# CUSTOMER SEGMENTATION REPORT

## Introduction:

Customer segmentation is the practice of dividing a company's customers into groups that reflect similarities among customers in each group. The goal of segmenting customers is to decide how to relate to customers in each segment to maximize the value of each customer to the business. Segmentation allows marketers to better tailor their marketing efforts to various audience subsets. Those efforts can relate to both communications and product development. Customer segmentation relies on identifying key differentiators that divide customers into groups that can be targeted. Information such as a customers' demographics (age, race, religion, gender, family size, ethnicity, income, education level), geography (where they live and work), psychographic (social class, lifestyle and personality characteristics) and behavioural (spending, consumption, usage and desired benefits) tendencies are taken into account when determining customer segmentation practices.

## Dataset:

This is a transnational data set that 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.

- *InvoiceNo: Invoice number. Nominal, a 6-digit integral number uniquely assigned to each transaction. If this code starts with the letter 'c', it indicates a cancellation.*
- *StockCode: Product (item) code. Nominal, a 5-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 5-digit integral number uniquely assigned to each customer.*
- *Country: Country name. Nominal, the name of the country where each customer resides.*

## Pre-processing:

Initially, it is having 5,41,910 Records. In our First step, we will be removing the NA Values from the dataset. Secondly, in some records UnitPrice is 0.0 it should not be like that so, we will remove those values also. In addition to it, we will check on with Cancelled Orders, that orders are not needed, so we remove that also. And in that Cancelled Orders Quantity is in the negative value we need to remove that also. And at last, we will be noticing the records which are remaining in the dataset. Finally, it has 3,92,735 Records. So, we have removed the outliers in the dataset. If we did not do pre-processing in the dataset means the accuracy of the model will go down and the model won’t predict the values accurately.

## Cohort Analysis:

A Cohort simply means that a Group of People have the same Characteristics. We have three types in it and they are,

- *Time Cohorts or Acquisition Cohorts: Groups are divided by First Activity.*
- *Behaviour Cohorts or Segment-Based Cohorts: Groups are divided by their Behaviours and Actions about your Service.*
- *Size Cohorts: Size-Based Cohorts refer to the various sizes of Customers who purchase a company's Products or Services.*

Cohort Analysis is a subset of Behavioural Analytics that takes the data from a given eCommerce platform, web application, or online game and rather than looking at all users as one unit, it breaks them into related groups for analysis. These related Groups, or Cohorts, usually share common characteristics or experiences within a defined Time-Span.

## Pareto Principle:

The Pareto principle states that for many outcomes, roughly 80% of consequences come from 20% of causes (the "Vital Few").

Other names for this principle are the 80/20 rule, the law of the vital few, or the principle of factor sparsity.

Let's implement Pareto's 80-20 rule to our dataset.

We have two hypotheses:

1. *80% of the Company's revenue comes from 20% of Total Customers.*
2. *80% of the Company's revenue comes from 20% of Total Products.*

## RFM Analysis:

Recency, Frequency, Monetary Value is a marketing analysis tool used to identify a company's or an organization's best customers by using certain measures. The RFM model is based on three quantitative factors:

- *Recency: How recently a customer has made a purchase.*
- *Frequency: How often a customer makes a purchase.*
- *Monetary Value: How much money a customer spends on purchases.*

RFM analysis numerically ranks a customer in each of these three categories, generally on a scale of 1 to 5 (the higher the number, the better the result). The "best" customer would receive a top score in every category.

## Clustering:

What is Clustering?

A way of grouping the data points into different clusters, consisting of similar data points.

Clustering or cluster analysis is a machine learning technique, which groups the unlabelled dataset.

What is Davis Bouldin Score?

The score is defined as the average similarity measure of each cluster with its most similar cluster, where similarity is the ratio of within-cluster distances to between-cluster distances. Thus, clusters that are farther apart and less dispersed will result in a better score. In this dataset, for the accuracy metrics, we have used Davis Bouldin Score as the Metrics.

In this dataset, we have used 9 Clustering Algorithms namely,

1. *Spectral Clustering*
2. *Mini-Batch K-Means Clustering*
3. *Agglomerative Clustering*
4. *DBSCAN Clustering*
5. *BIRCH Clustering*
6. *Affinity Propagation Clustering*
7. *Gaussian Mixture Clustering*
8. *K-Means Clustering*
9. *OPTICS Clustering*

We will be seeing the result based on these clustering algorithms only.

## Code:

Note: The Code is Interpreted in the Google Colab.

**Pre-Processing:**

```
from numpy import unique
from numpy import where
import numpy as np

import seaborn as sns
import pandas as pd

from matplotlib.ticker import PercentFormatter
import matplotlib.pyplot as plt

import datetime as dt

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering, Birch,
 OPTICS
from sklearn.cluster import AffinityPropagation, SpectralClustering, MeanSh
ift, MiniBatchKMeans

from sklearn.mixture import GaussianMixture

from sklearn.metrics import davies_bouldin_score

from yellowbrick.cluster import KElbowVisualizer

from itertools import combinations

pd.options.mode.chained_assignment = None
```

```
plt.rcParams["axes.facecolor"] = "#A2A2A2"
plt.rcParams["axes.grid"] = 1
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
df=pd.read_csv("/content/drive/MyDrive/Online Retail.csv",encoding= 'unicode_escape')
print("First Five Values of the Dataset.")
display(df.head())
print("**********************************")
print("Shape of the Dataset is",df.shape)
print("**********************************")
print("Last Five Values of the Dataset.")
display(df.tail())
```

```
df.info()
```

```
display(df.isnull().sum())
```

```
df[df.Description.isnull()]
```

```
df[df.Description.isnull()].CustomerID.nunique()
```

```
df[df.Description.isnull()].UnitPrice.value_counts()
```

```
df=df[df.Description.notnull()]
```

```
df[df.CustomerID.isnull()]
```

```
print("We had {} Observations Previously.".format(df.shape[0]))
df=df[df.CustomerID.notnull()]
print("Currently, We are having {} Observations after Removing Unknown Customers.".format(df.shape[0]))
```

```
df.isnull().sum()
```

```
df.shape
```

```
df[df.Description.str.len()<5]
```

```
df.InvoiceNo.value_counts()
```

```
df[df["InvoiceNo"].str.startswith("C")]
```

```
df['Cancelled']=df['InvoiceNo'].apply(lambda x:1 if x.startswith("C") else 0)
```

```
cancelled_invoiceNo=df[df.Cancelled==1].InvoiceNo.tolist()
cancelled_invoiceNo=[x[1:] for x in cancelled_invoiceNo]
cancelled_invoiceNo[:10]
```

```
df[df['InvoiceNo'].isin(cancelled_invoiceNo)]
```

```
df[df.InvoiceNo.str.len()!=6]
```

```
df=df[df.Cancelled==0]
```

```
df[df.StockCode.str.contains("^[a-zA-Z]")].StockCode.value_counts()
```

```
df[df.StockCode.str.contains("^[a-zA-Z]")].Description.value_counts()
```

```
df[df.StockCode.str.len()>5].StockCode.value_counts()
```

```
df[df.StockCode.str.len()>5].Description.value_counts()
```

```
df=df[~df.StockCode.str.contains("^[a-zA-Z]")]
df['Description']=df['Description'].str.lower()
```

```
df.groupby("StockCode")["Description"].nunique()[df.groupby("StockCode")["D
escription"].nunique()!=1]
```

```
df[df.StockCode=="16156L"].Description.value_counts()
```

```
df[df.StockCode=="17107D"].Description.value_counts()
```

```
df[df.StockCode=="85184C"].Description.value_counts()
```

```
df[df.StockCode=="90014C"].Description.value_counts()
```

```
df.CustomerID.value_counts()
```

```
customer_counts=df.CustomerID.value_counts().sort_values(ascending=False).h
ead(30)
fig,ax=plt.subplots(figsize=(10,8))
sns.barplot(y=customer_counts.index,x=customer_counts.values,orient='h',ax=
ax,order=customer_counts.index,palette='Reds_r')
plt.title("Customers that have Most Transactions")
plt.ylabel("Customers")
plt.xlabel("Transaction Count")
plt.show()
```

```
df.Country.value_counts()
```

```
country_counts=df.Country.value_counts().sort_values(ascending=False).head(
30)
```

```
fig,ax=plt.subplots(figsize=(18,10))
sns.barplot(y=country_counts.index,x=country_counts.values,orient='h',ax=ax
,order=country_counts.index,palette='Blues_r')
plt.title("Countries that have Most Transactions")
plt.xscale("Log")
plt.show()
```

```
df['UnitPrice'].describe()
```

```
df[df.UnitPrice==0].head()
```

```
print("We had {} Observations Previously.".format(df.shape[0]))
df=df[df.UnitPrice>0]
print("Currently, We are having {} Observations after Removing Records that
 have 0 Unit Price.".format(df.shape[0]))
```

```
fig, axes = plt.subplots(1, 3, figsize = (18, 6))
sns.kdeplot(df["UnitPrice"],ax=axes[0],color="#195190").set_title("Distribu
tion of Unit Price")
sns.boxplot(y=df["UnitPrice"],ax=axes[1],color="#195190").set_title("Boxplo
t for Unit Price")
sns.kdeplot(np.log(df["UnitPrice"]),ax=axes[2],color="#195190").set_title("
Log Unit Price Distribution")
plt.show()
```

```
print("Lower Limit for UnitPrice: "+str(np.exp(-2)))
print("Upper Limit for UnitPrice: "+str(np.exp(3)))
```

```
np.quantile(df.UnitPrice,0.99)
```

```
print("We had {} Observations Previously.".format(df.shape[0]))
df=df[(df.UnitPrice>0.1)&(df.UnitPrice<20)]
print("Currently, We are having {} Observations after Removing Unit Prices
Smaller than 0.1 and Greater than 20.".format(df.shape[0]))
```

```
fig,axes= plt.subplots(1,3,figsize=(18,6))
sns.kdeplot(df["UnitPrice"],ax=axes[0],color="#195190").set_title("Distribu
tion of Unit Price")
sns.boxplot(y=df["UnitPrice"],ax=axes[1],color="#195190").set_title("Boxplo
t for Unit Price")
sns.kdeplot(np.log(df["UnitPrice"]),ax=axes[2],color="#195190").set_title("
Log Unit Price Distribution")
fig.suptitle("Distribution of Unit Price (After Removing Outliers)")
plt.show()
```

```
df["Quantity"].describe()
```

```
fig,axes=plt.subplots(1,3,figsize=(18,6))
```

```
sns.kdeplot(df["Quantity"],ax=axes[0],color="#195190").set_title("Distribution of Quantity")
sns.boxplot(y=df["Quantity"],ax=axes[1],color="#195190").set_title("Boxplot for Quantity")
sns.kdeplot(np.log(df["Quantity"]),ax=axes[2],color="#195190").set_title("Log Quantity")
plt.show()
```

```
print("Upper Limit for Quantity: "+str(np.exp(5)))
```

```
np.quantile(df.Quantity,0.99)
```

```
print("We had {} Observations Previously.".format(df.shape[0]))
df=df[(df.Quantity<150)]
print("Currently, We are having {} Observations after Removing Quantities Greater than 150.".format(df.shape[0]))
```

```
df["TotalPrice"]=df["Quantity"]*df["UnitPrice"]
df['InvoiceDate']=pd.to_datetime(df['InvoiceDate'])
df['InvoiceDate']
```

```
df.drop("Cancelled",axis=1,inplace=True)
```

```
df.shape
```

```
df.to_csv("Online_Retail_Cleaned.csv",index=False)
```

**Cohort Analysis:**

```
print("Minimum Date: {} \nMaximum Date: {}".format(df.InvoiceDate.min(),df.InvoiceDate.max()))
print("Time Difference is: {}".format(df.InvoiceDate.max()-df.InvoiceDate.min()))
```

```
def get_month(x):
    return dt.datetime(x.year,x.month,1)
def get_dates(df,col):
    year=df[col].dt.year
    month=df[col].dt.month
    day=df[col].dt.day
    return year,month,day
```

```
df["InvoiceMonth"]=df["InvoiceDate"].apply(get_month)
df["CohortMonth"]=df.groupby("CustomerID")["InvoiceMonth"].transform("min")
```

```
df.head()
```

```
invoice_year,invoice_month,invoice_day=get_dates(df,"InvoiceMonth")
cohort_year,cohort_month,cohort_day=get_dates(df,"CohortMonth")
year_diff=invoice_year-cohort_year
month_diff=invoice_month-cohort_month
```

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

```
cohort_data=df.groupby(["CohortIndex","CohortMonth"])["CustomerID"].nunique
().reset_index()
cohort_pivot=cohort_data.pivot(index="CohortMonth",columns="CohortIndex",va
lues="CustomerID")
cohort_pivot
```

```
cohort_sizes=cohort_pivot.iloc[:,0]
retention=cohort_pivot.divide(cohort_sizes,axis=0)
retention.index=retention.index.strftime("%Y-%m")
retention
```

```
plt.rcParams["axes.facecolor"]="white"
fig,ax=plt.subplots(figsize=(14,10))
sns.heatmap(retention,cmap="Blues",annot=True,fmt=".2%",annot_kws={"fontsiz
e":12},cbar=False,ax=ax)
plt.title("Retention Rate Percentages - Monthly Cohorts")
plt.yticks(rotation=0)
plt.show()
```

```
customer_per_month=df.groupby("CohortMonth")["CustomerID"].nunique().values
customers=customer_per_month.cumsum()
customers=customers[::-1]
customers
```

```
customer_in_month=df.groupby("CohortIndex")["CustomerID"].nunique()
customer_in_month
```

```
plt.rcParams["axes.facecolor"]="White"
fig,ax=plt.subplots(figsize=(14,8),facecolor="#A2A2A2")
ax.grid(False)
x=customer_in_month.index
y=100*(customer_in_month/customers)
sns.lineplot(x=x,y=y,color="#101820",marker="o",markerfacecolor="#0EB8F1",m
arkeredgecolor="#000000")
for x,y in zip(x,y):
    plt.text(x,y+2,s=str(round(y,2))+"%")
plt.xlabel("Cohort Index")
plt.ylabel("Retention Rate %")
plt.title("Monthly Retention Rates for All Customers")
sns.despine()
plt.show()
```

```
monthly_customer_price_df=df.groupby("InvoiceMonth").agg({"TotalPrice":"sum
","CustomerID":"nunique"})
monthly_customer_price_df
```

```
fig,ax=plt.subplots(figsize=(16, 8),facecolor="#A2A2A2")
ax.set_facecolor("White")
```

```
sns.barplot(x=np.arange(len(monthly_customer_price_df.index)),y=monthly_cus
tomer_price_df.TotalPrice,ax=ax,color="#101820")
ax2=ax.twinx()
sns.lineplot(x=np.arange(len(monthly_customer_price_df.index)),y=monthly_cu
stomer_price_df.CustomerID,ax=ax2,color="#F1480F",marker="o",markerfacecolo
r="#0EB8F1",markeredgecolor="#000000")
ax.set_yticks([])
ax2.set_yticks([])
ax2.set_ylabel("Total Customer")
ax.set_ylabel("Total Price")
plt.title("Revenue & Customer Count per Month")
ax.text(-
0.75,1000000,"Bars represents Revenue \nLine represents Unique Customer Cou
nt",fontsize=13,alpha=0.8)
for x,y in zip(np.arange(len(monthly_customer_price_df.index)),monthly_cust
omer_price_df.CustomerID):
    ax2.text(x-0.1,y+20,y,color="white")
sns.despine(left=True,right=True,bottom=True,top=True)
plt.show()
```

**Pareto Principle:**

```
def prepare_pareto_data(df,col,price):
    df_price=pd.DataFrame(df.groupby(col)[price].sum())
    df_price=df_price.sort_values(price,ascending=False)
    df_price["CumulativePercentage"]=(df_price[price].cumsum()/df_price[pri
ce].sum()*100).round(2)
    return df_price
```

```
def create_pareto_plot(df,col,price,log=True):
    plt.rcParams["axes.facecolor"]="White"
    fig,ax=plt.subplots(figsize=(15,5),dpi=150,facecolor="#A2A2A2")
    plt.rcParams["axes.grid"]=False
    if log==True:
        sns.barplot(x=np.arange(len(df)),y=np.log(df[price]),ax=ax,color="#
101820")
        ax.set_ylabel("Total Price (Log - Scale)")
    else:
        sns.barplot(x=np.arange(len(df)),y=df[price],ax=ax,color="#101820")
    ax2=ax.twinx()
    sns.lineplot(x=np.arange(len(df)),y=df.CumulativePercentage,ax=ax2,colo
r="#0019AA")
    ax2.axhline(80,color="#008878",linestyle="dashed",alpha=1)
    ax2.axhline(90,color="#008878",linestyle="dashed",alpha=0.75)
    vlines=[int(len(df)*x/10) for x in range(1, 10)]
    for vline in vlines:
        ax2.axvline(vline,color="#008878",linestyle="dashed",alpha=0.1)
    interaction_80=(df.shape[0]-df[df.CumulativePercentage>=80].shape[0])
    ax2.axvline(interaction_80,color="#008878",linestyle="dashed",alpha=1)
    interaction_80_percentage=round((interaction_80/df.shape[0])*100)
    plt.text(interaction_80+25,95,str(interaction_80_percentage)+"%")
    prop=dict(arrowstyle="-|>",color="#000000",lw=1.5,ls="--")
```

```
    plt.annotate("",xy=(interaction_80-
10,80),xytext=(interaction_80+120,73),arrowprops=prop)
    interaction_90=(df.shape[0]-df[df.CumulativePercentage>=90].shape[0])
    ax2.axvline(interaction_90,color="#008878",linestyle="dashed",alpha=0.8
)
    interaction_90_percentage=round((interaction_90/df.shape[0])*100)
    plt.text(interaction_90+25,95,str(interaction_90_percentage)+"%")
    plt.annotate("",xy=(interaction_90-
10,90),xytext=(interaction_90+120,83),arrowprops=prop)
    ax2.yaxis.set_major_formatter(PercentFormatter())
    ax.set_yticks([])
    plt.xticks([])
    ax.set_ylabel("Revenue")
    ax2.set_ylabel("Cumulative Percentage")
    subject="Customers" if col=="CustomerID" else "Products"
    plt.title("Pareto Chart for "+subject)
    ax.set_xlabel(subject)
    sns.despine(left=True,right=True,bottom=True,top=True)
    plt.show()
```

```
customer_price=prepare_pareto_data(df,"CustomerID","TotalPrice")
customer_price.head(15)
```

```
create_pareto_plot(customer_price,"CustomerID","TotalPrice",log=False)
```

```
create_pareto_plot(customer_price,"CustomerID","TotalPrice",log=True)
```

```
item_price=prepare_pareto_data(df,"StockCode","TotalPrice")
item_price.head(15)
```

```
create_pareto_plot(item_price,"StockCode","TotalPrice",log=False)
```

```
create_pareto_plot(item_price,"StockCode","TotalPrice",log=True)
```

```
top_customers=customer_price[customer_price.CumulativePercentage<=80].index
.tolist()
products_for_top_customers=df[df.CustomerID.isin(top_customers)].Descriptio
n.drop_duplicates().values.tolist()
products_for_other_customers=df[~df.CustomerID.isin(top_customers)].Descrip
tion.drop_duplicates().values.tolist()
print(top_customers)
print(products_for_top_customers)
print(products_for_other_customers)
```

**RFM Analysis:**

```
print("Min date: {} \nMax date: {}".format(df.InvoiceDate.min(),df.InvoiceD
ate.max()))
```

```
last_day=df.InvoiceDate.max()+dt.timedelta(days=1)
```

```
rfm_table = df.groupby("CustomerID").agg({"InvoiceDate":lambda x:(last_day-
x.max()).days,"InvoiceNo":"nunique","TotalPrice":"sum"})
rfm_table.rename(columns={"InvoiceDate":"Recency","InvoiceNo":"Frequency","
TotalPrice":"Monetary"},inplace=True)
rfm_table.head(10)
```

```
r_labels=range(5,0,-1)
fm_labels=range(1,6)
rfm_table["R"]=pd.qcut(rfm_table["Recency"],5,labels=r_labels)
rfm_table["F"]=pd.qcut(rfm_table["Frequency"].rank(method='first'),5,labels
=fm_labels)
rfm_table["M"]=pd.qcut(rfm_table["Monetary"],5,labels=fm_labels)
rfm_table.head(10)
```

```
rfm_table["RFM_Segment"]=rfm_table["R"].astype(str)+rfm_table["F"].astype(s
tr)+rfm_table["M"].astype(str)
rfm_table["RFM_Score"]=rfm_table[["R","F","M"]].sum(axis=1)
rfm_table.head(10)
```

```
segt_map={
    r'[1-2][1-2]':'Hibernating',
    r'[1-2][3-4]':'At-Risk',
    r'[1-2]5':'Cannot Lose Them',
    r'3[1-2]':'About To Sleep',
    r'33':'Need Attention',
    r'[3-4][4-5]':'Loyal Customers',
    r'41':'Promising',
    r'51':'New Customers',
    r'[4-5][2-3]':'Potential Loyalists',
    r'5[4-5]':'Champions'
}
rfm_table['Segment']=rfm_table['R'].astype(str)+rfm_table['F'].astype(str)
rfm_table['Segment']=rfm_table['Segment'].replace(segt_map,regex=True)
rfm_table.head(10)
```

```
rfm_coordinates={"Champions":[3,5,0.8,1],
                   "Loyal Customers":[3,5,0.4,0.8],
                   "Cannot Lose Them":[4,5,0,0.4],
                   "At-Risk":[2,4,0,0.4],
                   "Hibernating":[0,2,0,0.4],
                   "About To Sleep":[0,2,0.4,0.6],
                   "Promising":[0,1,0.6,0.8],
                   "New Customers":[0,1,0.8,1],
                   "Potential Loyalists":[1,3,0.6,1],
                   "Need Attention":[2,3,0.4,0.6]
                }
```

```
fig,ax=plt.subplots(figsize=(19,15))
ax.set_xlim([0,5])
ax.set_ylim([0,5])
```

```
plt.rcParams["axes.facecolor"]="white"
palette=["#282828","#04621B","#971194","#F1480F","#4C00FF","#FF007B","#9736FF","#8992F3","#B29800","#80004C"]
for key,color in zip(rfm_coordinates.keys(),palette[:10]):
    coordinates=rfm_coordinates[key]
    ymin,ymax,xmin,xmax=coordinates[0],coordinates[1],coordinates[2],coordinates[3]
    ax.axhspan(ymin=ymin,ymax=ymax,xmin=xmin,xmax=xmax,facecolor=color)
    users=rfm_table[rfm_table.Segment==key].shape[0]
    users_percentage=(rfm_table[rfm_table.Segment==key].shape[0]/rfm_table.shape[0])*100
    avg_monetary=rfm_table[rfm_table.Segment==key]["Monetary"].mean()
    user_txt="\n\nTotal Users: "+str(users)+"("+str(round(users_percentage,2))+"%)"
    monetary_txt="\n\n\n\nAverage Monetary: "+str(round(avg_monetary,2))
    x=5*(xmin+xmax)/2
    y=(ymin+ymax)/2
    plt.text(x=x,y=y,s=key,ha="center",va="center",fontsize=18,color="white",fontweight="bold")
    plt.text(x=x,y=y,s=user_txt,ha="center",va="center",fontsize=14,color="white")
    plt.text(x=x,y=y,s=monetary_txt,ha="center",va="center",fontsize=14,color="white")
    ax.set_xlabel("Recency Score")
    ax.set_ylabel("Frequency Score")
sns.despine(left=True,bottom=True)
plt.show()
```

```
rfm_table2=rfm_table.reset_index()
rfm_monetary_size=rfm_table2.groupby("Segment").agg({"Monetary":"mean","CustomerID":"nunique"})
rfm_monetary_size.rename(columns={"Monetary":"MeanMonetary","CustomerID":"CustomerCount"},inplace=True)
rfm_monetary_size=rfm_monetary_size.sort_values("MeanMonetary",ascending=False)
```

```
plt.rcParams["axes.facecolor"]="White"
fig,ax=plt.subplots(figsize=(16,10),facecolor="White")
sns.barplot(x=rfm_monetary_size.MeanMonetary,y=rfm_monetary_size.index,ax=ax,color="#101820")
ax2=ax.twiny()
sns.lineplot(x=rfm_monetary_size.CustomerCount,y=rfm_monetary_size.index,ax=ax2,marker="o",linewidth=0,color="Yellow",markeredgecolor="Yellow")
ax2.axis("off")
for y,x in list(enumerate(rfm_monetary_size.CustomerCount)):
    ax2.text(x+10,y+0.05,str(x)+" Customer",color="Red",fontweight="bold")
plt.title("RFM Segments Details")
sns.despine(left=True,right=True,bottom=True,top=True)
plt.show()
```

```
rfm=rfm_table2.groupby("Segment").agg({"CustomerID":"nunique","Recency":"me
an","Frequency":"mean","Monetary":"mean"})
rfm.rename(columns={"CustomerID":"Segment Size"},inplace=True)
cm=sns.light_palette("#A2A2A2",as_cmap=True)
rfm.T.style.background_gradient(cmap=cm,axis=1)\
.set_precision(2)\
.highlight_min(axis=1,color="#195190")\
.highlight_max(axis=1,color="#D60000")
```

```
plt.rcParams["axes.facecolor"]="White"
plt.rcParams["axes.grid"]=False
sns.relplot(x="Recency",y="Frequency",size="Monetary",hue="Segment",data=rf
m_table2,palette=palette,height=10,aspect=2,sizes=(50,1000))
plt.show()
```

```
monetary_per_segment=(rfm_table2.groupby("Segment")["Monetary"].sum() /\
rfm_table2.groupby("Segment")["Monetary"].sum().sum()).sort_values(ascendin
g=False)
```

```
fig,ax=plt.subplots(figsize=(10,10),facecolor="White")
wedges,texts=ax.pie(monetary_per_segment.values,wedgeprops=dict(width=0.5),
startangle=-40,colors=palette)
bbox_props=dict(boxstyle="square,pad=0.3",fc="w",ec="k",lw=0.72)
kw=dict(arrowprops=dict(arrowstyle="-
"),bbox=bbox_props,zorder=0,va="center")
for i,p in enumerate(wedges):
    ang=(p.theta2-p.theta1)/2.+p.theta1
    y=np.sin(np.deg2rad(ang))
    x=np.cos(np.deg2rad(ang))
    horizontalalignment={-1:"right",1:"left"}[int(np.sign(x))]
    connectionstyle="angle,angleA=0,angleB={}".format(ang)
    kw["arrowprops"].update({"connectionstyle":connectionstyle})
    ax.annotate(monetary_per_segment.index[i]+" "+str(round(monetary_per_se
gment[i]*100,2))+"%",xy=(x, y),xytext=(1.35*np.sign(x),1.4*y),horizontalali
gnment=horizontalalignment,**kw)
plt.show()
```

```
rfm_clustering=rfm_table2[["Recency","Frequency","Monetary","Segment"]]
for col in ["Recency","Frequency","Monetary"]:
    scaler=StandardScaler()
    rfm_clustering[col]=np.log(rfm_clustering[col])
    rfm_clustering[col]=scaler.fit_transform(rfm_clustering[col].values.res
hape(-1,1))
rfm_melted=pd.melt(rfm_clustering,id_vars="Segment",value_vars=["Recency","
Frequency","Monetary"],var_name="RFM",value_name="Value")
fig,ax=plt.subplots(figsize=(15, 12),facecolor="#A2A2A2")
ax.set_facecolor("White")
sns.lineplot(x="RFM",y="Value",hue="Segment",data=rfm_melted,palette=palett
e)
ax.legend(bbox_to_anchor=(1.05,1),loc=2,borderaxespad=0.)
```

```
ax.set_yticks([])
ax.set_title("Snake Plot for RFM Segments")
plt.show()
```

```
ds={}
features=["Recency","Frequency","Monetary"]
for k in range(1,21):
    kmeans=KMeans(n_clusters=k,random_state=42)
    kmeans.fit(rfm_clustering[features])
    ds[k]=kmeans.inertia_
plt.figure(figsize=(12,8))
plt.title('Distortion Score Elbow')
plt.xlabel('K');
plt.ylabel('Distortion Score')
sns.pointplot(x=list(ds.keys()),y=list(ds.values()))
plt.show()
```

```
kmeans=KMeans(n_clusters=10,random_state=42)
kmeans.fit(rfm_clustering[features])
cluster=kmeans.labels_
fig,axes=plt.subplots(1,3,figsize=(24,8))
for i,feature in list(enumerate(combinations(["Recency","Frequency","Moneta
ry"],2))):
    sns.scatterplot(x=rfm_clustering[feature[0]],y=rfm_clustering[feature[1
]],hue=cluster,palette=palette[:len(set(cluster))],ax=axes[i]).set_title(fe
ature[0]+" - "+feature[1])
    sns.scatterplot(x=kmeans.cluster_centers_[:,0],y=kmeans.cluster_centers
_[:,1],s=250,color='#C0EB00',label='Centroids',marker="X",ax=axes[i],edgeco
lor="black")
plt.suptitle("Segmentation with KMeans - 10 Clusters")
for ax in axes:
    ax.set_facecolor("White")
    ax.grid(False)
plt.show()
```

```
fig,axes=plt.subplots(1,3,figsize=(18,6))
for ax in axes:
    ax.set_facecolor("White")
    ax.set_xlabel("Clusters")
sns.boxplot(x=cluster,y="Recency",data=rfm_clustering,ax=axes[0]).set_title
("Boxplot for Recency")
sns.boxplot(x=cluster,y="Frequency",data=rfm_clustering,ax=axes[1]).set_tit
le("Boxplot for Frequency")
sns.boxplot(x=cluster,y="Monetary",data=rfm_clustering,ax=axes[2]).set_titl
e("Boxplot for Monetary")
plt.show()
```

