Total Marks: 30 Marks

About the data set (COMPANY_Reviews)

The dataset contains Text reviews. The aim of this assignment is to classsify the review texts into two categories.

Attribute information:

Dataset has two columns: Column2: Review Text Column1: Two Classes. Positive and negative.

Table of Content

Introduction:

- Objective: To classify text reviews into two distinct categories: positive and negative sentiments.
- Dataset: Textual review data with two attributes: review text and sentiment label (Positive or Negative).
- Goal: To gain valuable insights into the sentiment conveyed by these reviews, which can be instrumental in various applications, such as customer feedback analysis and product sentiment tracking.

1. Import Libraries

Let us import the required libraries.

```
25...
import numpy as np
import re
import pandas as pd
from nltk.corpus import reuters
from string import punctuation
from nltk.corpus import stopwords
from nltk import word_tokenize
from sklearn.preprocessing import LabelEncoder
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score
```

Inference:

Importing libraries for data analysis and machine learning:

- numpy (np): numerical operations and array handling
- re: regular expressions for text preprocessing
- pandas (pd): data manipulation, loading, and exploration
- nltk: natural language processing tools, including stopwords and a Porter Stemmer
- string.punctuation: punctuation removal from text
- sklearn: machine learning tasks, including data preprocessing, feature extraction, model building, and evaluation
- sklearn.preprocessing.LabelEncoder: encoding the 'Sentiment' column into numerical values
- nltk.stem.PorterStemmer: word stemming
- sklearn.feature_extraction.text.CountVectorizer and sklearn.feature_extraction.text.TfidfVectorizer: converting text data into numerical feature vectors using Bag of Words (BoW) and TF-IDF (Term Frequency-Inverse Document Frequency) approaches
- sklearn.model_selection.train_test_split: splitting the dataset into training and testing sets

- sklearn.naive_bayes.GaussianNB: classification model
- sklearn.metrics.confusion_matrix and sklearn.metrics.accuracy_score: evaluating the classification model's performance

2.1 Read the data

```
df= pd.read_csv("sentiment_analysis.csv",encoding=" ISO-8859-1")
```

2.2 Dispay the first ten rows

```
df.head(10)
```

	Sentiment	Review
0	negative	The international electronic industry company ...
1	positive	With the new production plant the company woul...
2	positive	According to the company 's updated strategy f...
3	positive	FINANCING OF ASPOCOMP 'S GROWTH Aspocomp is ag...
4	positive	For the last quarter of 2010 , Componenta 's n...
5	positive	In the third quarter of 2010 , net sales incre...
6	positive	Operating profit rose to EUR 13.1 mn from EUR ...
7	positive	Operating profit totalled EUR 21.1 mn , up fro...
8	positive	TeliaSonera TLSN said the offer is in line wit...
9	positive	STORA ENSO , NORSKE SKOG , M-REAL , UPM-KYMMEN...

2.3 Display the information about the data

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1967 entries, 0 to 1966
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Sentiment  1967 non-null   object
 1   Review     1967 non-null   object
dtypes: object(2)
memory usage: 30.9+ KB
```

2.4 Encode the sentiment column values as 1 or 0

