Configuration Manual

MSc Research Project
Data Analytics

ADEBAYO J TORIOLA
Student ID: x19192118

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	ADEBAYO J TORIOLA
Student ID:	x19192118
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Jorge Basilio
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	771
Page Count:	13

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:	
Date:	15th August 2021

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):	

Prediction of Bitcoin Prices Using Deep learning and Sentiment Analysis on Bitcoin Tweets

ADEBAYO J TORIOLA
x19192118

1 Introduction

This configuration manual serves as guide to replicate this research project, it gives an overview of the hardware and software requirements used in running the codes from the data preparation to the implementation phase.
Research Study: Prediction of Prices Using Deep learning and Sentiment Analysis on Bitcoin Tweets and Bitcoin historical price data collected for hourly and per minutes records. The objective of this research is to measure the execution of the ARIMA and LSTM models on the local machine and cloud environment.

2 System Configuration

2.1 Hardware

The hardware configuration used for this research are show below, we have included the GPU as to determine if it will any significant effect on the time of execution of the code.

2.1.1 Local Machine Configuration

- Model : MacBook Air
- OS : MacOSBigSurOS
- Processor : 2.8GHz Quad-Core Apple M1 Chip
- Memory : 16GB
- Number of Core : 8
- Graphic Type : GPU

2.1.2 Cloud Environment

- Model : Google Colab
- OS : 1xTesla K80
- Processor : 2.3GHz

- Memory : 12GB GDDR5 VRAM
- Number of Core : 2496 CUDA cores
- Graphic Type : GPU

2.2 Software

The software used from the implementation are:

- Programming Language : Python
- IDE : Google Colab (Cloud Based Jupyter Notebook)
- Web Browser : Google Chrome
- Documentation : Overleaf
- Number of Core : 2496 CUDA cores
- Graphic Type : GPU

  The steps below gives details on how the Google Colaboratory environment is set up, screenshot of the step has been added to aid the replica of the experiment. To begin, a Google account is required to access the Colab environment.

  1. Sign in with the Gmail Username
  2. Import all the libraries required as captured in the coding section. Libraries that needs to be installed are:

