MET’S, Bhujbal Knowledge City, Institute of Engineering
Department of Information Technology

Class: TE Year 2022-2023

Subject: Data Science and Big Data Analytics Lab

314457: DS & BDA Lab

Credit Scheme: 01 credit **Exam scheme: PR –** 25marks**, TW –** 25 marks

PREREQUISITES:

1. Discrete mathematics
2. Database Management Systems, Data warehousing, Data mining
3. Programming in Python

COURSE OBJECTIVES:

1. To understand Big data primitives and fundamentals.
2. To understand the different Big data processing techniques.
3. To understand and apply the Analytical concept of Big data using Python.
4. To understand different data visualization techniques for Big Data.
5. To understand the application and impact of Big Data.
6. To understand emerging trends in Big data analytics

COURSE OUTCOMES:

On completion of the course, students will be able to–
CO1: Apply Big data primitives and fundamentals for application development.
CO2: Explore different Big data processing techniques with use cases.
CO3: Apply the Analytical concept of Big data using Python.
CO4: Visualize the Big Data using Tableau.
CO5: Design algorithms and techniques for Big data analytics.
CO6: Design and develop Big data analytic application for emerging trends.

Group A: Assignments based on the Hadoop

Assignment 1:

TITLE: Hadoop Installation on Single Node

OBJECTIVE:

1. To Learn and understand the Big data primitives and fundamentals.
2. To learn and understand the Hadoop framework for Big Data
3. To understand and practice installation and configuration of Hadoop.

SOFTWARE REQUIREMENTS:

1 Ubuntu stable version
2 Java

THEORY:

Introduction

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

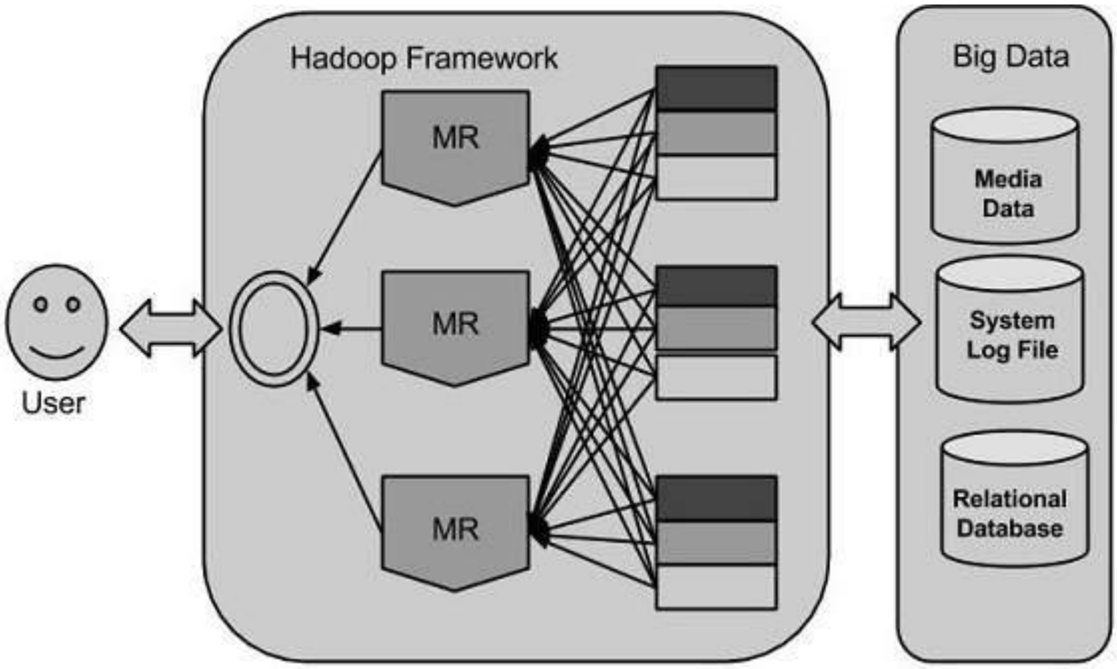
Big Data

Big data means really a big data, it is a collection of large datasets that cannot be processed using traditional computing techniques. Big data is not merely a data, rather it has become a complete subject, which involves various tools, technqiues and frameworks. Big data involves the data produced by different devices and applications. Given below are some of the fields that come under the umbrella of Big Data.

Hadoop

Hadoop is an Apache open-source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

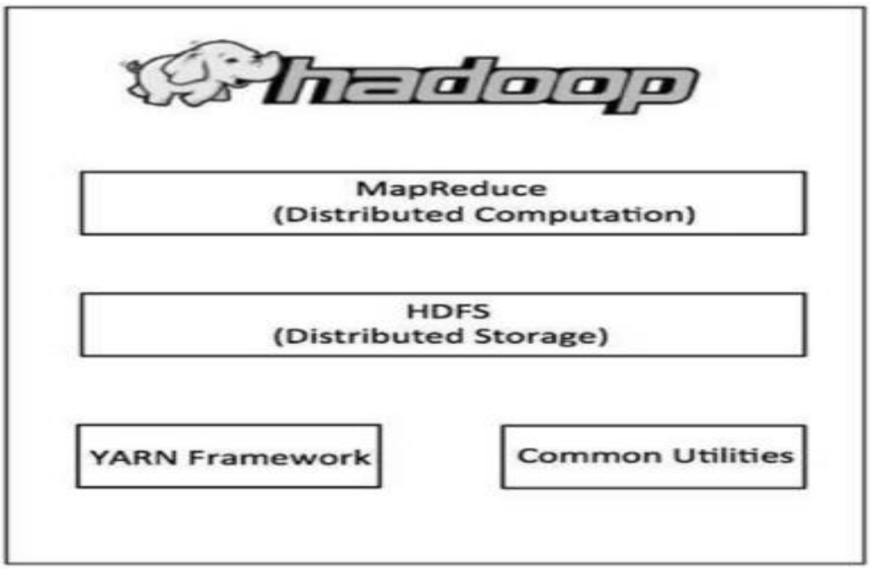
Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel on different CPU nodes. In short, Hadoop framework is capable enough to develop applications capable of running on clusters of computers and they could perform complete statistical analysis for a huge amount of data.



Hadoop Architecture

Hadoop framework includes following four modules:

- **Hadoop Common:** These are Java libraries and utilities required by other Hadoop modules. These libraries provide filesystem and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.
- **Hadoop YARN:** This is a framework for job scheduling and cluster resource management.
- **Hadoop Distributed File System (HDFS):** A distributed file system that provides high-throughput access to application data.
- **Hadoop MapReduce:** This is YARN-based system for parallel processing of large data sets.



Steps of Installation and configuration of Hadoop-

A. Installing Java

Hadoop framework is written in Java!!

\# Update the source list
sunita@sunita:~$**sudo apt-get update**

\# The OpenJDK project is the default version of Java
\# that is provided from a supported Ubuntu repository.

sunita@sunita:~$**sudo apt-get install default-jdk**

sunita@sunita:~$**java -version**
java version "1.7.0_91"
OpenJDK Runtime Environment (IcedTea 2.5.3) (7u71-2.5.3-0ubuntu0.14.04.1)
OpenJDK 64-Bit Server VM (build 24.65-b04, mixed mode)

B. Create User for Hadoop

sunita@sunita:~$**sudo addgroup hadoop**
sunita@sunita:~$ **sudo adduser --ingroup hadoop hduser**
sunita@sunita:~$ **sudo adduser hduser sudo**
sunita@sunita:~$ **sudo apt-get install openssh-server**
sunita@sunita:~$ **su –hduser**

C. Installing SSH (secure shell)

SSH or Secure Shell is a network communication protocol that enables two computers to communicate and share data. An inherent feature of ssh is that the communication between the two computers is encrypted meaning that it is suitable for use on insecure networks.

ssh has two main components:

1. **ssh** : The command we use to connect to remote machines - the client.
2. **sshd** : The daemon that is running on the server and allows clients to connect to the server.

The **ssh** is pre-enabled on Linux, but in order to start **sshd** daemon, we need to install **ssh** first.

sunita@sunita:~$**sudo apt-get install openssh-server**

Create and Setup SSH Certificates-

Hadoop requires SSH access to manage its nodes, i.e. remote machines plus our local machine. For our single-node setup of Hadoop, we therefore need to configure SSH access to localhost.

So, we need to have SSH up and running on our machine and configured it to allow SSH public key authentication.

Hadoop uses SSH (to access its nodes) which would normally require the user to enter a

password. *However, this requirement can be eliminated by creating and setting up SSH certificates using the following commands.* If asked for a filename just leave it blank and press the enter key to continue.
sunita@sunita:~$**ssh-keygen -t rsa -P ""**

sunita@sunita:~$**cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys**

The second command adds the newly created key to the list of authorized keys so that Hadoop can use ssh without prompting for a password.
~~[Note- In case of error – connection refused, we resolve by purge ssh and add it again~~
~~sunita@sunita:~$sudo apt-get purge openssh-server]~~

We can check if ssh works:
sunita@sunita:~$**ssh localhost**
sunita@sunita:~$ **which ssh**

D. Install Hadoop

Download Hadoop-
sunita@sunita:~$**wget** http://mirrors.sonic.net/apache/hadoop/common/hadoop-2.9.**0/hadoop-2.9.0.tar.gz**

sunita@sunita:~$**tar xvzf hadoop-2.9.0.tar.gz**

We want to move the Hadoop installation to the **/usr/local/hadoop** directory using the following command:

sunita@sunita:~$ **sudo mv hadoop-2.9.0 /usr/local/hadoop**
sunita@sunita:~$ **sudo chown -R hduser /usr/local**

E. Setup Configuration Files

The following files will have to be modified to complete the Hadoop setup:

~/.bashrc
/usr/local/hadoop/etc/hadoop/hadoop-env.sh
/usr/local/hadoop/etc/hadoop/core-site.xml
/usr/local/hadoop/etc/hadoop/mapred-site.xml
/usr/local/hadoop/etc/hadoop/hdfs-site.xml

1. sudo gedit ~/.bashrc:

Before editing the **.bashrc** file in our home directory, we need to find the path where Java has been installed to set the **JAVA_HOME** environment variable using the following command:
Now we can append the following to the end of **~/.bashrc**:

sunita@sunita:~$ **sudo gedit .bashrc**

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
```

sunita@sunita:~$**source .bashrc**
This command applies the changes made in the .bashrc file.

2. sudo gedit /usr/local/hadoop/etc/hadoop/hadoop-env.sh

We need to set **JAVA_HOME** by modifying **hadoop-env.sh** file.

sunita@sunita:~$**gedit /usr/local/hadoop/etc/hadoop/hadoop-env.sh**

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

Adding the above statement in the **hadoop-env.sh** file ensures that the value of JAVA_HOME variable will be available to Hadoop whenever it is started up.

3. sudo gedit /usr/local/hadoop/etc/hadoop/core-site.xml

The **/usr/local/hadoop/etc/hadoop/core-site.xml** file contains configuration properties that Hadoop uses when starting up. This file can be used to override the default settings that Hadoop starts with.

```
<property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:9000</value>
</property>
```

4. sudo gedit /usr/local/hadoop/etc/hadoop/hdfs-site.xml

```
<property>
        <name>dfs.replication</name>
        <value>1</value>
</property>
<property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/usr/local/hadoop_tmp/hdfs/namenode</value>
</property>
<property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/usr/local/hadoop_tmp/hdfs/datanode</value>
</property>
```

Create directories for Hadoop file System-
sunita@sunita:~$sudo mkdir -p /usr/local/hadoop_tmp
sunita@sunita:~$sudo mkdir -p sunita@sunita:~$/usr/local/hadoop_tmp/hdfs/namenode
sunita@sunita:~$sudo mkdir -p sunita@sunita:~$/usr/local/hadoop_tmp/hdfs/datanode
sunita@sunita:~$sudo chown -R hduser /usr/local/hadoop_tmp

5. sudo gedit /usr/local/hadoop/etc/hadoop/yarn-site.xml

```
<property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
</property>
<property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

6. sudo gedit /usr/local/hadoop/etc/hadoop/mapred-site.xml

```
<property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
</property>
```

F. Format the New Hadoop Filesystem

Now, the Hadoop file system needs to be formatted so that we can start to use it. The format command should be issued with write permission since it creates **current** directory under **/usr/local/hadoop_tmp/hdfs/namenode** folder:
sunita@sunita:~$**hdfs namenode -format**

Note that **hadoop namenode -format** command should be executed once before we start using Hadoop. If this command is executed again after Hadoop has been used, it'll destroy all the data on the Hadoop file system.

G. Starting Hadoop

Now it's time to start the newly installed single node cluster. We can use **start-all.sh** or (**start-dfs.sh** and **start-yarn.sh**)
sunita@sunita:~$**start-all.sh**
We can check if it's really up and running:
sunita@sunita:~$**jps**
9026 NodeManager
7348 NameNode
9766 Jps
8887 ResourceManager
7507 DataNode

As command says JPS which means **Java Process Status** which is used to list all the processes that are running on java virtual machine. The output means that we now have a functional instance of Hadoop running on our VPS (Virtual private server).

Hadoop Web Interfaces

Let's start the Hadoop again and see its Web UI:

Accessing HADOOP through browser

http://localhost:50070/

Verify all applications for cluster

http://localhost:8088/

CONCLUSION:

We have studied Hadoop installation and configuration.

Assignment 2:

TITLE: Design a distributed application using MapReduce

OBJECTIVE:

1. To explore different Big data processing techniques with use cases.
2. To study detailed concept of Map-Reduced.

SOFTWARE REQUIREMENTS:

1. Ubuntu stable version
2. GNU GCC Compiler
3. Hadoop
4. JDK 8

PROBLEM STATEMENT: - Design a distributed application using MapReduce and process it using a pseudo distribution mode on Hadoop platform.

THEORY:

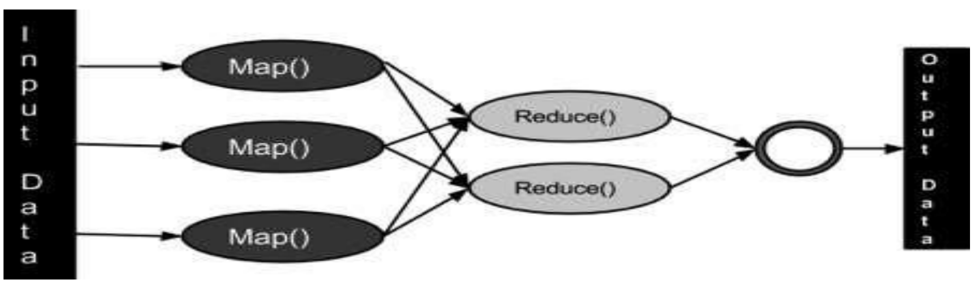
What is MapReduce?

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner. MapReduce is a processing technique and a program model for distributed computing based on java.

The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster are merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.



·0 **Map stage :** The map or mapper's job is to process the input data. Generally, the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
·1 **Reduce stage :** This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

Hadoop Distributed File System :

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

HDFS uses a **master/slave architecture** where master consists of a single **NameNode** that manages the file system metadata and one or more slave **DataNodes** that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system.
They also take care of block creation, deletion and replication based on instruction given by NameNode.

HDFS provides a shell like any other file system and a list of commands are available to interact with the file system.

HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware and provides high throughput access to application data and is suitable for applications having large datasets.

How does Hadoop work?

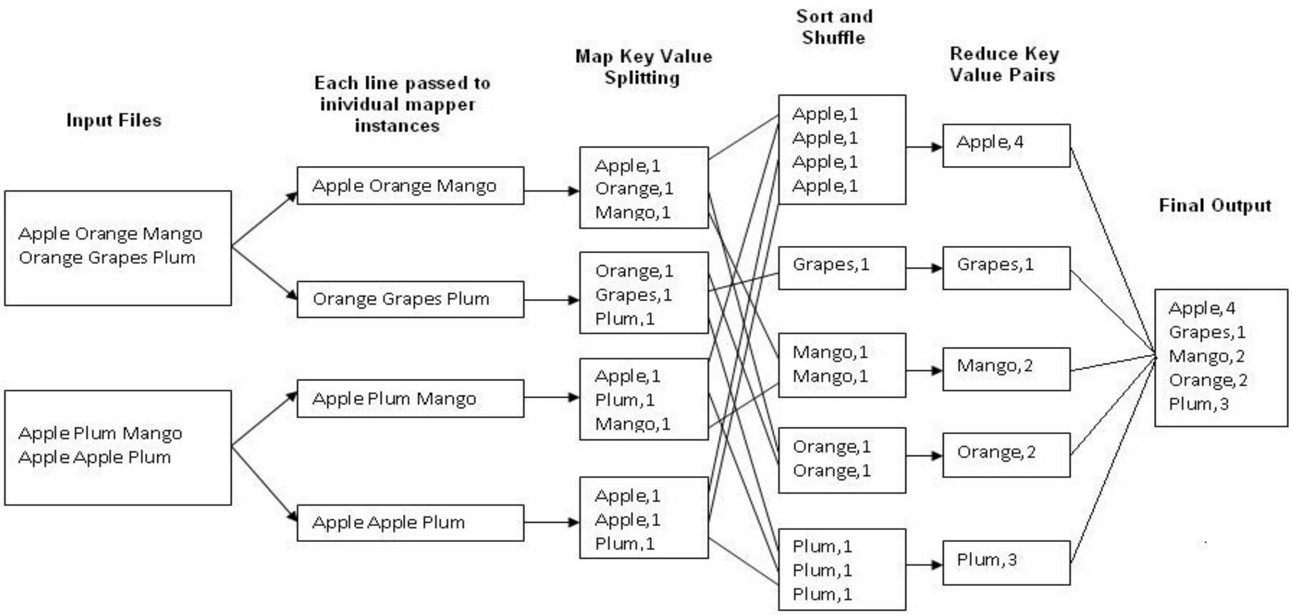
Hadoop runs code across a cluster of computers. This process includes the following core tasks that Hadoop performs:

- Data is initially divided into directories and files. Files are divided into uniform sized blocks (preferably 128MB/256MB).
- These files are then distributed across various cluster nodes for further processing.
- HDFS, being on top of the local file system, supervises the processing.
- Blocks are replicated for handling hardware failure.
- Checking that the code was executed successfully.
- Performing the sort that takes place between the map and reduce stages.
- Sending the sorted data to a certain computer.
- Writing the debugging logs for each job.

Example- Word Count using MapReduce

```
map(key, value):
// key: document name; value: text of document
        for each word w in value:
                emit(w, 1)

reduce(key, values):
// key: a word; values: an iterator over counts
        result = 0
        for each count v in values:
                result += v
        emit(key, result)
```



map (key=url, val=contents):
For each word w in contents, emit (w, "1")
reduce (key=word, values=uniq_counts):
Sum all "1"s in values list
Emit result "(word, sum)"

Conclusion : In this practical we successfully studied about distributed application using MapReduce on Hadoop platform.

Class: TE Year 2022-2023

Subject: Data Science and Big Data Analytics Lab

Assignment 3:

TITLE: Hadoop Ecosystem Components

OBJECTIVE:

1. To understand the different Big data processing techniques.
2. To study Hadoop ecosystem components.
3. To understand emerging trends in Big data analytics.

Study of Hadoop Ecosystem.

a. HDFS -> (Hadoop Distributed File System)- Storage component
b. YARN -> (Yet Another Resource Negotiator) - resource schedular for Hadoop
c. MapReduce -> Data processing using programming paradigm
d. Spark -> In-memory Data Processing – provide real time analytic power
e. PIG, HIVE-> Data Processing Services using Query (SQL-like)
f. HBase -> NoSQL Database on top of HDFS
g. Mahout, Spark MLlib -> Machine Learning ability
h. Flume, Sqoop -> Data Ingesting Services for structured and unstructured data

Conclusion : In this practical we successfully studied about components in Hadoop ecosystem.

Group B: Assignments based on Data Analytics using Python

Assignment 1:

TITLE: Perform basic operations on Datasets using Python

OBJECTIVE:

1. To understand and apply the Analytical concept of Big data using Python.
2. To study Python libraries for Data Analytics

SOFTWARE REQUIREMENTS:

1. Ubuntu 16.04
2. Python-3
3. Anaconda-Spyder/ Jupyter notebook/ Google colab

PROBLEM STATEMENT:

Perform the following operations using Python on the Facebook metrics data sets.

- Create data subsets
- Merge Data
- Sort Data
- Transposing Data
- Shape and reshape Data

THEORY:

Overview of Python Libraries for Data Scientists-

Many popular Python toolboxes/libraries:

- NumPy
- SciPy
- Pandas
- SciKit-Learn

Visualization libraries-

- matplotlib
- Seaborn

NumPy:

- introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects
- provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance
- many other python libraries are built on NumPy

SciPy:

- collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
- part of SciPy Stack built on NumPy

Pandas:

- adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)
- provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- allows handling missing data

SciKit-Learn:

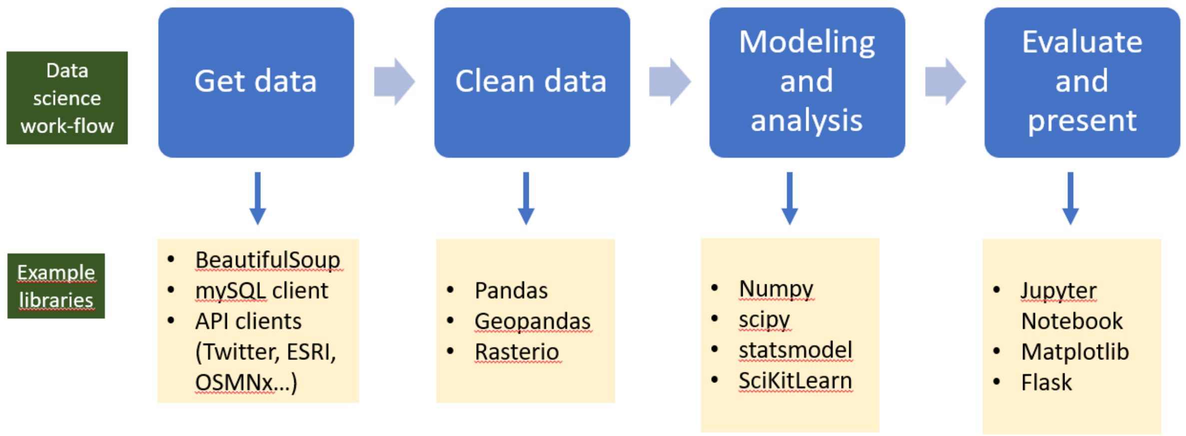
- provides machine learning algorithms: classification, regression, clustering, model validation etc.
- built on NumPy, SciPy and matplotlib

matplotlib:

- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.
- relatively low-level; some effort needed to create advanced visualization

Seaborn:

- based on matplotlib
- provides high level interface for drawing attractive statistical graphics



Standard way in which data is collected and stored file format. It is most used format for storing data is the spreadsheet format where data is stored in rows and columns. Each row is called a record Each column in a spreadsheet holds data belonging to same data type

Commonly used spreadsheet formats are comma separated values and excel sheets. Other formats include plain text, json, html, mp3 ,mp4 etc

Importing data-

```
import os
```

'os' library to change the working directory

```
import pandas as pd
```

'pandas' library to work with dataframes

Changing the working directory-

```
os. chdir("D:\Pandas")
```

Importing csv data-

```
data_csv=pd.read_csv('Iris_data_sample.csv')
```

Removing the extra id column by passing `index_col=0`

```
data_csv=pd.read_csv('Iris_data_sample.csv',index_col=0)
```

Index	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-set…
2	4.9	nan	1.4	0.2	nan
3	4.7	3.2	1.3	0.2	Iris-set…
4	??	3.1	1.5	0.2	Iris-set…
5	5	3.6	###	0.2	Iris-set…

Junk values can be converted to missing values by passing them as a list to the parameter ‘na_values’

```
data_csv=pd.read_csv('Iris_data_sample.csv',
                     index_col=0,na_values=["??"])
data_csv=pd.read_csv('Iris_data_sample.csv',
                     index_col=0,na_values=["??","###"])
```

Introduction to Pandas-

Pandas provides high-performance, easy-to-use data structures and analysis tools for the Python programming language

It is Open-source Python library providing high-performance data manipulation and analysis tool using its powerful data structures. Name pandas is derived from the word Panel Data –an econometrics term for multidimensional data

DataFrame-

Pandas deals with **dataframes object-**

Name	Dimension	Description
Dataframe	2	• two-dimensional size-mutable • potentially heterogeneous tabular data structure with labeled axes (rows and columns)

Pandas types vs Python types-

Pandas Type	Native Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the memory allocated to hold this character.
float64	float	Numeric characters with decimals. If a column contains numbers and NaNs(see below), pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	Values meant to hold time data. Look into these for time series experiments.

Data Frames attributes-

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

Data Frames methods-

df.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

There are two ways to create copies of Data Frames

- Shallow copy
- Deep copy

	Shallow copy	***Deep copy***
Function	`samp=cars_data.copy(deep=False)` `samp = cars_data`	`cars_data1=cars_data.copy(deep=True)`
Description	◦ It only creates a new variable that shares the reference of the original object ◦ Any changes made to a copy of object will be reflected in the original object as well	◦ In case of deep copy, a copy of object is copied in other object with no reference to the original ◦ Any changes made to a copy of object will not be reflected in the original object

Indexing and selecting data

- Python slicing operator **'[ ]'** and attribute/dot operator **'.'** are used for indexing
- Provides quick and easy access to pandas data structures
  a. To access a scalar value, the fastest way is to use the **at** and **iat** methods.
    - **at** provides label-based scalar lookups

```
In [29]: cars_data1.at[4,'FuelType']
Out[29]: 'Diesel'
```

    - **iat** provides integer-based lookups

```
In [30]: cars_data1.iat[5,6]
Out[30]: 0
```

  b. To access a group of rows and columns by label(s). **loc [ ]** can be used

```
In [31]: cars_data1.loc[:,'FuelType']
```

Selecting a column in a Data Frame-

Method 1: Subset the data frame using column name: df['Age']
Method 2: Use the column name as an attribute: df.age

Creating Subset-

There are a number of ways to subset the Data Frame:

- one or more columns
- one or more rows
- a subset of rows and columns

Rows and columns can be selected by their position or label

To subset the data we can apply Boolean indexing. This indexing is commonly known as a filter. For example, if we want to subset the rows in which the salary value is greater than $120K:

```
#Calculate mean salary for each professor rank:
df_sub = df[ df['salary'] > 120000 ]
```

 # subset of initial specified observations

```
dset.head(20)
# Subset of last specified observations
dset.tail(40)
# subset selecting specified columns only
sub1= dset[['pagetotallikes','type','category','comment']]
#subset having specfied colums and range of observations
sub2= dset[['pagetotallikes','type','category','comment']].loc[50:300]
suset=dset[['pagetotallikes','type']].loc[(dset['type']!=2)]
#subset having observations constrained on some column
sub6=dset[ dset['type']==3 ]

#------------------------------------------
## Create subset and merge dataset
sub2= dset[['pagetotallikes','type','category','comment']].loc[50:150]
sub3= dset[['pagetotallikes','type','category','comment']].loc[151:300]
sub4= dset[['pagetotallikes','type','category','comment']].loc[1:25]

# obsesrvations appended using pd.concat()
mergedSet=pd.concat([sub2,sub4])

#------------------------------------------------
## Merge on some common variable
#1. Create a dictionary where keys are column names and values are opbservation
d={'Student name':['raj','mahesh','jon'],'age':[22,23,25]}
#2. Pass the dictionary to DataFrame method
df=pd.DataFrame(d)
d1={'roll no':[1,2,3],'Student name':['raj','dilip','raam'],'age':[22,22,22]}
df1=pd.DataFrame(d1)

merged1=pd.merge (df, df1, on='Student Name')

##----------------------------------
### Sort observations in column values oder
sorteddset=dset.sort_values('pagetotallikes')
sorteddset=dset.sort_values('pagetotallikes', ascending=False)

#------------------------------------------------
### Transpose
Tsub4=sub4.transpose()
#------------------------------------------------
#shape and reshape like pivot table
df1.shape
p_table=pd.pivot_table(df1,index=['roll no','Student name'],values='age')
p_table.shape
```

Conclusion: Hence, we have studied create, merge, sort, transpose and reshape operations on Dataset.

Assignment 2:

TITLE: Perform Data preparation operation on Datasets using Python

OBJECTIVE:

1. To understand and apply the Analytical concept of Big data usingnPython.
2. To study Python libraries for Data Analytics

SOFTWARE REQUIREMENTS:

1. Ubuntu 16.04
2. Python-3
3. Anaconda-Spyder/ Jupyter notebook/ Google colab

PROBLEM STATEMENT:

Perform the following operations using Python on the Air quality and Heart Diseases data sets

- Data cleaning
- Data integration
- Data transformation
- Error correcting
- Data model building

THEORY:

Data cleaning, or data preparation is an essential part of statistical analysis. In fact, in practice it is often more time-consuming than the statistical analysis itself

Pandas Data Types-

The way information gets stored in a dataframe or a python object affects the analysis and outputs of calculations

- There are two main types of data
  1. numeric types and
  2. character types

1. Numeric data types includes integers and floats

◦ For example: integer – 10, float – 10.5

Pandas and base Python uses different names for data types

Python data type	Pandas data type	Description
int	int64	Numeric characters
float	float64	Numeric characters with decimals

◦ '64' simply refers to the memory allocated to store data in each cell which effectively relates to how many digits it can store in each "cell"
◦ 64 bits is equivalent to 8 bytes
◦ Allocating space ahead of time allows computers to optimize storage and processing efficiency

2. Character types

Strings are known as objects in pandas which can store values that contain numbers and / or characters

category	object
◦ A string variable consisting of only a few different values. Converting such a string variable to a categorical variable will save some memory	◦ The column will be assigned as object data type when it has mixed types (numbers and strings). If a column contains 'nan'(blank cells), pandas will default to object datatype.
◦ A categorical variable takes on a limited, fixed number of possible values	◦ For strings, the length is not fixed

Checking data types of each column-

dtypes returns a series with the data type of each column.

Syntax: `DataFrame.dtypes`

```
cars_data1.dtypes
```

```
Out[37]:
Price          int64
Age          float64
KM            object
FuelType      object
HP            object
```

Count of unique data types-

get_dtype_counts()returns counts of unique data types in the dataframe

Syntax: `DataFrame.get_dtype_counts()`

```
cars_data1.get_dtype_counts()
```

```
Out[38]:
float64    2
int64      4
object     4
dtype: int64
```

Selecting data based on data types-

DataFrame.select_dtypes() returns a subset of the columns from dataframe based on the column dtypes.

Syntax: `DataFrame.get_dtype_counts()`

```
cars_data1.get_dtype_counts()
```

```
Out[38]:
float64     2
int64       4
object      4
dtype: int64
```

Concise summary of dataframe-

`info()` returns a concise summary of a dataframe

- data type of index
- data type of columns
- count of non-null values
- memory usage

Syntax: `DataFrame.info()`

```
cars_data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
Price        1436 non-null int64
Age          1336 non-null float64
KM           1436 non-null object
FuelType     1336 non-null object
HP           1436 non-null object
MetColor     1286 non-null float64
Automatic    1436 non-null int64
CC           1436 non-null int64
Doors        1436 non-null object
Weight       1436 non-null int64
dtypes: float64(2), int64(4), object(4)
memory usage: 163.4+ KB
```

Unique elements of columns-

Syntax: `numpy.unique(array)`

```
print(np.unique(cars_data1['KM']))
```

```
['1' '10000' '100123' ... '99865' '99971' '??']
```

- 'KM' has special character to it - '??'
- Hence, it has been read as object instead of int64

Data Cleaning-

We need to know how missing values are represented in the dataset in order to make reasonable decisions. Python, by default replace blank values with 'nan'

The missing values also exist in the form of 'nan','??', '????' etc.

We can import the data considering other forms of missing values in a dataframe

```
cars_data = pd.read_csv('Toyota.csv',index_col=0,
                        na_values=["??","????"])
```

Two ways of approach

Fill the missing values by mean / median, in case of numerical variable

Fill the missing values with the class which has maximum count, in case of categorical variable

Imputing missing values of numerical variable-

- To fill NA/NaN values using the specified value

`DataFrame.fillna()`

```
cars_data2['Age'].fillna(cars_data2['Age'].mean(),
                          inplace = True)
```

- To fill NA/NaN values using the specified value

`DataFrame.fillna()`

```
cars_data2['KM'].fillna(cars_data2['KM'].median(),
                         inplace = True)
```

Imputing missing values of Categorical variables-

`Series.value_counts()` Returns a Series containing counts of unique values

• The values will be in descending order so that the first element is the most frequently-occurring element

• Excludes NA values by default

- To get the mode value of ***FuelType***

```
cars_data2['FuelType'].value_counts().index[0]
Out[29]: 'Petrol'
```

- To fill NA/NaN values using the specified value

`DataFrame.fillna()`

```
cars_data2['FuelType'].fillna(cars_data2['FuelType']\
                    .value_counts().index[0],\
                    inplace = True)
```

Imputing missing values using lambda functions-

- To fill the NA/ NaN values in both numerical and categorial variables at one stretch

```
cars_data3 = cars_data3.apply(lambda x:x.fillna(x.mean()) \
                              if x.dtype=='float' else \
                              x.fillna(x.value_counts().index[0]))
```

Data transformation-

Converting variable’s data types-

astype() method is used to explicitly convert data types from one to another

Syntax: `DataFrame.astype(dtype)`

Converting ‘MetColor’ ,‘Automatic’ to object data type:

```
cars_data['MetColor'] = cars_data['MetColor'].astype('object')
```

```
print(np.unique( cars_data['Doors']))
['2' '3' '4' '5' 'five' 'four' 'three']
```

- `replace()` is used to replace a value with the desired value
- Syntax: `DataFrame.replace([to_replace, value, …])`

```
cars_data['Doors'].replace('three',3,inplace=True)
cars_data['Doors'].replace('four',4,inplace=True)
cars_data['Doors'].replace('five',5,inplace=True)
```

```
cars_data['Doors']=cars_data['Doors'].astype('int64')
```

To check the count of missing values present in each column
`Dataframe.isnull.sum()` is used

Data Transformation using custom functions-

Functions are created using the command def and a colon with the statements to be executed
indented as a block. Since statements are not demarcated explicitly, It is essential to follow correct indentation practices

```
def function_name(parameters):
            statements
```

Example-

- Here, a function c_convert has been defined
- The function takes arguments and returns one value

```
def c_convert(val):
    val_converted = val/12
    return val_converted

cars_data1["Age_Converted"]=c_convert(cars_data1['Age'])
cars_data1["Age_Converted"]=round(cars_data1["Age_Converted"],1)
```

Function with multiple inputs and outputs-

- A multiple input multiple output function c_convert has been defined
- The function takes in two inputs
- The output is returned in the form of a list

```
def c_convert(val1,val2):
    val_converted = val1/12
    ratio         = val2/val1
    return [val_converted,ratio]
```

- Here, Age and KM columns of the data set are input to the function
- The outputs are assigned to 'Age_Converted' and 'km_per_month'

```
cars_data1["Age_Converted"],cars_data1["Km_per_month"] =
c_convert(cars_data1['Age'],cars_data1['KM'])
```

```
# Define a transformation function for normalization of variable-
def normalize(x):
   return (x - x.mean()) / x.std()

# Apply the transformation function to a column
merged_data["Age"] = merged_data["Age"].apply(normalize)
```

Error correcting operations -

```
# Check for duplicate rows
        print("Number of duplicate rows:", air_quality.duplicated().sum())
# Check for missing values
        print("Missing values:", air_quality.isnull().sum())
```

```
# Impute missing values with mean
        air_quality.fillna(air_quality.mean(), inplace=True)
# Drop irrelevant columns
        air_quality.drop(columns=["Date"], inplace=True)
# Check for quartile
        q1 = air_quality.quantile(0.25)
        q3 = air_quality.quantile(0.75)
```

‘

Conclusion: Hence, we have studied data Cleaning Operations in Python.

Assignment 3:

TITLE: Visualize the data using Python libraries matplotlib, seaborn

OBJECTIVE:

1. To understand and apply the Analytical concept of Big data using Python.
2. To study Python libraries for Data visualization

SOFTWARE REQUIREMENTS:

1. Ubuntu 16.04
2. Python-3
3. Anaconda-Spyder/ Jupyter notebook/ Google colab

PROBLEM STATEMENT:

Visualize the data using Python libraries matplotlib, seaborn by plotting histogram, scatter-plot and bar-plot

Data Visualization-

Data visualization allows us to quickly interpret the data and adjust different variables to see their effect. Technology is increasingly making it easier for us to do so.

Data visualization helps to-

o Observe the patterns
o Identify extreme values that could be anomalies
o Easy interpretation of data insights

Python offers multiple graphing libraries that offers diverse features-

• *matplotlib*	• to create 2D graphs and plots
• *pandas visualization*	• easy to use interface, built on Matplotlib
• *seaborn*	• provides a high-level interface for drawing attractive and informative statistical graphics
• *ggplot*	• based on R’s ggplot2, uses Grammar of Graphics
• *plotly*	• can create interactive plots

Create basic plots using Matplotlib library:

Matplotlib is a 2D plotting library which produces good quality figures

• Although it has its origins in emulating the MATLAB graphics commands, it is independent of MATLAB

• It makes heavy use of NumPy and other extension code to provide good performance even for large arrays

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

↓

'matplotlib' library to do visualization

We should import and clean data before applying data visualization-

- Importing data

```
cars_data = pd.read_csv('Toyota.csv',index_col=0,
                        na_values=["??","????"])
```

Variable explorer

Name	Type	Size
cars_data	DataFrame	(1436, 10)

- Removing missing values from the dataframe

```
cars_data.dropna(axis = 0, inplace=True)
```

Variable explorer

Name	Type	Size
cars_data	DataFrame	(1096, 10)

1. Scatter Plot

A scatter plot is a set of points that represents the values obtained for two different variables plotted on a horizontal and vertical axis.

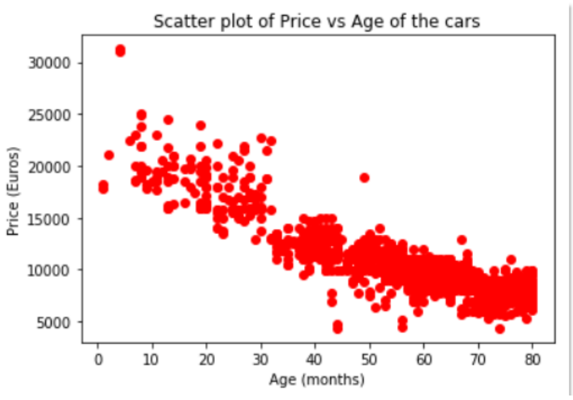
- When to use scatter plots?

Scatter plots are used to convey the relationship between two numerical variables.

Scatter plots are sometimes called correlation plots because they show how two variables are correlated.

```
               x                       y
plt.scatter(cars_data['Age'], cars_data['Price'], c='red')
plt.title('Scatter plot of Price vs Age of the cars')
plt.xlabel('Age (months)')
plt.ylabel('Price (Euros)')
plt.show()
```

- The price of the car decreases as age of the car increases



2. Histogram-

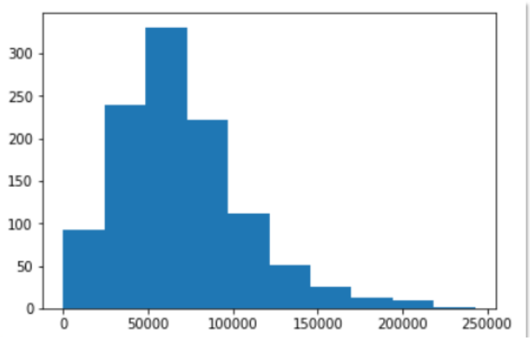
It is a graphical representation of data using bars of different heights. Histogram groups numbers into ranges and the height of each bar depicts the frequency of each range or bin

- When to use histograms?
  To represent the frequency distribution of numerical variables

x

```
plt.hist(cars_data['KM'])
```

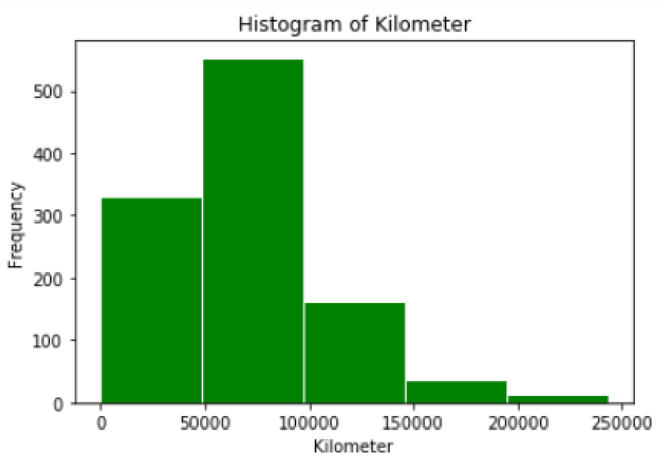
⟶ Histogram with default arguments



Histogram with set arguments color, edgecolor and bins-

```
plt.hist(cars_data['KM'],
         color     = 'green',
         edgecolor = 'white',
         bins      = 5)
plt.title('Histogram of Kilometer')
plt.xlabel('Kilometer')
plt.ylabel('Frequency')

plt.show()
```



Frequency distribution of kilometre of the cars shows that most of the cars have travelled between 50000 – 100000 km and there are only few cars with more distance travelled

3. Bar Plot-

A bar plot is a plot that presents categorical data with rectangular bars with lengths proportional to the counts that they represent.

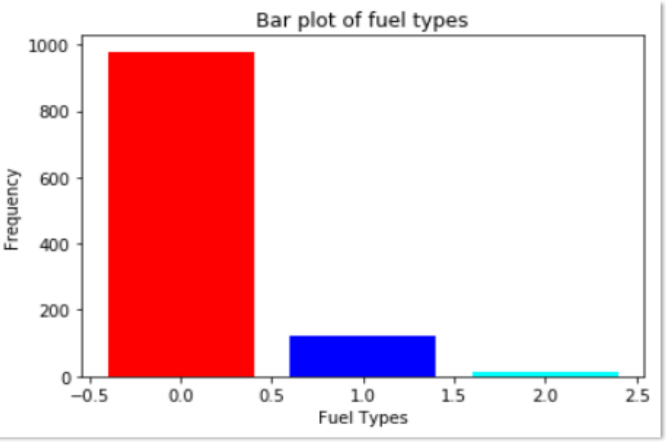
- When to use bar plot?
  To represent the frequency distribution of categorical variables
  A bar diagram makes it easy to compare sets of data between different groups

```
counts   = [979, 120, 12]
fuelType = ('Petrol', 'Diesel', 'CNG')
index    = np.arange(len(fuelType))
```

x height of the bars

```
plt.bar(index, counts, color=['red', 'blue', 'cyan'])
plt.title('Bar plot of fuel types')
plt.xlabel('Fuel Types')
plt.ylabel('Frequency')
plt.show()
```

- Frequency distribution of fuel type



Conclusion: Hence, we have studied data visualization using matplotlib.

Create basic plots using seaborn library:

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

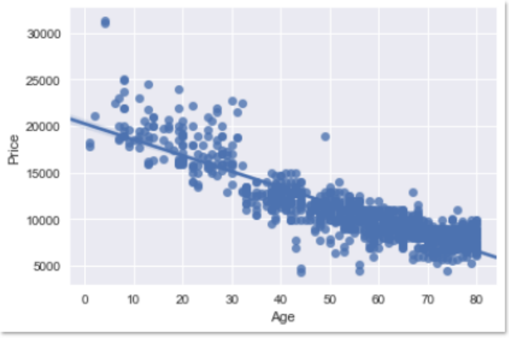
```
import seaborn as sns
```

→ 'seaborn' library to do visualization

1. Scatter Plot-

- Scatter plot of *Price vs Age* with default arguments

```
sns.set(style="darkgrid")
sns.regplot(x=cars_data['Age'], y=cars_data['Price'])
```



- By default, `fit_reg = True`
- It estimates and plots a regression model relating the x and y variables

- Using hue parameter, including another variable to show the fuel types categories with different colors

```
sns.lmplot(x='Age', y='Price', data=cars_data,
           fit_reg=False, hue='FuelType',
           legend=True, palette="Set1")
```

- Scatter plot of *Price vs Age by FuelType*

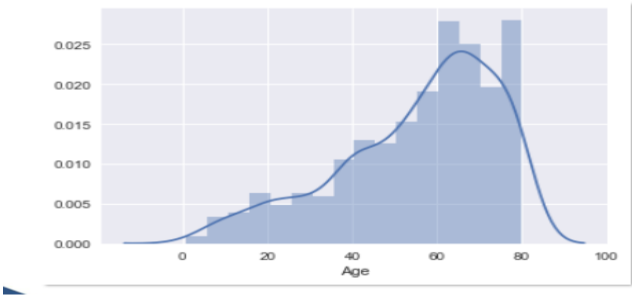
Similarly, custom the appearance of the markers using

- transparency
- shape
- size

2. Histogram-

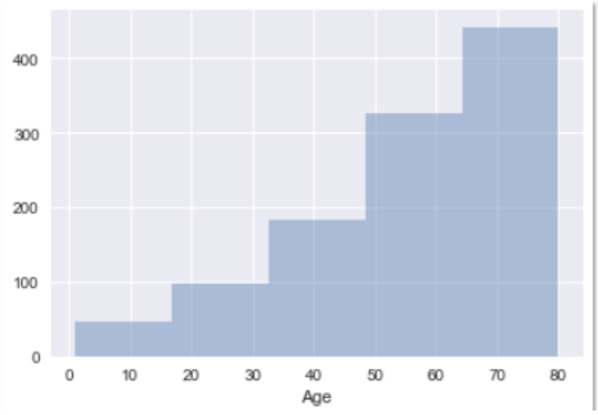
- Histogram with default kernel density estimate

```
sns.distplot(cars_data['Age'] )
```



- Histogram with fixed no. of bins

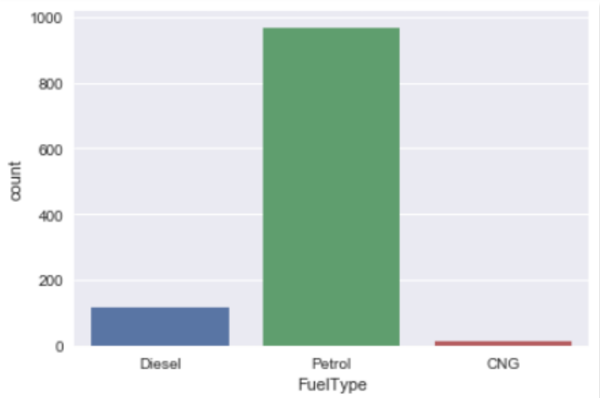
```
sns.distplot(cars_data['Age'], kde = False, bins=5 )
```



3. Bar Plot-

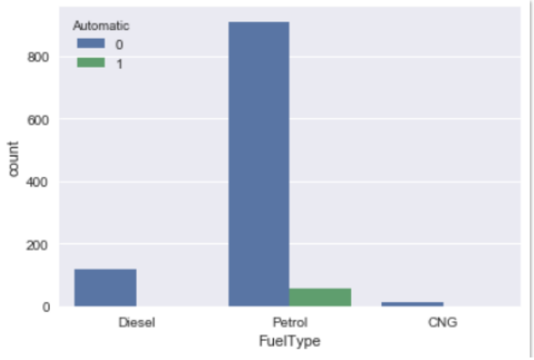
- Frequency distribution of fuel type of the cars

```
sns.countplot(x="FuelType", data=cars_data)
```



- Grouped bar plot of *FuelType* and *Automatic*

```
sns.countplot(x="FuelType", data=cars_data, hue = "Automatic")
```




```
pd.crosstab(index   = cars_data['Automatic'],
            columns = cars_data2['FuelType'],
            dropna  = True)
```

Out[5]:

FuelType	CNG	Diesel	Petrol
Automatic			
0	15	144	1104
1	0	0	73

Conclusion: Hence, we have studied data visualization using seaborn library.

Assignment 4:

TITLE: Data Visualization using Tableau

OBJECTIVE:

1. Visualize the Big Data using Tableau.
2. Design and develop Big data analytic application for emerging trends.

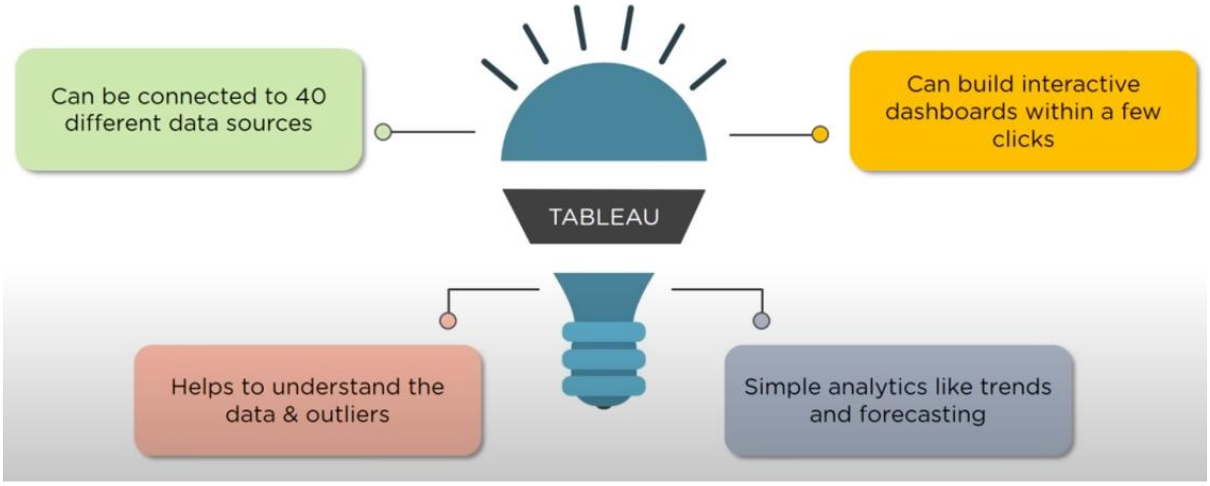
SOFTWARE REQUIREMENTS:

1. Ubuntu 16.04
2. Tableau Desktop / Tableau Public

PROBLEM STATEMENT:

Perform the data visualization operations using Sales order dataset using tableau.

1 Upload dataset CSV file
2 Plot Sales by region
3 Plot year, month, quarterwise Sale
4 Plot yearwise sale vs profit



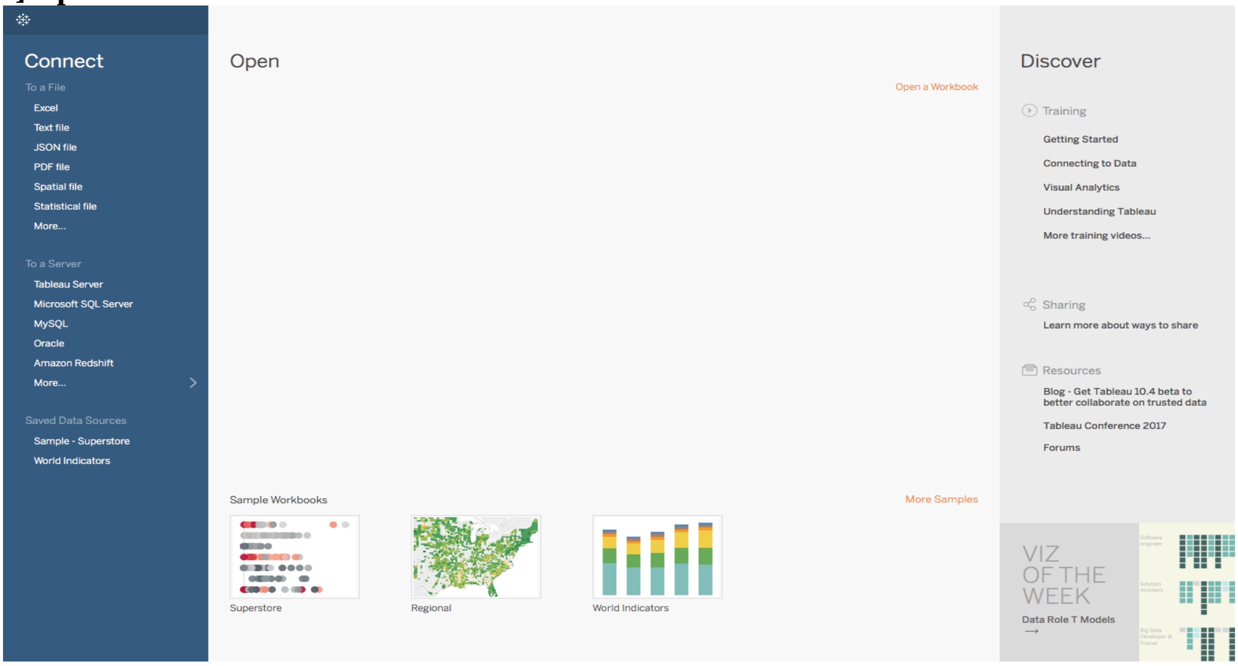
Theory:

Tableau is a Data Visualization tool that is widely used for Business Intelligence but is not limited to it. It helps create interactive graphs and charts in the form of dashboards and worksheets to gain business insights.

Tableau Public

Tableau Public is purely free of all costs and does not require any license. But it comes with a limitation that all of your data and workbooks are made public to all Tableau users.

