Predicting Stock Prices using Recurrent Neural Networks

ABSTRACT

The purpose of the project is to use Long Short-Term Memory based neural networks in order to predict the prices of different stocks of the Nifty-50 Index and to display the same in an organised manner, in order to understand the basic concepts that go into implementing a Neural Network or Machine Learning in a given application.

In order to predict, we use a concept of Recurrent Neural Network called, Long Short-Term Memory networks (LSTM). The evaluation of the same is through the analysis of errors (Mean Square Error and Root Mean Square Error) by comparing them to a traditional Statistical Time series method called Autoregressive Conditional Heteroskedasticity or ARCH.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

LSTM	Long Short-Term Memory
ARCH	Autoregressive Conditional Heteroskedasticity
GUI	Graphical User Interface
RNN	Recurrent Neural Networks
NIFTY	National Stock Exchange Fifty
NAN	Not A Number
IO	Input/Output
CSV	Comma Separated Values
HDF5	Hierarchical Data Format Version 5
API	Application Programming Interface
TPU	Tensor Processing Unit
GPU	Graphical Processing Unit
SVG	Scalable Vector Graphics
HTML	Hypertext Markup Language
DFD	Data Flow Diagram
BSE	Bombay Stock Exchange
LTP	Last Traded Price
VWAP	Volume Weighted Average Price
CPU	Central Processing Unit
MSE	Mean Squared Error
RMSE	Root Mean Squared Error

1. INTRODUCTION

Around the world, Investment firms, hedge funds and even individuals have been using financial models to better understand market behavior and, in the process, make profitable investments and trades. A large amount of data is available as historical stock prices and company performance data which is suitable for processing using machine learning algorithms. Investors make educated guesses by analyzing the data, they read the news, study the company history, look into industry trends and other kinds of factors that go into making a prediction. The prevailing theory is that stock prices are totally random and unpredictable but that raises the question as to why top financial consulting firms hire quantitative analysts to build predictive models. When we think about stock market traders, we have this idea of a trading floor being filled with adrenaline infused people running around yelling something into a phone but these days they're more likely to see rows of machine learning experts quietly sitting in front of computer screens. In fact, about 70% of all orders on Wall Street are now placed by software, we're now in the age of the algorithm.

1.1. PROJECT DESCRIPTION

The purpose of this project is to use a machine learning algorithm that can predict and forecast stock market prices by using an existing dataset to train the neural network. Predicting stock prices may be a difficult task for a human brain, but with the help of such a model, we can predict the stock prices taking various factors into account, with ease. So, with the help of existing datasets, we can predict the stock prices accurately.

1.2. EXISTING SYSTEM

In the present scenario with respect to stock trading, software tools help visualize data and give a summary of information which is then used by a person to take a call and speculate future prices. With a machine learning system in place, most

of this niche work can be automated. While in the recent years there has been a strive towards automation with most traders being replaced with algorithms, new and improved algorithms are always

on the rise. Presently many financial companies have their own forecasting tools available which gives users a general overview using machine learning or deep learning concepts like RNN i.e., Recurrent Neural Networks and LTSM i.e., Long Short-Term Memory.

1.3. OBJECTIVES

The main objective of this project is to try and incorporate an algorithm that can forecast stock prices effectively, using an LTSM based machine learning algorithm as a base and further comparing it with other approaches like a traditional time series forecasting method like ARCH [Auto Regressive Conditional Heteroskedasticity]. This forecasting is to be done on a Dataset of the different stocks of the NIFTY 50 index.

1.4. PURPOSE, SCOPE AND APPLICABILITY

1.4.1. PURPOSE

The purpose of this project is to forecast the price of stock markets shares and to obtain an accurate prediction using LSTM, and to compare it with a traditional time series forecasting.

1.4.2. SCOPE

The scope of the project is to use python to create a LSTM model which is essentially a deep learning model in order to use available data of the Nifty 50 companies, and to forecast using this data, the specific price for a certain company. And later on, compare it with a traditional Statistical forecasting.

1.4.3. APPLICABILITY

This machine learning algorithm would be applicable in the sense that it would help people interested in the ways of machine learning to gain a better understanding of how a learning model is applied into a certain domain, which in the case is Forecasting Stock prices. The applicability is limited at this level because a full-fledged application that could be built around this domain would be very complex and would require more time.

1.5. TOOL SURVEY

- Numpy

NumPy is a Python library used for working with arrays.It also has functions for working in the domain of linear algebra, fourier transform, and matrices.NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.NumPy stands for Numerical Python. The inclusion of arrays which are almost fifty times faster than traditional python lists makes the process of working with data more efficient.

- Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python.

Few of the most prominent features of the pandas toolkit are:

- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, DataFrame, etc. automatically align the data for you in computations
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data

- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets
- Intuitive merging and joining data sets
- Flexible reshaping and pivoting of data sets
- Hierarchical labeling of axes (possible to have multiple labels per tick)
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast HDF5 format
- Time series-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting, and lagging.

- Keras

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Keras is the high-level API of TensorFlow 2 which is an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity. Keras empowers engineers and researchers to take full advantage of the scalability and cross-platform capabilities of TensorFlow 2 on which you can run Keras on TPU or on large clusters of GPUs, and you can export your Keras models to run in the browser or on a mobile device.

- Plotly

Plotly is a python graphing library package that provides online graphing, analytics, and statistics tools for individuals and collaboration. It helps create interactive, publication-quality graphs. This package is free and open source enabling us to view the source, report issues and also contribute on GitHub. It is constantly updated to create a error free package and add new features overtime. This package provides over 30 chart types, including line plots, scatter

plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, multiple-axes, polar charts, bubble charts, scientific charts, 3D graphs, statistical charts, SVG maps, financial charts, and more. Plotly graphs are very flexible as their graphs can be viewed in Jupyter notebooks, standalone HTML files, or hosted online using Chart Studio Cloud.

2. SYSTEM ANALYSIS AND REQUIREMENTS

2.1. PROBLEM DEFINITION

The objective of this project is to accurately predict the future closing value of a given stock across a given period of time in the future. This project implements a Long Short Term Memory network – usually just called "LSTMs" to predict the closing price of the NIFTY50 market shares using a dataset of past prices, and to also compare the prediction with a simple time series forecasting method.

2.2. REQUIREMENTS SPECIFICATION

2.2.1. FUNCTIONAL REQUIREMENTS

For this specific use case, the main functional requirement is for the obtained LSTM model to forecast with precision and accuracy, and also a statistical ARCH prediction, for comparison.

2.2.2. NON-FUNCTIONAL REQUIREMENTS

A non-functional requirement could be a basic GUI on python which would let one switch between different available stocks and display the infographics in a more organized manner.

2.3. BLOCK DIAGRAM

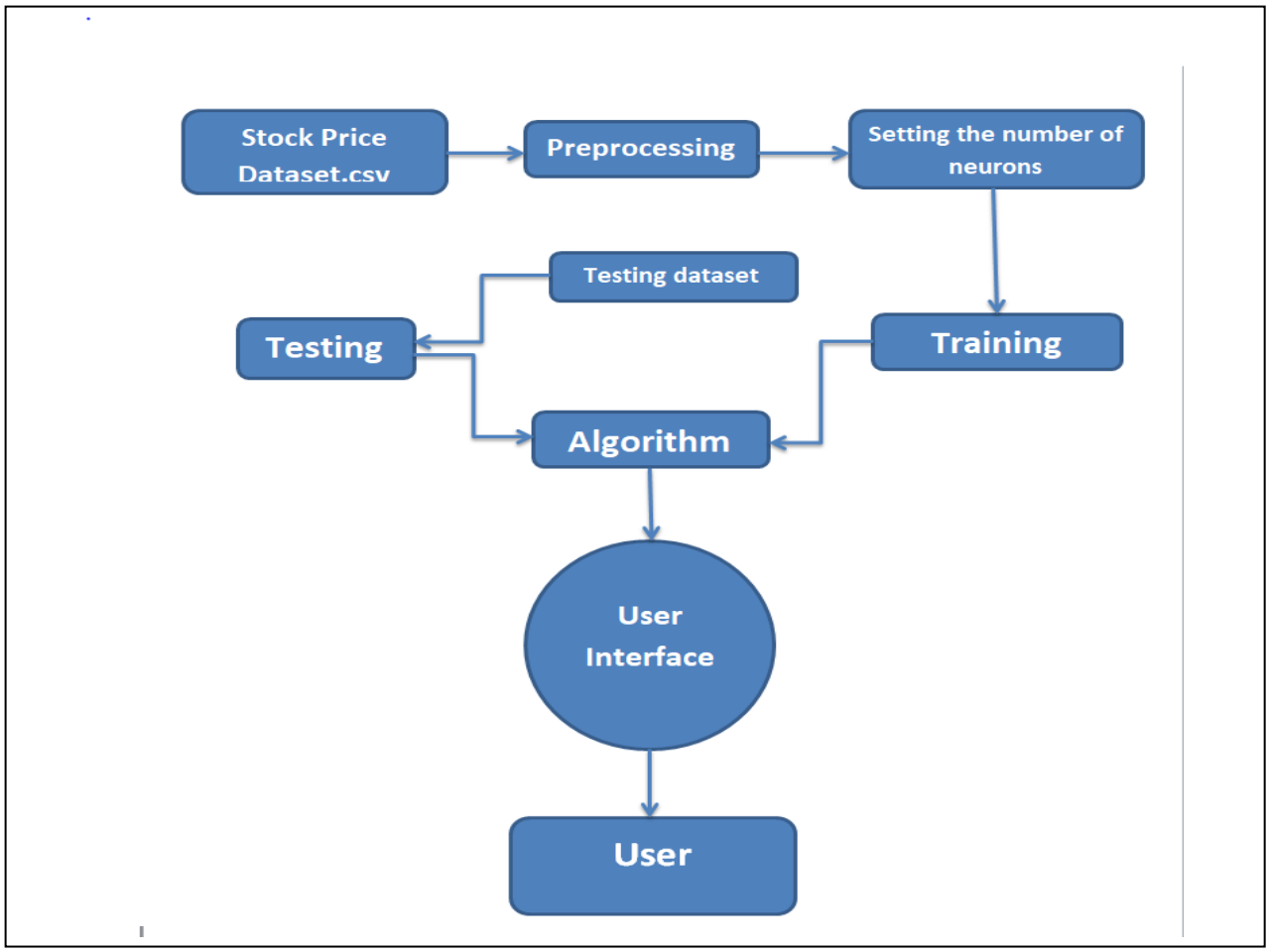

Fig 2.1 Block Diagram

2.4. SYSTEM REQUIREMENTS

2.4.1. USER CHARACTERISTICS

The requirements for a user of this tool are basically very minimal, as it just offers information in an organized manner. With that being said, anyone who understands stocks and stock markets would find the tool more insightful and useful than a person who is not familiar with these things.

