Importing Libraries

```
# import contractions library.
!pip install contractions
```

```
Collecting contractions
  Downloading contractions-0.1.73-py2.py3-none-any.whl (8.7 kB)
Collecting textsearch>=0.0.21 (from contractions)
  Downloading textsearch-0.0.24-py2.py3-none-any.whl (7.6 kB)
Collecting anyascii (from textsearch>=0.0.21->contractions)
  Downloading anyascii-0.3.2-py3-none-any.whl (289 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 289.9/289.9 kB 5.7 MB/s eta 0:00:00
Collecting pyahocorasick (from textsearch>=0.0.21->contractions)
  Downloading pyahocorasick-2.0.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.8/110.8 kB 14.8 MB/s eta 0:00:00
Installing collected packages: pyahocorasick, anyascii, textsearch, contractions
Successfully installed anyascii-0.3.2 contractions-0.1.73 pyahocorasick-2.0.0 textsearch-0.0.24
```

```
import pandas as pd
import numpy as np
# %load_ext nb_black
```

```
# library to suppress warnings or deprecation notes
import warnings

warnings.filterwarnings("ignore")
```

```
# import Regex, string and unicodedata.
import re, string, unicodedata

import contractions
```

```
# import BeautifulSoup.
from bs4 import BeautifulSoup

# import Natural Language Tool-Kit.
import nltk
```

```
# download Stopwords.
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
# import stopwords.
from nltk.corpus import stopwords

# import Tokenizer.
from nltk.tokenize import word_tokenize, sent_tokenize
```

```
# library to split data
from sklearn.model_selection import train_test_split, StratifiedKFold

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
```

```
# import wordcloud
import wordcloud
from wordcloud import STOPWORDS
from wordcloud import WordCloud
```

```
# remove the limit for the number of displayed columns
pd.set_option("display.max_columns", None)

# set the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)


# to get diferent metric scores
from sklearn.metrics import (
    recall_score,
    accuracy_score,
    confusion_matrix,classification_report,
    f1_score,
    precision_score,
    precision_recall_fscore_support
)


# import vectorizers
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer


# import rfc and cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score


# import word prepocessors
from nltk.tokenize import word_tokenize
from nltk.stem import LancasterStemmer, WordNetLemmatizer
```

Loading the dataset

```
df = pd.read_csv('/content/Tweets.csv')
```

```
df.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereas
0	570306133677760513	neutral	1.0000	N
1	570301130888122368	positive	0.3486	N
2	570301083672813571	neutral	0.6837	N
3	570301031407624196	negative	1.0000	Bad Fli
4	570300817074462722	negative	1.0000	Can't

```
texts = [[word.lower() for word in text.split()] for text in df]
```

```
df.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereas
0	570306133677760513	neutral	1.0000	N

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14640 entries, 0 to 14639
Data columns (total 15 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   tweet_id                      14640 non-null  int64  
 1   airline_sentiment             14640 non-null  object 
 2   airline_sentiment_confidence  14640 non-null  float64
 3   negativereason                9178 non-null   object 
 4   negativereason_confidence     10522 non-null  float64
 5   airline                       14640 non-null  object 
 6   airline_sentiment_gold        40 non-null     object 
 7   name                          14640 non-null  object 
 8   negativereason_gold           32 non-null     object 
 9   retweet_count                 14640 non-null  int64  
 10  text                          14640 non-null  object 
 11  tweet_coord                   1019 non-null   object 
 12  tweet_created                 14640 non-null  object 
 13  tweet_location                9907 non-null   object 
 14  user_timezone                 9820 non-null   object 
dtypes: float64(2), int64(2), object(11)
memory usage: 1.7+ MB
```

Observations:

There are 15 columns in the dataset. Half of the columns have null values. Considering both dependent and independent variables not having any null values, we will not do any null value processing. Most columns in the dataset are of object type. airline_sentiment is our dependent / target variable. text column is our independent variable that we will use for analysis. All other columns will be dropped at a later stage.

```
df.isnull().sum()
```

```
tweet_id                            0
airline_sentiment                   0
airline_sentiment_confidence        0
negativereason                   5462
negativereason_confidence        4118
airline                             0
airline_sentiment_gold          14600
name                                0
negativereason_gold             14608
retweet_count                       0
text                                0
tweet_coord                     13621
tweet_created                       0
tweet_location                   4733
user_timezone                    4820
dtype: int64
```

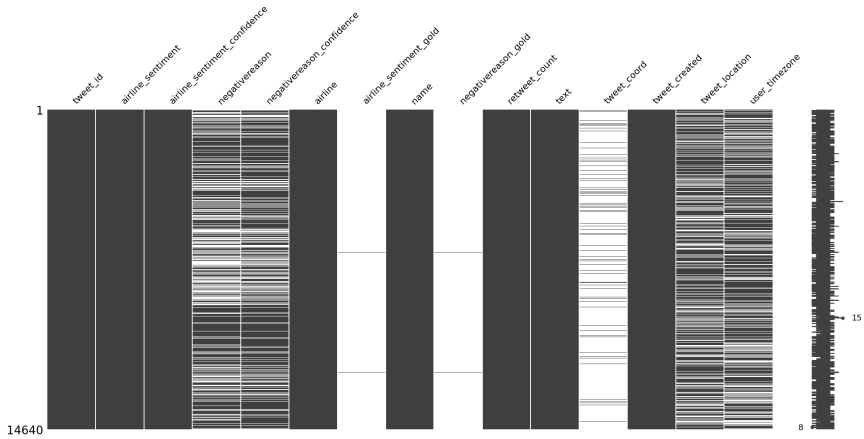
```
df.isnull().sum() / len(df) * 100
```

```
tweet_id                         0.000000
airline_sentiment                0.000000
airline_sentiment_confidence     0.000000
negativereason                  37.308743
negativereason_confidence       28.128415
airline                          0.000000
airline_sentiment_gold          99.726776
name                             0.000000
negativereason_gold             99.781421
retweet_count                    0.000000
text                             0.000000
tweet_coord                     93.039617
tweet_created                    0.000000
tweet_location                  32.329235
user_timezone                   32.923497
dtype: float64
```

Data Visualization

```
msno.matrix(df)
```

```
<Axes: >
```




```
 #Visualization of missing value using heatmap
plt.figure(figsize=(12,7))
sns.heatmap(df.isnull(), cmap = "Blues")
plt.title("Missing values?", fontsize = 15)
plt.show()
```

Interestingly, the only non-null values of the _gold columns seems to be the same entries for the most part. Meanwhile, there is some but not total overlap between location and timezone in terms of missing values.

