Creating Customer Segments by RFM score Using Clustering

Kanja Saha,12/22/ 2017

Steps

1. Import Libraries
2. Import Data
3. Preprocess Data
4. Explore Data
5. Implement Algorithms

Import Libraries

In general, import all libraries before importing data. However, for learning purpose, import libraries in each step as needed. This is will give a better understanding of the libraries and their functions.

In [1]:
```
# Import libraries necessary for this project
#For all data processing functions
#import panda as pd

#For all numerical processing
#import numpy as np

#from IPython.display import display # Allows the use of display() for DataFra
mes
#import matplotlib.pyplot as plt

# Import supplementary visualizations code visuals.py
#import visuals as vs
```

Import Data

In general, import all needed libraries before importing data. If this is a learning execise, import libraries in each step as needed. This is will give a better understanding of the libraries.

In [2]:
```
#import necessary libraries
import pandas as pd
from IPython.display import display

# Load the dataset into pandas dataframe
raw_data = pd.read_excel("Online_Retail.xlsx")
print ("Dataset has {} rows(samples) with {} columns(features) each.".format(*
raw_data.shape))

# display the top 5 rows of the dataset
raw_data.head(5)
```

```
Dataset has 541909 rows(samples) with 8 columns(features) each.
```

Out[2]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Co
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	Unit King
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	Unit King
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	Unit King
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	Unit King
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	Unit King

In [3]:
```
#summary of dataset's distribution
raw_data.describe()
```

Out[3]:

	Quantity	UnitPrice	CustomerID
count	541909.000000	541909.000000	406829.000000
mean	9.552250	4.611114	15287.690570
std	218.081158	96.759853	1713.600303
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13953.000000
50%	3.000000	2.080000	15152.000000
75%	10.000000	4.130000	16791.000000
max	80995.000000	38970.000000	18287.000000

In [4]:
```
#items with negative quantity implies returned items, and 0 implies no purchas
e
raw_data[raw_data.Quantity<=0].head(5)
```

Out[4]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
141	C536379	D	Discount	-1	2010-12-01 09:41:00	27.50	14527.0
154	C536383	35004C	SET OF 3 COLOURED FLYING DUCKS	-1	2010-12-01 09:49:00	4.65	15311.0
235	C536391	22556	PLASTERS IN TIN CIRCUS PARADE	-12	2010-12-01 10:24:00	1.65	17548.0
236	C536391	21984	PACK OF 12 PINK PAISLEY TISSUES	-24	2010-12-01 10:24:00	0.29	17548.0
237	C536391	21983	PACK OF 12 BLUE PAISLEY TISSUES	-24	2010-12-01 10:24:00	0.29	17548.0

In [5]:
```
from datetime import datetime
#calculate recency for each transaction
currentDT = pd.to_datetime(datetime.now().date())
raw_data['Recency']= (pd.to_datetime(datetime.now().date())-pd.to_datetime(raw
_data.InvoiceDate)).dt.days

# store the returned items in a dataset
returned_invoices=raw_data[raw_data.Quantity<0]
returned_invoices['Quantity']=returned_invoices['Quantity']*-1

# remove transactions that has negative or 0 quantity or with unit price
raw_data=raw_data[(raw_data.Quantity>0) & (raw_data.UnitPrice>0)]

# get the list of items that are purchased and then returned
keys = ['CustomerID', 'StockCode','Quantity']
i1 = raw_data.set_index(keys).index
i2 = returned_invoices.set_index(keys).index
raw_data_filtered=raw_data[~i1.isin(i2)]
```

```
C:\Users\kanja\Anaconda3\envs\py36\lib\site-packages\ipykernel_launcher.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
```

In [6]:
```
import numpy as np

#Calculate revenue(monetization) generated by each customer
def total_price(raw_data_filtered):
    x = (raw_data_filtered.Quantity * raw_data_filtered.UnitPrice)
    return np.sum(x)

data=pd.DataFrame()
data['monetization']=raw_data_filtered.groupby('CustomerID').apply(total_price
)
#get the recency of the account
data['recency']=raw_data.groupby('CustomerID').agg({'Recency': np.min})
#get the total number of transactions
data['frequency']=raw_data.groupby('CustomerID').Recency.nunique()

#save customer_id which is an index in the data frame
CustomerID=data.index
# show summary of the data distribution
data.describe()
```

Out[6]:

	monetization	recency	frequency
count	4324.000000	4324.000000	4324.000000
mean	1924.122274	2493.968085	3.873265
std	8228.468418	100.039894	5.956480
min	2.900000	2402.000000	1.000000
25%	305.475000	2419.000000	1.000000
50%	663.630000	2452.000000	2.000000
75%	1624.420000	2543.000000	4.000000
max	278528.420000	2775.000000	132.000000

Preprocess Data

In [7]:
```
from sklearn import preprocessing

print('Is there null value in the data frame? {}.' .format('Yes' if (data.isnu
ll().values.any()) else 'No' ))
if (data.isnull().values.any()):
    print(data.isnull().sum())

#normalize all the columns(features) so that all the values in the column lie
 between 0 and 1
#this way each features will get equal preference regardless of their actual r
ange
n_data=pd.DataFrame()
n_data = pd.DataFrame(preprocessing.normalize(data),columns=data.columns)
n_data.head(5)
```

```
Is there null value in the data frame? No.
```

Out[7]:

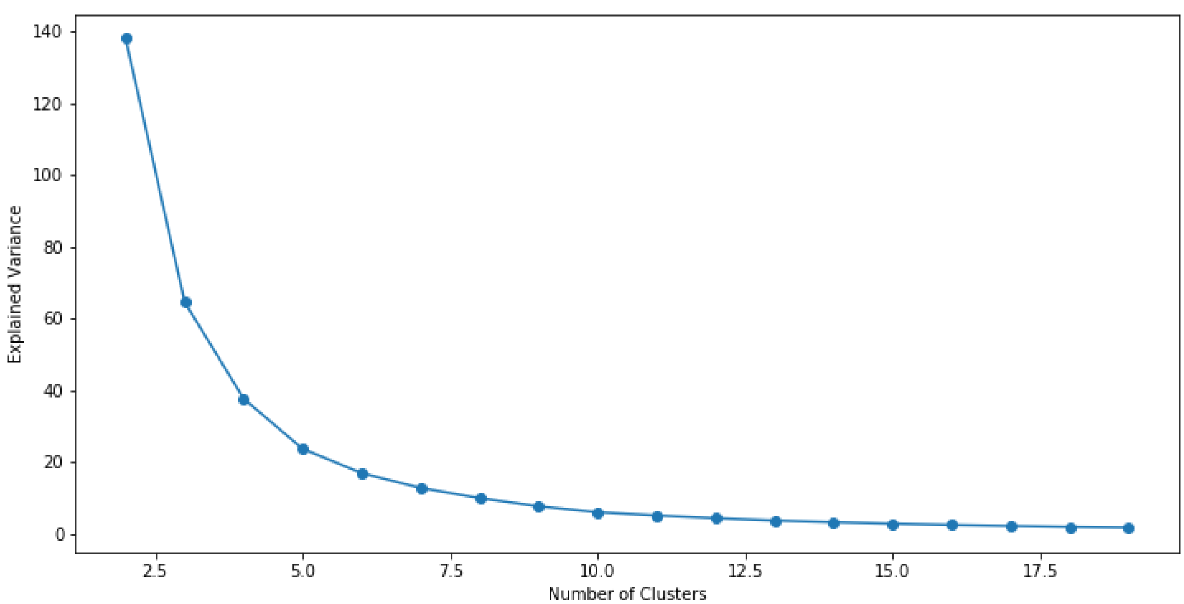
	monetization	recency	frequency
0	0.873333	0.487121	0.001418
1	0.587270	0.809390	0.001307
2	0.587636	0.809126	0.000334
3	0.122377	0.992484	0.000366
4	0.499385	0.866376	0.002488

Explore Data

In [8]:
```
#Elbow method & silhouette_score
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
# Pretty display for notebooks
%matplotlib inline

cluster_range = range( 2, 20 )
cluster_errors = []

for num_clusters in cluster_range:
    clusters = KMeans(num_clusters)
    clusters.fit(n_data)
    preds = clusters.predict(n_data)
    cluster_errors.append( clusters.inertia_ )
    score = silhouette_score(data, preds, metric='euclidean')
    print ("For K-means n_clusters = {}. The average silhouette_score is : {}"
.format(num_clusters, score))

clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "cluster_errors":
cluster_errors } )

plt.figure(figsize=(12,6))
plt.ylabel('Explained Variance')
plt.xlabel('Number of Clusters')
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors, marker = "o" )
```

```
For K-means n_clusters = 2. The average silhouette_score is : 0.5898505014145
282
For K-means n_clusters = 3. The average silhouette_score is : 0.4828100242967
8055
For K-means n_clusters = 4. The average silhouette_score is : 0.4654911613248
501
For K-means n_clusters = 5. The average silhouette_score is : 0.4531849302091
061
For K-means n_clusters = 6. The average silhouette_score is : 0.4327111002623
7675
For K-means n_clusters = 7. The average silhouette_score is : 0.4240064158466
9824
For K-means n_clusters = 8. The average silhouette_score is : 0.4201046290238
078
For K-means n_clusters = 9. The average silhouette_score is : 0.3846998789804
1297
For K-means n_clusters = 10. The average silhouette_score is : 0.388102440660
8065
For K-means n_clusters = 11. The average silhouette_score is : 0.358432650358
64397
For K-means n_clusters = 12. The average silhouette_score is : 0.359343803481
83653
For K-means n_clusters = 13. The average silhouette_score is : 0.359339297782
39027
For K-means n_clusters = 14. The average silhouette_score is : 0.348634587252
3635
For K-means n_clusters = 15. The average silhouette_score is : 0.330701837747
95064
For K-means n_clusters = 16. The average silhouette_score is : 0.310689520053
4717
For K-means n_clusters = 17. The average silhouette_score is : 0.313485826037
69914
For K-means n_clusters = 18. The average silhouette_score is : 0.302180640784
12055
For K-means n_clusters = 19. The average silhouette_score is : 0.286872762192
7523
```

Out[8]: [<matplotlib.lines.Line2D at 0x1b06a378518>]



Implement Algorithms

In [9]:
```
#Select a few observations to sample from the dataset
indices = [4000,3000,2,1400,1111]

# Create a DataFrame of the chosen samples
samples = pd.DataFrame(n_data.loc[indices], columns = n_data.keys()).reset_ind
ex(drop = True)
samples_z = pd.DataFrame(n_data.loc[indices], columns = n_data.keys()).reset_i
ndex(drop = True)
print ("Chosen samples of dataset:")

#calculate the z score to understand the location of the feauture for each cus
tomer in density plot
for col in n_data.columns:
    col_zscore = col + '_zscore'
    samples_z[col_zscore] = (samples_z[col] - data[col].mean())/data[col].std
()
samples_z
```

```
Chosen samples of dataset:
```

Out[9]:

	monetization	recency	frequency	monetization_zscore	recency_zscore	frequency_z
0	0.020554	0.999789	0.000399	-0.233835	-24.919741	-0.650194
1	0.072459	0.997371	0.000384	-0.233828	-24.919766	-0.650196
2	0.587636	0.809126	0.000334	-0.233766	-24.921647	-0.650205
3	0.982372	0.186938	0.000620	-0.233718	-24.927867	-0.650157
4	0.674233	0.738514	0.002441	-0.233755	-24.922353	-0.649851

In [10]:
```
#n_data.drop(['CustomerID'], axis = 1, inplace = True)
#n_data.drop(['cluster'], axis = 1, inplace = True)
```

In [11]:
```
#Find cluster centers and size using Kmeans
# Loop through clusters
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
range_n_clusters = range(5,6)
for n_clusters in range_n_clusters:
    #Apply your clustering algorithm of choice to the reduced data
    clusterer = KMeans(n_clusters=n_clusters).fit(n_data)

    #Predict the cluster for each data point
    preds = clusterer.predict(n_data)

    #Predict the cluster for each transformed sample data point
    sample_preds = clusterer.predict(samples)

    #cluster centers
    centers = clusterer.cluster_centers_
    score=silhouette_score(n_data, preds, metric='euclidean')

    #centroids of each cluster
    centers_df = pd.DataFrame(centers)
    centers_df.columns=data.columns

    cluster_size=np.bincount(preds)
    centers_df.insert(loc=0, column='cluster_size', value=cluster_size)
    centers_df_int64=centers_df#.astype('int64')

    print ("For n_clusters = {}".format(n_clusters))
    print("Silhouette Coefficient: %0.3f" % score )

    display(centers_df_int64)
```

```
For n_clusters = 5
Silhouette Coefficient: 0.591
```

	cluster_size	monetization	recency	frequency
0	486	0.776287	0.622749	0.001787
1	1058	0.302308	0.950841	0.001054
2	659	0.543670	0.835189	0.001539
3	298	0.947510	0.291997	0.001593
4	1823	0.104288	0.993267	0.000548

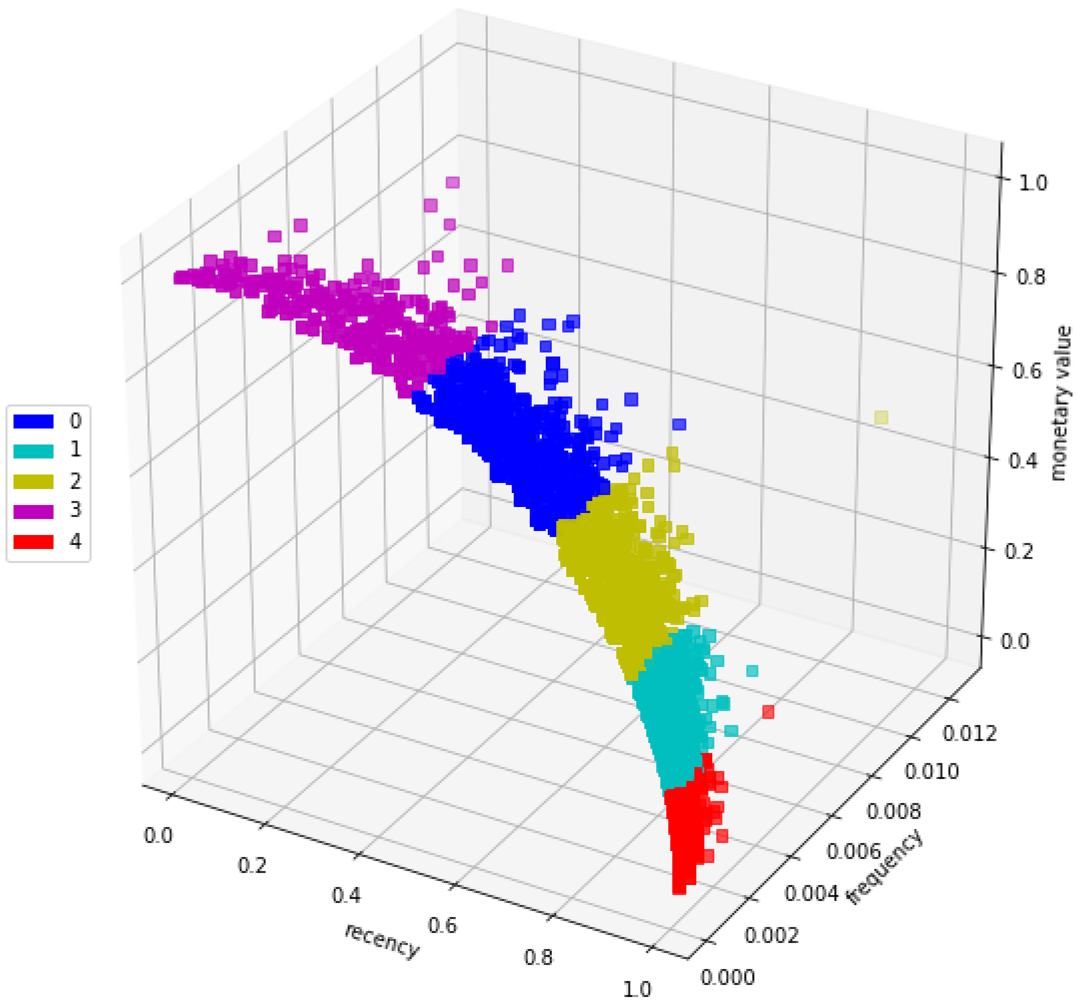
In [12]:
```
n_data['customer_id']=CustomerID
n_data['cluster'] = pd.Series(preds, index=n_data.index)
n_data.to_csv('normalizeddata_withcluster_rfm.csv', index=False)
display(n_data.head(5))


data['customer_id']=CustomerID
data['cluster'] = pd.Series(preds, index=data.index)

df=pd.merge(data, n_data, how='inner', on='customer_id', left_on=None, right_o
n=None,
         left_index=False, right_index=False, sort=True,
         suffixes=('_x', '_y'), copy=True, indicator=False
         )
df.to_csv('data_withcluster_rfm.csv', index=False)
df.head(5)
```

	monetization	recency	frequency	customer_id	cluster
0	0.873333	0.487121	0.001418	12347.0	0
1	0.587270	0.809390	0.001307	12348.0	2
2	0.587636	0.809126	0.000334	12349.0	2
3	0.122377	0.992484	0.000366	12350.0	4
4	0.499385	0.866376	0.002488	12352.0	2

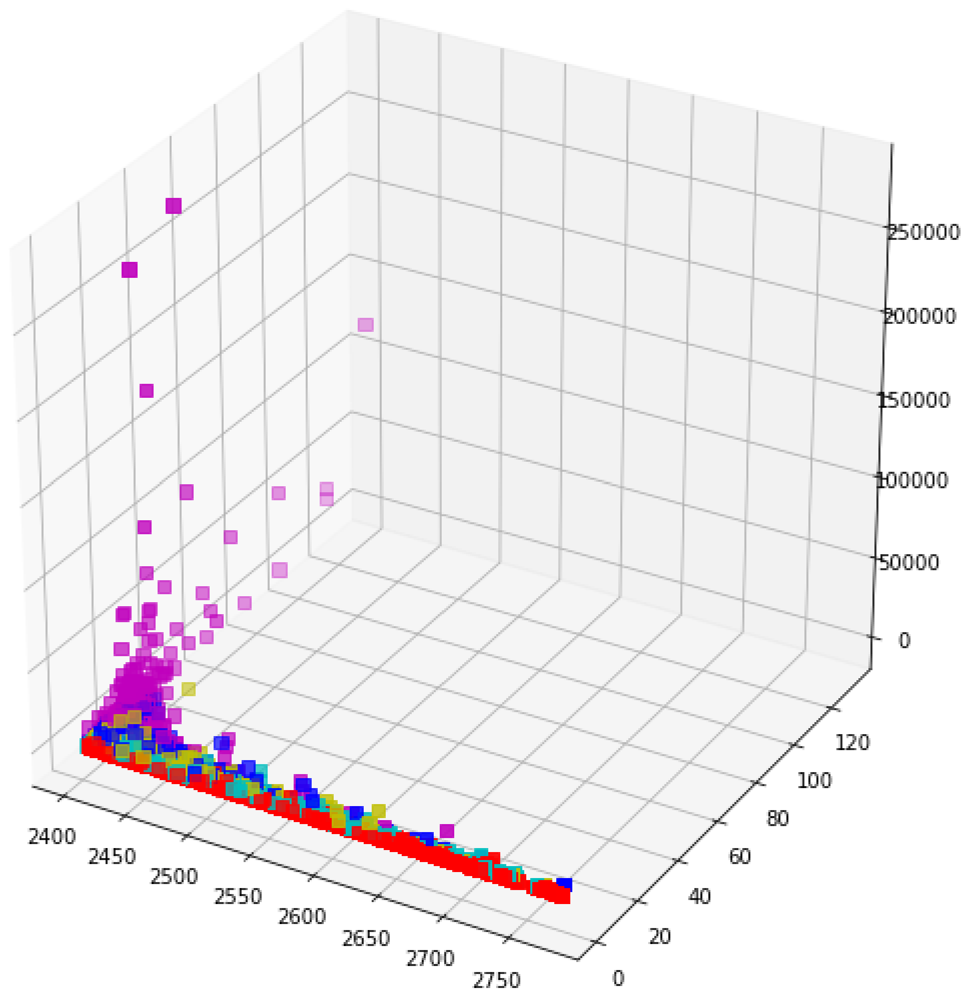
Out[12]:

	monetization_x	recency_x	frequency_x	customer_id	cluster_x	monetization_y	rece
0	4310.00	2404	7	12347.0	0	0.873333	0.48
1	1797.24	2477	4	12348.0	2	0.587270	0.80
2	1757.55	2420	1	12349.0	2	0.587636	0.80
3	334.40	2712	1	12350.0	4	0.122377	0.99
4	1405.28	2438	7	12352.0	2	0.499385	0.86

In [13]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
import numpy as np
from matplotlib.colors import LinearSegmentedColormap


fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, xlabel='recency',ylabel='frequency',zlabel='monetary
 value',projection='3d')
x = np.array(n_data['recency'])
y = np.array(n_data['frequency'])
z = np.array(n_data['monetization'])

colors=['b', 'c', 'y', 'm', 'r']
custom_cmap = LinearSegmentedColormap.from_list("my_colormap", colors)

clusters=[0,1,2,3,4]

# marker="s" for square, s=30 is for size of the square
ax.scatter(x,y,z, marker="s", c=n_data["cluster"], s=30, cmap=custom_cmap)
import matplotlib.patches as mpatches

recs = []
for i in range(0,len(colors)):
    recs.append(mpatches.Rectangle((0,0),1,1,fc=colors[i]))
plt.legend(recs,clusters,loc=6)

plt.show()
```


0
1
2
3
4
recency
frequency
monetary value
0.0
0.2
0.4
0.6
0.8
1.0
0.000
0.002
0.004
0.006
0.008
0.010
0.012

In [14]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
import numpy as np


fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')
x = np.array(data['recency'])
y = np.array(data['frequency'])
z = np.array(data['monetization'])

ax.scatter(x,y,z, marker="s", c=data["cluster"], s=40, cmap=custom_cmap)

plt.show()
```



In [15]:
```
#Find cluster centers and size using GaussianMixture
# Loop through clusters
from sklearn.metrics import silhouette_score
from sklearn.mixture import GaussianMixture

if 'customer_id' in n_data.columns:
    n_data.drop(['customer_id'], axis = 1, inplace = True)
if 'cluster' in n_data.columns:
    n_data.drop(['cluster'], axis = 1, inplace = True)

range_n_clusters = range(2,10)
for n in range_n_clusters:
    #Apply your clustering algorithm of choice to the reduced data
    clusterer = GaussianMixture(n_components=n).fit(n_data)

    #Predict the cluster for each data point
    preds = clusterer.predict(n_data)

    #Predict the cluster for each transformed sample data point
    sample_preds = clusterer.predict(samples)

    #cluster centers
    centers = clusterer.means_
    score = silhouette_score(n_data, preds, metric='mahalanobis')

    #centroids of each cluster
    centers_df = pd.DataFrame(centers)


    cluster_size=np.bincount(preds)
    centers_df.insert(loc=0, column='cluster_size', value=cluster_size)
    centers_df_int64=centers_df#.astype('int64')

    print ("For GMM with n_clusters = {}. The average silhouette_score is : {}
".format(n, score))
    display(centers_df_int64)
    #Display the predictions
    for i, pred in enumerate(sample_preds):
        print ("Sample point", i, "predicted to be in Cluster", pred)
```

```
For GMM with n_clusters = 2. The average silhouette_score is : 0.394642546398
23076
```

	cluster_size	0	1	2
0	1811	0.638615	0.707206	0.001552
1	2513	0.147678	0.985305	0.000660

```
Sample point 0 predicted to be in Cluster 1
Sample point 1 predicted to be in Cluster 1
Sample point 2 predicted to be in Cluster 0
Sample point 3 predicted to be in Cluster 0
Sample point 4 predicted to be in Cluster 0
For GMM with n_clusters = 3. The average silhouette_score is : 0.327332213852
85916
```

	cluster_size	0	1	2
0	1919	0.109815	0.992436	0.000562
1	1094	0.774514	0.581494	0.001703
2	1311	0.357744	0.928162	0.001165

```
Sample point 0 predicted to be in Cluster 0
Sample point 1 predicted to be in Cluster 0
Sample point 2 predicted to be in Cluster 1
Sample point 3 predicted to be in Cluster 1
Sample point 4 predicted to be in Cluster 1
For GMM with n_clusters = 4. The average silhouette_score is : 0.320551463734
3055
```

	cluster_size	0	1	2
0	1096	0.298934	0.951748	0.001040
1	638	0.873075	0.448000	0.001689
2	803	0.573427	0.811920	0.001595
3	1787	0.102225	0.993554	0.000544

```
Sample point 0 predicted to be in Cluster 3
Sample point 1 predicted to be in Cluster 3
Sample point 2 predicted to be in Cluster 2
Sample point 3 predicted to be in Cluster 1
Sample point 4 predicted to be in Cluster 2
For GMM with n_clusters = 5. The average silhouette_score is : 0.331821803528
13645
```

	cluster_size	0	1	2
0	482	0.769045	0.632002	0.001773
1	1075	0.294663	0.953275	0.001029
2	674	0.535875	0.840108	0.001533
3	316	0.943201	0.303478	0.001621
4	1777	0.101901	0.993600	0.000543

```
Sample point 0 predicted to be in Cluster 4
Sample point 1 predicted to be in Cluster 4
Sample point 2 predicted to be in Cluster 2
Sample point 3 predicted to be in Cluster 3
Sample point 4 predicted to be in Cluster 0
For GMM with n_clusters = 6. The average silhouette_score is : 0.292953809554
6279
```

	cluster_size	0	1	2
0	893	0.249165	0.967329	0.000919
1	307	0.945638	0.297042	0.001607
2	405	0.790824	0.606421	0.001781
3	447	0.610085	0.789586	0.001656
4	1646	0.094661	0.994555	0.000525
5	626	0.424102	0.903598	0.001323

```
Sample point 0 predicted to be in Cluster 4
Sample point 1 predicted to be in Cluster 4
Sample point 2 predicted to be in Cluster 3
Sample point 3 predicted to be in Cluster 1
Sample point 4 predicted to be in Cluster 3
For GMM with n_clusters = 7. The average silhouette_score is : 0.291490406504
9743
```

	cluster_size	0	1	2
0	143	0.978918	0.183445	0.001369
1	905	0.245142	0.968366	0.000907
2	446	0.594085	0.801896	0.001638
3	1616	0.093403	0.994709	0.000523
4	606	0.415565	0.907740	0.001308
5	271	0.891876	0.445302	0.001860
6	337	0.758993	0.647765	0.001724

```
Sample point 0 predicted to be in Cluster 3
Sample point 1 predicted to be in Cluster 3
Sample point 2 predicted to be in Cluster 2
Sample point 3 predicted to be in Cluster 0
Sample point 4 predicted to be in Cluster 2
For GMM with n_clusters = 8. The average silhouette_score is : 0.262648119260
8048
```

	cluster_size	0	1	2
0	579	0.355997	0.933360	0.001162
1	127	0.982233	0.168221	0.001278
2	1457	0.085811	0.995581	0.000506
3	363	0.663026	0.746172	0.001744
4	301	0.805760	0.588769	0.001750
5	865	0.215021	0.975726	0.000834
6	437	0.509083	0.859209	0.001493
7	195	0.915805	0.396843	0.001847

```
Sample point 0 predicted to be in Cluster 2
Sample point 1 predicted to be in Cluster 2
Sample point 2 predicted to be in Cluster 6
Sample point 3 predicted to be in Cluster 1
Sample point 4 predicted to be in Cluster 3
For GMM with n_clusters = 9. The average silhouette_score is : 0.228974117992
4036
```

	cluster_size	0	1	2
0	686	0.268523	0.962535	0.000972
1	256	0.820462	0.568929	0.001757
2	967	0.064880	0.997550	0.000475
3	486	0.404402	0.913471	0.001289
4	185	0.919021	0.389809	0.001856
5	323	0.695316	0.716427	0.001779
6	126	0.982656	0.166173	0.001269
7	402	0.549830	0.833649	0.001543
8	893	0.152435	0.987840	0.000641

```
Sample point 0 predicted to be in Cluster 2
Sample point 1 predicted to be in Cluster 2
Sample point 2 predicted to be in Cluster 7
Sample point 3 predicted to be in Cluster 6
Sample point 4 predicted to be in Cluster 5
```

In [16]:
```
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
from sklearn import metrics


if 'customer_id' in n_data.columns:
    n_data.drop(['customer_id'], axis = 1, inplace = True)
if 'cluster' in n_data.columns:
    n_data.drop(['cluster'], axis = 1, inplace = True)


clusterer_DB = DBSCAN(eps=.01, metric='euclidean', min_samples=1).fit(n_data)
labels_DB = clusterer_DB.labels_

#display(labels_DB)
display(len(set(labels_DB)))

score_DB=silhouette_score(n_data, labels_DB, metric='euclidean')

print("Silhouette Coefficient: %0.3f" % score_DB )

n_data['cluster']=labels_DB
n_data['customer_id']=CustomerID
n_data.head(10)

n_data[["customer_id", "cluster"]].to_csv('finaldatawithcluster_DB_monthly_frequency', index=False)
```

```
5

Silhouette Coefficient: -0.151
```

In [17]:
```
from sklearn.cluster import AffinityPropagation
from sklearn import metrics

if 'customer_id' in n_data.columns:
    n_data.drop(['customer_id'], axis = 1, inplace = True)
if 'cluster' in n_data.columns:
    n_data.drop(['cluster'], axis = 1, inplace = True)


# Compute Affinity Propagation
af = AffinityPropagation().fit(n_data)
cluster_centers_indices = af.cluster_centers_indices_
labels = af.labels_

n_clusters_ = len(cluster_centers_indices)


import matplotlib.pyplot as plt
from itertools import cycle

print(n_clusters)
n_data['cluster']=labels
n_data['customer_id']=CustomerID
n_data.head(10)

n_data[["customer_id", "cluster"]].to_csv('finaldatawithcluster_AP_monthly_fre
quency', index=False)
```

```
5
```

In [18]:
```
#AgglomerativeClustering

if 'customer_id' in n_data.columns:
    n_data.drop(['customer_id'], axis = 1, inplace = True)
if 'cluster' in n_data.columns:
    n_data.drop(['cluster'], axis = 1, inplace = True)


from sklearn.cluster import AgglomerativeClustering
# Affinity = {"“euclidean", “l1”, “l2”, “manhattan”,
# “cosine”}
# Linkage = {"ward"}#, “complete”, “average”}
Hclustering = AgglomerativeClustering(n_clusters=5, affinity='euclidean', link
age='ward')
Hclustering.fit_predict(n_data)
#ms = np.column_stack((ground_truth,Hclustering.labels_))
#df = pd.DataFrame(ms,
# columns = [‘Ground truth’,’Clusters’])
#pd.crosstab(df[‘Ground truth’], df[‘Clusters’],
# margins=True)

labels_HC = Hclustering.labels_

display(labels_HC)
len(set(labels_HC))

n_data['cluster']=labels_HC
n_data['customer_id']=CustomerID.astype('Int64')
display(n_data.head(10))
#true_centers_Kmeans_df_int64['cluster'] = pd.Series(true_centers_Kmeans_df_in
t64.index, index=true_centers_Kmeans_df_int64.index)
n_data[['customer_id','cluster']].to_csv('customerwithcluster_HC_monthly_frequ
ency.csv', index=False)
```

```
array([0, 2, 2, ..., 3, 2, 2], dtype=int64)
```

	monetization	recency	frequency	cluster	customer_id
0	0.873333	0.487121	0.001418	0	12347
1	0.587270	0.809390	0.001307	2	12348
2	0.587636	0.809126	0.000334	2	12349
3	0.122377	0.992484	0.000366	3	12350
4	0.499385	0.866376	0.002488	2	12352
5	0.034132	0.999417	0.000384	3	12353
6	0.379191	0.925318	0.000351	1	12354
7	0.172965	0.984928	0.000377	1	12355
8	0.757363	0.652994	0.000808	4	12356
9	0.930942	0.365168	0.000150	0	12357

In [19]:
```
#data.drop(['customer_id'], axis = 1, inplace = True)
#data.drop(['cluster'], axis = 1, inplace = True)

from scipy.spatial.distance import squareform, pdist
import seaborn as sns


distance_DB=pd.DataFrame(squareform(pdist(n_data.iloc[:, 1:])))

#display(distance_DB)
distance_DB.describe()

all_numbers=distance_DB.values.tolist()

flat_list = [item for sublist in all_numbers for item in sublist]

flat_list

np.histogram(flat_list)

n, bins, patches = plt.hist(flat_list)
print("n: ", n, sum(n))
print("bins: ", bins)
for i in range(len(bins)-1):
    print(bins[i+1] -bins[i])
print("patches: ", patches)
print(patches[1])

# sort the data:
sorted_flat_list = np.sort(flat_list)

# calculate the proportional values of samples
p = 1. * np.arange(len(flat_list)) / (len(flat_list) - 1)

# plot the sorted data:
fig = plt.figure()
ax1 = fig.add_subplot(121)
ax1.plot(p, sorted_flat_list)
ax1.set_xlabel('$p$')
ax1.set_ylabel('$x$')

ax2 = fig.add_subplot(122)
ax2.plot(sorted_flat_list, p)
ax2.set_xlabel('$x$')
ax2.set_ylabel('$p$')

sns.distplot(flat_list);
```

```
n:  [ 3537756.  3164962.  2806100.  2435148.  2066820.  1674352.  1304024.
   932666.   574456.   200692.] 18696976.0
bins:  [    0.           594.00003449  1188.00006898  1782.00010347  2376.000
13796
  2970.00017245  3564.00020694  4158.00024143  4752.00027592  5346.00031041
  5940.00034491]
594.000034491
594.000034491
594.000034491
594.000034491
594.000034491
594.000034491
594.000034491
594.000034491
594.000034491
594.000034491
patches:  <a list of 10 Patch objects>
Rectangle(594,0;594x3.16496e+06)
```