```
lbl= LabelEncoder()
df['Sentiment_enc']=lbl.fit_transform(df['Sentiment'])
df
```

	Sentiment	Review	Sentiment_enc
0	negative	The international electronic industry company ...	0
1	positive	With the new production plant the company woul...	1
2	positive	According to the company 's updated strategy f...	1
3	positive	FINANCING OF ASPOCOMP 'S GROWTH Aspocomp is ag...	1
4	positive	For the last quarter of 2010 , Componenta 's n...	1
...	...	...	...
1962	negative	HELSINKI Thomson Financial - Shares in Cargote...	0
1963	negative	LONDON MarketWatch -- Share prices ended lower...	0
1964	negative	Operating profit fell to EUR 35.4 mn from EUR ...	0
1965	negative	Net sales of the Paper segment decreased to EU...	0
1966	negative	Sales in Finland decreased by 10.5 % in Januar...	0

1967 rows × 3 columns

Inference:

Load and explore the data:

- Load the dataset using the pd.read_csv() function.
- Display the first ten rows of the dataset.
- Use the info() function to get an overview of the dataset.
- Encode the Sentiment column, converting positive to 1 and negative to 0.
- In this step, we utilized label encoding to convert sentiment labels ('positive' and 'negative') into numerical values (1 and 0).
- Label Encoding: Label encoding is a technique used to convert categorical labels into numerical values, making it easier for machine learning algorithms to process them.

3. Data preprocessing

3.1 Remove special characters and html tags

3.2 - Convert reviews into lowercase

3.3 - Removal of stop words

3.4 - Apply stemming

```
corpus=[]
for i in range(0, 1967):
    review = re.sub('[^a-zA-Z]', ' ', df['Review'][i])
    review=review.lower()
    review = review.split()
    ps = PorterStemmer()
    all_stopwords = stopwords.words('english')
    review = [ps.stem(word) for word in review if not word in set(all_stopwords)]
    review = ' '.join(review)
    corpus.append(review)
```

```
corpus
```

```
['intern electron industri compani elcoteq laid ten employe tallinn facil contrari earlier layoff compani contract rank offic worker daili postime report',
 'new product plant compani would increas capac meet expect increas demand would improv use raw materi therefor increas product profit',
 'accord compani updat strategi year baswar target long term net sale growth rang oper profit margin net sale',
 'financ aspocomp growth aspocomp aggress pursu growth strategi increasingli focus technolog demand hdi print circuit board pcb',
 'last quarter componenta net sale doubl eur eur period year earlier move zero pre tax profit pre tax loss eur',
 'third quarter net sale increas eur mn oper profit eur mn',
 'oper profit rose eur mn eur mn correspond period repres net sale',
 'oper profit total eur mn eur mn repres net sale',
 'teliasonera tlsn said offer line strategi increas ownership core busi hold would strengthen eesti telekom offer custom',
 'stora enso norsk skog real upm kymmen credit suiss first boston cfsb rais fair valu share four largest nordic forestri group',
 'purchas agreement ton gasolin deliveri hamina termin finland sign nest oil oyj averag platt index septemb plu eight us dollar per month',
 'finnish talentum report oper profit increas eur mn eur mn net sale total eur mn eur mn',
 'cloth retail chain sepp l sale increas eur mn oper profit rose eur mn eur mn',
 'consolid net sale increas reach eur oper profit amount eur compar loss eur prior year period',
```

Inference:

Preprocess the text data:

- Remove special characters and HTML tags from the review texts.
- Convert all text to lowercase.
- Apply stemming using the Porter Stemmer.
- Remove stop words using regular expressions.

Explanation:

- Removing special characters and HTML tags: This helps to clean the data and make it more consistent.

- Converting all text to lowercase: This ensures that all words are treated equally, regardless of their capitalization.
- Applying stemming using the Porter Stemmer: This reduces words to their root form, which can help to improve the performance of text processing algorithms.
- Removing stopwords using regular expressions: Stop words are common words that do not add much meaning to the text, such as "the", "and", and "in". Removing stop words can help to reduce noise and focus on more meaningful words. 3.1 Remove Special Character and
- HTML tags

```
def clean_text(text):
    text = re.sub('<.*?>', '', text)  # Remove HTML tags
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special characters
    return text

df['Review'] = df['Review'].apply(clean_text)
```

- Inference:
- The provided code defines a function called clean_text and then applies this function to the
- "Review" column of your DataFrame (df).
- **1. Function Definition (def clean_text(text):):o def clean_text(text): defines a Python function named clean_text. This function takes one argument called text, which represents the input text that needs to be cleaned.**
- **2. Text Cleaning Steps Inside the Function:o text = re.sub('<.*?>', '', text): This line uses the re.sub() function from the**
- **Python re module (regular expressions) to remove HTML tags from the input text. The regular expression <.*?> matches any HTML tag (e.g., <p>, , etc.) and replaces them with an empty string, effectively removing them.o text = re.sub(r'[^a-zA-Z\s]', '', text): This line uses another re.sub() call to remove special characters and symbols from the text. The regular expression**
- **[^a-zA-Z\s] matches any character that is not a letter (upper or lower case), a digit, or a whitespace character. It replaces these special characters with an empty string, effectively removing them.**
- **3. Applying the Cleaning Function to the "Review" Column:o df['Review'] = df['Review'].apply(clean_text): This line applies the clean_text function to each element (review text) in the "Review" column of your**
- **DataFrame (df). It cleans the text by removing HTML tags and special characters and then updates the "Review" column with the cleaned text.**

```
df['Review'] = df['Review'].str.lower()
```

- **<p>This is an example$<p>This is an example3.2 Convert reviews into lowercase**
- **The line of code df['Review'] = df['Review'].str.lower() is used to convert all the text in the**
- **"Review" column of your DataFrame df to lowercase.**

- **df['Review']: This part of the code specifies the column of the DataFrame that you want to work with, which is the "Review" column. It selects this column from the DataFrame and allows you to perform operations on its elements.**
- **.str.lower(): This is a pandas Series method called str.lower(). When applied to a**
- **Series (in this case, the "Review" column), it converts all the text (strings) within that Series to lowercase.**
- **Example:**
- **This is an examplethis is an example**

```
import nltk
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
df['Review'] = df['Review'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]))
```

- 3.3 Removal of stop words
- Input Sentence: "this is an example"
- After removing stopwords, the Output Sentence: "example"
- In this case, the words "this," "is," and "an" are considered stopwords and have been removed from the original sentence, leaving only the word "example" in the output.
- Stop word removal is a common preprocessing step in natural language processing to eliminate words that often do not contribute significantly to the meaning of a sentence or text.
- import nltk: This line imports the NLTK library, which is a popular Python library for natural language processing (NLP) tasks. NLTK provides various tools and resources for working with text data.
- nltk.download('stopwords'): NLTK includes a collection of datasets and resources that need to be downloaded separately. In this line, you are using the nltk.download()function to download the stopwords dataset for the English language. Stopwords are common words like "the," "and," "is," "in," etc., that are often removed from text data during text preprocessing because they typically do not carry significant meaning and can be noise in NLP tasks.
- stop_words = set(stopwords.words('english')): After downloading the English stopwords dataset, you create a set called stop_words that contains these stopwords.
- The stopwords.words('english') function call retrieves the list of English stopwords from NLTK's dataset. By converting this list into a set, you make it more efficient for later use because checking for membership in a set is faster than in a list.
- 3.4 Applying stemming

After applying stemming: i love product its amaz

```
stemmer = PorterStemmer()
df['Review'] = df['Review'].apply(lambda x: ' '.join([stemmer.stem(word) for word in x.split()]))
```

- Stemming is the process of reducing words to their root form or base form. It's often used to normalize words so that different variations of the same word are treated as the same word.
- However, applying stemming multiple times in succession is not typically necessary, as it can result in over-stemming and may lead to a loss of meaning.

- Original Review: "I loved the product, it's amazing!"
- In this example, the Porter Stemmer has reduced words to their root forms. Here's how some of the words were stemmed:
- □ "loved" -> "love"
- □ "amazing" -> "amaz"
- After applying stemming: i love product its amaz□ stemmer = PorterStemmer(): This line initializes an instance of the Porter Stemmer from the NLTK library. The Porter Stemmer is a commonly used stemming algorithm.
- □ df['Review'] = df['Review'].apply(lambda x: ' '.join([stemmer.stem(word) for word in x.split()])): This line applies stemming to the "Review" column of your DataFrame. It splits each review text into individual words, applies stemming to each word, and then joins the stemmed words back into a single string. Essentially, it's stemming each word in the "Review" column

4. Convert review text into feature vector and classification

4.1 - create BOW model

```
cv = CountVectorizer()
X = cv.fit_transform(corpus).toarray()
```

```
X
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

```
y = df.iloc[:, -1].values
```

```
y
```

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

Inference:

Step 4: Build and evaluate classification models

4.1 Create a Bag of Words (BoW) model:

- Purpose: To create a numerical representation of the text data that can be used by machine learning algorithms.
- Tool: CountVectorizer

- Output: A feature matrix X where each row represents a review and each column represents a word in the vocabulary. The value of each cell in the matrix is the frequency of the corresponding word in the review.
- Explanation:
- The BoW model is a simple way to represent text data numerically.
- It works by counting the frequency of each word in a document.
- This results in a feature vector for each document, where each feature represents a word in the vocabulary and the value of each feature is the frequency of the corresponding word in the document.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 10)
```

Inference:

4.2 Split the data into training and test sets:

Data splitting:

- Purpose: To divide the data into training and test sets to evaluate the performance of the classification model.
- Tool: train_test_split()
- Splitting: Random and representative split
- Sets: Training set and test set
- Features: Used for predictions
- Labels: What we're trying to predict
- Importance: Separating features and labels is essential for evaluating the model's effectiveness.
- Training set: The model learns and improves its predictive capabilities on the training set.
- Test set: The model's performance is evaluated on unseen data in the test set.
- Validation: Data splitting ensures that the model can make accurate predictions in real-world scenarios.

4.3 - Define the Classification model (any one like NB, SVM or Random forest) train the model

```
classifier = GaussianNB()
classifier.fit(X_train, y_train)


y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[1 1]
 [1 1]
 [1 1]
 ...
 [1 1]
 [0 1]
 [0 1]]
```

4.3 Define and train a Random Forest classifier:

- Define and instantiate a Random Forest classifier with 30 decision trees.
- Train the model on the training data using fit().
- Make predictions on the test set using predict().1. classifier = GaussianNB(): This line initializes a Gaussian Naive Bayes classifier.
- GaussianNB is suitable for classification tasks when the features follow a Gaussian (normal) distribution.
- 2. classifier.fit(X_train, y_train): This line trains the Gaussian Naive Bayes classifier on the training data. X_train represents the feature vectors (independent variables) of the training data, and y_train represents the corresponding target labels (dependent variable).
- 3. y_pred = classifier.predict(X_test): After training, this line predicts the target labels for the test data using the trained classifier. X_test contains the feature vectors of the test data, and y_pred will contain the predicted labels.
- 4. print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1)): This line prints the predicted labels (y_pred) and the actual labels (y_test) side by side for comparison. It uses NumPy to concatenate thetwo arrays vertically to create a 2-column output where the first column represents the predicted labels and the second column represents the actual labels.
- The printed output will show pairs of predicted and actual labels for the test data, allowing you to evaluate the performance of the classifier by comparing its predictions to the true labels

4.4 Classification metrics analysis and plot the confusion matrix

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[134  47]
 [137 273]]
```

```
accuracy_score(y_test, y_pred)
```

```
0.688663282571912
```

Inference:

4.4 Evaluate the Random Forest classifier:

- Calculate classification metrics such as precision, recall, F1-score, and support using classification_report.
- These metrics provide a comprehensive evaluation of the model's performance. A confusion matrix is a table that summarizes the performance of a classification model. The rows of the matrix represent the actual class labels, and the columns represent the predicted class labels. Each cell in the matrix contains the number of instances that were predicted to be in a particular class, given that they were actually in another class.
- In this example, the confusion matrix shows the performance of a classification model that is trying to predict whether a customer is satisfied or not satisfied with a product. The rows of the matrix represent the actual customer satisfaction, and the columns represent the predicted customer satisfaction.
- Inference:
- The first row of the matrix shows that the model correctly predicted that 131 customers were satisfied with the product. However, it also incorrectly predicted that 50 customers were satisfied with the product, when they were actually not satisfied.
- The second row of the matrix shows that the model correctly predicted that 277 customers were not satisfied with the product. However, it also incorrectly predicted that 133 customers were not satisfied with the product, when they were actually satisfied.
- The confusion matrix can be used to calculate a number of different performance metrics,

  such as accuracy, precision, recall, and F1 score.

- In this example, the accuracy of the model is 69%, meaning that it correctly predicted the customer satisfaction of the customers.
- The confusion matrix can also be used to identify areas where the model can be improved.
- For example, in this example, the model is more likely to incorrectly predict that a customer is satisfied, when they are actually not satisfied. This suggests that the model may need to be better trained to identify the characteristics of customers that are likely to be not satisfied.

4.5 - predict the class for your own review

```
new_review = 'The company laidoff the employee'
new_review = re.sub('[^a-zA-Z]', ' ', new_review)
new_review = new_review.lower()
new_review = new_review.split()
ps = PorterStemmer()
all_stopwords = stopwords.words('english')
all_stopwords.remove('not')
new_review = [ps.stem(word) for word in new_review if not word in set(all_stopwords)]
new_review = ' '.join(new_review)
new_corpus = [new_review]
new_X_test = cv.transform(new_corpus).toarray()
new_y_pred = classifier.predict(new_X_test)
print(new_y_pred)
```

```
[0]
```

Inference:

4.5 Predict the class for a new review (Naive Bayes):

- Preprocess the new review text in the same way as the dataset.
- Convert the preprocessed text into a feature vector using the BoW model.
- Pass the feature vector to the Naive Bayes classifier for prediction.
- Print the predicted class (0 for negative or 1 for positive) for the new review.
- We explained the scenario of predicting sentiment for new reviews, where a negative word like "laidoff" leads to a prediction of 0 (negative), and a positive word like "increase" results in a prediction of 1 (positive).
- This illustrates how the model classifies text based on learned patterns.

  Consider my own review to be “The company laid off the employee” after doing all the preprocessing steps and passing into the model. We are predicting it by passing the review.

  The model has correctly predicted as [0] negative class as the word laidoff is the negative sentiment in sentiment analysis.

4.6 - create feature vector using tf-idf

```
from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tv_matrix = tv.fit_transform(corpus)
tv_matrix = tv_matrix.toarray()

vocab = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)
```

Inference:

4.6 Create a Term Frequency-Inverse Document Frequency (TF-IDF) feature vector:

- Use TfidfVectorizer to transform the preprocessed text data into a TF-IDF feature matrix.
- The resulting matrix represents the importance of each term (word) in the corpus relative to the documents (reviews).
- TF (Term Frequency): Captures word importance within a document. It counts word occurrences without considering other documents.
- IDF (Inverse Document Frequency): Measures word distinctiveness. Words appearing in many documents get lower IDF weights, indicating their commonality.

The code you provided is using the TF-IDF (Term Frequency-Inverse Document Frequency) vectorization technique to convert the text data in the "Review" column of your DataFrame (df) into a numerical representation. TF-IDF is often used in text analysis to represent the importance of words in a document relative to a corpus of documents. Here's an explanation of the code:

1. from sklearn.feature_extraction.text import TfidfVectorizer: This line imports the TfidfVectorizer class from scikit-learn, which is used for TF-IDF vectorization.

2. tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True): Here, you create an instance of the TfidfVectorizer class with the following parameters:

o min_df=0.0: Specifies that a word must appear in at least 0% of the documents (no minimum threshold) to be included in the vocabulary.

o max_df=1.0: Specifies that a word must appear in at most 100% of the documents (no maximum threshold) to be included in the vocabulary.

o use_idf=True: Indicates that the Inverse Document Frequency (IDF) weighting should be applied.

3. tv_matrix = tv.fit_transform(df['Review']): This line fits the TfidfVectorizer to the "Review" column of your DataFrame (df) and transforms the text data into a TF-IDF matrix. Each row in the matrix represents a document (review), and each column represents a unique word in the entire corpus.

4. tv_matrix = tv_matrix.toarray(): The TF-IDF matrix is converted to a NumPy array for easier manipulation and inspection.

5. vocab = tv.get_feature_names(): This line retrieves the feature names (words) from the TF-IDF vectorizer. These feature names represent the words that make up the vocabulary used for TF-IDF vectorization.

6. pd.DataFrame(np.round(tv_matrix, 2), columns=vocab): Finally, this code creates a Pandas DataFrame from the TF-IDF matrix. It rounds the values to two decimal places for readability and assigns the feature names (words) as column names.

The resulting DataFrame contains TF-IDF values for each word in the vocabulary across all the reviews in your dataset. Each row represents a review, and each column represents a word, with TF-IDF values indicating the importance of each word in the corresponding review

4.7 Define the Classification model (any one like NB, SVM or Random forest) train the model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
```

```
# Create a Random Forest model with n_estimators=30
random_forest = RandomForestClassifier(n_estimators=30)

# Fit the model on the training data
random_forest.fit(X_train, y_train)

# Make predictions on the test set
y_pred = random_forest.predict(X_test)

# Generate a classification report
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.84      0.65      0.73       181
           1       0.86      0.94      0.90       410

    accuracy                           0.85       591
   macro avg       0.85      0.80      0.81       591
weighted avg       0.85      0.85      0.85       591
```

Inference:

4.7 Define and train a Random Forest classifier:

- random_forest = RandomForestClassifier(n_estimators=30): This line initializes a
- Random Forest classifier with 30 estimators. The n_estimators parameter specifies the number of decision trees in the forest, random_forest.fit(X_train, y_train): Here, the Random Forest model is fitted (trained)
  on the training data. X_train represents the feature vectors of the training data, and
- y_train represents the corresponding target labels.
  - y_pred = random_forest.predict(X_test): After training, the model is used to make predictions on the test set. X_test contains the feature vectors of the test data, and
- y_pred will contain the predicted labels.
  - report = classification_report(y_test, y_pred): This line generates a classification
- report by comparing the true labels (y_test) with the predicted labels (y_pred). The classification report includes metrics such as precision, recall, F1-score, and support for each class,print(report): Finally, the classification report is printed to the console, allowing you
  to assess the performance of the Random Forest model on your sentiment classification task.
- The classification report provides detailed information about the model's performance, including how well it identifies positive and negative sentiments. It's a valuable tool for

evaluating the quality of your classification model

4.8 - Classification metrics analysis

```
# Generate a classification report
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.84      0.65      0.73       181
           1       0.86      0.94      0.90       410

    accuracy                           0.85       591
   macro avg       0.85      0.80      0.81       591
weighted avg       0.85      0.85      0.85       591
```

Inference:

4.8 Predict the class for a new review :

- The classification report provides a detailed evaluation of a classification model's performance, including metrics such as precision, recall, F1-score, and support.
- Precision: Precision measures the accuracy of positive predictions made by the model. In a binary classification task (0 or 1), precision for class 0 is calculated as the ratio of true negatives to the sum of true negatives and false positives. For class 1, precision is calculated as the ratio of true positives to the sum of true positives and false positives. In your report:
- For class 0 (Negative), the precision is 0.79, which means that out of all predictions the model made as Negative, 79% were correct.
- For class 1 (Positive), the precision is 0.85, indicating that 85% of the predictions made as Positive were correct.
- Recall: Recall measures the model's ability to identify all relevant instances of a class. In a binary classification task, recall for class 0 is calculated as the ratio of true negatives to the sum of true negatives and false negatives. For class 1, recall is calculated as the ratio of true positives to the sum of true positives and false negatives.
- For class 0 (Negative), the recall is 0.64, meaning that the model correctly identified 64% of all actual Negative instances.
- For class 1 (Positive), the recall is 0.93, indicating that the model correctly identified 93% of all actual Positive instances.
- F1-Score: The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall. A higher F1-score indicates a better balance between precision and recall. In your report:
- For class 0 (Negative), the F1-score is 0.71.
- For class 1 (Positive), the F1-score is 0.89.
- Support: Support represents the number of actual occurrences of each class in the test dataset. In your report:
- For class 0 (Negative), there are 181 instances in the test data.

- For class 1 (Positive), there are 410 instances in the test data.
- Accuracy: Accuracy is the overall correctness of the model's predictions. It is the ratio of correctly predicted instances (both True Positives and True Negatives) to the total number of instances. In your report, the accuracy is 0.84, indicating that the model correctly classified 84% of the instances in the test dataset.
- Macro Avg: This row provides the average values of precision, recall, and F1-score across all classes. In your report, the macro-average precision is 0.82, macro-average recall is 0.78, and the macro-average F1-score is 0.80.
- Weighted Avg: This row provides the weighted average values of precision, recall, and F1-score, taking into account class imbalance. In your report, the weighted□average precision is 0.84, weighted-average recall is 0.84, and the weighted-average
- F1-score is 0.83.
- These metrics collectively give you a comprehensive view of how well your classification
- model is performing for each class and overall. In this case, the model seems to perform well, with higher precision and recall for the Positive class compared to the Negative class. The
- F1-score takes into account both precision and recall, and the weighted average F1-score is a good indicator of overall model performance, considering class imbalances if present

```
[49…]
new_review = 'The company laidoff the employee'
new_review = re.sub('[^a-zA-Z]', ' ', new_review)
new_review = new_review.lower()
new_review = new_review.split()
ps = PorterStemmer()
all_stopwords = stopwords.words('english')
all_stopwords.remove('not')
new_review = [ps.stem(word) for word in new_review if not word in set(all_stopwords)]
new_review = ' '.join(new_review)
new_corpus = [new_review]
new_X_test = cv.transform(new_corpus).toarray()
new_y_pred = classifier.predict(new_X_test)
print(new_y_pred)
```

```
[0]
```

Inference: In this case, the model predicts a sentiment of '0,' indicating a negative sentiment for the new review correctly as laid off refers to negative sentiment.

```
new_review1 = 'The company increased the production'
new_review1 = re.sub('[^a-zA-Z]', ' ', new_review1)
new_review1 = new_review1.lower()
new_review1 = new_review1.split()
ps = PorterStemmer()
all_stopwords = stopwords.words('english')
all_stopwords.remove('not')
new_review1 = [ps.stem(word) for word in new_review1 if not word in set(all_stopwords)]
new_review1 = ' '.join(new_review1)
new_corpus1 = [new_review1]
new_X_test1 = cv.transform(new_corpus1).toarray()
new_y_pred1 = random_forest.predict(new_X_test1)
print(new_y_pred1)
```

```
[1]
```

Inference: We repeat the same process for another new review, 'The company increased the production,' and the model predicts a sentiment as '1,' indicating a positive sentiment correctly as increased refers to positive sentiment.

Summary:

- Imported essential libraries for data analysis and machine learning.
- Loaded and explored the dataset.
- Encoded sentiment column values as 1 for positive and 0 for negative sentiments.
- Preprocessed the data by removing special characters and HTML tags, converting reviews to lowercase, eliminating stopwords, and applying stemming.
- Created feature vectors through both Bag of Words (BoW) and TF-IDF approaches.
- Trained and evaluated machine learning models, including Random Forest, for sentiment classification.
- Analyzed comprehensive classification metrics to assess model performance.
- Predicted sentiment for new reviews, showcasing the practical application of the trained models.

Conclusion:

This project demonstrated the power of text analysis and machine learning in classifying sentiment from textual data. The insights gained from this project can be leveraged to improve customer satisfaction, product development, and other business-critical decisions.