Using Twitter Sentiments to Predict Bitcoin Price Fluctuations

PROJECT SUPERVISOR

Dr. Zulfiqar Ali Memon

PROJECT CO-SUPERVISOR

Mr. Shoaib Raza

PROJECT TEAM

Muhammad Haider Zaidi	18K-0199
Muhammad Maaz Khan	18K-1077
Muhammad Ali Ansari	18K-1065

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

FAST SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES KARACHI CAMPUS

July 2022

Acknowledgement

Our project wouldn't have been possible without the support of our supervisors Dr. Zulfiqar Memon and Mr. Shoaib Raza who were always there for our support and guidance to improve our work by providing useful criticism and enthusiastic encouragement. Furthermore, we would like to express our deepest appreciation to our Jury Dr: Abdul Aziz guiding us.

Document Information

Category	Information
Customer	FAST-NUCES KHI
Project Title	Using Twitter Sentiments to Predict Bitcoin Price Fluctuations
Document Version	1.0
Author(s)	Muhammad Maaz Khan, Muhammad Haider Zaidi, Muhammad Ali Ansari
Approver(s)	Mr. Shoaib Raza OR Dr. Zulfiqar Memon
Issue Date	7th, June'22

Abbreviations

Term	Description
BTC	Bitcoin
LSTM	Long Short Term Memory (machine learning model)
NLP	Natural Language Processing
RNN	Recurrent Neural Network
VADER	Valence Aware Dictionary and sentiment Reasoner (sentiment analysis framework)
BERTH	Bidirectional Encoder Representations from Transformers (sentiment analysis framework)
XgBoost	Extreme Gradient Boosting (machine learning model)

Abstract

Bitcoin is volatile and challenging to predict. It does not have a central governing authority. It is controlled by the general public and its price is affected by socially constructed opinions. Sentiment Analysis is one of the techniques that can be used for cryptocurrency forecasting. This involves extracting sentiments from opinions of the public and other authorities. Our Work revolves around performing sentiment analysis on bitcoin related tweets, And using this for time series forecasting of bitcoin prices.

Contents

Figures

Tables

1. Introduction

1.1. Project Background

Bitcoin is a form of electronic cash with no governing financial institution which can be used for online transactions or as an exchange between any two parties. In recent years Bitcoin has taken over the world by storm. According to a study, there have been over 100 Million investors in bitcoin with 668 Million transactions up till now. Much like the stock market, different techniques are applied for forecasting so as to know when to buy or sell the cryptocurrency in order to gain maximum profit.
The cryptocurrency market has been volatile from the beginning but the last few months have been particularly a wild ride. There are numerous factors that contribute to its volatility that also makes the market challenging to predict accurately.
One of the prime factors that contribute to its volatility is that Bitcoin does not have a central governing authority and is controlled by the general public. For this reason, its price is affected by socially constructed opinions.

Primarily there are three techniques that are used to predict the market. Fundamental Analysis, Technical Analysis, and Lastly Sentiment Analysis. Using these three techniques we can have better insights for forecasting.
Sentiment analysis is a technique through which a piece of text can be analyzed to determine the sentiment behind it. It combines machine learning and natural language processing (NLP) to achieve this.
In regard to this context, Sentiments can arise from public opinion, let them be opinions of government officials, celebrities, experts, or any person that is influenced by this world.
When it comes to influencing the public Twitter has the potential to be more influential, more than Facebook; especially when it comes to the dissemination of news. Scientists are increasingly recognizing Twitter's predictive power for a wide range of events, and particularly for financial markets [1]

There are several hundreds of million bitcoin-related tweets, most of them spreading the news about how bitcoin is performing and speculations about how it will perform in the future. The idea is to use various machine learning techniques to find any correlation between public opinion and the actual price fluctuation and then actually forecasting the price fluctuation as these tweets continue to happen.

1.2. Scope

This system will assist authorities or employers in accurately predicting the graph of bitcoin or its upcoming trend.It will. It will be using technical analysis to predict the trend but the trend is also affected by tweets of several organizations or individuals therefore it will also consider their impact on the trends and will give the predicted graph accordingly.

1.3. Objective

Tweet sentiment analysis is an active field for studies in price forecasting. Using Twitter in sentiment analysis for Bitcoin has gained researchers interest in it, due to the large amount of news feeds per minute regarding Bitcoin. Our aim is to try improving over existing research works, with various machine learning and deep learning models by trying different methodologies and features. Furthermore, with the outcome of our research work we will provide a platform for Realtime bitcoin forecasting based on sentiments of tweets.

- See the impact of tweets on BTC price
- Realtime bitcoin forecasting of btc using twitter sentiments

2. Literature review

There has been a lot of research work in this domain in recent years. Famous researchers[5] in the past have proposed an electronic transaction system for a peer-to-peer network that is based on cryptographic proof instead of trust. The network is robust as it uses proof-of-work to record the public history of transactions[13]. A node can leave; again, join the network at will, and vote with CPU power. It can also express acceptance of valid blocks and rejection of invalid blocks. In [3], they used the LSTM model for sentiment analysis and mapped it with historical data, then they used Random Forest model for the predictions of Bitcoin prices for 2 days. In [1], they trained the ARIMAX and LSTM-based RNN with several combinations of financial and sentiment input features and we found that the best combination is made up of the sole BTC weighted price and tweets sentiment. From this result, we observed that not always a higher number of input features leads to better outcomes. In another research paper, the researchers observed that there is a strong correlation between the Bitcoin percentage shift and Twitter sentiment[2]. They mentioned that they removed some features from the tweets such as hashtags, Twitter users, number of tweets, and emoticons. Considering these features may also improve the results of predictions. Furthermore, another research paper mentioned that they used RNN model of machine learning for predicting the price fluctuation of Bitcoin using twitter sentiment analysis and they found also found a moderate correlation between the tweets' sentiments and Bitcoin prices[4] Stock market prediction based on public sentiments expressed on Twitter has been an intriguing field of research. The thesis of this work is to observe how well the changes in stock prices of a company, the rises and falls, are correlated with the public opinions being expressed in tweets about that company. Understanding the author's opinion from a piece of text is the objective of sentiment analysis. The present paper has employed two different textual representations, Word2vec and N-Gram, for analyzing the public sentiments in tweets[7].

