One University. One World. Yours.

Master of Science in Computing and Data Analytics

Big Data Project

Managing Info & Tech Systems
MCDA 5570

Submitted by:

Name	A#
Shubham Chumber	A00433094
Sachit Jain	A00432721
Gyaneshwar Rao Nampally	A00433014
Vamsi Manda	A00433234
Kshitij Parashar	A00430628

Table of Contents

Abstract

The aim of the project is to take large chunks of aggregated Big data collected over multiple years, process these datasets to extract market knowledge and thereby leverage these insights gained to make predictions and analyse trends. The Big data application works with companies financial stock data as warehoused by the S&P 500 indexing to understand the future trends in the pricing of the stocks for various companies subject to various external constraints.

Objective

Stock market globally trades shares of private and public hedged companies in order to generate capital in exchange for equivalent ownership for the respective organisation. Stocks propound huge financial gain prospects. However, associated with this opportunity is a huge risk and uncertainty involved and our objective is to bridge this gap with informational metrics. We aim to utilise Big data and its technologies in order to generate information and intelligence that can effectively help us predict stock prices. Due to unavailability of livestock data streams, we worked with Kaggle Dataset of stocks data collected over 5 years. We aim to integrate various libraries of the Hadoop ecosystem such as Sqoop, Hive, Spark and MLlib to interface and process data.

Project outline & flow

The pipelining process starts with loading and transforming the dataset. This helps us to prevent any erroneous observations. The data is then moved from a relational schema to a big data file structure deployed on a cluster with a simplistic architecture having the working node and the name node deployed on the server. The data file is transferred using Sqoop into Hive tables. In order to facilitate faster execution of the query, we partitioned the data into different years column. The processing of the data is carried out in Spark. The Hive transformed and partitioned data was transferred to the Spark by creating a spark instance in HiveQL. The resulting RDD was then clustered to group stocks into various categories on the basis of their performance followed by a time series analysis on dates and company profiles to understand future trends in the stock prices and performance.

Data Source

URL: https://www.kaggle.com/camnugent/sandp500

The folder individual_stocks_5yr contains files of data for individual stocks, labelled by their stock ticker name. The all_stocks_5yr.csv contains the same data, presented in a merged .csv file. Depending on the intended use (graphing, modelling etc.) the user may prefer one of these given formats.

All the files have the following columns:

- Date - in format: yy-mm-dd
- Open - price of the stock at market open (this is NYSE data so all in USD)
- High - Highest price reached in the day
- Low Close - Lowest price reached in the day
- Volume - Number of shares traded
- Name - the stock's ticker name

Architecture

The data is imported from CSV file to relational database. Using Sqoop, the data is transferred from relational database to Hadoop cluster. The data operations are performed by retrieving the data from Hive using Spark and analysis of data is performed using linear regression and K-Means algorithm.

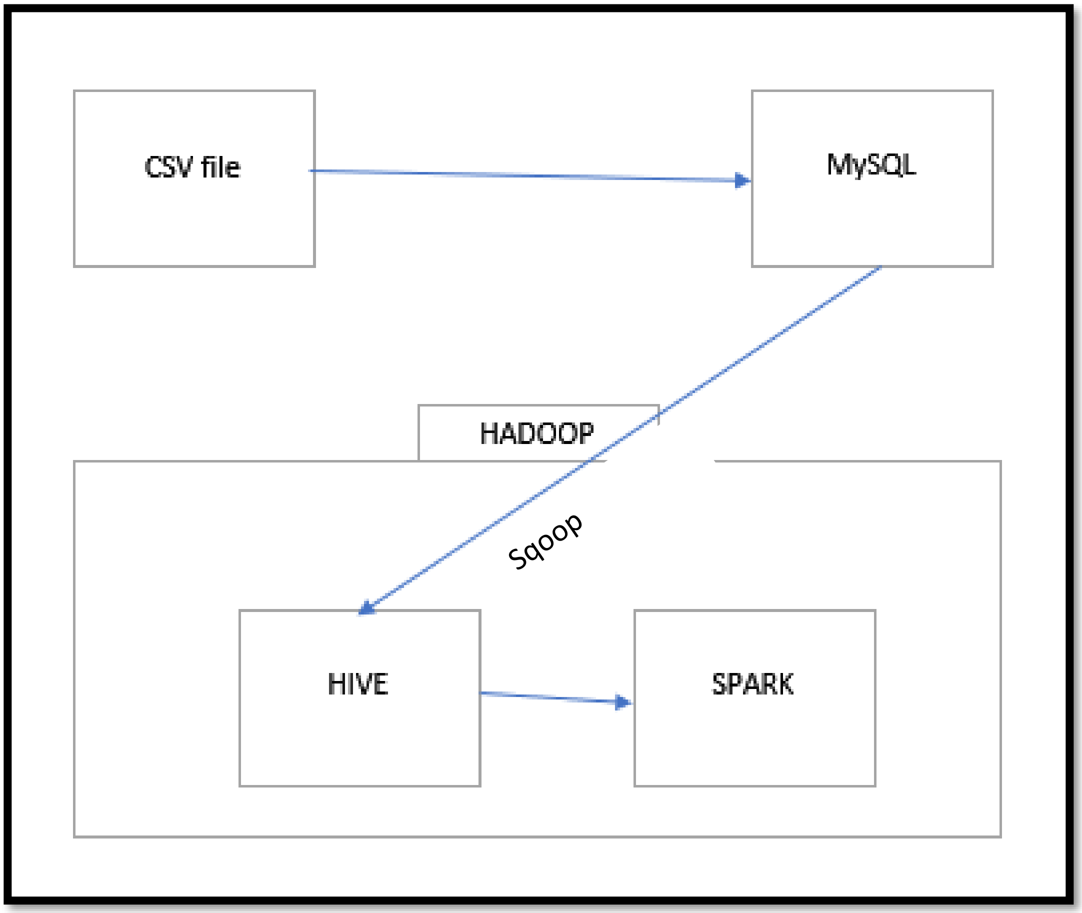

Figure 1 Architecture of Data Flow

Code Snippets and explanations

SQL pre-processing

Creating a table in SQL appropriate to the CSV file.

```
create table if not exists s_chumber.sandp500 ( date date, open
double(10,2), high double(10,2)   , low  double(10,2)   , close
double(10,2)  ,volume int, Name VARCHAR(50) ) engine = innodb;

alter table sandp500 add column year int not null;

update sandp500  set year = (select year(d.date) from sandp500 d
where d.date=s.date) ;
```

Transferring data from CSV to relational database (MySQL)

```
load data local infile
'/home/student_2019_winter/s_chumber/public_html/sandp500/all_stocks_
5yr.csv' into table s_chumber.sandp500 COLUMNS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"' ESCAPED BY '"' LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

Sqoop commands and data load to HDFS

Transferring data from relational database to HDFS cluster.

```
sqoop import --connect
jdbc:mysql://dev.cs.smu.ca:3306/s_chumber --username s_chumber --
password A00433094
--table s_chumber.sandp500 -m 1 --hive-import --create-hive-table  --
hive-table
sandp500 --target-dir '/apps/hive/warehouse/sandp500'
```

Linear Regression Model on Amazon stock

In order to create a Resilient Distributed Database managed over cluster we transferred our data tables by creating a spark session instance. The above code uses a Python interpreter and makes use of python data frames. The OS module function ‘abspath()’ allows us to return the location to the directory as the current directory that the spark application has started. The ‘spark’ instance created has various properties specified such as the main title, the location of the housed spark warehouse to access and an additional parameter that allows us to either access an already created warehouse or create if one doesn’t exist. We then collect the entire Hive data into a python dataframe using simple HQL commands and then replace the missing values with ‘ ’ (Single Quotes) and the result thus obtained is made visible for the first 5 rows.

```
%pyspark
from os.path import expanduser, join, abspath
from pyspark.sql import SparkSession
from pyspark.sql import Row
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import dates as dates
from IPython.display import display

# warehouse_location points to the default location for managed
databases and tables
warehouse_location = abspath('spark-warehouse')

spark = SparkSession \
```

```
    .builder \
    .appName("Python Spark SQL Hive integration example") \
    .config("spark.sql.warehouse.dir", warehouse_location) \
    .enableHiveSupport() \
    .getOrCreate()

# Queries are expressed in HiveQL
df = spark.sql("SELECT * FROM sandp500")
df = df.na.fill(' ')
df.show(5, truncate = False)
```

Output

```
+----------+-----+-----+-----+-----+--------+----+----+
|date      |open |high |low  |close|volume  |name|year|
+----------+-----+-----+-----+-----+--------+----+----+
|2013-02-08|15.07|15.12|14.63|14.75|8407500 |AAL |2013|
|2013-02-11|14.89|15.01|14.26|14.46|8882000 |AAL |2013|
|2013-02-12|14.45|14.51|14.1 |14.27|8126000 |AAL |2013|
|2013-02-13|14.3 |14.94|14.25|14.66|10259500|AAL |2013|
|2013-02-14|14.94|14.96|13.16|13.99|31879900|AAL |2013|
+----------+-----+-----+-----+-----+--------+----+----+
only showing top 5 rows
```

The schema for the data set can be demonstrated as mentioned below.

```
%pyspark
df.printSchema()
```

Output

```
root
|-- date: date (nullable = true)
|-- open: double (nullable = true)
|-- high: double (nullable = true)
|-- low: double (nullable = true)
|-- close: double (nullable = true)
|-- volume: integer (nullable = true)
|-- name: string (nullable = false)
|-- year: integer (nullable = true)
```

Renaming the column “Name” as “Ticks”

```
%pyspark
df = df.withColumnRenamed('Name','Ticks')
```

Calculating the average closing price of the individual companies and sorting them by descending order.

```
%pyspark
from pyspark.sql.functions import col
df.groupby('Ticks').agg({'close':
'mean'}).sort(col("avg(close)").desc()).show()
```

Output

```
+-----+------------------+
|Ticks|        avg(close)|
+-----+------------------+
| PCLN| 1312.873534551231|
| GOOG| 725.4033435897428|
|GOOGL| 682.2338443208897|
|  AZO| 619.7036536934078|
| AMZN| 576.8800397140589|
|  CMG| 493.2560047656866|
| REGN| 381.8330818109609|
|  MTD|356.29749801429693|
|  BLK|348.62896743447163|
| BIIB|295.41327243844336|
| EQIX| 290.4308816521043|
|  SHW|261.45109610802257|
|  ADS| 241.0440508339953|
|  GWW|230.36709293089712|
|  AGN|222.26103256552835|
|  TDG|215.10420174741841|
| CHTR|208.08840349483734|
|  ESS|207.71353455123113|
|  LMT|207.02929308975388|
| ORLY| 204.4118268467037|
+-----+------------------+
only showing top 20 rows
```

Considering stock of Amazon ("AMZN") and converting the dataset to Pandas for further operations.

```
%pyspark
from pyspark.sql.functions import unix_timestamp, from_unixtime
amzn=df.filter(df.Ticks == 'AMZN')
amzn_df = amzn.select(
    'open','high',
    'low',
    'volume',
    'year',
    from_unixtime(unix_timestamp('date', '%Y/%m/%d')).alias('date'),
    'Ticks',
    'close'
)
amzn_df = amzn_df.toPandas()
amzn_df = amzn_df.copy()
amzn_df.loc[:, 'date'] = pd.to_datetime(amzn_df.loc[:,'date'],
format="%Y/%m/%d")
```

The Schema of "AMZN" dataset is extracted using the below mentioned query.

```
%pyspark
amzn_df.info()
```

Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259 entries, 0 to 1258
Data columns (total 8 columns):
open      1259 non-null float64
high      1259 non-null float64
low       1259 non-null float64
volume    1259 non-null int32
year      1259 non-null int32
date      1259 non-null datetime64[ns]
Ticks     1259 non-null object
close     1259 non-null float64
dtypes: datetime64[ns](1), float64(4), int32(2), object(1)
memory usage: 68.9+ KB
```

Plotting the highs and lows of the stock per day according to the date mentioned in the dataset. While the high is mentioned as red and the low is mentioned in blue.

```
%pyspark
# Simple plotting of Amazon Stock Price
# Second Subplot
#plt.plot(amzn_df["date"],amzn_df["high"], color="blue")
plt.plot(amzn_df["date"], amzn_df["high"], 'r--',
amzn_df["date"],amzn_df["low"], 'b--')
plt.show()
```

Output

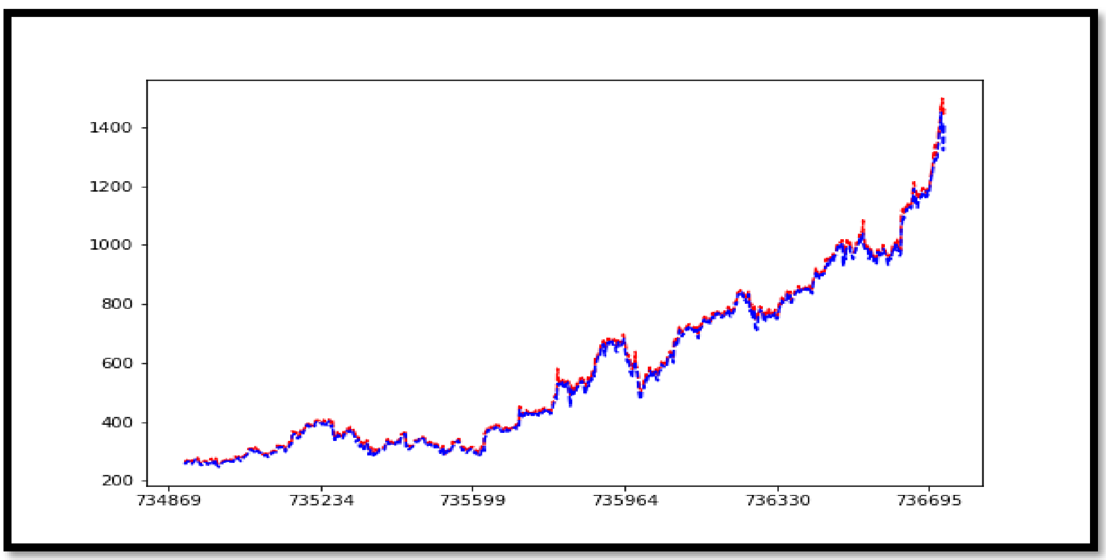

Figure 2 Plotting of High-Low Stock for Amazon

Plotting the volume of Amazon trade for each date is measured and plotted in orange as per the respective code.

```
%pyspark
# Simple plotting of Amazon Stock Price
# Fourth Subplot
plt.plot(amzn_df["date"],amzn_df["volume"], color="orange")
plt.show()
```

Output

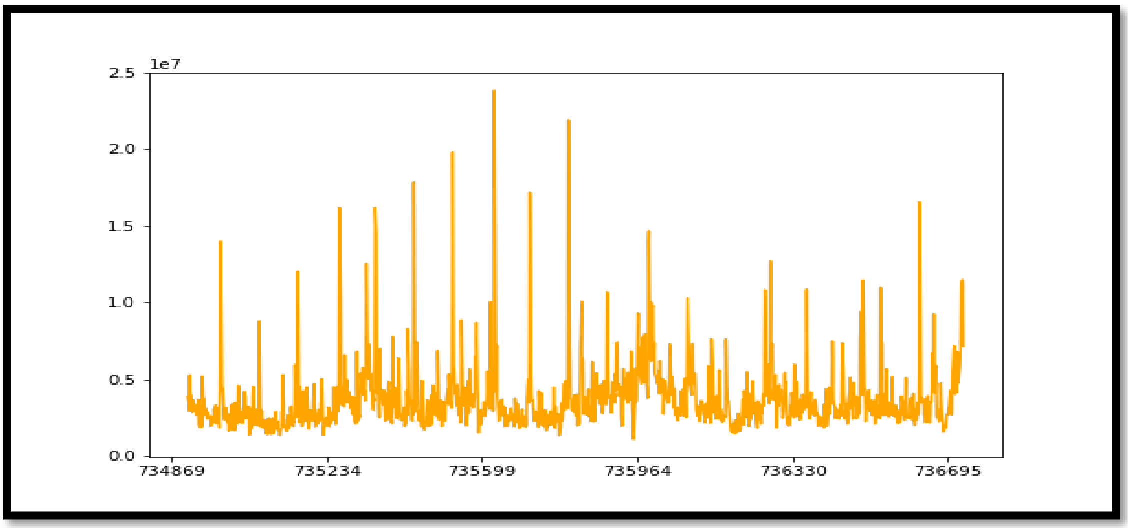

Figure 3 Everyday Volume Trade for Amazon

Considering the vector assembler for performing machine learning algorithm and choosing the feature columns (open, high, low, volume, year, date) as shown below into the “features” column for prediction.

```
%pyspark
amzn_df = spark.createDataFrame(amzn_df)
amzn_df.select(amzn_df.columns[:-2]).show()
amzn_df = amzn_df.select(
    'open',
    'high',
    'low',
    'volume',
    'year',
    unix_timestamp('date', '%Y/%m/%d').alias('date'),
    'Ticks',
    'close'
)
feature_columns = amzn_df.columns[:-2]

from pyspark.ml.feature import VectorAssembler
```

```
assembler = 
VectorAssembler(inputCols=feature_columns,outputCol="features")
```

