Import Libraries

In [1]:
```
# for data manipulation

import numpy as np
import pandas as pd

# for data visualization

import matplotlib.pyplot as plt
import seaborn as sns
from scipy.cluster import hierarchy

# for data preprossesing

from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.decomposition import PCA

# for model training

from sklearn.model_selection import train_test_split,GridSearchCV,RandomizedSearchCV
from sklearn.cluster import KMeans, AgglomerativeClustering

#for model evalueation

from sklearn.metrics import silhouette_score

# Miscellaneous

import warnings
warnings.filterwarnings("ignore")
```

Data Gathering

In [2]:
```
df = pd.read_csv(r"C:\Users\Dell\Downloads\customers.csv")
df
```

Out[2]:

	CustomerID	Gender	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

200 rows × 5 columns

Exploratory Data Analysis

In [3]:
```
df.shape
```

Out[3]:
```
(200, 5)
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)'],
      dtype='object')
```

In [5]:
```
df.info()                                                          # give overall information
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   CustomerID              200 non-null    int64 
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64 
 3   Annual Income (k$)      200 non-null    int64 
 4   Spending Score (1-100)  200 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

In [6]:
```
df.describe()                                            # give statistical information
```

Out[6]:

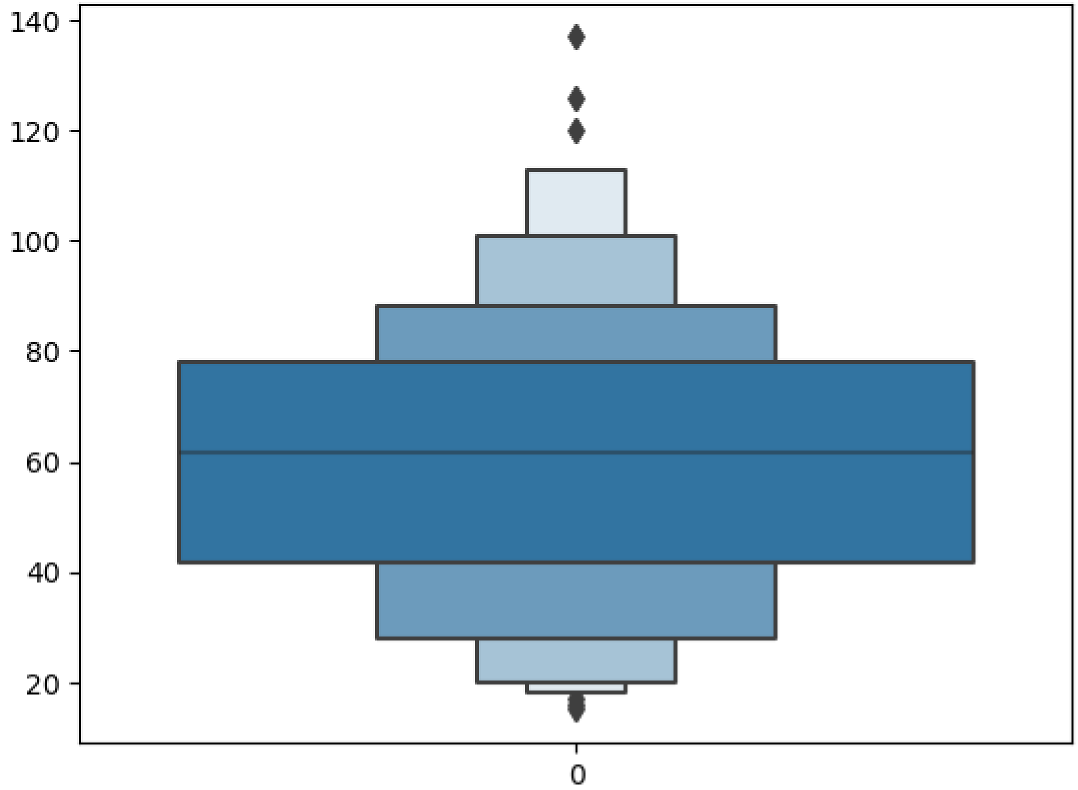
	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

In [7]:
```
df.isna().sum()                                          # give the number of missing values
```

Out[7]:
```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