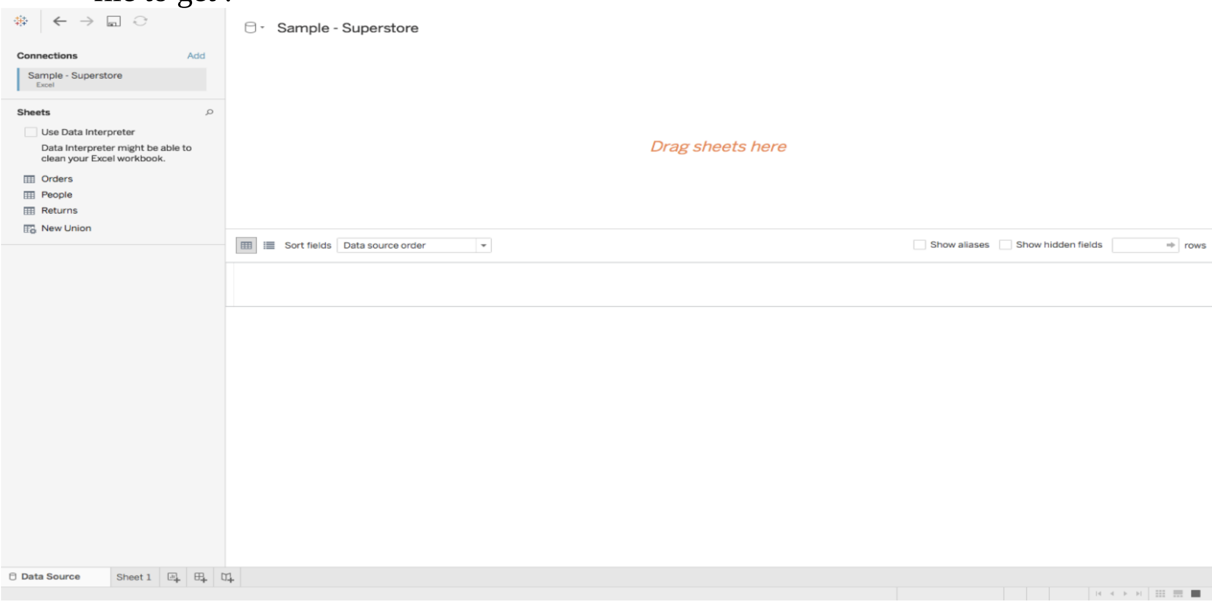
1] Upload dataset CSV file



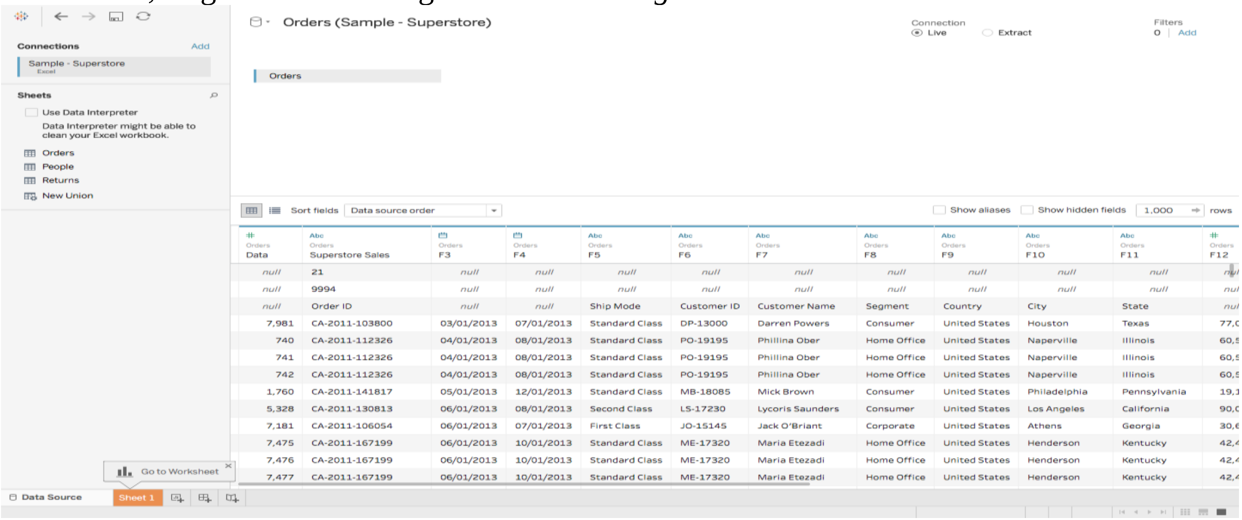
You should see a screen similar to the one above. This is where you import your data. As is visible, there are multiple formats that your data can be in. It can be in a flat-file such as Excel, CSV or you can directly load it from data servers too.

steps:

1. Since the data is in an Excel File, click on Excel and choose the Sample – Superstore.xls file to get :

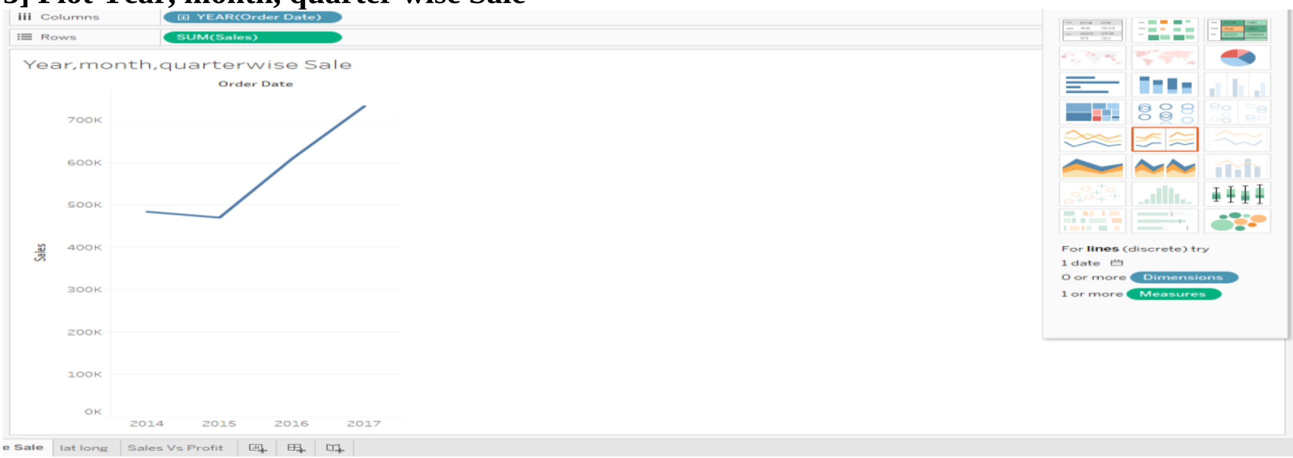


2. You can see three sheets on the screen, but we are only going to be dealing with Orders here, so go ahead and drag the same on *Drag sheets here :*

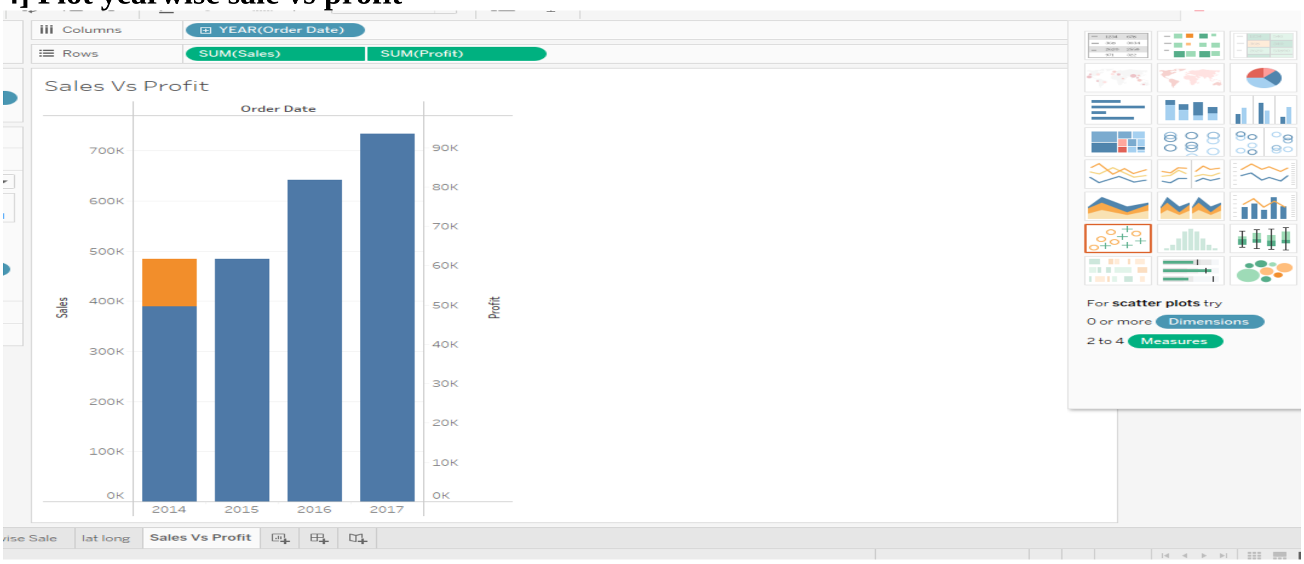


2] Plot Sales By region

3] Plot Year, month, quarter wise Sale



4] Plot yearwise sale vs profit



Conclusion: Hence, we have studied data visualization using tableau.