In [1]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
```

In [2]:
```python
train = pd.read_excel('train.xlsx')
test = pd.read_excel('test.xlsx')
```

In [3]:
```python
#append train test data
train['type']='train'
test['type']='test'
data = train.append(test, sort=False)
```

In [4]:
```python
data.head()
```

Out[4]:

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

In [5]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 541909 entries, 0 to 162572
Data columns (total 9 columns):
InvoiceNo      541909 non-null object
StockCode      541909 non-null object
Description    540455 non-null object
Quantity       541909 non-null int64
InvoiceDate    541909 non-null datetime64[ns]
UnitPrice      541909 non-null float64
CustomerID     406829 non-null float64
Country        541909 non-null object
type           541909 non-null object
dtypes: datetime64[ns](1), float64(2), int64(1), object(5)
memory usage: 41.3+ MB
```

In [6]:
```python
data.describe()
```

Out[6]:

	Quantity	UnitPrice	CustomerID
count	541909.000000	541909.000000	406829.000000
mean	9.552250	4.611114	15287.690570
std	218.081158	96.759853	1713.600303
min	-80995.000000	-11062.060000	12346.000000
25%	1.000000	1.250000	13953.000000
50%	3.000000	2.080000	15152.000000
75%	10.000000	4.130000	16791.000000
max	80995.000000	38970.000000	18287.000000

In [7]:
```python
data.isnull().sum()
```

Out[7]:

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

In [8]:
```python
#remove null values
data.dropna(subset=['CustomerID'], inplace=True)
```

In [9]:
```python
data.isnull().sum()
```

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

In [10]:
```
#Remove duplicate values
data[data.duplicated()].shape
```

Out[10]:
```
(3124, 9)
```

In [11]:
```
data.drop_duplicates(inplace=True)
```

In [12]:
```
#Reset the index
indexcol=np.array(list(range(0,len(data))))
data.set_index(indexcol,inplace=True)
```

In [13]:
```
#Remove the orders that were reversed
data = data[data['Quantity'] > 0]
```

In [17]:
```
#Customers who ordered more than once
n_orders = data.groupby(['CustomerID'])['InvoiceNo'].nunique()
mult_orders_perc = np.sum(n_orders > 1) / data['CustomerID'].nunique()
print(f'{100 * mult_orders_perc:.2f}% of customers ordered more than once.')
```

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

In [18]:
```
import datetime as dt
#Perform cohort analysis
def get_month(x) : return dt.datetime(x.year,x.month,1)
data['InvoiceMonth'] = data['InvoiceDate'].apply(get_month)
grouping = data.groupby('CustomerID')['InvoiceMonth']
data['CohortMonth'] = grouping.transform('min')
data.tail()
```

Out[18]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	type	InvoiceMonth	CohortMonth
403700	574102	22866	HAND WARMER SCOTTY DOG DESIGN	24	2011-11-03 10:27:00	2.10	16128.0	United Kingdom	test	2011-11-01	2011-03-01
403701	545226	22919	HERB MARKER MINT	12	2011-03-01 09:33:00	0.65	12428.0	Finland	test	2011-03-01	2011-03-01
403702	573160	22077	6 RIBBONS RUSTIC CHARM	12	2011-10-28 08:58:00	1.95	14359.0	United Kingdom	test	2011-10-01	2011-09-01
403703	552321	23204	CHARLOTTE BAG APPLES DESIGN	10	2011-05-09 09:15:00	0.85	17049.0	United Kingdom	test	2011-05-01	2011-03-01
403704	573359	21983	PACK OF 12 BLUE PAISLEY TISSUES	4	2011-10-30 12:48:00	0.39	14178.0	United Kingdom	test	2011-10-01	2011-08-01

In [19]:
```
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(data,'InvoiceMonth')
cohort_year,cohort_month,_ = get_month_int(data,'CohortMonth')

year_diff = invoice_year - cohort_year
month_diff = invoice_month - cohort_month

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

In [20]:
```python
#Count monthly active customers from each cohort
grouping = data.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[20]:

CohortIndex CohortMonth	1	2	3	4	5	6	7	8	9	10	11	12	13
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 [21]:
```python
# 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 axis
retention.round(3) * 100 #to show the number as percentage
```

