Split, Predict, Conquer: Machine Learning in Action

In [14]:
```
# Load necessary libraries

import pandas as pd #to manipulate the dataset
from sklearn.model_selection import train_test_split, GridSearchCV #for splitting the dataset
from sklearn.preprocessing import StandardScaler #to scale.  the model
from sklearn.linear_model import LogisticRegression #to perform Logistic Regression model
from sklearn.neighbors import KNeighborsClassifier # to perform K-Nearest Neighbors model
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score # to evaluate classification models
import matplotlib.pyplot as plt #to create static plot
import seaborn as sns # to visualization
from sklearn.cluster import KMeans, AgglomerativeClustering # to perform unsupervised clustering
import numpy as np # for numeric operations
import scipy.cluster.hierarchy as sch # to create dendrograms
from sklearn.decomposition import PCA #to perform PCA
```

PART I

Classification task - Introduction

1. Describe the goal of this classification task. Identify and justify your choice of independent variables (IVs).

Exactly predicting the probability of returning a product based on significant elements of their purchase behaviour is the goal of this grouping of physical activity. The company can use this information to better manage reimbursements, manage inventory, and customise customer service tactics. Features connected to purchases and consumer characteristics that are likely to influence return behaviour are among the independent variables (IVs) selected for this task. Variables like order frequency, order value, days since the last order, and number of returns in the past may be among them. These features have been selected because they offer quantifiable indicators of consumer satisfaction and conduct.

2. Outline the machine learning models you will implement. Describe the preprocessing steps, evaluation metrics, and your rationale for selecting them.

Two classification models, K-Nearest Neighbours (KNN) and Logistic Regression, were utilised. The dataset was split into training and testing sets, any missing values were taken care of, StandardScaler was used to standardise numerical features, and GridSearchCV was used to adjust the hyperparameters. The confusion matrix, which imagines true/false positives and negatives, precision, recall, and F1-score, as well as accuracy, which measures overall correctness, is amongst the evaluation metrics used in assessing the model's performance. In the binary sorting tasks, logistic regression was selected due to its ease of use, interpretability, and effectiveness as well as its capacity to provide light on the worth of variables.

In [15]:
```
# Load dataset for classification task + pre-processing steps

df = pd.read_csv('Dataset_classification.csv')
df.head() # to view first 5 row
```

Out[15]:

	discount_percentage	product_price	delivery_delay_days	customer_satisfaction	returned	time_on_product_page	product_weight_
0	0.18	0.87	1	0.15	0	0.28	0.
1	0.66	2.65	1	1.83	1	0.40	0.
2	2.38	3.20	0	2.54	1	1.40	0.
3	2.61	3.04	2	2.67	1	0.80	0.
4	0.56	0.49	0	0.46	0	0.06	0.

In [16]:
```
#list of independenet variables

features = [
    'discount_percentage',
    'product_price',
    'delivery_delay_days',
    'customer_satisfaction',
    'time_on_product_page',
    'days_since_last_purchase',
    'number_of_items_in_order',
    'session_length_minutes'
]


# Select the features (X) and target variable (y)
```

```
X = df[features]
y = df['returned']

# Split the dataset into training and test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)



# Set up the StandardScaler to standardize features

sc = StandardScaler()

# fit model into trainning dataset
X_train_scaled = sc.fit_transform(X_train)

#fir model into test dataset
X_test_scaled = sc.transform(X_test)
```

In [17]:
```
# Model 1

# Logistic Regression model

logit = LogisticRegression(max_iter=1000)


# Create the grid of hyperparameters:
 # - 'C' is the inverse of regularity strength; higher regularity is indicated by a lower number.
 # -'solver' chooses the model improvement algorithm.
param_grid_log = {
    'C': [0.01, 0.1, 1, 10],
    'solver': ['liblinear', 'lbfgs']
}


# Setup GridSearchCV to figure out the ideal hyperparameter combination.
 # - scoring='accuracy' evaluates models according to accuracy # - cv=5 performs 5-fold cross-validation
grid_log = GridSearchCV(logit, param_grid_log, cv=5, scoring='accuracy')


#  for Fitting the grid search on the scaled training data
grid_log.fit(X_train_scaled, y_train)

print("Best Parameters for Logistic Regression:", grid_log.best_params_)
print("Best Cross-Validated Accuracy:", grid_log.best_score_)
```

```
Best Parameters for Logistic Regression: {'C': 1, 'solver': 'liblinear'}
Best Cross-Validated Accuracy: 0.9040000000000001
```

In [18]:
```
# Model 2

#the K-Nearest Neighbors classifier model

knn_clf = KNeighborsClassifier()


# Create the grid of hyperparameters:
 #-'n_neighbors': the number of closest neighbours to take into account
 # - 'weights': how to assign weight to the neighbours

param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance']
}
# Use GridSearchCV to find the optimal K-NN hyperparameter combination.
grid_knn = GridSearchCV(knn_clf, param_grid_knn, cv=5, scoring='accuracy')

# the grid search on the scaled training data
grid_knn.fit(X_train_scaled, y_train)

print("Best Parameters for K-NN:", grid_knn.best_params_)
print("Best Cross-Validated Accuracy:", grid_knn.best_score_)
```

```
Best Parameters for K-NN: {'n_neighbors': 9, 'weights': 'uniform'}
Best Cross-Validated Accuracy: 0.9013333333333333
```

In [19]:
```
# evaluate a trained model on the test set

def evaluate_model(model, X_test, y_test, model_name):
    y_pred = model.predict(X_test)
    report = classification_report(y_test, y_pred, output_dict=False)
    print(f"\n{model_name} Evaluation After Grid-Search")

     # Display the confusion matrix
    print(f"\nConfusion Matrix")
```

```
    print(confusion_matrix(y_test, y_pred))

    print(f"\nClassification Report", report)


    # to Print the overall accuracy of the model
    print(f"\nAccuracy Score", accuracy_score(y_test, y_pred))

tuned_models_scaled = [
    (grid_log.best_estimator_, "Logistic Regression"),
    (grid_knn.best_estimator_, "K-Nearest Neighbors")
]

# Evaluate all models
for model, name in tuned_models_scaled:
    evaluate_model(model, X_test_scaled, y_test, name)
```

```
Logistic Regression Evaluation After Grid-Search

Confusion Matrix
[[113   6]
 [ 19 112]]

Classification Report               precision    recall  f1-score   support

           0       0.86      0.95      0.90       119
           1       0.95      0.85      0.90       131

    accuracy                           0.90       250
   macro avg       0.90      0.90      0.90       250
weighted avg       0.90      0.90      0.90       250


Accuracy Score 0.9

K-Nearest Neighbors Evaluation After Grid-Search

Confusion Matrix
[[116   3]
 [ 24 107]]

Classification Report               precision    recall  f1-score   support

           0       0.83      0.97      0.90       119
           1       0.97      0.82      0.89       131

    accuracy                           0.89       250
   macro avg       0.90      0.90      0.89       250
weighted avg       0.90      0.89      0.89       250


Accuracy Score 0.892
```

With a 90% accuracy rate, the Logistic Regression model after grid-sear accurately classified 112 returns and 113 non-returns, with six false positives (predicted returns that didn't develop) and 19 false negatives (predicted no returns that did). With an accuracy of 89.2%, the K-Nearest Neighbours model came in minute, showing 116 accurate non-returns, 107 right returns, 3 false positives, and 24 false negatives. With fewer missed actual returns (false negatives) as K-NN, Logistic Regression is a somewhat more effective choice even though both models perform well. It also showed a little higher overall accuracy and a more balanced distribution of faults.

Classification task - Discussion

- Evaluate the performance of each model. If applicable, explain any additional techniques you applied and how it improved - or not - the performance of the model.

K-Nearest Neighbours (KNN) and Logistic Regression showed strong performance, with Logistic Regression showing marginally superior overall outcomes. With a confusion matrix displaying 113 true negatives, 112 true positives, six incorrect positives, and 19 false negatives, logistic regression obtained an accuracy of 90%. It also means that the model missed 19 actual returns, even though it correctly expected the majority of outcomes. Hyperparameters such as regularisation strength (C), penalty, and solver have been altered using GridSearchCV.The optimal combination improves interpretability and model adaptation.

- Compare the models: Which model would you recommend for deployment? Justify your answer, explaining the grounds for the choice. Is this a good model overall?

It is advised to employ Logistic Regression for the deployment of one of the two models. It showed a better balance between false positives and false negatives and achieved a slightly higher accuracy score (90%) than K-Nearest Neighbours (89.2%). In circumstances in which failure to anticipate a product return (a false negative) could have operational or financial consequences, logistic regression produced fewer false negatives than KNN. In addition, the greater comprehension of logistic regression facilitates explaining of forecasts to stakeholders, which is important in practical business uses. Both models perform well in terms of performance, but Logistic

Regression is also less sensitive to noise or irrelevant details and is computationally more economical, especially if dealing with larger datasets. By employing GridSearchCV for hyperparameter modification, its performance was further improved while maintaining good standardisation. All things taken into account, this is a good model—it is scalable for deployment, balances mistakes well, performs with high accuracy, and is explicable. Because of those features, logistic regression is a dependable and valuable option for evaluating product returns in such a scenario.

- Explain your findings as if presenting them to a non-technical stakeholder. Focus on insights and actionable strategies.

To find out which customers are most likely to return items, I took two prediction models as part of my analysis. Although they both did well, I found that the Logistic Regression model gave findings that were simpler to understand and analyse and had a slightly higher accuracy rate, correctly predicting outcomes around ninety per cent of the time. One of my main findings is that specific buyer behaviours, such as past orders or the amount of time since their last buy, can indicate the possibility of a return, by forecasting these trends. In considering this, I recommend running the Logistic Regression model. It is a useful tool for actual business decisions since it is not only accurate but also constant and transparent. This approach can be employed to reduce return-related expenses, increase efficiency in operation, and provide more individualised customer service.

PART II

Clustering task - Introduction

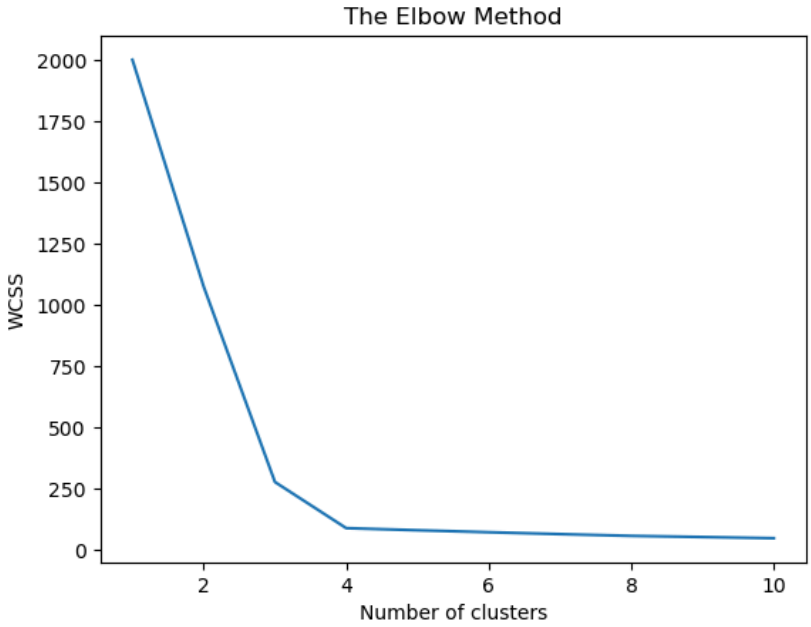
Briefly describe the purpose of this analysis

The goal of this analysis is to utilize unsupervised learning techniques to segment customers. The objective is to find independent customer groups with comparable spending and trips made by using clustering algorithms on features like budget and yearly_trips. This helps the company improve customer engagement, customize services, and develop targeted marketing strategies. Clustering is perfect for studying hidden patterns and identifying naturally occurring groupings in the data that can lead to actionable insights, since the method does not rely on predefined labels like classification does.

In [20]:
```
# Load dataset for clustering task
df = pd.read_csv('Dataset_cluster.csv')
df.head()
X = df[['yearly_trips','budget']]
```

In [21]:
```
# K Means
sc = StandardScaler()
X_scale = sc.fit_transform(X)

wcss = []

for i in range(1, 11):
    kmeans = KMeans(
        n_clusters=i,
        init='k-means++',
        n_init=10,
        random_state=42
    )

    kmeans.fit(X_scale)

    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()



kmeans = KMeans(n_clusters = 4, init = 'k-means++', random_state = 46)
y_kmeans = kmeans.fit_predict(X_scale)



kmeans = KMeans(n_clusters=4, init='k-means++', random_state=55)
y_kmeans = kmeans.fit_predict(X_scale)

# Define cluster colors and labels
colors = ['red', 'blue', 'green', 'orange']
labels = ['Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4']
X_array = X.values

# Plot each cluster

for i in range(4):
```

```
    plt.scatter(
        X_array[y_kmeans == i, 0],
        X_array[y_kmeans == i, 1],
        s=100,
        c=colors[i],
        label=labels[i]
    )
# Transform centroids back to original scale
centroids_original = sc.inverse_transform(kmeans.cluster_centers_)
# Plot the centroids

plt.scatter(
    centroids_original[:, 0],
    centroids_original[:, 1],
    s=300,
    c='yellow',
    label='Centroids',
    edgecolors='black'
)


plt.title('Clusters of Customers')
plt.xlabel('yearly_trips')
plt.ylabel('budget')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.grid(True)
plt.show()
```



Clustering task - Discussion Part a

The preprocessing

K-Means preprocessing involves loading the clustering dataset from Dataset_cluster.csv from the supplied notebook. The budget and yearly_trips are the pertinent features that were chosen for clustering. Calculating the distance between data points and the centre of the cluster is the foundation of K-Means clustering. Features that have wider ranges can have a disproportionate impact on the distance computation if their scales differ (for example, yearly_trips may range from 1 to 100, whereas budget may range from 100 to 10,000).

Implementation

The gradual K-Means technique is implemented with sklearn.cluster.K-Means starts with an initialisation stage in which k data points are chosen at random to serve as centroids (k-means++ is frequently used to boost convergence). Each data point is then assigned to its closest centroid in the assignment step, usually based on Euclidean distance, creating k different clusters. The centroids are then recalculated as the average of all data points within their respective clusters in an update phase that follows.

Output of the K-Means clustering algorithm

The cluster assignment for every data point and the clusters' final centroids are the main outcomes of the K-Means algorithm. The notebook shows the creation of a scatter plot for presentation, even if the clearly visible code snippets do not provide explicit numerical output for K-Means clustering (such as cluster labels for each data point or centroid coordinates). In order to provide a visual representation of the groups that have been generated, this plot would normally display data points coloured matching with their assigned cluster and maybe the cluster centroids. Annual trips and budget are suggested as the axes for visualising these clusters in the plot title and labels.

Strengths:

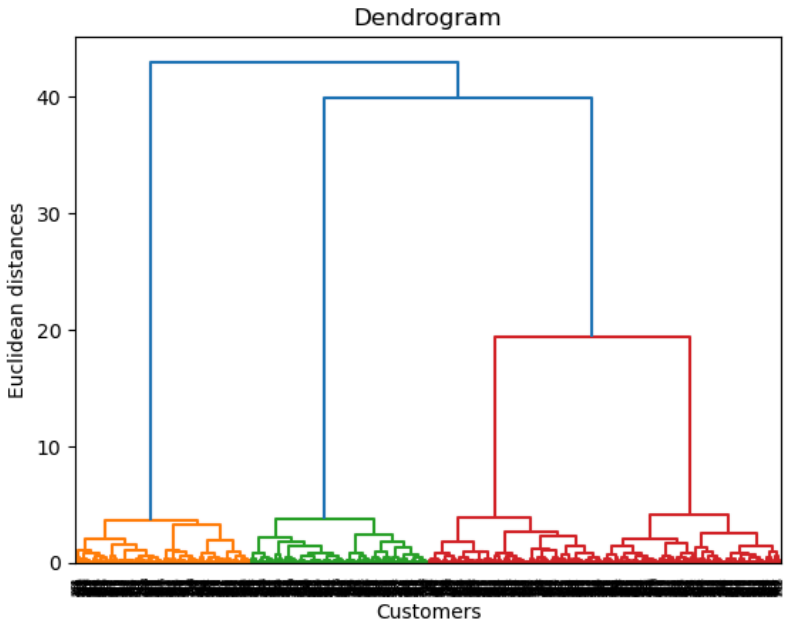
Considering its speed and ease of use, K-Means clustering is quite useful. It provides computational efficiency for large datasets and relatively fast convergence. It is a helpful instrument for quickly identifying first clusters during exploratory data analysis because of its scalability, which enables it to handle large numbers of data points.

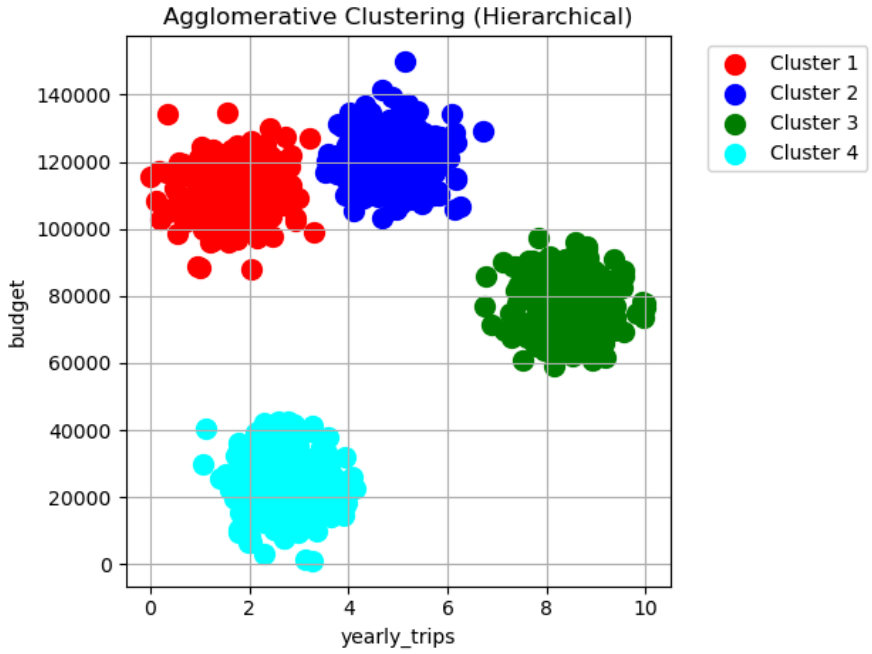
Weaknesses

Although K-means++ initialisation in scikit-learn helps prevent this issue, K-Means clustering has a number of disadvantages, including its sensitivity to the initial location of centroids, which can result in suboptimal or local optima. The requirement to pre-define the number of clusters (k) is an important drawback, frequently mandating the use of outside techniques like the Elbow approach or Silhouette analysis, neither of which was specifically illustrated here. Additionally, K-Means struggles with irregularly shaped, variable density, or interconnected clusters because it assumes that clusters be roughly spherical with equal variance. Additionally, because outliers have the ability to distort cluster boundaries and significantly affect centroid placements, it is extremely sensitive to them.

In [22]:
```
# Agglomerative Clustering

# Step 1: Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # X must be your feature DataFrame

# Agglomerative Clustering
dendrogram = sch.dendrogram(sch.linkage(X_scaled, method='ward'))

plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()

hc = AgglomerativeClustering(n_clusters=4, linkage='ward')
y_hc = hc.fit_predict(X_scaled)

X_array = X.to_numpy()

# Define colors and labels for clusters
colors = ['red', 'blue', 'green', 'cyan']
labels = ['Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4']

# Plot each cluster
for i in range(4):
    plt.scatter(
        X_array[y_hc == i, 0],
```

```
        X_array[y_hc == i, 1],
        s=100,
        c=colors[i],
        label=labels[i]
    )

plt.title('Agglomerative Clustering (Hierarchical)')
plt.xlabel('yearly_trips')
plt.ylabel('budget')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```





Clustering task - Discussion Part b

- Describe the preprocessing, implementation, and output of the Agglomerative Hierarchical clustering algorithm.

Preprocessing-Accurate distance estimates between data points are necessary for the Agglomerative Hierarchical Clustering algorithm to work correctly.Therefore, feature scaling was crucial. Budget and yearly_trips, the two features used for clustering, were scaled to a common level using StandardScaler. This stage ensures that no feature's magnitude changes cause it to significantly impact the clustering process.

Implementation-The Ward linkage method, which decreases the variance within each cluster during merging, was used together with scikit-learn's AgglomerativeClustering class to implement the strategy. Scipy's linkage and dendrogram functions were used to create a dendrogram prior to selecting the number of clusters.The dendrogram helped determine the right number of groups, in this case four,

and gave a visual depiction of the hierarchical clustering process.Each data point is first viewed as a separate cluster in the agglomerative process, which then repeatedly integrates the closest clusters until the target number is achieved.

Output- A scatter plot with unique colours and labels was used to show the resulting clusters. This data made it possible to compare consumer categories based on spending patterns and trip frequency in a simple manner. Targeted marketing or service efforts can be promoted by the relevant groups that the clusters showed.

-Discuss the performance of the clustering algorithm, highlighting strengths and weaknesses.

Based on yearly_trips and budget, the Agglomerative Hierarchical Clustering method efficiently separated the data into four separate clusters. Four clusters were chosen depending on the dendrogram's clear illustration of how data points merge at different relationship lengths of time. The scatter plot displays compact, well-separated clusters, suggesting that the model effectively discovered the dataset's natural classes.

The ability of Agglomerative Hierarchical Clustering to produce compact, clearly unique, and non-overlapping clusters is one of its main advantages; this shows that it successfully captured significant patterns in buyer behaviour. A further important advantage is its interpretability, which makes it easier to choose the right number of clusters because the dendrogram graphically depicts the hierarchy of cluster creation. In addition, unlike K-Means, the approach does not require the specification of initial centroids, which lowers result difficulty and offers stronger cluster outcome consistency.

Agglomerative Hierarchical Clustering has major limitations, especially with regard to scalability; due to its high analysing cost, it is less suitable for large datasets. Additionally, it is vulnerable to outliers and noise; even one anomalous point can distort the dendrogram and alter the cluster structure as entirety. In addition, the algorithm's merging procedure is irreversible, which means that two clusters cannot be split after they have been arrived. If early judgements are made incorrectly, this could result in problematic effects. Nevertheless, the method worked well and provided meaningful and interpretable clusters for this small-to-medium dataset with well-separated features.

Clustering task - Discussion Part c

- Compare the performance of the two clustering algorithms: Which one yields clearer business insights or more stable clusters? Justify your choice

Agglomerative Hierarchical Clustering and K-Means both provide clearly various clusters when you evaluate their performance in your analysis.In this instance, however, Agglomerative Hierarchical Clustering generates more stable clusters and more clear business insights.For decision-makers who want to divide up their client base at numerous levels of detail, the dendrogram offers a hierarchical and visual knowledge of how customer groups are formed. Moreover, the Agglomerative Clustering scatter plot displays consistent and well-separated clusters according to budget and annual visits, which facilitates analysing of the segments for targeting strategies or personalised marketing. Agglomerative Clustering gives more stable and interpretable segmentation in this scenario, where the dataset is small and features are well-separated.Its hierarchical structure and visual clarity are especially beneficial in developing actionable customer groups while acquiring business insights.

- Based on your preferred clustering algorithm, describe the resulting clusters. Provide a clear label for each cluster, explain its key characteristics, and suggest a relevant strategy for engaging it.

Cluster 1: Infrequent Visitors on a High Budget

Highlights: Very high budget (~120,000+) and few yearly journeys.

Strategy: Provide premium experiences, special offers, and journey packages. Cluster 2: Moderately Priced Travellers

Details: High budgets (~100,000+) and moderate travel frequency (4–6 trips annually).

Strategy: To encourage more frequent travel, implement incentive programs, package deals, and priority service improvements.

Cluster 3: Regular Tourists on a Moderate Budget

Features: Mid-level spending (~60,000–80,000) yet a high frequency of travel (8–10 trips annually).

Strategy: Draw attention to reasonably priced vacations, membership packages that prioritise frequency over cost, or subscription choices.

Cluster 4: Budget-conscious infrequent travellers

Features: Minimal spending and infrequent travel.

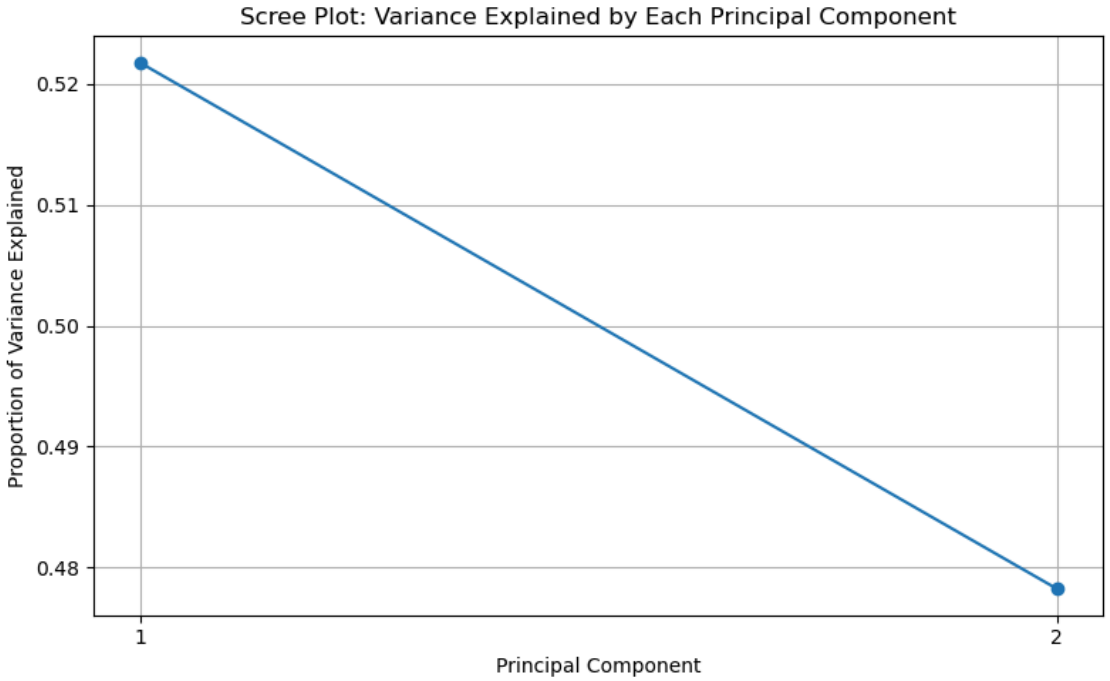
Strategy: Use promotions, temporary offers, and low-cost vacation packages to entice clients. Highlight value and ease of operation.

Additional part

- Apply PCA to reduce dimensionality and simplify the dataset. Evaluate how much variance is retained.

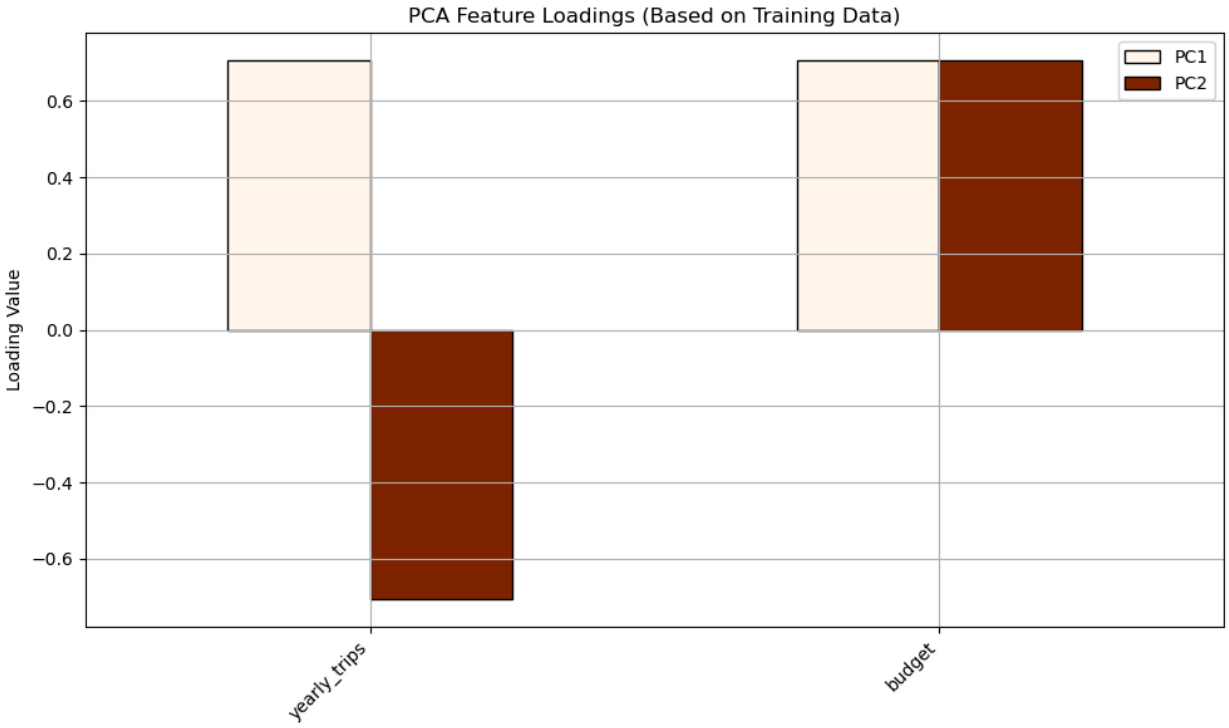
- Visualise and interpret the feature loadings of the principal components.
- Re-run your preferred clustering method using the PCA-reduced data. Visualise the new clusters.

In [23]:
```
# Analyse how many components you should compute
pca_full = PCA()
pca_full.fit(X_scaled)

# Variance explained by each component
explained_variance_ratio = pca_full.explained_variance_ratio_
components = np.arange(1, len(explained_variance_ratio) + 1)

# Plotting Scree Plot
plt.figure(figsize=(8, 5))
plt.plot(components, explained_variance_ratio, marker='o', linestyle='-')
plt.title('Scree Plot: Variance Explained by Each Principal Component')
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.xticks(components)
plt.grid(True)
plt.tight_layout()
plt.show()
```



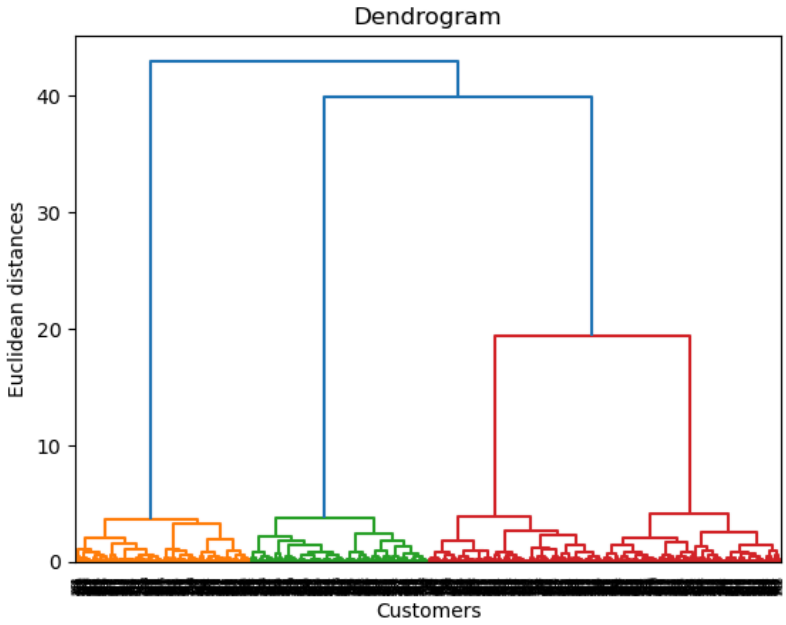
In [24]:
```
# Apply PCA with the number of components selected
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

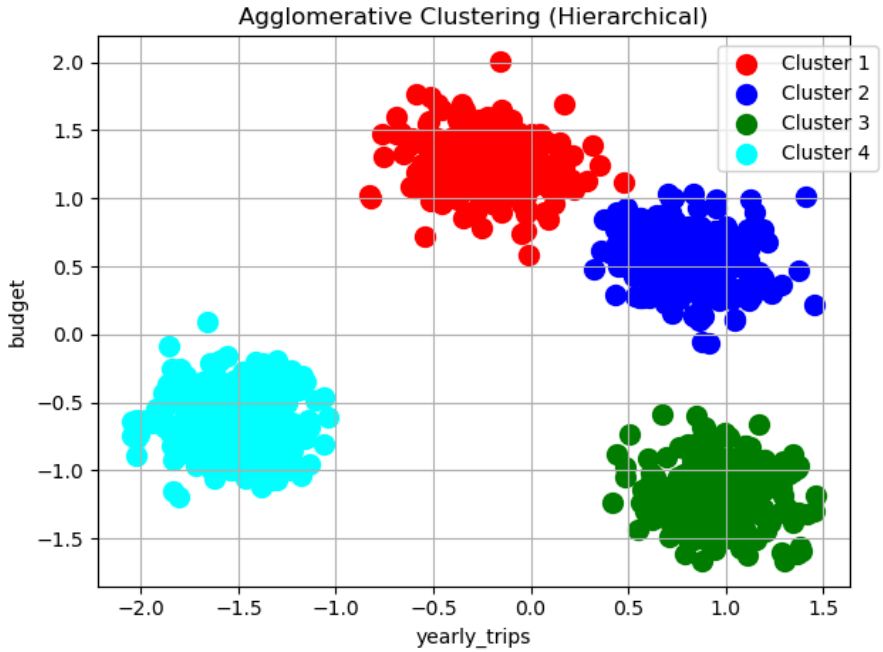
In [25]:
```
# Display how the original features contribute to the new components produced

loadings = pca.components_.T # Extract the principal component loadings

# Assuming feature_names holds your original column names
pca_loadings = pd.DataFrame(loadings, index=X.columns, columns=['PC1', 'PC2'])

pca_loadings.plot(kind='bar', figsize=(10, 6), colormap='Oranges', edgecolor='black')
plt.title('PCA Feature Loadings (Based on Training Data)')
plt.ylabel('Loading Value')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.grid(True)
plt.show()
```



In [26]:

```python
# Agglomerative Clustering

dendrogram = sch.dendrogram(sch.linkage(X_pca, method='ward'))

plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()

hc = AgglomerativeClustering(n_clusters=4, linkage='ward')
y_hc = hc.fit_predict(X_pca)

# Define colors and labels for clusters
colors = ['red', 'blue', 'green', 'cyan']
labels = ['Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4']

# Plot each cluster
for i in range(4):
    plt.scatter(
        X_pca[y_hc == i, 0],
        X_pca[y_hc == i, 1],
        s=100,
        c=colors[i],
        label=labels[i]
    )

plt.title('Agglomerative Clustering (Hierarchical)')
plt.xlabel('yearly_trips')
plt.ylabel('budget')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper right')
plt.grid(True)
plt.tight_layout()
plt.show()
```





Additional part - Discussion

- Discuss how the components produced through PCA relate to the original dataset.

The original features (budget, annual trips) are converted into new, uncorrelated variables known as principal components using Principal Component Analysis (PCA). These elements are arranged according to how much variance they represent in the dataset and are linear combinations of the original variables. Two principal components were chosen after each component's explained variance had been evaluated using a scree plot.

- Compare the clustering outcome from PCA-reduced data to your earlier clustering. Are the groups similar? Why or why not?

Though there might be a few differences in the boundaries created, the clustering result on PCA-reduced data generated groupings structurally similar to the original clustering. The reason for the similarity is that PCA ensured the underlying data patterns stayed intact by maintaining the majority of the variation from the original features (yearly_trips, budget) in the first two components. Nevertheless, since PCA converts the original feature space into new axes that are a linear combination of the input variables, the resulting clusters may undergo significant shifts in composition or position. These shifts may be caused by reduced dimensionality and the removal of tiny variance components that may have affected previous dividing limits.

Acknowledgement

I sincerely thank the course team for providing valuable resources, especially the coding files from Weeks 6, 8, and 9, which played a key role in shaping the structure and logic of my assignment. I also used ChatGPT to clarify certain technical concepts during the

development process. All tasks involving data visualization, clustering, model evaluation, and interpretation were completed independently, based on my own analysis and understanding. This submission reflects my personal learning journey, complemented by course content and the responsible use of digital tools to enhance comprehension.

In [ ]:

In [ ]: