Group Project Report: Stocks Big Data Project

Course Name: BUDT737: Big Data and Artificial Intelligence for Business
Group Name: 5:00 PM GROUP NO ROW
Group Members: Shubhangkar Girish Jain, Nishit Salot, Kathan Shah

Introduction:

Stock Markets have been one of the most lucrative ways to make money since the beginning of time. Predicting and Analyzing Stock Prices has been one of the biggest problems in the stock markets. To minimize speculations and increase the accuracy of data this project will focus on Analyzing and Predicting Stock Market movements and volatility. We will also be using the ARIMA model to predict future stock prices in this project..

The main focus of this project would be the Indian Stock Markets and more specifically the main index of Indian Markets and select stocks in that

Data Process

Initially, we stored the data in a Cloudera EC2 instance and attempted to run Impala queries but we ran into some issues and hence tried to use some other technology.

We then decided to store the data in an Amazon S3 bucket. Then we set up an Amazon EMR instance and used PySpark in Jupyter Notebook to run predictive and descriptive analyses on the data. We used the ARIMA model to conduct predictive analysis on stock prices.

Business Question:

The business questions we will be solving include:

- What is the price of the stock and the end of each fiscal year?
- Which stock has the highest returns?
- What trend is the stock following?
- List the days in which the stock experiences maximum volatility.

Data Set and Infrastructure:

This dataset is from Kaggle (https://www.kaggle.com/datasets/hk7797/stock-market-india) This dataset currently contains the 1-minute dataset of 150 NSE stock (50 Nifty 50 stock, 100 Nifty Midcap stocks) and 9 indices starting from 2017–01–01. The dataset contains millions of data points to be analyzed over the period of 4 years.

The data combined consisted of around 3GB. To analyze such huge datasets we pushed this data over in S3 buckets and accessed it from there to work in a cloud-based environment with more resources at our exposure.

Below are the variables included in the dataset analyzed:

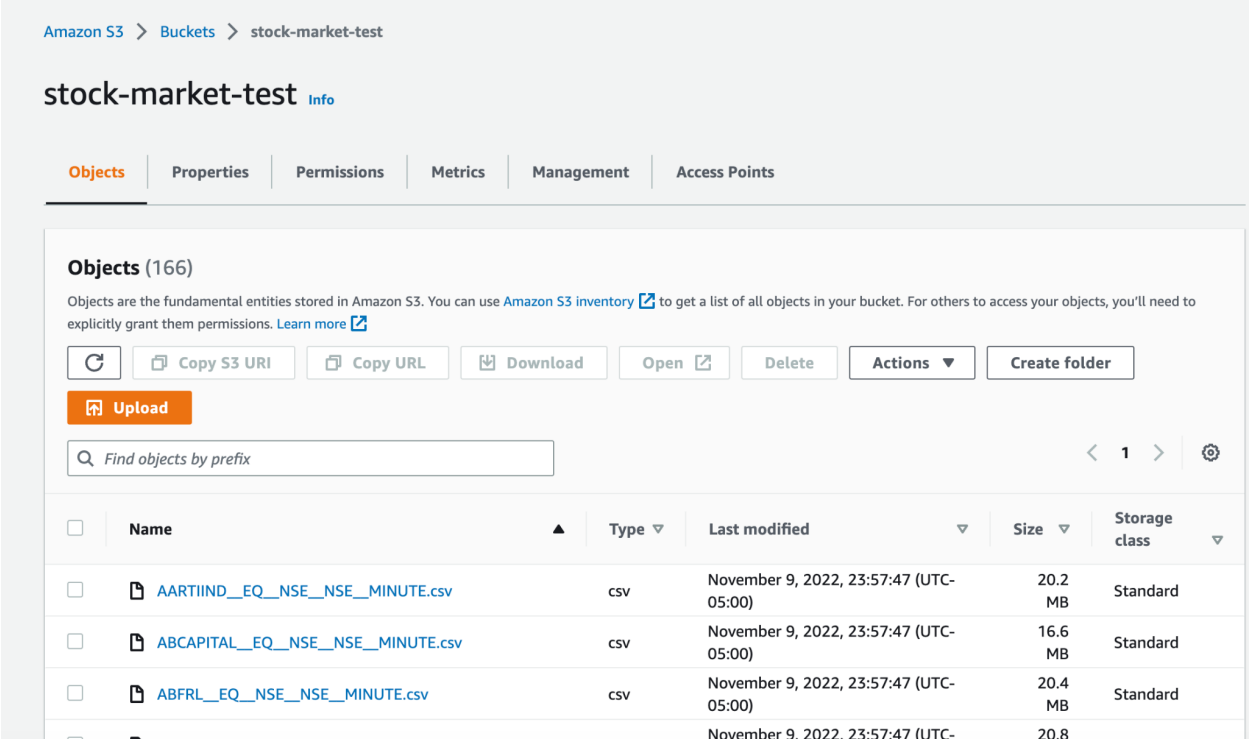
Variable Name	**Datatype**	**Description**	**Example**
timestamp	DateTime	Date and Time	12 Oct 2017, 9:45:00 am
open	float	Price at which security started trading that day	230
low	float	Lowest price till that time for that security on that day	108
high	float	Highest price till that time for that security on that day	250
close	float	Price at which security closed trading on that day or the last known price of the security on that day	209

To load the data a bucket was created in Amazon S3 called Stock-Market-Data. We uploaded all the CSV files in the bucket.

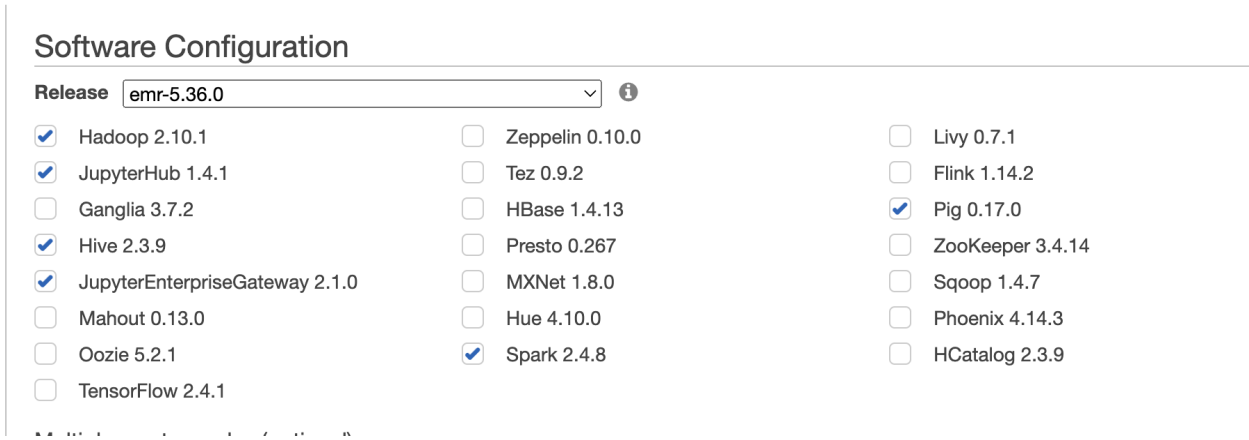
Next, through Amazon EMR, a cluster was created with the Stock-Market name, and we added steps to ensure logs and outputs were directed to their respective folder locations previously created. To gain access to our cluster, through Amazon EC2, a keypair was created. Our EMR cluster had 2 instances running for better performance, a master and a core, both with m5.xLarge configurations.

Once the EMR server was up and running we created a notebook and attached it to our cluster and bucket. We then ran SparkSQL queries on the dataset using PySpark to answer a few of the business questions and descriptive analytics on our data.

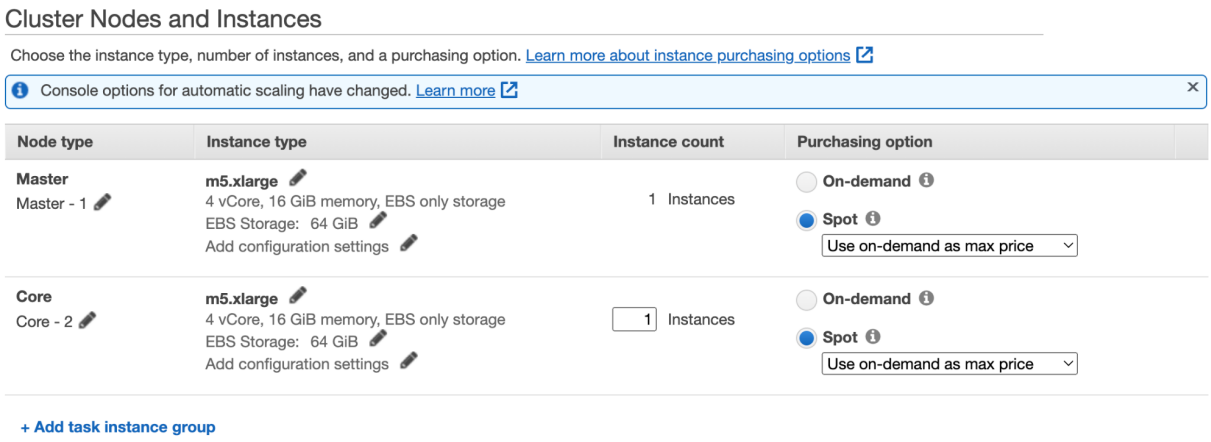
Snapshot of the S3 bucket



Snapshots of our EMR CONFIGURATIONS:



Instances used for processing of our data.



Additionally, we used boto3 to access the S3 bucket locally and work on the python file on our machine to build an ARIMA predictive model for the stock prices.

Data Analysis:

In order to better understand the stock market data, we performed certain descriptive analytics and answered some of the business questions that make sense when relating to the stock market data.

We used PySpark through the EMR instance to generate SQL queries and get results for half of the business questions and for the other half, we ran it on our local machines using Spyder wherein we imported our data from the S3 buckets using boto3.

The first question that we wanted to solve was how to make the data more readable. Since our data was minute to minute, it was not very user-friendly to read the data and most of the numbers differed in decimal places. So to answer this question, we wrote an SQL query in the PySpark terminal and got the below results.

In [131]:
```
#Resampling on Daily Basis
df.createOrReplaceTempView('data');
query2 = spark.sql("SELECT DATE(timestamp) as Date,FIRST(open) as OPEN,MIN(low) as LOW,MAX(high) as HIGH,LAST(close) as
query2.show(50)
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

+----------+------+------+------+------+--------+
|      Date|  OPEN|   LOW|  HIGH| CLOSE|  VOLUME|
+----------+------+------+------+------+--------+
|2017-01-02| 340.0| 337.3|348.95| 346.0|  8372.0|
|2017-01-03| 349.5| 341.8| 355.0| 350.0| 20826.0|
|2017-01-04| 353.3| 341.5|355.65| 350.0|131883.0|
|2017-01-05|352.35|349.55|354.95| 351.5|  8818.0|
|2017-01-06| 352.0| 349.0| 354.9| 349.1|  8128.0|
|2017-01-09|351.75| 348.0| 352.0| 348.5|  6310.0|
|2017-01-10| 350.0| 348.0|352.25| 350.5| 10186.0|
|2017-01-11| 350.5|346.75|351.35|351.25| 17155.0|
|2017-01-12| 350.5|349.05|351.95| 350.0|  8469.0|
|2017-01-13| 348.1| 348.1| 354.0|351.95| 12967.0|
|2017-01-16| 351.9| 350.0| 355.0|353.95|  8673.0|
|2017-01-17|353.55| 352.1| 354.5| 354.0|  7386.0|
|2017-01-18| 355.0| 352.5|360.75| 356.5| 27663.0|
|2017-01-19| 355.2|350.05| 359.7| 352.5| 16641.0|
|2017-01-20| 355.0| 342.5| 355.0|343.25| 28588.0|
|2017-01-23| 340.6|336.55|353.35| 353.0| 15594.0|
|2017-01-24| 350.0| 346.5|353.85| 346.5| 20011.0|
|2017-01-25| 347.1| 345.6|354.25| 351.5| 62783.0|
|2017-01-27| 352.5|352.45| 362.2| 360.5| 45212.0|
|2017-01-30| 361.5| 358.6| 364.9| 359.5| 43072.0|
|2017-01-31|357.55| 356.5|361.85| 360.0| 26472.0|
|2017-02-01|359.95|355.65| 367.5| 365.9| 23784.0|
|2017-02-02|367.45| 366.5| 377.5| 369.0| 36047.0|
|2017-02-03|371.95| 370.1| 385.0| 380.0| 60940.0|
|2017-02-06| 386.0|377.55| 398.3|390.95| 68809.0|
|2017-02-07| 394.5| 387.5| 399.5| 393.6| 37756.0|
|2017-02-08| 392.5|388.05|397.75|395.25| 31070.0|
|2017-02-09| 395.3| 386.5|398.75| 389.0| 13895.0|
|2017-02-10|393.25| 386.3|395.85| 395.0| 18439.0|
|2017-02-13|390.05|390.05|401.45| 396.8| 21977.0|
|2017-02-14| 398.4|391.05|398.95| 391.1| 10576.0|
|2017-02-15| 390.0| 386.5| 394.9| 386.5| 14547.0|
|2017-02-16| 386.0| 385.5|398.15| 395.0| 15682.0|
|2017-02-17| 392.5| 392.5|399.85|399.45| 71913.0|
|2017-02-20| 399.9| 392.8|399.95|393.25| 39114.0|
|2017-02-21|398.85| 393.5|398.85|394.05|  6570.0|
|2017-02-22| 397.7| 390.0| 397.7|392.45| 13446.0|
|2017-02-23|387.65| 384.5| 394.7| 385.5| 22312.0|
|2017-02-27|391.95|384.55|394.95| 392.5| 24916.0|
|2017-02-28| 393.2|386.65|394.95| 388.0| 10625.0|
|2017-03-01| 389.5|385.15|391.55| 386.3|  8594.0|
|2017-03-02|326.65|326.65|389.95| 382.0| 29332.0|
|2017-03-03| 382.0| 367.5| 384.8| 370.0| 28567.0|
|2017-03-06| 372.1|358.95|375.95| 372.1| 35631.0|
|2017-03-07|363.55| 363.5| 377.5|369.25| 13517.0|
|2017-03-08|372.15|363.75| 393.3| 386.5|153097.0|
|2017-03-09|385.45| 380.1|392.45|387.45| 28129.0|
|2017-03-10| 387.5| 384.0| 390.5| 387.0| 42900.0|
|2017-03-14| 390.0| 382.5|392.45| 385.5| 22723.0|
|2017-03-15|386.45|385.05| 397.5| 394.5|100879.0|
+----------+------+------+------+------+--------+
only showing top 50 rows
```

As you can see, the data is much more readable now and the user can observe how the price is changing each and every day.

The second question we decided on was to identify the percentage return of a stock over the entire time frame. Returns are one of the most important concepts when it comes to financial data and hence calculating it was our top priority.

In this question, we have calculated the returns using SQL in PySpark and have showcased it for one particular table which can be replicated across any table.

```
In [160]: #Returns on a Stock
df.createOrReplaceTempView('data');
query3 = spark.sql("SELECT LAST(close)/FIRST(close)* 100 as Percentage_Return from data")
query3.show()
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

+-----------------+
|Percentage_Return|
+-----------------+
|370.5882352941177|
+-----------------+
```

From the above screenshot, we clearly see that this particular stock has given us a return of close to 370%

For the third question, we wanted to calculate the volatility of a particular stock. Volatility tells us how much a stock actually moves on a particular day, be it upside or downside, and is an important measure to calculate the risk of the stock. This risk can also be used with return in order to calculate another important ratio called Sharpe which gives us the return over the risk of the equity.

To calculate the volatility, we subtracted the first open price available to us from the last close price available for a particular day and stored it as volatility. Then, we sorted these results in descending order to display the days where we had maximum volatility.

```
In [174]: #Amount Traded
df.createOrReplaceTempView('data');
query4 = spark.sql("SELECT DATE(timestamp) ,ROUND(LAST(close)-FIRST(open),2) as volatility from data group by DATE(time
query4.show()
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

+----------+----------+
| timestamp|volatility|
+----------+----------+
|2020-04-27|      95.0|
|2020-03-13|      88.8|
|2020-01-29|      88.0|
|2018-01-05|      85.0|
|2020-04-28|      68.0|
|2020-08-20|      57.1|
|2017-03-02|     55.35|
|2020-11-05|      54.0|
|2020-11-27|     51.35|
|2019-10-18|      48.5|
|2019-05-30|      45.0|
|2020-03-31|      44.0|
|2018-10-03|      43.0|
|2020-02-20|     42.15|
|2019-09-20|      41.2|
|2020-12-07|     41.15|
|2020-08-19|     40.95|
|2018-08-30|     39.75|
|2020-01-28|     39.15|
|2019-02-04|      38.5|
+----------+----------+
only showing top 20 rows
```

One interesting inference from this is that the majority of the days are in the year 2020 and also post-March which is the period when Covid onset in India. This also tells us how this virus dented the economy and how affected the stock market was in every country.

For the next question, we wanted to display the stock price of any equity at the end of each Fiscal year. We chose the fiscal year because most of the companies get their maximum revenue and sort their account books in the last quarter and we get to see the real price of the equity during this phase.

For this particular question, we have chosen Union Bank as our equity but it can be performed for any security.

Year	Value
2017	155.96093333333346
2018	94.97439999999999
2019	98.12440000000004

As you can see from the screenshot above, the price of the stock declined from 2017 to 2018 and just rose slightly from 2018 to 2019 indicating that there was not much growth at Union Bank and they needed to reformulate their strategies in order to increase their revenue or earnings to increase their share price.

In this question, we wanted to identify what trend the particular stock was following. A trend can help us choose a particular strategy and assist in identifying the right moment at which the stock might be purchased. If you are going long, then it is better to buy the stock when a downward trend is over and an upward trend is starting and vice-versa if you want to sell the stock. We have used python to identify the trend.

We have chosen the ITC stock as an example and as mentioned earlier, it can be used against any security.

```
ITC = df[80]
if ITC[-7:-6]['open'].values < ITC[-1:]['open'].values:
    print("This stock has an Upward Trend")
else:
    print("This stock has an Downward trend")
```

```
This stock has an Upward Trend
```

We can infer that ITC is following an upward trend and hence it would be a good time to purchase this stock or wait until a downward trend starts to sell the stock.

For the last question, we utilized all the securities in our dataset and identified which equity had the maximum return. We performed logarithmic daily returns and performed a cumulative

multiplication to get the total cumulative return for a security. We then exponent and raise it to this value (e^{value}) to get the actual total returns. We then ran a for loop to get the maximum value of all the returns.

```
: #Q3 Which stock gives the highest return?
df.loc[:,'daily_return'] = np.log(df['close']/df['close'].shift(1))
df= df.dropna()
df.loc[:,'cum_return'] = df['daily_return'].cumsum().apply(np.exp)
maximum = 0
index = 0
for i in range(0,159):
    temp=0
    temp = df[i][-1:]['cum_return'].values
    if temp>maximum:
        maximum = temp
        index=i

print("The stock Dhani gives the highest return of",maximum[0]*100,"%")
```

```
The stock Dhani gives the highest return of 1565.9289754748181 %
```

From the screenshot, we can infer that Dhani stock (Healthcare and Financial Services) has the highest return of close to 1600%.

These are some of the descriptive questions that we answered which are relevant to the stock market data.

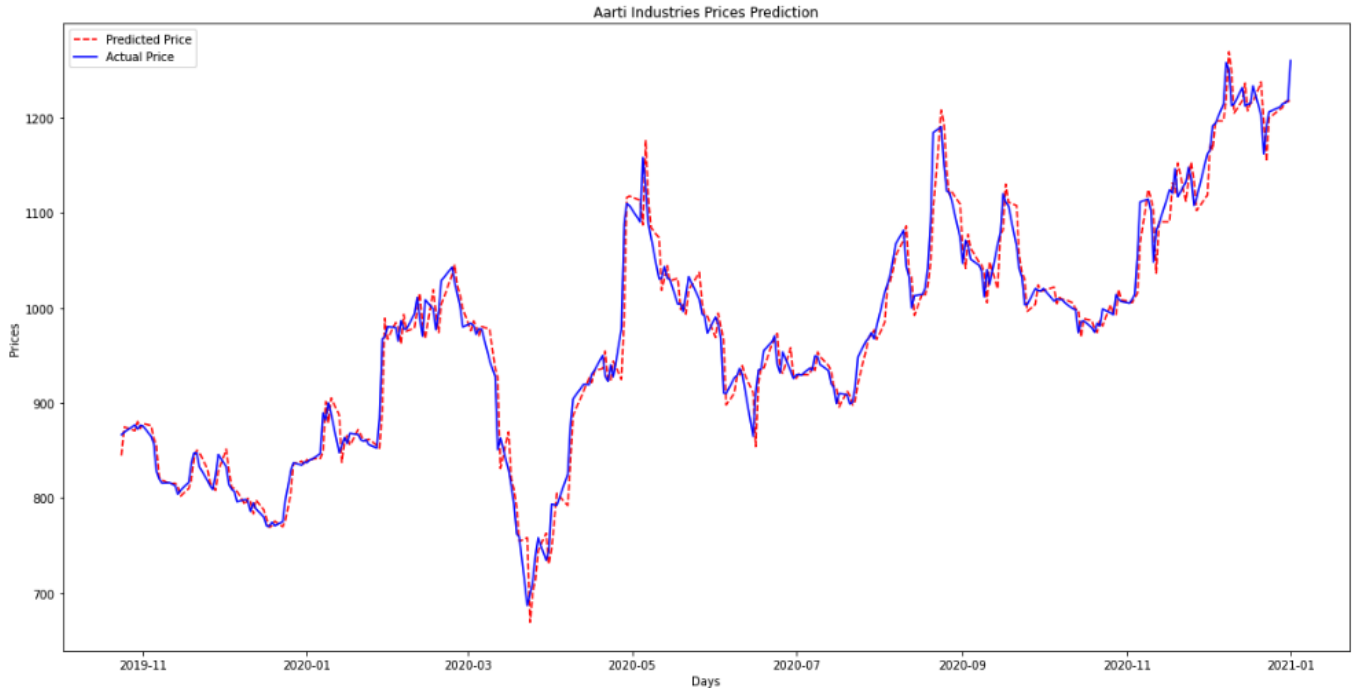
Prediction Model (ARIMA)

ARIMA, or **Autoregressive Integrated Moving Average**, is a type of time series forecasting model that is used to predict future values based on the past values of a time series. It is a linear model that attempts to capture the underlying structure of a time series, including any trends, seasonality, and residual errors. The model is specified by three parameters: the autoregressive (AR) term, the difference (I) term, and the moving average (MA) term. Together, these parameters determine the behavior of the model and can be adjusted to fit the time series data as closely as possible. ARIMA models are commonly used in economics, finance, and other fields where time series data are collected and analyzed.

In this project, we will use the ARIMA model to predict the future values of a stock market index. The ARIMA model is a type of time series forecasting model that is well-suited for predicting future values of a time series based on its past values. We will use historical data on the stock market index to train the ARIMA model, and then use the model to make predictions values of the index. This project aims to develop a model that can accurately forecast the future behavior of the stock market, which could be useful for investors and traders who want to make informed decisions about buying and selling stocks.

