In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
df=pd.read_excel('Online Retail.xlsx')
```

In [3]:
```
df.head()
```

Out[3]:

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

In [4]:
```
df.tail()
```

Out[4]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France

In [5]:
```
df['Description'].value_counts()
```

Out[5]:
```
WHITE HANGING HEART T-LIGHT HOLDER     2369
REGENCY CAKESTAND 3 TIER               2200
JUMBO BAG RED RETROSPOT                2159
PARTY BUNTING                          1727
LUNCH BAG RED RETROSPOT                1638
                                       ...
Missing                                   1
historic computer difference?....se       1
DUSTY PINK CHRISTMAS TREE 30CM            1
WRAP BLUE RUSSIAN FOLKART                 1
PINK BERTIE MOBILE PHONE CHARM            1
Name: Description, Length: 4223, dtype: int64
```

In [6]:
```
df.head()
```

Out[6]:

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

In [7]:
```
df.shape
```

Out[7]:
```
(541909, 8)
```

In [8]:
```
df.columns
```

Out[8]:
```
Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
```

In [9]:
```
df['UnitPrice'].value_counts()
```

Out[9]:
```
1.25      50496
1.65      38181
0.85      28497
2.95      27768
0.42      24533
          ...
84.21         1
46.86         1
28.66         1
156.45        1
224.69        1
Name: UnitPrice, Length: 1630, dtype: int64
```

In [10]:
```
df.describe()
```

Out[10]:

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

1.a) Missing data

In [11]:
```
df.isna().sum()
```

Out[11]:
```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

In [12]:
```
cols=['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country']

percentage_missinig_data=(df.isna().sum()/len(df))*100
print("percenatge_of_missing_data_for_each_feature")
print('{}'.format(percentage_missinig_data))
```

```
percenatge_of_missing_data_for_each_feature
InvoiceNo       0.000000
StockCode       0.000000
Description     0.268311
Quantity        0.000000
InvoiceDate     0.000000
UnitPrice       0.000000
CustomerID     24.926694
Country         0.000000
dtype: float64
```

We are now seeing here customerID has below 25%missing data so we can drop this missung data and Description has less than 1% missing values so we can drop this missong data.If missing data more 30% then generally we cant drop.

In [13]:
```
df.dropna(axis=0,inplace=True)
```

In [14]:
```
df.head()
```

Out[14]:

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

In [15]:
```
df.shape
```

Out[15]:
```
(406829, 8)
```

In [16]:
```
df.isna().sum()
```

Out[16]:
```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

In [17]:
```
### we are seeing here Quantity and Unitprice having min values negative side which is not possible
```

In [18]:
```
df.loc[df['Quantity']<0]
```

Out[18]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
141	C536379	D	Discount	-1	2010-12-01 09:41:00	27.50	14527.0	United Kingdom
154	C536383	35004C	SET OF 3 COLOURED FLYING DUCKS	-1	2010-12-01 09:49:00	4.65	15311.0	United Kingdom
235	C536391	22556	PLASTERS IN TIN CIRCUS PARADE	-12	2010-12-01 10:24:00	1.65	17548.0	United Kingdom
236	C536391	21984	PACK OF 12 PINK PAISLEY TISSUES	-24	2010-12-01 10:24:00	0.29	17548.0	United Kingdom
237	C536391	21983	PACK OF 12 BLUE PAISLEY TISSUES	-24	2010-12-01 10:24:00	0.29	17548.0	United Kingdom
...	...	...	...	...	...	...	...	...
540449	C581490	23144	ZINC T-LIGHT HOLDER STARS SMALL	-11	2011-12-09 09:57:00	0.83	14397.0	United Kingdom
541541	C581499	M	Manual	-1	2011-12-09 10:28:00	224.69	15498.0	United Kingdom
541715	C581568	21258	VICTORIAN SEWING BOX LARGE	-5	2011-12-09 11:57:00	10.95	15311.0	United Kingdom
541716	C581569	84978	HANGING HEART JAR T-LIGHT HOLDER	-1	2011-12-09 11:58:00	1.25	17315.0	United Kingdom
541717	C581569	20979	36 PENCILS TUBE RED RETROSPOT	-5	2011-12-09 11:58:00	1.25	17315.0	United Kingdom

8905 rows × 8 columns

In [19]:
```
df=df[(df['Quantity'] >0) & (df['UnitPrice'] >0)]
#df = df[(df['Quantity'] > 0) & (df['Price'] > 0)]
```

In [20]:
```
df.describe()
```

Out[20]:

	Quantity	UnitPrice	CustomerID
count	397884.000000	397884.000000	397884.000000
mean	12.988238	3.116488	15294.423453
std	179.331775	22.097877	1713.141560
min	1.000000	0.001000	12346.000000
25%	2.000000	1.250000	13969.000000
50%	6.000000	1.950000	15159.000000
75%	12.000000	3.750000	16795.000000
max	80995.000000	8142.750000	18287.000000

noe here we are seeing all values are postive

1.b) Treatment to duplicate data records

In [21]:
```
df.duplicated().sum()
```
Out[21]:
```
5192
```

In [22]:
```
df=df.drop_duplicates(keep=False)
```

In [23]:
```
df.duplicated().sum()
```
Out[23]:
```
0
```

In [24]:
```
df.shape
```
Out[24]:
```
(387883, 8)
```

Findings-Here we have treated duplicated values from dataset,previously it was 5225 now we can see there is no duplicate value there.

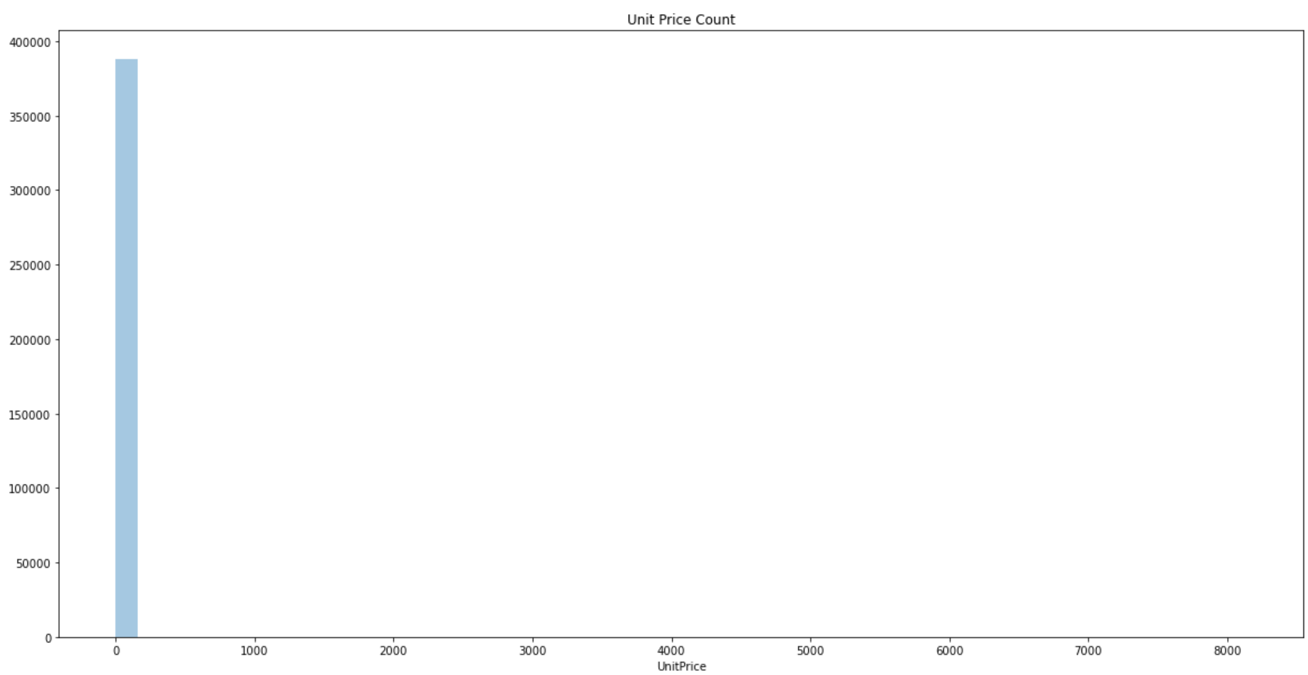
1.c) Descriptive analytics on the given data

In [25]:
```
df.head()
```
Out[25]:

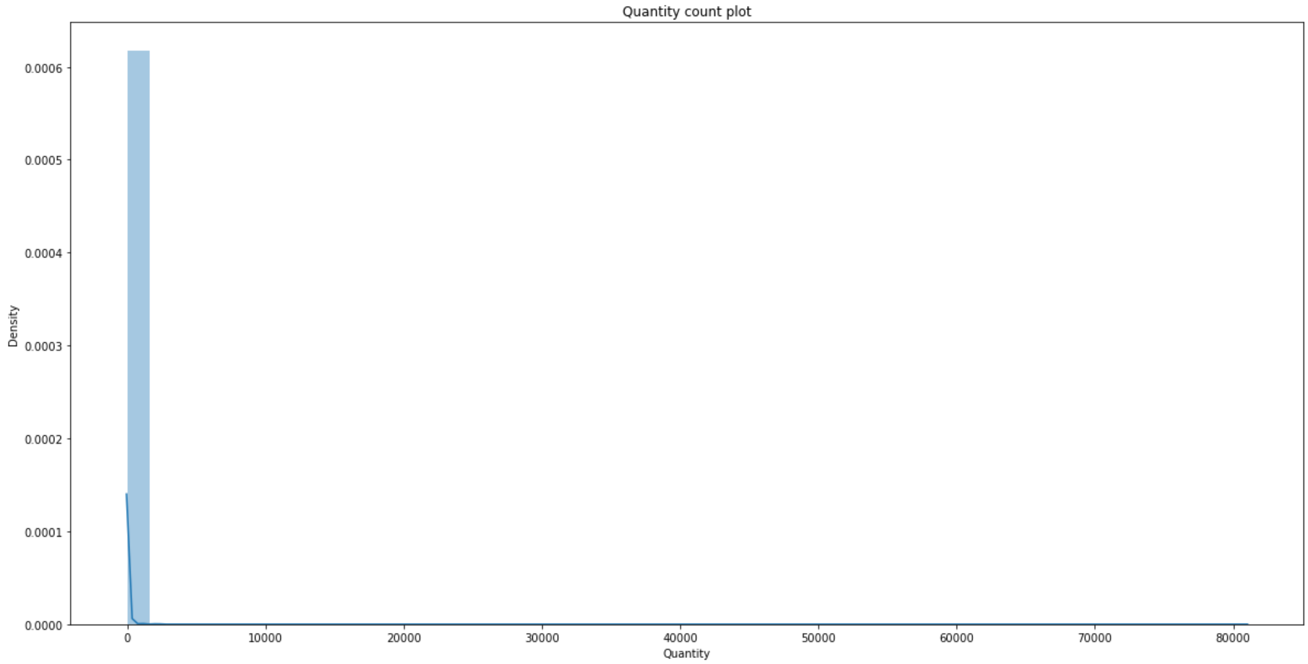
	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS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

In [26]:
```
plt.figure(figsize=(20,10))
sns.distplot(df["UnitPrice"],kde=False)
plt.title('Unit Price Count')
plt.show()
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



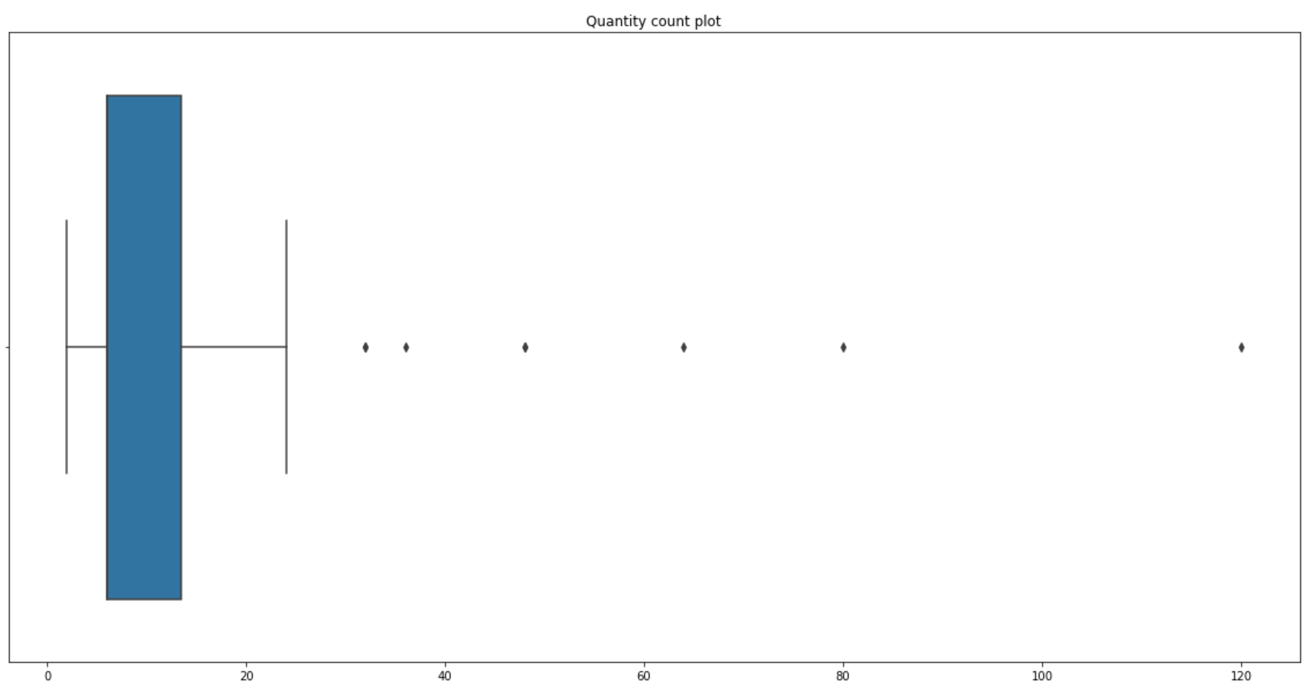
In [27]:
```
plt.figure(figsize=(20,10))
sns.distplot(df['Quantity'],kde=True)
plt.title('Quantity count plot')
plt.show()
```



In [28]:
```
df['box_Quantity']=df['Quantity'][:100]
df['box_Quantity'].head()
```

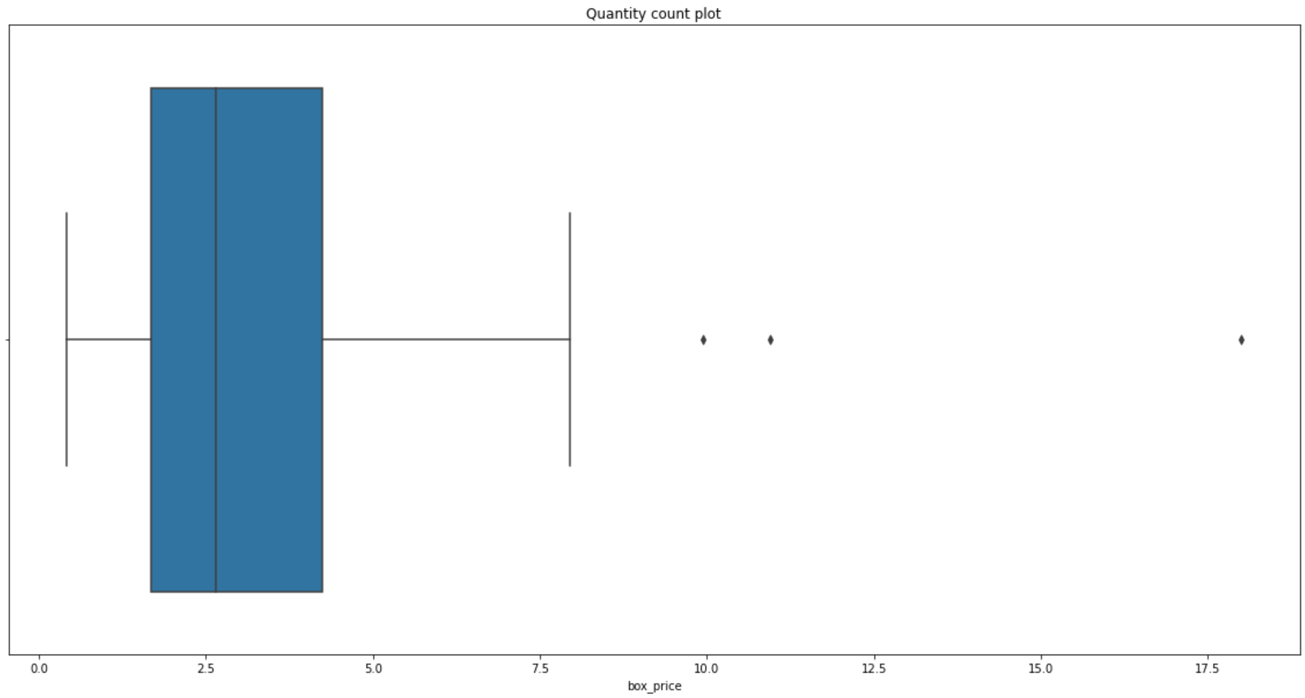
Out[28]:
```
0    6.0
1    6.0
2    8.0
3    6.0
4    6.0
Name: box_Quantity, dtype: float64
```

In [29]:
```

plt.figure(figsize=(20,10))
sns.boxplot(x='box_Quantity',data=df)
plt.title('Quantity count plot')
plt.show()
```



In [30]:
```
df['box_price']=df['UnitPrice'][:100]
```

In [31]:
```
plt.figure(figsize=(20,10))
sns.boxplot(x='box_price',data=df)
plt.title('Quantity count plot')
plt.show()
```



