Data Science Capstone project - Retail

# **Project Task:** Week 1 Data Cleaning:

1. Perform a preliminary data inspection and data cleaning.
2. Check for missing data and formulate an apt strategy to treat them.
3. Remove duplicate data records.
4. Perform descriptive analytics on the given data.

import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

# **Importing data**

df=pd.read\_excel('/Online Retail.xlsx')

df.head()

|  |  |  |  |
| --- | --- | --- | --- |
| **InvoiceNo** | **StockCode** | **Description** | **Quantity InvoiceDate UnitPrice CustomerID** |
| **0** 536365 | 85123A | WHITE HANGING HEART T- | 6 2010-12-01 2.55 17850.0  08:26:00 |
|  |  | LIGHT HOLDER |  |
| **1** 536365 | 71053 | WHITE METAL | 6 2010-12-01 3.39 17850.0  08:26:00 |
|  |  | LANTERN |  |

# **Data cleaning**

df.isnull().sum()

|  |  |
| --- | --- |
| InvoiceNo | 0 |
| StockCode | 0 |
| Description | 1454 |
| Quantity | 0 |
| InvoiceDate | 0 |
| UnitPrice | 0 |
| CustomerID | 135080 |
| Country | 0 |
| dtype: int64 |  |

df=df.dropna()

df.shape

(401604, 8)

# Checking & Removing duplicate values

df.duplicated().sum() 0

df.drop\_duplicates(keep='first', inplace=True)

df.shape

(401604, 8)

|  |  |  |  |
| --- | --- | --- | --- |
| df.describe() | **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 |

df.corr()

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Quantity** | **UnitPrice** | **CustomerID** |
| **Quantity** | 1.000000 | -0.001243 | -0.003457 |
| **UnitPrice** | -0.001243 | 1.000000 | -0.004524 |
| **CustomerID** | -0.003457 | -0.004524 | 1.000000 |

#let's do a copy of our df for next manipulations retail = df.copy()

# **Exploration of the data**

#calculate revenue per row and add new column

retail['Revenue'] = retail['Quantity'] \* retail['UnitPrice']

retail.InvoiceDate = pd.to\_datetime(retail['InvoiceDate'], format='%d-%m-%Y %H:%M')

# Let's visualize the top grossing months

retail\_month = retail[retail.InvoiceDate.dt.year==2011]

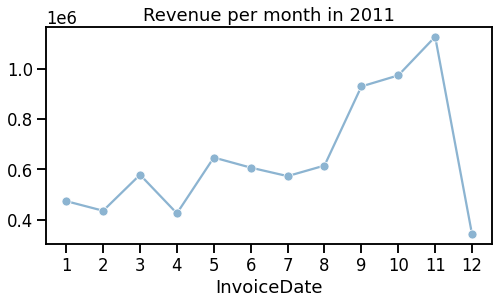
monthly\_gross = retail\_month.groupby(retail\_month.InvoiceDate.dt.month).Revenue.sum()

plt.figure(figsize=(8,4)) sns.set\_context("talk")

sns.set\_palette("PuBuGn\_d")

sns.lineplot(y=monthly\_gross.values,x=monthly\_gross.index, marker='o') plt.xticks(range(1,13))

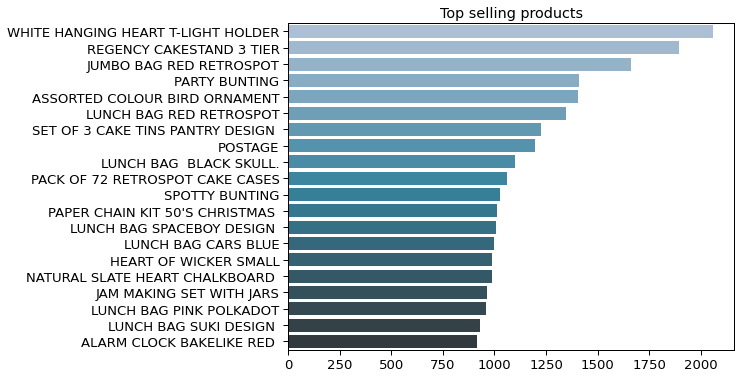
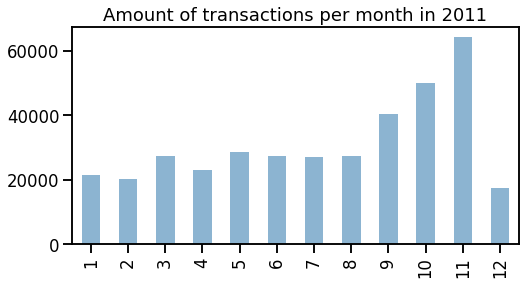
plt.title("Revenue per month in 2011") plt.show()



#amount of transactions per month plt.figure(figsize=(8,4))

retail[retail.InvoiceDate.dt.year==2011].InvoiceDate.dt.month.value\_counts(sort=False).plo plt.title("Amount of transactions per month in 2011")

plt.show()



