RFMT Segmentation Using K-Means Clustering

Yexi Yuan Sep 15, 2019 · 7 min read ★

Important note: This was created as part of my own personal learning process for data science in python. I find it extremely helpful when i write this down to help me learn better and faster. This was part of the course on DataCamp and the code is based on the course and other online resources I used. Please visit DataCamp for the original syllabus. I am not affiliated with DataCamp in any form and only use their resources as a learner.

On the final part of our customer segmentation journey we will be applying K-Means clustering method to segment our customer data. We will continue to use the features we've engineered in our RFM model. Additionally, we will be including **tenure** as a new feature for our model to create RFMT model.

What is K-Means?

K-Means is a popular and simple unsupervised machine learning algorithm. Put simply, the K-means algorithm identifies *k* number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. We will not dive into the specifics of how K-Means work so let us dive into the customer segmentation implementation.

K-Means Assumptions

These will be the 4 features that we are going to use to implement our K-Means model. Before we can fit the K-Means model to our data, we need to ensure that these key assumptions are fulfilled.

1. Distribution of variables
2. Variables with same average values
3. Variables with same variance

Looking at our RFMT summary statistics, we can see that our data set do not fulfill these key assumptions at the moment. Let's create a data pre-processing pipeline to prep our data to be K-Means ready.

Data Pre-Processing Pipeline

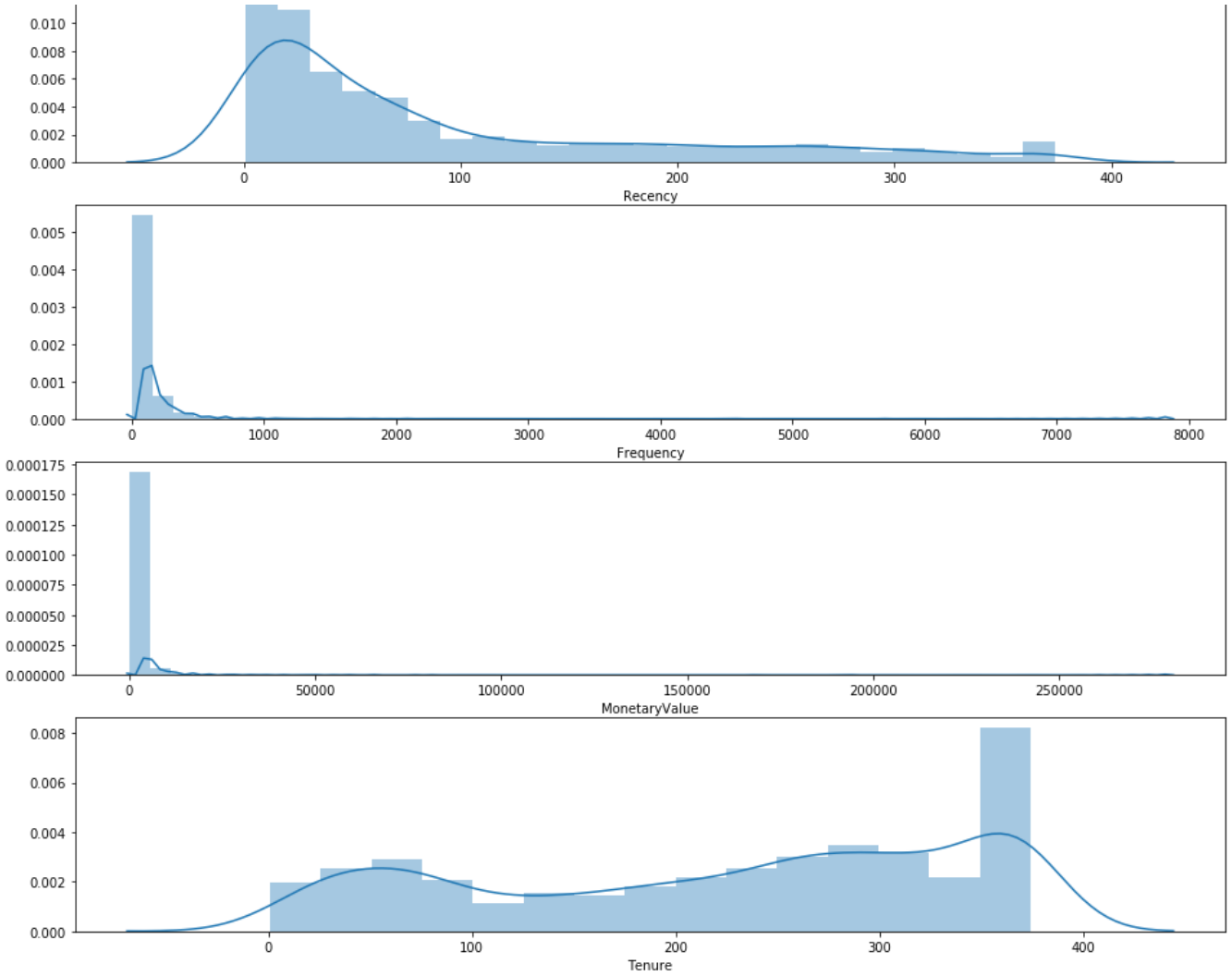
Below are the steps we need to take to perform our data transformation and dive into each step in details:

1. Unskew the data — We will be using log transformation to do this
2. Standardize to the same average values
3. Scale to the same standard deviation
4. Store as a separate array to be used for clustering

Data Skewness

Let's examine the shape of the distribution of our current data set.

```
# Plot RFM distributions
plt.figure(figsize=(16,14))

# Plot distribution of R
plt.subplot(4, 1, 1); sns.distplot(data_process['Recency'])

# Plot distribution of F
plt.subplot(4, 1, 2); sns.distplot(data_process['Frequency'])

# Plot distribution of M
plt.subplot(4, 1, 3); sns.distplot(data_process['MonetaryValue'])

# Plot distribution of T
plt.subplot(4, 1, 4); sns.distplot(data_process['Tenure'])

# Show the plot
plt.show()
```

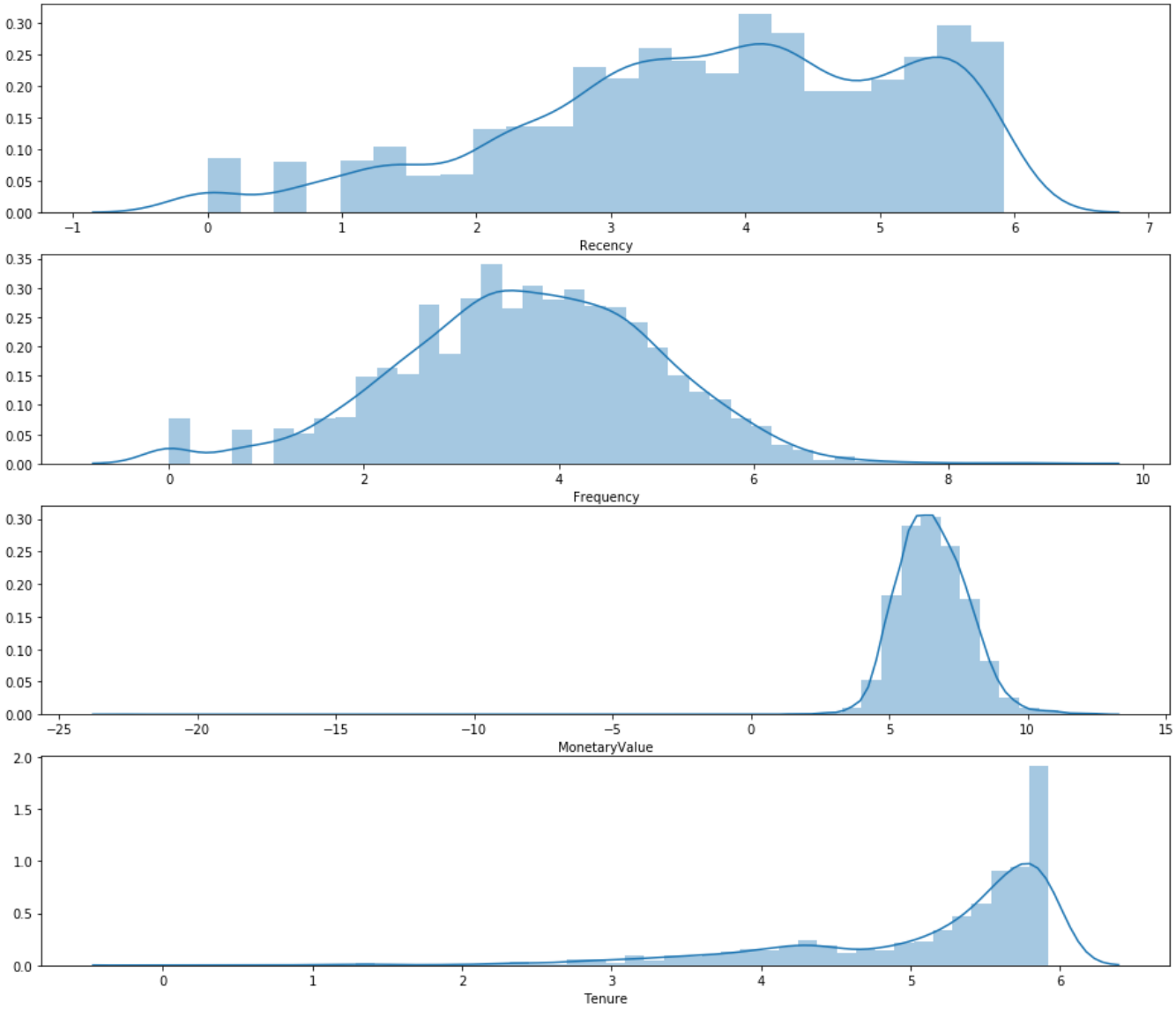


There is skewness in our RFMT data

As you can see, these is a general skewness of R, F, and M to the right. T has a more evenly distributed shape. To address this, we will apply Log Transformation to our data.

```
# Apply Log Transformation
data_process['MonetaryValue'] = data_process['MonetaryValue'] +
0.0000000001
recency_log = np.log(data_process['Recency'])
frequency_log = np.log(data_process['Frequency'])
monetary_log = np.log(data_process['MonetaryValue'])
tenure_log = np.log(data_process['Tenure'])

# Plot RFM distributions
plt.figure(figsize=(16,14))

# Plot distribution of R
plt.subplot(4, 1, 1); sns.distplot(recency_log)

# Plot distribution of F
plt.subplot(4, 1, 2); sns.distplot(frequency_log)

# Plot distribution of M
plt.subplot(4, 1, 3); sns.distplot(monetary_log)
```

```
# Plot distribution of M
plt.subplot(4, 1, 4); sns.distplot(tenure_log)

# Show the plot
plt.show()
```



After applying log transformation, we see a much more normally distributed data set

Great! Looks like we have normalized our RFM features. However, notice that by log transforming T, we've caused the data to left-skew. We have to always be careful when transforming data by examine through visualization or otherwise to ensure we create the most accurate representation of our customer segments.

Standardizing mean and standard deviation

	Recency	Frequency	MonetaryValue	Tenure
count	4339.000000	4339.000000	4339.000000	4339.000000
mean	92.518322	91.708689	2053.793018	223.259968
std	100.009747	228.792852	8988.248381	117.915703
min	1.000000	1.000000	0.000000	1.000000
25%	18.000000	17.000000	307.245000	113.000000
50%	51.000000	41.000000	674.450000	249.000000
75%	142.000000	100.000000	1661.640000	327.000000
max	374.000000	7847.000000	280206.020000	374.000000

In order to standard mean and standard deviation, we can use the following formula:

```
datamart_centered = datamart_rfm - datamart_rfm.mean()
datamart_scaled = datamart_rfm / datamart_rfm.std()
```

However, let's make use of SKLearn library's StandardScaler, to center and scale our data instead.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(data_process_log)
data_process_norm = scaler.transform(data_process_log)

data_process_norm_df = pd.DataFrame(data_process_norm)
data_process_norm_df.describe().round(2)
```

count	4339.00	4339.00	4339.00	4339.00
mean	-0.00	0.00	-0.00	0.00
std	1.00	1.00	1.00	1.00
min	-2.63	-2.77	-22.12	-5.68
25%	-0.61	-0.64	-0.64	-0.46
50%	0.11	0.03	-0.05	0.42
75%	0.83	0.70	0.62	0.72
max	1.51	3.99	4.45	0.86

Our feature set post-scaling has a mean of 0 and standard deviation of 1

Awesome! Now we have fulfilled the key assumptions in order for us to properly and accurately apply K-Means clustering to segment our customers.

Applying K-Means

Now let's get into the most exciting portion of any machine learning project — applying the ML model!

But before that, we need to choose the number of clusters for our K-Means model. This can be done in a few ways:

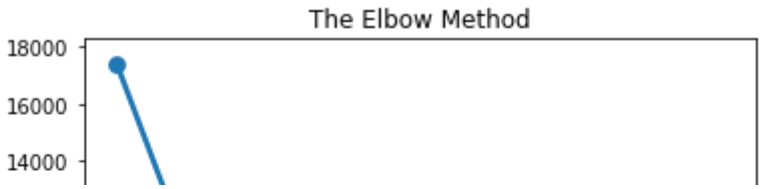
1. Visualization Method — The Elbow Criterion
2. Mathematical Method — Silhouette Coefficient
3. Experimentation and interpretation that makes the most business sense

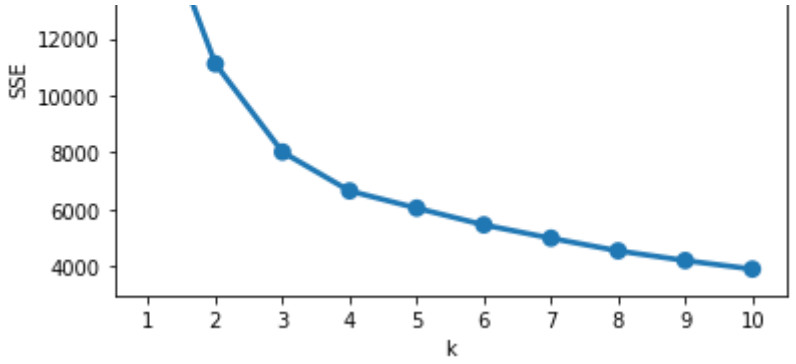
We will be using the elbow criterion method.

Plotting Number of Clusters Against Within-Cluster Sum-of-Squared-Errors (SSE)

The sum-of-squared-errors (SSE) is the sum of squared distances from every data point to their cluster center. We want to choose the optimal number of clusters that reduces the number SSE without over-fitting. Let's plot our elbow chart to determine that.

```
# Fit KMeans and calculate SSE for each *k*
sse = {}
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=1)
    kmeans.fit(data_process_norm)
    sse[k] = kmeans.inertia_

# Plot SSE for each *k*
plt.title('The Elbow Method')
plt.xlabel('k'); plt.ylabel('SSE')
sns.pointplot(x=list(sse.keys()), y=list(sse.values()))
plt.show()
```





SSE plot against number of clusters

Ideally, we want to choose the point on the elbow chart where the SSE stops decreasing at an increasing rate — i.e. the point where the change gradient of between the number of clusters becomes constant. For our model we will choose k=4.

Fitting our Data Set to K-Means

```
# Import KMeans from skLearn
from sklearn.cluster import KMeans

# Choose k=4 and fit data set to k-means model
kmeans = KMeans(n_clusters=4, random_state=1)
kmeans.fit(data_process_norm)

# Assign k-means labels to cluster labels
cluster_labels = kmeans.labels_

# Assign cluster labels to original pre-transformed data set
data_process_k4 = data_process.assign(Cluster = cluster_labels)

# Group data set by k-means cluster
data_process_k4.groupby(['Cluster']).agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'MonetaryValue': 'mean',
    'Tenure': ['mean', 'count']
}).round(0)
```

	Recency	Frequency	MonetaryValue	Tenure	
	mean	mean	mean	mean	count
Cluster					
0	11.0	276.0	6945.0	302.0	864
1	213.0	16.0	314.0	247.0	1153
2	34.0	34.0	492.0	49.0	878
3	80.0	78.0	1466.0	263.0	1444

Our RFMT data set grouped by their K-Means clusters

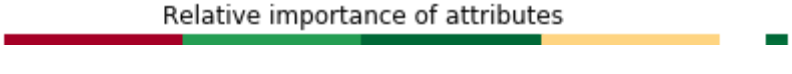
We can see that our grouped summary of the mean of R, F, M, and T that each cluster of customers places a different emphasis on our 4 features:

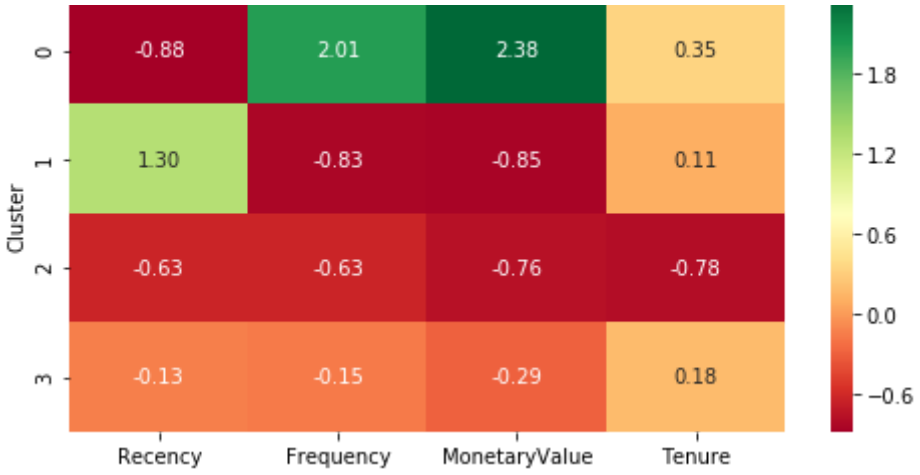
- **Cluster 0** has the highest MontaryValue mean and lowest Recency mean and the highest frequency mean — This is our ideal customer segment
- **Cluster 1** performs poorly across R, F, and M and has a long tenure in our database as well — we will need to design campaigns to activate them again
- **Cluster 2** shopped with us recently but have not spend as much or as frequently as we would like them to — perhaps some personalization of products targeted at them can help to maximize their lifetime-value?
- **Cluster 3** has spent quite a fair amount with us but has not shopped with us in the 3–4 months — We will need to do something before we lose them!

Relative Importance of RFMT among K-Means Clusters

We can visualize the relative importance of each feature for our 4 clusters through a heatmap.

```
# Calculate average RFM values for each cluster
cluster_avg = data_process_k4.groupby(['Cluster']).mean()

# Calculate average RFM values for the total customer population
population_avg = data_process.mean()

# Calculate relative importance of cluster's attribute value
compared to population
relative_imp = cluster_avg / population_avg - 1

# Initialize a plot with a figure size of 8 by 2 inches
plt.figure(figsize=(8, 4))

# Add the plot title
plt.title('Relative importance of attributes')

# Plot the heatmap
sns.heatmap(data=relative_imp, annot=True, fmt='.2f', cmap='RdYlGn')
plt.show()
```





Relative Importance of RFMT for each of our 3 clusters

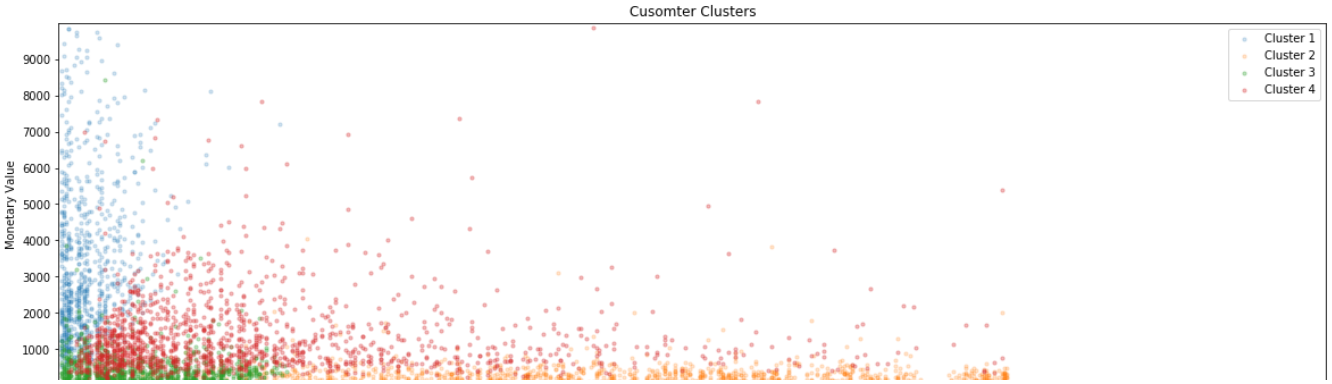
The heatmap provides a simple and easy way to understand how our K-Means model place relative importance of our RFMT attributes to assign each customer to their respective segments.

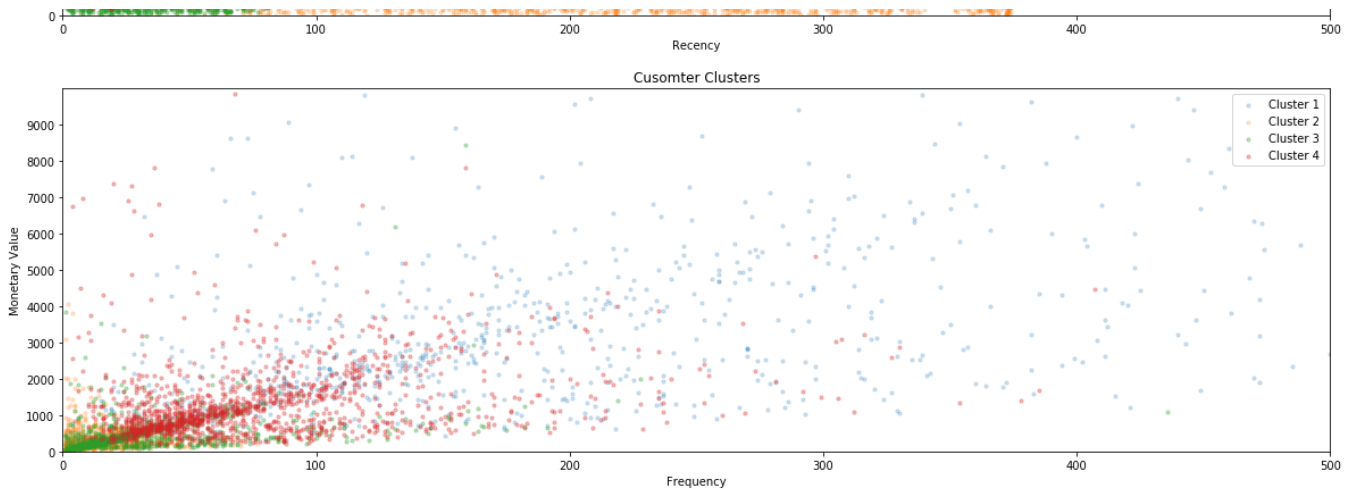
Visualizing Our K-Means Clusters

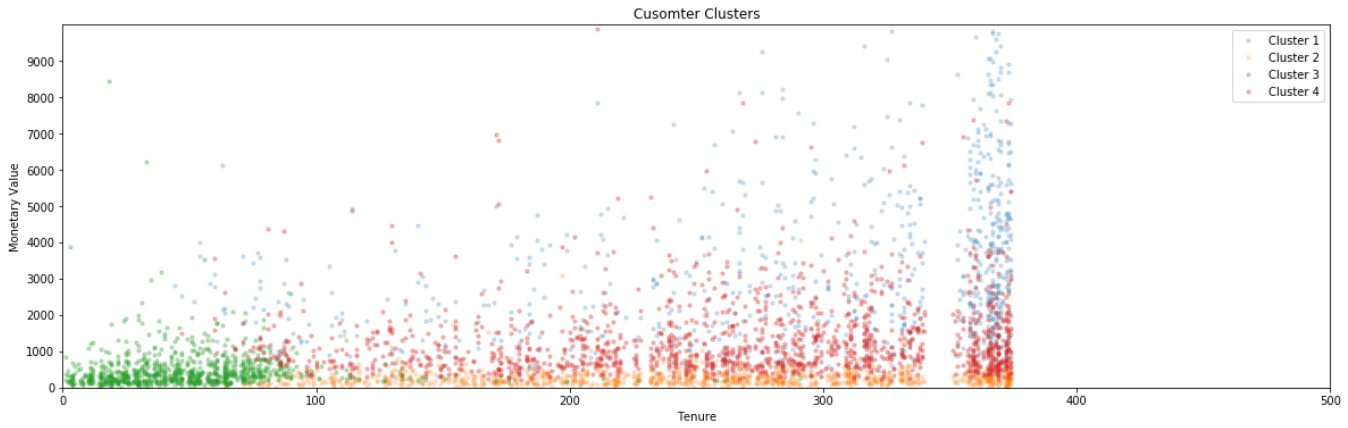
Lastly, let's visualize our K-Means Clusters in scatter plots to warp up our customer segmentation journey.

```
# Plot RFM distributions
plt.figure(figsize=(20,20))
plt.subplot(3, 1, 1);
plt.scatter(data_process_k4[cluster_labels == 0].loc[:,'Recency'],
data_process_k4[cluster_labels == 0].loc[:,'MonetaryValue'], s= 10,
cmap='rainbow', label = 'Cluster 1', alpha=0.2)
plt.scatter(data_process_k4[cluster_labels == 1].loc[:,'Recency'],
data_process_k4[cluster_labels == 1].loc[:,'MonetaryValue'], s= 10,
cmap='rainbow', label = 'Cluster 2', alpha=0.2)
plt.scatter(data_process_k4[cluster_labels == 2].loc[:,'Recency'],
data_process_k4[cluster_labels == 2].loc[:,'MonetaryValue'], s= 10,
cmap='rainbow', label = 'Cluster 3', alpha=0.3)
plt.scatter(data_process_k4[cluster_labels == 3].loc[:,'Recency'],
data_process_k4[cluster_labels == 3].loc[:,'MonetaryValue'], s= 10,
cmap='rainbow', label = 'Cluster 4', alpha=0.3)
plt.xticks(np.arange(0, 1000, 100))
plt.yticks(np.arange(0, 10000, 1000))
axes = plt.gca()
axes.set_xlim(0, 500)
axes.set_ylim(0, 10000)
plt.title('Cusomter Clusters')
plt.xlabel('Recency')
plt.ylabel('Monetary Value')
plt.legend()

plt.subplot(3, 1, 2);
plt.scatter(data_process_k4[cluster_labels == 0].loc[:,'Frequency'],
data_process_k4[cluster_labels == 0].loc[:,'MonetaryValue'], s= 10,
```

```
cmap='rainbow', label = 'Cluster 1', alpha=0.2)
plt.scatter(data_process_k4[cluster_labels == 1].loc[:,'Frequency'],
data_process_k4[cluster_labels == 1].loc[:,'MonetaryValue'], s= 10,
cmap='rainbow', label = 'Cluster 2', alpha=0.2)
plt.scatter(data_process_k4[cluster_labels == 2].loc[:,'Frequency'],
data_process_k4[cluster_labels == 2].loc[:,'MonetaryValue'], s= 10,
cmap='rainbow', label = 'Cluster 3', alpha=0.3)
plt.scatter(data_process_k4[cluster_labels == 3].loc[:,'Frequency'],
data_process_k4[cluster_labels == 3].loc[:,'MonetaryValue'], s= 10,
cmap='rainbow', label = 'Cluster 4', alpha=0.3)
plt.xticks(np.arange(0, 1000, 100))
plt.yticks(np.arange(0, 10000, 1000))
axes = plt.gca()
axes.set_xlim(0, 500)
axes.set_ylim(0, 10000)
plt.title('Cusomter Clusters')
plt.xlabel('Frequency')
plt.ylabel('Monetary Value')
plt.legend()

plt.subplot(3, 1, 3);
plt.scatter(data_process_k4[cluster_labels == 0].loc[:,'Tenure'],
data_process_k4[cluster_labels == 0].loc[:,'MonetaryValue'], s= 10,
cmap='rainbow', label = 'Cluster 1', alpha=0.2)
plt.scatter(data_process_k4[cluster_labels == 1].loc[:,'Tenure'],
data_process_k4[cluster_labels == 1].loc[:,'MonetaryValue'], s= 10,
cmap='rainbow', label = 'Cluster 2', alpha=0.2)
plt.scatter(data_process_k4[cluster_labels == 2].loc[:,'Tenure'],
data_process_k4[cluster_labels == 2].loc[:,'MonetaryValue'], s= 10,
cmap='rainbow', label = 'Cluster 3', alpha=0.3)
plt.scatter(data_process_k4[cluster_labels == 3].loc[:,'Tenure'],
data_process_k4[cluster_labels == 3].loc[:,'MonetaryValue'], s= 10,
cmap='rainbow', label = 'Cluster 4', alpha=0.3)
plt.xticks(np.arange(0, 1000, 100))
plt.yticks(np.arange(0, 10000, 1000))
axes = plt.gca()
axes.set_xlim(0, 500)
axes.set_ylim(0, 10000)
plt.title('Cusomter Clusters')
plt.xlabel('Tenure')
plt.ylabel('Monetary Value')
plt.legend()

plt.show()
```







Thanks for taking time to explore customer segmentation with through this 3-part series. If you enjoyed the content, stay tuned and I will be using exploring more data and perhaps build other interesting projects as well with Python.

Machine Learning