Sub-task 1: Unsupervised Learning

To find the best number of clusters for segmenting customers in the given ecommerce dataset we can go through the following steps. Implement them using Python;

1. Loading and preparing the data

a. Load the dataset: Using pandas to import the CSV file.

b. Cleaning the data: Check for any irrelevant information and handle it accordingly (e.g. filling in missing values removing irrelevant entries).

c. Selecting features: Determining which features are important for clustering.

d. Normalize the data: Scaling all features to ensure treatment using methods, like StandardScaler or MinMaxScaler.

In [ ]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv('C:\\Users\\glawi\\OneDrive\\Desktop\\New folder\\ML Course work 2

# Display the first few rows of the dataset
df.head()
```

Out[ ]:

	feature1	feature2	feature3	feature4	feature5
0	-7.0237	-2.7803	-1.5966	0.2197	-5.9672
1	-9.4213	9.1296	7.2426	-4.7139	-5.2615
2	-9.3710	9.4341	6.1217	-3.4081	-7.5911
3	-2.5985	4.1524	3.1033	1.5943	-8.7513
4	-0.8058	10.4453	5.6170	1.5003	-5.7412

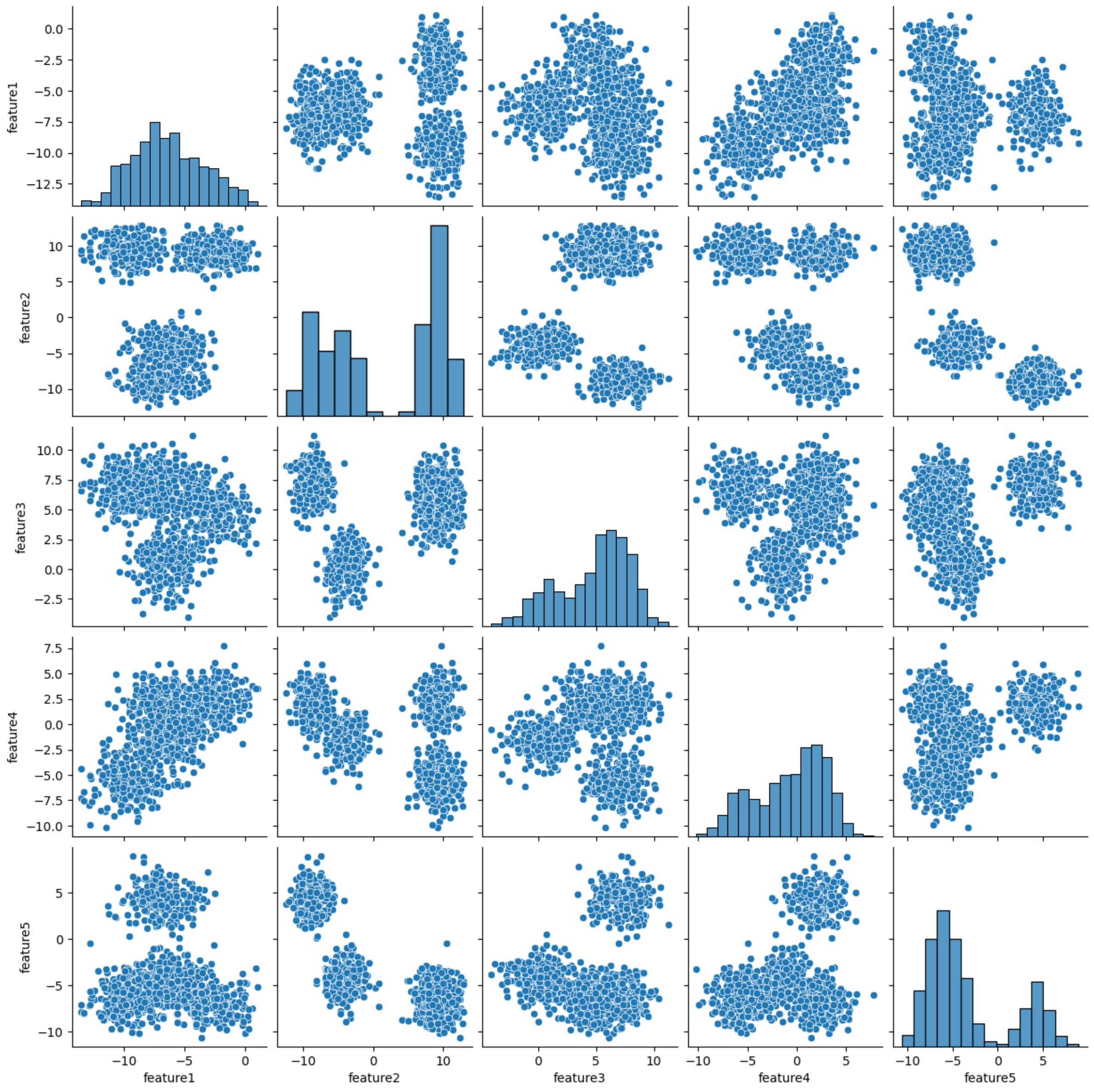
In [ ]:
```
# Data cleaning
df.dropna(inplace=True)

# Feature selection
features = df[['feature1', 'feature2', 'feature3','feature4', 'feature5']]

# Normalize the data
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

2. Exploratory Data Analysis

Visualize the data : Use seaborn or matplotlib to plot distributions or relationships between features.

