Customer Data Project 2022

September 8, 2022

1 Customer Data Project: Segmentation

September 2022 Fatih Catpinar

The aim of the project is to review the dataset, explore it and use machine learning and visualizations to help the marketing team to segment the customers in order to better target them for email campaigns to increase sales.

The analysis will answer the questions: What type of customer the marketing team needs to target? What are the characteristics of that customer? The objective is to classify a customer so the marketing team can send the right offers to the right clients.

In this project, the following topics are studied; data exploration and cleaning, customer lifetime value and RFM (recency, frequency, monetary) segmentation and analysis, K means clustering.

Also, additional possible solutions that can be offered with the provided data to help the marketing team is discussed.

1.1 Table of contents

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.express as px
```

1.2 1. Explore and clean the data

The first step is going to be to load the data and explore. Before we do more analyses and create classification models, we need to understand the data and make sure there is no incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. We will check if the data has duplicate information, or has any missing values.

[2]:
```
df = pd.read_csv('customer data.csv')
```

The data is the sales data. The size of the dataset is about 541909 rows and 8 columns. Data columns are - InvoiceNo - StockCode - Description - Quantity - InvoiceDate - UnitPrice - CustomerID - Country

Each row repesent a sale of a specific item. Each InvoiceNo only has one CustomerID. InvoiceNo can have multiple items, it means each invoice can be consist of multiple rows. Each item in the row has a description. Invoices has date, time and location information.

[3]:
```
# the shape of the data
df.shape
```

[3]:
```
(541909, 8)
```

[4]:
```
# take a look at the first rows of the datA
df.head()
```

[4]:
```
  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  12/1/2010 8:26       2.55     17850.0  United Kingdom
1  12/1/2010 8:26       3.39     17850.0  United Kingdom
2  12/1/2010 8:26       2.75     17850.0  United Kingdom
3  12/1/2010 8:26       3.39     17850.0  United Kingdom
4  12/1/2010 8:26       3.39     17850.0  United Kingdom
```

1.2.1 Handling duplicates

There are 5268 duplicated rows. We need to drop the duplicates since they might give you an inflated results while analysing the data.

[5]:
```
# Check if there are any duplicates
df.duplicated().sum()
```

[5]:
```
5268
```

[6]:
```
# Drop the duplicates
df = df.drop_duplicates()
```

[7]:
```
df.duplicated().sum()
```

[7]:
```
0
```

[8]:
```
df.shape
```

[8]:
```
(536641, 8)
```

1.2.2 Handling missing values

Missing values can cause bias in the machine learning models and reduce the accuracy of the model. We need to handle the mssing values to be able to create an healty segmentation model.

There are 1,454 missing 'Description' which is 0.27% of the data and 135,080 missing 'CustomerID' which is 25.16% of the data. Since the analysis is based on customers, need to remove missing values from the CustomerID column. Assuming that the orders with no CustomerID were not from the customers already in the dataset.

[9]:
```
# Check if there are any missing values
df.isna().sum()
```

[9]:
```
InvoiceNo          0
StockCode          0
Description     1454
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID    135037
Country            0
dtype: int64
```

[10]:
```
# Percentage of the missing values
print(df['Description'].isna().sum() / df.shape[0] * 100)
print(df['CustomerID'].isna().sum() / df.shape[0] * 100)
```

```
0.2709446352403189
25.163377378918124
```

[11]:
```
# Drop missing CustomerID
df = df.dropna(subset=['CustomerID'])
df.shape
```

[11]:
```
(401604, 8)
```

There was no need to handle the missing 'Description'. We have the stock code for the purchased items. But, removing missing 'CustomerID' took care of the missing 'Description' issue.

[12]:
```
df.isna().sum()
```

[12]:
```
InvoiceNo      0
StockCode      0
Description    0
```

```
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

1.2.3 Organizing the dataframe

[13]:
```
# Convert the type of the 'InvoiceDate' column to datetime
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
print(df['InvoiceDate'].min())
print(df['InvoiceDate'].max())
```

```
2010-12-01 08:26:00
2011-12-09 12:50:00
```

The data is collected between 2010-12-01 and 2011-12-09.

[14]:
```
# Create a total Purchased amount column for each row
df['Total_Purchase_Amount'] = df['Quantity'] * df['UnitPrice']
df.head()
```

[14]:
```
  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

   Total_Purchase_Amount
0                  15.30
1                  20.34
2                  22.00
3                  20.34
4                  20.34
```

1.2.4 Generate descriptive statistics

[15]:
```
df.describe()
```

[15]:
```
            Quantity      UnitPrice     CustomerID  Total_Purchase_Amount
count  401604.000000  401604.000000  401604.000000          401604.000000
mean       12.183273       3.474064   15281.160818              20.613638
std       250.283037      69.764035    1714.006089             430.352218
min    -80995.000000       0.000000   12346.000000         -168469.600000
25%         2.000000       1.250000   13939.000000               4.250000
50%         5.000000       1.950000   15145.000000              11.700000
75%        12.000000       3.750000   16784.000000              19.800000
max     80995.000000   38970.000000   18287.000000          168469.600000
```

The minimum value for 'Quantity' is negative which is not possible. Need to explore the reason for the negative values. If there is a special meaning that help us extract more inforation, we will use it. Otherevise we will remove the negative quantity data.

[16]:
```
df[df['Quantity'] < 0].head()
```

[16]:
```
    InvoiceNo StockCode                       Description  Quantity  \
141   C536379         D                          Discount        -1
154   C536383    35004C   SET OF 3 COLOURED  FLYING DUCKS        -1
235   C536391     22556    PLASTERS IN TIN CIRCUS PARADE        -12
236   C536391     21984  PACK OF 12 PINK PAISLEY TISSUES        -24
237   C536391     21983  PACK OF 12 BLUE PAISLEY TISSUES        -24

            InvoiceDate  UnitPrice  CustomerID         Country  \
141 2010-12-01 09:41:00      27.50     14527.0  United Kingdom
154 2010-12-01 09:49:00       4.65     15311.0  United Kingdom
235 2010-12-01 10:24:00       1.65     17548.0  United Kingdom
236 2010-12-01 10:24:00       0.29     17548.0  United Kingdom
237 2010-12-01 10:24:00       0.29     17548.0  United Kingdom

     Total_Purchase_Amount
141                 -27.50
154                  -4.65
235                 -19.80
236                  -6.96
237                  -6.96
```

[17]:
```
# df['Quantity'].sort_values()
```

[18]:
```
df[df['Quantity'] < 0].shape
```

[18]:
```
(8872, 9)
```

[19]:
```
(df[df['Quantity'] < 0]['InvoiceNo'].str[0] == "C").sum()
```

[19]: 8872

[20]:
```python
# Does all Invoice numbers with negative quantity start with "C" ?
(df[df['Quantity'] < 0]['InvoiceNo'].str[0] == "C").sum()
```

[20]: 8872

All of the negative 'Quantity' has "InvoiceNo" start with letter 'C'. It seems it means the invoice is cancelled. It is good to have the cancelled transections in the data, so that more analys can be made. What is the most canceled item? What is the percentage of the cancelation? Who canceled most? If the canceled orders are investigated more, there is a chance to prevent future cancellations. But, for the customer value segment study, the canceled invoices is going to be removed.

[21]:
```python
# Remove rows with negative 'Quantity'.
df = df[df['Quantity'] > 0]
```

[22]:
```python
df[df['Quantity'] < 0].shape
```

[22]: (0, 9)

[23]:
```python
(df['InvoiceNo'].str[0] == "C").sum()
```

[23]: 0

1.2.5 Check "StockCode" and "Description" features

[24]:
```python
# Check number of unique features.
df.nunique()
```

[24]:
```
InvoiceNo               18536
StockCode                3665
Description              3877
Quantity                  302
InvoiceDate             17286
UnitPrice                 441
CustomerID               4339
Country                    37
Total_Purchase_Amount    2940
dtype: int64
```

The "StockCode" and "Description" don't have same unique count. It might be because there are some StockCode with multiple Description.

[25]:
```python
df.groupby('StockCode')['Description'].unique()
```

[25]:
```
StockCode
10002     [INFLATABLE POLITICAL GLOBE ]
10080        [GROOVY CACTUS INFLATABLE]
```

```
10120                      [DOGGY RUBBER]
10123C           [HEARTS WRAPPING TAPE ]
10124A     [SPOTS ON RED BOOKCOVER TAPE]
                       …
C2                             [CARRIAGE]
DOT                      [DOTCOM POSTAGE]
M                                [Manual]
PADS        [PADS TO MATCH ALL CUSHIONS]
POST                            [POSTAGE]
Name: Description, Length: 3665, dtype: object
```

[26]:
```
# for example stock code '23081' has two descriptions
df.groupby('StockCode')['Description'].unique()['23081']
```

[26]:
```
array(['GREEN METAL BOX ARMY SUPPLIES', 'GREEN METAL BOX TOP SECRET'],
      dtype=object)
```

1.2.6 Check unique customer and item count.

[27]:
```
# Check number of unique features.
df.nunique()
```

[27]:
```
InvoiceNo                18536
StockCode                 3665
Description               3877
Quantity                   302
InvoiceDate              17286
UnitPrice                  441
CustomerID                4339
Country                     37
Total_Purchase_Amount     2940
dtype: int64
```

There are 4339 unique customers and there are 3665 unique items.

1.2.7 Explore the location feature

There are 37 countries. Investigate the following information to understand the country affect: - number of invoice for each country - number of transactions for each country - total purchase amount for each country - number of customers for each country - number of sold units for each country.

