In [1]:

```
#Importing relevant Libraries to work with
import pandas as pd
import numpy as np
import datetime as dt
import scipy as sp
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
#Importing Dataset for relevant Data
train= pd.read_excel(r'C:\Users\Student 20\Desktop\Online Retail.xlsx')
```

In [3]:

```
#Viewing the data head I will be working with
train.head()
```

Out[3]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

In [4]:

```
#Viewing the shape of dataset
print(train.shape)
```

```
(541909, 8)
```

In [5]:

```
#Viewing missing values in the Dataset
train.isnull().sum()
```

Out[5]:

```
InvoiceNo           0
```

```
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

In [6]:

```
#Missing value treatment through dropping the missing values
train=train.dropna()
```

In [7]:

```
#Viewing if the treatment was succesfull
train.isnull().sum()
```

Out[7]:

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

In [8]:

```
#Checking for duplicated data
train.duplicated(subset=None, keep='first').sum()
```

Out[8]:

```
5225
```

In [9]:

```
#Dropping duplicated data
train= train.drop_duplicates(subset=None, keep='first')
```

In [10]:

```
#Viewing if the duplicates have been dropped
train.duplicated(subset=None, keep='first').sum()
```

Out[10]:

```
0
```

In [11]:

```
#Checking dataset info
.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 401604 entries, 0 to 541908
Data columns (total 8 columns):
InvoiceNo      401604 non-null object
StockCode      401604 non-null object
Description    401604 non-null object
Quantity       401604 non-null int64
InvoiceDate    401604 non-null datetime64[ns]
UnitPrice      401604 non-null float64
CustomerID     401604 non-null float64
Country        401604 non-null object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 27.6+ MB
```

In [12]:

```
#Converting customerID to an integer to work better with it
train.CustomerID = train.CustomerID.apply(lambda x: int(x) if x == x else "")
```

In [13]:

```
#Checking if CustomerId has been converted
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 401604 entries, 0 to 541908
Data columns (total 8 columns):
InvoiceNo      401604 non-null object
StockCode      401604 non-null object
Description    401604 non-null object
Quantity       401604 non-null int64
InvoiceDate    401604 non-null datetime64[ns]
UnitPrice      401604 non-null float64
CustomerID     401604 non-null int64
Country        401604 non-null object
dtypes: datetime64[ns](1), float64(1), int64(2), object(4)
memory usage: 27.6+ MB
```

In [14]:

```
#Descriptive Analytics on my dataset
train.describe()
```

Out[14]:

	Quantity	UnitPrice	CustomerID
count	401604.000000	401604.000000	401604.000000

	Quantity	UnitPrice	CustomerID
mean	12.183273	3.474064	15281.160818
std	250.283037	69.764035	1714.006089
min	-80995.000000	0.000000	12346.000000
25%	2.000000	1.250000	13939.000000
50%	5.000000	1.950000	15145.000000
75%	12.000000	3.750000	16784.000000
max	80995.000000	38970.000000	18287.000000

In [15]:

```
train.head()
```

Out[15]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850	United Kingdom

In [16]:

```
#Preparing for Cohort analysis
def get_month(x): return dt.datetime(x.year, x.month, 1)
```

In [18]:

```
#Creating and defining Cohort groupings
train['InvoiceMonth'] = train['InvoiceDate'].apply(get_month)
grouping1 = train.groupby('CustomerID')['InvoiceMonth']
train['CohortMonth'] = grouping1.transform('min')
train.describe()
```

Out[18]:

	Quantity	UnitPrice	CustomerID
count	401604.000000	401604.000000	401604.000000
mean	12.183273	3.474064	15281.160818
std	250.283037	69.764035	1714.006089
min	-80995.000000	0.000000	12346.000000

	Quantity	UnitPrice	CustomerID
25%	2.000000	1.250000	13939.000000
50%	5.000000	1.950000	15145.000000
75%	12.000000	3.750000	16784.000000
max	80995.000000	38970.000000	18287.000000

In [19]:

```
def get_date_int(train, column):
    year = train[column].dt.year
    month = train[column].dt.month
    day = train[column].dt.day
    return year, month, day
invoice_year, invoice_month, _ = get_date_int(train, 'InvoiceMonth')
cohort_year, cohort_month, _ = get_date_int(train, 'CohortMonth')
```

In [20]:

```
years_diff = invoice_year - cohort_year
months_diff = invoice_month - cohort_month
```

In [21]:

```
#Creating and defining Cohort Indexing
train['CohortIndex'] = years_diff * 12 + months_diff + 1
train.head()
```

Out[21]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortM
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850	United Kingdom	2010-12-01	2010-12
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850	United Kingdom	2010-12-01	2010-12
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12
4	536365	84029E	RED WOOLLY	6	2010-12-01	3.39	17850	United	2010-12-01	2010-12

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortM
			HOTTIE WHITE HEART.		08:26:00			Kingdom		

In [22]:

```
# Count monthly active customers from each cohort
grouping_count = train.groupby(['CohortMonth', 'CohortIndex'])
cohort_data = grouping_count['CustomerID'].apply(pd.Series.nunique)
cohort_data = cohort_data.reset_index()
cohort_counts = cohort_data.pivot(index='CohortMonth',
                                  columns='CohortIndex',
                                  values='CustomerID')
print(cohort_counts.head())
```

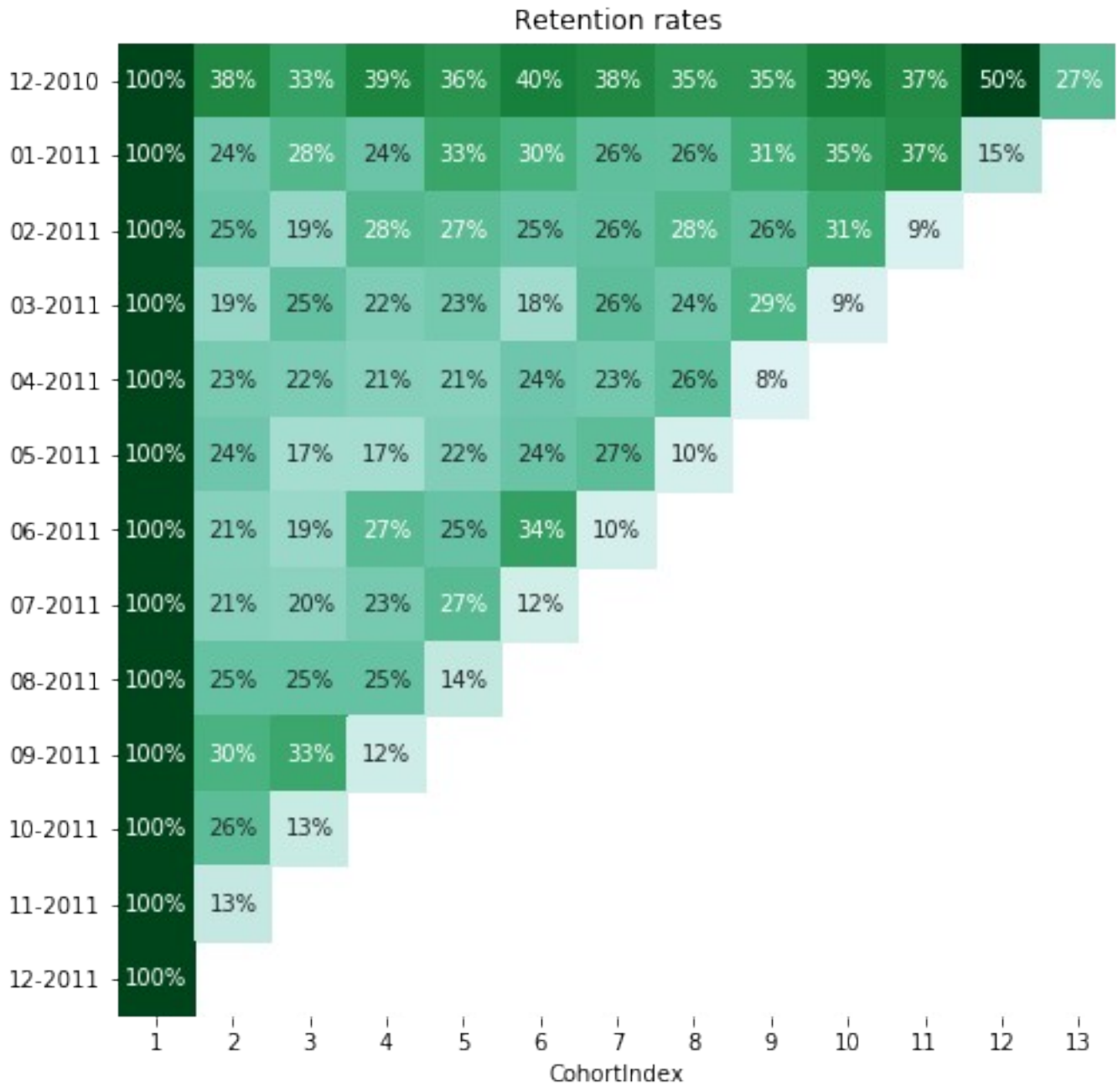
```
CohortIndex     1      2      3      4      5      6      7      8      9   \
CohortMonth
2010-12-01   948.0  362.0  317.0  367.0  341.0  376.0  360.0  336.0  336.0
2011-01-01   421.0  101.0  119.0  102.0  138.0  126.0  110.0  108.0  131.0
2011-02-01   380.0   94.0   73.0  106.0  102.0   94.0   97.0  107.0   98.0
2011-03-01   440.0   84.0  112.0   96.0  102.0   78.0  116.0  105.0  127.0
2011-04-01   299.0   68.0   66.0   63.0   62.0   71.0   69.0   78.0   25.0

CohortIndex     10     11     12     13
CohortMonth
2010-12-01   374.0  354.0  474.0  260.0
2011-01-01   146.0  155.0   63.0    NaN
2011-02-01   119.0   35.0    NaN    NaN
2011-03-01    39.0    NaN    NaN    NaN
2011-04-01     NaN    NaN    NaN    NaN
```

In [23]:

```
# --Calculate Retention Rate--
cohort_sizes = cohort_counts.iloc[:,0]
retention = cohort_counts.divide(cohort_sizes, axis=0)
retention.round(3) * 100
retention.index = retention.index.strftime('%m-%Y')
```

In [24]:

```
# Plot retention rates
plt.figure(figsize=(10, 8))
plt.title('Retention rates')
sns.heatmap(data = retention, annot = True, fmt = '.0%',vmin = 0.0,vmax = 0.5,cmap =
'BuGn')
plt.show()
```



In [25]:

```
#Creating and defining Total Price
train['TotalPrice']=train['Quantity']*train['UnitPrice']
train.head(10)
```

Out[25]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	Cohort
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850	United Kingdom	2010-12-01	2010-12
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	Cohort
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850	United Kingdom	2010-12-01	2010-12
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	2010-12-01 08:26:00	7.65	17850	United Kingdom	2010-12-01	2010-12
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	2010-12-01 08:26:00	4.25	17850	United Kingdom	2010-12-01	2010-12
7	536366	22633	HAND WARMER UNION JACK	6	2010-12-01 08:28:00	1.85	17850	United Kingdom	2010-12-01	2010-12
8	536366	22632	HAND WARMER RED POLKA DOT	6	2010-12-01 08:28:00	1.85	17850	United Kingdom	2010-12-01	2010-12
9	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	2010-12-01 08:34:00	1.69	13047	United Kingdom	2010-12-01	2010-12

In [26]:

```
#
train= train[np.isfinite(train['CustomerID'])]
```

In [27]:

```
train= train[train['Quantity'] > 0]
train.head()
```

Out[27]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortM
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850	United Kingdom	2010-12-01	2010-12
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850	United Kingdom	2010-12-01	2010-12
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12

In [28]:

```
train.shape
```

Out[28]:

```
(392732, 12)
```

In [29]:

```
#Calculate recency, frequency and Monetary
NOW = dt.datetime(2011,12,10)
train.head()
```

Out[29]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortM
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850	United Kingdom	2010-12-01	2010-12
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortM
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850	United Kingdom	2010-12-01	2010-12
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850	United Kingdom	2010-12-01	2010-12

In [30]:

```
RFM = train.groupby('CustomerID').agg({'InvoiceMonth': lambda x: (NOW -
x.max()).days, 'InvoiceNo': lambda x: len(x),'TotalPrice': lambda x: x.sum()})
RFM['InvoiceMonth'] = RFM['InvoiceMonth'].astype(int)
RFM.rename(columns={'InvoiceMonth': 'Recency',
                          'InvoiceNo': 'Frequency',
                         'TotalPrice': 'Monetaryvalue'}, inplace=True)
RFM.head(10)
```

Out[30]:

	Recency	Frequency	Monetaryvalue
CustomerID			
12346	343	1	77183.60
12347	9	182	4310.00
12348	100	31	1797.24
12349	39	73	1757.55
12350	312	17	334.40
12352	39	85	2506.04
12353	223	4	89.00
12354	253	58	1079.40
12355	223	13	459.40
12356	39	59	2811.43

In [31]:

```
RFM.describe()
```

Out[31]:

	Recency	Frequency	Monetaryvalue
count	4339.000000	4339.000000	4339.000000
mean	107.021203	90.512100	2048.215924
std	100.090370	225.515328	8984.248352
min	9.000000	1.000000	0.000000
25%	39.000000	17.000000	306.455000
50%	70.000000	41.000000	668.560000
75%	162.000000	98.000000	1660.315000
max	374.000000	7676.000000	280206.020000

In [32]:

```
quartiles = RFM.quantile(q=[0.25,0.50,0.75])
print(quartiles, type(quartiles))
```

```
      Recency  Frequency  Monetaryvalue
0.25     39.0       17.0        306.455
0.50     70.0       41.0        668.560
0.75    162.0       98.0       1660.315 <class 'pandas.core.frame.DataFrame'>
```

In [33]:

```
quartiles=quartiles.to_dict()
quartiles
```

Out[33]:

```
{'Recency': {0.25: 39.0, 0.5: 70.0, 0.75: 162.0},
 'Frequency': {0.25: 17.0, 0.5: 41.0, 0.75: 98.0},
 'Monetaryvalue': {0.25: 306.45500000000004,
  0.5: 668.5600000000002,
  0.75: 1660.315}}
```

In [34]:

```
## for Recency

def RClass(x,p,d):
    if x <= d[p][0.25]:
        return 1
    elif x <= d[p][0.50]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4

## for Frequency and Monetary value

def FMClass(x,p,d):
```

```
    if x <= d[p][0.25]:
        return 4
    elif x <= d[p][0.50]:
        return 3
    elif x <= d[p][0.75]:
        return 2
    else:
        return 1
```

In [35]:

```
RFMSeg = RFM
RFMSeg['R_Quartile'] = RFMSeg['Recency'].apply(RClass, args=('Recency',quartiles,))
RFMSeg['F_Quartile'] = RFMSeg['Frequency'].apply(FMClass,
args=('Frequency',quartiles,))
RFMSeg['M_Quartile'] = RFMSeg['Monetaryvalue'].apply(FMClass,
args=('Monetaryvalue',quartiles,))
```

In [36]:

```
RFMSeg['Total Score'] = RFMSeg['R_Quartile'] + RFMSeg['F_Quartile']
+RFMSeg['M_Quartile']
RFMSeg.head()
```

Out[36]:

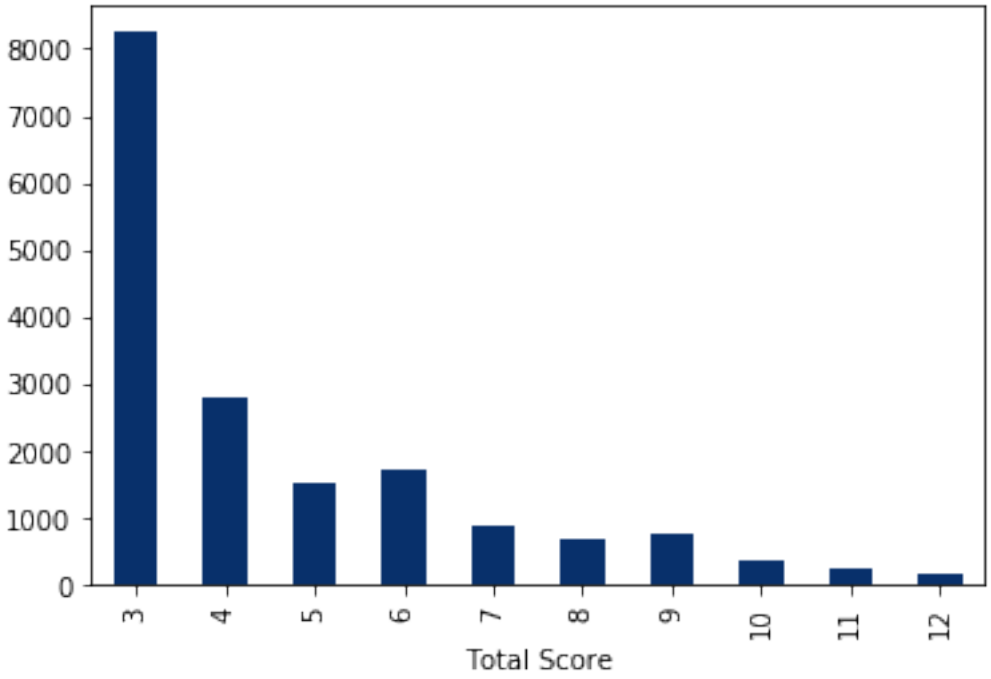
	Recency	Frequency	Monetaryvalue	R_Quartile	F_Quartile	M_Quartile	Total Score
CustomerID							
12346	343	1	77183.60	4	4	1	9
12347	9	182	4310.00	1	1	1	3
12348	100	31	1797.24	3	3	1	7
12349	39	73	1757.55	1	2	1	4
12350	312	17	334.40	4	4	3	11

In [37]:

```
RFMSeg.groupby('Total Score').agg('Monetaryvalue').mean().plot(kind='bar',
colormap='Blues_r')
```

Out[37]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x217937c57f0>
```

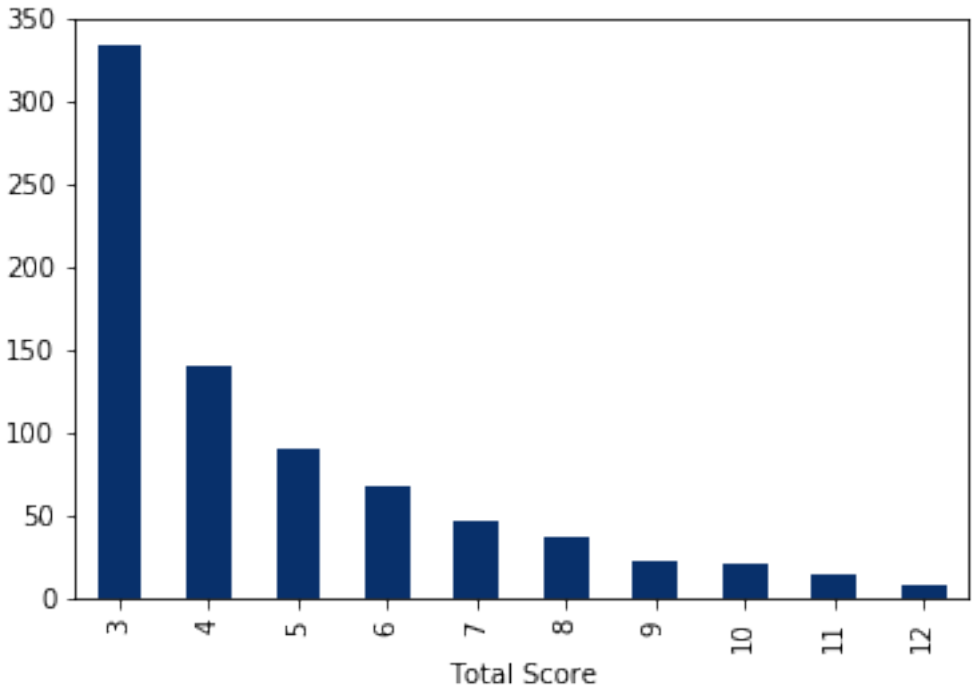


In [38]:

```
RFMSeg.groupby('Total Score').agg('Frequency').mean().plot(kind='bar',
colormap='Blues_r')
```

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x217940b44e0>
```

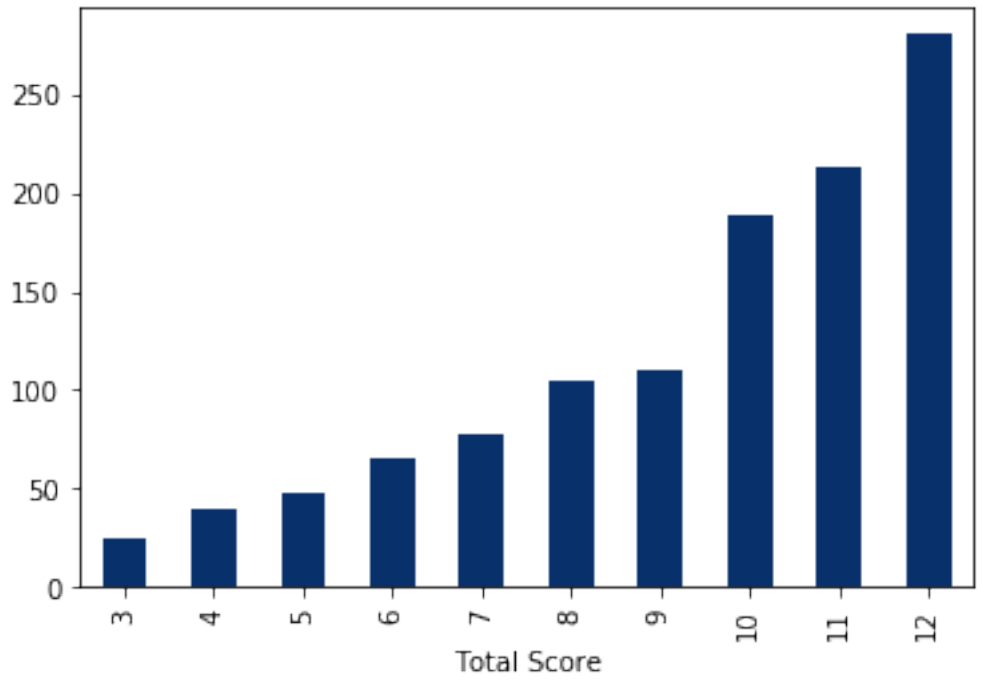


In [39]:

```
RFMSeg.groupby('Total Score').agg('Recency').mean().plot(kind='bar',
colormap='Blues_r')
```

Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21790180048>
```



In [40]:

```
#KMeans Clustering
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform( RFM )
import random
random.seed(9008)
X_sample = np.array(random.sample(X_scaled.tolist(),20))
```

In [41]:

```
clusters = KMeans(3)
clusters.fit( X_scaled )
```

Out[41]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [42]:

```
RFM["Cluster"] = clusters.labels_
RFM
type(X_scaled)
```

Out[42]:

```
numpy.ndarray
```

In [43]:

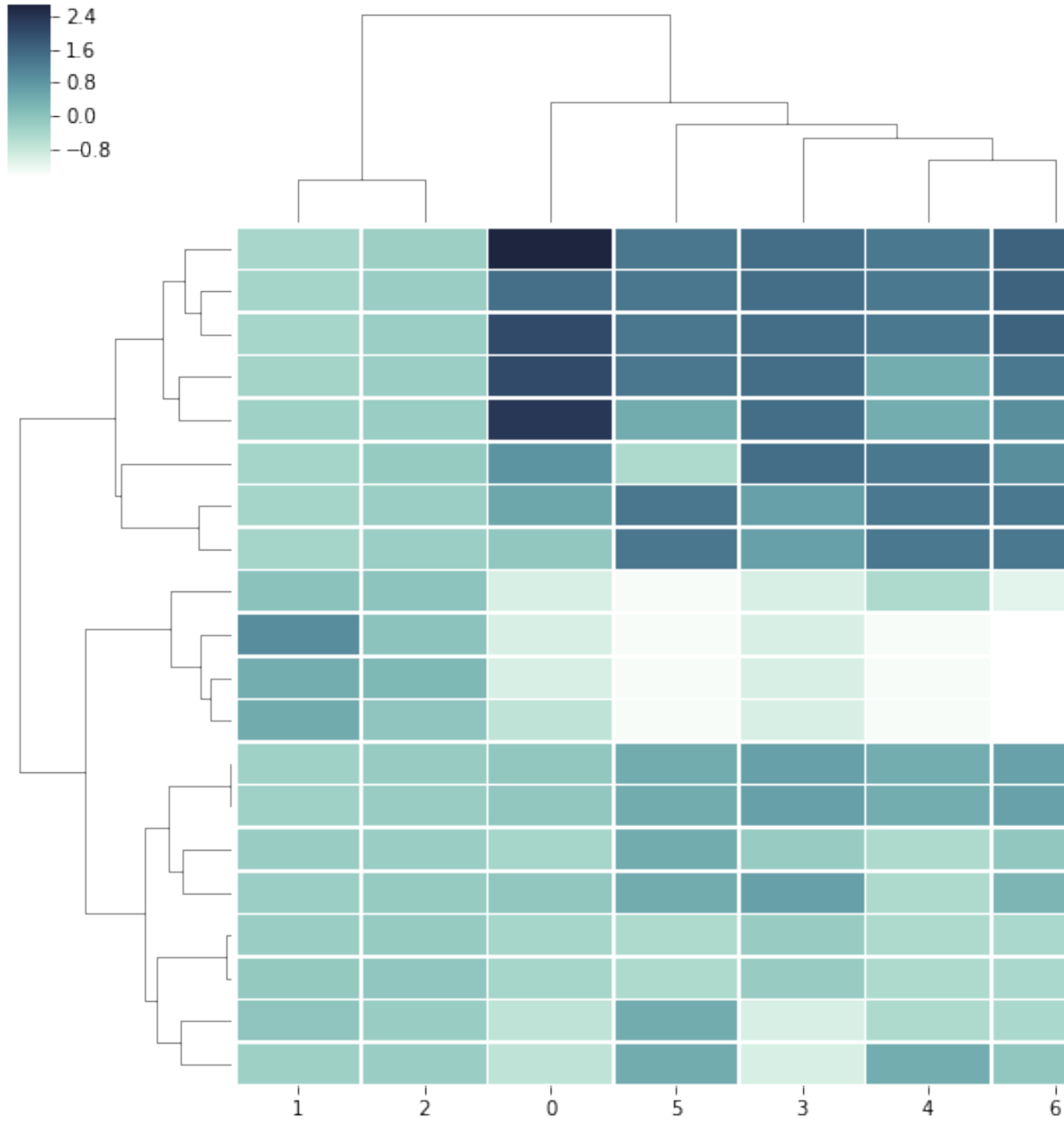
```
RFM.groupby('Cluster' ).mean()
```

Out[43]:

	Recency	Frequency	Monetaryvalue	R_Quartile	F_Quartile	M_Quartile	Total Score
Cluster							
0	47.950805	137.277318	2732.099496	1.422725	1.724423	1.736613	4.883761
1	174.492361	21.897486	479.950188	3.074421	3.412518	3.374076	9.861015
2	13.615385	2536.615385	125981.340000	1.000000	1.230769	1.000000	3.230769

In [44]:

```
RFM.drop( 'Cluster', axis = 1, inplace = True )
cmap = sns.cubehelix_palette(as_cmap=True, rot=-.3, light=1)
g = sns.clustermap(X_sample, cmap=cmap, linewidths=.5)
```



In [45]:

```
clusters = KMeans(3)  # 3 clusters
clusters.fit( X_scaled )
RFM["cluster_label"] = clusters.labels_
```

In [46]:

```
RFM.groupby('cluster_label').mean()
```

Out[46]:

	Recency	Frequency	Monetaryvalue	R_Quartile	F_Quartile	M_Quartile	Total Score
cluster_label							
0	174.492361	21.897486	479.950188	3.074421	3.412518	3.374076	9.861015
1	47.950805	137.277318	2732.099496	1.422725	1.724423	1.736613	4.883761
2	13.615385	2536.615385	125981.340000	1.000000	1.230769	1.000000	3.230769

In [47]:

```
RFM_0 = RFM[RFM.cluster_label == 0]
RFM_0.head(10)
```

Out[47]:

	Recency	Frequency	Monetaryvalue	R_Quartile	F_Quartile	M_Quartile	Total Score	cluster_label
CustomerID								
12346	343	1	77183.60	4	4	1	9	0
12350	312	17	334.40	4	4	3	11	0
12353	223	4	89.00	4	4	4	12	0
12354	253	58	1079.40	4	2	2	8	0
12355	223	13	459.40	4	4	3	11	0
12361	312	10	189.90	4	4	4	12	0
12363	131	23	552.00	3	3	3	9	0
12365	312	22	641.38	4	3	3	10	0
12367	9	11	168.90	1	4	4	9	0
12373	312	14	364.60	4	4	3	11	0

In [48]:

```
RFM_1 = RFM[RFM.cluster_label == 1]
RFM_1.head(10)
```

Out[48]:

	Recency	Frequency	Monetaryvalue	R_Quartile	F_Quartile	M_Quartile	Total Score	cluster_label
CustomerID								
12347	9	182	4310.00	1	1	1	3	1

	Recency	Frequency	Monetaryvalue	R_Quartile	F_Quartile	M_Quartile	Total Score	cluster_label
CustomerID								
12348	100	31	1797.24	3	3	1	7	1
12349	39	73	1757.55	1	2	1	4	1
12352	39	85	2506.04	1	2	1	4	1
12356	39	59	2811.43	1	2	1	4	1
12357	39	131	6207.67	1	1	1	3	1
12358	9	19	1168.06	1	3	2	6	1
12359	70	245	6310.03	2	1	1	4	1
12360	70	129	2662.06	2	1	1	4	1
12362	9	266	5226.23	1	1	1	3	1

In [49]:

```
RFM_2 = RFM[RFM.cluster_label == 2]
RFM_2.head(10)
```

Out[49]:

	Recency	Frequency	Monetaryvalue	R_Quartile	F_Quartile	M_Quartile	Total Score	cluster_label
CustomerID								
12415	39	716	124914.53	1	1	1	3	2
12748	9	4413	33053.19	1	1	1	3	2
13089	9	1814	58762.08	1	1	1	3	2
14096	9	5111	65164.79	1	1	1	3	2
14156	39	1395	117210.08	1	1	1	3	2
14646	9	2080	280206.02	1	1	1	3	2
14911	9	5672	143711.17	1	1	1	3	2
15311	9	2366	60632.75	1	1	1	3	2
16446	9	3	168472.50	1	4	1	6	2
17450	9	336	194390.79	1	1	1	3	2

In [50]:

```
RFM_0.mean()
```

Out[50]:

```
Recency          174.492361
Frequency         21.897486
Monetaryvalue    479.950188
R_Quartile         3.074421
F_Quartile         3.412518
M_Quartile         3.374076
Total Score        9.861015
cluster_label      0.000000
dtype: float64
```

In [51]:

```
RFM_1.mean()
```

Out[51]:

```
Recency           47.950805
Frequency        137.277318
Monetaryvalue   2732.099496
R_Quartile         1.422725
F_Quartile         1.724423
M_Quartile         1.736613
Total Score        4.883761
```

```
cluster_label       1.000000
dtype: float64
```

In [52]:

```
RFM_2.mean()
```

Out[52]:

```
Recency              13.615385
Frequency          2536.615385
Monetaryvalue    125981.340000
R_Quartile            1.000000
F_Quartile            1.230769
M_Quartile            1.000000
Total Score           3.230769
cluster_label         2.000000
dtype: float64
```

In [53]:

```
RFM.head(10)
```

Out[53]:

	Recency	Frequency	Monetaryvalue	R_Quartile	F_Quartile	M_Quartile	Total Score	cluster_label
CustomerID								
12346	343	1	77183.60	4	4	1	9	0
12347	9	182	4310.00	1	1	1	3	1
12348	100	31	1797.24	3	3	1	7	1
12349	39	73	1757.55	1	2	1	4	1
12350	312	17	334.40	4	4	3	11	0
12352	39	85	2506.04	1	2	1	4	1
12353	223	4	89.00	4	4	4	12	0
12354	253	58	1079.40	4	2	2	8	0
12355	223	13	459.40	4	4	3	11	0
12356	39	59	2811.43	1	2	1	4	1

In [54]:

```
RFM.groupby('cluster_label').mean()
```

Out[54]:

	Recency	Frequency	Monetaryvalue	R_Quartile	F_Quartile	M_Quartile	Total Score
cluster_label							
0	174.492361	21.897486	479.950188	3.074421	3.412518	3.374076	9.861015
1	47.950805	137.277318	2732.099496	1.422725	1.724423	1.736613	4.883761

	Recency	Frequency	Monetaryvalue	R_Quartile	F_Quartile	M_Quartile	Total Score
cluster_label							
2	13.615385	2536.615385	125981.340000	1.000000	1.230769	1.000000	3.230769