**Clustering:**

```
X=rfm_clustering[features]
```

```
num_clusters=2
```

```
predictions=kmeans.fit_predict(X)
kmeans=KMeans(n_clusters=num_clusters,max_iter=50)
kmeans.fit(X)
kmeans_score = davies_bouldin_score(X, predictions)
print("The Davies Bouldin Score: {:.5f}".format(kmeans_score))
```

```
db = DBSCAN(eps=0.8, min_samples=7, metric='euclidean')
db.fit(X)
predictions=kmeans.fit_predict(X)
dbscan_score = davies_bouldin_score(X, predictions)
print("The Davis Bouldin Score: {:.5f}".format(dbscan_score))
```

```
agg = AgglomerativeClustering(n_clusters=2)
yhat = agg.fit(X)
yhat_2 = agg.fit_predict(X)
clusters = unique(yhat)
agglo_score = davies_bouldin_score(X, yhat_2)
print("The Davis Bouldin Score: {:.5f}".format(agglo_score))
```

```
birch = Birch(threshold=0.01, n_clusters=2)
birch.fit(X)
yhat = birch.predict(X)
clusters = unique(yhat)
birch_score = davies_bouldin_score(X, yhat)
print("The Davis Bouldin Score: {:.5f}".format(birch_score))
```

```
optics = OPTICS(eps=0.8, min_samples=10)
yhat = optics.fit_predict(X)
clusters = unique(yhat)
optics_score = davies_bouldin_score(X, yhat)
print("The Davis Bouldin Score: {:.5f}".format(optics_score))
```

```
affpro = AffinityPropagation(damping=0.9)
affpro.fit(X)
yhat = affpro.predict(X)
clusters = unique(yhat)
affinpro_score = davies_bouldin_score(X, yhat)
print("The Davis Bouldin Score: {:.5f}".format(affinpro_score))
```

```
spec = SpectralClustering(n_clusters=2)
yhat = spec.fit_predict(X)
clusters = unique(yhat)
spec_score = davies_bouldin_score(X, yhat)
print("The Davis Bouldin Score: {:.5f}".format(spec_score))
```

```
mbkm = MiniBatchKMeans(n_clusters=2)
mbkm.fit(X)
yhat = mbkm.predict(X)
clusters = unique(yhat)
mbkmeans_score = davies_bouldin_score(X, yhat)
```

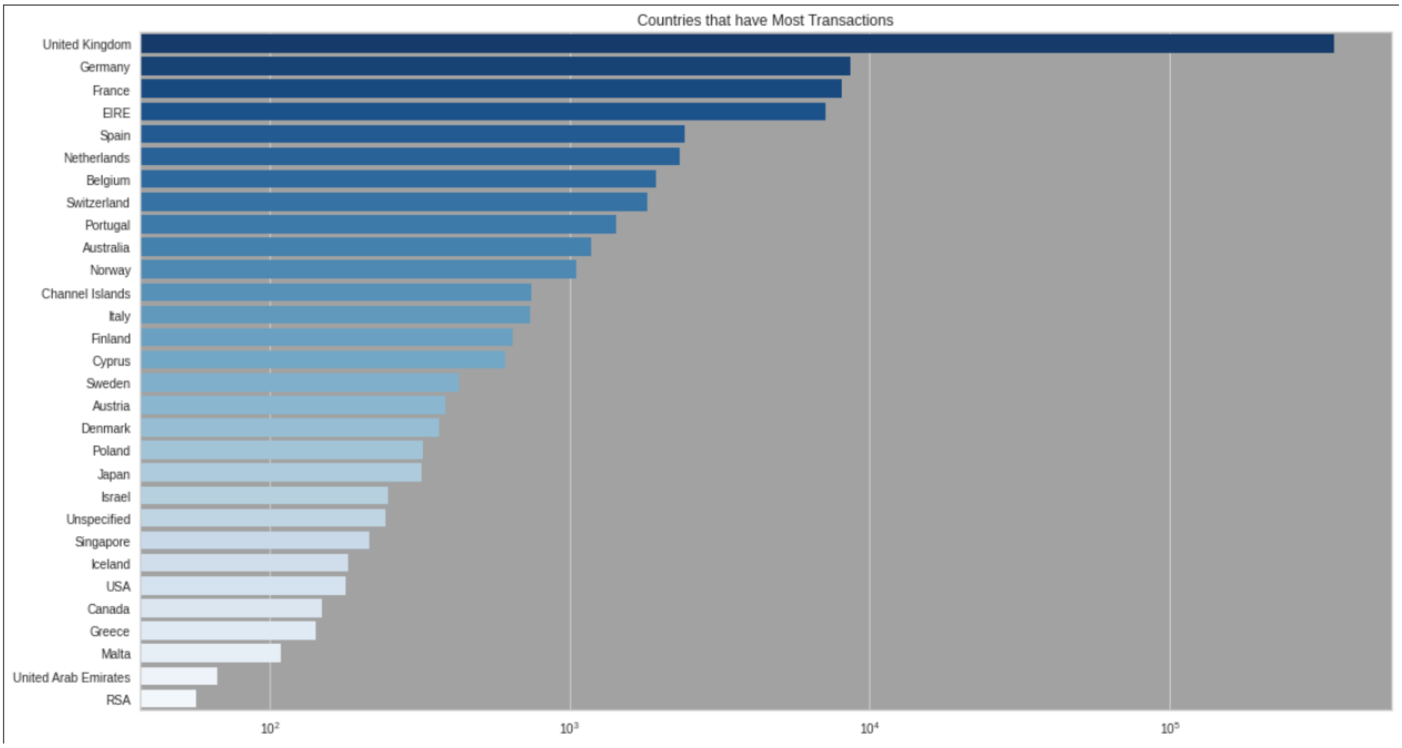
```
print("The Davis Bouldin Score: {:.5f}".format(mbkmeans_score))
```

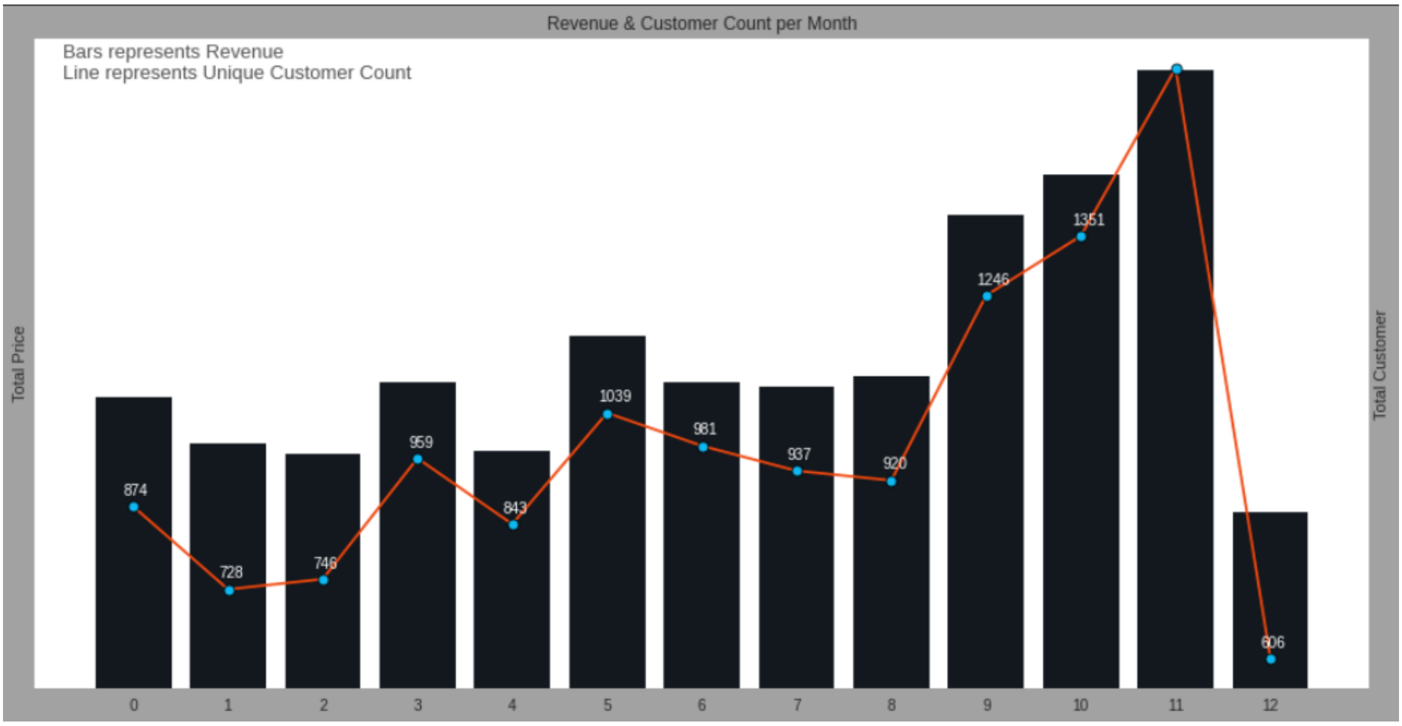
### Model Comparison:

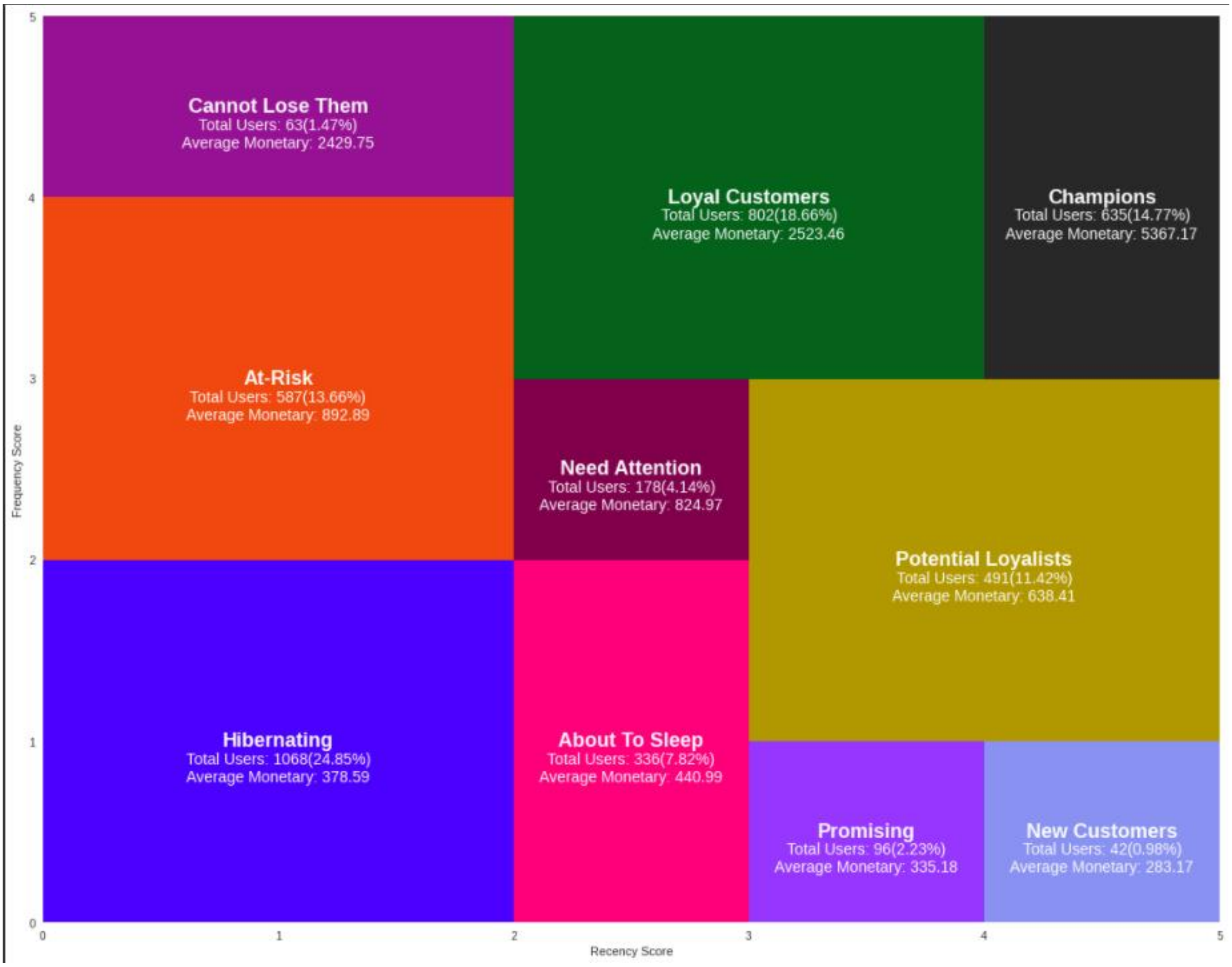
```
models=['Spectral Clustering','Mini-Batch K-Means Clustering','Agglomerative Clustering','DBSCAN Clustering','BIRCH Clustering','Affinity Propagation Clustering','Gaussian Mixture Clustering','K-Means Clustering','OPTICS Clustering']
scores=[spec_score,mbkmeans_score,agglo_score,dbscan_score,birch_score,affinpro_score,gaussmix_score,kmeans_score,optics_score]
score_table=pd.DataFrame({'Model':models,'Score':scores})
print(score_table.sort_values(by='Score',axis=0,ascending=True))
sns.barplot(x=score_table['Score'],y=score_table['Model'],palette='inferno').set_title('Clustering Models')
sns.relplot(x=score_table['Score'],y=score_table['Model'])
```

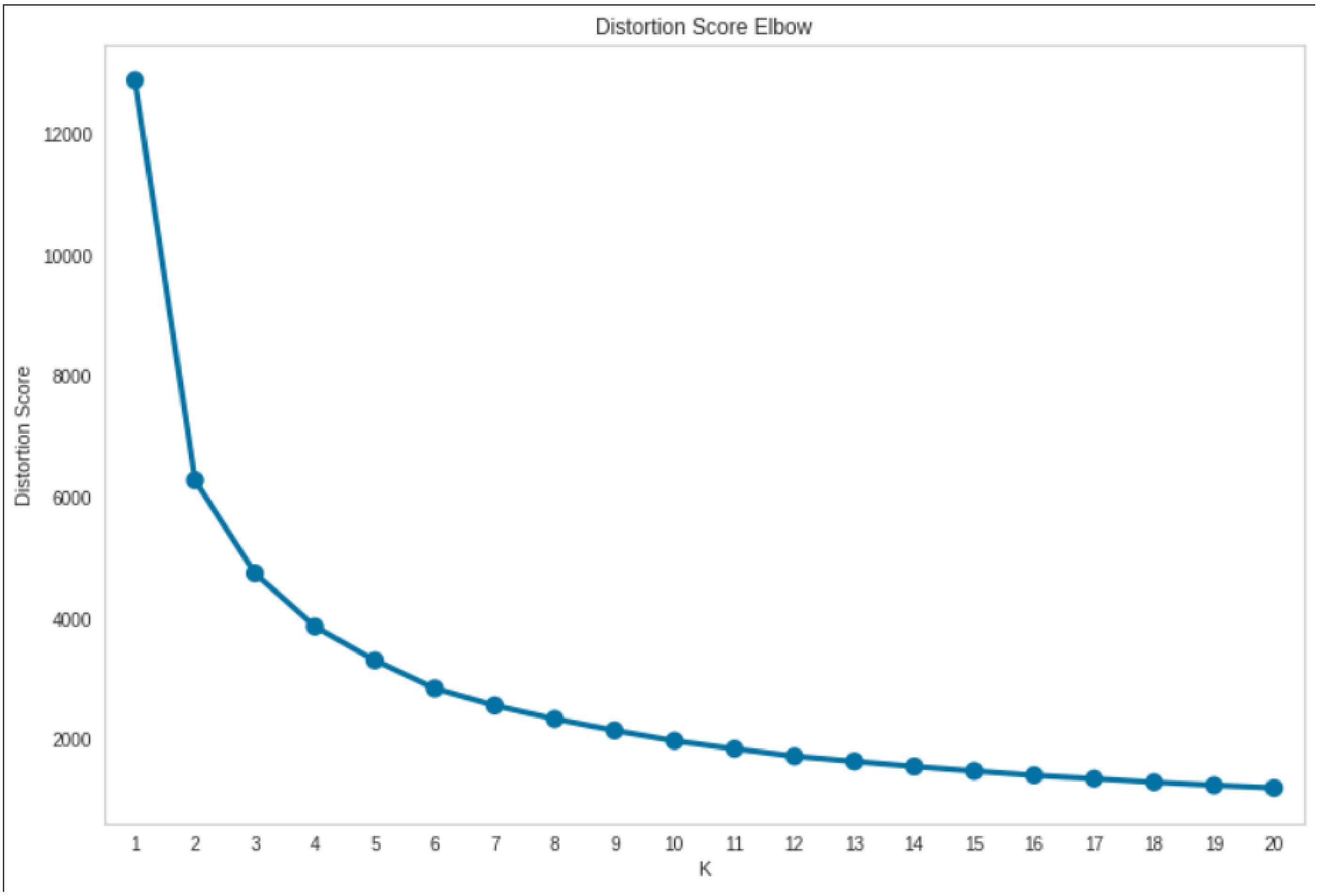
```
models=['Spectral Clustering','Agglomerative Clustering','DBSCAN Clustering','Mini-Batch K-Means Clustering']
scores=[spec_score,agglo_score,dbscan_score,mbkmeans_score]
score_table=pd.DataFrame({'Model':models,'Score':scores})
print(score_table.sort_values(by='Score',axis=0,ascending=True))
sns.barplot(x=score_table['Score'], y=score_table['Model']).set_title('Top 4 Models')
sns.relplot(x=score_table['Score'], y=score_table['Model'])
```

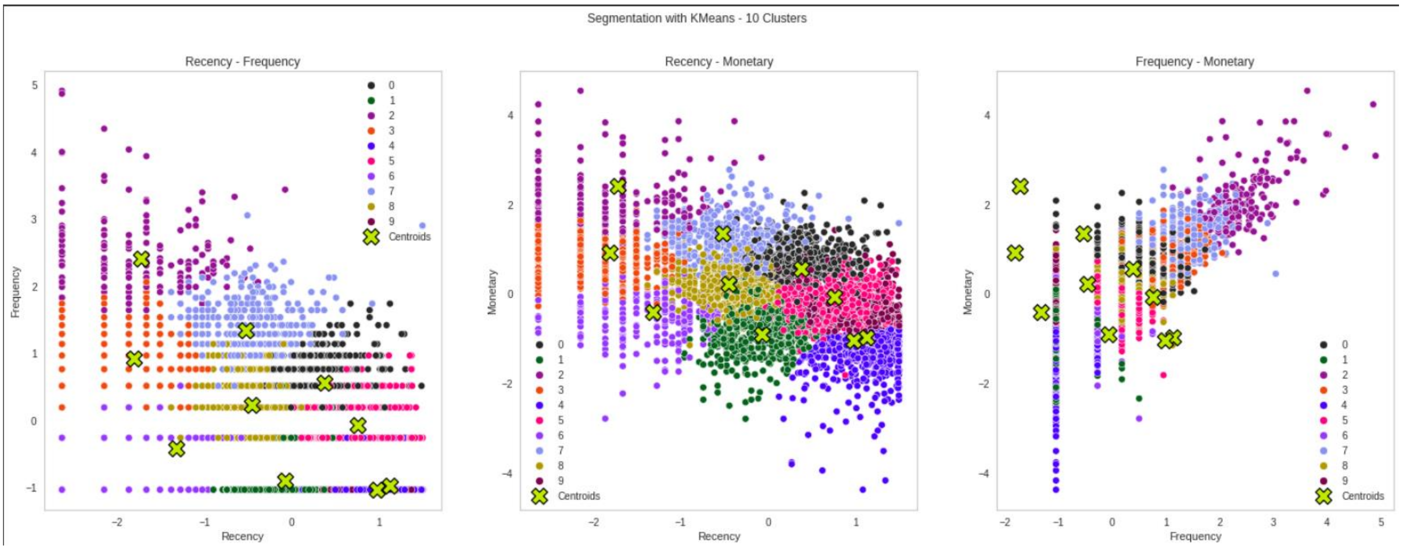
## Outputs:

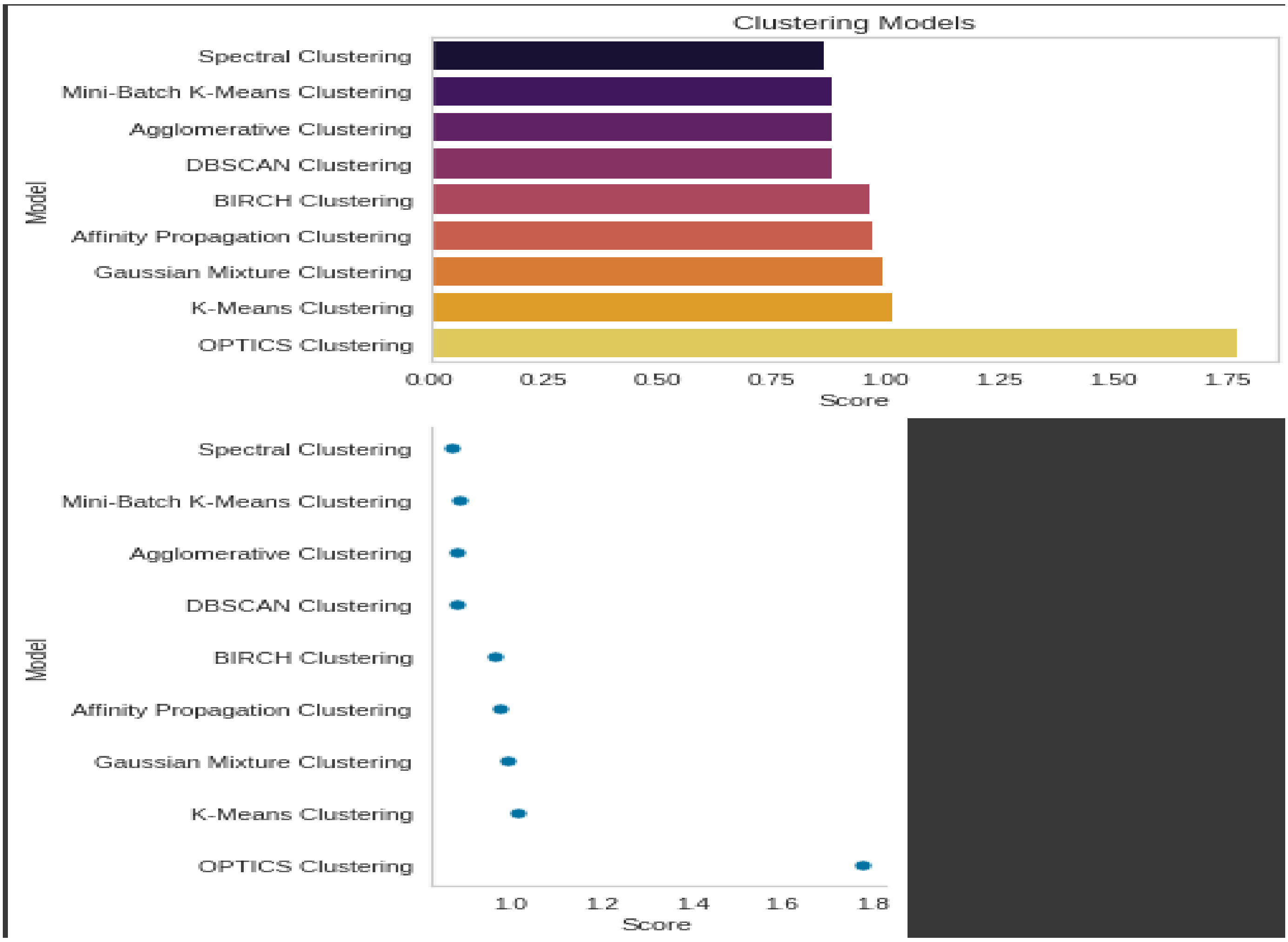



Revenue & Customer Count per Month
Bars represents Revenue
Line represents Unique Customer Count
Total Price
Total Customer
874
728
746
959
843
1039
981
937
920
1246
1351
606
0 1 2 3 4 5 6 7 8 9 10 11 12


Cannot Lose Them
Total Users: 63(1.47%)
Average Monetary: 2429.75
Loyal Customers
Total Users: 802(18.66%)
Average Monetary: 2523.46
Champions
Total Users: 635(14.77%)
Average Monetary: 5367.17
At-Risk
Total Users: 587(13.66%)
Average Monetary: 892.89
Need Attention
Total Users: 178(4.14%)
Average Monetary: 824.97
Potential Loyalists
Total Users: 491(11.42%)
Average Monetary: 638.41
Hibernating
Total Users: 1068(24.85%)
Average Monetary: 378.59
About To Sleep
Total Users: 336(7.82%)
Average Monetary: 440.99
Promising
Total Users: 96(2.23%)
Average Monetary: 335.18
New Customers
Total Users: 42(0.98%)
Average Monetary: 283.17
Frequency Score
Recency Score


Distortion Score Elbow
Distortion Score
K


Segmentation with KMeans - 10 Clusters
Recency - Frequency
Recency - Monetary
Frequency - Monetary
Recency
Frequency
Monetary
Centroids




## Results:

| Sl.No | Model | Davis-Bouldin Score |
|---|---|---|
| 1 | Spectral Clustering | 0.868714 |
| 2 | Mini-Batch K-Means Clustering | 0.884980 |
| 3 | Agglomerative Clustering | 0.880871 |
| 4 | DBSCAN Clustering | 0.881184 |
| 5 | BIRCH Clustering | 0.966185 |
| 6 | Affinity Propagation Clustering | 0.973934 |
| 7 | Gaussian Mixture Clustering | 0.992491 |
| 8 | K-Means Clustering | 1.014866 |
| 9 | OPTICS Clustering | 1.777294 |

## Conclusion:

In Conclusion, I would like to say that the good predicting models top 3 models are,

| Sl.No | Model | Davis-Bouldin Score |
|---|---|---|
| 1 | Spectral Clustering | 0.868714 |
| 2 | Agglomerative Clustering | 0.880871 |
| 3 | Mini-Batch K-Means Clustering | 0.884980 |

These three models can be used for predicting the values. We are using Customer Segmentation for Segmenting the customers based on their spending nature.

***THANK YOU***