In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
df = pd.read_excel('RetailPulseAssignmentData.xlsx')
```

In [3]:

```
df
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom

You are given a dataset that contains all the transactions occurring for a store.

On this dataset do an extensive EDA and explain your findings using relevant visualizations.

The above questions are just samples, we would like to know how this data can be analyzed to help the retail business make better decisions.

We will require both the notebook and the PDF version of it. Keep the PDF report short and to the point.

The presentation and simplicity of understanding the report can give you brownie points over other candidates.

Some of the questions you can answer in the EDA:

1. Can customers be segmented into different categories? If yes then perform analysis on the same and also propose categories. If no, then explain why?
2. How would you define a loyal customer?
3. What is the most popular time of year based on this sales data?
4. Is there any seasonality in data? Explain with supportive evidence.
5. Discuss customer's lifetime with respect to the given dataset.

First understanding the features of the dataset

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 525461 entries, 0 to 525460
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   Invoice      525461 non-null  object        
 1   StockCode    525461 non-null  object        
 2   Description  522533 non-null  object        
 3   Quantity     525461 non-null  int64         
 4   InvoiceDate  525461 non-null  datetime64[ns]
 5   Price        525461 non-null  float64       
 6   Customer ID  417534 non-null  float64       
 7   Country      525461 non-null  object        
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 32.1+ MB
```

In [5]:

```
df.describe()
```

Out[5]:

	Quantity	Price	Customer ID
count	525461.000000	525461.000000	417534.000000
mean	10.337667	4.688834	15360.645478
std	107.424110	146.126914	1680.811316
min	-9600.000000	-53594.360000	12346.000000
25%	1.000000	1.250000	13983.000000
50%	3.000000	2.100000	15311.000000
75%	10.000000	4.210000	16799.000000
max	19152.000000	25111.090000	18287.000000

Observation: There are some products which has negative Quantity values, considering them as the returned items

In [6]:

```
df
```

Out[6]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom
...	...	...	...	...	...	...	...	...
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.0	United Kingdom
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530.0	United Kingdom

525461 rows × 8 columns

Understanding the data

In [7]:

```
df.columns
```

Out[7]:

```
Index(['Invoice', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'Price', 'Customer ID', 'Country'],
      dtype='object')
```

In [8]:

```
df['Invoice'].nunique()
```

Out[8]:

```
28816
```

In [9]:

```
df['StockCode'].nunique()
```

Out[9]:

4632

In [10]:

```
df['Description'].nunique()
```

Out[10]:

4681

In [11]:

```
df['Customer ID'].nunique()
```

Out[11]:

4383

In [12]:

```
df['Country'].nunique()
```

Out[12]:

40

Observation: There's same invoice for different purchases as there are only 28816 unique invoices listed in the data

In [13]:

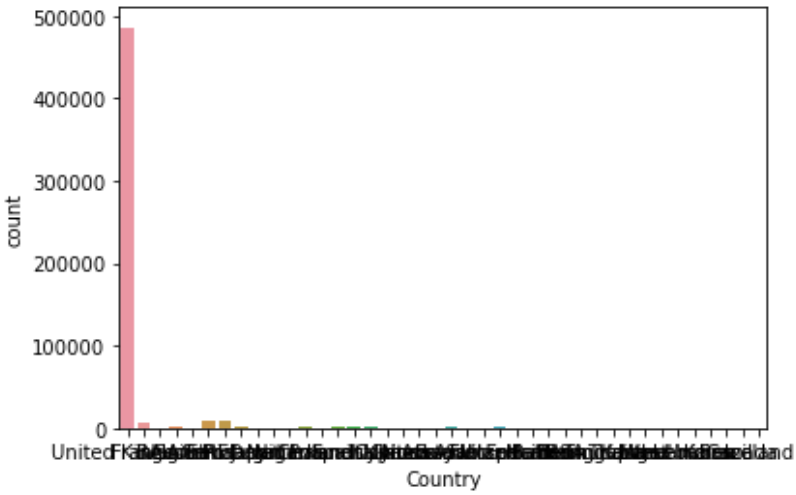
```
df.head(5)
```

Out[13]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom

In [14]:

```
sns.countplot(x='Country', data=df)
plt.show()
```



Mainly the records are available to united states only

In [15]:

```
grouped = df.groupby('Country')
count = grouped.size()
percentage = count / len(df) * 100
percentage
```

Out[15]:

```
Country
Australia               0.124462
Austria                 0.102196
Bahrain                 0.020363
Belgium                 0.200586
Bermuda                 0.006471
Brazil                  0.011799
Canada                  0.014654
Channel Islands         0.172420
Cyprus                  0.105431
Denmark                 0.081452
EIRE                    1.840289
Finland                 0.067369
France                  1.098464
Germany                 1.547023
Greece                  0.098390
Hong Kong               0.014463
Iceland                 0.013512
Israel                  0.014083
Italy                   0.139116
Japan                   0.042629
Korea                   0.011989
Lebanon                 0.002474
Lithuania               0.029308
Malta                   0.032733
Netherlands             0.526966
Nigeria                 0.006090
Norway                  0.070224
Poland                  0.036920
Portugal                0.209530
RSA                     0.021124
Singapore               0.022266
Spain                   0.243215
Sweden                  0.171659
Switzerland             0.225897
Thailand                0.014463
USA                     0.046435
United Arab Emirates    0.082214
United Kingdom         92.462048
Unspecified             0.058996
West Indies             0.010277
dtype: float64
```

92.46 percentage of the datavalues only for united states

In [16]:

```
df
```

Out[16]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom
...	...	...	...	...	...	...	...	...
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530.0	United Kingdom
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530.0	United Kingdom
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530.0	United Kingdom

525461 rows × 8 columns

In [17]:

```
print('unique Invoices count:', df['Invoice'].nunique())
print('unique Customers count:', df['Customer ID'].nunique())
```

```
unique Invoices count: 28816
unique Customers count: 4383
```

Obervation: There are repeating customers

Checking how frequently the customers appears

In [18]:

```
df['Customer ID']
```

Out[18]:

```
0         13085.0
1         13085.0
2         13085.0
3         13085.0
4         13085.0
           ...
525456    17530.0
525457    17530.0
525458    17530.0
525459    17530.0
525460    17530.0
Name: Customer ID, Length: 525461, dtype: float64
```

In [19]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 525461 entries, 0 to 525460
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   Invoice      525461 non-null  object        
 1   StockCode    525461 non-null  object        
 2   Description  522533 non-null  object        
 3   Quantity     525461 non-null  int64         
 4   InvoiceDate  525461 non-null  datetime64[ns]
 5   Price        525461 non-null  float64       
 6   Customer ID  417534 non-null  float64       
 7   Country      525461 non-null  object        
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 32.1+ MB
```

Removing all the rows with non available customer id

In [21]:

```
import math
df = df[df['Customer ID'].apply(lambda x: isinstance(x, (int, float)) and not math.isnan(x))]
```

In [22]:

```
df.count()
```

Out[22]:

```
Invoice        417534
StockCode      417534
Description    417534
Quantity       417534
InvoiceDate    417534
Price          417534
Customer ID    417534
Country        417534
dtype: int64
```

In [23]:

```
df.shape
```

Out[23]:

```
(417534, 8)
```

In [32]:

```
# type(df['Customer ID'].apply(lambda x: int(x)).iloc[1])
df['Customer ID'] = df['Customer ID'].apply(lambda x: int(x))
type(df['Customer ID'].iloc[1])

customers = df.groupby('Customer ID').count()
customers
```

```
C:\Users\Ashutosh Patidar\AppData\Local\Temp\ipykernel_27256\270864301.py:2: Set
tingWithCopyWarning:
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: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  df['Customer ID'] = df['Customer ID'].apply(lambda x: int(x))
```

Out[32]:

Customer ID	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Country
12346	46	46	46	46	46	46	46
12347	71	71	71	71	71	71	71
12348	20	20	20	20	20	20	20
12349	107	107	107	107	107	107	107
12351	21	21	21	21	21	21	21
...	...	...	...	...	...	...	...
18283	230	230	230	230	230	230	230
18284	29	29	29	29	29	29	29
18285	12	12	12	12	12	12	12
18286	70	70	70	70	70	70	70
18287	86	86	86	86	86	86	86

4383 rows × 7 columns

In [37]:

```
customer_freq = customers['Invoice']
```

In [38]:

```
customer_freq
```

Out[38]:

```
Customer ID
12346     46
12347     71
12348     20
12349    107
12351     21
         ...
18283    230
18284     29
18285     12
18286     70
18287     86
Name: Invoice, Length: 4383, dtype: int64
```

In [41]:

```
customer_freq.info()
```

```
<class 'pandas.core.series.Series'>
Int64Index: 4383 entries, 12346 to 18287
Series name: Invoice
Non-Null Count  Dtype
--------------  -----
4383 non-null   int64
dtypes: int64(1)
memory usage: 68.5 KB
```

In [42]:

```
customer_freq.describe()
```

Out[42]:

```
count    4383.000000
mean       95.262149
std       204.903882
min         1.000000
25%        18.000000
50%        44.000000
75%       103.000000
max      5710.000000
Name: Invoice, dtype: float64
```

In [45]:

```
type(customer_freq)
```

Out[45]:

```
pandas.core.series.Series
```

In [48]:

```
customer_freq = customer_freq.to_frame()
```

In [50]:

```
customer_freq.columns
```

Out[50]:

```
Index(['Invoice'], dtype='object')
```

In [53]:

```
customer_freq = customer_freq.reset_index()
```

In [54]:

```
customer_freq[]
```

Out[54]:

	Customer ID	Invoice
0	12346	46
1	12347	71
2	12348	20
3	12349	107
4	12351	21
...	...	...
4378	18283	230
4379	18284	29
4380	18285	12
4381	18286	70
4382	18287	86

