Name: Maroti Suryakant Patre

PRN No.: RBT21CB041

Import Required Libraries:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.svm import SVC
```

Import the necessary Python libraries for data manipulation, machine learning, and evaluation.

Load Your Dataset:

```
df = pd.read_csv('Employee.csv')
```

Load your employee dataset from a CSV file into a pandas DataFrame named df.

Feature Selection:

```
X = df[['Education', 'JoiningYear', 'City', 'PaymentTier', 'Age', 'Gender', 'EverBenched', 'ExperienceInCurrentDomain']]
y = df['LeaveOrNot']
```

Define the features (X) and the target variable (y) for your classification task. In this case, 'X' includes various attributes like education, joining year, city, etc., while 'y' represents the 'LeaveOrNot' variable, which you want to predict.

Data Preprocessing - Text Conversion:

```
X = X.astype(str)
```

Convert the features to strings to ensure they can be processed by the CountVectorizer.

Combining Text Features:

```
X['combined_features'] = X.apply(lambda row: ' '.join(row), axis=1)
```

Combine all the textual feature columns into a single column named 'combined_features' for each row. This is necessary to use CountVectorizer, which expects a single text column as input.

Split the Data:

```
X_train, X_test, y_train, y_test = train_test_split(X['combined_features'], y, test_size=0.2, random_state=42)
```

Split your data into a training set (X_train, y_train) and a testing set (X_test, y_test). The test_size parameter specifies that 20% of the data will be used for testing, and random_state is set to ensure reproducibility.

Text Vectorization - Count Vectorization:

```
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)
```

Create a CountVectorizer object to convert the text data (combined_features) into numerical format. fit_transform is used on the training data to fit the vectorizer and transform the training data, while transform is used on the test data to transform it based on the fitted vectorizer.

Multinomial Naive Bayes Classifier:

```
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_counts, y_train)
```

```
MultinomialNB
MultinomialNB()
```

Create a Multinomial Naive Bayes classifier object and train it on the training data with their corresponding labels.

Make Predictions:

```
y_pred = nb_classifier.predict(X_test_counts)
```

Use the trained classifier to make predictions on the test data.

Evaluation:

```
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```

Calculate the accuracy and generate a classification report to assess the model's performance.

Print Results:

```
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_rep)
```

```
Accuracy: 0.807733619763695
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.94      0.87       610
           1       0.83      0.55      0.66       321

    accuracy                           0.81       931
   macro avg       0.82      0.75      0.76       931
weighted avg       0.81      0.81      0.80       931
```

Print the accuracy and classification report, which includes precision, recall, F1-score, and support for each class in the classification problem.

Support Vector Machine Classifier:

```
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train_counts, y_train)
```

```
SVC(kernel='linear')
```

Create an SVM classifier using SVC (Support Vector Classification) with a linear kernel. You can experiment with different kernel options, such as 'rbf' or 'poly', based on your dataset.

Make Predictions:

```
y_pred = svm_classifier.predict(X_test_counts)
```

Use the trained SVM classifier to make predictions on the test data.

Evaluation:

```
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```

Calculate accuracy and generate a classification report to assess the model's performance.

Print Results:

```
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_rep)
```

```
Accuracy: 0.8109559613319012
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.93      0.87       610
           1       0.82      0.58      0.68       321

    accuracy                           0.81       931
   macro avg       0.81      0.76      0.77       931
weighted avg       0.81      0.81      0.80       931
```

Print the accuracy and classification report, including precision, recall, F1-score, and support for each class in the classification problem.