**CUSTOMER SEGMENTATION**

**A PROJECT REPORT**

In partial fulfilment of the requirements for the award of the degree

**MASTER OF COMPUTER APPLICATION**

**Of**

Maulana Abul Kalam Azad University of Technology

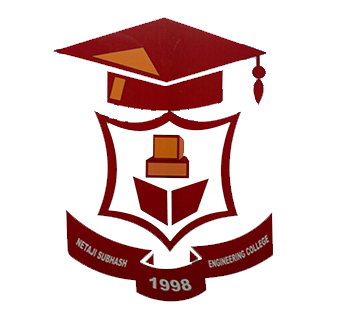
**BY**

**SWARNADEEP MONDAL**

**ROLL : 29101018018**

Under the guidance of

Dr. Atanu Das

****

**NETAJI SUBHASH ENGINEERING COLLEGE  
TECHNO CITY, GARIA, KOLKATA – 700152**

**Academic Year of Pass out: 2020-2021**

**DECLARATION**

We hereby declare that the project work being presented in the project proposal entitled **“CUSTOMER SEGMENTATION USING PYTHON WITH MACHINE LEARNING”** in partial fulfilment of the requirements for the award of the degree of **MASTER OF COMPUTER APPLICATION** at **NETAJI SUBHASH ENGINEERING COLLEGE** underMAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY is an authentic work carried out under the guidance of Dr. Atanu Das. The matter embodied in this project work has not been submitted elsewhere for the award of any degree of our knowledge and belief.

**Date:**

**Name of the Student:** Swarnadeep Mondal

(ROLL:29101018018)

**Signature of the students:**

**CERTIFICATE**

This is to certify that this proposal of minor project entitled **“CUSTOMER SEGMENTATION USING PYTHON WITH MACHINE LEARNING”** is a record of bona fide work, carried out by **Swarnadeep Mondal** under my guidance at **NETAJI SUBHASH ENGINEERING COLLEGE**. In my opinion, the report in its present form is in partial fulfilment ofthe requirements for the award of the degree of **MASTER OF COMPUTER APPLICATION** and as per regulations of the **NETAJI SUBHASH ENGINEERING COLLEGE.** To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report.

**Guide / Supervisor**

------------------------------------------------

Dr. Atanu Das

Sd/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**Head of the Department  
Dr. Atanu Das  
Netaji Subhash Engineering College  
Techno City, Garia, Kolkata – 700152**

**CERTIFICATE OF APPROVAL**

We hereby approve this dissertation titled  
**Customer Segmentation Using Python with Machine Learning**Carried out by

**Name : Swarnadeep Mondal**

**Roll : 29101018018**

**Reg.No. : 182910510015 of 2018-2019**

Under the guidance of  
**Dr. Atanu Das**

Of Netaji Subhash Engineering College, Kolkata in partial fulfilment of requirements for award of the Master of Computer Application of Maulana Abul Kalam Azad University of Technology

**Date: ..........................  
Examiner’s Signatures**

**1. …………………………………………**

**2. …………………………………………**

**3. …………………………………………**

**ACKNOWLEDGEMENT**

The achievement that is associated with the successful Completion of  
any task would be incomplete without mentioning the names of those  
people whose endless cooperation made it possible. The continuous  
support driven, by our Faculty Coordinator **Dr. Atanu Das**. His constant guidance and encouragement made all our efforts successful. We take this  
opportunity to express our deep gratitude towards our project mentor  
**Mr. Sofikul Mullick**, Project Engineer at Engineers Study Center,for giving such valuable suggestions guidance and  
encouragement during the development of this project work. Last but  
not the least we are grateful to all the faculty members of **Engineering  
Study Centre** for their support.

Swarnadeep Mondal

**Date :…………………..**

**CONTENT**

1.Abstract

2.Introduction

2.1. Dataset Description

2.2. Purpose

2.3. Objectives

2.4. Exploratory Data Analysis

2.5. Characteristic Relations

3. Methodology

3.1. Jupyter Notebook

3.2. Pandas

3.3. Numpy

3.4. Matplotlib

3.5. Scikit Learn

3.6. Seaborn

4. Implementation

4.1. What is Clustering?  
4.2. K-Means Clustering  
4.3. Modeling

4.4. Elbow Method

5. Project Limitations

6. Future Scope

7. Summary

8. Bibliography

**ABSTRACT**

Machine learning techniques have proven good performance in classification matters of all kinds: medical diagnosis, character recognition, credit default and fraud prediction, and also foreign exchange market prognosis. Customer segmentation in private banking sector, E-commerce websites is an important step for profitable and efficient business development, enabling financial institutions to address their products and services to homogeneous classes of customers. Customer segmentation plays an important role in customer relationship management. It allows companies to design and establish different strategies to maximize the value of customers. Customer segmentation refers to grouping customers into different categories based on shared characteristics such as age, location, spending habits, loyalty and so on. Similarly, clustering means putting things together in such a way that similar type of things remain in the same group. Using clustering techniques, companies can identify the several segments of customers allowing them to target the potential user base. In this machine learning project, we will make use of K-means clustering which is the essential algorithm for clustering unlabelled dataset.

**INTRODUCTION**

**Python** is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object oriented approach aim to help programmers write clear, logical code for small and large-scale projects

**Machine learning** (**ML**) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data”, in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

**Clustering** is a Machine Learning technique that involves the grouping of data points. In theory, data points that are in the same group should have similar properties and/or features, while data points in different groups should have highly dissimilar properties and/or features.

**Customer Segmentation** is fundamental to every Business. It allows business to provide targeted product marketing.

Lets see how we can use **RFM** and **K-mean** clustering analysis for customer segmentation.

**2.1. Dataset Description**

The dataset is is extracted from UCI’s Machine Learning Repository uploaded by **Dr. Daqing Chen.**

Download Link : <https://data.world/uci/online-retail>

* The dataset contains transactions occurring between 01/12/2010 and 09/12/2011
* No. of Observation : 541,909
* No. of features: 8
* No. of Transactions: 22,190
* No. of Customers: 4,372
* No. of Products: 3,684

|  |  |
| --- | --- |
| **Features** | **Description** |
| **InvoiceNo** | 6-digit integral number uniquely assigned to each transaction |
| **StockCode** | 5-digit integral number uniquely assigned to each |
| **Description** | Product name |
| **Quantity** | The quantities of each product (item) per transaction |
| **InvoiceDate** | The day and time when each transaction was generated |
| **UnitPrice** | Product price per unit in sterling £ |
| **CustomerID** | 5-digit integral number uniquely assigned to each customer |
| **Country** | The name of the country where each customer resides |

**2.2. Project Goal**

To be on par with the competition in market online (E-commerce Websites) and offline (Retail Shops) companies wants to understand their customers buying preferences, buying habits, buying frequencies etc. in order to:

* Pinpoint their marketing and sales strategy.
* Differentiate between their different target audience to promote their products.
* Determine the customer loyalty to the company and take actions accordingly.

To achieve the above goals we need to analyze the companies transactional dataset to create segments based on the customers purchase history

**METHODOLOGY**

**3.1. Jupyter Notebook**The Jupyter Notebook is an open-source web application that allows you to create and  
share documents that contain live code, equations, visualizations and narrative text. Uses  
include: data cleaning and transformation, numerical simulation, statistical modeling, data  
visualization, machine learning, and much more.  
In our project we used following packages:  
• Pandas (version : 1.1.5)  
• Numpy (version : 1.19.2)  
• Matplotlib (version : 3.3.2)  
• Scikit Learn (version : 0.23.2)  
• Seaborn (version : 0.11.1)

**3.2. Pandas**

Pandas is a software library written for the Python programming language for data  
manipulation and analysis. In particular, it offers data structures and operations for  
manipulating numerical tables and time series. It is free software released under the three-clause  
BSD license. The name is derived from the term "panel data", an econometrics term for data  
sets that include observations over multiple time periods for the same individuals. Its name is a  
play on the phrase "Python data analysis" itself. Wes McKinney started building what would  
become pandas at AQR Capital while he was a researcher there from 2007 to 2010.

**3.3. Numpy**

NumPy is the fundamental package for scientific computing in Python. It is a Python  
library that provides a multidimensional array object, various derived objects (such as masked  
arrays and matrices), and an assortment of routines for fast operations on arrays, including  
mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms,  
basic linear algebra, basic statistical operations, random simulation and much more.  
At the core of the NumPy package, is the ndarray object. This encapsulates ndimensional arrays of homogeneous data types, with many operations being performed in  
compiled code for performance.

**3.4. Matplotlib**

Matplotlib is a plotting library for the Python programming language and its numerical  
mathematics extension NumPy. It provides an object-oriented API for embedding plots into  
applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is  
also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely  
resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

**3.5. Scikit Learn**

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in  
Python. It provides a selection of efficient tools for machine learning and statistical modeling  
including classification, regression, clustering and dimensionality reduction via a consistence  
interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy  
and Matplotlib.

**3.6. Seaborn**

Seaborn is a library for making statistical graphics in Python. It builds on top of  
matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and  
understand your data. Its plotting functions operate on data frames and arrays containing whole  
datasets and internally perform the necessary semantic mapping and statistical aggregation to  
produce informative plots. Its dataset-oriented, declarative API lets you focus on what the  
different elements of your plots mean, rather than on the details of how to draw them.

**IMPLEMENTATION**

**4.1. What is Clustering?**

Imagine that you have a group of chocolates and liquorice candies. You are  
required to separate the two eatables. Intuitively, you are able to separate them based on  
their appearances. The process of segregating objects into groups based on their  
respective characteristics is called clustering. In clusters, the features of objects in a  
group are similar to other objects present in the same group.  
Clustering is used in various fields like image recognition, pattern analysis,  
medical informatics, genomics, data compression etc. It is part of the unsupervised  
learning algorithm in machine learning. This is because the data-points present are not  
labelled and there is no explicit mapping of input and outputs. As such, based on the  
patterns present inside, clustering takes place.

**4.2. K-Means Clustering**K-Means algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster’s centroid is at the minimum. The less variation we have within clusters, the more homogeneous the data points are within the same cluster.  
We then proceeded to perform K-means Clustering which will create different  
clusters to group similar spending activity based on their age and annual income. KMeans Clustering selects random values from the data and forms clusters assigned. The closest values from the centre of each cluster were taken to update the cluster and reshape the plot (just like k-NN). The closest values are based on Euclidean Distance.

**4.3. Modeling**

**Building the k-means model:**We need to visualize the data which we are going to use for the clustering. This will give us a fair idea about the data we're working on. This will give us a fair Idea and patterns about some of the data.

**4.4. Elbow Method**

The Elbow method runs k-means clustering on the dataset for a range of values for k (say from 1-10) and then for each value of k computes an average score for all clusters. By default, the distortion score is computed, the sum of square distances from each point to its assigned center.  
When these overall metrics for each model are plotted, it is possible to visually determine the best value for k. If the line chart looks like an arm, then the “elbow” (the point of inflection on the curve) is the best value of k. The “arm” can be either up or down, but if there is a strong inflection point, it is a good indication that the underlying model fits best at that point.  
We use the Elbow Method which uses Within Cluster Sum Of Squares (WCSS) against the the number of clusters (K Value) to figure out the optimal number of clusters value. WCSS measures sum of distances of observations from their cluster centroids which is given by the 16 below formula. where Yi is centroid for observation Xi. The main goal is to maximize number of clusters and in  
limiting case each data point becomes its own cluster centroid.  
It is clear, that the optimal number of clusters for our data are 5, as the slope of the curve is not steep enough after it. When we observe this curve, we see that last elbow comes at k = 5, it would be difficult to visualize the elbow if we choose the higher rang.

**PROJECT LIMITATION**

I worked on the backend part of the system thus there is no frontend work associated which can result in a more realistic look and focus on user experience.

**FUTURE SCOPE**

1. Since here only the backend part of the system is built, we can create a custom frontend which can result in a more realistic look and focus on user experience.
2. We can work on the project and use it for greater size dataset.

**SUMMARY**

Companies, Malls, super markets on Small Business Enterprises should carry out Market Basket Analysis for their business. This will enable companies to target specific groups of customers, a customer segmentation model allows for the effective allocation of marketing resources and the maximization of cross- and up-selling opportunities. When a group of customers is sent personalized messages as part of a marketing mix that is designed around their needs, it's easier for companies to send those customers special offers meant to encourage them to buy more products. Customer segmentation can also improve customer service and assist in customer loyalty and retention. As a by-product of its personalized nature, marketing materials sent out using customer segmentation tend to be more valued and appreciated by the customer who receives them as opposed to impersonal brand messaging that doesn't acknowledge purchase history or any kind of customer relationship Finally with customer segmentation Companies will stay a step ahead of competitors in specific sections of the market and identify new products that 21 exist or potential customers could be interested in or improving products to meet customer expectations.

**BIBLIOGRAPHY**

[**https://www.wikipedia.org/**](https://www.wikipedia.org/)

[**https://www.altexsoft.com/blog/datascience/machine-learning-project-structure-stages-roles-and-tools/**](https://www.altexsoft.com/blog/datascience/machine-learning-project-structure-stages-roles-and-tools/)

[**https://stackify.com/aws-lambda-with-python-a-complete-getting-started-guide/**](https://stackify.com/aws-lambda-with-python-a-complete-getting-started-guide/)

[**https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b9**](https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b9)

**CODE**

# ### 1. Importing the libraries and the data

# In[63]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

# ### 2. Importing the data from .csv file

# In[64]:

data = pd.read\_csv('Customer.csv', encoding='unicode\_escape')

data

# In[3]:

data.shape

# In[4]:

#droping all repeatative data from Country,CustomerID Columns

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

#printing the existing data after droping and also sorting the data

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

# In[5]:

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

# In[6]:

import matplotlib.pyplot as plt

fig = plt.figure()

ax = fig.add\_axes([0,0,1,1])

ax.bar(CC['Country'].head(5).values,CC['CustomerID'].head(5).values)

plt.show()

# In[7]:

#hence there is 90% of data is from UK so we are keeping only data of UK

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

data.shape

# In[8]:

data.describe()

# ### 4. Checking the data for inconsistencies and further cleaning the data if needed.

# In[9]:

data.isnull()

# In[10]:

data.isnull().sum()

# In[11]:

#droping the null data

data=data[pd.notnull(data['CustomerID'])]

# In[12]:

data.isnull().sum()

# In[13]:

#validate if there any negative values of Quantity

data.Quantity.min()

# In[14]:

#removing all the negative values in the Quantity Column

data=data[(data['Quantity']>=0)]

data.Quantity.min()

# In[15]:

#validate if there any negative values of UnitPrice

data.UnitPrice.min()

# In[16]:

#calculating the total amount and storing the data into a newly added column

data['TotalAmount']=data['Quantity']\*data['UnitPrice']

data

# In[17]:

data.InvoiceDate

# In[18]:

#Changing the data type of InvoiceDate from object to datetime

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

data.InvoiceDate

# In[19]:

#Searching for start and end date of this dataset

dataset=data.sort\_values(['InvoiceDate'])

dataset

# In[20]:

import datetime as dt

#Hence we want to calculate recency we set the system date as the letest date from the dataset

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

# In[21]:

# Calculating the Recency,Frequency,Monetary

RFMScore=data.groupby('CustomerID').agg({'InvoiceDate': lambda x: (Letest\_Date-x.max()).days, 'InvoiceNo': lambda x: len(x),

'TotalAmount': lambda x: x.sum()})

# In[22]:

#Changing the 'InvoiceDate' datatype from datetime to integer

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

# In[23]:

#Renaming the Column names

RFMScore.rename(columns={'InvoiceDate':'Recency','InvoiceNo':'Frequency','TotalAmount':'Monetary'},inplace=True)

# In[24]:

RFMScore.head()

# In[25]:

RFMScore.Recency.describe()

# In[26]:

#Recency distribution plot

import seaborn as sns

x = RFMScore['Recency']

ax = sns.distplot(x)

# In[27]:

RFMScore.Frequency.describe()

# In[28]:

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

import seaborn as sns

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

ax = sns.distplot(x)

# In[29]:

RFMScore.Monetary.describe()

# In[30]:

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

import seaborn as sns

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

ax = sns.distplot(x)

# In[31]:

#Creating Different quantile level to seggrigate the customer

quantiles=RFMScore.quantile(q=[0.2,0.4,0.6,0.8])

quantiles=quantiles.to\_dict()

# In[32]:

quantiles

# In[33]:

#Defining Function to allote points based on the customers

def RScore(x,p,d):

if x<=d[p][0.2]:

return 4

elif x<=d[p][0.4]:

return 3

elif x<=d[p][0.6]:

return 2

elif x<=d[p][0.8]:

return 1

else:

return 0

#Defining Function to allote points based on the customers

def FScore(x,p,d):

if x<=d[p][0.2]:

return 1

elif x<=d[p][0.4]:

return 2

elif x<=d[p][0.6]:

return 3

elif x<=d[p][0.8]:

return 4

else:

return 5

#Defining Function to allote points based on the customers

def MScore(x,p,d):

if x<=d[p][0.2]:

return 1

elif x<=d[p][0.4]:

return 2

elif x<=d[p][0.6]:

return 3

elif x<=d[p][0.8]:

return 4

else:

return 5

# In[34]:

RFMScore['R']=RFMScore['Recency'].apply(RScore, args=('Recency',quantiles))

RFMScore['F']=RFMScore['Frequency'].apply(RScore, args=('Frequency',quantiles))

RFMScore['M']=RFMScore['Monetary'].apply(MScore, args=('Monetary',quantiles))

# In[61]:

RFMScore.head(7)

# In[60]:

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

RFMScore.head(7)

# In[37]:

#Assign Loyalty Level to each customer

Loyalty\_Level = ['BAD','AVERAGE', 'GOOD', 'VALUABLE', 'PREMIUME']

Score\_cuts = pd.qcut(RFMScore.RFMTotal, q = 5, labels = Loyalty\_Level)

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

RFMScore.reset\_index().head()

# In[38]:

#Validate the data for RFMGroup = 111

RFMScore[RFMScore['RFM\_Loyalty\_Level']=='VALUABLE'].sort\_values('Monetary', ascending=False).reset\_index().head(10)

# In[39]:

import chart\_studio as cs

import plotly.offline as po

import plotly.graph\_objs as gobj

#Recency Vs Frequency

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

plot\_data = [

gobj.Scatter(

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

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

mode='markers',

name='BAD',

marker= dict(size= 10,

line= dict(width=1),

color= 'black',

opacity= 0.6

)

),

gobj.Scatter(

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

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

mode='markers',

name='AVERAGE',

marker= dict(size= 10,

line= dict(width=1),

color= 'red',

opacity= 0.6

)

),

gobj.Scatter(

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

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

mode='markers',

name='GOOD',

marker= dict(size= 10,

line= dict(width=1),

color= 'yellow',

opacity= 0.6

)

),

gobj.Scatter(

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

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

mode='markers',

name='VALUABLE',

marker= dict(size= 10,

line= dict(width=1),

color= 'green',

opacity= 0.6

)

),

gobj.Scatter(

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

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

mode='markers',

name='PREMIUME',

marker= dict(size= 10,

line= dict(width=1),

color= 'blue',

opacity= 0.6

)

),

]

plot\_layout = gobj.Layout(

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

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

title='Segments'

)

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

po.iplot(fig)

# ## Recency Vs Monetary

# In[40]:

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

plot\_data = [

gobj.Scatter(

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

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

mode='markers',

name='BAD',

marker= dict(size= 10,

line= dict(width=1),

color= 'black',

opacity= 0.6

)

),

gobj.Scatter(

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

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

mode='markers',

name='AVERAGE',

marker= dict(size= 10,

line= dict(width=1),

color= 'red',

opacity= 0.6

)

),

gobj.Scatter(

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

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

mode='markers',

name='GOOD',

marker= dict(size= 10,

line= dict(width=1),

color= 'yellow',

opacity= 0.6

)

),

gobj.Scatter(

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

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

mode='markers',

name='VALUABLE',

marker= dict(size= 10,

line= dict(width=1),

color= 'green',

opacity= 0.6

)

),

gobj.Scatter(

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

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

mode='markers',

name='PREMIUME',

marker= dict(size= 10,

line= dict(width=1),

color= 'blue',

opacity= 0.6

)

),

]

plot\_layout = gobj.Layout(

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

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

title='Segments'

)

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

po.iplot(fig)

# ## Frequency Vs Monetary

# In[41]:

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

plot\_data = [

gobj.Scatter(

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

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

mode='markers',

name='BAD',

marker= dict(size= 10,

line= dict(width=1),

color= 'black',

opacity= 0.8

)

),

gobj.Scatter(

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

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

mode='markers',

name='AVERAGE',

marker= dict(size= 10,

line= dict(width=1),

color= 'red',

opacity= 0.6

)

),

gobj.Scatter(

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

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

mode='markers',

name='GOOD',

marker= dict(size= 10,

line= dict(width=1),

color= 'yellow',

opacity= 0.6

)

),

gobj.Scatter(

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

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

mode='markers',

name='VALUABLE',

marker= dict(size= 10,

line= dict(width=1),

color= 'green',

opacity= 0.6

)

),

gobj.Scatter(

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

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

mode='markers',

name='PREMIUME',

marker= dict(size= 10,

line= dict(width=1),

color= 'blue',

opacity= 0.6

)

),

]

plot\_layout = gobj.Layout(

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

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

title='Segments'

)

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

po.iplot(fig)

# In[42]:

#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

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

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

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

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

# In[43]:

#Data distribution after data normalization for Recency

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

ax = sns.distplot(Recency\_Plot)

# In[44]:

#Data distribution after data normalization for Frequency

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

ax = sns.distplot(Frequency\_Plot)

# In[45]:

#Data distribution after data normalization for Monetary

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

ax = sns.distplot(Monetary\_Plot)

# In[46]:

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 = RFMScore.index, columns = Log\_Tfd\_Data.columns)

# In[47]:

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()

# In[48]:

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

KMean\_clust = KMeans(n\_clusters= 3, init= 'k-means++', max\_iter= 10000)

KMean\_clust.fit(Scaled\_Data)

#Find the clusters for the observation given in the dataset

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

RFMScore.head()

# In[55]:

from matplotlib import pyplot as plt

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

##Scatter Plot Frequency Vs Recency

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

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

ax = RFMScore.plot(

kind="scatter",

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

figsize=(10,5),

c = RFMScore['Color']

)

# In[50]:

from matplotlib import pyplot as plt

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

##Scatter Plot Recency Vs Monetary

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

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

ax = RFMScore.plot(

kind="scatter",

x="Recency", y="Monetary",

figsize=(10,8),

c = RFMScore['Color']

)

# In[51]:

from matplotlib import pyplot as plt

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

##Scatter Plot Monetary Vs Frequency

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

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

ax = RFMScore.plot(

kind="scatter",

x="Monetary", y="Frequency",

figsize=(10,8),

c = RFMScore['Color']

)