Zevi - Analytics Internship Assignment

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

import textblob
from textblob import TextBlob
import string
import re
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

# sklearn
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix,
from sklearn.metrics import confusion_matrix, classification_report
```

EDA

In [ ]:
```
data = pd.read_excel('Swiggy_Dataset.xlsx')
```

In [ ]:
```
data.columns
```

In [ ]:
```
print('Count of columns in the data is:  ', len(data.columns))
print('Count of rows in the data is:  ', len(data))
```

In [ ]:
```
data.duplicated().sum()
```

In [ ]:
```
data.head()
```

In [ ]:
```
data.tail()
```

In [ ]:
```
data.shape
```

In [ ]:
```
data.isnull().sum()
```

In [ ]:
```
data.info()
```

In [ ]:
```
data["retweeted"].fillna("True",inplace = True)
```

In [ ]:
```
data.isnull().sum()
```

In [ ]:
```
data.info()
```

In [ ]:
```
 from wordcloud import WordCloud, STOPWORDS
```

In [ ]:
```
plt.figure(figsize=(15,6))
wc = WordCloud(width=600, height=300).generate(' '.join(data.full_text))
plt.imshow(wc)
```

Defining set containing all stopwords in English.

In [ ]:
```
stopwordlist = ['a', 'about', 'above', 'after', 'again', 'ain', 'all', 'am', 'an',
             'and','any','are', 'as', 'at', 'be', 'because', 'been', 'before',
             'being', 'below', 'between','both', 'by', 'can', 'd', 'did', 'do',
             'does', 'doing', 'down', 'during', 'each','few', 'for', 'from',
             'further', 'had', 'has', 'have', 'having', 'he', 'her', 'here',
             'hers', 'herself', 'him', 'himself', 'his', 'how', 'i', 'if', 'in',
             'into','is', 'it', 'its', 'itself', 'just', 'll', 'm', 'ma',
             'me', 'more', 'most','my', 'myself', 'now', 'o', 'of', 'on', 'once',
             'only', 'or', 'other', 'our', 'ours','ourselves', 'out', 'own', 're','s
             't', 'than', 'that', "thatll", 'the', 'their', 'theirs', 'them',
             'themselves', 'then', 'there', 'these', 'they', 'this', 'those',
             'through', 'to', 'too','under', 'until', 'up', 've', 'very', 'was',
             'we', 'were', 'what', 'when', 'where','which','while', 'who', 'whom',
             'why', 'will', 'with', 'won', 'y', 'you', "youd","youll", "youre",
             "youve", 'your', 'yours', 'yourself', 'yourselves']
```

In [ ]:
```
STOPWORDS = set(stopwordlist)
def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])
data['full_text'] = data['full_text'].apply(lambda text: cleaning_stopwords(text))
data['full_text'].head()
```

In [ ]:
```
len(data.index)
```

In [ ]:
```
# Total tweets
print("Total tweets this period:", len(data.index))
```

In [ ]:
```
# Retweets
tweet_df = data.sort_values(by="retweet_count", ascending = False)
```

In [ ]:
```
tweet_df
```

Top 5 Retweet

In [ ]:
```
tweet_df = data.reset_index(drop = True)
print ("Mean retweets:", round(data['retweet_count'].mean(),2))
print("Top 5 RTed tweets:")
print('-----------------------')
for i in range(5):
    print(data['full_text'].loc[i],'-', data['retweet_count'].loc[i])
```

Sentiment Analysis

In [ ]:
```
data.columns
```

In [ ]:
```
text_df = data.drop(['date', 'favorite_count', 'followers_count', 'friends_count',
```

```
        'retweet_count', 'retweeted', 'screen_name', 'tweet_id',
       'user_id'], axis =1)
text_df.head()
```

In [ ]:
```
print(text_df['full_text'].iloc[0],"\n")
print(text_df['full_text'].iloc[1],"\n")
print(text_df['full_text'].iloc[2],"\n")
print(text_df['full_text'].iloc[3],"\n")
print(text_df['full_text'].iloc[4],"\n")
```

In [ ]:
```
def data_processing(text):
    text = re.sub(r'@[A-Za-z0-9]+','', text)
    text = re.sub(r'#','',text)
    text = re.sub(r'_', '',text)
    text = re.sub(r'RT[\s]+','', text)
    text = re.sub(r'https?:\/\/\S+','', text)
    text_tokens = word_tokenize(text)
    filtered_text = [w for w in text_tokens if not w in stop_words]
    return " ".join(filtered_text)

text_df['full_text'] = text_df['full_text'].apply(data_processing)
```

In [ ]:
```
text_df = text_df.drop_duplicates('full_text')
```

In [ ]:
```
stemmer = PorterStemmer()
def stemming(data):
    text = [stemmer.stem(word) for word in data]
    return data
```

In [ ]:
```
# text_df['full_text'] = data['full_text'].apply(lambda x: stemming(x))
```

In [ ]:
```
text_df
```

In [ ]:
```
print(text_df['full_text'].iloc[0],"\n")
print(text_df['full_text'].iloc[1],"\n")
print(text_df['full_text'].iloc[2],"\n")
print(text_df['full_text'].iloc[3],"\n")
print(text_df['full_text'].iloc[4],"\n")
```

In [ ]:
```
def polarity(text):
    return TextBlob(text).sentiment.polarity
```

In [ ]:
```
text_df['polarity'] = data['full_text'].apply(polarity)
```

In [ ]:
```
text_df.head()
```

In [ ]:
```
def sentiment(label):
    if label < 0:
        return "Negative"
    elif label == 0:
        return "Neutral"
    else:
        return "Positive"
```

In [ ]:
```
text_df['sentiment'] = text_df['polarity'].apply(sentiment)
```

In [ ]:
```
text_df.head()
```

In [ ]:
```
fig = plt.figure(figsize=(5,5))
sns.countplot(x='sentiment', data= text_df)
```

In [ ]:
```
fig = plt.figure(figsize=(7,7))
colors = ("yellowgreen", "gold","red")
wp = {'linewidth':2, 'edgecolor':"black"}
tags = text_df['sentiment'].value_counts()
explode = (0.1,0.1,0.1)
tags.plot(kind='pie', autopct = '%1.1f%%', shadow = True, colors = colors,
          startangle = 90, wedgeprops = wp, explode = explode, label ='')
plt.title('Distribution  of sentiment')
```

Top Positive Sentences

In [ ]:
```
pos_tweets = text_df[text_df.sentiment == 'Positive']
pos_tweets = pos_tweets.sort_values(['polarity'],ascending = False)
pos_tweets.head()
```

In [ ]:
```
text = ' '.join([word for word in pos_tweets['full_text']])
plt.figure(figsize=(15,6), facecolor='None')
wordcloud = WordCloud(max_words = 500, width=1600, height=800).generate(text)
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis("off")
plt.title('Most frequent words in positive tweets', fontsize=19)
plt.show()
```

Top Negative Sentances

In [ ]:
```
j =1
sortedDF = text_df.sort_values(by=['polarity'])
for i in range(0, sortedDF.shape[0]):
    if(sortedDF['sentiment'][i] == 'Negative'):
        print(str(j) + ')' +sortedDF['full_text'][i])
        print()
        j = j+1
```

Top Neutral Sentences

In [ ]:
```
j =1
sortedDF = text_df.sort_values(by=['polarity'])
for i in range(0, sortedDF.shape[0]):
    if(sortedDF['sentiment'][i] == 'Neutral'):
        print(str(j) + ')' +sortedDF['full_text'][i])
        print()
        j = j+1
```

In [ ]:
```
vect = CountVectorizer(ngram_range=(1,2)).fit(text_df['full_text'])
```

In [ ]:
```
feature_names = vect.get_feature_names()
print("Number of features: {}\n".format(len(feature_names)))
print("First 20 features:\n {}".format(feature_names[:20]))
```

In [ ]:
```
X = text_df['full_text']
Y = text_df['sentiment']
X = vect.transform(X)
```

In [ ]:
```
x_train, x_test, y_train,y_test = train_test_split(X,Y, test_size = 0.2, random_stat
```

In [ ]:
```
print("Size of x_train", (x_train.shape))
print("Size of y_train", (y_train.shape))
print("Size of x_test", (x_test.shape))
print("Size of y_test", (y_test.shape))
```

Logistic Regression Model

In [ ]:
```
logreg = LogisticRegression()
logreg.fit(x_train,y_train)
logreg_pred = logreg.predict(x_test)
logreg_acc = accuracy_score(logreg_pred, y_test)
print("Test accuracy: {:.2f}%".format(logreg_acc*100))
```

In [ ]:
```
print(confusion_matrix(y_test, logreg_pred))
print('\n')
print(classification_report(y_test, logreg_pred))
```

In [ ]:
```
style.use('classic')
cm = confusion_matrix(y_test, logreg_pred, labels=logreg.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels=logreg.classes_)
disp.plot()
```

In [ ]:
```
 from sklearn.model_selection import GridSearchCV
```

In [ ]:
```
param_grid={'C':[0.001, 0.01, 0.1, 10]}
grid = GridSearchCV(LogisticRegression(), param_grid)
grid.fit(x_train,y_train)
```

In [ ]:
```
print("Best parameters:", grid.best_params_)
```

In [ ]:
```
y_pred = grid.predict(x_test)
```

In [ ]:
```
#now improve the accuracy
logreg_acc = accuracy_score(y_pred, y_test)
print("Test accuracy: {:.2f}%".format(logreg_acc*100))
```

In [ ]:
```
print(confusion_matrix(y_test, logreg_pred))
print('\n')
print(classification_report(y_test, logreg_pred))
```

SVC

In [ ]:
```
from sklearn.svm import LinearSVC
```

In [ ]:
```
SVCmodel = LinearSVC()
SVCmodel.fit(x_train,y_train)
```

In [ ]:
```
svc_pred = SVCmodel.predict(x_test)
svc_acc = accuracy_score(svc_pred,y_test)
print("test accuracy: {:.2f}%".format(svc_acc*100))
```

In [ ]:
```
print(confusion_matrix(y_test, svc_pred))
print("\n")
print(classification_report(y_test, svc_pred))
```

perform hyper paramter tunning for svm model

In [ ]:
```
grid = {
    'C':[0.01, 0.1, 1, 10],
    'kernal':["linear","poly","rbf","sigmoid"],
    'degree':[1,3,5,7],
    'gamma' :[0.01,1]
}
grid = GridSearchCV(SVCmodel, param_grid)
grid.fit(x_train, y_train)
```

In [ ]:
```
print("Best parameter:", grid.best_params_)
```

In [ ]:
```
y_pred = grid.predict(x_test)
```

In [ ]:
```
logreg_acc = accuracy_score(y_pred, y_test)
print("Test accuracy:{:.2f}%".format(logreg_acc*100))
```

In [ ]:
```
print(confusion_matrix(y_test,y_pred))
print('\n')
print(classification_report(y_test, y_pred))
```

In [ ]:
```

```