Coursework

Big Data Analytics and Data Visualization(7153CEM)

National Stock Exchange (NSE)– Banking Sector, Visualization, Analysis and Prediction through Big Data Technologies using PySpark

Name: Jusel Justin

SID: 12646627

Mail ID: juseljustj@uni.coventry.ac.uk

Dataset: https://www.kaggle.com/datasets/sumandey/national-stock-exchange-banking-sectors

Module Leader: Dr. Marwan Faud

Abstract:

National Stock Exchange is in India, Limited (NSE) is the leading stock exchange under the ownership of Ministry of Finance, Government of India, which is located at Mumbai, Maharashtra, India established in 1992. It is India's largest exchange by turnover. One of the key issues around the world is to understand the trends of Stock Market. Banking is one of the main sectors in NSE India. In this project, let us understand the performances of each bank, the investors of which bank will gain a good return and simultaneously which investors of which bank will face loss.

Proposal:

- In this project we will discuss and analyze the Stock Market trends around the banking sectors of India.
- Linear Regression machine learning algorithm will be used to predict the stock market performance.
- PySpark will be used to load and process data.
- Clearly visualize using the visualization tool tableau.
- Predict future performances using the most suitable algorithms (Linear Regression) and other parameters.
- Discuss the findings.

Introduction:

Big Data Analytics is used to provide additional insights and context on trends in stock marketing .There are many applications of ML , Data Analytics and Data Science Tools in the field of Banking Sector . Sorting the data, managing it and organizing the data in the correct format has always been a task as it is generated in different formats like Structured, Unstructured, Semi-structured and Quasi-structured data. Data is generated by different organizations, and it becomes information when interpreted correctly. According to the Forbes article there are 2.5 quintillion bytes of data generated each day considering only Internet of things. Using big data analysis is the most ideal way for analyzing this huge amount of data. Big data analytics tools and techniques help in identifying patterns and thereby giving meaning to the data.

Considering the data set chosen we can determine the customer behavioral pattern, the profits gained, the losses encountered, the current market trends in stocks thereby, determining the solutions. In addition, by adding machine learning algorithms we can get a more accurate result.

Apache spark framework is used in this coursework as it is the best platform supporting programming languages like Scala, Java, R and Python. PySpark is released to support both Apache Spark and Python, It is a Python API for Spark. In this coursework we will be using python with spark that is PySpark to analyse the dataset. Python is easy to implement, the readability and maintenance of the code is way better as compared to that of Scala or Java.

Dataset;

The task in this project is to analyze the Stock market trends in the Banking sector. It is one of the major issues being faced by people around the world, this problem can be approached by using suitable statistical methods. The data set, National Stock Exchange- Banking Sector is chosen from Kaggle mainly focuses on the Indian market. The dataset consists of data from 2016-2021 of the bank's performances. The dataset consists of 15 columns and 41231 rows.

The outcome of this coursework will be to visualize and explore the dataset using tableau. Using PySpark to analyse the dataset. Regression models like Mean Absolute Error (MAE), Root mean square Error (RMSE) is used to acquire accuracy. Prediction of future performances is done by using Linear Regression algorithm. To understand the best fit model for the dataset R-squared and Adjusted R-squared will be used.

Installation:

1.Oracle VM Virtual Box with Ubuntu:

It is a cross platform virtualization software which runs in multiple operating system including windows . Oracle VM Virtual Box-6.1.38 is installed and run from https://www.virtualbox.org/ ,simultaneously Ubuntu 22.04.1 is installed from https://ubuntu.com/download/desktop. Ubuntu is a Linux OS. Virtual machine is installed to run ubuntu on Windows OS.

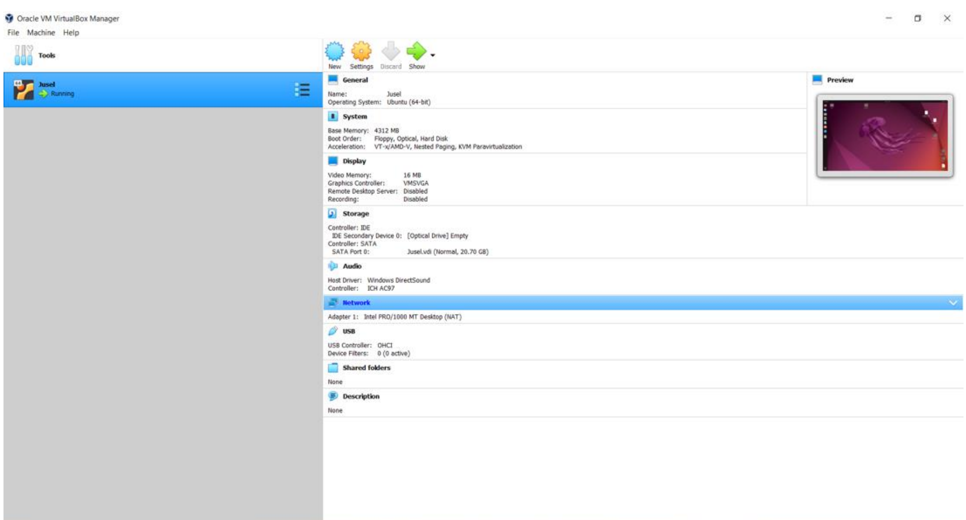


Diagram 1:Installation of virtual machine with ubuntu

2. JAVA:

JAVA is a high level object oriented programming language used to construct applications , games and other device softwares Let's check the version of java in the terminal before proceeding with the installation of spark and hadoop in the system with the command.

$ java –version

If not installed update the java version using

sudo apt update

sudo apt install openjdk-8-jdk -y

java -version

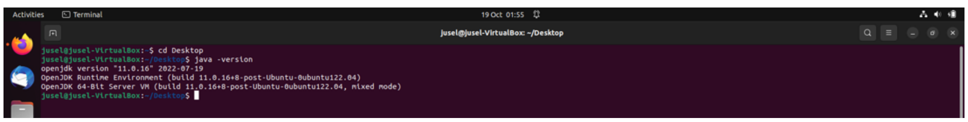


Diagram 2: *The Java version 11.0.16 is installed in the system*.

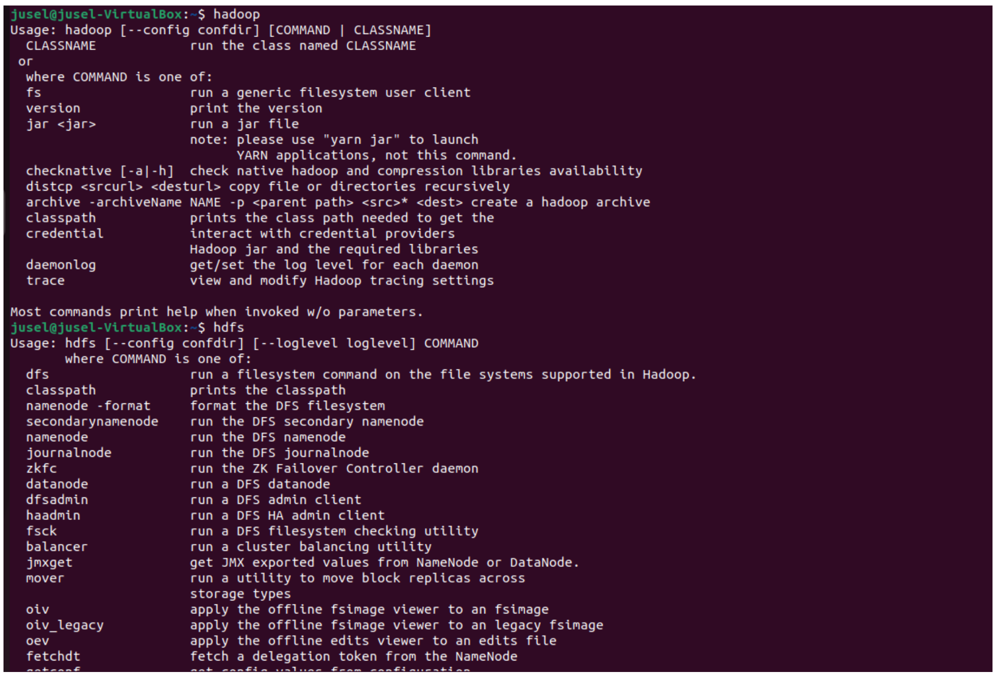
3.HADOOP:

Hadoop is a open source frame work that is used to affectively store data and process large data sets. Hadoop is downloaded and after downloading uncompressed using the command below.

$ tar –xzf hadoop-2.7.3.tar.gz

To check if the installation is compl;eted successfully use the command below

$ hadoop



<u>Diagram 3: Installation completed</u>

4.SPARK:

SPARK is also a open source framework which focuses on machine learning and real time work loads . It works on Microsoft windows , macOS and LINUX .To download spark effectively lets first check the python version in the terminal using the command below. The python version is 3.10.4

$ python3 --version

Diagram 4.python version

Un compress Spark by using command

$ tar –xzf spark-2.3.0-bin-hadoop2.7.tgz

Set file directory by using the command:

$ export SPARK_HOME=`pwd`/spark-2.3.0-bin-hadoop2.7 $

PATH=$SPARK_HOME/bin:$PATH

Run Spark shell using the command

$ spark-shell

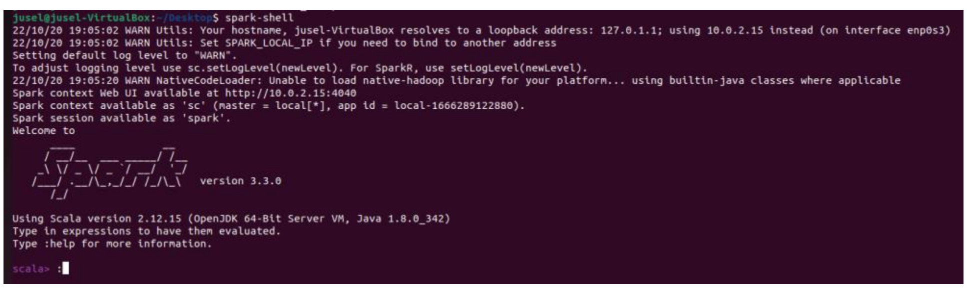


Diagram 5. Spark installed

SPARK job is a pluggable environment in spark which relies on management of cluster to launch . By clicking on the UI link, we can get to Spark Jobs.

