RetailAnalysis

October 10, 2023

[187]:
```python
import pandas as pd
import numpy as np
import datetime
import seaborn as sns
from matplotlib import pyplot as plt
```

[188]:
```python
data = pd.read_excel("Online Retail.xlsx")
```

[189]:
```python
data.describe()
```

[189]:
```
            Quantity      UnitPrice     CustomerID
count  541909.000000  541909.000000  406829.000000
mean        9.552250       4.611114   15287.690570
std       218.081158      96.759853    1713.600303
min    -80995.000000  -11062.060000   12346.000000
25%         1.000000       1.250000   13953.000000
50%         3.000000       2.080000   15152.000000
75%        10.000000       4.130000   16791.000000
max     80995.000000   38970.000000   18287.000000
```

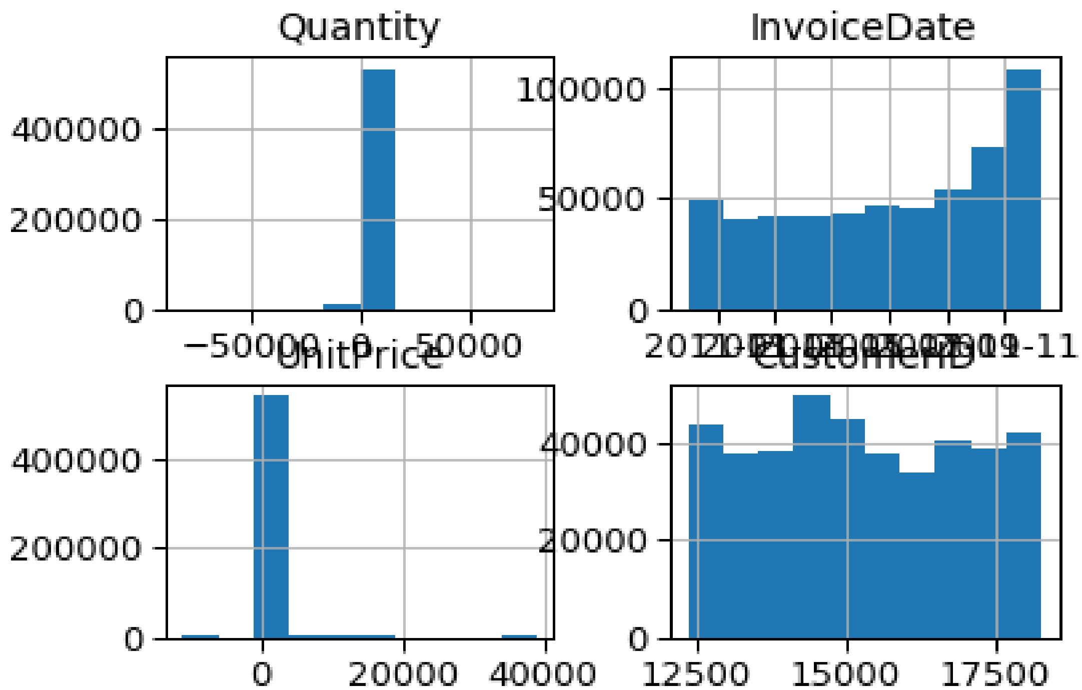
[190]:
```python
data.head()
```

[190]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

          InvoiceDate  UnitPrice  CustomerID         Country
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
```

[191]:
```python
data.hist()
```

[191]: array([[<AxesSubplot: title={'center': 'Quantity'}>,
            <AxesSubplot: title={'center': 'InvoiceDate'}>],
           [<AxesSubplot: title={'center': 'UnitPrice'}>,
            <AxesSubplot: title={'center': 'CustomerID'}>]], dtype=object)



[192]:
```
data.columns
```

[192]:
```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
```

[193]:
```
data.isnull().sum()
```

[193]:
```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

[194]:
```
#data["Description"].fillna("No description", inplace = True)
print ("No of records before dropping customer ID column")
print (len(data))
data.drop(data[data['CustomerID'].isna()].index, inplace = True)
data.reset_index(drop=True)
print ("No of records after dropping customer ID column")
print (len(data))
```

```
No of records before dropping customer ID column
541909
No of records after dropping customer ID column
406829
```

[195]:
```
#data['CustomerID'].fillna("Unknwon Customer",inplace = True)
print ("Is there any missing data in Description column after dropping the Null␣
↪Customer ID columns")
print (any(data['Description'].isna()==True))
missingdf = pd.DataFrame({'Columns' : data.columns.to_list(), 'No of missing␣
↪data after cleaning' : data.isna().sum()})
missingdf.style.hide_index()
```

```
Is there any missing data in Description column after dropping the Null Customer
ID columns
False
```

```
/tmp/ipykernel_84/2345165740.py:5: FutureWarning: this method is deprecated in
favour of `Styler.hide(axis="index")`
  missingdf.style.hide_index()
```

[195]: <pandas.io.formats.style.Styler at 0x7fc567efa170>

[196]:
```
data.isnull().sum(axis=0)
#axis tell to sum across rows
```

[196]:
```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

[197]:
```
len(data)
```

[197]: 406829

[198]:
```python
data.drop_duplicates(inplace = True)
```

[199]:
```python
len(data)
```

[199]:
```
401604
```

[200]:
```python
Unique_Code = data['StockCode'].unique()
len(Unique_Code)
```

[200]:
```
3684
```

[201]:
```python
group_by_Country = pd.DataFrame(data.groupby("Country") ["CustomerID"].
 ↪nunique())
group_by_Country.head(5)
```

[201]:

	CustomerID
Country	
Australia	9
Austria	11
Bahrain	2
Belgium	25
Brazil	1

[202]:
```python
high_customer = pd.DataFrame(group_by_Country).sort_values(by=['CustomerID']␣
 ↪,ascending = False)
high_customer.head(5)
```

[202]:

	CustomerID
Country	
United Kingdom	3950
Germany	95
France	87
Spain	31
Belgium	25

[203]:
```python
Low_customer = pd.DataFrame(group_by_Country).sort_values(by=['CustomerID']␣
 ↪,ascending = True)
Low_customer.head(5)
```

[203]:

	CustomerID
Country	
European Community	1
Lebanon	1
Iceland	1
RSA	1
Brazil	1

[204]:
```
group_by_customer = pd.DataFrame(data.groupby("CustomerID") ["InvoiceNo"].
  ↪nunique())
len(group_by_customer)
```

[204]: 4372

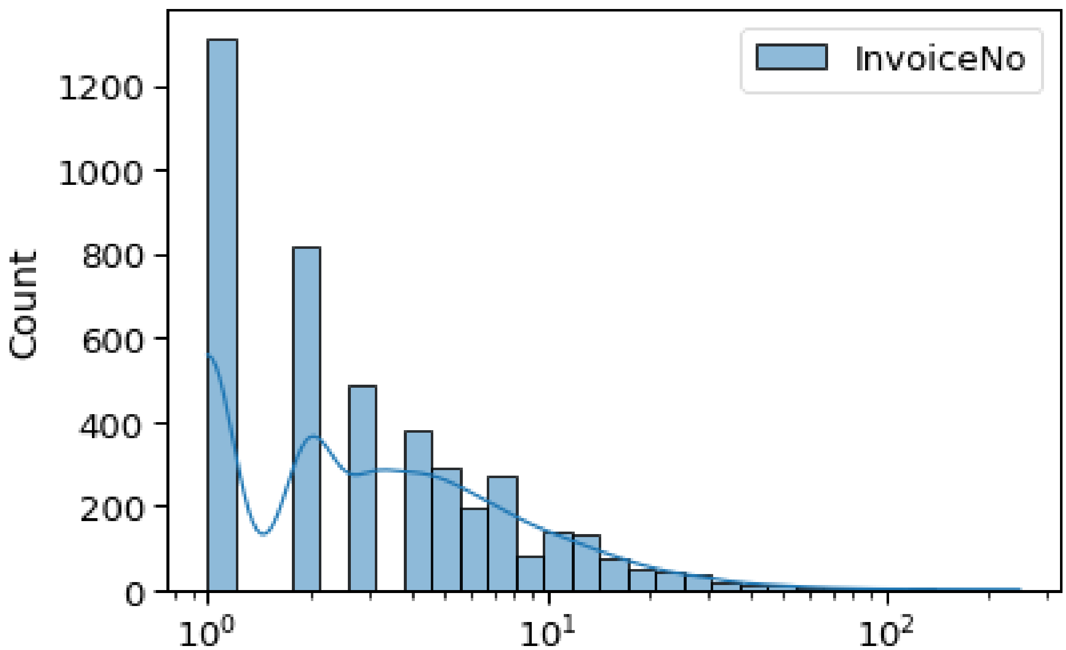
[205]:
```
bill_twice_perc = np.sum(group_by_customer > 1)/data['CustomerID'].nunique()
bill_twice_perc *100
```

[205]:
```
InvoiceNo    69.967978
dtype: float64
```

69% of customer ordered more than once

[206]:
```
sns.histplot(group_by_customer, kde = True,log_scale = True)
```

[206]: <AxesSubplot: ylabel='Count'>



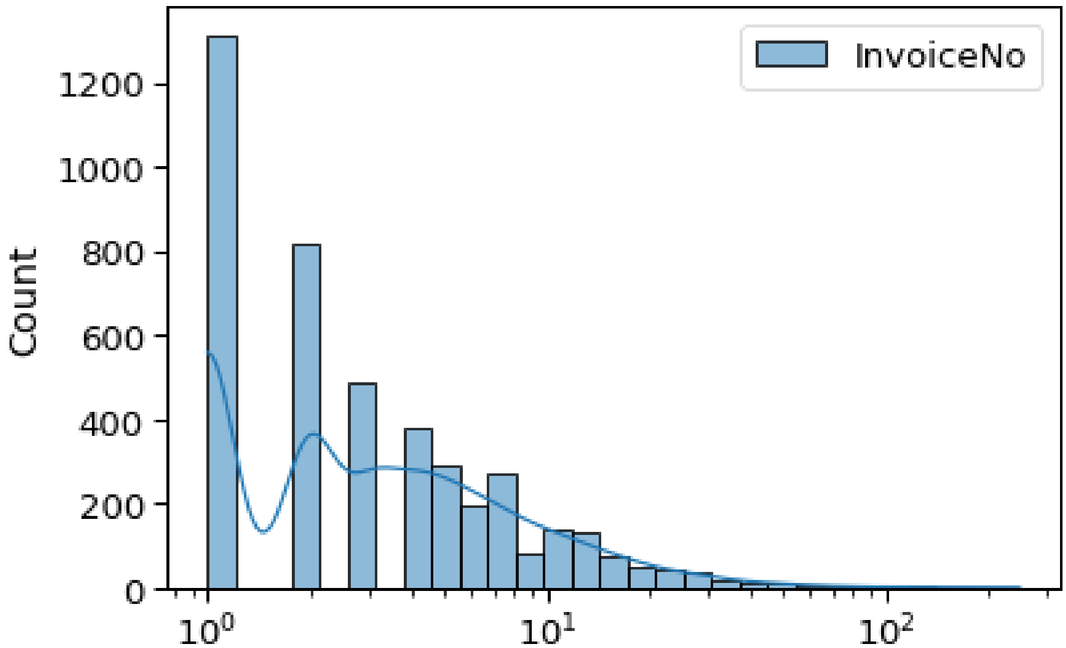
[207]:
```
bill_thrice_perc = np.sum(group_by_customer > 2)/data['CustomerID'].nunique()
bill_thrice_perc *100
```

[207]:
```
InvoiceNo    51.280878
dtype: float64
```

51% of cusotmer ordered thrice

[208]:
```
sns.histplot(group_by_customer, kde = True , log_scale = True)
```

[208]: <AxesSubplot: ylabel='Count'>



[209]:
```
group_by_code_high = pd.DataFrame(data.groupby("Country")["StockCode"].
    ↪nunique())
group_by_code_high.sort_values( by=["StockCode"] , ascending = False).head(10)
```

[209]:

Country	StockCode
United Kingdom	3661
EIRE	1950
Germany	1671
France	1523
Spain	1093
Switzerland	947
Netherlands	785
Belgium	778
Portugal	686
Australia	600

Top 10 countries with maximum order count

[210]:
```
group_by_code_low = pd.DataFrame(data.groupby("Country")["StockCode"].nunique())
group_by_code_low.sort_values( by = ["StockCode"], ascending = True).head(10)
```