3. External Interface Requirements

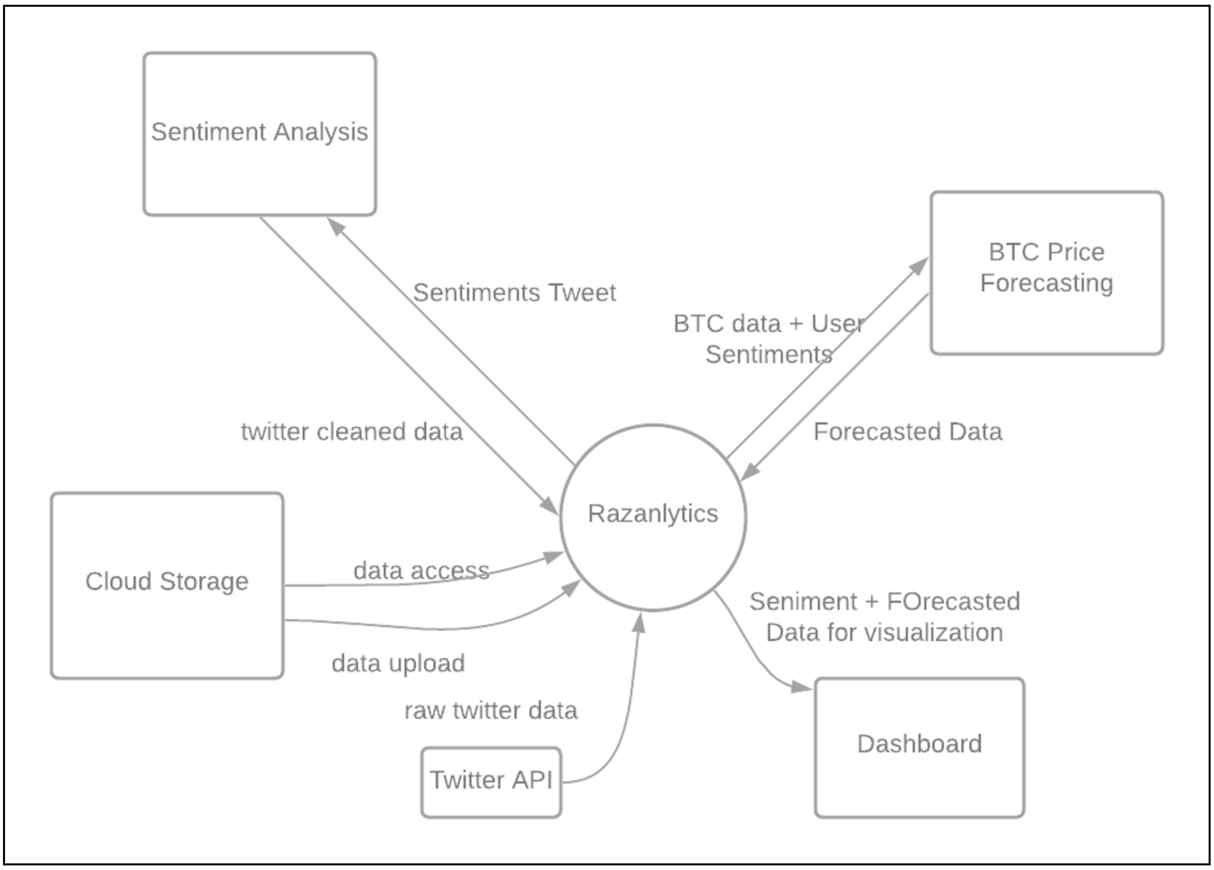

Figure 3: Context Diagram

3.1. Architecture

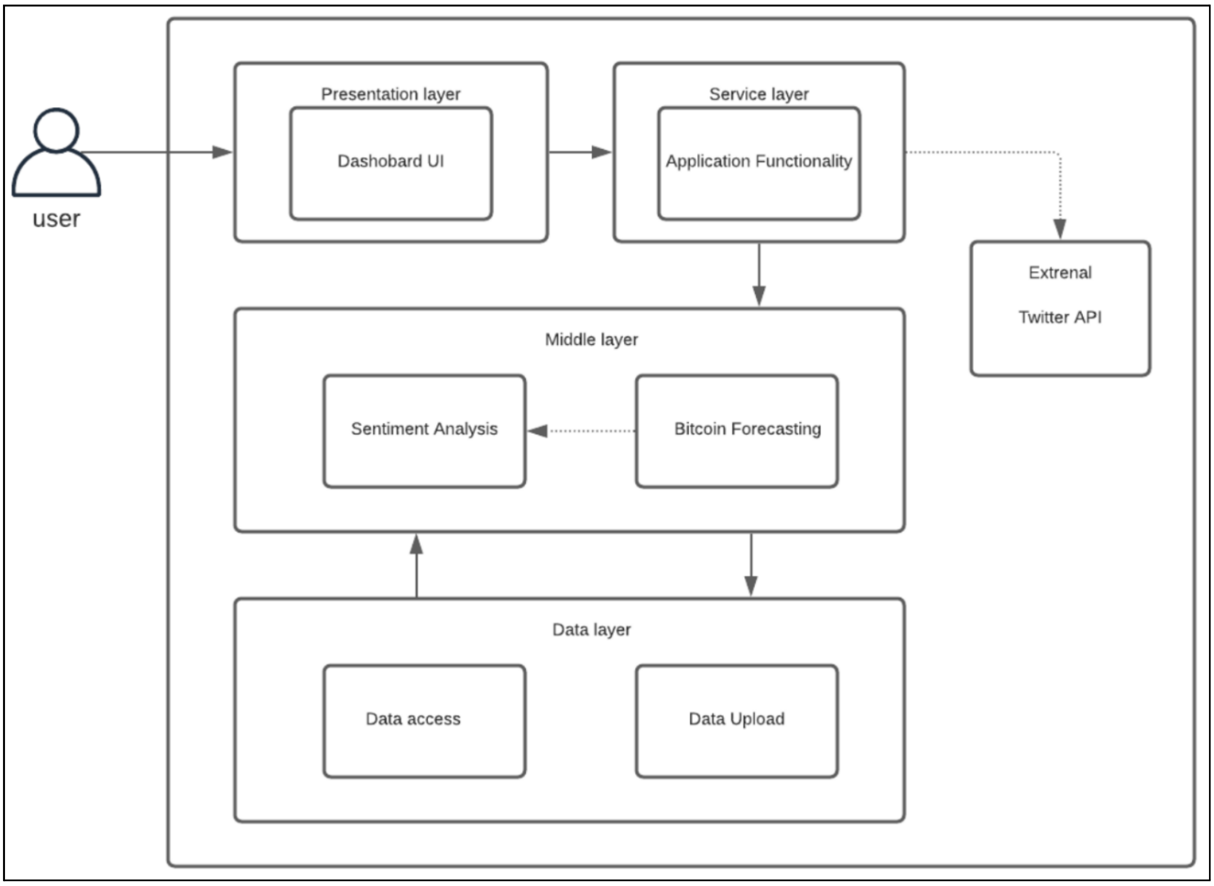

Figure 3.1: Architecture Diagram

3.2. Hardware Interfaces

- *ec2 instance aws*
- *linux server*

3.3. Software Interfaces

- apache spark(databricks)
- apache airflow.
- Streamlit Dashboard

3.4. Communications Interfaces

- *The AWS Storage will be accessed using HTTP Protocol to write or retrieve data safely through a web service. A web browser will be used to access the PowerBI Dashboard.*

3.5. System Design

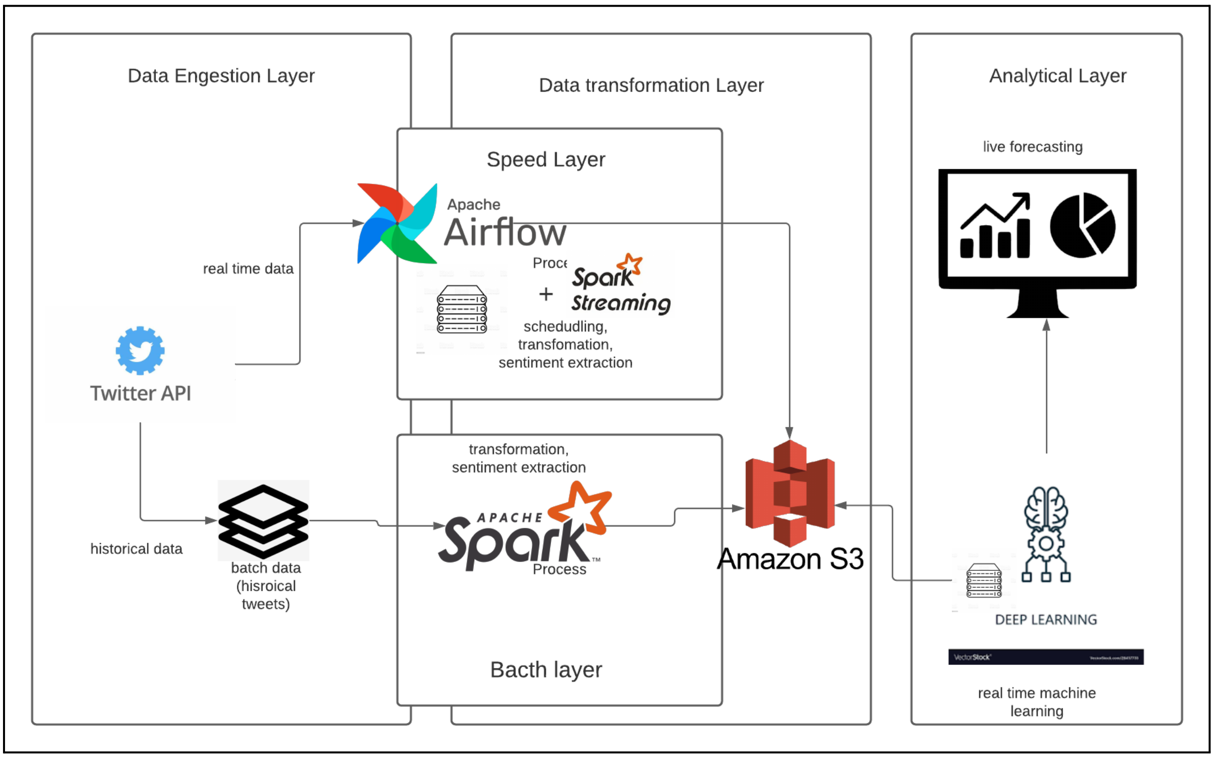

Figure 3.2: System Design Diagram

4. Requirement Analysis

4.1. Functional Requirements

4.1.1. Use Case Diagram

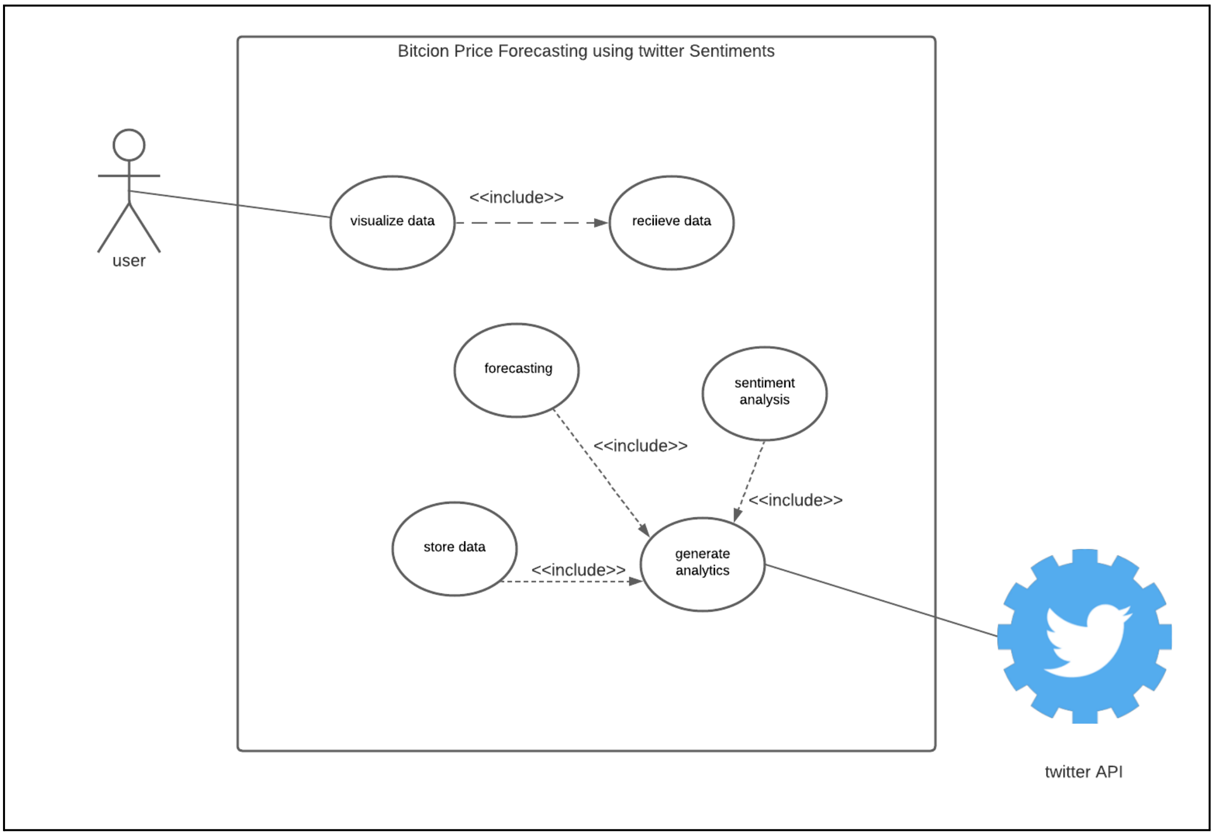

Figure 4.1: Use Case Diagram

4.1.2. Use Cases

Use case 1: data visualization		
Use case Id:	01	
Actors:	*User*	
Feature:	*visualizing the generated report*	
Pre-condition:	*NaN*	
Scenarios		
Step#	***Action***	***Description***
1.	View forecasting and sentiments	Visualize the sentiments from tweets upon a time frame, along with predicted price
2.	Receive Data	Receive data from API to generate visualization report
Use Case Cross referenced	None	
<Use case 2: generate analytics>		
Use case Id:	02	
Actors:	*twitter API*	
Feature:	*generating insights from tweets*	
Pre-condition:	*NaN*	
Scenarios		
Step#	***Action***	***Description***
1.	Data collection	On Real Time collect data on scheduled intervals, and apply transformation
2.	Sentiment analysis	Extract sentiments from tweets
3.	Bitcoin forecasting	Using the sentiment to forecast bitcoin price
Use Case Cross referenced	NaN	

Table 4.1: Use Case

4.2. Non-functional Requirements

4.2.1. Performance Requirements

The dashboard should visualize the data in less than 10 seconds.

