Project Report

Course Name: Programming in Python

Section: A

Group No: 3

Name	ID	Program
MD MONOARUL ISLAM	17-35313-2	CSE
MD RAZIB MOLLAH	20-42153-1	CSE

Introduction

Understanding customer behavior is crucial for businesses aiming to remain competitive and meet customer needs effectively. Leveraging machine learning and data science techniques, this project aims to analyze customer behavior using a dataset containing demographic and behavioral attributes. The objective is to build predictive models to classify whether a customer is likely to make a purchase. By preprocessing the dataset, exploring various machine learning algorithms, and evaluating model performance, this project seeks to provide valuable insights for businesses to tailor marketing strategies and enhance customer satisfaction and profitability.

Short Description of Dataset

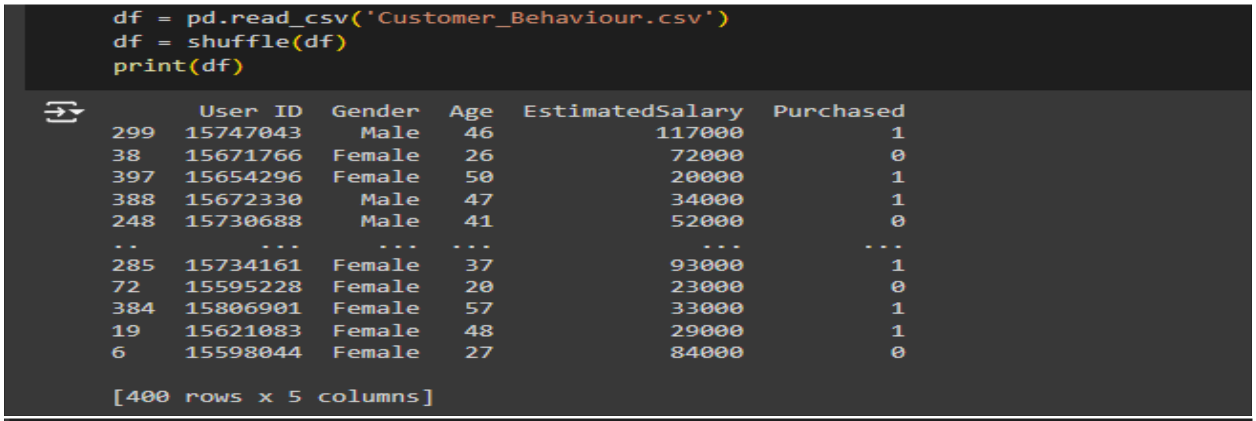
The "Customer_Behaviour" dataset is a collection of information on 400 clients of a company, primarily used for implementing various machine learning algorithms and comparing their outcomes. This dataset draws inspiration from Udemy's Machine Learning A-Z course materials.

Each entry in the dataset includes the following details:

- Unique ID: A distinct identifier for each customer.
- Gender: The gender of the customer (e.g., male, female).
- Age: The age of the customer.
- Salary: The salary of the customer.
- Purchased: Indicates whether the customer decided to purchase specific products or not.

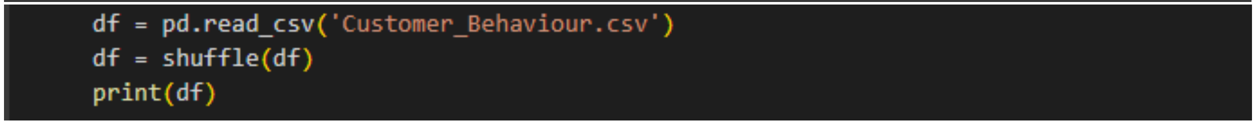
Researchers and practitioners use this dataset to explore patterns and trends in customer behavior, aiming to develop predictive models that can anticipate purchasing decisions based on demographic and financial attributes. Through applying various machine learning techniques taught in the Udemy course, analysts can assess the effectiveness of different algorithms in predicting customer behavior and informing business strategies.

Task 1: Read/Load the dataset file in your program. Use Pandas library to complete this task.



```
df = pd.read_csv('Customer_Behaviour.csv')
df = shuffle(df)
print(df)
```