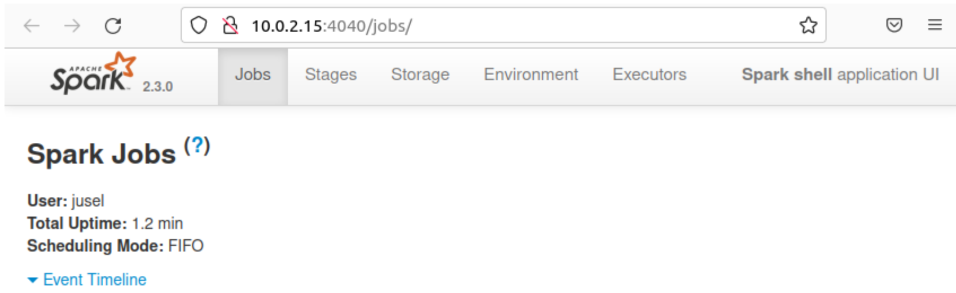


Diagram 6. Spark Jobs

In spark jobs we can notice the properties of java, hadoop and scala.

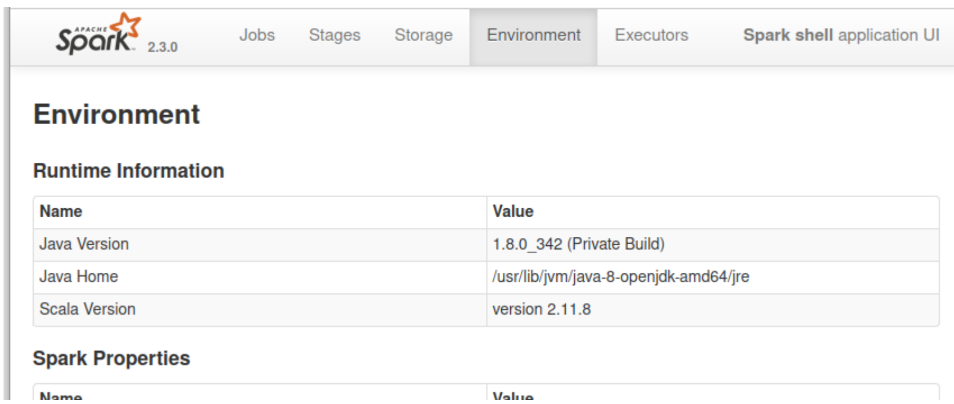


Diagram 7: Spark properties

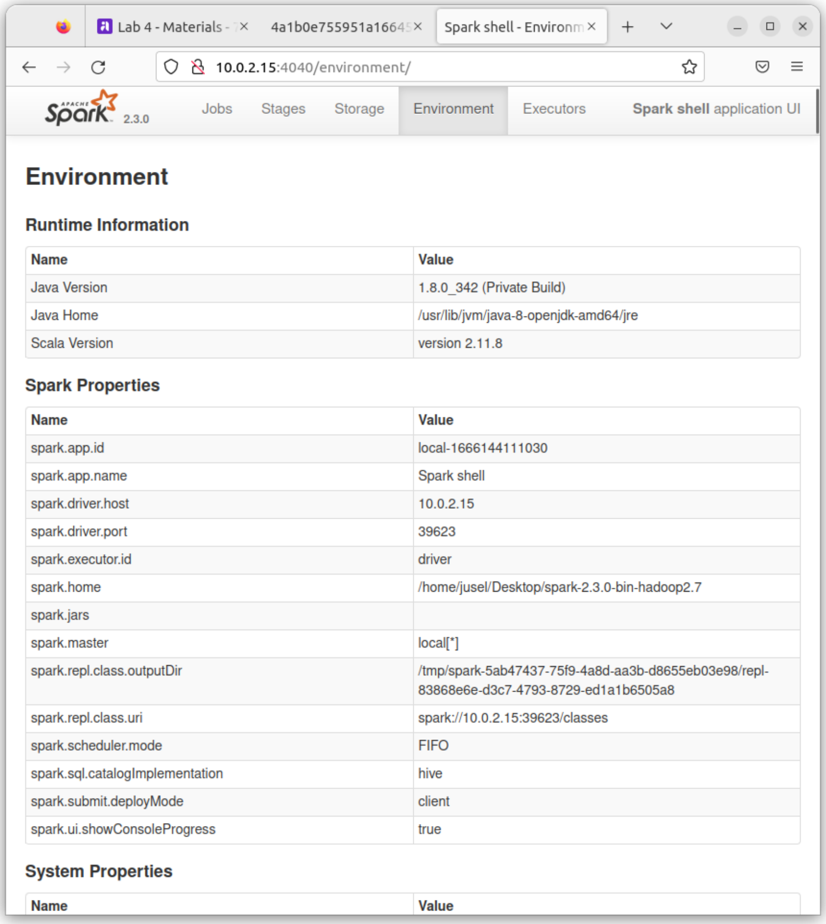


Diagram 8: Spark Environment

5. PYSPARK:

Install Pyspark with command given below:

$ pip install pyspark

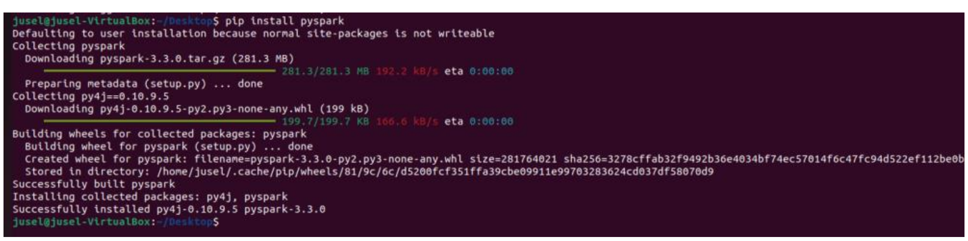


Diagram 9

And check if PySpark is installed with command

$ pyspark

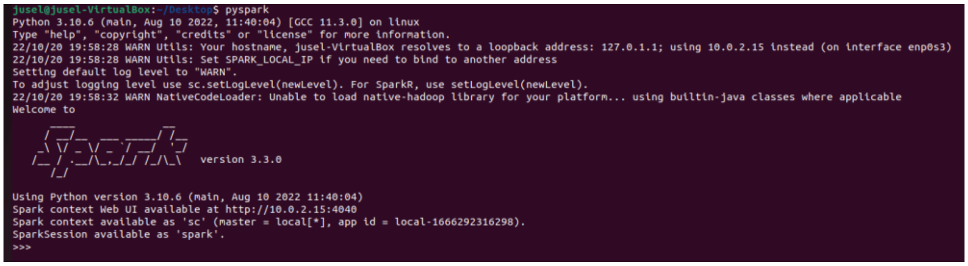


Diagram 10.Pyspark shell

6. JUPYTER:

Jupyter Notebook is user friendly as it allows the users to compile in one place which makes the entire process easy .To install jupyter the command

$ pip install jupyter

Add path by using the commands

$ export PYSPARK_DRIVER_PYTHON="jupyter"

$ export PYSPARK_DRIVER_PYTHON_OPTS="notebook"

$ export PYSPARK_PYTHON=python3

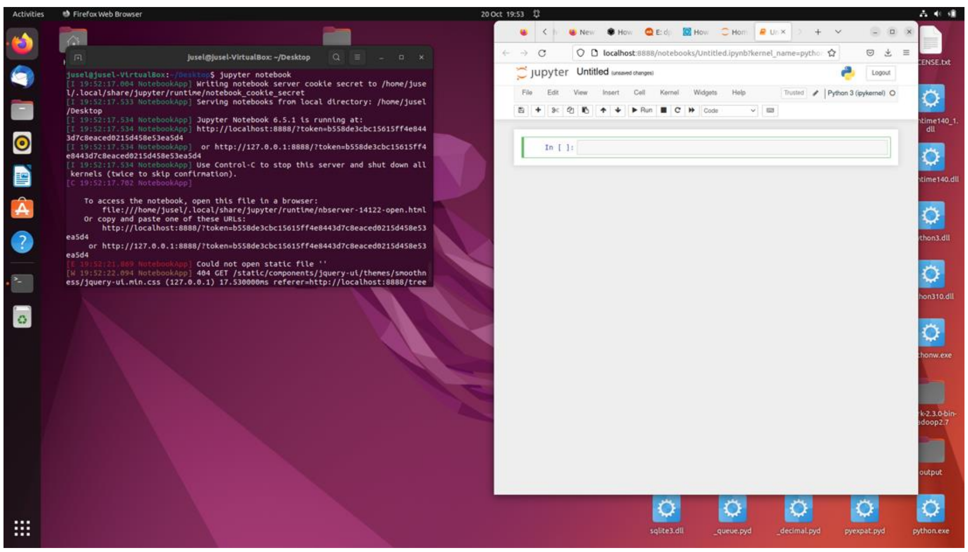


Diagram 11:Jupyter terminal

OTHER INSTALLATIONS:

To install numpy:

$ pip install numpy

Implementation

1.SPARK:

Spark was introduced to solve the inefficiency of MapReduce. It is a real-time and batch processing framework. MapReduce is only effective in batch processing. Spark resolves the problem by even doing real-time processing. Spark is an open-source platform suitable for large scale data processing. Spark supports SparkSQL, Spark Unified Stack, MLib framework for machine learning etc.. Spark was developed using Scala, it is available in Scala, Java, SQL, Python, R, C#, F#. In this coursework we are using Spark with python that is PySpark.

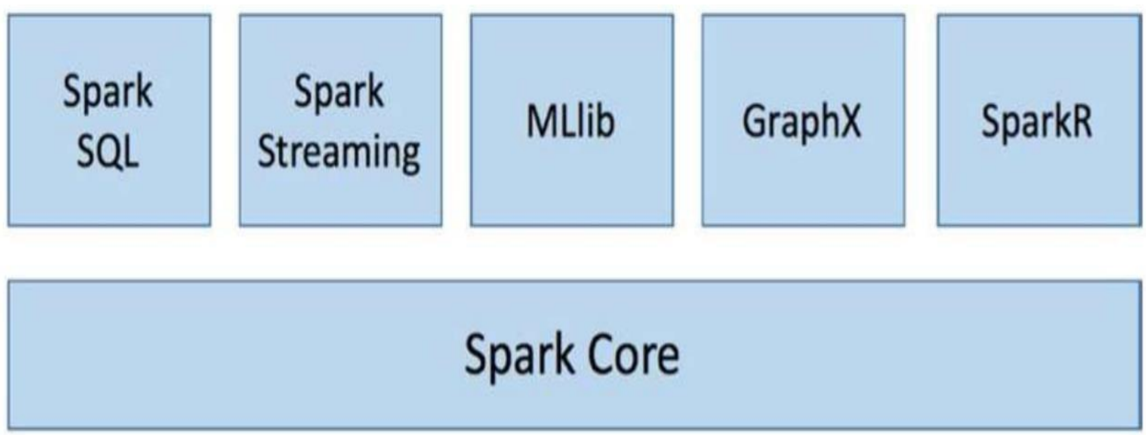


Diagram 12: From lecture 3 by Dr.Marwan Faud,2022

2.PySpark:

Pyspark is swift processing speed, PySpark is dynamic in nature, it is fault tolerant thereby, ensuring zero loss of data. PySpark is a tool in spark which supports RDD with the help of pythons py4j library. In this coursework we are using PySpark Shell because of its libraries.

3.RDD:

Resilient Distributed Dataset is the main data structure of spark. It is a low level object and effectively performs distributed tasks.It is the principal component of Spark. It supports read-only format. As it supports Distributed memory it performs better than MapReduce.

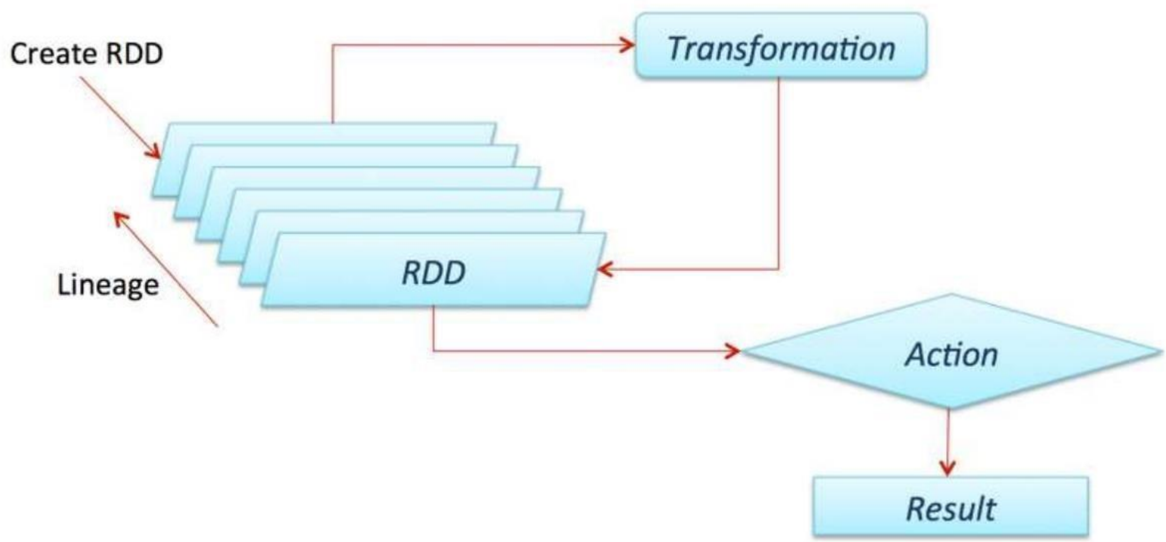


Diagram 13. From lecture 3 by Dr.Marwan Faud,2022

LINEAR REGRESSION MACHINE LEARNING ALGORITHM:

In this coursework we will be using linear regression algorithm to predict the stock market trends. We use linear regression in statistical modelling to compare relationship between two variables it makes the prediction and estimation simpler and easy to interpret on modular level.Machine learning plays a major role in the coursework as these algorithms are performed using different statistical methods. Linear regression parameters like,

- Mean Absolute Error.
- Root Mean Squared Error.
- R-Squared.
- Mean Squared Error.
- Adjusted R-Squared.

THE DATASET:

The dataset for this coureswork is extracted from kaggle. It is a dataset on National stock Exchange- Banking Sector which consists of 15 columns and 41231 rows, This dataset consists of 36 different banking trends in the past 5 years that is from 2016-2021 as mentioned earlier.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
DATE	SYMBOL	SERIES	PREV CLOS	OPEN	HIGH	LOW	LAST	CLOSE	VWAP	VOLUME	TURNOVE	TRADES	DELIVERAI	%DELIVERBLE	
01/01/2016	HDFC	EQ	1263.75	1261	1266.9	1250.65	1257.8	1258.45	1258.39	676161	8.51E+13	13230	#NAME?	0.4559	
04/01/2016	HDFC	EQ	1258.45	1250	1253.9	1212.05	1217.15	1216.7	1227.55	1995329	2.45E+14	78529	1360507	0.6818	
05/01/2016	HDFC	EQ	1216.7	1229.9	1233.45	1206.5	1208.15	1209.4	1219.5	2325929	2.84E+14	109820	1644980	0.7072	
06/01/2016	HDFC	EQ	1209.4	1209.6	1220.75	1202.4	1207.55	1209.3	1210.81	2746330	3.33E+14	96546	2001431	0.7288	
07/01/2016	HDFC	EQ	1209.3	1198.85	1203.55	1175	1176.35	1179.45	1186.35	1780298	2.11E+14	60151	1172564	0.6586	
08/01/2016	HDFC	EQ	1179.45	1189.85	1230	1169	1176.45	1174.4	1175.62	2467149	2.9E+14	79183	1703449	0.6905	
11/01/2016	HDFC	EQ	1174.4	1165	1171	1150.1	1162.35	1161.55	1158.32	2288312	2.65E+14	90184	1541783	0.6738	
12/01/2016	HDFC	EQ	1161.55	1169.4	1169.4	1146.05	1149	1151.85	1152.94	1942208	2.24E+14	60394	1307327	0.6731	
13/01/2016	HDFC	EQ	1151.85	1157	1173.65	1145.35	1166	1167.65	1157.4	1830178	2.12E+14	67493	1013586	0.5538	
14/01/2016	HDFC	EQ	1167.65	1154.9	1167	1147.15	1162	1159.3	1158.69	3250397	3.77E+14	88329	2243147	0.6901	
15/01/2016	HDFC	EQ	1159.3	1160.7	1167.3	1143.15	1148	1149.8	1155.28	1890878	2.18E+14	54238	1215113	0.6426	
18/01/2016	HDFC	EQ	1149.8	1140	1157.7	1125.1	1127.95	1131.8	1139.19	2262744	2.58E+14	48680	1569012	0.6934	
19/01/2016	HDFC	EQ	1131.8	1132.9	1157.15	1130.9	1151	1153	1146.74	1418630	1.63E+14	68378	861562	0.6073	
20/01/2016	HDFC	EQ	1153	1140.25	1198.1	1129.15	1138	1136.65	1136.18	2585069	2.94E+14	82183	1936503	0.7491	
21/01/2016	HDFC	EQ	1136.65	1147.05	1147.7	1121.15	1132	1131.1	1131.36	5451792	6.17E+14	98807	4097208	0.7515	
22/01/2016	HDFC	EQ	1131.1	1140	1163.2	1135.2	1159	1158.45	1154.39	2741456	3.16E+14	90734	1970887	0.7189	
25/01/2016	HDFC	EQ	1158.45	1165	1183.4	1155	1175	1174.6	1175.68	2457396	2.89E+14	105103	1957586	0.7966	
27/01/2016	HDFC	EQ	1174.6	1176.4	1188.45	1162	1162.5	1169.95	1175.94	2496568	2.94E+14	63831	1663988	0.6665	
28/01/2016	HDFC	EQ	1169.95	1150	1159.65	1138	1151.5	1147.75	1149.97	4077193	4.69E+14	145360	3012750	0.7389	

Diagram 14. Dataset in excel

We can notice the 15 attributes and the datatypes here are dates, symbol, series, string, doubles, floats, and integers.

Execution:

Loading data:

To load data in jupyter we require data frameworks from PySpark SQL library. We are importing required libraries and loading data.

```
In [2]: from pyspark.sql import SparkSession
        import pyspark.sql.functions as F
        import pyspark.sql.types as T
```

```
In [3]: spark = SparkSession.builder.appName('DataFrame').getOrCreate()
```

Diagram 15. data is loaded by csv file format

This is how the required data is presented on jupyter notebook, we have selected 10 rows to display by using the command show().

```
In [12]: df_pyspark = spark.read.csv('file:///home/jusel/Documents/NSE_BANKING_SECTOR.csv',header=True,inferSchema=True)
```

```
In [14]: df_pyspark.show(10)
```

```
+-------------------+------+------+----------+-------+-------+-------+-------+-------+-------+-------+-------------
------+------+-----------------+-----------------+
|               DATE|SYMBOL|SERIES|PREV CLOSE|   OPEN|   HIGH|    LOW|   LAST|  CLOSE|   VWAP| VOLUME|           TU
RNOVER|TRADES|DELIVERABLE VOLUME|       %DELIVERBLE|
+-------------------+------+------+----------+-------+-------+-------+-------+-------+-------+-------+-------------
------+------+-----------------+-----------------+
|2016-01-01 00:00:00|  HDFC|    EQ|   1263.75| 1261.0| 1266.9|1250.65| 1257.8|1258.45|1258.39| 676161|     8.5087506
875E13| 13230|           308262|           0.4559|
|2016-01-04 00:00:00|  HDFC|    EQ|   1258.45| 1250.0| 1253.9|1212.05|1217.15| 1216.7|1227.55|1995329|2.449370563550
000...| 78529|          1360507|0.6818000000000001|
|2016-01-05 00:00:00|  HDFC|    EQ|    1216.7| 1229.9|1233.45| 1206.5|1208.15| 1209.4| 1219.5|2325929|    2.83646403
125E14|109820|          1644980|           0.7072|
|2016-01-06 00:00:00|  HDFC|    EQ|    1209.4| 1209.6|1220.75| 1202.4|1207.55| 1209.3|1210.81|2746330|      3.325286
321E14| 96546|          2001431|           0.7288|
|2016-01-07 00:00:00|  HDFC|    EQ|    1209.3|1198.85|1203.55| 1175.0|1176.35|1179.45|1186.35|1780298|    2.11205540
675E14| 60151|          1172564|           0.6586|
|2016-01-08 00:00:00|  HDFC|    EQ|   1179.45|1189.85| 1230.0| 1169.0|1176.45| 1174.4|1175.62|2467149|     2.9004234
473E14| 79183|          1703449|           0.6905|
|2016-01-11 00:00:00|  HDFC|    EQ|    1174.4| 1165.0| 1171.0| 1150.1|1162.35|1161.55|1158.32|2288312|       2.65059
728E14| 90184|          1541783|           0.6738|
|2016-01-12 00:00:00|  HDFC|    EQ|   1161.55| 1169.4| 1169.4|1146.05| 1149.0|1151.85|1152.94|1942208|2.239246422050
000...| 60394|          1307327|           0.6731|
|2016-01-13 00:00:00|  HDFC|    EQ|   1151.85| 1157.0|1173.65|1145.35| 1166.0|1167.65| 1157.4|1830178|     2.1182475
953E14| 67493|          1013586|0.5538000000000001|
|2016-01-14 00:00:00|  HDFC|    EQ|   1167.65| 1154.9| 1167.0|1147.15| 1162.0| 1159.3|1158.69|3250397|    3.76619412
345E14| 88329|          2243147|           0.6901|
+-------------------+------+------+----------+-------+-------+-------+-------+-------+-------+-------+-------------
------+------+-----------------+-----------------+
only showing top 10 rows
```

Diagram 16. 10 rows displayed

By using the count() we have taken count of the rows, and the output is 41231 rows.

```
In [8]: df_pyspark.count()

Out[8]: 41231
```

By using take() command we can take each variable of the data type.

```
df_pyspark.take(1)
```

```
[Row(DATE='2016-01-01', SYMBOL='HDFC', SERIES='EQ', PREV CLOSE=1263.7
5, OPEN=1261.0, HIGH=1266.9, LOW=1250.65, LAST=1257.8, CLOSE=1258.45,
VWAP=1258.39, VOLUME=676161, TURNOVER=85087506875000.0, TRADES=13230,
DELIVERABLE VOLUME=308262, %DELIVERBLE=0.4559)]
```

To show all the columns in the dataset, Columns command is used.

```
# TO SHOW ALL COLUMNS
df_pyspark.columns[:]
```

```
['DATE',
 'SYMBOL',
 'SERIES',
 'PREV CLOSE',
 'OPEN',
 'HIGH',
 'LOW',
 'LAST',
 'CLOSE',
 'VWAP',
 'VOLUME',
 'TURNOVER',
 'TRADES',
 'DELIVERABLE VOLUME',
 '%DELIVERBLE']
```

To check the number of banks in the national stock exchange distinct() command is used. We have 20 banks present in this dataset.

```
In [24]: df_pyspark.select("SYMBOL").distinct().show()

[Stage 13:>                                                     (0 +
1) / 1]

+----------+
|    SYMBOL|
+----------+
| ICICIBANK|
|    AUBANK|
|       CUB|
|   KTKBANK|
|   CSBBANK|
| CENTRALBK|
|       PSB|
|   UCOBANK|
|      IDBI|
|       PNB|
| BANKINDIA|
|      HDFC|
|   J&KBANK|
|       IOB|
|   DCBBANK|
|  SURYODAY|
|  MAHABANK|
|EQUITASBNK|
|  IDFCBANK|
|   RBLBANK|
+----------+
only showing top 20 rows
```

Loading DataSet :

We can load the dataset through RDDs in python by using sc which stands for Spark context.

LOAD DATA WITH RDD

```
In [30]: from pyspark import SparkContext, SparkConf
         rdd1 = sc.textFile('file:///home/jusel/Documents/NSE_BANKING_SECTOR.csv')
         rdd1.take(10)

Out[30]: ['DATE,SYMBOL,SERIES,PREV CLOSE,OPEN,HIGH,LOW,LAST,CLOSE,VWAP,VOLUME,TURNOVER,TRADES,DELIVERABLE VOLUME,%DELIVERBLE
         ',
          '2016-01-01,HDFC,EQ,1263.75,1261.0,1266.9,1250.65,1257.8,1258.45,1258.39,676161,85087506875000.0,13230,308262,0.455
         9',
          '2016-01-04,HDFC,EQ,1258.45,1250.0,1253.9,1212.05,1217.15,1216.7,1227.55,1995329,244937056355000.03,78529,1360507,
         0.6818000000000001',
          '2016-01-05,HDFC,EQ,1216.7,1229.9,1233.45,1206.5,1208.15,1209.4,1219.5,2325929,283646403125000.0,109820,1644980,0.7
         072',
          '2016-01-06,HDFC,EQ,1209.4,1209.6,1220.75,1202.4,1207.55,1209.3,1210.81,2746330,332528632100000.0,96546,2001431,0.7
         288',
          '2016-01-07,HDFC,EQ,1209.3,1198.85,1203.55,1175.0,1176.35,1179.45,1186.35,1780298,211205540675000.0,60151,1172564,
         0.6586',
          '2016-01-08,HDFC,EQ,1179.45,1189.85,1230.0,1169.0,1176.45,1174.4,1175.62,2467149,290042344730000.0,79183,1703449,0.
         6905',
          '2016-01-11,HDFC,EQ,1174.4,1165.0,1171.0,1150.1,1162.35,1161.55,1158.32,2288312,265059728000000.0,90184,1541783,0.6
         738',
          '2016-01-12,HDFC,EQ,1161.55,1169.4,1169.4,1146.05,1149.0,1151.85,1152.94,1942208,223924642205000.03,60394,1307327,
         0.6731',
          '2016-01-13,HDFC,EQ,1151.85,1157.0,1173.65,1145.35,1166.0,1167.65,1157.4,1830178,211824759530000.0,67493,1013586,0.
         5538000000000001']
```

Map Function:

Applies a function to value of a pair RDD without changing the key.

```
#map functions
rdd2.map(lambda line:line.split(',')).take(1)
```

```
[['2016-01-01',
  'HDFC',
  'EQ',
  '1263.75',
  '1261.0',
  '1266.9',
  '1250.65',
  '1257.8',
  '1258.45',
  '1258.39',
  '676161',
  '85087506875000.0',
  '13230',
  '308262',
  '0.4559']]
```

Cleaning the Data:

Drop Duplicates:

dfcount() shows the remaining data after dropDuplicates command is used.

```
df = df_pyspark.dropDuplicates()
```

```
df.groupBy(df.columns)\
.count()\
.where(F.col('count')>1)\
.select(F.sum('count'))\
.show()
```

```
+----------+
|sum(count)|
+----------+
|      null|
+----------+
```

```
df.count()
```

```
41231
```

Drop null values:

We can drop all the null and misiing values in the dataset by using the command drop().

```
##DROPING NULL VALUES
df_pyspark.na.drop().show(5)
```

```
+----------+------+------+----------+-------+-------+-------+-------+-------+-------+-------+--------------------+
------+------------------+------------------+
|      DATE|SYMBOL|SERIES|PREV CLOSE|   OPEN|   HIGH|    LOW|   LAST|  CLOSE|   VWAP| VOLUME|            TURNOVER|
TRADES|DELIVERABLE VOLUME|       %DELIVERBLE|
+----------+------+------+----------+-------+-------+-------+-------+-------+-------+-------+--------------------+
------+------------------+------------------+
|2016-01-01|  HDFC|    EQ|   1263.75| 1261.0| 1266.9|1250.65| 1257.8|1258.45|1258.39| 676161|     8.5087506875E13|
13230|            308262|            0.4559|
|2016-01-04|  HDFC|    EQ|   1258.45| 1250.0| 1253.9|1212.05|1217.15| 1216.7|1227.55|1995329|2.449370563550000...|
78529|           1360507|0.6818000000000001|
|2016-01-05|  HDFC|    EQ|    1216.7| 1229.9|1233.45| 1206.5|1208.15| 1209.4| 1219.5|2325929|    2.83646403125E14|
109820|            1644980|            0.7072|
|2016-01-06|  HDFC|    EQ|    1209.4| 1209.6|1220.75| 1202.4|1207.55| 1209.3|1210.81|2746330|      3.325286321E14|
96546|           2001431|            0.7288|
|2016-01-07|  HDFC|    EQ|    1209.3|1198.85|1203.55| 1175.0|1176.35|1179.45|1186.35|1780298|    2.11205540675E14|
60151|           1172564|            0.6586|
+----------+------+------+----------+-------+-------+-------+-------+-------+-------+-------+--------------------+
------+------------------+------------------+
only showing top 5 rows
```

Filling Missing Values:

Fill missing values using fill() command

```
df_pyspark.na.fill(value='missingvalues', subset=['SYMBOL','TRADES']).show(5)
```

```
+----------+------+----------+-------+-------+-------+-------+-------+-------+-------+--------------------+------+
------------------+------------------+
|      DATE|SYMBOL|PREV CLOSE|   OPEN|   HIGH|    LOW|   LAST|  CLOSE|   VWAP| VOLUME|            TURNOVER|TRADES|
DELIVERABLE VOLUME|       %DELIVERBLE|
+----------+------+----------+-------+-------+-------+-------+-------+-------+-------+--------------------+------+
------------------+------------------+
|2016-01-01|  HDFC|   1263.75| 1261.0| 1266.9|1250.65| 1257.8|1258.45|1258.39| 676161|     8.5087506875E13| 13230|
308262|            0.4559|
|2016-01-04|  HDFC|   1258.45| 1250.0| 1253.9|1212.05|1217.15| 1216.7|1227.55|1995329|2.449370563550000...| 78529|
1360507|0.6818000000000001|
|2016-01-05|  HDFC|    1216.7| 1229.9|1233.45| 1206.5|1208.15| 1209.4| 1219.5|2325929|    2.83646403125E14|109820|
1644980|            0.7072|
|2016-01-06|  HDFC|    1209.4| 1209.6|1220.75| 1202.4|1207.55| 1209.3|1210.81|2746330|      3.325286321E14| 96546|
2001431|            0.7288|
|2016-01-07|  HDFC|    1209.3|1198.85|1203.55| 1175.0|1176.35|1179.45|1186.35|1780298|    2.11205540675E14| 60151|
1172564|            0.6586|
+----------+------+----------+-------+-------+-------+-------+-------+-------+-------+--------------------+------+
------------------+------------------+
only showing top 5 rows
```

Dropping values with unique values:

We will eliminate the values in the columns which has same values.

```
df_pyspark.select('SERIES').show()
```

```
+------+
|SERIES|
+------+
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
|    EQ|
+------+
only showing top 20 rows
```

We are dropping the column using drop()

```
df_pyspark = df_pyspark.drop("SERIES").show(5)
```

```
+----------+------+----------+-------+-------+-------+-------+-------+-------+-------+--------------------+------+
------------------+------------------+
|      DATE|SYMBOL|PREV CLOSE|   OPEN|   HIGH|    LOW|   LAST|  CLOSE|   VWAP| VOLUME|            TURNOVER|TRADES|
DELIVERABLE VOLUME|       %DELIVERBLE|
+----------+------+----------+-------+-------+-------+-------+-------+-------+-------+--------------------+------+
------------------+------------------+
|2016-01-01|  HDFC|   1263.75| 1261.0| 1266.9|1250.65| 1257.8|1258.45|1258.39| 676161|     8.5087506875E13| 13230|
308262|            0.4559|
|2016-01-04|  HDFC|   1258.45| 1250.0| 1253.9|1212.05|1217.15| 1216.7|1227.55|1995329|2.449370563550000...| 78529|
1360507|0.6818000000000001|
|2016-01-05|  HDFC|    1216.7| 1229.9|1233.45| 1206.5|1208.15| 1209.4| 1219.5|2325929|    2.83646403125E14|109820|
1644980|            0.7072|
|2016-01-06|  HDFC|    1209.4| 1209.6|1220.75| 1202.4|1207.55| 1209.3|1210.81|2746330|      3.325286321E14| 96546|
2001431|            0.7288|
|2016-01-07|  HDFC|    1209.3|1198.85|1203.55| 1175.0|1176.35|1179.45|1186.35|1780298|    2.11205540675E14| 60151|
1172564|            0.6586|
+----------+------+----------+-------+-------+-------+-------+-------+-------+-------+--------------------+------+
------------------+------------------+
only showing top 5 rows
```

Imputer Function:

By using this unique function we will fill all null values with mean,median and mode.

```
### To fill the null function with MEAN MEDAIN MODE ##we use imputer function
from pyspark.ml.feature import Imputer
imputer = Imputer(
inputCols=['PREV CLOSE','OPEN','HIGH','LOW','CLOSE','LAST','VWAP'],
outputCols=["{}_imputed".format(c)for c in ['PREV CLOSE','OPEN','HIGH','LOW','CLOSE','LAST','VWAP']]).setStrategy("r
```

```
## ADD IMPUTATION  TO DATAFRAME
imputer.fit(df_pyspark).transform(df_pyspark).show()
```

```
+----------+------+------+----------+-------+-------+-------+-------+-------+-------+-------+--------------------+-
-----+-----------------+-----------------+-----------+-----------+------------+------------+------------+------
-----------+-------------+
|      DATE|SYMBOL|SERIES|PREV CLOSE|   OPEN|   HIGH|    LOW|   LAST|  CLOSE|   VWAP| VOLUME|            TURNOVER|T
RADES|DELIVERABLE VOLUME|       %DELIVERBLE|LOW_imputed|VWAP_imputed|OPEN_imputed|HIGH_imputed|LAST_imputed|PREV CL
OSE_imputed|CLOSE_imputed|
+----------+------+------+----------+-------+-------+-------+-------+-------+-------+-------+--------------------+-
-----+-----------------+-----------------+-----------+-----------+------------+------------+------------+------
-----------+-------------+
|2016-01-01|  HDFC|    EQ|   1263.75| 1261.0| 1266.9|1250.65| 1257.8|1258.45|1258.39| 676161|     8.5087506875E13|
13230|           308262|           0.4559|    1250.65|      1258.39|       1261.0|       1266.9|       1257.8|
1263.75|       1258.45|
|2016-01-04|  HDFC|    EQ|   1258.45| 1250.0| 1253.9|1212.05|1217.15| 1216.7|1227.55|1995329|2.449370563550000...|
78529|          1360507|0.6818000000000001|    1212.05|      1227.55|       1250.0|       1253.9|      1217.15|
1258.45|        1216.7|
|2016-01-05|  HDFC|    EQ|    1216.7| 1229.9|1233.45| 1206.5|1208.15| 1209.4| 1219.5|2325929|     2.83646403125E14|1
09820|          1644980|           0.7072|     1206.5|       1219.5|       1229.9|      1233.45|      1208.15|
1216.7|        1209.4|
|2016-01-06|  HDFC|    EQ|    1209.4| 1209.6|1220.75| 1202.4|1207.55| 1209.3|1210.81|2746330|      3.325286321E14|
```

Describe():

The df_pyspark.describe() function shows the attributes of the data set. We can clearly understand the mean, median ,maximum,minimum and sd (standard deviation) through this.

```
df_pyspark.describe().show()
```

```
+-------+----------+-------+------+----------------+-----------------+----------------+-----------------+-----
-----------+-----------------+----------------+------------------+-------------------+------------------+----
-------------+------------------+
|summary|      DATE| SYMBOL|SERIES|      PREV CLOSE|             OPEN|            HIGH|              LOW|
LAST|           CLOSE|            VWAP|            VOLUME|           TURNOVER|            TRADES|DELIVERABLE
VOLUME|        %DELIVERBLE|
+-------+----------+-------+------+----------------+-----------------+----------------+-----------------+-----
-----------+-----------------+----------------+------------------+-------------------+------------------+----
-------------+------------------+
|  count|     41231|  41231| 41231|           41231|            41231|           41231|            41231|
41231|           41231|           41231|             41231|              41231|             41231|
41231|            41231|
|   mean|      null|   null|  null|291.9627525405654|292.35094710290934|296.5184836652005|287.72344837623086|291.9
936055395213|292.01308845286445|292.1607307608334|1.042650207695666E7|1.953615145271054E14|52218.115786665374| 302
6934.880963353| 0.4154164681914107|
| stddev|      null|   null|  null|452.5410277254348| 452.9678921810755|458.2247568688417| 447.0694317661111|452.7
173431704831| 452.7320642128108|452.6553137088171|2.953972120784208E7|4.038675009566605...| 88510.20733952372| 938
7528.100114819|0.19612223330677533|
|    min|2016-01-01| AUBANK|    EQ|             4.9|             4.95|            4.95|              4.8|
4.9|             4.9|            4.91|              9194|1.681627999999999...|                94|
7392|           0.0201|
|    max|2021-05-28|YESBANK|    EQ|         2860.45|           2871.0|          2896.0|           2838.0|
2861.55|         2860.45|         2867.92|        1264917719|  1.4982219064275E16|           1788274|
787086390|              1.0|
+-------+----------+-------+------+----------------+-----------------+----------------+-----------------+-----
-----------+-----------------+----------------+------------------+-------------------+------------------+----
-------------+------------------+
```

Analyzing the market trends:

Opening market value:

df_pyspark.describe(‘open’).show()

```
df_pyspark.describe('OPEN').show()
```

```
+-------+------------------+
|summary|              OPEN|
+-------+------------------+
|  count|             41231|
|   mean|292.35094710290934|
| stddev| 452.9678921810755|
|    min|              4.95|
|    max|            2871.0|
+-------+------------------+
```

Closing Market trend:

df_pyspark.describe(‘PREV CLOSE’).show()

```
df_pyspark.describe('PREV CLOSE').show()
```

```
+-------+-----------------+
|summary|       PREV CLOSE|
+-------+-----------------+
|  count|            41231|
|   mean|291.9627525405654|
| stddev|452.5410277254348|
|    min|              4.9|
|    max|          2860.45|
+-------+-----------------+
```

Market at its highest:

df_pyspark.describe('HIGH').show()

```
df_pyspark.describe('HIGH').show()
```

```
+-------+-----------------+
|summary|             HIGH|
+-------+-----------------+
|  count|            41231|
|   mean|296.5184836652005|
| stddev|458.2247568688417|
|    min|             4.95|
|    max|           2896.0|
+-------+-----------------+
```

Market at its lowest:

df_pyspark.describe('LAST').show()

```
df_pyspark.describe('LAST').show()
```

```
+-------+-----------------+
|summary|             LAST|
+-------+-----------------+
|  count|            41231|
|   mean|291.9936055395213|
| stddev|452.7173431704831|
|    min|              4.9|
|    max|          2861.55|
+-------+-----------------+
```

Maximum trading Volume:

df_pyspark.describe('TRADES').show()

```
df_pyspark.describe('TRADES').show()
```

```
+-------+------------------+
|summary|            TRADES|
+-------+------------------+
|  count|             41231|
|   mean|52218.115786665374|
| stddev| 88510.20733952372|
|    min|                94|
|    max|           1788274|
+-------+------------------+
```

GroupBy Function:

Groupby function can be used to group or combine data.

Df.groupBy('Symbol').count().show()

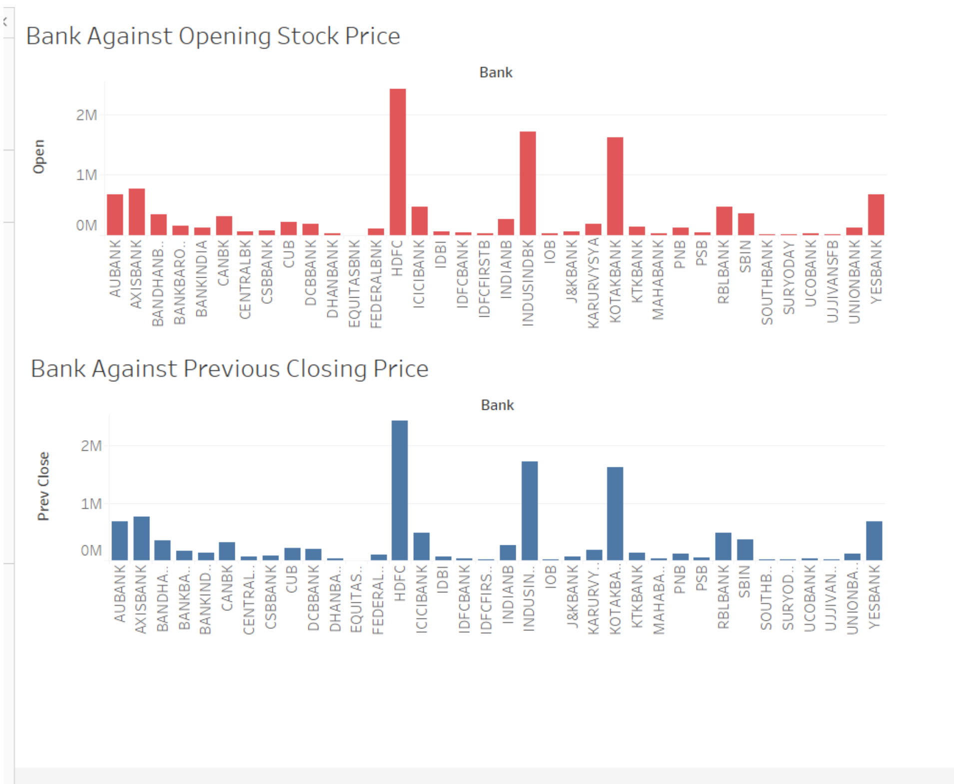
```
#count operation
df.groupBy('SYMBOL').count().show()
```

```
+----------+-----+
|    SYMBOL|count|
+----------+-----+
|  ICICIBANK| 1337|
|     AUBANK|  962|
|        CUB| 1337|
|    KTKBANK| 1337|
|    CSBBANK|  370|
|  CENTRALBK| 1337|
|        PSB| 1337|
|    UCOBANK| 1337|
|       IDBI| 1337|
|        PNB| 1337|
|  BANKINDIA| 1337|
|       HDFC| 1337|
|    J&KBANK| 1337|
|        IOB| 1337|
|    DCBBANK| 1337|
|   SURYODAY|   41|
|    MAHABANK| 1337|
|EQUITASBNK|  141|
|   IDFCBANK|  752|
|    RBLBANK| 1173|
+----------+-----+
only showing top 20 rows
```

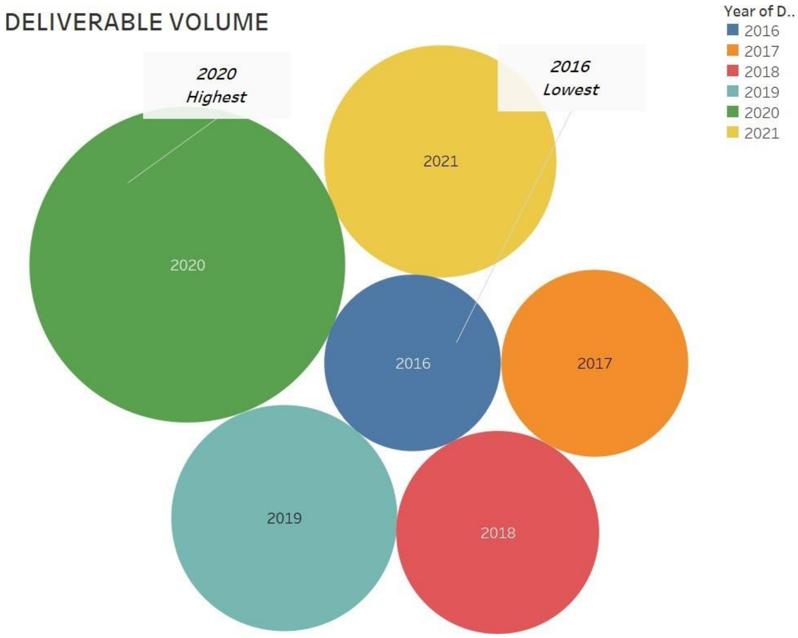
DATA VISUALISATION USING TABLEAU:

Data Visualization is a method to represent information or data graphically by emphasizing on patterns and trends in data . Using tableau is very simple and effective, Data visualization can also be called as the graphical representation of data. Data here can be read easily in the form of graphs, charts and maps. Data visualization techniques and tools provide an easy understanding when it comes to understanding trends and patterns of the data.

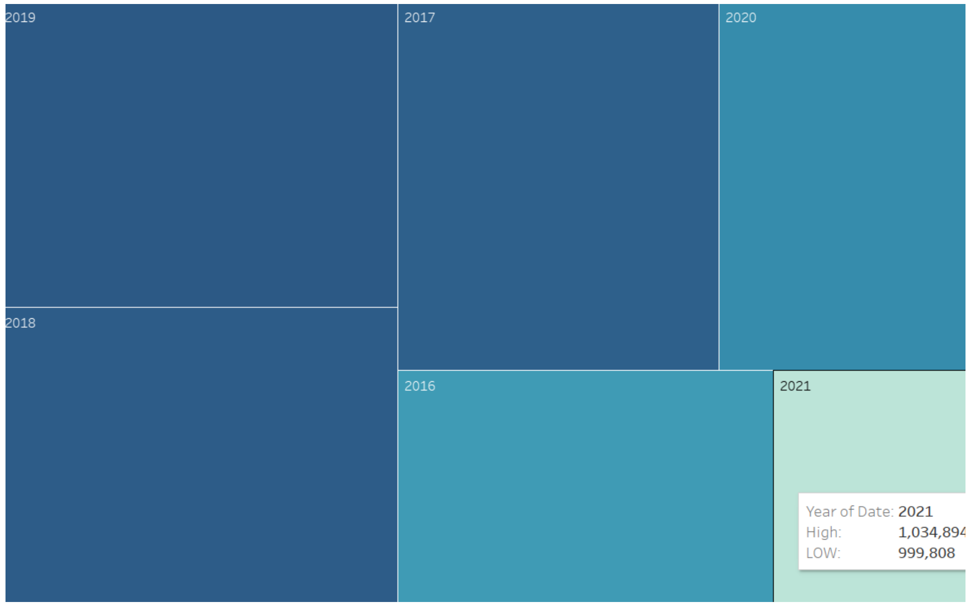
The initial opening and previous day closing price.



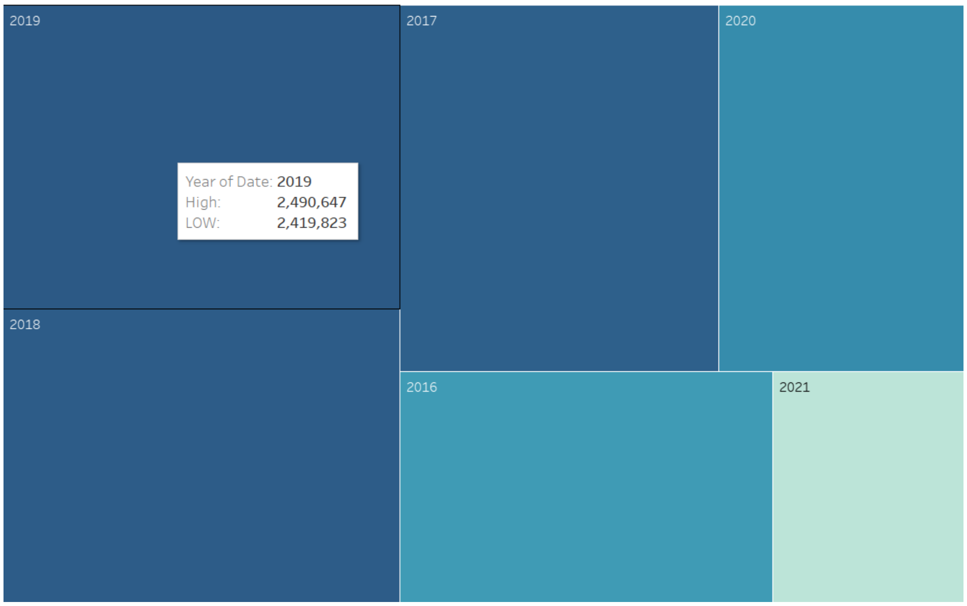
The total volume of 2020 is the highest as compared to that of 2016.