4.2.2. Safety Requirements

The data must be relevant.

4.2.3. Security Requirements

The data collection must be secure

4.2.4. User Documentation

Since our project visualizes the data in human readable format, there is no need for user documentation.

5. Design Details

5.1. Data Dictionary

5.1.1. Twitter data

Data 1						
Name	**Twitter DATA**					
Alias	n/a					
Where-used/how-used	Used for extracting sentiments and then further training the Model for forecasting					
Content description	Raw Data from twitter API, that needs to be transformed to be in a usable condition					
Column Name	**Description**	**Type**	**Length**	**Null able**	**Default Value**	**Key Type**
user_name	*usernames of tweet author*	*string*	*var*	*No*	*Null*	*n/a*
timestamp	*time at which tweet was created*	*timestamp*	*var*	*No*	*Null*	*PK*
likes	*no. of likes of tweets*	*int*	*var*	*No*	*Null*	*n/a*
retweets	*no. of time tweet was retweeted*	*int*	*var*	*No*	*Null*	*n/a*
text	*actual text of tweet*	*string*	*var*	*No*	*Null*	*n/a*
user_verified	*user verified: yes/no*	*bool*	*var*	*No*	*Null*	*n/a*
followers_count	*no. of followers of user*	*int*	*var*	*No*	*Null*	*n/a*

Table 5.1: Twitter Data Dictionary

5.1.2. Bitcoin Data

Data 2						
Name	Bitcoin Data					
Alias	n/a					
Where-used/how-used	bitcoin price cap is used for training time series model for forecasting, how sentiments effects bitcoin price with some time margin					
Content description	data gathered from yahoo finance					
Column Name	**Description**	**Type**	**Length**	**Null able**	**Default Value**	**Key Type**
timestamp	*[Description of the column]*	*timestamp*	*[ns]*	*No*	*Null*	*PK*
bitcoin_price_cap	*[Description of the column]*	*float*	*var*	*No*	*Null*	

Table 5.2: Bitcoin Data Dictionary

5.2. State Diagram

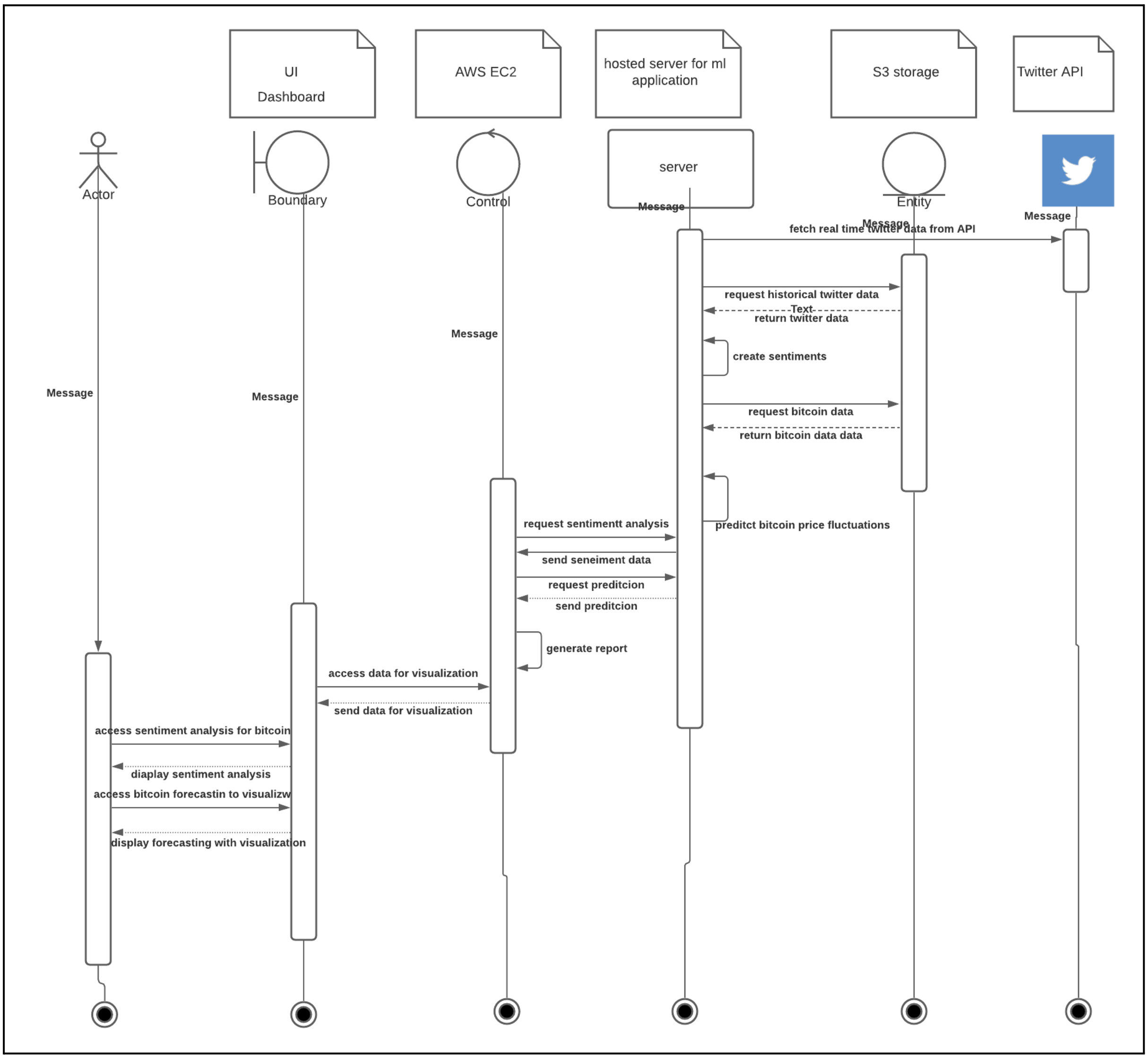

Figure 5.1: State Diagram

5.3. Sequence Diagram

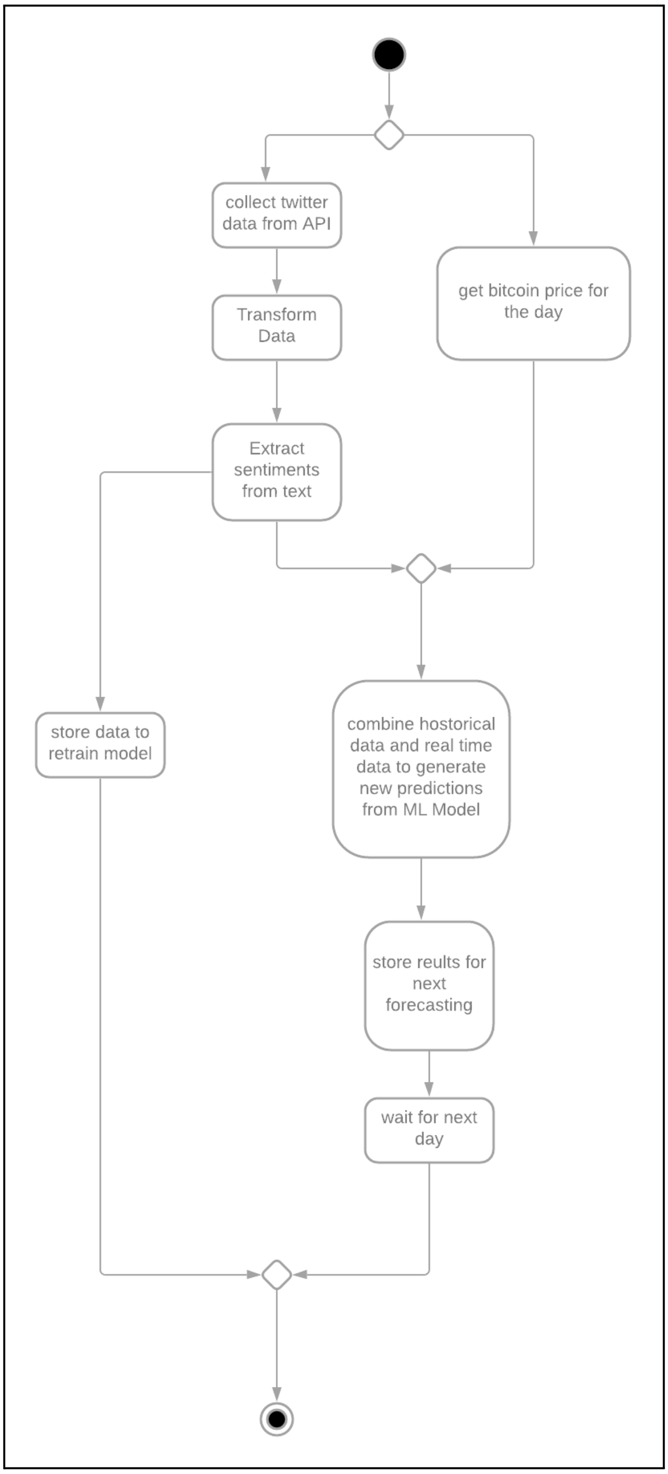

Figure 5.2: Sequence Diagram

6. Dashboard Interface

The Interface is supported by Browser. It was built using the streamlit framework.

Home page of the dashboard allows you to see forecasted results for 12 and 32 days. Predicted & current price can be viewed along with the sentiment scores.