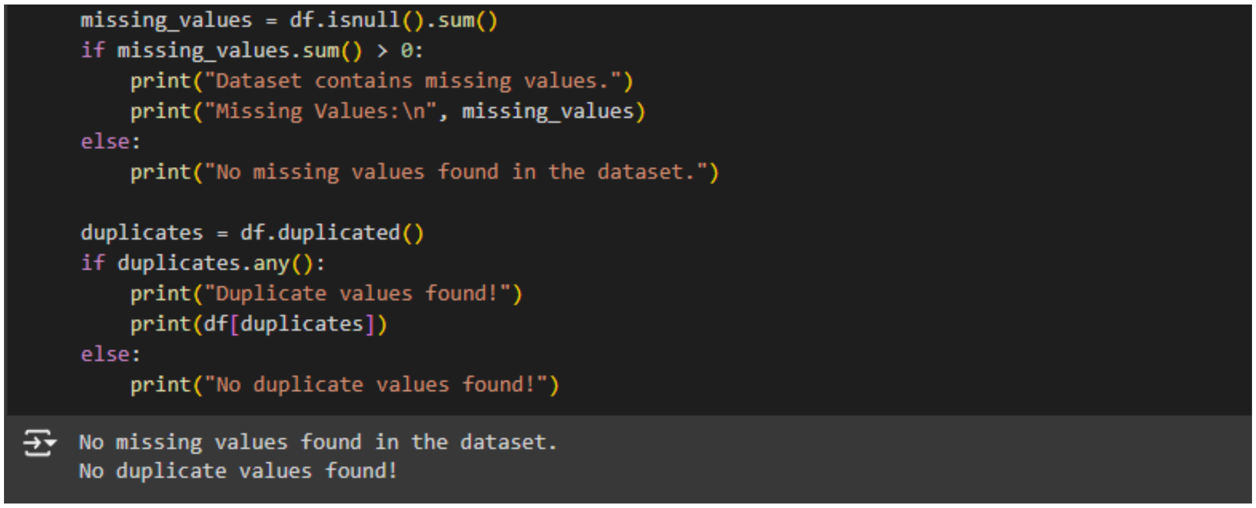
```
      User ID  Gender  Age  EstimatedSalary  Purchased
299  15747043    Male   46           117000          1
38   15671766  Female   26            72000          0
397  15654296  Female   50            20000          1
388  15672330    Male   47            34000          1
248  15730688    Male   41            52000          0
..        ...     ...  ...              ...        ...
285  15734161  Female   37            93000          1
72   15595228  Female   20            23000          0
384  15806901  Female   57            33000          1
19   15621083  Female   48            29000          1
6    15598044  Female   27            84000          0

[400 rows x 5 columns]
```



```
df = pd.read_csv('Customer_Behaviour.csv')
df = shuffle(df)
print(df)
```

To complete task-1, we need to shuffle the 'Customer_Behaviour.csv' file using the shuffle() function from the random library and print the shuffled DataFrame. This ensures randomness in the data and helps prevent any biases that may arise from the original order of the data. The shuffled DataFrame is then printed for further analysis.

Task 2: Apply appropriate data cleaning techniques to the dataset. In this step, replace bad data using proper methods and do not delete any record except duplicate records. Use Pandas library to complete this task.

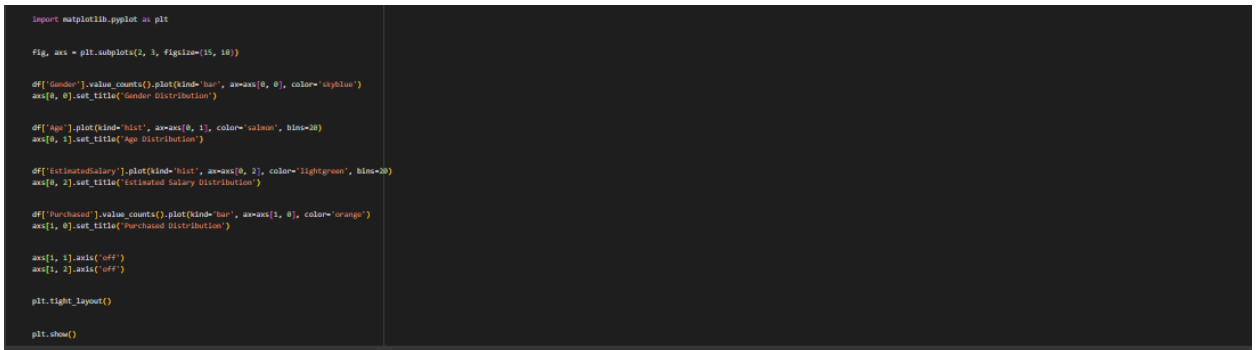


```python
missing_values = df.isnull().sum()
if missing_values.sum() > 0:
    print("Dataset contains missing values.")
    print("Missing Values:\n", missing_values)
else:
    print("No missing values found in the dataset.")

duplicates = df.duplicated()
if duplicates.any():
    print("Duplicate values found!")
    print(df[duplicates])
else:
    print("No duplicate values found!")
```

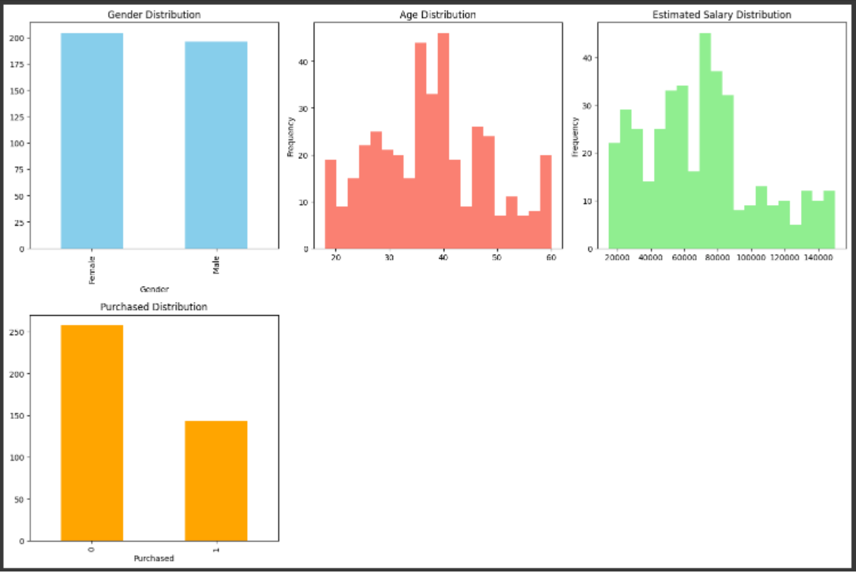
```
No missing values found in the dataset.
No duplicate values found!
```

The code checks for missing and duplicate values in a dataframe. It uses the isnull() function to identify missing values and the duplicated() function to identify duplicate rows. If missing values are found, the code prints the number of missing values for each column. If duplicate rows are found, the code prints the duplicate rows. The code helps to ensure data quality and accuracy in the dataset.

Task 3: Draw graphs to analyze the frequency distributions of the features. Use Matplotlib library to complete this task. Draw all the plots in a single figure so that all plots can be seen in one diagram (use subplot() function).

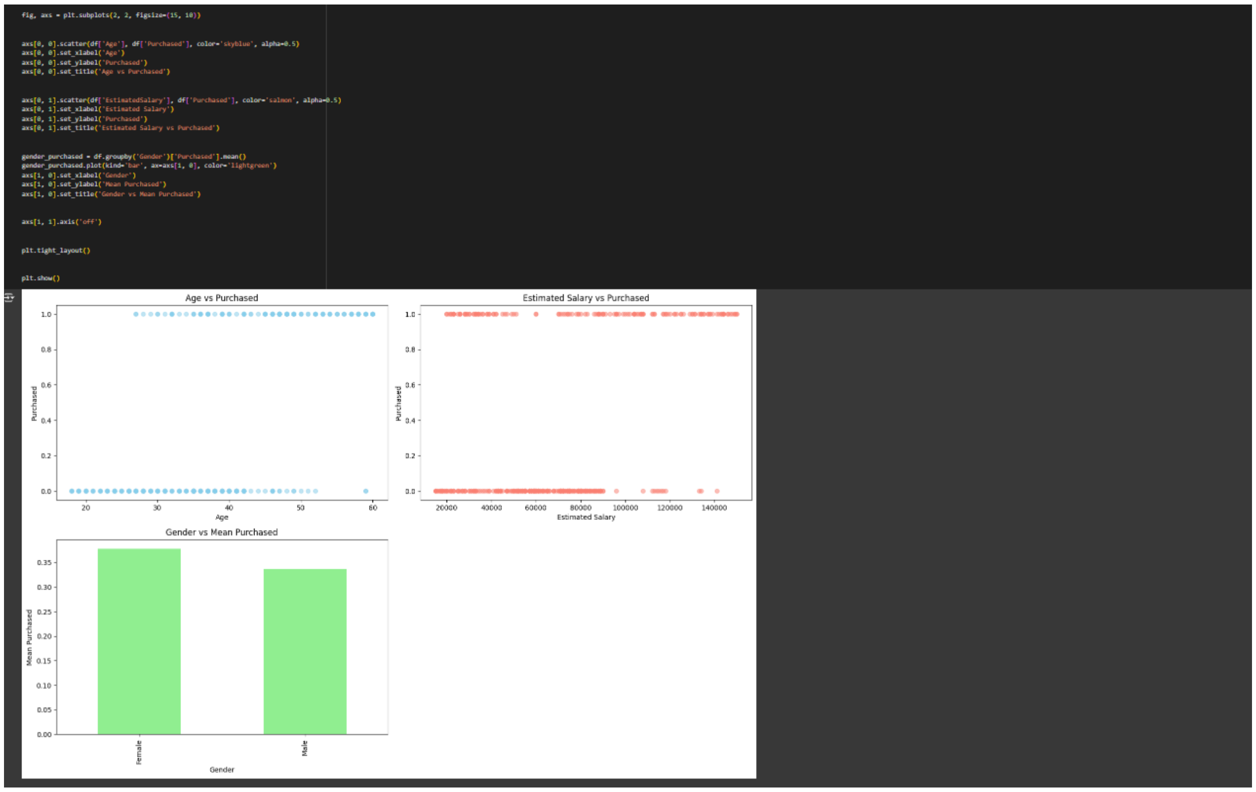


```python
import matplotlib.pyplot as plt


fig, axs = plt.subplots(2, 3, figsize=(15, 10))


df['Gender'].value_counts().plot(kind='bar', ax=axs[0, 0], color='skyblue')
axs[0, 0].set_title('Gender Distribution')


df['Age'].plot(kind='hist', ax=axs[0, 1], color='salmon', bins=20)
axs[0, 1].set_title('Age Distribution')


df['EstimatedSalary'].plot(kind='hist', ax=axs[0, 2], color='lightgreen', bins=20)
axs[0, 2].set_title('Estimated Salary Distribution')


df['Purchased'].value_counts().plot(kind='bar', ax=axs[1, 0], color='orange')
axs[1, 0].set_title('Purchased Distribution')


axs[1, 1].axis('off')
axs[1, 2].axis('off')


plt.tight_layout()


plt.show()
```