```
[210]:                       StockCode
Country
Saudi Arabia                         9
Bahrain                             16
Czech Republic                      25
Lithuania                           29
Brazil                              32
Lebanon                             45
European Community                  50
RSA                                 58
United Arab Emirates                68
Malta                               99
```

Bottom 10 countries with lowest order count

```
[211]: data_columns = data.dtypes.reset_index()
data_columns.columns = ["Feature type", "data type"]
data_columns.groupby("data type").agg("count").reset_index()
```

```
[211]:         data type  Feature type
0           int64             1
1  datetime64[ns]             1
2         float64             2
3          object             4
```

```
[212]: data_object= data.select_dtypes(include=[object])
data_object.head(3)
```

```
[212]:   InvoiceNo StockCode                          Description         Country
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER  United Kingdom
1    536365     71053                  WHITE METAL LANTERN  United Kingdom
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER  United Kingdom
```

```
[213]: data_object.isnull().sum()
```

```
[213]: InvoiceNo      0
StockCode      0
Description    0
Country        0
dtype: int64
```

```
[214]: data_float = data.select_dtypes(include=[float])
data_float.head(3)
```

```
[214]:    UnitPrice  CustomerID
0       2.55     17850.0
1       3.39     17850.0
2       2.75     17850.0
```

[215]:
```
data_float.isnull().sum()
```

[215]:
```
UnitPrice     0
CustomerID    0
dtype: int64
```

[216]:
```
data_int = data.select_dtypes(include=[int])
data_int.head(3)
```

[216]:

	Quantity
0	6
1	6
2	8

[217]:
```
data_int.isnull().sum()
```

[217]:
```
Quantity    0
dtype: int64
```

[218]:
```
data.columns
```

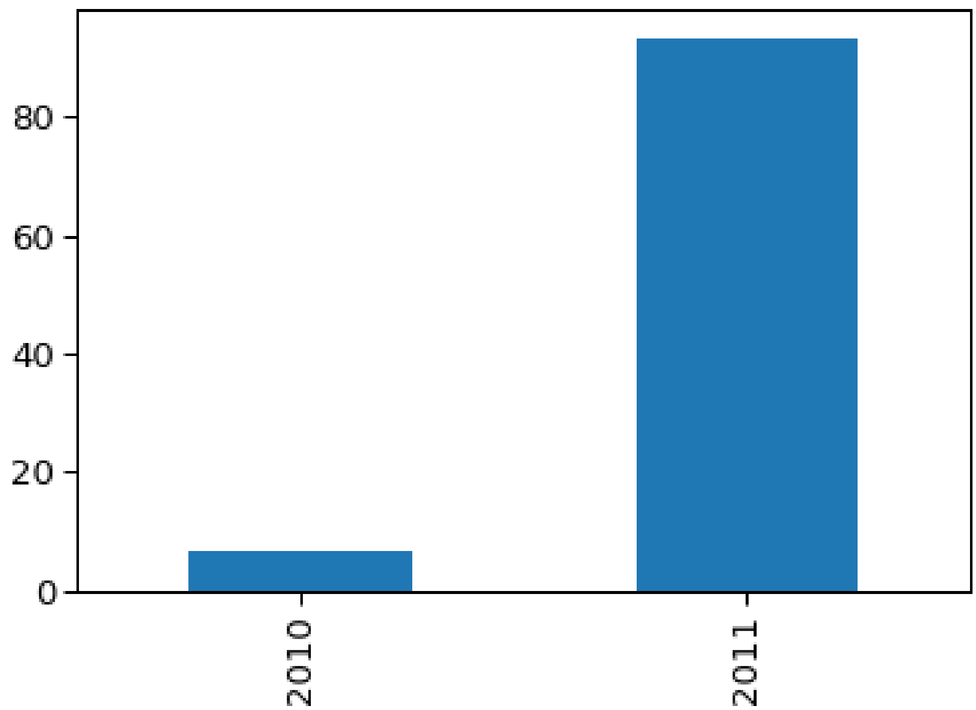
[218]:
```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
```

[219]:
```
data.Country.value_counts(normalize=True).head(10).mul(100).round(2).
↪astype(str) + ' %'
```

[219]:
```
United Kingdom    88.83 %
Germany            2.36 %
France             2.11 %
EIRE               1.86 %
Spain              0.63 %
Netherlands        0.59 %
Belgium            0.52 %
Switzerland        0.47 %
Portugal           0.37 %
Australia          0.31 %
Name: Country, dtype: object
```

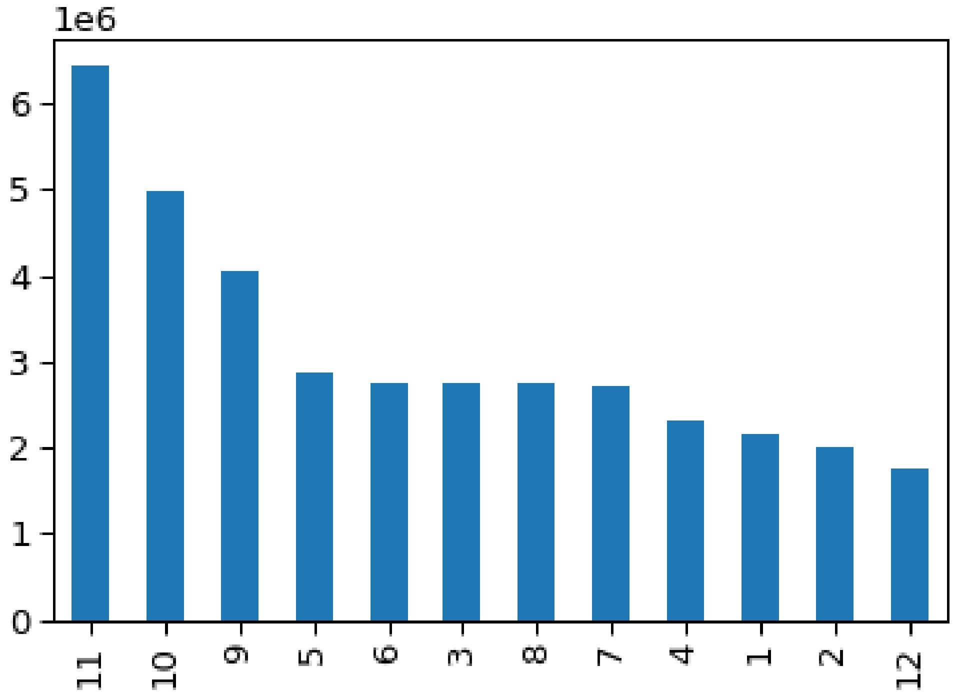
[220]:
```
data.InvoiceDate.dt.year.value_counts(normalize = True,sort = False).mul(100).
↪round(2).plot(kind='bar')
