A Project Report on

STOCK DATA ANALYSIS & PREDICTION

Submitted by

Sandeep Reddy Panyala – AP24122060005

Sri Harsha Vardhan Gogisetty – AP24122060007

Eswar Maddi – AP24122060014

DSC 502L

BIG DATA ANALYTICS

Under the guidance of

Dr. Rajiv Senapathi

Assistant Professor

Department of Computer Science and Engineering

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

Dec 2024

Certificate

Date: 02-Dec-24

This is to certify that the work present in this Project entitled "**Stock Data Analysis and Prediction**" has been carried out by **Sandeep Reddy Panyala, Sri Harsha Vardhan Gogisetty, Eswar Maddi** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of **Master of Technology** in School of Engineering and Sciences.

Supervisor

Dr. Rajiv Senapati
Assistant Professor
Department of Computer Science and Engineering.

TABLE OF CONTENTS

1 Abstract

The Stock market analysis and prediction play a pivotal role in understanding market trends and assisting investors in making informed decisions. This project leverages big data technologies like Apache Hadoop, Hive, Sqoop, and PySpark to process and analyze large-scale stock market data. The implementation focuses on aggregating key statistics such as average and maximum values of stock metrics (Close, Volume, Open, High, Low prices) across monthly and yearly timeframes using the MapReduce programming model.

This project explores a comprehensive dataset comprising 12 distinct stocks over a five-year period from 2018 to 2023. The project begins by importing raw stock market data from relational databases into the Hadoop Distributed File System (HDFS) using Sqoop. Hive is then employed as a central data warehouse for managing and querying the datasets, enabling high-level SQL-like queries for efficient data analysis.

MapReduce is implemented to compute key stock metrics grouped by monthly and yearly periods. The Mapper generates grouping keys and emits metric records. The Reducer aggregates these records to compute averages and track maximum values for each metric.

To enhance processing speed and scalability, PySpark is used to implement the analysis in a distributed computing environment. The PySpark API provides an intuitive interface for handling large datasets, ensuring seamless data transformation and parallel computation.

The resulting system integrates stock market data, performs detailed analytics, and provides actionable insights into stock performance trends. By combining Hadoop's storage capabilities, Hive's querying power, Sqoop's data transfer efficiency, and PySpark's computational strength, this project delivers a robust framework for stock price analysis and prediction. This enables better data-driven decision-making in financial markets.

2 Introduction

The stock market, an active marketplace where people and organizations invest in and trade shares, is a fundamental component of current finance. This market provides a platform for wealth generation and diversifying investments in addition to reflecting the economic health of countries. The amount of data produced in the stock market has increased as technological and globalized developments change the financial environment, offering analysts and investors equal opportunities and problems.

The capacity to extract valuable insights from large datasets has become critical in times marked by overload of information. An extremely valuable resource for analyzing market trends and behaviors is historical stock data, which comprises a wide range of measures such ticker symbols, opening and closing prices, daily highs and lows, and trading volumes. Stakeholders may identify trends, evaluate risk, and make well-informed decisions based on information rather than conjecture by analyzing this data.

This project focuses on an extensive dataset that includes 12 different equities over the course of five years, from 2018 to 2023. Through the use of data processing and analytical frameworks, particularly Hive and PySpark, this study seeks to understand the complexities of stock price fluctuations and the fundamental causes of them. For investigating other aspects of stock performance, such as instability, trading volume, and past price movements, the dataset offers a solid starting point.

Key objectives of this analysis include identifying patterns in stock performance, evaluating the impact of market events, and forecasting future price movements. Through exploratory data analysis (EDA) and machine learning methodologies, this research seeks to provide actionable insights that can enhance investment strategies and contribute to a deeper understanding of market dynamics.

3 Literature Review

Using big data analytics and machine learning, Zhihao Peng et al. [1] created a data pipeline based on Cloudera and Hadoop to predict daily growth in U.S. oil stocks. In addition to using real-time stock data from Yahoo Finance, they integrated tools such as Hadoop for distributed storage, Apache Spark for real-time analytics, and Flume for data injection. With an R-squared value of only 0.03 and a Mean Absolute Error (MAE) of 1.95%, the findings of a linear regression model used to predict stock price correlations were subpar, suggesting that linear regression is not appropriate for high-dimensional stock data.

To overcome the difficulties associated with high-dimensional financial data, Jithina Jose et al. [2] provide a strategy for stock prediction and analysis that makes use of Hadoop-based methodologies. The method optimizes trading tactics and improves forecast accuracy by combining logistic regression, genetic algorithms, and linear regression. The authors use a genetic algorithm to evolve the best parameters for trading strategies including Head and Shoulders, Double Bottom, and Double Top by converting NASDAQ order book data into tick data. Simulations conducted in real time demonstrate that these strategies outperform random trading in terms of earnings. The study concludes that an effective strategy for algorithmic trading is to combine evolutionary algorithms with pattern-detection methods.

K V D Kiran et al. [3], discusses a system for analysing historical stock data using Hadoop and Hue. It focuses on the impact of macroeconomic factors like inflation and currency fluctuations on stock performance. The system uses Big Data technologies to handle large datasets, which are imported into MySQL and then transferred to HDFS with Apache Sqoop. Hue provides a graphical interface for easy data analysis without needing complex programming, enabling users to query and visualize stock trends through Hive. The work demonstrates Hadoop's scalability and Hue's user-friendly features for efficient stock market analysis.

Abdelaziz Darwiesh et al. [4] proposes an intelligent risk management system for stock markets, utilizing social media data, particularly Twitter, to enhance risk identification and assessment. The methodology integrates big data analytics and natural language processing (NLP) to analyse user behaviour and sentiments, focusing on risks associated with markets, sessions, and prices. Tools like Twint for data scraping, Spark SQL, and Hadoop for data storage, and advanced NLP techniques are used. The approach identifies risks as financial, operational, or geopolitical, achieving an overall risk analysis accuracy of 73.33%. They have considered the NASDAQ stock market data.

4 Methodology

4.1 MapReduce

The goal of this MapReduce implementation is to process stock market data for various companies, represented in the form of daily records, and generate aggregated statistics for monthly, and yearly timeframes. The output includes averages and maximum values for key metrics such as close price, volume, open price, high price, and low price.

Driver

The Driver is responsible for configuring and executing the MapReduce job.

Key Functions

Define the Mapper and Reducer classes, set input and output paths and configure the output key and value types.

MapReduce Workflow

Mapper

The Mapper processes each row of the dataset and emits key-value pairs based on monthly and yearly groupings.

Key Functions

Parse the input line and extract relevant fields (e.g., Ticker, Date, Close, Volume, etc.).

Generate grouping keys:

Monthly Key: MONTHLY:<Ticker>|<YYYY-MM> (e.g., MONTHLY: AAPL|2023-05).

Yearly Key: YEARLY:<Ticker>|<YYYY> (e.g., YEARLY: AAPL|2023).

Emit the stock metrics (as a string) for each key.

Mapper Output

Key: MONTHLY:<Ticker>|<YYYY-MM> or YEARLY:<Ticker>|<YYYY>.

Value: A CSV-formatted string of metrics: <Close>, <Volume>, <Open>, <High>, <Low>.

Reducer

The Reducer aggregates all values for each key emitted by the Mapper. It computes the average and maximum values for Close, Volume, Open, High, and Low.

Key Functions

Aggregate the metrics:

Sum all values for Close, Volume, Open, High, and Low to calculate averages. Track the maximum values for each metric. Calculate averages by dividing sums by the count of records for the key. Emit the final summarized results.