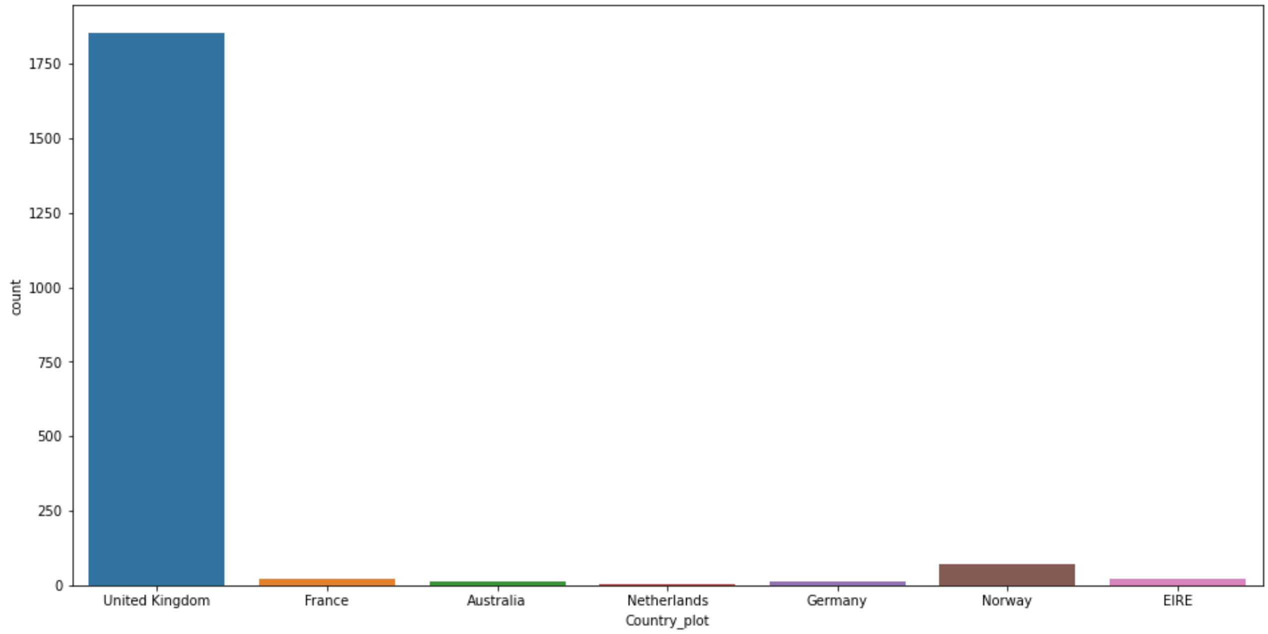
maximum datapoints in Quantity and Unit price are located near to zero.

In [32]:
```
df['Country'].value_counts()
```

Out[32]:
```
United Kingdom          344466
Germany                   9010
France                    8311
EIRE                      7216
Spain                     2474
Netherlands               2359
Belgium                   2031
Switzerland               1841
Portugal                  1445
Australia                 1180
Norway                    1071
Italy                      758
Channel Islands            746
Finland                    685
Cyprus                     593
Sweden                     449
Austria                    398
Denmark                    380
Poland                     330
Japan                      321
Israel                     242
Unspecified                238
Singapore                  222
Iceland                    182
USA                        179
Canada                     151
Greece                     145
Malta                      112
United Arab Emirates        68
European Community          60
RSA                         57
Lebanon                     45
Lithuania                   35
Brazil                      32
Czech Republic              25
Bahrain                     17
Saudi Arabia                 9
Name: Country, dtype: int64
```

In [33]:
```
df['Country_plot']=df['Country'][0:2000]
```

In [34]:
```
plt.figure(figsize=(16,8))
sns.countplot(x='Country_plot',data=df)
plt.show()
```



Here most of the cujstomers are belongs to Unites Kingdom

2.Cohort Analysis

For cohort analysis, we need three labels. These are payment period, cohort group and cohort period/index.To work with the time series, we need to convert the type of related feature. The format shuld be as in the dataset.

In [35]:
```
from operator import attrgetter
```

In [36]:
```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], format='%m/%d/%Y %H:%M')
```

Now, we need to create the cohort and order_month variables. The first one indicates the monthly cohort based on the first purchase date and the second one is the truncated month of the purchase date.

In [37]:
```
df['order_month'] = df['InvoiceDate'].dt.to_period('M')
```

In [38]:
```
df['cohort'] = df.groupby('CustomerID')['InvoiceDate'].transform('min').dt.to_period('M')
```

Then, we aggregate the data per cohort and order_month and count the number of unique customers in each group.