4383 rows × 2 columns

In [55]:

```
customer_freq = customer_freq.rename(columns={'Invoice': 'frequency'})
```

In [56]:

```
customer_freq
```

Out[56]:

	Customer ID	frequency
0	12346	46
1	12347	71
2	12348	20
3	12349	107
4	12351	21
...	...	...
4378	18283	230
4379	18284	29
4380	18285	12
4381	18286	70
4382	18287	86

4383 rows × 2 columns

In [62]:

```
customer_freq = customer_freq.sort_values('frequency', ascending=False)
```

In [64]:

```
customer_freq
```

Out[64]:

	Customer ID	frequency
1869	14911	5710
4058	17841	5114
1631	14606	3927
1291	14156	2710
254	12748	2665
...	...	...
3965	17715	1
4349	18246	1
2832	16219	1
14	12362	1
2103	15233	1

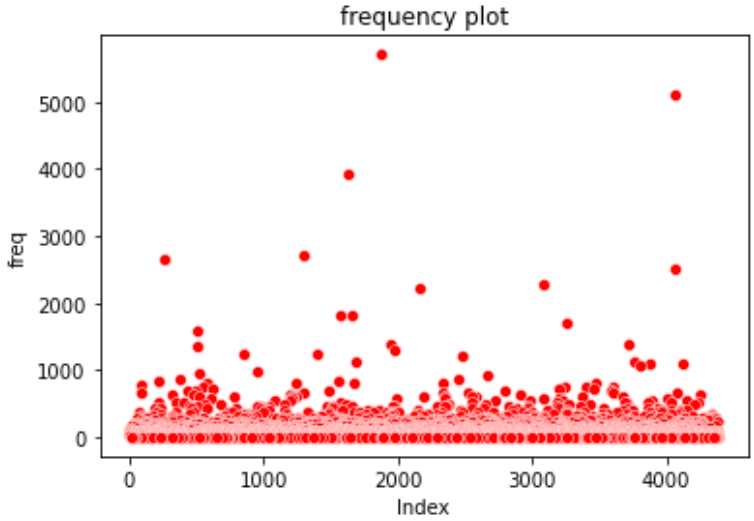
4383 rows × 2 columns

Observation: Some customers apprers so frequently

visualising the obervation

In [79]:

```
# plot the data as a scatter plot using seaborn
sns.scatterplot(x=customer_freq.index, y='frequency', data=customer_freq, color = "red")

# set the axis labels and title
plt.xlabel('Index')
plt.ylabel('freq')
plt.title('frequency plot')

# display the plot
plt.show()
```



Obeservation: coustomers must be segmented into different categories as some of the customers buy really frequenctly

Adding the record for amount purchsed by customers in the main df

In [80]:

```
df
```

Out[80]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085	United Kingdom
...	...	...	...	...	...	...	...	...
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530	United Kingdom
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530	United Kingdom
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530	United Kingdom
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530	United Kingdom
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530	United Kingdom

417534 rows × 8 columns

In [81]:

```
df['Amount'] = df['Price']*df['Quantity']
df['Amount']
```

```
C:\Users\Ashutosh Patidar\AppData\Local\Temp\ipykernel_27256\4118390999.py:1: Se
ttingWithCopyWarning:
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: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  df['Amount'] = df['Price']*df['Quantity']
```

Out[81]:

```
0          83.40
1          81.00
2          81.00
3         100.80
4          30.00
           ...
525456      5.90
525457      3.75
525458      3.75
525459      7.50
525460      3.90
Name: Amount, Length: 417534, dtype: float64
```

In [82]:

```
df
```

Out[82]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	Amou
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085	United Kingdom	83.
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	81.
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	81.
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085	United Kingdom	100.
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085	United Kingdom	30.
...	...	...	...	...	...	...	...	...	
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530	United Kingdom	5.
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530	United Kingdom	3.
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530	United Kingdom	3.
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530	United Kingdom	7.
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530	United Kingdom	3.

417534 rows × 9 columns

Finding the records for quantity and amount purchased by each customer

In [89]:

```
purchased = df.groupby('Customer ID').sum()
purchased
```

Out[89]:

Customer ID	Quantity	Price	Amount
12346	52	578.36	-64.68
12347	828	162.95	1323.32
12348	373	14.39	222.16
12349	988	899.49	2646.99
12351	261	49.46	300.93
...	...	...	...
18283	336	498.82	641.77
18284	493	116.09	436.68
18285	145	100.20	427.00
18286	592	306.55	1188.43
18287	1425	244.34	2340.61

4383 rows × 3 columns

In [90]:

```
purchased.nunique()
```

Out[90]:

```
Quantity    1773
Price       4119
Amount      4306
dtype: int64
```

In [95]:

```
purchased = purchased.reset_index()
purchased
```

Out[95]:

	Customer ID	Quantity	Price	Amount
0	12346	52	578.36	-64.68
1	12347	828	162.95	1323.32
2	12348	373	14.39	222.16
3	12349	988	899.49	2646.99
4	12351	261	49.46	300.93
...	...	...	...	...
4378	18283	336	498.82	641.77
4379	18284	493	116.09	436.68
4380	18285	145	100.20	427.00
4381	18286	592	306.55	1188.43
4382	18287	1425	244.34	2340.61

4383 rows × 4 columns

In [97]:

```
purchased['Customer ID'].nunique()
```

Out[97]:

```
4383
```

In [ ]:

```
purchased.drop('Price', axis=1, inplace=True)
```

In [105]:

```
print(purchased.columns)
```

```
Index(['Customer ID', 'Quantity', 'Amount'], dtype='object')
```

In [107]:

```
purchased
```

Out[107]:

	Customer ID	Quantity	Amount
0	12346	52	-64.68
1	12347	828	1323.32
2	12348	373	222.16
3	12349	988	2646.99
4	12351	261	300.93
...	...	...	...
4378	18283	336	641.77
4379	18284	493	436.68
4380	18285	145	427.00
4381	18286	592	1188.43
4382	18287	1425	2340.61

4383 rows × 3 columns

Now merging the both dataframes based on Customer ID

customer_table = customer_freq + purchased

In [110]:

```
customer_table = pd.merge(customer_freq, purchased, on='Customer ID')
customer_table
```

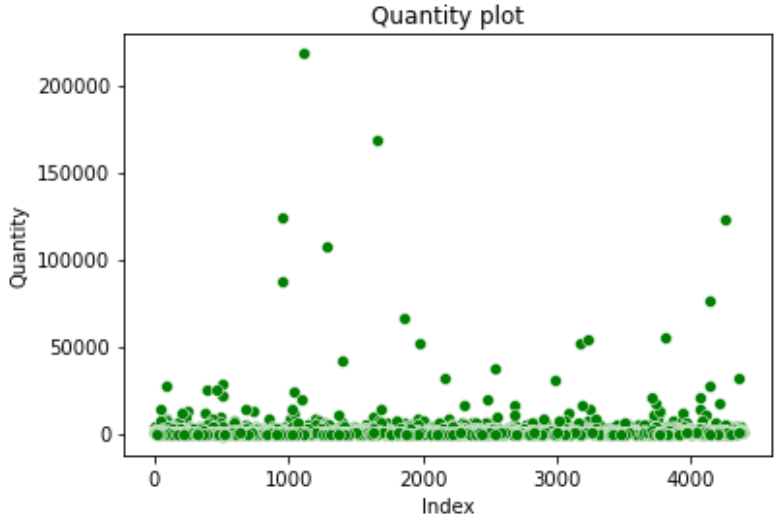
Out[110]:

	Customer ID	frequency	Quantity	Amount
0	14911	5710	66561	137675.91
1	17841	5114	14255	29175.41
2	14606	3927	9322	18380.53
3	14156	2710	106885	183180.55
4	12748	2665	12862	20898.03
...	...	...	...	...
4378	17715	1	192	163.20
4379	18246	1	48	162.72
4380	16219	1	48	40.80
4381	12362	1	1	130.00
4382	15233	1	12	59.40

4383 rows × 4 columns

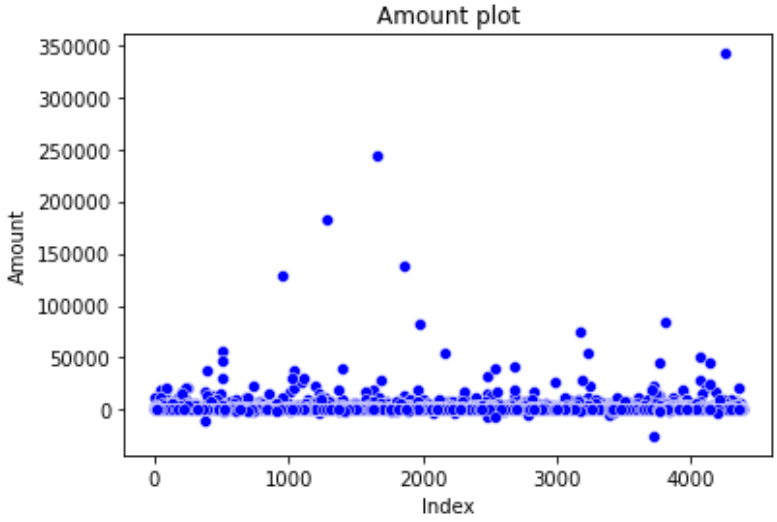
In [111]:

```
sns.scatterplot(x=customer_freq.index, y='Quantity',
                data=customer_table, color = "green")

plt.xlabel('Index')
plt.ylabel('Quantity')
plt.title('Quantity plot')

plt.show()
```



In [114]:

```
sns.scatterplot(x=customer_freq.index, y='Amount',
                data=customer_table, color = "blue")

plt.xlabel('Index')
plt.ylabel('Amount')
plt.title('Amount plot')

plt.show()
```



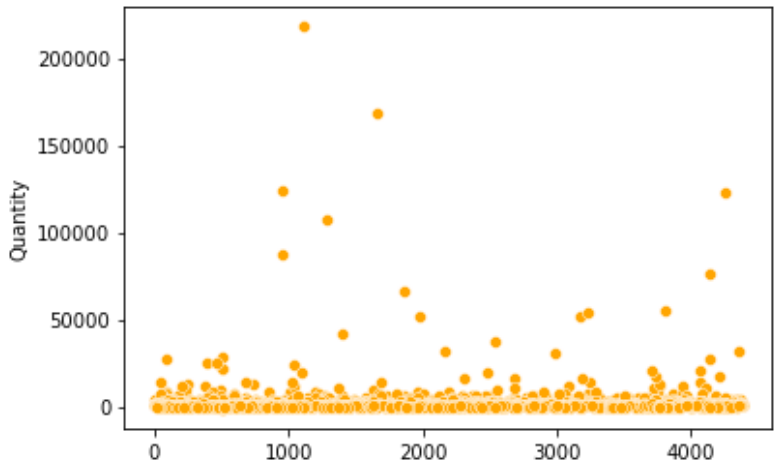
Plotting everything on a single scale

In [136]:

```
sns.scatterplot(x=customer_freq.index, y='Quantity',
                data=customer_table, color = "orange")
```

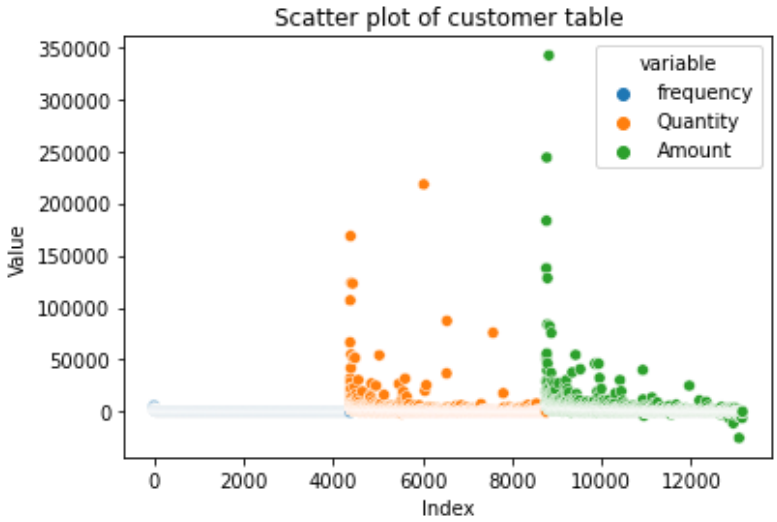
Out[136]:

```
<AxesSubplot:ylabel='Quantity'>
```



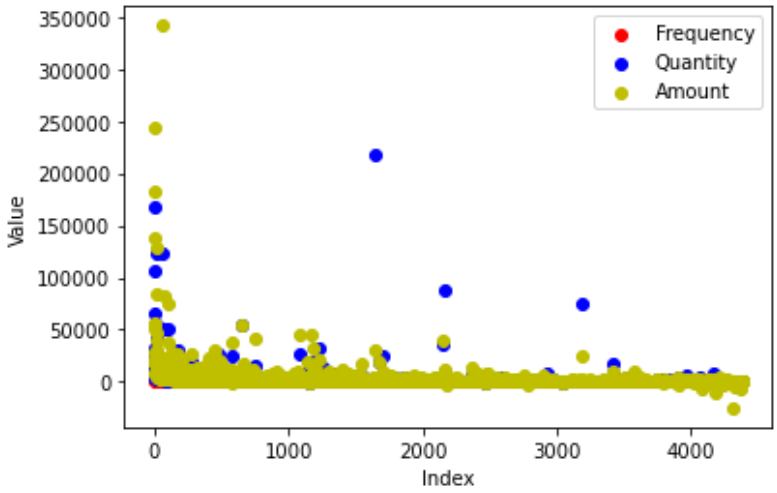
In [118]:

```
customer_table_melted = pd.melt(customer_table, value_vars=['frequency', 'Quantity', 'Amount']
sns.scatterplot(x=customer_table_melted.index, y='value', data=customer_table_melted, hue='var

plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Scatter plot of customer table')

plt.show()
```



In [122]:

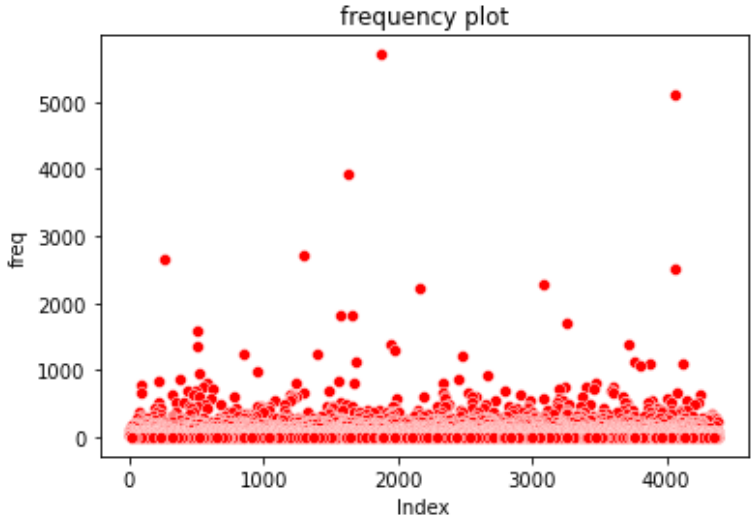
```
fig, ax = plt.subplots()
ax.scatter(customer_table.index, customer_table['frequency'], c='r', label='Frequency')
ax.scatter(customer_table.index, customer_table['Quantity'], c='b', label='Quantity')
ax.scatter(customer_table.index, customer_table['Amount'], c='y', label='Amount')

ax.legend()
ax.set_xlabel('Index')
ax.set_ylabel('Value')

plt.show()
```

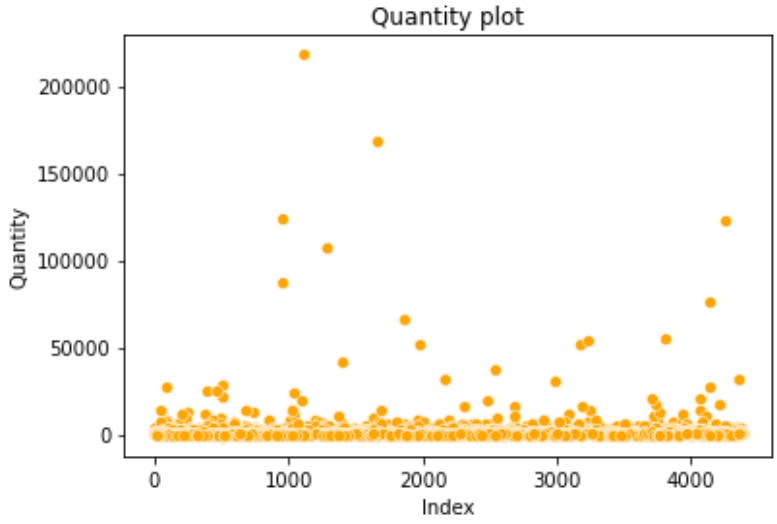


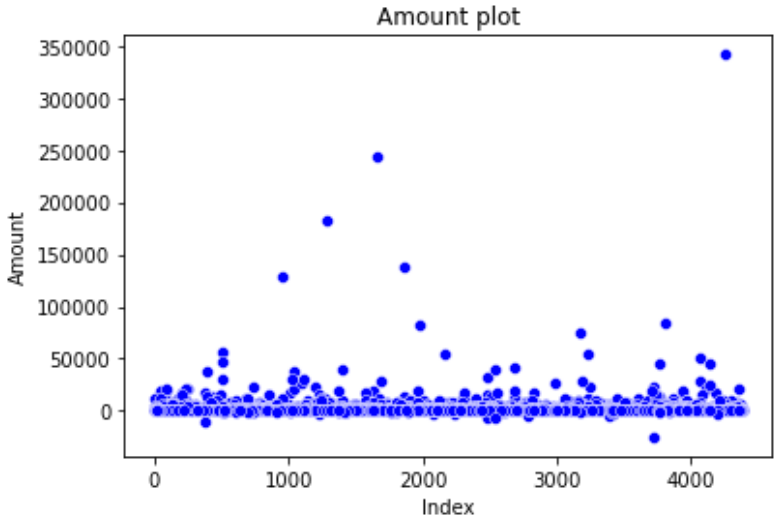
Plotting each three attribures frequency, Quantity and amount seperately and on the same graph to analyse the purchases

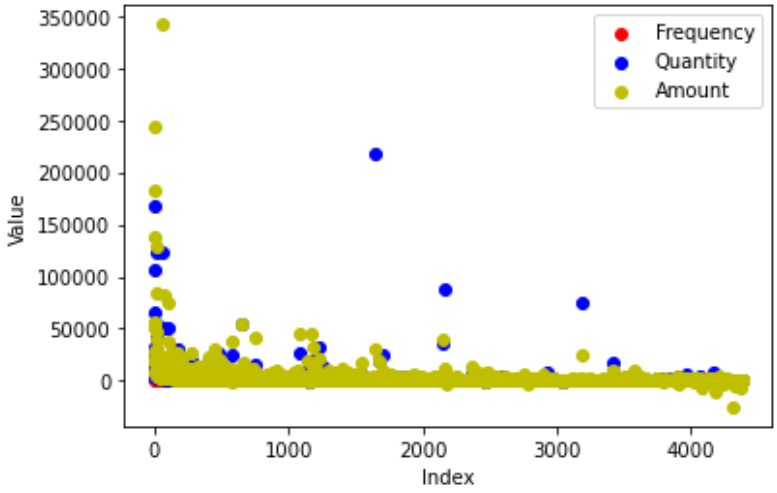
In [142]:

```
# plotting the frequncy
sns.scatterplot(x=customer_freq.index, y='frequency', data=customer_freq, color = "red")

plt.xlabel('Index')
plt.ylabel('freq')
plt.title('frequency plot')

plt.show()

# plotting the quantity
sns.scatterplot(x=customer_freq.index, y='Quantity',
                data=customer_table, color = "orange")

plt.xlabel('Index')
plt.ylabel('Quantity')
plt.title('Quantity plot')

plt.show()

# plotting the amount
sns.scatterplot(x=customer_freq.index, y='Amount',
                data=customer_table, color = "blue")

plt.xlabel('Index')
plt.ylabel('Amount')
plt.title('Amount plot')

plt.show()


# plotting in the same graph
fig, ax = plt.subplots()
ax.scatter(customer_table.index, customer_table['frequency'], c='r', label='Frequency')
ax.scatter(customer_table.index, customer_table['Quantity'], c='b', label='Quantity')
ax.scatter(customer_table.index, customer_table['Amount'], c='y', label='Amount')

ax.legend()
ax.set_xlabel('Index')
ax.set_ylabel('Value')

plt.show()
```





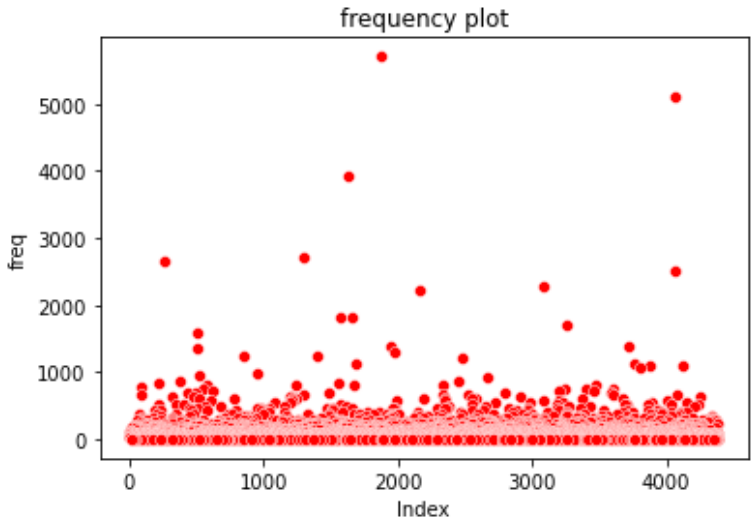




how many times a customer appeared

In [139]:

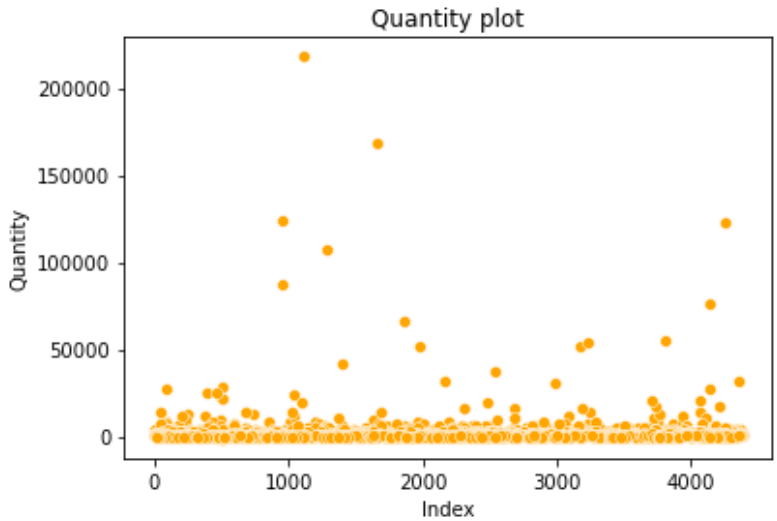
```
# plotting the frequncy
sns.scatterplot(x=customer_freq.index, y='frequency', data=customer_freq, color = "red")

plt.xlabel('Index')
plt.ylabel('freq')
plt.title('frequency plot')

plt.show()
```



how much quantity customer has purchased

In [141]:

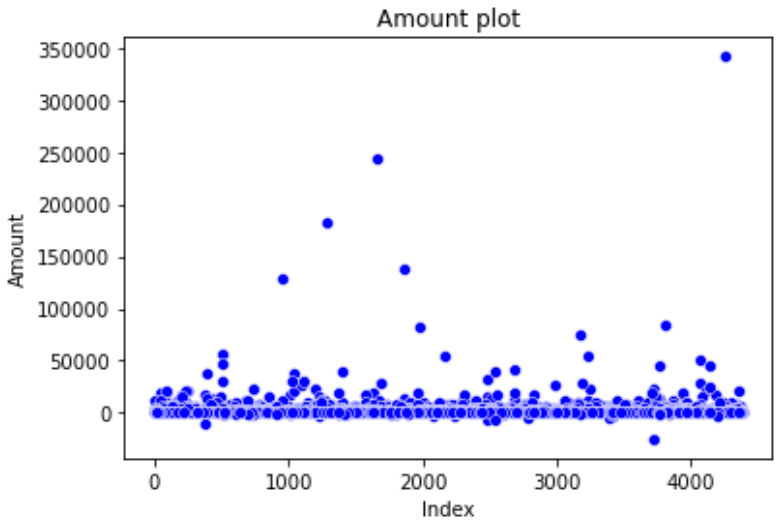
```
# plotting the quantity
sns.scatterplot(x=customer_freq.index, y='Quantity',
                data=customer_table, color = "orange")

plt.xlabel('Index')
plt.ylabel('Quantity')
plt.title('Quantity plot')

plt.show()
```



how much does a customer invested

In [143]:

```
# plotting the amount
sns.scatterplot(x=customer_freq.index, y='Amount',
                data=customer_table, color = "blue")

plt.xlabel('Index')
plt.ylabel('Amount')
plt.title('Amount plot')

plt.show()
```



Based on the above plots we can conclude that customers could be segmented into different categories as there are some customers purchased quite high amounts, some in normal range while some has negative values for amount, hence we can segment customers in three categories.

we can assign red, yellow, green tags to the customers

red for the customers which has negative amount purchased value, yellow for the normal buyers and green for customers which has its purchase amount above a certain limit

Segmenting the customes according to above criteria taking the limit for the yellow and red tags as 25K

RED CUSTOMERS

In [155]:

```
customer_table
```

Out[155]:

	Customer ID	frequency	Quantity	Amount
0	14911	5710	66561	137675.91
1	17841	5114	14255	29175.41
2	14606	3927	9322	18380.53
3	14156	2710	106885	183180.55
4	12748	2665	12862	20898.03
...	...	...	...	...
4378	17715	1	192	163.20
4379	18246	1	48	162.72
4380	16219	1	48	40.80
4381	12362	1	1	130.00
4382	15233	1	12	59.40

4383 rows × 4 columns

In [156]:

```
customer_table[customer_table['Amount']<=0]["Customer ID"].nunique()
```

Out[156]:

```
100
```

Hence there are 100 red customers

also they should'nt be treated as loyal customers

In [157]:

```
red_customers = customer_table[customer_table['Amount']<=0]
red_customers.reset_index(inplace = True, drop=True)
red_customers
```

Out[157]:

	Customer ID	frequency	Quantity	Amount
0	17017	181	3888	-486.70
1	13091	98	-1535	-450.04
2	12346	46	52	-64.68
3	14063	44	3846	-3767.20
4	14912	40	833	-92.94
...	...	...	...	...
95	17452	1	-1	-227.28
96	16151	1	-1	-4217.59
97	16154	1	-1	-10.50
98	17943	1	-3	-165.03
99	15849	1	-1	-5876.34

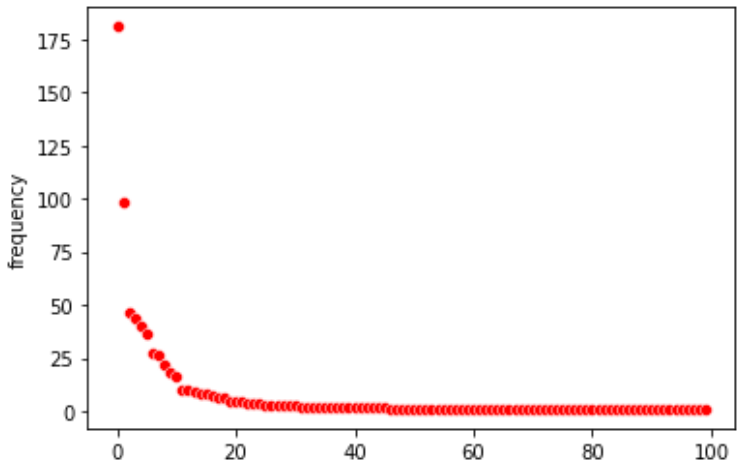
100 rows × 4 columns

In [175]:

```
sns.scatterplot(x=red_customers.index, y='frequency',
                data=red_customers, color = "red")

sns.scatterplot(x=red_customers.index, y='Quantity',
                data=red_customers, color = "red")
sns.scatterplot(x=red_customers.index, y='Amount',
                data=red_customers, color = "red")
```

Out[175]:

```
<AxesSubplot:ylabel='frequency'>
```



there are outliers

In [169]:

```
# checking the same for the green customers
green_customers = customer_table[customer_table['Amount']>20000]
green_customers.reset_index(inplace = True, drop=True)
green_customers['Customer ID'].nunique()
```

Out[169]:

```
40
```

In [171]:

```
green_customers.head()
```

Out[171]:

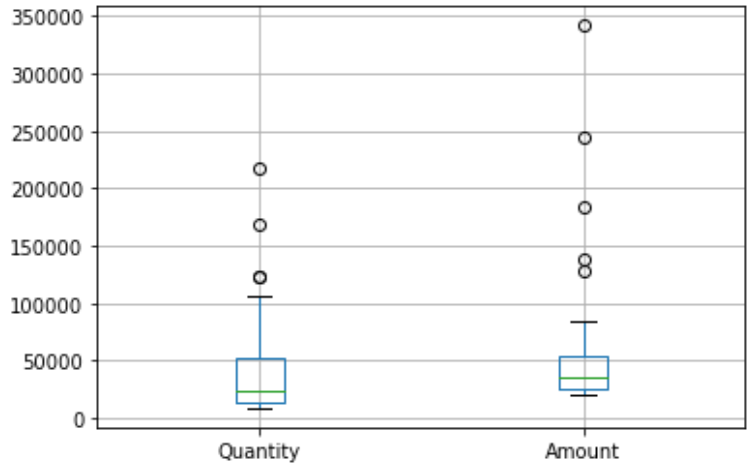
	Customer ID	frequency	Quantity	Amount
0	14911	5710	66561	137675.91
1	17841	5114	14255	29175.41
2	14156	2710	106885	183180.55
3	12748	2665	12862	20898.03
4	17850	2515	20901	50414.50

In [183]:

```
green_customers.boxplot(column=['Quantity','Amount'])
```

Out[183]:

```
<AxesSubplot:>
```



In [190]:

```
yellow_customers = customer_table[(customer_table['Amount'] < 20000) & (customer_table['Amount
# yellow_customers.reset_index(inplace = True, drop=True)
# yellow_customers['Customer ID'].nunique()
yellow_customers
```

Out[190]:

	Customer ID	frequency	Quantity	Amount
2	14606	3927	9322	18380.53
6	16549	2274	4908	9005.00
8	14527	1826	4506	18063.16
10	16782	1703	3562	8106.71
12	15005	1388	4548	7582.78
...	...	...	...	...
4378	17715	1	192	163.20
4379	18246	1	48	162.72
4380	16219	1	48	40.80
4381	12362	1	1	130.00
4382	15233	1	12	59.40

4243 rows × 4 columns

In [198]:

```
yellow_customers[yellow_customers['Quantity'] == 87167 ]
```

Out[198]:

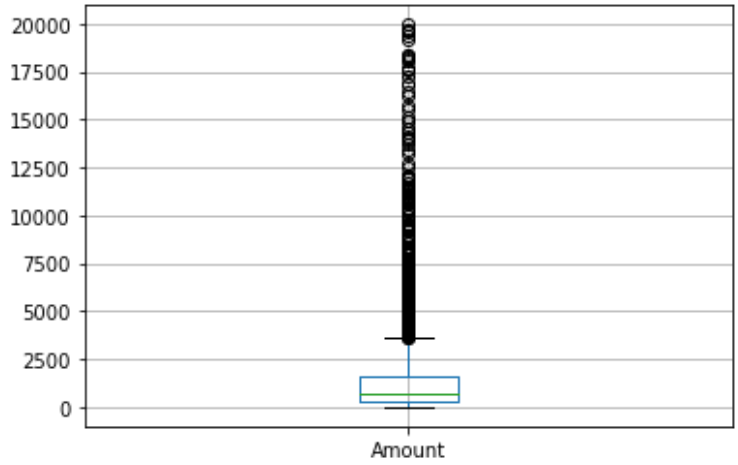
	Customer ID	frequency	Quantity	Amount
2161	13687	45	87167	11880.84

In [191]:

```
yellow_customers.boxplot(column='Amount')
```

Out[191]:

<AxesSubplot:>



In [194]:

```
yellow_customers['Quantity'].max()
```

Out[194]:

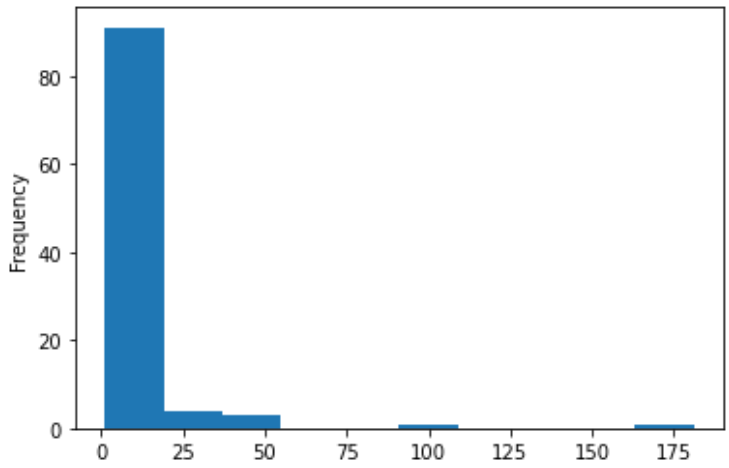
87167

In [211]:

```
red_customers['frequency'].plot.hist(bins=10)
```

Out[211]:

<AxesSubplot:ylabel='Frequency'>

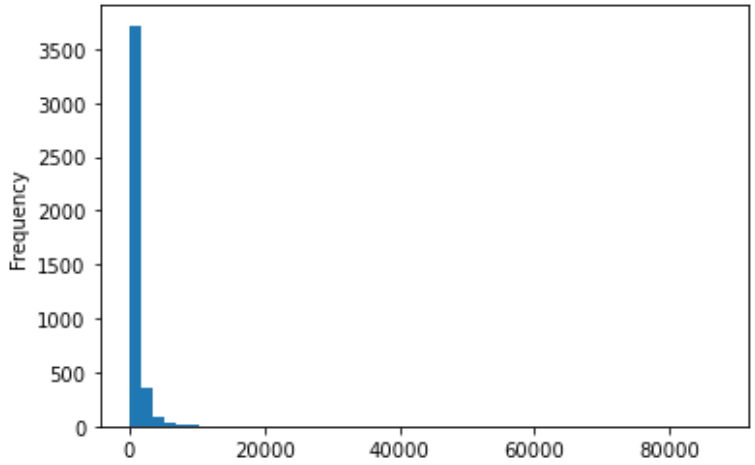


In [206]:

```
yellow_customers['Quantity'].plot.hist(bins=50)
```

Out[206]:

<AxesSubplot:ylabel='Frequency'>

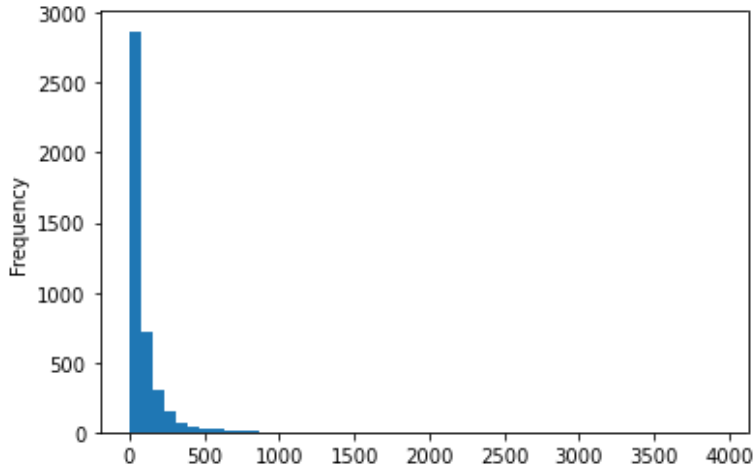


In [212]:

```
yellow_customers['frequency'].plot.hist(bins=50)
```

Out[212]:

<AxesSubplot:ylabel='Frequency'>

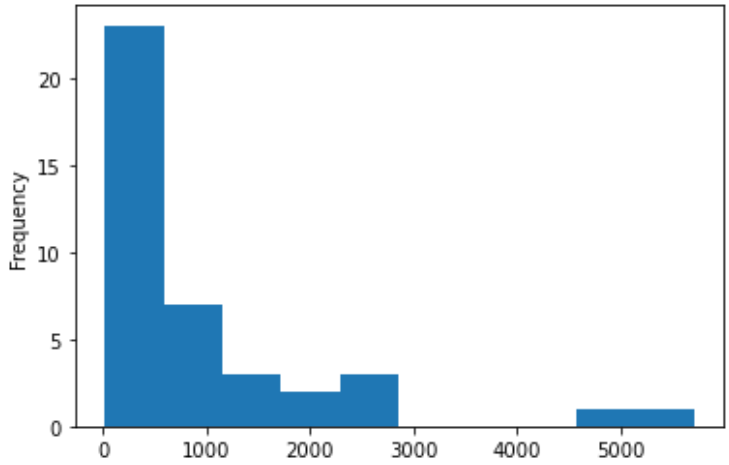


In [213]:

```
green_customers['frequency'].plot.hist(bins=10)
```

Out[213]:

<AxesSubplot:ylabel='Frequency'>

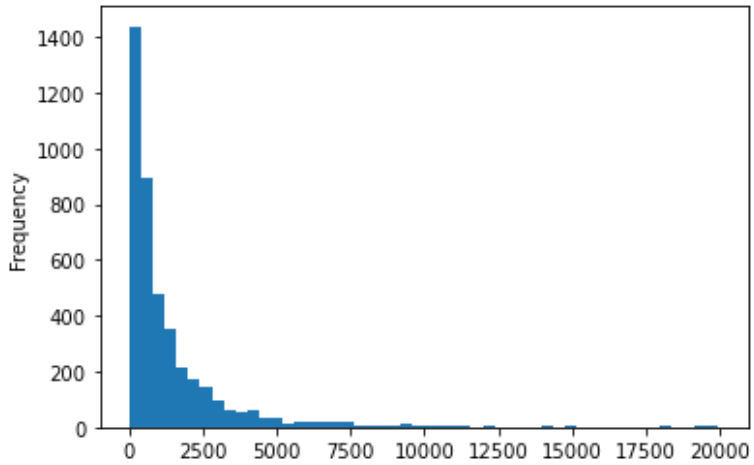


In [214]:

```
yellow_customers['Amount'].plot.hist(bins=50)
```

Out[214]:

<AxesSubplot:ylabel='Frequency'>



based on the above visualisation we could define a loyal customer as the ones who has there positive purchase amount without any returned products ie all the red customers can't be treated as the loyal customers

In [215]:

```
df
```

Out[215]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	Amou
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085	United Kingdom	83.
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	81.
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	81.
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085	United Kingdom	100.
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085	United Kingdom	30.
...	...	...	...	...	...	...	...	...	
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530	United Kingdom	5.
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530	United Kingdom	3.
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530	United Kingdom	3.
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530	United Kingdom	7.
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530	United Kingdom	3.

417534 rows × 9 columns

In [217]:

```
df['InvoiceDate']
```

Out[217]:

```
0         2009-12-01 07:45:00
1         2009-12-01 07:45:00
2         2009-12-01 07:45:00
3         2009-12-01 07:45:00
4         2009-12-01 07:45:00
                  ...
525456    2010-12-09 20:01:00
525457    2010-12-09 20:01:00
525458    2010-12-09 20:01:00
525459    2010-12-09 20:01:00
525460    2010-12-09 20:01:00
Name: InvoiceDate, Length: 417534, dtype: datetime64[ns]
```

In [264]:

```
df['year'] = df['InvoiceDate'].dt.year
df['month'] = df['InvoiceDate'].dt.month
df['day'] = df['InvoiceDate'].dt.day
df['hour'] = df['InvoiceDate'].dt.hour
time['weekday'] = df['InvoiceDate'].dt.weekday
```

```
C:\Users\Ashutosh Patidar\AppData\Local\Temp\ipykernel_27256\644059547.py:1: Set
tingWithCopyWarning:
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: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  df['year'] = df['InvoiceDate'].dt.year
C:\Users\Ashutosh Patidar\AppData\Local\Temp\ipykernel_27256\644059547.py:2: Set
tingWithCopyWarning:
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: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  df['month'] = df['InvoiceDate'].dt.month
C:\Users\Ashutosh Patidar\AppData\Local\Temp\ipykernel_27256\644059547.py:3: Set
tingWithCopyWarning:
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: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  df['day'] = df['InvoiceDate'].dt.day
C:\Users\Ashutosh Patidar\AppData\Local\Temp\ipykernel_27256\644059547.py:4: Set
tingWithCopyWarning:
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: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  df['hour'] = df['InvoiceDate'].dt.hour
C:\Users\Ashutosh Patidar\AppData\Local\Temp\ipykernel_27256\644059547.py:5: Set
tingWithCopyWarning:
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: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  time['weekday'] = df['InvoiceDate'].dt.weekday
```

In [265]:

```
df
```

Out[265]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	Amou
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085	United Kingdom	83.
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	81.
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	81.
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085	United Kingdom	100.
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085	United Kingdom	30.
...	...	...	...	...	...	...	...	...	
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530	United Kingdom	5.
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530	United Kingdom	3.
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530	United Kingdom	3.
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530	United Kingdom	7.
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530	United Kingdom	3.

417534 rows × 14 columns

ANALYSIS ON TIME

In [266]:

```
time = df[['year','month','day','hour','weekday']]
```

In [267]:

```
time['year'].nunique()
```

Out[267]:

```
2
```

In [268]:

```
len(time[time['year'] == 2009])
```

Out[268]:

```
31760
```

In [269]:

```
len(time[time['year'] == 2010])
```

Out[269]:

```
385774
```

In [270]:

```
time['year']
```

Out[270]:

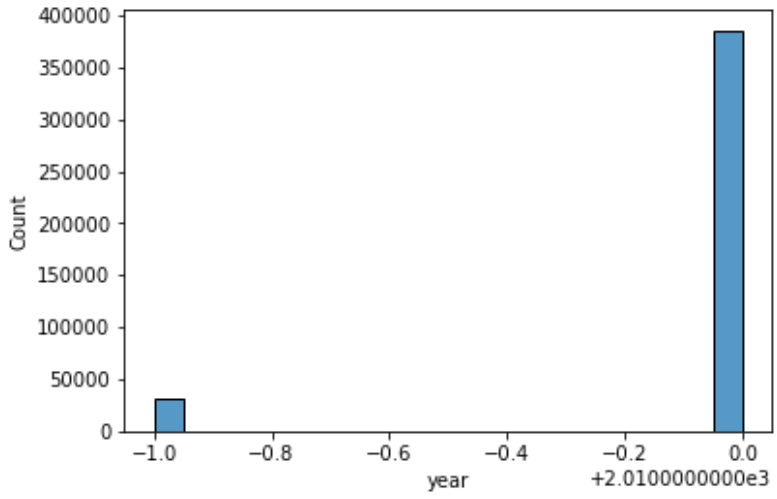
```
0         2009
1         2009
2         2009
3         2009
4         2009
          ...
525456    2010
525457    2010
525458    2010
525459    2010
525460    2010
Name: year, Length: 417534, dtype: int64
```

In [271]:

```
sns.histplot(data=time['year'])
```

Out[271]:

```
<AxesSubplot:xlabel='year', ylabel='Count'>
```



In [272]:

```
print(len(time[time['year'] == 2010])/len(df)*100)
```

92.39343382814333

We have mostly around 92.39% values for the year 2010 so we analysing for that year only

In [273]:

```
time_2010 = time[time["year"] == 2010]
```

In [274]:

```
time_2010
```

Out[274]:

	year	month	day	hour	weekday
45228	2010	1	4	9	0
45229	2010	1	4	9	0
45230	2010	1	4	9	0
45234	2010	1	4	10	0
45235	2010	1	4	10	0
...	...	...	...	...	...
525456	2010	12	9	20	3
525457	2010	12	9	20	3
525458	2010	12	9	20	3
525459	2010	12	9	20	3
525460	2010	12	9	20	3

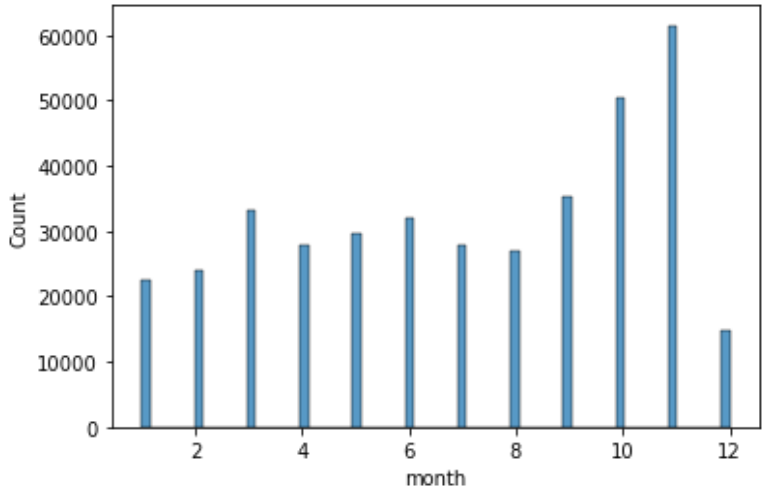
385774 rows × 5 columns

In [275]:

```
sns.histplot(data = time_2010['month'])
```

Out[275]:

<AxesSubplot:xlabel='month', ylabel='Count'>



we have the maximum sales in the month of NOVEMBER and then followed by OCTOBER

moreover we have peak sales in the months of september, october and november which supports the seasonality in the data

In [276]:

```
time_2009 = time[time["year"] == 2009]
time_2009
```

Out[276]:

	year	month	day	hour	weekday
0	2009	12	1	7	1
1	2009	12	1	7	1
2	2009	12	1	7	1
3	2009	12	1	7	1
4	2009	12	1	7	1
...	...	...	...	...	...
45219	2009	12	23	16	2
45220	2009	12	23	16	2
45221	2009	12	23	16	2
45222	2009	12	23	16	2
45227	2009	12	23	16	2

31760 rows × 5 columns

In [278]:

```
time_2009['month'].nunique()
```

Out[278]:

```
1
```

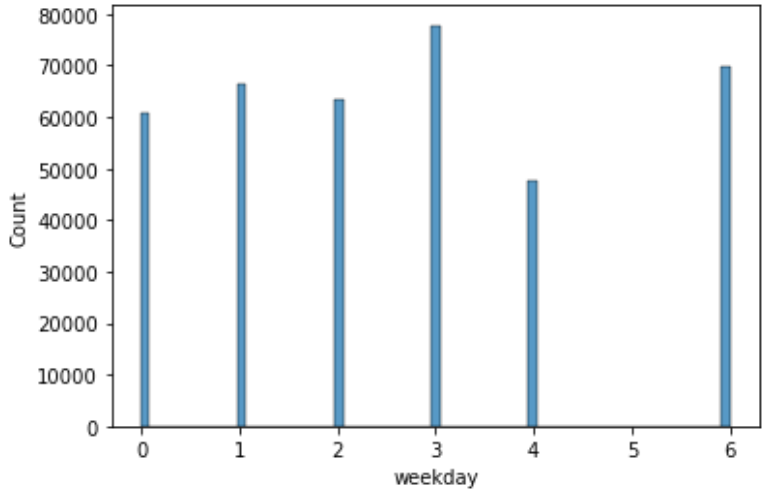
hence we have data only for december for the year 2009

In [281]:

```
sns.histplot(data = time_2010['weekday'])
```

Out[281]:

```
<AxesSubplot:xlabel='weekday', ylabel='Count'>
```



One very important thing to notice that we have nearly around no sales on SATURDAY

moreover

we have the maximum sales on the THURSDAY and lowest on FRIDAY but not that much difference as we daywise

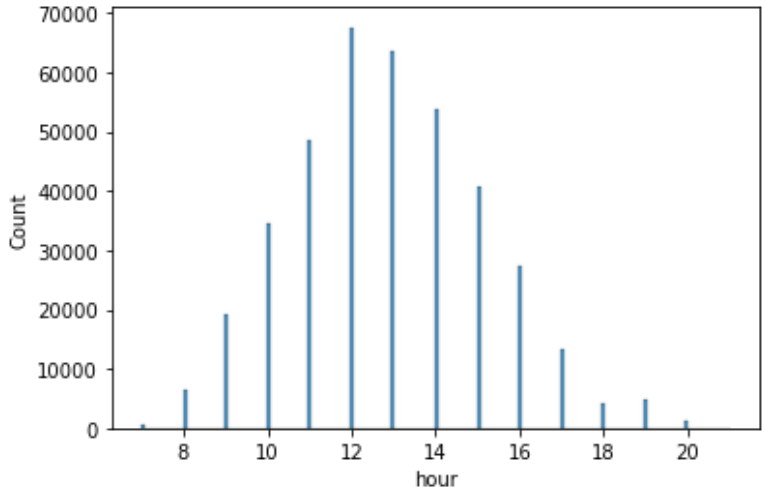
Now checking for the peak hour of the sale

In [283]:

```
sns.histplot(data = time_2010['hour'])
```

Out[283]:

```
<AxesSubplot:xlabel='hour', ylabel='Count'>
```



Here we could clearly see that everyday 12 PM and hours aound it is peak time for sale

Discussing customer's lifetime with respect to the given dataset.

considering the lifetime of the customer as the era where in the customer made his first and the last purchase form here

In [284]:

```
df
```

Out[284]:

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country	Amou
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085	United Kingdom	83.
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	81.
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085	United Kingdom	81.
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085	United Kingdom	100.
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085	United Kingdom	30.
...	...	...	...	...	...	...	...	...	
525456	538171	22271	FELTCRAFT DOLL ROSIE	2	2010-12-09 20:01:00	2.95	17530	United Kingdom	5.
525457	538171	22750	FELTCRAFT PRINCESS LOLA DOLL	1	2010-12-09 20:01:00	3.75	17530	United Kingdom	3.
525458	538171	22751	FELTCRAFT PRINCESS OLIVIA DOLL	1	2010-12-09 20:01:00	3.75	17530	United Kingdom	3.
525459	538171	20970	PINK FLORAL FELTCRAFT SHOULDER BAG	2	2010-12-09 20:01:00	3.75	17530	United Kingdom	7.
525460	538171	21931	JUMBO STORAGE BAG SUKI	2	2010-12-09 20:01:00	1.95	17530	United Kingdom	3.

417534 rows × 14 columns

In [293]:

```
max_time = df.groupby('Customer ID').max()['InvoiceDate']
min_time = df.groupby('Customer ID').min()['InvoiceDate']
```

```
C:\Users\Ashutosh Patidar\AppData\Local\Temp\ipykernel_27256\1000226141.py:1: Fu
tureWarning: Dropping invalid columns in DataFrameGroupBy.max is deprecated. In
a future version, a TypeError will be raised. Before calling .max, select only c
olumns which should be valid for the function.
  max_time = df.groupby('Customer ID').max()['InvoiceDate']
C:\Users\Ashutosh Patidar\AppData\Local\Temp\ipykernel_27256\1000226141.py:2: Fu
tureWarning: Dropping invalid columns in DataFrameGroupBy.min is deprecated. In
a future version, a TypeError will be raised. Before calling .min, select only c
olumns which should be valid for the function.
  min_time = df.groupby('Customer ID').min()['InvoiceDate']
```

In [295]:

```
max_time
```

Out[295]:

```
Customer ID
12346   2010-10-04 16:33:00
12347   2010-12-07 14:57:00
12348   2010-09-27 14:59:00
12349   2010-10-28 08:23:00
12351   2010-11-29 15:23:00
               ...
18283   2010-11-22 15:30:00
18284   2010-10-06 12:31:00
18285   2010-02-17 10:24:00
18286   2010-08-20 11:57:00
18287   2010-11-22 11:51:00
Name: InvoiceDate, Length: 4383, dtype: datetime64[ns]
```

In [303]:

```
min_time
```

Out[303]:

```
Customer ID
12346   2009-12-14 08:34:00
12347   2010-10-31 14:20:00
12348   2010-09-27 14:59:00
12349   2009-12-04 12:49:00
12351   2010-11-29 15:23:00
               ...
18283   2010-02-19 17:16:00
18284   2010-10-04 11:33:00
18285   2010-02-17 10:24:00
18286   2009-12-16 10:45:00
18287   2009-12-01 14:19:00
Name: InvoiceDate, Length: 4383, dtype: datetime64[ns]
```

In [298]:

```
customer_table
```

Out[298]:

	Customer ID	frequency	Quantity	Amount
0	14911	5710	66561	137675.91
1	17841	5114	14255	29175.41
2	14606	3927	9322	18380.53
3	14156	2710	106885	183180.55
4	12748	2665	12862	20898.03
...	...	...	...	...
4378	17715	1	192	163.20
4379	18246	1	48	162.72
4380	16219	1	48	40.80
4381	12362	1	1	130.00
4382	15233	1	12	59.40

4383 rows × 4 columns

In [306]:

```
len(max_time)
```

Out[306]:

```
4383
```

In [310]:

```
max_time
```

Out[310]:

```
Customer ID
12346   2010-10-04 16:33:00
12347   2010-12-07 14:57:00
12348   2010-09-27 14:59:00
12349   2010-10-28 08:23:00
12351   2010-11-29 15:23:00
                ...
18283   2010-11-22 15:30:00
18284   2010-10-06 12:31:00
18285   2010-02-17 10:24:00
18286   2010-08-20 11:57:00
18287   2010-11-22 11:51:00
Name: InvoiceDate, Length: 4383, dtype: datetime64[ns]
```

In [316]:

```
min_time
```

Out[316]:

```
Customer ID
12346   2009-12-14 08:34:00
12347   2010-10-31 14:20:00
12348   2010-09-27 14:59:00
12349   2009-12-04 12:49:00
12351   2010-11-29 15:23:00
               ...
18283   2010-02-19 17:16:00
18284   2010-10-04 11:33:00
18285   2010-02-17 10:24:00
18286   2009-12-16 10:45:00
18287   2009-12-01 14:19:00
Name: InvoiceDate, Length: 4383, dtype: datetime64[ns]
```

In [ ]:

```
customer_table
```

In [313]:

```
customer_table['max_time']
```

Out[313]:

```
0      NaT
1      NaT
2      NaT
3      NaT
4      NaT
        ..
4378   NaT
4379   NaT
4380   NaT
4381   NaT
4382   NaT
Name: max_time, Length: 4383, dtype: datetime64[ns]
```

In [317]:

```
max_time.reset_index()
# customer_table = pd.merge(max_time, customer_table, on='Customer ID')
```

Out[317]:

	Customer ID	InvoiceDate
0	12346	2010-10-04 16:33:00
1	12347	2010-12-07 14:57:00
2	12348	2010-09-27 14:59:00
3	12349	2010-10-28 08:23:00
4	12351	2010-11-29 15:23:00
...	...	...
4378	18283	2010-11-22 15:30:00
4379	18284	2010-10-06 12:31:00
4380	18285	2010-02-17 10:24:00
4381	18286	2010-08-20 11:57:00
4382	18287	2010-11-22 11:51:00

4383 rows × 2 columns

In [319]:

```
customer_table = pd.merge(max_time, customer_table, on='Customer ID')
```

In [321]:

```
min_time.reset_index()
```

Out[321]:

	Customer ID	InvoiceDate
0	12346	2009-12-14 08:34:00
1	12347	2010-10-31 14:20:00
2	12348	2010-09-27 14:59:00
3	12349	2009-12-04 12:49:00
4	12351	2010-11-29 15:23:00
...	...	...
4378	18283	2010-02-19 17:16:00
4379	18284	2010-10-04 11:33:00
4380	18285	2010-02-17 10:24:00
4381	18286	2009-12-16 10:45:00
4382	18287	2009-12-01 14:19:00

4383 rows × 2 columns

In [322]:

```
customer_table = pd.merge(min_time, customer_table, on='Customer ID')
customer_table
```

```
C:\Users\Ashutosh Patidar\AppData\Local\Temp\ipykernel_27256\1479712212.py:1: Fu
tureWarning: Passing 'suffixes' which cause duplicate columns {'InvoiceDate_y'}
in the result is deprecated and will raise a MergeError in a future version.
  customer_table = pd.merge(min_time, customer_table, on='Customer ID')
```

Out[322]:

	Customer ID	InvoiceDate_x	InvoiceDate_y	InvoiceDate_x	InvoiceDate_y	frequency	Quantity	A
0	12346	2009-12-14 08:34:00	2010-10-04 16:33:00	2010-10-04 16:33:00	2010-10-04 16:33:00	46	52	
1	12347	2010-10-31 14:20:00	2010-12-07 14:57:00	2010-12-07 14:57:00	2010-12-07 14:57:00	71	828	1
2	12348	2010-09-27 14:59:00	2010-09-27 14:59:00	2010-09-27 14:59:00	2010-09-27 14:59:00	20	373	
3	12349	2009-12-04 12:49:00	2010-10-28 08:23:00	2010-10-28 08:23:00	2010-10-28 08:23:00	107	988	2
4	12351	2010-11-29 15:23:00	2010-11-29 15:23:00	2010-11-29 15:23:00	2010-11-29 15:23:00	21	261	
...	...	...	...	...	...	...	...	
4378	18283	2010-02-19 17:16:00	2010-11-22 15:30:00	2010-11-22 15:30:00	2010-11-22 15:30:00	230	336	
4379	18284	2010-10-04 11:33:00	2010-10-06 12:31:00	2010-10-06 12:31:00	2010-10-06 12:31:00	29	493	
4380	18285	2010-02-17 10:24:00	2010-02-17 10:24:00	2010-02-17 10:24:00	2010-02-17 10:24:00	12	145	
4381	18286	2009-12-16 10:45:00	2010-08-20 11:57:00	2010-08-20 11:57:00	2010-08-20 11:57:00	70	592	1
4382	18287	2009-12-01 14:19:00	2010-11-22 11:51:00	2010-11-22 11:51:00	2010-11-22 11:51:00	86	1425	2

4383 rows × 10 columns

In [326]:

```
customer_table
```

Out[326]:

	Customer ID	InvoiceDate_x	InvoiceDate_y	InvoiceDate_x	InvoiceDate_y	frequency	Quantity	A
0	12346	2009-12-14 08:34:00	2010-10-04 16:33:00	2010-10-04 16:33:00	2010-10-04 16:33:00	46	52	
1	12347	2010-10-31 14:20:00	2010-12-07 14:57:00	2010-12-07 14:57:00	2010-12-07 14:57:00	71	828	1
2	12348	2010-09-27 14:59:00	2010-09-27 14:59:00	2010-09-27 14:59:00	2010-09-27 14:59:00	20	373	
3	12349	2009-12-04 12:49:00	2010-10-28 08:23:00	2010-10-28 08:23:00	2010-10-28 08:23:00	107	988	2
4	12351	2010-11-29 15:23:00	2010-11-29 15:23:00	2010-11-29 15:23:00	2010-11-29 15:23:00	21	261	
...	...	...	...	...	...	...	...	
4378	18283	2010-02-19 17:16:00	2010-11-22 15:30:00	2010-11-22 15:30:00	2010-11-22 15:30:00	230	336	
4379	18284	2010-10-04 11:33:00	2010-10-06 12:31:00	2010-10-06 12:31:00	2010-10-06 12:31:00	29	493	
4380	18285	2010-02-17 10:24:00	2010-02-17 10:24:00	2010-02-17 10:24:00	2010-02-17 10:24:00	12	145	
4381	18286	2009-12-16 10:45:00	2010-08-20 11:57:00	2010-08-20 11:57:00	2010-08-20 11:57:00	70	592	1
4382	18287	2009-12-01 14:19:00	2010-11-22 11:51:00	2010-11-22 11:51:00	2010-11-22 11:51:00	86	1425	2

4383 rows × 10 columns

In [328]:

```
customer_table = customer_table.T.drop_duplicates().T
customer_table
```

Out[328]:

	Customer ID	InvoiceDate_x	InvoiceDate_y	frequency	Quantity	Amount	max_time
0	12346	2009-12-14 08:34:00	2010-10-04 16:33:00	46	52	-64.68	NaT
1	12347	2010-10-31 14:20:00	2010-12-07 14:57:00	71	828	1323.32	NaT
2	12348	2010-09-27 14:59:00	2010-09-27 14:59:00	20	373	222.16	NaT
3	12349	2009-12-04 12:49:00	2010-10-28 08:23:00	107	988	2646.99	NaT
4	12351	2010-11-29 15:23:00	2010-11-29 15:23:00	21	261	300.93	NaT
...	...	...	...	...	...	...	...
4378	18283	2010-02-19 17:16:00	2010-11-22 15:30:00	230	336	641.77	NaT
4379	18284	2010-10-04 11:33:00	2010-10-06 12:31:00	29	493	436.68	NaT
4380	18285	2010-02-17 10:24:00	2010-02-17 10:24:00	12	145	427.0	NaT
4381	18286	2009-12-16 10:45:00	2010-08-20 11:57:00	70	592	1188.43	NaT
4382	18287	2009-12-01 14:19:00	2010-11-22 11:51:00	86	1425	2340.61	NaT

4383 rows × 7 columns

In [330]:

```
customer_table.drop(['max_time'], axis = 1)
```

Out[330]:

	Customer ID	InvoiceDate_x	InvoiceDate_y	frequency	Quantity	Amount
0	12346	2009-12-14 08:34:00	2010-10-04 16:33:00	46	52	-64.68
1	12347	2010-10-31 14:20:00	2010-12-07 14:57:00	71	828	1323.32
2	12348	2010-09-27 14:59:00	2010-09-27 14:59:00	20	373	222.16
3	12349	2009-12-04 12:49:00	2010-10-28 08:23:00	107	988	2646.99
4	12351	2010-11-29 15:23:00	2010-11-29 15:23:00	21	261	300.93
...	...	...	...	...	...	...
4378	18283	2010-02-19 17:16:00	2010-11-22 15:30:00	230	336	641.77
4379	18284	2010-10-04 11:33:00	2010-10-06 12:31:00	29	493	436.68
4380	18285	2010-02-17 10:24:00	2010-02-17 10:24:00	12	145	427.0
4381	18286	2009-12-16 10:45:00	2010-08-20 11:57:00	70	592	1188.43
4382	18287	2009-12-01 14:19:00	2010-11-22 11:51:00	86	1425	2340.61

4383 rows × 6 columns

In [333]:

```
customer_table = customer_table.rename(columns={'InvoiceDate_x': 'min_time', "InvoiceDate_y":
customer_table
```

Out[333]:

	Customer ID	min_time	max_time	frequency	Quantity	Amount	max_time
0	12346	2009-12-14 08:34:00	2010-10-04 16:33:00	46	52	-64.68	NaT
1	12347	2010-10-31 14:20:00	2010-12-07 14:57:00	71	828	1323.32	NaT
2	12348	2010-09-27 14:59:00	2010-09-27 14:59:00	20	373	222.16	NaT
3	12349	2009-12-04 12:49:00	2010-10-28 08:23:00	107	988	2646.99	NaT
4	12351	2010-11-29 15:23:00	2010-11-29 15:23:00	21	261	300.93	NaT
...	...	...	...	...	...	...	...
4378	18283	2010-02-19 17:16:00	2010-11-22 15:30:00	230	336	641.77	NaT
4379	18284	2010-10-04 11:33:00	2010-10-06 12:31:00	29	493	436.68	NaT
4380	18285	2010-02-17 10:24:00	2010-02-17 10:24:00	12	145	427.0	NaT
4381	18286	2009-12-16 10:45:00	2010-08-20 11:57:00	70	592	1188.43	NaT
4382	18287	2009-12-01 14:19:00	2010-11-22 11:51:00	86	1425	2340.61	NaT

4383 rows × 7 columns

In [337]:

```
customer_table.columns.values[6] = 'min_max2'
```

In [341]:

```
customer_table.drop("min_max2", axis = 1)
```

Out[341]:

	Customer ID	min_time	max_time	frequency	Quantity	Amount
0	12346	2009-12-14 08:34:00	2010-10-04 16:33:00	46	52	-64.68
1	12347	2010-10-31 14:20:00	2010-12-07 14:57:00	71	828	1323.32
2	12348	2010-09-27 14:59:00	2010-09-27 14:59:00	20	373	222.16
3	12349	2009-12-04 12:49:00	2010-10-28 08:23:00	107	988	2646.99
4	12351	2010-11-29 15:23:00	2010-11-29 15:23:00	21	261	300.93
...	...	...	...	...	...	...
4378	18283	2010-02-19 17:16:00	2010-11-22 15:30:00	230	336	641.77
4379	18284	2010-10-04 11:33:00	2010-10-06 12:31:00	29	493	436.68
4380	18285	2010-02-17 10:24:00	2010-02-17 10:24:00	12	145	427.0
4381	18286	2009-12-16 10:45:00	2010-08-20 11:57:00	70	592	1188.43
4382	18287	2009-12-01 14:19:00	2010-11-22 11:51:00	86	1425	2340.61

4383 rows × 6 columns

calculating the lifetime

In [343]:

```
customer_table['lifetime'] = customer_table['max_time'] - customer_table['min_time']
```

In [347]:

```
customer_table['lifetime_in_days'] = customer_table['lifetime'].dt.days
customer_table
```

Out[347]:

	Customer ID	min_time	max_time	frequency	Quantity	Amount	min_max2	lifetime	lifetime_in_
0	12346	2009-12-14 08:34:00	2010-10-04 16:33:00	46	52	-64.68	NaT	294 days 07:59:00	
1	12347	2010-10-31 14:20:00	2010-12-07 14:57:00	71	828	1323.32	NaT	37 days 00:37:00	
2	12348	2010-09-27 14:59:00	2010-09-27 14:59:00	20	373	222.16	NaT	0 days 00:00:00	
3	12349	2009-12-04 12:49:00	2010-10-28 08:23:00	107	988	2646.99	NaT	327 days 19:34:00	
4	12351	2010-11-29 15:23:00	2010-11-29 15:23:00	21	261	300.93	NaT	0 days 00:00:00	
...	...	...	...	...	...	...	...	...	
4378	18283	2010-02-19 17:16:00	2010-11-22 15:30:00	230	336	641.77	NaT	275 days 22:14:00	
4379	18284	2010-10-04 11:33:00	2010-10-06 12:31:00	29	493	436.68	NaT	2 days 00:58:00	
4380	18285	2010-02-17 10:24:00	2010-02-17 10:24:00	12	145	427.0	NaT	0 days 00:00:00	
4381	18286	2009-12-16 10:45:00	2010-08-20 11:57:00	70	592	1188.43	NaT	247 days 01:12:00	
4382	18287	2009-12-01 14:19:00	2010-11-22 11:51:00	86	1425	2340.61	NaT	355 days 21:32:00	

4383 rows × 9 columns

In [348]:

```
lifetime_table = customer_table[['Customer ID','lifetime_in_days']]
```

In [349]:

```
lifetime_table
```

Out[349]:

	Customer ID	lifetime_in_days
0	12346	294
1	12347	37
2	12348	0
3	12349	327
4	12351	0
...	...	...
4378	18283	275
4379	18284	2
4380	18285	0
4381	18286	247
4382	18287	355

4383 rows × 2 columns

In [ ]: