STOCK PRICE ANALYSIS AND PREDICTION AWS

AIM :

To load the Amazon Stock Price dataset to HDFS, implement suitable queries in PIG and HIVE to analyse the Amazon Stock Price dataset and make predictions on the stock market using a appropriate Machine Learning Model to forecast the stock values.

MACHINE LEARNING MODEL USED:

- The appropriate Machine Learning Model for Stock Prediction is **"The long short term memory - LSTM"**.
- We will use the Long Short-Term Memory(LSTM) method to create a Machine Learning model to forecast Microsoft Corporation stock values. They are used to make minor changes to the information by multiplying and adding. Long-term memory (LSTM) is a deep learning artificial recurrent neural network (RNN) architecture.
- Unlike traditional feed-forward neural networks, LSTM has feedback connections. It can handle single data points (such as pictures) as well as full data sequences (such as speech or video).

PIG INSTALLATION :

PROCEDURE :

1. Open Hadoop from Virtual box.

2. Pig can be downloaded in 2 ways.
   - Go to https://dlcdn.apache.org/pig/ and download pig 16 version.
   - In terminal type wget http://www-us.apache.org/dist/pig/pig-0.16.0/pig-0.16.0.tar.gz

3. After download completion, go to terminal and type the following command to unzip Pig :

   tar xvzf pig-0.16.0.tar.gz

4. To change the working directory to pig-0.16.0 folder use the following command :

   cd pig-0.16.0

5. Type the following command to hold pig files :

   sudo mkdir-p /usr/local/pignew

6. Move all files from pig-0.16.0 folder to /usr/local/pignew folder :

sudo mv * /usr/local/pignew

7. Open .bashrc file with the below command :

   nano .bashrc
8. Add the following to .bashrc file and save the file :

   #PIG VARIABLES

   export PIG_HOME=/usr/local/pignew

   export PATH=$PATH:$PIG_HOME/bin

   export PIG_CLASSPATH=$PIG_HOME/conf:$HADOOP_INSTAL/etc/Hadoop/bin

   export PIG_CONF_DIR=$PIG_HOME/conf

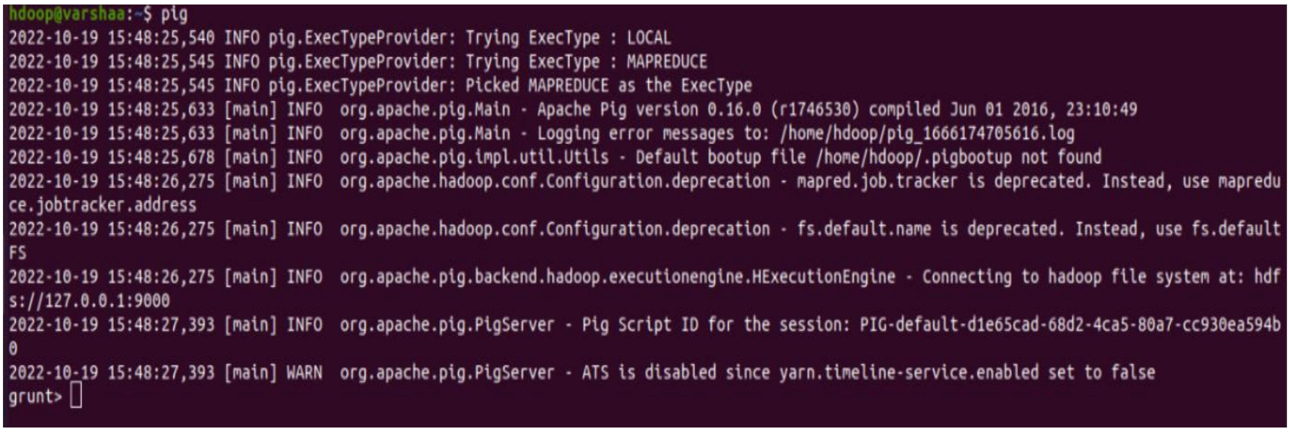
   export PIG_CLASSPATH=$PIG_CONF_DIR

   #PIG VARIABLES END
9. Use the following command to check whether the .bashrc file is updated :

   source .bashrc

10. Installation is completed ,Pig can be started by typing **pig** in the terminal.

11. Pig will load all the functionalities.
12. grunt> would be displayed in terminal ,this indicates the successful installation of pig.



13. Now the pig is ready to use.

HIVE INSTALLATION :

PROCEDURE :

1. Open Hadoop from Virtual box.

2. Hive can be downloaded in 2 ways.
   - Go to https://dlcdn.apache.org/hive/ and download hive 3.1.2 version.

- In terminal type https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz

3. After download completion, go to terminal and type the following command to unzip Hive :

    tar xvzf pig-0.16.0.tar.gz

4. Open .bashrc file with the below command :

    source nano .bashrc

5. Add the following to .bashrc file and save the file to set the path :

    export HIVE_HOME= /home/hdoop/apache-hive-3.1.2-bin
    export PATH=$PATH:$HIVE_HOME/bin

6. Use the following command to source .bashrc to reflect the changes :

    source ~/ .bashrc

7. To specify Hadoop path in configuration file :

    sudo nano $HIVE_HOME/bin/hive-config.sh and paste the below line
    export HADOOP_HOME=/home/hdoop/hadoop-3.2.1

8. To create hive directories in HDFS run the below commands one by one :

    hdfs dfs -mkdir /tmp

    hdfs dfs -chmod g+w /tmp

    hdfs dfs -mkdir -p /user/hive/warehouse

    hdfs dfs -chmod g+w /user/hive/warehouse

9. To specify the dataset used in hive use the following command :

    schematool -initAchema -dbType database

10. If u got error such as multiple bindings for ana explanation use the below command :

    rm $HIVE_HOME/lib/guava-19.0.jar

    $HADOOP_HOME/share/hadoop/hdfs/lib/guava-27.0-jre.jar
    $HIVE_HOME/lib/

11. Initialization of hive is completed ,Hive can be started by typing **hive** in the terminal.

12. Hive will load all the functionalities.
13. hive> would be displayed in terminal ,this indicates the successful installation of pig.

```
hdoop@amir-sukail:~$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hdoop/apache-hive-3.1.2-bin/lib/log4j-sl
f4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hdoop/hadoop-3.3.4/share/hadoop/common/l
ib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = bd16e599-1d59-43fc-a0ae-331d421c0b96

Logging initialized using configuration in jar:file:/home/hdoop/apache-hive-3.1.
2-bin/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true
Hive Session ID = d7c838a8-42f5-40d5-8020-5b287da0473e
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versio
ns. Consider using a different execution engine (i.e. spark, tez) or using Hive
1.X releases.
hive>
```

