**Import Required Libraries**

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns

#Text Preprocessing libraries
import string
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer

#ML Model Building Libraries
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

#Model Evaluation
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

# Task 2 - Loading Data set, EDA and initial data visualization

```
df=pd.read_csv('/content/drive/MyDrive/Master/NLP/Project2/updated/Corona_NLP_train.csv',encoding='latin-1')
df.head()
```

| | UserName | ScreenName | Location | TweetAt | OriginalTweet | Sentiment |
|---|---|---|---|---|---|---|
| **0** | 3799 | 48751 | London | 16-03-2020 | @MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i... | Neutral |
| **1** | 3800 | 48752 | UK | 16-03-2020 | advice Talk to your neighbours family to excha... | Positive |
| **2** | 3801 | 48753 | Vagabonds | 16-03-2020 | Coronavirus Australia: Woolworths to give elde... | Positive |
| **3** | 3802 | 48754 | NaN | 16-03-2020 | My food stock is not the only one which is emp... | Positive |
| **4** | 3803 | 48755 | NaN | 16-03-2020 | Me, ready to go at supermarket during the #COV... | Extremely Negative |

**Data Set Shape**

```
df.shape
```

```
(41157, 6)
```

**EDA**

```
def basic_eda(df, row_limit=5, list_elements_limit=10):
    ### rows and columns
    print('Info : There are {} columns in the dataset'.format(df.shape[1]))
    print('Info : There are {} rows in the dataset'.format(df.shape[0]))

    print("==================================================")

    ## data types
    print("\nData type information of different columns")
    dtypes_df = pd.DataFrame(df.dtypes).reset_index().rename(columns={0:'dtype', 'index':'column_name'})
    cat_df = dtypes_df[dtypes_df['dtype']=='object']
    num_df = dtypes_df[dtypes_df['dtype']!='object']
    print('Info : There are {} categorical columns'.format(len(cat_df)))
    print('Info : There are {} numerical columns'.format(len(dtypes_df)-len(cat_df)))

    if list_elements_limit >= len(cat_df):
        print("Categorical columns : ", list(cat_df['column_name']))
    else:
        print("Categorical columns : ", list(cat_df['column_name'])[:list_elements_limit])

    if list_elements_limit >= len(num_df):
        print("Numerical columns : ", list(num_df['column_name']))
    else:
        print("Numerical columns : ", list(num_df['column_name'])[:list_elements_limit])

    #dtypes_df['dtype'].value_counts().plot.bar()
    display(dtypes_df.head(row_limit))

    print("==================================================")
    print("\nDescription of numerical variables")

    #### Describibg numerical columns
    desc_df_num = df[list(num_df['column_name'])].describe().T.reset_index().rename(columns={'index':'column_name'})
    display(desc_df_num.head(row_limit))

    print("==================================================")
    print("\nDescription of categorical variables")

    desc_df_cat = df[list(cat_df['column_name'])].describe().T.reset_index().rename(columns={'index':'column_name'})
    display(desc_df_cat.head(row_limit))

    return


basic_eda(df)
```

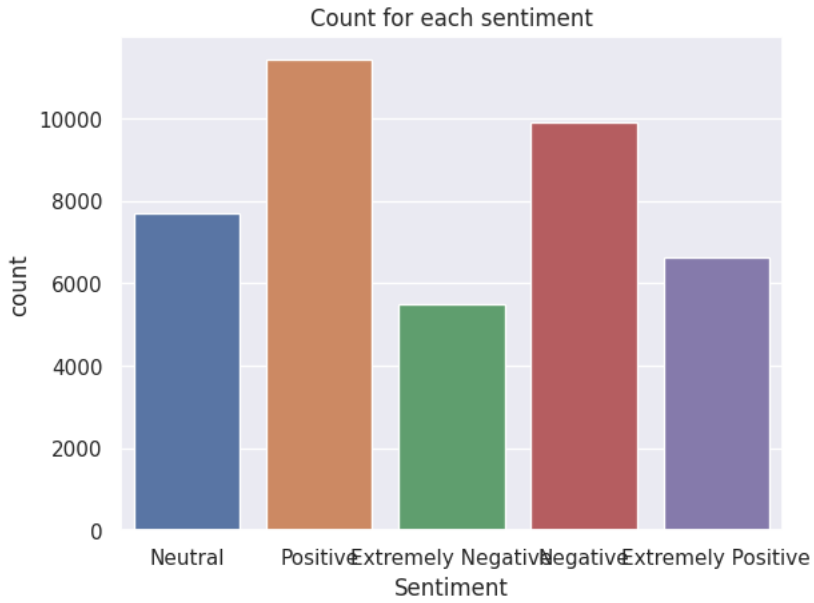
```
Info : There are 6 columns in the dataset
Info : There are 41157 rows in the dataset
==================================================

Data type information of different columns
Info : There are 4 categorical columns
Info : There are 2 numerical columns
```

