**Project Report – 2**

**On**

**Stock Market Prediction System**

**Subject Code: 3IT31**

**Subject: Mini Project**

**Academic Year: 2023 – 2024**

**Team Members: Pratham Satani (21IT417)**

**Vatsal Patel (21IT414)**

**Deep Hirapara (21IT403)**

**Batch: D – 10**

**Project Guide: Prof. K. J. Padhiyar**



**Birla Vishvakarma Mahavidyalaya**

**Engineering College, Vallabh Vidyanagar**

**[An Autonomous Institution]**

**14. Project Implementation (up to 60%)**

14.1 Data Acquisition and Management

* Aim
* Ensure data is captured without loss or corruption from diverse sources.
* Implement error-checking, and validation mechanisms to maintain data integrity.
* Handle real-time and/or historical data without delays.
* Clean, normalize, and transform raw data into usable, structured formats.
* Dataset
* Data to be used in this project will be downloaded from [finance.yahoo.com](https://finance.yahoo.com/).
* This data is free and fully open-source.
* Format of this data will be in CSV (comma-separated values) format.
* Timeframe of the data will be 10 years back from the current date.



* Data Acquisition
* Dataset will be downloaded using the ‘yfinance’ module of the Python Programming Language.
* Size of the dataset will be no larger than 1 MB.
* Following snippet shows the Data Acquisition methodology:

import yfinance as yf

stock = yf.Ticker(<name>).history(<period>)

* Downloaded data will be temporarily stored in the working memory allocated to the program and then once the task is completed, it will be deallocated.
* Data Preprocessing
* The values in the dataset are comparatively larger in magnitude than that accepted by the Machine Learning model.
* Thus, the values are normalized between [0, 1] using the ‘scikit-learn’ module’s data preprocessing functionality. Following code snippet shows how it is done.

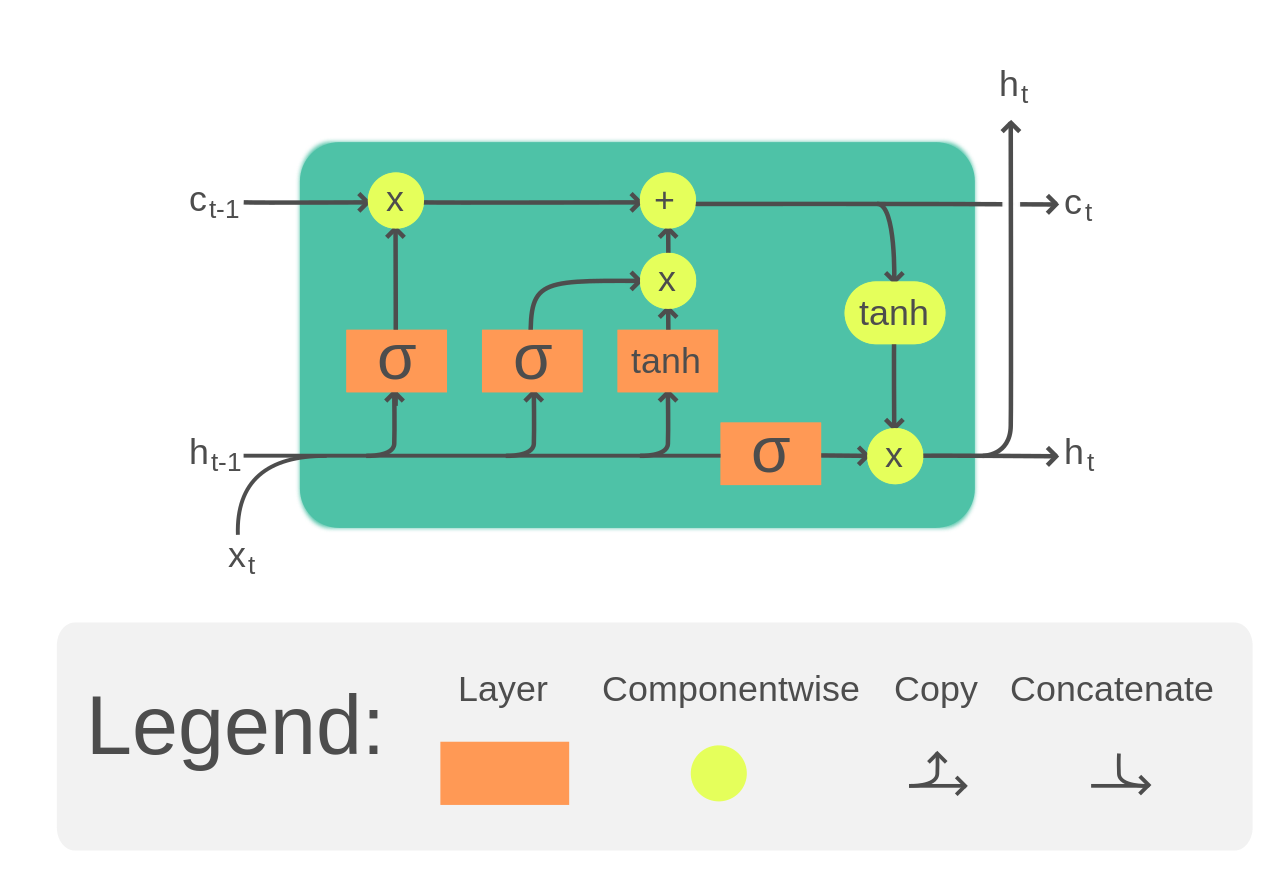
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature\_range=(0,1))

scaled\_data = scaler.fit\_transform(stock)

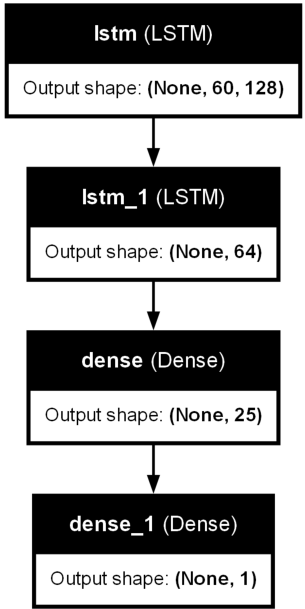


14.2 Price Prediction Module

* Aim
* Forecast the closing prices of selected stocks for a time horizon of up to two months.
* Utilize Long Short-Term Memory (LSTM) networks to model the sequential nature of stock price data.
* Build a sequential model architecture as outlined in Fig 14.2.2, detailing the layers and hyperparameters of the LSTM network.
* Specify the plotting approach used to compare actual stock prices with the model's predictions (as illustrated in Fig. 14.2.3).
* Methodology
* Future prices of selected stocks for up to 2 months will be predicted.
* Long Short-Term Memory (LSTM) will be used for prediction of the prices.
* The reason behind using LSTM in time-series prediction is that LSTMs have higher memory power than RNNs for a more extended period.

**Fig 14.2.1: Architecture of LSTM cell**

* Following image shows actual architecture of the model used for predicting the stock prices



**Fig 14.2.2: Architecture of the Actual ML Model**

* Procedure
* Following code snippet shows how the model is created and trained

model = Sequential()

model.add(LSTM(128, return\_sequences=True, input\_shape=(60,1)))

model.add(LSTM(64, return\_sequences=False))

model.add(Dense(25))

model.add(Dense(1))

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(x\_train, y\_train, batch\_size=1, epochs=1)

* Following code snippet shows how predictions are made using the trained model

predictions = model.predict(x\_test)

predictions = scaler.inverse\_transform(predictions)

* Plotting the results
* Following snippet shows how the results are plotted.