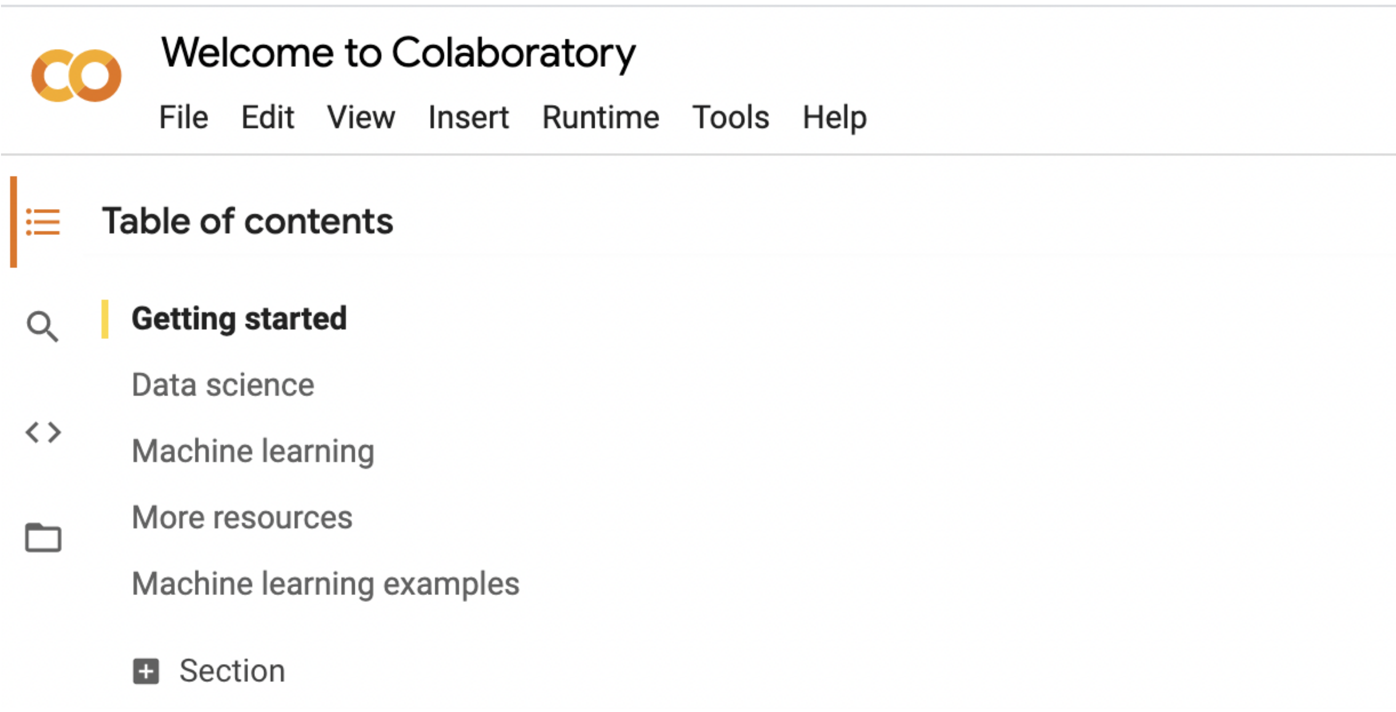

Figure 1: Sign in to Google Colaboratory

After sign in, if the cloud CPU and GPU is going to be used, then it is a must google drive be mounted. The step required that you import drive from google colab as shown below:

Other libraries needed to be imported includes:

1. Date Time
2. Pandas
3. Numpy
4. NLTK
5. Matplotlib
6. Sklearn
7. Statsmodels
8. Tensorflow
9. Keras

3 Data Preparation

This section describes the process which the dataset were uploaded to into the IDE, we have captured process in the Colab environment as shown below. Immediately the data is read into the working environment, then we commence the data preprocessing which include data cleansing and transformation. The two datasets used for the study are collected from kaggle [1] and cryptodataworld [2], this datasets provides information on the Bitcoin. The two data contain too much noise and missing value. The noise were handled using regular expression.

```
## Load the tweeets data
tweets_data = pd.read_csv('/content/drive/MyDrive/Bitcoin_tweets.csv')
```

```
##Exploratory Data analysis
```

```
# Check data summary
tweets_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800165 entries, 0 to 800164
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   user_name         800161 non-null  object
 1   user_location     424714 non-null  object
 2   user_description  515783 non-null  object
 3   user_created      360719 non-null  object
 4   user_followers    360717 non-null  float64
 5   user_friends      360717 non-null  object
 6   user_favourites   360717 non-null  object
 7   user_verified     360717 non-null  object
 8   date              360717 non-null  object
 9   text              360717 non-null  object
 10  hashtags          237474 non-null  object
 11  source            250905 non-null  object
 12  is_retweet        254219 non-null  object
dtypes: float64(1), object(12)
memory usage: 79.4+ MB
```

Figure 2: Loading Bitcoin Tweet from Drive

[1]https://www.kaggle.com/kaushiksuresh147/bitcoin-tweets
[2]https://www.cryptodatadownload.com/data/

Equalising the errors to coerce helps to load the records that are not in the correct date time format, this code also checks for null and missing values.

```
# Filtering the data with required fields
tweets_data1 = tweets_data[['date','text']]
```

```
## Select records of Non NA values
tweets_data2 = tweets_data1[~tweets_data1['date'].isnull()]
```

```
## Confirming Missing data again
tweets_data2.isnull().sum()
```

```
date     0
text     0
dtype: int64
```

```
tweets_data2 = tweets_data2.sort_values(by=['date'])

# Making Backup data
tweets_data3 = tweets_data2.copy()
```

```
# Dealing Mixed Datatypes

tweets_data3['DateTime']=pd.to_datetime(tweets_data3['date'], errors='coerce')
```

Figure 3: Date Type processing

Next is the use of regular expression to remove noise like special characters, URL, emoticons from the tweet data.

```
##text Cleansing
```

```
stops = nltk.corpus.stopwords.words("english")
```

```
def text_preprocess(text):
    text = re.sub(r'[^\w\s]', '', str(text).lower())
    text = re.sub(r'\w*\d+\w*', '', text)## getting rid of alpha numeric
    text = re.sub(r'[^a-zA-Z\s]','',text,re.I|re.A)##removing non letters
    text = re.sub('([#])|([^a-zA-Z])',' ',text)# removing the hashtag
    text = re.sub('<.*>', ' ', text) # removing the html tags
    text = re.sub('[^a-zA-Z\s]+',' ',text) # removing the punctuation
    text = re.sub('[ ]{2,}',' ',text) # removing the extra white space
    txtpost = text.split()
    #remove stopwords
    txtpost = [i for i in txtpost if i not in stops]

    tokens = " ".join(txtpost)
    return tokens
```

ve function cleanses the text by removing stopwords,unwanted text,symbols,punctuations,hashtags and extra te space.

```
# Clean data
tweets_data4['text_new'] = tweets_data4['text'].apply(lambda x: text_preprocess(x))
```