**Initial Data Visualization**

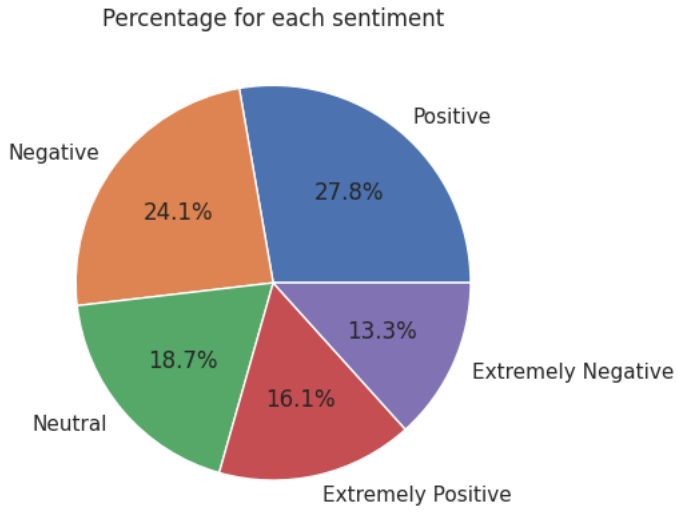
```
df['Sentiment'].value_counts()
```

```
Positive              11422
Negative               9917
Neutral                7713
Extremely Positive     6624
Extremely Negative     5481
Name: Sentiment, dtype: int64
```

```
sns.countplot(x=df['Sentiment'])
plt.title("Count for each sentiment")
plt.show()
```




```
sns.set(style="darkgrid")
sentiment_counts = df['Sentiment'].value_counts(normalize=True) * 100
plt.pie(sentiment_counts, labels = sentiment_counts.index, autopct='%1.1f%%')
plt.title("Percentage for each sentiment")
plt.show()
```



# Task 3 - Text Preprocessing

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41157 entries, 0 to 41156
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   UserName       41157 non-null  int64 
 1   ScreenName     41157 non-null  int64 
 2   Location       32567 non-null  object
 3   TweetAt        41157 non-null  object
 4   OriginalTweet  41157 non-null  object
 5   Sentiment      41157 non-null  object
dtypes: int64(2), object(4)
memory usage: 1.9+ MB
```

**As I have to do only sentiment analysis, so I only need two columns. I will drop rest of the columns.**

```
df.drop(['UserName', 'ScreenName', 'Location', 'TweetAt'], axis=1, inplace=True)
```

```
#Check for nulls
df.isnull().sum()
```

```
OriginalTweet    0
Sentiment        0
dtype: int64
```

```
#Check for duplicates
df.duplicated().sum()
```

```
0
```

```
df.head()
```

| | OriginalTweet | Sentiment |
|---|---|---|
| 0 | @MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i... | Neutral |
| 1 | advice Talk to your neighbours family to excha... | Positive |
| 2 | Coronavirus Australia: Woolworths to give elde... | Positive |
| 3 | My food stock is not the only one which is emp... | Positive |
| 4 | Me, ready to go at supermarket during the #COV... | Extremely Negative |

**Now I will classify the sentiments in only 2 categories.**

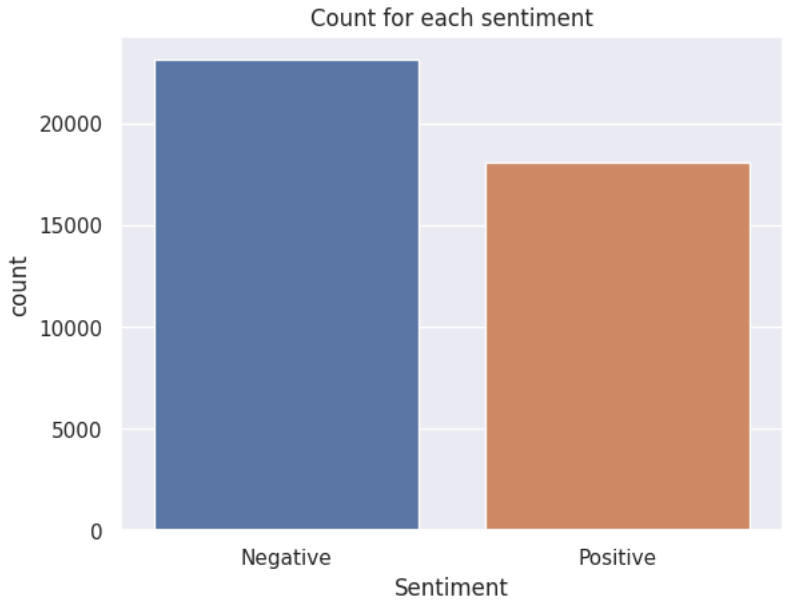
```
def categorize_sentiment(score):
    if score == 'Negative':
        return "Negative"
    elif score == 'Extremely Negative':
        return "Negative"
    elif score == 'Positive':
        return "Positive"
    elif score == 'Extremely Positive':
        return "Positive"
    else:
        return "Negative"
```

```
df['Sentiment'] = df['Sentiment'].apply(categorize_sentiment)
df.head()
```

| | OriginalTweet | Sentiment |
|---|---|---|
| 0 | @MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i... | Negative |
| 1 | advice Talk to your neighbours family to excha... | Positive |

**The 2 categories Visualization**

```
sns.countplot(x=df['Sentiment'])
plt.title("Count for each sentiment")
plt.show()
```




**Text Preprocessing**

```
#Let's make a function to preprocess the text
import nltk
nltk.download('stopwords')
stemmer = PorterStemmer()
stop_words = stopwords.words('english')

def clean_text(text):
    text = text.lower().split()

    #Remove punctuations
    text = [word.translate(str.maketrans('', '', string.punctuation)) for word in text]

    #Remove Stopwords
    text = [word for word in text if word not in stop_words]

    #Stemming
    text = [stemmer.stem(word) for word in text]

    #Joining the text
    joined = ' '.join(text)
    return joined
```

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

```
df = df[:1000]
df['text'] = df['OriginalTweet'].apply(clean_text)
df.head()
```

| | OriginalTweet | Sentiment | text |
|---|---|---|---|
| 0 | @MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i... | Negative | menyrbi philgahan chrisitv httpstcoifz9fan2pa ... |
| 1 | advice Talk to your neighbours family to excha... | Positive | advic talk neighbour famili exchang phone numb... |
| 3 | My food stock is not the only one which is emp... | Positive | food stock one empti pleas dont panic enough f... |

**Word Cloud**

```
pip install wordcloud
```

```
Requirement already satisfied: wordcloud in /usr/local/lib/python3.10/dist-packages (1.9.2)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.10/dist-packages (from wordcloud) (1.23.5)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from wordcloud) (9.4.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from wordcloud) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (4.45.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (23.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->wordcloud) (1
```