[28]:
```
# Number of invoice by 'Country'
df.groupby('Country')["InvoiceNo"].nunique().sort_values(ascending=False)[:10]
```

[28]:
```
Country
United Kingdom    16649
```

```
Germany             457
France              389
EIRE                260
Belgium              98
Netherlands          95
Spain                90
Australia            57
Portugal             57
Switzerland          51
Name: InvoiceNo, dtype: int64
```

[29]:
```python
# Number of transactions by 'Country'
df['Country'].value_counts()[:10]
```

[29]:
```
United Kingdom    349227
Germany             9027
France              8327
EIRE                7228
Spain               2480
Netherlands         2363
Belgium             2031
Switzerland         1842
Portugal            1453
Australia           1184
Name: Country, dtype: int64
```

[30]:
```python
# Total purchase amount by 'Country'
df.groupby('Country')["Total_Purchase_Amount"].sum().
 ↪sort_values(ascending=False)[:10]
```

[30]:
```
Country
United Kingdom    7.285025e+06
Netherlands       2.854463e+05
EIRE              2.652625e+05
Germany           2.286784e+05
France            2.089343e+05
Australia         1.384538e+05
Spain             6.155856e+04
Switzerland       5.644395e+04
Belgium           4.119634e+04
Sweden            3.836783e+04
Name: Total_Purchase_Amount, dtype: float64
```

[31]:
```python
# Number of customers by 'Country'
df.groupby('Country')["CustomerID"].nunique().sort_values(ascending=False)[:10]
```

[31]:
```
Country
United Kingdom    3921
Germany             94
France              87
Spain               30
Belgium             25
Switzerland         21
Portugal            19
Italy               14
Finland             12
Austria             11
Name: CustomerID, dtype: int64
```

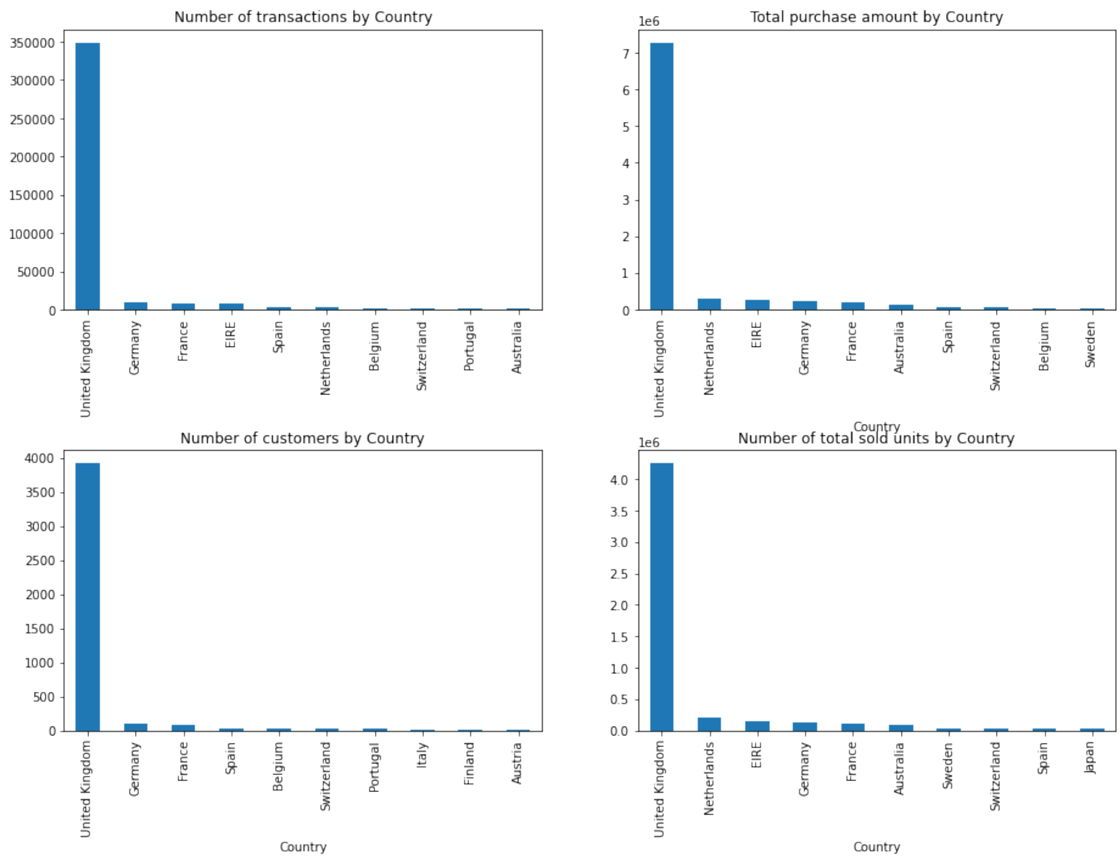
[32]:
```python
# Number of total sold units by 'Country'
df.groupby('Country')["Quantity"].sum().sort_values(ascending=False)[:10]
```

[32]:
```
Country
United Kingdom    4254037
Netherlands        200937
EIRE               140383
Germany            119156
France             111429
Australia           84199
Sweden              36078
Switzerland         30083
Spain               27944
Japan               26016
Name: Quantity, dtype: int64
```

1.2.8 Plot the country factor in the data

[33]:
```python
fig = plt.figure(figsize = (15, 10))

plt.subplot(2,2,1)
df['Country'].value_counts()[:10].plot(kind="bar", title='Number of␣
↪transactions by Country')

plt.subplot(2,2,2)
df.groupby('Country')["Total_Purchase_Amount"].sum().
↪sort_values(ascending=False)[:10].plot(
    kind="bar", title='Total purchase amount by Country')

plt.subplot(2,2,3)
df.groupby('Country')["CustomerID"].nunique().sort_values(ascending=False)[:10].
↪plot(
    kind="bar", title='Number of customers by Country')
```

```
plt.subplot(2,2,4)
df.groupby('Country')["Quantity"].sum().sort_values(ascending=False)[:10].
↪sort_values(ascending=False)[:10].plot(
    kind="bar", title='Number of total sold units by Country')

plt.xticks(rotation=90)

# set the spacing between subplots
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=0.9,
                    wspace=0.2,
                    hspace=0.5)

plt.show()
```



As seen in the bar plots, United Kingdom the most number of transactions, most total sales revenue, most customers and most total sold units. The most of the data is from United Kingdom.

We have an unbalanced data by country. To make sure that the culture bias create variation, we will focus on one country at a time. Since the biggegest sale is from UK, we should focus on the biggest country.

For the rest of the analysis, United Kingdom data is going to be used.

[34]:
```
df_UK = df[df['Country'] == 'United Kingdom']
df_UK.head()
```

[34]:
```
  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

   Total_Purchase_Amount
0                  15.30
1                  20.34
2                  22.00
3                  20.34
4                  20.34
```

[35]:
```
df_UK.shape
```

[35]:
```
(349227, 9)
```

1.2.9 What is the most sold item in UK?

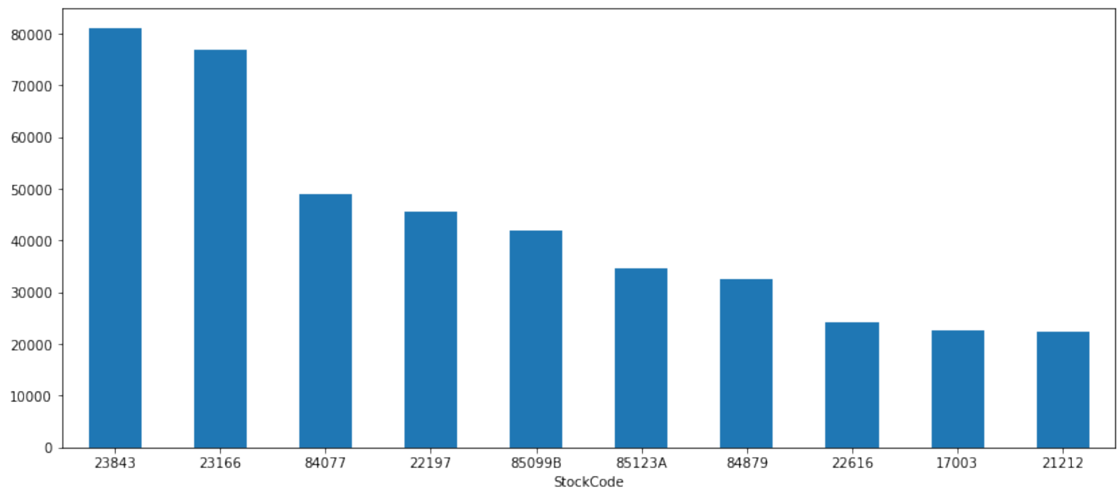
[36]:
```
# Most sold items by quantity
df_UK.groupby(["StockCode", "Description"])[["Quantity"]].sum().
↪sort_values(by="Quantity", ascending=False).head()
```

[36]:
```
                                              Quantity
StockCode Description
23843     PAPER CRAFT , LITTLE BIRDIE            80995
23166     MEDIUM CERAMIC TOP STORAGE JAR         76919
84077     WORLD WAR 2 GLIDERS ASSTD DESIGNS      49086
85099B    JUMBO BAG RED RETROSPOT                41878
85123A    WHITE HANGING HEART T-LIGHT HOLDER     34630
```

[37]:
```
df_UK['Quantity'].sort_values(ascending=False).head()
```

```
[37]: 540421    80995
      61619     74215
      502122    12540
      421632     4800
      206121     4300
      Name: Quantity, dtype: int64
```

```python
[38]: df_UK.groupby("StockCode")['Quantity'].sum().sort_values(ascending=False)[:10].
      ↪plot(kind="bar", figsize=(14, 6));
      plt.xticks(rotation=0);
```



1.2.10 What is the most frequent purchased item in UK?

```python
[39]: # Most frequent purchased items
      df_UK["StockCode"].value_counts().head(10)
```

```
[39]: 85123A    1936
      85099B    1461
      22423     1417
      84879     1320
      47566     1301
      20725     1135
      20727     1022
      22720     1013
      23203      993
      22383      977
      Name: StockCode, dtype: int64
```

```python
[40]: df_val_counts = pd.DataFrame(df_UK["StockCode"].value_counts())
      df_value_counts_reset = df_val_counts.reset_index()
```

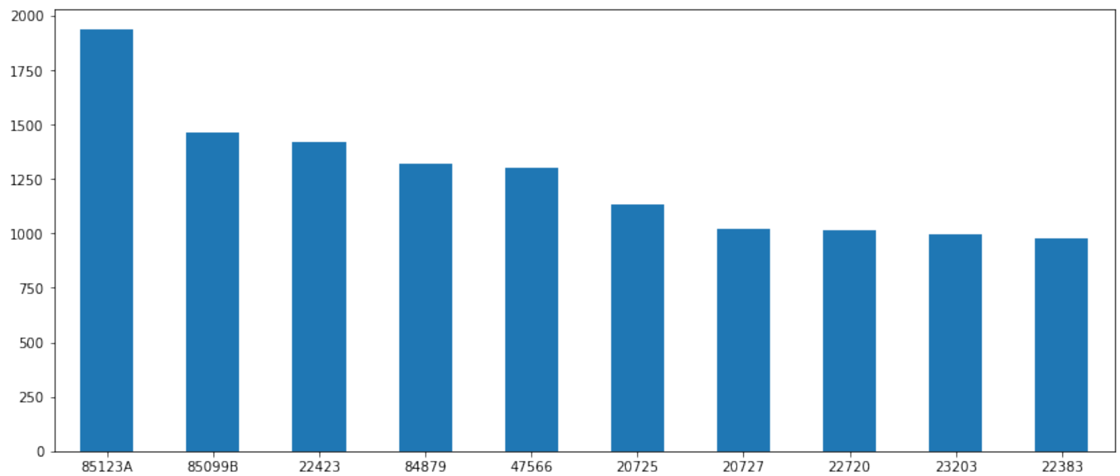
```
df_value_counts_reset.columns = ['StockCode', 'counts'] # change column names

pd.merge(df_value_counts_reset, df_UK[['StockCode', 'Description']],␣
 ↪how='left', on='StockCode').drop_duplicates().head()
```

[40]:

	StockCode	counts	Description
0	85123A	1936	WHITE HANGING HEART T-LIGHT HOLDER
1926	85123A	1936	CREAM HANGING HEART T-LIGHT HOLDER
1936	85099B	1461	JUMBO BAG RED RETROSPOT
3397	22423	1417	REGENCY CAKESTAND 3 TIER
4814	84879	1320	ASSORTED COLOUR BIRD ORNAMENT

[41]:

```
df_UK["StockCode"].value_counts().head(10).plot(kind="bar", figsize=(14, 6))
plt.xticks(rotation=0);
```



1.2.11 Plot the montly sales in UK

[42]:

```
df_UK['Month_Year'] = df_UK["InvoiceDate"].apply(lambda x: x.strftime("%B %Y"))
df_UK.head()
```

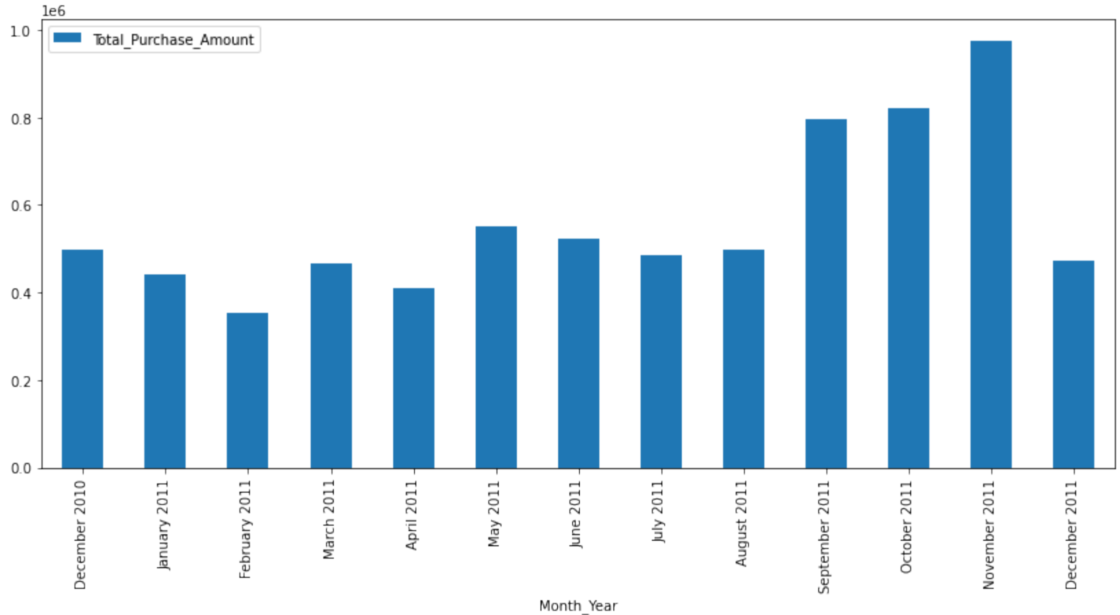
```
<ipython-input-42-543a6a1aad6a>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_UK['Month_Year'] = df_UK["InvoiceDate"].apply(lambda x: x.strftime("%B
%Y"))
```

[42]:
```
  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

   Total_Purchase_Amount     Month_Year
0                  15.30  December 2010
1                  20.34  December 2010
2                  22.00  December 2010
3                  20.34  December 2010
4                  20.34  December 2010
```

[43]:
```
df_UK_monthly = df_UK.groupby('Month_Year').agg({'Total_Purchase_Amount' :␣
↪'sum',
                                  'Quantity' : 'sum',
                                  'InvoiceDate': 'min'}).reset_index().
↪sort_values(by='InvoiceDate')
df_UK_monthly
```

[43]:
```
        Month_Year  Total_Purchase_Amount  Quantity         InvoiceDate
2    December 2010             496477.340    266577 2010-12-01 08:26:00
5     January 2011             440876.330    277699 2011-01-04 10:00:00
4    February 2011             354618.200    212808 2011-02-01 08:23:00
8       March 2011             465784.190    275426 2011-03-01 08:30:00
0       April 2011             408733.111    259594 2011-04-01 08:22:00
9         May 2011             550359.350    301117 2011-05-01 10:51:00
7        June 2011             523775.590    280321 2011-06-01 07:37:00
6        July 2011             484545.591    301553 2011-07-01 08:16:00
1      August 2011             497194.910    310102 2011-08-01 08:30:00
12  September 2011             794806.692    453422 2011-09-01 08:25:00
11    October 2011             821220.130    474675 2011-10-02 10:32:00
10   November 2011             975251.390    580772 2011-11-01 08:16:00
3    December 2011             471381.820    259971 2011-12-01 08:33:00
```

[44]:
```
ax = df_UK_monthly.plot.bar(x='Month_Year', y='Total_Purchase_Amount',␣
↪figsize=(14, 6), rot=90)
```



1.3 2. Customer lifetime value and RFM (recency, frequency, monetary) segmentation

After we explore and clean the data, we can now segment the customers in order to better target them for email campaigns to increase sales.

Customer Lifetime Value indicates the total revenue from the customer during the entire relationship. Customer Lifetime Value helps companies to focus on those potential customers who can bring in the more revenue in the future.

To understand the best customers, most profitable customer, and the lost customers, we will create recency, frequency, monetary column for each customer. In our case; - Number of Days from last purchase of the customer is recency - Number of Invoice of specific customer is frequency - Custumer Lifetime total Purchase value is monetary

[45]:
```
df_UK.head()
```

[45]:
```
  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

   Total_Purchase_Amount     Month_Year
0                  15.30  December 2010
1                  20.34  December 2010
2                  22.00  December 2010
3                  20.34  December 2010
4                  20.34  December 2010
```

[46]:
```
# df['InvoiceDate'].max()--> today
```

[47]:
```
# grouping customers

df_UK_customer = df_UK.groupby('CustomerID').agg(
                         {'InvoiceDate':lambda date:(df_UK['InvoiceDate'].
 ↪max()-date.max()).days+1,
                          'InvoiceNo':'nunique',
                          'Total_Purchase_Amount':'sum'})

df_UK_customer = df_UK_customer.reset_index()
df_UK_customer.columns = ['CustomerID',
                          'Number of Days from last purchase',
                          'Number of Invoice',
                          'Custumer Lifetime Purchase']
```

[48]:
```
# df_UK.groupby('CustomerID')['InvoiceDate'].agg(lambda date:
 ↪(df_UK['InvoiceDate']
#                                                              .max()-date.
 ↪max()).days+1)
```

[49]:
```
df_UK_customer.head()
```

[49]:
```
   CustomerID  Number of Days from last purchase  Number of Invoice  \
0     12346.0                                326                  1
1     12747.0                                  2                 11
2     12748.0                                  1                210
3     12749.0                                  4                  5
4     12820.0                                  3                  4

   Custumer Lifetime Purchase
0                    77183.60
1                     4196.01
2                    33053.19
3                     4090.88
4                      942.34
```

[50]:
```
df_UK_customer.columns = ['CustomerID','recency','frequency','monetary']
df_UK_customer.head()
```

[50]:

	CustomerID	recency	frequency	monetary
0	12346.0	326	1	77183.60
1	12747.0	2	11	4196.01
2	12748.0	1	210	33053.19
3	12749.0	4	5	4090.88
4	12820.0	3	4	942.34

1.3.1 RFM Generate descriptive statistics

[51]:
```
df_UK_customer.describe()
```

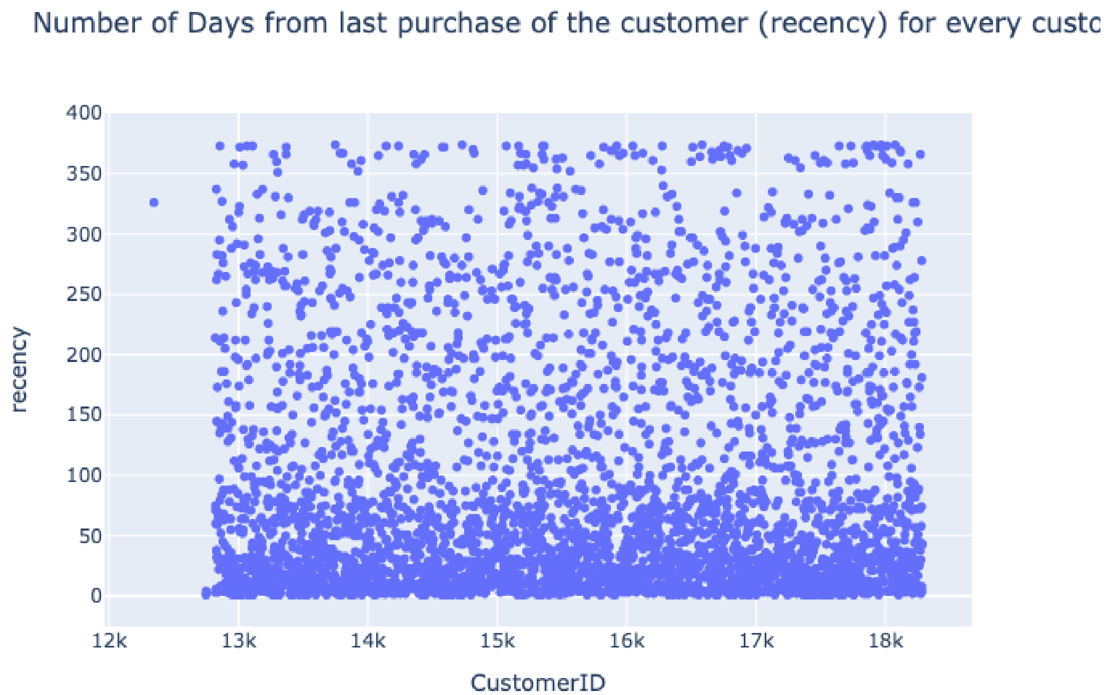
[51]:

	CustomerID	recency	frequency	monetary
count	3921.000000	3921.000000	3921.000000	3921.000000
mean	15561.471563	92.188472	4.246111	1857.950687
std	1576.823683	99.528995	7.205750	7477.736186
min	12346.000000	1.000000	1.000000	0.000000
25%	14208.000000	18.000000	1.000000	298.110000
50%	15569.000000	51.000000	2.000000	644.300000
75%	16913.000000	143.000000	5.000000	1570.810000
max	18287.000000	374.000000	210.000000	259657.300000

1.3.2 Plot recency

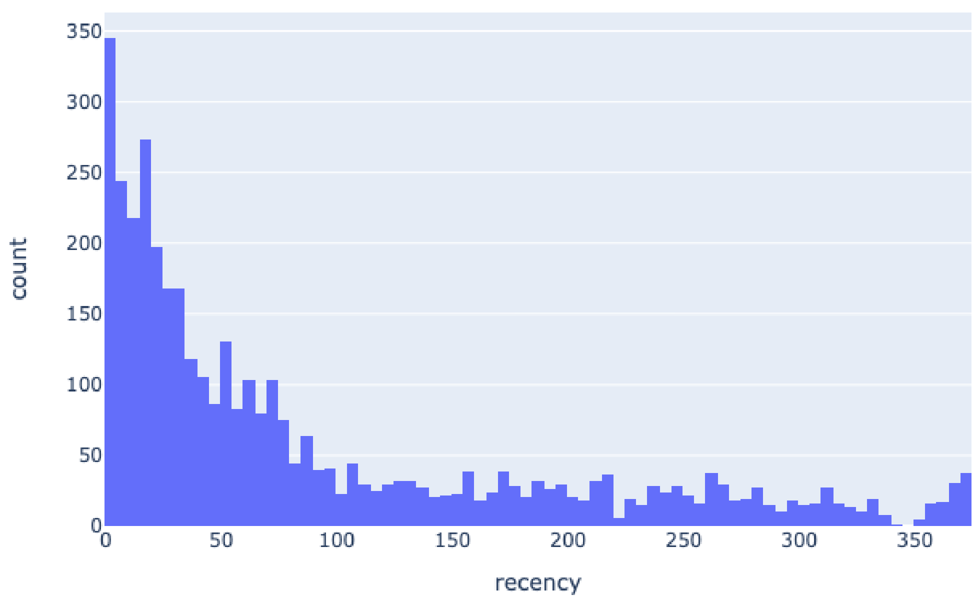
The following plot shows Number of Days from last purchase of the customer which is called recency for every customer

[52]:
```
fig = px.scatter(df_UK_customer, x="CustomerID", y="recency",

           title= "Number of Days from last purchase of the customer (recency)␣
↪for every customer")

fig.show("png")
```



[53]:
```
fig = px.histogram(df_UK_customer, x="recency", nbins=100,
               title= "The histogram of recency")

fig.show("png")
```

The histogram of recency



The above histogram shows that most of the customer last purchase date is 100 and lower.

