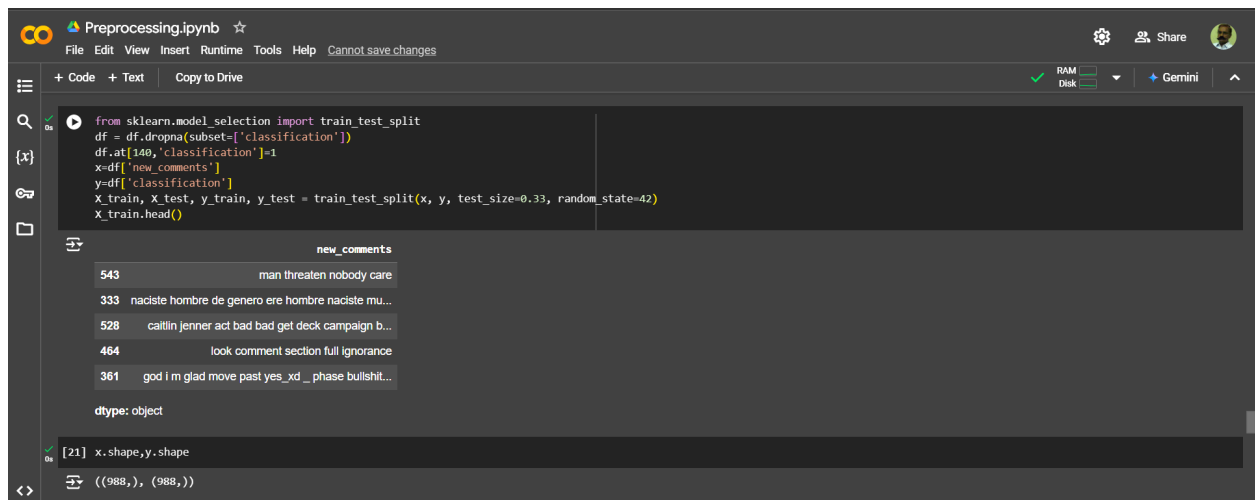


Milestone 2: Data Splitting & Model Training:

1. Splitting Data for Training and Testing

Train-Test Split: Imports the `train_test_split` function from Scikit-learn to divide the dataset into training and testing sets. Here, 33% of the data is allocated for testing.

Variables: `x` holds the cleaned comments, while `y` contains the classification labels.



```
from sklearn.model_selection import train_test_split
df = df.dropna(subset=['classification'])
df.at[140, 'classification'] = 1
x = df['new_comments']
y = df['classification']
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)
X_train.head()
```

| | new_comments |
|-----|---|
| 543 | man threaten nobody care |
| 333 | naciste hombre de genero ere hombre naciste mu... |
| 528 | caitlin jenner act bad bad get deck campaign b... |
| 464 | look comment section full ignorance |
| 361 | god i m glad move past yes_xd _ phase bullshit... |

dtype: object

```
[21] x.shape, y.shape
```

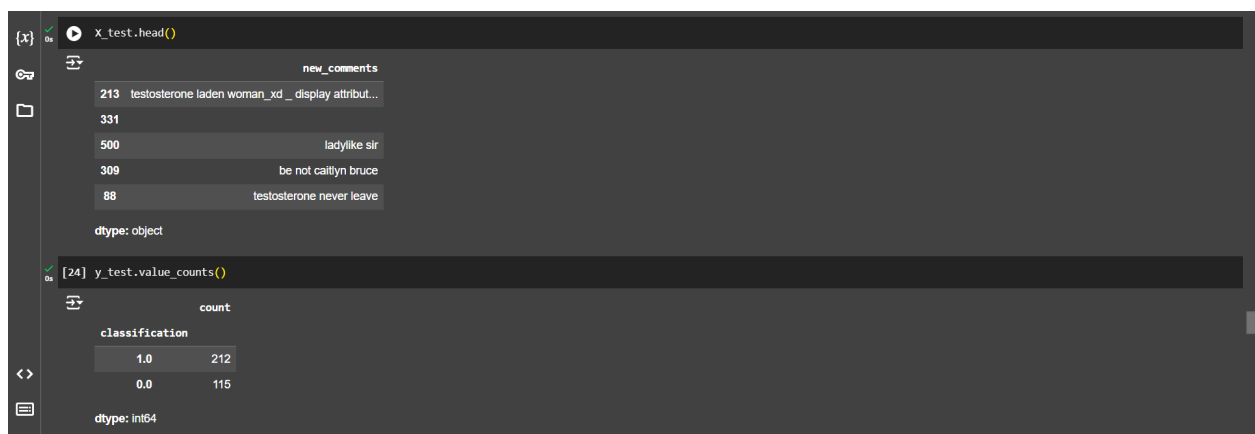
```
((988,), (988,))
```



```
[22] print(y_train.value_counts())
```

| classification | count |
|----------------|-------|
| 1.0 | 419 |
| 0.0 | 242 |

