PGP-Retail

Problem Statement:

It is a critical requirement for business to understand the value derived from a customer. RFM is a method used for analyzing customer value. Customer segmentation is the practice of segregating the customer base into groups of individuals based on some common characteristics such as age, gender, interests, and spending habits Perform customer segmentation using RFM analysis. The resulting segments can be ordered from most valuable (highest recency, frequency, and value) to least valuable (lowest recency, frequency, and value).

Project Task: Week 1

Data Cleaning:

1. Perform a preliminary data inspection and data cleaning.

a. Check for missing data and formulate an apt strategy to treat them.

b. Remove duplicate data records.

c. Perform descriptive analytics on the given data.

Data Transformation:

1. Perform cohort analysis (a cohort is a group of subjects that share a defining characteristic). Observe how a cohort behaves across time and compare it to other cohorts.

a. Create month cohorts and analyze active customers for each cohort.

b. Analyze the retention rate of customers.

In [1]:

```
#Importing Required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
#Importing train dataset
data_train=pd.read_excel('train.xlsx',parse_dates=['InvoiceDate'],infer_datetime_format=True)
data_train.head()
```

Out[2]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	558904	22292	HANGING CHICK YELLOW DECORATION	1	2011-07-04 16:18:00	1.25	NaN	United Kingdom
1	556072	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	8	2011-06-08 14:57:00	3.75	16126.0	United Kingdom
2	551739	21559	STRAWBERRY LUNCH BOX WITH CUTLERY	2	2011-05-04 10:58:00	2.55	18118.0	United Kingdom
3	541658	21988	PACK OF 6 SKULL PAPER PLATES	1	2011-01-20 12:16:00	0.85	15529.0	United Kingdom
4	538364	85099C	JUMBO BAG BAROQUE BLACK WHITE	10	2010-12-10 17:26:00	1.95	14448.0	United Kingdom

In [3]:

```
data_train.shape
```

```
data_train.shape
```

Out[3]:

```
(379336, 8)
```

In [4]:

```
#Importing test dataset
data_test=pd.read_excel('test.xlsx',parse_dates=['InvoiceDate'],infer_datetime_format=True)
data_test.head()
```

Out[4]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	562955	84660c	PINK STITCHED WALL CLOCK	3	2011-08-11 10:14:00	7.46	NaN	United Kingdom
1	548451	22707	WRAP MONSTER FUN	50	2011-03-31 11:25:00	0.42	17365.0	United Kingdom
2	568180	22534	MAGIC DRAWING SLATE SPACEBOY	12	2011-09-25 13:42:00	0.42	15429.0	United Kingdom
3	577078	47369B	BLUE GREEN EMBROIDERY COSMETIC BAG	1	2011-11-17 15:17:00	5.79	NaN	United Kingdom
4	C569891	22720	SET OF 3 CAKE TINS PANTRY DESIGN	-2	2011-10-06 15:46:00	4.95	13924.0	United Kingdom

In [5]:

```
data_test.shape
```

Out[5]:

```
(162573, 8)
```

In [6]:

```
#Appending train and test datasets into one:
data_final=pd.concat([data_train,data_test])
data_final.head()
```

Out[6]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	558904	22292	HANGING CHICK YELLOW DECORATION	1	2011-07-04 16:18:00	1.25	NaN	United Kingdom
1	556072	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	8	2011-06-08 14:57:00	3.75	16126.0	United Kingdom
2	551739	21559	STRAWBERRY LUNCH BOX WITH CUTLERY	2	2011-05-04 10:58:00	2.55	18118.0	United Kingdom
3	541658	21988	PACK OF 6 SKULL PAPER PLATES	1	2011-01-20 12:16:00	0.85	15529.0	United Kingdom
4	538364	85099C	JUMBO BAG BAROQUE BLACK WHITE	10	2010-12-10 17:26:00	1.95	14448.0	United Kingdom

In [7]:

```
#Size of the dataset
data_final.shape
```

Out[7]:

```
(541909, 8)
```

In [8]:

```
#Viewing the top few rows
```

```
#viewing the top few rows
data_final.head(n=10)
```

Out[8]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	558904	22292	HANGING CHICK YELLOW DECORATION	1	2011-07-04 16:18:00	1.25	NaN	United Kingdom
1	556072	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	8	2011-06-08 14:57:00	3.75	16126.0	United Kingdom
2	551739	21559	STRAWBERRY LUNCH BOX WITH CUTLERY	2	2011-05-04 10:58:00	2.55	18118.0	United Kingdom
3	541658	21988	PACK OF 6 SKULL PAPER PLATES	1	2011-01-20 12:16:00	0.85	15529.0	United Kingdom
4	538364	85099C	JUMBO BAG BAROQUE BLACK WHITE	10	2010-12-10 17:26:00	1.95	14448.0	United Kingdom
5	552306	84789	ENCHANTED BIRD PLANT CAGE	4	2011-05-08 15:20:00	3.75	13911.0	United Kingdom
6	561513	72351B	SET/6 PINK BUTTERFLY T-LIGHTS	1	2011-07-27 15:12:00	4.13	NaN	United Kingdom
7	566591	23057	BEADED CHANDELIER T-LIGHT HOLDER	4	2011-09-13 14:53:00	4.95	16036.0	United Kingdom
8	564516	84970L	SINGLE HEART ZINC T-LIGHT HOLDER	3	2011-08-25 14:45:00	2.08	NaN	United Kingdom
9	573582	23082	SET 6 PAPER TABLE LANTERN HEARTS	6	2011-10-31 14:23:00	3.75	16633.0	United Kingdom

In [9]:

```
type(data_final)
```

Out[9]:

```
pandas.core.frame.DataFrame
```

In [10]:

```
#As invoice number starting with "C" is cancellation and not required for our model..henceforth re
moving those rows
data_final[data_final["InvoiceNo"].str.startswith('C',na=False)]
```

Out[10]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
415	C549269	75049L	LARGE CIRCULAR MIRROR MOBILE	-9	2011-04-07 12:47:00	0.85	16701.0	United Kingdom
487	C572109	23064	CINDERELLA CHANDELIER	-1	2011-10-20 18:24:00	49.95	13350.0	United Kingdom
613	C537860	22180	RETROSPOT LAMP	-1	2010-12-08 16:15:00	9.95	16252.0	United Kingdom
834	C560540	23240	SET OF 4 KNICK KNACK TINS DOILEY	-1	2011-07-19 12:26:00	4.15	12415.0	Australia
874	C542910	20726	LUNCH BAG WOODLAND	-1	2011-02-01 15:38:00	1.45	17511.0	United Kingdom
...	...	...	...	...	...	...	...	...
162302	C558712	21735	TWO DOOR CURIO CABINET	-1	2011-07-01 13:06:00	12.75	17338.0	United Kingdom
162334	C550780	84507C	BLUE CIRCLES DESIGN MONKEY DOLL	-1	2011-04-20 13:39:00	2.55	17211.0	United Kingdom
162344	C553031	21533	RETROSPOT LARGE MILK JUG	-3	2011-05-12 19:43:00	4.95	13908.0	United Kingdom
162421	C542910	85123A	WHITE HANGING HEART T-LIGHT HOLDER	-1	2011-02-01 15:38:00	2.55	17511.0	United Kingdom
162458	C543368	22941	CHRISTMAS LIGHTS 10 REINDEER	-4	2011-02-07 14:46:00	8.50	18245.0	United Kingdom

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country

9288 rows × 8 columns

In [11]:

```
#Removing the rows starting with 'C'
data_final=data_final[~data_final["InvoiceNo"].str.startswith('C',na=False)]
data_final.head()
```

Out[11]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	558904	22292	HANGING CHICK YELLOW DECORATION	1	2011-07-04 16:18:00	1.25	NaN	United Kingdom
1	556072	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	8	2011-06-08 14:57:00	3.75	16126.0	United Kingdom
2	551739	21559	STRAWBERRY LUNCH BOX WITH CUTLERY	2	2011-05-04 10:58:00	2.55	18118.0	United Kingdom
3	541658	21988	PACK OF 6 SKULL PAPER PLATES	1	2011-01-20 12:16:00	0.85	15529.0	United Kingdom
4	538364	85099C	JUMBO BAG BAROQUE BLACK WHITE	10	2010-12-10 17:26:00	1.95	14448.0	United Kingdom

checking for duplicated values using below syntax

In [12]:

```
data_final.duplicated().sum()
```

Out[12]:

```
5231
```

There are total 5231 duplicated rows.let's look at that rows using below syntax

In [13]:

```
data_final.loc[data_final.duplicated(keep='first'),:]
```

Out[13]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
2878	575117	21098	CHRISTMAS TOILET ROLL	1	2011-11-08 14:22:00	1.25	12748.0	United Kingdom
5729	542107	21755	LOVE BUILDING BLOCK WORD	1	2011-01-25 13:38:00	5.95	16222.0	United Kingdom
7615	577778	21733	RED HANGING HEART T-LIGHT HOLDER	1	2011-11-21 16:10:00	2.95	16549.0	United Kingdom
8997	578781	22988	SOLDIERS EGG CUP	1	2011-11-25 11:54:00	1.25	15872.0	United Kingdom
14797	575583	20893	HANGING BAUBLE T-LIGHT HOLDER SMALL	1	2011-11-10 11:55:00	2.55	14456.0	United Kingdom
...	...	...	...	...	...	...	...	...
162381	541265	20726	LUNCH BAG WOODLAND	1	2011-01-16 16:23:00	1.65	17609.0	United Kingdom
162399	564727	22659	LUNCH BOX I LOVE LONDON	1	2011-08-28 12:31:00	1.95	16686.0	United Kingdom
162424	570818	22553	PLASTERS IN TIN SKULLS	1	2011-10-12 12:47:00	1.65	17841.0	United Kingdom
162437	570410	22505	MEMO BOARD COTTAGE DESIGN	1	2011-10-10 13:04:00	4.95	16776.0	United Kingdom
162524	570223	22496	SET OF 2 ROUND TINS DUTCH CHEESE	1	2011-10-09 13:11:00	2.95	15787.0	United Kingdom

5231 rows × 8 columns

In [14]:

```
#Removing the duplicate values
data_retail=data_final.drop_duplicates(keep='first')
```

In [15]:

```
data_retail.shape
```

Out[15]:

```
(527390, 8)
```

Initially size of dataset was (541909, 8) now its reduced to (527390, 8) after removing the duplicated values of 5231

In [16]:

```
#Checking for null values
data_retail.isnull().sum()
```

Out[16]:

```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     134658
Country             0
dtype: int64
```

There are 1454 null values in Description column and 134658 null values in CustomerID

In [17]:

```
#checking all null values in CustomerID column:
data_retail[data_retail.CustomerID.isnull()]
```

Out[17]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	558904	22292	HANGING CHICK YELLOW DECORATION	1	2011-07-04 16:18:00	1.25	NaN	United Kingdom
6	561513	72351B	SET/6 PINK BUTTERFLY T-LIGHTS	1	2011-07-27 15:12:00	4.13	NaN	United Kingdom
8	564516	84970l	SINGLE HEART ZINC T-LIGHT HOLDER	3	2011-08-25 14:45:00	2.08	NaN	United Kingdom
11	571931	21286	RETROSPOT CANDLE LARGE	2	2011-10-19 16:59:00	4.96	NaN	United Kingdom
21	547788	21398	RED POLKADOT COFFEE MUG	6	2011-03-25 12:00:00	1.63	NaN	United Kingdom
...	...	...	...	...	...	...	...	...
162550	545464	90200A	PURPLE SWEETHEART BRACELET	1	2011-03-03 09:10:00	4.15	NaN	United Kingdom
162555	553766	85150	LADIES & GENTLEMEN METAL SIGN	1	2011-05-19 10:44:00	4.96	NaN	United Kingdom
162558	541592	21896	POTTING SHED TWINE	1	2011-01-19 15:08:00	4.13	NaN	United Kingdom
162565	580367	22865	HAND WARMER OWL DESIGN	3	2011-12-02 16:39:00	4.13	NaN	United Kingdom
162567	579187	22746	POPPY'S PLAYHOUSE LIVINGROOM	2	2011-11-28 15:31:00	4.13	NaN	United Kingdom

134658 rows × 8 columns

In [18]:

```
#checking all null values in Description column:
data_retail[data_retail.Description.isnull()]
```

Out[18]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1099	544321	16033	NaN	120	2011-02-17 15:42:00	0.0	NaN	United Kingdom
1292	542505	79063D	NaN	2560	2011-01-28 12:04:00	0.0	NaN	United Kingdom
1894	540696	84562A	NaN	1	2011-01-11 09:14:00	0.0	NaN	United Kingdom
2099	542394	84452	NaN	65	2011-01-27 15:11:00	0.0	NaN	United Kingdom
2630	551430	20966	NaN	-3	2011-04-28 15:06:00	0.0	NaN	United Kingdom
...	...	...	...	...	...	...	...	...
161315	571010	22848	NaN	-16	2011-10-13 12:13:00	0.0	NaN	United Kingdom
161799	538042	21763	NaN	-4	2010-12-09 13:10:00	0.0	NaN	United Kingdom
161869	537439	37474	NaN	1	2010-12-06 17:01:00	0.0	NaN	United Kingdom
161979	547331	17013D	NaN	-20	2011-03-22 11:41:00	0.0	NaN	United Kingdom
162477	551668	22444	NaN	-10	2011-05-03 12:37:00	0.0	NaN	United Kingdom

1454 rows × 8 columns

In [19]:

```
# By dictionary,treating the null values as below:
retail=data_retail.fillna({'Description':'no data'})
```

In [20]:

```
retail.isnull().sum()
```

Out[20]:

```
InvoiceNo           0
StockCode           0
Description         0
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     134658
Country             0
dtype: int64
```

In [21]:

```
#Removing all null values from CustomerID column
retail.dropna(subset=['CustomerID'],inplace=True)
```

In [22]:

```
retail.isnull().sum() #No null values
```

Out[22]:

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

In [23]:

```
retail.head()
```

Out[23]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	556072	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	8	2011-06-08 14:57:00	3.75	16126.0	United Kingdom
2	551739	21559	STRAWBERRY LUNCH BOX WITH CUTLERY	2	2011-05-04 10:58:00	2.55	18118.0	United Kingdom
3	541658	21988	PACK OF 6 SKULL PAPER PLATES	1	2011-01-20 12:16:00	0.85	15529.0	United Kingdom
4	538364	85099C	JUMBO BAG BAROQUE BLACK WHITE	10	2010-12-10 17:26:00	1.95	14448.0	United Kingdom
5	552306	84789	ENCHANTED BIRD PLANT CAGE	4	2011-05-08 15:20:00	3.75	13911.0	United Kingdom

In [24]:

```
retail.shape #Shape of the dataset
```

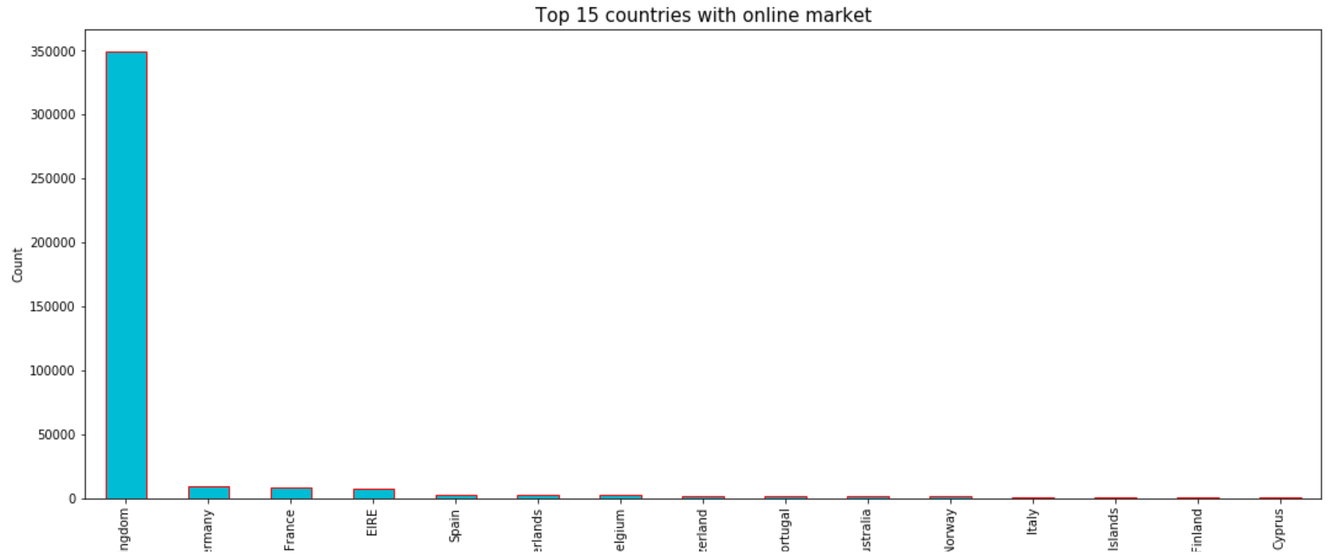
Out[24]:

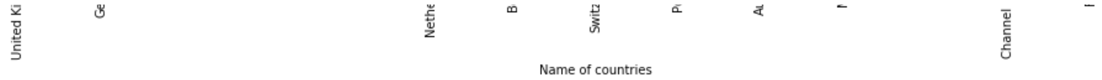
```
(392732, 8)
```

Performing descriptive analytics on given data

In [25]:

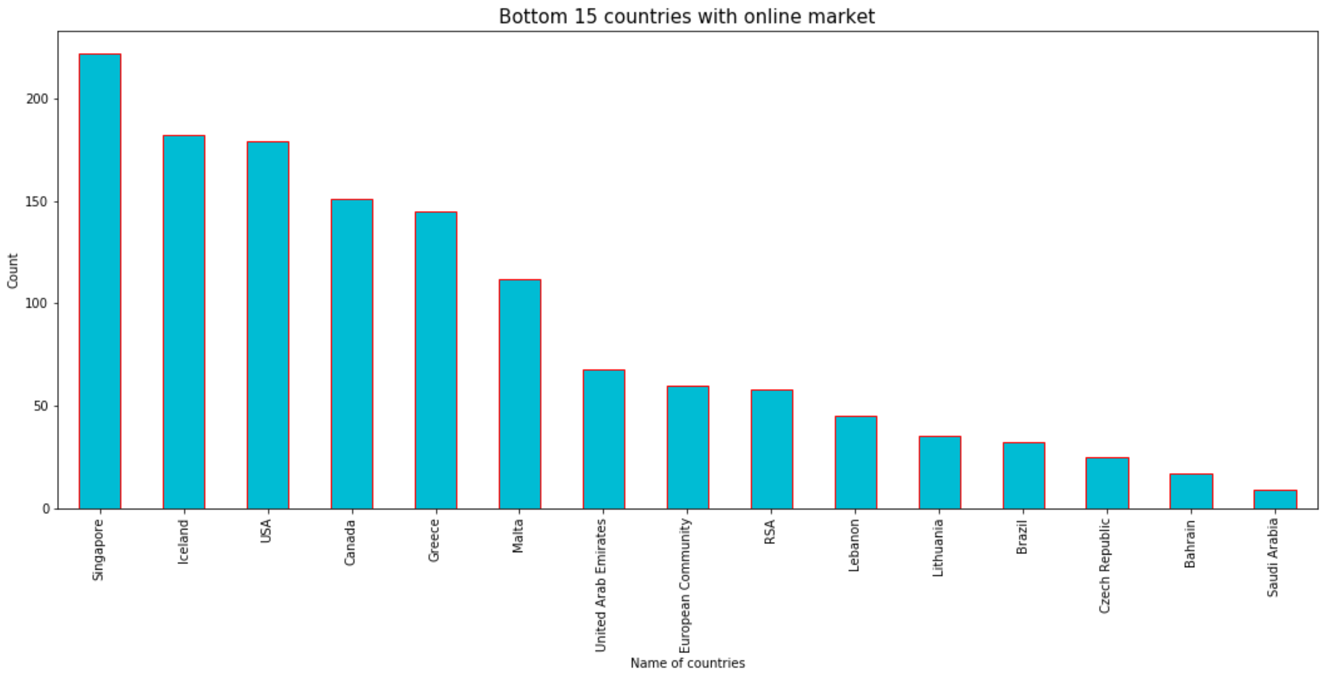
```
#Checking the different values of country column in the dataset
retail['Country'].value_counts().head(15).plot.bar(figsize =
(18,7),edgecolor='red',color='#00bcd4')
plt.title("Top 15 countries with online market",fontsize=15)
plt.xlabel("Name of countries")
plt.ylabel("Count")
plt.show()
```





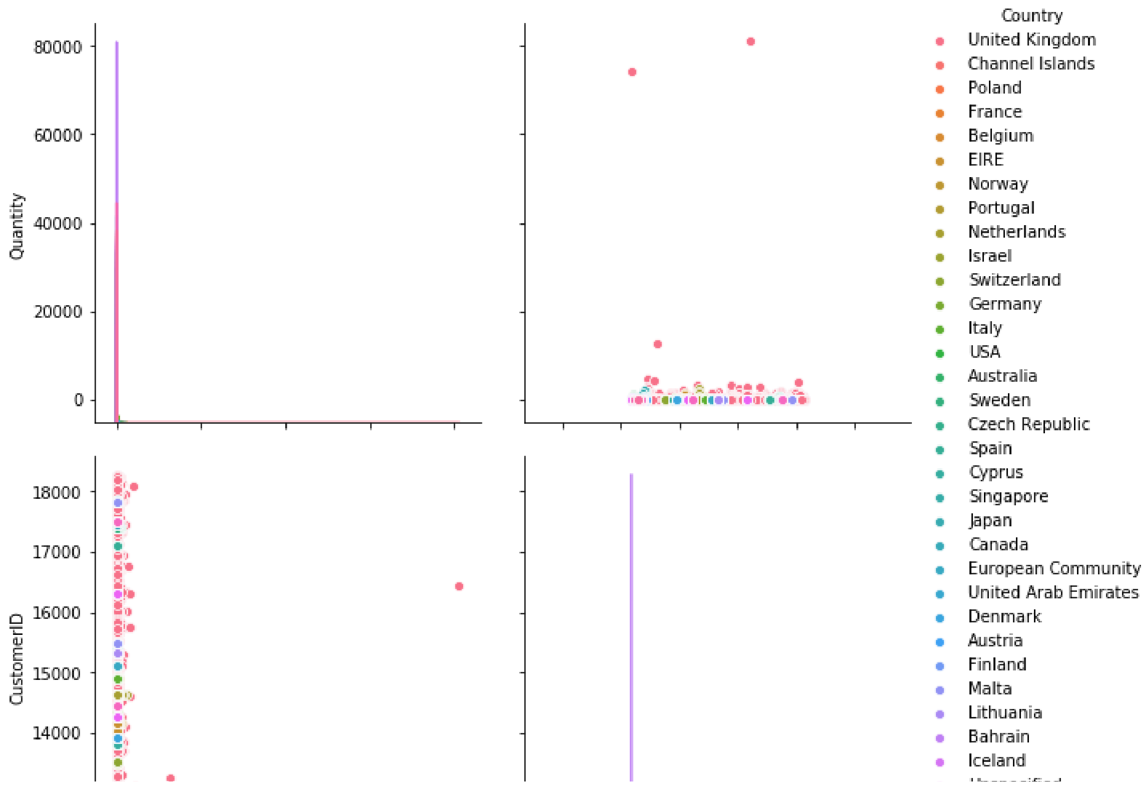
In [26]:

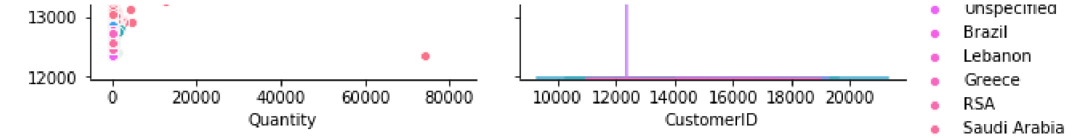
```
#Checking the different values of country column in the dataset
retail['Country'].value_counts().tail(15).plot.bar(figsize =
(18,7),edgecolor='red',color='#00bcd4')
plt.title("Bottom 15 countries with online market",fontsize=15)
plt.xlabel("Name of countries")
plt.ylabel("Count")
plt.show()
```



In [27]:

```
#Pairplot showing the relationship between quantity and customerid
ax=sns.pairplot(data=retail,vars=['Quantity','CustomerID'],hue='Country',height=4,palette='husl')
```





In [28]:

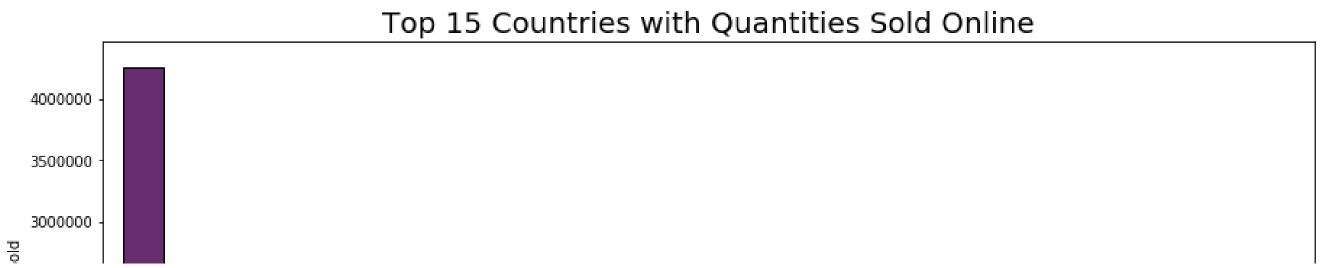
```
#Checking the maximum quantity of products sold from each country
retail['Quantity'].groupby(retail['Country']).agg('sum').sort_values(ascending=True)
```

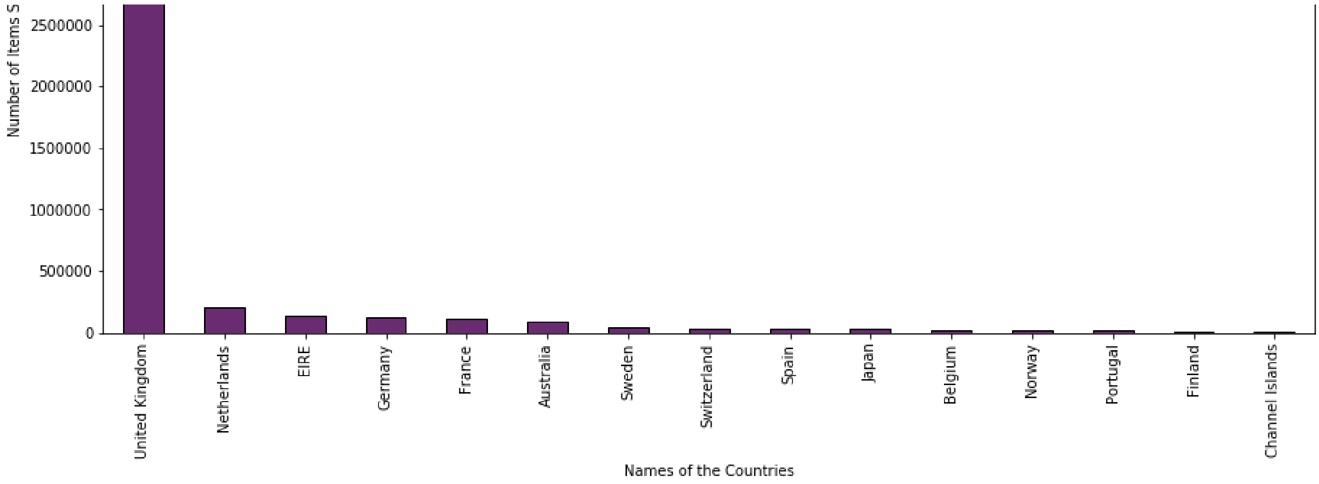
Out[28]:

```
Country
Saudi Arabia                 80
Bahrain                     260
RSA                         352
Brazil                      356
Lebanon                     386
European Community          499
Lithuania                   652
Czech Republic              671
Malta                       970
United Arab Emirates        982
Greece                     1557
Unspecified                1785
USA                        2458
Iceland                    2458
Canada                     2763
Poland                     3684
Israel                     4043
Austria                    4881
Singapore                  5241
Cyprus                     6340
Italy                      8112
Denmark                    8235
Channel Islands            9485
Finland                   10704
Portugal                  16095
Norway                    19338
Belgium                   23237
Japan                     26016
Spain                     27944
Switzerland               30083
Sweden                    36078
Australia                 84199
France                   111429
Germany                  119156
EIRE                     140383
Netherlands              200937
United Kingdom          4254037
Name: Quantity, dtype: int64
```

In [29]:

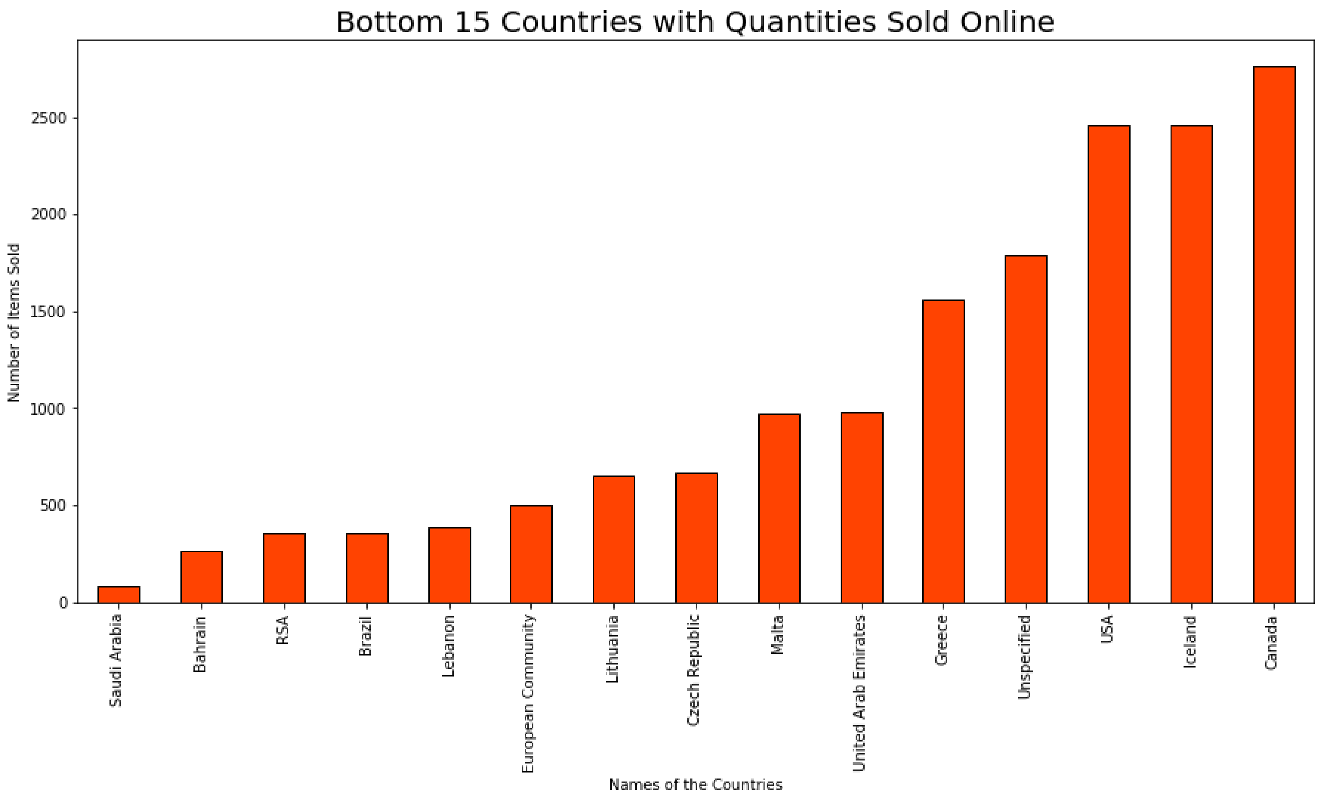
```
#Top 15 countries in terms of quantities according to countries:
retail['Quantity'].groupby(retail['Country']).agg('sum').sort_values(ascending=False).head(15).plot
.bar(figsize = (15, 7),edgecolor='black',color='#6a2c70')
plt.title('Top 15 Countries with Quantities Sold Online', fontsize = 20)
plt.xlabel('Names of the Countries')
plt.ylabel('Number of Items Sold')
plt.show()
```





In [30]:

```
#Bottom 15 countries in terms of quantities according to countries:
retail['Quantity'].groupby(retail['Country']).agg('sum').sort_values(ascending=True).head(15).plot.
bar(figsize = (15, 7),edgecolor='black',color='#ff4301')
plt.title('Bottom 15 Countries with Quantities Sold Online', fontsize = 20)
plt.xlabel('Names of the Countries')
plt.ylabel('Number of Items Sold')
plt.show()
```



In [31]:

```
retail.head()
```

Out[31]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	556072	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	8	2011-06-08 14:57:00	3.75	16126.0	United Kingdom
2	551739	21559	STRAWBERRY LUNCH BOX WITH CUTLERY	2	2011-05-04 10:58:00	2.55	18118.0	United Kingdom
					2011-01-20			United

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
3	541658	21988	PACK OF 6 SKULL PAPER PLATES	1	[illegible]	0.85	15529.0	United Kingdom
4	538364	85099C	JUMBO BAG BAROQUE BLACK WHITE	10	2010-12-10 17:26:00	1.95	14448.0	United Kingdom
5	552306	84789	ENCHANTED BIRD PLANT CAGE	4	2011-05-08 15:20:00	3.75	13911.0	United Kingdom

In [32]:

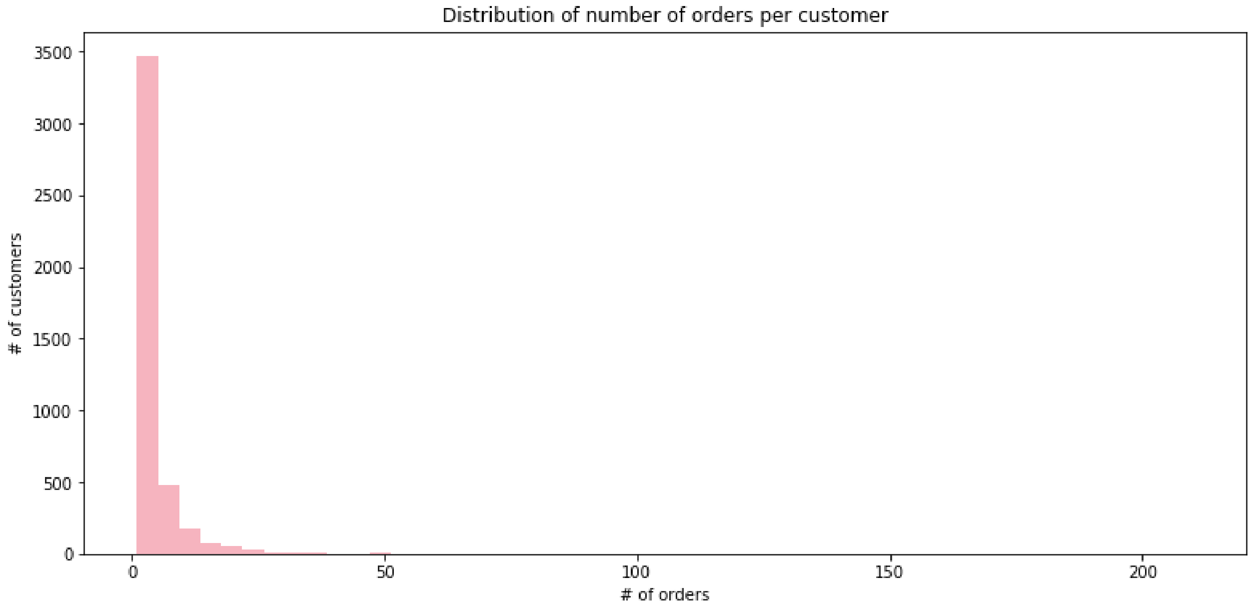
```
#Lets see how many orders placed by each customer
n_orders = retail.groupby(['CustomerID'])['InvoiceNo'].nunique()
mult_orders_perc = np.sum(n_orders > 1) / retail['CustomerID'].nunique()
print(f'{100 * mult_orders_perc:.2f}% of customers ordered more than once.')
```

```
65.57% of customers ordered more than once.
```

Using the code above, we can state that 65.57% of customers ordered more than once.

In [33]:

```
ax = sns.distplot(n_orders, kde=False, hist=True,color='#e94560')
fig=plt.gcf()
fig.set_size_inches(13,6)
ax.set(title='Distribution of number of orders per customer',
       xlabel='# of orders',
       ylabel='# of customers');
```



There are some infrequent cases of customers, who ordered more than 50 times.

Cohort Analysis

The dataset we are using for this example does not contain the customer sign-up date — the date when they registered with the retailer. That is why we assume that the cohort they belong to is based on the first purchase date. A possible downside of this approach is that the dataset does not contain the past data, and what we already see in this snapshot (between 01/12/2010 and 09/12/2011) includes recurring clients. In other words, the first purchase we see in this dataset might not be the actual first purchase of a given client. However, there is no way to account for this without having access to the entire historical dataset of the retailer.

As the first step, we keep only the relevant columns and drop duplicated values — one order (indicated by InvoiceNo) can contain multiple items (indicated by StockCode).

As the second step, we create the cohort and order_month variables. The first one indicates the monthly cohort based on the first purchase date (calculated per customer). The latter one is the truncated month of the purchase date.

In [34]:

```
retail['order_month'] = retail['InvoiceDate'].dt.to_period('M')
retail['cohort'] = retail.groupby('CustomerID')['InvoiceDate'] \
                  .transform('min') \
                  .dt.to_period('M')
```

In [35]:

```
retail.head()
```

Out[35]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	order_month	cohort
1	556072	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	8	2011-06-08 14:57:00	3.75	16126.0	United Kingdom	2011-06	2011-02
2	551739	21559	STRAWBERRY LUNCH BOX WITH CUTLERY	2	2011-05-04 10:58:00	2.55	18118.0	United Kingdom	2011-05	2010-12
3	541658	21988	PACK OF 6 SKULL PAPER PLATES	1	2011-01-20 12:16:00	0.85	15529.0	United Kingdom	2011-01	2010-12
4	538364	85099C	JUMBO BAG BAROQUE BLACK WHITE	10	2010-12-10 17:26:00	1.95	14448.0	United Kingdom	2010-12	2010-12
5	552306	84789	ENCHANTED BIRD PLANT CAGE	4	2011-05-08 15:20:00	3.75	13911.0	United Kingdom	2011-05	2011-02

Then, we aggregate the data per cohort and order_month and count the number of unique customers in each group. Additionally, we add the period_number, which indicates the number of periods between the cohort month and the month of the purchase.

In [36]:

```
from operator import attrgetter
```

In [37]:

```
retail_cohort = retail.groupby(['cohort', 'order_month']) \
               .agg(n_customers=('CustomerID', 'nunique')) \
               .reset_index(drop=False)
retail_cohort['period_number'] = (retail_cohort.order_month - retail_cohort.cohort).apply(attrgette
r('n'))
```

In [38]:

```
retail_cohort.head()
```

Out[38]:

	cohort	order_month	n_customers	period_number
0	2010-12	2010-12	885	0
1	2010-12	2011-01	324	1
2	2010-12	2011-02	286	2
3	2010-12	2011-03	340	3
4	2010-12	2011-04	321	4

The next step is to pivot the df_cohort table in a way that each row contains information about a given cohort and each column contains values for a certain period.

In [39]:

```
cohort_pivot = retail_cohort.pivot_table(index = 'cohort',
                                         columns = 'period_number',
                                         values = 'n_customers')
```

To obtain the retention matrix, we need to divide the values each row by the row's first value, which is actually the cohort size — all customers who made their first purchase in the given month.

In [40]:

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

In [41]:

```
import matplotlib.colors as mcolors
```

Lastly, we plot the retention matrix as a heatmap. Additionally, we wanted to include extra information regarding the cohort size. That is why we in fact created two heatmaps, where the one indicating the cohort size is using a white only colormap — no coloring at all.

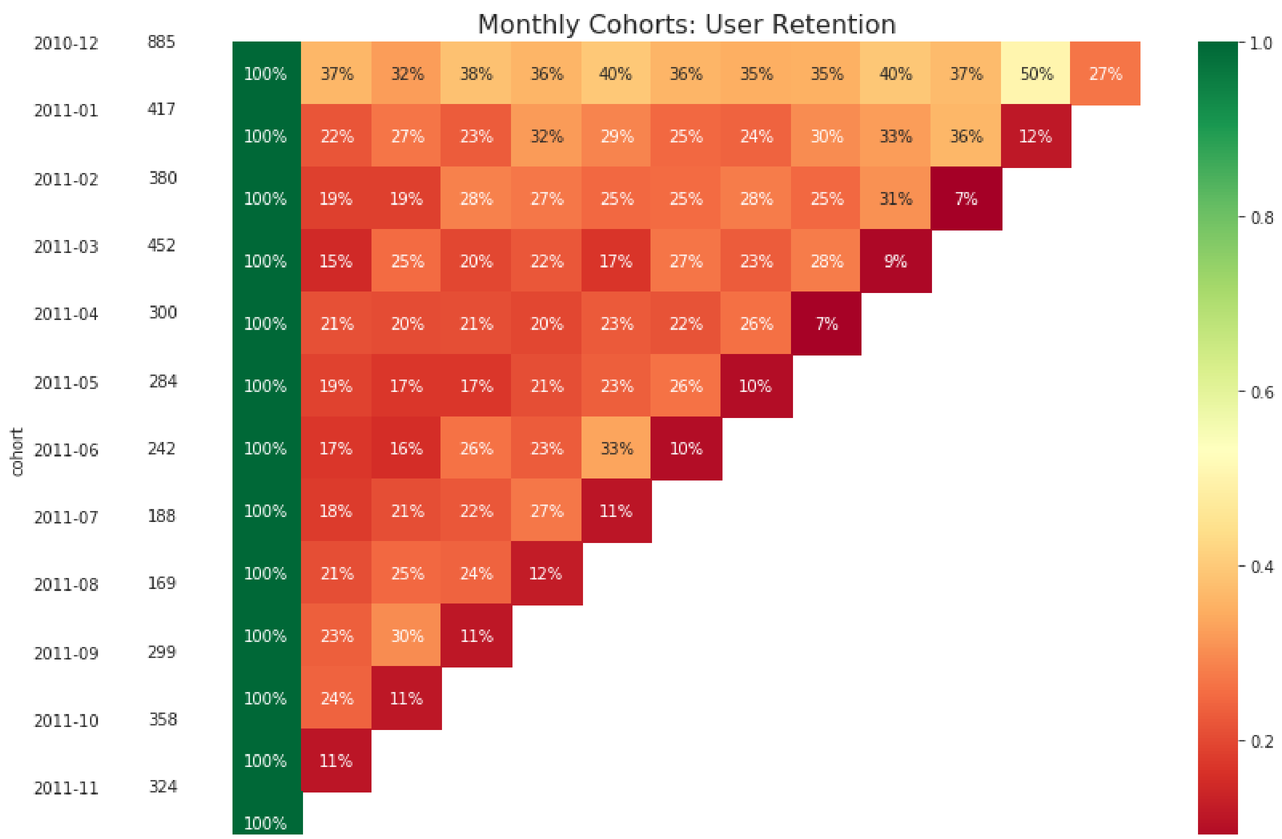
In [42]:

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

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

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

    fig.tight_layout()
```



In the image, we can see that there is a sharp drop-off in the second month (indexed as 1) already, on average around 80% of customers do not make any purchase in the second month. The first cohort (2010–12) seems to be an exception and performs surprisingly well as compared to the other ones. A year after the first purchase, there is a 50% retention. This might be a cohort of dedicated customers, who first joined the platform based on some already-existing connections with the retailer. However, from data alone, that is very hard to accurately explain. Throughout the matrix, we can see fluctuations in retention over time. This might be caused by the characteristics of the business, where clients do periodic purchases, followed by periods of inactivity.

In [43]:

```
### Trying one more time
import datetime as dt
def get_month(x) : return dt.datetime(x.year,x.month,1)
retail['InvoiceMonth'] = retail['InvoiceDate'].apply(get_month)
grouping = retail.groupby('CustomerID')['InvoiceMonth']
retail['CohortMonth'] = grouping.transform('min')
retail.tail()
```

Out[43]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	order_month	cohort	InvoiceMon
162568	574102	22866	HAND WARMER SCOTTY DOG DESIGN	24	2011-11-03 10:27:00	2.10	16128.0	United Kingdom	2011-11	2011-03	2011-11-(
162569	545226	22919	HERB MARKER MINT	12	2011-03-01 09:33:00	0.65	12428.0	Finland	2011-03	2011-03	2011-03-(
162570	573160	22077	6 RIBBONS RUSTIC CHARM	12	2011-10-28 08:58:00	1.95	14359.0	United Kingdom	2011-10	2011-09	2011-10-(
162571	552321	23204	CHARLOTTE BAG APPLES DESIGN	10	2011-05-09 09:15:00	0.85	17049.0	United Kingdom	2011-05	2011-03	2011-05-(
162572	573359	21983	PACK OF 12 BLUE PAISLEY TISSUES	4	2011-10-30 12:48:00	0.39	14178.0	United Kingdom	2011-10	2011-06	2011-10-(

In [44]:

```
def get_month_int (dframe,column):
    year = dframe[column].dt.year
    month = dframe[column].dt.month
    day = dframe[column].dt.day
    return year, month , day

invoice_year,invoice_month,_ = get_month_int(retail,'InvoiceMonth')
cohort_year,cohort_month,_ = get_month_int(retail,'CohortMonth')

year_diff = invoice_year - cohort_year
month_diff = invoice_month - cohort_month

retail['CohortIndex'] = year_diff * 12 + month_diff + 1
```

In [45]:

```
#Count monthly active customers from each cohort
grouping = retail.groupby(['CohortMonth', 'CohortIndex'])
cohort_data = grouping['CustomerID'].apply(pd.Series.nunique)
# Return number of unique elements in the object.
cohort_data = cohort_data.reset_index()
cohort_counts = cohort_data.pivot(index='CohortMonth',columns='CohortIndex',values='CustomerID')
cohort_counts
```

Out[45]:

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	885.0	324.0	286.0	340.0	321.0	352.0	321.0	309.0	313.0	350.0	331.0	445.0	235.0
2011-01-01	417.0	92.0	111.0	96.0	134.0	120.0	103.0	101.0	125.0	136.0	152.0	49.0	NaN
2011-02-01	380.0	71.0	71.0	108.0	103.0	94.0	96.0	106.0	94.0	116.0	26.0	NaN	NaN
2011-03-01	452.0	68.0	114.0	90.0	101.0	76.0	121.0	104.0	126.0	39.0	NaN	NaN	NaN
2011-04-01	300.0	64.0	61.0	63.0	59.0	68.0	65.0	78.0	22.0	NaN	NaN	NaN	NaN
2011-05-01	284.0	54.0	49.0	49.0	59.0	66.0	75.0	27.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	242.0	42.0	38.0	64.0	56.0	81.0	23.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	188.0	34.0	39.0	42.0	51.0	21.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	169.0	35.0	42.0	41.0	21.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	299.0	70.0	90.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	358.0	86.0	41.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	324.0	36.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	41.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [46]:

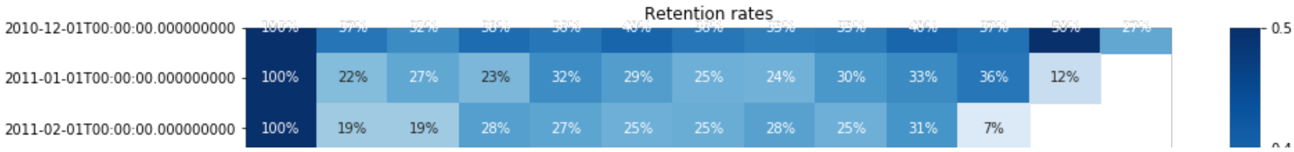
```
# Retention table
cohort_size = cohort_counts.iloc[:,0]
retention = cohort_counts.divide(cohort_size,axis=0) #axis=0 to ensure the divide along the row ax
is
retention.round(3) * 100 #to show the number as percentage
```

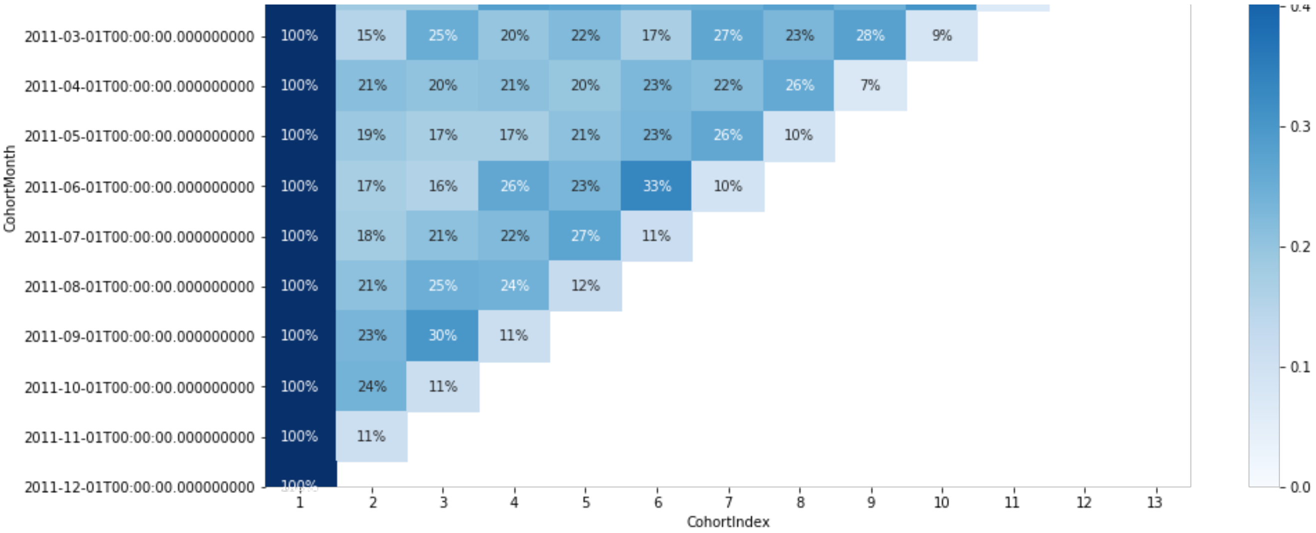
Out[46]:

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	100.0	36.6	32.3	38.4	36.3	39.8	36.3	34.9	35.4	39.5	37.4	50.3	26.6
2011-01-01	100.0	22.1	26.6	23.0	32.1	28.8	24.7	24.2	30.0	32.6	36.5	11.8	NaN
2011-02-01	100.0	18.7	18.7	28.4	27.1	24.7	25.3	27.9	24.7	30.5	6.8	NaN	NaN
2011-03-01	100.0	15.0	25.2	19.9	22.3	16.8	26.8	23.0	27.9	8.6	NaN	NaN	NaN
2011-04-01	100.0	21.3	20.3	21.0	19.7	22.7	21.7	26.0	7.3	NaN	NaN	NaN	NaN
2011-05-01	100.0	19.0	17.3	17.3	20.8	23.2	26.4	9.5	NaN	NaN	NaN	NaN	NaN
2011-06-01	100.0	17.4	15.7	26.4	23.1	33.5	9.5	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	100.0	18.1	20.7	22.3	27.1	11.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	100.0	20.7	24.9	24.3	12.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	100.0	23.4	30.1	11.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	100.0	24.0	11.5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	100.0	11.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	100.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [47]:

```
#Build the heatmap
plt.figure(figsize=(15, 8))
plt.title('Retention rates')
sns.heatmap(data=retention,annot = True,fmt = '.0%',vmin = 0.0,vmax = 0.5,cmap="Blues")
plt.show()
```



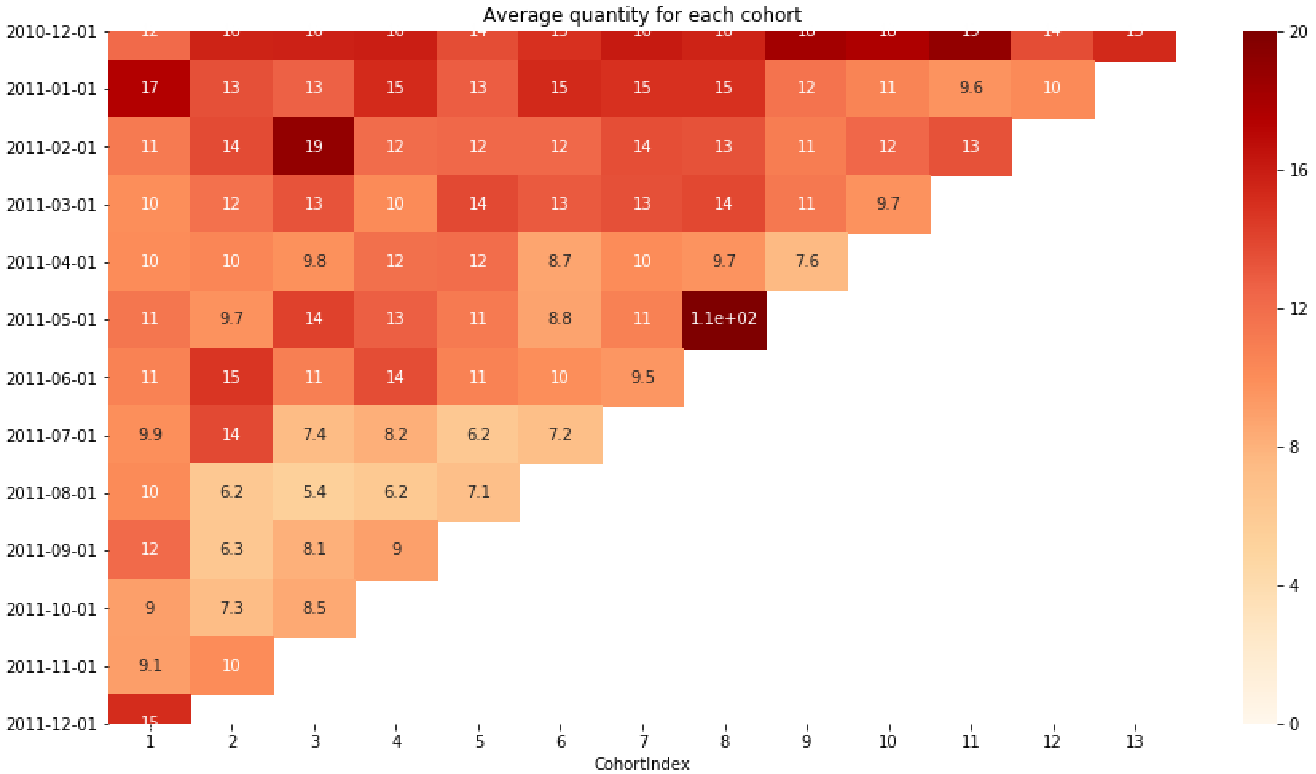


Average quantity for each cohort

In [48]:

```
#Average quantity for each cohort
grouping = retail.groupby(['CohortMonth', 'CohortIndex'])
cohort_data = grouping['Quantity'].mean()
cohort_data = cohort_data.reset_index()
average_quantity = cohort_data.pivot(index='CohortMonth',columns='CohortIndex',values='Quantity')
average_quantity.round(1)
average_quantity.index = average_quantity.index.date

#Build the heatmap
plt.figure(figsize=(15, 8))
plt.title('Average quantity for each cohort')
sns.heatmap(data=average_quantity,annot = True,vmin = 0.0,vmax =20,cmap="OrRd")
plt.show()
```



Project Task: Week 2 Data Modeling :

1. Build a RFM (Recency Frequency Monetary) model. Recency means the number of days since a customer made the last

purchase. Frequency is the number of purchase in a given period. It could be 3 months, 6 months or 1 year. Monetary is the total amount of money a customer spent in that given period. Therefore, big spenders will be differentiated among other customers such as MVP (Minimum Viable Product) or VIP.
2. Calculate RFM metrics.
3. Build RFM Segments. Give recency, frequency, and monetary scores individually by dividing them into quartiles.

b1. Combine three ratings to get a RFM segment (as strings).

b2. Get the RFM score by adding up the three ratings.

b3. Analyze the RFM segments by summarizing them and comment on the findings.

Note: Rate “recency" for customer who has been active more recently higher than the less recent customer, because each company wants its customers to be recent.

Note: Rate “frequency" and “monetary" higher, because the company wants the customer to visit more often and spend more money

The RFM values can be grouped in several ways:

1.Percentiles e.g. quantiles

2.Pareto 80/20 cut

3.Custom - based on business knowledge

We are going to implement percentile-based grouping.

Process of calculating percentiles:

Sort customers based on that metric

Break customers into a pre-defined number of groups of equal size

Assign a label to each group

In [49]:

```
#New Total Sum Column
retail['TotalSum'] = retail['UnitPrice']* retail['Quantity']

#Data preparation steps
print('Min Invoice Date:',retail.InvoiceDate.dt.date.min(),'max Invoice Date:',
       retail.InvoiceDate.dt.date.max())

retail.head(3)
```

```
Min Invoice Date: 2010-12-01 max Invoice Date: 2011-12-09
```

Out[49]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	order_month	cohort	InvoiceMonth
1	556072	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	8	2011-06-08 14:57:00	3.75	16126.0	United Kingdom	2011-06	2011-02	2011-06-01
2	551739	21559	STRAWBERRY LUNCH BOX WITH CUTLERY	2	2011-05-04 10:58:00	2.55	18118.0	United Kingdom	2011-05	2010-12	2011-05-01
3	541658	21988	PACK OF 6 SKULL PAPER PLATES	1	2011-01-20 12:16:00	0.85	15529.0	United Kingdom	2011-01	2010-12	2011-01-01

In [50]:

```
snapshot_date = retail['InvoiceDate'].max() + dt.timedelta(days=1)
snapshot_date
#The last day of purchase in total is 09 DEC, 2011. To calculate the day periods,
#let's set one day after the last one,or
#10 DEC as a snapshot_date. We will cound the diff days with snapshot_date.
```

Out[50]:

Timestamp('2011-12-10 12:50:00')

In [51]:

```
# Calculate RFM metrics
rfm = retail.groupby(['CustomerID']).agg({'InvoiceDate': lambda x : (snapshot_date - x.max()).days,
                                        'InvoiceNo':'count','TotalSum': 'sum'})
#Function Lambdea: it gives the number of days between hypothetical today and the last transaction

#Rename columns
rfm.rename(columns={'InvoiceDate':'Recency','InvoiceNo':'Frequency','TotalSum':'MonetaryValue'}
           ,inplace= True)

#Final RFM values
rfm.head()
```

Out[51]:

	Recency	Frequency	MonetaryValue
CustomerID			
12346.0	326	1	77183.60
12347.0	2	182	4310.00
12348.0	75	31	1797.24
12349.0	19	73	1757.55
12350.0	310	17	334.40

Note That :

We will rate "Recency" customer who have been active more recently better than the less recent customer,because each company wants its customers to be recent We will rate "Frequency" and "Monetary Value" higher label because we want Customer to spend more money and visit more often(that is different order than recency).

In [52]:

```
#Building RFM segments
r_labels =range(4,0,-1)
f_labels=range(1,5)
m_labels=range(1,5)
r_quartiles = pd.qcut(rfm['Recency'], q=4, labels = r_labels)
f_quartiles = pd.qcut(rfm['Frequency'],q=4, labels = f_labels)
m_quartiles = pd.qcut(rfm['MonetaryValue'],q=4,labels = m_labels)
rfm = rfm.assign(R=r_quartiles,F=f_quartiles,M=m_quartiles)

# Build RFM Segment and RFM Score
def add_rfm(x) : return str(x['R']) + str(x['F']) + str(x['M'])
rfm['RFM_Segment'] = rfm.apply(add_rfm,axis=1 )
rfm['RFM_Score'] = rfm[['R','F','M']].sum(axis=1)

rfm.head()
```

Out[52]:

	Recency	Frequency	MonetaryValue	R	F	M	RFM_Segment	RFM_Score
CustomerID								
12346.0	326	1	77183.60	1	1	4	114	6.0
12347.0	2	182	4310.00	4	4	4	444	12.0
12348.0	75	31	1797.24	2	2	4	224	8.0
12349.0	19	73	1757.55	3	3	4	334	10.0
12350.0	310	17	334.40	1	1	2	112	4.0

The Result is a Table which has a row for each customer with their RFM

Analyzing RFM Segments

Largest RFM segments It is always the best practice to investigate the size of the segments before you use them for targeting or other business Application.

In [53]:

```
rfm.groupby(['RFM_Segment']).size().sort_values(ascending=False)[:5]
```

Out[53]:

```
RFM_Segment
444    450
111    381
344    217
122    206
211    179
dtype: int64
```

Filtering on RFM segments

In [54]:

```
#Select bottom RFM segment "111" and view top 5 rows
rfm[rfm['RFM_Segment']=='111'].head()
```

Out[54]:

	Recency	Frequency	MonetaryValue	R	F	M	RFM_Segment	RFM_Score
CustomerID								
12353.0	204	4	89.00	1	1	1	111	3.0
12361.0	287	10	189.90	1	1	1	111	3.0
12401.0	303	5	84.30	1	1	1	111	3.0
12402.0	323	11	225.60	1	1	1	111	3.0
12441.0	367	11	173.55	1	1	1	111	3.0

Summary metrics per RFM Score

In [55]:

```
rfm.groupby('RFM_Score').agg({'Recency': 'mean','Frequency': 'mean',
                              'MonetaryValue': ['mean', 'count'] }).round(1)
```

Out[55]:

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
RFM_Score				
3.0	260.7	8.2	157.4	381
4.0	177.2	13.6	240.0	388
5.0	152.9	21.2	366.6	518
6.0	95.9	27.8	820.1	457
7.0	79.4	37.9	757.1	464
8.0	64.1	56.0	987.3	454
9.0	45.9	78.7	1795.1	414
10.0	32.4	110.5	2056.4	426
11.0	21.3	186.9	4062.0	387

	Recency	Frequency	MonetaryValue	
12.0	7.2	367.9	9285.9	450
	mean	mean	mean	count

Use RFM score to group customers into Gold, Silver and Bronze segments

In [56]:

```
def segments(df):
    if df['RFM_Score'] > 9 :
        return 'Gold'
    elif (df['RFM_Score'] > 5) and (df['RFM_Score'] <= 9 ):
        return 'Sliver'
    else:
        return 'Bronze'

rfm['General_Segment'] = rfm.apply(segments,axis=1)

rfm.groupby('General_Segment').agg({'Recency':'mean','Frequency':'mean',
                                    'MonetaryValue':['mean','count']}).round(1)
```

Out[56]:

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
General_Segment				
Bronze	192.2	15.1	266.5	1287
Gold	20.1	225.6	5246.8	1263
Sliver	72.0	49.4	1071.8	1789

Data Pre-Processing for Kmeans Clustering

We must check these Key k-means assumptions before we implement our Kmeans Clustering Mode

Symmetric distribution of variables (not skewed)

Variables with same average values

Variables with same variance

In [57]:

```
rfm_rfm = rfm[['Recency','Frequency','MonetaryValue']]
print(rfm_rfm.describe())
```

```
          Recency    Frequency  MonetaryValue
count  4339.000000  4339.000000    4339.000000
mean     92.518322    90.512100    2048.215924
std     100.009747   225.515328    8984.248352
min       1.000000     1.000000       0.000000
25%      18.000000    17.000000     306.455000
50%      51.000000    41.000000     668.560000
75%     142.000000    98.000000    1660.315000
max     374.000000  7676.000000  280206.020000
```

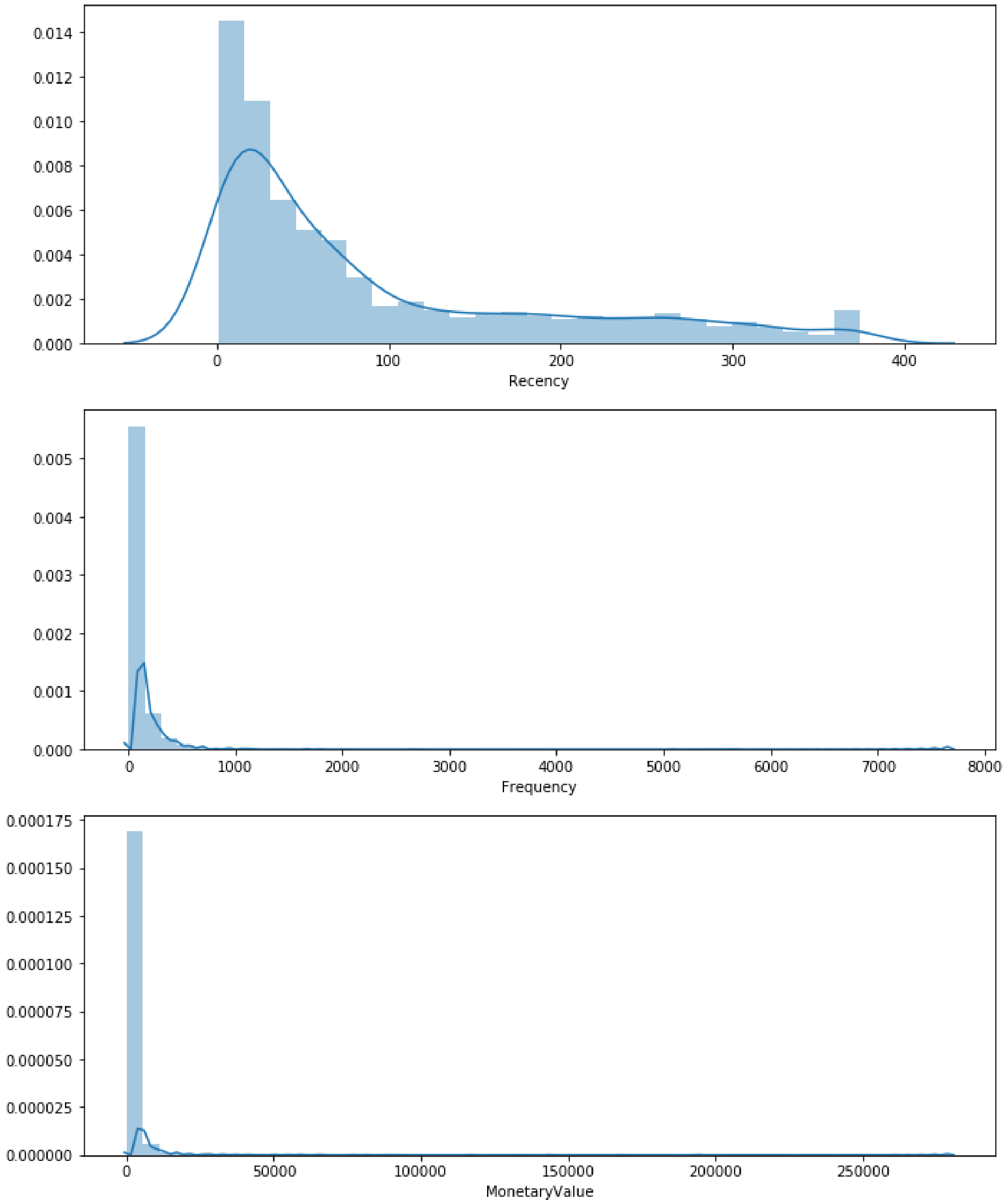
In [58]:

```
rfm['MonetaryValue']=rfm['MonetaryValue'].astype('int64')
```

In [59]:

```
# plot the distribution of RFM values
f,ax = plt.subplots(figsize=(10, 12))
plt.subplot(3, 1, 1); sns.distplot(rfm.Recency, label = 'Recency')
plt.subplot(3, 1, 2); sns.distplot(rfm.Frequency, label = 'Frequency')
```

```
plt.subplot(3, 1, 3); sns.distplot(rfm.MonetaryValue, label = 'Monetary Value')
plt.style.use('fivethirtyeight')
plt.tight_layout()
plt.show()
```



Also, there is another Problem: UnSymmetric distribution of variables (data skewed)

Soluation:Logarithmic transformation (positive values only) will manage skewness

We use these Sequence of structuring pre-processing steps:

Unskew the data - log transformation

Standardize to the same average values

Scale to the same standard deviation

Store as a separate array to be used for clustering

Why the sequence matters?

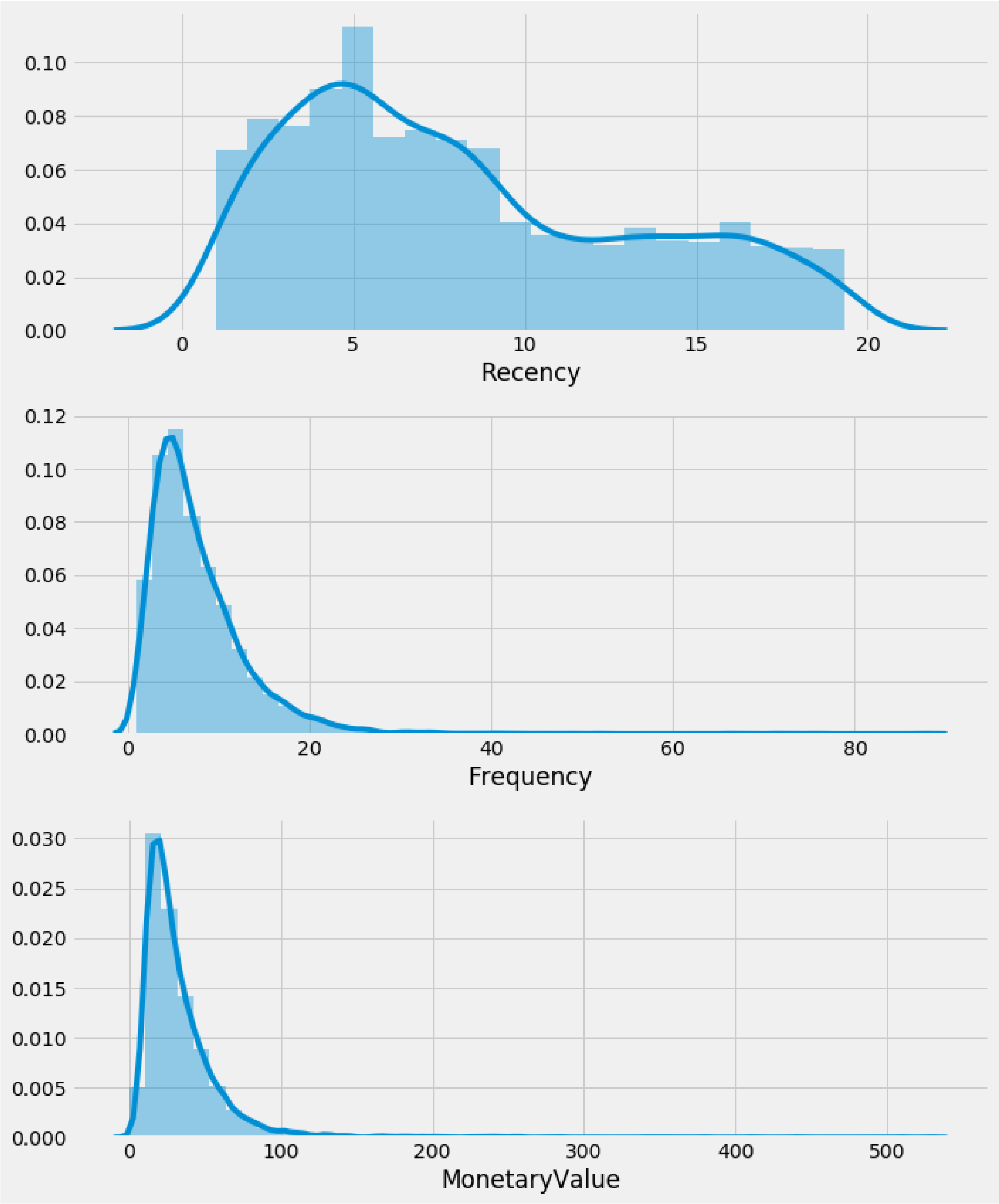
Log transformation only works with positive data

Normalization forces data to have negative values and log will not work

In [60]:

```
#Unskew the data with log transformation
rfm_log = rfm[['Recency', 'Frequency', 'MonetaryValue']].apply(np.sqrt, axis = 1).round(3)
#rfm_log = np.log(rfm_rfm)


# plot the distribution of RFM values
f,ax = plt.subplots(figsize=(10, 12))
plt.subplot(3, 1, 1); sns.distplot(rfm_log.Recency, label = 'Recency')
plt.subplot(3, 1, 2); sns.distplot(rfm_log.Frequency, label = 'Frequency')
plt.subplot(3, 1, 3); sns.distplot(rfm_log.MonetaryValue, label = 'Monetary Value')
plt.style.use('fivethirtyeight')
plt.tight_layout()
plt.show()
```



Project Task: Week 3

Data Modeling :

1. Create clusters using k-means clustering algorithm.

a. Prepare the data for the algorithm. If the data is asymmetrically distributed, manage the skewness with appropriate transformation.

Standardize the data.

b. Decide the optimum number of clusters to be formed.

c. Analyze these clusters and comment on the results.

In [61]:

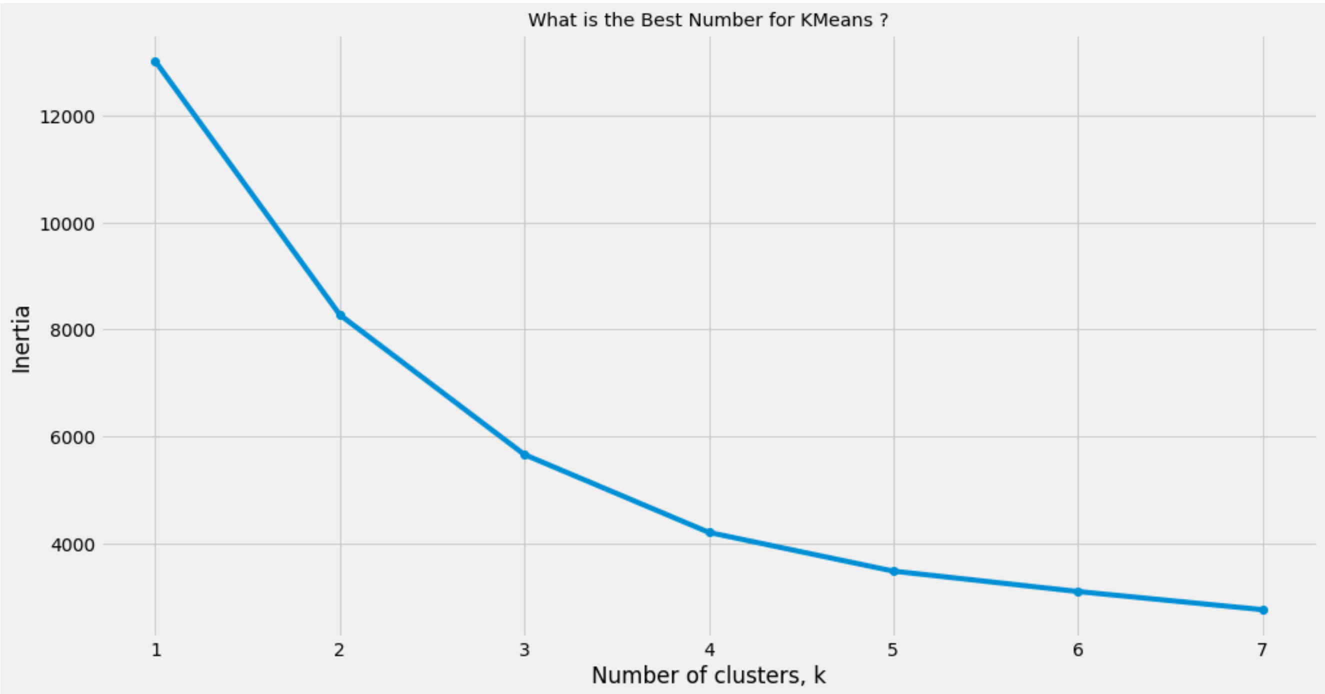
```
#Normalize the variables with StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(rfm_log)
#Store it separately for clustering
rfm_normalized= scaler.transform(rfm_log)
```

In [62]:

```
from sklearn.cluster import KMeans

#First : Get the Best KMeans
ks = range(1,8)
inertias=[]
for k in ks :
    # Create a KMeans clusters
    kc = KMeans(n_clusters=k,random_state=1)
    kc.fit(rfm_normalized)
    inertias.append(kc.inertia_)

# Plot ks vs inertias
f, ax = plt.subplots(figsize=(15, 8))
plt.plot(ks, inertias, '-o')
plt.xlabel('Number of clusters, k')
plt.ylabel('Inertia')
plt.xticks(ks)
plt.style.use('ggplot')
plt.title('What is the Best Number for KMeans ?')
plt.show()
```



Note That: We Choose No.KMeans = 3

In [63]:

```
# clustering
kc = KMeans(n_clusters= 3, random_state=1)
kc.fit(rfm_normalized)

#Create a cluster label column in the original DataFrame
```

```
cluster_labels = kc.labels_

#Calculate average RFM values and size for each cluster:
rfm_rfm_k3 = rfm_rfm.assign(K_Cluster = cluster_labels)

#Calculate average RFM values and sizes for each cluster:
rfm_rfm_k3.groupby('K_Cluster').agg({'Recency': 'mean','Frequency': 'mean',
                                      'MonetaryValue': ['mean', 'count'],}).round(0)
```

Out[63]:

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
K_Cluster				
0	39.0	59.0	1033.0	2449
1	225.0	27.0	489.0	1305
2	19.0	364.0	9778.0	585

Snake plots to understand and compare segments

Market research technique to compare different segments

Visual representation of each segment's attributes

Need to first normalize data (center & scale)

Plot each cluster's average normalized values of each attribute

In [64]:

```
rfm_normalized = pd.DataFrame(rfm_normalized,index=rfm_rfm.index,columns=rfm_rfm.columns)
rfm_normalized['K_Cluster'] = kc.labels_
rfm_normalized['General_Segment'] = rfm['General_Segment']
rfm_normalized.reset_index(inplace = True)

#Melt the data into a long format so RFM values and metric names are stored in 1 column each
rfm_melt = pd.melt(rfm_normalized,id_vars=['CustomerID','General_Segment','K_Cluster'],value_vars=
['Recency', 'Frequency', 'MonetaryValue'],
var_name='Metric',value_name='Value')
rfm_melt.head()
```

Out[64]:

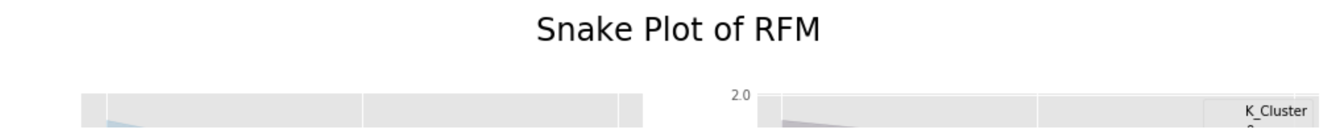
	CustomerID	General_Segment	K_Cluster	Metric	Value
0	12346.0	Sliver	2	Recency	1.964145
1	12347.0	Gold	2	Recency	-1.355554
2	12348.0	Sliver	0	Recency	0.089944
3	12349.0	Gold	0	Recency	-0.768058
4	12350.0	Bronze	1	Recency	1.874773

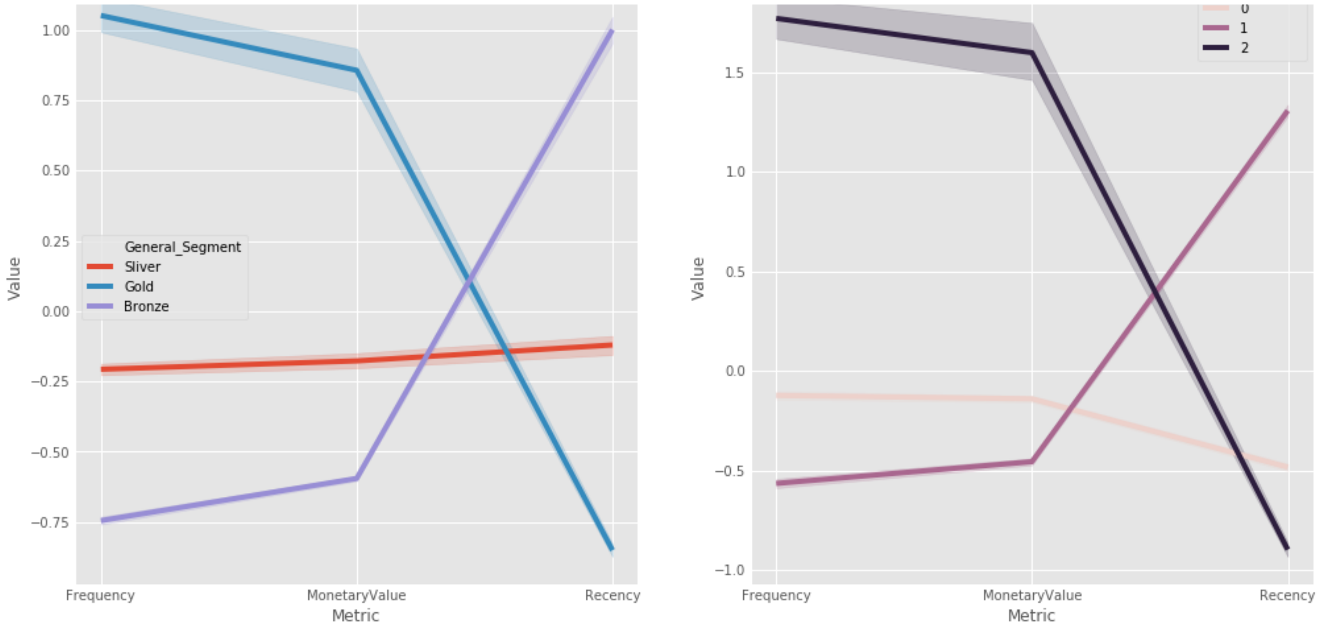
In [65]:

```
f, (ax1, ax2) = plt.subplots(1,2, figsize=(15, 8))
sns.lineplot(x = 'Metric', y = 'Value', hue = 'General_Segment', data = rfm_melt,ax=ax1)

# a snake plot with K-Means
sns.lineplot(x = 'Metric', y = 'Value', hue = 'K_Cluster', data = rfm_melt,ax=ax2)

plt.suptitle("Snake Plot of RFM",fontsize=24) #make title fontsize subtitle
plt.show()
```





Relative importance of segment attributes

Useful technique to identify relative importance of each segment's attribute

1.Calculate average values of each cluster

2.Calculate average values of population

3.Calculate importance score by dividing them and subtracting 1 (ensures 0 is returned when cluster average equals population average)

Let's try again with a heat map. Heat maps are a graphical representation of data where larger values were colored in darker scales and smaller values in lighter scales. We can compare the variance between the groups quite intuitively by colors.

In [66]:

```
# The further a ratio is from 0, the more important that attribute is for a segment relative to th
e total population
cluster_avg = rfm_rfm_k3.groupby(['K_Cluster']).mean()
population_avg = rfm_rfm.mean()
relative_imp = cluster_avg / population_avg - 1
relative_imp.round(2)
```

Out[66]:

	Recency	Frequency	MonetaryValue
K_Cluster			
0	-0.57	-0.35	-0.50
1	1.43	-0.70	-0.76
2	-0.79	3.02	3.77

In [67]:

```
# the mean value in total
total_avg = rfm.iloc[:, 0:3].mean()
# calculate the proportional gap with total mean
cluster_avg = rfm.groupby('General_Segment').mean().iloc[:, 0:3]
prop_rfm = cluster_avg/total_avg - 1
prop_rfm.round(2)
```

Out[67]:

	Recency	Frequency	MonetaryValue
General_Segment			

	Recency	Frequency	MonetaryValue
General_Segment			
Bronze	1.08	-0.83	-0.87
Gold	-0.78	1.49	1.56
Sliver	-0.22	-0.45	-0.48

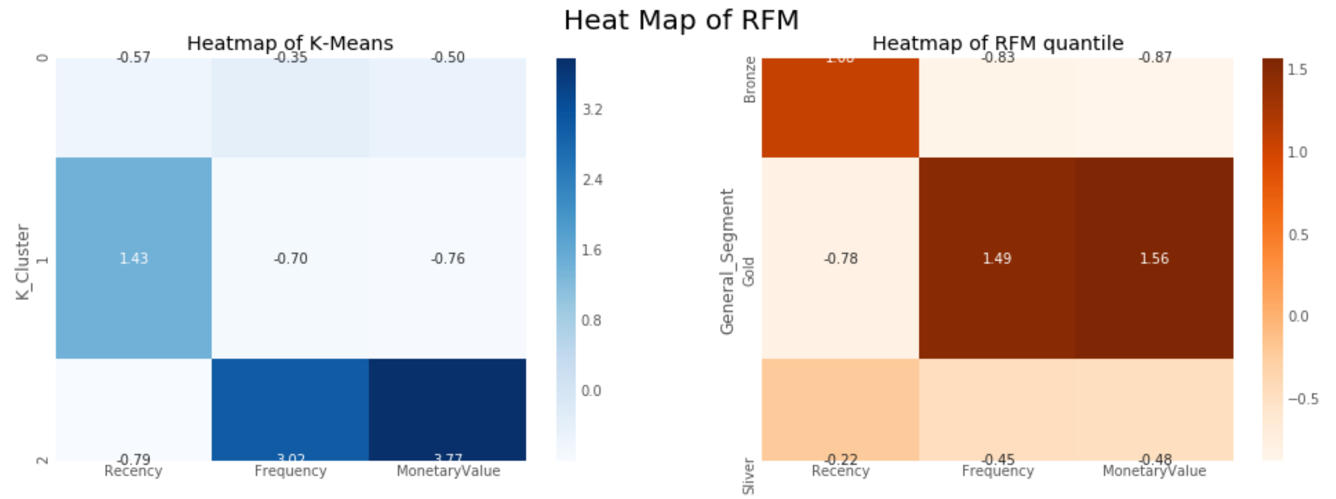
In [68]:

```
# heatmap with RFM
f, (ax1, ax2) = plt.subplots(1,2, figsize=(15, 5))
sns.heatmap(data=relative_imp, annot=True, fmt='.2f', cmap='Blues',ax=ax1)
ax1.set(title = "Heatmap of K-Means")

# a snake plot with K-Means
sns.heatmap(prop_rfm, cmap= 'Oranges', fmt= '.2f', annot = True,ax=ax2)
ax2.set(title = "Heatmap of RFM quantile")

plt.suptitle("Heat Map of RFM",fontsize=20) #make title fontsize subtitle

plt.show()
```



Project Task: Week 4

Data Reporting:

1. Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:

a. Country-wise analysis to demonstrate average spend. Use a bar chart to show the monthly figures

b. Bar graph of top 15 products which are mostly ordered by the users to show the number of products sold

c. Bar graph to show the count of orders vs. hours throughout the day

d. Plot the distribution of RFM values using histogram and frequency charts

e. Plot error (cost) vs. number of clusters selected

f. Visualize to compare the RFM values of the clusters using heatmap

For Tableau Dashboard[click here]()

In [70]:

```
from PIL import Image as PILImage
import base64, io, IPython
image = PILImage.open('Pgp-Retail analysis.jpg')
output = io.BytesIO()
image.save(output, format='PNG')
encoded_string = base64.b64encode(output.getvalue()).decode()
html = '<img src="data:image/png;base64,{}"/>'.format(encoded_string)
```

```
IPython.display.HTML(html)
```

Out[70]: