola", "Ikeja", "Osogbo", "Portharcourt"],
       "area": [6320, 36917, 3345, 9251, 11077],
       "population" : [2845380, 3178950, 9113605, 3416959, 5198605] }

df = pd.DataFrame(dict_data)

df
```

Out[32]:

	Capital	State	area	population
0	Umuahia	Abia	6320	2845380
1	Yola	Adamawa	36917	3178950
2	Ikeja	Lagos	3345	9113605
3	Osogbo	Osun	9251	3416959
4	Portharcourt	Rivers	11077	5198605

In [34]:

```
df.dtypes
```

Out[34]:

Capital object State object
area int64 population
int64 dtype: object

In [35]:

ZIP

```
# pandas Datadaframe from lists using zip.

# List1
Name = ['Ayo', 'Imran','Chucks', 'judith']

# List2
Age = [25, 30, 26, 22]

# get the list of tuples from two list and merge them by using zip().
list_of_tuples = list(zip(Name, Age))

# Converting lists of tuples into pandas Dataframe.
df = pd.DataFrame(list_of_tuples , columns = ['Name', 'Age'])

# Print data.
df
```

Out[35]:

	Name	Age
0	Ayo	25
1	Imran	30
2	Chucks	26
3	judith	22

SERIES

A Series represents a single column in memory, which is either independent or belongs to a Pandas DataFrame.

In [36]:

```
# Pandas Dataframe from Dicts of series.

import pandas as pd

# Intialise data to Dicts of series.
series_data = {"State": pd.Series(["Abia", "Adamawa", "Lagos", "Osun", "Rivers"]),
       "Capital": pd.Series(["Umuahia", "Yola", "Ikeja", "Osogbo", "Portharcourt"]),
       "area": pd.Series([6320, 36917, 3345, 9251, 11077]),
       "population": pd.Series([2845380, 3178950, 9113605, 3416959, 5198605]) }

# creates Dataframe.
df = pd.DataFrame(series_data)

# print the data.
df
```

Out[36]:

	Capital	State	area	population
0	Umuahia	Abia	6320	2845380
1	Yola	Adamawa	36917	3178950
2	Ikeja	Lagos	3345	9113605
3	Osogbo	Osun	9251	3416959
4	Portharcourt	Rivers	11077	5198605

External source -

CSV Another way to create a DataFrame is by importing a csv file using pd.read_csv

```
csv_df = pd.read_csv('Data/2006.csv')

csv_df
```

Out[37]:

	STATES	AREA (km2)	Population
0	Abia State	6320	2845380
1	Adamawa State	36917	3178950
2	Akwa Ibom State	7081	3178950
3	Anambra State	4844	4177828
4	Bauchi State	45837	4653066
5	Bayelsa State	10773	1704515
6	Benue State	34059	4253641
7	Borno State	70898	4171104
8	Cross River	20156	2892988

In [37]:

9	Delta State	17698	4112445
10	Ebonyi State	5670	2176947
11	Edo State	17802	3233366
12	Ekiti State	6353	2398957
13	Enugu State	7161	3267837
14	FCT	7315	1405201
15	Gombe State	18768	2365040
16	Imo State	5530	3927563
17	Jigawa State	23154	4361002
18	Kaduna State	46053	6113503
19	Kano State	20131	9401288
20	Katsina State	24192	5801584
21	Kebbi State	36800	3256541
22	Kogi State	29833	3314043
23	Kwara State	36825	2365353
24	Lagos State	3345	9113605
25	Nasarawa State	27117	1869377
26	Niger State	76363	3954772
27	Ogun State	16762	3751140
28	Ondo State	15500	3460877
29	Osun State	9251	3416959
30	Oyo State	28454	5580894
31	Plateau State	30913	3206531
	STATES	**AREA (km2)**	**Population**
32	Rivers State	11077	5198605
33	Sokoto State	25973	3702676
34	Taraba State	54473	2294800
35	Yobe State	45502	2321339
36	Zamfara State	39762	3278873

EXCEL- XLSX

In [38]:

```
Excel_df = pd.read_excel('Data/2006.xlsx' )

Excel_df
```

Out[38]:

	STATES	AREA (km2)	Population
0	Abia State	6320	2845380
1	Adamawa State	36917	3178950
2	Akwa Ibom State	7081	3178950
3	Anambra State	4844	4177828
4	Bauchi State	45837	4653066
5	Bayelsa State	10773	1704515
6	Benue State	34059	4253641
7	Borno State	70898	4171104
8	Cross River	20156	2892988
9	Delta State	17698	4112445
10	Ebonyi State	5670	2176947
11	Edo State	17802	3233366
12	Ekiti State	6353	2398957
13	Enugu State	7161	3267837
14	FCT	7315	1405201
15	Gombe State	18768	2365040
16	Imo State	5530	3927563
17	Jigawa State	23154	4361002
18	Kaduna State	46053	6113503
19	Kano State	20131	9401288
20	Katsina State	24192	5801584
21	Kebbi State	36800	3256541
22	Kogi State	29833	3314043
23	Kwara State	36825	2365353
24	Lagos State	3345	9113605
25	Nasarawa State	27117	1869377
26	Niger State	76363	3954772
27	Ogun State	16762	3751140
28	Ondo State	15500	3460877
29	Osun State	9251	3416959
30	Oyo State	28454	5580894
31	Plateau State	30913	3206531

In

	STATES	AREA (km2)	Population
32	Rivers State	11077	5198605
33	Sokoto State	25973	3702676
34	Taraba State	54473	2294800
35	Yobe State	45502	2321339
36	Zamfara State	39762	3278873

In [39]:

```
#By default, if no length is specified, it returns the first 5 rows
print(csv_df.head(), '\n')

#This returns the first 5 rows in Population Column
print(csv_df['Population' ].head())
```

```
      STATES   AREA (km2)    Population
0   Abia State      6320    2845380
1   Adamawa State     36917    3178950
2   Akwa Ibom State      7081    3178950
3   Anambra State      4844    4177828
4   Bauchi State      45837    4653066
0   2845380
1   3178950
2   3178950
3   4177828
4   4653066
Name: Population, dtype: int64
```

In [40]:

```
#By default, if no length is specified, it returns the last 5 rows
print(csv_df.tail(), '\n')

#This returns the last 5 rows in Population Column
print(csv_df['Population' ].tail())
```

```
      STATES   AREA (km2)    Population
32  Rivers State      11077    5198605
33  Sokoto State      25973    3702676
34  Taraba State      54473    2294800
35  Yobe State      45502    2321339
36  Zamfara State      39762    3278873
32  5198605
33  3702676
34  2294800
35  2321339
36  3278873
Name: Population, dtype: int64
```

[41]:

In

```
#For summary of descriptive statistics of the dataframe
csv_df.describe()
```

Out[41]:

	AREA (km2)	Population
count	37.000000	3.700000e+01
mean	24990.864865	3.775879e+06 **std**
	18243.870444	1.726418e+06 **min**
	3345.000000	1.405201e+06 **25%**
	9251.000000	2.845380e+06
50%	20156.000000	3.314043e+06
75%	36800.000000	4.177828e+06 **max**
	76363.000000	9.401288e+06

In [42]:

```
#To include summary of descriptive statistics of non numeric columns of the dataframe csv_df.describe(include='all')
```

Out[42]:

	STATES	AREA (km2)	Population
count	37	37.000000	3.700000e+01
unique	37	NaN	NaN
top	Ekiti State	NaN	NaN
freq	1	NaN	NaN
mean	NaN	24990.864865	3.775879e+06 **std** NaN
	18243.870444	1.726418e+06 **min** NaN 3345.000000	
	1.405201e+06 **25%** NaN 9251.000000 2.845380e+06		
50%	NaN	20156.000000	3.314043e+06
75%	NaN	36800.000000	4.177828e+06 **max** NaN
	76363.000000	9.401288e+06	

In [43]:

```
csv_df['Population'].mean()
```

In

Out[43]:

3775879.4594594594

Other descriptive statistics functions are:

- count() Number of non-null observations
- sum() Sum of values
- mean() Mean of Values
- median() Median of Values
- mode() Mode of values
- std() Standard Deviation of the Values
- min() Minimum Value
- max() Maximum Value
- abs() Absolute Value
- prod() Product of Values
- cumsum() Cumulative Sum
- cumprod() Cumulative Product

> Note: Functions like abs(), cumprod() throw exception when the DataFrame contains character or string data because such operations cannot be performed.

In [44]:

```
#To show the features in the dataset
csv_df.columns
```

Out[44]:

Index(['STATES', 'AREA (km2) ', 'Population'], dtype='object')

In [45]:

```
#To show even more information about the dataset

csv_df.info()
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37 entries, 0 to 36 Data columns
(total 3 columns):
STATES 37 non-null object AREA (km2)
37 non-null int64 Population 37 non-null
int64 dtypes: int64(2), object(1) memory usage:
968.0+ bytes

Pandas Idexing

There are several ways to index a Pandas DataFrame. These are:

Square bracket notation: One of the easiest ways to do this is by using square bracket notation.

In

Loc and iloc: loc is label-based, which means that you have to specify rows and columns based on their row and column labels. iloc is integer index based, so you have to specify rows and columns by their integer index Dot (.) notation:

[46]:

```
#Square bracket notation to access all observations of selected features

# Print out states column as Pandas Series
print(csv_df['STATES'])

# Print out state column as Pandas DataFrame
print(csv_df[['STATES']])

# Print out DataFrame with states and population columns
print(csv_df[['STATES', 'Population']])
```

```
0          Abia State
1       Adamawa State
2     Akwa Ibom State
3      Anambra State
4       Bauchi State
5       Bayelsa State
6        Benue State
7        Borno State
8        Cross River
9        Delta State
10       Ebonyi State
11          Edo State
12       Ekiti State
13       Enugu State
14               FCT
15       Gombe State
16         Imo State
17       Jigawa State
18       Kaduna State
```

Note: A single square bracket will output a pandas series while a double square bracket outputs a pandas dataframe

In [47]:

```
#To access features of selected observations (rows) from a DataFrame, square bracket can

# Print out first 4 observations
print(csv_df[0:4], '\n')

# Print out fifth, sixth, and seventh observation
print(csv_df[4:6])
```

```
          STATES  AREA (km2)   Population
0     Abia State        6320      2845380
1  Adamawa State       36917      3178950
2  Akwa Ibom State      7081      3178950
```

In

3	Anambra State	4844	4177828
STATES	AREA (km2)	Population	
4	Bauchi State	45837	4653066
5	Bayelsa State	10773	1704515

[48]:

```
#Using Loc and Iloc

#since the dataset contains no label-based index, we can only use interger based iloc

# Print out observation for the third state
print(csv_df.iloc[2])

# Print out observation for the 4th and 5th state
print(csv_df.iloc[3:5])
```

```
STATES          Akwa Ibom State
AREA (km2)              7081
Population           3178950
Name: 2, dtype: object
      STATES  AREA (km2)  Population
3   Anambra State        4844    4177828
4   Bauchi State        45837    4653066
```

Deleting features/rows in datasets

In [49]:

```
# Drop rows of column called population
df = csv_df.drop(['Population' ], axis = 1)

print (df)
```

```
             STATES  AREA (km2)
0        Abia State          6320
1     Adamawa State         36917
2   Akwa Ibom State          7081
3     Anambra State          4844
4      Bauchi State         45837
5     Bayelsa State         10773
6       Benue State         34059
7       Borno State         70898
8       Cross River         20156
9       Delta State         17698
10     Ebonyi State          5670
11        Edo State         17802
12      Ekiti State          6353
13      Enugu State          7161
14              FCT          7315
15      Gombe State         18768
16        Imo State          5530
17     Jigawa State         23154
```

#using del function del

df['Population'] print(df)

using pop function

df.pop('Population') print (df)
d pop(opu at o) p t (d)

ADDITING TO DATASET

[52]:

```
# adding more features to datset

df['Population' ] = csv_df['Population' ]
df
```

Out[52]:

	STATES	AREA (km2)	Population
0	Abia State	6320	2845380
1	Adamawa State	36917	3178950
2	Akwa Ibom State	7081	3178950
3	Anambra State	4844	4177828
4	Bauchi State	45837	4653066
5	Bayelsa State	10773	1704515
6	Benue State	34059	4253641
7	Borno State	70898	4171104
8	Cross River	20156	2892988
9	Delta State	17698	4112445
10	Ebonyi State	5670	2176947
11	Edo State	17802	3233366
12	Ekiti State	6353	2398957
13	Enugu State	7161	3267837
14	FCT	7315	1405201
15	Gombe State	18768	2365040
16	Imo State	5530	3927563
17	Jigawa State	23154	4361002
18	Kaduna State	46053	6113503
19	Kano State	20131	9401288
20	Katsina State	24192	5801584
21	Kebbi State	36800	3256541
22	Kogi State	29833	3314043
23	Kwara State	36825	2365353
24	Lagos State	3345	9113605
25	Nasarawa State	27117	1869377
26	Niger State	76363	3954772
27	Ogun State	16762	3751140
28	Ondo State	15500	3460877
29	Osun State	9251	3416959
30	Oyo State	28454	5580894

31	Plateau State	30913	3206531
32	Rivers State	11077	5198605
33	Sokoto State	25973	3702676
34	Taraba State	54473	2294800
35	Yobe State	45502	2321339
36	Zamfara State	39762	3278873

STATES **AREA (km2)**

Population

Changing Data type of Pandas datafram and pandas series

[53]:

```
#changing the dtype of features for Series object

df['Population' ] = df['Population' ].astype('float')

df

#or with the use of downcasting

pd.to_numeric(df['Population' ], downcast='integer' )
```

Out[53]:

```
0   2845380
1   3178950
2   3178950
3   4177828
4   4653066
5   1704515
6   4253641
7   4171104
8   2892988
9   4112445
10  2176947
11  3233366
12  2398957
13  3267837
14  1405201
15  2365040
16  3927563
17  4361002
18  6113503
19  9401288
20  5801584
21  3256541
22  3314043
23  2365353
24  9113605
25  1869377
26  3954772
27  3751140
28  3460877
29  3416959
30  5580894
31  3206531
32  5198605
33  3702676
34  2294800
35  2321339
36  3278873
```

Name: Population, dtype: int32 In [54]:

#changing the dtype of features for pandas dataframe

```
df[['Population', 'AREA (km2) ']] = df[['Population', 'AREA (km2) ']].astype(float) df[['Population', 'AREA (km2) ']]
```

Out[54]:

	Population	AREA (km2)
0	2845380.0	6320.0
1	3178950.0	36917.0
2	3178950.0	7081.0
3	4177828.0	4844.0
4	4653066.0	45837.0
5	1704515.0	10773.0
6	4253641.0	34059.0
7	4171104.0	70898.0
8	2892988.0	20156.0
9	4112445.0	17698.0
10	2176947.0	5670.0
11	3233366.0	17802.0
12	2398957.0	6353.0
13	3267837.0	7161.0
14	1405201.0	7315.0
15	2365040.0	18768.0
16	3927563.0	5530.0
17	4361002.0	23154.0
18	6113503.0	46053.0
19	9401288.0	20131.0
20	5801584.0	24192.0
21	3256541.0	36800.0
22	3314043.0	29833.0
23	2365353.0	36825.0
24	9113605.0	3345.0
25	1869377.0	27117.0
26	3954772.0	76363.0
27	3751140.0	16762.0
28	3460877.0	15500.0
29	3416959.0	9251.0
30	5580894.0	28454.0

	Population	AREA (km2)
31	3206531.0	30913.0
32	5198605.0	11077.0
33	3702676.0	25973.0
34	2294800.0	54473.0
35	2321339.0	45502.0
36	3278873.0	39762.0

In [55]:

```
#Adding a new column using the existing columns in DataFrame

df['AreaPopu' ]=df['AREA (km2) ' ]+df['Population' ]

df.columns
```

Out[55]:

```
Index(['STATES', 'AREA (km2) ', 'Population', 'AreaPopu'], dtype='object')
```

> Note: the new feature column must be of the same dimension existing columns

Pandas Method

Sorting

Pandas sorting could be done either by using index or value

In [56]:

```
df.sort_index(inplace = True, ascending = False)

df
```

Out[56]:

	STATES	AREA (km2)	Population	AreaPopu
36	Zamfara State	39762.0	3278873.0	3318635.0
35	Yobe State	45502.0	2321339.0	2366841.0
34	Taraba State	54473.0	2294800.0	2349273.0
33	Sokoto State	25973.0	3702676.0	3728649.0
32	Rivers State	11077.0	5198605.0	5209682.0
31	Plateau State	30913.0	3206531.0	3237444.0
30	Oyo State	28454.0	5580894.0	5609348.0
29	Osun State	9251.0	3416959.0	3426210.0
28	Ondo State	15500.0	3460877.0	3476377.0
27	Ogun State	16762.0	3751140.0	3767902.0

