CAPSTONE PROJECT

Upgrad AI ML Bootcamp

Project Title: Intensity Analysis (Using NLP and Python)

Submitted by

THAVASEELAN MOHAN

MAY 2024

Table of Contents

Project Summary

The main objective of the project is to develop an intelligent system using Natural language processing and Python programming to predict the intensity of the data using different machine learning models.

Overall approach

The raw data is unzipped, and all the csv files are converted into data frames to feed into different machine learning models. In data cleansing and pre-processing process, the duplicate records are removed before processing the data. Using NLTK, tokenization and Lemmatization libraries, all the stop words are removed, tokenized the data, lemmatized the data. The final data frame is exported into csv file. Data is visualized using different bar charts, pie chart, and word cloud visualization for each file.

Future enhancements

Implementation with traditional machine learning models produces maximum accuracy and metrics of 71%. Future implementation of Feed forward neural network with different Tensorflow hub models like Google 2000B corpus NNLM 50/2, 128 with Normalization will produce better accuracy with less epochs.

Data Acquisition

Libraries used

Below are the libraries used to predict Intensity data analysis.

Numpy – Used to perform complex numerical calculations.

Pandas – Used to perform data manipulation and analysis.

Re – Regular expression – used to extract text from large corpus based on business needs.

Matplotlib – used to visualize the data in different charts like bar, boxplot.

Seaborn – Extended version of matplotlib. To visualize violin plot and advanced visualization.

Nltk – Natural Language Toolkit – NLP important libraries like Tokenize, stop words and punctuations are available in this library. Lemmatization is used to convert the words to their basic root form.

Wordcloud – Used to visualize the words in the dataset.

Scikit learn – A powerful library used in machine learning to perform important functions like splitting the dataset, visualize the metrics.

Data Collection

The dataset is provided in the compressed file with three different csv files like happiness, angriness and sadness. The objective is to extract the data and process the data using python programming.

Below are the commands used to unzip the folder and extracts the contents for processing.

Before processing the data, the source file has been placed in github repository as public.

```
]: # URL of the zip file
url = 'https://github.com/ThavaseelanMohan/AIMLTraining/raw/main/Intensity_data.zip'

# Define the filename for the downloaded zip file
zip_filename = 'Intensity_data.zip'

# Define the directory where the files will be extracted
extracted_dir = 'Intensity_data'
```

```
]: # Download the zip file
urllib.request.urlretrieve(url, zip_filename)
print("Downloaded successfully.")
```

```
Downloaded successfully.
```

```
]: # Extract the contents of the zip file
with zipfile.ZipFile(zip_filename, 'r') as zip_ref:
    zip_ref.extractall(extracted_dir)
print("Extracted successfully.")
```

```
Extracted successfully.
```

```
]: # Delete the zip file
os.remove(zip_filename)
print("Zip file deleted.")
```

```
Zip file deleted.
```

Extracting the contents in the file

Extracting all the csv files from the local repository for ease processing.

```
In [15]: #List all the csv files
def myfunction(Intensity_data_files):
  print(Intensity_data_files)
file_path = '/Users/thavaseelan/Project/Intensity_data'

for file in os.listdir(file_path):
  if file.endswith(".csv"):
    myfunction(file)
```

```
sadness.csv
happiness.csv
angriness.csv
```

```
In [16]: df = pd.concat(map(pd.read_csv, glob.glob('/Users/thavaseelan/Project/Intensity_data/*.csv')))
```

Exploratory Data Analysis (EDA)

Based on the data, there are three different csv files namely, happiness.csv, angriness.csv and sadness.csv. There are two fields in each file namely, content and intensity. Content field contains the real tweets for processing. Intensity is the label. For the data, this is a supervised model because labels are available to classify the contents of the file. In total, there are 2039 lines with 2 columns each.

In [17]:
```
print(df)
```

```
                                               content  intensity
0    Never hurt people who love you a lot, because ...    sadness
1    Don't expect me to tell you what you did wrong...    sadness
2    I preferred walking away than fighting for you...    sadness
3    Moving forward in life isn't the hard part, it...    sadness
4    Never cry for anyone in your life, because tho...    sadness
..                                                 ...        ...
691            Is getting close to just walking away?  angriness
692  Did you ever just want to slap the stupid out ...  angriness
693  Is singing, if you're angry & you know it punc...  angriness
694  Telling someone to calm down just makes them s...  angriness
695  Your mistake is letting me go, my mistake was ...  angriness

[2039 rows x 2 columns]
```

In [18]:
```
df.shape
```

Out[18]:
```
(2039, 2)
```

In [19]:
```
df.columns
```

Out[19]:
```
Index(['content', 'intensity'], dtype='object')
```

In [20]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2039 entries, 0 to 695
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   content    2039 non-null   object
 1   intensity  2039 non-null   object
dtypes: object(2)
```

In [21]:
```
df.describe()
```

Out[21]:

	content	intensity
count	2039	2039
unique	1586	3
top	['You Hurt Me But I Still Love You.', 'True Lo...	happiness
freq	134	708

Data Imputation

NULL values check

Based on the dataset, there are no NULL values in any of the csv files.

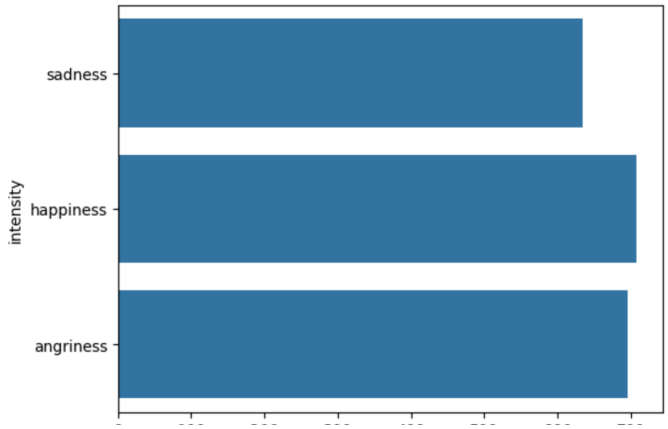
In [25]:
```
# Finding null values
df.isnull().sum()
```

Out[25]:
```
content      0
intensity    0
dtype: int64
```

Duplicate Check

447 duplicate contents are available in the dataset.

```
In [22]: df.duplicated().sum()
Out[22]: 447

In [23]: df['intensity'].unique()
Out[23]: array(['sadness', 'happiness', 'angriness'], dtype=object)

In [24]: sns.countplot(df['intensity'])
Out[24]: <Axes: xlabel='count', ylabel='intensity'>
```



Data Analysis

Visualizing the data for each file in data frame and find the duplicate records in each files using python script.

In [26]:
```
df[df['intensity']=='angriness']
```

Out[26]:

	content	intensity
0	Sometimes I'm not angry, I'm hurt and there's ...	angriness
1	Not available for busy people☺	angriness
2	I do not exist to impress the world. I exist t...	angriness
3	Everything is getting expensive except some pe...	angriness
4	My phone screen is brighter than my future 😕	angriness
...	...	...
691	Is getting close to just walking away?	angriness
692	Did you ever just want to slap the stupid out ...	angriness
693	Is singing, if you're angry & you know it punc...	angriness
694	Telling someone to calm down just makes them s...	angriness
695	Your mistake is letting me go, my mistake was ...	angriness

696 rows × 2 columns

In [27]:
```
df[df['intensity']=='happiness']
```

Out[27]:

	content	intensity
0	Wants to know how the hell I can remember word...	happiness
1	Love is a long sweet dream & marriage is an al...	happiness
2	The world could be amazing when you are slight...	happiness
3	My secret talent is getting tired without doin...	happiness
4	Khatarnaak Whatsapp Status Ever… Can\'t talk, ...	happiness
...	...	...
703	If I know what love is, it is because of you.	happiness
704	The spaces between your fingers are meant to b...	happiness
705	In you i H've Found the love of my life and my...	happiness
706	The magic of first love is our ignorance that ...	happiness
707	Love can't be found Where it doesn't exist ..	happiness

708 rows × 2 columns

```
In [28]: df[df['intensity']=='sadness']
```

Out[28]:

	content	intensity
0	Never hurt people who love you a lot, because ...	sadness
1	Don't expect me to tell you what you did wrong...	sadness
2	I preferred walking away than fighting for you...	sadness
3	Moving forward in life isn't the hard part, it...	sadness
4	Never cry for anyone in your life, because tho...	sadness
...	...	...
630	Stop crying over yesterday and start smiling f...	sadness
631	An Eye with Dust 'n A Heart with Trust Always ...	sadness
632	Tears come from the heart and not from the brain.	sadness
633	Sometimes you have to hold your head up high, ...	sadness
634	Instead of wiping your tears, wipe away the pe...	sadness

635 rows × 2 columns

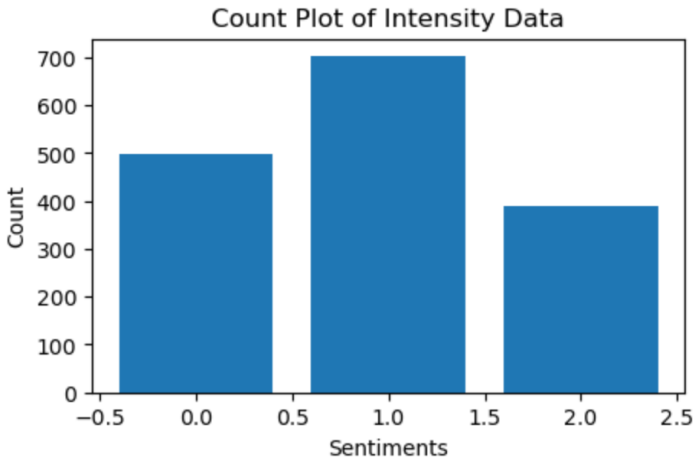
```
In [29]: df[df['intensity']=='angriness'].duplicated().sum()
Out[29]: 198

In [30]: df[df['intensity']=='happiness'].duplicated().sum()
Out[30]: 4

In [31]: df[df['intensity']=='sadness'].duplicated().sum()
Out[31]: 245
```

Data Visualization in matplotlib

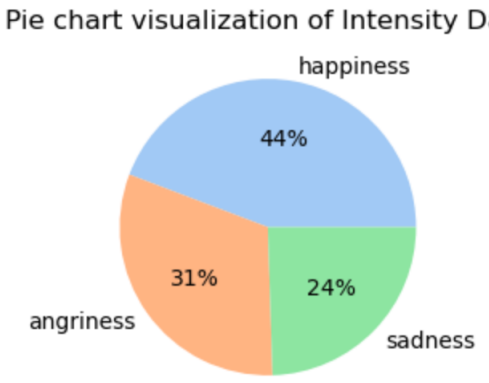
Content is visualized using bar chart and pie chart.

In [97]:
```
sentiment_count = df['intensity'].value_counts()
print(sentiment_count)
plt.figure(figsize=(5, 3))
plt.bar(sentiment_count.index, sentiment_count)
plt.title('Count Plot of Intensity Data')
plt.xlabel('Sentiments')
plt.ylabel('Count')
plt.show()
```

```
intensity
1    704
0    498
2    390
Name: count, dtype: int64
```



In [98]:
```
#Define plot size
plt.figure(figsize=[3,3])

#Define column to use
data = df["intensity"].value_counts(normalize=True)

#Define labels
labels = ["happiness","angriness","sadness"]

#Define color palette
colors = sns.color_palette('pastel')

#Create pie chart
plt.pie(data,labels=labels,colors=colors, autopct='%.0f%%')
plt.title("Pie chart visualization of Intensity Data")
plt.show()
```



Data Cleansing

Removing all the duplicates and clean the dataset for further processing using drop_duplicates and reset the index columns.

```
In [34]: df = df.drop_duplicates().reset_index(drop=True)
```

```
In [35]: df
```

Out[35]:

	content	intensity
0	Never hurt people who love you a lot, because ...	sadness
1	Don't expect me to tell you what you did wrong...	sadness
2	I preferred walking away than fighting for you...	sadness
3	Moving forward in life isn't the hard part, it...	sadness
4	Never cry for anyone in your life, because tho...	sadness
...	...	...
1587	The embarrassment after the anger is the bigge...	angriness
1588	The strong man is not the good wrestler; the s...	angriness
1589	A man is about as big as the things that make ...	angriness
1590	Is singing, if you're angry & you know it punc...	angriness
1591	Telling someone to calm down just makes them s...	angriness

1592 rows × 2 columns

```
In [36]: df.shape
```

```
Out[36]: (1592, 2)
```

```
In [37]: df.describe()
```

Out[37]:

	content	intensity
count	1592	1592
unique	1586	3
top	There is no excuse for cheating in a relations...	happiness
freq	2	704

```
In [38]: df[df['intensity']=='angriness'].duplicated().sum()
```

```
Out[38]: 0
```

```
In [39]: df[df['intensity']=='happiness'].duplicated().sum()
```

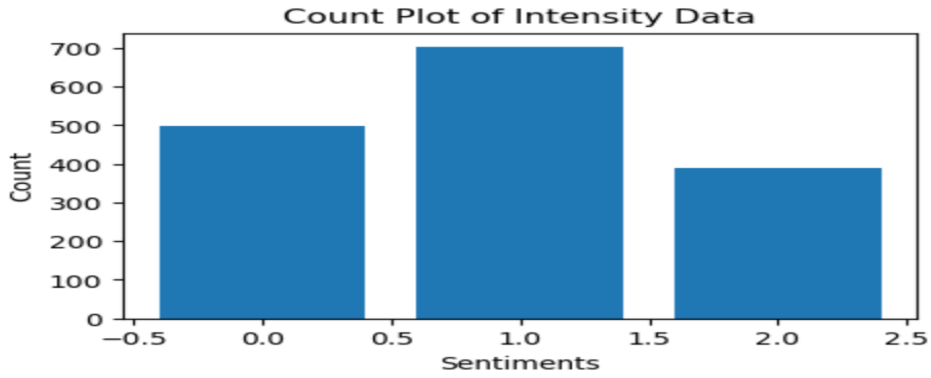
```
Out[39]: 0
```

```
In [40]: df[df['intensity']=='sadness'].duplicated().sum()
```

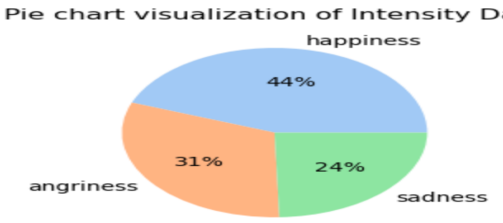
```
Out[40]: 0
```

In [99]:
```
sentiment_count = df['intensity'].value_counts()
print(sentiment_count)
plt.figure(figsize=(5, 3))
plt.bar(sentiment_count.index, sentiment_count)
plt.title('Count Plot of Intensity Data')
plt.xlabel('Sentiments')
plt.ylabel('Count')
plt.show()
```

```
intensity
1    704
0    498
2    390
Name: count, dtype: int64
```



In [100]:
```
#Define plot size
plt.figure(figsize=[3,3])

#Define column to use
data = df["intensity"].value_counts(normalize=True)

#Define labels
labels = ["happiness","angriness","sadness"]

#Define color palette
colors = sns.color_palette('pastel')

#Create pie chart
plt.pie(data,labels=labels,colors=colors, autopct='%.0f%%')
plt.title("Pie chart visualization of Intensity Data")
plt.show()
```



In [43]:
```
df
```

Out[43]:

	content	intensity
0	Never hurt people who love you a lot, because ...	sadness
1	Don't expect me to tell you what you did wrong...	sadness
2	I preferred walking away than fighting for you...	sadness
3	Moving forward in life isn't the hard part, it...	sadness
4	Never cry for anyone in your life, because tho...	sadness
...	...	...
1587	The embarrassment after the anger is the bigge...	angriness
1588	The strong man is not the good wrestler; the s...	angriness
1589	A man is about as big as the things that make ...	angriness
1590	Is singing, if you're angry & you know it punc...	angriness
1591	Telling someone to calm down just makes them s...	angriness

1592 rows × 2 columns

Data Pre-Processing

Before training the model, it is necessary to explore the data which is important in natural language processing.

Step 1: Converting the dataframe using contractions, tokenize and lemmatization with stop words.

```
def clean_text_with_sw(text):
        text = str(text).lower() # converting to lower
        text = " ".join([contractions.fix(expanded_word) for expanded_word in text.split()]) #Removing contractions
        text = re.sub('\[.*?\]', '', text) # Removing text in square brackets
        text = re.sub('https?://\S+|www\.\S+', '', text) #Removing links
        text = re.sub('[^a-zA-Z ]', '', text)
        text = re.sub('<.*?>+', '', text)
        text = re.sub('[%s]' % re.escape(punctuation), '', text) #Removing Punctuations
        text = re.sub('\w*\d\w*', '', text) #Removing words containing numbers
        text = re.sub('\b\w{1,3}\b', '', text) #Any words less than 3 characters are replaced by blanks
        tokenized_text = word_tokenize(str(text)) # Tokenizing text
        filtered_text = [word for word in tokenized_text if word not in stopwords.words('english')] #Removing stopwo
        lemmatization = [lemmatizer.lemmatize(word) for word in filtered_text] #Lemmatization
        lemmatization = ' '.join(lemmatization)
        #lemmatization = ' '.join(words for words in lemmatization.split() if len(words)>2)
        return text
```

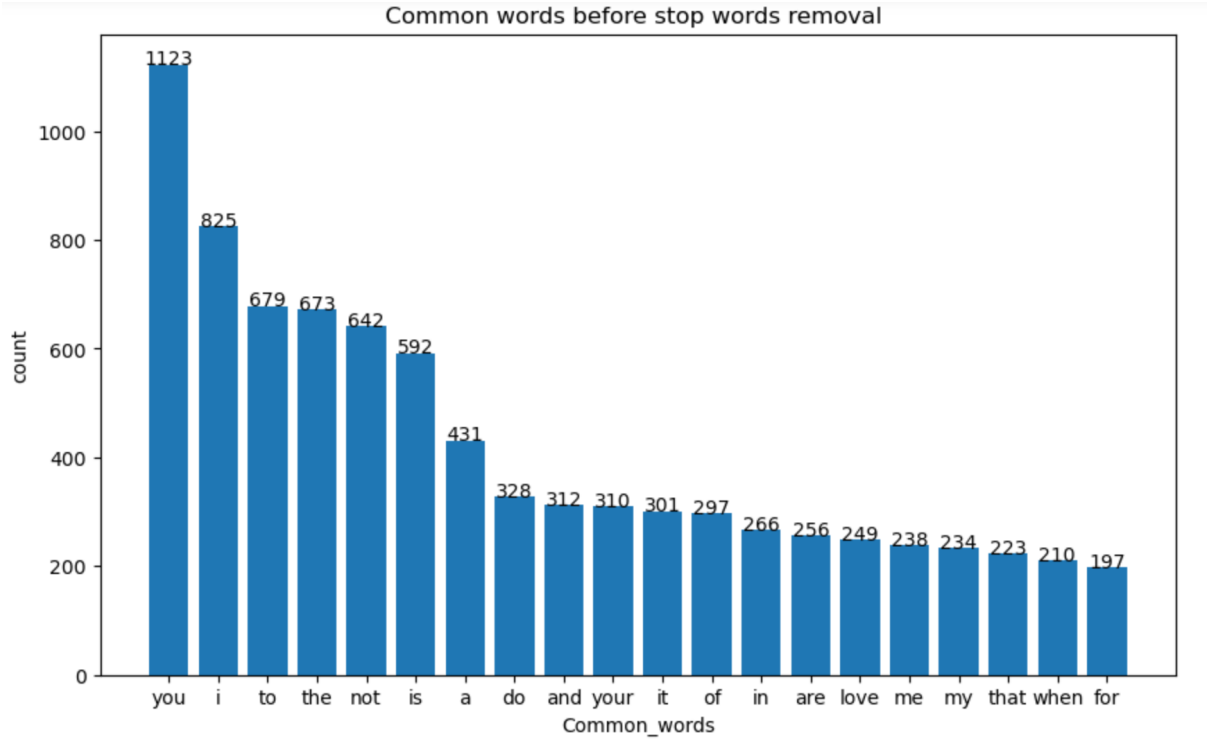
```
# apply clean text fuction on each contents in the training dataset
df['content_with_sw'] = df['content'].apply(lambda x:clean_text_with_sw(x))

df.head()
```

	content	intensity	content_with_sw
0	Never hurt people who love you a lot, because ...	sadness	never hurt people who love you a lot because t...
1	Don't expect me to tell you what you did wrong...	sadness	do not expect me to tell you what you did wron...
2	I preferred walking away than fighting for you...	sadness	i preferred walking away than fighting for you...
3	Moving forward in life isn't the hard part, it...	sadness	moving forward in life is not the hard part it...
4	Never cry for anyone in your life, because tho...	sadness	never cry for anyone in your life because thos...

```
df['word_list'] = df['content_with_sw'].apply(lambda x:word_tokenize(str(x)))
df.head()
```

	content	intensity	content_with_sw	word_list
0	Never hurt people who love you a lot, because ...	sadness	never hurt people who love you a lot because t...	[never, hurt, people, who, love, you, a, lot, ...
1	Don't expect me to tell you what you did wrong...	sadness	do not expect me to tell you what you did wron...	[do, not, expect, me, to, tell, you, what, you...
2	I preferred walking away than fighting for you...	sadness	i preferred walking away than fighting for you...	[i, preferred, walking, away, than, fighting, ...
3	Moving forward in life isn't the hard part, it...	sadness	moving forward in life is not the hard part it...	[moving, forward, in, life, is, not, the, hard...
4	Never cry for anyone in your life, because tho...	sadness	never cry for anyone in your life because thos...	[never, cry, for, anyone, in, your, life, beca...

```
top = Counter([item for sublist in df['word_list'] for item in sublist])
temp_df_before_cleansing = pd.DataFrame(top.most_common(20))
temp_df_before_cleansing.columns = ['Common_words','count']
temp_df_before_cleansing.style.background_gradient(cmap = 'Blues')
def addlabels(x,y):
  for i in range(len(x)):
    plt.text(i, y[i], y[i], ha = 'center')
x = temp_df_before_cleansing['Common_words']
y = temp_df_before_cleansing['count']
plt.figure(figsize = (10,6))
plt.title( " Common words before stop words removal  ")
plt.xlabel("Common_words")
plt.ylabel("count")
plt.bar(x,y)
addlabels(x, y)
plt.show()
plt.figure(figsize=(5, 3))
```



Data cleansing and removing stop words

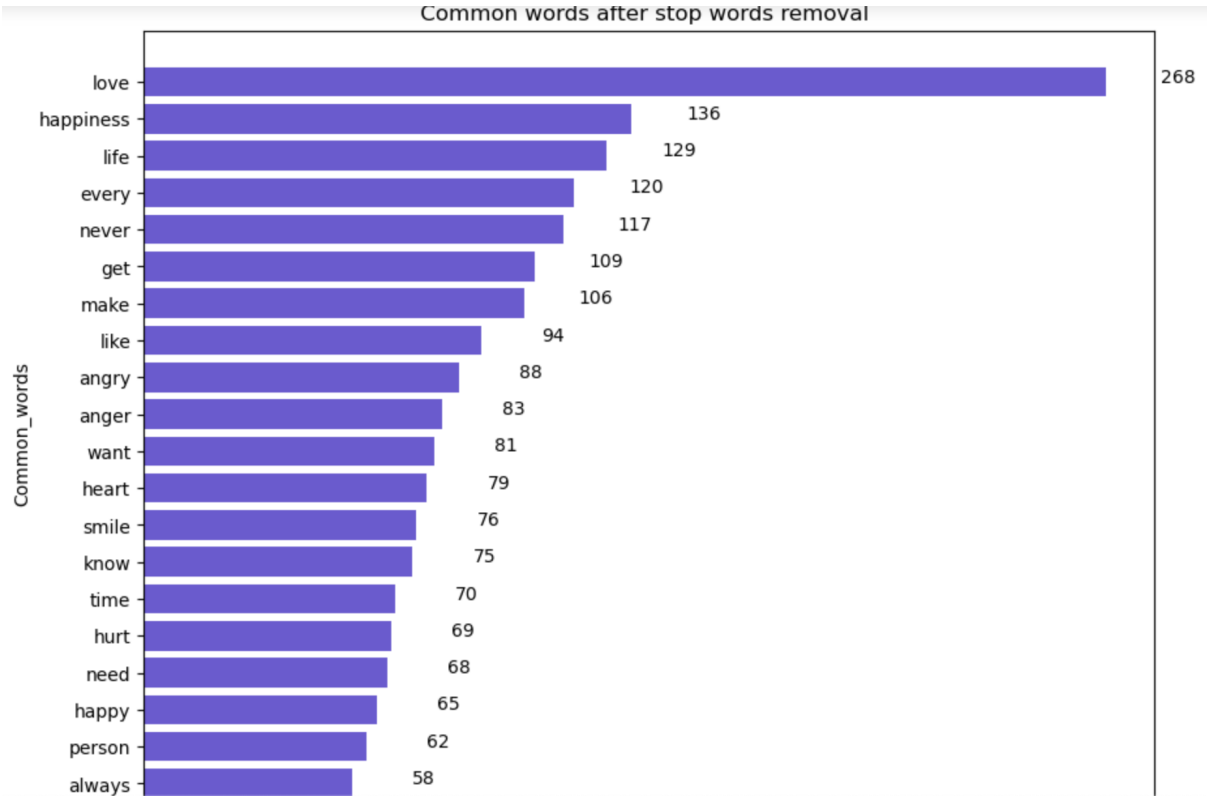
This step involves in removing stop words, removing web links, replacing all the unused words and applying lemmatization technique.

```
def Final_Cleansed_File(text):

  text = str(text).lower() # converting to lower
  text = " ".join([contractions.fix(expanded_word) for expanded_word in text.split()]) #Removing contractions
  text = re.sub('\[.*?\]', '', text) # Removing text in square brackets
  text = re.sub('https?://\S+|www\.\S+', '', text) #Removing links
  text = re.sub('[^a-zA-Z ]', '', text)
  text = re.sub('<.*?>+', '', text)
  text = re.sub('[%s]' % re.escape(punctuation), '', text) #Removing Punctuations
  text = re.sub('\w*\d\w*', '', text) #Removing words containing numbers
  text = re.sub('\b\w{1,3}\b', '', text) #Any words less than 3 characters are replaced by blanks
  text = ' '.join(words for words in text.split() if len(words)>2)
  text = re.sub("than|you|can|the|who|for|anyone|because|those|what|our|where|with|but|will|are|and|when|that|have|j
  tokenized_text = word_tokenize(str(text)) # Tokenizing text
  filtered_text = [word for word in tokenized_text if word not in stopwords.words('english')] #Removing stopwords
  lemmatization = [lemmatizer.lemmatize(word) for word in filtered_text] #Lemmatization
  lemmatization = ' '.join(lemmatization)
  text = re.sub("than|you|can|the|who|for|anyone|because|those|what|our|where|with|but|will|are|and|when|that|have|j
  text = ' '.join(words for words in lemmatization.split() if len(words)>2)
  return text
```

```
# apply clean text fuction on each twitte in the training dataset
df['Final_content'] = df['content'].apply(lambda x:Final_Cleansed_File(x))

df.head()
```

	content	intensity	content_with_sw	word_list	Final_content
0	Never hurt people who love you a lot, because ...	sadness	never hurt people who love you a lot because t...	[never, hurt, people, who, love, you, a, lot, ...	never hurt love lot hurt back probably choice ...
1	Don't expect me to tell you what you did wrong...	sadness	do not expect me to tell you what you did wron...	[do, not, expect, me, to, tell, you, what, you...	expect wrong figure ready correct itcos know m...
2	I preferred walking away than fighting for you...	sadness	i preferred walking away than fighting for you...	[i, preferred, walking, away, than, fighting, ...	preferred walking away fighting worth fighting...

```
top = Counter([item for sublist in df['Final_word_list'] for item in sublist])
temp_df_after_cleansing = pd.DataFrame(top.most_common(20))
temp_df_after_cleansing.columns = ['Common_words','count']
temp_df_after_cleansing.style.background_gradient(cmap = 'Blues')

x = temp_df_after_cleansing['Common_words']
y = temp_df_after_cleansing['count']
plt.figure(figsize = (10,8))
#indexes = np.arange(len(y))
plt.title( " Common words after stop words removal  ")
plt.xlabel("Count")
plt.ylabel("Common_words")
plt.barh( x,y, color = 'slateblue')
plt.gca().invert_yaxis()
#addlabels(x, y)
for i, v in enumerate(y):
  plt.text(v + 20, i, str(v), ha='center')
plt.show()
```



Word Cloud visualization

Below is the word cloud visualization for different data frames.

Word cloud visualization for raw data

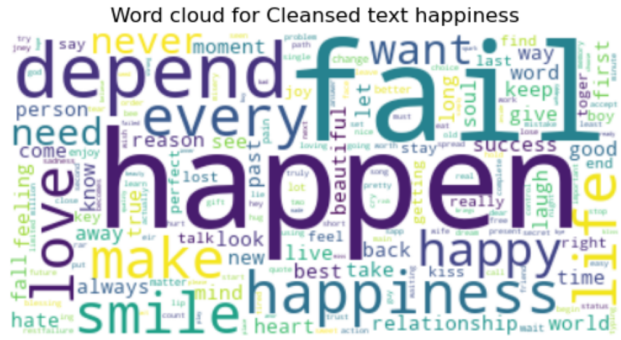
```
top = Counter([item for sublist in df['word_list'] for item in sublist])
temp_df_before_cleansing = pd.DataFrame(top.most_common(400))
temp_df_before_cleansing.columns = ['Common_words','count']
temp_df_before_cleansing.style.background_gradient(cmap = 'Blues')

wordcloud_text = ','.join(list(temp_df_before_cleansing['Common_words'].values))
wordcloud = WordCloud(background_color="white",max_words=5000, contour_width=3).generate(str(wordcloud_text))
plt.imshow(wordcloud, interpolation="bilinear")
plt.title("Word cloud for content")
plt.axis("off")
plt.show()
```

Word cloud visualization for Cleansed text

```
top = Counter([item for sublist in df['Final_word_list'] for item in sublist])
temp_df_after_cleansing = pd.DataFrame(top.most_common(400))
temp_df_after_cleansing.columns = ['Common_words','count']
temp_df_after_cleansing.style.background_gradient(cmap = 'Blues')

wordcloud_text = ','.join(list(temp_df_after_cleansing['Common_words'].values))
wordcloud = WordCloud(background_color="white",max_words=5000, contour_width=3).generate(str(wordcloud_text))
plt.imshow(wordcloud, interpolation="bilinear")
plt.title("Word cloud for Cleansed text")
plt.axis("off")
plt.show()
```

Word Cloud visualization for happiness

```
df1 = df[['intensity','Final_word_list']]
df1 = df1[df1['intensity']=='happiness']
top = Counter([item for sublist in df1['Final_word_list'] for item in sublist])
temp_df_after_cleansing = pd.DataFrame(top.most_common(400))
temp_df_after_cleansing.columns = ['Common_words','count']
temp_df_after_cleansing.style.background_gradient(cmap = 'Blues')

wordcloud_text = ','.join(list(temp_df_after_cleansing['Common_words'].values))
wordcloud = WordCloud(background_color="white",max_words=5000, contour_width=3).generate(str(wordcloud_text))
plt.imshow(wordcloud, interpolation="bilinear")
plt.title("Word cloud for Cleansed text happiness")
plt.axis("off")
plt.show()
```



Word Cloud visualization for angriness

```
df1 = df[['intensity','Final_word_list']]
df1 = df1[df1['intensity']=='angriness']
top = Counter([item for sublist in df1['Final_word_list'] for item in sublist])
temp_df_after_cleansing = pd.DataFrame(top.most_common(400))
temp_df_after_cleansing.columns = ['Common_words','count']
temp_df_after_cleansing.style.background_gradient(cmap = 'Blues')

wordcloud_text = ','.join(list(temp_df_after_cleansing['Common_words'].values))
wordcloud = WordCloud(background_color="white",max_words=5000, contour_width=3).generate(str(wordcloud_text))
plt.imshow(wordcloud, interpolation="bilinear")
plt.title("Word cloud for Cleansed text angriness")
plt.axis("off")
plt.show()
```

Word Cloud visualization for sadness

```
df1 = df[['intensity','Final_word_list']]
df1 = df1[df1['intensity']=='sadness']
top = Counter([item for sublist in df1['Final_word_list'] for item in sublist])
temp_df_after_cleansing = pd.DataFrame(top.most_common(400))
temp_df_after_cleansing.columns = ['Common_words','count']
temp_df_after_cleansing.style.background_gradient(cmap = 'Blues')

wordcloud_text = ','.join(list(temp_df_after_cleansing['Common_words'].values))
wordcloud = WordCloud(background_color="white",max_words=5000, contour_width=3).generate(str(wordcloud_text))
plt.imshow(wordcloud, interpolation="bilinear")
plt.title("Word cloud for Cleansed text sadness")
plt.axis("off")
plt.show()
```

Final data frame used to train and test using different models

```
df
```

	content	intensity	content_with_sw	word_list	Final_content	Final_word_list
0	Never hurt people who love you a lot, because ...	sadness	never hurt people who love you a lot because t...	[never, hurt, people, who, love, you, a, lot, ...	never hurt love lot hurt back probably choice ...	[never, hurt, love, lot, hurt, back, probably,...
1	Don't expect me to tell you what you did wrong...	sadness	do not expect me to tell you what you did wron...	[do, not, expect, me, to, tell, you, what, you...	expect wrong figure ready correct itcos know m...	[expect, wrong, figure, ready, correct, itcos,...
2	I preferred walking away than fighting for you...	sadness	i preferred walking away than fighting for you...	[i, preferred, walking, away, than, fighting, ...	preferred walking away fighting worth fighting...	[preferred, walking, away, fighting, worth, fi...
3	Moving forward in life isn't the hard part, it...	sadness	moving forward in life is not the hard part it...	[moving, forward, in, life, is, not, the, hard...	moving ward life hard leaving move ward	[moving, ward, life, hard, leaving, move, ward]
4	Never cry for anyone in your life, because tho...	sadness	never cry for anyone in your life because thos...	[never, cry, for, anyone, in, your, life, beca...	never cry life cry deserve tear deserving neve...	[never, cry, life, cry, deserve, tear, deservi...
...	...	...	...	...	...	...
1587	The embarrassment after the anger is the bigge...	angriness	the embarrassment after the anger is the bigge...	[the, embarrassment, after, the, anger, is, th...	embarrassment anger biggest humiliation person...	[embarrassment, anger, biggest, humiliation, p...
1588	The strong man is not the good wrestler; the s...	angriness	the strong man is not the good wrestler the st...	[the, strong, man, is, not, the, good, wrestle...	strong good wrestler strong control angry	[strong, good, wrestler, strong, control, angry]
1589	A man is about as big as the things that make ...	angriness	a man is about as big as the things that make ...	[a, man, is, about, as, big, as, the, things, ...	big make angry	[big, make, angry]
1590	Is singing, if you're angry & you know it punc...	angriness	is singing if you are angry you know it punch...	[is, singing, if, you, are, angry, you, know, ...	singing angry know punch face	[singing, angry, know, punch, face]
1591	Telling someone to calm down just makes them s...	angriness	telling someone to calm down just makes them s...	[telling, someone, to, calm, down, just, makes...	ing calm make angrier	[ing, calm, make, angrier]

1592 rows × 6 columns

Training and Testing models

Implementation Process

Step 1: Import the model using sklearn libraries.

Step 2: Split the training and test data with appropriate parameters and hyperparameters.

Step 3: Fit the model with training data.

Step 4: Predict the model with test data.

Step 5: Evaluate the model with test data.

Step 6: Evaluate various metrics like accuracy, precision, recall, f1-score and confusion matrix.

Step 6: Fine tune the model using imbalance technique SMOTE (Synthetic Minority Over-sampling Technique).

Hyper Parameters

Logistic Regression – maxIter

Random Forest classifier – n-estimators, random state, min_samples_split, min_samples_leaf

Multinomial Naïve Bayes Grid Search – Param-grid, n-jobs, cv, verbose

Gradient Boost Classifier – n-estimators, learning rate, max_depth

SMOTE – random_state, k_neighbors

Data Splitting

Training data is split to 70% and test data to 20% using train_test split. Text values are converted into vector using Count Vectorizer. Intensity values are encoded using LabelEncoder.

```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['intensity']= label_encoder.fit_transform(df['intensity'])

df['intensity'].unique()
```

```
array([2, 1, 0])
```

```
# create a count vertorizer object and set the size of the vocabulary to 8000
cv = CountVectorizer(max_features = 5000, ngram_range = (1,2), min_df = 3, max_df = 0.6)

# convert the dtype of final tweet column to unicode string and convert them to bag of words
X = cv.fit_transform(train_df['content']).toarray()
y = train_df['intensity']
```

```
X.shape
```

```
(1592, 873)
```

```
y.shape
```

```
(1592,)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

```
print("number of data in training dataset: ", len(X_train))
print("number of data in testing dataset: ", len(X_test))
```

```
number of data in training dataset:  1114
number of data in testing dataset:  478
```

Implementation

Intensity Data trained on six different classification models.

1. Logistic Regression
2. Gaussian Naïve Bayes
3. Multinomial Naïve Bayes
4. Random Forest Classifier
5. Multinomial Naïve Bayes Grid search
6. Gradient Boost Classifier

Before up-sampling

Implementation 1: Using Logistic Regression

```
# Initialize the Logistic Regression classifier
clf1 = LogisticRegression(max_iter=10000)
```

```
# Train the model on the training data
model1 = clf1.fit(X_train, y_train)
```

```
# Make predictions on the test data
pred_test1 = model1.predict(X_test)
```

```
# Calculate and display the weighted F1 score
display(accuracy_score(pred_test1, y_test)*100)
```

```
64.85355648535564
```

```
# Calculate and display the weighted F1 score
display(precision_score(pred_test1, y_test, average='weighted')*100)
```

```
65.89821122849571
```

```
# Calculate and display the weighted F1 score
display(recall_score(pred_test1, y_test, average='weighted')*100)
```

```
64.85355648535564
```

```
# Calculate and display the weighted F1 score
display(round(metrics.f1_score(pred_test1, y_test, average='weighted')* 100,2))
```

```
65.12
```

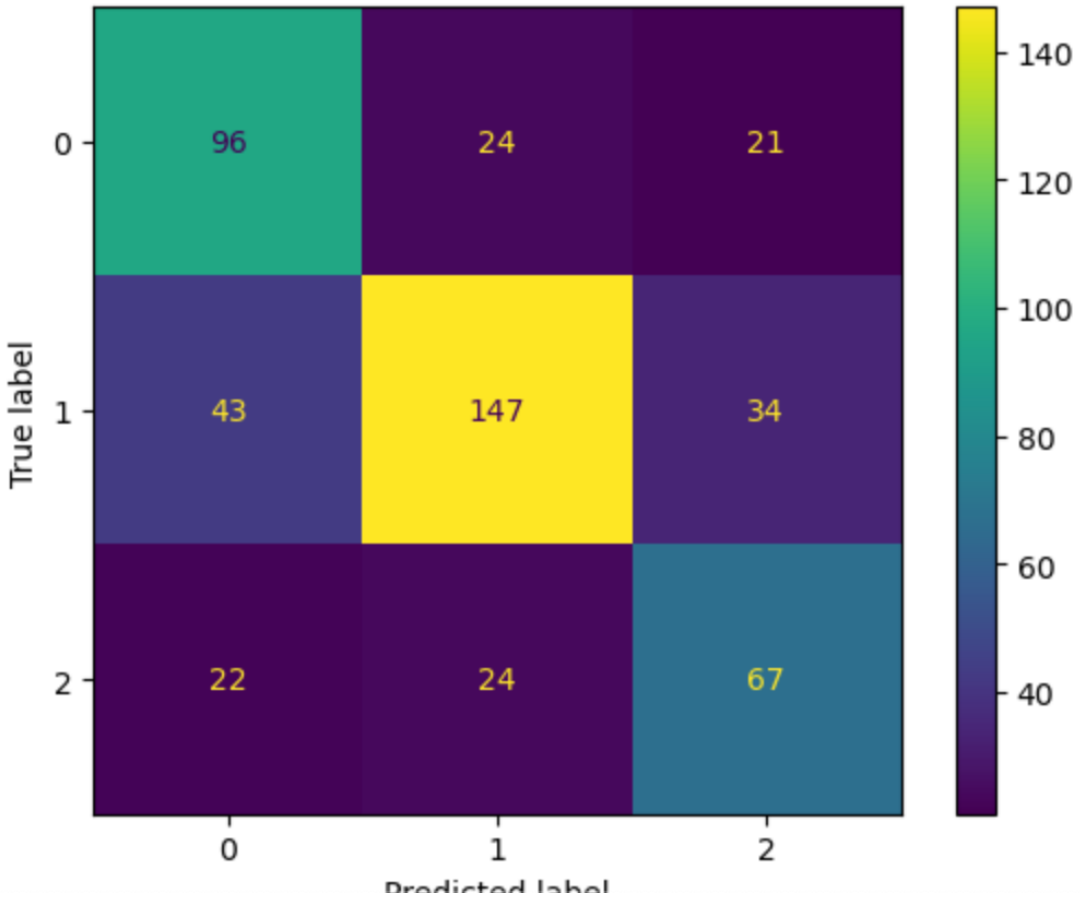
```
report1 = classification_report(y_test, pred_test1)
print(f"Classification Report: \n{report1}")
```

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

           0       0.68      0.60      0.64       161
           1       0.66      0.75      0.70       195
           2       0.59      0.55      0.57       122

    accuracy                           0.65       478
   macro avg       0.64      0.63      0.64       478
weighted avg       0.65      0.65      0.65       478
```

```
conf_matrix1 = confusion_matrix(pred_test1, y_test)
display(conf_matrix1)
```

```
array([[ 96,  24,  21],
       [ 43, 147,  34],
       [ 22,  24,  67]])
```

```
disp = ConfusionMatrixDisplay(conf_matrix1)
disp.plot()
plt.show()
```



Implementation 2: Using Gaussian Naïve Bayes Classifier

```
# Initialize the Gaussian Naive Bayes classifier
clf2 = GaussianNB()
```

```
# Train the model on the training data
model2 = clf2.fit(X_train, y_train)
```

```
# Make predictions on the test data
pred_test2 = model2.predict(X_test)
```

```
# Calculate and display the weighted F1 score
display(accuracy_score(pred_test2, y_test)*100)
```

```
52.30125523012552
```

```
# Calculate and display the weighted F1 score
display(precision_score(pred_test2, y_test, average='weighted')*100)
```

```
54.95041240347921
```

```
# Calculate and display the weighted F1 score
display(recall_score(pred_test2, y_test, average='weighted')*100)
```

```
52.30125523012552
```

```
# Calculate and display the weighted F1 score
display(round(metrics.f1_score(pred_test2, y_test, average='weighted')* 100,2))
```

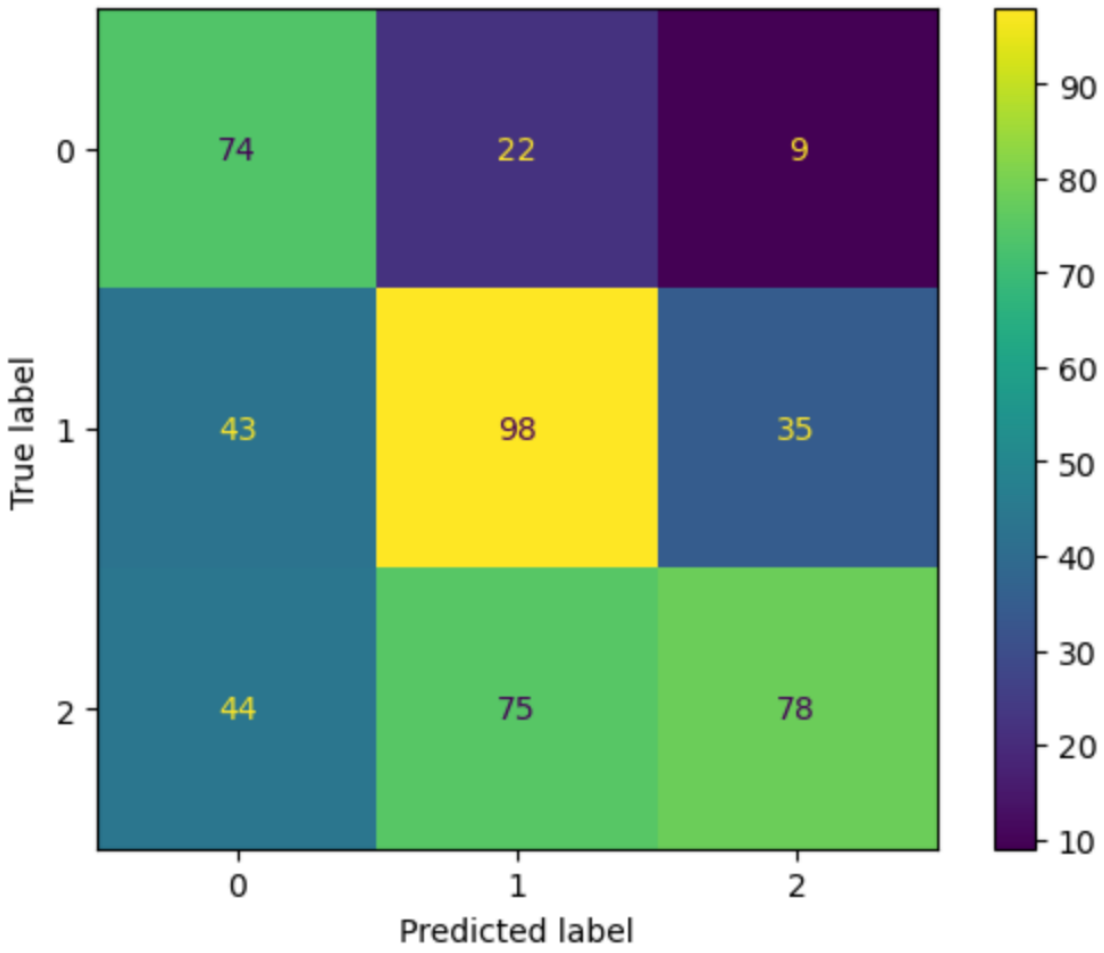
```
51.83
```

```
report2 = classification_report(y_test, pred_test2)
print(f"Classification Report: \n{report2}")
```

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

           0       0.70      0.46      0.56       161
           1       0.56      0.50      0.53       195
           2       0.40      0.64      0.49       122

    accuracy                           0.52       478
   macro avg       0.55      0.53      0.52       478
weighted avg       0.57      0.52      0.53       478
```

```
conf_matrix2 = confusion_matrix(pred_test2, y_test)
display(conf_matrix2)

disp = ConfusionMatrixDisplay(conf_matrix2)
disp.plot()
plt.show()
```

```
array([[74, 22,  9],
       [43, 98, 35],
       [44, 75, 78]])
```



Implementation 3: Using Multinomial Naïve Bayes Classifier

```
clf3 = MultinomialNB()
```

```
# Train the model on the training data
model3 = clf3.fit(X_train, y_train)
```

```
# Make predictions on the test data
pred_test3 = model3.predict(X_test)
```

```
# Calculate and display the weighted F1 score
display(accuracy_score(pred_test3, y_test)*100)
```

```
65.69037656903765
```

```
# Calculate and display the weighted F1 score
display(precision_score(pred_test3, y_test, average='weighted')*100)
```

```
67.35798996435946
```

```
# Calculate and display the weighted F1 score
display(recall_score(pred_test3, y_test, average='weighted')*100)
```

```
65.69037656903765
```

```
# Calculate and display the weighted F1 score
display(round(metrics.f1_score(pred_test3, y_test, average='weighted')* 100,2))
```

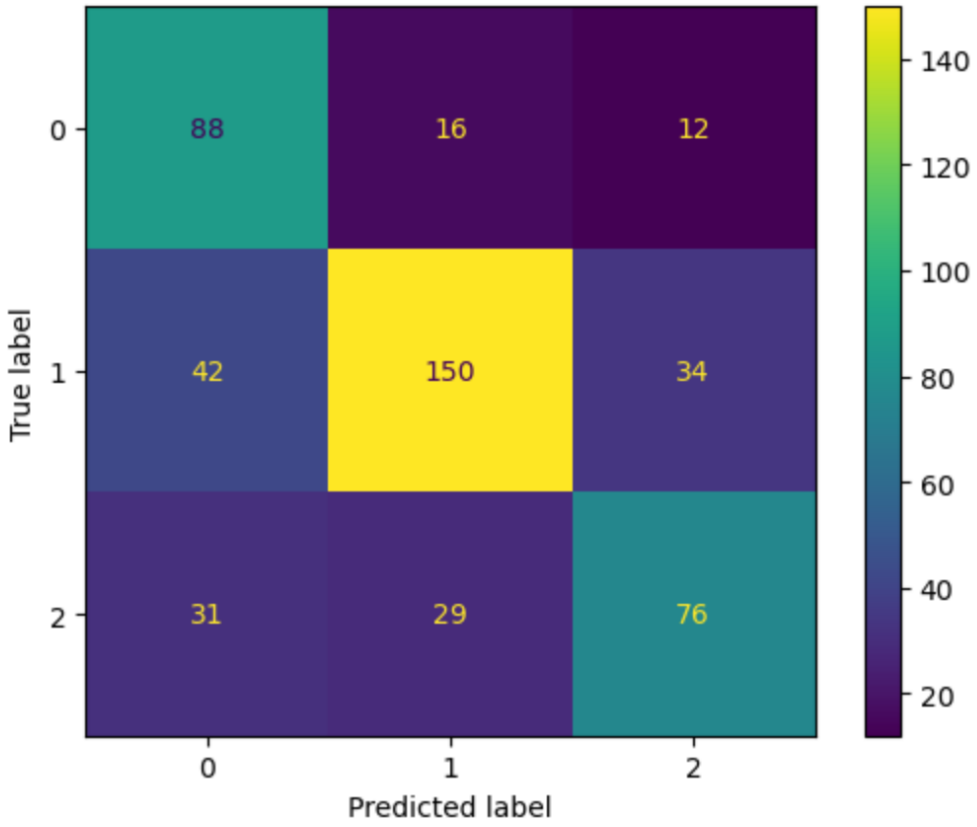
```
65.87
```

```
report3 = classification_report(y_test, pred_test3)
print(f"Classification Report: \n{report3}")
```

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

           0       0.76      0.55      0.64       161
           1       0.66      0.77      0.71       195
           2       0.56      0.62      0.59       122

    accuracy                           0.66       478
   macro avg       0.66      0.65      0.65       478
weighted avg       0.67      0.66      0.66       478
```

```
conf_matrix3 = confusion_matrix(pred_test3, y_test)
display(conf_matrix3)

disp = ConfusionMatrixDisplay(conf_matrix3)
disp.plot()
plt.show()
```

```
array([[ 88,  16,  12],
       [ 42, 150,  34],
       [ 31,  29,  76]])
```



Implementation 4: Using Random Forest Classifier

```
# Model Building
clf4 = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
# Train the model on the training data
model4 = clf4.fit(X_train, y_train)
```

```
# Make predictions on the test data
pred_test4 = model4.predict(X_test)
```

```
# Calculate and display the weighted F1 score
display(accuracy_score(pred_test4, y_test)*100)
```

```
61.92468619246861
```

```
# Calculate and display the weighted F1 score
display(precision_score(pred_test4, y_test, average='weighted')*100)
```

```
63.81728514926355
```

```
# Calculate and display the weighted F1 score
display(recall_score(pred_test4, y_test, average='weighted')*100)
```

