Phase 1: Data Cleaning and Preprocessing

In Phase 1, I've performed the following steps to clean and pre-process the dataset:

1. Created a pandas dataframe from retail_sales_dataset.csv
2. Standardized column names to exclude spaces and replace them with underscores
3. Changed date column's type to datetime
4. Handled any missing values
5. Removes rows with the date missing and the date ending in year 2024
6. Dropped the transaction_id and customer_id columns
7. Summarized dataset features
8. Normalized numeric columns for age, quantity, price_per_unit, and total_amount
9. Saved the processed dataset as processed_retail_sales_dataset.csv
10. Checked if normalization has worked for numerical columns

In [74]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix

#Loading the dataset
dataset_file_path = "C:\\Users\\TazeenQ\\team27_project\\data\\raw\\retail_sales
retail_data = pd.read_csv(dataset_file_path)

#Displaying the first couple of rows to check the data
retail_data.head(), retail_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Transaction ID    1000 non-null   int64
 1   Date              1000 non-null   object
 2   Customer ID       1000 non-null   object
 3   Gender            1000 non-null   object
 4   Age               1000 non-null   int64
 5   Product Category  1000 non-null   object
 6   Quantity          1000 non-null   int64
 7   Price per Unit    1000 non-null   int64
 8   Total Amount      1000 non-null   int64
dtypes: int64(5), object(4)
memory usage: 70.4+ KB
```

Out[74]:
```
(   Transaction ID        Date Customer ID  Gender  Age Product Category  \
 0               1  2023-11-24     CUST001    Male   34           Beauty
 1               2  2023-02-27     CUST002  Female   26         Clothing
 2               3  2023-01-13     CUST003    Male   50      Electronics
 3               4  2023-05-21     CUST004    Male   37         Clothing
 4               5  2023-05-06     CUST005    Male   30           Beauty

    Quantity  Price per Unit  Total Amount
 0         3              50           150
 1         2             500          1000
 2         1              30            30
 3         1             500           500
 4         2              50           100  ,
 None)
```

In [75]:
```
#Renaming column names
retail_data.rename(columns = lambda x: x.strip().replace(" ", "_").lower(), inpl

#Standardizing data types
#Converting date to datetime
retail_data['date'] = pd.to_datetime(retail_data['date'], errors = 'coerce')

#Handling missing values by replacing blanks with NaN and checking for missing v
retail_data.replace("", np.nan, inplace = True)
missing_summary = retail_data.isnull().sum()

#Dropping rows with date missing
retail_data.dropna(subset = ['date'], inplace = True)

#Filtering data, the dataset has values from 2023 and only 2 rows with data from
retail_data = retail_data[retail_data['date'].dt.year != 2024].reset_index(drop

#Dropping unwanted columns: transaction_id and customer_id
drop_these_columns = ['transaction_id', 'customer_id']
retail_data.drop(columns=drop_these_columns, inplace = True, errors = 'ignore')

#Summarizing dataset features
summary_stats = retail_data.describe(include = 'all')

#Choosing numerical columns to normalize
numerical_cols = ['age', 'quantity', 'price_per_unit', 'total_amount']

#Normalizng them
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(retail_data[numerical_cols])

#Creating a new DataFrame for the normalized data
normalized_dataset = retail_data.copy()
normalized_dataset[numerical_cols] = pd.DataFrame(normalized_data, columns = num

#Hot encoding my categorical variables
#categorical_columns = ['gender', 'product_category']
#normalized_dataset = pd.get_dummies(normalized_dataset, columns = categorical_c

#Saving the normalized dataset to a new file
cleaned_dataset_file_path = "C:\\Users\\TazeenQ\\team27_project\\data\\processed
normalized_dataset.to_csv(cleaned_dataset_file_path, index = False)

normalized_dataset.info(), normalized_dataset.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 998 entries, 0 to 997
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   date              998 non-null    datetime64[ns]
 1   gender            998 non-null    object
 2   age               998 non-null    float64
 3   product_category  998 non-null    object
 4   quantity          998 non-null    float64
 5   price_per_unit    998 non-null    float64
 6   total_amount      998 non-null    float64
dtypes: datetime64[ns](1), float64(4), object(2)
memory usage: 54.7+ KB
```

Out[75]:

```
(None,
         date  gender       age product_category  quantity  price_per_unit  \
  0 2023-11-24    Male  0.347826           Beauty  0.666667        0.052632
  1 2023-02-27  Female  0.173913         Clothing  0.333333        1.000000
  2 2023-01-13    Male  0.695652      Electronics  0.000000        0.010526
  3 2023-05-21    Male  0.413043         Clothing  0.000000        1.000000
  4 2023-05-06    Male  0.260870           Beauty  0.333333        0.052632

     total_amount
  0      0.063291
  1      0.493671
  2      0.002532
  3      0.240506
  4      0.037975  )
```

In [76]:

```
#Checking normalization for each numerical column
for col in numerical_cols:
    min_val = normalized_dataset[col].min()
    max_val = normalized_dataset[col].max()
    print(f"Column '{col}' - Min: {min_val}, Max: {max_val}")

#Checking if all values are between 0 and 1
if all((normalized_dataset[col].min() >= 0) & (normalized_dataset[col].max() <=
    print("All numerical columns are properly normalized.")
else:
    print("Some columns are not properly normalized.")
```

```
Column 'age' - Min: 0.0, Max: 1.0
Column 'quantity' - Min: 0.0, Max: 1.0
Column 'price_per_unit' - Min: 0.0, Max: 1.0
Column 'total_amount' - Min: 0.0, Max: 1.0
All numerical columns are properly normalized.
```

Phase 2: Initial Exploration with Visualizations

In Phase 2, I have ran the following exploratory analyses on the processed dataset:

1. Class Imbalance Analysis to see the distribution of high and low spenders. I've set the threshold for high spenders as those in the top 25%.
2. Categoric and Numeric Variable Analysis to see if there are any variations between gender and product category.

3. Chi-Square Test to see if there is any significant relationship between gender and product category.
4. Aggregate analysis of average spending by age and gender to see if either have an influence on spending behaviour.
5. Aggregate analysis of average spending by product category and age group to see if either have an influence on spending behaviour.
6. Correlation analysis of numerical features age, quantity, price_per_unit, total_amount, and high_spender.

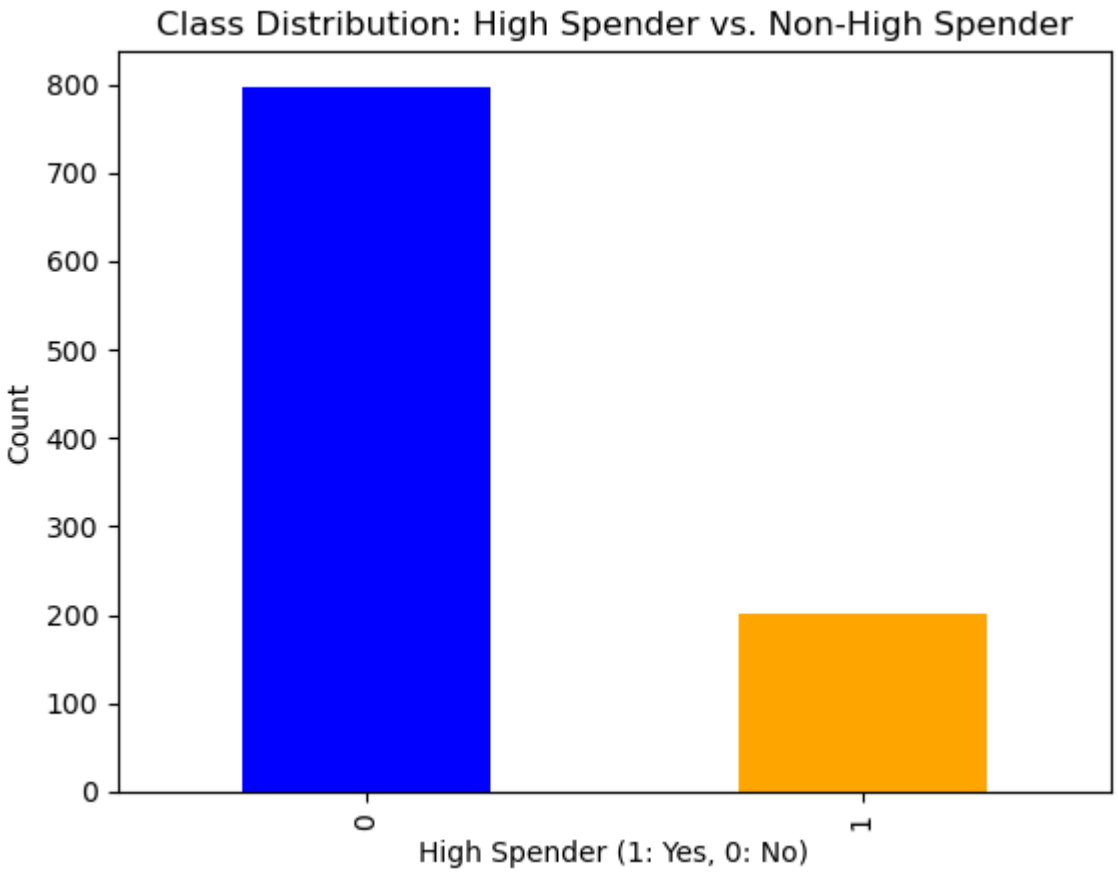
Class Imbalance Analysis

A balanced dataset should have a similar number of high spenders and low spenders. If there is an imbalance in the dataset, I will need to adjust the classification model with methods like taking a subset of the data, oversampling, or SMOTE.

As we can see for the plot below, this dataset is highly imbalanced because there are alot more low spenders than high spendes. This will likely contribute to poor model performance if I don't adjust the dataset.

Upon consultation with Farzaneh, I will take a subset of the dataset with equal number of high and low spenders, with low spenders being chosen randomly.

In [60]:
```
#Defining a threshold for high spender (top 25% of total_amount)
threshold = normalized_dataset['total_amount'].quantile(0.75)

#Creating a new column called high+spender
normalized_dataset['high_spender'] = normalized_dataset['total_amount'].apply(la

#Ploting the distribution
normalized_dataset['high_spender'].value_counts().plot(kind='bar', color=['blue'
plt.title("Class Distribution: High Spender vs. Non-High Spender")
plt.xlabel("High Spender (1: Yes, 0: No)")
plt.ylabel("Count")
plt.show()
```



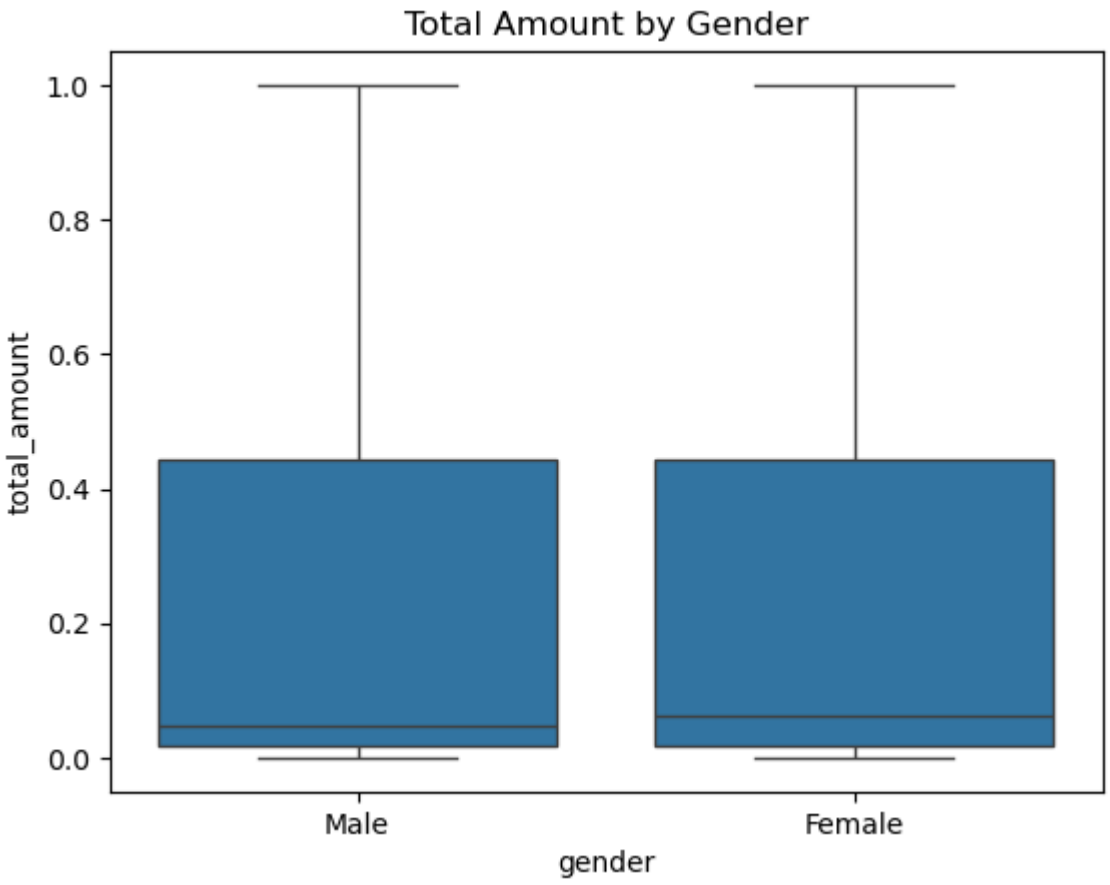
Categoric and Numeric Variable Analysis

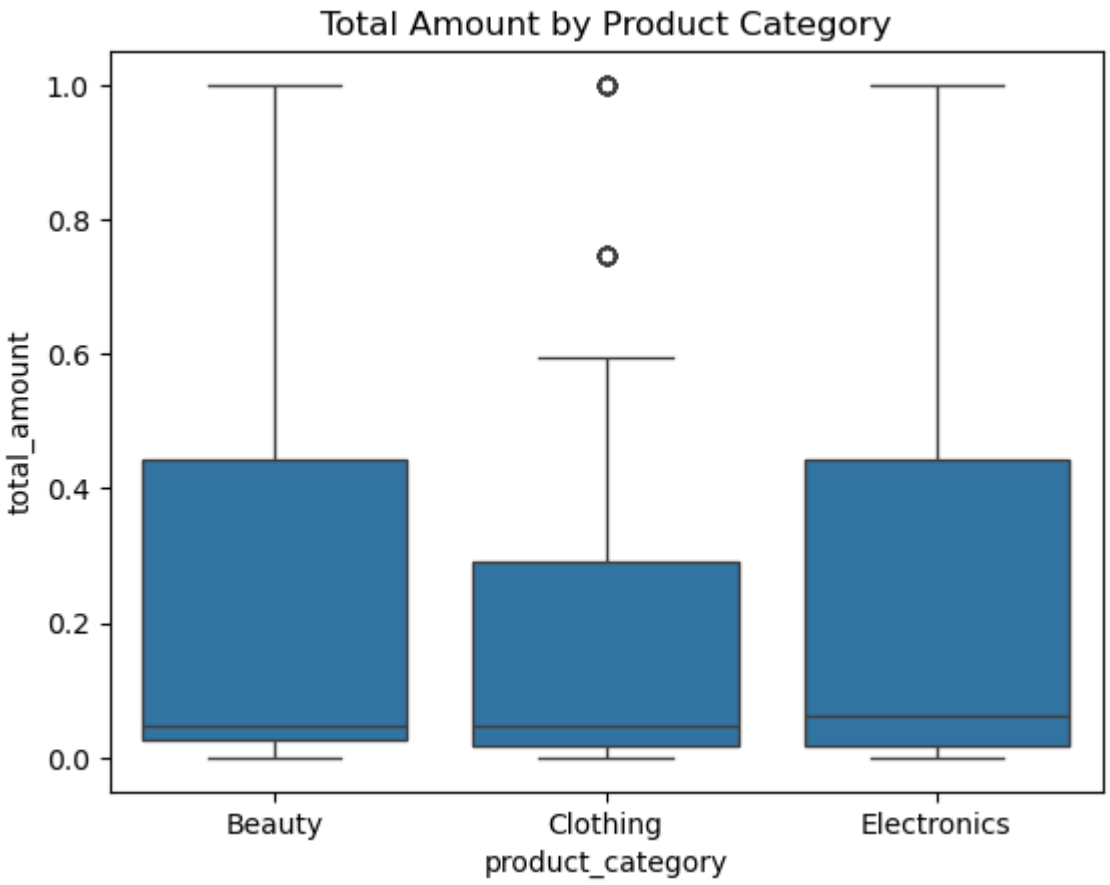
In general, differences in median spending between categories and by gender means more variation within groups. This analysis is also going to show us spending patterns.

As we can see in the plot below, customers consistently spend higher in Beauty and Electronics categories and lower spending in Clothing category.

However, there is very little variation in spending by gender. This means that the spending patterns are pretty similar for both the genders.

In [61]:
```
#Creating a box plot for gender
sns.boxplot(data=normalized_dataset, x='gender', y='total_amount')
plt.title("Total Amount by Gender")
plt.show()

#Creating a box plot for product category
sns.boxplot(data=normalized_dataset, x='product_category', y='total_amount')
plt.title("Total Amount by Product Category")
plt.show()
```





Chi-Square Tests (Checking association between gender and product category)

A p-value of less than 0.05 indicates a significant relationship between two variables.

As we can see in the output below, gender and product category do not have a significant relationship (p-value of 0.42).

This means that certain product categories do not appeal to one gender more than the other.

In [62]:
```
#Creating a contingency table
contingency_table = pd.crosstab(normalized_dataset['gender'], normalized_dataset

#Chi-Square test
chi2, p, dof, expected = chi2_contingency(contingency_table)
print(f"Chi2 Statistic: {chi2}, p-value: {p}")

#Printing results
if p < 0.05:
    print("Significant association between Gender and Product Category.")
else:
    print("No significant association between Gender and Product Category.")
```

```
Chi2 Statistic: 1.7536061144312953, p-value: 0.4161110708113299
No significant association between Gender and Product Category.
```

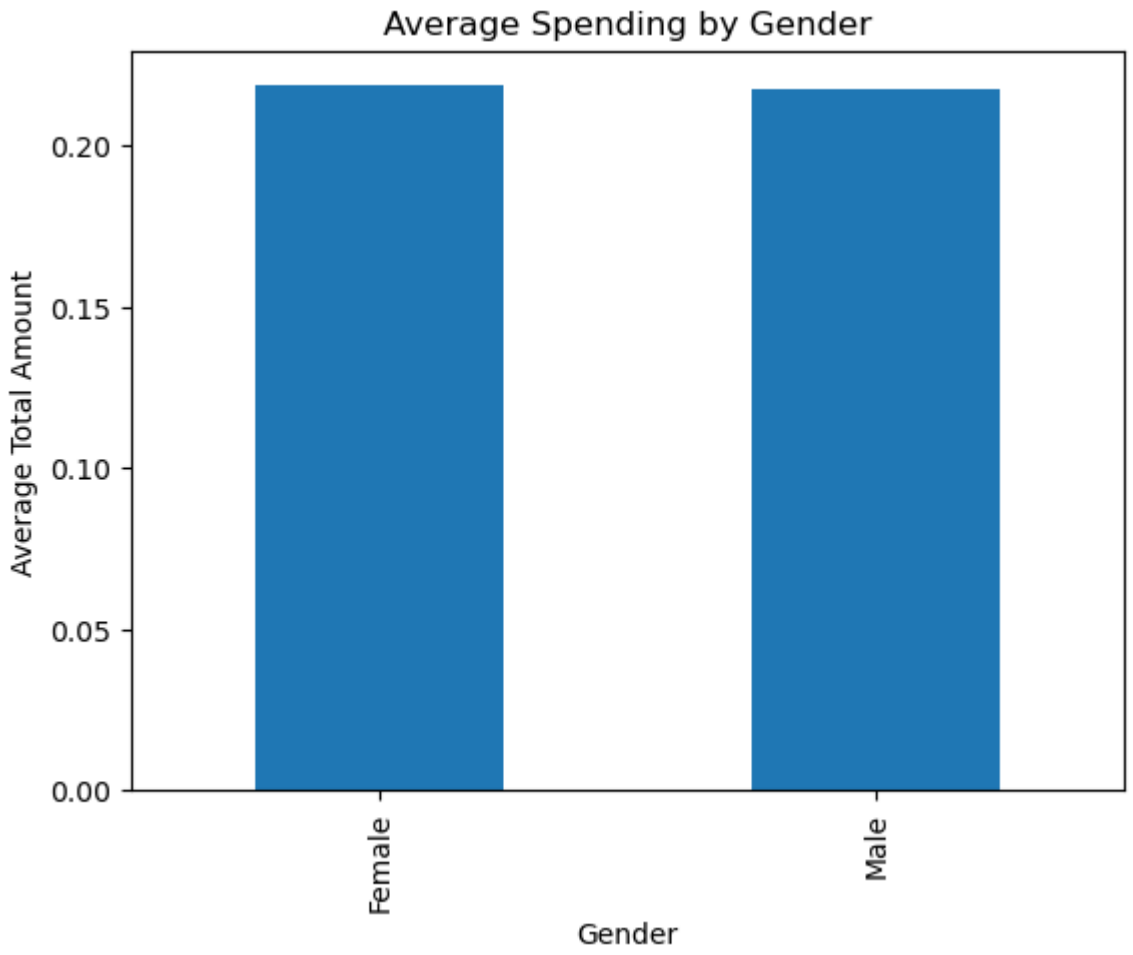
Aggregate Analysis (Average Spending by Age and Gender)

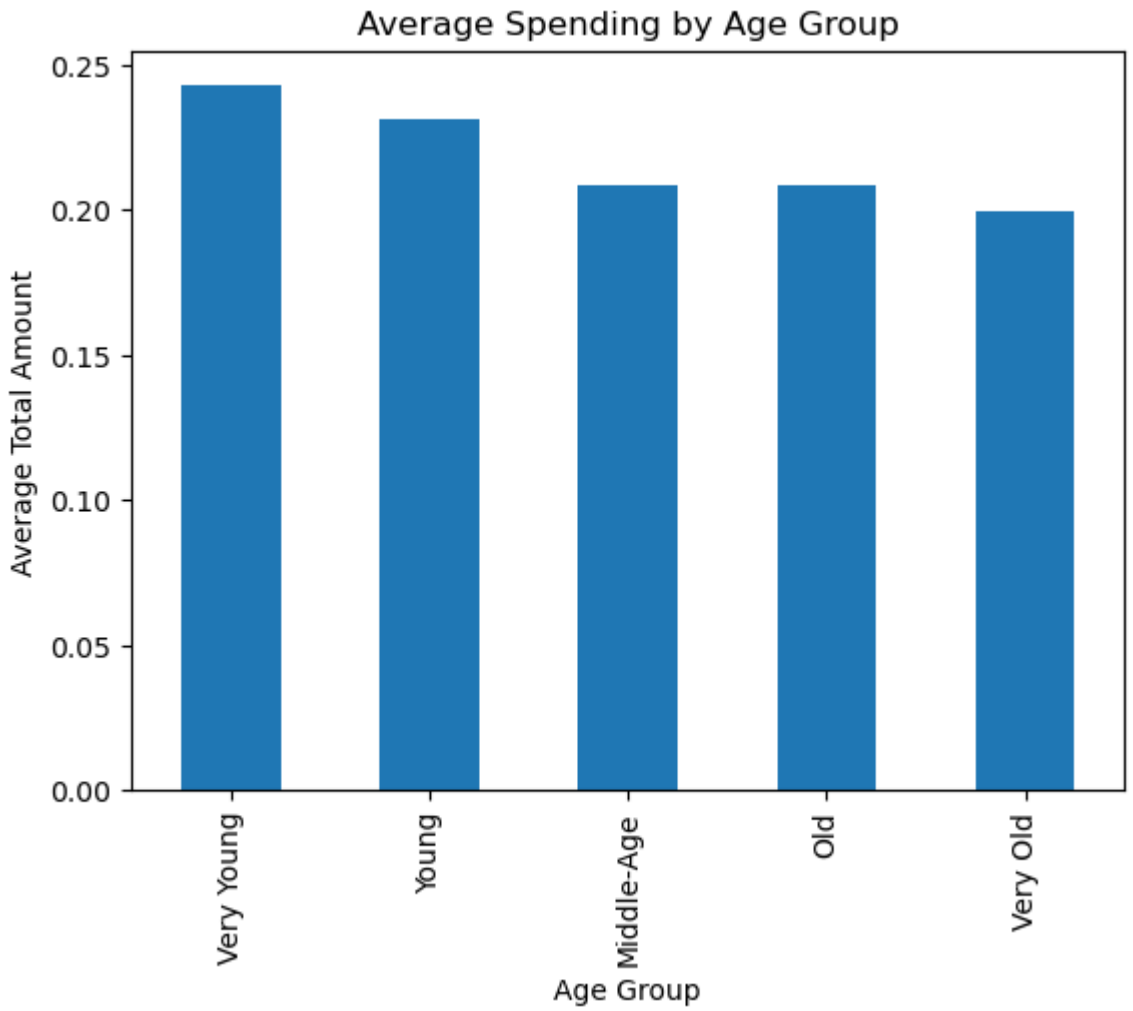
As we can see in the plots below, gender does not seem to play a significant role in influencing spending behaviour. This feature is, therefore, not important for classification of high spenders.

On the other hand, age may have a moderate impact on spending. We can see that 'Very Young' and 'Young' customers tends to spend more, on average, as compared to older customers. We can also see that spending decreases as age increase.

In [63]:
```
#Defining age groups
normalized_dataset['age_group'] = pd.cut(normalized_dataset['age'], bins=5, labe

#Aggregate total_amount by gender
gender_avg = normalized_dataset.groupby('gender')['total_amount'].mean()
print("Average Spending by Gender:")
print(gender_avg)

#Aggregate total_amount by age group
age_group_avg = normalized_dataset.groupby('age_group')['total_amount'].mean()
print("\nAverage Spending by Age Group:")
print(age_group_avg)

#Creating a bar plot for gender averages
gender_avg.plot(kind='bar', title="Average Spending by Gender", ylabel="Average
plt.show()

#Creating a bar plot for age group averages
age_group_avg.plot(kind='bar', title="Average Spending by Age Group", ylabel="Av
plt.show()
```

```
Average Spending by Gender:
gender
Female    0.218506
Male      0.217296
Name: total_amount, dtype: float64

Average Spending by Age Group:
age_group
Very Young    0.242671
Young         0.231016
Middle-Age    0.208561
Old           0.208642
Very Old      0.199670
Name: total_amount, dtype: float64
C:\Users\TazeenQ\AppData\Local\Temp\ipykernel_25344\2798269250.py:10: FutureWarni
ng: The default of observed=False is deprecated and will be changed to True in a
future version of pandas. Pass observed=False to retain current behavior or obser
ved=True to adopt the future default and silence this warning.
  age_group_avg = normalized_dataset.groupby('age_group')['total_amount'].mean()
```





Aggregate Analysis (Average Spending by Product Category and Age Group)

As we can see in the plot below, Very young and Young customers tend to spend more in each category. Again, spending decreases consistently as age increases, especially in Beauty category. Electronics spending shows higher average values overall, specially for very young and very old customers.

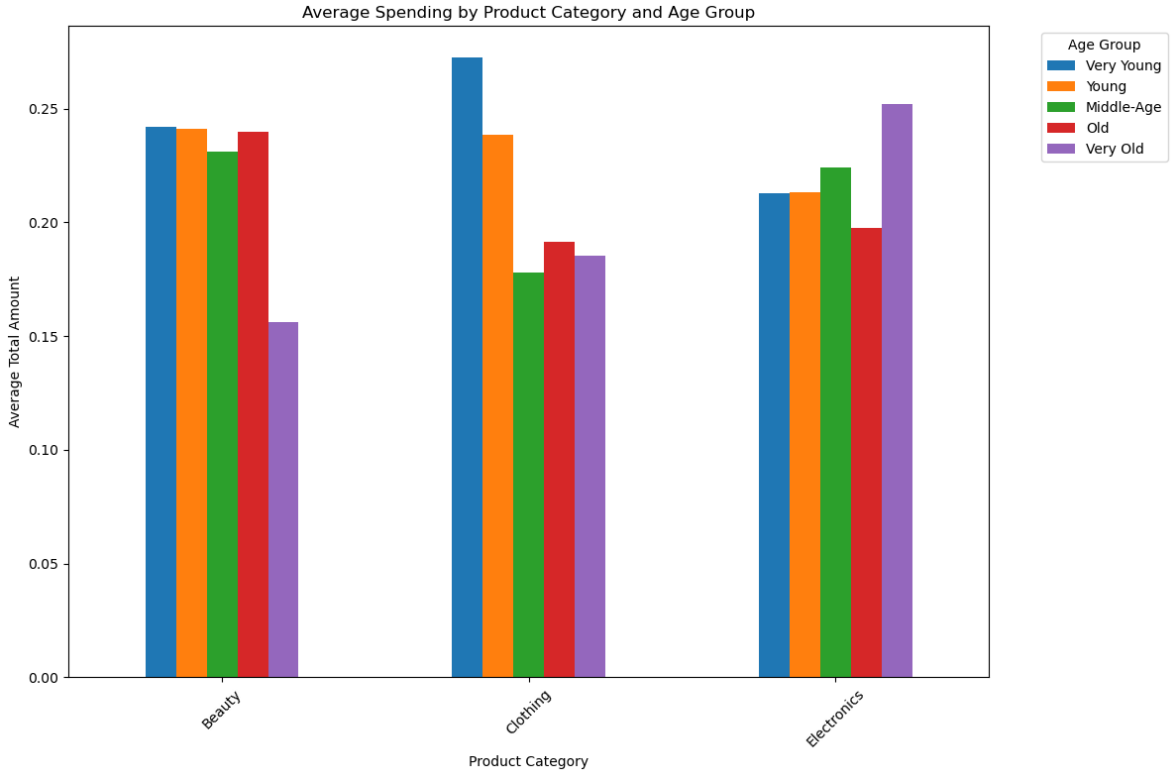
Younger customers also spend more on Electronics and Beauty, while older customers show a balanced but lower spending pattern across all three categories.

We can deduce from the plot that marketing efforts can target younger age groups for Electronics and Beauty products to maximize spending.

In [64]:
```
#Aggregate total amount by product category and age group
category_age_group_avg = normalized_dataset.groupby(['product_category', 'age_gr

#Creating a bar chart
plt.figure(figsize=(10, 6))
category_age_group_avg.plot(kind='bar', figsize=(12, 8))
plt.title("Average Spending by Product Category and Age Group")
plt.ylabel("Average Total Amount")
plt.xlabel("Product Category")
plt.xticks(rotation=45)
```

```
plt.legend(title="Age Group", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

```
C:\Users\TazeenQ\AppData\Local\Temp\ipykernel_25344\1652255262.py:2: FutureWarnin
g: The default of observed=False is deprecated and will be changed to True in a f
uture version of pandas. Pass observed=False to retain current behavior or observ
ed=True to adopt the future default and silence this warning.
  category_age_group_avg = normalized_dataset.groupby(['product_category', 'age_g
roup'])['total_amount'].mean().unstack()
<Figure size 1000x600 with 0 Axes>
```



Correlation Analysis of Numerical Features

Positive values (close to 1) indicate a strong positive correlation between variables and negative values (close to -1) indicate a strong negative correlation.

As we can see from the plot below, there is a strong positive correlation between price_per_unit and total_amount. This is natural, and self-explanatory.

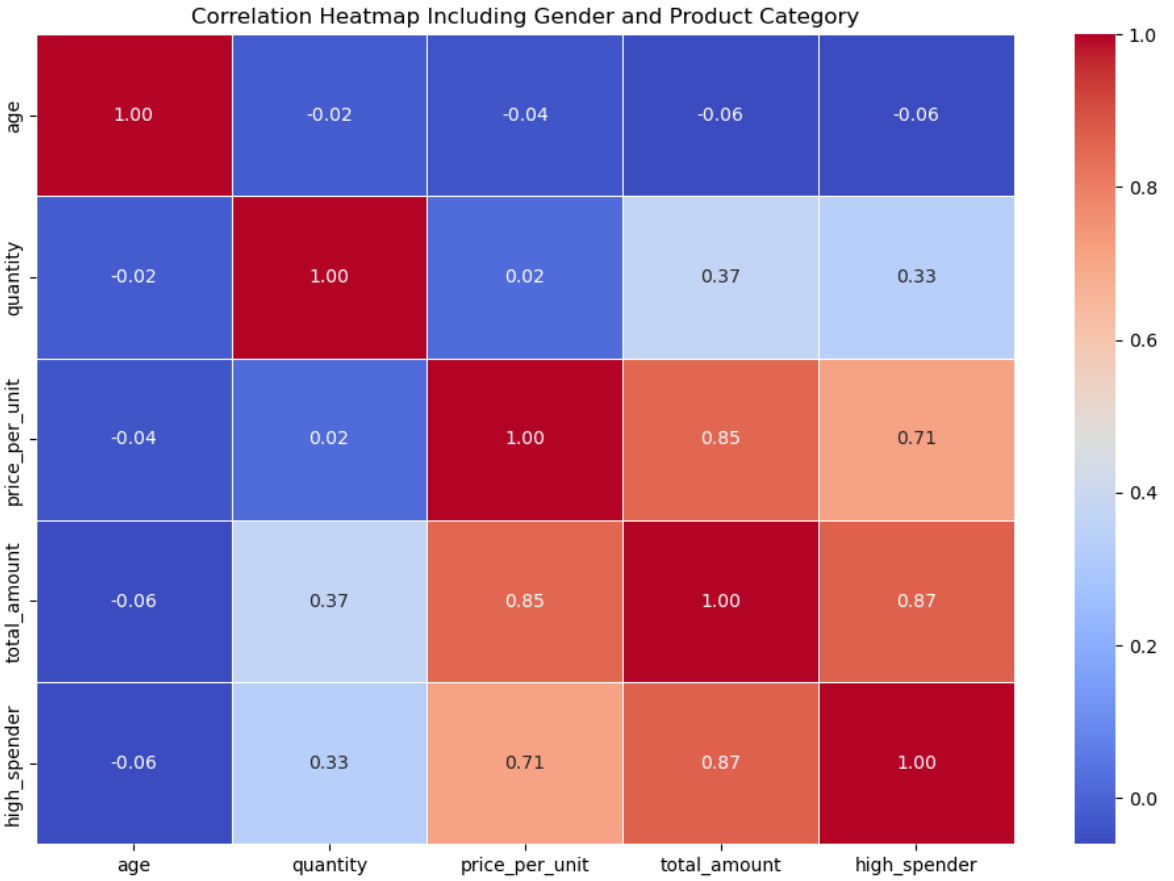
There is a moderately positive correlation between quantity and total_amount. Again, higher quantity bough would mean a higher total_amount.

There is weak or no correlation between other variables. This means that any of these variables is not a strong stand alone predictor for spending behaviour and customers are buying high quantities regardless of product's price per unit.

This analysis was inconclusive, so I will rely on the other types of exploratory analyses above.

In [65]:
```
#Selecting only numeric columns (not date)
```

```
numeric_columns = normalized_dataset.select_dtypes(include=['number'])

#Correlation matrix
correlation_matrix = numeric_columns.corr()

#Creating a correlation matrix as a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidt
plt.title("Correlation Heatmap Including Gender and Product Category")
plt.show()
```



Phase 3: Applying Statistical Model (Classification - K-neared Neighbor)

In Phase 3, I've run a classification model (K-nearest Neighbor) to see if we can determine whether a customer will be a high spender based on their demographics and purchase attributes.

Explanation and Interpretation of the Visualizations

Confusion Matrix (see below):

This matrix shows how well the KNN model classified the test data. Y-axis shows the actual labels, and X-axis shows the predicted labels:

- True Negatives (138): These are customers who are low spenders that were correctly classified.

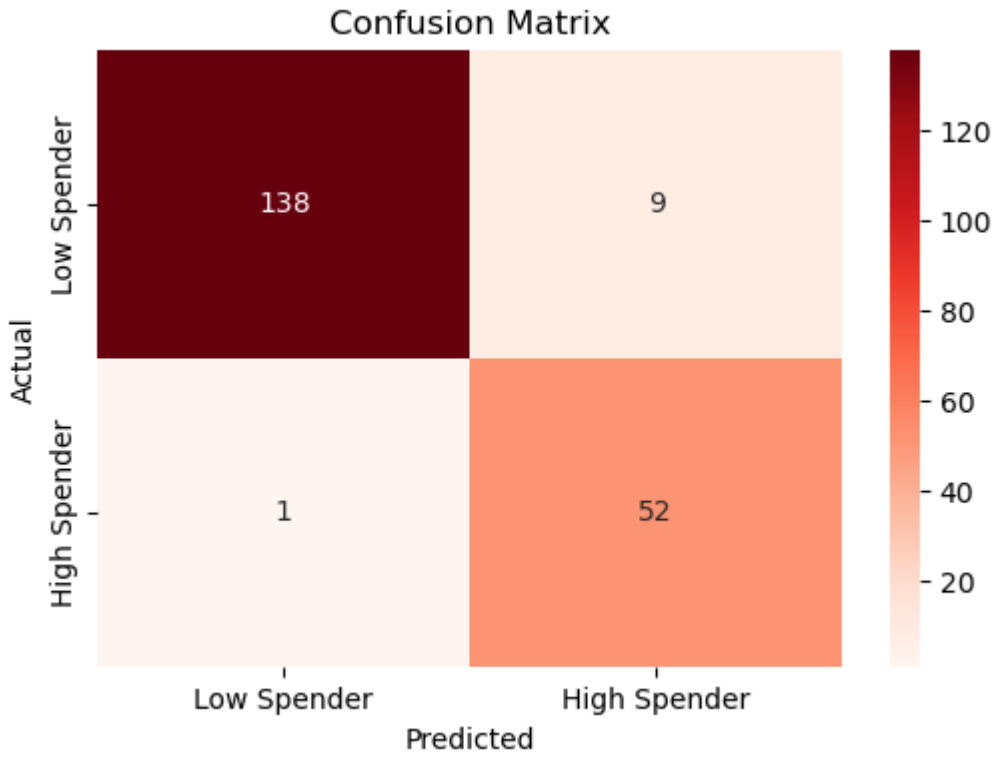
- False Positives (9): These are customers who are low spenders who were incorrectly classified as high spenders.
- False Negatives (1): These are customers who are high spenders who were incorrectly classified as low spenders.
- True Positives (52): These are customers who are high spenders that were correctly classified.

We can deduce that:

- The model performs well in identifying both High Spenders and Low Spenders.
- The small number of false negatives (1) indicates the model rarely misses identifying a High Spender.
- The false positives (9) indicate some overestimation of High Spenders.

In [80]:
```
#Loading the dataset
file_path = "C:\\Users\\TazeenQ\\team27_project\\data\\processed\\processed_reta
data = pd.read_csv(file_path)

#Defining the threshold for high spenders (top 25% of total_amount)
threshold = data['total_amount'].quantile(0.75)

#Creating a target variable: 1 for high spenders, 0 for low spenders
data['high_spender'] = (data['total_amount'] >= threshold).astype(int)

#Encoding categorical variables gender and product category
data_encoded = pd.get_dummies(data, columns=['gender', 'product_category'], drop

#Assigning features and target
X = data_encoded.drop(columns=['date', 'total_amount', 'high_spender'])
y = data_encoded['high_spender']

#Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

#Initializing the KNN model and training it
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

#Performing predictions
y_pred = knn.predict(X_test)

#Evaluating metrics
report = classification_report(y_test, y_pred, target_names=["Low Spender", "Hig
conf_matrix = confusion_matrix(y_test, y_pred)

#Printing the classification report
print("Classification Report:\n", classification_report(y_test, y_pred, target_n

#Plotting a confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Reds', xticklabels=["Low Spe
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

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

 Low Spender       0.99      0.94      0.97       147
High Spender       0.85      0.98      0.91        53

    accuracy                           0.95       200
   macro avg       0.92      0.96      0.94       200
weighted avg       0.96      0.95      0.95       200
```



Performance Metrics Bar Chart (see below):

This chart compares precision, recall, and F1-score for both classes (low and high spenders):

1. Precision measures the proportion of correctly predicted positive cases.
    - High Spenders precision is 85%, which means that 85% of predicted High Spenders are accurate.
    - Low Spenders precision is 99%, indicating very few misclassifications.
2. Recall measures the proportion of actual positives correctly identified.
    - High Spenders recall is 98%, indicating the model identifies most high spenders.
    - Low Spenders recall is 94%, showing slightly fewer correct classifications.
3. F1-Score is the harmonic mean of precision and recall.
    - High Spenders F1-score is 91%, indicating balanced performance.
    - Low Spenders F1-score is 97%, confirming high accuracy.

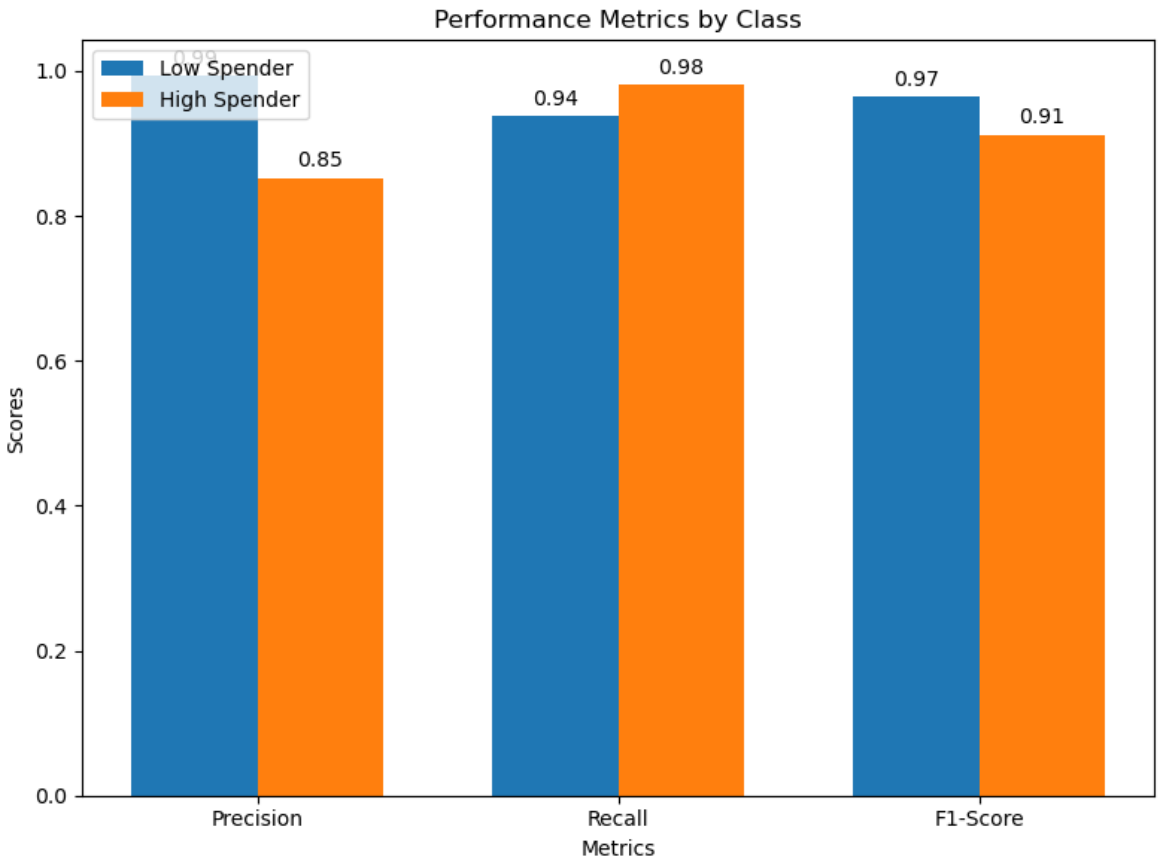
We can deduce that:

- The model performed really well in identifying High Spenders with very high recall (98%).
- Precision for High Spenders is slightly lower (85%), meaning a few false positives do

exist.

- Overall performance of the model is strong, with minimal trade-offs between precision and recall.

In [82]:
```
#Simplifiying bar chart for metrics
not_high_spender_scores = [report["Low Spender"]["precision"], report["Low Spend
high_spender_scores = [report["High Spender"]["precision"], report["High Spender

metrics = ["Precision", "Recall", "F1-Score"]
x = np.arange(len(metrics))  # the label locations
width = 0.35  # the width of the bars

#Creating the bar chart
fig, ax = plt.subplots(figsize=(8, 6))
bars1 = ax.bar(x - width/2, not_high_spender_scores, width, label="Low Spender")
bars2 = ax.bar(x + width/2, high_spender_scores, width, label="High Spender")

#Adding labels, title, and legend
ax.set_xlabel("Metrics")
ax.set_ylabel("Scores")
ax.set_title("Performance Metrics by Class")
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()

#Marking the bars with the score values
for bars in [bars1, bars2]:
    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height:.2f}', xy=(bar.get_x() + bar.get_width() / 2, heig
                    xytext=(0, 3), textcoords="offset points", ha='center', va='

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



Answering the Research Question

Our research question was:

Can we classify whether a customer will be a high spender based on their demographics (age, gender) and purchase attributes (product category, quantity)?

After running the KNN Model on our dataset, we can conclusively say that yes, the KNN model can effectively classify high spenders based on the purchase attributes.

We can also confidently say that:

- The model has an overall accuracy of 95%, indicating reliable predictions.
- It identifies high spenders with 98% recall, meaning nearly all high spenders are correctly classified.
- The slightly lower precision (85%) for high spenders suggests a small proportion of over-predictions, but this is balanced by the very high recall and F1-score.

Therefore, customer demographics (age, gender) and purchase attributes (product category, quantity) are strong predictors for classifying high spenders.