Importing the required libraries

```
In [1]: import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        import warnings
        warnings.filterwarnings('ignore')
        from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
        from sklearn.metrics import silhouette_score
        from scipy.stats import probplot
        from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN, SpectralC
        from sklearn.neighbors import NearestNeighbors
        from kneed import KneeLocator
        from clusteval import clusteval
        from sklearn.decomposition import PCA
        from datetime import date
```

Defining the customization settings

```
In [2]: plt.rcParams['figure.figsize'] = (12,8)
        pd.set_option('display.float_format',lambda x: '%.2f' % x)
```

Loading the dataset

```
In [ ]: df = pd.read_csv(r'C:\Users\anand\Downloads\back\Bank-Customer-Segmentation-Be
        df.head()
```

Out[ ]:

	TransactionID	CustomerID	CustomerDOB	CustGender	CustLocation	CustAc
0	T1	C5841053	10/1/94	F	JAMSHEDPUR	
1	T2	C2142763	4/4/57	M	JHAJJAR	
2	T3	C4417068	26/11/96	F	MUMBAI	
3	T4	C5342380	14/9/73	F	MUMBAI	
4	T5	C9031234	24/3/88	F	NAVI MUMBAI	

Examining the content of the dataset

```
In [4]: df.shape
```

```
Out[4]: (1048567, 9)
```

```
In [5]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048567 entries, 0 to 1048566
Data columns (total 9 columns):
 #   Column                   Non-Null Count    Dtype  
---  ------                   --------------    -----  
 0   TransactionID            1048567 non-null  object 
 1   CustomerID               1048567 non-null  object 
 2   CustomerDOB              1045170 non-null  object 
 3   CustGender               1047467 non-null  object 
 4   CustLocation             1048416 non-null  object 
 5   CustAccountBalance       1046198 non-null  float64
 6   TransactionDate          1048567 non-null  object 
 7   TransactionTime          1048567 non-null  int64  
 8   TransactionAmount (INR)  1048567 non-null  float64
dtypes: float64(2), int64(1), object(6)
memory usage: 72.0+ MB
```

In [6]:
```
df.describe()
```

Out[6]:

	CustAccountBalance	TransactionTime	TransactionAmount (INR)
count	1046198.00	1048567.00	1048567.00
mean	115403.54	157087.53	1574.34
std	846485.38	51261.85	6574.74
min	0.00	0.00	0.00
25%	4721.76	124030.00	161.00
50%	16792.18	164226.00	459.03
75%	57657.36	200010.00	1200.00
max	115035495.10	235959.00	1560034.99

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
TransactionID                 0
CustomerID                    0
CustomerDOB                3397
CustGender                 1100
CustLocation                151
CustAccountBalance         2369
TransactionDate               0
TransactionTime               0
TransactionAmount (INR)       0
dtype: int64
```

In [8]:
```
df[df.duplicated()]
```

Out[8]:

TransactionID	CustomerID	CustomerDOB	CustGender	CustLocation	CustAcc

Feature Engineering

In [9]:
```
df.drop(['TransactionID','CustomerID'],axis=1,inplace=True)
```

In [10]:
```
df['CustGender'].value_counts()
```

Out[10]:
```
M    765530
F    281936
T         1
Name: CustGender, dtype: int64
```

In [11]:
```
df.drop(df[df['CustGender'].isin(['T'])].index,axis=0,inplace=True)
```

In [12]:
```
df['CustGender'].unique()
```

Out[12]:
```
array(['F', 'M', nan], dtype=object)
```

In [13]:
```
df = df[~df['CustomerDOB'].isna()]
df.CustomerDOB = pd.to_datetime(df.CustomerDOB,errors='coerce')
```

In [14]:
```
def age(birthdate):
    today = date.today()
    age = today.year - birthdate.year - ((today.month, today.day) < (birthdate
    return age
```

In [15]:
```
df['Age'] = df.CustomerDOB.apply(age)
```

In [16]:
```
df = df[df['Age']>0]
df.drop('CustomerDOB',axis=1,inplace=True)
df.shape
```

Out[16]:
```
(970106, 7)
```

In [17]:
```
for col in df.columns:
    print("Percentage of null values of {}:".format(col),str(np.round(df[col].
```

```
Percentage of null values of CustGender: 0.09%
Percentage of null values of CustLocation: 0.02%
Percentage of null values of CustAccountBalance: 0.23%
Percentage of null values of TransactionDate: 0.0%
Percentage of null values of TransactionTime: 0.0%
Percentage of null values of TransactionAmount (INR): 0.0%
Percentage of null values of Age: 0.0%
```

In [18]:
```
for col in df.columns:
    print("Percentage of unique values of {}:".format(col),str(np.round(df[col
Percentage of unique values of CustGender: 0.0%
Percentage of unique values of CustLocation: 0.91%
Percentage of unique values of CustAccountBalance: 15.07%
Percentage of unique values of TransactionDate: 0.01%
Percentage of unique values of TransactionTime: 8.39%
Percentage of unique values of TransactionAmount (INR): 8.72%
Percentage of unique values of Age: 0.01%
```

In [19]:
```
df = df[~df['CustGender'].isna()]
```

In [20]:
```
df = df[~df['CustLocation'].isna()]
```

In [21]:
```
df.CustAccountBalance = df.CustAccountBalance.fillna(df.groupby('CustLocation'
```

In [22]:
```
df = df[~df.CustAccountBalance.isna()]
```

In [23]:
```
df.isna().sum()
```

Out[23]:
```
CustGender                 0
CustLocation               0
CustAccountBalance         0
TransactionDate            0
TransactionTime            0
TransactionAmount (INR)    0
Age                        0
dtype: int64
```

In [24]:
```
df.shape
```

Out[24]:
```
(968909, 7)
```

In [25]:
```
df.TransactionDate = pd.to_datetime(df.TransactionDate,errors='coerce')
```

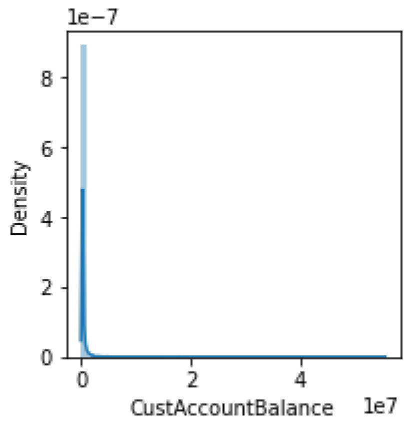
In [26]:
```
df['TransactionMonth'] = df.TransactionDate.dt.month
df['TransactionMonthName'] = df.TransactionDate.dt.month_name()
df['TransactionDay'] = df.TransactionDate.dt.day
df['TransactionDayName'] = df.TransactionDate.dt.day_name()
```

In [27]:
```
df.drop('TransactionDate',axis=1,inplace=True)
```

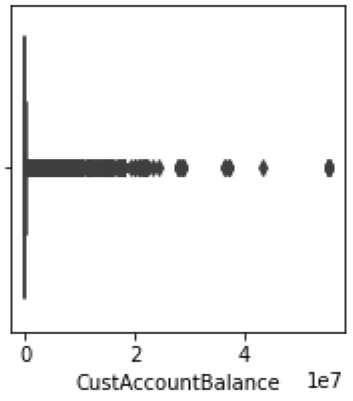
Exploratory Data Analysis(EDA)

In [28]:
```
for col in df.columns:
    if df[col].dtypes == np.float64:
        print("Skewness of {}:".format(col),df[col].skew())
        print("Kurtosis of {}:".format(col),df[col].kurt())
        plt.figure(figsize=(3,3))
        print("Distribution Plot of {}:".format(col))
        sns.distplot(df[col])
        plt.show()
        print("Box Plot of {}:".format(col))
        plt.figure(figsize=(3,3))
        sns.boxplot(df[col])
        plt.show()
        print("Quantile-Quantile Plot of {}:".format(col))
        plt.figure(figsize=(3,3))
        probplot(df[col],plot=plt,rvalue=True)
        plt.show()
```

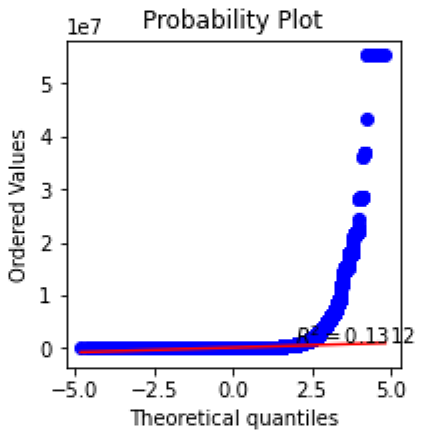
```
Skewness of CustAccountBalance: 36.61795538324552
Kurtosis of CustAccountBalance: 2650.4742083374363
Distribution Plot of CustAccountBalance:
```



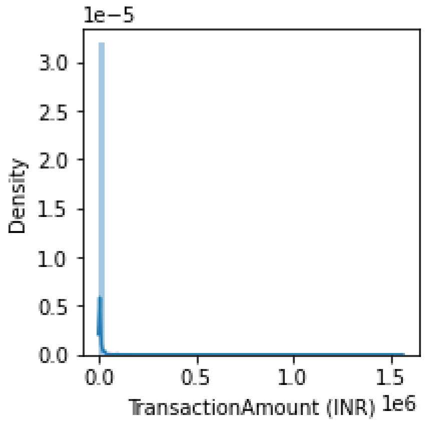
```
Box Plot of CustAccountBalance:
```



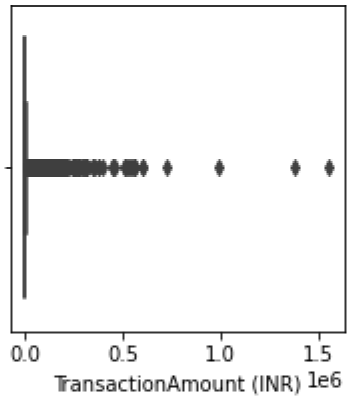
```
Quantile-Quantile Plot of CustAccountBalance:
```



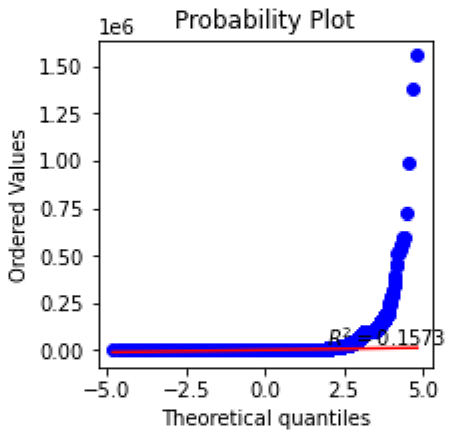
```
Skewness of TransactionAmount (INR): 54.35309885783315
Kurtosis of TransactionAmount (INR): 8863.538556781345
Distribution Plot of TransactionAmount (INR):
```



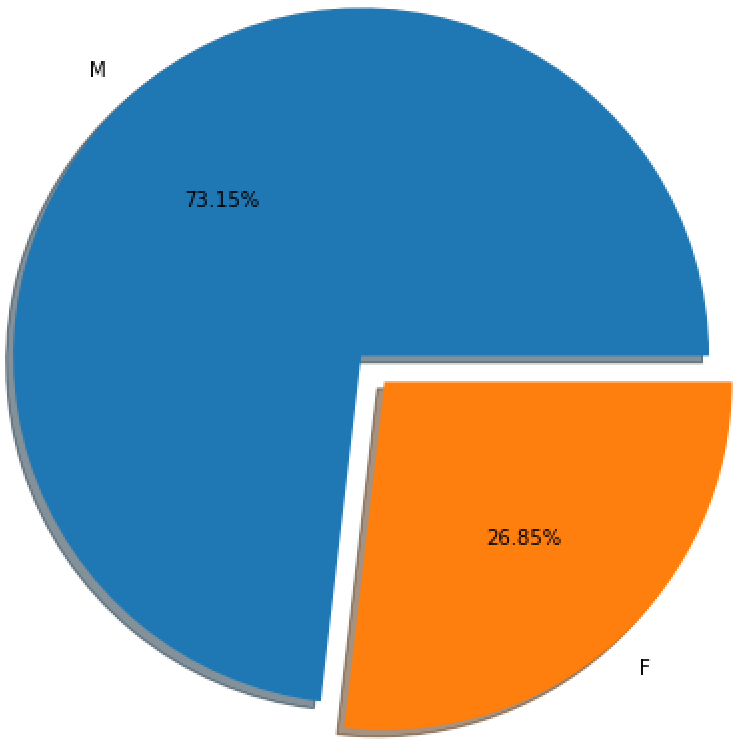
```
Box Plot of TransactionAmount (INR):
```



```
Quantile-Quantile Plot of TransactionAmount (INR):
```

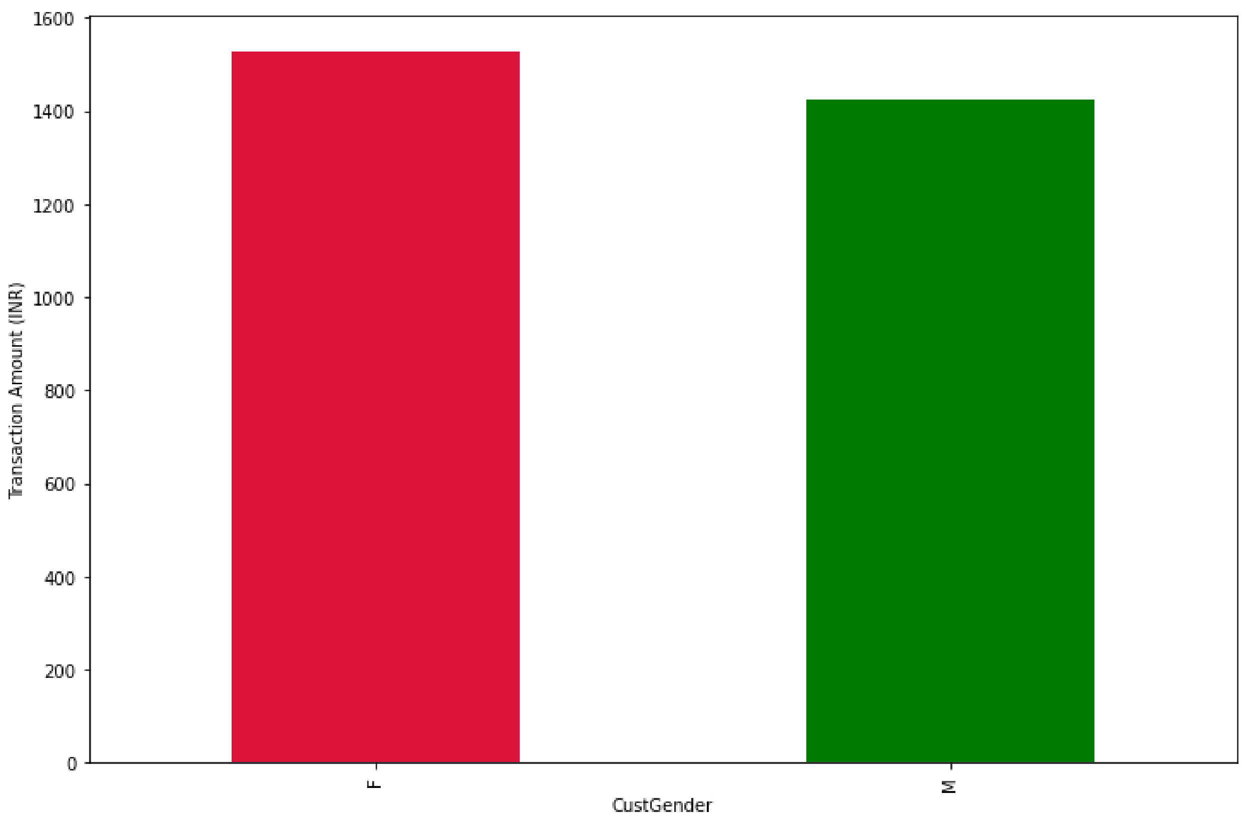


In [29]:
```python
labels = df.CustGender.value_counts().keys()
values = df.CustGender.value_counts().values
explode = (0.1,0)

