In [1]:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import re
import contractions
from unidecode import unidecode
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from string import punctuation
from rake_nltk import Rake
import yake
from nltk.stem import WordNetLemmatizer,LancasterStemmer
from gensim.models import Word2Vec
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from nltk.util import ngrams
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.naive_bayes import MultinomialNB
import yake
from rake_nltk import Rake
```

In [112]:

```python
import warnings
warnings.filterwarnings("ignore")
```

In [113]:

```python
df=pd.read_csv(r"C:\Users\kannu\OneDrive\Desktop\Pandas_csvs\Datasets\sentim
```

In [114]:

```python
df.shape
```

Out[114]:

```
(515738, 17)
```

In [118]:

```python
pos=df[["Positive_Review"]]
pos.columns=["Review"]
pos["Target"]=np.ones(len(pos),dtype=int)
```

In [119]:

```python
for ind,data in enumerate (pos["Review"]):
    if ((len(data)<14) and ((data.lower().__contains__("nothing")) or (data.
        pos["Target"].iloc[ind]=2
```

In [120]:

```python
a=" nothing"
b=" Nothing"
c="nothing"
a.lower().__contains__("nothing")
b.lower().__contains__("nothing")
c.lower().__contains__("nothing")
```

Out[120]:

```
True
```

In [122]:

```
neg=df[["Negative_Review"]]
neg.columns=["Review"]
neg["Target"]=np.zeros(len(neg),dtype=int)
```

In [123]:

```
# [word for word in neg["Review"] if len(word)<14]
```

In [124]:

```
for ind,data in enumerate (neg["Review"]):
    if ((len(data)<14) and ((data.lower().__contains__("nothing")) or (data.
        neg["Target"].iloc[ind]=2
```

In [125]:

```
neg["Target"].value_counts()
```

Out[125]:

```
0    364220
2    151518
Name: Target, dtype: int64
```

In [127]:

```
pos_df=pos.sample(500)
```

In [128]:

```
final_df=pd.concat([pos_df,neg_df],axis=0)
```

In [130]:

```
final_df.reset_index(drop=True,inplace=True)
```

In [131]:

```
final_df["Target"].value_counts()
x=final_df.drop("Target",axis=1)
y=final_df["Target"]
```

In [132]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random_state
x_train.shape,x_test.shape,y_train.shape
```

Out[132]:

```
((800, 1), (200, 1), (800,))
```

In [133]:

```
#Removing white spaces

def whitespace_removal(data):
    import re
    pattern=r"\s+"
    res=re.sub(pattern," ",data)
    return res

#contraction mapping

def expand_text(data):
    res=contractions.fix(data)
    return res

#accented character handling

def accent_handling(data):
    res=unidecode(data)
    return res
```

In [135]:

```
x_train["Review"]=x_train["Review"].apply(whitespace_removal)
x_train["Review"]=x_train["Review"].apply(expand_text)
x_train["Review"]=x_train["Review"].apply(accent_handling)
```

In [136]:

```
#datacleaning

stopword_list=stopwords.words("english")
stopword_list.remove("no")
stopword_list.remove("nor")
stopword_list.remove("not")


def cleaning(data):
    res=[word.lower() for word in word_tokenize(data) if (word.lower() not i
    return res
```

In [137]:

```
x_train_cleaned=x_train["Review"].apply(cleaning)
x_train_cleaned.reset_index(drop=True,inplace=True)

```

In [138]:

```
def ngram(data,n_range):
    ngram_list=[]
    op=ngrams(data,n_range)
    for ng in op:
        ngram_list.append(ng)

    return ngram_list
```

In [139]:

```
y_train.reset_index(drop=True,inplace=True)
```

In [140]:

```
#checking positive reviews

merged_data_for_ngrams=pd.concat([x_train_cleaned,y_train],axis=1)
pos=merged_data_for_ngrams.loc[merged_data_for_ngrams["Target"]==1]["Review"

```

In [141]:

```
y_train.index
```

Out[141]:

```
RangeIndex(start=0, stop=800, step=1)
```

In [144]:

```
#checking negative reviews

neg=merged_data_for_ngrams.loc[merged_data_for_ngrams["Target"]==0]["Review"
neg.apply(lambda x: ngram(x,4)).iloc[7]
```

Out[144]:

```
[]
```

In [145]:
```
#checking neutral reviews

neutral=merged_data_for_ngrams.loc[merged_data_for_ngrams["Target"]==2]["Rev
neutral.apply(lambda x: ngram(x,1)).iloc[17]
```

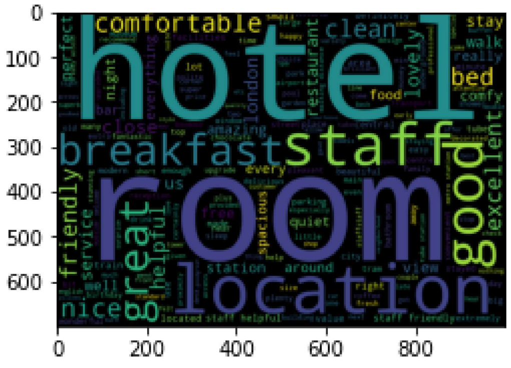
Out[145]:
```
[('no',), ('positive',)]
```

In [146]:
```
def text(data):
    return " ".join(data)

pos_string=pos.apply(text)
neg_string=neg.apply(text)
neutral_string=neutral.apply(text)
```

In [147]:
```
def df_2_string(df):
    final_text=[]
    list1=df.to_list()
    for sub_list in list1:
        val="".join(sub_list)

        final_text.append(val)
    return "".join(final_text)

```

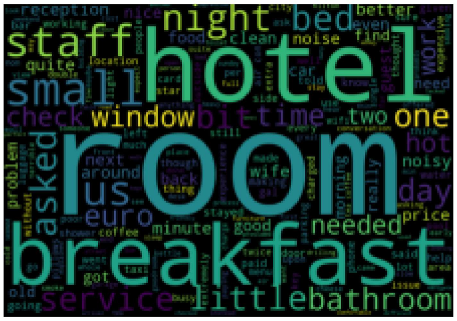
In [148]:
```
pos_text=df_2_string(pos_string)
neg_text=df_2_string(neg_string)
neutral_text=df_2_string(neutral_string)

```

In [217]:
```
#wordcloud of pos reviews
word_cloud_pos=WordCloud(height=700,width=1000,background_color="black",min_
plt.figure(figsize=(4,6))
plt.imshow(word_cloud_pos)
plt.show()
```



In [218]:

```
#wordcloud of neg reviews
word_cloud_neg=WordCloud(height=700,width=1000,background_color="black",min_
plt.figure(figsize=(4,6))
plt.imshow(word_cloud_neg)
plt.axis("off")
plt.show()
```



In [152]:

```
#lemmatization and stemming

def lemmatization(data):
    final_text=[]
    for word in data:
        lemmatizer=WordNetLemmatizer()
        lema=lemmatizer.lemmatize(word)
        final_text.append(lema)
    return " ".join(final_text)

def stemming(data):
    final_text=[]
    for word in data:
        stemmer=LancasterStemmer()
        stem=stemmer.stem(word)
        final_text.append(stem)
    return " ".join(final_text)
```

In [153]:

```
lemmatized=x_train_cleaned.apply(lemmatization)
stemmed=x_train_cleaned.apply(stemming)
```

In [158]:

```
#countvectorizer
count_vect=CountVectorizer()
bow=count_vect.fit_transform(lemmatized).A
```

In [160]:

```
#tfidf
tfidf_vect=TfidfVectorizer()
tfidf=tfidf_vect.fit_transform(lemmatized).A
```

In [162]:

```
#word to vect

#making in [[w1,w2,....],[w3,w4...]] format

def word_to_vec_ip(df):
    res=[word.split() for word in df]
    return res
```

In [163]:

```
word_vec_data=word_to_vec_ip(lemmatized)
# word_vec_data
```

In [164]:

```
word2vec=Word2Vec(word_vec_data,window=10,min_count=2)
word2vec.build_vocab(word_vec_data)
word2vec.train(word_vec_data,total_examples=word2vec.corpus_count,epochs=5)
```

Out[164]:

```
(25212, 39070)
```

In [166]:

```
def word2_vector(model,data):
    feature=[]
    zero_vector=np.zeros(model.vector_size) #a 1d array of 0.

    for vector in data:
        vectors=[]  #all vectors of a single doc will be here,[[vetor w1],[v
        for word in vector:

            if word in model.wv:
                try:
                    vectors.append(model.wv[word])
                except KeyError:
                    continue
        if vectors:
            vectors=np.array(vectors)   #it will have 2d array, means row an
            avg_vec=vectors.mean(axis=0)  # by axis=0 it will find mean roww
            feature.append(avg_vec)
        else:
            feature.append(zero_vector)



    return feature

feature=word2_vector(word2vec,word_vec_data)
```

In [167]:

```
#modelling
```

logistic_regression

In [168]:

```
#logistic regression
#with_count_vectorizer
log_reg_count=LogisticRegression()
log_reg_count.fit(bow,y_train)
log_reg_count.score(bow,y_train)
```

Out[168]:

```
0.98125
```

In [169]:

```
#with_tfidf
log_reg_tfidf=LogisticRegression()
log_reg_tfidf.fit(tfidf,y_train)
log_reg_tfidf.score(tfidf,y_train)
```

Out[169]:

```
0.975
```

In [170]:

```
#with word2vec
log_reg_word2vec=LogisticRegression()
log_reg_word2vec.fit(feature,y_train)
log_reg_word2vec.score(feature,y_train)
```

Out[170]:

```
0.6475
```

KNN

In [171]:

```
#with_count_vectorizer
knn_count=KNeighborsClassifier()
knn_count.fit(bow,y_train)
knn_count.score(bow,y_train)
```

Out[171]:

```
0.78375
```

In [172]:

```
#with_tfidf
knn_tfidf=KNeighborsClassifier()
knn_tfidf.fit(tfidf,y_train)
knn_tfidf.score(tfidf,y_train)
```

Out[172]:

```
0.35375
```

In [173]:

```
#with_word2vec
knn_word2vec=KNeighborsClassifier()
knn_word2vec.fit(feature,y_train)
knn_word2vec.score(feature,y_train)
```

Out[173]:

```
0.80625
```

DT

In [174]:

```python
#with count_vectorizer
dt_count=DecisionTreeClassifier()
dt_count.fit(bow,y_train)
dt_count.score(bow,y_train)

```

Out[174]:

```
0.99625
```

In [175]:

```python
#with tfidf
dt_tfidf=DecisionTreeClassifier()
dt_tfidf.fit(tfidf,y_train)
dt_tfidf.score(tfidf,y_train)

```

Out[175]:

```
0.99625
```

In [176]:

```python
# with word to vec
dt_word2vec=DecisionTreeClassifier()
dt_word2vec.fit(feature,y_train)
dt_word2vec.score(feature,y_train)

```

Out[176]:

```
0.98875
```

Random Forest

In [177]:

```python
# with count vectorizer
rf_count=RandomForestClassifier()
rf_count.fit(bow,y_train)
rf_count.score(bow,y_train)
```

Out[177]:

```
0.99625
```

In [178]:

```python
#with tfidf
rf_tfidf=RandomForestClassifier()
rf_tfidf.fit(tfidf,y_train)
rf_tfidf.score(tfidf,y_train)
```

Out[178]:

```
0.99625
```

In [179]:

```python
#word2vec
rf_word2vec=RandomForestClassifier()
rf_word2vec.fit(feature,y_train)
rf_word2vec.score(feature,y_train)
```

Out[179]:

```
0.98875
```

Adaboost

In [180]:

```
#count_vectorizer
ada_count=AdaBoostClassifier()
ada_count.fit(bow,y_train)
ada_count.score(bow,y_train)
```

Out[180]:

```
0.68375
```

In [181]:

```
#tfidf
ada_tfidf=AdaBoostClassifier()
ada_tfidf.fit(tfidf,y_train)
ada_tfidf.score(tfidf,y_train)
```

Out[181]:

```
0.8475
```

In [182]:

```
#word2vec
ada_word2vec=AdaBoostClassifier()
ada_word2vec.fit(feature,y_train)
ada_word2vec.score(feature,y_train)
```

Out[182]:

```
0.5825
```

Naive_bayes

In [183]:

```
#count_vect
naive_count=MultinomialNB()
naive_count.fit(bow,y_train)
naive_count.score(bow,y_train)
```

Out[183]:

```
0.935
```

In [184]:

```
naive_tfidf=MultinomialNB()
naive_tfidf.fit(tfidf,y_train)
naive_tfidf.score(tfidf,y_train)
```

Out[184]:

```
0.94875
```