NLP PROJECT

PLAYSTORE UBER- REVIEW ANALYSIS

SNEHA SEN – B2022112

CODE EXPLANATION

Initial Requirement

```
In [1]: import pandas as pd
        import nltk
        from textblob import TextBlob
        from collections import Counter
        import matplotlib.pyplot as plt
        from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
        from sklearn import decomposition
        import numpy as np
        import re
        from nltk.stem.porter import PorterStemmer
        from nltk.tokenize import word_tokenize
        from nltk.stem import WordNetLemmatizer, PorterStemmer
        from sklearn.model_selection import train_test_split
        import os
        from nltk.corpus import stopwords
        from nltk.stem import WordNetLemmatizer
        from sklearn.cluster import KMeans
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.decomposition import LatentDirichletAllocation
```

The above python code is the import of various libraries which are explained below:

1. pandas: Used for data manipulation and analysis.
2. nltk: The Natural Language Toolkit is a library used for text pre-processing tasks, such as tokenization, stemming, and lemmatization.
3. textblob: A library that provides easy-to-use tools for common NLP tasks like sentiment analysis and part-of-speech tagging.
4. collections: A module used for counting the occurrence of elements in a list.
5. matplotlib: A data visualization library for creating static, animated, and interactive visualizations in Python.
6. sklearn: A machine learning library for Python that includes various tools for classification, clustering, and data pre-processing.
7. numpy: A library for performing mathematical operations on arrays and matrices.
8. re: A module used for regular expressions, which are a powerful tool for pattern matching and text manipulation.
9. PorterStemmer: A module used for stemming words to their base form.
10. WordNetLemmatizer: A module used for lemmatizing words to their base form.
11. train_test_split: A function used for splitting the data into training and testing sets.
12. os: A module used for interacting with the operating system.
13. stopwords: A list of commonly occurring words (like "the", "and", "a") that are typically removed from text data.
14. KMeans: A clustering algorithm used for grouping data points into clusters.
15. MultinomialNB: A classification algorithm used for text classification.
16. accuracy_score, precision_score, recall_score, f1_score: Evaluation metrics used for measuring the performance of machine learning models.

17. TfidfVectorizer: A class used for converting text data into a matrix of TF-IDF features.
18. LatentDirichletAllocation: A class used for performing topic modeling on text data using the Latent Dirichlet Allocation algorithm.

In [2]:
```
import os
import pandas as pd

# Set the directory path where the Excel files are located
directory_path = "C:/Users/Sneha/Documents/NLP_Uber"

# Create an empty DataFrame to store the merged data
merged_data = pd.DataFrame()

# Loop through all the files in the directory
for file_name in os.listdir(directory_path):
    # Check if the file is an Excel file
    if file_name.endswith(".xlsx"):
        # Read the Excel file into a DataFrame
        file_path = os.path.join(directory_path, file_name)
        data = pd.read_excel(file_path)
        # Append the data to the merged_data DataFrame
        merged_data = merged_data.append(data)

# Write the merged data to a new Excel file called "merged_review.xlsx"
merged_data.to_excel(os.path.join(directory_path, "merged_review.xlsx"), index=False)
```

Here is a brief explanation of each step:

1. import os and pandas libraries: os is a module used for interacting with the operating system, and pandas is a Python library for data manipulation and analysis.
2. Set the directory path: A directory path where Excel files are located is assigned to the variable "directory_path".
3. Create an empty DataFrame: An empty DataFrame called "merged_data" is created to store the merged data.
4. Loop through all the files in the directory: A for loop is used to loop through all the files in the directory.
5. Check if the file is an Excel file: An if statement is used to check if the file has a ".xlsx" extension.
6. Read the Excel file into a DataFrame: If the file is an Excel file, the pandas "read_excel" function is used to read the data from the file into a DataFrame called "data".
7. Append the data to the merged_data DataFrame: The "append" method is used to append the data from the current Excel file to the "merged_data" DataFrame.
8. Write the merged data to a new Excel file: Finally, the "to_excel" method is used to write the merged data to a new Excel file called "merged_review.xlsx" in the same directory as the original files. The "index=False" argument is used to exclude the index column from the output file.

In [3]:
```
df = pd.read_excel('merged_review.xlsx')
df.head(10)
```

Out[3]:

	Date	Stars	Sentiment	Review
0	29-10-2019	1	Negative	I had an accident with an Uber driver in Mexic...
1	28-10-2019	1	Negative	I have had my account completely hacked to whe...
2	27-10-2019	1	Negative	I requested an 8 mile ride in Boston on a Satu...
3	27-10-2019	1	Negative	I've been driving off and on with the company ...
4	25-10-2019	1	Negative	Uber is overcharging for Toll fees. When In Fl...
5	24-10-2019	1	Negative	I had an airport flight today. Uber would not ...
6	24-10-2019	1	Negative	I worked for Uber and Lyft for 2.5 years and a...
7	23-10-2019	1	Positive	In July of this year I had sushi delivered to ...
8	23-10-2019	1	Negative	My driver, Rohan was nice, but when I tried to...
9	21-10-2019	1	Negative	I had seven fraudulent Uber transactions over ...

This lines executed to use pandas library to extract the dataset and have a view of the sentiments.

Phase 1

Q1. Overall sentiment if the dataset

In [4]:
```
# Creating a new column for sentiment scores
df['sentiment_score'] = df['Review'].apply(lambda x: TextBlob(x).sentiment.polarity)

# Defining a function to categorize sentiment scores into positive, negative or neutral
def get_sentiment_category(score):
    if score > 0:
        return 'Positive'
    else:
        return 'Negative'


# Creating a new column for sentiment categories
df['sentiment_category'] = df['sentiment_score'].apply(get_sentiment_category)

# Printing the overall sentiment of the product
print('Overall sentiment:', df['sentiment_category'].value_counts().idxmax())
```

```
Overall sentiment: Positive
```

Performs sentiment analysis on the text data in a DataFrame column called "Review" using the TextBlob library. Here's a brief explanation of each step:

1. Create a new column for sentiment scores: A new column called "sentiment_score" is created in the DataFrame called "df". The "apply" method is used with a lambda function to apply TextBlob's sentiment analysis to each review in the "Review" column and extract the polarity score.

2. Define a function to categorize sentiment scores: A function called "get_sentiment_category" is defined to categorize sentiment scores as positive or negative. If the score is greater than 0, the function returns 'Positive', otherwise it returns 'Negative'.
3. Create a new column for sentiment categories: A new column called "sentiment_category" is created in the DataFrame by applying the "get_sentiment_category" function to the "sentiment_score" column.
4. Print the overall sentiment of the product: The code prints the overall sentiment of the product by counting the number of positive and negative sentiment categories in the "sentiment_category" column and returning the most frequent category using the "idxmax" method.

Overall, this code performs sentiment analysis on text data and categorizes it as positive or negative, allowing for an easy way to understand the sentiment of the product.

```
In [5]: df
```

Out[5]:

	Date	Stars	Sentiment	Review	sentiment_score	sentiment_category
0	29-10-2019	1	Negative	I had an accident with an Uber driver in Mexic...	-0.185556	Negative
1	28-10-2019	1	Negative	I have had my account completely hacked to whe...	0.033333	Positive
2	27-10-2019	1	Negative	I requested an 8 mile ride in Boston on a Satu...	0.053333	Positive
3	27-10-2019	1	Negative	I've been driving off and on with the company ...	0.363939	Positive
4	25-10-2019	1	Negative	Uber is overcharging for Toll fees. When In Fl...	-0.016667	Negative
...	...	...	...	...	...	...
2102	2016-02-23 00:00:00	1	Positive	Wow! Where do I begin?! I applied to be a driv...	0.059375	Positive
2103	2016-02-22 00:00:00	2	Positive	The first time ever in my life that I used ...	0.209773	Positive
2104	2016-02-22 00:00:00	4	Negative	Short trip take a cab. Longer run out of town...	0.000000	Negative
2105	2016-02-21 00:00:00	1	Positive	If you are contemplating driving for Uber in N...	0.176061	Positive
2106	2016-02-21 00:00:00	5	Positive	We went on vacation to Washington dc...we took...	0.712500	Positive

2107 rows × 6 columns

To look at the dataframe if giving an elaborated sentiment score and sentiment category

Q2. Step 1 - Collecting all positive sentiments

In [6]:
```
# create a new dataframe with only positive sentiment sentences-
positive_df = df.loc[df['sentiment_score'] > 0, ['Date', 'Stars', 'Sentiment','Review','sentiment_category']]

# print the positive dataframe
print(positive_df.head(10))
```

```
          Date  Stars Sentiment  \
1   28-10-2019      1  Negative
2   27-10-2019      1  Negative
3   27-10-2019      1  Negative
5   24-10-2019      1  Negative
10  21-10-2019      1  Negative
11  20-10-2019      1  Negative
13  18-10-2019      1  Negative
14  17-10-2019      1  Positive
16  14-10-2019      1  Negative
17  14-10-2019      1  Negative

                                               Review sentiment_category
1   I have had my account completely hacked to whe...           Positive
2   I requested an 8 mile ride in Boston on a Satu...           Positive
3   I've been driving off and on with the company ...           Positive
5   I had an airport flight today. Uber would not ...           Positive
10  Our driver never showed up and Uber cancelled ...           Positive
11  When the service worked, it was good, and tech...           Positive
13  They didn't have cars available at Bush Intern...           Positive
14  I have been punished for circumstances beyond ...           Positive
16  Upfront pricing not true pricing. Uber pads th...           Positive
17  I don't believe there is any way to contact Ub...           Positive
```

This code creates a new DataFrame called "positive_df" by filtering the original DataFrame called "df" to only include reviews with a positive sentiment score. Here's a brief explanation of each step:

1. Filter the original DataFrame to only include positive sentiment reviews: A new DataFrame called "positive_df" is created by filtering the "df" DataFrame using the "loc" method. The condition "df['sentiment_score'] > 0" filters the rows of the DataFrame to only include reviews with a positive sentiment score. The columns 'Date', 'Stars', 'Sentiment', 'Review', and 'sentiment_category' are selected for the new DataFrame using the list of column names.
2. Print the positive DataFrame: The code prints the first 10 rows of the "positive_df" DataFrame using the "head" method. This allows us to examine the content of the DataFrame to ensure it was filtered correctly.

In [7]:
```
# Required Preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = stopwords.words('english')
VERB_CODES = {'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'}
stop_words = set(stopwords.words('english'))
new_stop_words = ['the','and','uber','get','would','raleigh','sacca','lyft','empresa',
                  'dinero','que','got','use','used','one','youll','youve','boyfriend',
                  'mother','epect','mom','picture','sister','usd','said','first']
stop_words.update(new_stop_words)
```

This code sets up the required pre-processing steps for the text data. Here's a brief explanation of each step:

1. WordNetLemmatizer: The WordNetLemmatizer object is created from the NLTK library. This object is used to perform lemmatization on the text data, which involves reducing words to their base form (e.g., "running" -> "run").

2. Stopwords: Stopwords are common words that are often removed from text data because they don't carry much meaning (e.g., "the", "and", "a"). The stopwords for the English language are loaded from the NLTK library using the "stopwords.words('english')" method.
3. VERB_CODES: The VERB_CODES is a set that contains the codes for different types of verbs (e.g., 'VB' for base form, 'VBD' for past tense). This will be used later in the code to identify verbs for lemmatization.
4. Additional stopwords: A list of additional stopwords is created and added to the stop_words set. This list includes words that are specific to the context of the text data and may not be relevant for analysis.

```
In [8]: # Preprocessing function
def preprocess(text):
    text = text.lower()
    text=text.strip()  #get rid of leading/trailing whitespace
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text) # remove all the symbols which are not belong to this set
    text = re.sub(r'X.*X', '', text)  # # remove all the masked words that start with 'X' and end with 'X'.
    text = re.sub(r'x.*x', '', text)  # remove all the masked words that start with 'x' and end with 'x'.
    temp_sent =[]
    words = nltk.word_tokenize(text)
    for word in words:         # keep all the words that do not belong to stop word list and contain only alphabetic character
        lemmatized = lemmatizer.lemmatize(word)
        if lemmatized not in stop_words and lemmatized.isalpha() and len(lemmatized)>2:
            temp_sent.append(lemmatized)

    finalsent = ' '.join(temp_sent)
    return finalsent

positive_df["Review_new"]= positive_df["Review"].apply(preprocess) # Creating a new preprocessed column
```

The code defines a preprocessing function named preprocess which performs several text cleaning operations like lowercasing, removing leading/trailing whitespaces, removing symbols, removing masked words that start and end with 'X' or 'x', tokenizing, lemmatizing, removing stop words and retaining only alphabetic words with a length greater than 2.

The function is applied to the 'Review' column of the 'positive_df' dataframe using the apply method, and the resulting preprocessed text is stored in a new column named 'Review_new'.

Step 2 – Clustering

```
In [9]: # Clean the reviews by removing stop words and stemming
vectorizer = TfidfVectorizer(stop_words='english')
reviews = positive_df['Review_new']
tfidf_matrix = vectorizer.fit_transform(reviews)
```

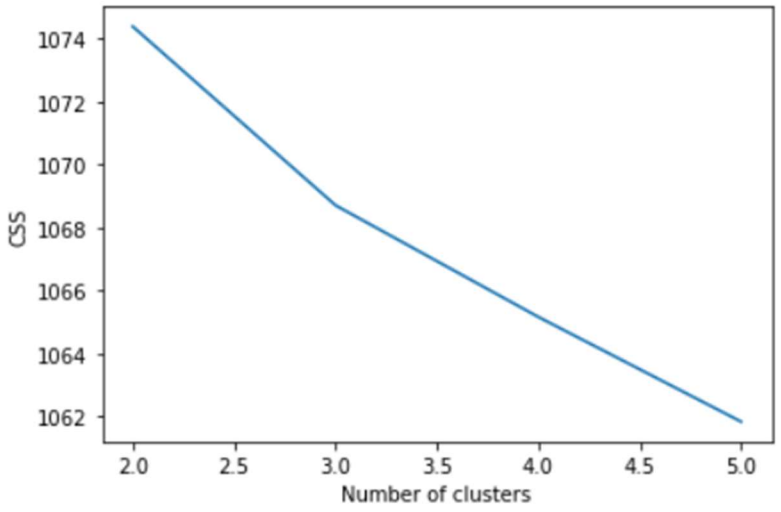
The above code explains the cleaning of stop words from reviews those were recorded and stemming them.

```
In [10]: css = []
for i in range(2, 6):
    kmeans = KMeans(n_clusters=i, max_iter=100, n_init=10, random_state=42).fit(tfidf_matrix)
    css.append(kmeans.inertia_)
```

This code is performing KMeans clustering on a term frequency-inverse document frequency (tf-idf) matrix, which is represented by tfidf_matrix. The KMeans() function is being used to create KMeans models with varying numbers of clusters ranging from 2 to 5. For each model, the inertia_ attribute is being calculated and stored in a list called css. The inertia_ attribute is a measure of how far the

points within each cluster are from the centroid of the cluster. The purpose of this code is to evaluate the optimal number of clusters to use for the KMeans clustering algorithm by analysing the within-cluster sum of squares (CSS) for each number of clusters. The elbow method can be applied to the CSS values to determine the optimal number of clusters to use for the model.

```
In [11]: import matplotlib.pyplot as plt
         plt.plot(range(2, 6), css)
         plt.xlabel('Number of clusters')
         plt.ylabel('CSS')
         plt.show()
```



The above code is the explanation of the how to represent the K-Means clustering in a line chart format as a visual representation.

```
In [12]: # Perform cluster analysis using K-means clustering algorithm on 3 or 4
num_clusters = 3
km = KMeans(n_clusters=num_clusters)
km.fit(tfidf_matrix)

# Add the cluster labels to the DataFrame
clusters = km.labels_.tolist()
positive_df['Cluster'] = clusters

# Assign names to the clusters based on the most frequent words in each cluster
cluster_names = {}
for i in range(num_clusters):
    cluster_reviews = positive_df[positive_df['Cluster'] == i]['Review_new']
    blob = TextBlob(' '.join(cluster_reviews))
    words = [word.lower() for word in blob.words if len(word) > 2]
    word_counts = {}
    for word in words:
        if word not in word_counts:
            word_counts[word] = 0
        word_counts[word] += 1
    sorted_words = sorted(word_counts.items(), key=lambda x: x[1], reverse=True)
    cluster_names[i] = ' '.join([word[0] for word in sorted_words[:10]])

# Print the DataFrame with the cluster labels and names
for i in range(num_clusters):
    print('Cluster', i, '-', cluster_names[i])
    print()
```

```
Cluster 0 - driver car time service dont company ride driving good like

Cluster 1 - driver ride time trip charged minute charge never app service

Cluster 2 - account card email ride driver credit phone customer time service
```

This code performs cluster analysis using the K-means clustering algorithm on a DataFrame called positive_df that contains only the positive sentiment sentences. The number of clusters is set to 3 using the num_clusters variable.

The K-means algorithm is fitted on the tfidf_matrix matrix, and the resulting cluster labels are added to the positive_df DataFrame. The names of the clusters are then assigned based on the most frequent words in each cluster.

The cluster_names dictionary is used to store the names of the clusters. It is created by iterating over each cluster, extracting the reviews in that cluster, and applying text preprocessing techniques such as removing stop words and lemmatization using the TextBlob and nltk libraries.

The most frequently occurring words in each cluster are then selected using a word count dictionary, and the top 10 words are concatenated to form the name of the cluster. Finally, the cluster_names dictionary is printed along with the cluster number to display the name of each cluster.

Step 3 – Topic Modelling

In [13]:
```
# Create a bag-of-words matrix
vectorizer = CountVectorizer()
bow_matrix = vectorizer.fit_transform(positive_df['Review_new'])

# Perform Latent Dirichlet Allocation topic modeling
num_topics = 10
lda = LatentDirichletAllocation(n_components=num_topics, max_iter=10, learning_method='online', random_state=42)
lda.fit(bow_matrix)

# Print the top 10 words for each topic
for i, topic in enumerate(lda.components_):
    print(f"Topic {i}:")
    print([vectorizer.get_feature_names()[j] for j in topic.argsort()[:-11:-1]])
```

```
Topic 0:
['wifi', 'booking', 'robert', 'bill', 'sudden', 'fee', 'unscrupulous', 'faggot', 'ueber', 'ola']
Topic 1:
['daughter', 'injury', 'eat', 'obvious', 'hospital', 'prom', 'toll', 'worry', 'height', 'joined']
Topic 2:
['gift', 'card', 'paypal', 'credit', 'last', 'night', 'purchased', 'brought', 'accept', 'sept']
Topic 3:
['kansa', 'todo', 'photo', 'addition', 'quiere', 'esta', 'beep', 'basis', 'epenses', 'senior']
Topic 4:
['data', 'debit', 'card', 'prepaid', 'danger', 'accept', 'allow', 'personal', 'gift', 'ruin']
Topic 5:
['orleans', 'new', 'street', 'ave', 'port', 'tampa', 'wife', 'god', 'airport', 'mobile']
Topic 6:
['insurance', 'account', 'rider', 'team', 'company', 'money', 'customer', 'month', 'case', 'big']
Topic 7:
['order', 'address', 'app', 'location', 'try', 'cant', 'business', 'york', 'driver', 'email']
Topic 8:
['driver', 'company', 'car', 'vehicle', 'clean', 'always', 'like', 'good', 'people', 'safe']
Topic 9:
['driver', 'ride', 'time', 'service', 'trip', 'car', 'never', 'charged', 'customer', 'app']
```

The above code is performing topic modeling using Latent Dirichlet Allocation (LDA) on the bag-of-words matrix of positive reviews.

First, a bag-of-words matrix is created using CountVectorizer from scikit-learn library, which is a matrix representation of the frequency of words in each document (in this case, positive reviews).

Next, LDA is applied to the bag-of-words matrix to extract topics from the positive reviews. The number of topics is set to 10, and the algorithm is run for a maximum of 10 iterations using the online learning method.

Finally, the top 10 words for each topic are printed using the get_feature_names() function of the vectorizer and the argsort() function to sort the topic in descending order.

Conclusion drafted - Good Things with our Product-

1. good Driving
2. Phone service is good
3. People feel safe
4. Vehicle clean
5. Unsrupulus service

Q3. Step 1 – Collecting all the negative comments

```
In [14]: # create a new dataframe with only negative sentiment sentences-
         negative_df = df.loc[df['sentiment_score'] < 0, ['Date', 'Stars', 'Sentiment','Review','sentiment_category']]

         # print the negative dataframe
         print(negative_df.head(10))
```

```
                  Date  Stars Sentiment  \
0           29-10-2019      1  Negative
4           25-10-2019      1  Negative
6           24-10-2019      1  Negative
7           23-10-2019      1  Positive
8           23-10-2019      1  Negative
9           21-10-2019      1  Negative
12          18-10-2019      1  Positive
15          17-10-2019      1  Negative
18          13-10-2019      1  Positive
19  2019-12-10 00:00:00      1  Negative

                                       Review sentiment_category
0   I had an accident with an Uber driver in Mexic...          Negative
4   Uber is overcharging for Toll fees. When In Fl...          Negative
6   I worked for Uber and Lyft for 2.5 years and a...          Negative
7   In July of this year I had sushi delivered to ...          Negative
8   My driver, Rohan was nice, but when I tried to...          Negative
9   I had seven fraudulent Uber transactions over ...          Negative
12  Being a responsible driver, I made the decisio...          Negative
15  Don't use it, that's my recommendation!. I ord...          Negative
18  Drive with Uber you need to get better. I curr...          Negative
19  Uber quoted me a fare of Ã,Â£14. Their driver ...          Negative
```

This code creates a new DataFrame called "negativee_df" by filtering the original DataFrame called "df" to only include reviews with a negative sentiment score. Here's a brief explanation of each step:

1. Filter the original DataFrame to only include negative sentiment reviews: A new DataFrame called "negative_df" is created by filtering the "df" DataFrame using the "loc" method. The condition "df['sentiment_score'] > 0" filters the rows of the DataFrame to only include reviews with a negative sentiment score. The columns 'Date', 'Stars', 'Sentiment', 'Review', and 'sentiment_category' are selected for the new DataFrame using the list of column names.
2. Print the negative DataFrame: The code prints the first 10 rows of the "negative_df" DataFrame using the "head" method. This allows us to examine the content of the DataFrame to ensure it was filtered correctly.

In [15]:
```
# Required Preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = stopwords.words('english')
VERB_CODES = {'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'}
stop_words = set(stopwords.words('english'))
new_stop_words = ['the','and','uber','get','would','raleigh','sacca','lyft','empresa','dinero',
                  'que','got','use','used','one','youll','youve','boyfriend','mother','epect',
                  'mom','picture','sister','hello']
stop_words.update(new_stop_words)
```

This code sets up the required pre-processing steps for the text data. Here's a brief explanation of each step:

1. WordNetLemmatizer: The WordNetLemmatizer object is created from the NLTK library. This object is used to perform lemmatization on the text data, which involves reducing words to their base form (e.g., "running" -> "run").
2. Stopwords: Stopwords are common words that are often removed from text data because they don't carry much meaning (e.g., "the", "and", "a"). The stopwords for the English language are loaded from the NLTK library using the "stopwords.words('english')" method.
3. VERB_CODES: The VERB_CODES is a set that contains the codes for different types of verbs (e.g., 'VB' for base form, 'VBD' for past tense). This will be used later in the code to identify verbs for lemmatization.
4. Additional stopwords: A list of additional stopwords is created and added to the stop_words set. This list includes words that are specific to the context of the text data and may not be relevant for analysis.

In [16]:
```
# Preprocessing function
def preprocess(text):
    text = text.lower()
    text=text.strip()  #get rid of leading/trailing whitespace
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text) # remove all the symbols which are not belong to this set
    text = re.sub(r'X.*X', '', text)  # # remove all the masked words that start with 'X' and end with 'X'.
    text = re.sub(r'x.*x', '', text)  # remove all the masked words that start with 'x' and end with 'x'.
    temp_sent =[]
    words = nltk.word_tokenize(text)
    for word in words:         # keep all the words that do not belong to stop word list and contain only alphabetic character
        lemmatized = lemmatizer.lemmatize(word)
        if lemmatized not in stop_words and lemmatized.isalpha() and len(lemmatized)>2:
            temp_sent.append(lemmatized)

    finalsent = ' '.join(temp_sent)
    return finalsent

negative_df["Review_new"]= negative_df["Review"].apply(preprocess) # Creating a new preprocessed column
```

The code defines a pre-processing function named pre-process which performs several text cleaning operations like lowercasing, removing leading/trailing whitespaces, removing symbols, removing masked words that start and end with 'X' or 'x', tokenizing, lemmatizing, removing stop words and retaining only alphabetic words with a length greater than 2.

The function is applied to the 'Review' column of the 'negative_df' dataframe using the apply method, and the resulting pre-processed text is stored in a new column named 'Review_new'.

Step 2 – Clustering

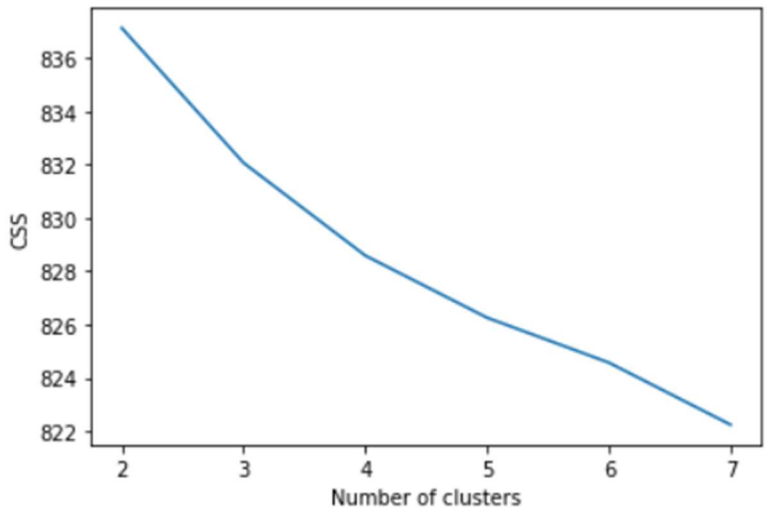
```
In [18]: # Clean the reviews by removing stop words and stemming
         vectorizer = TfidfVectorizer(stop_words='english')
         reviews = negative_df['Review_new']
         tfidf_matrix = vectorizer.fit_transform(reviews)
```

The above code explains the cleaning of stop words from reviews those were recorded and stemming them.

```
In [10]: css = []
         for i in range(2, 6):
             kmeans = KMeans(n_clusters=i, max_iter=100, n_init=10, random_state=42).fit(tfidf_matrix)
             css.append(kmeans.inertia_)
```

This code is performing KMeans clustering on a term frequency-inverse document frequency (tf-idf) matrix, which is represented by tfidf_matrix. The KMeans() function is being used to create KMeans models with varying numbers of clusters ranging from 2 to 5. For each model, the inertia_ attribute is being calculated and stored in a list called css. The inertia_ attribute is a measure of how far the points within each cluster are from the centroid of the cluster. The purpose of this code is to evaluate the optimal number of clusters to use for the KMeans clustering algorithm by analysing the within-cluster sum of squares (CSS) for each number of clusters. The elbow method can be applied to the CSS values to determine the optimal number of clusters to use for the model.

```
In [20]: import matplotlib.pyplot as plt
         plt.plot(range(2, 8), css)
         plt.xlabel('Number of clusters')
         plt.ylabel('CSS')
         plt.show()
```



The above code is the explanation of the how to represent the K-Means clustering in a line chart format as a visual representation.

In [21]:
```
# Perform cluster analysis using K-means clustering algorithm on 3 or 4
num_clusters = 4
km = KMeans(n_clusters=num_clusters)
km.fit(tfidf_matrix)

# Add the cluster labels to the DataFrame
clusters = km.labels_.tolist()
negative_df['Cluster'] = clusters

# Assign names to the clusters based on the most frequent words in each cluster
cluster_names = {}
for i in range(num_clusters):
    cluster_reviews = negative_df[negative_df['Cluster'] == i]['Review_new']
    blob = TextBlob(' '.join(cluster_reviews))
    words = [word.lower() for word in blob.words if len(word) > 2]
    word_counts = {}
    for word in words:
        if word not in word_counts:
            word_counts[word] = 0
        word_counts[word] += 1
    sorted_words = sorted(word_counts.items(), key=lambda x: x[1], reverse=True)
    cluster_names[i] = ' '.join([word[0] for word in sorted_words[:10]])

# Print the DataFrame with the cluster labels and names
for i in range(num_clusters):
    print('Cluster', i, '-', cluster_names[i])
    print()
```

```
Cluster 0 - card credit driver ride account charge service time charged never

Cluster 1 - car driver ride time company people work driving service like

Cluster 2 - account ride service customer time phone app email driver charged

Cluster 3 - driver time ride car trip charged minute customer never back
```

This code performs cluster analysis using the K-means clustering algorithm on a DataFrame called negative_df that contains only the negative sentiment sentences. The number of clusters is set to 3 using the num_clusters variable.

The K-means algorithm is fitted on the tfidf_matrix matrix, and the resulting cluster labels are added to the negative_df DataFrame. The names of the clusters are then assigned based on the most frequent words in each cluster.

The cluster_names dictionary is used to store the names of the clusters. It is created by iterating over each cluster, extracting the reviews in that cluster, and applying text pre-processing techniques such as removing stop words and lemmatization using the TextBlob and nltk libraries.

The most frequently occurring words in each cluster are then selected using a word count dictionary, and the top 10 words are concatenated to form the name of the cluster. Finally, the cluster_names dictionary is printed along with the cluster number to display the name of each cluster.

Step 3 – Topic Modelling

```
In [22]: # Create a bag-of-words matrix
vectorizer = CountVectorizer()
bow_matrix = vectorizer.fit_transform(negative_df['Review_new'])

# Perform Latent Dirichlet Allocation topic modeling
num_topics = 10
lda = LatentDirichletAllocation(n_components=num_topics, max_iter=10, learning_method='online', random_state=42)
lda.fit(bow_matrix)

# Print the top 10 words for each topic
for i, topic in enumerate(lda.components_):
    print(f"Topic {i}:")
    print([vectorizer.get_feature_names()[j] for j in topic.argsort()[:-11:-1]])
```

```
Topic 0:
['phone', 'website', 'number', 'app', 'account', 'email', 'license', 'code', 'ask', 'contact']
Topic 1:
['water', 'hidden', 'cleaning', 'como', 'century', 'tpa', 'bottle', 'indian', 'happy', 'cleaned']
Topic 2:
['password', 'negative', 'link', 'reset', 'data', 'reporter', 'verification', 'cell', 'setting', 'stay']
Topic 3:
['support', 'business', 'company', 'dont', 'ticket', 'make', 'three', 'receive', 'time', 'issue']
Topic 4:
['percent', 'ber', 'arm', 'trained', 'happens', 'mouth', 'sprinted', 'sped', 'flawed', 'slave']
Topic 5:
['damage', 'dare', 'knowledge', 'crap', 'chicago', 'estimate', 'invite', 'altered', 'market', 'filthy']
Topic 6:
['inspection', 'dealership', 'kid', 'handle', 'leasing', 'approved', 'chennai', 'touch', 'resort', 'apartment']
Topic 7:
['hotel', 'appointment', 'driver', 'take', 'lack', 'rate', 'concert', 'leave', 'foreign', 'center']
Topic 8:
['driver', 'ride', 'time', 'car', 'service', 'customer', 'charged', 'never', 'charge', 'trip']
Topic 9:
['suburb', 'awful', 'urber', 'heck', 'jive', 'veteran', 'army', 'surcharge', 'cheapskate', 'uncooperative']
```

The above code is performing topic modeling using Latent Dirichlet Allocation (LDA) on the bag-of-words matrix of positive reviews.

First, a bag-of-words matrix is created using CountVectorizer from scikit-learn library, which is a matrix representation of the frequency of words in each document (in this case, positive reviews).

Next, LDA is applied to the bag-of-words matrix to extract topics from the negative reviews. The number of topics is set to 10, and the algorithm is run for a maximum of 10 iterations using the online learning method.

Finally, the top 10 words for each topic are printed using the get_feature_names() function of the vectorizer and the argsort() function to sort the topic in descending order.

Conclusion drafted - Bad Things about our Product-

1. Bad condition of Car in Canada
2. Uncooperative
3. Booking facility horrible as compared to competition
4. Too much information asked while installing like number, email, licence
5. Verification with code sometime fails.

Phase 2

Q1. Take the most recent 25% of the reviews and done the phase 1 analysis once again to understand if the findings are consistent.

```
In [3]: df = pd.read_excel('merged_review.xlsx')
        df.head(10)
```

Out[3]:

	Date	Stars	Sentiment	Review
0	29-10-2019	1	Negative	I had an accident with an Uber driver in Mexic...
1	28-10-2019	1	Negative	I have had my account completely hacked to whe...
2	27-10-2019	1	Negative	I requested an 8 mile ride in Boston on a Satu...
3	27-10-2019	1	Negative	I've been driving off and on with the company ...
4	25-10-2019	1	Negative	Uber is overcharging for Toll fees. When In Fl...
5	24-10-2019	1	Negative	I had an airport flight today. Uber would not ...
6	24-10-2019	1	Negative	I worked for Uber and Lyft for 2.5 years and a...
7	23-10-2019	1	Positive	In July of this year I had sushi delivered to ...
8	23-10-2019	1	Negative	My driver, Rohan was nice, but when I tried to...
9	21-10-2019	1	Negative	I had seven fraudulent Uber transactions over ...

The lines executed to use pandas library to extract the dataset and have a view of the sentiments.

```
In [117]: # Convert the 'Date' column to datetime format
          df['Date'] = pd.to_datetime(df['Date'])

          # Sort the dataframe by date in descending order
          Uber_df_sorted = df.sort_values('Date', ascending=False)

          # Calculate the number of rows in the dataframe
          num_rows = Uber_df_sorted.shape[0]

          # Calculate the number of rows to keep (i.e. the most recent 25%)
          num_rows_to_keep = int(num_rows * 0.25)

          # Take the most recent 25% of the records
          Uber_df_recent = Uber_df_sorted.head(num_rows_to_keep)
```

This code processes a pandas DataFrame that contains reviews for Uber by doing the following steps:

1. Converts the 'Date' column to datetime format using pd.to_datetime() method.
2. Sorts the DataFrame in descending order of date using sort_values() method with ascending=False parameter.
3. Calculates the number of rows in the DataFrame using the shape() method and stores it in num_rows variable.
4. Calculates the number of rows to keep, which is the most recent 25% of the records, and stores it in num_rows_to_keep variable using integer division.
5. Selects the most recent 25% of the records using head() method and stores it in Uber_df_recent variable.

```
In [119]: import pandas as pd
          import nltk
          from textblob import TextBlob
          from collections import Counter
          import matplotlib.pyplot as plt
```

```
In [120]: def preprocess(text):
              text = text.lower()
              text=text.strip()  #get rid of leading/trailing whitespace
              text = re.sub(r'[^a-zA-Z0-9\s]', '', text) # remove all the symbols which are not belong to this set
              text = re.sub(r'X.*X', '', text)  # # remove all the masked words that start with 'X' and end with 'X'.
              text = re.sub(r'x.*x', '', text)
              return text

          Uber_df_recent["Review"]= Uber_df_recent["Review"].apply(preprocess) # Creating a new preprocessed column
```

Initially we imported libraries like – pandas, nltk, textblob, collection and matplotlib. Then, function preprocess() definition takes a text string as input and performs several pre-processing steps on it such as converting it to lowercase, removing leading/trailing whitespaces, removing non-alphanumeric characters, and removing masked words that start with 'X' or 'x' and end with 'X' or 'x'. Then it applies this preprocess() function to the "Review" column of a Pandas DataFrame called Uber_df_recent using the apply() method and creates a new preprocessed column called "Review".

```
In [122]: # Creating a new column for sentiment scores
          Uber_df_recent['sentiment_score'] = Uber_df_recent['Review'].apply(lambda x: TextBlob(x).sentiment.polarity)

          # Defining a function to categorize sentiment scores into positive, negative or neutral
          def get_sentiment_category(score):
              if score > 0:
                  return 'Positive'
              else:
                  return 'Negative'


          # Creating a new column for sentiment categories
          Uber_df_recent['sentiment_category'] = Uber_df_recent['sentiment_score'].apply(get_sentiment_category)

          # Printing the overall sentiment of the product
          print('Overall sentiment:', Uber_df_recent['sentiment_category'].value_counts().idxmax())
```

```
Overall sentiment: Negative
```

Performs sentiment analysis on the text data in a DataFrame column called "Review" using the TextBlob library. Here's a brief explanation of each step:

1. Create a new column for sentiment scores: A new column called "sentiment_score" is created in the DataFrame called "df". The "apply" method is used with a lambda function to apply TextBlob's sentiment analysis to each review in the "Review" column and extract the polarity score.
2. Define a function to categorize sentiment scores: A function called "get_sentiment_category" is defined to categorize sentiment scores as positive or negative. If the score is greater than 0, the function returns 'Positive', otherwise it returns 'Negative'.
3. Create a new column for sentiment categories: A new column called "sentiment_category" is created in the DataFrame by applying the "get_sentiment_category" function to the "sentiment_score" column.
4. Print the overall sentiment of the product: The code prints the overall sentiment of the product by counting the number of positive and negative sentiment categories in the "sentiment_category" column and returning the most frequent category using the "idxmax" method.

Overall, this code performs sentiment analysis on text data and categorizes it as positive or negative, allowing for an easy way to understand the sentiment of the product.

In conclusion, we can say, that the most recent 25% of the data (reviews) given by users are negative. It means, the service provided by Uber has deteriorated over the time which can be concerning factor for the company.

```
In [123]: Uber_df_recent
```

Out[123]:

	Date	Stars	Sentiment	Review	sentiment_score	sentiment_category
19	2019-12-10	1	Negative	uber quoted me a fare of 14 their driver turne...	-0.241250	Negative
77	2019-12-06	1	Negative	i am a senior citizen and i tried to sign up f...	-0.450000	Negative
20	2019-11-10	1	Positive	driver was fine and so was ride experience of ...	0.211111	Positive
34	2019-11-09	5	Positive	the drivers are well trainedwell disciplined a...	0.000000	Negative
44	2019-11-07	1	Negative	i requested a ride from fort worth south hende...	0.200000	Positive
...	...	...	...	...	...	...
521	2018-06-12	1	Negative	yesterday i took an uber from union nj to stat...	-0.300000	Negative
522	2018-06-12	1	Negative	beware anyone who rides with uber they are dea...	0.042857	Positive
523	2018-06-12	1	Negative	yesterday i took an uber from the airport 43 d...	-0.156250	Negative
524	2018-06-10	1	Negative	not only was my driver unable to find me we ha...	-0.075000	Negative
525	2018-06-10	1	Negative	on 672018 i used my uber gift card to try to b...	-0.185714	Negative

526 rows × 6 columns

To look at the dataframe if giving an elaborated sentiment score and sentiment category

Q2. Step 1 - Collecting all positive sentiments

```
In [124]: # create a new dataframe with only positive sentiment sentences-
positive_df = Uber_df_recent.loc[Uber_df_recent['sentiment_score'] > 0, ['Date', 'Stars', 'Sentiment','Review','sentiment_categor

# print the positive dataframe
positive_df.head(10)
```

Out[124]:

	Date	Stars	Sentiment	Review	sentiment_category
20	2019-11-10	1	Positive	driver was fine and so was ride experience of ...	Positive
44	2019-11-07	1	Negative	i requested a ride from fort worth south hende...	Positive
112	2019-11-05	1	Negative	i use uber service at least 20 times a month i...	Positive
1	2019-10-28	1	Negative	i have had my account completely hacked to whe...	Positive
3	2019-10-27	1	Negative	ive been driving off and on with the company s...	Positive
2	2019-10-27	1	Negative	i requested an 8 mile ride in boston on a satu...	Positive
5	2019-10-24	1	Negative	i had an airport flight today uber would not a...	Positive
10	2019-10-21	1	Negative	our driver never showed up and uber cancelled ...	Positive
11	2019-10-20	1	Negative	when the service worked it was good and tech s...	Positive
13	2019-10-18	1	Negative	they didnt have cars available at bush interna...	Positive

This code creates a new DataFrame called "positive_df" by filtering the original DataFrame called "df" to only include reviews with a positive sentiment score. Here's a brief explanation of each step:

1. Filter the original DataFrame to only include positive sentiment reviews: A new DataFrame called "positive_df" is created by filtering the "df" DataFrame using the "loc" method. The condition "df['sentiment_score'] > 0" filters the rows of the DataFrame to only include reviews with a positive sentiment score. The columns 'Date', 'Stars', 'Sentiment', 'Review', and 'sentiment_category' are selected for the new DataFrame using the list of column names.

2. Print the positive DataFrame: The code prints the first 10 rows of the "positive_df" DataFrame using the "head" method. This allows us to examine the content of the DataFrame to ensure it was filtered correctly.

In [386]:
```
# Required Preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = stopwords.words('english')
VERB_CODES = {'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'}
stop_words = set(stopwords.words('english'))
new_stop_words = ['the','and','uber','get','would','raleigh','sacca','empresa','dinero','que','got','use','used','one',
                  'youll','youve','said','first','make', 'try','take','ive', 'also', 'could', 'however', 'epect',
                 'driver', 'ride', 'car', 'charge', 'take',  'even', 'never', 'way', 'trip', 'tell', 'say']
# 'boyfriend','mother',,'mom','picture','sister','usd', 'lyft'
stop_words.update(new_stop_words)
```

This code sets up the required pre-processing steps for the text data. Here's a brief explanation of each step:

1. WordNetLemmatizer: The WordNetLemmatizer object is created from the NLTK library. This object is used to perform lemmatization on the text data, which involves reducing words to their base form (e.g., "running" -> "run").
2. Stopwords: Stopwords are common words that are often removed from text data because they don't carry much meaning (e.g., "the", "and", "a"). The stopwords for the English language are loaded from the NLTK library using the "stopwords.words('english')" method.
3. VERB_CODES: The VERB_CODES is a set that contains the codes for different types of verbs (e.g., 'VB' for base form, 'VBD' for past tense). This will be used later in the code to identify verbs for lemmatization.
4. Additional stopwords: A list of additional stopwords is created and added to the stop_words set. This list includes words that are specific to the context of the text data and may not be relevant for analysis.

In [387]:
```
stop_words = set(stopwords.words('english'))
stop_words.update(new_stop_words)


def preprocess(text):
    text = text.lower()
    text=text.strip()  #get rid of leading/trailing whitespace
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text) # remove all the symbols which are not belong to this set
    text = re.sub(r'X.*X', '', text)  # # remove all the masked words that start with 'X' and end with 'X'.
    text = re.sub(r'x.*x', '', text)  # remove all the masked words that start with 'x' and end with 'x'.
    temp_sent =[]
    tokens = nltk.word_tokenize(text)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    doc = ' '.join(filtered_tokens)
    words = nltk.word_tokenize(doc)
    tags = nltk.pos_tag(words)
    for i, word in enumerate(words):
        if tags[i][1] in VERB_CODES:
            lemmatized = lemmatizer.lemmatize(word, 'v')
        else:
            lemmatized = lemmatizer.lemmatize(word)
        if lemmatized not in stop_words and lemmatized.isalpha() and len(lemmatized)>2:
            temp_sent.append(lemmatized)
    finalsent = ' '.join(temp_sent)
    return finalsent

positive_df["clean_text"]= positive_df["Review"].apply(preprocess) # Creating a new preprocessed column
```

The code defines a preprocessing function named preprocess which performs several text cleaning operations like lowercasing, removing leading/trailing whitespaces, removing symbols, removing masked words that start and end with 'X' or 'x', tokenizing, lemmatizing, removing stop words and retaining only alphabetic words with a length greater than 2.

The function is applied to the 'Review' column of the 'positive_df' dataframe using the apply method, and the resulting preprocessed text is stored in a new column named 'Review_new'.

```
In [389]: # normalize_corpus = np.vectorize(normalize_document)

          norm_corpus = list(positive_df['clean_text'])
          norm_corpus

Out[389]: ['fine experience deal coupon look human talk amaze find think enough hire customer service kind couldnt find human online',
           'request fort worth south henderson street irving teimum busy time',
           'service least time month additional stop contact help website helpful receive response additional stop justify turn additio
          nal amount ask time refund money add account future received response matter fact help desk keep ping without response time h
          appen time money refund address every time add',
           'account completely hack sign view someone spend credit debit card link account dial number available safety line need wait
          another department email didnt hear anything almost hour prompt message help almost day still receive phone call helpful supp
          ort via email nightmare worst customer service ever experience file police report person catch fraudulently account look atto
          rney identity theft priority rectify situation end',
```

The code explains taking ‘clean text’ column of the "positive_df" dataframe in list format and executing it.

```
In [390]: def func(n):
              return n*n

          normalize = np.vectorize(func)

          norm = normalize([1,2,3,4])
          print(norm)

          [ 1  4  9 16]

In [391]: from sklearn.feature_extraction.text import CountVectorizer

          cv = CountVectorizer(ngram_range=(1, 3), min_df=25, max_df=0.9
                               )
          cv_matrix = cv.fit_transform(norm_corpus)
          cv_matrix.shape

Out[391]: (262, 37)
```

The first part vectorized the random numbers 1 to 4 to square of the numbers.

The second code defines a CountVectorizer object with the specified parameters, including using n-grams of size 1 to 3, only including terms that appear in at least 25 documents, and excluding terms that appear in more than 90% of the documents. It then applies the CountVectorizer to the preprocessed corpus "norm_corpus", generating a sparse matrix of term frequencies. The shape of the resulting matrix is printed to the console.

In [392]:
```
# view dense representation
# warning might give a memory error if data is too big
cv_matrix = cv_matrix.toarray()
cv_matrix
# get all unique words in the corpus
vocab = cv.get_feature_names()
# show document feature vectors
df = pd.DataFrame(cv_matrix, columns=vocab)
df
```

```
C:\Users\priye\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names is deprec
ated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[392]:

	account	airport	app	ask	back	call	cancel	come	company	customer	...	order	pay	phone	pick	price	service	show	time	wait	want
0	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	0
2	1	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	1	0	5	0	0
3	3	0	0	0	0	1	0	0	0	1	...	0	0	1	0	0	1	0	0	1	0
4	0	0	0	0	0	0	0	0	1	0	...	0	2	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
257	6	0	1	0	2	0	0	0	0	0	...	0	0	3	0	0	1	2	2	0	0
258	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	1	0	0
259	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	0
260	1	0	0	0	0	0	0	1	0	1	...	0	0	0	1	0	0	0	0	0	0
261	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

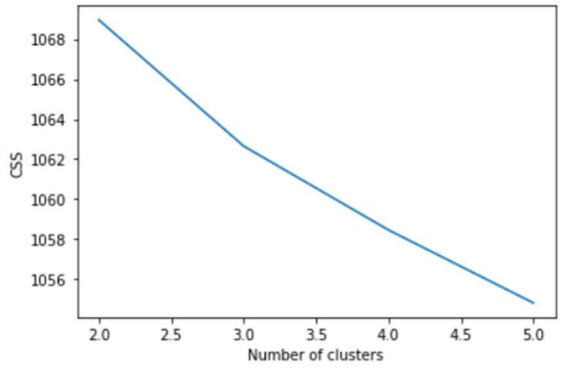
262 rows × 37 columns

This code first converts the sparse matrix cv_matrix into a dense representation using the toarray() method. Then it creates a dataframe df with the dense matrix, where the columns are the unique words in the corpus, obtained from the get_feature_names() method of the CountVectorizer object cv. The resulting dataframe displays the count of each word in each document of the corpus.

Step 2 – Clustering

In [393]:
```
css = []
for i in range(2, 6):
    kmeans = KMeans(n_clusters=i, max_iter=100, n_init=10, random_state=42).fit(tfidf_matrix)
    css.append(kmeans.inertia_)
```

In [394]:
```
import matplotlib.pyplot as plt
plt.plot(range(2, 6), css)
plt.xlabel('Number of clusters')
plt.ylabel('CSS')
plt.show()
```



This first code is performing KMeans clustering on a term frequency-inverse document frequency (tf-idf) matrix, which is represented by tfidf_matrix. The KMeans() function is being used to create

KMeans models with varying numbers of clusters ranging from 2 to 5. For each model, the inertia_ attribute is being calculated and stored in a list called css. The inertia_ attribute is a measure of how far the points within each cluster are from the centroid of the cluster. The purpose of this code is to evaluate the optimal number of clusters to use for the KMeans clustering algorithm by analysing the within-cluster sum of squares (CSS) for each number of clusters. The elbow method can be applied to the CSS values to determine the optimal number of clusters to use for the model.

The second code is the explanation of the how to represent the K-Means clustering in a line chart format as a visual representation.

```
In [395]: from sklearn.cluster import KMeans

          # Let us build the first model
          NUM_CLUSTERS = 3
          km1 = KMeans(n_clusters=NUM_CLUSTERS, max_iter=100, n_init=15, random_state=42).fit(cv_matrix)
          km1

Out[395]: KMeans(max_iter=100, n_clusters=3, n_init=15, random_state=42)

In [396]: km1.labels_

Out[396]: array([1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 0, 1, 1, 0, 0,
                 1, 0, 1, 1, 1, 0, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                 1, 1, 1, 1, 2, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
                 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 0, 0, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 2, 2, 2, 2, 1, 2, 0, 1, 1, 2, 2, 0, 0, 1, 1, 2, 1, 1, 0, 1, 1,
                 1, 0, 1, 1, 1, 0, 2, 0, 1, 1, 1, 2, 1, 2, 0, 0, 0, 1, 0, 0, 2, 1,
                 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 2, 1, 1, 0, 1])
```

This code uses the KMeans clustering algorithm from the scikit-learn library to build a model with 3 clusters based on a bag-of-words matrix created using CountVectorizer. The max_iter parameter sets the maximum number of iterations of the KMeans algorithm for a single run, while n_init sets the number of times the algorithm will be run with different random initializations. The random_state parameter ensures that the results are reproducible. The fit() method is used to fit the KMeans model to the input data (cv_matrix), and the resulting model is stored in the km1 variable.

Then the labels are executed for KMeans cluster formed.

In [397]:
```
from collections import Counter

Counter(km1.labels_)
```

Out[397]:
```
Counter({1: 211, 2: 19, 0: 32})
```

In [398]:
```
# Adding Cluster levels in the original dataframe
positive_df['kmeans_cluster'] = km1.labels_
positive_df.head(10)
```

Out[398]:

	Date	Stars	Sentiment	Review	sentiment_category	clean_text	kmeans_cluster
20	2019-11-10	1	Positive	driver was fine and so was ride experience of ...	Positive	fine experience deal coupon look human talk am...	1
44	2019-11-07	1	Negative	i requested a ride from fort worth south hende...	Positive	request fort worth south henderson street irvi...	1
112	2019-11-05	1	Negative	i use uber service at least 20 times a month i...	Positive	service least time month additional stop conta...	2
1	2019-10-28	1	Negative	i have had my account completely hacked to whe...	Positive	account completely hack sign view someone spen...	2
3	2019-10-27	1	Negative	ive been driving off and on with the company s...	Positive	drive company since since drive new place noti...	1
2	2019-10-27	1	Negative	i requested an 8 mile ride in boston on a satu...	Positive	request mile boston saturday night around come...	1
5	2019-10-24	1	Negative	i had an airport flight today uber would not a...	Positive	airport flight today accept credit card venmo ...	1
10	2019-10-21	1	Negative	our driver never showed up and uber cancelled ...	Positive	show cancel cancellation fee cancel dispute cr...	1
11	2019-10-20	1	Negative	when the service worked it was good and tech s...	Positive	service work good tech support respond quickly...	1
13	2019-10-18	1	Negative	they didnt have cars available at bush interna...	Positive	didnt available bush international airport oct...	1

This code uses the Counter function from the collections module to count the number of data points in each cluster assigned by the KMeans clustering algorithm. Then, it assigns the cluster labels to a new column in the positive_df DataFrame, and displays the first 10 rows of the DataFrame to confirm the new column was added.

```
In [399]: Review_clusters = (positive_df[['kmeans_cluster', 'Stars']]
                             .sort_values(by=['kmeans_cluster', 'Stars'],
                                          ascending=False)
                             .groupby('kmeans_cluster').head(20))
          #movie_clusters = movie_clusters.copy(deep=True)
          Review_clusters.head(30)
```

Out[399]:

	kmeans_cluster	Stars
112	2	1
1	2	1
114	2	1
65	2	1
109	2	1
185	2	1
250	2	1
251	2	1
253	2	1
257	2	1
265	2	1
267	2	1
270	2	1

This code is creating a new DataFrame called Review_clusters which is derived from the positive_df DataFrame. It selects only two columns, kmeans_cluster and Stars, sorts them in descending order based on the values in kmeans_cluster followed by Stars, groups the values by kmeans_cluster, selects the top 20 records for each kmeans_cluster, and assigns the result to Review_clusters. Finally, it prints the first 30 records of Review_clusters.

```
In [400]: feature_names = cv.get_feature_names()
          topn_features = 8
          ordered_centroids = km1.cluster_centers_.argsort()[:, ::-1]

          # get key features for each cluster
          # get movies belonging to each cluster
          for cluster_num in range(NUM_CLUSTERS):
              key_features = [feature_names[index]
                              for index in ordered_centroids[cluster_num, :topn_features]]
          #     parks = Review_clusters[Review_clusters['kmeans_cluster'] == cluster_num]['Park'].values.tolist()
              print('CLUSTER #'+str(cluster_num+1))
              print('Key Features:', key_features)
          #     print('Popular parks:', parks)
              print('-'*80)
```

```
CLUSTER #1
Key Features: ['customer', 'service', 'cancel', 'number', 'company', 'phone', 'app', 'call']
--------------------------------------------------------------------------------
CLUSTER #2
Key Features: ['time', 'service', 'drive', 'app', 'airport', 'pay', 'show', 'pick']
--------------------------------------------------------------------------------
CLUSTER #3
Key Features: ['account', 'time', 'money', 'back', 'service', 'day', 'app', 'pay']
--------------------------------------------------------------------------------
```

This code is used to extract the key features for each of the clusters generated using KMeans clustering. It first gets the feature names from the CountVectorizer used earlier in the code. It then

identifies the top n (in this case 8) features for each of the clusters by sorting the cluster centroids in descending order and getting the top features for each. It then prints the cluster number, the key features, and a separator to differentiate between clusters.

Step 3 – Topic Modelling

```
In [402]: # Create a bag-of-words matrix
vectorizer = CountVectorizer()
bow_matrix = vectorizer.fit_transform(positive_df['clean_text'])

# Perform Latent Dirichlet Allocation topic modeling
num_topics = 5
lda = LatentDirichletAllocation(n_components=num_topics, max_iter=10, learning_method='online', random_state=42)
lda.fit(bow_matrix)

# Print the top 10 words for each topic
for i, topic in enumerate(lda.components_):
    print(f"Topic {i+1}:")
    print([vectorizer.get_feature_names()[j] for j in topic.argsort()[:-11:-1]])
```

```
Topic 1:
['app', 'time', 'card', 'cancel', 'request', 'service', 'min', 'call', 'want', 'problem']
Topic 2:
['destination', 'time', 'service', 'pool', 'come', 'address', 'location', 'good', 'provide', 'pick']
Topic 3:
['account', 'pay', 'time', 'week', 'service', 'email', 'phone', 'customer', 'number', 'picture']
Topic 4:
['cancel', 'payment', 'company', 'order', 'insurance', 'credit', 'refund', 'card', 'price', 'work']
Topic 5:
['airport', 'time', 'wait', 'home', 'know', 'show', 'minute', 'last', 'call', 'good']
```

This code is performing Latent Dirichlet Allocation (LDA) topic modeling on a text dataset after creating a bag-of-words matrix using CountVectorizer. The text dataset is assumed to be stored in a pandas DataFrame column named 'clean_text'. The code creates an LDA model with 5 topics using LatentDirichletAllocation from scikit-learn library, fits the model to the bag-of-words matrix, and then prints the top 10 words for each of the 5 topics.

Topic 1: Cancelled service/ ride
Topic 2: Pick and drop
Topic 3: Customer Service
Topic 4: Cancellation and insurance policy
Topic 5: Good Refund Policy

Q3. Step 1 - Collecting all negative sentiments

In [452]:
```
# create a new dataframe with only positive sentiment sentences-
negative_df = Uber_df_recent.loc[Uber_df_recent['sentiment_score'] < 0, ['Date', 'Stars', 'Sentiment','Review','sentiment_categor

# print the positive dataframe
negative_df.head(10)
```

Out[452]:

	Date	Stars	Sentiment	Review	sentiment_category
19	2019-12-10	1	Negative	uber quoted me a fare of 14 their driver turne...	Negative
77	2019-12-06	1	Negative	i am a senior citizen and i tried to sign up f...	Negative
111	2019-11-05	1	Negative	rates can be astronomical cancellation rates a...	Negative
6	2019-10-24	1	Negative	i worked for uber and lyft for 25 years and al...	Negative
8	2019-10-23	1	Negative	my driver rohan was nice but when i tried to a...	Negative
7	2019-10-23	1	Positive	in july of this year i had sushi delivered to ...	Negative
9	2019-10-21	1	Negative	i had seven fraudulent uber transactions over ...	Negative
12	2019-10-18	1	Positive	being a responsible driver i made the decision...	Negative
16	2019-10-14	1	Negative	upfront pricing not true pricing uber pads the...	Negative
17	2019-10-14	1	Negative	i dont believe there is any way to contact ube...	Negative

This code creates a new DataFrame called "negativee_df" by filtering the original DataFrame called "df" to only include reviews with a negative sentiment score. Here's a brief explanation of each step:

1. Filter the original DataFrame to only include negative sentiment reviews: A new DataFrame called "negative_df" is created by filtering the "df" DataFrame using the "loc" method. The condition "df['sentiment_score'] > 0" filters the rows of the DataFrame to only include reviews with a negative sentiment score. The columns 'Date', 'Stars', 'Sentiment', 'Review', and 'sentiment_category' are selected for the new DataFrame using the list of column names.
2. Print the negative DataFrame: The code prints the first 10 rows of the "negative_df" DataFrame using the "head" method. This allows us to examine the content of the DataFrame to ensure it was filtered correctly.

In [488]:
```
# Required Preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = stopwords.words('english')
VERB_CODES = {'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'}
stop_words = set(stopwords.words('english'))
new_stop_words = ['the','and','uber','get','would','raleigh','sacca','empresa','dinero','que','got','use','used','one',
                  'youll','youve','said','first','make', 'try','take','ive', 'also', 'could', 'however', 'epect','dont', 'become'
                  'set','driver', 'ride', 'car', 'charge', 'take',  'even', 'never', 'way', 'trip', 'tell', 'say', 'give']
# 'boyfriend','mother',,'mom','picture','sister','usd', 'lyft'
stop_words.update(new_stop_words)
```

This code sets up the required pre-processing steps for the text data. Here's a brief explanation of each step:

1. WordNetLemmatizer: The WordNetLemmatizer object is created from the NLTK library. This object is used to perform lemmatization on the text data, which involves reducing words to their base form (e.g., "running" -> "run").
2. Stopwords: Stopwords are common words that are often removed from text data because they don't carry much meaning (e.g., "the", "and", "a"). The stopwords for the English language are loaded from the NLTK library using the "stopwords.words('english')" method.
3. VERB_CODES: The VERB_CODES is a set that contains the codes for different types of verbs (e.g., 'VB' for base form, 'VBD' for past tense). This will be used later in the code to identify verbs for lemmatization.
4. Additional stopwords: A list of additional stopwords is created and added to the stop_words set. This list includes words that are specific to the context of the text data and may not be relevant for analysis.

In [489]:
```
stop_words = set(stopwords.words('english'))
stop_words.update(new_stop_words)


def preprocess(text):
    text = text.lower()
    text=text.strip()  #get rid of leading/trailing whitespace
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text) # remove all the symbols which are not belong to this set
    text = re.sub(r'X.*X', '', text)  # # remove all the masked words that start with 'X' and end with 'X'.
    text = re.sub(r'x.*x', '', text)  # remove all the masked words that start with 'x' and end with 'x'.
    temp_sent =[]
    tokens = nltk.word_tokenize(text)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    doc = ' '.join(filtered_tokens)
    words = nltk.word_tokenize(doc)
    tags = nltk.pos_tag(words)
    for i, word in enumerate(words):
        if tags[i][1] in VERB_CODES:
            lemmatized = lemmatizer.lemmatize(word, 'v')
        else:
            lemmatized = lemmatizer.lemmatize(word)
        if lemmatized not in stop_words and lemmatized.isalpha() and len(lemmatized)>2:
            temp_sent.append(lemmatized)
    finalsent = ' '.join(temp_sent)
    return finalsent

negative_df["clean_text"]= negative_df["Review"].apply(preprocess) # Creating a new preprocessed column
```

The code defines a preprocessing function named preprocess which performs several text cleaning operations like lowercasing, removing leading/trailing whitespaces, removing symbols, removing masked words that start and end with 'X' or 'x', tokenizing, lemmatizing, removing stop words and retaining only alphabetic words with a length greater than 2.

The function is applied to the 'Review' column of the 'negative_df' dataframe using the apply method, and the resulting preprocessed text is stored in a new column named 'Review_new'.

In [491]:
```
# normalize_corpus = np.vectorize(normalize_document)

norm_corpus = list(negative_df['clean_text'])
norm_corpus
```

```
request corrected bill beware',
 'july year sushi deliver mexico ubereats subsidiary five occasion total cost peso bill dollar credit card something discover
ed end month pay credit card contact customer support provide number call customer service work via email messaging due glitc
h app refund',
 'seven fraudulent transaction three day midoctober credit card nearly week unsuccessfully customer service help totally unco
operative refuse engage fraudulent someone system impose cancel credit card new credit card leave credit card company battle
awful buyer beware know service',
 'responsible decision drink local cantina friend live within couple mile share second person drop sole key house storage wor
k fell pocket realize leave backseat almost immediately contact unfortunately friend order friend contact netremely protect r
eceive response',
 'upfront pricing true pricing pad bill much wait time park house front porch wait watch book fee come toll surcharge plus ti
me distance drop block destination another call show house want hookup start grope front porch customer service brick wall cr
edit cash matter walk rest',
 'believe contact yisak honda civic still pick hope worst grope safety seriously pray hurt predatory',
 'drive need better currently drive concerned advantage people disability pick customer star rating think something wrong sti
ll decide accept picked gentleman realize blind conversation quickest pick ever ask usually keep request ubers know decline s
tar rider pas opportunity pick people destination kindest man ever meet realized low rating disability',
 'call woman jump drive despite show call complain fare refund flat fee refuse refund error ignore email course ill stick cab
lyft',
 'event show let start love event didnt work guest unable offer contact answer search online luck support shouldnt hard',
```

The code explains taking ‘clean text’ column of the "negative_df" dataframe in list format and executing it.

```
In [492]: def func(n):
              return n*n

          normalize = np.vectorize(func)

          norm = normalize([1,2,3,4])
          print(norm)
```

```
[ 1  4  9 16]
```

```
In [493]: from sklearn.feature_extraction.text import CountVectorizer

          cv = CountVectorizer(ngram_range=(1, 2), min_df=22, max_df=0.8
                               )
          cv_matrix = cv.fit_transform(norm_corpus)
          cv_matrix.shape
```

```
Out[493]: (224, 49)
```

The first part vectorized the random numbers 1 to 4 to square of the numbers.

The second code defines a CountVectorizer object with the specified parameters, including using n-grams of size 1 to 2, only including terms that appear in at least 22 documents, and excluding terms that appear in more than 80% of the documents. It then applies the CountVectorizer to the preprocessed corpus "norm_corpus", generating a sparse matrix of term frequencies. The shape of the resulting matrix is printed to the console.

In [494]:
```
# view dense representation
# warning might give a memory error if data is too big
cv_matrix = cv_matrix.toarray()
cv_matrix
# get all unique words in the corpus
vocab = cv.get_feature_names()
# show document feature vectors
df = pd.DataFrame(cv_matrix, columns=vocab)
df
```

```
C:\Users\priye\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names is deprec
ated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[494]:

	account	another	app	ask	away	back	book	call	cancel	card	...	response	service	show	still	support	time	wait	want	work	wrong
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	1	1	1	0	0	0
3	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	1	0
4	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
219	0	0	0	0	0	3	0	5	0	0	...	0	1	1	0	0	1	0	0	0	0
220	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
221	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
222	0	0	0	0	0	1	0	0	0	0	...	0	1	0	0	0	1	1	1	0	0
223	1	1	0	1	0	0	2	0	0	2	...	0	1	0	0	0	1	0	0	1	1

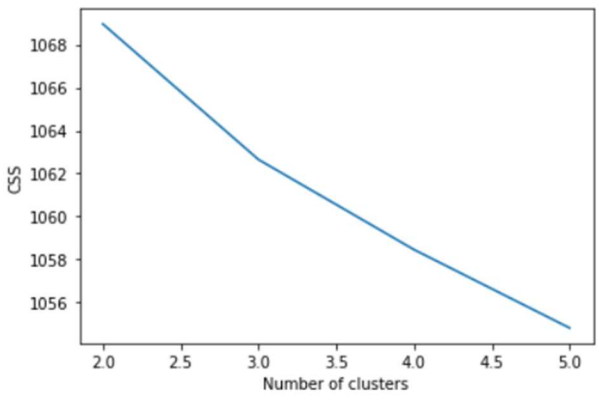
224 rows × 49 columns

This code first converts the sparse matrix cv_matrix into a dense representation using the toarray() method. Then it creates a dataframe df with the dense matrix, where the columns are the unique words in the corpus, obtained from the get_feature_names() method of the CountVectorizer object cv. The resulting dataframe displays the count of each word in each document of the corpus.

Step 2 – Clustering

In [495]:
```
css = []
for i in range(2, 6):
    kmeans = KMeans(n_clusters=i, max_iter=100, n_init=10, random_state=42).fit(tfidf_matrix)
    css.append(kmeans.inertia_)
```

In [496]:
```
import matplotlib.pyplot as plt
plt.plot(range(2, 6), css)
plt.xlabel('Number of clusters')
plt.ylabel('CSS')
plt.show()
```



This first code is performing KMeans clustering on a term frequency-inverse document frequency (tf-idf) matrix, which is represented by tfidf_matrix. The KMeans() function is being used to create KMeans models with varying numbers of clusters ranging from 2 to 6. For each model, the inertia_

attribute is being calculated and stored in a list called css. The inertia_ attribute is a measure of how far the points within each cluster are from the centroid of the cluster. The purpose of this code is to evaluate the optimal number of clusters to use for the KMeans clustering algorithm by analysing the within-cluster sum of squares (CSS) for each number of clusters. The elbow method can be applied to the CSS values to determine the optimal number of clusters to use for the model.

The second code is the explanation of the how to represent the K-Means clustering in a line chart format as a visual representation.

```
In [497]: from sklearn.cluster import KMeans

          # Let us build the first model
          NUM_CLUSTERS = 3
          km1 = KMeans(n_clusters=NUM_CLUSTERS, max_iter=100, n_init=15, random_state=42).fit(cv_matrix)
          km1

Out[497]: KMeans(max_iter=100, n_clusters=3, n_init=15, random_state=42)

In [498]: km1.labels_

Out[498]: array([0, 1, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                 2, 0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0, 1, 0, 0, 0, 0, 1,
                 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                 0, 1, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                 1, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 1, 0, 0, 1, 0, 0, 0, 1,
                 0, 0, 0, 2])
```

This code uses the KMeans clustering algorithm from the scikit-learn library to build a model with 3 clusters based on a bag-of-words matrix created using CountVectorizer. The max_iter parameter sets the maximum number of iterations of the KMeans algorithm for a single run, while n_init sets the number of times the algorithm will be run with different random initializations. The random_state parameter ensures that the results are reproducible. The fit() method is used to fit the KMeans model to the input data (cv_matrix), and the resulting model is stored in the km1 variable.

Then the labels are executed for KMeans cluster formed.

In [499]:
```
from collections import Counter

Counter(km1.labels_)
```

Out[499]:
```
Counter({0: 189, 1: 22, 2: 13})
```

In [500]:
```
# Adding Cluster levels in the original dataframe
negative_df['kmeans_cluster'] = km1.labels_
negative_df.head(10)
```

Out[500]:

	Date	Stars	Sentiment	Review	sentiment_category	clean_text	kmeans_cluster
19	2019-12-10	1	Negative	uber quoted me a fare of 14 their driver turne...	Negative	quote fare turn late long route lose nearly co...	0
77	2019-12-06	1	Negative	i am a senior citizen and i tried to sign up f...	Negative	senior citizen sign website create account sup...	1
111	2019-11-05	1	Negative	rates can be astronomical cancellation rates a...	Negative	rate astronomical cancellation rate high custo...	0
6	2019-10-24	1	Negative	i worked for uber and lyft for 25 years and al...	Negative	work lyft year eat mostly foreign laborer ille...	0
8	2019-10-23	1	Negative	my driver rohan was nice but when i tried to a...	Negative	rohan nice add tip notice switch fare french q...	0
7	2019-10-23	1	Positive	in july of this year i had sushi delivered to ...	Negative	july year sushi deliver mexico ubereats subsid...	2
9	2019-10-21	1	Negative	i had seven fraudulent uber transactions over ...	Negative	seven fraudulent transaction three day midocto...	2
12	2019-10-18	1	Positive	being a responsible driver i made the decision...	Negative	responsible decision drink local cantina frien...	0
16	2019-10-14	1	Negative	upfront pricing not true pricing uber pads the...	Negative	upfront pricing true pricing pad bill much wai...	0
17	2019-10-14	1	Negative	i dont believe there is any way to contact ube...	Negative	believe contact yisak honda civic still pick h...	0

This code uses the Counter function from the collections module to count the number of data points in each cluster assigned by the KMeans clustering algorithm. Then, it assigns the cluster labels to a new column in the positive_df DataFrame, and displays the first 10 rows of the DataFrame to confirm the new column was added.

In [501]:
```
Review_clusters = (negative_df[['kmeans_cluster', 'Stars']]
                   .sort_values(by=['kmeans_cluster', 'Stars'],
                                ascending=False)
                   .groupby('kmeans_cluster').head(20))
#movie_clusters = movie_clusters.copy(deep=True)
Review_clusters.head(30)
```

Out[501]:

	kmeans_cluster	Stars
339	2	2
7	2	1
9	2	1
64	2	1
73	2	1
49	2	1
220	2	1
224	2	1
312	2	1
401	2	1
489	2	1
496	2	1
525	2	1
77	1	1
117	1	1

This code is creating a new DataFrame called Review_clusters which is derived from the negative_df DataFrame. It selects only two columns, kmeans_cluster and Stars, sorts them in descending order

based on the values in kmeans_cluster followed by Stars, groups the values by kmeans_cluster, selects the top 20 records for each kmeans_cluster, and assigns the result to Review_clusters. Finally, it prints the first 30 records of Review_clusters.

In [502]:
```
feature_names = cv.get_feature_names()
topn_features = 8
ordered_centroids = km1.cluster_centers_.argsort()[:, ::-1]

# get key features for each cluster
# get movies belonging to each cluster
for cluster_num in range(NUM_CLUSTERS):
    key_features = [feature_names[index]
                        for index in ordered_centroids[cluster_num, :topn_features]]
#     parks = Review_clusters[Review_clusters['kmeans_cluster'] == cluster_num]['Park'].values.tolist()
    print('CLUSTER #'+str(cluster_num+1))
    print('Key Features:', key_features)
#     print('Popular parks:', parks)
    print('-'*80)
```

```
CLUSTER #1
Key Features: ['time', 'service', 'pay', 'customer', 'drive', 'call', 'account', 'back']
--------------------------------------------------------------------------------
CLUSTER #2
Key Features: ['phone', 'call', 'number', 'app', 'email', 'customer', 'service', 'account']
--------------------------------------------------------------------------------
CLUSTER #3
Key Features: ['card', 'credit', 'cancel', 'account', 'service', 'company', 'customer', 'time']
--------------------------------------------------------------------------------
```

This code is used to extract the key features for each of the clusters generated using KMeans clustering. It first gets the feature names from the CountVectorizer used earlier in the code. It then identifies the top n (in this case 8) features for each of the clusters by sorting the cluster centroids in descending order and getting the top features for each. It then prints the cluster number, the key features, and a separator to differentiate between clusters.

Step 3 – Topic Modelling

In [503]:
```
# Create a bag-of-words matrix
vectorizer = CountVectorizer()
bow_matrix = vectorizer.fit_transform(negative_df['clean_text'])

# Perform Latent Dirichlet Allocation topic modeling
num_topics = 5
lda = LatentDirichletAllocation(n_components=num_topics, max_iter=10, learning_method='online', random_state=42)
lda.fit(bow_matrix)

# Print the top 10 words for each topic
for i, topic in enumerate(lda.components_):
    print(f"Topic {i+1}:")
    print([vectorizer.get_feature_names()[j] for j in topic.argsort()[:-11:-1]])
```

```
Topic 1:
['drive', 'home', 'cost', 'price', 'back', 'airport', 'walmart', 'lyft', 'ever', 'like']
Topic 2:
['phone', 'app', 'call', 'service', 'customer', 'card', 'time', 'email', 'account', 'number']
Topic 3:
['time', 'drive', 'pay', 'account', 'book', 'people', 'ask', 'fee', 'fare', 'nothing']
Topic 4:
['credit', 'dollar', 'show', 'card', 'place', 'year', 'check', 'picture', 'allegedly', 'fraudulent']
Topic 5:
['location', 'minute', 'mile', 'cab', 'book', 'rider', 'premium', 'always', 'pay', 'arrival']
```

This code is performing Latent Dirichlet Allocation (LDA) topic modeling on a text dataset after creating a bag-of-words matrix using CountVectorizer. The text dataset is assumed to be stored in a pandas DataFrame column named 'clean_text'. The code creates an LDA model with 5 topics using

LatentDirichletAllocation from scikit-learn library, fits the model to the bag-of-words matrix, and then prints the top 10 words for each of the 5 topics.

Topic 1: Competitor comparison : lyft
Topic 2: customer service
Topic 3: Fair difference in app for Customer and Driver
Topic 4: Fraud
Topic 5: premium service

Phase 3

Your management wants to identify three critical problems in the most recent 25% of data. To find that, collect all the negative sentiment sentences. Build an n-gram frequency model to identify the three most critical phases. Find all the sentences that contain any of these critical phases. Do a topic model on those sentences and check if the topics are also related to these three critical phases.

```
In [418]: df = pd.read_excel('merged_review.xlsx')
          df.head(10)
```

Out[418]:

	Date	Stars	Sentiment	Review
0	29-10-2019	1	Negative	I had an accident with an Uber driver in Mexic...
1	28-10-2019	1	Negative	I have had my account completely hacked to whe...
2	27-10-2019	1	Negative	I requested an 8 mile ride in Boston on a Satu...
3	27-10-2019	1	Negative	I've been driving off and on with the company ...
4	25-10-2019	1	Negative	Uber is overcharging for Toll fees. When In Fl...
5	24-10-2019	1	Negative	I had an airport flight today. Uber would not ...
6	24-10-2019	1	Negative	I worked for Uber and Lyft for 2.5 years and a...
7	23-10-2019	1	Positive	In July of this year I had sushi delivered to ...
8	23-10-2019	1	Negative	My driver, Rohan was nice, but when I tried to...
9	21-10-2019	1	Negative	I had seven fraudulent Uber transactions over ...

The lines executed to use pandas library to extract the dataset and have a view of the sentiments.

In [419]:
```
# Convert the 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Sort the dataframe by date in descending order
Uber_df_sorted = df.sort_values('Date', ascending=False)

# Calculate the number of rows in the dataframe
num_rows = Uber_df_sorted.shape[0]

# Calculate the number of rows to keep (i.e. the most recent 25%)
num_rows_to_keep = int(num_rows * 0.25)

# Take the most recent 25% of the records
Uber_df_recent = Uber_df_sorted.head(num_rows_to_keep)
```

```
C:\Users\priye\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py:1047: UserWarning: Parsing '29-10-2019' in DD/MM/YY
YY format. Provide format or specify infer_datetime_format=True for consistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
C:\Users\priye\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py:1047: UserWarning: Parsing '28-10-2019' in DD/MM/YY
YY format. Provide format or specify infer_datetime_format=True for consistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
C:\Users\priye\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py:1047: UserWarning: Parsing '27-10-2019' in DD/MM/YY
YY format. Provide format or specify infer_datetime_format=True for consistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
```

This code performs the following tasks:

- It converts the 'Date' column in the given pandas dataframe 'df' to datetime format using the 'pd.to_datetime()' method.
- It sorts the dataframe in descending order based on the 'Date' column using the 'sort_values()' method.
- It calculates the total number of rows in the sorted dataframe using the 'shape()' method.
- It calculates the number of rows to keep which is the most recent 25% of the total rows in the dataframe.
- Finally, it creates a new dataframe 'Uber_df_recent' by taking the most recent 25% records from the sorted dataframe using the 'head()' method.

In [420]:
```
stop_words = stopwords.words('english')
stop_words = set(stopwords.words('english'))
new_stop_words = ['the','and','uber','get','would','raleigh','sacca','empresa','dinero','que','got','use','used','one',
                  'youll','youve','said','first','make', 'try','take','ive', 'also', 'could', 'however', 'epect','dont', 'become'
                 'take',  'even', 'never', 'way', 'trip', 'tell', 'say', 'give', 'many', 'much', 'able', 'good', 'great', 'right'
                 'really', 'sure', 'nice', 'better', 'new', 'fine', 'ho','de', 'que', 'na', 'might', 'ave', 'begin', 'think']
stop_words.update(new_stop_words)
```

The code creates a set of stop words for use in natural language processing tasks. It starts by importing a list of stop words for the English language from the nltk library. It then adds several custom stop words, including words that are common in the context of Uber ride-sharing but are not informative in the analysis, such as "uber", "trip", and "ride". The resulting set of stop words can be used to filter out common and uninformative words from text data in order to focus on more meaningful content.

In [421]:
```
def preprocess(text):
    text = text.lower()
    text=text.strip()  #get rid of leading/trailing whitespace
    # remove all symbols except alphanumeric characters and period
    text = re.sub(r'[^\w\s.]', '', text)
    tokens = nltk.word_tokenize(text)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    doc = ' '.join(filtered_tokens)
    return doc

Uber_df_recent["Review"]= Uber_df_recent["Review"].apply(preprocess) # Creating a new preprocessed column
```

The code defines a Python function named "preprocess" that takes a string of text as input and performs several text preprocessing steps on it. These steps include converting the text to lowercase, removing leading/trailing whitespaces, removing all symbols except alphanumeric characters and

periods, tokenizing the text into individual words, filtering out stop words, and finally joining the remaining words back into a string.

The code then applies the "preprocess" function to the "Review" column of a pandas DataFrame named "Uber_df_recent" using the "apply" method. The output of this operation is a new column named "Review" in the DataFrame, which contains the preprocessed version of the original text in the "Review" column.

```
In [422]: # Creating a new column for sentiment scores
          Uber_df_recent['sentiment_score'] = Uber_df_recent['Review'].apply(lambda x: TextBlob(x).sentiment.polarity)

          # Defining a function to categorize sentiment scores into positive, negative or neutral
          def get_sentiment_category(score):
              if score > 0:
                  return 'Positive'
              else:
                  return 'Negative'


          # Creating a new column for sentiment categories
          Uber_df_recent['sentiment_category'] = Uber_df_recent['sentiment_score'].apply(get_sentiment_category)

          # Printing the overall sentiment of the product
          print('Overall sentiment:', Uber_df_recent['sentiment_category'].value_counts().idxmax())

          Overall sentiment: Negative
```

Performs sentiment analysis on the text data in a DataFrame column called "Review" using the TextBlob library. Here's a brief explanation of each step:

1. Create a new column for sentiment scores: A new column called "sentiment_score" is created in the DataFrame called "df". The "apply" method is used with a lambda function to apply TextBlob's sentiment analysis to each review in the "Review" column and extract the polarity score.
2. Define a function to categorize sentiment scores: A function called "get_sentiment_category" is defined to categorize sentiment scores as positive or negative. If the score is greater than 0, the function returns 'Positive', otherwise it returns 'Negative'.
3. Create a new column for sentiment categories: A new column called "sentiment_category" is created in the DataFrame by applying the "get_sentiment_category" function to the "sentiment_score" column.
4. Print the overall sentiment of the product: The code prints the overall sentiment of the product by counting the number of positive and negative sentiment categories in the "sentiment_category" column and returning the most frequent category using the "idxmax" method.

Overall, this code performs sentiment analysis on text data and categorizes it as positive or negative, allowing for an easy way to understand the sentiment of the product.

```
In [423]: # specify the file path and name
          file_path = 'output.txt'

          # extract the 'text' column and write it to a file
          Uber_df_recent['Review'].to_csv(file_path, header=None, index=None, sep=' ')

In [424]: # Let us work with Trip advisor data
          text = open("C:\\Users\\priye\\Desktop\\ET proj\\output.txt", encoding="utf-8").read()

In [425]: blob = TextBlob(text)
          blob.sentiment

Out[425]: Sentiment(polarity=-0.03806149138575263, subjectivity=0.44920501673028834)

In [426]: #Display all sentences separately
          blob.sentences

Out[426]: [Sentence(""quoted fare ãâ14 ."),
           Sentence("driver turned late took long route lost charged nearly ãâ35 ."),
           Sentence("complained reduced ãâ26 ."),
           Sentence("refund money ãâ14 quoted me.this theft.do thieves dishonest ."),
           Sentence(""senior citizen tried sign ."),
           Sentence("went website create account ."),
           Sentence("supplied name email address cell phone password ."),
           Sentence("afterward realized correct email address ."),
           Sentence("thus began vicious cycle ."),
           Sentence("tried change email address phone provided already ."),
           Sentence("kept asking send screenshot phone ."),
           Sentence("idea entails ."),
           Sentence("find phone customer service agent ."),
           Sentence("nowhere go complete transaction ."),
           Sentence("ended going gogograndparent lyft prefer ."),
```

The code reads a text file containing reviews from Uber_df_recent dataframe, saves it to a file named 'output.txt', and then reads the same file to analyze the sentiment of the reviews using TextBlob. It displays the overall sentiment of the text and then prints out each sentence in the text separately.

```
In [427]: len(blob.sentences)

Out[427]: 2887

In [428]: # List of all sentiment words
          blist = blob.sentiment_assessments.assessments
          blist

Out[428]: [(['late'], -0.3, 0.6, None),
           (['long'], -0.05, 0.4, None),
           (['nearly'], 0.1, 0.4, None),
           (['complained'], -0.3, 0.2, None),
           (['dishonest'], -0.3, 0.5, None),
           (['vicious'], -1.0, 1.0, None),
           (['complete'], 0.1, 0.4, None),
           (['human'], 0.0, 0.1, None),
           (['enough'], 0.0, 0.5, None),
           (['kind'], 0.6, 0.9, None),
           (['human'], 0.0, 0.1, None),
           (['worth'], 0.3, 0.1, None),
           (['exactly'], 0.25, 0.25, None),
```

The code is calculating the length of sentences extracted from a text file and then creating a list of sentiment words using the TextBlob library. The blob.sentiment_assessments.assessments function is returning a list of tuples, where each tuple contains a sentiment word, its polarity score, and subjectivity score.

In [429]:
```
nsentence = []
for p in range(len(blob.sentences)):
    if blob.sentences[p].sentiment_assessments.assessments != []:
        pl = blob.sentences[p].sentiment_assessments.assessments[0][1]
        if pl < 0:
            nsentence.append(blob.sentences[p])

print(nsentence[0:25])
```

```
[Sentence("driver turned late took long route lost charged nearly ãâ35 ."), Sentence("complained reduced ãâ26 ."), Sentence("re
fund money ãâ14 quoted me.this theft.do thieves dishonest ."), Sentence("thus began vicious cycle ."), Sentence(""service leas
t 20 times month charged additional stop contacted help website less helpful ."), Sentence("customer service support unhelpful
members hard understand ."), Sentence("anyall interactions company frustrating negative ."), Sentence("drivers conversation un
pleasant ."), Sentence("foreigner curious ask brought mexico ."), Sentence("became rude asked came look mexican husband ."), Se
ntence("took wrong route made several requested stops ."), Sentence("someone spent 1k rides credit debit cards linked account d
ialed number available safety line told need wait another department emailed ."), Sentence("nightmare worst customer service ev
er experienced ."), Sentence("driver took toll route contact company long back issue overcharge ."), Sentence("ill damned app w
ouldnt accept gift card ."), Sentence("laura"
"worked lyft 2.5 years drivers ate mostly foreign laborers illegally afraid fight back corporate crime ."), Sentence("5 attempt
s rectify problem received nothing irrelevant automated replies ."), Sentence("terrible customer service zero relevant assistan
ce 7 days requesting corrected bill ."), Sentence("contacted customer support provide number call customer service work via ema
il messaging told due glitch app charges refunded ."), Sentence("cancel credit card credit card leave credit card company batt
le awful ."), Sentence("unfortunately doesnt problems encountered ."), Sentence("nobody complain case driver rude offensive cha
rge whatever wish always extra less"
"punished circumstances beyond control ."), Sentence("asked repeatedly someone done wrong done differently talk ."), Sentence
("matter charge excuse ."), Sentence("hope worst groping .")]
```

This code is performing sentiment analysis on sentences using the TextBlob library. It first calculates the sentiment score for each sentence using the sentiment_assessments attribute of the sentence object. It then filters out the sentences with a negative sentiment score and stores them in a list called 'nsentence'. Finally, it prints the first 25 sentences in 'nsentence'.

In [430]:
```
# Number of positive sentences
len(nsentence)
```

Out[430]: 553

In [431]:
```
senti_neg = []
for i in range(len(blist)):
    plt = blist[i][1]
    if plt > 0:
        senti_neg.append(blist[i][0])

senti_neg
#Len(senti_pos)
```

Out[431]:
```
[['nearly'],
 ['complete'],
 ['kind'],
 ['worth'],
 ['exactly'],
 ['favorite'],
 ['busy'],
 ['busy'],
 ['high'],
 ['available'],
 ['completely'],
 ['available'],
 ['experienced'],
 ['full'],
 ['quickly'],
 ['high'],
 ['barely'],
 ['relevant'],
 ['full'],
```

The code calculates the number of positive sentences by iterating through the blist list and appending all the words with positive polarity scores to the senti_neg list.

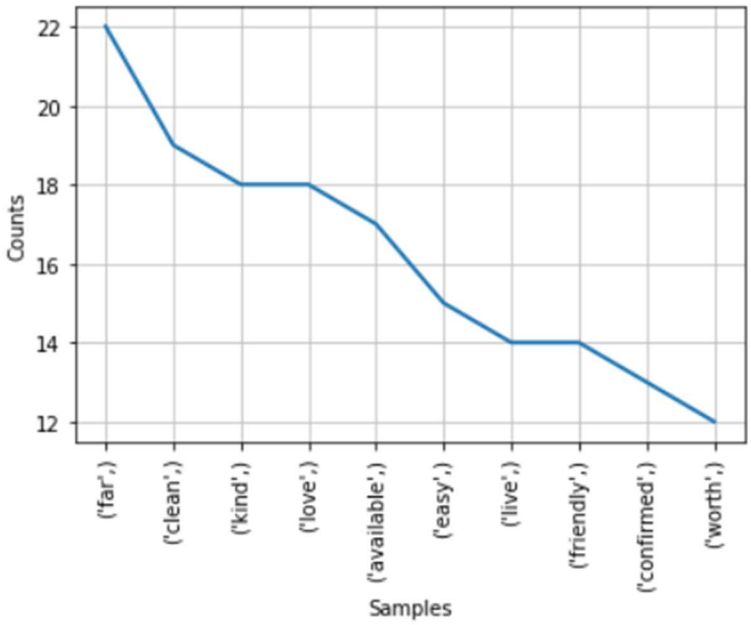
```
In [432]: # Number of positive sentences
          len(nsentence)

Out[432]: 553

In [433]: nsenti = []
          for i in range(len(senti_neg)):
              plist = senti_neg[i]
              for j in range(len(plist)):
                  nsenti.append(plist[j])

          nsenti
          #len(psenti)

Out[433]: ['nearly',
           'complete',
           'kind',
           'worth',
           'exactly',
           'favorite',
           'busy',
           'busy',
           'high',
           'available',
           'completely',
           'available',
           'experienced',
           'full',
```

This code snippet defines an empty list called `nsenti` and then iterates through the `senti_neg` list (which contains all the positive sentiment words extracted from the text), and then iterates through each word in each element of `senti_neg` to append them to the nsenti list. This essentially flattens the `senti_neg` list of lists into a single list of positive sentiment words called `nsenti`. The final line of code is a comment indicating that the length of the psenti list is being checked, but since psenti is not defined anywhere in this code snippet, it is not actually doing anything.

```
In [434]: len(set(nsenti))

Out[434]: 215

In [435]: ngram_freq = nltk.FreqDist() #WE INITIALIZED A FREQUENCY COUNTER

          for ngram in nltk.ngrams(nsenti, 1):
              ngram_freq[ngram] += 1
          ngram_freq.most_common(238)

Out[435]: [(('far',), 22),
           (('clean',), 19),
           (('kind',), 18),
           (('love',), 18),
           (('available',), 17),
           (('easy',), 15),
           (('live',), 14),
           (('friendly',), 14),
           (('confirmed',), 13),
           (('worth',), 12),
           (('high',), 12),
           (('full',), 12),
           (('safe',), 12),
           (('apparently',), 11),
           (('courteous',), 11),
           (('early',), 11),
           (('complete',), 10),
           (('experienced',), 10),
           (('absolutely',), 10),
```

This code calculates the frequency of individual words in a list of negative sentiment words. It uses the nltk package's FreqDist function to create a frequency counter, initializes it, and then iterates through each individual word (as a unigram) in the list of negative sentiment words to count their frequency. Finally, the code outputs the 238 most commonly occurring words in the list with their respective frequencies.

```
In [436]: ngram_freq.plot(10)
```




```
Out[436]: <AxesSubplot:xlabel='Samples', ylabel='Counts'>
```

Three most critical phrases: far, clean, kind

The code resembles the frequency distribution plot in the form of a line graph of the 'ngram_freq' frequency counter.

```
In [437]: # find all sentences that contain a critical phrase
matches = []
for sentence in df['Review']:
    for phrase in nsenti:
        if phrase in sentence.lower():
            matches.append(sentence)
            break

# print the matched sentences
for match in matches:
    print(match)
```

```
I had an accident with an Uber driver in Mexico City. The car that I got into had no side mirror. The Brakes were not working
properly either. I almost got into an accident twice. The driver's conversation was unpleasant. Being a foreigner he was very
curious to ask where I am from and what brought me to Mexico. I replied to be a tourist and through that conversation is ove
r. He became very rude and asked me if I came to look for a Mexican husband. I never answered and kept quiet. He took the wro
ng route and made several in requested stops. Having in mind it was uberX. He continued asked me whom I sleep with. He litera
lly stopped the car and asked me to wait for him text someone. I asked him to let me go and take a different driver but he lo
cked the doors and didn't allow me.
```

This code searches through all the sentences in a dataframe to find those containing a phrase from the nsenti list, which is a list of negative sentiment words. It stores the matched sentences in a list called matches. Finally, the code prints all the matched sentences.

In [438]:
```
# Preprocessing function
def preprocess(text):
    text = text.lower()
    text=text.strip()  #get rid of leading/trailing whitespace
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text) # remove all the symbols which are not belong to this set
    text = re.sub(r'X.*X', '', text)  # # remove all the masked words that start with 'X' and end with 'X'.
    text = re.sub(r'x.*x', '', text)  # remove all the masked words that start with 'x' and end with 'x'.
    temp_sent =[]
    tokens = nltk.word_tokenize(text)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    doc = ' '.join(filtered_tokens)
    words = nltk.word_tokenize(doc)
    tags = nltk.pos_tag(words)
    for i, word in enumerate(words):
        if tags[i][1] in VERB_CODES:
            lemmatized = lemmatizer.lemmatize(word, 'v')
        else:
            lemmatized = lemmatizer.lemmatize(word)
        if lemmatized not in stop_words and lemmatized.isalpha() and len(lemmatized)>2:
            temp_sent.append(lemmatized)
    finalsent = ' '.join(temp_sent)
    return finalsent

if type(matches) == list:
    matches = pd.DataFrame(matches, columns=['Column_Name'])
matches['Preprocessed_Column'] = matches['Column_Name'].apply(preprocess)
```

This code defines a function `preprocess` that preprocesses text data by converting it to lowercase, removing leading and trailing whitespaces, removing symbols that are not alphanumeric, removing masked words that start and end with 'X' or 'x', tokenizing the text, filtering out stop words, and lemmatizing the words using NLTK's WordNetLemmatizer.

After defining the function, the code checks if the variable `matches` is a list and converts it to a pandas DataFrame with a single column named `Column_Name` if it is. It then creates a new column named `Preprocessed_Column` in the DataFrame by applying the `preprocess` function to each row in the `Column_Name` column.

In [439]:
```
# Create a bag-of-words matrix
vectorizer = CountVectorizer()
bow_matrix = vectorizer.fit_transform(matches['Preprocessed_Column'])

# Perform Latent Dirichlet Allocation topic modeling
num_topics = 4
lda = LatentDirichletAllocation(n_components=num_topics, max_iter=10, learning_method='online', random_state=42)
lda.fit(bow_matrix)

# Print the top 10 words for each topic
for i, topic in enumerate(lda.components_):
    print(f"Topic {i+1}:")
    print([vectorizer.get_feature_names()[j] for j in topic.argsort()[:-11:-1]])
```

```
Topic 1:
['driver', 'time', 'drive', 'service', 'ride', 'airport', 'price', 'car', 'like', 'mile']
Topic 2:
['driver', 'customer', 'account', 'time', 'company', 'service', 'email', 'drive', 'money', 'phone']
Topic 3:
['driver', 'ride', 'charge', 'car', 'time', 'card', 'call', 'minute', 'app', 'cancel']
Topic 4:
['driver', 'vehicle', 'safe', 'car', 'drive', 'insurance', 'break', 'accident', 'damage', 'service']
```

This code is performing Latent Dirichlet Allocation (LDA) topic modeling on a text dataset after creating a bag-of-words matrix using CountVectorizer. The text dataset is assumed to be stored in a pandas DataFrame column named 'Preprocessed_Column'. The code creates an LDA model with 4 topics using LatentDirichletAllocation from scikit-learn library, fits the model to the bag-of-words matrix, and then prints the top 10 words for each of the 4 topics.

Conclusion: Topic 1: Airport ride Topic 2: Customer service Topic 3: Topic 4: Accident and insurance

Phase 4

Preprocessing the whole dataframe

In [23]:
```
'''# Required Preprocessing
lemmatizer = WordNetLemmatizer()
stop_words = stopwords.words('english')
VERB_CODES = {'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'}
stop_words = set(stopwords.words('english'))
new_stop_words = ['place', 'park','Kolkata','kolkata','visit','bridge','one','time','also']
stop_words.update(new_stop_words)
'''
```

Out[23]:
```
"# Required Preprocessing\nlemmatizer = WordNetLemmatizer()\nstop_words = stopwords.words('english')\nVERB_CODES = {'VB', 'VB
D', 'VBG', 'VBN', 'VBP', 'VBZ'}\nstop_words = set(stopwords.words('english'))\nnew_stop_words = ['place', 'park','Kolkata','kol
kata','visit','bridge','one','time','also']\nstop_words.update(new_stop_words)\n"
```

In [25]:
```
# Define a function to clean and preprocess the text
def clean_text(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove punctuations and special characters
    text = re.sub(r'[^\w\s]', '', text)

    # Tokenize the text
    tokens = word_tokenize(text)

    # Remove stop words
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]

    # Lemmatize the words
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Stem the words
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(token) for token in tokens]

    # Join the tokens back into a single string
    text = ' '.join(tokens)

    return text

# Apply the clean_text function to the review column
df['Review_new'] = df['Review'].apply(clean_text)
```

In [26]:
```
df
```

Out[26]:

	Date	Stars	Sentiment	Review	sentiment_score	sentiment_category	Review_new
0	29-10-2019	1	Negative	I had an accident with an Uber driver in Mexic...	-0.185556	Negative	accid uber driver mexico citi car got side mir...
1	28-10-2019	1	Negative	I have had my account completely hacked to whe...	0.033333	Positive	account complet hack sign view someon spent 1k...
2	27-10-2019	1	Negative	I requested an 8 mile ride in Boston on a Satu...	0.053333	Positive	request 8 mile ride boston saturday night arou...
3	27-10-2019	1	Negative	I've been driving off and on with the company ...	0.363939	Positive	ive drive compani sinc 2016 sinc drive new pla...
4	25-10-2019	1	Negative	Uber is overcharging for Toll fees. When In Fl...	-0.016667	Negative	uber overcharg toll fee florida toll fee estim...
...	...	...	...	...	...	...	...
2102	2016-02-23 00:00:00	1	Positive	Wow! Where do I begin?! I applied to be a driv...	0.059375	Positive	wow begin appli driver wait wait waitedwait co...
2103	2016-02-22 00:00:00	2	Positive	The first time ever in my life that I used ...	0.209773	Positive	first time ever life use uber chitown new year...
2104	2016-02-22 00:00:00	4	Negative	Short trip take a cab. Longer run out of town...	0.000000	Negative	short trip take cab longer run town airport uber
2105	2016-02-21 00:00:00	1	Positive	If you are contemplating driving for Uber in N...	0.176061	Positive	contempl drive uber nj think againmayb 3x ask ...
2106	2016-02-21 00:00:00	5	Positive	We went on vacation to Washington dc...we took...	0.712500	Positive	went vacat washington dcwe took uber citi 4 da...

2107 rows × 7 columns

This code defines a function called `clean_text` that cleans and preprocesses a text input. The function performs the following operations on the input text:

1. converts the text to lowercase
2. removes punctuations and special characters
3. tokenizes the text
4. removes stop words
5. lemmatizes the words
6. stems the words
7. joins the tokens back into a single string

After defining the function, it is applied to the 'Review' column of the DataFrame 'df' and the cleaned and preprocessed text is stored in a new column called 'Review_new'.

```
In [27]: # Split the dataset into training and testing data
         X_train, X_test, y_train, y_test = train_test_split(df['Review_new'], df['sentiment_category'], random_state=42)

         # Convert text data into numerical data using count vectorization
         vectorizer = CountVectorizer(stop_words='english')
         X_train = vectorizer.fit_transform(X_train)
         X_test = vectorizer.transform(X_test)

         # Train the model using Multinomial Naive Bayes
         clf = MultinomialNB()
         clf.fit(X_train, y_train)

         # Predict the sentiment of the reviews in the test dataset
         y_pred = clf.predict(X_test)

         # Evaluate the performance of the model using accuracy score
         accuracy = accuracy_score(y_test, y_pred)
         print("Accuracy of the classification model is: {:.2f}%".format(accuracy*100))

         Accuracy of the classification model is: 72.68%
```

This code is splitting a dataset into training and testing data using the `train_test_split` function from scikit-learn. Then, it is using the `CountVectorizer` to convert text data into numerical data. Next, a `MultinomialNB` model is trained using the training dataset. After training, the model is used to predict the sentiment of the reviews in the test dataset. Finally, the accuracy of the model is evaluated using the `accuracy_score` function from scikit-learn, and the result is printed as a percentage. After the execution, the accuracy of the classification model is 72.68%.

```
In [28]: # Split the dataset into training and testing data
         X_train, X_test, y_train, y_test = train_test_split(df['Review_new'], df['Sentiment'], random_state=42)

         # Convert text data into numerical data using count vectorization
         vectorizer = CountVectorizer(stop_words='english')
         X_train = vectorizer.fit_transform(X_train)
         X_test = vectorizer.transform(X_test)

         # Train the model using Multinomial Naive Bayes
         clf = MultinomialNB()
         clf.fit(X_train, y_train)

         # Predict the sentiment of the reviews in the test dataset
         y_pred = clf.predict(X_test)

         # Evaluate the performance of the model using accuracy score
         accuracy = accuracy_score(y_test, y_pred)
         print("Accuracy of the classification model is: {:.2f}%".format(accuracy*100))

         Accuracy of the classification model is: 80.46%
```

This code is implementing a text classification model using Multinomial Naive Bayes algorithm to predict the sentiment of reviews. The dataset is split into training and testing data using the `train_test_split` function from scikit-learn. The text data is converted into numerical data using

CountVectorizer, which creates a matrix of word counts for each document. The MultinomialNB classifier is then trained on the training data and used to predict the sentiment of the reviews in the test dataset. Finally, the accuracy of the classification model is evaluated using the `accuracy_score` function from scikit-learn and printed to the console. After the execution, the accuracy of the classification model is 80.46%.

Conclusion:

Manually predicted Sentiment have higher accuracy, i.e. of 80.46% whereas classifier based on Sentiment analyser (Textblob) has lower accuracy i.e. of 72.86%.

CONCLUSION

After looking at the reviews of the Uber app, it was seen the sentiments of the customers might be different compared to the rating they gave. For which the positive and negative sentiment reviews were collected and performed topic modelling and cluster modelling on the data to find which topics are doing well or identifying possible drawbacks of the product. Then the same thing was done on 25% of the collected data. After that collected the negative sentiments of this 25% data and then built a n-gram frequency model to identify three most critical phases and checked for sentences containing these critical phases. Then a classification model is built check the robustness to manually go through each review and tag "POSITIVE" or "NEGATIVE". Then checked got the result and the accuracy.

Thank You