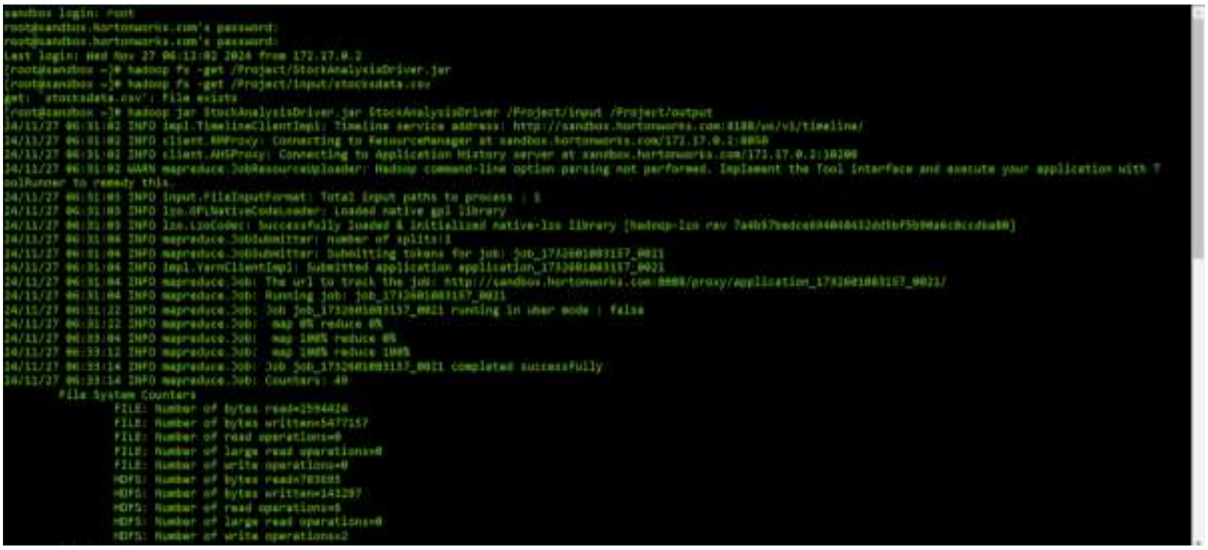
Reducer Output

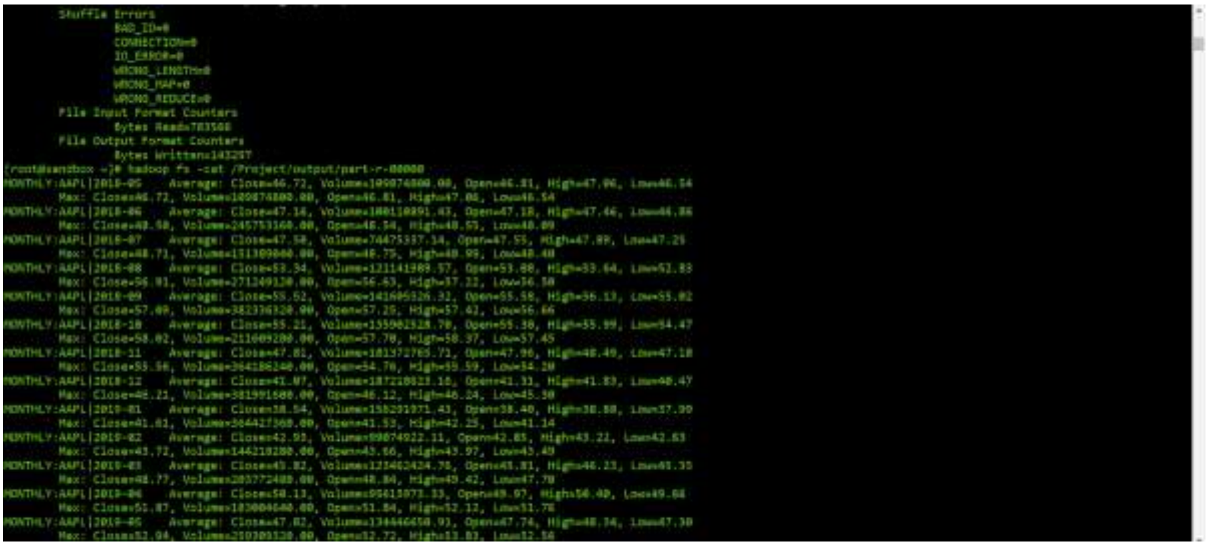
Key: Grouping key (e.g., MONTHLY: AAPL|2023-05).

Value: Two lines:

Average: Average: Close=<val>, Volume=<val>, Open=<val>, High=<val>, Low=<val>.

Maximum: Max: Close=<val>, Volume=<val>, Open=<val>, High=<val>, Low=<val>.





The above output gives the average of all the features of the dataset excluding the Ticker, Dates with respect to each stock, per every month.

```
WEEKLY:AAPL|2023-W22    Average: Close=107.30, Volume=110651359.30, Open=107.19, High=108.49, Low=105.98
        Max: Close=182.01, Volume=426884800.00, Open=182.63, High=182.94, Low=179.12
WEEKLY:AMZN|2023-W22    Average: Close=122.07, Volume=83004993.99, Open=122.13, High=123.63, Low=120.48
        Max: Close=186.57, Volume=311345600.00, Open=187.20, High=188.65, Low=184.84
WEEKLY:BRK-B|2023-W22   Average: Close=247.85, Volume=4656395.01, Open=247.92, High=249.72, Low=245.87
        Max: Close=359.57, Volume=22303500.00, Open=361.39, High=362.10, Low=355.53
WEEKLY:GOOGL|2023-W22   Average: Close=89.61, Volume=34913868.93, Open=89.56, High=90.60, Low=88.57
        Max: Close=149.84, Volume=132345540.00, Open=151.25, High=151.55, Low=148.90
WEEKLY:META|2023-W22    Average: Close=219.33, Volume=23946662.46, Open=219.20, High=222.33, Low=216.27
        Max: Close=382.18, Volume=232316600.00, Open=381.68, High=384.33, Low=378.81
WEEKLY:MSFT|2023-W22    Average: Close=208.68, Volume=30164146.52, Open=208.61, High=210.82, Low=206.32
        Max: Close=343.11, Volume=110945000.00, Open=344.62, High=349.67, Low=342.20
WEEKLY:NVDA|2023-W22    Average: Close=131.66, Volume=46693580.88, Open=131.56, High=134.15, Low=128.92
        Max: Close=401.11, Volume=250920160.00, Open=405.95, High=419.38, Low=399.49
WEEKLY:QQQ|2023-W22     Average: Close=265.77, Volume=48004093.54, Open=265.68, High=267.94, Low=263.27
        Max: Close=403.99, Volume=199448100.00, Open=405.57, High=408.71, Low=402.58
WEEKLY:SPY|2023-W22     Average: Close=355.27, Volume=85723022.48, Open=355.23, High=357.49, Low=352.76
        Max: Close=477.71, Volume=392220700.00, Open=479.22, High=479.98, Low=476.06
WEEKLY:TSLA|2023-W22    Average: Close=144.41, Volume=136459869.00, Open=144.48, High=147.93, Low=140.81
        Max: Close=409.97, Volume=914080943.00, Open=411.47, High=414.50, Low=405.67
WEEKLY:TSM|2023-W22     Average: Close=76.65, Volume=9933653.78, Open=76.70, High=77.56, Low=75.78
        Max: Close=140.66, Volume=60793170.00, Open=141.61, High=145.00, Low=139.42
WEEKLY:V|2023-W22       Average: Close=192.90, Volume=8269259.63, Open=192.90, High=194.85, Low=190.93
        Max: Close=250.93, Volume=38379570.00, Open=250.05, High=252.67, Low=248.22
```

The above output represents the average of all with respect to every stock for the last week.

```
YEARLY:GOOGL|2020        Average: Close=73.95, Volume=39962114.31, Open=73.88, High=74.83, Low=72.98
        Max: Close=91.25, Volume=108357760.00, Open=91.03, High=92.19, Low=90.85
YEARLY:GOOGL|2021        Average: Close=124.23, Volume=30518727.78, Open=124.15, High=125.26, Low=123.03
        Max: Close=149.84, Volume=97881640.00, Open=149.98, High=150.97, Low=148.90
YEARLY:GOOGL|2022        Average: Close=114.76, Volume=34768810.72, Open=114.88, High=116.48, Low=113.20
        Max: Close=148.00, Volume=123199220.00, Open=151.25, High=151.55, Low=145.52
YEARLY:GOOGL|2023        Average: Close=102.05, Volume=37964794.47, Open=101.63, High=103.21, Low=100.57
        Max: Close=125.05, Volume=119455000.00, Open=125.64, High=126.43, Low=122.74
YEARLY:META|2018         Average: Close=167.73, Volume=25439471.62, Open=167.74, High=169.92, Low=165.61
        Max: Close=217.50, Volume=169059900.00, Open=215.72, High=218.62, Low=214.27
YEARLY:META|2019         Average: Close=181.64, Volume=16153162.00, Open=181.57, High=183.45, Low=179.65
        Max: Close=208.10, Volume=77005780.00, Open=208.67, High=208.93, Low=206.59
YEARLY:META|2020         Average: Close=234.55, Volume=22452856.44, Open=234.35, High=237.94, Low=230.79
        Max: Close=303.91, Volume=76343940.00, Open=300.16, High=304.67, Low=293.05
YEARLY:META|2021         Average: Close=321.17, Volume=18868744.48, Open=321.10, High=325.01, Low=317.41
        Max: Close=382.18, Volume=65654040.00, Open=381.68, High=384.33, Low=378.81
YEARLY:META|2022         Average: Close=180.19, Volume=35587379.88, Open=180.24, High=183.84, Low=176.88
        Max: Close=338.54, Volume=232316600.00, Open=339.95, High=343.09, Low=337.19
YEARLY:META|2023         Average: Close=194.56, Volume=28594959.03, Open=193.65, High=197.05, Low=191.57
        Max: Close=264.72, Volume=150475700.00, Open=265.25, High=268.65, Low=261.29
YEARLY:MSFT|2018         Average: Close=106.68, Volume=31684836.08, Open=106.80, High=107.81, Low=105.51
        Max: Close=115.61, Volume=110945000.00, Open=115.42, High=116.18, Low=114.93
YEARLY:MSFT|2019         Average: Close=130.38, Volume=24533254.25, Open=130.34, High=131.23, Low=129.31
        Max: Close=158.96, Volume=55442870.00, Open=159.45, High=159.55, Low=158.22
YEARLY:MSFT|2020         Average: Close=193.03, Volume=37668574.98, Open=192.91, High=195.47, Low=190.38
        Max: Close=231.65, Volume=97073560.00, Open=229.27, High=232.86, Low=227.35
YEARLY:MSFT|2021         Average: Close=275.94, Volume=26014353.21, Open=275.67, High=278.02, Low=273.44
        Max: Close=343.11, Volume=69870640.00, Open=344.62, High=349.67, Low=342.20
YEARLY:MSFT|2022         Average: Close=268.92, Volume=31224383.59, Open=269.11, High=272.50, Low=265.29
        Max: Close=334.75, Volume=90428850.00, Open=335.35, High=338.00, Low=329.78
YEARLY:MSFT|2023         Average: Close=273.96, Volume=30891630.10, Open=273.45, High=276.52, Low=270.87
        Max: Close=332.89, Volume=69527370.00, Open=335.23, High=335.94, Low=330.52
YEARLY:NVDA|2018         Average: Close=57.57, Volume=50047740.43, Open=57.71, High=58.59, Low=56.64
        Max: Close=72.34, Volume=196162880.00, Open=72.33, High=73.19, Low=71.40
YEARLY:NVDA|2019         Average: Close=43.65, Volume=45592350.37, Open=43.61, High=44.25, Low=43.01
        Max: Close=59.84, Volume=250920160.00, Open=60.13, High=60.45, Low=59.60
```

The above output represents the average of all the features of the stock with respect to the year.

4.2 Sqoop

Sqoop stands for SQL-to-Hadoop, is an open-source tool in hadoop ecosystem which is designed for efficiently transferring bulk data in between Apache Hadoop and Structured databases which includes MySQL, PostgreSQL, and Oracle. In this project we have utilized the tool Sqoop, to import the data from the Relational Database Management Systems like MySQL into Hadoop Distributed File System and integrate this imported data into the Hive tool for execution of the queries on the data for betterment of analysing the data. In the below we can see the implementation of importing the data into Hive from the Structured database by leveraging the Sqoop tool.

Initially to connect with the mysql in hadoop we need to use the command of - mysql -u root -h sandbox.hortonworks.com -p, this allows to to connect with the mysql and after executing this command it asks for the password and it is, hadoop. After execution you will be entering into the mysql environment. Below you can see the databases and tables present in a particular database.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| hive               |
| mysql              |
| performance_schema |
| project            |
| ranger             |
+--------------------+
6 rows in set (0.00 sec)
```

```
mysql> use project;
Database changed
mysql> show tables;
+-------------------+
| Tables_in_project |
+-------------------+
| stock             |
+-------------------+
1 row in set (0.00 sec)
```

The data has already been in the project named database, of the table names stocks, by entering the command - select * from stocks limit 5. This command displays a total of 5 rows which are at the top of the table.

```
mysql> LOAD DATA INFILE '/var/lib/mysql-files/stocksdata.csv' INTO TABLE stocks FIELDS TERMINATED
 BY ','  ENCLOSED BY '"' LINES TERMINATED BY '\n' IGNORE 1 LINES;
Query OK, 15108 rows affected, 628 warnings (1.16 sec)
Records: 15108  Deleted: 0  Skipped: 0  Warnings: 628

mysql> select * from stocks limit 5;
+--------+------------+--------+----------+--------+--------+--------+
| Ticker | Dates      | Close  | Volume   | Open   | High   | Low    |
+--------+------------+--------+----------+--------+--------+--------+
| AAPL   | 05/31/2023 | 177.25 | 99625290 | 177.33 | 179.35 | 176.76 |
| AAPL   | 05/30/2023 | 177.30 | 55964400 | 176.96 | 178.99 | 176.57 |
| AAPL   | 05/26/2023 | 175.43 | 54834980 | 173.32 | 175.77 | 173.11 |
| AAPL   | 05/25/2023 | 172.99 | 56058260 | 172.41 | 173.90 | 171.69 |
| AAPL   | 05/24/2023 | 171.84 | 45143490 | 171.09 | 172.42 | 170.52 |
+--------+------------+--------+----------+--------+--------+--------+
5 rows in set (0.01 sec)
```

For the implementation of the Sqoop, exit from the mysql environment. Now in the normal root of SSH Client, you need to enter the below commands for the migration

```
[root@sandbox ~]# sqoop import \
> --connect jdbc:mysql://localhost/project \
> --username root \
> --password hadoop \
> --table stocks \
> --driver com.mysql.jdbc.Driver \
> --hive-import \
> --hive-table stockmarket.detail \
> --m 1
Warning: /usr/hdp/2.5.0.0-1245/accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
24/11/29 13:42:58 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6.2.5.0.0-1245
24/11/29 13:42:58 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure.
 Consider using -P instead.
24/11/29 13:42:58 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for output. You can ove
rride
24/11/29 13:42:58 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by, etc.
24/11/29 13:42:58 WARN sqoop.ConnFactory: Parameter --driver is set to an explicit driver however
 appropriate connection manager is not being set (via --connection-manager). Sqoop is going to fa
ll back to org.apache.sqoop.manager.GenericJdbcManager. Please specify explicitly which connectio
n manager should be used next time.
24/11/29 13:42:58 INFO manager.SqlManager: Using default fetchSize of 1000
24/11/29 13:42:58 INFO tool.CodeGenTool: Beginning code generation
24/11/29 13:42:59 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM stocks AS t W
HERE 1=0
24/11/29 13:42:59 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM stocks AS t W
HERE 1=0
24/11/29 13:42:59 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/hdp/2.5.0.0-1245/hadoop
-mapreduce
Note: /tmp/sqoop-root/compile/624968129ea2db1436685dc1adbda810/stocks.java uses or overrides a de
precated API.
Note: Recompile with -Xlint:deprecation for details.
24/11/29 13:43:05 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-root/compile/62496812
9ea2db1436685dc1adbda810/stocks.jar
24/11/29 13:43:05 INFO mapreduce.ImportJobBase: Beginning import of stocks
24/11/29 13:43:06 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM stocks AS t W
```

of the data from mysql to Hive. By utilizing the sqoop, we would be connecting to the mysql database by entering our database name at the end, for every sandbox execution environments the username will be root, and the password is hadoop, update the table name, create a hive database and a table with column values in it and give the databasename.tablename as the location to store the imported data, at final declare how many mappers should be executed for this process.

```
                FILE: Number of bytes written=162691
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=87
                HDFS: Number of bytes written=779194
                HDFS: Number of read operations=4
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Other local map tasks=1
                Total time spent by all maps in occupied slots (ms)=12972
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=12972
                Total vcore-milliseconds taken by all map tasks=12972
                Total megabyte-milliseconds taken by all map tasks=3243000
        Map-Reduce Framework
                Map input records=15108
                Map output records=15108
                Input split bytes=87
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=193
                CPU time spent (ms)=2360
                Physical memory (bytes) snapshot=145682432
                Virtual memory (bytes) snapshot=1934544896
                Total committed heap usage (bytes)=35651584
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=779194
24/11/29 13:43:59 INFO mapreduce.ImportJobBase: Transferred 760.9316 KB in 52.6457 seconds (14.45
38 KB/sec)
24/11/29 13:43:59 INFO mapreduce.ImportJobBase: Retrieved 15108 records.
```

```
24/11/29 13:47:03 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM stocks AS t W
HERE 1=0
24/11/29 13:47:03 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM stocks AS t W
HERE 1=0
24/11/29 13:47:03 WARN hive.TableDefWriter: Column Close had to be cast to a less precise type in
 Hive
24/11/29 13:47:03 WARN hive.TableDefWriter: Column Open had to be cast to a less precise type in
Hive
24/11/29 13:47:03 WARN hive.TableDefWriter: Column High had to be cast to a less precise type in
Hive
24/11/29 13:47:03 WARN hive.TableDefWriter: Column Low had to be cast to a less precise type in H
ive
24/11/29 13:47:03 INFO hive.HiveImport: Loading uploaded data into Hive

Logging initialized using configuration in jar:file:/usr/hdp/2.5.0.0-1245/hive/lib/hive-common-1.
2.1000.2.5.0.0-1245.jar!/hive-log4j.properties
OK
Time taken: 2.893 seconds
Loading data to table stockmarket.detail
Table stockmarket.detail stats: [numFiles=1, numRows=0, totalSize=779194, rawDataSize=0]
OK
Time taken: 2.844 seconds
```

```
hive> show databases;
OK
default
foodmart
project
stock
stockmarket
xademo
Time taken: 1.596 seconds, Fetched: 6 row(s)
hive> use stockmarket;
OK
Time taken: 0.323 seconds
hive> select * from detail limit 5;
OK
AAPL    05/31/2023      177.25  99625290        177.33  179.35  176.76
AAPL    05/30/2023      177.3   55964400        176.96  178.99  176.57
AAPL    05/26/2023      175.43  54834980        173.32  175.77  173.11
AAPL    05/25/2023      172.99  56058260        172.41  173.9   171.69
AAPL    05/24/2023      171.84  45143490        171.09  172.42  170.52
Time taken: 0.598 seconds, Fetched: 5 row(s)
```

In case if the commands or the names of the database or tables or the locations are given incorrectly then the execution of importing the data via sqoop might fail. In case if all the details have been perfectly matched then the sqoop operation takes place. As you can see in the above ones, it was present in the hive environment and the data has been successfully loaded into the table named detail and of the database named project in the hive environment. Now utilizing this data we would be performing some analysis in the Hive environment.

4.3 Hive

Apache Hive is a data warehouse which is built on top of Hadoop. It is utilized for managing and querying large amounts of datasets in distributed storage systems. Hive helps in providing an abstraction which utilizes MapReduce, which simplifies the process of querying large amounts of datasets. Apache Hive offers a high-level interface for the users which enables them to write the queries like the queries we execute in SQL. Hive was developed by Facebook.

In this project we have utilized Hive as a central data warehouse system which helps in managing the data, querying and processing large datasets. The dataset we included has a total of 12 different types of stocks. The features are Ticker - describes the name of the stock, Dates - describes the timestamp with respect to the stock, Open - describes the price of the stock at the opening session of the market, Close - describes the price of the stock at the closing session of the market, High - describes the highest price of the stock with respect to each stock every day, Low - describes the lowest price of the stock during the trading session of each day and Volume - describes about the total number of shares traded with respect to the stock per day.

Since Hive provides a distributed storage and processing system, it will be effective for handling large volumes of structured data, in the perspective of our project, we have included the Stocks dataset into the Hadoop environment. The dataset consists of around 15000 records of the data. It consists of 5 years of data from 2018 to 2023. Hive scalability helps in good management and processing of this data efficiently. By utilizing the CLI - Command Line Interface provided by the Hive environment which will be like the SQL-like query language, makes it easier for analysing large datasets equipped with MapReduce.

We have included the entire process of the Hive operations we have performed in the Hadoop environment. We have accessed the tools by using the Sandbox Hortonworks platforms. We have loaded the dataset into the HDFS. Open the SSH Client and login into your account by entering the details. Later, type the hive in the command interface so that the environment changes from the normal to hive interface.

```
sandbox login: root
root@sandbox.hortonworks.com's password:
Last login: Wed Nov 27 03:48:22 2024 from 172.17.0.2
[root@sandbox ~]# hive

Logging initialized using configuration in file:/etc/hive/2.5.0.0-1245/0/hive-log4j.properties
hive> show databases;
```

After clicking the hive command, we can see that the interface has been changed from the [root@sandbox ~] # to hive>. This Hive acts so like the SQL interface. First, we need to know the databases present in the environment, if we require any database with a new one, we can create a new database. For knowing the existing databases, we have the command hive> show databases; this command displays all the existing databases present in the Hive instance. For creating a new database, we can use the command hive> create ‘database database_name’. This creates a new database in the Hive instance. I have created a new database without merging all the files into one database.

```
hive> show databases;
OK
default
foodmart
project
stock
stockmarket
xademo
Time taken: 2.123 seconds, Fetched: 6 row(s)
hive> use stockmarket;
OK
Time taken: 0.54 seconds
```

I have created a database with name stock market. For performing operations in the database, we use the command hive> ‘use database_name’. This command allows us to use the database and perform operations within the database. For knowing the existing tables residing in the database we use command hive> ‘show tables. This command lists out all the tables present in the database. Since its a newly created database there aren’t any tables in the database. For creating a table, we use the command hive> ’create table table_name(column1_name datatype, column2_name datatype, column3_name datatype, ...);’. Here we are loading a stocks dataset into the table, which is in csv format, so we add additional lines for the above query with including ’row format delimited fields terminated by’,’’ and ’TBLPROPERTIES (’skip.header.line.count’=’1’);

```
hive> create table details(Ticker STRING, Dates STRING,Close DECIMAL,Volume INT,Open DECIMAL,High DECIMAL,Low DECIMAL)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 0.97 seconds
```

The above lines indicate – row format refers to how individual records (or rows) of data are organized or stored. Delimited fields refers that the data in each row is broken into fields (or columns) based on a specific delimiter. A delimiter is a character that marks the boundary between different fields (or data values) within the row. Terminated by ',': This part specifies that the delimiter separating the fields in a row is a comma (,). Upon creation of the table to check the column fileds and datatypes are correctly specified, we can use the command hive> ‘describe table_name’. This displays the columns and the related datatypes of the column which are present in the table.

```
hive> describe details;
OK
ticker                  string
dates                   string
close                   decimal(10,0)
volume                  int
open                    decimal(10,0)
high                    decimal(10,0)
low                     decimal(10,0)
Time taken: 0.841 seconds, Fetched: 7 row(s)
```

After that by using the command hive> load data local inpath 'dataset path' into table table_name; we can load the dataset file into the table which is created in the database of Hive instance. For every successful execution of the command, we can check the OK message which is being projected on the screen.

```
hive> LOAD DATA LOCAL INPATH 'stocksdata.csv' INTO TABLE details;
Loading data to table stockmarket.details
Table stockmarket.details stats: [numFiles=1, numRows=0, totalSize=783566, rawDataSize=0]
OK
Time taken: 1.924 seconds
hive> SELECT * FROM details LIMIT 5;
OK
AAPL    05/31/2023      177     99625290        177     179     177
AAPL    05/30/2023      177     55964400        177     179     177
AAPL    05/26/2023      175     54834980        173     176     173
AAPL    05/25/2023      173     56058260        172     174     172
AAPL    05/24/2023      172     45143490        171     172     171
Time taken: 0.385 seconds, Fetched: 5 row(s)
```

We can check if the data is correctly loaded or not by using the command hive> `SELECT * FROM table_name', this command lists out all the records which are present in the table. In case if you need to display a few numbers of records of the table we can use the command, hive> `SELECT * FROM table_name limit 10(number of records)'. This command displays only the top 10 records of the table. In the above I have passed the query to display the top 5 records of the table. Up to now this is the process of the changing environment to hive, creating and managing the database, table creation and loading of the dataset into tables. For making some decisions on the considered data we need to perform some analysis which helps in understanding the existing data better.

For processing the table contents, we first checked the total number of records associated with each unique stock name. So, we have used the below command from the image to return the total number of records associated with each stock name in the table. As you can see the query returns the output of the number of records associated with each stock name. There is a total of 12 stocks present in the table and each stock has the same number of records associated with value of 1259.

```
hive> select ticker, count(*) as record_count
    > from details
    > group by ticker;
Query ID = root_20241127043414_0807fdd5-74d8-4c6b-b53c-86c99c2311d1
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.


Status: Running (Executing on YARN cluster with App id application_1732601083157_0015)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .........     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......    SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 25.98 s
----------------------------------------------------------------------------------------------
OK
AAPL    1259
AMZN    1259
BRK-B   1259
GOOGL   1259
META    1259
MSFT    1259
NVDA    1259
QQQ     1259
SPY     1259
TSLA    1259
TSM     1259
V       1259
Time taken: 44.364 seconds, Fetched: 12 row(s)
```

For further we have tried to analyse the Closing value of each stock in the table. We found the highest and lowest closing prices for each of the stock. This query groups the data by each stock (ticker symbol) and calculates the maximum and minimum closing prices (close) for that stock. This helps identify the price range a stock has traded within. Such insights are useful for investors or analysts to understand stock performance over time.

```
hive> select ticker, MAX(Close) as max_close_price, MIN(Close) as min_close_price
    > from details
    > group by ticker;
Query ID = root_20241127043634_649470a8-10b9-4cf3-8116-61d867bf4268
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1732601083157_0015)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 11.99 s
----------------------------------------------------------------------------------------------
OK
AAPL    182     36
AMZN    187     67
BRK-B   360     162
GOOGL   150     49
META    382     89
MSFT    343     94
NVDA    401     32
QQQ     404     144
SPY     478     223
TSLA    410     12
TSM     141     34
V       251     122
Time taken: 13.578 seconds, Fetched: 12 row(s)
```

```
hive> select year,ticker,avg_price_range
    > from(select YEAR(FROM_UNIXTIME(UNIX_TIMESTAMP(dates,'MM/dd/yyyy')))AS year,
    > ticker,
    > AVG(high-low) AS avg_price_range,
    > ROW_NUMBER() OVER (PARTITION BY YEAR(FROM_UNIXTIME(UNIX_TIMESTAMP(dates,'MM/dd/yyyy'))) ORDER BY AVG(high-low) DESC) AS rank
    > FROM details
    > GROUP BY YEAR(FROM_UNIXTIME(UNIX_TIMESTAMP(dates,'MM/dd/yyyy'))),ticker
    > ) ranked_data
    > WHERE rank=1;
Query ID = root_20241127050186_f778d3c4-f9a4-4de8-81c5-c0a1e90a2e7b
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.


Status: Running (Executing on YARN cluster with App id application_1732601083157_0016)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
Reducer 3 ......   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 37.74 s
----------------------------------------------------------------------------------------------
OK
NULL    TSLA    7.4435
2018    META    4.4783
2019    META    3.8212
2020    META    7.0129
2021    TSLA    10.8289
2022    TSLA    13.7815
2023    NVDA    9.1774
Time taken: 67.327 seconds, Fetched: 7 row(s)
hive>
```

The above query identifies the stocks name column ticker with the highest average daily price range (high - low) for each year in the dataset. It calculates the average price range for each ticker by grouping the data by year and ticker and using the AVG () function. The ROW_NUMBER () function ranks tickers within each year based on their average price range, with rank 1 assigned to the highest. The outer query filters the results to only include the top-ranked ticker (highest average price range) for each year. This query helps analysts determine the most volatile stock annually, providing insights into price movement trends and potential opportunities for traders.

```
hive> select d.ticker,d.dates,d.open from details d
    > JOIN(SELECT ticker, MAX(open) AS max_open
    > FROM details
    > GROUP BY ticker
    > ) m
    > ON d.ticker=m.ticker AND d.open=m.max_open;
Query ID = root_20241127052401_8a865287-3d89-4c64-9b8e-197a83ff0d92
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.


Status: Running (Executing on YARN cluster with App id application_1732601083157_0017)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
Map 3 ..........   SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 29.06 s
----------------------------------------------------------------------------------------------
OK
AAPL    01-04-2022      183
AMZN    07-12-2021      187
BRK-B   03/29/2022      361
BRK-B   03/28/2022      361
GOOGL   02-02-2022      151
META    09/13/2021      382
META    09-02-2021      382
MSFT    11/22/2021      345
NVDA    05/30/2023      406
QQQ     11/22/2021      406
SPY     01-04-2022      479
TSLA    11-04-2021      411
TSM     02/16/2021      142
V       07/28/2021      250
V       07/16/2021      250
```

The above query retrieves the ticker, date, and opening price for the record with the highest opening price (Open – column name) for each stock ticker in the dataset. By using a subquery to calculate the maximum opening price for each ticker, and then joining it back with the original dataset, it ensures that the full record corresponding to this maximum value is fetched. This is particularly useful for identifying the day each stock opened at its highest price, which can provide valuable insights for historical analysis, peak performance evaluation, and identifying trading opportunities.

```
FAILED: SemanticException [Error 10004]: Line 1:232 Invalid table alias or column reference
hive> select d.ticker,d.dates,d.Close from details d
    > JOIN(SELECT ticker,MAX(Close) AS max_close
    > FROM details
    > GROUP BY ticker) m
    > ON d.ticker=m.ticker AND d.open=m.max_close;
Query ID = root_20241129151217_d2787099-65dd-473b-8112-312728596c16
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.


Status: Running (Executing on YARN cluster with App id application_1732883356534_0008)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
Map 3 ..........   SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 34.44 s
----------------------------------------------------------------------------------------------
OK
AMZN    07-12-2021      186
GOOGL   11/19/2021      149
GOOGL   11-08-2021      149
META    09/13/2021      377
META    09-02-2021      375
MSFT    12/28/2021      341
MSFT    11/19/2021      343
QQQ     12/28/2021      402
SPY     12/30/2021      476
SPY     12/28/2021      477
TSM     01/13/2022      139
Time taken: 43.892 seconds, Fetched: 11 row(s)
```

The above query returns the highest value of the closing price with respect to every stock included in the dataset, displaying the timestamp that on which date, that the stock closing price is high.

```
hive> SELECT d.ticker,d.dates,d.high, YEAR(FROM_UNIXTIME(UNIX_TIMESTAMP(d.dates,'MM/dd/yyyy'))) AS year
    > FROM details d
    > JOIN (
    > SELECT
    > ticker,
    > YEAR(FROM_UNIXTIME(UNIX_TIMESTAMP(dates,'MM/dd/yyyy'))) AS year,
    > MAX(high) AS max_high
    > FROM details
    > GROUP BY ticker, YEAR(FROM_UNIXTIME(UNIX_TIMESTAMP(dates,'MM/dd/yyyy')))
    > ) m
    > ON d.ticker=m.ticker
    > AND YEAR(FROM_UNIXTIME(UNIX_TIMESTAMP(d.dates,'MM/dd/yyyy')))=m.year
    > AND d.high=m.max_high;
Query ID = root_20241127053628_33cb3aff-0bb3-488b-b89a-ce27d42a372a
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.


Status: Running (Executing on YARN cluster with App id application_1732601083157_0018)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
Map 3 ..........   SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 23.57 s
----------------------------------------------------------------------------------------------
OK
```

AMZN	09/28/2018	101	2018
AMZN	09/27/2018	101	2018
AMZN	08/31/2018	101	2018
AMZN	08/30/2018	101	2018
AMZN	07/17/2019	101	2019
AMZN	07/16/2019	101	2019
AMZN	07/15/2019	101	2019
AMZN	10/13/2020	175	2020
AMZN	08/31/2020	175	2020
AMZN	07/13/2021	189	2021
AMZN	03/29/2022	171	2022
AMZN	05/30/2023	123	2023
BRK-B	09/21/2018	223	2018
BRK-B	09/20/2018	223	2018
BRK-B	12/20/2019	228	2019
BRK-B	12/17/2019	228	2019
BRK-B	12/16/2019	228	2019
BRK-B	11/24/2020	235	2020
BRK-B	11/17/2020	235	2020
BRK-B	12/30/2021	302	2021
BRK-B	12/16/2021	302	2021

The above query helps in identifying the highest price (high) recorded by each stock ticker for every year in the dataset, along with the date it occurred. It achieves this by first calculating the maximum high price for each ticker and year combination using a subquery. The main query then joins this result back with the original dataset to retrieve the full record (including the date and price) corresponding to these maximum values. This query is particularly useful for annual performance analysis of stocks, allowing analysts or investors to pinpoint peak price movements for each stock in any given year. It provides valuable insights into market trends, helps identify the most profitable trading days, and offers context for historical comparisons across years. The results can also be used to assess volatility or the impact of specific events on stock performance in a particular year.

4.4 Pyspark

Importing packages:

In python, to use the spark module we need to install the pyspark package using the pip command.

PySpark is a Python API that allows users to interact with Apache Spark, an open-source distributed computing framework

pip install pyspark

4.4.1 Loading of Data

Creating spark session

SparkSession is an entry point into all functionality in Spark, and it is required if we want to build dataframe operations, SQL queries, and machine learning tasks in PySpark.

appName specifies a name for the application, in this case, "Stock Price Analysis." This helps in identifying the application when monitoring tasks in the Spark UI.

getOrCreate ensures that a Spark session is either created if it doesn't already exist or retrieves an existing one.

Reading the Stock Data

read.csv () Specifies the path to the directory or file containing the stock data.

show () displays the first five rows of the DataFrame in a tabular format, helping to visually inspect the data.

printSchema() defines the structure and data types of each column.

4.4.2 Data Preprocessing

Parsing Date Column

To ensure accurate analysis, the stock data underwent a series of data cleaning steps using PySpark. These steps involved parsing dates, converting numerical columns, and changing data types for consistency.

The ParsedDate column is created by converting the string format ("MM/DD/YYYY") into a DateType using a user-defined function (UDF). This allows for efficient date-based operations such as filtering, aggregation, and sorting.

2.3.2 Parsing Numerical Columns

Many numerical columns in the dataset, such as Open, Close/Last, Low, and High, contained string representations (e.g., "$123.45") or required normalization. These were cleaned and converted to FloatType

The num_parser function strips dollar signs and converts the values into floats.

The withColumn() method was used to apply this transformation to each relevant column.

This ensures that the columns are numeric, facilitating mathematical operations like calculations and aggregations.

Converting Volume to Integer Type

The Volume column, often stored as a string, needs to be converted to an IntegerType to align with its numerical nature.

A UDF (parse_int) is created to parse and convert values to integers.

The .withColumn() method is used to apply this transformation.

After the data cleaning procedures, the schema was inspected to confirm the successful transformations:

Dates were parsed and stored as DateType.

Numerical columns like Open, Close, Low, High, and Volume were appropriately typed as FloatType or IntegerType.

The data cleaning steps ensures

- Proper data types for analysis.
- Improved data consistency and usability.
- Elimination of formatting issues, enabling seamless integration into analytical workflows.

4.4.3 Exploratory Data Analysis

After cleaning the stock data, EDA techniques were applied to understand the dataset's characteristics and derive meaningful insights.

summary	Volume	Open	Low	High	Close
count	15108	15108	15108	15108	15108
mean	5.1868408793685466E7	180.09656566181036	177.9982781513109	182.1253348687101	180.1256089860054
stddev	5.496484129953464E7	101.16125813324383	100.26590135955234	101.96625521621753	101.1489178216854З
min	961133	12.07	11.8	12.45	11.93
max	914080943	479.22	476.06	479.98	477.71

The method describe () is used to provide the basic descriptive statistics such as count, mean, standard deviation, minimum, and maximum values for selected columns (Volume, Open, Low, High, and Close).

This helps in understanding the data's central tendencies, spread, and outliers.

Calculating Maximum Stock Price for Various Stocks

cleaned_stocks.groupBy("Ticker").max("Open"). show (15)

```
+------+---------+
|Ticker|max(Open)|
+------+---------+
| BRK-B|   361.39|
|  AAPL|   182.63|
|  META|   381.68|
|  TSLA|   411.47|
|  AMZN|    187.2|
|  MSFT|   344.62|
|   TSM|   141.61|
|   QQQ|   405.57|
|     V|   250.05|
| GOOGL|   151.25|
|   SPY|   479.22|
|  NVDA|   405.95|
+------+---------+
```

Groups the data by the Ticker column (stock identifier) and calculates the maximum opening price.

This highlights the peak stock price for each stock, useful for identifying top-performing stocks.

Aggregating Stock Statistics

```
cleaned_stocks.groupBy("Ticker").agg(
    func.max("Open").alias("MaxStockPrice"),
    func.sum("Volume").alias("TotalVolume")
).show(15)
```

```
+------+-------------+------------+
|Ticker|MaxStockPrice| TotalVolume|
+------+-------------+------------+
| BRK-B|       361.39|  5862401321|
|  AAPL|       182.63|139310061360|
|  META|       381.68| 30148848043|
|  TSLA|       411.47|171802975076|
|  AMZN|        187.2|104503287430|
|  MSFT|       344.62| 37976660472|
|   TSM|       141.61| 12506470104|
|   QQQ|       405.57| 60437153773|
|     V|       250.05| 10410997871|
| GOOGL|       151.25| 43956560981|
|   SPY|       479.22|107925285300|
|  NVDA|       405.95| 58787218324|
+------+-------------+------------+
```

func.max("Open"): Computes the maximum opening price (MaxStockPrice) for each stock.

func.sum("Volume"): Aggregates the total trading volume (TotalVolume) for each stock.

Combines price and volume metrics, providing a more comprehensive view of stock performance.

Date Manipulation for Yearly Maximum Prices

cleaned_stocks=(cleaned_stocks.withColumn("Year", func.year(cleaned_stocks.ParsedDate)).withColumn("Month",func.month(cleaned_stocks.ParsedDate)).withColumn("Day",func.dayofmonth(cleaned_stocks.ParsedDate)).withColumn("Week", func.weekofyear(cleaned_stocks.ParsedDate)))

```
+------+----------+-------+------+------+------+------+----+-----+---+----+
|Ticker|ParsedDate| Volume|  Open|   Low|  High| Close|Year|Month|Day|Week|
+------+----------+-------+------+------+------+------+----+-----+---+----+
| BRK-B|2023-05-31|6175417|321.12|319.39|322.41|321.08|2023|    5| 31|  22|
| BRK-B|2023-05-30|3232461|321.86| 319.0|322.47|322.19|2023|    5| 30|  22|
| BRK-B|2023-05-26|3229873|320.44|319.67|322.63| 320.6|2023|    5| 26|  21|
| BRK-B|2023-05-25|4251935|320.56|317.71|320.56|319.02|2023|    5| 25|  21|
| BRK-B|2023-05-24|3075393|322.71|319.56| 323.0| 320.2|2023|    5| 24|  21|
| BRK-B|2023-05-23|4031342|328.19|322.97|329.27|323.11|2023|    5| 23|  21|
| BRK-B|2023-05-22|2763422|330.75|328.35|331.49|329.13|2023|    5| 22|  21|
| BRK-B|2023-05-19|4323538| 331.0|329.12|333.94|330.39|2023|    5| 19|  20|
| BRK-B|2023-05-18|2808329|326.87|325.85|329.98|329.76|2023|    5| 18|  20|
| BRK-B|2023-05-17|3047626|325.02|324.82|328.26|327.39|2023|    5| 17|  20|
+------+----------+-------+------+------+------+------+----+-----+---+----+
only showing top 10 rows
```

Year: Extracts the year from the ParsedDate column.

Month: Extracts the month for monthly analysis.

Day: Extracts the specific day of the month.

Week: Identifies the week number of the year.

This enables time-based analysis, such as finding the maximum stock price for each year or observing seasonal trends.

Yearly Stock Price Analysis

```
yearly = cleaned_stocks.groupBy(['Ticker', 'Year']).agg(func.max("Open").alias("YearlyHigh"), func.min("Open").alias("YearlyLow"))
yearly.show()
```

```
+------+----+----------+---------+
|Ticker|Year|YearlyHigh|YearlyLow|
+------+----+----------+---------+
|  META|2020|    300.16|   139.75|
| BRK-B|2023|     331.0|   294.68|
|  MSFT|2019|    159.45|    99.55|
|  MSFT|2021|    344.62|   212.17|
| BRK-B|2018|     224.0|   185.43|
|  META|2021|    381.68|    247.9|
|  TSLA|2019|      29.0|    12.07|
|  META|2018|    215.72|    123.1|
|  AAPL|2020|    138.05|    57.02|
|  MSFT|2020|    229.27|   137.01|
|  TSLA|2021|    411.47|   184.18|
| BRK-B|2021|    300.88|   228.21|
|  TSLA|2018|      25.0|    17.02|
|  MSFT|2018|    115.42|    95.14|
|  AMZN|2020|    177.35|    82.08|
|  META|2022|    339.95|    90.08|
|  AAPL|2022|    182.63|   127.99|
|  META|2019|    208.67|   128.99|
|  TSLA|2020|    233.33|    24.98|
|  AMZN|2022|    170.44|     82.8|
+------+----+----------+---------+
only showing top 20 rows
```

func.max("Open"): Calculates the highest opening price for each stock in each year (YearlyHigh).

func.min("Open"): Calculates the lowest opening price for each stock in each year (YearlyLow).

A summary of the annual performance of each stock. This highlights the range of stock price fluctuations over the year, helping identify high-performing and volatile stocks.

Monthly Stock Price Analysis

```
monthly = cleaned_stocks.groupBy(['Ticker', 'Year', 'Month']).agg(
    func.max("Open"). alias("MonthHigh"),
    func.min("Open"). alias("MonthLow")
)
monthly.show()
```

```
+------+----+-----+---------+--------+
|Ticker|Year|Month|MonthHigh|MonthLow|
+------+----+-----+---------+--------+
| BRK-B|2022|   10|   297.98|  260.58|
|  META|2020|    6|   241.28|  209.75|
| BRK-B|2018|    9|   222.13|  209.21|
|  MSFT|2022|    6|    275.2|  243.86|
|  MSFT|2021|    2|   245.03|  230.01|
|  MSFT|2020|    1|   174.05|  157.08|
| BRK-B|2021|   10|   290.85|  273.02|
| BRK-B|2020|   10|   216.74|  200.03|
|  TSLA|2023|    4|   199.91|  152.64|
|  TSLA|2019|    4|    19.22|   15.72|
|  AMZN|2018|    5|    81.15|   81.15|
|  MSFT|2019|    6|   137.45|  121.28|
|  META|2023|    4|   239.89|  208.84|
|  AMZN|2020|    2|   108.65|   90.73|
|  MSFT|2022|   10|   247.93|  219.85|
|  AMZN|2022|    3|   170.38|  136.68|
|  TSLA|2020|   10|   151.48|  135.63|
| BRK-B|2019|    9|   212.24|  201.19|
|  TSLA|2021|   12|    386.9|  303.57|
| BRK-B|2021|    6|   292.91|   275.0|
+------+----+-----+---------+--------+
only showing top 20 rows
```

func.max("Open"): Calculates the highest opening price for each stock in each month (MonthHigh).

func.min("Open"): Calculates the lowest opening price for each stock in each month (MonthLow).

This captures monthly trends and seasonality in stock prices and helps identify periods of significant price changes or seasonal patterns.

Weekly Stock Price Analysis

```
weekly = cleaned_stocks.groupBy(['Ticker', 'Year', 'Week']).agg(
   func.max("Open"). alias("WeekHigh"),
   func.min("Open"). alias("WeekLow")
)
weekly.show()
```

```
+------+----+----+--------+-------+
|Ticker|Year|Week|WeekHigh|WeekLow|
+------+----+----+--------+-------+
| BRK-B|2022|  14|   352.0| 341.17|
| BRK-B|2022|  10|  326.59| 322.49|
| BRK-B|2021|  14|  264.22| 260.02|
|  META|2022|  43|  131.68|  97.98|
|  META|2020|   6|  212.51| 203.44|
|  TSLA|2022|  20|  255.72| 235.67|
|  TSLA|2020|  19|   52.92|  46.73|
|  TSLA|2020|  16|   51.49|  39.34|
|  TSLA|2018|  39|   20.86|  18.02|
| BRK-B|2018|  48|  217.23|  209.3|
|  MSFT|2022|   6|  309.87| 301.25|
|  MSFT|2021|   2|  218.47| 213.52|
|  META|2022|  40|  140.49| 136.76|
|  AAPL|2020|  27|   91.96|  88.31|
| BRK-B|2020|  19|  180.05|  173.4|
|  MSFT|2020|   1|  158.78| 158.32|
|  META|2020|  36|  298.88| 287.25|
|  AAPL|2021|  25|  134.45|  130.3|
|  AAPL|2020|  46|   120.5| 115.55|
| BRK-B|2021|  32|  291.81| 287.01|
+------+----+----+--------+-------+
only showing top 20 rows
```

func.max("Open"): Finds the highest opening price for each stock in each week (WeekHigh).

func.min("Open"): Finds the lowest opening price for each stock in each week (WeekLow).

This captures weekly price variations, which are critical for short-term trading strategies and highlights short-term trends and market behavior.

Calculating Weekly Price Spread

```
weekly.withColumn("Spread", weekly['WeekHigh'] - weekly['WeekLow']).show()
```

```
+------+----+----+--------+-------+----------+
|Ticker|Year|Week|WeekHigh|WeekLow|    Spread|
+------+----+----+--------+-------+----------+
| BRK-B|2022|  14|   352.0| 341.17| 10.829987|
| BRK-B|2022|  10|  326.59| 322.49|  4.100006|
| BRK-B|2021|  14|  264.22| 260.02|  4.200012|
|  META|2022|  43|  131.68|  97.98|  33.69999|
|  META|2020|   6|  212.51| 203.44|  9.069992|
|  TSLA|2022|  20|  255.72| 235.67| 20.050003|
|  TSLA|2020|  19|   52.92|  46.73| 6.1899986|
|  TSLA|2020|  16|   51.49|  39.34| 12.150002|
|  TSLA|2018|  39|   20.86|  18.02| 2.8400002|
| BRK-B|2018|  48|  217.23|  209.3| 7.9299927|
|  MSFT|2022|   6|  309.87| 301.25|  8.619995|
|  MSFT|2021|   2|  218.47| 213.52|  4.949997|
|  META|2022|  40|  140.49| 136.76|  3.730011|
|  AAPL|2020|  27|   91.96|  88.31| 3.6500015|
| BRK-B|2020|  19|  180.05|  173.4|  6.650009|
|  MSFT|2020|   1|  158.78| 158.32|0.45999146|
|  META|2020|  36|  298.88| 287.25| 11.630005|
|  AAPL|2021|  25|  134.45|  130.3|  4.149994|
|  AAPL|2020|  46|   120.5| 115.55|  4.949997|
| BRK-B|2021|  32|  291.81| 287.01|  4.799988|
+------+----+----+--------+-------+----------+
only showing top 20 rows
```

A new column Spread was added, representing the range of price variation for each week.

This quantifies weekly volatility, aiding in risk assessment for short-term investors and allows identification of highly volatile stocks or periods with significant price changes.

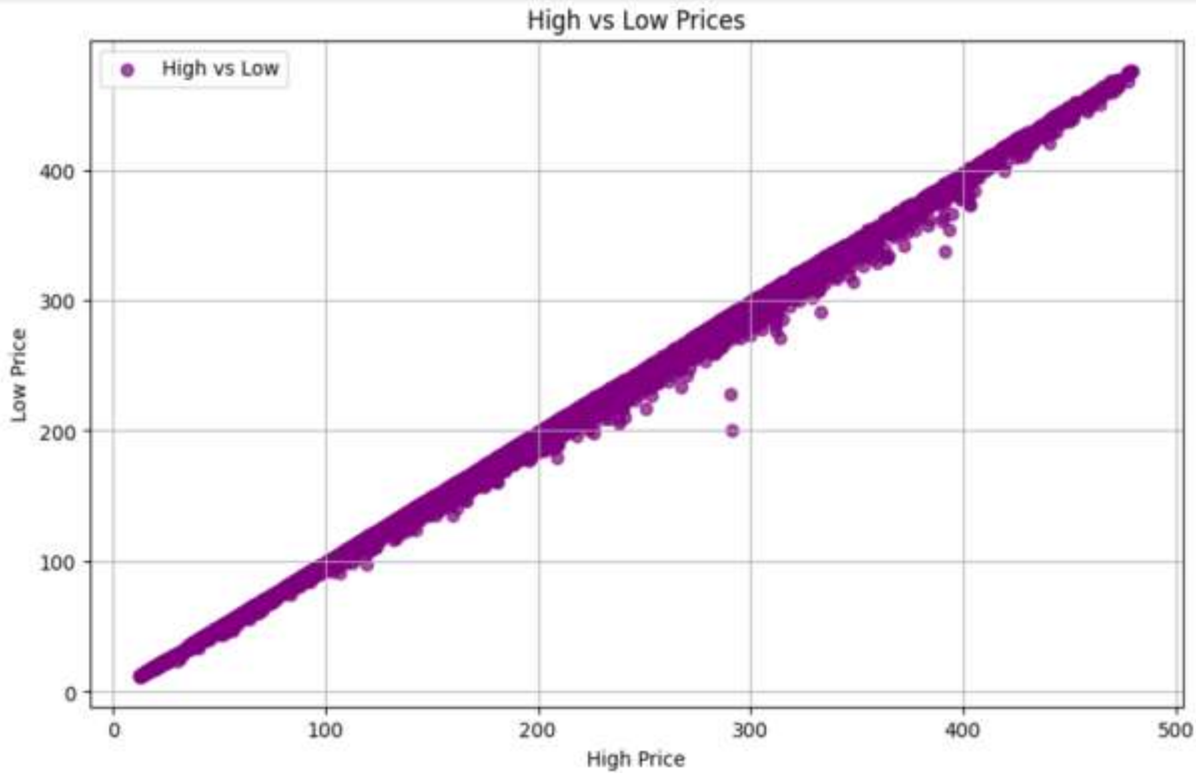
The stock data after data processing:

Ticker	Year	Month	Day	Week	Volume	Open	Low	High	Close	YearlyHigh	YearlyLow	WeekHigh	WeekLow	MonthHigh	MonthLow
BRK-B	2023	5	31	22	6175417	321.12	319.39	322.41	321.08	331.0	294.68	321.86	321.12	331.0	320.44
BRK-B	2023	5	30	22	3232461	321.86	319.0	322.47	322.19	331.0	294.68	321.86	321.12	331.0	320.44
BRK-B	2023	5	26	21	3229873	320.44	319.67	322.63	320.6	331.0	294.68	330.75	320.44	331.0	320.44
BRK-B	2023	5	25	21	4251935	320.56	317.71	320.56	319.02	331.0	294.68	330.75	320.44	331.0	320.44
BRK-B	2023	5	24	21	3075393	322.71	319.56	323.0	320.2	331.0	294.68	330.75	320.44	331.0	320.44

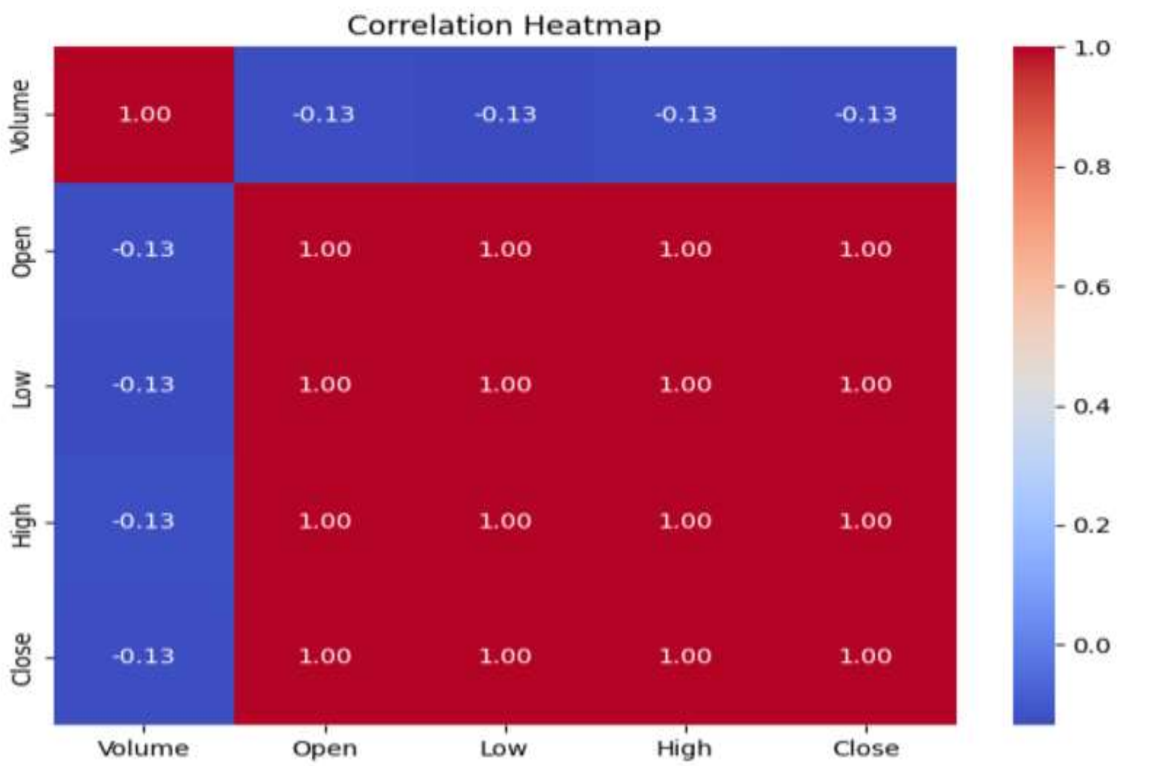
only showing top 5 rows

Plotting of close price vs Index:

Plotting of High vs Low Prices:

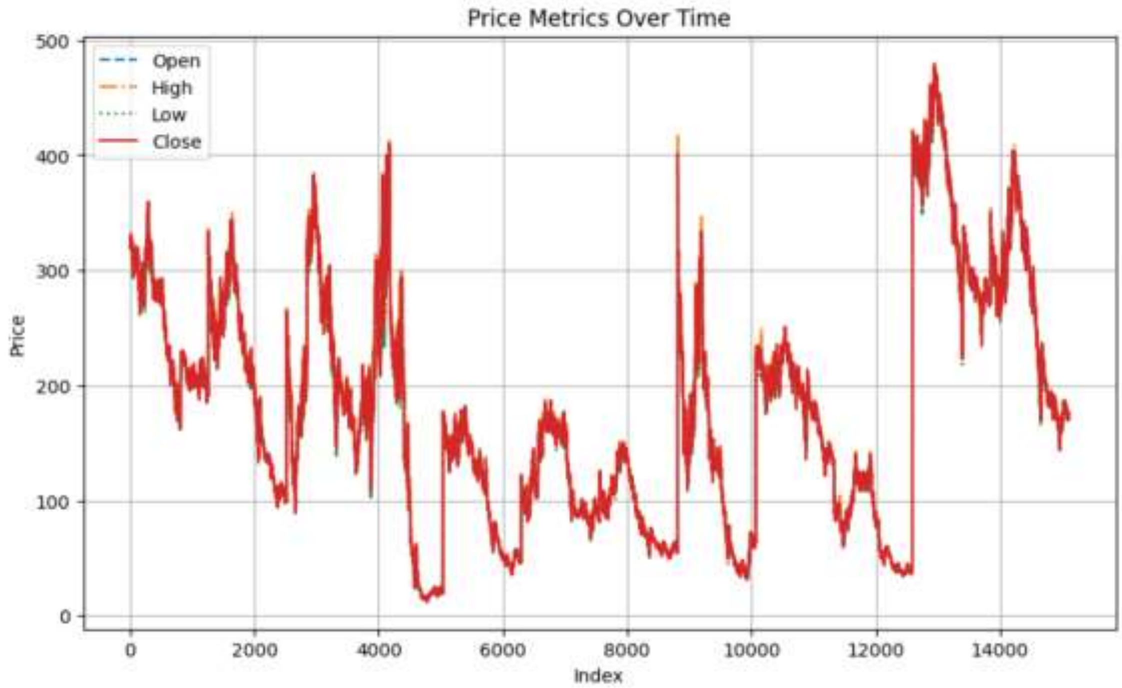


Correlation Heatmap:



The correlation heatmap identifies the correlation between the features the correlation is found for the features Volume, Close, Open, High, Low. It ranges from –1 to 1 where –1 refers to features are negatively correlated, 0 refers to no correlation, 1 refers to positive correlation.

Price Metrics over Time vs Index plotting:



By plotting a stock's price metrics (e.g., Open, Close, High, and Low) alongside the index we can identify whether the stock is outperformance or underperformance. If the stock price consistently increases at a steeper rate than the index, it indicates the stock is outperforming the market. If the stock lags the index or declines when the index rises, it suggests underperformance.

4.4.4 Model Selection

Feature Engineering

The first step involved assembling relevant features into a single vector. The selected features were derived from exploratory data analysis (EDA)

Features Description:

Volume: Total stock volume traded.

Open, Low, High: Intraday price metrics.

YearlyHigh, YearlyLow: Maximum and minimum stock prices for the year.

WeekHigh, WeekLow: Weekly price extremes.

MonthHigh, MonthLow: Monthly price extremes.

The VectorAssembler was used to combine these features into a single column, features, for use in machine learning:

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

final_stocks = assembler.transform(final_stocks)

Data Preparation

The dataset was split into training and testing subsets:

train_data, test_data = final_stocks.randomSplit([0.8, 0.2], seed=42)

Training Data: 80% of the dataset, used to train the linear regression model.

Testing Data: 20% of the dataset, reserved for evaluating the model's performance.

Model Training

A Linear Regression model was trained using the features column as the predictor and the Close column as the target variable:

lr = LinearRegression(featuresCol="features", labelCol="Close")

lr_model = lr.fit(train_data)

Linear regression is a straightforward and interpretable regression method, making it suitable for analyzing relationships between stock prices and their predictors.

A Random Forest Regressor was used for training the model:

rf = RandomForestRegressor(featuresCol="features", labelCol="Close", numTrees=50, maxDepth=10, seed=42)

rf_model = rf.fit(train_data)

numTrees=50: The model builds 50 decision trees to make predictions, ensuring a strong ensemble approach.

maxDepth=10: Limits the depth of each tree to prevent overfitting.

seed=42: Ensures reproducibility of results.

The advantages of random forest are that it handles non-linear relationships effectively, reduces overfitting by averaging predictions across multiple trees, automatically accounts for feature importance during training.

4.4.5 Model Evaluation

The model was evaluated on the test data using the Root Mean Squared Error (RMSE) metric, R^2, MAE.

RMSE: Root Mean Square Error measures the average deviation between predicted and actual closing prices. Lower values indicate better model performance.

evaluator = RegressionEvaluator(labelCol="Close", predictionCol="prediction", metricName="rmse")

rmse = evaluator.evaluate(predictions)

R² (Coefficient of Determination): This indicates how well the model explains the variance in the target variable.

mae_evaluator = RegressionEvaluator(labelCol="Close", predictionCol="prediction", metricName="mae")

mae = mae_evaluator.evaluate(predictions)

MAE: It represents the average absolute difference between predicted and actual values.

train_predictions = lr_model.transform(train_data)

train_r2 = r2_evaluator.evaluate(train_predictions)

test_r2 = r2_evaluator.evaluate(predictions)

4.4.6 Predictions

The trained model was used to predict the closing prices for the test data:

predictions = lr_model.transform(test_data)

predictions.select("prediction", "Close"). show ()

```
R² Score for Train Data: 0.9997512142649044
R² Score for Test Data: 0.9998039957143493
Root Mean Squared Error (RMSE): 1.4077940456962728
Mean Absolute Error (MAE): 0.9190876465741273
+-----------------+------+
|       prediction| Close|
+-----------------+------+
| 317.8487732700164|317.43|
| 321.8461201383309|322.49|
|186.49289324434295|186.02|
| 205.6375499150663|204.99|
|300.83416623086356|299.98|
| 312.7890788135992|312.62|
|   226.30252954011|226.53|
|204.36388748404096| 204.1|
| 213.3737143214504|215.16|
|210.53938980510037|209.99|
|226.56493042943418| 226.8|
|286.50767285931903|285.31|
|  211.715309698274| 211.8|
| 325.4870225428558|324.34|
|204.40836003143886|203.88|
|305.85637176857114|303.43|
|324.73671225644216|323.22|
|210.51404303447995| 210.9|
| 287.8897341763043|288.72|
|302.41275580990794|304.27|
+-----------------+------+
only showing top 20 rows
```

Predicted vs Actual: The predictions for closing prices were compared to the actual values to assess the model's effectiveness visually and numerically.

predictions = rf_model.transform(test_data) predictions.select("prediction", "Close"). show ()

```
R² Score for Train Data: 0.9984576295064094
R² Score for Test Data: 0.9981182439796717
Root Mean Squared Error (RMSE): 4.3620234642756825
Mean Absolute Error (MAE): 2.4587351644624267
+------------------+------+
|        prediction| Close|
+------------------+------+
| 313.0497680522891|317.43|
| 327.0045659653018|322.49|
|188.85020575874188|186.02|
|205.36339867059368|204.99|
| 297.8687238727541|299.98|
| 314.4003915447609|312.62|
|223.10432415227802|226.53|
|204.47124558801804| 204.1|
| 214.3978792744508|215.16|
|211.53137690487802|209.99|
|226.05428562293375| 226.8|
|  288.808064611046|285.31|
|211.48831852143542| 211.8|
| 325.3229620118367|324.34|
|204.80238696509107|203.88|
| 306.5286760746083|303.43|
| 327.0045659653018|323.22|
| 209.5005788077691| 210.9|
|286.19610574605383|288.72|
| 298.1821302916428|304.27|
+------------------+------+
only showing top 20 rows
```

5 Conclusion

This project effectively showcases the integration of big data tools and machine learning techniques for comprehensive stock price analysis and prediction. The use of MapReduce allowed for efficient processing and aggregation of stock data based on monthly and yearly groupings, enabling the calculation of key metrics such as average and maximum values for Close, Volume, Open, High, and Low prices. Hive was utilized to perform insightful analyses, such as identifying the stocks with the highest average daily price range and the highest recorded prices for each year. These capabilities underline the power of big data technologies in handling large-scale datasets and generating meaningful financial insights.

The predictive modeling component, implemented using PySpark, provided actionable results. Linear Regression achieved a remarkably high R^2 score of 0.9998 on test data, showcasing its ability to closely approximate the actual stock prices. Random Forest also demonstrated strong performance with an R^2 score of 0.9981, though it showed a slightly higher error margin compared to Linear Regression. These outcomes highlight the utility of machine learning models in predicting stock prices with significant accuracy, offering valuable tools for investors and financial analysts to make data-driven decisions.

Overall, the project underscores the importance of combining big data analytics and machine learning to analyze and predict stock market trends effectively. By leveraging tools like MapReduce, Hive, Sqoop, and PySpark, we processed and analyzed large datasets efficiently, while machine learning models provided precise predictions. The project demonstrates the potential for further enhancements, such as incorporating additional features, exploring advanced machine learning or deep learning techniques, and implementing real-time data processing. These improvements could make the system even more robust, scalable, and applicable for real-world financial forecasting and decision-making.

6 References

1. Peng, Z., 2019, January. Stocks analysis and prediction using big data analytics. In 2019 international conference on intelligent transportation, big data & smart City (ICITBS) (pp. 309-312). IEEE.
2. Jose, J., Mana, S.C. and Samhitha, B.K., 2019. An efficient system to predict and analyze stock data using Hadoop techniques. International Journal of Recent Technology and Engineering (IJRTE), 8(2), pp.2277-3878.
3. Sirisha, N. and Kiran, K.V.D., 2017, December. Stock exchange analysis using Hadoop user experience (Hue). In 2017 International Conference on Intelligent Sustainable Systems (ICISS) (pp. 1141-1144). IEEE.
4. Darwiesh, A., El-Baz, A.H. and Elhoseny, M., 2024. Intelligent risk management system for enhancing performance of stock market applications. Expert Systems with Applications, 249, p.123493.