DATA MINING ASSIGNMENT -2

By:

TATHAGATA CHAKRABORTY

(MASTER OF SCIENCE, BUSINESS ANALYTICS)

STUDENT NUMBER: 40379173

MODULE CODE: MGT_7216

MODULE NAME: DATA MINING

QUEEN'S UNIVERSITY BELFAST | MANAGEMENT SCHOOL

INTRODUCTION

Twitter and other social media sites have developed into significant global information and communication hubs. Twitter is a popular medium for people to communicate their ideas, thoughts, and opinions with the globe, extolling over 192 million daily active users.

Twitter is a potent tool for connecting with customers, promoting brands, and connecting with target audiences for businesses and individuals. One such figure who has used Twitter to develop his own brand and interact with his fans is Elon Musk, the CEO of Tesla and SpaceX. With more than 58 million followers, Elon Musk's Twitter account has established itself as a well-liked resource for knowledge and motivation for people all around the world.

In this report, we will analyse the Twitter data of Elon Musk's Twitter channel to extract insights and identify patterns in the tweets. We will use various data mining and machine learning techniques to perform sentiment analysis, feature engineering, and predictive modelling on the Twitter data to provide actionable insights for the owner of the Twitter channel. The insights gained from this analysis can help the owner of the Twitter channel to understand their audience, create better content, and increase engagement with their followers.

BACKGROUND

Entrepreneur, innovator, and philanthropist Elon Musk is well-known for his work in the fields of electric vehicles, space travel, and renewable energy. He is also noted for his philanthropic efforts. He is renowned for taking an unusual approach to business and for being willing to take chances in order to succeed. Elon Musk has amassed a sizable following on Twitter thanks to his humorous, frequently irreverent, and even contentious posts. Elon Musk's Twitter account has grown in popularity as a source of information in recent years regarding Tesla, SpaceX, and other projects. Musk has used his Twitter account to interact with his followers, introduce new products, and offer his opinions on the news. Musk's Twitter account is a great source of data for study because his tweets frequently receive a lot of attention and engagement from his followers.

Musk's Twitter data can be analysed to gain knowledge about the topics that appeal to his followers, how his followers feel about his tweets, and the elements that influence engagement with his tweets. Musk and his team can use this information to improve their Twitter strategy and raise follower engagement.

METHODOLOGY

The following methodology describes the process undertaken to extract meaningful insights from the Twitter data, including pre-processing and analysis techniques applied to enhance the quality and usefulness of the data for decision-making purposes.

Data Collection

1. Data was collected by extracting the tweets from Elon Musk's Twitter dataset.
2. The data was collected for a period of 10 months, from January 2022 to October 2022.

Data Pre-processing

1. All twitter handle mentions (@username) were removed.
2. Punctuations were removed.
3. Emojis/images were removed.
4. URLs were removed.
5. Number of hashtags in each tweet were counted.
6. Stop words were handled by removing commonly used English stop words.
7. The remaining words were tokenized.
8. Features were extracted from the dataset to enhance visualisation, such as the number of likes, and retweets, and the timestamp for each tweet.
9. Timestamps were converted to month, day of week and time of day.

Data Analysis

- Basic statistical observations were made, including the frequency of tweets, and the average number of likes and retweets over time.
- Mean likes, and retweets were plotted and grouped by month, day, and time of day for the tweets.
- Sentiment scores were calculated for each tweet using a pre-trained sentiment analysis model using the 'nltk' library in Python.
- A word cloud was created to display the most popular words contained in the tweets in the data set.
- Top 5 tweets with the most likes were selected for sentiment analysis, and the findings were discussed.
- Concordance and collocations were extracted.
- Three different machine learning algorithms were trained on the data to predict the number of likes a tweet is likely to get, and their accuracy was compared.
- The results of the statistical observations, sentiment analysis, and machine learning models were presented and discussed.
- Topic modelling was performed using LDA (Latent Dirichlet Allocation) to determine which topics were talked about the most.
- Observations from the data were highlighted, including the most popular topics, sentiment trends, and the impact of various features on the number of likes a tweet receives.

RESULTS AND DISCUSSION

Data Preparation:

To pre-process the data, several steps were taken to clean the Twitter data and prepare it for analysis. First, all Twitter handle mentions were removed to ensure the privacy of users and eliminate noise from the data. Punctuations, emojis, and URLs were also removed to ensure the text was in a standard format that could be analysed accurately.

```
# removes pattern in the input text
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for word in r:
        input_txt = re.sub(word, "", input_txt)
    return input_txt
```

```
[ ]  # remove twitter handles (@user) and URLs
     df['clean_tweet'] = np.vectorize(remove_pattern)(df['Tweets'], "@[\w]*|http\S+")
```

The code block above depicts the removal of mentions and URLs from the tweets.

In a similar manner emojis and punctuations were removed from the tweets.

```
]  # remove punctuations from clean_tweet column
   df['clean_tweet'] = df['clean_tweet'].apply(remove_punctuation)
```

```
[ ]  # remove emojis from text in 'clean_tweet' column and store in a new column
     df['clean_tweet'] = df['clean_tweet'].apply(remove_emoji)
```

The functions ‘remove_punctuation’ and ‘remove_emoji’ are defined in the code which can be accessed in the Notebook file OR in the appendix section.

To separate words into individual tokens and to remove irrelevant words in the dataset, we first tokenized the text and then removed stop words such as "the" and "a". This was done to improve the accuracy of the analysis by reducing the noise in the dataset.

```
# tokenize text in 'clean_tweet' column and store in a new column
from nltk.tokenize import word_tokenize
nltk.download('punkt')
df['tokenized_tweet'] = df['clean_tweet'].apply(word_tokenize)
```

```
#remove stop words from the tokenized tweets
nltk.download('stopwords')
from nltk.corpus import stopwords

# remove stop words from tokenized_tweet column
stop_words = set(stopwords.words('english'))
df['tokenized_tweet'] = df['tokenized_tweet'].apply(lambda x: [word for word in x if word.lower() not in stop_words])
```

To enhance visualization and understanding of the dataset, additional features were extracted such as the number of likes, retweets, and the timestamp for each tweet.

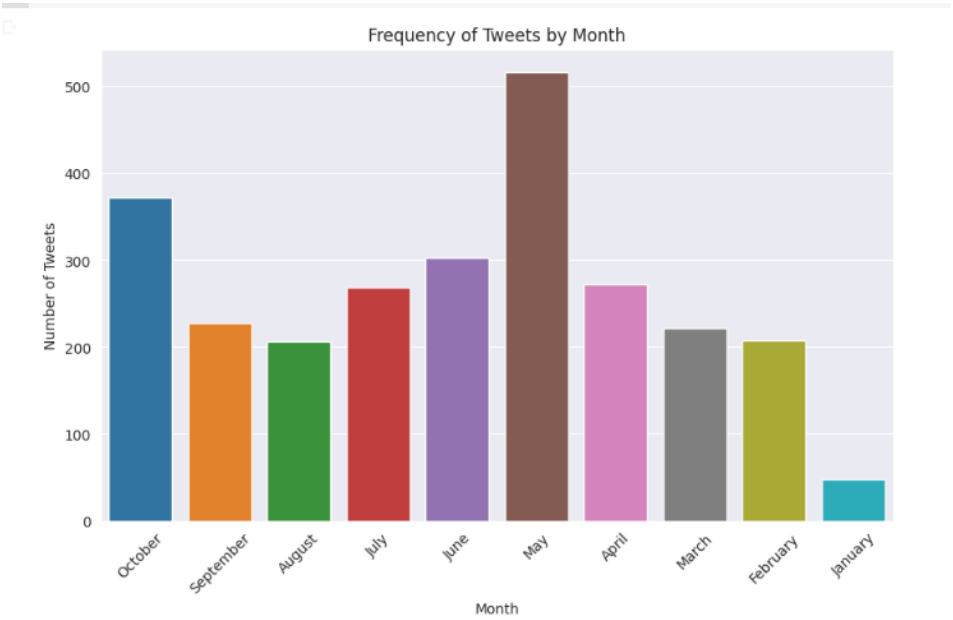
```
import datetime
# convert the date column to a datetime object
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y %H:%M')

# define a function to extract the month from a datetime object
def extract_month(date):
    return date.strftime('%B')

# apply the function to each row of the date column to create a new month column
df['Month_of_Tweet'] = df['Date'].apply(extract_month)

# define a function to extract the time of day from a datetime object
def extract_time_of_day(date):
    return date.strftime('%H:%M:%S')

# apply the function to each row of the date column to create a new time_of_day column
df['time_of_day'] = df['Date'].apply(extract_time_of_day)

# extract the day of the week from the date column and create a new column
df['day_of_week'] = df['Date'].dt.day_name()
```

The timestamps were converted to month, day of week, and time of day, which could provide insights into the most active times on Twitter and which topics were trending at different times.

These steps are consistent with best practices for pre-processing textual data for machine learning applications (Tibrewal & Shukla, 2021; Pranita & Tanvi, 2020). The use of stop words removal and tokenization are common methods for text preprocessing (Jain & Saini, 2021). Additionally, the extraction of features such as timestamps has been used in previous studies to understand user behaviour on Twitter (Cheng, Danescu-Niculescu-Mizil, & Leskovec, 2014).

Data Visualisation :

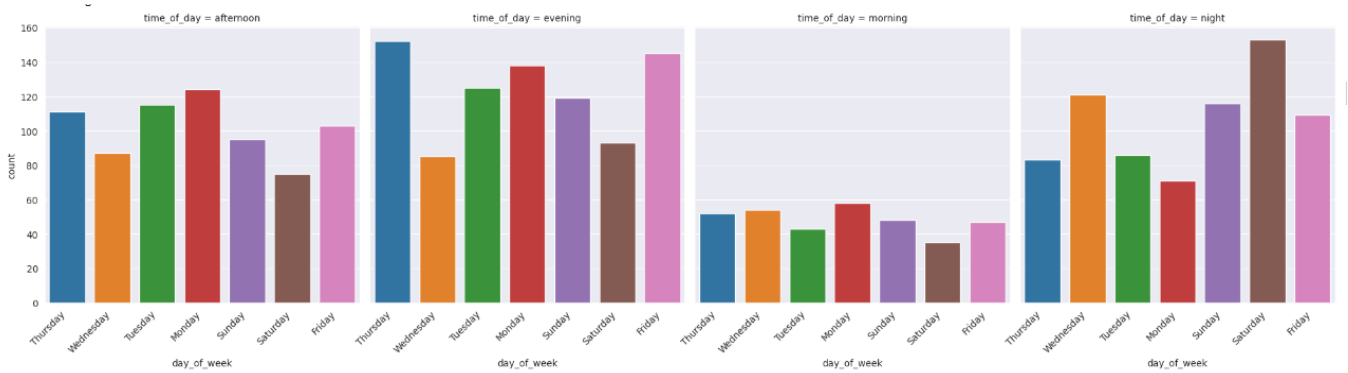
We plot the number of the tweets tweeted by Musk by each month.



From the plot it is evident that most number of tweets were posted in May followed by October and June.

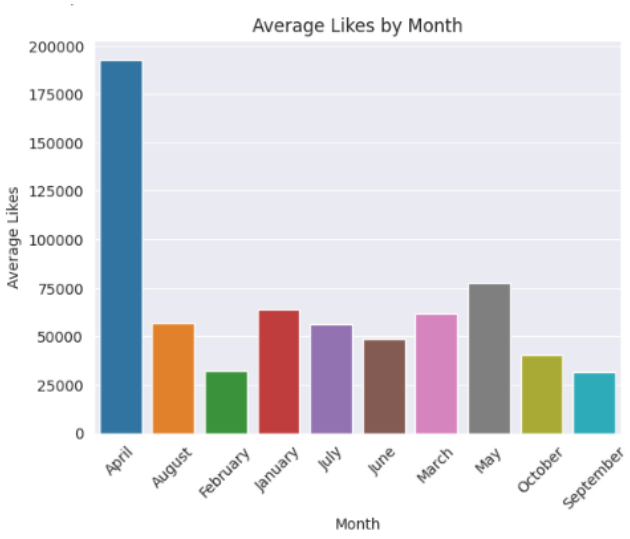
However, the number of tweets posted shows very little variation across the days of the week.

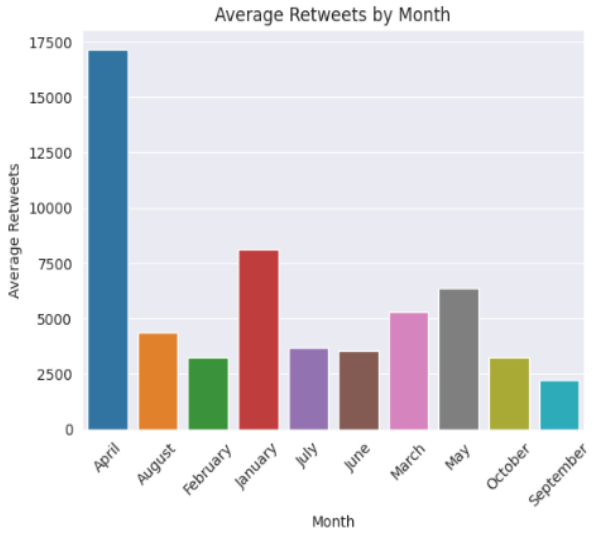
Thursday and Friday show marginally higher frequency of tweets than the remaining days, however overall, there is not much variation in tweet frequency across the days of the week.



It was also noticed that the least frequency of tweets was during morning time and the highest number of tweets was posted during evening across all days of the week.

Despite May having the highest number of tweets, it was seen that the highest average likes and retweets were both found in the tweets posted in April.





Text-mining and Sentiment Analysis :

We have found the frequency of the most occurring words in tweets. By analysing a large dataset of tweets, we can identify the words that are used most frequently and determine the themes and topics that are prevalent. This information can then be used to create a word cloud, which visually represents the most frequently occurring words in the dataset.

From the word cloud that we have generated , it can be seen that:

- Amp
- Tesla
- Twitter
- Starlink

are the most frequently occurring words in the tweets posted by Musk.

Furthermore, we have also extracted the words which occur together most frequently.

Word Cloud for Frequently occurring words

Word Cloud for collocations

Another way to analyse the language in tweets is to identify collocations. Collocations are pairs of words that frequently appear together in a particular language.

By identifying collocations in tweets, we can gain a better understanding of how language is used in the context of social media.

From the word cloud we can identify that:

- ai day
- fsd beta
- Sustainable energy
- Free speech
- Tesla ai

are some of the most common collocations which occur frequently.

Concordance is a tool that allows us to examine the context in which a particular word appears. By examining the concordance of frequently occurring words in tweets, we can gain insights into the opinions and attitudes of Twitter users. Overall, analysing the language in tweets can provide valuable information about the trends and topics that are currently being discussed on social media.

```
##Find concordance of frequently occuring words and collocations
from nltk.text import Text
# create a Text object from the tokenized tweets
text = Text(df['tokenized_tweet'].sum())

# use the concordance method to find occurrences of a word in the text
text.concordance('Tesla')
```

```
Displaying 25 of 150 matches:
lexity otherwise possible Varies lot Tesla amp SpaceX good positions many compa
gful thrust Norway rocks case missed Tesla AI Day Satellite constellation capab
ve Excited announce start production Tesla Semi Truck deliveries Dec Tiny heliu
ssian majorities prefer Russia First Tesla consumer tax credit years amp ask on
us sickkk Would cool make game using Tesla Sim literally city amp neighborhood
ce expedite costs amp relieve stress Tesla team Aiming steadier deliveries intr
eat team Beautiful Exactly Recording Tesla AI Day Join little giving brain Wego
I Day show immense depth amp breadth Tesla AI compute hardware amp robotics Yup
need race anything else Webcast link Tesla AI Day starts precisely California t
Day starts precisely California time Tesla team awesome honor work cgi hands Op
st coal shipment arrived Hawaii time Tesla Megapack batteries enable sustainabl
mely concerning ironic Henrik Fisker Tesla Karma need add lot batteries grid bu
 Looks awesome Maybe little electric Tesla boats retrofuturistic Victorian desi
r Still thing bigger Looks good roll Tesla owners cars probably right order mag
lation Please lmk happens Absolutely Tesla fleet Australia growing rapidly need
ng rapidly need ramp service general Tesla North America aiming hour service An
utting lot time personally advancing Tesla service make awesome Hopefully start
make awesome Hopefully starting felt Tesla owners right credit Tesla owners cha
rting felt Tesla owners right credit Tesla owners change appointments less hour
 reciprocal Lot people still realize Tesla makes uninterruptible power supplies
werwall possible end year Cool Order Tesla Powerwall battery blackout protectio
```

For instance, in the above snippet, we have tried to produce the concordance for the word ‘Tesla’ in all the tweets. We can see that in most of the contexts, Musk has either in some way been appreciative of Tesla’s progress or has been speculative of further modifications or developments.

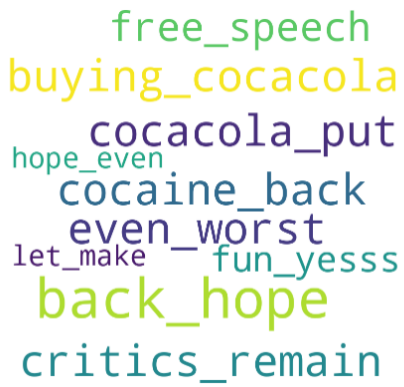
We have repeated this step for a multitude of frequently occurring words in order to get a sense of the context in which the words have been used.

We have now generated the sentiment score for the tweets using the ‘nltk’ package in Python. The top 5 tweets consisting of the highest number of Likes have been extracted and the previous mining techniques have been applied on them.

```
# apply the get_tweet_sentiment function to the 'clean_tweet' column and create a new column for the sentiment
df['Sentiment_Score'] = df['clean_tweet'].apply(get_tweet_sentiment)

#subset the top 5 tweets with most likes and tweets with their sentiment score
top5_tweets = df.nlargest(5, 'Likes')
print(top5_tweets[['clean_tweet', 'Likes', 'Sentiment_Score']])
```

```
                                        clean_tweet    Likes  \
2219  Next I'm buying CocaCola to put the cocaine ba...  4780787
2244  I hope that even my worst critics remain on Tw...  3232772
2216                    Let's make Twitter maximum fun  2650644
2243                                             Yesss  2608578
2215                    Listen I can't do miracles ok   2581112

     Sentiment_Score
2219         neutral
2244        negative
2216        positive
2243         neutral
2215        positive
```

From the above snippet we can infer that for the top 5 tweets with most likes, 40% of the tweets have a positive sentiment while only 20% has a negative undertone, the remaining being neutral.



We have also pooled the collocations in the top 5 most liked tweets of Musk.

It can be seen from the word cloud that these phrases have been used in conjunction very frequently in the top 5 tweets.

The phrase 'free speech' is found to be common with the collocation word cloud mapped for the entire dataset, thus it can be concluded that 'free speech' garners more support from Musk's followers and twitter users.

We further try to determine the sentiment scores of the tweets containing only 'Tesla' or 'Amp' in them as they are some of the most frequently occurring words in Musk's tweets.

```
#subsetting the dataset for tweets containing 'Tesla'
tesla_tweets = df[df['clean_tweet'].str.contains('Tesla',case=False)]

sentiment_counts = tesla_tweets['Sentiment_Score'].value_counts()
print('Sentiment Count For tweets containing "Tesla"')
print(sentiment_counts)
```

```
Sentiment Count For tweets containing "Tesla"
positive    79
neutral     36
negative    29
Name: Sentiment_Score, dtype: int64
```

From the analysis, it is clear that most tweets containing 'Tesla' have a positive sentiment score.

```
#subsetting the dataset for tweets containing 'amp'
amp_tweets = df[df['clean_tweet'].str.contains('amp',case= False)]

sentiment_counts = amp_tweets['Sentiment_Score'].value_counts()
print('Sentiment Count For tweets containing "Starlink"')
print(sentiment_counts)
```

```
Sentiment Count For tweets containing "Starlink"
positive    148
negative     64
neutral      43
Name: Sentiment_Score, dtype: int64
```

It can also be seen that Musk is probably most optimistic about 'amp'.

This observation is in line with the finding that AMP is a digital currency that is gaining popularity in the crypto world. Musk has been known to influence the price of various cryptocurrencies with his tweets, and AMP is no exception. In July 2021, Musk tweeted about AMP, causing its price to surge by over 25% within a matter of hours.

We have also reverified these findings by performing topic modelling using LDA (Latent Dirichlet Allocation) algorithm that the most talked about topics in the most popular tweets were Tesla, Twitter, and Amp.

Machine Learning Predictive models :

We first adopt a simple linear regression model to predict the number of likes of the tweets. In the process we figured that, 'Sentiment Score' and 'time_of_day' have high p-value, thus have been removed from the model analysis predictor list, BUT the performance of the model was not affected by their absence and remained relatively same.

```
Linear Regression Model:
Mean Squared Error: 2165938939.4296894
Root Mean Squared Error: 46539.649111587525
R-squared score: 0.8632236751401086
Coefficients: [   8.48277829  817.01269449 -520.31358667]
```

The model generated an **R-squared value of 0.8632**, thus almost **86%** of the variance in the data can be explained by the linear regression model.

We then take a rather debatable approach, i.e we apply random forest algorithm for the same purpose. The computer science community is divided regarding this idea as to IF random forest algorithm can be used for a NON-classification problem and a continuous target variable.

One reason why Random Forest may not be suitable for regression problems is that it is prone to overfitting, which can lead to poor predictive performance. Overfitting occurs when the model fits the training data too closely and captures noise and randomness in the data rather than the

underlying patterns and relationships. In regression problems, overfitting can lead to a model that predicts the training data well but performs poorly on new, unseen data.

```
Random Forest Model:
Mean Squared Error: 2165938939.4296894
R-squared Score: 0.885806308555283
Root Mean Squared Error: 42524.44905411132
```

The model metrics show that random forest performs slightly better than linear regression by generating an **R-squared value of 0.8858** .

Thus, **88.58%** of the variance can be explained by this model.

We finally perform KNN to obtain a predictive model for the same purpose.

```
KNN Regression Model:
Mean Squared Error: 2165938939.4296894
R-squared Score: 0.8785729037659875
Root Mean Squared Error: 43850.59032197438
```

KNN performs slightly worse than random forest but is still better than linear regression **with an R-squared of 0.8785**.

Hence, we can infer from the modelling that random forest algorithm provides the most reliable and accurate model amongst the three.

CONCLUSION

From the analysis conducted over the dataset we have finalised a handful of suggestions which shall help in further expanding the outreach of the twitter channel in question and garnering more support and following.

Key Insights and suggestions :

- Most likes and retweets were observed for tweets posted in April.
- More tweets should be posted during morning to improve engagement.
- Tweets must involve AMP, Tesla, or twitter related discussion.
- Tweets with more positive undertone received more support from followers.
- May records the highest number of tweets but surprisingly has comparably low average likes, and retweets. Topics of tweets in May should revolve around the aforementioned topics.
- Musk may also maintain supporting issues like ‘free speech’ and ‘sustainable energy’, which may help him increase engagement.

Moreover, Musk must continue his constructive discussions around Tesla, AI, cryptocurrency, Starlink, and twitter which is clearly inviting a lot of attention and engagement from his followers.

REFERENCES

- Cheng, J., Danescu-Niculescu-Mizil, C., & Leskovec, J. (2014). How community feedback shapes user behavior. In Proceedings of the 8th International AAAI Conference on Weblogs and Social Media.
- Jain, N., & Saini, R. (2021). Sentiment analysis of social media data using machine learning techniques. In 2021 3rd International Conference on Communication and Electronics Systems (ICCES) (pp. 846-851). IEEE.
- Pranita, G., & Tanvi, S. (2020). A study of techniques for text data preprocessing. In 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS) (pp. 138-142). IEEE.
- Tibrewal, A., & Shukla, A. (2021). Text Preprocessing Techniques for Natural Language Processing Applications: A Review. In Advances in Artificial Intelligence and Soft Computing (pp. 101-113). Springer.

APPENDIX

```
#!/usr/bin/env python
# coding: utf-8




# In[1]:



#IMPORTING ALL NECESSARY LIBRARIES
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import string
import nltk



# In[3]:



from google.colab import drive
drive.mount('/content/drive')



# LOADING THE ELON MUSK TWEETS DATASET

# In[4]:
```

```
df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Elon_Musk_Tweet.csv')
df.head()
```

```
# In[5]:
```

```
df.info()
```

```
# In[6]:
```

```
# defining function which removes pattern in the input text
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for word in r:
        input_txt = re.sub(word, "", input_txt)
    return input_txt
```

```
# In[7]:
```

```
# remove twitter handles (@user) and URLs
df['clean_tweet'] = np.vectorize(remove_pattern)(df['Tweets'], "@[\w]*|http\S+")
```

```
# In[8]:
```

```
# define a function to extract hashtags from a tweet
def extract_hashtags(tweet):
    hashtags = re.findall(r'#\w+', tweet)
    return hashtags

# apply the function to each tweet in the 'tweet' column
df['hashtags'] = df['clean_tweet'].apply(extract_hashtags)

# create a new dataframe to store the count of each hashtag
hashtags_df = pd.DataFrame(df['hashtags'].explode().value_counts())

# rename the column to 'count'
hashtags_df.rename(columns={'hashtags': 'count'}, inplace=True)

# reset the index so that the hashtags become a column
hashtags_df.reset_index(inplace=True)

# rename the column to 'hashtag'
hashtags_df.rename(columns={'index': 'hashtag'}, inplace=True)

# merge the hashtag counts back into the original dataframe
df['hashtags'] = df['hashtags'].apply(lambda x: ' '.join(x))
hashtags_df['hashtag'] = hashtags_df['hashtag'].apply(lambda x: ' '.join(x))
df = df.merge(hashtags_df, how='left', left_on='hashtags', right_on='hashtag')

# fill missing values in the 'count' column with 0
df['count'].fillna(0, inplace=True)

# drop the 'hashtag' column
df.drop(columns='hashtag', inplace=True)
```

```
# It was found that there are NO hashtags in any tweets in the dataset.
# Proceeding to remove punctuations, and emojis from the tweets.

# In[9]:


# define function to remove punctuations
def remove_punctuation(text):
    translator = str.maketrans('', '', string.punctuation)
    return text.translate(translator)


# In[10]:


# remove punctuations from clean_tweet column
df['clean_tweet'] = df['clean_tweet'].apply(remove_punctuation)


# In[11]:


df.head()


# In[12]:


# define function to remove emojis
```

```
def remove_emoji(text):
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002500-\U00002BEF"  # chinese char
                           u"\U00002702-\U000027B0"
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           u"\U0001f926-\U0001f937"
                           u"\U00010000-\U0010ffff"
                           u"\u2640-\u2642"
                           u"\u2600-\u2B55"
                           u"\u200d"
                           u"\u23cf"
                           u"\u23e9"
                           u"\u231a"
                           u"\ufe0f"  # dingbats
                           u"\u3030"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)


# In[13]:


# remove emojis from text in 'clean_tweet' column and store in a new column
df['clean_tweet'] = df['clean_tweet'].apply(remove_emoji)
```

In[14]:

df.head()

Dropping all tweets which are empty after removal of emojis, URLs, mentions.

In[15]:

```
# remove rows where the tweet is an empty string or blank
df = df.loc[df['clean_tweet'].str.strip() != '']
```

Tokenizing the cleaned tweets.

In[16]:

```
# tokenize text in 'clean_tweet' column and store in a new column
from nltk.tokenize import word_tokenize
nltk.download('punkt')
df['tokenized_tweet'] = df['clean_tweet'].apply(word_tokenize)
```

Further checking for any non alphabetical characters in the tweets and removing them.

In[17]:

```
# define function to remove non-alphabetical words
def remove_non_alpha(words):
    return [word for word in words if word.isalpha()]


# In[18]:


# remove non-alphabetical words from the tokenized_tweet column
df['tokenized_tweet'] = df['tokenized_tweet'].apply(remove_non_alpha)


# Removing Stop-Words from the tweets to improve the analysis.

# In[19]:


#remove stop words from the tokenized tweets
nltk.download('stopwords')
from nltk.corpus import stopwords

# remove stop words from tokenized_tweet column
stop_words = set(stopwords.words('english'))
df['tokenized_tweet'] = df['tokenized_tweet'].apply(lambda x: [word for word in x if word.lower() not in stop_words])


# Converting the timestamp of the tweet into month,day and time of tweet.

# In[20]:
```

```
import datetime
# convert the date column to a datetime object
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y %H:%M')

# define a function to extract the month from a datetime object
def extract_month(date):
    return date.strftime('%B')

# apply the function to each row of the date column to create a new month column
df['Month_of_Tweet'] = df['Date'].apply(extract_month)

# define a function to extract the time of day from a datetime object
def extract_time_of_day(date):
    return date.strftime('%H:%M:%S')

# apply the function to each row of the date column to create a new time_of_day column
df['time_of_day'] = df['Date'].apply(extract_time_of_day)

# extract the day of the week from the date column and create a new column
df['day_of_week'] = df['Date'].dt.day_name()



# In[21]:



# categorize the time of day column into morning, noon, evening, night categories

# convert the 'time_of_day' column to datetime type
df['time_of_day'] = pd.to_datetime(df['time_of_day'])
```

```
# extract the hour component as an integer
df['hour'] = df['time_of_day'].dt.hour

def categorize_time(hour):
    if hour < 6:
        return 'night'
    elif hour < 12:
        return 'morning'
    elif hour < 18:
        return 'afternoon'
    else:
        return 'evening'

df['time_of_day'] = df['hour'].apply(categorize_time)
df.drop('hour', axis=1, inplace=True)



# In[22]:



df.head()



# STATISTICAL VISUALISATION OF THE CLEANED DATASET

# Producing the frequency of tweets posted by Month

# In[23]:



tweet_freq = df['Month_of_Tweet'].value_counts()
```

```
print(tweet_freq)


# Plotting the tweet frequency by Month

# In[24]:


sns.set_style('darkgrid')
plt.figure(figsize=(10, 6))
ax = sns.countplot(x='Month_of_Tweet', data=df)
plt.xticks(rotation=45)
plt.title('Frequency of Tweets by Month')
plt.xlabel('Month')
plt.ylabel('Number of Tweets')
plt.show()
plt.savefig('p1.png')


# Plotting the tweet frequency by day of the week.

# In[25]:


sns.set_style('darkgrid')
plt.figure(figsize=(10, 6))
ax = sns.countplot(x='day_of_week', data=df)
plt.xticks(rotation=45)
plt.title('Frequency of Tweets by Day of Week')
plt.xlabel('Day')
plt.ylabel('Number of Tweets')
```

```
plt.show()
```

Plotting the tweet frequency by time of day when tweet was pposted.

In[26]:

```
sns.set_style('darkgrid')
plt.figure(figsize=(10, 6))
ax = sns.countplot(x='time_of_day', data=df)
plt.xticks(rotation=45)
plt.title('Frequency of Tweets by Time of Day')
plt.xlabel('Time of Day')
plt.ylabel('Number of Tweets')
plt.show()
```

Faceting the frequency plots by time of day, and day of the week

In[27]:

```
ax = sns.catplot(x='day_of_week', kind='count', data=df, col='time_of_day')
ax.set_xticklabels(rotation=45, ha='right')
```

Calculation average Likes received in tweets per month and plotting the chart.

In[28]:

```
likes_by_month = df.groupby('Month_of_Tweet')['Likes'].mean().reset_index()
print(likes_by_month)

# plot the mean likes by month using seaborn barplot
sns.barplot(x='Month_of_Tweet', y='Likes', data=likes_by_month)

# set the plot title and axis labels
plt.title('Average Likes by Month')
plt.xlabel('Month')
plt.ylabel('Average Likes')
plt.xticks(rotation=45)

# show the plot
plt.show()



# Plotting average retweets by month

# In[29]:



retweets_by_month = df.groupby('Month_of_Tweet')['Retweets'].mean().reset_index()
print(retweets_by_month)

# plot the mean likes by month using seaborn barplot
sns.barplot(x='Month_of_Tweet', y='Retweets', data=retweets_by_month)

# set the plot title and axis labels
plt.title('Average Retweets by Month')
plt.xlabel('Month')
```

```
plt.ylabel('Average Retweets')
plt.xticks(rotation=45)

# show the plot
plt.show()



# FREQUENCY DISTRIBUTION OF WORDS

# Finding the most frequently occuring words in the tweets.

# In[30]:



# create frequency distribution of words in tokenized_tweet column
freq_dist = nltk.FreqDist(word for words in df['tokenized_tweet'] for word in words)

# print 10 most common words
print(freq_dist.most_common(10))



# Tabulating the frequently occuring words.

# In[31]:



freq_dist.tabulate(5)



# Creating a word cloud for most frequently occuring tweets
```

In[32]:

#create a word cloud containing most poplar words in the tweet

from wordcloud import WordCloud

concatenate all the tweets into a single string

text = ' '.join(df['clean_tweet'])

create a WordCloud object

wordcloud = WordCloud(width=800, height=800, background_color='white',
min_font_size=10).generate(text)

plot the word cloud

plt.figure(figsize=(8,8), facecolor=None)

plt.imshow(wordcloud)

plt.axis("off")

plt.tight_layout(pad=0)

plt.show()

COLLOCATION AND CONCORDANCE OF FREQUENTLY OCCURING WORDS

Finding the frequent collocations and concordance of frequently occuring words in the tweets.

In[33]:

#Finding Collocations in the cleaned tweets

from nltk.collocations import BigramAssocMeasures, BigramCollocationFinder

```
# Concatenate all tweets into one long string
text = ' '.join(df['clean_tweet'])

# Tokenize the text and remove stopwords
tokens = nltk.word_tokenize(text)
stop_words = set(stopwords.words('english'))
tokens = [token.lower() for token in tokens if token.isalpha() and token.lower() not in stop_words]

# Find bigram collocations
finder = BigramCollocationFinder.from_words(tokens)
measures = BigramAssocMeasures()
collocations = finder.nbest(measures.raw_freq, 10)

# Print the collocations
print(collocations)

# convert the collocations into a string for generating the word cloud
text = " ".join(["_".join(collocation) for collocation in collocations])

# create the word cloud
wordcloud = WordCloud(width=800, height=800, background_color='white').generate(text)

# plot the word cloud
plt.figure(figsize=(8, 8), facecolor=None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()



# In[34]:
```

##Find concordance of frequently occuring words and collocations

```
from nltk.text import Text

# create a Text object from the tokenized tweets

text = Text(df['tokenized_tweet'].sum())


# use the concordance method to find occurrences of a word in the text

text.concordance('Tesla') #replacing the word by 'Twitter','Amp','Starlink' to find concordance.
```

SENTIMENT ANALYSIS ON TOP 5 TWEETS WITH MOST LIKES

Generating sentiment scores of the tweets.Subsetting the top 5 tweets with most likes and analysing their sentiment scores.

In[35]:

```
#subsetting the top 5 tweets with most number of likes

# import the necessary libraries

from textblob import TextBlob


# create a function to get the sentiment of each tweet

def get_tweet_sentiment(tweet):

    # create a TextBlob object of the tweet

    blob = TextBlob(tweet)


    # get the sentiment polarity (-1 to 1) and subjectivity (0 to 1) of the tweet

    sentiment_polarity = blob.sentiment.polarity

    sentiment_subjectivity = blob.sentiment.subjectivity
```

```
    # classify the sentiment based on polarity
    if sentiment_polarity > 0:
        return 'positive'
    elif sentiment_polarity == 0:
        return 'neutral'
    else:
        return 'negative'


# apply the get_tweet_sentiment function to the 'clean_tweet' column and create a new column for the sentiment
df['Sentiment_Score'] = df['clean_tweet'].apply(get_tweet_sentiment)

#subset the top 5 tweets with most likes and tweets with their sentiment score
top5_tweets = df.nlargest(5, 'Likes')
print(top5_tweets[['clean_tweet', 'Likes', 'Sentiment_Score']])


# In[36]:


top5_tweets.head(5)


# Finding and Printing Collocations in the top 5 tweets.

# In[37]:


# Concatenate all tweets into one long string
```

```
text = ' '.join(top5_tweets['clean_tweet'])

# Tokenize the text and remove stopwords
tokens = nltk.word_tokenize(text)
stop_words = set(stopwords.words('english'))
tokens = [token.lower() for token in tokens if token.isalpha() and token.lower() not in stop_words]

# Find bigram collocations
finder = BigramCollocationFinder.from_words(tokens)
measures = BigramAssocMeasures()
collocations = finder.nbest(measures.raw_freq, 10)

# Print the collocations
print(collocations)

# convert the collocations into a string for generating the word cloud
text = " ".join(["_".join(collocation) for collocation in collocations])

# create the word cloud
wordcloud = WordCloud(width=800, height=800, background_color='white').generate(text)

# plot the word cloud
plt.figure(figsize=(8, 8), facecolor=None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()



# **SENTIMENT ANALYSIS OF TWEETS CONTAINING 'TESLA', AND/OR 'AMP'**
```

Subsetting the tweets containing 'Tesla' and/or 'Amp' and analysing their sentiment scores.

In[38]:

```
#subsetting the dataset for tweets containing 'Tesla'
tesla_tweets = df[df['clean_tweet'].str.contains('Tesla',case=False)]

sentiment_counts = tesla_tweets['Sentiment_Score'].value_counts()
print('Sentiment Count For tweets containing "Tesla"')
print(sentiment_counts)
```

In[39]:

```
#subsetting the dataset for tweets containing 'amp'
amp_tweets = df[df['clean_tweet'].str.contains('amp',case= False)]

sentiment_counts = amp_tweets['Sentiment_Score'].value_counts()
print('Sentiment Count For tweets containing "Starlink"')
print(sentiment_counts)
```

TOPIC MODELLING FOR ELON MUSK'S TWEETS

Extracting the most frequently occuring topoics in the top 500 tweets in the dataset.

In[40]:

```
import gensim
from gensim import corpora

# create a list of tokenized tweets considering only top 500 tweets based on likes
tweets = df.nlargest(500, 'Likes')
tweets = tweets['clean_tweet'].apply(lambda x: x.split())

# create dictionary
dictionary = corpora.Dictionary(tweets)

# create document term matrix
doc_term_matrix = [dictionary.doc2bow(tweet) for tweet in tweets]

# build LDA model
lda_model = gensim.models.ldamodel.LdaModel(doc_term_matrix, num_topics=5,
id2word=dictionary, passes=50, random_state=42)

# print topics and their top words
for topic in lda_model.show_topics(num_topics=5, formatted=False):
    print('Topic {}: {}'.format(topic[0], [w[0] for w in topic[1]]))



# **MACHINE LEARNING ALGORITHM FOR PREDICTING LIKES OF TWEETS**
#
# LINEAR REGRESSION MODEL

# In[41]:



#factorize categorical variables in the dataset
df['Sentiment_Score'] = pd.factorize(df['Sentiment_Score'])[0]
```

```
df['Month_of_Tweet'] = pd.factorize(df['Month_of_Tweet'])[0]
df['day_of_week'] = pd.factorize(df['day_of_week'])[0]
df['time_of_day'] = pd.factorize(df['time_of_day'])[0]


# In[42]:


#import relevant libraries
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# select the features to use for prediction
X = df[['Retweets','Month_of_Tweet','day_of_week']]
y = df['Likes']

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# create a linear regression model
lr_model = LinearRegression()

# fit the model on the training data
lr_model.fit(X_train, y_train)

# make predictions on the test data
y_pred = lr_model.predict(X_test)

# calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)
```

```
print('Linear Regression Model:')
print('Mean Squared Error:', mse)

#calculate root mean squared error
rmse = mean_squared_error(y_test, y_pred, squared = False)
print('Root Mean Squared Error:', rmse)

# R2 score
from sklearn.metrics import r2_score
print('R-squared score:', r2_score(y_test, y_pred))

# Coefficients
print('Coefficients:', lr_model.coef_)



# In[43]:



from sklearn.feature_selection import f_regression
# Calculate the F-statistic and p-values
f_vals, p_vals = f_regression(X_train, y_train)

# Print the F-statistic and p-values for each feature
for i in range(len(X.columns)):
    print('Feature:', X.columns[i])
    print('F-Statistic:', f_vals[i])
    print('p-value:', p_vals[i])
    print('-----------------------------------')
```

Sentiment Score and time_of_day have high p-value, thus have been removed from the model analysis BUT the performance of the model was not affected by their absence and remained relatively same.

RANDOM FOREST ALGORITHM

In[44]:

```
from sklearn.ensemble import RandomForestRegressor

# select the features to use for prediction
X = df[['Retweets','Month_of_Tweet','day_of_week']]
y = df['Likes']

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the Random Forest model with default hyperparameters
rf_model = RandomForestRegressor()

# Fit the model to the training data
rf_model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = rf_model.predict(X_test)

# Calculate the RMSE and R2 score
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
```

```
print('Random Forest Model:')
print('Mean Squared Error:', mse)
print('R-squared Score:', r2)
print('Root Mean Squared Error:', rmse)


# KNN ALGORITHM

# In[45]:


from sklearn.neighbors import KNeighborsRegressor

# select the features to use for prediction
X = df[['Retweets','Month_of_Tweet','day_of_week']]
y = df['Likes']

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the KNN model with default hyperparameters
knn_model = KNeighborsRegressor()

# Fit the model to the training data
knn_model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = knn_model.predict(X_test)

# Calculate the RMSE and R2 score
rmse = mean_squared_error(y_test, y_pred, squared=False)
```

```
r2 = r2_score(y_test, y_pred)


print('KNN Regression Model:')
print('Mean Squared Error:', mse)
print('R-squared Score:', r2)
print('Root Mean Squared Error:', rmse)
```