Data Analytics For Business

Final Mini Project

Understand Business Problem

CRISA Consumer Segmentation

Problem:

- How can CRISA segment the market based on two key sets of variables more directly related to the purchase process and to brand loyalty
  - Two key sets are: Purchase behavior and Basis of purchase

Understand Data

Importing Libraries and Read in CSV File

Mini Project: Segmenting Consumersof Bath Soap

- **Section**: Section-01 (8:00AM)
- **Names**: Leighton Joy, Alex Bibat, Chase Petri, Adam White
- **Due Date**: 05/10/2023
- **Purpose**: test final knowledge on clustering and logistic regression

1. Use k-means clustering to identify clusters of households based on the variables that describe purchase behavior (those variables in the table above).

In [47]:

```
# import the libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import metrics
```

In [48]:

```
# load the dataset
df= pd.read_csv('BathSoapHousehold.csv')
df.head()
```

Out[48]:

	Member id	SEC	FEH	MT	SEX	AGE	EDU	HS	CHILD	CS	...	PropCat 6	PropCat 7	PropCat 8	PropCat 9	PropCat 10	PropCat 11	PropCat 12	PropCat 13	PropCat 14	Pr
0	1010010	4	3	10	1	4	4	2	4	1	...	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.028037	0.0	0.130841	0.3
1	1010020	3	2	10	2	2	4	4	2	1	...	0.347048	0.026834	0.016100	0.014311	0.0	0.059034	0.000000	0.0	0.080501	0.0
2	1014020	2	3	10	2	4	5	6	4	1	...	0.121212	0.033550	0.010823	0.008658	0.0	0.000000	0.016234	0.0	0.561688	0.0
3	1014030	4	0	0	0	4	0	0	5	0	...	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.600000	0.0
4	1014190	4	1	10	2	3	4	4	3	1	...	0.000000	0.000000	0.048193	0.000000	0.0	0.000000	0.000000	0.0	0.144578	0.0

5 rows × 46 columns

Imported the libraries need to run the mini project

We displayed the top values to see what the data was like

Rename Columns and Assign X to Independent Variables

In [50]:
```
# create new dataset with relevant columns
df2 = pd.DataFrame(soap[['No. of Brands', 'Brand Runs', 'Total Volume', 'No. of  Trans', 'Value', 'Trans / Brand Runs',
df2.head()
```

Out[50]:

	No. of Brands	Brand Runs	Total Volume	No. of Trans	Value	Trans / Brand Runs	Vol/Tran	Avg. Price
0	3	17	8025	24	818.0	1.41	334.38	10.19
1	5	25	13975	40	1681.5	1.60	349.38	12.03
2	5	37	23100	63	1950.0	1.70	366.67	8.44
3	2	4	1500	4	114.0	1.00	375.00	7.60
4	3	6	8300	13	591.0	2.17	638.46	7.12

In [51]:
```
# rename column names
df2.rename(columns = {'Trans / Brand Runs': 'Trans/Brand Runs', 'No. of  Trans': 'No. of Trans', 'Avg. Price ': 'Avg. P
            inplace = True)
df2.head()
```

Out[51]:

	No. of Brands	Brand Runs	Total Volume	No. of Trans	Value	Trans/Brand Runs	Vol/Tran	Avg. Price
0	3	17	8025	24	818.0	1.41	334.38	10.19
1	5	25	13975	40	1681.5	1.60	349.38	12.03
2	5	37	23100	63	1950.0	1.70	366.67	8.44
3	2	4	1500	4	114.0	1.00	375.00	7.60
4	3	6	8300	13	591.0	2.17	638.46	7.12

In [52]:
```
# Make of a copy of the predictors
X = df2.copy()
X.head()
```

Out[52]:

	No. of Brands	Brand Runs	Total Volume	No. of Trans	Value	Trans/Brand Runs	Vol/Tran	Avg. Price
0	3	17	8025	24	818.0	1.41	334.38	10.19
1	5	25	13975	40	1681.5	1.60	349.38	12.03
2	5	37	23100	63	1950.0	1.70	366.67	8.44
3	2	4	1500	4	114.0	1.00	375.00	7.60
4	3	6	8300	13	591.0	2.17	638.46	7.12

Then we created a copy of the original data

Rename the columns to work with the libraries

Clustering

Scale Independent Variables

```
In [53]: # standardize the dataset and fit the scaler
         from sklearn.preprocessing import StandardScaler

         scaler = StandardScaler()
         scaler.fit(X)

Out[53]: StandardScaler()

In [54]: X.columns

Out[54]: Index(['No. of Brands', 'Brand Runs', 'Total Volume', 'No. of Trans', 'Value',
                'Trans/Brand Runs', 'Vol/Tran', 'Avg. Price'],
               dtype='object')

In [56]: # transform data and save values as _z
         X[['No. of Brands_z', 'Brand Runs_z', 'Total Volume_z', 'No. of Trans_z', 'Value_z', 'Tr
         X.head()

Out[56]:
```

	No. of Brands	Brand Runs	Total Volume	No. of Trans	Value	Trans/Brand Runs	Vol/Tran	Avg. Price	No. of Brands_z	Brand Runs_z	Total Volume_z	No. of Trans_z	V
0	3	17	8025	24	818.0	1.41	334.38	10.19	-0.403364	0.120173	-0.501007	-0.410811	-0.5
1	5	25	13975	40	1681.5	1.60	349.38	12.03	0.863748	0.890306	0.265360	0.508057	0.3
2	5	37	23100	63	1950.0	1.70	366.67	8.44	0.863748	2.045506	1.440672	1.828930	0.6
3	2	4	1500	4	114.0	1.00	375.00	7.60	-1.036920	-1.131294	-1.341436	-1.559396	-1.3
4	3	6	8300	13	591.0	2.17	638.46	7.12	-0.403364	-0.938760	-0.465587	-1.042532	-0.8

We needed to scale the data with StandardScaler

We used the new scaled values to create new columns with _z after them

Create Silhouette Function To Find Optimal K

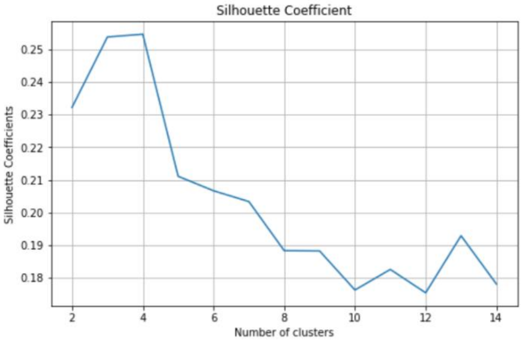
```
In [57]: # define a silhouette function
def silhouette(min_k,max_k, X):
    silhouette_avgs = []

    # --- try k from 2 to maximum number of labels ---
    for k in range(min_k, max_k):
        kmean = KMeans(n_clusters = k).fit(X)
        score = metrics.silhouette_score(X, kmean.labels_)
        print('Silhouette Coefficient for k = ',k,' is ', score)
        silhouette_avgs.append(score)

    # --- the optimal k is the one with the highet average silhouette ---
    Optimal_K = silhouette_avgs.index(max(silhouette_avgs)) + min_k
    print('Optimal K is ', Optimal_K)

    f, ax = plt.subplots(figsize=(8,5))
    ax.plot(range(min_k, max_k),silhouette_avgs)

    plt.title('Silhouette Coefficient')
    plt.xlabel('Number of clusters')
    plt.ylabel('Silhouette Coefficients')
    plt.grid(True)
    plt.show()
```

Code to run a silhouette from sklearn to find the ideal number of clusters

Observe Silhouette Coefficients to Find Optimal K

```
silhouette(2,15,X[['No. of Brands_z', 'Brand Runs_z', 'Total Volume_z', 'No. of Trans_z', 'Value_z', 'Trans/Brand Runs_
```

```
Silhouette Coefficient for k =  2  is  0.23226798276683952
Silhouette Coefficient for k =  3  is  0.2538459915403394
Silhouette Coefficient for k =  4  is  0.2546937819766269
Silhouette Coefficient for k =  5  is  0.2111238306354919
Silhouette Coefficient for k =  6  is  0.20668757179903363
Silhouette Coefficient for k =  7  is  0.2033792786570909
Silhouette Coefficient for k =  8  is  0.18831964834769432
Silhouette Coefficient for k =  9  is  0.18823617148137714
Silhouette Coefficient for k =  10  is  0.1762589716176871
Silhouette Coefficient for k =  11  is  0.18256015332771944
Silhouette Coefficient for k =  12  is  0.17539507049084505
Silhouette Coefficient for k =  13  is  0.19289938661294
Silhouette Coefficient for k =  14  is  0.1780935446226228
Optimal K is  4
```



In [61]:
```
# fit model with data
kmeans_soap = KMeans(n_clusters = 4)
kmeans_soap.fit(X[['No. of Brands_z', 'Brand Runs_z', 'Total Volume_z', 'No. of Trans_z', 'Value_z', 'Trans/Brand Runs_
```

We ran a silhouette from sklearn to find the optimal K. The K value is the optimal number of clusters for our data set

Create K Means Cluster With Optimal K

```
In [61]: # fit model with data
         kmeans_soap = KMeans(n_clusters = 4)
         kmeans_soap.fit(X[['No. of Brands_z', 'Brand Runs_z', 'Total Volume_z', 'No. of Trans_z', 'Value_z', 'Trans/Brand Runs_

Out[61]: KMeans(n_clusters=4)

In [63]: # obtain the labels
         cluster = kmeans_soap.labels_
         cluster

Out[63]: array([0, 1, 1, 0, 0, 1, 0, 0, 2, 0, 1, 1, 2, 0, 0, 1, 1, 0, 0, 0, 0, 3,
                2, 0, 0, 1, 3, 0, 0, 0, 3, 0, 0, 3, 3, 0, 0, 0, 3, 0, 0, 0, 2, 0,
                2, 0, 0, 0, 3, 1, 2, 0, 0, 2, 3, 2, 2, 1, 3, 0, 1, 1, 3, 0, 1, 0,
                1, 1, 0, 3, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 3, 0, 0, 1, 0, 2,
                0, 0, 1, 2, 0, 0, 1, 1, 2, 1, 0, 0, 0, 0, 2, 1, 0, 1, 1, 0, 1, 0,
                0, 0, 0, 1, 2, 1, 1, 0, 1, 1, 0, 1, 0, 2, 2, 2, 2, 0, 0, 2, 1, 1,
                0, 0, 3, 0, 0, 1, 1, 2, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                1, 1, 1, 0, 0, 2, 0, 3, 2, 1, 2, 1, 2, 0, 1, 0, 1, 0, 0, 3, 2, 1,
                3, 0, 3, 0, 0, 2, 1, 1, 1, 0, 2, 1, 2, 1, 0, 1, 0, 0, 1, 1, 1, 1,
                1, 0, 3, 2, 2, 0, 3, 0, 0, 2, 0, 1, 0, 0, 1, 0, 2, 0, 0, 1, 0, 1,
                0, 2, 2, 0, 0, 0, 0, 1, 0, 0, 0, 2, 3, 3, 0, 2, 1, 0, 3, 0, 1, 1,
                0, 0, 0, 0, 0, 3, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 2, 1, 0, 0, 1, 1,
                1, 0, 0, 0, 0, 1, 2, 2, 0, 0, 0, 0, 1, 1, 0, 0, 2, 0, 1, 2, 2, 0,
                1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0,
                0, 0, 0, 1, 1, 1, 2, 1, 0, 0, 1, 3, 2, 1, 2, 1, 0, 0, 1, 0, 1, 0,
                1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 2, 0, 1, 0, 1, 0, 1, 0, 0, 0,
                0, 3, 1, 1, 1, 1, 1, 1, 0, 0, 1, 2, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 2, 1, 0, 0, 1, 0,
                0, 0, 0, 3, 0, 1, 2, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0,
                1, 1, 2, 0, 0, 0, 0, 1, 0, 0, 1, 0, 2, 0, 0, 0, 0, 3, 0, 1, 0, 0,
                0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 1, 1, 1, 0, 0,
                0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 2, 2, 1, 1, 0, 1, 0, 1, 1, 0,
                0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 2, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0,
                0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 2, 0, 0, 1, 0,
```

Here we fit the model with the data and then printed the 600 values. These are the clusters which the values have been assigned to 0-3

Find Cluster Centers and Add Them to a Dataframe

[64]:
```
# obtain centroids
cluster_center = kmeans_soap.cluster_centers_
cluster_center
```

[64]:
```
array([[-0.44346251, -0.49672577, -0.50572134, -0.56365233, -0.50253629,
        -0.15616554, -0.11312966,  0.01515554],
       [ 0.90234034,  1.00025531,  0.15896651,  0.88499154,  0.33411686,
        -0.27609669, -0.48222461,  0.25998408],
       [-0.09750938, -0.05410296,  2.04110837,  0.22685214,  1.72155916,
         0.0405349 ,  1.86779271, -0.45866719],
       [-1.10246037, -1.27403381,  0.34852598, -0.32367653, -0.23689603,
         3.49614947,  0.7729394 , -1.01390754]])
```

[65]:
```
# merge data to create clusters
soap_cluster = pd.concat([X,pd.DataFrame(cluster, columns=['cluster'])],
                         axis=1)
soap_cluster.head()
```

[65]:

	No. of Brands	Brand Runs	Total Volume	No. of Trans	Value	Trans/Brand Runs	Vol/Tran	Avg. Price	No. of Brands_z	Brand Runs_z	Total Volume_z	No. of Trans_z	Value_z	Trans/B Ru
0	3	17	8025	24	818.0	1.41	334.38	10.19	-0.403364	0.120173	-0.501007	-0.410811	-0.588594	-0.46
1	5	25	13975	40	1681.5	1.60	349.38	12.03	0.863748	0.890306	0.265360	0.508057	0.389966	-0.39
2	5	37	23100	63	1950.0	1.70	366.67	8.44	0.863748	2.045506	1.440672	1.828930	0.694243	-0.35
3	2	4	1500	4	114.0	1.00	375.00	7.60	-1.036920	-1.131294	-1.341436	-1.559396	-1.386401	-0.62
4	3	6	8300	13	591.0	2.17	638.46	7.12	-0.403364	-0.938760	-0.465587	-1.042532	-0.845841	-0.17

Found the centroids of the data

Logistic Regression

Find Impactful Independent Variables And Split Variables into Test and Train

In [51]:

```
# find correlations between variables
soap_cluster[['No. of Brands_z', 'Brand Runs_z', 'Total Volume_z', 'No. of Trans_z', 'Value_z', 'Trans/Brand Runs_z', 'Vol/Tr
```

Out[51]:

	No. of Brands_z	Brand Runs_z	Total Volume_z	No. of Trans_z	Value_z	Trans/Brand Runs_z	Vol/Tran_z	Avg. Price_z	cluster
No. of Brands_z	1.000000	0.688973	0.213046	0.543916	0.302480	-0.318563	-0.230881	0.097313	-0.536383
Brand Runs_z	0.688973	1.000000	0.252600	0.774296	0.383639	-0.372869	-0.311542	0.174813	-0.585194
Total Volume_z	0.213046	0.252600	1.000000	0.494565	0.876424	0.198094	0.620242	-0.250296	0.234676
No. of Trans_z	0.543916	0.774296	0.494565	1.000000	0.571126	0.016534	-0.234622	0.060278	-0.369720
Value_z	0.302480	0.383639	0.876424	0.571126	1.000000	0.047604	0.417338	0.154447	0.042882
Trans/Brand Runs_z	-0.318563	-0.372869	0.198094	0.016534	0.047604	1.000000	0.220249	-0.252831	0.564398
Vol/Tran_z	-0.230881	-0.311542	0.620242	-0.234622	0.417338	0.220249	1.000000	-0.348726	0.531848
Avg. Price_z	0.097313	0.174813	-0.250296	0.060278	0.154447	-0.252831	-0.348726	1.000000	-0.300551
cluster	-0.536383	-0.585194	0.234676	-0.369720	0.042882	0.564398	0.531848	-0.300551	1.000000

In [52]:

```
plit data into training and testing
m sklearn.model_selection import train_test_split

rain, X_test, y_train, y_test = train_test_split(soap_cluster.loc[:, ['Total Volume_z', 'Trans/Brand Runs_z', 'Vol/Tran_z']],
                                                 soap_cluster['cluster'],
                                                 test_size = 0.3,
                                                 random_state = 1)
rain.head()
```

Out[52]:

	Total Volume_z	Trans/Brand Runs_z	Vol/Tran_z
241	-0.311026	-0.302701	-0.974924
400	-1.447697	-0.621624	-0.312010
286	-0.488127	-0.371865	-0.809608
379	-0.202188	-0.337283	-0.784302
314	4.435303	0.020064	1.660118

The most impactful variables were:

1. Trans/Brand_Runs_z
2. Vol/Tran_z
3. Total_Volume_z

Then we split the data into training and testing with 70% of the values used for training

Create Linear Regression Model and Get Prediction Probabilities

```
In [84]: y_train.head()

Out[84]: 241    1
         400    0
         286    1
         379    1
         314    2
         Name: cluster, dtype: int32
```

```
In [85]: # train the model
         from sklearn.linear_model import LogisticRegression

         x = X_train
         y = y_train
         log_regress = LogisticRegression(solver = 'liblinear')
         log_regress.fit(X = x, y = y)

         # intercept and coefficients
         print(log_regress.intercept_)
         print(log_regress.coef_)

         [ 0.09228282 -1.77357947 -4.01759573 -4.24058975]
         [[-3.32392131 -0.43532688  1.89339801]
          [ 2.11557167 -1.64222701 -3.26561464]
          [ 2.09966493 -0.97197184  1.63963364]
          [-0.57095918  2.0381374   0.29782236]]
```

```
In [86]: # test the model
         test_prob = log_regress.predict_proba(X = X_test)

         preds_prob = pd.DataFrame(test_prob)

         preds_prob.head()
```

Out[86]:

	0	1	2	3
0	0.950296	0.024927	0.016059	0.008718
1	0.923375	0.061207	0.007447	0.007971

Train and test the model

Retrieve Predictions Part 1

```
In [87]: # get predicted class labels
preds = log_regress.predict(X = X_test)

preds_class = pd.DataFrame(preds)
preds_class.columns = ['Prediction']

# actual diagnosis from historical data, y_test
original_result = pd.DataFrame(y_test.values)
original_result.columns = ['Original Result']

# merge the three dataframes together
result = pd.concat([preds_prob, preds_class, original_result], axis = 1)
print(result.head())
```

```
          0         1         2         3  Prediction  Original Result
0  0.950296  0.024927  0.016059  0.008718           0                0
1  0.923375  0.061207  0.007447  0.007971           0                0
2  0.484344  0.503598  0.007268  0.004790           1                1
3  0.820028  0.138757  0.035644  0.005572           0                1
4  0.262030  0.004393  0.724960  0.008617           2                2
```

```
In [95]: # print Confusion Matrix for Training data
metrics.ConfusionMatrixDisplay.from_estimator(log_regress,
                                              X_train,
                                              y_train,
                                             cmap = 'Blues',
                                             colorbar = False)

plt.title('Confusion Matrix - Train Data');
```

Retrieve Predictions Part 2

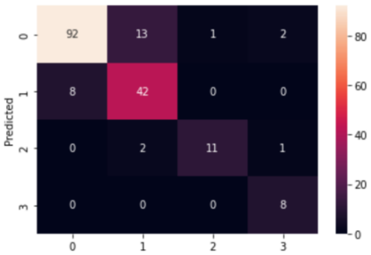
Run a metrics

```
preds = log_regress.predict(X_train)
print(metrics.classification_report(y_true = y_train,
                                    y_pred = preds));
```

	precision	recall	f1-score	support
0	0.84	0.93	0.88	216
1	0.85	0.75	0.80	140
2	0.97	0.85	0.91	46
3	0.88	0.83	0.86	18
accuracy			0.86	420
macro avg	0.89	0.84	0.86	420
weighted avg	0.86	0.86	0.86	420

```
In [90]: # print Confusion Matrix for testing data
print('Confusion Matrix - Testing Data')
confusion_matrix = pd.crosstab(preds,
                               y_test,
                               rownames=['Predicted'],
                               colnames=['Actual'])
sns.heatmap(confusion_matrix, annot=True);
```

Confusion Matrix - Testing Data



Print Classification Report

```
[91]: # view summary of common classification metrics
print('Metrices - Testing Data')
print(metrics.classification_report(y_true = y_test,
                                    y_pred = preds));
```

```
Metrices - Testing Data
              precision    recall  f1-score   support

           0       0.85      0.92      0.88       100
           1       0.84      0.74      0.79        57
           2       0.79      0.92      0.85        12
           3       1.00      0.73      0.84        11

    accuracy                           0.85       180
   macro avg       0.87      0.83      0.84       180
weighted avg       0.85      0.85      0.85       180
```

What factors contribute to the outcome the best? What and how can the crew in the Marketing department do with the results?

The factors tha best contributed to the outcome were 'Total Volume', 'Trans/Brand Runs', 'Vol/Tran'. The crew in the Marketing department can use the results to run better marketing campaigns, better target potential customers, and deploy promotion budgets more effectively.