# Plot the data

train = data[:training\_data\_len]

valid = data[training\_data\_len:]

valid['Predictions'] = predictions

# Visualize the data

plt.figure(figsize=(16,6))

plt.title('Model')

plt.xlabel('Date', fontsize=18)

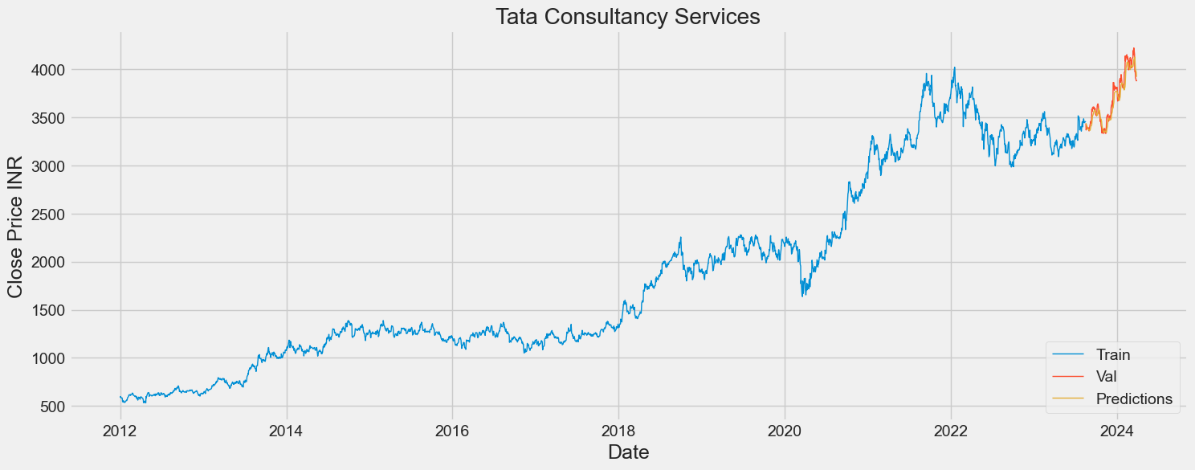
plt.ylabel('Close Price INR', fontsize=18)

plt.plot(train['Close'], linewidth=1)

plt.plot(valid[['Close', 'Predictions']], linewidth=1)

plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')

plt.show()



**Fig 14.2.3: 2-months’ price prediction for TCS**

14.3 News Feed Module

* Aim
* Aggregate news headlines from several sources.
* Provide the cleaned data to the Sentiment Analysis module.
* The current source is [marketaux.com](https://www.marketaux.com/).
* Methodology
* The module is specifically designed to gather targeted news related to stock market movements. It filters news based on stock symbols and potentially regional preferences.
* It forgoes the use of a dedicated news library like NewsApiClient and interacts directly with the MarketAux API using basic HTTP requests.
* Following code snippet demonstrates the working of this module

conn = http.client.HTTPConnection("api.marketaux.com")

params = urllib.parse.urlencode(

            {

                "api\_token": CONSTANTS.marketaux\_api\_token,

                "symbols": f"SENSEX",

                "countries": "in"

            }

        )

conn.request("GET","/v1/news/all?{}".format(params))

res = conn.getresponse()

data = res.read()

print(data.decode("utf-8"))



**Fig 14.3: News fetched from the Internet using the News Feed Module**

14.4 Sentiment Analysis Module

* Aim
* To extract relevant sentiment indicators from various textual sources, quantifying and categorizing them to determine the overall market or public opinion about a specific stock or sector.
* Develop or implement techniques to accurately detect positive, negative, and neutral sentiment within the gathered textual data.
* Consider classifying sentiment into finer-grained emotions (e.g., fear, optimism, joy) for more nuanced analysis.
* Methodology
* Combines a rule-based lexicon (the custom dictionary) with the machine-learning-trained VADER model for sentiment analysis.
* Explicitly retrieves news related to stocks, making sentiment analysis more tailored to stock price prediction.
* Provides simplified sentiment representation (positive, negative, neutral) for potential integration with a price prediction model.
* Following code snippet shows the functioning of the Sentiment Analysis Module.

def getNews(self, query):

self.news = self.News.get\_everything(query)

def preprocessText(self, text):

    tokens = word\_tokenize(text.lower())

filtered\_tokens = [token for token in tokens if token not in stopwords.words('english')]

    lemmatizer = WordNetLemmatizer()

    lemmatized\_tokens = [lemmatizer.lemmatize(token) for token in filtered\_tokens]

    processed\_text = '.join(lemmatized\_tokens)

    return processed\_text

def getSentiments(self):

self.getNews("Sensex")

self.news = self.news.filter(items=["text"])

scores = self.news["text"].apply(self.analyzer.polarity\_scores)

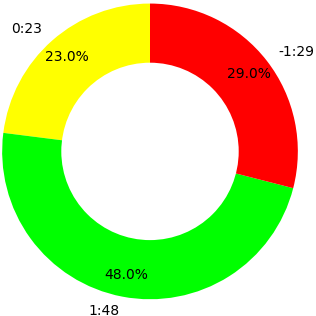
scores\_df = pd.DataFrame.from\_records(scores)

self.news = self.news.join(scores\_df)

idx = 0

self.news["compound"] = self.news["compound"].apply(lambda x: 1 if x > 0.2 else (-1 if x < -0.2 else 0))

* After the above processing, the overall sentiment is shown in the form of a Pie Chart.



**Fig 14.4: Overall market sentiment,**

**Green: Bullish; Yellow: Neutral; Red: Bearish**

**15. Test Cases**

15.1 Test Case: RELIANCE.NS

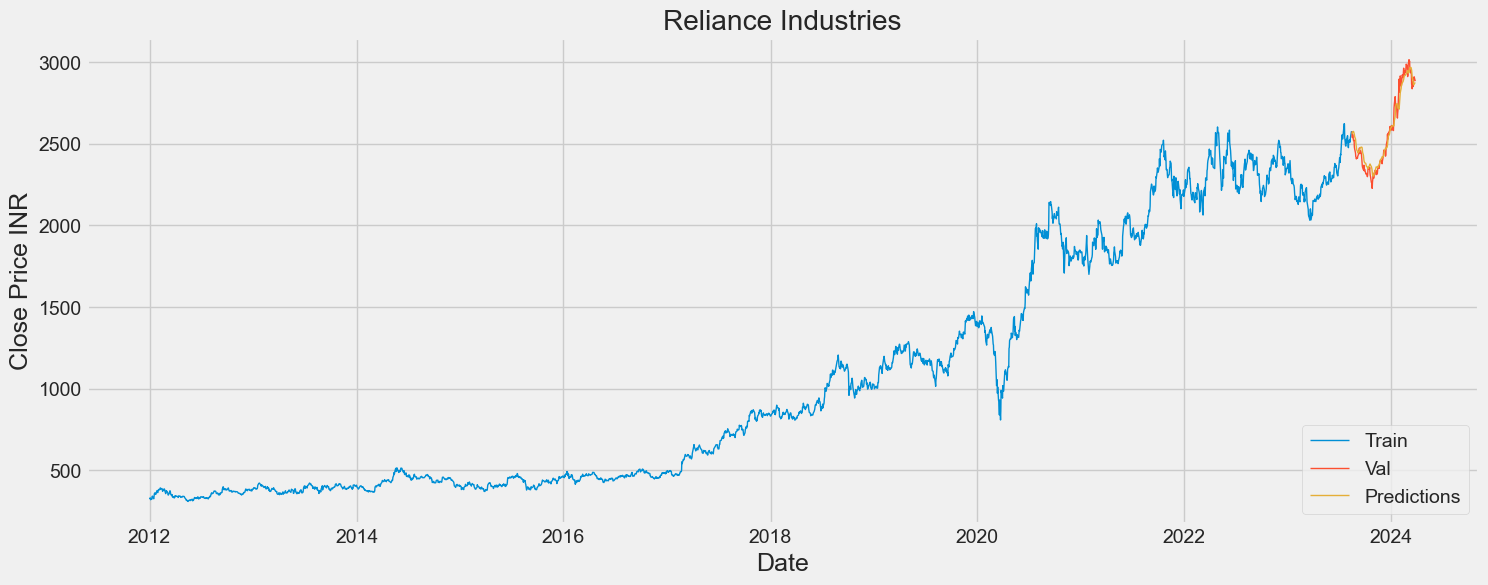
* Data Acquisition and Management
* Raw Data



* Pre-processed Data

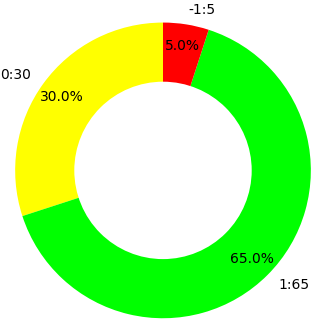


* Price Prediction Module
* 2-month’s predicted price of RELIANCE.NS



**Fig 15.1.1: Predicted price movement of RELIANCE.NS**

* News Feed Module
* News related to RELIANCE INDUSTRIES
* Sentiment Analysis Module
* Overall sentiment for RELIANCE.NS



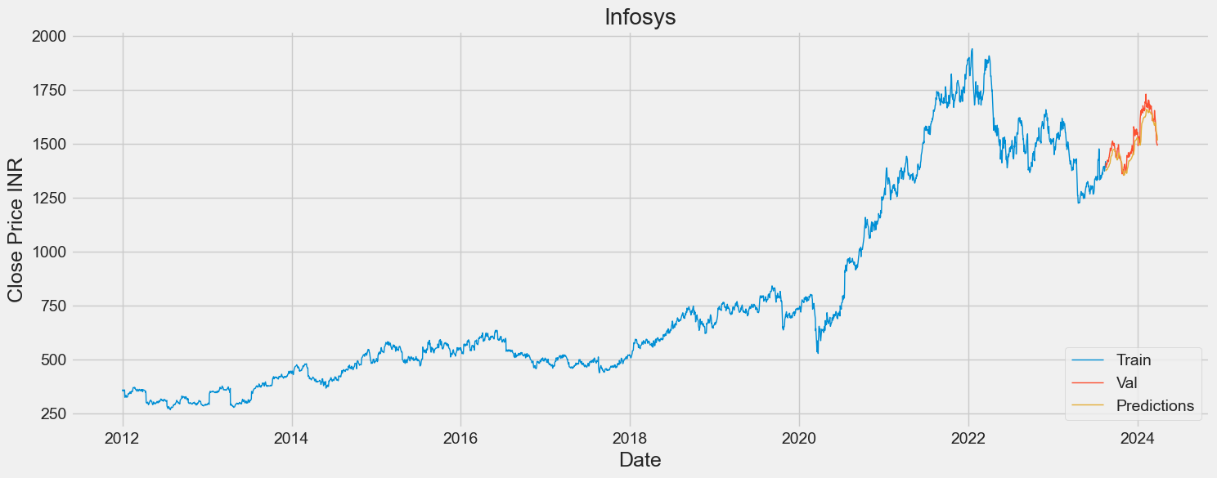
**Fig 15.1.2: Overall sentiment for RELANCE.NS**

15.2 Test Case: INFY.NS

* Data Acquisition and Management
* Raw Data



* Pre-processed Data
* Price Prediction Module
* 2 months’ predicted price of INFY.NS

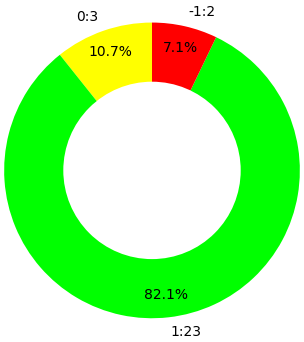


**Fig 15.2.1: Predicted price movement of INFY.NS**

* News Feed Module
* News related to INFY



* Sentiment Analysis Module
* Overall market sentiment for INFY.NS



**Fig 152.2: Overall sentiment for INFY.NS**

**Remarks/Suggestions:**

**Signature**