```
61.92468619246861
```

```
# Calculate and display the weighted F1 score
display(round(metrics.f1_score(pred_test4, y_test, average='weighted')* 100,2))
```

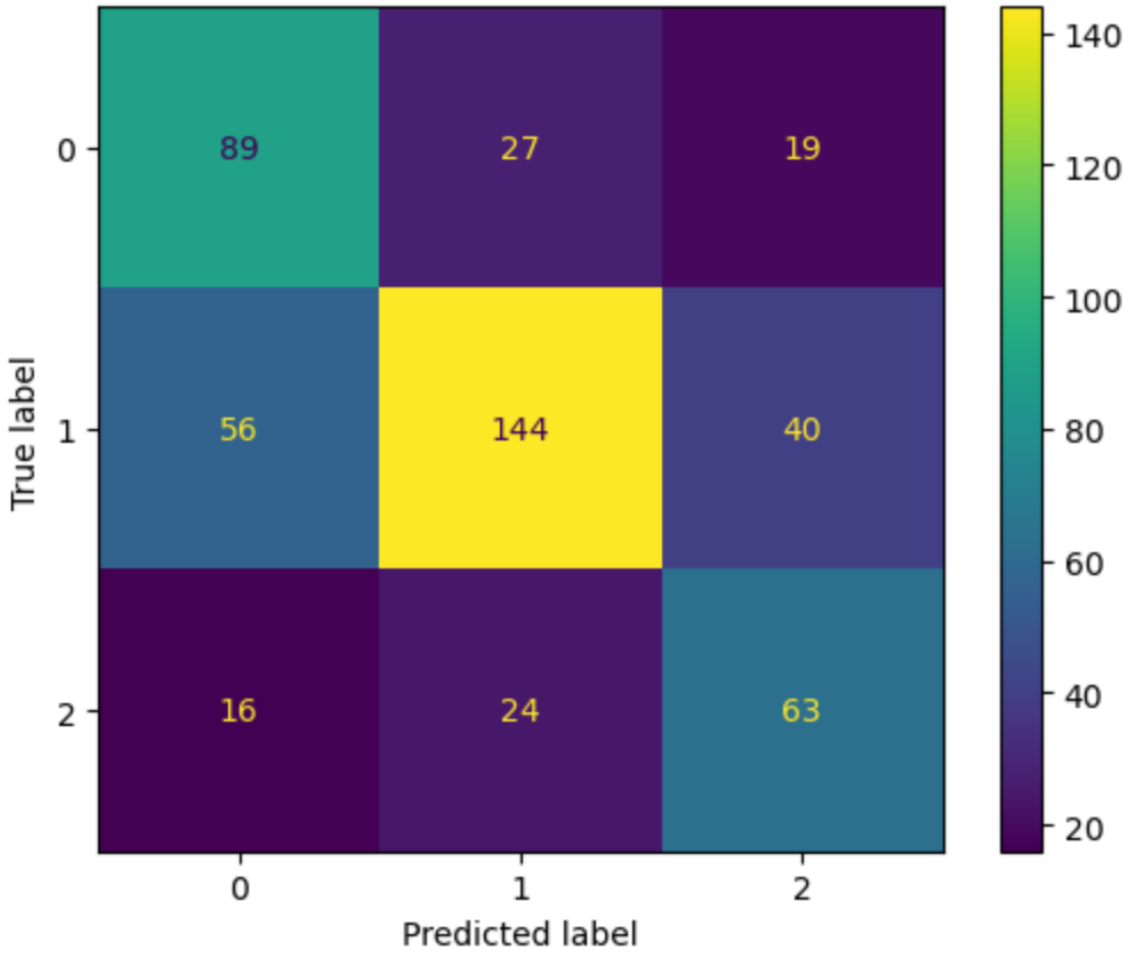
```
62.29
```

```
report4 = classification_report(y_test, pred_test4)
print(f"Classification Report: \n{report4}")
```

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

           0       0.66      0.55      0.60       161
           1       0.60      0.74      0.66       195
           2       0.61      0.52      0.56       122

    accuracy                           0.62       478
   macro avg       0.62      0.60      0.61       478
weighted avg       0.62      0.62      0.62       478
```

```
conf_matrix4 = confusion_matrix(pred_test4, y_test)
display(conf_matrix4)

disp = ConfusionMatrixDisplay(conf_matrix4)
disp.plot()
plt.show()
```

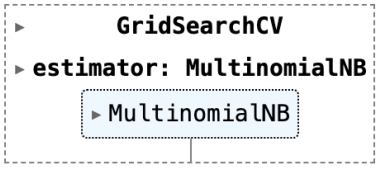
```
array([[ 89,  27,  19],
       [ 56, 144,  40],
       [ 16,  24,  63]])
```



Implementation 5: Using Multinomial Naïve Bayes Grid Search

```
params = {'alpha': [0.01,0.1,0.5,1,10],}

model7 = GridSearchCV(MultinomialNB(), param_grid=params, n_jobs=-1, cv=5, verbose=5)
model7.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```




```
# Make predictions on the test data
pred_test7 = model7.predict(X_test)
```

```
print('Train Accuracy : %.3f'%model7.best_estimator_.score(X_train, y_train))
print('Test Accuracy : %.3f'%model7.best_estimator_.score(X_test, y_test))
print('Best Accuracy Through Grid Search : %.3f'%model7.best_score_)
print('Best Parameters : ',model7.best_params_)
```

```
Train Accuracy : 0.852
Test Accuracy : 0.651
Best Accuracy Through Grid Search : 0.677
Best Parameters :  {'alpha': 0.5}
```

```
# Calculate and display the weighted F1 score
display(accuracy_score(pred_test7, y_test)*100)
```

```
65.06276150627615
```

```
# Calculate and display the weighted F1 score
display(precision_score(pred_test7, y_test, average='weighted')*100)
```

```
66.58942206358532
```

```
# Calculate and display the weighted F1 score
display(recall_score(pred_test7, y_test, average='weighted')*100)
```

```
65.06276150627615
```

```
# Calculate and display the weighted F1 score
display(round(metrics.f1_score(pred_test7, y_test, average='weighted')* 100,2)
```

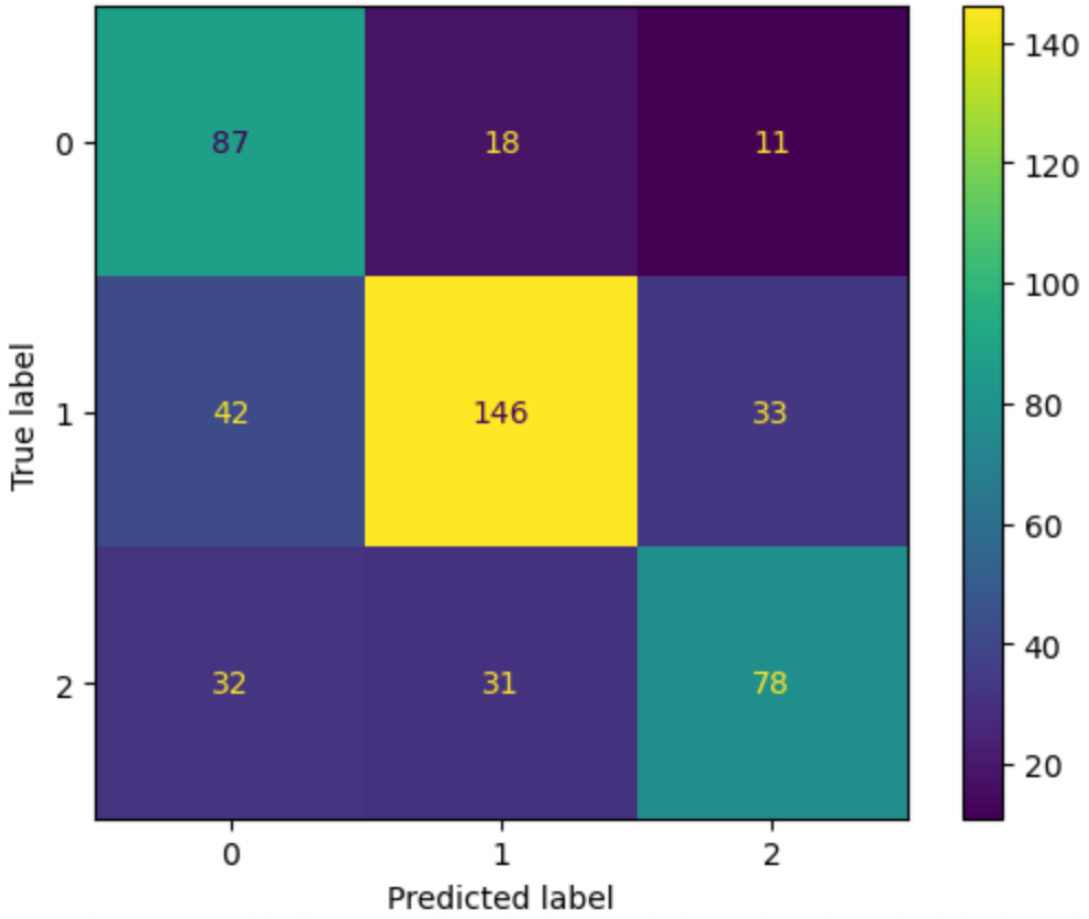
```
65.19
```

```
report7 = classification_report(y_test, pred_test7)
print(f"Classification Report: \n{report7}")
```

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

           0       0.75      0.54      0.63       161
           1       0.66      0.75      0.70       195
           2       0.55      0.64      0.59       122

    accuracy                           0.65       478
   macro avg       0.65      0.64      0.64       478
weighted avg       0.66      0.65      0.65       478
```

```
conf_matrix7= confusion_matrix(pred_test7, y_test)
display(conf_matrix7)

disp = ConfusionMatrixDisplay(conf_matrix7)
disp.plot()
plt.show()
```

```
array([[ 87,  18,  11],
       [ 42, 146,  33],
       [ 32,  31,  78]])
```



Implementation 6: Using Gradient Boost Classifier

```
model8 = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_dep
```

```
model8.fit(X_train,y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=1.0, max_depth=1)
```

```
# Make predictions on the test data
pred_test8 = model8.predict(X_test)
```

```
# Calculate and display the weighted F1 score
display(accuracy_score(pred_test8, y_test)*100)
```

```
64.22594142259415
```

```
# Calculate and display the weighted F1 score
display(precision_score(pred_test8, y_test, average='weighted')*100)
```

```
65.89219439729554
```

```
display(recall_score(pred_test8, y_test, average='weighted')*100)
```

```
64.22594142259415
```

```
display(round(metrics.f1_score(pred_test8, y_test, average='weighted')* 100,2))
```

