Exploratory Data Analysis

In [ ]:

Menu

Imports

In [3]:
```
import pandas as pd
import numpy as np
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
import os

#turn off warnings
import warnings
warnings.filterwarnings('ignore')
```

Read File

In [4]:
```
data_path = '../cleaned_data'

file_path = data_path + '/clean_sales_df.csv'
```

In [5]:
```
dates = ['InvoiceDate']
data = pd.read_csv(file_path,index_col=0,header=0,parse_dates = dates)
```

In [473]:
```
data.head()
```

Out[473]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	Date	I
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085	United Kingdom	2009-12-01 00:00:00	
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	2009-12-01 00:00:00	
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	2009-12-01 00:00:00	
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085	United Kingdom	2009-12-01 00:00:00	
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085	United Kingdom	2009-12-01 00:00:00	

In [474]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 770448 entries, 0 to 1067369
Data columns (total 17 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Invoice      770448 non-null  int64
 1   StockCode    770448 non-null  object
 2   Description  770448 non-null  object
 3   Quantity     770448 non-null  int64
 4   InvoiceDate  770448 non-null  datetime64[ns]
 5   Price        770448 non-null  float64
 6   Customer ID  770448 non-null  int64
 7   Country      770448 non-null  object
 8   Date         770448 non-null  object
 9   Revenue      770448 non-null  float64
 10  Year         770448 non-null  int64
 11  Month        770448 non-null  int64
 12  Day          770448 non-null  int64
 13  Quarter      770448 non-null  int64
 14  Week         770448 non-null  int64
 15  Week_day     770448 non-null  int64
 16  Day_of_year  770448 non-null  int64
dtypes: datetime64[ns](1), float64(2), int64(10), object(4)
memory usage: 105.8+ MB
```

back to menu

Customer Cohorts

In [70]:
```
#make a copy of data to work with
df_cohort = data.copy()
df_cohort.shape
```

Out[70]: (770448, 17)

How many customers are there?

In [71]:
```
print('The number of customers = ',df_cohort.groupby('Customer ID')['Custom
```

```
The number of customers =  5835
```

```
>create a new column for all the order months
```

In [72]:
```
"""extracts the month from the date column, takes in a date returns the mon
    subtracted from invoice month

parameters:
    takes in the date
Returns:
    year and month from date, the day will be the first day of month
"""
def get_month(d):
    return datetime.datetime(d.year,d.month,1)

df_cohort['order_month'] = df_cohort['InvoiceDate'].apply(get_month)
```

In [73]:
```
df_cohort.tail()
```

Out[73]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	Da
1067365	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680	France	201 12- 00:00:
1067366	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680	France	201 12- 00:00:
1067367	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680	France	201 12- 00:00:
1067368	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680	France	201 12- 00:00:
1067369	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680	France	201 12- 00:00:

> Time based cohorts, group customers based on the month they made their first purchase

In [74]:
```
#group by customer id and invoice_month to find the month the customers mad
df_cohort['cohort_month'] = df_cohort.groupby('Customer ID')['order_month']
df_cohort.head()
```

Out[74]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	Date	I
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085	United Kingdom	2009-12-01 00:00:00	
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	2009-12-01 00:00:00	
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	2009-12-01 00:00:00	
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085	United Kingdom	2009-12-01 00:00:00	
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085	United Kingdom	2009-12-01 00:00:00	

> I need to create the cohort index which is the number of months since the first invoice, so I have to extract the month and year from the invoice

In [75]:
```
"""Extract the year and month from invoice

parameter:
    Takes in the dataframe and the column that has the date where year and

Returns:
    Year, month, and day that are assigned to objects
"""
#create a function that extracts the year and month from invoice and cohort
def get_date(df,col):
    year = df[col].dt.year
    month = df[col].dt.month
    day = df[col].dt.day
    return year,month,day
```

In [76]:
```
invoice_year,invoice_month,_ = get_date(df_cohort,'order_month')
cohort_year,cohort_month,_ = get_date(df_cohort,'cohort_month')
```

In [77]:
```
year_diff = invoice_year - cohort_year
month_diff = invoice_month - cohort_month

df_cohort['cohort_index'] = year_diff*12 + month_diff +1
```

In [78]:
```
df_cohort.head()
```

Out[78]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	Date	I
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085	United Kingdom	2009-12-01 00:00:00	
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	2009-12-01 00:00:00	
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	2009-12-01 00:00:00	
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085	United Kingdom	2009-12-01 00:00:00	
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085	United Kingdom	2009-12-01 00:00:00	

In [79]:
```
#group by cohort_month and cohort_index and count the number of unique cust
groups = df_cohort.groupby(['cohort_month','cohort_index'])
cohorts = groups['Customer ID'].nunique()
cohorts = cohorts.reset_index()

cohorts.head()
```

Out[79]:

	cohort_month	cohort_index	Customer ID
0	2009-12-01	1	949
1	2009-12-01	2	330
2	2009-12-01	3	317
3	2009-12-01	4	403
4	2009-12-01	5	359

> It is better to do the same thing using functions, that way I can choose the period for cohorts

In [22]:
```
"""
Function takes a dataframe of transactions and returns aquisition cohort an
used in customer analysis and cohort analysis matrix

cohort_month is customer aquisition month
order_month is when customer makes an order

parameters:
    Takes in a dataframe, required columns are customer id and invoice date
    also takes in a period which takes values M (monthly), Q (quaterly), Y
Returns:
    A dataframe with customer id, invoiceDate cohort_month (or month of aqu