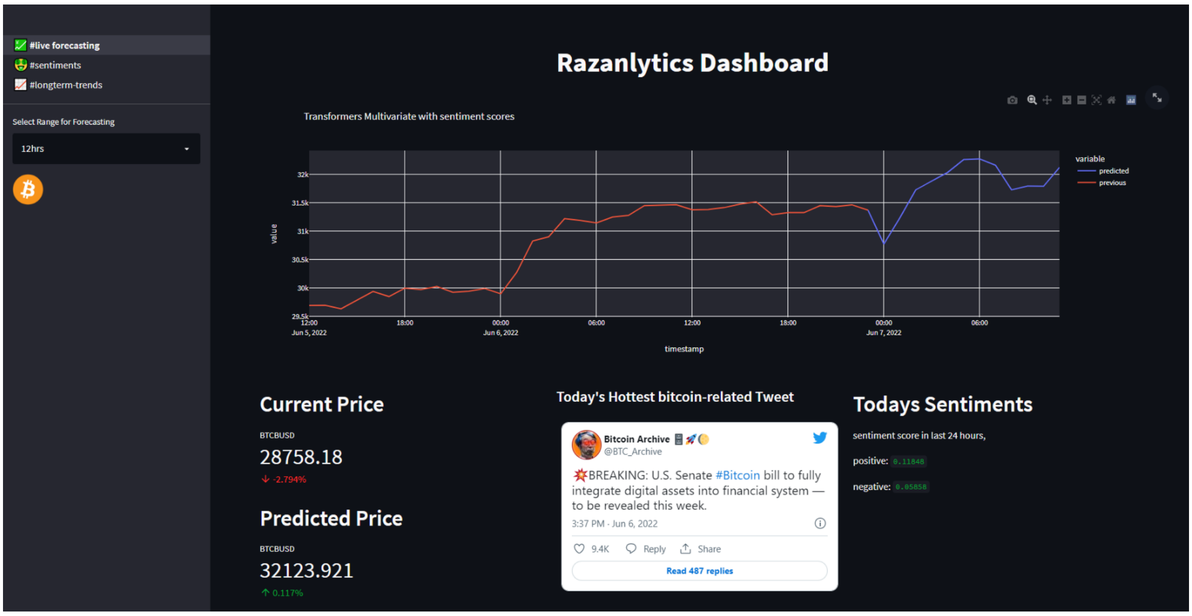

Figure 6.1: Home Screen (page1)

The second tab of the dashboard allows you to analyze the sentiment trends corresponding to its bitcoin price over the specified range.

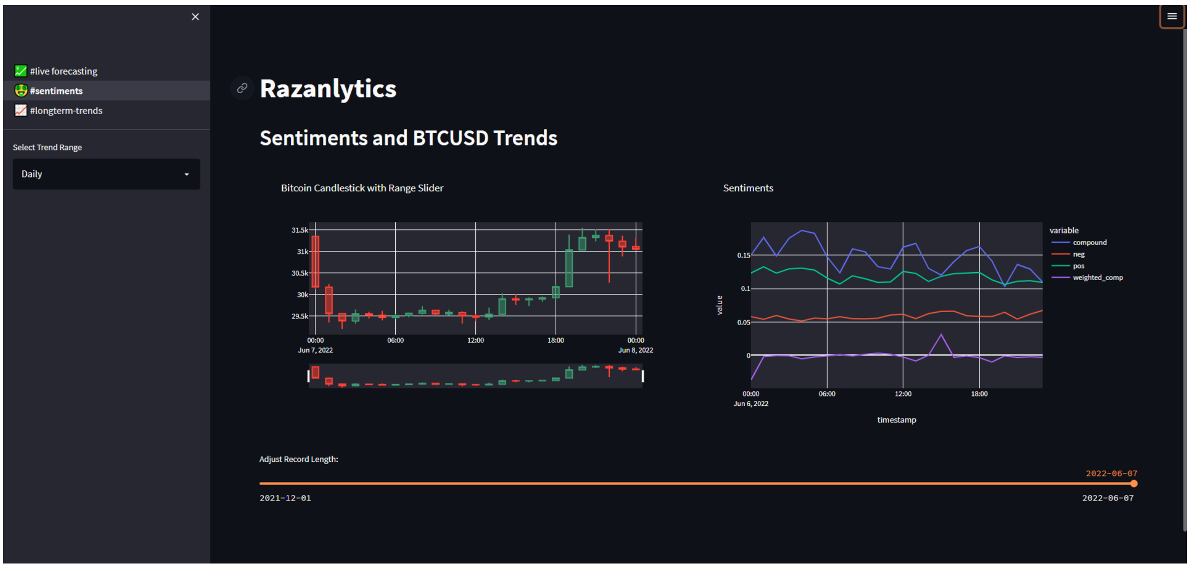

Figure 6.2: Trend Analyser (page2)

7. Methodology

Our methodology can be divided into three phases. That are as follows

7.1. Data Collection & Data Preprocessing

In this phase we designed a mechanism for collecting and storing real-time data, Our data sources were the twitter and Binance API. From the twitter API we extracted the tweets and corresponding data to it such as likes count, retweets count etc. We used Apache Airflow for scheduling of real-time data collection and data processing. While collecting tweets we applied several filters to decrease the noise in the data. These filters were based on keywords that were common in promotional tweets which were mostly from twitter-bots. We applied several data cleaning steps including: removing urls, filtering hashtags, removing digits, punctuation contractions, bringing all the text to the same case And lastly tokenization.

7.2. Sentiment Extraction And Feature Engineering

We applied various sentiment extraction techniques to see which was the most suitable. These included BERTH, TextBlob and VADER. VADER sentiment extraction was most suitable for our use case as it is specifically designed for social media texts. Vader is a lexicon based rule based sentiment analysis library. It uses a massive dictionary fabricated from millions of social media posts. And it can also work with emojis. On the other hand TextBlob was not suitable as it can not capture negative sentiments that well which is critical in our use case in order to be able to predict drops, Meanwhile BERTH is very much resource intensive and therefore not suitable to run daily on real-time tweets.

Secondly we did not apply various NLP methodologies before sentiment extraction as VADER doesn't have a requirement for that.

To make the sentiment more specific to our use case we extended the VADER dictionary with crafted sentiment score to specific keywords that had a high term document-inverse frequency score.

```
new_words = {

    'decrease': -1.5,
    'decreasing': -1.5,
    'decreased': -1.5,
    'increase': 1.5,
    'increasing': 1.5,
    'increased': 1.5,
    'rocket': 1.5,
    'rocketed': 1.5,
    'fire': 1.5,
    'bull':2.0,
    'bulls':2.0,
    'bullish':2.0,
    'bear':-2.0,
    'bears':-2.0,
    'bearish':-2.0,
    'drop':-3.0,
    'dropped':-3.0,
    'droping':-3.0,
    'low':-2.5,
    'lower':-2.5,
    'lowest':-3.5,
    'dip':-2.5,
    'diped':-2.5,
    'crash': -3.5,
    'crashed': -3.5,
    'crashing': -3.5,
    'up': 1.7,
    'down': -1.8,
    'peak': 2.5,
    'peaked': 2.5
    #'hit':-1.5
}
```

Figure 7.1: VADER personalized Sentiment Scores

We calculated a weighted sentiment score against each tweet based on its popularity. For this purpose we used the following equations.

$$compound_w = \frac{1}{1 + e^{-(compound*|likes| + compound*|lretweets| + compound)}}$$

$$negative_w = \frac{1}{1 + e^{-(negative*|likes| + negative*|lretweets| + negative)}}$$

$$postive_w = \frac{1}{1 + e^{-(postive*|likes| + postive*|lretweets| + poitive)}}$$

Equation 7.1: Weighted score

After aggregating our data on an hourly basis. We evaluated the correlation between the bitcoin prices and the rest of the features using different techniques to see which shows the highest correlation. In “open_methode_1” we used the product of popularity attributes scaled and the sentiment value to check the correlation with open price, Similarly in “open_methode_1” we used *Equation 6.1* without the bias value added. For the rest of the correlations we used *Equation 6.1* with different hourly shifts to see bitcoin prices are affected from sentiments over time.

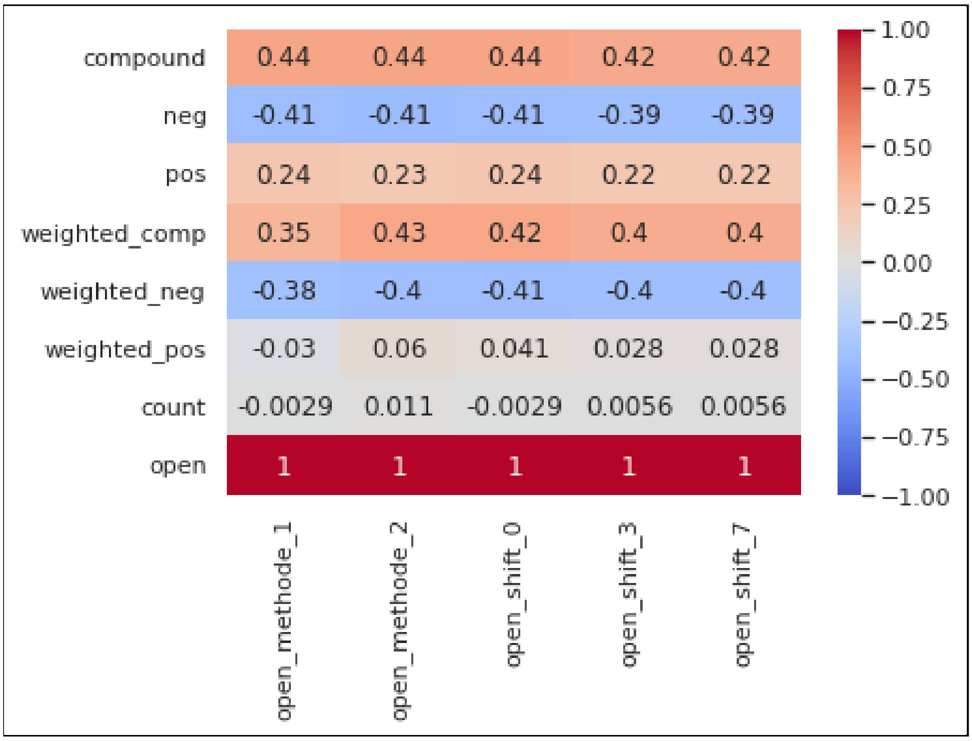

Figure 7.2: Correlation Dataframe

From the above correlation we can conclude that, the effect of sentiments on the bitcoin price are highest within the 1 hour time frame, And negative as well as compound and weighted compound have a clear effect on the bitcoin prices. We used the methodologies used to obtain features for “open_shift_0” to be used in our ML models.

7.3. Machine Learning Models

7.3.1. LSTM

LSTM is one the mature techniques in machine learning for solving forecasting problems, we also tested the LSTM deep learning model for forecasting the price of bitcoin. LSTM stands for Long Short Term Memory. As its name suggests, this model understands the long term trends and is great for forecasting

problems. This model works great with sequential data where the past data or history is an important factor for future or upcoming data. With our bitcoin and sentiment data it takes the input of 46 data points to predict every 12 points.

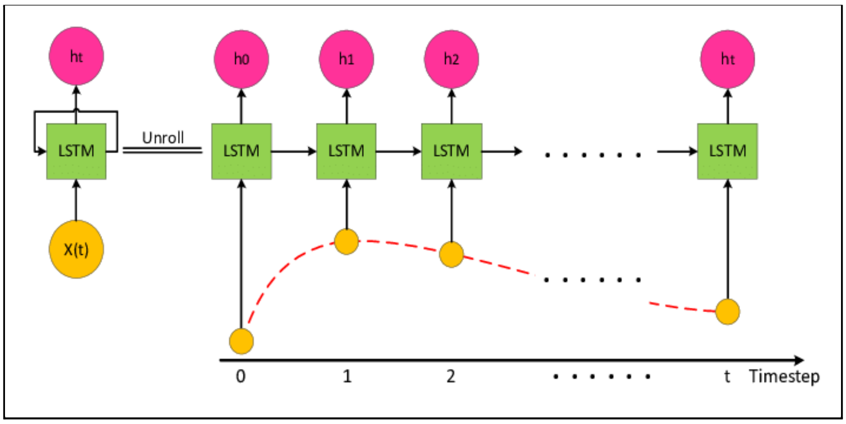

Figure 7.3: LSTM Architecture

7.3.2. XGBoost

Gradient boosting is a decision-tree-based ensemble Machine Learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of multiple models using a gradient boosting mechanism. The Univariate LSTM and predicted sentiment scores were fed into XgBoost Algorithm which helped in improving the existing results from the multivariate LSTM model.

7.3.3. Transformers

Transformer is a state-of-the-art deep learning model introduced in 2017. It is an encoder-decoder architecture whose core feature is the ‘multi-head attention’ mechanism, which is able to draw intra-dependencies within the input vector and within the output vector (‘self-attention’) as well as inter-dependencies between input and output vectors (‘encoder-decoder attention’). Which makes it tremendously-

successful in learning complex trends in bitcoin price fluctuations. The multi-head attention mechanism is highly parallelizable, which makes the transformer architecture very suitable to be trained with GPUs. Every 24 hours of the data points were used to predict the next 12 hours of data points.

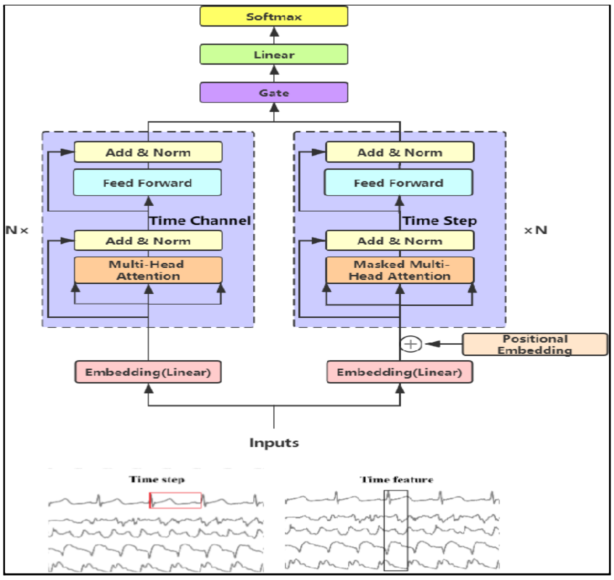

Figure 7.4: Transformer Architecture

8. Results

Inorder to train our Models we used historical data from the time period of 2018 - 2021 and for validation we took the month of April and May in 2022. For evaluating our model performance we used mean square error, which is an accuracy measure for time series model give as,

$$MSE = \frac{1}{n} \Sigma \underbrace{\left(y - \widehat{y} \right)^2}_{\text{The square of the difference between actual and predicted}}$$

Equation 7.2: Mean Squared Error

8.1. LSTM

Mean Squared Error is 7% with using sentiments in multivariate, 11% without sentiments in univariate.

Figure 7.3: Forecasting using LSTM Multivariate

8.2. XGBoost

Mean Squared Error is 5% with using sentiments in multivariate.

Figure 7.3: Forecasting using XgBoosting Multivariate

8.3. Transformers

Mean Squared Error is 2% with using sentiments in multivariate. And 7% using univariate without sentiment scores.

Figure 7.3: Forecasting using Transformer Multivariate

9. Conclusion

Each of the three models showed a significant improvement in results when we used the model with sentiment scores. The sentiment scores helped in predicting sudden drops and rises where the general trend does not hold in the series. In terms of model comparison the Transformers model outperformed former models. Transformers gain an advantage due to its attention mechanism. In RNNs such as LSTM often the bottleneck issue is found where it's difficult to hold all the information in long term trends, due to its sequential nature. In transformers; however, this issue is resolved by giving attention to relevant trends all together.

10. Code Snippets

```python
from airflow import DAG
from airflow.operators.python import PythonOperator, BranchPythonOperator
from airflow.operators.bash import BashOperator
from datetime import datetime,timedelta

import tweepy
import boto3
import pandas as pd

AWSAccessKeyId=''
AWSSecretKey=''

region='ap-south-1'


s3 = boto3.resource(
    service_name='s3',
    region_name=region,
    aws_access_key_id=AWSAccessKeyId,
    aws_secret_access_key=AWSSecretKey
)




def twitter_data_collection():

    global s3

    baerer_token = ""

    api_key = ""
    api_secret = ""
    access_token = ""
    access_token_secret = ""

    # Creating the authentication object
    auth = tweepy.OAuthHandler(api_key, api_secret)
    # Setting your access token and secret
    auth.set_access_token(access_token, access_token_secret)
    # Creating the API object while passing in auth information
    api = tweepy.API(auth,wait_on_rate_limit = True)
    with open('test.txt','w') as file:
        file.write('test')

    with open('~/variable.txt','r') as file:
        curr_time = file.read()

    search_term = '(bitcoin OR btc OR #bitcoin) -giveaway -filter:retweets -filter:replies'
    tweets = tweepy.Cursor(api.search_tweets, q = search_term, lang = 'en',until = curr_time, tweet_mode = 'extended').items(50)

```

figure continue…

```python
    with open(f"tweets_2021-11-30.csv",'a', encoding='utf-8_sig',newline='') as csvfile: #using utf-8_sig instead of utf-8 for valid encodding of special chars
      csvwriter = csv.writer(csvfile)
      csvwriter.writerow(['user_name','timestamp','likes','retweets','text','user_verified','followers_count'])
      for tweet in tweets:
        csvwriter.writerow([tweet.user.screen_name,tweet.created_at,tweet.favorite_count,tweet.retweet_count,tweet.full_text, tweet.user.verified, tweet.user.followers_count])



    s3.Bucket('bitcoin-tweets').upload_file(Filename='tweets_2021-11-30.csv', Key='tweets_2021-11-30.csv')

    curr_time = [ int(val) for val in curr_time]

    date = datetime(curr_time)
    date +=timedelta(days=1)

    s = str(date)[:10]

    with open('~/variable.txt','w') as file:
        file.write(s)


    return 'Whatever you return gets printed in the logs'



 ##################################################

WORkFLOW_DAG_ID = 'twitter_data_dag'
WORKFLOW_START_DATE = datetime.combine(
   datetime.today(),
   datetime.min.time()) # yesterday

WORKFLOW_SCHEDULE_INTERVAL = '30 8 * * *'

WORKFLOW_DEFAULT_ARGS = {
 'start_date': WORKFLOW_START_DATE,
 'email_on_failure': True,
 'email_on_retry':True,
 'retries': 5,
 'retry_delay': timedelta(minutes=5)
}

dag = DAG(dag_id = WORkFLOW_DAG_ID,
          start_date = WORKFLOW_START_DATE,
          schedule_interval = WORKFLOW_SCHEDULE_INTERVAL,
          default_args = WORKFLOW_DEFAULT_ARGS,
          catchup = False
  )


run_this = PythonOperator(
    task_id='twitter_data_collection',
    python_callable=twitter_data_collection,
  dag = dag
)

run_this
```

user	text	score
workwithai	web design osr recruitment lowestoft united kingdom 📋 more info ai aijobs artificialintelligence php jobs hiring careers lowestoft united kingdom bitcoin eth crypto	(0.5, 0.5)
crypto__mak	'master' of alternative investments does not have a clue about bitcoin bitcoinanalysis cryptocurrency blackstone cryptocurrencynews	(0.0, 0.0)
Maclovin6618	what if the baby had a shot gunsay bitcoinkatkatbooommmripple😂😂	(0.0, 0.0)
pemilijan	buysell altcoin changes with up to x leverage at primexbt 🤑💰 join right away and convert your into ✅ ✅ receive money even if btc is falling 📉📉 gto storj cro ltc aion gnt xem fct nex wan	(0.042857142857142844, 0.34285714285714286)
ttcsalam93	drife ieo crypto blockchain ethereum bitcoin ether cryptocurrency tokensale	(0.0, 0.0)