```
print("Percentage null or na values in df")
((df.isnull() | df.isna()).sum() * 100 / df.index.size).round(2)
```

```
Percentage null or na values in df
tweet_id                         0.00
airline_sentiment                0.00
airline_sentiment_confidence     0.00
negativereason                  37.31
negativereason_confidence       28.13
airline                          0.00
airline_sentiment_gold          99.73
name                             0.00
negativereason_gold             99.78
retweet_count                    0.00
text                             0.00
tweet_coord                     93.04
tweet_created                    0.00
tweet_location                  32.33
user_timezone                   32.92
dtype: float64
```

airline_sentiment_gold, negativereason_gold have more than 99% missing data And tweet_coord have nearly 93% missing data. It will be better to delete these columns as they will not provide any constructive information

```
del df["tweet_coord"]
```

```
del df["airline_sentiment_gold"]
del df["negativereason_gold"]
```

```
df.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereas
0	570306133677760513	neutral	1.0000	N
1	570301130888122368	positive	0.3486	N
2	570301083672813571	neutral	0.6837	N
3	570301031407624196	negative	1.0000	Bad Fli
4	570300817074462722	negative	1.0000	Can't

```
freq = df.groupby("negativereason").size()
```

we can't fill it will affect in bad way for example we have positive reviwe and we fill the values with mode that means with Customer Service Issue it is missmatch and can be affect on train model so we keep the data as it is.

```
# Checking duplicates
df.duplicated().sum()
```

```
39
```

```
# Dropping duplicates
df.drop_duplicates(inplace = True)
```

```
df.duplicated().sum()
```

```
0
```

```
df.sample(n = 10)
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negative
2415	569218838190907393	positive	0.6487	
1069	569937143314849792	negative	1.0000	D L
55	569996412286582784	negative	0.6939	Flight P
293	568840560347373569	positive	1.0000	
983	569976131748823040	negative	1.0000	La
8786	567848171155496960	negative	0.6551	C
6325	568055480226066434	negative	1.0000	C Servic
1223	569878139003740163	negative	1.0000	C Servic
14601	569592830307508224	negative	1.0000	La
7354	569648229295329280	positive	1.0000	

```
df.describe().T
```

	count	mean	std	min	
tweet_id	14601.0	5.692156e+17	7.782706e+14	5.675883e+17	5.68558
airline_sentiment_confidence	14601.0	8.999022e-01	1.629654e-01	3.350000e-01	6.92300
negativereason_confidence	10501.0	6.375749e-01	3.303735e-01	0.000000e+00	3.60500
retweet_count	14601.0	8.280255e-02	7.467231e-01	0.000000e+00	0.00000

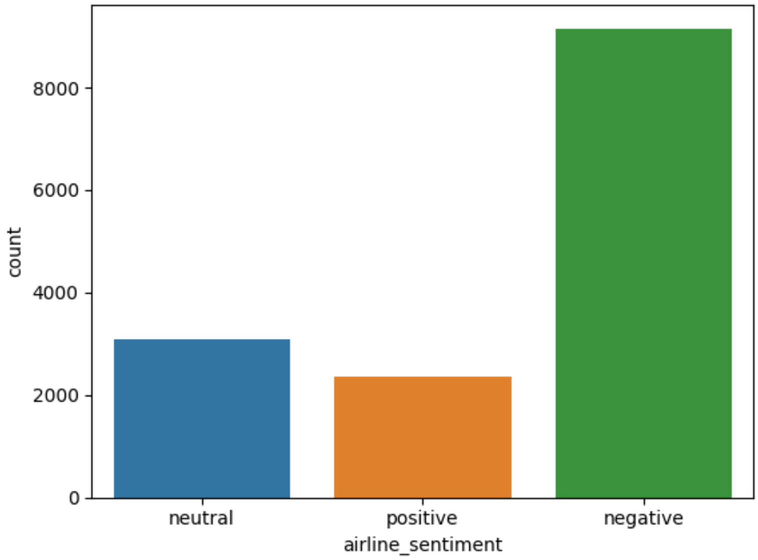
Exploratory Data Analysis(EDA)

```
df.nunique()
```

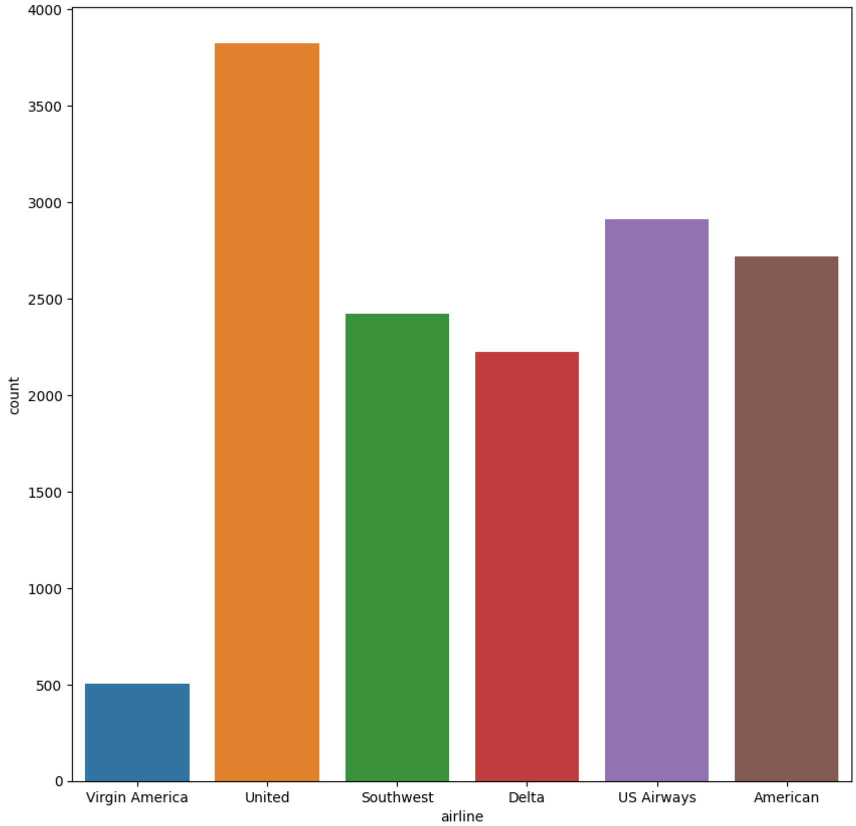
```
tweet_id                        14485
airline_sentiment                   3
airline_sentiment_confidence     1023
negativereason                     10
negativereason_confidence        1410
airline                             6
name                             7701
retweet_count                      18
```

```
text                         14427
tweet_created                14247
tweet_location                3081
user_timezone                   85
dtype: int64
```