```

[220]:
```
<AxesSubplot: >
```

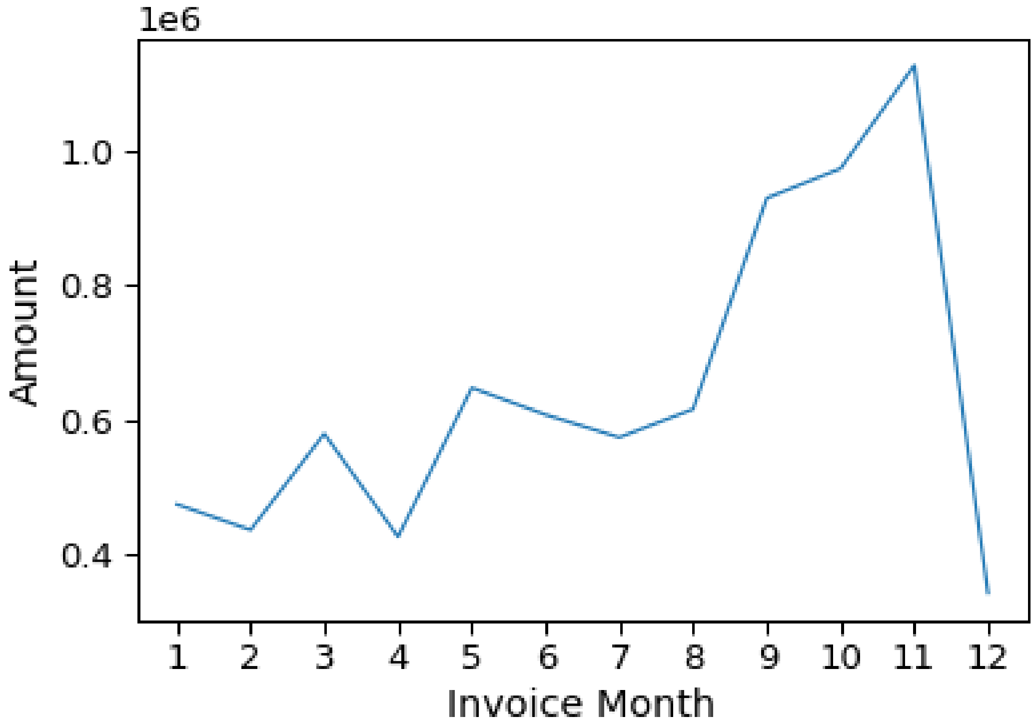


[221]:
```
data["Amount"] = data.UnitPrice * data.Quantity
month_amount =data[data.InvoiceDate.dt.year==2011].groupby(data.InvoiceDate.dt.
 ↪month).Amount.sum()
data[data.InvoiceDate.dt.year==2011].InvoiceDate.dt.month.value_counts(sort =␣
 ↪True).mul(100).round(2).plot(kind='bar')
month_amount
```

[221]:
```
InvoiceDate
1      473731.900
2      435534.070
3      578576.210
4      425222.671
5      647011.670
6      606862.520
7      573112.321
8      615078.090
9      929356.232
10     973306.380
11    1126815.070
12     341539.430
Name: Amount, dtype: float64
```

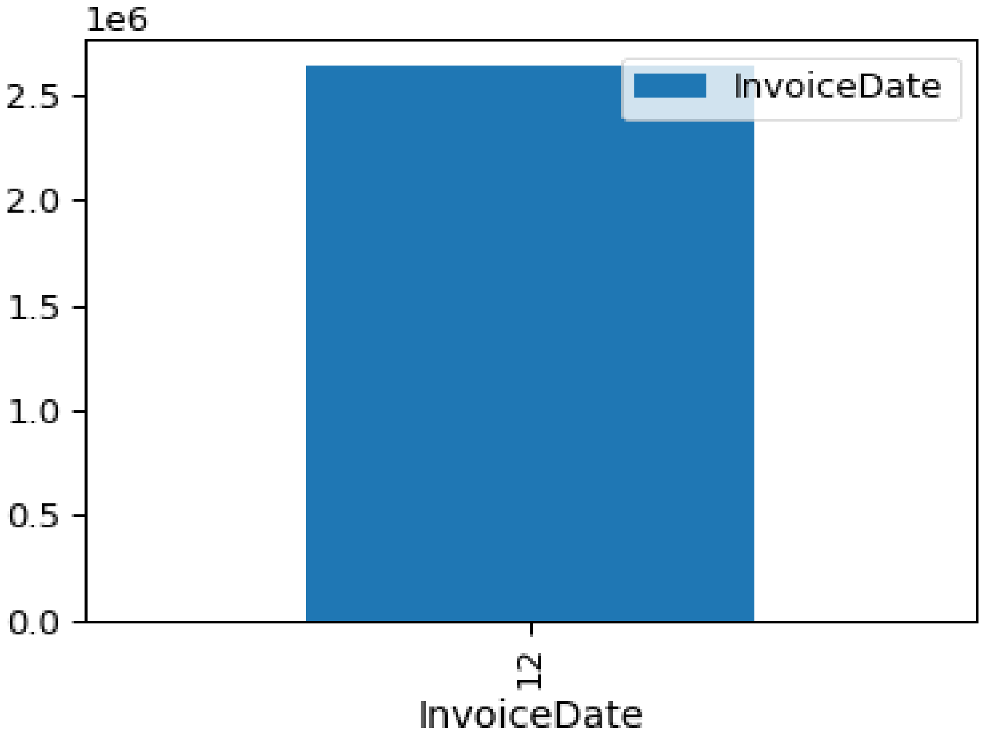


[222]:
```
sns.lineplot(y=month_amount.values,x=month_amount.index)
plt.xlabel('Invoice Month')
plt.ylabel("Amount")
plt.xticks(range(1,13))
plt.show()
```



[223]:
```
data[data.InvoiceDate.dt.year==2010].InvoiceDate.dt.month.value_counts(sort =
 ↪False).mul(100).round(2).plot(kind='bar')
month_amount2 =data[data.InvoiceDate.dt.year==2010].groupby(data.InvoiceDate.dt.
 ↪month).Amount.sum()
month_amount2
sns.lineplot(y=month_amount2.values,x=month_amount2.index)
```

[223]: <AxesSubplot: xlabel='InvoiceDate'>

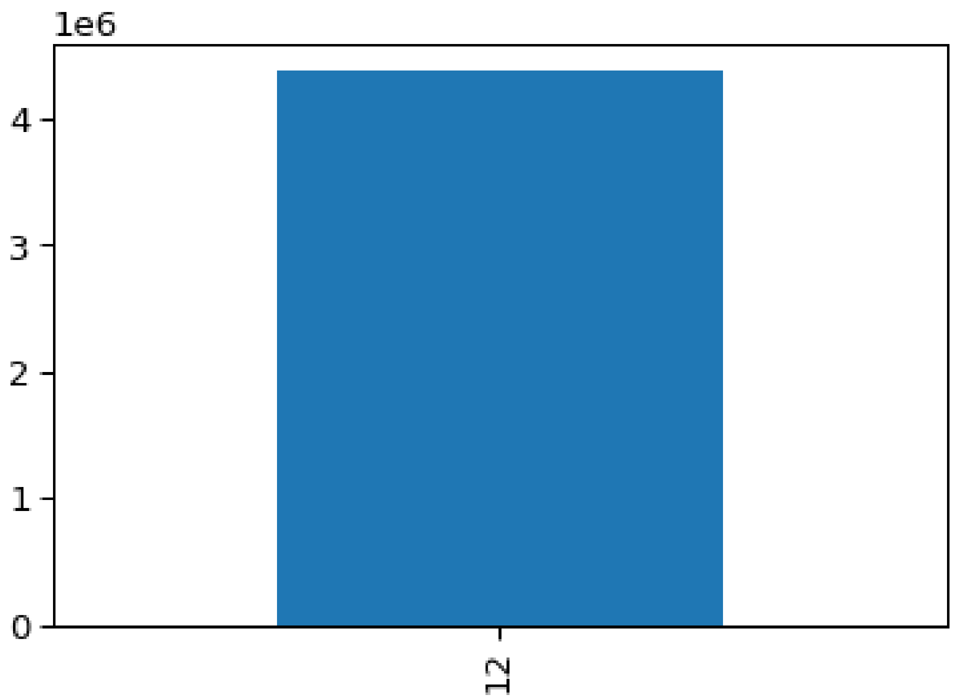


[224]:
```
month_amount2 =data[data.InvoiceDate.dt.year==2010].groupby(data.InvoiceDate.dt.
↪month).Amount.sum()
month_amount2
#sns.lineplot(y=month_amount2.values,x=month_amount2.index)
```

[224]:
```
InvoiceDate
12    552372.86
Name: Amount, dtype: float64
```

[225]:
```
data[data.InvoiceDate.dt.month==12].InvoiceDate.dt.month.value_counts(sort =␣
↪False).mul(100).round(2).plot(kind='bar')
```

[225]:
```
<AxesSubplot: >
```



[226]:
```
data.columns
```

[226]:
```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country', 'Amount'],
      dtype='object')
```

[227]:
```
group_by_amount_high = pd.DataFrame(data.groupby("Country")["Amount"].sum())
```

[228]:
```
group_by_amount_high.sort_values(by =["Amount"], ascending = False).head(5).
 ↪round()
```

[228]:
```
                    Amount
Country
United Kingdom  6747156.0
Netherlands      284662.0
EIRE             250002.0
Germany          221509.0
France           196626.0
```

[229]:
```
group_by_amount_low= pd.DataFrame(data.groupby("Country") ["Amount"].sum())
```

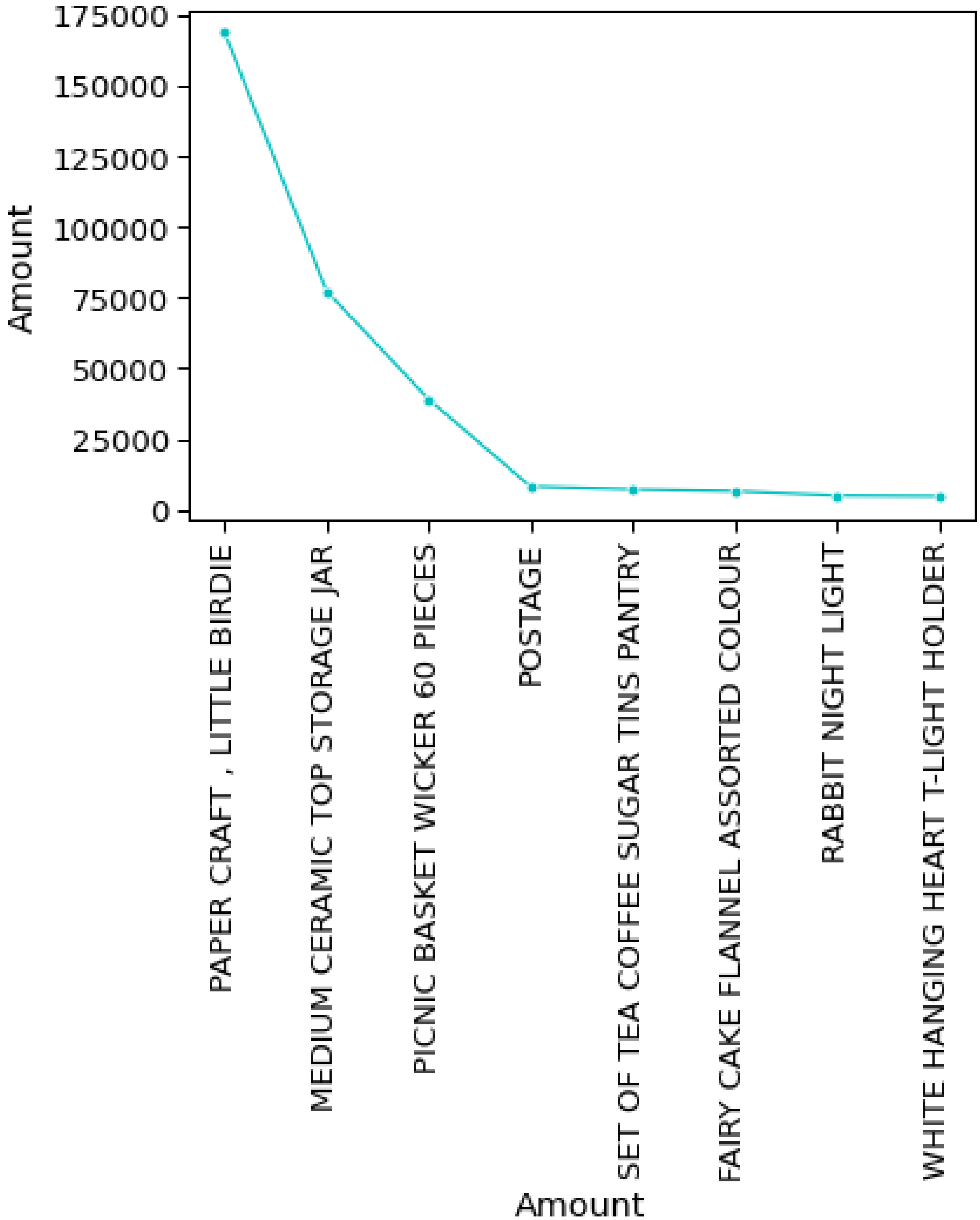
[230]:
```
group_by_amount_low.sort_values(by = ["Amount"],ascending = True).head(5).
 ↪round(0)
```

[230]:

	Amount
Country	
Saudi Arabia	131.0
Bahrain	548.0
Czech Republic	708.0
RSA	1002.0
Brazil	1144.0

[231]:

```
desc = data.sort_values(by='Amount', ascending=False)['Description'].head(10)
price = data.sort_values(by='Amount', ascending=False)['Amount'].head(10)
sns.lineplot(y=price,x=desc, marker='o', color='c')
plt.xticks(rotation=90)
plt.xlabel('Description')
plt.xlabel("Amount")
plt.show()
```



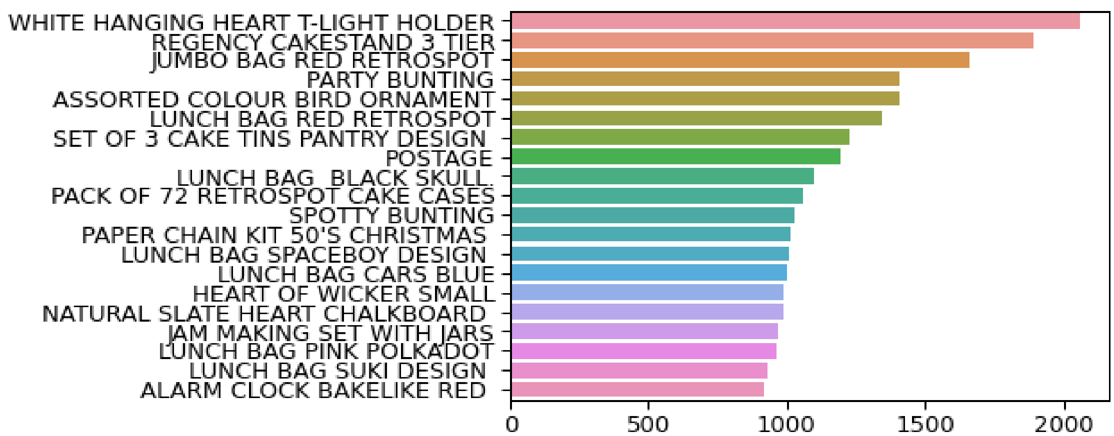
[232]:
```
print ("First business transaction date is {}".format(data.InvoiceDate.min()))
print ("Last business transaction  date is {}".format(data.InvoiceDate.max()))
```

```
First business transaction date is 2010-12-01 08:26:00
Last business transaction  date is 2011-12-09 12:50:00
```

[233]:
```
#pd.DataFrame(data["Description"].value_counts())
top_products = data['Description'].value_counts()[:20]
top_products
```

[233]:
```
WHITE HANGING HEART T-LIGHT HOLDER    2058
REGENCY CAKESTAND 3 TIER              1894
JUMBO BAG RED RETROSPOT               1659
PARTY BUNTING                         1409
ASSORTED COLOUR BIRD ORNAMENT         1405
LUNCH BAG RED RETROSPOT               1345
SET OF 3 CAKE TINS PANTRY DESIGN      1224
POSTAGE                               1196
LUNCH BAG  BLACK SKULL.               1099
PACK OF 72 RETROSPOT CAKE CASES       1062
SPOTTY BUNTING                        1026
PAPER CHAIN KIT 50'S CHRISTMAS        1013
LUNCH BAG SPACEBOY DESIGN             1006
LUNCH BAG CARS BLUE                   1000
HEART OF WICKER SMALL                  990
NATURAL SLATE HEART CHALKBOARD         989
JAM MAKING SET WITH JARS               966
LUNCH BAG PINK POLKADOT                961
LUNCH BAG SUKI DESIGN                  932
ALARM CLOCK BAKELIKE RED               917
Name: Description, dtype: int64
```

[234]:
```
sns.barplot(y=top_products.index,x=top_products.values)
plt.figure(figsize = (10,7))
sns.set_context("paper", font_scale=1.5)
plt.show()
```




```
<Figure size 720x504 with 0 Axes>
```

[235]:
```
return_products = pd.DataFrame(data['Quantity']<0)
return_products.value_counts(normalize = True).mul(100).round(1).astype(str) +␣
↪' %'
```

[235]:
```
Quantity
False       97.8 %
True         2.2 %
dtype: object
```

2.2% are returned products which is insignificant in terms of count

[236]:
```
#returned product amount-todo
#outlier detection in retail analysis

def outlierDetection(datacolumn):
    #Sort the data in ascending order
    sorted(datacolumn)

    #GET Q1 and Q3
    Q1,Q3 = np.percentile(datacolumn, [25,75])

    #Calc IQR
    IQR = Q3 - Q1

    #Calc LowerRange
    lr = Q1 - (1.5 * IQR)

    #Calc Upper Range
    ur = Q3 + (1.5 * IQR)

    return lr,ur

#Outliers detection are considered only for numeric columns.ie Quantity , Unit␣
↪Price and Total Price

def outlier_treatment(drop_col = False):
    for col in data.columns[[3,5,8]]:
        lowerRange,upperRange = outlierDetection(data[col])
        if not data[(data[col] > upperRange) | (data[col] < lowerRange)].empty:
            print ("Detected  outliers for this column %r " % col)
            #hdataUpdated.drop(hdataUpdated[(hdataUpdated[col] > upperRange) |␣
↪(hdataUpdated[col] < lowerRange)].index , inplace=drop_col)
```

Cohort Analysis

Types of cohorts:

- Time Cohorts:

They are customers who signed up for a product or service during a particular time frame. Analyzing these cohorts shows the customers' behavior depending on the time they started using the company's products or services. The time may be monthly or quarterly even daily.

- Behaviour Cohorts:

They are customers who purchased a product or subscribed to a service in the past. It groups customers by the type of product or service they signed up. Customers who signed up for basic level services might have different needs than those who signed up for advanced services. Understaning the needs of the various cohorts can help a company design custom-made services or products for particular segments.

- Size Cohorts:

Size cohorts refer to the various sizes of customers who purchase company's products or services. This categorization can be based on the amount of spending in some periodic time after acquisition or the product type that the customer spent most of their order amount in some period of time.

For cohort analysis, there are a few labels that we have to create:

Invoice period - A string representation of the year and month of a single transaction/invoice. Cohort group - A string representation of the the year and month of a customer's first purchase. This label is common across all invoices for a particular customer. Cohort period/Index- A integer representation a customer's stage in its "lifetime". The number represents the number of months passed since the first purchase.

[237]:
```
data['OrderMonth'] = data['InvoiceDate'].dt.to_period('M')
data.head(2)
```

[237]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6

          InvoiceDate  UnitPrice  CustomerID         Country  Amount  \
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom   15.30
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom   20.34

  OrderMonth
0    2010-12
1    2010-12
```

[238]:
```
data['Cohort'] = data.groupby('CustomerID')['InvoiceDate'].transform('min').dt.
 ↪to_period('M')
data.head(2)
```

[238]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6

          InvoiceDate  UnitPrice  CustomerID         Country  Amount  \
```

```
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom   15.30
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom   20.34

  OrderMonth   Cohort
0    2010-12  2010-12
1    2010-12  2010-12
```

[239]:
```python
data_cohort=pd.DataFrame(data.groupby(['Cohort', 'OrderMonth']).
    agg(n_customers=('CustomerID', 'nunique')).reset_index(drop=False))
```

[240]:
```python
data_cohort.head(5)
```

[240]:
```
    Cohort OrderMonth  n_customers
0  2010-12    2010-12          948
1  2010-12    2011-01          362
2  2010-12    2011-02          317
3  2010-12    2011-03          367
4  2010-12    2011-04          341
```

[241]:
```python
#scatter plot for all clusters-todo
from operator import attrgetter
data_cohort["PeriodNumber"] = (data_cohort.OrderMonth - data_cohort.Cohort).
    apply(attrgetter('n'))
```

[242]:
```python
data_cohort
```

[242]:
```
     Cohort OrderMonth  n_customers  PeriodNumber
0   2010-12    2010-12          948             0
1   2010-12    2011-01          362             1
2   2010-12    2011-02          317             2
3   2010-12    2011-03          367             3
4   2010-12    2011-04          341             4
..      ...        ...          ...           ...
86  2011-10    2011-11           93             1
87  2011-10    2011-12           46             2
88  2011-11    2011-11          321             0
89  2011-11    2011-12           43             1
90  2011-12    2011-12           41             0

[91 rows x 4 columns]
```

[243]:
```python
cohort_pivot = data_cohort.pivot_table(index =
    "Cohort",columns="PeriodNumber",values="n_customers")
cohort_pivot
```

[243]:
```
PeriodNumber     0      1      2      3      4      5      6      7      8   \
Cohort
```

```
2010-12        948.0  362.0  317.0  367.0  341.0  376.0  360.0  336.0  336.0
2011-01        421.0  101.0  119.0  102.0  138.0  126.0  110.0  108.0  131.0
2011-02        380.0   94.0   73.0  106.0  102.0   94.0   97.0  107.0   98.0
2011-03        440.0   84.0  112.0   96.0  102.0   78.0  116.0  105.0  127.0
2011-04        299.0   68.0   66.0   63.0   62.0   71.0   69.0   78.0   25.0
2011-05        279.0   66.0   48.0   48.0   60.0   68.0   74.0   29.0    NaN
2011-06        235.0   49.0   44.0   64.0   58.0   79.0   24.0    NaN    NaN
2011-07        191.0   40.0   39.0   44.0   52.0   22.0    NaN    NaN    NaN
2011-08        167.0   42.0   42.0   42.0   23.0    NaN    NaN    NaN    NaN
2011-09        298.0   89.0   97.0   36.0    NaN    NaN    NaN    NaN    NaN
2011-10        352.0   93.0   46.0    NaN    NaN    NaN    NaN    NaN    NaN
2011-11        321.0   43.0    NaN    NaN    NaN    NaN    NaN    NaN    NaN
2011-12         41.0    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN

PeriodNumber      9      10      11      12
Cohort
2010-12        374.0  354.0  474.0  260.0
2011-01        146.0  155.0   63.0    NaN
2011-02        119.0   35.0    NaN    NaN
2011-03         39.0    NaN    NaN    NaN
2011-04          NaN    NaN    NaN    NaN
2011-05          NaN    NaN    NaN    NaN
2011-06          NaN    NaN    NaN    NaN
2011-07          NaN    NaN    NaN    NaN
2011-08          NaN    NaN    NaN    NaN
2011-09          NaN    NaN    NaN    NaN
2011-10          NaN    NaN    NaN    NaN
2011-11          NaN    NaN    NaN    NaN
2011-12          NaN    NaN    NaN    NaN
```

[244]:
```
cohort_size = cohort_pivot.iloc[:,0]
#cohort_size
```

[245]:
```
retention_matrix = cohort_pivot.divide(cohort_size, axis = 0)
retention_matrix
```

[245]:
```
PeriodNumber   0         1         2         3         4         5         6   \
Cohort
2010-12        1.0  0.381857  0.334388  0.387131  0.359705  0.396624  0.379747
2011-01        1.0  0.239905  0.282660  0.242280  0.327791  0.299287  0.261283
2011-02        1.0  0.247368  0.192105  0.278947  0.268421  0.247368  0.255263
2011-03        1.0  0.190909  0.254545  0.218182  0.231818  0.177273  0.263636
2011-04        1.0  0.227425  0.220736  0.210702  0.207358  0.237458  0.230769
2011-05        1.0  0.236559  0.172043  0.172043  0.215054  0.243728  0.265233
2011-06        1.0  0.208511  0.187234  0.272340  0.246809  0.336170  0.102128
2011-07        1.0  0.209424  0.204188  0.230366  0.272251  0.115183       NaN
2011-08        1.0  0.251497  0.251497  0.251497  0.137725       NaN       NaN
```

```
2011-09       1.0  0.298658  0.325503  0.120805       NaN       NaN       NaN
2011-10       1.0  0.264205  0.130682       NaN       NaN       NaN       NaN
2011-11       1.0  0.133956       NaN       NaN       NaN       NaN       NaN
2011-12       1.0       NaN       NaN       NaN       NaN       NaN       NaN

PeriodNumber        7         8         9        10        11        12
Cohort
2010-12      0.354430  0.354430  0.394515  0.373418  0.500000  0.274262
2011-01      0.256532  0.311164  0.346793  0.368171  0.149644       NaN
2011-02      0.281579  0.257895  0.313158  0.092105       NaN       NaN
2011-03      0.238636  0.288636  0.088636       NaN       NaN       NaN
2011-04      0.260870  0.083612       NaN       NaN       NaN       NaN
2011-05      0.103943       NaN       NaN       NaN       NaN       NaN
2011-06           NaN       NaN       NaN       NaN       NaN       NaN
2011-07           NaN       NaN       NaN       NaN       NaN       NaN
2011-08           NaN       NaN       NaN       NaN       NaN       NaN
2011-09           NaN       NaN       NaN       NaN       NaN       NaN
2011-10           NaN       NaN       NaN       NaN       NaN       NaN
2011-11           NaN       NaN       NaN       NaN       NaN       NaN
2011-12           NaN       NaN       NaN       NaN       NaN       NaN
```

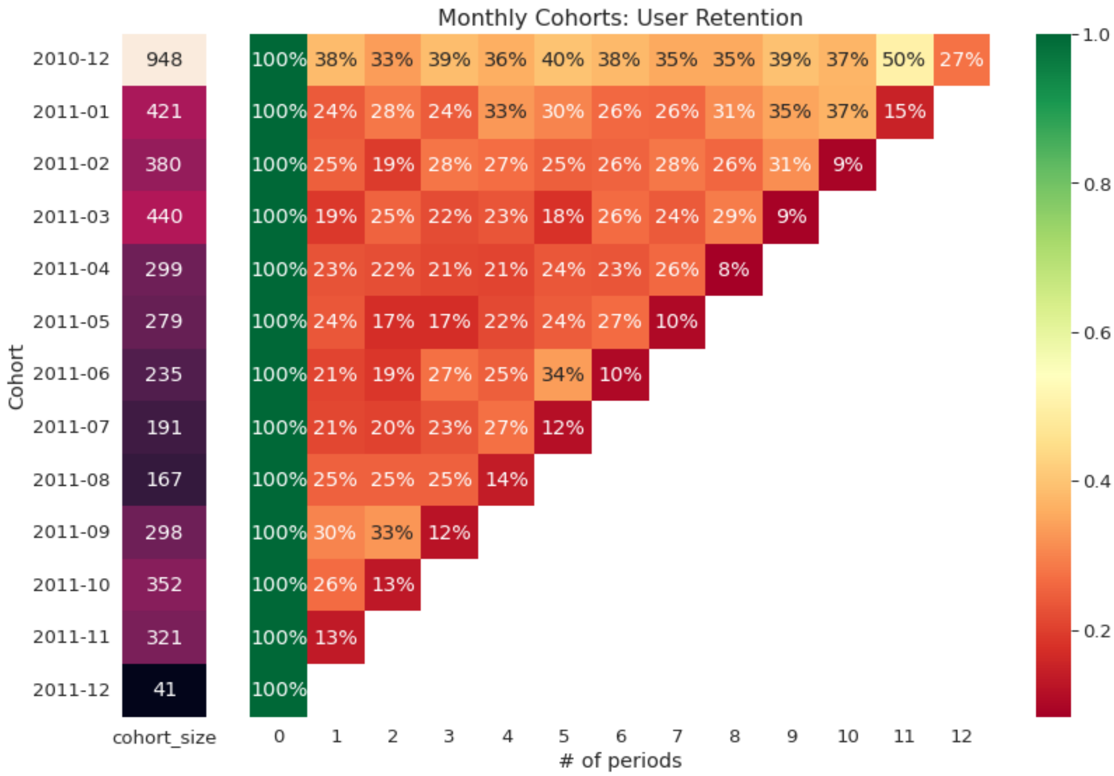
[246]:

```
with sns.axes_style("white"):
    fig, ax = plt.subplots(1, 2, figsize=(12, 8), sharey=True,␣
 ↪gridspec_kw={'width_ratios': [1, 11]})

    # retention matrix
    sns.heatmap(retention_matrix,
                mask=retention_matrix.isnull(),
                annot=True,
                fmt='.0%',
                cmap='RdYlGn',
                ax=ax[1])
    ax[1].set_title('Monthly Cohorts: User Retention', fontsize=16)
    ax[1].set(xlabel='# of periods',
              ylabel='')

    # cohort size
    cohort_size_df = pd.DataFrame(cohort_size).rename(columns={0:␣
 ↪'cohort_size'})
    #white_cmap = mcolors.ListedColormap(['white'])
    sns.heatmap(cohort_size_df,
                annot=True,
                cbar=False,
                fmt='g',
                #cmap=white_cmap,
                ax=ax[0])
```

```
    fig.tight_layout()
```



[247]:
```
import datetime as dt
data['InvoiceDate'].max()
```

[247]: Timestamp('2011-12-09 12:50:00')

[248]:
```
latestdate = dt.datetime(2011,12,10)
```

[249]:
```
rfmtable=data.groupby('CustomerID').agg({'InvoiceDate': lambda x: (latestdate - x.max()).days, 'InvoiceNo': lambda x: len(x),'UnitPrice': lambda x: x.sum()})
rfmtable=rfmtable.rename(columns={'InvoiceDate': 'recency',
                                  'InvoiceNo': 'frequency',
                                  'UnitPrice': 'monetary'})
rfmtable
```

[249]:
```
            recency  frequency  monetary
CustomerID
12346.0         325          2      2.08
12347.0           2        182    481.21
12348.0          75         31    178.71
12349.0          18         73    605.10
```

```
12350.0         310         17     65.30
…             …        …       …
18280.0         277         10     47.65
18281.0         180          7     39.36
18282.0           7         13     62.68
18283.0           3        721   1174.33
18287.0          42         70    104.55

[4372 rows x 3 columns]
```

[250]:
```
recency = data.groupby('CustomerID').agg({'InvoiceDate': lambda x: (latestdate␣
  ↪- x.max()).days})
recency.max()
```

[250]:
```
InvoiceDate    373
dtype: int64
```

[251]:
```
frequency = data.groupby("CustomerID").InvoiceNo.nunique().reset_index().
  ↪rename(columns={'InvoiceNo':'Frequency'})
frequency.max()
```

[251]:
```
CustomerID    18287.0
Frequency       248.0
dtype: float64
```

[252]:
```
monetary = data.groupby('CustomerID').Amount.sum().reset_index().
  ↪rename(columns={'Amount':'Monetary'})
monetary.max()
```

[252]:
```
CustomerID     18287.00
Monetary      279489.02
dtype: float64
```

Customer segments with RFM Model

The simplest way to create customers segments from RFM Model is to use Quantiles. We assign a score from 1 to 4 to Recency, Frequency and Monetary. Four is the best/highest value, and one is the lowest/worst value. A final RFM score is calculated simply by combining individual RFM score numbers.

[253]:
```
# RFM Quantiles
quantiles = rfmtable.quantile(q=[0.25,0.5,0.75])
quantiles
```

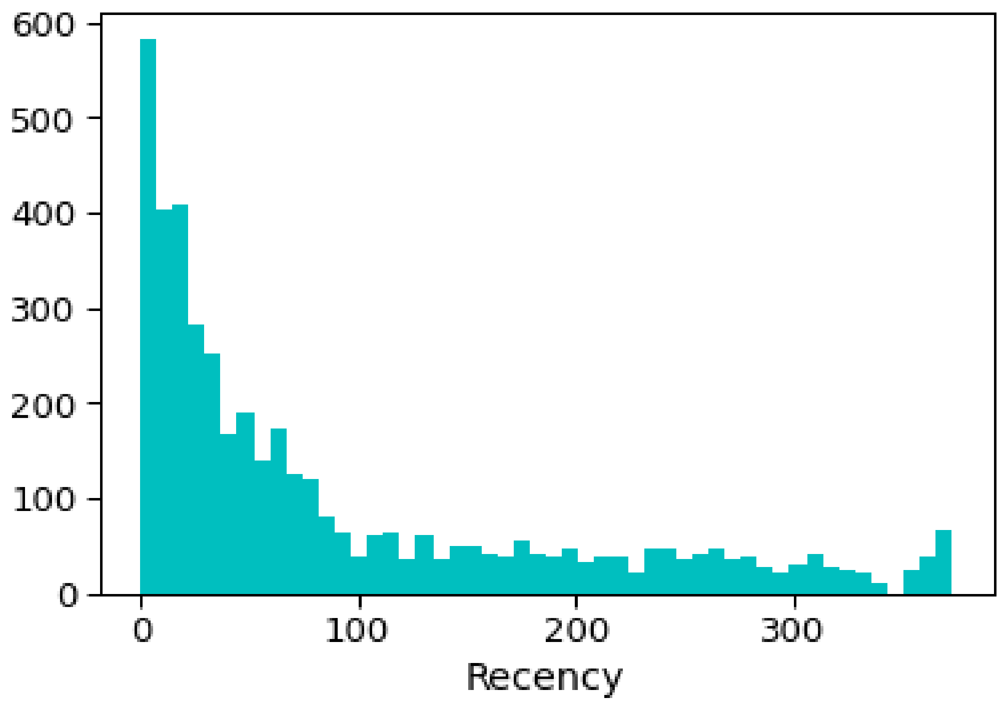
[253]:

	recency	frequency	monetary
0.25	16.0	17.00	52.7300
0.50	50.0	41.00	128.9250
0.75	143.0	99.25	299.0975

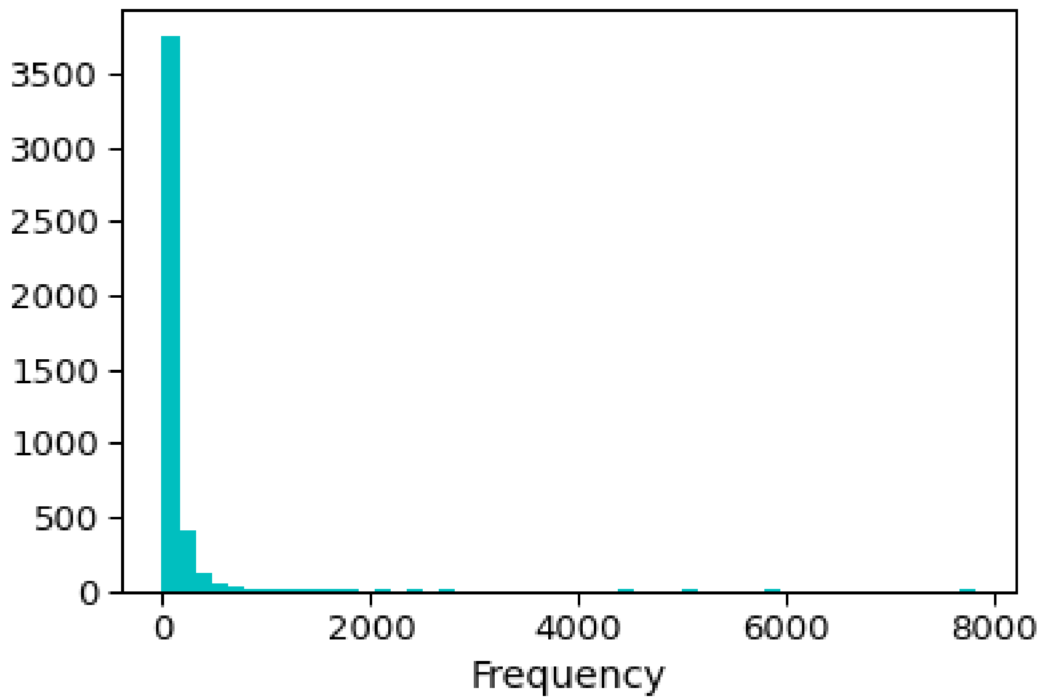
[254]:
```
# Let's convert quartile information into a dictionary so that cutoffs can be␣
↪picked up.
quantiles=quantiles.to_dict()
quantiles
```

[254]:
```
{'recency': {0.25: 16.0, 0.5: 50.0, 0.75: 143.0},
 'frequency': {0.25: 17.0, 0.5: 41.0, 0.75: 99.25},
 'monetary': {0.25: 52.730000000000004, 0.5: 128.925, 0.75: 299.0975}}
```

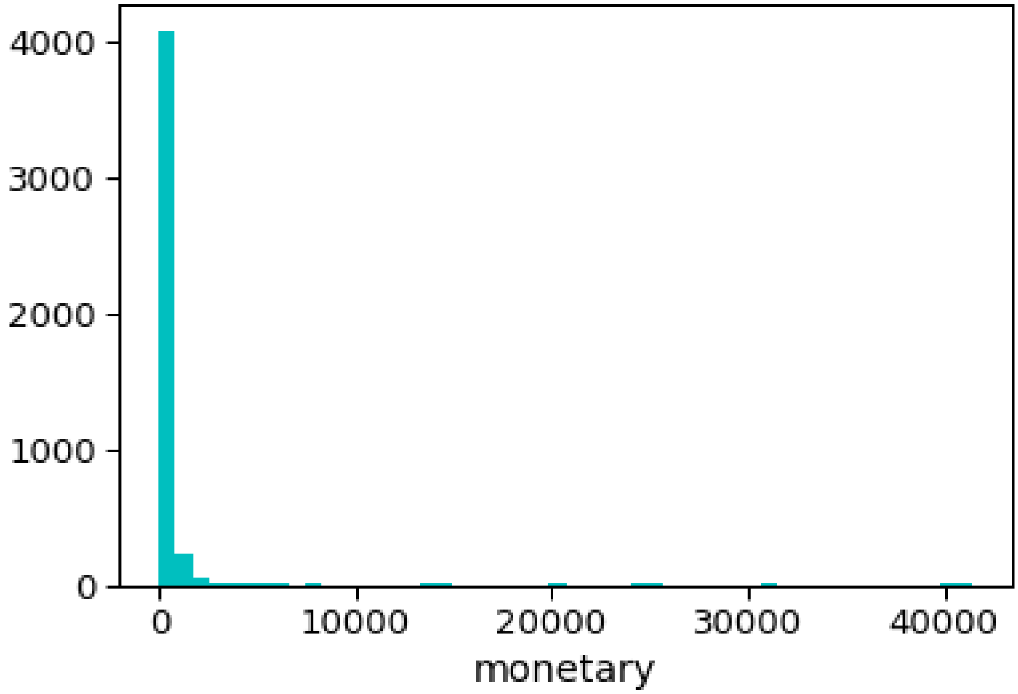
[255]:
```
#Let us visualize the histogram charts for Recency, Frequency and Monetary
plt.hist(rfmtable.recency, bins = 50, color='c')
plt.xlabel('Recency')
plt.show()
```



[256]:
```
plt.hist(rfmtable.frequency, bins = 50, color='c')
plt.xlabel('Frequency')
plt.show()
```



[257]:
```
monetary.max()
plt.hist(rfmtable.monetary, bins = 50, color='c')
plt.xlabel('monetary')
plt.show()
```



will create two segmentation classes since, high recency is bad, while high frequency and monetary value is good

[258]:
```
# 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
```

[259]:
```
# 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
```

[260]:
```
rfm_segment = rfmtable.copy()
```

[261]:
```
rfm_segment['R_Quartile'] = rfm_segment['recency'].apply(RScore,␣
↪args=('recency',quantiles,))
rfm_segment['F_Quartile'] = rfm_segment['frequency'].apply(FMScore,␣
↪args=('frequency',quantiles,))
rfm_segment['M_Quartile'] = rfm_segment['monetary'].apply(FMScore,␣
↪args=('monetary',quantiles,))
```

[262]:
```
# define segments
Segment =  ['Platinum Customers',
                     'Big Spenders',
                     'High Spend New Customers',
                     'Lowest-Spending Active Loyal Customers',
                     'Recent Customers',
                     'Good Customers Almost Lost',
                     'Churned Best Customers',
                     'Lost Cheap Customers ']
RFM = [
               ['444', '443'],
               ['114', '124', '134', '144', '214', '224', '234', '244', '314',␣
↪'324', '334', '344', '414', '424', '434', '444'],
               ['413', '314', '313', '414'],
               ['331', '341', '431', '441'],
               ['422', '423', '424', '432', '433', '434', '442', '443', '444'],
               ['244', '234', '243', '233'],
               ['144', '134', '143', '133'],
               ['122', '111', '121', '112', '221', '212', '211']
               ]
```

[263]:
```
# dictionary for each segment to map them against each customer
Description = ['Customers who bought most recently, most often and spend the␣
↪most',
               'Customers who spend the most',
               'New Customers who spend the most',
               'Active Customers who buy very often but spend less ',
               'Customers who have purchased recently',
               'Customers who were frequent and good spenders who are becoming␣
↪very inactive',
               'Customers who were frequent and good spenders who are lost␣
↪contributing to attrition',
               'Customers who purchased long ago , less frequent and very␣
↪little']
```

```python
Marketing = ['No price incentives, New products and Loyalty Programs',
                      'Market your most expensive products',
                      'Price Incentives',
                      'Promote economical cost effective products in daily use',
                      'Discounts and promote a variety of product sells',
                      'Aggressive Price Incentives',
                      'Monitor close communication with customers with constant␣
 ↪feedback and rework ',
                      'Dont spend too much time to re-acquire',
                      ]
rfm_segments = pd.DataFrame({'Segment': Segment , 'RFM' : RFM , 'Description':␣
 ↪Description, 'Marketing': Marketing})
rfm_segments
```

[263]:

```
                                  Segment  \
0                      Platinum Customers
1                             Big Spenders
2                 High Spend New Customers
3  Lowest-Spending Active Loyal Customers
4                         Recent Customers
5               Good Customers Almost Lost
6                   Churned Best Customers
7                     Lost Cheap Customers

                                                RFM  \
0                                        [444, 443]
1  [114, 124, 134, 144, 214, 224, 234, 244, 314, …
2                              [413, 314, 313, 414]
3                              [331, 341, 431, 441]
4     [422, 423, 424, 432, 433, 434, 442, 443, 444]
5                              [244, 234, 243, 233]
6                              [144, 134, 143, 133]
7                 [122, 111, 121, 112, 221, 212, 211]

                                        Description  \
0  Customers who bought most recently, most often…
1                          Customers who spend the most
2                      New Customers who spend the most
3  Active Customers who buy very often but spend …
4                  Customers who have purchased recently
5  Customers who were frequent and good spenders …
6  Customers who were frequent and good spenders …
7  Customers who purchased long ago , less freque…

                                          Marketing
0  No price incentives, New products and Loyalty …
```

```
1                 Market your most expensive products
2                                    Price Incentives
3  Promote economical cost effective products in …
4   Discounts and promote a variety of product sells
5                         Aggressive Price Incentives
6  Monitor close communication with customers wit…
7              Dont spend too much time to re-acquire
```

[264]:

```
rfm_segment.head()
```

[264]:

```
            recency  frequency  monetary  R_Quartile  F_Quartile  M_Quartile
CustomerID
12346.0         325          2      2.08           1           1           1
12347.0           2        182    481.21           4           4           4
12348.0          75         31    178.71           2           2           3
12349.0          18         73    605.10           3           3           4
12350.0         310         17     65.30           1           1           2
```

[269]:

```
rfm_segment[rfm_segment.monetary == rfm_segment.monetary.max()]
```

[269]:

```
            recency  frequency  monetary  R_Quartile  F_Quartile  M_Quartile
CustomerID
14096.0           4       5128  41376.33           4           4           4
```

For analysis it is critical to combine the scores to create a single score. There are few approaches. One approach is to just concatenate the scores to create a 3 digit number between 111 and 444. Here the drawback is too many categories (4x4x4).

[270]:

```
rfm_segment['RFMScore'] = rfm_segment.R_Quartile.map(str) \
                            + rfm_segment.F_Quartile.map(str) \
                            + rfm_segment.M_Quartile.map(str)
rfm_segment.head()
```

[270]:

```
            recency  frequency  monetary  R_Quartile  F_Quartile  M_Quartile  \
CustomerID
12346.0         325          2      2.08           1           1           1
12347.0           2        182    481.21           4           4           4
12348.0          75         31    178.71           2           2           3
12349.0          18         73    605.10           3           3           4
12350.0         310         17     65.30           1           1           2

           RFMScore
CustomerID
12346.0         111
12347.0         444
12348.0         223
12349.0         334
```

```
12350.0         112
```

[271]:
```
rfm_segment[rfm_segment.monetary == rfm_segment.monetary.max()]
```

[271]:
```
            recency  frequency  monetary  R_Quartile  F_Quartile  M_Quartile  \
CustomerID
14096.0           4       5128  41376.33           4           4           4

           RFMScore
CustomerID
14096.0         444
```

[272]:
```
# Reset the index to create a customer_ID column
rfm_segment.reset_index(inplace=True)
```

[274]:
```
rfm_segment.head()
```

[274]:
```
   CustomerID  recency  frequency  monetary  R_Quartile  F_Quartile  \
0     12346.0      325          2      2.08           1           1
1     12347.0        2        182    481.21           4           4
2     12348.0       75         31    178.71           2           2
3     12349.0       18         73    605.10           3           3
4     12350.0      310         17     65.30           1           1

   M_Quartile RFMScore
0           1      111
1           4      444
2           3      223
3           4      334
4           2      112
```

[275]:
```
import itertools
```

[276]:
```
# Highest frequency as well as monetary value with least recencycy
platinum_customers = ['444', '443']
print ("Platinum Customers                     : {}".format(platinum_customers))
```

```
Platinum Customers                     : ['444', '443']
```

[278]:
```
# Get all combinations of [1, 2, 3,4] and length 2
big_spenders_comb =  itertools.product([1, 2, 3,4],repeat = 2)
# Print the obtained combinations
big_spenders = []
for i in list(big_spenders_comb):
    item = (list(i))
    item.append(4)
    big_spenders.append( ("".join(map(str,item))))
```

```
print ("Big Spenders                          : {}".format(big_spenders))
```

```
Big Spenders                          : ['114', '124', '134', '144', '214',
'224', '234', '244', '314', '324', '334', '344', '414', '424', '434', '444']
```

[279]:
```
#High-spending New Customers - This group consists of those customers in 1-4-1␣
↪and 1-4-2.
#These are customers who transacted only once, but very recently and they spent␣
↪a lot

high_spend_new_customers = ['413', '314' ,'313','414']
print ("High Spend New Customers              : {}".
↪format(high_spend_new_customers))
```

```
High Spend New Customers              : ['413', '314', '313', '414']
```

[280]:
```
#Low spent active loyal customers
lowest_spending_active_loyal_customers_comb =  itertools.product([ 3,4], repeat␣
↪= 2)
lowest_spending_active_loyal_customers = []
for i in list(lowest_spending_active_loyal_customers_comb):
    item = (list(i))
    item.append(1)
    lowest_spending_active_loyal_customers.append( ("".join(map(str,item))))
print ("Lowest Spending Active Loyal Customers : {}".
↪format(lowest_spending_active_loyal_customers))
```

```
Lowest Spending Active Loyal Customers : ['331', '341', '431', '441']
```

[281]:
```
# recent customer
recent_customers_comb =  itertools.product([ 2,3,4], repeat = 2)
recent_customers = []
for i in list(recent_customers_comb):
    item = (list(i))
    item.insert(0,4)
    recent_customers.append( ("".join(map(str,item))))
print ("Recent Customers                      : {}".format(recent_customers))
```

```
Recent Customers                      : ['422', '423', '424', '432', '433',
'434', '442', '443', '444']
```

[283]:
```
#almost lost customers with good FM

almost_lost = ['244', '234', '243', '233']        #  Low R - Customer's␣
↪shopping less often now who used to shop a lot
print ("Good Customers Almost Lost            : {}".format(almost_lost))
```

```
Good Customers Almost Lost            : ['244', '234', '243', '233']
```

[284]:
```python
#Lost customer

churned_best_customers = ['144', '134' ,'143','133']
print ("Churned Best Customers                    : {}".
 ↪format(churned_best_customers))
```

```
Churned Best Customers                    : ['144', '134', '143', '133']
```

[285]:
```python
# Customer's shopped long ago but with less frequency and monetary value
lost_cheap_customers = ['122','111' ,'121','112','221','212' ,'211']
print ("Lost Cheap Customers                      : {}".
 ↪format(lost_cheap_customers))
```

```
Lost Cheap Customers                      : ['122', '111', '121', '112', '221',
'212', '211']
```

[289]:
```python
#dictionary for each segment to map customers against each category
segment_dict = {
    'Platinum Customers':platinum_customers,
    'Big Spenders':      big_spenders,
    'High Spend New Customers':high_spend_new_customers,
    'Lowest-Spending Active Loyal Customers' :␣
 ↪lowest_spending_active_loyal_customers ,
    'Recent Customers': recent_customers,
    'Good Customers Almost Lost':almost_lost,
    'Churned Best Customers':   churned_best_customers,
    'Lost Cheap Customers ': lost_cheap_customers,
}
```

[290]:
```python
def find_key(value):
    for k, v in segment_dict.items():
        if value in v:
            return k
rfm_segment['Segment'] = rfm_segment.RFMScore.map(find_key)
```

[291]:
```python
# Allocate all remaining customers to others segment category
rfm_segment.Segment.fillna('others', inplace=True)
rfm_segment.sample(10)
```

[291]:

	CustomerID	recency	frequency	monetary	R_Quartile	F_Quartile	\
2017	15087.0	281	15	30.54	1	1	
3126	16579.0	365	1	2.55	1	1	
3140	16597.0	4	7	11.35	4	1	
4144	17973.0	51	15	48.66	2	1	
2156	15261.0	135	19	22.79	2	2	
2913	16282.0	339	11	30.05	1	1	
4338	18239.0	218	88	252.73	1	3	

```
776     13365.0        8        132    413.51           4            4
2152    15256.0      148          6     10.90           1            1
2502    15723.0      364         38    100.75           1            2

      M_Quartile RFMScore                 Segment
2017           1      111    Lost Cheap Customers
3126           1      111    Lost Cheap Customers
3140           1      411                  others
4144           1      211    Lost Cheap Customers
2156           1      221    Lost Cheap Customers
2913           1      111    Lost Cheap Customers
4338           3      133  Churned Best Customers
776            4      444      Platinum Customers
2152           1      111    Lost Cheap Customers
2502           2      122    Lost Cheap Customers
```

[294]:

```python
# Best Customers who's recency, frequency as well as monetary attribute is␣
↪highest.
rfm_segment[rfm_segment.RFMScore=='444'].sort_values('monetary',␣
↪ascending=False).head()
```

[294]:

```
      CustomerID  recency  frequency  monetary  R_Quartile  F_Quartile  \
1300     14096.0        4       5128  41376.33           4           4
1895     14911.0        1       5898  31025.29           4           4
4042     17841.0        1       7812  19956.37           4           4
330      12748.0        0       4459  14698.31           4           4
154      12536.0        7        273  13255.22           4           4

      M_Quartile RFMScore             Segment
1300           4      444  Platinum Customers
1895           4      444  Platinum Customers
4042           4      444  Platinum Customers
330            4      444  Platinum Customers
154            4      444  Platinum Customers
```

[296]:

```python
# Biggest spenders
rfm_segment[rfm_segment.RFMScore=='334'].sort_values('monetary',␣
↪ascending=False).head()
```

[296]:

```
      CustomerID  recency  frequency  monetary  R_Quartile  F_Quartile  \
5        12352.0       36         95   2211.10           3           3
441      12909.0       39         96    642.61           3           3
3        12349.0       18         73    605.10           3           3
3509     17095.0       22         77    501.94           3           3
111      12483.0       17         81    484.21           3           3

      M_Quartile RFMScore       Segment
```

```
5               4      334  Big Spenders
441             4      334  Big Spenders
3               4      334  Big Spenders
3509            4      334  Big Spenders
111             4      334  Big Spenders
```

[297]:
```python
# customers that you must retain are those whose monetary and frequency was␣
↪high but recency reduced quite a lot recently
rfm_segment[rfm_segment.RFMScore=='244'].sort_values('monetary',␣
↪ascending=False).head()
```

[297]:
```
      CustomerID  recency  frequency  monetary  R_Quartile  F_Quartile  \
328      12744.0       51        229  25108.89           2           4
2394     15581.0      120        148   3606.00           2           4
3427     16984.0       78        411   1493.23           2           4
2586     15834.0       70        272   1241.27           2           4
2466     15674.0       73        135   1065.22           2           4

      M_Quartile RFMScore       Segment
328            4      244  Big Spenders
2394           4      244  Big Spenders
3427           4      244  Big Spenders
2586           4      244  Big Spenders
2466           4      244  Big Spenders
```

[298]:
```python
rfm_segment.to_excel('RFM Segment.xlsx')
```

[300]:
```python
rfm_segment.Segment.value_counts()
rfm_segment.recency
```

[300]:
```
0       325
1         2
2        75
3        18
4       310
       …
4367    277
4368    180
4369      7
4370      3
4371     42
Name: recency, Length: 4372, dtype: int64
```

[ ]:
```python

```

[ ]:
```python

```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```