INFO-I 535

Management, Access, and Use of Big and Complex Data

Course Project: Big Data Concepts

on

Microsoft Stock Time Series Analysis

Developed By

Sai Teja Burla

Student ID: 2001058983

MS in Data Science

Instructed By

Dr. Inna Kouper

Indiana University Bloomington

Developed at

Indiana University Bloomington

INTRODUCTION

Microsoft is a multinational technology company that has been one of the most successful companies in the world for several decades. Its stock is traded on the Nasdaq exchange under the symbol MSFT. Their stocks are one of the most widely held and actively traded stocks in the world.

We make use of time series analysis to predict future stock prices of Microsoft based on their historical stock prices. Using this method can provide a person with some valuable insights into various things about the company like its performance, market trends, and potential investment opportunities.

Time series analysis is basically a statistical technique that involves studying the patterns and trends of data over time to identify potential relationships and predict future values which in this case would be stocks related data. Time series data is a sequence of data points collected at regular intervals over time.

This project is developed by keeping the idea of big data and its tools and technologies in mind. Implementation of various big data technologies that were taught as a part of this course was done, they are: **J2 Virtual Machine** (virtualization tool), **MongoDB** (NoSQL database), **PySpark** (Distributed Data Processing) and **GitHub** (Publishing Code and Results).

BACKGROUND

The overall idea of the project is stock prediction using time series analysis. Stock prediction time series analysis is an interesting and important topic because of its potential to help investors and traders make informed decisions about buying, selling, and holding stocks. It involves using statistical models and machine learning algorithms to analyze historical data and predict the future stock prices using the same. The analysis of stock market data can also help identify opportunities for investment in emerging markets or sectors, helping investors to diversify their portfolios and mitigate risks.

This field is continuously evolving as new data sources and modeling techniques are developed, and also as the stock market becomes more complex and dynamic. There is a continuous need to improve the accuracy and reliability of stock prediction systems to help investors make better decisions in an increasingly competitive market.

All of the above reasons pushed me to implement a stock prediction time series analysis for Microsoft stock data in this case. I used jetstream2 as a virtualization tool where I implemented the entire project as the dataset and implementation for the same would be too heavy for my local system. MongoDB was installed inside the j2 VM to use that for storing and pulling the data into the Jupyter Notebook for time series analysis. PySpark and Pandas are used to code the actual time series analysis and to do visualizations to gain insights about the data and finally storing all the files with obtained results into GitHub to share it with everyone.

METHODOLOGY

To meet the requirements of the course project, I made use of multiple tools that correspond to different modules of this course as a part of this project. These tools played an essential role that helped in making my project goals achievable. The steps taken to solve this project along with the tools i needed to implement it are mentioned briefly as follows:

1. I first set up a **Jetstream2** Virtual Machine (VM) for doing my entire project
2. Then I had to set up **Jupyter Notebook** and **MongoDB** to work on the VM
3. Then I collected the **Microsoft Stock dataset** from the source
4. Ingested the data into MongoDB and stored it to work on it from a Jupyter Notebook
5. Connected and pulled the data from MongoDB to the Jupyter Notebook
6. Did preprocessing using **Pandas** library and visualized the data using **Matplotlib**, **Seaborn** and **Plotly** libraries
7. I also made models in two ways:
    a. Using the combination of Pandas and **Statsmodels** libraries
    b. Using the combination of **PySpark** and **Prophet** libraries
8. Finally, I uploaded the dataset, code and results on **GitHub** and posted a brief ReadMe about the project

The following few sections explain a few of the above-mentioned tools in detail with respect to my project and their technological set-up is also explained.

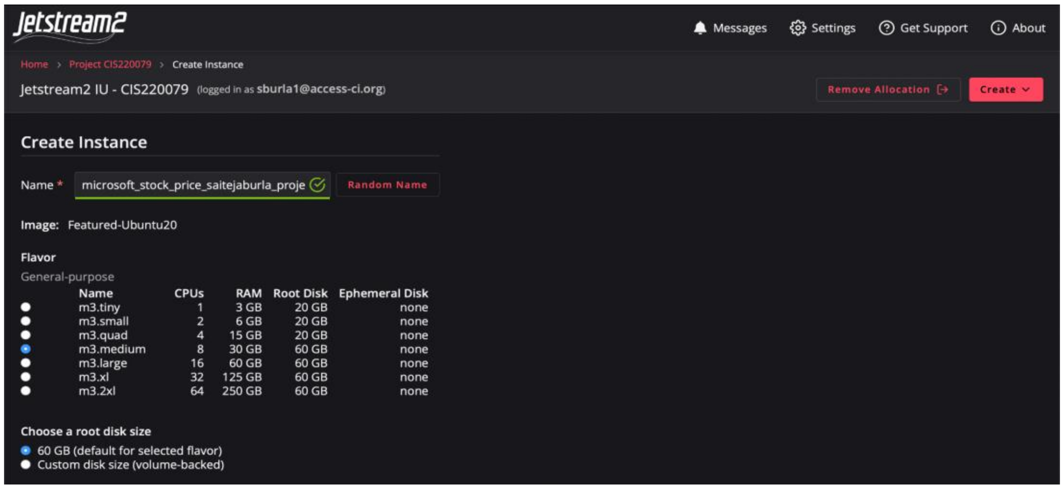
Jetstream2

High-performance computing (HPC) platform Jetstream2 is a cloud-based system made for research and scientific computation. It gives users access to a flexible computing environment with virtual machines (VMs) that may be set up for a variety of use cases, including the processing of big data.
A few advantages of using Jetstream2 for big data projects are as follows:

- Scalability - To accommodate a big data project's needs, the platform may be scaled up or down giving the user the flexibility to add or subtract computing resources as necessary. This makes it simple to handle complicated data processing jobs and enormous volumes of data.
- Customizability - Users have a large selection of pre-configured VMs to pick from or they can build their own unique VMs with certain software and customizations.
- Cost effective - Because the users only pay for the resources they use, Jetstream2 is an affordable option for big data projects.
- Collaboration - Users can share data and work together in real-time on data processing tasks which offers a collaborative environment for large data projects even if the team members working on the project are spread out across the country.
- Accessibility - Users may work on large data projects from any location with an internet connection.

The following are the set-up steps performed for Jetstream2 for this project:
After logging into my Jetstream2 account I had to simply make a VM instance under this particular course. I used the following specifications for my instance:

- Ubuntu 20.04
- m3.medium
- 8 CPUs
- 30GB RAM
- 60GB Root Disk
- Web Desktop Enabled



Home > Project CIS220079 > Create Instance

Jetstream2 IU - CIS220079 (logged in as sburla1@access-ci.org)

Create Instance

Name * microsoft_stock_price_saitejaburla_proje

Image: Featured-Ubuntu20

Flavor

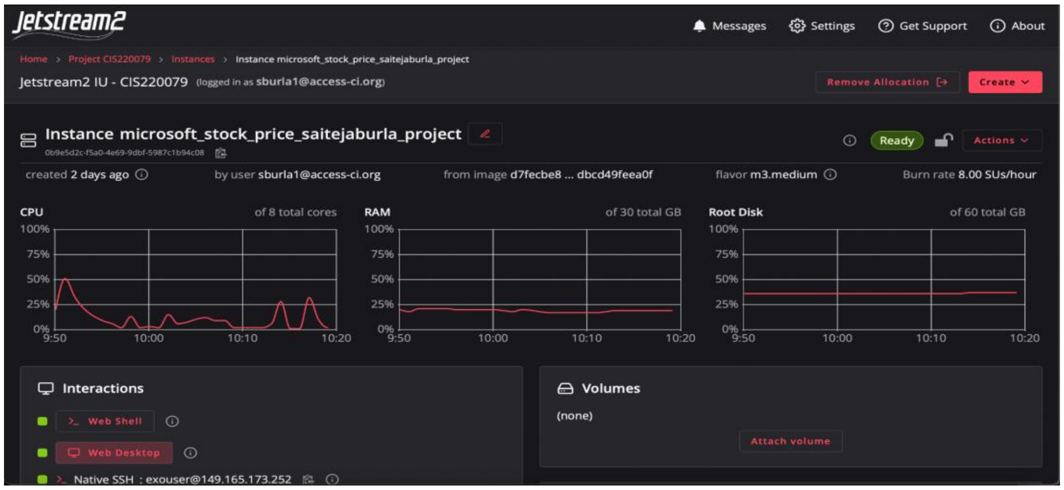
General-purpose

	Name	CPUs	RAM	Root Disk	Ephemeral Disk
○	m3.tiny	1	3 GB	20 GB	none
○	m3.small	2	6 GB	20 GB	none
○	m3.quad	4	15 GB	20 GB	none
●	m3.medium	8	30 GB	60 GB	none
○	m3.large	16	60 GB	60 GB	none
○	m3.xl	32	125 GB	60 GB	none
○	m3.2xl	64	250 GB	60 GB	none

Choose a root disk size

[✓ 60 GB (default for selected flavor)]
Custom disk size (volume-backed)

After the instance was made this is how the instance dashboard page looks like (this screenshot was taken after I was done with the entire project):



Home > Project CIS220079 > Instances > Instance microsoft_stock_price_saitejaburla_project

Jetstream2 IU - CIS220079 (logged in as sburla1@access-ci.org)

Instance microsoft_stock_price_saitejaburla_project

0b9e5d2c-f5a0-4e69-9dbf-5987c1b94c08

Ready

created 2 days ago   by user sburla1@access-ci.org   from image d7fecbe8 ... dbcd49feea0f   flavor m3.medium   Burn rate 8.00 SUs/hour

CPU of 8 total cores   RAM of 30 total GB   Root Disk of 60 total GB

Interactions

Native SSH : exouser@149.165.173.252

Volumes

(none)

Over the course of the project, I made use of shelve action and unshelve actions to make sure my instance is not running and consuming resources when I was not using it. At the end of the project, I deleted this instance.

MongoDB

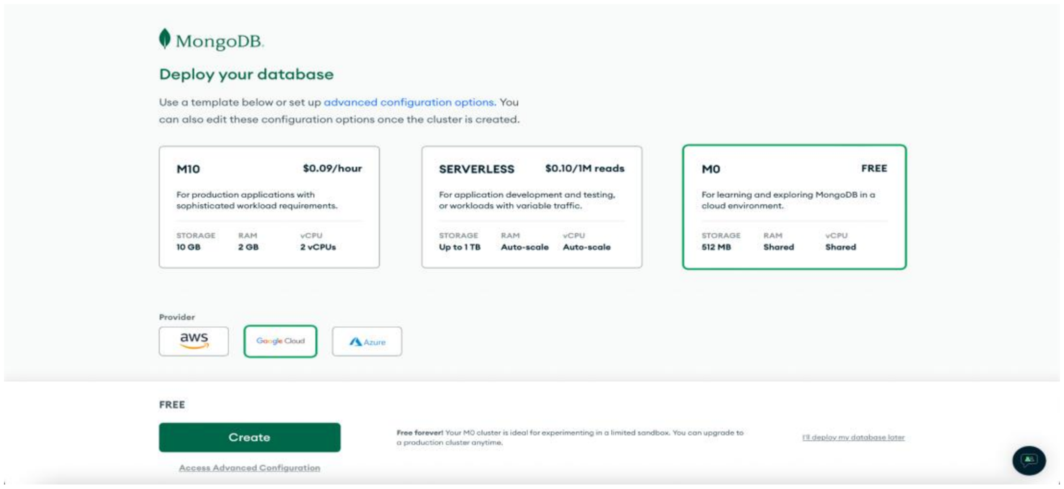
Large volumes of unstructured and semi-structured data can be stored and managed using MongoDB, a well-known open-source NoSQL document-oriented database. MongoDB is often chosen because of its scalability, performance, and flexibility.

A few advantages of using MongoDB for big data projects are as follows:

- Document Oriented - Because MongoDB is a document-oriented database, it stores data in documents that can be customized and resemble JSON rather than in tables with set columns and rows. This makes it simple to scale up or down as necessary and to store and retrieve complicated, hierarchical data structures.
- Scalability - Because MongoDB is made to scale horizontally across multiple servers, it can easily handle workloads with high traffic and large amounts of data. Sharding divides data among numerous servers and permits parallel processing to help achieve this.
- High performance - MongoDB supports in-memory caching, automatic indexing, and native aggregation and is built for high-performance data access and query processing.
- Flexibility - With its support for ad-hoc querying, dynamic schema building, and real-time data processing, MongoDB is extremely flexible. This makes it simple to accommodate changing requirements and evolving data structures over time.

The following are the set-up steps performed for MongoDB for this project:

Create an Atlas account and make a free cluster.


MongoDB.

Deploy your database

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

M10 $0.09/hour	SERVERLESS $0.10/1M reads	M0 FREE
For production applications with sophisticated workload requirements.	For application development and testing, or workloads with variable traffic.	For learning and exploring MongoDB in a cloud environment.
STORAGE 10 GB, RAM 2 GB, vCPU 2 vCPUs	STORAGE Up to 1 TB, RAM Auto-scale, vCPU Auto-scale	STORAGE 512 MB, RAM Shared, vCPU Shared

Provider: aws, Google Cloud, Azure

FREE

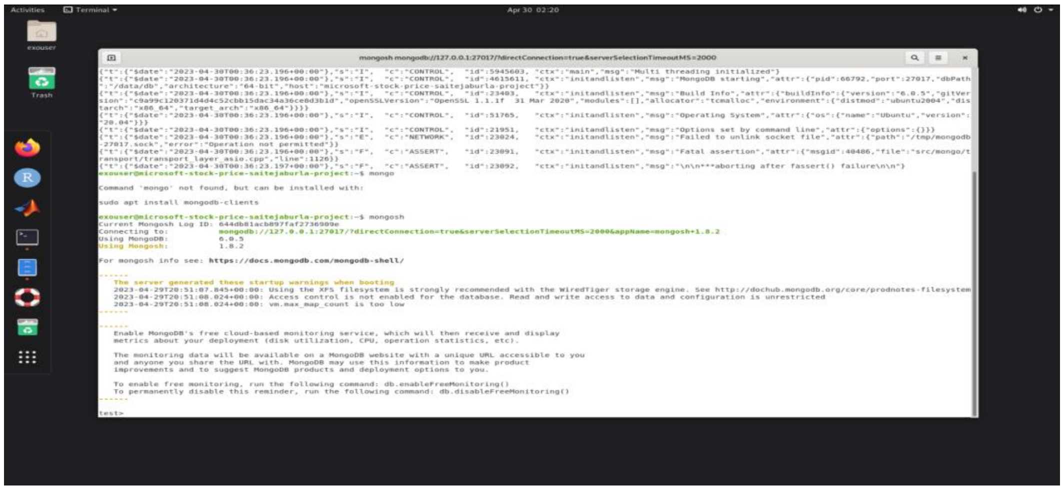
Create

Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

I'll deploy my database later

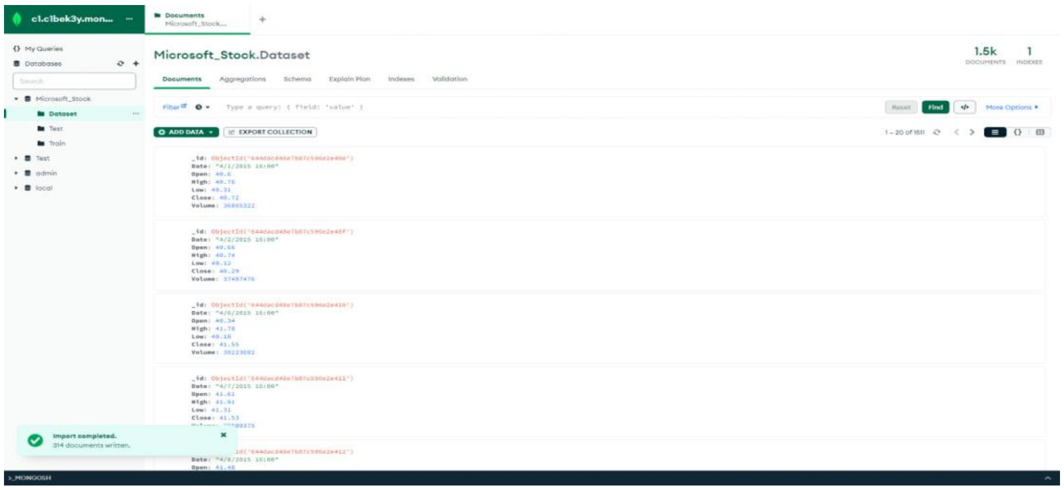
Access Advanced Configuration

Install community edition server using the command line on Jetstream2 VM.



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
{"t":{"$date":"2023-04-30T00:36:23.196+00:00"},"s":"I",  "c":"CONTROL",  "id":5945603, "ctx":"main","msg":"Multi threading initialized"}
{"t":{"$date":"2023-04-30T00:36:23.196+00:00"},"s":"I",  "c":"CONTROL",  "id":4615611, "ctx":"initandlisten","msg":"MongoDB starting","attr":{"pid":66792,"port":27017,"dbPath":"/data/db","architecture":"64-bit","host":"microsoft-stock-price-saitejaburla-project"}}
{"t":{"$date":"2023-04-30T00:36:23.196+00:00"},"s":"I",  "c":"CONTROL",  "id":23403,   "ctx":"initandlisten","msg":"Build Info","attr":{"buildInfo":{"version":"6.0.5","gitVersion":"c9a99c120371d4d4c52cbb15dac34a36ce8d3b1d","openSSLVersion":"OpenSSL 1.1.1f  31 Mar 2020","modules":[],"allocator":"tcmalloc","environment":{"distmod":"ubuntu2004","distarch":"x86_64","target_arch":"x86_64"}}}}
{"t":{"$date":"2023-04-30T00:36:23.196+00:00"},"s":"I",  "c":"CONTROL",  "id":51765,   "ctx":"initandlisten","msg":"Operating System","attr":{"os":{"name":"Ubuntu","version":"20.04"}}}
{"t":{"$date":"2023-04-30T00:36:23.196+00:00"},"s":"I",  "c":"CONTROL",  "id":21951,   "ctx":"initandlisten","msg":"Options set by command line","attr":{"options":{}}}
{"t":{"$date":"2023-04-30T00:36:23.196+00:00"},"s":"E",  "c":"NETWORK",  "id":23024,   "ctx":"initandlisten","msg":"Failed to unlink socket file","attr":{"path":"/tmp/mongodb-27017.sock","error":"Operation not permitted"}}
{"t":{"$date":"2023-04-30T00:36:23.196+00:00"},"s":"F",  "c":"ASSERT",   "id":23091,   "ctx":"initandlisten","msg":"Fatal assertion","attr":{"msgid":40486,"file":"src/mongo/transport/transport_layer_asio.cpp","line":1126}}
{"t":{"$date":"2023-04-30T00:36:23.197+00:00"},"s":"F",  "c":"ASSERT",   "id":23092,   "ctx":"initandlisten","msg":"\n\n***aborting after fassert() failure\n\n"}
exouser@microsoft-stock-price-saitejaburla-project:~$ mongo

Command 'mongo' not found, but can be installed with:

sudo apt install mongodb-clients

exouser@microsoft-stock-price-saitejaburla-project:~$ mongosh
Current Mongosh Log ID: 644db81acb897faf2736909e
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.8.2
Using MongoDB:          6.0.5
Using Mongosh:          1.8.2

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2023-04-29T20:51:07.845+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
   2023-04-29T20:51:08.024+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2023-04-29T20:51:08.024+00:00: vm.max_map_count is too low
------

------
   Enable MongoDB's free cloud-based monitoring service, which will then receive and display
   metrics about your deployment (disk utilization, CPU, operation statistics, etc).

   The monitoring data will be available on a MongoDB website with a unique URL accessible to you
   and anyone you share the URL with. MongoDB may use this information to make product
   improvements and to suggest MongoDB products and deployment options to you.

   To enable free monitoring, run the following command: db.enableFreeMonitoring()
   To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
------

test>
```

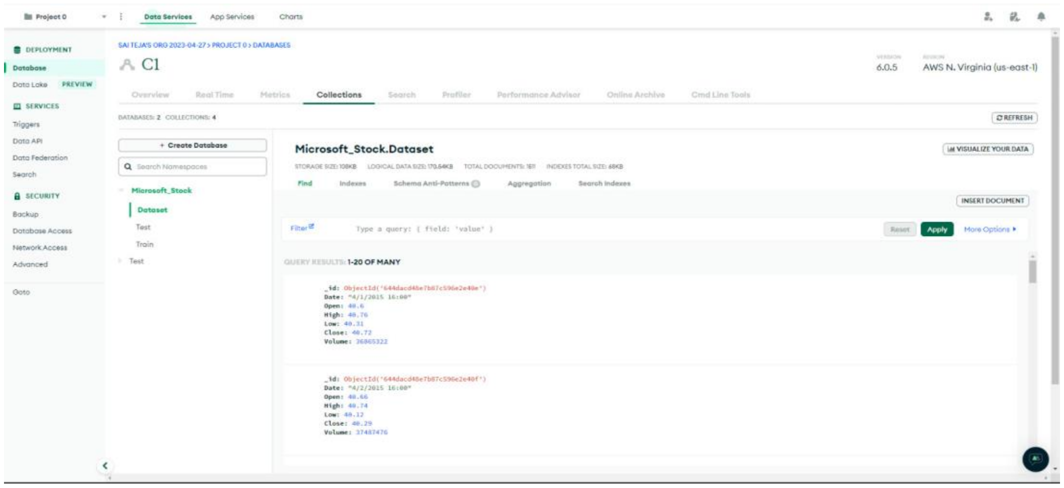
Download MongoDB Compass and connect Atlas cluster and establish a connection with it. After this we ingest the dataset into the created database via Compass.



Microsoft_Stock.Dataset

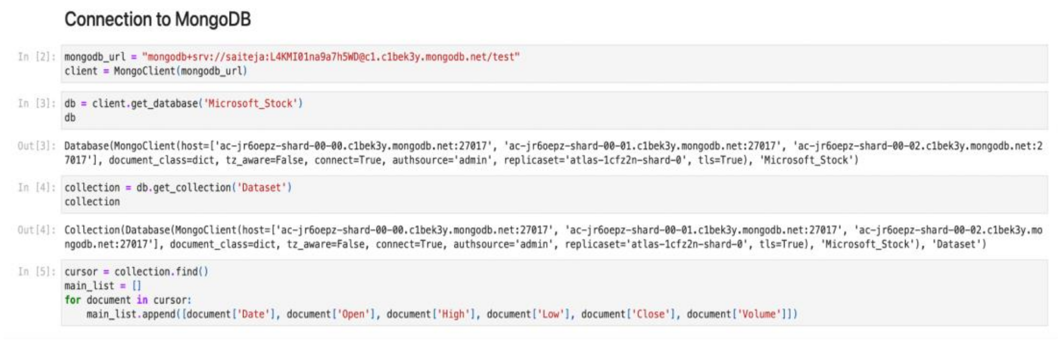
```
_id: ObjectId('644dacd48e7b87c596e2e40e')
Date: "4/1/2015 16:00"
Open: 40.6
High: 40.76
Low: 40.31
Close: 40.72
Volume: 36865322

_id: ObjectId('644dacd48e7b87c596e2e40f')
Date: "4/2/2015 16:00"
Open: 40.66
High: 40.74
Low: 40.12
Close: 40.29
Volume: 37487476

_id: ObjectId('644dacd48e7b87c596e2e410')
Date: "4/6/2015 16:00"
Open: 40.34
High: 41.78
Low: 40.18
Close: 41.55
Volume: 39223692

_id: ObjectId('644dacd48e7b87c596e2e411')
Date: "4/7/2015 16:00"
Open: 41.61
High: 41.91
Low: 41.31
Close: 41.53
Volume: 28809375

_id: ObjectId('644dacd48e7b87c596e2e412')
Date: "4/8/2015 16:00"
Open: 41.48
```

We check if the connection is properly established by checking on Atlas.



```
_id: ObjectId('644dacd48e7b87c596e2e40e')
Date: "4/1/2015 16:00"
Open: 40.6
High: 40.76
Low: 40.31
Close: 40.72
Volume: 36865322

_id: ObjectId('644dacd48e7b87c596e2e40f')
Date: "4/2/2015 16:00"
Open: 40.66
High: 40.74
Low: 40.12
Close: 40.29
Volume: 37487476
```

We then install the pymongo library to connect MongoDB with Jupyter Notebook.



Connection to MongoDB

```
In [2]: mongodb_url = "mongodb+srv://saiteja:L4KMI01na9a7h5WD@c1.c1bek3y.mongodb.net/test"
        client = MongoClient(mongodb_url)

In [3]: db = client.get_database('Microsoft_Stock')
        db

Out[3]: Database(MongoClient(host=['ac-jr6oepz-shard-00-00.c1bek3y.mongodb.net:27017', 'ac-jr6oepz-shard-00-01.c1bek3y.mongodb.net:27017', 'ac-jr6oepz-shard-00-02.c1bek3y.mongodb.net:2
        7017'], document_class=dict, tz_aware=False, connect=True, authsource='admin', replicaset='atlas-1cfz2n-shard-0', tls=True), 'Microsoft_Stock')

In [4]: collection = db.get_collection('Dataset')
        collection

Out[4]: Collection(Database(MongoClient(host=['ac-jr6oepz-shard-00-00.c1bek3y.mongodb.net:27017', 'ac-jr6oepz-shard-00-01.c1bek3y.mongodb.net:27017', 'ac-jr6oepz-shard-00-02.c1bek3y.mo
        ngodb.net:27017'], document_class=dict, tz_aware=False, connect=True, authsource='admin', replicaset='atlas-1cfz2n-shard-0', tls=True), 'Microsoft_Stock'), 'Dataset')

In [5]: cursor = collection.find()
        main_list = []
        for document in cursor:
            main_list.append([document['Date'], document['Open'], document['High'], document['Low'], document['Close'], document['Volume']])
```

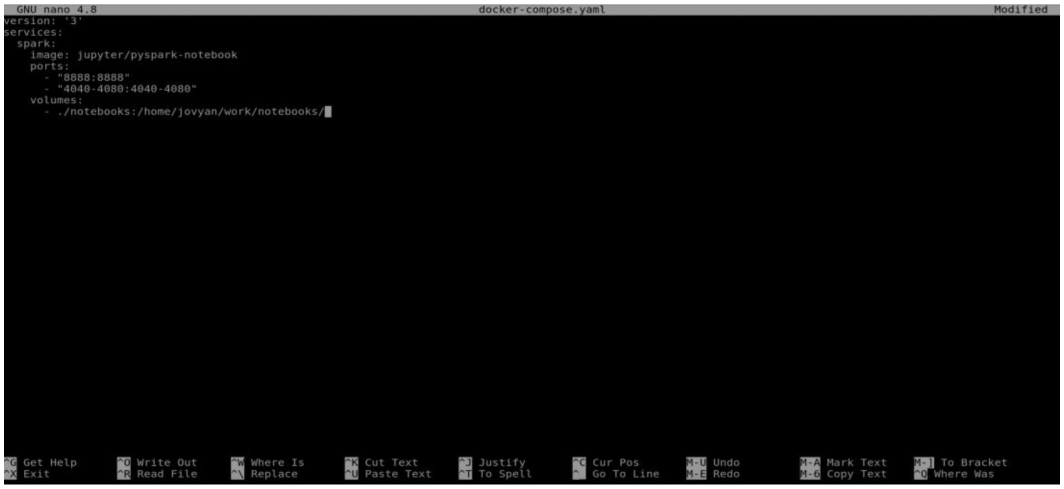
We can now make use of the dataset for further analysis.

Jupyter Notebook

We needed to set up a jupyter lab in the Jetstream2 VM for the purpose of executing the entire time series analysis. We make use of Jupyter Notebook because the user interface and the features it provides are all user friendly and good for executing a mini big data project.

The following are the set-up steps performed for Jupyter Notebook for this project:

After setting up the Jetstream2 VM we open the web shell and write the following into the docker-compose.yaml file.



```
  GNU nano 4.8                    docker-compose.yaml                    Modified
version: '3'
services:
  spark:
    image: jupyter/pyspark-notebook
    ports:
      - "8888:8888"
      - "4040-4080:4040-4080"
    volumes:
      - ./notebooks:/home/jovyan/work/notebooks/
```

After this we execute it and we get an IP address which we can open in the browser inside the VM and we get the Jupyter Notebook.

Microsoft Stock Dataset

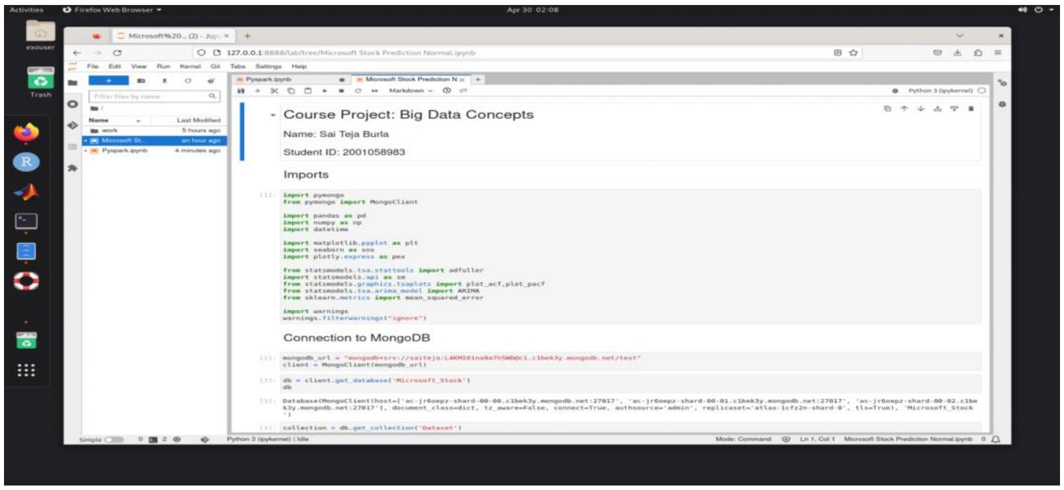
The dataset used as a part of this project is the historical data for the stocks of the company Microsoft Inc. This dataset was collected from google sheets using the command ‘GOOGLEFINANCE’ and was published by Kaggle. It consists of daily stock data from the year 2015 to 2021 and contains a total of 1511 unique rows. The dataset has a total of 6 columns, they are listed as follows:

- Date - The corresponding date of a certain stock price in the format ‘mm/dd/yyyy 16:00:00’
- Open - Opening price of the stock in decimals
- Close - Closing price of the stock in decimals
- High - Highest price of the stock in the day in decimals
- Low - Lowest price of the stock in the day in decimals
- Volume - Number of shares traded in the day in numbers

Time Series Analysis

As mentioned earlier I had implemented two notebooks of time series analysis on this dataset one using Pandas and Statistic Models and another using PySpark and Prophet. Let us see the steps in both of these notebooks.
The following steps are done as a part of Pandas and Statistic Models notebook:

1. Imported all the required libraries
2. Connected to MongoDB using the pymongo library and get the data and store it in a dataframe
3. Preprocessed the ‘Date’ column from object to datetime datatype with the format ‘YYYY-MM-DD’ for further use
4. Checked if there were any missing values in the dataset which there weren't any in this case
5. Checked for any outliers in the data using a boxplot
6. Plotted a few visualizations to gain insights into the data this includes a correlation heatmap as well
7. Preparing the data for time series analysis by only keeping ‘Date’ and ‘Close’ columns and making the ‘Date’ column as the index
8. Performed Augmented Dickey Fuller test
9. Plotted the Decomposition plots to check if the data had any trend and seasonality in it
10. Plotted ACF and PACF plots for finding the hyperparameter values for our time series models
11. Split the data into train and test and plotted this in the same graph
12. Implemented ARIMA and SARIMAX models for predicting Close price values on the test data
13. Plotted graphs and calculated MSE value for checking whether the predicted values are accurate or not and to what rate are they accurate



Course Project: Big Data Concepts

Name: Sai Teja Burla

Student ID: 2001058983

Imports

```
[1]: import pymongo
     from pymongo import MongoClient

     import pandas as pd
     import numpy as np
     import datetime

     import matplotlib.pyplot as plt
     import seaborn as sns
     import plotly.express as pex

     from statsmodels.tsa.stattools import adfuller
     import statsmodels.api as sm
     from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
     from statsmodels.tsa.arima_model import ARIMA
     from sklearn.metrics import mean_squared_error

     import warnings
     warnings.filterwarnings("ignore")
```

Connection to MongoDB

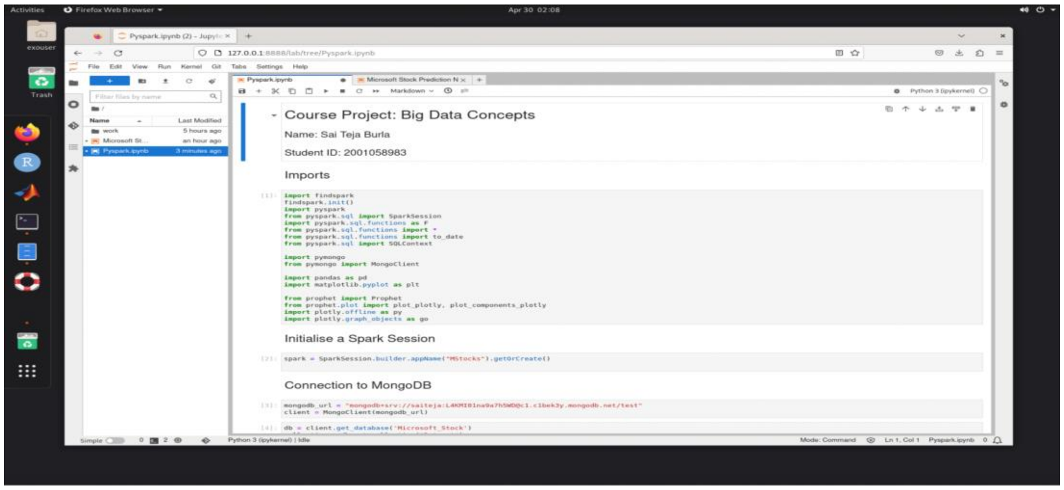
```
[2]: mongodb_url = "mongodb+srv://saiteja:L4KMI81na9a7h5WD@c1.c1bek3y.mongodb.net/test"
     client = MongoClient(mongodb_url)

[3]: db = client.get_database('Microsoft_Stock')
     db

[3]: Database(MongoClient(host=['ac-jr6oepz-shard-00-00.c1bek3y.mongodb.net:27017', 'ac-jr6oepz-shard-00-01.c1bek3y.mongodb.net:27017', 'ac-jr6oepz-shard-00-02.c1be
     k3y.mongodb.net:27017'], document_class=dict, tz_aware=False, connect=True, authsource='admin', replicaset='atlas-1cfz2n-shard-0', tls=True), 'Microsoft_Stock
     ')

[4]: collection = db.get_collection('Dataset')
```

Apr 30 02:08
127.0.0.1:8888/lab/tree/Microsoft Stock Prediction Normal.ipynb
Pyspark.ipynb
Microsoft Stock Prediction N
Python 3 (ipykernel) | Idle
Mode: Command
Ln 1, Col 1

I implemented a different model using PySpark and Prophet library to compare the two and see if using PySpark would have a significant difference in processing or not.
The following steps are done as a part of PySpark and Prophet notebook:

1. Imported all the required libraries

2. Initialize a spark session
3. Connected to MongoDB using the pymongo library and get the data and store it in a PySpark dataframe
4. Preprocessed the ‘Date’ column from object to datetime datatype with the format ‘YYYY-MM-DD’ for further use
5. Checked if there were any missing values in the dataset which there weren't any in this case
6. Split the data into train and test and plotted this in the same graph
7. Implemented Prophet model on the train data and forecasted the Closing price for the test data
8. Plotted a graph to see if the results given by the model make sense


Course Project: Big Data Concepts

Name: Sai Teja Burla

Student ID: 2001058983

Imports

```
[1]: import findspark
     findspark.init()
     import pyspark
     from pyspark.sql import SparkSession
     import pyspark.sql.functions as F
     from pyspark.sql.functions import *
     from pyspark.sql.functions import to_date
     from pyspark.sql import SQLContext

     import pymongo
     from pymongo import MongoClient

     import pandas as pd
     import matplotlib.pyplot as plt

     from prophet import Prophet
     from prophet.plot import plot_plotly, plot_components_plotly
     import plotly.offline as py
     import plotly.graph_objects as go
```

Initialise a Spark Session

```
[2]: spark = SparkSession.builder.appName("MStocks").getOrCreate()
```

Connection to MongoDB

```
[3]: mongodb_url = "mongodb+srv://saiteja:L4KMI81na9a7h5WD@c1.c1bek3y.mongodb.net/test"
     client = MongoClient(mongodb_url)

[4]: db = client.get_database('Microsoft_Stock')
```

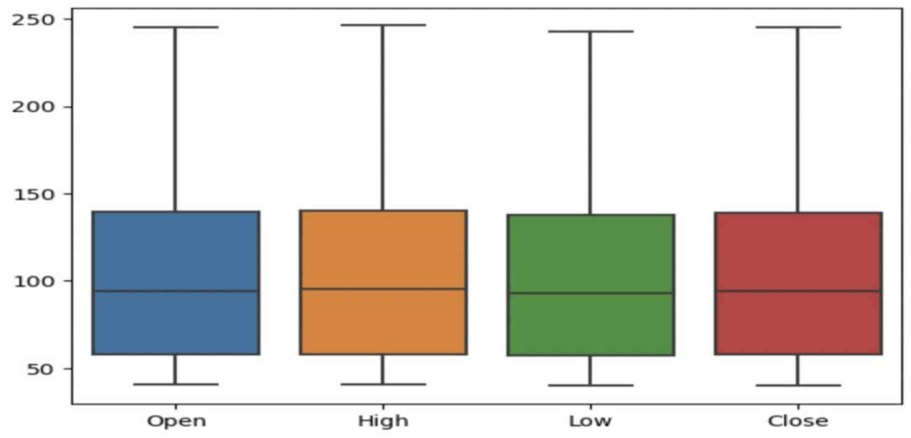
GitHub

GitHub is a popular web-based platform that is used for software development, collaboration, and version control. It provides a centralized location for storing, managing, and sharing code and other project files, as well as a wide range of tools and features for collaboration, issue tracking, and continuous integration.
I made use of GitHub to store a copy of the dataset as a backup as well as store all the code files with results. I also made a readme.md file with brief details on the implemented project and the dataset.
Link to the Repository: https://github.com/BurlaSaiTeja/Microsoft_Stock_Time_Series_Analysis

RESULTS

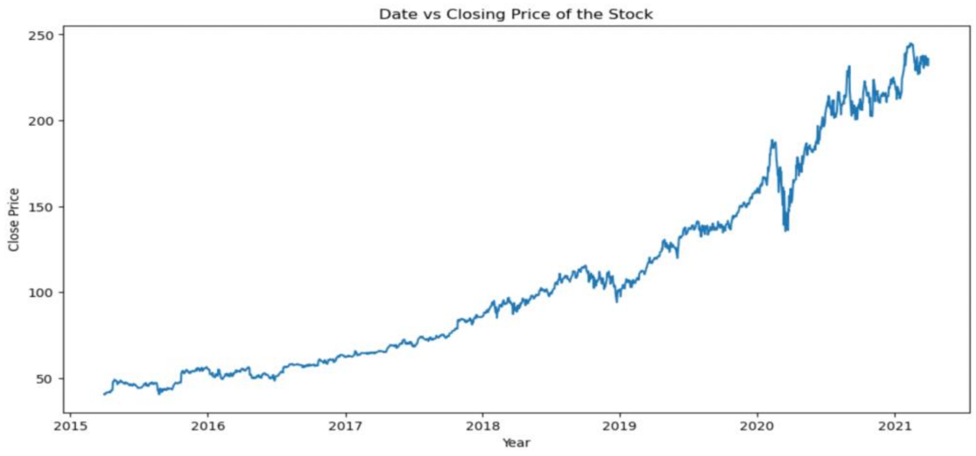
As a part of the results, I would like to show the visualizations for each part of the code and discuss the insights or what each of the visualizations represents.
Outlier Analysis - Outliers is a data point that is significantly different from other data points in a dataset. I had plotted the following boxplot on all the columns that showed me that there are no outliers in the dataset and therefore outlier removal was not required.

```
Out[13]:  <Axes: >
```

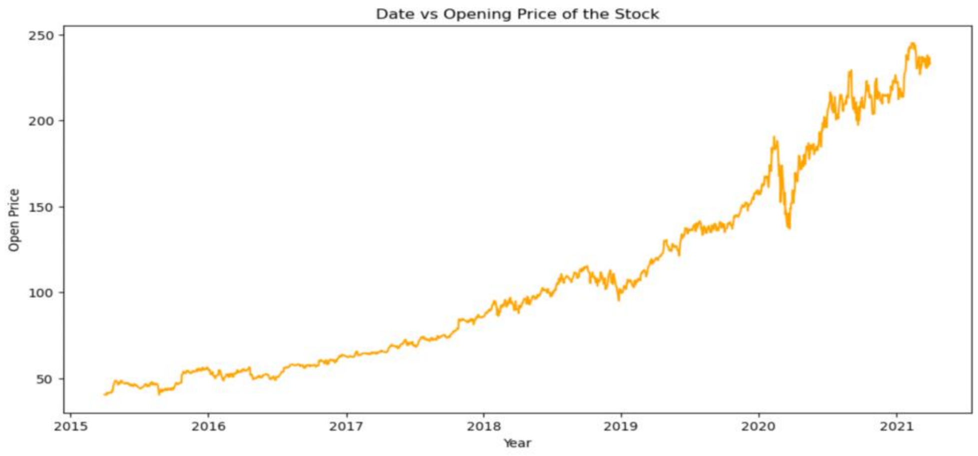


Visualizing every column in the dataset with the 'Date' column - This visualization shows us if there is any possible trend in the data for each of the columns. Except the volume graph we can see that all the other graphs show a growing trend.

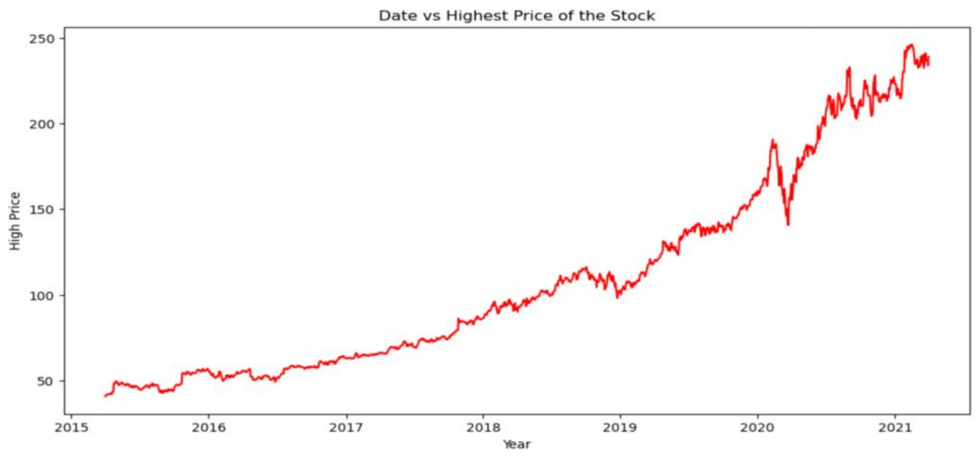
```
Out[14]:  [<matplotlib.lines.Line2D at 0x7f4eac43f700>]
```



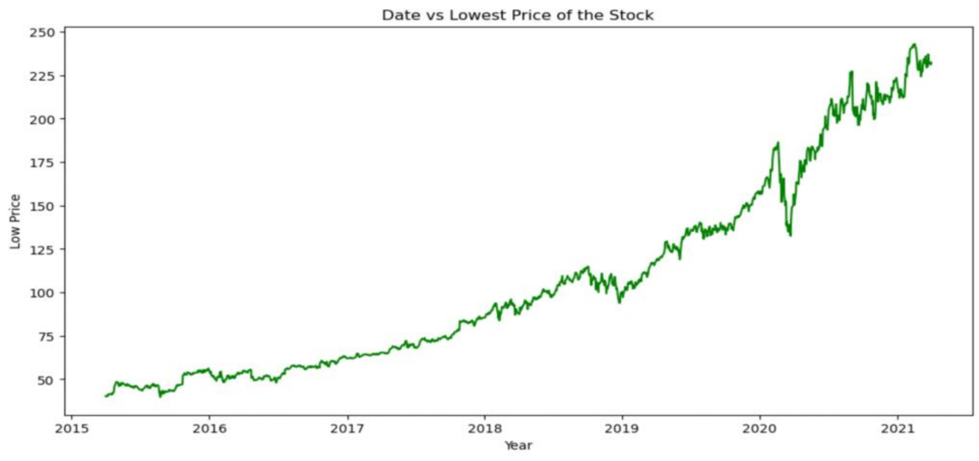
```
Out[15]:  [<matplotlib.lines.Line2D at 0x7f4eac2a66e0>]
```



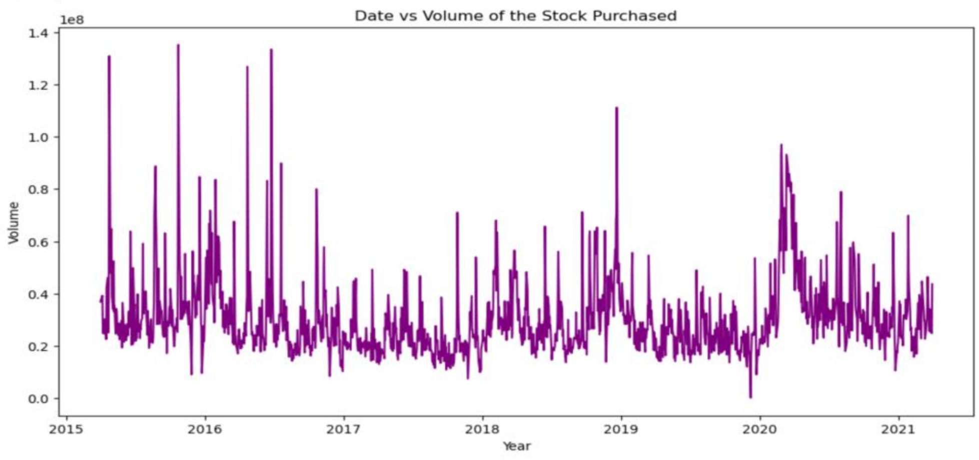
```
Out[16]:  [<matplotlib.lines.Line2D at 0x7f4eac077a00>]
```



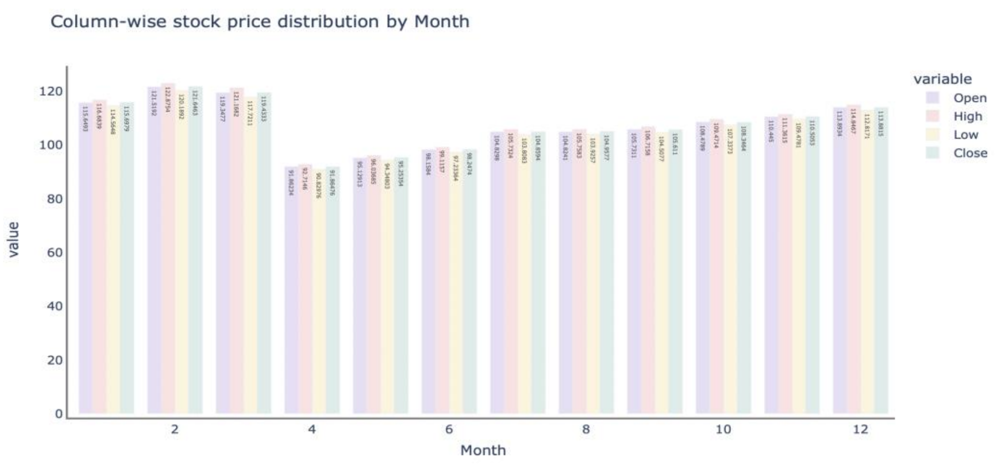
```
Out[17]:  [<matplotlib.lines.Line2D at 0x7f4eac0f9480>]
```



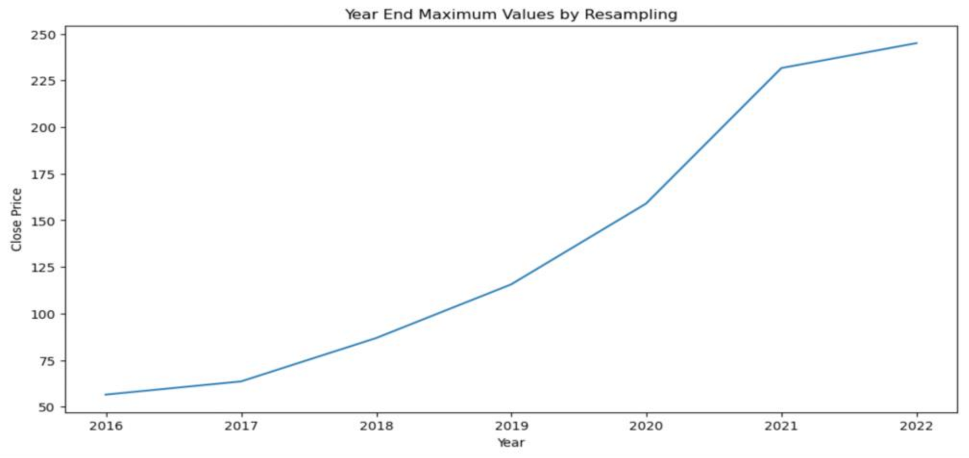
```
Out[18]:  [<matplotlib.lines.Line2D at 0x7f4e9e752650>]
```

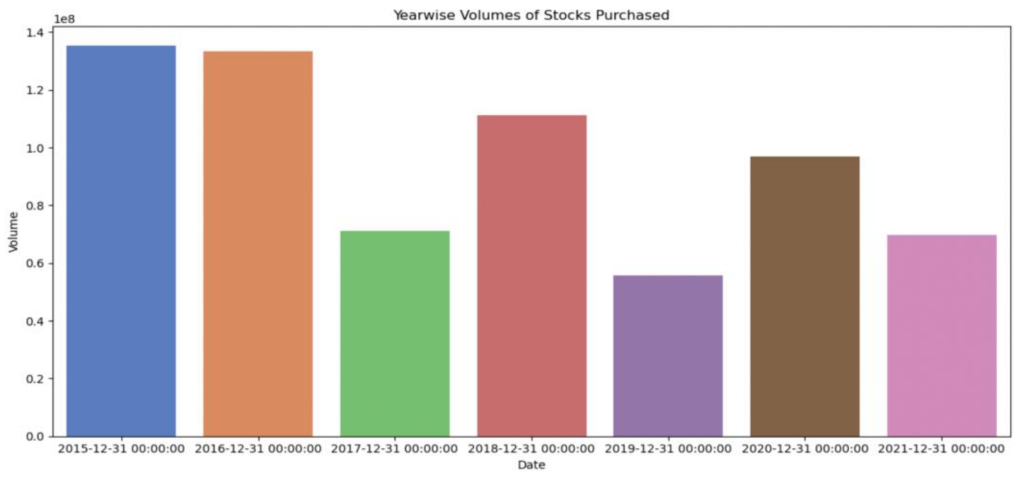


Other Visualizations - I had plotted a few additional graphs just to see and understand and play around with the data.

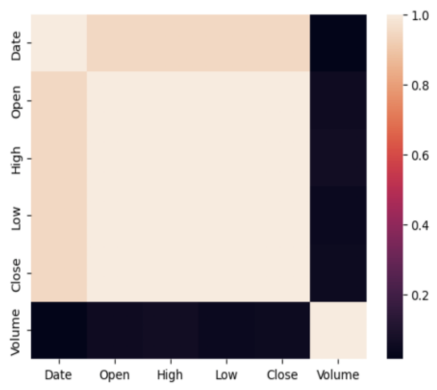

Column-wise stock price distribution by Month
value
Month
variable
Open
High
Low
Close

```
Out[22]: [<matplotlib.lines.Line2D at 0x7f4e9cd3d990>]
```


Year End Maximum Values by Resampling
Close Price
Year

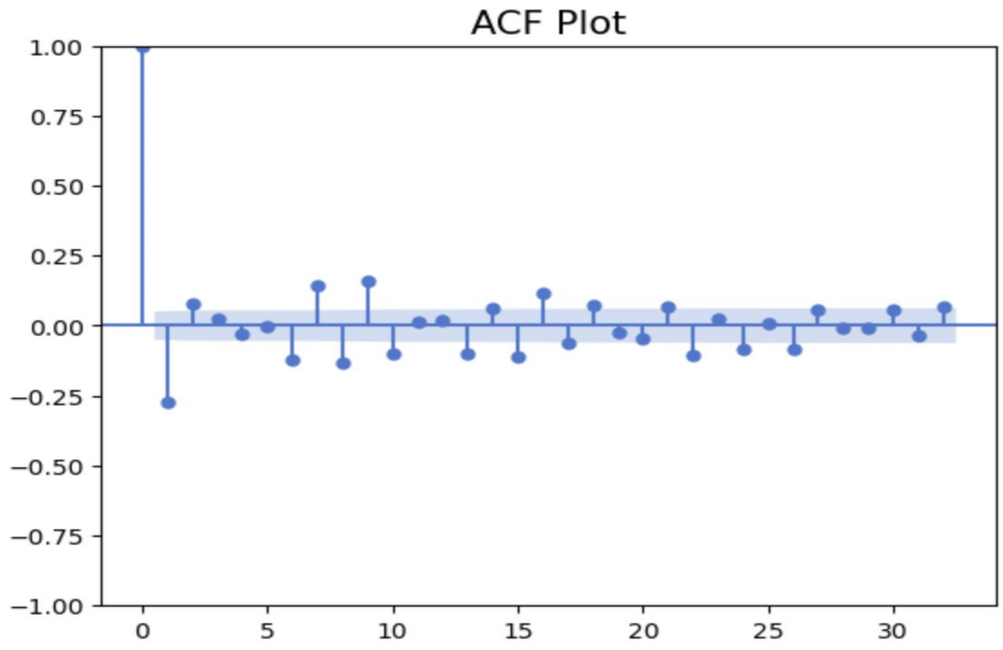

Yearwise Volumes of Stocks Purchased
Volume
Date
1e8

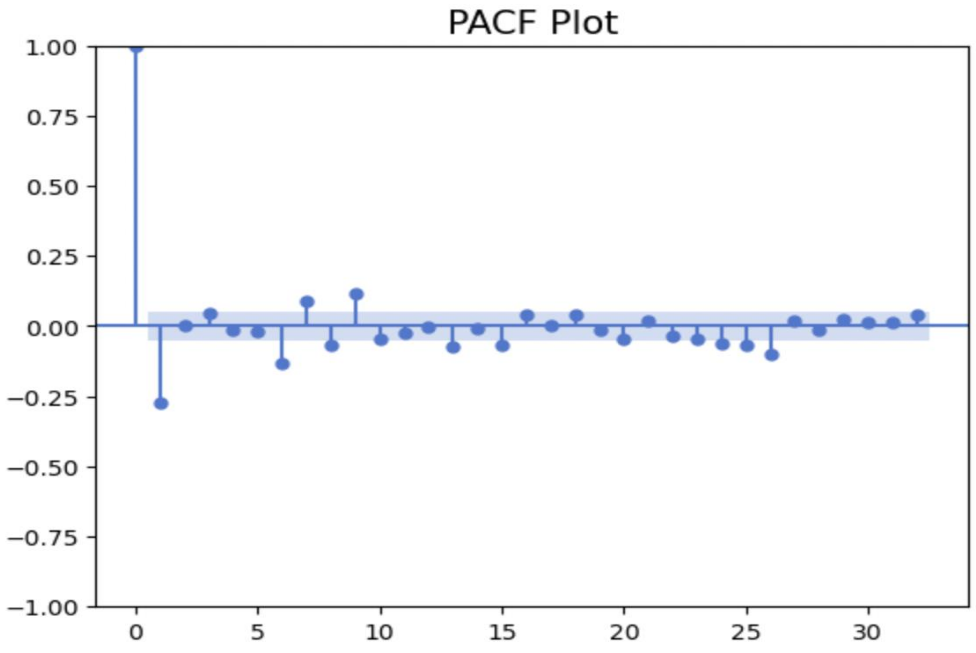
```
Out[24]: <Axes: >
```


Date
Open
High
Low
Close
Volume

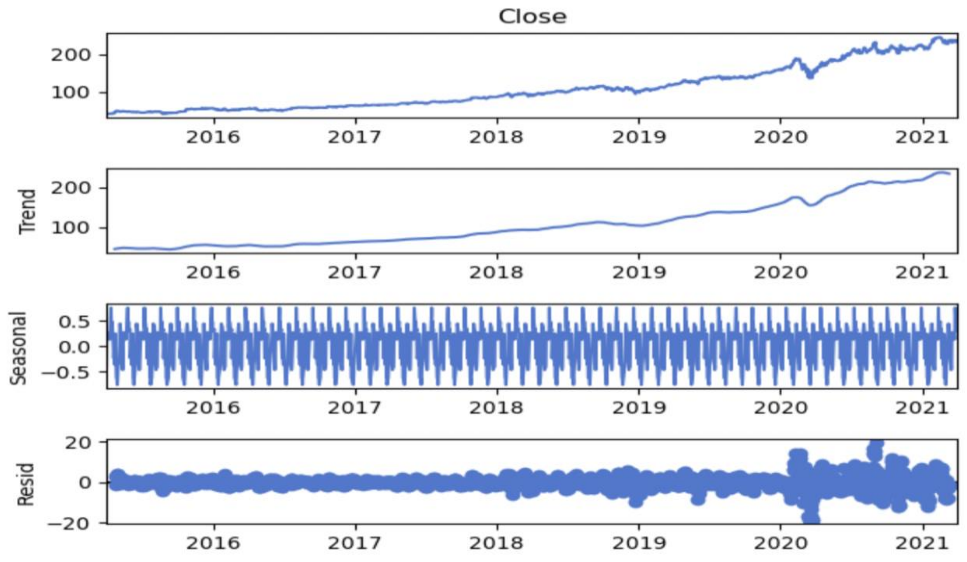
Augmented Dickey Fuller Test and ACF-PACF plots - It is a hypothesis test to check if the data is stationary or not. We check this by the value of p that comes out of this test that is if p value is greater than 0.05 the data is not stationary else it is stationary. This test is done to find if this data can be used to plot ACF and PACF plots and get hyperparameter values for statistical models which perform the time series analysis. If the data is not stationary then we differentiate and then plot these graphs to get the parameters. In this data case the data itself was not stationary so I performed differentiation and plotted ACF and PACF plots to get the hyperparameter values. The following are ADF test, ACF and PACF plot results for the differentiated data.

```
ADF Test Statistic : -10.043852364231784
p-value : 1.472030729039202e-17
#lags used : 24
Number of observations used : 1486
Strong evidence against the null hypothesis, reject the null hypothesis.
```

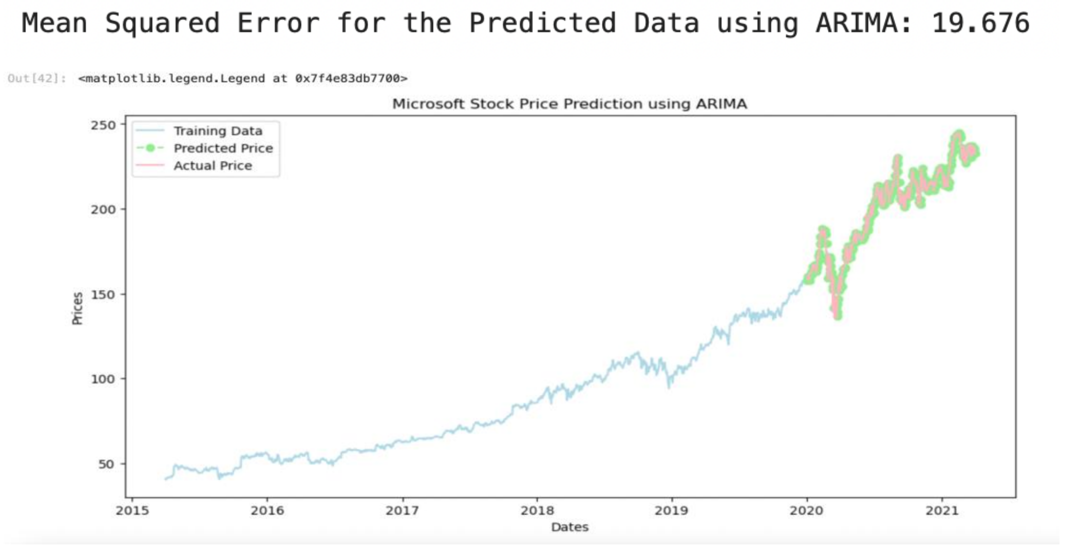




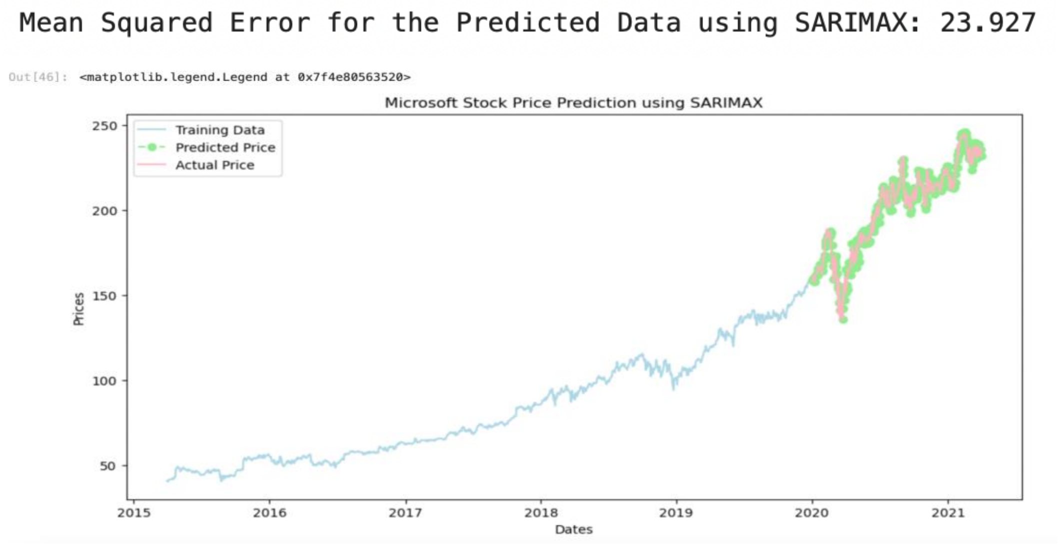
Decomposition Plots - These are used to see if there is any trend or seasonality in the data. Based on the graphs I understood that there is a growing trend and there is a seasonality as well.



ARIMA and SARIMA Model - Using all the insights from above I implemented two statistical models. I made graphs as well as calculated the MSE values for both. Based on the below results it can be concluded that ARIMA is the better model.

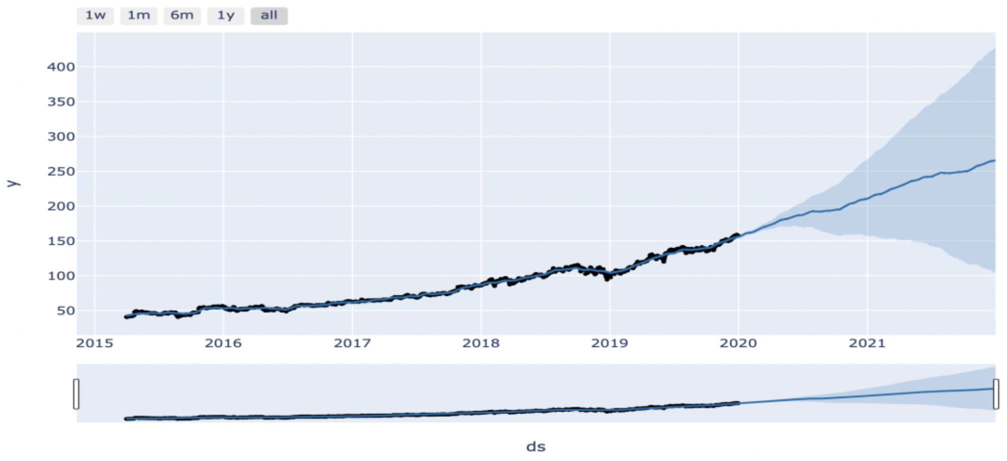


```
Mean Squared Error for the Predicted Data using ARIMA: 19.676

Out[42]: <matplotlib.legend.Legend at 0x7f4e83db7700>
```



```
Mean Squared Error for the Predicted Data using SARIMAX: 23.927

Out[46]: <matplotlib.legend.Legend at 0x7f4e80563520>
```

Prophet Model PySpark - This model was implemented to see how well PySpark would perform while doing Time Series Analysis as opposed to regular Panda. The model might not have done as well as the ones I implemented using Pandas but I could see how fast it was computing the results.



DISCUSSIONS

There are a few observations that could be made from the result section:

1. From the initial visualizations we can see that there is an increasing trend in the stock prices as the years increase. Also, we can see that except volume all the other columns are highly correlated from the correlation graphs.
2. As we compare the three models that were implemented to predict stock prices on the test data we can see that ARIMA performs the best with the least MSE value of 19.68 which means that the difference between the actual stock price and predicted stock price is around 19$ which is a pretty decent error to have.
3. While implementing I noticed that the model I ran using the PySpark library was faster compared to those that were implemented using Pandas which shows that distributed computing is a good method to use for this type of problem statement.

The tools and technologies and analysis that were implemented as a part of this project gave me a good real-world view for the theoretical concepts I learnt as a part of this course. The topics that were used from the course as a part of my project are as follows:

1. **Virtualization** - As a part of virtualization in the course we learnt about different tools like Jetstream2, GCP and there are many others like AWS, Azure, etc. The reason why I chose to even use virtualization in the first place is due to the dataset which is about stock prices for a company on a daily basis. This data is an everyday growing data and storing and analyzing this on a local system would not be the best option hence the use of Virtualization. In this I made a choice of using Jetstream2 because of all of its advantages that I had mentioned in the methodology section.
2. **Ingest and Storage** - Since this data is huge and it increases everyday as new stock prices for each day could get added to it, we need a good database to ingest and store it in, a database that enables us to store a variety of data that would let us to easily modify, store and access it. For this purpose, I went with MongoDB which is a NoSQL database.
   Ingest refers to the process of inserting data into a MongoDB database. In this project's case this was done using MongoDB's import/export tools using the dataset in the form of a csv file. Ingesting data is an important step in using MongoDB, as it is necessary to have data in the database before it can be queried and analyzed.
   Storage, on the other hand, refers to how data is stored within the MongoDB database. MongoDB stored this dataset in the form of documents, which are similar to rows in a traditional relational database. Each document is stored in a collection, which is similar to a table in a relational database. MongoDB uses a flexible schema that allows for dynamic and evolving data structures, which makes it well-suited for handling unstructured and semi-structured data.
   MongoDB stores data in a binary format called BSON (Binary JSON), which is a JSON-like format that supports additional data types such as dates which is the kind of data we are working with so this NoSQL database was pretty much ideal for this dataset.
3. **Lifecycle and Pipelines** -
   Lifecycle steps included in this project:
   a. **Plan** - This aspect was covered as for starting any experiment we should at least be able to plan out what data, software, etc. are needed based on our problem statement. If this part was not included, I do not think the experiment would have made sense.
   b. **Acquire** - I acquired the data from Kaggle. It is the data of daily stock prices of Microsoft over the years 2015 - 2021.

c. **Process** - I processed the data by working on the ‘Date’ column to get it to the preferred pattern I needed to work further with it.

d. **Analyze** - I analyzed the results with the help of a graph that I plot. The graph basically shows the predictions of the models.

e. **Preserve** - I then preserve the dataset along with the results by storing them in a GitHub Repository.

f. **Publish/Share** - I published and shared my results with others by making the repository public and sharing the link with others.

g. **Describe** - I had written markdown texts throughout the python notebook to help the user that is looking through this project understand what is going on in it.

High level architecture of the pipeline:

- I first started the pipeline with data ingestion from MongoDB
- Preprocessed the ‘Date’ column and checked for any missing values or outliers. In this case none were present.
- Visualized the data to understand the it
- Splitting data into Train and Test
- Model building
- Final predicted result of the pipeline in the form of a graph

4. **Analytics using PySpark** - I had implemented the prophet model using the PySpark library and saw the power of distributed computing. Even though the Pandas library was working almost equally good, the speed of processing was clearly better in PySpark. Since we are working with a growing dataset PySpark would be a better choice to help run analysis on the data.

Barriers and failures that occurred during the course of implementing this project:
The biggest barrier I had faced while working on this project was to get MongoDB to work on the VM. It kept giving me multiple errors which got resolved after a few days of spending time on trying to resolve the same. The error basically got resolved when I made a separate environment for installing everything that MongoDB needs by being careful of the versions of each of the installations that I was doing. Other than this everything that I had in mind for this project was implemented and the results are presented as a part of this report.

CONCLUSION

In this project we made use of the Microsoft Stock data to perform Time Series Analysis to predict the future stock prices based on historical stock price data. We made use of the Jetstream2 VM as our virtualization tool where everything is hosted. MongoDB was utilized to ingest and store our dataset.

We pulled the data from MongoDB and performed preprocessing, visualizations and implemented three models which are ARIMA, SARIMAX and Prophet using Pandas dataframe for the first two and PySpark dataframe for the third one. We observe that ARIMA proved to be the best model in terms of its predictions with the least calculated MSE value of 19.68. In terms of processing PySpark was better than Pandas.

Overall, this project helped me connect various concepts that I learnt as a part of this course and apply them in a real-world setting. I had lots of fun implementing this and learning the intricate details of each of these concepts while implementing them.

REFERENCES

[1.] https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/

[2.] https://stackoverflow.com/questions/49554139/boxplot-of-multiple-columns-of-a-pandas-dataframe-on-the-same-figure-seaborn

[3.] https://venngage.com/blog/pastel-color-palettes/

[4.] https://practicaldatascience.co.uk/data-science

[5.] https://towardsdatascience.com/pyspark-import-any-data-f2856cda45fd

[6.] https://online.stat.psu.edu/stat510/lesson/4/4.1

[7.] https://www.analyticsvidhya.com/blog/2021/08/understanding-bar-plots-in-python-beginners-guide-to-data-visualization/

[8.] https://www.mongodb.com/docs/spark-connector/current/python-api/

[9.] https://www.mongodb.com/docs/launch-manage/

[10.] https://www.cnbc.com/quotes/MSFT

[11.] https://www.kaggle.com/datasets/vijayvvenkitesh/microsoft-stock-time-series-analysis