> Our main goal in this part of project is to group the clients into customer segments by using a clustering technique. Here, we will use K-Means clustering approach as its unsupervised machine learning model and it suits best for the creating categories. So we will analyse the customers by using the RFM (recency, frequency, monetary value) approach. This method is used to evaluate customer value.

```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy  import array
from math import sqrt
import matplotlib.pyplot as plt
from pyspark import SparkConf, SparkContext
# conf = SparkConf().setMaster("local").setAppName("Kmeans")
# sc = SparkContext.getOrCreate(conf = conf)
# sc
sc.version

Out[1]: '2.4.0'
```

Loading the dataset and pyspark Libraries into the Noteboook

```
from pyspark.sql.functions import unix_timestamp
from pyspark.sql.functions import from_unixtime, to_date, date_format,datediff
from pyspark.sql.functions import col, expr, when
from pyspark.sql.functions import mean, min, max, sum, round, count, datediff, to_date
from pyspark.sql.functions import to_utc_timestamp, unix_timestamp, lit,datediff, col

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# load the packages
import seaborn as sns; sns.set(rc={'figure.figsize':(16,9)}) #visualization tool
import datetime as dt
```

```
# File location and type
file_location = "/FileStore/tables/d17export.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df)
```

product_category_name	product_category_name_english	seller_id	seller_zip_code_prefix	seller_city	seller_state	product_id
perfumaria	perfumery	325f3178fb58e2a9778334621eecdbf9	6790	taboao da serra	SP	6782d593f63105318f46bbf76
esporte_lazer	sports_leisure	a17f621c590ea0fab3d5d883e1630ec6	18055	sorocaba	SP	e95ee6822b66ac6058e2e4af
utilidades_domesticas	housewares	1b4c3a6f53068f0b6944d2d005c9fc89	88730	sao ludgero	SC	e9a69340883a438c3f91739d
telefonia	telephony	ea8482cd71df3c1969d7b9473ff13abc	4160	sao paulo	SP	036734b5a58d5d4f46b0616c
cama_mesa_banho	bed_bath_table	d1c281d3ae149232351cd8c8cc885f0d	14940	ibitinga	SP	b1434a8f79cb3528540d9b21

Showing the first 1000 rows.

```
# df.printSchema()
```

In the first step all null values are removed and timestamp columns are converted to date format.

```
df_km = df.select('customer_id', 'customer_unique_id', 'order_purchase_timestamp', 'order_approved_at', 'order_id', 'order_item_id', 'payment_value')
df_km = df_km.dropna(how='any')
df_km = df_km.withColumn("order_purchase_timestamp",date_format('order_purchase_timestamp','yyyy-MM-dd'))
df_km = df_km.withColumn("order_approved_at",date_format('order_approved_at','yyyy-MM-dd'))
df_km.createTempView('df_km')
# df_km.show()
```

In the next step we will be calculating Recency, Frequency and Monetary.

RFM stands for the three dimensions:

Recency—How recently did the customer purchase?

Since the date of our last order was 2018-9-13, we'll consider it as the most recent one. Then, we'll subtract each order date for the customer to get the Duration elapsed between current date and the order date.After that for every order the min of the Duration will the most recent one for the each customer.

Frequency—How often do they purchase?

Frequency is calculated by countin the unique order_id numbers for each customer.

Monetary Value—How much do they spend?

Monetary value is calculated by calculating the sum of the payment value of each product grouping by every customer.

https://en.wikipedia.org/wiki/RFM_(customer_value (https://en.wikipedia.org/wiki/RFM_(customer_value))

```
current = dt.datetime(2018,9,13)
df_km = df_km.withColumn('Duration', datediff(lit(current), 'order_purchase_timestamp'))
recency = df_km.groupBy('customer_id').agg(min('Duration').alias('Recency'))
monetary = df_km.groupBy('customer_id').agg(round(sum('payment_value'), 2).alias('Monetary'))
frequency = sqlContext.sql("""select customer_id, count(order_id) as Frequency from df_km group by customer_id""")
```

```
rfm = recency.join(frequency,'customer_id', how = 'inner')\
             .join(monetary,'customer_id', how = 'inner')
```

```
rfm.show(10)
```

```
+--------------------+-------+---------+--------+
|         customer_id|Recency|Frequency|Monetary|
```

```
+--------------------+-------+---------+--------+
|f82d1df095bf073a1...|     44|        1|  117.57|
|6442504c76c94895e...|    391|        1|   73.34|
|e139948a02a65f3d6...|     87|        1|  140.62|
|39fca5d473f90b6a8...|    341|        1|   62.01|
|fb6f8e0cb2c65dfad...|    220|        1|  155.27|
|59b4f1123326b6a40...|    479|        2|  136.36|
|baf94989390857ffc...|    135|        1|  146.34|
|8baeca32aac79a831...|     28|        1|   59.87|
|dd940eccd485dc083...|    546|        1|  217.59|
|35148f0d009a1f17f...|    293|        1|  115.94|
+--------------------+-------+---------+--------+
only showing top 10 rows


rfm = rfm.filter(rfm.Monetary < 305)
rfm = rfm.dropna(how='any')
# rfm.createTempView('rfm36')
# rfm.describe().show()


# %sql
# select Recency from rfm36


Cols = ["Recency", "Frequency", "Monetary"]
```

> As we are using pyspark.ml library, which is based on dataframe api and takes an input of dense vector. we will be converting the RFM columns into dense vector and then standardizing it by MinMax scaler library to achieve the standardization between 0-1.

https://spark.apache.org/docs/2.1.0/api/python/pyspark.ml.html#pyspark.ml.feature.MinMaxScalerModel (https://spark.apache.org/docs/2.1.0/api/python/pyspark.ml.html#pyspark.ml.feature.MinMaxScalerModel)
https://spark.apache.org/docs/2.1.0/api/python/pyspark.ml.html#pyspark.ml.feature.VectorAssembler (https://spark.apache.org/docs/2.1.0/api/python/pyspark.ml.html#pyspark.ml.feature.VectorAssembler)

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import MinMaxScaler, VectorAssembler
assembler = VectorAssembler(inputCols= Cols , outputCol="rfm")
# scaler = MinMaxScaler(inputCol="rfm",outputCol="features")
pipeline = Pipeline(stages=[assembler])
model=pipeline.fit(rfm)
transformed = model.transform(rfm)
transformed.show(5)
```

```
+--------------------+-------+---------+--------+------------------+
|         customer_id|Recency|Frequency|Monetary|               rfm|
+--------------------+-------+---------+--------+------------------+
|f82d1df095bf073a1...|     44|        1|  117.57| [44.0,1.0,117.57]|
|6442504c76c94895e...|    391|        1|   73.34| [391.0,1.0,73.34]|
|e139948a02a65f3d6...|     87|        1|  140.62| [87.0,1.0,140.62]|
|39fca5d473f90b6a8...|    341|        1|   62.01| [341.0,1.0,62.01]|
|fb6f8e0cb2c65dfad...|    220|        1|  155.27|[220.0,1.0,155.27]|
+--------------------+-------+---------+--------+------------------+
only showing top 5 rows
```

```
scaler = MinMaxScaler(inputCol="rfm",outputCol="features")
scalerModel =  scaler.fit(transformed)
scaledData = scalerModel.transform(transformed)
scaledData.show(5,False)
```

```
+--------------------------------+-------+---------+--------+------------------+----------------------------------------+
|customer_id                     |Recency|Frequency|Monetary|rfm               |features                                |
+--------------------------------+-------+---------+--------+------------------+----------------------------------------+
|f82d1df095bf073a1298faed123b844b|44     |1        |117.57  |[44.0,1.0,117.57] |[0.04663923182441701,0.0,0.3655877573131094]|
|6442504c76c94895e7a26f621fa42b8f|391    |1        |73.34   |[391.0,1.0,73.34] |[0.522633744855967,0.0,0.21583829902491874] |
|e139948a02a65f3d63dca4169cd990b8|87     |1        |140.62  |[87.0,1.0,140.62] |[0.1056241426611797,0.0,0.44362811484290354]|
|39fca5d473f90b6a8b4cd39f5834202a|341    |1        |62.01   |[341.0,1.0,62.01] |[0.4540466392318244,0.0,0.1774783315276273] |
|fb6f8e0cb2c65dfada05252697877460|220    |1        |155.27  |[220.0,1.0,155.27]|[0.2880658436213992,0.0,0.49322860238353194]|
+--------------------------------+-------+---------+--------+------------------+----------------------------------------+
only showing top 5 rows
```

For optimum number of clusters on which the customers are segmented,First we are calulating the sum of squares for cluster ranging from 2 to 9 and then plotting a graph between sum of squares and number of clusters

https://spark.apache.org/docs/latest/ml-clustering.html (https://spark.apache.org/docs/latest/ml-clustering.html)

```
from pyspark.ml.clustering import KMeans
import numpy as np

cost = np.zeros(10)
for k in range(2,10):
  kmeans = KMeans()\
        .setK(k)\
        .setSeed(1) \
        .setFeaturesCol("features")\
        .setPredictionCol("cluster")
  model = kmeans.fit(scaledData)
  cost[k] = model.computeCost(scaledData)
```

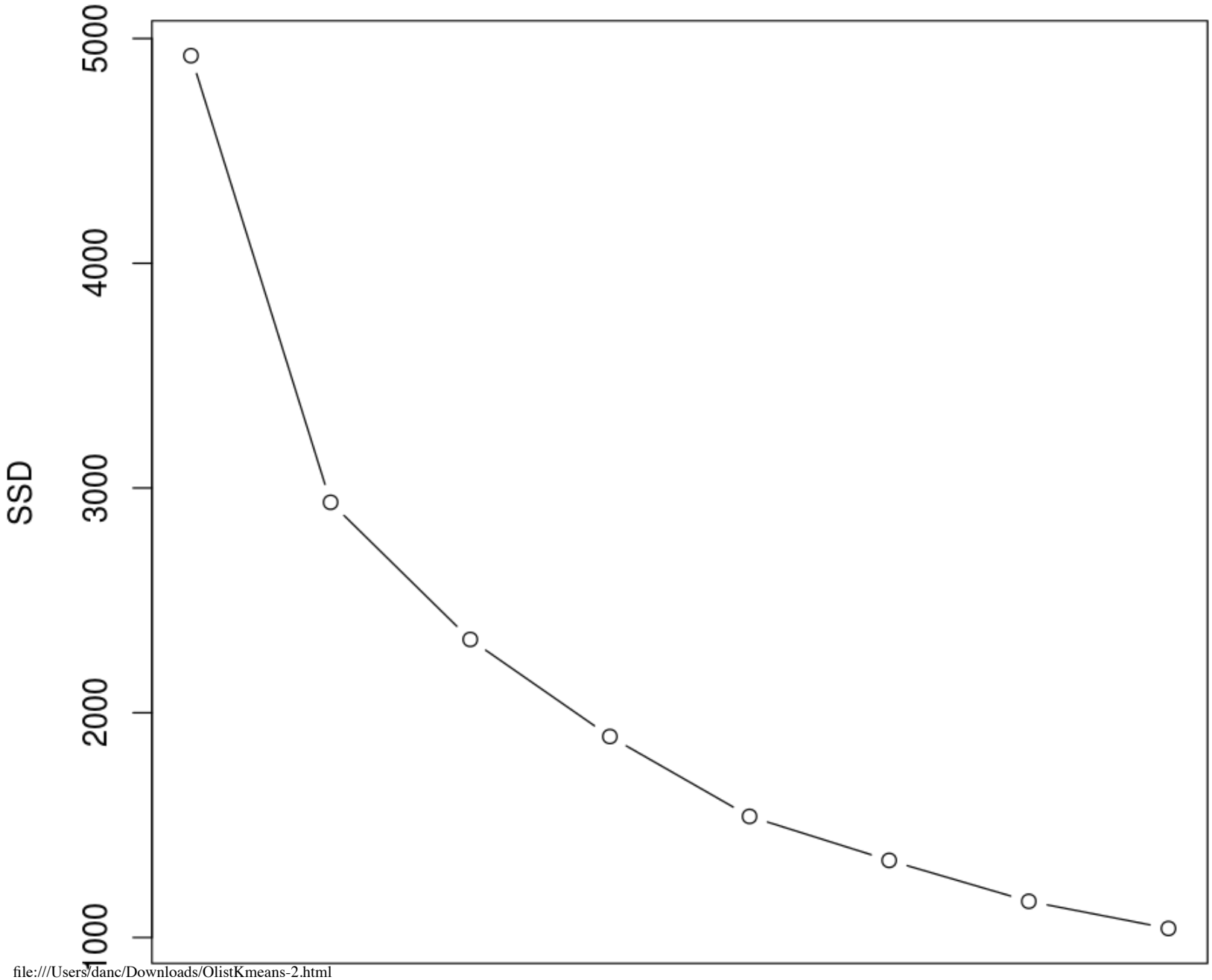
Sum of squares for clusters

```
cost[:]

Out[16]:
array([    0.        ,     0.        ,  4923.57861937,  2936.43345664,
        2326.17682717,  1894.25379678,  1538.86965369,  1343.14125795,
        1160.97649374,  1040.32316154])
```

Plot of sum of squared distances for k number of Clusters

```
%r
library(SparkR)
Clusters <- c(2,3,4,5,6,7,8,9)
SSD <- c(4923.57861937,  2936.43345664,
        2326.17682717,  1894.25379678,  1538.86965369,  1343.14125795,
        1160.97649374,  1040.32316154)
plot(Clusters, SSD,  xaxt = "n", type = "b")
```


SSD
5000
4000
3000
2000
1000

By elbow method, we can determine that the no. of optimum clusters can be determined at clusters = 3 or 6.we can clearly see that when clusters = 3 there is a bend and and a smooth curve untill 6 again a smooth curve.
So i am considering both optimum clusters visualize

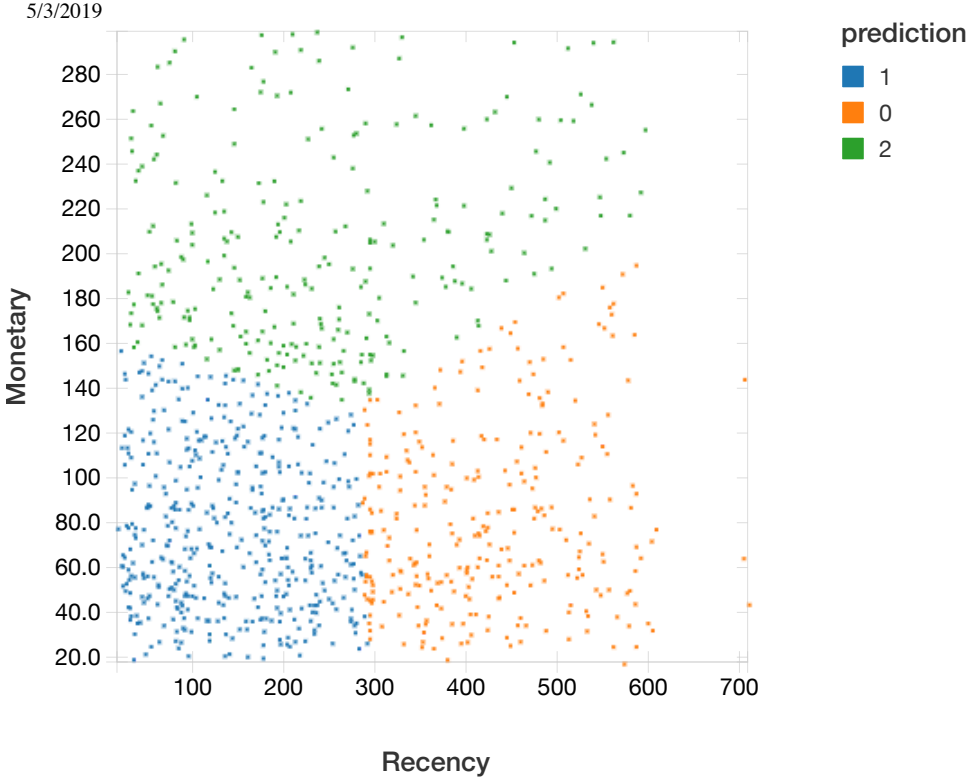
https://www.scikit-yb.org/en/latest/api/cluster/elbow.html (https://www.scikit-yb.org/en/latest/api/cluster/elbow.html)

Predicting Based on Clusters value K = 3

```
from pyspark.ml.clustering import KMeans
import numpy as np
model = KMeans().setK(3).setFeaturesCol("features").fit(scaledData)
# # Make predictions
predictions = model.transform(scaledData).select("Recency", "Frequency", "Monetary", "prediction")
# predictions.show(5,False)


predictions.createTempView("Predic3")


%sql
select * from Predic3
```



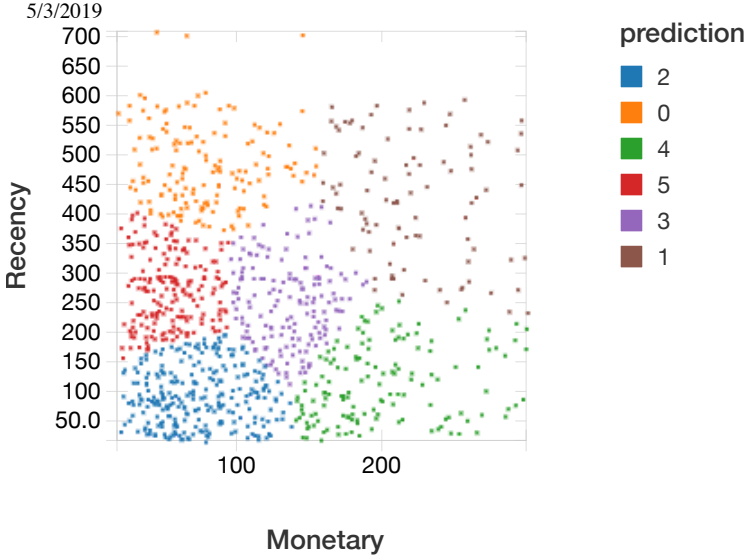
Showing sample based on the first 1000 rows.

Predicting Based on Clusters K =6

```
from pyspark.ml.clustering import KMeans
import numpy as np
model6 = KMeans().setK(6).setFeaturesCol("features").fit(scaledData)
predictions6 = model6.transform(scaledData).select("Recency", "Frequency", "Monetary", "prediction")
```

```
predictions6.createTempView("Predic6")
```

```
%sql
select * from Predic6
```



Showing sample based on the first 1000 rows.

Other way to Find Optimum Clusters

The Below code is used to find out silhouette Coeffcient to Determine the Number of Clusters, where each block of code has been run 8 times with different clusters to save the shilloute coefficients.This method can be easily performed by Looping and saving,But due to Heap Memory Size Restriction in databricks community edition. we choose to it in a manual way

http://ros-developer.com/2017/12/04/silhouette-coefficient-finding-optimal-number-clusters/ (http://ros-developer.com/2017/12/04/silhouette-coefficient-finding-optimal-number-clusters/)
https://spark.apache.org/docs/latest/ml-clustering.html (https://spark.apache.org/docs/latest/ml-clustering.html)

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

silh_val = []
kmeans02 = KMeans()\
        .setK(2)\
        .setSeed(1) \
        .setFeaturesCol("features")\
        .setPredictionCol("prediction")
model02 = kmeans02.fit(scaledData)
predictions02 = model02.transform(scaledData)
   # Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions02)
silh_val.append(silhouette)


from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
#   silh_val = []
kmeans3 = KMeans()\
        .setK(3)\
        .setSeed(1) \
        .setFeaturesCol("features")\
        .setPredictionCol("prediction")
model3 = kmeans3.fit(scaledData)
predictions3 = model3.transform(scaledData)
   # Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions3)
silh_val.append(silhouette)
```

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
kmeans4 = KMeans()\
        .setK(4)\
        .setSeed(1) \
        .setFeaturesCol("features")\
        .setPredictionCol("prediction")
model4 = kmeans4.fit(scaledData)
predictions4 = model4.transform(scaledData)
# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions4)
silh_val.append(silhouette)
```

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
kmeans5 = KMeans()\
        .setK(5)\
        .setSeed(1) \
        .setFeaturesCol("features")\
        .setPredictionCol("prediction")
model5 = kmeans5.fit(scaledData)
predictions5 = model5.transform(scaledData)
# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions5)
silh_val.append(silhouette)
```

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
kmeans6 = KMeans()\
        .setK(6)\
        .setSeed(1) \
        .setFeaturesCol("features")\
        .setPredictionCol("prediction")
model6 = kmeans6.fit(scaledData)
predictions6 = model6.transform(scaledData)
   # Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions6)
silh_val.append(silhouette)
```

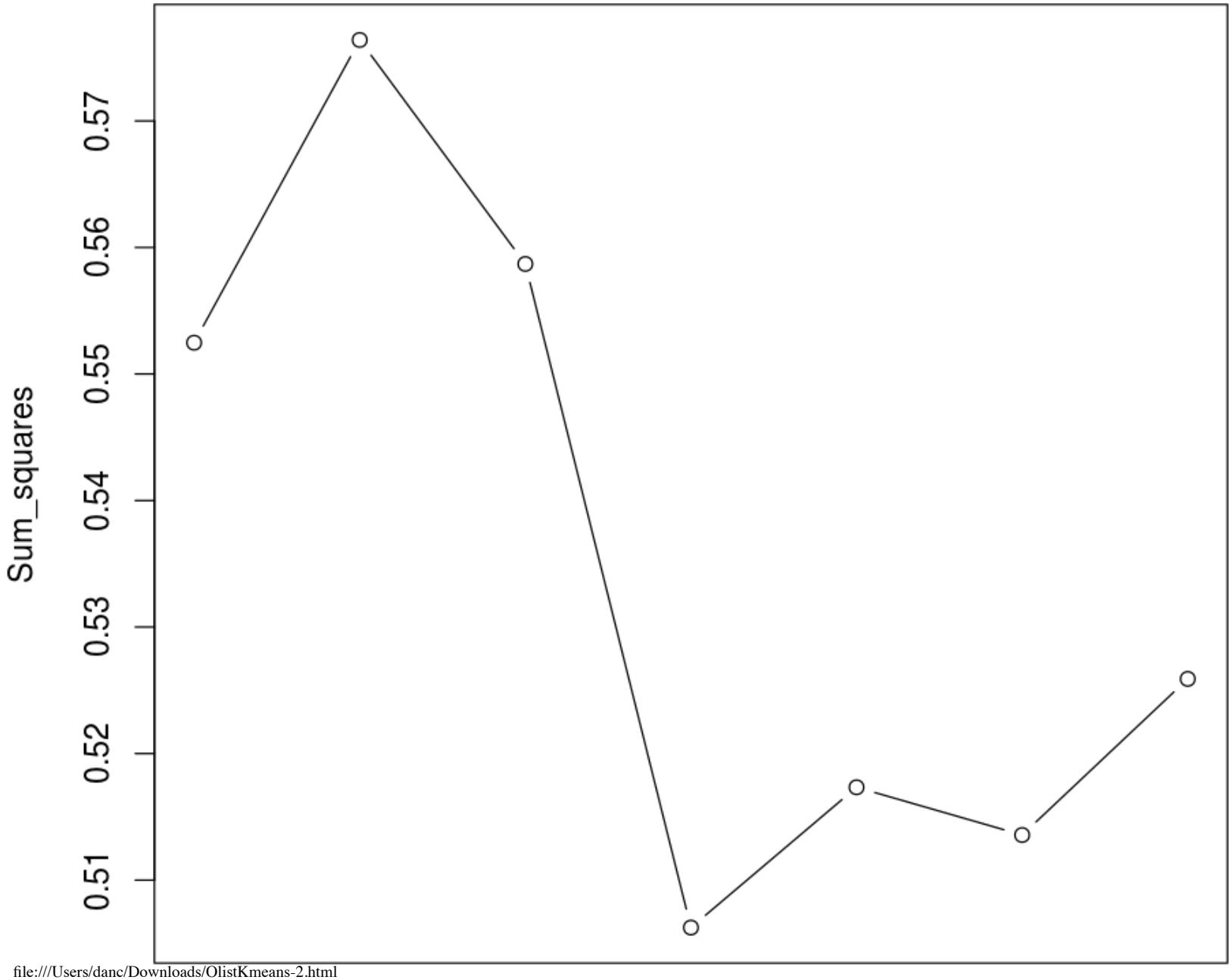
```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
kmeans7 = KMeans()\
        .setK(7)\
        .setSeed(1) \
        .setFeaturesCol("features")\
        .setPredictionCol("prediction")
model7 = kmeans7.fit(scaledData)
predictions7 = model7.transform(scaledData)
   # Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions7)
silh_val.append(silhouette)
```

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
kmeans8 = KMeans()\
        .setK(8)\
        .setSeed(1) \
        .setFeaturesCol("features")\
        .setPredictionCol("prediction")
model8 = kmeans8.fit(scaledData)
predictions8 = model8.transform(scaledData)
   # Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions8)
silh_val.append(silhouette)
```

```
silh_val[:]
```

```
Out[28]:
[0.5524746964944,
 0.5764018954070922,
 0.5586979401699084,
 0.506247172990792,
 0.517334171674388,
 0.5135625826705568,
 0.5259004486127394]
```

```
%r
library(SparkR)
library(ggplot2)
Clusters <- c(2,3,4,5,6,7,8)
Sum_squares <- c(0.5524746964944,
 0.5764018954070922,
 0.5586979401699084,
 0.506247172990792,
 0.517334171674388,
 0.5135625826705568,
 0.5259004486127394)
plot(Clusters,Sum_squares,  xaxt = "n", type = "b")
```


Sum_squares
0.51
0.52
0.53
0.54
0.55
0.56
0.57

The silhouette coefficient quantifies the quality of clustering achieved -- so the number of clusters that maximizes the silhouette coefficient will be optimum number of clusters that can be used for Building the Model.

So From the above Plot, with number of clusters k =3 the model is builded and Customer segmentation is predicted.

https://stats.stackexchange.com/questions/10540/how-to-interpret-mean-of-silhouette-plot (https://stats.stackexchange.com/questions/10540/how-to-interpret-mean-of-silhouette-plot)
https://en.wikipedia.org/wiki/Silhouette_(clustering (https://en.wikipedia.org/wiki/Silhouette_(clustering))

```
%sql
select * from predic3
```


prediction
1
0
2
1.00
1.50
2.00
2.50
3.00
Monetary
Frequency
Recency
3.00
2.80
2.60
2.40
2.20
2.00
1.80
1.60
1.40
1.20
1.00
700
650
600
550
500
450
400
350

Showing sample based on the first 1000 rows.

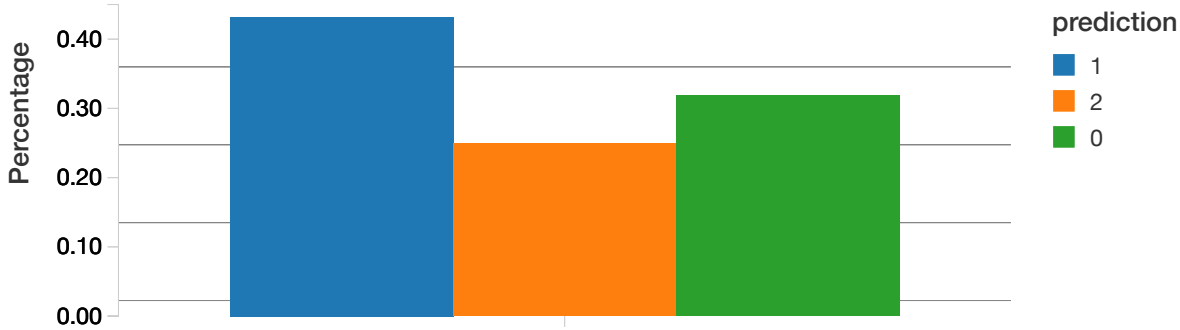
Calculating the Centers of Clusters when Number of Clusters = 3.!!

```
# Shows the result.
centers = model3.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Cluster Centers:
[ 0.56022552  0.00326534  0.24905321]
[ 0.18646024  0.00215229  0.22529766]
[ 0.30743224  0.00757261  0.66181217]
```

PLOT 1

```
%sql
select (count(prediction)/(select(count(*)) from predic3)) as Percentage, prediction from predic3 group by prediction
```



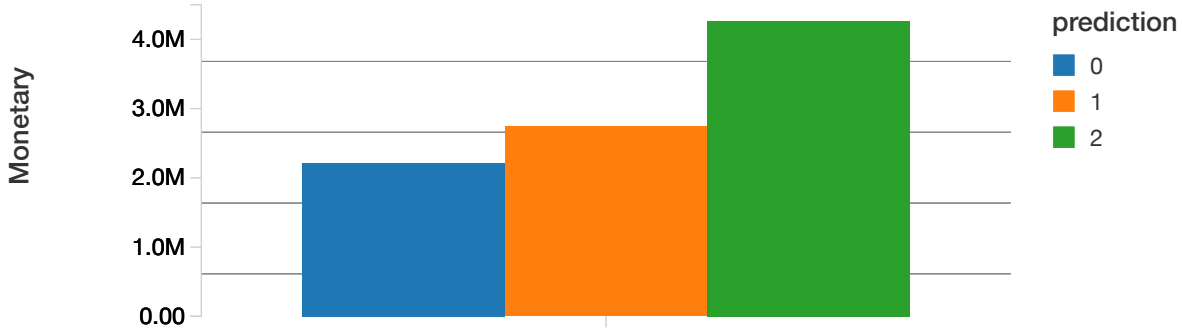
From the Table, we can see that Cluster 1 has the highest percentage of customers base with 43.1% and then followed by cluster 0 with 31.8%

```
%sql
select avg(Monetary), prediction from predic3 group by prediction
```

avg(Monetary)	prediction
76.14112048861745	1
205.0594975478412	2
83.12981718326819	0

PLOT 2

```
%sql
select * from predic3
```



From the above plots, we can see that the average monetary value in each clusters
where cluster 2 has the highest average;the customer base in the customer 2 can be categorised into high spending customers
The cluster 1 has the least average:cluster 1 can be categorised as Least spendn customers
Cluster 0 has the medium average, which can be categorised into middle of both spenders.

So in conclusion, we formed 3 clusters,where the customers can be segmented into 3 categories based on the Recency, Frequency and Monetary value. If we take Monetary value as one of the criteria the clusters can be categorised into High, Medium and Low Spenders

Based on this Customer Segmentation, the marketing department can categorise the customers and recommend the respective category products for respective Customer Base

```
Command skipped
```