2.4.2. SOFTWARE AND HARDWARE REQUIREMENTS

The main requirements for software for developing this tool is a system with python. In particular, packages like keras, scikit learn, TensorFlow,

matplotlib and pandas. The only hardware requirement for this system is any computer with python installed.

However, if the same has to be implemented on a large scale, it would require a software to work with big data analytics and a computing system with large amounts of available storage in order to facilitate analysis of that scale.

2.4.3. CONSTRAINTS

The major constraints for this system are the tight schedule and the dearth for resources, and as it stands none of the team members have any experience in the domain with this being our first project of the sort.

2.5. CONCEPTUAL MODELS

2.5.1. DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a way to visualize the flow of information within a system. A good and detailed DFD can give a huge amount of information regarding the system and the system requirements graphically. The DFD can be manual, automated or a combination of both. DFDs are built using standardized symbols and notation to describe various entities and their relationships.

Data flow diagrams are also categorized by level. It started from level 0, which is the most basic diagram and DFDs get increasingly complex as the level increases.

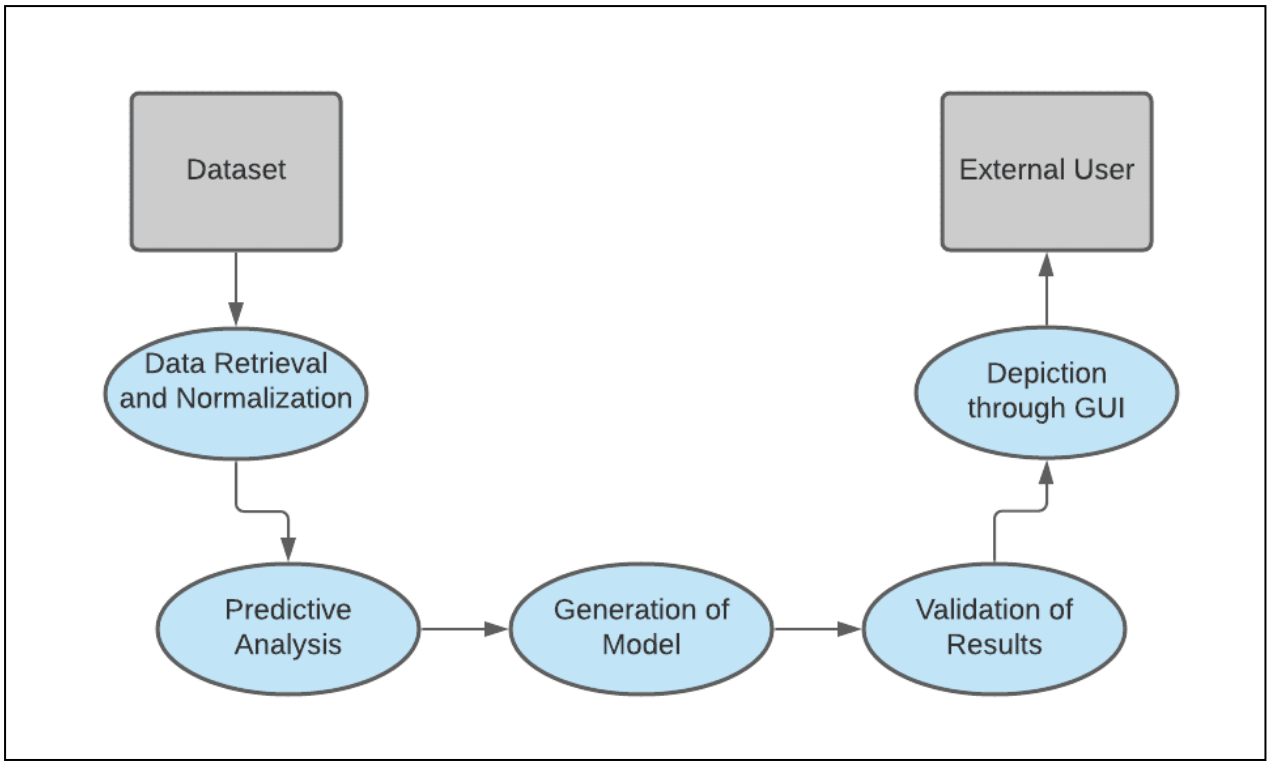

Fig 2.2 Level 0 - DFD

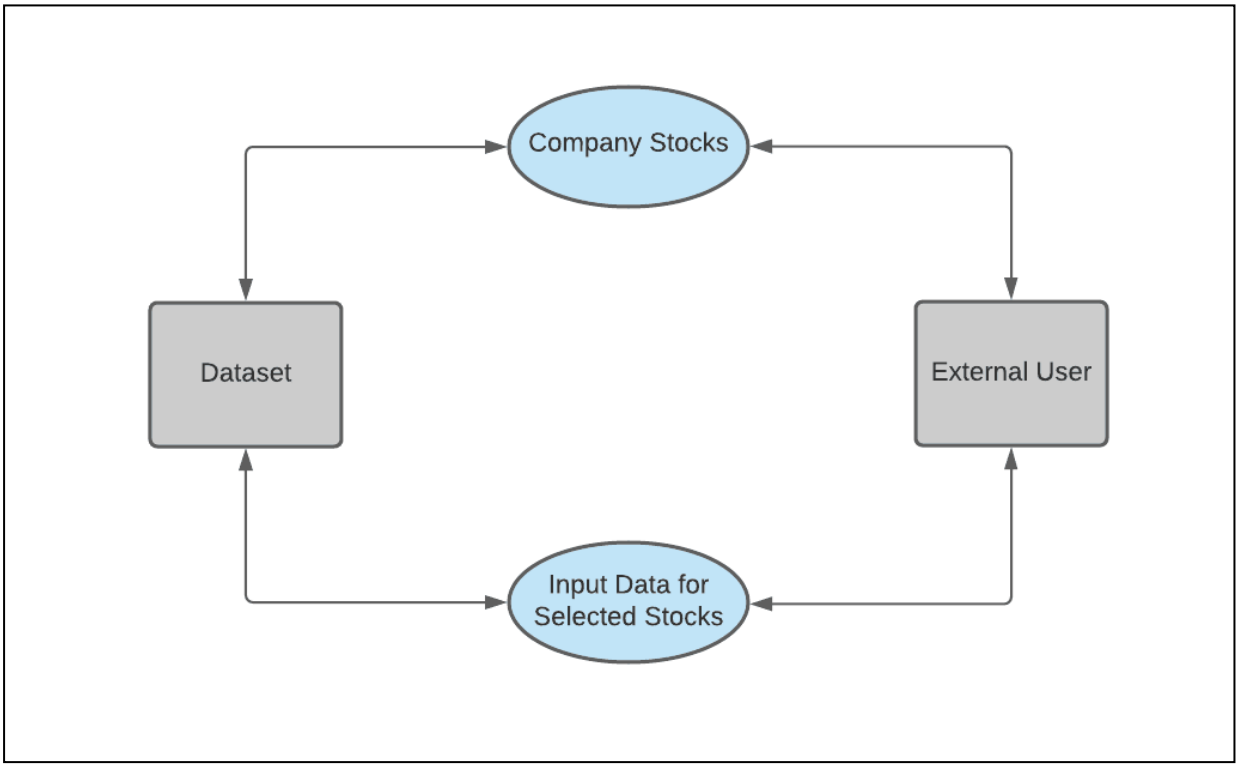

Fig 2.3 Level 1 – DFD (Data Retrieval & Transformation)

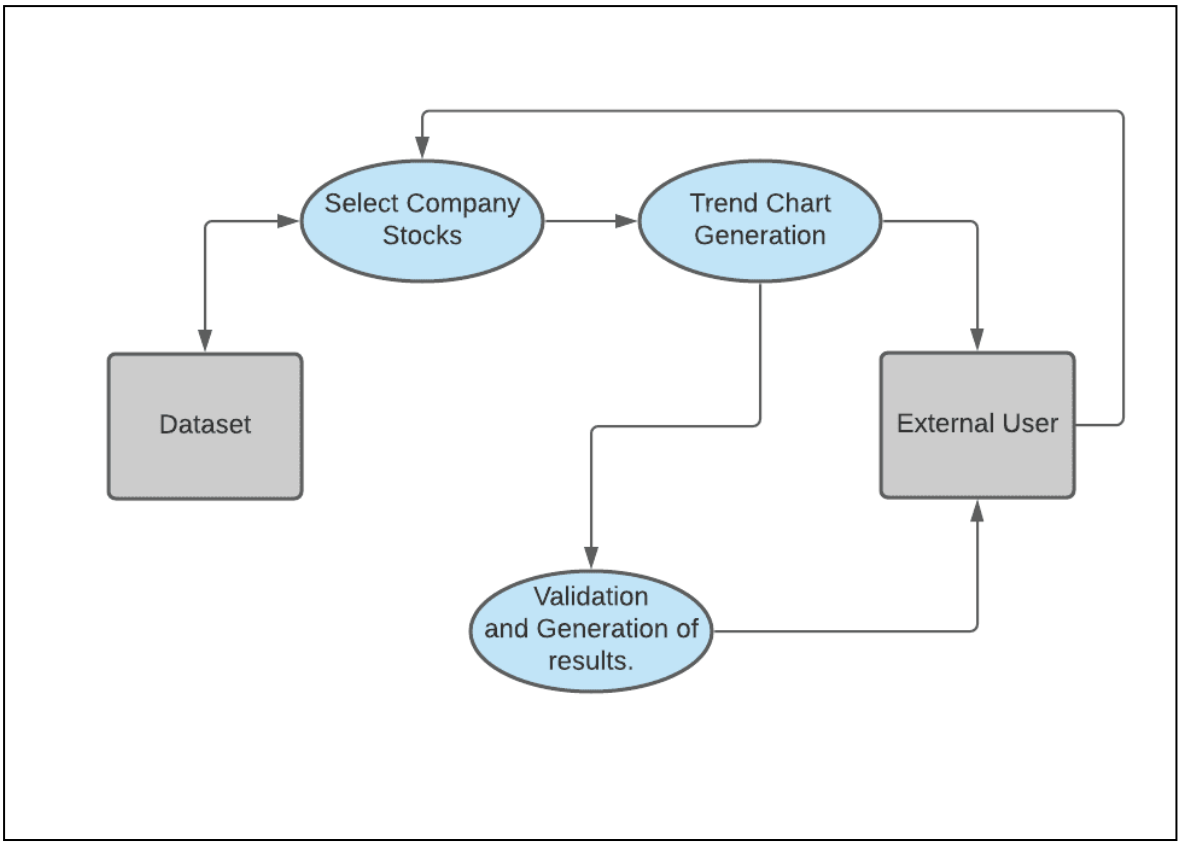

Fig 2.4 Level – 1 DFD (Technical Indicators Trend Charts Generation)

