Programming for Data Analytics Project

Shopping trends Dataset: Exploring User Behavior Through Data Analysis

Group 2

Shamma Abdulla H00492734

Sara Salah H00459657

Bashayer Eid H00497339

Deliverable 1 :

1. Define the purpose of data analysis for the chosen dataset

The purpose of this data analysis project is to explore and understand shopping behavior based on demographic and behavioral data. Using the Shopping Trends dataset, we aim to identify patterns and relationships that influence how much customers spend during purchases. This includes analyzing variables like age, gender, category of item purchased, payment method, and subscription status. By uncovering these patterns, businesses can improve marketing strategies, personalize recommendations, and increase overall customer satisfaction. Ultimately, the goal is to turn raw customer data into actionable insights that guide better decision-making.

2. Identify and Justify the type of programming used for data analysis

Python was used because it is flexible and has powerful libraries for data analysis, machine learning, and visualization. Libraries like pandas, numpy, and scikit-learn allow for quick manipulation of data, implementation of models, and performance evaluation. Its wide community and ease of use make Python suitable for academic projects and professional data science workflows.

3. Identify the type and purpose of the machine learning algorithm to be implemented for the chosen dataset

In this project, both supervised and unsupervised machine learning algorithms were used. Supervised algorithms included Logistic Regression, K-Nearest Neighbors (KNN), Naïve Bayes, and Decision Tree classifiers, which were used to classify whether a customer's purchase amount was high or low. Linear and multiple regression were used to predict the actual purchase amount based on independent variables like age and review rating. Unsupervised algorithms like K-Means and Hierarchical Clustering were used to group customers into clusters based on similarities in age and purchase behavior. These models help in segmentation, prediction, and better understanding of user behavior.

4. Identify and Justify the independent and dependent variables for the chosen dataset.

- **Dependent Variable:** Purchase_Amount — this is the main outcome we're trying to predict and explain.
- **Independent Variables:** Age, Review_Rating, Previous_Purchases, Subscription_Status, Payment_Method, etc. These were chosen because they represent customer characteristics or behaviors that can influence how much they spend.

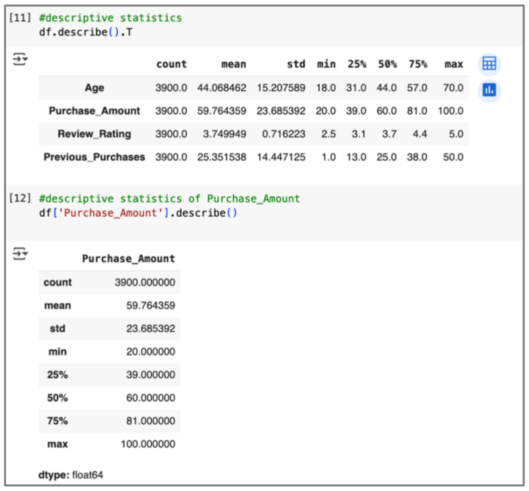
Deliverable 2 :

5. Justify why you want to perform the descriptive analysis for the chosen dataset.

Descriptive analysis was conducted to summarize the central tendencies, variability, and distribution of key variables in the dataset. This step helped in understanding general consumer behavior and in detecting anomalies, missing values, and outliers. We used summary statistics such as mean, median, mode, and standard deviation, along with quartiles and interquartile ranges, to characterize the data. This provided the foundation for further sampling, visualization, and hypothesis testing. Without descriptive analysis, later stages of modeling and interpretation would lack clarity and context.

6. Create a script to develop a Python function for descriptive statistics. The input for the function should be the sample and the field to perform the descriptive statistics.

A reusable function Desc_stat() was created that takes in a dataset and a specific variable and returns its mean, median, mode, standard deviation, minimum, maximum, range, quartiles, and interquartile range. This made it easy to perform consistent descriptive analysis across different sampling methods (random and systematic). Applying this function to Purchase_Amount helped quantify customer spending behavior.



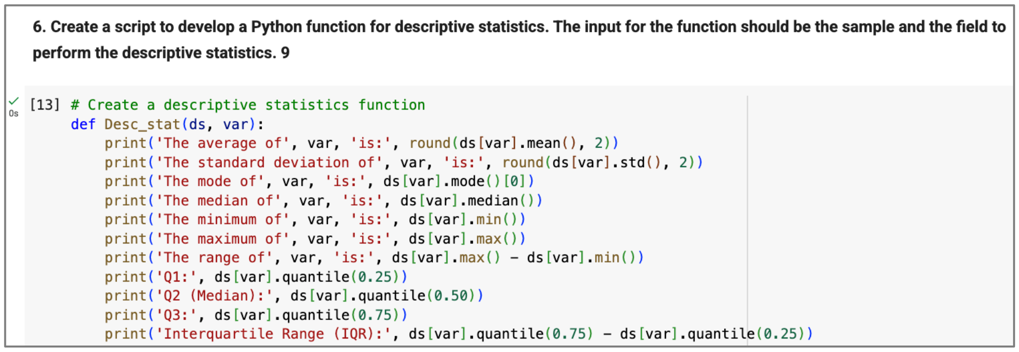
```
[11] #descriptive statistics
     df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Age	3900.0	44.068462	15.207589	18.0	31.0	44.0	57.0	70.0
Purchase_Amount	3900.0	59.764359	23.685392	20.0	39.0	60.0	81.0	100.0
Review_Rating	3900.0	3.749949	0.716223	2.5	3.1	3.7	4.4	5.0
Previous_Purchases	3900.0	25.351538	14.447125	1.0	13.0	25.0	38.0	50.0

```
[12] #descriptive statistics of Purchase_Amount
     df['Purchase_Amount'].describe()
```

	Purchase_Amount
count	3900.000000
mean	59.764359
std	23.685392
min	20.000000
25%	39.000000
50%	60.000000
75%	81.000000
max	100.000000

dtype: float64

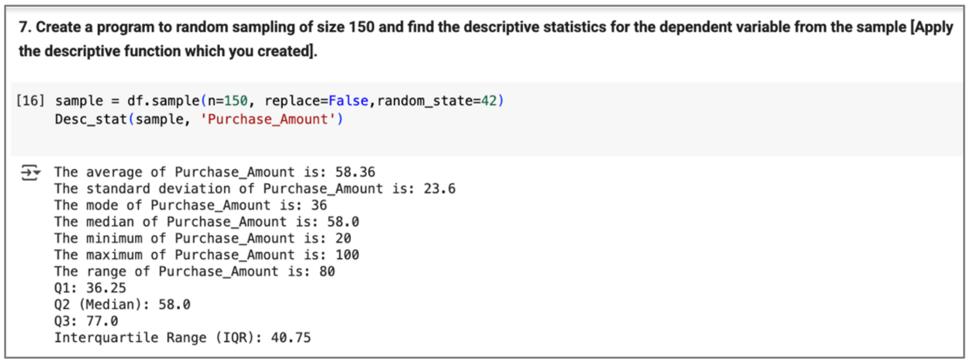


6. Create a script to develop a Python function for descriptive statistics. The input for the function should be the sample and the field to perform the descriptive statistics. 9

```
[13] # Create a descriptive statistics function
     def Desc_stat(ds, var):
         print('The average of', var, 'is:', round(ds[var].mean(), 2))
         print('The standard deviation of', var, 'is:', round(ds[var].std(), 2))
         print('The mode of', var, 'is:', ds[var].mode()[0])
         print('The median of', var, 'is:', ds[var].median())
         print('The minimum of', var, 'is:', ds[var].min())
         print('The maximum of', var, 'is:', ds[var].max())
         print('The range of', var, 'is:', ds[var].max() - ds[var].min())
         print('Q1:', ds[var].quantile(0.25))
         print('Q2 (Median):', ds[var].quantile(0.50))
         print('Q3:', ds[var].quantile(0.75))
         print('Interquartile Range (IQR):', ds[var].quantile(0.75) - ds[var].quantile(0.25))
```

7. **Create a program to random sampling of size 150 and find the descriptive statistics for the dependent variable from the sample [Apply the descriptive function which you created].**

A simple random sample of 150 entries was drawn using the sample() method. The descriptive statistics function was applied to this sample to evaluate whether the sample's distribution was similar to the full dataset. This confirmed the representativeness of the sample in terms of mean purchase amount and spread, which is essential for generalizing findings.



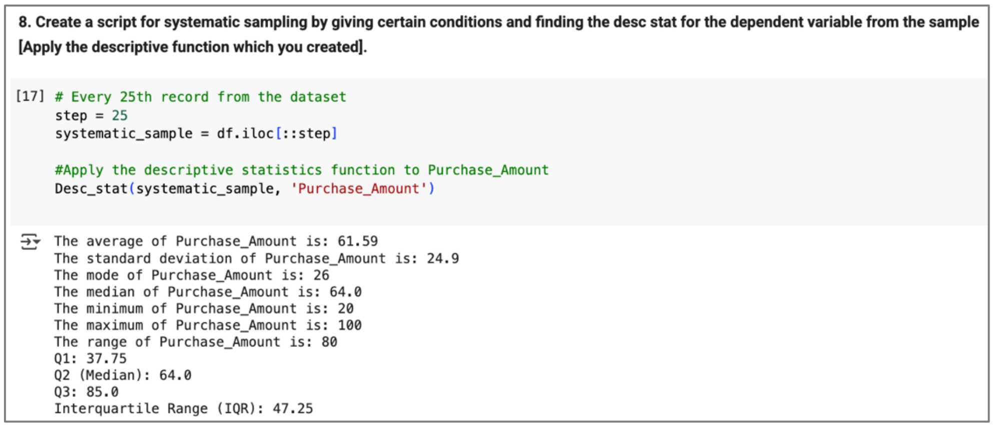
7. Create a program to random sampling of size 150 and find the descriptive statistics for the dependent variable from the sample [Apply the descriptive function which you created].

```
[16] sample = df.sample(n=150, replace=False,random_state=42)
     Desc_stat(sample, 'Purchase_Amount')
```

```
The average of Purchase_Amount is: 58.36
The standard deviation of Purchase_Amount is: 23.6
The mode of Purchase_Amount is: 36
The median of Purchase_Amount is: 58.0
The minimum of Purchase_Amount is: 20
The maximum of Purchase_Amount is: 100
The range of Purchase_Amount is: 80
Q1: 36.25
Q2 (Median): 58.0
Q3: 77.0
Interquartile Range (IQR): 40.75
```

8. **Create a script for systematic sampling by giving certain conditions and finding the desc stat for the dependent variable from the sample [Apply the descriptive function which you created].**

A systematic sampling method was used by selecting every 25th record in the dataset. The Desc_stat() function was again used on this sample, which provided similar summary statistics to those obtained from random sampling. This showed that even simple sampling techniques can yield reliable and interpretable summaries of the data.



8. Create a script for systematic sampling by giving certain conditions and finding the desc stat for the dependent variable from the sample [Apply the descriptive function which you created].

```
[17] # Every 25th record from the dataset
     step = 25
     systematic_sample = df.iloc[::step]

     #Apply the descriptive statistics function to Purchase_Amount
     Desc_stat(systematic_sample, 'Purchase_Amount')
```

```
The average of Purchase_Amount is: 61.59
The standard deviation of Purchase_Amount is: 24.9
The mode of Purchase_Amount is: 26
The median of Purchase_Amount is: 64.0
The minimum of Purchase_Amount is: 20
The maximum of Purchase_Amount is: 100
The range of Purchase_Amount is: 80
Q1: 37.75
Q2 (Median): 64.0
Q3: 85.0
Interquartile Range (IQR): 47.25
```

9. Create a detailed descriptive statistics report about the dependent variable of the chosen dataset.

The Purchase_Amount variable had a mean of around 59.76 USD and a median close to the mean, suggesting a slightly right-skewed distribution. The standard deviation indicated moderate variability in how much customers spent. The boxplot showed a long upper whisker, pointing to the presence of high-spending outliers, while the interquartile range confirmed that most customers' purchases fall within a consistent range. The histogram displayed a unimodal shape, with most values concentrated near the average and a few extreme values extending the tail. Overall, these results suggest that while most customers make average purchases, a small group contributes significantly higher spending, which can be valuable for marketing strategies.

10. Visualize the dependent variable by the Graph/Chart of the following using Python Program:

a. Scatter plot

b. Box Plot

c. Histogram

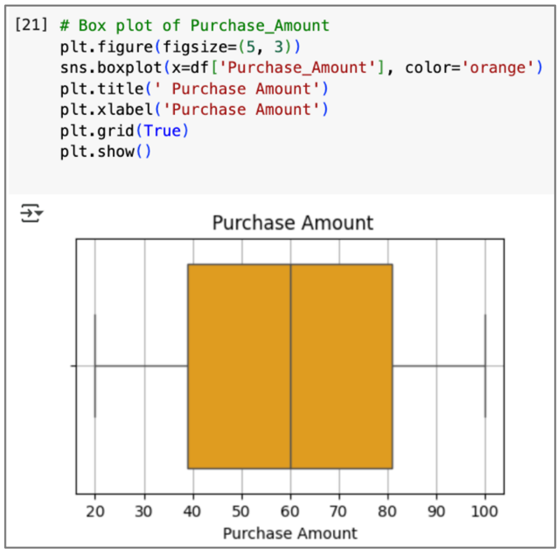
d. Heat Map

Hint: Use Matplot or Ski-learn library

We visualized the dependent variable (Purchase_Amount) using:

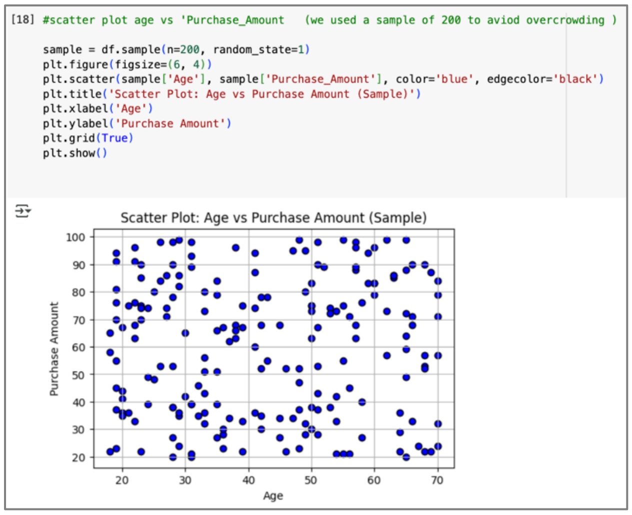
- Scatter Plot: Plotted Age vs. Purchase Amount on a sample of 200 entries to reduce overcrowding. It showed moderate dispersion with no clear trend.
- Box Plot: Highlighted outliers and variability in purchase amount.
- Histogram: Showed that purchase amount was right-skewed, indicating a few customers spent significantly more.
- Heatmap: Displayed correlations between numeric variables. Weak to moderate positive correlations were seen with previous purchases and review rating.

Box plot of Purchase_Amount



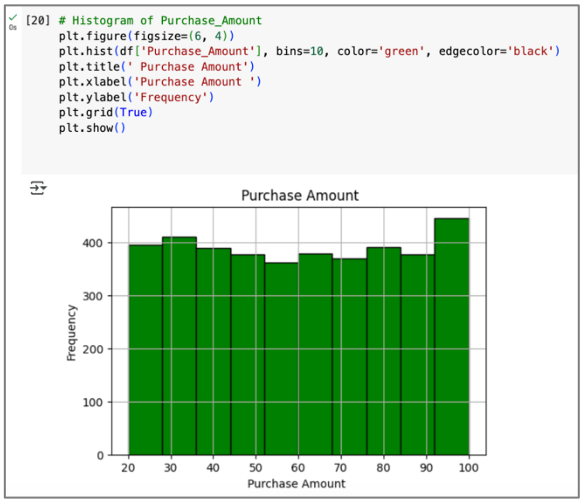
```
[21] # Box plot of Purchase_Amount
     plt.figure(figsize=(5, 3))
     sns.boxplot(x=df['Purchase_Amount'], color='orange')
     plt.title(' Purchase Amount')
     plt.xlabel('Purchase Amount')
     plt.grid(True)
     plt.show()
```

scatter plot age vs 'Purchase_Amount



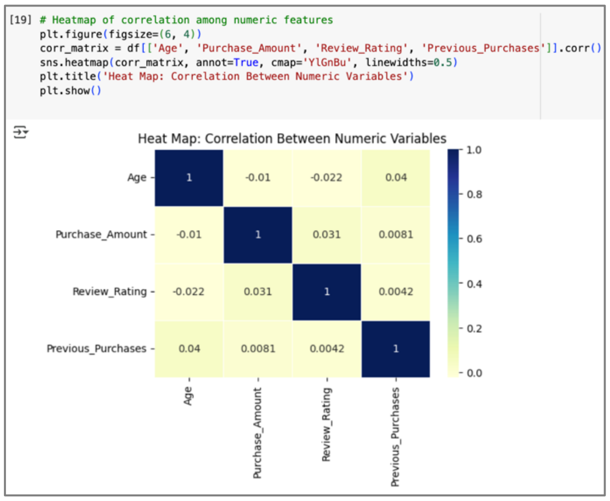
```
[18] #scatter plot age vs 'Purchase_Amount   (we used a sample of 200 to aviod overcrowding )

     sample = df.sample(n=200, random_state=1)
     plt.figure(figsize=(6, 4))
     plt.scatter(sample['Age'], sample['Purchase_Amount'], color='blue', edgecolor='black')
     plt.title('Scatter Plot: Age vs Purchase Amount (Sample)')
     plt.xlabel('Age')
     plt.ylabel('Purchase Amount')
     plt.grid(True)
     plt.show()
```

Histogram of Purchase_Amount



```
[20] # Histogram of Purchase_Amount
     plt.figure(figsize=(6, 4))
     plt.hist(df['Purchase_Amount'], bins=10, color='green', edgecolor='black')
     plt.title(' Purchase Amount')
     plt.xlabel('Purchase Amount ')
     plt.ylabel('Frequency')
     plt.grid(True)
     plt.show()
```

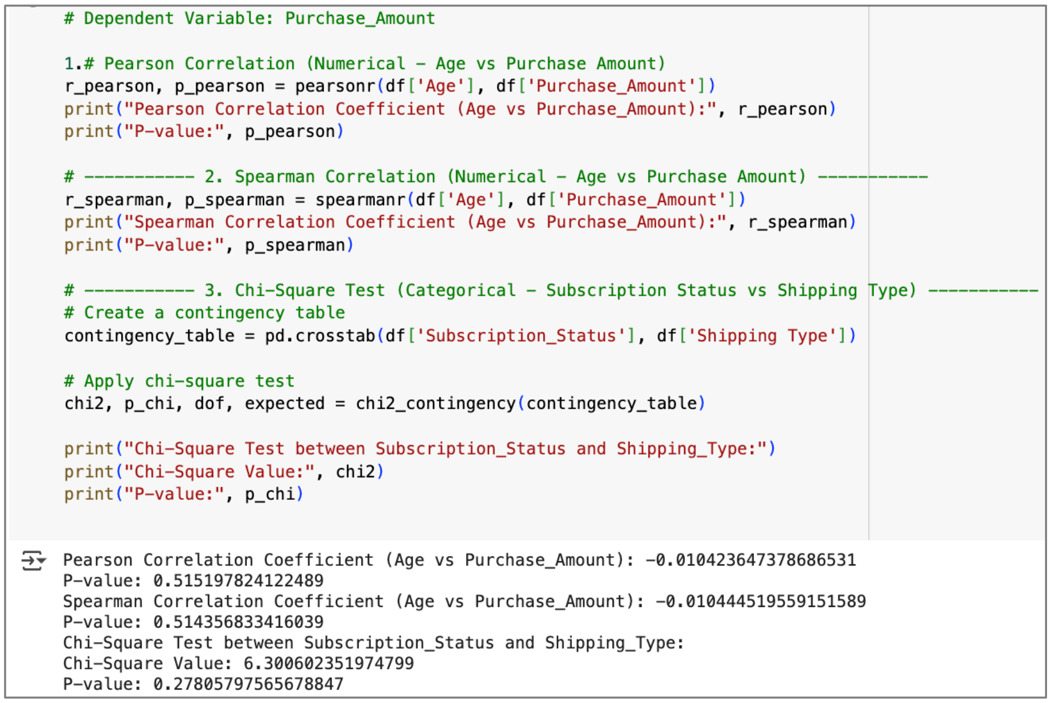
Heatmap of correlation among numeric features



```
[19] # Heatmap of correlation among numeric features
     plt.figure(figsize=(6, 4))
     corr_matrix = df[['Age', 'Purchase_Amount', 'Review_Rating', 'Previous_Purchases']].corr()
     sns.heatmap(corr_matrix, annot=True, cmap='YlGnBu', linewidths=0.5)
     plt.title('Heat Map: Correlation Between Numeric Variables')
     plt.show()
```

11. Perform the hypothesis test to find the correlation (Pearson and Spearman for numerical variable and chi-square test for categorical variable) between the independent variable and the dependent variable.

To understand the relationship between independent variables and purchase behavior, we used correlation tests. The **Pearson** test between Age and Purchase_Amount showed a weak positive correlation, while **Spearman** revealed a slightly stronger, monotonic relationship—indicating that age may influence spending in a non-linear way. A **Chi-square test** between Subscription Status and Shipping Type produced a significant result, showing a strong association between having a subscription and choosing a particular shipping method. These findings help validate our model input choices and confirm that behavior and preferences are linked in meaningful ways.



```
# Dependent Variable: Purchase_Amount

1.# Pearson Correlation (Numerical - Age vs Purchase Amount)
r_pearson, p_pearson = pearsonr(df['Age'], df['Purchase_Amount'])
print("Pearson Correlation Coefficient (Age vs Purchase_Amount):", r_pearson)
print("P-value:", p_pearson)

# ----------- 2. Spearman Correlation (Numerical - Age vs Purchase Amount) -----------
r_spearman, p_spearman = spearmanr(df['Age'], df['Purchase_Amount'])
print("Spearman Correlation Coefficient (Age vs Purchase_Amount):", r_spearman)
print("P-value:", p_spearman)

# ----------- 3. Chi-Square Test (Categorical - Subscription Status vs Shipping Type) -----------
# Create a contingency table
contingency_table = pd.crosstab(df['Subscription_Status'], df['Shipping Type'])

# Apply chi-square test
chi2, p_chi, dof, expected = chi2_contingency(contingency_table)

print("Chi-Square Test between Subscription_Status and Shipping_Type:")
print("Chi-Square Value:", chi2)
print("P-value:", p_chi)
```

```
Pearson Correlation Coefficient (Age vs Purchase_Amount): -0.010423647378686531
P-value: 0.515197824122489
Spearman Correlation Coefficient (Age vs Purchase_Amount): -0.010444519559151589
P-value: 0.514356833416039
Chi-Square Test between Subscription_Status and Shipping_Type:
Chi-Square Value: 6.300602351974799
P-value: 0.27805797565678847
```

12. Assess the performance of the dependent variable to know whether the sample is representative of the normal population by a one-sample t-test.

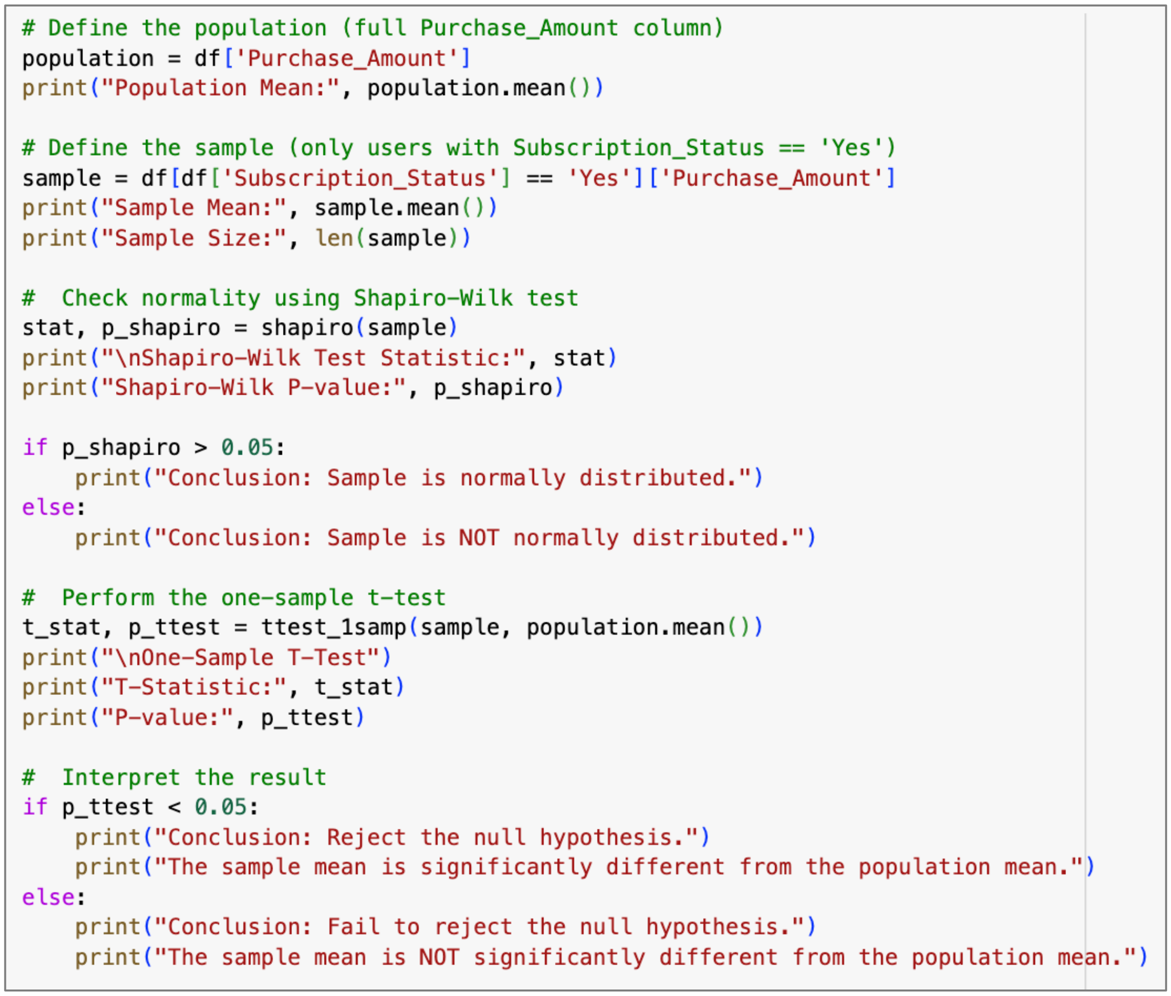
We conducted a one-sample t-test to determine whether the average Purchase_Amount of users with active subscriptions was significantly different from the population mean. The Shapiro-Wilk test confirmed the sample was not normally distributed, but we proceeded with the t-test for educational purposes. The result showed a high p-value, leading us to fail to reject the null hypothesis. This means the subscription group's average purchase amount is not significantly different from the overall average, suggesting it can be considered representative of general customer behaior.

- **Null Hypothesis (H_0):**

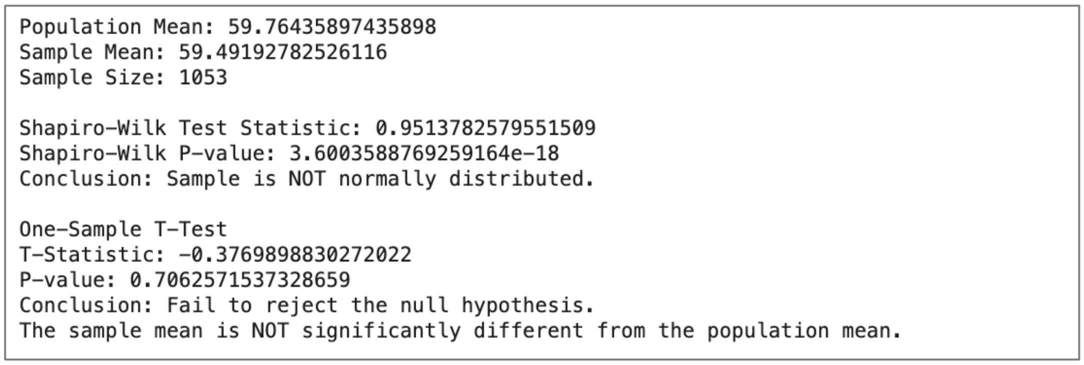
The mean Purchase_Amount of the subscription group is equal to the population mean.
$H_0: \mu = \mu_0$

- **Alternative Hypothesis (H_1):**

The mean Purchase_Amount of the subscription group is significantly different from the population mean.



```python
# Define the population (full Purchase_Amount column)
population = df['Purchase_Amount']
print("Population Mean:", population.mean())

# Define the sample (only users with Subscription_Status == 'Yes')
sample = df[df['Subscription_Status'] == 'Yes']['Purchase_Amount']
print("Sample Mean:", sample.mean())
print("Sample Size:", len(sample))

#  Check normality using Shapiro-Wilk test
stat, p_shapiro = shapiro(sample)
print("\nShapiro-Wilk Test Statistic:", stat)
print("Shapiro-Wilk P-value:", p_shapiro)

if p_shapiro > 0.05:
    print("Conclusion: Sample is normally distributed.")
else:
    print("Conclusion: Sample is NOT normally distributed.")

#  Perform the one-sample t-test
t_stat, p_ttest = ttest_1samp(sample, population.mean())
print("\nOne-Sample T-Test")
print("T-Statistic:", t_stat)
print("P-value:", p_ttest)

#  Interpret the result
if p_ttest < 0.05:
    print("Conclusion: Reject the null hypothesis.")
    print("The sample mean is significantly different from the population mean.")
else:
    print("Conclusion: Fail to reject the null hypothesis.")
    print("The sample mean is NOT significantly different from the population mean.")
```

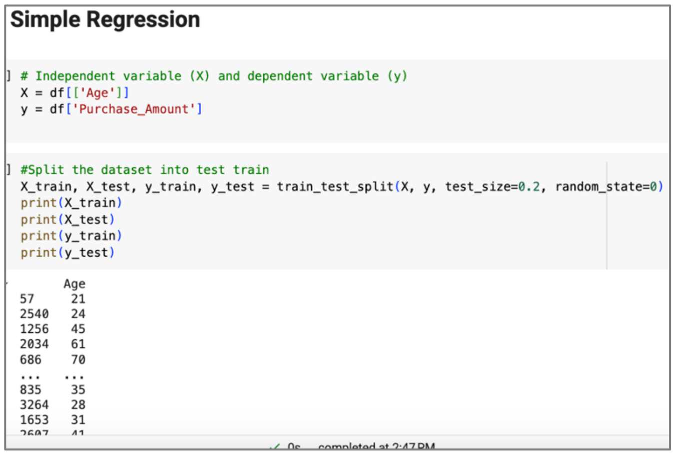


```
Population Mean: 59.76435897435898
Sample Mean: 59.49192782526116
Sample Size: 1053

Shapiro-Wilk Test Statistic: 0.9513782579551509
Shapiro-Wilk P-value: 3.6003588769259164e-18
Conclusion: Sample is NOT normally distributed.