```
all_words = ' '.join(word for word in df['text'])


wordcloud = WordCloud(
    width = 1000,
    height = 600,
    background_color = 'skyblue',
    min_font_size = 10).generate(all_words)
plt.figure(figsize = (10, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad = 0)
plt.show()
```


## Task 4 – Text Representation

**POS Tagging**

```
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

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

```
from nltk.tokenize import word_tokenize
from nltk import pos_tag

# convert text into word_tokens with their tags
def pos_tagging(text):
    word_tokens = word_tokenize(text)
    return pos_tag(word_tokens)

document= " ".join(df["text"])
pos_tagging(document)
```

```
    ('worker', 'NN'),
    ...]
```

**Cosine Similarity**

```
import numpy as np

def cosine_similarity(x, y):

    # Ensure length of x and y are the same
    if len(x) != len(y) :
        return None
    else:
    # Compute the dot product between x and y
      dot_product = np.dot(x, y)

    # Compute the L2 norms (magnitudes) of x and y
      magnitude_x = np.sqrt(np.sum(x**2))
      magnitude_y = np.sqrt(np.sum(y**2))

    # Compute the cosine similarity
    cosine_similarity = dot_product / (magnitude_x * magnitude_y)

    return cosine_similarity


from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df["text"])



from sklearn.metrics.pairwise import cosine_similarity

cos_sim_1_2 = cosine_similarity(X.getrow(0), X.getrow(1))
cos_sim_1_3 = cosine_similarity(X.getrow(0), X.getrow(2))
cos_sim_2_3 = cosine_similarity(X.getrow(1), X.getrow(2))

print('Cosine Similarity between:')
print('\tDocument 1 and Document 2:', cos_sim_1_2)
print('\tDocument 1 and Document 3:', cos_sim_1_3)
print('\tDocument 2 and Document 3:', cos_sim_2_3)
```

```
Cosine Similarity between:
        Document 1 and Document 2: [[0.]]
        Document 1 and Document 3: [[0.]]
        Document 2 and Document 3: [[0.04828045]]
```

**Word2vec**

```
from gensim.models import Word2Vec
import nltk
nltk.download('punkt')
```

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

```
from tqdm import tqdm
import gensim
from gensim.models import Word2Vec
from sklearn.metrics.pairwise import cosine_similarity


import spacy
nlp = spacy.load("en_core_web_sm")


sentences = [[tok.text for tok in nlp(row)] for row in tqdm(df["text"])]
```

```
100%|██████████| 1000/1000 [00:16<00:00, 59.78it/s]
```

```
model = gensim.models.Word2Vec(sentences, min_count = 1, vector_size = 100, window = 5)
```

```
model.wv.similarity('virus', 'corona')
```

```
0.25841334
```

```
model.wv.most_similar('virus', topn=10)
```

```
[('100000', 0.39127880334854126),
 ('pickup', 0.3772350549697876),
 ('return', 0.37485602498054504),
 ('raw', 0.37481024861335754),
 ('children', 0.3738054633140564),
 ('energi', 0.3724808692932129),
 ('canâ\x92t', 0.36885061860084534),
 ('buyer', 0.368661105632782),
 ('hungri', 0.3645276725292206),
 ('soar', 0.35844725370407104)]
```

```
model.wv.most_similar('lockdown', topn=10)
```

```
[('home', 0.921994149684906),
 ('us', 0.917965829372406),
 ('suppli', 0.9173396825790405),
 ('get', 0.915720522403717),
 ('tell', 0.9155095815658569),
 ('amp', 0.9149031639099121),
 ('peopl', 0.9148663282394409),
 ('stock', 0.9148194789886475),
 ('supermarket', 0.9144884943962097),
 ('price', 0.913968563079834)]
```

```
a = model.wv['lockdown'].reshape(1,-1)
b = model.wv['fear'].reshape(1,-1)
```

```
a
```

```
array([[-0.01334648,  0.01293831,  0.01497127,  0.00479792,  0.0085624 ,
        -0.02976038,  0.00530198,  0.03384963, -0.00201643, -0.00332835,
        -0.01419284, -0.03356962, -0.00958356,  0.00912999,  0.00784219,
        -0.0147979 , -0.00541215, -0.01750105, -0.00293127, -0.02348854,
         0.01171135,  0.00425653,  0.0162493 , -0.00856071, -0.0134266 ,
        -0.00714657, -0.01684705,  0.00076632, -0.01942884,  0.0087498 ,
         0.0207898 ,  0.00247428,  0.00164646, -0.01411153, -0.01632436,
         0.01414434,  0.00540644, -0.00974451, -0.01667973, -0.03616789,
         0.01110709, -0.01463543, -0.00960748, -0.00058477,  0.01305821,
        -0.00487604, -0.00847906, -0.0093684 ,  0.01444342,  0.01420276,
         0.00063789, -0.0218746 , -0.0098608 , -0.00973442, -0.01100855,
         0.00363076,  0.01971969,  0.0066254 , -0.02216922,  0.00255615,
         0.00205688, -0.00200952,  0.00648989,  0.00621015, -0.01739972,
         0.01630204, -0.0005941 ,  0.01915076, -0.02427489,  0.01905368,
        -0.01801357,  0.00972119,  0.01124174, -0.00581505,  0.01207064,
         0.00621461,  0.00649219, -0.00381485, -0.01236975, -0.00174134,
        -0.01493039, -0.01295233, -0.01213723,  0.0253871 ,  0.00286227,
         0.00322055, -0.00661948,  0.01530799,  0.02170568,  0.00604184,
         0.02473039,  0.00506919, -0.00467051, -0.00255888,  0.01936262,
         0.01015805,  0.01757591, -0.00929166,  0.00081607,  0.01113569]],
      dtype=float32)
```

```
cosine_similarity(a, b)
```

```
array([[0.8838581]], dtype=float32)
```