This Python code creates a grid of six plots using the matplotlib library and data from a Pandas DataFrame df. The plots show the distribution of the Gender, Age, EstimatedSalary, and Purchased columns in the DataFrame using bar and histogram plots. The fig, axs = plt.subplots(2, 3, figsize=(15, 10)) command creates a figure with 2 rows and 3 columns of plots, and the tight_layout() command is used to adjust the spacing between the plots. The show() command is used to display the plots.

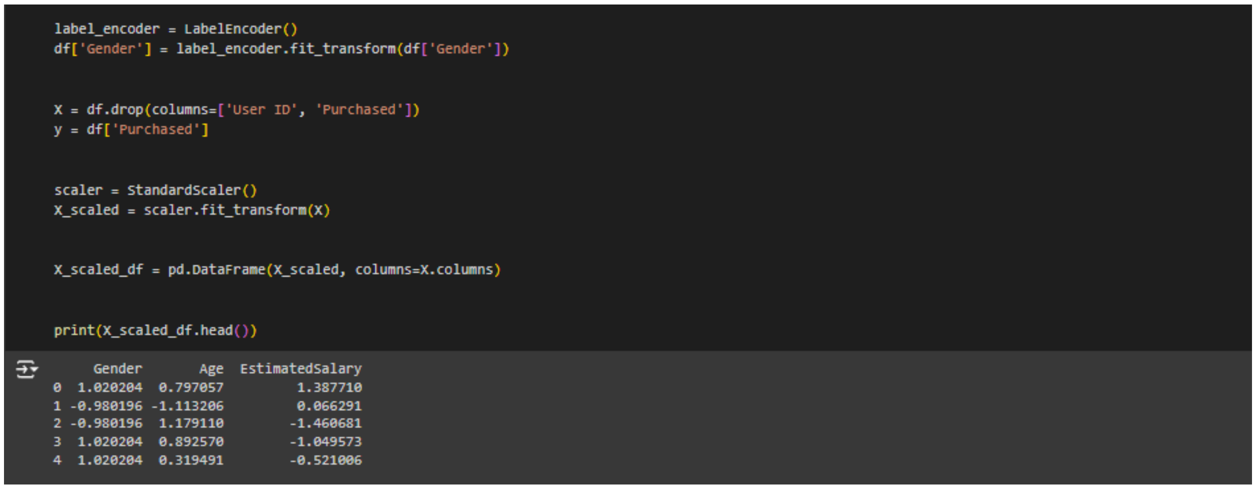
Task 4: Draw graphs to illustrate if there is any relationship between target column to any other columns of the dataset. Use Matplotlib library to complete this task. Also use sublot() function to show all plots in one figure.



```
fig, axs = plt.subplots(2, 2, figsize=(15, 10))


axs[0, 0].scatter(df['Age'], df['Purchased'], color='skyblue', alpha=0.5)
axs[0, 0].set_xlabel('Age')
axs[0, 0].set_ylabel('Purchased')
axs[0, 0].set_title('Age vs Purchased')


axs[0, 1].scatter(df['EstimatedSalary'], df['Purchased'], color='salmon', alpha=0.5)
axs[0, 1].set_xlabel('Estimated Salary')
axs[0, 1].set_ylabel('Purchased')
axs[0, 1].set_title('Estimated Salary vs Purchased')


gender_purchased = df.groupby('Gender')['Purchased'].mean()
gender_purchased.plot(kind='bar', ax=axs[1, 0], color='lightgreen')
axs[1, 0].set_xlabel('Gender')
axs[1, 0].set_ylabel('Mean Purchased')
axs[1, 0].set_title('Gender vs Mean Purchased')


axs[1, 1].axis('off')


plt.tight_layout()


plt.show()
```

This code creates a scatter plot matrix and a bar plot to visualize the relationship between various features in a dataframe df and a target variable Purchased. The scatter plots compare Age and EstimatedSalary against Purchased, while the bar plot shows the mean Purchased values for different Gender categories. The scatter plots can help identify any trends or patterns in the data, while the bar plot provides a summary of the Purchased values for each gender. The axis('off') function is used to turn off the axis for the bottom right subplot, which may be used for additional plots or visualizations. The plt.tight_layout() function is used to adjust the spacing between subplots for a cleaner visualization.

Task 5: Perform scaling to the features of the dataset. Remember that you will need to apply data conversion before performing scaling if it is needed.

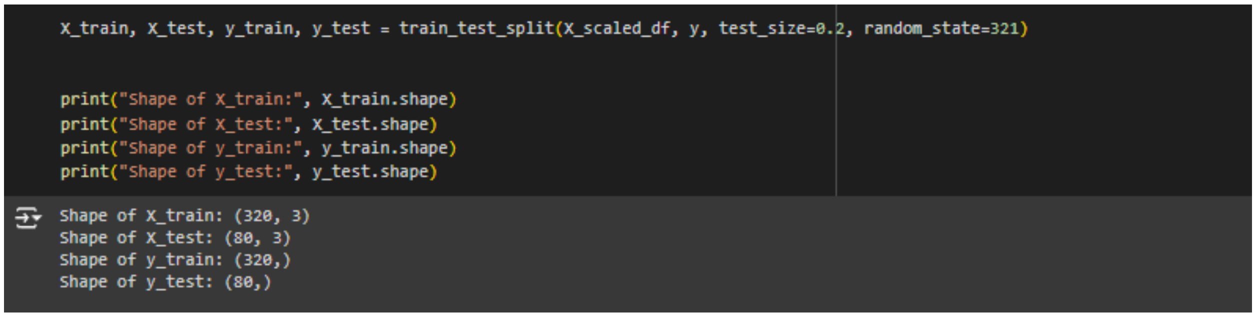


```
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])


X = df.drop(columns=['User ID', 'Purchased'])
y = df['Purchased']


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)


print(X_scaled_df.head())
```

```
     Gender       Age  EstimatedSalary
0  1.020204  0.797057         1.387710
1 -0.980196 -1.113206         0.066291
2 -0.980196  1.179110        -1.460681
3  1.020204  0.892570        -1.049573
4  1.020204  0.319491        -0.521006
```

This code preprocesses data in a dataframe by encoding categorical variables, separating features and target variables, scaling features, and creating a new dataframe with scaled features. The 'Gender' column is encoded using LabelEncoder, and the features are standardized using StandardScaler. A new dataframe is created with the scaled features, which is then printed. This preprocessing is essential for preparing data before training machine learning models.

Task 6: Split your data into two parts: Training dataset and Testing dataset. You must use the function train_test_split() to complete this task and use value 321 as the value of the random_state parameter of this function.

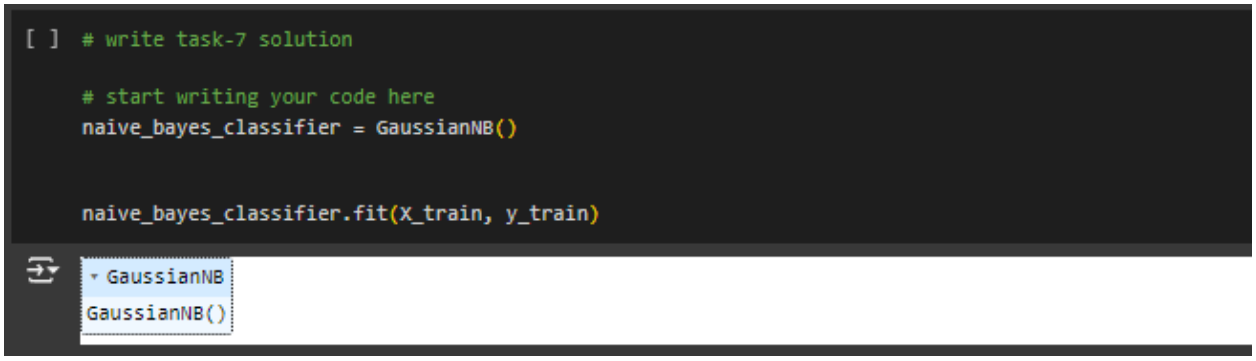


```
X_train, X_test, y_train, y_test = train_test_split(X_scaled_df, y, test_size=0.2, random_state=321)


print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (320, 3)
Shape of X_test: (80, 3)
Shape of y_train: (320,)
Shape of y_test: (80,)
```