```
ax = sns.countplot(x = "airline_sentiment", data = df)
```

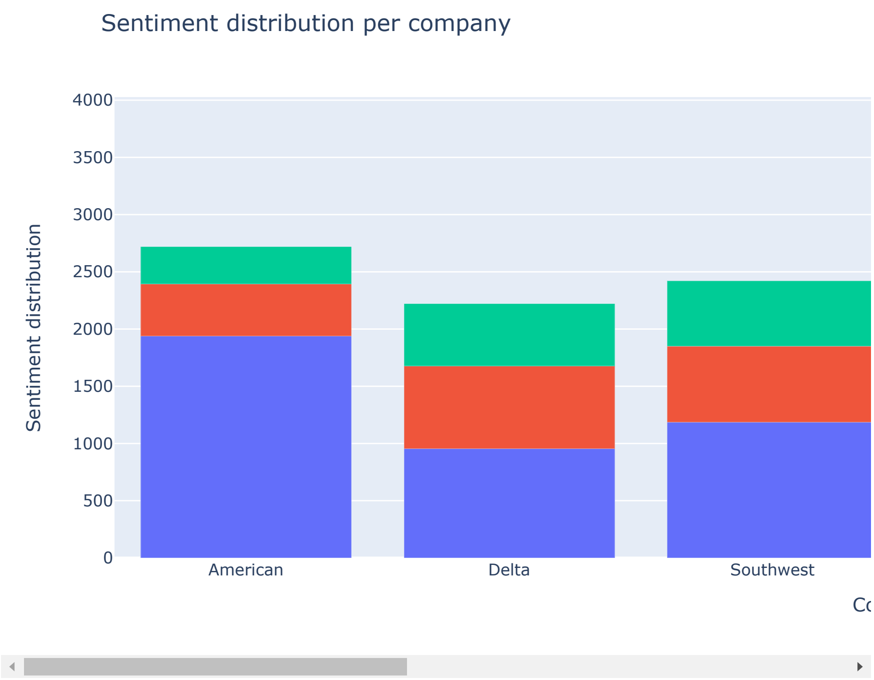



```
plt.figure(figsize = (10, 10))
ax = sns.countplot(x = "airline", data = df)
```



** Stacked bar chart to show the distribution of reviews per company**

```
import plotly.graph_objects as go
crosstab_sentiments=pd.crosstab(df.airline, df.airline_sentiment)
companies=list(crosstab_sentiments.index)

fig = go.Figure(data=[
    go.Bar(name=col_name, x=companies, y=list(crosstab_sentiments[col_name]))
for col_name in list(crosstab_sentiments.columns)])
# Change the bar mode
fig.update_layout(barmode='stack',
                  title='Sentiment distribution per company',
                  yaxis=dict(title='Sentiment distribution'),
                 xaxis=dict(title='Companies'))
fig.show()
```



Stacked bar chart to show negative reasons distributions per company

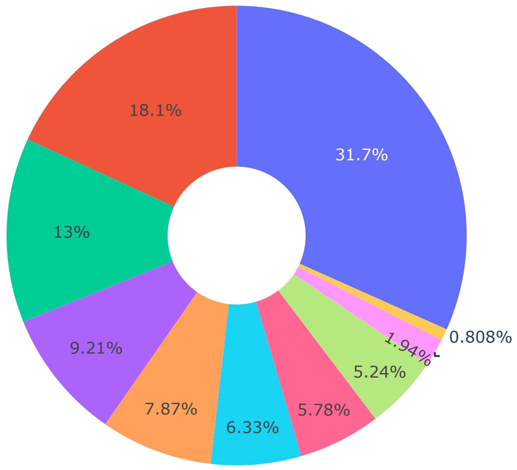
```
crosstab_neg_reasons = pd.crosstab(df["airline"], df["negativereason"])
companies = list(crosstab_neg_reasons.index)

fig = go.Figure(data = [
    go.Bar(name = col_name, x = companies, y = list(crosstab_neg_reasons[col_name]))
for col_name in list(crosstab_neg_reasons.columns)])

fig.update_layout(barmode = "stack",
                 title = "Negative Reasons Distribution per Company",
                 yaxis = dict(title = "Negative reasons Distribution"),
                 xaxis = dict(title = "Companies"))
fig.show()
```

Negative Reasons Distribution per Company

Pie chart to check the overall distribution for negative reasons.

```
labels = list(crosstab_neg_reasons.columns)
values = [crosstab_neg_reasons[col_name].sum() for col_name in labels]

# Use `hole` to create a donut-like pie chart
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.3)])
fig.update_layout(title='Overall distribution for negative reasons')
fig.show()
```

Overall distribution for negative reasons




```
df.drop(df.loc[df["airline_sentiment"] == "neutral"].index, inplace = True)
```

Vectorization Process

```
data = df[
    ["airline_sentiment", "text"]
]
data.head()
```

	airline_sentiment	text
1	positive	@VirginAmerica plus you've added commercials t...
3	negative	@VirginAmerica it's really aggressive to blast...
4	negative	@VirginAmerica and it's a really big bad thing...
5	negative	@VirginAmerica seriously would pay $30 a fligh...
6	positive	@VirginAmerica yes, nearly every time I fly VX...

```
X = df["text"]
y = df["airline_sentiment"]
```

```
X
```

```
1        @VirginAmerica plus you've added commercials t...
3        @VirginAmerica it's really aggressive to blast...
4        @VirginAmerica and it's a really big bad thing...
5        @VirginAmerica seriously would pay $30 a fligh...
6        @VirginAmerica yes, nearly every time I fly VX...
                              ...
```

```
14633    @AmericanAir my flight was Cancelled Flightled...
14634            @AmericanAir right on cue with the delays👌
14635    @AmericanAir thank you we got on a different f...
14636    @AmericanAir leaving over 20 minutes Late Flig...
14638    @AmericanAir you have my money, you change my ...
Name: text, Length: 11510, dtype: object
```

```
y
```

```
1        positive
3        negative
4        negative
5        negative
6        positive
           ...
14633    negative
14634    negative
14635    positive
14636    negative
14638    negative
Name: airline_sentiment, Length: 11510, dtype: object
```

Train Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(9208,) (2302,) (9208,) (2302,)
```

```
tfidf = TfidfVectorizer(stop_words="english")
```

```
tfidf.fit(X_train)
```

```
▾          TfidfVectorizer
TfidfVectorizer(stop_words='english')
```

```
print(tfidf.get_feature_names_out())
```

```
['00' '000' '000ft' ... 'zv6cfpohl5' 'zvfmxnuelj' 'zzps5ywve2']
```

```
print(tfidf.vocabulary_)
```

```
{'jetblue': 5633, 'did': 3478, 'idea': 5246, 'offered': 7007, 'pay': 7303, 'tix': 9682, 'airport': 1441, 'bc': 1996, 'told': 9717,
```

```
data[data["airline_sentiment"] == "negative"]["text"]
```

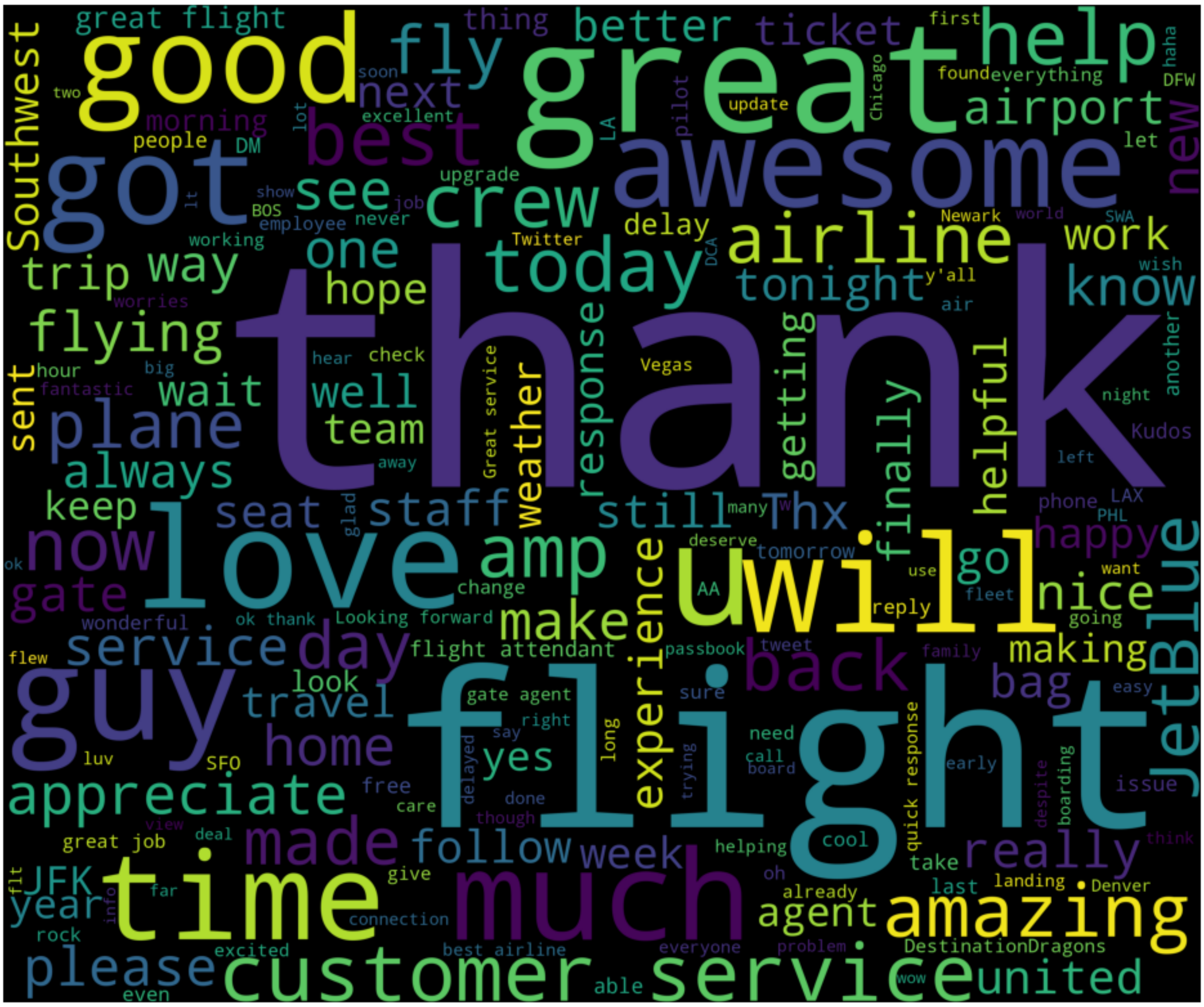
```
3        @VirginAmerica it's really aggressive to blast...
4        @VirginAmerica and it's a really big bad thing...
5        @VirginAmerica seriously would pay $30 a fligh...
15           @VirginAmerica SFO-PDX schedule is still MIA.
17       @VirginAmerica  I flew from NYC to SFO last we...
                               ...
14631    @AmericanAir thx for nothing on getting us out...
14633    @AmericanAir my flight was Cancelled Flightled...
14634            @AmericanAir right on cue with the delays👌
14636    @AmericanAir leaving over 20 minutes Late Flig...
14638    @AmericanAir you have my money, you change my ...
Name: text, Length: 9157, dtype: object
```

```
count_vect = CountVectorizer(stop_words="english")
neg_matrix = count_vect.fit_transform(data[data["airline_sentiment"]=="negative"]["text"])
freqs = zip(count_vect.get_feature_names_out(), neg_matrix.sum(axis=0).tolist()[0])
# Sort from largest to smallest
print(sorted(freqs, key=lambda x: -x[1])[:100])
```

```
[('flight', 2937), ('united', 2899), ('usairways', 2375), ('americanair', 2089), ('southwestair', 1214), ('jetblue', 1051), ('cance
```

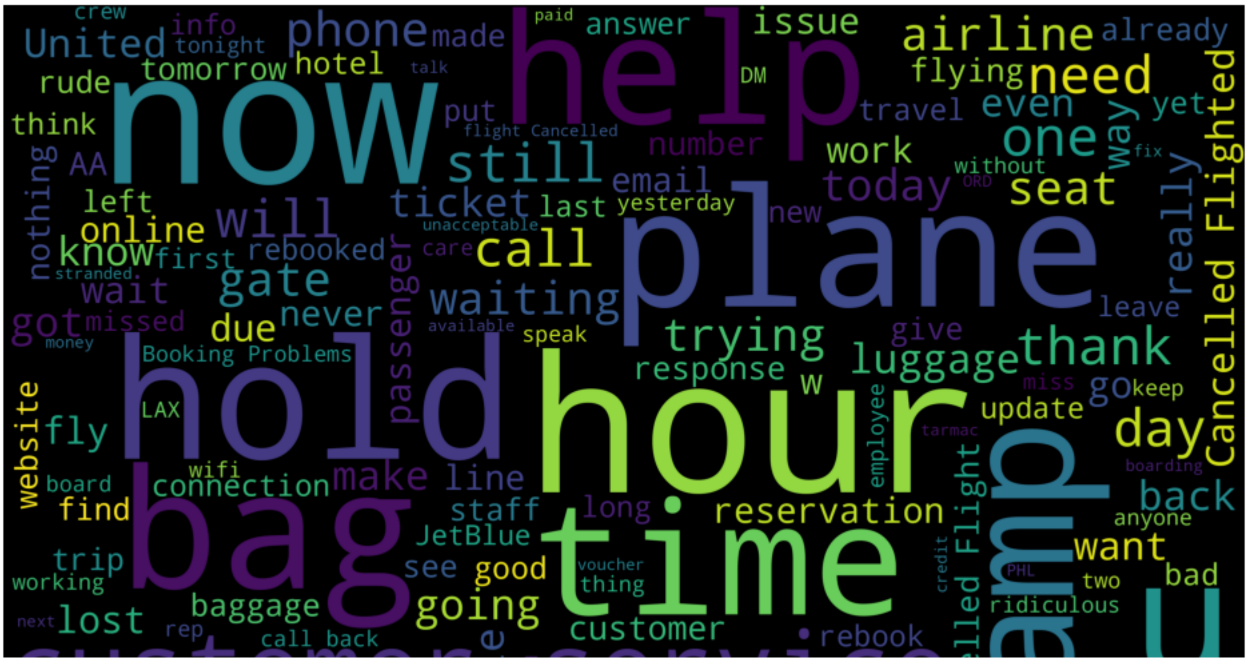
Wordcloud for Positive Reasons

```
new_df = data[data["airline_sentiment"] == "positive"]
words = " ".join(new_df["text"])
cleaned_word = " ".join([word for word in words.split() if "http" not in word and not word.startswith("@") and word != "RT"])
wordcloud = WordCloud(stopwords = STOPWORDS,
                      background_color = "black", width = 3000, height = 2500).generate(cleaned_word)
plt.figure(figsize = (12, 12))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```



Wordcloud for Negative Reasons

