MULTI-CLASS CLASSIFICATION OF TWEETS

CETM47 - Machine Learning And Data Analytics

By Reshmith Ramesh

Reshmith Ramesh

Bi29lh / 219417668

University of Sunderland

Department of technology

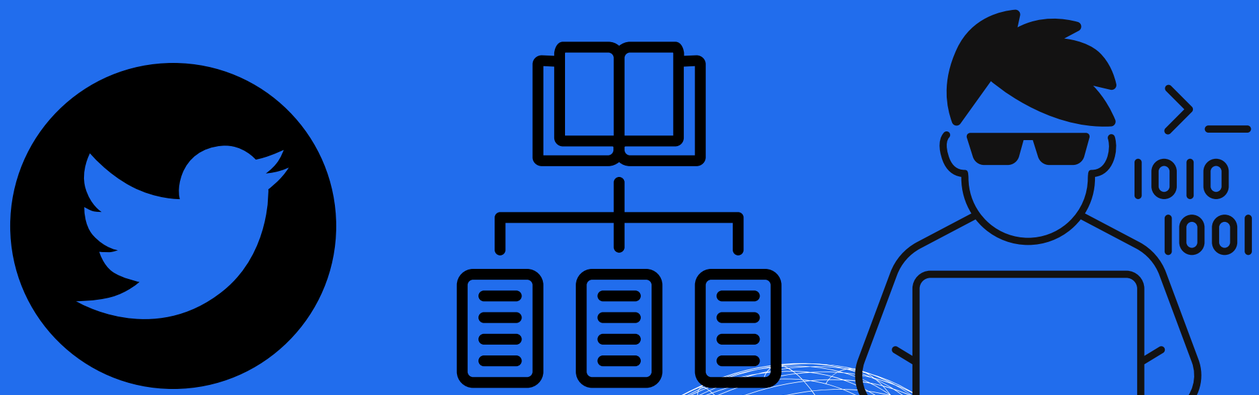
What is NLP?

What if Computers can understand you in your language?

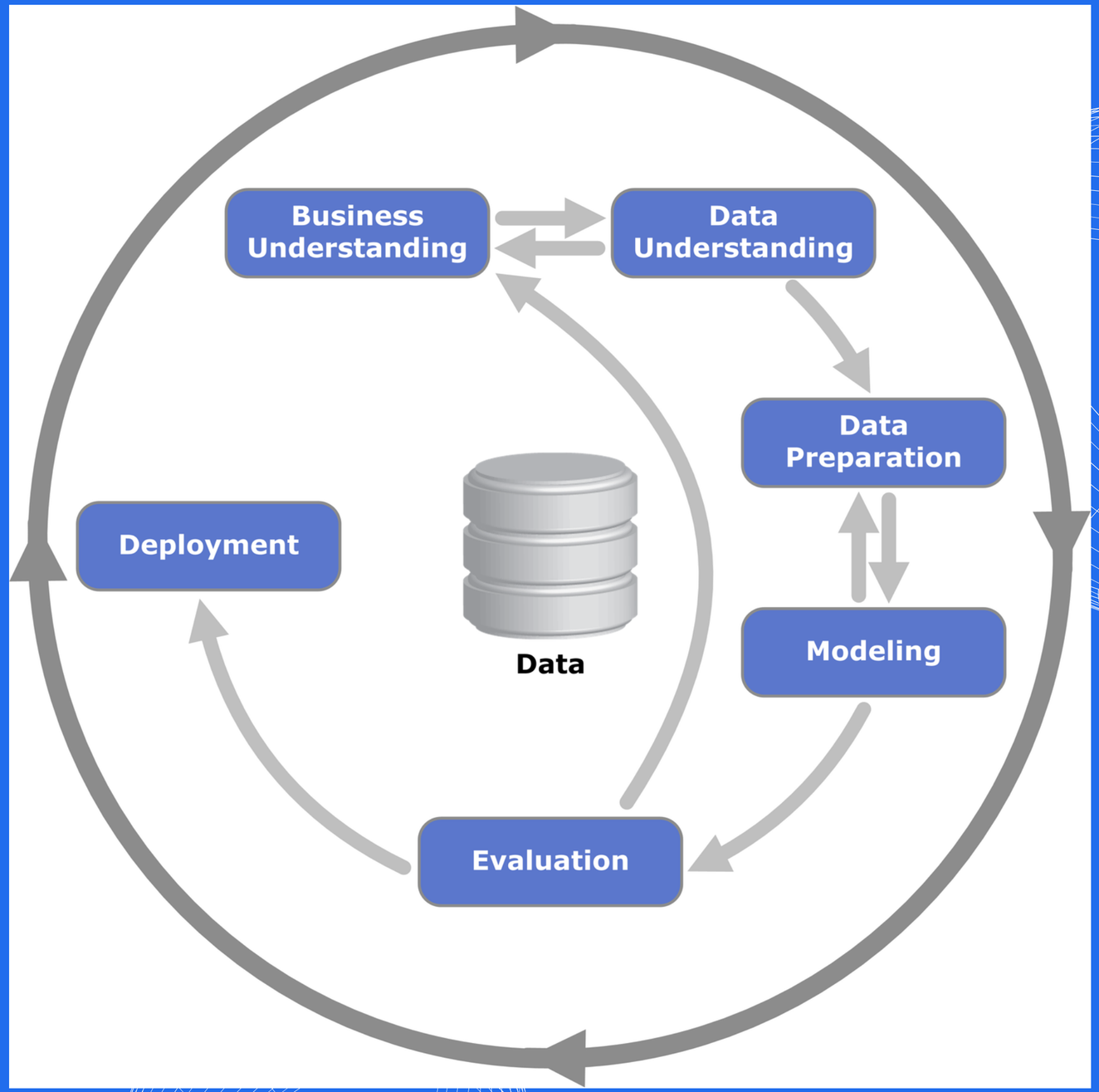
Objectives:

1. Multi-class classification
2. Follow CRISP-DM
3. Two feature extraction methods
4. Three algorithms
5. Multiple Experiments
6. Evaluation and Comparison
7. Final NLP Model

What is Multi-Class Classification?

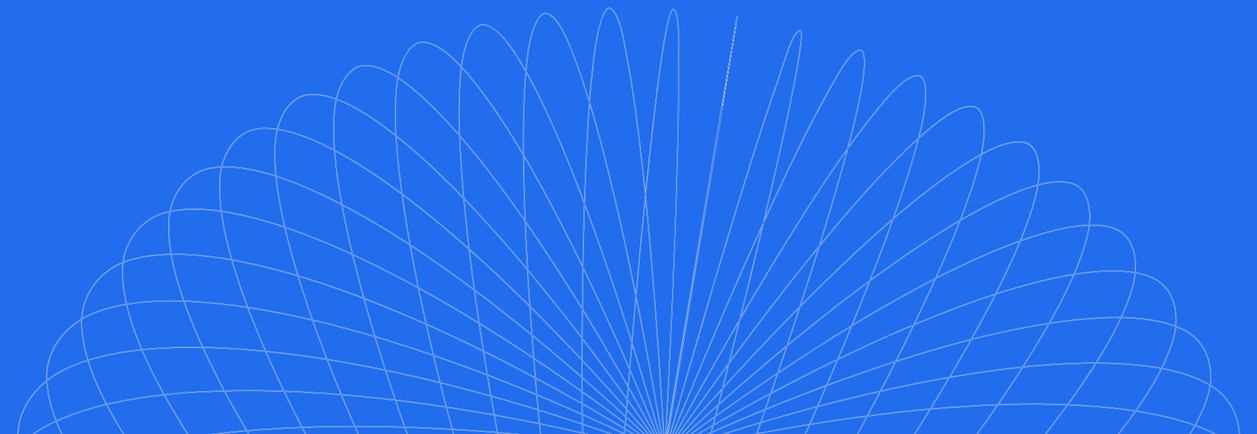


What is CRISP - DM



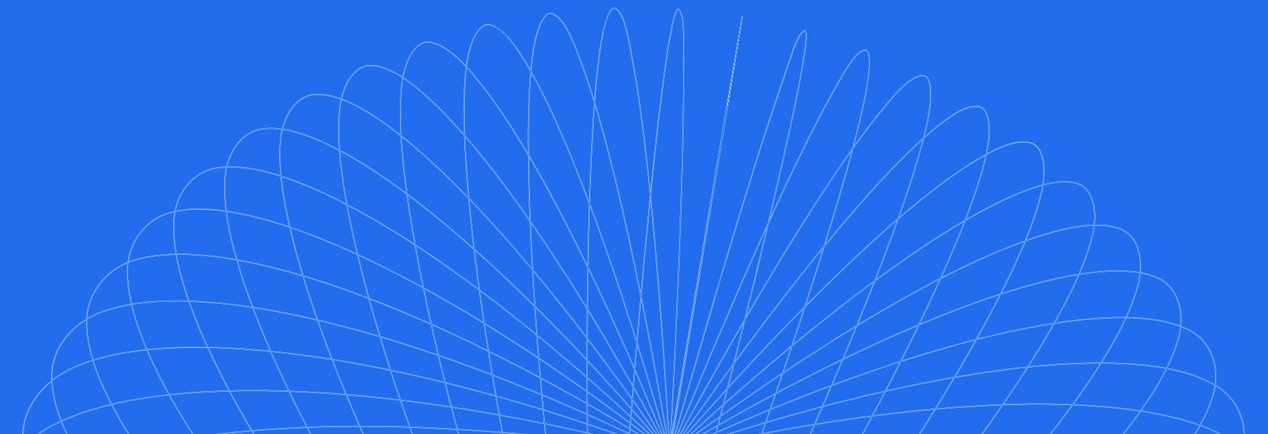
Feature Extractions

1. Bag-Of-Words
2. TF-IDF



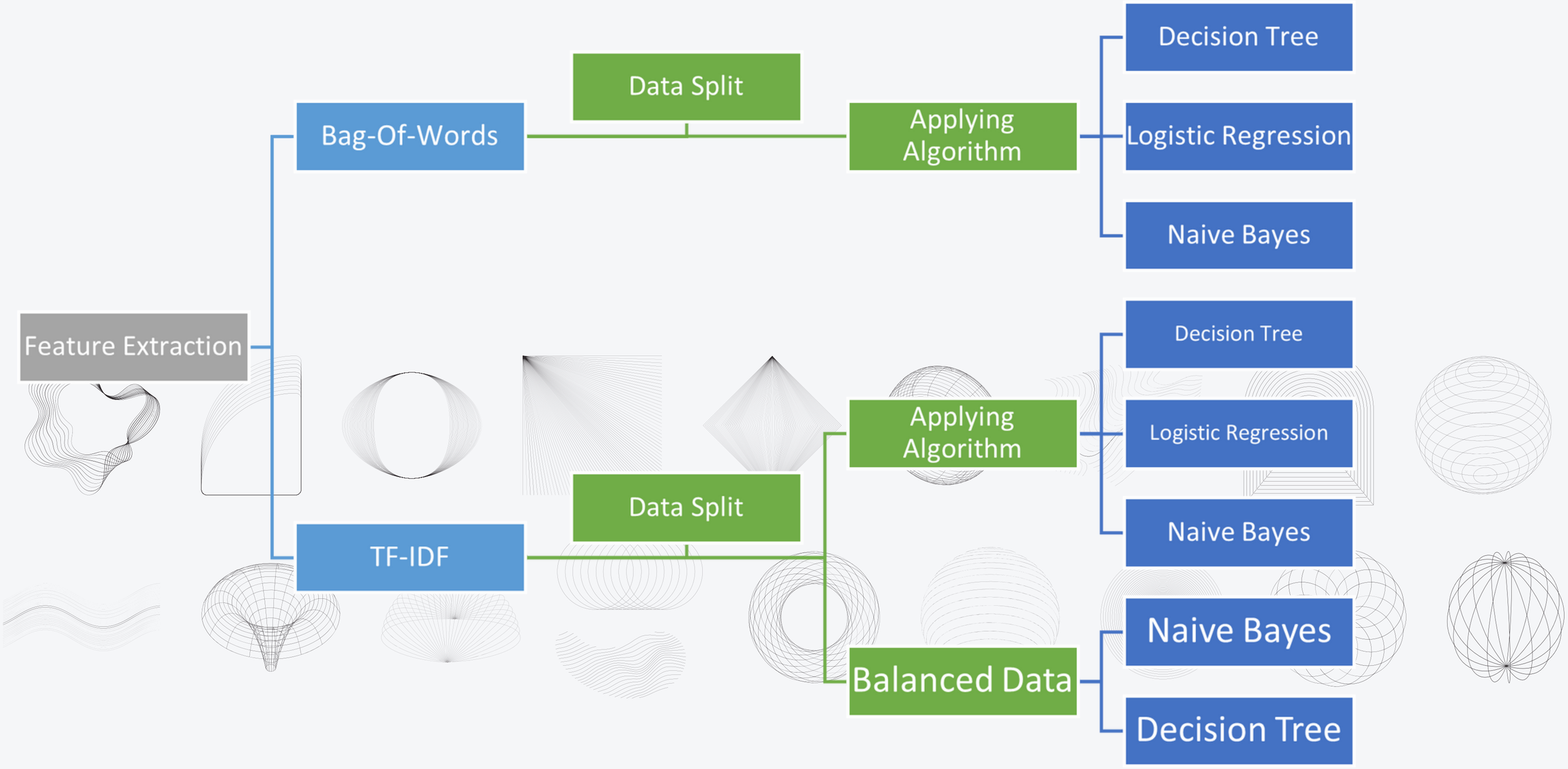
Algorithms

1. Decision Tree
2. Logistic Regression
3. Naive Bayes



Two Approaches with multiple experiments

Approach 1


Feature Extraction
Bag-Of-Words
Data Split
Applying Algorithm
Decision Tree
Logistic Regression
Naive Bayes
TF-IDF
Data Split
Applying Algorithm
Decision Tree
Logistic Regression
Naive Bayes
Balanced Data
Naive Bayes
Decision Tree

In this video

1. Code Walkthrough
2. Results and Evaluation
3. Future works and Conclusion

In [1]:
```
#All the necessary libraries are imported here

import numpy as np
import pandas as pd
import json
import spacy
from plotly.offline import init_notebook_mode, iplot, plot
import plotly as py
init_notebook_mode(connected=True)
import plotly.graph_objs as go

from wordcloud import wordcloud
import html
from html.parser import HTMLParser

import seaborn as sns
import string
import plotly.express as px
import numpy as np
import pandas as pd
import numpy              as np
import pandas             as pd
import matplotlib.pyplot as plt
import seaborn           as sns
import plotly.graph_objs as go
import plotly.express    as px
import nltk
from nltk.corpus import stopwords
from nltk.corpus import opinion_lexicon
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
```

In [2]:
```
#Importing the JSON file as mydata

with open('CETM47-22_23-AS2-Data.json') as file:
    mydata = json.load(file)
```

In [3]:
```
#print(mydata)
```

In [4]:
```
#########Data Exploration and Pre-processing############
```

In [5]:
```
#Checking the number of tweets or entries in the dataset
number_of_tweets = len(mydata)
print("Number of tweets in my dataset:", number_of_tweets)
```

```
Number of tweets in my dataset: 6443
```

In [6]:
```
#There are 6443 tweet entries
#This goes with the information provided in the assignment details.
```

In [7]:
```
#Finding all the labels in the dataset

label_names = set(tweets['label_name'] for tweets in mydata)

print("label_names:")
for label_names in label_names:
    print(label_names)
```

```
label_names:
science_&_technology
daily_life
business_&_entrepreneurs
sports_&_gaming
pop_culture
arts_&_culture
```

In [9]:
```
#Checking if there are any duplicate tweets in the dataset based on "id"

ids = set()
duplicate_count = 0

for tweet in mydata:
    tweet_id = tweet['id']

    if tweet_id in ids:
        duplicate_count += 1
    else:
        ids.add(tweet_id)

# Printing the result
print("Count of duplicate tweets:", duplicate_count)
```

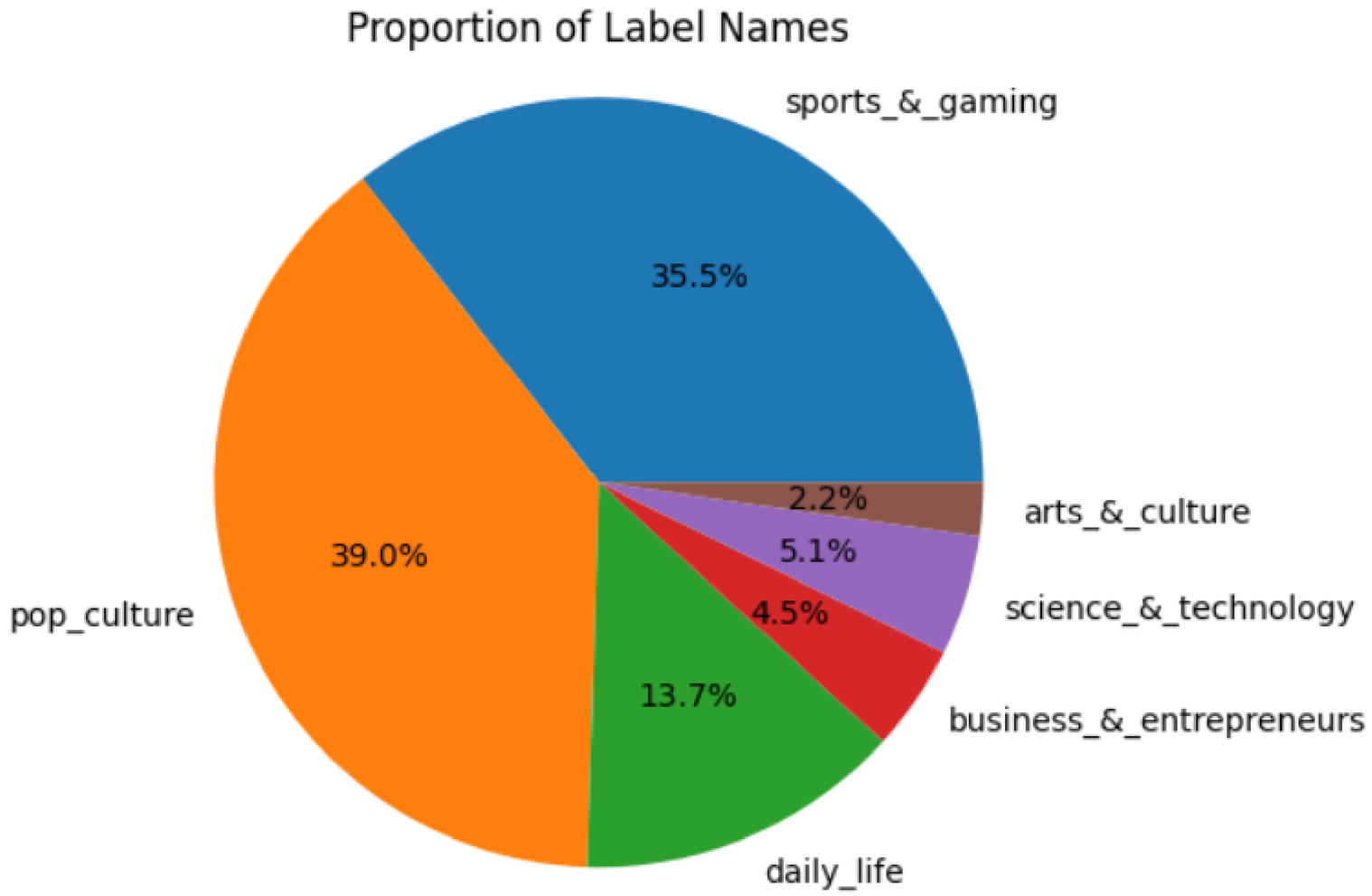
```
Count of duplicate tweets: 0
```

In [10]:
```
#We can see that there are no duplicate tweets in the dataset based on "id"
```

In [11]:
```
#However, let us check if there are any duplicate based on the text. It is possible that users copy and paste other tweets
```

In [12]:
```
texts = set()
duplicate_text = 0

for tweet in mydata:
    tweet_text = tweet['text'].strip()  # Strip leading and trailing whitespaces

    if tweet_text in texts:
        duplicate_text += 1
    else:
        texts.add(tweet_text)
```

```
plt.title('Proportion of Label Names')
plt.axis('equal')  # Equal aspect ratio ensures that the pie is drawn as a circle
plt.show()
```



In [18]:
```
#checking if there are any missing values

if new_data2.isnull().values.any():
```

```
new_data2.head(10)
```

	text	date	label	id	label_name	cleaned_text
0	The {@Clinton LumberKings@} beat the {@Cedar R...	2019-09-08	4	1170516324419866624	sports_&_gaming	The {@Clinton LumberKings@} beat the {@Cedar R...
1	I would rather hear Eli Gold announce this Aub...	2019-09-08	4	1170516440690176006	sports_&_gaming	I would rather hear Eli Gold announce this Aub...
2	Someone take my phone away, I'm trying to not ...	2019-09-08	4	1170516543387709440	sports_&_gaming	Someone take my phone away, I'm trying to not ...
3	A year ago, Louisville struggled to beat an FC...	2019-09-08	4	1170516620466429953	sports_&_gaming	A year ago, Louisville struggled to beat an FC...
4	Anyone know why the #Dodgers #Orioles game nex...	2019-09-08	4	1170516711411310592	sports_&_gaming	Anyone know why the #Dodgers #Orioles game nex...
5	I don't care. you gave him a shot, he is strug...	2019-09-08	4	1170516891053580288	sports_&_gaming	I don't care. you gave him a shot, he is strug...
6	Okay how can I watch the {@Arkansas State Foot...	2019-09-08	4	1170516916554936322	sports_&_gaming	Okay how can I watch the {@Arkansas State Foot...
7	Check out largest crowds ever for a basketball...	2019-09-08	4	1170516940902805504	sports_&_gaming	Check out largest crowds ever for a basketball...
8	I voted #WeWantNCAAFootball on {{USERNAME}}	2019-09-08	4	1170517092489187328	sports_&_gaming	I voted #WeWantNCAAFootball on {{USERNAME}} . ...
9	Streaming a new game #minionmasters come stop ...	2019-09-08	4	1170546366566846464	sports_&_gaming	Streaming a new game #minionmasters come stop ...

Data Transformation

1. Removed all usernames (@Username)
2. Removed special characters
3. Removed URL and Username words
4. Converted all words to lower_case
5. Converted all emoticons to its textual representation
6. Converted apostraphes
7. Removed instances of "\u"
8. Removed more special characters
9. Removed Stopwords
10. Tokenized
11. Lemmatized

In [24]:
```
#Lets remove all the mentions of #users (such as @username ) from the tweet and store them in the 'cleaned_text'

#We can use Pandas, str.replace method for this

new_data2['cleaned_text'] = new_data2['cleaned_text'].str.replace(r'@\w+', '', regex=True)
#new_data2.head(10)
```

In [25]:
```
#removing the #special_characters from the text. This will be reflected in the "cleaned_text" column

new_data2['cleaned_text'] = new_data2['cleaned_text'].str.replace(r'[{}[\]()?<>!"£$%^&*\\//@]', '', regex=True)
#new_data2.head(10)
```

In [26]:
```
#From exploring the dataset, we have seen that there are many "URL" and "USERNAME" in the text, we can remove those using the sam

new_data2['cleaned_text'] = new_data2['cleaned_text'].str.replace(r'\b(?:URL|USERNAME)\b', '', regex=True)
#new_data2.head(10)
```

In [27]:
```
#Now can change all the text to #lower_case. This is better for the model
#There are several reasons to convert the words to lower case such as Normalization, word embedding etc.


new_data2['cleaned_text'] = new_data2['cleaned_text'].apply(lambda x: x.lower())
#new_data2.head(10)
```

In [28]:
```
# word_counts = new_data2['text'].str.split().apply(len).sum()
# print("Total number of words in the 'text' column:", word_counts)

# word_count = new_data2['cleaned_text'].str.split().apply(len).sum()
# print("Total number of words in the 'cleaned_text' column:", word_count)
```

In [29]:
```
#Creating a dictionary for the emoticons
```

```
},
{
    "text": "The #Dbacks, marked by mediocrity for
      four months, are hotter than any team in the
      game right now. In tonight s  emergency  episode
      , {{USERNAME}} answers many of your questions
      and poses a few new ones. Tune in here: {{URL}}
      ",
    "date": "2019-09-08",
    "label": 4,
    "id": 1170577365467983872,
    "label_name": "sports_&_gaming"
}
```


```
: #From exploring the dataset, we have seen that there are many "URL" and "USERNAME" in the text, we can remove those using the sam

new_data2['cleaned_text'] = new_data2['cleaned_text'].str.replace(r'\b(?:URL|USERNAME)\b', '', regex=True)
#new_data2.head(10)
```

"USERNAME" and "URL" in several tweets.

```
{
  "text": "I have little to no voice left, I\u2019m
    completely exhausted...and wouldn\u2019t change
    it for the world. Nothing more exciting as a
    broadcaster to call a game winning play as time
    expires. Tonight was my second time calling a GW
    FG in a monumental {{USERNAME}} win over
    Georgetown.",
  "date": "2019-09-08",
  "label": 4,
```


```
#We can see that there are many instances of \u followed by numbers or characters. We can use regex to remove these
```

```
new_data2['cleaned_text'] = new_data2['cleaned_text'].str.replace(r'\\u\w{4}', '', regex=True)

#new_data2.head(10)
```

Instances of "\u xxxx" are removed

```
#Creating a dictionary for the emoticons
#this dictionary will be used to replace any emojis in the text

#reference: https://github.com/NeelShah18/emot/blob/master/emot/emo_unicode.py


Emoji_dic = {
    u":-)":"Happy face or smiley",
    u":-))":"Very Happy face or smiley",
    u":-)))":"Very very Happy face or smiley",
    u":)":"Happy face or smil
    u":))":"Very Happy face o
    u":)))":"Very very Happy
    u":-]":"Happy face or smi
    u":]":"Happy face or smil
    u":-3":"Happy face smiley
    u":3":"Happy face smiley"
    u":->":"Happy face smiley
    u":>":"Happy face smiley"
    u"8-)":"Happy face smiley
    u":o)":"Happy face smiley
    u":-}":"Happy face smiley
    u":}":"Happy face smiley"
    u":-)":"Happy face smiley
    u":c)":"Happy face smiley
    u":^)":"Happy face smiley
    u"=]":"Happy face smiley",
    u"=)":"Happy face smiley",
    u":-D":"Laughing, big grin or laugh with glasses",
    u":D":"Laughing, big grin or laugh with glasses",
    u"8-D":"Laughing, big grin or laugh with glasses",
    u"8D":"Laughing, big grin or laugh with glasses",
```

```
St. Frances Academy 6. {{URL}} via {@MaxPreps@} ', 'date': '2019-09-15', 'lab
'sports_&_gaming'}, {'text': 'A fantastic game with {{USERNAME}} u/9 girls ye
esent {@Essex County FA@} , thank you to Ryman for bringing the Colebrook gir
re  vs ⚫⚽ {{USERNAME}} {{USERNAME}} {{USERNAME}} ', 'date': '2019-09-15',
e': 'sports_&_gaming'}, {'text': '{@Paytm@} IND vs SA  September 15 🕗 7 PM
15', 'label': 4, 'id': 1173141906177445890, 'label_name': 'sports_&_gaming'},
ERNAME}} vs {{USERNAME}} Res', 'date': '2019-09-15', 'label': 4, 'id': 117314
{'text': 'Game Day...  ⚽| vs #OxleathersFC | {{USERNAME}} Div 3 ⏰| 1030 |
| The Lowers, Abbots Bromley School   #CmonTheStags', 'date': '2019-09-15',
e': 'sports_&_gaming'}, {'text': 'Season food schedule so far:  Wk 1 vs Miami
izona - Chicken chimichangas / choc n banana chimichangas. Wk 3 vs Kansas -
k Wk 5 vs Pittsburgh - Pierogies & primanti sandwich', 'date': '2019-09-15',
```

Emojis are converted to textual representation

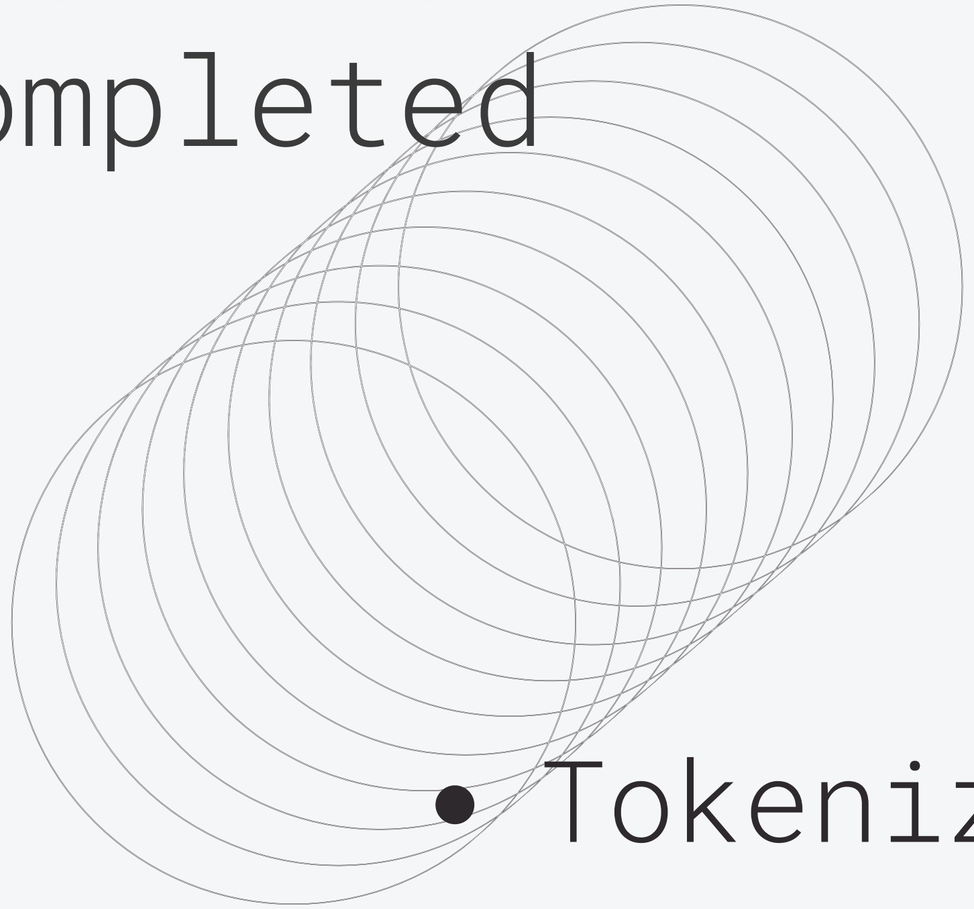
```
#It is common to use apostrophe in the words instead of using the entire word while tweeting.
#This can be converted to proper word which will be benefical for the NLP model.
#Using spaCy library, we can convert all the apostraphe words to proper words.

#A new function is created and then applied to the dataframe for the column "cleaned_text"
#reference: https://spacy.io

#Loading the english language model 'en_core_web_sm'

nlp = spacy.load('en_core_web_sm')

# Function to convert apostrophe words to proper words

def convert_apostrophe_words(text):
    doc = nlp(text)
    converted_text = ' '.join([token.lemma_ if token.lemma_ != "-PRON-" else token.text for token in doc])
    return converted_text

# Apply the conversion to the "cleaned_text" column in the dataset

new_data2['cleaned_text'] = new_data2['cleaned_text'].apply(convert_apostrophe_words)

# Display the updated dataset
#new_data2.head(10)
```

Apostraphe words converted to normal words using SpaCy

```
#Making sure if all the special characters are removed.

characters_to_check = r'!"#$%&\'()*+,-./:;<=>?@[\]^_`{|}~'
pattern = '[' + re.escape(characters_to_check) + ']'


new_data2['cleaned_text'] = new_data2['cleaned_text'].str.replace(pattern, '', regex=True)
```

Making sure special characters are removed.

Data Transformation is almost completed



- Tokenization
- Lemmatization

Tokenized the cleaned_text

```
# #Lets Tokenize the Text
# #Necessary libraries are already imported
# #The tokenized text are stored on another column

new_data2['tok_text'] = new_data2['cleaned_text'].apply(lambda x: word_tokenize(x))
```

```
new_data2.head(10)
```

	text	date	label	id	label_name	cleaned_text	tok_text
0	The {@Clinton LumberKings@} beat the {@Cedar R...	2019-09-08	4	1170516324419866624	sports_&_gaming	lumberking beat rapid kernel 4 0 game 1 wester...	[lumberking, beat, rapid, kernel, 4, 0, game, ...
1	I would rather hear Eli Gold announce this Aub...	2019-09-08	4	1170516440690176006	sports_&_gaming	I would rather hear eli gold announce auburn g...	[I, would, rather, hear, eli, gold, announce, ...
2	Someone take my phone away, I'm trying to not ...	2019-09-08	4	1170516543387709440	sports_&_gaming	someone take phone away I 'm try look blackhaw...	[someone, take, phone, away, I, ', m, try, loo...
3	A year ago, Louisville struggled to beat an FC...	2019-09-08	4	1170516620466429953	sports_&_gaming	year ago louisville struggle beat fcs opponent...	[year, ago, louisville, struggle, beat, fcs, o...
4	Anyone know why the #Dodgers #Orioles game nex...	2019-09-08	4	1170516711411310592	sports_&_gaming	anyone know dodger oriole game next thursday 9...	[anyone, know, dodger, oriole, game, next, thu...
5	I don't care. you gave him a shot, he is strug	2019-09-08	4	1170516891053580288	sports_&_gaming	I care give shot struggle put joey see offer g	[I, care, give, shot, struggle, put, joey, see

word_tokenize from NLTK library is used

Lemmatized the Tokenized text

```
nltk.download('wordnet')
```

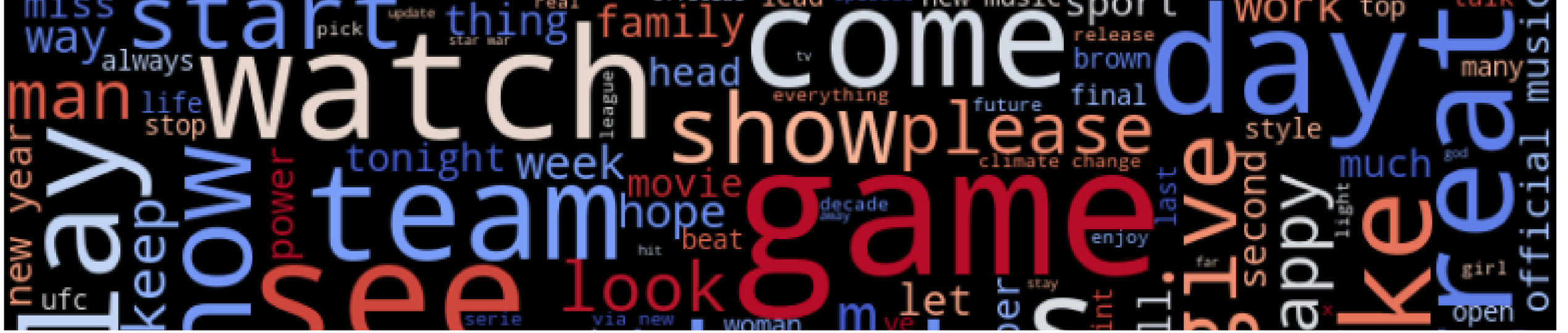
```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\reshm\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
True
```

```
#Lemmatization is done and a new column is created

lemmatizing = WordNetLemmatizer()

new_data2['lemm_text'] = new_data2['tok_text'].apply(lambda x: ' '.join([lemmatizing.lemmatize(i) for i in x]))
new_data2['lemm_text'].head(10)
```

```
0    lumberking beat rapid kernel 4 0 game 1 wester...
1    I would rather hear eli gold announce auburn g...
2    someone take phone away I ’ m try look blackha...
3    year ago louisville struggle beat fcs opponent...
4    anyone know dodger oriole game next thursday 9...
5    I care give shot struggle put joey see offer g...
6             okay I watch state football unlv game fb
7    check large crowd ever basketball game phil ar...
8          I vote wewantncaafootball ’ good video game
9    stream new game minionmaster come stop join I ...
Name: lemm_text, dtype: object
```

'wordnet' package from NLTK is used

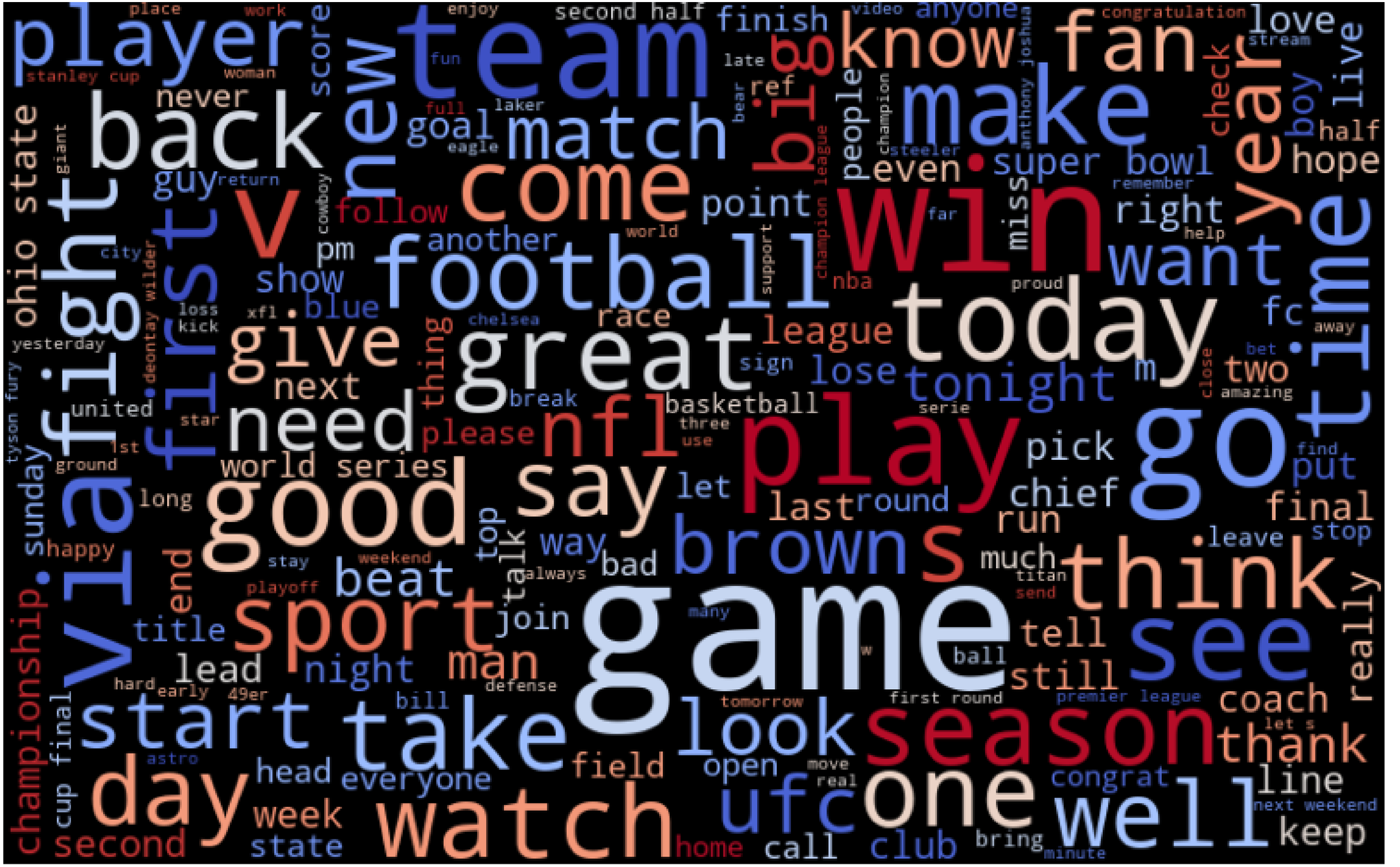



```
#We can visualize the words in the Lemmatized Text

all_words = ' '.join([text for text in new_data2['lemm_text']])
from wordcloud import WordCloud
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110,background_color='black', colormap='coolwa

plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.title("Most Common words in Lemmatized Text")
plt.show()
```

