K-Means Clustering Algorithm

September 12, 2023

0.1 K-Means Clustering Algorithm: Credit Card Customer Segmentation

In this project, I will play the role of a data scientist working for a credit card company. I have been given a dataset containing information about the company's clients and asked to help segment them into different groups in order to apply different business strategies for each type of customer.

The company expects to receive a group for each client and also an explanation of the characteristics of each group and what are the main points that make them different.

In a planning meeting with the Data Science coordinator, it was decided that I should use the K-means algorithm to segment the data.

In order to use the algorithm properly and achieve all the goals that the company has set for me, I will go through the following steps:

- Analyze the dataset;
- Prepare the data for modeling;
- Find an appropriate number of clusters;
- Segment the data;
- Interpret and explain the results.

I will start by importing the packages that I will use.

[38]:
```
pip install kneed
```

```
Requirement already satisfied: kneed in /opt/conda/lib/python3.10/site-packages
(0.8.5)
Requirement already satisfied: scipy>=1.0.0 in /opt/conda/lib/python3.10/site-
packages (from kneed) (1.9.1)
Requirement already satisfied: numpy>=1.14.2 in /opt/conda/lib/python3.10/site-
packages (from kneed) (1.21.6)
Note: you may need to restart the kernel to use updated packages.
```

[39]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from kneed import KneeLocator
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
```

1 Exploring the Data

[40]:
```
df = pd.read_csv('customer_segmentation.csv')
```

[41]:
```
df.head()
```

[41]:
```
   customer_id  age gender  dependent_count education_level marital_status  \
0    768805383   45      M                3     High School        Married
1    818770008   49      F                5        Graduate         Single
2    713982108   51      M                3        Graduate        Married
3    769911858   40      F                4     High School        Unknown
4    709106358   40      M                3      Uneducated        Married

   estimated_income  months_on_book  total_relationship_count  \
0             69000              39                         5
1             24000              44                         6
2             93000              36                         4
3             37000              34                         3
4             65000              21                         5

   months_inactive_12_mon  credit_limit  total_trans_amount  \
0                       1       12691.0                1144
1                       1        8256.0                1291
2                       1        3418.0                1887
3                       4        3313.0                1171
4                       1        4716.0                 816

   total_trans_count  avg_utilization_ratio
0                 42                  0.061
1                 33                  0.105
2                 20                  0.000
3                 20                  0.760
4                 28                  0.000
```

[42]:
```
df.describe()
```

[42]:
```
        customer_id           age  dependent_count  estimated_income  \
count  1.012700e+04  10127.000000     10127.000000      10127.000000
mean   7.391776e+08     46.325960         2.346203      62078.206774
std    3.690378e+07      8.016814         1.298908      39372.861291
min    7.080821e+08     26.000000         0.000000      20000.000000
25%    7.130368e+08     41.000000         1.000000      32000.000000
50%    7.179264e+08     46.000000         2.000000      50000.000000
75%    7.731435e+08     52.000000         3.000000      80000.000000
max    8.283431e+08     73.000000         5.000000     200000.000000

       months_on_book  total_relationship_count  months_inactive_12_mon  \
```

```
count    10127.000000              10127.000000             10127.000000
mean        35.928409                  3.812580                 2.341167
std          7.986416                  1.554408                 1.010622
min         13.000000                  1.000000                 0.000000
25%         31.000000                  3.000000                 2.000000
50%         36.000000                  4.000000                 2.000000
75%         40.000000                  5.000000                 3.000000
max         56.000000                  6.000000                 6.000000

       credit_limit  total_trans_amount  total_trans_count  \
count  10127.000000        10127.000000       10127.000000
mean    8631.953698         4404.086304          64.858695
std     9088.776650         3397.129254          23.472570
min     1438.300000          510.000000          10.000000
25%     2555.000000         2155.500000          45.000000
50%     4549.000000         3899.000000          67.000000
75%    11067.500000         4741.000000          81.000000
max    34516.000000        18484.000000         139.000000

       avg_utilization_ratio
count           10127.000000
mean                0.274894
std                 0.275691
min                 0.000000
25%                 0.023000
50%                 0.176000
75%                 0.503000
max                 0.999000
```

Looking at the summary statisitics here, the data makes sense for the variables is covers.

[43]:
```
df.isnull().sum()
```

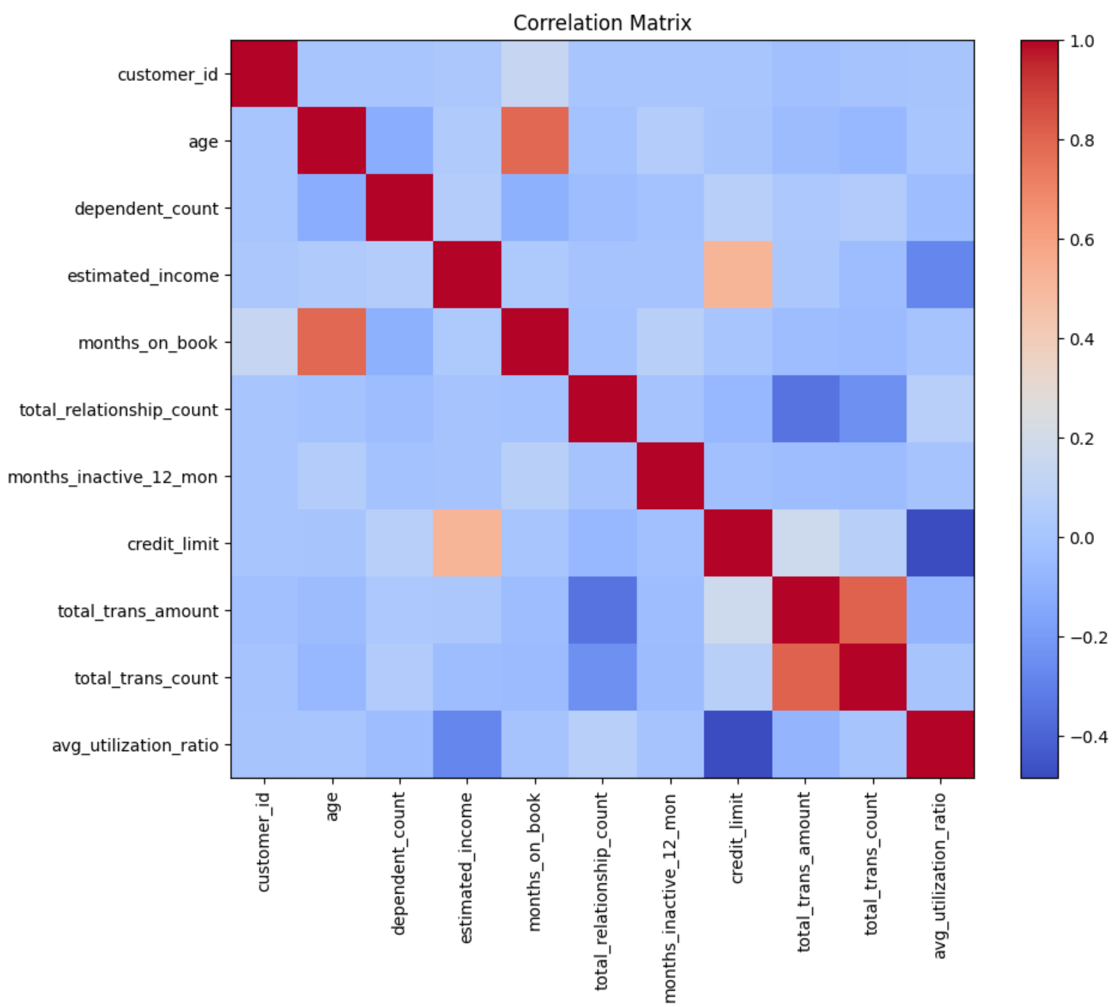
[43]:
```
customer_id                 0
age                         0
gender                      0
dependent_count             0
education_level             0
marital_status              0
estimated_income            0
months_on_book              0
total_relationship_count    0
months_inactive_12_mon      0
credit_limit                0
total_trans_amount          0
total_trans_count           0
avg_utilization_ratio       0
```

```
dtype: int64
```

There aren't any missing values in the data.

[44]:
```python
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='none', aspect='auto')
plt.colorbar()
plt.xticks(range(len(correlation_matrix)), correlation_matrix.columns, rotation=90)
plt.yticks(range(len(correlation_matrix)), correlation_matrix.columns)
plt.title('Correlation Matrix')
```

[44]:
```
Text(0.5, 1.0, 'Correlation Matrix')
```

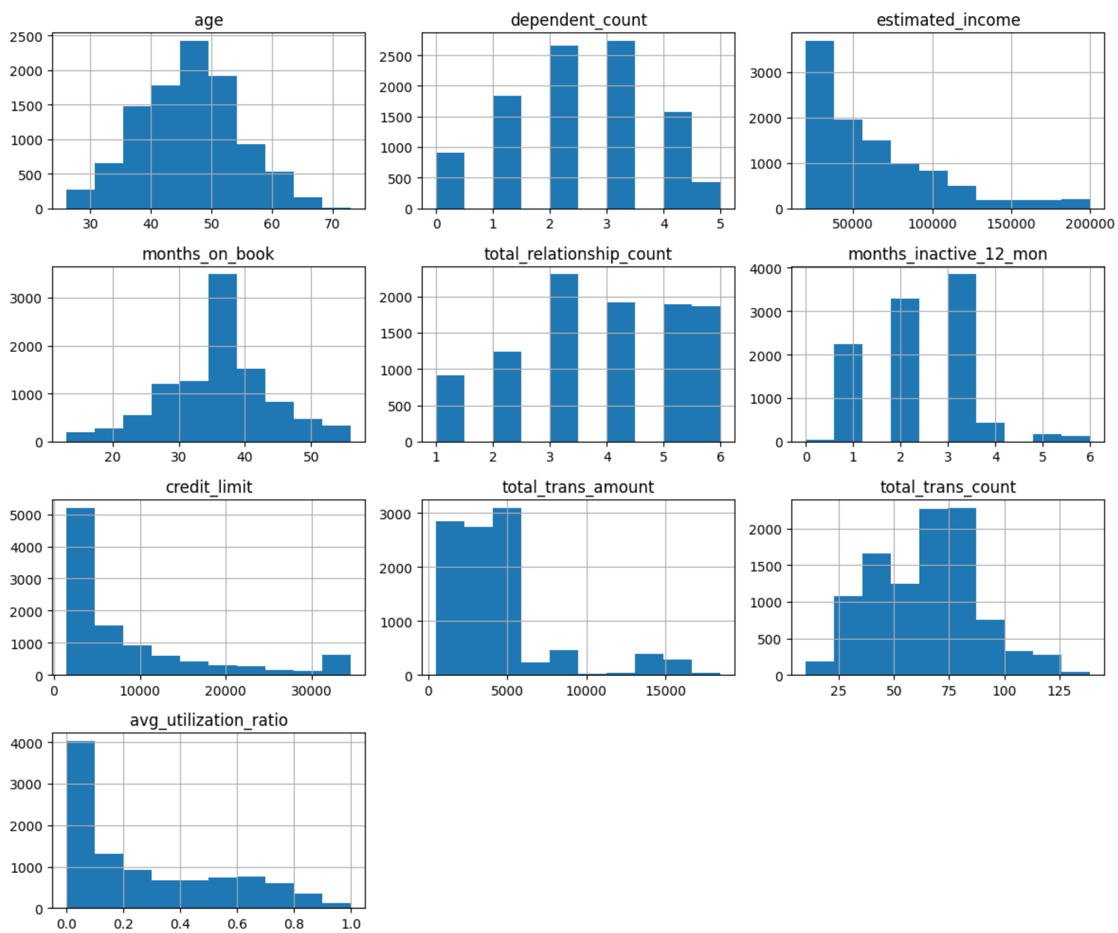


Most of the variables present weak correlations between each other, but there are some I can highlight:

- Age is strongly correlated with how long the person has been a customer (months_on_book);
- Credit limit is positively correlated with the estimated income and negatively correlated with the average utilization ratio;
- The total number of transactions (total_trans_count) is strongly correlated with the total amount transitioned (total_trans_amount).

[45]:
```
fig, ax = plt.subplots(figsize=(12, 10))

#Removing the customer's id before plotting the distributions
df.drop('customer_id', axis=1).hist(ax=ax)

plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_64/579634717.py:4: UserWarning: To output multiple subplots, the
figure containing the passed axes is being cleared
  df.drop('customer_id', axis=1).hist(ax=ax)
```



The bar plots here show that most distributions in the dataframe are skewed.

1.1 Preprocessing: Feature Scaling using Z-Score Normalization

Now, I will scale all numerical columns using the `StandardScaler` class so they all have a mean of 0 and standard deviation of 1.

[46]:
```
numerical_features = ['age', 'dependent_count', 'months_on_book',␣
↪'total_relationship_count',
                      'months_inactive_12_mon', 'credit_limit',␣
↪'total_trans_amount',
                      'total_trans_count', 'avg_utilization_ratio']

# Extract the numerical data
numerical_data = df[numerical_features]

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the data
scaled_data = scaler.fit_transform(numerical_data)

# Create a DataFrame with the scaled data and columns
scaled_df = pd.DataFrame(data=scaled_data, columns=numerical_features)
```

I can see that all numerical features are now properly scaled. Now, I will add the gender column to `scaled_df` with the following mapping: 1 for M and 0 for F

[47]:
```
scaled_df['gender'] = df['gender'].apply(lambda x: 1 if x == 'M' else 0)
```

[48]:
```
scaled_df.head()
```

[48]:
```
        age  dependent_count  months_on_book  total_relationship_count  \
0 -0.165406         0.503368        0.384621                  0.763943
1  0.333570         2.043199        1.010715                  1.407306
2  0.583058         0.503368        0.008965                  0.120579
3 -0.789126         1.273283       -0.241473                 -0.522785
4 -0.789126         0.503368       -1.869317                  0.763943

   months_inactive_12_mon  credit_limit  total_trans_amount  \
0               -1.327136      0.446622           -0.959707
1               -1.327136     -0.041367           -0.916433
2               -1.327136     -0.573698           -0.740982
3                1.641478     -0.585251           -0.951758
4               -1.327136     -0.430877           -1.056263

   total_trans_count  avg_utilization_ratio  gender
0          -0.973895              -0.775882       1
```

```
1          -1.357340             -0.616276       0
2          -1.911206             -0.997155       1
3          -1.911206              1.759686       0
4          -1.570365             -0.997155       1
```

Finally, I will map the `education_level` column in the `scaled_df` column in the following way:

- Uneducated - 0
- High School - 1
- College - 2
- Graduate - 3
- Post-Graduate - 4
- Doctorate - 5

[49]:
```
education_mapping = {
    'Uneducated': 0,
    'High School': 1,
    'College': 2,
    'Graduate': 3,
    'Post-Graduate': 4,
    'Doctorate': 5
}

scaled_df['education_level'] = df['education_level'].map(education_mapping)
```

[50]:
```
scaled_df.head()
```

[50]:
```
        age  dependent_count  months_on_book  total_relationship_count  \
0 -0.165406         0.503368        0.384621                  0.763943
1  0.333570         2.043199        1.010715                  1.407306
2  0.583058         0.503368        0.008965                  0.120579
3 -0.789126         1.273283       -0.241473                 -0.522785
4 -0.789126         0.503368       -1.869317                  0.763943

   months_inactive_12_mon  credit_limit  total_trans_amount  \
0               -1.327136      0.446622           -0.959707
1               -1.327136     -0.041367           -0.916433
2               -1.327136     -0.573698           -0.740982
3                1.641478     -0.585251           -0.951758
4               -1.327136     -0.430877           -1.056263

   total_trans_count  avg_utilization_ratio  gender  education_level
0          -0.973895              -0.775882       1                1
1          -1.357340              -0.616276       0                3
2          -1.911206              -0.997155       1                3
3          -1.911206               1.759686       0                1
4          -1.570365              -0.997155       1                0
```

Since the `marital_status` column in the orignal dataframe can't be separted in order of magnitude, I will use one-hot encoding on it.

[51]:
```
marital_status_encoded = pd.get_dummies(df['marital_status'],␣
↪prefix='marital_status')

scaled_df = pd.concat([scaled_df, marital_status_encoded], axis=1)
```

[52]:
```
scaled_df.head()
```

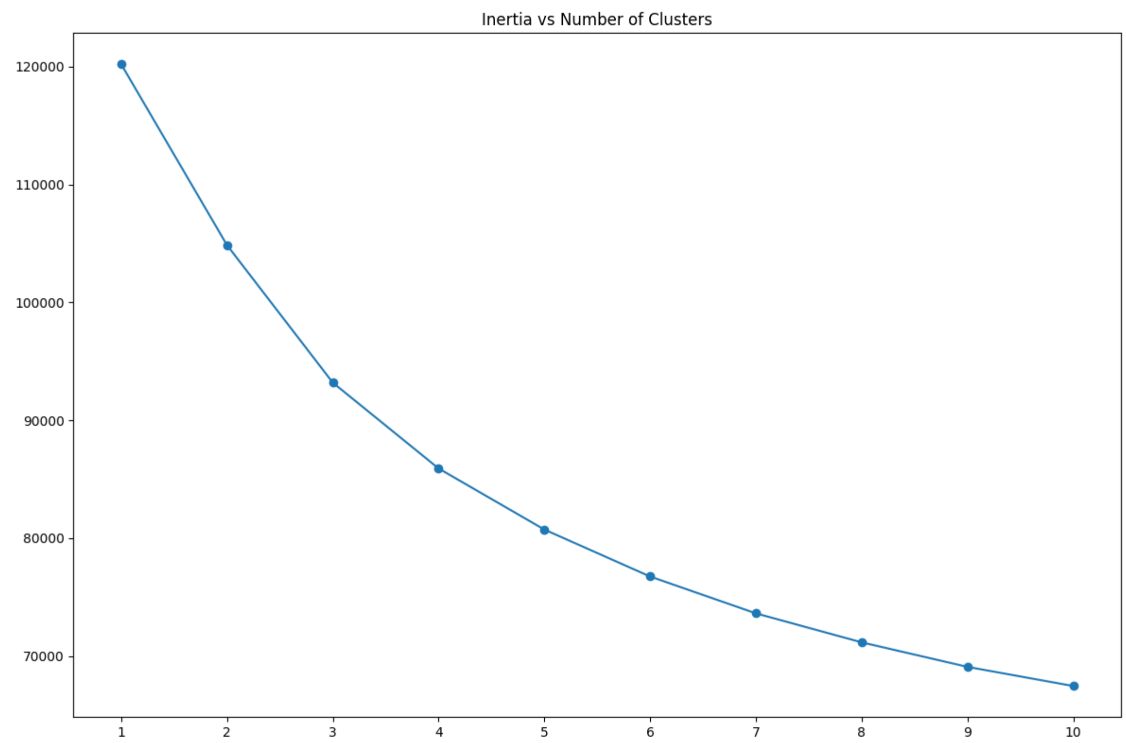
[52]:
```
        age  dependent_count  months_on_book  total_relationship_count  \
0 -0.165406         0.503368        0.384621                  0.763943
1  0.333570         2.043199        1.010715                  1.407306
2  0.583058         0.503368        0.008965                  0.120579
3 -0.789126         1.273283       -0.241473                 -0.522785
4 -0.789126         0.503368       -1.869317                  0.763943

   months_inactive_12_mon  credit_limit  total_trans_amount  \
0               -1.327136      0.446622           -0.959707
1               -1.327136     -0.041367           -0.916433
2               -1.327136     -0.573698           -0.740982
3                1.641478     -0.585251           -0.951758
4               -1.327136     -0.430877           -1.056263

   total_trans_count  avg_utilization_ratio  gender  education_level  \
0          -0.973895              -0.775882       1                1
1          -1.357340              -0.616276       0                3
2          -1.911206              -0.997155       1                3
3          -1.911206               1.759686       0                1
4          -1.570365              -0.997155       1                0

   marital_status_Divorced  marital_status_Married  marital_status_Single  \
0                        0                       1                      0
1                        0                       0                      1
2                        0                       1                      0
3                        0                       0                      0
4                        0                       1                      0

   marital_status_Unknown
0                       0
1                       0
2                       0
3                       1
4                       0
```

1.2 Choosing Number of Clusters (K) with Elbow Method

Now, I will choose the optimal K by looking at an Elbow Curve.

[53]:

```
def plot_elbow_curve(df, max_clusters=10):
    inertias = []

    for k in range(1, max_clusters+1):
        model = KMeans(n_clusters=k)
        cluster = model.fit_predict(df)
        inertias.append(model.inertia_)

    plt.figure(figsize=(12, 8))
    plt.plot(range(1, max_clusters+1), inertias, marker='o')
    plt.xticks(ticks=range(1, max_clusters+1), labels=range(1, max_clusters+1))
    plt.title('Inertia vs Number of Clusters')

    plt.tight_layout()
    plt.show()

    return inertias

inertias = plot_elbow_curve(scaled_df)
print(inertias)
```




```
[120209.61913696052, 104859.42237496469, 93190.27610220725, 85908.23031097917,
80722.76277888907, 76736.2446877103, 73610.56882778769, 71150.0175642037,
69071.21117731852, 67445.39805939124]
```

Looking at the graph, I will use the K value of 6.

1.3 Running K-Means Algorithm with K=6

[54]:
```
model = KMeans(n_clusters=6)
clusters = model.fit_predict(scaled_df)

clusters_series = pd.Series(clusters)
df['Cluster'] = clusters_series + 1

print(df['Cluster'].value_counts())
```

```
6    2807
2    2381
4    1550
3    1288
5    1151
1     950
Name: Cluster, dtype: int64
```

After segmenting the data into 6 clusters, I can see that clusters 2, 4, and 3 have very similar number of occurences, while clusters 5, 1, and 6 have very similar number of occurences.

[55]:
```
model.inertia_
```

[55]:
```
76736.14053004631
```

[56]:
```
model.cluster_centers_
```

[56]:
```
array([[-0.13507309, -0.04043519, -0.10030536, -1.04627928, -0.15323135,
         0.62181075,  2.62704152,  1.79027379, -0.37989373,  0.61263158,
         2.01473684,  0.07684211,  0.44631579,  0.4       ,  0.07684211],
       [-0.01623693,  0.31347726, -0.01097579,  0.06678521,  0.01248237,
        -0.37535248, -0.20687004, -0.06862797,  0.17467158,  0.37184874,
         0.49579832,  0.07016807,  0.47058824,  0.37394958,  0.08529412],
       [-1.41090997, -0.66733314, -1.35727058,  0.33070788, -0.16947624,
        -0.34490732, -0.39578113, -0.32751519,  0.14885999,  0.4670287 ,
         2.15593483,  0.07680372,  0.44142746,  0.42668735,  0.05508146],
       [ 1.34858407, -0.9748692 ,  1.25024624,  0.18989628,  0.17249301,
        -0.35086044, -0.41509292, -0.4509529 ,  0.1655649 ,  0.42645161,
         2.27806452,  0.04129032,  0.51096774,  0.40451613,  0.04322581],
       [ 0.02767642,  0.23088507,  0.02595071,  0.08813115, -0.01578292,
         2.07295704, -0.25227897, -0.27502463, -0.84198649,  0.90695652,
         2.00608696,  0.09391304,  0.40695652,  0.41130435,  0.08782609],
       [-0.04861357,  0.4978844 , -0.03446964,  0.0046453 ,  0.03030727,
        -0.38919612, -0.19930984, -0.03561365,  0.1655837 ,  0.35470085,
         3.27029915,  0.08440171,  0.46794872,  0.36431624,  0.08333333]])
```

[57]:
```
model.n_iter_
```

[57]:
```
29
```

When running the K-Means algorithm with K=4, I am able to produce the following results:

- Lowest SSE Value:85910.45
- The coordinates of the centroids
- Number of iterations required to converge: 7

[58]:
```
model.labels_[:5]
```

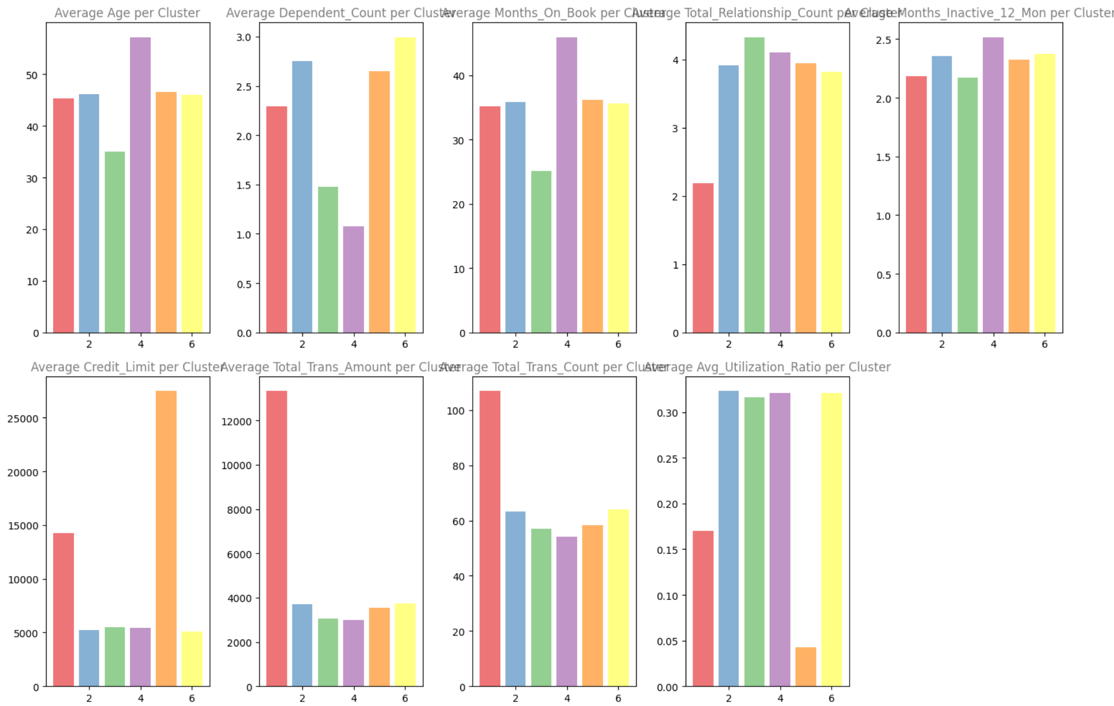
[58]:
```
array([1, 5, 5, 1, 1], dtype=int32)
```

1.4 Interpreting Results - Exploring Clusters for Numerical Variables

Now, I will interpret the results and summarize the characteristics of each cluster and differentiate them from each other based on the variables used for the segmentation.

Now, I will analyze the numerical variables and see how they behave in each cluster.

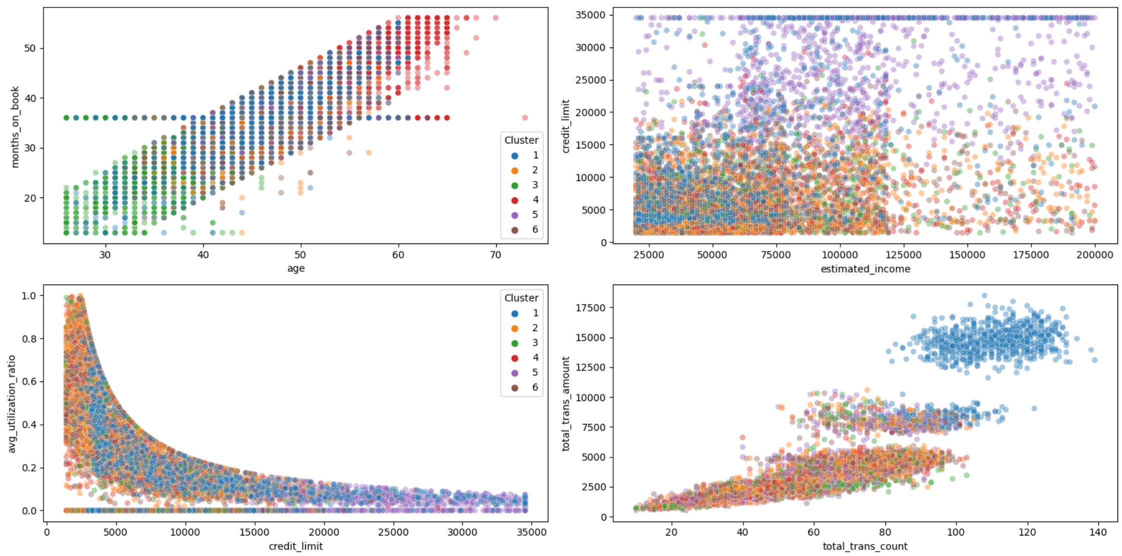
[59]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Define the number of rows and columns in the subplot grid
num_rows = 2
num_cols = 5  # Increase the number of columns to 4 to accommodate 7 subplots

# Create a new figure
fig = plt.figure(figsize=(15, 10))

for i, column in enumerate(numerical_features):
    df_plot = df.groupby('Cluster')[column].mean()

    # Calculate the row and column index for the subplot
    row_index = i // num_cols
    col_index = i % num_cols

    # Add the subplot
    ax = fig.add_subplot(num_rows, num_cols, i + 1)

    # Plot the data in the subplot
    ax.bar(df_plot.index, df_plot, color=sns.color_palette('Set1'), alpha=0.6)
    ax.set_title(f'Average {column.title()} per Cluster', alpha=0.5)

plt.tight_layout()
plt.show()
```



From looping the original dataframe, I found the following information:

- Cluster 4, on average, has the oldest clients while Cluster 3 has the youngest. These two clusters also have the lowest number of dependedents, on average.
- Months on Book is the least for Cluster 3, on average which is expected.
- Cluster 1, on average, has the lowest relationship count but the highest transaction amounts.
- Cluster 5, on average, has the highest credit limit but also the lowest credit utilization rate.

[61]:
```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16, 8))
sns.scatterplot(x='age', y='months_on_book', hue='Cluster', data=df, palette='tab10', alpha=0.4, ax=ax1)
sns.scatterplot(x='estimated_income', y='credit_limit', hue='Cluster', data=df, palette='tab10', alpha=0.4, ax=ax2, legend=False)
sns.scatterplot(x='credit_limit', y='avg_utilization_ratio', hue='Cluster', data=df, palette='tab10', alpha=0.4, ax=ax3)
sns.scatterplot(x='total_trans_count', y='total_trans_amount', hue='Cluster', data=df, palette='tab10', alpha=0.4, ax=ax4, legend=False)

plt.tight_layout()
plt.show()
```



From these scatterplots, I can draw these conclusions:

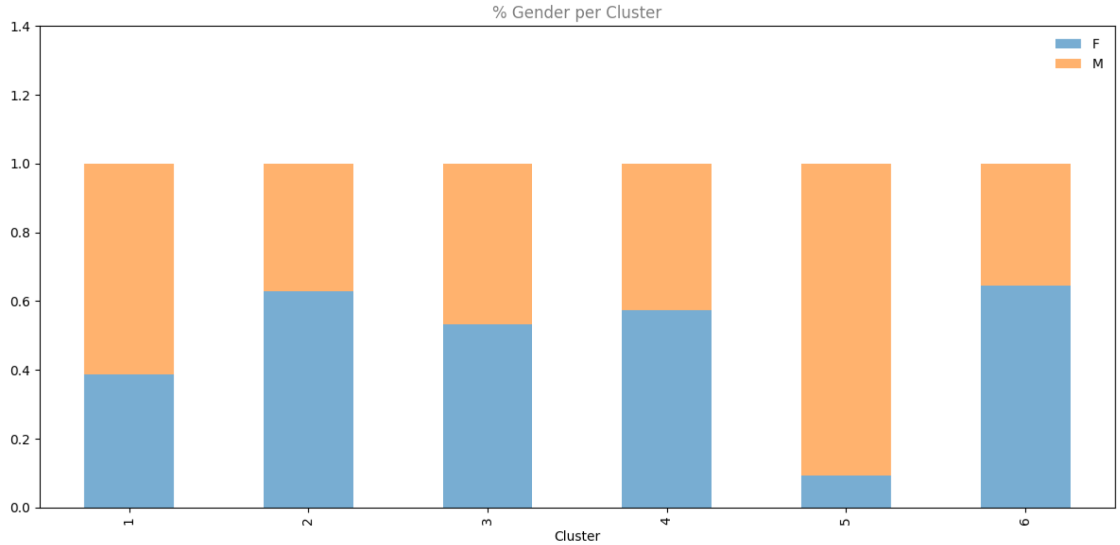
- Cluster 1 has the highest amount of money transitioned.
- Cluster 2 has the lowest credit limit and estimated income and the highest utilization rate.
- Cluster 5 has the highest credit limit.
- Older clients are grouped in Cluster 5.

1.5 Interpreting Results - Exploring Clusters for Categorical Variables

Now, I want understand how the categorical column `Gender` impacts the cluster split.

Here are the questions I want to answer: - Is the cluster with customers from only one gender? - Are any clusters equally divided between men and women?

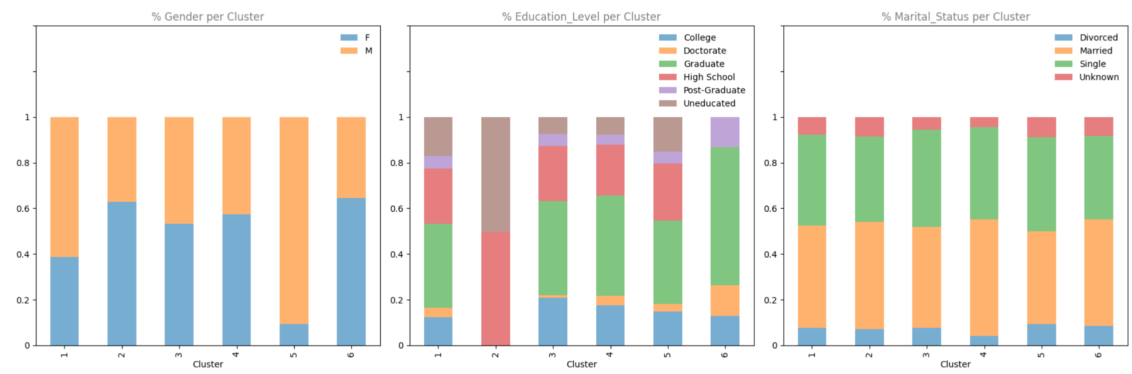
[65]:
```
plot_df = pd.crosstab(
  index=df['Cluster'], columns=df['gender'],
  values=df['gender'], aggfunc='size', normalize='index'
)



fig, ax = plt.subplots(figsize=(12,6))
plot_df.plot.bar(stacked=True, ax=ax, alpha=0.6)
ax.set_title(f'% Gender per Cluster', alpha=0.5)

ax.set_ylim(0, 1.4)
ax.legend(frameon=False)
ax.xaxis.grid(False)

plt.tight_layout()
plt.show()
```



For the categorical columns, I will plot the percentual distribution of each variable in each cluster.

[66]:
```
cat_columns = df.select_dtypes(include=['object'])

fig = plt.figure(figsize=(18, 6))
for i, col in enumerate(cat_columns):
    plot_df = pd.crosstab(index=df['Cluster'], columns=df[col], values=df[col],␣
 ↪aggfunc='size', normalize='index')
    ax = fig.add_subplot(1, 3, i+1)
    plot_df.plot.bar(stacked=True, ax=ax, alpha=0.6)
    ax.set_title(f'% {col.title()} per Cluster', alpha=0.5)

    ax.set_ylim(0, 1.4)
    ax.legend(frameon=False)
    ax.xaxis.grid(False)

    labels = [0, 0.2, 0.4, 0.6, 0.8, 1]
    ax.set_yticklabels(labels)

plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_64/2465878031.py:15: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels(labels)
```



Considering the categorical variables, I notice:

- Cluster 5 is largely male, while cluster 6 is largely female.
- Cluster 2 is largely composed of high school graduates and those who are uneducated.
- Cluster has mostly graduates.
- Marital Status is evenly split across the clusters.

Now, I will evaluate the clustering performance of the model I ran above because the Elbow Method does not evaluate clustering performance using ground truth labels.

[ ]:
```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
from sklearn.metrics import adjusted_rand_score
```

[ ]:
```
scaled_df, true_labels = make_moons(
    n_samples=250, noise=0.05, random_state=42
)
scaled_features = scaler.fit_transform(scaled_df)
```

[ ]:
```
# Instantiate k-means and dbscan algorithms
kmeans = KMeans(n_clusters=4)
dbscan = DBSCAN(eps=0.3)

# Fit the algorithms to the features
kmeans.fit(scaled_features)
dbscan.fit(scaled_features)

# Compute the silhouette scores for each algorithm
kmeans_silhouette = silhouette_score(
    scaled_features, kmeans.labels_
).round(2)
dbscan_silhouette = silhouette_score(
   scaled_features, dbscan.labels_
).round (2)
```

[ ]:
```
print(kmeans_silhouette)
print(dbscan_silhouette)
```

The silhouette coefficient is higher for the K-Means algorithm. The DBSCAN algorithm appears to find more natural clusters according to the shape of the data.

The ARI metric uses true cluster asssignments to measure the similarity between true and predicted labels. I will use this method to evaluate the clustering performance.

[ ]:
```
ari_kmeans = adjusted_rand_score(true_labels, kmeans.labels_)
ari_dbscan = adjusted_rand_score(true_labels, dbscan.labels_)

round(ari_kmeans, 2)
```

[ ]:
```
round(ari_dbscan, 2)
```

ARI shows that DBSCAN is the best choice compared to the K-Means because an ARI of 1 indicates perfeclty labeled clusters.

1.6 Conclusion

As demanded by the company, I now have listed the most important characteristics of each cluster. Below are some suggestions and insights into each one of them.

Cluster 1:

- Relationship Count: Lowest on average.
- Transaction Amounts: Highest on average.
- Suggestion: Cluster 1 seems to consist of clients with a lower number of relationships but higher transaction amounts. Consider tailoring marketing strategies to these clients to maximize transaction revenue.

Cluster 2:

- Credit Limit: Lowest on average.
- Estimated Income: Lowest on average.
- Credit Utilization Rate: Highest on average.
- Suggestion: Cluster 2 appears to have clients with lower credit limits, lower income, and higher credit utilization rates. Offer financial education or credit management services to help clients in this cluster improve their financial health.

Cluster 3:

- Average Age: Youngest on average.
- Average Number of Dependents: Lowest on average.
- Months on Book: Least on average.
- Suggestion: Cluster 3 consists of younger clients who have been with the bank for a shorter time. Focus on building long-term relationships with these clients to maximize their value over time.

Cluster 4:

- Average Age: Oldest on average.

- Average Number of Dependents: Lowest on average.
- Suggestion: Cluster 4 has the oldest clients with the fewest dependents. Consider offering retirement planning or wealth management services to cater to the needs of this demographic.

Cluster 5:

- Credit Limit: Highest on average.
- Credit Utilization Rate: Lowest on average.
- Suggestion: Cluster 5 appears to consist of clients with high credit limits and low credit utilization rates. Encourage these clients to make use of their credit lines responsibly and offer premium financial products.

Cluster 6:

- Gender: Largely female.
- Suggestion: Cluster 6 is predominantly female. Consider tailoring marketing campaigns or services to the specific needs and preferences of female clients.

Education and Marital Status:

- Cluster 2: Largely composed of high school graduates and those who are uneducated.
- Cluster 4: Mostly graduates.
- Marital Status is evenly split across the clusters.

Additional Insights:

- Cluster 5 has the highest credit limit and consists of older clients. These clients might be financially stable and could be targeted for premium services.
- Cluster 2 has lower credit limits and higher credit utilization rates. Consider offering credit counseling or credit limit increase options.
- Marital status doesn't appear to vary significantly across clusters, so marital status may not be a strong indicator of behavior or needs in this context.

It's important to conduct further analysis, possibly using statistical techniques or machine learning models, to identify patterns and trends more precisely and to validate these insights. Additionally, consider conducting surveys or interviews with clients in each cluster to gather more specific information about their financial needs and preferences.

[ ]:
```

```