

IBM MACHINE LEARNING

UNSUPERVISED MACHINE LEARNING MODELS FOR MALL CUSTOMER SEGMENTATION



Fatima, Sayeda

10/23/2022

Table of Contents

1) Project Overview and Objective	2
1a) Problem Overview	2
1b) Objectives	2
1c) Implications for Business:	2
2) About the Dataset	3
2a) Brief Description of Chosen Data Set	3
2b) Summary of Data Attributes	3
2c) Main Aim of Analysis	3
3) Data Exploration, Data Cleansing and Features Engineering	4
3a) Data Exploration:	4
3b) Data Cleansing Actions & Features Engineering:	7
4) Summary of Training Different Clustering Models	13
4a) Summarizing Employed Models	13
1) K-Means Clustering Algorithm:	13
2) Mini-Batch K-Means Clustering Algorithm:	15
3) Hierarchical Agglomerative Clustering Algorithm:	17
4) Density-Based Spatial Clustering of Applications with Noise (DBSCAN) Algorithm:	19
5) Ordering Points to Identify the Clustering Structure (OPTICS) Algorithm:	21
5) Result Summary and Recommended Model	23
5a) Result Summary	23
5b) Model Choice and Justification	23
6) Summary Key Findings and Insights	24
6a) Cluster Description	24
6b) Cluster Information Assignment	24
6c) Cluster Assignment Result	25
7) Future Recommendations	26
8 Useful Links	27
8a) Link to Other Useful Models	27
8b) Model Evaluation and Scoring	27
8c) SKLEARN Unsupervised Model Types and their Parameters	27
9) Github Link to Assignment Notebook	27

1) Project Overview and Objective

1a) Problem Overview

One of the main post-Covid challenges faced by shopping malls is sustaining revenue and maintaining profits. This challenge has been even more compounded by a falling pounds and global onset of another recession. Consequently, both big and small shopping malls have to devote more efforts to ensure attracting customers through discounts and offers specific to their interests to ensure repeat purchase and customer loyalty. It is here that unsupervised machine learning models can be very useful to gain deeper insight into underlying factors and their relationship in driving customer segmentation.

1b) Objectives

Hence, the main objective of the following unsupervised machine learning modelling and analysis is targeted towards answering the following queries:

- Into how many clusters can the customers be segmented?
- What are the main characteristic driving this segmentation?

1c) Implications for Business:

As a consequence, implementation of the model will enable the business:

- To segment customers into separate groups.
- To gain insight into customer characteristics contributing to identified grouping for sales strategy formulation.
- Target specific offers based on customers segmentation and cluster contributory characteristics can which will likely lead to increased purchase, customer loyalty and sustainable profits.

2) About the Dataset

2a) Brief Description of Chosen Data Set

This project uses a hypothetical dataset 'Customer Mall' which seems to have been acquired for regular customers visiting a shopping a mall and was downloaded from the following link:

<https://www.kaggle.com/code/vjchoudhary7/kmeans-clustering-in-customer-segmentation/data>

2b) Summary of Data Attributes

The dataset exhibits 200 data points (rows) and 5 features (columns) reflecting on customers' characteristics where, based on their spending, each customer has been assigned a Spending Score. Of these, the main four features are 'Age', 'Annual Income', 'Spending Score' and 'Gender'.

2c) Main Aim of Analysis

Hence, the aim of this project is segmentation of mall customers to aid formulation of target marketing strategy and its implementation. By segmenting customers into clusters, specific offers can be targeted to each cluster which will likely lead to increased purchase, customer loyalty and sustainable profits.

Therefore, the aim of this analysis is to:

- Segment customers into different groups and clusters
- Reflect on the main characteristic driving this segmentation

A2T63J6JM74V7uafcf

https://scentscientists.com/products/rose-geraniol-10ml?variant=37899469029552¤cy=GBP&utm_medium=product_sync&utm_source=google&utm_content=sag_organic&utm_campaign=sag_organic&utm_campaign=gs-2021-01-08&utm_source=google&utm_medium=smart_campaign&gclid=Cj0KCQjw48OaBhDWARIsAMd966BNXOfbyPl62YecvCqLoRog1oo7ok0zvYvxMtrKOpnWpW-QBPhjbgQaApKHEALw_wcB

3) Data Exploration, Data Cleansing and Features Engineering

Since the quality of any machine learning model highly depends on quality of data, hence, this stage is not only most important but is also time consuming. Hence, it was conducted in a step-by-step process.

3a) Data Exploration:

- Data was first loaded into pandas dataframe

```
1 customer_data = pd.read_csv(input_file_path)
2 display_alert_color("Load and Read Dataset","darkbrown" ,"warning")
3 customer_data
```

Load and Read Dataset

	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 x 5 columns

- Column types were then explored

```
1 display_alert_color("Display Data Info","darkbrown" ,"warning")
2 customer_data.info()
```

Display Data 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   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
```

- Automated Exploratory Data Analysis was performed using Sweetviz

```

1 display_alert("Initial Exploratory Data Analysis", "success")
2 display_alert_color("Perform Quick EDA: To see dataset's distribution and its dispersion. ", "teal", "warning")
3 df_eda = sv.analyze(customer_data) # Use Sweetviz for Automated EDA
4 #df_eda.show_html() # Uncomment to Generate Online Report
5 display_alert("Method to Generate Preliminary Exploratory Data Analysis ", "warning")

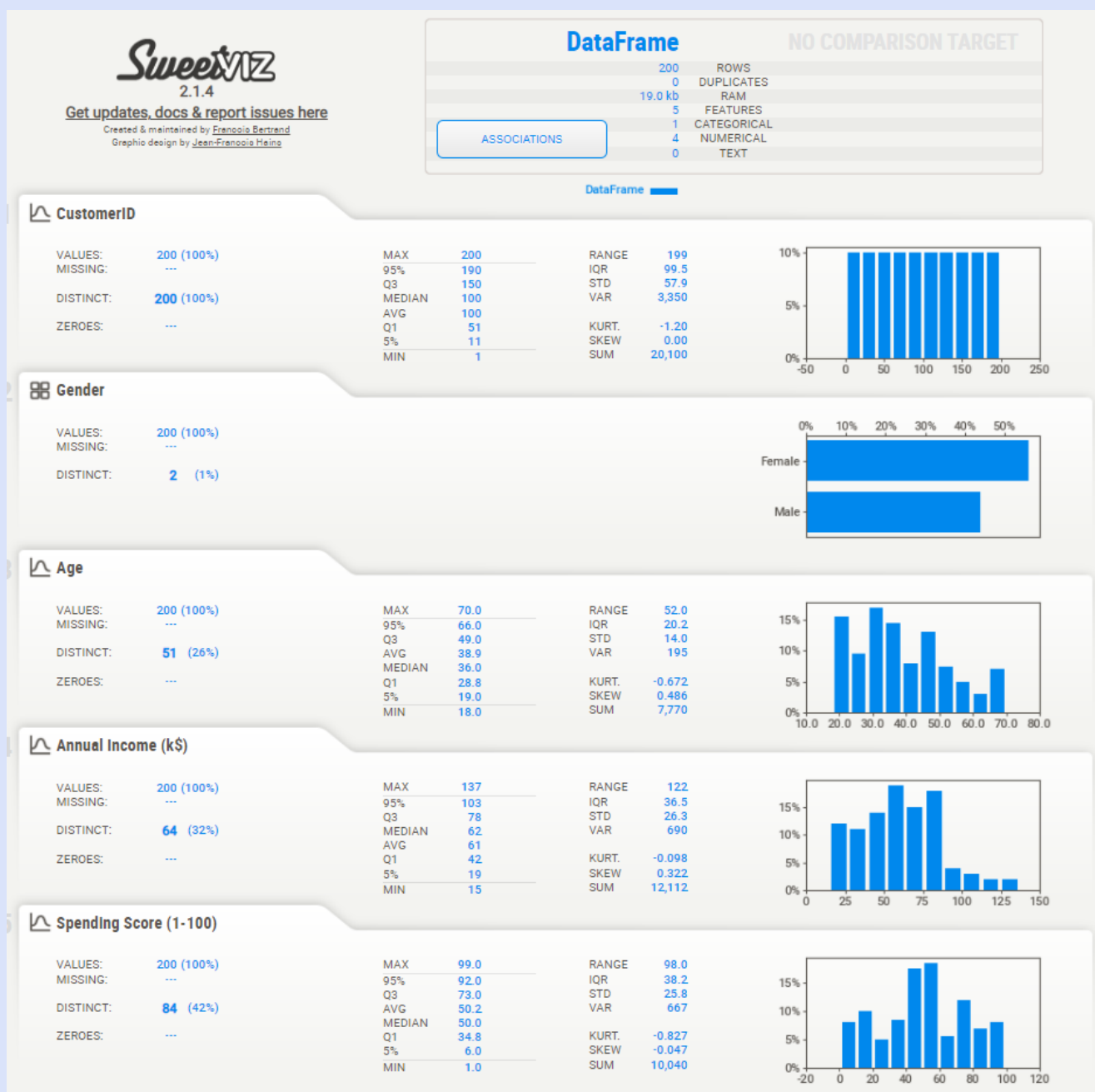
```

Initial Exploratory Data Analysis

Perform Quick EDA: To see dataset's distribution and its dispersion.

Done! Use 'show' commands to display/save. [100%] 00:00 -> (00:00 left)

Report SWEETVIZ_REPORT.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop up, regardless, the report IS save d in your notebook/colab files.



- Additional descriptive statistics were computed to summarize shape of a dataset's distribution, its dispersion and central tendency

Summary Statistics

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
CustomerID	200.0	0	0	0	100.50	57.879185	1.0	50.75	100.5	150.25	200.0
Gender	200.0	2	Female	112	0.00	0.000000	0.0	0.00	0.0	0.00	0.0
Age	200.0	0	0	0	38.85	13.969007	18.0	28.75	36.0	49.00	70.0
Annual Income (k\$)	200.0	0	0	0	60.56	26.264721	15.0	41.50	61.5	78.00	137.0
Spending Score (1-100)	200.0	0	0	0	50.20	25.823522	1.0	34.75	50.0	73.00	99.0

```
1 print(colored("\nData Analysis Summary:", 'cyan', attrs=['bold']))
2 analyze
```

Data Analysis Summary:

	Columns	types	counts	distincts	nulls	% nulls	uniques	skewness	kurtosis	Corr_Spending Score (1-100)
0	Spending Score (1-100)	int64	200	84	0	0.0	0	-0.047220	-0.826629	1.000000
1	CustomerID	int64	200	200	0	0.0	0	0.000000	-1.200000	0.013835
2	Annual Income (k\$)	int64	200	64	0	0.0	0	0.321843	-0.098487	0.009903
3	Gender	object	200	2	0	0.0	0	0.000000	0.000000	0.000000
4	Age	int64	200	51	0	0.0	0	0.485569	-0.671573	-0.327227

3b) Data Cleansing Actions & Features Engineering:

In machine learning, feature selection is the method to reduce the number of input variables during developing predictive modelling. This reduction in input variables is necessary not only to minimize computational cost of modeling but also to achieve performance improvement of the model.

Among widely practices feature selection approaches include statistical-based feature selection methods which use statistical measures to evaluate relationship between each input variable and the target variable and then select those exhibiting strongest relationship with the latter. While these methods can be both speedy and effective, however, the ultimate choice of statistical measure is largely dependant on data types of both of these variables.

Irrespective of the statistical measure being employed, two dominant feature selection techniques, that is supervised and unsupervised, exist where the former can be further categorized into wrapper, filter and intrinsic techniques. Filter-based feature selection methods employs statistical measures to evaluate correlation between input and output variables so that those exhibiting highest correlations are selected. Statistical measures employed in filter-based feature selection are normally univariate in nature since they evaluate relationship of single input variables one by one with target variable, disregarding their interaction with each other.

Consequently, adopting filter-based feature selection methods, the project approached filter engineering in the following steps.

An automated data cleansing method was created to do the following:

- Drop Columns with Unique Values Less than threshold of 2
- Drop Highly Skewed & Low Correlation Columns with target
- Drop Columns with High Nan Values

Method to Drop Columns

- 1) With Distinct < 2
- 2) With High Skewness and Low Correlation to Target
- 3) Drop Columns With High Nan Values

```

1 def drop_cols(*name):
2     n = name # Extract Dataframe by Name...this will create a 3d tuple
3     n = (n[0]) # Convert Tuple to DataFrame
4     df_name = [x for x in globals() if globals()[x] is n][0] # Extract Name of Imported Dataframe
5
6     n = n.fillna(0) #Fill Remaining Missing Values with Zero
7     # Find Mean of Null, Nan and Zero Values Before Any Drops
8     m0 = n.isin([' ', 'NULL', 'NaN', 0]).mean().sort_values(axis=0, ascending=False, inplace=False, kind='quicksort', na_posit
9
10
11     # 1) Drop Columns with Unique Values Less than threshold
12     unique_counts = pd.DataFrame.from_records([(col, n[col].nunique()) for col in n.columns], # get unique counts
13         columns=['Column_Name', 'Unique']).sort_values(by=['Unique'])
14     unique = unique_counts[(unique_counts['Unique'] < 2)] #If threshold is Less than 2 then
15     drop1 = (unique['Column_Name']).tolist() # First List of columns to drop
16
17     print(colored("\nDrop 1: ", 'blue', attrs=['bold']))
18     +colored(drop1, 'magenta', attrs=['bold']))
19
20     # 2) Drop Highly Skewed & Low Correlation Columns with target
21     drop2 = analyze[(analyze[column] != 1.000000) & ((analyze['skewness'] > 1) & (analyze[column] < 0))]
22     print(colored("\nDrop 2: \n ", 'blue', attrs=['bold']))
23     +colored(drop2, 'magenta', attrs=['bold']))
24
25     drop2 = drop2.sort_values(by='Columns', ascending=True)
26     drop2 = drop2['Columns'].tolist() # Second List of columns to drop
27
28     drop = drop1 + drop2 + delete_features # Final List of columns to drop
29     print(colored("\nFinal Column Drop List: ", 'blue', attrs=['bold']))
30     +colored(drop, 'magenta', attrs=['bold']))
31
32     if target in drop: # Remove Target from List
33         drop = drop.remove(target)
34     else:
35         print(target, " Not Found.")
36
37     n = n.drop(columns=[col for col in n if col in drop]) # Drop Dataframe Columns if in List
38
39     # Find Mean of Null, Nan and Zero Values Before Dropping
40     m1 = n.isin([' ', 'NULL', 'NaN', 0]).mean().sort_values(axis=0, ascending=False, inplace=False, kind='quicksort', na_posit
41
42     # 3) Drop Columns With High Nan Values
43     #drop_thresh = .90 # Identify Drop Threshold
44     #n = n.loc[:, df.isin([' ', 'NULL', 'NaN', 0]).mean() > drop_thresh] # drop columns if Mean is > 0.90
45
46     n = n.fillna(0) #Fill Remaining Missing Values with Zero
47     #n = n.replace(["NaN"], 0).sort_values(by=target, ascending=False) # Replace all Nan Values with Zero
48
49     # Find Mean of Null, Nan and Zero Values After Dropping
50     m2 = n.isin([' ', 'NULL', 'NaN']).mean().sort_values(axis=0, ascending=False, inplace=False, kind='quicksort', na_positio
51
52     #Print Results
53     print(colored("\nDataframe Average Null Values Before Any Drops\n ", 'blue', attrs=['bold']))
54     +colored(m0, 'magenta', attrs=['bold']))
55     +colored("\n\n Low Distict Columns to Drop: ", 'green', attrs=['bold']))
56     +colored(drop1, 'red', attrs=['bold']))
57     +colored("\n\nDataframe Average Null Values After Dropping Highly Skewed Columns\n ", 'green', attrs=['bold']))
58     +colored(m1, 'red', attrs=['bold']))
59     +colored("\n\n Drop Columns if Mean is > 0.90 \n", 'green', attrs=['bold']))
60     +colored("\n\nDataframe Average Null Values After Drop and 'Nan' Replacement\n", 'blue', attrs=['bold']))
61     +colored(m2, 'magenta', attrs=['bold']))
62     +colored(type(m2), 'magenta', attrs=['bold'))
63
64     return n
65
66 # Return Function
67 n = drop_cols(df)
68 df = n.copy()
69 df.info()

```

Null values were summed, and data was found to exhibit zero null values. Thus, no filling of null values was required.

Dataframe Average Null Values Before Any Drops

```
Spending Score (1-100)    0.0
Annual Income (k$)        0.0
Age                       0.0
Gender                    0.0
CustomerID                0.0
dtype: float64
```

Low Distict Columns to Drop: []

Dataframe Average Null Values After Dropping Highly Skewed Columns

```
Spending Score (1-100)    0.0
Annual Income (k$)        0.0
Age                       0.0
Gender                    0.0
dtype: float64
```

Drop Columns if Mean is > 0.90

Dataframe Average Null Values After Drop and 'Nan' Replacement

```
Spending Score (1-100)    0.0
Annual Income (k$)        0.0
Age                       0.0
Gender                    0.0
dtype: float64<class 'pandas.core.series.Series'>
```

Data Encoding: Additionally, Data Encoding of Object or String Columns was carried out to facilitate any statistical computation during features selection process. Hence, after deep copying of original dataset, a function was created and employed to encode object data using Scikit-learn label encoder.

Method to Encode Object Type Columns:

```

1 # Method to encode object/string columns
2 def encoder(*name):
3     # Accept an argument, return a value.
4     n = name # Extract Dataframe by Name...this will create a 3d tuple
5     n = (n[0]) # Convert Tuple to To Dataframe
6     df_name = [x for x in globals() if globals()[x] is n][0] # Extract Name of Imported Dataframe
7
8     # 1) List all Object/String Columns
9     from sklearn import preprocessing
10    cat_columns = n.select_dtypes(include=[object]) # Get Object Type Columns to Convert to Encoded Categories
11
12    categorical_column = list(cat_columns.columns) # List of columns to for Label encoding
13
14    print(colored("\n\nColumns Requiring Encoding: \n", 'blue', attrs=['bold'])
15          + colored(categorical_column, 'green', attrs=['bold']))
16
17    # Make Empty Dataframe to decode encoded data Later
18    decode_features = pd.DataFrame()
19
20    ##### Employ Scikit-Learn Label encoding to encode object data #####
21    lab_enc = preprocessing.LabelEncoder()
22    for col in categorical_column:
23        n[col] = lab_enc.fit_transform(n[col])
24        name_mapping = dict(zip(lab_enc.classes_, lab_enc.transform(lab_enc.classes_)))
25
26        ##### Decode Encoded Data #####
27        feature_df = pd.DataFrame([name_mapping])
28        feature_df = feature_df.astype(str)
29        feature_df = (col + "_" + feature_df.iloc[0:])
30        feature_df["Feature"] = col
31        decode_features = decode_features.append(feature_df) # Append Dictionaries to Empty Dataframe for Later Decoding
32
33        ##### Print Encoded Data #####
34        print(colored("Feature: \n", 'blue', attrs=['bold'])
35              + colored(col, 'red', attrs=['bold'])
36              + colored("\nMapping: \n", 'blue', attrs=['bold'])
37              + colored(name_mapping, 'green', attrs=['bold'])
38              + colored("\n\nType n: ", 'blue', attrs=['bold'])
39              + colored(type(n), 'magenta', attrs=['bold'])
40              )
41    n.head(3)
42
43    ##### 2) Make Decoded Factor Dataframe with Description #####
44    factor_list = decode_features.T # Transpose Dataframe and place in new dataframe
45    factor_list = factor_list.replace(np.nan, "/") # nan values with forward slash
46    factor_list["Factors"] = factor_list.astype(str).agg("").join,axis=1).replace(r'[^\\w\\s]|/',' ', regex=True) # Aggregate &
47    factor_list.reset_index() # Reset index before copying/assigning it to a new column
48    factor_list["Description"] = factor_list.index # Assign index to column
49
50    return n, factor_list

```

Now Encoding Categorical Data:

Columns Requiring Encoding:

['Gender']

Feature:

Gender

Mapping:

{'Female': 0, 'Male': 1}

Type n: <class 'pandas.core.frame.DataFrame'>

Encoded Dataframe

<class 'pandas.core.frame.DataFrame'>

Outlier Treatment: Supervised learning models like K_Means are sensitive to outliers. Hence, an automated method was created to replace outliers with “Mode”, that is the most common value.

Method to Explore and Adjust Outliers:

Replace Outlier Values with Mode (Most Frequent Value)

```

1 def outliers(*name):
2     n = name # Extract Dataframe by Name...this will create a 3d tuple
3     n = (n[0]) # Convert Tuple to To Dataframe
4     df_name = [x for x in globals() if globals()[x] is n][0] # Extract Name of Imported Dataframe
5
6     cols = n.columns # All Columns
7
8     # Numeric Columns
9     numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
10    numeric_cols = n.select_dtypes(include=numerics)
11    numeric_cols = numeric_cols.columns.tolist()
12
13    # Object Columns
14    categorical_cols = list(set(cols) - set(numeric_cols))
15
16    # Skewed Columns
17    skewed_cols = analyze[(analyze['skewness'] > 0) | (analyze['skewness'] < 0)]
18    skewed_cols = skewed_cols['Columns'].tolist()
19    skewed_cols.remove(target) # Remove target column
20
21    # Replace Outliers
22    for col in n.columns:
23        if col in skewed_cols:
24            print(colored(col, 'magenta', attrs=['bold']))
25            + colored(" is Skewed...", 'blue', attrs=['bold'])
26            )
27
28            mode = n[col].mode()
29            mode = mode[0]
30
31            if col in numeric_cols:
32                print(colored(col, 'magenta', attrs=['bold']))
33                + colored(" Column Type is: ", 'blue', attrs=['bold'])
34                + colored(n[col].dtypes, 'red', attrs=['bold'])
35                )
36                #Calculate quantiles and IQR
37                Q1 = n[col].quantile(0.25) # Same as np.percentile but maps (0,1) and not (0,100)
38                Q3 = n[col].quantile(0.75)
39                IQR = Q3 - Q1
40                # Replace with Mode
41                n[col] = np.where((n[col] < (Q1 - 1.5 * IQR)) | (n[col] > (Q3 + 1.5 * IQR)), mode, n[col])
42
43                print(colored("\nReplaced ", 'blue', attrs=['bold']))
44                + colored(col, 'magenta', attrs=['bold'])
45                + colored(" Skewed Values by Mode: ", 'blue', attrs=['bold'])
46                + colored(mode, 'red', attrs=['bold'])
47                + colored("\n", 'magenta', attrs=['bold'])
48                + colored((n[col]), 'green', attrs=['bold'])
49                )
50
51            else:
52                print("")
53    df_transformed = n.copy()
54    return df_transformed

```

Apply 'outliers' Method to New Dataframe

```

Age is Skewed...
Age Column Type is: int64

Replaced Age Skewed Values by Mode: 32
0      19
1      21
2      20
3      23
4      31
..
195    35
196    45
197    32
198    32
199    30
Name: Age, Length: 200, dtype: int64
Annual Income (k$) is Skewed...
Annual Income (k$) Column Type is: int64

Replaced Annual Income (k$) Skewed Values by Mode: 54
0      15
1      15
2      16

```

Data Scaling: Due to dependency of clustering algorithms on distance matrix, data scaling was carried out.

Data Scaling:

Because Clustering algorithms use distance metric to make categories, hence, data scaling is recommended.

```
1 scaler = StandardScaler()
2 scaled_df = scaler.fit_transform(df_encoded)
3 scaled_df
```

```
array([[ 1.12815215, -1.42456879, -1.78535193, -0.43480148],
       [ 1.12815215, -1.28103541, -1.78535193,  1.19570407],
       [-0.88640526, -1.3528021 , -1.74543796, -1.71591298],
       [-0.88640526, -1.13750203, -1.74543796,  1.04041783],
       [-0.88640526, -0.56336851, -1.70552399, -0.39597992],
       [-0.88640526, -1.20926872, -1.70552399,  1.00159627],
       [-0.88640526, -0.27630176, -1.66561002, -1.71591298],
       [-0.88640526, -1.13750203, -1.66561002,  1.70038436],
       [ 1.12815215,  1.80493225, -1.62569604, -1.83237767],
       [-0.88640526, -0.6351352 , -1.62569604,  0.84631002],
       [ 1.12815215,  2.02023231, -1.62569604, -1.4053405 ],
       [-0.88640526, -0.27630176, -1.62569604,  1.89449216],
       [-0.88640526,  1.37433211, -1.58578207, -1.36651894],
       [-0.88640526, -1.06573534, -1.58578207,  1.04041783],
       [ 1.12815215, -0.13276838, -1.58578207, -1.44416206],
       [ 1.12815215, -1.20926872, -1.58578207,  1.11806095],
       [-0.88640526, -0.27630176, -1.5458681 , -0.59008772],
       [ 1.12815215, -1.3528021 , -1.5458681 ,  0.61338066],
       [ 1.12815215,  0.94373197, -1.46604016, -0.82301709],
```

4) Summary of Training Different Clustering Models

4a) Summarizing Employed Models

Following three main classifier models have been used to predict employee attrition.

1) K-Means Clustering Algorithm:

Due to the focus on segmentation, the popular K-Means Clustering Algorithm with optimal parameters was employed.

MODEL 1: KMeans Clustering Algorithm with Optimal Parameter Tuning

```

1  display_alert_color("MODEL1: KMeans Clustering Algorithm RESULTS", "purple","warning")
2
3  results=pd.DataFrame(columns=['Model','Clusters','Silhouette Score','Davies Bouldin Score'])
4
5  model_name = "KMeans"
6
7  # Define Parameters
8  n_digits = len(np.unique(df_encoded)) # Get Lenth of dataframe to get maximum cluster range
9  algorithm = ["lloyd","elkan","auto","full"]
10
11 # Loop by Parameters
12 for i in range(2,n_digits):
13     for a in algorithm:
14         kmeans_labels = KMeans(n_clusters=i,algorithm=a, random_state=123).fit_predict(df_encoded)
15         sil = (metrics.silhouette_score(df_encoded,kmeans_labels, metric='euclidean').round(3))
16         dbs = (metrics.davies_bouldin_score(df_encoded,kmeans_labels).round(3))
17         results = results.append({'Cluster': i, 'Algorithm':a, 'Silhouette Score': sil, 'Davies Bouldin Score': dbs}, ignore
18         results.sort_values(by=['Silhouette Score'], inplace=True, ascending=False) # Sort by highest 'Silhouette Score'
19
20 best_cluster = int(results['Cluster'].iloc[0])
21 best_algorithm = results['Algorithm'].iloc[0]
22
23 model_KMeans = KMeans(n_clusters=best_cluster, algorithm=best_algorithm)
24 model_KMeans.fit(df_encoded) # Fit Model
25 predicted_labels_KMeans = model_KMeans.fit_predict(df_encoded) # Predict Data
26
27 sil = (metrics.silhouette_score(df_encoded,predicted_labels_KMeans).round(3))
28 dbs = (metrics.davies_bouldin_score(df_encoded,predicted_labels_KMeans).round(3))
29
30 # Capture Model Results & Append to Main Result Frame
31 results_KMeans = pd.DataFrame.from_dict({'Model':model_name,'Algorithm':best_algorithm,'Clusters':best_cluster,'Silhouette Sc
32 Combined_Results = Combined_Results.append(results_KMeans,ignore_index=True) # Append model results to main result dataframe
33
34 print(colored("Total Possible Clusters: ", 'blue', attrs=['bold']))
35     + colored(n_digits, 'red', attrs=['bold'])
36     + colored("\n\nCluster Results:\n", 'blue', attrs=['bold'])
37     + colored(results.head(4), 'blue', attrs=['bold'])
38     + colored("\n\nCombined Model Results:\n", 'green', attrs=['bold'])
39     + colored(Combined_Results, 'magenta', attrs=['bold'])
40
41 # Plot Figure
42 import plotly.express as px
43 fig = px.scatter_3d(df_encoded, x='Age', y='Annual Income (k$)', z='Spending Score (1-100)', color=model_KMeans.fit_predict(
44 fig.update_layout(coloraxis_colorbar=dict(yanchor="top", y=1, x=0, ticks="outside"))
45 # fig.update_layout is being used to make the 'color' legend shift towards Left side, otherwise the 'color' and 'Gender' Leg
46 # fig.show()
```


MODEL1: KMeans Clustering Algorithm RESULTS

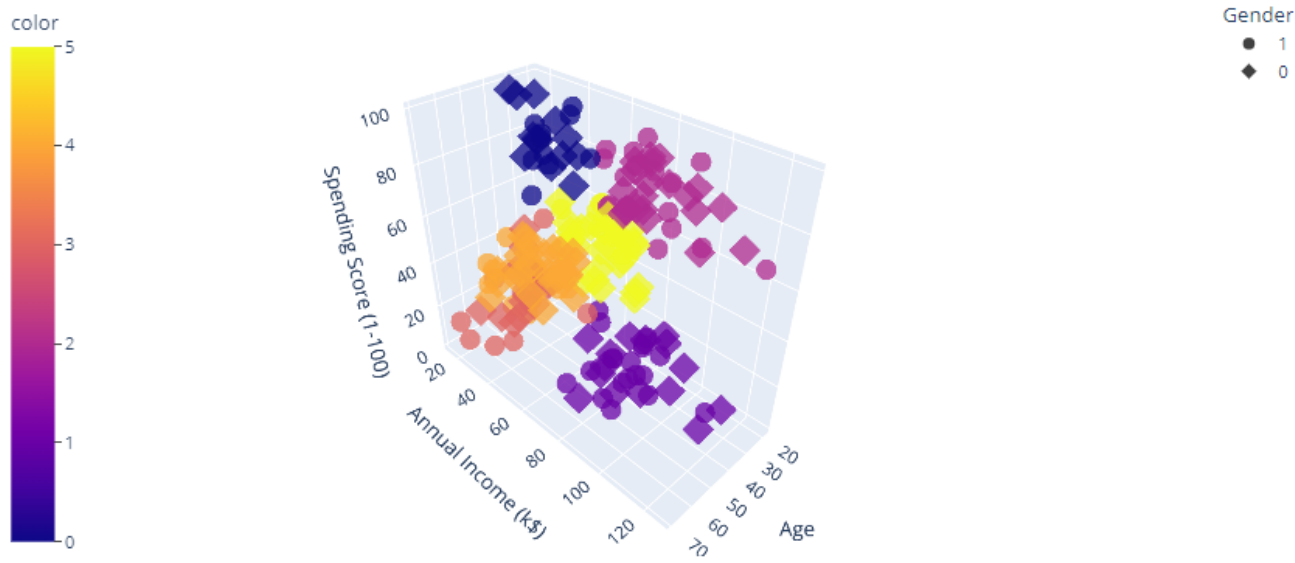
Total Possible Clusters: 101

Cluster Results:

	Model	Clusters	Silhouette Score	Davies Bouldin Score	Algorithm	Model#
0	KMeans	6	0.453	0.75	lloyd	Model 1
3	KMeans	6	0.453	0.75	full	Model 1
1	KMeans	6	0.453	0.75	auto	Model 1
2	KMeans	6	0.453	0.75	elkan	Model 1

Combined Model Results:

Model#	Model	Clusters	Silhouette Score	Davies Bouldin Score	
0	Model 1	KMeans	6	0.453	0.75



2) Mini-Batch K-Means Clustering Algorithm:

Mini-Batch K-Means Clustering Algorithm with optimal parameters, which is faster than K-Means due to utilizing random fixed size data batches, was employed.

MODEL 2: MiniBatchKMeans Clustering Algorithm with Optimal Parameter Tuning

```

1 display_alert_color("MODEL 2: MiniBatchKMeans Clustering Algorithm RESULTS", "purple","warning")
2
3 results=pd.DataFrame(columns=['Model','Clusters','Silhouette Score','Davies Bouldin Score'])
4
5 model_name = "MiniBatchKMeans "
6
7 # Define Parameters
8 n_digits = len(np.unique(df_encoded)) # Get Lenth of dataframe to get maximum cluster range
9
10 # Loop by Parameters
11 for i in range(2,n_digits):
12     MiniBatchKMeans_labels = MiniBatchKMeans(n_clusters=i, random_state=123).fit_predict(df_encoded)
13     sil = (metrics.silhouette_score(df_encoded,MiniBatchKMeans_labels, metric='euclidean').round(3))
14     dbs = (metrics.davies_bouldin_score(df_encoded,MiniBatchKMeans_labels).round(3))
15     results = results.append({'Model': model_name, 'Clusters': i, 'Silhouette Score': sil, 'Davies Bouldin Score': dbs}, ign
16     results.sort_values(by=['Silhouette Score'], inplace=True, ascending=False) # Sort by highest 'Silhouette Score'
17
18 # Collect Best Parameters
19 best_cluster = int(results['Clusters'].iloc[0])
20
21 model_MiniBatchKMeans = MiniBatchKMeans(n_clusters=best_cluster)
22 model_MiniBatchKMeans.fit(df_encoded) # Fit Model
23 predicted_labels_MiniBatchKMeans = model_MiniBatchKMeans.fit_predict(df_encoded) # Predict Data
24
25 sil = (metrics.silhouette_score(df_encoded,predicted_labels_MiniBatchKMeans).round(3))
26 dbs = (metrics.davies_bouldin_score(df_encoded,predicted_labels_MiniBatchKMeans).round(3))
27
28 # Capture Model Results & Append to Main Result Frame
29 results_MiniBatchKMeans =pd.DataFrame.from_dict({'Model':model_name,'Clusters':best_cluster,'Silhouette Score': sil,'Davies
30 Combined_Results = Combined_Results.append(results_MiniBatchKMeans,ignore_index=True) # Append model results to main result
31
32 print(colored("Total Possible Clusters: ", 'blue', attrs=['bold'])
33       + colored(n_digits, 'red', attrs=['bold']))
34       + colored("\n\nCluster Results:\n", 'blue', attrs=['bold'])
35       + colored(results.head(4), 'blue', attrs=['bold'])
36       + colored("\n\nCombined Model Results:\n", 'green', attrs=['bold'])
37       + colored(Combined_Results, 'magenta', attrs=['bold']))
38
39 # Plot Figure
40 import plotly.express as px
41 fig = px.scatter_3d(df_encoded, x='Age', y='Annual Income (k$)', z='Spending Score (1-100)', color=model_MiniBatchKMeans.fit
42 fig.update_layout(coloraxis_colorbar=dict(yanchor="top", y=1, x=0, ticks="outside"))
43 # fig.update_layout is being used to make the 'color' Legend shift towards left side, otherwise the 'color' and 'Gender' Leg
44 # fig.show()

```

MODEL 2: MiniBatchKMeans Clustering Algorithm RESULTS

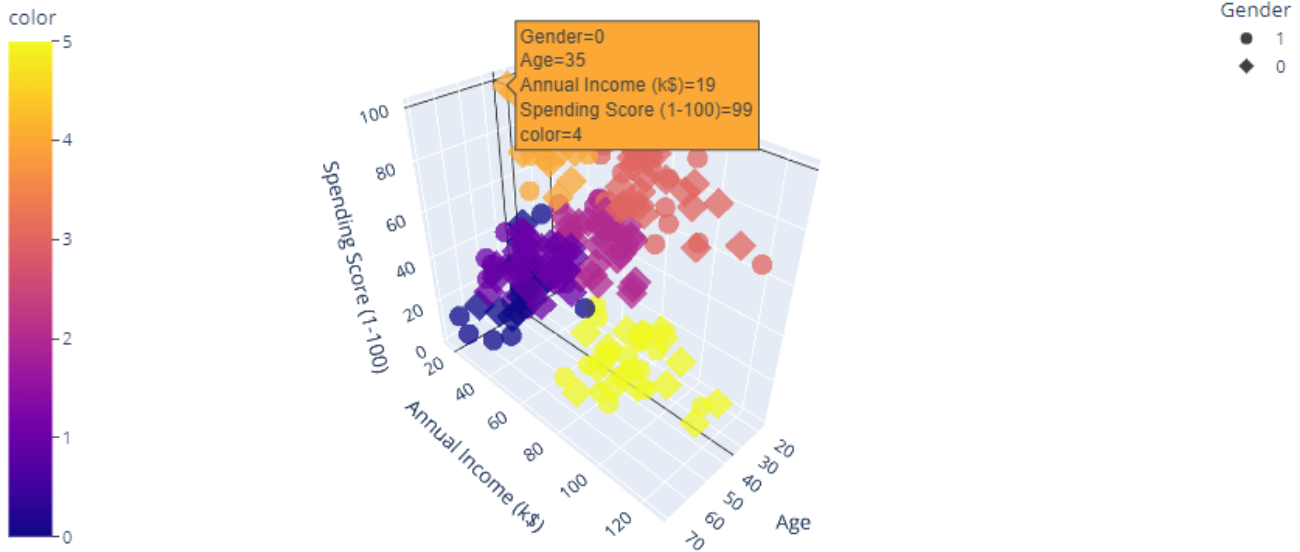
Total Possible Clusters: 101

Cluster Results:

	Model	Clusters	Silhouette Score	Davies Bouldin Score	Model 2
0	MiniBatchKMeans	6	0.453	0.753	Model#
1	MiniBatchKMeans	7	0.434	0.805	Model#
2	MiniBatchKMeans	5	0.427	0.873	Model#
3	MiniBatchKMeans	8	0.412	0.941	Model#

Combined Model Results:

Model#	Model	Clusters	Silhouette Score	Davies Bouldin Score
0	Model 1 KMeans	6	0.453	0.750
1	Model 2 MiniBatchKMeans	6	0.452	0.746



3) Hierarchical Agglomerative Clustering Algorithm:

Hierarchical Agglomerative Clustering Algorithm with optimal parameters, which starts with smaller clusters to merge them into bigger ones, was employed.

MODEL 3: Hierarchical/Agglomerative Clustering Algorithm with Optimal Parameter Tuning

```

1 display_alert_color("MODEL 3: Hierarchical/Agglomerative Clustering Algorithm RESULTS", "purple","warning")
2
3 results=pd.DataFrame(columns=['Model','Clusters','Silhouette Score','Davies Bouldin Score'])
4
5 model_name = "Hierarchical"
6
7 # Define Parameters
8 n_digits = len(np.unique(df_encoded)) # Get Lenth of dataframe to get maximum cluster range
9
10 linkage = ['ward', 'complete', 'average', 'single']
11
12 # Loop by Parameters
13 for i in range(2,n_digits):
14     for l in linkage:
15         predicted_labels = AgglomerativeClustering(n_clusters=i,linkage=l)
16         clusters = predicted_labels.fit_predict(scaled_df)
17         sil = (metrics.silhouette_score(df_encoded,clusters, metric='euclidean').round(3)) # If linkage is "ward", only "euc
18         dbs = (metrics.davies_bouldin_score(df_encoded,clusters).round(3))
19         results = results.append({'Model': model_name, 'Clusters': i, 'Linkage':l, 'Silhouette Score': sil, 'Davies Bouldin
20         results.sort_values(by=['Silhouette Score'], inplace=True, ascending=False) # Sort by highest 'Silhouette Score'
21
22 results = results[results['Clusters'] != 0.0]
23
24 # Collect Best Parameters
25 best_cluster = int(results['Clusters'].iloc[0])
26 best_linkage = results['Linkage'].iloc[0]
27
28 model_AgglomerativeClustering = AgglomerativeClustering(n_clusters=best_cluster, linkage=best_linkage, affinity='euclidean')
29 model_AgglomerativeClustering.fit_predict(df_encoded) # Fit Model
30 predicted_labels_Hierarchical = model_AgglomerativeClustering.fit_predict(df_encoded) # Predict Data
31
32 sil = (metrics.silhouette_score(scaled_df,predicted_labels_Hierarchical).round(3))
33 dbs = (metrics.davies_bouldin_score(scaled_df,predicted_labels_Hierarchical).round(3))
34
35 # Capture Model Results & Append to Main Result Frame
36 results_Hierarchical = pd.DataFrame.from_dict({'Model':model_name,'Clusters':best_cluster,'Silhouette Score': sil,'Davies Bou
37 Combined_Results = Combined_Results.append(results_Hierarchical,ignore_index=True).replace("NaN","") # Append model results
38
39 print(colored("Total Possible Clusters: ", 'blue', attrs=['bold'])
40       + colored(n_digits, 'red', attrs=['bold']))
41       + colored("\n\nCluster Results:\n", 'blue', attrs=['bold'])
42       + colored(results.head(4), 'blue', attrs=['bold'])
43       + colored("\n\nCombined Model Results:\n", 'green', attrs=['bold'])
44       + colored(Combined_Results, 'magenta', attrs=['bold']))
45
46 # Plot Figure
47 import plotly.express as px
48 fig = px.scatter_3d(df_encoded, x='Age', y='Annual Income (k$)', z='Spending Score (1-100)', color=model_AgglomerativeCluste
49 fig.update_layout(coloraxis_colorbar=dict(yanchor="top", y=1, x=0, ticks="outside"))
50 # fig.update_layout is being used to make the 'color' legend shift towards left side, otherwise the 'color' and 'Gender' Leg

```

MODEL 3: Hierarchical/Agglomerative Clustering Algorithm RESULTS

Total Possible Clusters: 101

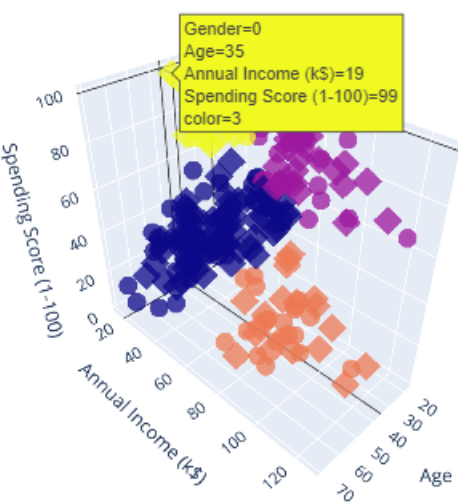
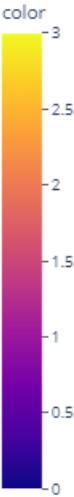
Cluster Results:

	Model	Clusters	Silhouette Score	Davies Bouldin Score	Linkage \
0	Hierarchical	4	0.358	1.157	ward
1	Hierarchical	3	0.290	1.193	ward
2	Hierarchical	2	0.258	1.501	ward
3	Hierarchical	5	0.249	1.304	complete

Model#
0 Model 3
1 Model 3
2 Model 3
3 Model 3

Combined Model Results:

Model#	Model	Clusters	Silhouette Score	Davies Bouldin Score	
0	Model 1	KMeans	6	0.453	0.750
1	Model 2	MiniBatchKMeans	6	0.452	0.746
2	Model 3	Hierarchical	4	0.225	1.509



Gender
● 1
◆ 0

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

DBSCAN Algorithm with optimal parameters, which groups closely packed points together, was employed.

MODEL 4: Density-Based Spatial Clustering of Applications with Noise (DBSCAN) Algorithm with Optimal Parameter Tuning

```

1 display_alert_color("MODEL 4: DBSCAN Algorithm RESULTS", "purple","warning")
2
3 results=pd.DataFrame(columns=['Model','Clusters','Silhouette Score','Davies Bouldin Score','Eps','Min_Samples'])
4
5 model_name = "DBSCAN"
6
7 n_digits = int(len(np.unique(df_encoded)) * 0.50) # Get half the length of dataframe to get fair cluster range
8 algorithm = ['auto','ball_tree','kd_tree','brute']
9
10 # Loop by Parameters
11 for i in range(1,n_digits):
12     for a in algorithm:
13         clusters = DBSCAN(eps=i*0.5, min_samples=1, algorithm=a).fit_predict(scaled_df)
14         if len(np.unique(clusters))>=2:
15             results=results.append({'Model':model_name, 'Algorithm':a, 'Eps':i*0.5,'Min_Samples':i,'Clusters':len(np.unique(clusters))})
16             results.sort_values(by=['Silhouette Score'], inplace=True, ascending=False) # Sort by highest 'Silhouette Score'
17
18 # Find Best Parameters
19 best_eps = results['Eps'].iloc[0]
20 best_min_samples = int(results['Min_Samples'].iloc[0])
21 best_clusters = results['Clusters'].iloc[0]
22 best_algorithm = results['Algorithm'].iloc[0]
23
24 # Make Model with Best Parameters
25 model_DBSCAN = DBSCAN(algorithm=best_algorithm, eps=best_eps, min_samples=best_min_samples)
26 predicted_labels_DBSCAN = model_DBSCAN.fit_predict(scaled_df)
27 sil = (metrics.silhouette_score(scaled_df,predicted_labels_DBSCAN).round(3))
28 dbs = (metrics.davies_bouldin_score(scaled_df,predicted_labels_DBSCAN).round(3))
29
30 # Capture Model Results & Append to Main Result Frame
31 results_DBSCAN =pd.DataFrame.from_dict({'Model':model_name, 'Clusters':best_clusters,'Silhouette Score': sil,'Davies Bouldin Score':dbs})
32 Combined_Results = Combined_Results.append(results_DBSCAN,ignore_index=True) # Append model results to main result dataframe
33
34 print(colored("Total Possible Clusters: ", 'blue', attrs=['bold']))
35     + colored(n_digits, 'red', attrs=['bold'])
36     + colored("\n\nCluster Results:\n", 'blue', attrs=['bold'])
37     + colored(results.head(4), 'blue', attrs=['bold'])
38     + colored("\n\nCombined Model Results:\n", 'green', attrs=['bold'])
39     + colored(Combined_Results, 'magenta', attrs=['bold'])
40
41 # Plot Figure (use import plotly.express as px)
42 fig = px.scatter_3d(df_encoded, x='Age', y='Annual Income (k$)', z='Spending Score (1-100)', color=predicted_labels_DBSCAN,
43 fig.update_layout(coloraxis_colorbar=dict(yanchor="top", y=1, x=0, ticks="outside"))
44 # fig.update_layout is being used to make the 'color' legend shift towards left side, otherwise the 'color' and 'Gender' Leg
45 # fig.show()
```


MODEL 4: DBSCAN Algorithm RESULTS

Total Possible Clusters: 50

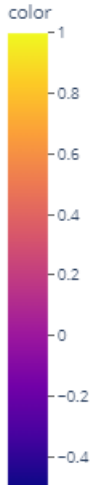
Cluster Results:

	Model	Clusters	Silhouette Score	Davies Bouldin Score	Eps	Min_Samples	\
0	DBSCAN	2	0.275464	1.614761	1.5	3	
3	DBSCAN	2	0.275464	1.614761	2.0	4	
1	DBSCAN	64	0.182812	0.501533	0.5	1	
2	DBSCAN	5	0.153174	1.978556	1.0	2	

	Algorithm	Model#
0	brute	Model 4
3	brute	Model 4
1	brute	Model 4
2	brute	Model 4

Combined Model Results:

	Model#	Model	Clusters	Silhouette Score	Davies Bouldin Score
0	Model 1	KMeans	6	0.453	0.75
1	Model 2	MiniBatchKMeans	6	0.452	0.746
2	Model 3	Hierarchical	4	0.225	1.509
3	Model 4	DBSCAN	2	0.275	1.615



5) Ordering Points to Identify the Clustering Structure (OPTICS) Algorithm:

Ordering Points to Identify the Clustering Structure (OPTICS) Algorithm with optimal parameters, which orders data points so that spatially closest points become neighbours in the ordering, was employed.

MODEL 5: Ordering Points To Identify the Clustering Structure (OPTICS) Algorithm with Optimal Parameter Tuning

```

1  display_alert_color("MODEL 5: Ordering Points To Identify the Clustering Structure (OPTICS) Algorithm RESULTS", "purple", "wa
2  results = pd.DataFrame(columns=['Covariance Type', 'Clusters', 'Silhouette Score', 'Davies Bouldin Score'])
3
4  model_name = "OPTICS"
5  model = 'Model 5'
6
7  #Define Parameters
8  algorithm = ['auto', 'ball_tree', 'kd_tree', 'brute']
9
10 n_digits = int(len(np.unique(df_encoded))* 0.10) # Get half the Length of dataframe to get fair cluster range # scaled_df
11 n_clusters= np.arange(2,n_digits)
12 print(n_digits)
13
14 # Loop by Parameters
15 for a in algorithm:
16     for j in n_clusters:
17         OPTICS_cluster=OPTICS(min_cluster_size=j, algorithm=a)
18         clusters= OPTICS_cluster.fit_predict(df_encoded)
19
20 if len(np.unique(clusters))>=2:
21     results=results.append({'Model#': model, 'Model': model_name, "Algorithm": a, "Clusters":j, "Silhouette Score":metrics.sil
22     results.sort_values(by=['Silhouette Score'], inplace=True, ascending=False) # Sort by highest 'Silhouette Score'
23
24 # Find Best Parameters
25 best_algorithm = results['Algorithm'].iloc[0]
26 best_clusters = int(results['Clusters'].iloc[0]) #j
27
28 # Make Model with Best Parameters
29 model_OPTICS = OPTICS(p=best_clusters, algorithm=best_algorithm).fit_predict(df_encoded)
30 predicted_labels_OPTICS = model_OPTICS.fit_predict(df_encoded)
31 pred5 = predicted_labels_OPTICS.copy()
32 sil = (metrics.silhouette_score(scaled_df, predicted_labels_OPTICS).round(3))
33 dbs = (metrics.davies_bouldin_score(scaled_df, predicted_labels_OPTICS).round(3))
34
35 # Capture Model Results & Append to Main Result Frame
36 results_OPTICS = pd.DataFrame.from_dict({'Model#': model, 'Model': model_name,
37     'Clusters':best_clusters,
38     'Silhouette Score': sil,
39     'Davies Bouldin Score': dbs}, orient='index').T # Make Dataframe & Transpose
40 Combined_Results = Combined_Results.append(results_OPTICS,ignore_index=True) # Append model results to main result dataframe
41
42 print(colored("Total Possible Clusters: ", 'blue', attrs=['bold']))
43     + colored(n_digits, 'red', attrs=['bold'])
44     + colored("\n\nCluster Results:\n", 'blue', attrs=['bold'])
45     + colored(results.head(4), 'blue', attrs=['bold'])
46     + colored("\n\nCombined Model Results:\n", 'green', attrs=['bold'])
47     + colored(Combined_Results, 'magenta', attrs=['bold'])
48
49 # Plot Figure (use import plotly.express as px)
50 fig = px.scatter_3d(df_encoded, x='Age', y='Annual Income (k$)', z='Spending Score (1-100)', color=predicted_labels_OPTICS,
51 fig.update_layout(coloraxis_colorbar=dict(yanchor="top", y=1, x=0, ticks="outside"))
52 # fig.update_layout is being used to make the 'color' legend shift towards left side, otherwise the 'color' and 'Gender' Leg
53 # fig.show()
```

MODEL 5: Ordering Points To Identify the Clustering Structure (OPTICS) Algorithm RESULTS

10

Total Possible Clusters: 10

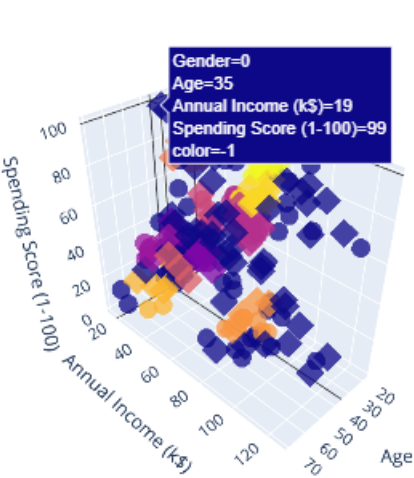
Cluster Results:

	Covariance	Type	Clusters	Silhouette Score	Davies Bouldin Score	Algorithm	\
0		NaN	9	-0.05971	3.094712	brute	

	Model	Model#
0	OPTICS	Model 5

Combined Model Results:

	Model#	Model	Clusters	Silhouette Score	Davies Bouldin Score
0	Model 1	KMeans	6	0.453	0.75
1	Model 2	MiniBatchKMeans	6	0.452	0.746
2	Model 3	Hierarchical	4	0.225	1.509
3	Model 4	DBSCAN	2	0.275	1.615
4	Model 5	OPTICS	9	-0.126	2.591



5) Result Summary and Recommended Model

5a) Result Summary

Result Summary is given below:

	Model	Clusters	Silhouette Score	Davies Bouldin Score
1	KMeans	6	0.453	0.75
2	MiniBatchKMeans	6	0.453	0.768
4	DBSCAN	2	0.275	1.615
3	Hierarchical	4	0.225	1.509
5	OPTICS	49	-0.107	2.505

5b) Model Choice and Justification

Recommended Model and Justification:

Model Scoring has been carried out using both Silhouette and Davies Bouldin Scores, although, you may use only one of them.

The Silhouette Score or Silhouette Coefficient is a metric employed to compute goodness of a clustering technique which aims to gauge similarity of a datapoint to other datapoints in a given cluster relative to datapoints outside its cluster.

The value of Silhouette Score ranges from -1 to 1, where a score of:

- 1 suggests clusters are well apart from each other and plainly distinguishable.
- 0 suggests the distance between clusters is insignificant and so clusters are indifferent.
- 1 suggests incorrect cluster assignment.

Therefore, when scoring an unsupervised model with Silhouette Score, a higher score would imply a better goodness of fit. Hence, we will select the model with Maximum Silhouette Score.

Alternatively, Davies Bouldin Score may be used which measures the average similarity of every cluster with a cluster most similar to it.

Hence, lower average similarity indicates better cluster separation and model performance.

In this case, the Model KMeansClustering is yielding Maximum Silhouette Score of 0.453 and Minimum Davies Bouldin Score of 0.75.

Hence, we recommend this model for Mall Customer Segmentation.

6) Summary Key Findings and Insights

Model figures clearly show age, annual income and spending score values for each cluster which aided in gaining deeper insight into cluster characteristics. Since age is scattered across all clusters, it does not seem to be a distinctive feature. Consequently, based on further descriptive analysis, cluster descriptions were assigned as follows:

6a) Cluster Description

```

1 display_alert_color("6) SUMMARY KEY FINDINGS & INSIGHTS", "darkblue","info")
2 display_alert_color("Extract Cluster Information", "darkgreen","warning")
3
4 model = Combined_Results['Model#'].iloc[1] # Get Model Number
5
6 # Get Target Variable Predictions
7 if model == 'Model 1':#
8     prediction = pred1.copy()
9 elif model == 'Model 2':
10    prediction = pred2.copy()
11 elif model == 'Model 3':
12    prediction = pred3.copy()
13 elif model == 'Model 4':
14    prediction = pred4.copy()
15 elif model == 'Model 5':#
16    prediction = pred5.copy()
17 else:
18     print("")
19 no_clusters = (np.unique(prediction) )
20
21 print(colored("Unique Cluster List: ", 'blue', attrs=['bold']))
22     + colored(list(no_clusters), 'red', attrs=['bold']))
23
24 display_alert_color("Interpretation of Cluster Information", "darkgreen","warning")
25
26 # Decode encoded fields
27 df_interpret = df_encoded.copy()
28 df_interpret['Gender'] = df_interpret['Gender'].map({0: 'Female', 1: 'Male'}) # Decode Encoded Data
29 df_interpret['labels'] = prediction
30 #df_interpret = df_interpret.sort_values(["labels","Annual Income (k$)", "Spending Score (1-100)"],ascending = [True, True,
31
32 df_interpret['label description'] = df_interpret['labels'].map({
33     0: 'low income - medium spending', # blue cluster
34     1: 'upper high income - high spending', # purple cluster
35     2: 'medium income - medium spending', # pink cluster
36     3: 'high income - low spending', # orange cluster
37     4: 'high income - high spending', # yellow cluster
38     5: 'low income - high spending'}) # yellow cluster
39
40 df_interpret.head(5)

```

6b) Cluster Information Assignment

Extract Cluster Information

Unique Cluster List: [0, 1, 2, 3, 4, 5]

Interpretation of Cluster Information

	Gender	Age	Annual Income (k\$)	Spending Score (1-100)	labels	label description
0	Male	19	15	39	0	low income - medium spending
1	Male	21	15	81	3	high income - low spending
2	Female	20	16	6	0	low income - medium spending
3	Female	23	16	77	3	high income - low spending
4	Female	31	17	40	0	low income - medium spending

6c) Cluster Assignment Result

Cluster 0:

	Gender	Age	Annual Income (k\$)	Spending Score (1-100)	labels	\
0	Male	19	15	39	0	

label description

0 low income - medium spending

Cluster 1:

	Gender	Age	Annual Income (k\$)	Spending Score (1-100)	labels	\
155	Female	27	78	89	1	

label description

155 upper high income - high spending

Cluster 2:

	Gender	Age	Annual Income (k\$)	Spending Score (1-100)	labels	\
66	Female	43	48	50	2	

label description

66 medium income - medium spending

Cluster 3:

	Gender	Age	Annual Income (k\$)	Spending Score (1-100)	labels	\
33	Male	18	33	92	3	

label description

33 high income - low spending

Cluster 4:

	Gender	Age	Annual Income (k\$)	Spending Score (1-100)	labels	\
138	Male	19	74	10	4	

label description

138 high income - high spending

Cluster 5:

	Gender	Age	Annual Income (k\$)	Spending Score (1-100)	labels	\
65	Male	18	48	59	5	

label description

65 low income - high spending

7) Future Recommendations

Despite its novel approach to automatically pick best model with best parameters, the project is not without its shortcomings.

To begin with, it only utilized a very small dataset with merely five features.

Hence, it may have discounted other important factors that may impact customer segmentation.

Furthermore, the project could have employed other unsupervised models to ensure even more improved results.

Future recommendations are, hence, as follows:

- 1) Use a larger dataset with more features.
- 2) Employ more unsupervised models like MeanShift, Birch etc.
- 3) Try other evaluation metrics like Calinski-Harabasz Index or Adjusted Rand Index and see if there is a difference in results.

8 Useful Links

8a) Link to Other Useful Models

- <https://medium.com/@mbektas/customer-segmentation-with-clustering-algorithms-in-python-be2e021035a>
- <https://www.kaggle.com/code/aayush7kumar/clustering-using-k-means-hierarchical-and-dbscan/notebook>
- https://github.com/eklavyasaxena/IBM-Unsupervised-Machine-Learning-Capstone-k-Means-Clustering/blob/master/Exploring-the-Taste-of-NYC-Neighborhoods/Project_Notebook.ipynb
- https://goodboychan.github.io/python/machine_learning/natural_language_processing/vision/2020/10/26/01-K-Means-Clustering-for-Imagery-Analysis.html

8b) Model Evaluation and Scoring

- <https://towardsdatascience.com/how-to-evaluate-unsupervised-learning-models-3aa85bd98aa2>
- <https://medium.com/@mbektas/customer-segmentation-with-clustering-algorithms-in-python-be2e021035a>
- <https://towardsdatascience.com/cheat-sheet-to-implementing-7-methods-for-selecting-optimal-number-of-clusters-in-python-898241e1d6ad>

8c) SKLEARN Unsupervised Model Types and their Parameters

- https://scikit-learn.org/stable/unsupervised_learning.html
- <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>
- <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html>

9) Github Link to Assignment Notebook

<https://github.com/FATIMASP/IBM-MACHINE-LEARNING-CERTIFICATION/blob/main/Unsupervised%20Machine%20Learning/UNSUPERVISED%20LEARNING%20MALL%20CUSTOMERS%20FINAL%20MODEL.ipynb>