**Retail – Capstone project**

DESCRIPTION

**Problem Statement**

* It is a critical requirement for business to understand the value derived from a customer. RFM is a method used for analyzing customer value.
* Customer segmentation is the practice of segregating the customer base into groups of individuals based on some common characteristics such as age, gender, interests, and spending habits
* Perform customer segmentation using RFM analysis. The resulting segments can be ordered from most valuable (highest recency, frequency, and value) to least valuable (lowest recency, frequency, and value).

**Dataset Description**

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

|  |  |
| --- | --- |
| **Variables** | **Description** |
| InvoiceNo | Invoice number. Nominal, a six digit integral number uniquely assigned to each transaction. If this code starts with letter 'c', it indicates a cancellation |
| StockCode | Product (item) code. Nominal, a five digit integral number uniquely assigned to each distinct product |
| Description | Product (item) name. Nominal |
| Quantity | The quantities of each product (item) per transaction. Numeric |
| InvoiceDate | Invoice Date and time. Numeric, the day and time when each transaction was generated |
| UnitPrice | Unit price. Numeric, product price per unit in sterling |
| CustomerID | Customer number. Nominal, a six digit integral number uniquely assigned to each customer |
| Country | Country name. Nominal, the name of the country where each customer resides |

**Project Task: Week 1**

**Data Cleaning:**

1. Perform a preliminary data inspection and data cleaning.

a. Check for missing data and formulate an apt strategy to treat them.

b. Remove duplicate data records.

c. Perform descriptive analytics on the given data.

**Data Transformation:**

2. Perform cohort analysis (a cohort is a group of subjects that share a defining characteristic). Observe how a cohort behaves across time and compare it to other cohorts.

a. Create month cohorts and analyze active customers for each cohort.

b. Analyze the retention rate of customers.

**Project Task: Week 2**

**Data Modeling :**

1. Build a RFM (Recency Frequency Monetary) model. Recency means the number of days since a customer made the last purchase. Frequency is the number of purchase in a given period. It could be 3 months, 6 months or 1 year. Monetary is the total amount of money a customer spent in that given period. Therefore, big spenders will be differentiated among other customers such as MVP (Minimum Viable Product) or VIP.

2. Calculate RFM metrics.

3. Build RFM Segments. Give recency, frequency, and monetary scores individually by dividing them into quartiles.

b1. Combine three ratings to get a RFM segment (as strings).

b2. Get the RFM score by adding up the three ratings.

b3. Analyze the RFM segments by summarizing them and comment on the findings.

Note: Rate “recency" for customer who has been active more recently higher than the less recent customer, because each company wants its customers to be recent.

Note: Rate “frequency" and “monetary" higher, because the company wants the customer to visit more often and spend more money

**Project Task: Week 3**

**Data Modeling :**

1. Create clusters using k-means clustering algorithm.

a. Prepare the data for the algorithm. If the data is asymmetrically distributed, manage the skewness with appropriate transformation. Standardize the data.

b. Decide the optimum number of clusters to be formed.

c. Analyze these clusters and comment on the results.

**Project Task: Week 4**

**Data Reporting:​​​​​​​**

1. Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:

a. Country-wise analysis to demonstrate average spend. Use a bar chart to show the monthly figures

b. Bar graph of top 15 products which are mostly ordered by the users to show the number of products sold

c. Bar graph to show the count of orders vs. hours throughout the day

d. Plot the distribution of RFM values using histogram and frequency charts

e. Plot error (cost) vs. number of clusters selected

f. Visualize to compare the RFM values of the clusters using heatmap

*#Import necessary libraries*

get\_ipython().run\_line\_magic('matplotlib', 'inline')

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

*#Importing RetailData*

Rtl\_data = pd.read\_excel('RetailData.xlsx', encoding = 'unicode\_escape')

Rtl\_data.head()

*#Check the shape (number of columns and rows) in the dataset*

Rtl\_data.shape

*#Customer distribution by country*

country\_cust\_data=Rtl\_data[['Country','CustomerID']].drop\_duplicates()

country\_cust\_data.groupby(['Country'])['CustomerID'].aggregate('count').reset\_index().sort\_values('CustomerID', ascending=False)

*#Keep only United Kingdom data*

Rtl\_data = Rtl\_data.query("Country=='United Kingdom'").reset\_index(drop=True)

*#Check for missing values in the dataset*

Rtl\_data.isnull().sum(axis=0)

*#Remove missing values from CustomerID column, can ignore missing values in description column*

Rtl\_data = Rtl\_data[pd.notnull(Rtl\_data['CustomerID'])]

*#Validate if there are any negative values in Quantity column*

Rtl\_data.Quantity.min()

*#Validate if there are any negative values in UnitPrice column*

Rtl\_data.UnitPrice.min()

*#Filter out records with negative values*

Rtl\_data = Rtl\_data[(Rtl\_data['Quantity']>0)]

*#Convert the string date field to datetime*

Rtl\_data['InvoiceDate'] = pd.to\_datetime(Rtl\_data['InvoiceDate'])

*#Add new column depicting total amount*

Rtl\_data['TotalAmount'] = Rtl\_data['Quantity'] \* Rtl\_data['UnitPrice']

*#Check the shape (number of columns and rows) in the dataset after data is cleaned*

Rtl\_data.shape

Rtl\_data.head()

*# ## RFM Modelling*

*#Recency = Latest Date - Last Inovice Data, Frequency = count of invoice no. of transaction(s), Monetary = Sum of Total*

*#Amount for each customer*

import datetime as dt

*#Set Latest date 2011-12-10 as last invoice date was 2011-12-09. This is to calculate the number of days from recent purchase*

Latest\_Date = dt.datetime(2011,12,10)

*#Create RFM Modelling scores for each customer*

RFMScores = Rtl\_data.groupby('CustomerID').agg({'InvoiceDate': lambda x: (Latest\_Date - x.max()).days, 'InvoiceNo': lambda x: len(x), 'TotalAmount': lambda x: x.sum()})

*#Convert Invoice Date into type int*

RFMScores['InvoiceDate'] = RFMScores['InvoiceDate'].astype(int)

*#Rename column names to Recency, Frequency and Monetary*

RFMScores.rename(columns={'InvoiceDate': 'Recency',

'InvoiceNo': 'Frequency',

'TotalAmount': 'Monetary'}, inplace=True)

RFMScores.reset\_index().head()

*#Descriptive Statistics (Recency)*

RFMScores.Recency.describe()

*#Recency distribution plot*

import seaborn as sns

x = RFMScores['Recency']

ax = sns.distplot(x)

*#Descriptive Statistics (Frequency)*

RFMScores.Frequency.describe()

*#Frequency distribution plot, taking observations which have frequency less than 1000*

import seaborn as sns

x = RFMScores.query('Frequency < 1000')['Frequency']

ax = sns.distplot(x)

*#Descriptive Statistics (Monetary)*

RFMScores.Monetary.describe()

*#Monateray distribution plot, taking observations which have monetary value less than 10000*

import seaborn as sns

x = RFMScores.query('Monetary < 10000')['Monetary']

ax = sns.distplot(x)

*#Split into four segments using quantiles*

quantiles = RFMScores.quantile(q=[0.25,0.5,0.75])

quantiles = quantiles.to\_dict()

quantiles

*#Functions to create R, F and M segments*

def RScoring(x,p,d):

if x <= d[p][0.25]:

return 1

elif x <= d[p][0.50]:

return 2

elif x <= d[p][0.75]:

return 3

else:

return 4

def FnMScoring(x,p,d):

if x <= d[p][0.25]:

return 4

elif x <= d[p][0.50]:

return 3

elif x <= d[p][0.75]:

return 2

else:

return 1

*#Calculate Add R, F and M segment value columns in the existing dataset to show R, F and M segment values*

RFMScores['R'] = RFMScores['Recency'].apply(RScoring, args=('Recency',quantiles,))

RFMScores['F'] = RFMScores['Frequency'].apply(FnMScoring, args=('Frequency',quantiles,))

RFMScores['M'] = RFMScores['Monetary'].apply(FnMScoring, args=('Monetary',quantiles,))

RFMScores.head()

*#Calculate and Add RFMGroup value column showing combined concatenated score of RFM*

RFMScores['RFMGroup'] = RFMScores.R.map(str) + RFMScores.F.map(str) + RFMScores.M.map(str)

*#Calculate and Add RFMScore value column showing total sum of RFMGroup values*

RFMScores['RFMScore'] = RFMScores[['R', 'F', 'M']].sum(axis = 1)

RFMScores.head()

*#Assign Loyalty Level to each customer*

Loyalty\_Level = ['Platinum', 'Gold', 'Silver', 'Bronze']

Score\_cuts = pd.qcut(RFMScores.RFMScore, q = 4, labels = Loyalty\_Level)

RFMScores['RFM\_Loyalty\_Level'] = Score\_cuts.values

RFMScores.reset\_index().head()

*#Validate the data for RFMGroup = 111*

RFMScores[RFMScores['RFMGroup']=='111'].sort\_values('Monetary', ascending=False).reset\_index().head(10)

import chart\_studio as cs

import plotly.offline as po

import plotly.graph\_objs as gobj

*#Recency Vs Frequency*

graph = RFMScores.query("Monetary < 50000 and Frequency < 2000")

plot\_data = [

gobj.Scatter(

x=graph.query("RFM\_Loyalty\_Level == 'Bronze'")['Recency'],

y=graph.query("RFM\_Loyalty\_Level == 'Bronze'")['Frequency'],

mode='markers',

name='Bronze',

marker= dict(size= 7,

line= dict(width=1),

color= 'blue',

opacity= 0.8

)

),

gobj.Scatter(

x=graph.query("RFM\_Loyalty\_Level == 'Silver'")['Recency'],

y=graph.query("RFM\_Loyalty\_Level == 'Silver'")['Frequency'],

mode='markers',

name='Silver',

marker= dict(size= 9,

line= dict(width=1),

color= 'green',

opacity= 0.5

)

),

gobj.Scatter(

x=graph.query("RFM\_Loyalty\_Level == 'Gold'")['Recency'],

y=graph.query("RFM\_Loyalty\_Level == 'Gold'")['Frequency'],

mode='markers',

name='Gold',

marker= dict(size= 11,

line= dict(width=1),

color= 'red',

opacity= 0.9

)

),

gobj.Scatter(

x=graph.query("RFM\_Loyalty\_Level == 'Platinum'")['Recency'],

y=graph.query("RFM\_Loyalty\_Level == 'Platinum'")['Frequency'],

mode='markers',

name='Platinum',

marker= dict(size= 13,

line= dict(width=1),

color= 'black',

opacity= 0.9

)

),

]

plot\_layout = gobj.Layout(

yaxis= {'title': "Frequency"},

xaxis= {'title': "Recency"},

title='Segments'

)

fig = gobj.Figure(data=plot\_data, layout=plot\_layout)

po.iplot(fig)

*#Frequency Vs Monetary*

graph = RFMScores.query("Monetary < 50000 and Frequency < 2000")

plot\_data = [

gobj.Scatter(

x=graph.query("RFM\_Loyalty\_Level == 'Bronze'")['Frequency'],

y=graph.query("RFM\_Loyalty\_Level == 'Bronze'")['Monetary'],

mode='markers',

name='Bronze',

marker= dict(size= 7,

line= dict(width=1),

color= 'blue',

opacity= 0.8

)

),

gobj.Scatter(

x=graph.query("RFM\_Loyalty\_Level == 'Silver'")['Frequency'],

y=graph.query("RFM\_Loyalty\_Level == 'Silver'")['Monetary'],

mode='markers',

name='Silver',

marker= dict(size= 9,

line= dict(width=1),

color= 'green',

opacity= 0.5

)

),

gobj.Scatter(

x=graph.query("RFM\_Loyalty\_Level == 'Gold'")['Frequency'],

y=graph.query("RFM\_Loyalty\_Level == 'Gold'")['Monetary'],

mode='markers',

name='Gold',

marker= dict(size= 11,

line= dict(width=1),

color= 'red',

opacity= 0.9

)

),

gobj.Scatter(

x=graph.query("RFM\_Loyalty\_Level == 'Platinum'")['Frequency'],

y=graph.query("RFM\_Loyalty\_Level == 'Platinum'")['Monetary'],

mode='markers',

name='Platinum',

marker= dict(size= 13,

line= dict(width=1),

color= 'black',

opacity= 0.9

)

),

]

plot\_layout = gobj.Layout(

yaxis= {'title': "Monetary"},

xaxis= {'title': "Frequency"},

title='Segments'

)

fig = gobj.Figure(data=plot\_data, layout=plot\_layout)

po.iplot(fig)

*#Recency Vs Monetary*

graph = RFMScores.query("Monetary < 50000 and Frequency < 2000")

plot\_data = [

gobj.Scatter(

x=graph.query("RFM\_Loyalty\_Level == 'Bronze'")['Recency'],

y=graph.query("RFM\_Loyalty\_Level == 'Bronze'")['Monetary'],

mode='markers',

name='Bronze',

marker= dict(size= 7,

line= dict(width=1),

color= 'blue',

opacity= 0.8

)

),

gobj.Scatter(

x=graph.query("RFM\_Loyalty\_Level == 'Silver'")['Recency'],

y=graph.query("RFM\_Loyalty\_Level == 'Silver'")['Monetary'],

mode='markers',

name='Silver',

marker= dict(size= 9,

line= dict(width=1),

color= 'green',

opacity= 0.5

)

),

gobj.Scatter(

x=graph.query("RFM\_Loyalty\_Level == 'Gold'")['Recency'],

y=graph.query("RFM\_Loyalty\_Level == 'Gold'")['Monetary'],

mode='markers',

name='Gold',

marker= dict(size= 11,

line= dict(width=1),

color= 'red',

opacity= 0.9

)

),

gobj.Scatter(

x=graph.query("RFM\_Loyalty\_Level == 'Platinum'")['Recency'],

y=graph.query("RFM\_Loyalty\_Level == 'Platinum'")['Monetary'],

mode='markers',

name='Platinum',

marker= dict(size= 13,

line= dict(width=1),

color= 'black',

opacity= 0.9

)

),

]

plot\_layout = gobj.Layout(

yaxis= {'title': "Monetary"},

xaxis= {'title': "Recency"},

title='Segments'

)

fig = gobj.Figure(data=plot\_data, layout=plot\_layout)

po.iplot(fig)

*# ## K-Means Clustering*

*#Handle negative and zero values so as to handle infinite numbers during log transformation*

def handle\_neg\_n\_zero(num):

if num <= 0:

return 1

else:

return num

*#Apply handle\_neg\_n\_zero function to Recency and Monetary columns*

RFMScores['Recency'] = [handle\_neg\_n\_zero(x) for x in RFMScores.Recency]

RFMScores['Monetary'] = [handle\_neg\_n\_zero(x) for x in RFMScores.Monetary]

*#Perform Log transformation to bring data into normal or near normal distribution*

Log\_Tfd\_Data = RFMScores[['Recency', 'Frequency', 'Monetary']].apply(np.log, axis = 1).round(3)

*#Data distribution after data normalization for Recency*

Recency\_Plot = Log\_Tfd\_Data['Recency']

ax = sns.distplot(Recency\_Plot)

*#Data distribution after data normalization for Frequency*

Frequency\_Plot = Log\_Tfd\_Data.query('Frequency < 1000')['Frequency']

ax = sns.distplot(Frequency\_Plot)

*#Data distribution after data normalization for Monetary*

Monetary\_Plot = Log\_Tfd\_Data.query('Monetary < 10000')['Monetary']

ax = sns.distplot(Monetary\_Plot)

from sklearn.preprocessing import StandardScaler

*#Bring the data on same scale*

scaleobj = StandardScaler()

Scaled\_Data = scaleobj.fit\_transform(Log\_Tfd\_Data)

*#Transform it back to dataframe*

Scaled\_Data = pd.DataFrame(Scaled\_Data, index = RFMScores.index, columns = Log\_Tfd\_Data.columns)

from sklearn.cluster import KMeans

sum\_of\_sq\_dist = {}

for k in range(1,15):

km = KMeans(n\_clusters= k, init= 'k-means++', max\_iter= 1000)

km = km.fit(Scaled\_Data)

sum\_of\_sq\_dist[k] = km.inertia\_

*#Plot the graph for the sum of square distance values and Number of Clusters*

sns.pointplot(x = list(sum\_of\_sq\_dist.keys()), y = list(sum\_of\_sq\_dist.values()))

plt.xlabel('Number of Clusters(k)')

plt.ylabel('Sum of Square Distances')

plt.title('Elbow Method For Optimal k')

plt.show()

*#Perform K-Mean Clustering or build the K-Means clustering model*

KMean\_clust = KMeans(n\_clusters= 3, init= 'k-means++', max\_iter= 1000)

KMean\_clust.fit(Scaled\_Data)

*#Find the clusters for the observation given in the dataset*

RFMScores['Cluster'] = KMean\_clust.labels\_

RFMScores.head()

from matplotlib import pyplot as plt

plt.figure(figsize=(7,7))

*##Scatter Plot Frequency Vs Recency*

Colors = ["red", "green", "blue"]

RFMScores['Color'] = RFMScores['Cluster'].map(lambda p: Colors[p])

ax = RFMScores.plot(

kind="scatter",

x="Recency", y="Frequency",

figsize=(10,8),

c = RFMScores['Color']

)

RFMScores.head()

**Observations:**

RFM model has 4 clusters whereas in K-Means there are only 3 clusters which can be derived from the elbow graph