14. Now the pig is ready to use.

PYSPARK INSTALLATION :

PROCEDURE :

1. Make sure you have java installed in your machine and check your java version using 'java --version' command.
2. Head over to the Spark homepage.
3. Select the Spark release and package type as following and download the .tgz file.
4. Save the file to your local machine and click 'Ok'.
5. Open your terminal and go to the recently downloaded file.
6. Let's extract the file using the following command.
   tar –xvzf spark-2.4.6-bin-hadoop2.7.tgz
7. After extracting the file, the new file is created and shown using the list('ls') command.
8. Lets Configure the environment variable in Hadoop.
9. Let's open the 'bashrc' file using 'vim editor' by the command 'vim ~/.bashrc'.
10. Provide the following information according to your suitable path on your computer. In my case, the following were the required path to my Spark location, Python path, and Java path. Also, first press 'Esc' and then type ":wq" to save and exit from vim.
    export SPARK_HOME=~/Downloads/spark-2.4.6-bin-hadoop2.7
    export PATH=$PATH:$SPARK_HOME/bin
    export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH
    export PYSPARK_PYTHON=python3
    export PATH=$PATH:$JAVA_HOME/jre/bin
11. To make a final change, save, and exit. This results in accessing the pyspark command everywhere in the directory.
    Source ~/.bashrc
12. Open pyspark using 'pyspark' command, and the final message will be shown as below.

```
hdoop@varshaa-dilli:~$ pyspark
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/11/08 14:28:29 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.3.1
      /_/

Using Python version 3.8.10 (default, Jun 22 2022 20:18:18)
Spark context Web UI available at http://varshaa-dilli:4040
Spark context available as 'sc' (master = local[*], app id = local-1667897912488).
SparkSession available as 'spark'.
>>>
```

13. >>> would be displayed in terminal which indicates the successful installation of Pyspark.
14. Now the Pyspark is ready to use.

PIG QUERIES :

To load the dataset to HDFS :

```
hdoop@varshaa:~$ cd Desktop
hdoop@varshaa:~/Desktop$ mkdir BigData
hdoop@varshaa:~/Desktop$ mkdir BigData/Input
hdoop@varshaa:~/Desktop$ mkdir BigData/sample
hdoop@varshaa:~/Desktop$ cd
hdoop@varshaa:~$ export HADOOP_CLASSPATH=$(hadoop classpath)
hdoop@varshaa:~$ echo $HADOOP_CLASSPATH
/home/hdoop/hadoop-3.2.4/etc/hadoop:/home/hdoop/hadoop-3.2.4/share/hadoop/common/lib/*:/home/hdoop/hadoop-3.2.4/share/hadoop/common/*:/home/hdoop/hadoop-3.2.4/share/hadoop/hdfs:/home/hdoop/hadoop-3.2.4/share/hadoop/hdfs/lib/*:/home/hdoop/hadoop-3.2.4/share/hadoop/hdfs/*:/home/hdoop/hadoop-3.2.4/share/hadoop/mapreduce/lib/*:/home/hdoop/hadoop-3.2.4/share/hadoop/mapreduce/*:/home/hdoop/hadoop-3.2.4/share/hadoop/yarn:/home/hdoop/hadoop-3.2.4/share/hadoop/yarn/lib/*:/home/hdoop/hadoop-3.2.4/share/hadoop/yarn/*
```

```
hdoop@varshaa:~$ hadoop fs -mkdir /BigData_AWS
hdoop@varshaa:~$ hadoop fs -mkdir /BigData_AWS/Project
```

```
hdoop@varshaa:~$ hadoop fs -put '/home/hdoop/Desktop/BigData/Input/AWS_Project.txt' /BigData_AWS/Project
```

To load the dataset in PIG :

Loading data with datatype specification :

```
grunt> aws = LOAD '/AWS_Project/Input/AWS_Project.txt' USING PigStorage(',') AS (Date:datetime,Open:double,High:double,Low:double,Close:double,AdjClose:double,Volume:int);
2022-10-18 00:38:33,842 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
```

Loading data without datatype specification :

```
grunt> aws = LOAD '/BigData_AWS/Project/AWS_Pro.txt' USING PigStorage(',') AS (Date,Open,High,Low,Close,AdjClose,Volume);
2022-10-18 12:31:26,434 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
grunt>
```

To display the datatype of the columns in dataset :

describe aws;

Output :

```
aws: {Date: datetime,Open: double,High: double,Low: double,Close: double,AdjClose: double,Volume: int}
```

To display first 5 rows of the dataset :

AWS = limit aws 5;

dump AWS;

Output :

```
(,,,,,,)
(1997-05-15T00:00:00.000+05:30,2.4375,2.5,1.927083,1.958333,1.958333,72156000)
(1997-05-16T00:00:00.000+05:30,1.96875,1.979167,1.708333,1.729167,1.729167,14700000)
(1997-05-19T00:00:00.000+05:30,1.760417,1.770833,1.625,1.708333,1.708333,6106800)
(1997-05-20T00:00:00.000+05:30,1.729167,1.75,1.635417,1.635417,1.635417,5467200)
```

To print the total value of volume :

aws_grp = GROUP aws ALL;

result = FOREACH aws_grp GENERATE SUM(aws.Voulme);

dump result;

Output :

```
(4.5110057291E10)
grunt>
```

To print the maximum volume :

aws_grp = GROUP aws ALL;

result = FOREACH aws_grp GENERATE MAX(aws.Volume);

dump result;

Output :

```
(1.043292E8)
grunt>
```

To print the minimum volume :

aws_grp = GROUP aws ALL;

result = FOREACH aws_grp GENERATE MIN(aws.Volume);

dump result;

```
(487200.0)
grunt>
```

To illustrate table :

illustrate aws;

Output :

```
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
| aws     | Date:bytearray     | Open:bytearray     | High:bytearray     | Low:bytearray     | Close:bytearray     | AdjClose:bytearray     |
Volume:bytearray     |
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
|         | 2003-01-31         | 21.639999          | 22.27              | 21.559999         | 21.85               | 21.85                  |
6262500              |
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

To give explanation about the table :

explain aws;

Output :

```
# New Logical Plan:
#-----------------------------------------------
aws: (Name: LOStore Schema: Date#179:bytearray,Open#180:bytearray,High#181:bytearray,Low#182:bytearray,Close#183:bytearray,AdjClose#184:bytear
ray,Volume#185:bytearray)
|
|---aws: (Name: LOForEach Schema: Date#179:bytearray,Open#180:bytearray,High#181:bytearray,Low#182:bytearray,Close#183:bytearray,AdjClose#184:
bytearray,Volume#185:bytearray)
    |   |
    |   (Name: LOGenerate[false,false,false,false,false,false,false] Schema: Date#179:bytearray,Open#180:bytearray,High#181:bytearray,Low#182:
bytearray,Close#183:bytearray,AdjClose#184:bytearray,Volume#185:bytearray)ColumnPrune:OutputUids=[179, 180, 181, 182, 183, 184, 185]ColumnPrun
e:InputUids=[179, 180, 181, 182, 183, 184, 185]
    |   |   |
    |   |   Date:(Name: Project Type: bytearray Uid: 179 Input: 0 Column: (*))
    |   |   |
    |   |   Open:(Name: Project Type: bytearray Uid: 180 Input: 1 Column: (*))
    |   |   |
    |   |   High:(Name: Project Type: bytearray Uid: 181 Input: 2 Column: (*))
    |   |   |
    |   |   Low:(Name: Project Type: bytearray Uid: 182 Input: 3 Column: (*))
    |   |   |
    |   |   Close:(Name: Project Type: bytearray Uid: 183 Input: 4 Column: (*))
```

```
    |   |   |
    |   |   Volume:(Name: Project Type: bytearray Uid: 185 Input: 6 Column: (*))
    |   |
    |   |---(Name: LOInnerLoad[0] Schema: Date#179:bytearray)
    |   |
    |   |---(Name: LOInnerLoad[1] Schema: Open#180:bytearray)
    |   |
    |   |---(Name: LOInnerLoad[2] Schema: High#181:bytearray)
    |   |
    |   |---(Name: LOInnerLoad[3] Schema: Low#182:bytearray)
    |   |
    |   |---(Name: LOInnerLoad[4] Schema: Close#183:bytearray)
    |   |
    |   |---(Name: LOInnerLoad[5] Schema: AdjClose#184:bytearray)
    |   |
    |   |---(Name: LOInnerLoad[6] Schema: Volume#185:bytearray)
    |
    |---aws: (Name: LOLoad Schema: Date#179:bytearray,Open#180:bytearray,High#181:bytearray,Low#182:bytearray,Close#183:bytearray,AdjClose#184
:bytearray,Volume#185:bytearray)RequiredFields:null
2022-10-19 17:21:52,142 [main] INFO  org.apache.pig.impl.util.SpillableMemoryManager - Selected heap (PS Old Gen) of size 699400192 to monitor
. collectionUsageThreshold = 489580128, usageThreshold = 489580128
#-----------------------------------------------
# Physical Plan:
#-----------------------------------------------
aws: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-16
|
|---aws: New For Each(false,false,false,false,false,false,false)[bag] - scope-15
    |   |
    |   Project[bytearray][0] - scope-1
    |   |
    |   Project[bytearray][1] - scope-3
```

```
|---aws: New For Each(false,false,false,false,false,false,false)[bag] - scope-15
    |   |
    |   Project[bytearray][0] - scope-1
    |   |
    |   Project[bytearray][1] - scope-3
    |   |
    |   Project[bytearray][2] - scope-5
    |   |
    |   Project[bytearray][3] - scope-7
    |   |
    |   Project[bytearray][4] - scope-9
    |   |
    |   Project[bytearray][5] - scope-11
    |   |
    |   Project[bytearray][6] - scope-13
    |
    |---aws: Load(/BigData_AWS/Project/AWS_Pro.txt:PigStorage(',')) - scope-0

2022-10-19 17:21:52,227 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation threshold: 1
00 optimistic? false
2022-10-19 17:21:52,246 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size before op
timization: 1
2022-10-19 17:21:52,246 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size after opt
imization: 1
#--------------------------------------------------
# Map Reduce Plan
#--------------------------------------------------
MapReduce node scope-17
Map Plan
aws: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-16
|
```

```
2022-10-19 17:21:52,246 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size before op
timization: 1
2022-10-19 17:21:52,246 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size after opt
imization: 1
#--------------------------------------------------
# Map Reduce Plan
#--------------------------------------------------
MapReduce node scope-17
Map Plan
aws: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-16
|
|---aws: New For Each(false,false,false,false,false,false,false)[bag] - scope-15
    |   |
    |   Project[bytearray][0] - scope-1
    |   |
    |   Project[bytearray][1] - scope-3
    |   |
    |   Project[bytearray][2] - scope-5
    |   |
    |   Project[bytearray][3] - scope-7
    |   |
    |   Project[bytearray][4] - scope-9
    |   |
    |   Project[bytearray][5] - scope-11
    |   |
    |   Project[bytearray][6] - scope-13
    |
    |---aws: Load(/BigData_AWS/Project/AWS_Pro.txt:PigStorage(',')) - scope-0--------
Global sort: false
----------------
```

<u>To generate Date :</u>

a = FOREACH aws GENERATE Date;

b = FOREACH a GENERATE Date ,GetYear(Date);

dump b;

Output :

