Customer_Segmentation

November 2, 2025

1 Import the Necessary Liabraries

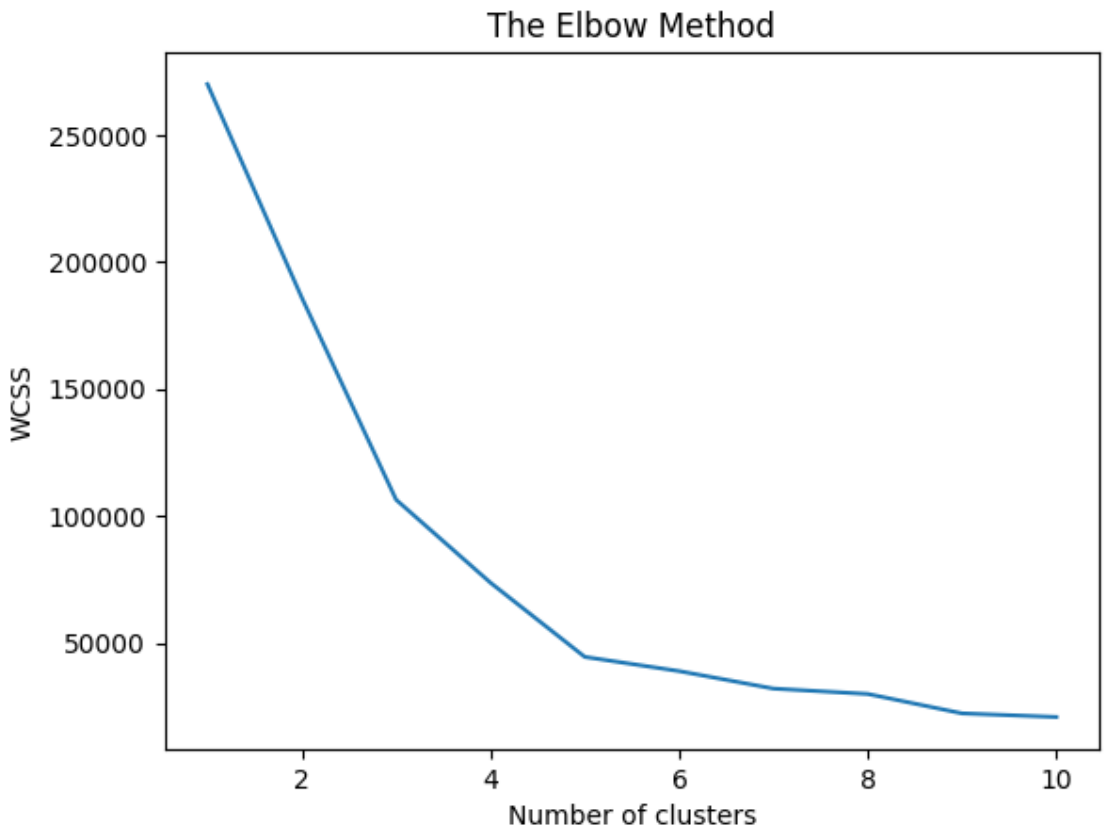
[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2 Load Dataset

[2]:
```
dataset = pd.read_csv(r"/content/Mall_Customers (1).csv")
X = dataset.iloc[:, [3, 4]].values
```