```
new_df = data[data["airline_sentiment"] == "negative"]
words = " ".join(new_df["text"])
cleaned_word = " ".join([word for word in words.split() if "http" not in word and not word.startswith("@") and word != "RT"])
wordcloud = WordCloud(stopwords = STOPWORDS,
                      background_color = "black", width = 3000, height = 2500).generate(cleaned_word)
plt.figure(figsize = (12, 12))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```




```
data.drop(data.loc[data["airline_sentiment"] == "neutral"].index, inplace = True)
```

Data Scaling

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

le.fit(data["airline_sentiment"])
data["airline_sentiment_Encoded"] = le.transform(data["airline_sentiment"])
data.head()
```

	airline_sentiment	text	airline_sentiment_Encoded
1	positive	@VirginAmerica plus you've added commercials t...	1
3	negative	@VirginAmerica it's really aggressive to blast...	0
4	negative	@VirginAmerica and it's a really big bad thing...	0
5	negative	@VirginAmerica seriously would pay $30 a fligh...	0
6	positive	@VirginAmerica yes, nearly every time I fly VX...	1

```
def tweet_to_words(tweet):
    letters_only = re.sub("[^a-zA-Z]", " ", tweet)
    words = letters_only.lower().split()
    stops = set(stopwords.words("english"))
    meaningful_words = [w for w in words if not w in stops]
    return(" ".join( meaningful_words ))


nltk.download("stopwords")
data["clean_tweet"] = data["text"].apply(lambda x: tweet_to_words(x))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11510 entries, 1 to 14638
Data columns (total 4 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   airline_sentiment          11510 non-null  object
 1   text                       11510 non-null  object
 2   airline_sentiment_Encoded  11510 non-null  int64 
 3   clean_tweet                11510 non-null  object
dtypes: int64(1), object(3)
memory usage: 449.6+ KB
```

Defining X and y

```
X = data["clean_tweet"]
y = data["airline_sentiment"]
```

```
print(X.shape, y.shape)
```

```
(11510,) (11510,)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(8632,) (2878,) (8632,) (2878,)
```

```
vect = CountVectorizer()
vect.fit(X_train)
```

```
▾ CountVectorizer
CountVectorizer()
```

```
X_train_dtm = vect.transform(X_train)
X_test_dtm = vect.transform(X_test)
```

Tuning

```
vect_tunned = CountVectorizer(stop_words = "english", ngram_range = (1, 2), min_df = 0.1, max_df = 0.7, max_features = 100)
vect_tunned
```

```
▾                                CountVectorizer
CountVectorizer(max_df=0.7, max_features=100, min_df=0.1, ngram_range=(1, 2),
                stop_words='english')
```

Model Building

```
from sklearn.svm import SVC
model = SVC(kernel = "linear", random_state = 10)
model.fit(X_train_dtm, y_train)
pred = model.predict(X_test_dtm)
```

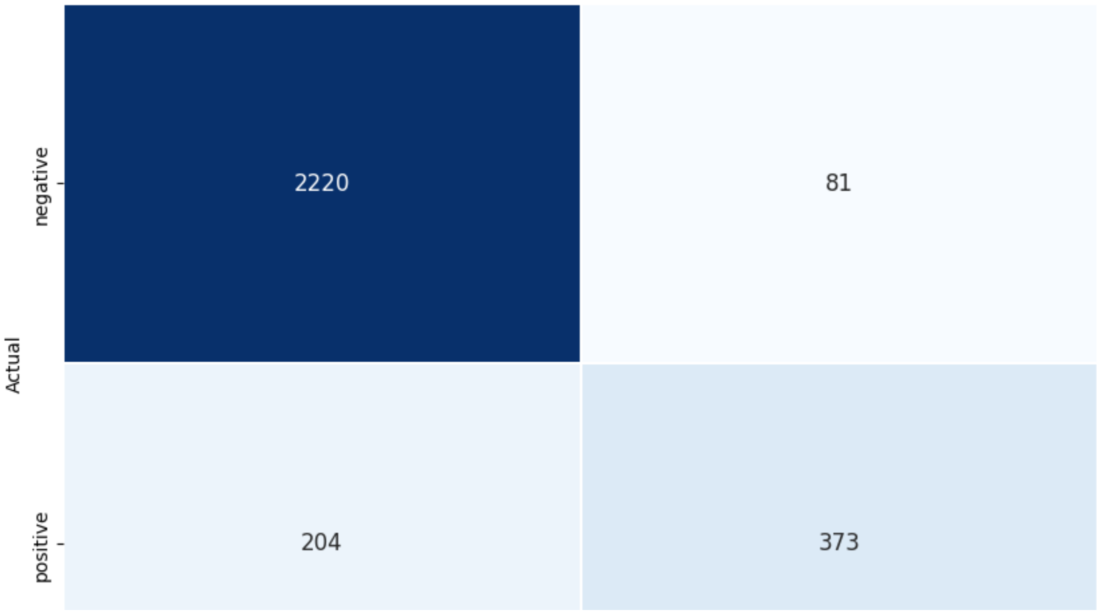
```
print("Accuracy Score: ", accuracy_score(y_test, pred) * 100)
```

```
Accuracy Score:  90.7574704656011
```

```
print("Confusion Matrix\n\n", confusion_matrix(y_test, pred))
```

```
Confusion Matrix

 [[2179  122]
 [ 144  433]]
```

```
# Visualizing the confusion matrix using a heatmap
conf_matrix_df = pd.DataFrame(data=conf_matrix, columns=le.classes_, index=le.classes_)
plt.rcParams['figure.figsize'] = [10, 7]
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues', cbar=False, linewidths=0.1, annot_kws={'size': 12})
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

    negative       0.94      0.95      0.94      2301
    positive       0.78      0.75      0.77       577

    accuracy                           0.91      2878
   macro avg       0.86      0.85      0.85      2878
weighted avg       0.91      0.91      0.91      2878
```

Conclusions

As we you can see above we have plotted the confusion matrix for predicted sentiments and actual sentiments (negative and positive)

SVM Classifier gives us the best accuracy score i.e 91% precision scores according to the classification report

The confusion matrix shows the TP,TN,FP,FN for sentiments(negative, positive).

Decision Tree Classifier

```
 from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import re
from nltk.corpus import stopwords
from sklearn.preprocessing import LabelEncoder


# Splitting the data into features (X) and target variable (y)
X = data["clean_tweet"]
y = data["airline_sentiment_Encoded"]


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


# Vectorizing the text data
vect = CountVectorizer()
X_train_dtm = vect.fit_transform(X_train)
X_test_dtm = vect.transform(X_test)


# Initializing and training the Decision Tree model
dt_model = DecisionTreeClassifier(random_state=10)
dt_model.fit(X_train_dtm, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=10)
```

```
# Predicting on the test set
pred = dt_model.predict(X_test_dtm)
```

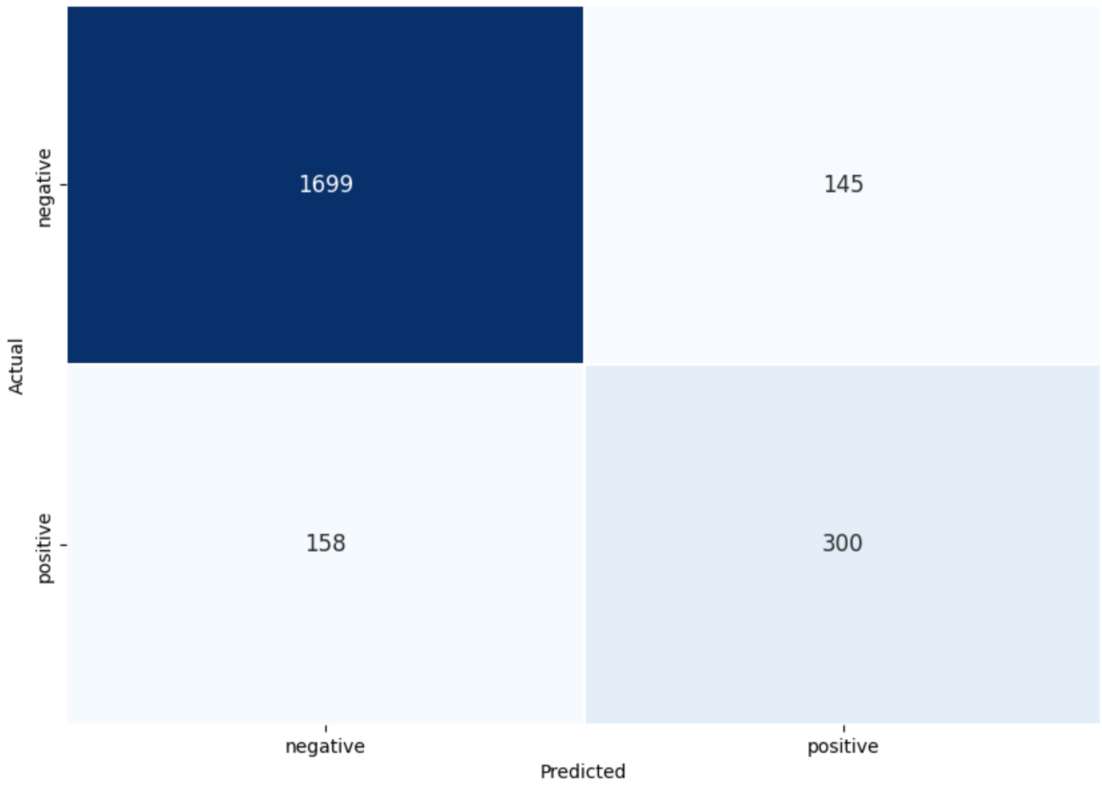
```
# Evaluating the model
accuracy = accuracy_score(y_test, pred)
print("Accuracy Score: {:.2f}%".format(accuracy * 100))
```

```
Accuracy Score: 86.84%
```

```
# Displaying the confusion matrix
conf_matrix = confusion_matrix(y_test, pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Confusion Matrix:
 [[1699  145]
 [ 158  300]]
```

```
# Visualizing the confusion matrix using a heatmap
conf_matrix_df = pd.DataFrame(data=conf_matrix, columns=le.classes_, index=le.classes_)
plt.rcParams['figure.figsize'] = [10, 7]
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues', cbar=False, linewidths=0.1, annot_kws={'size': 12})
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
# Displaying the classification report
report = classification_report(y_test, pred, target_names=le.classes_)
print("Classification Report:\n", report)
```

```
Classification Report:
               precision    recall  f1-score   support

    negative       0.91      0.92      0.92      1844
    positive       0.67      0.66      0.66       458

    accuracy                           0.87      2302
   macro avg       0.79      0.79      0.79      2302
weighted avg       0.87      0.87      0.87      2302
```

Conclusion:

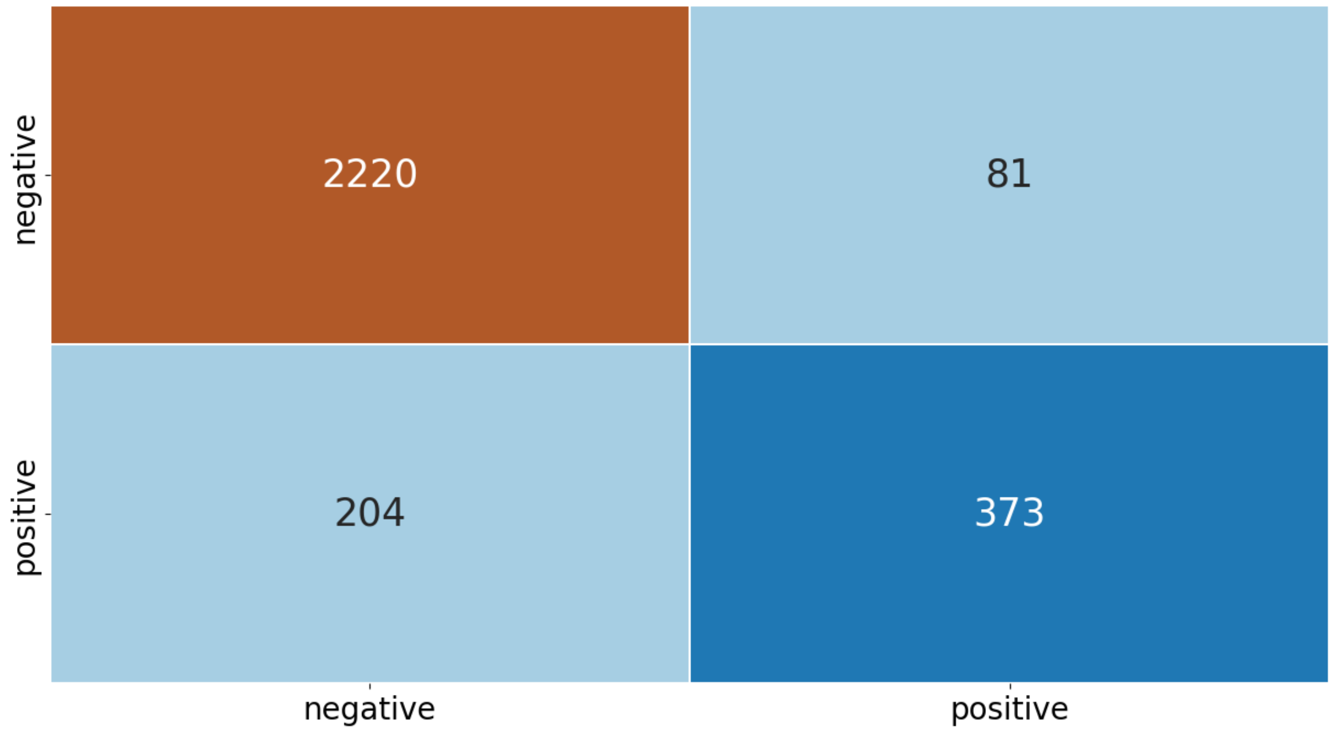
The Decision Tree Classifier exhibits a reasonable overall performance with an accuracy of 87%.** This suggests that the model effectively predicts the correct class for a majority of instances based on the given features.**

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd


# Initialize Random Forest model
model = RandomForestClassifier(random_state=10)
model.fit(X_train_dtm, y_train)
pred = model.predict(X_test_dtm)


# Calculate and print accuracy
accuracy = accuracy_score(y_test, pred)
print("Accuracy Score: {:.2f}%".format(accuracy * 100))


# Display confusion matrix
conf_matrix = confusion_matrix(y_test, pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Confusion Matrix:
 [[2220   81]
 [ 204  373]]
```

```
# Visualize confusion matrix using a heatmap
conf_matrix_df = pd.DataFrame(data=conf_matrix, columns=model.classes_, index=model.classes_)
plt.rcParams['figure.figsize'] = [15, 8]
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Paired', cbar=False, linewidths=0.1, annot_kws={'size': 25})
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```




```
# Display classification report
report = classification_report(y_test, pred)
print("Classification Report:\n", report)
```

```
Classification Report:
              precision    recall  f1-score   support

    negative       0.92      0.96      0.94      2301
    positive       0.82      0.65      0.72       577

    accuracy                           0.90      2878
   macro avg       0.87      0.81      0.83      2878
weighted avg       0.90      0.90      0.90      2878
```

Conclusion:

The Random Forest Classifier demonstrates strong overall accuracy, achieving a score of 90%. This indicates the model's ability to correctly classify instances into their respective classes based on the given features.

While the model performs well in identifying the "negative" class with high precision and recall (0.92 and 0.96, respectively), it faces challenges in accurately classifying the "positive" class, showing lower precision (0.82) and recall (0.65). This suggests potential areas for improvement, particularly in capturing positive instances.

K-Nearest-Neighbors(KNN)

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import re
from nltk.corpus import stopwords
from sklearn.preprocessing import LabelEncoder
```

```
# Splitting the data into features (X) and target variable (y)
X = data["clean_tweet"]
y = data["airline_sentiment_Encoded"]
```

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

```
# Vectorizing the text data
vect = CountVectorizer()
X_train_dtm = vect.fit_transform(X_train)
X_test_dtm = vect.transform(X_test)
```

```
# Initializing and training the KNN model
knn_model = KNeighborsClassifier()
knn_model.fit(X_train_dtm, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
# Predicting on the test set
pred = knn_model.predict(X_test_dtm)
```

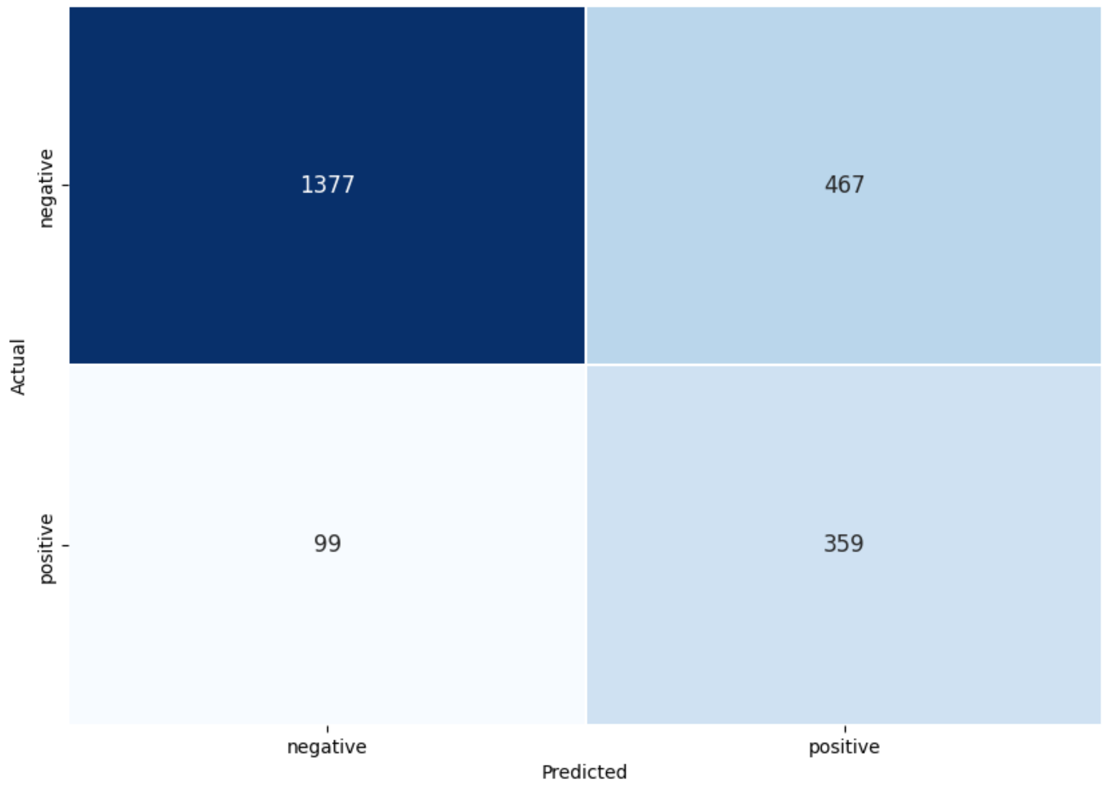
```
# Evaluating the model
accuracy = accuracy_score(y_test, pred)
print("Accuracy Score: {:.2f}%".format(accuracy * 100))
```

```
Accuracy Score: 75.41%
```

```
# Displaying the confusion matrix
conf_matrix = confusion_matrix(y_test, pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Confusion Matrix:
 [[1377  467]
 [  99  359]]
```

```
# Visualizing the confusion matrix using a heatmap
conf_matrix_df = pd.DataFrame(data=conf_matrix, columns=le.classes_, index=le.classes_)
plt.rcParams['figure.figsize'] = [10, 7]
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues', cbar=False, linewidths=0.1, annot_kws={'size': 12})
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
# Displaying the classification report
report = classification_report(y_test, pred, target_names=le.classes_)
print("Classification Report:\n", report)
```

```
 Classification Report:
               precision    recall  f1-score   support

    negative       0.93      0.75      0.83      1844
    positive       0.43      0.78      0.56       458

    accuracy                           0.75      2302
   macro avg       0.68      0.77      0.69      2302
weighted avg       0.83      0.75      0.78      2302
```

Naive Bayes Classifier

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import re
from nltk.corpus import stopwords
from sklearn.preprocessing import LabelEncoder
```

```
# Splitting the data into features (X) and target variable (y)
X = data["clean_tweet"]
y = data["airline_sentiment_Encoded"]
```

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

```
# Vectorizing the text data
vect = CountVectorizer()
X_train_dtm = vect.fit_transform(X_train)
X_test_dtm = vect.transform(X_test)
```

```
# Initializing and training the Naive Bayes model
nb_model = MultinomialNB()
nb_model.fit(X_train_dtm, y_train)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
# Predicting on the test set
pred_nb = nb_model.predict(X_test_dtm)
```

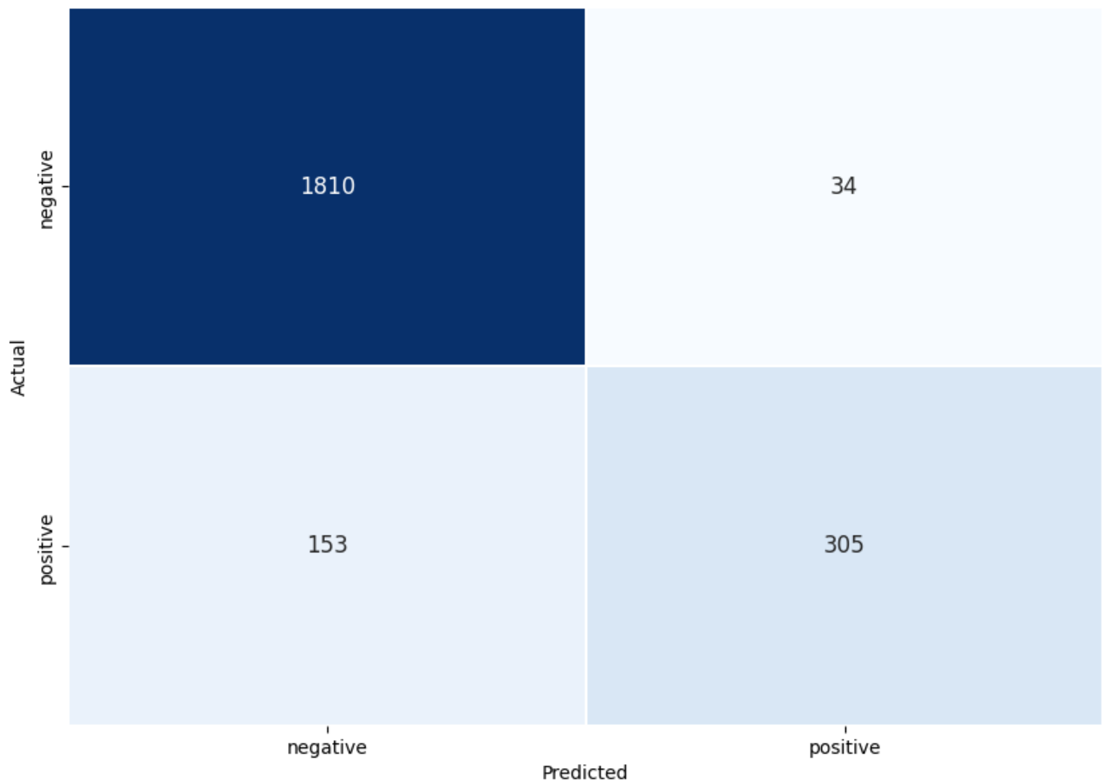
```
# Evaluating the model
accuracy_nb = accuracy_score(y_test, pred_nb)
print("Naive Bayes - Accuracy Score: {:.2f}%".format(accuracy_nb * 100))
```

```
Naive Bayes - Accuracy Score: 91.88%
```

```
# Displaying the confusion matrix
conf_matrix_nb = confusion_matrix(y_test, pred_nb)
print("Naive Bayes - Confusion Matrix:\n", conf_matrix_nb)
```

```
Naive Bayes - Confusion Matrix:
 [[1810   34]
 [ 153  305]]
```

```
# Visualizing the confusion matrix using a heatmap
conf_matrix_df_nb = pd.DataFrame(data=conf_matrix_nb, columns=le.classes_, index=le.classes_)
plt.rcParams['figure.figsize'] = [10, 7]
sns.heatmap(conf_matrix_df_nb, annot=True, fmt='d', cmap='Blues', cbar=False, linewidths=0.1, annot_kws={'size': 12})
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
# Displaying the classification report
report_nb = classification_report(y_test, pred_nb, target_names=le.classes_)
print("Naive Bayes - Classification Report:\n", report_nb)
```

```
Naive Bayes - Classification Report:
               precision    recall  f1-score   support

    negative       0.92      0.98      0.95      1844
```

```
    positive       0.90      0.67      0.77       458

    accuracy                           0.92      2302
   macro avg       0.91      0.82      0.86      2302
weighted avg       0.92      0.92      0.91      2302
```

Conclusion:

The **Naive Bayes** Classifier demonstrates strong overall accuracy as compared to other ML models, achieving an accuracy score of **92%**. This indicates the model's ability to correctly classify instances into their respective classes based on the given features.

The classifier performs exceptionally well in identifying the "negative" class, with high precision (0.92) and recall (0.98). However, there is room for improvement in classifying the "positive" class, as reflected by a slightly lower precision (0.90) and recall (0.67). This suggests potential areas for fine-tuning, particularly in capturing positive instances.

The macro and weighted average F1-scores are both 0.86, indicating a balanced performance in terms of precision and recall across classes. The model provides a robust trade-off between precision and recall, offering a comprehensive evaluation of its effectiveness.