# Let's visualize some top products from the whole range top\_products = retail['Description'].value\_counts()[:20] plt.figure(figsize=(8,6))

sns.set\_context("paper", font\_scale=1.5) sns.barplot(y = top\_products.index,

x = top\_products.values, palette='PuBuGn\_d')

plt.title("Top selling products") plt.show()

plt.savefig('top\_products.png')

<Figure size 432x288 with 0 Axes>

# **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.

* 1. Create month cohorts and analyze active customers for each cohort.
  2. Analyze the retention rate of customers.
     1. **Cohort Analysis**

import datetime as dt

#creating invoice month column to see first month when customer purchased

retail['InvoiceMonth'] = retail['InvoiceDate'].apply(lambda x: dt.datetime(x.year, x.month

retail.InvoiceMonth

|  |  |
| --- | --- |
| 0 | 2010-12-01 |
| 1 | 2010-12-01 |
| 2 | 2010-12-01 |
| 3 | 2010-12-01 |
| 4 | 2010-12-01 |
|  | ... |
| 541904 | 2011-12-01 |
| 541905 | 2011-12-01 |
| 541906 | 2011-12-01 |
| 541907 | 2011-12-01 |
| 541908 | 2011-12-01 |

Name: InvoiceMonth, Length: 401604, dtype: datetime64[ns]

grouping = retail.groupby('CustomerID')['InvoiceMonth'] #assign smallest invoice value to each customer

retail['CohortMonth'] = grouping.transform('min') retail.head()

|  |  |  |  |
| --- | --- | --- | --- |
| **InvoiceNo** | **StockCode** | **Description** | **Quantity InvoiceDate UnitPrice CustomerID** |
| **0** 536365 | 85123A | WHITE HANGING HEART T- | 6 2010-12-01 2.55 17850.0  08:26:00 |
|  |  | LIGHT HOLDER |  |
| **1** 536365 | 71053 | WHITE METAL | 6 2010-12-01 3.39 17850.0  08:26:00 |
|  |  | LANTERN |  |
| **2** 536365 | 84406B | CREAM CUPID HEARTS | 8 2010-12-01 2.75 17850.0  08:26:00 |
|  |  | COAT  HANGER |  |

#function to extract year, month, day as integers def get\_date\_int(df, column):

year = df[column].dt.year

month = df[column].dt.month day = df[column].dt.day

return year, month, day

#extract month

invoice\_year, invoice\_month, \_ = get\_date\_int(retail, 'InvoiceMonth') cohort\_year, cohort\_month, \_ = get\_date\_int(retail, 'CohortMonth')

years\_diff = invoice\_year - cohort\_year

months\_diff = invoice\_month - cohort\_month

# Extract the difference in days from all previous values

retail['CohortIndex'] = years\_diff \* 12 + months\_diff + 1 retail.head()

|  |  |  |  |
| --- | --- | --- | --- |
| **InvoiceNo** | **StockCode** | **Description** | **Quantity InvoiceDate UnitPrice CustomerID** |
| **0** 536365 | 85123A | WHITE HANGING HEART T- | 6 2010-12-01 2.55 17850.0  08:26:00 |
|  |  | LIGHT HOLDER |  |
| **1** 536365 | 71053 | WHITE METAL | 6 2010-12-01 3.39 17850.0  08:26:00 |
|  |  | LANTERN |  |
| **2** 536365 | 84406B | CREAM CUPID HEARTS | 8 2010-12-01 2.75 17850.0  08:26:00 |
|  |  | COAT  HANGER |  |

#count monthly active customers from each cohort

grouping = retail.groupby(['CohortMonth', 'CohortIndex'])

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

cohort\_counts = cohort\_data.pivot(index='CohortMonth', columns = 'CohortIndex', values='Cu

#Customer retention

cohort\_sizes = cohort\_counts.iloc[:,0]

retention = cohort\_counts.divide(cohort\_sizes, axis=0) retention = retention.round(3) \* 100

retention.head(20)

## CohortIndex 1 2 3 4 5 6 7 8 9 10 11 12 1

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

**CohortMonth**

**2010-12-01** 100.0 38.2 33.4 38.7 36.0 39.7 38.0 35.4 35.4 39.5 37.3 50.0 27.

**2011-01-01** 100.0 24.0 28.3 24.2 32.8 29.9 26.1 25.7 31.1 34.7 36.8 15.0 Na

**2011-02-01** 100.0 24.7 19.2 27.9 26.8 24.7 25.5 28.2 25.8 31.3 9.2 NaN Na

**2011-03-01** 100.0 19.1 25.5 21.8 23.2 17.7 26.4 23.9 28.9 8.9 NaN NaN Na

**2011-04-01** 100.0 22.7 22.1 21.1 20.7 23.7 23.1 26.1 8.4 NaN NaN NaN Na

**2011-05-01** 100.0 23.7 17.2 17.2 21.5 24.4 26.5 10.4 NaN NaN NaN NaN Na

**2011-06-01** 100.0 20.9 18.7 27.2 24.7 33.6 10.2 NaN NaN NaN NaN NaN Na

**2011-07-01** 100.0 20.9 20.4 23.0 27.2 11.5 NaN NaN NaN NaN NaN NaN Na

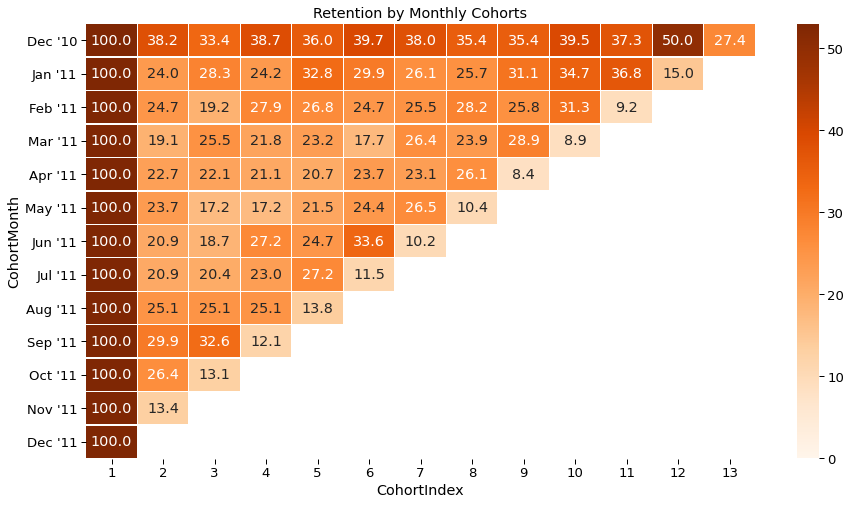
**2011-08-01** 100.0 25.1 25.1 25.1 13.8 NaN NaN NaN NaN NaN NaN NaN Na

**2011-09-01** 100.0 29.9 32.6 12.1 NaN NaN NaN NaN NaN NaN NaN NaN Na

**2011-10-01** 100.0 26.4 13.1 NaN NaN NaN NaN NaN NaN NaN NaN NaN Na

**2011-11-01** 100.0 13.4 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na

**2011-12-01** 100.0 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na



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

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

plt.figure(figsize=(15,8))

plt.title('Retention by Monthly Cohorts') sns.heatmap(data=retention,

annot = True,

cmap = "Oranges", vmin = 0.0,

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

linewidth = 0.3,

yticklabels=month\_list)

plt.show()

# **Project Task: Week 2 Data Modeling : RFM analysis 2.RFM** **analysis**

which customers are the best ones by examining how recently a customer has purchased

(recency), how often they purchase (frequency), and how much the customer spends (monetary)

#12 months of data

print('Min:{}; Max:{}'.format(min(retail.InvoiceDate), max(retail.InvoiceDate)))

Min:2010-12-01 08:26:00; Max:2011-12-09 12:50:00

#calculate revenue per row and add new column

retail['MonetaryValue'] = retail['Quantity'] \* retail['UnitPrice']

retail.MonetaryValue

|  |  |
| --- | --- |
| 0 | 15.30 |
| 1 | 20.34 |
| 2 | 22.00 |
| 3 | 20.34 |
| 4 | 20.34 |
|  | ... |
| 541904 | 10.20 |
| 541905 | 12.60 |
| 541906 | 16.60 |
| 541907 | 16.60 |
| 541908 | 14.85 |

Name: MonetaryValue, Length: 401604, dtype: float64

#let's look at amount spend per customer (revenue contributed) M-Monetary

retail\_mv = retail.groupby(['CustomerID']).agg({'MonetaryValue': sum}).reset\_index() retail\_mv.head()

|  |  |
| --- | --- |
| **CustomerID** | **MonetaryValue** |
| **0** 12346.0 | 0.00 |
| **1** 12347.0 | 4310.00 |
| **2** 12348.0 | 1797.24 |
| **3** 12349.0 | 1757.55 |
| **4** 12350.0 | 334.40 |

#F-frequency (how many purchases each customer made)

retail\_f = retail.groupby('CustomerID')['InvoiceNo'].count() retail\_f = retail\_f.reset\_index()

retail\_f.head()

## CustomerID InvoiceNo

**0** 12346.0 2

**1** 12347.0 182

#merge previous dataframes together (mv+f)

**2** 12348.0 31

**3** 12349.0 73

**4** 12350.0 17

retail\_mv\_f = pd.merge(retail\_mv, retail\_f, on='CustomerID', how='inner') retail\_mv\_f.head()

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **MonetaryValue** | **InvoiceNo** |
| **0** 12346.0 | 0.00 | 2 |
| **1** 12347.0 | 4310.00 | 182 |
| **2** 12348.0 | 1797.24 | 31 |
| **3** 12349.0 | 1757.55 | 73 |
| **4** 12350.0 | 334.40 | 17 |

#R-recency

#last transaction date

retail['InvoiceDate'] = pd.to\_datetime(retail['InvoiceDate'],format='%d-%m-%Y %H:%M') max\_date = max(retail['InvoiceDate'])

#difference between last date and transaction date retail['Diff'] = max\_date - retail['InvoiceDate'] retail.head()

## InvoiceNo StockCode Description Quantity InvoiceDate UnitPrice CustomerID

WHITE

HANGING 2010-12-01

**0** 536365 85123A HEART T- 6 08:26:00 2.55 17850.0

LIGHT

HOLDER

WHITE

**1** 536365 71053 METAL 6 2010-12-01 3.39 17850.0

LANTERN 08:26:00

#recency per customer (last transaction date)

retail\_r = retail.groupby('CustomerID')['Diff'].min() retail\_r = retail\_r.reset\_index()

# Extract number of days only

retail\_r['Diff'] = retail\_r['Diff'].dt.days

CREAM

retail\_rfm.head()

CUPID

**2** 536365 84406B HEARTS 8 2010-12-01 2.75 17850.0

COAT 08:26:00

HANGER

KNITTED UNION

**3** 536365 84029G FLAG HOT 6 2010-12-01 3.39 17850.0

WATER 08:26:00

BOTTLE

RED WOOLLY

**4** 536365 84029E HOTTIE 6 2010-12-01 3.39 17850.0

WHITE 08:26:00

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **MonetaryValue** | **Frequency** | **Recency** |
| **0** 12346.0 | 0.00 | 2 | 325 |
| **1** 12347.0 | 4310.00 | 182 | 1 |
| **2** 12348.0 | 1797.24 | 31 | 74 |
| **3** 12349.0 | 1757.55 | 73 | 18 |
| **4** 12350.0 | 334.40 | 17 | 309 |

#merge R dataframe with FM

HEART.

retail\_rfm = pd.merge(retail\_mv\_f, retail\_r, on='CustomerID', how='inner') retail\_rfm.columns = ['CustomerID', 'MonetaryValue', 'Frequency', 'Recency'] retail\_rfm.head()

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **MonetaryValue** | **Frequency** | **Recency** |
| **0** 12346.0 | 0.00 | 2 | 325 |
| **1** 12347.0 | 4310.00 | 182 | 1 |
| **2** 12348.0 | 1797.24 | 31 | 74 |
| **3** 12349.0 | 1757.55 | 73 | 18 |
| **4** 12350.0 | 334.40 | 17 | 309 |

cols = retail\_rfm.columns.tolist() cols

['CustomerID', 'MonetaryValue', 'Frequency', 'Recency']

#changed columns order

cols = ['CustomerID', 'Recency', 'Frequency', 'MonetaryValue'] retail\_rfm = retail\_rfm[cols]

retail\_rfm.head()

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **Recency Frequency** | **MonetaryValue** |
| **0** 12346.0 | 325 2 | 0.00 |
| **1** 12347.0 | 1 182 | 4310.00 |
| **2** 12348.0 | 74 31 | 1797.24 |
| **3** 12349.0 | 18 73 | 1757.55 |
| **4** 12350.0 | 309 17 | 334.40 |
| # create labels and r\_labels = range(4, | assign them to tree 0, -1) | percentile groups |

r\_groups = pd.qcut(retail\_rfm.Recency, q = 4, labels = r\_labels) f\_labels = range(1, 5)

f\_groups = pd.qcut(retail\_rfm.Frequency, q = 4, labels = f\_labels) m\_labels = range(1, 5)

m\_groups = pd.qcut(retail\_rfm.MonetaryValue, q = 4, labels = m\_labels)

# make a new column for group labels retail\_rfm['R'] = r\_groups.values

retail\_rfm['F'] = f\_groups.values retail\_rfm['M'] = m\_groups.values # sum up the three columns

retail\_rfm['RFM\_Segment'] = retail\_rfm.apply(lambda x: str(x['R']) + str(x['F']) + str(x[' retail\_rfm['RFM\_Score'] = retail\_rfm[['R', 'F', 'M']].sum(axis = 1)

retail\_rfm.head()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **Recency** | **Frequency** | **MonetaryValue** | **R** | **F** | **M** | **RFM\_Segment** | **RFM\_Score** |
| **0** 12346.0 | 325 | 2 | 0.00 | 1 | 1 | 1 | 1.01.01.0 | 3 |
| **1** 12347.0 | 1 | 182 | 4310.00 | 4 | 4 | 4 | 4.04.04.0 | 12 |
| **2** 12348.0 | 74 | 31 | 1797.24 | 2 | 2 | 4 | 2.02.04.0 | 8 |
| **3** 12349.0 | 18 | 73 | 1757.55 | 3 | 3 | 4 | 3.03.04.0 | 10 |
| **4** 12350.0 | 309 | 17 | 334.40 | 1 | 1 | 2 | 1.01.02.0 | 4 |

# assign labels from total score

score\_labels = ['Green', 'Bronze', 'Silver', 'Gold']

score\_groups = pd.qcut(retail\_rfm.RFM\_Score, q = 4, labels = score\_labels) retail\_rfm['RFM\_Level'] = score\_groups.values

retail\_rfm.sort\_values(by='RFM\_Score', ascending=False) retail\_rfm.head(10)

## CustomerID Recency Frequency MonetaryValue R F M RFM\_Segment RFM\_Score RF

**0** 12346.0 325 2 0.00 1 1 1 1.01.01.0 3

**1** 12347.0 1 182 4310.00 4 4 4 4.04.04.0 12

**2** 12348.0 74 31 1797.24 2 2 4 2.02.04.0 8

**3** 12349.0 18 73 1757.55 3 3 4 3.03.04.0 10

**4** 12350.0 309 17 334.40 1 1 2 1.01.02.0 4

retail\_rfm\_levels = retail\_rfm.groupby('RFM\_Level')['CustomerID'].count().reset\_index(name retail\_rfm\_levels.head()

**5** 12352.0 35 95 1545.41 3 3 3 3.03.03.0 9

**6** 12353.0 203 4 89.00 1 1 1 1.01.01.0 3

**7** 12354.0 231 58 1079.40 1 3 3 1.03.03.0 7

**8** 12355.0 213 13 459.40 1 1 2 1.01.02.0 4

**9** 12356.0 22 59 2811.43 3 3 4 3.03.04.0 10

|  |  |
| --- | --- |
| **RFM\_Level** | **counts** |
| **0** Green | 1298 |
| **1** Bronze | 908 |
| **2** Silver | 1322 |
| **3** Gold | 844 |

#let's try to do more detailed segmentation segment\_dict = {

'Best Customers':'444', # Highest frequency as well as monetary value with least 'Loyal Customers':'344', # High frequency as well as monetary value with good rece 'Potential Loyalists':'434', # High recency and monetary value, average frequency

'Big Spenders':'334', # High monetary value but good recency and frequency valu 'At Risk Customers':'244', # Customer's shopping less often now who used to shop a l 'Can’t Lose Them':'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 'Lost Cheap Customers':'122' # Customer's shopped long ago but with less frequency and

}

# Swap the key and value of dictionary

dict\_segment = dict(zip(segment\_dict.values(),segment\_dict.keys()))

# Allocate segments to each customer as per the RFM score mapping

retail\_rfm['Segment'] = retail\_rfm.RFM\_Segment.map(lambda x: dict\_segment.get(x))

# Allocate all remaining customers to others segment category retail\_rfm.Segment.fillna('others', inplace=True)

retail\_rfm.sample(10)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **CustomerID** | **Recency** | **Frequency** | **MonetaryValue** | **R** | **F** | **M** | **RFM\_Segment** | **RFM\_Score** |
| **3351** | 16881.0 | 66 | 1 | 432.00 | 2 | 1 | 2 | 2.01.02.0 | 5 |
| **570** | 13091.0 | 20 | 24 | 319.82 | 3 | 2 | 2 | 3.02.02.0 | 7 |
| **4132** | 17958.0 | 116 | 7 | 508.46 | 2 | 1 | 2 | 2.01.02.0 | 5 |
| **1790** | 14764.0 | 43 | 7 | 250.55 | 3 | 1 | 1 | 3.01.01.0 | 5 |
| **1269** | 14052.0 | 19 | 56 | 225.36 | 3 | 3 | 1 | 3.03.01.0 | 7 |
| **1597** | 14503.0 | 2 | 164 | 3543.26 | 4 | 4 | 4 | 4.04.04.0 | 12 |
| **3602** | 17234.0 | 182 | 14 | 149.89 | 1 | 1 | 1 | 1.01.01.0 | 3 |

retail\_rfm\_levels

**465** 12940.0 45 100 862.44 3 4 3 3.04.03.0 10

**1851** 14854.0 77 130 2730.09 2 4 4 2.04.04.0 10

**3428** 16985.0 16 121 5461.62 4 4 4 4.04.04.0 12

retail\_rfm\_segments = retail\_rfm[retail\_rfm.Segment!='other'].groupby('Segment')['Customer

retail\_rfm\_segments.iloc[:8]

**Segment counts**

**0**

others

4372

|  |  |
| --- | --- |
| **RFM\_Level** | **counts** |
| **0** Green | 1298 |
| **1** Bronze | 908 |
| **2** Silver | 1322 |
| **3** Gold | 844 |

# **Project Task: Week 3 Data Modeling : Create clusters using k-means clustering algorithm.**

# **3.k-Means Clustering**

# copying the data into new variable df\_kmeans = retail\_rfm.copy()

# taking only relevant columns df\_kmeans = df\_kmeans.iloc[:,:4] df\_kmeans.head()

**CustomerID Recency Frequency MonetaryValue**

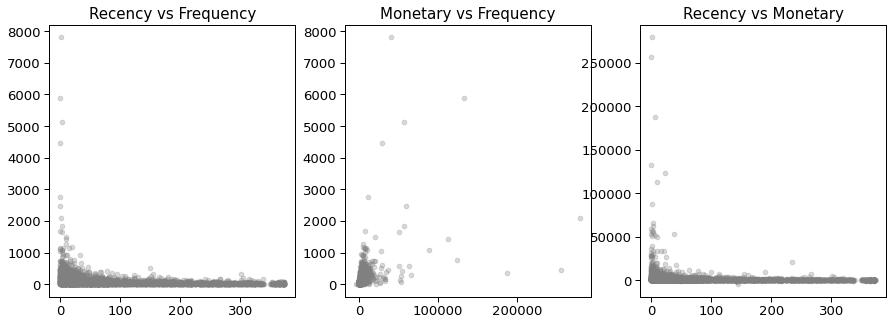
**0** 12346.0 325 2 0.00

**1** 12347.0 1 182 4310.00

**2** 12348.0 74 31 1797.24

**3** 12349.0 18 73 1757.55

**4** 12350.0 309 17 334.40



plt.figure(figsize=(15,5)) plt.subplot(1,3,1)

plt.scatter(df\_kmeans.Recency, df\_kmeans.Frequency, color='grey', alpha=0.3) plt.title('Recency vs Frequency', size=15)

plt.subplot(1,3,2)

plt.scatter(df\_kmeans.MonetaryValue, df\_kmeans.Frequency, color='grey', alpha=0.3) plt.title('Monetary vs Frequency', size=15)

plt.subplot(1,3,3)

plt.scatter(df\_kmeans.Recency, df\_kmeans.MonetaryValue, color='grey', alpha=0.3) plt.title('Recency vs Monetary', size=15)

plt.show()

# checking the distribution of the variables

column = ['Recency','Frequency','MonetaryValue'] plt.figure(figsize=(15,5))

for i,j in enumerate(column): plt.subplot(1,3,i+1)

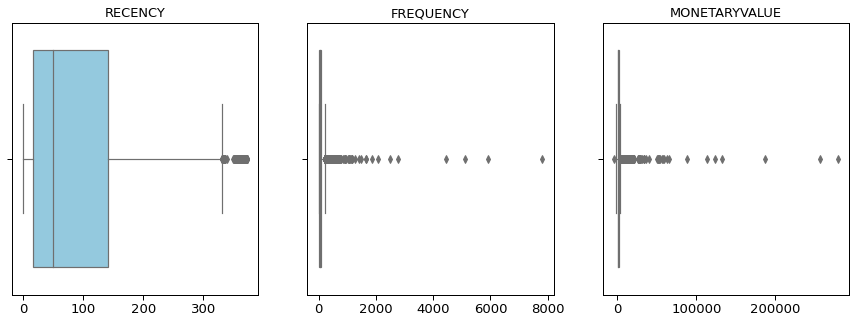
sns.boxplot(df\_kmeans[j], color='skyblue') plt.xlabel('')

plt.title('{}'.format(j.upper()), size=13) plt.show()

/usr/local/lib/python3.7/dist-packages/seaborn/\_decorators.py:43: FutureWarning: Pas FutureWarning

/usr/local/lib/python3.7/dist-packages/seaborn/\_decorators.py:43: FutureWarning: Pas FutureWarning

/usr/local/lib/python3.7/dist-packages/seaborn/\_decorators.py:43: FutureWarning: Pas FutureWarning



# Removing outliers for Monetary

Q1 = df\_kmeans.MonetaryValue.quantile(0.05) Q3 = df\_kmeans.MonetaryValue.quantile(0.95)

IQR = Q3 - Q1

df\_kmeans = df\_kmeans[(df\_kmeans.MonetaryValue >= Q1 - 1.5\*IQR) & (df\_kmeans.MonetaryValue

# Removing outliers for Recency

Q1 = df\_kmeans.Recency.quantile(0.05) Q3 = df\_kmeans.Recency.quantile(0.95)

IQR = Q3 - Q1

df\_kmeans = df\_kmeans[(df\_kmeans.Recency >= Q1 - 1.5\*IQR) & (df\_kmeans.Recency <= Q3 + 1.5

# Removing outliers for Frequency

Q1 = df\_kmeans.Frequency.quantile(0.05) Q3 = df\_kmeans.Frequency.quantile(0.95)

IQR = Q3 - Q1

df\_kmeans = df\_kmeans[(df\_kmeans.Frequency >= Q1 - 1.5\*IQR) & (df\_kmeans.Frequency <= Q3 +

# resetting the index

df\_kmeans = df\_kmeans.reset\_index(drop=True) df\_kmeans.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 4295 entries, 0 to 4294

Data columns (total 4 columns):

# Column Non-Null Count Dtype

* + - 1. CustomerID 4295 non-null float64
      2. Recency 4295 non-null int64
      3. Frequency 4295 non-null int64
      4. MonetaryValue 4295 non-null float64 dtypes: float64(2), int64(2)

memory usage: 134.3 KB

# looking at random 5 rows df\_kmeans.sample(5)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **CustomerID** | **Recency** | **Frequency** | **MonetaryValue** |
| **3151** | 16696.0 | 134 | 10 | 657.90 |
| **1661** | 14632.0 | 262 | 8 | 114.56 |
| **4065** | 17968.0 | 373 | 81 | 265.10 |

from sklearn.preprocessing import StandardScaler

**2091** 15234.0 276 14 197.00

**4077** 17984.0 144 48 152.68



# removing customer id as it will not used in making cluster

df\_kmeans=df\_kmeans.drop(['CustomerID'],axis=1)

KeyError

Traceback (most recent call last)

<ipython-input-92-a30c62376666> in <module>()

1 # removing customer id as it will not used in making cluster

----> 2 df\_kmeans=df\_kmeans.drop(['CustomerID'],axis=1)

3 frames

/usr/local/lib/python3.7/dist-packages/pandas/core/indexes/base.py in drop(self, labels, errors)

5285 if mask.any():

5286 if errors != "ignore":

-> 5287 raise KeyError(f"{labels[mask]} not found in axis") 5288 indexer = indexer[~mask]

5289 return self.delete(indexer) KeyError: "['CustomerID'] not found in axis"

SEARCH STACK OVERFLOW

|  |  |  |
| --- | --- | --- |
| df\_kmeans.head()  **Recency Frequency** | **MonetaryValue** |  |
| **0** 325 2 | 0.00 |  |
| **1** 1 182 | 4310.00 |  |
| **2** 74 31 | 1797.24 |  |
| **3** 18 73 | 1757.55 |  |
| **4** 309 17 | 334.40 |  |
| # scaling the variables and | store it in different | df |

standard\_scaler = StandardScaler()

df\_kmeans\_norm = standard\_scaler.fit\_transform(df\_kmeans)

# converting it into dataframe

df\_kmeans\_norm = pd.DataFrame(df\_kmeans\_norm)

df\_kmeans\_norm.columns = ['Recency','Frequency','MonetaryValue'] df\_kmeans\_norm.head()

|  |  |  |
| --- | --- | --- |
| **Recency** | **Frequency** | **MonetaryValue** |
| **0** 2.302166 | -0.750830 | -0.722542 |
| **1** -0.906150 | 1.057112 | 1.735232 |
| **2** -0.183289 | -0.459551 | 0.302333 |
| **3** -0.737813 | -0.037698 | 0.279700 |

# Initially without any knowledge we are clustering the data into 5 clusters. The only intution to do is as in RFM we categorize the data into 5 categories. Later we look different methods to decide the optimal value for k.

**4** 2.143731 -0.600168 -0.531850

!pip3 install KMeans

Requirement already satisfied: KMeans in /usr/local/lib/python3.7/dist-packages (1.0



!pip install scikit-learn

Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-package Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.7/dist-packag Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-package Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.7/dist-packag



from sklearn import cluster

from sklearn.cluster import KMeans

# Kmeans with K=5

model\_clus5 = KMeans(n\_clusters = 5) model\_clus5.fit(df\_kmeans\_norm)

KMeans(algorithm='auto', copy\_x=True, init='k-means++', max\_iter=300,

n\_clusters=5, n\_init=10, n\_jobs=None, precompute\_distances='auto', random\_state=None, tol=0.0001, verbose=0)

# checking the labels model\_clus5.labels\_

array([2, 3, 1, ..., 1, 0, 1], dtype=int32)

df\_kmeans['clusters'] = model\_clus5.labels\_ df\_kmeans.head()

|  |  |  |  |
| --- | --- | --- | --- |
| **Recency** | **Frequency** | **MonetaryValue** | **clusters** |
| **0** 325 | 2 | 0.00 | 2 |
| **1** 1 | 182 | 4310.00 | 3 |
| **2** 74 | 31 | 1797.24 | 1 |

df\_kmeans.groupby('clusters').mean().round(0)

**3** 18 73 1757.55 1

**4** 309 17 334.40 2

|  |  |  |  |
| --- | --- | --- | --- |
| **Clusters** | **Recency** | **Frequency** | **MonetaryValue** |
| **0** | 19.0 | 381.0 | 4018.0 |
| **1** | 50.0 | 40.0 | 633.0 |
| **2** | 252.0 | 25.0 | 387.0 |
| **3** | 30.0 | 144.0 | 2497.0 |
| **4** | 20.0 | 297.0 | 9332.0 |

# Finding the Optimal Number of Clusters

Elbow Curve to get the right number of Clusters

# Elbow-curve/SSD ssd = []

for num\_clusters in list(range(1,11)):

model\_clus = KMeans(n\_clusters = num\_clusters, max\_iter=50) model\_clus.fit(df\_kmeans\_norm)

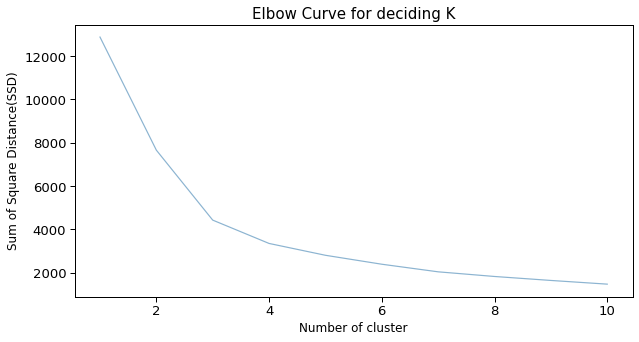
ssd.append(model\_clus.inertia\_)

# plot the SSDs for each n\_clusters plt.figure(figsize=(10,5))

plt.plot(np.arange(1,11,1), ssd)

plt.xlabel('Number of cluster', size=12)

plt.ylabel('Sum of Square Distance(SSD)', size=12) plt.title('Elbow Curve for deciding K', size=15) plt.show()



# Silhouette analysis

for num\_clusters in list(range(2,11)): # intialise kmeans

model\_clus = KMeans(n\_clusters = num\_clusters, max\_iter=50) model\_clus.fit(df\_kmeans\_norm)

cluster\_labels = model\_clus.labels\_

from sklearn.metrics import silhouette\_score

# silhouette score

silhouette\_avg = silhouette\_score(df\_kmeans\_norm, cluster\_labels)

print("For n\_clusters={0}, the silhouette score is {1}".format(num\_clusters, silhouette\_av

For n\_clusters=10, the silhouette score is 0.3814908101545369

# From the elbow curve we observe the elbow at cluster 3 and cluster 4. Also from Silhouette analysis we see the value is better when number of cluster will be 3 rather than 4. So we now categorize the data into 3 clusters and check their RFM values and its distribution.

# Kmeans with K=3

model\_clus3 = KMeans(n\_clusters = 3) model\_clus3.fit(df\_kmeans\_norm)

KMeans(algorithm='auto', copy\_x=True, init='k-means++', max\_iter=300,

n\_clusters=3, n\_init=10, n\_jobs=None, precompute\_distances='auto', random\_state=None, tol=0.0001, verbose=0)

df\_kmeans['clusters'] = model\_clus3.labels\_ df\_kmeans.head()

## Recency Frequency MonetaryValue clusters

df\_kmeans.groupby('clusters').mean().round(0)

**0** 325 2 0.00 0

**1** 1 182 4310.00 2

**2** 74 31 1797.24 1

**3** 18 73 1757.55 1

**4** 309 17 334.40 0

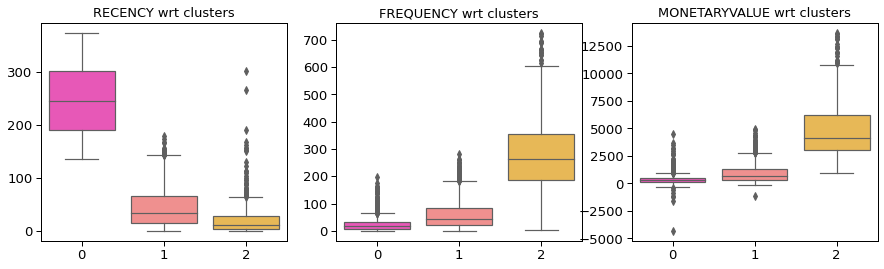
|  |  |  |  |
| --- | --- | --- | --- |
| **Clusters** | **Recency** | **Frequency** | **MonetaryValue** |
| **0** | 248.0 | 25.0 | 387.0 |
| **1** | 44.0 | 59.0 | 948.0 |
| **2** | 22.0 | 285.0 | 4901.0 |

column = ['Recency','Frequency','MonetaryValue'] plt.figure(figsize=(15,4))

for i,j in enumerate(column): plt.subplot(1,3,i+1)

sns.boxplot(y=df\_kmeans[j], x=df\_kmeans['clusters'], palette='spring') plt.title('{} wrt clusters'.format(j.upper()), size=13)

plt.ylabel('')

plt.xlabel('') plt.show()

# Creating figure

fig = plt.figure(figsize = (8, 5)) ax = plt.axes(projection ="3d")

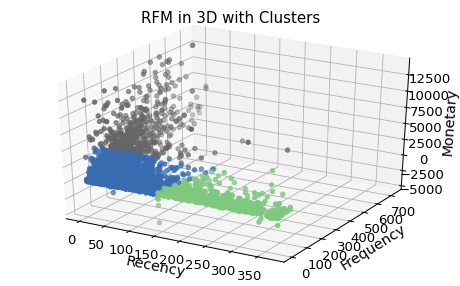
# Creating plot

ax.scatter3D(df\_kmeans.Recency, df\_kmeans.Frequency, df\_kmeans.MonetaryValue, c=df\_kmeans. ax.set\_xlabel('Recency')

ax.set\_ylabel('Frequency') ax.set\_zlabel('Monetary')

plt.title('RFM in 3D with Clusters', size=15) ax.set(facecolor='white')

plt.show()



# Observations: In the above 3D graph, I put all the three variable into 3 axis and added the cluster variable to differentiate the points. Grey points is the group of customers whose Recency is high,

Frequency is low and Monetary value is also low. Green points are the group of customers

whose Recency is low, Frequency is better than grey ones and Monetary is good. Blue points are the group of customers whose Recency is low(that is good), Frequency is better than the other two and Monetary is high.

### **Project Task: Week 4**

**Data Reporting:​​​​​​​ Tableau**

Please find the below link for the tableau dashboard/

Link: https://public.tableau.com/app/profile/sunil3466/viz/OnlineRetailRFManalysis/RFMAnalysis