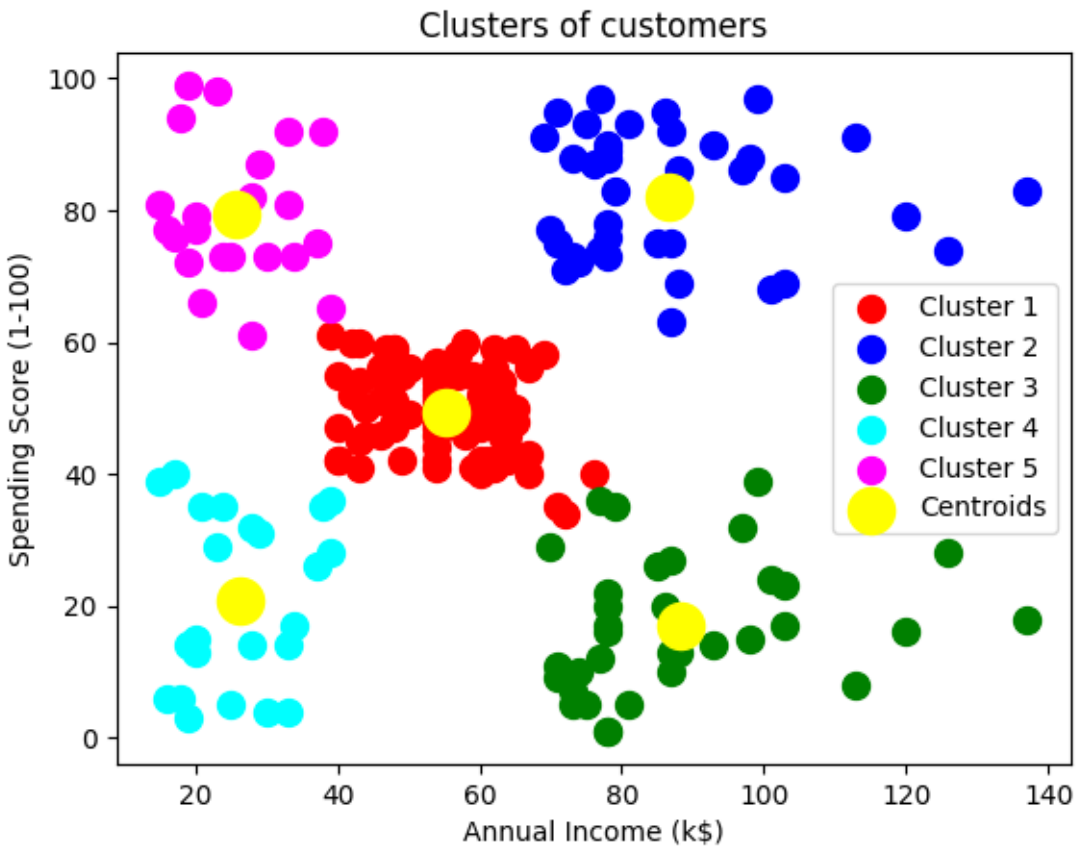
[3]:
```
from sklearn.cluster import KMeans
wcss = []
```

[4]:
```
for i in range(1,11):
  kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
  kmeans.fit(X)
  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()
```



[5]:
```
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=0)
y_kmeans = kmeans.fit_predict(X)
```

[6]:
```
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 = 'blue',␣
 ↪label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green',␣
 ↪label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan',␣
 ↪label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta',␣
 ↪label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s =␣
 ↪300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



[7]:
```
y_kmeans
```

[7]:
```
array([3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4,
       3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 0,
       3, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 1, 2, 1, 2, 1,
       0, 1, 2, 1, 2, 1, 2, 1, 2, 1, 0, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
       2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
       2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
       2, 1], dtype=int32)
```

[8]:
```
dataset['cluster'] = y_kmeans
```

[9]:
```
import os
os.getcwd()
```

[9]:
```
'/content'
```

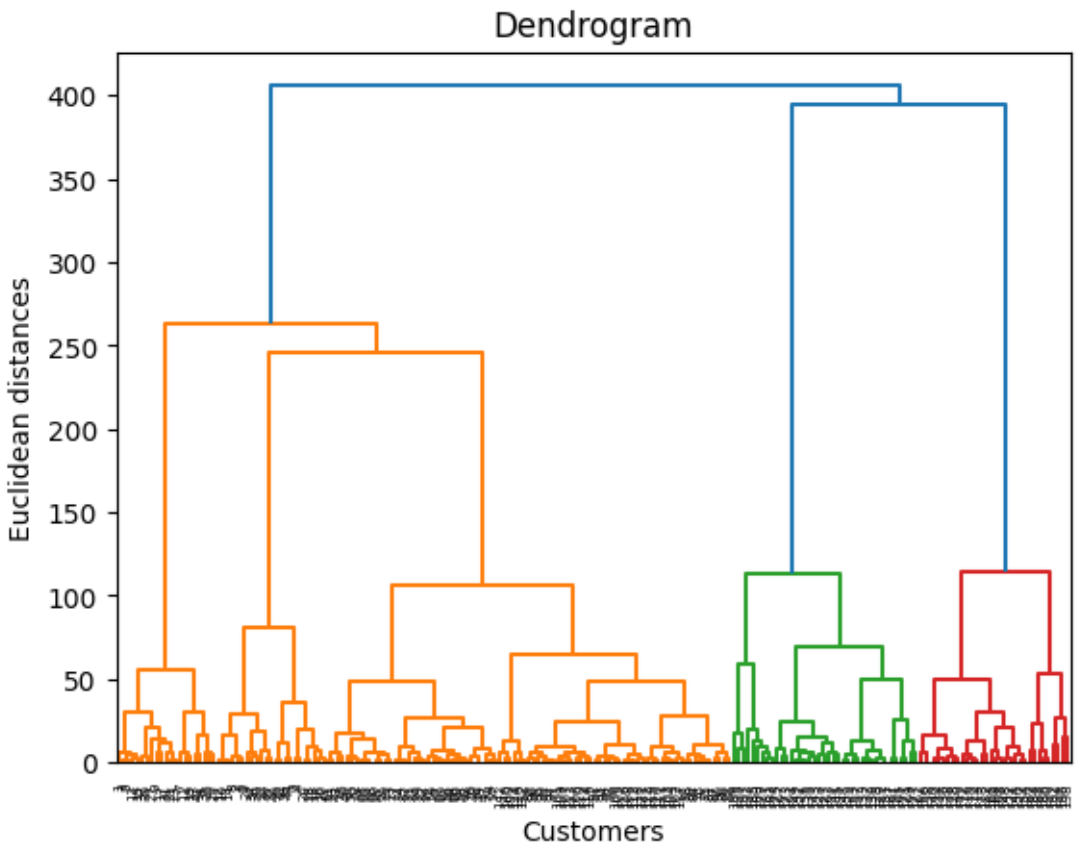
[10]:
```
dataset
```

[10]:
```
     CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)  \
0             1    Male   19                  15                      39
1             2    Male   21                  15                      81
2             3  Female   20                  16                       6
3             4  Female   23                  16                      77
4             5  Female   31                  17                      40
..          …     …  …                 …                       …
195         196  Female   35                 120                      79
196         197  Female   45                 126                      28
197         198    Male   32                 126                      74
198         199    Male   32                 137                      18
199         200    Male   30                 137                      83

     cluster
0          3
1          4
2          3
3          4
4          3
..       …
195        1
196        2
197        1
198        2
199        1

[200 rows x 6 columns]
```

[ ]:
```

```

[11]:
```
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
```

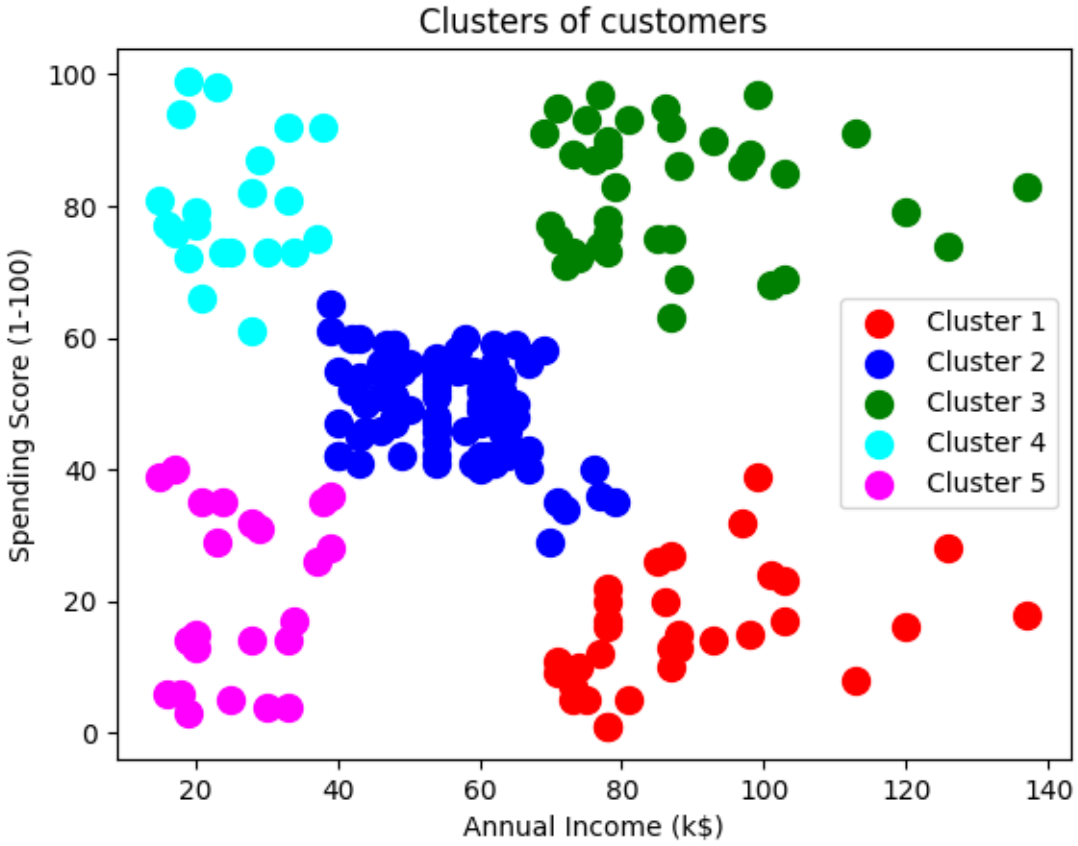
[12]:
```
dendogram = sch.dendrogram(sch.linkage(X, method='ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
```



[14]:
```
# Training the Hierchical Clustring model on the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters=5, linkage='ward')
y_hc = hc.fit_predict(X)
```

[15]:
```
# Visulising the clusters
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('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
```

```
plt.legend()
plt.show()
```



[16]:
```
dataset['cluster'] = y_hc
```

[17]:
```
y_hc
```

[17]:
```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2])
```

[18]:
```
dataset
```

[18]:
```
     CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)  \
0             1    Male   19                  15                      39
1             2    Male   21                  15                      81
2             3  Female   20                  16                       6
3             4  Female   23                  16                      77
4             5  Female   31                  17                      40
..          …     …  …                 …                       …
195         196  Female   35                 120                      79
196         197  Female   45                 126                      28
197         198    Male   32                 126                      74
198         199    Male   32                 137                      18
199         200    Male   30                 137                      83

     cluster
0          4
1          3
2          4
3          3
4          4
..       …
195        2
196        0
197        2
198        0
199        2

[200 rows x 6 columns]
```

[ ]:

3 Combining K-Means and Hierachical Clustering with Gradio Interface

[19]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as sch
import gradio as gr
import warnings
warnings.filterwarnings('ignore')
```

[20]:
```
# Set style for better visualizations
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")
```

[21]:
```
def load_data():
    """Load and return the customer dataset"""
    np.random.seed(42)
    n_samples = 200

    # Create synthetic data similar to Mall_Customers
    data = {
        'CustomerID': range(1, n_samples + 1),
        'Gender': np.random.choice(['Male', 'Female'], n_samples),
        'Age': np.random.normal(40, 10, n_samples).astype(int),
        'Annual Income (k$)': np.random.normal(60, 15, n_samples).astype(int),
        'Spending Score (1-100)': np.random.normal(50, 20, n_samples).
 ↪astype(int)
    }

    # Ensure values are within reasonable ranges
    data['Age'] = np.clip(data['Age'], 18, 70)
    data['Annual Income (k$)'] = np.clip(data['Annual Income (k$)'], 15, 140)
    data['Spending Score (1-100)'] = np.clip(data['Spending Score (1-100)'], 1,␣
 ↪100)

    return pd.DataFrame(data)
```

[22]:
```
def perform_kmeans(X, n_clusters=5):
    """Perform K-Means clustering and return results"""
    # Standardize the features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Apply K-Means
    kmeans = KMeans(n_clusters=n_clusters, init='k-means++', random_state=42,␣
 ↪n_init=10)
    y_kmeans = kmeans.fit_predict(X_scaled)

    # Calculate silhouette score
    silhouette_avg = silhouette_score(X_scaled, y_kmeans)

    # Get centroids in original scale
    centroids_original = scaler.inverse_transform(kmeans.cluster_centers_)

    return y_kmeans, silhouette_avg, centroids_original
```

[24]:
```python
def perform_hierarchical(X, n_clusters=5, linkage_method='ward'):
    """Perform Hierarchical clustering and return results"""
    # Standardize the features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Apply Hierarchical Clustering
    hc = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage_method)
    y_hc = hc.fit_predict(X_scaled)

    # Calculate silhouette score
    silhouette_avg = silhouette_score(X_scaled, y_hc)

    return y_hc, silhouette_avg
```

[26]:
```python
def create_dendrogram(X, method='ward'):
    """Create and return a dendrogram plot"""
    plt.figure(figsize=(12, 6))
    dendrogram = sch.dendrogram(sch.linkage(X, method=method))
    plt.title('Dendrogram')
    plt.xlabel('Customers')
    plt.ylabel('Euclidean distances')
    plt.tight_layout()
    return plt
```

[27]:
```python
def plot_clusters(X, y, algorithm_name, centroids=None):
    """Create and return a cluster visualization plot"""
    plt.figure(figsize=(10, 6))

    colors = ['red', 'blue', 'green', 'cyan', 'magenta', 'orange', 'purple',␣
 ↪'brown', 'pink', 'gray']
    cluster_names = ['Careful', 'Standard', 'Target', 'Careless', 'Sensible',
                     'Budget', 'Premium', 'Explorers', 'Traditional', 'Trendy']

    n_clusters = len(np.unique(y))

    for i in range(n_clusters):
        plt.scatter(X[y == i, 0], X[y == i, 1], s=50, c=colors[i],
                    label=f'Cluster {i+1}: {cluster_names[i]}', alpha=0.7)

    if centroids is not None:
        plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='yellow',
                    marker='D', edgecolor='black', label='Centroids')

    plt.title(f'Customer Segments - {algorithm_name}', fontweight='bold')
    plt.xlabel('Annual Income (k$)')
    plt.ylabel('Spending Score (1-100)')
```

```
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    return plt
```

[28]:
```
def plot_elbow_method(X, max_clusters=10):
    """Create and return an elbow method plot"""
    wcss = []

    for i in range(1, max_clusters + 1):
        kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42,␣
↪n_init=10)
        kmeans.fit(X)
        wcss.append(kmeans.inertia_)

    plt.figure(figsize=(10, 6))
    plt.plot(range(1, max_clusters + 1), wcss, marker='o', linestyle='--')
    plt.title('Elbow Method for Optimal Number of Clusters', fontweight='bold')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
    plt.xticks(range(1, max_clusters + 1))
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    return plt
```

[29]:
```
def analyze_clusters(dataset, cluster_labels, algorithm_name):
    """Analyze and return cluster statistics"""
    dataset = dataset.copy()
    dataset['Cluster'] = cluster_labels

    cluster_stats = dataset.groupby('Cluster').agg({
        'Age': 'mean',
        'Annual Income (k$)': 'mean',
        'Spending Score (1-100)': 'mean',
        'CustomerID': 'count'
    }).rename(columns={'CustomerID': 'Count'}).round(2)

    # Add interpretation
    interpretations = []
    for cluster_id in cluster_stats.index:
        income = cluster_stats.loc[cluster_id, 'Annual Income (k$)']
        spending = cluster_stats.loc[cluster_id, 'Spending Score (1-100)']

        if income > 75 and spending > 60:
            interpretation = "High-income, high-spending customers. Prime␣
↪targets for premium products."
        elif income > 75 and spending <= 60:
```

```
            interpretation = "High-income but cautious spenders. Need␣
↪persuasion to spend more."
        elif income <= 75 and spending > 60:
            interpretation = "Moderate income but high spenders. Value-seeking␣
↪customers."
        else:
            interpretation = "Moderate income, cautious spenders.␣
↪Budget-conscious customers."

        interpretations.append(interpretation)

    cluster_stats['Interpretation'] = interpretations

    return cluster_stats
```

[30]:
```
def run_customer_segmentation(n_clusters, algorithm, linkage_method='ward'):
    """Main function to run customer segmentation"""
    # Load data
    dataset = load_data()
    X = dataset[['Annual Income (k$)', 'Spending Score (1-100)']].values

    # Run selected algorithm
    if algorithm == "K-Means":
        labels, silhouette, centroids = perform_kmeans(X, n_clusters)
        cluster_plot = plot_clusters(X, labels, "K-Means Clustering", centroids)
    else:
        labels, silhouette = perform_hierarchical(X, n_clusters, linkage_method)
        cluster_plot = plot_clusters(X, labels, "Hierarchical Clustering")

    # Create additional visualizations
    elbow_plot = plot_elbow_method(X)
    dendrogram_plot = create_dendrogram(X, linkage_method)

    # Analyze clusters
    cluster_stats = analyze_clusters(dataset, labels, algorithm)

    # Add cluster labels to dataset
    dataset_with_clusters = dataset.copy()
    dataset_with_clusters['Cluster'] = labels

    return (cluster_plot, elbow_plot, dendrogram_plot,
            f"Silhouette Score: {silhouette:.4f}", cluster_stats,␣
↪dataset_with_clusters)

# Create Gradio interface
with gr.Blocks(title="Customer Segmentation Analysis") as demo:
    gr.Markdown("#  Customer Segmentation Analysis")
```

```
    gr.Markdown("This application segments customers using clustering␣
↪algorithms to identify distinct customer groups for targeted marketing.")

    with gr.Row():
        with gr.Column(scale=1):
            n_clusters = gr.Slider(minimum=2, maximum=10, value=5, step=1,
                                   label="Number of Clusters")
            algorithm = gr.Radio(choices=["K-Means", "Hierarchical"],
                                 value="K-Means", label="Clustering Algorithm")
            linkage_method = gr.Dropdown(choices=["ward", "complete",␣
↪"average", "single"],
                                         value="ward", label="Linkage Method (for␣
↪Hierarchical)")
            run_btn = gr.Button("Run Analysis", variant="primary")

        with gr.Column(scale=2):
            with gr.Tab("Clusters Visualization"):
                cluster_plot = gr.Plot(label="Customer Segments")

            with gr.Tab("Elbow Method"):
                elbow_plot = gr.Plot(label="Elbow Method Plot")

            with gr.Tab("Dendrogram"):
                dendrogram_plot = gr.Plot(label="Dendrogram")

            with gr.Tab("Cluster Statistics"):
                silhouette_output = gr.Textbox(label="Model Performance")
                cluster_stats = gr.Dataframe(label="Cluster Analysis",
                                             headers=["Cluster", "Avg Age", "Avg␣
↪Income",
                                                      "Avg Spending", "Count",␣
↪"Interpretation"])

            with gr.Tab("Data with Clusters"):
                data_output = gr.Dataframe(label="Customer Data with Cluster␣
↪Assignments")

    # Set up event handling
    run_btn.click(
        fn=run_customer_segmentation,
        inputs=[n_clusters, algorithm, linkage_method],
        outputs=[cluster_plot, elbow_plot, dendrogram_plot,
                silhouette_output, cluster_stats, data_output]
    )

    # Add examples
```

```
    gr.Examples(
        examples=[
            [5, "K-Means", "ward"],
            [4, "Hierarchical", "complete"],
            [6, "K-Means", "ward"],
            [3, "Hierarchical", "average"]
        ],
        inputs=[n_clusters, algorithm, linkage_method],
        outputs=[cluster_plot, elbow_plot, dendrogram_plot, silhouette_output,␣
↪cluster_stats, data_output],
        fn=run_customer_segmentation,
        cache_examples=True
    )
```

4 Run the application

[31]:
```
if __name__ == "__main__":
    demo.launch(share=True)
```

```
Caching examples at: '/content/.gradio/cached_examples/24'
Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://a4cce556b9f2e95681.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades,
run `gradio deploy` from the terminal in the working directory to deploy to
Hugging Face Spaces (https://huggingface.co/spaces)
```

```
<IPython.core.display.HTML object>
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: