Customer Behavior Analysis

This data set contains one-year trade information for all online orders placed on an e-commerce online trade website. The owner of this website focuses on gift sales and has many clients acting like wholesalers at the same time.

Data Cleaning ¶

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
sales_df=pd.read_excel('F:/From dish E/DA/python/CASE STUDY/Online Retail.xlsx',sheet_n
ame='Online Retail')
```

In [3]:

```
sales_df.head()
```

Out[3]:

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

In [4]:

```
sales_df.dtypes
```

Out[4]:

```
InvoiceNo              object
StockCode              object
Description            object
Quantity                int64
InvoiceDate    datetime64[ns]
UnitPrice             float64
CustomerID            float64
Country                object
dtype: object
```

InvoiceNo: The serial number of invoices. If begins with C, it means this order was canceled.
StockCode: The product code. Each product has a unique code.
CustomerID: Every customer has a unique 5 digits code.

In [5]:

```
# rename the column
sales_df.rename(columns={'InvoiceDate': 'InvoiceTime'}, inplace= True)
```

In [6]:

```
#delete the duplicated number
#If the value of all fields in two records are same,we define it as a duplicated value
 and only keep one of them.
rows_before = sales_df.shape[0]
sales_df.drop_duplicates(inplace= True)
rows_after = sales_df.shape[0]
print(rows_before,rows_after)
```

```
541909 536641
```

In [7]:

```
#reset the index
sales_df.reset_index(drop=True,inplace=True)
# view the null value
# The column 'Desctiptive' stores decription of products,we don't need to process them
sales_df.isnull().sum()
```

Out[7]:

```
InvoiceNo          0
StockCode          0
Description      1454
Quantity           0
InvoiceTime        0
UnitPrice          0
CustomerID     135037
Country            0
dtype: int64
```

In [8]:

```
sales_df[sales_df.isnull().values==True].head()
```

Out[8]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceTime	UnitPrice	Custom
605	536414	22139	NaN	56	2010-12-01 11:52:00	0.00	NaN
605	536414	22139	NaN	56	2010-12-01 11:52:00	0.00	NaN
1407	536544	21773	DECORATIVE ROSE BATHROOM BOTTLE	1	2010-12-01 14:32:00	2.51	NaN
1408	536544	21774	DECORATIVE CATS BATHROOM BOTTLE	2	2010-12-01 14:32:00	2.51	NaN
1409	536544	21786	POLKADOT RAIN HAT	4	2010-12-01 14:32:00	0.85	NaN

In [9]:

```
#check whether there is customer with ID '0'
#Since there is no customer whose 'CustomerID' is 0, we use 0 to fill all null values i
n the 'CustomerId' column.
sales_df[sales_df['CustomerID']==0]
```

Out[9]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceTime	UnitPrice	CustomerID	C

In [10]:

```
sales_df['CustomerID'].fillna(0, inplace=True)
sales_df['Description'].fillna(0, inplace=True)
(sales_df['CustomerID']==0).sum()
```

Out[10]:

135037

In [11]:

```
#unification process of time
sales_df['InvoiceTime']=pd.to_datetime(sales_df['InvoiceTime'],errors='coerce')
sales_df['Date']=pd.to_datetime(sales_df['InvoiceTime'].dt.date,errors='coerce') #inval
id parsing---set as NaN
sales_df['Month']=sales_df['Date'].dt.month
```

In [12]:

```
# variable type transformation
sales_df['Quantity']=sales_df['Quantity'].astype('int32')
sales_df['UnitPrice']=sales_df['UnitPrice'].astype('float')
sales_df['CustomerID']=sales_df['CustomerID'].astype('int32')
```

In [13]:

```
#calculate the sum price for each record, add a new column
sales_df['SumPrice']=sales_df['Quantity']*sales_df['UnitPrice']
```

In [14]:

```
# outlier processing
sales_df.describe()
```

Out[14]:

	Quantity	UnitPrice	CustomerID	Month	SumPrice
count	536641.000000	536641.000000	536641.000000	536641.000000	536641.000000
mean	9.620029	4.632656	11435.904653	7.544820	18.123861
std	219.130156	97.233118	6795.044250	3.508696	380.656263
min	-80995.000000	-11062.060000	0.000000	1.000000	-168469.600000
25%	1.000000	1.250000	0.000000	5.000000	3.750000
50%	3.000000	2.080000	14336.000000	8.000000	9.870000
75%	10.000000	4.130000	16241.000000	11.000000	17.400000
max	80995.000000	38970.000000	18287.000000	12.000000	168469.600000

In [15]:

```
query_a=sales_df['InvoiceNo'].str.contains('A')==True
sales_df.loc[query_a,:]
sales_df.drop(sales_df['InvoiceNo'].str.contains('A')==True,inplace=True)
```

Negative values occur in column 'Quantity"UnitPrice"SumPrice'
The causes of this situation was some of these orders have been cancelled or set as 0 (may because of promotions)
Orders start with C : cancelled order
Orders with total price 0: free order
Orders start with A: from the description we find these order are used to adjust bad debt, filter them.

In [16]:

```
sales_df[(sales_df['Quantity']<=0|(sales_df['UnitPrice']<0))].head()
```

Out[16]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceTime	UnitPrice	CustomerI
141	C536379	D	Discount	-1	2010-12-01 09:41:00	27.50	14527
154	C536383	35004C	SET OF 3 COLOURED FLYING DUCKS	-1	2010-12-01 09:49:00	4.65	15311
235	C536391	22556	PLASTERS IN TIN CIRCUS PARADE	-12	2010-12-01 10:24:00	1.65	17548
236	C536391	21984	PACK OF 12 PINK PAISLEY TISSUES	-24	2010-12-01 10:24:00	0.29	17548
237	C536391	21983	PACK OF 12 BLUE PAISLEY TISSUES	-24	2010-12-01 10:24:00	0.29	17548

In [17]:

```
#recoginze canceled orders
query_c = sales_df['InvoiceNo'].str.contains('C')==True
sales_cancel = sales_df.loc[query_c,:].copy()
sales_success = sales_df.loc[-query_c,:].copy()
```

In [18]:

```
#find whether cancelled orders and successful orders have intersections
sales_cancel['SrcInvoiceNo']=sales_cancel['InvoiceNo'].str.split('C',expand=True)[1]
pd.merge(sales_cancel,sales_success,left_on='SrcInvoiceNo',right_on='InvoiceNo')
```

Out[18]:

	InvoiceNo_x	StockCode_x	Description_x	Quantity_x	InvoiceTime_x	UnitPrice_x

0 rows × 23 columns

In [19]:

```
print('orders that have been cancelled:',sales_cancel.shape,'other orders:',sales_succe
ss.shape)
sales_success.describe()
sales_success=sales_success[-(sales_success['Quantity']<=0|(sales_success['UnitPrice']<
0))]
sales_success.describe()
query_free=sales_success['UnitPrice'] == 0
sales_success=sales_success.loc[-query_free,:]
#drop the cancelled&free orders and orders which quantity or unitprice is minus zero
```

```
orders that have been cancelled: (9251, 12) other orders: (527388, 11)
```

In [20]:

```
sales_success.describe() #data cleaning part is end here
```

Out[20]:

	Quantity	UnitPrice	CustomerID	Month	SumPrice
count	524876.000000	524876.000000	524876.000000	524876.00000	524876.000000
mean	10.616618	3.922576	11437.707731	7.55222	20.275408
std	156.280328	36.093096	6799.515061	3.50816	271.694084
min	1.000000	0.001000	0.000000	1.00000	0.001000
25%	1.000000	1.250000	0.000000	5.00000	3.900000
50%	4.000000	2.080000	14350.000000	8.00000	9.920000
75%	11.000000	4.130000	16245.000000	11.00000	17.700000
max	80995.000000	13541.330000	18287.000000	12.00000	168469.600000

Exploratory Data Analysis

From the perspective of orders

In [22]:

```
#Group all data by 'InvoiceNo' and calculate the sum of Quantity and SumPrice
invoice_grouped=sales_success.groupby('InvoiceNo')[['Quantity','SumPrice']].sum()
invoice_grouped.head()
#get the per order transaction and associated purchase rate by executing describe()
invoice_grouped.describe()
```

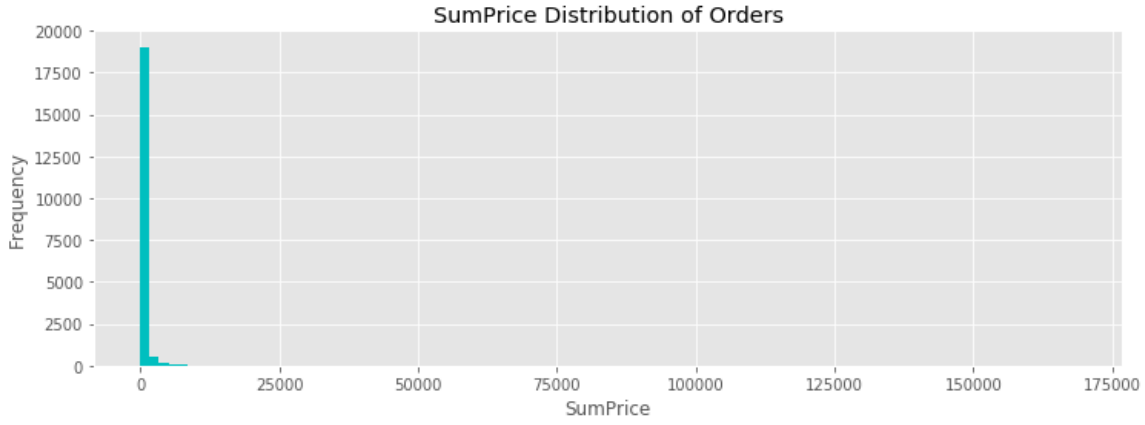
Out[22]:

	Quantity	SumPrice
count	19960.000000	19960.000000
mean	279.178758	533.170098
std	955.011964	1780.412701
min	1.000000	0.380000
25%	69.000000	151.695000
50%	150.000000	303.300000
75%	296.000000	493.462500
max	80995.000000	168469.600000

During the 1 yr statistical period, we have 19960 effective orders, per order transaction was £533.17, associated purchase rate was around 279. It indicates that the dominating business of this e-commerce website was wholesale.
The overall difference among all orders were apparent, and customers with super-strong purchasing power existed.

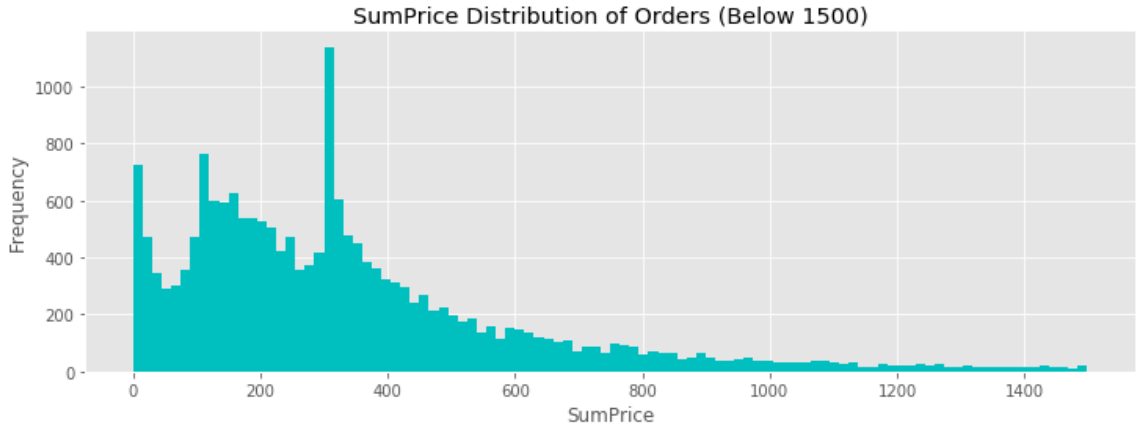
In [26]:

```
#draw a histogram named 'SumPrice Distribution of orders'
invoice_grouped['SumPrice'].hist(bins = 100, figsize = (12, 4), color = 'c')
plt.title('SumPrice Distribution of Orders')
plt.ylabel('Frequency')
plt.xlabel('SumPrice')
plt.show()
```



In [27]:

```
#some SumPrice data is too big to provide a good visualization effect, here we filter o
rders with SumPrice over 1500
invoice_grouped[invoice_grouped.SumPrice < 1500]['SumPrice'].hist(bins = 100, figsize =
(12, 4), color = 'c')
plt.title('SumPrice Distribution of Orders (Below 1500)')
plt.ylabel('Frequency')
plt.xlabel('SumPrice')
plt.show()
```



the SumPrice of orders concentrate within £400, and three peak values occurred within the following periods: within £20, £100~230, £300~320, and the highest was in £300~320

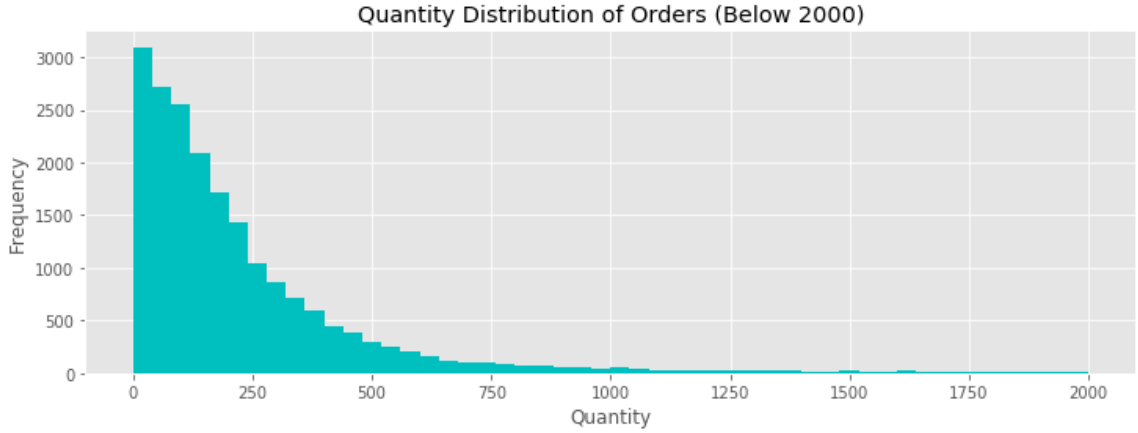
This can be a point for further exploration.

In [29]:

```
#draw a histogram named 'Quantity distribution of Orders'
invoice_grouped['Quantity'].hist(bins = 100, figsize = (12, 4), color = 'c')
plt.title('Quantity Distribution of Orders')
plt.ylabel('Frequency')
plt.xlabel('Quantity')
plt.show()
```

In [30]:

```
#some outliers make the quantity range too wide and affect the visualization effect, he
re we filter orders with quantity over 2000
invoice_grouped[invoice_grouped.Quantity < 2000]['Quantity'].hist(bins = 50, figsize =
(12, 4), color = 'c')
plt.title('Quantity Distribution of Orders (Below 2000)')
plt.ylabel('Frequency')
plt.xlabel('Quantity')
plt.show()
```

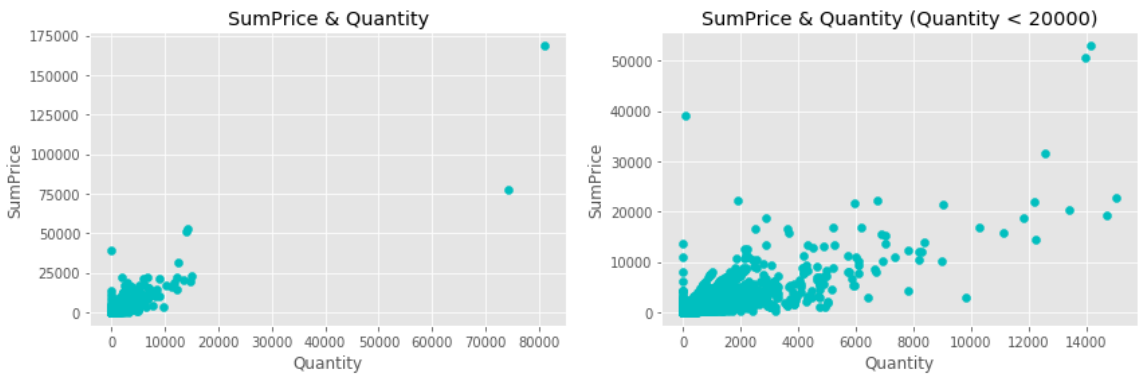


The shape of the distrbution of goods quantities showed a long-tail dustribution, most orders had the their goods quantities within 250, the more the per-order goods quantity was, the less the number of orders was.

In [32]:

```
#Draw a scatter plot to find the relationship between SumPrice and order quantity.
plt.figure(figsize=(14,4))
plt.subplot(121)
plt.scatter(invoice_grouped['Quantity'], invoice_grouped['SumPrice'], color = 'c')
plt.title('SumPrice & Quantity')
plt.ylabel('SumPrice')
plt.xlabel('Quantity')

plt.subplot(122)
plt.scatter(invoice_grouped[invoice_grouped.Quantity < 20000]['Quantity'], invoice_grou
ped[invoice_grouped.Quantity < 20000]['SumPrice'], color = 'c')
plt.title('SumPrice & Quantity (Quantity < 20000)')
plt.ylabel('SumPrice')
plt.xlabel('Quantity')
plt.show()
```



In general,the variable SumPrice has a positive correlation with Quantity. However,there are some outliers(high sumprice and low quantity).
This is a point for further exploration.

From the perspective of customers

In [34]:

```
#Here we only analyse customers with customer ID(not 0)
sales_customer=sales_success[sales_success.CustomerID!=0].copy()
#group all data by customer and InvoiceNo first
customer_grouped=sales_customer.groupby(['CustomerID','InvoiceNo'])[['Quantity','SumPri
ce']].sum().reset_index()
customer_grouped.head()
```

Out[34]:

	CustomerID	InvoiceNo	Quantity	SumPrice
0	12346	541431	74215	77183.60
1	12347	537626	319	711.79
2	12347	542237	315	475.39
3	12347	549222	483	636.25
4	12347	556201	196	382.52

In [35]:

```
#group all data by customer again
customer_grouped = customer_grouped.groupby('CustomerID').agg({'InvoiceNo':np.size,'Qua
ntity':np.sum,'SumPrice':np.sum})
customer_grouped.head()
```

Out[35]:

	InvoiceNo	Quantity	SumPrice
CustomerID			
12346	1	74215	77183.60
12347	7	2458	4310.00
12348	4	2341	1797.24
12349	1	631	1757.55
12350	1	197	334.40

In [36]:

```
customer_grouped.describe()
```

Out[36]:

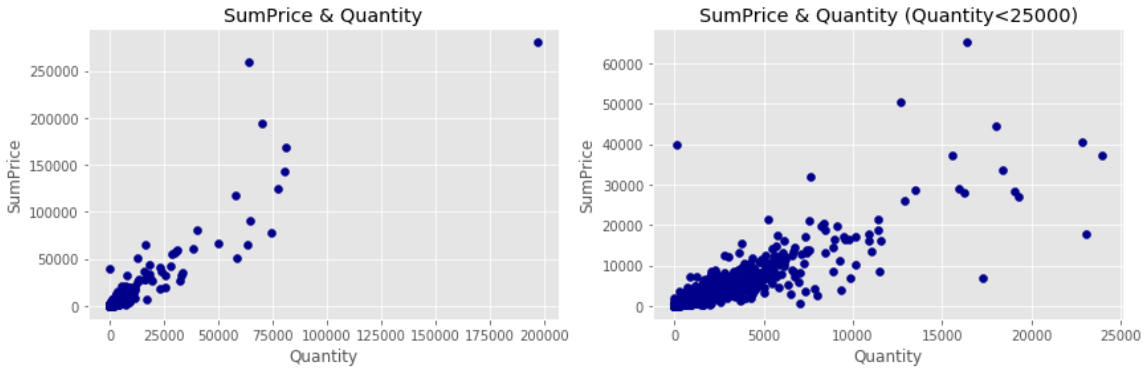
	InvoiceNo	Quantity	SumPrice
count	4338.000000	4338.000000	4338.000000
mean	4.272015	1187.641770	2048.679865
std	7.697998	5043.619358	8985.227179
min	1.000000	1.000000	3.750000
25%	1.000000	159.000000	306.482500
50%	2.000000	378.000000	668.570000
75%	5.000000	989.750000	1660.597500
max	209.000000	196915.000000	280206.020000

The mean value for per customer order amount was 4; the median was 2; also, over 25% of customers only placed their order once and didn't retain.
The per customer order amount was 1187, which exceeded the Q3 value. The customer who purchased most bought 196915 products in total.
The per customer transaction was £2049, which exceeded the Q3 value as well.
In short, noticeable purchasing power differences occurred among all customers. Customers with high expenditure made the mean value high.

In [38]:

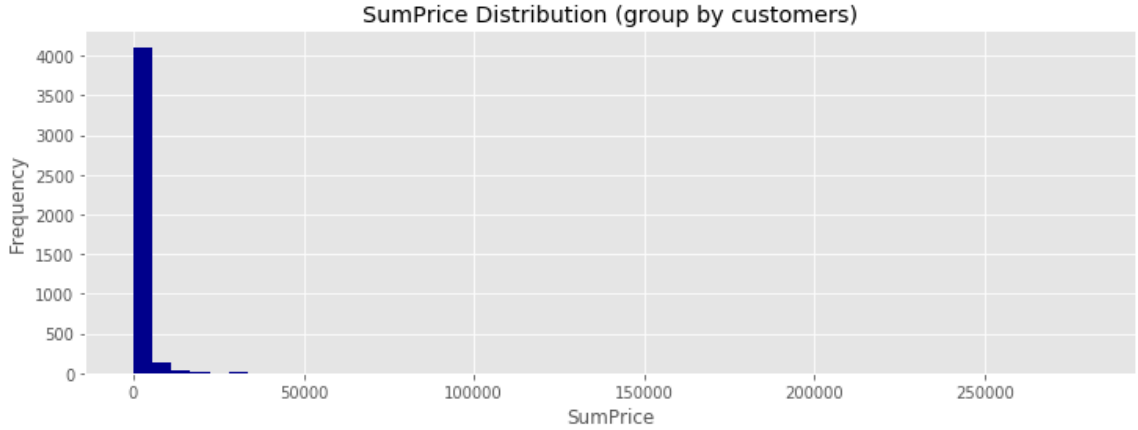
```
#draw a scatter diagram with the data related to SumPrice and Quantity(grouped by custo
mer)
plt.figure(figsize=(14,4))
plt.subplot(121)
plt.scatter(customer_grouped['Quantity'], customer_grouped['SumPrice'], color = 'darkbl
ue')
plt.title('SumPrice & Quantity')
plt.ylabel('SumPrice')
plt.xlabel('Quantity')

plt.subplot(122)
plt.scatter(customer_grouped[customer_grouped.Quantity < 25000]['Quantity'], customer_g
rouped[customer_grouped.Quantity < 25000]['SumPrice'], color = 'darkblue')
plt.title('SumPrice & Quantity (Quantity<25000)')
plt.ylabel('SumPrice')
plt.xlabel('Quantity')
plt.show()
```



In [39]:

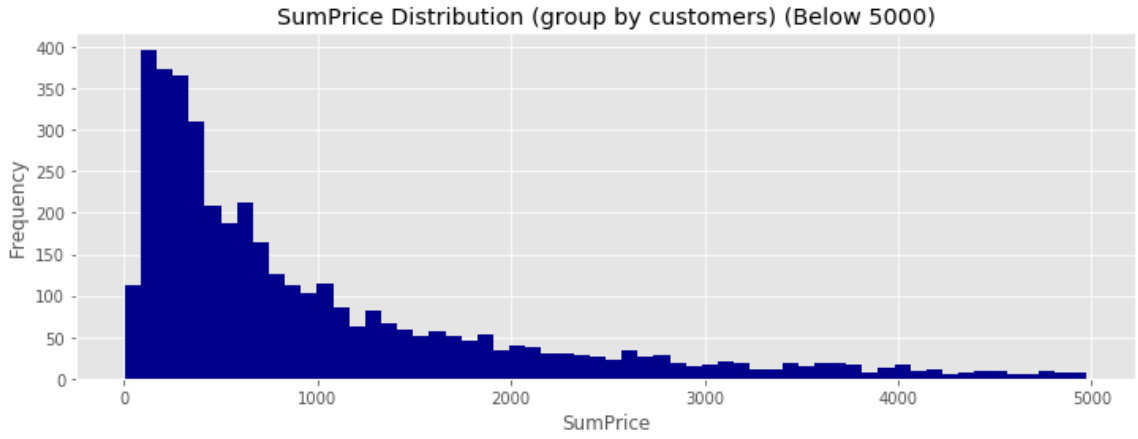
```
customer_grouped.SumPrice.hist(bins = 50, figsize = (12, 4), color = 'darkblue')
plt.title('SumPrice Distribution (group by customers)')
plt.ylabel('Frequency')
plt.xlabel('SumPrice')
plt.show()
```



From the histogram, we find most of the customers didn't have strong purchasing power. If we focus on the SumPrice Distribution of Customers with order SumPrice below 5000, we got a single peak long-tail distribution graph. The peak value located in the period (£83,£333)

In [41]:

```
customer_grouped[customer_grouped.SumPrice < 5000].SumPrice.hist(bins = 60, figsize = (
12, 4), color = 'darkblue')
plt.title('SumPrice Distribution (group by customers) (Below 5000)')
plt.ylabel('Frequency')
plt.xlabel('SumPrice')
plt.show()
```



From the perspective of products

In [42]:

```
#Unit price for some products fluctuates.
#example:
sales_success[sales_success['StockCode'] == '85123A'][['UnitPrice','Date']]
#sales_success.loc[sales_success['StockCode'] == '85123A',['UnitPrice','Date']]
sales_success.loc[sales_success['StockCode'] == '85123A',:].UnitPrice.value_counts()
```

Out[42]:

```
2.95    1702
2.55     358
5.79     157
5.91      25
3.20       5
3.24       4
2.40       1
Name: UnitPrice, dtype: int64
```

The price of one specific product can be various in different months, thus we calculated the mean cost of each product.
(Group the products with the stock code, calculate the total sales and total sold amount, and use total sales/total number of units).

In [44]:

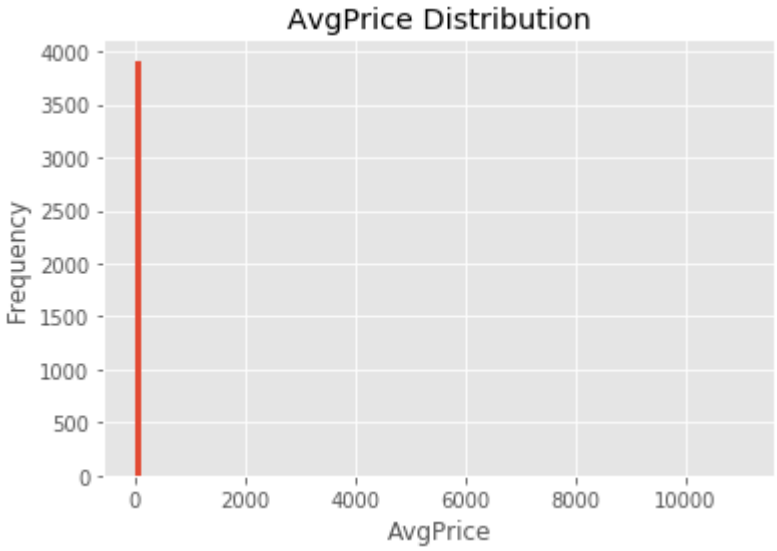
```
goods_grouped = sales_success.groupby('StockCode')[['Quantity', 'SumPrice']].sum()
goods_grouped['AvgPrice'] = goods_grouped['SumPrice'] / goods_grouped['Quantity']
goods_grouped.head()
```

Out[44]:

	Quantity	SumPrice	AvgPrice
StockCode			
10002	860	759.89	0.883593
10080	303	119.09	0.393036
10120	192	40.32	0.210000
10125	1295	993.99	0.767560
10133	2856	1539.60	0.539076

In [45]:

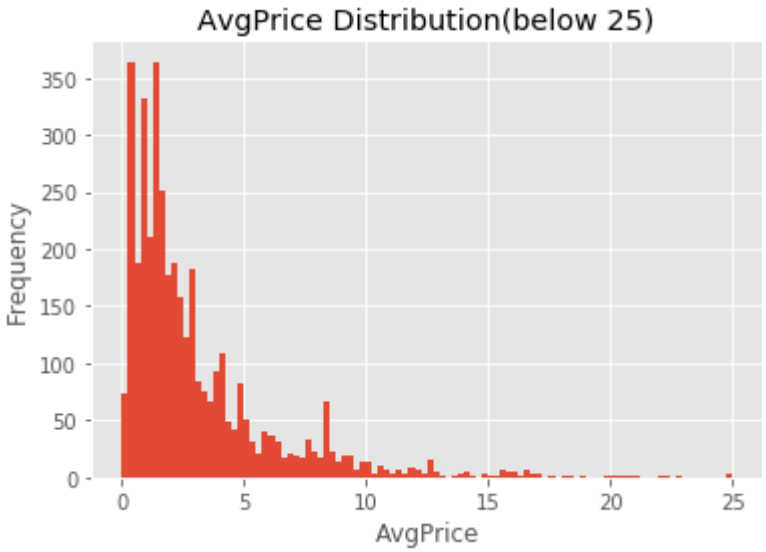
```
goods_grouped.AvgPrice.hist(bins=100)
plt.title('AvgPrice Distribution')
plt.ylabel('Frequency')
plt.xlabel('AvgPrice')
plt.show()
```



The price of most products was less than £100. Here we filtered the outlier.

In [97]:

```
goods_grouped[goods_grouped.AvgPrice<25]['AvgPrice'].hist(bins=100)
plt.title('AvgPrice Distribution(below 25)')
plt.ylabel('Frequency')
plt.xlabel('AvgPrice')
plt.show()
```

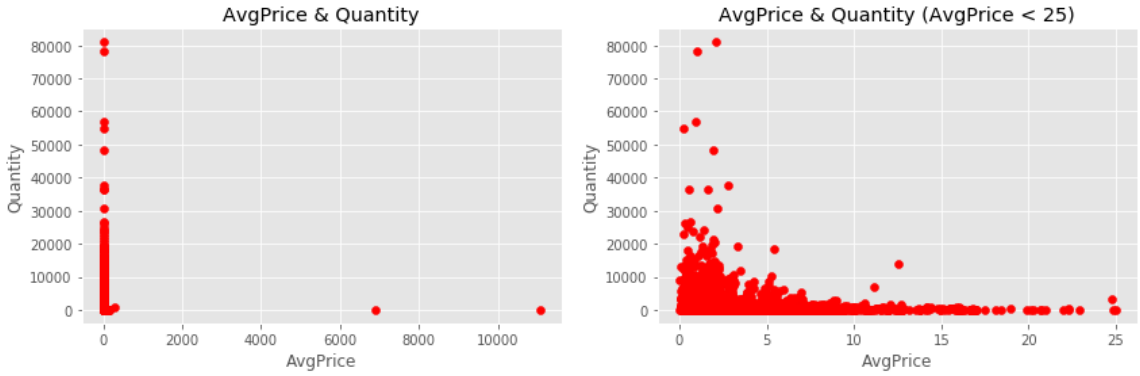


The peak value of products unit price was £1-2. A few products had their unit price higher than £10. Thus, the value proposition of this e-commerce website was focusing on the wholesale of small commodities with low unit prices.

In [49]:

```
plt.figure(figsize=(14,4))
plt.subplot(121)
plt.scatter(goods_grouped['AvgPrice'], goods_grouped['Quantity'], color = 'r')
plt.title('AvgPrice & Quantity')
plt.ylabel('Quantity')
plt.xlabel('AvgPrice')

plt.subplot(122)
plt.scatter(goods_grouped[goods_grouped.AvgPrice < 25]['AvgPrice'],goods_grouped[goods_
grouped.AvgPrice < 25]['Quantity'], color = 'r')
plt.title('AvgPrice & Quantity (AvgPrice < 25)')
plt.ylabel('Quantity')
plt.xlabel('AvgPrice')
plt.show()
```

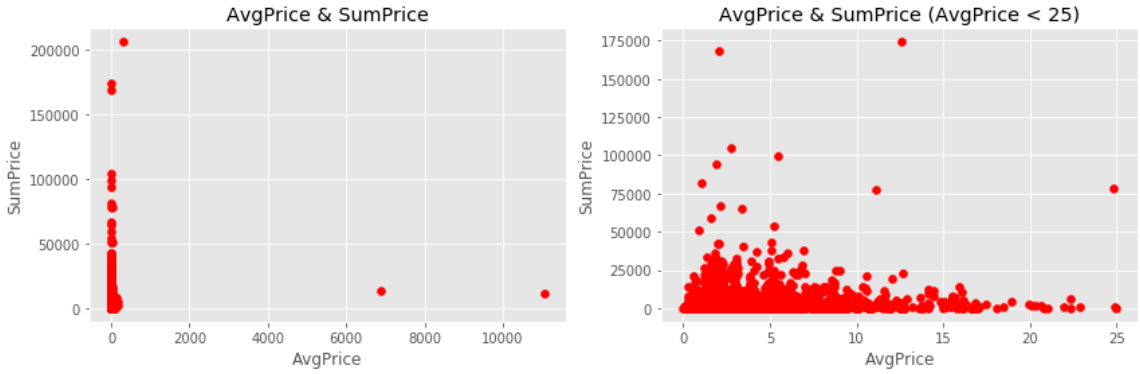


From the sales quantity, products with price lower than £5 got the customer's preference.

In [51]:

```
plt.figure(figsize=(14,4))
plt.subplot(121)
plt.scatter(goods_grouped['AvgPrice'], goods_grouped['SumPrice'], color = 'r')
plt.title('AvgPrice & SumPrice')
plt.ylabel('SumPrice')
plt.xlabel('AvgPrice')

plt.subplot(122)
plt.scatter(goods_grouped[goods_grouped.AvgPrice < 25]['AvgPrice'], goods_grouped[goods
_grouped.AvgPrice < 25]['SumPrice'], color = 'r')
plt.title('AvgPrice & SumPrice (AvgPrice < 25)')
plt.ylabel('SumPrice')
plt.xlabel('AvgPrice')
plt.show()
```



Products with a low unit price also contributed to high SumPrice. This kind of commodities became the leading source of total sales. Although some goods had a high unit price, the high unit price didn't result in high sales. Here we suggest the purchasing department select more low price products to enlarge product types in low price area.

From the perspective of time

In [54]:

```
time_grouped = sales_success.groupby('InvoiceNo').agg({'Date': np.min, 'Month': np.min,
'Quantity': np.sum, 'SumPrice': np.sum}).reset_index()
```

In [55]:

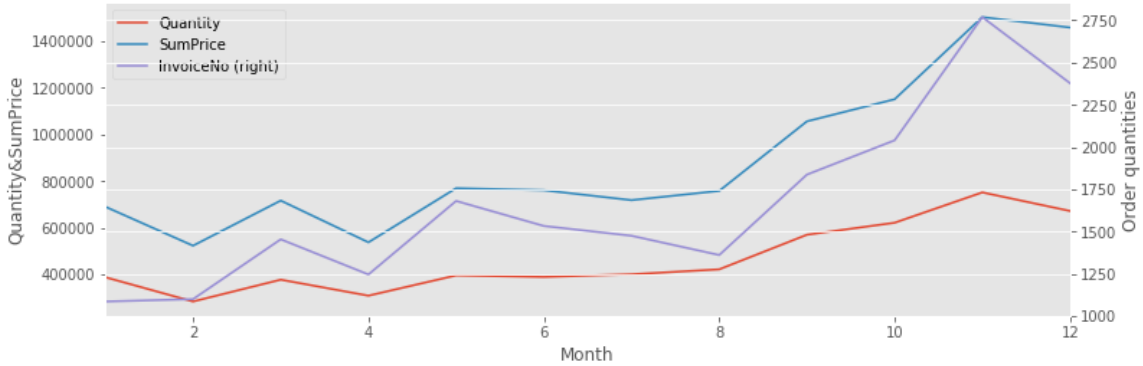
```
time_grouped.head()
```

Out[55]:

	InvoiceNo	Date	Month	Quantity	SumPrice
0	536365	2010-12-01	12	28	103.48
1	536366	2010-12-01	12	12	22.20
2	536367	2010-12-01	12	83	278.73
3	536368	2010-12-01	12	15	70.05
4	536369	2010-12-01	12	3	17.85

In [56]:

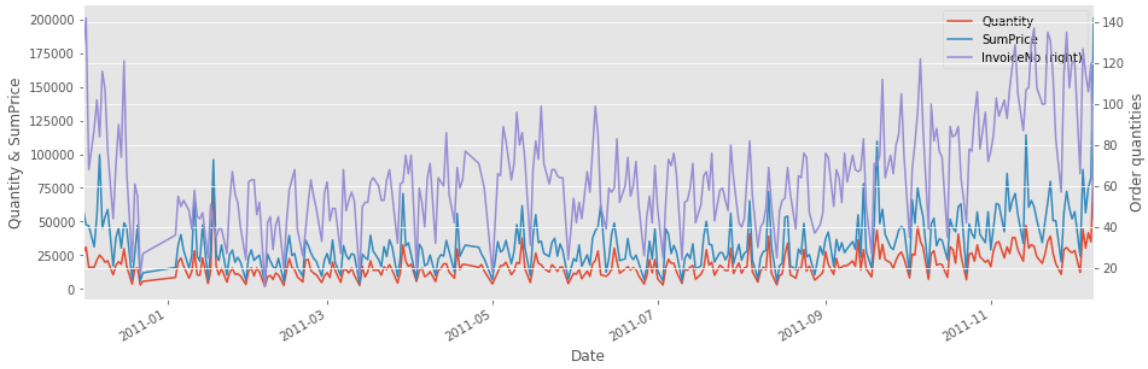
```
month=time_grouped.groupby('Month').agg({'Quantity':np.sum,'SumPrice':np.sum,'InvoiceN
o':np.size})
month_plot=month.plot(secondary_y='InvoiceNo',x_compat=True,figsize=(12,4))
month_plot.set_ylabel('Quantity&SumPrice')
month_plot.right_ax.set_ylabel('Order quantities')
plt.show()
```



We drew the plot with double axes on a monthly basis to show the wholesales, sales volume, and order amount per month. One point to mention here is that we only had data for 9 days in December and that's why we saw a 'decrease' in December which against the general trend. All three plots had similar trends. The sales conditions remained stable relatively from January to August. And then kept growing between September and November. Consider the dominating business of this e-commerce platform was selling gifts, the sales would be affected obviously by holidays, such as Halloween, Thanks Giving Day, Black Friday and Boxing day.

In [58]:

```
day=time_grouped.groupby('Date').agg({'Quantity':np.sum,'SumPrice':np.sum,'InvoiceNo':n
p.size}).plot(secondary_y='InvoiceNo',x_compat=True,figsize=(15,5))
day.set_ylabel('Quantity & SumPrice')
day.right_ax.set_ylabel('Order quantities')
plt.show()
```

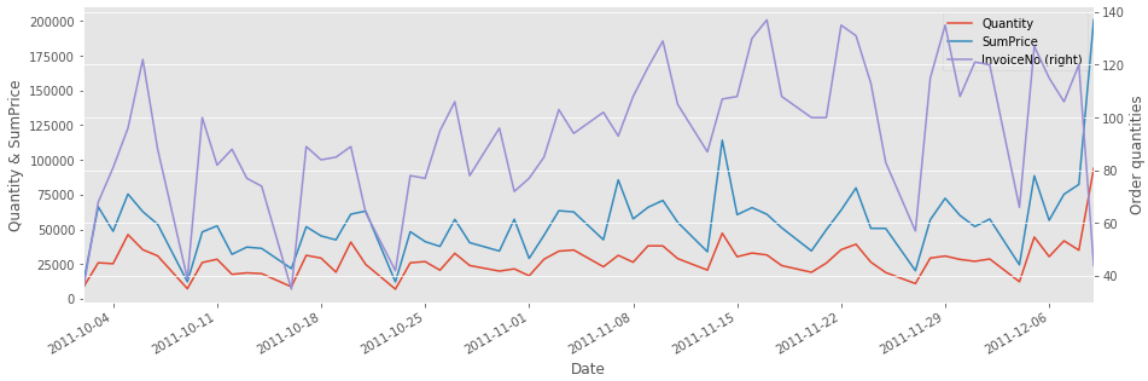


We draw the plot with double axes daily to show the sales, sales volume, and order amount per day. As we analyzed before, Quantity and SumPrice show similar trends. Here we saw a surge in sales in the last month. We drew a graph of this period separately.

In [60]:

```
#for the convience to get time period,set time as index
time_grouped=time_grouped.set_index('Date')
day_part=time_grouped['2011-10-01':'2011-12-09'].groupby('Date').agg({'Quantity': np.su
m, 'SumPrice': np.sum, 'InvoiceNo': np.size}).plot(secondary_y = 'InvoiceNo', x_compat=
True,figsize = (15, 5))
day_part.set_ylabel('Quantity & SumPrice')
day_part.right_ax.set_ylabel('Order quantities')
plt.show()

sales_success[sales_success.Date=='2011-12-09'].sort_values(by='SumPrice',ascending=Fal
se).head()
```



Out[60]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceTime	UnitPrice	Cust
535160	581483	23843	PAPER CRAFT , LITTLE BIRDIE	80995	2011-12-09 09:15:00	2.08	1644
536279	581498	DOT	DOTCOM POSTAGE	1	2011-12-09 10:26:00	1714.17	0
535647	581492	DOT	DOTCOM POSTAGE	1	2011-12-09 10:03:00	933.17	0
535164	581485	20749	ASSORTED COLOUR MINI CASES	84	2011-12-09 09:38:00	6.35	1738
536434	581566	23404	HOME SWEET HOME BLACKBOARD	144	2011-12-09 11:50:00	3.26	1810

On 9th, Dec, there was an apparent decrease in order quantity, however, we got the highest sales in this sampling interval, that indicates there should be some orders with large goods quantity on that day. After browsing the sales data, a customer from the UK bought 80000+ papercraft on 9th, Dec, caused the sales surge. It would be better if the e-commerce platform to pay more attention to this high-value customer by taking actions like setting fixed customer service staff to this client.

From the perspective of zones

In [62]:

```
#To find the congruent relationship between Customer and Country
sales_country=sales_customer[['CustomerID','Country']].drop_duplicates(subset=['Custome
rID','Country'])
#calcuate sum price on the basis of customer ID
country_grouped = sales_customer.groupby('CustomerID')[['SumPrice']].sum().reset_index
()
#use square brackets to make it a dataframe
#merge 2 tables
country_grouped = country_grouped.merge(sales_country, on='CustomerID')
country_grouped=country_grouped.groupby('Country').agg({'SumPrice':np.sum,'CustomerID':
np.size})
#build a new field to calculate average amount
country_grouped['AvgAmount']=country_grouped['SumPrice']/country_grouped['CustomerID']
country_grouped.sort_values(by='SumPrice',ascending=False)
#country_grouped.plot.bar()
#plt.show()
```

Out[62]:

	SumPrice	CustomerID	AvgAmount
Country			
United Kingdom	7284989.004	3920	1858.415562
Netherlands	285446.340	9	31716.260000
EIRE	265262.460	3	88420.820000
Germany	228678.400	94	2432.748936
France	208934.310	87	2401.543793
Australia	139843.950	9	15538.216667
Spain	66470.260	30	2215.675333
Switzerland	57222.850	21	2724.897619
Belgium	47971.210	25	1918.848400
Sweden	38367.830	8	4795.978750
Japan	37416.370	8	4677.046250
Norway	36165.440	10	3616.544000
Portugal	33375.840	19	1756.623158
Finland	22546.080	12	1878.840000
Singapore	21279.290	1	21279.290000
Channel Islands	20440.540	9	2271.171111
Denmark	19774.400	9	2197.155556
Italy	17483.240	14	1248.802857
Austria	16775.840	11	1525.076364
Cyprus	16518.540	8	2064.817500
Poland	7334.650	6	1222.441667
Israel	7215.840	3	2405.280000
Greece	4760.520	4	1190.130000
Iceland	4310.000	1	4310.000000
Canada	3666.380	4	916.595000
USA	3580.390	4	895.097500
Malta	2725.590	2	1362.795000
Unspecified	2660.770	4	665.192500
United Arab Emirates	1902.280	2	951.140000
Lebanon	1693.880	1	1693.880000
Lithuania	1661.060	1	1661.060000
European Community	1300.250	1	1300.250000

	SumPrice	CustomerID	AvgAmount
Country			
Brazil	1143.600	1	1143.600000
RSA	1002.310	1	1002.310000
Czech Republic	826.740	1	826.740000
Bahrain	548.400	2	274.200000
Saudi Arabia	145.920	1	145.920000

It seems that most of the customers were still from the the UK and the primary source of overseas income was mostly from the neighboring countries of the United Kingdom. This phenomenon might relate to logistic costs and language factors, or it might because of the influence of this e-commerce platform was attenuated gradually by distance. We may try to increase the overseas popularity by launching more advertisements. At the same time, add more language choices for the website. And also, provide more transparent and convenient solutions for decreasing the overseas logistics costs and optimizing logistic procedures.

Analysis of customer behaviors

Customer life circle

In [64]:

```
# the earlist purchase time
mindate = sales_customer.groupby('CustomerID')[['Date']].min()
# the last purchase time间
maxdate = sales_customer.groupby('CustomerID')[['Date']].max()
```

In [65]:

```
mindate.Date.value_counts().head(10)
```

Out[65]:

```
2010-12-01    95
2010-12-02    93
2010-12-08    83
2010-12-06    70
2010-12-05    69
2010-12-09    67
2010-12-16    58
2010-12-07    50
2010-12-03    46
2010-12-15    42
Name: Date, dtype: int64
```

In [66]:

```
maxdate.Date.value_counts().head(10)
```

Out[66]:

```
2011-12-08    103
2011-12-06     94
2011-12-05     94
2011-12-07     90
2011-12-01     79
2011-11-29     77
2011-11-22     74
2011-12-02     72
2011-11-30     71
2011-11-17     64
Name: Date, dtype: int64
```

Since we only have data for the one year period we don't know their buying actions before/after this period. Real-life cycles for some of them would be longer, and this brought limitations to our analysis. The date of initial consumptions occurred frequently at the beginning of the statistical period, and the date of last consumptions occurred mostly at the end of the statistical period. That means the actual life cycle of a large number of users must be longer.

In [68]:

```
#interval time
(maxdate-mindate).head(10)
```

Out[68]:

	Date
CustomerID	
12346	0 days
12347	365 days
12348	283 days
12349	0 days
12350	0 days
12352	260 days
12353	0 days
12354	0 days
12355	0 days
12356	303 days

In [69]:

```
life_time=maxdate-mindate
life_time.describe()
```

Out[69]:

	Date
count	4338
mean	130 days 18:31:02.240663
std	132 days 05:03:08.004046
min	0 days 00:00:00
25%	0 days 00:00:00
50%	93 days 00:00:00
75%	252 days 00:00:00
max	373 days 00:00:00

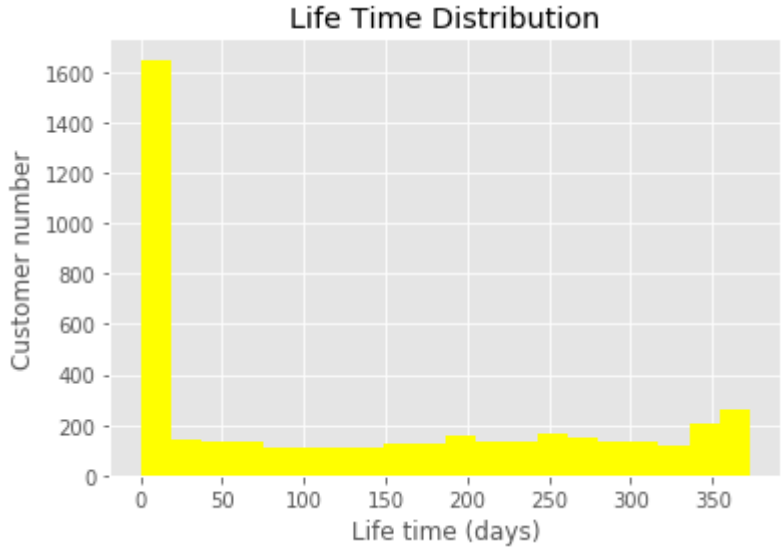
We had 4338 clients with CustomerID, the average life cycle was 130 days, the median was 93 days, which means some loyal customers make the mean value larger. The minimum and Q1 values were both 0, indicate more than 25% of clients only consumed once. The distribution of the life cycle was polarization.

In [71]:

```
#covert timedelta type to int
life_time['life_time']=life_time['Date'].dt.days
#np.dtype(life_time['Date'].dt.days)#transfer to int 64
```

In [72]:

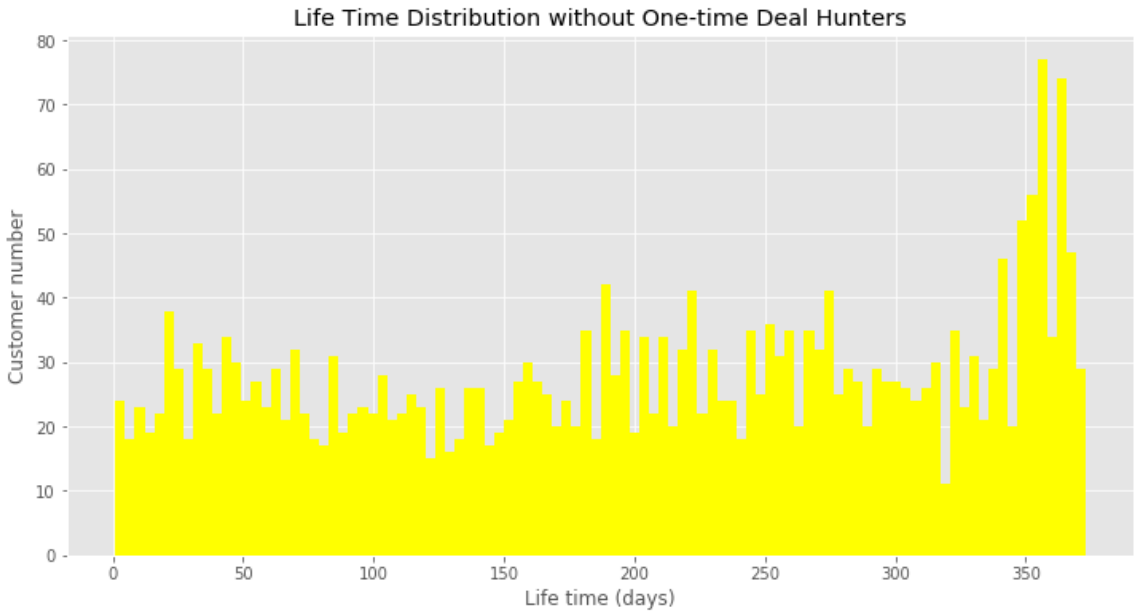
```
life_time['life_time'].hist(bins = 20, color = 'yellow')
plt.title('Life Time Distribution')
plt.ylabel('Customer number')
plt.xlabel('Life time (days)')
plt.show()
```



Many customers only consumed once and then left, this website needs to focus more on improving customer's initial purchase experience.
We may build an online mark system and do some phone investigations to get feedback. Also, activities aim at attracting second purchase should be considered, for instance: offering coupons with time limitation.

In [74]:

```
life_time[life_time['life_time'] > 0].life_time.hist(bins = 100, figsize = (12, 6), col
or = 'yellow')
plt.title('Life Time Distribution without One-time Deal Hunters')
plt.ylabel('Customer number')
plt.xlabel('Life time (days)')
plt.show()
```



We filtered the customer with life cycle 0 and plot the remaining data again, about 25% of clients had a life cycle length between 170~330 days, this group had a high-quality life cycle. And customers with life cycle longer than 330-day can be viewed as having a good user sickness, we had a lot of cases located in this period, which was a good sign.

In [76]:

```
bin=[0,75,170,330,400]
pd.cut(life_time[life_time['life_time']>0]['life_time'],bins=bin).value_counts()
```

Out[76]:

```
(170, 330]    1198
(75, 170]      577
(0, 75]        517
(330, 400]     498
Name: life_time, dtype: int64
```

In [77]:

```
#mean purchase period for customer brought more than once
life_time[life_time['life_time'] > 0].life_time.mean()
```

Out[77]:

```
203.32867383512544
```

The average lifecycle for customer with more than twice the purchase was 203, far higher than the total mean life cycle length 103.
This company should put more effort into maintaining new customers.

'Customer life cycle' is the time difference between the first and last purchase. However, the 'customer retention period' focuses on customers' periodic behaviors. Due to the dataset incompleteness we mentioned before, the outcome had limitations here as well.

Customer retention condition

In [80]:

```
sales_customer.head()
mindate_1=mindate.copy().reset_index()
customer_retention=sales_customer.merge(mindate_1,on='CustomerID',how='inner',suffixes=
('','Min'))
customer_retention.head()
```

Out[80]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceTime	UnitPrice	CustomerID
0	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850
1	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850
2	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850
3	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	2010-12-01 08:26:00	7.65	17850
4	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	2010-12-01 08:26:00	4.25	17850

In [81]:

```
customer_retention['DateDiff']=(customer_retention.Date-customer_retention.DateMin).dt.
days
date_bins=[0,3,7,30,60,90,100]
customer_retention['DateDiffBin']=pd.cut(customer_retention['DateDiff'],bins=date_bins)
customer_retention[customer_retention['DateDiffBin'].isnull().values==False].head()
```

Out[81]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceTime	UnitPrice	CustomerID
82	536600	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-02 08:32:00	2.55	17850
83	536600	71053	WHITE METAL LANTERN	6	2010-12-02 08:32:00	3.39	17850
84	536600	82483	WOOD 2 DRAWER CABINET WHITE FINISH	4	2010-12-02 08:32:00	4.95	17850
85	536600	82486	WOOD S/3 CABINET ANT WHITE FINISH	2	2010-12-02 08:32:00	6.95	17850
86	536600	82482	WOODEN PICTURE FRAME WHITE FINISH	6	2010-12-02 08:32:00	2.10	17850

In [82]:

```
customer_retention['DateDiffBin'].value_counts()
```

Out[82]:

```
(30, 60]     28252
(60, 90]     25876
(7, 30]      17148
(90, 100]     9275
(3, 7]        3559
(0, 3]        1311
Name: DateDiffBin, dtype: int64
```

For customers with only one consumption, the rentention period was 0, and we didn't count them in this part.

In [84]:

```
customer_retention.DateMin.describe()
```

Out[84]:

```
count                  392690
unique                    305
top       2010-12-01 00:00:00
freq                    35423
first     2010-12-01 00:00:00
last      2011-12-09 00:00:00
Name: DateMin, dtype: object
```

In [85]:

```
#use pivot table to see each customer consumed how much money in which period
retention_pivot = customer_retention.pivot_table(index = ['CustomerID'], columns = ['Da
teDiffBin'], values = ['SumPrice'], aggfunc= np.sum)
retention_pivot.head(100)
```

Out[85]:

	SumPrice					
DateDiffBin	**(0, 3]**	**(3, 7]**	**(7, 30]**	**(30, 60]**	**(60, 90]**	**(90, 100]**
CustomerID						
12346	NaN	NaN	NaN	NaN	NaN	NaN
12347	NaN	NaN	NaN	475.39	NaN	NaN
12348	NaN	NaN	NaN	227.44	NaN	NaN
12349	NaN	NaN	NaN	NaN	NaN	NaN
12350	NaN	NaN	NaN	NaN	NaN	NaN
12352	NaN	NaN	1104.98	160.33	NaN	NaN
12353	NaN	NaN	NaN	NaN	NaN	NaN
12354	NaN	NaN	NaN	NaN	NaN	NaN
12355	NaN	NaN	NaN	NaN	NaN	NaN
12356	NaN	NaN	NaN	NaN	481.46	NaN
12357	NaN	NaN	NaN	NaN	NaN	NaN
12358	NaN	NaN	NaN	NaN	NaN	NaN
12359	NaN	NaN	1838.91	NaN	NaN	NaN
12360	NaN	NaN	NaN	NaN	534.70	NaN
12361	NaN	NaN	NaN	NaN	NaN	NaN
12362	NaN	NaN	NaN	NaN	495.24	NaN
12363	NaN	NaN	NaN	NaN	NaN	NaN
12364	NaN	NaN	NaN	79.80	299.06	NaN
12365	NaN	NaN	NaN	NaN	NaN	NaN
12367	NaN	NaN	NaN	NaN	NaN	NaN
12370	277.2	NaN	NaN	NaN	938.39	NaN
12371	NaN	NaN	360.00	NaN	NaN	NaN
12372	NaN	NaN	NaN	NaN	515.70	NaN
12373	NaN	NaN	NaN	NaN	NaN	NaN
12374	NaN	NaN	NaN	NaN	NaN	NaN
12375	NaN	NaN	NaN	NaN	227.20	NaN
12377	NaN	NaN	NaN	626.60	NaN	NaN
12378	NaN	NaN	NaN	NaN	NaN	NaN
12379	NaN	NaN	NaN	NaN	392.40	NaN
12380	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...

	SumPrice					
DateDiffBin	**(0, 3]**	**(3, 7]**	**(7, 30]**	**(30, 60]**	**(60, 90]**	**(90, 100]**
CustomerID						
12432	700.0	NaN	NaN	NaN	NaN	NaN
12433	NaN	1867.98	NaN	NaN	NaN	NaN
12434	NaN	NaN	NaN	NaN	NaN	NaN
12435	NaN	NaN	NaN	NaN	NaN	NaN
12436	NaN	NaN	NaN	NaN	NaN	NaN
12437	NaN	NaN	150.50	439.76	147.71	NaN
12438	NaN	NaN	NaN	NaN	NaN	NaN
12441	NaN	NaN	NaN	NaN	NaN	NaN
12442	NaN	NaN	NaN	NaN	NaN	NaN
12444	NaN	NaN	NaN	757.07	562.54	NaN
12445	NaN	NaN	NaN	NaN	NaN	NaN
12446	NaN	NaN	NaN	NaN	NaN	NaN
12447	NaN	NaN	NaN	NaN	NaN	NaN
12448	NaN	NaN	NaN	NaN	NaN	NaN
12449	NaN	NaN	NaN	777.95	NaN	1087.7
12450	NaN	NaN	96.00	NaN	NaN	NaN
12451	NaN	NaN	NaN	NaN	NaN	NaN
12452	NaN	NaN	NaN	NaN	NaN	NaN
12453	NaN	NaN	NaN	NaN	NaN	NaN
12454	NaN	NaN	NaN	NaN	NaN	NaN
12455	15.0	NaN	NaN	NaN	NaN	NaN
12456	NaN	NaN	NaN	NaN	NaN	NaN
12457	219.0	179.00	NaN	NaN	NaN	NaN
12458	NaN	NaN	NaN	NaN	NaN	NaN
12461	NaN	NaN	NaN	200.00	NaN	NaN
12462	NaN	NaN	NaN	NaN	NaN	NaN
12463	NaN	NaN	NaN	297.38	337.84	NaN
12464	NaN	NaN	149.95	NaN	NaN	NaN
12465	NaN	NaN	NaN	NaN	NaN	NaN
12468	NaN	NaN	NaN	NaN	NaN	NaN

100 rows × 6 columns

In [86]:

```
retention_pivot.mean()
```

Out[86]:

```
          DateDiffBin
SumPrice  (0, 3]         587.662911
          (3, 7]         366.199325
          (7, 30]        519.275532
          (30, 60]       595.992305
          (60, 90]       593.322921
          (90, 100]      632.466067
dtype: float64
```

In [87]:

```
#0-1 convert
retention_pivot_trans=retention_pivot.fillna(0).applymap(lambda x:1 if x > 0 else 0)
retention_pivot_trans.head()
```

Out[87]:

	SumPrice					
DateDiffBin	**(0, 3]**	**(3, 7]**	**(7, 30]**	**(30, 60]**	**(60, 90]**	**(90, 100]**
CustomerID						
12346	0	0	0	0	0	0
12347	0	0	0	1	0	0
12348	0	0	0	1	0	0
12349	0	0	0	0	0	0
12350	0	0	0	0	0	0

In [88]:

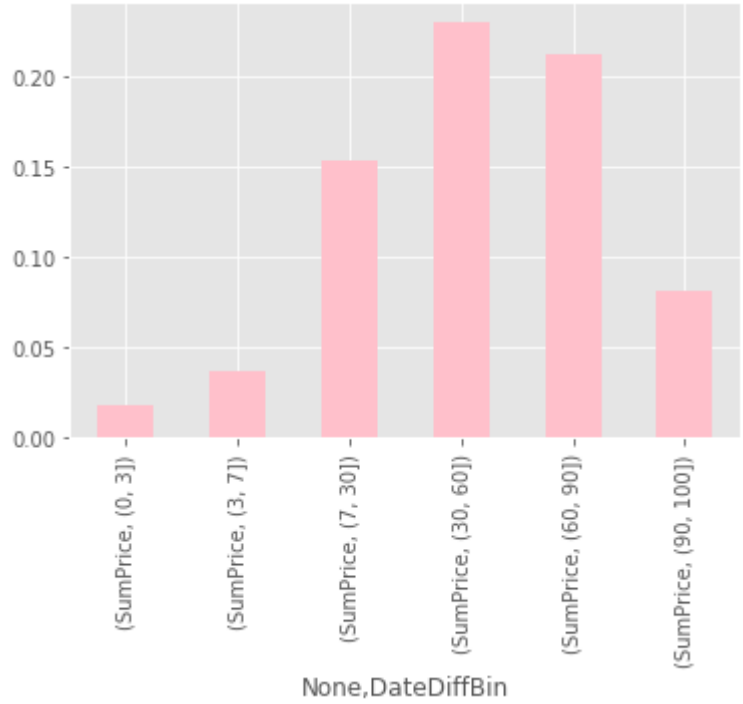
```
retention_pivot_trans.sum()/ retention_pivot_trans.count()
```

Out[88]:

```
          DateDiffBin
SumPrice  (0, 3]         0.018211
          (3, 7]         0.037575
          (7, 30]        0.153757
          (30, 60]       0.230060
          (60, 90]       0.212310
          (90, 100]      0.082065
dtype: float64
```

In [89]:

```
(retention_pivot_trans.sum()/ retention_pivot_trans.count()).plot.bar(color = 'pink')
plt.show()
```



Customers came back in:
0~3 days:3.2%
4~7 days:6.6%
the second month:37.4%
the third month:40.5%
90~180 days:67%
The high loyalty customers might not consumption frequently but their user stickiness was pretty good.

Purchase period

In [91]:

```
#We calculate the difference between two adjacent purchase.
sales_cycle=customer_retention.drop_duplicates(subset=['CustomerID','Date'],keep='firs
t')
sales_cycle.sort_values(by='Date',ascending=True)
def diff(group):
    d=group.DateDiff-group.DateDiff.shift()
    return d
last_diff=sales_cycle.groupby('CustomerID').apply(diff)
last_diff.head(10)
```

Out[91]:

```
CustomerID
12346       193892     NaN
12347       93544      NaN
            93575     50.0
            93604     71.0
            93628     63.0
            93646     54.0
            93668     90.0
            93715     37.0
12348       156818     NaN
            156835    40.0
Name: DateDiff, dtype: float64
```

In [92]:

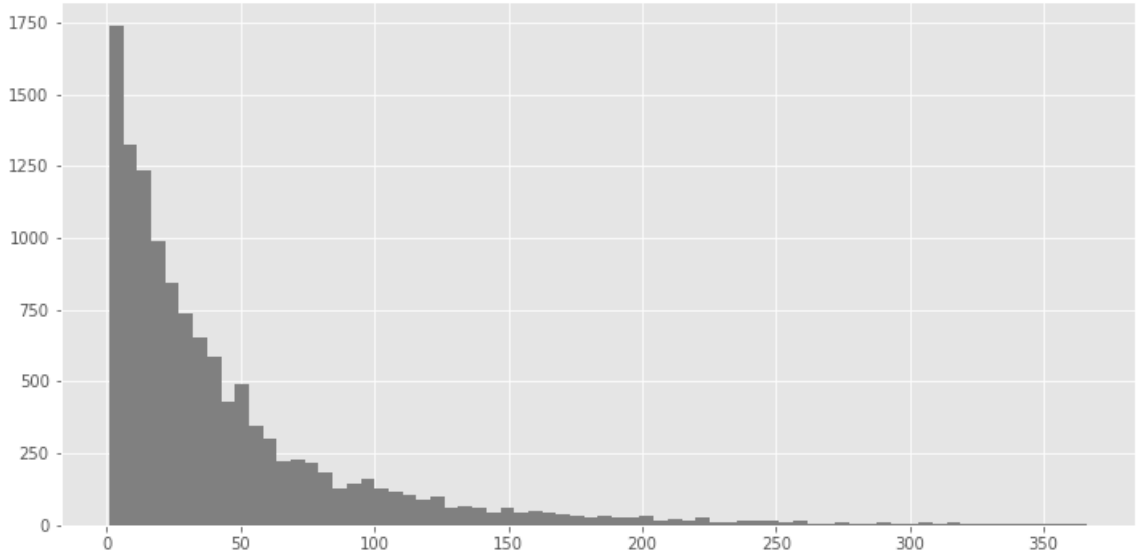
```
last_diff.describe()
```

Out[92]:

```
count    12425.000000
mean        45.656901
std         53.067262
min          1.000000
25%         12.000000
50%         28.000000
75%         58.000000
max        366.000000
Name: DateDiff, dtype: float64
```

In [93]:

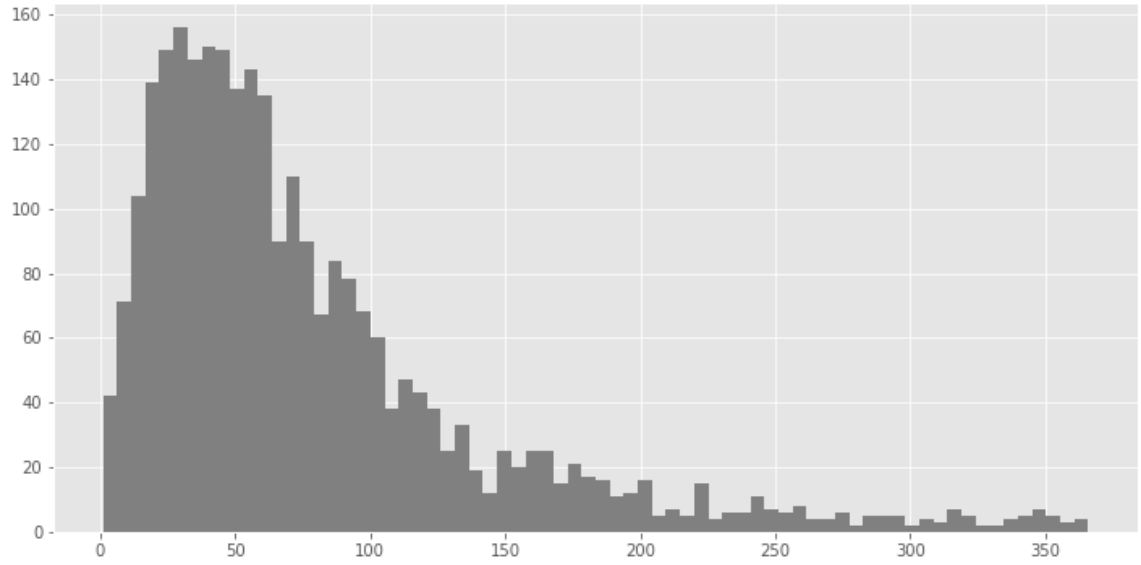
```
last_diff.hist(bins = 70,figsize = (12, 6), color = 'grey')
plt.show()
```



The purchase period for all orders showed a long-tail distribution.
most intervals were not long

In [95]:

```
last_diff_customer = last_diff.groupby('CustomerID').mean()
last_diff_customer.mean()
last_diff_customer.hist(bins = 70,figsize = (12, 6), color = 'grey')
plt.show()
```



Group the data by 'CustomerID' again. The plot was a right-skewed distribution with the peak value around 15~70 days, indicating most customers concentrated there. We suggest this company send promotion information to customers monthly.