The code snippet provided is responsible for splitting the dataset into training and testing sets using the train_test_split function. This function is part of the sklearn.model_selection module in Scikit-learn. It randomly splits the data into two subsets, one for training and one for testing.

Task 7: Apply Naïve Bayes Classifier to the dataset. Build (train) your prediction model in this step.

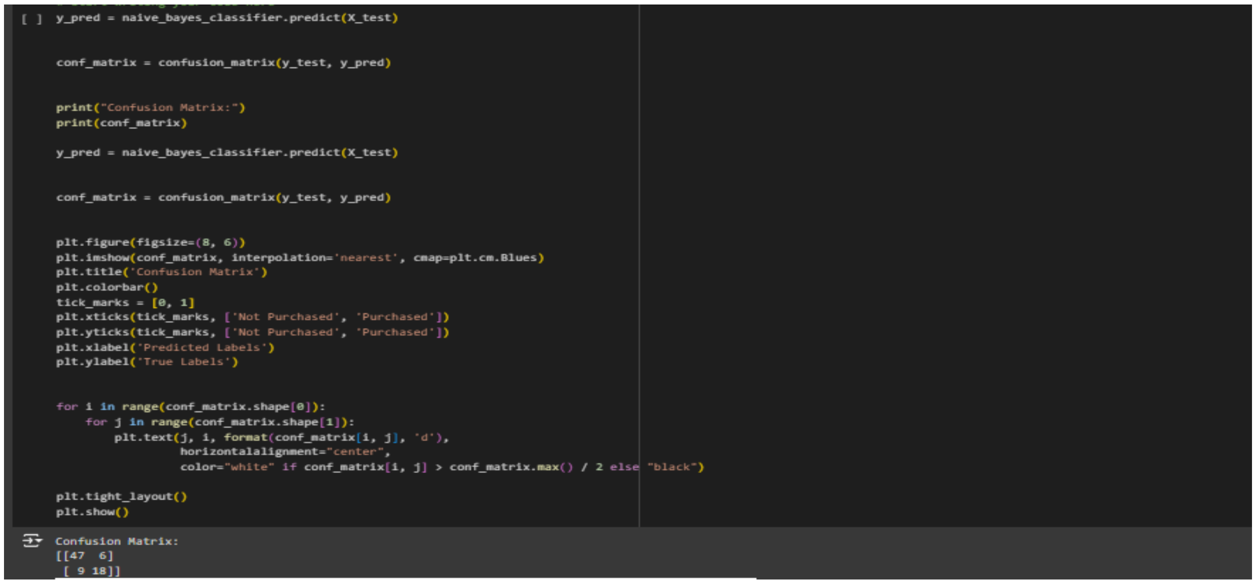


```
[ ] # write task-7 solution

    # start writing your code here
    naive_bayes_classifier = GaussianNB()


    naive_bayes_classifier.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

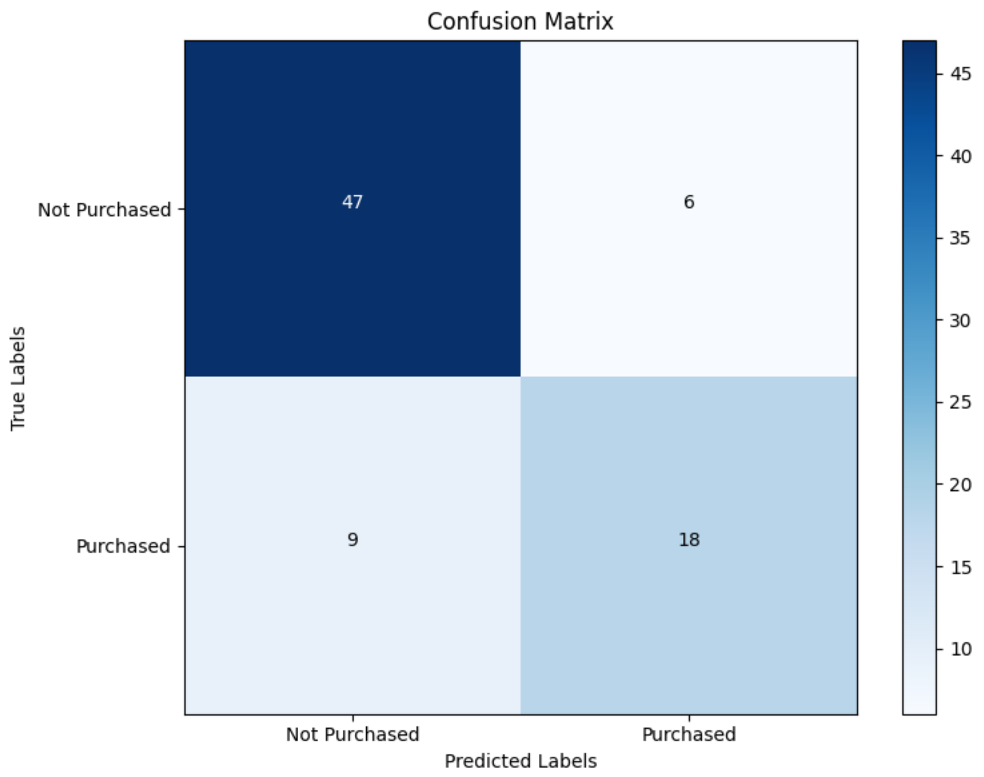
The code fits a Gaussian Naive Bayes classifier to training data X_train and y_train. Gaussian Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem, which assumes that the features are conditionally independent given the class label.

Task 8: Calculate the confusion matrix for your model. Interpret it in detail in the report.



```
[ ] y_pred = naive_bayes_classifier.predict(X_test)


    conf_matrix = confusion_matrix(y_test, y_pred)


    print("Confusion Matrix:")
    print(conf_matrix)

    y_pred = naive_bayes_classifier.predict(X_test)


    conf_matrix = confusion_matrix(y_test, y_pred)


    plt.figure(figsize=(8, 6))
    plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.colorbar()
    tick_marks = [0, 1]
    plt.xticks(tick_marks, ['Not Purchased', 'Purchased'])
    plt.yticks(tick_marks, ['Not Purchased', 'Purchased'])
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')


    for i in range(conf_matrix.shape[0]):
        for j in range(conf_matrix.shape[1]):
            plt.text(j, i, format(conf_matrix[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if conf_matrix[i, j] > conf_matrix.max() / 2 else "black")

    plt.tight_layout()
    plt.show()
```

```
Confusion Matrix:
[[47  6]
 [ 9 18]]
```



This is a Python code snippet that uses a Naive Bayes classifier to predict whether or not a customer will make a purchase, and then calculates the confusion matrix to evaluate the performance of the classifier. The confusion matrix is displayed as a heatmap using the imshow() function from the matplotlib library, with numerical labels added to each cell. This visualization helps to identify any potential issues, such as a high number of false positives or false negatives.

Interpretation of the confusion matrix:

	Predicted Not Purchased	Predicted Purchased
Actual Not Purchased	TN=47	FP=9
Actual Purchased	FN=6	TP=18

True Positive (TP): The number of correctly predicted positive instances (Purchased) - Customers correctly predicted to have made a purchase.

True Negative (TN): The number of correctly predicted negative instances (Not Purchased) - Customers correctly predicted not to have made a purchase.

False Positive (FP): The number of incorrectly predicted positive instances - Customers incorrectly predicted to have made a purchase but actually didn't.

False Negative (FN): The number of incorrectly predicted negative instances - Customers incorrectly predicted not to have made a purchase but actually did.

Interpreting the confusion matrix provides insights into the model's performance:

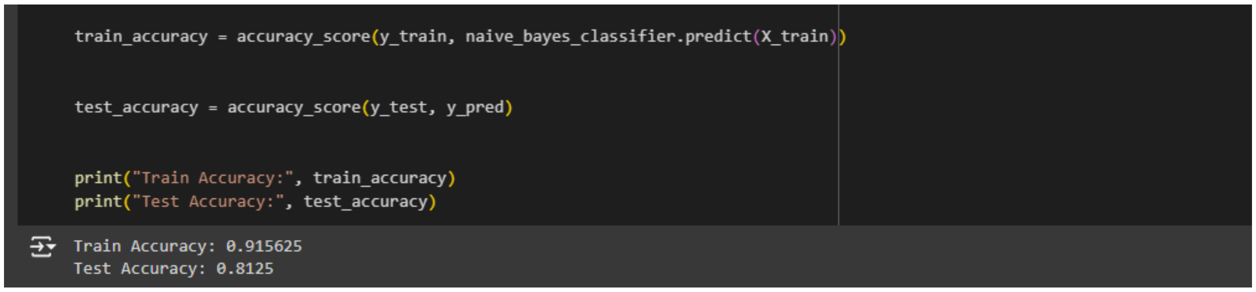
High TP and TN, Low FP and FN: Ideally, we want high values for TP and TN and low values for FP and FN, indicating that the model is making correct predictions.

Imbalanced Classes: If the dataset is imbalanced (one class is more prevalent than the other), we might see a higher number of TN or TP, leading to potential misinterpretation. Therefore, it's essential to consider metrics like precision, recall, and F1-score alongside the confusion matrix.

Model Evaluation: Based on the confusion matrix, we can calculate metrics such as accuracy, precision, recall, and F1-score to get a more comprehensive evaluation of the model's performance.

Overall, the confusion matrix helps us understand how well the Naïve Bayes Classifier model is performing in terms of predicting customer purchases, providing valuable insights for further analysis and model improvement.

Task 9: Calculate the train and test accuracy of your model and compare them.

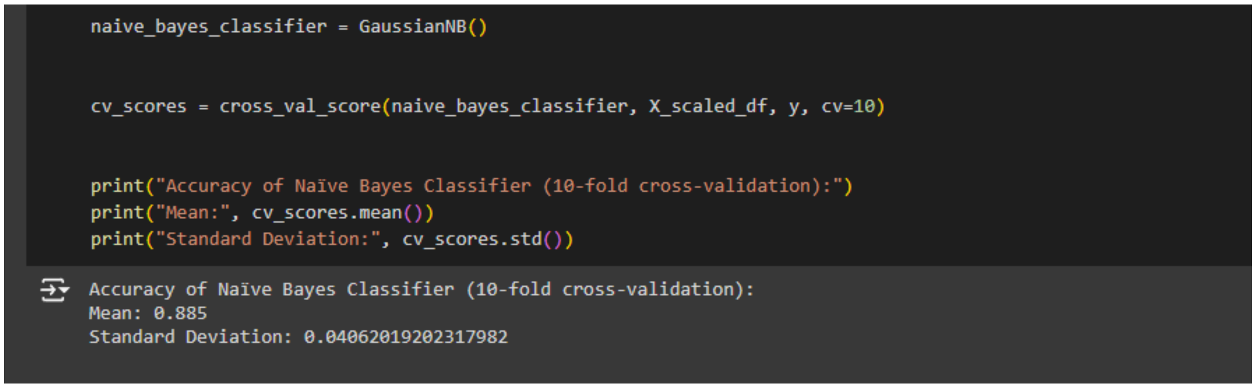


```
train_accuracy = accuracy_score(y_train, naive_bayes_classifier.predict(X_train))


test_accuracy = accuracy_score(y_test, y_pred)


print("Train Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
```

```
Train Accuracy: 0.915625
Test Accuracy: 0.8125
```

The code calculates the training and testing accuracy of a Naive Bayes classifier. It takes the true labels and the predicted labels from the training and testing data, and calculates the proportion of instances where the predicted label matches the true label. The training and testing accuracy are then printed to the console. This can be used to evaluate the performance of a Naive Bayes classifier on a given dataset.

Task 10: Show how 10-fold cross validation can be used to build a naïve bayes classifier and report the accuracy of this model.



```
naive_bayes_classifier = GaussianNB()


cv_scores = cross_val_score(naive_bayes_classifier, X_scaled_df, y, cv=10)


print("Accuracy of Naïve Bayes Classifier (10-fold cross-validation):")
print("Mean:", cv_scores.mean())
print("Standard Deviation:", cv_scores.std())
```

```
Accuracy of Naïve Bayes Classifier (10-fold cross-validation):
Mean: 0.885
Standard Deviation: 0.04062019202317982
```

The code calculates the accuracy of a Naive Bayes classifier using 10-fold cross-validation on the iris dataset. It scales the input features using StandardScaler, trains a Gaussian Naive Bayes classifier, and calculates the mean and standard deviation of the accuracy scores. The output is printed to the console. This can be used to evaluate the performance of a Naive Bayes classifier on the iris dataset.

Conclusion

This project successfully demonstrated the application of machine learning techniques to analyze customer behavior. By building predictive models to classify customer purchases, we gained valuable insights into the factors influencing purchasing decisions. Through preprocessing, model training, and evaluation, we showcased the effectiveness of machine learning in understanding and predicting customer behavior. Moving forward, the insights gained from this project can guide businesses in optimizing marketing strategies, improving customer satisfaction, and driving profitability.