In [22]:
```python
#Build the heatmap
import matplotlib.pyplot as plt
plt.figure(figsize=(15, 8))
plt.title('Retention rates')
sns.heatmap(data=retention,annot = True,fmt = '.0%',vmin = 0.0,vmax = 0.5,cmap="BuPu_r")
plt.show()
```

In [23]:

```
#Average quantity for each cohort
grouping = data.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="BuGn_r")
plt.show()
```

RFM

In [24]:

```
#Recency
data['Recent']=(pd.to_datetime(data['InvoiceDate'].max()) - pd.to_datetime(data['InvoiceDate'])).dt.days
data = data[data['Recent'] <= 366]
Recency=data.groupby(['CustomerID'], as_index=False)['Recent'].max()
Recency.columns = ['CustomerID','Recency']
Recency.head()
```

Out[24]:

	CustomerID	Recency
0	12346.0	325
1	12347.0	366
2	12348.0	357
3	12349.0	18
4	12350.0	309

In [25]:

```
Recency.shape
```

Out[25]: (4289, 2)

In [26]:

```
#Frequency
Frequency = data.groupby(['CustomerID'], as_index=False)['InvoiceNo'].count()
Frequency.columns = ['CustomerID','Frequency']
Frequency.head()
```

Out[26]:

	CustomerID	Frequency
0	12346.0	1
1	12347.0	182
2	12348.0	31
3	12349.0	73
4	12350.0	17

In [27]:
```
#Monetary
data['Totalsales'] = data['Quantity']*data['UnitPrice']
Monetary = data.groupby(['CustomerID'], as_index=False)['Totalsales'].agg('sum')
Monetary.columns = ['CustomerID','Monetary']
Monetary.head()
```

```
C:\Users\bedant\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWar
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: http://pandas.pydata.org/pandas-docs/stable/index
```

Out[27]:

	CustomerID	Monetary
0	12346.0	77183.60
1	12347.0	4310.00
2	12348.0	1797.24
3	12349.0	1757.55
4	12350.0	334.40

In [28]:
```
temp_df = pd.merge(Recency,Frequency,on='CustomerID')
RFM_metrics = pd.merge(temp_df,Monetary,on='CustomerID')
```

In [29]:
```
RFM_metrics.head()
```

Out[29]:

	CustomerID	Recency	Frequency	Monetary
0	12346.0	325	1	77183.60
1	12347.0	366	182	4310.00
2	12348.0	357	31	1797.24
3	12349.0	18	73	1757.55
4	12350.0	309	17	334.40

In [31]:
```
quantiles = RFM_metrics.quantile(q=[0.25,0.5,0.75])
```

In [32]:
```
quantiles
```

Out[32]:

	CustomerID	Recency	Frequency	Monetary
0.25	13810.0	105.0	17.0	305.54
0.50	15289.0	241.0	41.0	663.65
0.75	16774.0	315.0	98.0	1643.93

In [33]:
```
quantiles.to_dict()
```

Out[33]:
```
{'CustomerID': {0.25: 13810.0, 0.5: 15289.0, 0.75: 16774.0},
 'Recency': {0.25: 105.0, 0.5: 241.0, 0.75: 315.0},
 'Frequency': {0.25: 17.0, 0.5: 41.0, 0.75: 98.0},
 'Monetary': {0.25: 305.54, 0.5: 663.65, 0.75: 1643.9300000000003}}
```

In [34]:
```
#Define function for the most frequent and high spending customer
def FMscore(x,c,d):
    if x <= d[c][0.25]:
        return 1
    elif x <= d[c][0.50]:
        return 2
    elif x <= d[c][0.75]:
        return 3
    else:
        return 4
```

In [35]:
```
#Define function for the most Recent customer
def Rscore(x,c,d):
    if x <= d[c][0.25]:
        return 4
    elif x <= d[c][0.50]:
        return 3
    elif x <= d[c][0.75]:
        return 2
    else:
        return 1
```

In [36]:
```
rfm_segmentation = RFM_metrics
rfm_segmentation['R_Quartile'] = rfm_segmentation['Recency'].apply(Rscore, args=('Recency',quantiles))
rfm_segmentation['F_Quartile'] = rfm_segmentation['Frequency'].apply(FMscore, args=('Frequency',quantiles))
rfm_segmentation['M_Quartile'] = rfm_segmentation['Monetary'].apply(FMscore, args=('Monetary',quantiles))
```

In [37]:
```
rfm_segmentation.head()
```

Out[37]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile
0	12346.0	325	1	77183.60	1	1	4
1	12347.0	366	182	4310.00	1	4	4
2	12348.0	357	31	1797.24	1	2	4
3	12349.0	18	73	1757.55	4	3	4
4	12350.0	309	17	334.40	2	1	2

In [38]:
```
#RFM segment
rfm_segmentation['RFMSegment'] = rfm_segmentation['R_Quartile'].map(str) + rfm_segmentation['F_Quartile'].map(str) + rfm_segment
rfm_segmentation['RFMScore'] = rfm_segmentation.R_Quartile + rfm_segmentation.F_Quartile + rfm_segmentation.M_Quartile
rfm_segmentation.head()
```

Out[38]:

	CustomerID	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMSegment	RFMScore
0	12346.0	325	1	77183.60	1	1	4	114	6
1	12347.0	366	182	4310.00	1	4	4	144	9
2	12348.0	357	31	1797.24	1	2	4	124	7
3	12349.0	18	73	1757.55	4	3	4	434	11
4	12350.0	309	17	334.40	2	1	2	212	5

In [39]:
```
rfm_segmentation = rfm_segmentation.sort_values('RFMScore',ascending=False)
```

In [40]:
```
#rfm_segmentation.to_csv('rfm.csv')
```

In [41]:
```
rfm_segmentation.head()
```

In [42]:
```
print("Best Customers: {}".format(len(rfm_segmentation[rfm_segmentation['RFMScore'] == 12])))
print("Frequent Customers: {}".format(len(rfm_segmentation[rfm_segmentation['F_Quartile'] == 4])))
print("Money spending Customers: {}".format(len(rfm_segmentation[rfm_segmentation['M_Quartile'] == 4])))
print("Lost Customers: {}".format(len(rfm_segmentation[rfm_segmentation['RFMScore'] == 3])))
```

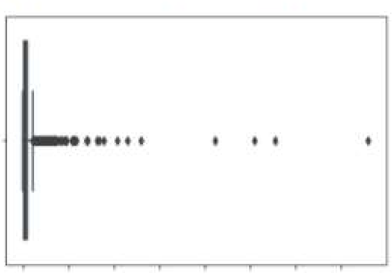
```
Best Customers: 38
Frequent Customers: 1060
Money spending Customers: 1072
Lost Customers: 96
```

In [43]:
```
#Normalize and standardize the data
rfm_segmentation.skew()
```

Out[43]:
```
CustomerID     0.005121
Recency       -0.346045
Frequency     18.040369
Monetary      19.543001
R_Quartile    -0.009163
F_Quartile     0.018803
M_Quartile     0.000376
RFMSegment     0.000494
RFMScore      -0.191462
dtype: float64
```

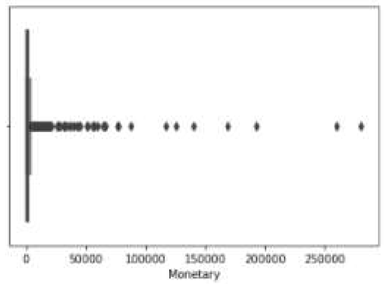
In [44]:
```
#UnitPrice
sns.boxplot(rfm_segmentation['Frequency'])
```

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x23c117205c0>



In [45]:
```
#Quantity
sns.boxplot(rfm_segmentation['Monetary'])
```

Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x23c103556d8>



In [46]:
```
#Remove Outliers
rfm_segmentation1 = rfm_segmentation[(rfm_segmentation.Frequency < 3000 ) & (rfm_segmentation.Monetary < 10000)]
```

In [47]:
```
rfm_segmentation1.skew()
```

Out[47]:
```
CustomerID     0.002752
Recency       -0.320016
Frequency      4.378659
Monetary       2.453273
R_Quartile    -0.042128
F_Quartile     0.053051
M_Quartile     0.039195
RFMSegment    -0.035278
RFMScore      -0.158621
dtype: float64
```

In [48]:
```
rfm_segmentation1['Frequency'] = np.log(rfm_segmentation1['Frequency']+0.01)
```

In [49]:
```
rfm_segmentation1['Monetary'] = np.log(rfm_segmentation1['Monetary']+0.01)
```

```
C:\Users\bedant\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

In [50]:
```
rfm_segmentation1.skew()
```

Out[50]:
```
CustomerID     0.002752
Recency       -0.320016
Frequency     -0.320026
Monetary      -0.242656
R_Quartile    -0.042128
F_Quartile     0.053051
M_Quartile     0.039195
RFMSegment    -0.035278
RFMScore      -0.158621
dtype: float64
```

In [51]:
```
rfm_segmentation1.shape
```

Out[51]: (4186, 9)

In [53]:
```
final = rfm_segmentation1.iloc[:,1:4]
```

In [54]:
```
#Standardise the data

from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
data2 = scale.fit_transform(final)
data2
```

Out[54]:
```
array([[-1.38702039,  1.77737963,  1.55695303],
       [-1.23784808,  1.4991496 ,  1.47478015],
       [-1.33437134,  1.13561726,  1.24422827],
       ...,
       [ 1.28053163, -1.74484045, -1.86558628],
       [ 1.34195552, -0.76570372, -0.78815411],
       [ 1.11380962, -0.66125912, -1.05288387]])
```

In [55]:
```
final.head()
```

Out[55]:

	Recency	Frequency	Monetary
1609	53	5.888906	8.291496
3752	70	5.533429	8.196577
1298	59	5.068907	7.930264
3438	34	5.023946	7.934259
398	63	5.743035	7.685124

In [56]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

print(__doc__)

X = data2

range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)
```

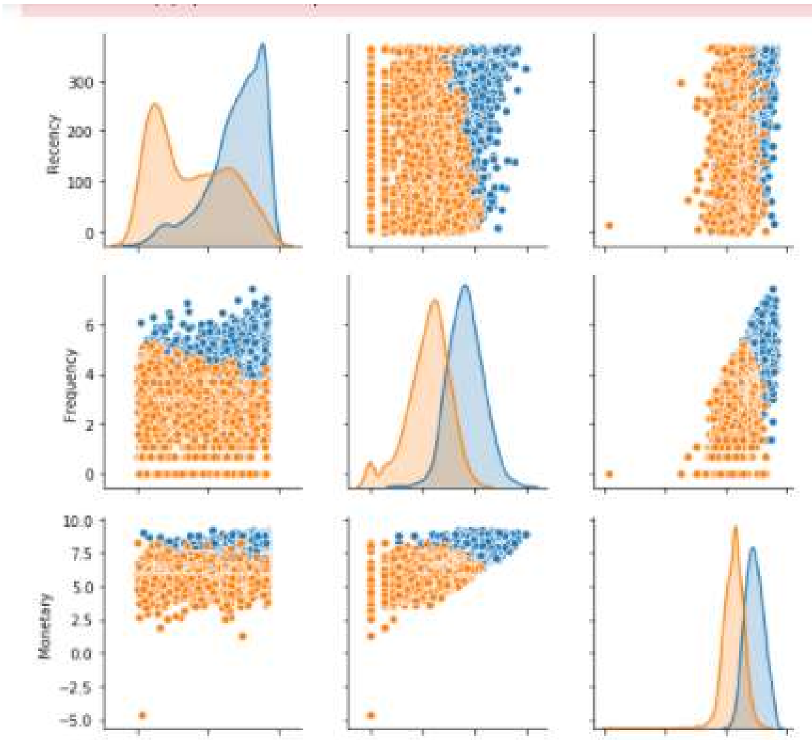
```python
    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7,
                c=colors, edgecolor='k')

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="white", alpha=1, s=200, edgecolor='k')
```

```python
    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                    s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')

plt.show()
```

```
Automatically created module for IPython interactive environment
For n_clusters = 2 The average silhouette_score is : 0.38013836380490273
For n_clusters = 3 The average silhouette_score is : 0.3742839206122452
For n_clusters = 4 The average silhouette_score is : 0.33274966031463526
For n_clusters = 5 The average silhouette_score is : 0.31561443144249934
For n_clusters = 6 The average silhouette_score is : 0.3094804251916038
For n_clusters = 7 The average silhouette_score is : 0.27931271503121946
For n_clusters = 8 The average silhouette_score is : 0.27027420717276895
For n_clusters = 9 The average silhouette_score is : 0.2686337128256559
For n_clusters = 10 The average silhouette_score is : 0.2648362310079509
For n_clusters = 11 The average silhouette_score is : 0.2716967605982414
For n_clusters = 12 The average silhouette_score is : 0.2693955775876731
For n_clusters = 13 The average silhouette_score is : 0.2586566799552891
For n_clusters = 14 The average silhouette_score is : 0.2600270893983479
For n_clusters = 15 The average silhouette_score is : 0.2551969236061192
For n_clusters = 16 The average silhouette_score is : 0.25894566745866177
```

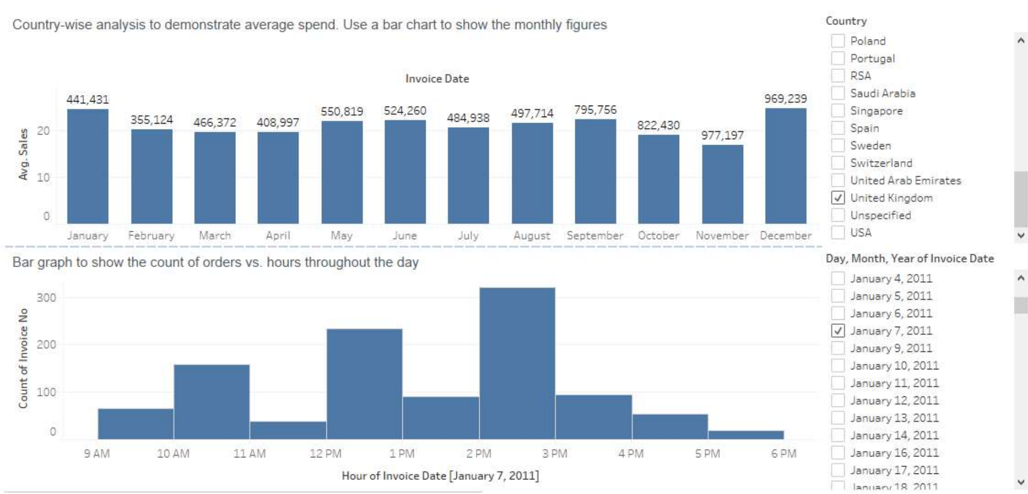
[57]:

```python
#Since the silhouette score is more at clusters 2, we will use the value
model = KMeans(n_clusters=2)
model.fit(data2)
group = model.predict(data2)
```



Recency
Frequency
Monetary

Tableau:



Country-wise analysis to demonstrate average spend. Use a bar chart to show the monthly figures
Invoice Date
Avg. Sales
441,431
355,124
466,372
408,997
550,819
524,260
484,938
497,714
795,756
822,430
977,197
969,239
January
February
March
April
May
June
July
August
September
October
November
December
Bar graph to show the count of orders vs. hours throughout the day
Count of Invoice No
9 AM
10 AM
11 AM
12 PM
1 PM
2 PM
3 PM
4 PM
5 PM
6 PM
Hour of Invoice Date [January 7, 2011]
Country
Poland
Portugal
RSA
Saudi Arabia
Singapore
Spain
Sweden
Switzerland
United Arab Emirates
United Kingdom
Unspecified
USA
Day, Month, Year of Invoice Date
January 4, 2011
January 5, 2011
January 6, 2011
January 7, 2011
January 9, 2011
January 10, 2011
January 11, 2011
January 12, 2011
January 13, 2011
January 14, 2011
January 16, 2011
January 17, 2011

Plot the distribution of RFM values using histogram and frequency charts

Plot error (cost) vs. number of clusters selected

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

Visualize to compare the RFM values of the clusters using heatmap

Measure Values

25 2,177

Group	Avg. Frequency	Avg. Monetary	Avg. Recency
0	25	404	151
1	136	2,177	280