In [57]:

```
# sorting data frame by values. The argument will use column name
#axis -> 0 means sorting will be done row-wise
#ascending -> False
df.sort_values("STATES", axis = 0, ascending = False,
                 inplace = True)
df
```

Out[57]:

	STATES	AREA (km2)	Population	AreaPopu
36	Zamfara State	39762.0	3278873.0	3318635.0
35	Yobe State	45502.0	2321339.0	2366841.0
34	Taraba State	54473.0	2294800.0	2349273.0
33	Sokoto State	25973.0	3702676.0	3728649.0
32	Rivers State	11077.0	5198605.0	5209682.0
31	Plateau State	30913.0	3206531.0	3237444.0
30	Oyo State	28454.0	5580894.0	5609348.0
29	Osun State	9251.0	3416959.0	3426210.0
28	Ondo State	15500.0	3460877.0	3476377.0
27	Ogun State	16762.0	3751140.0	3767902.0
26	Niger State	76363.0	3954772.0	4031135.0
25	Nasarawa State	27117.0	1869377.0	1896494.0
24	Lagos State	3345.0	9113605.0	9116950.0
23	Kwara State	36825.0	2365353.0	2402178.0
22	Kogi State	29833.0	3314043.0	3343876.0
21	Kebbi State	36800.0	3256541.0	3293341.0
20	Katsina State	24192.0	5801584.0	5825776.0
19	Kano State	20131.0	9401288.0	9421419.0
18	Kaduna State	46053.0	6113503.0	6159556.0
17	Jigawa State	23154.0	4361002.0	4384156.0
16	Imo State	5530.0	3927563.0	3933093.0
15	Gombe State	18768.0	2365040.0	2383808.0
14	FCT	7315.0	1405201.0	1412516.0
13	Enugu State	7161.0	3267837.0	3274998.0
12	Ekiti State	6353.0	2398957.0	2405310.0
11	Edo State	17802.0	3233366.0	3251168.0
10	Ebonyi State	5670.0	2176947.0	2182617.0
9	Delta State	17698.0	4112445.0	4130143.0
8	Cross River	20156.0	2892988.0	2913144.0

	STATES	AREA (km2)	Population	AreaPopu
6	Benue State	34059.0	4253641.0	4287700.0
5	Bayelsa State	10773.0	1704515.0	1715288.0
4	Bauchi State	45837.0	4653066.0	4698903.0
3	Anambra State	4844.0	4177828.0	4182672.0
2	Akwa Ibom State	7081.0	3178950.0	3186031.0
1	Adamawa State	36917.0	3178950.0	3215867.0
0	Abia State	6320.0	2845380.0	2851700.0

Pandas DataFrame String operations

Method	Description
lower()	Converts strings in the Series/Index to lower case.
upper()	Converts strings in the Series/Index to upper case.
len()	Computes string length
strip()	Helps strip whitespace(including newline) from each string in the Series/index from both the sides
split(' ')	Splits each string with the given pattern.
cat(sep=' ')	Concatenates the series/index elements with given separator.
get_dummies()	Returns the DataFrame with One-Hot Encoded values.
contains(pattern)	Returns a Boolean value True for each element if the substring contains in the element, else False.
replace(a,b)	Replaces the value a with the value b.
repeat(value)	Repeats each element with specified number of times.
count(pattern)	Returns count of appearance of pattern in each element.
startswith(pattern)	Returns true if the element in the Series/Index starts with the pattern.
endswith(pattern)	Returns true if the element in the Series/Index ends with the pattern.
find(pattern)	Returns the first position of the first occurrence of the pattern.
findall(pattern)	Returns a list of all occurrence of the pattern.
swapcase	Swaps the case lower/upper.
islower()	Checks whether all characters in each string in the Series/Index in lower case or not. Returns Boolean
isupper()	Checks whether all characters in each string in the Series/Index in upper case or not. Returns Boolean.
isnumeric()	Checks whether all characters in each string in the Series/Index are numeric. Returns Boolean.

In [ ]:

for further reading:

https://pbpython.com/pandas_dtypes.html (https://pbpython.com/pandas_dtypes.html)

https://en.wikipedia.org/wiki/Matrix_(mathematics) (https://en.wikipedia.org/wiki/Matrix_(mathematics))

https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/

(https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/)

In [ ]:

Day 3

SUPERVISED LEARNING: INTRODUCTION TO REGRESSION ANALYSIS

REGRESSION ANALYSIS

This is the first lesson in which we will be specifically focusing on machine learning algorithms and their practical implementation in Python using real world datasets. We will start with Regression Analysis and Predictive Modelling.
We assume that you have basic knowledge of Python programming and have used the basic libraries such as NumPy, SciPy, Pandas, Matplotlib and scikit-learn,as mentioned in the last week's notes. If you haven't, here are some links for you.

1. NumPy
2. SciPy
3. Pandas
4. Matplotlib
5. scikit-learn

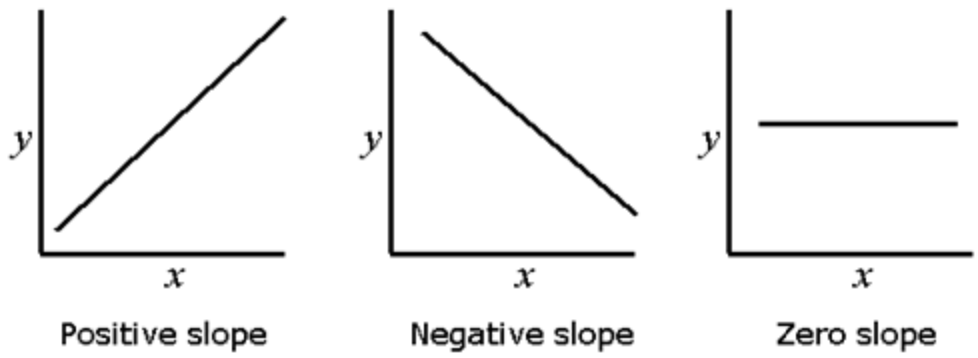
First, we will have a quick introduction to building models in Python, and what better way to start than one of the very basic models, linear regression? Linear regression will be the first algorithm used and you will also learn more complex regression models.

First, we will have a quick introduction to building models in Python, and what better way to start than one of the very basic models, linear regression? Linear regression will be the first algorithm used and you will also learn more complex regression models.

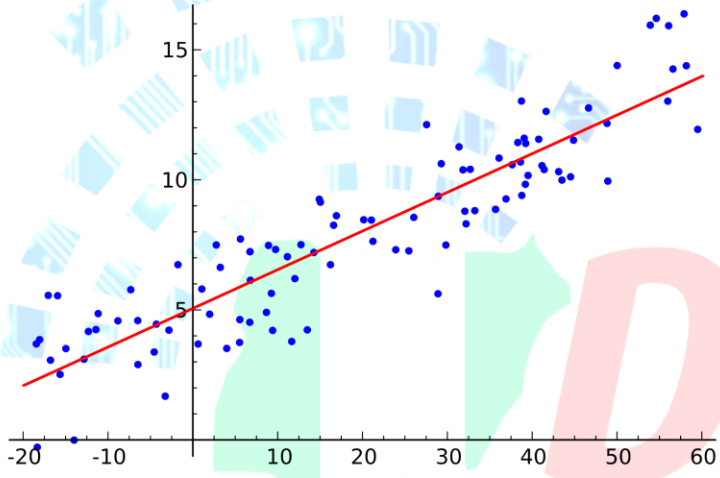
The Concept of Linear Regression

Linear Regression is considered the most natural learning algorithm for modelling data, primarily because it is easy to interpret and models efficiently for most natural problems. It belongs to the family of "Linear Models/Predictors" in machine learning(one of the most useful hypothesis space)
Linear regression is a statistical model that examines the linear relationship between two (simple linear regression, or SLR) or more (multiple linear regression, or MLR) variables, a dependent variable and independent variable(s)

A linear relationship basically means that when one (or more) independent variables increase (or decrease), the dependent variable increases (or decreases) too.



As you can see, a linear relationship can be positive (the independent variable goes up, and dependent variable goes up) or negative (the independent variable goes up, but the dependent variable goes down).



A Little Bit About the Math and the Intuition

Mathematically, the linear regression model can be defined by a dependent variable 'Y', also called the regressand and an independent variable(or set of independent variables) 'X', also called the regressor(s), and a sample space 'n'.

Let Y be a dependent variable and X an independent variable. Then we can define the general form of a simple linear regression model as:

$y = \beta_0 + \beta_1 X + \varepsilon$.....(1)

where $\beta's$ are the parameters to be estimated and ε the random error. A population model for a multiple linear regression that relates an independent variable y_i with k1 predictor variables is written as: $y_i = \beta_0 +$

$\beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_k x_{ik} + (i = i, 2, ..., n)$.....(2)

where $\beta's$ are the parameters to be estimated. In practice, the exact value of y cannot be found. So the estimate $\hat{y}$ of y is given as:

$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 \hat{X} + \varepsilon$.....(3)

Assumptions of the Linear Regression Model

- The linear regression model assumes a linear relationship between the dependent and independent variables.

$$y = X\beta + \varepsilon$$

Then, $E(y|X) = \beta X$

- It assumes the variables follow a normal distribution and that the features are Identically Independently distributed(I.I.D) with Zero mean($\mu = 0$) and variance σ^2

$$\varepsilon_i \sim N(0,\sigma^2)$$

$$Cov(\varepsilon_i,\varepsilon_j) = 0, \text{ if } i \neq j$$

- Little or no auto-correlation.Given X of dimension (n x k) , then rank(X) = k • No Homoscedasticity;

$$\forall i = 1, 2, ...n; var(\varepsilon_i) = \sigma_2$$

LINEAR REGRESSION AS REGRESSION ALGORITHM

As a learning algorithm, the linear regression models the relationship between some “explanatory” variables(X) and some real valued outcome(Y).

The linear regression falls under the context of Supervised Learning. It is classified as a regression algorithm because its output is a range of continuous real values. This set of algorithms models the data to:

Fit a line of the form: $y = X\theta$, $y \in R_N$. Where N= size of the data

Loss function: The squared loss function is one common way of measuring the discrepancy between the predicted values and the actual values. It is given as:

$$L(h, (x, y)) = (h(x) - y)_2.$$

The empirical risk function equivalent for this loss function is the Mean Squared Error(Error Minimization of the difference between Predicted ($Y^$) and Actual(Y)) i.e Mimimizing the square deviation(Mean Squared Error/ Cost Function/Error Metrics/Criteria

The Error / Criteria :

- This can be calculated using the Mean Squared Error, given by: $MSE = \frac{1}{N}\sum_{i=1}^{n}(h_\theta(x) - y^i)^2$

Where:

$h_\theta(x) = \hat{y}$

There are other loss functions such as the mean absolute error(MAE) and the root mean squared error(RMSE), However, we will focus on the MSE for this purpose.

TREE-BASED REGRESSION

Tree-based models are used for predicting a response given a set of explanatory variables when the regression function is characterized by a certain degree of complexity i.e when the function

to model is not linear. These types of models have gained popularity recently, as they are very efficient in modeling large datasets and can handle complexity.
All tree-based models can be used for either regression (predicting numerical values) or classification (predicting categorical values)
There are three common types in use;

1. Decision tree models: the foundation of all tree-based models.
2. Random forest models, an “ensemble” method which builds many decision trees in parallel.
3. Gradient boosting models, an “ensemble” method which builds many decision trees sequentially.

The tree based models use an approach of stratifying or segmenting the predictor space into a number of simple regions.
For the Decision tree models, it uses a simple decision process for if-else(conditional statements). Since the set of splitting rules used to segment the predictor space can be summarized in a tree.

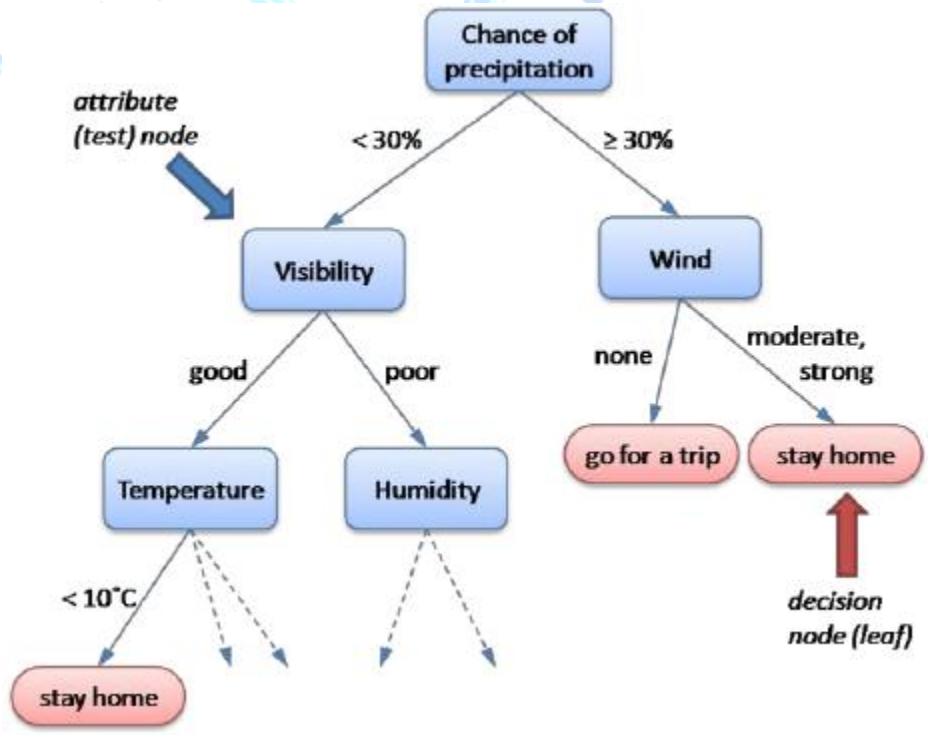


Fig: Example of Decision based model

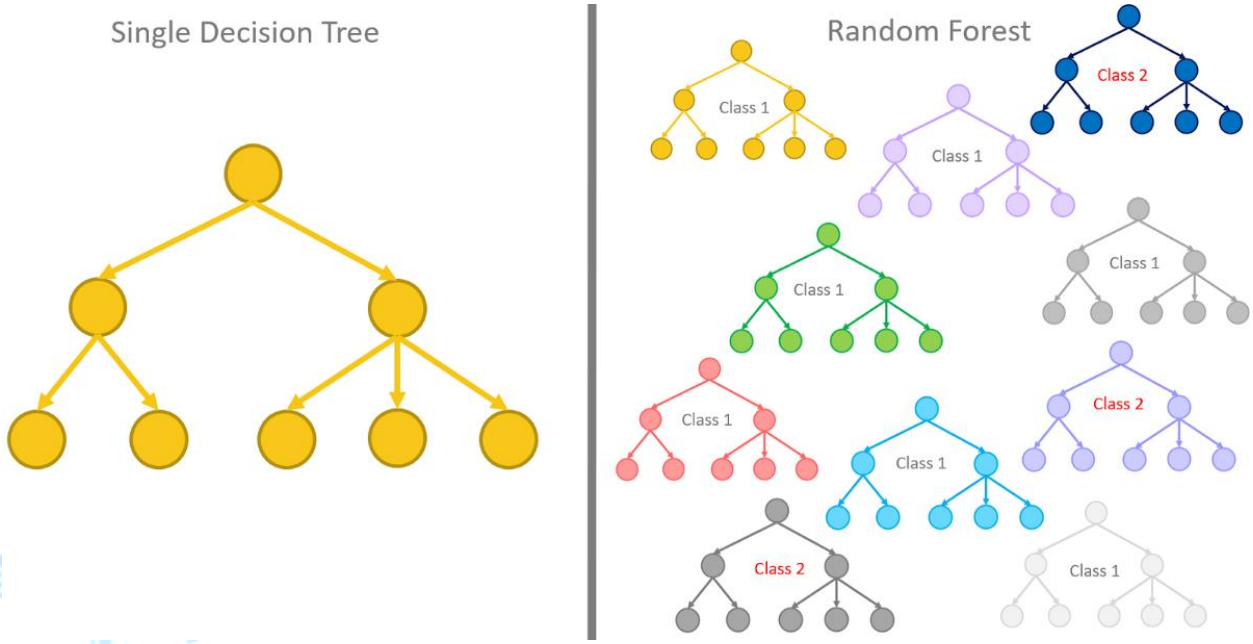


Fig: Random forest compared to single decision tree

For the gradient boosting tree models, it is simply converting the weak tree learners into strong learners using some sort of weight measure. Let's not go into much details for now.

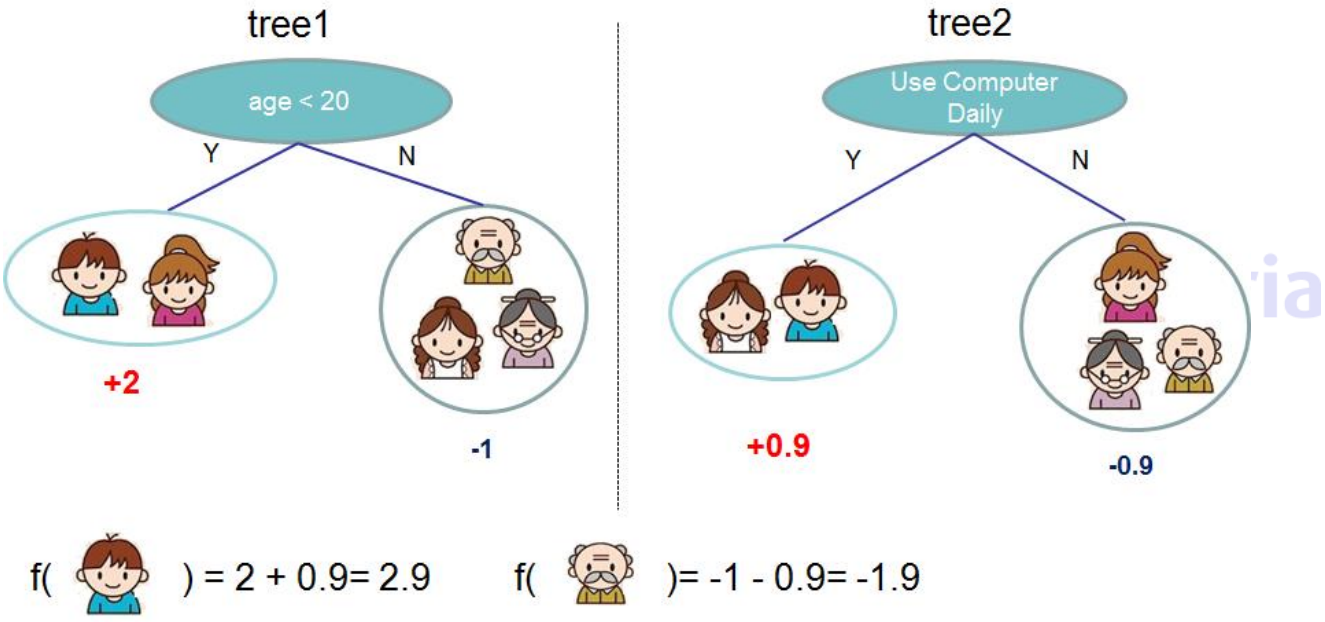


Fig: Gradient Boosting tree models

ACTIVITY FOR REGRESSION

Examples of tasks that can be solved using regression
Before a dataset can be train on a regression model, the label must be a continuous variable not discret

- Predicting salary from years of experience
- Determining Glucose level from Age of patients
- Predicting salary from years of experience
- Predicting students' grades based on total study time.
- Predicting examination score based on students' test score etc.

Regression Machine Learning Model using Mama Tee restaurant dataset.
The objective of the regression task is to predict the amount of tip (gratuity in Nigeria naira) given to a food server based on total_bill, gender, smoker (whether they smoke in the party or not), day (day of the week for the party), time (time of the day whether for lunch or dinner), and size (size of the party) in Mama Tee restaurant..

Label: The label for this problem is tip.

Features: There are 6 features and they include total bill, gender, smoker, day, time, and size.

We plan to use the following regression models (regressor) to predict the amount of tips that will be given during a particular party in the restaurant:

- Ordinary Least Square (OLS)
- Support Vector Machine (SVM)
- Extreme Gradient Boosting (XGBoost)
- Decision Tree
- Random Forest

Import Python modules
We need to import some packages that will enable us to explore the data and build machine learning models.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pandas_profiling import ProfileReport # Please install pandas profiling if you don't have it installed already
```

```
tip = pd.read_csv("../Data/tips.csv")

tip.head(10)
```

	total_bill	tip	gender	smoker	day	time	size
0	2125.50	360.79	Male	No	Thur	Lunch	1
1	2727.18	259.42	Female	No	Sun	Dinner	5
2	1066.02	274.68	Female	Yes	Thur	Dinner	4
3	3493.45	337.90	Female	No	Sun	Dinner	1
4	3470.56	567.89	Male	Yes	Sun	Lunch	6
5	2411.08	296.48	Female	Yes	Thur	Lunch	2
6	4607.43	374.96	Female	No	Thur	Dinner	4
7	1165.21	700.87	Female	No	Mon	Dinner	2
8	2895.04	347.71	Male	No	Sat	Dinner	5
9	2622.54	253.97	Male	Yes	Thur	Lunch	6

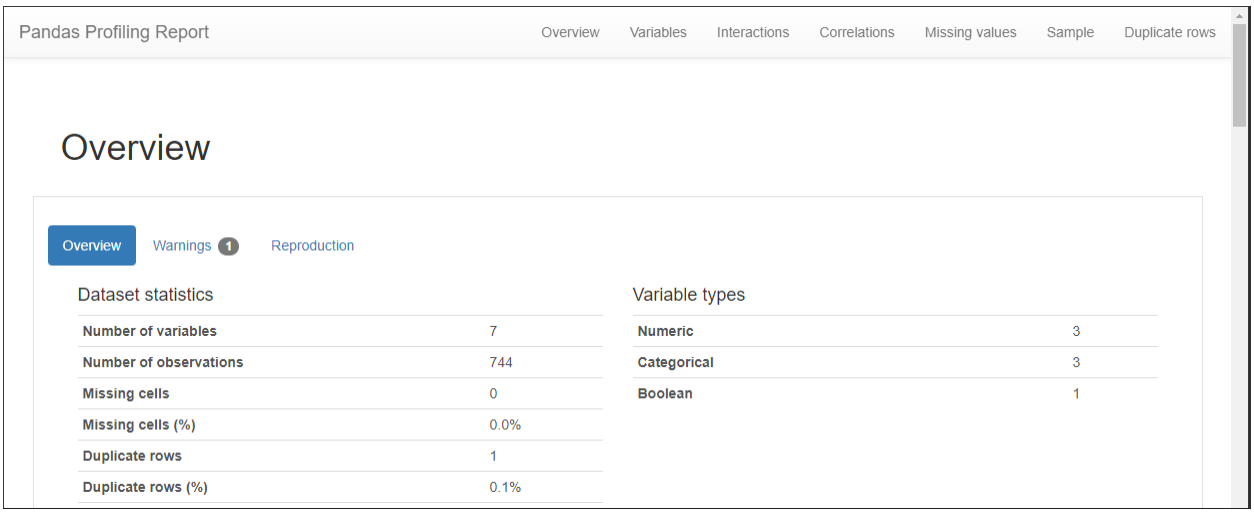
```
tip.shape
```

```
(744, 7)
```

We can use pandas_profiling to do some data exploration before training our models

```
tip.profile_report()
```

```
Summarize dataset:   0%|          | 0/20 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

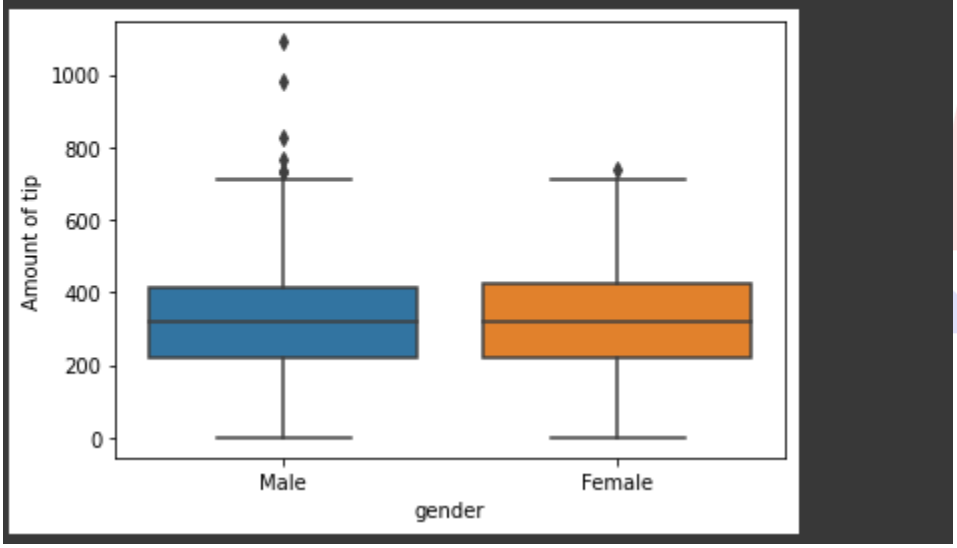


Relationship with categorical variables

tip vs. gender

```
sns.boxplot(x = "gender", y = "tip", data = tip)

plt.ylabel("Amount of tip");
```



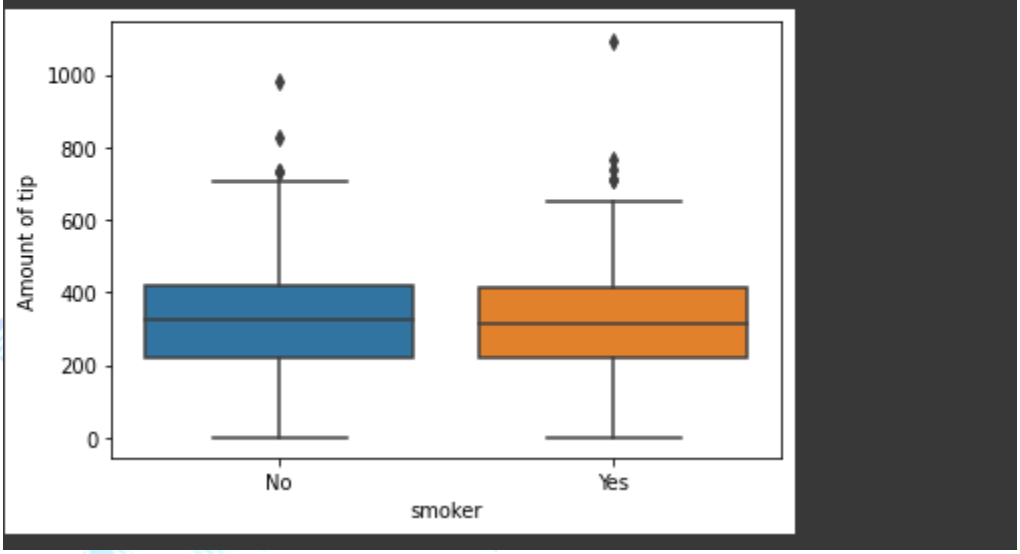

```
#sns.countplot(x = "gender" , hue = "tip", data = tip);
```

The amount of tips given by both gender is almost the same although there was an extreme amount of tip given by some men.

tip vs. smoker

```
sns.boxplot(x = "smoker", y = "tip", data = tip)

plt.ylabel("Amount of tip");
```

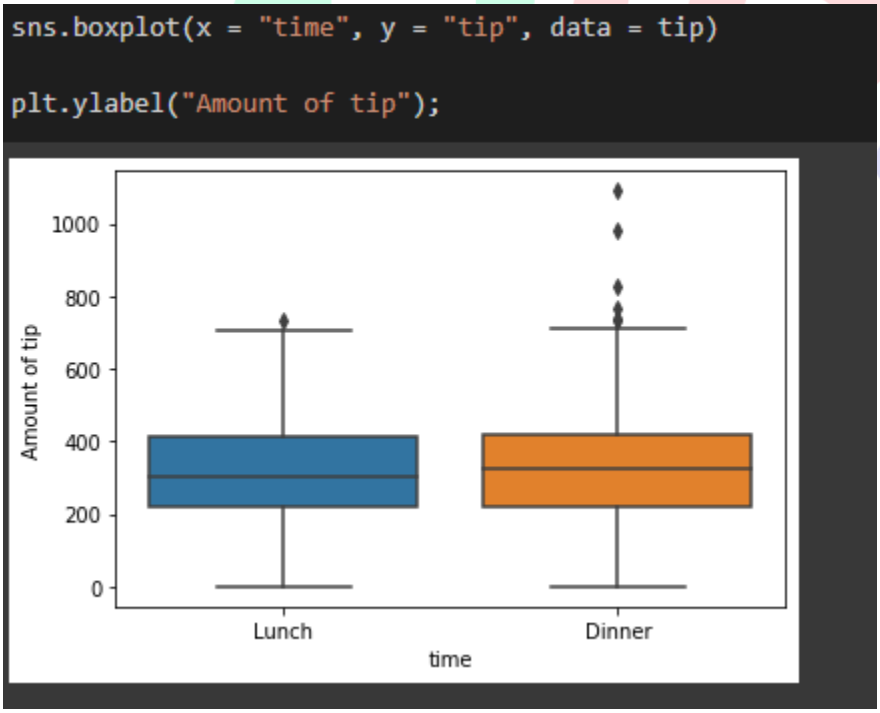


Smokers and non-smokers gave almost the same amount of tip.

tip vs. time

```
sns.boxplot(x = "time", y = "tip", data = tip)

plt.ylabel("Amount of tip");
```



Lunch and dinner gave almost the same amount of tip.

Model building

After getting some insight about the data, we can now prepare the data for machine learning modelling

- Importing machine learning models.

```
from sklearn import metrics # For model evaluation

from sklearn.model_selection import train_test_split # To divide the data into training and test set
```

Data Preprocessing

- Separating features and the label from the data

Now is the time to build machine learning models for the task of predicting the amount of tip that would be given for any party in the restaurant. Therefore, we shall separate the set of features (X) from the label (Y).

```
tip.head(4)
```

	total_bill	tip	gender	smoker	day	time	size
0	2125.50	360.79	Male	No	Thur	Lunch	1
1	2727.18	259.42	Female	No	Sun	Dinner	5
2	1066.02	274.68	Female	Yes	Thur	Dinner	4
3	3493.45	337.90	Female	No	Sun	Dinner	1

```
# split data into features and target

X = tip.drop(["tip"], axis= "columns") # dropping the label variable (tip) from the data

y = tip["tip"]
```

```
X.head()
```

	total_bill	gender	smoker	day	time	size
0	2125.50	Male	No	Thur	Lunch	1
1	2727.18	Female	No	Sun	Dinner	5
2	1066.02	Female	Yes	Thur	Dinner	4
3	3493.45	Female	No	Sun	Dinner	1
4	3470.56	Male	Yes	Sun	Lunch	6

```
y.head()
```

```
0    360.79
1    259.42
2    274.68
3    337.90
4    567.89
Name: tip, dtype: float64
```

Since the label is continuous, this is a regression task.

- One-hot encoding

As discussed in Part 3, we need to create a one-hot encoding for all the categorical features in the data because some algorithms cannot work with categorical data directly. They require all input variables and output variables to be numeric. In this case, we will create a one-hot encoding for gender, smoker, day and time by using pd.get_dummies().

```
pd.get_dummies(X)
```

	total_bill	size	gender_Female	gender_Male	smoker_No	smoker_Yes	day_Fri	day_Mon	day_Sat	day_Sun	day_Thur	day_Tues	day_Wed	time_Dinner	time_Lunch
0	2125.50	1	0	1	1	0	0	0	0	0	1	0	0	0	1
1	2727.18	5	1	0	1	0	0	0	0	1	0	0	0	1	0
2	1066.02	4	1	0	0	1	0	0	0	0	1	0	0	1	0
3	3493.45	1	1	0	1	0	0	0	0	1	0	0	0	1	0
4	3470.56	6	0	1	0	1	0	0	0	1	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
739	3164.27	3	0	1	1	0	0	0	1	0	0	0	0	1	0
740	2962.62	2	1	0	0	1	0	0	1	0	0	0	0	1	0
741	2471.03	2	0	1	0	1	0	0	1	0	0	0	0	1	0
742	1942.38	2	0	1	1	0	0	0	1	0	0	0	0	1	0
743	2047.02	2	1	0	1	0	0	0	0	0	1	0	0	1	0

744 rows × 15 columns

We now save this result of one-hot encoding into X.

```
X = pd.get_dummies(X)
```

```
X.head()
```

	total_bill	size	gender_Female	gender_Male	smoker_No	smoker_Yes	day_Fri	day_Mon	day_Sat	day_Sun	day_Thur	day_Tues	day_Wed	time_Dinner	time_Lunch
0	2125.50	1	0	1	1	0	0	0	0	0	1	0	0	0	1
1	2727.18	5	1	0	1	0	0	0	0	1	0	0	0	1	0
2	1066.02	4	1	0	0	1	0	0	0	0	1	0	0	1	0
3	3493.45	1	1	0	1	0	0	0	0	1	0	0	0	1	0
4	3470.56	6	0	1	0	1	0	0	0	1	0	0	0	0	1

```
X.shape
```

```
(744, 15)
```

- Split the data into training and test set

We will split our dataset (Features (X) and Label (Y)) into training and test data by using train_test_split() function from sklearn. The training set will be 80% while the test set will be 20%. The random_state that is set to 1234 is for all of us to have the same set of data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state= 1234)
```

We now have the pair of training data (X_train, y_train) and test data (X_test, y_test)

- Model training

We will use the training data to build the model and then use test data to make prediction and evaluation respectively.

Linear Regression

Let's train a linear regression model with our training data. We need to import the Linear regression from the sklearn model.

```
# Fitting Linear Regression to the Training set

from sklearn.linear_model import LinearRegression
```

We now create an object of class LinearRegression to train the model on

```
linearmodel = LinearRegression()

linearmodel.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

linearmodel.fit trained the Linear regression model. The model is now ready to make predictions for the unknown label by using only the features from the test data (X_test).

```
linearmodel.predict(X_test)
```

```
array([367.43848391, 284.17027121, 274.19116018, 346.46402511,
       298.85746332, 294.14453677, 257.66130848, 283.23527194,
       311.53154902, 266.76688909, 287.50691095, 268.12328087,
       319.29057874, 379.389987  , 298.34394797, 344.98937081,
       375.77087624, 345.96103113, 257.77234067, 383.58762963,
       301.02748847, 296.47261286, 377.37706494, 380.998273  ,
       323.44463928, 325.96091299, 296.45549431, 295.80850357,
       376.11352327, 394.93799767, 314.34660701, 252.39149885,
       371.36736855, 329.21139694, 309.21545846, 335.19229996,
       353.93437564, 292.97091347, 329.96148959, 302.46622122,
       248.40367105, 328.12726277, 343.86442963, 343.74524418,
       389.50836497, 321.67127403, 344.84535554, 358.10847206,
       328.46831958, 391.63641292, 307.10446628, 359.07045622,
       362.31824303, 386.93674845, 371.71312874, 326.74602401,
       327.49791297, 345.53085245, 323.75874404, 351.67709068,
       319.64524402, 313.14934587, 348.15326222, 360.45811712,
       306.91179633, 319.03745527, 328.04033607, 352.84161322,
       266.22156106, 477.16606715, 301.82766308, 303.90383646,
       428.42015497, 359.69267108, 314.79886804, 278.82273827,
       365.89102395, 370.66552738, 361.86935996, 361.77603556,
       308.23605818, 324.76075509, 319.54290807, 297.48867913,
       288.95904974, 318.41254282, 272.37056957, 336.74732874,
       301.46625143, 320.17217658, 342.05461416, 257.83682254,
       327.32758401, 346.17248349, 299.41684507, 329.94144723,
       357.78937594, 318.81996671, 354.70399229, 348.94553361,
       306.10788825, 322.62891935, 273.7982918 , 309.45524806,
       331.57805986, 305.95972438, 353.91856112, 326.29069159,
       286.90692453, 340.83773251, 233.98704784, 285.59478073,
       316.56499468, 285.77371173, 328.88911506, 331.82533493,
       313.90738322, 357.35779187, 305.36187312, 362.77983922,
       293.6591636 , 329.95139692, 375.65561297, 313.8301789 ,
       396.44799953, 290.74699689, 307.71970097, 336.9735287 ,
```

let's save the prediction result into linearmodel_prediction. This is what the model predicted for us.

```
linearmodel_prediction = linearmodel.predict(X_test)
```

Model Evaluation

Since the prediction is continuous, we can only measure how far the prediction is from the actual values. Let's check the error for each prediction.

```
y_test - linearmodel_prediction
```

```
535     24.961516
718   -127.210271
277    117.118840
391     29.585975
586    -80.857463
          ...
28     251.233516
146   -113.455516
616    159.553272
234     27.214263
359     33.720087
Name: tip, Length: 149, dtype: float64
```

The positive ones show that the prediction is higher than the actual values while the negative ones are below the actual values. Let's now measure this error by using the Root Mean Squared Error (RMSE).

```
MSE = metrics.mean_squared_error(y_test, linearmodel_prediction)
```

```
MSE
```

```
20201.41527694898
```

We now take the square root of the Mean Squared Error to get the value of the RMSE.

```
np.sqrt(MSE)
```

```
142.1316828752442
```

Therefore, the RMSE for the linear regression is 142.1316828752442.

Random Forest Model

Let's train a Random Forest model with our training data. We need to import the model from the sklearn module.

```
from sklearn.ensemble import RandomForestRegressor

randomforestmodel = RandomForestRegressor()

randomforestmodel.fit(X_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

randomforestmodel.fit() trained the Random Forest model on the training data. The model is now ready to make prediction for the unknown label by using only the features from the test data (X_test).

```
randomforestmodel_prediction = randomforestmodel.predict(X_test)

MSE = metrics.mean_squared_error(y_test, randomforestmodel_prediction)

MSE
```

```
25034.353288601207
```

We now take the square root of the Mean Squared Error to get the value of the RMSE.

```
np.sqrt(MSE)
```

```
154.49312068742455
```

Therefore, the RMSE for the Random Forest Model is 160.3155113080993.

Extreme Gradient Boost (XGBoost) Model

Let's train an XGBoost model with our training data. We need to import the XGBoost model from the xgboost module.

```
from xgboost import XGBRegressor # Please install xgboost libarary if you don't have it installed already

xgboostmodel = XGBRegressor(use_label_encoder=False)

xgbboostmodel = xgboostmodel.fit(X_train, y_train)
```

xgboostmodel.fit() trained the XGBoost model on the training data. The model is now ready to make prediction for the unknown label by using only the features from the test data (X_test).

```
xgbboostmodel_prediction = xgboostmodel.predict(X_test)
```

You can call on xgbboostmodel_prediction to see the prediction,

```
MSE = metrics.mean_squared_error(y_test, xgbboostmodel_prediction)

MSE

23702.66797889845
```

We now take the square root of the Mean Squared Error to get the value of the RMSE.

```
np.sqrt(MSE)

171.0289233753799
```

Therefore, the RMSE for the Gradient Boost (XGBoost) Model is 171.0289233753799

Support Vector Machine (SVM)

Let's train a Support Vector Machine model with our training data. We need to import the Support Vector Machine model from the sklearn module.

```
from sklearn.svm import SVR

SVMmodel = SVR()

SVMmodel.fit(X_train, y_train)

SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

SVMmodel.fit() trained the Support Vector Machine on the training data. The model is now ready to make prediction for the unknown label by using only the features from the test data (X_test).

```
SVMmodel_prediction = SVMmodel.predict(X_test)
```

You can call on SVMmodel_prediction to see what has been predicted.

```
MSE = metrics.mean_squared_error(y_test, SVMmodel_prediction)

MSE

19853.340298954365
```

We now take the square root of the Mean Squared Error to get the value of the RMSE.

```
np.sqrt(MSE)

140.90188181480886
```

Therefore, the RMSE for the Support Vector Machine (SVM) is 140.90188181480886
You can call on SVMmodel_prediction to see the prediction

Decision Tree

Let's train a Decision Tree model with our training data. We need to import the Decision Tree model from the sklearn module.

```
from sklearn.tree import DecisionTreeRegressor

decisiontree =  DecisionTreeRegressor()

decisiontree.fit(X_train, y_train)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

decisiontree.fit() trained the Decision Tree on the training data. The model is now ready to make prediction for the unknown label by using only the features from the test data (X_test).

```
decisiontree_prediction = decisiontree.predict(X_test)
```

You can call on decisiontree_prediction to see what has been predicted.

```
MSE = metrics.mean_squared_error(y_test, decisiontree_prediction)

MSE
```

```
47207.83781879195
```


We now take the square root of the Mean Squared Error to get the value of the RMSE.

```
np.sqrt(MSE)
```

```
222.1130565544185
```

Therefore, the RMSE for the Decision Tree is 215.84571333501313.

Models Summary

Having trained all the five (5) models, we can see that the best model that can accurately predict the amount of tips that would be given for a given party in the restaurant is the model with the lowest RMSE and that is Support Vector Machine (SVM).

Class Activity

Importing Scikit-learn Module

Use the following models to predict the amount of tips that would be given for a given party in the restaurant. Your teacher has also included how to import those models for you.

- **K Nearest Neighbor:** from sklearn.neighbors import KNeighborsRegressor
- **Ridge Regression:** from sklearn.linear_model import Ridge
- **Gradient Boost:** from sklearn.ensemble import GradientBoostingRegressor
- Which of the three (3) models is the best in terms of RMSE?

DAY 3: SUPERVISED LEARNING: CLASSIFICATION

Comparison between Regression and Classification

In the previous lesson, you were introduced to Regression analysis and different types of regression algorithms in Machine learning. In this lesson, you will explore the difference between the two different types of Supervised Machine Learning approaches; Regression and Comparison , then also get introduced to Classification and various Algorithms under this category.

Regression and Classification are both part of a class of Machine Learning Algorithms called "Supervised Learning Algorithms". Remember that Machine Learning is all about generating future occurrences/predictions of an event while making reference to past occurrences of the event in question. And for this particular class of Machine Learning(Supervised learning), we have a labelled data with known examples with which the algorithm is trained on to predict for unknown data. As we saw for regression, there is a target column containing the expected outcome for the problem, which in this case was in a continuous range of real values, and the prediction was also in this form. For the classification problem, it is slightly different. This time, the target value will be in a discrete range of values.

In Machine Learning, classification is considered a subset of Regression, hence you find many classes of algorithms applicable in both areas. One of the key differences between the both of them is how the problem is phrased, and the desired output should be. For regression, the problem is phrased as "prediction quantity" and the desired output is a continuous range of real values.

Whereas for classification, the problem is phrased as "predicting a label/A class" and the desired output is a discrete set of values.

In simpler terms, the classification is a sub-category of supervised Machine Learning methods where data points are grouped into classes as output, using a decision boundary to separate each class.

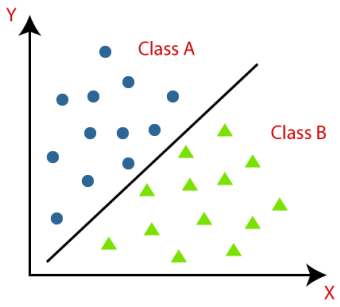


Fig: How Classification works

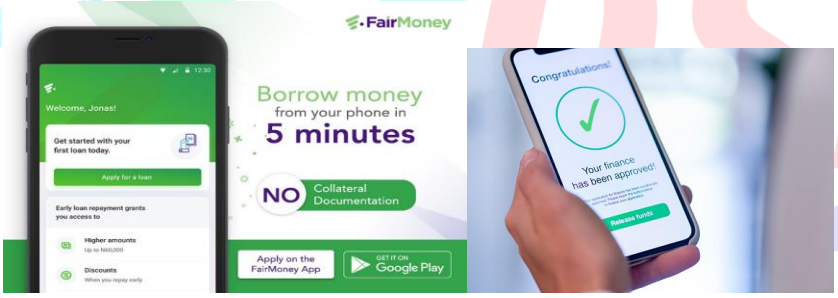
So while Regression algorithms can be used to solve the regression problems such as Weather Prediction, House price prediction, (predicting quantity) etc, Classification Algorithms are used to solve classification problems such as Identification of spam emails, Speech Recognition, Identification of cancer cells,(predicting labels/classes) etc. You certainly encountered/used classification models in real-life without even realizing it.Have you ever wondered how google filters your mail into Spam or not Spam? Or how recent smartphones use AI cameras and tag objects/assign labels such as Room, food , cat et cetera?

Other classical real world examples are;

- Customer behavior prediction: Customers can be classified in different categories based on their buying patterns, web store browsing patterns etc.(Jumia,amazon uses this, what is shown to you is not what is being shown to me)

- Insurance Prediction: If a particular customer will get an insurance or not;

- Loan Prediction/Credit-worthiness: Access bank and some other financial institutions in Nigeria and generally Africa are actively using this. Remember Paylater, Opay, Piggybank, how do they determine who pays back the loan? how do they determine who gets the loan? Or how much the customer can get? MTN uses this too, when you try to borrow credit.



Source: Here and Here

- Malware classification : Classify the new / emerging malwares on the basis of comparable features of similar/previous malwares.

- Medical diagnosis: used in the techniques and tools that can help in the diagnosis of diseases. It is used for the analysis of the clinical parameters and their combination for

the prognosis example prediction of disease progression for the extraction of medical knowledge for the outcome research, for therapy planning and patient monitoring.

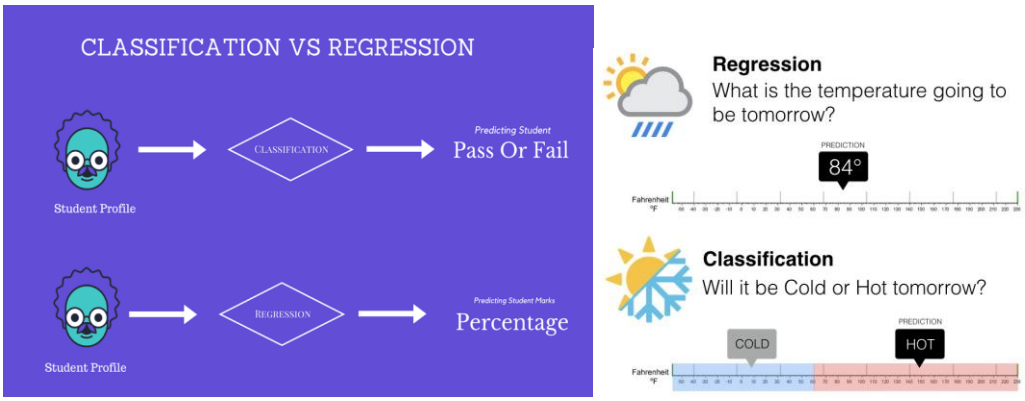


Fig: An example of how a problem is phrased differently for both regression and classification. Image Source: Here

One other important difference between both of them is the loss function used. In machine learning, the loss function, also called the evaluation metrics or criterion, is the measure used to test how well/accurate the machine learning algorithm is able to model the problem i.e how close or far is the predicted outputs from the ground truth.

Let's see a summary of the key similarities and differences between these two approaches.

SIMILARITIES:

- Both belong to the class of Machine Learning Algorithms called "Supervised Learning".
- Both model a problem by learning a mapping function/relationship from the input(X) to the output(Y), using known examples.

DIFFERENCES:

Property	Regression Algorithms	Classification Algorithms
Output/Target Variable	Predicts a continuous real value	Predicts a class/discrete values
Learning function	A line of best fit	A decision boundary
Evaluation metrics/Loss functions	Mean squared error, Mean absolute error, Root mean squared error	Accuracy, F1 score, Area under the curve

There are generally two types of classification problem;

- Binary Classification: Prediction two classes example Spam or not spam, Cancer or not, Man or Woman, Rainy or Sunny et cetera
- MultiClassification: Predicting more than two classes, Example: Cat, dog or Human, Rainy, Sunny or Cloudy.

BINARY CLASSIFICATION

This type of classification involves predicting between two classes/Labels(spam or not spam). Typically, one class is taken as the positive class, and the other as the negative class. This depends on what the problem statement is.For example, Say we want to build a model that predicts who gets a promotion in a company or not, Here, the positive class if "Promotion(yes)" and Negative class is "Not Promoted(No)".
So a model is built that creates a decision boundary between these two classes(Promoted(the positive class) and Not promoted(the negative class)).

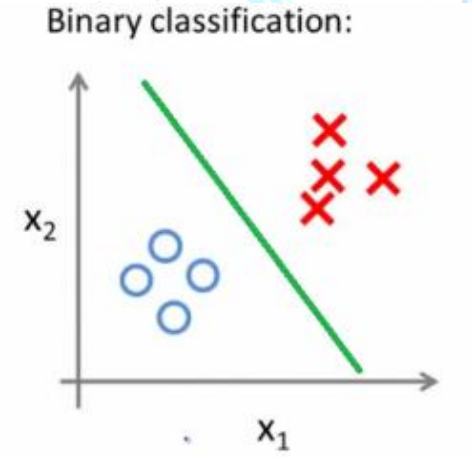


Image Source: Here

MULTICLASSIFICATION

MultiClassification involves more than two classes. e.g., classify a set of images of fruits into oranges, apples, or pears.
How many types of fruits do you see in the image below?How would you classify them?How many classes did you obtain? Two or more?
How many tribes do we have here/what are the different types of people present in the audience?

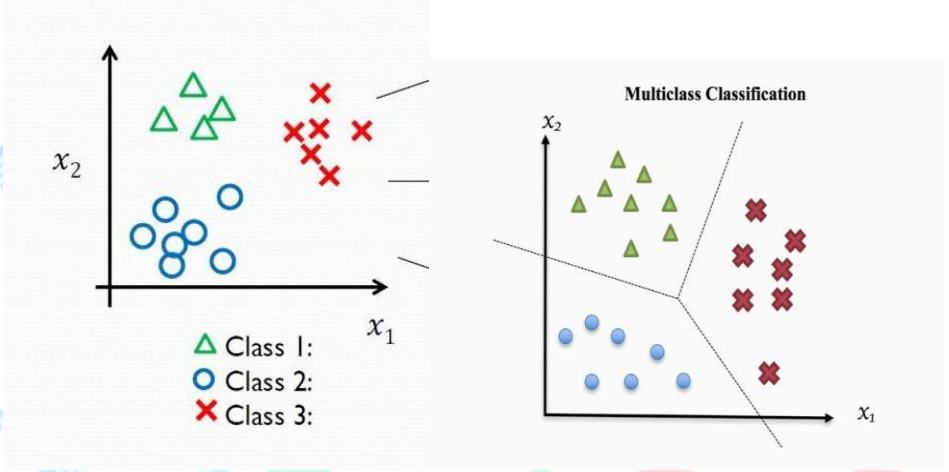


Image sources: Fruits Google

BINARY VERSUS MULTICLASS

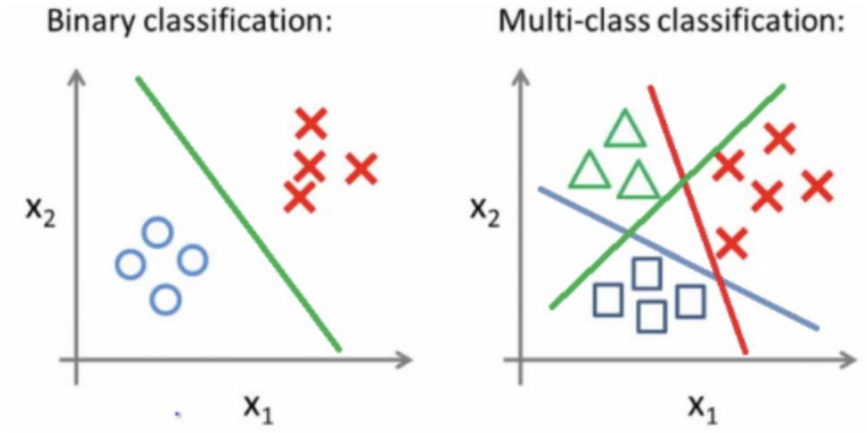


Image Source: Google

ALGORITHMS FOR CLASSIFICATION

- Logistic regression
- Decision Trees
- Support vector Machine
- Random Forest
- Naive Bayes
- K Nearest Neighbour

The Scikit-learn

Scikit-learn is a library in Python that provides many supervised learning and unsupervised algorithms. It's built upon some of the packages you already familiar with, like NumPy, Pandas, and Matplotlib!

The functionality that scikit-learn provides include:

- Regression
- Classification
- Clustering
- Model selection
- Preprocessing

Installation

The easiest way to install scikit-learn is using:

pip install -U scikit-learn

or

conda install -c conda-forge scikit-learn

N.b The Anaconda Package comes with this library pre-installed

Importing Scikit-learn Module

Some of the classification models that can be imported from sklearn library includes:

- Logistic Regression: from sklearn.linear_model import LogisticRegression
- K Nearest Neighbor: from sklearn.neighbors import KNeighborsClassifier
- Support Vector Machine: from sklearn.svm import SVC
- Decision Trees Classifier: from sklearn.tree import DecisionTreeClassifier
- Random Forest Classifier: from sklearn.ensemble import RandomForestClassifier
- Gradient Boost Classifier: from sklearn.ensemble import GradientBoostingClassifier

Activity : Building Classification Machine Learning Model for AXA Mansard Medical Insurance: Binary Classification

The implementation is in the Python programming language and the dataset used is the 'Insurance Dataset"

Problem statement
You work as an analyst in the marketing department of a company that provides various medical insurance in Nigeria. Your manager is unhappy with the low sales volume of a specific kind of insurance. The data engineer provides you with a sample dataset for those that visit the company website for medical insurance.

The dataset contains the following columns:

- User ID
- Gender
- Age
- Salary
- Purchase: An indicator of whether the users purchased (1/Positive Class) or not-purchased (0/Negative Class) a particular product.

As we must have guessed, this is a binary classification problem, with the positive class being "Purchased" and the negative class being "Not purchased".

In Machine learning, we have something called the "No Free Lunch Theorem". Which simply implies that we do not try one single algorithm for a given problem and decide it is the best. We have to choose a class of algorithms(Binary classifiers in b=this case), then train and predict on each of them, compare performance using the chosen metrics, and choose the best one. Note that one Algorithm is not always the best one across all problems.

For this problem, We plan to use the following classifiers to predict the classes 'Purchased' or 'not-purchased'.

- Logistic regression

- Random forest(Tree-based)
- Naive Bayes
- XGBoost(Tree-Based)
- Support Vector Machine

Import Python modules

We need to import some packages that will enable us to explore the data and build machine learning models

```
#Importing the required libraries
import numpy as np #for linear algebra/data preprocessing
import pandas as pd #for data preprocessing
import matplotlib.pyplot as plt #For visualization
import seaborn as sns #for visualization
```

Read the Data

```
#Reading the data as a  .csv file and checking the first 20 observations
insurance = pd.read_csv("../Data/Medical_insurance_dataset.csv")

insurance.head(20)
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	not-purchased
1	15810944	Male	35	20000	not-purchased
2	15668575	Female	26	43000	not-purchased
3	15603246	Female	27	57000	not-purchased
4	15804002	Male	19	76000	not-purchased
5	15728773	Male	27	58000	not-purchased

Dropping a Column/Feature not useful for prediction/Modeling

The User ID is a random number generated for every customer that comes to the company for medical insurance. Therefore, it is not useful in prediciting whether the person will buy medical insurance or not. It will hence be removed from the data, as it is not useful for modeling.

```
[ ]  #Dropping User ID column
     insurance.drop(["User ID"], axis= "columns", inplace= True)
```

Next, we transform or Encode the Target column 'Purchased' to discrete values 1 representing purchased and 0 representing "not-purchased".This will transform the output variable (label) to be numeric values, which is important for a machine learning model.

```
#Mapping the values 1 and to transform the target column
insurance["Purchased"] = insurance["Purchased"].apply(lambda x: 1 if x == "purchased" else 0)
```

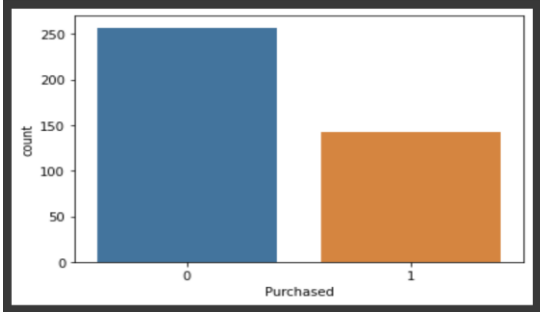
```
#Reading the data again
insurance.head(10)
```

	Gender	Age	EstimatedSalary	Purchased
0	Male	19	19000	0
1	Male	35	20000	0
2	Female	26	43000	0
3	Female	27	57000	0
4	Male	19	76000	0
5	Male	27	58000	0

Exploratory Data Analysis

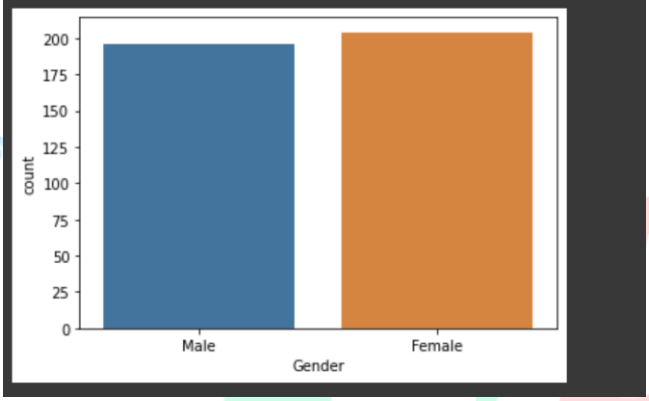
Fact generated by data exploratory will help us to know those features that can predict whether a person will purchase medical insurance or not. Let us start by visualizing the proportion of those that want to buy medical insurance or not.

```
sns.countplot(x = "Purchased", data = insurance);
```



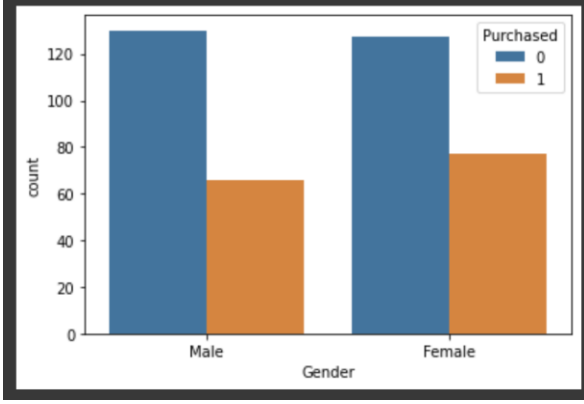
As you can see, the majority of those that visit the medical insurance company did not want to buy the insurance. This is an example of class imbalance. That is, there is no equal proportion of those that will buy or not.

```
sns.countplot(x = "Gender", data = insurance);
```



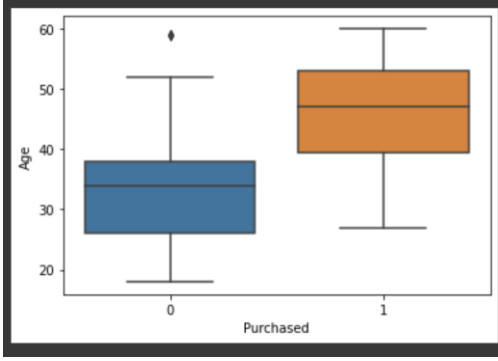
The proportion of males are almost the same as females.

```
sns.countplot(x = "Gender" , hue = "Purchased", data = insurance);
```



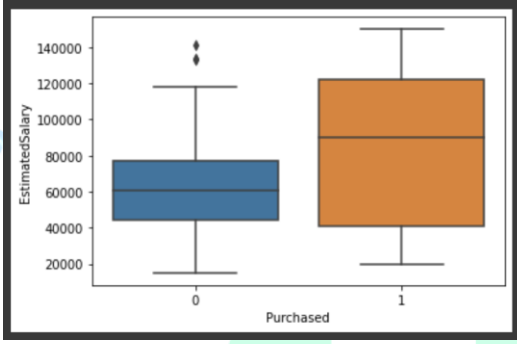
It seems that females wanted to purchase the insurance when compared with males.

```
sns.boxplot(x = "Purchased", y = "Age", data = insurance);
```



From the look of things, other people purchased insurance compared with younger people.

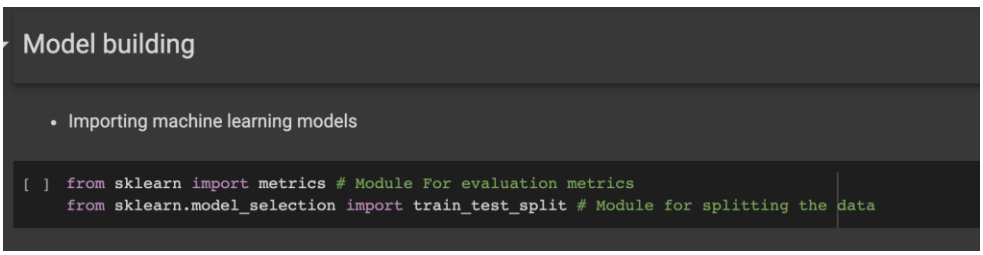
```
sns.boxplot(x = "Purchased", y = "EstimatedSalary", data = insurance);
```



People that earned higher salary purchased the insurance while those that earned low did not purchase the insurance. Of course, it is expected you purchase a medical insurance when you have money.

Model building

- Importing machine learning models



- **Preparing the Data for modeling(**Separating features and the label from the data)
  Now is the time to build machine learning models for the task of predicting whether the customers will buy medical insurance or not. Therefore, we shall separate the set of features (X) from the label (Y).

```
# split data into features and target

X = insurance.drop(["Purchased"], axis= "columns") # droping the label variable (Purchased) from the data

y = insurance["Purchased"]
```

```
[ ] X.head()
```

	Gender	Age	EstimatedSalary
0	Male	19	19000
1	Male	35	20000
2	Female	26	43000
3	Female	27	57000
4	Male	19	76000

- As discussed in Part 3, we need to create a one-hot encoding for all the categorical features in the data because some algorithms cannot work with categorical data directly. They require all input variables and output variables to be numeric. In this case, we will create a one-hot encoding for the gender feature by using *pd.get_dummies()*. As shown below:

```
[ ] pd.get_dummies(insurance["Gender"])
```

	Female	Male
0	0	1
1	0	1
2	1	0
3	1	0
4	0	1
...	...	...
395	1	0
396	0	1
397	1	0
398	0	1
399	1	0

400 rows × 2 columns

-

In fact, pd.get_dummies() is very powerful to actually locate the categorical features and create a one-hot encoding for them. For example:

```
pd.get_dummies(X)
```

	Age	EstimatedSalary	Gender_Female	Gender_Male
0	19	19000	0	1
1	35	20000	0	1
2	26	43000	1	0
3	27	57000	1	0
4	19	76000	0	1
...	...	...	...	...
395	46	41000	1	0
396	51	23000	0	1
397	50	20000	1	0
398	36	33000	0	1
399	49	36000	1	0

400 rows × 4 columns

We now save this result of one-hot encoding into X.

```
[ ]  X = pd.get_dummies(X)
```

Split the data into training and test set

As discussed in A, We will split our dataset (Features (X) and Label (Y)) into training and test data by using train_test_split() function from the sklearn. The training set will be 80% while the test set will be 20%. The random_state that is set to 1234 is for all of us to have the same set of data.

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state= 1234)
```

We now have the pair of training data `(X_train, y_train)` and test data `(X_test, y_test)`

Model training

We will use the training data to build the model and then use test data to make prediction and evaluation respectively.

1. Logistic regression:
   Let's train a Logistic regression model with our training data. We need to import the Logistic regression from the sklearn model

```
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
```

We now create an object of class LogisticRegression() to train the model on

```
#Instance for the logistiv regression model
logisticmodel = LogisticRegression()

logisticmodel.fit(X_train, y_train) #Fitting the train set
```

```
LogisticRegression()
```

l*ogisticmodel.fit* trained the Logistic regression model. The model is now ready to make prediction for the unknown label by using only the features from the test data (X_test).

```
logisticmodel.predict(X_test)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

Let's save the prediction result into logistic_prediction. This is what the model predicted for us.

```
#Making a prediction
logistic_prediction = logisticmodel.predict(X_test)
```

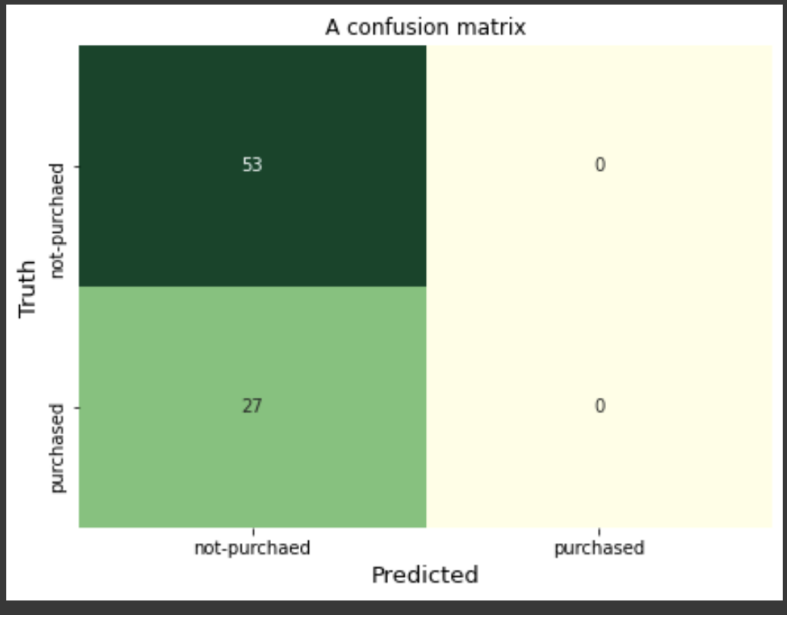
Model evaluation:

Since we know the true label in the test set (i.e. y_test), we can compare this prediction with it, hence evaluate the logistic model. I have created a function that will help you visualize a confusion matrix for the logistic model and you can call on it henceforth to check the performance of any model.

```
def ConfusionMatrix(ytest, ypred, label = ["Negative", "Positive"]):
    "A beautiful confusion matrix function to check the model performance"
    from sklearn.metrics import confusion_matrix
    import seaborn as sns
    cm = confusion_matrix(ytest, ypred)
    plt.figure(figsize=(7, 5))
    sns.heatmap(cm, annot = True, cbar = False, fmt = 'd', cmap = 'winter')
    plt.xlabel('Predicted', fontsize = 13)
    plt.xticks([0.5, 1.5], label)
    plt.yticks([0.5, 1.5], label)
    plt.ylabel('Truth', fontsize = 13)
    plt.title('A confusion matrix');
```

By using the ConfusionMatrix() function, we have:

```
ConfusionMatrix(y_test, logistic_prediction, label= ["not-purchaed", "purchased"])
```



Interpretation of the Logistics Regression model evaluation performance

There are 53 True Negatives (TN): predicting that the customer will not buy the insurance and truly the customer did not buy the insurance.

There are 27 False Negative (FN): predicting that the customer will not buy the insurance and the customer actually bought the insurance.

We can check the accuracy by using:

```
metrics.accuracy_score(y_test, logistic_prediction)
```

```
0.6625
```

The accuracy of the model is 66.25% . We cannot trust this accuracy since the data is class imbalanced. Therefore, we are going to use F1 score instead.

```
metrics.f1_score(y_test, logistic_prediction)
```

```
0.0
```

Naive Bayes model

Let's train a Naive Bayes model with our training data. We need to import the Naive Model from the sklearn model;

```
from sklearn.naive_bayes import GaussianNB

naivemodel = GaussianNB()

naivemodel.fit(X_train, y_train)
```

```
GaussianNB()
```

naivemodel.fit() trained the Naive Bayes model. The model is now ready to make prediction for the unknown label by using only the features from the test data (X_test).

```
naivemodel_prediction = naivemodel.predict(X_test)
```

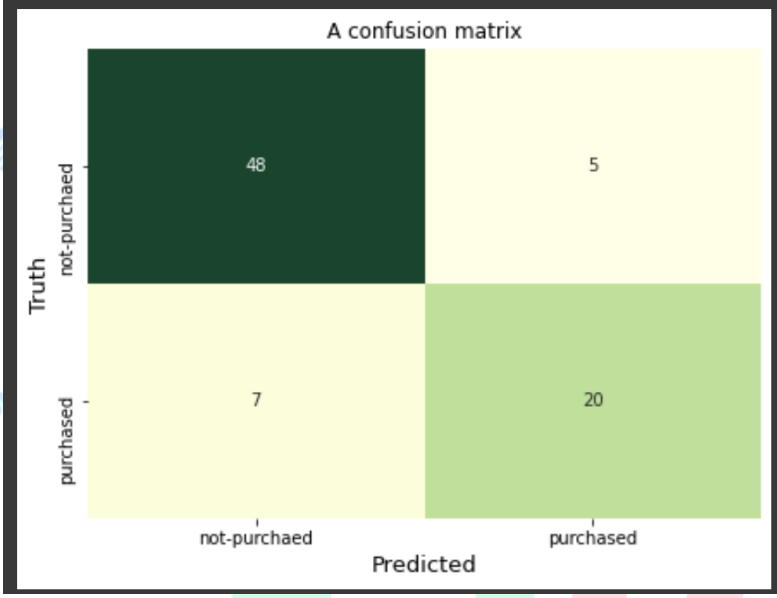
You can call one naivemodel_predictionto see the prediction

```
naivemodel_prediction
```

```
array([0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], dtype=int64)
```

By using the ConfusionMatrix() function, we can see how the model performed:

```
ConfusionMatrix(y_test, naivemodel_prediction, label= ["not-purchaed", "purchased"])
```



Interpretation of the Naive Bayes model evaluation performance

- There are 48 True Negatives (TN): predicting that the customer will not buy the insurance and truly the customer did not buy the insurance.

- There are 20 True Positives (TP): predicting that the customer will buy the insurance and truly the customer did buy the insurance.

- There are 7 False Negatives (FN): predicting that the customer will not buy the insurance and the customer actually bought the insurance.

- There are 5 False Positives (FN): predicting that the customer will buy the insurance and the customer did not buy the insurance.

Evaluation metrics

We are going to check the accuracy and F1 score of the models.

We can check the accuracy by using:

```
metrics.accuracy_score(y_test, naivemodel_prediction)
0.85
```

The accuracy of the model is 85%

We can check the F1 score by using:

```
metrics.f1_score(y_test, naivemodel_prediction)
0.7692307692307692
```

The F1 score of the model is 76.9%

As you can see, this model seems good in predicting whether a patient will buy insurance or not.

Random Forest Model

Let's train a Random Forest model with our training data. We need to import the Random Forest model from the sklearn module

```
from sklearn.ensemble import RandomForestClassifier

randomforestmodel = RandomForestClassifier()

randomforestmodel.fit(X_train, y_train)
RandomForestClassifier()
```

randomforestmodel.fit() trained the Random Forest model on the training data. The model is now ready to make predictions for the unknown label by using only the features from the test data (X_test).

```
randomforestmodel_prediction = randomforestmodel.predict(X_test)
```

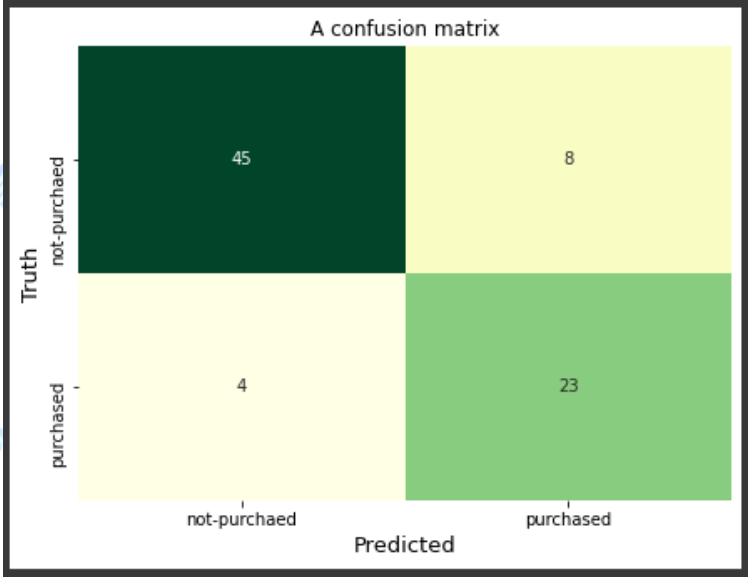
You can call the *randomforestmodel_prediction* method to see the prediction.

```
randomforestmodel_prediction
```

```
array([0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], dtype=int64)
```

By using the ConfusionMatrix() function, we can see how the model performed:

```
ConfusionMatrix(y_test, randomforestmodel_prediction, label= ["not-purchaed", "purchased"])
```



Interpretation of the Random Forest model evaluation performance

- There are 44 True Negatives (TN): predicting that the customer will not buy the insurance and truly the customer did not buy the insurance.

- There are 23 True Positives (TP): predicting that the customer will buy the insurance and truly the customer did buy the insurance.

- There are 4 False Negatives (FN): predicting that the customer will not buy the insurance and the customer actually bought the insurance.

- There are 9 False Positives (FN): predicting that the customer will buy the insurance and the customer did not buy the insurance.

Evaluation metrics

We are going to check the accuracy and F1 score of the model.

We can check the accuracy by using:

```
metrics.accuracy_score(y_test, randomforestmodel_prediction)
```

```
0.85
```

The accuracy of the model is 83.75%

We can check the F1 score by using:

```
metrics.f1_score(y_test, randomforestmodel_prediction)
```

```
0.7931034482758621
```

The F1 score of the model is 77.97%

As you can see, this model seems good in predicting whether a patient will buy insurance or not.

Extreme Gradient Boost (XGBoost) Model

Let's train an XGBoost model with our training data. We need to import the XGBoost model from the sklearn module but before we do that, we need to install the module because it is not available in the sklearn.

How to install XGBoost

Go to your terminal and type pip install xgboost

pip install xgboost

After installation, you can now import it as follows:

```
from xgboost import XGBClassifier

xgboostmodel = XGBClassifier(use_label_encoder=False)

xgbboostmodel = xgboostmodel.fit(X_train, y_train)
```

```
[19:13:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061:
```

xgboostmodel.fit() trained the XGBoost model on the training data. The model is now ready to make prediction for the unknown label by using only the features from the test data (X_test).

```
xgbboostmodel_prediction = xgboostmodel.predict(X_test)
```

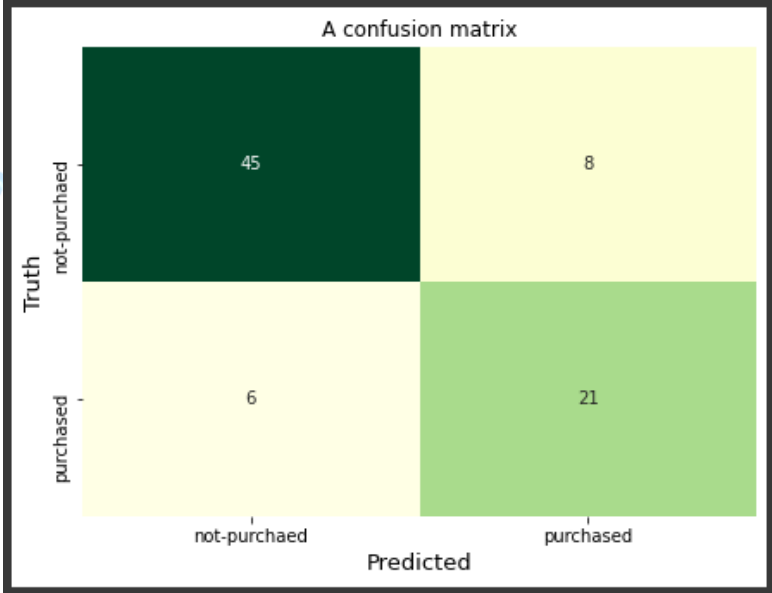
You can call on xgbboostmodel_prediction to see the prediction.

```
xgbboostmodel_prediction
```

```
array([0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0])
```

By using the ConfusionMatrix() function, we can see how the model performed:

```
ConfusionMatrix(y_test, xgbboostmodel_prediction, label= ["not-purchaed", "purchased"])
```



Interpretation of the XGBoost model evaluation performance

- There are 45 True Negatives (TN): predicting that the customer will not buy the insurance and truly the customer did not buy the insurance.

- There are 21 True Positives (TP): predicting that the customer will buy the insurance and truly the customer did buy the insurance.

- There are 6 False Negatives (FN): predicting that the customer will not buy the insurance and the customer actually bought the insurance.

- There are 8 False Positives (FN): predicting that the customer will buy the insurance and the customer did not buy the insurance.

Evaluation metrics

We are going to check the accuracy and F1 score of the model.

We can check the accuracy by using:

```
metrics.accuracy_score(y_test, xgbboostmodel_prediction)
```

The accuracy of the model is 82.5%

We can check the F1 score by using:

```
metrics.f1_score(y_test, xgbboostmodel_prediction)
```

The F1 score of the model is 75%

As you can see, this model seems good in predicting whether a patient will buy insurance or not.

Support Vector Machine (SVM)

Let's train a Support Vector Machine model with our training data. We need to import the Support Vector Machine model from the sklearn module

```
from sklearn.svm import SVC

SVMmodel = SVC()

SVMmodel.fit(X_train, y_train)
```

SVMmodel.fit() trained the Support Vector Machine on the training data. The model is now ready to make predictions for the unknown label by using only the features from the test data (X_test).

```
SVMmodel_prediction = SVMmodel.predict(X_test)
```

You can call on SVMmodel_prediction to see what has been predicted.

```
SVMmodel_prediction
```

By using the ConfusionMatrix() function, we can see how the model performed:

```
ConfusionMatrix(y_test, SVMmodel_prediction, label= ["not-purchaed", "purchased"])
```

Interpretation of the Support Vector model evaluation performance

- There are 50 True Negatives (TN): predicting that the customer will not buy the insurance and truly the customer did not buy the insurance.
- There are 14 True Positives (TP): predicting that the customer will buy the insurance and truly the customer did buy the insurance.
- There are 13 False Negatives (FN): predicting that the customer will not buy the insurance and the customer actually bought the insurance.
- There are 3 False Positives (FN): predicting that the customer will buy the insurance and the customer did not buy the insurance.

Evaluation metrics

We are going to check the accuracy and F1 score of the model.

We can check the accuracy by using:

```
metrics.accuracy_score(y_test, SVMmodel_prediction)
```

The accuracy of the model is 80%

We can check the F1 score by using:

```
metrics.f1_score(y_test, SVMmodel_prediction)
```

The F1 score of the model is 63.6%

As you can see, this model seems good in predicting whether a patient will buy insurance or not.

```
## Models Summary

+---------------+----------+------------+
| Model (s)     | Accuracy | F1-score   |
+===============+==========+============+
| Logistic regression | 66.25 | 0 |
+---------------+----------+------------+
| Naive Bayes   | 85       | 76.92      |
+---------------+----------+------------+
| Random Forest | 83.75    | 77.97      |
+---------------+----------+------------+
| XGBoost       | 82.5     | 75         |
+---------------+----------+------------+
| SVM           | 80       | 63.63      |
+---------------+----------+------------+
```

Having train all the five (5) models, we can see that the best model that can accurately predict whether a customer will buy the insurance or not is the Random Forest Model.

Class Activities

- Importing Scikit-learn Module
- Use the following models to predict whether a customer will buy insurance or not. Your teacher has also included how to import those models for you.

- K Nearest Neighbor: from sklearn.neighbors import KNeighborsClassifier

- Decision Trees Classifier: from sklearn.tree import DecisionTreeClassifier

- Gradient Boost Classifier: from sklearn.ensemble import GradientBoostingClassifier

Which of the three (3) models is the best in term of the F1 score?

Further reading:

Classification and Regression Modelling

Regression or Classification: How do I know what to use?

Binary Classification and MultiClassification

Multiclass ScikitLearn

Multiclass

RESOURCES

Datacamp Tutorial

ScikitLearn official Tutorial

Machine Learning with ScikitLearn Github tutorial

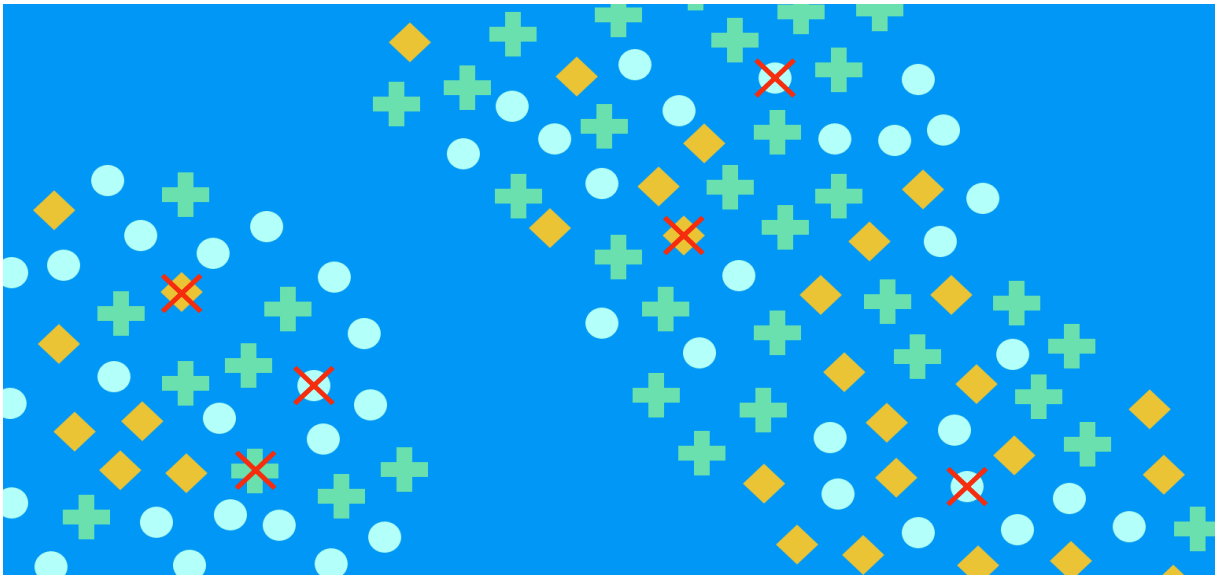
Day 4

Unsupervised Machine Learning

Content:

1. **What is clustering**
2. **Use of clustering algorithms**
3. **K-mean clustering**
4. **Hierarchical clustering**
5. **Introduction to neural networks and Deep learning**

What is clustering



Clustering or cluster analysis is a machine learning technique, which groups the unlabeled dataset.
"A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."

Use of clustering algorithms

There are many clustering algorithms to choose from and no single best clustering algorithm for all cases. Instead, it is a good idea to explore a range of clustering algorithms and different configurations for each algorithm. Many algorithms use similarity or distance measures between examples in the feature space in an effort to discover dense regions of observations. As such, it is often good practice to scale data prior to using clustering algorithms.
A list of 10 of the more popular algorithms is as follows:

- Affinity Propagation
- Agglomerative Clustering
- BIRCH
- DBSCAN

- K-Means
- Mini-Batch K-Means
- Mean Shift
- OPTICS
- Spectral Clustering
- Mixture of Gaussians

Each algorithm offers a different approach to the challenge of discovering natural groups in data.

When you should use clustering

- When you're starting from a large, unstructured dataset
- When you don't know how many or which classes your data is divided into
- When manually dividing and annotating your data is too resource-intensive
- When you're looking for anomalies in your data

Applications of Clustering in different fields

- **Marketing:** It can be used to characterize & discover customer segments for marketing purposes.
- **Biology:** It can be used for classification among different species of plants and animals.
- **Libraries:** It is used in clustering different books on the basis of topics and information.
- **Insurance:** It is used to acknowledge the customers, their policies and identifying the frauds.
- **City Planning:** It is used to make groups of houses and to study their values based on their geographical locations and other factors present.
- **Earthquake studies:** By learning the earthquake-affected areas we can determine the dangerous zones.

K-mean Clustering

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms.
A centroid is the imaginary or real location representing the center of the cluster.
Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares.
In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.
The 'means' in the K-means refers to averaging of the data; that is, finding the centroid.

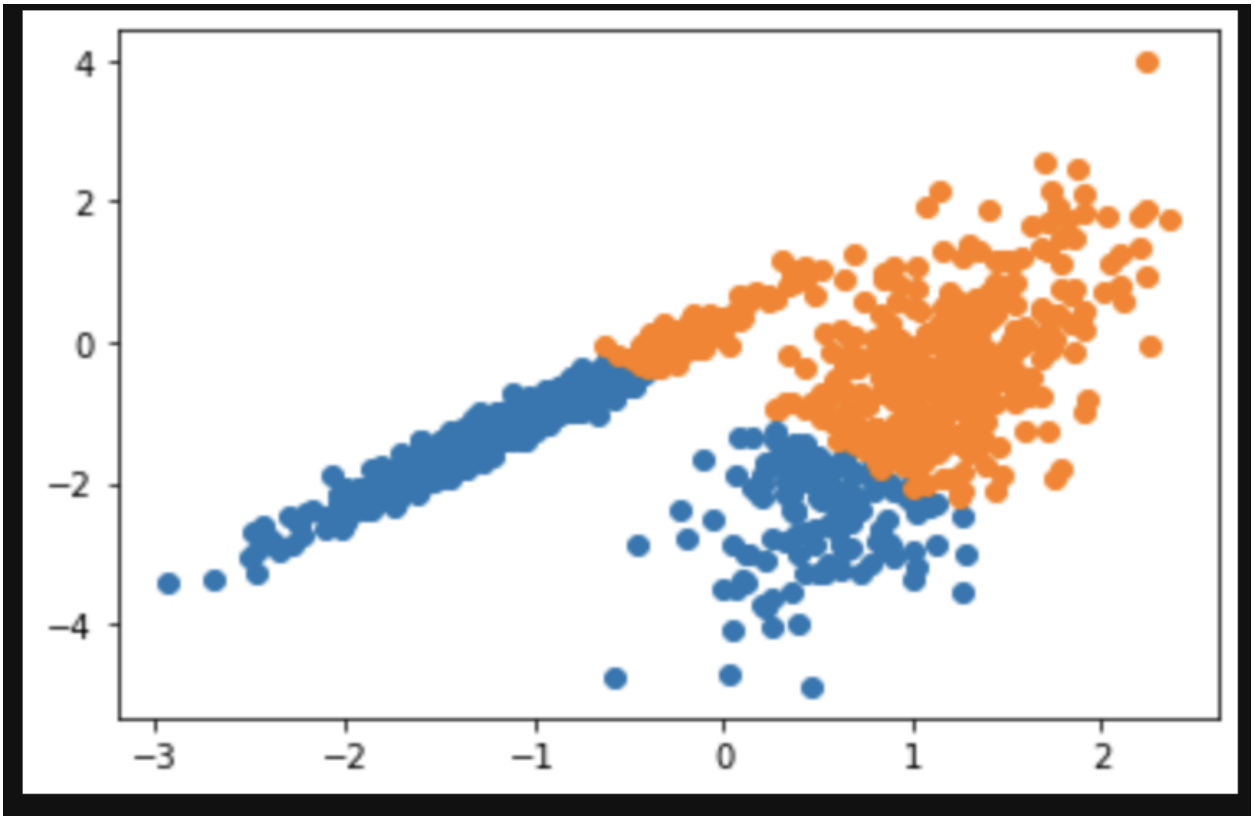
How the K-means algorithm works

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids
It halts creating and optimizing clusters when either:
The centroids have stabilized — there is no change in their values because the clustering has been successful.

K-means algorithm example problem

```
[1]: # k-means clustering
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
from matplotlib import pyplot
# define dataset
X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=4)
# define the model
model = KMeans(n_clusters=2)
# fit the model
model.fit(X)
# assign a cluster to each example
yhat = model.predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
    # get row indexes for samples with this cluster
    row_ix = where(yhat == cluster)
    # create scatter of these samples
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```

Running the example fits the model on the training dataset and predicts a cluster for each example in the dataset. A scatter plot is then created with points colored by their assigned cluster.



In this case, a reasonable grouping is found, although the unequal equal variance in each dimension makes the method less suited to this dataset.

Hierarchical Clustering Algorithm

is an unsupervised clustering algorithm which involves creating clusters that have predominant ordering from top to bottom.

The algorithm groups similar objects into groups called ***clusters***. The endpoint is a set of clusters or groups, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other.

This clustering technique is divided into two types:

- **Agglomerative Hierarchical Clustering**: It's a "bottom-up" Approach: *each observation starts in its own cluster, and pairs of clusters are merged as one moves up the* ***hierarchy.***

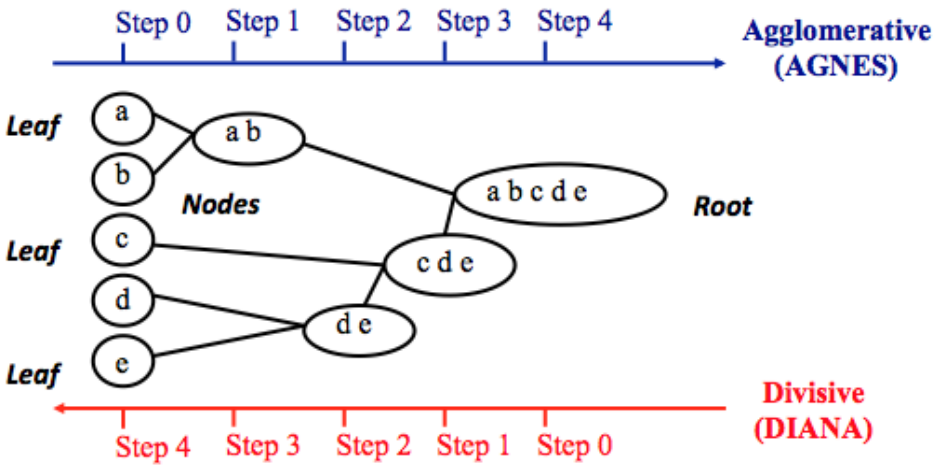
  How does it work?

  Make each data point a single-point cluster → forms N clusters

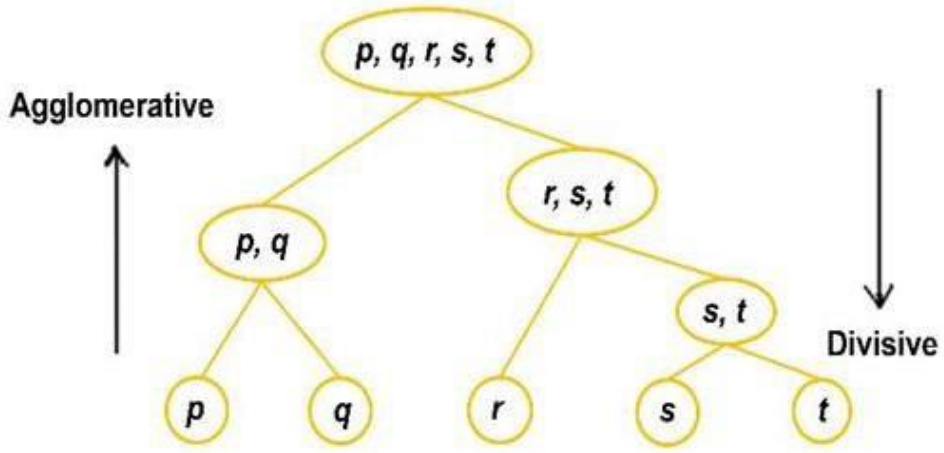
  Take the two closest data points and make them one cluster → forms N-1 clusters

  Take the two closest clusters and make them one cluster → Forms N-2 clusters.

  Repeat step-3 until you are left with only one cluster.



- Divisive Hierarchical Clustering: n Divisive or DIANA (Divisive Analysis Clustering) is a top-down clustering method where we assign all of the observations to a single cluster and then partition the cluster to two least similar clusters. Finally, we proceed recursively on each cluster until there is one cluster for each observation. So, this clustering approach is exactly opposite to Agglomerative clustering.



Introduction to neural networks

Neural networks and deep learning are big topics in Computer Science and in the technology industry, they currently provide the best solutions to many problems in image recognition, speech recognition and natural language processing.

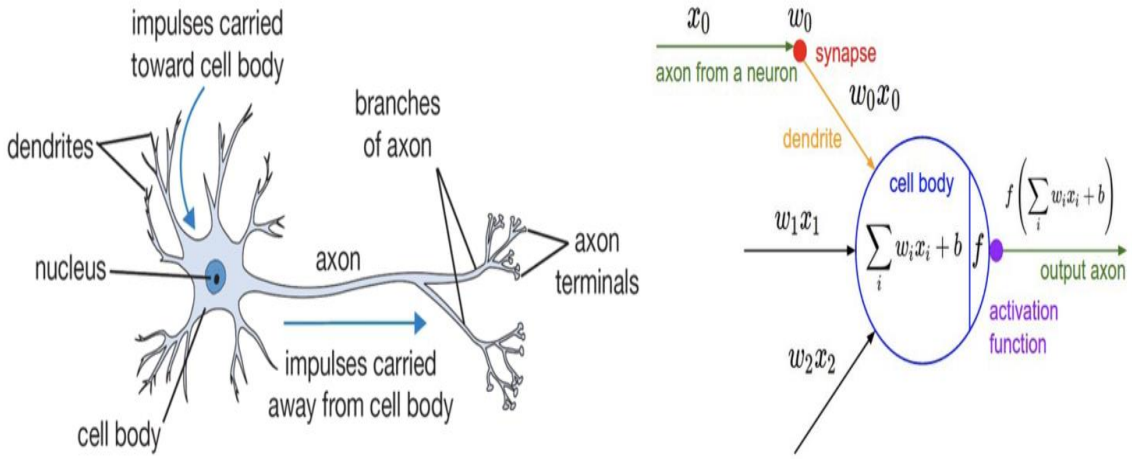
The definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as:

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

Biological motivation and connections

The basic computational unit of the brain is a **neuron**. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately 10^{14} — 10^{15} **synapses**. The

diagram below shows a cartoon drawing of a biological neuron (left) and a common mathematical model (right).



biological neuron (left) and a common mathematical model (right)

Neural Network Architecture

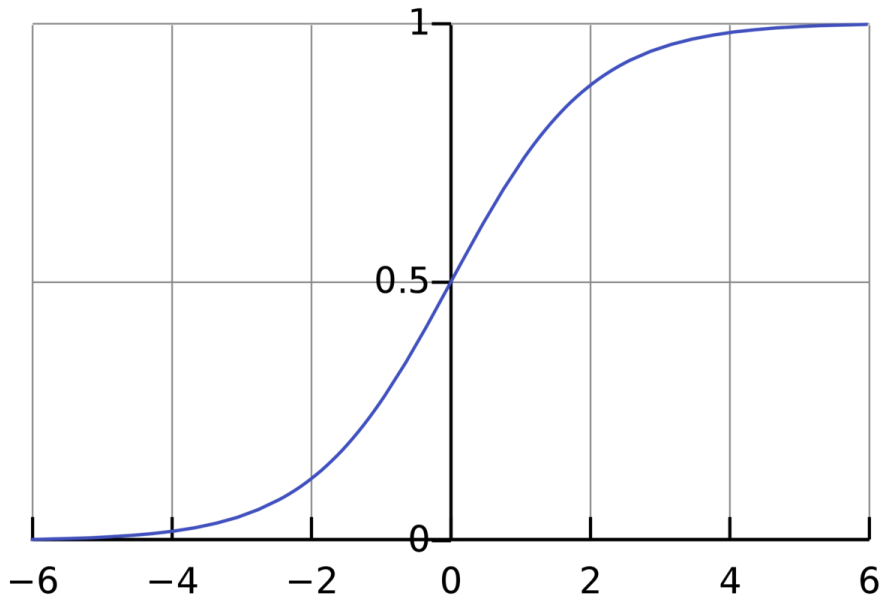
- **Input Nodes (input layer):** No computation is done here within this layer, they just pass the information to the next layer (hidden layer most of the time). A block of nodes is also called **layer**.
- **Hidden nodes (hidden layer):** In Hidden layers is where intermediate processing or computation is done, they perform computations and then transfer the weights (signals or information) from the input layer to the following layer (another hidden layer or to the output layer). It is possible to have a neural network without a hidden layer and I'll come later to explain this.
- **Output Nodes (output layer):** Here we finally use an activation function that maps to the desired output format (e.g. softmax for classification).
- **Connections and weights:** The *network* consists of connections, each connection transferring the output of a neuron i to the input of a neuron *j*. In this sense *i* is the predecessor of *j* and *j* is the successor of *i*, Each connection is assigned a weight *Wij.*
- **Activation function:** the **activation function** of a node defines the output of that node given an input or set of inputs. A standard computer chip circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input. This is similar to the behavior of the linear perceptron in neural networks. However, it is the *nonlinear* activation function that allows such networks to compute nontrivial problems using only a small number of nodes. In artificial neural networks this function is also called the transfer function.

Commonly used activation functions

Every activation function (or *non-linearity*) takes a single number and performs a certain fixed mathematical operation on it. Here are some activations functions you will often find in practice:

- **Sigmoid**

- **Tanh**
- **ReLU**
- **Leaky ReLU**



- **Learning rule:** The *learning rule* is a rule or an algorithm which modifies the parameters of the neural network, in order for a given input to the network to produce a favored output. This *learning* process typically amounts to modifying the weights and thresholds.

Types of Neural Networks

The three most important types of neural networks are: Artificial Neural Networks (ANN); Convolution Neural Networks (CNN), and Recurrent Neural Networks (RNN).

Activity for Clustering: Customer Segmentation

USE CASE: Customer Segmentation based on Annual income The dataset is a very simple data just to demonstrate with code how k-means works. Find below the link to the dataset and the code used uploaded to the Github repository. **REQUIRED PYTHON LIBRARIES**

- Numpy
- Pandas
- Matlplotlib
- ScikitLearn

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

Separating the data into X and Y.
Not Dependent and Independent as in Supervised. Remember that Clustering has no labels. Y here represents what we want to cluster with i.e our reference point

Y here is the annual income as that defines the objective of the analysis. For X, the 'CustomerID' column will be omitted because it plays no role here.

```
#Importing the dataset
df = pd.read_csv('Mall_Customers.csv')
print(df.shape)
df.head()
```

```
(200, 5)
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
X = df.iloc[:, [2,3,4]]
y = df.iloc[:, 3]
```

```
X
```

	Age	Annual Income (k$)	Spending Score (1-100)
0	19	15	39
1	21	15	81
2	20	16	6
3	23	16	77
4	31	17	40
...	...	...	...
195	35	120	79
196	45	126	28
197	32	126	74
198	32	137	18
199	30	137	83

200 rows × 3 columns

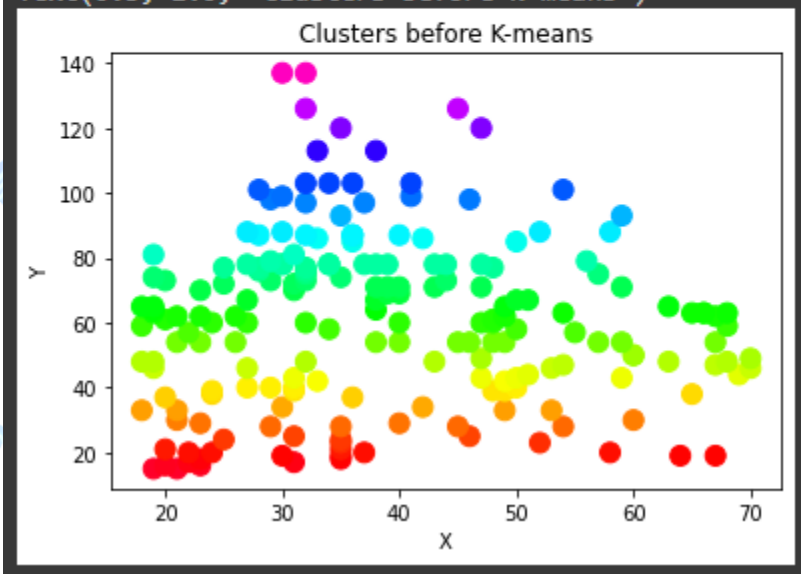
```
y
```

```
0       15
1       15
2       16
3       16
4       17
      ...
195    120
196    126
197    126
198    137
199    137
Name: Annual Income (k$), Length: 200, dtype: int64
```

Visualize data points before Clustering.

```
#Visualize data before K-means is applied
plt.figure(figsize=(6,4))
plt.scatter(X.iloc[:,0],X.iloc[:,1], c=y, s=100, cmap='gist_rainbow')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Clusters before K-means')
```

```
Text(0.5, 1.0, 'Clusters before K-means')
```

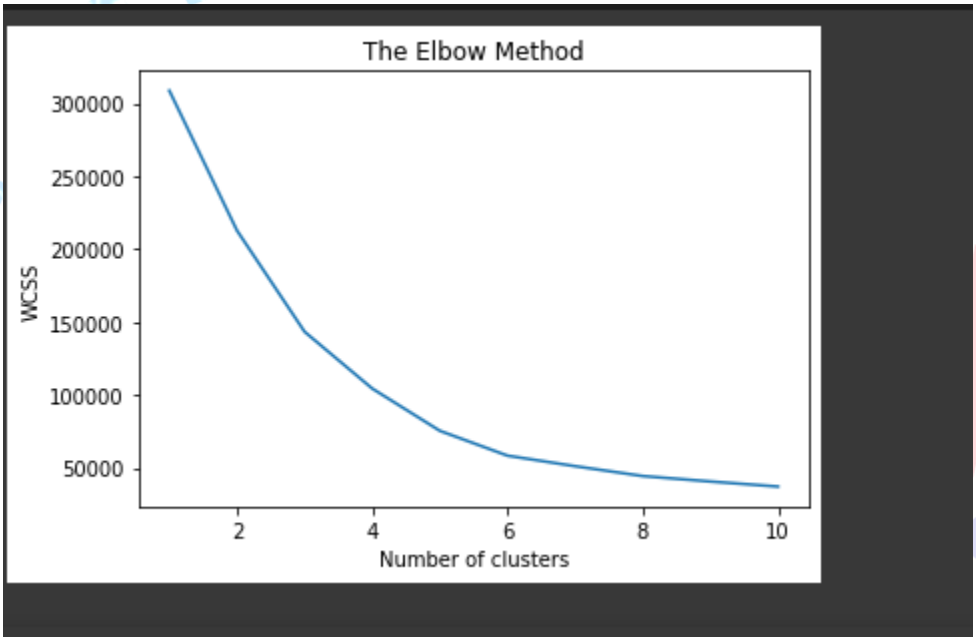


Split Dataset into Train and test set for model development.

```
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)
```

Find K using the Elbow method, and plot results using Matplotlib.

```
# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
#Initiate Wcss using empty list
wcss = []
#initiate iteration process using a for loop
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    #Append results to wcss list
    wcss.append(kmeans.inertia_)
#Plot results
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



As can be seen, the optimal value of k is 5.
Now, Print Labels predicted by the model.

```
# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

```
#Centers identified and calculates
centers = kmeans.cluster_centers_
centers
```

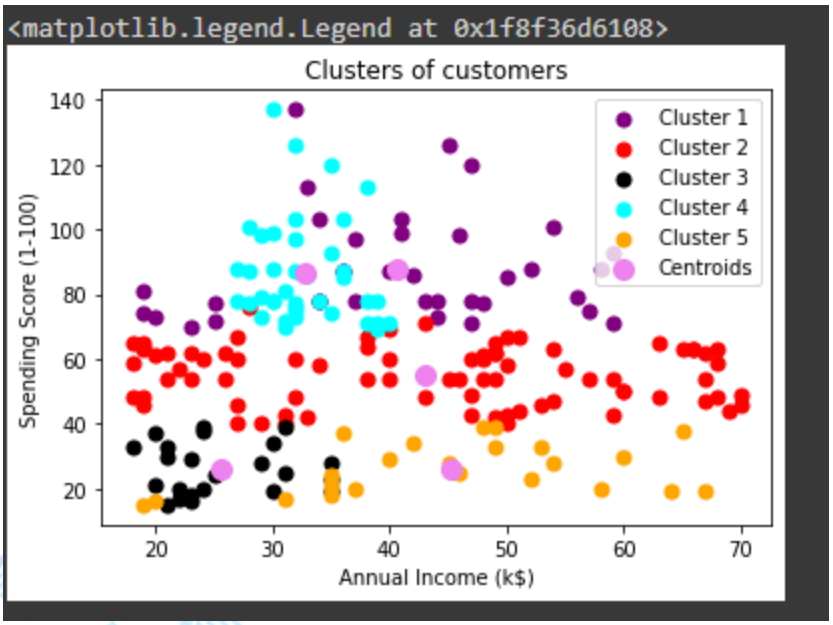
```
array([[40.66666667, 87.75      , 17.58333333],
       [43.08860759, 55.29113924, 49.56962025],
       [25.52173913, 26.30434783, 78.56521739],
       [32.69230769, 86.53846154, 82.12820513],
       [45.2173913 , 26.30434783, 20.91304348]])
```

```
kmeans.labels_
```

```
array([4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,
       4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,
       4, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 0, 3, 1, 3, 0, 3, 0, 3,
       0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 1, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3,
       0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3,
       0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3,
       0, 3], dtype=int32)
```

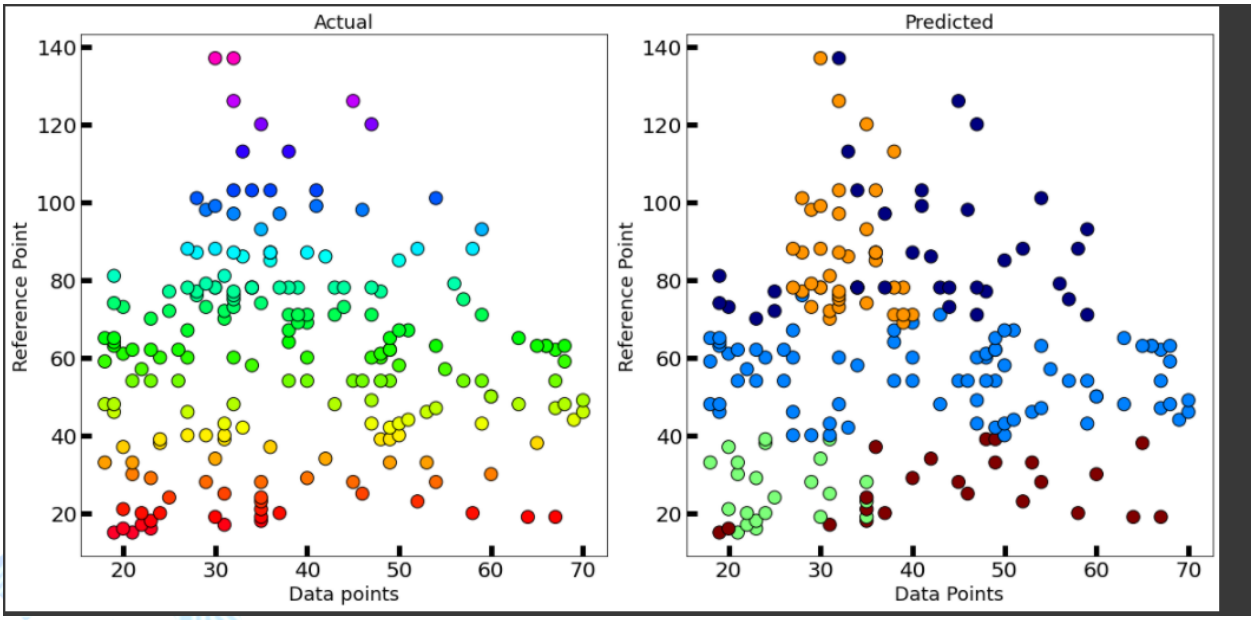
Visualize Clusters by K-means

```
#Convert X to an np array
X_1=np.array(X)
# Visualising the clusters
#1st CLuster
plt.scatter(X_1[y_kmeans == 0, 0], X_1[y_kmeans == 0, 1], s = 50, c = 'purple', label = 'Cluster 1')
#2nd Cluster
plt.scatter(X_1[y_kmeans == 1, 0], X_1[y_kmeans == 1, 1], s = 50, c = 'red', label = 'Cluster 2')
#3rd Cluster
plt.scatter(X_1[y_kmeans == 2, 0], X_1[y_kmeans == 2, 1], s = 50, c = 'black', label = 'Cluster 3')
#4th Cluster
plt.scatter(X_1[y_kmeans == 3, 0], X_1[y_kmeans == 3, 1], s = 50, c = 'cyan', label = 'Cluster 4')
#5th Cluster
plt.scatter(X_1[y_kmeans == 4, 0], X_1[y_kmeans == 4, 1], s = 50, c = 'orange', label = 'Cluster 5')
#Number of centriods against lables
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 100, c = 'violet', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
```



Visualize actual clusters versus Predicted Clusters for comparison.

```
#Visualizing Clusters after k-means
new_labels = kmeans.labels_
#Visualize data with reference to before and after k-means clustering
#Actual vs predicted plot
fig, axes = plt.subplots(1, 2, figsize=(16,8))
axes[0].scatter(X.iloc[:, 0], X.iloc[:, 1], c=y, cmap='gist_rainbow',
edgecolor='k', s=150)
axes[1].scatter(X.iloc[:, 0], X.iloc[:, 1], c=new_labels, cmap='jet',
edgecolor='k', s=150)
axes[0].set_xlabel('Data points', fontsize=18)
axes[0].set_ylabel('Reference Point', fontsize=18)
axes[1].set_xlabel('Data Points', fontsize=18)
axes[1].set_ylabel('Reference Point', fontsize=18)
axes[0].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
axes[1].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
axes[0].set_title('Actual', fontsize=18)
axes[1].set_title('Predicted', fontsize=18)
plt.tight_layout()
```



Bonus on How to Plot Clusters using the Yellowbrick Library

As a bonus tip, Let's use a special library which makes it easier with fewer lines of code to calculate and visualize the Elbow method plot and optimal k-value for the model. This library is called the ' YelloBrickVisualizer'. It works with the Matplolib and ScikitLearn as major wrappers and works best with ScikitLearn version 0.20 or later and Matplotlib version 3.0.1 or later. Take a look at the plot for the Elbow method and K-value using this Library.

Install Yellowbrick using

https://anaconda.org/DistrictDataLabs/yellowbrick

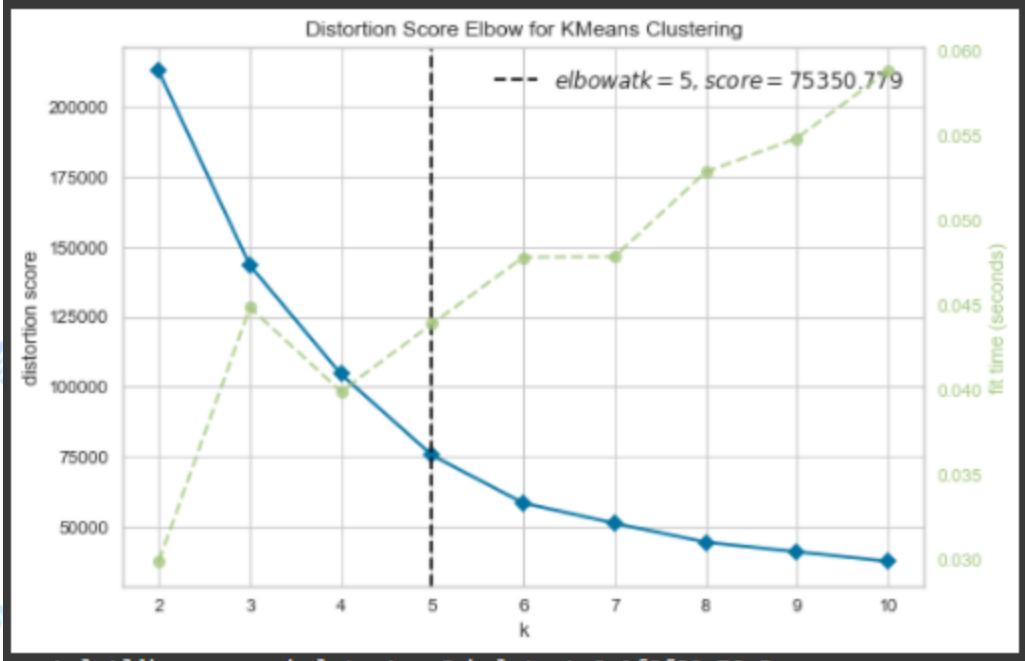
https://pypi.org/project/yellowbrick/

```
#import Yellowbrick
from yellowbrick.cluster import KElbowVisualizer
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144:
  warnings.warn(message, FutureWarning)
```

```
model = KMeans()
visualizer = KElbowVisualizer(model, k=(2,11))

visualizer.fit(X)
visualizer.show()
```




```
<matplotlib.axes._subplots.AxesSubplot at 0x1f8f39c72c8>
```

Day 5

- Revision and Feedback
- End to End machine learning model development- in class.
    - You will develop a machine learning model from scratch in class
- Hackathon Readiness: Useful Hackathon Links

1. Video on model development and making Kaggle submission:
   Watch on Youtube: https://youtu.be/13pIdHtnJA0
   Download: http://bit.ly/AICities_walkthrough

2. Exploratory Data Analysis, Feature Engineering and Modelling using Supermarket Sales Data. Part 1. By Rising Odegua https://towardsdatascience.com/exploratory-data-analysis-feature-engineering-and-modelling-using-supermarket-sales-data-part-1-228140f89298

3. A Practical Guide to Feature Engineering in Python By Rising Odegua https://heartbeat.fritz.ai/a-practical-guide-to-feature-engineering-in-python-8326e40747c8

4. train_test_split Vs StratifiedShuffleSplit - Brain John By Brain John https://medium.com/@411.codebrain/train-test-split-vs-stratifiedshufflesplit-374c3dbdcc36

5. How to Enter Your First Kaggle Competition https://towardsdatascience.com/how-to-enter-your-first-kaggle-competition-4717e7b232db

6. How to Enter a Simple Kaggle Competition https://towardsdatascience.com/how-to-enter-a-simple-kaggle-competition-9705faf3a1b7

7. Introduction to Exploratory Data Analysis (EDA) - Code Heroku https://medium.com/code-heroku/introduction-to-exploratory-data-analysis-eda-c0257f888676

8. Data Preprocessing for Machine Learning - Data-Driven Investor https://medium.com/datadriveninvestor/data-preprocessing-for-machine-learning-188e9eef1d2c

9. Processing Data To Improve Machine Learning Models Accuracy https://medium.com/fintechexplained/processing-data-to-improve-machine-learning-models-accuracy-de17c655dc8e

10. How To Develop a Machine Learning Model From Scratch https://towardsdatascience.com/machine-learning-general-process-8f1b510bd8af

11. All Machine Learning Models Explained in 6 Minutes

https://towardsdatascience.com/all-machine-learning-models-explained-in-6-minutes-9fe30ff6776a

12. 7 Ways to Improve your Predictive Models - Rants on Machine Learning
https://medium.com/rants-on-machine-learning/7-ways-to-improve-your-predictive-models-753705eba3d6

13. 3 ways to improve your Machine Learning results without more data
https://towardsdatascience.com/3-ways-to-improve-your-machine-learning-results-without-more-data-f2f0fe78976e

Finally, integrity is a core value at Data Science Nigeria, as such, we expect all participants to maintain honesty throughout the period of this competition- submission files, notebooks and code solutions should not be exchanged among participants. Multiple accounts and other dishonest acts are extremely frowned upon and would immediately lead to a forfeit of certification rights.

Remember, everyone is a winner!