**NGram- Unigram**

```
from collections import defaultdict
```

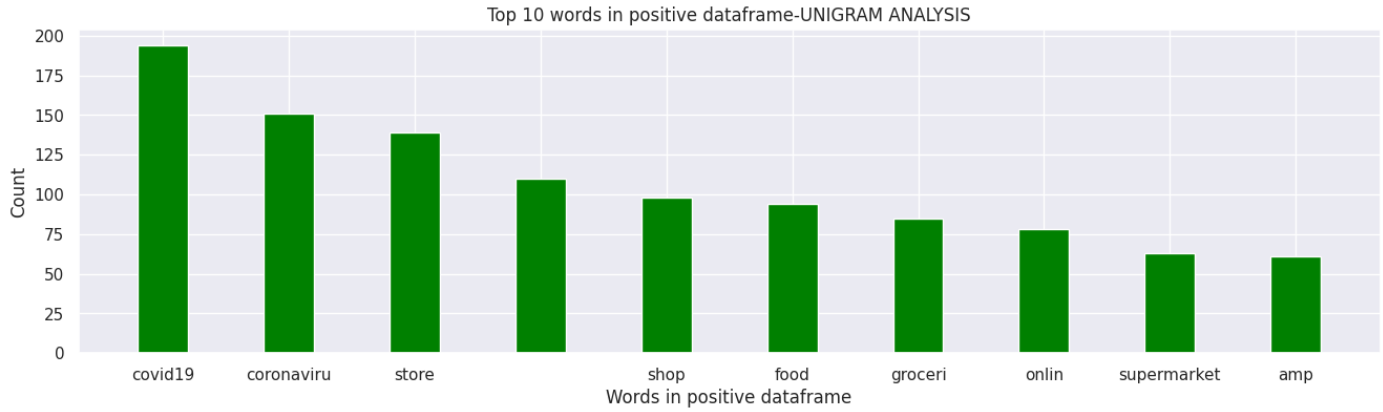
```
#method to generate n-grams:
#params:
#text-the text for which we have to generate n-grams
#ngram-number of grams to be generated from the text(1,2,3,4 etc., default value=1)


def generate_N_grams(text,ngram=1):
  words=[word for word in text.split(" ") if word not in set(stopwords.words('english'))]
  print("Sentence after removing stopwords:",words)
  temp=zip(*[words[i:] for i in range(0,ngram)])
  ans=[' '.join(ngram) for ngram in temp]
  return ans


positiveValues=defaultdict(int)
negativeValues=defaultdict(int)


#get the count of every word in the dataframe where sentiment="positive"
for textt in df[df.Sentiment=="Positive"].text:
  for word in generate_N_grams(textt):
    positiveValues[word]+=1

#get the count of every word in the dataframe where sentiment="negative"
for textt in df[df.Sentiment=="Negative"].text:
  for word in generate_N_grams(textt):
    negativeValues[word]+=1
```

```
Sentence after removing stopwords: ['know', 'youa(x3z', 'live', 'apocalyps', 'peopl', 'fight', 'cashier', '100', 'roll', 'supermarke
Sentence after removing stopwords: ['everyon', 'keep', 'eye', 'account', 'get', 'debit', 'card', 'shut', 'fraudul', 'charg', 'food',
Sentence after removing stopwords: ['yesterday', 'ask', 'pharmaci', 'constitu', 'hike', 'price', 'essenti', 'good', 'appal', 'learn',
Sentence after removing stopwords: ['connect', 'essenti', 'time', 'crisi', 'cwaunion', 'amp', 'alli', 'ask', 'broadband', 'ceo', 'lif
Sentence after removing stopwords: ['winatlifeonlin', 'rest', 'assur', 'team', 'tirelessli', 'work', 'remov', 'polici', 'prohibit', '
```

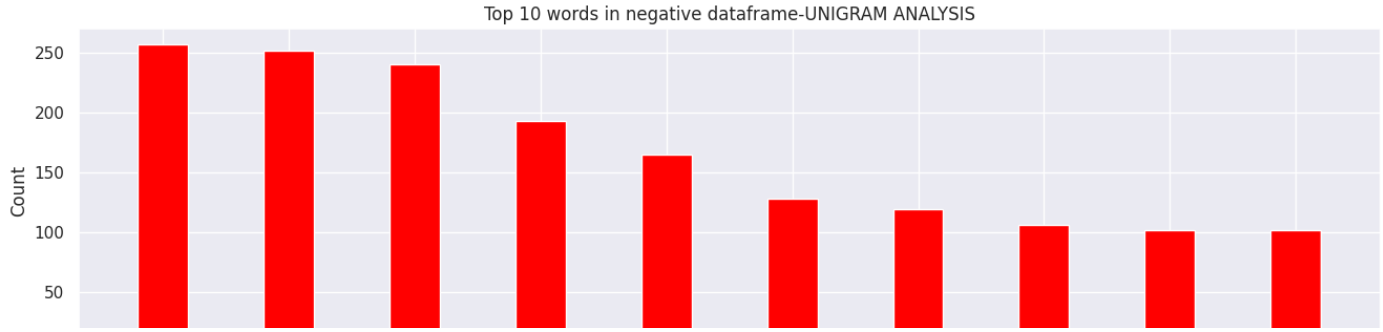
```
#focus on more frequently occuring words for every sentiment=>
#sort in DO wrt 2nd column in each of positiveValues and negativeValues

df_positive=pd.DataFrame(sorted(positiveValues.items(),key=lambda x:x[1],reverse=True))
df_negative=pd.DataFrame(sorted(negativeValues.items(),key=lambda x:x[1],reverse=True))
```

```
pd1=df_positive[0][:10]
pd2=df_positive[1][:10]

ned1=df_negative[0][:10]
ned2=df_negative[1][:10]
```

```
plt.figure(1,figsize=(16,4))

plt.bar(pd1,pd2, color ='green',
        width = 0.4)

plt.xlabel("Words in positive dataframe")
plt.ylabel("Count")
plt.title("Top 10 words in positive dataframe-UNIGRAM ANALYSIS")

plt.savefig("positive-unigram.png")
plt.show()
```




```
plt.figure(1,figsize=(16,4))

plt.bar(ned1,ned2, color ='red',
        width = 0.4)

plt.xlabel("Words in negative dataframe")
plt.ylabel("Count")
plt.title("Top 10 words in negative dataframe-UNIGRAM ANALYSIS")

plt.savefig("negative-unigram.png")
plt.show()
```

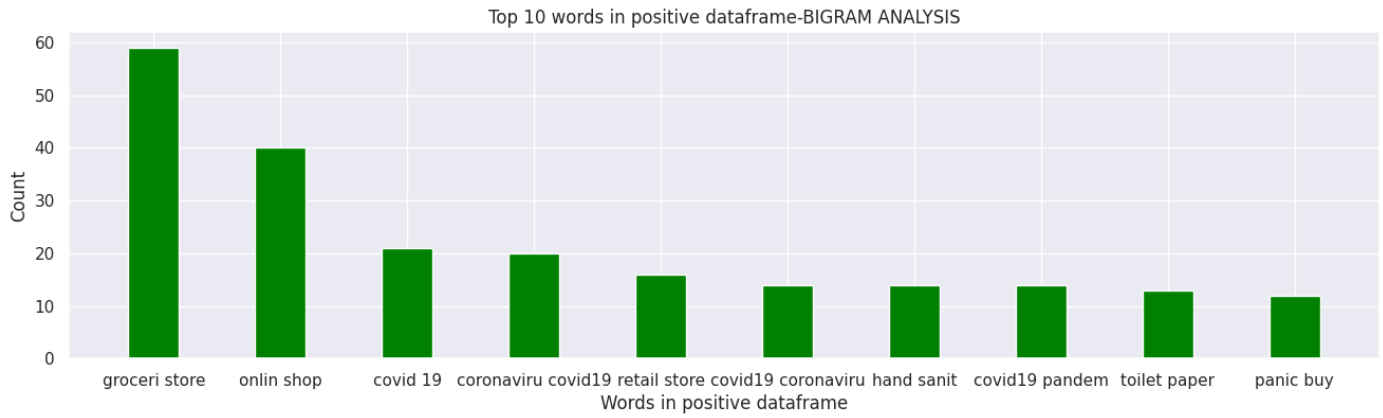



**NGram- Bigrams**

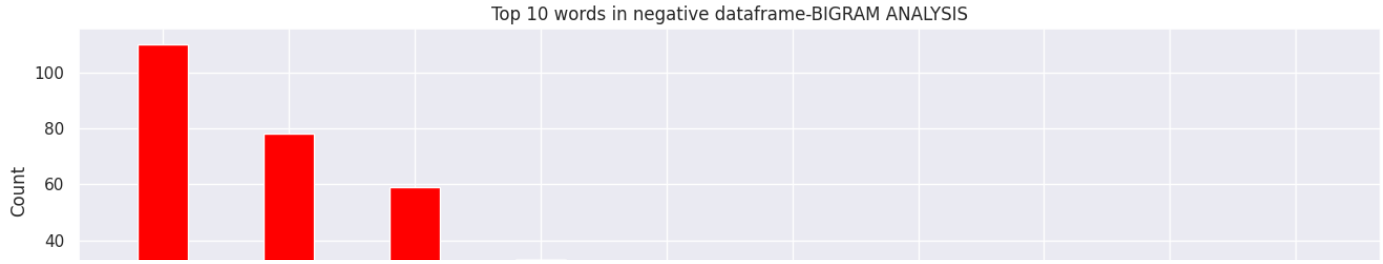

```
positiveValues2=defaultdict(int)
negativeValues2=defaultdict(int)


#get the count of every word in the dataframe where sentiment="positive"
for textt in df[df.Sentiment=="Positive"].text:
  for word in generate_N_grams(textt,2):
    positiveValues2[word]+=1

#get the count of every word in the dataframe where sentiment="negative"
for textt in df[df.Sentiment=="Negative"].text:
  for word in generate_N_grams(textt,2):
    negativeValues2[word]+=1
```

```
Sentence after removing stopwords: ['cia', 'weve', 'got', 'corona', 'hous', '', 'coronaoutbreak', 'corona', 'coronaviru', 'coronaviru
Sentence after removing stopwords: ['know', 'youâ\x92r', 'live', 'apocalyps', 'peopl', 'fight', 'cashier', 'loo', 'roll', 'supermarke
Sentence after removing stopwords: ['everyon', 'keep', 'eye', 'account', 'get', 'debit', 'card', 'shut', 'fraudul', 'charg', 'food',
Sentence after removing stopwords: ['yesterday', 'ask', 'pharmaci', 'constitu', 'hike', 'price', 'essenti', 'good', 'appal', 'learn',
Sentence after removing stopwords: ['connect', 'essenti', 'time', 'crisi', 'cwaunion', 'amp', 'alli', 'ask', 'broadband', 'ceo', 'lif
Sentence after removing stopwords: ['winatlifeonlin', 'rest', 'assur', 'team', 'tirelessli', 'work', 'remov', 'polici', 'prohibit', '
```

```
df_positive2=pd.DataFrame(sorted(positiveValues2.items(),key=lambda x:x[1],reverse=True))
df_negative2=pd.DataFrame(sorted(negativeValues2.items(),key=lambda x:x[1],reverse=True))


pd1bi=df_positive2[0][:10]
pd2bi=df_positive2[1][:10]

ned1bi=df_negative2[0][:10]
ned2bi=df_negative2[1][:10]


plt.figure(1,figsize=(16,4))

plt.bar(pd1bi,pd2bi, color ='green',
        width = 0.4)

plt.xlabel("Words in positive dataframe")
plt.ylabel("Count")
plt.title("Top 10 words in positive dataframe-BIGRAM ANALYSIS")

plt.savefig("positive-bigram.png")
plt.show()
```




```
plt.figure(1,figsize=(16,4))

plt.bar(ned1bi,ned2bi, color ='red',
        width = 0.4)

plt.xlabel("Words in negative dataframe")
plt.ylabel("Count")
plt.title("Top 10 words in negative dataframe-BIGRAM ANALYSIS")

plt.savefig("negative-bigram.png")
plt.show()
```

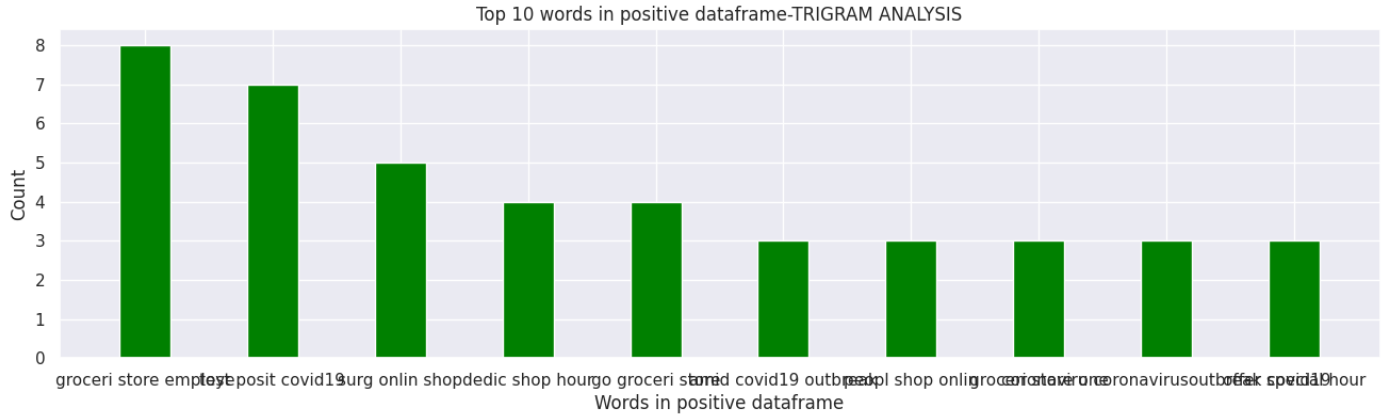



**NGram- Trigrams**

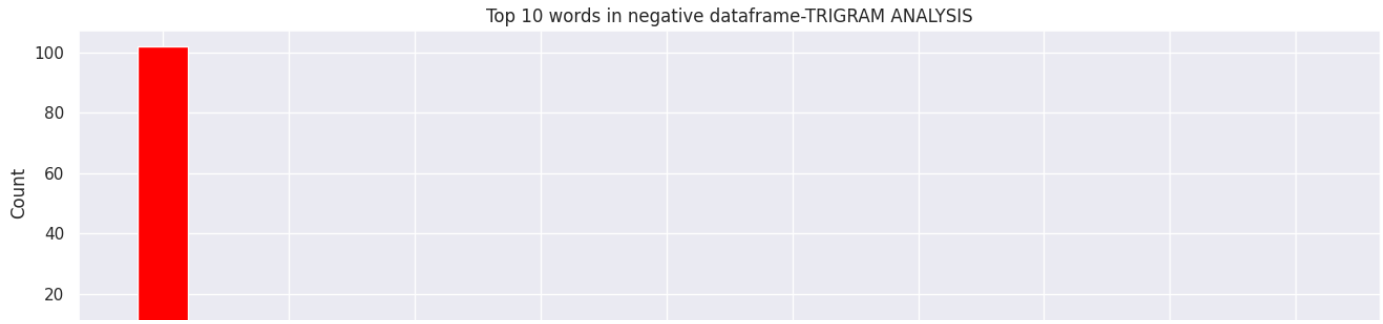
```
positiveValues3=defaultdict(int)
negativeValues3=defaultdict(int)


#get the count of every word in the dataframe where sentiment="positive"
for textt in df[df.Sentiment=="Positive"].text:
  for word in generate_N_grams(textt,3):
    positiveValues3[word]+=1

#get the count of every word in the dataframe where sentiment="negative"
for textt in df[df.Sentiment=="Negative"].text:
  for word in generate_N_grams(textt,3):
    negativeValues3[word]+=1
```

```
Sentence after removing stopwords: ['everyon', 'keep', 'eye', 'account', 'get', 'debit', 'card', 'shut', 'fraudul', 'charg', 'food',
Sentence after removing stopwords: ['yesterday', 'ask', 'pharmaci', 'constitu', 'hike', 'price', 'essenti', 'good', 'appal', 'learn',
Sentence after removing stopwords: ['connect', 'essenti', 'time', 'crisi', 'cwaunion', 'amp', 'alli', 'ask', 'broadband', 'ceo', 'lif
Sentence after removing stopwords: ['winatlifeonlin', 'rest', 'assur', 'team', 'tirelessli', 'work', 'remov', 'polici', 'prohibit', '
```

```
#focus on more frequently occuring words for every sentiment=>
#sort in DO wrt 2nd column in each of positiveValues and negativeValues


df_positive3=pd.DataFrame(sorted(positiveValues3.items(),key=lambda x:x[1],reverse=True))
df_negative3=pd.DataFrame(sorted(negativeValues3.items(),key=lambda x:x[1],reverse=True))


pd1tri=df_positive3[0][:10]
pd2tri=df_positive3[1][:10]

ned1tri=df_negative3[0][:10]
ned2tri=df_negative3[1][:10]


plt.figure(1,figsize=(16,4))

plt.bar(pd1tri,pd2tri, color ='green',
        width = 0.4)

plt.xlabel("Words in positive dataframe")
plt.ylabel("Count")
plt.title("Top 10 words in positive dataframe-TRIGRAM ANALYSIS")

plt.savefig("positive-trigram.png")
plt.show()
```




```
plt.figure(1,figsize=(16,4))

plt.bar(ned1tri,ned2tri, color ='red',
        width = 0.4)

plt.xlabel("Words in negative dataframe")
plt.ylabel("Count")
plt.title("Top 10 words in negative dataframe-TRIGRAM ANALYSIS")

plt.savefig("negative-trigram.png")
plt.show()
```




## Task 5 –Text Classification / Prediction

**Steps in Modelling process using sklearn package**

- Split the data into training and test sets (80% train, 20% test)
- Extract features from the training data using TfidfVectorizer.
- Transform the test data into the same feature vector as the training data.
- Train the classifier
- Evaluate the classifier

```
df.head()
```

| | OriginalTweet | Sentiment | text |
|---|---|---|---|
| 0 | @MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i... | Negative | menyrbi philgahan chrisitv httpstcoifz9fan2pa ... |
| 1 | advice Talk to your neighbours family to excha... | Positive | advic talk neighbour famili exchang phone numb... |
| 2 | Coronavirus Australia: Woolworths to give elde... | Positive | coronaviru australia woolworth give elderli di... |
| 3 | My food stock is not the only one which is emp... | Positive | food stock one empti pleas dont panic enough f... |
| 4 | Me, ready to go at supermarket during the #COV... | Negative | readi go supermarket covid19 outbreak im paran... |

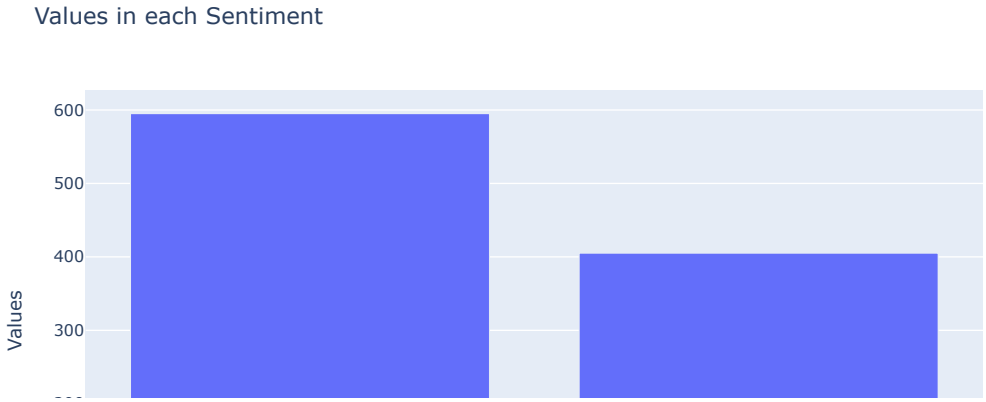
```
df['Sentiment'] = df.Sentiment.map({'Positive':1, 'Negative':0})
```

```
docs = list(df['text'])
tfidf_vectorizer = TfidfVectorizer(use_idf=True, max_features = 20000)
tfidf_vectorizer_vectors = tfidf_vectorizer.fit_transform(docs)
docs = tfidf_vectorizer_vectors.toarray()
```

```
X = docs
y = df['Sentiment']
print(X.shape, y.shape)
```

```
(1000, 4836) (1000,)
```

```
import plotly.graph_objects as go
fig = go.Figure([go.Bar(x=y.value_counts().index, y=y.value_counts().tolist())])
fig.update_layout(
    title="Values in each Sentiment",
    xaxis_title="Sentiment",
    yaxis_title="Values")
fig.show()
```




```
#Train-Test Split
#split the data into 80% training and 20% testing
SEED=123
X_train,X_test,y_train,y_test=train_test_split(X, y, test_size=0.2, random_state=SEED, stratify=y)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(800, 4836) (800,)
(200, 4836) (200,)
```

**Decision Tree Classifier**

```
from sklearn.tree import DecisionTreeClassifier
SEED=123
dt = DecisionTreeClassifier(random_state=SEED)
%time dt.fit(X_train, y_train)

y_pred_train = dt.predict(X_train)
y_pred_test = dt.predict(X_test)
print("\nTraining Accuracy score:",accuracy_score(y_train, y_pred_train))
print("Testing Accuracy score:",accuracy_score(y_test, y_pred_test))
```

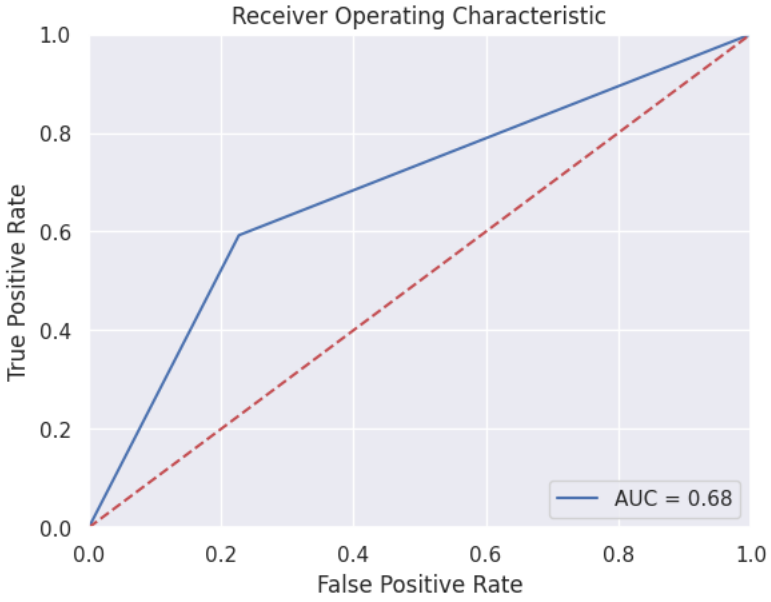
```
CPU times: user 401 ms, sys: 402 µs, total: 401 ms
Wall time: 402 ms

Training Accuracy score: 1.0
Testing Accuracy score: 0.7
```

```
cm = confusion_matrix(y_test, y_pred_test)
# print('Confusion matrix\n', cm)

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive', 'Actual Negative'],
                         index=['Predict Positive', 'Predict Negative'])
sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
plt.show()
```

```
print(classification_report(y_test, y_pred_test, target_names=['Negative', 'Positive']))
```

```
              precision    recall  f1-score   support

    Negative       0.74      0.77      0.75       119
    Positive       0.64      0.59      0.62        81

    accuracy                           0.70       200
   macro avg       0.69      0.68      0.68       200
weighted avg       0.70      0.70      0.70       200
```

```
import sklearn.metrics as metrics
probs = dt.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

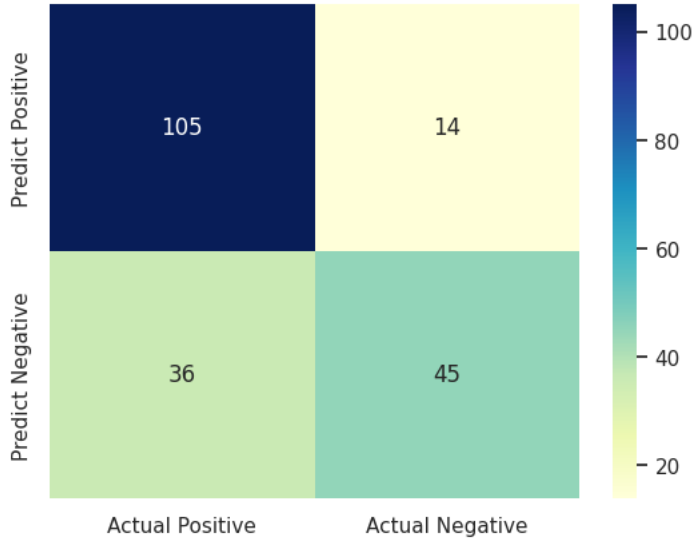



**Support Vector Machines Classifier**

```
from sklearn.svm import LinearSVC
svc =  LinearSVC(class_weight='balanced')
%time svc.fit(X_train, y_train)

y_pred_train = svc.predict(X_train)
y_pred_test = svc.predict(X_test)
print("\nTraining Accuracy score:",accuracy_score(y_train, y_pred_train))
print("Testing Accuracy score:",accuracy_score(y_test, y_pred_test))
```

```
CPU times: user 29.9 ms, sys: 0 ns, total: 29.9 ms
Wall time: 36.7 ms

Training Accuracy score: 1.0
Testing Accuracy score: 0.75
```

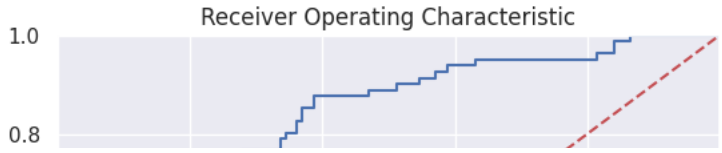
```
cm = confusion_matrix(y_test, y_pred_test)
# print('Confusion matrix\n', cm)

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive', 'Actual Negative'],
                         index=['Predict Positive', 'Predict Negative'])
sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
plt.show()
```




```
print(classification_report(y_test, y_pred_test, target_names=['Negative', 'Positive']))
```

```
                  precision    recall  f1-score   support

        Negative       0.74      0.88      0.81       119
        Positive       0.76      0.56      0.64        81

        accuracy                           0.75       200
       macro avg       0.75      0.72      0.73       200
    weighted avg       0.75      0.75      0.74       200
```

```
probs = svc._predict_proba_lr(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




**Inference:** From the evaluation metrics provided, it is evident that the SVM classifier outperforms the Decision Tree Classifier in terms of accuracy, precision, recall, F1-score, and AUC. The Decision Tree Classifier has an accuracy of 0.70, which means it correctly predicts the target variable 70% of the time. Similarly, it has a precision of 0.70, indicating that 70% of the positive predictions made by the classifier are true positives. The recall for the Decision Tree Classifier is 0.68, meaning that 68% of the actual positive instances are correctly identified. The F1-score, which combines precision and recall, is 0.70.

On the other hand, the SVM classifier has a higher accuracy of 0.75, indicating that it has an overall higher percentage of correct predictions compared to the Decision Tree Classifier. It also has a precision of 0.75, meaning 75% of the positive predictions made by the SVM classifier are true positives. The recall for the SVM classifier is 0.72, indicating that it correctly identifies 72% of the actual positive instances. The F1-score for the SVM classifier is 0.74, which is higher than that of the Decision Tree Classifier.

When considering the AUC score, which measures the overall performance of a classifier by taking into account the trade-off between true positive rate and false positive rate, the SVM classifier also outperforms the Decision Tree Classifier. The Decision Tree Classifier has an AUC of 0.68, while the SVM classifier has a higher AUC of 0.81. A higher AUC value typically suggests a better-performing model.

Based on these evaluation metrics, it can be inferred that the SVM classifier is more effective in predicting and classifying the data compared to