Unsupervised Learning

Tan Wen Tao Bryan
2214449
DAAA/FT/2A/01

Project Objective

- To build an **unsupervised machine learning model** to perform customer segmentation to **identify characteristics** of the clusters and **recommend** methods to **retain the most valuable** customer cluster

Background Info

Customer Segmentation

Customer segmentation is the process performed when discovering insights that define specific groupings of customers.
Why is customer segmentation important? Customer segmentation allows the marketers and companies to determine what campaigns, offers or products will attract specific groups of customers. It will not only focus on the short term value of a marketing action, but also the long term customer lifetime value (CLV) impact that a marketing action will bring.
Some benefits of Customer Segmentation:

- Improved Inventory Management
- Improved Customer Retention
- Create more specific sales and marketing strategies that cater to new and existing customer groups
- Stay competitive with larger retailers

Customer Insights that are Essential to Shopping Mall Marketing

The world is constantly changing. With new trends constantly taking place, customers traits, behaviours and expectations change too. By focusing on the customers that are visiting the mall, shopping malls can meet their demands by adapting to the current trend to bring in new customers and staying relevant to the existing ones in order to retain them.
These are examples of customer insights that are needed to provide customers with a personalized experience:

1. Demographics
   - Gender
   - Age
   - Ethnicity
   - Income
   - Marriage Status
   - Education
   - Occupation
   - Race
   - Religion
2. Psychographics
   - Interests
   - Hobbies
   - Lifestyles
   - Values
   - Attitudes & Beliefs
   - Pain points
   - Needs
   - Motivations & Goals
3. Geographics Location
   - Distance from home/office
   - Mode of transport
4. Behavioural Factors - Focus more on purchases and interactions rather than opinions and thoughts
   - Frequency of visit to the mall
   - Spending amount
   - Customer loyalty
   - Brand interactions (ex. Social Media, Newsletter)
   - Purchasing habits (ex. Online shopping, Physical shopping)

Step 1: EDA

Descriptive Statistics

```
#Make a copy to prevent mutation
customer_ds = customer_df.copy()

#Shape of dataset
print(customer_ds.shape)
```

```
(200, 5)
```

```
print(customer_ds.info())
```

```
<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   Income (k$)             200 non-null    int64
 4   How Much They Spend     200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
None
```

Observations:

- 200 rows and 5 columns
- 1 categorical column, 4 numerical columns
- All columns has an equal number of observations
- Customer_ID has a unique value for every row so it can be removed later
- No missing values in every row

```
#Descriptive Stats
customer_stats=customer_ds.describe(include="all").T
customer_stats["percentage of most freq value"] = customer_stats["freq"]/len(customer_ds)*100
display(customer_stats)
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max	percentage of most freq value
CustomerID	200.0	NaN	NaN	NaN	100.5	57.879185	1.0	50.75	100.5	150.25	200.0	NaN
Gender	200	2	Female	112	NaN	NaN	NaN	NaN	NaN	NaN	NaN	56.0
Age	200.0	NaN	NaN	NaN	38.85	13.969007	18.0	28.75	36.0	49.0	70.0	NaN
Income (k$)	200.0	NaN	NaN	NaN	60.56	26.264721	15.0	41.5	61.5	78.0	137.0	NaN
How Much They Spend	200.0	NaN	NaN	NaN	50.2	25.823522	1.0	34.75	50.0	73.0	99.0	NaN

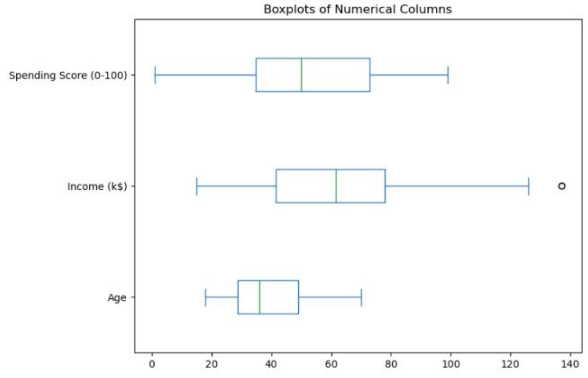
Observations:

- Dataset has more female than males, about 56% of the dataset.
- Average customer age in the dataset is in the late 30s (i.e. 38.85).
- Average customer annual income in the dataset is about 60k dollars.
- Average annual customer spending score is about 50/100.

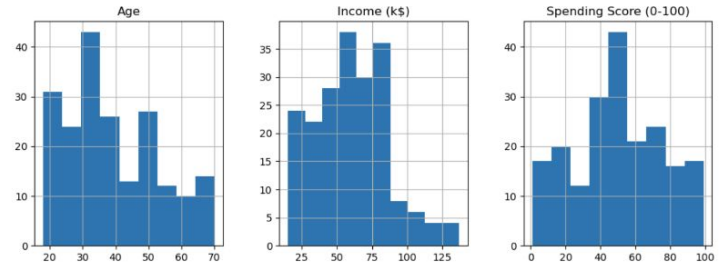
Distribution (Numerical)

```
numerical_var=[
    "Age",
    "Income (k$)",
    "Spending Score (0-100)"
]
```

```
#Plot boxplots to show numerical distribution
fig, ax = plt.subplots(figsize=(8,6))
customer_ds[numerical_var].plot.box(vert=False, ax=ax)
plt.title("Boxplots of Numerical Columns")
plt.show()
```




```
#Plot histograms to show numerical distribution
fig, ax = plt.subplots(1, 3, figsize=(12,4))
customer_ds[numerical_var].hist(bins="auto", ax=ax)
plt.show()
```

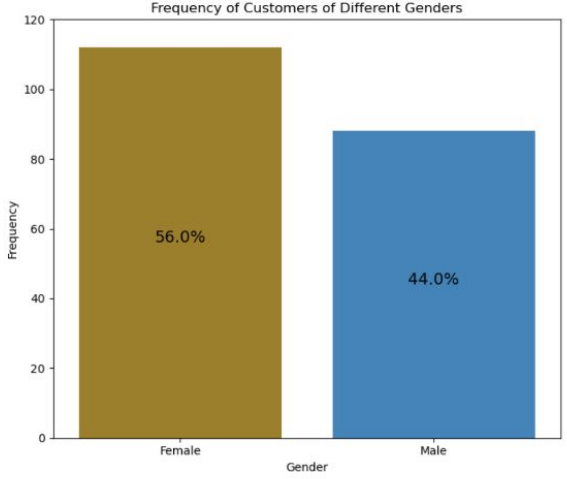


Observations:

- Spending Score (0-100) seems to have a normal distribution which is reasonable since it is a score that rates how much the customers spend in the mall.
- Age and Income seems to be slightly positively skewed.
- Most customers are around the late 30s.

Distribution (Categorical)

```
#Barplot to show frequency of gender
plt.figure(figsize=(7,6))
gender_counts = customer_ds["Gender"].value_counts()
gender_proportions = gender_counts/gender_counts.sum()*100

color_palette=["#b58900","#268bd2"]
sns.barplot(x=gender_counts.index, y=gender_counts, palette=color_palette)
plt.xlabel("Gender")
plt.ylabel("Frequency")
plt.title("Frequency of Customers of Different Genders")

#Add percentage annotations
for i, proportion in enumerate(gender_proportions):
    plt.annotate(f'{proportion:.1f}%',(i,proportion), ha='center', fontsize=14)

#Add y-axis
plt.gca().set_yticks([i for i in range(0,126,20)])
plt.gca().set_yticklabels([i for i in range(0,126,20)])
plt.tight_layout()
plt.show()
```

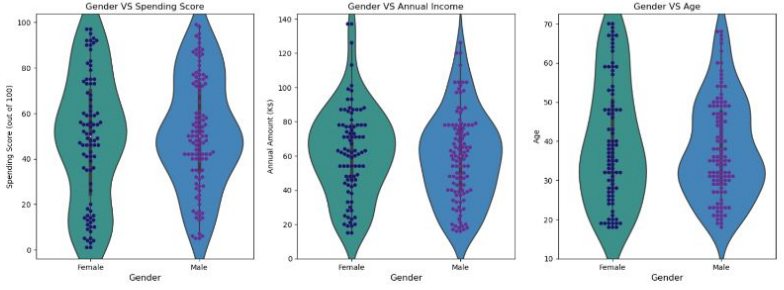


Observations:

- Female customers has a larger frequency than male customers in the dataset.

Step 1: EDA (Distribution)

```
#Violinplot to show distribution for each gender
fig,axes = plt.subplots(nrows=1, ncols=3, figsize=(18,6))
sns.violinplot(x="Gender", y="Spending Score (0-100)", data=customer_ds, palette=["#2aa198","#268bd2"], ax=axes[0])
sns.swarmplot(x="Gender", y="Spending Score (0-100)", data=customer_ds, palette=["midnightblue","rebeccapurple"], ax=axes[0])
sns.violinplot(x="Gender", y="Income (k$)", data= customer_ds, palette=["#2aa198","#268bd2"], ax=axes[1])
sns.swarmplot(x="Gender", y="Income (k$)", data= customer_ds, palette=["midnightblue","rebeccapurple"], ax=axes[1])
sns.violinplot(x="Gender", y="Age", data=customer_ds, palette=["#2aa198","#268bd2"], ax=axes[2])
sns.swarmplot(x="Gender", y="Age", data=customer_ds, palette=["midnightblue","rebeccapurple"], ax=axes[2])

axes[0].set_title('Gender VS Spending Score')
axes[0].set_xlabel('Gender', fontsize=12)
axes[0].set_xticklabels(["Female","Male"])
axes[0].set_ylabel('Spending Score (out of 100)')

axes[1].set_title('Gender VS Annual Income')
axes[1].set_xlabel('Gender', fontsize=12)
axes[1].set_xticklabels(["Female","Male"])
axes[1].set_ylabel('Annual Amount (K$)')
axes[1].set_yticks(np.arange(0,141,20))

axes[2].set_title('Gender VS Age')
axes[2].set_xlabel('Gender', fontsize=12)
axes[2].set_xticklabels(["Female","Male"])
axes[2].set_ylabel('Age')
axes[2].set_yticks(np.arange(10,71,10))

fig.suptitle("Distribution of Spending Score, Annual Income & Age for Each Gender", fontsize=14, fontweight="bold")
plt.show()
```

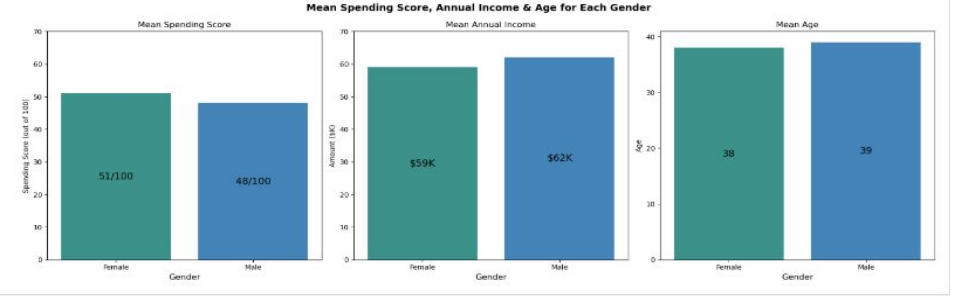


Observations:

- For Spending Score, most data points are clustered between 40 and 60 out of 100
- For Annual Income, most data points are clustered between 60 and 80
- For Age, most data points are clustered between 30 and 40
- Distribution between genders are roughly quite similar

```
#Barplot to show mean spending amount, annual income and age for each gender
spendingAmount_male=0
spendingAmount_female=0
income_male=0
income_female=0
age_male=0
age_female=0

#Calculate the total spending amount, total annual income and total age for each gender
for i in range(len(customer_ds)):
    if customer_ds["Gender"][i]=="Male":
        spendingAmount_male+=customer_ds["Spending Score (0-100)"][i]
        income_male+=customer_ds["Income (k$)"][i]
        age_male+=customer_ds["Age"][i]
    if customer_ds["Gender"][i]=="Female":
        spendingAmount_female+=customer_ds["Spending Score (0-100)"][i]
        income_female+=customer_ds["Income (k$)"][i]
        age_female+=customer_ds["Age"][i]

#Mean spending scores for both genders
genders_spendingAmount_mean=[int(spendingAmount_female/customer_ds["Gender"].value_counts()[0]),
                             int(spendingAmount_male/customer_ds["Gender"].value_counts()[1])]
genders_spendingAmount_meanSeries=pd.Series(data=genders_spendingAmount_mean)

#Mean annual income for both genders
genders_annualIncome_mean=[int(income_female/customer_ds["Gender"].value_counts()[0]),
                           int(income_male/customer_ds["Gender"].value_counts()[1])]
genders_annualIncome_meanSeries=pd.Series(data=genders_annualIncome_mean)

#Mean age for both genders
genders_age_mean=[int(age_female/customer_ds["Gender"].value_counts()[0]),
                  int(age_male/customer_ds["Gender"].value_counts()[1])]
genders_age_meanSeries=pd.Series(data=genders_age_mean)

fig,axes = plt.subplots(nrows=1, ncols=3, figsize=(18,6))
plots1=sns.barplot(x=["Female","Male"], y=genders_spendingAmount_meanSeries, palette=["#2aa198","#268bd2"], ax=axes[0])
plots2=sns.barplot(x=["Female","Male"], y=genders_annualIncome_meanSeries, palette=["#2aa198","#268bd2"], ax=axes[1])
plots3=sns.barplot(x=["Female","Male"], y=genders_age_meanSeries, palette=["#2aa198","#268bd2"], ax=axes[2])
```

```
for p in axes[0].patches:
    width, height = p.get_width(), p.get_height()
    x, y=p.get_xy()
    axes[0].annotate(f'{int(p.get_height())}/100',
                     (x+width/2, y+height/2), ha='center', va='center', fontsize=14)
axes[0].set_title('Mean Spending Score')
axes[0].set_xlabel('Gender', fontsize=12)
axes[0].set_ylabel('Spending Score (out of 100)')
axes[0].set_yticks(np.arange(0,71,10))

for p in axes[1].patches:
    width, height = p.get_width(), p.get_height()
    x, y=p.get_xy()
    axes[1].annotate(f'${int(p.get_height())}K',
                     (x+width/2, y+height/2), ha='center', va='center', fontsize=14)
axes[1].set_title('Mean Annual Income')
axes[1].set_xlabel('Gender', fontsize=12)
axes[1].set_ylabel('Amount ($K)')
axes[1].set_yticks(np.arange(0,71,10))

for p in axes[2].patches:
    width, height = p.get_width(), p.get_height()
    x, y=p.get_xy()
    axes[2].annotate(f'{int(p.get_height())}',
                     (x+width/2, y+height/2), ha='center', va='center', fontsize=14)
axes[2].set_title('Mean Age')
axes[2].set_xlabel('Gender', fontsize=12)
axes[2].set_ylabel('Age')
axes[2].set_yticks(np.arange(0,41,10))

fig.suptitle("Mean Spending Score, Annual Income & Age for Each Gender", fontsize=14, fontweight="bold")
plt.tight_layout()
plt.show()
```



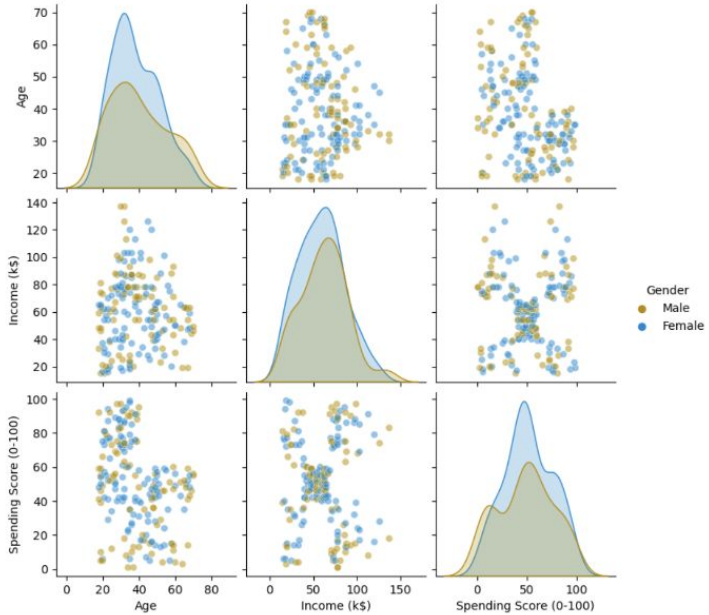
Observations:

- Mean spending score for female customers is a little higher than male customers
- Mean annual income for male customers is a little higher than female customers
- Mean age for male and females are mostly in the late 30s
- Difference between the mean spending score, the mean annual income and mean age for each gender is not very big

Step 1: EDA (Bivariate)

Bivariate Relationship

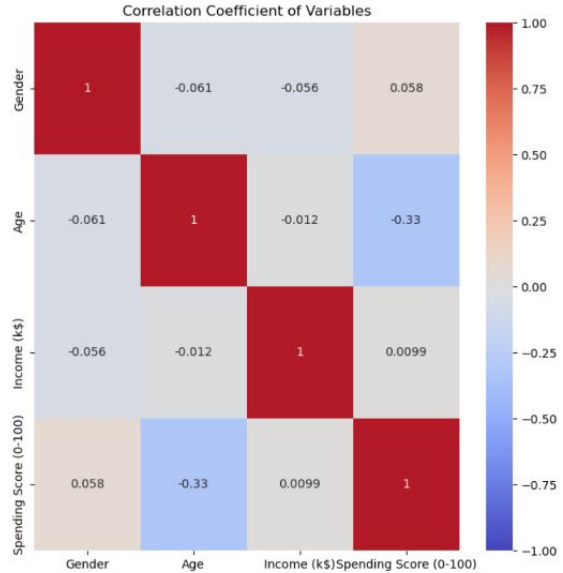
```
#Pairplot of variables
color_palette=["#b58900","#268bd2"]
sns.pairplot(data=customer_ds, hue="Gender", plot_kws={"alpha":0.5}, palette=color_palette)
plt.show()
```



Observations:

- Clear clusters were shown for the correlation between income and spending score (out of 100)
- Other pairs of variables show that no clear clusters were formed

```
#Heatmap to show correlation coefficient of the variables
plt.figure(figsize=(8,8))
sns.heatmap(data=customer_ds.corr(), annot=True, cmap="coolwarm", vmin=-1)
plt.title("Correlation Coefficient of Variables")
plt.show()
```



Observations:

- Spending Amount (How Much They Spend) & Age has a weak negative correlation of -0.33, which can imply that older customers might spend lesser than younger customers.
- Other features have an extremely weak correlation, which can imply that these othe features are independent of one another.

Step 1: EDA (Hopkins Statistics)

Hopkins Statistics

Used to assess the clustering tendency of a data set by measuring the probability that a given data set is generated by uniform data distribution.

H_0: Dataset is uniformly distributed and has no meaningful clusters
H_1: Dataset is not uniformly distributed and contains meaningful clusters

$$H = \frac{\sum_{j=1}^{m} ujd_j}{\sum_{j=1}^{m} ujd_j + \sum_{j=1}^{m} wjd_j}$$

```
#Hopkins Statistics
def hopkins(X):
    d = X.shape[1] #number of features in the dataset
    n = len(X) #number of rows
    #no. of randomly selected points for comparison with uniformly distributed points
    m = int(0.1 * n) #10% of the total number of data points

    #find nearest neighbour distances for both the original data points and uniformly distributed random points
    nbrs = NearestNeighbors(n_neighbors=1).fit(X.values)
    rand_X = sample(range(0, n, 1), m)\

    #original data points
    ujd = []
    #uniformly distributed data points
    wjd = []
    for j in range(0, m):
        u_dist, _ = nbrs.kneighbors(uniform(np.amin(X,axis=0),np.amax(X,axis=0),d).reshape(1, -1), 2, return_distance=True)
        ujd.append(u_dist[0][1])
    w_dist, _ = nbrs.kneighbors(X.iloc[rand_X[j]].values.reshape(1, -1), 2, return_distance=True)
    wjd.append(w_dist[0][1])

    H = sum(ujd) / (sum(ujd) + sum(wjd))

    #if H evaluates a NaN, it is set to 0
    if isnan(H):
        print(ujd, wjd)
        H = 0
    return H

hopkins(customer_ds)
```

```
0.9853421015180737
```

Observations:

- Since Hopkins Statistic is greater than 0.5 and very close to 1, this suggest that customer_ds has significantly clusterable data, containing meaningful data.

Step 2: Data Cleaning/Feature Engineering

Drop Unwanted Columns & Rename Columns

- Drop Customer ID as it is unique for every row
- Rename How Much They Spend to Spending Score (out of 100) as we prove that it is a score and not the amount

```
customer_df.drop("CustomerID", axis=1, inplace=True)
customer_df = customer_df.rename(columns={"How Much They Spend":"Spending Score (0-100)"})
#Create a copy of customer_df for 3d visualisation
customer_original = customer_df.copy()
display(customer_df.head())
```

	Gender	Age	Income (k$)	Spending Score (0-100)
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

Categorical Encoding

- Needs to encode categorical data as numerical data so that the data can be processed by the model

One-Hot Encoding

- Used for Gender column
- Meant for nominal data

```
#Encode for Gender
enc1 = OneHotEncoder(sparse_output=False, categories="auto")
gender_ds = enc1.fit_transform(customer_df[["Gender"]].values.reshape(-1,1))
gender_ds = pd.DataFrame(gender_ds, columns=["Female", "Male"]).drop("Male",axis=1)
customer_df[["Gender"]]=gender_ds.values
customer_encode = customer_df.copy()
customer_df.head()
```

	Gender	Age	Income (k$)	Spending Score (0-100)
0	0.0	19	15	39
1	0.0	21	15	81
2	1.0	20	16	6
3	1.0	23	16	77
4	1.0	31	17	40

Standardization

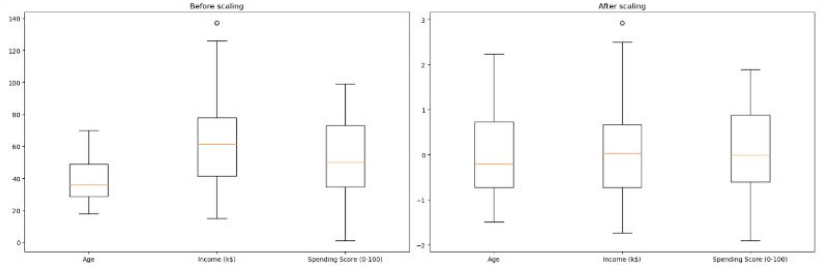
- Using StandardScaler to standardize the scale of the data so that the mean=0 and standard deviation=1 makes data consistent for model to use due to having different magnitude and units

```
#Standardize all numerical columns
num_cols=["Age","Income (k$)","Spending Score (0-100)"]
customer_df[num_cols]=StandardScaler().fit_transform(customer_df[num_cols])
customer_df
```

	Gender	Age	Income (k$)	Spending Score (0-100)
0	0.0	-1.424569	-1.738999	-0.434801
1	0.0	-1.281035	-1.738999	1.195704
2	1.0	-1.352802	-1.700830	-1.715913
3	1.0	-1.137502	-1.700830	1.040418
4	1.0	-0.563369	-1.662660	-0.395980
...	...	...	...	...
195	1.0	-0.276302	2.268791	1.118061
196	1.0	0.441365	2.497807	-0.861839
197	0.0	-0.491602	2.497807	0.923953
198	0.0	-0.491602	2.917671	-1.250054
199	0.0	-0.635135	2.917671	1.273347

200 rows × 4 columns

```
#Visualisation to show that numerical variables are scaled
fig,ax=plt.subplots(1,2, figsize=(18,6), tight_layout=True)
ax[0].boxplot(customer_ds[num_cols])
ax[0].set_title("Before scaling")
ax[1].boxplot(customer_df[num_cols])
ax[1].set_title("After scaling")
ax[0].set_xticks([1,2,3], num_cols)
ax[1].set_xticks([1,2,3], num_cols)
plt.show()
```



Step 2: Feature Engineering (Dimension Reduction)

Dimension Reduction

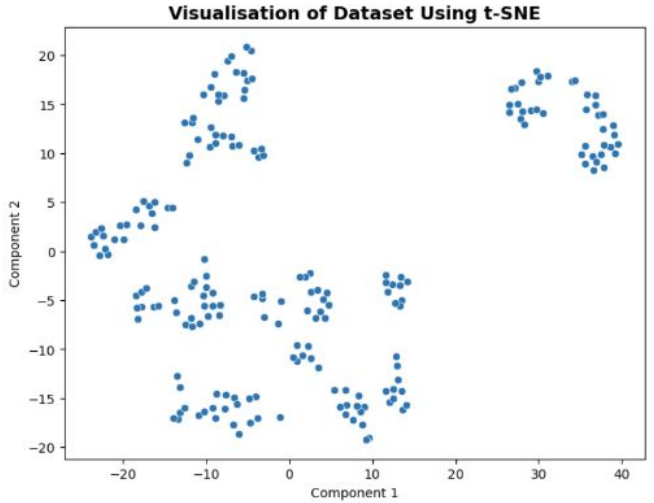
Transformation of data from a high-dimensional space into a low-dimensional space such that there is not much loss of information
Methods used for dimension reduction:

1. PCA (Principal Component Analysis)
   - PCA is used for dataset with high dimension
   - **Unable to use** as dataset is small with only 5 columns and 200 rows
2. t-SNE (t-distributed Stochastic Neighbor Embedding)
   - Tool to visualise high- dimensional data in 2 dimensions
   - Able to account for non-linear relationships
   - Uses local approaches: maps nearby points on higher dimensions to nearby points on lower dimensions also
   - Uses global approaches: attempt to preserve geometry at all scales so keeping nearby points close together while keeping far away points away from each other

- Only used to visualise the data clusters in **2D for the ease of visualisation**

```
optimal_perplexity = round(np.sqrt(customer_df.shape[0]))
tsne = TSNE(learning_rate=50, perplexity=optimal_perplexity, random_state=111, n_components=2)
tsne_df = tsne.fit_transform(customer_df)
```

```
#Apply t-SNE as a form of visualisation
plt.figure(figsize=(8,6))
sns.scatterplot(x=tsne_df[:, 0], y=tsne_df[:,1])
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.title("Visualisation of Dataset Using t-SNE", fontsize=14, fontweight="bold")
plt.show()
```



Observations:

- Using t-SNE shows clear clusters that can be formed from the dataset, hence I will use t-SNE to visualise how each unsupervised algorithm cluster the dataset

Step 3: Model Selection & Evaluation

The following **clustering algorithms** are used:

- **K-Means Clustering** (Centroid-based)
- **Agglomerative Clustering** (Hierarchical-based)
- **Gaussian Mixture Model Clustering** (Distribution-based)
- **Affinity Propagation** (Message Passing)
- **DBSCAN** (Density-based)

Step 3: K-Means

K-Means Clustering

Important Parameters:

init='k-means++'

- specifies a procedure to initialize the centroids before proceeding with standard k-means algorithm.

Procedures for k-means++:

1. First cluster is chosen at random.
2. Distance of each data point from the centroid that has already been selected is computed using nearest mean.
3. New centroid is chosen that has the maximum probability of being proportional to the distance.
4. Steps 2 and 3 are repeated until k clusters have been chosen.

Selecting the optimal number of k

- Elbow Method (Inertia)
- Silhouette Analysis

Elbow Method (Inertia)

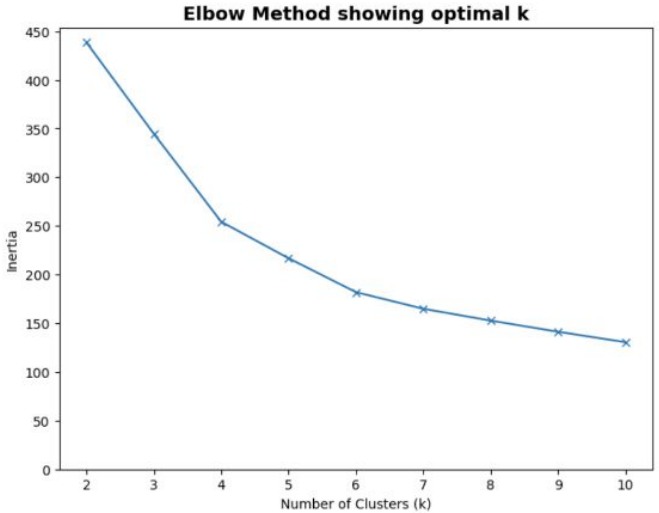
- Inertia is the sum of the squared distances to the closest cluster center
- Number has to be as small as possible
- If k is equal to the number of samples, inertia = 0

$$Inertia = \sum_{i=1}^{N}(x_i - C_k)^2$$

- N = number of samples within the dataset
- C_k = centroid of kth cluster
- x_i = ith data point

```
#Store the inertia for clusters between 2 to 11
inertia = []
for k in range(2,11):
    kmeans=KMeans(n_clusters=k, init='k-means++', random_state=111).fit(customer_df)
    inertia.append(kmeans.inertia_)
```

```
#Plot an elbow method to find the optimal number of clusters
plt.figure(figsize=(8,6))
plt.plot(range(2,11), inertia, 'x-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method showing optimal k', fontsize=14, fontweight="bold")
plt.yticks(np.arange(0,451,50))
plt.show()
```



Observations:

- Cluster value is where the inertia value stops decreasing at a large rate and remain constant.
- Therefore, optimal number of clusters for K-Means is 6.

Step 3: K-Means

Silhouette Analysis

- Measures how dense and well separated the clusters are
- Higher Silhouette Coefficient Score means that the model has better-defined clusters
- Ranges from -1 and 1

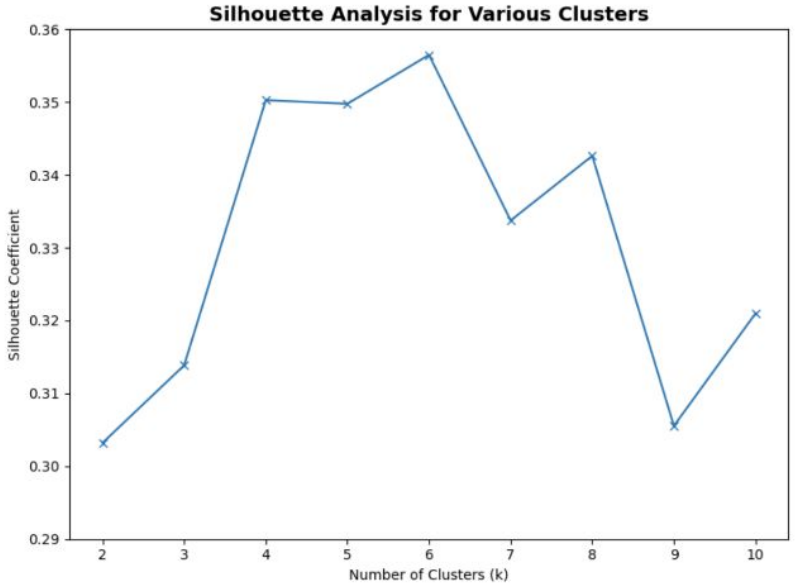
$$s = \frac{b - a}{max(a, b)}$$

- a = mean distance between a sample and all other points in the same class
- b = mean distance between a sample and all other points in the next nearest cluster

```
: #Get a silhouette coefficient for every cluster to get the most optimal number of clusters
  silhouette_scores=[]
  centroid=[]
  for k in range(2,11):
      kmeans=KMeans(n_clusters=k, init='k-means++', random_state=111).fit(customer_df)
      label = kmeans.labels_
      centroid.append(kmeans.cluster_centers_)
      sil_coef=silhouette_score(customer_df, label, metric='euclidean')
      silhouette_scores.append(sil_coef)
      print("For n_clusters={}, The Silhouette Coefficient is {:.3f}".format(k, sil_coef))
```

```
For n_clusters=2, The Silhouette Coefficient is 0.303
For n_clusters=3, The Silhouette Coefficient is 0.314
For n_clusters=4, The Silhouette Coefficient is 0.350
For n_clusters=5, The Silhouette Coefficient is 0.350
For n_clusters=6, The Silhouette Coefficient is 0.356
For n_clusters=7, The Silhouette Coefficient is 0.334
For n_clusters=8, The Silhouette Coefficient is 0.343
For n_clusters=9, The Silhouette Coefficient is 0.306
For n_clusters=10, The Silhouette Coefficient is 0.321
```

```
#Plot a line graph as a form of visualisation
plt.figure(figsize=(8,6))
plt.plot(range(2,11), silhouette_scores, "x-")
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Coefficient')
plt.title('Silhouette Analysis for Various Clusters', fontsize=14, fontweight="bold")
plt.yticks(np.arange(0.29, 0.361,0.01))

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



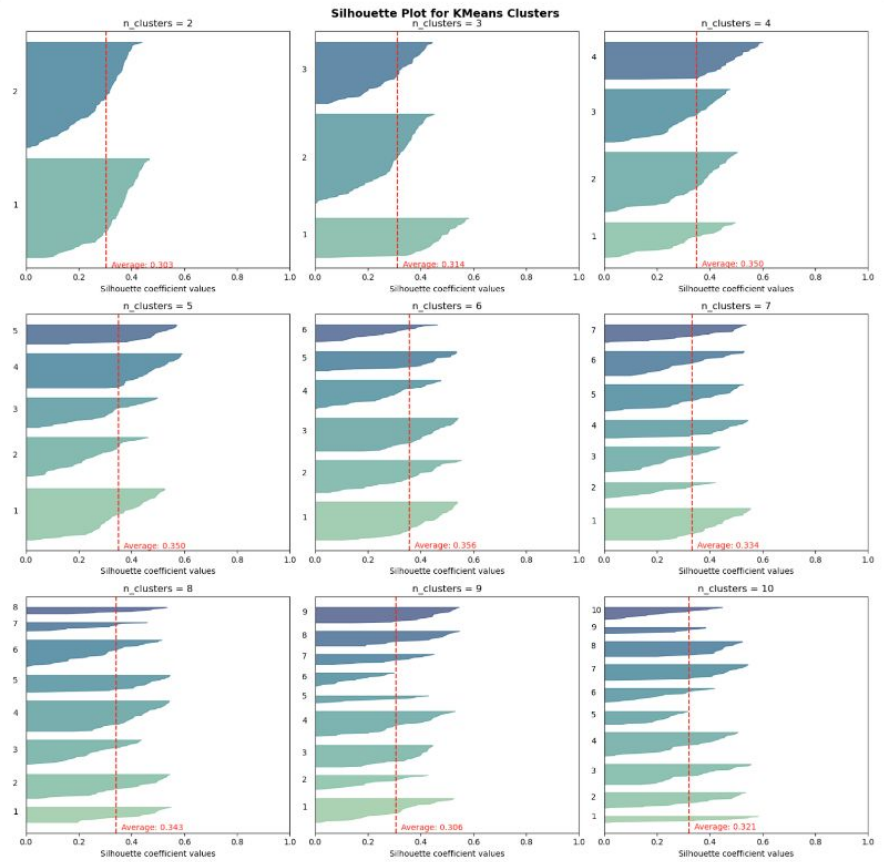
Observations:

- Highest silhouette score for KMeans is 0.356
- Most optimal number of clusters is 6

Step 3: K-Means

```
#Plot silhouette plots, code from sklearn documentation
def plot_silhouettePlot(clusterer, X, n_clusters, ax = None):
    if ax is None:
        fig, ax = plt.subplots(figsize=(10,5))

    ax.set_title(
    f"n_clusters = {max(n_clusters)}")

    #Compute the silhouette scores for each sample
    cluster_labels = clusterer.fit_predict(X)
    #Color list
    color_list = list(sns.color_palette("crest", len(n_clusters)+1).as_hex())
    # Compute the silhouette scores for each sample
    silhouette_avg = silhouette_score(X, cluster_labels)
    sample_silhouette_values = silhouette_samples(X, cluster_labels)
    ax.set_xlim([0, 1])

    y_lower = 10
    for i in range(max(n_clusters)):
        #Aggregate the silhouette scor es for samples belonging to cluster i and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = color_list[i]
        ax.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values, facecolor=color,
                         edgecolor=color, alpha=0.7)

        #Label the silhouette plots with their cluster numbers at the middle
        ax.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i+1))

        #Compute the new y_lower for next plot
        y_lower = y_upper + 10

    #The vertical line for average silhouette score of all the values
    ax.axvline(x=silhouette_avg, color="red", linestyle="--")
    ax.text(silhouette_avg + 0.02, 0, f"Average: {silhouette_avg:.3f}", color="red")
    ax.set_yticks([])
    ax.set_xticks([0, 0.2, 0.4, 0.6, 0.8, 1])
    ax.set_xlabel("Silhouette coefficient values")
```

```
#Creating silhouette plot for kmeans
cluster_range = [i for i in range(2, 11)]
fig, ax = plt.subplots(3,3, figsize=(15, 15))
fig.suptitle("Silhouette Plot for KMeans Clusters", fontsize=14, fontweight="bold")
for i in cluster_range:
    clusterer = KMeans(n_clusters=i, init='k-means++', random_state=111)
    plot_silhouettePlot(clusterer, customer_df, cluster_range[:i-1], ax=ax[(i-2)//3, (i-2)%3])
plt.tight_layout()
plt.show()
```



Observations:

- The silhouette plot shows which cluster label has a higher cohesion than the other labels for every n_cluster
- Red vertical line shows the average silhouette value for all clusters, n_clusters = 6 has the best silhouette score 0.356 and cluster label 2 has the highest cohesion

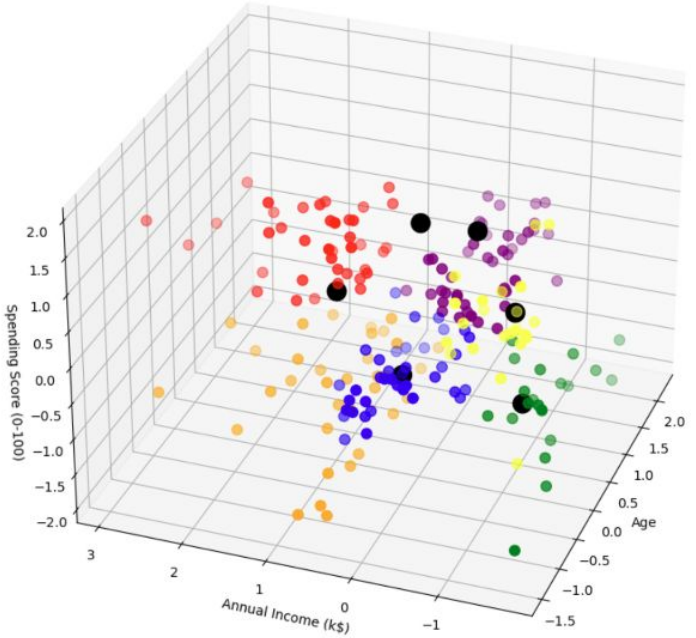
Step 3: K-Means

3D Scatterplot Visualisation

- Visualise the data points in 3D based on Age, Annual Income and Spending Score

```
#Initiate k=6
kmeans_new=KMeans(n_clusters=6, random_state=111, init='k-means++').fit(customer_df)

#Create a copy to include predicted cluster as a new column
kmeansclusters_new=customer_df.copy()
kmeansclusters_new["cluster_pred"]=kmeans_new.predict(customer_df)
kmeans_newCentroids=kmeans_new.cluster_centers_

#Reassign gender encoded values as categorical labels
gender={0:'Male', 1:"Female"}
kmeansclusters_new["Gender"]=kmeansclusters_new['Gender'].map(gender)
```

```
#Plot a 3D scatterplot to show how clusters are being formed in 3D
fig = plt.figure(figsize=(20,10))
ax = fig.add_subplot(111, projection='3d')

#Plot all data points in 3D for the 6 clusters
colors=['purple', 'blue', 'red', 'orange', 'yellow', 'green']
for i in range(6):
    ax.scatter(kmeansclusters_new["Age"][kmeansclusters_new["cluster_pred"]==i],
               kmeansclusters_new["Income (k$)"][kmeansclusters_new["cluster_pred"]==i],
               kmeansclusters_new["Spending Score (0-100)"][kmeansclusters_new["cluster_pred"]==i], c=colors[i], s=60)

#Add the centroids
ax.scatter(kmeans_newCentroids[:,0], kmeans_newCentroids[:,1], kmeans_newCentroids[:,2], s=200, c='black', alpha=1)
ax.view_init(30,200)
plt.xlabel("Age")
plt.ylabel("Annual Income (k$)")
ax.set_zlabel("Spending Score (0-100)")
plt.show()
```



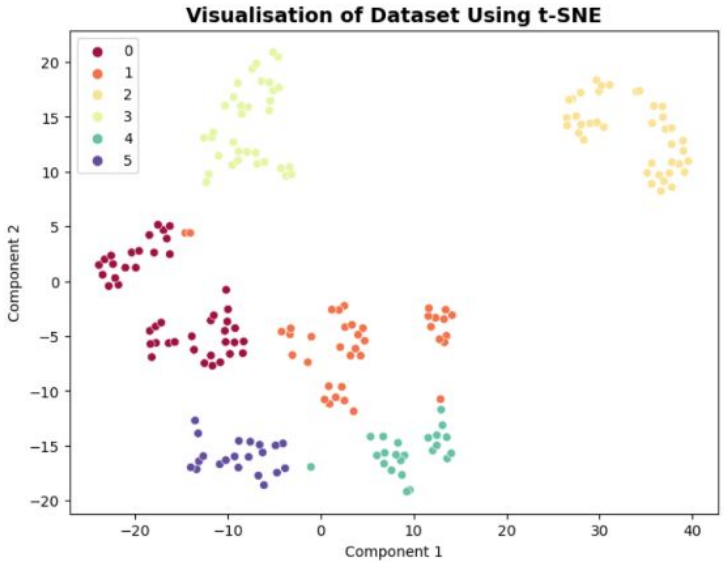
Observations:

- A 3D visualisation can look quite confusing so visualising the dataset in 2D will be much clearer.

Step 3: K-Means

2D t-SNE Visualisation

```
#Visualising dataset clustered by kmeans
plt.figure(figsize=(8,6))
kmeans_tsne=KMeans(n_clusters=6, random_state=111, init='k-means++')
kmeans_tsne.fit(customer_df)
sns.scatterplot(x=tsne_df[:, 0], y=tsne_df[:,1], hue= kmeans_tsne.labels_, palette='Spectral')
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.title("Visualisation of Dataset Using t-SNE", fontsize=14, fontweight="bold")
plt.show()
```



Observations:

- Clusters are pretty well formed.

Step 3: Agglomerative Clustering

Agglomerative Clustering

- Using the bottom-up approach, starts with many small clusters and merge them together to form bigger clusters
- Clusters are grouped together if the data points are similar

Formulas Used:

Ward Linkage Method

- Dendrogram uses Ward Linkage method which measures the distance between clusters (the sum of all squared distances within all clusters)
- Ward Method keep the growth of the clusters merge as small as possible

Euclidean Distance

- Length of a line segment between two points, calculated from the Cartesian coordinates of the points using the Pythagoras Theorem

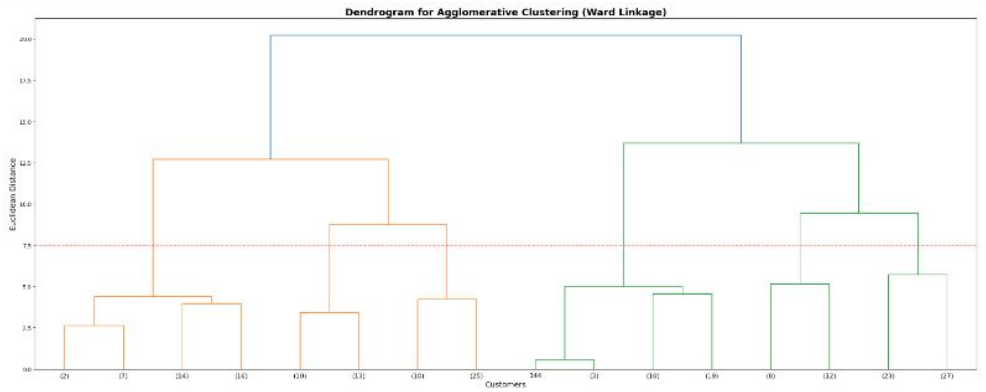
$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

- p, q = two points in Euclidean n-space
- q_i, p_i = Euclidean vectors, starting from the origin of the space (initial point)
- n = n-space

Dendrogram

- Explains the relationship between all data points in the dataset
- Construct it bottoms-up by merging individual data points and subclusters and go all the way to the top

```
def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # Create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)

#Setting distance_threshold=0 ensures we compute the full tree.
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)

model = model.fit(customer_df)
plt.figure(figsize=(25,10))
plt.title("Dendrogram for Agglomerative Clustering (Ward Linkage)", fontsize=18, fontweight="bold")

#Plot the top three levels of the dendrogram
plot_dendrogram(model, truncate_mode="level", p=3, show_leaf_counts=True)
plt.xlabel("Customers", fontsize=14)
plt.ylabel("Euclidean Distance", fontsize=14)

#Plot a horizontal line to show optimal number of clusters
plt.axhline(y=7.5, color='red', linestyle="--")
plt.tight_layout()
plt.show()
```



Observations:

- The number of vertical lines cut by the horizontal line gives us the optimal number of clusters which is 6.

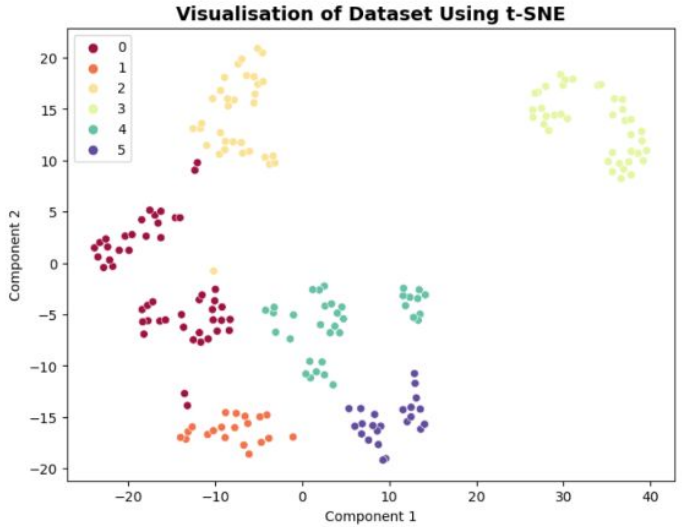
```
#Get a silhouette coefficient for every cluster to get the most optimal number of clusters
silhouette_scores=[]
for k in range(2,11):
    aggCluster=AgglomerativeClustering(n_clusters=k).fit(customer_df)
    label = aggCluster.labels_
    sil_coef=silhouette_score(customer_df, label, metric='euclidean')
    silhouette_scores.append(sil_coef)
    print("For n_clusters={}, The Silhouette Coefficient is {:.3f}".format(k, sil_coef))
```

```
For n_clusters=2, The Silhouette Coefficient is 0.292
For n_clusters=3, The Silhouette Coefficient is 0.310
For n_clusters=4, The Silhouette Coefficient is 0.330
For n_clusters=5, The Silhouette Coefficient is 0.348
For n_clusters=6, The Silhouette Coefficient is 0.350
For n_clusters=7, The Silhouette Coefficient is 0.315
For n_clusters=8, The Silhouette Coefficient is 0.325
For n_clusters=9, The Silhouette Coefficient is 0.323
For n_clusters=10, The Silhouette Coefficient is 0.325
```

Step 3: Agglomerative Clustering

2D t-SNE Visualisation

```
#Visualising dataset clustered by agglomerative clustering
plt.figure(figsize=(8,6))
agglo_tsne=AgglomerativeClustering(n_clusters=6)
agglo_tsne.fit(customer_df)
sns.scatterplot(x=tsne_df[:, 0], y=tsne_df[:,1], hue= agglo_tsne.labels_, palette='Spectral')
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.title("Visualisation of Dataset Using t-SNE", fontsize=14, fontweight="bold")
plt.show()
```



Observations:

- Clusters are pretty well-defined except 1 data point in (2)

Step 3: GMM

Gaussian Mixture Model

- Gaussian mixture models can handle very oblong clusters as they account for variance of the dataset, unlike K-Means
- Performs soft clustering where each sample has a probability to be associated with each cluster
- Estimate the parameters of these Gaussian components and assign the data points to the most likely component
- Made up of 2 steps
  - Step 1 - Expectation Step:
    - Initialize the parameters of the Gaussian components (means, covariances and mixing coefficients)
    - Using Gaussian density function, the responsibilities or probabilities of each data point belonging to each component
    - Update the responsibilities for each component, which reflect the likelihood of data point belonging to a specific componen
    - Posterior probability of each data point i belonging to cluster j:

$$r_{ij} = \pi_j \times N(x_i|\mu_j, \sum_j) / \sum_k \pi_k \times N(x_i|\mu_k, \sum_k)$$

      - r_{ij} : posterior probability
      - π_j : mixing coefficient of component j, weight parameter of cluster j
      - $N(x_i|\mu_j, \sum_j)$: Probability Density Function for data point x_i, given by the mean μ_j and covariance matrix $\sum_j$ of component j
  - Step 2 - Maximization Step:
    - Update the parameters of the Gaussian components based on the responsibilities obtained from Step 1
    - Calculate the new estimates for the means, covariances and mixing coefficients of the Gaussian components using the data points and the respective responsibilities
    - Normalize the mixing coefficients until they sum up to 1
    - Steps are repeated iteratively until convergence, where the parameters of the Gaussian components and the responsibilities stabilize
    - Convergence criteria can be based on changes in log likelihood or other predefined threshold

AIC (Akaike Information Criterian)

- Entails a calculation of maximum log-likelihood and a penalty term

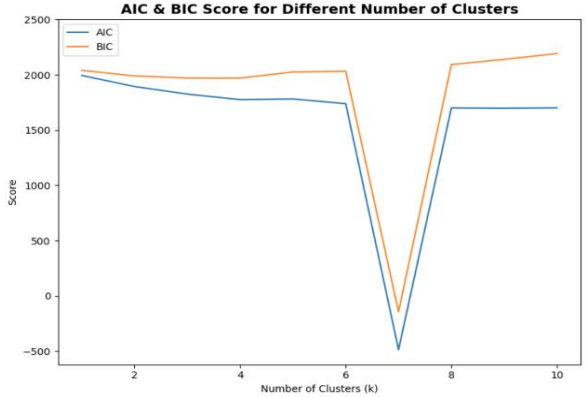
$$AIC_i = -2logL_i + 2p_i$$

BIC (Bayesian Information Criterian)

- Entails a calculation of maximum log-likelihood and a higher penalty term for a higher number of parameters

$$BIC_i = -2logL_i + p_i logn$$

- L_i: Max likelihood - parameter with the highest probability of correctly representing the relationship between input & output
- p: Number of parameters
- n: Number of values in the dataset
- The lower the value is, the better the model is




```
Number of Clusters (7) with the Lowest AIC: -485.981
Number of Clusters (7) with the Lowest BIC: -142.956
```

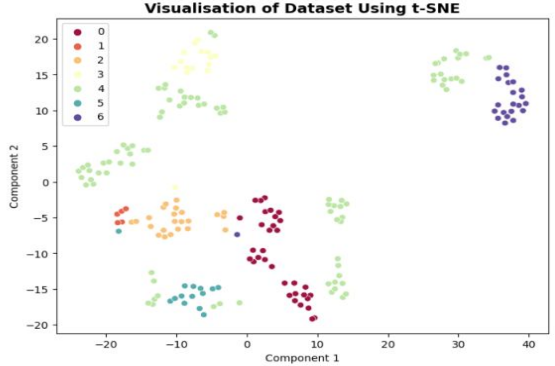
Observations:

- Ideal number of clusters with the lowest prediction errors are 7.

```
#Create empty dictionary for AIC & BIC values
aic_score={}
bic_score={}

#Loop through different number of clusters
for i in range(1,11):
    #Create GMM
    gmm = GaussianMixture(n_components=i, random_state=111).fit(customer_df)
    #AIC score
    aic_score[i] = gmm.aic(customer_df)
    #BIC score
    bic_score[i] = gmm.bic(customer_df)

#Number of clusters with lowest aic and bic score
min_aic_clusters = min(aic_score, key=aic_score.get)
min_bic_clusters = min(bic_score, key=bic_score.get)

#Visualisation
plt.figure(figsize=(8,6))
plt.plot(list(aic_score.keys()), list(aic_score.values()), label='AIC')
plt.plot(list(bic_score.keys()), list(bic_score.values()), label='BIC')
plt.legend(loc='best')
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Score")
plt.title("AIC & BIC Score for Different Number of Clusters", fontsize=14, fontweight="bold")
plt.yticks(np.arange(-500, 2501, 500))
plt.tight_layout()
plt.show()

print(f'Number of Clusters ({min_aic_clusters}) with the Lowest AIC: {min(aic_score.values()):.3f}')
print(f'Number of Clusters ({min_bic_clusters}) with the Lowest BIC: {min(bic_score.values()):.3f}')
```

2D t-SNE Visualisation

```
#Visualising dataset clustered by Gaussian Mixture Model
plt.figure(figsize=(8,6))
gmm_tsne=GaussianMixture(n_components=7, random_state=111)
gmm_tsne.fit(customer_df)
gmm_labels=gmm_tsne.predict(customer_df)
sns.scatterplot(x=tsne_df[:, 0], y=tsne_df[:,1], hue= gmm_labels, palette='Spectral')
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.title("Visualisation of Dataset Using t-SNE", fontsize=14, fontweight="bold")
plt.show()
```



Observations:

- Clusters are not really well-defined compared to k-means & Agglomerative Clustering hence GMM will not be good model to be used for this dataset.
- Reason why: The data points do not follow a Gaussian distribution (becos of Income & Gender) & they do not have an elliptical shape

Step 3: Affinity Propagation

Affinity Propagation

- Creates clusters by sending messages between data points until convergence
- Each data point sends messages to other points informing its targets of each target's relative attractiveness to the sender.
- Each target then responds to all senders with a reply informing each sender of its availability to associate with the sender, given the attractiveness of the messages that it has received from all other senders.
- Senders reply to the targets with messages informing each target of the target's revised relative attractiveness to the sender, given the availability messages it has received from all targets.
- The message-passing procedure proceeds until a consensus is reached.
- Once the sender is associated with one of its targets, that target becomes the point's exemplar.
- All points with the same exemplar are placed in the same cluster.

Uses 4 matrices to clusters:

- Similarity matrix
  - Contains values that corresponds to how similar two objects are
- Responsibility matrix
  - Contains values that corresponds to how responsible one object is for another
- Availabilty matrix
  - Contains values that corresponds to how available one object is to be an example for one another
- Criterion matrix
  - Each cell is the simply sum of availability matrix and responsibility matrix at that location
  - The highest criterion value of each row is then designated as an exemplar

Selecting the optimal number of clusters

Important Parameters:

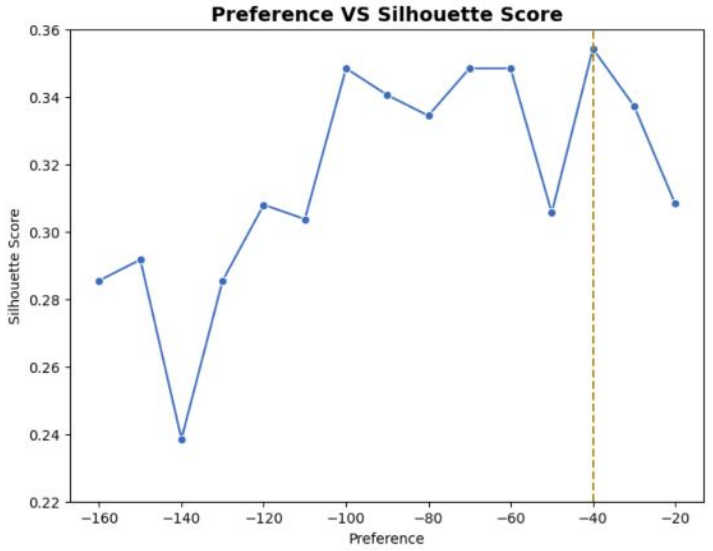
Preference

- Control the number of exemplars to be used

```
#Find the best preference score to get the best silhouette score
ap_cluster, ap_sil_score = [], []

#Set the range of preference values
pref = range(-160, -19, 10)

for p in pref:
    ap = AffinityPropagation(preference=p, random_state=111).fit(customer_df)
    ap_cluster.append(len(ap.cluster_centers_indices_))
    ap_sil_score.append(silhouette_score(customer_df, ap.labels_))

ap_df = pd.DataFrame()
ap_df['Preference'] = pref
ap_df["Number of Clusters"] = ap_cluster
ap_df["Silhouette Score"] = ap_sil_score
ap_df = ap_df.sort_values(by="Silhouette Score", ascending=False)
ap_df
```

	Preference	Number of Clusters	Silhouette Score
12	-40	6	0.354338
6	-100	4	0.348505
9	-70	4	0.348505
10	-60	4	0.348505
7	-90	4	0.340592
13	-30	6	0.337345
8	-80	4	0.334459
14	-20	8	0.308334
4	-120	3	0.308015
11	-50	6	0.305633
5	-110	3	0.303757
1	-150	2	0.291737
0	-160	2	0.285518
3	-130	2	0.285518
2	-140	2	0.238543

```
#Plot a graph (Preference VS Silhouette Score)
plt.figure(figsize=(8,6))
ax=plt.axes()
ax=sns.lineplot(x=ap_df["Preference"], y=ap_df["Silhouette Score"], marker="o", color="#316FBE")
plt.axvline(x=-40, color="#b58900", linestyle="--")
plt.xlabel("Preference")
plt.ylabel("Silhouette Score")
plt.yticks(np.arange(0.22, 0.37, 0.02))
plt.title("Preference VS Silhouette Score", fontweight="bold", fontsize=14)
plt.show()
```



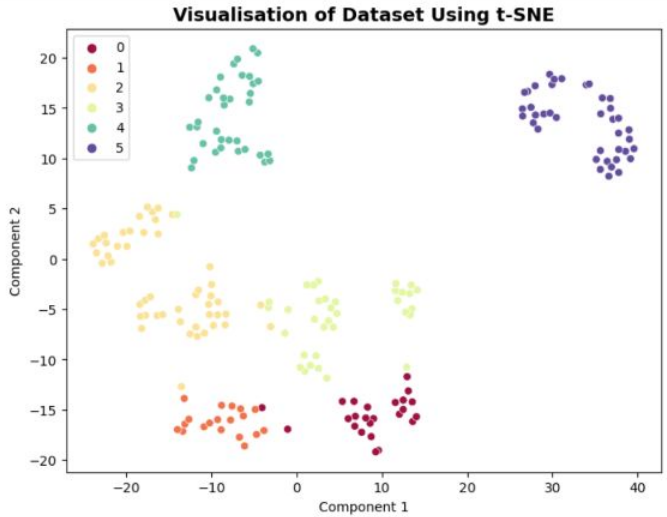
Observations:

- Best preference value to be used is -40 which results in the highest silhouette score of 0.35

Step 3: Affinity Propagation

2D t-SNE Visualisation

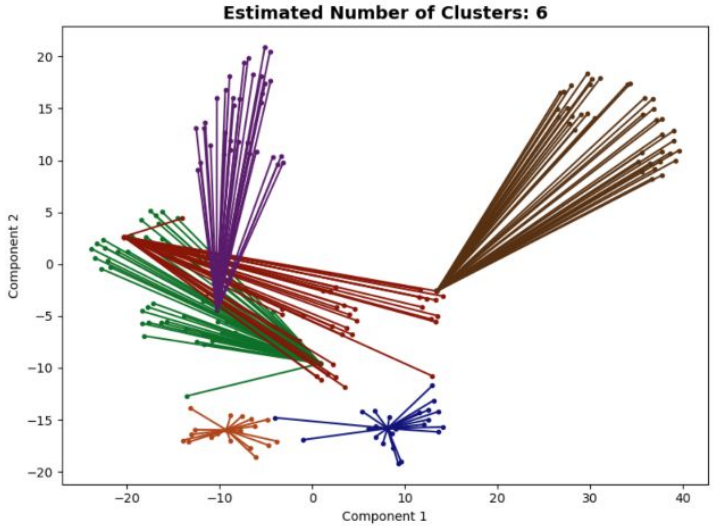
```
#Visualising dataset clustered in 2D
plt.figure(figsize=(8,6))
ap_tsne=AffinityPropagation(preference=-40, random_state=111)
ap_tsne.fit(customer_df)
ap_labels=ap_tsne.predict(customer_df)

sns.scatterplot(x=tsne_df[:, 0], y=tsne_df[:,1], hue= ap_labels, palette='Spectral')
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.title("Visualisation of Dataset Using t-SNE", fontsize=14, fontweight="bold")
plt.show()
```



Observations:

- The 6 clusters formed are quite clear and quite reasonable.

```
#Shows how exemplars are being formed
plt.figure(figsize=(8,6))
plt.clf()

ap_tsne=AffinityPropagation(preference=-40, random_state=111)
ap_tsne.fit(customer_df)
ap_labels=ap_tsne.predict(customer_df)
ap_clustersCentresIndices = ap.cluster_centers_indices_
ap_n_clusters = len(np.unique(ap_labels))

color_list = list(sns.color_palette("dark", (ap_n_clusters+1)).as_hex())

for k, col in zip(range(ap_n_clusters), colors):
    class_members = ap_labels == k
    cluster_center = tsne_df[ap_clustersCentresIndices[k]]
    plt.scatter(
        tsne_df[class_members, 0], tsne_df[class_members, 1], color=color_list[k], marker="."
    )
    plt.scatter(
        cluster_center[0], cluster_center[1], s=14, color=color_list[k], marker="o"
    )
    for x in tsne_df[class_members]:
        plt.plot([cluster_center[0], x[0]], [cluster_center[1], x[1]], color=color_list[k])

plt.title(f"Estimated Number of Clusters: {ap_n_clusters}", fontsize=14, fontweight="bold")
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.tight_layout()
plt.show()
```



Step 3: DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

- Able to find arbitrary shaped clusters and clusters with noise (i.e. outliers)
- Can separate clusters of high density from clusters of low density

Points are classified as core point, border point or outlier:

- Core Point
  - A point that has at least min_samples number of points in its surrounding area with radius eps
- Border Point
  - A point that is reachable from the core point and there are less than min_samples number of points within the surrounding area
- Noise
  - A point that is neither a core point or a border point like an outlier.

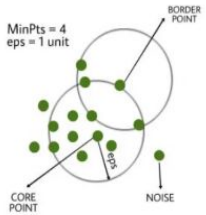


Image Source: Geeks For Geeks

How Clusters are Determined in DBSCAN

1. Divide the dataset into n dimensions
2. For each point in the dataset, DBSCAN forms an n dimensional shape around each data point and counts how many data points will fall within that shape.
3. DBSCAN counts this shape as a cluster. DBSCAN iteratively expands the cluster, by going through each individual point within the cluster, and counting the number of other data points nearby.

```
#Trying out default parameters
dbscan = DBSCAN().fit(customer_df)
dbscan_clusters = len(np.unique(dbscan.labels_))
dbscan_silScore = silhouette_score(customer_df, dbscan.labels_)
print(f'Silhouette Score (DBSCAN): {dbscan_silScore:.3f}')
```

```
Silhouette Score (DBSCAN): -0.011
```

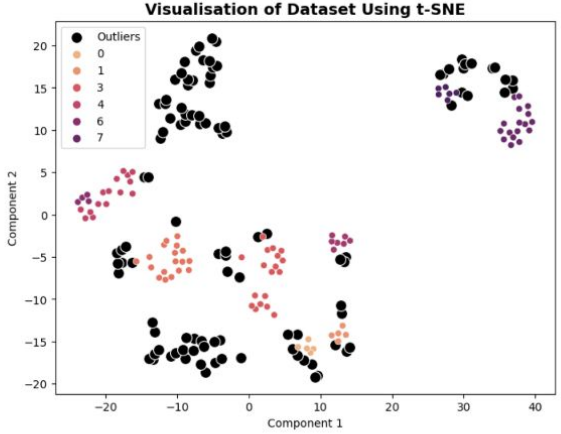
Observations:

- Based on the default parameters, the silhouette score shows that the coefficient is -0.011 which is negative. Change to the model parameters is required to increase the score.

2D t-SNE Visualisation (Default)

- First, we shall take a look at the scatterplot to see how clusters are being clustered based on the default parameters

```
#Visualising dataset in 2D
plt.figure(figsize=(8,6))
dbscan_tsne = DBSCAN().fit(customer_df)
dbscan_tsne.fit(customer_df)
dbscan_labels=dbscan_tsne.labels_
dbscan_clusters = len(np.unique(dbscan.labels_))

#Create a mask to plot the outliers in the DBSCAN scatterplot
mask = dbscan_labels == -1
updated_labels=[]
for label in dbscan_labels:
    if label != -1:
        updated_labels.append(label)

points_labelsExOutliers = tsne_df[~mask]
points_Outliers = tsne_df[mask]

sns.scatterplot(x=points_Outliers[:, 0], y=points_Outliers[:,1],
                label="Outliers", s=100, color='black')
sns.scatterplot(x=points_labelsExOutliers[:, 0], y=points_labelsExOutliers[:,1],
                hue=updated_labels, palette="flare")

plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.title("Visualisation of Dataset Using t-SNE", fontsize=14, fontweight="bold")
plt.legend(loc="best")
plt.show()
print(f"Estimated Number of Clusters: {dbscan_clusters}")
```




```
Estimated Number of Clusters: 10
```

Observations:

- As you can see in the graph, the data points that are considered as noise/outliers makes up quite a big number
- Clusters formed are not that reasonable which is why I will update the parameters to form reasonable clusters

Step 4: Model Improvement (DBSCAN Scaled)

DBSCAN

Important Parameters:

Eps

- Maximum distance between two samples for one to be considered as in the neighbourhood of the other. If the distance between two points is lower or equal to eps, they are considered neighbours.
- If the eps value is too small then large part of the data will be considered as outliers. If it is very large then the clusters will merge and majority of the data points will be in the same clusters.

Min_samples

- Minimum number of points in an Eps neighbourhood of that point.
- Larger dataset = Larger min_samples
- Derive number of min_samples from number of dimensions. ex. min_samples >= D+1

4) Model Improvement

- Some models shown here can be further improved by adjusting their hyperparameters
- We will then evaluate the models based on several score metrics:
  - Silhouette Score
  - Calinski Harabasz Score
  - Davies Bouldin Index
- Models that can no longer be improved in Model Selection & Evaluation stage:
  - **Agglomerative Clustering**
  - **Affinity Propagation**
  - **K-Means**

Scaled Customer Data

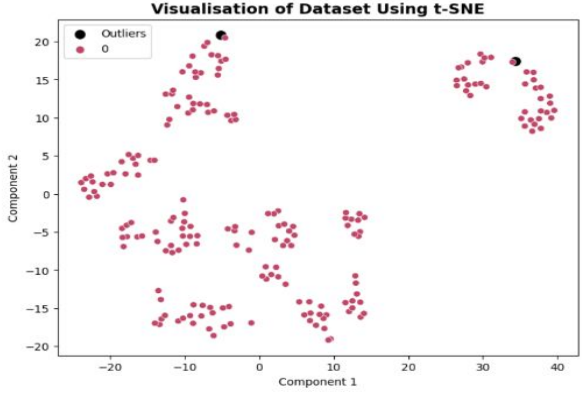
- Trying DBSCAN with the scaled customer data

```
#Trying out different values of the parameters to see whether it gives the best silhouette score
eps_vals = np.arange(0.5,1.2, 0.01)
min_samples_vals = np.arange(5,11)
param_list = list(product(eps_vals, min_samples_vals))

dbscan_clusters=[]
dbscan_silScore=[]

for param in param_list:
    dbscan = DBSCAN(eps = param[0], min_samples = param[1]).fit(customer_df)
    dbscan_clusters.append(len(np.unique(dbscan.labels_)))
    dbscan_silScore.append(silhouette_score(customer_df, dbscan.labels_))

dbscan_df = pd.DataFrame.from_records(param_list, columns=["Eps", "Min Samples"])
dbscan_df["No of Clusters"] = dbscan_clusters
dbscan_df["Silhouette Scores"] = dbscan_silScore
dbscan_df = dbscan_df.sort_values(by="Silhouette Scores", ascending=False)
display(dbscan_df.head(70))
```

	Eps	Min Samples	No of Clusters	Silhouette Scores
410	1.18	7	2	0.319870
409	1.18	6	2	0.319870
397	1.16	6	2	0.319870
416	1.19	7	2	0.319870
415	1.19	6	2	0.319870
...	...	...	...	...
314	1.02	7	2	0.257723
326	1.04	7	2	0.257723
313	1.02	6	2	0.257723
312	1.02	5	2	0.257723
324	1.04	5	2	0.257552

70 rows × 4 columns

```
#Visualising dataset in 2D
plt.figure(figsize=(8,6))
dbscan_tsne = DBSCAN(eps=1.18, min_samples=7).fit(customer_df)
dbscan_tsne.fit(customer_df)
dbscan_labels=dbscan_tsne.labels_
dbscan_clusters = len(np.unique(dbscan.labels_))

#Create a mask to plot the outliers in the DBSCAN scatterplot
mask = dbscan_labels == -1
updated_labels=[]
for label in dbscan_labels:
    if label != -1:
        updated_labels.append(label)

points_labelsExOutliers = tsne_df[~mask]
points_Outliers = tsne_df[mask]

sns.scatterplot(x=points_Outliers[:, 0], y=points_Outliers[:,1],
                label="Outliers", s=100, color='black')
sns.scatterplot(x=points_labelsExOutliers[:, 0], y=points_labelsExOutliers[:,1],
                hue=updated_labels, palette="flare")

plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.title("Visualisation of Dataset Using t-SNE", fontsize=14, fontweight="bold")
plt.legend(loc="best")
plt.show()
print(f"Estimated Number of Clusters: {dbscan_clusters}")
print(f'Silhouette Score: {silhouette_score(customer_df, dbscan_labels):.3f}')
```




```
Estimated Number of Clusters: 2
Silhouette Score: 0.320
```

Observations:

- Even though the silhouette score for DBSCAN is descently high, the clusters that are formed are not reasonable.
- Just 1 whole cluster and the outliers

Step 4: Model Improvement (Unscaled)

Unscaled Customer Data

- Trying DBSCAN with the unscaled customer data since working with scaled customer data was unsuccessful
- For DBSCAN, scaling the data is not necessary as it is a density-based clustering algorithm that is less sensitive to the scale of the features
- DBSCAN focus on the local density of points to form cluster, rather than the absolute values of the features

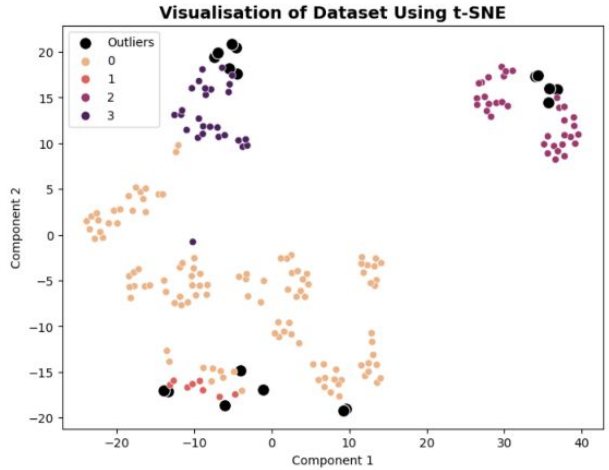
```
#Unscaled dataset that I will use DBSCAN
display(customer_encode.head())
```

	Gender	Age	Income (k$)	Spending Score (0-100)
0	0.0	19	15	39
1	0.0	21	15	81
2	1.0	20	16	6
3	1.0	23	16	77
4	1.0	31	17	40

```
#Trying out different values of the parameters to see whether it gives the best silhouette score
eps_vals = np.arange(10, 16)
min_samples_vals = np.arange(7, 14)
param_list = list(product(eps_vals, min_samples_vals))

dbscan_clusters=[]
dbscan_silScore=[]

for param in param_list:
    dbscan = DBSCAN(eps = param[0], min_samples = param[1]).fit(customer_encode)
    dbscan_clusters.append(len(np.unique(dbscan.labels_)))
    dbscan_silScore.append(silhouette_score(customer_encode, dbscan.labels_))

dbscan_df = pd.DataFrame.from_records(param_list, columns=["Eps", "Min Samples"])
dbscan_df["No of Clusters"] = dbscan_clusters
dbscan_df["Silhouette Scores"] = dbscan_silScore
dbscan_df = dbscan_df.sort_values(by="Silhouette Scores", ascending=False)
display(dbscan_df.head())
```

	Eps	Min Samples	No of Clusters	Silhouette Scores
35	15	7	5	0.289210
36	15	8	4	0.268895
41	15	13	4	0.265917
33	14	12	4	0.258949
29	14	8	4	0.258866

```
#Visualising dataset in 2D
plt.figure(figsize=(8,6))
dbscan_tsne = DBSCAN(eps=15, min_samples=7).fit(customer_encode)
dbscan_labels=dbscan_tsne.labels_
dbscan_clusters = len(np.unique(dbscan_tsne.labels_))

#Create a mask to plot the outliers in the DBSCAN scatterplot
mask = dbscan_labels == -1
updated_labels=[]
for label in dbscan_labels:
    if label != -1:
        updated_labels.append(label)

points_labelsExOutliers = tsne_df[~mask]
points_Outliers = tsne_df[mask]

sns.scatterplot(x=points_Outliers[:, 0], y=points_Outliers[:,1],
                label="Outliers", s=100, color='black')
sns.scatterplot(x=points_labelsExOutliers[:, 0], y=points_labelsExOutliers[:,1],
                hue=updated_labels, palette="flare")

plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.title("Visualisation of Dataset Using t-SNE", fontsize=14, fontweight="bold")
plt.legend(loc="best")
plt.show()
print(f"Estimated Number of Clusters: {dbscan_clusters}")
print(f'Silhouette Score (DBSCAN): {silhouette_score(customer_encode,dbscan_labels):.3f}')
```




```
Estimated Number of Clusters: 5
Silhouette Score (DBSCAN): 0.289
```

Observations:

- Other than the outliers that are formed, there are 4 clusters formed that are quite clear.
- Not as defined compared to the clusters of the other models

Step 4: Model Improvement (GMM)

GMM

Important Parameters:

n_components

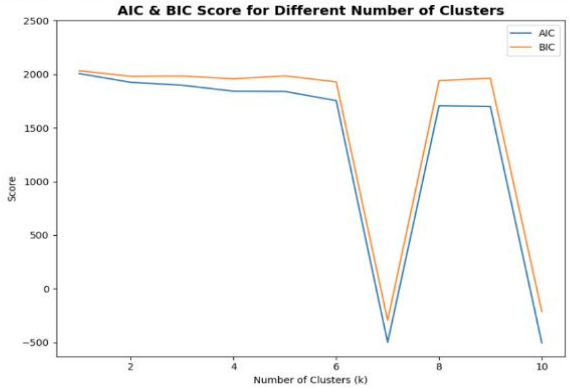
- Number of mixture components

covariance_type

- 4 types of covariance parameters
  1. full
     - Each component has its own general covariance matrix
  2. tied
     - All components share the same general covariance matrix
  3. diag
     - Each component has its own diagonal covariance matrix
  4. spherical
     - Each component has its own single variance

```
#Trying out different values of the parameters to see whether it gives the best silhouette score
n_components = np.arange(2,11)
covariance_type = ["full", "tied","diag", "spherical"]
param_list = list(product(n_components, covariance_type))

gmm_silScore=[]
gmm_aic=[]
gmm_bic=[]

for param in param_list:
    gmm = GaussianMixture(n_components=param[0], covariance_type=param[1], random_state=111).fit(customer_df)
    gmm_label=gmm.predict(customer_df)
    gmm_silScore.append(silhouette_score(customer_df, gmm_label))
    gmm_aic.append(gmm.aic(customer_df))
    gmm_bic.append(gmm.bic(customer_df))

gmm_df = pd.DataFrame.from_records(param_list, columns=["Number of Components", "Covariance Type"])
gmm_df["Silhouette Scores"] = gmm_silScore
gmm_df["BIC"] = gmm_bic
gmm_df["AIC"] = gmm_aic

gmm_df = gmm_df.sort_values(by=["BIC","AIC"], ascending=True)
display(gmm_df.head())
```

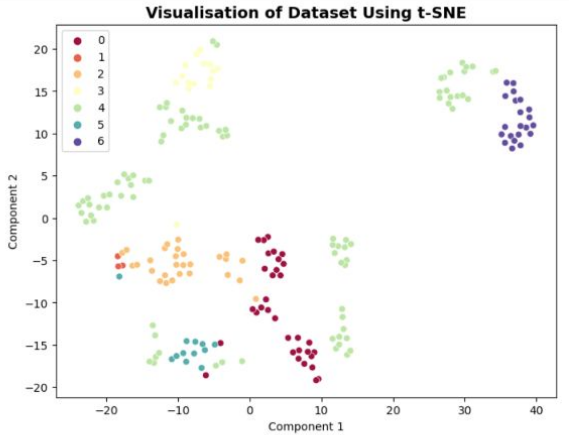
	Number of Components	Covariance Type	Silhouette Scores	BIC	AIC
22	7	diag	0.038593	-294.694562	-499.190258
34	10	diag	-0.033326	-209.796883	-503.347128
20	7	full	0.052778	-142.956203	-485.981209
27	8	spherical	0.315721	1918.812986	1763.792070
18	6	diag	0.316221	1928.810833	1754.000013




```
Number of Clusters (10) with the Lowest AIC: -503.347
Number of Clusters (7) with the Lowest BIC: -294.695
```

Observations:

- Clusters formed with lowest BIC after improving the scores is 7.
- BIC is used as it penalizes with a larger number of parameters more serverly.
- Best covariance type is diag



Observations:

- However, clusters formed are not extremely reasonable and defined compared to the other models.

Step 4: Model Evaluation (Score Metrics)

Model Evaluation

After improving the models, we will use a couple ways to evaluate the best 2 models to interpret the clusters.

1. Score Metrics

- Silhouette Score
  - Measures quality of clustering based on distances between data points within and between clusters
  - Assigns probability to each data point in different clusters
- Calinski Harabasz Score
  - Measures the sum of between-cluster dispersion against the sum of within-cluster dispersion, where dispersion is the sum of distance squared
  - Higher ratio means cluster is far away from the other clusters and that it is more defined

$$\text{Calinski Harabasz Score} = \frac{B}{W} \times \frac{N-k}{k-1}$$

  - B: Sum of squared distance between cluster centroids and overall centroid (Between cluster dispersion)
  - W: Sum of squared distance between the samples within each cluster and the respective cluster centroids (Within cluster dispersion)
  - N: total number of samples in the dataset
  - k: Number of clusters
- Davies-Bouldin Index
  - Measures the average similarity between clusters, where similarity compares the size of clusters against the between-cluster distance
  - Lower score means that the cluster is relatively small compared to the distance to another cluster, hence well defined

$$DBI = \frac{1}{k}\sum_{i=1}^{k}\max_{j\neq i}\left(\frac{S_i + S_j}{|c_i - c_j|}\right)$$

  - k: Number of clusters
  - S_i: Scatter or average dissimilarity within cluster i
  - c_i: Centroid of cluster i
  - $|c_i - c_j|$: Euclidean distance between centroids $c_i and c_j$

2. 2D-Visualisation (t-SNE)

- Using t-SNE data reduction method, clusters formed can be visualised to see whether the clusters formed make sense or not

Things to Note:

- These scores metrics values tend to be higher for Density-based models
- Silhouette Score & Calinski Harabasz Score does not work for GMM as GMM are probablistic models

Observations:

- These score metrics are supposed to have high values for density-based models like DBSCAN but the values are still lower than K-Means, Affinity Propagation & Agglomerative, which shows that DBSCAN is not suitable for this dataset
- K-Means is the best model according to the 3 score metrics which means that the clusters are the most dense and most well defined
- Affinity Propagation seems to be better than Agglomerative according to Calinski Harabasz but not as strong according to Davies Bouldin Index
- The clusters in Affinity Propagation are far away from the clusters than Agglomerative but clusters are less separated compared to Agglomerative

Score Metrics

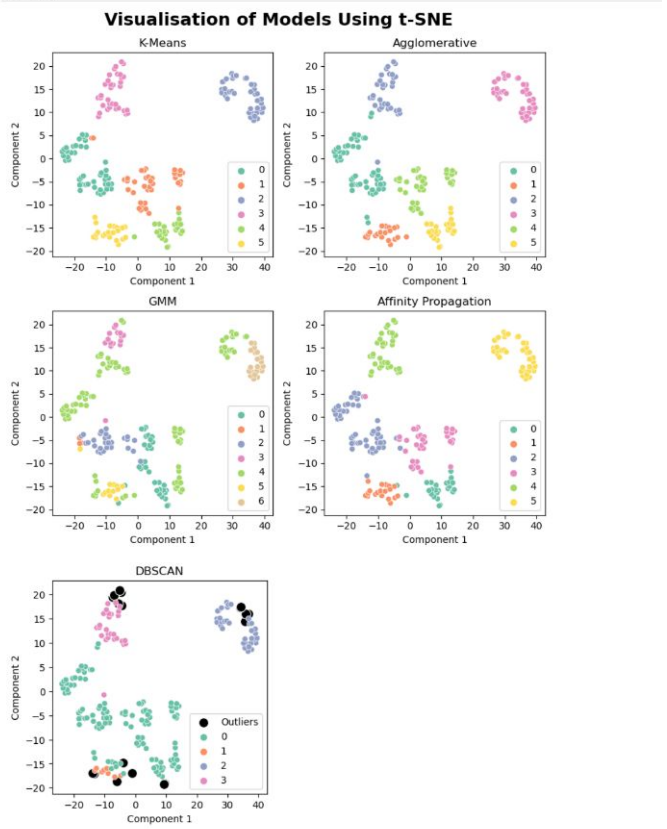
```
#K-Means
kmeans_new=KMeans(n_clusters=6, random_state=111, init='k-means++').fit(customer_df)
kmeans_label = kmeans_new.labels_
kmeans_silcoef=silhouette_score(customer_df, kmeans_label, metric='euclidean')
kmeans_ch = calinski_harabasz_score(customer_df, kmeans_label)
kmeans_dbi = davies_bouldin_score(customer_df, kmeans_label)

#Agglomerative Clustering
agglo_new = AgglomerativeClustering(n_clusters=6, linkage="ward").fit(customer_df)
agglo_label = agglo_new.labels_
agglo_silcoef=silhouette_score(customer_df, agglo_label, metric='euclidean')
agglo_ch = calinski_harabasz_score(customer_df, agglo_label)
agglo_dbi = davies_bouldin_score(customer_df, agglo_label)

#GMM
gmm_new=GaussianMixture(n_components=7,covariance_type="diag", random_state=111).fit(customer_df)
gmm_label=gmm_new.predict(customer_df)
gmm_silcoef=silhouette_score(customer_df, gmm_label, metric='euclidean')
gmm_ch = calinski_harabasz_score(customer_df, gmm_label)
gmm_dbi = davies_bouldin_score(customer_df, gmm_label)

#Affinity Propagation
ap_new = AffinityPropagation(preference=-40, random_state=111).fit(customer_df)
ap_label = ap_new.predict(customer_df)
ap_silcoef=silhouette_score(customer_df, ap_label, metric='euclidean')
ap_ch = calinski_harabasz_score(customer_df, ap_label)
ap_dbi = davies_bouldin_score(customer_df, ap_label)

#DBSCAN
dbscan_new = DBSCAN(eps=15, min_samples=7).fit(customer_encode)
dbscan_label=dbscan_new.labels_
dbscan_silcoef=silhouette_score(customer_encode, dbscan_label, metric='euclidean')
dbscan_ch = calinski_harabasz_score(customer_df, dbscan_label)
dbscan_dbi = davies_bouldin_score(customer_df, dbscan_label)

#Place the labels and clusters in a dataframe
silcoef_list = [kmeans_silcoef, agglo_silcoef, gmm_silcoef, ap_silcoef, dbscan_silcoef]
calinski_harabasz_list = [kmeans_ch, agglo_ch, gmm_ch, ap_ch, dbscan_ch]
davies_bouldin_list = [kmeans_dbi, agglo_dbi, gmm_dbi, ap_dbi, dbscan_dbi]
model_list = ["K-Means", "Agglomerative", "GMM", "Affinity Propagation", "DBSCAN"]
metrics_df = pd.DataFrame(data={"Silhouette Score":silcoef_list, "Calinski Harabasz": calinski_harabasz_list,
                                "Davies Bouldin Index": davies_bouldin_list},
                          index=model_list)
metrics_df = metrics_df.sort_values(by=["Silhouette Score","Calinski Harabasz", "Davies Bouldin Index"],
                                    ascending=[False, False, True])
metrics_df
```

	Silhouette Score	Calinski Harabasz	Davies Bouldin Index
K-Means	0.356486	99.654879	1.005090
Affinity Propagation	0.354338	98.424085	1.016217
Agglomerative	0.350444	95.257661	1.008615
DBSCAN	0.289210	26.629181	2.461337
GMM	0.038593	22.019003	1.486906

Step 4: Model Evaluation (t-SNE 2D Visualisation)

2D-Visualisation (t-SNE)

```
#Plot a subplot of the first 4 models (K-Means, Agglomerative, GMM, Affinity Propagation)
fig, ax = plt.subplots(2,2, figsize=(8,8))
ax=ax.flatten()
model_list = ["K-Means", "Agglomerative", "GMM", "Affinity Propagation", "DBSCAN"]
labels=[kmeans_label, agglo_label, gmm_label, ap_label, dbscan_label]

fig.suptitle("Visualisation of Models Using t-SNE", fontsize=18, fontweight="bold")
for i in range(0, 4):
    sns.scatterplot(x=tsne_df[:, 0], y=tsne_df[:,1], hue=labels[i], palette='Set2', ax=ax[i])
    ax[i].legend(loc="best")
    ax[i].set_xlabel("Component 1")
    ax[i].set_ylabel("Component 2")
    ax[i].set_title(model_list[i], fontsize=12)

plt.tight_layout()
plt.show()

#Plot 1 more plot for DBSCAN
fig2 = plt.figure(figsize=(4,4))

#Create a mask to plot the outliers in the DBSCAN scatterplot
mask = dbscan_label == -1
updated_labels=[]
for label in dbscan_label:
    if label != -1:
        updated_labels.append(label)

points_labelsExOutliers = tsne_df[~mask]
points_Outliers = tsne_df[mask]

sns.scatterplot(x=points_Outliers[:, 0], y=points_Outliers[:,1],
                label="Outliers", s=100, color='black')
sns.scatterplot(x=points_labelsExOutliers[:, 0], y=points_labelsExOutliers[:,1],
                hue=updated_labels, palette="Set2")
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.title("DBSCAN", fontsize=12)
plt.show()
```



Observations:

- According to visualisation, K-Means, Agglomerative & Affinity Propagation clusters seemed to be more well defined compared to GMM & DBSCAN

Step 5: Model Interpretation (Scatterplot)

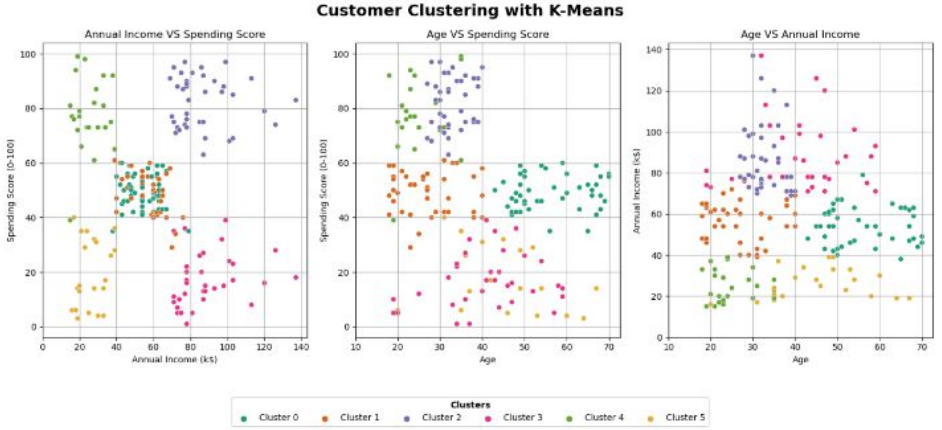
Cluster Intepretation

- Visualise the individual data points in 2D to identify characteristics of the clusters using the best two models
- K-Means & Affinity Propagation

K-Means Scatterplot Analysis

```
#Initiate K-Means
kmeans_new=KMeans(n_clusters=6, random_state=111, init='k-means++').fit(customer_df)

#Use original dataset to show the actual values to predict
customer_original["K-Means Clusters"]=kmeans_new.predict(customer_df)
labels = np.unique(customer_original["K-Means Clusters"])
display(customer_original.head())
```

	Gender	Age	Income (k$)	Spending Score (0-100)	K-Means Clusters
0	Male	19	15	39	4
1	Male	21	15	81	4
2	Female	20	16	6	5
3	Female	23	16	77	4
4	Female	31	17	40	5




```
#Add legend
legend_labels=[f"Cluster {label}" for label in labels]

#Plot Annual Income VS Spending Score
fig, ax=plt.subplots(1, 3, figsize=(15, 6))
scatter1=sns.scatterplot(x=customer_original["Income (k$)"], y=customer_original["Spending Score (0-100)"],
                         hue=customer_original["K-Means Clusters"], palette='Dark2', ax=ax[0])
ax[0].set_title("Annual Income VS Spending Score", fontsize=12)
ax[0].set_xlabel("Annual Income (k$)")
ax[0].set_ylabel("Spending Score (0-100)")
ax[0].grid(True)
ax[0].set_xticks(np.arange(0, 143, 20))

#Plot Age VS Spending Score
scatter2=sns.scatterplot(x=customer_original["Age"], y=customer_original["Spending Score (0-100)"],
                         hue=customer_original["K-Means Clusters"], palette='Dark2', ax=ax[1])
ax[1].set_title("Age VS Spending Score", fontsize=12)
ax[1].set_xlabel("Age")
ax[1].set_ylabel("Spending Score (0-100)")
ax[1].grid(True)
ax[1].set_xticks(np.arange(10, 75, 10))

#Plot Age VS Annual Income
scatter3=sns.scatterplot(x=customer_original["Age"], y=customer_original["Income (k$)"],
                         hue=customer_original["K-Means Clusters"], palette='Dark2', ax=ax[2])
ax[2].set_title("Age VS Annual Income", fontsize=12)
ax[2].set_xlabel("Age")
ax[2].set_ylabel("Annual Income (k$)")
ax[2].grid(True)
ax[2].set_xticks(np.arange(10, 75, 10))
ax[2].set_yticks(np.arange(0, 143, 20))

#Turn off respective legends in subplots
ax[0].legend_=None
ax[1].legend_=None
ax[2].legend_=None

#Create a shared legend
handles, labels = scatter1.get_legend_handles_labels()
legend = fig.legend(handles, legend_labels, loc='center', bbox_to_anchor=(0.5, -0.1), ncol=len(legend_labels))
legend.set_title("Clusters", prop={'weight':'bold'})

fig.suptitle("Customer Clustering with K-Means", fontsize=18, fontweight="bold")
plt.tight_layout()
plt.show()
```

Observations:

- **Cluster 0**: Customers aged above 40 who earns about 40k-70k annual income and have a spending score of 40-60
- **Cluster 1**: Customers aged 15-40 who earns about 40k-70k annual income and have a spending score of 40-60
- **Cluster 2**: Customers aged 25-40 who earns above 70k annual income and have a spending score above 60
- **Cluster 3**: Customers aged 20-60 who earns above 70k annual income and have a spending score below 40
- **Cluster 4**: Customers aged 15-35 who earns below 40k annual income and have a spending score above 60
- **Cluster 5**: Customers aged 30-70 who earns below 40k annual income and have a spending score below 40

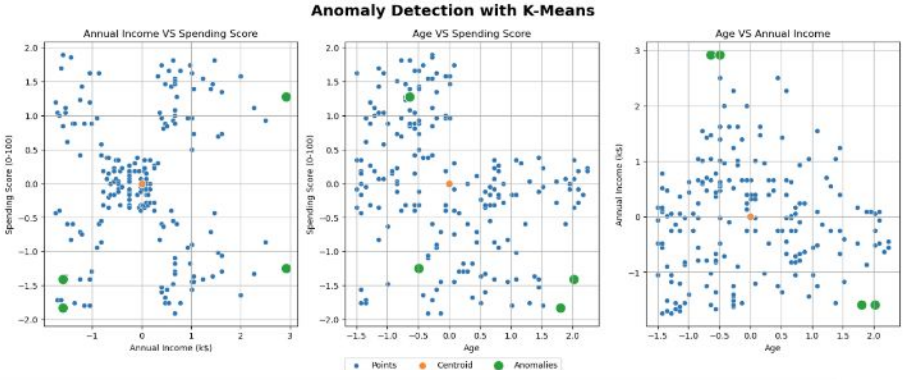
Step 5: Model Interpretation (Anomaly Detection)

Anomaly Detection (K-Means)

- Detect any particular anomalies in the dataset using 1 cluster for K-Means

```
#Get K-Means model with only 1 cluster
kmeans_anomaly=KMeans(n_clusters=1, random_state=111, init='k-means++').fit(customer_df)

#Get centroids (K-Means)
centers=kmeans_anomaly.cluster_centers_

#Compute distances
distances = kmeans_anomaly.transform(customer_df)

#Identify anomalies
sorted_idx = np.argsort(distances.ravel())[::-1][:4]
```

```
#Visualise the outliers in the clusters
fig, ax = plt.subplots(1, 3, figsize=(15, 6))
ax=ax.flatten()

#Plot Annual Income VS Spending Score
scatter1 = sns.scatterplot(x=customer_df["Income (k$)"], y=customer_df["Spending Score (0-100)"], label="Points", ax=ax[0])
sns.scatterplot(x=centers[:,2], y=centers[:,3], label="Centroid", ax=ax[0], s=75)
sns.scatterplot(x=customer_df["Income (k$)"][sorted_idx], y=customer_df["Spending Score (0-100)"][sorted_idx],
                label="Anomalies", s=150, ax=ax[0], palette="Dark2")
ax[0].set_title("Annual Income VS Spending Score", fontsize=12)
ax[0].set_xlabel("Annual Income (k$)")
ax[0].set_ylabel("Spending Score (0-100)")
ax[0].grid(True)

#Plot Age VS Spending Score
scatter2=sns.scatterplot(x=customer_df["Age"], y=customer_df["Spending Score (0-100)"], label="Points", ax=ax[1])
sns.scatterplot(x=centers[:,1], y=centers[:,3], label="Centroid", ax=ax[1], s=75)
sns.scatterplot(x=customer_df["Age"][sorted_idx], y=customer_df["Spending Score (0-100)"][sorted_idx],
                label="Anomalies", s=150, ax=ax[1], palette="Dark2")
ax[1].set_title("Age VS Spending Score", fontsize=12)
ax[1].set_xlabel("Age")
ax[1].set_ylabel("Spending Score (0-100)")
ax[1].grid(True)

#Plot Age VS Annual Income
scatter3=sns.scatterplot(x=customer_df["Age"], y=customer_df["Income (k$)"], label="Points", ax=ax[2])
sns.scatterplot(x=centers[:,1], y=centers[:,2], label="Centroid", ax=ax[2], s=75)
sns.scatterplot(x=customer_df["Age"][sorted_idx], y=customer_df["Income (k$)"][sorted_idx],
                label="Anomalies", s=150, ax=ax[2], palette="Dark2")
ax[2].set_title("Age VS Annual Income", fontsize=12)
ax[2].set_xlabel("Age")
ax[2].set_ylabel("Annual Income (k$)")
ax[2].grid(True)

#Turn off respective legends in subplots
for i in range(0,3):
    ax[i].legend_=None

#Create a shared legend
legend_labels=["Points", "Centroid", "Anomalies"]
handles, labels = scatter1.get_legend_handles_labels()
legend = fig.legend(handles, legend_labels, loc='center', bbox_to_anchor=(0.5, -0.01), ncol=len(legend_labels))

fig.suptitle("Anomaly Detection with K-Means", fontsize=18, fontweight="bold")
plt.tight_layout()
plt.show()
```




```
#Extract the anomalies
anomaly_X = customer_df.iloc[sorted_idx][["Gender", "Age", "Income (k$)", "Spending Score (0-100)"]]
display(anomaly_X)

#Check which clusters do they belong to in the unscaled data
anomalies = customer_original.iloc[sorted_idx][["Gender", "Age", "Income (k$)", "Spending Score (0-100)", "K-Means Clusters"]]
display(anomalies)
```

	Gender	Age	Income (k$)	Spending Score (0-100)
199	0.0	-0.635135	2.917671	1.273347
198	0.0	-0.491602	2.917671	-1.250054
8	0.0	1.804932	-1.586321	-1.832378
10	0.0	2.020232	-1.586321	-1.405340

	Gender	Age	Income (k$)	Spending Score (0-100)	K-Means Clusters
199	Male	30	137	83	2
198	Male	32	137	18	3
8	Male	64	19	3	5
10	Male	67	19	14	5

Observations:

- The 4 anomalies belong to data points in Clusters 2, 3 and 5 where the first row is the most extreme anomaly.
- First 2 anomalies earn more annual income than the other customers
- Last 2 anomalies who are much older than the other customers earn way lesser annual income than the other customers

Step 5: Model Interpretation (Features Analysis)

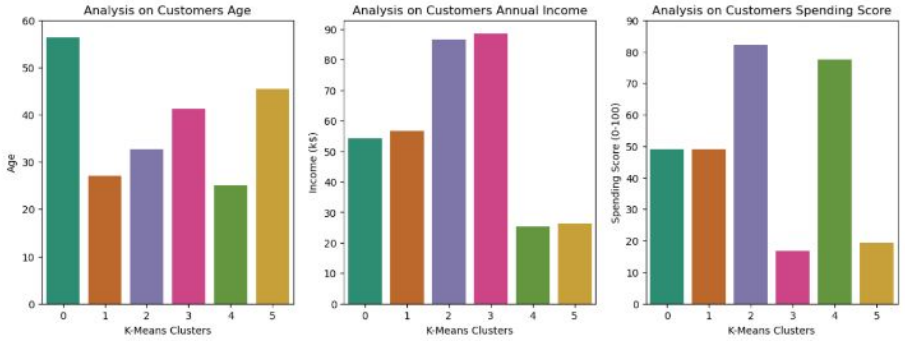
Deeper Average Analysis of K-Means

- Reveals the average age, annual income, spending score and predominant gender in each cluster using K-Means

```
#Retrieve the average value for each cluster
avg_data = customer_original.groupby(["K-Means Clusters"], as_index=False).mean()
print(avg_data)
```

```
   K-Means Clusters        Age  Income (k$)  Spending Score (0-100)
0                 0  56.333333    54.266667               49.066667
1                 1  27.000000    56.657895               49.131579
2                 2  32.692308    86.538462               82.128205
3                 3  41.264706    88.500000               16.764706
4                 4  25.000000    25.260870               77.608696
5                 5  45.523810    26.285714               19.380952
```

```
#Reveals average age for each cluster
fig, ax=plt.subplots(1, 3, figsize=(15, 5))
sns.barplot(x="K-Means Clusters", y="Age", palette="Dark2", data=avg_data, ax=ax[0])
ax[0].set_title("Analysis on Customers Age")
ax[0].set_yticks(np.arange(0,61,10))

sns.barplot(x="K-Means Clusters", y="Income (k$)", palette="Dark2", data=avg_data, ax=ax[1])
ax[1].set_title("Analysis on Customers Annual Income")
ax[1].set_yticks(np.arange(0,91,10))

sns.barplot(x="K-Means Clusters", y="Spending Score (0-100)", palette="Dark2", data=avg_data, ax=ax[2])
ax[2].set_title("Analysis on Customers Spending Score")
ax[2].set_yticks(np.arange(0,91,10))
plt.show()
```




```
#Reveals gender for each cluster, predominantly female/male or equal amount
gender_count=pd.DataFrame(customer_original.groupby(['K-Means Clusters', 'Gender'])["Gender"].count())
gender_count
```

		Gender
K-Means Clusters	Gender	
0	Female	26
	Male	19
1	Female	26
	Male	13
2	Female	21
	Male	18
3	Female	14
	Male	20
4	Female	13
	Male	10
5	Female	13
	Male	8

Observations:

- Clusters 0, 1 and 5 has predominantly more females than males
- Clusters 2 and 4 has almost an equal amount of females and males
- Cluster 3 has predominantly more males than females

K-Means Clusters Attributes

- Cluster 0 (turquoise): Moderate annual income, Moderate spending score
  - Mid 50s
  - 54k Moderate Annual Income
  - Moderate Spending Score of 49
  - Predominantly females
- Cluster 1 (orange): Moderate annual income, Moderate spending score
  - Late 20s
  - 56k Moderate Annual Income
  - Moderate Spending Score of 49
  - Predominantly females
- Cluster 2 (violet): High annual income, High spending score
  - Early 30s
  - 86K High Annual Income
  - High Spending Score of 49
  - Approximately equal proportion of males and females
- Cluster 3 (pink): High annual income, Low spending score
  - Early 40s
  - 88k High Annual Income
  - Low Spending Score of 16
  - Predominantly males
- Cluster 4 (light green): Low annual income, High spending score
  - Mid 20s
  - 25k Low Annual Income
  - High Spending Score of 78
  - Approximately equal proportion of males and females
- Cluster 5 (yellow): Low annual income, Low spending score
  - Mid 40s
  - 26k Low Annual Income
  - Low Spending Score of 19
  - Predominantly females

Step 5: Clusters Interpretation

K-Means Clusters Interpretation

- **Cluster 0: Near Retired**
  - Group of people who are oldest in age with a moderate annual income and moderate spending score. They are likely saving up for their retirement fund, meanwhile spending on only the necessary items they require.
- **Cluster 1: Middle Income**
  - Group of people who make and spend their money at a moderate level. They are cautious with how they spend their money as they are not extremely wealthy and do not have a stable source of income. As the age of this cluster is also late 20s, they are young parents who have children to support and must be frugal sometimes to reduce financial issues.
- **Cluster 2: High SES Individuals**
  - Group of individuals who earns and spends high amount of money. They have done well early in their careers even at the young age of 30. They will also spend their money at a fairly high rate at restaurants and stores at the shopping mall.
- **Cluster 3: Cautious Spenders**
  - Group of people who earns highest annual income but has the lowest spending score. They consist of middle-aged people who are saving up money for future goals like moving to another property in another city or state or building up an emergency fund for career advancement
- **Cluster 4: Impulsive Shoppers**
  - Group of young people aged mid 20 who just started out in their career, earning low amount of money but spending way over their means. They enjoy their relaxed and luxurious lifestyle by spending money on fashion to follow trends which might pose a financial threat to them as a possible consequence
- **Cluster 5: Low SES Individuals**
  - Group of individuals who earns and spends low amount of money. Aged mid 40s, they do not have a stable source of income and can only spend their money on daily necessities like groceries and clothing

Summary

- K-Means is my **best model** based on the **silhouette score, calinski harabasz & davies-bouldin index** score metrics which means clusters formed are **best defined** using K-Means.
- **Most important** group of customers: **High SES individuals**.
- **Reason**: **Successful** in their careers at a **young age** & most likely have **disposable income** for discretionary spending
- **Recommendation**: Provide a **premium shopping experience** for customers by **upscaling stores & brands**
- **Examples**: Ensure the mall offers a **wide range of high-end boutiques & luxury fashion retailers** that cater to the preference of this group, encouraging them to **continue shopping** at the mall