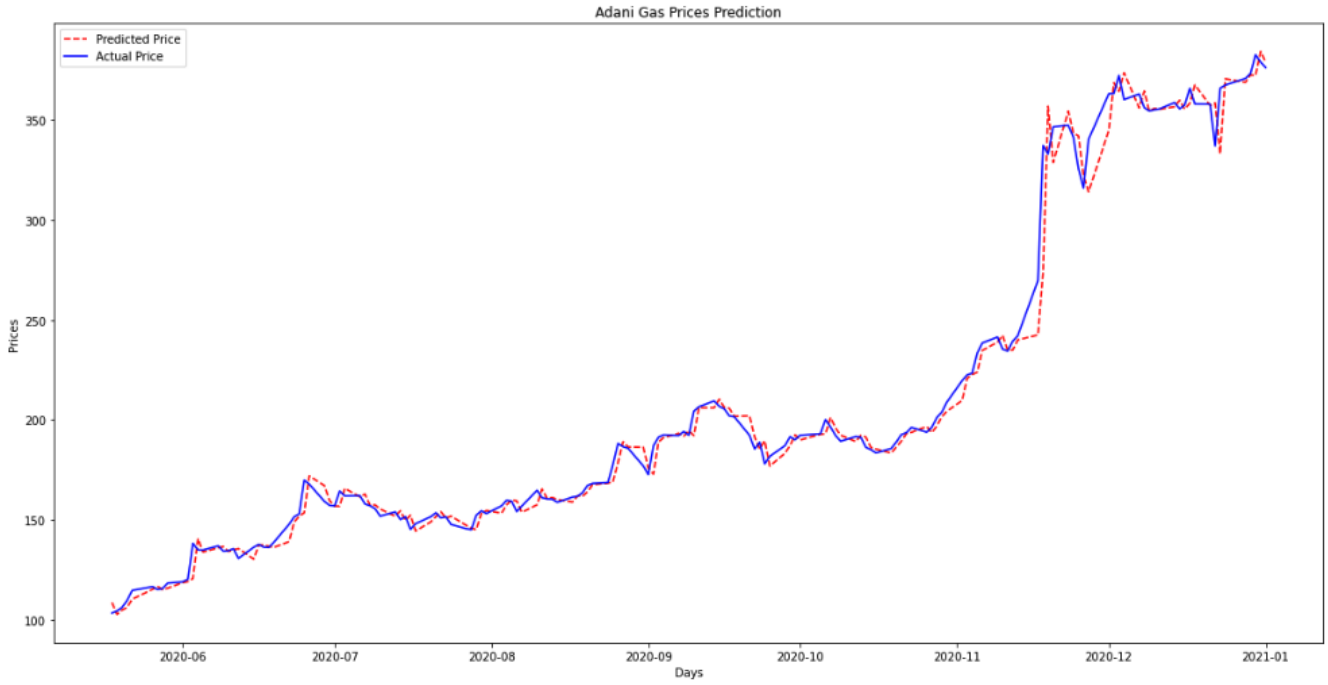
For this model, we just need the close price of the stock for the given time period and split that data into training and testing. The statistical measure used to evaluate our performance is Mean Squared Error.

```
Testing Mean Squared Error is 506.4753180312519
```



As we can see above, the model does a good job in predicting the price of Aarti Industries. The red-dashed lines indicate the predictions made by the model while the blue line is the actual data.The performance is reflective in our Mean Squared Error which is just 506.

```
Testing Mean Squared Error is 76.29030694398338
```



When we see the predictions for Adani Gas, we see that the model has performed better when compared with Aarti Industries. Though there are several places where we see the lines do not overlap, it overall does a good job and gives a very low Mean Squared Error of 76.

CONCLUSION

Overall, we can conclude that stock prices are volatile and not easy to predict, yet some factors contribute to volatility more than others. While we think that more effort and analysis are required to build out a model accurate enough for use by traders and investors, here are the findings that we came across:-

1. Prediction models should not be used to trade the stock markets with real money.

2. Analyzing previous prices is a good way to get a rough idea of future price movements.

3. Traders and Investors can use the findings of this model and method as one of the factors while analyzing data.

4. Stock market price movements are controlled a lot more by external factors than they are by previous numbers and patterns

To improve our analysis, we thought of several other methods including Regression, SVM, time outlier detection, and analyzing detailed stock factors and numbers including external factors. And also, Suggesting tactical moves to maximize gains. After further analysis, the model will have more predictive power to improve overall efficiency and increase profits for traders and investors.