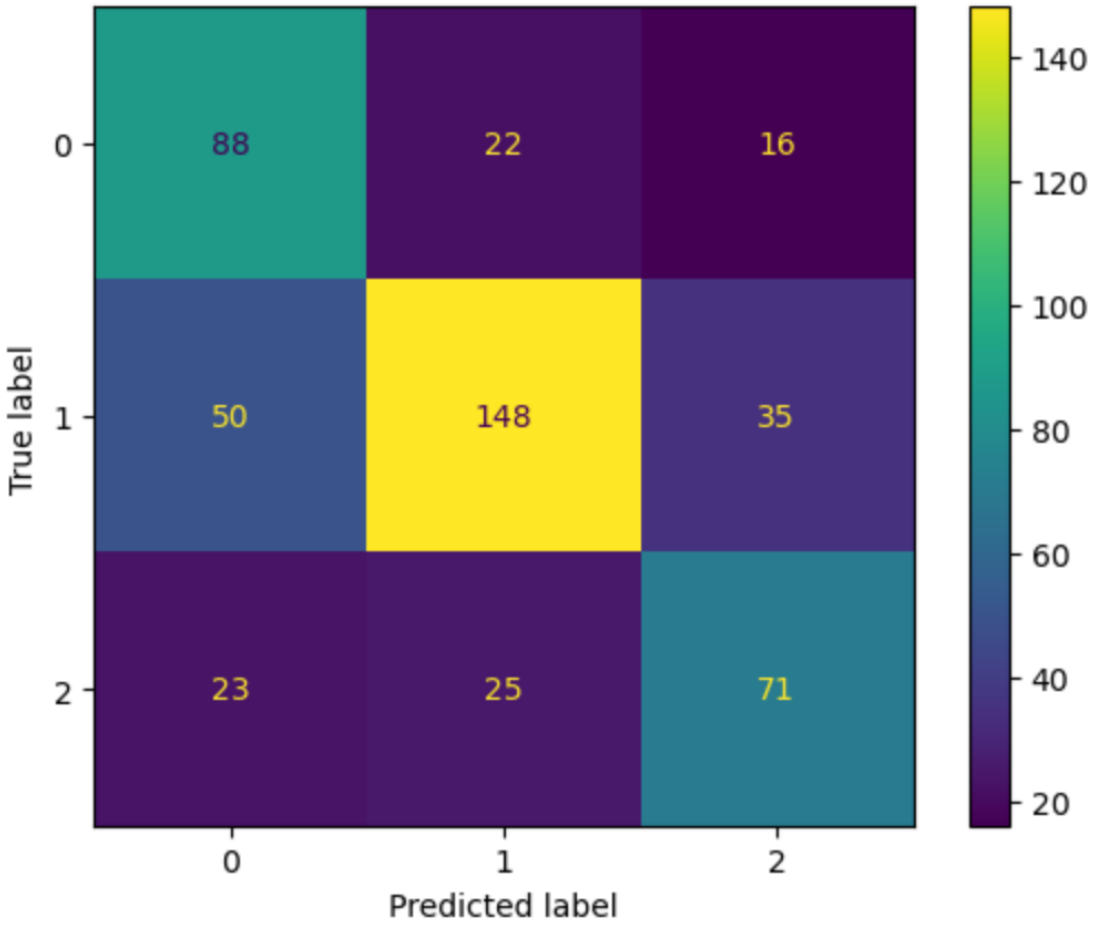
```
64.54
```

```
report8 = classification_report(y_test, pred_test8)
print(f"Classification Report: \n{report8}")
```

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

           0       0.70      0.55      0.61       161
           1       0.64      0.76      0.69       195
           2       0.60      0.58      0.59       122

    accuracy                           0.64       478
   macro avg       0.64      0.63      0.63       478
weighted avg       0.65      0.64      0.64       478
```

```
conf_matrix8= confusion_matrix(pred_test8, y_test)
display(conf_matrix8)

disp = ConfusionMatrixDisplay(conf_matrix8)
disp.plot()
plt.show()
```

```
array([[ 88,  22,  16],
       [ 50, 148,  35],
       [ 23,  25,  71]])
```



After Up-Sampling – Balancing the imbalance data using SMOTE Technique

```
!pip install -U threadpoolctl
```

```
Requirement already satisfied: threadpoolctl in /Users/thavaseelan/anaconda3/lib/python3.11/site-packages (3.5.0)
```

```
from imblearn.over_sampling import SMOTE
```

```
smote = SMOTE(random_state = 42, k_neighbors=2)
smote
```

```
SMOTE(k_neighbors=2, random_state=42)
```

```
X_resampling,y_resampling = smote.fit_resample(X,y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_resampling, y_resampling, test_size = 0.3, random_state = 42)
```

Implementation 1: Using Logistic Regression

```
# Initialize the Logistic Regression classifier
clf01 = LogisticRegression(max_iter=10000)
```

```
# Train the model on the training data
model01 = clf01.fit(X_train, y_train)
```

```
# Make predictions on the test data
pred_test01 = model01.predict(X_test)
```

```
print('Accuracy is:', accuracy_score(pred_test01, y_test)*100)
print('Precision is:', precision_score(pred_test01, y_test, average='weighted')*100)
print('Recall is:', recall_score(pred_test01, y_test, average='weighted')*100)
print('f1-score is:', round(metrics.f1_score(pred_test01, y_test, average='weighted')* 100,2))
```

```
Accuracy is: 70.82018927444796
Precision is: 71.69184416930615
Recall is: 70.82018927444796
f1-score is: 70.74
```

```
report01 = classification_report(y_test, pred_test01)
print(f"Classification Report: \n{report1}")
```

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

           0       0.68      0.60      0.64       161
           1       0.66      0.75      0.70       195
           2       0.59      0.55      0.57       122

    accuracy                           0.65       478
   macro avg       0.64      0.63      0.64       478
weighted avg       0.65      0.65      0.65       478
```

Implementation 2: Using Gaussian Naïve Bayes Classifier

```
# Initialize the Gaussian Naive Bayes classifier
clf02 = GaussianNB()
```

```
# Train the model on the training data
model02 = clf02.fit(X_train, y_train)
```

```
# Make predictions on the test data
pred_test02 = model02.predict(X_test)
```

```
print('Accuracy is:', accuracy_score(pred_test02, y_test)*100)
print('Precision is:', precision_score(pred_test02, y_test, average='weighted')*100)
print('Recall is:', recall_score(pred_test02, y_test, average='weighted')*100)
print('f1-score is:', round(metrics.f1_score(pred_test02, y_test, average='weighted')* 100,2))
```

```
Accuracy is: 58.83280757097792
Precision is: 65.29913705625444
Recall is: 58.83280757097792
f1-score is: 59.61
```

```
report02 = classification_report(y_test, pred_test02)
print(f"Classification Report: \n{report02}")
```

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

           0       0.74      0.42      0.53       211
           1       0.62      0.55      0.58       213
           2       0.52      0.80      0.63       210

    accuracy                           0.59       634
   macro avg       0.62      0.59      0.58       634
weighted avg       0.62      0.59      0.58       634
```

Implementation 3: Using Multinomial Naïve Bayes Classifier

```
clf03 = MultinomialNB()
```

```
# Train the model on the training data
model03 = clf03.fit(X_train, y_train)
```

```
# Make predictions on the test data
pred_test03 = model03.predict(X_test)
```

```
print('Accuracy is:', accuracy_score(pred_test03, y_test)*100)
print('Precision is:', precision_score(pred_test03, y_test, average='weighted')*100)
print('Recall is:', recall_score(pred_test03, y_test, average='weighted')*100)
print('f1-score is:', round(metrics.f1_score(pred_test03, y_test, average='weighted')* 100,2))
```

```
Accuracy is: 66.08832807570978
Precision is: 68.7161286074273
Recall is: 66.08832807570978
f1-score is: 66.26
```

```
report03 = classification_report(y_test, pred_test03)
print(f"Classification Report: \n{report03}")
```

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

           0       0.75      0.55      0.63       211
           1       0.70      0.65      0.67       213
           2       0.58      0.79      0.67       210

    accuracy                           0.66       634
   macro avg       0.68      0.66      0.66       634
weighted avg       0.68      0.66      0.66       634
```

Implementation 4: Using Random Forest Classifier

```
# Model Building
clf04 = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
# Train the model on the training data
model04 = clf04.fit(X_train, y_train)
```

```
# Make predictions on the test data
pred_test04 = model04.predict(X_test)
```

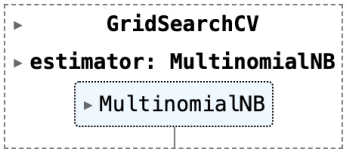
```
print('Accuracy is:', accuracy_score(pred_test04, y_test)*100)
print('Precision is:', precision_score(pred_test04, y_test, average='weighted')*100)
print('Recall is:', recall_score(pred_test04, y_test, average='weighted')*100)
print('f1-score is:', round(metrics.f1_score(pred_test04, y_test, average='weighted')* 100,2))
```

```
Accuracy is: 65.61514195583597
Precision is: 68.54141990299526
Recall is: 65.61514195583597
f1-score is: 65.76
```

```
report04 = classification_report(y_test, pred_test04)
print(f"Classification Report: \n{report04}")
```

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

           0       0.73      0.62      0.67       211
           1       0.72      0.55      0.63       213
           2       0.57      0.80      0.67       210

    accuracy                           0.66       634
   macro avg       0.68      0.66      0.65       634
weighted avg       0.68      0.66      0.65       634
```

Implementation 5: Using Multinomial Naïve Bayes Grid search




```
# Make predictions on the test data
pred_test07 = model07.predict(X_test)
```

```
print('Accuracy is:', accuracy_score(pred_test07, y_test)*100)
print('Precision is:', precision_score(pred_test07, y_test, average='weighted')*100)
print('Recall is:', recall_score(pred_test07, y_test, average='weighted')*100)
print('f1-score is:', round(metrics.f1_score(pred_test07, y_test, average='weighted')* 100,2))
```

```
Accuracy is: 66.08832807570978
Precision is: 68.42409654000973
Recall is: 66.08832807570978
f1-score is: 66.28
```

```
report07 = classification_report(y_test, pred_test07)
print(f"Classification Report: \n{report07}")
```

```
Classification Report:
              precision    recall  f1-score   sup Javascript enabled for notebook display

           0       0.74      0.55      0.63       211
           1       0.70      0.65      0.67       213
           2       0.59      0.79      0.67       210

    accuracy                           0.66       634
   macro avg       0.68      0.66      0.66       634
weighted avg       0.68      0.66      0.66       634
```

```
print('Train Accuracy : %.3f'%model07.best_estimator_.score(X_train, y_train))
print('Test Accuracy : %.3f'%model07.best_estimator_.score(X_test, y_test))
print('Best Accuracy Through Grid Search : %.3f'%model7.best_score_)
print('Best Parameters : ',model07.best_params_)
```

```
Train Accuracy : 0.802
Test Accuracy : 0.661
Best Accuracy Through Grid Search : 0.677
Best Parameters :  {'alpha': 0.5}
```

Implementation 6: Using Gradient Boost Classifier

```
clf08 = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1)
```

```
# Train the model on the training data
model08 = clf08.fit(X_train, y_train)
```

```
# Make predictions on the test data
pred_test08 = model08.predict(X_test)
```

```
print('Accuracy is:', accuracy_score(pred_test08, y_test)*100)
print('Precision is:', precision_score(pred_test08, y_test, average='weighted')*100)
print('Recall is:', recall_score(pred_test08, y_test, average='weighted')*100)
print('f1-score is:', round(metrics.f1_score(pred_test08, y_test, average='weighted')* 100,2))
```

```
Accuracy is: 66.40378548895899
Precision is: 68.33863372299018
Recall is: 66.40378548895899
f1-score is: 66.37
```

```
report08 = classification_report(y_test, pred_test08)
print(f"Classification Report: \n{report08}")
```

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

           0       0.72      0.61      0.66       211
           1       0.75      0.61      0.67       213
           2       0.58      0.78      0.66       210

    accuracy                           0.66       634
   macro avg       0.68      0.66      0.66       634
weighted avg       0.68      0.66      0.66       634
```

Metrics Comparison

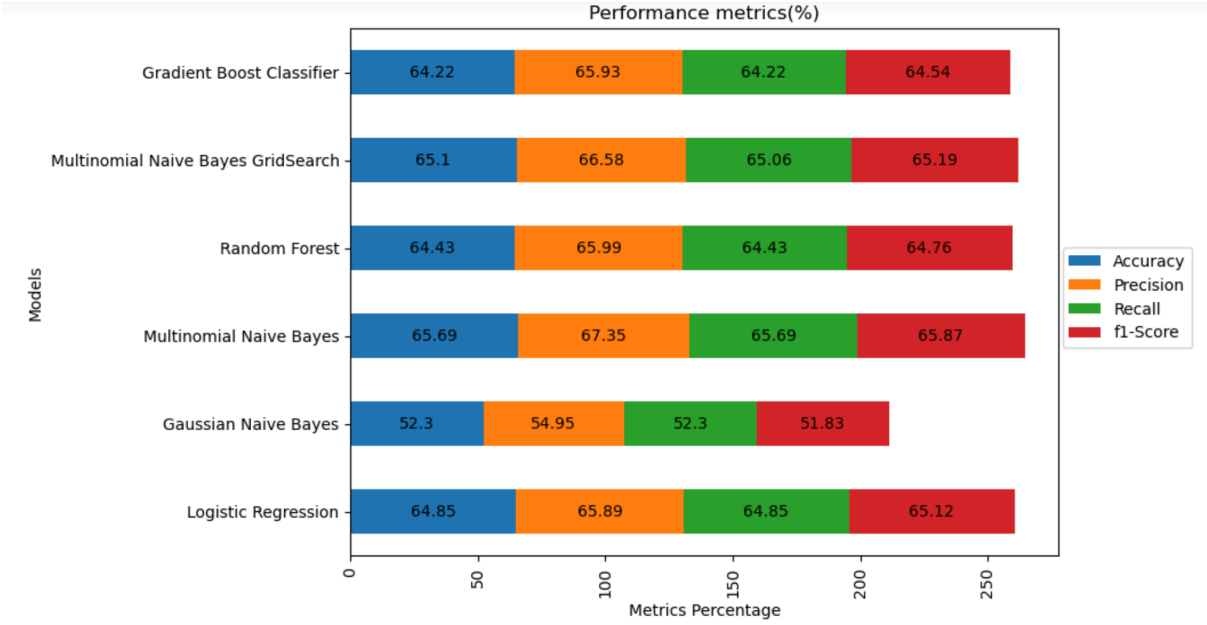
Before up-sampling

Below are the various metrics compared with different models.

```
df1 = pd.DataFrame([
    ['Logistic Regression', 64.85, 65.89, 64.85, 65.12],
    ['Gaussian Naive Bayes', 52.30, 54.95, 52.30, 51.83],
    ['Multinomial Naive Bayes', 65.69, 67.35, 65.69, 65.87],
    ['Random Forest', 64.43, 65.99, 64.43, 64.76],
    ['Multinomial Naive Bayes GridSearch', 65.10, 66.58, 65.06, 65.19],
    ['Gradient Boost Classifier', 64.22, 65.93, 64.22, 64.54]
                ],
    columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'f1-Score']
)

print("Performance metrics for Intensity Analysis before SMOTE sampling(%)")
print(df1)
```

```
Performance metrics for Intensity Analysis before SMOTE sampling(%)
                          Model  Accuracy  Precision  Recall  f1-Score
0           Logistic Regression     64.85      65.89   64.85     65.12
1         Gaussian Naive Bayes      52.30      54.95   52.30     51.83
2      Multinomial Naive Bayes      65.69      67.35   65.69     65.87
3                Random Forest      64.43      65.99   64.43     64.76
4  Multinomial Naive Bayes GridSearch     65.10      66.58   65.06     65.19
5      Gradient Boost Classifier      64.22      65.93   64.22     64.54
```

```
# plot data in stack manner of bar type
b = df1.plot(x='Model', kind='barh', stacked=True,figsize = (8,6), xlabel='Metrics Percentage', ylabel = 'Models',
        title='Performance metrics(%)')

for c in b.containers:
  #labels = [v.get_height() if v.get_height() > 0 else '' for v in c]
  b.bar_label(c,  label_type = 'center')
plt.legend(loc='upper right', bbox_to_anchor=(0.6, 0., 0.6, 0.6))
plt.xticks(rotation=90)

plt.show()
```

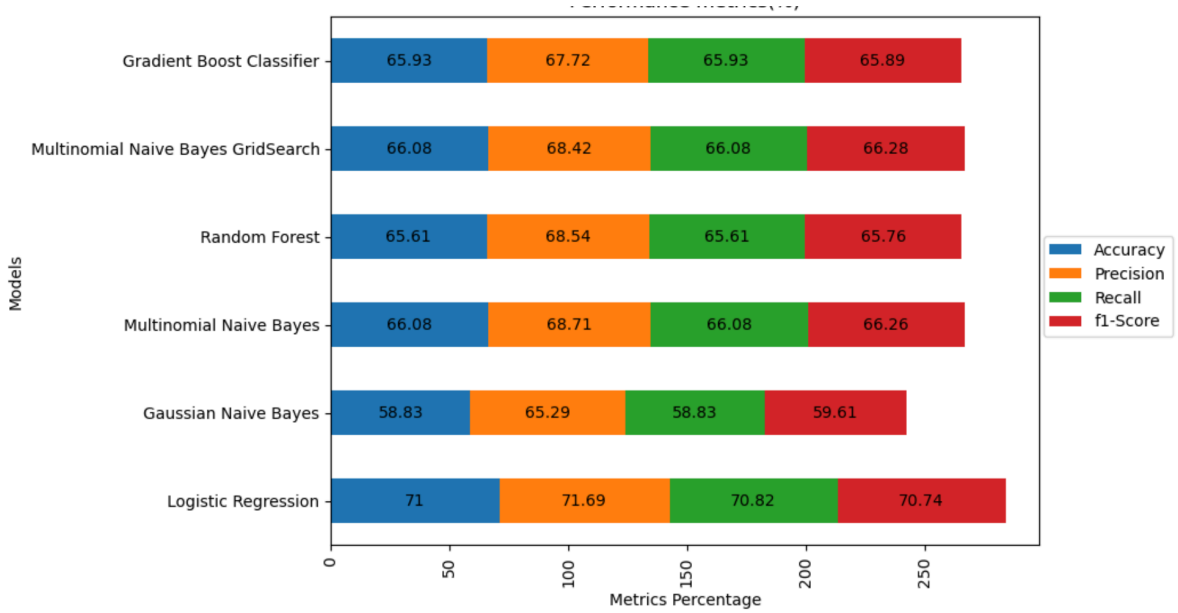


After up-sampling

```
df2 = pd.DataFrame([
    ['Logistic Regression', 71, 71.69, 70.82, 70.74],
    ['Gaussian Naive Bayes', 58.83, 65.29, 58.83, 59.61],
    ['Multinomial Naive Bayes', 66.08, 68.71, 66.08, 66.26],
    ['Random Forest', 65.61, 68.54, 65.61, 65.76],
    ['Multinomial Naive Bayes GridSearch', 66.08, 68.42, 66.08, 66.28],
    ['Gradient Boost Classifier', 65.93, 67.72, 65.93, 65.89]
                ],
    columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'f1-Score']
)

print("Performance metrics for Intensity Analysis after SMOTE sampling(%)")
print(df2)
```

```
Performance metrics for Intensity Analysis after SMOTE sampling(%)
                            Model  Accuracy  Precision  Recall  f1-Score
0             Logistic Regression     71.00      71.69   70.82     70.74
1           Gaussian Naive Bayes      58.83      65.29   58.83     59.61
2        Multinomial Naive Bayes      66.08      68.71   66.08     66.26
3                  Random Forest      65.61      68.54   65.61     65.76
4  Multinomial Naive Bayes GridSearch  66.08      68.42   66.08     66.28
5       Gradient Boost Classifier      65.93      67.72   65.93     65.89
```

```
# plot data in stack manner of bar type
b = df2.plot(x='Model', kind='barh', stacked=True,figsize = (8,6), xlabel='Metrics Percentage', ylabel = 'Models',
        title='Performance metrics(%)')

for c in b.containers:
  #labels = [v.get_height() if v.get_height() > 0 else '' for v in c]
  b.bar_label(c,  label_type = 'center')
plt.legend(loc='upper right', bbox_to_anchor=(0.6, 0., 0.6, 0.6))
plt.xticks(rotation=90)

plt.show()
```



Results Discussion

Before up-sampling:

Multinomial Naïve Bayes Classifier produced good accuracy of 65.69%. Comparing other metrics, True positives are identified, and precision and recall is good 67.35% and 65.69%. Overall, Multinomial Naïve Bayes produces good accuracy, precision, recall

and f1-score before up-sampling. Perhaps, Training data accuracy is 85% for Multinomial Naïve Bayes Grid Search classifier.

After up-sampling:

SMOTE technique is used to balance the imbalance dataset. Metrics performance is good after up-sampling. Logistic Regression gives good accuracy of 71%. While positives are also correctly identified with precision of 72% and recall of 71% and f1-score of 70.5%.

Reason for less accuracy of 71% achieved

In sadness,csv sentiment, there are 245 duplicate entries and in angriness.csv, there is 198 duplicate entries. Whereas, in happiness.csv, there is only 4 duplicate entries, which makes an imbalance while splitting the data. This causes less accuracy of 71% in traditional machine learning training models. This is the reason for achieving less accuracy. This can be improved using any neural network implementation.

Some of the sentiments are neutral and found in all files. Keywords like sad, happy are found in all three csv files, which makes the model to identify wrongly, in some cases. Data Cleansing has been done carefully, to avoid missing any data during the training process.

Deployment

Based on the above analysis, Logistic Regression produces good accuracy. Pickle file is generated and attached in the zip folder named Logisticregression.pickle.

```
import pickle

file_name = "LogisticRegression.pickle"
pickle.dump(clf01, open(file_name, "wb"))
```

Conclusion

Model is trained with accuracy of 86% in Multinomial Naïve Bayes Classifier. Testing accuracy is 71% for Logistic Regression classifier after up-sampling with SMOTE technique.