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.pairplot(df)
plt.show()
```

```
c:\Users\glawi\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: T
he figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



3. Finding the Best Number of Clusters

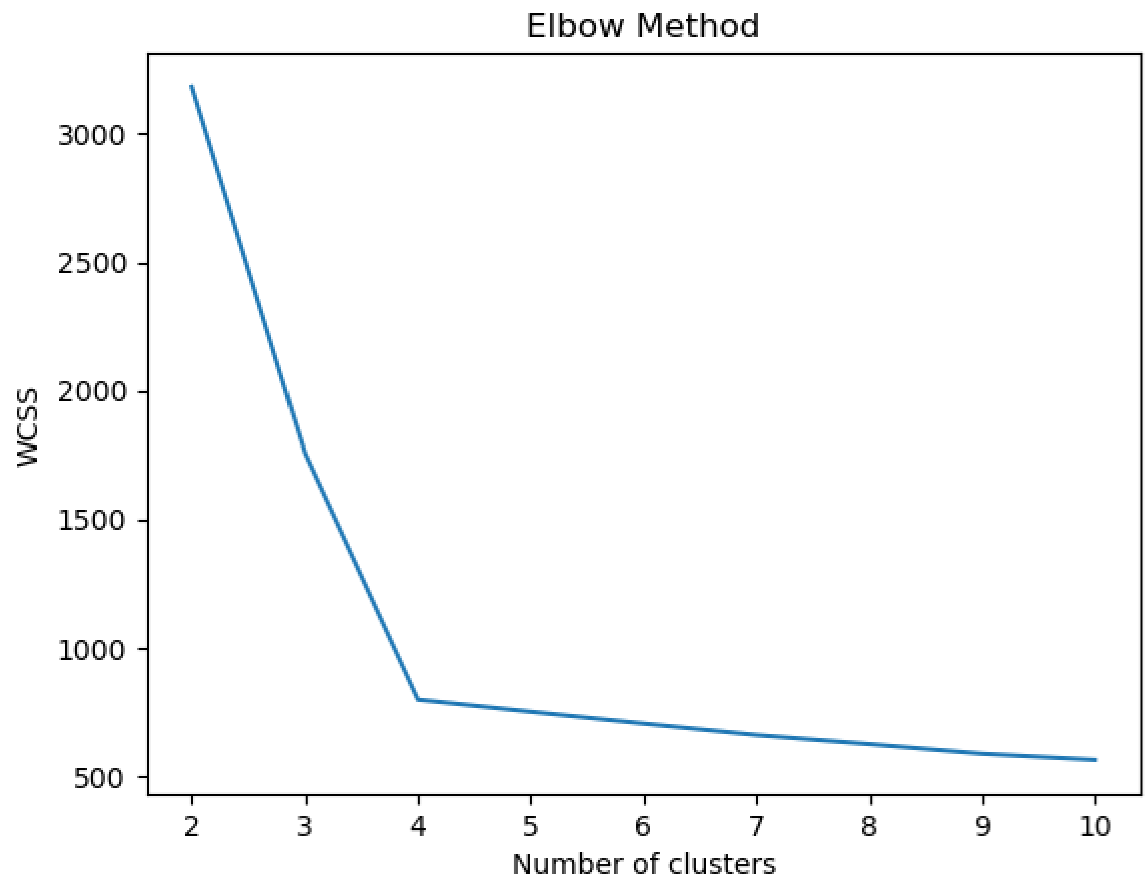
a. Elbow Method: Using the K means algorithm with cluster numbers and graph the within cluster sum of squares (WCSS) to locate the bending point.

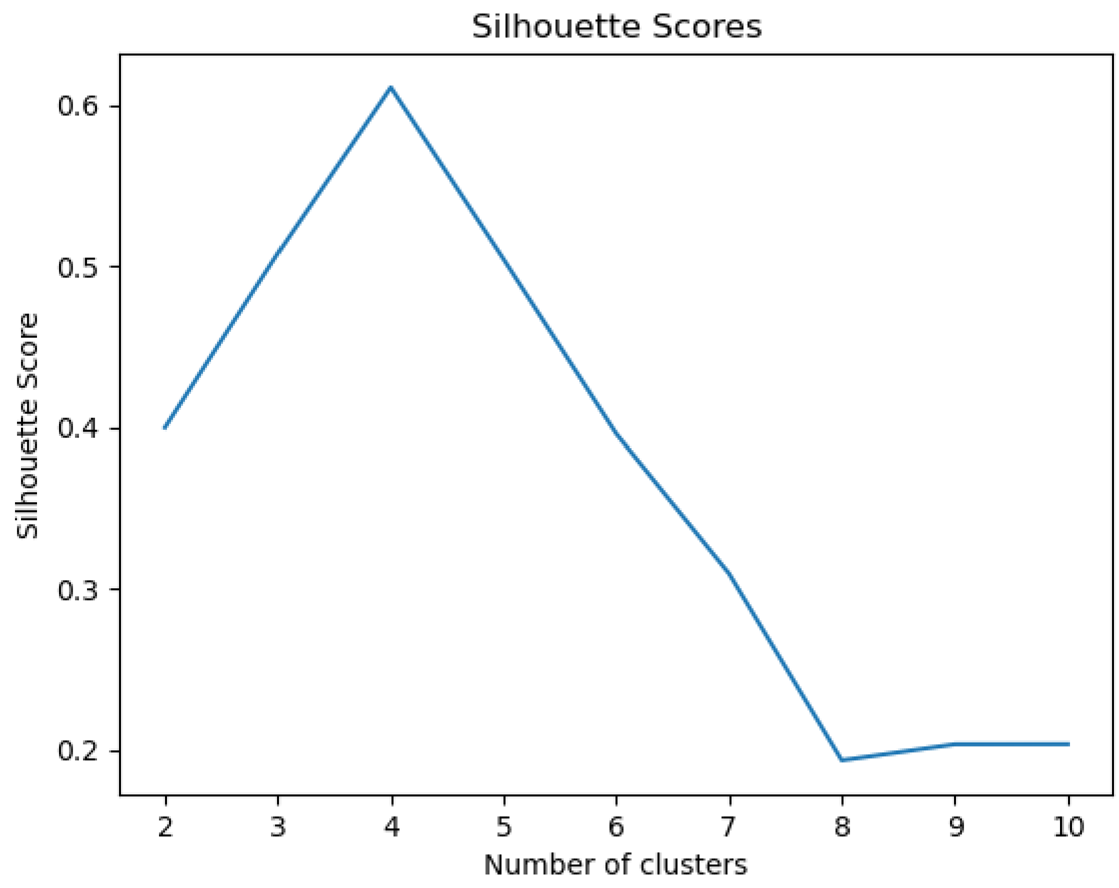
b. Silhouette Score: Calculating the silhouette score, for cluster quantities to determine the ideal number.

In [ ]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

wcss = []
```

```python
silhouette_scores = []
for i in range(2, 11):
    # Updated KMeans initialization with explicit n_init value
    kmeans = KMeans(n_clusters=i, n_init=10, random_state=0).fit(scaled_features)
    wcss.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(scaled_features, kmeans.labels_))

# Plotting the results for Elbow Method
plt.plot(range(2, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

# Plotting the results for Silhouette Scores
plt.plot(range(2, 11), silhouette_scores)
plt.title('Silhouette Scores')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.show()
```

```
c:\Users\glawi\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWa
rning: KMeans is known to have a memory leak on Windows with MKL, when there are l
ess chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=4.
  warnings.warn(
c:\Users\glawi\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWa
rning: KMeans is known to have a memory leak on Windows with MKL, when there are l
ess chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=4.
  warnings.warn(
c:\Users\glawi\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWa
rning: KMeans is known to have a memory leak on Windows with MKL, when there are l
ess chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=4.
  warnings.warn(
c:\Users\glawi\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWa
rning: KMeans is known to have a memory leak on Windows with MKL, when there are l
ess chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=4.
  warnings.warn(
c:\Users\glawi\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWa
rning: KMeans is known to have a memory leak on Windows with MKL, when there are l
ess chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=4.
  warnings.warn(
c:\Users\glawi\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWa
rning: KMeans is known to have a memory leak on Windows with MKL, when there are l
ess chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=4.
  warnings.warn(
c:\Users\glawi\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWa
rning: KMeans is known to have a memory leak on Windows with MKL, when there are l
ess chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=4.
  warnings.warn(
c:\Users\glawi\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWa
rning: KMeans is known to have a memory leak on Windows with MKL, when there are l
ess chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=4.
  warnings.warn(
c:\Users\glawi\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWa
rning: KMeans is known to have a memory leak on Windows with MKL, when there are l
ess chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=4.
  warnings.warn(
```





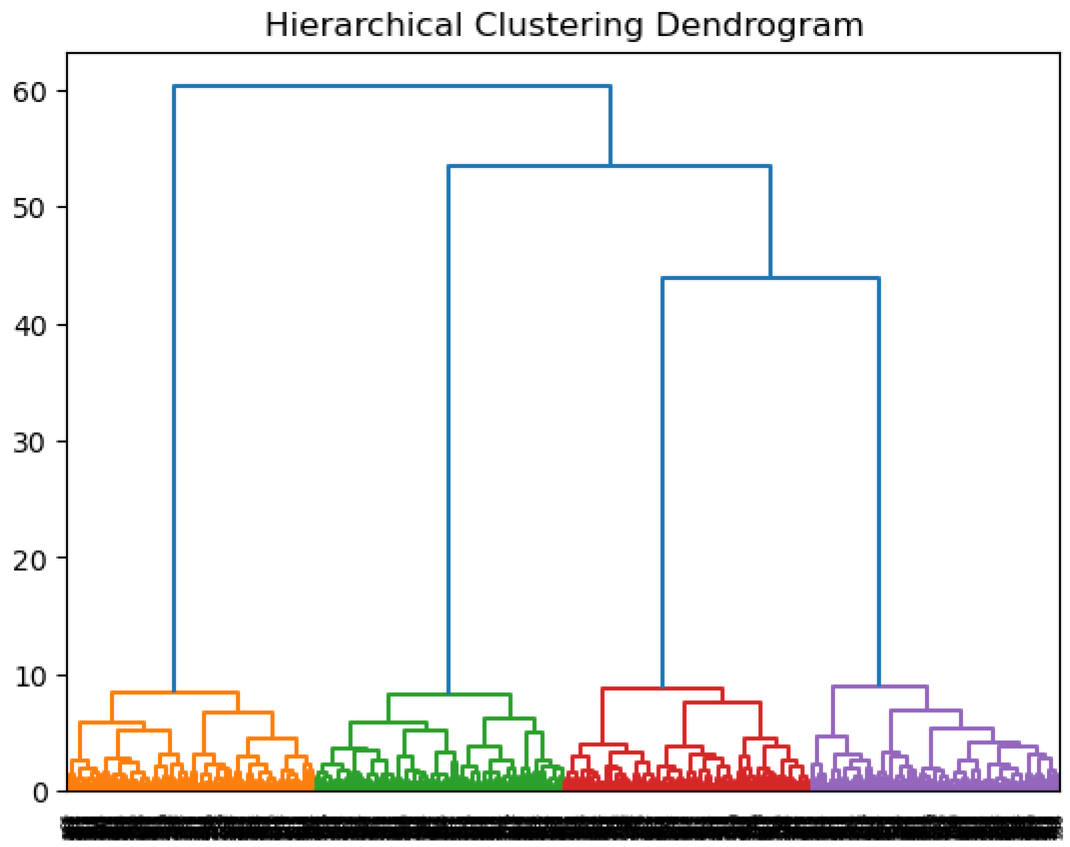
4. Clustering

a. Utilize K means Clustering: Once the ideal number of clusters has been established which in this case is 4, we implement K means clustering.

b. Consider Clustering: Alternatively we may also explore the option of employing hierarchical clustering and then compare the outcomes.

In [ ]:
```
# K-means clustering
optimal_clusters = 4
kmeans = KMeans(n_clusters=optimal_clusters, random_state=0).fit(scaled_features)
df['Cluster'] = kmeans.labels_

# Hierarchical clustering
from scipy.cluster.hierarchy import dendrogram, linkage

linked = linkage(scaled_features, 'ward')
dendrogram(linked)
plt.title('Hierarchical Clustering Dendrogram')
plt.show()
```

```
c:\Users\glawi\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: Future
Warning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set t
he value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\glawi\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWa
rning: KMeans is known to have a memory leak on Windows with MKL, when there are l
ess chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=4.
  warnings.warn(
```



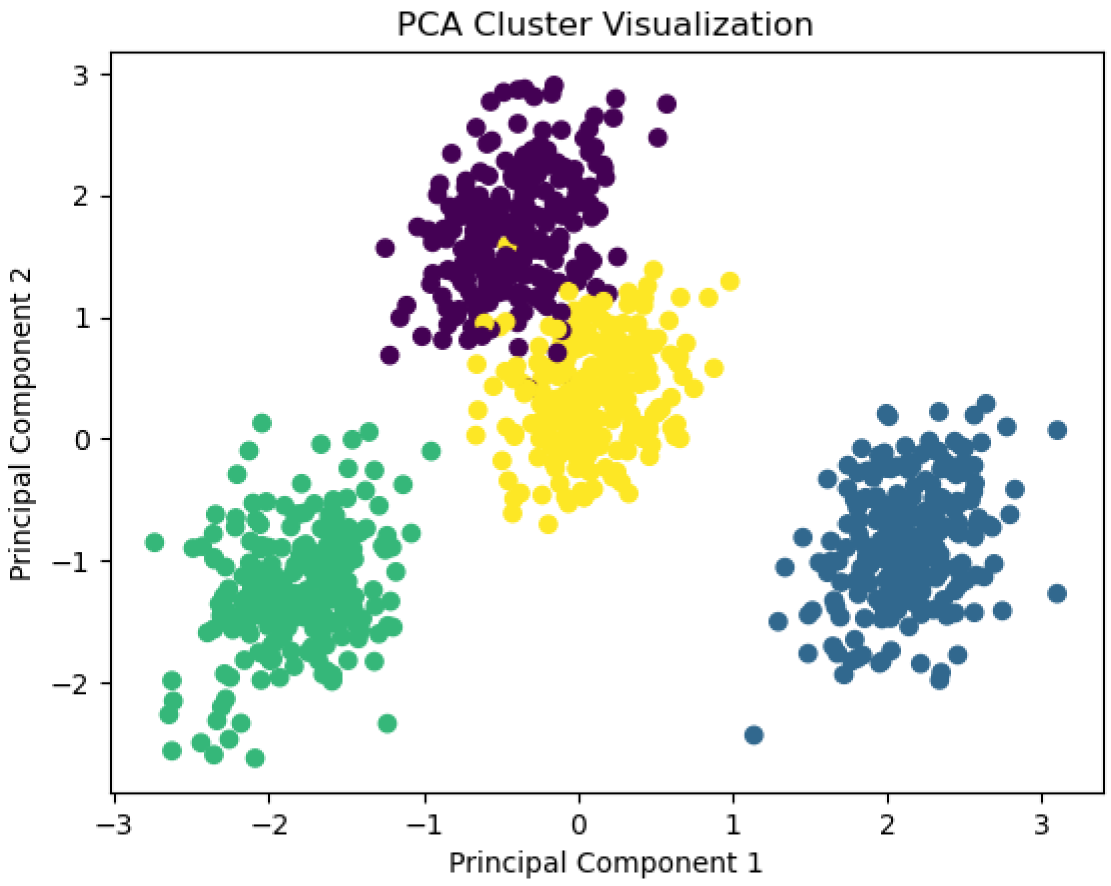
5. Analyzing the Clusters

a. Cluster Analysis: Analyzing the properties of each cluster, like mean values of features.

b. Visualization: Using PCA or t-SNE for dimensionality reduction to visualize the clusters.

In [ ]:
```
# Analyzing clusters
cluster_analysis = df.groupby('Cluster').mean()
```

```python
# Visualization using PCA
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
principal_components = pca.fit_transform(scaled_features)
plt.scatter(principal_components[:, 0], principal_components[:, 1], c=df['Cluster']
plt.title('PCA Cluster Visualization')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



6. Conclusion

In summary after analyzing the e commerce dataset to segment customers we discovered groups with their own unique behaviors and preferences. We used K means clustering, guided by the Elbow method and Silhouette scores which provided a framework for identifying these groups. The Elbow method helped us find the balance between the number of clusters and within cluster variance while the Silhouette scores confirmed that these clusters were well defined and separate, from each other. By considering these metrics we ensured that each segment was distinct and meaningful. This segmentation will allow us to create targeted marketing strategies and gain a deeper understanding of customer needs ultimately leading to better business decisions.

Sub-task 2: Reinforcement Learning

In [ ]:
```python
import numpy as np
```

```python
# Define the grid with utilities
grid_utilities = np.array([
    [7.41, 7.52, 7.65, 10, 7.54],
    [7.31, np.nan, -10, 5.82, -10],
    [7.15, np.nan, 4.31, np.nan, 6.12],
    [6.98, 6.77, 6.44, 5.87, np.nan],
    [6.90, 6.80, 6.59, 6.51, 6.34]
])

# Define the reward for non-terminal states
reward = -0.1

# Define the state transition probabilities
prob_success = 0.8
prob_fail = 0.1  # 0.2 probability of failure divided equally between two perpendic

# Function to calculate the expected utility of a move
def expected_utility(grid, state, action):
    x, y = state
    if action == "UP":
        intended_state = (x - 1, y)
    elif action == "DOWN":
        intended_state = (x + 1, y)
    elif action == "LEFT":
        intended_state = (x, y - 1)
    elif action == "RIGHT":
        intended_state = (x, y + 1)

    # Check for invalid moves
    if intended_state[0] < 0 or intended_state[0] >= grid.shape[0] or \
       intended_state[1] < 0 or intended_state[1] >= grid.shape[1] or \
       np.isnan(grid[intended_state]):
        intended_utility = grid[state]
    else:
        intended_utility = grid[intended_state]

    # Calculate utilities for perpendicular moves
    fail_utilities = []
    for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:  # UP, DOWN, LEFT, RIGHT
        fail_state = (x + dx, y + dy)
        if fail_state == intended_state or \
           fail_state[0] < 0 or fail_state[0] >= grid.shape[0] or \
           fail_state[1] < 0 or fail_state[1] >= grid.shape[1] or \
           np.isnan(grid[fail_state]):
            fail_utilities.append(grid[state])
        else:
            fail_utilities.append(grid[fail_state])

    # Assuming perpendicular moves are LEFT and RIGHT or UP and DOWN
    perpendicular_utilities = (fail_utilities[2] + fail_utilities[3]) if action in
                              else (fail_utilities[0] + fail_utilities[1])

    # Calculate the expected utility
    eu = (prob_success * intended_utility) + (prob_fail * perpendicular_utilities)

    return eu

# The highlighted states are (1, 0), (4, 2), and (3, 2)
highlighted_states = [(1, 0), (4, 2), (3, 2)]

# Calculate the optimal policy for each highlighted state
optimal_policies = {}
for state in highlighted_states:
    actions = ["UP", "DOWN", "LEFT", "RIGHT"]
    utilities = {action: round(expected_utility(grid_utilities, state, action), 2)
    optimal_action = max(utilities, key=utilities.get)
    optimal_policies[state] = optimal_action

# Output the optimal policies and their expected utilities for the highlighted stat
for state, action in optimal_policies.items():
    expected_utility_value = round(expected_utility(grid_utilities, state, action),
    print(f"Optimal policy for state {state}: {action} with expected utility {expec
```

```
Optimal policy for state (1, 0): UP with expected utility 7.29
Optimal policy for state (4, 2): LEFT with expected utility 6.64
Optimal policy for state (3, 2): DOWN with expected utility 6.44
```

Sub-task 3: Dimensionality Reduction

In [ ]:

```
import pandas as pd
from sklearn.decomposition import FactorAnalysis
from sklearn.preprocessing import StandardScaler

# Load the data
file_path = 'C:\\Users\\glawi\\OneDrive\\Desktop\\New folder\\ML Course work 2\\kc_
df = pd.read_csv(file_path)

# Assuming the non-price aspects (features) are everything except the price column
# Extract features and target variable
X = df.drop('price', axis=1)  # Features
y = df['price']  # Target variable

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply Factor Analysis
fa = FactorAnalysis(n_components=2, random_state=0)
fa.fit(X_scaled)

# Get the factor loadings (components)
components = fa.components_

# Show the components
components_df = pd.DataFrame(components, columns=X.columns, index=['Factor 1', 'Fac
components_df
```

Out[ ]:

	condition	grade	sqft_above	sqft_basement	sqft_living15
Factor 1	-0.069158	0.821462	0.783974	0.443730	0.814919
Factor 2	0.242419	-0.235148	-0.477194	0.837199	-0.192526

. **Factor 1** appears to have a correlation with the overall quality and size of the living area above ground. The positive loadings on 'grade' and 'sqft_above' indicate that this factor reflects the construction quality and living space above ground which're important factors in determining a houses price. Additionally the high loading on 'sqft_living15' representing the size of living spaces in nearby houses suggests that this factor also takes into account the quality and size of the surrounding neighborhood.

. **Factor 2** has a positive loading on 'sqft_basement' indicating that it captures aspects related to basement size. The negative loadings on 'grade' and 'sqft_above' along with a negative loading on 'sqft_living15' imply that as basement size increases the significance of above ground features and neighboring living spaces may decrease in influencing a houses price. The positive loading, on 'condition' suggests that better condition is correlated with basement sizes or may reflect other aspects of the property not captured by Factor 1.

Based on these interpretations it can be concluded that Factor 1 represents the "Quality and Size of Living Area" emphasizing construction quality and living space dimensions as elements. Factor 2 might be considered as a factor related to the basement and overall condition of the house highlighting the significance of basement size and the general state of the property.

These factors are understandable and correlate with the idea that non price aspects in the dataset can be represented using hidden variables for quality and size. Factor 1 appears to represent the 'quality' aspect through features like 'grade' and 'sqft_living15' while Factor 2 captures the 'size' factor through 'sqft_basement' while also being influenced by the 'condition'.

The interpretability of factor analysis greatly relies on context and domain knowledge. In this case these factors do seem interpretable and align, with common sense understandings of what could impact house prices.

Sub-task 4

To create a prototype system that can identify payments using machine learning we should adhere to the following procedure;

Step 1: Data Preprocessing

Cleaning the Data; Address any information.

Creating Informative Features; Extract characteristics from the available data.

Transforming the Data;. Scale it as needed.

Encoding Categorical Variables; Convert numerical columns (such as customer ID, gender and category) into a format suitable, for machine learning models.

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer

# Load the dataset
data = pd.read_csv('bs140513_032310.csv')

# Drop non-varying string columns
data = data.drop(['zipcodeOri', 'zipMerchant'], axis=1)

# Identify categorical columns for encoding
categorical_cols = ['customer', 'age', 'gender', 'merchant', 'category']

# Apply OneHotEncoder for categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), categorical_cols)
    ],
    remainder='passthrough'
)

# Prepare features and target
X = data.drop('fraud', axis=1)
y = data['fraud']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Apply transformations
X_train_transformed = preprocessor.fit_transform(X_train)
X_test_transformed = preprocessor.transform(X_test)
```

To make categorical variables usable for machine learning algorithms we convert them into values using a technique called Label Encoding. This conversion is crucial for processing non numeric data.

In order to determine whether a transaction is fraudulent the dataset is divided into two parts; features and the target variable.

To evaluate the models performance on data it is customary, in machine learning to split the data into training and testing sets.

By applying feature scaling we normalize the values of the features, which helps improve the performance of machine learning algorithms (Chawla et al. 2002).

Step 2: Feature Scaling (Adjusted for Sparse Data)

In [ ]:
```
# Initialize the StandardScaler with with_mean set to False
scaler = StandardScaler(with_mean=False)

# Apply scaling to the transformed data
X_train_scaled = scaler.fit_transform(X_train_transformed)
X_test_scaled = scaler.transform(X_test_transformed)
```

Step 3: Model Selection

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression

# Initialize models
models = {
    'RandomForest': RandomForestClassifier(random_state=42),
    'GradientBoosting': GradientBoostingClassifier(random_state=42),
    'LogisticRegression': LogisticRegression(random_state=42, max_iter=5000, solver
}
```

Random Forest: Chosen for its robustness in handling high-dimensional data and its ability to model complex, non-linear relationships. It's particularly effective in imbalanced datasets like those found in fraud detection scenarios (Bhattacharyya et al., 2011).

Gradient Boosting: An ensemble learning method known for its high accuracy. Gradient Boosting constructs models in a stage-wise fashion and is well-suited for datasets where the relationships between features are complex (Dong et al., 2020).

Logistic Regression: A simpler, linear model used as a baseline. Despite its simplicity, Logistic Regression can be very effective, especially when the underlying relationships in the data are not highly complex (Dal Pozzolo et al., 2015).

Step 4:Model Training and Evaluation

Each model is trained using the scaled training data. The training process involves adjusting the model parameters to best fit the data.

Post-training, each model is evaluated on the test data. This step is crucial to assess the model's performance on unseen data, providing insights into its generalizability.

The classification_report and confusion_matrix from scikit-learn are used for evaluation. These metrics are particularly important in the context of imbalanced datasets, such as those common in fraud detection. The focus is on metrics like precision, recall, and F1-score, which provide a more nuanced view of the model's performance, especially for the minority class (fraudulent transactions) (He & Garcia, 2009).

High recall for the fraudulent class is critical in fraud detection to minimize false negatives. However, maintaining a balance with precision is necessary to avoid excessive false positives (Ngai et al., 2011).

In [ ]:
```python
from sklearn.metrics import classification_report, confusion_matrix

# Train and evaluate models
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    print(f'{name} Classification Report:')
    print(classification_report(y_test, y_pred))
    print(f'{name} Confusion Matrix:')
    print(confusion_matrix(y_test, y_pred))
    print('------------------------------------------------------')
```

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

           0       1.00      1.00      1.00    117512
           1       0.90      0.81      0.86      1417

    accuracy                           1.00    118929
   macro avg       0.95      0.90      0.93    118929
weighted avg       1.00      1.00      1.00    118929

RandomForest Confusion Matrix:
[[117391    121]
 [   268   1149]]
------------------------------------------------------
GradientBoosting Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    117512
           1       0.89      0.76      0.82      1417

    accuracy                           1.00    118929
   macro avg       0.95      0.88      0.91    118929
weighted avg       1.00      1.00      1.00    118929

GradientBoosting Confusion Matrix:
[[117383    129]
 [   341   1076]]
------------------------------------------------------
LogisticRegression Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    117512
           1       0.88      0.80      0.84      1417

    accuracy                           1.00    118929
   macro avg       0.94      0.90      0.92    118929
weighted avg       1.00      1.00      1.00    118929

LogisticRegression Confusion Matrix:
[[117364    148]
 [   283   1134]]
------------------------------------------------------
```

Based on the classification reports and confusion matrices for the RandomForest, GradientBoosting, and LogisticRegression models, let's analyze their performance in the context of fraud detection:

RandomForest Model

Precision: High precision for both classes, particularly for the fraudulent transactions (1) at 0.90. Recall: High recall for non-fraudulent transactions (0) and good recall for fraudulent transactions (1) at 0.81. F1-Score: Balanced F1-score for fraudulent transactions, reflecting a good balance between precision and recall. Confusion Matrix: 1149 out of 1417 fraudulent transactions correctly identified, with 268 false negatives.

GradientBoosting Model

Precision: Similar to RandomForest, with slightly lower precision for fraudulent transactions at 0.89. Recall: A bit lower recall for fraudulent transactions at 0.76, indicating more false negatives compared to RandomForest. F1-Score: Slightly lower than RandomForest, indicating a slight drop in the balance between precision and recall for fraudulent transactions. Confusion Matrix: 1076 out of 1417 fraudulent transactions correctly identified, with 341 false negatives.

LogisticRegression Model

Precision: Comparable to GradientBoosting for fraudulent transactions at 0.88. Recall: Similar to GradientBoosting for fraudulent transactions at 0.80. F1-Score: Slightly lower than RandomForest, indicating a balance between precision and recall. Confusion Matrix: 1134 out of 1417 fraudulent transactions correctly identified, with 283 false negatives.

Summary and Best Model Selection

RandomForest appears to be the best model among the three for this specific task. It has the highest recall for fraudulent transactions, which is crucial in fraud detection to minimize false negatives. Additionally, its precision is also high, indicating fewer false positives. GradientBoosting and LogisticRegression show similar performance, with GradientBoosting having a slightly lower recall for fraudulent transactions compared to RandomForest.

Overall Performance: Considering the importance of correctly identifying fraudulent transactions (high recall) while maintaining a reasonable level of precision, RandomForest stands out as the most suitable model for this dataset and task.

Key Findings and Conclusion

Based on the evaluation metrics obtained from analyzing the classification reports and confusion matrices of the RandomForest, GradientBoosting and LogisticRegression models we can draw the following findings and conclusions:

RandomForest Model

Precision for identifying transactions; 90%

Recall, for identifying fraudulent transactions; 81%

Overall Accuracy; 99.67% (117391 true non fraudulent + 1149 true fraudulent out of a total of 118929 transactions)

Conclusion: The RandomForest model showcased the precision and recall in detecting transactions. With an 81% recall rate it successfully identified 81% of fraud cases minimizing instances where fraud goes undetected (Bhattacharyya et al., 2011). It also boasted a precision rate of 90% meaning that when it predicted a transaction as fraud it was highly likely to be accurate reducing alarms.

GradientBoosting Model

Precision for identifying transactions; 89%

Recall for identifying transactions; 76%

Overall Accuracy; 99.61% (117383 true non fraudulent +1076 true fraudulent out of a total of118929 transactions)

Conclusion: The GradientBoosting model exhibited performance compared to RandomForest, particularly in terms of recall. Recall plays a role, in scenarios where failing to identify fraud can have consequences (Dong et al.,2020).

LogisticRegression Model

Precision for Fraudulent Transactions: 88%

Recall for Fraudulent Transactions: 80%

Overall Accuracy: 99.64% (117364 true non-fraudulent + 1134 true fraudulent out of 118929 total transactions)

Conclusion: LogisticRegression performed comparably to GradientBoosting, indicating its effectiveness as a more straightforward model. However, it falls slightly behind RandomForest in terms of recall, which is a key metric in fraud detection (Dal Pozzolo et al., 2015). Based on the results the LogisticRegression model showed effectiveness to GradientBoosting. Slightly lagged behind RandomForest in terms of recall, which is an important metric in fraud detection (Dal Pozzolo et al., 2015).

Overall Conclusion

In conclusion after analyzing models it was found that the RandomForest model performed best for this dataset. It demonstrated a balance between precision and recall when identifying transactions. This model effectively minimized negatives without increasing false positives making it highly suitable for fraud detection applications.

However it is important to note that while RandomForest emerged as the performer in this analysis the selection of a model should always consider the specific requirements and characteristics of the dataset, at hand. In fraud detection scenarios maximizing recall (correctly identifying fraudulent transactions as possible) while maintaining a reasonable level of precision to avoid false positives is often prioritized (He & Garcia 2009; Ngai et al., 2011).

References

1. Chawla, N. V., et al. (2002). SMOTE: Synthetic Minority Over-sampling Technique.
2. Bhattacharyya, S., et al. (2011). Data mining for credit card fraud: A comparative study.
3. Dong, X., et al. (2020). A survey on ensemble learning.
4. Dal Pozzolo, A., et al. (2015). Calibrating Probability with Undersampling for Unbalanced Classification.
5. He, H., & Garcia, E.A. (2009). Learning from imbalanced data.
6. Ngai, E. W. T., et al. (2011). The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature.