PROJECT REPORT

ARTIFICIAL INTELLIGENCE

BAHRIA UNIVERSITY KARACHI CAMPUS
Department of Computer Science

Module wise Work/Task Distribution

DEPARTMENT OF COMPUTER SCIENCE
WEB ENGINEERING LAB
PROJECT FORM

ABSTRACT/PROBLEM STATEMENT

The popularity of cryptocurrencies skyrocketed in 2017 due to several consecutive months of exponential growth of their market capitalization. The prices peaked at more than $800 billion in January 2018. Although machine learning has been successful in predicting stock market prices through a host of different time series models, its application in predicting cryptocurrency prices has been quite restrictive. The reason behind this is obvious as prices of cryptocurrencies depend on a lot of factors like technological progress, internal competition, pressure on the markets to deliver, economic problems, security issues, political factor etc. Their high volatility leads to the great potential of high profit if intelligent inventing strategies are taken. Unfortunately, due to their lack of indexes, cryptocurrencies are relatively unpredictable compared to traditional financial predictions like stock market prediction.

TABLE OF CONTENT

CHAPTER 1

CHAPTER 2

CHAPTER 3

CHAPTER 4

CHAPTER 5

CHAPTER 6

CHAPTER 1

1. INTRODUCTION

This project will be giving us predictions on the most volatile market throughout the world. We will be working on live data which is retrieved through API from https://min-api.cryptocompare.com/data/histoday?fsym=BTC&tsym=CAD&limit=365. The coin we have chosen to work on is BITCOIN. Currently we are using data based on the time period of a year which means our data comprises of data from year 2021 till the present day and moving on.

1.1 Project Description

In this project we will be going through a four step process to predict cryptocurrency prices:

1. Getting real-time cryptocurrency data.
2. Prepare data for training and testing.
3. Predict the price of cryptocurrency using LSTM neural network.
4. Visualize the prediction results.

1.2 Scope of the Project

As we can already observe that the world is showing huge interest towards the crypto market, therefore we can easily conclude that this project will become a need for all those who are interested in getting involved in crypto.

1.3 Project Features

The project will consider all the live data input and after applying all the applied functions it will predict the rise and fall of BITCOIN respectively.

1.4 REQUIREMENTS SPECIFICATION

- *The program can be accessed through any IDE that can run python on it and has access to the listed libraries.*
- *For example: Google Colab, Jupyter Notebook, Spyder etc.*

1.5 Characteristics of Project

➢ Our project is working on the following libraries:

```
%tensorflow_version 2.x
import json
import requests
from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout, LSTM
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
```

- from sklearn.metrics import mean_absolute_error
- %matplotlib inline

➢ In these libraries we have also used magic function of matplotlib that enables the inline plotting, where the plots/graphs will be displayed just below the cell where your plotting commands are written.

1.6 DATASET

The dataset contains total of 5 features. The details for them are as follows:

- Close Price — It is the market close price for currency for that particular day.
- High Price — It is highest price of currency for the day.
- Low Price — It is the lowest price for currency for that day.
- Open Price — It is market open price for currency for that day.
- Volume — The volume of currency that is being in trade for that day.

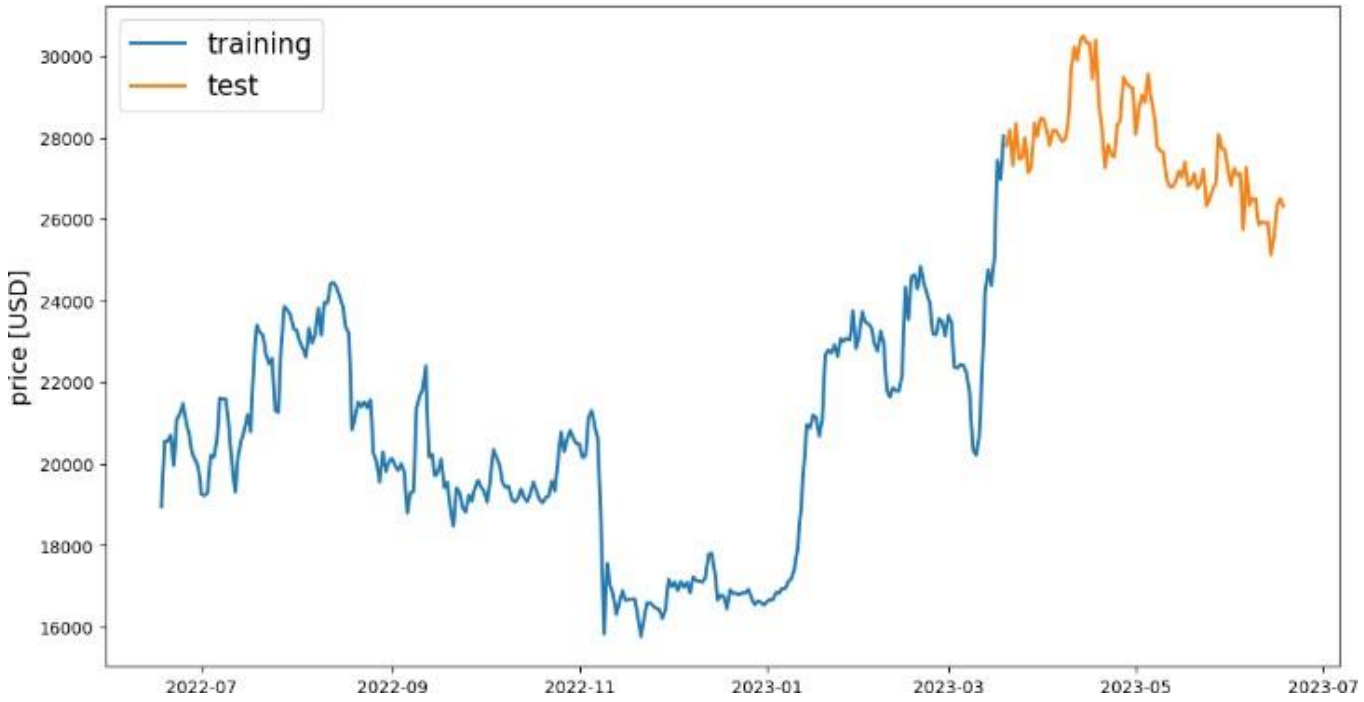
time	high	low	open	volumefrom	volumeto	close	conversionType	conversionSymbol
2022-06-18	20722.69	17600.75	20432.26	100138.03	1.898531e+09	18954.25	direct	
2022-06-19	20776.87	17936.00	18954.25	61181.89	1.190970e+09	20553.48	direct	
2022-06-20	21016.02	19616.32	20553.48	48976.59	9.959407e+08	20550.91	direct	
2022-06-21	21694.70	20339.03	20550.91	45262.22	9.542922e+08	20699.21	direct	
2022-06-22	20862.67	19756.23	20699.21	50640.15	1.027726e+09	19956.16	direct	
...	...	...	...	...	...	...	...	...
2023-06-14	26073.49	24834.48	25925.20	26097.60	6.673818e+08	25126.91	direct	
2023-06-15	25743.71	24762.12	25126.91	32797.00	8.240060e+08	25575.24	direct	
2023-06-16	26478.23	25164.41	25575.24	27490.08	7.113611e+08	26329.75	direct	
2023-06-17	26776.83	26170.63	26329.75	12515.42	3.315899e+08	26510.42	direct	
2023-06-18	26689.18	26260.18	26510.42	9341.76	2.476430e+08	26321.86	direct	

[366 rows x 8 columns]

We checked for null values as well so that we can perform any required preprocessing on the data.

```
Out[3]: high          0
        low           0
        open          0
        volumefrom    0
        volumeto      0
        close         0
        dtype: int64
```

A graph between testing and training data:

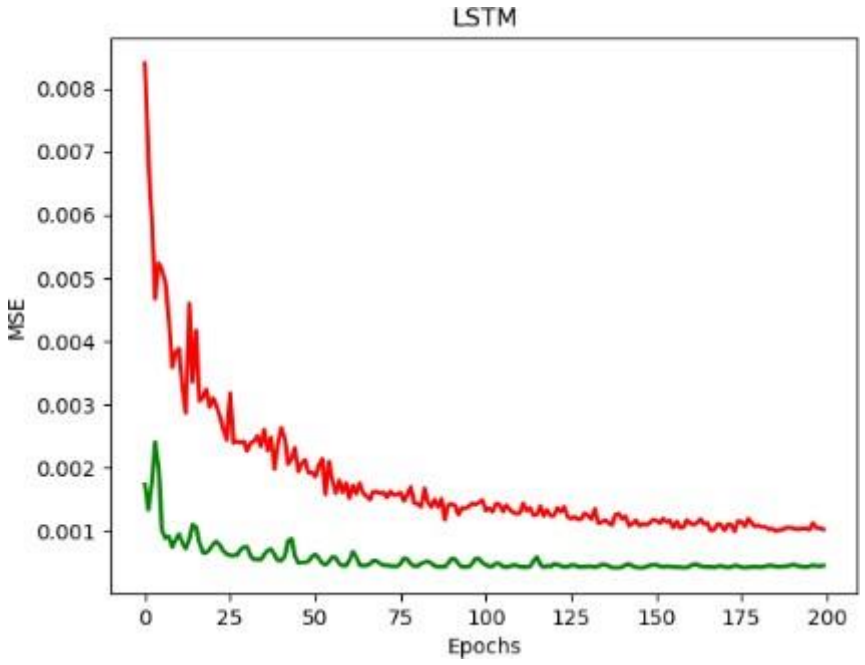



```
Out[8]: high          3.102378e+04
        low           3.001762e+04
        open          3.040437e+04
        volumefrom    4.819747e+04
        volumeto      1.474218e+09
        close         3.049301e+04
        Name: 2023-04-14 00:00:00, dtype: float64
```

Formation of LSTM model:

In [15]:
```
# build LSTM model with specified parameters
model = build_lstm_model(
    X_train, output_size=1, neurons=lstm_neurons, dropout=dropout, loss=loss,
    optimizer=optimizer)

# fit the model on training data and validate on test data
history = model.fit(
    X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=batch_size, verbose=1, shuffle=True)
```

```
Epoch 1/200
2/2 [==============================] - 7s 943ms/step - loss: 0.0084 - val_loss: 0.0017
Epoch 2/200
2/2 [==============================] - 0s 64ms/step - loss: 0.0069 - val_loss: 0.0013
Epoch 3/200
2/2 [==============================] - 0s 65ms/step - loss: 0.0060 - val_loss: 0.0017
Epoch 4/200
2/2 [==============================] - 0s 61ms/step - loss: 0.0047 - val_loss: 0.0024
Epoch 5/200
2/2 [==============================] - 0s 59ms/step - loss: 0.0052 - val_loss: 0.0020
Epoch 6/200
2/2 [==============================] - 0s 59ms/step - loss: 0.0051 - val_loss: 0.0010
Epoch 7/200
2/2 [==============================] - 0s 60ms/step - loss: 0.0049 - val_loss: 8.8171e-04
Epoch 8/200
2/2 [==============================] - 0s 64ms/step - loss: 0.0044 - val_loss: 9.0871e-04
Epoch 9/200
2/2 [==============================] - 0s 97ms/step - loss: 0.0036 - val_loss: 7.4595e-04
Epoch 10/200
```

In [17]:
```
# Get the target values for the test set
targets = test[target_col][window_len:]

# Predict the target values for the test set using the trained model
preds = model.predict(X_test).squeeze()

# Calculate the mean absolute error between the predicted and actual target values for the test set
mae = mean_absolute_error(preds, y_test)
mae
```

```
3/3 [==============================] - 1s 5ms/step
```

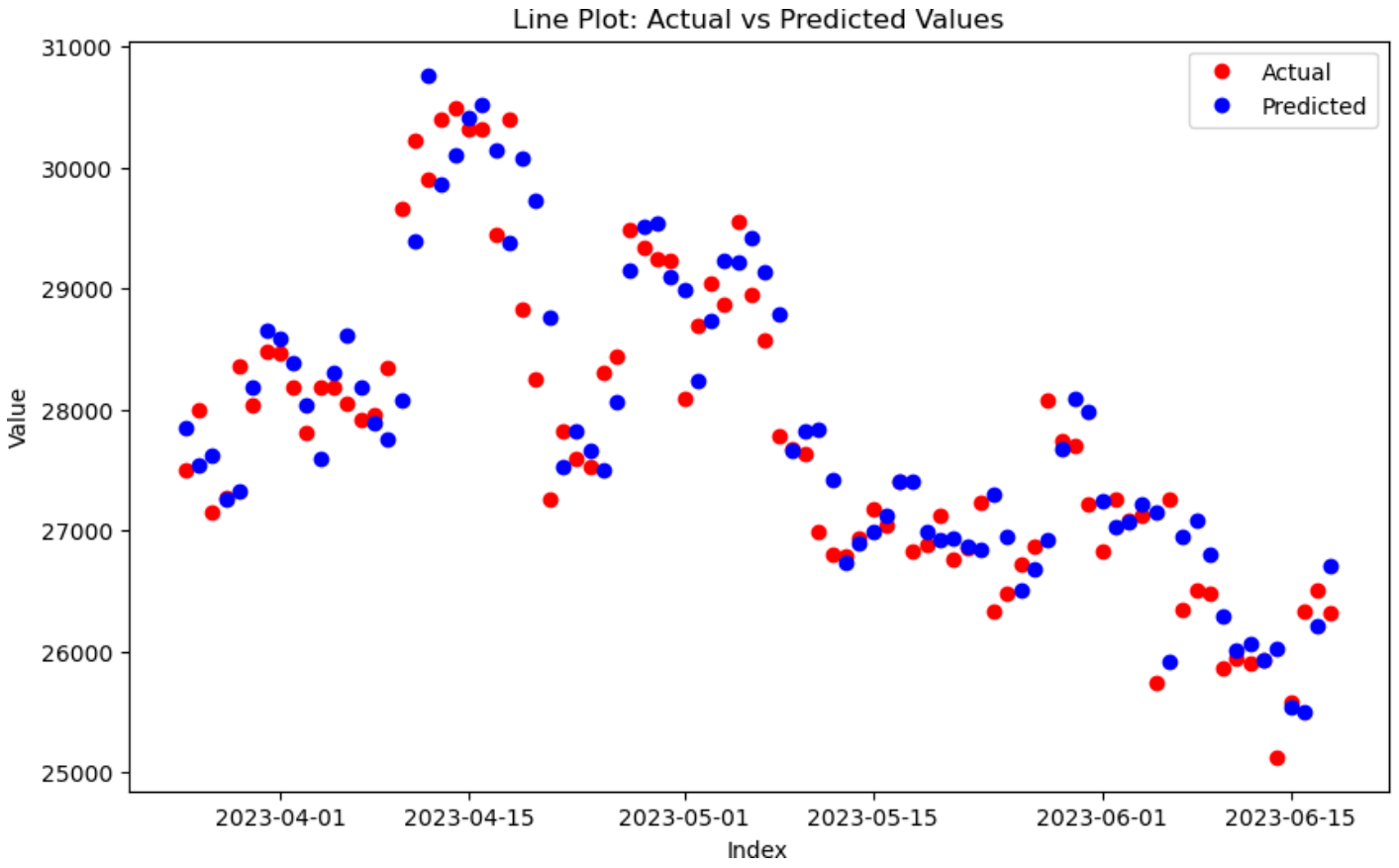
Out[17]: 0.01610508643943266

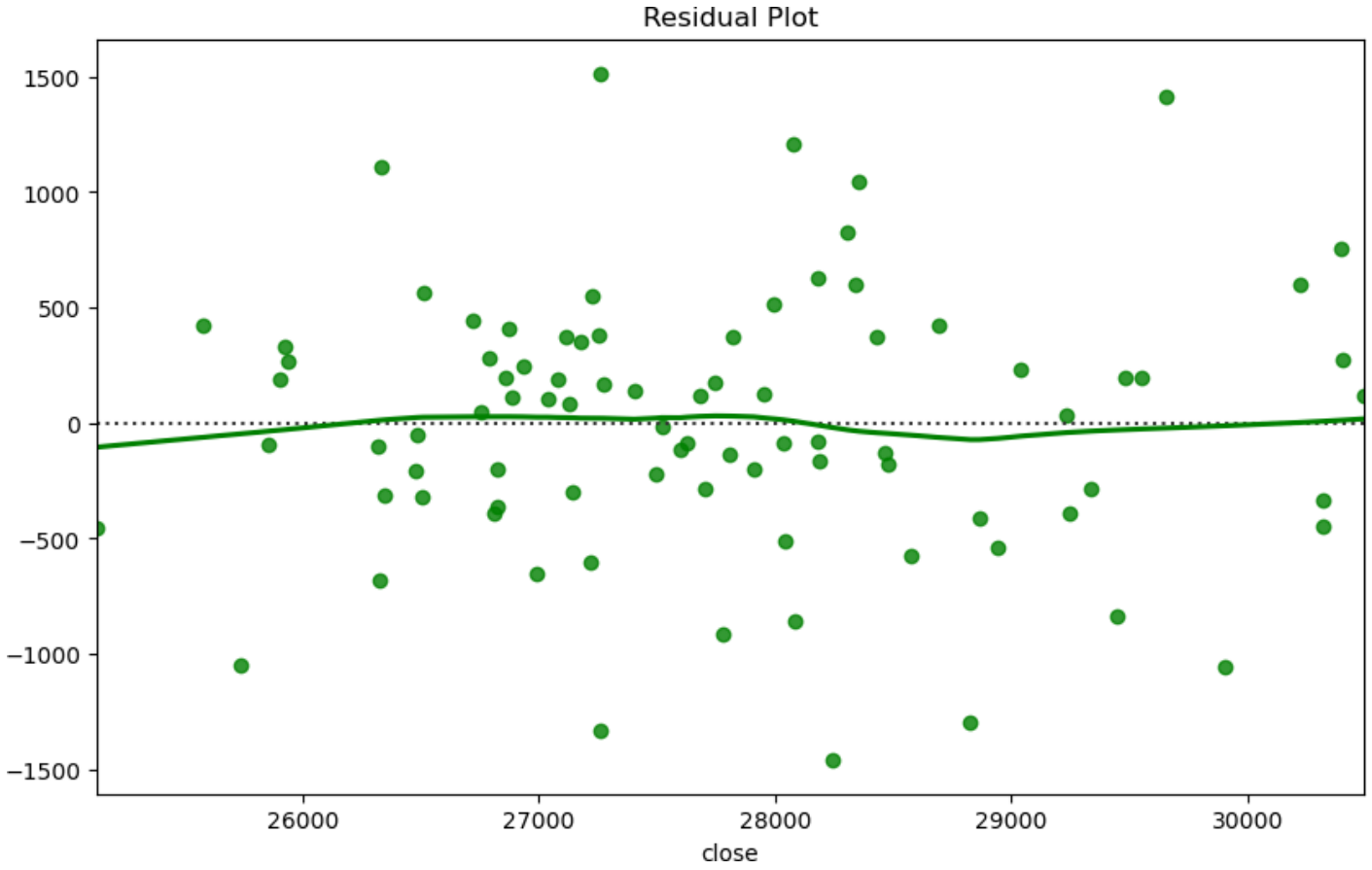
In [18]:
```
from sklearn.metrics import mean_squared_error # import the mean_squared_error function from sklearn library
MAE=mean_squared_error(preds, y_test) # calculate the mean squared error between predicted and actual values
MAE # display the mean squared error
```

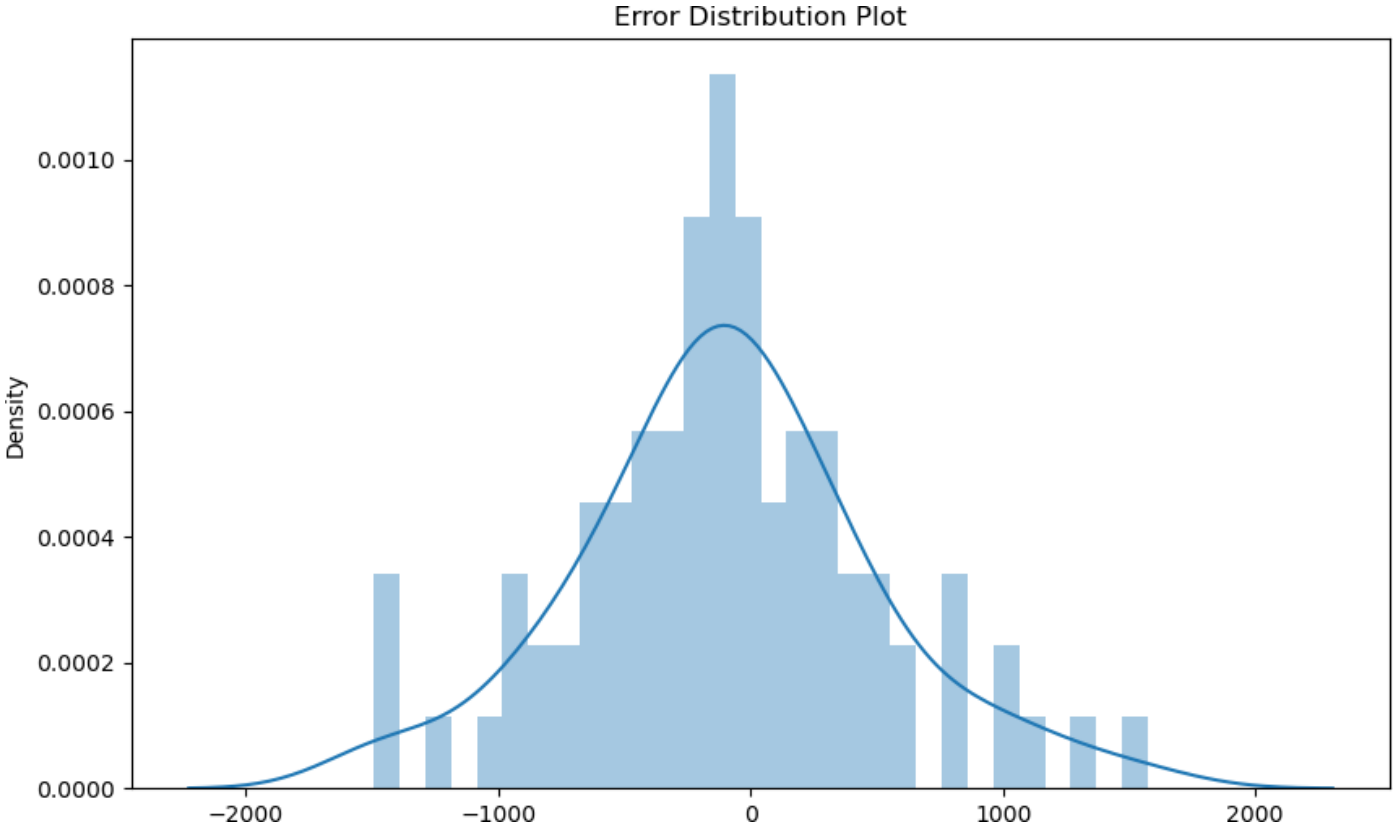
Out[18]: 0.0004499334586373896

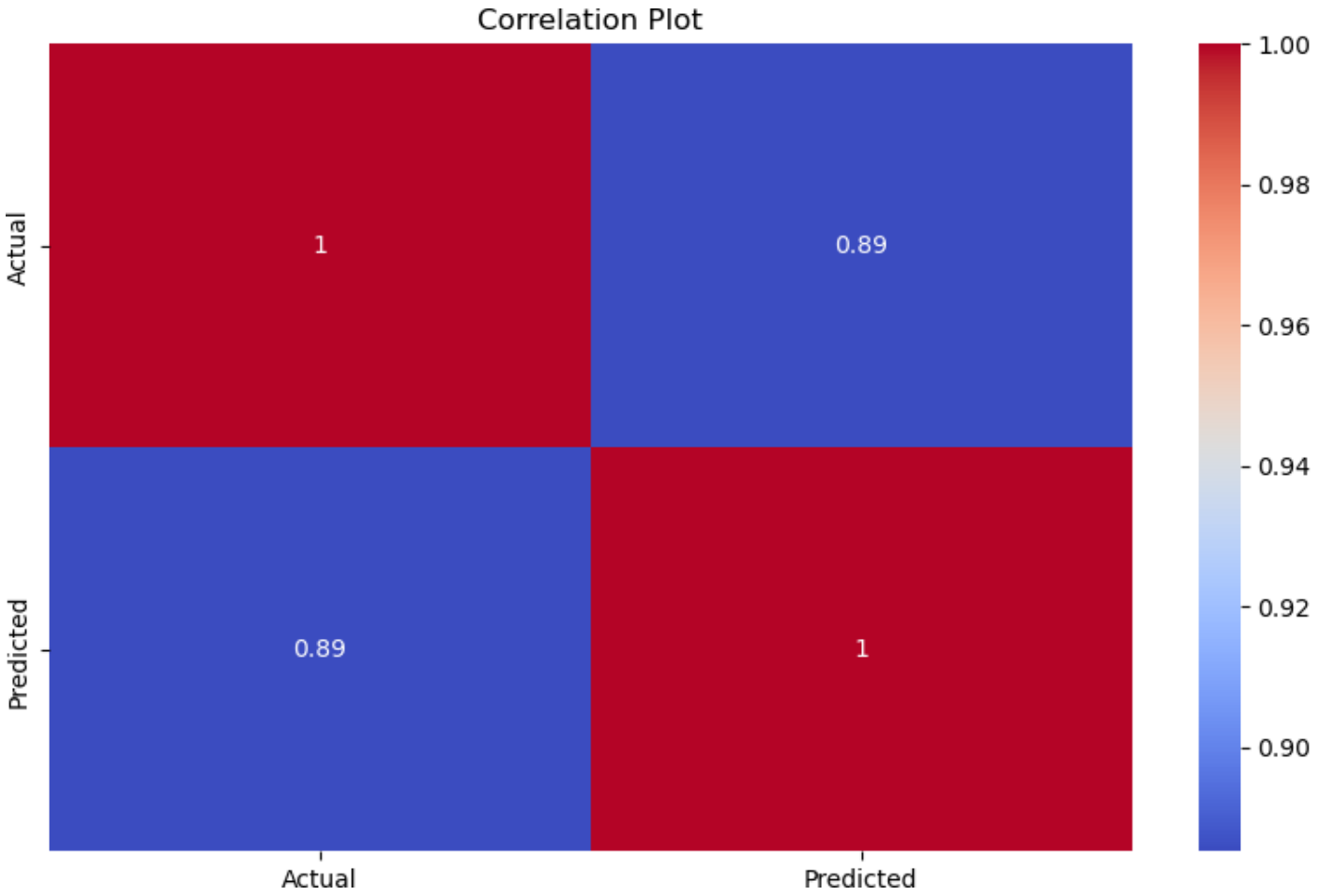
In [19]:
```
# import r2_score from the sklearn.metrics module
from sklearn.metrics import r2_score

# calculate the R-squared value using r2_score function from sklearn.metrics
# y_test contains the actual values of the target variable (bitcoin price)
# preds contains the predicted values of the target variable using the trained LSTM model
R2 = r2_score(y_test, preds)

# print the R-squared value
print(R2)
```

```
0.6882577102003578
```


Residual Plot
1500
1000
500
0
−500
−1000
−1500
26000
27000
28000
29000
30000
close


Error Distribution Plot
Density
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
−2000
−1000
0
1000
2000



CHAPTER 2

2. LITERATURE REVIEW

https://ieeexplore.ieee.org/abstract/document/8374483

https://medium.com/activewizards-machine-learning-company/bitcoin-price-forecasting-with-deep-learning-algorithms-eb578a2387a3

https://ieeexplore.ieee.org/document/8534911

Above are some research papers regarding predictions of BITCOIN and cryptocurrency.

CHAPTER 3

3. METHODOLOGY

We have used USD exchange rate and stored the real time data into a pandas data-frame. We used to_datetime() method to convert string Date time into Python Date time object. This is necessary as Date time objects in the file are read as a string object. Performing operations like time difference on a string rather a Date Time object is much easy.

Next, we split the data into two sets — training set and test set with 75% and 25% data respectively. The decision made here is just for the purpose of this tutorial.

3.1 OBJECTIVES

The objectives of this project are to forecast cryptocurrency prices using all the trading features like price, volume, open, high, low values present in the dataset.

CHAPTER 4

4. SYSTEM IMPLEMENTATION

4.1 Introduction

Our system is based on working of PHP and MySQL. We'll be attaching each and every code that would be covering all attributes of our website.

4.2 Code

```
# Import required libraries
import tensorflow as tf
import json
import requests
from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout, LSTM
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.metrics import mean_absolute_error

# Define the API endpoint to retrieve cryptocurrency data
cryptoApi = "https://min-api.cryptocompare.com/data/histoday?fsym=BTC&tsym=USD&limit=365" #1 year data

# Make the API request and store the response as a Pandas DataFrame
crypto = requests.get(cryptoApi)
data = pd.DataFrame(json.loads(crypto.content)['Data'])

# Set the DataFrame index to the 'time' column and convert it to a datetime object
data = data.set_index('time')
data.index = pd.to_datetime(data.index, unit='s')

# Set the target column to 'close'
target_col = 'close'

# Print the resulting DataFrame
print(data)
data.drop(["conversionType", "conversionSymbol"], axis = 'columns', inplace = True) #dropping unnecessary columns
data.isna().sum() #no missing values
```

```
def train_test_split(df, test_size=0.25):
    # Calculate the row index to split the data into train and test sets
    split_row = len(df) - int(test_size * len(df))
    # Slice the data up to the split row index for the training set
    train_data = df.iloc[:split_row]
    # Slice the data from the split row index to the end for the test set
    test_data = df.iloc[split_row:]
    # Return the training and test data sets
    return train_data, test_data
# Split the data into train and test sets with a test size of 25%
train, test = train_test_split(data, test_size=0.25)
def line_plot(line1, line2, label1=None, label2=None, title='', lw=2):
    # Create a new figure and axis object with the specified size
    fig, ax = plt.subplots(1, figsize=(13, 7))

    # Plot the first line with the specified label and line width
    ax.plot(line1, label=label1, linewidth=lw)

    # Plot the second line with the specified label and line width
    ax.plot(line2, label=label2, linewidth=lw)

    # Set the y-axis label and title of the plot
    ax.set_ylabel('price  [USD]',  fontsize=14)
    ax.set_title(title, fontsize=16)

    # Add a legend to the plot with the best location and font size
    ax.legend(loc='best', fontsize=16);
# line_plot is a custom function that takes in two data series and plots them on the same chart
# It is not shown here, but it should have been defined somewhere else in the notebook

train, test = train_test_split(data, test_size=0.25)
# This calls the train_test_split function defined earlier and splits the data into train and test sets,
# using a test size of 0.25 (meaning 25% of the data will be used for testing)

line_plot(train[target_col], test[target_col], 'training', 'test', title='')
# This calls the line_plot function to plot the train and test sets on the same chart,
# with the x-axis labeled as 'training' and 'test', and with a custom title provided (which is an empty string in this case)
# The target_col variable should have been defined earlier in the notebook and should contain the name of the column
# that contains the target variable (in this case, the closing price of Bitcoin)
data.loc[data['close'].idxmax()] #when bitcoin was at its highest in the past year
# This function normalizes the input dataframe to a zero base
def normalise_zero_base(df):
    return df / df.iloc[0] - 1

# This function normalizes the input dataframe to a minimum-maximum scale
def normalise_min_max(df):
    return (df - df.min()) / (data.max() - df.min())
# Define a function that takes a pandas dataframe and extracts a sequence of windowed data
def extract_window_data(df, window_len=5, zero_base=True):

    # Initialize an empty list to hold the windowed data
    window_data = []

    # Loop through the rows of the dataframe, stopping before the last `window_len` rows
    for idx in range(len(df) - window_len):

        # Extract a slice of the dataframe with a length of `window_len`
```

```
        tmp = df[idx: (idx + window_len)].copy()

        # If `zero_base` is True, normalize the windowed data so that the first value is zero
        if zero_base:
            tmp = normalise_zero_base(tmp)

        # Add the windowed data to the list
        window_data.append(tmp.values)

    # Return the list of windowed data as a numpy array
    return np.array(window_data)
# define function to prepare data for training
def prepare_data(df, target_col, window_len=10, zero_base=True, test_size=0.25):
    # split data into train and test sets
    train_data, test_data = train_test_split(df, test_size=test_size)

    # extract windowed data
    X_train = extract_window_data(train_data, window_len, zero_base)
    X_test = extract_window_data(test_data, window_len, zero_base)

    # get target values for train and test sets
    y_train = train_data[target_col][window_len:].values
    y_test = test_data[target_col][window_len:].values

    # normalize target values if zero_base=True
    if zero_base:
        y_train = y_train / train_data[target_col][:-window_len].values - 1
        y_test = y_test / test_data[target_col][:-window_len].values - 1

    return train_data, test_data, X_train, X_test, y_train, y_test
def build_lstm_model(input_data, output_size, neurons=100, activ_func='linear',
                     dropout=0.2, loss='mse', optimizer='adam'):
    # create a sequential model
    model = Sequential()
    # add LSTM layer with specified number of neurons and input shape
    model.add(LSTM(neurons, input_shape=(input_data.shape[1], input_data.shape[2])))
    # add dropout layer with specified rate
    model.add(Dropout(dropout))
    # add dense layer with specified number of output units
    model.add(Dense(units=output_size))
    # add activation function to the dense layer
    model.add(Activation(activ_func))
    # compile the model with specified loss function and optimizer
    model.compile(loss=loss, optimizer=optimizer)
    return model
# Set the random seed to make the results reproducible
np.random.seed(42)

# Define the window length for creating sequences of data
window_len = 5

# Define the test size as a fraction of the overall dataset size
test_size = 0.25

# Define whether to use a zero baseline or not
zero_base = True
```

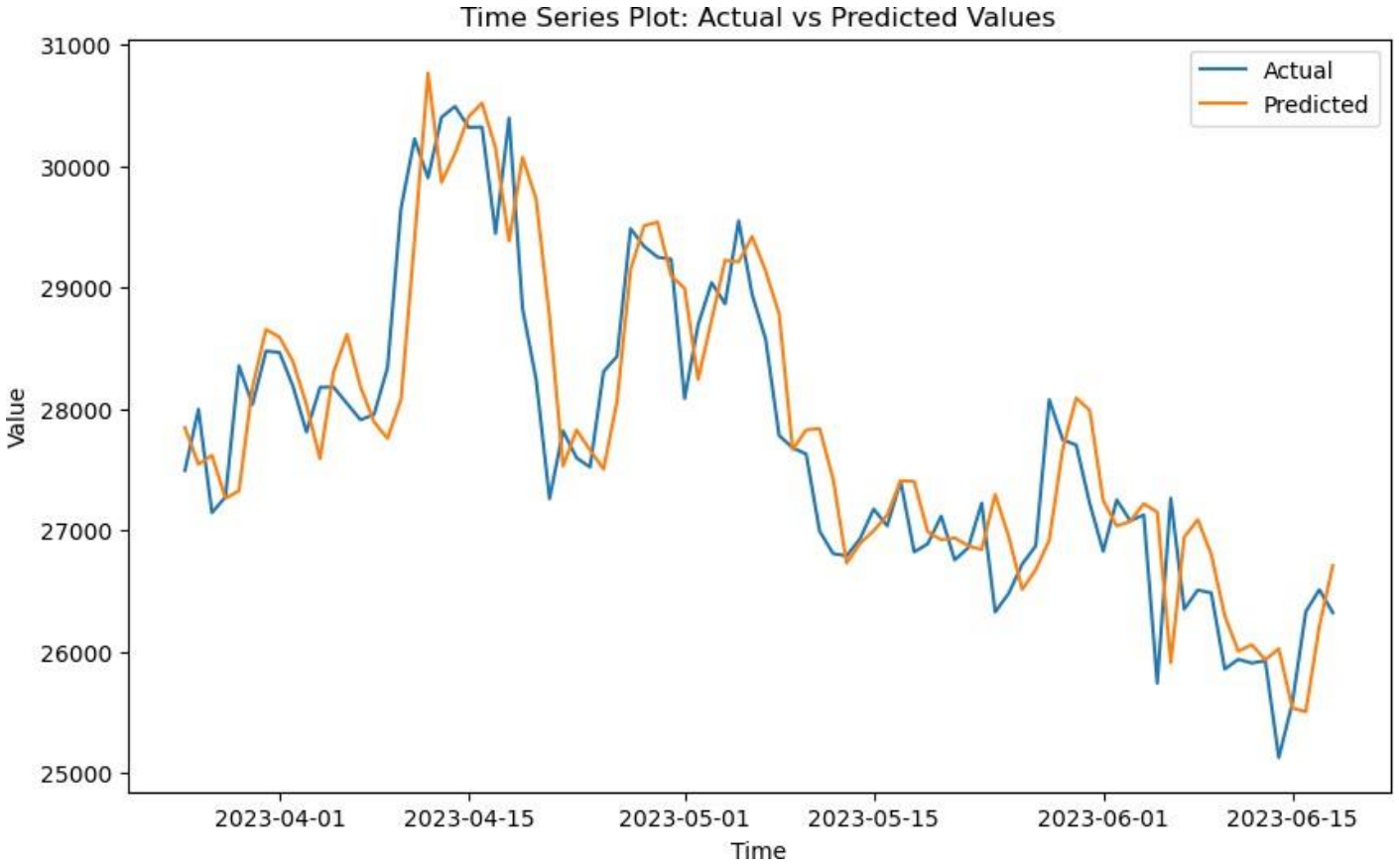
```
# Define the number of neurons in the LSTM layer
lstm_neurons = 100

# Define the number of epochs to train the model for
epochs = 200

# Define the batch size for training the model
batch_size = 200

# Define the loss function to be used during training
loss = 'mse'

# Define the dropout rate to be used during training
dropout = 0.2

# Define the optimizer to be used during training
optimizer = 'adam'
# This line calls the prepare_data function with the provided arguments
# and stores the returned values in the corresponding variables.
train, test, X_train, X_test, y_train, y_test = prepare_data(data, target_col, window_len=window_len,
zero_base=zero_base, test_size=test_size)
# build LSTM model with specified parameters
model = build_lstm_model(
    X_train, output_size=1, neurons=lstm_neurons, dropout=dropout, loss=loss,
    optimizer=optimizer)

# fit the model on training data and validate on test data
history = model.fit(
    X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=batch_size, verbose=1, shuffle=True)
import matplotlib.pyplot as plt

# Plotting the training loss over epochs in red
plt.plot(history.history['loss'], 'r', linewidth=2, label='Train loss')

# Plotting the validation loss over epochs in green
plt.plot(history.history['val_loss'], 'g', linewidth=2, label='Validation loss')

# Adding a title to the plot
plt.title('LSTM')

# Labeling the x-axis
plt.xlabel('Epochs')

# Labeling the y-axis
plt.ylabel('MSE')

# Displaying the plot
plt.show()
# Get the target values for the test set
targets = test[target_col][window_len:]

# Predict the target values for the test set using the trained model
preds = model.predict(X_test).squeeze()

# Calculate the mean absolute error between the predicted and actual target values for the test set
mae = mean_absolute_error(preds, y_test)
mae
```

```
from sklearn.metrics import mean_squared_error # import the mean_squared_error function from sklearn library
MAE=mean_squared_error(preds, y_test) # calculate the mean squared error between predicted and actual values
MAE # display the mean squared error
# import r2_score from the sklearn.metrics module
from sklearn.metrics import r2_score

# calculate the R-squared value using r2_score function from sklearn.metrics
# y_test contains the actual values of the target variable (bitcoin price)
# preds contains the predicted values of the target variable using the trained LSTM model
R2 = r2_score(y_test, preds)

# print the R-squared value
print(R2)
# Multiply the predicted values with the target values
# to get the predicted values in actual scale
preds = test[target_col].values[:-window_len] * (preds + 1)

# Create a pandas Series object with the index same as targets
# and the predicted values as data
preds = pd.Series(index=targets.index, data=preds)

# Plot the actual and predicted values using a line plot
line_plot(targets, preds, 'actual', 'prediction', lw=3)
import matplotlib.pyplot as plt
import seaborn as sns

# This line plot shows the actual values vs predicted values. It gives a visual representation of how close the predicted values are to the actual values.
plt.figure(figsize=(10,6))
plt.plot(targets, 'ro', label='Actual')
plt.plot(preds, 'bo', label='Predicted')
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Line Plot: Actual vs Predicted Values')
plt.legend()
plt.show()

# Hexbin Plot
#This hexbin plot shows the actual values vs predicted values. The darker areas show where there are more data points.
#It gives a visual representation of how close the predicted values are to the actual values.
plt.figure(figsize=(10,6))
plt.hexbin(targets, preds, gridsize=50, cmap='Greens')
plt.colorbar(label='Count in Bin')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Hexbin Plot: Actual vs Predicted Values')
plt.show()


# Time Series Plot
#This time series plot shows how the actual values and predicted values change over time.
#It gives a visual representation of how close the predicted values are to the actual values over time
plt.figure(figsize=(10,6))
plt.plot(targets, label='Actual')
plt.plot(preds, label='Predicted')
plt.xlabel('Time')
```

```
plt.ylabel('Value')
plt.title('Time Series Plot: Actual vs Predicted Values')
plt.legend()
plt.show()

# Residual Plot
#This residual plot shows the residuals (errors) vs the actual values.
#It gives a visual representation of the error over the range of actual values. The smoother line shows the trend in the errors.
residuals = targets - preds
plt.figure(figsize=(10,6))
sns.residplot(targets, residuals, lowess=True, color="g")
plt.title('Residual Plot')
plt.show()

# Error Distribution Plot
#This error distribution plot shows the histogram of the errors.
#It gives a visual representation of how the errors are distributed, which can indicate if there are any biases.
plt.figure(figsize=(10,6))
sns.distplot(residuals, bins=30, kde=True)
plt.title('Error Distribution Plot')
plt.show()

# Correlation Plot
correlation_df = pd.DataFrame({'Actual': targets, 'Predicted': preds})
corr = correlation_df.corr()
plt.figure(figsize=(10,6))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Plot')
plt.show()
```

4.3 Analytical/Comparative Discussion

The project gives really precise and accurate results and we can finally predict the stats of crypto with ease through these analysis.



As we can see the predicted value is rather close the actual value and is accurate enough to predict the rise or fall of the crypto market.

CHAPTER 5

5. *TESTING*

5.1 Testing Methods

We used different parameters to challenge our algorithms from time to time and to alter the predictions of our algorithm until finally we were able to achieve its highest precision. We have similarly used different quantities of data for our analysis.

5.2 Future Enhancements

We look forward to make our project near to perfect in making predictions and then after achieving that milestone we plan on further including different Crypto coins as well so that we can offer predictions for them as well.

5.3 Conclusion

We trained the 2-layers Long Short Term Memory Neural Network as well as Gated Recurrent Unit Neural Network using Bitcoin Historical Data. These models can be used to predict future price movements of bitcoin. The performance of the models is quite good.
We predicted cryptocurrency prices in real time using LSTM neural network. We went through a four step process of getting real-time cryptocurrency data, preparing data for training and testing, predicting the prices using LSTM neural network and visualizing the prediction results. Feel free to play with the hyper-parameters or try out different neural network architectures for better results.

CHAPTER 6

6. *REFERENCES*

• A great overview of LSTMs:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

• Keras LSTM tutorial:
https://keras.io/layers/recurrent/#lstm

• LSTM for time series prediction:
https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

• LSTM for text generation:
https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/

• Visualizing LSTMs:
https://distill.pub/2019/visualizing-lstm/

• LSTM vs GRU:
https://machinelearningmastery.com/lstm-vs-gru-for-sequence-prediction/

• Bidirectional LSTMs:
https://machinelearningmastery.com/bidirectional-long-short-term-memory-networks-in-

python/

• LSTM hyperparameters:
https://machinelearningmastery.com/tune-lstm-hyperparameters-keras/

• LSTM regularization techniques:
https://machinelearningmastery.com/regularization-techniques-for-training-stable-lstm-networks/

• LSTM vs CNN:
https://machinelearningmastery.com/cnn-vs-lstm-for-sequence-prediction/

• Advanced LSTM architectures:
https://machinelearningmastery.com/advanced-lstm-architectures-for-sequence-prediction

https://ieeexplore.ieee.org/abstract/document/8374483

https://medium.com/activewizards-machine-learning-company/bitcoin-price-forecasting-with-deep-learning-algorithms-eb578a2387a3

https://ieeexplore.ieee.org/document/8534911