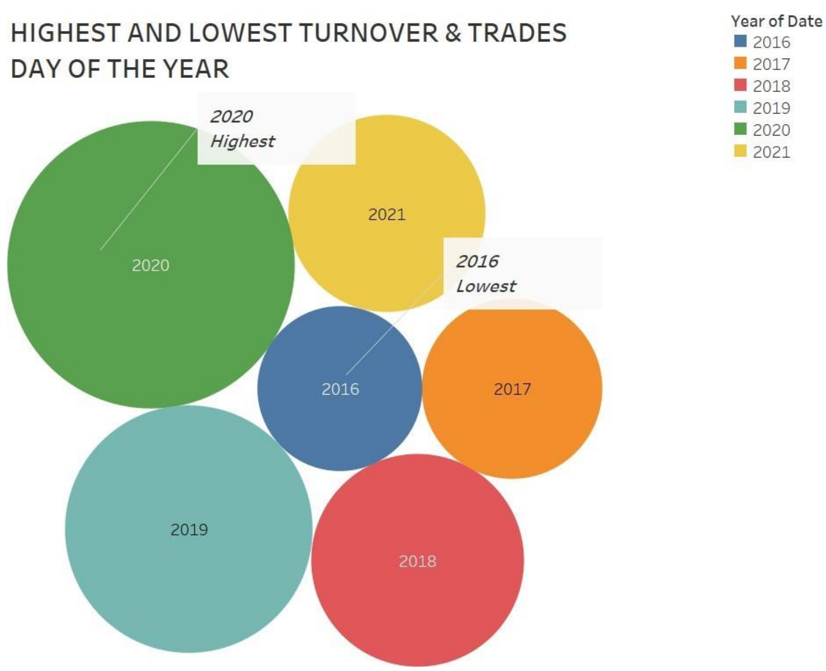
Lowest Market Day in the year is 2021



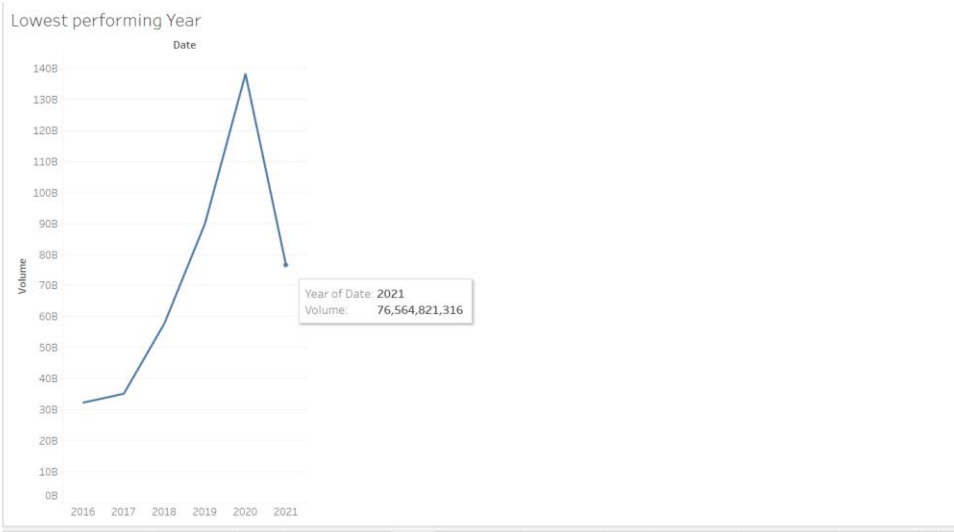
Highest Market day in the year 2019



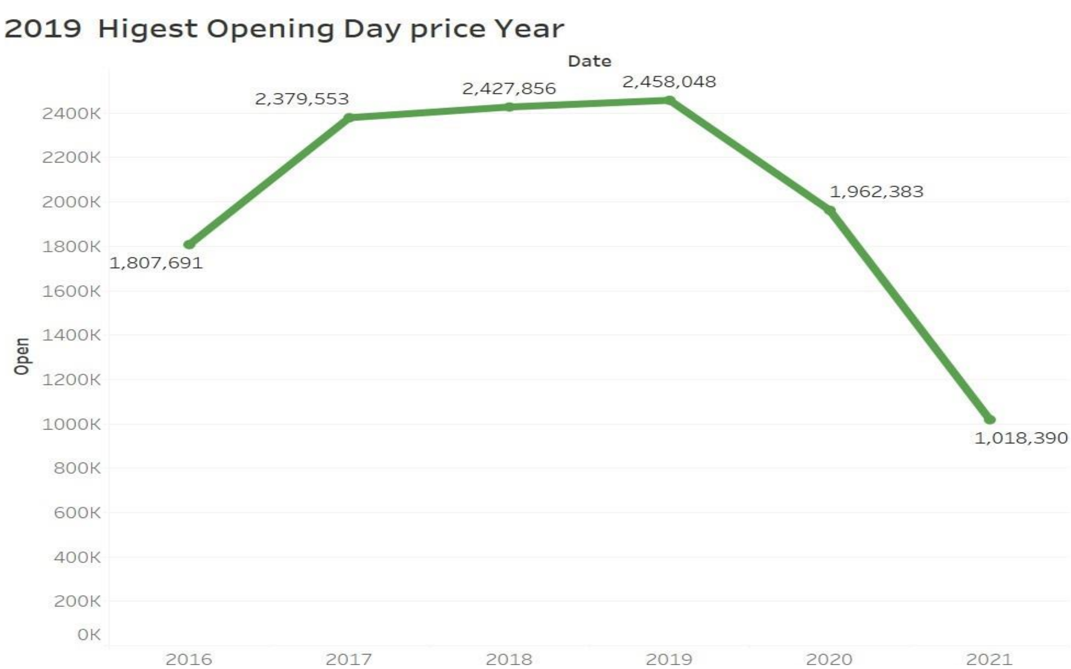
Highest turnover day of the year is 2020 and the lowest is 2016.



2021 is the lowest performing Year.

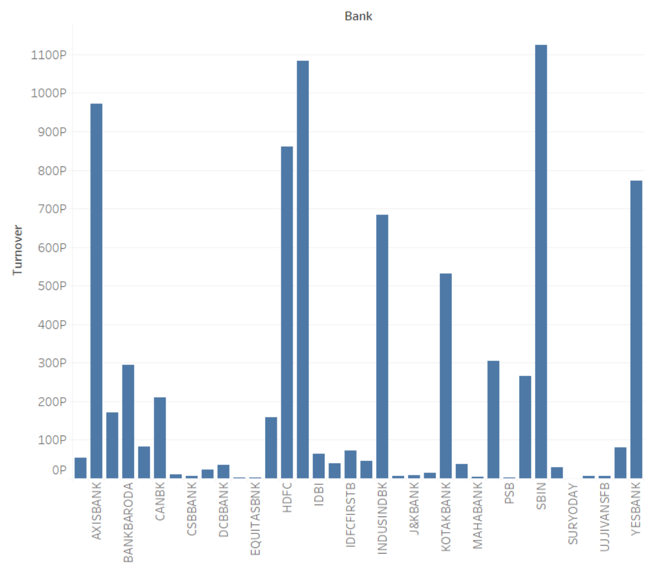


Highest performing year is 2019



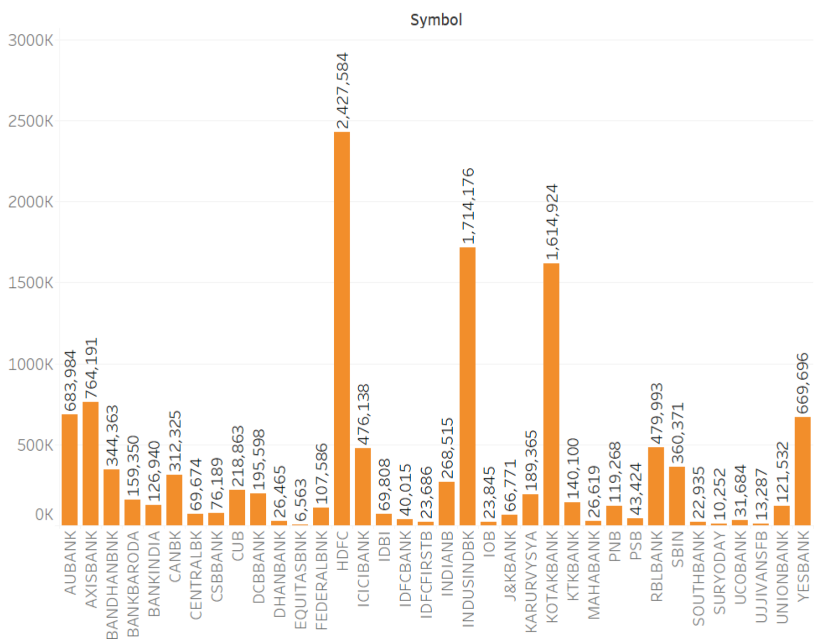
We can notice that the turnover of SBI Bank is the highest.We can conclude that SBI bank has highest turnover as compared to that of other banks.

Banks Against highest turnover

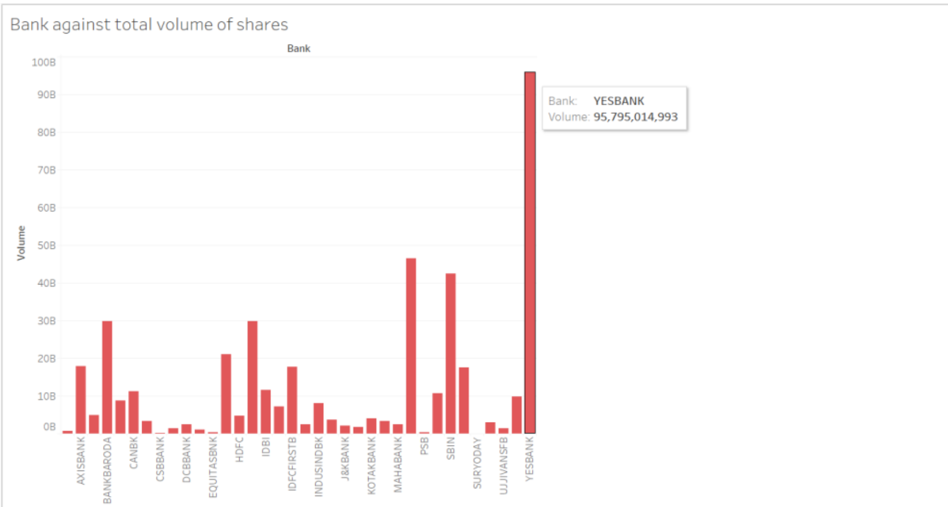


We can understand that HDFC bank has the highest volume weighted Average between 2016-2021

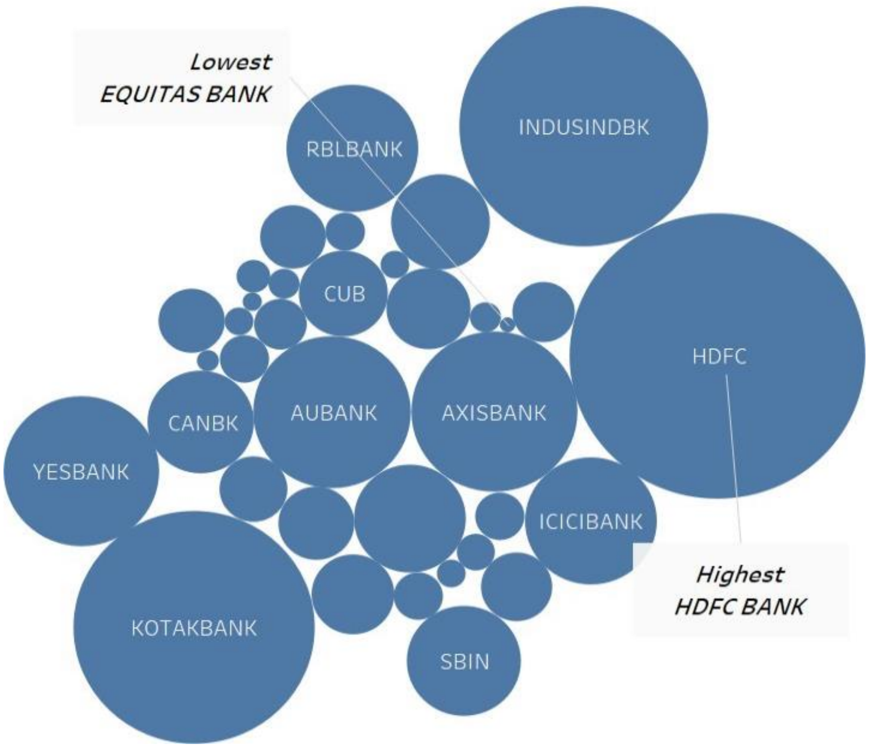
HDFC HASVOLUME WEIGHTED AVERAGE PRICE



We can notice that the highest volume of shares was by YES bank in National Stock Exchange.



Highest and Lowest performing banks are HDFC and Equitas bank simultaneously.



Linear Regression Model:

Machine learning algorithms are part of AI. We use machine learning to induce a huge amount of data into the computer algorithm and train the computer to analyze the data and take data driven decisions, predictions and recommendations based on the input data.

Machine learning algorithms cover Linear Regression, classification, logistic regression, Naïve Bayes, K, Random Forest etc. In this data set Linear regression model will be used to predict outputs.

Training and testing machine learning algorithms:

To execute Linear regression model, we must import assembler vectors, vectors from ML libraries by using

var=['PREVCLOSE','OPEN','HIGH','LOW','LAST','CLOSE','VOLUME','TURNOVER',' TRADES','DELIVERABLE VOLUME','%DELIVERBLE']

assembler = VectorAssembler (inputCols=var, outputCol ='features')

```
train_data, test_data = final_df.randomSplit([0.7,0.3])
```

```
train_data.describe().show()
```

```
+-------+-----------------+
|summary|             VWAP|
+-------+-----------------+
|  count|            28865|
|   mean| 294.0291675038975|
| stddev|453.27546490382133|
|    min|             4.91|
|    max|          2847.59|
+-------+-----------------+
```

```
test_data.describe().show()
```

```
+-------+-----------------+
|summary|             VWAP|
+-------+-----------------+
|  count|            12366|
|   mean|287.79938298560575|
| stddev| 451.192610143522|
|    min|             4.94|
|    max|          2867.92|
+-------+-----------------+
```

Chosen 0.7 and 0.3 respectively

Linear Regression:

```
from pyspark.ml.regression import LinearRegression
```

```
lm = LinearRegression(labelCol='VWAP')
```

```
model = lm.fit(train_data)
```

	Coefficients
PREV CLOSE	-2.685918e-03
OPEN	-3.321131e-02
HIGH	4.307286e-01
LOW	3.630813e-01
LAST	-1.745913e-01
CLOSE	4.159906e-01
VOLUME	2.837304e-09
TURNOVER	2.818528e-16
TRADES	-2.545833e-06
DELIVERABLE VOLUME	-3.658491e-09
%DELIVERBLE	-2.229632e-01

Parameters

```
print("MEAN ABOSULTE ERROR: ", res.meanAbsoluteError)
print("MEAN SQUARE ERROR: ", res.meanSquaredError)
print("ROOT MEAN SQUARE ERROR: ", res.rootMeanSquaredError)
print("R2: ", res.r2)
print("Adj R2: ", res.r2adj)
```

```
MEAN ABOSULTE ERROR:  0.6582458837188474
MEAN SQUARE ERROR:  2.673658497937767
ROOT MEAN SQUARE ERROR:  1.6351325628027127
R2:  0.9999868653924705
Adj R2:  0.9999868536974177
```

R squared= 99%

Adjusted R Squared= 99%

Mean absolute Error=0.65

Mean Squared Error= 2.67

Root Mean Squared error=1.63

Discussion of findings:

Through great understanding and consideration of the problem in the data set I have come to a conclusion that through this coursework we have clearly understood the various performances and trends of banks in india under the national stock exchange. We can understand the various patterns and trends in the data through visualization tool such as tableau. We could understand that between 2016-2021, the lowest market price was in 2021 and the highest was in the year 2019.We noticed that SBI bank has the highest turnover, meanwhile Equitas Bank had the lowest turnover. HDFC has the highest volume weighted average price in the year between 2016 – 2021. The top 3 performing banks are HDFC , Indus and Kotak Banks respectively. HDFC has encountered the lowest market day yet performed well . YES Bank has the highest volume of shares

Linear regression model performed extremely well with the data set it had an R squared and adjacent R squared of 0.99 which equals to 99 % which means higher the R squared , higher the regression. Hence, it is the best algorithm to perform prediction in stock marketing.

We also learnt that the Highest and Lowest performing banks are HDFC and Equitas bank correspondinly . We can notice that the highest volume of shares was by YES bank in the National Stock Exchange. We can understand that HDFC bank has the highest volume weighted Average between 2016-2021and the highest volume of shares was by YES bank in the National Stock Exchange.

Conclusion:

Through statistical tools and visualization, we have been able to solve and predict the problems in the stock marketing- banking sector. We have used tableau exclusively as our data visualization tool. Through this we can conclude that HDFC bank is performing highest, followed by Indus bank and then Kotak Mahindra bank. The banks performing at their lowest are Surya day Bank and Equitas bank. Through this we understand that the investors of HDFC bank will be stable and the investors of Equitas bank will take loss according to the data collected between 2016-2021.

REFERENCES:

- *TechVidvan Tutorial's. Big Data and Machine Learning – Journey as Beautiful as Sunset* *https://techvidvan.com/tutorials/big-data-and-machine-learning*
- *Dr Marwan Faud .(2021). Lecture 3*
- *Smritis. (2019). What is Mean Squared Error, Mean Absolute Error, Root Mean Squared Error and R Squared?.*

- *YouTube: Pyspark with Python*
- *YouTube: Linear regression with PySpark*
- *Apache Spark - RDD - Tutorialspoint*

- *Installation: https://www.youtube.com/watch?v=h7U2mRVM84U*

Appendix:

```
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql import SQLContext
import pyspark.sql.functions as F

import pyspark.sql.types as T

spark = SparkSession.builder.appName('DataFrame').getOrCreate()

spark

sc= SparkContext.getOrCreate()
sqlContext=SQLContext(sc)

df_pyspark =
spark.read.csv('file:///home/jusel/Documents/NSE_BANKING_SECTOR.csv',header=True,inferSchema=
True)

df_pyspark.show()

df_pyspark.count()

df_pyspark.take(1)
```

```
# display columns
df_pyspark.columns

df_pyspark.select("SYMBOL").distinct().show()

# schema.
df_pyspark.printSchema()

df = df_pyspark.dropDuplicates()

df_pyspark.groupBy(df_pyspark.columns)\
.count()\
.where(F.col('count')>1)\
.select(F.sum('count'))\
.show()

df_pyspark.count()

#TRADES COLUMN
df_pyspark.select('TRADES').show()

#MULTIPLE COLUNMNS
df_pyspark.select(['TRADES','SYMBOL']).show()

df_pyspark.dtypes

from pyspark import SparkContext, SparkConf
rdd1 = sc.textFile('file:///home/jusel/Documents/NSE_BANKING_SECTOR.csv')
rdd1.take(10)
```

```
rdd_head= rdd1.first()
rdd2 = rdd1.filter(lambda line:line!=rdd_head)

rdd2.first()

#map functions
rdd2.map(lambda line:line.split(',')).take(1)

df_pyspark.registerTempTable('data_table')
sqlContext.sql('select * from data_table').show(5)

sqlContext.sql('select SYMBOL from data_table').show(1)

sqlContext.sql('select distinct(SYMBOL) from data_table').show()

sqlContext.sql('select max(LAST) from data_table').show()

sqlContext.sql('select min(LAST) from data_table').show()
```

Averages

```
import pyspark.sql.functions as F
avg_rent = df_pyspark.groupby().agg(F.avg('VWAP')).cache()
avg_rent.show()
```

Describe Function

```
df_pyspark.describe().show()

df_pyspark.describe('SYMBOL').show()
```

df_pyspark.describe('TRADES').show()

df_pyspark.describe('OPEN').show()

df_pyspark.describe('CLOSE').show()

df_pyspark.describe('HIGH').show()

df_pyspark.describe('LOW').show()

df_pyspark.describe('VWAP').show()

df_pyspark.describe('PREV CLOSE').show()

df_pyspark.describe('LAST').show()

df_pyspark.describe('VOLUME').show()

df_pyspark.describe('TURNOVER').show()

df_pyspark.describe('DELIVERABLE VOLUME').show()

df_pyspark.describe('%DELIVERBLE').show()

Dropping columns

df_pyspark.select('SERIES').show()

#DROPING NULL VALUES

df.na.drop()

```
df.na.fill(value='missingvalues', subset=['SYMBOL','TRADES']).show(5)

#To fill the null function with MEAN MEDAIN MODE
from pyspark.ml.feature import Imputer
imputer = Imputer(
inputCols=['PREV CLOSE','OPEN','HIGH','LOW','CLOSE','LAST','VWAP'],
outputCols=["{}_imputed".format(c)for c in ['PREV CLOSE','OPEN','HIGH','LOW','CLOSE','LAST','VWAP']]).setStrategy("mean")

# IMPUTATION
imputer.fit(df).transform(df).show(5)

# groupby
df.groupBy('SYMBOL').sum().show()

#Avg values of dataset
df.groupBy('SYMBOL').mean().show()

#count operation
df.groupBy('SYMBOL').count().show()v

#max operation
df.groupBy('SYMBOL').max().show()

df.cache()

import pandas as pd

df_pandas=df.toPandas()

df_pandas.head(15)
```

Machine Learning

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

coef_var =['PREV CLOSE','OPEN','HIGH','LOW','LAST','CLOSE','VOLUME','TURNOVER','TRADES','DELIVERABLE VOLUME','%DELIVERBLE']
assembler = VectorAssembler(inputCols=coef_var,
                            outputCol ='features')

output = assembler.transform(df)

final_df=output.select('features','VWAP')

train_data, test_data = final_df.randomSplit([0.7,0.3])

train_data.describe().show()

test_data.describe().show()

print(f"Train set length: {train_data.count()} records")
print(f"Test set length:{test_data.count()} records")
```

Linear Regression Model

```
from pyspark.ml.regression import LinearRegression

lm = LinearRegression(labelCol='VWAP')

model = lm.fit(train_data)

import pandas as pd

pd.DataFrame({"Coefficients":model.coefficients}, index = coef_var)

res = model.evaluate(test_data)
```

```
res.residuals.show()

res.residuals.show()

print("MEAN ABOSULTE ERROR: ", res.meanAbsoluteError)
print("MEAN SQUARE ERROR: ", res.meanSquaredError)
print("ROOT MEAN SQUARE ERROR: ", res.rootMeanSquaredError)
print("R2: ", res.r2)
print("Adj R2: ", res.r2adj)
```