Customer Segmentation Project for an eCommerce Company

In this project the customers of an English eCommerce Company will be segmented. The data belonging 2010 and 2011 will be retrieved from "https://archive.ics.uci.edu/ml/datasets/Online+Retail+II (https://archive.ics.uci.edu/ml/datasets/Online+Retail+II)". RFM analysis method will be used to segment the customers.

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns

# to display all columns and rows:
pd.set_option('display.max_columns', None); pd.set_option('display.max_rows', None);


pd.set_option('display.float_format', lambda x: '%.0f' % x)
import matplotlib.pyplot as plt
```

In [2]:

```
#The data downloaded from "https://archive.ics.uci.edu/ml/datasets/Online+Retail+II" in
excel format and uploaded.

df_2010_2011 = pd.read_excel("online_retail_II.xlsx", sheet_name = "Year 2010-2011")
```

In [3]:

```
df = df_2010_2011.copy()
```

In [4]:

```
df.shape
```

Out[4]:

```
(541910, 8)
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541910 entries, 0 to 541909
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   Invoice      541910 non-null  object        
 1   StockCode    541910 non-null  object        
 2   Description  540456 non-null  object        
 3   Quantity     541910 non-null  int64         
 4   InvoiceDate  541910 non-null  datetime64[ns]
 5   Price        541910 non-null  float64       
 6   Customer ID  406830 non-null  float64       
 7   Country      541910 non-null  object        
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

In [6]:

```
df.head()
```

Out[6]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	3	17850	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3	17850	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	3	17850	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3	17850	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3	17850	United Kingdom

In [7]:

```
#The numbers of unique items

df['Description'].nunique()
```

Out[7]:

```
4223
```

In [8]:

```
#The numbers of items one by one

df["Description"].value_counts().head()
```

Out[8]:

```
WHITE HANGING HEART T-LIGHT HOLDER    2369
REGENCY CAKESTAND 3 TIER              2200
JUMBO BAG RED RETROSPOT               2159
PARTY BUNTING                         1727
LUNCH BAG RED RETROSPOT               1638
Name: Description, dtype: int64
```

In [9]:

```
#The most ordered product?

df.groupby("Description").agg({"Quantity":"sum"}).sort_values("Quantity", ascending = False).head()
```

Out[9]:

Description	Quantity
WORLD WAR 2 GLIDERS ASSTD DESIGNS	53847
JUMBO BAG RED RETROSPOT	47363
ASSORTED COLOUR BIRD ORNAMENT	36381
POPCORN HOLDER	36334
PACK OF 72 RETROSPOT CAKE CASES	36039

In [10]:

```
#Total invoices

df['Invoice'].nunique()
```

Out[10]:

```
25900
```

In [11]:

```
#How much money has been earned in average per invoice?

df["TotalPrice"] = df["Quantity"]*df["Price"]
```

In [12]:

```
df.head()
```

Out[12]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	TotalP
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	3	17850	United Kingdom	
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3	17850	United Kingdom	
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	3	17850	United Kingdom	
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3	17850	United Kingdom	
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3	17850	United Kingdom	

In [13]:

```
# Total amount per invoice

df.groupby("Invoice").agg({"TotalPrice":"sum"}).head()
```

Out[13]:

	TotalPrice
Invoice	
536365	139
536366	22
536367	279
536368	70
536369	18

In [14]:

```
#The most expensive products

df.sort_values('Price', ascending = False).head()
```

Out[14]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Countr
222681	C556445	M	Manual	-1	2011-06-10 15:31:00	38970	15098	Unite Kingdo
524602	C580605	AMAZONFEE	AMAZON FEE	-1	2011-12-05 11:36:00	17836	nan	Unite Kingdo
43702	C540117	AMAZONFEE	AMAZON FEE	-1	2011-01-05 09:55:00	16888	nan	Unite Kingdo
43703	C540118	AMAZONFEE	AMAZON FEE	-1	2011-01-05 09:57:00	16454	nan	Unite Kingdo
16356	C537651	AMAZONFEE	AMAZON FEE	-1	2010-12-07 15:49:00	13541	nan	Unite Kingdo

In [15]:

```
# Invoices including C (returns) and A (has problem) should be excluded

df_new = df[~(df['Invoice'].str.contains('C',na=False) | df['Invoice'].str.contains('A'
,na=False))]
```

In [16]:

```
df_new.sort_values("Price", ascending = False).head()
```

Out[16]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
15017	537632	AMAZONFEE	AMAZON FEE	1	2010-12-07 15:08:00	13541	nan	United Kingdom
173382	551697	POST	POSTAGE	1	2011-05-03 13:46:00	8143	16029	United Kingdom
297725	562955	DOT	DOTCOM POSTAGE	1	2011-08-11 10:14:00	4505	nan	United Kingdom
268028	560373	M	Manual	1	2011-07-18 12:30:00	4288	nan	United Kingdom
422377	573080	M	Manual	1	2011-10-27 14:20:00	4161	12536	France

In [17]:

```
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 532619 entries, 0 to 541909
Data columns (total 9 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   Invoice      532619 non-null  object        
 1   StockCode    532619 non-null  object        
 2   Description  531165 non-null  object        
 3   Quantity     532619 non-null  int64         
 4   InvoiceDate  532619 non-null  datetime64[ns]
 5   Price        532619 non-null  float64       
 6   Customer ID  397925 non-null  float64       
 7   Country      532619 non-null  object        
 8   TotalPrice   532619 non-null  float64       
dtypes: datetime64[ns](1), float64(3), int64(1), object(4)
memory usage: 40.6+ MB
```

In [18]:

```
# Number of orders by country

df_new["Country"].value_counts()
```

Out[18]:

```
United Kingdom          487619
Germany                   9042
France                    8409
EIRE                      7894
Spain                     2485
Netherlands               2363
Belgium                   2031
Switzerland               1967
Portugal                  1501
Australia                 1185
Norway                    1072
Italy                      758
Channel Islands            748
Finland                    685
Cyprus                     614
Sweden                     451
Unspecified                446
Austria                    398
Denmark                    380
Poland                     330
Japan                      321
Israel                     295
Hong Kong                  284
Singapore                  222
Iceland                    182
USA                        179
Canada                     151
Greece                     145
Malta                      112
United Arab Emirates        68
European Community          60
RSA                         58
Lebanon                     45
Lithuania                   35
Brazil                      32
Czech Republic              25
Bahrain                     18
Saudi Arabia                 9
Name: Country, dtype: int64
```

In [19]:

```
# The amount of income by country

df_new.groupby("Country").agg({"TotalPrice":"sum"}).sort_values("TotalPrice", ascending
= False).head()
```

Out[19]:

Country	TotalPrice
United Kingdom	9014160
Netherlands	285446
EIRE	283454
Germany	228867
France	209733

In [20]:

```
# the most returned product (PAPER CRAFT , LITTLE BIRDIE)

df_return = df[df['Invoice'].str.contains('C',na=False)]
groupedC = df_return.groupby('Description',sort=False).agg({'Quantity':'sum','Price':'m
ax','TotalPrice':'sum'})
groupedC.sort_values('Quantity',ascending=True).head()
```

Out[20]:

Description	Quantity	Price	TotalPrice
PAPER CRAFT , LITTLE BIRDIE	-80995	2	-168470
MEDIUM CERAMIC TOP STORAGE JAR	-74494	1	-77480
ROTATING SILVER ANGELS T-LIGHT HLDR	-9376	3	-322
Manual	-4066	38970	-146784
FAIRY CAKE FLANNEL ASSORTED COLOUR	-3150	2	-6591

Data preparation

In [21]:

```
df_new.isnull().sum()
```

Out[21]:

```
Invoice            0
StockCode          0
Description     1454
Quantity           0
InvoiceDate        0
Price              0
Customer ID    134694
Country            0
TotalPrice         0
dtype: int64
```

In [22]:

```
df_new.dropna(inplace = True)
```

```
C:\Users\Acer\anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

In [23]:

```
df_new.shape
```

Out[23]:

```
(397925, 9)
```

In [24]:

```
df_new.describe([0.01,0.05,0.10,0.25,0.50,0.75,0.90,0.95, 0.99]).T
```

Out[24]:

	count	mean	std	min	1%	5%	10%	25%	50%	75%	90%
Quantity	397925	13	180	1	1	1	1	2	6	12	24
Price	397925	3	22	0	0	0	1	1	2	4	6
Customer ID	397925	15294	1713	12346	12415	12627	12883	13969	15159	16795	17725
TotalPrice	397925	22	309	0	1	1	2	5	12	20	35

Customer Segmentation with RFM Analysis

In [25]:

```
df_new.head()
```

Out[25]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	TotalP
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	3	17850	United Kingdom	
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3	17850	United Kingdom	
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	3	17850	United Kingdom	
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3	17850	United Kingdom	
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3	17850	United Kingdom	

In [26]:

```
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 397925 entries, 0 to 541909
Data columns (total 9 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Invoice      397925 non-null  object
 1   StockCode    397925 non-null  object
 2   Description  397925 non-null  object
 3   Quantity     397925 non-null  int64
 4   InvoiceDate  397925 non-null  datetime64[ns]
 5   Price        397925 non-null  float64
 6   Customer ID  397925 non-null  float64
 7   Country      397925 non-null  object
 8   TotalPrice   397925 non-null  float64
dtypes: datetime64[ns](1), float64(3), int64(1), object(4)
memory usage: 30.4+ MB
```

In [27]:

```
df_new["InvoiceDate"].min()
```

Out[27]:

```
Timestamp('2010-12-01 08:26:00')
```

In [28]:

```
df_new["InvoiceDate"].max()
```

Out[28]:

```
Timestamp('2011-12-09 12:50:00')
```

In [29]:

```
#Since these are old data, the last date in data set will be used as today's date

import datetime as dt
today_date = dt.datetime(2011,12,9)
```

In [30]:

```
today_date
```

Out[30]:

```
datetime.datetime(2011, 12, 9, 0, 0)
```

In [31]:

```
df_new.groupby("Customer ID").agg({"InvoiceDate":"max"}).head()
```

Out[31]:

Customer ID	InvoiceDate
12346	2011-01-18 10:01:00
12347	2011-12-07 15:52:00
12348	2011-09-25 13:13:00
12349	2011-11-21 09:51:00
12350	2011-02-02 16:01:00

In [32]:

```
df_new["Customer ID"] = df_new["Customer ID"].astype(int)
```

```
C:\Users\Acer\anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
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-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

In [33]:

```
(today_date - df_new.groupby("Customer ID").agg({"InvoiceDate":"max"})).head()
```

Out[33]:

Customer ID	InvoiceDate
12346	324 days 13:59:00
12347	1 days 08:08:00
12348	74 days 10:47:00
12349	17 days 14:09:00
12350	309 days 07:59:00

In [34]:

```
temp_df = (today_date - df_new.groupby("Customer ID").agg({"InvoiceDate":"max"}))
```

In [35]:

```
temp_df.rename(columns={"InvoiceDate": "Recency"}, inplace = True)
```

In [36]:

```
temp_df.head()
```

Out[36]:

Customer ID	Recency
12346	324 days 13:59:00
12347	1 days 08:08:00
12348	74 days 10:47:00
12349	17 days 14:09:00
12350	309 days 07:59:00

In [37]:

```
#Exculiding hours

recency_df = temp_df["Recency"].apply(lambda x: x.days)
```

In [38]:

```
recency_df.head()
```

Out[38]:

```
Customer ID
12346    324
12347      1
12348     74
12349     17
12350    309
Name: Recency, dtype: int64
```

Frequency

In [39]:

```
temp_df = df_new.groupby(["Customer ID","Invoice"]).agg({"Invoice":"count"})
```

In [40]:

```
temp_df.head()
```

Out[40]:

		Invoice
Customer ID	**Invoice**	
12346	**541431**	1
12347	**537626**	31
	542237	29
	549222	24
	556201	18

In [41]:

```
temp_df.groupby("Customer ID").agg({"Invoice":"count"}).head()
```

Out[41]:

	Invoice
Customer ID	
12346	1
12347	7
12348	4
12349	1
12350	1

In [42]:

```
freq_df = temp_df.groupby("Customer ID").agg({"Invoice":"count"})
freq_df.rename(columns={"Invoice": "Frequency"}, inplace = True)
freq_df.head()
```

Out[42]:

Customer ID	Frequency
12346	1
12347	7
12348	4
12349	1
12350	1

Monetary

In [43]:

```
monetary_df = df_new.groupby("Customer ID").agg({"TotalPrice":"sum"})
```

In [44]:

```
monetary_df.head()
```

Out[44]:

Customer ID	TotalPrice
12346	77184
12347	4310
12348	1797
12349	1758
12350	334

In [45]:

```
monetary_df.rename(columns={"TotalPrice": "Monetary"}, inplace = True)
```

In [46]:

```
print(recency_df.shape,freq_df.shape,monetary_df.shape)
```

```
(4339,) (4339, 1) (4339, 1)
```

In [47]:

```
rfm = pd.concat([recency_df, freq_df, monetary_df],  axis=1)
```

In [48]:

```
rfm.head()
```

Out[48]:

Customer ID	Recency	Frequency	Monetary
12346	324	1	77184
12347	1	7	4310
12348	74	4	1797
12349	17	1	1758
12350	309	1	334

In [49]:

```
#qcut  segmenting into 5 quartile with qcut and ordering
#5 will be the best for all them

rfm["RecencyScore"] = pd.qcut(rfm["Recency"], 5, labels = [5, 4, 3, 2, 1])
```

In [50]:

```
rfm["FrequencyScore"] = pd.qcut(rfm['Frequency'].rank(method = 'first'), 5, labels = [1
, 2, 3, 4, 5])
```

In [51]:

```
rfm["MonetaryScore"] = pd.qcut(rfm['Monetary'], 5, labels = [1, 2, 3, 4, 5])
```

In [52]:

```
rfm.head()
```

Out[52]:

Customer ID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryScore
12346	324	1	77184	1	1	5
12347	1	7	4310	5	5	5
12348	74	4	1797	2	4	4
12349	17	1	1758	4	1	4
12350	309	1	334	1	1	2

In [53]:

```
rfm.describe([0.2,0.3,0.4,0.6,0.8]).T
```

Out[53]:

	count	mean	std	min	20%	30%	40%	50%	60%	80%	max
Recency	4339	91	100	-1	12	21	31	49	70	178	372
Frequency	4339	4	8	1	1	1	2	2	3	6	210
Monetary	4339	2054	8988	0	250	352	490	674	942	2058	280206

In [54]:

```
pd.set_option('display.float_format', lambda x: '%.0f' % x)
import matplotlib.pyplot as plt
```

In [55]:

```
(rfm['RecencyScore'].astype(str) +
 rfm['FrequencyScore'].astype(str) +
 rfm['MonetaryScore'].astype(str)).head()
```

Out[55]:

```
Customer ID
12346    115
12347    555
12348    244
12349    414
12350    112
dtype: object
```

In [56]:

```
rfm["RFM_SCORE"] = rfm['RecencyScore'].astype(str) + rfm['FrequencyScore'].astype(str)
+ rfm['MonetaryScore'].astype(str)
```

In [57]:

```
rfm.head()
```

Out[57]:

Customer ID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryScore	F
12346	324	1	77184	1	1	5	
12347	1	7	4310	5	5	5	
12348	74	4	1797	2	4	4	
12349	17	1	1758	4	1	4	
12350	309	1	334	1	1	2	

In [58]:

```
rfm.describe().T
```

Out[58]:

	count	mean	std	min	25%	50%	75%	max
Recency	4339	91	100	-1	16	49	140	372
Frequency	4339	4	8	1	1	2	5	210
Monetary	4339	2054	8988	0	307	674	1662	280206

In [59]:

```
# The best segment '555'

rfm[rfm["RFM_SCORE"] == "555"].head()
```

Out[59]:

Customer ID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryScore	F
12347	1	7	4310	5	5	5	
12362	2	10	5226	5	5	5	
12417	2	9	3649	5	5	5	
12433	-1	7	13376	5	5	5	
12437	0	18	4951	5	5	5	

In [60]:

```
rfm[rfm["RFM_SCORE"] == "111"].head()
```

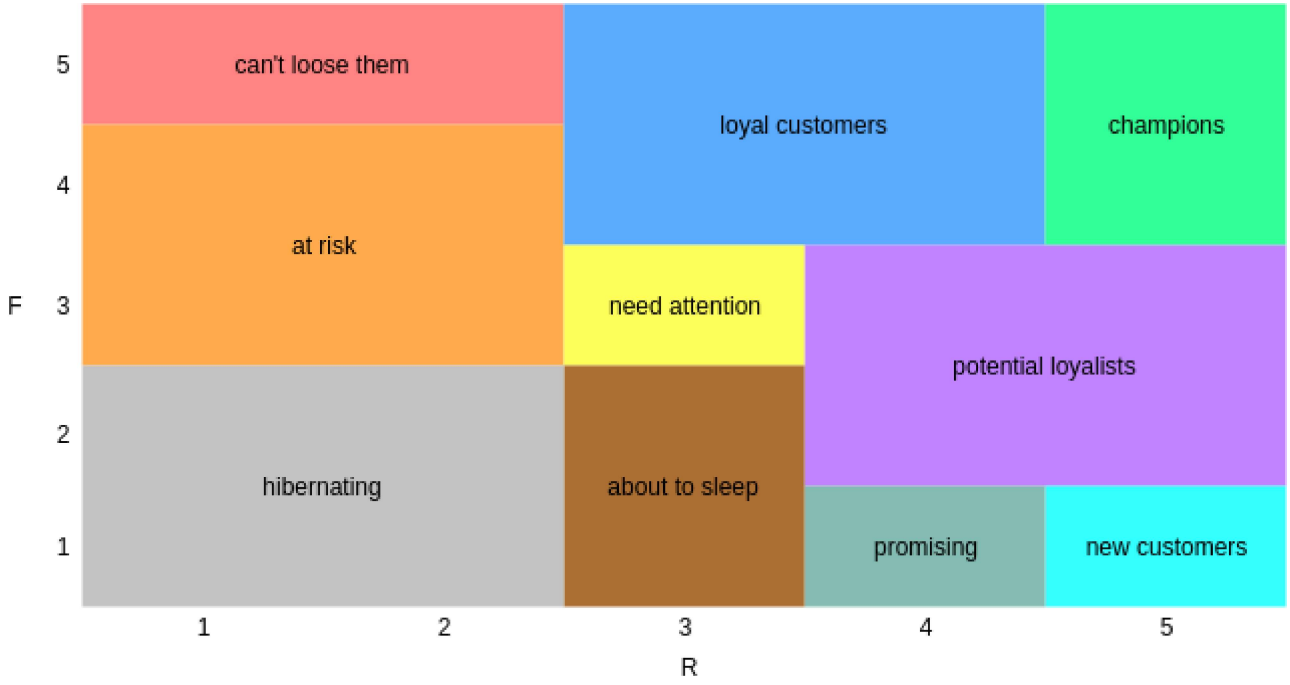
Out[60]:

Customer ID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryScore	F
12353	203	1	89	1	1	1	
12361	286	1	190	1	1	1	
12401	302	1	84	1	1	1	
12402	322	1	226	1	1	1	
12441	365	1	174	1	1	1	

In order to make a clearer and better interpretation I will segment the customers into below 10 segments based on their Recency (R) and Frequency(F) scores.

Segment	Description
Champions	Bought recently, buy often and spend the most
Loyal Customers	Buy on a regular basis. Responsive to promotions.
Potential Loyalists	Recent customers with average frequency.
New Customers	Bought most recently, but not often.
Promising	Recent shoppers, but haven't spent much.
Need Attention	Above average recency, frequency and monetary values. May not have bought very recently though.
About to Sleep	Below average recency and frequency. Will lose them if not reactivated.
At Risk	Purchased often but a long time ago. Need to bring them back!
Can't Loose Them	Used to purchase frequently but haven't returned for a long time.
Hibernating	Last purchase was long back and low number of orders. May be lost.

The resulting matrix of the above 10 customer segment:



In [61]:

```
# For a Regular Expression (recency - frequency)

seg_map = {
    r'[1-2][1-2]': 'Hibernating',
    r'[1-2][3-4]': 'At Risk',
    r'[1-2]5': 'Can\'t Loose',
    r'3[1-2]': 'About to Sleep',
    r'33': 'Need Attention',
    r'[3-4][4-5]': 'Loyal Customers',
    r'41': 'Promising',
    r'51': 'New Customers',
    r'[4-5][2-3]': 'Potential Loyalists',
    r'5[4-5]': 'Champions'
}
```

In [62]:

```
rfm['Segment'] = rfm['RecencyScore'].astype(str) + rfm['FrequencyScore'].astype(str)
rfm['Segment'] = rfm['Segment'].replace(seg_map, regex=True)
rfm.head()
```

Out[62]:

Customer ID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryScore	F
12346	324	1	77184	1	1	5	
12347	1	7	4310	5	5	5	
12348	74	4	1797	2	4	4	
12349	17	1	1758	4	1	4	
12350	309	1	334	1	1	2	

In [63]:

```
rfm[["Segment", "Recency","Frequency","Monetary"]].groupby("Segment").agg(["mean","count"])
```

Out[63]:

	Recency		Frequency		Monetary	
	mean	count	mean	count	mean	count
Segment						
About to Sleep	51	352	1	352	472	352
At Risk	152	593	3	593	1085	593
Can't Loose	131	63	8	63	2796	63
Champions	4	633	12	633	6858	633
Hibernating	216	1071	1	1071	489	1071
Loyal Customers	32	819	6	819	2864	819
Need Attention	50	187	2	187	898	187
New Customers	5	42	1	42	388	42
Potential Loyalists	15	484	2	484	1041	484
Promising	21	95	1	95	291	95

In [64]:

```
#Finding customers i a group

rfm[rfm["Segment"] == "Need Attention"].head()
```

Out[64]:

Customer ID	Recency	Frequency	Monetary	RecencyScore	FrequencyScore	MonetaryScore	F
12360	51	3	2662	3	3	5	
12372	70	3	1298	3	3	4	
12413	65	3	758	3	3	3	
12456	43	3	3181	3	3	5	
12536	42	3	12602	3	3	5	

In [65]:

```
rfm[rfm["Segment"] == "Need Attention"].index
```

Out[65]:

```
Int64Index([12360, 12372, 12413, 12456, 12536, 12577, 12759, 12808, 12879,
            12885,
            ...
            17928, 17934, 17999, 18147, 18160, 18205, 18228, 18252, 18261,
            18270],
           dtype='int64', name='Customer ID', length=187)
```

In [66]:

```
# na_df  Need Attention DataFrame
na_df = pd.DataFrame()
na_df["NeedAttentionID"] = rfm[rfm["Segment"] == "Need Attention"].index
```

In [67]:

```
na_df.head()
```

Out[67]:

	NeedAttentionID
0	12360
1	12372
2	12413
3	12456
4	12536

In [68]:

```
na_df.to_csv("need_attention.csv")
```

We can list any group of costumer in csv or excel format. We can determine new strategies according to customer groups

In [69]:

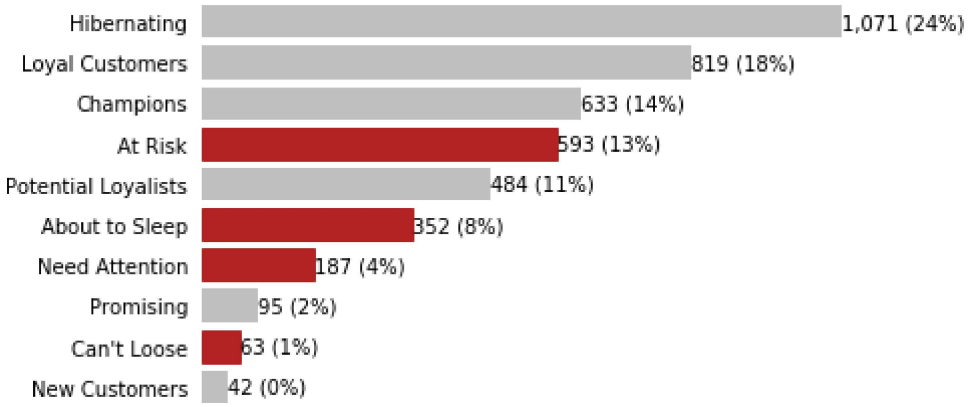
```
# The numbers and pertentage of the customers in each segment
segments_counts = rfm['Segment'].value_counts().sort_values(ascending=True)

fig, ax = plt.subplots()

bars = ax.barh(range(len(segments_counts)),
               segments_counts,
               color='silver')
ax.set_frame_on(False)
ax.tick_params(left=False,
               bottom=False,
               labelbottom=False)
ax.set_yticks(range(len(segments_counts)))
ax.set_yticklabels(segments_counts.index)

for i, bar in enumerate(bars):
        value = bar.get_width()
        if segments_counts.index[i] in ['At Risk', 'Can\'t Loose', 'About to Sleep', 'N
eed Attention',]:
            bar.set_color('firebrick')
        ax.text(value,
                bar.get_y() + bar.get_height()/2,
                '{:,} ({:}%)'.format(int(value),
                                     int(value*100/segments_counts.sum())),
                va='center',
                ha='left'
               )

plt.show()
```



Based on our RFM analysis results we can work on different marketing techniques to retain customers who are in the segments of "at risk, about the sleep, need attention, and can't loose". Besides we need to work on our other customers to increase our sales.