# 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 |

# **Write Up**

## **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.

### **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.

541909 rows × 8 columns

8 columns are available.

The item related are - Stock Code (Quantifiable), & Description

Sale Related are - Invoice number and Invoice Date & Quantity & Unit Price

Customer Related are - Customer Id & Country

The main or basic inferred data are - Spending pattern, Spending categories, Customer Spending Behavior

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

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 541909 entries, 0 to 541908

Data columns (total 8 columns):

# Column Non-Null Count Dtype

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

0 InvoiceNo 541909 non-null object

1 StockCode 541909 non-null object

2 Description 540455 non-null object

3 Quantity 541909 non-null int64

4 InvoiceDate 541909 non-null datetime64[ns]

5 UnitPrice 541909 non-null float64

6 CustomerID 406829 non-null float64

7 Country 541909 non-null object

dtypes: datetime64[ns](1), float64(2), int64(1), object(4)

memory usage: 33.1+ MB

Column - Description, & CustomerID, have some Null Values in them

Description Column is No Value add so there is no problem in dropping it. Since Customer Id is the main identifying element, Unique Identifying Entity, it's absence would be difficult to fill through Unlike cost or Sale Unit, where we could use average, we cannot use any other means to treat this except deletion.

Identifying & Removing Null Values. Starting from the column that has max null values. If it clears the null values in other columns, we would not need to repeat the activity with other columns

InvoiceNo 0

StockCode 0

Description 0

Quantity 0

InvoiceDate 0

UnitPrice 0

CustomerID 0

Country 0

dtype: int64

b. Identify & Remove duplicate data records.

Get names of indexes for which Column Unit Price which has value negative value. Assuming that the shop keeper does not pay customer to purchase. There is one such instance, removing this data set as incorrect

c. Perform descriptive analytics on the given data.

However, not doing the same with Quantity. The assumption here is that these might be billed in previous cycle which is not included in this database and were returned to seller in this cycle. Approx ~2% was returned

Table

Description automatically generated

#Counting % of Returns

2.209141343213713

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

Chart, box and whisker chart

Description automatically generated

AS we see from Standard Deviation and also the boxplot that there are lot of outliers in Quantity ordered and in Unit Price Now for Desriptive Analysis.

Quantity UnitPrice CustomerID

Quantity 1.000000 -0.001243 -0.003457

UnitPrice -0.001243 1.000000 -0.004524

CustomerID -0.003457 -0.004524 1.000000

Logo

Description automatically generated

Buyers are from 36 different countries

#Unique Customers are 4372 out of which #Only 79 buyer purchased once

#Percentage of single purchaser is, 1.80

#Repeat Customers are 4293

|  | **Freq** | **CustomerID** |
| --- | --- | --- |
| **17841.0** | 7812 | 17841.0 |
| **14911.0** | 5898 | 14911.0 |
| **14096.0** | 5128 | 14096.0 |
| **12748.0** | 4459 | 12748.0 |
| **14606.0** | 2759 | 14606.0 |
| **...** | ... | ... |
| **15423.0** | 2 | 15423.0 |
| **14642.0** | 2 | 14642.0 |
| **13130.0** | 2 | 13130.0 |
| **13298.0** | 2 | 13298.0 |
| **14821.0** | 2 | 14821.0 |

4293 rows × 2 columns

Change in Corel seen sightly when we plot only the customer who purchased once

Quantity UnitPrice CustomerID

Quantity 1.000000 -0.048550 -0.201286

UnitPrice -0.048550 1.000000 0.082662

CustomerID -0.201286 0.082662 1.000000

Shape, background pattern, square

Description automatically generated

A picture containing shape

Description automatically generated

69.97% of customers ordered more than one item.

Chart

Description automatically generated with medium confidence

Chart, bar chart

Description automatically generated

# Check the oldest and latest date in the dataset.

Oldest date is - 2010-12-01 08:26:00

Latest date is - 2011-12-09 12:50:00

# Count of transactions in different years

A picture containing chart

Description automatically generated

Most of the records belong to 2011. Now doing monthly break up. And we see that max transaction is in Nov & Oct. Could be Black Friday or Halloween. Dec is lesser does not indicating advance buying, because in this data, sales till 09th is considered.

Chart, bar chart

Description automatically generated

#Monthly Sales & monthly\_gross is analysis. It follows the same pattern

Chart, line chart

Description automatically generated

# Boxplot to for Quantity distribution is checked for outliers. Outliers seen in #Unit price distribution

Graphical user interface, application

Description automatically generated

Chart, histogram

Description automatically generated

### **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.

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.

Time cohorts Time cohorts are customers who signed up for a product or service during a particular time frame. Analysing these cohorts shows the customers’ behaviour depending on the time they started using the company’s products or services. The time may be monthly or quarterly, even daily.

#Assigning Cohort to each group

#monthly cohorts based on the month each customer has made their first transaction.

# Group by CustomerID and select the InvoiceMonth value

Calculate time offset in months

Calculating time offset for each transaction allows you to report the metrics for each cohort in a comparable fashion.

First, we will create some variables that capture the integer value of years and months for Invoice and Cohort Date

Min & max in each cohort is

(CohortIndex

1 1.000000

2 0.133956

3 0.130682

4 0.120805

5 0.137725

6 0.115183

7 0.102128

8 0.103943

9 0.083612

10 0.088636

11 0.092105

12 0.149644

13 0.274262

dtype: float64,

CohortIndex

1 1.000000

2 0.381857

3 0.334388

4 0.387131

5 0.359705

6 0.396624

7 0.379747

8 0.354430

9 0.354430

10 0.394515

11 0.373418

12 0.500000

13 0.274262

#retention rate or Active Customers

Table

Description automatically generated

#average price per cohort

Table

Description automatically generated

Avg Quantity by purchased per month

A picture containing chart

Description automatically generated

## **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

### Building RFM

**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.**

RFM Analysis

RFM analysis is a customer segmentation technique that uses past purchase behavior to divide customers into groups. RFM helps divide customers into various categories or clusters to identify customers who are more likely to respond to promotions and also for future personalization services.

**2. Calculate RFM metrics.**

Recency (R): Time since last purchase  
Frequency (F): Total number of purchases  
Monetary (M): Total purchase value

For RFM need to divide customers into four equal groups according to the distribution of values for recency, frequency, and monetary value. Four equal groups across three variables create 64 (4x4x4) different customer segments.

For example:

Customer with most recent purchase (R=4),  
Customer with most quantity (F=4),  
Customer who spent the most (M=4)

This customer belongs to RFM segment 4-4-4 (Best Customers), (R=4, F=4, M=4)

**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.**

#For recency, need to get the date difference since the last purchase.   
#For this using the last purchase date on the database as today's date  
#last date available in our dataset  
# Drop Current to date that we took for calculation. That is no more required  
#Now finding out Frequency - how often or how many a customer used the product of a company.

Doing the same on the Spending or Monetary

Monetary is the total amount of money a customer spent in that given period. Therefore, big spenders will be differentiated with other customers such as MVP or VIP.

#Also creating a separate table for Customers for RFM. First merging frequency with regency and then that table with monetary

#RFM Table integrity Check is done - Passed

Then Assigning Score to the RFM before categorization. Will help later in plotting & Creating the Segments

'Best Customers':'444', # Highest frequency as well as monetary value with least recency  
'Loyal Customers':'344', # High frequency as well as monetary value with good recency  
'Big Spenders':'334', # High monetary value but good recency and frequency values  
'Almost Lost':'244', # Customer's shopping less often now who used to shop a lot  
'Lost Customers':'144', # Customer's shopped long ago who used to shop a lot.  
'Recent Customers':'443', # Customer's who recently started shopping a lot but with less monetary value  
'Lost Cheap Customers':'122', # Customer's shopped long ago but with less frequency and monetary value  
'No Harm to Lose Cheap Customers':'211' # Customer's shopped sometime back ago but with less frequency and monetary value

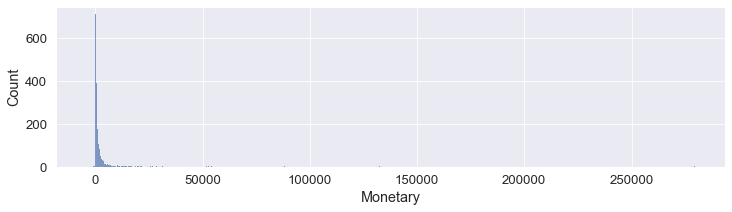
# Swap the key and value of dictionary. So that Lookup is from value to Customer type and not vice-versa

|  |  |  |
| --- | --- | --- |
| Segment | # | Reference |
| Almost Lost | 90 |  |
| Best Customers | 482 | These need Promotion materials and other engagement |
| Big Spenders | 55 | These need exclusive product - high end newsletters |
| Lost Cheap Customers | 151 | They may come back but focus spending is not recommended |
| Lost Customers | 13 | No effort to win them back |
| Loyal Customers | 225 | They need focused product list and AI built-in recommendation |
| No Harm to Lose Cheap Customers | 177 | Won't spend of this category |
| Recent Customers | 99 | Need to keep their interest alive - promote newer things on their spending using product recommendation |

Chart, bar chart

Description automatically generated

Table

Description automatically generated with medium confidence 

Chart, bar chart

Description automatically generated

|  | **CustomerID** | **Recency** | **Frequency** | **Monetary** | **R\_Quartile** | **F\_Quartile** | **M\_Quartile** | **RFM\_Score** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 |
| **mean** | 15299.677722 | 91.581199 | 5.075480 | 1893.531433 | 2.510979 | 2.349039 | 2.500000 | 7.360018 |
| **std** | 1722.390705 | 100.772139 | 9.338754 | 8218.696204 | 1.117084 | 1.151264 | 1.118162 | 2.872703 |
| **min** | 12346.000000 | 0.000000 | 1.000000 | -4287.630000 | 1.000000 | 1.000000 | 1.000000 | 3.000000 |
| **25%** | 13812.750000 | 16.000000 | 1.000000 | 291.795000 | 2.000000 | 1.000000 | 1.750000 | 5.000000 |
| **50%** | 15300.500000 | 50.000000 | 3.000000 | 644.070000 | 3.000000 | 2.000000 | 2.500000 | 7.000000 |
| **75%** | 16778.250000 | 143.000000 | 5.000000 | 1608.335000 | 4.000000 | 3.000000 | 3.250000 | 10.000000 |
| **max** | 18287.000000 | 373.000000 | 248.000000 | 279489.020000 | 4.000000 | 4.000000 | 4.000000 | 12.000000 |

## **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.

K-Means Clustering

From the above plots and rfm\_table, we see that data is highly skewed.

It needs to be transformed and scale the data first because K-Means assumes that the variables should have a symmetric distributions(not skewed) and they should have same average values as well as same variance.Also, noticed, -ve value in monetery.

minimum range of value starts from 1 otherwise log transformation may lead to errors in graph plotting as well as K-Means clustering. After that we will utilize log transformation and scaling to make data available for for K-Means clustering.

The k-means algorithm is an unsupervised clustering algorithm. It takes a bunch of unlabeled points and tries to group them into “k” number of clusters.

It is unsupervised because the points have no external classification.

Step 0: Preparing the data; scaling and removal of -ve values

Step 1: Determine K value by Elbow method and specify the number of clusters K

Step 2: Randomly assign each data point to a cluster

Step 3: Determine the cluster centroid coordinates

Step 4: Determine the distances of each data point to the centroids and re-assign each point to the closest cluster centroid based upon minimum distance

Step 5: Calculate cluster centroids again

Step 6: Repeat steps 4 and 5 until we reach global optima where no improvements are possible and no switching of data points from one cluster to other.

# Create a copy of rfm table for scaled calculation

Since it is unsupervised learning, we do not need to define the the Segment & RFM\_Score. We need the raw 3 components to find the clusters. Later we would add it in main table to see which cluster the cusomer belongs to.

Separating the three main inputs for K-Clustering and scale it

| **Recency** | **Frequency** | **Monetary** |
| --- | --- | --- |
| **0** | 5.786897 | 0.693147 | 8.363723 |
| **1** | 1.098612 | 1.945910 | 9.059358 |
| **2** | 4.330733 | 1.386294 | 8.713725 |
| **3** | 2.944439 | 0.000000 | 8.707182 |
| **4** | 5.739793 | 0.000000 | 8.438806 |

Chart

Description automatically generated

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

Finding out the optimum value of the clusters using elbow method and using the feature in Kmean called inertia\_

Inertia measures how well a dataset was clustered by K-Means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster.

A good model is one with low inertia AND a low number of clusters (K). However, this is a tradeoff because as K increases, inertia decreases.

In Figure below the slowdown occurs at 5 but sharp cut starts at 3. So, we take 5 or 3 as the number of cluster = k = 5

Chart, line chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

|  | **CustomerID** | **Recency** | **Frequency** | **Monetary** | **R\_Quartile** | **F\_Quartile** | **M\_Quartile** | **RFMScore** | **Segment** | **RFM\_Score** | **Cluster** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 12346.0 | 326 | 2 | 4288.63 | 1 | 2 | 1 | 121 | None | 4 | 0 |
| **1** | 12347.0 | 3 | 7 | 8598.63 | 4 | 4 | 4 | 444 | Best Customers | 12 | 2 |
| **2** | 12348.0 | 76 | 4 | 6085.87 | 2 | 3 | 4 | 234 | None | 9 | 1 |
| **3** | 12349.0 | 19 | 1 | 6046.18 | 3 | 1 | 4 | 314 | None | 8 | 3 |
| **4** | 12350.0 | 311 | 1 | 4623.03 | 1 | 1 | 2 | 112 | None | 4 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **4367** | 18280.0 | 278 | 1 | 4469.23 | 1 | 1 | 1 | 111 | None | 3 | 0 |
| **4368** | 18281.0 | 181 | 1 | 4369.45 | 1 | 1 | 1 | 111 | None | 3 | 0 |
| **4369** | 18282.0 | 8 | 3 | 4465.23 | 4 | 2 | 1 | 421 | None | 7 | 4 |
| **4370** | 18283.0 | 4 | 16 | 6334.16 | 4 | 4 | 4 | 444 | Best Customers | 12 | 2 |
| **4371** | 18287.0 | 43 | 3 | 6125.91 | 3 | 2 | 4 | 324 | None | 9 | 1 |

**c. Analyze these clusters and comment on the results.**

Analyze each Cluster

Cluster 0 & 4 does not matter for us. Their RFM Score avg is low, and as we see they do not fall under any specialized marketing plan. They are the ones, who have very low RF&M. Though their number is high. They may be chanced customer who happen to drop in by some add etc.

1 & 3 has few categories that we had defined. Still a lot of effort is not to be directed on this cluster. Normal exposure to brand is good enough.

2 is the category we should be foucsing out attention to. Their RFM avg is 12

Snake plots

Market research technique to compare different segments

Visual representation of each segment's attributes

Plot each cluster's average normalized values of each attribute

To plot this we should have normalized data distribution and all the attributes in a single column. We will use pandas melt facility to achieve that

Chart, line chart

Description automatically generated

Also, Relative importance is found by dividing Cluster avg with Population Avg

0, 1, 2, & 3 show similar spending.

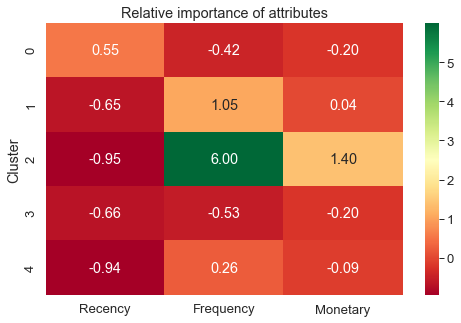
4 & 3 have are very infrequent. 0 shows average frequency of vist, - These are low spending but have a good volume of transaction. 0 cannot be ignores, thought a lot of effort or resources may not be given.

but hightest is 2, followed by 1 - they are good spender with good frequency. We would need to make sure we retain them.

Surprizingly 2 shows low recency. So these are planned buyers, not impulsive ones, whole 0 are the impulsive ones.

Now we get cluster average and population av erage to see the relative importance of each cluster

Then plot it in heat map



## **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

Timeline

Description automatically generated with medium confidence

# **Code & Screen Shots**



import pandas as pd

import numpy as np

import seaborn as sns

from operator import attrgetter

import matplotlib.colors as mcolors

import matplotlib.pyplot as plt

import datetime as dt

from scipy.stats import skewnorm

import scipy.stats as stats

from sklearn.preprocessing import LabelEncoder

import pylab as p

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.model\_selection import learning\_curve

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import classification\_report, confusion\_matrix

df=pd.read\_excel(r'D:\OneDrive\Studies\AI - ML\Capstone Project\OnlineRetail.xlsx',sheet\_name='Online Retail')

df.head()

Graphical user interface

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, email

Description automatically generated with medium confidence

Chart, box and whisker chart

Description automatically generated

corr = df.corr()

print(corr)

sns.heatmap(corr,

xticklabels=corr.columns,

yticklabels=corr.columns)

Logo

Description automatically generated

#Unique Countries

pd.DataFrame(df['Country'].unique())

A picture containing table

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

SBuyer['CustomerID']= SBuyer.index

SBuyer

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

df.Country.value\_counts(normalize=True).head(10).mul(100).round(1).astype(str) + '%'

Graphical user interface, text, application, email

Description automatically generated

# Creating histogram

fig, ax = plt.subplots(figsize =(10, 7))

ax.hist(n\_orders, bins = [0, 25, 50, 75, 100])

# Show plot

plt.show()

ax.set(title='Distribution of number of orders per customer',

xlabel='# of orders',

ylabel='# of customers');

Chart

Description automatically generated with medium confidence

# Check the oldest and latest date in the dataset.

print(f'Oldest date is - {df.InvoiceDate.min()}\n')

print(f'Latest date is - {df.InvoiceDate.max()}')

Oldest date is - 2010-12-01 08:26:00

Latest date is - 2011-12-09 12:50:00

#Monthly Sales

# importing DateTime module to convert extracted dates

def get\_month(x):

return dt.datetime(x.year, x.month, 1)

df['InvoiceMonth'] = df['InvoiceDate'].apply(get\_month)

df.head()

Chart

Description automatically generated

# Count of transactions in different years

df.InvoiceDate.dt.year.value\_counts(sort=False).plot(kind='bar', rot=45);

Chart, histogram

Description automatically generated

Most of the records belong to 2011. Now doing monthly break up. And we see that max transaction is in Nov & Oct. Could be Black Friday or Halloween. Dec is lesser does not indicating advance buying, because in this data, sales till 09th is condidered.

df[df.InvoiceDate.dt.year==2011].InvoiceDate.dt.month.value\_counts(sort=False).plot(kind='bar')

Chart, bar chart

Description automatically generated

df['Total\_cost'] = df['UnitPrice']\*df['Quantity']

df

Graphical user interface, application, table

Description automatically generated

monthly\_gross = df[df.InvoiceDate.dt.year==2011].groupby(df.InvoiceDate.dt.month).Total\_cost.sum()

plt.figure(figsize=(10,5))

sns.lineplot(y=monthly\_gross.values,x=monthly\_gross.index, marker='o');

plt.xticks(range(1,13))

plt.show();

Chart, line chart

Description automatically generated

# Boxplot to for Quantity distribution

sns.boxplot(y='Quantity', data=df, orient='h');

Chart, box and whisker chart

Description automatically generated

#Unit price distribution

sns.set(style="darkgrid")

plt.boxplot(df['UnitPrice'], vert = 0)

#plt.tight\_layout()

plt.show()

#sns.boxplot(y='UnitPrice', data=df)

Chart, histogram

Description automatically generated

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.

Time cohorts Time cohorts are customers who signed up for a product or service during a particular time frame. Analysing these cohorts shows the customers’ behaviour depending on the time they started using the company’s products or services. The time may be monthly or quarterly, even daily.

#Assigning Cohor to each group

group = df.groupby('CustomerID')['InvoiceMonth']

group.head()

0 2011-12-01

1 2011-01-01

4 2010-12-01

10 2011-04-01

20 2011-04-01

...

541900 2011-07-01

541903 2011-05-01

541906 2011-11-01

541907 2011-01-01

541908 2011-12-01

Name: InvoiceMonth, Length: 21206, dtype: datetime64[ns]

df['Month'] = df.groupby('CustomerID')['InvoiceMonth'].transform('min')

df

| **InvoiceNo** | **StockCode** | **Description** | **Quantity** | **InvoiceDate** | **UnitPrice** | **CustomerID** | **Country** | **InvoiceMonth** | **Total\_cost** | **Month** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | C581484 | 23843 | PAPER CRAFT , LITTLE BIRDIE | -80995 | 2011-12-09 09:27:00 | 2.08 | 16446.0 | United Kingdom | 2011-12-01 | -168469.6 | 2011-05-01 |
| **1** | C541433 | 23166 | MEDIUM CERAMIC TOP STORAGE JAR | -74215 | 2011-01-18 10:17:00 | 1.04 | 12346.0 | United Kingdom | 2011-01-01 | -77183.6 | 2011-01-01 |
| **4** | C536757 | 84347 | ROTATING SILVER ANGELS T-LIGHT HLDR | -9360 | 2010-12-02 14:23:00 | 0.03 | 15838.0 | United Kingdom | 2010-12-01 | -280.8 | 2010-12-01 |
| **10** | C550456 | 21108 | FAIRY CAKE FLANNEL ASSORTED COLOUR | -3114 | 2011-04-18 13:08:00 | 2.10 | 15749.0 | United Kingdom | 2011-04-01 | -6539.4 | 2011-01-01 |
| **20** | C550456 | 21175 | GIN + TONIC DIET METAL SIGN | -2000 | 2011-04-18 13:08:00 | 1.85 | 15749.0 | United Kingdom | 2011-04-01 | -3700.0 | 2011-01-01 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **541903** | 554868 | 22197 | SMALL POPCORN HOLDER | 4300 | 2011-05-27 10:52:00 | 0.72 | 13135.0 | United Kingdom | 2011-05-01 | 3096.0 | 2011-05-01 |
| **541904** | 573008 | 84077 | WORLD WAR 2 GLIDERS ASSTD DESIGNS | 4800 | 2011-10-27 12:26:00 | 0.21 | 12901.0 | United Kingdom | 2011-10-01 | 1008.0 | 2011-03-01 |
| **541906** | 578841 | 84826 | ASSTD DESIGN 3D PAPER STICKERS | 12540 | 2011-11-25 15:57:00 | 0.00 | 13256.0 | United Kingdom | 2011-11-01 | 0.0 | 2011-11-01 |
| **541907** | 541431 | 23166 | MEDIUM CERAMIC TOP STORAGE JAR | 74215 | 2011-01-18 10:01:00 | 1.04 | 12346.0 | United Kingdom | 2011-01-01 | 77183.6 | 2011-01-01 |
| **541908** | 581483 | 23843 | PAPER CRAFT , LITTLE BIRDIE | 80995 | 2011-12-09 09:15:00 | 2.08 | 16446.0 | United Kingdom | 2011-12-01 | 168469.6 | 2011-05-01 |

401604 rows × 11 columns

#monthly cohorts based on the month each customer has made their first transaction.

def get\_month(x):

return dt.datetime(x.year,x.month,1)

# Create InvoiceMonth column

df['InvoiceMonth'] = df['InvoiceDate'].apply(get\_month)

# Group by CustomerID and select the InvoiceMonth value

grouping = df.groupby('CustomerID')['InvoiceMonth']

# Assign a minimum InvoiceMonth value to the dataset

df['Month'] = grouping.transform('min')

Calculate time offset in months Calculating time offset for each transaction allows you to report the metrics for each cohort in a comparable fashion.

First, we will create some variables that capture the integer value of years and months for Invoice and Cohort Date

def get\_date\_int(df, column):

year = df[column].dt.year

month = df[column].dt.month

return year, month

# Get the integers for date parts from the `InvoiceMonth` column

invoice\_year, invoice\_month = get\_date\_int(df,'InvoiceMonth')

# Get the integers for date parts from the `CohortMonth` column

cohort\_year, cohort\_month = get\_date\_int(df,'Month')

# Calculate difference in years

years\_diff = invoice\_year - cohort\_year

# Calculate difference in months

months\_diff = invoice\_month - cohort\_month

# Extract the difference in months from all previous values

df['CohortIndex'] = years\_diff \* 12 + months\_diff + 1

#Sanity Check to see if the Cohort Index is of different number

df['CohortIndex']

0 8

1 1

4 1

10 4

20 4

..

541903 1

541904 8

541906 1

541907 1

541908 8

Name: CohortIndex, Length: 401604, dtype: int64

df.head()

|  | **InvoiceNo** | **StockCode** | **Description** | **Quantity** | **InvoiceDate** | **UnitPrice** | **CustomerID** | **Country** | **InvoiceMonth** | **Total\_cost** | **Month** | **CohortIndex** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | C581484 | 23843 | PAPER CRAFT , LITTLE BIRDIE | -80995 | 2011-12-09 09:27:00 | 2.08 | 16446.0 | United Kingdom | 2011-12-01 | -168469.6 | 2011-05-01 | 8 |
| **1** | C541433 | 23166 | MEDIUM CERAMIC TOP STORAGE JAR | -74215 | 2011-01-18 10:17:00 | 1.04 | 12346.0 | United Kingdom | 2011-01-01 | -77183.6 | 2011-01-01 | 1 |
| **4** | C536757 | 84347 | ROTATING SILVER ANGELS T-LIGHT HLDR | -9360 | 2010-12-02 14:23:00 | 0.03 | 15838.0 | United Kingdom | 2010-12-01 | -280.8 | 2010-12-01 | 1 |
| **10** | C550456 | 21108 | FAIRY CAKE FLANNEL ASSORTED COLOUR | -3114 | 2011-04-18 13:08:00 | 2.10 | 15749.0 | United Kingdom | 2011-04-01 | -6539.4 | 2011-01-01 | 4 |
| **20** | C550456 | 21175 | GIN + TONIC DIET METAL SIGN | -2000 | 2011-04-18 13:08:00 | 1.85 | 15749.0 | United Kingdom | 2011-04-01 | -3700.0 | 2011-01-01 | 4 |

#retention rate or Active Customers

grouping = df.groupby(['Month', 'CohortIndex'])

# Count the number of unique values per customer ID

cohort\_data = grouping['CustomerID'].apply(pd.Series.nunique).reset\_index()

# Create a pivot

cohort\_counts = cohort\_data.pivot(index='Month', columns='CohortIndex', values='CustomerID')

cohort\_counts

| **CohortIndex** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Month** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2010-12-01** | 948.0 | 362.0 | 317.0 | 367.0 | 341.0 | 376.0 | 360.0 | 336.0 | 336.0 | 374.0 | 354.0 | 474.0 | 260.0 |
| **2011-01-01** | 421.0 | 101.0 | 119.0 | 102.0 | 138.0 | 126.0 | 110.0 | 108.0 | 131.0 | 146.0 | 155.0 | 63.0 | NaN |
| **2011-02-01** | 380.0 | 94.0 | 73.0 | 106.0 | 102.0 | 94.0 | 97.0 | 107.0 | 98.0 | 119.0 | 35.0 | NaN | NaN |
| **2011-03-01** | 440.0 | 84.0 | 112.0 | 96.0 | 102.0 | 78.0 | 116.0 | 105.0 | 127.0 | 39.0 | NaN | NaN | NaN |
| **2011-04-01** | 299.0 | 68.0 | 66.0 | 63.0 | 62.0 | 71.0 | 69.0 | 78.0 | 25.0 | NaN | NaN | NaN | NaN |
| **2011-05-01** | 279.0 | 66.0 | 48.0 | 48.0 | 60.0 | 68.0 | 74.0 | 29.0 | NaN | NaN | NaN | NaN | NaN |
| **2011-06-01** | 235.0 | 49.0 | 44.0 | 64.0 | 58.0 | 79.0 | 24.0 | NaN | NaN | NaN | NaN | NaN | NaN |
| **2011-07-01** | 191.0 | 40.0 | 39.0 | 44.0 | 52.0 | 22.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2011-08-01** | 167.0 | 42.0 | 42.0 | 42.0 | 23.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2011-09-01** | 298.0 | 89.0 | 97.0 | 36.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2011-10-01** | 352.0 | 93.0 | 46.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2011-11-01** | 321.0 | 43.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2011-12-01** | 41.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

Graphical user interface, text, application

Description automatically generated

retention.min(), retention.max()

(CohortIndex

1 100.000000

2 13.395639

3 13.068182

4 12.080537

5 13.772455

6 11.518325

7 10.212766

8 10.394265

9 8.361204

10 8.863636

11 9.210526

12 14.964371

13 27.426160

dtype: float64,

CohortIndex

1 100.000000

2 38.185654

3 33.438819

4 38.713080

5 35.970464

6 39.662447

7 37.974684

8 35.443038

9 35.443038

10 39.451477

11 37.341772

12 50.000000

13 27.426160

dtype: float64)

month\_list = ["Dec '10", "Jan '11", "Feb '11", "Mar '11", "Apr '11",\

"May '11", "Jun '11", "Jul '11", "Aug '11", "Sep '11", \

"Oct '11", "Nov '11", "Dec '11"]

retention = retention/100

# Initialize inches plot figure

plt.figure(figsize=(15,7))

# Add a title

plt.title('Retention by Monthly Cohorts')

# Create the heatmap

sns.heatmap(data=retention,

annot = True,

#fmt= '.0%',

cmap = "GnBu",

vmin = 0.0,

vmax = list(retention.max().sort\_values(ascending = False))[1]+3,

fmt = '.1%',

linewidth = 0.3,

yticklabels=month\_list)

plt.show();

Table

Description automatically generated

#average price per cohort

# Create a groupby object and pass the monthly cohort and cohort index as a list

grouping = df.groupby(['Month', 'CohortIndex'])

# Calculate the average of the unit price column

cohort\_data = grouping['UnitPrice'].mean()

# Reset the index of cohort\_data

cohort\_data = cohort\_data.reset\_index()

# Create a pivot

average\_price = cohort\_data.pivot(index='Month', columns='CohortIndex', values='UnitPrice')

average\_price.round(1)

average\_price.index = average\_price.index.date

average\_price

| **CohortIndex** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2010-12-01** | 3.216682 | 3.182040 | 3.207467 | 3.603758 | 2.937803 | 4.996508 | 3.184572 | 3.235695 | 3.511560 | 3.035982 | 3.309705 | 2.835557 | 2.759449 |
| **2011-01-01** | 3.505492 | 3.653572 | 3.069534 | 8.439024 | 3.157803 | 3.172919 | 2.918498 | 2.749649 | 2.641686 | 5.489040 | 2.886220 | 2.635897 | NaN |
| **2011-02-01** | 3.355968 | 4.469638 | 4.824106 | 3.150045 | 2.987616 | 2.792577 | 2.812985 | 3.214380 | 2.894988 | 2.946092 | 3.217742 | NaN | NaN |
| **2011-03-01** | 3.302802 | 4.990095 | 3.655094 | 3.289768 | 3.616562 | 2.758381 | 2.843273 | 2.809136 | 2.707846 | 2.466172 | NaN | NaN | NaN |
| **2011-04-01** | 3.431172 | 3.958074 | 3.300128 | 2.673439 | 3.028297 | 2.867185 | 2.902668 | 2.812492 | 2.636564 | NaN | NaN | NaN | NaN |
| **2011-05-01** | 4.662054 | 3.243691 | 2.652761 | 3.167391 | 2.667158 | 2.495751 | 2.615408 | 2.560400 | NaN | NaN | NaN | NaN | NaN |
| **2011-06-01** | 10.490030 | 3.205283 | 3.343994 | 2.835952 | 2.553037 | 3.550657 | 2.293928 | NaN | NaN | NaN | NaN | NaN | NaN |
| **2011-07-01** | 4.493676 | 3.480495 | 2.752121 | 2.701985 | 2.403989 | 2.366635 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2011-08-01** | 3.028246 | 5.425904 | 5.714033 | 7.046410 | 6.830066 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2011-09-01** | 3.235116 | 3.584834 | 2.957893 | 2.625593 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2011-10-01** | 4.053162 | 2.678140 | 2.596869 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2011-11-01** | 2.641554 | 2.335018 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2011-12-01** | 2.288479 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

plt.figure(figsize=(15, 7))

# Add a title

plt.title('Average Spend by Monthly Cohorts')

# Create the heatmap

sns.heatmap(data = average\_price,

annot=True,

vmin = 0.0,

# vmax =20,

cmap='Greens',

vmax = list(average\_price.max().sort\_values(ascending = False))[1]+3,

fmt = '.1f',

linewidth = 0.3,

yticklabels=month\_list)

plt.show();

Table

Description automatically generated with medium confidence

#average quantity per cohort

# Create a groupby object and pass the monthly cohort and cohort index as a list

grouping = df.groupby(['Month', 'CohortIndex'])

# Calculate the average of the Quantity column

cohort\_data = grouping['Quantity'].mean()

# Reset the index of cohort\_data

cohort\_data = cohort\_data.reset\_index()

# Create a pivot

average\_quantity = cohort\_data.pivot(index='Month', columns='CohortIndex', values='Quantity')

average\_quantity

Graphical user interface, application

Description automatically generated

Table

Description automatically generated

#Creating a copy of df as safe copy. Will be using df1 for changes

df1 = df

df1

#For recency, need to get the date difference since the last purchase.

#For this using the last purchase date on the database as today's date

Graphical user interface, application, table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

#RFM Segments

# Arguments (x = value, p = recency, monetary\_value, frequency, d = quantiles dict)

def RScore(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

# Arguments (x = value, p = recency, monetary\_value, frequency, k = quantiles dict)

def FMScore(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

#rfm\_table['segment'] = rfm\_table.copy()

rfm\_table['R\_Quartile'] = rfm\_table['Recency'].apply(RScore, args=('Recency',quantiles,))

rfm\_table['F\_Quartile'] = rfm\_table['Frequency'].apply(FMScore, args=('Frequency',quantiles,))

rfm\_table['M\_Quartile'] = rfm\_table['Monetary'].apply(FMScore, args=('Monetary',quantiles,))

rfm\_table

|  | **CustomerID** | **Recency** | **Frequency** | **Monetary** | **R\_Quartile** | **F\_Quartile** | **M\_Quartile** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 12346.0 | 325 | 2 | 0.00 | 1 | 2 | 1 |
| **1** | 12347.0 | 2 | 7 | 4310.00 | 4 | 4 | 4 |
| **2** | 12348.0 | 75 | 4 | 1797.24 | 2 | 3 | 4 |
| **3** | 12349.0 | 18 | 1 | 1757.55 | 3 | 1 | 4 |
| **4** | 12350.0 | 310 | 1 | 334.40 | 1 | 1 | 2 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **4367** | 18280.0 | 277 | 1 | 180.60 | 1 | 1 | 1 |
| **4368** | 18281.0 | 180 | 1 | 80.82 | 1 | 1 | 1 |
| **4369** | 18282.0 | 7 | 3 | 176.60 | 4 | 2 | 1 |
| **4370** | 18283.0 | 3 | 16 | 2045.53 | 4 | 4 | 4 |
| **4371** | 18287.0 | 42 | 3 | 1837.28 | 3 | 2 | 4 |

4372 rows × 7 columns

rfm\_table['RFMScore'] = rfm\_table.R\_Quartile.map(str) \

+ rfm\_table.F\_Quartile.map(str) \

+ rfm\_table.M\_Quartile.map(str)

rfm\_table

| **CustomerID** | **Recency** | **Frequency** | **Monetary** | **R\_Quartile** | **F\_Quartile** | **M\_Quartile** | **RFMScore** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 12346.0 | 325 | 2 | 0.00 | 1 | 2 | 1 | 121 |
| **1** | 12347.0 | 2 | 7 | 4310.00 | 4 | 4 | 4 | 444 |
| **2** | 12348.0 | 75 | 4 | 1797.24 | 2 | 3 | 4 | 234 |
| **3** | 12349.0 | 18 | 1 | 1757.55 | 3 | 1 | 4 | 314 |
| **4** | 12350.0 | 310 | 1 | 334.40 | 1 | 1 | 2 | 112 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **4367** | 18280.0 | 277 | 1 | 180.60 | 1 | 1 | 1 | 111 |
| **4368** | 18281.0 | 180 | 1 | 80.82 | 1 | 1 | 1 | 111 |
| **4369** | 18282.0 | 7 | 3 | 176.60 | 4 | 2 | 1 | 421 |
| **4370** | 18283.0 | 3 | 16 | 2045.53 | 4 | 4 | 4 | 444 |
| **4371** | 18287.0 | 42 | 3 | 1837.28 | 3 | 2 | 4 | 324 |

4372 rows × 8 columns

Text

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated

Timeline

Description automatically generated

# Plot distribution of Frequency

plt.figure(figsize=(12,10))

plt.subplot(3, 1, 2); sns.histplot(rfm\_table['Frequency'])

Table

Description automatically generated

# Checking the distribution of variables.

plt.figure(figsize=(12,10))

# Plot distribution of Monetary

plt.subplot(3, 1, 3); sns.histplot(rfm\_table['Monetary'])

A picture containing table

Description automatically generated

# Checking the distribution of variables.

plt.figure(figsize=(12,10))

# Plot distribution of RFM\_Score Segment

plt.subplot(3, 1, 3); sns.histplot(rfm\_table['RFM\_Score'])

Chart, bar chart

Description automatically generated

rfm\_table.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 4372 entries, 0 to 4371

Data columns (total 10 columns):

# Column Non-Null Count Dtype

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

0 CustomerID 4372 non-null float64

1 Recency 4372 non-null int64

2 Frequency 4372 non-null int64

3 Monetary 4372 non-null float64

4 R\_Quartile 4372 non-null int64

5 F\_Quartile 4372 non-null int64

6 M\_Quartile 4372 non-null int64

7 RFMScore 4372 non-null object

8 RFM\_Score 4372 non-null int64

9 Segment 1292 non-null object

dtypes: float64(2), int64(6), object(2)

memory usage: 375.7+ KB

rfm\_table.describe()

|  | **CustomerID** | **Recency** | **Frequency** | **Monetary** | **R\_Quartile** | **F\_Quartile** | **M\_Quartile** | **RFM\_Score** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 |
| **mean** | 15299.677722 | 91.581199 | 5.075480 | 1893.531433 | 2.510979 | 2.349039 | 2.500000 | 7.360018 |
| **std** | 1722.390705 | 100.772139 | 9.338754 | 8218.696204 | 1.117084 | 1.151264 | 1.118162 | 2.872703 |
| **min** | 12346.000000 | 0.000000 | 1.000000 | -4287.630000 | 1.000000 | 1.000000 | 1.000000 | 3.000000 |
| **25%** | 13812.750000 | 16.000000 | 1.000000 | 291.795000 | 2.000000 | 1.000000 | 1.750000 | 5.000000 |
| **50%** | 15300.500000 | 50.000000 | 3.000000 | 644.070000 | 3.000000 | 2.000000 | 2.500000 | 7.000000 |
| **75%** | 16778.250000 | 143.000000 | 5.000000 | 1608.335000 | 4.000000 | 3.000000 | 3.250000 | 10.000000 |
| **max** | 18287.000000 | 373.000000 | 248.000000 | 279489.020000 | 4.000000 | 4.000000 | 4.000000 | 12.000000 |

# Create a copy of rfm table for scaled calculation

rfm\_s = rfm\_table.copy()

# Shift all values in the column by adding absolute of minimum value to each value, thereby making each value positive.

rfm\_s.Monetary = rfm\_s.Monetary + abs(rfm\_s.Monetary.min()) + 1

rfm\_s.Recency = rfm\_s.Recency + abs(rfm\_s.Recency.min()) + 1

# Check the summary of new values

rfm\_s.describe()

|  | **CustomerID** | **Recency** | **Frequency** | **Monetary** | **R\_Quartile** | **F\_Quartile** | **M\_Quartile** | **RFM\_Score** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 | 4372.000000 |
| **mean** | 15299.677722 | 92.581199 | 5.075480 | 6182.161433 | 2.510979 | 2.349039 | 2.500000 | 7.360018 |
| **std** | 1722.390705 | 100.772139 | 9.338754 | 8218.696204 | 1.117084 | 1.151264 | 1.118162 | 2.872703 |
| **min** | 12346.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 3.000000 |
| **25%** | 13812.750000 | 17.000000 | 1.000000 | 4580.425000 | 2.000000 | 1.000000 | 1.750000 | 5.000000 |
| **50%** | 15300.500000 | 51.000000 | 3.000000 | 4932.700000 | 3.000000 | 2.000000 | 2.500000 | 7.000000 |
| **75%** | 16778.250000 | 144.000000 | 5.000000 | 5896.965000 | 4.000000 | 3.000000 | 3.250000 | 10.000000 |
| **max** | 18287.000000 | 374.000000 | 248.000000 | 283777.650000 | 4.000000 | 4.000000 | 4.000000 | 12.000000 |

rfm\_s.head()

|  | **CustomerID** | **Recency** | **Frequency** | **Monetary** | **R\_Quartile** | **F\_Quartile** | **M\_Quartile** | **RFMScore** | **RFM\_Score** | **Segment** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 12346.0 | 326 | 2 | 4288.63 | 1 | 2 | 1 | 121 | 4 | None |
| **1** | 12347.0 | 3 | 7 | 8598.63 | 4 | 4 | 4 | 444 | 12 | Best Customers |
| **2** | 12348.0 | 76 | 4 | 6085.87 | 2 | 3 | 4 | 234 | 9 | None |
| **3** | 12349.0 | 19 | 1 | 6046.18 | 3 | 1 | 4 | 314 | 8 | None |
| **4** | 12350.0 | 311 | 1 | 4623.03 | 1 | 1 | 2 | 112 | 4 | None |

raw\_data = rfm\_s[['Recency','Frequency','Monetary']]

data\_log = np.log(raw\_data)

# Initialize a standard scaler and fit it

scaler = StandardScaler()

scaler.fit(data\_log)

# Scale and center the data

data\_normalized = scaler.transform(data\_log)

# Create a pandas DataFrame

data\_norm = pd.DataFrame(data=data\_log, index=raw\_data.index, columns=raw\_data.columns)

data\_norm.head()

Chart, bar chart

Description automatically generated

Chart, timeline

Description automatically generated with medium confidence

sse = {}

# Fit KMeans and calculate SSE for each k

for k in range(1, 21):

# Initialize KMeans with k clusters

kmeans = KMeans(n\_clusters=k, random\_state=1)

# Fit KMeans on the normalized dataset

kmeans.fit(data\_norm)

# Assign sum of squared distances to k element of dictionary

sse[k] = kmeans.inertia\_

plt.figure(figsize=(12,8))

plt.title('The Elbow Method')

plt.xlabel('k');

plt.ylabel('Sum of squared errors')

sns.pointplot(x=list(sse.keys()), y=list(sse.values()))

plt.show()

Chart, line chart

Description automatically generated

#Confirming as the same

from sklearn.metrics import silhouette\_score

wcss\_silhouette = []

for i in range(2,12):

km = KMeans(n\_clusters=i, random\_state=0,init='k-means++').fit(data\_norm)

preds = km.predict(data\_norm)

silhouette = silhouette\_score(data\_norm,preds)

wcss\_silhouette.append(silhouette)

print("Silhouette score for number of cluster(s) {}: {}".format(i,silhouette))

5-

plt.figure(figsize=(10,5))

plt.title("The silhouette coefficient method \nfor determining number of clusters\n",fontsize=16)

plt.scatter(x=[i for i in range(2,12)],y=wcss\_silhouette,s=150,edgecolor='k')

plt.grid(True)

plt.xlabel("Number of clusters",fontsize=14)

plt.ylabel("Silhouette score",fontsize=15)

plt.xticks([i for i in range(2,12)],fontsize=14)

plt.yticks(fontsize=15)

plt.show()

Silhouette score for number of cluster(s) 2: 0.46320292333881263

Silhouette score for number of cluster(s) 3: 0.37882385562792953

Silhouette score for number of cluster(s) 4: 0.38942882326747985

Silhouette score for number of cluster(s) 5: 0.38480602348390036

Silhouette score for number of cluster(s) 6: 0.3814620438903689

Silhouette score for number of cluster(s) 7: 0.37740088768033014

Silhouette score for number of cluster(s) 8: 0.3749214782348163

Silhouette score for number of cluster(s) 9: 0.379746242901208

Silhouette score for number of cluster(s) 10: 0.37572690058058267

Silhouette score for number of cluster(s) 11: 0.3869596058631919

Chart, scatter chart

Description automatically generated

#Implementation of K-Means Clustering

plt.figure(figsize=(15,5))

model = KMeans(n\_clusters = 5)

model.fit(data\_norm)

#Extract cluster labels from labels\_ attribute

cluster\_labels = model.labels\_

centers = np.array(model.cluster\_centers\_)

plt.plot()

colormap = np.array(['Red', 'Blue', 'Green', 'Orange', 'Brown', 'Black'])

z = plt.scatter(data\_norm.Recency, data\_norm.Frequency, data\_norm.Monetary, c = colormap[cluster\_labels])

plt.scatter(centers[:,0], centers[:,1], marker="x", color='Black')

Graphical user interface, application

Description automatically generated

Application, table

Description automatically generated

Graphical user interface, table

Description automatically generated

Graphical user interface, application, table

Description automatically generated

Table

Description automatically generated with medium confidence

#Melt the data into along format so RFM values and metric names are stored in 1 column each

data\_melt = pd.melt(data\_norm\_k5.reset\_index(),

id\_vars=['CustomerID', 'Cluster'],

value\_vars=['Recency', 'Frequency', 'Monetary'],

var\_name='Attribute',

value\_name='Value')

data\_melt

|  | **CustomerID** | **Cluster** | **Attribute** | **Value** |
| --- | --- | --- | --- | --- |
| **0** | 12346 | 0 | Recency | 5.786897 |
| **1** | 12347 | 4 | Recency | 1.098612 |
| **2** | 12348 | 3 | Recency | 4.330733 |
| **3** | 12349 | 2 | Recency | 2.944439 |
| **4** | 12350 | 0 | Recency | 5.739793 |
| **...** | ... | ... | ... | ... |
| **13111** | 18280 | 0 | Monetary | 8.404971 |
| **13112** | 18281 | 0 | Monetary | 8.382392 |
| **13113** | 18282 | 1 | Monetary | 8.404076 |
| **13114** | 18283 | 4 | Monetary | 8.753712 |
| **13115** | 18287 | 3 | Monetary | 8.720283 |

13116 rows × 4 columns

plt.figure(figsize=(15,5))

sns.lineplot(x="Attribute", y="Value", hue='Cluster', palette = 'hls', data=data\_melt)

Chart, line chart

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

Chart, funnel chart

Description automatically generated

Graphical user interface, application

Description automatically generated

Table

Description automatically generated with medium confidence

Graphical user interface, text, application, email

Description automatically generated

A screenshot of a computer

Description automatically generated

Table

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

# **Links**

Python Code: <https://github.com/Naseha/Python/blob/main/Capstone%20Project%203%20-%20Retail%20.ipynb>

Document: <https://github.com/Naseha/Python/blob/main/Capstone%20Project%203%20-%20Retail%20.pdf>

Tableau Link: <https://public.tableau.com/app/profile/naseha/viz/CapstoneProject3v1_0-SimpiliLearnOnlineRetail/Country-wiseMonthwiseDetailedDashboard>