National College of Ireland

Enhancing Stock Market Predictions with Deep Neural Networks and Time Series Analysis

MSc Research Project
Data Analytics

RAKESH PIDUGU
X22167706

School of Computing
National College of Ireland

Supervisor: Abdul Shahid

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: RAKESH PIDUGU

Student ID: x22167706

Programme: Msc in Data analytics **Year:** 2023-2024

Module: Research Project

Lecturer: Abdul Shahid

Submission Due Date: 25/04/2024

Project Title: Enchancing Stock Market Predictions with Deep Neural Networks And Time Series Analysis

Word Count: 756 **Page Count:** 12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: RAKESH PIDUGU

Date: 25/04/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

CONFIGURATION MANUAL

RAKESH PIDUGU
x22167706

1. Introduction

This manual situates that, how to perform the developments to run the code which was implementation for prediction of Microsoft Stock prices . The code is written in Python language where we keeped the all pre-requestiques for running of the developed program.

2. System Specification

The development of the forecasting of the Microsoft Stock prices was developed throughout the following hardware configurations:

- Process: Intel i5 10 generation
- Operating System: Windows 11
- Ram: 8 GB
- Solid State Drive (SSD): 512GB

3. Softwares Used:

The following tools are used which helps in the development of this project of Microsoft stock prices prediction:

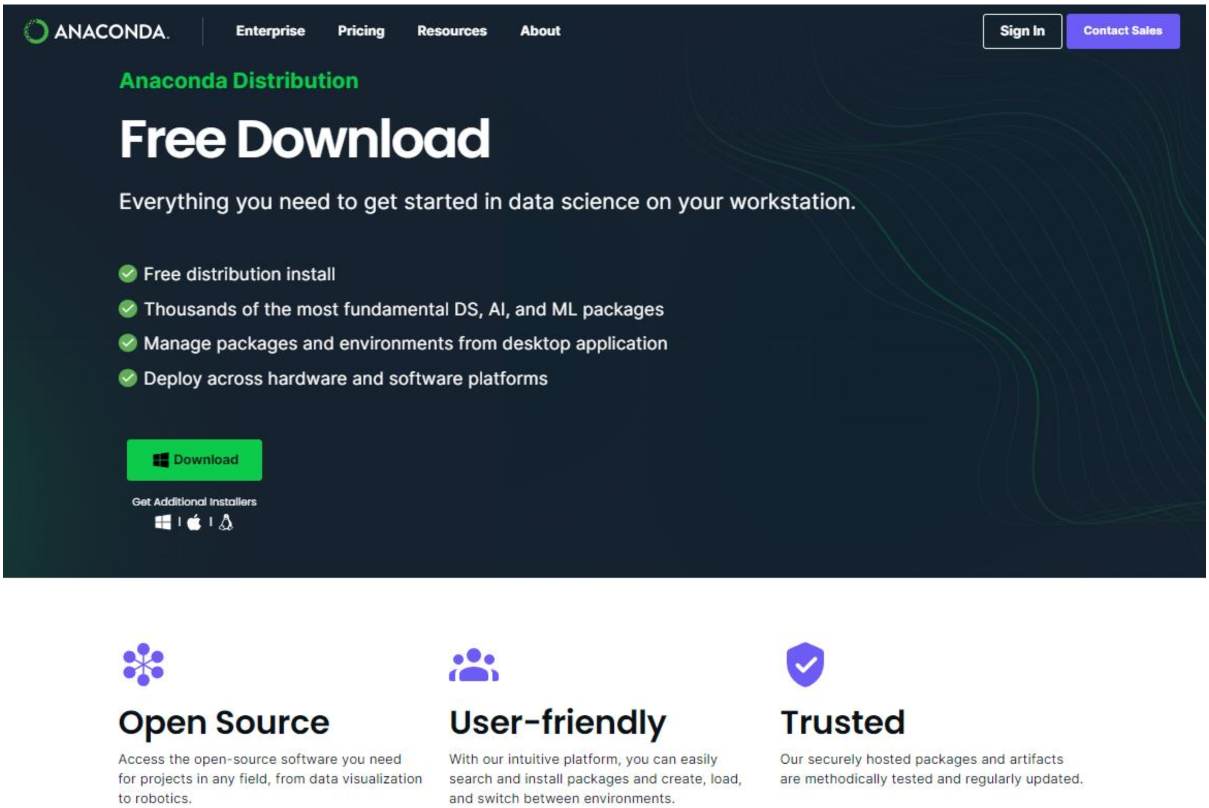
- Python
- Anaconda
- Jupyter

4. Steps to Download and Install the Software:

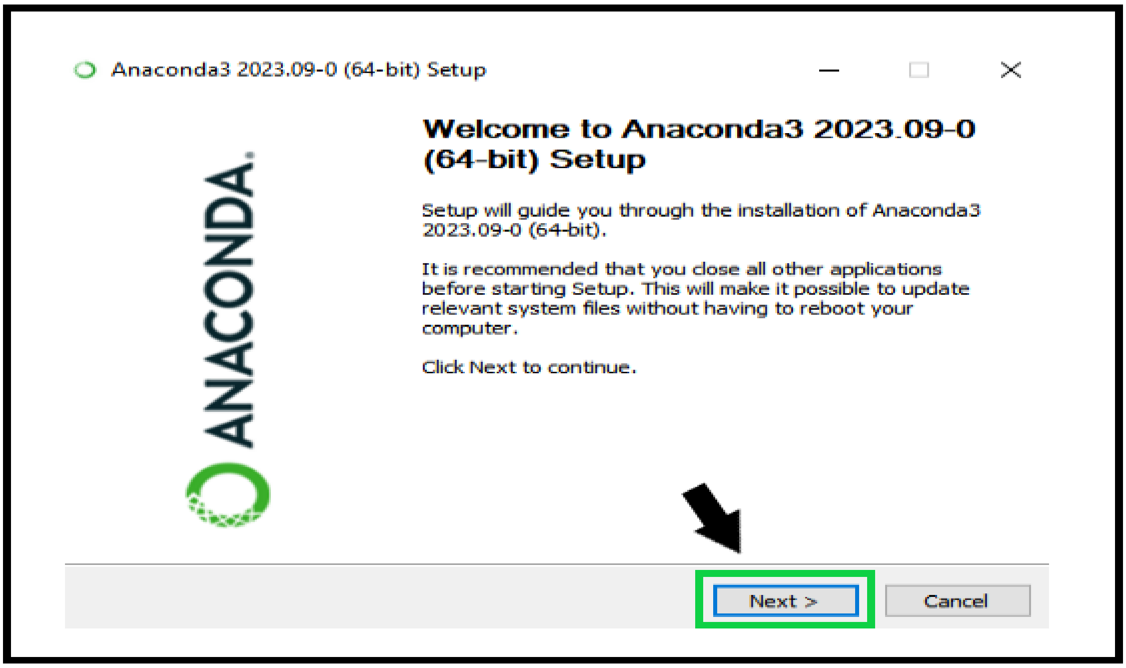
The sections describes how to install the anaconda

- Download and install the Anaconda from their official website: https://www.anaconda.com

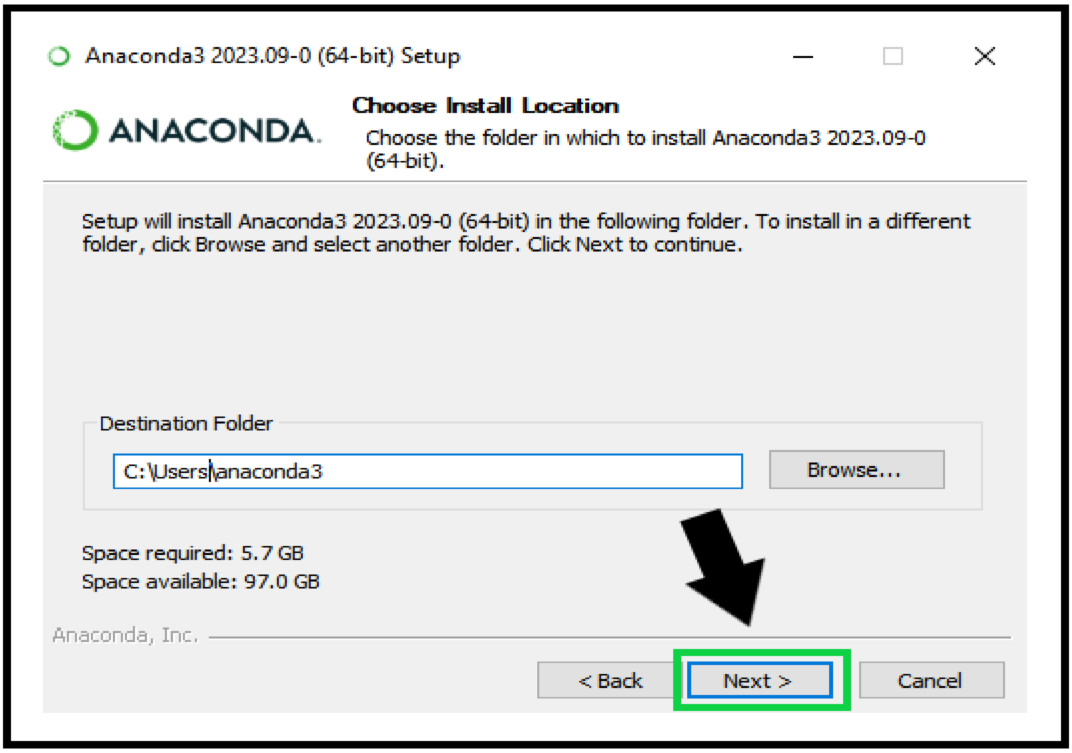
- Click on Download to download the anaconda to your operating system.



- After downlaoding the anaconda from their offical website.
- Follow the instructions to install through steps provided on the website.
- Open downloaded anaconda's setup application to install the software



- Now Click on Next as depicted to the above image illustration to proceed with next step



- Specify the path where you wants to install the application and then click on "Next" till nex to install the application as depicted to the above image illustration

5. Dataset Source

The dataset for this study, I used the dataset from kaggle which is known for collaborates with other users and publishes the dataset. So I chosen the dataset for historical Microsoft Stock dataset which is from 1986-2021

Dataset Source: Micosoft Stocks (Historical Dataset)

6. Execution of the Code Implementation

Open the jupyter from the anaconda's navigator and then open the files to run that development of the Microsoft Stocks prices prediction

As the information represented are the process or step to execute the code implementation for Microsoft Stock prices.

A. Import the required libraries

```python
#import the required libaries
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from keras.models import Sequential, Model
from sklearn.preprocessing import MinMaxScaler
from keras.layers import Dense, LSTM, Dropout, GRU, SimpleRNN, Reshape, Add, concatenate
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

B. Load the fetched dataset from the kaggle to the jupyter environment/

In [2]:
```python
#load the Microsoft previous data
dataFrame = pd.read_csv("MSFT.csv")
```

This dataset appears to contain information related to Microsoft (MSFT) price and trading data. Each row in the dataset represents a specific timestamp and includes the following columns:

Column	Description
Date	The date and time in a human-readable format (e.g., "1986-03-13").
Open	The opening price of Microsoft stock at that date.
High	The highest price of Microsoft stock during the time period covered by that date.
Low	The lowest price of Microsoft stock during the same time period.
Close	The closing price of Microsoft stock at that date.
Volume	The volume of Microsoft stocks traded during that time period, measured in MSFT (Microsoft).

This dataset we use to analyze the historical price and trading activity of Microsoft Stocks in relation to the U.S. dollar over specific time intervals.

In [3]:
```python
#view the first 5 values of the attributes
dataFrame.head()
```

Out[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.061378	1031788800
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.063570	308160000
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.064667	133171200
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.063022	67766400
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.061926	47894400

```
#basic information about the dataset
dataFrame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9008 entries, 0 to 9007
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       9008 non-null   object 
 1   Open       9008 non-null   float64
 2   High       9008 non-null   float64
 3   Low        9008 non-null   float64
 4   Close      9008 non-null   float64
 5   Adj Close  9008 non-null   float64
 6   Volume     9008 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 492.8+ KB
```

The dataset contains the 9008 rows with 9 different columns

C. Preprocess the dataset

```
In [12]: # Convert the 'Date' column to a datetime format
         dataFrame['Date'] = pd.to_datetime(dataFrame['Date'])
```

```
In [13]: # Reset the index of the DataFrame
         dataFrame = dataFrame.reset_index(drop=True)
```

Now view the preprocessed dataset

```
#view the processed dataframe first five attributes
dataFrame.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.061378	1031788800
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.063570	308160000
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.064667	133171200
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.063022	67766400
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.061926	47894400

D. After the dataset is get preprocessed it is now ready for the further steps includes splitting of dataset

Splitting of Dataset

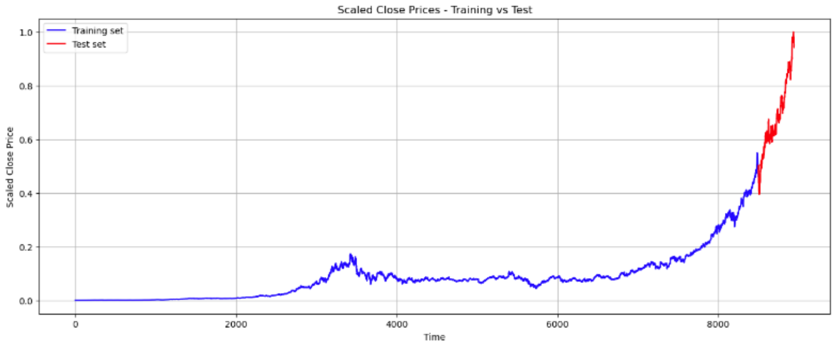
- **Here the dataset is get splitted in the training and testing dataframes**

```
# Prepare the data for deep learning models
def prepare_data(data, look_back):
    X, y = [], []
    for i in range(len(data)-look_back):
        X.append(data[i:(i+look_back), 0])
        y.append(data[i+look_back, 0])
    return np.array(X), np.array(y)

look_back = 60
X, y = prepare_data(scaled_data, look_back)

# Reshape data for LSTM
X = np.reshape(X, (X.shape[0], X.shape[1], 1))
```

```
# Splitting the dataset into training and testing sets
train_size = int(len(X) * 0.95)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```



E. Now initialize and train the deep learning models.

1. Long-Short Term Memory (LSTM)

Model Initialization & Training

- **Model Development: Various deep learning architectures, including LSTM, GRU, ResNet, and RNN, are implemented and trained on the historical stock price data. These models are designed to capture temporal dependencies, nonlinear relationships, and complex patterns present in financial time series data.**

Long-Short Term Memory Model (LSTM)

In [39]:
```
# Build the LSTM model
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
```

```
WARNING:tensorflow:From c:\Users\rohit\anaconda3\Lib\site-packages\keras\src\backend.py:873: The name
tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.
```

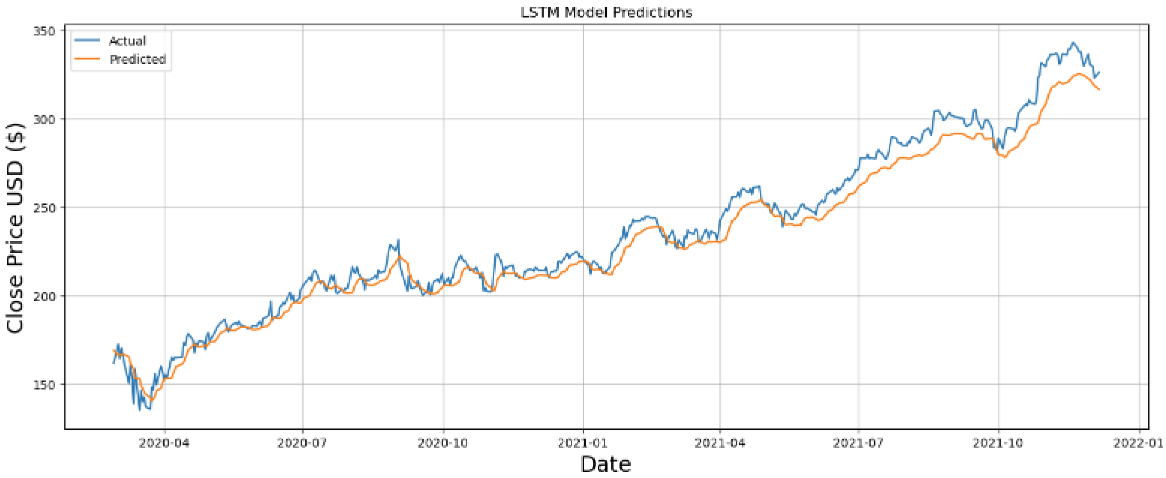
In [40]:
```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
WARNING:tensorflow:From c:\Users\rohit\anaconda3\Lib\site-packages\keras\src\optimizers\__init__.py:3
09: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

In [41]:
```
# Train the model
model.fit(X_train, y_train, batch_size=1, epochs=1)
```

```
WARNING:tensorflow:From c:\Users\rohit\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: T
he name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue i
nstead.

8500/8500 [==============================] - 110s 13ms/step - loss: 2.5587e-04
```



2. Gated Recurrent Unit (GRU)

Gated Recurrent Unit Model (GRU)

```
# Build the GRU model
model_gru = Sequential()
model_gru.add(GRU(128, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model_gru.add(GRU(64, return_sequences=False))
model_gru.add(Dense(25))
model_gru.add(Dense(1))
```

```
# Compile the GRU model
model_gru.compile(optimizer='adam', loss='mean_squared_error')
```

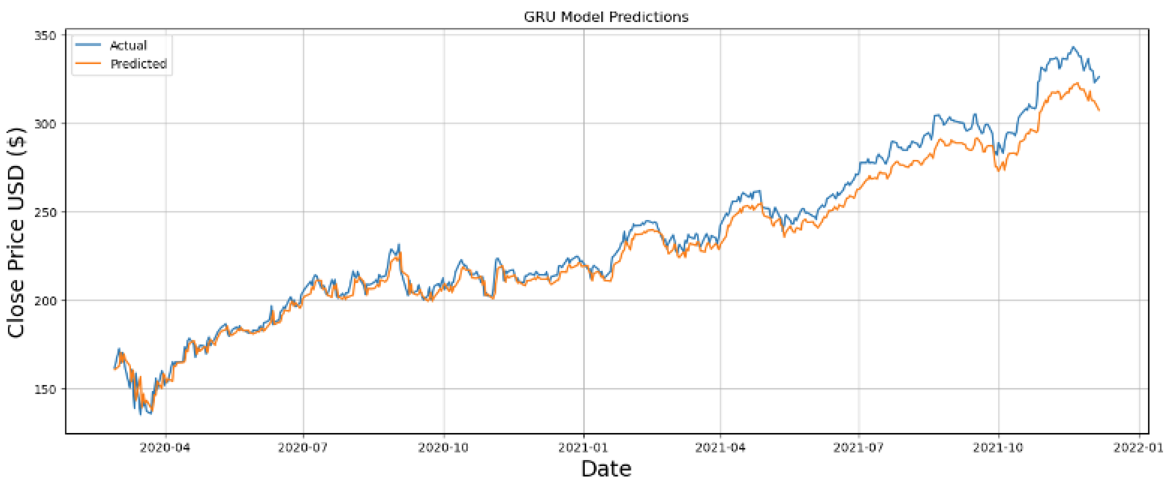
```
# Train the GRU model
model_gru.fit(X_train, y_train, batch_size=1, epochs=1)
```

```
8500/8500 [==============================] - 117s 14ms/step - loss: 1.1285e-04
```

```
<keras.src.callbacks.History at 0x1a239445710>
```

```
# Predictions using GRU model
predictions_gru = model_gru.predict(X_test)
predictions_gru = scaler.inverse_transform(predictions_gru)
```

```
14/14 [==============================] - 1s 9ms/step
```



3. Residual Network (ResNet)

Residual Network Model (ResNet)

```
# Build the ResNet model
model_resnet = Sequential()
model_resnet.add(LSTM(128, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model_resnet.add(LSTM(64, return_sequences=True))
model_resnet.add(Reshape((-1,)))
model_resnet.add(Dense(25))
model_resnet.add(Dense(1))
```

```
# Compile the ResNet model
model_resnet.compile(optimizer='adam', loss='mean_squared_error')
```

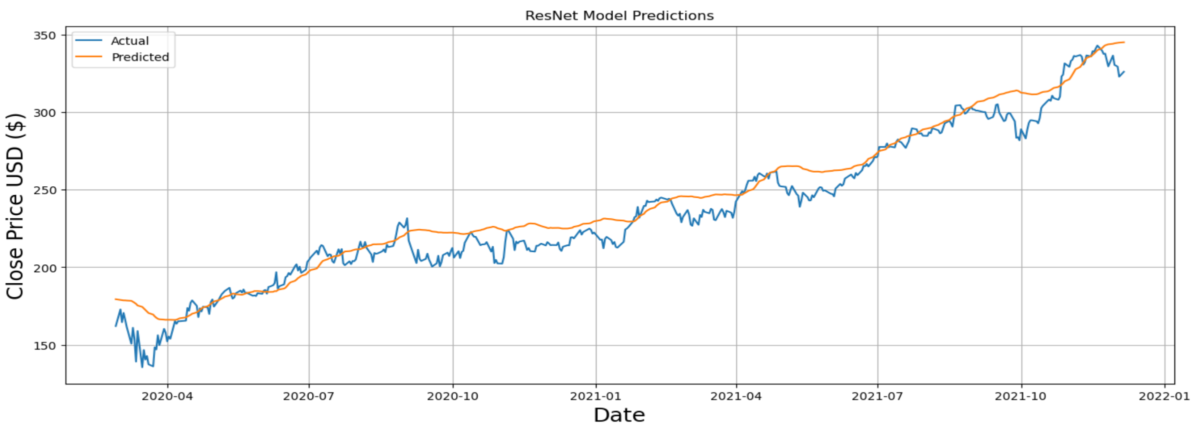
```
# Train the ResNet model
model_resnet.fit(X_train, y_train, batch_size=1, epochs=1)
```

```
8500/8500 [==============================] - 114s 13ms/step - loss: 3.9590e-04
```

```
<keras.src.callbacks.History at 0x1a22a52dd90>
```

```
# Predictions using ResNet model
predictions_resnet = model_resnet.predict(X_test)
predictions_resnet = scaler.inverse_transform(predictions_resnet)
```

```
14/14 [==============================] - 1s 11ms/step
```



4. Recurrent Neural Network (RNN)

Recurrent Neural Network Model (RNN)

```
# Build RNN model
model_rnn = Sequential()
model_rnn.add(SimpleRNN(128,return_sequences=True, input_shape = (X_train.shape[1], 1)))
model_rnn.add(SimpleRNN(64, return_sequences=False))
model_rnn.add(Dense(25))
model_rnn.add(Dense(1))
```

```
# Compile the RNN model
model_rnn.compile(optimizer='adam', loss='mean_squared_error')
```

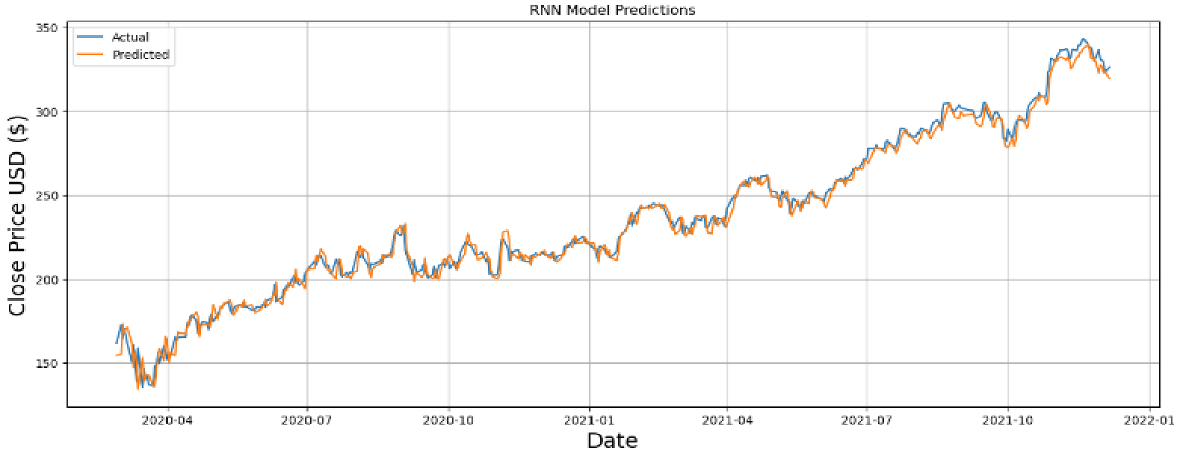
```
# Train the RNN model
model_rnn.fit(X_train, y_train, batch_size=1, epochs=1)
```

```
8500/8500 [==============================] - 76s 9ms/step - loss: 8.1043e-04
```

```
<keras.src.callbacks.History at 0x1a232603150>
```

```
# Predictions using RNN model
predictions_rnn = model_rnn.predict(X_test)
predictions_rnn = scaler.inverse_transform(predictions_rnn)
```

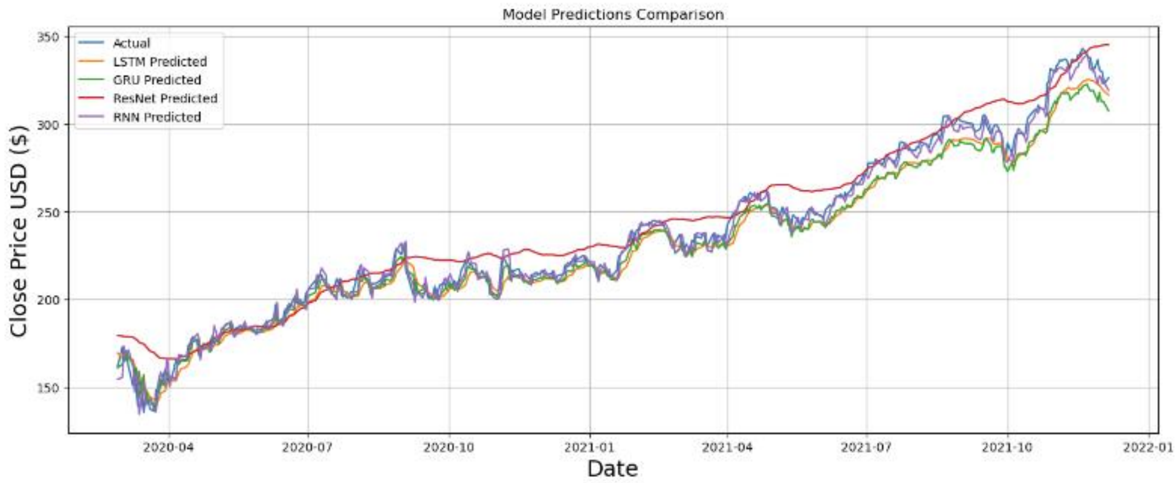
```
14/14 [==============================] - 0s 5ms/step
```



Metrics Results of Deep learning models

Model	RMSE	MAE
LSTM	233.30	229.24
GRU	233.40	229.54
RNN	238.47	233.99
ResNet	246.16	241.68

Date	Close	LSTM Predictions	GRU Predictions	RNN Predictions	ResNet Predictions
2020-02-28	162.009995	169.268890	160.877731	154.434311	179.350037
2020-03-02	172.789993	166.481445	162.881531	155.267242	178.885635
2020-03-03	164.509995	166.995728	170.545700	173.281296	178.705261
2020-03-04	170.550003	166.444656	164.981628	167.898895	178.608826
2020-03-05	166.270004	167.059341	169.817184	170.085800	178.607208



Hybrid Model

Hybrid Model

- **Building upon the insights gained from individual models, hybrid models are developed by integrating multiple deep learning architectures. These hybrid models aim to leverage the complementary strengths of different architectures to improve prediction accuracy and robustness.**

```
# Define input shape
input_shape = (X_train.shape[1], 1)

# Define LSTM branch
lstm_branch = Sequential()
lstm_branch.add(LSTM(128, return_sequences=True, input_shape=input_shape))
lstm_branch.add(LSTM(64, return_sequences=False))

# Define GRU branch
gru_branch = Sequential()
gru_branch.add(GRU(128, return_sequences=True, input_shape=input_shape))
gru_branch.add(GRU(64, return_sequences=False))

# Define ResNet branch
resnet_branch = Sequential()
resnet_branch.add(LSTM(128, return_sequences=True, input_shape=input_shape))
resnet_branch.add(LSTM(64, return_sequences=True))
resnet_branch.add(Reshape((-1,)))
resnet_branch.add(Dense(25))

#Define RNN branch
rnn_branch = Sequential()
rnn_branch.add(SimpleRNN(128,return_sequences=True, input_shape = input_shape))
rnn_branch.add(SimpleRNN(64, return_sequences=False))
rnn_branch.add(Dense(25))
rnn_branch.add(Dense(1))
```

```
# Concatenate branches
combined_model = concatenate([lstm_branch.output, gru_branch.output, resnet_branch.output, rnn_branch.output])

# Additional Dense layers
combined_model = Dense(25)(combined_model)
combined_model = Dense(1)(combined_model)
```

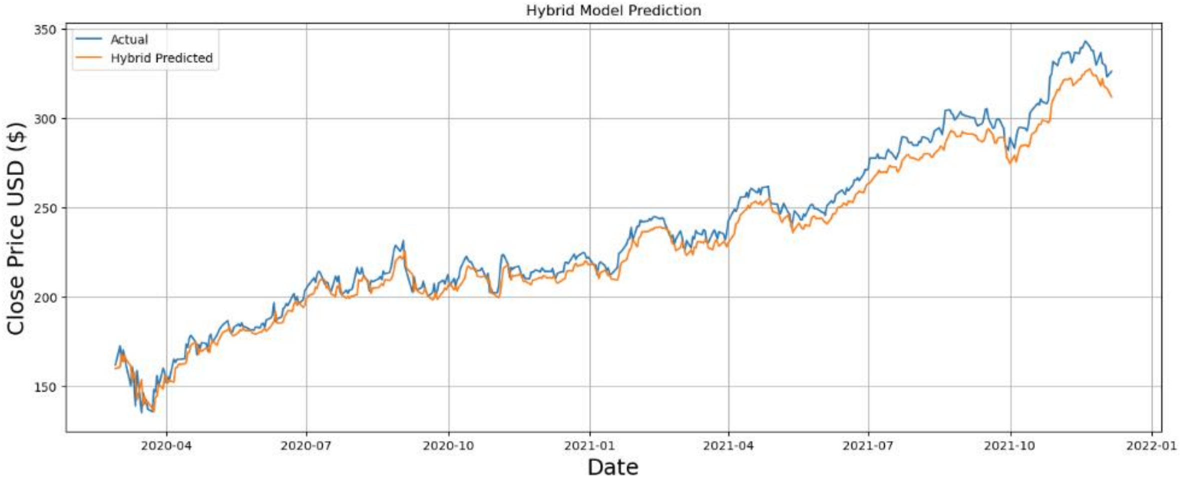
```
# Create model
hybrid_model = Model(inputs=[lstm_branch.input, gru_branch.input, resnet_branch.input, rnn_branch.input], outputs=combined_model)

# Compile the model
hybrid_model.compile(optimizer='adam', loss='mean_squared_error')
```

```
# Train the model
hybrid_model.fit([X_train, X_train, X_train, X_train], y_train, batch_size=1, epochs=1)
```

```
8500/8500 [==============================] - 210s 24ms/step - loss: 1.5448e-04
```

This plot demonstrates the predictions of hybrid model after integrating the multiple deep learning models.



Date	Close	LSTM Predictions	GRU Predictions	RNN Predictions	ResNet Predictions	Hybrid Predictions
2020-02-28	162.009995	169.268890	160.877731	154.434311	179.350037	160.027878
2020-03-02	172.789993	166.481445	162.881531	155.267242	178.885635	160.841934
2020-03-03	164.509995	166.995728	170.545700	173.281296	178.705261	168.110443
2020-03-04	170.550003	166.444656	164.981628	167.898895	178.608826	163.759842
2020-03-05	166.270004	167.059341	169.817184	170.085800	178.607208	167.407547

This configure manual provides the a comprehensive guide for configuring, execution of code , and understanding the Bitcoin future price forecasting implementation.

References

Anaconda: https://docs.anaconda.com/free/anaconda/install/windows/
Kaggle Dataset Source:Micosoft Stocks (Historical Dataset)