**Stock Price Analysis and Prediction**

Kaustuv Majumder (50360369)

Ieshaan Sharmaa ()

Niraj Chetry ()

December 2021

1. **Introduction**

Yahoo finance API is a popular data source for financial data related to stock market. This data can be used to get the live data of stock, as well as historical stock performance.

Financial data is of great importance for investors and analysts as they tell us about the health of the companies as well as the market in general.

1. **Objective**

The aim of this project is to extract all stock related information from Yahoo finance and store it in relational data models. This project also aims to demonstrate the end-to-end data ecosystem using a practical use case. The project has the following components:

* Data extraction from API (Yahoo Finance API)
* Data transformation and enrichment
* Data modelling
* Data quality check
* Predictive Analytic
* Sentiment Analysis

1. **Target Users**

The following user groups have been identified so far:

* Retail stock investors
* Stock market analysts

1. **Database Details**

The data model has been designed to accommodate stock information. This consists the following:

* Historical data of stocks (Day wise trend)
* Live status from the market
* Analytical Data related to stocks
  1. **Data Source**

Data from Yahoo finance has been used to build this project. yfinance is a popular python wrapper to access this API for free. Thus, this library has been used in this project to access Yahoo Finance API.

* 1. **Data Model**

The relational model consists of the following tables.

Note - Some of these tables are used for storing actual financial data; while other relations are used to support analytic and predictions.

**COMPANY**

This table contains basic information of the company

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** | **Description** |
| SYMBOL | Varchar(5) | Primary Key | The stock symbol as per the market convention |
| NAME | Varchar(50) | Not Null | The company name |

**COMPANY\_DETAIL**

This table contains detail information of the company

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint 1** | **Constraint 2** | **Description** |
| SYMBOL | Varchar(5) | Unique | Foreign Key (COMPANY.SYMBOL) | The stock symbol |
| CITY | Varchar(20) | Not Null |  | The city in which the company is registered |
| SECTOR | Varchar(50) | Not Null |  | Company’s business (e.g. Finance, Technology) |
| SUMMARY | Text | Not Null |  | The company description as provided by Yahoo finance |

**STOCK\_HISTORY**

This table contains the historical information of the stocks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint 1** | **Constraint 2** | **Description** |
| SYMBOL | Varchar(5) | Primary Key | Foreign Key (COMPANY.SYMBOL) | The stock symbol |
| DATE | Date |  | The record date |
| OPEN | Float8 | Not Null |  | Valuation at market open on that day |
| HIGH | Float8 | Not Null |  | Highest intraday valuation on that day |
| LOW | Float8 | Not Null |  | Lowest intraday valuation on that day |
| CLOSE | Float8 | Not Null |  | Valuation at market closure on that day |
| VOLUME | Int8 | Not Null |  | The volume of stocks in circulation |
| DIVIDEND | Int8 |  |  | Dividend announced (if any) |
| STOCK\_SPLIT | Int8 |  |  | Stock Split announced (if any) |

**STOCK\_MATRIX**

This table contains the live stock data as fetched from the api.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint 1** | **Constraint 2** | **Description** |
| SYMBOL | Varchar(5) | Unique | Foreign Key (COMPANY.SYMBOL) | The stock symbol |
| DAY\_HIGH | Float8 | Not Null |  | Highest intraday valuation |
| DAY\_LOW | Float8 | Not Null |  | Lowest intraday valuation |
| PRICE | Float8 | Not Null |  | Current price |
| RECOMMENDATION | Varchar(5) | Not Null |  | Buy/Sell recommendation |

**MEANS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** | **Description** |
| SYMBOL | Varchar(5) | Foreign Key (COMPANY.SYMBOL) | The stock symbol |
| MEAN | Text | Not Null |  |
| TARGET\_MEAN | Float8 | Not Null |  |

