Customer Segmentation (Coeur de cible, February 2019)

Data visualization and RFM(Recency, Frequency and Monetary) analysis using Python-Customer Segmentation

We are in the middle of a digital transformation and most of our daily needs such as purchasing items, travelling or searching on internet (clothes, phones, food, etc.) generate a large amount of data. Companies today rely on this data to analyse and understand their customers' behaviour and to segment these customers, in order to improve their marketing campaigns. **Customer segmentation** is a very common method used by retailers.

RFM analysis is a technique often used to perform in customer segmentation. RFM will take into account the **recency**, (i.e. the date on which the customer made his last order), then it will take into account the **frequency** of orders and the **amount** of the purchased items (over a given period of time or the last order) to establish the different customer segments.

As an example of RFM analysis, we will use retail customer data in this study, using Python and some of its visualization libraries and tools.

Our data set contains information about customers in different states of the US. Those customers made 5009 purchasing orders online between **2016–01–02 to 2019–12–30**. The features (columns) are :

Order ID : Unique order identifier online.
Order Date : Date when to customer order an item.
Customer ID : Unique customer identifier.
State : The name of the country where each customer resides.
Product ID : Unique Product identifier.
Category : Product Category name.
Quantity : The quantities of each product per transaction.
Price : Unit price of a product.

We would like to remind that data cleaning is done, (i.e. we removed all the errors terms and missing data in our set before using it) We did not mentioned this previous step, as it is not the main goal of the project.

Let’s do a little **data visualization** in order to better understand our data.

First of all, we import these libraries.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
import chart_studio
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
```

We import our data set to see how it looks.

df=pd.read_csv(‘Sales.csv’)
df.head()

Order ID	Order Date	Customer ID	State	Product ID	Product	Quantity	Price
CA-2018-152156	08/11/2018	CG-12520	Kentucky	FUR-BO-10001798	Bookcases	2	130.9800
CA-2018-152156	08/11/2018	CG-12520	Kentucky	FUR-CH-10000454	Chairs	3	243.9800
CA-2018-138688	12/06/2018	DV-13045	California	OFF-LA-10000240	Labels	2	7.3100
US-2017-108966	11/10/2017	SO-20335	Florida	FUR-TA-10000577	Tables	5	191.5155
US-2017-108966	11/10/2017	SO-20335	Florida	OFF-ST-10000760	Storage	2	11.1840

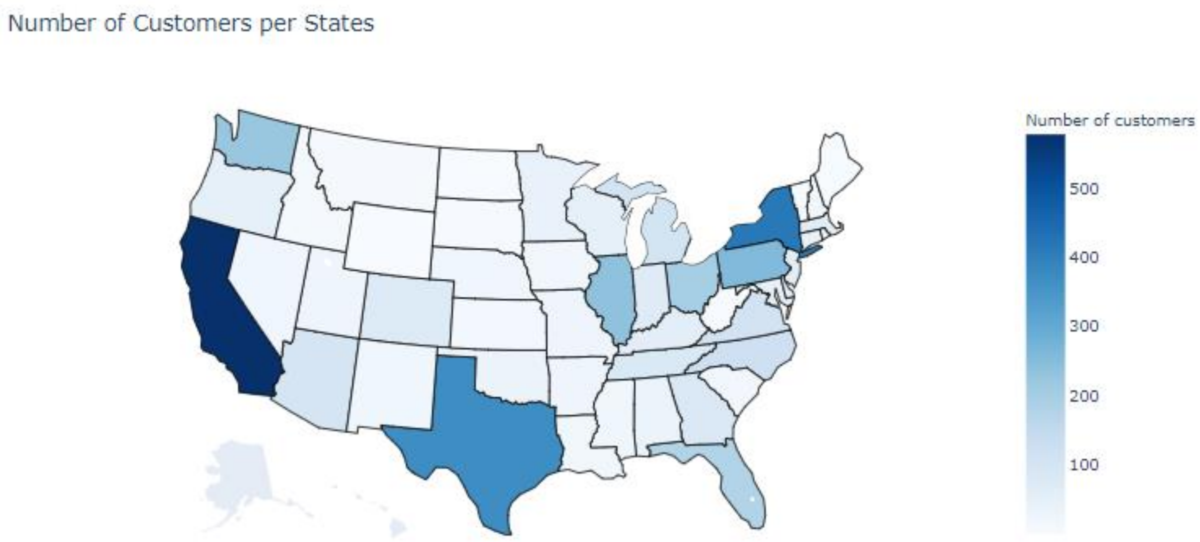
The first lines of the retail data set

```
codes= {'Kentucky':'KY', 'California':'CA', 'Florida':'FL', 'North Carolina':'NC','Washington':'WA',
         'Texas':'TX', 'Wisconsin':'WI', 'Utah':'UT', 'Nebraska':'NE','Pennsylvania':'PA', 'Illinois':'IL',
         'Minnesota':'MN', 'Michigan':'MI', 'Delaware':'DE','Indiana':'IN', 'New York':'NY', 'Arizona':'AZ',
         'Virginia':'VA', 'Tennessee':'TN','Alabama':'AL', 'South Carolina':'SC', 'Oregon':'OR', 'Colorado':'CO',
         'Iowa':'IA', 'Ohio':'OH',
        'Missouri':'MO', 'Oklahoma':'OK', 'New Mexico':'NM', 'Louisiana':'LA', 'Connecticut':'CT', 'New Jersey':'NJ',
         'Massachusetts':'MA', 'Georgia':'GA', 'Nevada':'NV', 'Rhode Island':'RI',
        'Mississippi':'MS', 'Arkansas':'AR', 'Montana':'MT', 'New Hampshire':'NH', 'Maryland':'MD',
        'District of Columbia':'DC', 'Kansas':'KS', 'Vermont':'VT', 'Maine':'ME',
        'South Dakota':'SD', 'Idaho':'ID', 'North Dakota':'ND', 'Wyoming':'WY',
        'West Virginia':'WV'}
df['code']=df['State'].map(codes)
#We're looking at the number of distinct customers by states.
temp=df.groupby('code')['Customer ID'].nunique().sort_values(ascending=False).reset_index()
fig = go.Figure(data=go.Choropleth(
    locations=temp['code'],
    z=temp['Customer ID'],
    locationmode='USA-states',
    colorscale='Blues',
    autocolorscale=False,
    marker_line_color='black', # line markers between states
    colorbar_title="Number of customers"
))
fig.update_layout(
    title_text='Number of Customers per States',
    geo = dict(
        scope='usa',
        projection=go.layout.geo.Projection(type = 'albers usa'),
        showlakes=True,
        lakecolor='rgb(255, 255, 255)'),

)
fig.show()

```

Here is our map chart containing clients by state



Number of Customers per State in the USA

As you can see, there are some states with a low number of customers. We decided to consider just the first 12 states because they are at least twice as representative in number of subscribers as the others.

But again, it all depends on what marketing strategy you decide to lead.

```
Entrée [168]:
top_12_state=list(customer_states['State'].unique())
df=df[df.State.isin(top_12_state)]
df.drop('code',inplace=True,axis=1)#We remove the column 'code' that we added to make the map chart
df.head()
```

Out[168]:

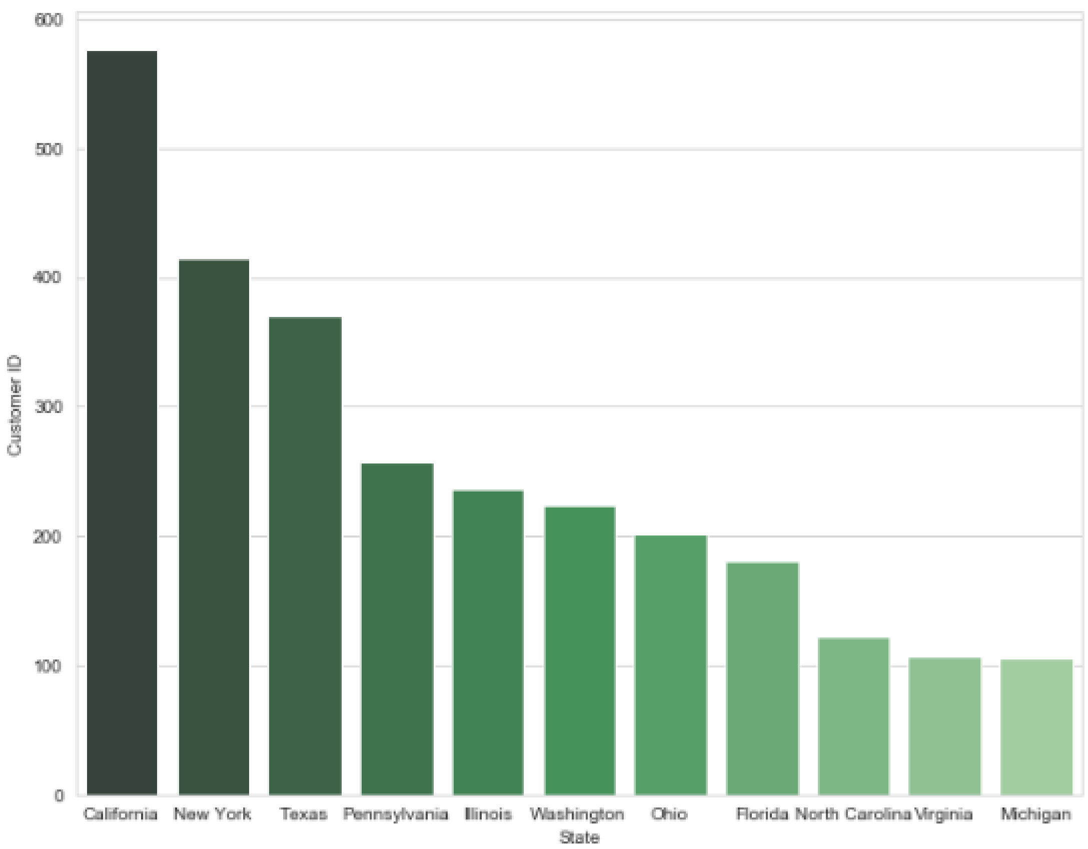
	Order ID	Order Date	Customer ID	State	Product ID	Product	Quantity	Price
2	CA-2018-138688	12/06/2018	DV-13045	California	OFF-LA-10000240	Labels	2	7.3100
3	US-2017-108966	11/10/2017	SO-20335	Florida	FUR-TA-10000577	Tables	5	191.5155
4	US-2017-108966	11/10/2017	SO-20335	Florida	OFF-ST-10000760	Storage	2	11.1840
5	CA-2016-115812	09/06/2016	BH-11710	California	FUR-FU-10001487	Furnishings	7	6.9800
6	CA-2016-115812	09/06/2016	BH-11710	California	OFF-AR-10002833	Art	4	1.8200

Head of new data set

A visualization of these selected states will be:

```
customer_states=df.groupby('State')['Customer ID'].nunique().sort_values(ascending=False).reset_index().head(11)
plt.figure(figsize=(10,8))
sns.barplot(data=customer_states,x='State',y='Customer ID',palette="Greens_d",orient=True)
```

The result is:



Top 12 States With the high Level of Customers in the USA

Now, let's start our [RFM](#)analysis.

- **We start by calculating the recency (R) of a customer.**

In order to determine how much time has passed since his last purchase, we need a reference date from which to start calculating. Then we make the difference between the two dates to calculate the amount of time passed.

Suppose that we decide to do our analysis 2 days (this date is our reference date) after the last transaction record date of our data set.

Entrée [176]:

```
df['Order Date'] = pd.to_datetime(df['Order Date'])#We transform the column into a date type
df_recency=df# To not modify df
reference_date=pd.to_datetime('1/1/2020')# our reference date
df_recency= df_recency.groupby(by='Customer ID', as_index=False)['Order Date'].max()
df_recency.columns = ['Customer ID','max_Date']
#The difference between the reference date and the last date of a customer's purchase(max_date)
#is used to obtain the number of days (recency).
df_recency['Recency'] = df_recency['max_Date'].apply(lambda row: (reference_date - row).days)
df_recency.drop('max_Date',inplace=True,axis=1)
df_recency[['Customer ID','Recency']].head()
```

Out[176]:

	Customer ID	Recency
0	AA-10315	669
1	AA-10375	50
2	AA-10480	261
3	AA-10645	235
4	AB-10015	1277

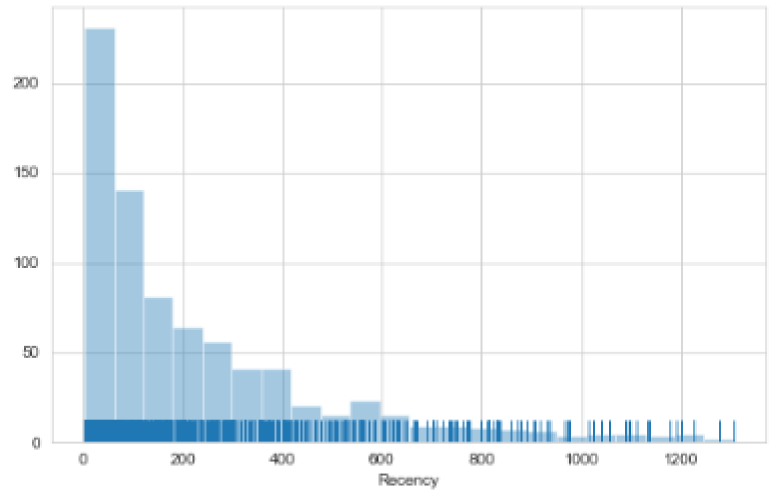
Head of recency

Let’s look at customer distribution based on recency.

Entrée [179]:

```
plt.figure(figsize=(8,5))
sns.distplot(df_recency.Recency, kde=False,rug=True )
```

Out[179]: <matplotlib.axes._subplots.AxesSubplot at 0x1b1f79f4160>



Histogram of customer’s recency

Here, we can note that the histogram is biased towards the left side and hence this is a sign of distribution which is a right-skewed distribution and also we can see that the rug plot is crowded between 0 and 400. Based on that we can see that we have a high concentration of customers in the last 400 days, i.e. the last 4 months.

- **Now we're going to determine the frequency (F) at which customers buy the products.**

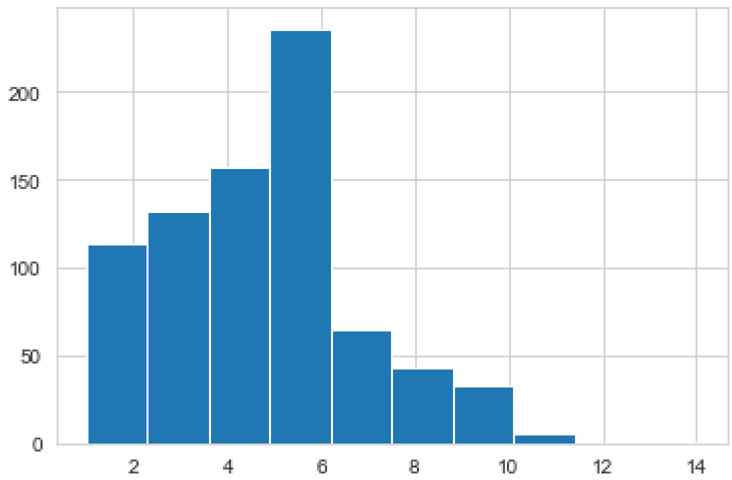
Entrée [255]:

```
df_frequency=df #We're taking back the data we had before we calculated the recency
df_frequency = df_frequency.groupby('Customer ID')['Order ID'].nunique()
df_frequency.reset_index()
df_frequency.columns=['Customer ID','Frequency']
df_frequency.head()
```

Out[255]:

```
Customer ID
AA-10315    4
AA-10375    3
AA-10480    2
AA-10645    3
AB-10015    2
Name: Order ID, dtype: int64
```

Head of frequency



Histogram of Customer's Frequency

In our database, we can see that the majority of customers do not buy more than 10 times.it is not really enough because our data is over a period of 04 years.

- **Now at the end, we're going to determine the Monetary(M).**

```
df_monetary=df
df_monetary['Monetary']=df_monetary['Quantity']*df_monetary['Price']
df_monetary = df_monetary.groupby('Customer ID',as_index=False)['Monetary'].sum()
df_monetary.columns = ['Customer ID','Monetary']
df_monetary.head()
```

	Customer ID	Monetary
0	AA-10315	4645.540
1	AA-10375	65.744
2	AA-10480	27.112
3	AA-10645	208.418
4	AB-10015	61.336

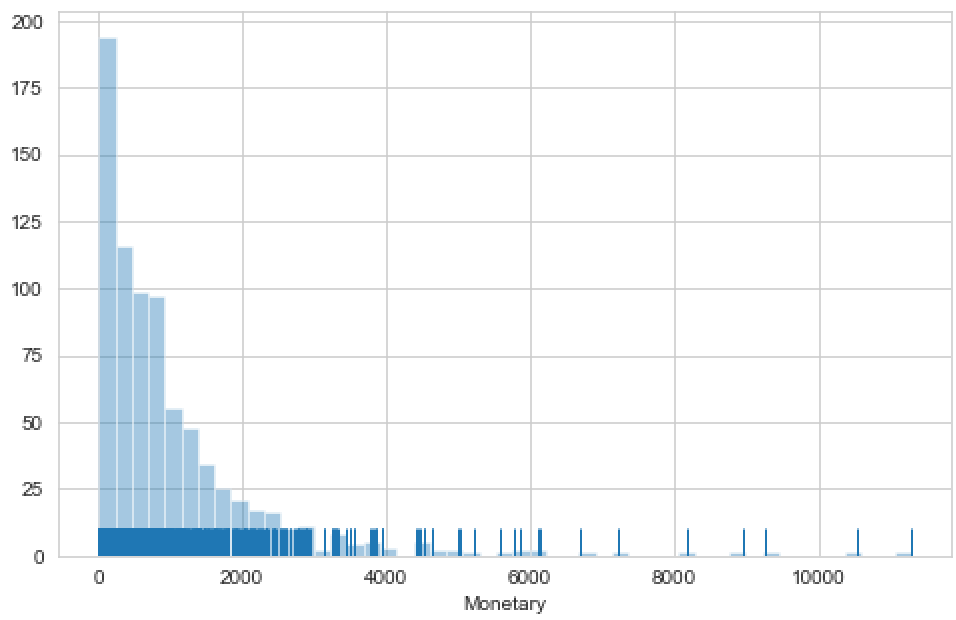
Head of Monetary

Let's look at customer distribution based on recency

Entrée [257]:

```
plt.figure(figsize=(8,5))
sns.distplot(df_monetary.Monetary, kde=False,rug=True )
```

Out[257]: <matplotlib.axes._subplots.AxesSubplot at 0x1b1f765a7b8>



Histogram of customer's Monetary

It is marked that not many customers spend more than $3,000.

Now let's check the **RFM table**.

Entrée [277]:
```
#We merge R and F
recency_frequency_monetary= recency_frequency.merge(df_monetary,on='Customer ID')
# Then we merge R_F and M
recency_frequency_monetary.set_index('Customer ID',inplace=True)
#Now we have the RFM
recency_frequency_monetary.head()
```

Out[277]:

Customer ID	Recency	Frequency	Monetary
AA-10315	669	4	4645.540
AA-10375	50	3	65.744
AA-10480	261	2	27.112
AA-10645	235	3	208.418
AB-10015	1277	2	61.336

Head of RFM

Thanks to our **RFM table**, we are now going to propose a customer segmentation strategy.

Customer Segmentation

Depending on the company's objectives, customers can be segmented in several ways so that it is financially possible to make marketing campaigns. The ideal customers for e-commerce companies are generally the most recent ones compared to the date of study (our reference date), who are very frequent and who spend enough.
The **RFM factors** are therefore very important if you want to know that they are the right customers to send promotional e-mails or to offer new services.

Based on the RFM table, we will assign a score to each customer between 1 and 3 for each RFM value of a customer.

3 is the best score. 1 is the worst score. The RFM score of a client is calculated by combining the three scores obtained at R, F and M. For example, the client with ID-1 has a score of 3 in Recency, a score of 3 in Frequency and a score of 3 in Amount (**Monetary**). His RFM score is therefore 3–3–3. In concrete terms, **this is a client who bought most recently and most often, and he spent the most.**

To find the R score, we decide to take the lowest monthly interval, because our distribution is very skewed to the right, so we have to choose only 1 month.

Below we are developing a function which allows us to find the recency score for each customer.

```python
def recency_score(x):
    if x["Recency"] <=60:
        recency = 3
    elif x["Recency"] >60  and x["Recency"] <= 120:
         recency = 2
    else:
        recency = 1
    return recency
recency_frequency_monetary["R"] = recency_frequency_monetary.apply(recency_score, axis=1)
```

We do the same for the frequency of customer purchases.

```python
def frequency_score(x):
    if x["Frequency"] <=4:
        frequency = 1
    elif x["Frequency"] >4  and x["Frequency"] <=8:
         frequency = 2
    else:
        frequency = 3
    return frequency
recency_frequency_monetary["F"] = recency_frequency_monetary.apply(frequency_score, axis=1)
```

Another simple way to calculate a Monetary score is to use tertiles. The **'qcut'** function of pandas will divide the entire range of unique "Monetary" in 3 equal parts. (intervals highlighted in yellow on the image above).

For more information on the "qcut" function see the documentation on **"pd.qcute( )".**

```
Entrée [911]:   pd.qcut(recency_frequency_monetary['Monetary'], q=3).head()

  Out[911]: 0     (992.91, 11291.28]
            1       (2.807, 356.519]
            2       (2.807, 356.519]
            3       (2.807, 356.519]
            4       (2.807, 356.519]
            Name: Monetary, dtype: category
            Categories (3, interval[float64]): [(2.807, 356.519] < (356.519, 992.91] < (992.91, 11291.28]]
```

These intervals will then be labelled from 1 to 3, in the same way as the recency and the frequency.

```python
monetary_score = pd.qcut(recency_frequency_monetary['Monetary'], q=3, labels=range(1, 4))
recency_frequency_monetary = recency_frequency_monetary.assign(M = monetary_score.values)
```

So we get our final table with all the scores of recency, frequency, and Monetary

```
Entrée [451]:  recency_frequency_monetary.head()
```

Out[451]:

Customer ID	Recency	Frequency	Monetary	R	F	M
AA-10315	669	4	4645.540	1	1	3
AA-10375	50	3	65.744	3	1	1
AA-10480	261	2	27.112	1	1	1
AA-10645	235	3	208.418	1	1	1
AB-10015	1277	2	61.336	1	1	1

head of R, F, M

The final RFM score will be obtained by concatenating all the different R, F and M scores. This will allow us to define customer segments. As customer segmentation really depends on companies's objectives, we will just settle for the most common segments found in the marketing field. We have been inspired by the segments defined by **Joao Correia** in his article that you will find **here**.

```
Entrée [461]:
def rfm_score(row):
    return str(row['R']) + str(row['F']) + str(row['M'])
recency_frequency_monetary['RFM_Score'] = recency_frequency_monetary.apply(rfm_score, axis=1)
recency_frequency_monetary.head()
```

Out[461]:

Customer ID	Recency	Frequency	Monetary	R	F	M	RFM_Score
AA-10315	669	4	4645.540	1	1	3	113
AA-10375	50	3	65.744	3	1	1	311
AA-10480	261	2	27.112	1	1	1	111
AA-10645	235	3	208.418	1	1	1	111
AB-10015	1277	2	61.336	1	1	1	111

Head of RFM_Score

The common key segments that we will identify in our data set, are shown in the table below.

Segment	RFM	Description
Best Customers	333	Bought most recently and most often, and spend the most
Loyal Customers	1X1	Buy most frequently
Big Spenders	XX3	Spend the most
Almost Lost	233	Haven't purchased for some time, but purchased frequently and spend the most
Lost Customers	133	Haven't purchased for some time, but purchased frequently and spend the most
Lost Cheap Customers	111	Last purchased long ago, purchased few, and spent little

Key segments (**Joao Correia, July 2016)**

The code below allows us to create a new column "segment" which represents the segment in which our customer is located. We first start to identify the segments: **'Lost Cheap customers','Lost Customer,'Best Customers', and 'Almost Customers'. We assign 'others' for others**

```
best=list(recency_frequency_monetary.loc[recency_frequency_monetary["RFM_Score"]=="333"].index)
lost_cheap=list(recency_frequency_monetary.loc[recency_frequency_monetary["RFM_Score"]=="111"].index)
lost=list(recency_frequency_monetary.loc[recency_frequency_monetary["RFM_Score"]=="133"].index)
lost_almost=list(recency_frequency_monetary.loc[recency_frequency_monetary["RFM_Score"]=="233"].index)
for i in recency_frequency_monetary.index:
    if i in lost_cheap:
        recency_frequency_monetary.Segment.iloc[i]='Lost Cheap Customers'
    elif i in lost:
        recency_frequency_monetary.Segment.iloc[i]='Lost Costumer'
    elif i in best:
        recency_frequency_monetary.Segment.iloc[i]='Best Customers'
    elif i in lost_almost:
        recency_frequency_monetary.Segment.iloc[i]='Almost Lost'
    else:
        recency_frequency_monetary.Segment.iloc[i]='Others'
```

The customers in the **'Loyal customers'** segment are then identified and assigned to 'segment' as well.

```
loyal=list(recency_frequency_monetary.loc[recency_frequency_monetary["F"]==3].index)
loyal2=[]
for i in loyal:
    if i not in best and i not in lost_cheap and i not in lost_almost and i not in lost:
        loyal2.append(i)
for i in recency_frequency_monetary.index:
    if i in loyal2:
            recency_frequency_monetary.Segment.iloc[i]='Loyal Customers'
```

We do the same thing for the **'Big Spenders'** segment.

```python
big=list(recency_frequency_monetary.loc[recency_frequency_monetary["M"]==3].index)
big2=[]
for i in big:
    if i not in best and i not in lost_cheap and i not in lost_almost and i not in lost:
        big2.append(i)
for i in recency_frequency_monetary.index:
    if i in big2:
            recency_frequency_monetary.Segment.iloc[i]='Big Spenders'
```

We, therefore, have our customers in the segments that we have defined.

```python
recency_frequency_monetary.head()
```

	Customer ID	Recency	Frequency	Monetary	R	F	M	RFM_Score	Segment
0	AA-10315	669	4	4645.540	1	1	3	113	Big Spenders
1	AA-10375	50	3	65.744	3	1	1	311	Others
2	AA-10480	261	2	27.112	1	1	1	111	Lost Cheap Customers
3	AA-10645	235	3	208.418	1	1	1	111	Lost Cheap Customers
4	AB-10015	1277	2	61.336	1	1	1	111	Lost Cheap Customers

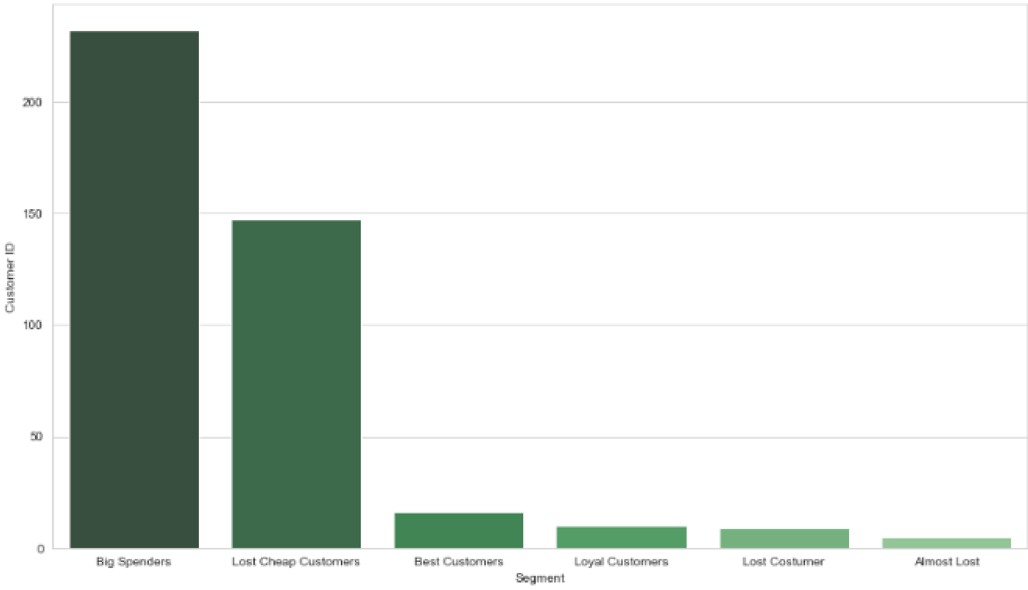
Head of segment

Of course, these segments can be defined differently according to the objectives that the company sets itself. Customers in the **“others”** segment can allow us to define even more. Given that we have three groups (R, F, M) labelled from 1 to 3, we have **3X3X3=27 possibilities of segmentation.**

We are going to do a small visualization of our segments. Then, we will plot the key segments and then give some marketing recommendations that the company could take to retain those customers

```
sq1=recency_frequency_monetary.groupby('Segment')['Customer ID'].nunique().sort_values(ascending=False).reset_index()
plt.figure(figsize=(14,8))
sq1.drop([0],inplace=True)
sns.barplot(data=sq1,x='Segment',y='Customer ID',palette="Greens_d",orient=True)
```

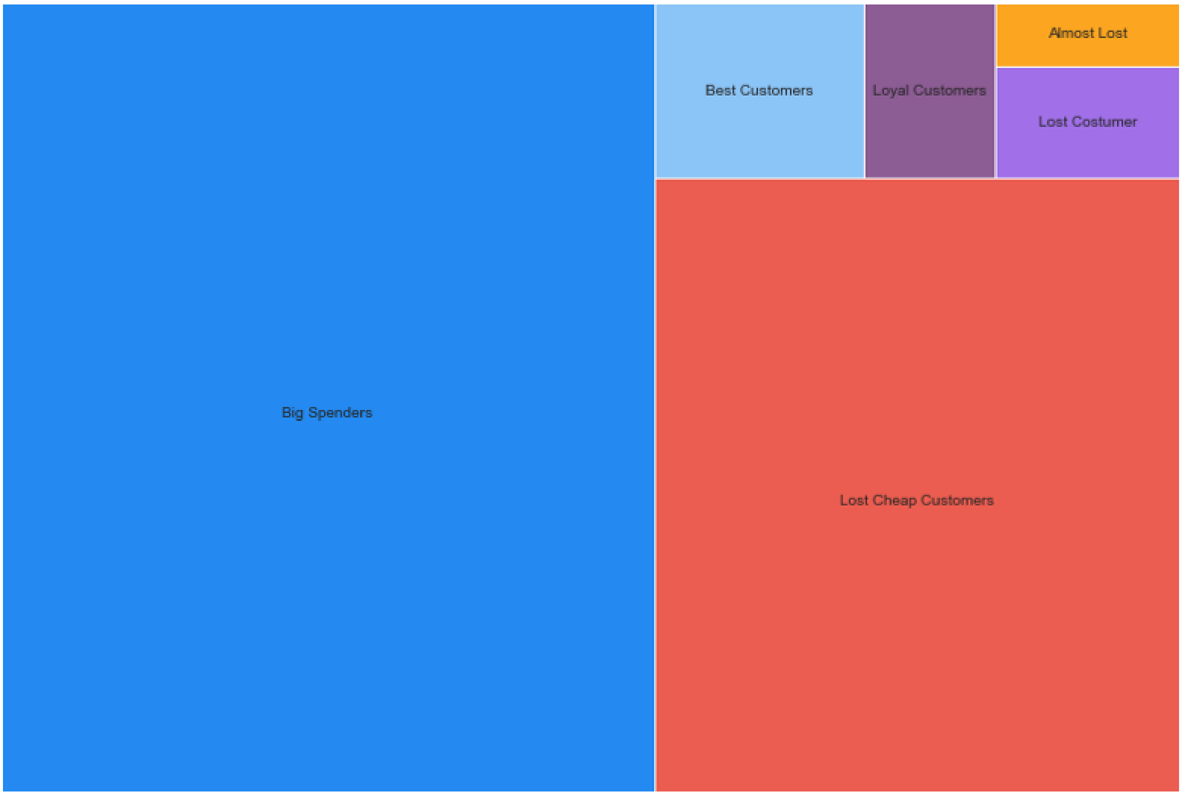
<matplotlib.axes._subplots.AxesSubplot at 0x1b1fdda06a0>



Customers per segments

Another more expressive visualization to show the distribution of segments using the squarify plot of matplotlib gives:

```python
import squarify
cmap = matplotlib.cm.coolwarm
mini = min(sq1['Customer ID'])
maxi = max(sq1['Customer ID'])
norm = matplotlib.colors.Normalize(vmin=mini, vmax=maxi)
colors = [cmap(norm(value)) for value in sq1['Customer ID']]
fig = plt.gcf()
ax = fig.add_subplot()
fig.set_size_inches(14, 10)
squarify.plot(sizes=sq1['Customer ID'],
              label=sq1.Segment, alpha=1,color=colors)
plt.axis('off')
plt.show()
```



Customer's Segments

These two graphs show that the "Big spenders", "Lost Cheap Customers" segment are the highest and "Almost Lost" the lowest.

Recommandations

- **Best Customers**: Rewards them for their multiples purchases. They can be early adopters to very new products. Suggest them "Refer a friend". Also, they can be the most loyal customers that have the habit to order.
- **Lost Cheap Customers:** Send them personalized emails to encourage them to order.
- **Big Spenders:** Inform them about the discounts to keep them spending more and more money on your products
- **Loyal Customers:** Create loyalty cards in which they can gain points each time of purchasing and these points could transfer into a discount

In our next articles, we will show how machine learning can help in customer segmentation and we will focus on **unsupervised learning** algorithms such as **Kmeans.**