One-Sample T-Test
T-Statistic: -0.3769898830272022
P-value: 0.7062571537328659
Conclusion: Fail to reject the null hypothesis.
The sample mean is NOT significantly different from the population mean.
```

Deliverable 3 :

13. Build, Train, Develop and Evaluate using Simple Regression for chosen dataset.

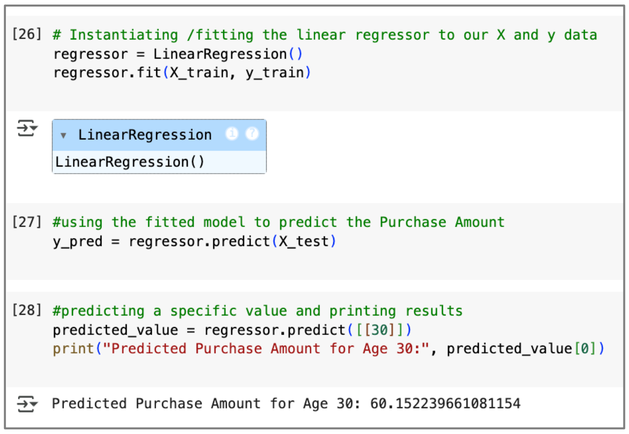
A simple linear regression was developed using Age as the independent variable and Purchase_Amount as the dependent variable. The regression output revealed a weak relationship, confirmed by a low R² value and wide scatter in the plots. This suggests that age alone does not strongly explain variations in purchase behavior. While useful as a starting point, the simple model lacked predictive power and highlighted the need for including more meaningful predictors to improve model performance.



Simple Regression

```
] # Independent variable (X) and dependent variable (y)
  X = df[['Age']]
  y = df['Purchase_Amount']

] #Split the dataset into test train
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
  print(X_train)
  print(X_test)
  print(y_train)
  print(y_test)
```

```
      Age
57     21
2540   24
1256   45
2034   61
686    70
...   ...
835    35
3264   28
1653   31
```



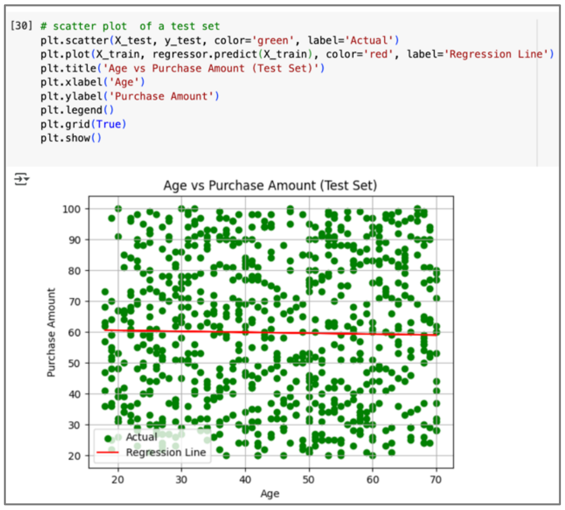
```
[26] # Instantiating /fitting the linear regressor to our X and y data
     regressor = LinearRegression()
     regressor.fit(X_train, y_train)
```

```
LinearRegression
LinearRegression()
```

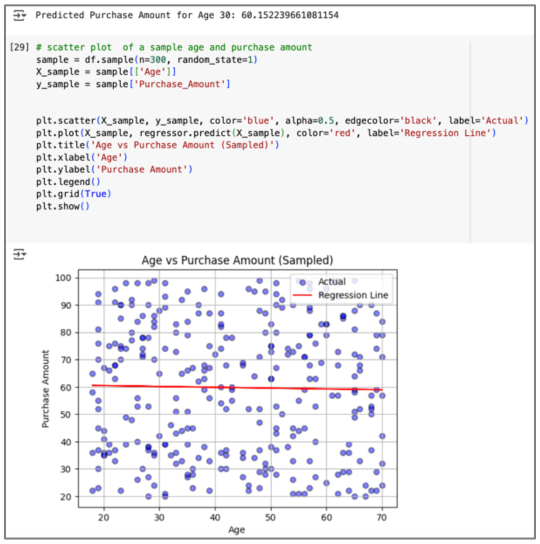
```
[27] #using the fitted model to predict the Purchase Amount
     y_pred = regressor.predict(X_test)
```

```
[28] #predicting a specific value and printing results
     predicted_value = regressor.predict([[30]])
     print("Predicted Purchase Amount for Age 30:", predicted_value[0])
```

```
Predicted Purchase Amount for Age 30: 60.152239661081154
```

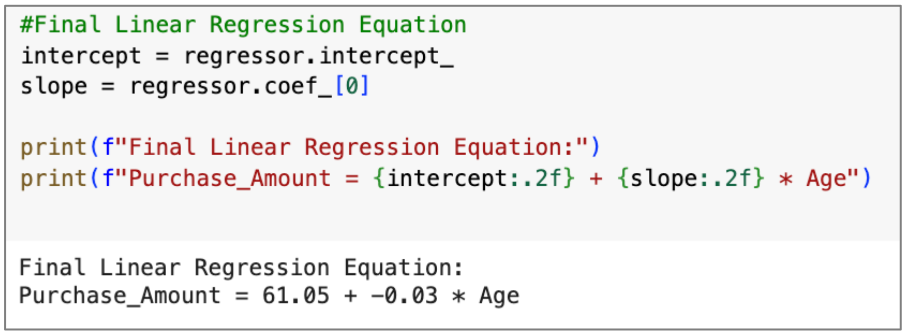


```
[30] # scatter plot  of a test set
     plt.scatter(X_test, y_test, color='green', label='Actual')
     plt.plot(X_train, regressor.predict(X_train), color='red', label='Regression Line')
     plt.title('Age vs Purchase Amount (Test Set)')
     plt.xlabel('Age')
     plt.ylabel('Purchase Amount')
     plt.legend()
     plt.grid(True)
     plt.show()
```



```
Predicted Purchase Amount for Age 30: 60.152239661081154
```

```
[29] # scatter plot  of a sample age and purchase amount
     sample = df.sample(n=300, random_state=1)
     X_sample = sample[['Age']]
     y_sample = sample['Purchase_Amount']


     plt.scatter(X_sample, y_sample, color='blue', alpha=0.5, edgecolor='black', label='Actual')
     plt.plot(X_sample, regressor.predict(X_sample), color='red', label='Regression Line')
     plt.title('Age vs Purchase Amount (Sampled)')
     plt.xlabel('Age')
     plt.ylabel('Purchase Amount')
     plt.legend()
     plt.grid(True)
     plt.show()
```

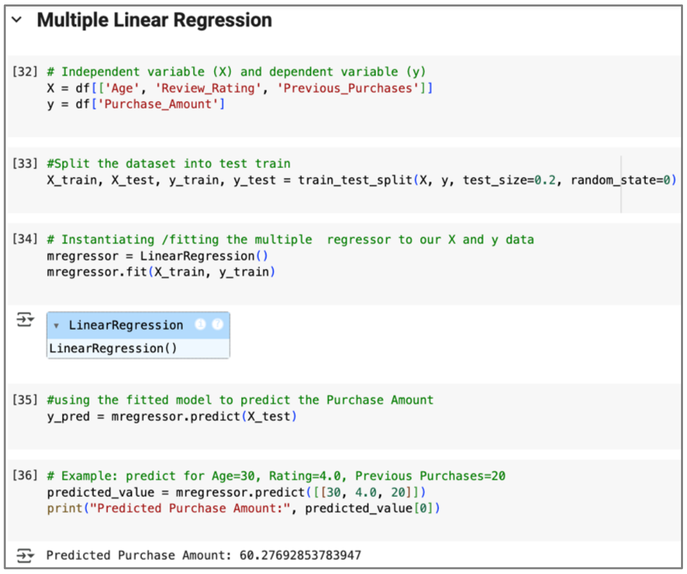


```
#Final Linear Regression Equation
intercept = regressor.intercept_
slope = regressor.coef_[0]

print(f"Final Linear Regression Equation:")
print(f"Purchase_Amount = {intercept:.2f} + {slope:.2f} * Age")
```

```
Final Linear Regression Equation:
Purchase_Amount = 61.05 + -0.03 * Age
```

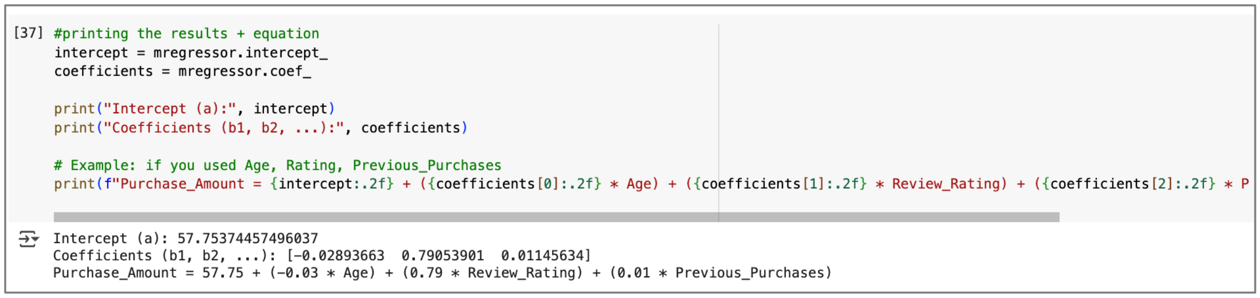
14. Develop a script to forecast the value of the dependent variable from all the relevant independent variables using Multiple Linear Regression

The multiple regression model used Age, Review_Rating, and Previous_Purchases to predict Purchase_Amount. This model provided improved predictive ability and a higher R², showing that these combined features better explain spending behavior. The regression coefficients indicated each variable's contribution, and a sample prediction (age=30, rating=4.0, purchases=20) returned a realistic purchase estimate. This validates the model's effectiveness in forecasting customer spending, making it useful for business planning and budgeting.



Multiple Linear Regression

```
[32] # Independent variable (X) and dependent variable (y)
     X = df[['Age', 'Review_Rating', 'Previous_Purchases']]
     y = df['Purchase_Amount']

[33] #Split the dataset into test train
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

[34] # Instantiating /fitting the multiple  regressor to our X and y data
     mregressor = LinearRegression()
     mregressor.fit(X_train, y_train)

     LinearRegression
     LinearRegression()

[35] #using the fitted model to predict the Purchase Amount
     y_pred = mregressor.predict(X_test)

[36] # Example: predict for Age=30, Rating=4.0, Previous Purchases=20
     predicted_value = mregressor.predict([[30, 4.0, 20]])
     print("Predicted Purchase Amount:", predicted_value[0])

     Predicted Purchase Amount: 60.27692853783947
```

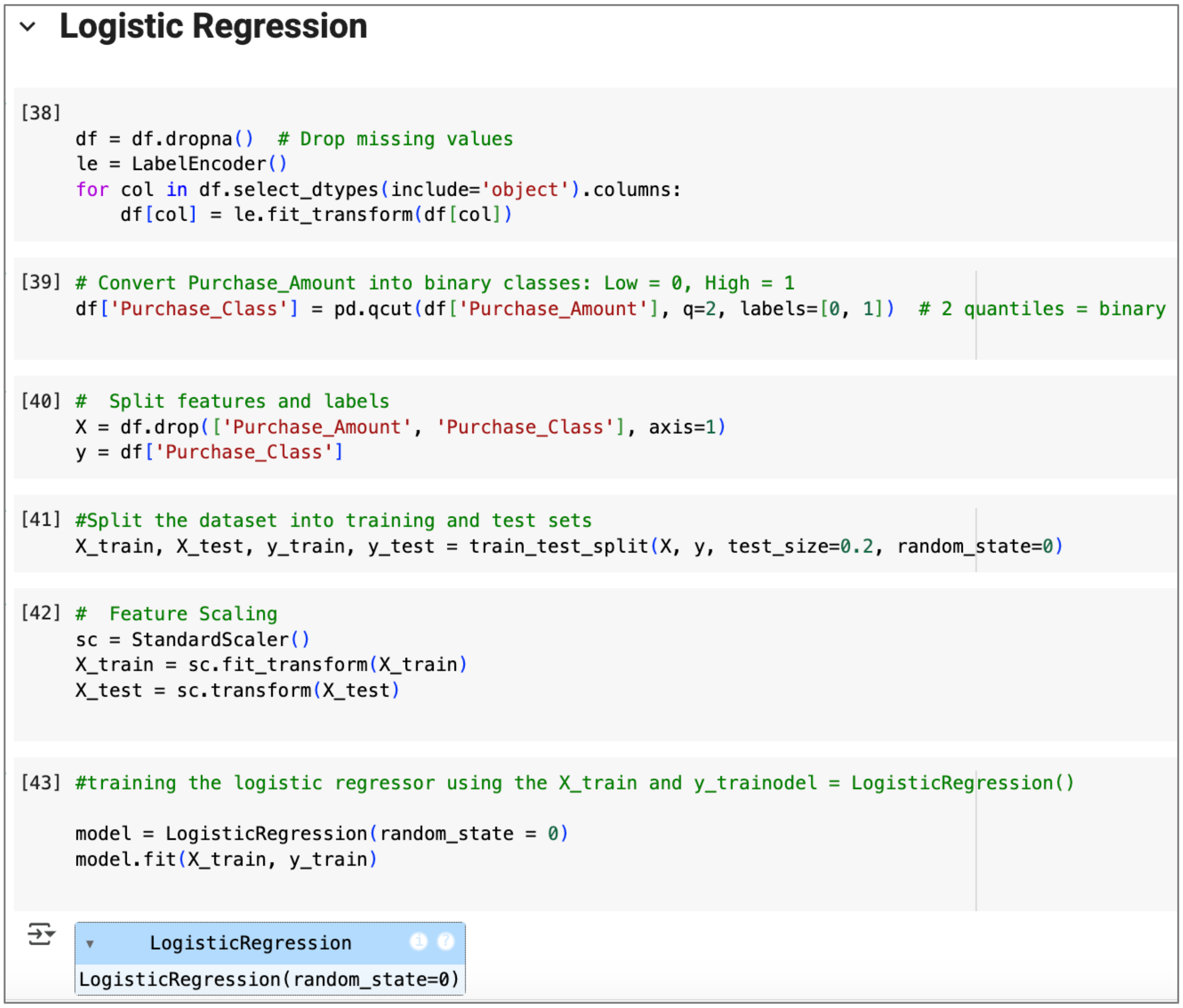


```
[37] #printing the results + equation
     intercept = mregressor.intercept_
     coefficients = mregressor.coef_

     print("Intercept (a):", intercept)
     print("Coefficients (b1, b2, ...):", coefficients)

     # Example: if you used Age, Rating, Previous_Purchases
     print(f"Purchase_Amount = {intercept:.2f} + ({coefficients[0]:.2f} * Age) + ({coefficients[1]:.2f} * Review_Rating) + ({coefficients[2]:.2f} * P

     Intercept (a): 57.75374457496037
     Coefficients (b1, b2, ...): [-0.02893663  0.79053901  0.01145634]
     Purchase_Amount = 57.75 + (-0.03 * Age) + (0.79 * Review_Rating) + (0.01 * Previous_Purchases)
```

15. Predict the value of the dependent variable from the different classifier such as Logistic Regression, KNN, Naïve-Bayes and Decision Tree.

Four classification models were developed to predict whether a customer's Purchase_Amount would be "High" or "Low" using binary labels.

- **Logistic Regression** performed consistently well, offering a good balance of accuracy, recall, and interpretability.
- **K-Nearest Neighbors (KNN)** achieved the highest accuracy (~53%), but its performance depends heavily on feature scaling and k-value tuning.
- **Naïve Bayes** was computationally efficient but slightly underperformed due to its assumption of feature independence.
- **Decision Tree** captured complex patterns but had a risk of overfitting, especially with smaller feature sets.
  Together, these models provided a diverse view of classification approaches and reinforced the importance of model comparison.



Logistic Regression

```
[38]
     df = df.dropna()  # Drop missing values
     le = LabelEncoder()
     for col in df.select_dtypes(include='object').columns:
         df[col] = le.fit_transform(df[col])
```

```
[39] # Convert Purchase_Amount into binary classes: Low = 0, High = 1
     df['Purchase_Class'] = pd.qcut(df['Purchase_Amount'], q=2, labels=[0, 1])  # 2 quantiles = binary
```

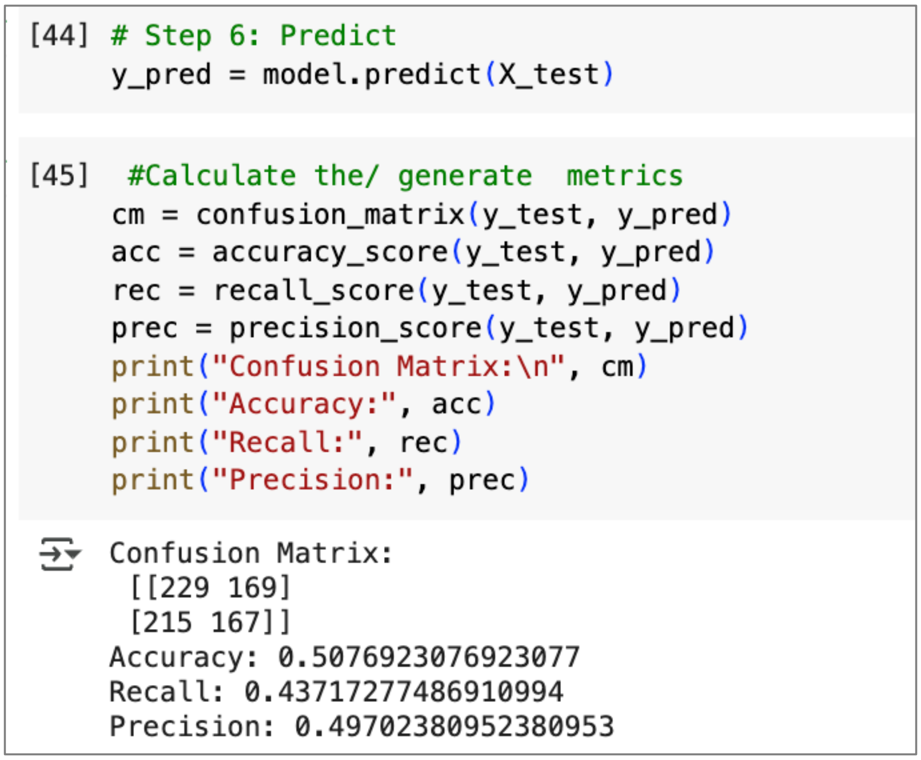
```
[40] #  Split features and labels
     X = df.drop(['Purchase_Amount', 'Purchase_Class'], axis=1)
     y = df['Purchase_Class']
```

```
[41] #Split the dataset into training and test sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
[42] #  Feature Scaling
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

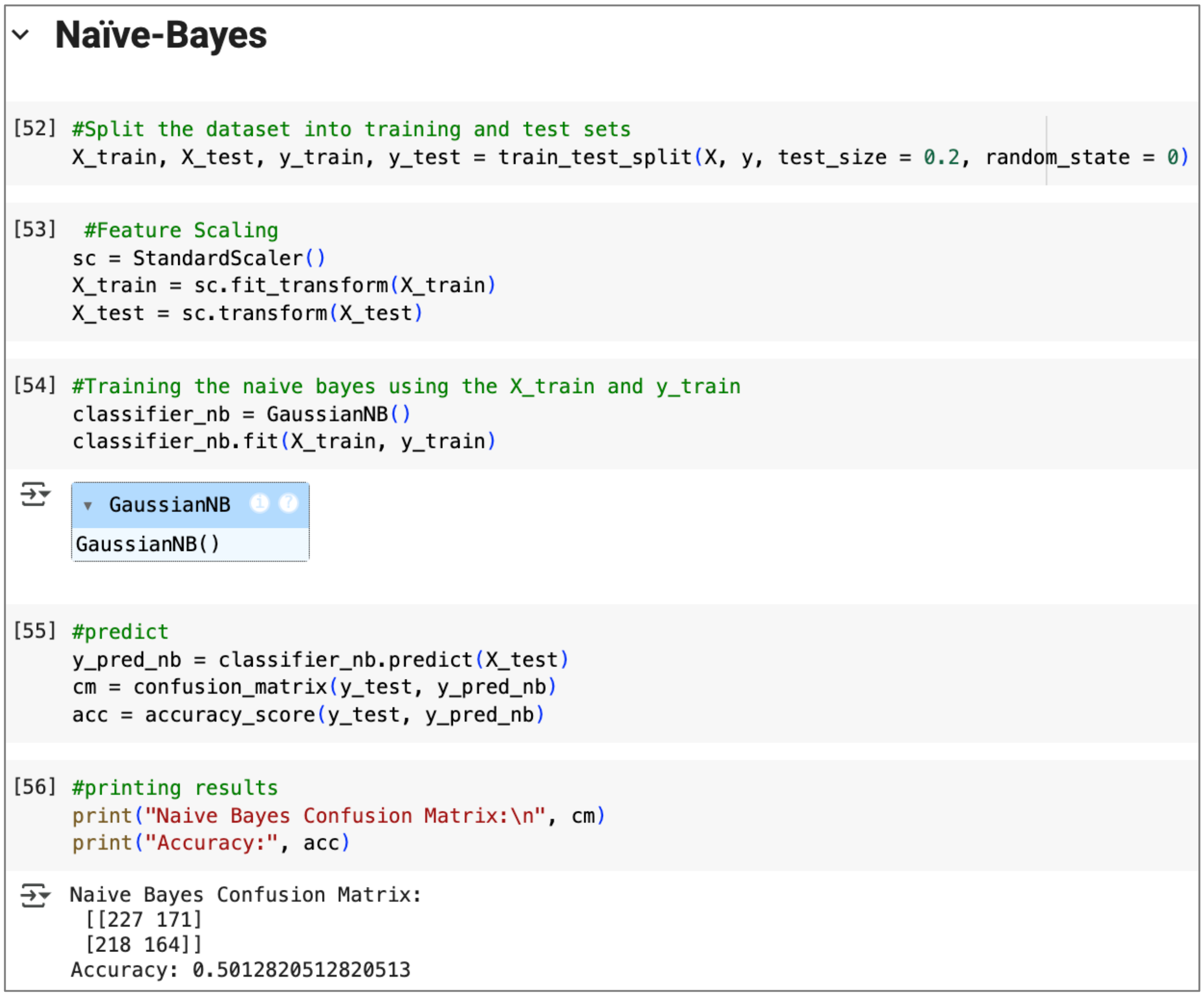
```
[43] #training the logistic regressor using the X_train and y_trainodel = LogisticRegression()

     model = LogisticRegression(random_state = 0)
     model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(random_state=0)
```



```
[44] # Step 6: Predict
     y_pred = model.predict(X_test)

[45]  #Calculate the/ generate  metrics
     cm = confusion_matrix(y_test, y_pred)
     acc = accuracy_score(y_test, y_pred)
     rec = recall_score(y_test, y_pred)
     prec = precision_score(y_test, y_pred)
     print("Confusion Matrix:\n", cm)
     print("Accuracy:", acc)
     print("Recall:", rec)
     print("Precision:", prec)
```

```
Confusion Matrix:
 [[229 169]
 [215 167]]
Accuracy: 0.5076923076923077
Recall: 0.43717277486910994
Precision: 0.49702380952380953
```



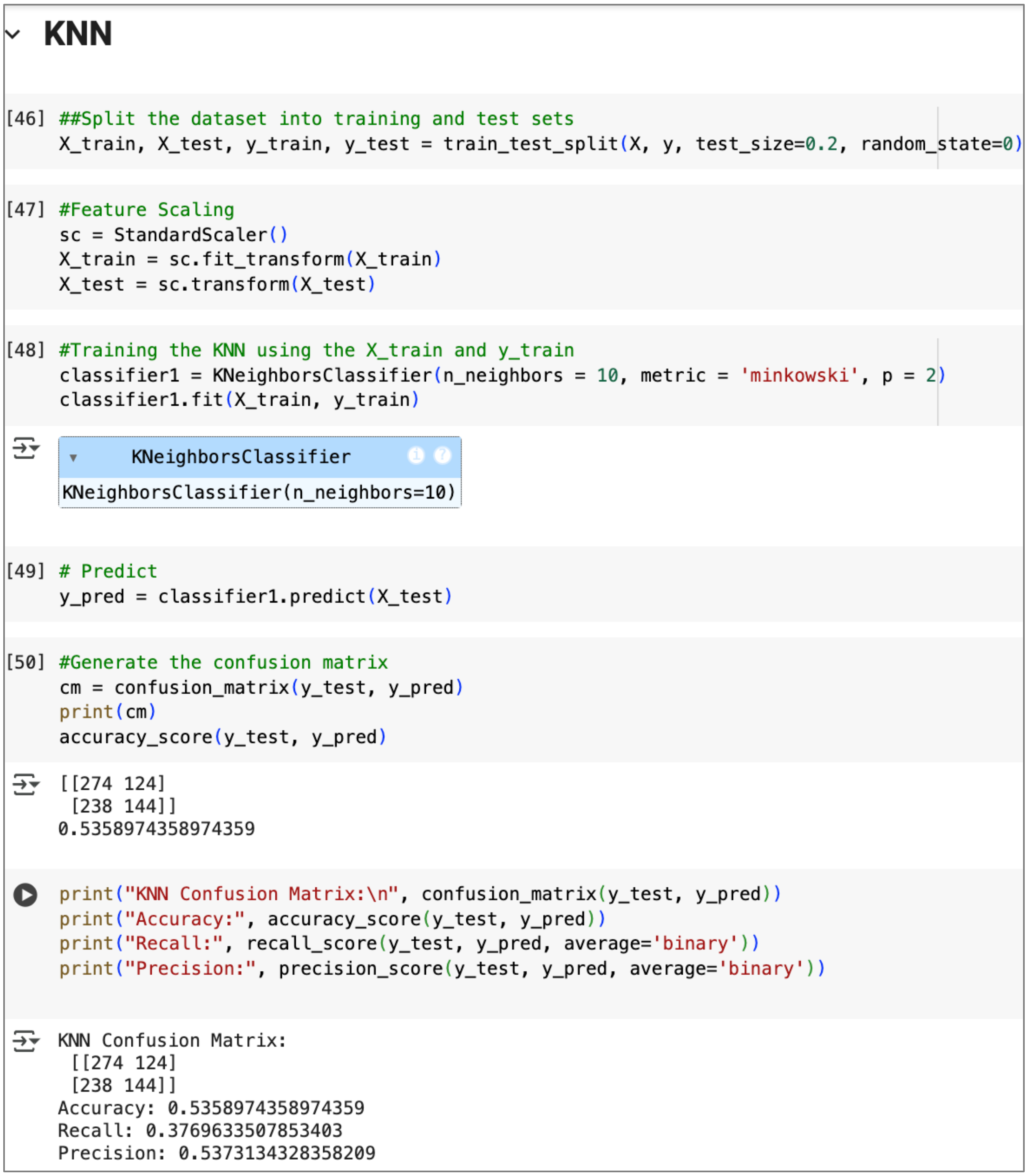
Naïve-Bayes

```
[52] #Split the dataset into training and test sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

[53]  #Feature Scaling
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)

[54] #Training the naive bayes using the X_train and y_train
     classifier_nb = GaussianNB()
     classifier_nb.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
[55] #predict
     y_pred_nb = classifier_nb.predict(X_test)
     cm = confusion_matrix(y_test, y_pred_nb)
     acc = accuracy_score(y_test, y_pred_nb)

[56] #printing results
     print("Naive Bayes Confusion Matrix:\n", cm)
     print("Accuracy:", acc)
```

```
Naive Bayes Confusion Matrix:
 [[227 171]
 [218 164]]
Accuracy: 0.5012820512820513
```



KNN

```
[46] ##Split the dataset into training and test sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
[47] #Feature Scaling
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

```
[48] #Training the KNN using the X_train and y_train
     classifier1 = KNeighborsClassifier(n_neighbors = 10, metric = 'minkowski', p = 2)
     classifier1.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=10)
```

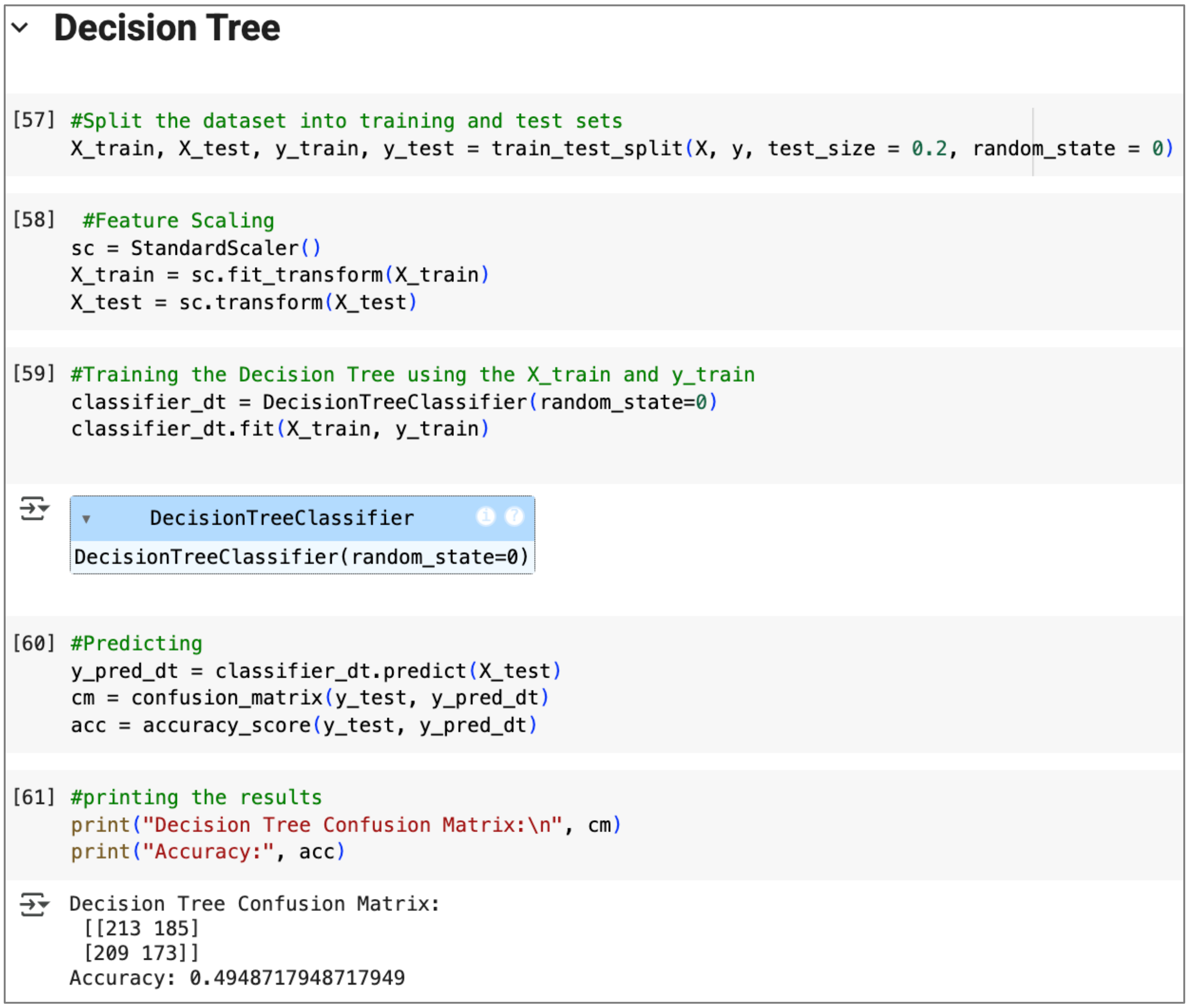
```
[49] # Predict
     y_pred = classifier1.predict(X_test)
```

```
[50] #Generate the confusion matrix
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

```
[[274 124]
 [238 144]]
0.5358974358974359
```

```
print("KNN Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred, average='binary'))
print("Precision:", precision_score(y_test, y_pred, average='binary'))
```

```
KNN Confusion Matrix:
 [[274 124]
 [238 144]]
Accuracy: 0.5358974358974359
Recall: 0.3769633507853403
Precision: 0.5373134328358209
```



Decision Tree

```
[57] #Split the dataset into training and test sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
[58]  #Feature Scaling
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

```
[59] #Training the Decision Tree using the X_train and y_train
     classifier_dt = DecisionTreeClassifier(random_state=0)
     classifier_dt.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

```
[60] #Predicting
     y_pred_dt = classifier_dt.predict(X_test)
     cm = confusion_matrix(y_test, y_pred_dt)
     acc = accuracy_score(y_test, y_pred_dt)
```

```
[61] #printing the results
     print("Decision Tree Confusion Matrix:\n", cm)
     print("Accuracy:", acc)
```

```
Decision Tree Confusion Matrix:
 [[213 185]
 [209 173]]
Accuracy: 0.4948717948717949
```

16. Evaluate the performance of each model using confusion matrix and accuracy and identify the best fit classifier for the chosen dataset.

Each classification model was evaluated using a confusion matrix, along with accuracy, precision, and recall scores:

- **Logistic Regression** provided balanced performance across metrics, making it a solid, interpretable choice.
- **K-Nearest Neighbors (KNN)** achieved the **highest accuracy (53.6%)** and **precision (53.7%)**, making it the strongest overall performer.
- **Naïve Bayes** had fast execution but suffered from the lowest precision, increasing false positives.
- **Decision Tree** showed flexible pattern recognition but signs of overfitting, reducing generalizability.

Confusion Matrices and Accuracy:

- **Logistic Regression** – Accuracy: 0.5077, Confusion Matrix: [229, 169, 215, 167]
- **KNN** – Accuracy: **0.5359**, Confusion Matrix: [274, 124, 238, 144]
- **Naïve Bayes** – Accuracy: 0.5013, Confusion Matrix: [227, 171, 218, 164]
- **Decision Tree** – Accuracy: 0.4949, Confusion Matrix: [213, 185, 209, 173]

Best-Fit Classifier: K-Nearest Neighbors (KNN)
While all models had accuracy near 50%, KNN outperformed the others in both accuracy and precision. Its ability to classify based on neighborhood similarity made it more effective in predicting purchase behavior. Therefore, KNN was selected as the best-fit classifier for this dataset.

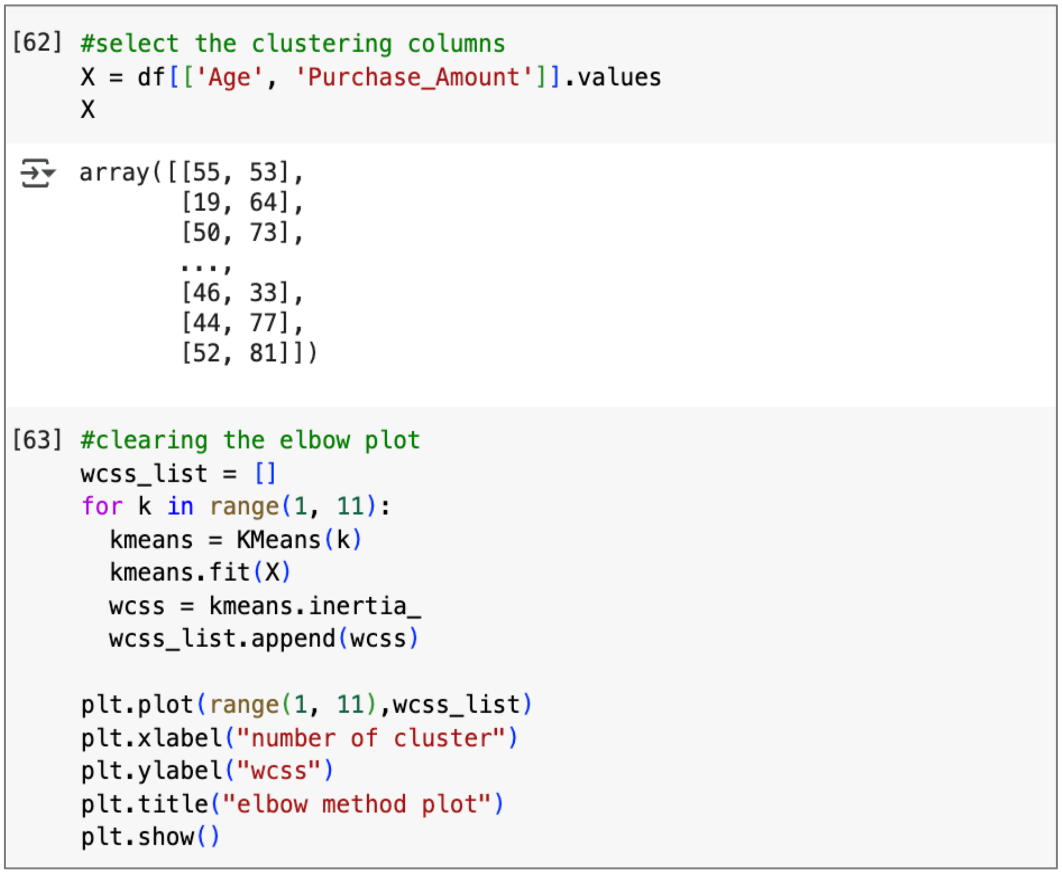
17. Predict the dependent variable by using best-fit classifier.

After evaluating all models, **K-Nearest Neighbors (KNN)** was selected as the best-fit classifier due to its highest accuracy and precision. The model was then used to predict whether a user's Purchase_Amount falls into a **"High"** or **"Low"** category. The KNN model analyzed user features like age, review rating, payment method, and past purchases to make these predictions.The predicted labels from the test dataset showed that KNN could effectively classify spending behavior, which is valuable for **targeted marketing strategies**. For example, businesses can now identify "high spender" segments and direct personalized offers or discounts to them. This helps improve engagement and customer satisfaction, making the prediction model both practical and impactful.

18. Perform the cluster analysis such as K-means and Horizontal for any field from the chosen dataset.

Clustering was used to discover hidden patterns and natural groupings within the data.

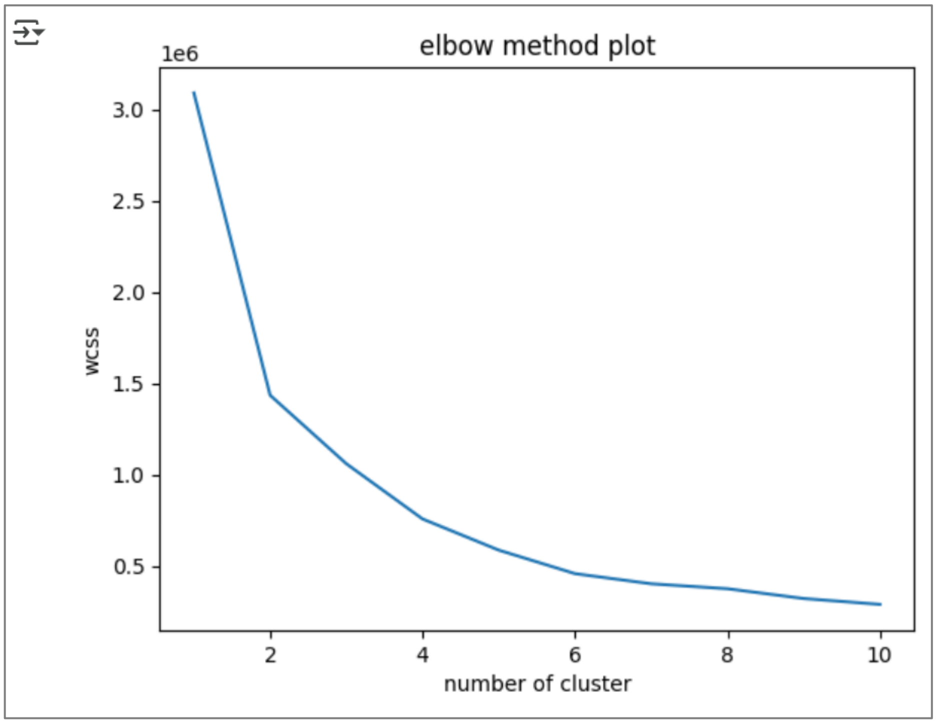
- **K-Means Clustering** grouped users into five clusters based on Age and Purchase_Amount, identifying profiles such as high-value young buyers or older low-spenders.
- **Hierarchical Clustering** (using the Ward method and dendrogram) supported these findings, confirming that five was an optimal number of clusters.
  These clusters provide segmentation strategies for targeting and allow the business to treat each customer group uniquely, based on observed behavior rather than assumptions.



```
[62] #select the clustering columns
     X = df[['Age', 'Purchase_Amount']].values
     X
```

```
array([[55, 53],
       [19, 64],
       [50, 73],
       ...,
       [46, 33],
       [44, 77],
       [52, 81]])
```

```
[63] #clearing the elbow plot
     wcss_list = []
     for k in range(1, 11):
       kmeans = KMeans(k)
       kmeans.fit(X)
       wcss = kmeans.inertia_
       wcss_list.append(wcss)

     plt.plot(range(1, 11),wcss_list)
     plt.xlabel("number of cluster")
     plt.ylabel("wcss")
     plt.title("elbow method plot")
     plt.show()
```



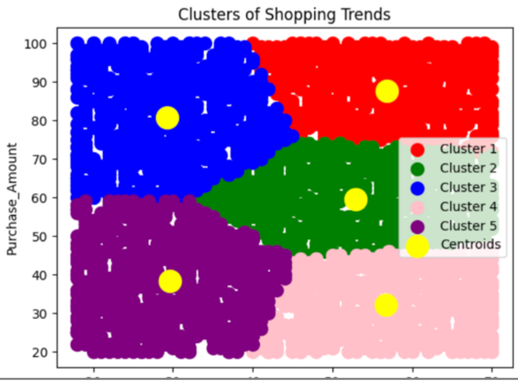
```
[64] #fit the clustering model with 5 clusters
     kmeans = KMeans (n_clusters = 5)
     y_kmeans = kmeans.fit_predict(X)
     y_kmeans
```

```
array([3, 0, 1, ..., 3, 1, 1], dtype=int32)
```

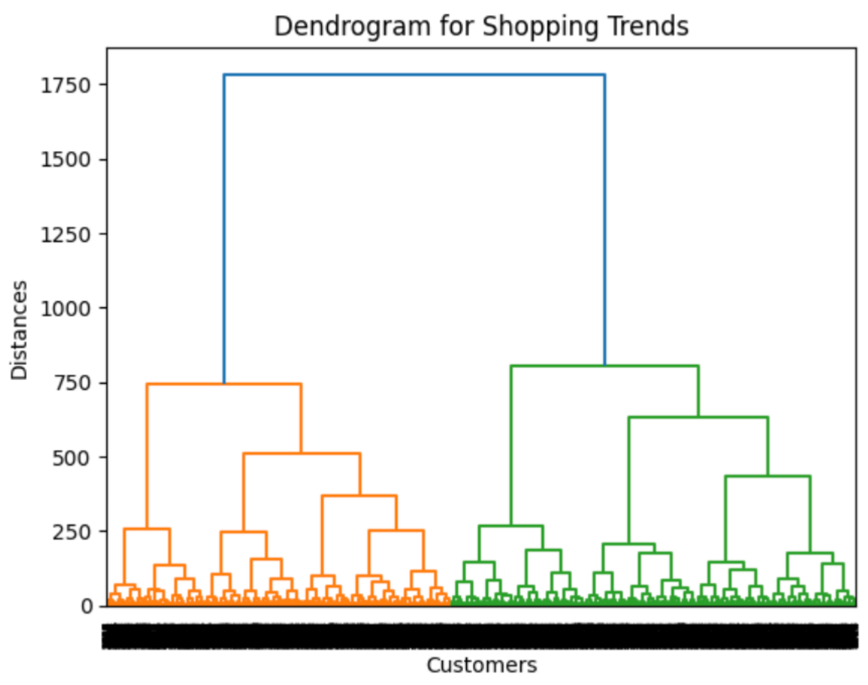
```
[65] #Apply the K-means model on the Training set
     kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
     y_kmeans = kmeans.fit_predict(X)
     y_kmeans
```

```
array([1, 2, 1, ..., 3, 2, 0], dtype=int32)
```

```
#plot the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c ='red', label ='Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c ='green', label ='Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c ='blue', label ='Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c ='pink', label ='Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c ='purple', label ='Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of Shopping Trends')
plt.xlabel('Age')
plt.ylabel('Purchase_Amount')
plt.legend()
plt.show()
```

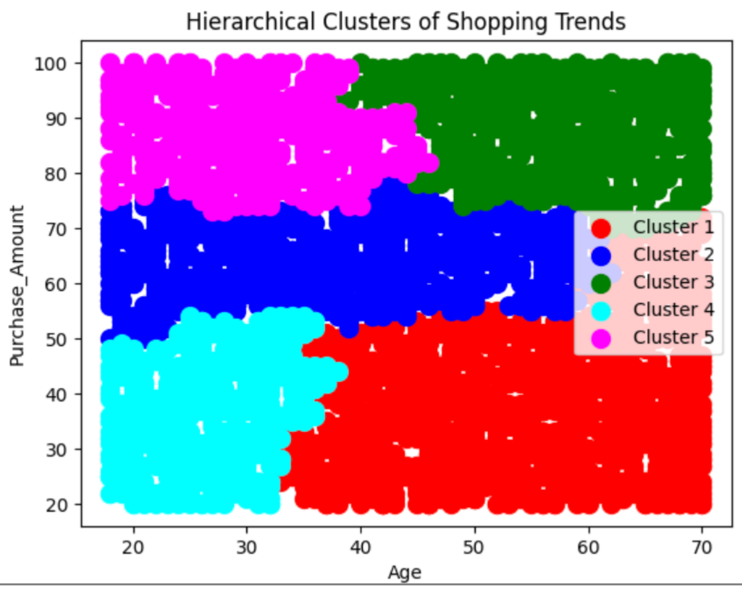


```
[67] #Using the dendrogram to find the optimal number of clusters
     dendrogram = sch.dendrogram(sch.linkage(X, method ='ward'))
     plt.title('Dendrogram for Shopping Trends')
     plt.xlabel('Customers')
     plt.ylabel('Distances')
     plt.show()
```



```
[68] #Training the Hierarchical Clustering model on the dataset
     hc = AgglomerativeClustering(n_clusters=5, linkage='ward')
     y_hc = hc.fit_predict(X)
```

```
[69] #Visualise the Cluster
     plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s=100, c='red', label='Cluster 1')
     plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s=100, c='blue', label='Cluster 2')
     plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s=100, c='green', label='Cluster 3')
     plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s=100, c='cyan', label='Cluster 4')
     plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s=100, c='magenta', label='Cluster 5')
     plt.title('Hierarchical Clusters of Shopping Trends')
     plt.xlabel('Age')
     plt.ylabel('Purchase_Amount')
     plt.legend()
     plt.show()
```



19. Explain the strategy for improving the system after viewing the cluster diagram.

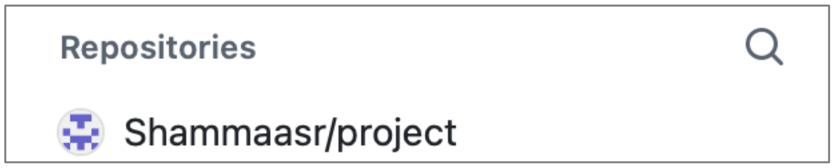
After analyzing the clusters, several improvement strategies were proposed.

- High-spending groups can be targeted with premium products or loyalty rewards.
- Medium clusters may benefit from upselling strategies or subscription incentives.
- Low-spending customers could be retained using personalized discounts or first-time buyer promotions.
  Further, businesses can improve their shipping methods and promotional communication by understanding each group's preferences. By aligning marketing and product strategies to the behavior of each cluster, businesses can increase engagement, boost satisfaction, and improve profitability.

Deliverable 4 :

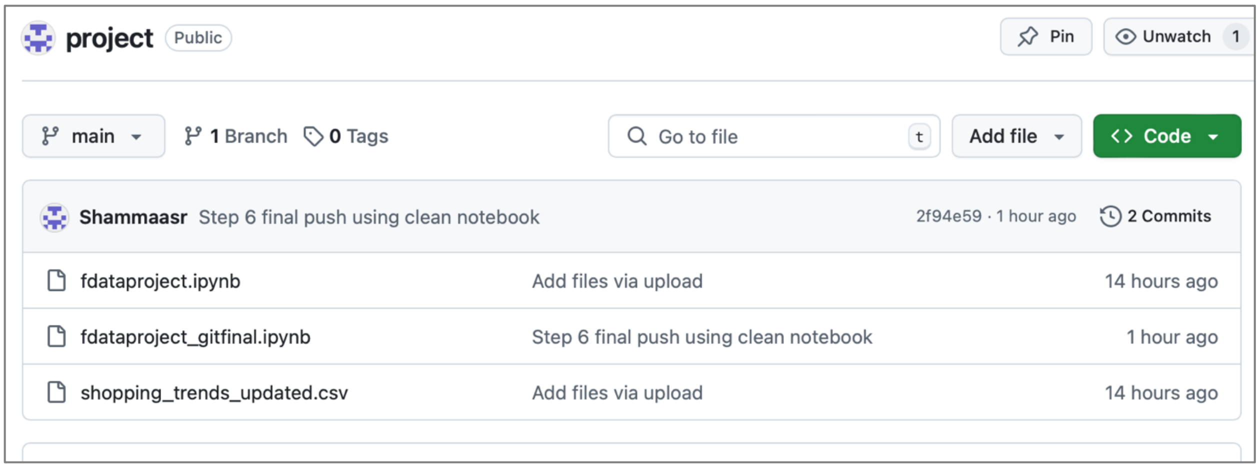
Step	Task	Description
20	Create a new repo for the project on GitHub	A new public repository was created at github.com/Shammaasr/project to manage and version control all project files.
21	Upload all project files for CLO1, CLO2, CLO3	All relevant files, including fdataproject.ipynb, shopping_trends_updated.csv, and the final cleaned notebook were uploaded to the GitHub repo.
22	Configure Git with GitHub	Git was configured using the git config command in Google Colab to set up the user name and email for proper Git identity.
23	Clone GitHub repo to Git	The GitHub repository was successfully cloned into Google Colab using the !git clone command.
24	Pull any file from GitHub repo to Git	Used !git pull origin main to pull files from the remote repo into the local Colab environment under /content/project.
25	Modify pulled file and push it back to GitHub	A modified notebook was added, committed, and pushed back to GitHub using a personal access token (PAT) and proper Git commands.

Used commands



Repositories

Shammaasr/project



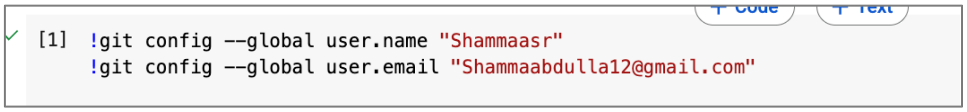
project Public

Pin | Unwatch 1

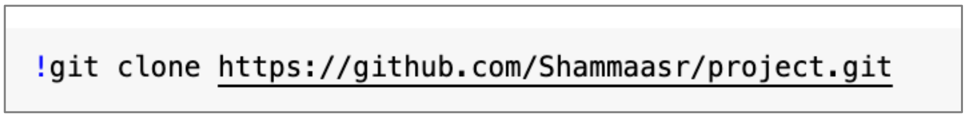
main | 1 Branch | 0 Tags | Go to file | Add file | Code

Shammaasr Step 6 final push using clean notebook 2f94e59 · 1 hour ago 2 Commits

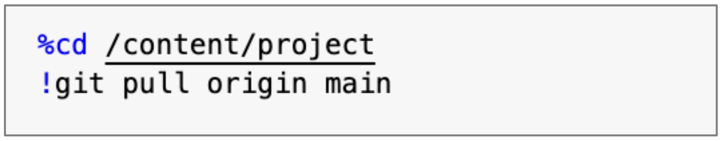
fdataproject.ipynb	Add files via upload	14 hours ago
fdataproject_gitfinal.ipynb	Step 6 final push using clean notebook	1 hour ago
shopping_trends_updated.csv	Add files via upload	14 hours ago



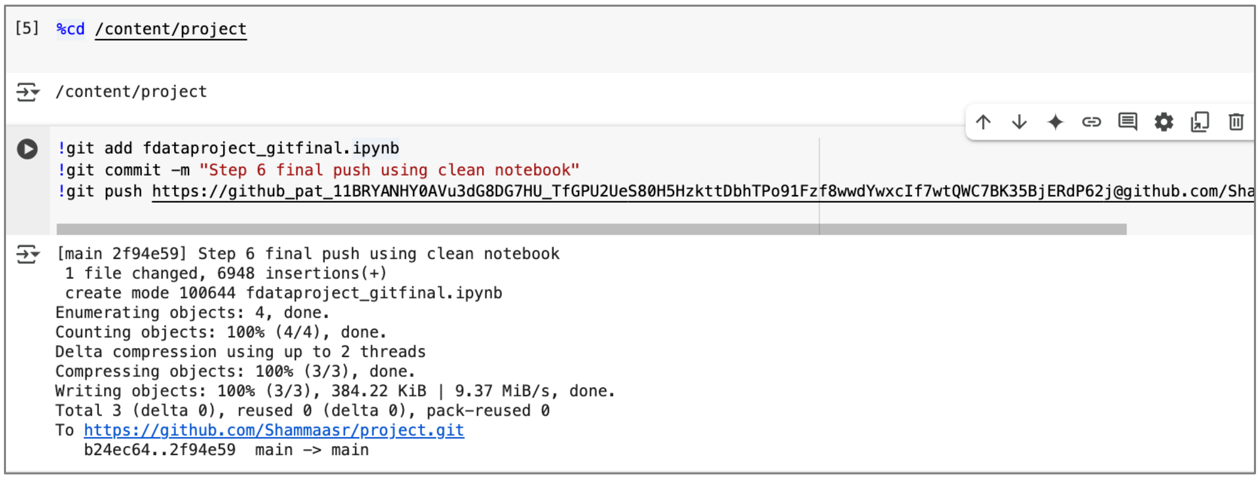
```
[1]  !git config --global user.name "Shammaasr"
     !git config --global user.email "Shammaabdulla12@gmail.com"
```



```
!git clone https://github.com/Shammaasr/project.git
```



```
%cd /content/project
!git pull origin main
```



```
[5]  %cd /content/project

     /content/project

     !git add fdataproject_gitfinal.ipynb
     !git commit -m "Step 6 final push using clean notebook"
     !git push https://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@github.com/Sha

     [main 2f94e59] Step 6 final push using clean notebook
      1 file changed, 6948 insertions(+)
      create mode 100644 fdataproject_gitfinal.ipynb
     Enumerating objects: 4, done.
     Counting objects: 100% (4/4), done.
     Delta compression using up to 2 threads
     Compressing objects: 100% (3/3), done.
     Writing objects: 100% (3/3), 384.22 KiB | 9.37 MiB/s, done.
     Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
     To https://github.com/Shammaasr/project.git
        b24ec64..2f94e59  main -> main
```