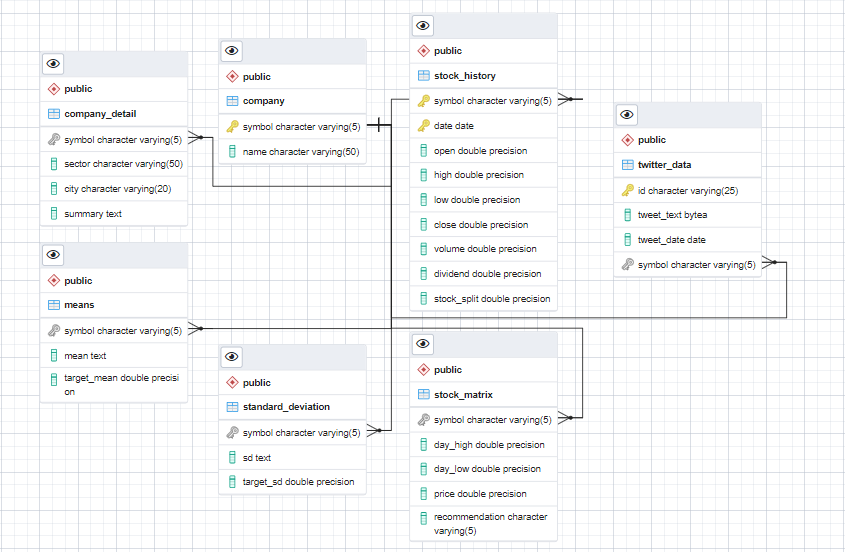
**STANDARD\_DEVIATION**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** | **Description** |
| SYMBOL | Varchar(5) | Foreign Key (COMPANY.SYMBOL) | The stock symbol |
| SD | Text | Not Null |  |
| TARGET\_SD | Float8 | Not Null |  |

**TWITTER\_DATA**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Constraint** | **Description** |
| ID |  | Primary Key |  |
| TWEET\_TEXT |  | Not Null |  |
| TWEET\_DATE |  | Not Null |  |
| SYMBOL |  |  | The stock symbol |

* 1. **Entity Relation Diagram**



* 1. **Update Policy.**

The tables are updated as per the following logic

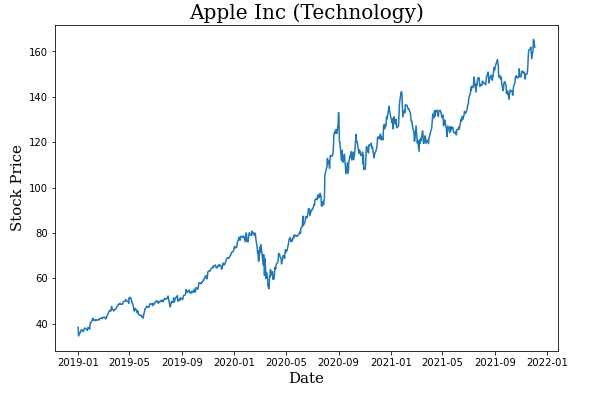
* Company - Static Table, manually updated if required
* Company Detail - Static table, manually updated if required
* Stock History - Updated once every day
* Stock Matrix - Updated as per user request
* Mean – Static table. Updated only if the LSTM model is re-trained
* Standard Deviation - Static table. Updated only if the LSTM model is re-trained
* Twitter Data – Updated when new tweets are fetched from twitter api.
  1. **Functional Dependencies**

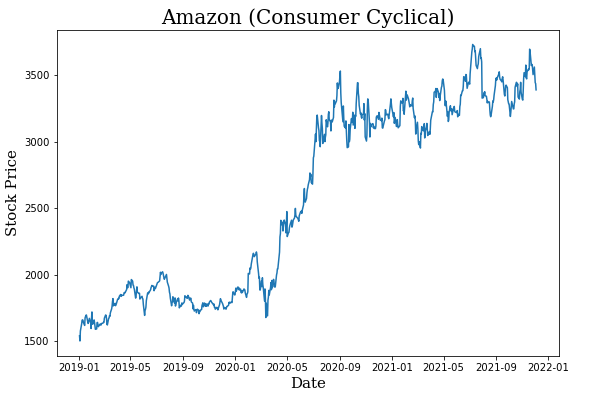
|  |  |
| --- | --- |
| **Relation** | **Functional Dependencies** |
| COMPANY | Symbol -> Name |
| COMPANY\_DETAIL | Symbol -> Sector  Symbol -> City  Symbol -> Summary |
| STOCK\_HISTORY | {Symbol, Date} -> Open  {Symbol, Date} -> High  {Symbol, Date} -> Low  {Symbol, Date} -> Close |
| STOCK\_MATRIX | Symbol -> Day High  Symbol -> Day Low  Symbol-> Price  Symbol } -> Recommendation |