plt.pie(values,labels=labels,explode=explode,shadow=True,autopct='%1.2f%%');
```



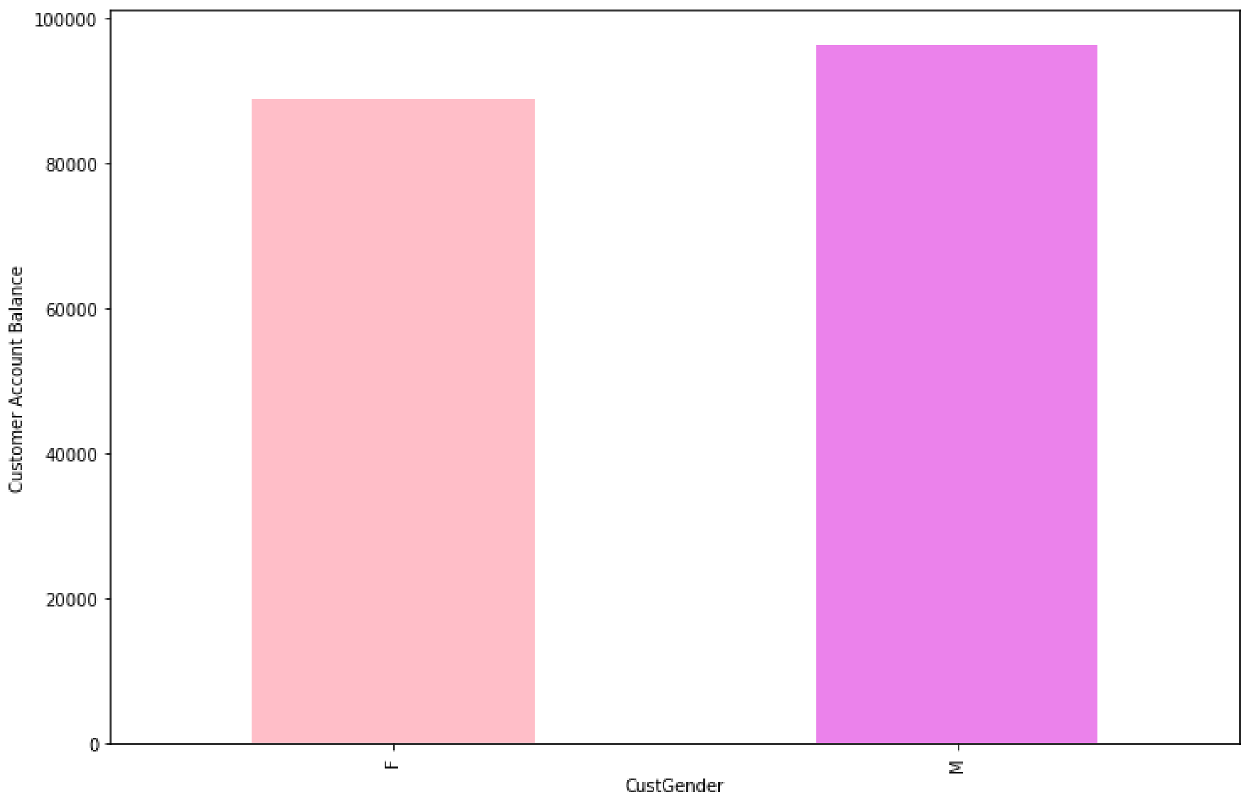
A significant proportion of bank customers are males accounting for about 73% of the total share.

In [30]:
```
ax = df.groupby('CustGender')['TransactionAmount (INR)'].mean().plot(kind='bar
plt.ylabel('Transaction Amount (INR)');
```



At an average, females make slightly higher transactions than males.

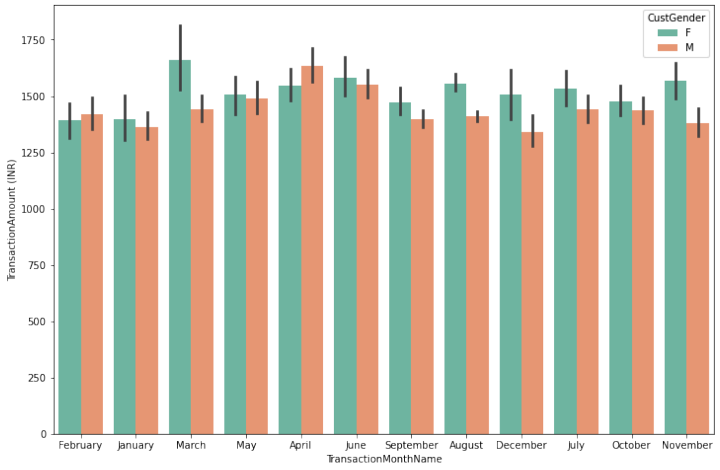
```
In [31]: ax = df.groupby('CustGender')['CustAccountBalance'].mean().plot(kind='bar',col
         plt.ylabel('Customer Account Balance');
```



At an average, male customers have marginally higher account balances in comparison to their female counterparts.

```
In [32]: sns.barplot(data=df,x='TransactionMonthName',y='TransactionAmount (INR)',palet
         plt.title('Monthly Comparison of Spending Habits of Male & Female Customers',f
```

Monthly Comparison of Spending Habits of Male & Female Customers

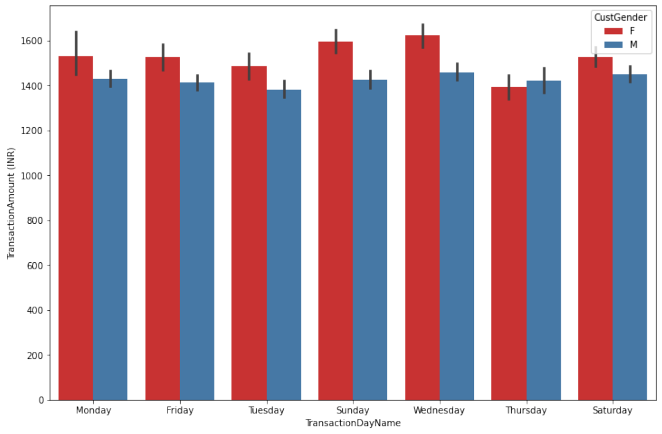


Male customers make higher transactions than their female counterparts only during the months of February and April. In all the remaining months, female customers perform greater transactions. Therefore, the bank must provide more special offers and incentives to female customers as they are likely to be substantially active in making larger transactions almost throughout the entire year.

Higher value transactions are mostly done during the months of March, April and June.

```
In [33]: sns.barplot(data=df,x='TransactionDayName',y='TransactionAmount (INR)',palette
         plt.title('Weekday-Wise Comparison of Spending Habits of Male & Female Custome
```

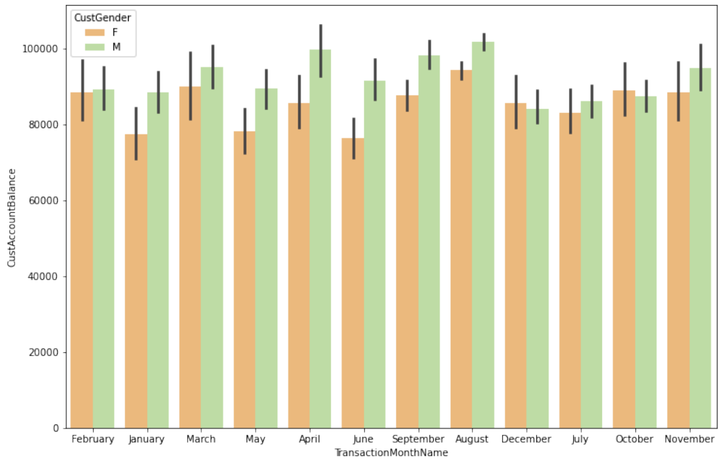
Weekday-Wise Comparison of Spending Habits of Male & Female Customers



Female customers perform higher transactions than their male counterparts in almost every week day.

```
In [34]: sns.barplot(data=df,x='TransactionMonthName',y='CustAccountBalance',palette='S
         plt.title('Monthly Comparison of Account Balances of Male & Female Customers',
```

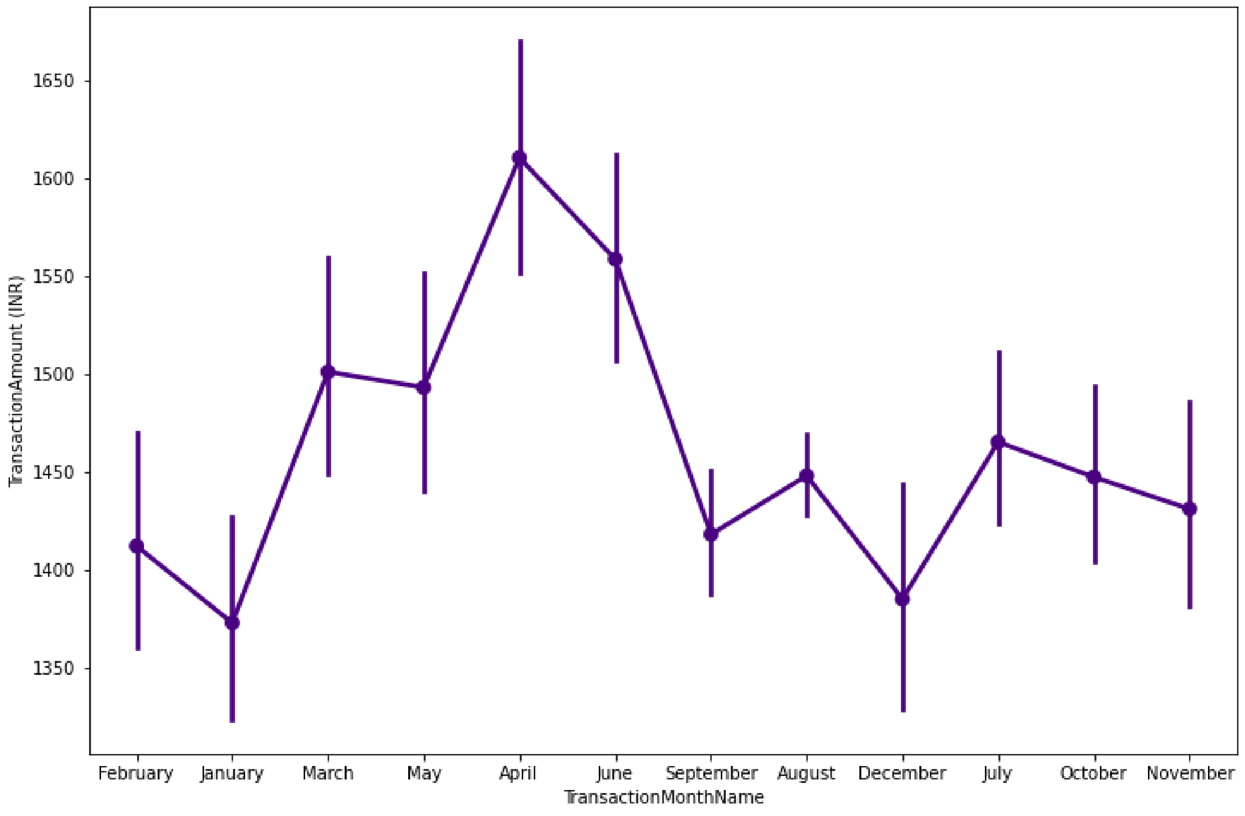
Monthly Comparison of Account Balances of Male & Female Customers



Male customers predominantly have greater account balances as compared to their female counterparts. This is evident from the fact that they generally make comparatively low value transactions than female customers.

```
In [35]: sns.pointplot(data=df,x='TransactionMonthName',y='TransactionAmount (INR)',col
```

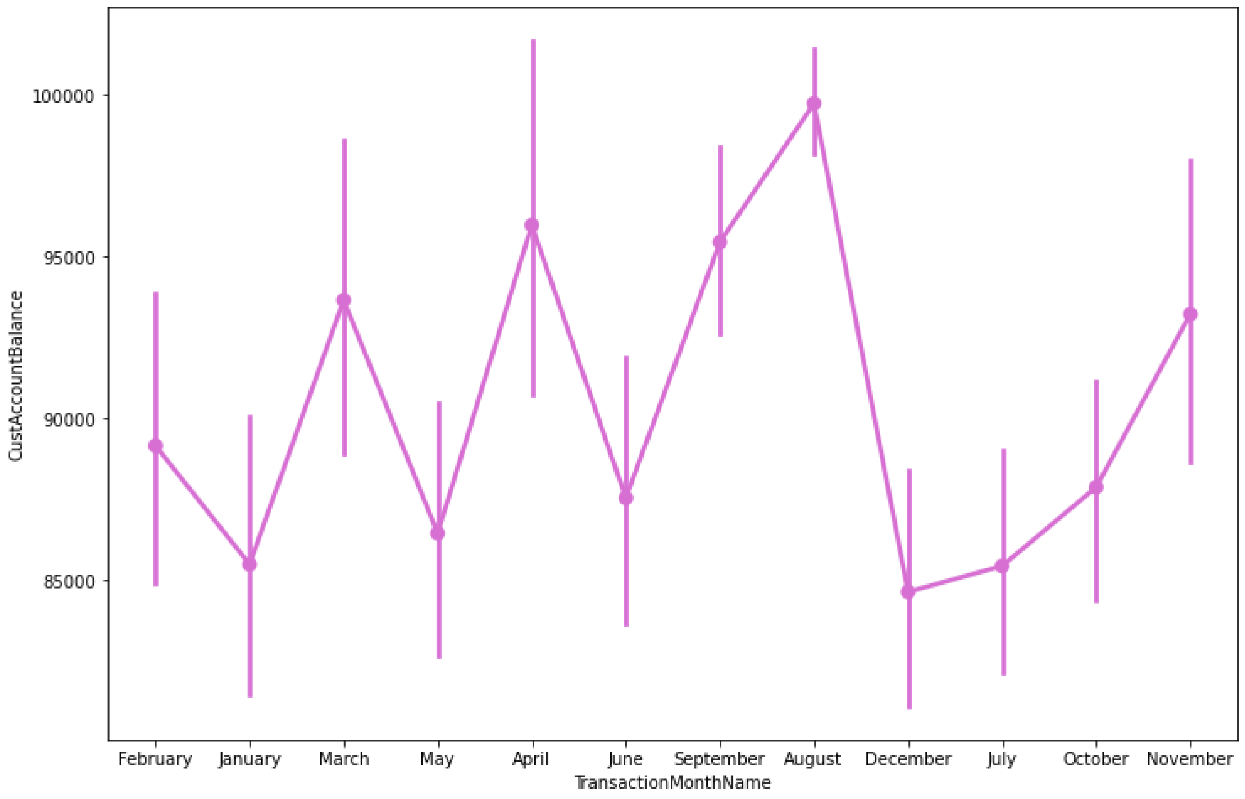
```
Out[35]: <AxesSubplot:xlabel='TransactionMonthName', ylabel='TransactionAmount (INR)'>
```



Highest value transactions are performed during the summer months while least value transactions are made in the winter months.

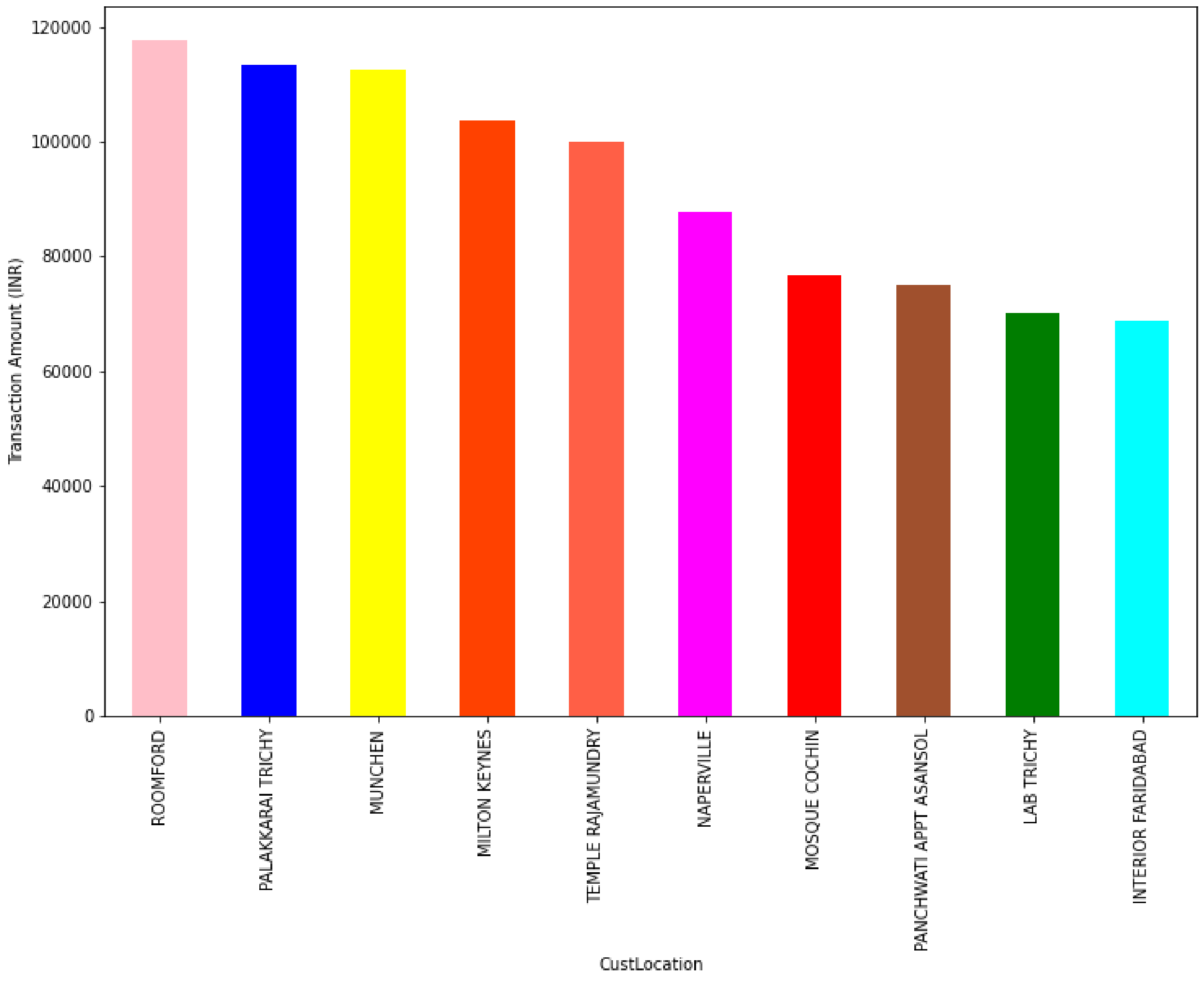
```
In [36]: sns.pointplot(data=df,x='TransactionMonthName',y='CustAccountBalance',color='c
```

```
Out[36]: <AxesSubplot:xlabel='TransactionMonthName', ylabel='CustAccountBalance'>
```



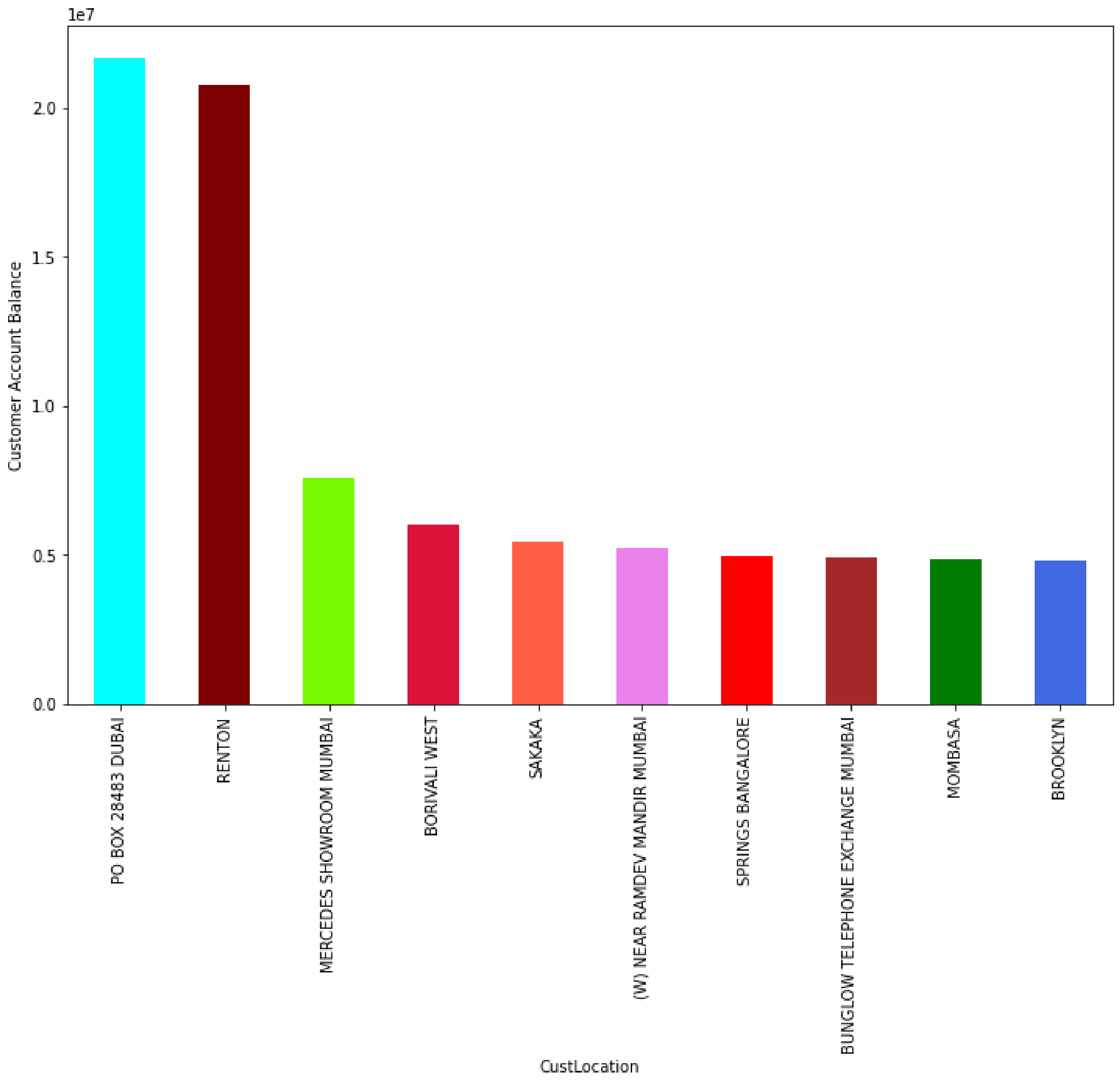
Commonly, the account balances of customers are highest in the months of August and September whereas they are lowest in the months of July and December. In the festive months, the customer account balances drop significantly which makes sense as people usually expend more money during the festive season.

In [37]:
```
df.groupby('CustLocation')['TransactionAmount (INR)'].mean().sort_values(ascer
plt.ylabel('Transaction Amount (INR)');
```



Customers belonging to the Roomford bank branch in United Kingdom mostly perform the highest transactions which are closely followed by the bank branches in Palakkarai Trichy(Tamil Nadu, India) and Munchen in Germany.

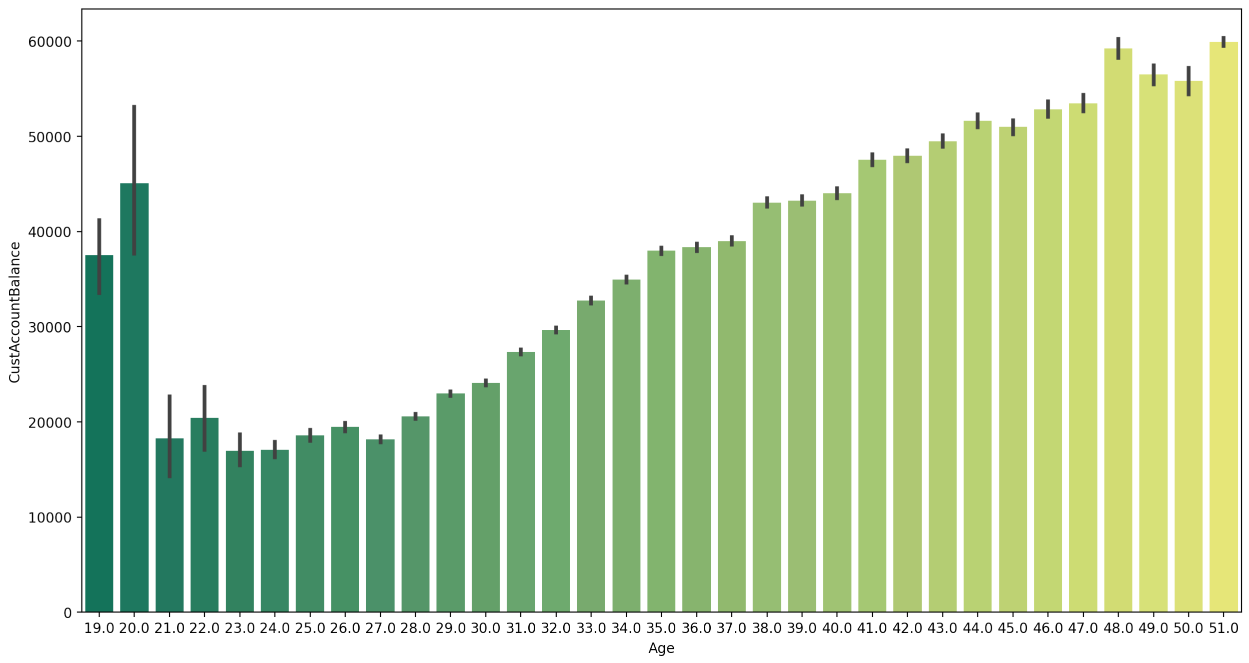
```
In [38]: df.groupby('CustLocation')['CustAccountBalance'].mean().sort_values(ascending=
         plt.ylabel('Customer Account Balance');
```



Customers living in PO Box 28483 Dubai bank branch have the highest account balances among all customers.

```
In [156…  plt.figure(figsize=(15,8),dpi=200)
          sns.barplot(data=df,x='Age',y='CustAccountBalance',palette='summer')
```

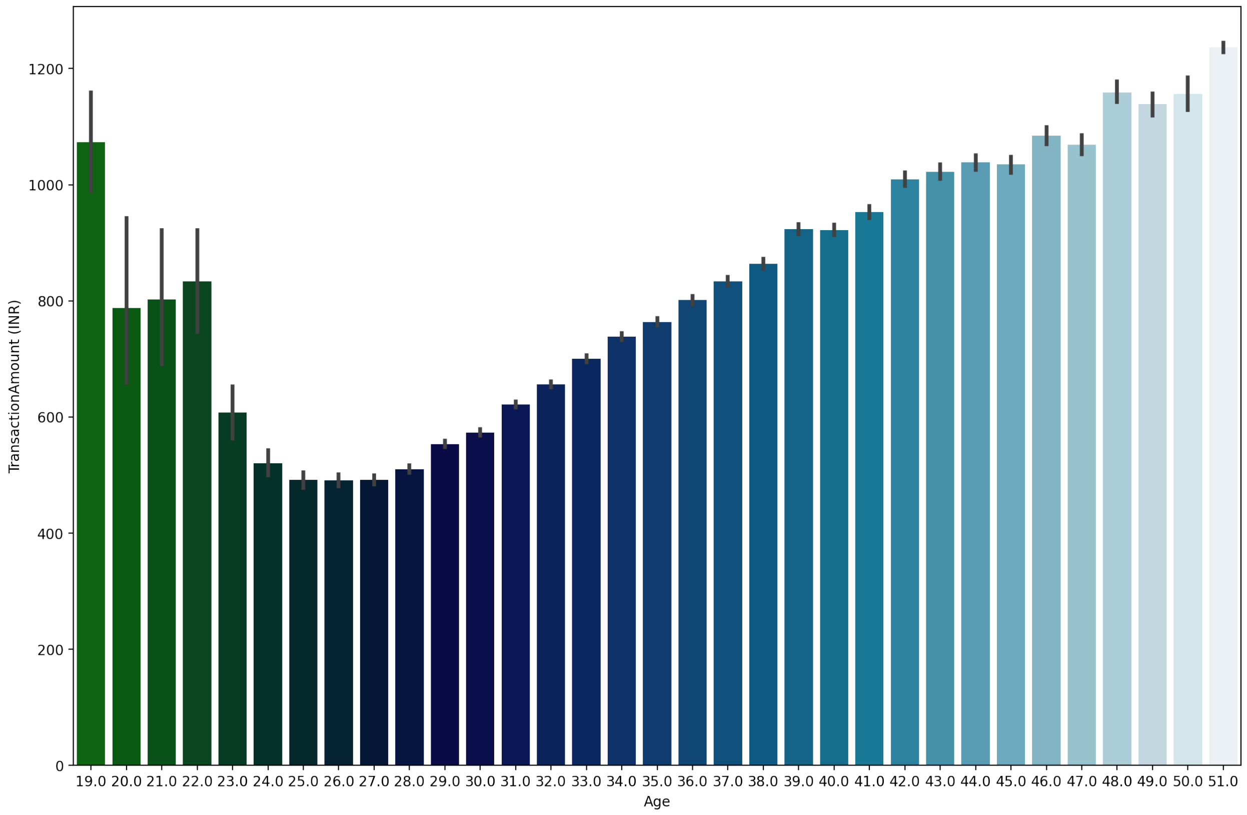
```
Out[156…  <AxesSubplot:xlabel='Age', ylabel='CustAccountBalance'>
```



In general, the account balances of customers rise abruptly with increase in their ages, attaining a peak by the old age of 50 years or more, although there are a few excepions to this matter. The account balances of younger adults in the age range of 19 to 22 years have higher account balances in comparison to their surrounding age groups.

```
In [160…  plt.figure(figsize=(15,10),dpi=200)
          sns.barplot(data=df,x='Age',y='TransactionAmount (INR)',palette='ocean')
```

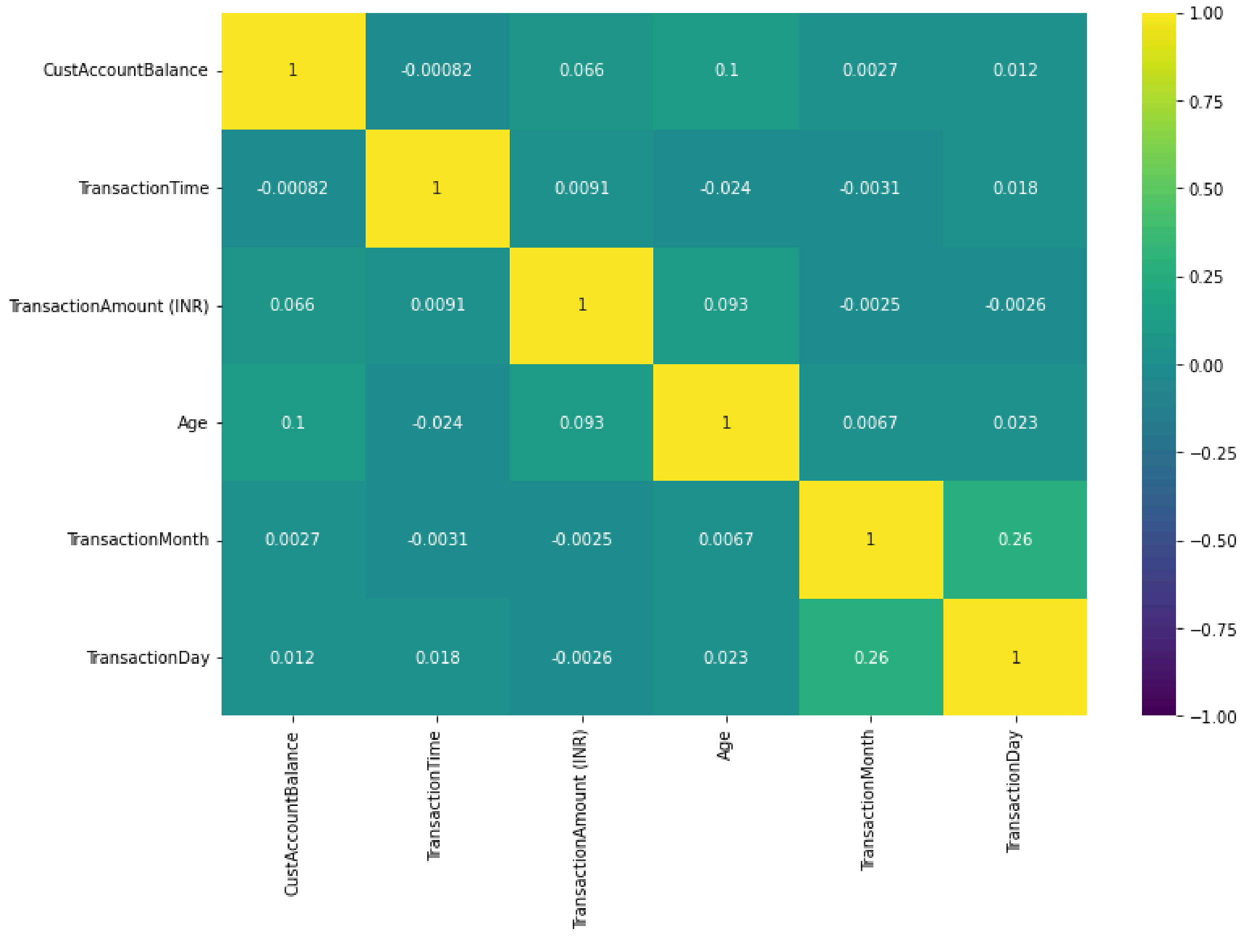
```
Out[160…  <AxesSubplot:xlabel='Age', ylabel='TransactionAmount (INR)'>
```



Likewise, an exactly similar trend prevails in case of transaction amounts as well. Younger adult customers, in the age range of 19 to 24 years, perform exorbitant transactions as compared to their surrounding age groups. This is most probably due to the reason that younger adults between the ages of 18 to 24 years have their own career aspirations to fulfil as a consequence of which they generally have lavish and extravagant demands for fulfilling their passions and interests and for facilitating their development through all means. Nevertheless, middle age senior adults and elderly customers make comparatively more extortionate transactions for managing their families and livelihoods.

```
In [39]: sns.heatmap(df.corr(),annot=True,cmap='viridis',vmin=-1,vmax=1)
```

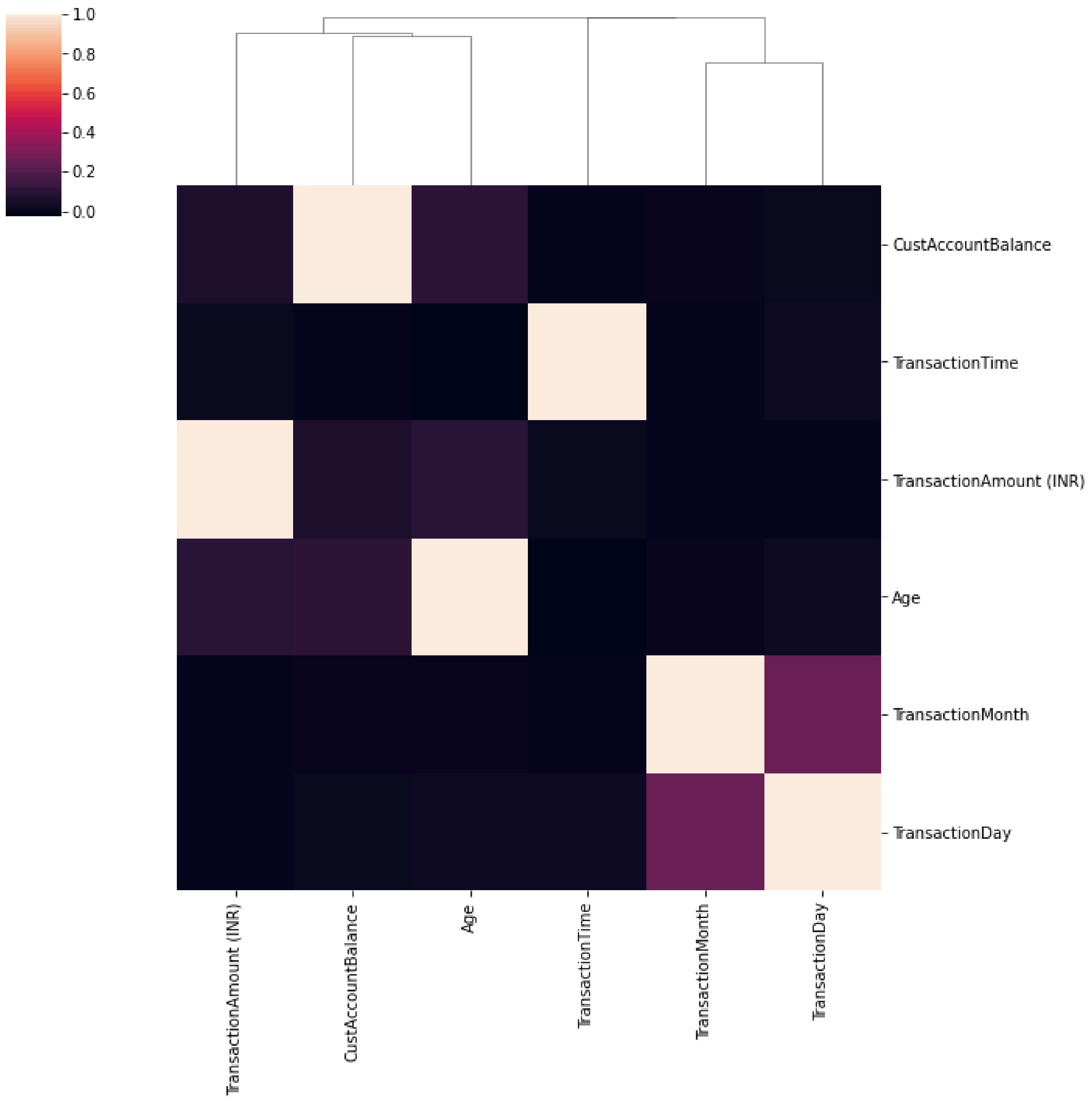
```
Out[39]: <AxesSubplot:>
```



	CustAccountBalance	TransactionTime	TransactionAmount (INR)	Age	TransactionMonth	TransactionDay
CustAccountBalance	1	-0.00082	0.066	0.1	0.0027	0.012
TransactionTime	-0.00082	1	0.0091	-0.024	-0.0031	0.018
TransactionAmount (INR)	0.066	0.0091	1	0.093	-0.0025	-0.0026
Age	0.1	-0.024	0.093	1	0.0067	0.023
TransactionMonth	0.0027	-0.0031	-0.0025	0.0067	1	0.26
TransactionDay	0.012	0.018	-0.0026	0.023	0.26	1

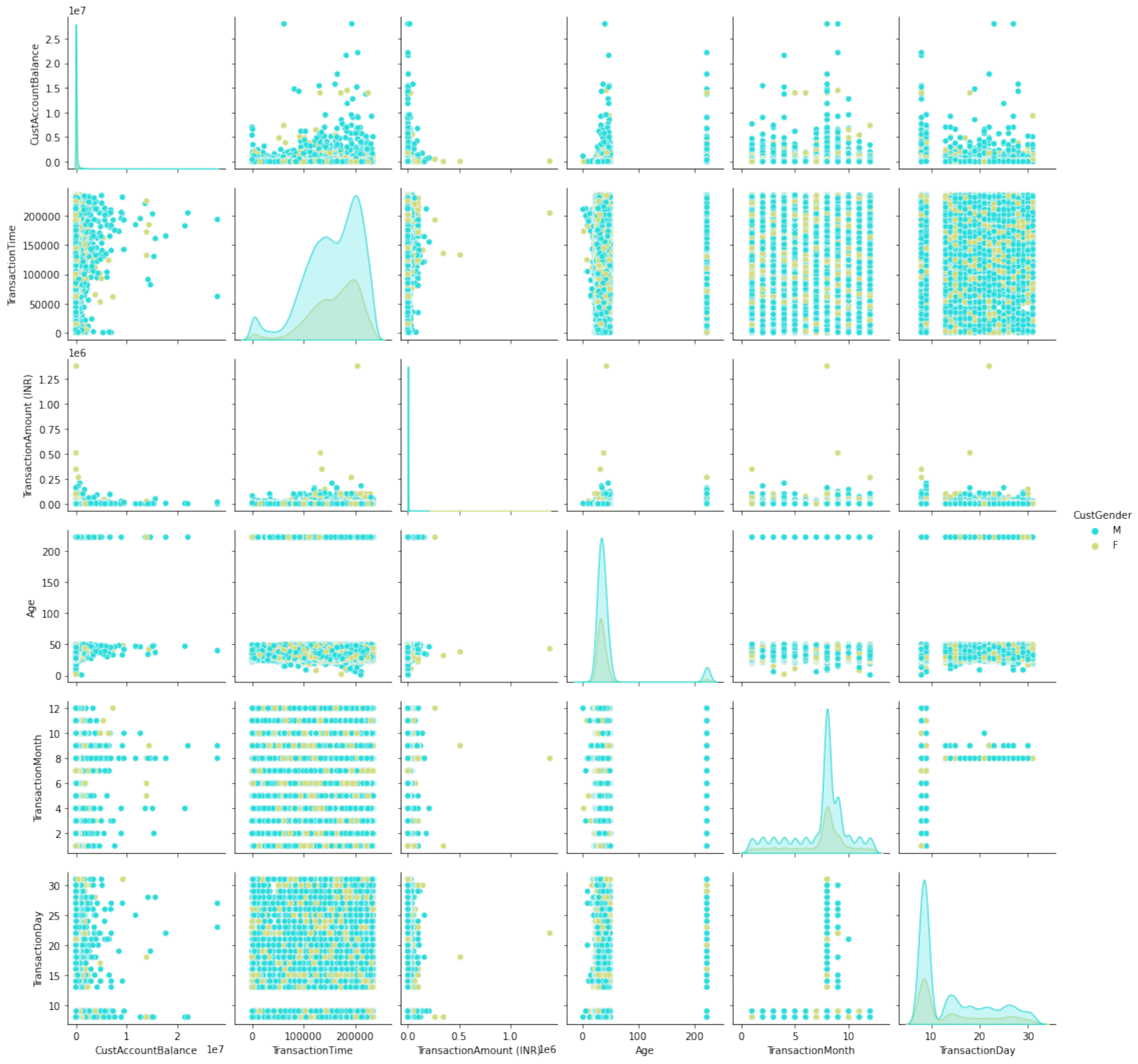
```
In [40]: sns.clustermap(df.corr(),row_cluster=False)
```

```
Out[40]: <seaborn.matrix.ClusterGrid at 0x1f59a745a30>
```



```
In [41]: sns.pairplot(df.sample(n=50000),hue='CustGender',palette='rainbow')
```

```
Out[41]: <seaborn.axisgrid.PairGrid at 0x1f59a749a30>
```



Feature Engineering Continued

```
In [42]: df.CustGender.replace(['F','M'],[0,1],inplace=True)
         df.CustGender = df.CustGender.astype(np.int64)
```

Categorical Encoding

```
In [43]: encoder = LabelEncoder()
         df.CustLocation = encoder.fit_transform(df.CustLocation)
         df.CustLocation = df.CustLocation.astype(np.int64)
```

```
In [44]: df.drop(['TransactionMonthName','TransactionDayName'],axis=1,inplace=True)
```

Feature Scaling

```
In [45]: scaler = StandardScaler()
         scaled_df = scaler.fit_transform(df)
         scaled_df = pd.DataFrame(scaled_df,columns=df.columns)
         scaled_df.head()
```

Out[45]:

	CustGender	CustLocation	CustAccountBalance	TransactionTime	Transaction
0	-1.65	-0.23	-0.15	-0.27	
1	-1.65	0.48	-0.15	-0.28	
2	-1.65	0.48	1.55	-0.28	
3	-1.65	0.64	-0.18	0.46	
4	-1.65	-0.26	-0.08	0.32	

Unsupervised Machine Learning

```
In [46]: temp = scaled_df.sample(n=50000)
```

K Means Clustering

```
In [47]: base_kmeans = KMeans(random_state=101)
         base_kmeans.fit(temp)

Out[47]: KMeans(random_state=101)
```

```
In [48]: labels = set(base_kmeans.labels_)
         labels

Out[48]: {0, 1, 2, 3, 4, 5, 6, 7}
```

```
In [49]: print("Silhouette Score:",str(np.round(silhouette_score(temp,base_kmeans.label

        Silhouette Score: 21.64%
```

```
In [50]: for i in range(1,11):
             for init in ['k-means++','random']:
                 for algo in ['auto','full','elkan']:
                     kmeans = KMeans(n_clusters=i,init=init,algorithm=algo,random_state
                     kmeans.fit(temp)
                     print("Number of Clusters: {}".format(i))
                     print("Initialization Algorithm: {}".format(init))
                     print("Algorithm: {}".format(algo))
                     print("Sum of Squared Distance: %d" % kmeans.inertia_)
```

```
            print('-----------------')
```

```
Number of Clusters: 1
Initialization Algorithm: k-means++
Algorithm: auto
Sum of Squared Distance: 389968
-----------------
Number of Clusters: 1
Initialization Algorithm: k-means++
Algorithm: full
Sum of Squared Distance: 389968
-----------------
Number of Clusters: 1
Initialization Algorithm: k-means++
Algorithm: elkan
Sum of Squared Distance: 389968
-----------------
Number of Clusters: 1
Initialization Algorithm: random
Algorithm: auto
Sum of Squared Distance: 389968
-----------------
Number of Clusters: 1
Initialization Algorithm: random
Algorithm: full
Sum of Squared Distance: 389968
-----------------
Number of Clusters: 1
Initialization Algorithm: random
Algorithm: elkan
Sum of Squared Distance: 389968
-----------------
Number of Clusters: 2
Initialization Algorithm: k-means++
Algorithm: auto
Sum of Squared Distance: 339192
-----------------
Number of Clusters: 2
Initialization Algorithm: k-means++
Algorithm: full
Sum of Squared Distance: 339192
-----------------
Number of Clusters: 2
Initialization Algorithm: k-means++
Algorithm: elkan
Sum of Squared Distance: 339192
-----------------
Number of Clusters: 2
Initialization Algorithm: random
Algorithm: auto
Sum of Squared Distance: 339754
-----------------
Number of Clusters: 2
Initialization Algorithm: random
Algorithm: full
Sum of Squared Distance: 339754
```

```
-----------------
Number of Clusters: 2
Initialization Algorithm: random
Algorithm: elkan
Sum of Squared Distance: 339754
-----------------
Number of Clusters: 3
Initialization Algorithm: k-means++
Algorithm: auto
Sum of Squared Distance: 294160
-----------------
Number of Clusters: 3
Initialization Algorithm: k-means++
Algorithm: full
Sum of Squared Distance: 294160
-----------------
Number of Clusters: 3
Initialization Algorithm: k-means++
Algorithm: elkan
Sum of Squared Distance: 294160
-----------------
Number of Clusters: 3
Initialization Algorithm: random
Algorithm: auto
Sum of Squared Distance: 303729
-----------------
Number of Clusters: 3
Initialization Algorithm: random
Algorithm: full
Sum of Squared Distance: 303729
-----------------
Number of Clusters: 3
Initialization Algorithm: random
Algorithm: elkan
Sum of Squared Distance: 303729
-----------------
Number of Clusters: 4
Initialization Algorithm: k-means++
Algorithm: auto
Sum of Squared Distance: 257782
-----------------
Number of Clusters: 4
Initialization Algorithm: k-means++
Algorithm: full
Sum of Squared Distance: 257782
-----------------
Number of Clusters: 4
Initialization Algorithm: k-means++
Algorithm: elkan
Sum of Squared Distance: 257782
-----------------
Number of Clusters: 4
Initialization Algorithm: random
Algorithm: auto
```

```
Sum of Squared Distance: 257782
-----------------
Number of Clusters: 4
Initialization Algorithm: random
Algorithm: full
Sum of Squared Distance: 257782
-----------------
Number of Clusters: 4
Initialization Algorithm: random
Algorithm: elkan
Sum of Squared Distance: 257782
-----------------
Number of Clusters: 5
Initialization Algorithm: k-means++
Algorithm: auto
Sum of Squared Distance: 235212
-----------------
Number of Clusters: 5
Initialization Algorithm: k-means++
Algorithm: full
Sum of Squared Distance: 235212
-----------------
Number of Clusters: 5
Initialization Algorithm: k-means++
Algorithm: elkan
Sum of Squared Distance: 235212
-----------------
Number of Clusters: 5
Initialization Algorithm: random
Algorithm: auto
Sum of Squared Distance: 230769
-----------------
Number of Clusters: 5
Initialization Algorithm: random
Algorithm: full
Sum of Squared Distance: 230769
-----------------
Number of Clusters: 5
Initialization Algorithm: random
Algorithm: elkan
Sum of Squared Distance: 230769
-----------------
Number of Clusters: 6
Initialization Algorithm: k-means++
Algorithm: auto
Sum of Squared Distance: 207129
-----------------
Number of Clusters: 6
Initialization Algorithm: k-means++
Algorithm: full
Sum of Squared Distance: 207129
-----------------
Number of Clusters: 6
Initialization Algorithm: k-means++
Algorithm: elkan
Sum of Squared Distance: 207129
-----------------
Number of Clusters: 6
Initialization Algorithm: random
Algorithm: auto
Sum of Squared Distance: 208615
-----------------
Number of Clusters: 6
Initialization Algorithm: random
Algorithm: full
Sum of Squared Distance: 208615
-----------------
Number of Clusters: 6
Initialization Algorithm: random
Algorithm: elkan
Sum of Squared Distance: 208615
-----------------
Number of Clusters: 7
Initialization Algorithm: k-means++
Algorithm: auto
Sum of Squared Distance: 187365
-----------------
Number of Clusters: 7
Initialization Algorithm: k-means++
Algorithm: full
Sum of Squared Distance: 187365
-----------------
Number of Clusters: 7
Initialization Algorithm: k-means++
Algorithm: elkan
Sum of Squared Distance: 187365
-----------------
Number of Clusters: 7
Initialization Algorithm: random
Algorithm: auto
Sum of Squared Distance: 187322
-----------------
Number of Clusters: 7
Initialization Algorithm: random
Algorithm: full
Sum of Squared Distance: 187322
-----------------
Number of Clusters: 7
Initialization Algorithm: random
Algorithm: elkan
Sum of Squared Distance: 187322
-----------------
Number of Clusters: 8
Initialization Algorithm: k-means++
Algorithm: auto
Sum of Squared Distance: 178582
-----------------
Number of Clusters: 8
```

```
Initialization Algorithm: k-means++
Algorithm: full
Sum of Squared Distance: 178582
-----------------
Number of Clusters: 8
Initialization Algorithm: k-means++
Algorithm: elkan
Sum of Squared Distance: 178582
-----------------
Number of Clusters: 8
Initialization Algorithm: random
Algorithm: auto
Sum of Squared Distance: 171367
-----------------
Number of Clusters: 8
Initialization Algorithm: random
Algorithm: full
Sum of Squared Distance: 171367
-----------------
Number of Clusters: 8
Initialization Algorithm: random
Algorithm: elkan
Sum of Squared Distance: 171367
-----------------
Number of Clusters: 9
Initialization Algorithm: k-means++
Algorithm: auto
Sum of Squared Distance: 158824
-----------------
Number of Clusters: 9
Initialization Algorithm: k-means++
Algorithm: full
Sum of Squared Distance: 158824
-----------------
Number of Clusters: 9
Initialization Algorithm: k-means++
Algorithm: elkan
Sum of Squared Distance: 158824
-----------------
Number of Clusters: 9
Initialization Algorithm: random
Algorithm: auto
Sum of Squared Distance: 158829
-----------------
Number of Clusters: 9
Initialization Algorithm: random
Algorithm: full
Sum of Squared Distance: 158829
-----------------
Number of Clusters: 9
Initialization Algorithm: random
Algorithm: elkan
Sum of Squared Distance: 158829
-----------------
```

```
Number of Clusters: 10
Initialization Algorithm: k-means++
Algorithm: auto
Sum of Squared Distance: 148814
-----------------
Number of Clusters: 10
Initialization Algorithm: k-means++
Algorithm: full
Sum of Squared Distance: 148814
-----------------
Number of Clusters: 10
Initialization Algorithm: k-means++
Algorithm: elkan
Sum of Squared Distance: 148814
-----------------
Number of Clusters: 10
Initialization Algorithm: random
Algorithm: auto
Sum of Squared Distance: 148818
-----------------
Number of Clusters: 10
Initialization Algorithm: random
Algorithm: full
Sum of Squared Distance: 148818
-----------------
Number of Clusters: 10
Initialization Algorithm: random
Algorithm: elkan
Sum of Squared Distance: 148818
-----------------
```

In [51]:
```
optimized_kmeans = KMeans(n_clusters=10,init='k-means++',algorithm='elkan',rar
optimized_kmeans.fit(temp)
print("Silhouette Score:",str(np.round(silhouette_score(temp,optimized_kmeans.
```

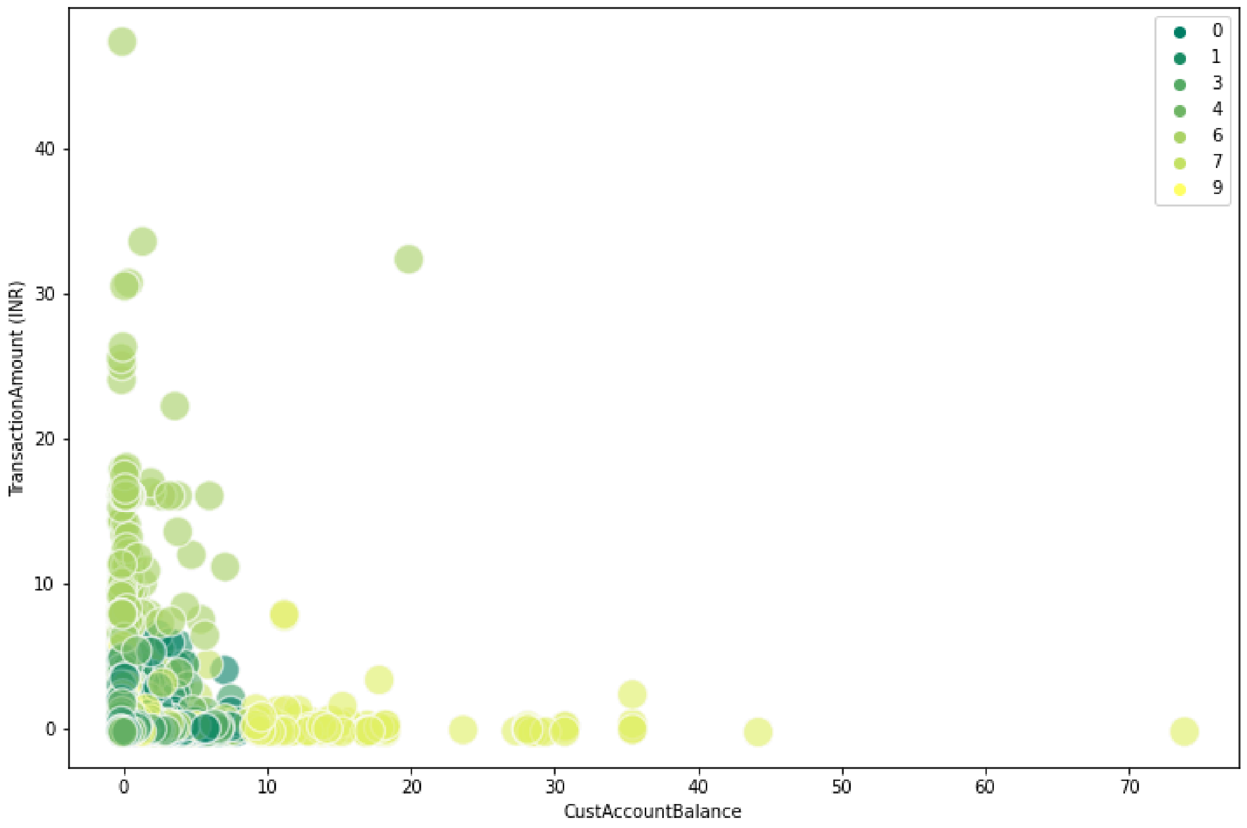
```
Silhouette Score: 23.3%
```

In [52]:
```
print("Estimated number of clusters:", len(set(optimized_kmeans.labels_)), set
print("Estimated number of noise points:", list(optimized_kmeans.labels_).cour
```

```
Estimated number of clusters: 10 {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Estimated number of noise points: 0
```

In [53]:
```
sns.scatterplot(data=temp,x='CustAccountBalance',y='TransactionAmount (INR)',h
```

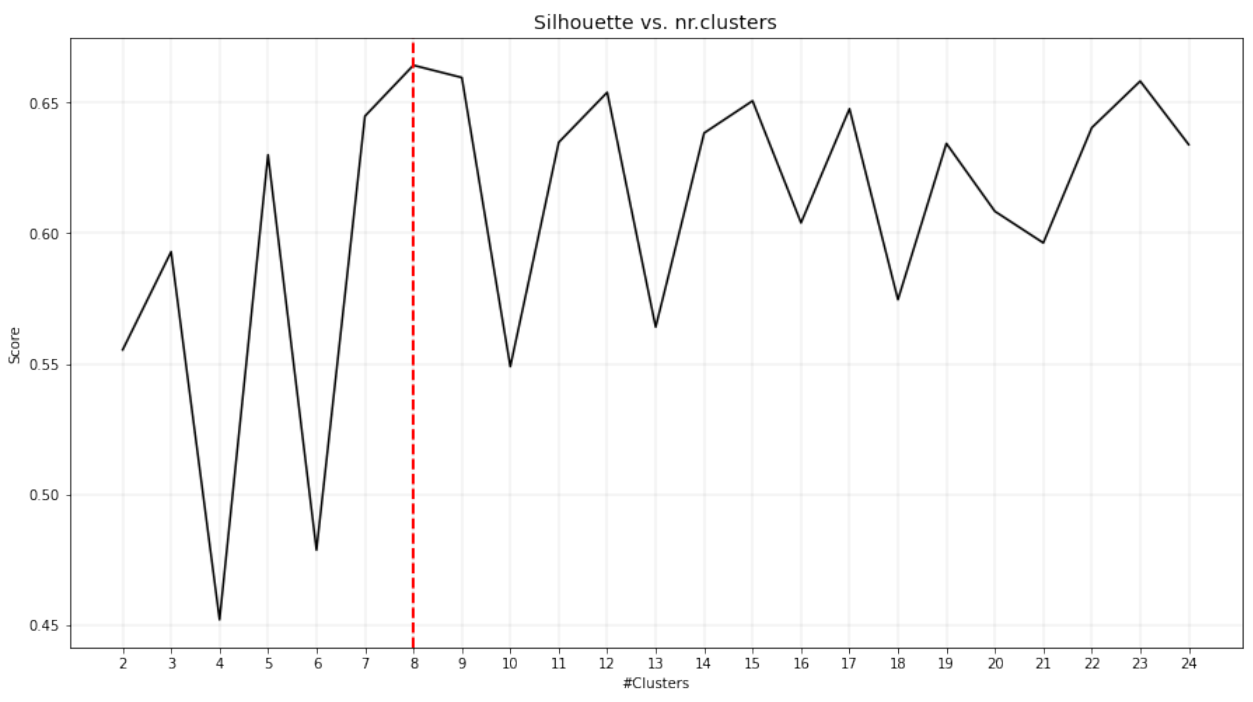
Out[53]:
```
<AxesSubplot:xlabel='CustAccountBalance', ylabel='TransactionAmount (INR)'>
```



```
In [54]: ce = clusteval(cluster='kmeans',savemem=True,verbose=4)
         results = ce.fit(np.reshape(np.ravel(temp.sample(5000)),(-1,1)))
         cluster_labels = results['labx']
```

```
[clusteval] >Fit using kmeans with metric: euclidean, and linkage: ward
[clusteval] >Evaluate using silhouette.
[clusteval] >Save memory enabled for kmeans with evaluation silhouette.
10
0%|██████████████████████████████████████████████████████████████████████████
██████| 23/23 [08:53<00:00, 23.20s/it]
[clusteval] >Optimal number clusters detected: [8].
[clusteval] >Fin.
```

```
In [55]: ce.plot()
```



```
Out[55]: (<Figure size 1080x576 with 1 Axes>,
          <AxesSubplot:title={'center':'Silhouette vs. nr.clusters'}, xlabel='#Cluster
         s', ylabel='Score'>)
```

Hierarchical Clustering

```
In [56]: mmscaler = MinMaxScaler()
         minmax_scaled_df = mmscaler.fit_transform(df)
         minmax_scaled_df = pd.DataFrame(minmax_scaled_df,columns=df.columns)
         minmax_scaled_df.head()
```

Out[56]:

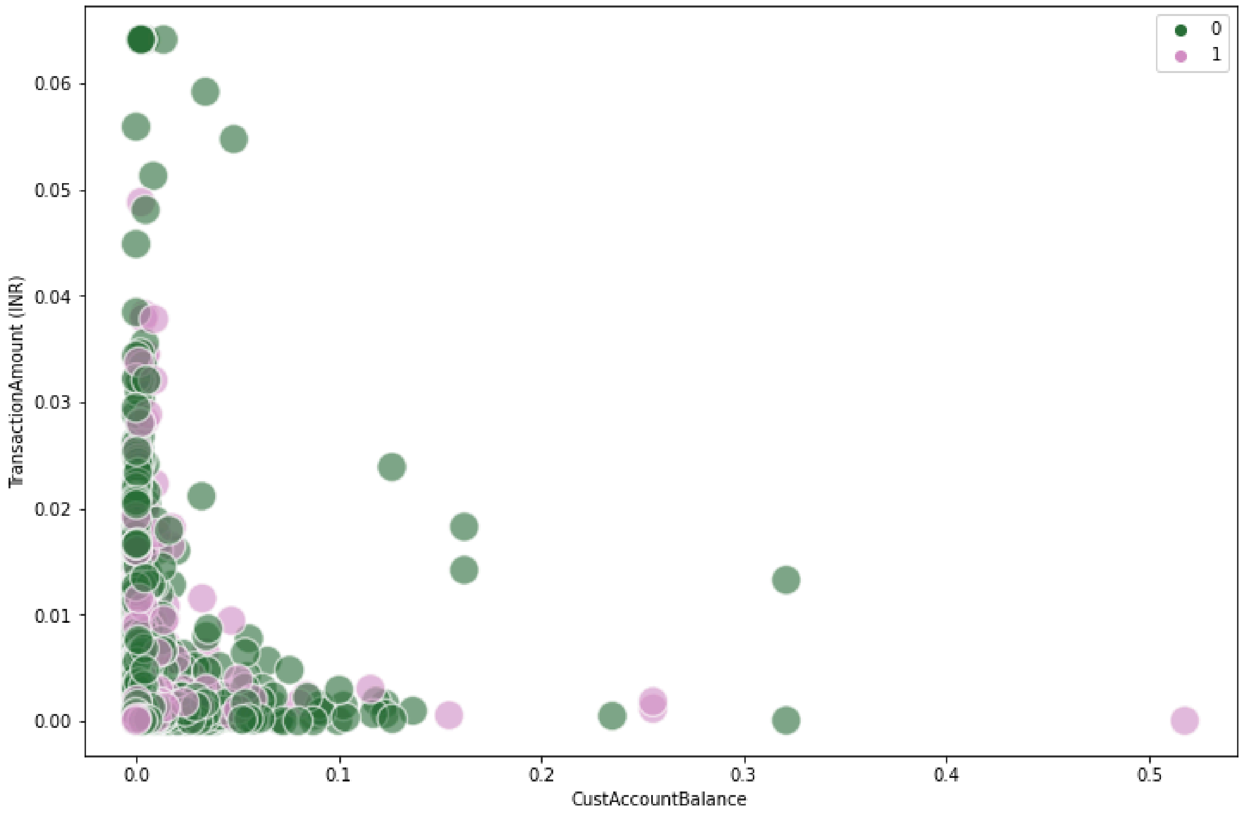
	CustGender	CustLocation	CustAccountBalance	TransactionTime	Transaction
0	0.00	0.38	0.00	0.61	
1	0.00	0.56	0.00	0.60	
2	0.00	0.56	0.02	0.60	
3	0.00	0.60	0.00	0.77	
4	0.00	0.37	0.00	0.74	

```
In [57]: temp2 = minmax_scaled_df.sample(n=20000)
```

```
In [58]: base_hc = AgglomerativeClustering()
         base_hc.fit(temp2)
```

```
Out[58]: AgglomerativeClustering()
```

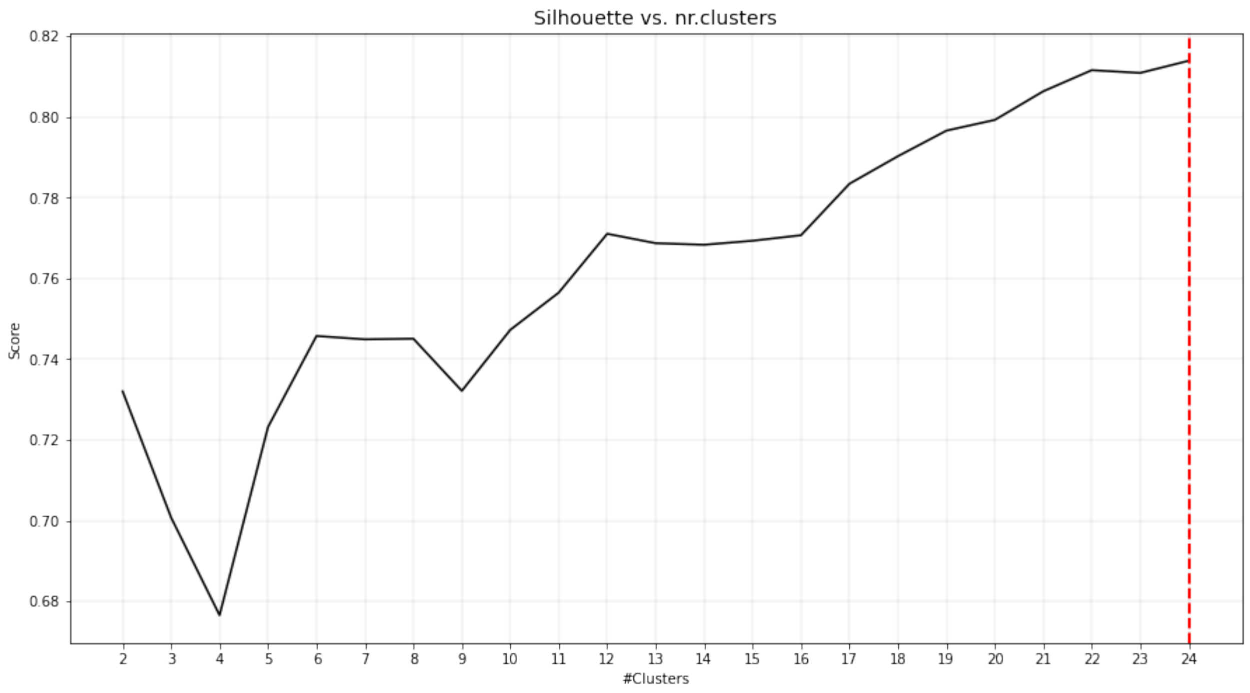
```
In [59]: print("Estimated number of clusters:", set(base_hc.labels_))

         Estimated number of clusters: {0, 1}

In [60]: print("Estimated number of noise points:", list(base_hc.labels_).count(-1))

         Estimated number of noise points: 0

In [61]: print("Silhouette Score:",str(np.round(silhouette_score(temp2,base_hc.labels_)

         Silhouette Score: 41.83%

In [62]: sns.scatterplot(temp2['CustAccountBalance'],temp2['TransactionAmount (INR)'],h

Out[62]: <AxesSubplot:xlabel='CustAccountBalance', ylabel='TransactionAmount (INR)'>
```



```
In [63]: temp2_samp = temp2.sample(500)
         ce = clusteval(verbose=4)
         results = ce.fit(np.reshape(np.ravel(temp2_samp),(-1,1)))
         results

         [clusteval] >Fit using agglomerative with metric: euclidean, and linkage: ward
         [clusteval] >Evaluate using silhouette.
         10
         0%|██████████████████████████████████████████████████████████████████████████████
         ██████| 23/23 [00:05<00:00,  4.13it/s]
         [clusteval] >Compute dendrogram threshold.
         [clusteval] >Optimal number clusters detected: [24].
         [clusteval] >Fin.
```

```
Out[63]: {'evaluate': 'silhouette',
          'score':     clusters  score
          0          2   0.73
          1          3   0.70
          2          4   0.68
          3          5   0.72
          4          6   0.75
          5          7   0.74
          6          8   0.75
          7          9   0.73
          8         10   0.75
          9         11   0.76
          10        12   0.77
          11        13   0.77
          12        14   0.77
          13        15   0.77
          14        16   0.77
          15        17   0.78
          16        18   0.79
          17        19   0.80
          18        20   0.80
          19        21   0.81
          20        22   0.81
          21        23   0.81
          22        24   0.81,
          'labx': array([ 8, 11,  0, ...,  7, 20,  0], dtype=int32),
          'fig': {'silscores': array([0.73201178, 0.70076176, 0.67656633, 0.72318167,
         0.7457209 ,
                  0.74489544, 0.74505237, 0.73213675, 0.74724807, 0.75646611,
                  0.77104052, 0.76870641, 0.76832357, 0.76931071, 0.77067574,
                  0.7834333 , 0.79025257, 0.79658599, 0.79924515, 0.80633523,
                  0.81156296, 0.81088104, 0.81389328]),
           'sillclust': array([ 2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12.,
         13., 14.,
                  15., 16., 17., 18., 19., 20., 21., 22., 23., 24.]),
           'clustcutt': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
         16, 17, 18,
                  19, 20, 21, 22, 23, 24])},
          'max_d': 0.24457325037383307,
          'max_d_lower': 0.2400642909610462,
          'max_d_upper': 0.24908220978661993}
```

```
In [64]: ce.plot()
```



```
Out[64]: (<Figure size 1080x576 with 1 Axes>,
          <AxesSubplot:title={'center':'Silhouette vs. nr.clusters'}, xlabel='#Cluster
         s', ylabel='Score'>)
```

Density-based Spatial Clustering with Applications with Noise(DBSCAN)

```
In [180…  base_dbscan = DBSCAN()
          base_dbscan.fit(temp)

Out[180…  DBSCAN()
```

```
In [181…  print("Estimated number of clusters:", set(base_dbscan.labels_))

        Estimated number of clusters: {0, 1, 2, -1}
```

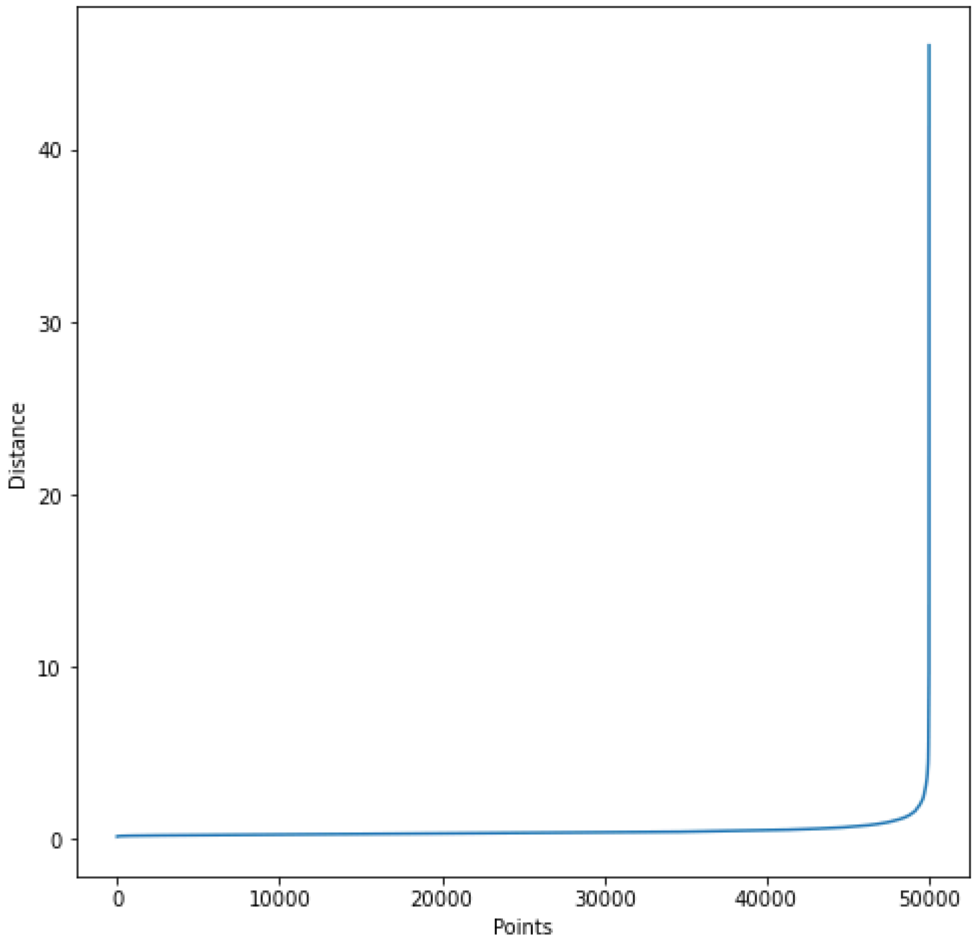
```
In [182…  print("Estimated number of noise points:", list(base_dbscan.labels_).count(-1)

        Estimated number of noise points: 475
```

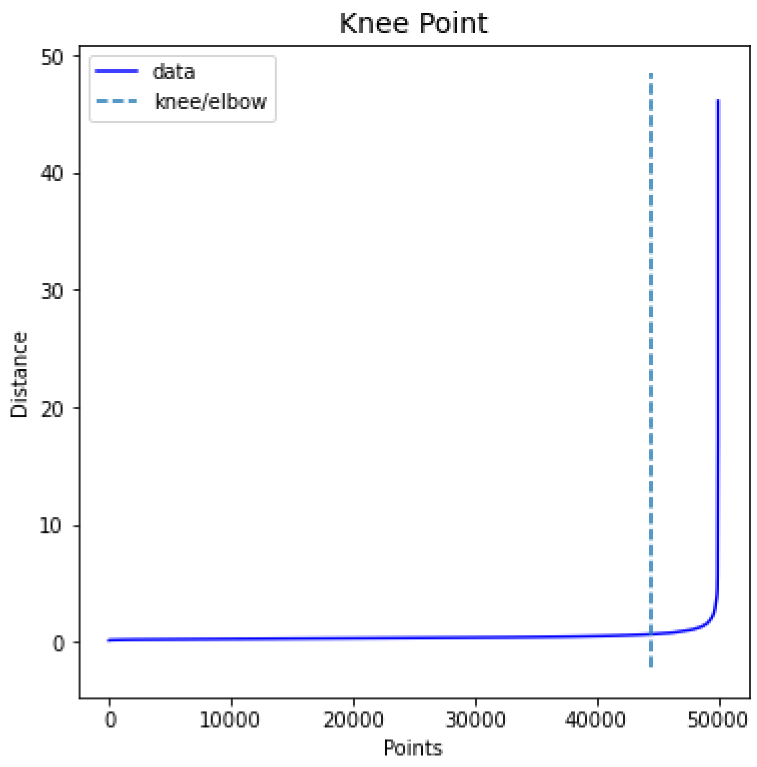
```
In [183…  print("Silhouette Score:",str(np.round(silhouette_score(temp,base_dbscan.label

        Silhouette Score: -27.94%
```

```
In [69]:  def get_kdist_plot(X=None, k=None, radius_nbrs=1.0):
              nbrs = NearestNeighbors(n_neighbors=k, radius=radius_nbrs).fit(X)
              distances, indices = nbrs.kneighbors(X)
              distances = np.sort(distances[:,k-1], axis=0)
              plt.figure(figsize=(8,8))
              plt.plot(distances)
              plt.xlabel("Points")
              plt.ylabel("Distance")
```

```
    plt.show()
    i = np.arange(len(distances))
    knee = KneeLocator(i, distances, S=1, curve='convex', direction='increasir
    knee.plot_knee()
    plt.xlabel("Points")
    plt.ylabel("Distance")
    plt.show()
    return distances[knee.knee]
```

```
In [70]: k = 2 * temp.shape[-1]
         x = get_kdist_plot(temp,k)
         print("Knee Point:",x)
```





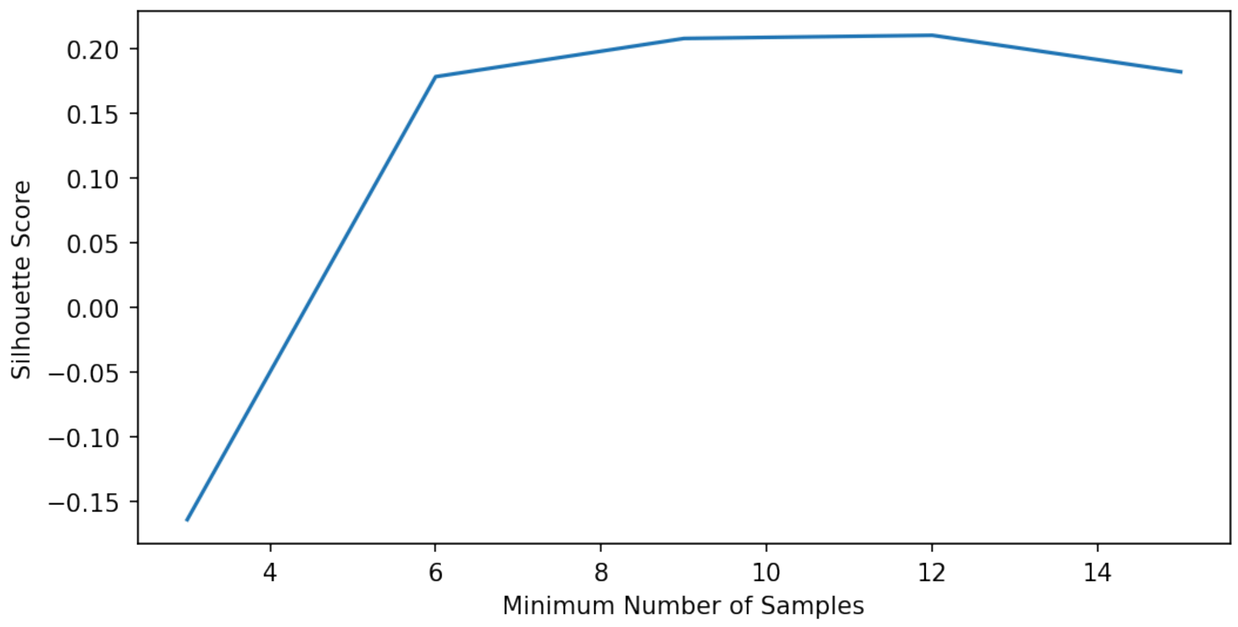
```
Knee Point: 0.6920513133696032
```

```
In [71]: ms = np.arange(3,2*temp.shape[1],3)
         silhouette_scores = []

         for i in ms:
             dbscan = DBSCAN(eps=x,min_samples=i)
             dbscan.fit(temp)
             silhouette_scores.append(silhouette_score(temp,dbscan.labels_))
             print("{} Minimum Samples Tested!".format(i))
```

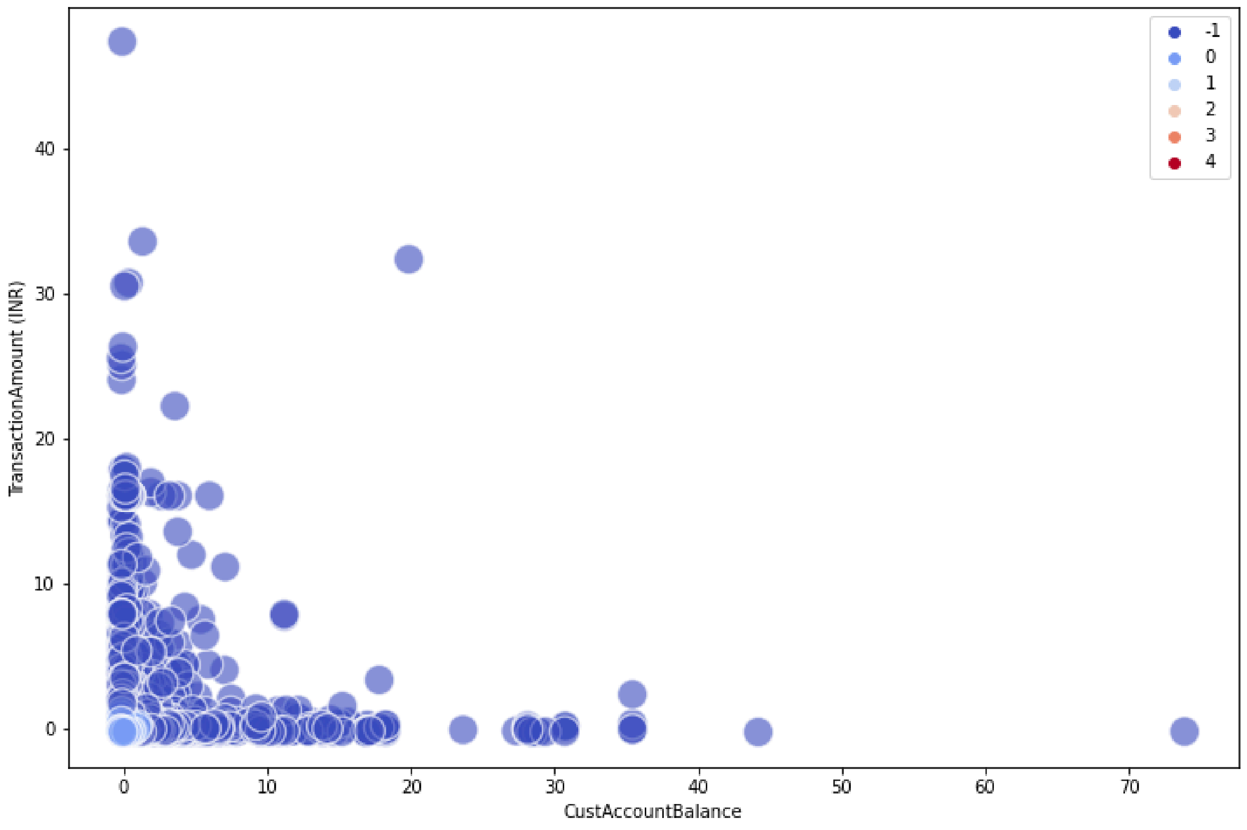
```
3 Minimum Samples Tested!
6 Minimum Samples Tested!
9 Minimum Samples Tested!
12 Minimum Samples Tested!
15 Minimum Samples Tested!
```

```
In [72]: plt.figure(figsize=(8,4),dpi=150)
         sns.lineplot(ms,silhouette_scores)
         plt.xlabel('Minimum Number of Samples')
         plt.ylabel('Silhouette Score')
```

```
Out[72]: Text(0, 0.5, 'Silhouette Score')
```



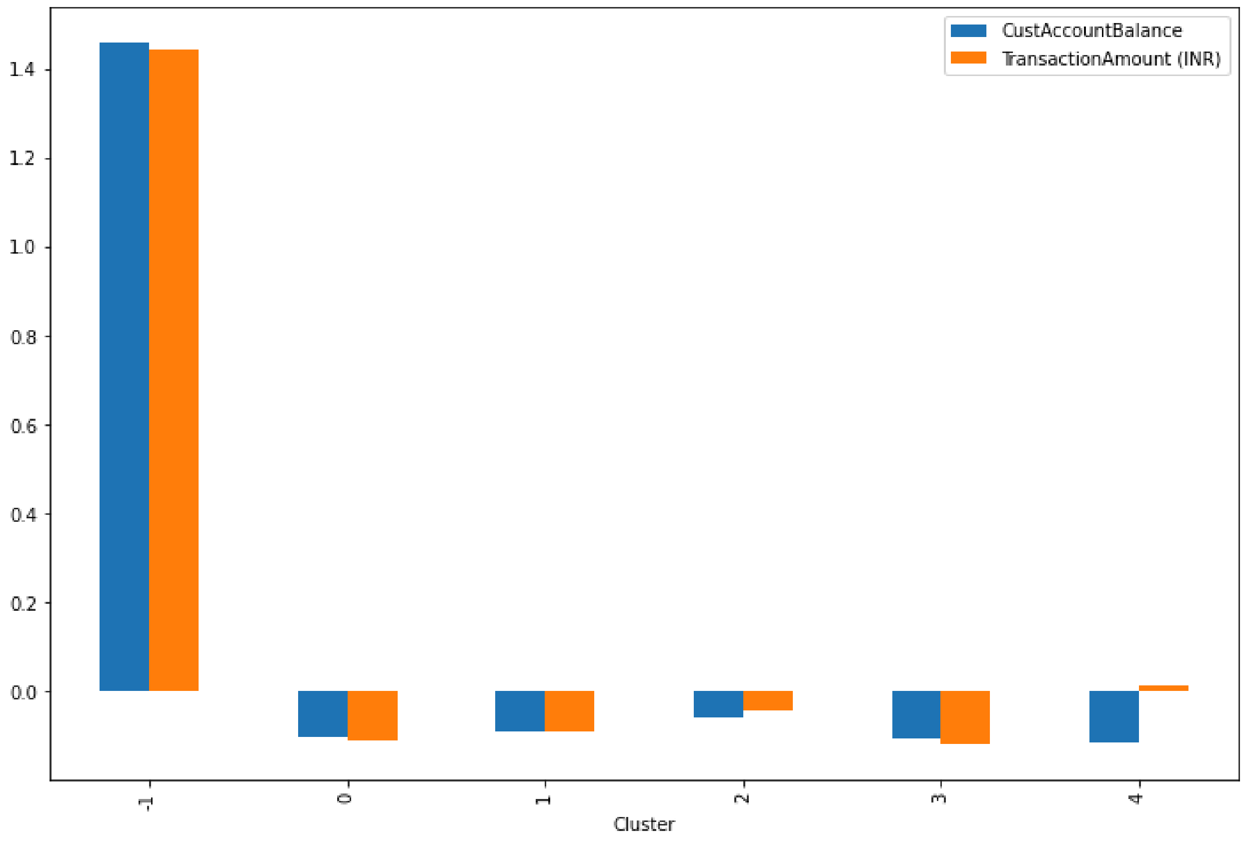
```
In [175…  optimized_dbscan = DBSCAN(eps=x,min_samples=11)
          optimized_dbscan.fit(temp)

Out[175…  DBSCAN(eps=0.6920513133696032, min_samples=11)

In [176…  print("Estimated number of clusters:",set(optimized_dbscan.labels_))

         Estimated number of clusters: {0, 1, -1}

In [177…  print("Estimated number of noise points:",list(optimized_dbscan.labels_).count

         Estimated number of noise points: 473

In [178…  print("Silhouette Score:",str(np.round(silhouette_score(temp,optimized_dbscan.

         Silhouette Score: -22.8%

In [77]:  sns.scatterplot(data=temp,x='CustAccountBalance',y='TransactionAmount (INR)',h

Out[77]:  <AxesSubplot:xlabel='CustAccountBalance', ylabel='TransactionAmount (INR)'>
```



```
In [78]: temp['Cluster'] = optimized_dbscan.labels_
```

```
In [79]: temp.groupby('Cluster')[['CustAccountBalance','TransactionAmount (INR)']].mear
```

```
Out[79]: <AxesSubplot:xlabel='Cluster'>
```



Determination of Optimum Epsilon Value for DBSCAN Model

In [159…

```
temp = temp.sample(500)
ce = clusteval(cluster='dbscan',verbose=3)
results = ce.fit(np.reshape(np.ravel(temp),(-1,1)))
results
```

```
[clusteval] >Fit using dbscan with metric: euclidean, and linkage: ward
[clusteval] >Gridsearch across epsilon..
[clusteval] >Evaluate using silhouette..
10
0%|████████████████████████████████████████████████████████████████████████████
████| 245/245 [02:56<00:00,  1.39it/s]
[clusteval] >Compute dendrogram threshold.
[clusteval] >Optimal number clusters detected: [2].
[clusteval] >Fin.
```

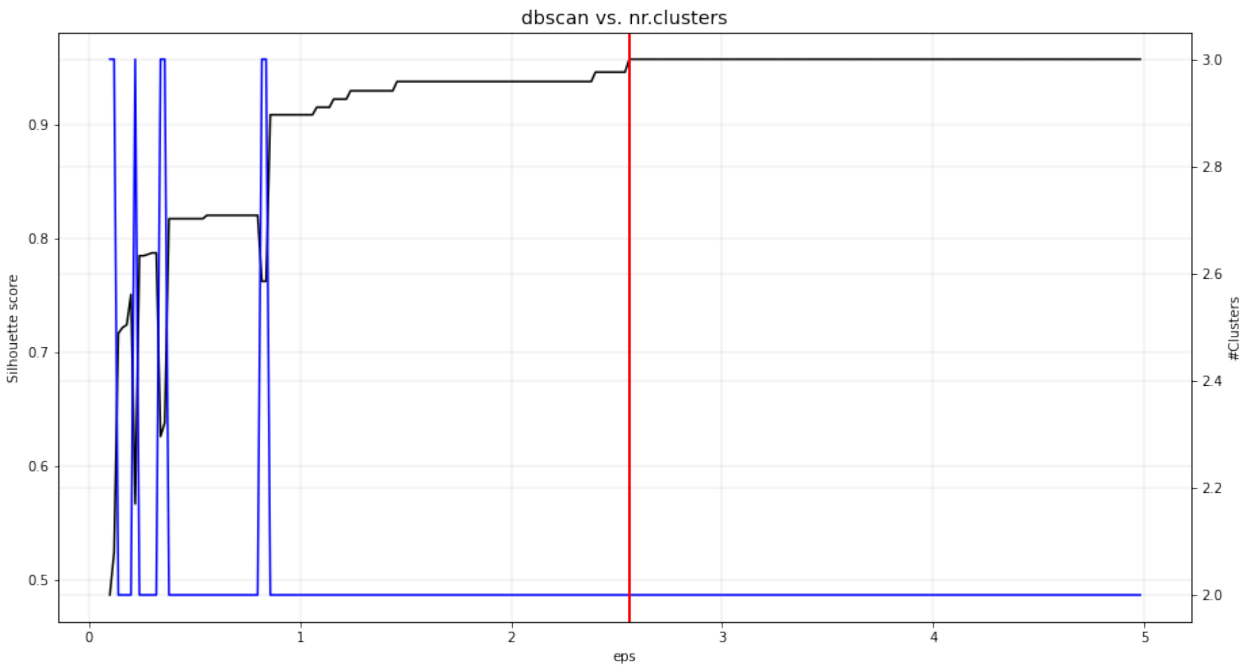
```
Out[159… {'evaluate': 'dbscan',
          'labx': array([0, 0, 0, ..., 0, 0, 0], dtype=int64),
          'fig': {'eps': array([0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 , 0.22, 0.24, 0.26,
         0.28, 0.3 ,
                  0.32, 0.34, 0.36, 0.38, 0.4 , 0.42, 0.44, 0.46, 0.48, 0.5 , 0.52,
                  0.54, 0.56, 0.58, 0.6 , 0.62, 0.64, 0.66, 0.68, 0.7 , 0.72, 0.74,
                  0.76, 0.78, 0.8 , 0.82, 0.84, 0.86, 0.88, 0.9 , 0.92, 0.94, 0.96,
                  0.98, 1.  , 1.02, 1.04, 1.06, 1.08, 1.1 , 1.12, 1.14, 1.16, 1.18,
                  1.2 , 1.22, 1.24, 1.26, 1.28, 1.3 , 1.32, 1.34, 1.36, 1.38, 1.4 ,
                  1.42, 1.44, 1.46, 1.48, 1.5 , 1.52, 1.54, 1.56, 1.58, 1.6 , 1.62,
                  1.64, 1.66, 1.68, 1.7 , 1.72, 1.74, 1.76, 1.78, 1.8 , 1.82, 1.84,
                  1.86, 1.88, 1.9 , 1.92, 1.94, 1.96, 1.98, 2.  , 2.02, 2.04, 2.06,
                  2.08, 2.1 , 2.12, 2.14, 2.16, 2.18, 2.2 , 2.22, 2.24, 2.26, 2.28,
                  2.3 , 2.32, 2.34, 2.36, 2.38, 2.4 , 2.42, 2.44, 2.46, 2.48, 2.5 ,
                  2.52, 2.54, 2.56, 2.58, 2.6 , 2.62, 2.64, 2.66, 2.68, 2.7 , 2.72,
                  2.74, 2.76, 2.78, 2.8 , 2.82, 2.84, 2.86, 2.88, 2.9 , 2.92, 2.94,
                  2.96, 2.98, 3.  , 3.02, 3.04, 3.06, 3.08, 3.1 , 3.12, 3.14, 3.16,
                  3.18, 3.2 , 3.22, 3.24, 3.26, 3.28, 3.3 , 3.32, 3.34, 3.36, 3.38,
                  3.4 , 3.42, 3.44, 3.46, 3.48, 3.5 , 3.52, 3.54, 3.56, 3.58, 3.6 ,
                  3.62, 3.64, 3.66, 3.68, 3.7 , 3.72, 3.74, 3.76, 3.78, 3.8 , 3.82,
                  3.84, 3.86, 3.88, 3.9 , 3.92, 3.94, 3.96, 3.98, 4.  , 4.02, 4.04,
                  4.06, 4.08, 4.1 , 4.12, 4.14, 4.16, 4.18, 4.2 , 4.22, 4.24, 4.26,
                  4.28, 4.3 , 4.32, 4.34, 4.36, 4.38, 4.4 , 4.42, 4.44, 4.46, 4.48,
                  4.5 , 4.52, 4.54, 4.56, 4.58, 4.6 , 4.62, 4.64, 4.66, 4.68, 4.7 ,
                  4.72, 4.74, 4.76, 4.78, 4.8 , 4.82, 4.84, 4.86, 4.88, 4.9 , 4.92,
                  4.94, 4.96, 4.98]),
           'silscores': array([0.48646649, 0.52376845, 0.71613409, 0.72125866, 0.72374
         983,
                  0.75039628, 0.56639651, 0.78436888, 0.78436888, 0.78556084,
                  0.78691888, 0.78691888, 0.62570887, 0.63764787, 0.81688864,
                  0.81688864, 0.81688864, 0.81688864, 0.81688864, 0.81688864,
                  0.81688864, 0.81688864, 0.81688864, 0.81988652, 0.81988652,
                  0.81988652, 0.81988652, 0.81988652, 0.81988652, 0.81988652,
                  0.81988652, 0.81988652, 0.81988652, 0.81988652, 0.81988652,
                  0.81988652, 0.76197565, 0.76197565, 0.90819581, 0.90819581,
                  0.90819581, 0.90819581, 0.90819581, 0.90819581, 0.90819581,
                  0.90819581, 0.90819581, 0.90819581, 0.90819581, 0.91476177,
                  0.91476177, 0.91476177, 0.91476177, 0.92193707, 0.92193707,
                  0.92193707, 0.92193707, 0.92924488, 0.92924488, 0.92924488,
                  0.92924488, 0.92924488, 0.92924488, 0.92924488, 0.92924488,
                  0.92924488, 0.92924488, 0.92924488, 0.93736677, 0.93736677,
                  0.93736677, 0.93736677, 0.93736677, 0.93736677, 0.93736677,
                  0.93736677, 0.93736677, 0.93736677, 0.93736677, 0.93736677,
                  0.93736677, 0.93736677, 0.93736677, 0.93736677, 0.93736677,
                  0.93736677, 0.93736677, 0.93736677, 0.93736677, 0.93736677,
                  0.93736677, 0.93736677, 0.93736677, 0.93736677, 0.93736677,
                  0.93736677, 0.93736677, 0.93736677, 0.93736677, 0.93736677,
                  0.93736677, 0.93736677, 0.93736677, 0.93736677, 0.93736677,
                  0.93736677, 0.93736677, 0.93736677, 0.93736677, 0.93736677,
                  0.93736677, 0.93736677, 0.93736677, 0.93736677, 0.93736677,
                  0.94562036, 0.94562036, 0.94562036, 0.94562036, 0.94562036,
                  0.94562036, 0.94562036, 0.94562036, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464,
                  0.95700464, 0.95700464, 0.95700464, 0.95700464, 0.95700464]),
           'sillclust': array([3., 3., 2., 2., 2., 2., 3., 2., 2., 2., 2., 2., 3., 3.,
         2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 3., 3., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                  2., 2., 2., 2., 2., 2., 2.]),
           'idx': 123},
          'n_clusters': 1,
          'max_d': 16.360121103143733,
          'max_d_lower': 15.265486162923423,
          'max_d_upper': 17.45475604336404}
```

```
In [160…  cluster_labels = results['labx']
          cluster_labels
```

```
Out[160…  array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

```
In [161…  print("Distinct Cluster Labels Detected:",np.unique(cluster_labels))

         Distinct Cluster Labels Detected: [-1  0]
```

```
In [162…  ce.plot()
```



```
Out[162…  (<Figure size 1080x576 with 2 Axes>,
           (<AxesSubplot:title={'center':'dbscan vs. nr.clusters'}, xlabel='eps', ylabe
          l='Silhouette score'>,
            <AxesSubplot:ylabel='#Clusters'>))
```

```
In [171…  enhanced_dbscan = DBSCAN(eps=3,min_samples=2*temp.shape[1])
          enhanced_dbscan.fit(temp)
```

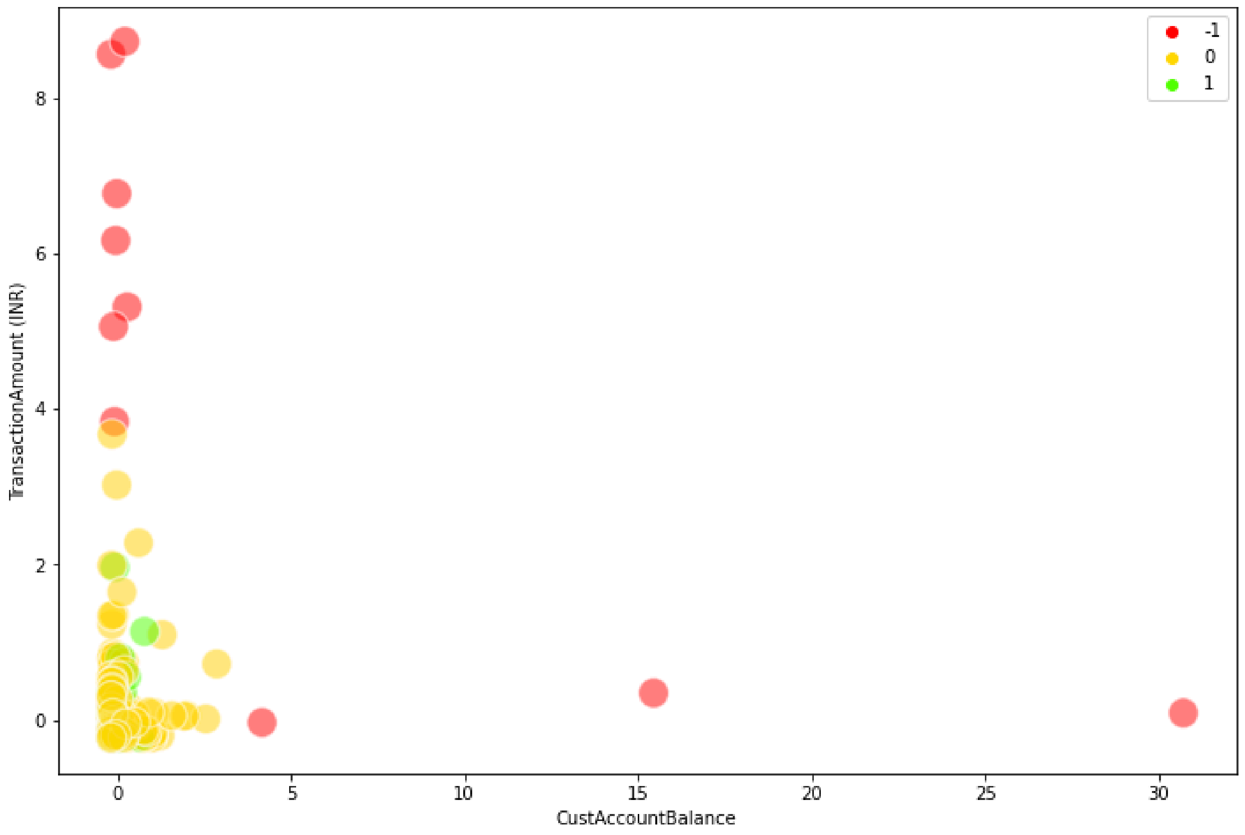
```
Out[171…  DBSCAN(eps=3, min_samples=16)
```

```
In [172…  print("Estimated number of clusters:",len(set(enhanced_dbscan.labels_)),set(er
```

```
Estimated number of clusters: 3 {0, 1, -1}
```

```
In [173…  sns.scatterplot(data=temp,x='CustAccountBalance',y='TransactionAmount (INR)',h
```

```
Out[173…  <AxesSubplot:xlabel='CustAccountBalance', ylabel='TransactionAmount (INR)'>
```



In [174…

```
print("Silhouette Score:",str(np.round(silhouette_score(temp,enhanced_dbscan.l
```

```
Silhouette Score: 39.89%
```

The optimized DBSCAN model has classified the entire population of customers primarily into two major groups, one of them consists of all those customers who have modest account balance and make low-value transactions whereas the miscellaneous group includes either the customers who have high account balances and spend very less money through transactions or those who have minimal account balance but expend a large amount of cash through high-value transactions.

Principal Component Analysis(PCA)

In [152…

```
temp = scaled_df.sample(10000)
pca = PCA(n_components=2)
pca_components = pca.fit_transform(temp)
pca_components = pd.DataFrame(pca_components,columns=['PC1','PC2'])
pca_components.head()
```

```
Out[152…
```

	PC1	PC2
0	-0.44	1.44
1	-0.24	-0.34
2	-0.37	0.75
3	-0.71	0.10
4	0.17	-0.96

```
In [153…  base_spectral = SpectralClustering(random_state=101)
```

```
In [154…  base_spectral.fit(pca_components)
```

```
Out[154…  SpectralClustering(random_state=101)
```

```
In [155…  print("Estimated number of clusters:",len(set(base_spectral.labels_)))
          print("Clusters:",set(base_spectral.labels_))
```

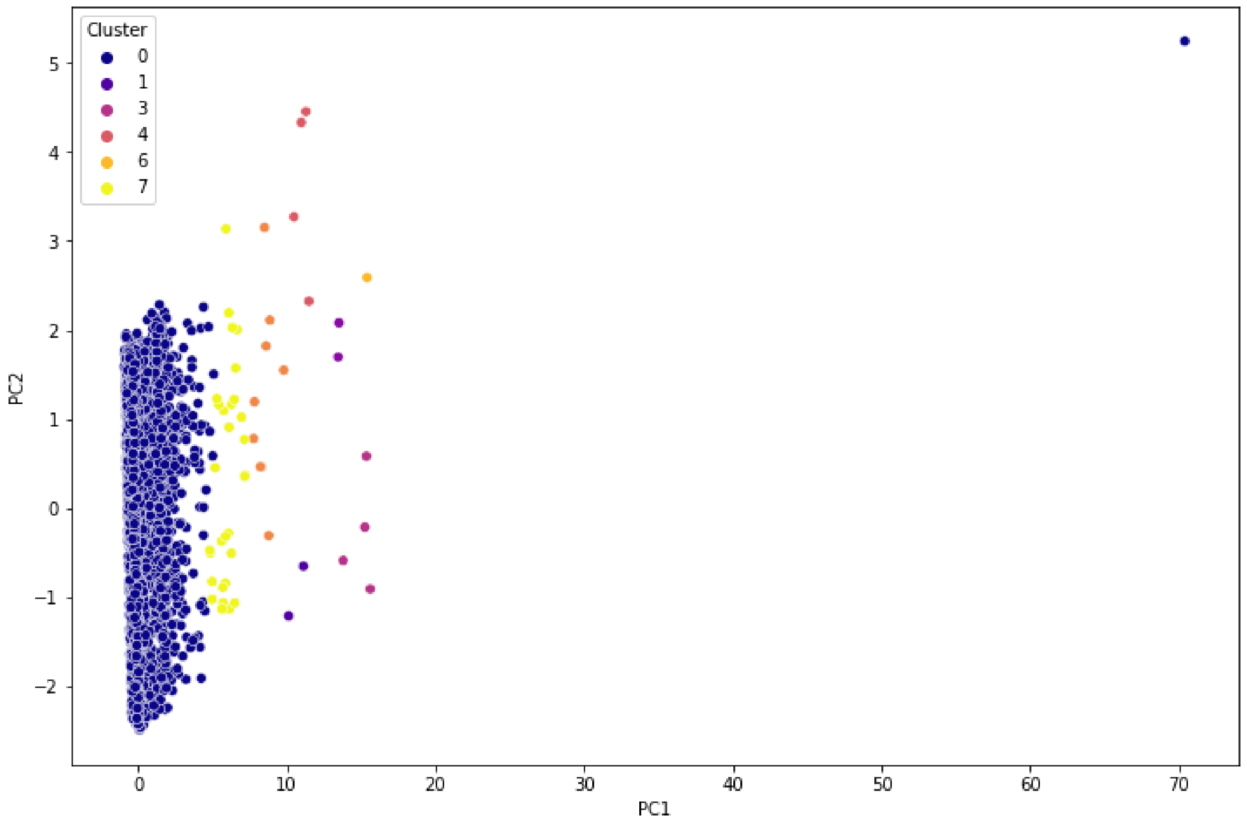
```
Estimated number of clusters: 8
Clusters: {0, 1, 2, 3, 4, 5, 6, 7}
```

```
In [156…  pca_components['Cluster'] = base_spectral.labels_
          print("Silhouette Score:",str(np.round(silhouette_score(pca_components,pca_com
```

```
Silhouette Score: 83.31%
```

```
In [157…  sns.scatterplot(pca_components['PC1'],pca_components['PC2'],hue=pca_components
```

```
Out[157…  <AxesSubplot:xlabel='PC1', ylabel='PC2'>
```

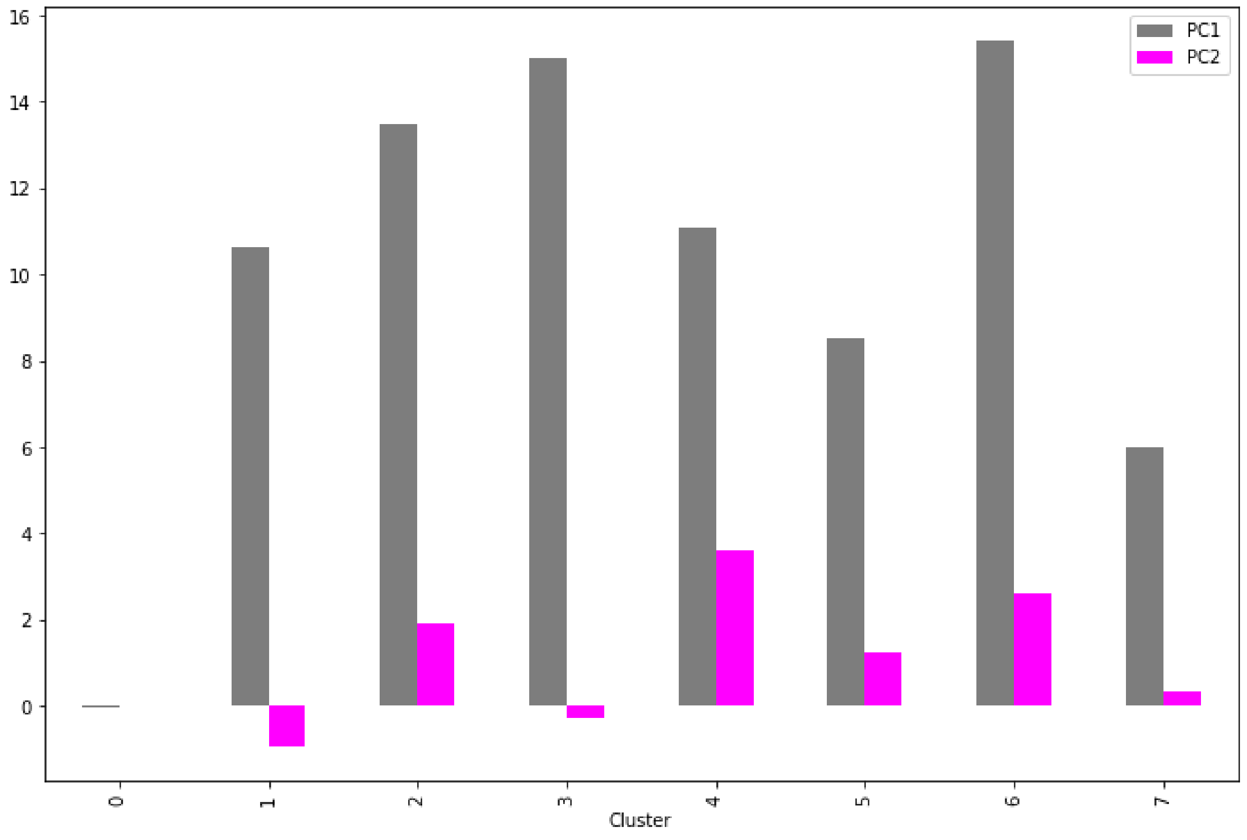


In [158…

```
pca_components.groupby('Cluster')[['PC1','PC2']].mean().plot(kind='bar',color=
```

Out[158…

```
<AxesSubplot:xlabel='Cluster'>
```



On Original Feature Space

```
In [118…  temp = scaled_df.sample(n=10000)
```

```
In [119…  base_spectral = SpectralClustering(random_state=101)
          base_spectral.fit(temp)
```

```
Out[119…  SpectralClustering(random_state=101)
```

```
In [120…  print("Estimated number of clusters:",set(base_spectral.labels_))
```

```
Estimated number of clusters: {0, 1}
```

```
In [121…  print("Estimated number of noise points:",list(base_spectral.labels_).count(-1
```

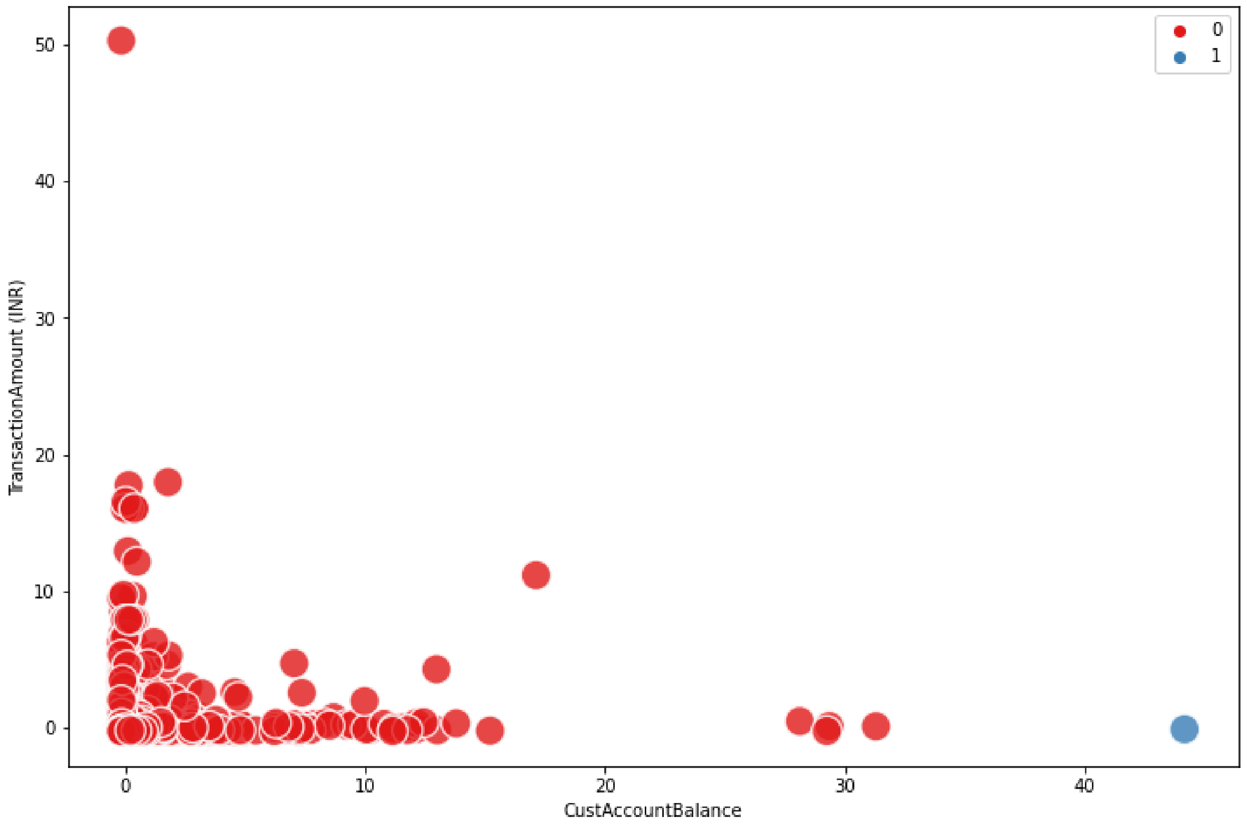
```
Estimated number of noise points: 0
```

```
In [122…  print("Silhouette Score:",str(np.round(silhouette_score(temp,base_spectral.lab
```

```
Silhouette Score: 92.08%
```

```
In [123…  sns.scatterplot(data=temp,x='CustAccountBalance',y='TransactionAmount (INR)',h
```

```
Out[123…  <AxesSubplot:xlabel='CustAccountBalance', ylabel='TransactionAmount (INR)'>
```



The Spectral clustering algorithm has segregated the customers into two distinct groups. The first group comprises the dynamic customers who have lower account balance and mostly expend less cash on transactions barring a few who perform large-value transactions whereas the second group includes the more conservative and money saving-minded people who, inspite of having really high account balances, spend the least amount of money in transactions, thereby judiciously maintaining their savings accounts.