```
(1997-05-15,1997)
(1997-05-16,1997)
(1997-05-19,1997)
(1997-05-20,1997)
(1997-05-21,1997)
(1997-05-22,1997)
(1997-05-23,1997)
(1997-05-27,1997)
(1997-05-28,1997)
(1997-05-29,1997)
(1997-05-30,1997)
(1997-06-02,1997)
(1997-06-03,1997)
(1997-06-04,1997)
(1997-06-05,1997)
(1997-06-06,1997)
(1997-06-09,1997)
(1997-06-10,1997)
(1997-06-11,1997)
(1997-06-12,1997)
(1997-06-13,1997)
(1997-06-16,1997)
```

```
(2021-10-07,2021)
(2021-10-08,2021)
(2021-10-11,2021)
(2021-10-12,2021)
(2021-10-13,2021)
(2021-10-14,2021)
(2021-10-15,2021)
(2021-10-18,2021)
(2021-10-19,2021)
(2021-10-20,2021)
(2021-10-21,2021)
(2021-10-22,2021)
(2021-10-25,2021)
(2021-10-26,2021)
(2021-10-27,2021)
```

To filter a particular date :

dt_filter = filter aws by Date==’2021-10-20’;

dump dt_filter;

Output :

```
(2021-10-20,3452.659912,3462.860107,3400.370117,3415.060059,3415.060059,2139800)
```

To display the unique years :

dt = Foreach aws generate GetYear(Date);

unique_dt = distinct dt;

dump unique_dt;

Output :

```
(1997)
(1998)
(1999)
(2000)
(2001)
(2002)
(2003)
(2004)
(2005)
(2006)
(2007)
(2008)
(2009)
(2010)
(2011)
(2012)
(2013)
(2014)
(2015)
(2016)
(2017)
(2018)
(2019)
(2020)
(2021)
grunt>
```

To display the ceil values of High and Low stock price:

math_fn = FOREACH aws GENERATE High, CEIL(High), Low, CEIL(Low);

result = limit math_fn 5;

dump result;

Output:

```
(2.5,3.0,1.927083,2.0)
(1.979167,2.0,1.708333,2.0)
(1.770833,2.0,1.625,2.0)
(1.75,2.0,1.635417,2.0)
(1.645833,2.0,1.375,2.0)
grunt>
```

To display the floor values of Open and Close stock price:

math_fn = FOREACH aws GENERATE Open, CEIL(Open), Close, CEIL(Close);

result = limit math_fn 5;

dump result;

Output:

```
(2.4375,2.0,1.958333,1.0)
(1.96875,1.0,1.729167,1.0)
(1.760417,1.0,1.708333,1.0)
(1.729167,1.0,1.635417,1.0)
(1.635417,1.0,1.427083,1.0)
grunt>
```

To order the Open price in Descending order:

ordering = ORDER aws BY Open DESC;

order_limit = limit ordering 5;

Dump order_limit;

Output:

```
(2017-08-03,999.469971,999.5,984.590027,986.919983,986.919983,3255800)
(2017-06-02,998.98999,1008.47998,995.669983,1006.72998,1006.72998,3752300)
(2017-06-21,998.700012,1002.719971,992.650024,1002.22998,1002.22998,2922500)
(2017-06-01,998.590027,998.98999,991.369995,995.950012,995.950012,2454800)
(2017-06-20,998,1004.880005,992.02002,992.590027,992.590027,4076800)
grunt>
```

To order the Close price in Aescending order:

ordering = ORDER aws BY Close ASC;

order_limit = limit ordering 5;

Dump order_limit;

Output:

```
(1997-05-22,1.4375,1.447917,1.3125,1.395833,1.395833,11776800)
(1997-06-04,1.479167,1.489583,1.395833,1.416667,1.416667,3080400)
(1997-05-21,1.635417,1.645833,1.375,1.427083,1.427083,18853200)
(1997-06-03,1.53125,1.53125,1.479167,1.479167,1.479167,1183200)
(1997-06-27,1.515625,1.515625,1.479167,1.489583,1.489583,1188000)
grunt>
```

HIVE QUERIES :

Create Table :

create table aws(dte string,open double,high double,low double,close double,adj double,volume double)

row format delimited fields terminated by ‘,’ stored as textfile tblproperties(“skip.header.line.count=”1”) ;

Output :

```
OK
Time taken: 1.327 seconds
```

To load the data :

load data inpath ‘/Project/Input/aws.txt’ overwrite into table aws;

Output :

```
Loading data to table default.aws
OK
Time taken: 1.044 seconds
```

To view databases :

show databases;

Output :

```
OK
aws
default
name
Time taken: 0.207 seconds, Fetched: 3 row(s)
```

To view tables :

Show tables;

Output :

```
OK
aws
Time taken: 0.07 seconds, Fetched: 1 row(s)
```

To describe the table :

describe aws;

Output :

```
OK
d                       string
open                    double
high                    double
low                     double
close                   double
adj                     double
volume                  bigint
Time taken: 0.463 seconds, Fetched: 7 row(s)
```

To alter table name :

alter table amazon rename to aws;

Output :

```
OK
Time taken: 0.142 seconds
```

To alter column name :

alter table aws

change column dt dte string;

Output :

```
OK
Time taken: 0.153 seconds
```

To select all rows :

Select * from aws;

Output :

```
2021-10-13      3269.709961     3288.379883     3261.090088     3284.280029     3
284.280029      2420100
2021-10-14      3302.449951     3312.600098     3290.780029     3299.860107     3
299.860107      2109500
2021-10-15      3311.419922     3410.419922     3304.0  3409.02002      3409.020
02      5175100
2021-10-18      3388.360107     3449.169922     3385.100098     3446.73999      3
446.73999       3174100
2021-10-19      3434.290039     3454.689941     3422.0  3444.149902     3444.149
902     2386100
2021-10-20      3452.659912     3462.860107     3400.370117     3415.060059     3
415.060059      2139800
2021-10-21      3414.25 3440.280029     3403.0  3435.01001      3435.01001      1
881400
2021-10-22      3421.0  3429.840088     3331.300049     3335.550049     3335.550
049     3133800
2021-10-25      3335.0  3347.800049     3297.699951     3320.370117     3320.370
117     2226000
2021-10-26      3349.51001      3416.120117     3343.97998      3376.070068     3
376.070068      2693700
2021-10-27      3388.0  3412.0  3371.453369     3396.189941     3396.189941     1
080291
Time taken: 0.186 seconds, Fetched: 6155 row(s)
```

To print the open price in ascending order :

select * from aws order by open;

Output :

```
77.360107       3845900
2021-07-14      3708.850098     3717.659912     3660.830078     3681.679932     36
81.679932       3296600
2021-07-07      3717.379883     3734.199951     3678.909912     3696.580078     36
96.580078       5328100
2021-07-09      3722.52002      3748.0  3693.399902     3719.340088     3719.3400
88      3748200
2021-07-12      3744.0  3757.290039     3696.790039     3718.550049     3718.5500
49      2571600
Time taken: 49.512 seconds, Fetched: 6155 row(s)
```

To print number of unique count in the volume column :

select volume, count(*) from aws group by volume;

Output :

```
OK
487200  1
571200  1
574800  1
590400  1
591600  1
620400  1
624000  1
693600  1
712800  1
722400  1
732000  1
751200  1
780000  1
814800  1
881300  1
913200  1
984400  1
999600  1
1003200 2
1041600 1
```

To sort volume in ascending order and display the corresponding date :

select dte, volume from aws sort by volume;

Output :

```
OK
1997-12-26      487200
1997-08-12      571200
1997-07-21      574800
1997-08-13      590400
1997-06-02      591600
1997-07-25      620400
1997-08-21      624000
1997-06-13      693600
1997-08-22      712800
1997-08-29      722400
1997-09-02      732000
1997-06-24      751200
1997-07-18      780000
1997-10-13      814800
2019-12-24      881300
1997-06-16      913200
2012-12-24      984400
1997-08-20      999600
1997-08-19      1003200
1997-06-19      1003200
```

To display maximum opening price :

select max(open) as MaximumOpenPrice

from aws;

Output :

```
OK
3744.0
Time taken: 26.2 seconds, Fetched: 1 row(s)
```

To display minimum opening price :

select min(open) as MinimumOpenPrice

from aws;

Output :

```
OK
1.40625
Time taken: 30.711 seconds, Fetched: 1 row(s)
```

To display the maximum volume in each year :

select year(dte), max(volume)

from aws

group by year(dte);

Output :

```
2000    51838200
2001    50689200
2002    56645900
2003    39972400
2004    35927200
2005    60518600
2006    76985200
2007    104329200
2008    42885900
2009    58305800
2010    42421100
2011    24134200
2012    22116900
2013    14030000
2014    19801100
2015    23856100
2016    14677600
2017    16565000
2018    14963800
2019    11506200
2020    15567300
2021    9957100
Time taken: 23.377 seconds, Fetched: 25 row(s)
```

<u>To print the total number of volumes :</u>

select sum(volume) from aws;

Output :

```
OK
45110057291
Time taken: 23.588 seconds, Fetched: 1 row(s)
```

<u>To display the total volume between ‘2021-01-04’ and ‘2021-10-27’ :</u>

select sum(volume)

from aws

where dte between ‘2021-01-04’ and ‘2021-10-27’;

Output :

```
OK
699755791
Time taken: 21.029 seconds, Fetched: 1 row(s)
```

<u>To display average of high price :</u>

select avg(high) from aws;

Output :

```
OK
526.2161318467921
Time taken: 30.96 seconds, Fetched: 1 row(s)
```

PREDICTION USING ML MODEL:

Importing the Libraries

```
#Importing the Libraries
import pandas as pd
import numpy as np
import matplotlib. pyplot as plt
import matplotlib
'%matplotlib inline'
import sklearn
import matplotlib. dates as mandates
import sklearn
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error, r2_score
from sklearn import linear_model
from keras.models import Sequential
from keras. layers import Dense
import keras. backend as K
from keras. callbacks import EarlyStopping
from keras. optimizers import Adam
from keras. models import load_model
from keras. layers import LSTM
from keras. utils.vis_utils import plot_model
```

Loading the Stock Market Prediction Data

```
#Get the Dataset
df=pd.read_csv("Amazon.csv",na_values=['null'],index_col='Date',parse_dates=True,infer_datetime_format=True)
print(df.head())
```

```
                Open      High       Low     Close  Adj Close    Volume
Date
1997-05-15  2.437500  2.500000  1.927083  1.958333   1.958333  72156000
1997-05-16  1.968750  1.979167  1.708333  1.729167   1.729167  14700000
1997-05-19  1.760417  1.770833  1.625000  1.708333   1.708333   6106800
1997-05-20  1.729167  1.750000  1.635417  1.635417   1.635417   5467200
1997-05-21  1.635417  1.645833  1.375000  1.427083   1.427083  18853200
```

Data Pre-Processing

Check for Null Values by printing the DataFrame Shape

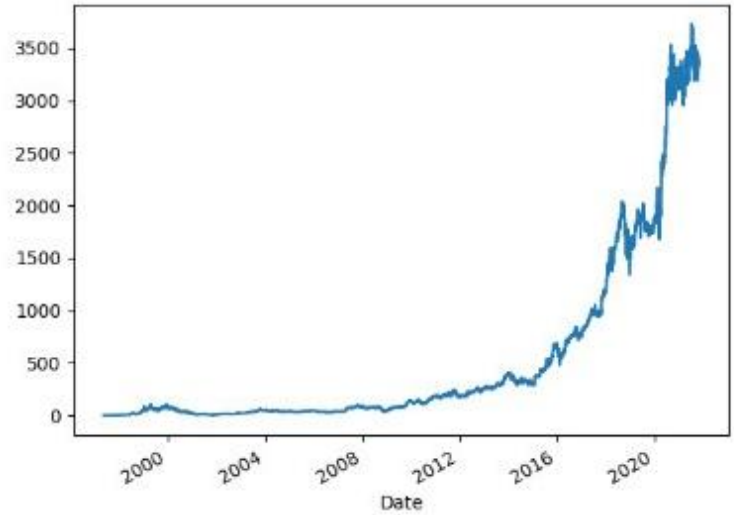
```
#Print the shape of Dataframe  and Check for Null Values
print("Dataframe Shape: ", df. shape)
print("Null Value Present: ", df.isnull().values.any())
```

```
Dataframe Shape:  (6155, 6)
Null Value Present:  False
```

Data Visualizations

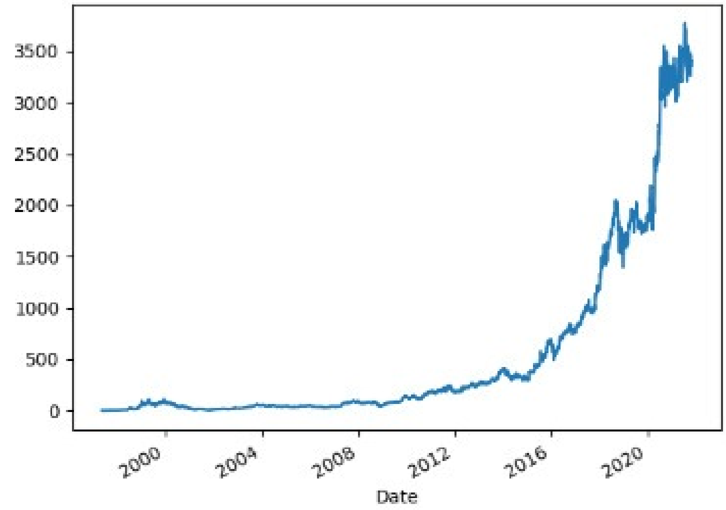
Adj Close Plot:

```
#Plot the True Adj Close Value
df['Adj Close'].plot()
plt.show()
```



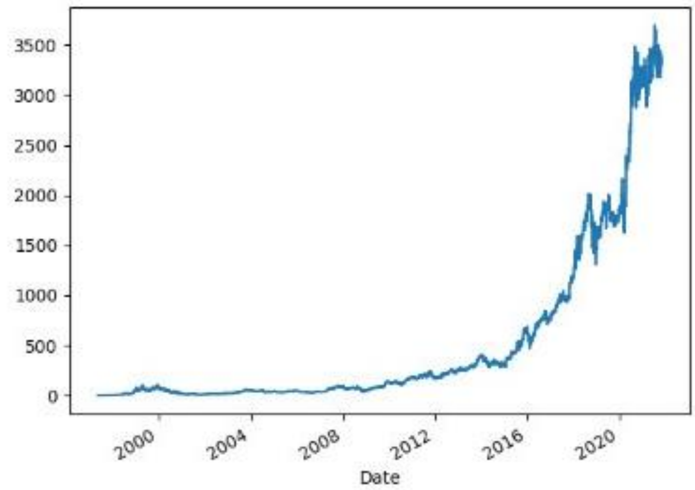
High Price Plot:

```
#Plot the True High Value
df['High'].plot()
plt.show()
```



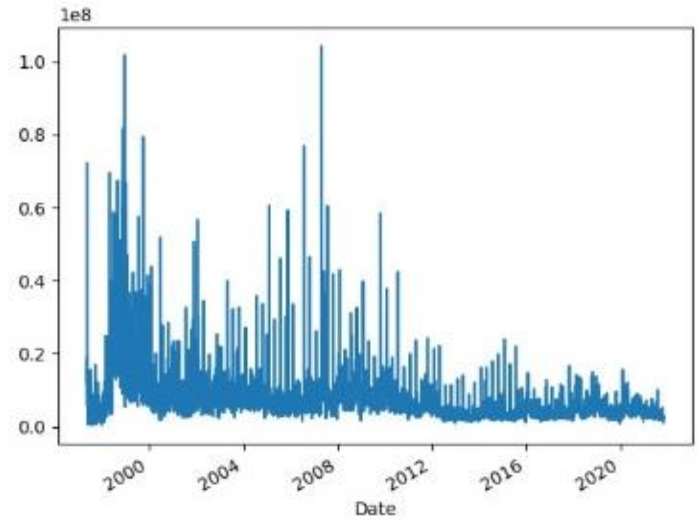
Low Price Plot:

```
#Plot the True Low Value
df['Low'].plot()
plt.show()
```



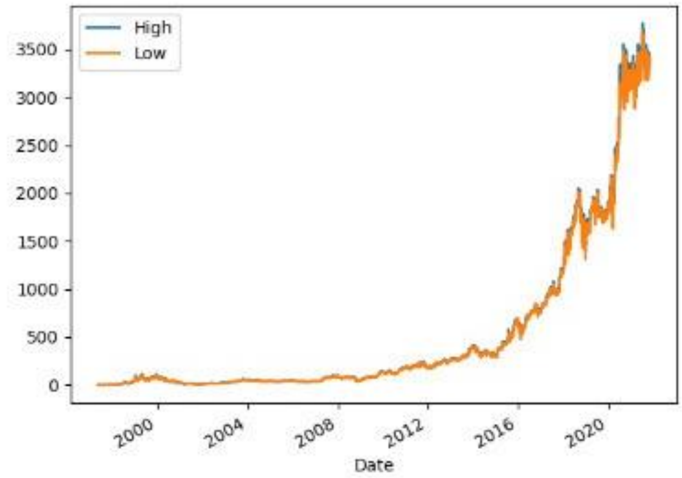
Volume Plot:

```
#Plot the True Volume Value
df['Volume'].plot()
plt.show()
```



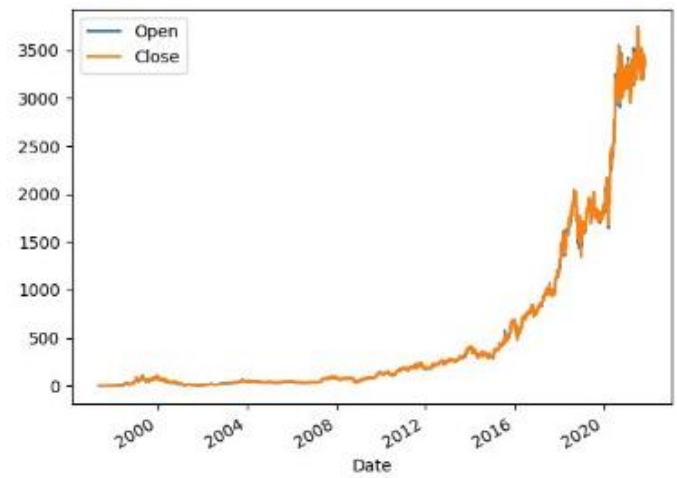
High VS Low Price Plot:

```
#Plot the High VS Low values
high=df['High']
low=df['Low']
plt.legend()
plt.show()
```



Open Price VS Close Price Plot:

```
#Plot the Open VS Close values
opening=df['Open']
closing=df['Close']
plt.legend()
plt.show()
```



<u>Setting the Target Variable and Selecting the Features</u>

```
#Set Target Variable
output_var = pd.DataFrame(df['Adj Close'])
#Selecting the Features
features = ['Open', 'High', 'Low', 'Volume']
```

<u>Scaling</u>

```
#Scaling
scaler = MinMaxScaler()
feature_transform = scaler.fit_transform(df[features])
feature_transform= pd.DataFrame(columns=features, data=feature_transform, index=df.index)
print(feature_transform.head())
```

	Open	High	Low	Volume
Date				
1997-05-15	0.000276	0.000279	0.000166	0.690172
1997-05-16	0.000150	0.000141	0.000107	0.136869
1997-05-19	0.000095	0.000086	0.000085	0.054117
1997-05-20	0.000086	0.000080	0.000087	0.047957
1997-05-21	0.000061	0.000052	0.000017	0.176865

Creating a Training Set and a Test Set for Stock Market Prediction

```
#Splitting to Training set and Test set
timesplit= TimeSeriesSplit(n_splits=10)
for train_index, test_index in timesplit.split(feature_transform):
        X_train, X_test = feature_transform[:len(train_index)], feature_transform[len(train_index): (len(train_index)+len(test_index))]
        y_train, y_test = output_var[:len(train_index)].values.ravel(), output_var[len(train_index): (len(train_index)+len(test_index))].values.ravel()
```

Data Processing For LSTM

```
#Process the data for LSTM
trainX =np.array(X_train)
testX =np.array(X_test)
X_train = trainX.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test = testX.reshape(X_test.shape[0], 1, X_test.shape[1])
```

Building the LSTM Model for Stock Market Prediction

```
#Building the LSTM Model
lstm = Sequential()
lstm.add(LSTM(32, input_shape=(1, trainX.shape[1]), activation='relu', return_sequences=False))
lstm.add(Dense(1))
lstm.compile(loss='mean_squared_error', optimizer='adam')
plot_model(lstm, show_shapes=True, show_layer_names=True)
```

Training the Stock Market Prediction Model

```
#Model Training
history=lstm.fit(X_train, y_train, epochs=100, batch_size=8, verbose=1, shuffle=False)
```

```
Epoch 1/100
700/700 [==============================] - 2s 2ms/step - loss: 287838.5938
Epoch 2/100
700/700 [==============================] - 1s 2ms/step - loss: 276506.9062
Epoch 3/100
700/700 [==============================] - 1s 2ms/step - loss: 265478.0000
Epoch 4/100
700/700 [==============================] - 1s 2ms/step - loss: 253743.8906
```

```
Epoch 5/100
700/700 [==============================] - 1s 2ms/step - loss: 241721.1406
Epoch 6/100
700/700 [==============================] - 1s 2ms/step - loss: 229750.9688
Epoch 7/100
700/700 [==============================] - 2s 2ms/step - loss: 218177.4688
Epoch 8/100
700/700 [==============================] - 2s 2ms/step - loss: 207114.7344
Epoch 9/100
700/700 [==============================] - 2s 2ms/step - loss: 196542.2344
Epoch 10/100
700/700 [==============================] - 2s 2ms/step - loss: 186322.3281
Epoch 11/100
700/700 [==============================] - 2s 2ms/step - loss: 176229.9219
Epoch 12/100
700/700 [==============================] - 2s 2ms/step - loss: 165942.0312
Epoch 13/100
700/700 [==============================] - 1s 2ms/step - loss: 155053.0469
Epoch 14/100
700/700 [==============================] - 1s 2ms/step - loss: 143356.9844
Epoch 15/100
700/700 [==============================] - 2s 2ms/step - loss: 131118.8906
Epoch 16/100
700/700 [==============================] - 1s 2ms/step - loss: 118869.0703
Epoch 17/100
700/700 [==============================] - 1s 2ms/step - loss: 107006.9219
Epoch 18/100
700/700 [==============================] - 1s 2ms/step - loss: 95712.4062
Epoch 19/100
700/700 [==============================] - 1s 2ms/step - loss: 85067.6016
Epoch 20/100
700/700 [==============================] - 2s 2ms/step - loss: 75130.3125
Epoch 21/100
700/700 [==============================] - 2s 2ms/step - loss: 65945.1484
Epoch 22/100
700/700 [==============================] - 2s 2ms/step - loss: 57538.4141
Epoch 23/100
700/700 [==============================] - 1s 2ms/step - loss: 49906.4375
Epoch 24/100
700/700 [==============================] - 1s 2ms/step - loss: 43021.7148
Epoch 25/100
700/700 [==============================] - 1s 2ms/step - loss: 36852.3672
Epoch 26/100
700/700 [==============================] - 2s 2ms/step - loss: 31374.6445
Epoch 27/100
700/700 [==============================] - 1s 2ms/step - loss: 26559.5918
```

```
Epoch 91/100
700/700 [==============================] - 2s 2ms/step - loss: 41.6428
Epoch 92/100
700/700 [==============================] - 1s 2ms/step - loss: 41.1335
Epoch 93/100
700/700 [==============================] - 2s 2ms/step - loss: 40.6536
Epoch 94/100
700/700 [==============================] - 1s 2ms/step - loss: 40.2047
Epoch 95/100
700/700 [==============================] - 2s 2ms/step - loss: 39.7859
Epoch 96/100
700/700 [==============================] - 2s 2ms/step - loss: 39.3970
Epoch 97/100
700/700 [==============================] - 1s 2ms/step - loss: 39.0365
Epoch 98/100
700/700 [==============================] - 1s 2ms/step - loss: 38.7030
Epoch 99/100
700/700 [==============================] - 2s 2ms/step - loss: 38.3955
Epoch 100/100
700/700 [==============================] - 2s 2ms/step - loss: 38.1114
18/18 [==============================] - 0s 2ms/step
```

```
[[1796.3949]
 [1803.0771]
 [1786.7048]
 [1782.4584]
 [1798.558 ]
 [1823.5421]
 [1815.1453]
 [1825.8591]
 [1823.6205]
 [1785.5626]
 [1765.5543]
 [1772.5658]
 [1761.2906]
 [1791.4886]
 [1792.2683]
 [1784.8164]
 [1807.9333]
 [1830.9371]
 [1839.2596]
 [1842.5955]
 [1822.5349]
 [1822.8237]
 [1845.8944]
 [1845.0508]
 [1821.1879]
 [1816.4862]
```

```
 [1828.3041]
 [1815.3898]
 [1784.3043]
 [1778.6254]
 [1753.0977]
 [1757.8656]
 [1742.2335]
 [1730.336 ]
 [1748.9088]
 [1726.1323]
 [1713.1298]
 [1734.5033]
 [1740.2994]
 [1724.0249]
 [1727.382 ]
 [1731.6023]
 [1745.0358]
 [1736.9495]
 [1758.615 ]
 [1782.122 ]
 [1797.576 ]
```

```
 [1810.6488]
 [1809.3542]
 [1802.4275]
 [1802.6865]
 [1789.2445]
 [1780.5413]
 [1783.0823]
 [1770.5967]
 [1761.6136]
 [1756.7864]
 [1743.8942]
 [1759.278 ]
 [1754.2025]
 [1745.8417]
 [1744.6615]
 [1766.7611]
 [1790.3549]
 [1812.4419]
 [1819.0233]
 [1795.6498]
 [1765.4493]
 [1779.7935]
 [1761.1833]
 [1754.1964]
 [1760.0919]
 [1750.1265]
 [1748.3698]
 [1758.7526]
 [1768.5643]
 [1770.0079]
 [1787.7197]
 [1796.9532]
 [1787.8356]
```

```
 [1772.8134]
 [1785.1732]
 [1795.4554]
 [1810.6488]
 [1809.3542]
 [1802.4275]
 [1802.6865]
 [1789.2445]
 [1780.5413]
 [1783.0823]
 [1770.5967]
 [1761.6136]
 [1756.7864]
 [1743.8942]
 [1759.278 ]
 [1754.2025]
 [1745.8417]
 [1744.6615]
 [1766.7611]
 [1790.3549]
 [1812.4419]
 [1819.0233]
 [1795.6498]
 [1765.4493]
 [1779.7935]
 [1761.1833]
 [1754.1964]
 [1760.0919]
 [1750.1265]
 [1748.3698]
 [1758.7526]
 [1768.5643]
 [1770.0079]
 [1787.7197]
 [1796.9532]
 [1787.8356]
```

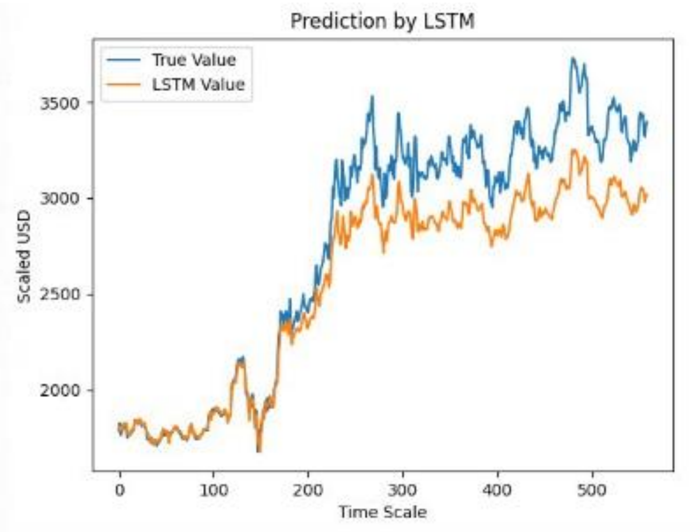
```
 [2953.4717]
 [2943.2036]
 [2929.3303]
 [2911.6006]
 [2890.8774]
 [2889.162 ]
 [2937.3435]
 [2940.9705]
 [2914.833 ]
 [2904.0515]
 [2919.7468]
 [2937.058 ]
 [2978.2485]
 [3011.0425]
 [3026.2222]
 [3025.9536]
 [3012.1519]
 [3001.8665]
 [2953.7822]
 [2984.167 ]
 [2989.5303]]
```

LSTM Prediction

```
#LSTM Prediction
y_pred= lstm.predict(X_test)
print(y_pred)
```

Comparing Predicted vs True Adjusted Close Value – LSTM

```
#Predicted vs True Adj Close Value - LSTM
plt.plot(y_test, label='True Value')
plt.plot(y_pred, label='LSTM Value')
plt.title("Prediction by LSTM")
plt.xlabel('Time Scale')
plt.ylabel('Scaled USD')
plt.legend()
plt.show()
```



ANALYSIS OF THE OUTPUT:

- The Amazon Stock Price for the year 1997 – 2021 were analysed and the maximum, minimum of high, low, open, close price, highest – lowest volume, count of volume year-wise and so on were calculated on both Pig and Hive.
- Also, the machine learning models for stock market were used for forecasting.
- This model saves our time and resources and also it outperforms people in terms of performance in predicting.
- Using these results, we can draw the inferences about the dataset and just predict how the next year's result will be.