Figure 10.1: Data Extraction & Cleaning Code

```python
import pyspark
import pandas as pd

from pyspark.sql.types import *
from pyspark.sql.functions import udf,col

from whatthelang import WhatTheLang
import contractions
import nltk

nltk.download('stopwords')


dbutils.fs.cp("/FileStore/bitcoin-tweets-2016-2019/tweets.csv", # **learn reading data directly from dbfs
             "file:/databricks/driver/tweets.csv")

df = pd.read_csv('tweets.csv', delimiter=';', skiprows=0, lineterminator='\n' )
df = df.loc[:,["text"]]
```

```python
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")
df = spark.createDataFrame(df)

df.persist(pyspark.StorageLevel.MEMORY_AND_DISK)
df.repartition(8)
```

```python
def get_lang(s:str)-> str:

    return WhatTheLang().predict_lang(s)

get_lang('hello how are you')

predict_lang = udf(lambda z: get_lang(z),StringType())
spark.udf.register("predict_lang", predict_lang)


df = df.withColumnRenamed('text\r','text')\
  .withColumn('lang',predict_lang('text')) \
  .where(col('lang') == 'en') \
  .drop('lang') \
  .drop('id') \
  .drop('url') \
  .drop('fullname')
```

```python
  .drop('fullname')

#df.filter(col("text").rlike("(?i)^*follow$|(?i)^*subscribe$")).show(truncate=False)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
/databricks/python/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3165: DtypeWarning: Columns (3) have mixed types.Specify dtype option
on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

```python
import regex
import string

def transform(text:str)->str:

  #Convert to lower case
  text = text.lower()
  #Convert www.* or https?://* to URL
  text = regex.sub('((www\.[^\s]+)|(https?://[^\s]+))',' ',text)
  #Remove @username
  text = regex.sub('@[^\s]+',' ',text)
  #Remove contractions
  text = ' '.join([contractions.fix(word) for word in text.split()])
  #Remove Punctuations and Numbers
  text = ''.join([i for i in text if ( not i.isdigit() and i not in string.punctuation)])

  return text


get_transform = udf(lambda z: transform(z),StringType())
spark.udf.register("get_transform", get_transform)

df = df.withColumn('text',get_transform('text'))
df.display()
```

figure continue…

TEXT BLOB

```
import regex
import string
import contractions

def transform(text:str)->str:

  #Convert to lower case
  text = text.lower()
  #Convert www.* or https?://* to URL
  text = regex.sub('((www\.[^\s]+)|(https?://[^\s]+))',' ',text)
  #Remove @username
  text = regex.sub('@[^\s]+',' ',text)
  #Remove contractions
  text = ' '.join([contractions.fix(word) for word in text.split()])
  #Remove Punctuations and Numbers
  text = ''.join([i for i in text if ( not i.isdigit() and i not in string.punctuation)])

  return text


df['text'] = df['text'].apply(lambda x: transform(x))
```

```
text = ''

for val in df['text']:
  text += ' ' + val
```

```
# Import package
import matplotlib.pyplot as plt
# Define a function to plot word cloud
def plot_cloud(wordcloud):
    # Set figure size
    plt.figure(figsize=(40, 30))
    # Display image
    plt.imshow(wordcloud)
    # No axis details
    plt.axis("off");
```

```
from wordcloud import WordCloud, STOPWORDS
# Generate word cloud
wordcloud = WordCloud(width= 3000, height = 2000, random_state=1, background_color='black', colormap='Pastel1', collocations=False, stopwords = STOPWO
# Plot
plot_cloud(wordcloud)
```

figure continue…

VADER

adding words to vader dicionary

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import emoji
```

```
analyser = SentimentIntensityAnalyzer()

new_words = {

    'decrease': -1.5,
    'decreasing': -1.5,
    'decreased': -1.5,
    'increase': 1.5,
    'increasing': 1.5,
    'increased': 1.5,
    'rocket': 1.5,
    'rocketed': 1.5,
    'fire': 1.5,
    'bull':2.0,
    'bulls':2.0,
    'bullish':2.0,
    'bear':-2.0,
    'bears':-2.0,
    'bearish':-2.0,
    'drop':-3.0,
    'dropped':-3.0,
    'droping':-3.0,
    'low':-2.5,
    'lower':-2.5,
    'lowest':-3.5,
    'dip':-2.5,
    'diped':-2.5,
    'crash': -3.5,
    'crashed': -3.5,
    'crashing': -3.5,
    'up': 1.7,
    'down': -1.8,
    'peak': 2.5,
    'peaked': 2.5
    #'hit':-1.5
}
```

```
analyser.lexicon.update(new_words)
```

```
def apply_vader(row):
  score = analyser.polarity_scores(row['text'])
  return pd.Series([score['compound'], score['neg'], score['neu'], score['pos']])

df[['compound', 'neg', 'neu', 'pos']] = df.apply(apply_vader ,axis=1,)
```

```
def remove_stopword(x):
    x = x.split()
    return [y for y in x if y not in stopwords.words('english')]

df['text'] = df['text'].apply(lambda x:remove_stopword(x))
```

```
top = Counter([item for sublist in df.loc[df['compound']>=0.2]['text'] for item in sublist])
temp = pd.DataFrame(top.most_common(1000))
temp.columns = ['Common_words','count']
temp.style.background_gradient(cmap='Blues')
```

```
df.sort_values(by=['compound'], ascending=False)
```

```
#redefining sentiments to emojis
```

```
#create a domain specfic dictionary for vader
```

figure continue…

	compound	neg	pos	weighted_comp	weighted_neg	weighted_pos	count	open
time								
2021-11-30 02:00:00+00:00	0.238187	0.041932	0.129963	0.593775	0.535233	0.584588	1920.0	57342.23
2021-11-30 03:00:00+00:00	0.195387	0.047686	0.118205	0.581798	0.533827	0.574934	1895.0	57422.98
2021-11-30 04:00:00+00:00	0.229451	0.043506	0.122816	0.604458	0.529832	0.580733	1858.0	57149.33
2021-11-30 05:00:00+00:00	0.180722	0.053008	0.118401	0.578368	0.534701	0.574109	2036.0	57258.68
2021-11-30 06:00:00+00:00	0.210184	0.049462	0.125099	0.584154	0.530428	0.571678	2266.0	56300.00
...	...	...	...	...	...	...	...	...
2022-06-06 19:00:00+00:00	0.141527	0.058153	0.113233	0.561899	0.537099	0.570748	2338.0	31328.19
2022-06-06 20:00:00+00:00	0.103024	0.064454	0.106370	0.550823	0.537045	0.566685	2346.0	31451.46
2022-06-06 21:00:00+00:00	0.136252	0.054313	0.111042	0.562962	0.534756	0.569828	2028.0	31434.53
2022-06-06 22:00:00+00:00	0.129417	0.061523	0.111838	0.558216	0.536731	0.571340	1865.0	31466.01
2022-06-06 23:00:00+00:00	0.109691	0.067343	0.109399	0.550167	0.537689	0.565390	1709.0	31373.46

4534 rows × 8 columns

Figure 10.6: Sentiment Extraction Code

```
import pandas as pd
import requests

import pandas_datareader.data as pdr
from datetime import datetime

import plotly.offline as py
import plotly.graph_objs as go


import matplotlib.pyplot as plt
import seaborn as sns

"""start = datetime(2018,2,19)
end = datetime(2021,6,1)
df = pdr.DataReader('BTC-USD','yahoo',start,end)

df.head()"""

###########
dft = pd.read_csv('/content/drive/MyDrive/fyp/sentiments/btc.csv',skiprows=1)
dft['date'] = pd.to_datetime(dft.date)

df = dft[(dft['date'] >= init_datetime) & (dft['date'] < fin_datetime)]


data = [go.Candlestick(x=df.date,open=df.open,high=df.high,low=df.low,close=df.close)]
layout = go.Layout(title='Bitcoin Candlestick with Range Slider',xaxis={'rangeslider':{'visible':True}})

fig = go.Figure(data=data,layout=layout)



##############

dfs = pd.read_csv(f'/content/drive/MyDrive/fyp/sentiments/tweets_{init_datetime}.csv.csv')

dfs['timestamp'] = pd.to_datetime(dfs.timestamp)
dfs = dfs.resample('H', on='timestamp').agg({'compound':'mean'})
```

```
fig.show(renderer="colab")
```

```
plt.figure(figsize = (15,8))
sns.lineplot(x = 'timestamp', y = 'compound',data = dfs)
```

Figure 10.2:Sentiment Analysis Code

Checking effect of shift

```
[ ]
SHIFT = 4
df_shift = df.copy()
df_shift.open = df_shift.open.shift(SHIFT)
df_shift = df_shift[SHIFT:]
df_shift
```

Checking correlation

```
[ ]
scaler = MinMaxScaler()
scaled = scaler.fit_transform(df)
column_names = ["compound", "neg",  "pos",  "weighted_comp",  "weighted_neg", "weighted_pos", "count",  "open"]
df_scaled = pd.DataFrame(scaled)
df_scaled.columns = column_names

df_corr = df_scaled.corr(method="pearson")

df_corrP = pd.DataFrame(df_corr["open"].sort_values(ascending=False))
df_corrP
```

```
df_corr['open_methode_2']  = df_corrP['open_methode_2']
```

```
ax = sns.heatmap(df_corrF, annot=True,vmin=-1, cmap="coolwarm"  )
```

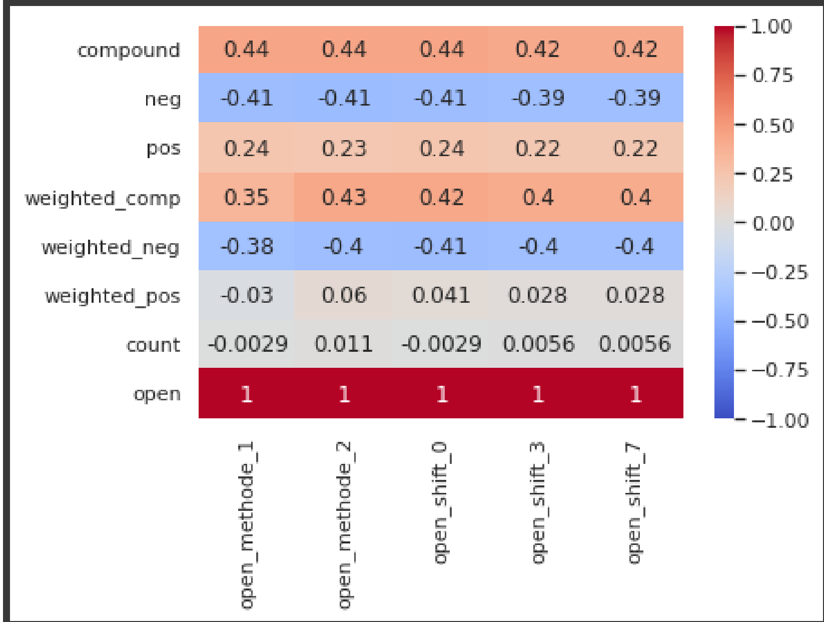

Figure 10.3:Correlation Code

```
import xgboost
from math import sqrt
import numpy as np
from matplotlib import pyplot
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential, load_model
from keras.layers import Dense, LSTM, Dropout
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.optimizers import gradient_descent_v2
# from tensorflow.keras.optimizers import SGD
from keras import activations
from sklearn.metrics import r2_score
import datetime
import os
import seaborn as sns

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
  n_vars = 1 if type(data) is list else data.shape[1]
  df = pd.DataFrame(data);
  # print(df);
  cols, names = list(), list();
  # input sequence (t-n, ... t-1)
  for i in range(n_in, 0, -1):
    cols.append(df.shift(i));
    names += [(f'var{j+1}(t-{i})') for j in range(n_vars)];
  # forecast sequence (t, t+1, ... t+n)
  df = df[10]
  for i in range(0, n_out):
    cols.append(df.shift(-i))
    if i == 0:
      names.append(f'var1(t)')
      # names += [(f'var{j+1}(t)') for j in range(n_vars)]
    else:
      names.append(f'var1(t+{i})')
      # names += [(f'var{j+1}(t+{i})') for j in range(n_vars)];
  # put it all together
  agg = pd.concat(cols, axis=1);
  print(names)
```

figure continue…

```
return agg

+ Code  + Text

df_merged_initial = pd.read_csv('/content/drive/MyDrive/btc_tweet_sentiment/btc-sent-f4.csv')

df_merged_initial.time = pd.to_datetime(df_merged_initial.time)

df_merged_initial.isnull().values.any()

lse

df_merged_initial[df_merged_initial.index.duplicated]

time  compound  neg  pos  weighted_comp  weighted_neg  weighted_pos  count  open

df_merged_initial = df_merged_initial.set_index('time')
df_merged_initial = df_merged_initial.sort_index()

pd.date_range(start = '2021-11-30 02:00:00+00:00', end = '2022-05-30 01:00:00+00:00', freq='1h' ).difference(df_merged_initial.index)

tetimeIndex([], dtype='datetime64[ns, UTC]', freq=None)

df_merged_initial = df_merged_initial.astype('float')

df_merged_initial
```

```
# df_tr = df_merged_initial.drop(["count",], axis=1)
df_tr = df_merged_initial.drop(["count",'compound', 'neg', 'weighted_neg', 'weighted_comp', 'weighted_pos', 'pos',], axis=1)

df_tr["month"] = df_tr.index.month
df_tr["day_of_week"] = df_tr.index.day_of_week
df_tr["day_of_month"] = df_tr.index.day
# dict_days = {0:"1_Mon", 1:"2_Tue", 2:"3_Wed", 3:"4_Thu", 4:"5_Fri", 5:"6_Sat", 6:"7_Sun"}
# df_tr["weekday"] = df_tr["wday"].apply(lambda x: dict_days[x])


df_tr["hour"] = df_tr.index.hour

df_tr = df_tr.astype({"hour":"category", "day_of_week":"category", "day_of_month": "category", "month": "category"})

# df_tr = df_tr[['month', 'day_of_month', 'day_of_week', 'hour', 'open']]
df_tr = df_tr[['compound', 'neg', 'weighted_neg', 'weighted_comp', 'weighted_pos', 'pos', 'month', 'day_of_month', 'day_of_week', 'hour', 'open']]

values = df_tr.values

values = values.astype('float32')

values

array([[2.3035483e-01, 4.4723179e-02, 5.2190965e-01, ... 1.0000000e+00
```

figure continue…

```
n_hours = 6
n_features = 11
n_future_hours = 1
# frame as supervised learning
reframed = series_to_supervised(values, n_hours, n_future_hours)
```

```
['var1(t-6)', 'var2(t-6)', 'var3(t-6)', 'var4(t-6)', 'var5(t-6)', 'var6(t-6)', 'var7(t-6)', 'var8(t-6)',
```

```
values = reframed.values
n_train_hours =  3514
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
```

```
test_hours = df_merged_initial.index[n_train_hours:]
test_hours = test_hours[n_hours:]
```

```
n_obs = n_hours * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_future_hours:]
test_X, test_y = test[:, :n_obs], test[:, -n_future_hours:]
print(train_X.shape, test_X.shape, train_y.shape, test_y.shape)
```

```
(3514, 66) (872, 66) (3514, 1) (872, 1)
```

figure continue…

```
reg = xgboost.XGBRegressor(objective='reg:squarederror', n_estimators=1000, nthread=24)
reg.fit(train_X, train_y)
```

```
XGBRegressor(n_estimators=1000, nthread=24, objective='reg:squarederror')
```

```
model = Sequential()
model.add(LSTM(256, activation=activations.relu, input_shape=(n_hours, n_features), return_sequences=True))
model.add(LSTM(128, activation=activations.relu, return_sequences=True))
model.add(LSTM(64, activation=activations.relu, return_sequences=False))
model.add(Dropout(0.1))
model.add(Dense(n_future_hours))
```

```
opt = gradient_descent_v2.SGD(learning_rate=0.1, momentum=0.8)
model.compile(loss='mae', optimizer=opt, metrics=['mse'])

checkpoint_path = "new_data_model3_xg_only_price.hdf5"
checkpoint_dir = os.path.dirname(checkpoint_path)

checkpoint = ModelCheckpoint(filepath=checkpoint_path,
                             monitor='val_loss',
                             verbose=1,
                             save_best_only=True,
                             mode='min')

earlystopping = EarlyStopping(monitor='val_loss', patience=10)

callbacks = [checkpoint, earlystopping]
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_33 (LSTM)              (None, 6, 256)            268288

 lstm_34 (LSTM)              (None, 6, 128)            197120

 lstm_35 (LSTM)              (None, 64)                49408

 dropout_11 (Dropout)        (None, 64)                0
```

```
import plotly.express as px


df_res = pd.DataFrame()

df_res['actual'] = [x[0] for x in actual[:24*15, 0:1]]
df_res['predicted'] = [x[0] for x in predicted[:24*15, 0:1]]
df_res['date'] = [x for x in test_hours[:24*15]]


fig = px.line(df_res, x="date", y=['actual','predicted'],
              hover_data={"date": "|%B %d, %Y"},
              title='Transformer Multivariate with sentiments')

fig.show()
```

Figure 10.4: xgboost and lstm forecasting Code

```

from darts.utils.timeseries_generation import gaussian_timeseries, linear_timeseries, sine_
from darts.models import TransformerModel
from darts.metrics import mape, smape

from darts.dataprocessing.transformers import Scaler
from darts import TimeSeries

import pandas as pd
from darts import TimeSeries
```

```


#df = pd.read_csv('/content/drive/MyDrive/btc_tweet_sentiment/btc-sent-new.csv')

df = pd.read_csv('/content/drive/MyDrive/btc_tweet_sentiment/btc-sent-f2.csv')

df.time = pd.to_datetime(df.time)
df = df.set_index('time')
df = df.sort_index()
df = df.astype('float')

df['timestamp'] = df.index.tz_localize(None)
```

```
df.tail()
```

figure continue…

```
series_neg = TimeSeries.from_dataframe(df, 'timestamp', 'weighted_comp')

neg_scaled = Scaler()
series_neg_scaled = neg_scaled.fit_transform(series_neg)

neg_train, neg_val = series_neg_scaled[:-1000], series_neg_scaled[-1000:]
```

```
series_open = TimeSeries.from_dataframe(df, 'timestamp', 'open')

open_scaled = Scaler()
series_open_scaled = open_scaled.fit_transform(series_open)

open_train, open_val = series_open_scaled[:-1000], series_open_scaled[-1000:]
```

```
series_compound = TimeSeries.from_dataframe(df, 'timestamp', 'compound')

compound_scaled = Scaler()
series_compound_scaled = compound_scaled.fit_transform(series_compound)

compound_train, compound_val = series_compound_scaled[:-1000], series_compound_scaled[-1000:]
```

d on model2

```
series_weighted_comp = TimeSeries.from_dataframe(df, 'timestamp', 'weighted_comp')

weighted_comp_scaled = Scaler()
series_weighted_comp_scaled = weighted_comp_scaled.fit_transform(series_weighted_comp)

weighted_comp_train, weighted_comp_val = series_weighted_comp_scaled[:-1000], series_weighted_comp_scaled[-1000:]
```

```
series_weighted_neg = TimeSeries.from_dataframe(df, 'timestamp', 'weighted_neg')

weighted_neg_scaled = Scaler()
series_weighted_neg_scaled = weighted_neg_scaled.fit_transform(series_weighted_neg)

```

figure continue…

added on model2

```
series_weighted_comp = TimeSeries.from_dataframe(df, 'timestamp', 'weighted_comp')

weighted_comp_scaled = Scaler()
series_weighted_comp_scaled = weighted_comp_scaled.fit_transform(series_weighted_comp)

weighted_comp_train, weighted_comp_val = series_weighted_comp_scaled[:-1000], series_weighted_comp_scaled[-1000:]
```

```
series_weighted_neg = TimeSeries.from_dataframe(df, 'timestamp', 'weighted_neg')

weighted_neg_scaled = Scaler()
series_weighted_neg_scaled = weighted_neg_scaled.fit_transform(series_weighted_neg)

weighted_neg_train, weighted_neg_val = series_weighted_neg_scaled[:-1000], series_weighted_neg_scaled[-1000:]
```

```
series_weighted_pos = TimeSeries.from_dataframe(df, 'timestamp', 'weighted_pos')

weighted_pos_scaled = Scaler()
series_weighted_pos_scaled = weighted_pos_scaled.fit_transform(series_weighted_pos)

weighted_pos_train, weighted_pos_val = series_weighted_pos_scaled[:-1000], series_weighted_pos_scaled[-1000:]
```

```
import matplotlib.pyplot as plt


weighted_neg_train.plot()
weighted_neg_val.plot()
```

figure continue…

```
yhat = reg.predict(test_X)
yhat = yhat.reshape(len(yhat), 1)
```

```
yhat = model.predict(test_X)
# invert scaling for forecast
```

```
actual = y_scaler.inverse_transform(test_y.reshape(len(test_y), 1))
```

```
predicted = y_scaler.inverse_transform(yhat)
```

```
reg.save_model("new_data_model4_xgb_legit_only_price.hdf5")
```

```
import plotly.express as px


df_res = pd.DataFrame()

df_res['actual'] = [x[0] for x in actual[:24*15, 0:1]]
df_res['predicted'] = [x[0] for x in predicted[:24*15, 0:1]]
df_res['date'] = [x for x in test_hours[:24*15]]


fig = px.line(df_res, x="date", y=['actual','predicted'],
              hover_data={"date": "|%B %d, %Y"},
              title='Transformer Multivariate with sentiments')

fig.show()
```

figure continue…

```
1 mx = TransformerModel.load_model('/content/drive/MyDrive/rz-transformer/model1.pth.tar')
```

```
/usr/local/lib/python3.7/dist-packages/torchmetrics/utilities/prints.py:36: UserWarning: Torchmetrics v0.9 introduced a new argument c
                not been set for this class (_ResultMetric). The property determines if `update` by
                default needs access to the full metric state. If this is not the case, significant speedups can be
                achieved and we recommend setting this to `False`.
                We provide an checking function
                `from torchmetrics.utilities import check_forward_no_full_state`
                that can be used to check if the `full_state_update=True` (old and potential slower behaviour,
                default for now) or if `full_state_update=False` can be used safely.

  warnings.warn(*args, **kwargs)
```

```
1 model = TransformerModel(input_chunk_length=24, output_chunk_length=12, n_epochs=46, random_state=0)
```

```
1 model.fit([open_train,compound_train,neg_train,weighted_comp_train,weighted_neg_train,weighted_neg_train], verbose=True)
```

```
[2022-06-06 23:12:46,097] INFO | darts.models.forecasting.torch_forecasting_model | Train dataset contains 20142 samples.
[2022-06-06 23:12:46,097] INFO | darts.models.forecasting.torch_forecasting_model | Train dataset contains 20142 samples.
2022-06-06 23:12:46 darts.models.forecasting.torch_forecasting_model INFO: Train dataset contains 20142 samples.
[2022-06-06 23:12:46,132] INFO | darts.models.forecasting.torch_forecasting_model | Time series values are 64-bits; casting model to f
[2022-06-06 23:12:46,132] INFO | darts.models.forecasting.torch_forecasting_model | Time series values are 64-bits; casting model to f
2022-06-06 23:12:46 darts.models.forecasting.torch_forecasting_model INFO: Time series values are 64-bits; casting model to float64.
[2022-06-06 23:12:46,141] WARNING | darts.models.forecasting.torch_forecasting_model | DeprecationWarning: kwarg `verbose` is deprecat
[2022-06-06 23:12:46,141] WARNING | darts.models.forecasting.torch_forecasting_model | DeprecationWarning: kwarg `verbose` is deprecat
2022-06-06 23:12:46 darts.models.forecasting.torch_forecasting_model WARNING: DeprecationWarning: kwarg `verbose` is deprecated and wi
2022-06-06 23:12:46 pytorch_lightning.utilities.rank_zero INFO: GPU available: False, used: False
2022-06-06 23:12:46 pytorch_lightning.utilities.rank_zero INFO: TPU available: False, using: 0 TPU cores
2022-06-06 23:12:46 pytorch_lightning.utilities.rank_zero INFO: IPU available: False, using: 0 IPUs
2022-06-06 23:12:46 pytorch_lightning.utilities.rank_zero INFO: HPU available: False, using: 0 HPUs
2022-06-06 23:12:46 pytorch_lightning.callbacks.model_summary INFO:
  | Name                | Type                | Params
------------------------------------------------------------
0 | criterion           | MSELoss             | 0
1 | encoder             | Linear              | 128
2 | positional_encoding | _PositionalEncoding | 0
3 | transformer         | Transformer         | 548 K
4 | decoder             | Linear              | 780
------------------------------------------------------------
```

figure continue…

```
pred = model.predict(n=32, series=open_val[:-32])

```

```
pred = model.predict(n=32, series=open_val[:-32])
print('MAPE = {:.2f}%'.format(mape(pred,open_val)))
```

```
pred = model.predict(n=32, series=open_train)
print('MAPE = {:.2f}%'.format(mape(pred,open_val)))
```

```
open_train[-36:].plot()
open_train[-1].append(pred).plot()
open_train[-1].append(open_val[:32]).plot()
```

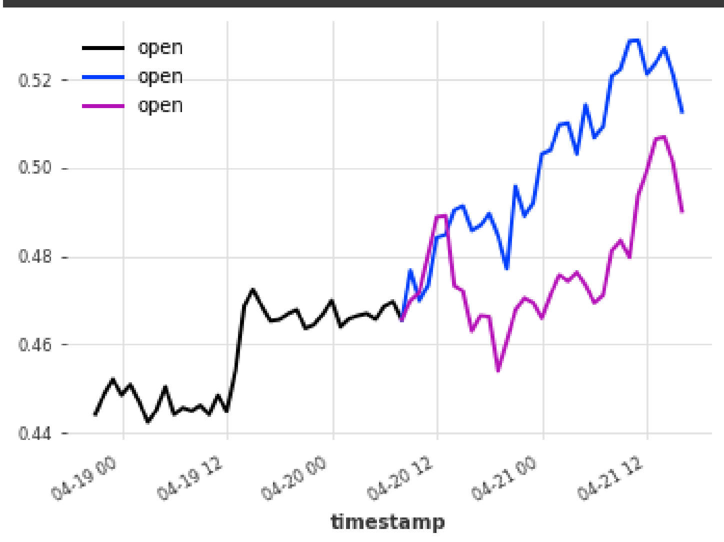

Figure 10.5: Transformers forecasting Code

11. References

1. *Giulia Serafini, Ping Yi, Qingquan Zhang, Marco Brambilla, Jiayue Wang, Yiwei Hu, Beibei Li.* ***“Sentiment-Driven Price Prediction of the Bitcoin based on Statistical and Deep Learning Approaches”****, presented in 2020 International Joint Conference on Neural Networks (IJCNN), published by IEEE, 28 September 2020*

2. *Otabek Sattarov, Heung Seok Jeon, Ryumduck Oh, Jun DongLee,* ***“Forecasting Bitcoin Price Fluctuation by Twitter Sentiment Analysis”****, presented in 2020 International Conference on Information Science and Communications Technologies (ICISCT), published by IEEE, 19 February 2021*

3. *Abid Inamdar, Aarti Bhagtani, Suraj Bhatt, Pooja M. Shetty.* ***“Predicting Cryptocurrency Value using Sentiment Analysis”****, presented in 2019 International Conference on Intelligent Computing and Control Systems (ICCS), published by IEEE, 16 April 2020*

4. *Dibakar Raj Pant, Prasanga Neupane, Anuj Poudel, Anup Kumar Pokhrel, Bishnu Kumar Lama.* ***“Recurrent Neural Network Based Bitcoin Price Prediction by Twitter Sentiment Analysis”****, presented in 2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS), published by IEEE, 24 December 2018*

5. *Jiayun Luo.* ***“Bitcoin price prediction in the time of COVID-19”****, presented in 2020 Management Science Informatization and Economic Innovation Development Conference (MSIEID), Communication and Security (ICCCS), published by IEEE, 23 March 2021*

6. *Apoorva Aggarwal, Isha Gupta, Novesh Garg, Anurag Goel.* ***“Deep Learning Approach to Determine the Impact of Socio Economic Factors on Bitcoin Price Prediction”****, presented in 2019 Twelfth International Conference on Contemporary Computing (IC3), published by IEEE, 19 September 2019*

7. *Ahmed M. Balfagih, Vlado Keselj.* ***“Evaluating Sentiment C1assifiers for Bitcoin Tweets in Price Prediction Task”****, presented in 2019 IEEE International Conference on Big Data (Big Data), published by IEEE, 24 February 2020*

8. *Aditi Mittal, Vipasha Dhiman, Ashi Singh, Chandra Prakash.* ***“Short-Term Bitcoin Price Fluctuation Prediction Using Social Media and Web Search Data”****, presented in 2019 Twelfth International Conference on Contemporary Computing (IC3), published by IEEE, 19 September 2019*

9. *Toni Pano, Rasha Kashef.* ***“A Corpus of BTC Tweets in the Era of COVID-19”****, presented in 2020 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), published by IEEE, 08 October 2020*

10. *Lun Li, Ali Arab, Jiqiang Liu, Jingxian Liu, Zhu Han.* ***“Bitcoin Options Pricing Using LSTM-Based Prediction Model and Blockchain Statistics”****, presented in 2019*

IEEE International Conference on Blockchain (Blockchain), published by IEEE, 02 January 2020

11. *Mehmet Can ERDOĞAN, Murat CANAYAZ.* ***"Crypto-Currency Sentiment Analyse on Social Media"****, presented in 2018 International Conference on Artificial Intelligence and Data Processing (IDAP), published by IEEE, 24 January 2019*

12. *Pavitra Mohanty, Darshan Patel, Parth Patel, Sudipta Roy.* ***"Predicting Fluctuations in Cryptocurrencies' Price using users' Comments and Real-time Prices"****, presented in 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), published by IEEE, 01 July 2019*

13. *Arti Jain, Shashank Tripathi, Harsh Dhar Dwivedi, Pranav Saxena.* ***"Forecasting Price of Cryptocurrencies Using Tweets Sentiment Analysis"****, presented in 2018 Eleventh International Conference on Contemporary Computing (IC3), published by IEEE, 12 November 2018*

14. *Shaomi Rahman, Jonayed Nafis Hemel, Syed Junayed Ahmed Anta, Hossain Al Muhee, Jia Uddin.* ***"Sentiment Analysis Using R: An Approach to Correlate Cryptocurrency Price Fluctuations with Change in User Sentiment Using Machine Learning"****, presented in 2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR), published by IEEE, 14 February 2019*

15. *Shubhankar Mohapatra; Nauman Ahmed; Paulo Alencar.* ***"KryptoOracle: A Real-Time Cryptocurrency Price Prediction Platform Using Twitter Sentiments"****, presented in 2019 IEEE International Conference on Big Data (Big Data), published by IEEE, 24 February 2020*

Submission date: 13th, June'22