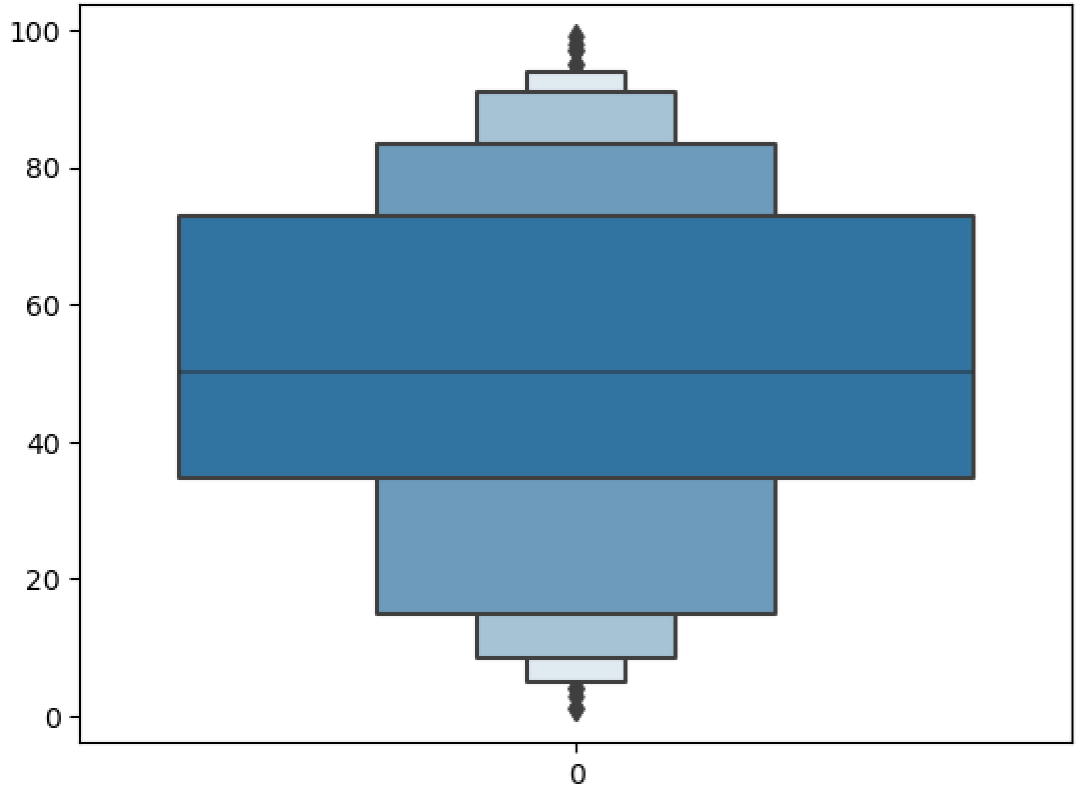
In [8]:
```
sns.boxenplot(df["Annual Income (k$)"])
```

Out[8]: <Axes: >



In [9]:
```
sns.boxenplot(df["Spending Score (1-100)"])
```

Out[9]: <Axes: >

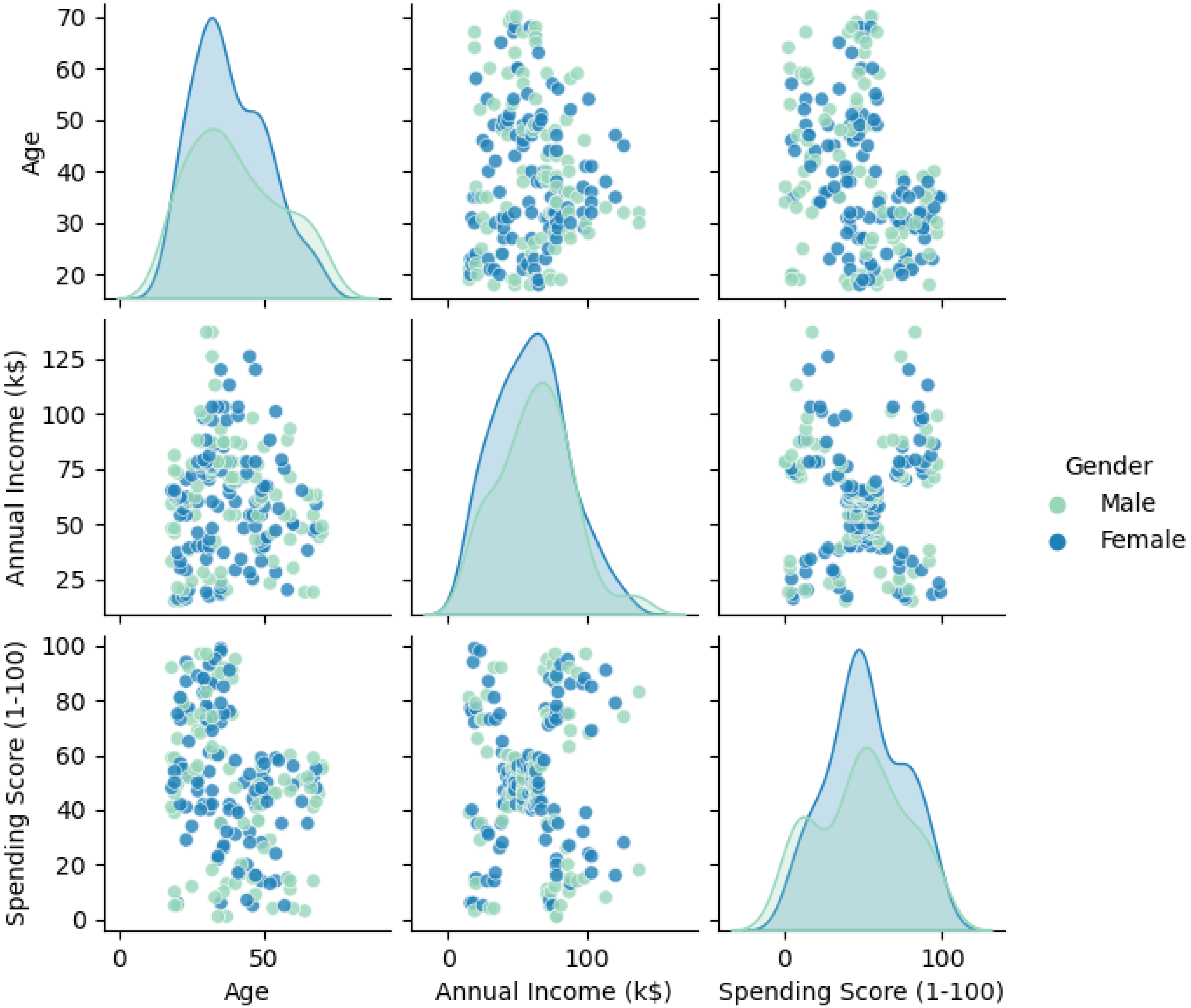


In [10]:
```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3-Q1
LowerTail = Q1-1.5*IQR
UpperTail = Q3+1.5*IQR
```

In [11]:
```
show_outliers = (df<LowerTail)|(df>UpperTail)
outlier_count = show_outliers.sum()
outlier_count
```

Out[11]:
```
Age                       0
Annual Income (k$)        2
CustomerID                0
Gender                    0
Spending Score (1-100)    0
dtype: int64
```

In [12]:
```
sns.pairplot(df, x_vars = ["Age", "Annual Income (k$)", "Spending Score (1-100)"],
               y_vars = ["Age", "Annual Income (k$)", "Spending Score (1-100)"],
               hue = "Gender",
               kind= "scatter",
               palette = "YlGnBu",
               height = 2,
               plot_kws={"s": 35, "alpha": 0.8});
```


Age
Annual Income (k$)
Spending Score (1-100)
Gender
Male
Female
70
60
50
40
30
20
125
100
75
50
25
100
80
60
40
20
0
0
50
0
100
0
100

In [13]:
```
sns.heatmap(df.corr(),annot=True)
```

Out[13]: <Axes: >

Feature Engineering

In [14]:
```
df = df.drop(["CustomerID"],axis=1)
df
```

Out[14]:

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

200 rows × 4 columns

In [15]:
```
# handling outliers
```

In [16]:
```
def remove_outliers(column):
    q1 = column.quantile(0.25)
    q3 = column.quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    return column[(column >= lower_bound) & (column <= upper_bound)]
```

In [17]:
```
df = df.apply(lambda col: remove_outliers(col) if np.issubdtype(col.dtype, np.number) else col)
```

In [18]:
```
show_outliers = (df<LowerTail)|(df>UpperTail)
outlier_count = show_outliers.sum()
outlier_count
```

Out[18]:
```
Age                       0
Annual Income (k$)        0
CustomerID                0
Gender                    0
Spending Score (1-100)    0
dtype: int64
```

In [19]:
```
# handling missing values
```

In [20]:
```
df.isna().sum()
```

Out[20]:
```
Gender                    0
Age                       0
Annual Income (k$)        2
Spending Score (1-100)    0
dtype: int64
```

In [21]:
```
df["Annual Income (k$)"].value_counts()
```

Out[21]:
```
54.0     12
78.0     12
60.0      6
87.0      6
71.0      6
         ..
58.0      2
59.0      2
16.0      2
61.0      2
126.0     2
Name: Annual Income (k$), Length: 63, dtype: int64
```

In [22]:
```
df["Annual Income (k$)"].unique()
```

Out[22]:
```
array([ 15.,  16.,  17.,  18.,  19.,  20.,  21.,  23.,  24.,  25.,  28.,
        29.,  30.,  33.,  34.,  37.,  38.,  39.,  40.,  42.,  43.,  44.,
        46.,  47.,  48.,  49.,  50.,  54.,  57.,  58.,  59.,  60.,  61.,
        62.,  63.,  64.,  65.,  67.,  69.,  70.,  71.,  72.,  73.,  74.,
        75.,  76.,  77.,  78.,  79.,  81.,  85.,  86.,  87.,  88.,  93.,
        97.,  98.,  99., 101., 103., 113., 120., 126.,  nan])
```

In [23]:
```
df["Annual Income (k$)"].describe()
```

Out[23]:
```
count    198.000000
mean      59.787879
std       25.237259
min       15.000000
25%       40.500000
50%       61.000000
75%       77.750000
max      126.000000
Name: Annual Income (k$), dtype: float64
```

In [24]:
```
df["Annual Income (k$)"] = df["Annual Income (k$)"].fillna(59.)
```

In [25]:
```
df.isna().sum()
```

Out[25]:
```
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

encoding

In [26]:
```
df = pd.get_dummies(df,columns = ["Gender"])                              # OneHotEncoding
```

In [27]:
```
df
```

Out[27]:

	Age	Annual Income (k$)	Spending Score (1-100)	Gender_Female	Gender_Male
0	19	15.0	39	0	1
1	21	15.0	81	0	1
2	20	16.0	6	1	0
3	23	16.0	77	1	0
4	31	17.0	40	1	0
...	...	...	...	...	...
195	35	120.0	79	1	0
196	45	126.0	28	1	0
197	32	126.0	74	0	1
198	32	59.0	18	0	1
199	30	59.0	83	0	1

200 rows × 5 columns

scaling

In [28]:
```
MinMax = MinMaxScaler()
```

In [29]:
```
MinMax.fit_transform(df)
```

Out[29]:
```
array([[0.01923077, 0.        , 0.3877551 , 0.        , 1.        ],
       [0.05769231, 0.        , 0.81632653, 0.        , 1.        ],
       [0.03846154, 0.00900901, 0.05102041, 1.        , 0.        ],
       [0.09615385, 0.00900901, 0.7755102 , 1.        , 0.        ],
       [0.25      , 0.01801802, 0.39795918, 1.        , 0.        ],
       [0.07692308, 0.01801802, 0.76530612, 1.        , 0.        ],
       [0.32692308, 0.02702703, 0.05102041, 1.        , 0.        ],
       [0.09615385, 0.02702703, 0.94897959, 1.        , 0.        ],
       [0.88461538, 0.03603604, 0.02040816, 0.        , 1.        ],
       [0.23076923, 0.03603604, 0.7244898 , 1.        , 0.        ],
       [0.94230769, 0.03603604, 0.13265306, 0.        , 1.        ],
       [0.32692308, 0.03603604, 1.        , 1.        , 0.        ],
       [0.76923077, 0.04504505, 0.14285714, 1.        , 0.        ],
       [0.11538462, 0.04504505, 0.7755102 , 1.        , 0.        ],
       [0.36538462, 0.04504505, 0.12244898, 0.        , 1.        ],
       [0.07692308, 0.04504505, 0.79591837, 0.        , 1.        ],
       [0.32692308, 0.05405405, 0.34693878, 1.        , 0.        ],
       [0.03846154, 0.05405405, 0.66326531, 0.        , 1.        ],
       [0.65384615, 0.07207207, 0.28571429, 0.        , 1.        ],
```

Model Training

In [30]:
```
x = df
```

In [31]:
```
# PCA
```

In [32]:
```
pca = PCA(n_components=2).fit(x)
```

In [33]:
```
print(pca.components_)
```

```
[[ 2.28460339e-01 -9.13265887e-02 -9.69258851e-01 -1.14163494e-03
   1.14163494e-03]
 [ 2.73920310e-02  9.95798954e-01 -8.73694087e-02 -5.93911284e-04
   5.93911284e-04]]
```

In [34]:
```
print(pca.explained_variance_)
```

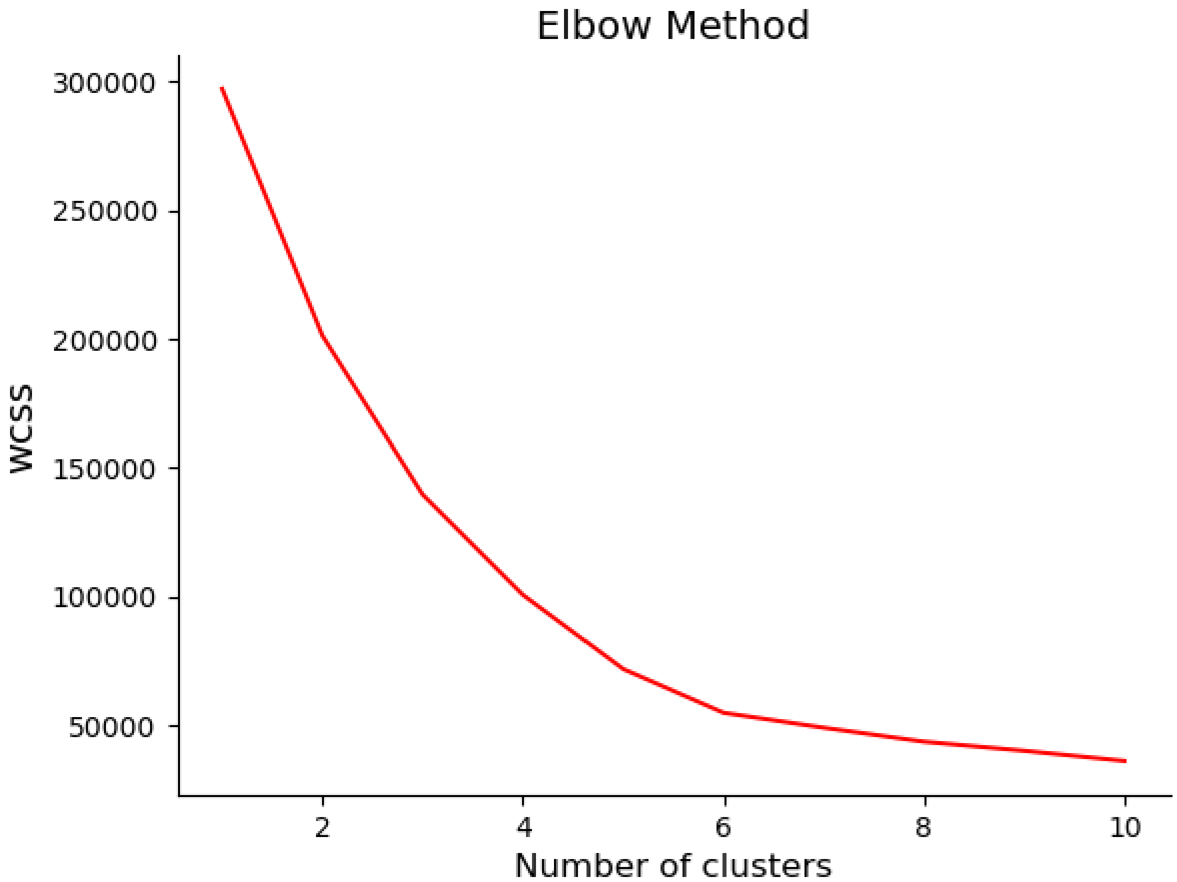
```
[695.28953822 630.0000616 ]
```

In [35]:
```
# Transform samples using the PCA fit
pca_2d = pca.transform(x)
```

Kmeans Clustering

In [36]:

```
wcss = []
for i in range(1,11):
    km = KMeans(n_clusters=i,init='k-means++', max_iter=300, n_init=10, random_state=0)
    km.fit(x)
    wcss.append(km.inertia_)
plt.plot(range(1,11),wcss, c="#FF0000")
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)
plt.title('Elbow Method', size=14)
plt.xlabel('Number of clusters', size=12)
plt.ylabel('wcss', size=14)
plt.show()
```

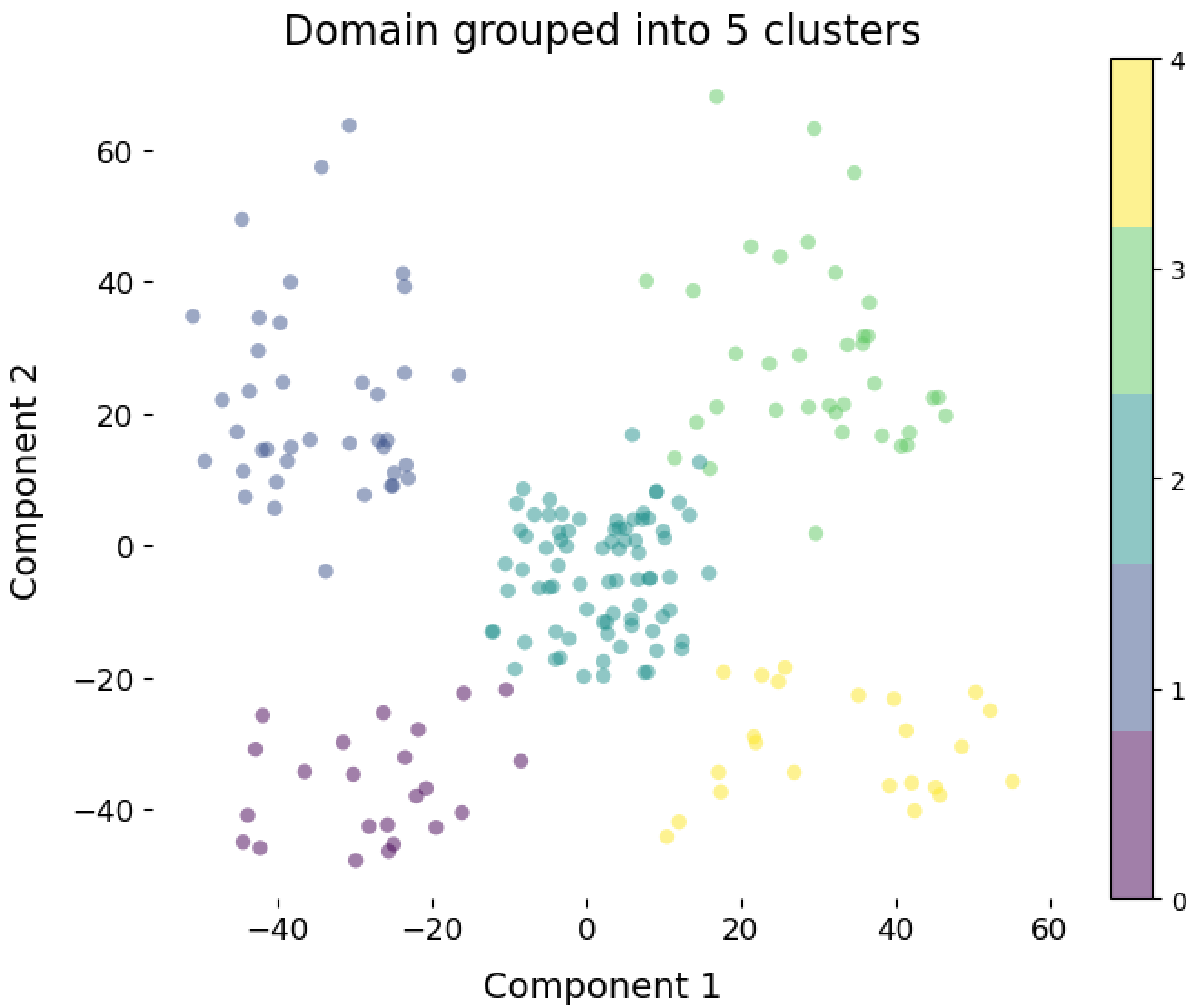



```
Kmeans algorithm
n_clusters: Number of clusters. In our case 5
init: k-means++. Smart initialization
max_iter: Maximum number of iterations of the k-means algorithm for a single run
n_init: Number of time the k-means algorithm will be run with different centroid seeds.
random_state: Determines random number generation for centroid initialization.
```

In [37]:
```
kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=10, n_init=10, random_state=0)
```

In [38]:
```
# Fit and predict
y_means = kmeans.fit_predict(x)
```

In [39]:
```
fig, ax = plt.subplots(figsize = (8, 6))

plt.scatter(pca_2d[:, 0], pca_2d[:, 1],
            c=y_means,
            edgecolor="none",
            cmap=plt.cm.get_cmap("viridis", 5),
            alpha=0.5)

plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)
plt.gca().spines["bottom"].set_visible(False)
plt.gca().spines["left"].set_visible(False)

plt.xticks(size=12)
plt.yticks(size=12)

plt.xlabel("Component 1", size = 14, labelpad=10)
plt.ylabel("Component 2", size = 14, labelpad=10)

plt.title('Domain grouped into 5 clusters', size=16)


plt.colorbar(ticks=[0, 1, 2, 3, 4]);

plt.show()
```



Model Evaluation

In [40]:
```
# Calculate Silhouette Score
```

In [41]:
```
silhouette_avg = silhouette_score(x,y_means)
```

In [42]:
```
print("The average silhouette_score is :", silhouette_avg)
```

```
The average silhouette_score is : 0.4436313586082965
```

The silhouette score is a metric used to evaluate the quality of clustering in a dataset. It measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette score ranges from -1 to 1. A score close to +1 indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. A score around 0 indicates that the object is on or very close to the decision boundary between two neighboring clusters.

In my case, a silhouette score of 0.441 is relatively high and close to 1, which suggests that the clustering is appropriate and the objects are well-matched to their own clusters compared to neighboring clusters. This indicates a good separation between clusters.

In [43]:
```
#wcss : within cluster sum of squares
kmeans.inertia_
```

Out[43]:
```
71736.45165554191
```

The within-cluster sum of squares (WCSS) is metric commonly used in clustering. It measures the compactness of the clusters. A lower WCSS indicates that the data points within each cluster are closer to each other, which typically implies better clustering performance.

```
In my case, a within-cluster sum of squares of 71736.45165554191 suggests that the clusters formed by the
algorithm have relatively low dispersion, meaning the data points within each cluster are close to each
other. This is generally desirable in clustering, as it indicates that the algorithm has successfully
grouped similar data points together.
```

centroids

In [44]:
```
centroids = pd.DataFrame(kmeans.cluster_centers_, columns = ["Age", "Annual Income", "Spending", "Male", "Fema
```

In [45]:
```
centroids.index_name = "ClusterID"
```

In [46]:
```
centroids["ClusterID"] = centroids.index
centroids = centroids.reset_index(drop=True)
```

In [47]:
```
centroids
```

Out[47]:

	Age	Annual Income	Spending	Male	Female	ClusterID
0	25.521739	26.304348	78.565217	0.608696	0.391304	0
1	32.692308	84.538462	82.128205	0.538462	0.461538	1
2	43.088608	55.291139	49.569620	0.582278	0.417722	2
3	40.666667	85.583333	17.583333	0.472222	0.527778	3
4	45.217391	26.304348	20.913043	0.608696	0.391304	4

In [48]:
```
X_new = np.array([[43, 76, 56, 0, 1]])

new_customer = kmeans.predict(X_new)
print(f"The new customer belongs to segment {new_customer[0]}")
```

```
The new customer belongs to segment 2
```

further classification using hierarchical clustering

In [49]:
```
# Perform hierarchical clustering
modelAC = AgglomerativeClustering(n_clusters=5)                    # You can specify the numb
modelAC.fit(x)
```

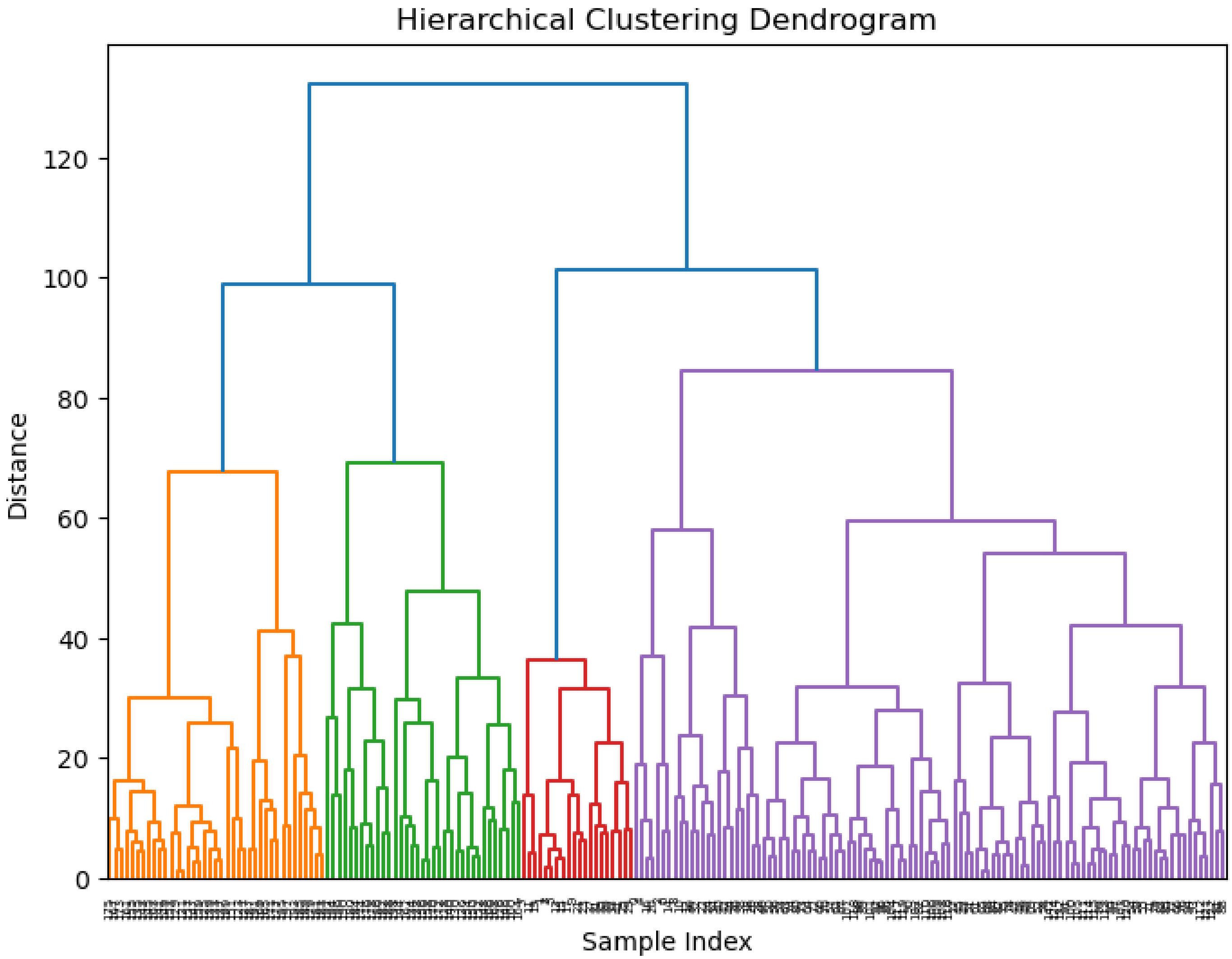
Out[49]:
```
AgglomerativeClustering(n_clusters=5)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [50]:
```
# Perform hierarchical clustering
Z = hierarchy.linkage(x, method='complete')                        # You can use different linka
```

In [51]:

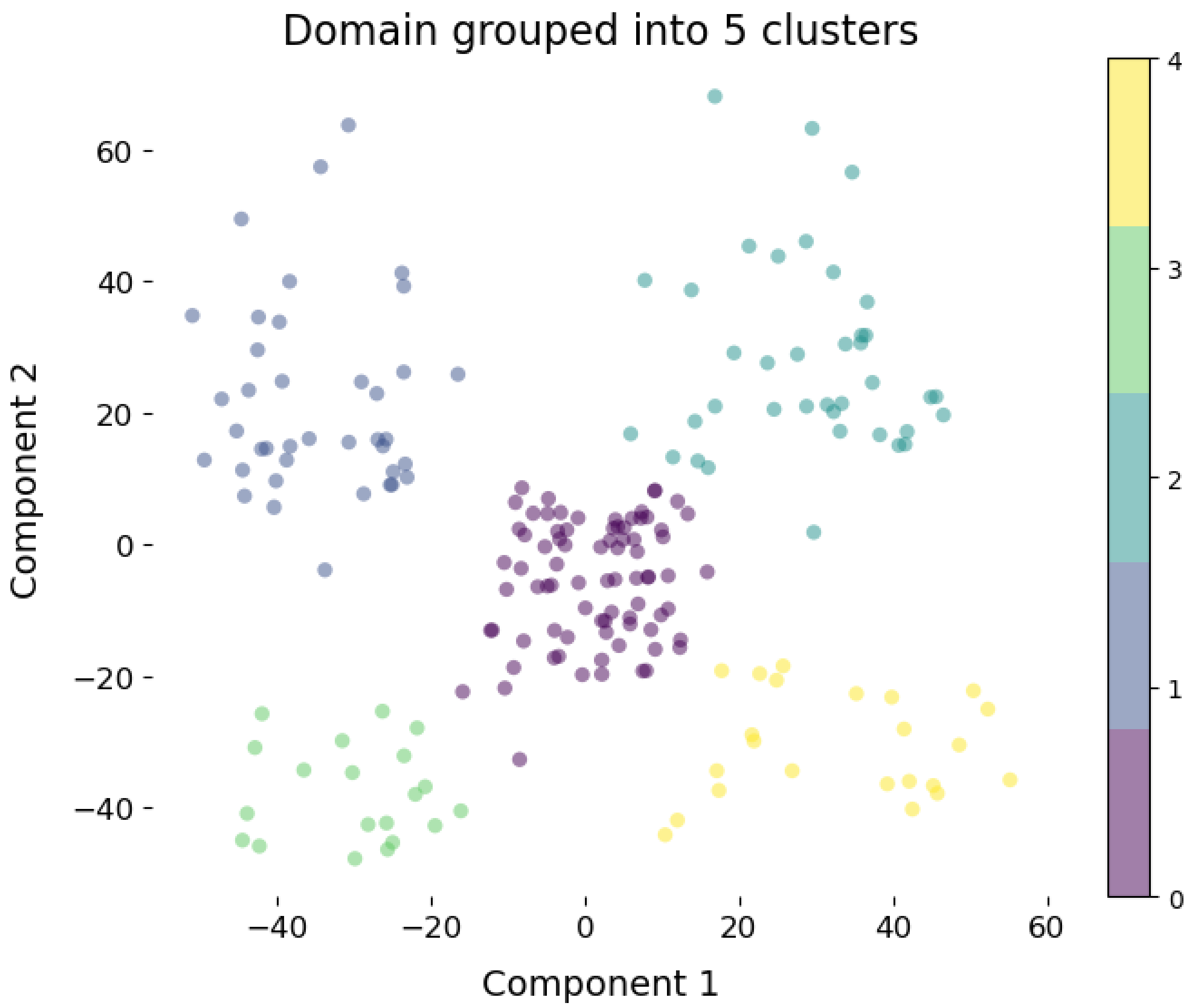
```
# Plot the dendrogram
plt.figure(figsize=(8, 6))
dn = hierarchy.dendrogram(Z)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```



In [52]:

```
# Fit and predict
y_means_AC = modelAC.fit_predict(x)
```

In [53]:
```
fig, ax = plt.subplots(figsize = (8, 6))

plt.scatter(pca_2d[:, 0], pca_2d[:, 1],
            c=y_means_AC,
            edgecolor="none",
            cmap=plt.cm.get_cmap("viridis", 5),
            alpha=0.5)

plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)
plt.gca().spines["bottom"].set_visible(False)
plt.gca().spines["left"].set_visible(False)

plt.xticks(size=12)
plt.yticks(size=12)

plt.xlabel("Component 1", size = 14, labelpad=10)
plt.ylabel("Component 2", size = 14, labelpad=10)

plt.title('Domain grouped into 5 clusters', size=16)


plt.colorbar(ticks=[0, 1, 2, 3, 4]);

plt.show()
```



In [54]:
```
# Calculate Silhouette Score
```

In [55]:

```
silhouette_avg_AC = silhouette_score(x,y_means_AC)
```

In [56]:

```
print("The average silhouette_score is :", silhouette_avg_AC)
```

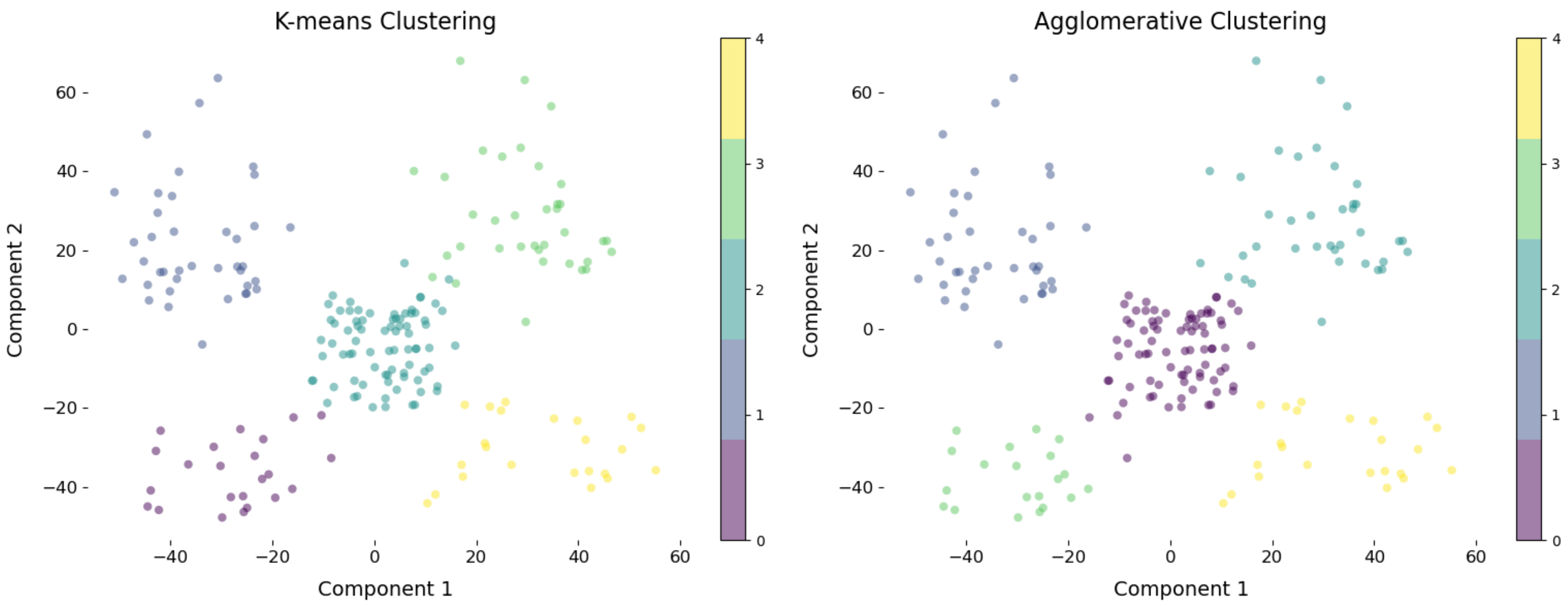
```
The average silhouette_score is : 0.44105474643115394
```

Comparision Between Kmeans And Agglomerative Clustering

In [57]:

```
import matplotlib.pyplot as plt

# Create figure and axes for subplots
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Scatterplot for K-means clustering
scatterplot_kmeans = axes[0].scatter(pca_2d[:, 0], pca_2d[:, 1],
                                     c=y_means,
                                     edgecolor="none",
                                     cmap=plt.cm.get_cmap("viridis", 5),
                                     alpha=0.5)
axes[0].set_title('K-means Clustering', size=16)
axes[0].set_xlabel("Component 1", size=14, labelpad=10)
axes[0].set_ylabel("Component 2", size=14, labelpad=10)
axes[0].spines["top"].set_visible(False)
axes[0].spines["right"].set_visible(False)
axes[0].spines["bottom"].set_visible(False)
axes[0].spines["left"].set_visible(False)
axes[0].tick_params(axis='both', which='major', labelsize=12)
fig.colorbar(scatterplot_kmeans, ax=axes[0], ticks=[0, 1, 2, 3, 4])

# Scatterplot for Agglomerative clustering
scatterplot_agglomerative = axes[1].scatter(pca_2d[:, 0], pca_2d[:, 1],
                                            c=y_means_AC,
                                            edgecolor="none",
                                            cmap=plt.cm.get_cmap("viridis", 5),
                                            alpha=0.5)
axes[1].set_title('Agglomerative Clustering', size=16)
axes[1].set_xlabel("Component 1", size=14, labelpad=10)
axes[1].set_ylabel("Component 2", size=14, labelpad=10)
axes[1].spines["top"].set_visible(False)
axes[1].spines["right"].set_visible(False)
axes[1].spines["bottom"].set_visible(False)
axes[1].spines["left"].set_visible(False)
axes[1].tick_params(axis='both', which='major', labelsize=12)
fig.colorbar(scatterplot_agglomerative, ax=axes[1], ticks=[0, 1, 2, 3, 4])

plt.tight_layout()
```

```
plt.show()
```



Conclusion

```
Initially, the process involved importing libraries and datasets, followed by EDA and feature engineering.
Feature extraction using PCA was then performed to enhance analytical clarity.

In the model training phase, the optimal number of clusters was determined using the elbow method. K-means
clustering was subsequently executed, with evaluation based on silhouette score and WCSS (inertia).
Centroids within each cluster were identified, and a demonstration was provided for clustering new
datapoints.
```

Additionally, hierarchical clustering was conducted and compared against K-means clustering. The results, assessed through visual inspection and evaluation metrics, indicated the formation of meaningful clusters suitable for analysis.