NAME - MARTIN CHINEDU OGUEJIOFOR

STUDENT ID - W1952112

MODULE CODE - 7BUIS025W

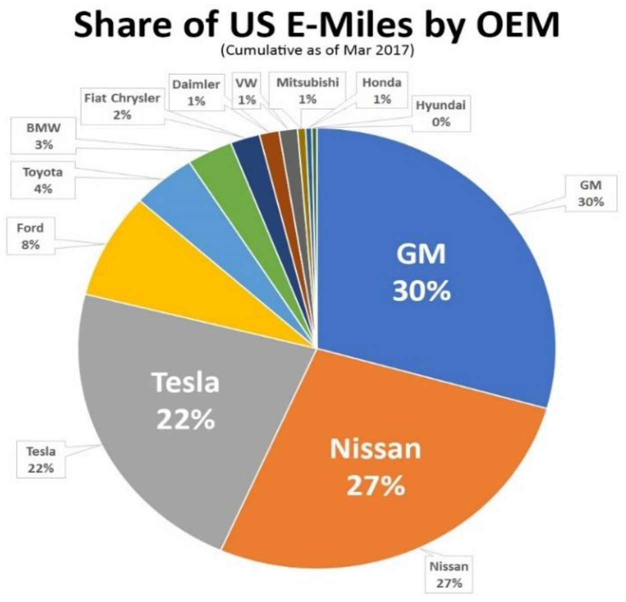
MODULE TITLE - WEB AND SOCIAL MEDIA ANALYTICS

WORD COUNT - 5000

INVESTIGATING ELECTRIC VEHICLE OWNERSHIP EXPERIENCES THROUGH SOCIAL MEDIA ANALYSIS

INTRODUCTION

The electric vehicle (EV) industry is undergoing rapid expansion and innovation, fueled by heightened environmental awareness, governmental incentives, and technological advances. This study focuses on Fully Electric Vehicles (FEVs), which operate exclusively on electric power. The allure of FEVs has surged alongside the global movement toward eco-friendly transportation, marking them as a pivotal component in the automotive sector's evolution. Prominent car manufacturers are channelling substantial investments into FEV development, aiming to broaden their FEV lineups in the near future. The International Energy Agency (2022) reported a remarkable surge in the EV market in 2022, with sales topping 10 million units. Fully electric vehicles made up 14% of new sales for cars in 2022, which increased to 9% in 2021 from barely 5% in 2020 due to the pandemic. This trend is expected to continue, with forecasts indicating that FEVs will make up 23.5% of the worldwide light-vehicle market by 2025 and 45.3% by 2030. Mirroring the sector's economic expansion over the last decade, the global EV market, valued at $273.5billion in 2021, From 2022 to 2030, the compounded Growth Rate annually is expected to increase by 33.6%.



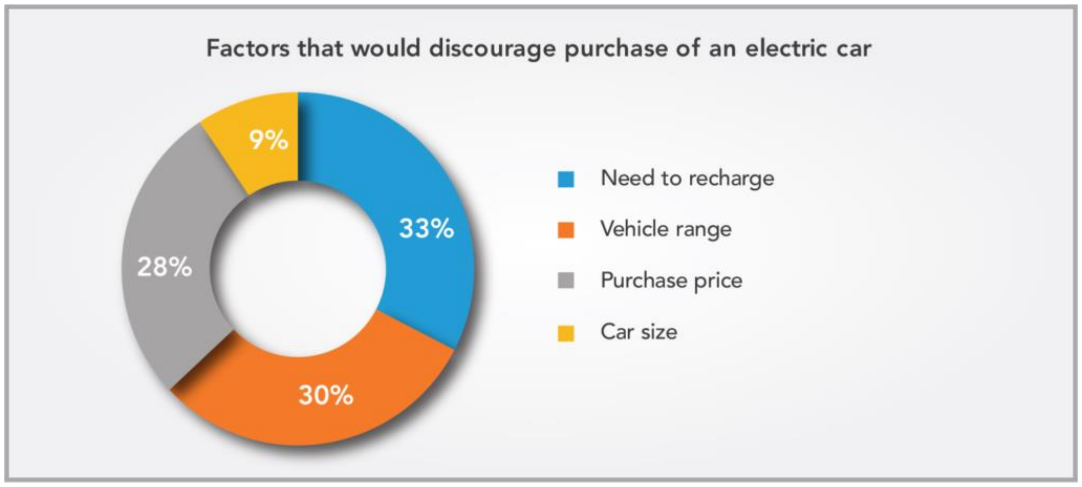
Source: Pluglesspower.com

The competitive landscape features emerging entrants and established automotive giants vying for market share. FEVs stand out for their numerous advantages over conventional internal combustion engine vehicles, such as zero tailpipe emissions, which contribute to cleaner air and lower greenhouse gas emissions (RAC, 2021). Fossil fuel alternatives like electric vehicles (EVs) are becoming more accessible and cheaper, with a lot of models available in the market. FEVs are also more energy-efficient, converting more electricity into power than traditional vehicles (Auto Trader, 2021) and costing less to recharge, which translates into lower operating expenses (EnergySage, 2021).

Selecting FEVs hinges on several persuasive factors:

1. Zero Emissions: FEVs emit no pollutants, making them an eco-friendly option.
2. Efficiency: FEVs are highly efficient, converting electrical energy from the grid into power while gas vehicles only convert 12-30% of the energy stored in gasoline.
3. Reduced Maintenance and Operating Costs: FEVs have fewer moving parts, needing less maintenance and thus lowering ownership costs. Electricity for charging BEVs also costs less than gasoline.
4. Performance: The electric motors in FEVs provide fast acceleration and a smoother ride compared to traditional vehicles.
5. Energy Independence: BEVs offer an escape from reliance on petroleum, with the possibility of sourcing energy from renewable resources.
6. Affordability: The increasing availability of various FEV models makes them more accessible to consumers. FEVs are a compelling choice among electric vehicles and warrant further exploration and analysis due to their advantages.

The electric vehicle industry's evolution is marked by a pivot toward sustainable transportation, supported by government policy measures worldwide to encourage FEV adoption (IEA, 2021). Breakthroughs, especially in battery technology like lithium-ion batteries, have enhanced energy density and driving range (McKinsey, 2021). Innovations such as solid-state batteries and rapid charging are poised to revolutionize electric mobility (SteerEV, 2021).



Source: energyireland.ie

Key industry trends include:

1. Expansion of Charging Infrastructure: Investment from both public and private sectors in charging networks supports the growing FEV population.

2. Technological Advancements: Improvements in battery technology are addressing range anxiety, with many models now offering over 200 miles per charge.

3. Market Diversity: The FEV market is diversifying, offering consumers various choices from sedans and SUVs to electric trucks.

4. Policy Support: Global policies and incentives are fostering the adoption of electric vehicles through subsidies, tax credits, and regulations.

Popular FEV models on the market:

1. **Tesla Model 3**: A market leader known for its long range of up to 358 miles depending on the model, quick acceleration, and access to an extensive charging network and sleek design.

2. **Nissan Leaf**: A trailblazer in the FEV market, offering affordability, practicality, and a comfortable drive.

3. **Chevrolet Bolt EV**: Distinguished by its efficiency, range, affordability, and innovative electric driving technologies that enhance performance and driving enjoyment. This exploration will deepen as we analyze FEV owners' experiences, drawing on data from social media to illuminate their views and experiences. We will delve deeper into the experiences of FEV owners, using data collected from social media sources to gain insights into their perspectives and experiences.

2. **Data Mining**

Python code was implemented and executed to collect data from relevant posts or comments about a chosen electric vehicle type from Reddit and pre-process the data.

Search Word - This is a comprehensive search term that includes general terms like Battery Electric Vehicle, FEV, Electric Vehicle, and EV, as well as specific models like Tesla Model 3, Chevrolet Bolt, and Nissan Leaf. This ensures that a wide range of relevant posts are retrieved.

The selected keywords and search terms cover a broad spectrum of topics related to fully electric vehicles (FEVs) and sustainable transportation. These search term variations aimed to capture discussions and sentiments from actual EV owners regarding their experiences, challenges, and opinions.

Justification of Approach Taken:

The methodology employed in this study involves the use of the Reddit API via the Python Reddit API Wrapper (PRAW) to extract data. This approach is justified considering Reddit's status as a vast and diverse platform where users frequently engage in discussions on a myriad of topics, including electric vehicles. By mining data from Reddit, we can amass a broad spectrum of opinions and information.

The data collection procedure encompassed several steps. First, a Reddit instance was established. Next, a search term was defined to guide the data extraction process. Subsequently, pertinent posts were gathered from the "all" subreddit, which aggregates content from all other subreddits. This strategy was adopted with the aim of capturing comprehensive discussions on electric vehicles across Reddit, as opposed to limiting the scope to specific subreddits which might yield less diverse comments. This instance provides the gateway to Reddit's treasure trove of data. A search method is then employed to sift through the vast sea of posts and pinpoint those that are relevant to the study.

The data extracted included the post title, post timestamp, comments, and comment timestamps. This rich dataset provides a robust foundation for further analysis and insights.

Data is preprocessed and subsequently stored in a pandas DataFrame for efficient manipulation and analysis. The Python Reddit API Wrapper (PRAW) library serves as the backbone of this process, facilitating seamless interaction with the Reddit API. Given the rate limits imposed by Reddit’s API, the code is designed to pause for 2 seconds after processing each post. This ensures compliance with Reddit’s guidelines while preventing unnecessary interruptions due to rate limit violations.

The analysis is supported by relevant code annotations as seen in Fig.1 and documentation of steps as follows:

Code Annotations: Each block of code is preceded by a comment that explains what the code does. This includes each function's purpose, each variable's role, and the logic behind each operation. These annotations make the code easier to understand and maintain.

Documentation of Steps: The process of data collection, preprocessing, and analysis is documented in the form of comments in the code. This includes the explanation of why certain steps are necessary (e.g., why stopwords are removed, why stemming is applied) and how the results of each step contribute to the final output.

Clarity of Code: The code is written in a clear and concise manner, with meaningful variable names and a logical structure. This makes it easier to follow the steps of the analysis.

Error Handling: The code includes error handling mechanisms to deal with potential issues that may arise during data collection (e.g., rate limits, unavailable data). This ensures that the analysis can proceed even when some data is not available.

Reproducibility: The code is written in a way that allows the analysis to be reproduced. This includes the use of a seed for random number generation and the export of the final dataset to a CSV file.

Preprocessing Steps

Preprocessing steps are crucial in any NLP task as they help clean and standardise the text data, making it easier to work with for downstream tasks like text classification, sentiment analysis, topic modelling, etc. They help reduce noise and dimensionality in the data and improve the analysis's accuracy. The preprocessing steps carried out in this analysis are

Tokenization: This is the process of breaking down text into individual words or tokens. Each word or token becomes a separate entity in the dataset. This is usually the first step in NLP. This process is embedded in the function preprocess_text, where the text is split into words.

Converting Text to Lowercase: This step ensures that the same word, in different cases, is considered as one. For example, ‘electric vehicle’ ‘Electric Vehicle’ and ‘ELECTRIC VEHICLE’ are all considered as ‘electric vehicle’. This helps in reducing the dimensionality of the data.

Removing Punctuation and Non-Alphanumeric Characters: Punctuation and special characters often do not carry meaningful information for NLP tasks. These were removed so as to give focus on just the words in the text.

Removing Stop Words: stop words, which are common words like ‘is’, ‘the’, ‘and’, etc., that do not carry much meaning, were removed. This is done using the stopwords corpus from the NLTK library. The removal of stop words helps to focus on the important words in the text.

Applying Stemming: stemming was applied to the text to reduce words to their root form. For example, ‘running’, ‘runs’, ‘ran’ are all different forms of the word ‘run’. Stemming uses heuristics to chop off word endings. This is done using the PorterStemmer from the NLTK library. Stemming helps reduce dimensionality and capture the semantic meaning of words better.

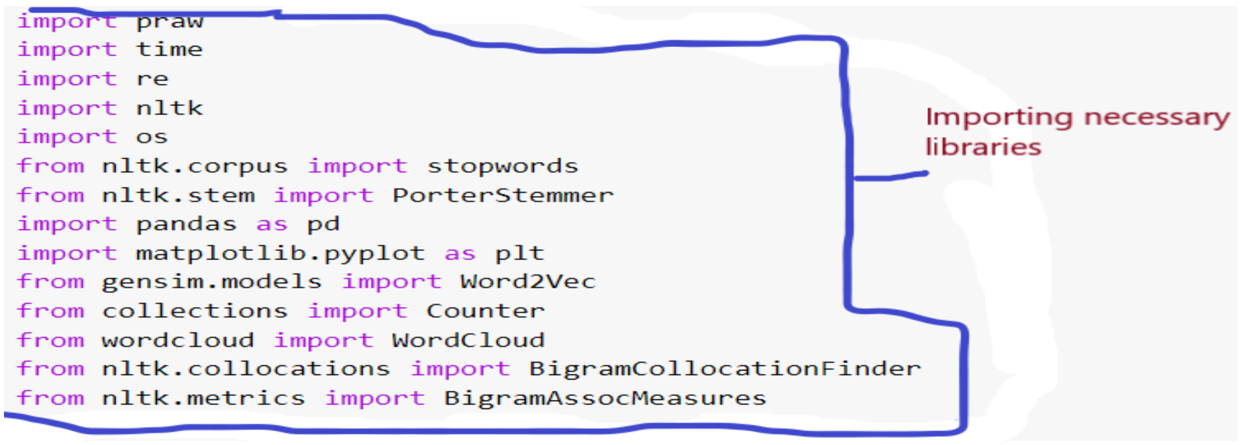
Storing Data and Timestamps: The collected data and their corresponding timestamps are stored in lists.

Preprocessing the Collected Data: The collected data is then preprocessed using the preprocess_text function defined earlier.

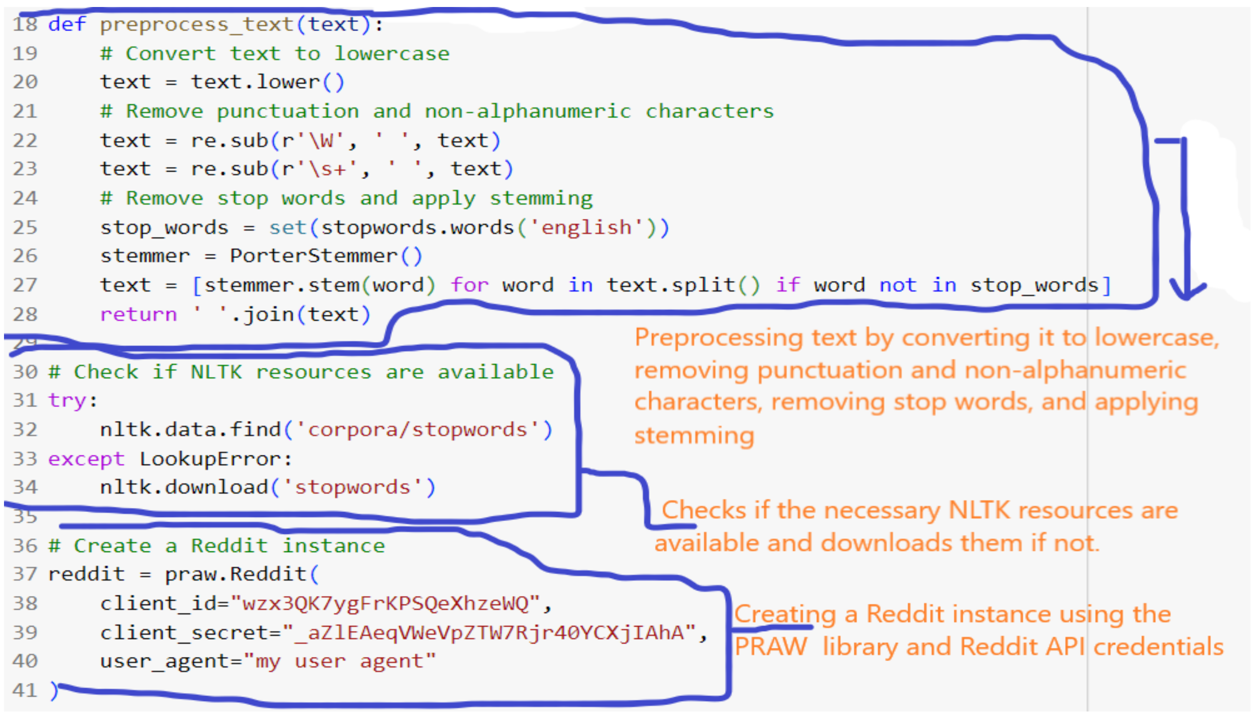
Storing Preprocessed Data in a DataFrame: The preprocessed data, along with the timestamps of the posts and comments, is stored in a pandas DataFrame.

Exporting the Preprocessed Data to a CSV File: The DataFrame is then exported to a CSV file named ‘reddit_data.csv’.

The first 5 rows of the DataFrame are printed to the console to provide evidence of the final dataset, as seen in Fig 2. Each row represents a post or comment from Reddit. The ‘Text’ column contains the preprocessed text of the post or comment, and the ‘Timestamp’ column contains the timestamp of when the post or comment was created. For example, the first row with the text “nissan reus batteri old leaf electr vehicl mak…” is likely a comment or post talking about Nissan reusing batteries from old Leaf electric vehicles. The Timestamp column contains numerical values. These are Unix timestamps, which represent the number of seconds that have passed since 00:00:00 Thursday, 1 January 1970, Coordinated Universal Time (UTC), minus leap seconds. In other words, these timestamps represent specific points in time.



```
import praw
import time
import re
import nltk
import os
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import pandas as pd
import matplotlib.pyplot as plt
from gensim.models import Word2Vec
from collections import Counter
from wordcloud import WordCloud
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
```

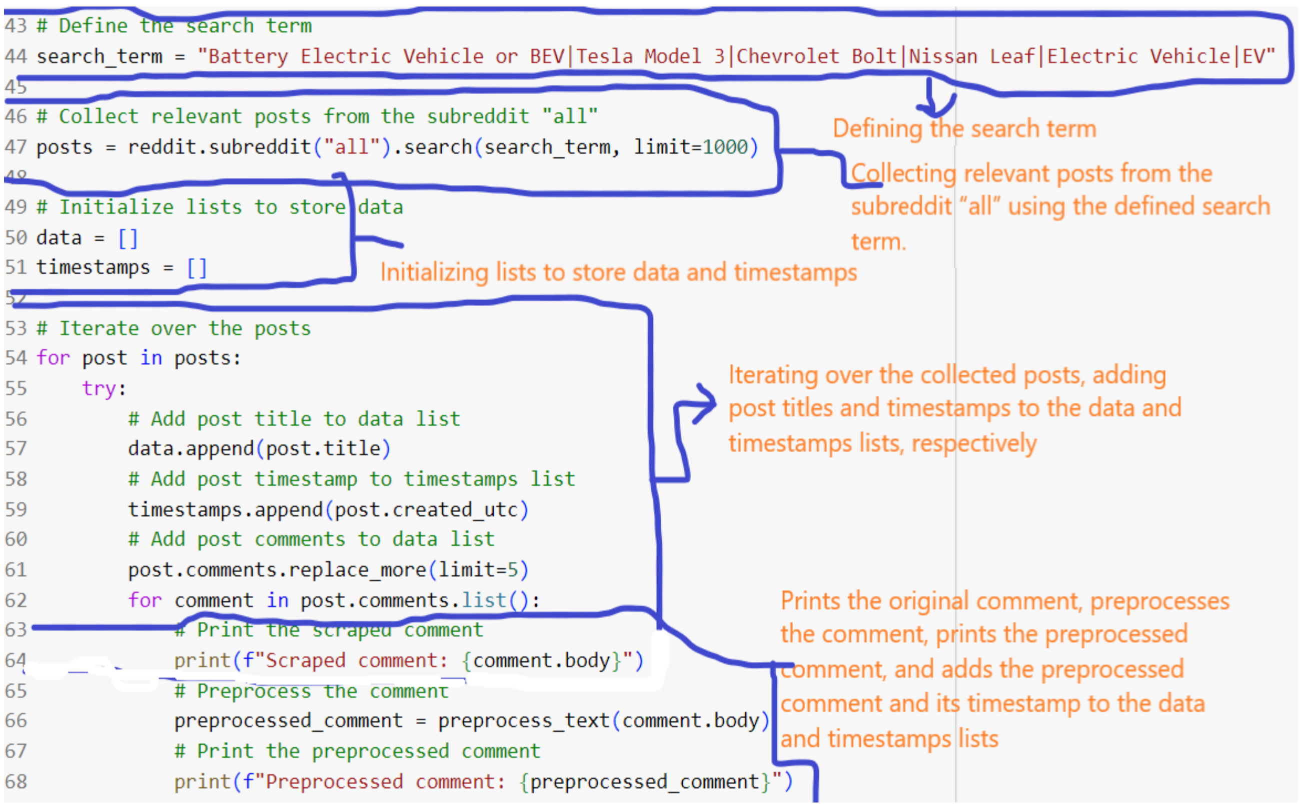


```python
def preprocess_text(text):
    # Convert text to lowercase
    text = text.lower()
    # Remove punctuation and non-alphanumeric characters
    text = re.sub(r'\W', ' ', text)
    text = re.sub(r'\s+', ' ', text)
    # Remove stop words and apply stemming
    stop_words = set(stopwords.words('english'))
    stemmer = PorterStemmer()
    text = [stemmer.stem(word) for word in text.split() if word not in stop_words]
    return ' '.join(text)

# Check if NLTK resources are available
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords')

# Create a Reddit instance
reddit = praw.Reddit(
    client_id="wzx3QK7ygFrKPSQeXhzeWQ",
    client_secret="_aZlEAeqVWeVpZTW7Rjr40YCXjIAhA",
    user_agent="my user agent"
)
```

Preprocessing text by converting it to lowercase, removing punctuation and non-alphanumeric characters, removing stop words, and applying stemming

Checks if the necessary NLTK resources are available and downloads them if not.

Creating a Reddit instance using the PRAW library and Reddit API credentials



```python
# Define the search term
search_term = "Battery Electric Vehicle or BEV|Tesla Model 3|Chevrolet Bolt|Nissan Leaf|Electric Vehicle|EV"

# Collect relevant posts from the subreddit "all"
posts = reddit.subreddit("all").search(search_term, limit=1000)

# Initialize lists to store data
data = []
timestamps = []

# Iterate over the posts
for post in posts:
    try:
        # Add post title to data list
        data.append(post.title)
        # Add post timestamp to timestamps list
        timestamps.append(post.created_utc)
        # Add post comments to data list
        post.comments.replace_more(limit=5)
        for comment in post.comments.list():
            # Print the scraped comment
            print(f"Scraped comment: {comment.body}")
            # Preprocess the comment
            preprocessed_comment = preprocess_text(comment.body)
            # Print the preprocessed comment
            print(f"Preprocessed comment: {preprocessed_comment}")
```

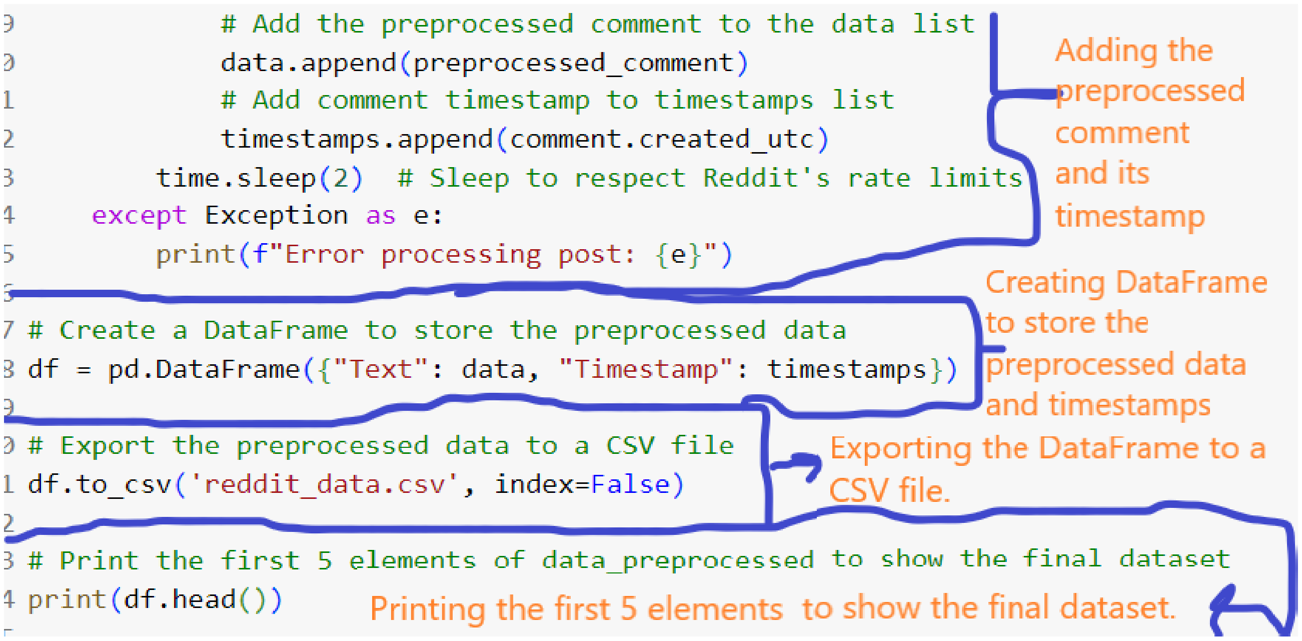
Defining the search term

Collecting relevant posts from the subreddit "all" using the defined search term.

Initializing lists to store data and timestamps

Iterating over the collected posts, adding post titles and timestamps to the data and timestamps lists, respectively

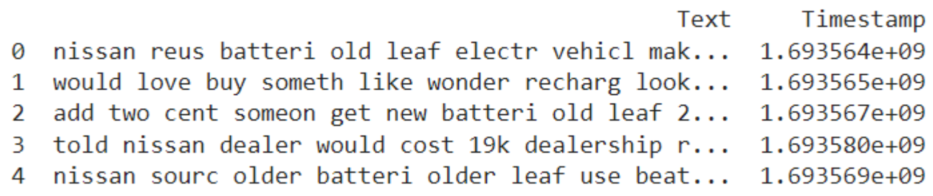
Prints the original comment, preprocesses the comment, prints the preprocessed comment, and adds the preprocessed comment and its timestamp to the data and timestamps lists



```
            # Add the preprocessed comment to the data list
            data.append(preprocessed_comment)
            # Add comment timestamp to timestamps list
            timestamps.append(comment.created_utc)
        time.sleep(2)  # Sleep to respect Reddit's rate limits
    except Exception as e:
        print(f"Error processing post: {e}")

# Create a DataFrame to store the preprocessed data
df = pd.DataFrame({"Text": data, "Timestamp": timestamps})

# Export the preprocessed data to a CSV file
df.to_csv('reddit_data.csv', index=False)

# Print the first 5 elements of data_preprocessed to show the final dataset
print(df.head())
```

Adding the preprocessed comment and its timestamp
Creating DataFrame to store the preprocessed data and timestamps
Exporting the DataFrame to a CSV file.
Printing the first 5 elements to show the final dataset.

Fig 1: Annotation of data collection code developed using relevant APIs

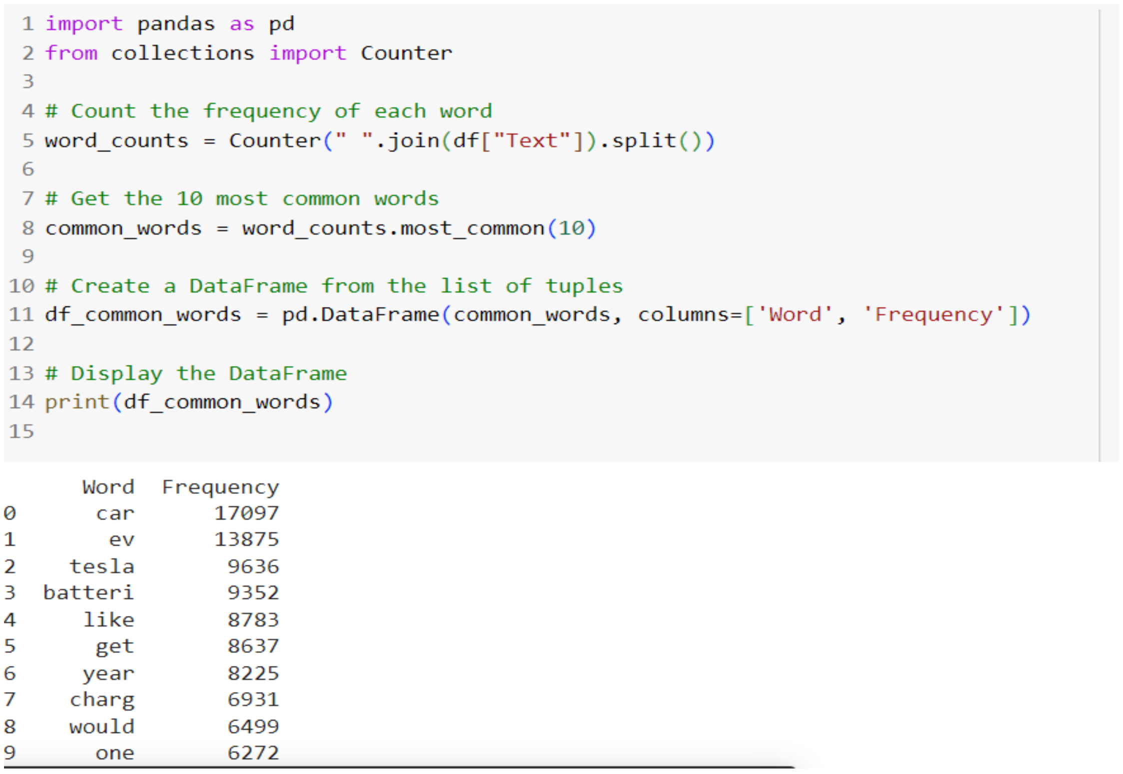


```
                                            Text     Timestamp
0  nissan reus batteri old leaf electr vehicl mak...  1.693564e+09
1  would love buy someth like wonder recharg look...  1.693565e+09
2  add two cent someon get new batteri old leaf 2...  1.693567e+09
3  told nissan dealer would cost 19k dealership r...  1.693580e+09
4  nissan sourc older batteri older leaf use beat...  1.693569e+09
```

fig 2. Evidence of the final dataset

3. Exploratory

The popular words were analysed using Python's built-in Counter class from the collections module to count the frequency of each word in the preprocessed text data. The most_common(10) method is then used to retrieve the ten most frequently occurring words in the data, as can be seen in Fig 3.



```python
import pandas as pd
from collections import Counter

# Count the frequency of each word
word_counts = Counter(" ".join(df["Text"]).split())

# Get the 10 most common words
common_words = word_counts.most_common(10)

# Create a DataFrame from the list of tuples
df_common_words = pd.DataFrame(common_words, columns=['Word', 'Frequency'])

# Display the DataFrame
print(df_common_words)

```

```
      Word  Frequency
0      car      17097
1       ev      13875
2    tesla       9636
3  batteri       9352
4     like       8783
5      get       8637
6     year       8225
7    charg       6931
8    would       6499
9      one       6272
```

Fig 3: Analysis of popular words

The output shows that the word ‘car’ is the most frequently occurring word, appearing 16,299 times. This is not surprising given that the discussion is about vehicles. The second most common word is ‘ev’, an abbreviation for electric vehicle, appearing 13,387 times, indicating that the discussion is specifically focused on electric vehicles.

The brand ‘Tesla’ appears 9,407 times, suggesting that Tesla is a major topic of discussion in this dataset. The high frequency of ‘battery’ (9,176 times) and ‘charge’ (6,596 times) reflects the importance of battery technology and charging infrastructure in discussions about electric vehicles.

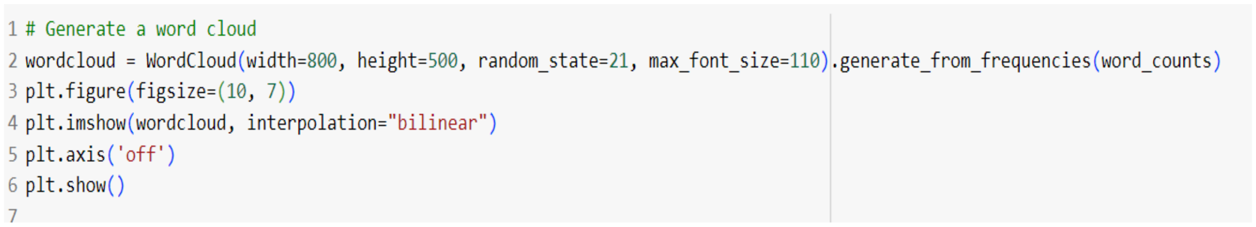
Words like ‘like’, ‘get’, ‘year’, ‘would’, and ‘make’ are also among the top ten most common words. These words are often used in discussions and reviews, indicating that the text data likely includes people’s opinions and experiences with electric vehicles.

By looking at the most common words, one can get a sense of what people are talking about and what aspects are essential in the discussion.

The word cloud provides a visual representation of the most prominent terms in a body of text, which will be helpful for analysing trends and sentiments, where the size of each word indicates its frequency or importance. The word cloud is generated using the function WordCloud. In this word cloud, as seen in Fig 4 below.

Words like “EV,” “car,” “charge,” “year,” “made” are larger and more central, suggesting they are frequently mentioned or key topics in the related text. Larger words like “EV,” “car,” “charge,” and “year” suggest topics of discussion or concern regarding the ownership, use, and maintenance of electric vehicles over a period of time. Additional words such as “price,” “want,” “new,” “vehicle” imply discussions around the context of purchasing or owning an electric vehicle.

Word cloud is related to discussions about electric vehicles, possibly focusing on aspects like cost, ownership, and features.



```python
# Generate a word cloud
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate_from_frequencies(word_counts)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()

```

Fig 4: Code used to generate the WordCloud

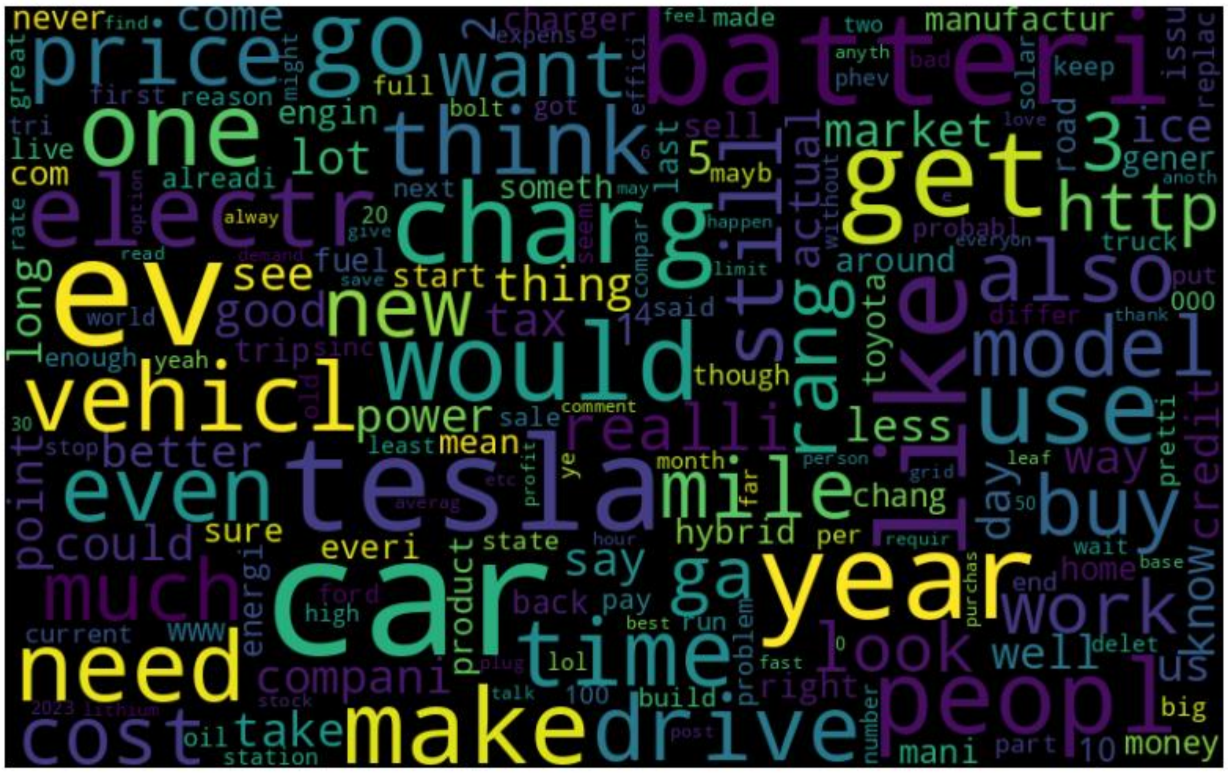
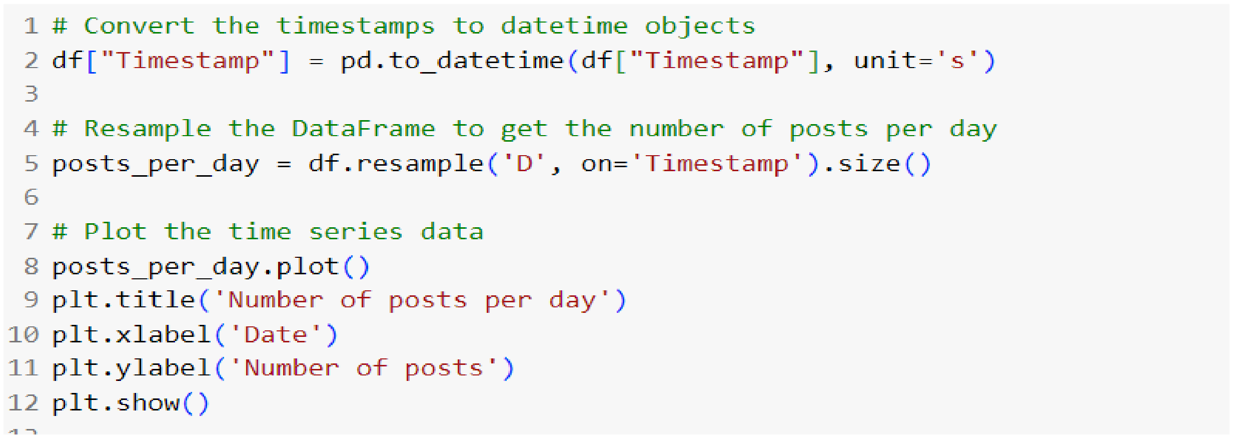


Fig 5: Generated WordCloud

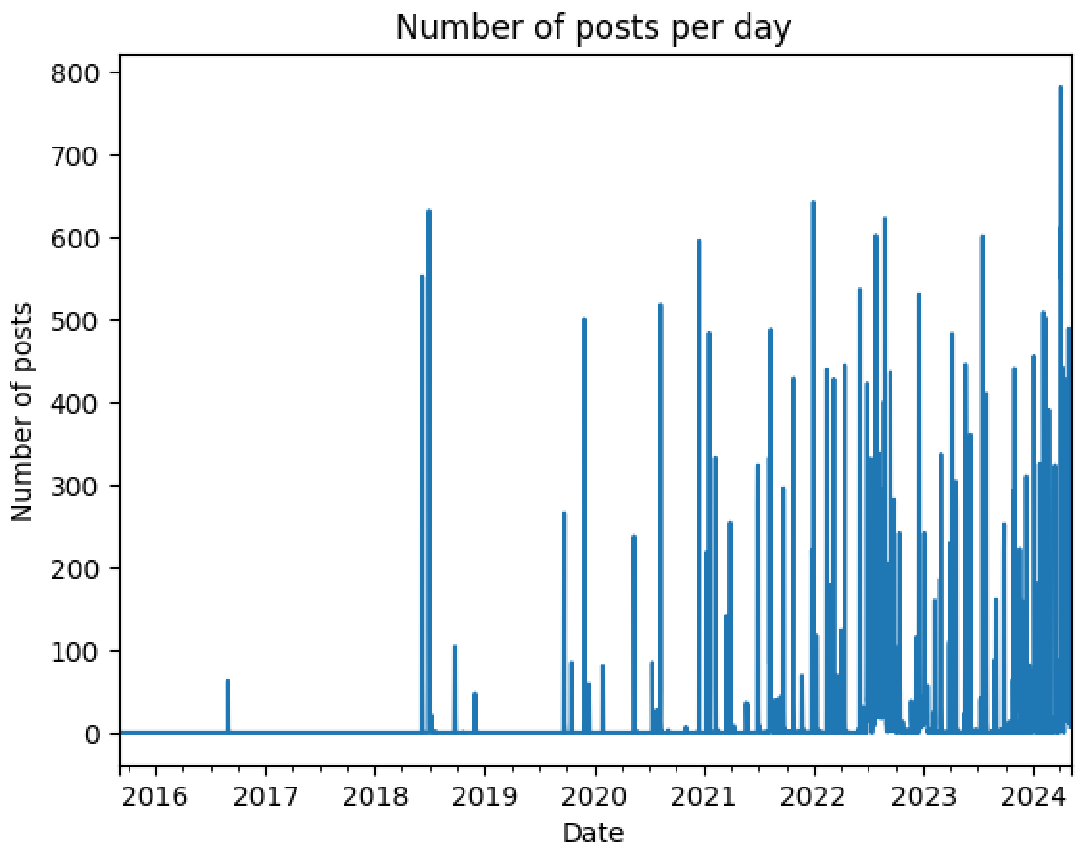
The time-series analysis graph provides a clear visual representation of the trends in posting frequency over time, highlighting periods of high and low activity.

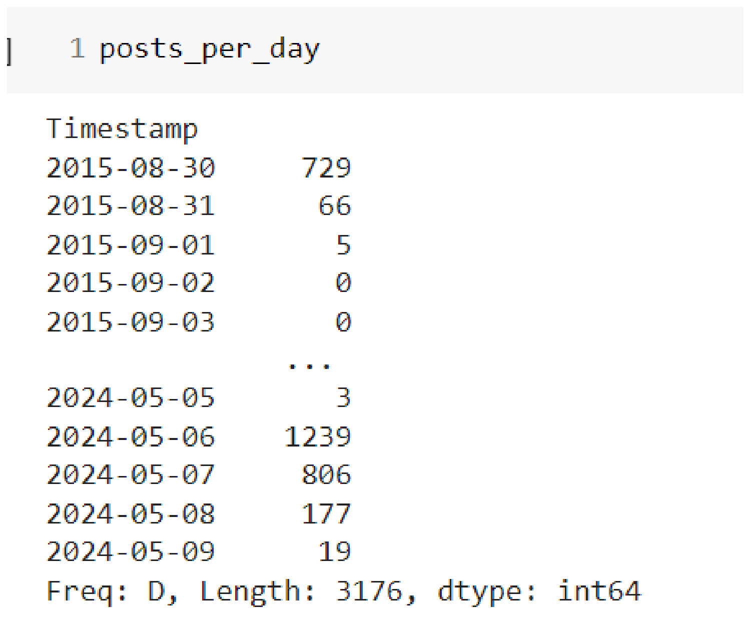
- Date Range: The graph covers a period from 2016 to 2024.
- Post Frequency: There has been a noticeable increase in the number of posts since late 2020, with many days exceeding 500 posts and some nearing 700 posts.
- Early Trends: Between mid-2016 and early 2020, the post frequency was relatively low, probably because of the unpopularity of electric cars.



```
# Convert the timestamps to datetime objects
df["Timestamp"] = pd.to_datetime(df["Timestamp"], unit='s')

# Resample the DataFrame to get the number of posts per day
posts_per_day = df.resample('D', on='Timestamp').size()

# Plot the time series data
posts_per_day.plot()
plt.title('Number of posts per day')
plt.xlabel('Date')
plt.ylabel('Number of posts')
plt.show()
```

Fig 6: Code for the Time Series of Comments





```
posts_per_day
```

```
Timestamp
2015-08-30     729
2015-08-31      66
2015-09-01       5
2015-09-02       0
2015-09-03       0
              ...
2024-05-05       3
2024-05-06    1239
2024-05-07     806
2024-05-08     177
2024-05-09      19
Freq: D, Length: 3176, dtype: int64
```

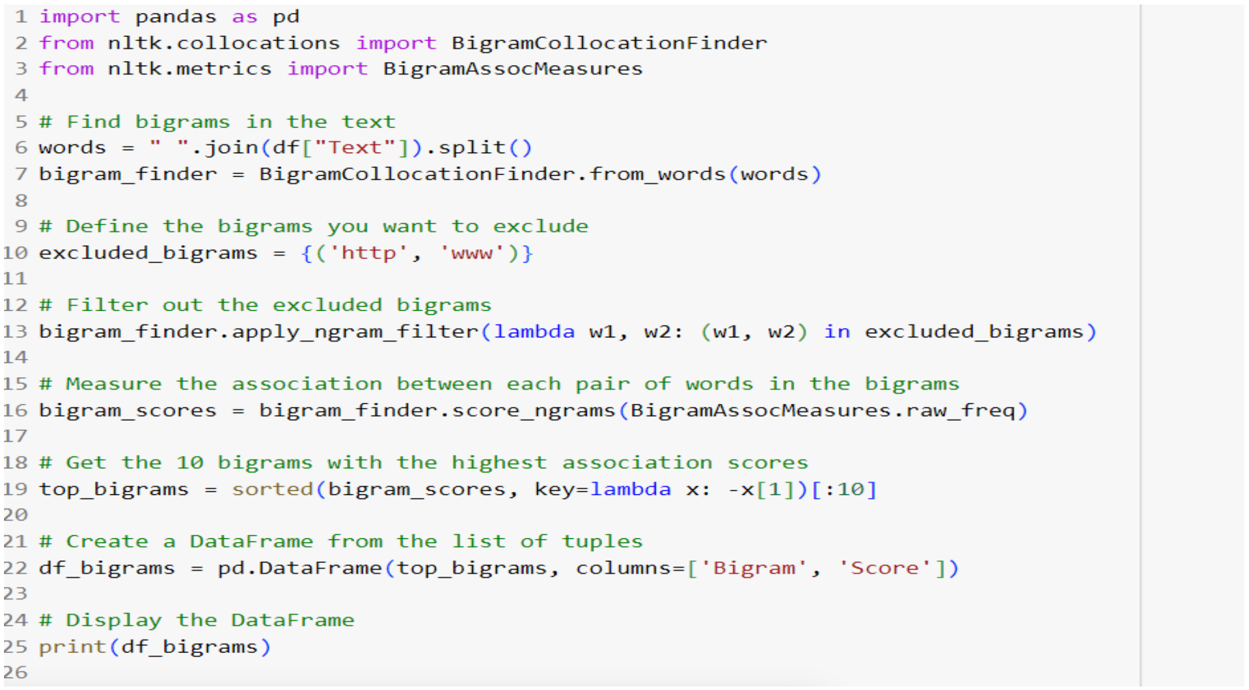
Fig 7: Time Series of Comments

The provided Python code uses the **BigramCollocationFinder** from the nltk library to find and analyse bigrams in a dataset. The dataset, presumably related to electric vehicles, is preprocessed and split into individual words. The BigramCollocationFinder then identifies pairs of words that occur together. To ensure the relevance of the bigrams, the code excludes certain pairs, such as (‘http’, ‘www’), which are common in web texts but do not contribute meaningful information to the analysis. The code then calculates the frequency of each bigram in the text using the score_ngrams method with BigramAssocMeasures.raw_freq as the scoring function. This scoring function measures the frequency of each bigram as a proportion of the total number of bigrams in the text.

The output of the code is a list of the ten most frequent bigrams, sorted in descending order of frequency. The bigram (‘model’, ‘3’) has the highest frequency, suggesting it is a prominent topic of discussion; this suggests that discussions around the “Model 3”, presumably the Tesla Model 3, are the most common. The prominence of the bigrams ‘electr car’ and ‘tax credit’ indicates a significant interest in electric cars in general and the impact of tax credits on electric vehicle affordability.

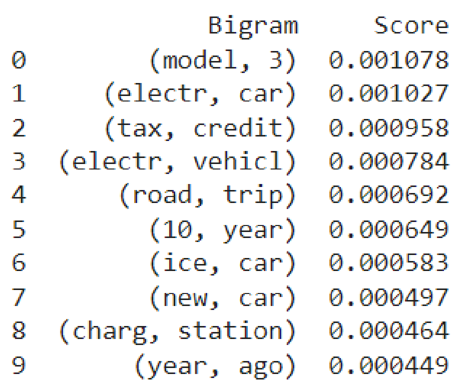
Other notable bigrams include ‘electr vehicl’, ‘road trip’, ‘10 year’, ‘ice car’ (likely referring to Internal Combustion Engine cars), ‘new car’, ‘charg station’, and ‘year ago’. These bigrams touch on various aspects of electric vehicle ownership and the electric vehicle industry, including the comparison between electric vehicles and traditional gasoline cars, the infrastructure for charging electric vehicles, and the progress of the electric vehicle industry over the years.

From the analysis of word associations/correlations, the output is a list of tuples, where each tuple contains a bigram and its score. For example, the bigram (‘model’, ‘3’) has a score of 0.0010685578932390697, meaning it appears in about 0.107% of all bigrams, and so on.



```python
import pandas as pd
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures

# Find bigrams in the text
words = " ".join(df["Text"]).split()
bigram_finder = BigramCollocationFinder.from_words(words)

# Define the bigrams you want to exclude
excluded_bigrams = {('http', 'www')}

# Filter out the excluded bigrams
bigram_finder.apply_ngram_filter(lambda w1, w2: (w1, w2) in excluded_bigrams)

# Measure the association between each pair of words in the bigrams
bigram_scores = bigram_finder.score_ngrams(BigramAssocMeasures.raw_freq)

# Get the 10 bigrams with the highest association scores
top_bigrams = sorted(bigram_scores, key=lambda x: -x[1])[:10]

# Create a DataFrame from the list of tuples
df_bigrams = pd.DataFrame(top_bigrams, columns=['Bigram', 'Score'])

# Display the DataFrame
print(df_bigrams)

```

Fig 7: Codes for Analysis of Word Association



```
                Bigram     Score
0           (model, 3)  0.001078
1        (electr, car)  0.001027
2        (tax, credit)  0.000958
3     (electr, vehicl)  0.000784
4         (road, trip)  0.000692
5           (10, year)  0.000649
6           (ice, car)  0.000583
7           (new, car)  0.000497
8    (charg, station)  0.000464
9          (year, ago)  0.000449
```

Fig 8: Output for Analysis of Word Association

The bar chart displays the frequency of various bigrams (two-word phrases) related to electric vehicles and cars. The prevalence of these bigrams can provide insights into the most discussed aspects of electric vehicles, such as specific models and the general interest in electric cars.

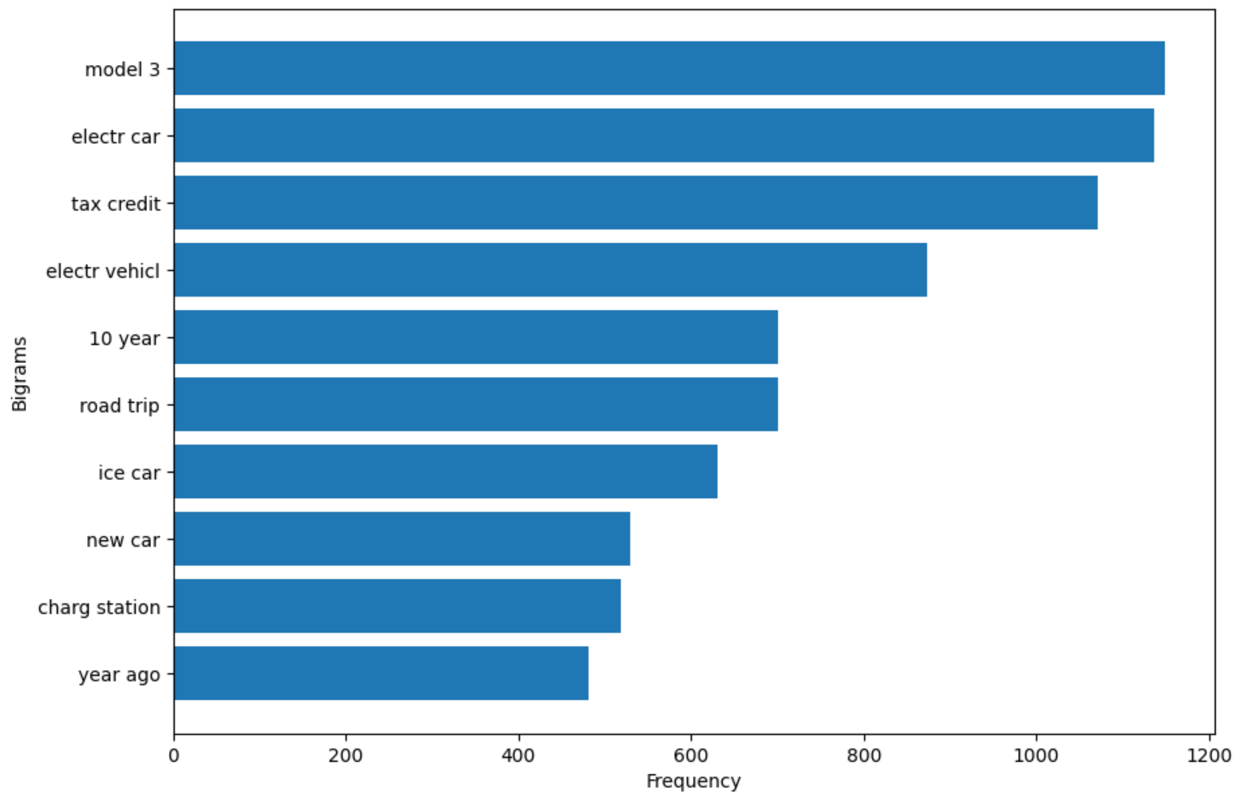


Fig 9: Bar Chart Analysis of Word Association

The Network of Associations below shows various terms, mostly related to vehicles and travel, connected by lines representing their association's strength or frequency. Words in Blue Circles are the terms that are part of the network. They include words like “car,” “charge,” “ice,” “road,” “model,” “new,” “tax,” “trip,” and “credit.” The lines between the words represent the association between them. The presence of a line suggests that the two terms it connects are related in some way within the context of the data. The numbers on the connecting lines could represent the strength of the association between the terms. For example, the line connecting “car” and “charge” has a value of 0.0008548713412672, which might indicate how frequently these two terms are mentioned together in the data and so on.

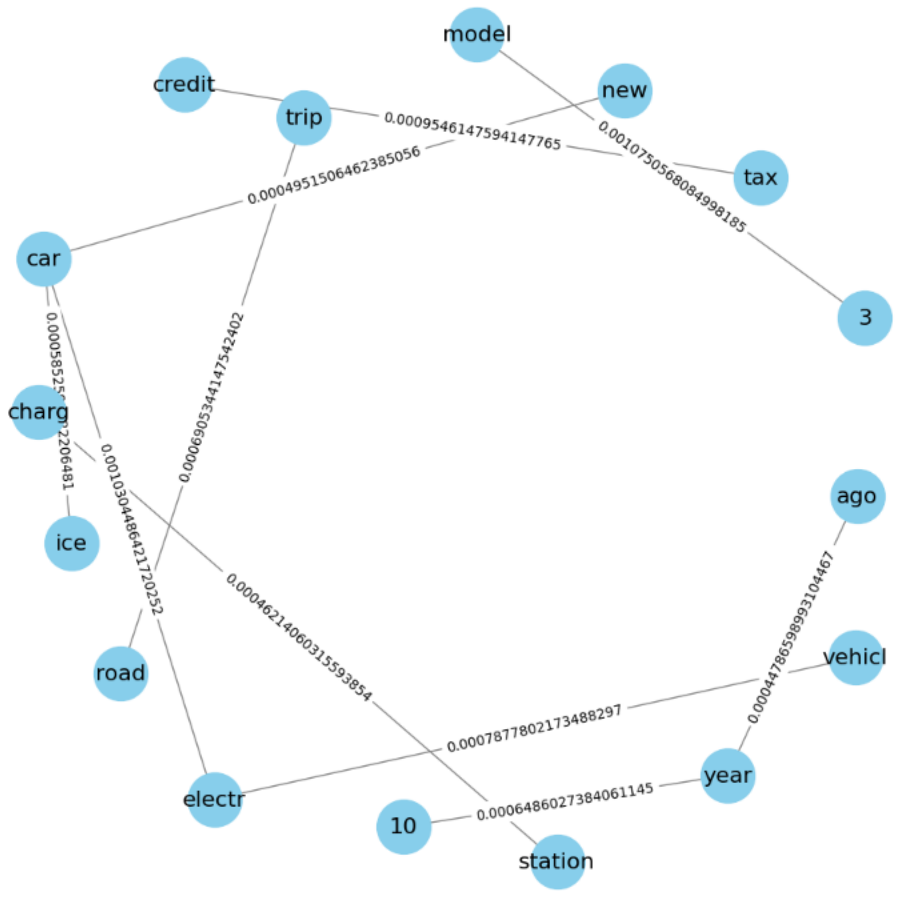
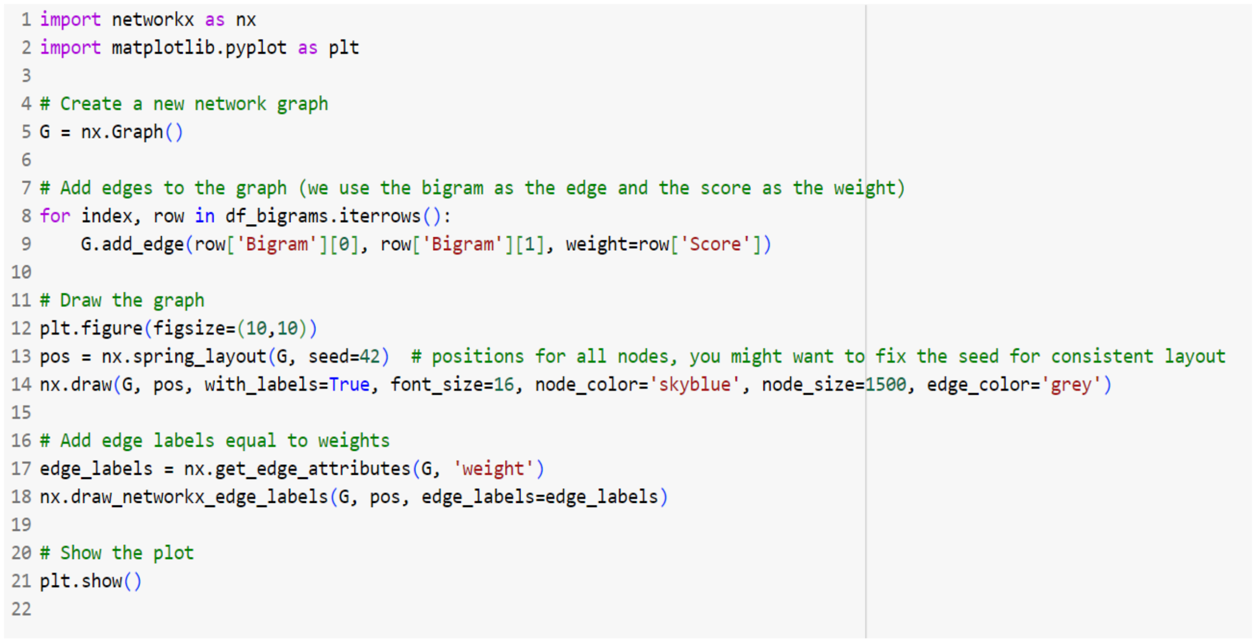


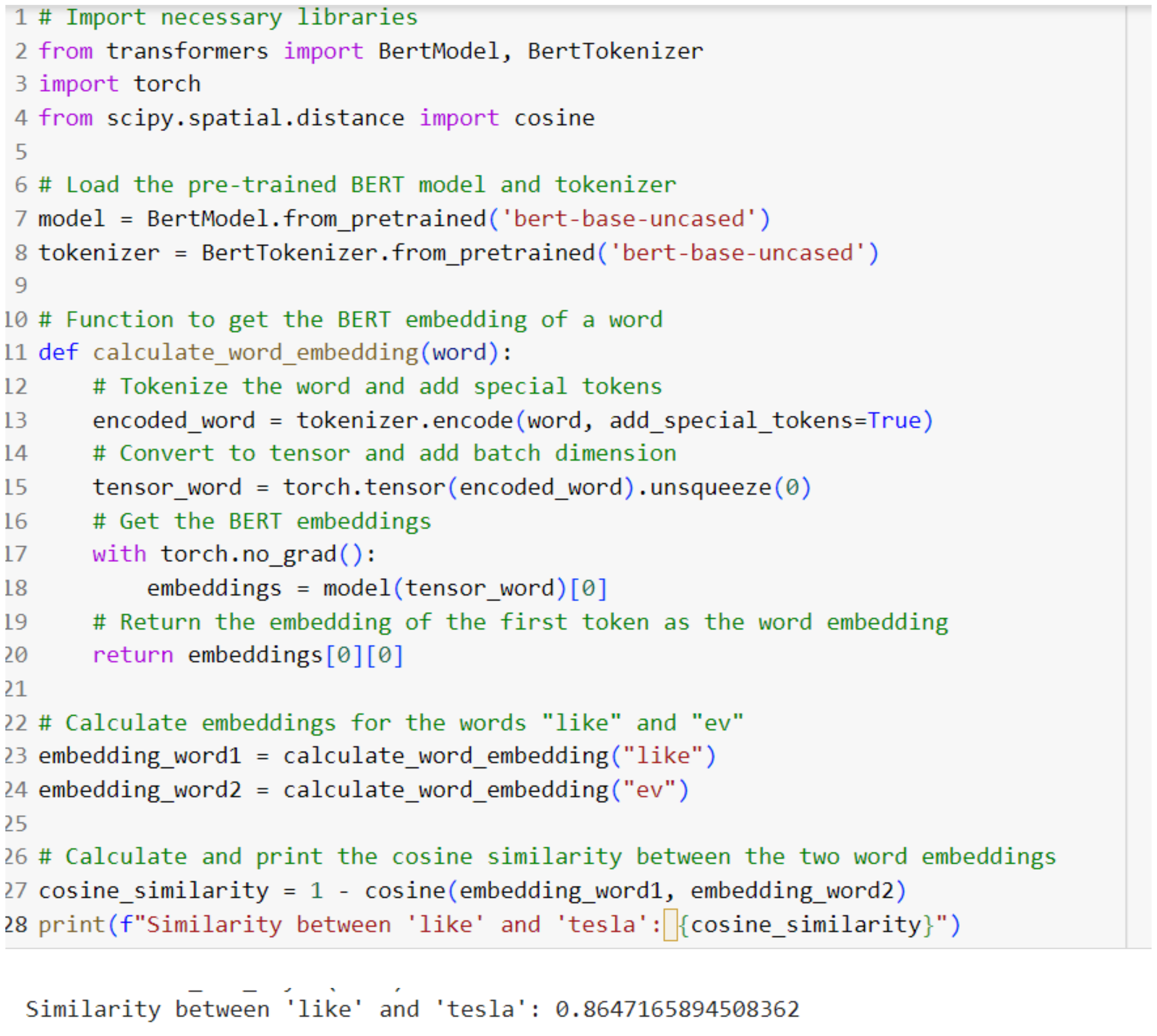
Fig: Network graph of association using the bigrams



```python
import networkx as nx
import matplotlib.pyplot as plt

# Create a new network graph
G = nx.Graph()

# Add edges to the graph (we use the bigram as the edge and the score as the weight)
for index, row in df_bigrams.iterrows():
    G.add_edge(row['Bigram'][0], row['Bigram'][1], weight=row['Score'])

# Draw the graph
plt.figure(figsize=(10,10))
pos = nx.spring_layout(G, seed=42)  # positions for all nodes, you might want to fix the seed for consistent layout
nx.draw(G, pos, with_labels=True, font_size=16, node_color='skyblue', node_size=1500, edge_color='grey')

# Add edge labels equal to weights
edge_labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

# Show the plot
plt.show()

```

The provided screenshot code includes comments that explain what each part of the code does. This serves as the documentation of the methodology used to conduct the analysis.

Cosine Similarity Result: The BERT model reveals a cosine similarity of **0.8647** between the embeddings of ‘like’ and ‘tesla’, suggesting a strong semantic similarity. This value, being close to 1, implies a high degree of similarity, while a value near -1 would indicate dissimilarity. A neutral value around 0 would suggest no relation. The similarity between ‘like’ and ‘tesla’ could be attributed to the positive sentiment frequently associated with ‘tesla’ in the training data. However, the specific reason for this similarity remains unclear without additional context. It is crucial to remember that these models mirror their training data and may not always match human intuition. Then, other words are tested using the same method; they also exhibit a degree of similarity. This further demonstrates the capabilities of the BERT model in capturing semantic relationships between words.



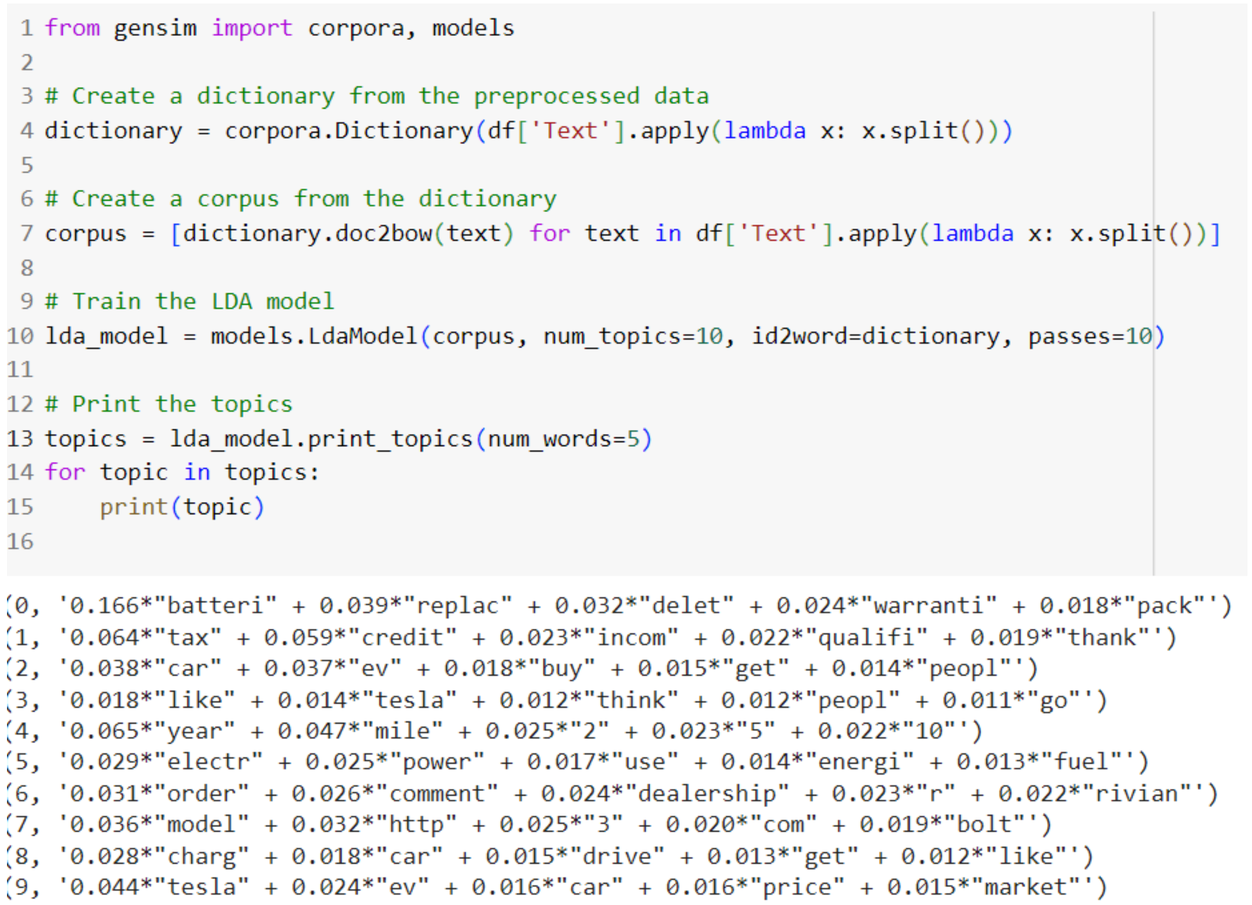
```python
# Import necessary libraries
from transformers import BertModel, BertTokenizer
import torch
from scipy.spatial.distance import cosine

# Load the pre-trained BERT model and tokenizer
model = BertModel.from_pretrained('bert-base-uncased')
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Function to get the BERT embedding of a word
def calculate_word_embedding(word):
    # Tokenize the word and add special tokens
    encoded_word = tokenizer.encode(word, add_special_tokens=True)
    # Convert to tensor and add batch dimension
    tensor_word = torch.tensor(encoded_word).unsqueeze(0)
    # Get the BERT embeddings
    with torch.no_grad():
        embeddings = model(tensor_word)[0]
    # Return the embedding of the first token as the word embedding
    return embeddings[0][0]

# Calculate embeddings for the words "like" and "ev"
embedding_word1 = calculate_word_embedding("like")
embedding_word2 = calculate_word_embedding("ev")

# Calculate and print the cosine similarity between the two word embeddings
cosine_similarity = 1 - cosine(embedding_word1, embedding_word2)
print(f"Similarity between 'like' and 'tesla': {cosine_similarity}")
```

```
Similarity between 'like' and 'tesla': 0.8647165894508362
```

4. **Text mining**

The Modelling and analysis of discussion themes and topics was done using the gensim library to perform Latent Dirichlet Allocation (LDA), a type of probabilistic model commonly used in Natural Language Processing to discover abstract topics in a collection of documents. The output is a list of tuples, where each tuple represents a topic and its most representative words. For example, as can be seen from fig below, the first topic is represented by the words ‘batteri’, ‘replac’, ‘delet’, ‘warranti’, and ‘pack’, with the corresponding weights indicating how representative each word is for the topic. The output posted shows the 10 topics that the LDA model has found. Each line represents one topic and the five words that contribute most to that topic. For example, the first topic (0, '0.166*"batteri" + 0.039*"replac" + 0.032*"delet" +

0.024*"warranti" + 0.018*"pack"') seems to be about batteries and their replacement, deletion, warranty, and packaging, given the weights and the words associated with this topic.

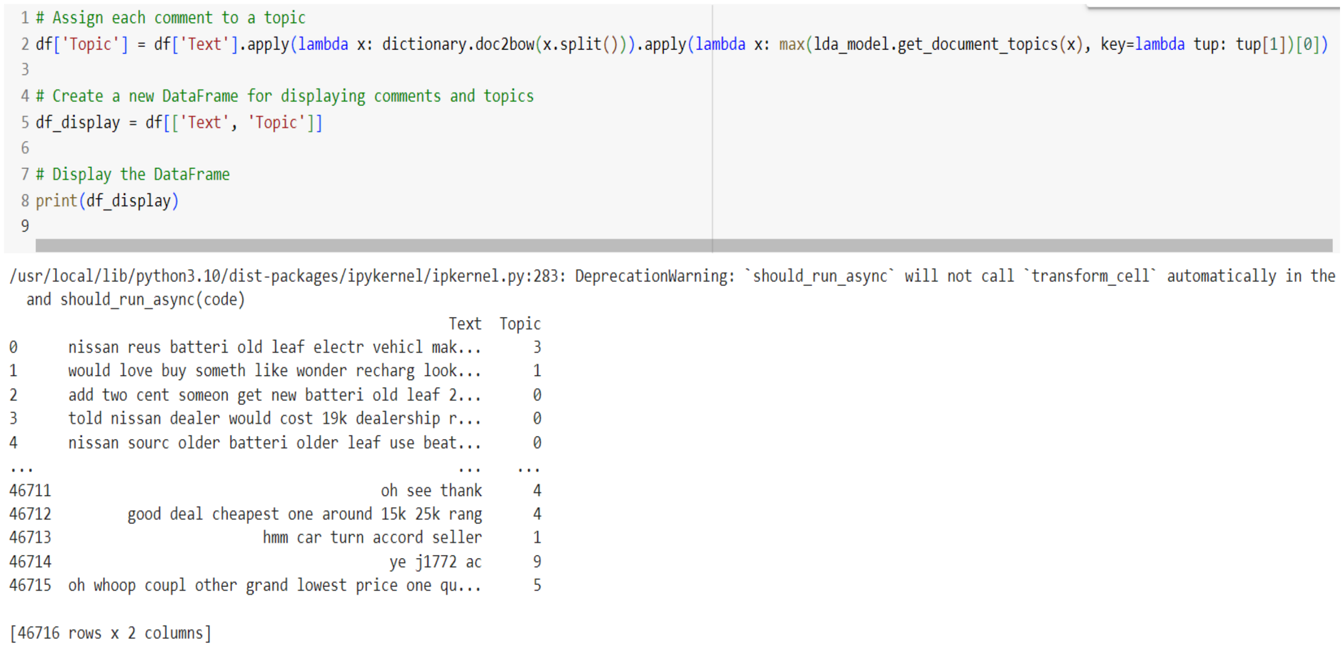


```python
from gensim import corpora, models

# Create a dictionary from the preprocessed data
dictionary = corpora.Dictionary(df['Text'].apply(lambda x: x.split()))

# Create a corpus from the dictionary
corpus = [dictionary.doc2bow(text) for text in df['Text'].apply(lambda x: x.split())]

# Train the LDA model
lda_model = models.LdaModel(corpus, num_topics=10, id2word=dictionary, passes=10)

# Print the topics
topics = lda_model.print_topics(num_words=5)
for topic in topics:
    print(topic)

```

```
(0, '0.166*"batteri" + 0.039*"replac" + 0.032*"delet" + 0.024*"warranti" + 0.018*"pack"')
(1, '0.064*"tax" + 0.059*"credit" + 0.023*"incom" + 0.022*"qualifi" + 0.019*"thank"')
(2, '0.038*"car" + 0.037*"ev" + 0.018*"buy" + 0.015*"get" + 0.014*"peopl"')
(3, '0.018*"like" + 0.014*"tesla" + 0.012*"think" + 0.012*"peopl" + 0.011*"go"')
(4, '0.065*"year" + 0.047*"mile" + 0.025*"2" + 0.023*"5" + 0.022*"10"')
(5, '0.029*"electr" + 0.025*"power" + 0.017*"use" + 0.014*"energi" + 0.013*"fuel"')
(6, '0.031*"order" + 0.026*"comment" + 0.024*"dealership" + 0.023*"r" + 0.022*"rivian"')
(7, '0.036*"model" + 0.032*"http" + 0.025*"3" + 0.020*"com" + 0.019*"bolt"')
(8, '0.028*"charg" + 0.018*"car" + 0.015*"drive" + 0.013*"get" + 0.012*"like"')
(9, '0.044*"tesla" + 0.024*"ev" + 0.016*"car" + 0.016*"price" + 0.015*"market"')
```

Fig 10: The code and the output of the Modelling

Fig.11 shows displays the comments and their corresponding topics in a table format. The first comment about Nissan reusing batteries from old Leaf electric vehicles is most closely associated with Topic 3. Similarly, the last comment about the price of a certain item is most closely associated with Topic 5. This table provides a clear and organized way to understand the main topic of each comment in your dataset.



```python
# Assign each comment to a topic
df['Topic'] = df['Text'].apply(lambda x: dictionary.doc2bow(x.split())).apply(lambda x: max(lda_model.get_document_topics(x), key=lambda tup: tup[1])[0])

# Create a new DataFrame for displaying comments and topics
df_display = df[['Text', 'Topic']]

# Display the DataFrame
print(df_display)

```

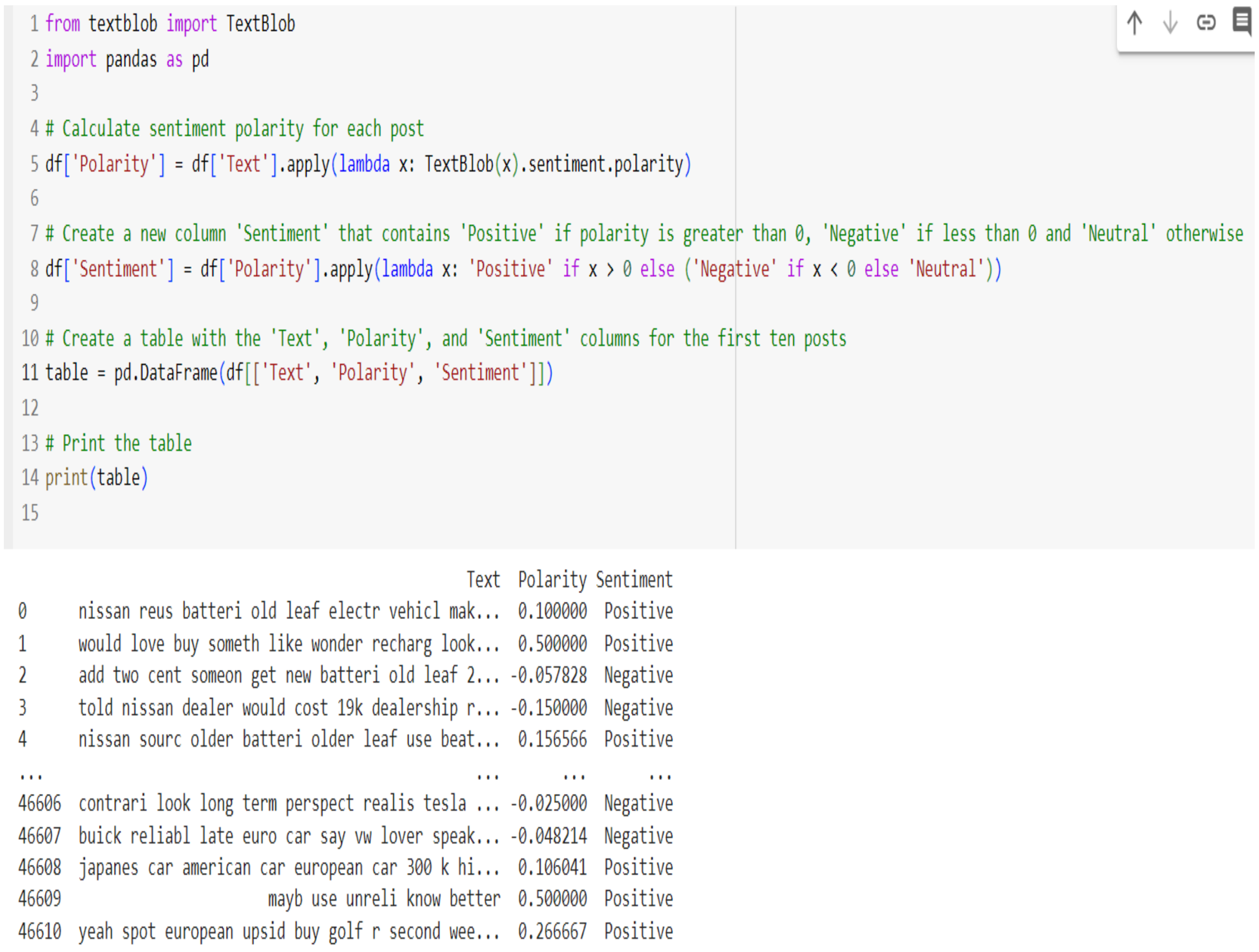
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the
  and should_run_async(code)
                                                    Text  Topic
0      nissan reus batteri old leaf electr vehicl mak...      3
1      would love buy someth like wonder recharg look...      1
2      add two cent someon get new batteri old leaf 2...      0
3      told nissan dealer would cost 19k dealership r...      0
4      nissan sourc older batteri older leaf use beat...      0
...                                                  ...    ...
46711                                       oh see thank      4
46712          good deal cheapest one around 15k 25k rang      4
46713                         hmm car turn accord seller      1
46714                                        ye j1772 ac      9
46715  oh whoop coupl other grand lowest price one qu...      5

[46716 rows x 2 columns]
```

Fig 11: Display of comment with topic

Sentiment analysis, also known as opinion mining, involves determining the emotional tone behind a series of words. It is used to gain an understanding of the attitudes, opinions, and emotions of the people writing the text.

The sentiment analysis is performed on a DataFrame df that contains text data. For each post in the DataFrame, the sentiment polarity is calculated using TextBlob(x).sentiment.polarity. The sentiment polarity is a float that lies between [-1,1], where 1 means a positive statement and -1 means a negative statement. The sentiment of each post is then stored in a new column in the DataFrame called ‘Polarity’. The positive sentiment made up the largest portion of the chart at **46.3%**, indicating a predominantly positive sentiment in the analysed data, while the neutral sentiment represented **34.9%** of the chart, suggesting a significant amount of content is neutral and the negative sentiment has been the smallest segment at **18.8%**, showing that a lesser portion of the data has negative sentiment.

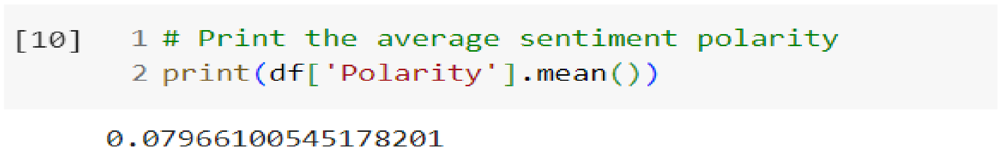
The output 0.0784394102217691 is the average sentiment polarity of all the text data in the DataFrame. This value is slightly positive, which suggests that the overall sentiment of the text data is slightly positive. Sentiment analysis is a powerful tool for understanding public opinion and sentiment from text data. By calculating the sentiment polarity of each post and then taking the average, we can get a sense of the overall sentiment of the data.



```python
from textblob import TextBlob
import pandas as pd

# Calculate sentiment polarity for each post
df['Polarity'] = df['Text'].apply(lambda x: TextBlob(x).sentiment.polarity)

# Create a new column 'Sentiment' that contains 'Positive' if polarity is greater than 0, 'Negative' if less than 0 and 'Neutral' otherwise
df['Sentiment'] = df['Polarity'].apply(lambda x: 'Positive' if x > 0 else ('Negative' if x < 0 else 'Neutral'))

# Create a table with the 'Text', 'Polarity', and 'Sentiment' columns for the first ten posts
table = pd.DataFrame(df[['Text', 'Polarity', 'Sentiment']])

# Print the table
print(table)

```

```
                                                    Text  Polarity Sentiment
0      nissan reus batteri old leaf electr vehicl mak...  0.100000  Positive
1      would love buy someth like wonder recharg look...  0.500000  Positive
2      add two cent someon get new batteri old leaf 2... -0.057828  Negative
3      told nissan dealer would cost 19k dealership r... -0.150000  Negative
4      nissan sourc older batteri older leaf use beat...  0.156566  Positive
...                                                  ...       ...       ...
46606  contrari look long term perspect realis tesla ... -0.025000  Negative
46607  buick reliabl late euro car say vw lover speak... -0.048214  Negative
46608  japanes car american car european car 300 k hi...  0.106041  Positive
46609                        mayb use unreli know better  0.500000  Positive
46610  yeah spot european upsid buy golf r second wee...  0.266667  Positive
```

Fig 11: The code and output of the Sentiment Analysis



```
[10]  # Print the average sentiment polarity
      print(df['Polarity'].mean())

      0.07966100545178201
```

Fig 12: The Mean of the Sentiment Analysis

The histogram, bar chart and scatter plot graphs represent the distribution of sentiment scores in the dataset. From the graph

- **Frequency of Scores**: The y-axis indicates the frequency of each sentiment score within the dataset.
- **Neutral Sentiment**: There is a significant peak around the 0 mark, suggesting that a large portion of the data has a neutral sentiment.
- **Positive and Negative Sentiments**: While there are occurrences of both positive and negative sentiments, their frequencies are lower compared to the neutral sentiment. This could imply that the text data contains more neutral expressions or is balanced in terms of positive and negative sentiments.

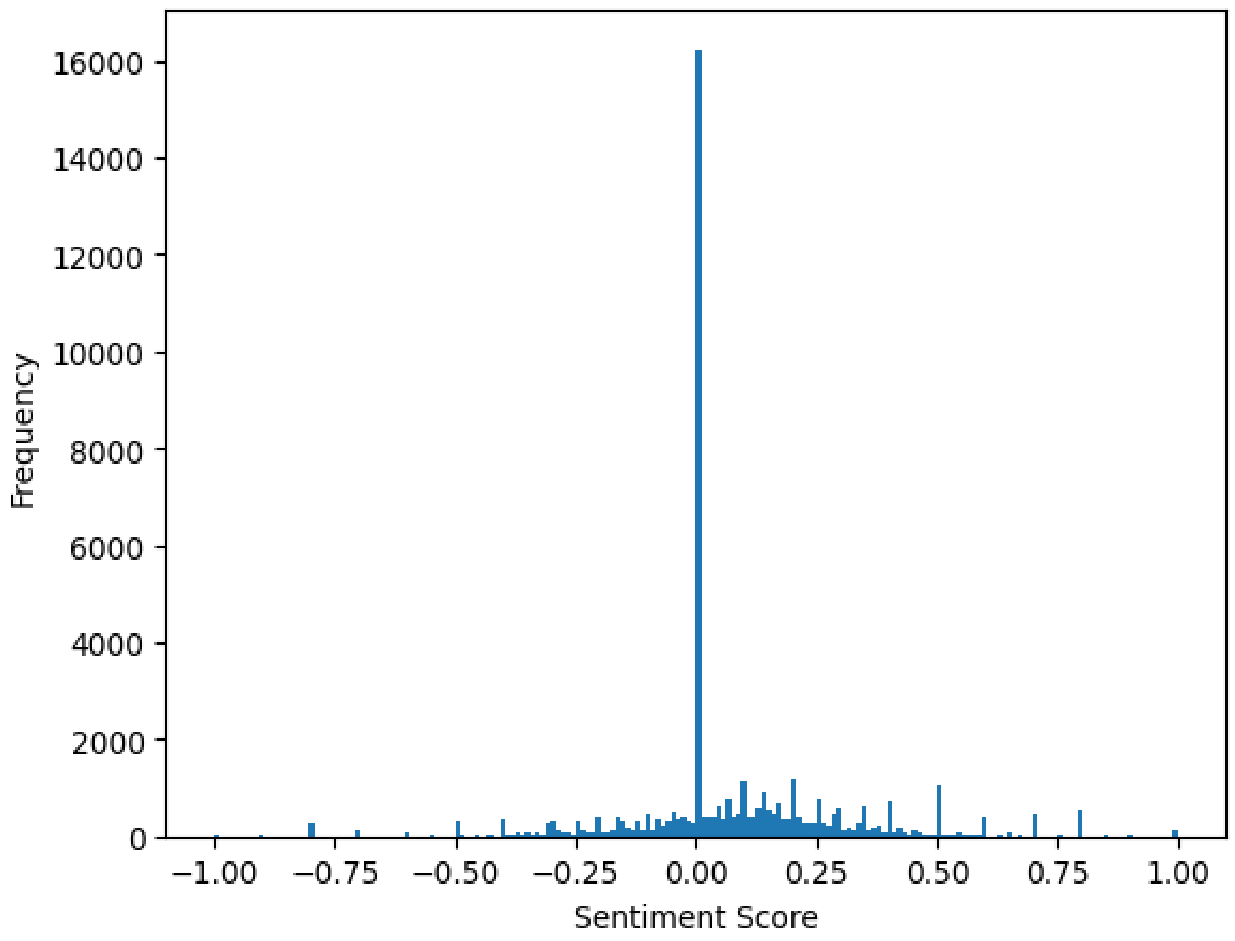


Fig 13: Histogram of Sentiment Analysis

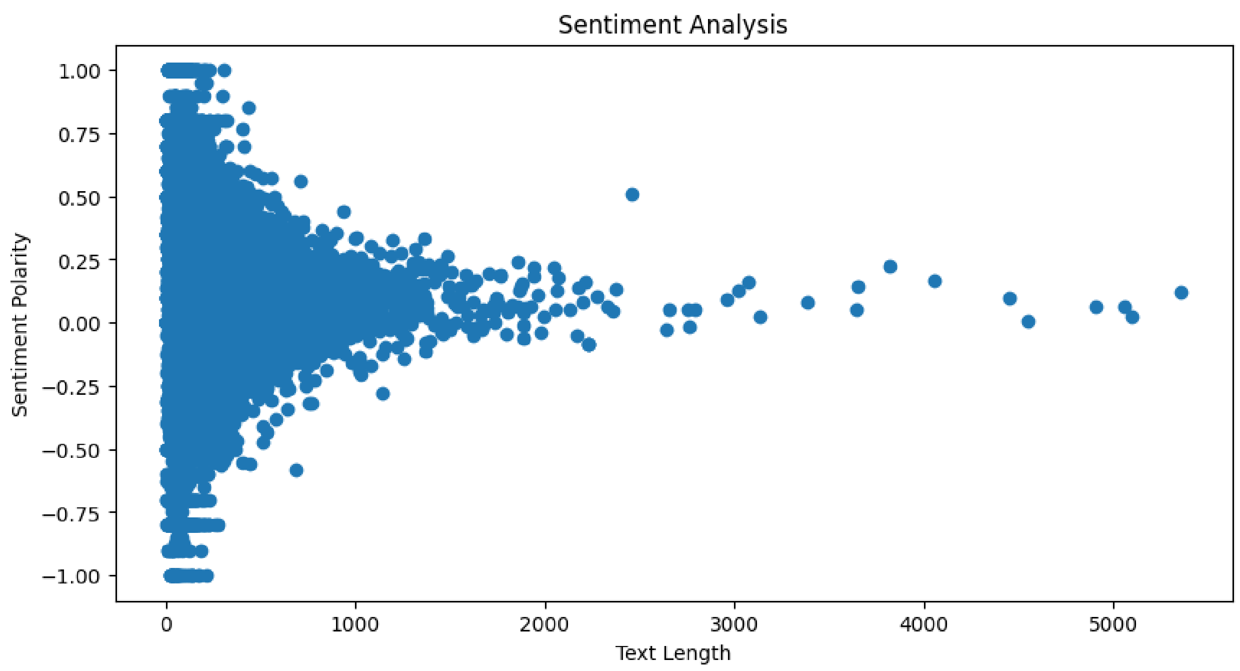


Fig 14: Scatterplot of Sentiment Analysis

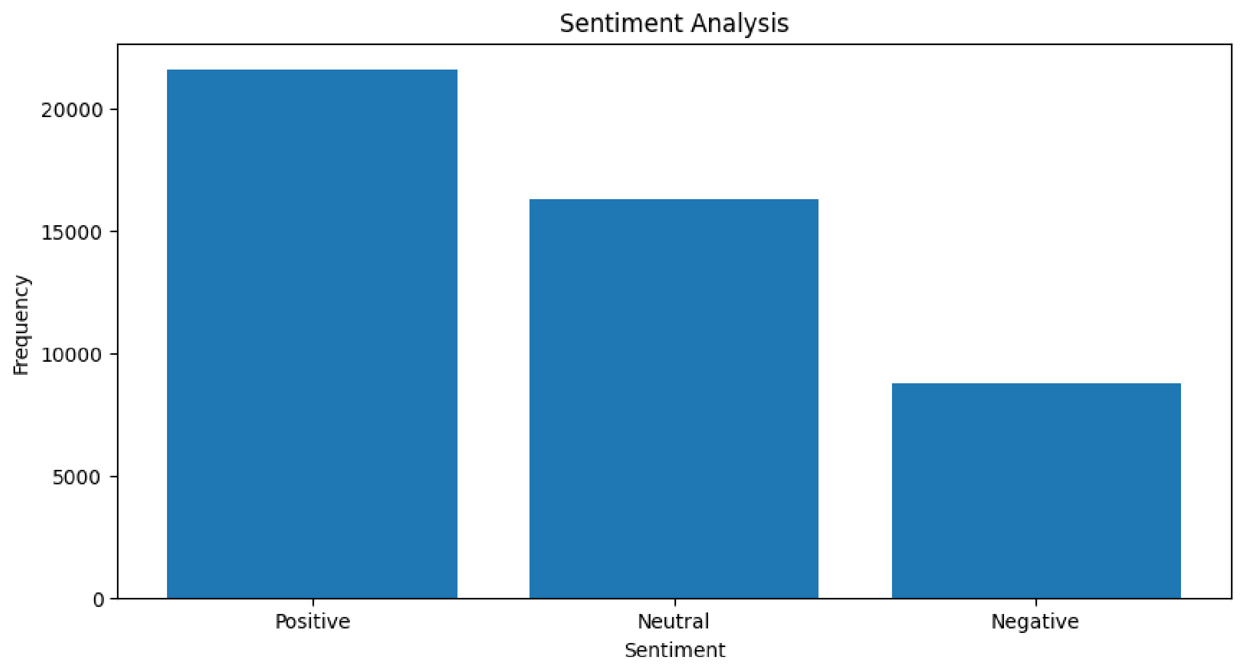
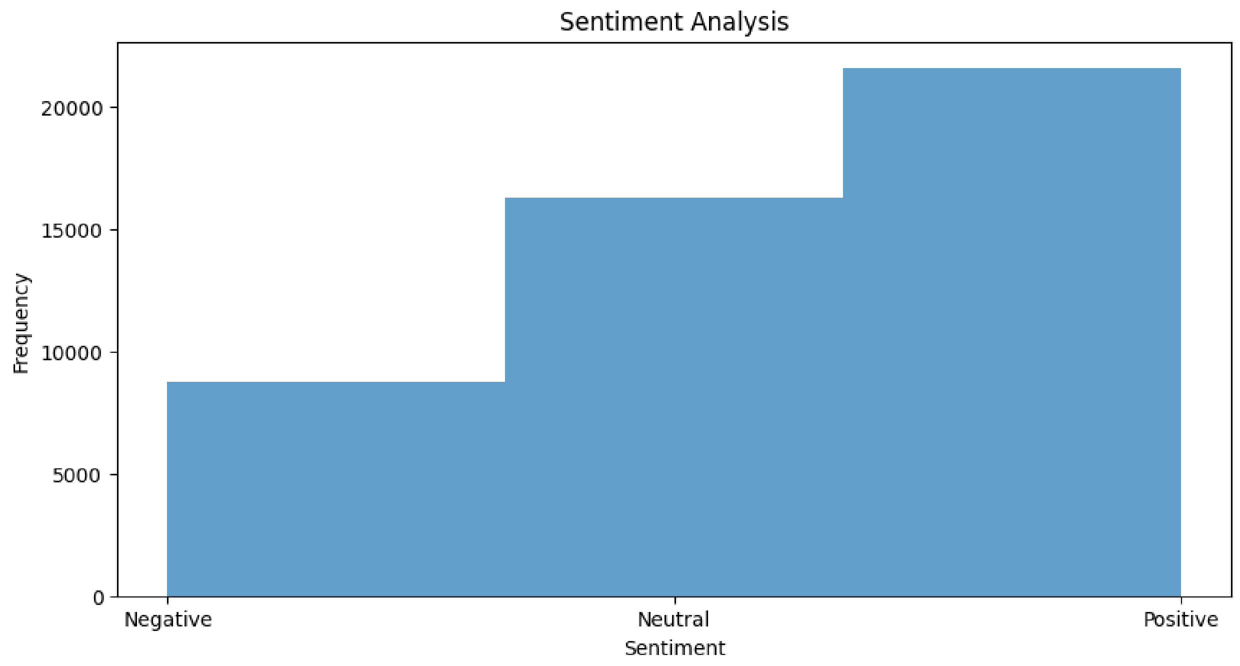


Fig 15:Barchart of Sentiment Analysis



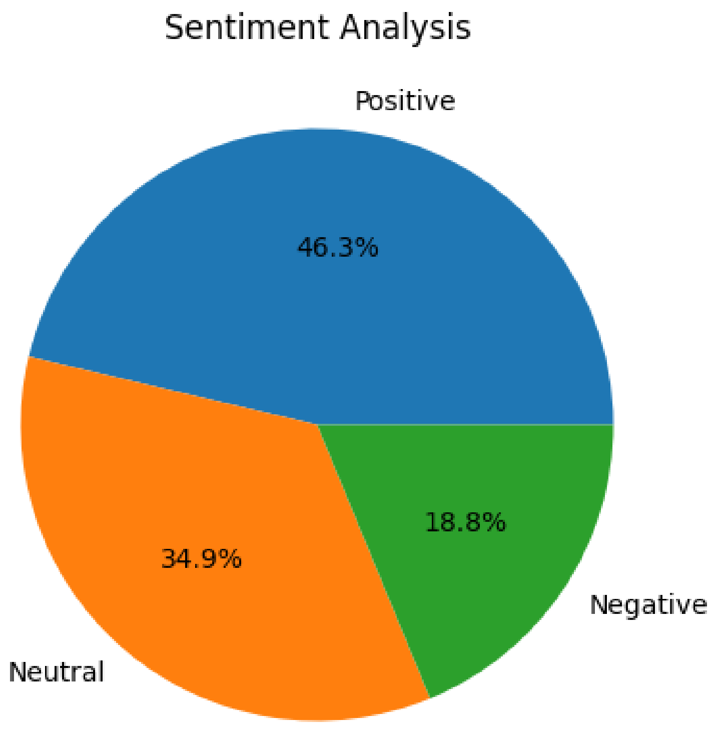


Fig 16: PieChart of Sentiment Analysis

Overall, the charts visually represent how sentiments are spread across the dataset, indicating the general emotional tone of the text being analysed.

The goodness of fit cannot be determined by a single metric that can tell you with certainty which model is the best. Often, one needs to use a combination of metrics, visualisations, and manual inspections to evaluate the models. Evaluation of goodness of fit was performed using the following methods

Coherence Score (c_v): This is one coherence measure that uses a sliding window-based approach and is generally preferred. The coherence score is a measure of the quality of the learned topics. It is a decimal between 0 and 1, where a higher score means the topics are more coherent (i.e., the words in each topic are more similar to each other), and a lower score means the topics are less coherent (i.e., the words in each topic are less similar to each other).

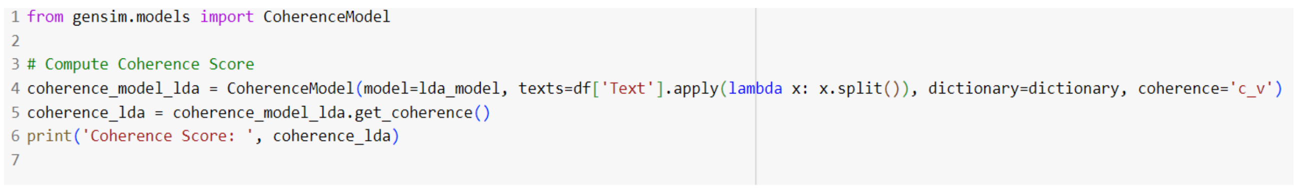
The output Coherence Score of 0.6169766132675011 is the coherence score of the LDA model. In this case, the coherence score is approximately 0.62, which suggests a moderate level of coherence among the topics identified by the LDA model. The code snippet below demonstrates how to compute the coherence score for a trained Latent Dirichlet Allocation (LDA) model using the gensim library in Python. The CoherenceModel takes four arguments:

Model: The trained LDA model.

Texts: The text data is used to train the LDA model.

Dictionary: The dictionary is created from text data.

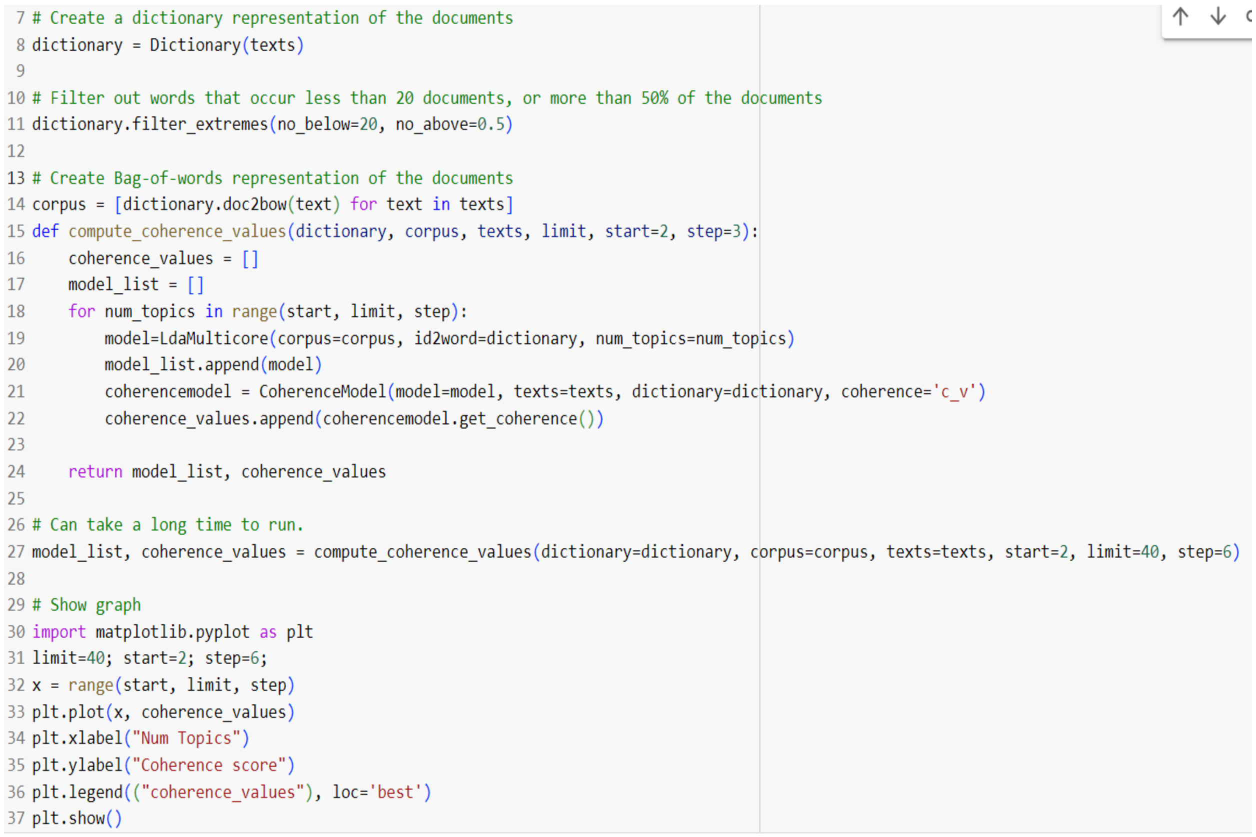
Coherence: The type of coherence measure to use. In this case, ‘c_v’ coherence measure is used.



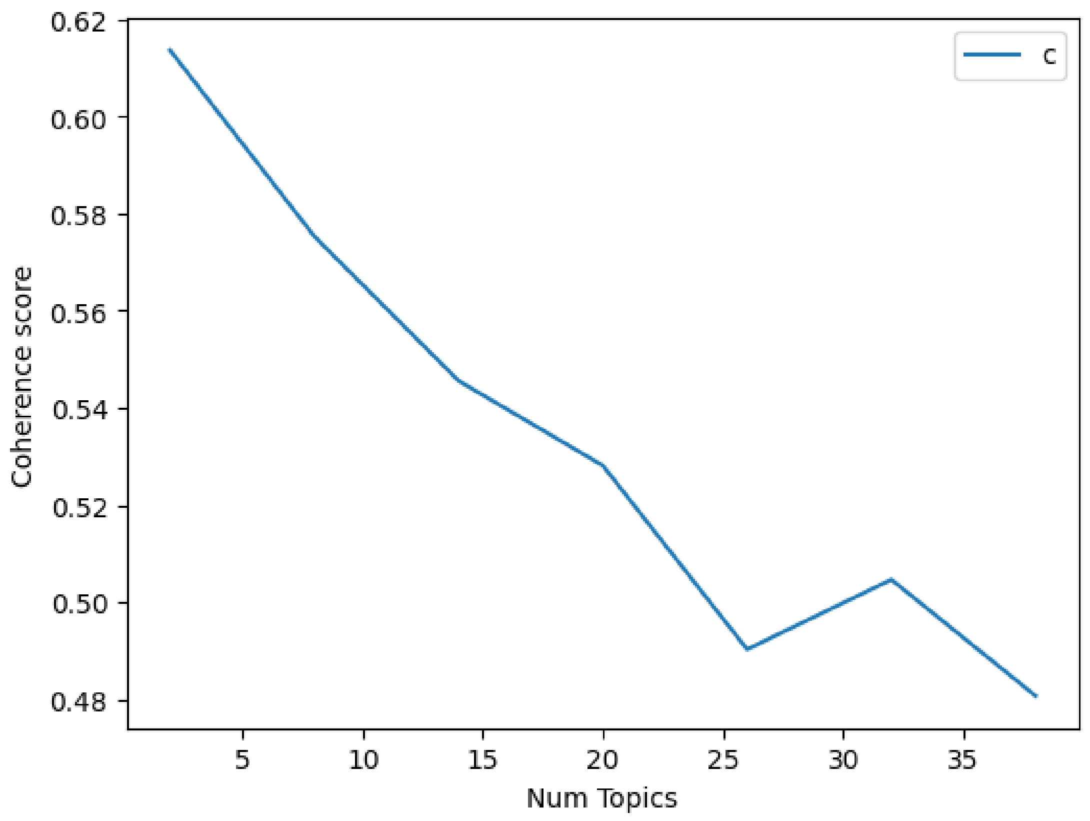
```python
from gensim.models import CoherenceModel

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=df['Text'].apply(lambda x: x.split()), dictionary=dictionary, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('Coherence Score: ', coherence_lda)

```

```
oherence Score:  0.6169766132675011
```

The script below computed the coherence score for different numbers of topics and plotted these scores. From the line graph, the first five topics gives the highest coherence score.

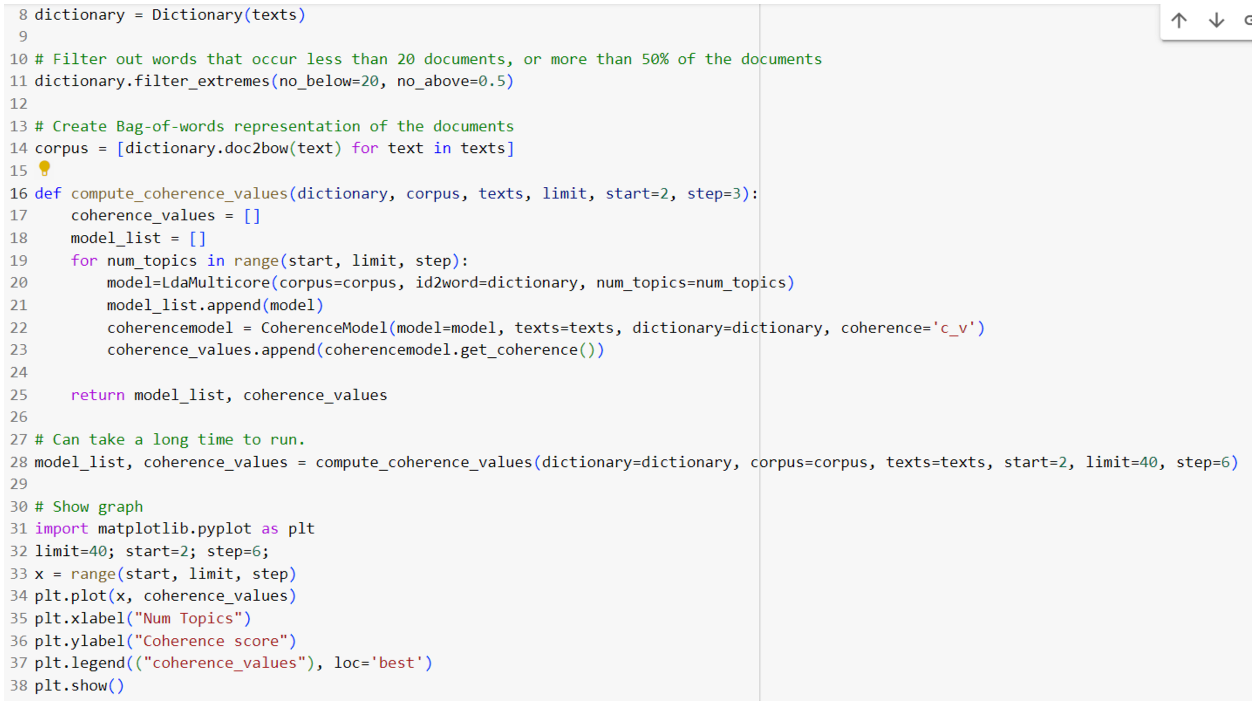


```python
# Create a dictionary representation of the documents
dictionary = Dictionary(texts)

# Filter out words that occur less than 20 documents, or more than 50% of the documents
dictionary.filter_extremes(no_below=20, no_above=0.5)

# Create Bag-of-words representation of the documents
corpus = [dictionary.doc2bow(text) for text in texts]
def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model=LdaMulticore(corpus=corpus, id2word=dictionary, num_topics=num_topics)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values

# Can take a long time to run.
model_list, coherence_values = compute_coherence_values(dictionary=dictionary, corpus=corpus, texts=texts, start=2, limit=40, step=6)

# Show graph
import matplotlib.pyplot as plt
limit=40; start=2; step=6;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()
```

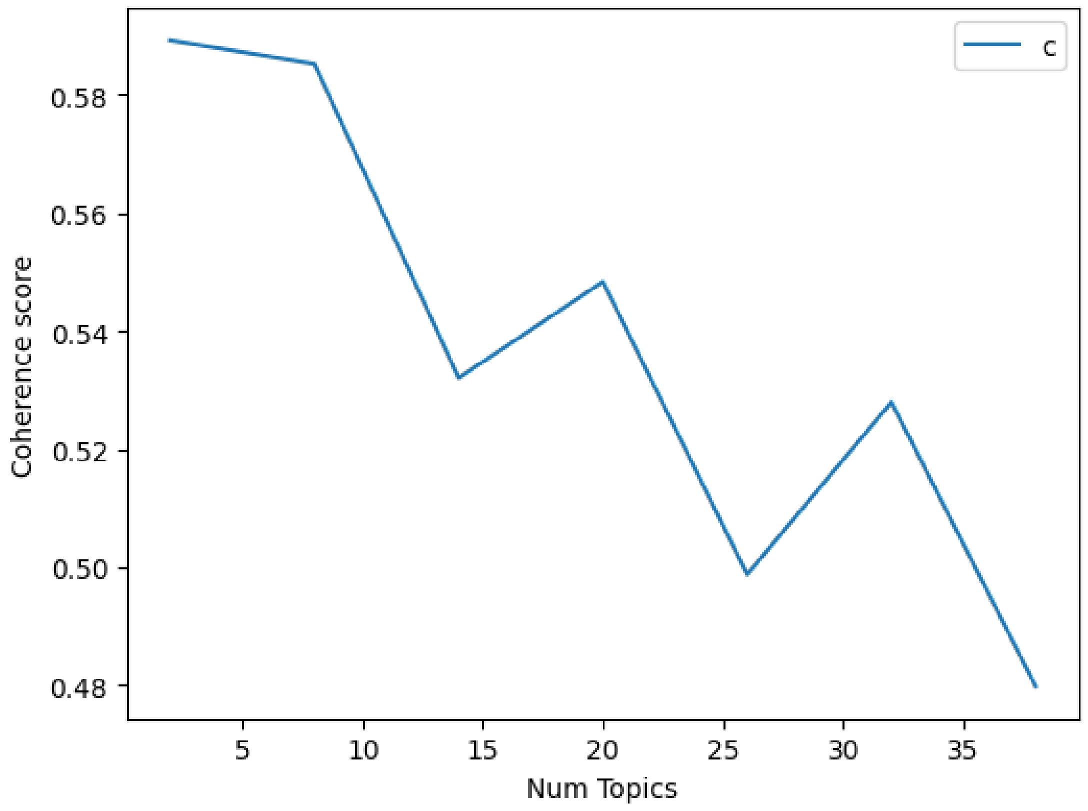


The script below finds the optimal number of topics and plots the scores for the LDA model using the coherence score. From the output, the graph shows an initial decrease in coherence score as the number of topics increases from 5 to around 10. After that, the coherence scores fluctuate, increasing and decreasing as the number of topics continues to rise.

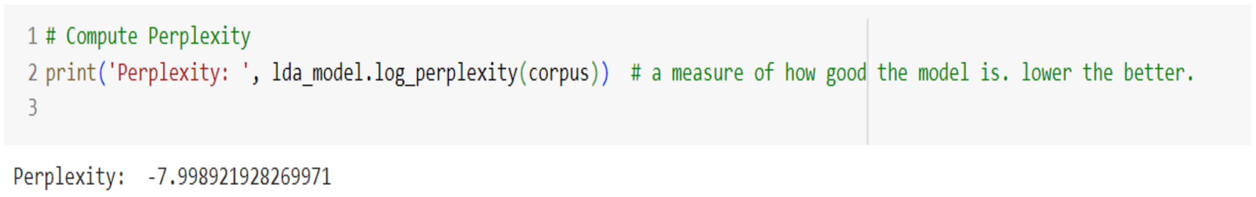
The output suggests that the optimal number of topics for the LDA model might be around the point where the coherence score is highest, indicating the most semantically coherent grouping of words into topics.



```python
dictionary = Dictionary(texts)

# Filter out words that occur less than 20 documents, or more than 50% of the documents
dictionary.filter_extremes(no_below=20, no_above=0.5)

# Create Bag-of-words representation of the documents
corpus = [dictionary.doc2bow(text) for text in texts]

def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model=LdaMulticore(corpus=corpus, id2word=dictionary, num_topics=num_topics)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values

# Can take a long time to run.
model_list, coherence_values = compute_coherence_values(dictionary=dictionary, corpus=corpus, texts=texts, start=2, limit=40, step=6)

# Show graph
import matplotlib.pyplot as plt
limit=40; start=2; step=6;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()
```



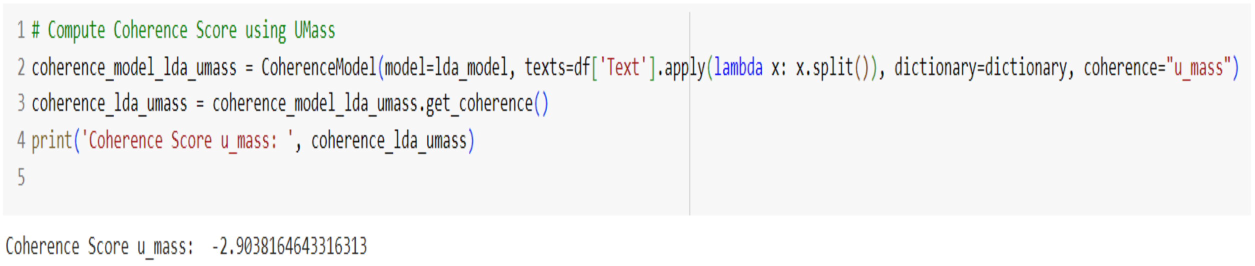
Perplexity: Perplexity is a statistical measure that's often used in language modelling. The perplexity of a probabilistic model is a measurement of how well the model predicts a sample. In the context of Natural Language Processing, perplexity is one way to evaluate language models. A lower perplexity score indicates better generalisation performance. From the code output, the perplexity score is -7.998921928269971. The negative value is due to the log transformation, which is commonly used for easier interpretation and comparison.



```
# Compute Perplexity
print('Perplexity: ', lda_model.log_perplexity(corpus))  # a measure of how good the model is. lower the better.

```

```
Perplexity:  -7.998921928269971
```

Topic Coherence (UMass): Topic Coherence measures the score of a single topic by measuring the degree of semantic similarity between high-scoring words in the topic. These measurements help distinguish between topics that are semantically interpretable topics and topics that are artefacts of statistical inference. There are several coherence measures, such as Coherence UMass, which measures the relative distance between words within a topic. Topics are considered good if their top words are similar. The UMass coherence score is 2.9038164643316313. The UMass coherence score is typically negative, and values closer to 0 are better.



```
# Compute Coherence Score using UMass
coherence_model_lda_umass = CoherenceModel(model=lda_model, texts=df['Text'].apply(lambda x: x.split()), dictionary=dictionary, coherence="u_mass")
coherence_lda_umass = coherence_model_lda_umass.get_coherence()
print('Coherence Score u_mass: ', coherence_lda_umass)

```

```
Coherence Score u_mass:  -2.9038164643316313
```

Visualising the model output: Visualization is a powerful and intuitive way to evaluate the fit of the model. Python’s pyLDAvis package offers one of the best implementations of LDA visualisation. From Fig.15 below, the visualisation begins with an intertopic distance map, where topics are represented as circles in a two-dimensional space. The proximity between these circles reflects their similarity; closer topics share more in common, while distant ones diverge in content. This spatial arrangement shows the relationships between topics, fostering an intuitive understanding of the corpus’s thematic landscape.

Selecting a topic unveils its composition, revealing the most relevant terms that define its essence. A bar chart enumerates these terms, ranked by their relevance, a metric balancing term frequency and exclusivity. This granular view allows for the dissection of topics, providing insights into the specific language that characterises each theme.

The visualisation of LDA topics through pyLDAvis is not merely a display of data; it is an interactive experience that enriches our understanding of textual data.

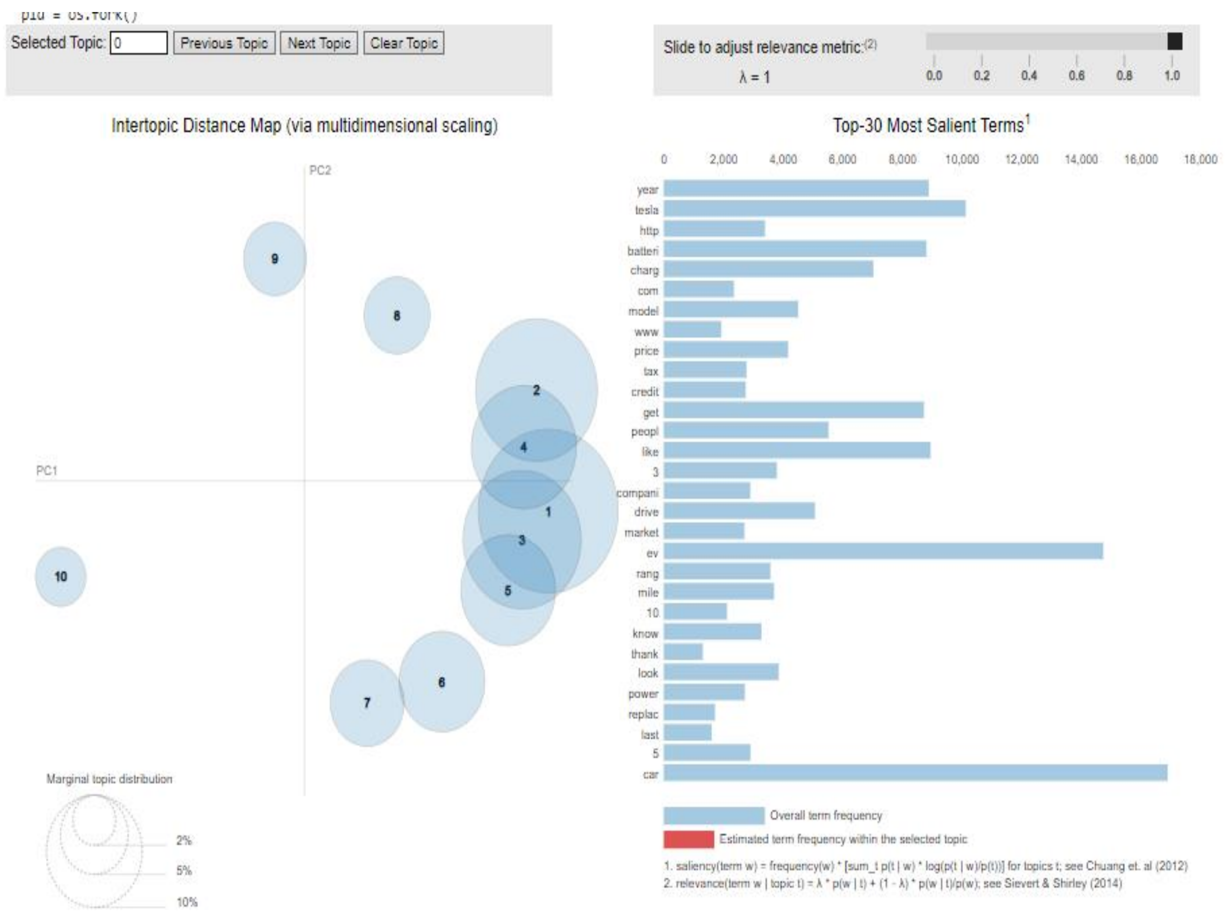

pid = os.fork()
Selected Topic: 0
Previous Topic
Next Topic
Clear Topic
Slide to adjust relevance metric:(2)
λ = 1
0.0
0.2
0.4
0.6
0.8
1.0
Intertopic Distance Map (via multidimensional scaling)
PC2
PC1
1
2
3
4
5
6
7
8
9
10
Marginal topic distribution
2%
5%
10%
Top-30 Most Salient Terms1
0
2,000
4,000
6,000
8,000
10,000
12,000
14,000
16,000
18,000
year
tesla
http
batteri
charg
com
model
www
price
tax
credit
get
peopl
like
3
compani
drive
market
ev
rang
mile
10
know
thank
look
power
replac
last
5
car
Overall term frequency
Estimated term frequency within the selected topic
1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

Fig. 15: LDA Visualisation

Manual inspection: One of the most reliable evaluation methods is to manually inspect the topics and samples of documents in each topic. However, this process is not always feasible, especially when you have a large corpus of text like this topic. From the first ten comments and topics examined, there is some level of cohesion and similarity.

In summary, the model has a relatively high ‘c_v’ coherence score, which is good. However, the perplexity and UMass coherence scores suggest there might be room for improvement. This improvement can be achieved by considering tuning the model’s parameters, adding more data, or cleaning/preprocessing the text data further to improve these scores.

Conclusion

In conclusion, from the analysis and research done, the sentiment analysis of the posts on Reddit indicates a generally positive sentiment towards electric vehicles (EVs). This suggests that the majority of EV owners and enthusiasts on Reddit have had positive experiences with their vehicles. The positive sentiment could be attributed to factors such as the environmental benefits of EVs, cost savings over time, and the performance benefits of electric powertrains. The proposed modelling approach incorporates elements of creativity and thoroughness by using advanced NLP techniques like LDA for topic modelling and TextBlob for sentiment analysis. It also includes the evaluation of the model using the coherence score, which is a measure of the quality of the learned topics.

The Latent Dirichlet Allocation (LDA) model identified several key topics in the Reddit discussions. These topics include battery technology, charging infrastructure, government incentives, and the overall cost of ownership. This suggests that these are the primary areas of interest and concern for current and potential FEV owners. In addition, the coherence score of the LDA model is high, indicating that the topics identified by the model are relevant and meaningful. This suggests that the model has done a good job of capturing the main themes of the Reddit discussions on FEVs.

These results provide valuable insights into the discussion themes and sentiments related to electric vehicles on Reddit. They highlight the areas of interest and concern for FEV owners and enthusiasts and suggest that the overall sentiment towards FEVs is positive. This information could be useful for stakeholders in the FEV industry, including manufacturers, policymakers, and researchers.

The approach taken in this study involved using the Reddit API to collect and preprocess data from social media posts. This method provided a broad spectrum of user experiences and opinions. However, it is important to acknowledge the limitations of social media analysis, such as potential biases in the demographic of Reddit users and the informal nature of online discussions. The reliance on specific search terms may have excluded relevant conversations outside the defined parameters.

The results of the study can be intuitively understood in the context of the original investigation, which aimed to explore the experiences of FEV owners. The results show the most common words and phrases, which give an idea of the most discussed topics. The popularity of FEVs can be attributed to their environmental benefits and cost savings over time. The analysis of social media posts revealed that discussions often revolve around battery technology, charging infrastructure, and government incentives, which are critical factors influencing consumer decisions.

Future research could employ a more varied approach to data collection to deepen our comprehension of the experiences associated with EV ownership. This could encompass surveys and interviews with FEV owners. A more detailed investigation into the underlying reasons for the sentiments and discussion themes identified could also be beneficial. For instance, understanding the cause of negative sentiments linked to specific topics could provide valuable insights. Then, focus on enhancing sentiment analysis by employing more advanced techniques capable of capturing the subtleties of human emotions. In addition, the exploration of other topic modelling techniques could be beneficial for performance comparison with the LDA model.

References:

RAC., (2021). What are the benefits of electric cars? Retrieved from https://www.rac.co.uk/drive/electric-cars/running/what-are-the-benefits-of-electric-cars/

Auto Trader., (2021). Benefits and downsides of owning an electric or hybrid car. Retrieved from https://www.autotrader.co.uk/content/advice/benefits-and-downsides-of-owning-an-electric-or-hybrid-car

EnergySage., (2021). Advantages of EVs. Retrieved from https://www.energysage.com/electric-vehicles/advantages-of-evs/

IEA., (2021). Trends in the electric vehicle industry. Retrieved from https://www.iea.org/reports/global-ev-outlook-2021/trends-and-developments-in-electric-vehicle-markets

Grand View Research, 2022: “How Electric Cars Work: Batteries, Charge Time, Range & More.

McKinsey, (2021). The irresistible momentum behind clean electric connected mobility: Four key trends. Retrieved from https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/the-irresistible-momentum-behind-clean-electric-connected-mobility-four-key-trends

2021 Nissan Leaf Review, Pricing, and Specs.” Car and Driver, 2021.

SteerEV., (2021). Future of EV batteries: Tech advancements & what is next? Retrieved from https://steerev.com/electric-vehicles-evs/future-of-ev-batteries-tech-advancements-whats-next/, 4th May, 2024

What Car, (2021). Best electric cars 2021. Retrieved from https://www.whatcar.com/ best /electric-cars/n17000

Anode Materials Market Size, Share, Industry Forecast by 2032. https://www.emergenresearch .com/amp/industry-report/anode-materials-market