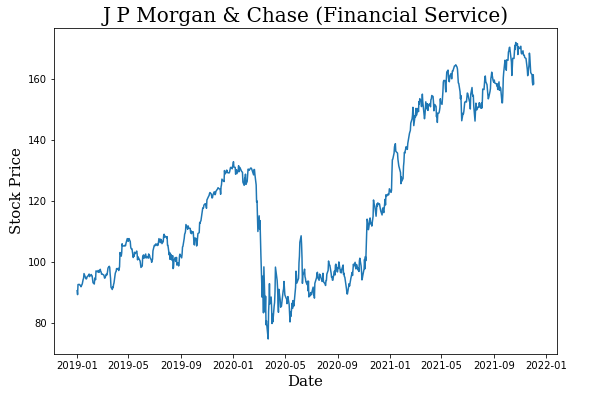
1. **Execution Steps**
2. Execute drop.sql to drop the existing tables. This is just to make the project re-runnable.
3. Execute create.sql to create all the relations.
4. Execute load.sql to load data from csv files.
5. Execute Update.py to update the database with the latest market data (optional)
6. Execute Data\_Analysis.ipnyb notebook to visualize the data analysis.
7. Execute Prediction.py for stock prediction
8. Execute Sentiment.py for sentiment analysis.

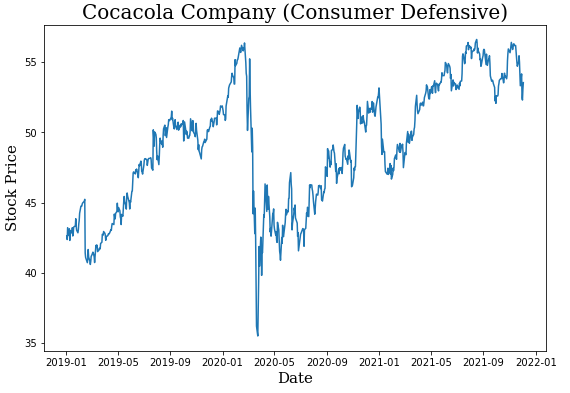
Note – User can use Stock\_Insight.ipnyb notebook to get all the information aggregated at one place.

1. **Data Analysis**
2. **Best Performing stock in each sector (Last three years)**

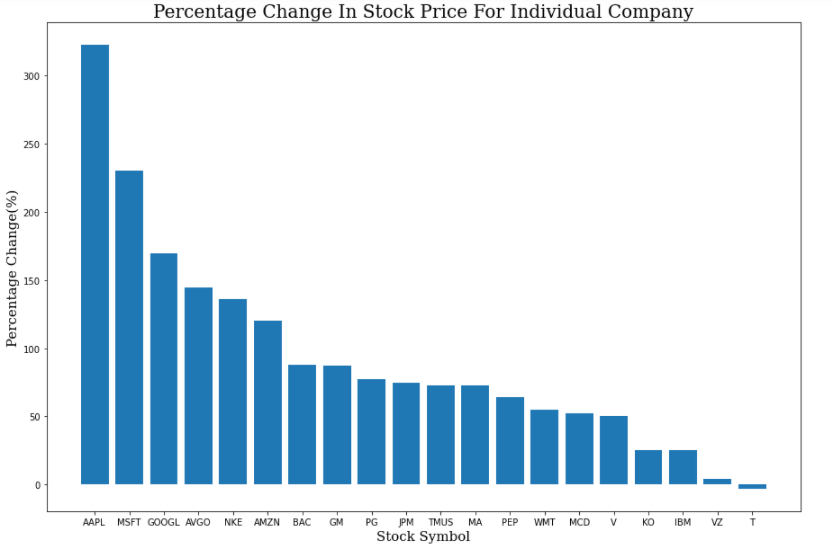




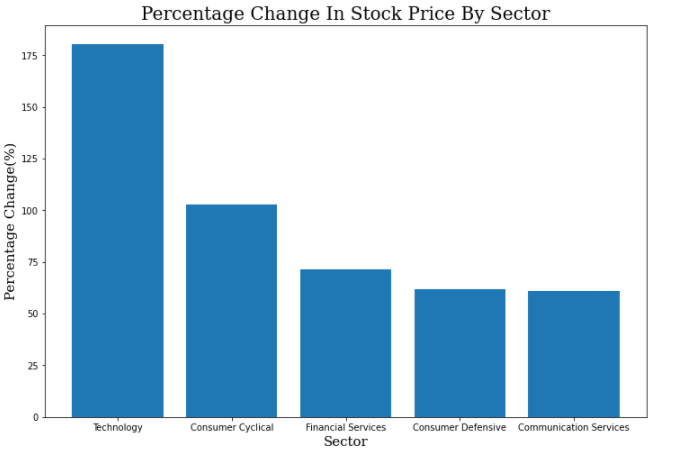




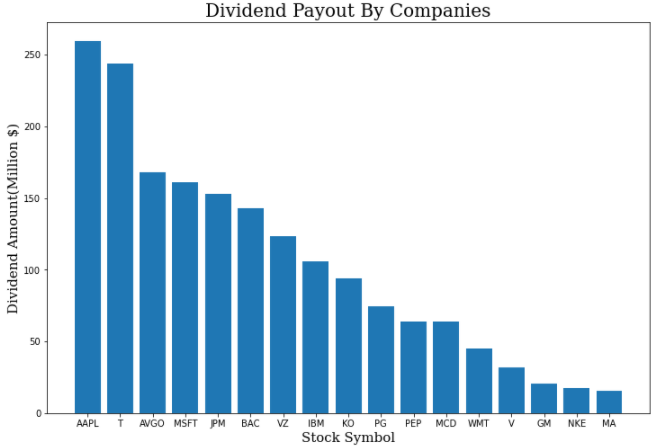
1. **Percent change of stock price for each company in the last three years**



1. **Percent change of stock price for each sector in the last three years**



1. **Dividend Payout by Companies in the last three years**



1. **Query Performance Analysis**

The performance analysis of some of the complex queries have been performed. Below are the results:

* **Percent change of stock price for each company**

*with a as (*

*select ROW\_NUMBER() OVER (order by symbol) as row\_, symbol, date, close*

*from stock\_history),*

*b as (*

*select symbol, min(row\_) as min\_row, max(row\_) as max\_row*

*from a*

*where date> to\_date('01012019', 'mmddyyyy')*

*group by symbol),*

*c as (*

*select a.symbol, a.close as oldest\_close*

*from a, b*

*where a.symbol = b.symbol and a.row\_ = b.min\_row),*

*d as (*

*select a.symbol, a.close as latest\_close*

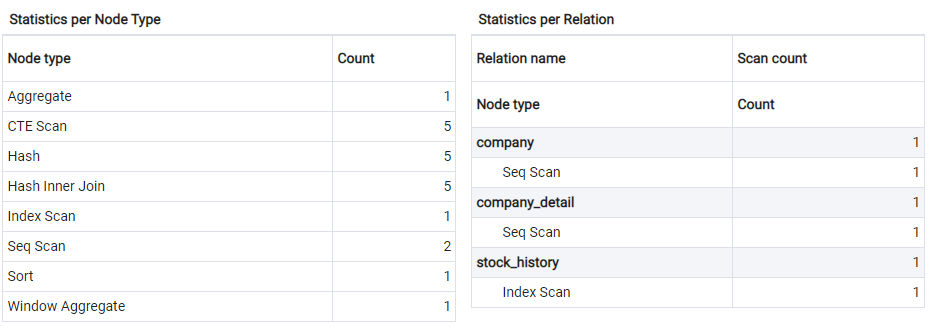
*from a, b*

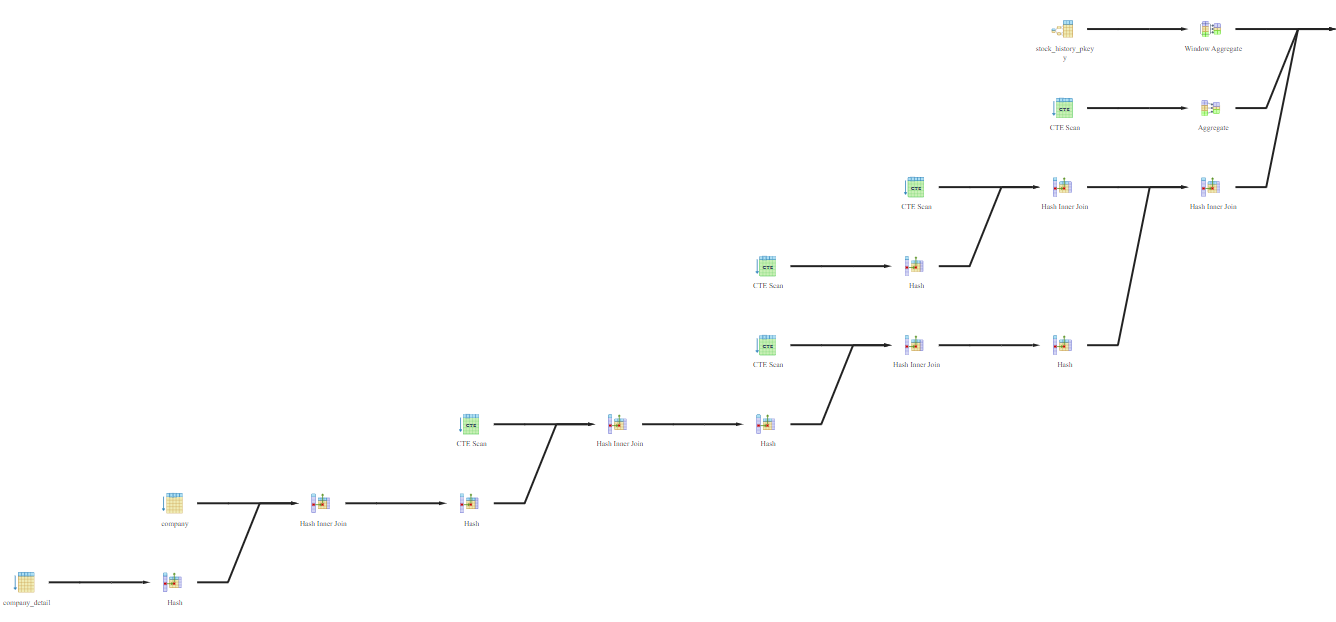
*where a.symbol = b.symbol and a.row\_ = b.max\_row)*

*select \*, ((latest\_close-oldest\_close)/oldest\_close)\*100 as pcpt\_change*

*from c natural join d natural join company natural join company\_detail*

*order by pcpt\_change desc*





* **Percent change of stock price for each sector**

*with a as (*

*select ROW\_NUMBER() OVER (order by symbol) as row\_, symbol, date, close*

*from stock\_history),*

*b as (*

*select symbol, min(row\_) as min\_row, max(row\_) as max\_row*

*from a*

*where date> to\_date('01012019', 'mmddyyyy')*

*group by symbol),*

*c as (*

*select a.symbol, a.close as oldest\_close*

*from a, b*

*where a.symbol = b.symbol and a.row\_ = b.min\_row),*

*d as (*

*select a.symbol, a.close as latest\_close*

*from a, b*

*where a.symbol = b.symbol and a.row\_ = b.max\_row),*

*e as (*

*select \*, ((latest\_close-oldest\_close)/oldest\_close)\*100 as pcpt\_change*

*from c natural join d natural join company natural join company\_detail*

*order by pcpt\_change desc*

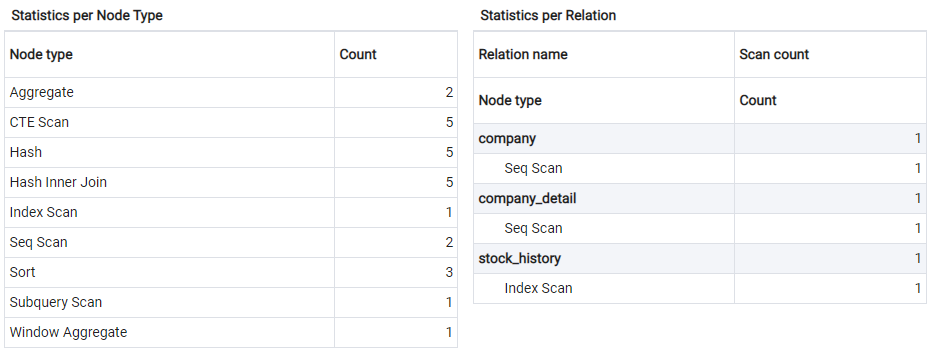
*)*

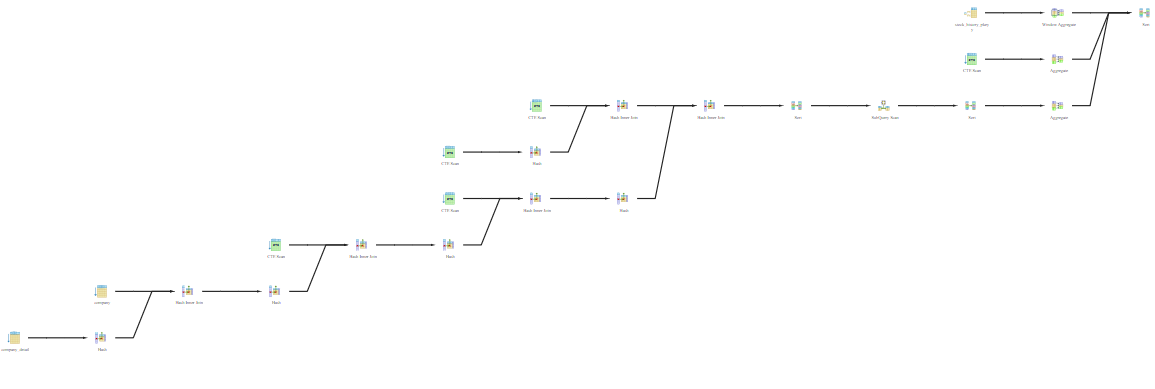
*select sector, sum(pcpt\_change)/count(\*) as pcpt\_change\_by\_sector*

*from e*

*group by sector*

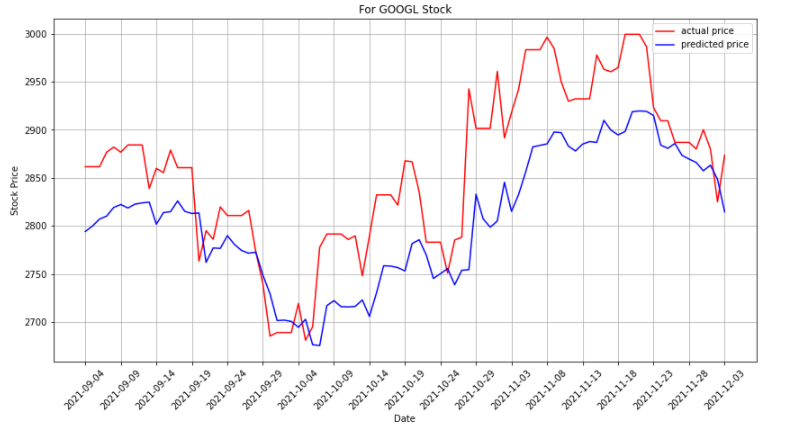
*order by pcpt\_change\_by\_sector desc*



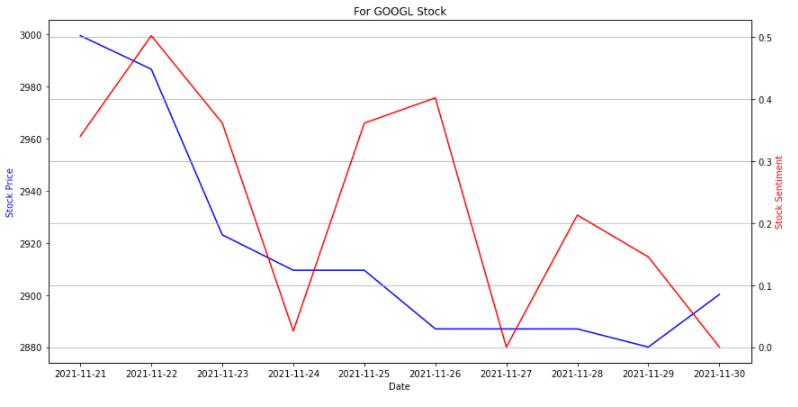


1. **Predictive Modelling**

The stock-market data is analyzed and modelled to design a live prediction for stock price. The historical data obtained from Yahoo Finance API is non-stationary time-series data and this information is utilized to train a Deep Learning model to predict the future stock price. *Long-Short Term Memory* network or shortly *LSTM*, which is an enhanced version of *Recurrent Neural Network* (*RNN*) is applied for this purpose. *LSTM* models are very appropriate for complex time-series data, as it can store and memorize very old information. The historical data is divided into Train and Test followed by applying the LSTM model on the Train data to determine the performance on unseen Test data. Attached below is a snap for the actual stock price vs predicted stock price for GOOGLE obtained from the model. For the next phase of the project, the saved models for different stocks will be applied for live prediction in Front-End.



1. **Sentiment Analysis**



1. **Conclusion**