import pandas as pd

Pandas is a powerful library for data manipulation and analysis. If you have any specific tasks or questions related to pandas or need assistance with using it, please feel free to ask, and I'll be happy to help.

file\_path = "/content/Restaurant\_Reviews.tsv"

df = pd.read\_table(file\_path)

Reading the tsv file which contains Reviews

The code snippet **df['Liked'].value\_counts()** is used to calculate and display the counts of unique values in a specific column of a Pandas DataFrame.

**.value\_counts()**: This is a Pandas DataFrame method that is applied to the selected column. It counts the number of occurrences of each unique value in that column and returns the result as a new Pandas Series.

The unique values in this column are typically categories or labels that represent whether customers liked or disliked something (e.g., "Liked" or "Disliked" in your case).

import seaborn as sns

sns.countplot(df['Liked'])

This line of code uses the **countplot** function from Seaborn to create a bar plot that displays the counts of unique values in the "Liked" column of your Pandas DataFrame (**df**).

sns.distplot(df[('Liked')])

This line of code uses the **distplot** function from Seaborn to create a distribution plot for the values in the "Liked" column of your Pandas DataFrame (**df**).

x = df['Review'].values

y = df['Liked'].values

In the code snippet you provided, you are creating two NumPy arrays, **x** and **y**, from two columns in your Pandas DataFrame, **df**. Let's break down each part of the code:

1. **x = df['Review'].values**
   * This line of code extracts the values from the "Review" column of your Pandas DataFrame **df** and assigns them to the NumPy array **x**.
   * Assuming the "Review" column contains text or string data, **x** will be an array of text values.
2. **y = df['Liked'].values**
   * This line of code extracts the values from the "Liked" column of your Pandas DataFrame **df** and assigns them to the NumPy array **y**.
   * Assuming the "Liked" column contains numeric values (e.g., 0 or 1) or categorical values representing whether something is liked or not, **y** will be an array of these values.

x is input and y is output for Machine Learning model.

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,random\_state = 0)

This line imports the **train\_test\_split** function from the **sklearn.model\_selection** module. This function is used to split your data into training and testing sets.

* **x** and **y** are the NumPy arrays you defined earlier, where **x** contains your input features (e.g., text data from the "Review" column) and **y** contains your target variable (e.g., "Liked" column values).
* **train\_test\_split** takes four arguments: **x**, **y**, **random\_state**, and others (not shown here).
* **x\_train** and **x\_test** will contain the input features for the training and testing sets, respectively.
* **y\_train** and **y\_test** will contain the corresponding target values for the training and testing sets, respectively.
* **random\_state** is set to 0, which is used to control the randomization of the data splitting process. Setting it to a specific value ensures that the split remains the same when you run the code multiple times, which can be helpful for reproducibility.

from sklearn.feature\_extraction.text import CountVectorizer

vect = CountVectorizer(stop\_words='english')

x\_train\_vect = vect.fit\_transform(x\_train)

x\_test\_vect = vect.transform(x\_test)

This line imports the **CountVectorizer** class from scikit-learn's **feature\_extraction.text** module. The **CountVectorizer** is a tool for converting a collection of text documents to a matrix of token counts.

* Here, you create a **CountVectorizer** object called **vect**.
* The **stop\_words='english'** argument tells the vectorizer to remove common English stop words (e.g., "the," "and," "in") from the text data during the tokenization process. Removing stop words can help reduce noise in text data.
* **x\_train** contains the training data, which is a NumPy array of text data (likely from the "Review" column) in your dataset.
* **x\_test** contains the testing data, which is another NumPy array of text data.
* **fit\_transform** is used on the training data (**x\_train**). It first "fits" the vectorizer to the training data to learn the vocabulary of words in the training set and then transforms the training data into a document-term matrix (DTM). Each row in the DTM represents a document (a review in this case), and each column represents a unique word from the training data, with cell values indicating the count of each word in each document.
* **transform** is used on the testing data (**x\_test**). It transforms the testing data into a DTM using the same vocabulary learned from the training data. This ensures that the training and testing data are represented consistently.

from sklearn.svm import SVC

model = SVC()

model.fit(x\_train\_vect,y\_train)

y\_pred = model.predict(x\_test\_vect)

y\_pred

This line imports the **SVC** (Support Vector Classification) class from scikit-learn's **svm** module. **SVC** is a type of SVM classifier that can be used for classification tasks.

Here, you create an instance of the **SVC** classifier and assign it to the variable **model**.

* + This line trains the SVM classifier (**model**) on the training data represented by **x\_train\_vect** (the document-term matrix created from the training text data) and **y\_train** (the corresponding target labels).

After training the model, you use it to make predictions on the test data (**x\_test\_vect**). The **predict** method takes the DTM of the test data as input and returns the predicted labels.

1. **y\_pred**:
   * The variable **y\_pred** contains the predicted labels for the test data based on the trained SVM model.

You can use the **y\_pred** values to evaluate the performance of your SVM classifier, for example, by comparing them to the true labels in **y\_test** to calculate metrics like accuracy, precision, recall, or F1-score.

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_pred,y\_test)

This line imports the **accuracy\_score** function from scikit-learn's **metrics** module. The **accuracy\_score** function is used to calculate the accuracy of classification predictions.

* + his line calculates the accuracy of your model's predictions.
  + **y\_pred** contains the predicted labels generated by your SVM classifier for the test data.
  + **y\_test** contains the true or actual labels for the test data, which you're comparing to the predicted labels.

The **accuracy\_score** function compares the predicted labels (**y\_pred**) to the true labels (**y\_test**) and returns the accuracy as a floating-point number. The accuracy represents the proportion of correctly classified samples in your test dataset.

from sklearn.naive\_bayes import MultinomialNB

model1 = MultinomialNB()

model1.fit(x\_train\_vect,y\_train)

y\_pred1 = model1.predict(x\_test\_vect)

y\_pred1

This line imports the **MultinomialNB** class from scikit-learn's **naive\_bayes** module. **MultinomialNB** is a type of Naive Bayes classifier suitable for text classification tasks.

* + Here, you create an instance of the **MultinomialNB** classifier and assign it to the variable **model1**.

This line trains the Multinomial Naive Bayes classifier (**model1**) on the training data represented by **x\_train\_vect** (the document-term matrix created from the training text data) and **y\_train** (the corresponding target labels).

After training the model, you use it to make predictions on the test data (**x\_test\_vect**). The **predict** method takes the DTM of the test data as input and returns the predicted labels.

**y\_pred1**:

* The variable **y\_pred1** contains the predicted labels for the test data based on the trained Multinomial Naive Bayes model.

from sklearn.pipeline import make\_pipeline

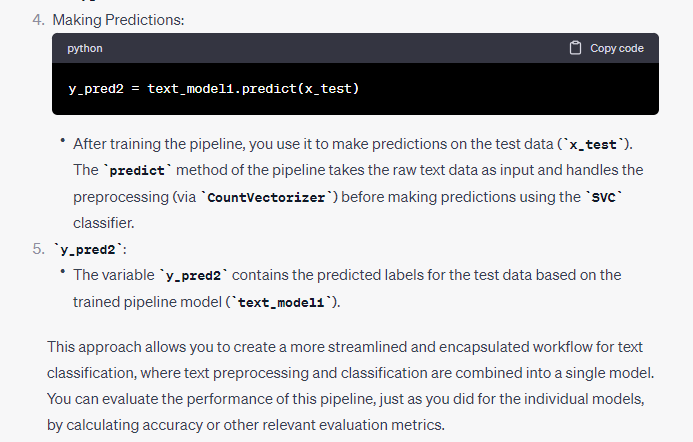
text\_model1 = make\_pipeline(CountVectorizer(),SVC())

text\_model1.fit(x\_train,y\_train)

y\_pred2 = text\_model1.predict(x\_test)

y\_pred2



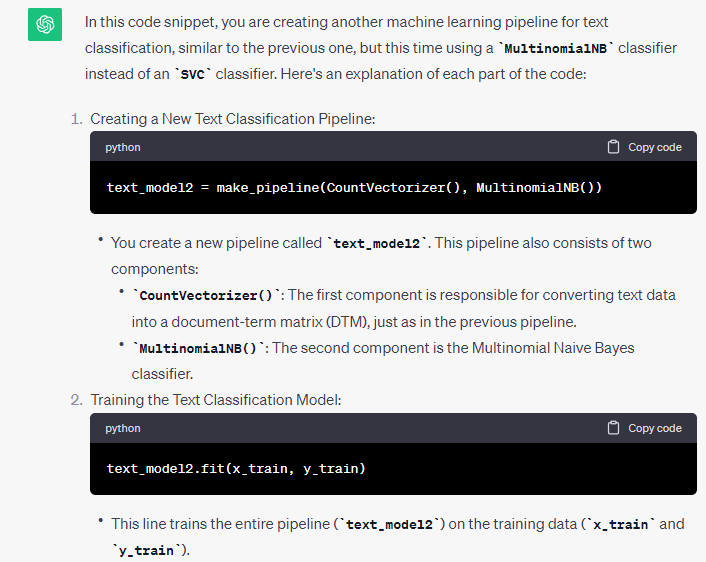


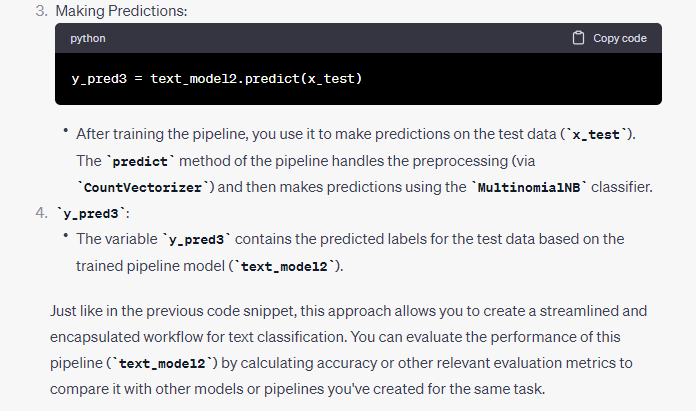
text\_model2 = make\_pipeline(CountVectorizer(),MultinomialNB())

text\_model2.fit(x\_train,y\_train)

y\_pred3 = text\_model2.predict(x\_test)

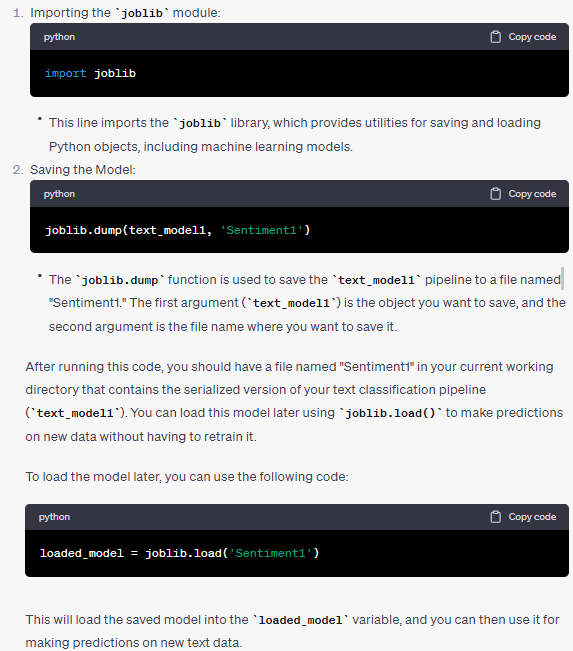
y\_pred3



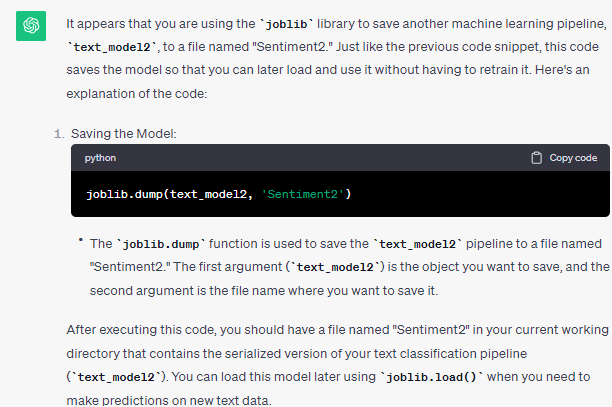


import joblib

joblib.dump(text\_model1,'Sentiment1')



joblib.dump(text\_model2,'Sentiment2')



import joblib

# Load the sentiment model

text\_model3 = joblib.load('Sentiment1')

# Define a mapping dictionary for prediction results

label\_mapping = {1: 'positive', 0: 'negative'}

# Perform prediction

prediction = text\_model3.predict(["The real disappointment was our waiter."])

# Convert prediction result to label

sentiment\_label = label\_mapping[prediction[0]]

print(sentiment\_label)