[54]:
```
# The customers that made a purchase most recently
df_UK_customer.sort_values(by=['recency'], ascending=True).head()
```

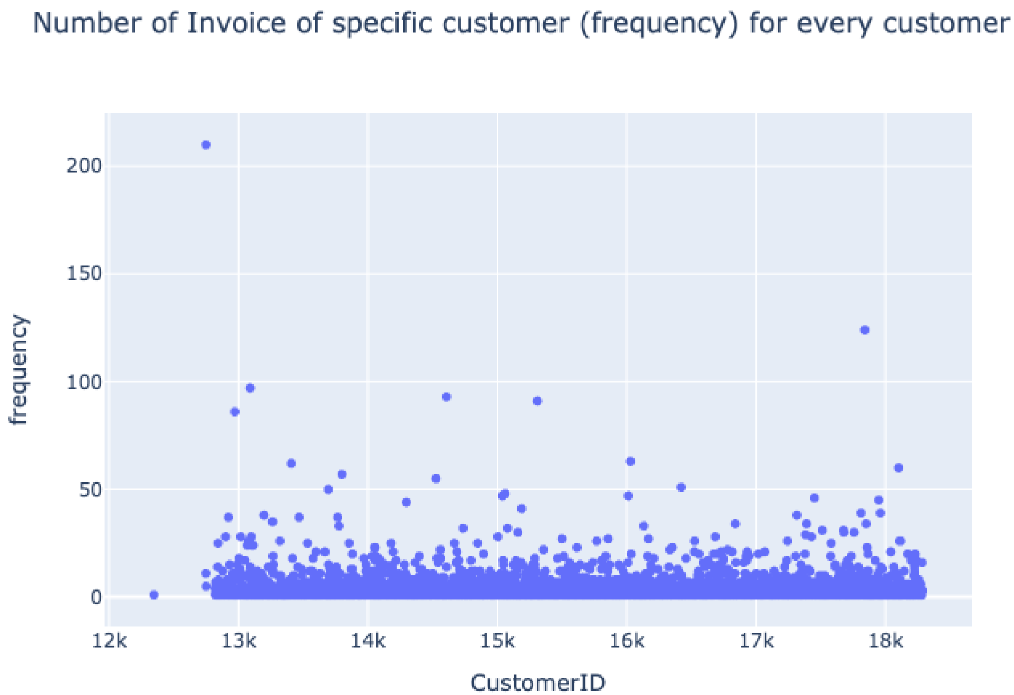
[54]:

	CustomerID	recency	frequency	monetary
1795	15344.0	1	3	563.94
3373	17528.0	1	8	3628.50
2727	16626.0	1	17	4413.10
759	13890.0	1	10	1883.81
2957	16933.0	1	2	563.23

1.3.3 Plot frequency

The following plot shows Number of Invoice of specific customer is frequency for every customer

[55]:
```
fig = px.scatter(df_UK_customer, x="CustomerID", y="frequency",
           title= "Number of Invoice of specific customer (frequency) for every␣
↪customer")

fig.show("png")
```



[56]:
```
# The customers that purchased most frequetly
df_UK_customer.sort_values(by=['frequency'], ascending=False).head()
```

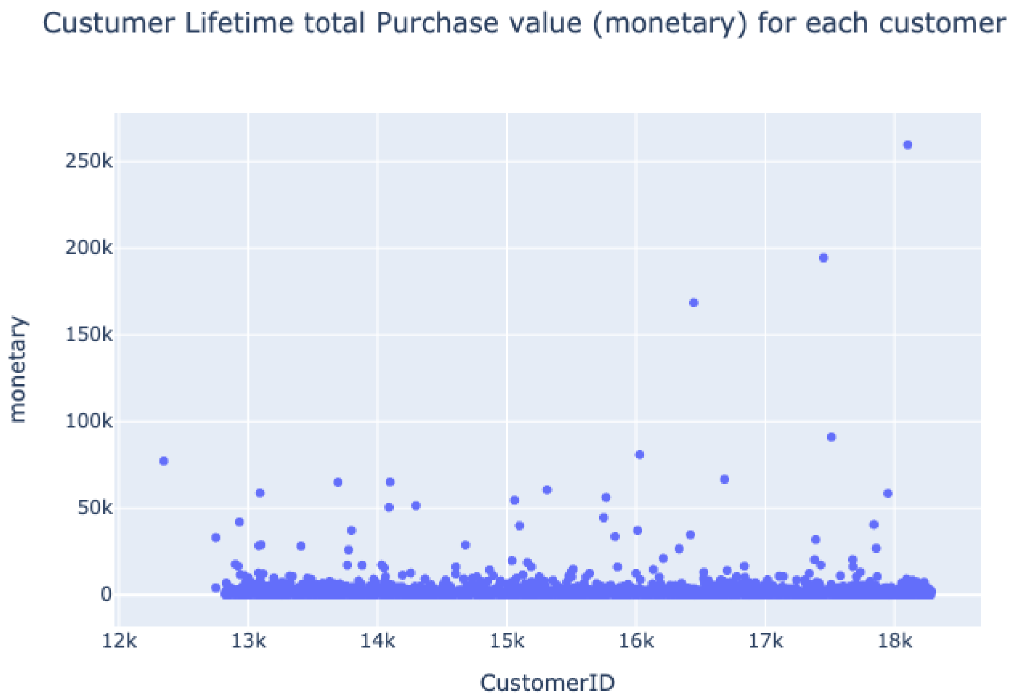
[56]:

	CustomerID	recency	frequency	monetary
2	12748.0	1	210	33053.19
3594	17841.0	2	124	40519.84
191	13089.0	3	97	58762.08
1268	14606.0	1	93	12076.15
1772	15311.0	1	91	60632.75

1.3.4 Plot monetary

The following plot shows the Custumer Lifetime total Purchase value is monetary

[57]:
```
fig = px.scatter(df_UK_customer, x="CustomerID", y="monetary",
          title="Custumer Lifetime total Purchase value (monetary) for each␣
 ↪customer")

fig.show("png")
```



[58]:
```
# The customers that spend the most money
df_UK_customer['monetary'].sort_values(ascending=False).head()
```

[58]:
```
3784    259657.30
3315    194390.79
2599    168472.50
3357     91062.38
2295     80850.84
Name: monetary, dtype: float64
```

1.3.5 RFM Model

We can create customer segments from an RFM model by using Quartiles. We will assign a score from 1 to 4 to each category (Recency, Frequency, and Monetary). 1 is the lowest score and 4 is best score.

[59]:
```
tresholds = df_UK_customer.quantile(q = [0.25, 0.50, 0.75])
tresholds
```

[59]:

	CustomerID	recency	frequency	monetary
0.25	14208.0	18.0	1.0	298.11
0.50	15569.0	51.0	2.0	644.30

```
0.75     16913.0    143.0        5.0   1570.81
```

[60]:
```
def recency_scoring(df_UK_customer):
    if df_UK_customer['recency'] <= tresholds['recency'].iloc[0]:
        score = 4
    elif df_UK_customer['recency'] <= tresholds['recency'].iloc[1]:
        score = 3
    elif df_UK_customer['recency'] <= tresholds['recency'].iloc[2]:
        score = 2
    else:
        score = 1
    return score

def frequency_scoring(df_UK_customer):
    if df_UK_customer['frequency'] <= tresholds['frequency'].iloc[0]:
        score = 1
    elif df_UK_customer['frequency'] <= tresholds['frequency'].iloc[1]:
        score = 2
    elif df_UK_customer['frequency'] <= tresholds['frequency'].iloc[2]:
        score = 3
    else:
        score = 4
    return score

def monetary_scoring(df_UK_customer):
    if df_UK_customer['monetary'] <= tresholds['monetary'].iloc[0]:
        score = 1
    elif df_UK_customer['monetary'] <= tresholds['monetary'].iloc[1]:
        score = 2
    elif df_UK_customer['monetary'] <= tresholds['monetary'].iloc[2]:
        score = 3
    else:
        score = 4
    return score
```

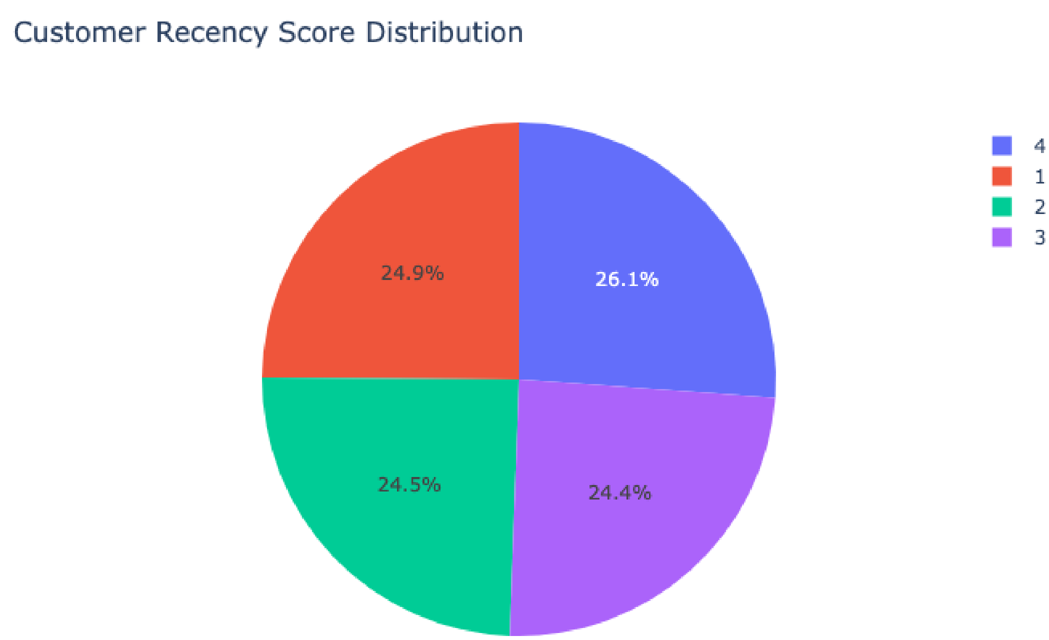
[61]:
```
df_UK_customer['recency_score'] = df_UK_customer.apply(recency_scoring, axis=1)
df_UK_customer['frequency_score'] = df_UK_customer.apply(frequency_scoring,␣
 ↪axis=1)
df_UK_customer['monetary_score'] = df_UK_customer.apply(monetary_scoring,␣
 ↪axis=1)
df_UK_customer.head()
```

[61]:

	CustomerID	recency	frequency	monetary	recency_score	frequency_score	\
0	12346.0	326	1	77183.60	1	1	
1	12747.0	2	11	4196.01	4	4	
2	12748.0	1	210	33053.19	4	4	
3	12749.0	4	5	4090.88	4	3	

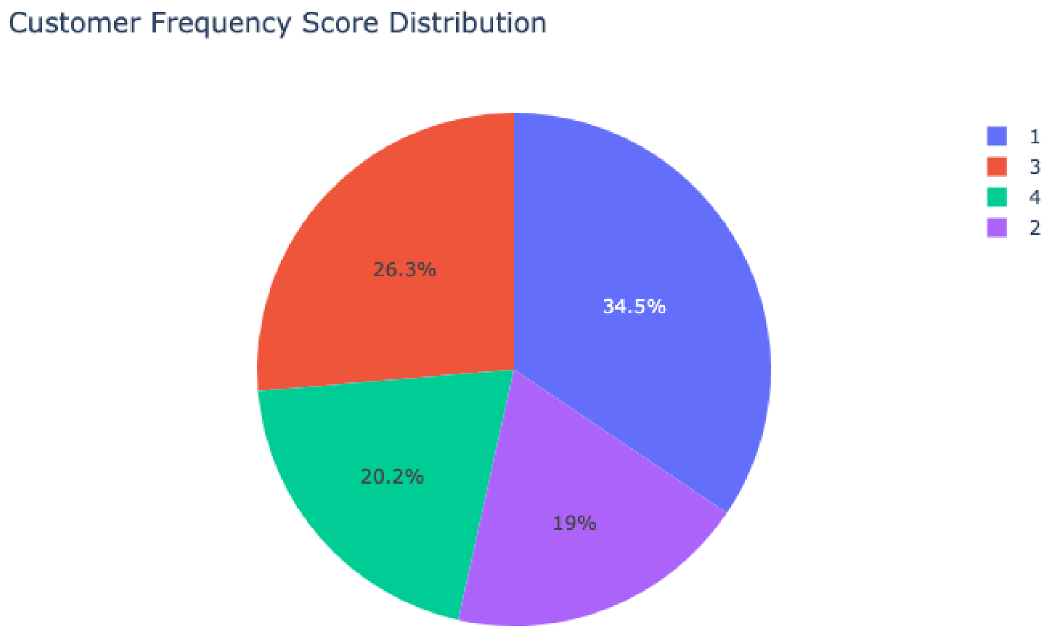
```
4      12820.0         3            4     942.34               4                 3

   monetary_score
0               4
1               4
2               4
3               4
4               3
```

[62]:
```
fig = px.pie(values = df_UK_customer['recency_score'].value_counts(),
             names = (df_UK_customer["recency_score"].value_counts()).index,
             title = 'Customer Recency Score Distribution')
fig.show("png")
```



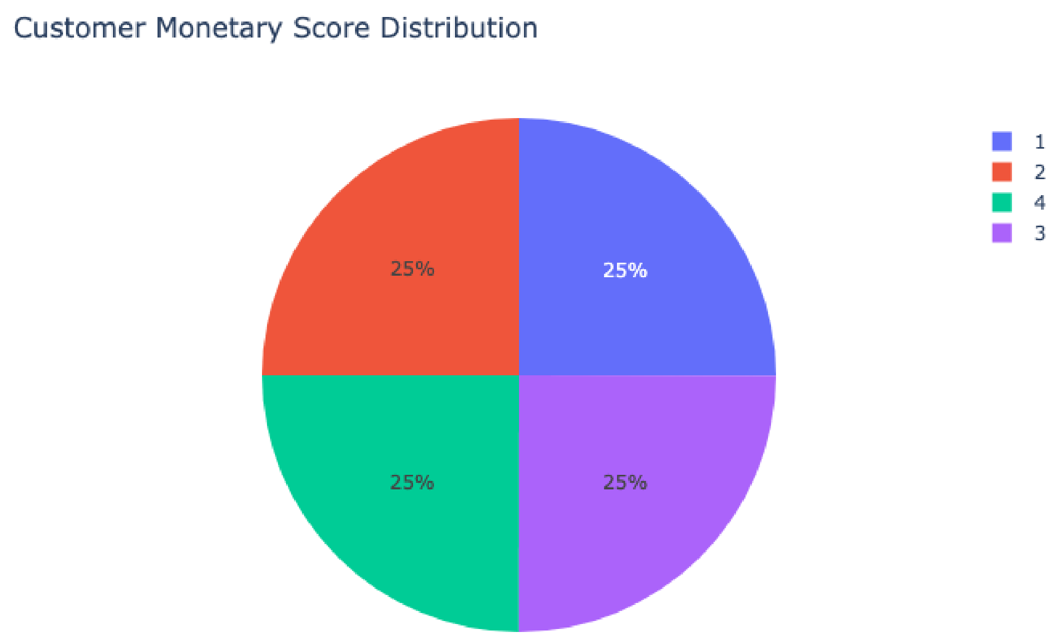
As seen from the pie chart above; after the quartile scoring the distribution of the customers are almost same for the recency. It means the number of customer whose last purchase is - less than 18 days - between 18 and 51 days - between 51 and 143 days - greater than 143 days are very close.

[63]:
```
fig = px.pie(values = df_UK_customer['frequency_score'].value_counts(),
             names = (df_UK_customer['frequency_score'].value_counts()).index,
             title = 'Customer Frequency Score Distribution')
fig.show("png")
```



You are looking at the customer frequency score distribution pie chart above. - The customers who only buy 1 time: %34.5. - The customers who made a purchase 2 times: %19. - The customers who made a purchase betweeen 2 and 5 times: %26.3. - The customers who made a purchase more than 5 times: %20.2.

[64]:
```
fig = px.pie(values = df_UK_customer['monetary_score'].value_counts(),
             names = (df_UK_customer['monetary_score'].value_counts()).index,
             title = 'Customer Monetary Score Distribution')
fig.show("png")
```



You are looking at the customer monately score distribution pie chart above.

1. The customers who spend 298.11 and less
2. The customers who spend between 298.11 and 644.30
3. The customers who spend between 644.30 and 1570.81
4. The customers who spend 1570.81 and up

1.3.6 Combine the RFM Scores

We can concat the recency, frequency, monetary scores; so that we willl have one combined score to segment the costumers

[65]:
```
df_UK_customer['RFM_Score'] = (df_UK_customer['recency_score'].astype(str) +
                               df_UK_customer['frequency_score'].astype(str) +
                               df_UK_customer['monetary_score'].astype(str))
df_UK_customer.head()
```

[65]:
```
   CustomerID  recency  frequency  monetary  recency_score  frequency_score  \
0     12346.0      326          1  77183.60              1                1
1     12747.0        2         11   4196.01              4                4
2     12748.0        1        210  33053.19              4                4
3     12749.0        4          5   4090.88              4                3
4     12820.0        3          4    942.34              4                3
```

```
   monetary_score RFM_Score
0               4       114
1               4       444
2               4       444
3               4       434
4               3       433
```

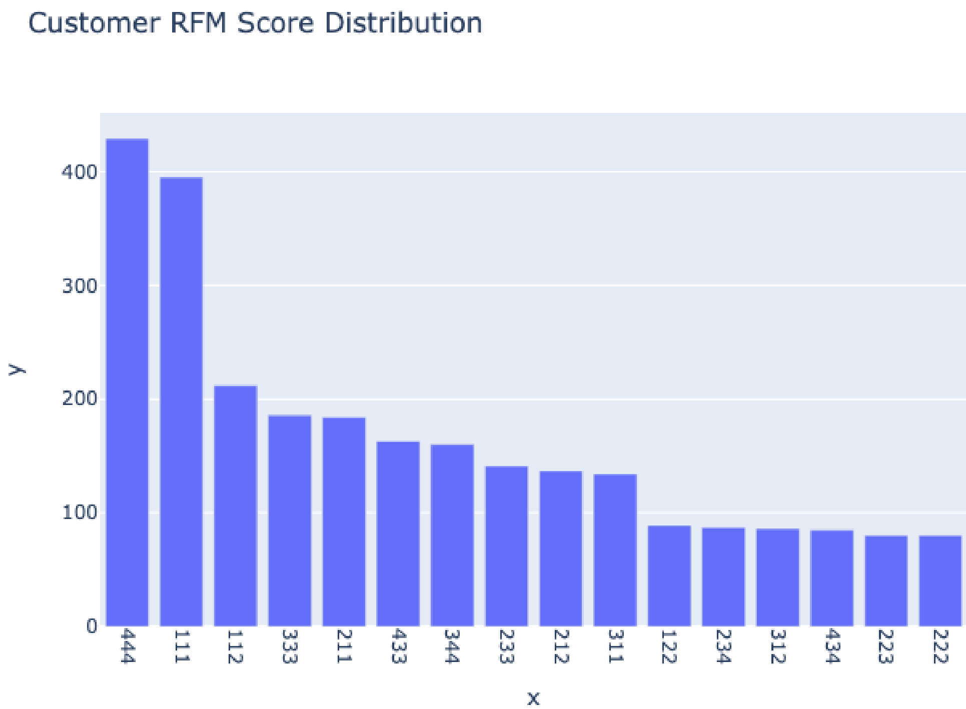
We can define the best customer as RFM_Score is 444. Since 4 is the best score for each category.

[66]:
```
# Best customers
df_UK_customer[df_UK_customer['RFM_Score']=='444'].sort_values('monetary',␣
↪ascending=False).head()
```

[66]:
```
      CustomerID  recency  frequency   monetary  recency_score  \
3784     18102.0        1         60  259657.30              4
3315     17450.0        8         46  194390.79              4
3357     17511.0        3         31   91062.38              4
2767     16684.0        4         28   66653.56              4
903      14096.0        4         17   65164.79              4

      frequency_score  monetary_score RFM_Score
3784                4               4       444
3315                4               4       444
3357                4               4       444
2767                4               4       444
903                 4               4       444
```

1.3.7 Distrubution of the customers based on RFM Scores

[67]:
```
fig = px.bar(df_UK_customer,
             x = df_UK_customer['RFM_Score'].value_counts()[:16].index,
             y = df_UK_customer['RFM_Score'].value_counts()[:16].values,
             title = 'Customer RFM Score Distribution')
fig.update_xaxes(tickangle=90)
fig.show("png")
```



We can categorize the customers based on their RFM score. - Champion = if frequent buyer and makes large purchase. - Top Loyal Customer = if recent buyer and makes large purchase. - Loyal Customer = if recent buyer (3) - Top Recent Customer = if recent buyer (4) and makes large purchase. - Recent Customer = if recent buyer (4) - Top Customer Needed Attention = if makes large purchase but recency is 2,3 - Customer Needed Attention = recency is 2,3 - Top Lost Customer = recency is 1, and makes large purchase. - Lost Customer = recency is 1, and makes small purchase.

[68]:
```
def categorizer(rfm):
    if (rfm[0] in ['2', '3', '4']) & (rfm[1] in ['4']) & (rfm[2] in ['4']):
        rfm = 'Champion'
    elif (rfm[0] in ['3']) & (rfm[1] in ['1', '2', '3', '4']) & (rfm[2] in␣
 ↪['3', '4']):
        rfm = 'Top Loyal Customer'
    elif (rfm[0] in ['3']) & (rfm[1] in ['1', '2', '3', '4']) & (rfm[2] in␣
 ↪['1', '2']):
        rfm = 'Loyal Customer'
    elif (rfm[0] in ['4']) & (rfm[1] in ['1', '2', '3', '4']) & (rfm[2] in␣
 ↪['3', '4']):
        rfm = 'Top Recent Customer'
    elif (rfm[0] in ['4']) & (rfm[1] in ['1', '2', '3', '4']) & (rfm[2] in␣
 ↪['1', '2']):
```

```
        rfm = 'Recent Customer'
    elif (rfm[0] in ['2', '3']) & (rfm[1] in ['1', '2', '3', '4']) & (rfm[2] in␣
↪['3', '4']):
        rfm = 'Top Customer Needed Attention'
    elif (rfm[0] in ['2', '3']) & (rfm[1] in ['1', '2', '3', '4']) & (rfm[2] in␣
↪['1', '2']):
        rfm = 'Customer Needed Attention'
    elif (rfm[0] in ['1']) & (rfm[1] in ['1', '2', '3', '4']) & (rfm[2] in␣
↪['3', '4']):
        rfm = 'Top Lost Customer'
    elif (rfm[0] in ['1']) & (rfm[1] in ['1', '2', '3', '4']) & (rfm[2] in␣
↪['1', '2']):
        rfm = 'Lost Customer'
    return rfm
```

[69]:
```
df_UK_customer['Customer_Category'] = df_UK_customer["RFM_Score"].
↪apply(categorizer)
df_UK_customer.head()
```

[69]:
```
   CustomerID  recency  frequency  monetary  recency_score  frequency_score  \
0     12346.0      326          1  77183.60              1                1
1     12747.0        2         11   4196.01              4                4
2     12748.0        1        210  33053.19              4                4
3     12749.0        4          5   4090.88              4                3
4     12820.0        3          4    942.34              4                3

   monetary_score RFM_Score    Customer_Category
0               4       114    Top Lost Customer
1               4       444             Champion
2               4       444             Champion
3               4       434  Top Recent Customer
4               3       433  Top Recent Customer
```

[70]:
```
df_UK_customer['Customer_Category'].value_counts(dropna=False)
```

[70]:
```
Lost Customer                   803
Champion                        653
Customer Needed Attention       507
Loyal Customer                  408
Top Customer Needed Attention   391
Top Loyal Customer              390
Top Recent Customer             353
Recent Customer                 243
Top Lost Customer               173
Name: Customer_Category, dtype: int64
```

[71]:
```
fig = px.histogram(df_UK_customer,
                   x = df_UK_customer['Customer_Category'].value_counts().index,
                   y = df_UK_customer['Customer_Category'].value_counts().
 ↪values,
                   title = 'Customer Category Distribution',
                   labels = dict(x = "Customer_Category", y ="counts"))

fig.update_xaxes(tickangle=-90)
fig.show()
```

1.4 3. Customer clustering with K means

In this section, we will segment the customers into different classes.

Possible algorithms are Logistic Regression, K-means Clustering, and K-nearest Neighbor. We don't have labels. We need to use an Unsupervised classification. We want to use a simple, cost effective algoritm. K-Means is very easy and simple to implement. It is highly scalable, can be applied to both small and large datasets.

In order to find the best number of clusters that best represents the data, we will use elbow methd and the silhouette score.

[72]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import PowerTransformer
from sklearn.cluster import KMeans
from yellowbrick.cluster import SilhouetteVisualizer
```

Standardize the data Since kmeans use distance-based measurements to determine the similarity between data, it is a good idea to's standardize the data to have a mean of zero and a standard deviation of one.

[73]:
```
df_RFM = df_UK_customer[['recency','frequency','monetary']]
df_RFM.head()
```

[73]:

	recency	frequency	monetary
0	326	1	77183.60
1	2	11	4196.01
2	1	210	33053.19
3	4	5	4090.88
4	3	4	942.34

[74]:
```
X = np.array(df_RFM)
X
```

[74]:
```
array([[3.260000e+02, 1.000000e+00, 7.718360e+04],
       [2.000000e+00, 1.100000e+01, 4.196010e+03],
       [1.000000e+00, 2.100000e+02, 3.305319e+04],
```

```
       …,
       [8.000000e+00, 2.000000e+00, 1.780500e+02],
       [4.000000e+00, 1.600000e+01, 2.045530e+03],
       [4.300000e+01, 3.000000e+00, 1.837280e+03]])
```

[75]:
```python
xmu = np.mean(X, axis = 0)
print("Mu")
print(xmu)

xsd = np.std(X, axis = 0, ddof =1)
print("\nSigma")
print(xsd)
```

```
Mu
[  92.18847233    4.24611069 1857.95068707]

Sigma
[9.95289950e+01 7.20574996e+00 7.47773619e+03]
```

[76]:
```python
xstdz = (X - xmu)/xsd
print("The Standardized X Matrix:\n")
print(xstdz)
```

```
The Standardized X Matrix:

[[ 2.34918003e+00 -4.50488944e-01  1.00733227e+01]
 [-9.06152748e-01  9.37291657e-01  3.12669404e-01]
 [-9.16200071e-01  2.85541256e+01  4.17174911e+00]
 …
 [-8.45868808e-01 -3.11710884e-01 -2.24653645e-01]
 [-8.86058101e-01  1.63118196e+00  2.50850402e-02]
 [-4.94212489e-01 -1.72932824e-01 -2.76429745e-03]]
```

[77]:
```python
# scaler = StandardScaler()
# scaler.fit(df_RFM)
# X = scaler.transform(df_RFM)
X = xstdz
```

[78]:
```python
df_RFM_scaled = pd.DataFrame(X, columns = ['recency', 'frequency', 'monetary'])
df_RFM_scaled.head()
```

[78]:

	recency	frequency	monetary
0	2.349180	-0.450489	10.073323
1	-0.906153	0.937292	0.312669
2	-0.916200	28.554126	4.171749
3	-0.886058	0.104623	0.298610
4	-0.896105	-0.034155	-0.122445

1.4.1 Check Data Skewness

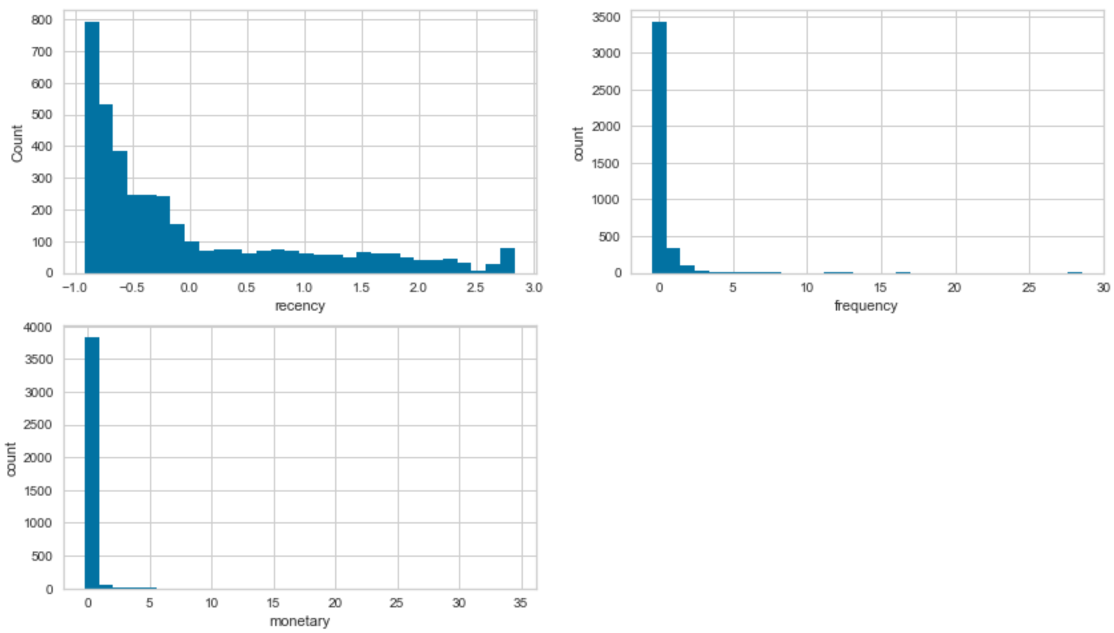
One of the key kmeans assumptions is symmetric distribution of variables. Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right.

The data is skewed as seen from the following plots. We need to handle it before applying the machine learning model.

[79]:
```
df_RFM_scaled = pd.DataFrame(X)
df_RFM_scaled.skew()
```

[79]:
```
0     1.245606
1    10.806240
2    20.219866
dtype: float64
```

[80]:
```
plt.figure(figsize=(14, 8))

plt.subplot(2,2,1)
plt.hist(X[:,0], density=False, bins=30)  # density=False would make counts
plt.ylabel('Count')
plt.xlabel('recency');

plt.subplot(2,2,2)
plt.hist(X[:,1], density=False, bins=30)  # density=False would make counts
plt.ylabel('count')
plt.xlabel('frequency');

plt.subplot(2,2,3)
plt.hist(X[:,2], density=False, bins=30)  # density=False would make counts
plt.ylabel('count')
plt.xlabel('monetary');
plt.show()
```



1.4.2 Handle Skewness

[81]:
```python
# Apply a power transform featurewise to make data more Gaussian-like.
pt = PowerTransformer(method='yeo-johnson')
trans = pt.fit_transform(df_RFM_scaled)
rfm_trans = pd.DataFrame(trans)
rfm_trans.head()
```

[81]:

	0	1	2
0	1.690945	-1.075770	2.204545
1	-1.323019	1.663917	1.786827
2	-1.351571	2.222539	2.202953
3	-1.266510	0.908723	1.768308
4	-1.294666	0.612450	0.114364

[82]:
```python
rfm_trans.skew()
```

[82]:
```
0    0.333068
1    0.444732
2    0.661158
dtype: float64
```

1.4.3 Find the number of clusters (k) for the k-means model

We will try two methods to define the best k.

The first one is the elbow method. Elbow method is a way to find out the best value of k. For a ramage of k values, it calculates the sum of the square of the points and calculates the average distance. Finally, we will plot a graph between k-values and the sum of the square to get the k value. At some point, our graph will decrease abruptly. That point will be considered as a value of k.

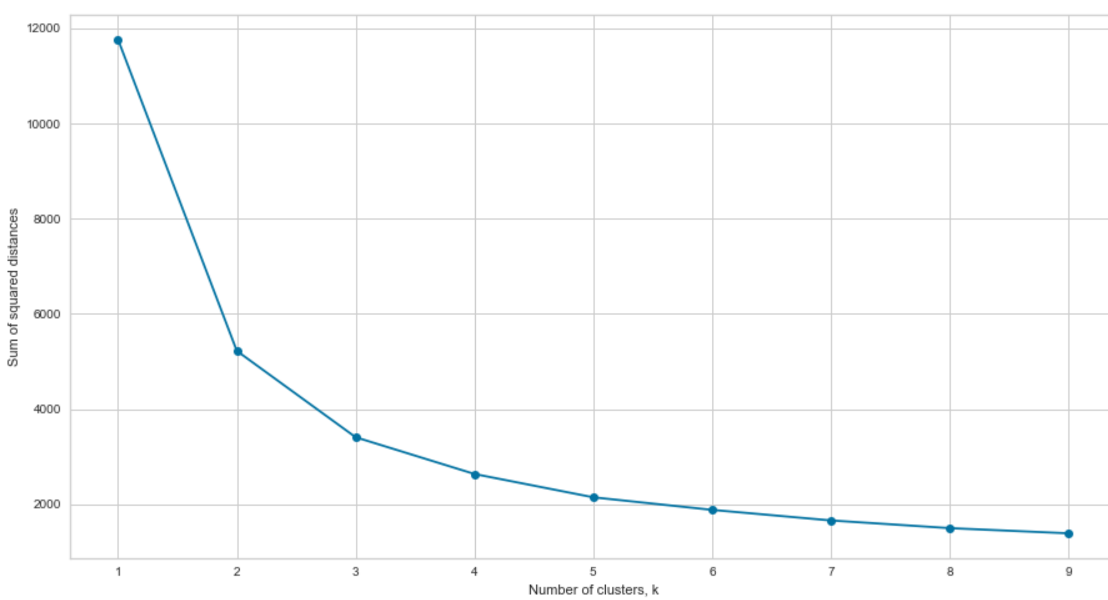
The second method is Silhouette score. The Silhouette score is used to measure the degree of separation between clusters. The value of Silhouette score varies from -1 to 1. If the score is 1, the cluster is dense and well-separated than other clusters.

Elbow Method to find best k

[83]:
```
X = np.array(rfm_trans)
```

Using the elbow method to determine the optimal number of clusters for k-means clustering

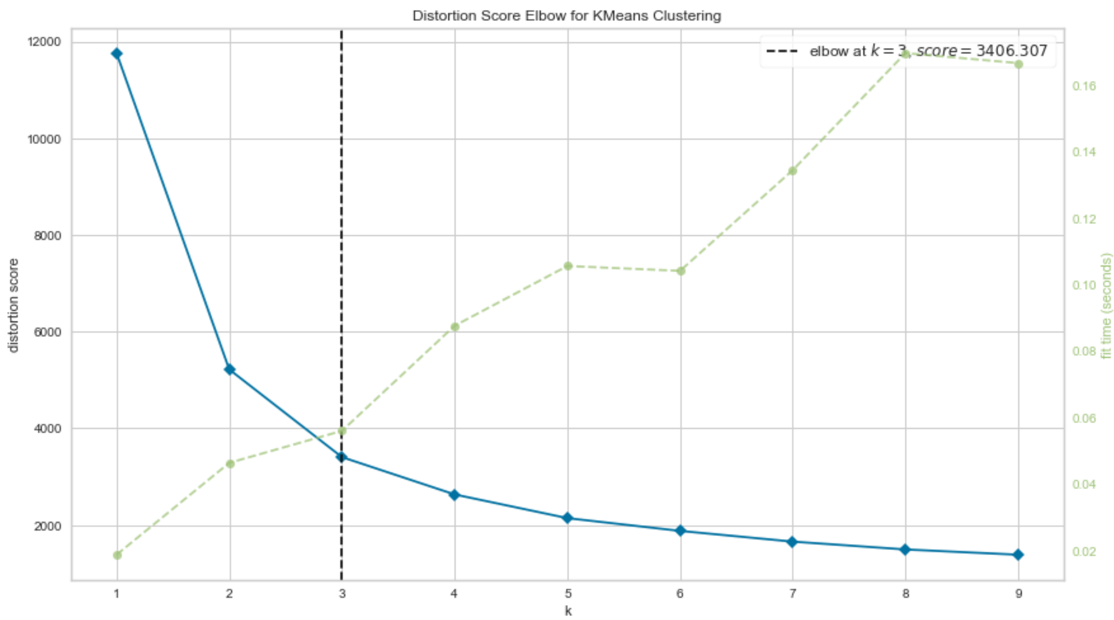
[84]:
```
# Elbow Method
k_range = range(1, 10)
sum_squared_distances = []
for k in k_range:
    model = KMeans(n_clusters=k, random_state=42)
    model.fit(X)
    sum_squared_distances.append(model.inertia_)

# Plot ks vs inertias
f, ax = plt.subplots(figsize=(15, 8))
plt.plot(k_range, sum_squared_distances, '-o')
plt.xlabel('Number of clusters, k')
plt.ylabel('Sum of squared distances')
plt.xticks(k_range)
plt.show()
```



From the above we can see that at k=3 the plot decrease abruptly. We can select k as 3 or 4.

We can also use KElbowVisualizer package to see the best k value on plot below.

[85]:
```
from yellowbrick.cluster import KElbowVisualizer

model = KMeans()
visualizer = KElbowVisualizer(model, k=(1, 10))

plt.figure(figsize=(14, 8))
visualizer.fit(X)        # Fit the data to the visualizer
visualizer.show()
```



[85]:
```
<AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'},
xlabel='k', ylabel='distortion score'>
```

Silhouette score method to find the best k

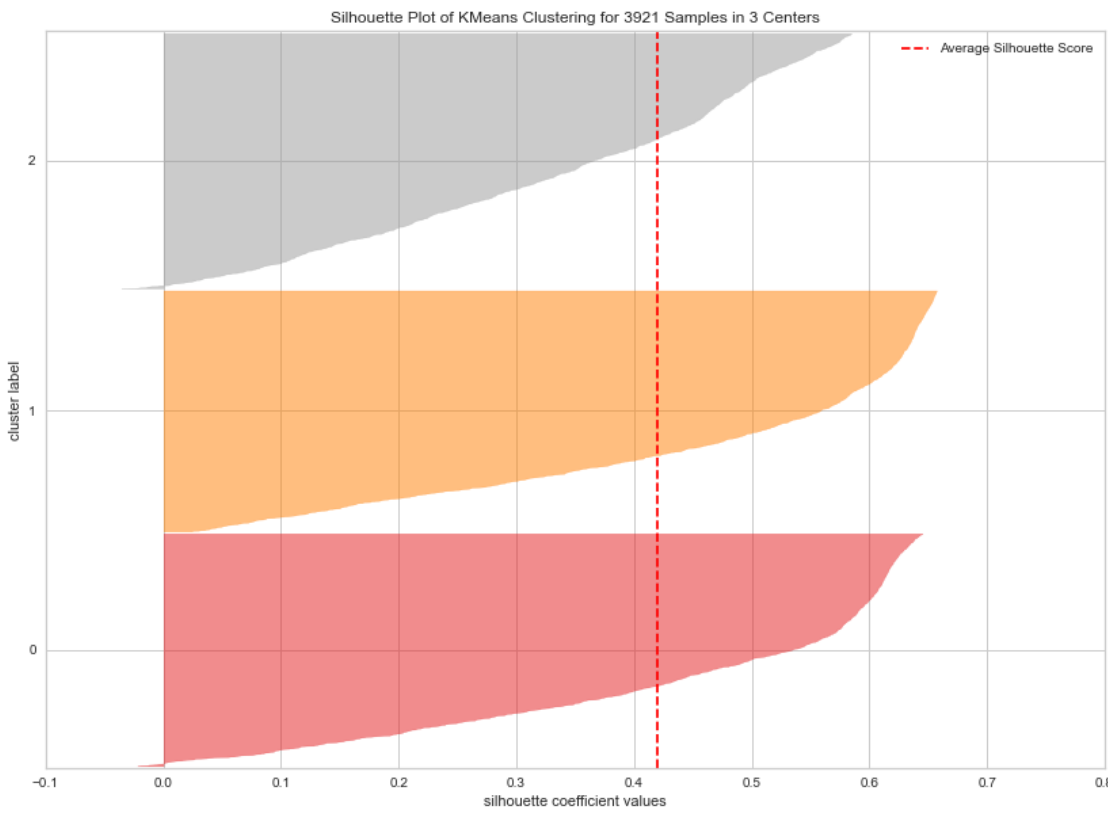
[86]:
```
for k in range(2, 10):
    model = KMeans(n_clusters=k)
    model.fit(X)
    sum_squared_distances.append(model.inertia_)
    print("Number of cluster: " + str(k) + "--> silhouette_score: " +␣
 ↪str(silhouette_score(xstdz, model.labels_)))
```

```
Number of cluster: 2--> silhouette_score: 0.26639905964990745
Number of cluster: 3--> silhouette_score: 0.2842971972762908
```

```
Number of cluster: 4--> silhouette_score: 0.17835035080925069
Number of cluster: 5--> silhouette_score: 0.14173107669260424
Number of cluster: 6--> silhouette_score: 0.12093018732409684
Number of cluster: 7--> silhouette_score: 0.09798918615882206
Number of cluster: 8--> silhouette_score: 0.11870178892835129
Number of cluster: 9--> silhouette_score: 0.09383809871770772
```

As seen above the maximum silhouette_score is when the cluster is 3. So we pick the best number of cluster as 3.

[87]:
```
# visualize silhouette when model with k = 3
model = KMeans(n_clusters=3)
visualizer = SilhouetteVisualizer(model)
plt.figure(figsize=(14, 10))

visualizer.fit(X)     # Fit the data to the visualizer
visualizer.poof();
```



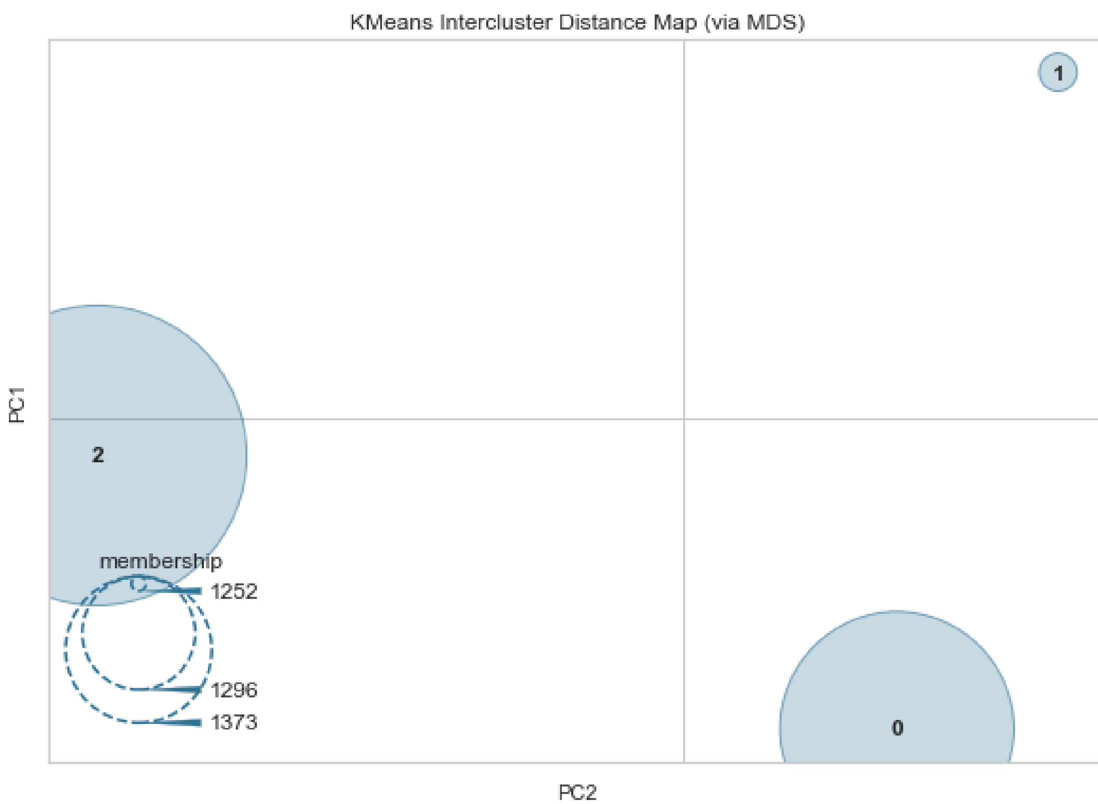
The best cluster number is 3. And we will use k =3. We can still use more clusters if we want to segment the data in more classes. But for this project I will use 3 classes.

1.5 K-means Model

[88]:
```
# visualize the K-means model
```

[89]:
```
from yellowbrick.cluster import InterclusterDistance
plt.rcParams["figure.figsize"] = (10, 7)

# Instantiate the clustering model and visualizer
model = KMeans(3)
visualizer = InterclusterDistance(model)

visualizer.fit(X)  # Fit the data to the visualizer
visualizer.show()  # Finalize and render the figure
```

```
/Users/fatih/opt/anaconda3/lib/python3.8/site-
packages/sklearn/manifold/_mds.py:512: UserWarning:

The MDS API has changed. ``fit`` now constructs an dissimilarity matrix from
data. To use a custom dissimilarity matrix, set ``dissimilarity='precomputed'``.
```



[89]: <AxesSubplot:title={'center':'KMeans Intercluster Distance Map (via MDS)'}, xlabel='PC2', ylabel='PC1'>

[90]:
```
# find the k-means cluster labels
model = KMeans(n_clusters = 3).fit(X)
df_RFM = df_RFM.copy()
df_RFM['Cluster'] = list(model.labels_)
df_RFM.head()
```

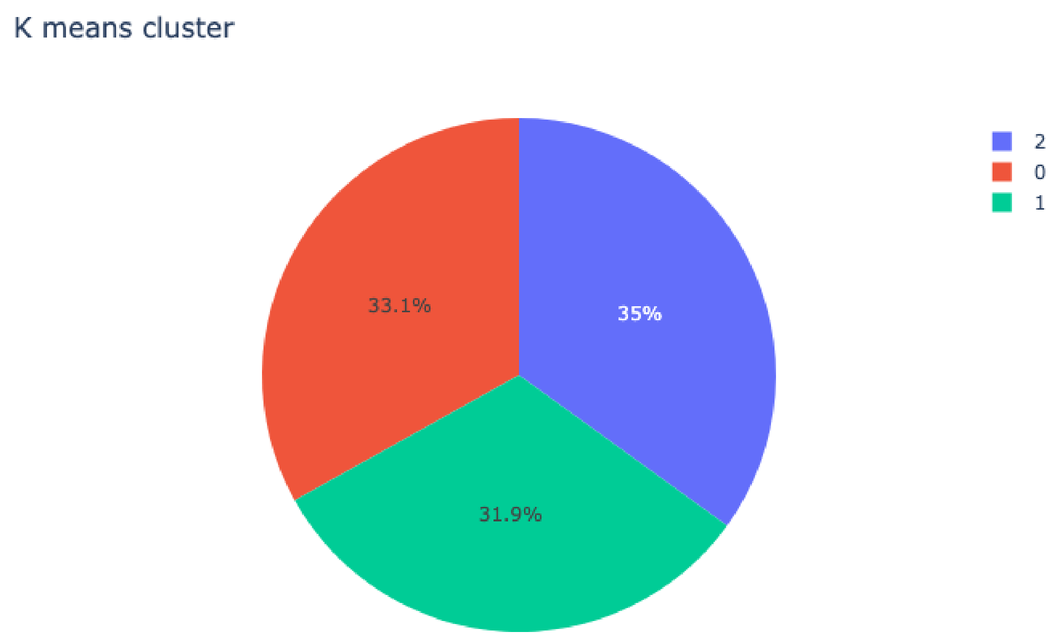
[90]:

	recency	frequency	monetary	Cluster
0	326	1	77183.60	0
1	2	11	4196.01	1
2	1	210	33053.19	1
3	4	5	4090.88	1
4	3	4	942.34	1

[91]:
```
df_RFM['Cluster'].value_counts()
```

[91]:
```
2    1371
0    1298
1    1252
Name: Cluster, dtype: int64
```

[92]:
```
fig = px.pie(values = df_RFM['Cluster'].value_counts(),
             names = (df_RFM['Cluster'].value_counts()).index,
             title = 'K means cluster')
fig.show("png")
```



As seen above we have devided the custumers into 3 groups. But we still don't know what they mean. We can do more analtyz and see the caracteristic of each cluster. We want to learn which one is the most frequent buyer, which group is the top customer, exc.. We can also do some investigatuion and lean what type of items each cluster buys.

1.5.1 What does each cluster means?

To answer this question we can first use our logic in the RFM model. First we need to merge the data so see what customer categories each cluster have.

[93]:
```
df_RFM.head()
```

[93]:

	recency	frequency	monetary	Cluster
0	326	1	77183.60	0
1	2	11	4196.01	1
2	1	210	33053.19	1
3	4	5	4090.88	1
4	3	4	942.34	1

[94]:
```
df_UK_customer.head()
```

[94]:
```
   CustomerID  recency  frequency  monetary  recency_score  frequency_score  \
0     12346.0      326          1  77183.60              1                1
1     12747.0        2         11   4196.01              4                4
2     12748.0        1        210  33053.19              4                4
3     12749.0        4          5   4090.88              4                3
4     12820.0        3          4    942.34              4                3

   monetary_score RFM_Score    Customer_Category
0               4       114    Top Lost Customer
1               4       444             Champion
2               4       444             Champion
3               4       434  Top Recent Customer
4               3       433  Top Recent Customer
```

[95]:
```
# merge data to get both RFM categories and kmean clusters

df_clusters = pd.DataFrame()
df_clusters['CustomerID'] = df_UK_customer['CustomerID']
df_clusters['Cluster'] = df_RFM['Cluster']
df_clusters['RFM_Score'] = df_UK_customer['RFM_Score']
df_clusters['Customer_Category'] = df_UK_customer['Customer_Category']
df_clusters.head()
```

[95]:
```
   CustomerID  Cluster RFM_Score    Customer_Category
0     12346.0        0       114    Top Lost Customer
1     12747.0        1       444             Champion
2     12748.0        1       444             Champion
3     12749.0        1       434  Top Recent Customer
4     12820.0        1       433  Top Recent Customer
```

[96]:
```
df_clusters.groupby('Cluster')['Customer_Category'].unique()
```

[96]:
```
Cluster
0    [Top Lost Customer, Lost Customer, Customer Ne…
1    [Champion, Top Recent Customer, Top Customer N…
2    [Top Customer Needed Attention, Customer Neede…
Name: Customer_Category, dtype: object
```

[97]:
```
for row in df_clusters.groupby('Cluster')['Customer_Category'].unique():
    print(row)
```

```
['Top Lost Customer' 'Lost Customer' 'Customer Needed Attention'
 'Top Customer Needed Attention']
['Champion' 'Top Recent Customer' 'Top Customer Needed Attention'
 'Top Lost Customer' 'Top Loyal Customer' 'Recent Customer']
['Top Customer Needed Attention' 'Customer Needed Attention'
 'Recent Customer' 'Loyal Customer' 'Top Loyal Customer'
```

```
'Top Recent Customer']
```

[98]:
```
# for row in df_clusters.groupby('Cluster')['RFM_Score'].unique():
#     print(row)
```

We understand that - Cluster 0 means Currrent Customer - Cluster 1 means Top Customer - Clister 2 means Lost Customer

1.6 4. Additional possible solutions with the data

1. Sales recomendation

With the data we have, we can anticipate the customers and do sales recomendations.

2. Sales forecast

From the available date we can additionaly do and sales forcast.

3. Customer Segmentation based on consumption habits

The customers can be segmented the customers by their consumption habits. The item category information would be helpful. The customers can be classified based the category of products, the number of purchase, and total payment.

Additional data that might help with the proposed solutions:

- Item categories. (Household, Gardening, Chirstmas)
- Data with longer period of time
- Customer demographic info
- If online sales: Customer Search data, click data

1.7 5. Conclusion

In this project, Customer Lifetime Value RFM Segmentation, K-Means Clustering are used to help the marketing team to segment the customers in order to better target them for email campaigns to increase sales.

We know that different groups require different marketing approaches and we want to figure out which group can boost the profit the most.

To do that, the customers in the dataset are divided into clusters with RFM Segmentation based on customer purchase history and custumer Lifetime value. We were able to categoriged the data into 9 groups including "Top customer need attention" and "customers need attention" so that the marketing team can prioritize their strategy.

With Kmeans unsupervised machine learning algorithm, we have classified the customers into three clusters. Cluster 0 means "Current Customer", Cluster 1 means "Top Customer", and Clister 2 means "Lost Customers". The marketing team can still use the clustered data to personalize the promotions to each group based on their needs. However, RFM model gives more information than K-means algorithm. We can re run the algorith with more features and plot the results to understand the custumor type more clearly.

[ ]:
```

```