"""
def get_cohort(df,period='M'):
    df = df[['Customer ID', 'InvoiceDate']]
    df = df.assign(cohort_month=df.groupby('Customer ID')['InvoiceDate'].tr
    df = df.assign(order_month=df['InvoiceDate'].dt.to_period(period))
    return df
```

In [519]:
```
get_cohort(data,'M').head()
```

Out[519]:

	Customer ID	InvoiceDate	cohort_month	order_month
1067365	12680	2011-12-09 12:50:00	2011-08	2011-12
1067366	12680	2011-12-09 12:50:00	2011-08	2011-12
1067367	12680	2011-12-09 12:50:00	2011-08	2011-12
1067368	12680	2011-12-09 12:50:00	2011-08	2011-12
1067369	12680	2011-12-09 12:50:00	2011-08	2011-12

In [35]:
```
rt attrgtter to obtain the integer values of subtracting date periods
operator import attrgetter

lculates the retention of customers in each month after aquisition and retu

eters:
ataframe that has purchase history for customers
eriod either M, Q

ns:
 dataframe with customer_count, invoiceDate cohort_month (or month of aquis

et_retention(df, period = 'M'):
f= get_cohort(df,period).groupby(['cohort_month','order_month']).agg({
                                        'Customer ID':'nunique'}).reset_index(dr

f['cohort_index'] = df['order_month'] - df['cohort_month']

'Subtraction of a Period from another Period will give a DateOffset, which l
'so I need to apply attrgetter which will return the integer for the period
f['cohort_index'] = df['cohort_index'].apply(attrgetter('n'))+1 # n stands
f.rename(columns={'Customer ID':'customer_count'},inplace=True)
eturn df[['cohort_month','cohort_index','customer_count']]
```

In [82]:
```
#copy the dataframe incase there is an error
df_cohort2 = data.copy()
```

In [83]:
```
#call get_retention function on dataframe
cohort2 = get_retention(df_cohort2,'M')
cohort2
```

Out[83]:

	cohort_month	cohort_index	customer_count
0	2009-12	1	949
1	2009-12	2	330
2	2009-12	3	317
3	2009-12	4	403
4	2009-12	5	359
...	...	...	...
320	2011-10	2	70
321	2011-10	3	35
322	2011-11	1	191
323	2011-11	2	27
324	2011-12	1	27

325 rows × 3 columns

In [80]:
```
#compare new dataframe cohort2 to cohort dataframe created earlier
cohorts
```

Out[80]:

	cohort_month	cohort_index	Customer ID
0	2009-12-01	1	949
1	2009-12-01	2	330
2	2009-12-01	3	317
3	2009-12-01	4	403
4	2009-12-01	5	359
...	...	...	...
320	2011-10-01	2	70
321	2011-10-01	3	35
322	2011-11-01	1	191
323	2011-11-01	2	27
324	2011-12-01	1	27

325 rows × 3 columns

they are exactly identical

In [84]:
```
cohort_q = get_retention(df_cohort2,'Q')
cohort_q.head()
```

Out[84]:

	cohort_month	cohort_index	customer_count
0	2009Q4	1	949
1	2009Q4	2	597
2	2009Q4	3	616
3	2009Q4	4	562
4	2009Q4	5	662

In [ ]:
```

```

Back to Menu

Customer Retension, or Time Based Cohorts

Time based cohorts groups customers by specific time frames.
To create the cohort dataframe pivot the cohorts dataframe with starting_month as index, the cohort_index as columns and unique number of customer id as values

In [85]:
```
counts = cohort2.pivot(index='cohort_month',columns='cohort_index',values='

counts
```

Out[85]:

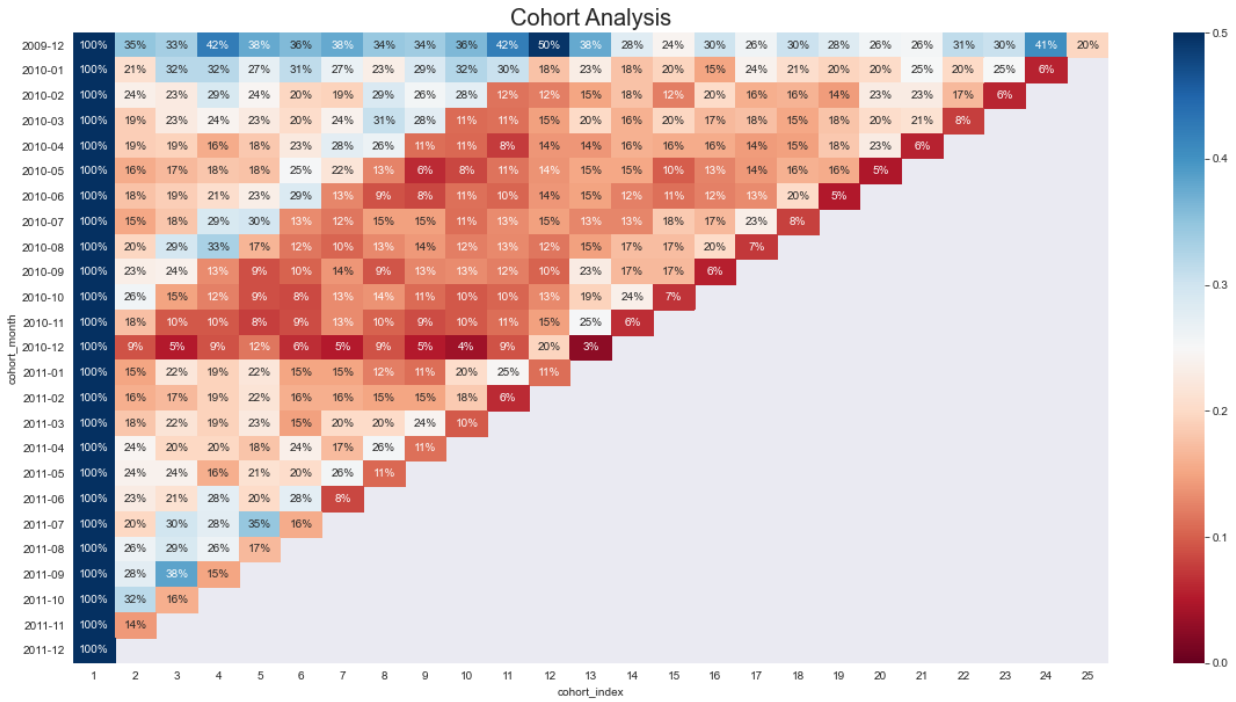
cohort_index	1	2	3	4	5	6	7	8	9	10	...	16	17
cohort_month													
2009-12	949.0	330.0	317.0	403.0	359.0	342.0	358.0	327.0	321.0	344.0	...	287.0	248.0
2010-01	364.0	76.0	116.0	116.0	99.0	114.0	97.0	85.0	105.0	118.0	...	55.0	87.0
2010-02	375.0	88.0	85.0	109.0	92.0	74.0	72.0	108.0	96.0	103.0	...	76.0	60.0
2010-03	441.0	83.0	100.0	107.0	100.0	90.0	106.0	136.0	122.0	48.0	...	74.0	77.0
2010-04	292.0	55.0	56.0	46.0	53.0	66.0	81.0	77.0	32.0	31.0	...	46.0	42.0
2010-05	254.0	40.0	43.0	46.0	45.0	64.0	55.0	32.0	16.0	21.0	...	32.0	36.0
2010-06	266.0	47.0	50.0	55.0	62.0	76.0	34.0	24.0	22.0	30.0	...	33.0	35.0
2010-07	189.0	29.0	34.0	55.0	56.0	25.0	22.0	28.0	28.0	21.0	...	32.0	44.0
2010-08	163.0	32.0	48.0	54.0	27.0	19.0	17.0	21.0	23.0	20.0	...	33.0	12.0
2010-09	236.0	55.0	56.0	31.0	21.0	24.0	33.0	22.0	30.0	31.0	...	13.0	NaN
2010-10	370.0	96.0	55.0	46.0	33.0	31.0	49.0	51.0	40.0	35.0	...	NaN	NaN
2010-11	325.0	57.0	31.0	31.0	25.0	28.0	43.0	33.0	28.0	31.0	...	NaN	NaN
2010-12	77.0	7.0	4.0	7.0	9.0	5.0	4.0	7.0	4.0	3.0	...	NaN	NaN
2011-01	73.0	11.0	16.0	14.0	16.0	11.0	11.0	9.0	8.0	15.0	...	NaN	NaN
2011-02	125.0	20.0	21.0	24.0	27.0	20.0	20.0	19.0	19.0	23.0	...	NaN	NaN
2011-03	177.0	32.0	38.0	34.0	40.0	26.0	36.0	36.0	43.0	17.0	...	NaN	NaN
2011-04	106.0	26.0	21.0	21.0	19.0	25.0	18.0	27.0	12.0	NaN	...	NaN	NaN
2011-05	113.0	27.0	27.0	18.0	24.0	23.0	29.0	12.0	NaN	NaN	...	NaN	NaN
2011-06	109.0	25.0	23.0	30.0	22.0	30.0	9.0	NaN	NaN	NaN	...	NaN	NaN
2011-07	101.0	20.0	30.0	28.0	35.0	16.0	NaN	NaN	NaN	NaN	...	NaN	NaN
2011-08	107.0	28.0	31.0	28.0	18.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
2011-09	185.0	51.0	71.0	28.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
2011-10	220.0	70.0	35.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
2011-11	191.0	27.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
2011-12	27.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN

25 rows × 25 columns

In [86]:
```
counts.to_clipboard()
```

In [554]:
```
#calculate the retention rate
cohort_size = counts.iloc[:,0]
retention = counts.divide(cohort_size,axis=0)
retention = retention.round(3)
```

In [562]:
```
#to see customer churn in two years
plt.figure(figsize=(20,10))
plt.title('Cohort Analysis', fontsize=20)
sns.heatmap(retention,
            annot=True,
            fmt='.0%',
            vmin=0,
            vmax=0.5,
            cmap='RdBu');
```



>From the heatmap

In [88]:
```
counts_q = cohort_q.pivot(index='cohort_month',columns='cohort_index',value
counts_q
```

Out[88]:

cohort_index	1	2	3	4	5	6	7	8	9
cohort_month									
2009Q4	949.0	597.0	616.0	562.0	662.0	470.0	472.0	458.0	532.0
2010Q1	1180.0	607.0	578.0	597.0	435.0	441.0	455.0	416.0	NaN
2010Q2	812.0	318.0	357.0	178.0	252.0	231.0	252.0	NaN	NaN
2010Q3	588.0	270.0	151.0	153.0	188.0	184.0	NaN	NaN	NaN
2010Q4	772.0	159.0	187.0	164.0	272.0	NaN	NaN	NaN	NaN
2011Q1	375.0	150.0	121.0	133.0	NaN	NaN	NaN	NaN	NaN
2011Q2	328.0	140.0	126.0	NaN	NaN	NaN	NaN	NaN	NaN
2011Q3	393.0	201.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011Q4	438.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

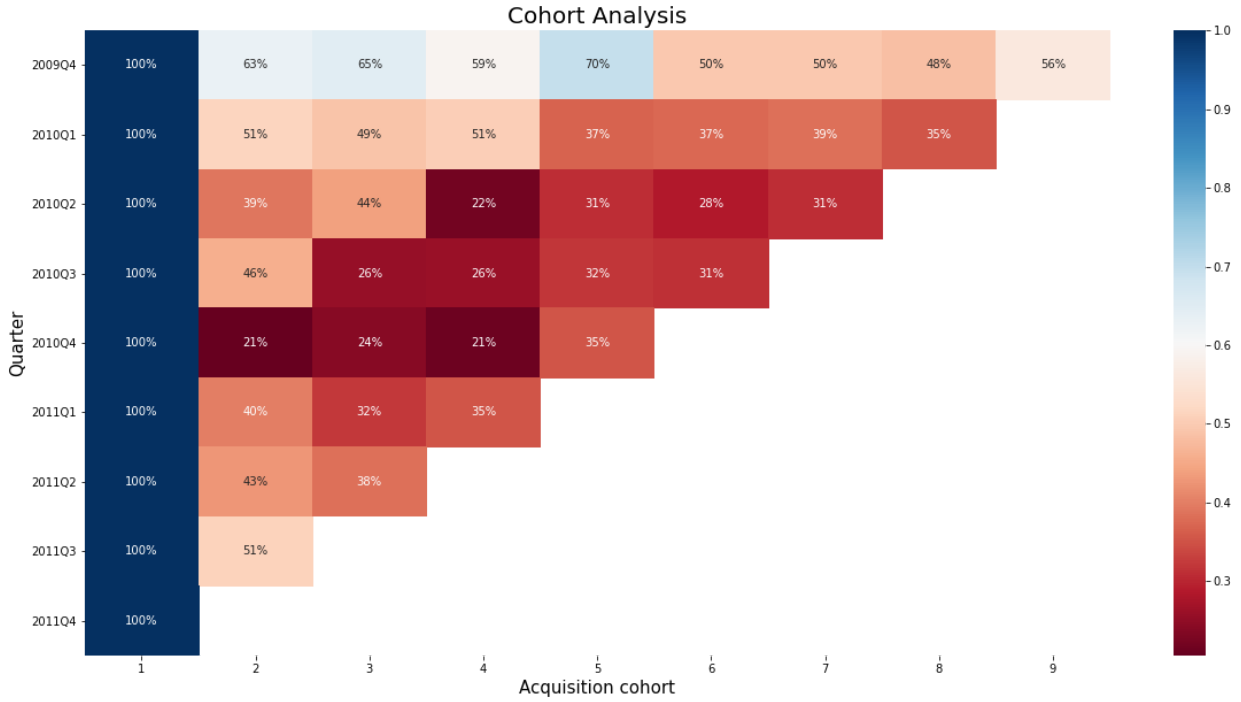
In [92]:
```
retention_q = counts_q.divide(counts_q.iloc[:,0],axis=0)
#retention_q = retention_q * 100
```

In [93]:
```
retention_q
```

Out[93]:

cohort_index	1	2	3	4	5	6	7	8	9
cohort_month									
2009Q4	1.0	0.629083	0.649104	0.592202	0.697576	0.495258	0.497366	0.482613	0.56059
2010Q1	1.0	0.514407	0.489831	0.505932	0.368644	0.373729	0.385593	0.352542	NaN
2010Q2	1.0	0.391626	0.439655	0.219212	0.310345	0.284483	0.310345	NaN	NaN
2010Q3	1.0	0.459184	0.256803	0.260204	0.319728	0.312925	NaN	NaN	NaN
2010Q4	1.0	0.205959	0.242228	0.212435	0.352332	NaN	NaN	NaN	NaN
2011Q1	1.0	0.400000	0.322667	0.354667	NaN	NaN	NaN	NaN	NaN
2011Q2	1.0	0.426829	0.384146	NaN	NaN	NaN	NaN	NaN	NaN
2011Q3	1.0	0.511450	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011Q4	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [97]:
```
#to see quarterly customer churn
plt.figure(figsize=(20,10))
plt.title('Cohort Analysis', fontsize=20)
sns.heatmap(retention_q ,
            annot=True,
           fmt='.0%',
            cmap='RdBu',
            square=False)
plt.xlabel("Acquisition cohort",fontsize=15)
plt.ylabel("Quarter",fontsize=15)
plt.yticks(rotation=3600);
```



Back to Menu

Cohorts by Quantity

Find cohorts for qunatity

In [381]:
```
#use groups where data is grouped by starting month and cohort index
cohorts_quantity = groups['Quantity'].mean()
cohorts_quantity = cohorts_quantity.reset_index()
counts_quantity = cohorts_quantity.pivot(index='starting_month', columns='c
counts_quantity.index = pd.Series(counts_quantity.index).dt.date
counts_quantity
```

Out[381]:

cohort_index	1	2	3	4	5	6	7	
starting_month								
2009-12-01	13.170201	23.564968	21.455408	22.915848	13.484278	14.911626	14.365262	14.357
2010-01-01	9.649686	14.608108	9.770020	13.315420	13.124953	13.347418	12.094064	12.686
2010-02-01	10.110324	10.164062	16.322460	13.297696	10.322200	12.388014	12.352587	10.781
2010-03-01	10.347095	11.000000	11.551850	13.028485	10.802438	13.562606	13.209404	11.792
2010-04-01	10.783380	8.991254	10.621313	14.777320	12.715290	10.619926	10.353885	10.049
2010-05-01	12.365226	9.489717	10.344961	12.630582	13.157895	9.487756	7.634234	6.878
2010-06-01	10.588939	8.973659	13.529930	10.802518	11.467917	14.178734	11.364812	17.889
2010-07-01	8.273721	9.057199	7.294813	8.979180	7.752495	10.054104	9.732143	9.300
2010-08-01	8.426244	8.080910	8.087692	8.151358	10.344498	7.617925	9.065217	8.296
2010-09-01	27.825662	9.619368	8.344894	11.013141	18.968696	8.048237	14.304348	12.247
2010-10-01	9.323815	6.805456	6.958425	9.905546	6.856540	5.310547	6.615176	6.328
2010-11-01	7.897890	9.322670	11.086226	8.284483	7.849791	9.522606	9.010435	13.265
2010-12-01	8.228125	10.044872	27.830645	8.059701	16.907609	4.300000	9.111111	21.949
2011-01-01	8.085018	4.631429	5.223077	45.973077	11.373810	11.691318	8.458472	15.420
2011-02-01	10.304444	19.490148	37.288026	10.055629	12.375000	7.935268	13.931122	18.362
2011-03-01	9.406102	11.990502	15.053047	9.079876	15.043194	13.019749	13.806233	13.692
2011-04-01	8.707668	9.024862	6.798851	9.197652	8.035912	8.627688	6.315399	9.404
2011-05-01	11.319474	10.978261	14.061386	13.535604	10.828076	9.013006	12.664901	8.498
2011-06-01	9.504752	17.946367	10.458937	13.200000	9.349630	10.066815	10.281553	
2011-07-01	9.003435	12.860724	6.610075	7.799591	6.013353	7.149038	NaN	
2011-08-01	9.712936	6.115521	5.037238	5.693073	6.781818	NaN	NaN	
2011-09-01	10.726237	6.447735	7.902130	8.135705	NaN	NaN	NaN	
2011-10-01	8.719118	7.419573	8.195545	NaN	NaN	NaN	NaN	
2011-11-01	10.559648	10.231604	NaN	NaN	NaN	NaN	NaN	
2011-12-01	20.294788	NaN	NaN	NaN	NaN	NaN	NaN	

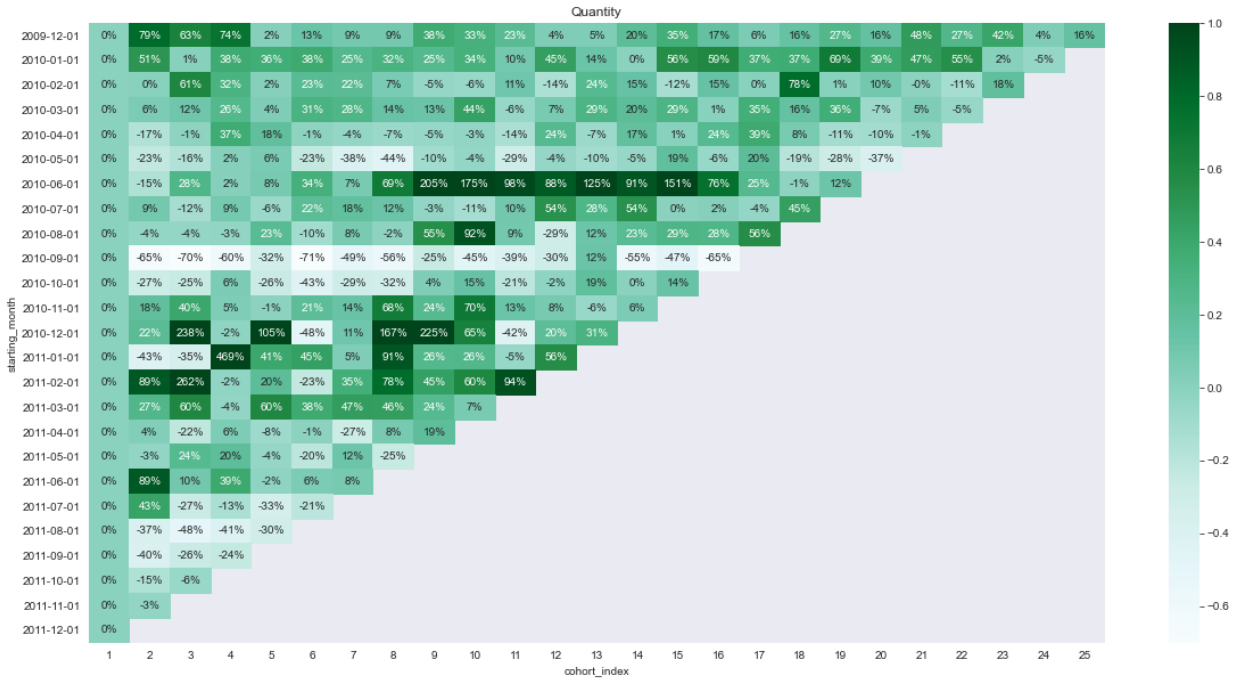
25 rows × 25 columns

In [390]:
```
#% of quantity
quantity_size = counts_quantity.iloc[:,0]
quantity_retention = round(counts_quantity.subtract(quantity_size,axis=0),2
quantity_retention = quantity_retention.divide(quantity_size,axis=0)
quantity_retention.max().max()
```

Out[390]:
```
4.6864458835598874
```

In [393]:
```
plt.figure(figsize=(20,10))
plt.title('Quantity')
sns.heatmap(quantity_retention,
            annot=True,
            fmt='.0%',
            vmin=-0.7,
            vmax=1,
           cmap='BuGn')
```

Out[393]:
```
<AxesSubplot:title={'center':'Quantity'}, xlabel='cohort_index', ylabel
='starting_month'>
```



In [225]:
```
quantity_retention
```

Out[225]:

cohort_index	1	2	3	4	5	6	7	8	
starting_month									
2009-12-01	0.0	-0.070143	0.006525	0.013050	0.048937	0.084824	-0.047306	0.066881	0.1
2010-01-01	0.0	0.043536	-0.017150	0.019789	0.102903	-0.001319	0.084433	0.116095	0.1
2010-02-01	0.0	-0.120417	0.086550	0.090313	0.040139	0.169336	0.173099	0.126689	-0.0
2010-03-01	0.0	0.035924	0.017962	-0.019245	0.061583	0.153958	0.111620	-0.029509	-0.0
2010-04-01	0.0	-0.141740	0.041294	0.200891	0.014509	0.056919	-0.024553	-0.043526	-0.0
2010-05-01	0.0	-0.007879	0.308610	0.315176	0.569943	0.066975	-0.131323	-0.165467	0.0
2010-06-01	0.0	-0.018204	0.180742	0.106625	0.105324	-0.161237	-0.078018	0.256159	0.0
2010-07-01	0.0	0.253663	-0.005545	0.231485	-0.048515	-0.005545	0.067921	0.052673	0.0
2010-08-01	0.0	-0.029174	0.007956	0.018565	0.267866	-0.039782	0.102107	0.262562	-0.1
2010-09-01	0.0	-0.202611	-0.204981	-0.114932	0.072277	-0.181284	-0.175359	-0.146923	0.0
2010-10-01	0.0	-0.264296	-0.180372	0.169099	-0.161584	-0.361997	-0.190393	-0.219202	-0.1
2010-11-01	0.0	0.108014	0.426219	-0.026274	-0.242303	0.078821	-0.121151	0.143046	0.1
2010-12-01	0.0	0.248628	1.173167	-0.162278	0.863499	-0.360287	0.355821	1.037687	0.0
2011-01-01	0.0	-0.407540	-0.390380	0.238804	0.168736	0.386090	0.095808	0.669223	0.1
2011-02-01	0.0	-0.031446	-0.111947	-0.171064	0.270433	0.000000	0.286784	0.454075	-0.0
2011-03-01	0.0	-0.025521	0.097223	-0.093577	0.401043	0.224827	0.251563	-0.082639	0.1
2011-04-01	0.0	0.002457	-0.185485	0.114239	-0.029481	0.041765	-0.244447	-0.090900	0.1
2011-05-01	0.0	-0.014945	0.171297	0.152903	0.225330	-0.041387	0.196589	-0.050584	
2011-06-01	0.0	0.338564	0.174743	0.308530	0.116040	0.030034	0.300339	NaN	
2011-07-01	0.0	0.510741	-0.142952	-0.047188	-0.249819	-0.065230	NaN	NaN	
2011-08-01	0.0	-0.474759	-0.518634	-0.409507	-0.246379	NaN	NaN	NaN	
2011-09-01	0.0	-0.348536	-0.191418	-0.109540	NaN	NaN	NaN	NaN	
2011-10-01	0.0	-0.109171	-0.039575	NaN	NaN	NaN	NaN	NaN	
2011-11-01	0.0	-0.152399	NaN	NaN	NaN	NaN	NaN	NaN	
2011-12-01	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

25 rows × 25 columns

In [401]:
```
def get_cohorts(df,period='M'):
    df = data[['Customer ID','Invoice','Date','Revenu']]
    df.head()
```

In [405]:
```
df = data[['Customer ID','Invoice','Date','Revenue']]

df= df.assign(order_cohort = data.groupby('Customer ID')['Revenue'].transfo
df.head()
```

Out[405]:

	Customer ID	Invoice	Date	Revenue	order_cohort
0	13085	489434	2009-12-01	83.4	10.2
1	13085	489434	2009-12-01	81.0	10.2
2	13085	489434	2009-12-01	81.0	10.2
3	13085	489434	2009-12-01	100.8	10.2
4	13085	489434	2009-12-01	30.0	10.2

In [ ]:
```
```

Back to Menu

Recency, Frequency, Monetary Value

> Recency, Frequency and Monetary value for each customer. We want our customer to be recent and active
> recency: means number of days since the last transaction
> frequency: means number of transactions in the past period, for me 2 years
> monetary value: means how much the customer has spent

In [176]:
```
#first remove the guest from the dataframe
data_no_guest = data[data['Customer ID'] !=1]
```

In [177]:
```
data_no_guest.shape
```

Out[177]:
```
(746373, 20)
```

In [178]:
```
print('min: {},max: {}'.format(data_no_guest.InvoiceDate.min(),data_no_gues
```

```
min: 2009-12-01 07:45:00,max: 2011-12-09 12:50:00
```

> I'll create a hypothetical date to represent current date, so I will take the highest date and add one day to it. This data represents the current date on which we do analysis

In [179]:
```
snapshot_date = max(data_no_guest.InvoiceDate) + datetime.timedelta(days=1)
snapshot_date
```

Out[179]:
```
Timestamp('2011-12-10 12:50:00')
```

> Create a dataframe for recency, activity and revenue for each customer. Recency measures the days since most recent invoice, Invoice measures the number of invoices per customer, which is activity, and Revenue measures the sum of all revenue per customer.

In [180]:
```
data_rfm = data_no_guest.groupby('Customer ID').agg({\
                                                'InvoiceDate': lambda x: (snapsho
                                                 'Invoice': 'count',\
                                                 'Revenue': 'sum'})
data_rfm.head()
```

Out[180]:

Customer ID	InvoiceDate	Invoice	Revenue
12346	529	24	169.36
12347	2	221	4671.93
12348	249	27	421.76
12349	19	170	3498.94
12350	310	16	294.40

In [181]:
```
#rename some columns
data_rfm.rename(columns={'InvoiceDate':'Recency',\
                         'Invoice': 'Frequency',\
                          'Revenue':'MonetaryValue'},inplace=True)
print(data_rfm.shape)
data_rfm.head()
```

```
(5731, 3)
```

Out[181]:

Customer ID	Recency	Frequency	MonetaryValue
12346	529	24	169.36
12347	2	221	4671.93
12348	249	27	421.76
12349	19	170	3498.94
12350	310	16	294.40

- Now I have to give the customers labels according to their recency, frequency

and monetary value. I do this by calculating the quartiles for eah customer.

- Recency: measures days since last transaction, so customers who have been active more recently will be rated better than less recent ones.
- So the dataframe will be sorted into 4 quanrtiles from 1 to 4 by the recency of customers, 4 being more recent

In [182]:
```
labels = range(4,0,-1)
recency_q = pd.qcut(data_rfm['Recency'],q=4,labels=labels)
data_rfm['R'] = recency_q.values
data_rfm.head()
```

Out[182]:

Customer ID	Recency	Frequency	MonetaryValue	R
12346	529	24	169.36	1
12347	2	221	4671.93	4
12348	249	27	421.76	2
12349	19	170	3498.94	4
12350	310	16	294.40	2

In [ ]:
```
#the higher frequency the better
labels_f = range(1,5)
frequency_q = pd.qcut(data_rfm['Frequency'],q=4,labels = labels_f)
data_rfm = data_rfm.assign(F = frequency_q.values)
```

In [184]:
```
data_rfm.head()
```

Out[184]:

Customer ID	Recency	Frequency	MonetaryValue	R	F
12346	529	24	169.36	1	2
12347	2	221	4671.93	4	4
12348	249	27	421.76	2	2
12349	19	170	3498.94	4	4
12350	310	16	294.40	2	1

In [188]:
```
#higher monetary value is better
labels_m = range(1,5)
mon_val_q = pd.qcut(data_rfm['MonetaryValue'],q=4, labels = labels_m)
data_rfm['M'] = mon_val_q.values
data_rfm.head()
```

Out[188]:

Customer ID	Recency	Frequency	MonetaryValue	R	F	M
12346	529	24	169.36	1	2	1
12347	2	221	4671.93	4	4	4
12348	249	27	421.76	2	2	2
12349	19	170	3498.94	4	4	4
12350	310	16	294.40	2	1	1

Now I want to concatenate the three values R,F, and M

In [196]:
```
#function that takes the values and concatenates them
def concat_rfm(df):
    return str(df.R)+str(df.F)+str(df.M)
data_rfm['RFM_segment'] = data_rfm.apply(concat_rfm,axis=1)
data_rfm.head()
```

Out[196]:

Customer ID	Recency	Frequency	MonetaryValue	R	F	M	RFM_segment
12346	529	24	169.36	1	2	1	121
12347	2	221	4671.93	4	4	4	444
12348	249	27	421.76	2	2	2	222
12349	19	170	3498.94	4	4	4	444
12350	310	16	294.40	2	1	1	211

In [238]:
```
#sum up the values
data_rfm['RFM_score'] = data_rfm[['R','F','M']].sum(axis=1)
data_rfm.head()
```

Out[238]:

	Customer ID	Recency	Frequency	MonetaryValue	R	F	M	RFM_segment	RFM_score
0	12346	529	24	169.36	1	2	1	121	4
1	12347	2	221	4671.93	4	4	4	444	12
2	12348	249	27	421.76	2	2	2	222	6
3	12349	19	170	3498.94	4	4	4	444	12
4	12350	310	16	294.40	2	1	1	211	4

In [263]:
```
#select top 5 customers with lowest rfm segment of 111
data_rfm[data_rfm['RFM_segment']==111][:5]
```

Out[263]:

	Customer ID	Recency	Frequency	MonetaryValue	R	F	M	RFM_segment	RFM_score
39	12387	415	9	143.94	1	1	1	111	3
44	12392	591	7	234.75	1	1	1	111	3
52	12400	414	11	205.25	1	1	1	111	3
67	12416	657	11	202.56	1	1	1	111	3
111	12460	456	17	296.65	1	1	1	111	3

In [265]:
```
#the number of customers is
data_rfm.shape[0]
```

Out[265]:
```
5731
```

In [229]:
```
#save file
data_rfm.to_csv(data_path + '/rfm_data.csv')
```

In [231]:
```
data_rfm.to_pickle(data_path + '/rfm_data.pkl')
```

In [250]:
```
#sort and see 10 largest rfm segments
data_rfm['RFM_segment'].value_counts()[:10]
```

Out[250]:
```
444    617
111    531
344    332
211    306
233    237
122    234
222    234
333    222
433    200
322    180
Name: RFM_segment, dtype: int64
```

In [284]:
```
#summary statistics
data_rfm.groupby('RFM_score').agg({\
                                        'Recency':'mean',
                                         'Frequency':'mean',
                                         'MonetaryValue':['mean','count']}).rou
```

Out[284]:

RFM_score	Recency mean	Frequency mean	MonetaryValue mean	MonetaryValue count
3	542.79	9.80	163.35	531
4	375.87	16.89	231.75	593
5	306.41	24.31	371.25	586
6	227.21	34.42	525.84	600
7	187.45	50.11	763.06	599
8	154.89	73.33	1094.10	588
9	98.41	107.13	1515.01	533
10	70.82	167.38	2564.28	547
11	36.07	261.12	4103.78	537
12	10.47	541.73	8967.93	617

It is more intuitive if the rfm segments were given names like gold, silver and bronze. I will create a function that takes a dataframe and returns segements' labels gold, silver or bronze according to their segment score

In [311]:
```
def label_segments(df):
    if df['RFM_score'] >= 9:
        return 'Gold'
    elif (df['RFM_score'] < 9) and (df['RFM_score']>= 5): #and :
        return 'Silver'
    else:
        return 'Bronze'
```

In [312]:
```
data_rfm['segment_label'] = data_rfm.apply(label_segments,axis=1)
data_rfm.head()
```

Out[312]:

	Customer ID	Recency	Frequency	MonetaryValue	R	F	M	RFM_segment	RFM_score	segment_l
0	12346	529	24	169.36	1	2	1	121	4	Br
1	12347	2	221	4671.93	4	4	4	444	12	
2	12348	249	27	421.76	2	2	2	222	6	S
3	12349	19	170	3498.94	4	4	4	444	12	
4	12350	310	16	294.40	2	1	1	211	4	Br

In [330]:
```
#now get the summary stats for these labels
data_rfm.groupby('segment_label').agg({
                    'Recency':'mean',
                    'Frequency':'mean',
                    'MonetaryValue':['mean','count']
}).round(2)
```

Out[330]:

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
segment_label				
Bronze	454.72	13.54	199.44	1124
Gold	52.38	278.93	4452.60	2234
Silver	218.81	45.53	688.35	2373

The threshold for gold, silver and bronze was chosen arbitrarily, but there is a better way to better cluster customers. Businesses use clustering for customer segmentation. The clustering results segment customers into groups with similar purchase histories, which businesses can then use to create targeted advertising campaigns. Kmeans clustering is the easiest and most used unsupervised learning method to do this. Kmeans clustering is going to be done in the notebook called Modeling

Back to Menu

In [603]:
```
data_rfm.head()
```

Out[603]:

	Customer ID	Recency	Frequency	MonetaryValue	R	F	M	RFM_segment	RFM_score	segment_l
0	12346	529	24	169.36	1	2	1	121	4	Br
1	12347	2	221	4671.93	4	4	4	444	12	
2	12348	249	27	421.76	2	2	2	222	6	S
3	12349	19	170	3498.94	4	4	4	444	12	
4	12350	310	16	294.40	2	1	1	211	4	Br

In [605]:
```
#save dataframe to file
data_rfm.to_csv(data_path+'/rfm_data.csv')
```

In [ ]:
```

```