Project I: Customer Segmentation

Erin Howland

DSC680

25 Sep 2022

Business Problem

Customer segmentation can be used to track geographic, demographic, psychographic, and behavioral data from customers for product improvements, to improve marketing techniques, and ultimately, increased sales.

Background/History

Early marketing did not take into account customer segmentation – all customers, current or potential, were considered part of a single group (Nandakumar, 2016). Undifferentiated customers aren't the reality, though, so once companies had better knowledge of data collection and storage, they were able to increasingly segment their customers into more accurate groupings. Different customers have different needs and different purchasing habits and power; for example, marketing a Mercedes to a college student probably isn't going to get a very good return on investment. Understanding your customers – current and potential – can help develop effective marketing campaigns and grow the business. It can also help to point out areas of growth potential. For example, and auto maker may look at data to determine if the development of a new type of vehicle would be a good idea or not (think Saturn's foray into crossover SUVs in the early 2000s). By examining customer data, particularly when done in conjunction with financial analysis, a company can make wiser decisions on how to develop and market its products and services.

Data Explanation (Data Prep/Data Dictionary/etc.)

I used a dataset from Kaggle to use for customer segmentation (2022). The process of cleaning and preparing the data was pretty standard: check for and handle missing data and ensure all data was able to be read by an algorithm (i.e. convert strings to numeric). Once the data was prepped, the model building could begin.

Methods

Customer segmentation is ultimately a classification task. There are two general categories from which to select an algorithm: supervised and unsupervised learning algorithms. The benefits of supervised learning is that labeled data is easier to understand and explain to a board/stockholders, though it can be more difficult to obtain. Unsupervised learning, while less easy to explain to a board/stockholders, is better at finding patterns that humans might not be able to easily visualize which could lead to a new business vertical (Joy, n.d.). Because each of these two groups has their pros and cons, I decided to look at segmentation using algorithms from each group. I selected four supervised learning algorithms – logistic regression, random forest classifier, decision tree classifier, and KNN – and two unsupervised learning algorithms – KMeans and Gaussian mixture.

Analysis

As expected the unsupervised learning algorithms were difficult to comprehend when compared to the easily measured results of the supervised learning algorithms. The supervised learning algorithm results can be summed up in a table:

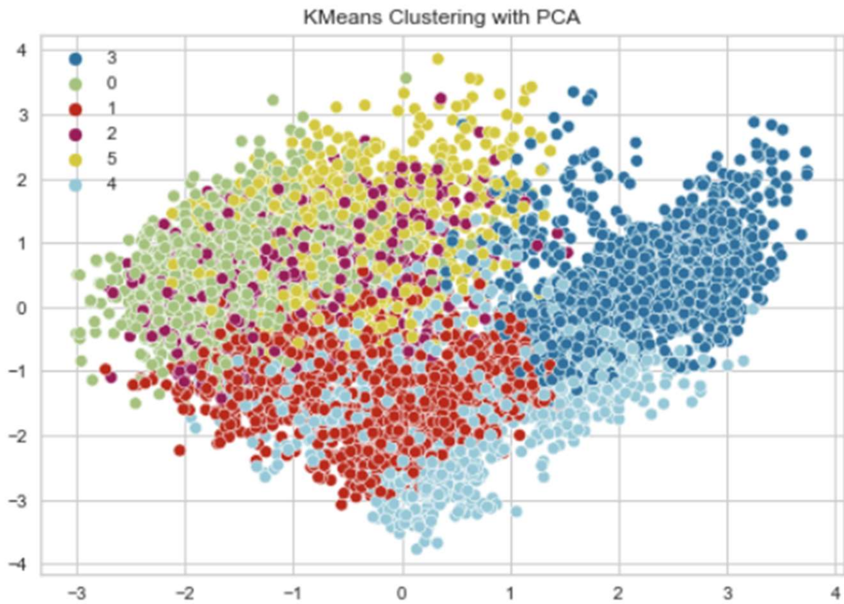
	Model	Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F1
0	Logistic Regression	49.364735	47.645601	45.446265	44.246809	47.788177	45.685218	43.385447	41.739478
1	DTC	100.000000	43.742255	100.000000	42.944249	100.000000	42.814016	100.000000	42.869354
2	RFC	100.000000	49.938042	100.000000	48.371756	100.000000	48.720002	100.000000	48.499604
3	KNN	63.634955	46.716233	63.855739	46.446785	63.288812	45.927231	63.346458	46.033737

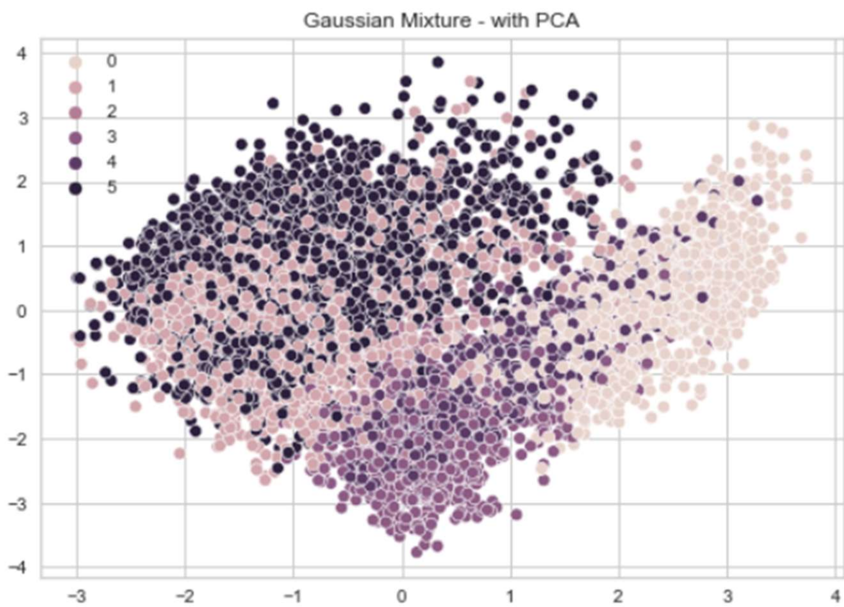
Here we see a comparison of the four supervised training algorithms used: logistic regression, decision tree classifier (DTC) , random forest classifier (RFC), and K nearest neighbours (KNN). I wanted to compare training and test accuracy, precision, recall, and F1 scores.

- Accuracy – tells us how close to truth we are
- Precision – “the ratio of correctly predicted positive observations to the total predicted positive observations” (Exsilio Solutions, 2016)
- Recall – tells us how sensitive the model is, or, put another way, of the positive outcomes, how many did we label (Exsilio Solutions, 2016)
- F1 Score – the weighted average of precision and recall

DTC and RFC have scores of 100% for training data because they are given all the information they need so they will predict accurately on training data every time. We’re mostly interested in the test results. We can see that across the board, RFC has the highest scores for accuracy, precision, recall, and F1. KNN isn’t far behind in terms of scoring when we look at the training data. Compared to RFC, there’s significantly less drop in scores between the training and test data.

Understanding the unsupervised algorithms is more difficult. Probably the best way to consider this is by looking at the visuals they produce to see which one creates more distinctive segments and by looking at the silhouette score of the model. A silhouette score is used to show, numerically, how good the clustering technique is, and the values range from -1 to 1, with -1 meaning the clusters are assigned improperly, 0 meaning clusters are indifferent/not significant, and 1 meaning the clusters are clearly distinct. When we look at KMeans (top) and Gaussian mixture (bottom), KMeans seems to have more distinct groupings, and this is reflected in their silhouette scores, which is 0.170 for KMeans and 0.103 for the Gaussian mixture.





Things that could change this include changing the number of clusters we want to see – I used the elbow score for KMeans to determine the number of clusters I would use, which ended up being six clusters. I used six for both KMeans and Gaussian so we could get as equal comparison between the two models. Changing the number of clusters can have a fairly significant effect on the result, which is worth noting as none of these scores are particularly impressive; scores will naturally increase as you decrease the number of clusters. If you're interested in seeing the change in visuals from above and below the elbow or what KMeans clustering visual looks like without PCA applied, see the code in the Appendix.

Conclusion

I have two general selections as we have two types of algorithms. Of the two tested unsupervised learning algorithms, it seems KMeans does a better job than Gaussian, at least if we go based on the segmentation visuals and silhouette score based on the number of clusters to select from the elbow score. Because of the different ways different algorithms work, it could be that the two work differently enough that comparing scores by using the elbow score for KMeans on both algorithms may not actually be as fair as I thought it would be. From the supervised learning algorithms, I would select random forest or KNN. Which we ultimately use would depend on a few factors: further testing on more data and preference of the decision makers.

Assumptions

Classification models on supervised learning algorithms is fairly simple, but customer segmentation is more nuanced than what a binary response can provide. To adjust for the use of multiclass variables, an average had to be selected. I used sklearn for this, which provides four averages to choose from: macro, micro, weighted average, and samples. According to the documentation from sklearn (n.d.), these are how each are treated:

- Micro – "calculate metrics globally by considering each element of the label indicator matrix as a label
- Macro – "calculate metrics for each label, and find their weighted mean; does not take label imbalance into account"
- Weighted – "calculate metrics for each label and find their average, weighted by the support (the number of true instances for each label)"
- Samples – "calculate metrics for each instance and find their average"

I decided to use macro, though we could do additional testing to see if there are significant differences with the use of one of the other available options.

We also must assume that the data we are given is not missing a factor or factors that would significantly change the results of testing. And, as mentioned earlier, I assumed that using the same number of clusters for both KMeans and Gaussian would be a fair comparison, and it's possible that assumption is not correct.

Limitations

Limitations of this analysis includes time – with more time, more data could have been tested and/or more algorithms be tested, particularly unsupervised learning algorithms. We are also always at the mercy of the data collected – its completeness, its relevance, the total amount of data available can all make a difference in the results.

Challenges

Unsupervised algorithms are more difficult to determine accuracy and precision. In fact, they are inherently not set up in a way that allows for these kinds of measures. Because of this, it was not possible to get a consistent measure by which the various tested algorithms could be compared across both general categories (supervised and unsupervised). Rather, we can get fair comparisons within each category. This can make is difficult to speak to decision makers about since it can be difficult to help them fully understand why we can't make a full comparison between two algorithm categories. While unsupervised learning is traditionally used for customer segmentation, it's also possible that because we don't fully understand the ins and outs of how they work in the same way we do for supervised learning, the recommendation of the use of an unsupervised learning technique could end up being potentially dangerous if, for example, a company makes a decision based on the algorithm but the algorithm was actually incorrect.

Future Uses/Additional Applications

Additional application of segmentation can be used in market research should the company wish to come out with a new product/service. It could also be used if the company wanted to create a new branch or subsidiary – segmentation would allow for analysis of current customers who would either move from one product/subsidiary to another or whether they would increase the number of products/services owned as a result of the growth or diversification of the company and/or products /services offers.

Recommendations

Were we to go ahead with this project now, I would recommend the use of KMeans clustering, though I would want to better dial in the model first. I would choose this algorithm because unsupervised learning is traditionally used for customer segmentation, and for good reason. However, KNN, as a supervised algorithm, is easier to explain to a board and to stockholders. I would present both as options and discuss their general pros and cons and let the decision makers do just that – make a decision.

Implementation Plan

Customer segmentation would be best used when combined with other research, especially financial analysis. Because we now have a way to segment customers, we can now apply additional information to determine how that might shift things and whether a project would be worth the cost of doing (e.g. a sale, a new product/service, etc.).

Ethical Assessment

Data collection is typically the weakest point in a data science project. We often do not know the methods used to collect the data – were the customers aware and had they consented to this data being collected and used? What potential biases went into the selection of criteria to be collected that could have been amplified by the training algorithms? As data scientists, we have to do the best we can with the information we have available. When possible, getting information as to how, when, where, etc. the data was collected can give us some insight as to whether there are adjustments we should take into account when building our models – sometimes we can get this from the initial data exploration (e.g. seeing is we have an imbalanced dataset), but not always – and if possible, we can ask about the data collection process so we know if it’s even ethical to work on the data to begin with.

Sources

Exsilio Solutions. (2016, Sep 9). “Accuracy, Precision, Recall, & F1 Score: Interpretation of Performance Measures.” Accessed on 24 September 2022 from https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/.

Gupta, Yash. (2020). “Customer Segmentation Dataset.” Kaggle. Accessed on 4 September 2022 from https://www.kaggle.com/datasets/yashgupta011/customer-segmentation-dataset.

Joy, Ashwin. (n.d.). “Pros and Cons of Unsupervised Learning.” Pythonista Planet. Accessed on 24 September 2022 from https://pythonistaplanet.com/pros-and-cons-of-unsupervised-learning/.

Kash. (2022). “Customer Segmentation Classification.” Kaggle. Accessed on 4 September 2022 from https://www.kaggle.com/datasets/kaushiksuresh147/customer-segmentation.

Kumar, Ajitesh. (2020, Sep 4). “Micro-average & Macro-average Scoring Metrics – Python.” VitalFlux. Accessed on 24 September 2022 from https://vitalflux.com/micro-average-macro-average-scoring-metrics-multi-class-classification-python/.

McGregor, Milecia. (2020, Sep 21). “8 Clustering Algorithms in Machine Learning that All Data Scientists Should Know.” Free Code Camp. Accessed on 4 September 2022 from https://www.freecodecamp.org/news/8-clustering-algorithms-in-machine-learning-that-all-data-scientists-should-know/.

Nandakumar, Priya Ragavi. (2016, Nov 28). “Evolution of Customer Segmentation.” LinkedIn. Accessed on 24 September 2022 from https://www.linkedin.com/pulse/evolution-customer-segmentation-priya-ragavi-nandakumar/.

Profile Tree. (n.d.). “The 4 Types of Customer Segmentation: How to Correctly Apply Them and the Value They Add.” Accessed on 4 September 2022 from https://profiletree.com/customer-segmentation/.

Scikit Learn Documentation. (n.d.). “sklearn.metrics.average_precision_score.” Accessed on 24 September 2022 from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html.

Project 1: Customer Segmentation Erin Howland DSC680 Appendix

In [1]:

```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, confusion_matrix,
precision_score, recall_score, f1_score

from collections import Counter
```

In [2]:

```
# import initial datasets
# https://www.kaggle.com/datasets/kaushiksuresh147/customer-segmentation
df_test = pd.read_csv(r'C:\GitHub\DSC680\Project
1\Datasets\Kaggle_kaushiksuresh147\Test.csv')
df_train = pd.read_csv(r'C:\GitHub\DSC680\Project
1\Datasets\Kaggle_kaushiksuresh147\Train.csv')
```

INITIAL DATA EXPLORATION

In [3]:

```
df_train.head()
```

Out[3]:

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Fa
0	462809	Male	No	22	No	Healthcare	1.0	Low	
1	462643	Female	Yes	38	Yes	Engineer	NaN	Average	
2	466315	Female	Yes	67	Yes	Engineer	1.0	Low	
3	461735	Male	Yes	67	Yes	Lawyer	0.0	High	
4	462669	Female	Yes	40	Yes	Entertainment	NaN	High	

In [4]:

```
df_test.head()
```

Out[4]:

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Famil

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Famil
0	458989	Female	Yes	36	Yes	Engineer	0.0	Low	
1	458994	Male	Yes	37	Yes	Healthcare	8.0	Average	
2	458996	Female	Yes	69	No	NaN	0.0	Low	
3	459000	Male	Yes	59	No	Executive	11.0	High	
4	459001	Female	No	19	No	Marketing	NaN	Low	

In [5]:

```
# df_test information

print('df_test shape: \n', df_test.shape, '\n')
print('df_test.describe: \n', df_test.describe(), '\n\n', 'df_test.info:
\n')
print(df_test.info(), '\n')
print('df_test null: \n', df_test.isnull().sum(), '\n')
print('df_test unique: \n', df_test.nunique())
```

```
df_test shape: 
 (2627, 11) 

df_test.describe: 
                   ID          Age  Work_Experience  Family_Size
count    2627.000000  2627.000000      2358.000000  2514.000000
mean   463433.918919    43.649791         2.552587     2.825378
std      2618.245698    16.967015         3.341094     1.551906
min    458989.000000    18.000000         0.000000     1.000000
25%    461162.500000    30.000000         0.000000     2.000000
50%    463379.000000    41.000000         1.000000     2.000000
75%    465696.000000    53.000000         4.000000     4.000000
max    467968.000000    89.000000        14.000000     9.000000 

 df_test.info: 

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2627 entries, 0 to 2626
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               2627 non-null   int64  
 1   Gender           2627 non-null   object 
 2   Ever_Married     2577 non-null   object 
 3   Age              2627 non-null   int64  
 4   Graduated        2603 non-null   object 
 5   Profession       2589 non-null   object 
 6   Work_Experience  2358 non-null   float64
 7   Spending_Score   2627 non-null   object 
 8   Family_Size      2514 non-null   float64
 9   Var_1            2595 non-null   object 
 10  Segmentation     2627 non-null   object 
dtypes: float64(2), int64(2), object(7)
memory usage: 225.9+ KB
None 

df_test null: 
```

```
 ID                  0
Gender               0
Ever_Married        50
Age                  0
Graduated           24
Profession          38
Work_Experience    269
Spending_Score       0
Family_Size        113
Var_1               32
Segmentation         0
dtype: int64

df_test unique:
 ID                2627
Gender               2
Ever_Married         2
Age                 67
Graduated            2
Profession           9
Work_Experience     15
Spending_Score       3
Family_Size          9
Var_1                7
Segmentation         4
dtype: int64
```

In [6]:

```python
# df_train information


print('df_train shape: \n', df_train.shape, '\n')
print('df_train.describe: \n', df_train.describe(), '\n\n', 'df_train.info: \n')
print(df_train.info(), '\n')
print('df_train null: \n', df_train.isnull().sum(), '\n')
print('df_train unique: \n', df_train.nunique())
```

```
df_train shape: 
 (8068, 11) 

df_train.describe: 
                   ID          Age  Work_Experience  Family_Size
count    8068.000000  8068.000000      7239.000000  7733.000000
mean   463479.214551    43.466906         2.641663     2.850123
std      2595.381232    16.711696         3.406763     1.531413
min    458982.000000    18.000000         0.000000     1.000000
25%    461240.750000    30.000000         0.000000     2.000000
50%    463472.500000    40.000000         1.000000     3.000000
75%    465744.250000    53.000000         4.000000     4.000000
max    467974.000000    89.000000        14.000000     9.000000 

 df_train.info: 

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8068 entries, 0 to 8067
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   ID               8068 non-null   int64 
 1   Gender           8068 non-null   object
```

```
 2   Ever_Married     7928 non-null   object
 3   Age              8068 non-null   int64
 4   Graduated        7990 non-null   object
 5   Profession       7944 non-null   object
 6   Work_Experience  7239 non-null   float64
 7   Spending_Score   8068 non-null   object
 8   Family_Size      7733 non-null   float64
 9   Var_1            7992 non-null   object
 10  Segmentation     8068 non-null   object
dtypes: float64(2), int64(2), object(7)
memory usage: 693.5+ KB
None

df_train null:
 ID                   0
Gender               0
Ever_Married       140
Age                  0
Graduated           78
Profession         124
Work_Experience    829
Spending_Score       0
Family_Size        335
Var_1               76
Segmentation         0
dtype: int64

df_train unique:
 ID                 8068
Gender                2
Ever_Married          2
Age                  67
Graduated             2
Profession            9
Work_Experience      15
Spending_Score        3
Family_Size           9
Var_1                 7
Segmentation          4
dtype: int64
```

PREP TRAINING AND TEST DATA FOR ANALYSIS

In [7]:

```
# drop inapplicable cols: train
df_train.drop(columns = 'Var_1', inplace = True)
df_train.head()
```

Out[7]:

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Fa
0	462809	Male	No	22	No	Healthcare	1.0	Low	
1	462643	Female	Yes	38	Yes	Engineer	NaN	Average	
2	466315	Female	Yes	67	Yes	Engineer	1.0	Low	
3	461735	Male	Yes	67	Yes	Lawyer	0.0	High	
4	462669	Female	Yes	40	Yes	Entertainment	NaN	High	

In [8]:

```
# handle null data: train
```

```python
# fill null object data (see df_train.info) with most common response
object_col = df_train.select_dtypes(include = 'object').columns

for column in object_col:
    df_train[column].fillna(df_train[column].mode()[0], inplace = True)

# fill null float64
float_col = df_train.select_dtypes(include = 'float64').columns

for column in float_col:
    df_train[column].fillna(df_train[column].mean(), inplace = True)

# no null int64 data
```

In [9]:

```python
# vew cleaned training data: train
print('df_train shape: \n', df_train.shape, '\n')
print('df_train.describe: \n', df_train.describe(), '\n\n', 'df_train.info:
\n')
print(df_train.info(), '\n')
print('df_train null: \n', df_train.isnull().sum(), '\n')
print('df_train unique: \n', df_train.nunique())
```

```
df_train shape: 
 (8068, 10) 

df_train.describe: 
                   ID          Age  Work_Experience  Family_Size
count    8068.000000  8068.000000      8068.000000  8068.000000
mean   463479.214551    43.466906         2.641663     2.850123
std      2595.381232    16.711696         3.226972     1.499278
min    458982.000000    18.000000         0.000000     1.000000
25%    461240.750000    30.000000         0.000000     2.000000
50%    463472.500000    40.000000         1.000000     2.850123
75%    465744.250000    53.000000         4.000000     4.000000
max    467974.000000    89.000000        14.000000     9.000000 

 df_train.info: 

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8068 entries, 0 to 8067
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               8068 non-null   int64  
 1   Gender           8068 non-null   object 
 2   Ever_Married     8068 non-null   object 
 3   Age              8068 non-null   int64  
 4   Graduated        8068 non-null   object 
 5   Profession       8068 non-null   object 
```

```
 6   Work_Experience  8068 non-null   float64
 7   Spending_Score   8068 non-null   object 
 8   Family_Size      8068 non-null   float64
 9   Segmentation     8068 non-null   object 
dtypes: float64(2), int64(2), object(6)
memory usage: 630.4+ KB
None

df_train null:
 ID                 0
Gender             0
Ever_Married       0
Age                0
Graduated          0
Profession         0
Work_Experience    0
Spending_Score     0
Family_Size        0
Segmentation       0
dtype: int64

df_train unique:
 ID                 8068
Gender                2
Ever_Married          2
Age                  67
Graduated             2
Profession            9
Work_Experience      16
Spending_Score        3
Family_Size          10
Segmentation          4
dtype: int64
```

In [10]:
```
# handle non-numeric data: train
labelencoder = LabelEncoder()

for column in object_col:
    df_train[column] = labelencoder.fit_transform(df_train[column])
```

In [11]:
```
# view cleaned training df
df_train.head()
```

Out[11]:

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Famil
0	462809	1	0	22	0	5	1.000000	2	
1	462643	0	1	38	1	2	2.641663	0	
2	466315	0	1	67	1	2	1.000000	2	
3	461735	1	1	67	1	7	0.000000	1	
4	462669	0	1	40	1	3	2.641663	1	

In [12]:
```
# vew cleaned training data info
```

```python
print('df_train shape: \n', df_train.shape, '\n')
print('df_train.describe: \n', df_train.describe(), '\n\n', 'df_train.info:
\n')
print(df_train.info(), '\n')
print('df_train null: \n', df_train.isnull().sum(), '\n')
print('df_train unique: \n', df_train.nunique())
```

```
df_train shape: 
 (8068, 10) 

df_train.describe: 
                   ID       Gender  Ever_Married          Age    Graduated  \
count    8068.000000  8068.000000   8068.000000  8068.000000  8068.000000   
mean   463479.214551     0.547471      0.592836    43.466906     0.625434   
std      2595.381232     0.497772      0.491336    16.711696     0.484041   
min    458982.000000     0.000000      0.000000    18.000000     0.000000   
25%    461240.750000     0.000000      0.000000    30.000000     0.000000   
50%    463472.500000     1.000000      1.000000    40.000000     1.000000   
75%    465744.250000     1.000000      1.000000    53.000000     1.000000   
max    467974.000000     1.000000      1.000000    89.000000     1.000000   

        Profession  Work_Experience  Spending_Score  Family_Size  Segmentation  
count  8068.000000      8068.000000     8068.000000  8068.000000   8068.000000  
mean      2.746901         2.641663        1.359941     2.850123      1.561973  
std       2.541418         3.226972        0.848418     1.499278      1.139029  
min       0.000000         0.000000        0.000000     1.000000      0.000000  
25%       0.000000         0.000000        1.000000     2.000000      1.000000  
50%       3.000000         1.000000        2.000000     2.850123      2.000000  
75%       5.000000         4.000000        2.000000     4.000000      3.000000  
max       8.000000        14.000000        2.000000     9.000000      3.000000   

 df_train.info: 

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8068 entries, 0 to 8067
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               8068 non-null   int64  
 1   Gender           8068 non-null   int32  
 2   Ever_Married     8068 non-null   int32  
 3   Age              8068 non-null   int64  
 4   Graduated        8068 non-null   int32  
 5   Profession       8068 non-null   int32  
 6   Work_Experience  8068 non-null   float64
 7   Spending_Score   8068 non-null   int32  
 8   Family_Size      8068 non-null   float64
 9   Segmentation     8068 non-null   int32  
dtypes: float64(2), int32(6), int64(2)
memory usage: 441.3 KB
None 

df_train null: 
 ID                 0
Gender             0
Ever_Married       0
Age                0
Graduated          0
Profession         0
Work_Experience    0
Spending_Score     0
```

```
Family_Size        0
Segmentation       0
dtype: int64

df_train unique:
 ID                 8068
Gender                2
Ever_Married          2
Age                  67
Graduated             2
Profession            9
Work_Experience      16
Spending_Score        3
Family_Size          10
Segmentation          4
dtype: int64
```

CREATE MODELS

In [13]:

```
df_model = df_train
```

In [14]:

```
# view heatmap
plt.figure(figsize = (12, 10))
sns.heatmap(df_model.corr(), annot = True)
plt.title('Correlation Heatmap')
plt.show()
```

In [15]:

```
# scale data
ss = StandardScaler()
col_to_scale = ['ID',
                'Gender',
                'Ever_Married',
                'Age',
                'Graduated',
                'Profession',
                'Work_Experience',
                'Spending_Score',
                'Family_Size']
df_model[col_to_scale] = ss.fit_transform(df_model[col_to_scale])
```

CREATE MODELS: SUPERVISED

In [16]:

```
# create fns to show results of accuracy tests
```

```
def score(clf, X_train, y_train, X_test, y_test, train = True):
    if train:
        y_pred = clf.predict(X_train)
        print('Train Result: \n -------------------------------')
        print('Accuracy:', accuracy_score(y_train,y_pred), '\n')
        print('Classification Report \n-------------------------------')
        print('Precision:', precision_score(y_train, y_pred,
average='macro'))
        print('Recall:', recall_score(y_train, y_pred, average='macro'))
        print('F1-score:', f1_score(y_train, y_pred, average='macro'))
        print('-------------------------------\n\nConfusion Matrix:\n',
confusion_matrix(y_train, y_pred))
    elif train == False:
        y_pred = clf.predict(X_test)
        print('Train Result: \n -------------------------------')
        print('Accuracy:', accuracy_score(y_test,y_pred), '\n')
        print('Classification Report \n-------------------------------')
        print('Precision:', precision_score(y_test, y_pred,
average='macro'))
        print('Recall:', recall_score(y_test, y_pred, average='macro'))
        print('F1-score:', f1_score(y_test, y_pred, average='macro'))
        print('-------------------------------\n\nConfusion Matrix:\n',
confusion_matrix(y_test, y_pred))
```

In [17]:

```
# create training and test sets
# target: Segmentation
# features: remaining cols

# drop segmentation since it's the target
X = df_model.drop('Segmentation', axis = 1)
y = df_model['Segmentation']

# split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 42)
```

In [18]:

```
# check shapes
print('X_train shape: ', X_train.shape)
```

```python
print('X-test shape:  ', X_test.shape)
print('y_train shape: ', y_train.shape)
print('y_test shape:  ', y_test.shape)
```

```
X_train shape:  (6454, 9)
X-test shape:   (1614, 9)
y_train shape:  (6454,)
y_test shape:   (1614,)
```

In [19]:

```python
# Model 1: Logistic Regression
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression(solver = 'liblinear')
LR.fit(X_train, y_train)

# score
score(LR, X_train, y_train, X_test, y_test, train=True)
score(LR, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
 ------------------------------
Accuracy: 0.49364735048032227

Classification Report
------------------------------
Precision: 0.45446265143774545
Recall: 0.47788177491063827
F1-score: 0.4338544712561858
------------------------------

Confusion Matrix:
 [[ 700   70  357  454]
 [ 501   84  655  249]
 [ 247   58 1027  258]
 [ 271   31  117 1375]]
Train Result:
 ------------------------------
Accuracy: 0.476456009913259

Classification Report
------------------------------
Precision: 0.4424680938726484
Recall: 0.4568521833800443
F1-score: 0.41739477787785917
------------------------------

Confusion Matrix:
 [[153  17  98 123]
 [112  24 179  54]
 [ 74  15 234  57]
 [ 73  10  33 358]]
```

In [20]:

```python
# begin building model comparison df
test_score_A = accuracy_score(y_test, LR.predict(X_test)) * 100
```

```
train_score_A = accuracy_score(y_train, LR.predict(X_train)) * 100

test_score_P = precision_score(y_test, LR.predict(X_test), average =
'macro') * 100
train_score_P = precision_score(y_train, LR.predict(X_train), average =
'macro') * 100

test_score_R = recall_score(y_test, LR.predict(X_test), average = 'macro')
* 100
train_score_R = recall_score(y_train, LR.predict(X_train), average =
'macro') * 100

test_score_F = f1_score(y_test, LR.predict(X_test), average = 'macro') *
100
train_score_F = f1_score(y_train, LR.predict(X_train), average = 'macro') *
100
```

In [21]:

```
comparison_df = pd.DataFrame(data = [['Logistic Regression',
                                      train_score_A, test_score_A,
                                      train_score_P, test_score_P,
                                      train_score_R, test_score_R,
                                      train_score_F, test_score_F]],
                             columns = ['Model',
                                        'Train Accuracy', 'Test Accuracy',
                                        'Train Precision', 'Test Precision',
                                        'Train Recall', 'Test Recall',
                                        'Train F1', 'Test F1'])


comparison_df
```

Out[21]:

	Model	Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F1
0	Logistic Regression	49.364735	47.645601	45.446265	44.246809	47.788177	45.685218	43.385447	41.739478

In [22]:

```
# Model 2: Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier

DTC = DecisionTreeClassifier(random_state = 42)
```

```
DTC.fit(X_train, y_train)


# score
score(DTC, X_train, y_train, X_test, y_test, train=True)
score(DTC, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
 ===============================
Accuracy: 1.0

Classification Report
-------------------------------
Precision: 1.0
Recall: 1.0
F1-score: 1.0
-------------------------------

Confusion Matrix:
 [[1581    0    0    0]
 [   0 1489    0    0]
 [   0    0 1590    0]
 [   0    0    0 1794]]
Train Result:
 ===============================
Accuracy: 0.43742255266418834

Classification Report
-------------------------------
Precision: 0.4294424905494124
Recall: 0.42814016397757804
F1-score: 0.4286935374054116
-------------------------------

Confusion Matrix:
 [[132  96  66  97]
 [ 90 128 102  49]
 [ 69 110 167  34]
 [ 94  54  47 279]]
```

In [23]:

```
# add DTC to comparison df
test_score_A = accuracy_score(y_test, DTC.predict(X_test)) * 100
train_score_A = accuracy_score(y_train, DTC.predict(X_train)) * 100

test_score_P = precision_score(y_test, DTC.predict(X_test), average =
'macro') * 100
train_score_P = precision_score(y_train, DTC.predict(X_train),average =
'macro') * 100

test_score_R = recall_score(y_test, DTC.predict(X_test), average = 'macro')
* 100
train_score_R = recall_score(y_train, DTC.predict(X_train), average =
'macro') * 100
```

```
test_score_F = f1_score(y_test, DTC.predict(X_test),average = 'macro') *
100
train_score_F = f1_score(y_train, DTC.predict(X_train),average = 'macro') *
100
```

In [24]:

```
# add DTC to comparison df
DTC_df = pd.DataFrame(data = [['DTC',
                               train_score_A, test_score_A,
                               train_score_P, test_score_P,
                               train_score_R, test_score_R,
                               train_score_F, test_score_F]],
                      columns = ['Model',
                                 'Train Accuracy', 'Test Accuracy',
                                 'Train Precision', 'Test Precision',
                                 'Train Recall', 'Test Recall',
                                 'Train F1', 'Test F1'])

comparison_df = comparison_df.append(DTC_df, ignore_index = True)

comparison_df
```

Out[24]:

	Model	Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F
0	Logistic Regression	49.364735	47.645601	45.446265	44.246809	47.788177	45.685218	43.385447	41.73947
1	DTC	100.000000	43.742255	100.000000	42.944249	100.000000	42.814016	100.000000	42.86935

In [25]:

```
# Model 3: Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

RFC = RandomForestClassifier(n_estimators = 1000, random_state = 42)
RFC.fit(X_train, y_train)

# score
score(RFC, X_train, y_train, X_test, y_test, train=True)
score(RFC, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
```

```
 -----------------------------
Accuracy: 1.0

Classification Report
-----------------------------
Precision: 1.0
Recall: 1.0
F1-score: 1.0
-----------------------------

Confusion Matrix:
 [[1581    0    0    0]
 [   0 1489    0    0]
 [   0    0 1590    0]
 [   0    0    0 1794]]
Train Result:
 -----------------------------
Accuracy: 0.4993804213135068

Classification Report
-----------------------------
Precision: 0.4837175565490106
Recall: 0.4872000183374079
F1-score: 0.4849960366042675
-----------------------------

Confusion Matrix:
 [[153  86  54  98]
 [ 97 124 111  37]
 [ 54  81 202  43]
 [ 86  39  22 327]]
```

In [26]:

```
# add RFC to comparison df
test_score_A = accuracy_score(y_test, RFC.predict(X_test)) * 100
train_score_A = accuracy_score(y_train, RFC.predict(X_train)) * 100

test_score_P = precision_score(y_test, RFC.predict(X_test), average =
'macro') * 100
train_score_P = precision_score(y_train, RFC.predict(X_train),average =
'macro') * 100

test_score_R = recall_score(y_test, RFC.predict(X_test), average = 'macro')
* 100
train_score_R = recall_score(y_train, RFC.predict(X_train), average =
'macro') * 100

test_score_F = f1_score(y_test, RFC.predict(X_test),average = 'macro') *
100
train_score_F = f1_score(y_train, RFC.predict(X_train),average = 'macro') *
100
```

In [27]:

```python
RFC_df = pd.DataFrame(data = [['RFC',
                               train_score_A, test_score_A,
                               train_score_P, test_score_P,
                               train_score_R, test_score_R,
                               train_score_F, test_score_F]],
                      columns = ['Model',
                                 'Train Accuracy', 'Test Accuracy',
                                 'Train Precision', 'Test Precision',
                                 'Train Recall', 'Test Recall',
                                 'Train F1', 'Test F1'])

comparison_df = comparison_df.append(RFC_df, ignore_index = True)

comparison_df
```

Out[27]:

	Model	Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F
0	Logistic Regression	49.364735	47.645601	45.446265	44.246809	47.788177	45.685218	43.385447	41.73947
1	DTC	100.000000	43.742255	100.000000	42.944249	100.000000	42.814016	100.000000	42.86935
2	RFC	100.000000	49.938042	100.000000	48.371756	100.000000	48.720002	100.000000	48.49960

In [28]:

```python
# Model 4: KNN
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

# score
score(knn, X_train, y_train, X_test, y_test, train=True)
score(knn, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
 ------------------------------
Accuracy: 0.6363495506662535

Classification Report
------------------------------
Precision: 0.6385573936280862
Recall: 0.6328881185823654
F1-score: 0.6334645779164707
------------------------------

Confusion Matrix:
```

```
 [[1073  210  121  177]
 [ 345  789  253  102]
 [ 204  279 1001  106]
 [ 307  149   94 1244]]
Train Result:
 -------------------------------
Accuracy: 0.46716232961586124

Classification Report
-------------------------------
Precision: 0.46446785205875374
Recall: 0.4592723097282566
F1-score: 0.4603373678299473
-------------------------------

Confusion Matrix:
 [[166  82  64  79]
 [ 92 124 114  39]
 [ 75  86 187  32]
 [131  34  32 277]]
```

In [29]:

```
# add KNN to comparison df
test_score_A = accuracy_score(y_test, knn.predict(X_test)) * 100
train_score_A = accuracy_score(y_train, knn.predict(X_train)) * 100

test_score_P = precision_score(y_test, knn.predict(X_test),average =
'macro') * 100
train_score_P = precision_score(y_train, knn.predict(X_train), average =
'macro') * 100

test_score_R = recall_score(y_test, knn.predict(X_test), average = 'macro')
* 100
train_score_R = recall_score(y_train, knn.predict(X_train),average =
'macro') * 100

test_score_F = f1_score(y_test, knn.predict(X_test),average = 'macro') *
100
train_score_F = f1_score(y_train, knn.predict(X_train), average = 'macro')
* 100
```

In [30]:

```
knn_df = pd.DataFrame(data = [['KNN',
                                train_score_A, test_score_A,
                                train_score_P, test_score_P,
                                train_score_R, test_score_R,
                                train_score_F, test_score_F]],
                      columns = ['Model',
                                 'Train Accuracy', 'Test Accuracy',
```

```
                                    'Train Precision', 'Test Precision',
                                    'Train Recall', 'Test Recall',
                                    'Train F1', 'Test F1'])

comparison_df = comparison_df.append(knn_df, ignore_index = True)

comparison_df
```

Out[30]:

	Model	Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F
0	Logistic Regression	49.364735	47.645601	45.446265	44.246809	47.788177	45.685218	43.385447	41.73947
1	DTC	100.000000	43.742255	100.000000	42.944249	100.000000	42.814016	100.000000	42.86935
2	RFC	100.000000	49.938042	100.000000	48.371756	100.000000	48.720002	100.000000	48.49960
3	KNN	63.634955	46.716233	63.855739	46.446785	63.288812	45.927231	63.346458	46.03373

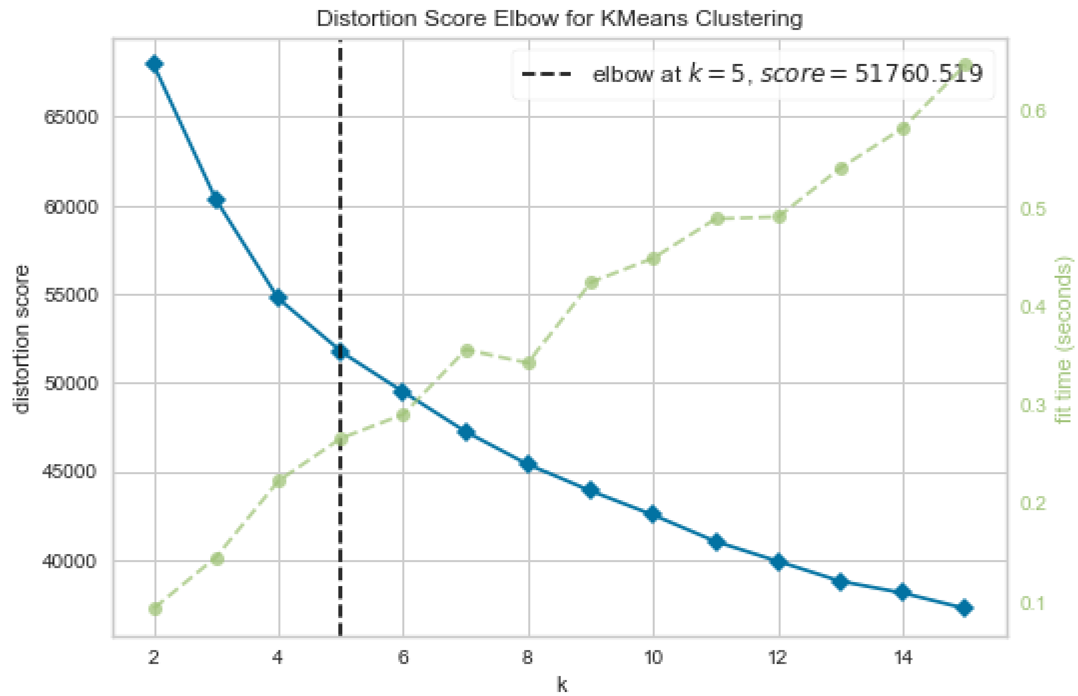
CREATE MODELS: UNSUPERVISED

In [31]:

```
# Model 5: K Means
from yellowbrick.cluster import KElbowVisualizer
from sklearn.cluster import KMeans

# visualization of distortion score of KMeans clustering
def elbow(df, k = 7):
    model = KElbowVisualizer(KMeans(), k = k)
    model.fit(df)
    model.show()

elbow(df_model, k = 15)
```



In [32]:
```
pca = PCA().fit(df_model)

first_comps = pd.DataFrame(zip(pca.components_[0],
                               df_model.columns),
                               columns = ['weights', 'features'])
first_comps['abs_weights'] = np.abs(first_comps['weights'])
first_comps.sort_values('abs_weights', ascending = False)
```

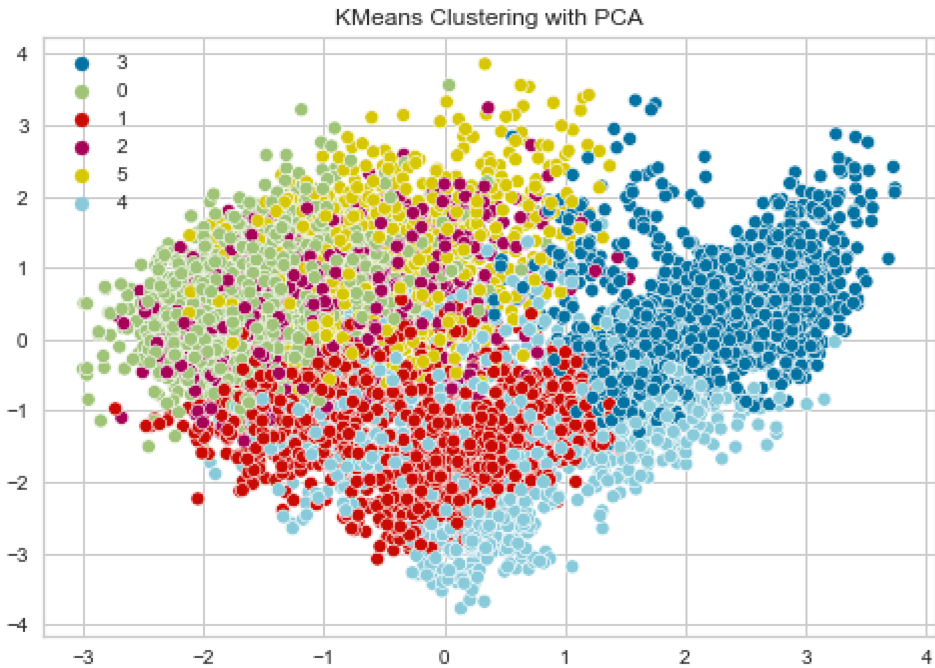
Out[32]:

	weights	features	abs_weights
2	-0.507497	Ever_Married	0.507497
3	-0.451493	Age	0.451493
9	0.408109	Segmentation	0.408109
7	0.402441	Spending_Score	0.402441
4	-0.319818	Graduated	0.319818
5	0.238715	Profession	0.238715
8	0.202591	Family_Size	0.202591
6	0.089321	Work_Experience	0.089321
1	-0.040323	Gender	0.040323
0	-0.013180	ID	0.013180

In [33]:
```
# KMeans with PCA
cust_pca = PCA(n_components = 0.99).fit_transform(df_model)
```

```
kmeans = KMeans(n_clusters = 6).fit(cust_pca)
hue = [str(i) for i in kmeans.labels_]
print('Silhouette Score: ', silhouette_score(cust_pca, kmeans.labels_))
sns.scatterplot(x = cust_pca[:, 0],
                y = cust_pca[:, 1],
                hue = hue)
plt.title('KMeans Clustering with PCA')
plt.show()
```

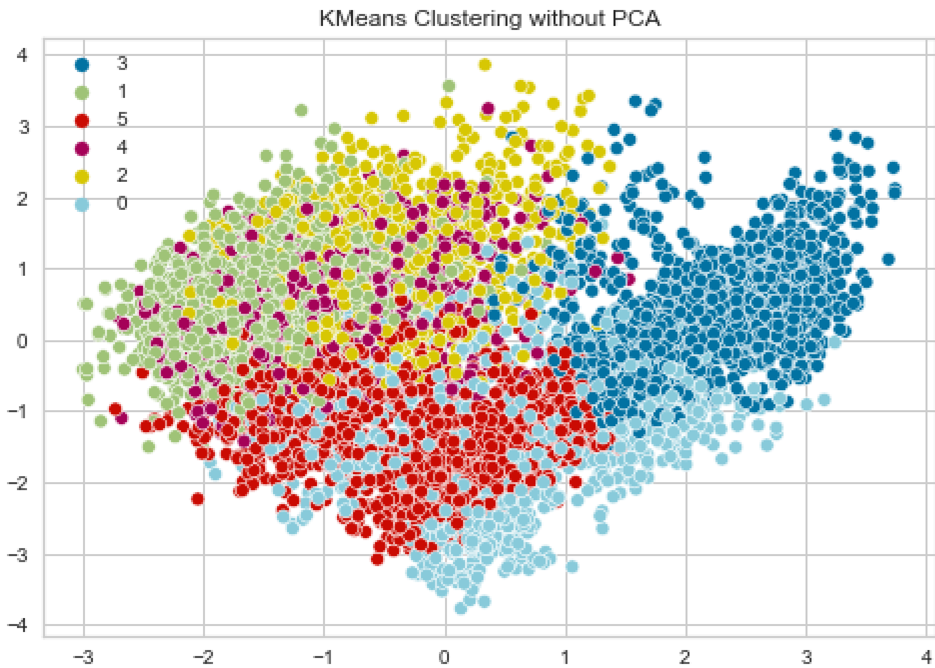
```
Silhouette Score:  0.16961948606350602
```



In [34]:

```
# KMeans without PCA - 6 clusters
kmeans = KMeans(n_clusters = 6).fit(df_model)
hue = [str(i) for i in kmeans.labels_]
print('Silhouette Score: ', silhouette_score(cust_pca, kmeans.labels_))
sns.scatterplot(x = cust_pca[:, 0],
                y = cust_pca[:, 1],
                hue = hue)
plt.title('KMeans Clustering without PCA')
plt.show()
```

```
Silhouette Score:  0.16966198513052666
```

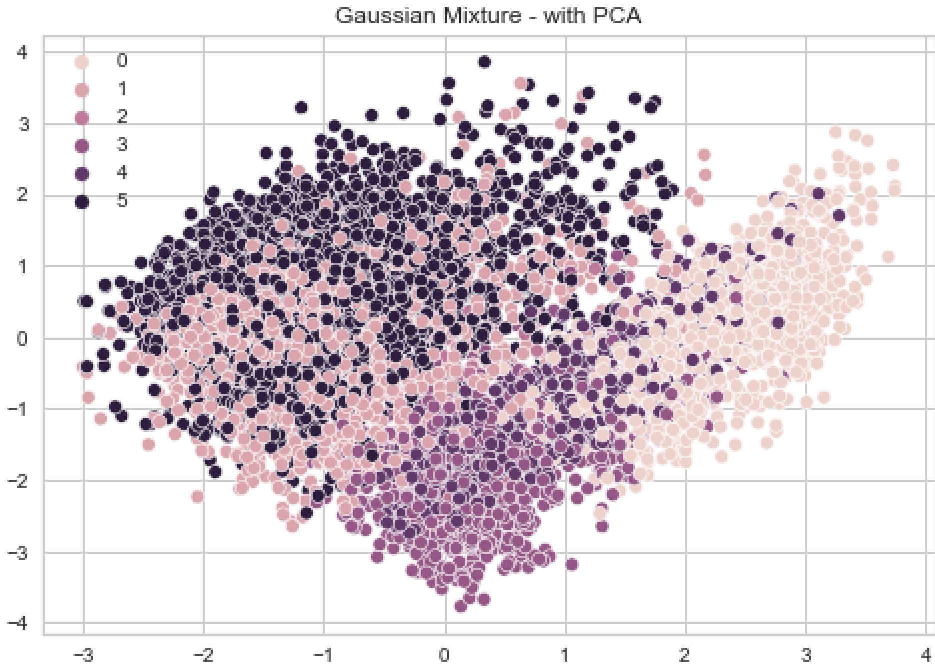


In [42]:

```
# Model 6: Gaussian Mixture
from sklearn.mixture import GaussianMixture

# with PCA
gm = GaussianMixture(6)
gm.fit(cust_pca)
labels = gm.predict(cust_pca)
print('Silhouette Score: ', silhouette_score(cust_pca, labels))
sns.scatterplot(x = cust_pca[:, 0],
                y = cust_pca[:, 1],
                hue = labels)
plt.title('Gaussian Mixture - with PCA')
plt.show()
```

```
Silhouette Score:  0.10277672057113534
```



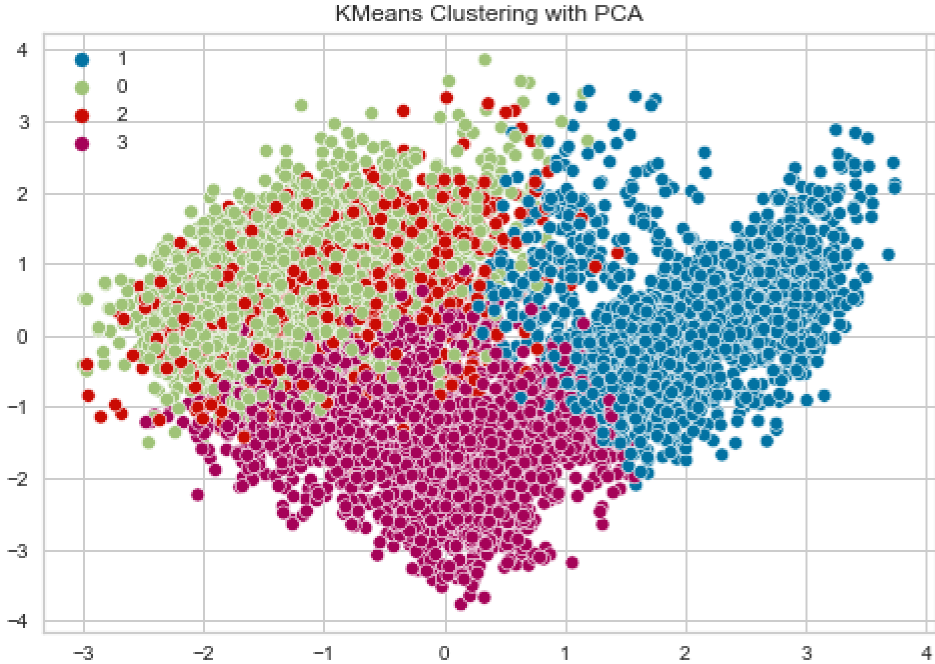
PLAYING WITH DIFFERENT CLUSTER NUM

In [36]:

```
# KMeans with PCA - 4 clusters
cust_pca = PCA(n_components = 0.99).fit_transform(df_model)


kmeans = KMeans(n_clusters = 4).fit(cust_pca)
hue = [str(i) for i in kmeans.labels_]
print('Silhouette Score: ', silhouette_score(cust_pca, kmeans.labels_))
sns.scatterplot(x = cust_pca[:, 0],
                y = cust_pca[:, 1],
                hue = hue)
plt.title('KMeans Clustering with PCA')
plt.show()
```

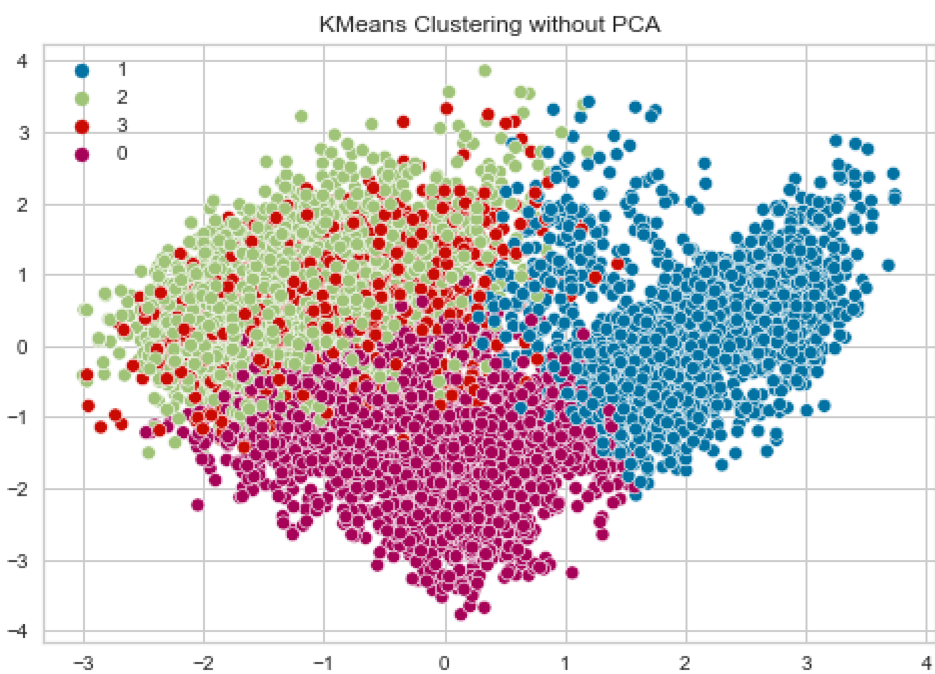
```
Silhouette Score:  0.18340509725458004
```



In [37]:

```
# KMeans without PCA - 4 clusters
kmeans = KMeans(n_clusters = 4).fit(df_model)
hue = [str(i) for i in kmeans.labels_]
print('Silhouette Score: ', silhouette_score(cust_pca, kmeans.labels_))
sns.scatterplot(x = cust_pca[:, 0],
                y = cust_pca[:, 1],
                hue = hue)
plt.title('KMeans Clustering without PCA')
plt.show()
```

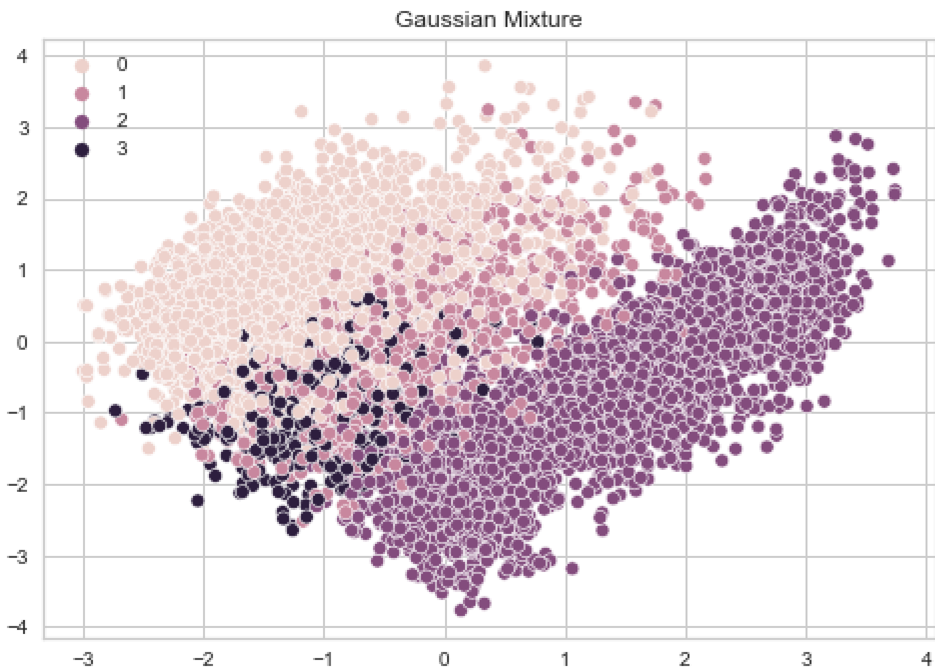
```
Silhouette Score:  0.18340509725458004
```



In [38]:

```python
# Gaussian with PCA - 4 clusters
gm = GaussianMixture(4)
gm.fit(cust_pca)
labels = gm.predict(cust_pca)
print('Silhouette Score: ', silhouette_score(cust_pca, labels))
sns.scatterplot(x = cust_pca[:, 0],
                y = cust_pca[:, 1],
                hue = labels)
plt.title('Gaussian Mixture')
plt.show()
```

```
Silhouette Score:  0.08339868648222354
```

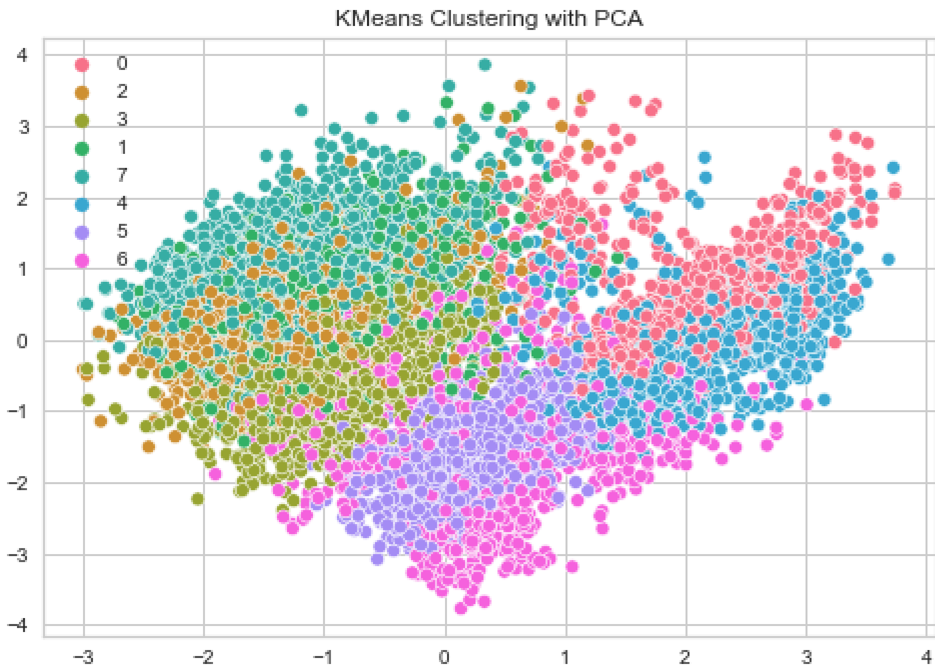


In [39]:

```python
# KMeans with PCA - 8 clusters
cust_pca = PCA(n_components = 0.99).fit_transform(df_model)


kmeans = KMeans(n_clusters = 8).fit(cust_pca)
hue = [str(i) for i in kmeans.labels_]
print('Silhouette Score: ', silhouette_score(cust_pca, kmeans.labels_))
sns.scatterplot(x = cust_pca[:, 0],
                y = cust_pca[:, 1],
                hue = hue)
plt.title('KMeans Clustering with PCA')
plt.show()
```

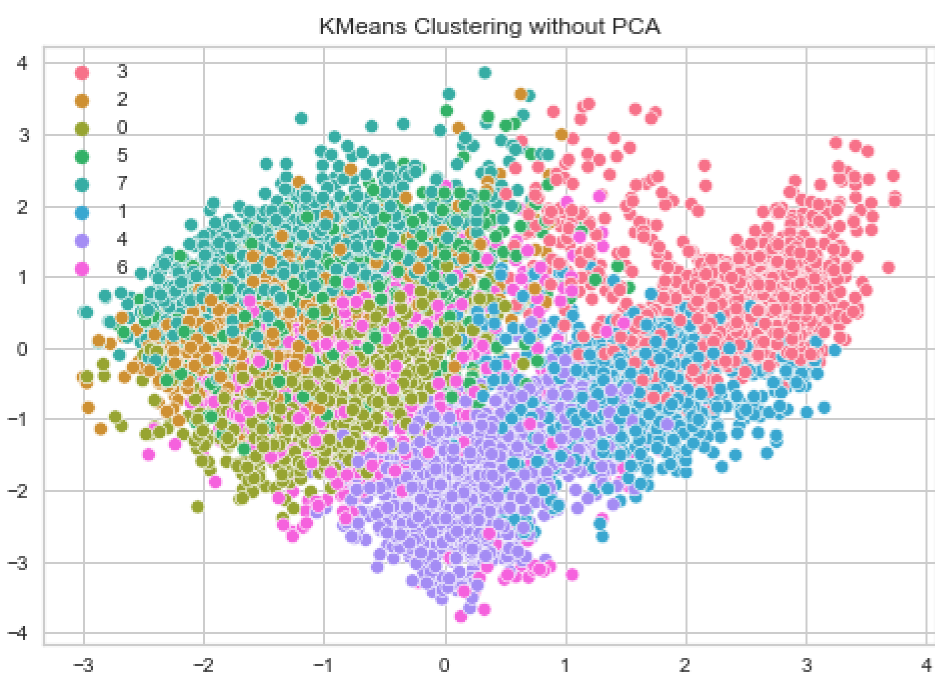
```
Silhouette Score:  0.14660991853090502
```



In [40]:

```
# KMeans without PCA - 8 clusters
kmeans = KMeans(n_clusters = 8).fit(df_model)
hue = [str(i) for i in kmeans.labels_]
print('Silhouette Score: ', silhouette_score(cust_pca, kmeans.labels_))
sns.scatterplot(x = cust_pca[:, 0],
                y = cust_pca[:, 1],
                hue = hue)
plt.title('KMeans Clustering without PCA')
plt.show()
```

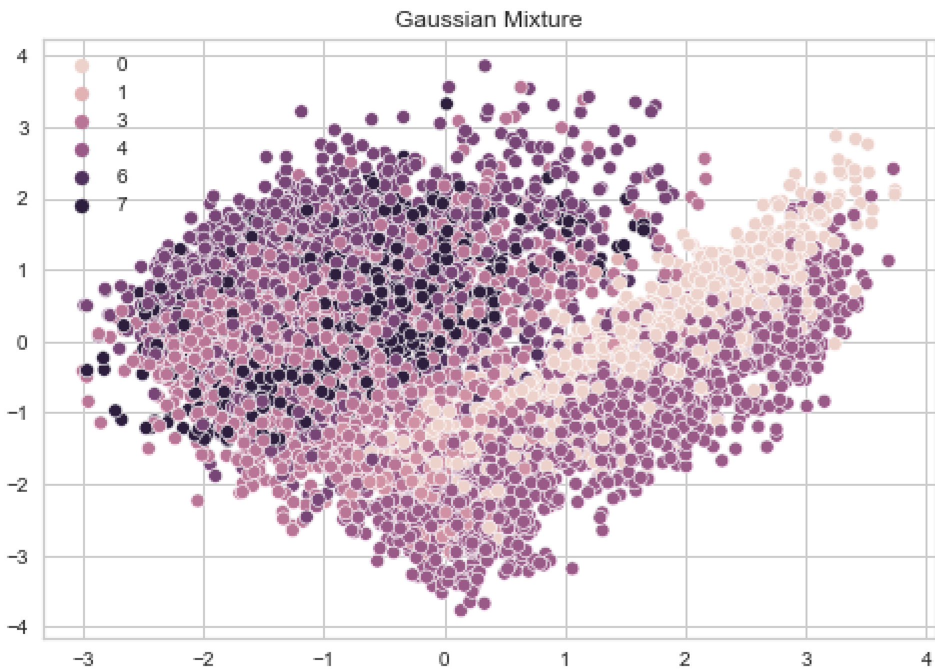
```
Silhouette Score:  0.14735199205417393
```



In [41]:

```
# Gaussian with PCA - 8 clusters
gm = GaussianMixture(8)
gm.fit(cust_pca)
labels = gm.predict(cust_pca)
print('Silhouette Score: ', silhouette_score(cust_pca, labels))
sns.scatterplot(x = cust_pca[:, 0],
                y = cust_pca[:, 1],
                hue = labels)
plt.title('Gaussian Mixture')
plt.show()
```

```
Silhouette Score:  0.08007860655428567
```



In [ ]:

```

```