JOY KARMOKAR

ID: 18-39263-3, Sec: E

Assessment-Final Term
Data Warehousing and Data Mining

Q.1: **Apriori Algorithm**

When we go to a store, would we not want the aisles to be ordered in such a manner that reduces our efforts to buy things. For example, I would want the toothbrush, the paste, the mouth wash and other dental products on a single aisle because when I buy, I tend to buy them together. This is done by a way in which we find associations between items.

In order to understand the concept better, let's take a simple dataset let's name it as Coffee dataset consisting of a few hypothetical transactions. The Coffee dataset consisting of items purchased from a retail store.

Coffee dataset:

Transaction	Item 1	Item 2	Item 3
1	Milk	Sugar	Coffee Powder
2	Milk	Sugar	Coffee Powder
3	Milk	Sugar	Coffee Powder
4	Milk	Sugar	----------
5	Milk	Sugar	----------

The Association Rules:

For this dataset, we can write the following association rules: (Rules are just for illustrations and understanding of the concept. They might not represent the actuals).

Rule 1: If Milk is purchased, then Sugar is also purchased.

Rule 2: If Sugar is purchased, then Milk is also purchased.

Rule 3: If Milk and Sugar are purchased, Then Coffee powder is also purchased in 60% of the transactions.

Generally, association rules are written in "IF-THEN" format. I can also use the term "Antecedent" for IF (LHS) and "Consequent" for THEN (RHS).

From the above rules, we understand the following explicitly:

- Whenever Milk is purchased, Sugar is also purchased or vice versa.
- If Milk and Sugar are purchased then the coffee powder is also purchased. This is true in 3 out of the 5 transactions.

For example, if we see {Milk} as a set with one item and {Coffee} as another set with one item, we will use these to find sets with two items in the dataset such as {Milk, Coffee} and then later see which products are purchased with both of these in our basket.

Therefore, now we will search for a suitable right-hand side or Consequent. If someone buys Coffee with Milk, we will represent it as {Coffee} => {Milk} where Coffee becomes the LHS and Milk the RHS.

When we use these to explore more k-item sets, we might find that {Coffee, Milk} => {Tea}. That means the people who buy Coffee and Milk have a possibility of buying Tea as well.

Let us see how the item sets are actually built using the Apriori.

LHS	RHS	Count
Milk	-------	300
Coffee	-------	200
Tea	-------	200
Sugar	-------	150
Milk	Coffee	100
Tea	Sugar	80
Milk, Coffee	Tea	40
Milk, Coffee, Tea	Sugar	10

Apriori envisions an iterative approach where it uses k-Item sets to search for (k+1)-Item sets. The first 1-Item sets are found by gathering the count of each item in the set. Then the 1-Item sets are used to find 2-Item sets and so on until no more k-Item sets can be explored; when all our items land up in one final observation as visible in our last row of the table above**.** One exploration takes one scan of the complete dataset. An Item set is a mathematical set of products in the basket.

Q.2: **Linear Regression**

I will use data on house sales in King County to predict house prices using simple (one input) linear regression. This is the part of University of Washington Machine learning specialization. I will perform below things:

- Use Python 3, Use NumPy, Pandas, Scikit-learn to compute important summary statistics.
- Implement function to compute the Linear Regression weights using the closed form solution.
- Implement function to make predictions of the output given the input feature.
- Turn the regression around to predict the input given the output.
- Compare two different models for predicting house prices.

At first load libraries.

In [124]:
```
import numpy as np
import pandas as pd
from sklearn import cross_validation
```

Load house sales data and dataset is from house sales in King County, the region where the city of Seattle, WA is located.

In [125]:
```
sales = pd.read_csv('kc_house_data.csv',dtype = {'bathrooms':float, 'waterfront':int,

                    'sqft_above':int, 'sqft_living15':float, 'grade':int, 'yr_renovated':int,

                    'price':float, 'bedrooms':float, 'zipcode':str, 'long':float, 'sqft_lot15':float,

                    'sqft_living':float, 'floors':str, 'condition':int, 'lat':float, 'date':str,

                    'sqft_basement':int, 'yr_built':int, 'id':str, 'sqft_lot':int, 'view':int})
```

Split data into training and testing.

In [139]:
```
train_data,test_data =cross_validation.train_test_split(sales,test_size=0.2, random_state=0)
```

Build a generic simple linear regression function.

In [128]:
```
def simple_linear_regression(input_feature, output):

    df = pd.DataFrame({'X':input_feature.as_matrix(),'Y':output.as_matrix()})
    correlation = df.corr(method='pearson').iloc[1,0]

    # use the formula for the slope
    slope = correlation * np.std(output) / np.std(input_feature)

    # use the formula for the intercept
    intercept = np.mean(output) - slope * np.mean(input_feature)

    return (intercept, slope)
```

In [129]:
```
sqft_intercept, sqft_slope = simple_linear_regression(train_data['sqft_living'], train_data['price'])
```

Predicting Values now that we have the model parameters: intercept & slope we can make predictions.

In [130]:
```
def get_regression_predictions(input_feature, intercept, slope):
    # calculate the predicted values:
    predicted_values = intercept + slope * input_feature

    return predicted_values
```

Now that I can calculate a prediction given the slope and intercept let's make a prediction. Use or alter the following to find out the estimated price for a house with 2650 square feet according to the square feet model I estimated above.

In [131]:
```
my_house_sqft = 2650
estimated_price = get_regression_predictions(my_house_sqft, sqft_intercept, sqft_slope)
print ("The estimated price for a house with {} squarefeet is {}" .format(my_house_sqft, estimated_price))
```

```
The estimated price for a house with 2650 squarefeet is 704259.6128700661
```

Residual Sum of Squares now that I have a model and can make predictions let's evaluate our model using Residual Sum of Squares (RSS).

In [132]:
```
def get_residual_sum_of_squares(input_feature, output, intercept, slope):
    # First get the predictions
    predicted_value = intercept + slope * input_feature
    # then compute the residuals (since we are squaring it doesn't matter which order you subtract)
    RSS = (predicted_value - output) ** 2
    RSS = RSS.sum()
    return(RSS)
```

In [133]:
```
rss_prices_on_sqft = get_residual_sum_of_squares(test_data['sqft_living'], test_data['price'], sqft_intercept)
print('The RSS of predicting Prices based on Square Feet is : {}'.format(str(rss_prices_on_sqft)))
```

```
The RSS of predicting Prices based on Square Feet is : 267770022739753.47
```

Predict the squarefeet given price. I will predict the square foot given the price. Since I have an equation y = a + b*x we can solve the function for x.

In [134]:
```
def inverse_regression_predictions(output, intercept, slope):
    #Use this equation to compute the inverse predictions:
    estimated_feature = (output - intercept)/slope
    return estimated_feature
```

Now that I have a function to compute the square feet given the price from our simple regression model let's see how big I might expect a house that costs $800000 to be.

In [135]:
```
my_house_price = 800000
estimated_squarefeet = inverse_regression_predictions(my_house_price, sqft_intercept, sqft_slope)
print("The estimated squarefeet for a house worth {} is {}".format(my_house_price, estimated_squarefeet))
```

```
The estimated squarefeet for a house worth 800000 is 2987.151366648074
```

New model: estimate prices from bedrooms. I have made one model for predicting house prices using square feet, but there are many other features. Using simple linear regression function to estimate the regression parameters from predicting Prices based on number of bedrooms

In [136]:
```
# Estimate the slope and intercept for predicting 'price' based on 'bedrooms'
bedrooms_intercept, bedrooms_slope = simple_linear_regression(train_data['bedrooms'], train_data['price'])
```

Test our Linear Regression Algorithm. Now I have two models for predicting the price of a house. Calculate the RSS on the TEST data. Compute the RSS from predicting prices using bedrooms and from predicting prices using square feet.

In [137]:
```
# Compute RSS when using bedrooms on TEST data:
rss_prices_on_bedrooms = get_residual_sum_of_squares(test_data['bedrooms'], test_data['price'],
bedrooms_intercept, bedrooms_slope)
print('The RSS of predicting Prices based on Bedrooms Feet is : {}'.format(str(rss_prices_on_bedrooms)))
```

```
The RSS of predicting Prices based on Bedrooms Feet is : 472745600358876.1
```

In [138]:
```
# Compute RSS when using squarefeet on TEST data:
rss_prices_on_sqft = get_residual_sum_of_squares(test_data['sqft_living'], test_data['price'],
sqft_intercept, sqft_slope)
print('The RSS of predicting Prices based on Square Feet is : {}'.format(str(rss_prices_on_sqft)))
```

```
The RSS of predicting Prices based on Square Feet is : 267770022739753.47
```

Q.3:

This report shows how to perform a mall customers segmentation using Machine Learning algorithms. Three techniques will be presented and compared: **K-Means, Affinity Propagation and DBSCAN.**

Reading data: In this section, raw data will be read, overviewed and checked if cleaning is required.

In [1]:

```
# importing basic libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats

print("pandas version: {}".format(pd.__version__))
print("numpy version: {}".format(np.__version__))
print("seaborn version: {}".format(sns.__version__))
```

```
pandas version: 0.25.3
numpy version: 1.18.1
seaborn version: 0.9.0
```

In [2]:

```
mall_data = pd.read_csv('../input/customer-segmentation-tutorial-in-python/Mall_Customers.csv')

print('There are {} rows and {} columns in our dataset'.format(mall_data.shape[0],mall_data.shape[1]))
```

```
There are 200 rows and 5 columns in our dataset
```

In [3]:

```
mall_data.head()
```

Out[3]:

	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

In [4]:

```
mall_data.info()
```

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

There are 5 columns:

- Customer ID - numerical - unique customer number, integer
- Gender - categorical - binary (Male/Female)
- Age - numerical - integer
- Annual Income (k$) - numerical - integer
- Spending Score (1-100) - numerical - integer

There is one binary, categorical column: gender. You may be tempted to one-hot encode it for the clustering. It is:

- technically possible
- theoretically not forbidden
- practically not recommended

In [5]:
```
mall_data.describe()
```

Out[5]:

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

In [6]:
```
mall_data.isnull().sum()
```

Out[6]:
```
CustomerID              0
Gender                  0
Age                     0
Annual Income (k$)      0
Spending Score (1-100)  0
dtype: int64
```

There are no missing data. This simplifies the analysis but it is a very unlikely scenario in a real-life where analysts spend a significant amount of time cleaning their data before the core analysis is performed

Customer's segmentation:

The very first step in a clustering analysis is importing **K-Means**

In [20]:
```
from sklearn.cluster import KMeans
```

For clustering only numeric columns are used.

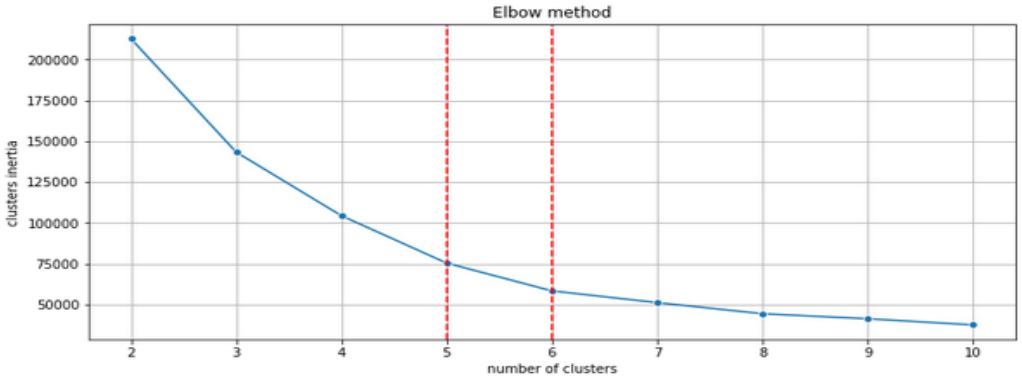
In [21]:
```
X_numerics = mall_data[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']] # subset with nu
meric variables only
```

In order to find an appropriate number of clusters, the elbow method will be used. In this method for this case, the inertia for a number of clusters between 2 and 10 will be calculated. The rule is to choose the number of clusters where you see a kink or "an elbow" in the graph.

In [22]:
```
from sklearn.metrics import silhouette_score
n_clusters = [2,3,4,5,6,7,8,9,10] # number of clusters
clusters_inertia = [] # inertia of clusters
s_scores = [] # silhouette scores

for n in n_clusters:
    KM_est = KMeans(n_clusters=n, init='k-means++').fit(X_numerics)
    clusters_inertia.append(KM_est.inertia_)    # data for the elbow method
    silhouette_avg = silhouette_score(X_numerics, KM_est.labels_)
    s_scores.append(silhouette_avg) # data for the silhouette score method
```

The graph below shows the inertia for selected range of clusters.

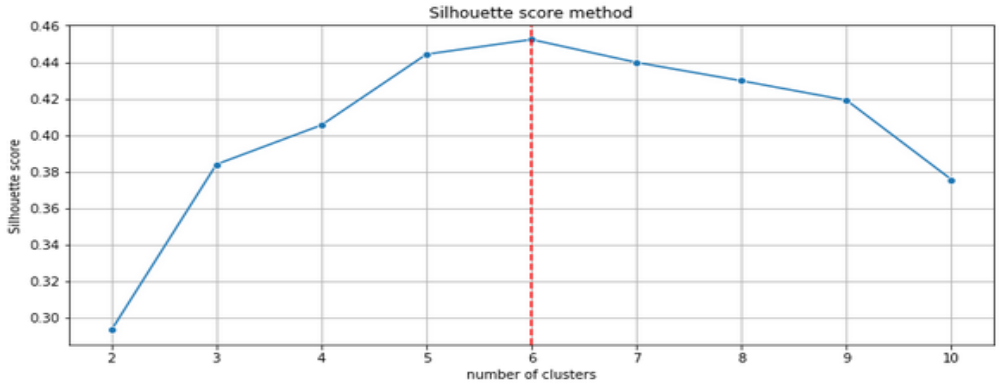
In [23]:
```
fig, ax = plt.subplots(figsize=(12,5))
ax = sns.lineplot(n_clusters, clusters_inertia, marker='o', ax=ax)
ax.set_title("Elbow method")
ax.set_xlabel("number of clusters")
ax.set_ylabel("clusters inertia")
ax.axvline(5, ls="--", c="red")
ax.axvline(6, ls="--", c="red")
plt.grid()
plt.show()
```



There is no clear "elbow" visible. A choice of 5 or 6 clusters seems to be fair. Let's see the silhouette score.

In [24]:

```
fig, ax = plt.subplots(figsize=(12,5))
ax = sns.lineplot(n_clusters, s_scores, marker='o', ax=ax)
ax.set_title("Silhouette score method")
ax.set_xlabel("number of clusters")
ax.set_ylabel("Silhouette score")
ax.axvline(6, ls="--", c="red")
plt.grid()
plt.show()
```



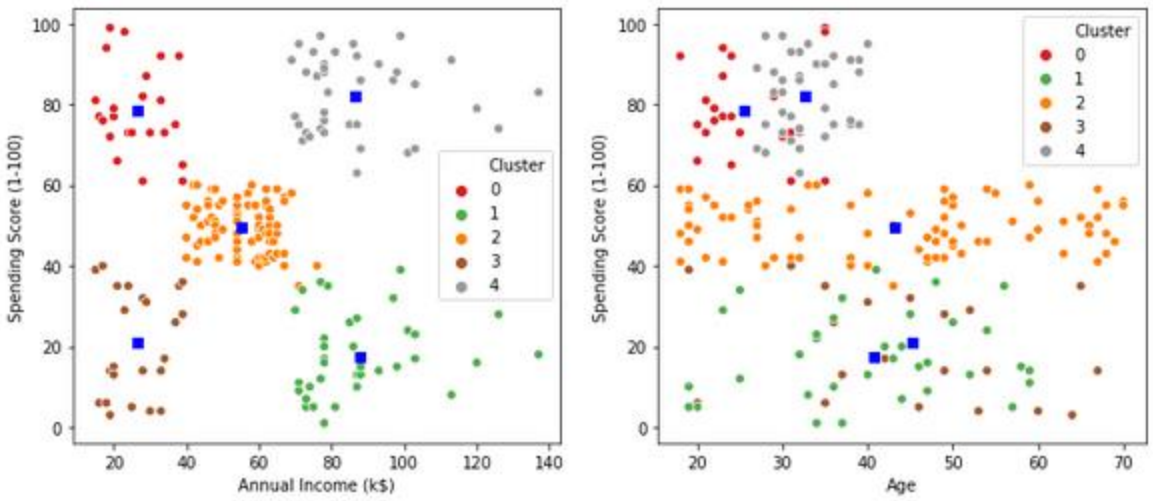
Silhouette score method indicates the best options would be respectively 6 or 5 clusters. Let's compare both.

In [25]:

```
KM_5_clusters = KMeans(n_clusters=5, init='k-means++').fit(X_numerics) # initialise and fit K-Me
ans model

KM5_clustered = X_numerics.copy()
KM5_clustered.loc[:,'Cluster'] = KM_5_clusters.labels_ # append labels to points
```

In [26]:

```
fig1, (axes) = plt.subplots(1,2,figsize=(12,5))


scat_1 = sns.scatterplot('Annual Income (k$)', 'Spending Score (1-100)', data=KM5_clustered,
                hue='Cluster', ax=axes[0], palette='Set1', legend='full')

sns.scatterplot('Age', 'Spending Score (1-100)', data=KM5_clustered,
                hue='Cluster', palette='Set1', ax=axes[1], legend='full')

axes[0].scatter(KM_5_clusters.cluster_centers_[:,1],KM_5_clusters.cluster_centers_[:,2], marker
='s', s=40, c="blue")
axes[1].scatter(KM_5_clusters.cluster_centers_[:,0],KM_5_clusters.cluster_centers_[:,2], marker
='s', s=40, c="blue")
plt.show()
```



K-Means algorithm generated the following 5 clusters:

- clients with **low** annual income and **high** spending score
- clients with **medium** annual income and **medium** spending score
- clients with **high** annual income and **low** spending score
- clients with **high** annual income and **high** spending score
- clients with **low** annual income and **low** spending score

There are no distinct groups is terms of customer's age.

In [27]:
```
KM_clust_sizes = KM5_clustered.groupby('Cluster').size().to_frame()
KM_clust_sizes.columns = ["KM_size"]
KM_clust_sizes
```

Out[27]:

	KM_size
Cluster	
0	23
1	36
2	79
3	23
4	39

The biggest cluster is a cluster number 1 with 79 observations ("medium-medium" clients). There are two the smallest ones each containing 23 observations (cluster 3 "high-high" and cluster 0 "low-high" clients).

Customer's segmentation:

The first step - importing **Affinity Propagation**

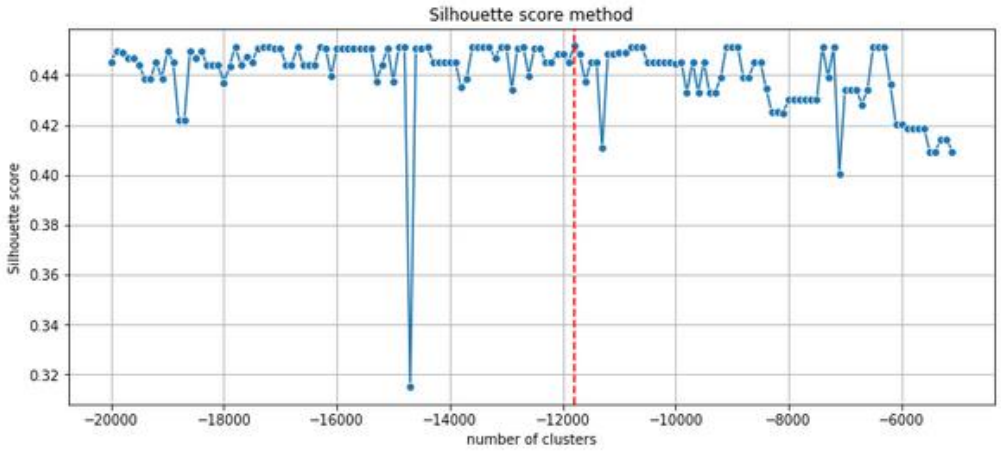
In [42]:
```
from sklearn.cluster import AffinityPropagation
```

In [43]:
```
no_of_clusters = []
preferences = range(-20000,-5000,100) # arbitraty chosen range
af_sil_score = [] # silouette scores

for p in preferences:
    AF = AffinityPropagation(preference=p, max_iter=200).fit(X_numerics)
    no_of_clusters.append((len(np.unique(AF.labels_))))
    af_sil_score.append(silhouette_score(X_numerics, AF.labels_))

af_results = pd.DataFrame([preferences, no_of_clusters, af_sil_score], index=['preference','clu
sters', 'sil_score']).T
af_results.sort_values(by='sil_score', ascending=False).head() # display only 5 best scores
```

Out[43]:

	preference	clusters	sil_score
82	-11800.0	6.0	0.451649
27	-17300.0	6.0	0.451491
51	-14900.0	6.0	0.451491
52	-14800.0	6.0	0.451440
28	-17200.0	6.0	0.451440

In [44]:
```
fig, ax = plt.subplots(figsize=(12,5))
ax = sns.lineplot(preferences, af_sil_score, marker='o', ax=ax)
ax.set_title("Silhouette score method")
ax.set_xlabel("number of clusters")
ax.set_ylabel("Silhouette score")
ax.axvline(-11800, ls="--", c="red")
plt.grid()
plt.show()
```



In [45]:
```
AF = AffinityPropagation(preference=-11800).fit(X_numerics)
```

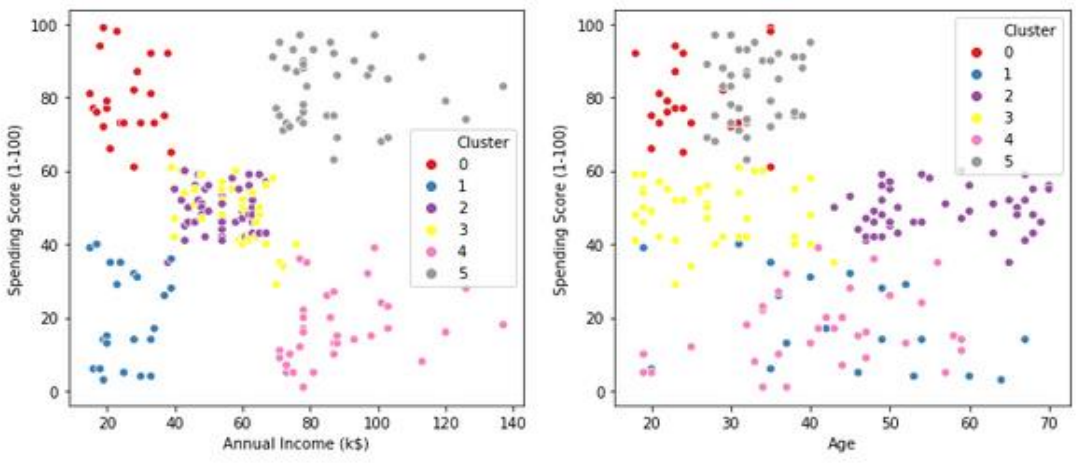
In [46]:
```
AF_clustered = X_numerics.copy()
AF_clustered.loc[:,'Cluster'] = AF.labels_ # append labels to points
```

In [47]:
```
AF_clust_sizes = AF_clustered.groupby('Cluster').size().to_frame()
AF_clust_sizes.columns = ["AF_size"]
AF_clust_sizes
```

Out[47]:

Cluster	AF_size
0	22
1	22
2	44
3	39
4	34
5	39

In [48]:
```
fig3, (ax_af) = plt.subplots(1,2,figsize=(12,5))


scat_1 = sns.scatterplot('Annual Income (k$)', 'Spending Score (1-100)', data=AF_clustered,
                hue='Cluster', ax=ax_af[0], palette='Set1', legend='full')

sns.scatterplot('Age', 'Spending Score (1-100)', data=AF_clustered,
                hue='Cluster', palette='Set1', ax=ax_af[1], legend='full')

plt.setp(ax_af[0].get_legend().get_texts(), fontsize='10')
plt.setp(ax_af[1].get_legend().get_texts(), fontsize='10')
plt.show()
```



Clusters generated by the Affinity Propagation algorithm created relatively even-sized clusters similar to ones created by K-Means.

Customer's segmentation:

The first step - importing **DBSCAN**

In [34]:
```
from sklearn.cluster import DBSCAN
```

To choose the best combination of the algorithm parameters I will first create a matrix of investigated combinations.

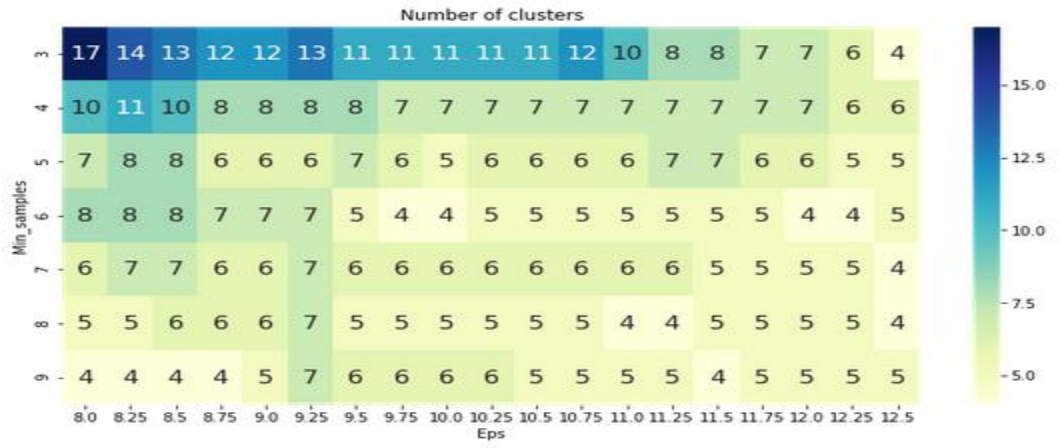
In [35]:
```
from itertools import product

eps_values = np.arange(8,12.75,0.25) # eps values to be investigated
min_samples = np.arange(3,10) # min_samples values to be investigated
DBSCAN_params = list(product(eps_values, min_samples))
```

Collecting number of generated clusters.

In [36]:
```
no_of_clusters = []
sil_score = []

for p in DBSCAN_params:
    DBS_clustering = DBSCAN(eps=p[0], min_samples=p[1]).fit(X_numerics)
    no_of_clusters.append(len(np.unique(DBS_clustering.labels_)))
    sil_score.append(silhouette_score(X_numerics, DBS_clustering.labels_))
```

A heat plot below shows how many clusters were generated by the algorithm for the respective parameter's combinations.

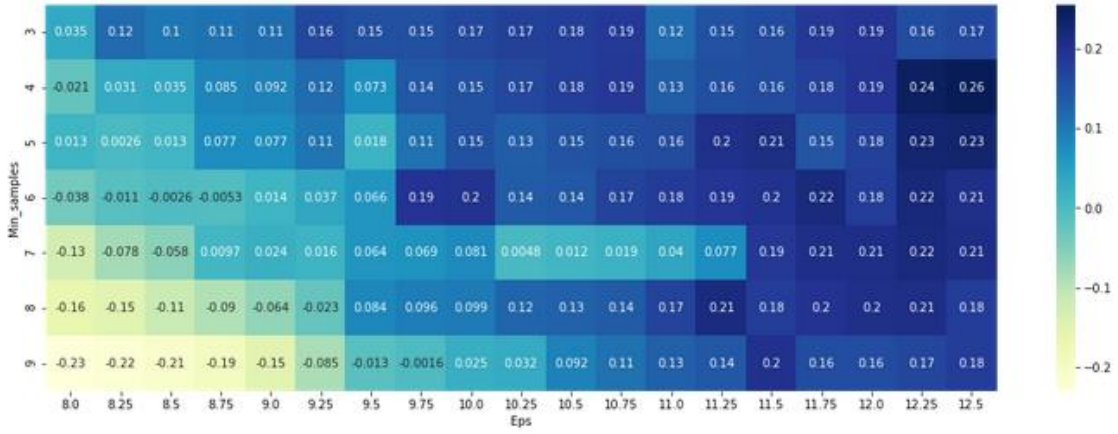
In [37]:
```
tmp = pd.DataFrame.from_records(DBSCAN_params, columns =['Eps', 'Min_samples'])
tmp['No_of_clusters'] = no_of_clusters

pivot_1 = pd.pivot_table(tmp, values='No_of_clusters', index='Min_samples', columns='Eps')

fig, ax = plt.subplots(figsize=(12,6))
sns.heatmap(pivot_1, annot=True,annot_kws={"size": 16}, cmap="YlGnBu", ax=ax)
ax.set_title('Number of clusters')
plt.show()
```



As the heat plot above shows, the number of clusters vary from 17 to 4.

In [38]:

```
tmp = pd.DataFrame.from_records(DBSCAN_params, columns =['Eps', 'Min_samples'])
tmp['Sil_score'] = sil_score

pivot_1 = pd.pivot_table(tmp, values='Sil_score', index='Min_samples', columns='Eps')

fig, ax = plt.subplots(figsize=(18,6))
sns.heatmap(pivot_1, annot=True, annot_kws={"size": 10}, cmap="YlGnBu", ax=ax)
plt.show()
```



Global maximum is 0.26 for eps=12.5 and min_samples=4.

In [39]:

```
DBS_clustering = DBSCAN(eps=12.5, min_samples=4).fit(X_numerics)

DBSCAN_clustered = X_numerics.copy()
DBSCAN_clustered.loc[:,'Cluster'] = DBS_clustering.labels_ # append labels to points
```

Clusters sizes.

In [40]:

```
DBSCAN_clust_sizes = DBSCAN_clustered.groupby('Cluster').size().to_frame()
DBSCAN_clust_sizes.columns = ["DBSCAN_size"]
DBSCAN_clust_sizes
```

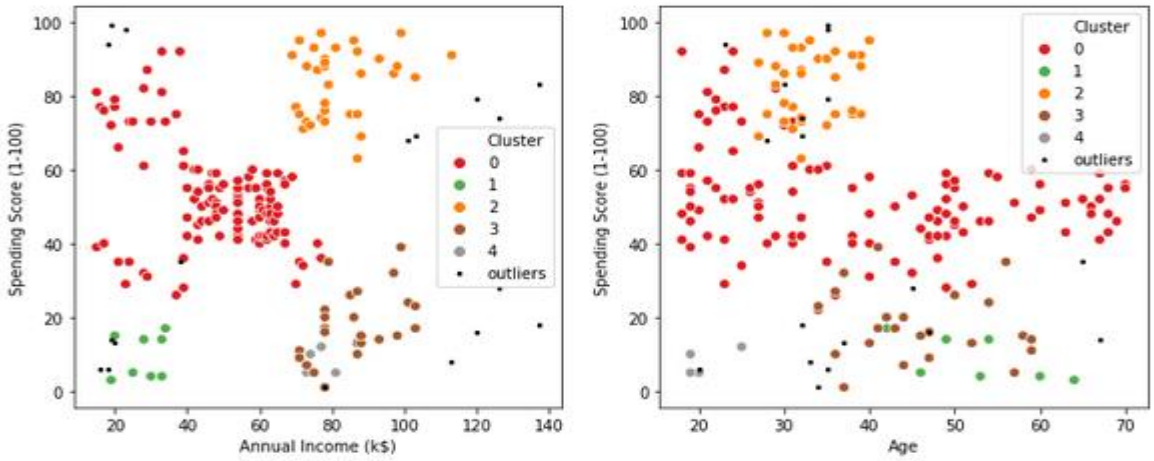
Out[40]:

Cluster	DBSCAN_size
-1	18
0	112
1	8
2	34
3	24
4	4

DBSCAN created 5 clusters plus outlier's cluster (-1). Sizes of clusters 0-4 vary significantly - some have only 4 or 8 observations. There are 18 outliers.

In [41]:

```
outliers = DBSCAN_clustered[DBSCAN_clustered['Cluster']==-1]

fig2, (axes) = plt.subplots(1,2,figsize=(12,5))


sns.scatterplot('Annual Income (k$)', 'Spending Score (1-100)',
                data=DBSCAN_clustered[DBSCAN_clustered['Cluster']!=-1],
                hue='Cluster', ax=axes[0], palette='Set1', legend='full', s=45)

sns.scatterplot('Age', 'Spending Score (1-100)',
                data=DBSCAN_clustered[DBSCAN_clustered['Cluster']!=-1],
                hue='Cluster', palette='Set1', ax=axes[1], legend='full', s=45)

axes[0].scatter(outliers['Annual Income (k$)'], outliers['Spending Score (1-100)'], s=5, label=
'outliers', c="k")
axes[1].scatter(outliers['Age'], outliers['Spending Score (1-100)'], s=5, label='outliers', c="
k")
axes[0].legend()
axes[1].legend()

plt.setp(axes[0].get_legend().get_texts(), fontsize='10')
plt.setp(axes[1].get_legend().get_texts(), fontsize='10')

plt.show()
```



The graph above shows that there are some outliers - these points do not meet distance and minimum samples requirements to be recognized as a cluster.

- The customer segmentation allows to understand the behavior of different customers and accordingly plan our marketing strategy. After we create the clusters, we learned how to understand the profiles of the customers in different segments. That's why I'm choosing the technique and the dataset.

Q.4:

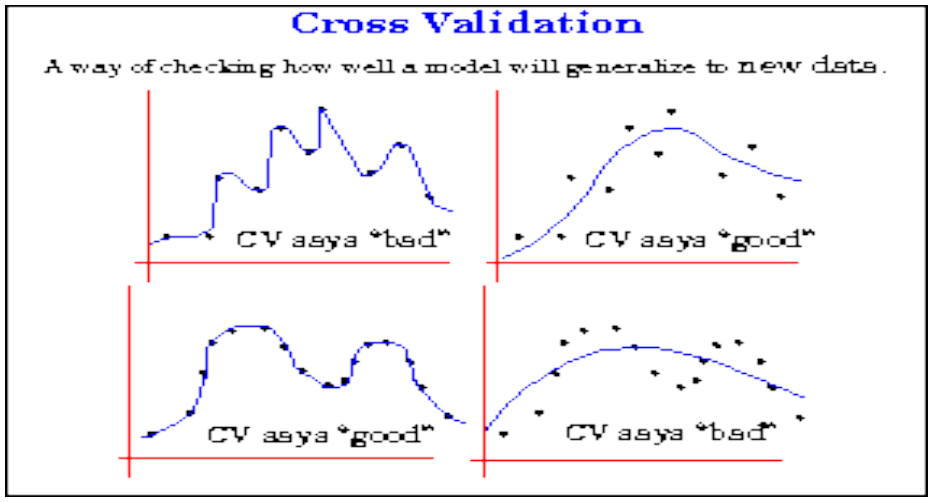
Cross validation:

Cross validation is a model evaluation method that is better than residuals. The problem with residual evaluations is that they do not give an indication of how well the learner will do when it is asked to make new predictions for data it has not already seen. One way to overcome this problem is to not use the entire data set when training a learner. Some of the data is removed before training begins. Then when training is done, the data that was removed can be used to test the performance of the learned model on new data. This is the basic idea for a whole class of model evaluation methods called cross validation.

The holdout method is the simplest kind of cross validation. The data set is separated into two sets, called the training set and the testing set. The function approximator fits a function using the training set only. Then the function approximator is asked to predict the output values for the data in the testing set it has never seen these output values before. The errors it makes are accumulated as before to give the mean absolute test set error, which is used to evaluate the model. The advantage of this method is that it is usually preferable to the residual method and takes no longer to compute. However, its evaluation can have a high variance. The evaluation may depend heavily on which data points end up in the training set and which end up in the test set, and thus the evaluation may be significantly different depending on how the division is made.

K-fold cross validation is one way to improve over the holdout method. The data set is divided into k subsets, and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form a training set. Then the average error across all k trials is computed. The advantage of this method is that it matters less how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set k-1 time. The variance of the resulting estimate is reduced ask is increased. The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation. A variant of this method is to randomly divide the data into a test and training set k different times. The advantage of doing this is that you can independently choose how large each test set is and how many trials you average over.

Leave-one-out cross validation is K-fold cross validation taken to its logical extreme, with K equal to N, the number of data points in the set. That means that N separate times, the function approximator is trained on all the data except for one point and a prediction is made for that point. As before the average error is computed and used to evaluate the model. The evaluation given by leave-one-out cross validation error (LOO-XVE) is good, but at first pass it seems very expensive to compute. Fortunately, locally weighted learners can make LOO predictions just as easily as they make regular predictions. That means computing the LOO-XVE takes no more time than computing the residual error and it is a much better way to evaluate models. We will see shortly that Vizier relies heavily on LOO-XVE to choose its metacodes.



Feature selection:

Feature selection is one of the important concepts of machine learning, which highly impacts the performance of the model. As machine learning works on the concept of "Garbage in Garbage Out", so we always need to input the most appropriate and relevant dataset to the model in order to get a better result. A feature is an attribute that has an impact on a problem or is useful for the problem, and choosing the important features for the model is known as feature selection. Each machine learning process depends on feature engineering, which mainly contains two processes; which are Feature Selection and Feature Extraction. Although feature selection and extraction processes may have the same objective, both are completely different from each other. The main difference between them is that feature selection is about selecting the subset of the original feature set, whereas feature extraction creates new features. Feature selection is a way of reducing the input variable for the model by using only relevant data in order to reduce overfitting in the model. So, we can define feature Selection as, "It is a process of automatically or manually selecting the subset of most appropriate and relevant features to be used in model building." Feature selection is performed by either including the important features or excluding the irrelevant features in the dataset without changing them.

Feature selection is the study of algorithms for reducing dimensionality of data to improve machine learning performance. For a dataset with N features and M dimensions or features, attributes, feature selection aims to reduce M *to* M′ and $M' \leq M$. It is an important and widely used approach to dimensionality reduction. Another effective approach is feature extraction. One of the key distinctions of the two approaches lies at their outcomes. Assuming we have four features F_1, F_2, F_3, F_4, if both approaches result in 2 features, the 2 selected features are a subset of 4 original features say, F_1, F_3, but the 2 extracted features are some combination of 4 original features, $F_1' = \sum a_i F_i$ and $F_2' = \sum b_i F_i$, where a_i, b_i are some constants. Feature selection is commonly used in applications where original features need to be retained.