wordCloud and matplotlib are used for the visualization

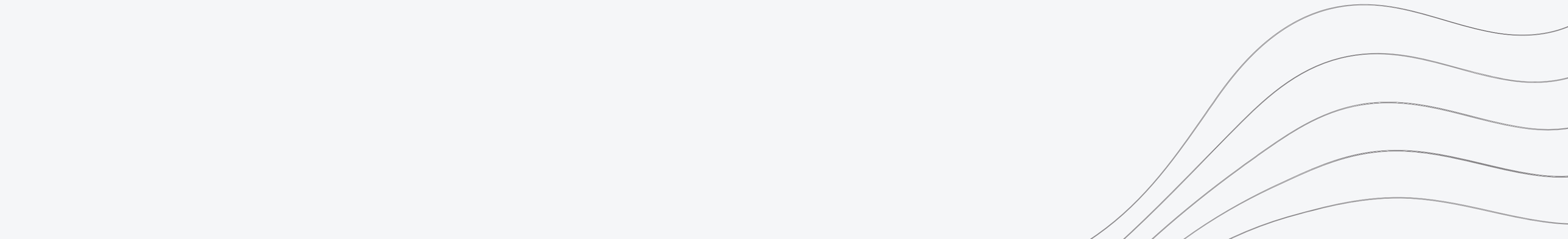
Most Common Words in Pop Culture)

Most Common Words in Sports and Gaming)



- The datapreprepration is completed.
- The transformed data is stored in lemm_text. This is used for training the models.

Feature Extraction using Bag-Of-Words and data split



```
#Using BoW method, we are extraction features

#from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer()

bow_count = count_vect.fit_transform(new_data2['lemm_text'])
```

```
#from sklearn.preprocessing import StandardScaler
```

```
#Scaling

bow_scale = StandardScaler(with_mean=False).fit_transform(bow_count)
bow_scale
```

```
<6442x17986 sparse matrix of type '<class 'numpy.float64'>'
        with 91030 stored elements in Compressed Sparse Row format>
```

```
#Splitting the data for features extracted using BoW
```

```
X = bow_scale
Y = new_data2['label_name']

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.3, stratify=Y, random_state=42)
```

```
print("Unique labels in train set:", np.unique(Y_train))
print("Unique labels in test set:", np.unique(Y_test))
```

```
Unique labels in train set: ['arts_&_culture' 'business_&_entrepreneurs' 'daily_life' 'pop_culture'
 'science_&_technology' 'sports_&_gaming']
```

```
print("Unique labels in train set:", np.unique(Y_train))
print("Unique labels in test set:", np.unique(Y_test))
```

```
Unique labels in train set: ['arts_&_culture' 'business_&_entrepreneurs' 'daily_life' 'pop_culture'
 'science_&_technology' 'sports_&_gaming']
Unique labels in test set: ['arts_&_culture' 'business_&_entrepreneurs' 'daily_life' 'pop_culture'
 'science_&_technology' 'sports_&_gaming']
```

All the labels are included in the training and testing data

In [66]:
```
IGClassifier=DecisionTreeClassifier(criterion='entropy',max_depth=5)
IGClassifier.fit(X_train,Y_train)
IGClassifier
```

Out[66]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=5)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [67]:
```
bow_test =IGClassifier.predict(X_test)
#predictions
len(bow_test)
```

Out[67]:
```
1933
```

In [68]:
```
predBow = pd.DataFrame({'Actual' : Y_test.values.ravel(), 'Predicted' : bow_test})
predBow.head()
```

Out[68]:

	Actual	Predicted
0	sports_&_gaming	sports_&_gaming
1	pop_culture	pop_culture
2	pop_culture	pop_culture
3	business_&_entrepreneurs	pop_culture
4	sports_&_gaming	pop_culture

Decision Tree model

The performance of the mmodel is evaluated - not satisfactory

Feature extraction using TF-IDF is tried now

In [80]:
```
tfidf_vect = TfidfVectorizer()
tfidf_count = tfidf_vect.fit_transform(new_data2['lemm_text'])
```

In [81]:
```
#Scaling

tfidf_scale = StandardScaler(with_mean=False).fit_transform(tfidf_count)
tfidf_scale
```

Out[81]:
```
<6442x17986 sparse matrix of type '<class 'numpy.float64'>'
        with 91030 stored elements in Compressed Sparse Row format>
```

In [82]:
```
#splitting the data

x = tfidf_count
y = new_data2['label_name']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3)
```

In [83]:
```
x_train.shape, x_test.shape
```

Out[83]:
```
((4509, 17986), (1933, 17986))
```

In [84]:
```
y_train.shape, y_test.shape
```

Out[84]:
```
((4509,), (1933,))
```

```
# count the number of mismatches
count = 0
for i in range(0, len(predTfidf)):
    if tfidf_test[i] != y_test.values.ravel()[i]:
        count = count + 1
print("Count of Wrong Prediction : " ,count)
```

```
Count of Wrong Prediction :  1183
```

```
#Converting y_test to an array for confusion Matrix

label_encoder = LabelEncoder()
y_test = label_encoder.fit_transform(y_test)
```

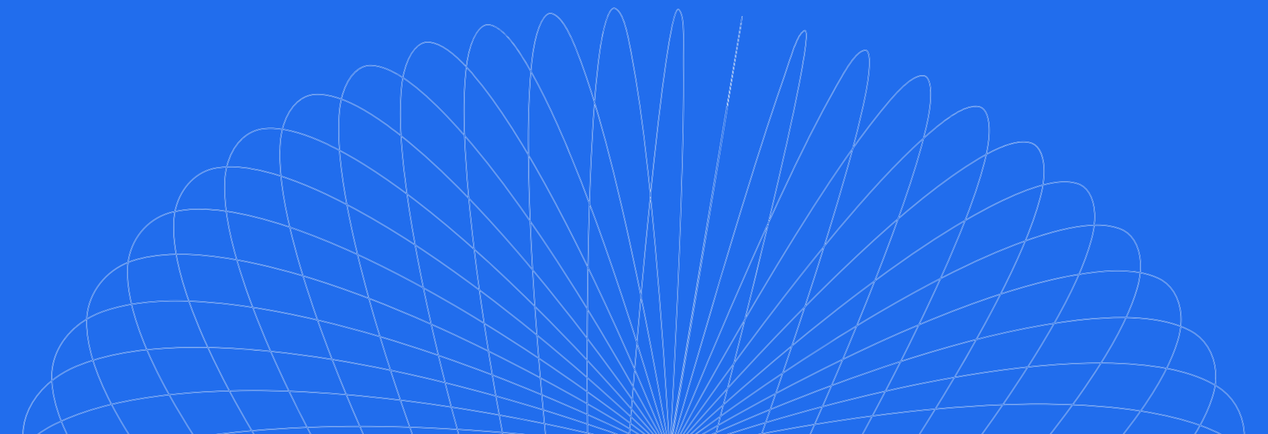
```
#converting predBow to array

predTfidf = np.argmax(predTfidf, axis=1)
```

Evaluating the model

```
print('Classification Report of Decision Tree - 1st approach(TF-IDF)')
print('\n')
print(classification_report(y_test,predTfidf, zero_division = 1))
```

```
Classification Report of Decision Tree - 1st approach(TF-IDF)


              precision    recall  f1-score   support

           0       0.00      0.00      0.00        50
           1       0.21      1.00      0.35        85
           2       1.00      0.00      0.00       266
           3       1.00      0.00      0.00       750
           4       1.00      0.00      0.00       102
```

Other Algorithms are now applied using the training and testing data from B-o-W and TF-IDF



Logistic Regression Model

In [96]:
```
lreg = LogisticRegression(max_iter=5000)
lreg.fit(X_train, Y_train)
```

Out[96]:

LogisticRegression

LogisticRegression(max_iter=5000)

In [97]:
```
#Prediction

predlog = lreg.predict_proba(X_test)

predint = predlog[:,1] >= 0.3
predint = predint.astype(int)
```

In [98]:
```
print('Classification Report of Logistic Regression - 1st approach(BoW)')
print('\n')
print(classification_report(y_test,predint, zero_division = 1))
```

```
Classification Report of Logistic Regression - 1st approach(BoW)


              precision    recall  f1-score   support

           0       0.03      0.98      0.05        53
           1       0.03      0.01      0.02        88
           2       1.00      0.00      0.00       235
           3       1.00      0.00      0.00       752
           4       1.00      0.00      0.00        88
```

Naive Bayes Model

In [108]:
```
#Using BoW data
#Creating a classifier

naive_classifier = MultinomialNB()
```

In [109]:
```
naive_classifier.fit(X_train, Y_train)
```

Out[109]:
```
MultinomialNB
MultinomialNB()
```

In [110]:
```
#Prediction

naive_pred = naive_classifier.predict(X_test)
```

In [111]:
```
Y_test.shape, naive_pred.shape
```

Out[111]:
```
((1933,), (1933,))
```

In [113]:
```
# Y_test11 = Y_test.astype(int)
# naive_pred11 = naive_pred.astype(int)
```

In [114]:
```
#facing issues. This could be solved by using "label" which has integers instead of "label_name" for training model
```

Issue: Use of "label_name" for the training data as the type is "series"
Solution: Use "label" which is integer

Naive Bayes Redone

In [114]:
```
#facing issues. This could be solved by using "label" which has integers instead of "label_name" for training model
```

In [115]:
```
#using the BoW method to split the dataset again. This time, we are using "label"
X1 = bow_scale
Y1 = new_data2['label']

X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1,Y1, test_size = 0.3, stratify=Y, random_state=42)
```

In [116]:
```
naive_classifier.fit(X1_train, Y1_train)
```

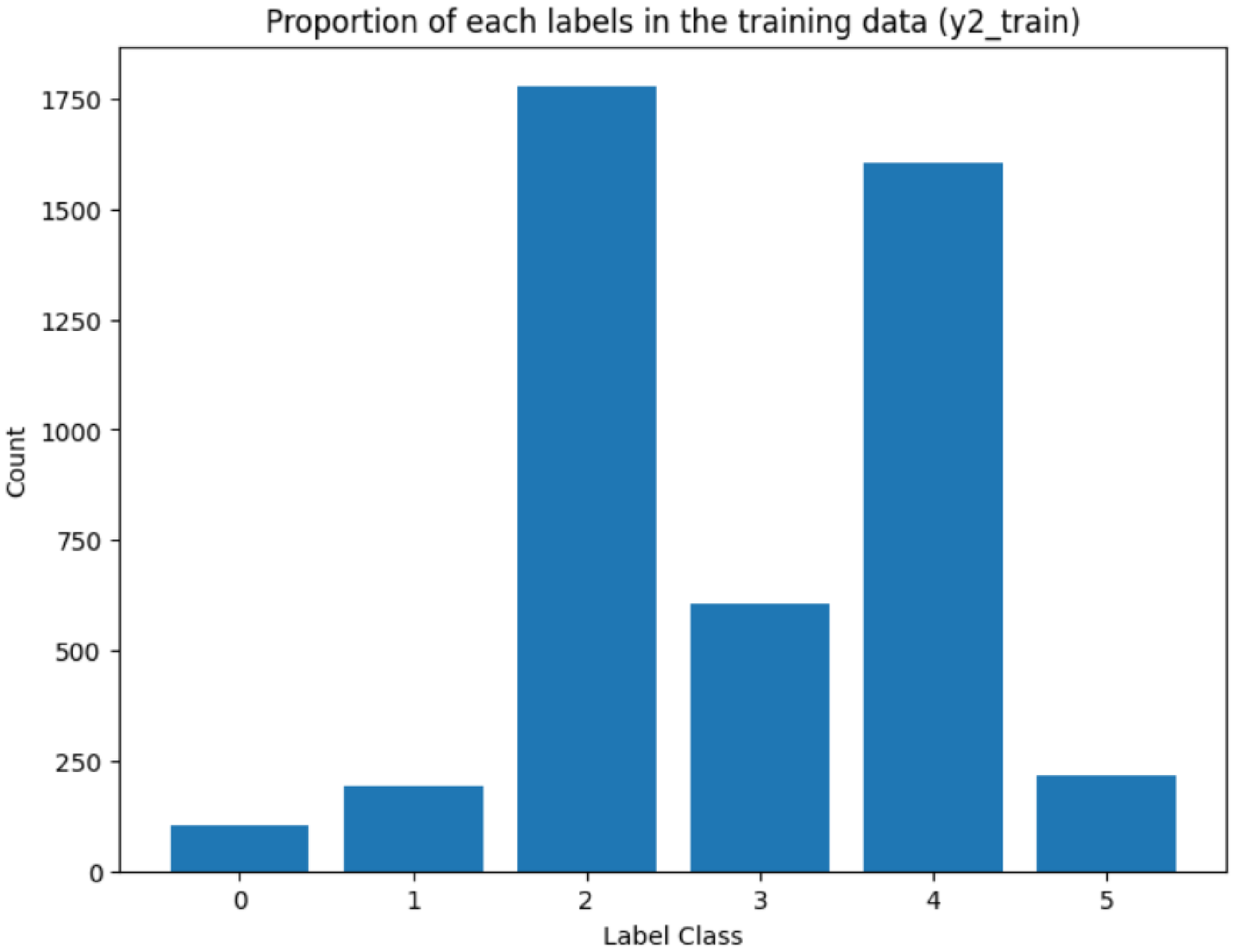
Out[116]:
```
MultinomialNB
MultinomialNB()
```

In [117]:
```
#Prediction

naive_pred1 = naive_classifier.predict(X1_test)
```

In [118]:
```
#evaluating the model
print("Classification report of Naive Bayes- 1st approach (BoW)")
print("\n")
classification_naive1 = classification_report(Y1_test, naive_pred1)
print(classification_naive1)
```

```
Classification report of Naive Bayes- 1st approach (BoW)


              precision    recall  f1-score   support

           0       0.04      0.79      0.08        43
           1       0.28      0.52      0.37        86
```

The models are still not satisfactory.
This could be because of class imbalance in the training data


Proportion of each labels in the training data (y2_train)
Count
1750
1500
1250
1000
750
500
250
0
0
1
2
3
4
5
Label Class

Used SMOTE to balance the data

```
# Create the SMOTE oversampler
smote = SMOTE(random_state=42)

# Apply SMOTE to the training data
x2_train_balanced, y2_train_balanced = smote.fit_resample(x2_train, y2_train)
```

In [131]:
```
class_counts = np.bincount(y2_train_balanced)

class_labels = np.unique(y2_train_balanced)

#visualizing

plt.figure(figsize=(8, 6))
plt.bar(class_labels, class_counts)
plt.xlabel('Label class')
plt.ylabel('Count')
plt.title('Proportion of each label in the training data(y2_train) after SMOTE')
plt.show()
```

Used balanced data in Naive Bayes model and Decision Tree

Out[132]:

MultinomialNB

MultinomialNB()

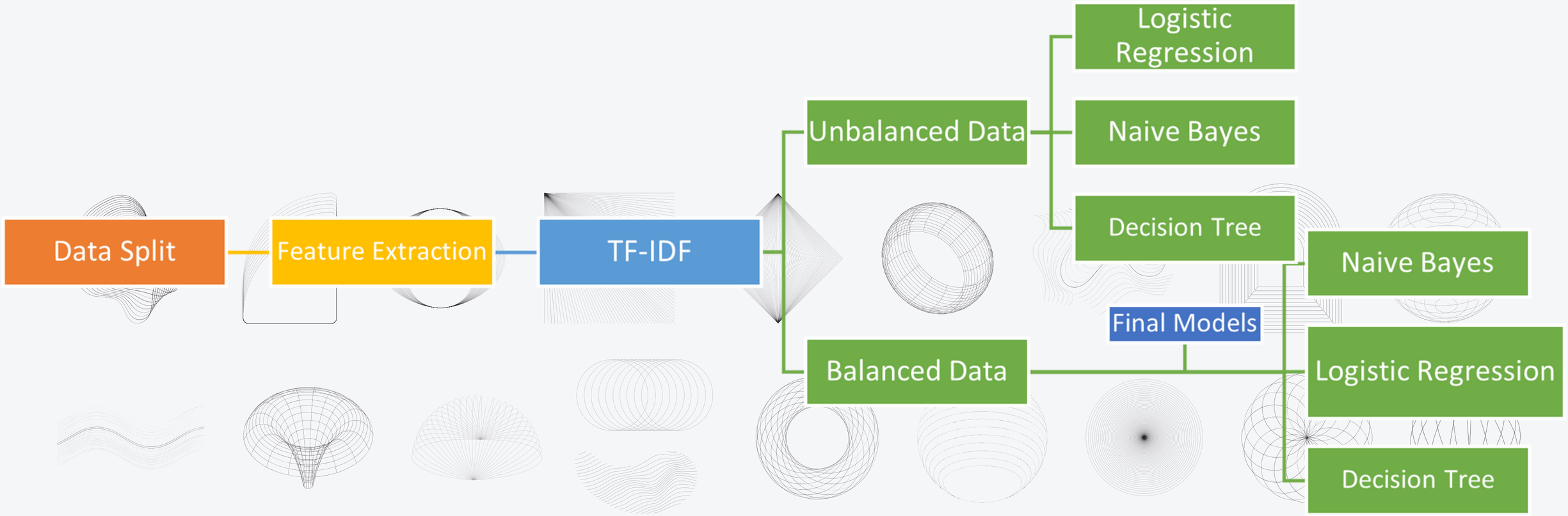
In [133]:
```
naive_pred3 = naive_classifier3.predict(x2_test)
```

In [134]:
```
#evaluating the model
print("Classification report of Naive bayes after SMOTE balancing- 1st approach")
print("\n")
classification_naive3 = classification_report(Y_test, naive_pred3)
print(classification_naive3)
```

```
Classification report of Naive bayes after SMOTE balancing- 1st approach


              precision    recall  f1-score   support

           0       0.01      0.02      0.02        43
           1       0.04      0.05      0.04        86
           2       0.13      0.27      0.17       265
           3       0.40      0.20      0.26       754
           4       0.06      0.39      0.10        98
           5       0.29      0.07      0.11       687

    accuracy                           0.16      1933
   macro avg       0.15      0.17      0.12      1933
weighted avg       0.28      0.16      0.17      1933
```

It is evident that there is some positive effects of balancing in the model.

Approach 2


Data Split
Feature Extraction
TF-IDF
Unbalanced Data
Balanced Data
Logistic Regression
Naive Bayes
Decision Tree
Final Models
Naive Bayes
Logistic Regression
Decision Tree

In [160]:
```
#Splitting the data into X2 and Y2

X2 = new_data2['lemm_text']
Y2 = new_data2['label']

seed = 42
test_size = 0.3

X2_train, X2_test, Y2_train, Y2_test = train_test_split(X2,Y2, test_size = 0.2, random_state = seed, stratify=new_data2['label'])
```

In [161]:
```
print(X2_train.shape, X2_test.shape, Y2_train.shape, Y2_test.shape)
```

```
(5153,) (1289,) (5153,) (1289,)
```

In [162]:
```
#Using the TF-IDF, we are fitting the training and testing data

vectorizer = TfidfVectorizer()
```

In [163]:
```
#fit on the training data
X2_train = vectorizer.fit_transform(X2_train)

#transform the test data
X2_test = vectorizer.transform(X2_test)
```

In [164]:
```
X2_train.shape, X2_test.shape
```

Logistic Regression model in 2nd approach

In [165]:
```
clf = LogisticRegression()
clf.fit(X2_train,Y2_train)
train_predictions = clf.predict(X2_train)
test_predictions = clf.predict(X2_test)
```

In [166]:
```
print('Accuracy Score on training set %.5f' %accuracy_score(Y2_train,train_predictions))
print('Accuracy Score on test set %.5f' %accuracy_score(Y2_test,test_predictions))
```

```
Accuracy Score on training set 0.89016
Accuracy Score on test set 0.77269
```

In [167]:
```
# print('Classification Report Training set')
# print('\n')
# print(classification_report(Y2_train,train_predictions))
```

In [168]:
```
print('Classification Report for Logistic Regression (2nd approach-unbalanced)')
print('\n')
print(classification_report(Y2_test,test_predictions, zero_division = 1))
```

```
Classification Report for Logistic Regression (2nd approach-unbalanced)


              precision    recall  f1-score   support

           0       1.00      0.00      0.00        29
           1       0.91      0.18      0.29        57
           2       0.72      0.92      0.81       503
```

New training data is used and developed three models with Decision Tree, Logistic Regression and Naive bayes

The training data is Balanced again using SMOTE

Balanced data is applied for Naive Bayes and other models

In [186]:
```
X2_train_balanced, Y2_train_balanced = smote.fit_resample(X2_train, Y2_train)
```

In [187]:
```
naive_classifier_balanced = MultinomialNB()
naive_classifier_balanced.fit(X2_train_balanced, Y2_train_balanced)
```

Out[187]:
```
MultinomialNB
MultinomialNB()
```

In [188]:
```
predict_naive_balanced = naive_classifier_balanced.predict(X2_test)
```

In [189]:
```
#evaluating the model
print('Classification Report for Naive Bayes with balanced -2nd approach')
print('\n')
print(classification_report(Y2_test, predict_naive_balanced, zero_division=1))
```

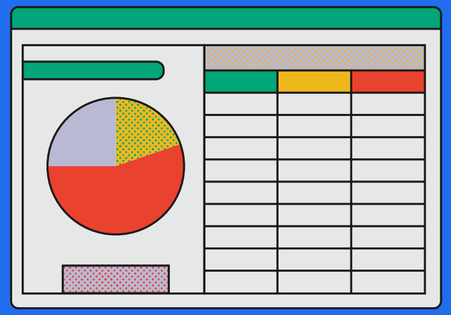
```
Classification Report for Naive Bayes with balanced -2nd approach


              precision    recall  f1-score   support

           0       0.33      0.45      0.38        29
           1       0.41      0.58      0.48        57
           2       0.90      0.77      0.83       503
           3       0.52      0.61      0.56       177
           4       0.92      0.88      0.90       458
           5       0.48      0.75      0.58        65
```

Significant Improvement of performance in the models.

Evaluation

Performance Metrics:

- **Accuracy**
- **Precision**
- **Recall**
- **F1 score**



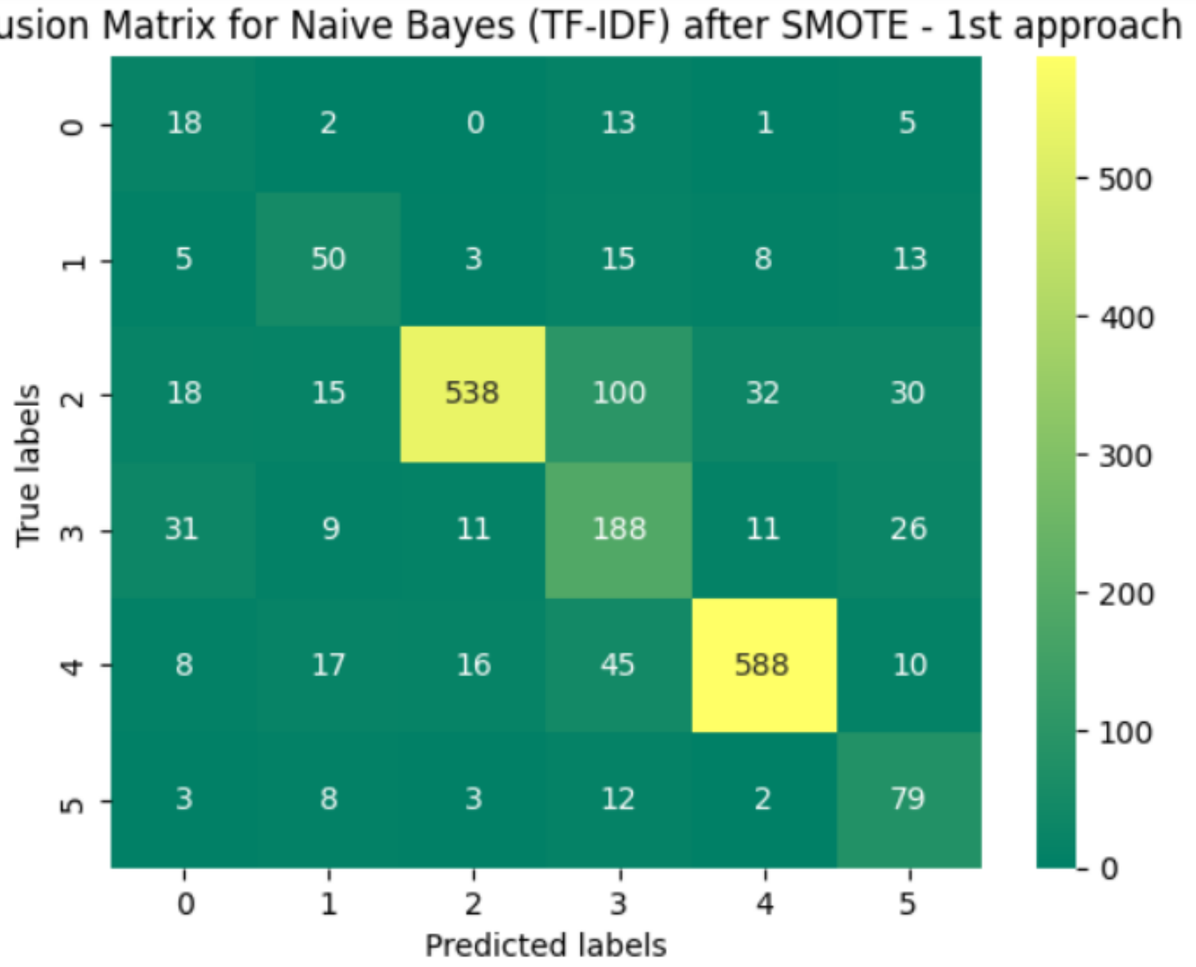
Tables are made with the performance metrics of each models for comparison

Approach 1

Algorithm	Accuracy %		Precision (macro average)		Recall (macro average)		F1 Score	
	B-O-W	TF-IDF	B-O-W	TF-IDF	B-O-W	TF-IDF	B-O-W	TF-IDF
Decision Tree	4%	4%	0.70	0.70	0.17	0.17	0.06	0.06
Logistic Regression	3%	3%	0.68	0.84	0.17	0.17	0.01	0.02
Naïve Bayes	40%	10%	0.48	0.62	0.49	0.17	0.40	0.06

Classification report of Naive bayes after SMOTE balancing- 1st approach

	precision	recall	f1-score	support
0	0.01	0.02	0.02	43
1	0.04	0.05	0.04	
2	0.13	0.27	0.17	
3	0.40	0.20	0.26	
4	0.06	0.39	0.10	
5	0.29	0.07	0.11	
accuracy			0.16	
macro avg	0.15	0.17	0.12	
weighted avg	0.28	0.16	0.17	



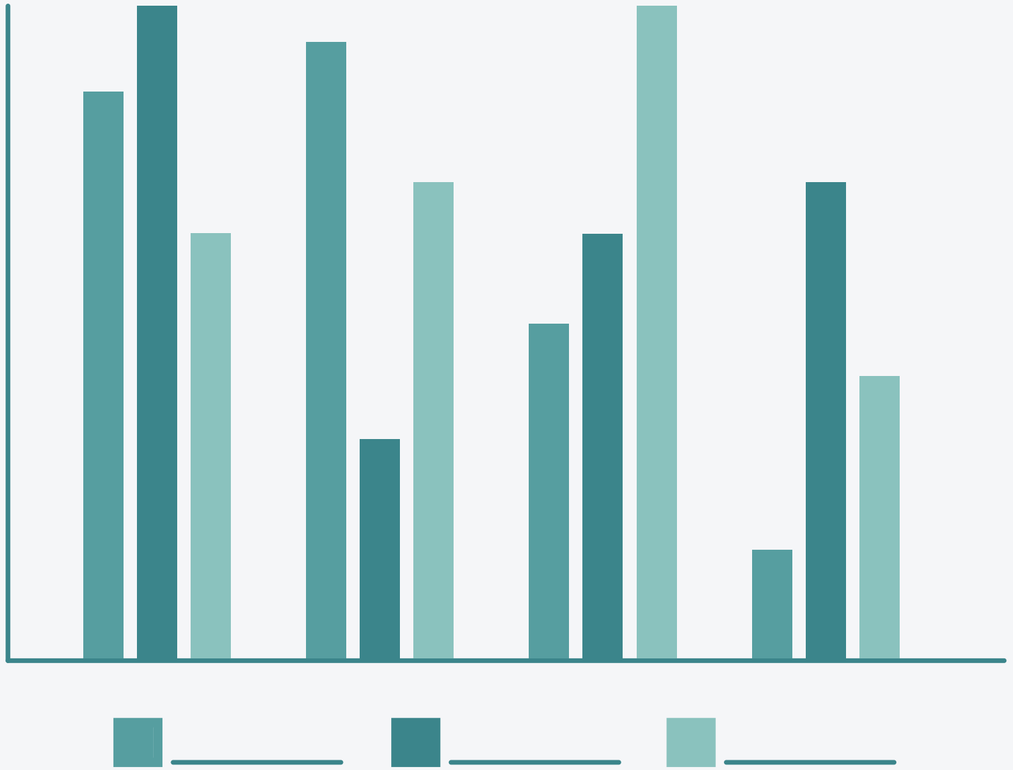
In [154]:
```
#evaluating the model
print("Classification Report of Decision Tree after SMOTE - 1st approach")
print("\n")
classification_decisiontree = classification_report(y2_test_en, dc_balanced_en, zero_division=1)
print(classification_decisiontree)
```

```
Classification Report of Decision Tree after SMOTE - 1st approach


              precision    recall  f1-score   support

           0       0.00      0.03      0.00        39
           1       0.03      0.02      0.02        94
           2       1.00      0.00      0.00       733
           3       1.00      0.00      0.00       276
           4       1.00      0.00      0.00       684
           5       1.00      0.00      0.00       107

    accuracy                           0.00      1933
   macro avg       0.67      0.01      0.00      1933
weighted avg       0.93      0.00      0.00      1933
```

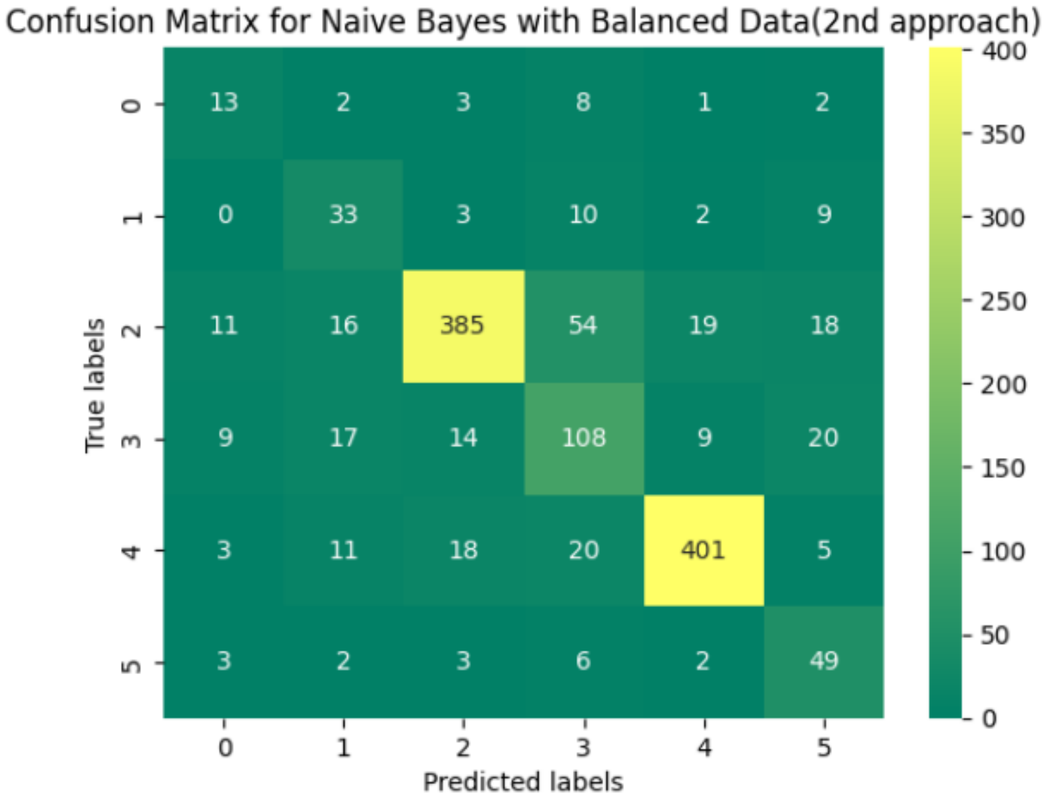
Approach 2

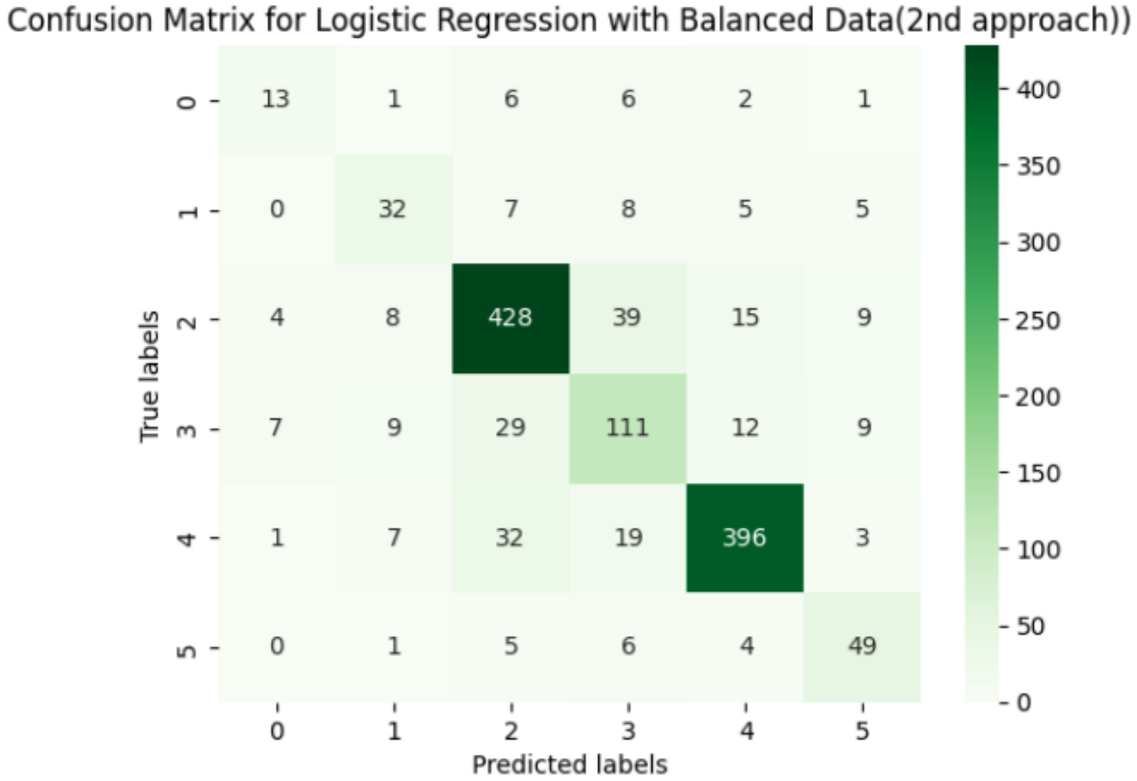
Algorithm	Accuracy %		Precision (macro average)		Recall (macro average)		F1 Score	
	Unbalanced	Balanced	unbalanced	balanced	unbalanced	balanced	unbalanced	balanced
Decision Tree	49%	0	0.84	0.67	0.23	0	0.21	0
Logistic Regression	77	80	0.86	0.68	0.47	0.68	0.51	0.68
Naïve Bayes	71	77	0.90	0.59	0.32	0.67	0.29	0.77

Comparison of best performing models



Naive Bayes model vs Logistic Regression model





Classification Report for Naive Bayes with balanced -2nd approach

	precision	recall	f1-score	support
0	0.33	0.45	0.38	29
1	0.41	0.58	0.48	57
2	0.90	0.77	0.83	503
3	0.52	0.61	0.56	177
4	0.92	0.88	0.90	458
5	0.48	0.75	0.58	65
accuracy			0.77	1289
macro avg	0.59	0.67	0.62	1289
weighted avg	0.80	0.77	0.78	1289

Classification Report for Logistic Regression with balanced - 2nd approach

	precision	recall	f1-score	support
0	0.52	0.45	0.48	29
1	0.55	0.56	0.56	57
2	0.84	0.85	0.85	503
3	0.59	0.63	0.61	177
4	0.91	0.86	0.89	458
5	0.64	0.75	0.70	65
accuracy			0.80	1289
macro avg	0.68	0.68	0.68	1289
weighted avg	0.80	0.80	0.80	1289

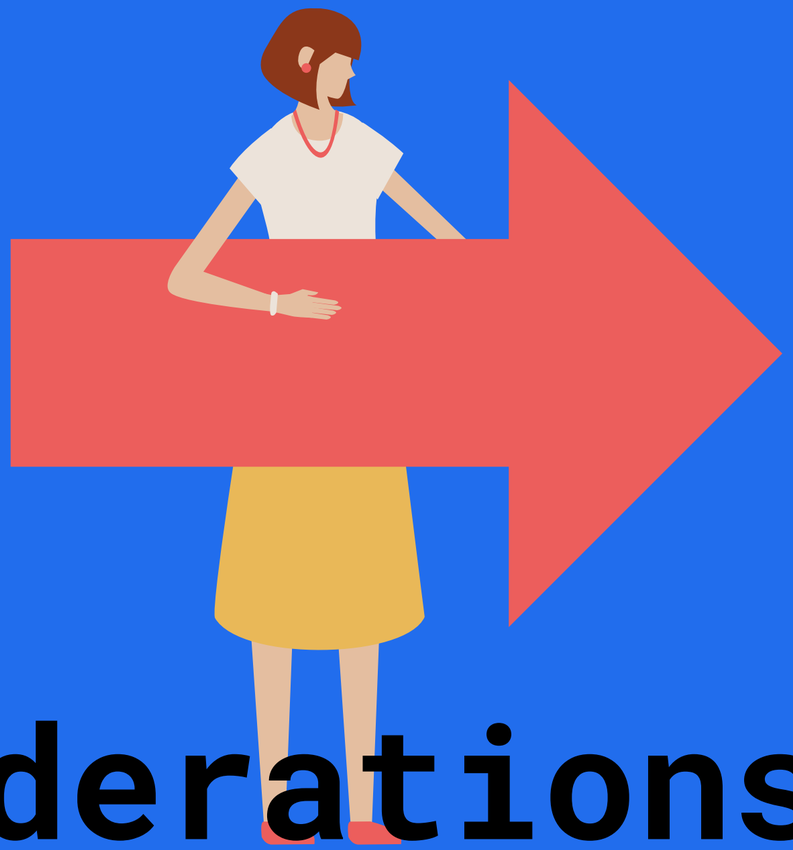
Naive Bayes:
Accuracy: 77%
F1 Score: 0.62

Logistic Regression:
Accuracy: 80&
F1 Score: 0.68

LR: Logistic Regression	Precision		Recall		F Score	
NB: Naïve Bayes	LR	NB	LR	NB	LR	NB
0	0.52	0.33	0.45	0.45	0.48	0.38
1	0.55	0.41	0.56	0.58	0.56	0.48
2	0.84	0.90	0.85	0.77	0.85	0.83
3	0.59	0.52	0.63	0.61	0.61	0.56
4	0.91	0.92	0.86	0.88	0.89	0.90
5	0.64	0.48	0.75	0.75	0.70	0.58

Future Works

Things to improve



Considerations for future work

Thank You