Name: count, dtype: int64



```
X_test.head()
```

| | new_comments |
|-----|---|
| 213 | testosterone laden woman_xd _ display attribut... |
| 331 | |
| 500 | ladylike sir |
| 309 | be not caitlyn bruce |
| 88 | testosterone never leave |

dtype: object

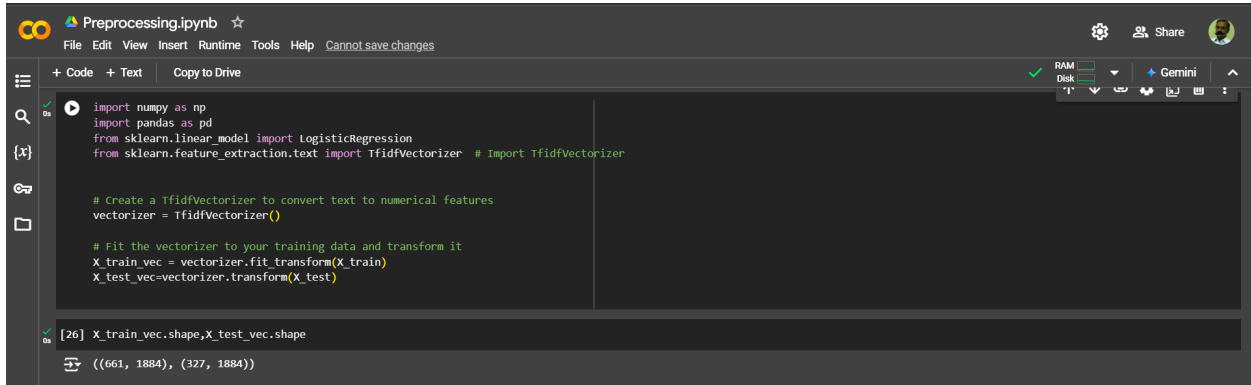
```
[24] y_test.value_counts()
```

| classification | count |
|----------------|-------|
| 1.0 | 212 |
| 0.0 | 115 |

dtype: int64

1. **from sklearn.model_selection import train_test_split:**
 - This line imports the `train_test_split` function from the `sklearn.model_selection` module, which is part of the Scikit-learn library. This function is used to split datasets into training and testing subsets for machine learning models.
2. **x = df['new_comments']:**
 - Here, the variable `x` is assigned the `new_comments` column from the DataFrame `df`. This column contains the preprocessed text data (comments) that will be used as features for training the model.
3. **y = df['classification']:**
 - The variable `y` is assigned the `classification` column from the DataFrame `df`. This column contains the target labels (classifications) that correspond to each comment. In a supervised learning task, this is the output the model will learn to predict based on the input features in `x`.
4. **X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42):**
 - This line calls the `train_test_split` function to split the data into training and testing sets.
 - **x and y:** These are the input features and target labels, respectively.
 - **test_size=0.33:** This parameter specifies the proportion of the dataset to include in the test split. In this case, 33% of the data will be reserved for testing, while the remaining 67% will be used for training.
 - **random_state=42:** This parameter sets the random seed for reproducibility. By specifying a random state, you ensure that the split will produce the same results each time you run the code. This is useful for debugging and for consistent results during experimentation.
 - The function returns four variables: `X_train`, `X_test`, `y_train`, and `y_test`, which represent the training and testing sets of features and labels, respectively.
5. **X_train.head():**
 - This line calls the `head()` method on the `X_train` DataFrame to display the first few rows of the training data. This helps you quickly inspect the training set to verify that the split has been performed correctly.
6. **y_test.value_counts():** Counts the occurrences of each unique value in `y_test`.

2. Model Training:



```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer # Import TfidfVectorizer

# Create a TfidfVectorizer to convert text to numerical features
vectorizer = TfidfVectorizer()

# Fit the vectorizer to your training data and transform it
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

[26] X_train_vec.shape, X_test_vec.shape

((661, 1884), (327, 1884))

This project seems to be a text classification task where logistic regression is used to classify text data. Here's a detailed breakdown of each component in your code:

1. Import Libraries

- **numpy** and **pandas** are popular libraries in Python for data manipulation. **numpy** is often used for numerical operations, and **pandas** is mainly used for handling data in table format.
- **LogisticRegression** from **sklearn.linear_model** provides a logistic regression model, commonly used for binary or multi-class classification tasks.
- **TfidfVectorizer** from **sklearn.feature_extraction.text** is a tool to convert a collection of raw text into numerical data by applying TF-IDF (Term Frequency-Inverse Document Frequency) weighting, which gives more weight to informative words and less to common words.

2. Vectorizing Text with TfidfVectorizer:

- **TfidfVectorizer** transforms the text data into a matrix of TF-IDF features. TF-IDF helps give insight into how important a word is within a document, considering how frequently it appears in a document relative to other documents in the dataset.

How TF-IDF Works :

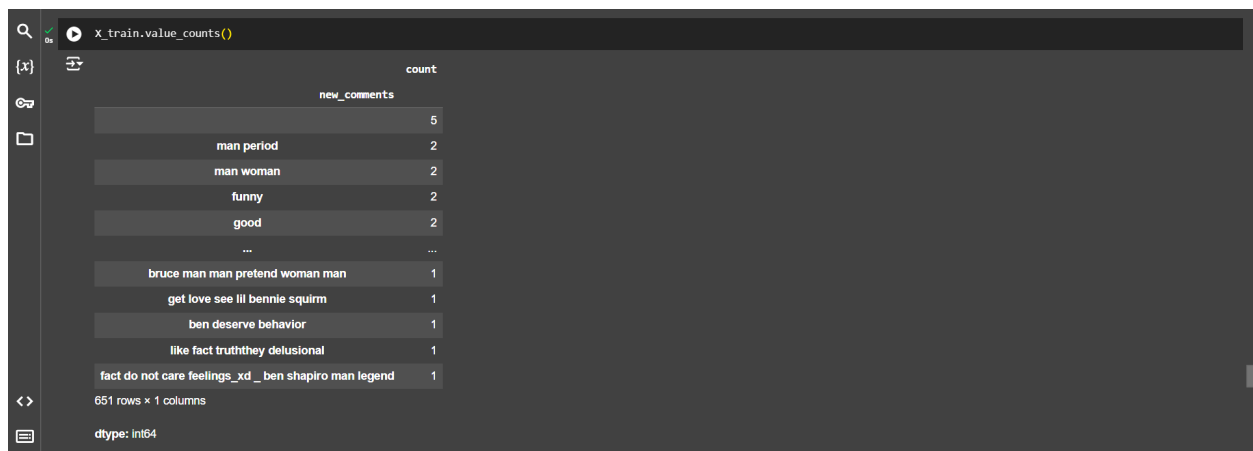
- **Term Frequency (TF)** measures how often a term appears in a document.
- **Inverse Document Frequency (IDF)** measures how common or rare a term is across all documents. Words that appear frequently across many documents (e.g., "the," "and") get lower weights.
- Together, the TF-IDF score reflects how important a word is to a specific document in the context of the whole dataset.

3. Fitting and Transforming Training Data

- Here, `fit_transform()` is called on the training data (`X_train`).
 - `fit` learns the vocabulary from the training data and calculates the IDF for each term.
 - `transform` converts the training data into a sparse matrix of TF-IDF features.
- This results in `X_train_vec`, a matrix where each row represents a document, and each column represents a unique term in the vocabulary. The values in the matrix are the TF-IDF scores of each term in each document.

4. Transforming Test Data

- In this step, `transform()` is called on the test data (`X_test`) without fitting it again.
- This uses the vocabulary and IDF values learned from the training data to transform `X_test` into the same TF-IDF matrix format as `X_train_vec`. This ensures that the model receives input in a consistent format for both training and testing.



| | count |
|---|-------|
| new_comments | 5 |
| man period | 2 |
| man woman | 2 |
| funny | 2 |
| good | 2 |
| ... | ... |
| bruce man man pretend woman man | 1 |
| get love see ill bennie squirm | 1 |
| ben deserve behavior | 1 |
| like fact truththey delusional | 1 |
| fact do not care feelings_xd_ben shapiro man legend | 1 |

651 rows x 1 columns

dtype: int64

`X_train.value_counts()` would count the occurrences of each unique row in the `X_train` Series or DataFrame. This method is often used to understand the distribution of unique entries in a dataset. Here's a breakdown of how it works and where it might be used:

Explanation:

- **`X_train`:** This is typically your training data, which may contain text data or other features.
 - If `X_train` is a Series (like a single column of text data), `X_train.value_counts()` will count the number of times each unique value (e.g., a specific sentence or text snippet) appears in the training set.

- If `X_train` is a DataFrame, `X_train.value_counts()` will count the occurrences of each unique row in the entire DataFrame (this is more rare and only supported in pandas 1.1.0 and later).

Logistic Regression Model:

```

# Now use the transformed data for training
lr = LogisticRegression()
lr.fit(X_train_vec, y_train)

LogisticRegression
LogisticRegression()

[29] #now lets predict for test data
y_pred=lr.predict(X_test_vec)
pred_df=pd.DataFrame(data={'tested_comments':X_test,'y_predicted':y_pred,'y_actual':y_test})
pred_df.head()

```

| | tested_comments | y_predicted | y_actual |
|-----|---|-------------|----------|
| 213 | testosterone laden woman_xd_display attribut... | 1.0 | 1.0 |
| 331 | | 1.0 | 0.0 |
| 500 | ladylike sir | 1.0 | 1.0 |
| 309 | be not calliyn bruce | 1.0 | 1.0 |
| 88 | testosterone never leave | 1.0 | 1.0 |

- `lr` is an instance of the `LogisticRegression` class from scikit-learn.
- `LogisticRegression()` initializes a logistic regression classifier.
- Logistic regression is a commonly used algorithm for classification tasks. Despite the name, it's mainly used to predict categorical outcomes.
- `.fit()` is the method used to train the model. Here, the logistic regression model learns from the training data.
- `X_train_vec` is the TF-IDF-transformed version of the training data. It's a matrix where each row represents a document, and each column represents a term in the vocabulary, with values representing the TF-IDF scores.
- `y_train` is the target labels (the correct classes for each document in `X_train_vec`), which the model uses to learn patterns in the data.
- `y_pred` stores the predictions made by the logistic regression model for each entry in the test set.
- `X_test_vec` is the test data that has been transformed using `TfidfVectorizer`, matching the format used to train the model.

CODE: `pred_df = pd.DataFrame(data={'tested_comments': X_test, 'y_predicted': y_pred, 'y_actual': y_test})`

This line creates a new `DataFrame` called `pred_df`, which helps you see how the model's predictions compare to the actual test labels.

`tested_comments` contains the original test comments from `X_test` (i.e., the text data before transformation).

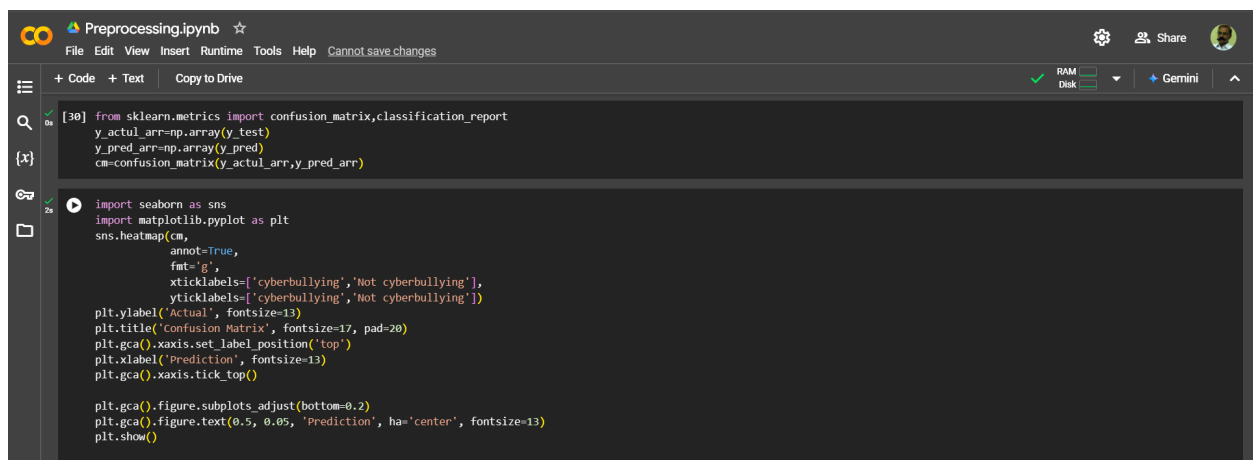
`y_predicted` contains the model's predictions for each comment.

`y_actual` contains the actual labels for each comment, so you can see whether the prediction (`y_predicted`) matches the true label (`y_actual`).

CODE: `pred_df.head()`

This `pred_df` `DataFrame` provides a quick and interpretable way to:

- Check the accuracy of your model by visually comparing predictions.
- Analyze specific misclassifications by examining the text in `tested_comments` where `y_predicted` and `y_actual` differ.
- `.head()` displays the first few rows of `pred_df` to give you a quick look at how well the model performed on the test data.



```
[30] from sklearn.metrics import confusion_matrix, classification_report
y_actul_arr=np.array(y_test)
y_pred_arr=np.array(y_pred)
cm=confusion_matrix(y_actul_arr,y_pred_arr)

import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['cyberbullying','Not cyberbullying'],
            yticklabels=['cyberbullying','Not cyberbullying'])
plt.ylabel('Actual', fontsize=13)
plt.title('Confusion Matrix', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Prediction', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Prediction', ha='center', fontsize=13)
plt.show()
```

Code Breakdown

- `confusion_matrix` and `classification_report` are imported from `sklearn.metrics`, which provides tools for evaluating classification models.
 - `confusion_matrix`: Provides a matrix that shows the counts of true positive, true negative, false positive, and false negative predictions.

- **classification_report**: Generates a report that includes precision, recall, and F1 score for each class.
- `y_actual_arr` and `y_pred_arr` convert `y_test` and `y_pred` into numpy arrays. This step isn't strictly necessary (since `confusion_matrix` can handle lists or pandas Series directly), but it can sometimes ensure compatibility with certain metrics functions.
- `cm` stores the confusion matrix, which compares the actual labels (`y_actual_arr`) and predicted labels (`y_pred_arr`).
- This matrix helps in evaluating how well the model performed by showing the count of correct and incorrect predictions for each class.

Interpreting the Confusion Matrix

The confusion matrix `cm` will be a 2x2 matrix if this is a binary classification problem. It has the following format:

| | |
|-----------------|-----------------|
| True Negatives | False Negatives |
| False Positives | True Positives |

This code visualizes the confusion matrix `cm` using a heatmap, which makes it easier to interpret the model's performance in a visually intuitive way. Here's a breakdown of each part:

Code : `import seaborn as sns`

`import matplotlib.pyplot as plt`

- `seaborn` is a data visualization library built on top of Matplotlib that provides an easier and more visually appealing way to create plots.
- `matplotlib.pyplot` is the standard plotting library in Python.

Code : `sns.heatmap(cm, annot=True, fmt='g', xticklabels=['cyberbullying', 'Not cyberbullying'], yticklabels=['cyberbullying', 'Not cyberbullying'])`

- `sns.heatmap(cm, ...)` creates a heatmap based on the confusion matrix `cm`.
- `annot=True` displays the values in each cell of the heatmap.
- `fmt='g'` formats the annotations as integers instead of scientific notation.
- `xticklabels` and `yticklabels` provide labels for the x-axis (predicted classes) and y-axis (actual classes) to make it clear which class each axis represents.

Code : `plt.ylabel('Actual', fontsize=13)`

`plt.title('Confusion Matrix', fontsize=17, pad=20)`

- `plt.ylabel('Actual', fontsize=13)` labels the y-axis as "Actual" with font size 13.
- `plt.title('Confusion Matrix', fontsize=17, pad=20)` gives the plot a title with font size 17 and some padding above the title.

Code : `plt.gca().xaxis.set_label_position('top')`

`plt.xlabel('Prediction', fontsize=13)`

`plt.gca().xaxis.tick_top()`

- `plt.gca()` gets the current axis for customization.
- `xaxis.set_label_position('top')` and `xaxis.tick_top()` place the x-axis label and ticks at the top of the plot, which is often helpful for confusion matrices.

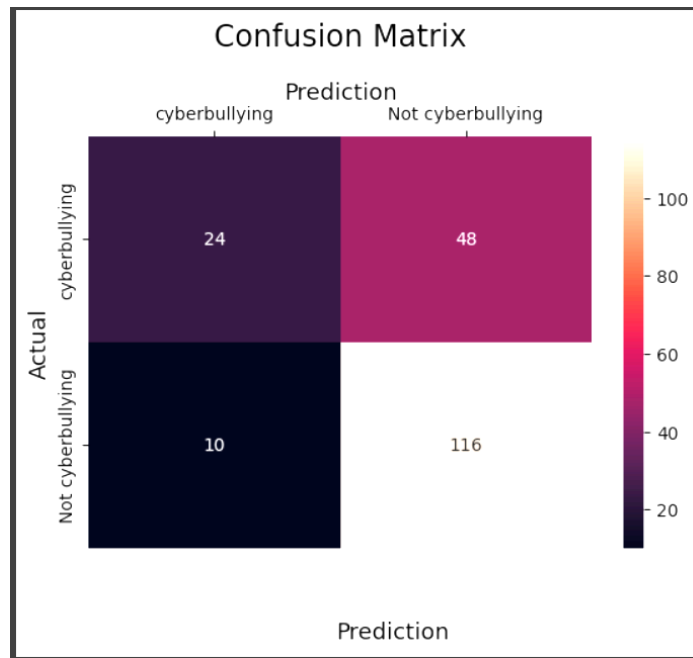
Code : `plt.gca().figure.subplots_adjust(bottom=0.2)`

`plt.gca().figure.text(0.5, 0.05, 'Prediction', ha='center', fontsize=13)`

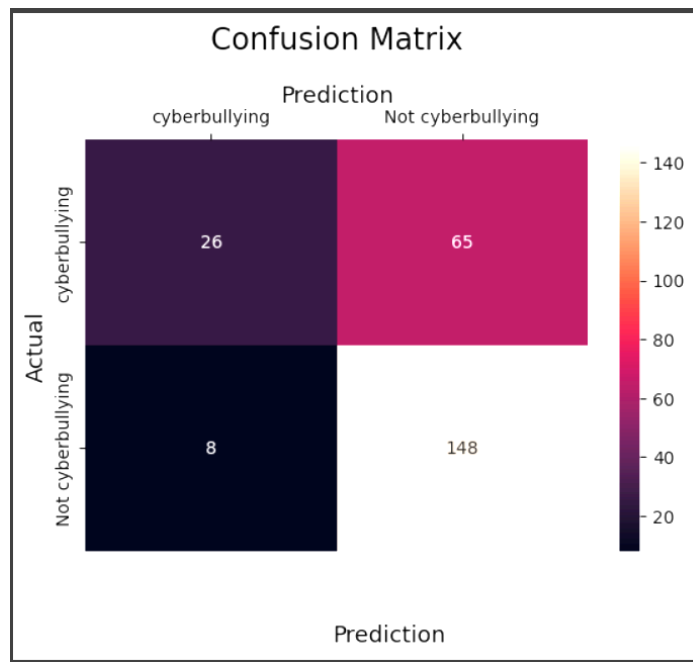
`plt.show()`

- `subplots_adjust(bottom=0.2)` provides a bit of space at the bottom of the plot.
- `figure.text(...)` adds a text label "Prediction" below the plot for additional clarity.
- `plt.show()` displays the heatmap.

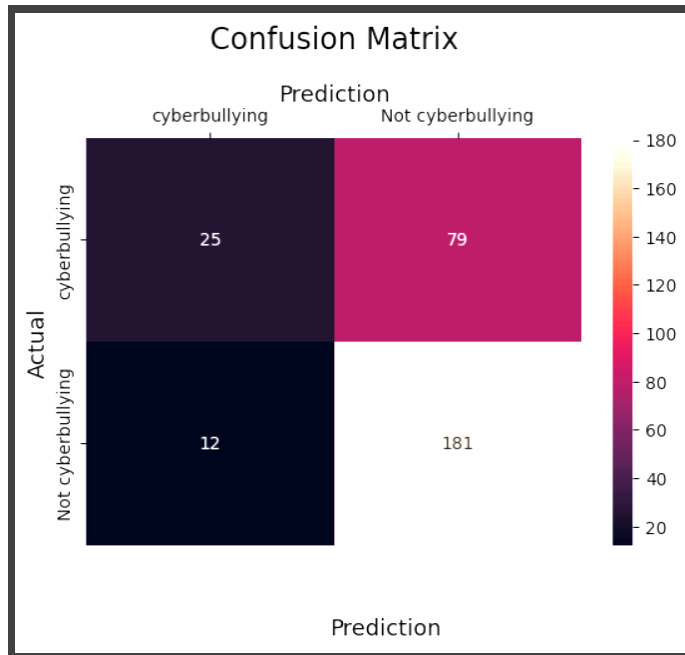
CONFUSION MATRIX:



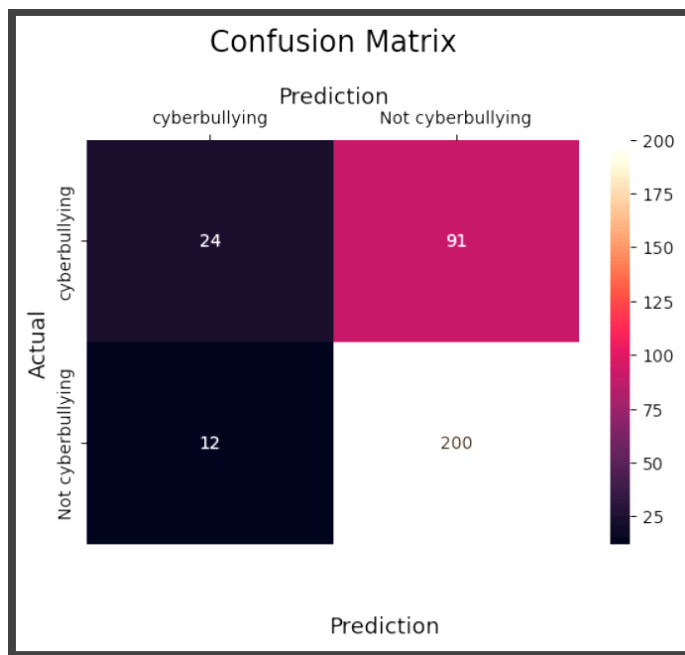
TEST SIZE : 0.2



TEST SIZE : 0.25



TEST SIZE : 0.3



TEST SIZE : 0.33

The `classification_report` provides detailed metrics for evaluating the performance of a classification model on each class.

CODE: `print(classification_report(y_actual_arr, y_pred_arr))`

Output Metrics Explained

- 1 . **Precision:** Measures the proportion of true positive predictions among all positive predictions (i.e., the accuracy of positive predictions).
 - Formula: $\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$
 - A high precision score for a class indicates that when the model predicts that class, it is often correct.
- 2 . **Recall:** Measures the proportion of true positives identified among all actual positives (i.e., the ability to find all positive instances).
 - Formula: $\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$
 - A high recall score for a class indicates that the model is good at identifying most instances of that class.
- 3 . **F1 Score:** The harmonic mean of precision and recall. It's a balanced measure that gives insight into the model's performance across both metrics.
 - Formula: $\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})}$
 - A high F1 score indicates a good balance between precision and recall.
- 4 . **Support:** The number of actual instances in each class. It helps give context to precision and recall scores by showing how many true examples of each class were in the dataset.

The `classification_report` provides a comprehensive view of model performance, especially useful when evaluating a classifier in a multi-class or imbalanced data context.

The `accuracy_score` function calculates the overall accuracy of the model's predictions, which is the ratio of correct predictions to the total number of predictions.

`accuracy_score(y_test, y_pred)` computes the accuracy by comparing the true labels (`y_test`) to the predicted labels (`y_pred`).

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

ACCURACY RATES:

TEST SIZE : 0.2 - 70.70 %

```
[33] print(classification_report(y_actul_arr, y_pred_arr))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0.0 | 0.71 | 0.33 | 0.45 | 72 |
| 1.0 | 0.71 | 0.92 | 0.80 | 126 |
| accuracy | | | 0.71 | 198 |
| macro avg | 0.71 | 0.63 | 0.63 | 198 |
| weighted avg | 0.71 | 0.71 | 0.67 | 198 |

```
<> from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.7070707070707071
```

TEST SIZE : 0.25 - 70.45 %

```
print(classification_report(y_actul_arr, y_pred_arr))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0.0 | 0.76 | 0.29 | 0.42 | 91 |
| 1.0 | 0.69 | 0.95 | 0.80 | 156 |
| accuracy | | | 0.70 | 247 |
| macro avg | 0.73 | 0.62 | 0.61 | 247 |
| weighted avg | 0.72 | 0.70 | 0.66 | 247 |

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.7044534412955465
```

TEST SIZE : 0.3 - 69.36 %

```
[69] print(classification_report(y_actul_arr, y_pred_arr))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0.0 | 0.68 | 0.24 | 0.35 | 104 |
| 1.0 | 0.70 | 0.94 | 0.80 | 193 |
| accuracy | | | 0.69 | 297 |
| macro avg | 0.69 | 0.59 | 0.58 | 297 |
| weighted avg | 0.69 | 0.69 | 0.64 | 297 |

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.6936026936026936
```

TEST SIZE: 0.33 - 68.50 %

```
[105] print(classification_report(y_actul_arr, y_pred_arr))
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0.0 | 0.67 | 0.21 | 0.32 | 115 |
| 1.0 | 0.69 | 0.94 | 0.80 | 212 |
| accuracy | | | 0.69 | 327 |
| macro avg | 0.68 | 0.58 | 0.56 | 327 |
| weighted avg | 0.68 | 0.69 | 0.63 | 327 |

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.6850152905198776
```

Random Forest :

```
{x} RANDOMFOREST CLASSIFIER
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(X_train_vec,y_train)

RandomForestClassifier
RandomForestClassifier()

[108] y_rfc_pred=rfc.predict(X_test_vec)
y_rfc_pred_arr=np.array(y_rfc_pred)
cm=confusion_matrix(y_actul_arr,y_rfc_pred_arr)
```

```
[109] import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['cyberbullying','Not cyberbullying'],
            yticklabels=['cyberbullying','Not cyberbullying'])
plt.ylabel('Actual', fontsize=13)
plt.title('Confusion Matrix', fontsize=17, pad=20)
plt.gca().xaxis.set_label_position('top')
plt.xlabel('Prediction', fontsize=13)
plt.gca().xaxis.tick_top()

plt.gca().figure.subplots_adjust(bottom=0.2)
plt.gca().figure.text(0.5, 0.05, 'Prediction', ha='center', fontsize=13)
plt.show()
```

Code Explanation :

- `RandomForestClassifier()` initializes a random forest model, a type of ensemble method that combines multiple decision trees to improve classification performance.
- The `RandomForestClassifier` can handle high-dimensional data and is robust to overfitting.

CODE: `rfc.fit(X_train_vec, y_train)`

- This trains the random forest classifier (`rfc`) on the TF-IDF-transformed training data (`X_train_vec`) and corresponding labels (`y_train`).
- Each decision tree in the forest will learn patterns in the TF-IDF feature space to distinguish between classes.

CODE: `y_rfc_pred = rfc.predict(X_test_vec)`

- This line generates predictions for the test data (`X_test_vec`) using the trained random forest model.
- The result, `y_rfc_pred`, contains the predicted class labels for each entry in the test set.

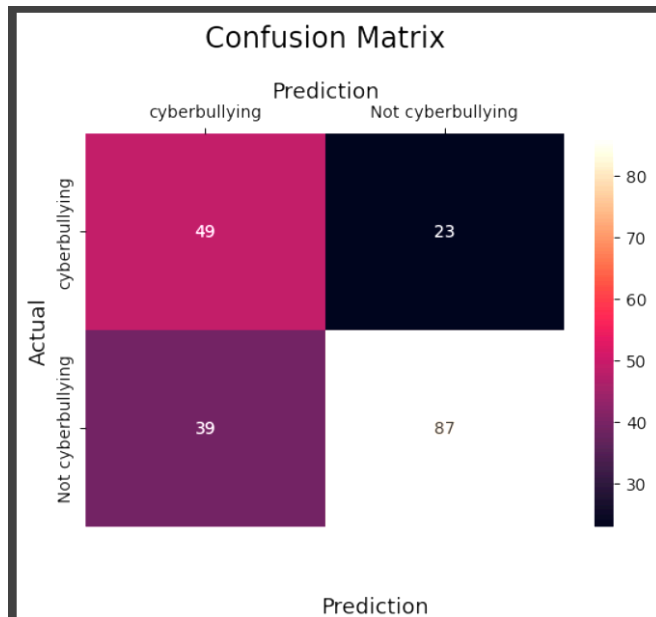
CODE: `y_rfc_pred_arr = np.array(y_rfc_pred)`

- Converts `y_rfc_pred` to a numpy array, `y_rfc_pred_arr`, which can make it easier to compare with `y_actual_arr` in certain metrics.

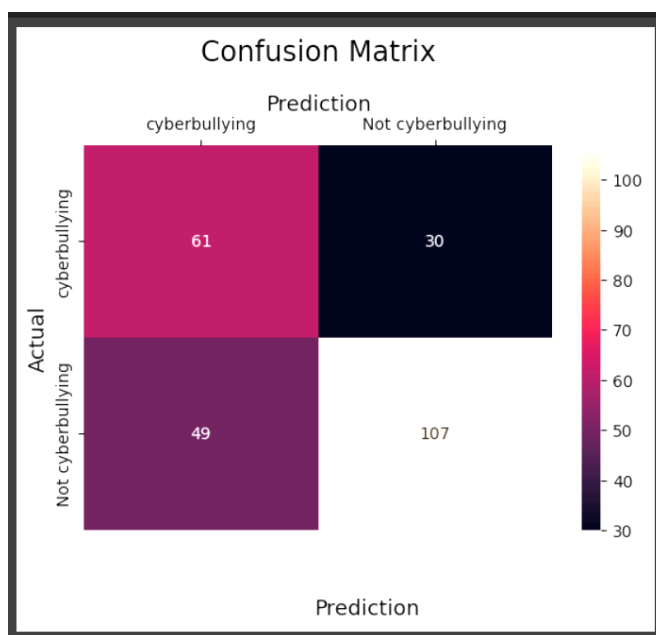
CODE: `cm = confusion_matrix(y_actual_arr, y_rfc_pred_arr)`

- `cm` stores the confusion matrix for the random forest predictions compared to the actual test labels.
- This matrix helps in analyzing the model's performance in terms of true positives, true negatives, false positives, and false negatives.

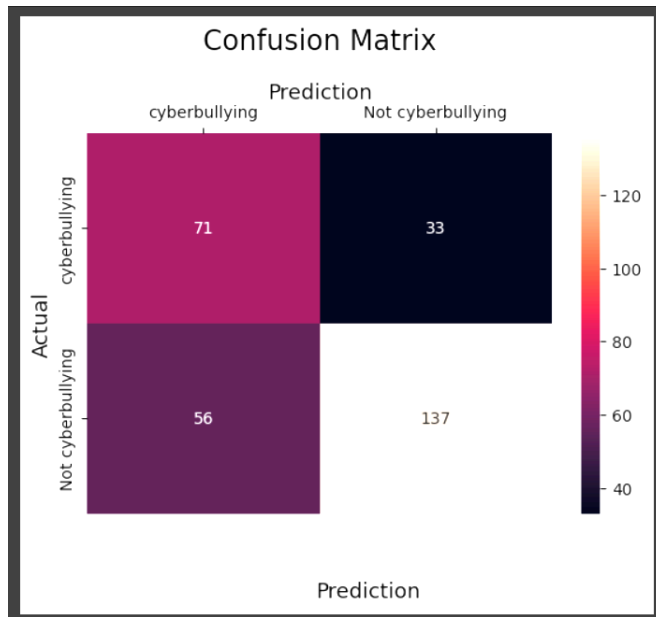
CONFUSION MATRIX :



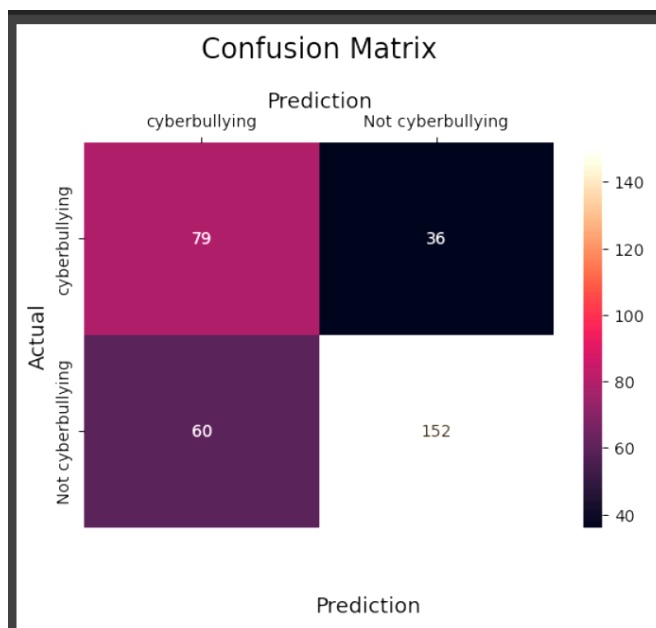
TEST SIZE : 0.2



TEST SIZE : 0.25



TEST SIZE : 0.3



TEST SIZE : 0.33

ACCURACY RATES :

TEST SIZE : 0.2 - 68.68 %

```
0a accuracy_score(y_test, y_rfc_pred)
0.68686868686869
```

TEST SIZE : 0.25 - 68.01%

```
+ Code + Text
0a [148] accuracy_score(y_test, y_rfc_pred)
0.680161943319838
```

TEST SIZE : 0.3 - 70.03 %

```
0a [130] accuracy_score(y_test, y_rfc_pred)
0.7003367003367004
```

TEST SIZE : 0.33 - 71.25 %

```
0a [184] accuracy_score(y_test, y_rfc_pred)
0.7125382262996942
```