Output

```
+------+------+------+-------+----+-------------------+
|  open|  high|   low| volume|year|               date|
+------+------+------+-------+----+-------------------+
| 261.4|265.25|260.56|3879078|2013|2013-02-08 00:00:00|
| 263.2|263.25| 256.6|3403403|2013|2013-02-11 00:00:00|
|259.19|260.16| 257.0|2938660|2013|2013-02-12 00:00:00|
|261.53|269.96| 260.3|5292996|2013|2013-02-13 00:00:00|
|267.37|270.65| 265.4|3462780|2013|2013-02-14 00:00:00|
|267.63|268.92|263.11|3979832|2013|2013-02-15 00:00:00|
|265.91|270.11| 264.5|2853752|2013|2013-02-19 00:00:00|
| 270.2| 274.3|266.37|3528862|2013|2013-02-20 00:00:00|
|265.12|269.48|263.25|3637396|2013|2013-02-21 00:00:00|
|266.62|267.11|261.61|3123402|2013|2013-02-22 00:00:00|
|266.94|268.69|259.65|3032109|2013|2013-02-25 00:00:00|
|260.89|262.04|255.73|3348011|2013|2013-02-26 00:00:00|
| 259.4|265.83|256.86|2908010|2013|2013-02-27 00:00:00|
|261.81| 267.0|260.63|2667199|2013|2013-02-28 00:00:00|
|263.27| 266.6|261.04|2956724|2013|2013-03-01 00:00:00|
|265.36| 273.3|264.14|3452783|2013|2013-03-04 00:00:00|
| 274.0|276.68|269.99|3685983|2013|2013-03-05 00:00:00|
|275.76|276.49|271.83|2050452|2013|2013-03-06 00:00:00|
| 274.1| 274.8|271.85|1938987|2013|2013-03-07 00:00:00|
| 275.0|275.44| 271.5|1879762|2013|2013-03-08 00:00:00|
+------+------+------+-------+----+-------------------+
only showing top 20 rows
```

Preparation of the dataset and the vector in the feature's column into the variable amzn_df_2

```
%pyspark
amzn_df_2 = assembler.transform(amzn_df)
amzn_df_2.show()
```

Output

```
+------+------+------+-------+----+----------+-----+------+--------------------+
|  open|  high|   low| volume|year|      date|Ticks| close|            features|
+------+------+------+-------+----+----------+-----+------+--------------------+
| 261.4|265.25|260.56|3879078|2013|1360281600| AMZN|261.95|[261.4,265.25,260...|
| 263.2|263.25| 256.6|3403403|2013|1360540800| AMZN|257.21|[263.2,263.25,256...|
|259.19|260.16| 257.0|2938660|2013|1360627200| AMZN| 258.7|[259.19,260.16,25...|
|261.53|269.96| 260.3|5292996|2013|1360713600| AMZN|269.47|[261.53,269.96,26...|
|267.37|270.65| 265.4|3462780|2013|1360800000| AMZN|269.24|[267.37,270.65,26...|
|267.63|268.92|263.11|3979832|2013|1360886400| AMZN|265.09|[267.63,268.92,26...|
|265.91|270.11| 264.5|2853752|2013|1361232000| AMZN|269.75|[265.91,270.11,26...|
| 270.2| 274.3|266.37|3528862|2013|1361318400| AMZN|266.41|[270.2,274.3,266....|
|265.12|269.48|263.25|3637396|2013|1361404800| AMZN|265.94|[265.12,269.48,26...|
|266.62|267.11|261.61|3123402|2013|1361491200| AMZN|265.42|[266.62,267.11,26...|
|266.94|268.69|259.65|3032109|2013|1361750400| AMZN|259.87|[266.94,268.69,25...|
|260.89|262.04|255.73|3348011|2013|1361836800| AMZN|259.36|[260.89,262.04,25...|
```

```
| 259.4|265.83|256.86|2908010|2013|1361923200| AMZN|263.25|[259.4,265.83,256...|
|261.81| 267.0|260.63|2667199|2013|1362009600| AMZN|264.27|[261.81,267.0,260...|
|263.27| 266.6|261.04|2956724|2013|1362096000| AMZN|265.74|[263.27,266.6,261...|
|265.36| 273.3|264.14|3452783|2013|1362355200| AMZN|273.11|[265.36,273.3,264...|
| 274.0|276.68|269.99|3685983|2013|1362441600| AMZN|275.59|[274.0,276.68,269...|
|275.76|276.49|271.83|2050452|2013|1362528000| AMZN|273.79|[275.76,276.49,27...|
| 274.1| 274.8|271.85|1938987|2013|1362614400| AMZN|273.88|[274.1,274.8,271....|
| 275.0|275.44| 271.5|1879762|2013|1362700800| AMZN|274.19|[275.0,275.44,271...|
+------+------+------+-------+----+----------+-----+------+--------------------+
only showing top 20 rows
```

Splitting the dataset into 70% train set and 30% test set and performing linear Regression. We calculated the Mean Absolute Error, Root Mean Squared Error and R Squared, which confirms the confidence of prediction and the error that it can give.

```
%pyspark
train, test = amzn_df_2.randomSplit([0.7, 0.3])
%pyspark
from pyspark.ml.regression import LinearRegression
algo = LinearRegression(featuresCol="features", labelCol="close")
model = algo.fit(train)
%pyspark
evaluation_summary = model.evaluate(test)
%pyspark
evaluation_summary.meanAbsoluteError
evaluation_summary.rootMeanSquaredError
evaluation_summary.r2
```

Output

```
2.222462705537086
3.208489872866394
0.9998757569842011
```

Prediction performed on test data and the predicted values are stored in the table as shown below in the output.

```
%pyspark
predictions = model.transform(test)
%pyspark
predictions.select(predictions.columns).show()
```

Output

```
+------+------+------+-------+----+----------+-----+------+--------------------+-----------
------+
|  open|  high|   low| volume|year|      date|Ticks| close|            features|
prediction|
+------+------+------+-------+----+----------+-----+------+--------------------+-----------
------+
```

```
|248.94|252.93|245.78|3922605|2013|1367452800|
AMZN|252.55|[248.94,252.93,24...|249.71932361197193|
|255.92|259.68|255.63|2295573|2013|1365379200| AMZN|258.95|[255.92,259.68,25...|
258.9883874829368|
|256.11| 257.0|252.68|2805646|2013|1363824000|
AMZN|253.39|[256.11,257.0,252...|254.10564038397243|
|256.14|259.25| 254.7|3513934|2013|1367539200|
AMZN|258.05|[256.14,259.25,25...|257.42879027367906|
|256.31|259.74|252.91|3133399|2013|1367884800|
AMZN|257.73|[256.31,259.74,25...|256.61567013949025|
|256.87| 260.3|255.33|2676986|2013|1367971200|
AMZN|258.68|[256.87,260.3,255...|258.62676772143595|
|258.09| 259.5|253.42|2347669|2013|1367798400| AMZN|255.72|[258.09,259.5,253...|
255.8176982768147|
|258.58|259.43| 254.5|2513758|2013|1364169600| AMZN|256.02|[258.58,259.43,25...|
256.1587479815382|
|258.75|265.93| 257.9|2874824|2013|1364342400| AMZN| 265.3|[258.75,265.93,25...|
264.4737673108135|
|259.19|260.16| 257.0|2938660|2013|1360627200| AMZN|
258.7|[259.19,260.16,25...|258.10874758987575|
| 259.3|261.49|257.12|2719833|2013|1363564800| AMZN|257.89|[259.3,261.49,257...|
259.4308314510296|
|259.35| 264.6|258.03|2119071|2013|1366588800|
AMZN|263.55|[259.35,264.6,258...|263.13625389606943|
| 259.4|265.83|256.86|2908010|2013|1361923200|
AMZN|263.25|[259.4,265.83,256...|263.21254894345464|
|260.89|262.04|255.73|3348011|2013|1361836800| AMZN|259.36|[260.89,262.04,25...|
257.7495175679976|
|261.53|269.96| 260.3|5292996|2013|1360713600| AMZN|269.47|[261.53,269.96,26...|
267.4755047400289|
|261.78|265.98|259.32|2322407|2013|1365552000| AMZN|264.77|[261.78,265.98,25...|
263.7123478627627|
| 264.5|269.87| 264.5|2270594|2013|1366675200| AMZN| 268.9|[264.5,269.87,264...|
269.2926535083625|
|265.36| 273.3|264.14|3452783|2013|1362355200| AMZN|273.11|[265.36,273.3,264...|
271.3735306368685|
|265.71|268.57|265.62|1667469|2013|1369872000| AMZN|266.83|[265.71,268.57,26...|
268.2442122483335|
|265.82|267.38|264.06|2473257|2013|1364428800|
AMZN|266.49|[265.82,267.38,26...|265.72462381713046|
+------+------+------+-------+----+----------+-----+------+------------------+------------
------+
only showing top 20 rows
```

Plotting the graph between actual and prediction of “closing price”.

```
%pyspark
# Simple plotting of Amazon Stock Price actual and predicted
plt.plot(plot_actual_prediction['date'], plot_actual_prediction['close'], 'r--
',plot_actual_prediction['date'], plot_actual_prediction['prediction'], 'b--')
plt.show()
```

Output

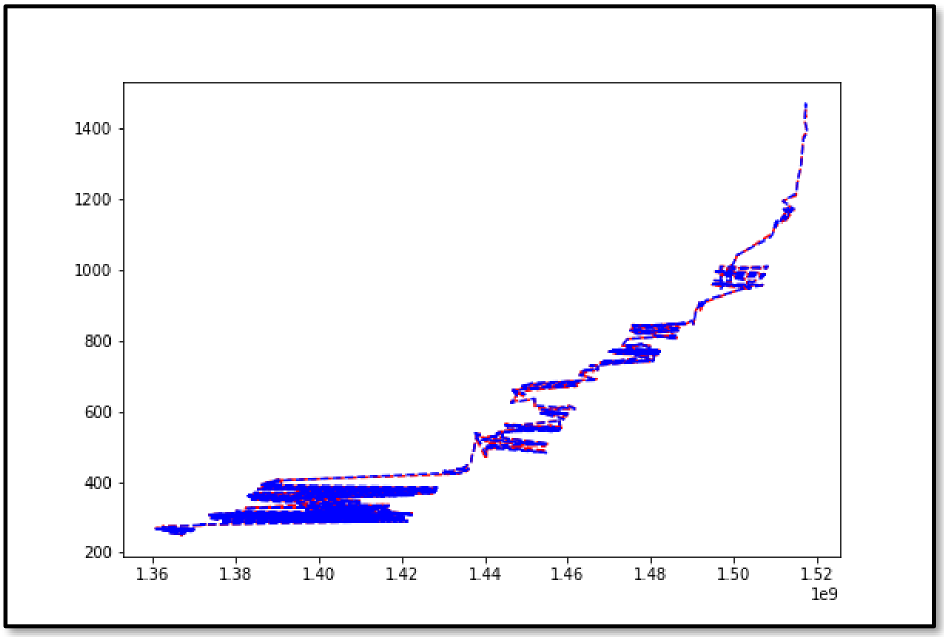

Figure 4 Comparison of actual closing price and predicted closing price

K-Means Clustering on all the stocks

Choosing a dataset df_2 with specific column to perform clustering operations.

```
%pyspark
df_2 = df.select(
    'open',
    'high',
    'low',
    'volume',
    'year',
     from_unixtime(unix_timestamp('date', 'YYYY-MM-
dd')).alias('date'),
    'Ticks',
    'close'
)
df_2.show()
```

Output

```
+-----+-----+-----+--------+----+-------------------+-----+-----+
| open| high|  low|  volume|year|               date|Ticks|close|
+-----+-----+-----+--------+----+-------------------+-----+-----+
|15.07|15.12|14.63| 8407500|2013|2013-02-08 00:00:00|  AAL|14.75|
|14.89|15.01|14.26| 8882000|2013|2013-02-11 00:00:00|  AAL|14.46|
|14.45|14.51| 14.1| 8126000|2013|2013-02-12 00:00:00|  AAL|14.27|
| 14.3|14.94|14.25|10259500|2013|2013-02-13 00:00:00|  AAL|14.66|
|14.94|14.96|13.16|31879900|2013|2013-02-14 00:00:00|  AAL|13.99|
|13.93|14.61|13.93|15628000|2013|2013-02-15 00:00:00|  AAL| 14.5|
|14.33|14.56|14.08|11354400|2013|2013-02-19 00:00:00|  AAL|14.26|
|14.17|14.26|13.15|14725200|2013|2013-02-20 00:00:00|  AAL|13.33|
|13.62|13.95| 12.9|11922100|2013|2013-02-21 00:00:00|  AAL|13.37|
|13.57| 13.6|13.21| 6071400|2013|2013-02-22 00:00:00|  AAL|13.57|
| 13.6|13.76| 13.0| 7186400|2013|2013-02-25 00:00:00|  AAL|13.02|
|13.14|13.42| 12.7| 9419000|2013|2013-02-26 00:00:00|  AAL|13.26|
|13.28|13.62|13.18| 7390500|2013|2013-02-27 00:00:00|  AAL|13.41|
|13.49|13.63|13.39| 6143600|2013|2013-02-28 00:00:00|  AAL|13.43|
|13.37|13.95|13.32| 7376800|2013|2013-03-01 00:00:00|  AAL|13.61|
| 13.5|14.07|13.47| 8174800|2013|2013-03-04 00:00:00|  AAL| 13.9|
|14.01|14.05|13.71| 7676100|2013|2013-03-05 00:00:00|  AAL|14.05|
|14.52|14.68|14.25|13243200|2013|2013-03-06 00:00:00|  AAL|14.57|
| 14.7|14.93| 14.5| 9125300|2013|2013-03-07 00:00:00|  AAL|14.82|
|14.99| 15.2|14.84|10593700|2013|2013-03-08 00:00:00|  AAL|14.92|
+-----+-----+-----+--------+----+-------------------+-----+-----+
only showing top 20 rows
```

We are calculating the difference between the open and close sale on the market for all stocks for every particular day and storing the value in 'diff' column.

```
%pyspark
from pyspark.sql import functions as F
from pyspark.sql.window import Window

my_window = Window.partitionBy("Ticks").orderBy("date")

df = df.withColumn("prev_value", F.lag(df.close).over(my_window))
df = df.withColumn("diff", F.when(F.isnull(df.close - df.prev_value),
0)
                                .otherwise(df.close - df.prev_value))

df.show()
```

Output

```
+----------+-----+-----+-----+-----+-------+-----+----+----------+--------------------+
|      date| open| high|  low|close| volume|Ticks|year|prev_value|                diff|
+----------+-----+-----+-----+-----+-------+-----+----+----------+--------------------+
|2013-02-08|93.11|93.99| 92.9|93.66|1059844| ALXN|2013|      null|                 0.0|
|2013-02-11|93.66|93.97|91.98| 92.3|1707183| ALXN|2013|     93.66| -1.3599999999999994|
|2013-02-12|92.34|92.43|89.81|90.21|3115852| ALXN|2013|      92.3| -2.0900000000000034|
|2013-02-13|90.12|92.21|89.85|92.01|2496134| ALXN|2013|     90.21|  1.8000000000000114|
|2013-02-14|94.52|95.25|84.52|87.63|5936469| ALXN|2013|     92.01|   -4.38000000000001|
|2013-02-15|87.93|88.47|85.53|86.01|3690421| ALXN|2013|     87.63| -1.6199999999999903|
```

```
|2013-02-19|82.77|85.47|81.82|83.39|3996771| ALXN|2013|     86.01| -2.6200000000000045|
|2013-02-20|84.44|85.28| 83.3|84.46|4062371| ALXN|2013|     83.39|  1.0699999999999932|
|2013-02-21|84.38|87.98|83.62|87.28|3895591| ALXN|2013|     84.46|  2.8200000000000074|
|2013-02-22|87.28| 87.8|86.93|87.44|2005904| ALXN|2013|     87.28|  0.1599999999999966|
|2013-02-25|87.19|87.89|86.18|86.29|1949996| ALXN|2013|     87.44| -1.1499999999999915|
|2013-02-26|86.41|86.75|85.17|85.36|2840012| ALXN|2013|     86.29| -0.9300000000000068|
|2013-02-27|85.31|87.32|84.47|86.54|1455350| ALXN|2013|     85.36|  1.1800000000000068|
|2013-02-28|86.24|87.75| 86.0|86.74|1324733| ALXN|2013|     86.54| 0.19999999999998863|
|2013-03-01|86.74|87.07|86.09|86.77|1506792| ALXN|2013|     86.74|0.030000000000001137|
|2013-03-04|86.69|89.93|86.11| 89.9|2328466| ALXN|2013|     86.77|  3.1300000000000097|
|2013-03-05|90.17| 91.9|90.03|91.01|2383043| ALXN|2013|      89.9|  1.1099999999999994|
|2013-03-06|91.21| 93.5| 91.0|93.18|2446183| ALXN|2013|     91.01|  2.1700000000000017|
|2013-03-07| 92.7|93.17|91.46|92.24|1289377| ALXN|2013|     93.18| -0.9400000000000119|
|2013-03-08|92.69|92.99|90.76|91.06|1466183| ALXN|2013|     92.24| -1.1799999999999926|
+----------+-----+-----+-----+-----+-------+-----+----+----------+--------------------+
only showing top 20 rows
```

The Data preparation for performing K- means algorithm, to identify the performing stocks is as mentioned below.
Calculating the mean of "difference" for all the stocks. (Returns)
Calculating the standard deviation of "difference" for all the stocks. (Variance)
Performing the join operation to have Returns, Variance for all the stocks.

```
%pyspark
df_2 = df.groupby('Ticks').agg({'diff':
'mean'}).sort(col("avg(diff)").desc())
df_3 = df.groupby('Ticks').agg({'diff':
'stddev'}).sort(col("stddev(diff)").desc())
%pyspark
from math import sqrt
from pyspark.sql.functions import col
inner_join = df_2.join(df_3, df_2.Ticks ==
df_3.Ticks).select(df_2['Ticks'],df_2['avg(diff)'],df_3['stddev(diff)
'])
finalDf = inner_join.select(
    'Ticks',
    (inner_join['avg(diff)']* 252).alias('Returns'),
    (inner_join['stddev(diff)']* sqrt(252)).alias('Variance')
)
finalDf.show()
```

Output

```
+-----+------------------+------------------+
|Ticks|           Returns|          Variance|
+-----+------------------+------------------+
| PCLN|236.55774424146145| 339.0525844307868|
| AMZN| 231.1494519459889|165.59832073200545|
|GOOGL| 132.5712152501986|147.95479803913338|
| GOOG| 126.6771692307692|156.12945264418852|
|  MTD|  84.2288482922955| 75.97953584596931|
|  AZO| 68.54239872915014|126.84246533360646|
|  BLK| 58.86671961874502| 73.14434068153317|
```

```
| CHTR| 56.57690230341541|  57.9488421114929|
|  NOC|54.357140587768065| 31.58857198784057|
|   BA| 54.35513899920572| 35.41306913026967|
|  LMT|51.524892772041305| 33.79580876007726|
|  SHW| 47.98007942811755|54.031987349015495|
| NFLX| 47.77992057188245| 41.72859607051411|
| NVDA| 43.32038125496425|32.847938182386024|
| ISRG| 41.35482128673549| 54.73454488524473|
| EQIX| 41.30678316123908|  64.6503794135348|
| AVGO| 40.44409849086577| 43.10941398871787|
| ALGN| 40.35202541699762| 35.55764354353066|
|  HII| 37.91609213661636| 33.56747232995161|
|  HUM| 36.73915806195393|43.745854162300816|
+-----+------------------+------------------+
only showing top 20 rows
```

Choosing Returns and Variance as the features and creating a vector to train the model using K-means algorithm.

```
%pyspark
from pyspark.ml.feature import VectorAssembler

vecAssembler = VectorAssembler(inputCols=["Returns", "Variance"],
outputCol="features")
new_final_df = vecAssembler.transform(finalDf)
new_final_df.show()
```

Output

```
+-----+------------------+------------------+--------------------+
|Ticks|           Returns|          Variance|            features|
+-----+------------------+------------------+--------------------+
| PCLN|236.55774424146145| 339.0525844307868|[236.557744241461...|
| AMZN| 231.1494519459889|165.59832073200545|[231.149451945988...|
|GOOGL| 132.5712152501986|147.95479803913338|[132.571215250198...|
| GOOG| 126.6771692307692|156.12945264418852|[126.677169230769...|
|  MTD|  84.2288482922955| 75.97953584596931|[84.2288482922955...|
|  AZO| 68.54239872915014|126.84246533360646|[68.5423987291501...|
|  BLK| 58.86671961874502| 73.14434068153317|[58.8667196187450...|
| CHTR| 56.57690230341541|  57.9488421114929|[56.5769023034154...|
|  NOC|54.357140587768065| 31.58857198784057|[54.3571405877680...|
|   BA| 54.35513899920572| 35.41306913026967|[54.3551389992057...|
|  LMT|51.524892772041305| 33.79580876007726|[51.5248927720413...|
|  SHW| 47.98007942811755|54.031987349015495|[47.9800794281175...|
| NFLX| 47.77992057188245| 41.72859607051411|[47.7799205718824...|
| NVDA| 43.32038125496425|32.847938182386024|[43.3203812549642...|
| ISRG| 41.35482128673549| 54.73454488524473|[41.3548212867354...|
| EQIX| 41.30678316123908|  64.6503794135348|[41.3067831612390...|
| AVGO| 40.44409849086577| 43.10941398871787|[40.4440984908657...|
| ALGN| 40.35202541699762| 35.55764354353066|[40.3520254169976...|
|  HII| 37.91609213661636| 33.56747232995161|[37.9160921366163...|
|  HUM| 36.73915806195393|43.745854162300816|[36.7391580619539...|
+-----+------------------+------------------+--------------------+
only showing top 20 rows
```

Performing K-means algorithm on the datasets to identify different clusters. While k is chosen as 5.

```
%pyspark
from pyspark.ml.clustering import KMeans

kmeans = KMeans(k=5, seed=1)  #  clusters here
model = kmeans.fit(new_final_df.select('features'))
%pyspark
transformed = model.transform(new_final_df)
transformed.show()
```

Output

```
+-----+-----------------+-----------------+--------------------+----------+
|Ticks|          Returns|         Variance|            features|prediction|
+-----+-----------------+-----------------+--------------------+----------+
| PCLN|236.55774424146145| 339.0525844307868|[236.557744241461...|         1|
| AMZN| 231.1494519459889|165.59832073200545|[231.149451945988...|         2|
|GOOGL| 132.5712152501986|147.95479803913338|[132.571215250198...|         2|
| GOOG| 126.6771692307692|156.12945264418852|[126.677169230769...|         2|
|  MTD|  84.2288482922955| 75.97953584596931|[84.2288482922955...|         4|
|  AZO| 68.54239872915014|126.84246533360646|[68.5423987291501...|         2|
|  BLK| 58.86671961874502| 73.14434068153317|[58.8667196187450...|         4|
| CHTR| 56.57690230341541|  57.9488421114929|[56.5769023034154...|         4|
|  NOC|54.357140587768065| 31.58857198784057|[54.3571405877680...|         4|
|   BA| 54.35513899920572| 35.41306913026967|[54.3551389992057...|         4|
|  LMT|51.524892772041305| 33.79580876007726|[51.5248927720413...|         4|
|  SHW| 47.98007942811755|54.031987349015495|[47.9800794281175...|         4|
| NFLX| 47.77992057188245| 41.72859607051411|[47.7799205718824...|         4|
| NVDA| 43.32038125496425|32.847938182386024|[43.3203812549642...|         4|
| ISRG| 41.35482128673549| 54.73454488524473|[41.3548212867354...|         4|
| EQIX| 41.30678316123908|  64.6503794135348|[41.3067831612390...|         4|
| AVGO| 40.44409849086577| 43.10941398871787|[40.4440984908657...|         4|
| ALGN| 40.35202541699762| 35.55764354353066|[40.3520254169976...|         4|
|  HII| 37.91609213661636| 33.56747232995161|[37.9160921366163...|         0|
|  HUM| 36.73915806195393|43.745854162300816|[36.7391580619539...|         4|
+-----+-----------------+-----------------+--------------------+----------+
only showing top 20 rows
```

```
%pyspark
pddf_pred = transformed.toPandas().set_index('Ticks')
pddf_pred.head()
```

Output

```
        Returns  ...  prediction
Ticks            ...
PCLN   236.557744  ...           1
AMZN   231.149452  ...           2
GOOGL  132.571215  ...           2
GOOG   126.677169  ...           2
MTD     84.228848  ...           4
[5 rows x 4 columns]
```

Plotting the scatter plot where the classes (0,1,2,3,4) are pictorially represented in the plot. Distinguished companies are clustered according to their variance and returns, which would help the investor to understand the moving dynamics and the performance of the stocks and create a portfolio.

```
%pyspark
import seaborn as sns
import matplotlib.pyplot as plt
facet = sns.lmplot(data=pddf_pred, x='Returns', y='Variance',
hue='prediction',
                   fit_reg=False, legend=True, legend_out=True)
%pyspark
```

Output

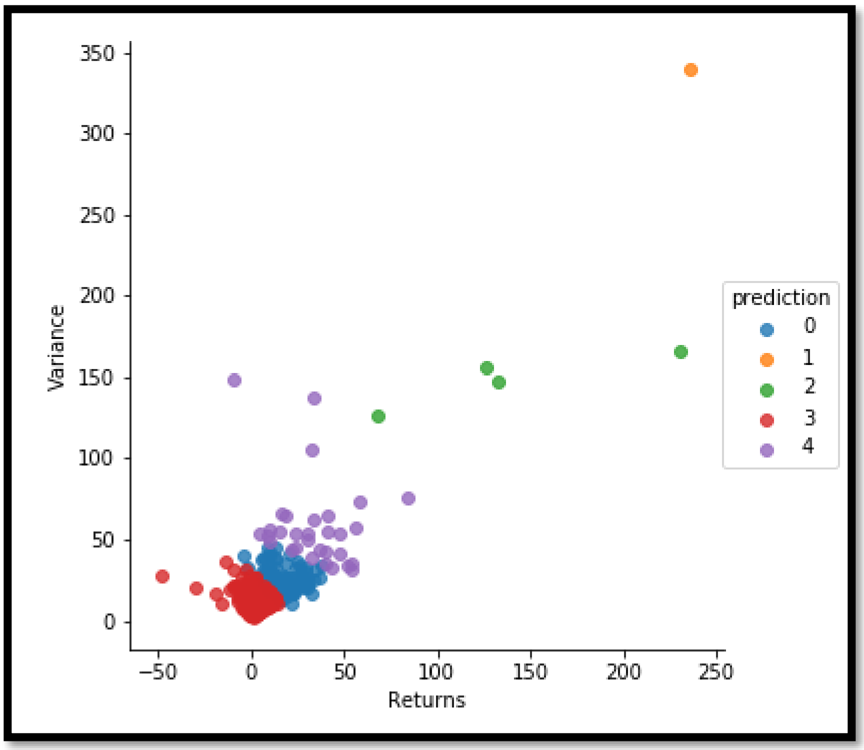

Figure 5 Variance Vs Returns Scatter plot for all stocks