2.5.2. ENTITY RELATIONSHIP DIAGRAM

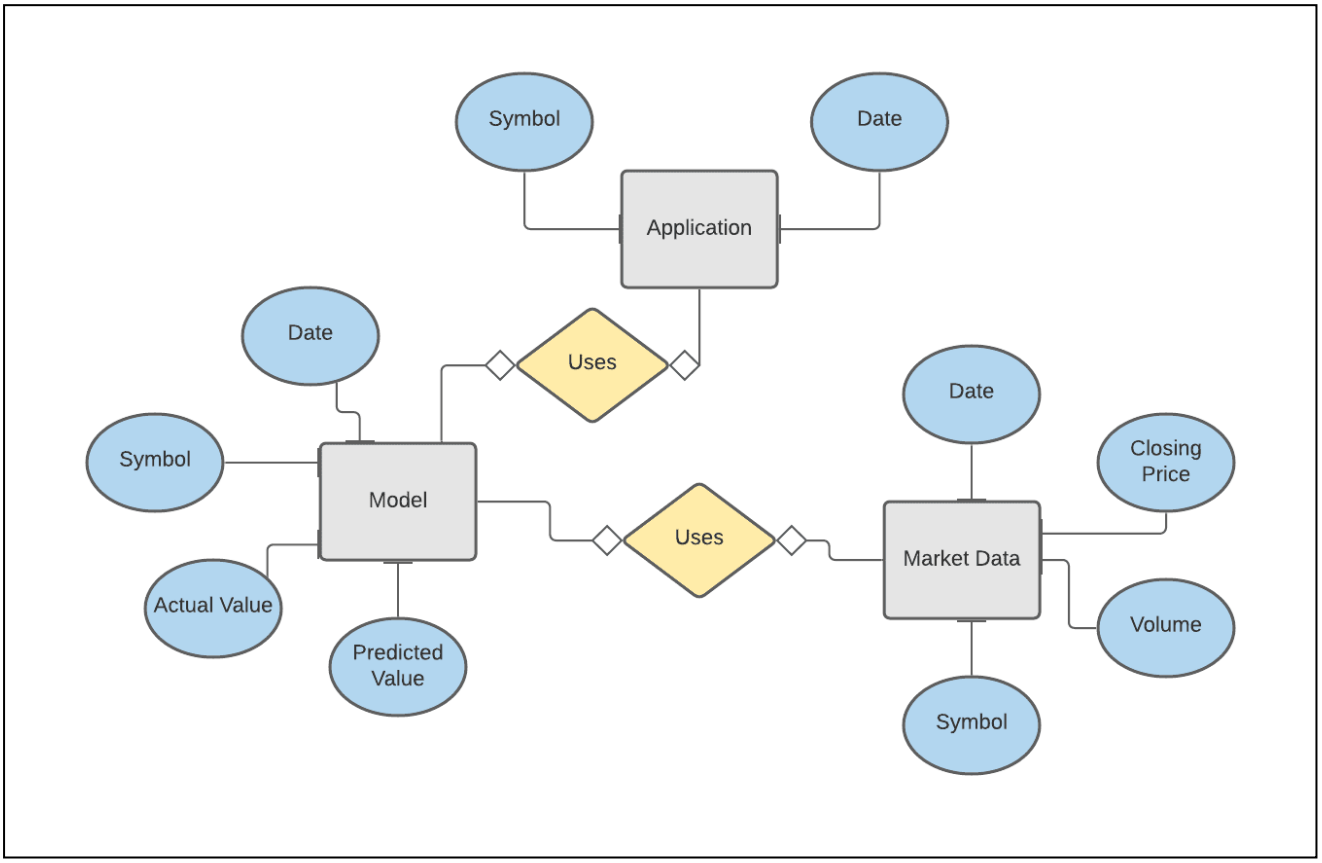

Fig 2.5 ER Diagram

3. SYSTEM DESIGN

3.1. SYSTEM ARCHITECTURE

3.1.1. USE CASE DIAGRAM

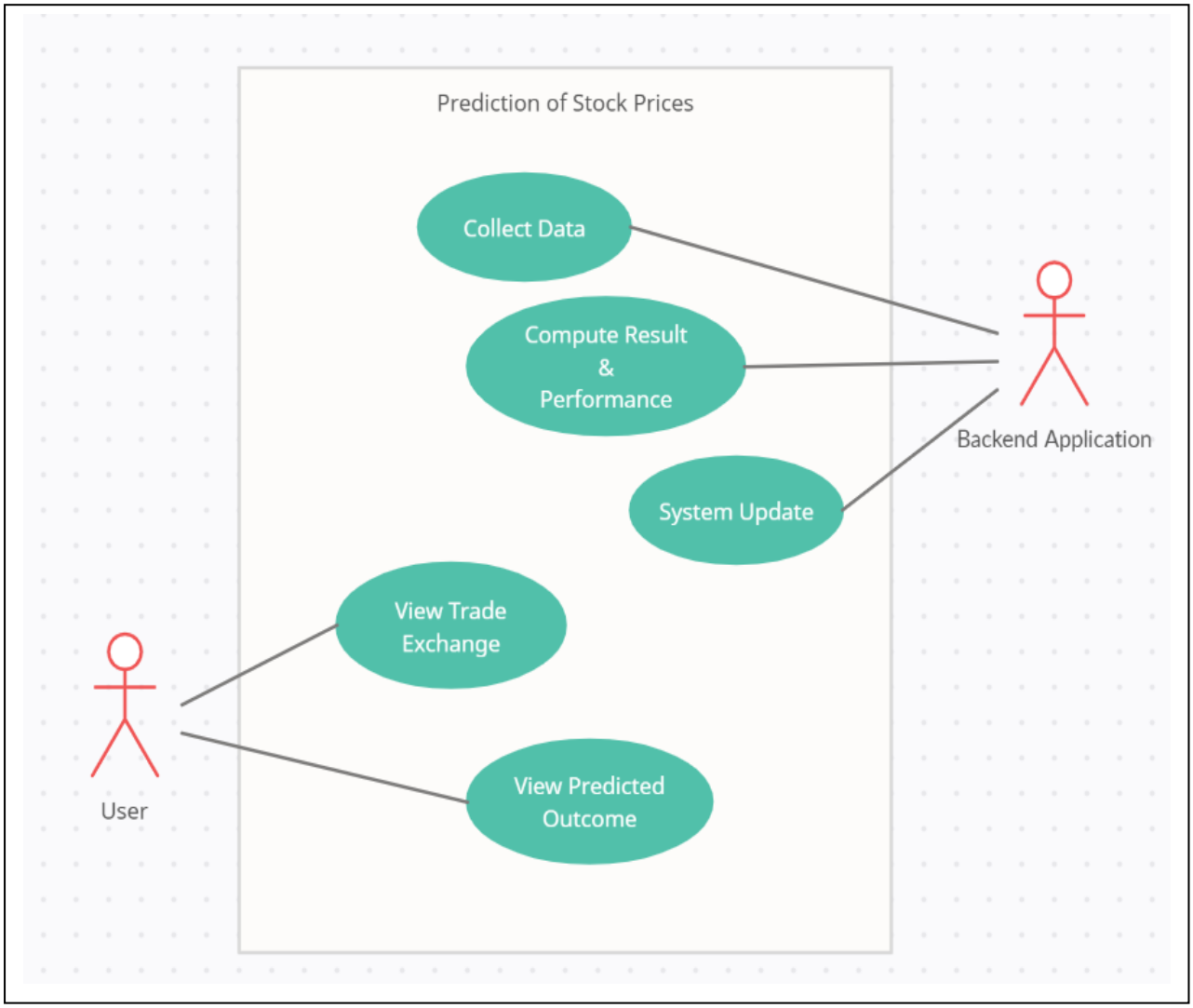

Fig 3.1 Use Case Diagram

3.1.2. USE CASE DESCRIPTION

- Use case ID: 1
  Use case name: Collect data
  Description: All required stock data of NIFTY50 will be already collected and fed into the model. The Backend Application will receive the data for the system.
- Use case ID: 2
  Use case name: Compute result and performance
  Description: Prediction result will be handled and generated by the Backend Application. The system will be built, through which the result of prediction and system performance will be analyzed.
- Use case ID: 3
  Use case name: System update
  Description: With the change of market and technology regular update of system is required. The predicted result of stock exchange and their actual price will be manually-updated by the Application Backend on a regular basis.
- Use case ID: 4
  Use case name: View traded exchange
  Description: Company trading which is held at the NIFTY50 India can be viewed by user as well as their graph.
- Use Case ID: 5
  Use Case Name: View predicted outcome
  Description: This use case is the most important in the whole project. The key feature of this project is to predict the value of NIFTY50 stocks. Thus, this will be available in the user interface and the viewer can observe them.

3.2. DATASET DESIGN

The dataset being used is the NIFTY50 dataset from 2000 to 2020. This is basically the data from the National Stock Exchange. The National Stock Exchange of India Limited (NSE) is the leading stock exchange of India, located in Mumbai. The NSE was established in 1992 as the first demutualized electronic exchange in the country. NSE was the first exchange in the country to provide a modern, fully automated screen-based electronic trading system which offered easy trading facilities to the investors spread across the length and breadth of the country. The NIFTY 50 index is National Stock Exchange of India's benchmark broad based stock market index for the Indian equity market. It represents the weighted average of 50 largest Indian company stocks in 12 sectors and is one of the two main stock indices used in India, the other being the BSE Sensex.

The different variables which are included in the dataset are:

- Date: This variable records the date of the stock market entry.
- Symbol: This variable records the name of the Ticker that is used to represent the respective stock. For example, HDFC bank is known as HDFC, Infosys Tech as INFOSYSTCH and so on.
- Previous Days Close Price: This variable represents the price at which the stock closed on the previous trading day.
- Open Price: This variable represents the price at which the stock opens on the market for that respective day.

Note – The previous day close price and the opening price need not necessarily be the same because of After Market hour orders which might affect the price.

- Highest and Lowest Price: These variables as expected record the values of highest price and lowest price at which the stock trades for the day.
- Last Traded Price: The Last traded price (LTP) usually differs from the closing price of the day. This is because the closing price of the day on NSE is the weighted

  average price of the last 30 mins of trading. Last traded price of the day is the actual last traded price.

- Close Price: Like said before the closing price is the weighted average price of the last 30 minutes of trading.
- Volume Weighted Average Price: The volume weighted average price (VWAP) is a trading benchmark used by traders that gives the average price a stock has traded at throughout the day, based on both volume and price.

3.3. SYSTEM CONFIGURATION

The whole build of this project is developed on a laptop integrated with an Intel(R) Core (TM) i5-8300H CPU @ 2.30GHz CPU with 8GB ram running on Windows 10 Home Single Language version 20H2. The software used is Jupyter Notebook and it is coded in Python 3.7.3 script which is used to create models and train them in using the inbuilt packages keras and tensorflow to predict the stock prices in the near future.

3.4. INTERFACE DESIGN AND PROCEDURAL DESIGN

3.4.1. APPLICATION FLOW/ CLASS DIAGRAM

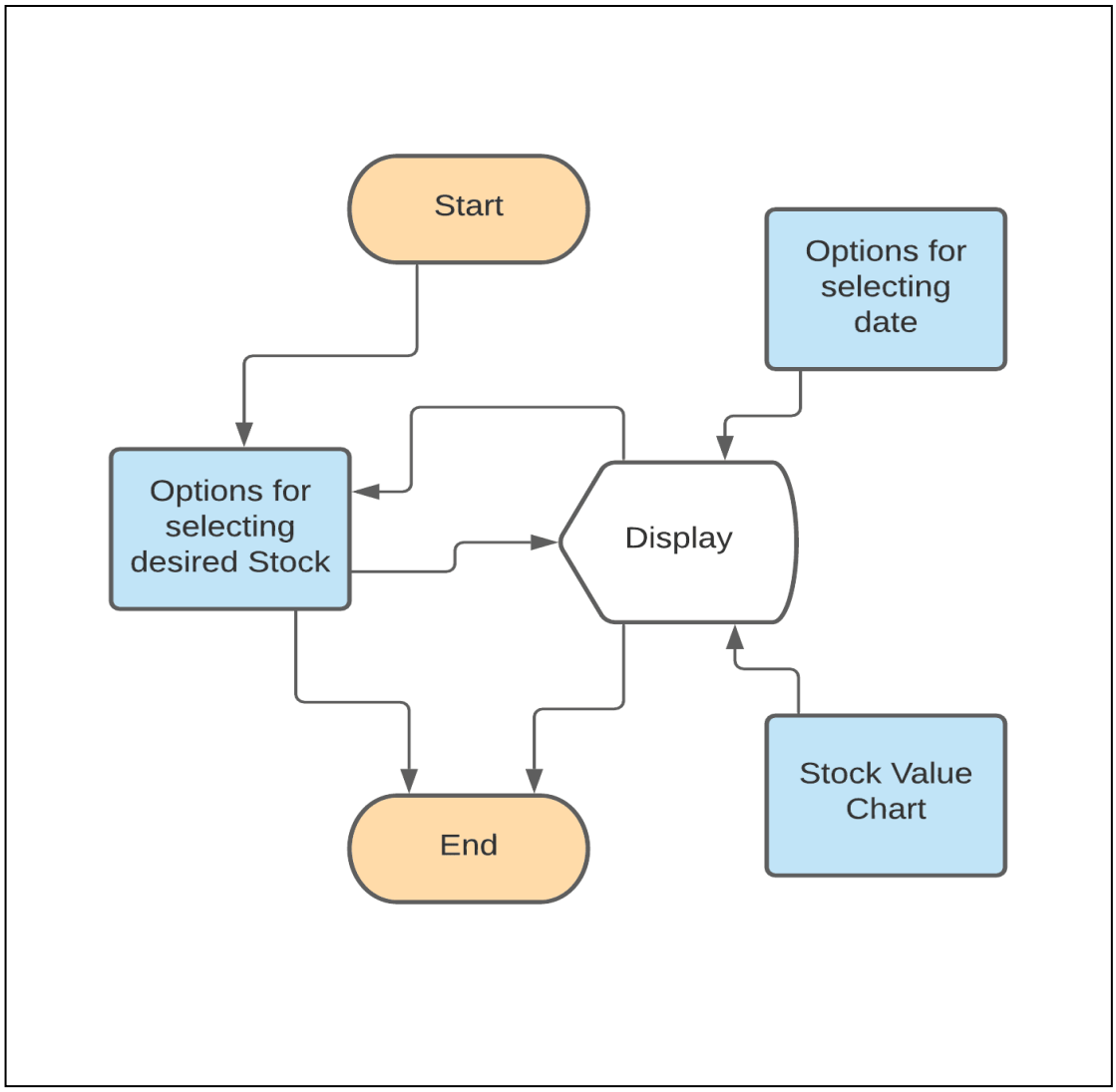

Fig 3.2 Application Flow Diagram

4. IMPLEMENTATION

4.1. IMPLEMENTATION APPROACHES

The main objective of the project is to predict the future values of stock prices. Using Neural Network and Deep Learning, a univariate model is created using Long Short-Term Memory (LSTM) and taking the Close price values from the data set. The model is then used to predict the future values for certain days. Though it is nearly impossible to predict stock prices since it depends on plenty of factors, this model predicts the future values taking only one factor hence it will not be fit using it as a reference when investing in stock prices. Another drawback of the model is that the model uses a predefined data set. Though it is possible to obtain a real time data set since the values of stock prices changes daily, it requires a lot of money and time to obtain a real time data set for stock prices.

The model predicts the future values for stock prices for 15 days. The higher the predicted number of days the higher the error. This is because the model predicts the future value by taking the last 30 data points from the obtained values. The predicted value is then added to the obtained values and then the model predicts the next values. So, to predict the 15th future value, the model uses 16 data points from the pre-existing values and 14 predicted values. Hence, due to this the error value increases. For a multivariate model, all the factors need to be predicted first in order to predict the actual stock price, hence it requires more time and the prediction accuracy will also be less due to the consideration of predicted factor values.

To give the project its substance, we implemented the codes as units which work separately. The main module has the majority of the project which includes importing the data and pre-processing the data before it can be implemented. It also consists of the main model which predicts and forecasts based on a stock selected. And then we have the ARCH model which is in its own unit, and that basically implements an ARCH model on a given stock and gets the mean square error for comparison. Then there’s a visualize unit which basically uses plotly to implement the interactive graphs used throughout the other two units.

The data however has been cleaned to remove the unwanted variables even before it is imported and this was done on Microsoft Excel 2016. Similarly, stocks which had no data till 2020 were removed from the data and not considered.

4.2. CODING STANDARDS

The guidelines provided here are intended to improve the readability of code and are followed across the project.[1]

- Indentation:

  Spaces are the preferred indentation method rather than tabs. 4 spaces are used per indentation level. Continuation lines should align wrapped elements either vertically using Python's implicit line joining inside parentheses, brackets and braces. The 4-space rule then becomes optional for continuation lines. Tabs should be used solely to remain consistent with code that is already indented with tabs. Python 3 disallows mixing the use of tabs and spaces for indentation.

  The closing brace/ bracket/ parenthesis on multiline constructs may either line up under the first non-whitespace character of the last line of any list or it may be lined up under the first character of the line that starts the multiline construct.

- Maximum Line Length

  All lines must be limited to a maximum of 79 characters. Some teams strongly prefer a longer line length. For code maintained exclusively or primarily by a team that can reach agreement on this issue, it is okay to increase the line length limit up to 99 characters, provided that comments and docstrings are still wrapped at 72 characters.

- Blank Lines

  Top-level function must be surrounded with two blank lines. Extra blank lines may be used (sparingly) to separate groups of related functions. Blank lines may be omitted between a bunch of related one-liners. Blank lines in functions can be also used to indicate logical sections.

- Imports

  Imports are always put at the top of the file, just after any module comments and docstrings, and before modules, global variables and constants. Imports should usually be on separate lines and should be grouped in the following order:

  - Standard library imports.

  - Related third party imports.
  - Local application/library specific imports.
  - You should put a blank line between each group of imports.

- Naming conventions

  Never use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names. Function names should be lowercase, with words separated by underscores as necessary to improve readability.

- String Quotes

  In Python, single-quoted strings and double-quoted strings are the same. There is no strict rule on their usage but it is recommended to pick one and follow it throughout. This project uses double-quotes to define strings in order to avoid backslashes in the string that contains single quotes. It improves readability. For triple-quoted strings, always use double quote characters to be consistent with the docstring convention.

- Whitespace in Expressions and Statements

  Avoid extraneous whitespace in the following situations:

  - Immediately inside parentheses, brackets or braces.
  - Between a trailing comma and a following close parenthesis.
  - Immediately before a comma, semicolon, or colon.
  - Immediately before the open parenthesis that starts the argument list of a function call.
  - Immediately before the open parenthesis that starts an indexing or slicing.
  - More than one space around an assignment (or other) operator to align it with another

  Around the = sign when used to indicate a keyword argument, or when used to indicate a default value for an unannotated function parameter. When combining an argument annotation with a default value, however, do use spaces around the = sign.

  To improve readability, always surround these binary operators with a single space on either side: assignment (=), augmented assignment (+=, -= etc.),

comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not), Booleans (and, or, not).

- Trailing Commas

  While trailing commas are redundant, they are often helpful when a version control system is used, when a list of values, arguments or imported items is expected to be extended over time. The pattern is to put each value (etc.) on a line by itself, always adding a trailing comma, and add the close parenthesis/bracket/brace on the next line. However, it does not make sense to have a trailing comma on the same line as the closing delimiter.

- Comments

  Comments should be complete sentences. The first word should be capitalized, unless it is an identifier that begins with a lowercase letter (never alter the case of identifiers!). Block comments generally consist of one or more paragraphs built out of complete sentences, with each sentence ending in a period.

- Documentation Strings

  Write docstrings for all public modules, functions, classes, and methods. They are not necessary for non-public methods, but you should have a comment that describes what the method does. This comment should appear after the def line.

  The """ that ends a multiline docstring should be on a line by itself. However, for one liner docstrings, please keep the closing """ on the same line.

- Other recommendations:

  Compound statements (multiple statements on the same line) are generally discouraged.

4.3. CODING DETAILS

The screenshots from the Main file are as follows [2]-[11]:

Packages

In [1]:
```
"""
Importing the required standard library modules that are required for the project.

Importing visualize.py, a user-defined module developed specifically for the project.
"""

import math
import numpy as np
import pandas as pd
import keras
import tensorflow as tf

from keras.preprocessing.sequence import TimeseriesGenerator
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout,Activation

import visualize as vs
```

Importing Data

In [2]:
```
"""
Importing the locally stored dataset.

: variable StockPrice: DataFrame with columns as ["Date","Symbol","Open","Close","Volume","VWAP"]
"""


StockPrice = pd.read_excel("STOCKPRICE.xlsx")
```

In [3]:
```
"""
List of all Stock Symbols present in dataset.
Printing the number of stock companies present.

: variable stockSymbols: A list of all stock companies present in the dataset.
"""


stockSymbols = ["ADANIPORTS", "ASIANPAINT", "AXISBANK", "BAJAJFINSV", "BAJFINANCE",
                "BPCL", "BRITANNIA", "CIPLA", "COALINDIA", "DRREDDY", "EICHERMOT", "GAIL", "GRASIM",
                "HCLTECH", "HDFC", "HDFCBANK", "HEROHONDA", "HEROMOTOCO", "HINDALCO", "HINDUNILVR",
                "ICICIBANK", "INDUSINDBK", "INFRATEL", "INFY", "IOC", "ITC", "JSWSTEEL", "KOTAKBANK",
                "LT", "M&M", "MARUTI", "NESTLEIND", "NTPC", "ONGC", "POWERGRID", "RELIANCE",
                "SBIN", "SESAGOA", "SHREECEM", "SUNPHARMA", "TATAMOTORS", "TATASTEEL", "TCS",
                "TECHM", "TITAN", "ULTRACEMCO", "UPL", "VEDL","WIPRO", "ZEEL",
               ]


len(stockSymbols)
```

Out[3]: 50

Selecting Stock

In [4]:
```
"""
Accepting user input to predict selected stock. Finding index of selected stock in stockSymbols.

: variable stockName: Stores the stock name input by the user.
: variable i: Stores the index of stockName.
"""


stockName = input("Enter the Stock Name: ")
stockName = stockName.upper()


i = stockSymbols.index(stockName)
print(i)
```

In [5]:
```
"""
Creating a pandas DataFrame of only selected stock.

: variable df: DataFrame of stockName with columns as ["Date","Symbol","Open","Close","Volume","VWAP"]
"""


df = StockPrice[StockPrice.Symbol == stockSymbols[i]]
print(df.info())
```

In [6]:
```
"""
Dropping all columns except Close and changing the index of the dataframe to store dates.

: variable df: DataFrame of stockName with columns as ["Date", "Close"] and index as dates.
"""


df["Date"] = pd.to_datetime(df["Date"])
df.set_axis(df["Date"], inplace=True)
df.drop(columns=["Open", "VWAP", "Symbol", "Volume"], inplace=True)
```

Splitting into Train and Test

In [9]:
```
"""
Splitting the selected stock dataset into training data and test data in an 80%-20% proportion.
Displaying the size of training dataset and testing dataset.

: varible close_data: A numpy array that contains only Close prices of stockName
: variable split_percent: Stores proportion of splitting the dataset into training dataset and testing dataset
: variable close_train: A numpy array that contains 80% of close_data that will be used to train the model.
: variable close_test: A numpy array that contains 20% of close_data that will be used to test the performance of the model.
: variable date_train: A pandas Series that contains 80% of Dates required for visualising training dataset
: variable date_test: A pandas Series that contains 20% of Dates required for visualising testing dataset
"""


close_data = df["Close"].values
close_data = close_data.reshape((-1,1))

split_percent = 0.80
split = int(split_percent * len(close_data))

close_train = close_data[:split]
close_test = close_data[split:]

date_train = df["Date"][:split]
date_test = df["Date"][split:]

print(len(close_train))
print(len(close_test))
```

In [10]:
```
"""
Creating a TimeSeriesGenerator for which is fed to train and test the model.

: variable look_back: Stores the number of previous days' data to use, to predict the value for the next day.
: variable train_generator: A time series sample with input and output components as training dataset.
: variable test_generator: A time series sample with input and output components as testing dataset.
"""


look_back = 30

train_generator = TimeseriesGenerator(close_train, close_train, length=look_back, batch_size = 20)
test_generator = TimeseriesGenerator(close_test, close_test, length=look_back, batch_size = 1)
```

Building the LSTM Model

In [11]:
```
"""
Building the model architecture with 2 layers
(1 LSTM layer with 10 neurons & linear activation function, 1 Dense Layer with 1 neuron).
Compile defines the loss function, the optimizer and the metrics. A compiled model is needed to train because training
uses the loss function and the optimizer.

: variable num_epochs: Stores the number of epochs the model performs.
: variable fittModel: Contains the trained model.
"""


model = Sequential()

model.add(
    LSTM(10,
        activation="linear", input_shape = (look_back, 1))
)

model.add(Dense(1))


model.compile(optimizer="adam", loss="mse")


num_epochs = 10


fittModel = model.fit_generator(train_generator, epochs = num_epochs, verbose = 2)
```

EPOCHS VS LOSS

In [13]:
```
"""
Visualising Epochs vs Loss by using a user-defined function in visualize.py

: variable loss_train: A list that stores the loss of each epoch when training the model.
: variable epochs: A list that stores the number of epochs.
"""


loss_train = fittModel.history["loss"]
epochs = list(range(1, num_epochs+1))

vs.plot_epochs_loss(epochs, loss_train)
```

Model Evaluation:

In [14]:
```
"""
Evaluating the mean squared error and root mean squared error of the model during training and testing.

: variable trainScore: Stores the value of mean squared error of the model when trained.
: variable testScore: Stores the value of mean squared error of the model when tested.
"""


trainScore = model.evaluate(train_generator, verbose = 2)
print("Train Score: %.8f MSE (%.8f RMSE)" % (trainScore, math.sqrt(trainScore)))

print("\n")

testScore = model.evaluate(test_generator, verbose = 2)
print("Test Score: %.8f MSE (%.8f RMSE)" % (testScore, math.sqrt(testScore)))
```

```
87/87 - 1s - loss: 1185.5872
Train Score: 1185.58715820 MSE (34.43235627 RMSE)


410/410 - 1s - loss: 2164.2971
Test Score: 2164.29711914 MSE (46.52200683 RMSE)
```

The screenshots from the ARCH model are as follows [12]:

In [10]:
```
size = int(len(df)*0.8)
stock=df.Close.pct_change().dropna()
train,test = stock[:size],stock[size:]
```

In [11]:
```
model = arch_model(train, p=25,q=0)
```

In [12]:
```
model_fit = model.fit()
```

In [14]:
```
# forecast the test set
yhat = model_fit.forecast(horizon=len(test))
```

In [17]:
```
rolling_predictions = []
test_size = len(test)

for i in range(test_size):
    train = stock[:-(test_size-i)]
    model = arch_model(train, p=25, q=0)
    model_fit = model.fit(disp='off')
    pred = model_fit.forecast(horizon=1)
    rolling_predictions.append(np.sqrt(pred.variance.values[-1,:][0]))
```

In [18]:
```
rolling_predictions = pd.Series(rolling_predictions, index=stock.index[-test_size:])
```

In [19]:
```
plt.figure(figsize=(10,4))
true, = plt.plot(stock[-test_size:])
preds, = plt.plot(rolling_predictions)
plt.title('Volatility Prediction - Rolling Forecast', fontsize=20)
plt.legend(['True Returns', 'Predicted Volatility'], fontsize=16)
```

The screenshots from the visualize.py file are as follows:

```
""" Importing plotly module for interactive visualisation of the Closing Prices of Stock Shares. """

import plotly.graph_objects as go


def plot_basic(stocks, title):

    """
    Plots basic plotly plot.

    :param stocks: DataFrame having all the necessary data.
    :param title: Title of the plot.
    """

    trace1 = go.Scatter(
        x = stocks.index,
        y = stocks.Close,
        mode = "lines",
        name = "Data",
        )
    layout = go.Layout(
        title = title,
        xaxis = {"title": "Date"},
        yaxis = {"title": "Close"},
        )
    fig = go.Figure(data = [trace1], layout = layout)
    fig.show()
```

```
def plot_epochs_loss(epochs, loss_train):

    """
    Prints a plotly plot for the Epochs vs Loss.

    :param epochs: List of the number of epochs.
    :param loss_train: List of the respective training loss.
    """

    trace1 = go.Scatter(
        x = epochs,
        y = loss_train,
        mode = "lines",
        name = "Data",
        )
    layout = go.Layout(
        title = "Epochs vs Training Loss",
        xaxis = {"title": "Epochs"},
        yaxis = {"title": "Training Loss"},
        )
    fig = go.Figure(data = [trace1], layout = layout)
    fig.show()
```

```
def plot_lstm_prediction(close_train, date_train, close_test, date_test, prediction, title):

    """
    Prints a plotly plot for train, test and prediction against date.

    :param close_train: numpy array of training dataset.
    :param date_train: pandas Series that contains the respective dates for data points in close_train.
    :param close_test: numpy array of testing dataset.
    :param date_test: pandas Series that contains the respective dates for data points in close_test.
    :param prediction: numpy array of predicted values by the model.
    :param title: Title of the plot.
    """

    trace1 = go.Scatter(
        x = date_train,
        y = close_train,
        mode = "lines",
        name = "Data",
        )
    trace2 = go.Scatter(
        x = date_test,
        y = prediction,
        mode = "lines",
        name = "Prediction",
        )
    trace3 = go.Scatter(
        x = date_test,
        y = close_test,
        mode ="lines",
        name = "Ground Truth",
        )
    layout = go.Layout(
        title = title,
        xaxis = {"title": "Date"},
        yaxis = {"title": "Close"},
        )
    fig = go.Figure(data = [trace1, trace2, trace3], layout = layout)
    fig.show()
```

```
def plot_lstm_forecasting(close_train, date_train, close_test, date_test, prediction, forecast_dates, forecast, title):
    """
    Prints a plotly plot for train, test, prediction and forecasting  against date.

    :param close_train: numpy array of training dataset.
    :param date_train: pandas Series that contains the respective dates for data points in close_train.
    :param close_test: numpy array of testing dataset.
    :param date_test: pandas Series that contains the respective dates for data points in close_test.
    :param prediction: numpy array of predicted values by the model.
    :param forecast: numpy array of forecasted prices by the model.
    :param forecast_dates: list of the respective future dates for data points in forecast.
    :param title: Title of the plot.
    """
    trace1 = go.Scatter(
        x = date_train,
        y = close_train,
        mode = "lines",
        name = "Data",
        )
    trace2 = go.Scatter(
        x = date_test,
        y = prediction,
        mode = "lines",
        name = "Prediction",
        )
    trace3 = go.Scatter(
        x = date_test,
        y = close_test,
        mode = "lines",
        name = "Ground Truth",
        )

    trace4 = go.Scatter(
        x = forecast_dates,
        y = forecast,
        mode ="lines",
        name = "Future",
        )
    layout = go.Layout(
        title = title,
        xaxis = {"title": "Date"},
        yaxis = {"title": "Close"},
        )
    fig = go.Figure(data = [trace1, trace2, trace3, trace4], layout = layout)
    fig.show()
```

4.4. SCREENSHOTS

The following screenshots show the different outcomes of the predictions when the model is trained and used on different stocks. Each stock is represented by two images, one showing the entire model predicted and forecasted, the other showing the forecast in particular. In the figures, the blue part of the graph represents the data used for training. The green part depicts the actual value of the stock which is split for testing. The red part depicts the values predicted by the model and the Purple represents future forecasted values.

These are the obtained output graphs when different stocks are selected.

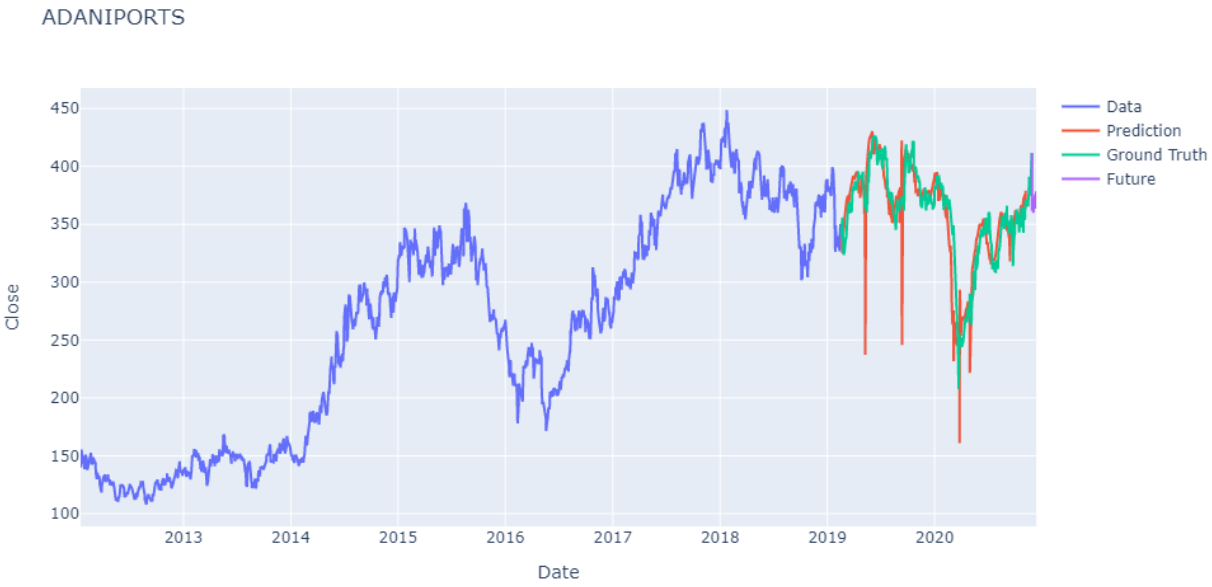

Fig 4.1 Adaniports Prediction Full

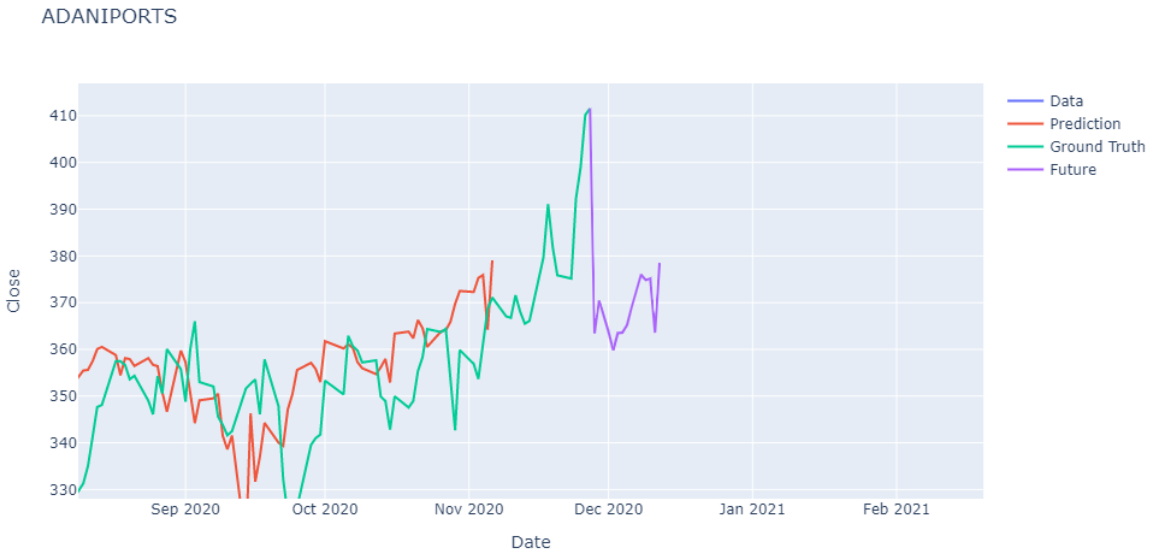

Fig 4.2 Adaniports Forecast

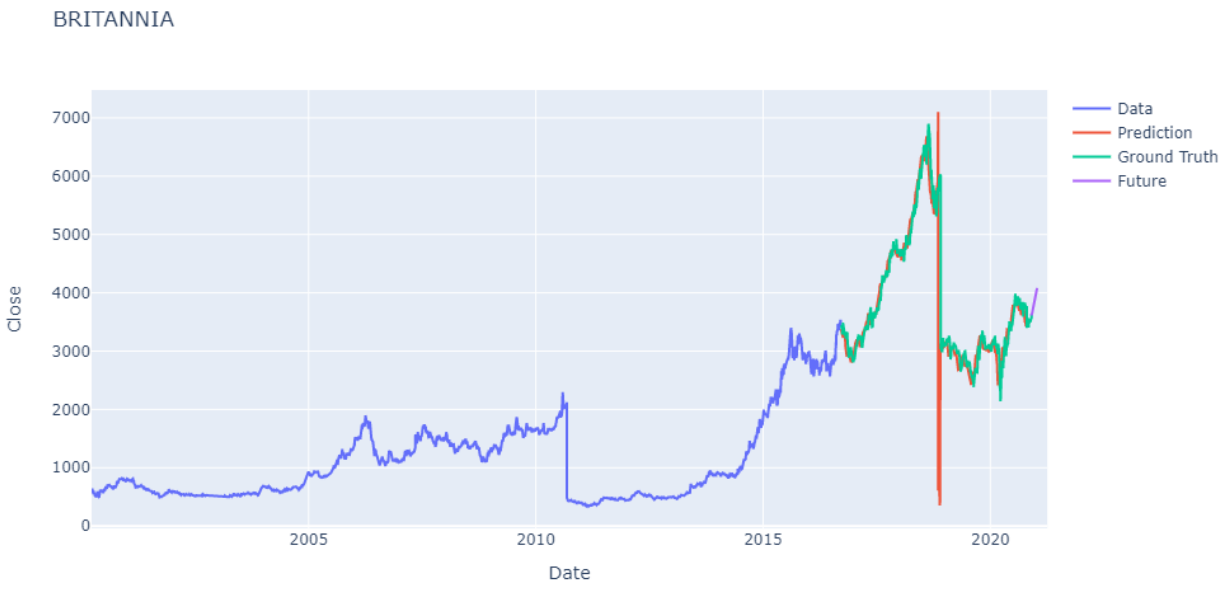

Fig 4.3 Britannia Prediction Full

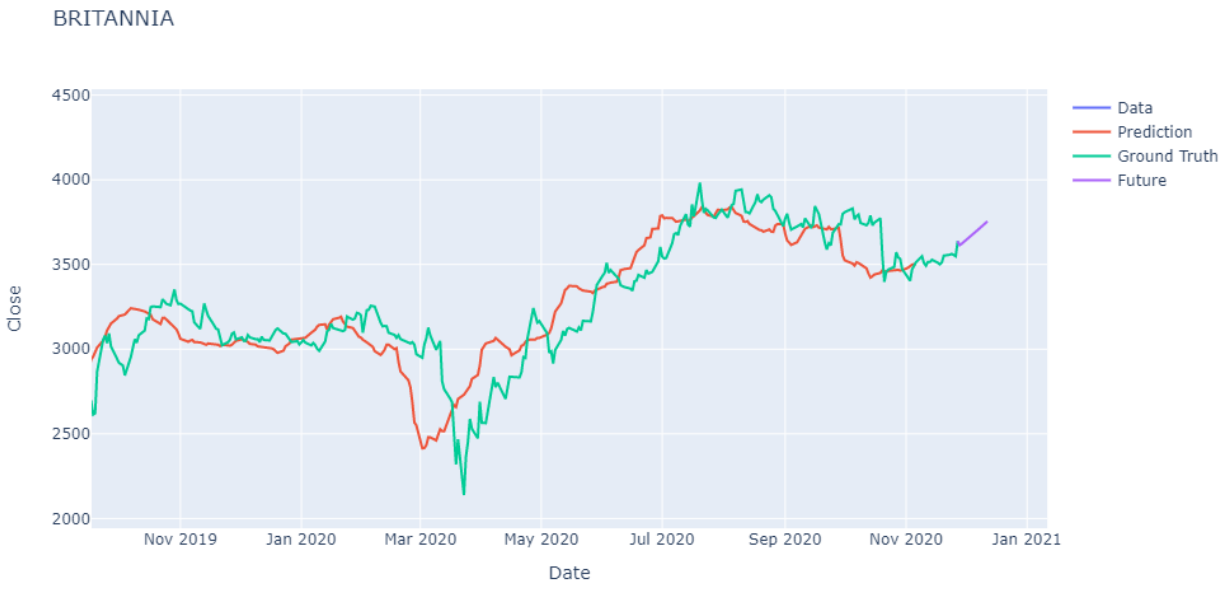

Fig 4.4 Britannia Forecast

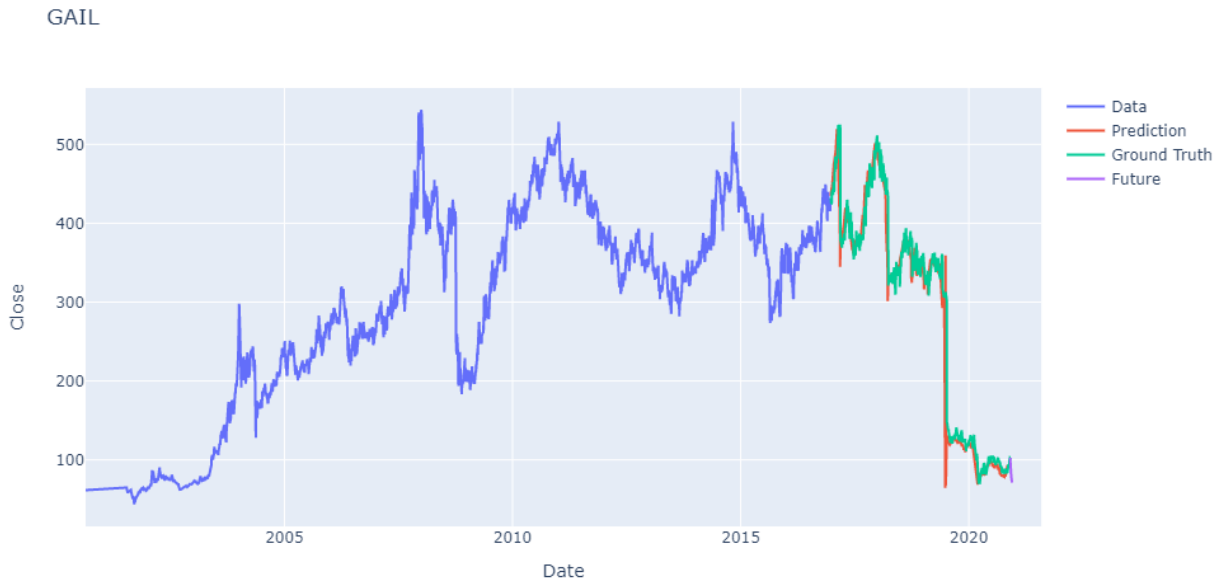

Fig 4.5 GAIL Prediction Full

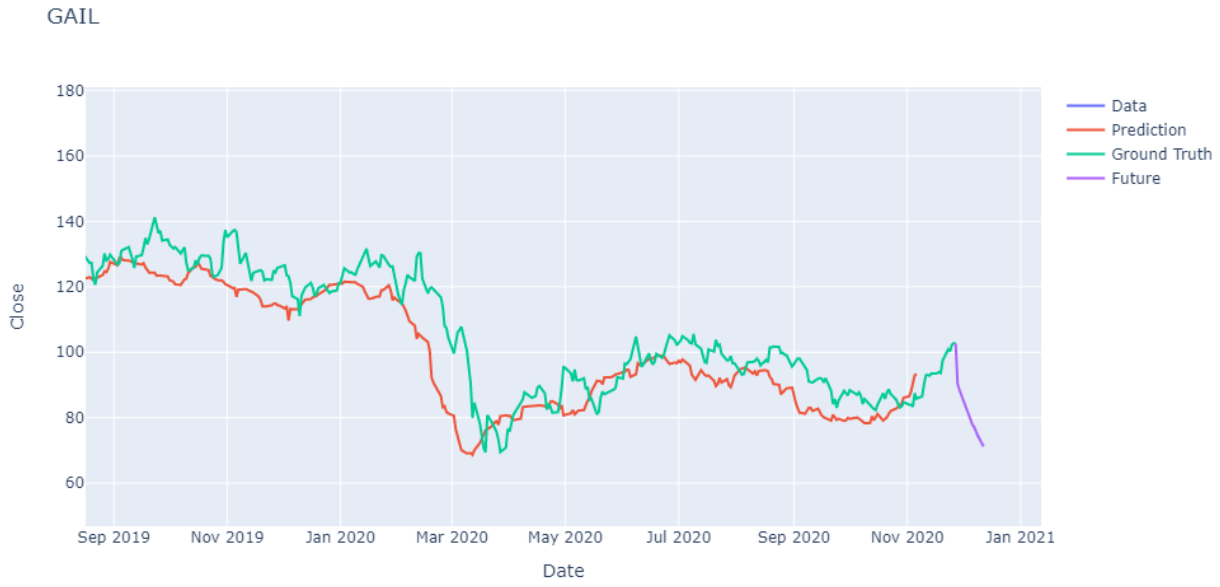

Fig 4.6 GAIL Forecast

Fig 4.7 Nestle Prediction Full

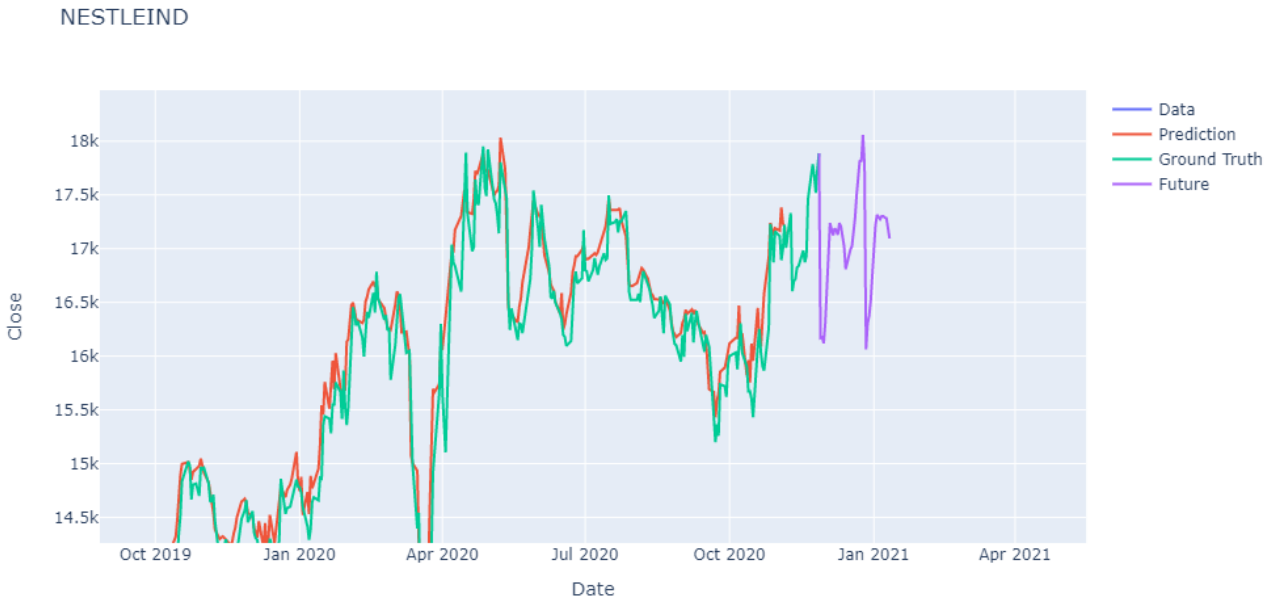

Fig 4.8 Nestle Forecast

Fig 4.9 Titan Prediction Full

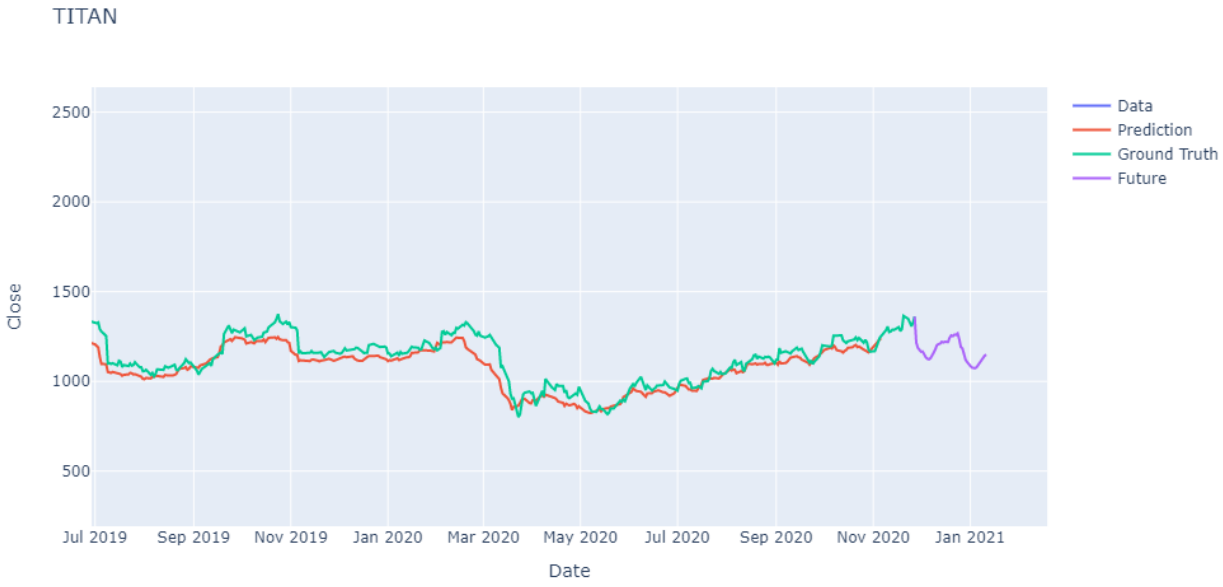

Fig 4.10 Titan Forecast

5. TESTING

5.1. TEST CASES

The main case to test for in this project is the working of the prediction and forecasting modules. And in doing so, the code generated one of the following results:

Case 1:

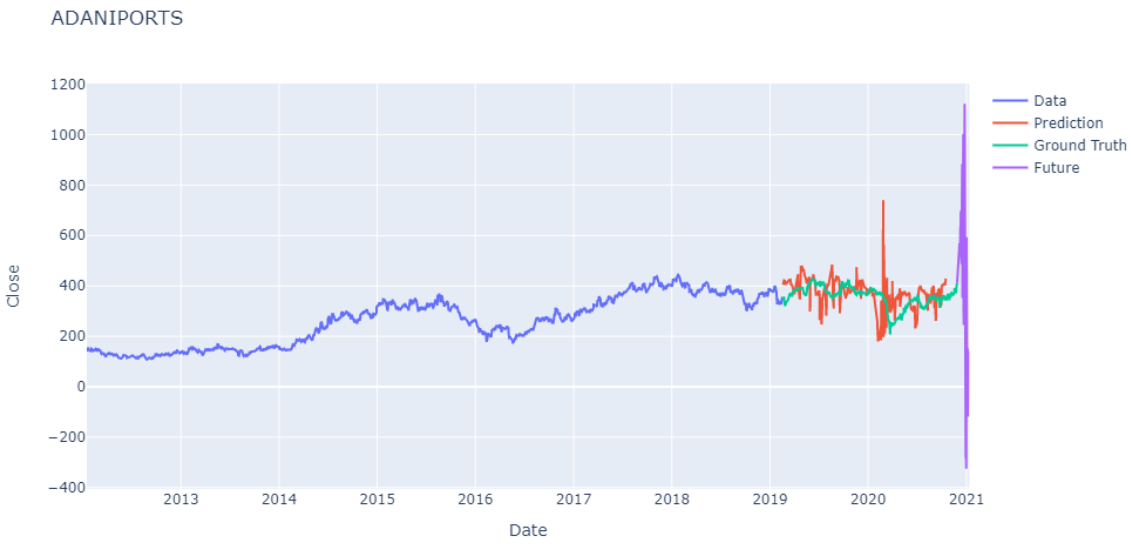

Fig 5.1 Bad Prediction

The predictions are erratic and the forecast follows a similar pattern.

Case 2:

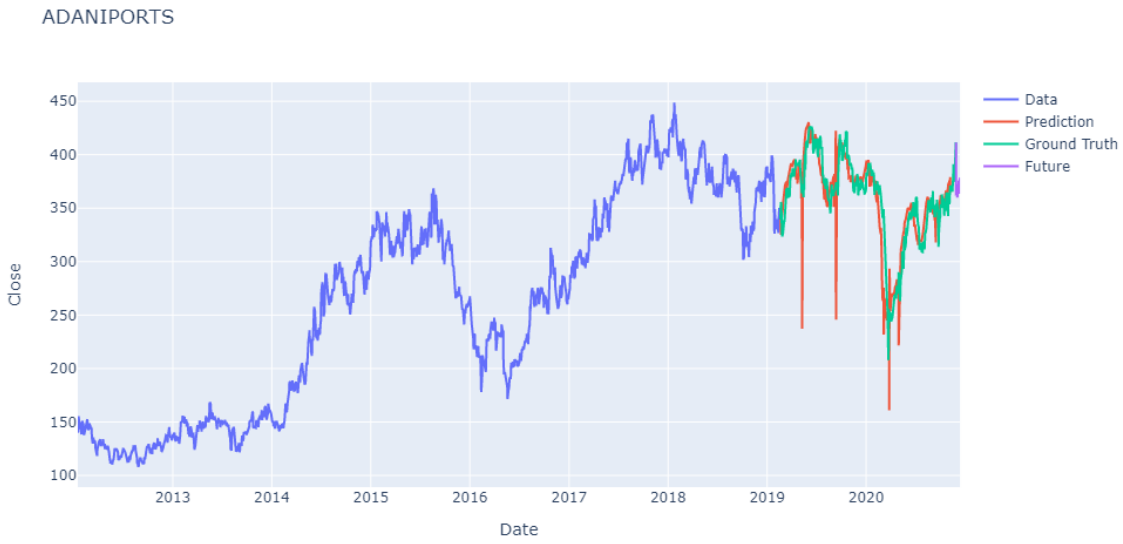

Fig 5.2 Good Prediction

Both the prediction and the forecast are stable and stay true to the actual parameters.

As it is observed in the images above, the fig 5.2. performs better and any model should follow along the same lines for it to be accurate. In other words, this would be the desirable model of the two.

5.2. TESTING APPROACHES

The main approach used to test in the case of this project is Unit testing as we just have two units that have to be tested individually, which are the main LSTM part of the project and the comparative ARCH model. There is no scope for integration testing as we decided not to go ahead with the user interface.

The main method used for comparing the model is by comparison of the MSE and RMSE, which are the Mean Square Error and the Root Mean square error respectively. The comparison takes place between the LSTM and the ARCH models as the ARCH model is often used as an alternative method to obtain information about the stock

market. And by comparing the LSTM against it, we can get a better understanding of which can be more accurate.

5.3. TEST REPORTS

ARCH

MSE and RMSE

```
In [20]: mse=np.mean((rolling_predictions-stock)**2)
         print(mse)
         rmse=mse**0.5
         print(rmse)

         10.90490549625318
         3.302257636262377
```

Fig 5.3 MSE and RMSE of ARCH Model

LSTM

MSE and RMSE

```
In [30]: """
         Evaluating the mean squared error and root mean squared error of the model during training and testing.

         : variable trainScore: Stores the value of mean squared error of the model when trained.
         : variable testScore: Stores the value of mean squared error of the model when tested.
         """


         trainScore = model.evaluate(train_generator, verbose = 2)
         print("Train Score: %.8f MSE (%.8f RMSE)" % (trainScore, math.sqrt(trainScore)))

         print("\n")

         testScore = model.evaluate(test_generator, verbose = 2)
         print("Test Score: %.8f MSE (%.8f RMSE)" % (testScore, math.sqrt(testScore)))

         87/87 - 0s - loss: 56.4943
         Train Score: 56.49431229 MSE (7.51626984 RMSE)


         410/410 - 1s - loss: 117.0277
         Test Score: 117.02768707 MSE (10.81793359 RMSE)
```

Fig 5.4 MSE and RMSE of LSTM Model

The above images find out the MSE and RMSE of just the Adaniports stock. This was done as training ARCH modes takes a substantial amount of time and different data would have different parameters and consequently different model architectures. There is no sure way of automating the same. And so we decided to stick to just one stock.

From the above figures, we can see that the ARCH model has a better score for prediction (RMSE-3.302) as compared to the LSTM model (RMSE-10.8179) but then in the ARCH model we lose out on the more accurate predictions of trend and prediction into the future. Furthermore, the LSTM model used is a very basic one with few neurons and still the RMSEs of these models are somewhat comparable. Given the proper computing power, we can make LSTMs with higher numbers of neurons and better optimize them to obtain results which would be far better than the ARCH model.

6. CONCLUSION

6.1. DESIGN AND IMPLEMENTATION ISSUES

One of the main issues faced while implementing the project was getting consistent results for prediction and forecasting using the neural network. This is essentially because of how neural networks work to optimize their weights and each time the model is run, different weights get assigned leading to different results. The other issue faced was the optimization of the neural network itself as there is no sure way of optimizing it. Most of the sources referred to state that the optimization is a hit or miss. So, a large number of trials were required for determining the four different parameters which are number of neurons, epochs, lookback values and batch sizes.

6.2. ADVANTAGES AND LIMITATIONS

The main advantage of this project is that all of our results are represented in a graphical manner which makes it easy to understand. The project also includes forecasting into the future which was not found in many other referred projects.

The limitations on the other hand, is mainly the fact that neural networks require lots of computational power. If more machine learning or deep learning-oriented processing power like Tensor Processing Units were available, more experiments could be conducted with respect to fine-tuning the hyperparameters, with fairly larger neural networks. The present work is limited to tuning a basic LSTM architecture. Other types of LSTM models can be similarly tuned.

6.3. FUTURE SCOPE OF THE PROJECT

The project makes use of an LSTM model on stock data that is a downloaded copy of the data from the past. In order to improve, one could integrate the project with an API that gets the real time data off the internet, for more valid and useful predictions. The project also makes use of a model that is univariate (close price), and a model that works on multivariate data could be built. Further, a sentiment analysis module could be built to analyse the sentiment of people on different social media platforms like Twitter or Reddit and this could be introduced as a variable into the LSTM forecasting module in order to get more accurate, real-time forecasts.

APPENDICES

If anyone wants to make use of the code used for this project found in the GitHub repository, firstly make sure to install the required packages used in this particular code like plotly, keras, statsmodels and arch.

The majority of the LSTM is included in the file called Stock Price Forecasting.ipynb file. To choose a particular stock from the included stocks in the dataset, type in the name of the stock from the given list in the Selecting Stock section.

The LSTM model is under the “Building the LSTM Model”. The different parameters like the number of neurons, epochs, batch size and look back values can be experimented within the model. We have used those particular parameters as they worked the best for us. There is no sure way of getting these parameters right except by trial and error.

The ARCH model which is used as a comparison can be found in the ARCH file. Further, to organize the information better and help improve understanding a UI can be developed, possibly based on a html page, into which interactive Plotly graphs can be integrated.

All the required files are available in the following GitHub repository: github.com/RYXNE/Prediction-of-Stock-Prices-Using-LSTM

REFERENCES

[1] Rossum,Guido;Warsaw,Barry;Coghlan,Nick,"PEP 8 -- Style Guide for Python Code" ,5 Jul. 2001.
< https://www.python.org/dev/peps/pep-0008/#code-lay-out >

[2] Yadav,Anita;C.K.Jha;Sharan,Aditi,"Optimizing LSTM for time series prediction in Indian stock market",Procedia Computer Science,Volume 167,2020,ISSN 1877-0509.
< https://www.sciencedirect.com/science/article/pii/S1877050920307237 >

[3] Brownlee,Jason,"How to use Learning Curves to Diagnose Machine Learning Model Performance", 27 Feb. 2019.
<https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/ >

[4] Rockikz, Abdou, "How to Predict Stock Prices in Python using Tensorflow 2 and Keras", January 2021.
<https://www.thepythoncode.com/article/stock-price-prediction-in-python-using-tensorflow-2-and-keras >

[5] Loukas, Serafeim, "Time - Series Forecasting: Predicting Stock Prices using Facebook's Prophet Model", 13 July 2020.
< https://towardsdatascience.com/time-series-forecasting-predicting-stock-prices-using-facebooks-prophet-model-9ee1657132b5 >

[6] Mehtab,Sidra; Sen,Jaydip; Dutta,Abjishek; "Stock Price Prediction Using Machine Learning and LSTM-Based Deep Learning Models", 20 September 2020.
< https://arxiv.org/abs/2009.10819 >

[7] Ahmed,Yacoub, "Predicting stock prices using deep learning", 11 october 2019.
<https://towardsdatascience.com/getting-rich-quick-with-machine-learning-and-stock-market-predictions-696802da94fe >

[8] Singh,Aishwarya, "Stock Price Prediction Using Machine Learning and Deep Learning Techniques (with Python codes), 25 October 2018.

<https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/ >

[9] Dhyani,Rajat, “Stock Price Predictor”, 5 June 2018.
< https://github.com/Rajat-dhyani/Stock-Price-Predictor >

[10] Walia,Mrinal, “Long Short Term Memory (LSTM) and how to implement LSTM using Python”, 30 August 2020.
< https://datascienceplus.com/long-short-term-memory-lstm-and-how-to-implement-lstm-using-python/ >

[11] Mwiti,Derrick,”Using a Keras Long Short-Term Memory (LSTM) Model to Predict Stock Prices”, November 2018.
< https://www.kdnuggets.com/2018/11/keras-long-short-term-memory-lstm-model-predict-stock-prices.html >

[12] Brownlee,Jason” How to Model Volatility with ARCH and GARCH for Time Series Forecasting in Python”,24 Aug. 2018
<https://machinelearningmastery.com/develop-arch-and-garch-models-for-time-series-forecasting-in-python/>

[13] Lakshmanan, Swetha, “How, When and Why Should You Normalize / Standardize / Rescale Your Data?“, 16 May 2019.
<https://towardsai.net/p/data-science/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff#:~:text=For%20machine%20learning%2C%20every%20dataset,%2C%20and%20income(x2).&text=So%20we%20normalize%20the%20data,variables%20to%20the%20same%20range >