IBM MACHINE LEARNING

UNSUPERVISED MACHINE LEARNING MODELS FOR MALL CUSTOMER SEGMENTATION

Fatima, Sayeda

10/23/2022

Table of Contents

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 characteristicescan 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&currency=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

```
customer_data = pd.read_csv(input_file_path)
display_alert_color("Load and Read Dataset","darkbrown" ,"warning")
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 × 5 columns

- Column types were then explored

```
display_alert_color("Display Data Info","darkbrown" ,"warning")
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

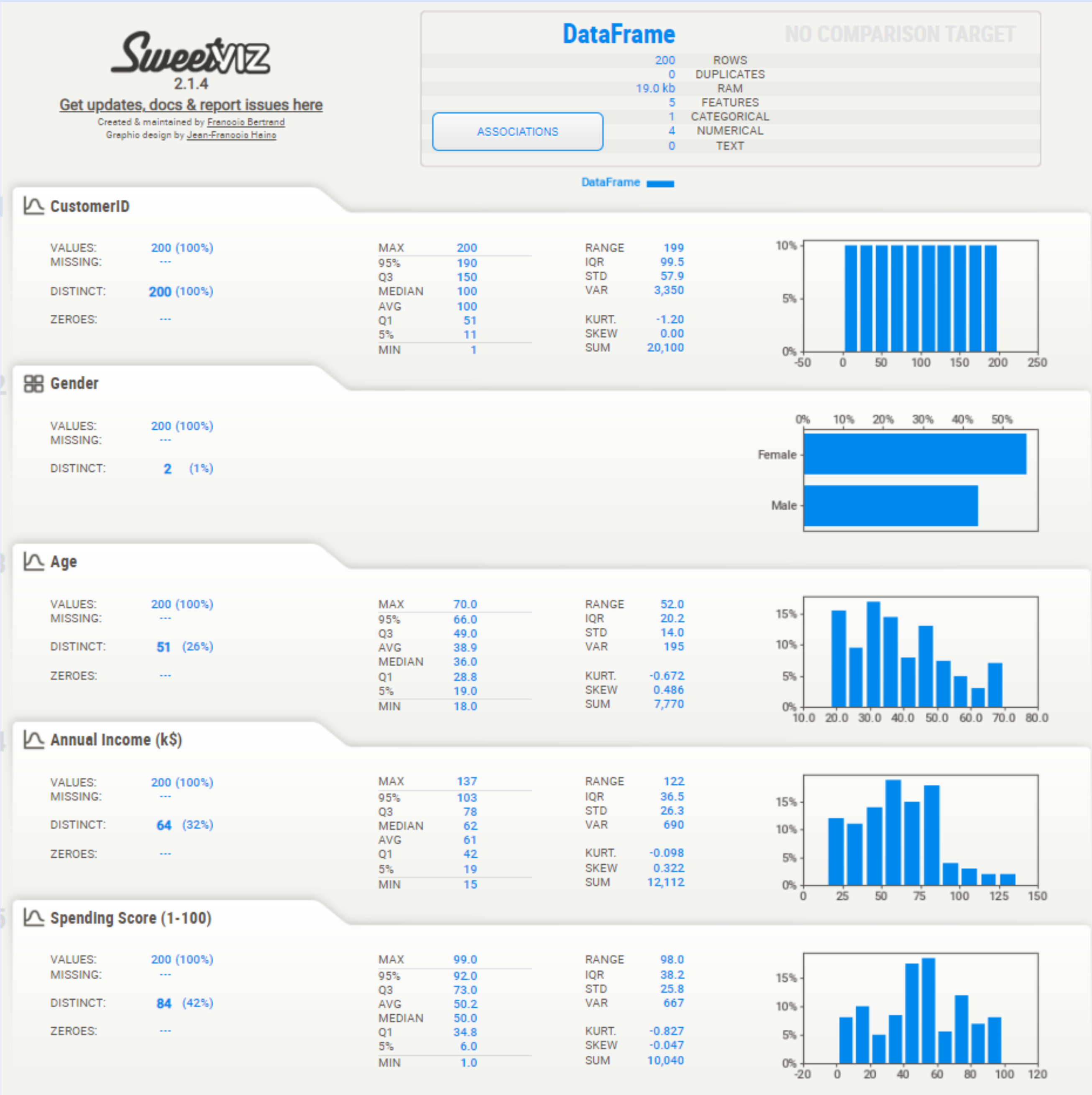
```
display_alert("Initial Exploratory Data Analysis", "success")
display_alert_color("Perform Quick EDA: To see dataset’s distribution and its dispersion. ","teal", "warning")
df_eda = sv.analyze(customer_data) # Use Sweetviz for Automated EDA
#df_eda.show_html() # Uncomment to Generate Online Report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

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

```python
def drop_cols(*name):
    n = name # Extract Dataframe by Name...this will create a 3d tuple
    n = (n[0]) # Convert Tuple to To Dataframe
    df_name = [x for x in globals() if globals()[x] is n][0] # Extract Name of Imported Dataframe

    n = n.fillna(0) #Fill Remaining Missing Values with Zero
    # Find Mean of Null, Nan and Zero Values Before Any Drops
    m0 = n.isin([' ','NULL','NaN', 0]).mean().sort_values(axis=0, ascending=False, inplace=False, kind='quicksort', na_posit

    #  1) Drop Columns with Unique Values Less than threshold
    unique_counts = pd.DataFrame.from_records([(col, n[col].nunique()) for col in n.columns], # get unique counts
                          columns=['Column_Name', 'Unique']).sort_values(by=['Unique'])
    unique = unique_counts[(unique_counts['Unique'] < 2)] #If threshold is lesss than 2 then
    drop1 = (unique['Column_Name'].tolist()) # First List of columns to drop

    print(colored("\nDrop 1: ", 'blue', attrs=['bold'])
          +colored(drop1, 'magenta', attrs=['bold']))

    # 2) Drop Highly Skewed & Low Correlation Columns with target
    drop2 = analyze[(analyze[column] != 1.000000) & ((analyze['skewness'] > 1) & (analyze[column] < 0))]
    print(colored("\nDrop 2: \n ", 'blue', attrs=['bold'])
          +colored(drop2, 'magenta', attrs=['bold']))

    drop2 = drop2.sort_values(by='Columns', ascending=True)
    drop2 = drop2['Columns'].tolist() # Second List of columns to drop

    drop = drop1 + drop2 + delete_features # Final List of columns to drop
    print(colored("\nFinal Column Drop List: ", 'blue', attrs=['bold'])
          +colored(drop, 'magenta', attrs=['bold']))

    if target in drop: # Remove Target from List
        drop = drop.remove(target)
    else:
        print(target, " Not Found.")

    n = n.drop(columns=[col for col in n if col in drop]) # Drop Dataframe Columns if in List

    # Find Mean of Null, Nan and Zero Values Before Dropping
    m1 = n.isin([' ','NULL','NaN', 0]).mean().sort_values(axis=0, ascending=False, inplace=False, kind='quicksort', na_posit

    # 3) Drop Columns With High Nan Values
    #drop_thresh = .90 # Identify Drop Threshold
    #n = n.loc[:, df.isin([' ','NULL', 'NaN',0]).mean() > drop_thresh] # drop columns if Mean is > 0.90

    n = n.fillna(0) #Fill Remaining Missing Values with Zero
    #n = n.replace(["NaN"], 0).sort_values(by=target, ascending=False) # Replace all Nan Values with Zero

    # Find Mean of Null, Nan and Zero Values After Dropping
    m2 = n.isin([' ','NULL','NaN']).mean().sort_values(axis=0, ascending=False, inplace=False, kind='quicksort', na_position

    #Print Results
    print(colored("\nDataframe Average Null Values Before Any Drops\n ", 'blue', attrs=['bold'])
          +colored(m0, 'magenta', attrs=['bold'])
          +colored("\n\n Low Distict Columns to Drop: ", 'green', attrs=['bold'])
          + colored(drop1, 'red', attrs=['bold'])
          +colored("\n\nDataframe Average Null Values After Dropping Highly Skewed Columns\n ", 'green', attrs=['bold'])
          +colored(m1, 'red', attrs=['bold'])
          +colored("\n\n Drop Columns if Mean is > 0.90 \n", 'green', attrs=['bold'])
          + colored("\nDataframe Average Null Values After Drop and 'Nan' Replacement\n", 'blue', attrs=['bold'])
          +colored(m2, 'magenta', attrs=['bold'])
          +colored(type(m2), 'magenta', attrs=['bold'])
         )
    return n

# Return Function
n = drop_cols(df)
df = n.copy()
#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:

```python
# Method to encode object/string columns
def encoder(*name):
    # Accept an argument, return a value.
    n = name # Extract Dataframe by Name...this will create a 3d tuple
    n = (n[0]) # Convert Tuple to To Dataframe
    df_name = [x for x in globals() if globals()[x] is n][0] # Extract Name of Imported Dataframe

    # 1) List all Object/String Columns
    from sklearn import preprocessing
    cat_columns = n.select_dtypes(include=[object]) # Get Object Type Columns to Convert to Encoded Categories

    categorical_column = list(cat_columns.columns)# list of columns to for label encoding

    print(colored("\n\nColumns Requiring Encoding: \n", 'blue', attrs=['bold'])
                + colored(categorical_column, 'green', attrs=['bold']))

    # Make Empty Dataframe to decode encoded data later
    decode_features = pd.DataFrame()

    ##### Employ Scikit-learn label encoding to encode object data #####
    lab_enc = preprocessing.LabelEncoder()
    for col in categorical_column:
            n[col] = lab_enc.fit_transform(n[col])
            name_mapping = dict(zip(lab_enc.classes_, lab_enc.transform(lab_enc.classes_)))

            ##### Decode Encoded Data #####
            feature_df = pd.DataFrame([name_mapping])
            feature_df = feature_df.astype(str)
            feature_df= (col + "_" + feature_df.iloc[0:])
            feature_df["Feature"] = col
            decode_features = decode_features.append(feature_df)# Append Dictionaries to Empty Dataframe for Later Decoding

            ##### Print Encoded Data #####
            print(colored("Feature: \n", 'blue', attrs=['bold'])
                  + colored(col, 'red', attrs=['bold'])
                  + colored("\nMapping: \n", 'blue', attrs=['bold'])
                  + colored(name_mapping, 'green', attrs=['bold'])
                  + colored("\n\nType n: ", 'blue', attrs=['bold'])
                  + colored(type(n), 'magenta', attrs=['bold'])
                 )
    n.head(3)

    ##### 2) Make Decoded Factor Dataframe with Description #####
    factor_list = decode_features.T # Transpose Dataframe and place in new dataframe
    factor_list = factor_list.replace(np.nan, "/") # nan values with forward slash
    factor_list["Factors"] = factor_list.astype(str).agg("".join,axis=1).replace(r'[^\w\s]|/', '', regex=True) # Aggregate A
    factor_list.reset_index() # Reset index before copying/assigning it to a new column
    factor_list['Description'] = factor_list.index  # Assign index to column

    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)

```python
def outliers(*name):
    n = name # Extract Dataframe by Name...this will create a 3d tuple
    n = (n[0]) # Convert Tuple to To Dataframe
    df_name = [x for x in globals() if globals()[x] is n][0] # Extract Name of Imported Dataframe

    cols = n.columns # All Columns

    # Numeric Columns
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    numeric_cols = n.select_dtypes(include=numerics)
    numeric_cols = numeric_cols.columns.tolist()

    # Object Columns
    categorical_cols = list(set(cols) - set(numeric_cols))

    # Skewed Columns
    skewed_cols = analyze[(analyze['skewness'] > 0) | (analyze['skewness'] < 0)]
    skewed_cols = skewed_cols['Columns'].tolist()
    skewed_cols.remove(target) # Remove target column

    # Replace Outliers
    for col in n.columns:
        if col in skewed_cols:
            print(colored(col, 'magenta', attrs=['bold'])
                  + colored(" is Skewed... ", 'blue', attrs=['bold'])
                  )

            mode = n[col].mode()
            mode = mode[0]

            if col in numeric_cols:
                print(colored(col, 'magenta', attrs=['bold'])
                      + colored(" Column Type is: ", 'blue', attrs=['bold'])
                      + colored(n[col].dtypes, 'red', attrs=['bold'])
                      )
                #Calculate quantiles and IQR
                Q1 = n[col].quantile(0.25) # Same as np.percentile but maps (0,1) and not (0,100)
                Q3 = n[col].quantile(0.75)
                IQR = Q3 - Q1
                # Replace with Mode
                n[col] = np.where((n[col] < (Q1 - 1.5 * IQR)) | (n[col] > (Q3 + 1.5 * IQR)), mode, n[col])

                print(colored("\nReplaced ", 'blue', attrs=['bold'])
                      +colored(col, 'magenta', attrs=['bold'])
                      + colored(" Skewed Values by Mode: ", 'blue', attrs=['bold'])
                      + colored(mode, 'red', attrs=['bold'])
                      + colored('\n', 'magenta', attrs=['bold'])
                      + colored((n[col]), 'green', attrs=['bold'])
                      )

            else:
                print("")
    df_transformed = n.copy()
    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 Clusturing algorithms use distance metric to make categories, hence, data scaling is recommended.

```
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df_encoded)
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) Machine Learning Algorithm Approaches

Although, While data level and algorithm ensemble approaches do exist for dealing with imbalanced datasets, nevertheless, an automated optimal parameter search method was created to achieve best class reweighting along with isolating other optimal model parameters. This approach was employed because best hyper-parameters are not automatically learnt within estimators and its manual search not only slows down model development but may also lead to ineffective model construction. Hence, exhaustive cv grid search approach was used to pass parameter arguments to the constructor in order to find optimal parameters for each model.

Logistic Regression (LR) Models

Grid Search Method to Find 'Best Parameters'to 'Build Logistic Regression WITH Best Class Weights'

```
# Grid Search Method to find Best Hyperparameters for a Logistic Regression Model
def grid_search_lr(X_train, y_train):
    # Parameters
    params_grid = {
    'class_weight': [{0:0.1, 1:0.9}, {0:0.2, 1:0.8}, {0:0.3, 1:0.7}],
    'solver': ['lbfgs', 'saga', 'liblinear', 'newton-cg', 'sag']
    }
    # LR Model
    lr_model = LogisticRegression(random_state=rs, max_iter=1000)

    # Search Best Parameters
    grid_search = GridSearchCV(estimator = lr_model,
                               param_grid = params_grid,
                               scoring='f1',
                               cv = 5, verbose = 1)
    # Train Model with Best Parameters
    grid_search.fit(X_train, y_train)

    # Get Best/optimal parameters
    best_lrparams = grid_search.best_params_
    return best_lrparams
```

Get Optimal Parameters for LR Model using Grid Search LR Method above

```
final_lrprams = grid_search_lr(X_train, y_train) # From the cell above, Call grid_search_rf(X_train, y_train)

final_lrprams_df = pd.DataFrame([final_lrprams]) # Dictionary To dataframe
print(final_lrprams_df)

# Make Optimal Variables
optimal_lr_class_weight = (final_lrprams_df.at[0,'class_weight'])
optimal_solver = (final_lrprams_df.at[0,'solver'])
print('Optimal LR Class Weights: ', optimal_lr_class_weight)
print('Optimal Solver: ', optimal_solver)

# Define Optimal Parameters
optimal_lr_params = {'class_weight': optimal_lr_class_weight, 'solver': optimal_solver}
print(optimal_lr_params)
```

```
Fitting 5 folds for each of 15 candidates, totalling 75 fits
       class_weight     solver
0  {0: 0.2, 1: 0.8}  newton-cg
Optimal LR Class Weights:  {0: 0.2, 1: 0.8}
Optimal Solver:  newton-cg
{'class_weight': {0: 0.2, 1: 0.8}, 'solver': 'newton-cg'}
```

Random Forest (RF) Models

Grid Search Method to Find 'Best Parameters'to 'Build Random Forest WITH Class Weights'

```
# Method for Grid Search Hyperparameters for a Random Forest Model
def grid_search_rf(X_train, y_train):
    # Parameters
    params_grid = {
    'max_depth': [2*n+1 for n in range(10) ], #[5, 10, 15, 20],
    'n_estimators': [2*n+1 for n in range(20)], #[25, 50, 100],
    'min_samples_split': [2, 5],
    'class_weight': [{0:0.1, 1:0.9}, {0:0.2, 1:0.8}, {0:0.3, 1:0.7}]
    }
    # RF Model
    rf_model = RandomForestClassifier(random_state=rs)

    # Search Best Parameters
    grid_search = GridSearchCV(estimator = rf_model,
                               param_grid = params_grid,
                               scoring='f1',
                               cv = 5, verbose = 1)
    # Train Model with Best Parameters
    grid_search.fit(X_train, y_train)

    # Get Best/optimal parameters
    best_params = grid_search.best_params_
    #Best_Score = grid_search.best_score_
    #accuracy = get_accuracy(X_train, X_test, y_train, y_test, grid_search.best_estimator_)
    return best_params
```

Get `Optimal Parameters` for RF Model using Grid Search RF Method above ¶

```
#Calculate StartTime to Measure Script Execution Time at the End of Script
start_time = datetime.now()

#Get Optimal Parameters for RF Model using "grid_search_rf" Method to Find 'Best/Optimal Parameters'
best_params = grid_search_rf(X_train, y_train) # From the cell above, Call grid_search_rf(X_train, y_train)

best_params_df = pd.DataFrame([best_params]) # Dictionary To dataframe
print(best_params_df)

# Make Optimal Parameter Variables
optimal_class_weight = (best_params_df.at[0,'class_weight'])
print(optimal_class_weight)
optimal_max_depth = (best_params_df.at[0,'max_depth'])
print(optimal_max_depth)
optimal_min_samples_split = (best_params_df.at[0,'min_samples_split'])
print(optimal_min_samples_split)
optimal_n_estimators = (best_params_df.at[0,'n_estimators'])
print(optimal_n_estimators)

# Define Optimal Parameters
optimal_rf_params = {'bootstrap': True,
                     'class_weight': optimal_class_weight,
                     'max_depth': optimal_max_depth,
                     'min_samples_split': optimal_min_samples_split,
                     'n_estimators': optimal_n_estimators}
print(optimal_rf_params)

#Print Total Execution Time
print('Model Execution Time: ', datetime.now() - start_time)
```

```
Fitting 5 folds for each of 1200 candidates, totalling 6000 fits
       class_weight  max_depth  min_samples_split  n_estimators
0  {0: 0.1, 1: 0.9}          7                  5            23
{0: 0.1, 1: 0.9}
7
5
23
{'bootstrap': True, 'class_weight': {0: 0.1, 1: 0.9}, 'max_depth': 7, 'min_samples_split': 5, 'n_estimators': 23}
```

eXtreme Gradient Boosting (XGB) Model

Grid Search Method to Find 'Best Parameters'to 'Build XGB WITH Best Class Weights'

```
#Calculate StartTime to Measure Model Execution Time at the End
xgb_start_time = datetime.now()

# Method for Grid Search Hyperparameters for a Random Forest Model
def grid_search_xgb(X_train, y_train):
    # Parameters
    params_grid = {
    'max_depth': [5, 10, 15, 20], #[2*n+1 for n in range(10) ],
    'n_estimators': [100, 300, 500],#[2*n+1 for n in range(20)],
    'min_samples_split': [2, 5, 8],
    }
    # RF Model
    xgb_model = GradientBoostingClassifier(random_state=rs)

    # Search Best Parameters
    grid_search = GridSearchCV(estimator = xgb_model,
                               param_grid = params_grid,
                               scoring='f1',
                               cv = 5, verbose = 1)
    # Train Model with Best Parameters
    grid_search.fit(X_train, y_train)

    # Get Best/optimal parameters
    best_params = grid_search.best_params_
    #Best_Score = grid_search.best_score_
    #accuracy = get_accuracy(X_train, X_test, y_train, y_test, grid_search.best_estimator_)
    return best_params
```

Get `Optimal Parameters` for XGB Model using Grid Search XGB Method above

```
xgb_params = grid_search_xgb(X_train, y_train) # From the cell above, Call grid_search_xgb(X_train, y_train)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

```
xgb_params_df = pd.DataFrame([xgb_params]) # Dictionary To dataframe
print(xgb_params_df)

# Make Optimal Parameter Variables
xgb_opt_max_depth = (xgb_params_df.at[0,'max_depth'])
print(xgb_opt_max_depth)
xgb_opt_min_samples_split = (xgb_params_df.at[0,'min_samples_split'])
print(xgb_opt_min_samples_split)
xgb_opt_n_estimators = (xgb_params_df.at[0,'n_estimators'])
print(xgb_opt_n_estimators)

# Define Optimal Parameters
xgb_optimal_params = {#'bootstrap': True,
                      #'class_weight': xgb_opt_class_weight,
                      'max_depth': xgb_opt_max_depth,
                      'min_samples_split': xgb_opt_min_samples_split,
                      'n_estimators': xgb_opt_n_estimators}
print(xgb_optimal_params)
```

```
   max_depth  min_samples_split  n_estimators
0          5                  2           500
5
2
500
{'max_depth': 5, 'min_samples_split': 2, 'n_estimators': 500}
```

Additionally, the following approaches were also combined with optimal parameters to find a model with best scores:

I) Data Level Approaches:

i) Synthetic Minority Over-sampling Technique (SMOTE):

Due to highly imbalance class distribution, employee data contains very few instances of minority class for any classification model to explicitly learn decision boundary. A popular approach to tackle this problem is oversampling minority class examples which are close in the feature space using SMOTE. This approach allowed us to achieve a balanced class distribution.

ii) Random under-sampling:

Using random under-sampling examples from majority class were deleted to achieve a balanced class distribution.

iii) Random over-sampling:

Random oversampling was employed to duplicate examples from minority class to achieve a balanced class distribution.

II) Algorithm Ensemble Approach:

i) Boosting:

Using boosting, a sequential aggregate of base classifier was created on weighted versions of training data set which focused on misclassified samples at every stage of creating new classifiers based on sample weights that were altered as per classifier's performance. Boosting was achieved using XGB Classifier.

4b) Summarizing Employed Models

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

1) K-Means Clustering Algorithm:

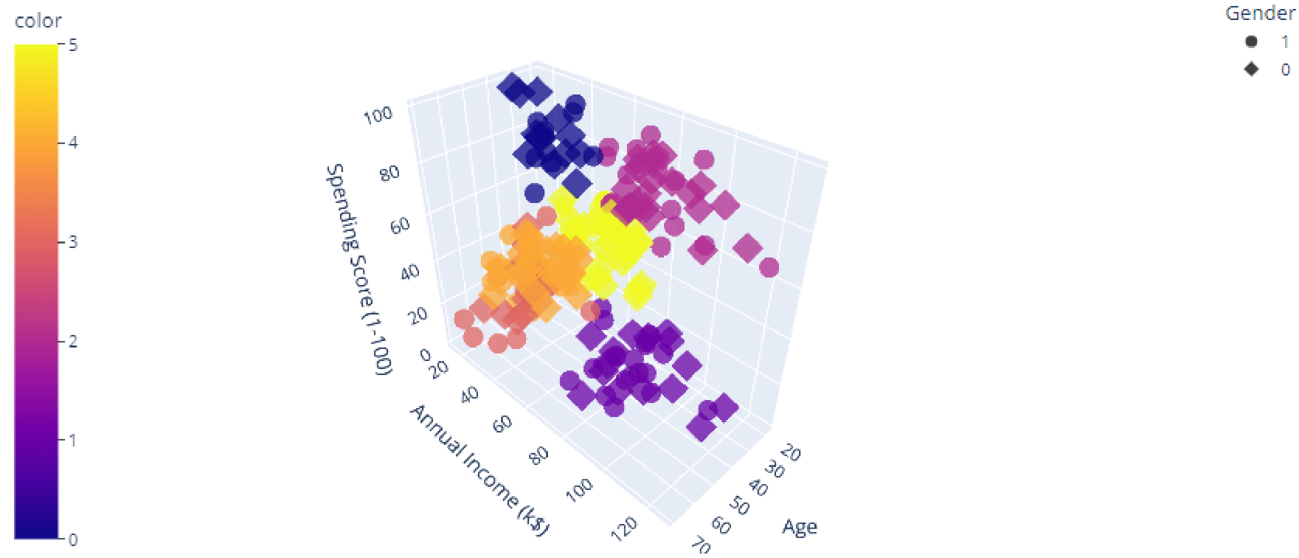
Due to the focus on segmentation, the popular K-Means Clustering Algorithm with optimal parameters was employed.

MODEL 1: KMeans Clustering Algorithm with Optimal Parameter Tuning

```
display_alert_color("MODEL1: KMeans Clustering Algorithm RESULTS", "purple","warning")

results=pd.DataFrame(columns=['Model','Clusters','Silhouette Score','Davies Bouldin Score'])

model_name = "KMeans"

# Define Parameters
n_digits = len(np.unique(df_encoded)) # Get lenth of dataframe to get maximum cluster range
algorithm = ["lloyd","elkan","auto","full"]

# Loop by Parameters
for i in range(2,n_digits):
    for a in algorithm:
        kmeans_labels = KMeans(n_clusters=i,algorithm=a, random_state=123).fit_predict(df_encoded)
        sil = (metrics.silhouette_score(df_encoded,kmeans_labels, metric='euclidean').round(3))
        dbs = (metrics.davies_bouldin_score(df_encoded,kmeans_labels).round(3))
        results = results.append({'Cluster': i, 'Algorithm':a, 'Silhouette Score': sil, 'Davies Bouldin Score': dbs}, ignore
        results.sort_values(by=['Silhouette Score'], inplace=True, ascending=False) # Sort by highest 'Silhouette Score'

best_cluster = int(results['Cluster'].iloc[0])
best_algorithm = results['Algorithm'].iloc[0]

model_KMeans = KMeans(n_clusters=best_cluster, algorithm=best_algorithm)
model_KMeans.fit(df_encoded) # Fit Model
predicted_labels_KMeans = model_KMeans.fit_predict(df_encoded) # Predict Data

sil = (metrics.silhouette_score(df_encoded,predicted_labels_KMeans).round(3))
dbs = (metrics.davies_bouldin_score(df_encoded,predicted_labels_KMeans).round(3))

# Capture Model Results & Append to Main Result Frame
results_KMeans =pd.DataFrame.from_dict({'Model':model_name,'Algorithm':best_algorithm,'Clusters':best_cluster,'Silhouette Sc
Combined_Results = Combined_Results.append(results_KMeans,ignore_index=True) # Append model results to main result dataframe

print(colored("Total Possible Clusters: ", 'blue', attrs=['bold'])
      + colored(n_digits, 'red', attrs=['bold'])
      + colored("\n\nCluster Results:\n", 'blue', attrs=['bold'])
      + colored(results.head(4), 'blue', attrs=['bold'])
      + colored("\n\nCombined Model Results:\n", 'green', attrs=['bold'])
      + colored(Combined_Results, 'magenta', attrs=['bold']))

# Plot Figure
import plotly.express as px
fig = px.scatter_3d(df_encoded, x='Age', y='Annual Income (k$)', z='Spending Score (1-100)', color=model_KMeans.fit_predict(
fig.update_layout(coloraxis_colorbar=dict(yanchor="top", y=1, x=0, ticks="outside"))
# fig.update_layout is being used to make the 'color' legend shift towards left side, otherwise the 'color' and 'Gender' leg
# 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:

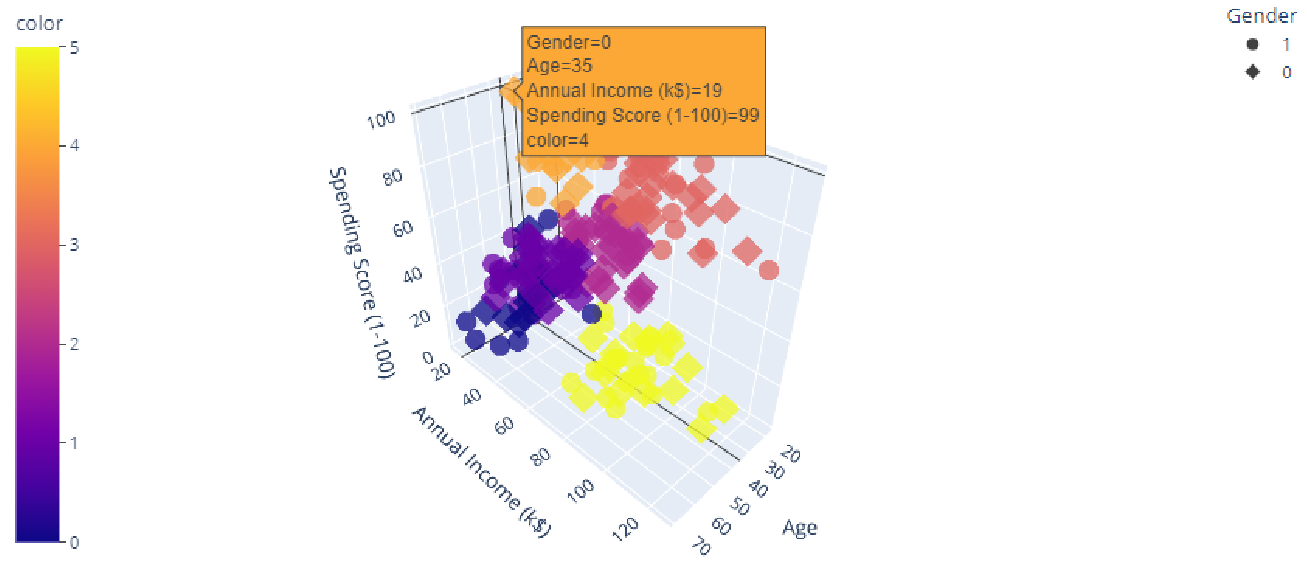
Mini-Batch K-Means Clustering Algorithm with optimal parameters, which is faster than K-Means due to utilizing random fixed size data batches, was employed.

MODEL 2: MiniBatchKMeans Clustering Algorithm with Optimal Parameter Tuning

```
display_alert_color("MODEL 2: MiniBatchKMeans Clustering Algorithm RESULTS", "purple","warning")

results=pd.DataFrame(columns=['Model','Clusters','Silhouette Score','Davies Bouldin Score'])

model_name = "MiniBatchKMeans "

# Define Parameters
n_digits = len(np.unique(df_encoded)) # Get lenth of dataframe to get maximum cluster range

# Loop by Parameters
for i in range(2,n_digits):
    MiniBatchKMeans_labels = MiniBatchKMeans(n_clusters=i, random_state=123).fit_predict(df_encoded)
    sil = (metrics.silhouette_score(df_encoded,MiniBatchKMeans_labels, metric='euclidean').round(3))
    dbs = (metrics.davies_bouldin_score(df_encoded,MiniBatchKMeans_labels).round(3))
    results = results.append({'Model': model_name, 'Clusters': i, 'Silhouette Score': sil, 'Davies Bouldin Score': dbs}, ign
    results.sort_values(by=['Silhouette Score'], inplace=True, ascending=False) # Sort by highest 'Silhouette Score'

# Collect Best Parameters
best_cluster = int(results['Clusters'].iloc[0])

model_MiniBatchKMeans = MiniBatchKMeans(n_clusters=best_cluster)
model_MiniBatchKMeans.fit(df_encoded) # Fit Model
predicted_labels_MiniBatchKMeans = model_MiniBatchKMeans.fit_predict(df_encoded) # Predict Data

sil = (metrics.silhouette_score(df_encoded,predicted_labels_MiniBatchKMeans).round(3))
dbs = (metrics.davies_bouldin_score(df_encoded,predicted_labels_MiniBatchKMeans).round(3))

# Capture Model Results & Append to Main Result Frame
results_MiniBatchKMeans =pd.DataFrame.from_dict({'Model':model_name,'Clusters':best_cluster,'Silhouette Score': sil,'Davies
Combined_Results = Combined_Results.append(results_MiniBatchKMeans,ignore_index=True) # Append model results to main result

print(colored("Total Possible Clusters: ", 'blue', attrs=['bold'])
      + colored(n_digits, 'red', attrs=['bold'])
      + colored("\n\nCluster Results:\n", 'blue', attrs=['bold'])
      + colored(results.head(4), 'blue', attrs=['bold'])
      + colored("\n\nCombined Model Results:\n", 'green', attrs=['bold'])
      + colored(Combined_Results, 'magenta', attrs=['bold']))

# Plot Figure
import plotly.express as px
fig = px.scatter_3d(df_encoded, x='Age', y='Annual Income (k$)', z='Spending Score (1-100)', color=model_MiniBatchKMeans.fit
fig.update_layout(coloraxis_colorbar=dict(yanchor="top", y=1, x=0, ticks="outside"))
# fig.update_layout is being used to make the 'color' legend shift towards left side, otherwise the 'color' and 'Gender' leg
# 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:

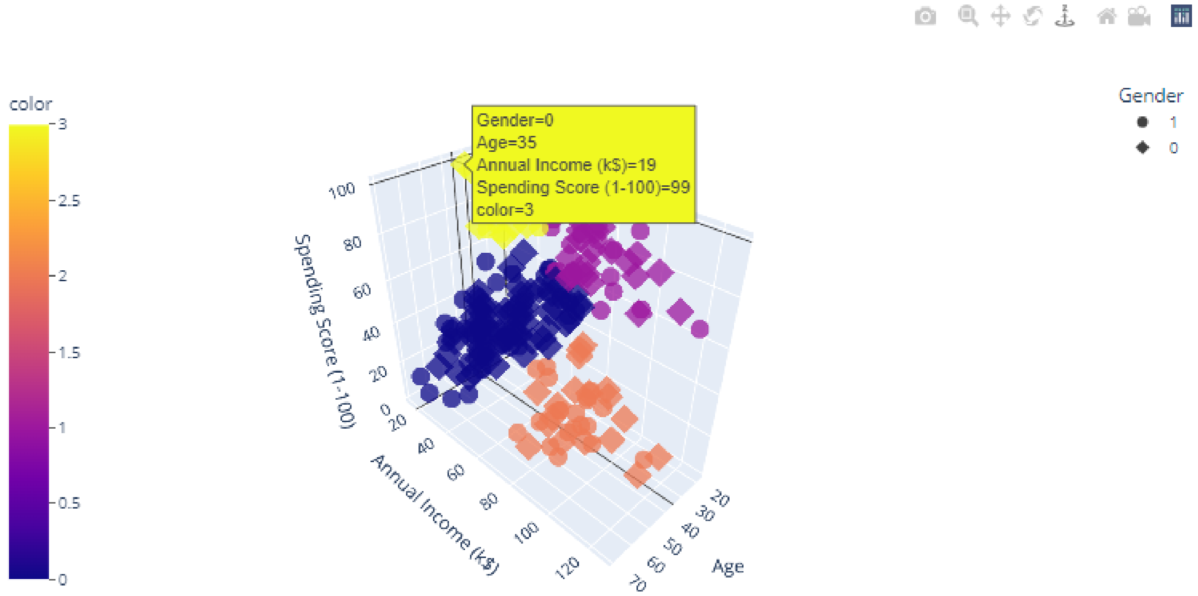
Hierarchical Agglomerative Clustering Algorithm with optimal parameters, which starts with smaller clusters to merge them into bigger ones, was employed.

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

```
display_alert_color("MODEL 3:  Hierarchical/Agglomerative Clustering Algorithm RESULTS", "purple","warning")

results=pd.DataFrame(columns=['Model','Clusters','Silhouette Score','Davies Bouldin Score'])

model_name = "Hierarchical"

# Define Parameters
n_digits = len(np.unique(df_encoded)) # Get lenth of dataframe to get maximum cluster range

linkage = ['ward', 'complete', 'average', 'single']

# Loop by Parameters
for i in range(2,n_digits):
    for l in linkage:
        predicted_labels = AgglomerativeClustering(n_clusters=i,linkage=l)
        clusters = predicted_labels.fit_predict(scaled_df)
        sil = (metrics.silhouette_score(df_encoded,clusters, metric='euclidean').round(3)) # If linkage is “ward”, only “euc
        dbs = (metrics.davies_bouldin_score(df_encoded,clusters).round(3))
        results = results.append({'Model': model_name, 'Clusters': i, 'Linkage':l, 'Silhouette Score': sil, 'Davies Bouldin
        results.sort_values(by=['Silhouette Score'], inplace=True, ascending=False)  # Sort by highest 'Silhouette Score'

results = results[results['Clusters'] != 0.0]

# Collect Best Parameters
best_cluster = int(results['Clusters'].iloc[0])
best_linkage = results['Linkage'].iloc[0]

model_AgglomerativeClustering = AgglomerativeClustering(n_clusters=best_cluster, linkage=best_linkage, affinity='euclidean')
model_AgglomerativeClustering.fit_predict(df_encoded) # Fit Model
predicted_labels_Hierarchical = model_AgglomerativeClustering.fit_predict(df_encoded) # Predict Data

sil = (metrics.silhouette_score(scaled_df,predicted_labels_Hierarchical).round(3))
dbs = (metrics.davies_bouldin_score(scaled_df,predicted_labels_Hierarchical).round(3))

# Capture Model Results & Append to Main Result Frame
results_Hierarchical =pd.DataFrame.from_dict({'Model':model_name,'Clusters':best_cluster,'Silhouette Score': sil,'Davies Bou
Combined_Results = Combined_Results.append(results_Hierarchical,ignore_index=True).replace("NaN","") # Append model results

print(colored("Total Possible Clusters: ", 'blue', attrs=['bold'])
      + colored(n_digits, 'red', attrs=['bold'])
      + colored("\n\nCluster Results:\n", 'blue', attrs=['bold'])
      + colored(results.head(4), 'blue', attrs=['bold'])
      + colored("\n\nCombined Model Results:\n", 'green', attrs=['bold'])
      + colored(Combined_Results, 'magenta', attrs=['bold']))

# Plot Figure
import plotly.express as px
fig = px.scatter_3d(df_encoded, x='Age', y='Annual Income (k$)', z='Spending Score (1-100)', color=model_AgglomerativeCluste
fig.update_layout(coloraxis_colorbar=dict(yanchor="top", y=1, x=0, ticks="outside"))
# 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
```



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

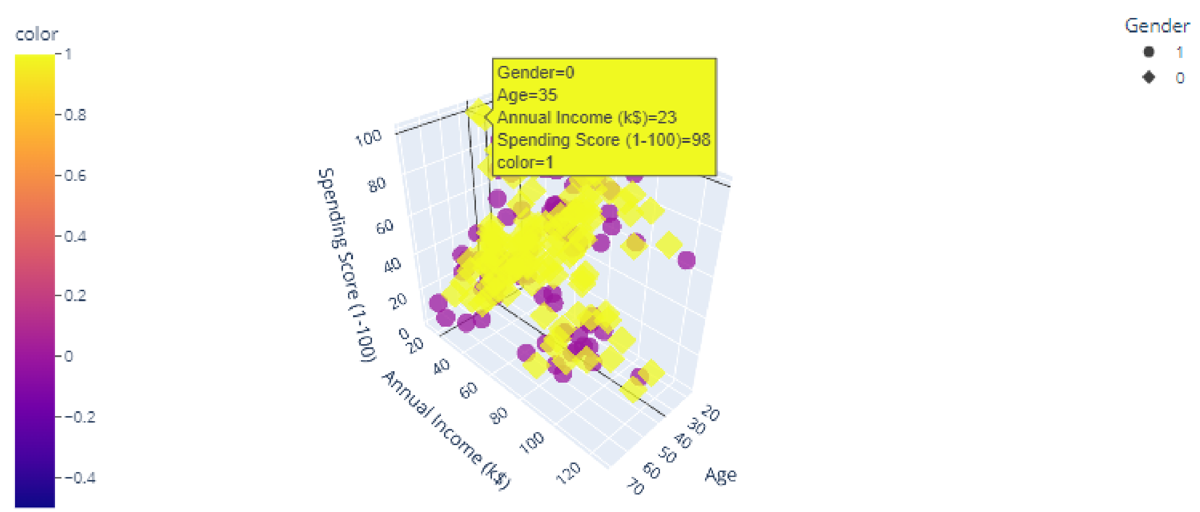
DBSCAN Algorithm with optimal parameters, which groups closely packed points together, was employed.

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

```
display_alert_color("MODEL 4: DBSCAN Algorithm RESULTS", "purple","warning")

results=pd.DataFrame(columns=['Model','Clusters','Silhouette Score','Davies Bouldin Score','Eps','Min_Samples'])

model_name = "DBSCAN"

n_digits = int(len(np.unique(df_encoded)) * 0.50) # Get half the length of dataframe to get fair cluster range
algorithm = ['auto','ball_tree','kd_tree','brute']

# Loop by Parameters
for i in range(1,n_digits):
    for a in algorithm:
        clusters = DBSCAN(eps=i*0.5, min_samples=i, algorithm=a).fit_predict(scaled_df)
    if len(np.unique(clusters))>=2:
        results=results.append({'Model':model_name, 'Algorithm':a, 'Eps':i*0.5,'Min_Samples':i,'Clusters':len(np.unique(clus
        results.sort_values(by=['Silhouette Score'], inplace=True, ascending=False)  # Sort by highest 'Silhouette Score'

# Find Best Parameters
best_eps = results['Eps'].iloc[0]
best_min_samples = int(results['Min_Samples'].iloc[0])
best_clusters = results['Clusters'].iloc[0]
best_algorithm = results['Algorithm'].iloc[0]

# Make Model with Best Parameters
model_DBSCAN = DBSCAN(algorithm=best_algorithm, eps=best_eps, min_samples=best_min_samples)
predicted_labels_DBSCAN = model_DBSCAN.fit_predict(scaled_df)
sil = (metrics.silhouette_score(scaled_df,predicted_labels_DBSCAN).round(3))
dbs = (metrics.davies_bouldin_score(scaled_df,predicted_labels_DBSCAN).round(3))

# Capture Model Results & Append to Main Result Frame
results_DBSCAN =pd.DataFrame.from_dict({'Model':model_name, 'Clusters':best_clusters,'Silhouette Score': sil,'Davies Bouldin
Combined_Results = Combined_Results.append(results_DBSCAN,ignore_index=True) # Append model results to main result dataframe

print(colored("Total Possible Clusters: ", 'blue', attrs=['bold'])
        + colored(n_digits, 'red', attrs=['bold'])
        + colored("\n\nCluster Results:\n", 'blue', attrs=['bold'])
        + colored(results.head(4), 'blue', attrs=['bold'])
        + colored("\n\nCombined Model Results:\n", 'green', attrs=['bold'])
        + colored(Combined_Results, 'magenta', attrs=['bold']))

# Plot Figure (use import plotly.express as px)
fig = px.scatter_3d(df_encoded, x='Age', y='Annual Income (k$)', z='Spending Score (1-100)', color=predicted_labels_DBSCAN,
fig.update_layout(coloraxis_colorbar=dict(yanchor="top", y=1, x=0, ticks="outside"))
# fig.update_layout is being used to make the 'color' legend shift towards left side, otherwise the 'color' and 'Gender' leg
# 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:

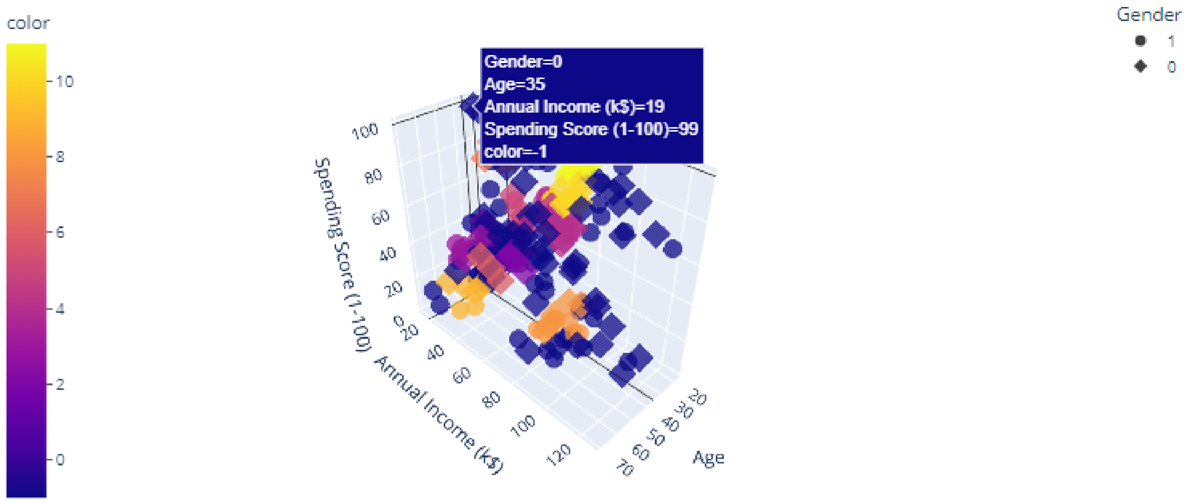
Ordering Points to Identify the Clustering Structure (OPTICS) Algorithm with optimal parameters, which orders data points so that spatially closest points become neighbours in the ordering, was employed.

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

```
display_alert_color("MODEL 5: Ordering Points To Identify the Clustering Structure (OPTICS) Algorithm RESULTS", "purple","wa
results = pd.DataFrame(columns=['Covariance Type','Clusters','Silhouette Score','Davies Bouldin Score'])

model_name = "OPTICS"
model = 'Model 5'

#Define Parameters
algorithm = ['auto','ball_tree','kd_tree','brute']

n_digits = int(len(np.unique(df_encoded))* 0.10) # Get half the length of dataframe to get fair cluster range # scaled_df
n_clusters= np.arange(2,n_digits)
print(n_digits)

# Loop by Parameters
for a in algorithm:
    for j in n_clusters:
        OPTICS_cluster=OPTICS(min_cluster_size=j, algorithm=a)
        clusters= OPTICS_cluster.fit_predict(df_encoded)

if len(np.unique(clusters))>=2:
    results=results.append({'Model#': model, 'Model': model_name,"Algorithm": a, "Clusters":j,"Silhouette Score":metrics.sil
    results.sort_values(by=['Silhouette Score'], inplace=True, ascending=False) # Sort by highest 'Silhouette Score'

# Find Best Parameters
best_algorithm = results['Algorithm'].iloc[0]
best_clusters = int(results['Clusters'].iloc[0]) #j

# Make Model with Best Parameters
model_OPTICS = OPTICS(p=best_clusters, algorithm=best_algorithm)#.fit_predict(df_encoded)
predicted_labels_OPTICS = model_OPTICS.fit_predict(df_encoded)
pred5 = predicted_labels_OPTICS.copy()
sil = (metrics.silhouette_score(scaled_df,predicted_labels_OPTICS).round(3))
dbs = (metrics.davies_bouldin_score(scaled_df,predicted_labels_OPTICS).round(3))

# Capture Model Results & Append to Main Result Frame
results_OPTICS = pd.DataFrame.from_dict({'Model#': model,'Model': model_name,
                                  'Clusters':best_clusters,
                                  'Silhouette Score': sil,
                                  'Davies Bouldin Score': dbs}, orient='index').T # Make Dataframe & Transpose
Combined_Results = Combined_Results.append(results_OPTICS,ignore_index=True) # Append model results to main result dataframe

print(colored("Total Possible Clusters: ", 'blue', attrs=['bold'])
      + colored(n_digits, 'red', attrs=['bold'])
      + colored("\n\nCluster Results:\n", 'blue', attrs=['bold'])
      + colored(results.head(4), 'blue', attrs=['bold'])
      + colored("\n\nCombined Model Results:\n", 'green', attrs=['bold'])
      + colored(Combined_Results, 'magenta', attrs=['bold']))

# Plot Figure (use import plotly.express as px)
fig = px.scatter_3d(df_encoded, x='Age', y='Annual Income (k$)', z='Spending Score (1-100)', color=predicted_labels_OPTICS,
fig.update_layout(coloraxis_colorbar=dict(yanchor="top", y=1, x=0, ticks="outside"))
# fig.update_layout is being used to make the 'color' legend shift towards left side, otherwise the 'color' and 'Gender' leg
# 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

```
display_alert_color("6) SUMMARY KEY FINDINGS & INSIGHTS", "darkblue","info")
display_alert_color("Extract Cluster Information", "darkgreen","warning")

model = Combined_Results['Model#'].iloc[1] # Get Model Number

# Get Target Variable Predictions
if model == 'Model 1':#
    prediction  = pred1.copy()
elif model == 'Model 2':
    prediction  = pred2.copy()
elif model == 'Model 3':
    prediction  = pred3.copy()
elif model == 'Model 4':
    prediction  = pred4.copy()
elif model == 'Model 5':#
    prediction  = pred5.copy()
else:
    print("")
no_clusters = (np.unique(prediction) )

print(colored("Unique Cluster List: ", 'blue', attrs=['bold'])
      + colored(list(no_clusters), 'red', attrs=['bold']))

display_alert_color("Interpretation of Cluster Information", "darkgreen","warning")

# Decode encoded fields
df_interpret = df_encoded.copy()
df_interpret['Gender'] = df_interpret['Gender'].map({0: 'Female', 1: 'Male'}) # Decode Encoded Data
df_interpret['labels'] = prediction
#df_interpret = df_interpret.sort_values(["labels","Annual Income (k$)", "Spending Score (1-100)"],ascending = [True, True,

df_interpret['label description'] = df_interpret['labels'].map({
    0: 'low income - medium spending', # blue cluster
    1: 'upper high income - high spending', # purple cluster
    2: 'medium income - medium spending', # pink cluster
    3: 'high income - low spending', # orange cluster
    4: 'high income - high spending', # yellow cluster
    5: 'low income - high spending'}) # yellow cluster

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

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