Figure 4: Data Cleaning

3.1 Data Transformation

In the next code, VADER sentiment analyser is used to calculate the polarity and intensity by assigning a positive, neutral, negative score for each tweet.

```
# VADAR implementation

from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA

sia = SIA()

# Getting polarity scores via Vader
result = tweets_data4['text_new'].apply(lambda x: sia.polarity_scores(x))

# List transformation
result1 = list(result)

#Score to  data frame
vader_sentiment = pd.DataFrame.from_records(result1)

### to Combine tweets data and sentiments
# resetting the index of both dataframes
tweets_data4 = tweets_data4.reset_index()
vader_sentiment = vader_sentiment.reset_index()

# concat both dataframes
tweets_data5 =pd.concat([tweets_data4,vader_sentiment],axis=1)
```

Figure 5: Data Cleaning

The figure below show the loading of Bitcoin price into the IDE

```
# Loading Bitcoin data
# header =1 makes 2 nd row as header as 1 st row has text.
Bitfinex_BTCUSD = pd.read_csv('/content/drive/MyDrive/Bitfinex_BTCUSD_minute.csv',header = 1)
Bitstamp_BTCUSD = pd.read_csv('/content/drive/MyDrive/Bitstamp_BTCUSD_2021_minute.csv',header = 1)
gemini_BTCUSD = pd.read_csv('/content/drive/MyDrive/gemini_BTCUSD_2021_1min.csv',header = 1)
```

```
# Renaming Date to date

gemini_BTCUSD = gemini_BTCUSD.rename({'Date':'date'},axis=1)

##sort values by date

Bitfinex_BTCUSD = Bitfinex_BTCUSD.sort_values(by=['date'])
Bitstamp_BTCUSD = Bitstamp_BTCUSD.sort_values(by=['date'])
gemini_BTCUSD = gemini_BTCUSD.sort_values(by=['date'])
```

Figure 6: Reading Bitcoin Price

3.1.1 Features Extraction

Figure below shows how the records are selection from the three exchanges and how missing records is filled with previous value.

```
##Subsetting into date between 2018-05-15 06:00:00' : '2021-07-31 00:00:00 i.e of same range

gemini_BTCUSD1 = gemini_BTCUSD.loc['2021-01-01 00:00:00' : '2021-08-12 00:09:00' ]
Bitfinex_BTCUSD1 = Bitfinex_BTCUSD.loc['2021-01-01 00:00:00' : '2021-08-12 00:09:00' ]
Bitstamp_BTCUSD1 = Bitstamp_BTCUSD.loc['2021-01-01 00:00:00' : '2021-08-12 00:09:00' ]
```

```
# Map the bitcoin data into minute level and imputing the missing values

# Frequency of 'T' makes it to minute level

gemini_BTCUSD1 = gemini_BTCUSD1.resample('T').bfill()
Bitfinex_BTCUSD1 = Bitfinex_BTCUSD1.resample('T').bfill()
Bitstamp_BTCUSD1 = Bitstamp_BTCUSD1.resample('T').bfill()
```

Figure 7: Date Selection on the Price Data

3.1.2 Merged Price Data

In this section the average of Open, High, Low and Close price were taken and merged to form a comprehensive merged dataframe called sampledf.

```
## Merging of 3 bitcoin datasets

merged = Bitstamp_BTCUSD1.merge(Bitfinex_BTCUSD1,on='date').merge(gemini_BTCUSD1,on='date')
```

```
## Aggregating the mean (variable wise) of three attributes

merged['avg_close'] = merged[['close_x','close_y','Close']].mean(axis =1)
merged['avg_open'] = merged[['open_x','open_y','Open']].mean(axis =1)
merged['avg_high'] = merged[['high_x','high_y','High']].mean(axis =1)
merged['avg_low'] = merged[['low_x','low_y','Low']].mean(axis =1)
merged['avg_volume'] = merged[['Volume USD_x','Volume USD_y','Volume']].mean(axis =1)
sampledf = merged[['date','avg_close','avg_open','avg_high','avg_low','avg_volume']]
```

Figure 8: Date Selection on the Price Data

Codes caption in figure 9 shows how the plot of the closing price for visualization in order to gain an insight of the price movement.

```
import seaborn as sns
sns.set(font_scale=2.0)
tweet_price.set_index('date')['avg_close'].plot(figsize=(30, 10), linewidth=0.8, color='maroon')
plt.xlabel("Date")
plt.ylabel("Average Close price",labelpad=15)
plt.title("Close Price Movement", fontsize=40);
```

Figure 9: Plot of the Closing price

4 Model Implementation

This section describe the executable models that were used for this research, since we are dealing with time series data ARIMA and LSTM models used as the codes for each of the models are shown in the screenshot.

4.1 ARIMA

```
from statsmodels.tsa.arima.model import ARIMA
```

e above summary shows SARIMAX results by default in smmodels package its ARIMA by default using parameters like exog and endog
ing it SARIMAX which is useful for multivariate analysis. We can also use this class from statsmodels.tsa.statespace.sarimax import
IMAX,both will be same

```
import statsmodels.api as sm
import matplotlib
from pylab import rcParams
rcParams['figure.figsize'] = 26, 10
## knowing the Trend ,seasonality,randomness and seasonality
decomposition = sm.tsa.seasonal_decompose(tweet_price1['avg_close'].values, model='additive',period = 1440)
fig = decomposition.plot()
plt.show()
```

Figure 10: import ARIMA library

The next phase of the code is splitting the dataset into train and testing for the price forecast.

```
## Making these varibales as exogenous variables for sarimax model
ts_data = tweet_price1.copy()
ts_data['diffS']=ts_data['avg_close'].diff()
ts_data['lag']=ts_data['diffS'].shift()
ts_data['diffavg_open']=ts_data['avg_open'].diff()
ts_data['lag_avg_open']=ts_data['diffavg_open'].shift()
ts_data['diffavg_high']=ts_data['avg_high'].diff()
ts_data['lag_avg_high']=ts_data['diffavg_high'].shift()
ts_data['diffavg_low']=ts_data['avg_low'].diff()
ts_data['lag_diffavg_low']=ts_data['diffavg_low'].shift()
ts_data['diffavg_volume']=ts_data['avg_volume'].diff()
ts_data['lag_diffavg_volume']=ts_data['diffavg_volume'].shift()
```

```
# Setting Date as index

ts_data = ts_data.set_index('date',drop= True)
ts_data.dropna(inplace=True)

## Train- Test Split

train = ts_data.iloc[0:199000,:]# train data
test = ts_data.iloc[199000:,:]    ## test data
predictions = []
```

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

Figure 11: Split Data for ARIMA Model

To perform the prediction on the test data in order to validate the model the code is executed.

```
steps = len(ts_data) - len(train)
print(steps)
## Prediction
predictions= results3.forecast(steps=steps,exog=test[['lag','lag_avg_open','lag_avg_high','lag_diffavg_low','lag_diffavg_v
actual = test['avg_close'].values

##Validation
from sklearn.metrics import mean_squared_error

test_set_rmse = (np.sqrt(mean_squared_error(actual, predictions)))
print(test_set_rmse)

236
241.9080445977797
```

Figure 12: ARIMA model Prediction

The code below is initiated to calculated the execution time of the model

```
#SARIMAX Model Implementation
start_time = time.time()


model3=SARIMAX(endog=train['avg_close'],exog=train[['lag','lag_avg_open','lag_avg_high','lag_diffavg_low','lag
results3=model3.fit()


end_time = time.time()
execution_time = end_time-start_time
print('Time Taken:', time.strftime("%H:%M:%S",time.gmtime(execution_time)))

Time Taken: 00:01:53
```

Figure 13: Execution Time Calculation

The code below is run in order to determine the best p,d,q for the ARIMA model.

```
%%time
# SARIMAX Model
start_time = time.time()

sxmodel = pm.auto_arima(train['avg_close'], exogenous=train[['lag','lag_avg_open','lag_avg_high','lag_diffavg_low','lag_diffavg_volume','compound']],
                           start_p = 1,start_q=1,
                           test='adf',
                           max_p=2, max_q=2, m=1,
                           start_P=0, seasonal=True,
                           d=None, trace=True,
                           error_action='ignore',
                           suppress_warnings=True,
                           stepwise=True)

end_time = time.time()
execution_time = end_time-start_time
print('Time Taken:', time.strftime("%H:%M:%S",time.gmtime(execution_time)))

Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=2233552.769, Time=114.21 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=2233529.136, Time=85.67 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=2233523.427, Time=95.05 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=2233523.519, Time=104.76 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=4462526.912, Time=45.50 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=2233518.082, Time=114.75 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=2233530.858, Time=124.69 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=2233516.098, Time=102.19 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=2233521.442, Time=85.44 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=2233528.834, Time=112.33 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=2233550.530, Time=101.65 sec

Best model:  ARIMA(2,1,0)(0,0,0)[0]
Total fit time: 1089.609 seconds
Time Taken: 00:18:09
CPU times: user 18min 34s, sys: 4min 36s, total: 23min 11s
Wall time: 18min 9s
```

Figure 14:

5 LSTM

LSTM Implementation

```
[ ] price_dataFrame = tweet_price1[['avg_open','avg_high','avg_low','avg_volume','avg_close']]
    sent_dataFrame = tweet_price1[['compound']]

    # Feature Scaling

    # price features scaling/standardization

    prices = price_dataFrame.values.reshape(-1, price_dataFrame.shape[1])
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_prices = scaler.fit_transform(prices)

    sentiment = sent_dataFrame.values.reshape(-1, sent_dataFrame.shape[1])

    # Spliting train and test sets
    train_size = int(len(scaled_prices) * 0.90)

    test_size = len(scaled_prices) - train_size

    train, test = scaled_prices[0:train_size,:], scaled_prices[train_size:len(scaled_prices),:]

    print(f'Train set size: {len(train)}, Test set size: {len(test)}')

    Train set size: 179314, Test set size: 19924
```

Figure 15: Data Split for LSTM

ting up Configuration for the model

```
## Set the seed
from numpy.random import seed
seed(1)
import tensorflow as tf
from tensorflow.python.framework.random_seed import set_random_seed
set_random_seed(2)
import os
os.environ['TF_DETERMINISTIC_OPS'] = '1'
```

```
#Create default session
sess = tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(log_device_placement=True))

Device mapping: no known devices.
```

```
from tensorflow.python.keras import backend as K
from keras import __version__
```

Figure 16: Tensorflow Session

6 Hourly Data

```python
# importing libraries

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from math import sqrt
from sklearn.metrics import mean_squared_error
```

```python
# Building model
start_time = time.time()

model = Sequential()
model.add(LSTM(100, input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(100))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
```

Figure 17: KERAS Parameters

```python
# Plotting ground truth vs Predicted

yhat_train = model.predict(trainX)

plt.figure(1)
plt.subplot(2, 1, 1)
plt.plot(trainY, label='Groundtruth', color='orange')
plt.plot(yhat_train, label='Predicted', color='purple')
plt.title("Training")
plt.ylabel("Scaled Price")
plt.legend(loc='upper left')

plt.subplot(2, 1, 2)
plt.plot(testY, label='Groundtruth', color='orange')
plt.plot(yhat_test, label='Predicted', color='purple')
plt.title("Test")
plt.ylabel("Scaled Price")
plt.legend(loc='upper left')

plt.show()
```

Figure 18: Prediction graph

7 Section 6

```
#HISTORICAL PRICE IN PER HOUR
```

```
#SETTING THE TWEET DATA FOR HOURLY BITCOIN PRICE
```

ouble-click (or enter) to edit

```
hourly_tweet_data4 = tweets_data5
```

```
#LOAD THE HOURLY PRICE
```

ouble-click (or enter) to edit

```
##Load Bitcoin datasets
```

```
Bitfinex_BTCUSD_1hr = pd.read_csv('/content/drive/MyDrive/Bitfinex_BTCUSD_1h.csv')
Bitstamp_BTCUSD_1hr = pd.read_csv('/content/drive/MyDrive/Bitstamp_BTCUSD_1h.csv')
gemini_BTCUSD_1hr = pd.read_csv('/content/drive/MyDrive/gemini_BTCUSD_1hr.csv')
```

```
len(Bitfinex_BTCUSD_1hr),len(Bitstamp_BTCUSD_1hr),len(gemini_BTCUSD_1hr)
```

```
(28145, 28147, 50937)
```

```
##Renaming date
gemini_BTCUSD_1hr.loc[:,'date']=gemini_BTCUSD_1hr.Date
```

Figure 19: Hourly Data

```
##Setting the date as index for subsetting
gemini_BTCUSD_1hr = gemini_BTCUSD_1hr.set_index(['date'],drop=False)
Bitfinex_BTCUSD_1hr = Bitfinex_BTCUSD_1hr.set_index(['date'],drop=False)
Bitstamp_BTCUSD_1hr = Bitstamp_BTCUSD_1hr.set_index(['date'],drop=False)
```

```
##Subsetting into date between 2021-01-101 01:00:00' : '2021-07-31 00:00:00 i.e of same range
gemini_BTCUSD_1hr1 = gemini_BTCUSD_1hr.loc['2021-01-01 00:00:00' : '2021-07-31 00:00:00' ]
Bitfinex_BTCUSD_1hr1 = Bitfinex_BTCUSD_1hr.loc['2021-01-01 00:00:00' : '2021-07-31 00:00:00' ]
Bitstamp_BTCUSD_1hr1 = Bitstamp_BTCUSD_1hr.loc['2021-01-01 00:00:00' : '2021-07-31 00:00:00' ]
```

```
gemini_BTCUSD_1hr1 = gemini_BTCUSD_1hr1.reset_index(drop =True)
```

```
Bitfinex_BTCUSD_1hr1 = Bitfinex_BTCUSD_1hr1.reset_index(drop =True)
```

```
Bitfinex_BTCUSD_1hr1 = Bitfinex_BTCUSD_1hr1.reset_index(drop =True)
```

✓ 0s completed at 13:58

Figure 20: Extracting Hourly Data

```
##Getting Aggregates of polarity data to hourly by resample
```

```
hourly_tweet_updtd = hourly_tweet_data6.resample('H').agg(dict(compound= 'mean',compound_norm='mean',neg='mean',pos='mean',neu='mean')).ffill()
```

```
##Re assign date as column from date index
hourly_tweet_updtd['date']=hourly_tweet_updtd.index
```

```
##reset index by dropping it
hourly_tweet_updtd = hourly_tweet_updtd.reset_index(drop = True)
```

```
hourly_tweet_updtd
```

Figure 21: Aggregating the hourly data

```
#reset the dates for merging
sampledf_hr2 = sampledf_hr1.rename_axis(None)
twt_pol_hourly1 = twt_pol_hourly .rename_axis(None)
```

```
sampledf_hr5 = sampledf_hr2.copy()
```

```
sampledf_hr5 = sampledf_hr5.reset_index(drop=True)
```

```
sampledf_hr5 = sampledf_hr5.rename_axis(None)
sampledf_hr5
```

Figure 22: Merged Hourly Dataset

```
import matplotlib.pyplot as plt
tweet_price_hr.shape
```

```
(3322, 11)
```

```
#from statsmodels.tsa.arima_model import ARIMA
#statsmodels.tsa.arima.model.ARIMA
from statsmodels.tsa.arima.model import ARIMA
```

```
from sklearn.model_selection import train_test_split

X = tweet_price_hr['avg_close'].values
train = X[0:3000]# train data
test = X[3000:]  #test data
predictions = []
```

```
steps = len(tweet_price_hr) - len(train)
print(steps)
```

```
322
```

Figure 23: Training Hourly Data

```python
# importing libraries

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from math import sqrt
from sklearn.metrics import mean_squared_error
```

```python
# Building model
start_time = time.time()

model = Sequential()
model.add(LSTM(100, input_shape=(trainX.shape[1], trainX.shape[2]), return_se
model.add(Dropout(0.2))
model.add(LSTM(100))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
```

```python
%%time
history = model.fit(trainX, trainY, epochs=100, batch_size=100, shuffle=False
end_time = time.time()
execution_time = end_time-start_time
print('Time Taken:', time.strftime("%H:%M:%S",time.gmtime(execution_time)))
```

Figure 24: Keras Library

```python
# Processing the Predictions
a = np.zeros((yhat_test_hr.shape[0], 4))
# Stacking the predictions for mapping
yhat_test_hr = np.hstack([a, yhat_test_hr])
# Inverse predictions transformations
yhat_test_hr_inverse = scaler.inverse_transform(yhat_test_hr)
```

```python
# 7 - Plot price (Inverse transform)
#yhat_test_hr_inverse = scaler.inverse_transform(yhat_test_hr)
predicted_price_hr = yhat_test_hr_inverse[:, 4]

testY = testY.reshape(-1, 1)
testY = np.hstack([a, testY])
testY_inverse = scaler.inverse_transform(testY)
real_price_hr = testY_inverse[:, 4]

plt.plot(real_price_hr, label='Actual', color='royalblue')
plt.plot(predicted_price_hr, label='Predicted', color='indianred')
plt.title("Predicted vs Actual")
plt.ylabel("Price")
plt.legend(loc='upper left')

plt.show()
```

Figure 25: Prediction on the Hourly Data