Customer Segmentation using Unsupervised Learning

Akash Nataraj
Darshan Puttanna
Pramukh Nagendra
Supreeth Betigeri

What is the project about?

What do we aim to achieve with this project?

We aim to achieve *cohort analysis* - A descriptive analytics tool.

Cohort Analysis helps with analysis of high level trends better by providing insights on metrics across both the product and product lifetime.

Now, what is a *cohort*? - A cohort is a group of subjects who share a defining characteristic. We can observe how a cohort behaves across time and compare it to other cohorts. Cohorts are often used in medicine, psychology, econometrics, ecology, and many other areas to perform a cross-section (compare difference across subjects) at intervals through time.

Types of Cohorts

1. *Time Cohorts* are customers who signed up for a product or service during a particular time frame. Analyzing these cohorts shows the customers' behavior depending on the time they started using the company's products or services. The time may be monthly or quarterly even daily.
2. *Behavior cohorts* are customers who purchased a product or subscribed to a service in the past. It groups customers by the type of product or service they signed up. Customers who signed up for basic level services might have different needs than those who signed up for advanced services. Understanding the needs of the various cohorts can help a company design custom-made services or products for particular segments.
3. *Size cohorts* refer to the various sizes of customers who purchase company's products or services. This categorization can be based on the amount of spending in some periodic time after acquisition or the product type that the customer spent most of their order amount in some period of time.

Dataset Details

We have sourced our data from Kaggle, labeled “Online Retail Data Set from UCI ML Repo”.

This is a transnational data set which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail.The company mainly sells unique all-occasion gifts. Many customers of the company are wholesalers.

Content of Data Set:

Data Set Characteristics: Multivariate, Sequential, Time-Series

Number of Instances: 541909

Area: Business

Attribute Characteristics: Integer, Real, String, Date

Number of Attributes: 8

Date Donated: 2015-11-06

Exploring and Cleaning the dataset

- We perform several dataset cleanup functions like checks for missing data from the columns.
- We drop any duplicate items in order to avoid redundancy, to ensure there are no abnormalities in the data.
- Check if any values in the dataset do not correspond accordingly (Example- the minimum for a unit price = 0, in case it is free/discounted. In case, there are negative values in such columns, we drop the values)

In [107]:
```
df.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
InvoiceNo      541909 non-null object
StockCode      541909 non-null object
Description    540455 non-null object
Quantity       541909 non-null int64
InvoiceDate    541909 non-null datetime64[ns]
UnitPrice      541909 non-null float64
CustomerID     406829 non-null float64
Country        541909 non-null object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

In [108]:
```
df.isnull().sum()
```
Out[108]:
```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

In [109]:
```
df= df.dropna(subset=['CustomerID'])
```

In [110]:
```
df.isnull().sum().sum()
```
Out[110]: 0

In [111]:
```
df.duplicated().sum()
```
Out[111]: 5225

In [112]:
```
df = df.drop_duplicates()
```

In [113]:
```
df.duplicated().sum()
```
Out[113]: 0

In [114]:
```
df.describe()
```
Out[114]:

	Quantity	UnitPrice	CustomerID
count	401604.000000	401604.000000	401604.000000
mean	12.183273	3.474064	15281.160818
std	250.283037	69.764035	1714.006089
min	-80995.000000	0.000000	12346.000000
25%	2.000000	1.250000	13939.000000
50%	5.000000	1.950000	15145.000000
75%	12.000000	3.750000	16784.000000
max	80995.000000	38970.000000	18287.000000

In [115]:
```
df=df[(df['Quantity']>0) & (df['UnitPrice']>0)]
df.describe()
```
Out[115]:

	Quantity	UnitPrice	CustomerID
count	392692.000000	392692.000000	392692.000000
mean	13.119702	3.125914	15287.843865
std	180.492832	22.241836	1713.539549
min	1.000000	0.001000	12346.000000
25%	2.000000	1.250000	13955.000000
50%	6.000000	1.950000	15150.000000
75%	12.000000	3.750000	16791.000000
max	80995.000000	8142.750000	18287.000000

In [116]:
```
df.shape
```
Out[116]: (392692, 8)

Cohort analysis labels

For cohort analysis, there are a few labels that would need to be created.

1. Invoice Period - A string representation of the year and month of a single transaction/invoice.
2. Cohort Group - A string representation of the year and month of a customer’s first purchase. This label is common across all invoices for a particular customer.
3. Cohort Period/Cohort Index - An integer representation of a customer’s stage in the “lifetime”. The number represents the number of months since the first purchase.

Cohort labels

In [117]:

```
def get_month(x) : return dt.datetime(x.year,x.month,1)
df['InvoiceMonth'] = df['InvoiceDate'].apply(get_month)
grouping = df.groupby('CustomerID')['InvoiceMonth']
df['CohortMonth'] = grouping.transform('min')
df.tail()
```

Out[117]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortMonth
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France	2011-12-01	2011-08-01
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France	2011-12-01	2011-08-01
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France	2011-12-01	2011-08-01
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France	2011-12-01	2011-08-01
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France	2011-12-01	2011-08-01

In [118]:

```
def get_month_int (dframe,column):
    year = dframe[column].dt.year
    month = dframe[column].dt.month
    day = dframe[column].dt.day
    return year, month , day

invoice_year,invoice_month,_ = get_month_int(df,'InvoiceMonth')
cohort_year,cohort_month,_ = get_month_int(df,'CohortMonth')

year_diff = invoice_year - cohort_year
month_diff = invoice_month - cohort_month

df['CohortIndex'] = year_diff * 12 + month_diff + 1
```

Cohort labels contd…

In [119]:

```
#Count monthly active customers from each cohort
grouping = df.groupby(['CohortMonth', 'CohortIndex'])
cohort_data = grouping['CustomerID'].apply(pd.Series.nunique)
# Return number of unique elements in the object.
cohort_data = cohort_data.reset_index()
cohort_counts = cohort_data.pivot(index='CohortMonth',columns='CohortIndex',values='CustomerID')
cohort_counts
```

Out[119]:

CohortIndex CohortMonth	1	2	3	4	5	6	7	8	9	10	11	12	13
2010-12-01	885.0	324.0	286.0	340.0	321.0	352.0	321.0	309.0	313.0	350.0	331.0	445.0	235.0
2011-01-01	417.0	92.0	111.0	96.0	134.0	120.0	103.0	101.0	125.0	136.0	152.0	49.0	NaN
2011-02-01	380.0	71.0	71.0	108.0	103.0	94.0	96.0	106.0	94.0	116.0	26.0	NaN	NaN
2011-03-01	452.0	68.0	114.0	90.0	101.0	76.0	121.0	104.0	126.0	39.0	NaN	NaN	NaN
2011-04-01	300.0	64.0	61.0	63.0	59.0	68.0	65.0	78.0	22.0	NaN	NaN	NaN	NaN
2011-05-01	284.0	54.0	49.0	49.0	59.0	66.0	75.0	27.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	242.0	42.0	38.0	64.0	56.0	81.0	23.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	188.0	34.0	39.0	42.0	51.0	21.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	169.0	35.0	42.0	41.0	21.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	299.0	70.0	90.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	358.0	86.0	41.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	323.0	36.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	41.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [120]:

```
# Retention table
cohort_size = cohort_counts.iloc[:,0]
retention = cohort_counts.divide(cohort_size,axis=0) #axis=0 to ensure the divide along the row axis
retention.round(3) * 100 #to show the number as percentage
```

Out[120]:

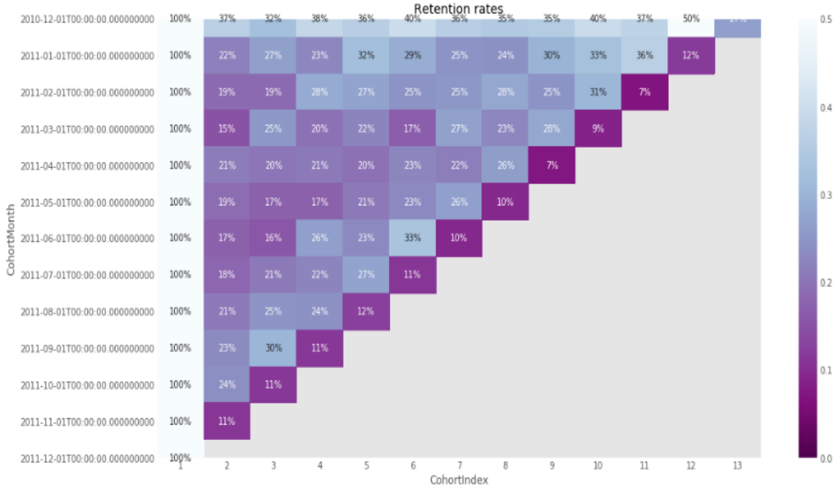
CohortIndex CohortMonth	1	2	3	4	5	6	7	8	9	10	11	12	13
2010-12-01	100.0	36.6	32.3	38.4	36.3	39.8	36.3	34.9	35.4	39.5	37.4	50.3	26.6
2011-01-01	100.0	22.1	26.6	23.0	32.1	28.8	24.7	24.2	30.0	32.6	36.5	11.8	NaN
2011-02-01	100.0	18.7	18.7	28.4	27.1	24.7	25.3	27.9	24.7	30.5	6.8	NaN	NaN
2011-03-01	100.0	15.0	25.2	19.9	22.3	16.8	26.8	23.0	27.9	8.6	NaN	NaN	NaN
2011-04-01	100.0	21.3	20.3	21.0	19.7	22.7	21.7	26.0	7.3	NaN	NaN	NaN	NaN
2011-05-01	100.0	19.0	17.3	17.3	20.8	23.2	26.4	9.5	NaN	NaN	NaN	NaN	NaN
2011-06-01	100.0	17.4	15.7	26.4	23.1	33.5	9.5	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	100.0	18.1	20.7	22.3	27.1	11.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	100.0	20.7	24.9	24.3	12.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	100.0	23.4	30.1	11.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	100.0	24.0	11.5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	100.0	11.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	100.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Heat map for customer retention table

- Customer retention is a very useful metric to understand how many, of all the customers are still active. Retention gives you the percentage of active customers compared to the total number of customers.

In [121]:

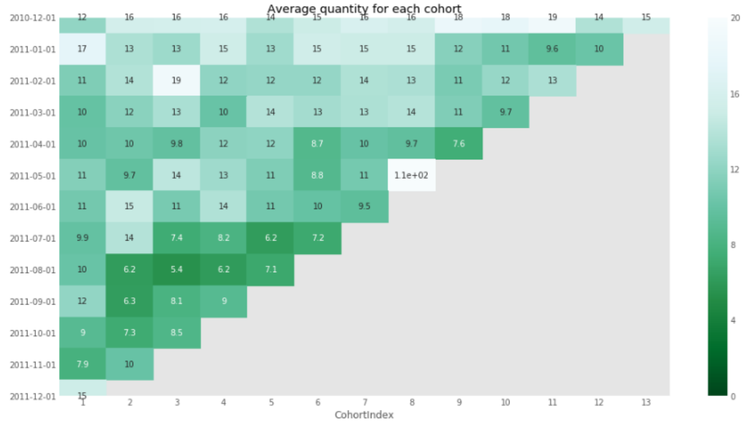
```
#Build the heatmap
plt.figure(figsize=(15, 8))
plt.title('Retention rates')
sns.heatmap(data=retention,annot = True,fmt = '.0%',vmin = 0.0,vmax = 0.5,cmap="BuPu_r")
plt.show()
```



In [122]:

```
#Average quantity for each cohort
grouping = df.groupby(['CohortMonth', 'CohortIndex'])
cohort_data = grouping['Quantity'].mean()
cohort_data = cohort_data.reset_index()
average_quantity = cohort_data.pivot(index='CohortMonth',columns='CohortIndex',values='Quantity')
average_quantity.round(1)
average_quantity.index = average_quantity.index.date

#Build the heatmap
plt.figure(figsize=(15, 8))
plt.title('Average quantity for each cohort')
sns.heatmap(data=average_quantity,annot = True,vmin = 0.0,vmax =20,cmap="BuGn_r")
plt.show()
```



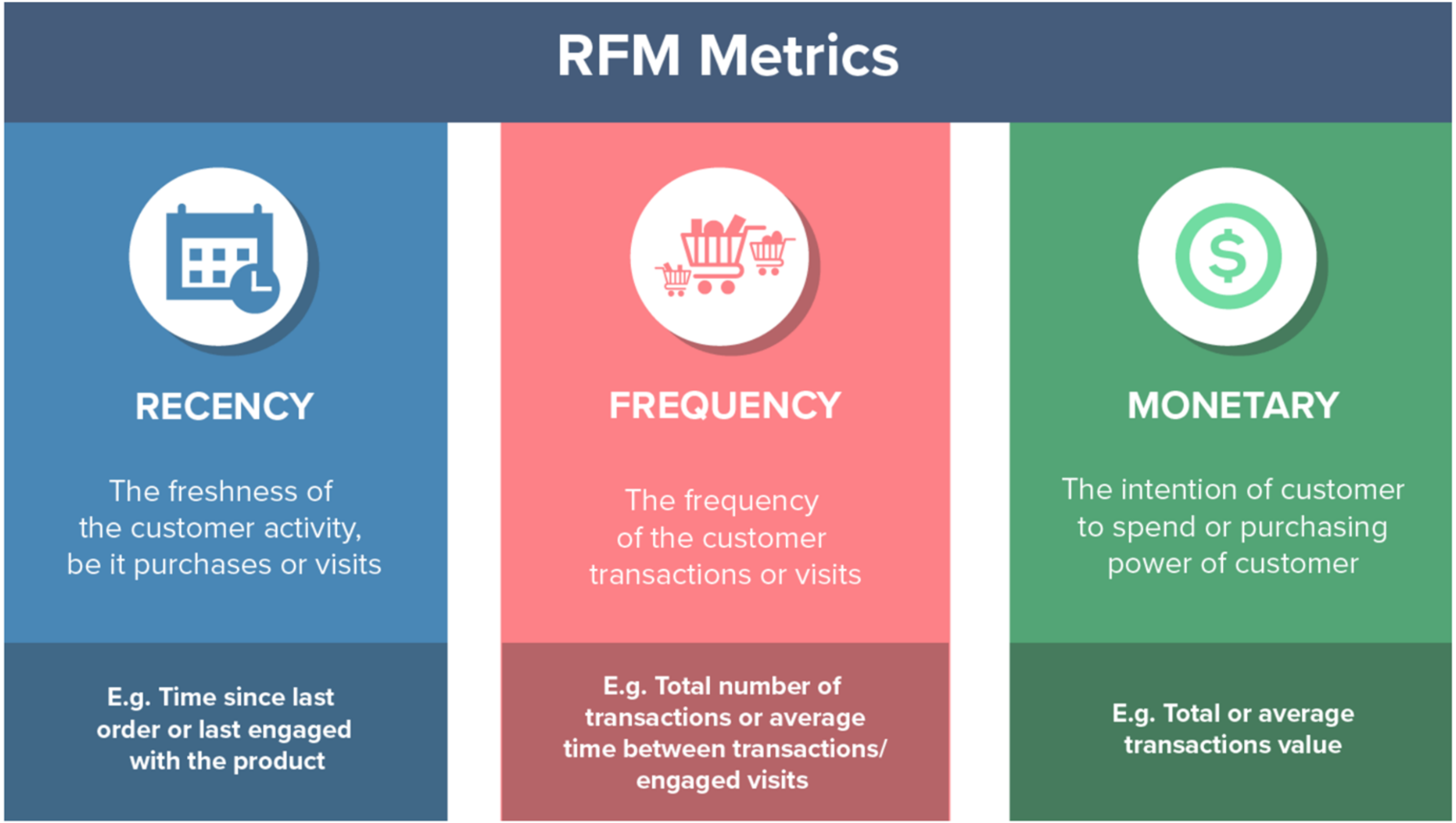
Recency, Frequency, Monetary value

What is RFM? - RFM is an acronym of recency, frequency and monetary.

- Recency is about when was the last order of a customer. It means the number of days since a customer made the last purchase. If it’s a case for a website or an app, this could be interpreted as the last visit day or the last login time.
- Frequency is about the number of purchase in a given period. It could be 3 months, 6 months or 1 year. So we can understand this value as for how often or how many times a customer used the product of a company. The bigger the value is, the more engaged the customers are. Could we say them as our VIP? Not necessary. Cause we also have to think about how much they actually paid for each purchase, which means monetary value.
- Monetary is the total amount of money a customer spent in that given period. Therefore big spenders will be differentiated with other customers such as MVP or VIP.

Once we have calculated these numbers, next step is to categorize them into some sort of categorization such as high, medium, or low.

Visual Representation for simpler understanding



RFM groupings and implementation.

We can break the customers into groups of equal size based on percentile value of each metric:

1. Percentile e.g. quantiles
2. Pareto 80/20 cut
3. Custom - based on business knowledge (use existing knowledge from previous business insights about certain threshold values for each metric).

We are going to implement percentile-based grouping.

Process of calculating percentiles is fairly simple:

1. Sort customers based on that metric
2. Break customers into a pre-defined number of groups of equal size.
3. Assign a label to each group

RFM metrics and segmentation.

What is the RFM metrics?

We will rate "Recency" customers, who have been active more recently better than the less recent customer,because each company wants its customers to be recent

We will rate "Frequency" and "Monetary Value", a higher label because we want Customer to spend more money and visit more often(that is different order than recency).

In [125]:

```
# Calculate RFM metrics
rfm = df.groupby(['CustomerID']).agg({'InvoiceDate': lambda x : (snapshot_date - x.max()).days,
                                      'InvoiceNo':'count','TotalSum': 'sum'})
#Function Lambdea: it gives the number of days between hypothetical today and the last transaction

#Rename columns
rfm.rename(columns={'InvoiceDate':'Recency','InvoiceNo':'Frequency','TotalSum':'MonetaryValue'}
           ,inplace= True)

#Final RFM values
rfm.head()
```

Out[125]:

CustomerID	Recency	Frequency	MonetaryValue
12346.0	326	1	77183.60
12347.0	2	182	4310.00
12348.0	75	31	1797.24
12349.0	19	73	1757.55
12350.0	310	17	334.40

RFM segments contd...

In [126]:

```
#Building RFM segments
r_labels =range(4,0,-1)
f_labels=range(1,5)
m_labels=range(1,5)
r_quartiles = pd.qcut(rfm['Recency'], q=4, labels = r_labels)
f_quartiles = pd.qcut(rfm['Frequency'],q=4, labels = f_labels)
m_quartiles = pd.qcut(rfm['MonetaryValue'],q=4,labels = m_labels)
rfm = rfm.assign(R=r_quartiles,F=f_quartiles,M=m_quartiles)

# Build RFM Segment and RFM Score
def add_rfm(x) : return str(x['R']) + str(x['F']) + str(x['M'])
rfm['RFM_Segment'] = rfm.apply(add_rfm,axis=1 )
rfm['RFM_Score'] = rfm[['R','F','M']].sum(axis=1)

rfm.head()
```

Out[126]:

CustomerID	Recency	Frequency	MonetaryValue	R	F	M	RFM_Segment	RFM_Score
12346.0	326	1	77183.60	1	1	4	114	6.0
12347.0	2	182	4310.00	4	4	4	444	12.0
12348.0	75	31	1797.24	2	2	4	224	8.0
12349.0	19	73	1757.55	3	3	4	334	10.0
12350.0	310	17	334.40	1	1	2	112	4.0

In [128]:

```python
rfm.groupby('RFM_Score').agg({'Recency': 'mean','Frequency': 'mean',
                              'MonetaryValue': ['mean', 'count'] }).round(1)
```

Out[128]:

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
RFM_Score				
3.0	260.7	8.2	157.4	381
4.0	177.2	13.6	240.0	388
5.0	152.9	21.2	366.6	518
6.0	95.9	27.9	820.8	457
7.0	79.6	38.0	758.1	463
8.0	64.1	56.0	987.3	454
9.0	45.9	78.7	1795.1	414
10.0	32.4	110.5	2056.4	426
11.0	21.3	186.9	4062.0	387
12.0	7.2	367.8	9285.9	450

In [129]:

```python
def segments(df):
    if df['RFM_Score'] > 9 :
        return 'Gold'
    elif (df['RFM_Score'] > 5) and (df['RFM_Score'] <= 9 ):
        return 'Sliver'
    else:
        return 'Bronze'

rfm['General_Segment'] = rfm.apply(segments,axis=1)

rfm.groupby('General_Segment').agg({'Recency':'mean','Frequency':'mean',
                                    'MonetaryValue':['mean','count']}).round(1)
```

Out[129]:

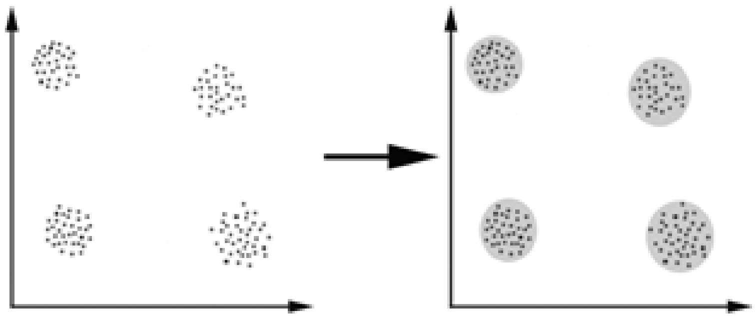
	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
General_Segment				
Bronze	192.2	15.1	266.5	1287
Gold	20.1	225.6	5246.8	1263
Sliver	72.0	49.4	1072.4	1788

Unsupervised Learning - k Means Clustering

One of the most commonly used learning algorithms in unsupervised learning is *Clustering*. It is used to find hidden patterns or to group in data for exploratory data analysis.

Therefore, a cluster is a group of objects that are “related” to each other and “dissimilar” to the objects belonging to other clusters.

Clustering can be considered the most important unsupervised problem of learning; therefore, as with any other problem of this kind, it deals with finding a structure in a set of unlabeled data. A loose clustering concept could be "the process of grouping objects into groups whose members are in some way similar."

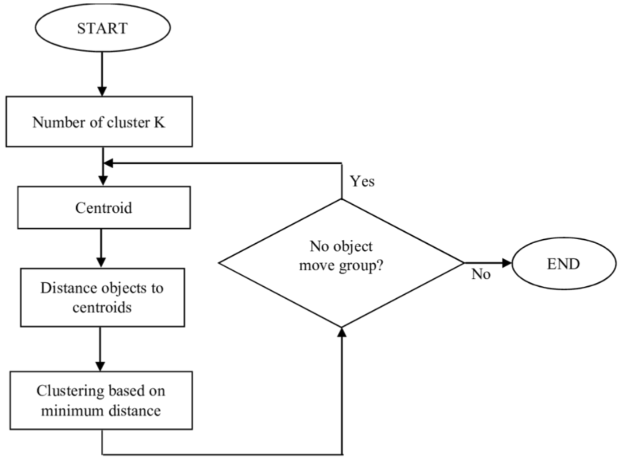


Unsupervised Learning - kMeans Clustering

- k-Means clustering is one of the most commonly used clustering method.
- It is used to cluster data points into a number (k) of mutually exclusive clusters.

The ideas behind k-means clustering are simple to visualize and understand.

The k-means algorithm organizes a set of observations that are represented as feature vectors into clusters based on their similarity. Their similarity is in turn based on a distance metric of k centroids (the centroid being the center of a cluster based on the mean of that cluster's members). The closest centroid (also represented as a feature vector) is the cluster in which a feature vector should be a member.



Data preprocessing for k-means

Before we proceed to implement k-means clustering, we would have to ensure that several key k-means assumptions are met.

- Symmetric distribution of variables (not skewed)
- Variables with same average values
- Variables with same variance.

In [27]:

```
rfm_rfm = rfm[['Recency','Frequency','MonetaryValue']]
print(rfm_rfm.describe())
```

```
          Recency    Frequency  MonetaryValue
count  4338.000000  4338.000000    4338.000000
mean     92.536422    90.523744    2048.688081
std     100.014169   225.506968    8985.230220
min       1.000000     1.000000       3.750000
25%      18.000000    17.000000     306.482500
50%      51.000000    41.000000     668.570000
75%     142.000000    98.000000    1660.597500
max     374.000000  7676.000000  280206.020000
```

From the snippet, we can observe that Mean and Variance are unequal.

To counter this issue, we would scale the variables using scaler from the scikit-learn library.

Our next major hurdle in pre-processing data for k-means is that, the distribution of variables is unsymmetric (skewed data)

To work around this, we use logarithmic transformations by which we can manage the skewness.

Using sequence of structuring pre-processing steps helps us in preparing the data for k-means implementation.

1. Unskew the data using log (positive values only) transformations.
2. Standardize to the same average values.
3. Scale to same standard deviation
4. Store as a separate array to be used for clustering.

Prepping the data in the exact same sequence matters because,

- Log transformations only works with positive data
- Normalization forces data to have negative values, post which log functions would not work.

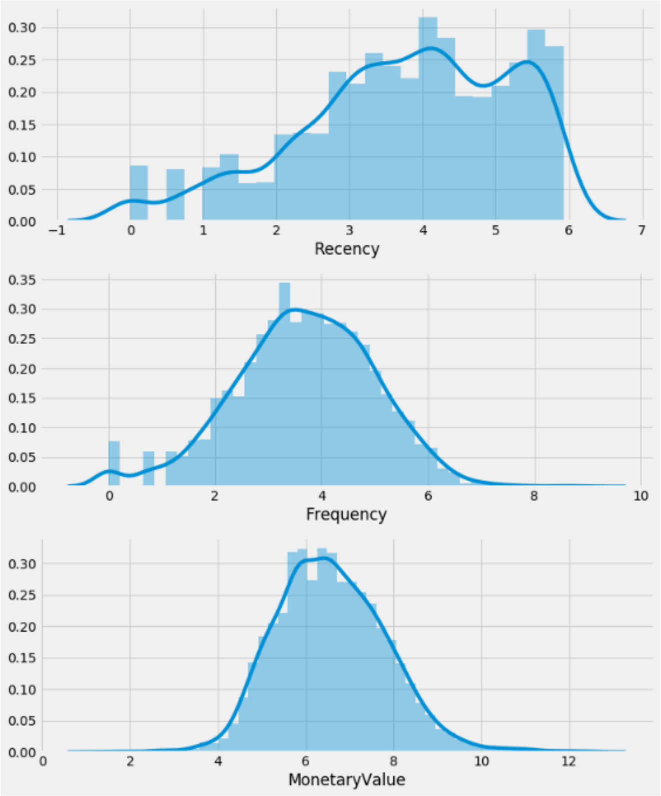
Code snippet

In [29]:

```
#Unskew the data with log transformation
rfm_log = rfm[['Recency', 'Frequency', 'MonetaryValue']].apply(np.log, axis = 1).round(3)
#or rfm_log = np.log(rfm_rfm)



# plot the distribution of RFM values
f,ax = plt.subplots(figsize=(10, 12))
plt.subplot(3, 1, 1); sns.distplot(rfm_log.Recency, label = 'Recency')
plt.subplot(3, 1, 2); sns.distplot(rfm_log.Frequency, label = 'Frequency')
plt.subplot(3, 1, 3); sns.distplot(rfm_log.MonetaryValue, label = 'Monetary Value')
plt.style.use('fivethirtyeight')
plt.tight_layout()
plt.show()
```



Implementation of k-means

We implement k-means clustering in 4 key steps:

1. Data pre-processing
2. Choosing k (the number of clusters)
3. Running k-means clustering on pre-processed data
4. Analyzing RFM values of each other.

Step 1. Normalize the data that has been unskewed, using scaler from the scikit-learning library.

In [30]:
```
#Normalize the variables with StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(rfm_log)
#Store it separately for clustering
rfm_normalized= scaler.transform(rfm_log)
```

Implementation contd..

Step 2. Choosing k (number of clusters)

Several general methods to define the number of cluster are:

1. Visual method - Elbow Criterion
2. Mathematical methods - Silhouette coefficient
3. Experimentation and interpretation.

For our project, we have chosen the visual method of “Elbow Criterion”

Elbow criterion is decided by plotting a number of clusters against within-sum-of-squared-errors (SSE) - sum of squared distances from every data point to their cluster center.

We then identify an “elbow” in this plot.

“Elbow” here is a point representing an *‘optimal’* number of clusters.

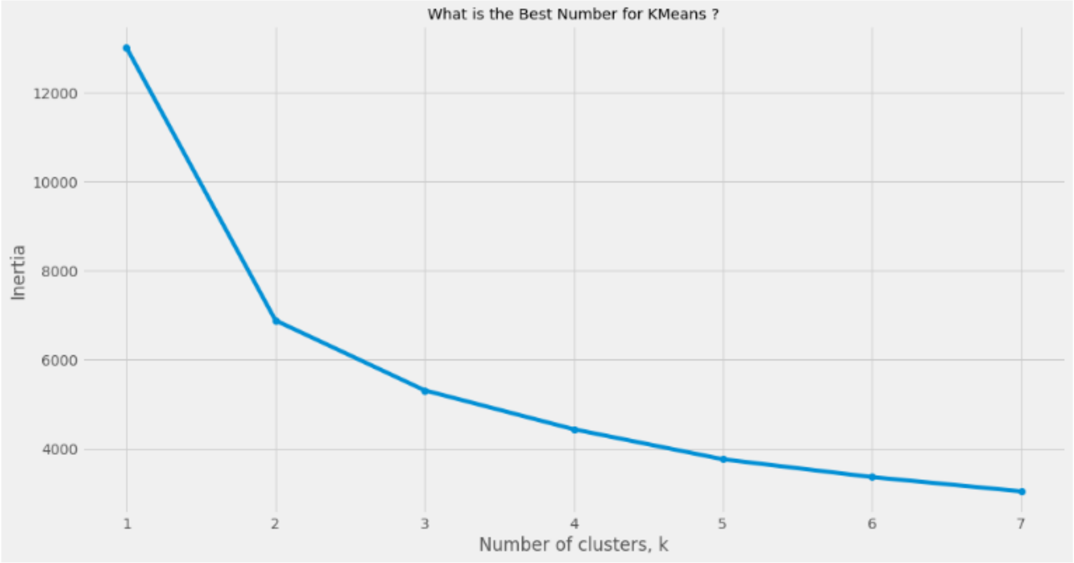
Creating clusters - Choosing the best k

In [133]:

```
from sklearn.cluster import KMeans

ks = range(1,8)
inertias=[]
for k in ks :
    # Create a KMeans clusters
    kc = KMeans(n_clusters=k,random_state=1)
    kc.fit(rfm_normalized)
    inertias.append(kc.inertia_)

# Plot ks vs inertias
f, ax = plt.subplots(figsize=(15, 8))
plt.plot(ks, inertias, '-o')
plt.xlabel('Number of clusters, k')
plt.ylabel('Inertia')
plt.xticks(ks)
plt.style.use('ggplot')
plt.title('What is the Best Number for KMeans ?')
plt.show()
```



We choose k = 3, as a best approximation to an ‘elbow’ k value.

In [134]:

```python
# clustering
kc = KMeans(n_clusters= 3, random_state=1)
kc.fit(rfm_normalized)

#Create a cluster label column in the original DataFrame
cluster_labels = kc.labels_

#Calculate average RFM values and size for each cluster:
rfm_rfm_k3 = rfm.assign(K_Cluster = cluster_labels)

#Calculate average RFM values and sizes for each cluster:
rfm_rfm_k3.groupby('K_Cluster').agg({'Recency': 'mean','Frequency': 'mean',
                                     'MonetaryValue': ['mean', 'count'],}).round(0)
```

Out[134]:

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
K_Cluster				
0	171.0	15.0	293.0	1527
1	13.0	260.0	6574.0	953
2	69.0	65.0	1170.0	1858

Visualization and comparison of segments

Using the k-clusters selected and created, we can compare the segments by plotting snake plots of normalized data, and cluster's average normalized values of each attributes.

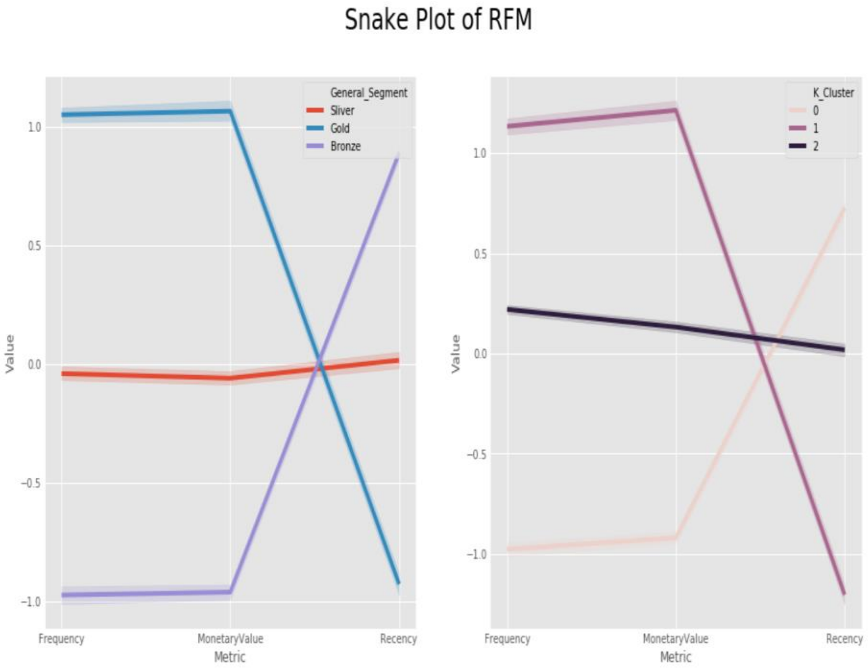
In [33]:

```
rfm_normalized = pd.DataFrame(rfm_normalized,index=rfm_rfm.index,columns=rfm_rfm.columns)
rfm_normalized['K_Cluster'] = kc.labels_
rfm_normalized['General_Segment'] = rfm['General_Segment']
rfm_normalized.reset_index(inplace = True)

#Melt the data into a long format so RFM values and metric names are stored in 1 column each
rfm_melt = pd.melt(rfm_normalized,id_vars=['CustomerID','General_Segment','K_Cluster'],value_va
rs=['Recency', 'Frequency', 'MonetaryValue'],
var_name='Metric',value_name='Value')
rfm_melt.head()
```

Out[33]:

	CustomerID	General_Segment	K_Cluster	Metric	Value
0	12346.0	Sliver	2	Recency	1.409982
1	12347.0	Gold	1	Recency	-2.146578
2	12348.0	Sliver	2	Recency	0.383648
3	12349.0	Gold	2	Recency	-0.574961
4	12350.0	Bronze	0	Recency	1.375072



Analysis and relative importance of segment attributes

- Useful technique to identify relative importance of each segment's attribute
- Calculate average values of each cluster
- Calculate average of population of dataset
- Calculate importance of RFM score by dividing them and subtracting 1 (ensures 0 is returned when

In [35]:

```
# The further a ratio is from 0, the more important that attribute is for a segment relative to t
he total population
cluster_avg = rfm_rfm_k3.groupby(['K_Cluster']).mean()
population_avg = rfm_rfm.mean()
relative_imp = cluster_avg / population_avg - 1
relative_imp.round(2)
```

Out[35]:

	Recency	Frequency	MonetaryValue
K_Cluster			
0	0.85	-0.84	-0.86
1	-0.86	1.88	2.21
2	-0.25	-0.28	-0.43

In [36]:

```
# the mean value in total
total_avg = rfm.iloc[:, 0:3].mean()
# calculate the proportional gap with total mean
cluster_avg = rfm.groupby('General_Segment').mean().iloc[:, 0:3]
prop_rfm = cluster_avg/total_avg - 1
prop_rfm.round(2)
```

Out[36]:

	Recency	Frequency	MonetaryValue
General_Segment			
Bronze	1.08	-0.83	-0.87
Gold	-0.78	1.49	1.56
Sliver	-0.22	-0.45	-0.48

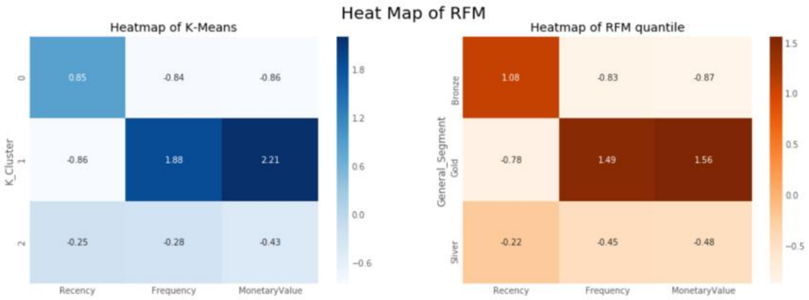
In [37]:

```
# heatmap with RFM
f, (ax1, ax2) = plt.subplots(1,2, figsize=(15, 5))
sns.heatmap(data=relative_imp, annot=True, fmt='.2f', cmap='Blues',ax=ax1)
ax1.set(title = "Heatmap of K-Means")

# a snake plot with K-Means
sns.heatmap(prop_rfm, cmap= 'Oranges', fmt= '.2f', annot = True,ax=ax2)
ax2.set(title = "Heatmap of RFM quantile")

plt.suptitle("Heat Map of RFM",fontsize=20) #make title fontsize subtitle

plt.show()
```



Conclusion

Results of the project - With the clustering and segmentation of customers using RFM, and RFM metrics, we are able to find behavioral aspects of the customers towards a product, and the loyalty towards the product, along the lines of the lifetime of a product.

Why k-means works with the data set? - Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. Cluster analysis is the most commonly used method in unsupervised learning. The clusters are modeled using a measure of similarity, which we define using k-means to find the similarities in data set, and group them accordingly.

Source and References.

- Rose, K., Gurewitz, E., & Fox, G.,”A deterministic annealing approach to clustering. Pattern Recognition Letters”, 1990, Letters, 11(11), 589–594.
- Paul Beckwith, “Unsupervised Machine Learning” September 2018, accessed on March 4, 2020.
- Hofmann, T., Puzicha, J., & Jordan, M. I., “Unsupervised learning from dyadic data in Advances in neural Information Processing Systems”, 1999, Vol. 11, MIT Press.
- Memoona Khanum, Tahira Mahboob “A Survey on Unsupervised Machine Learning Algorithms for Automation, Classification and Maintenance”, International Journal of Computer Applications (0975 – 8887) Volume 119 – No.13, June 2015.
- T.Hofmann, “Unsupervised Learning by Probabilistic Latent Semantic Analysis.”, 2001, pp 177–196.
- https://www.datacamp.com/courses/customer-segmentation-in-python
- https://www.kaggle.com/mahmoudelfahl/cohort-analysis-customer-segmentation-with-rfm
- https://www.kaggle.com/jihyeseo/online-retail-data-set-from-uci-ml-repo/version/1

Thank you.

Questions?