In [39]:
```
df_cohort = df.groupby(['cohort', 'order_month']).agg(n_customers=('CustomerID', 'nunique')).reset_index(drop=False)
```

In [40]:
```
df_cohort['period_number'] = (df_cohort.order_month - df_cohort.cohort).apply(attrgetter('n'))
```

In [41]:
```
df_cohort.shape
```

Out[41]:
```
(91, 4)
```

In [42]:
```
df_cohort.head()
```

Out[42]:

	cohort	order_month	n_customers	period_number
0	2010-12	2010-12	885	0
1	2010-12	2011-01	324	1
2	2010-12	2011-02	286	2
3	2010-12	2011-03	340	3
4	2010-12	2011-04	321	4

Then, we aggregate the data per cohort and order_month and count the number of unique customers in each group.

In [43]:
```
cohort_pivot = df_cohort.pivot_table(index='cohort', columns='period_number', values='n_customers')
```

In [44]:
```
cohort_pivot
```

Out[44]:

period_number	0	1	2	3	4	5	6	7	8	9	10	11	12
cohort													
2010-12	885.0	324.0	286.0	340.0	321.0	352.0	321.0	309.0	313.0	350.0	331.0	445.0	235.0
2011-01	417.0	92.0	111.0	96.0	134.0	120.0	103.0	101.0	125.0	136.0	152.0	49.0	NaN
2011-02	380.0	71.0	71.0	108.0	103.0	94.0	96.0	106.0	94.0	116.0	26.0	NaN	NaN
2011-03	452.0	68.0	114.0	90.0	101.0	76.0	121.0	104.0	126.0	39.0	NaN	NaN	NaN
2011-04	300.0	64.0	61.0	63.0	59.0	68.0	65.0	78.0	22.0	NaN	NaN	NaN	NaN
2011-05	284.0	54.0	49.0	49.0	59.0	66.0	75.0	27.0	NaN	NaN	NaN	NaN	NaN
2011-06	242.0	42.0	38.0	64.0	56.0	81.0	23.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07	188.0	34.0	39.0	42.0	51.0	21.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08	169.0	35.0	42.0	41.0	21.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09	299.0	70.0	90.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10	358.0	86.0	41.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11	323.0	36.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12	41.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

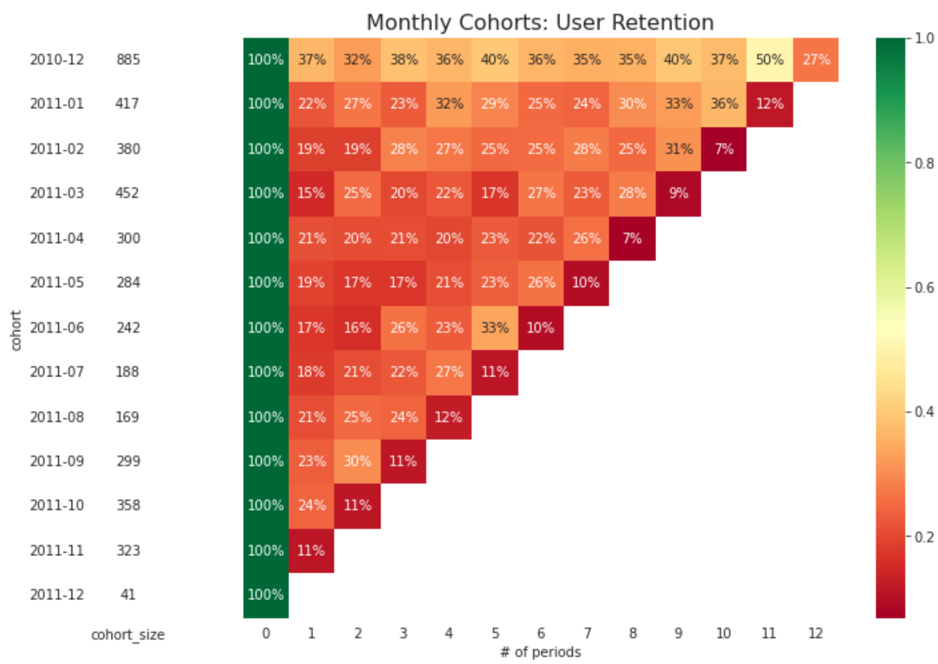
Actually, cohort_pivot shows us what we want to see. But we need to convert the table to see more clearly.

In [45]:
```
cohort_size = cohort_pivot.iloc[:, 0]
```

In [46]:
```
retention_matrix = cohort_pivot.divide(cohort_size, axis=0)
```

Lastly, we plot the retention matrix as a heatmap. Additionally, we wanted to include extra information regarding the cohort size. That is why we in fact created two heatmaps, where the one indicating the cohort size is using a white only colormap — no coloring at all.

In [47]:
```
with sns.axes_style("white"):
    fig, ax = plt.subplots(1, 2, figsize=(12, 8), sharey=True, gridspec_kw={'width_ratios': [1, 11]})

    # retention matrix
    sns.heatmap(retention_matrix,
                mask=retention_matrix.isnull(),
                annot=True,
                fmt='.0%',
                cmap='RdYlGn',
                ax=ax[1])
    ax[1].set_title('Monthly Cohorts: User Retention', fontsize=16)
    ax[1].set(xlabel='# of periods',
              ylabel='')

    # cohort size
    import matplotlib.colors as mcolors
    cohort_size_df = pd.DataFrame(cohort_size).rename(columns={0: 'cohort_size'})
    white_cmap = mcolors.ListedColormap(['white'])
    sns.heatmap(cohort_size_df,
                annot=True,
                cbar=False,
                fmt='g',
                cmap=white_cmap,
                ax=ax[0])
```



This is all about Cohort Analysis

3.a) Calculate RFM metrics.

In [48]:
```
## we first calculate Recency
```

In [49]:
```
df_recency = df.groupby(by='CustomerID',
                        as_index=False)['InvoiceDate'].max()
df_recency.columns = ['CustomerID', 'LastPurchaseDate']
recent_date = df_recency['LastPurchaseDate'].max()
df_recency['Recency'] = df_recency['LastPurchaseDate'].apply(
    lambda x: (recent_date - x).days)
df_recency.head()
```

Out[49]:

	CustomerID	LastPurchaseDate	Recency
0	12346.0	2011-01-18 10:01:00	325
1	12347.0	2011-12-07 15:52:00	1
2	12348.0	2011-09-25 13:13:00	74
3	12349.0	2011-11-21 09:51:00	18
4	12350.0	2011-02-02 16:01:00	309

In [50]:
```
## now we go for frequency
```

In [51]:
```
frequency_df = df.drop_duplicates().groupby(
    by=['CustomerID'], as_index=False)['InvoiceDate'].count()
frequency_df.columns = ['CustomerID', 'Frequency']
frequency_df.head()
```

Out[51]:

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

In [52]:
```
## and finally we will move towards monetary
```

In [53]:
```
df['Total'] = df['UnitPrice']*df['Quantity']
monetary_df = df.groupby(by='CustomerID', as_index=False)['Total'].sum()
monetary_df.columns = ['CustomerID', 'Monetary']
monetary_df.head()
```

Out[53]:

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

In [54]:
```
## now we are merging all together
```

In [55]:
```
rf_df = df_recency.merge(frequency_df, on='CustomerID')
rfm_df = rf_df.merge(monetary_df, on='CustomerID').drop(
    columns='LastPurchaseDate')
rfm_df.head()
```

Out[55]:

	CustomerID	Recency	Frequency	Monetary
0	12346.0	325	1	77183.60
1	12347.0	1	182	4310.00
2	12348.0	74	31	1797.24
3	12349.0	18	73	1757.55
4	12350.0	309	17	334.40

In [56]:
```
rfm_df['Monetary'].rank(ascending=True)
```

Out[56]:
```
0       4329.0
1       4004.0
2       3339.0
3       3314.0
4       1248.0
         ...
4333     579.0
4334     106.0
4335     561.0
4336    3449.0
4337    3366.0
Name: Monetary, Length: 4338, dtype: float64
```

3.b) Build RFM Segments.

In [57]:
```
rfm_df['R_rank'] = rfm_df['Recency'].rank(ascending=False)
rfm_df['F_rank'] = rfm_df['Frequency'].rank(ascending=True)
rfm_df['M_rank'] = rfm_df['Monetary'].rank(ascending=True)

# normalizing the rank of the customers
rfm_df['R_rank_norm'] = (rfm_df['R_rank']/rfm_df['R_rank'].max())*100
rfm_df['F_rank_norm'] = (rfm_df['F_rank']/rfm_df['F_rank'].max())*100
rfm_df['M_rank_norm'] = (rfm_df['M_rank']/rfm_df['M_rank'].max())*100

rfm_df.drop(columns=['R_rank', 'F_rank', 'M_rank'], inplace=True)

rfm_df.head()
```

Out[57]:

	CustomerID	Recency	Frequency	Monetary	R_rank_norm	F_rank_norm	M_rank_norm
0	12346.0	325	1	77183.60	3.751165	0.829876	99.792531
1	12347.0	1	182	4310.00	97.914725	88.370217	92.300599
2	12348.0	74	31	1797.24	38.513514	42.611803	76.970954
3	12349.0	18	73	1757.55	74.137931	67.358230	76.394652
4	12350.0	309	17	334.40	5.370457	25.080682	28.769018

Calculating RFM score

In [58]:
```
rfm_df['RFM_Score'] = 0.15*rfm_df['R_rank_norm']+0.28 * \
    rfm_df['F_rank_norm']+0.57*rfm_df['M_rank_norm']
rfm_df['RFM_Score'] *= 0.05
rfm_df = rfm_df.round(2)
rfm_df[['CustomerID', 'RFM_Score']].head(7)
```

Out[58]:

	CustomerID	RFM_Score
0	12346.0	2.88
1	12347.0	4.60
2	12348.0	3.08
3	12349.0	3.68
4	12350.0	1.21
5	12352.0	3.83
6	12353.0	0.27

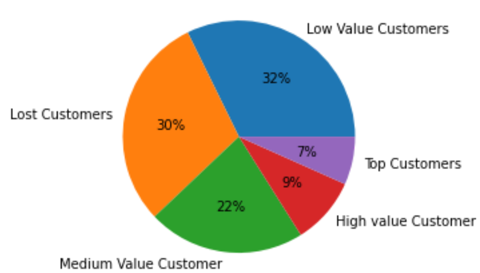
In [62]:
```
rfm_df["Customer_segment"] = np.where(rfm_df['RFM_Score'] >
                                      4.5, "Top Customers",
                                      (np.where(
                                        rfm_df['RFM_Score'] > 4,
                                        "High value Customer",
                                        (np.where(
    rfm_df['RFM_Score'] > 3,
                             "Medium Value Customer",
                             np.where(rfm_df['RFM_Score'] > 1.6,
                            'Low Value Customers', 'Lost Customers'))))))
rfm_df[['CustomerID', 'RFM_Score', 'Customer_segment']].head(20)
```

Out[62]:

	CustomerID	RFM_Score	Customer_segment
0	12346.0	2.88	Low Value Customers
1	12347.0	4.60	Top Customers
2	12348.0	3.08	Medium Value Customer
3	12349.0	3.68	Medium Value Customer
4	12350.0	1.21	Lost Customers
5	12352.0	3.83	Medium Value Customer
6	12353.0	0.27	Lost Customers
7	12354.0	2.79	Low Value Customers
8	12355.0	1.48	Lost Customers
9	12356.0	3.84	Medium Value Customer
10	12357.0	4.32	High value Customer
11	12358.0	3.02	Medium Value Customer
12	12359.0	4.36	High value Customer
13	12360.0	3.95	Medium Value Customer
14	12361.0	0.66	Lost Customers
15	12362.0	4.71	Top Customers
16	12363.0	1.93	Low Value Customers
17	12364.0	3.64	Medium Value Customer
18	12365.0	1.89	Low Value Customers
19	12367.0	1.27	Lost Customers

3.c).Analyse the RFM Segments by summarizing them

In [63]:
```
plt.pie(rfm_df.Customer_segment.value_counts(),
        labels=rfm_df.Customer_segment.value_counts().index,
        autopct='%.0f%%')
plt.show()
```



Findings- 1.There are only 16% customers are top and high value customers and rest all belongs to median and low value segment

2.Even thogh 30% customers get lost.

4.Modeling

In [64]:
```
df.head()
```

Out[64]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	box_Quantity	box_price	Country_plot	order_month	cohort	Tot
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	6.0	2.55	United Kingdom	2010-12	2010-12	15.3
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	6.0	3.39	United Kingdom	2010-12	2010-12	20.3
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	8.0	2.75	United Kingdom	2010-12	2010-12	22.0
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	6.0	3.39	United Kingdom	2010-12	2010-12	20.3
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	6.0	3.39	United Kingdom	2010-12	2010-12	20.3

In [65]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

In [66]:
```
df['Country']=le.fit_transform(df['Country'])
df['Description']=le.fit_transform(df['Description'])
df['InvoiceNo']=le.fit_transform(df['InvoiceNo'])
```

In [67]:
```
df.head()
```

Out[67]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	box_Quantity	box_price	Country_plot	order_month	cohort	Tot
0	0	85123A	3698	6	2010-12-01 08:26:00	2.55	17850.0	35	6.0	2.55	United Kingdom	2010-12	2010-12	15.3
1	0	71053	3706	6	2010-12-01 08:26:00	3.39	17850.0	35	6.0	3.39	United Kingdom	2010-12	2010-12	20.3
2	0	84406B	858	8	2010-12-01 08:26:00	2.75	17850.0	35	8.0	2.75	United Kingdom	2010-12	2010-12	22.0
3	0	84029G	1804	6	2010-12-01 08:26:00	3.39	17850.0	35	6.0	3.39	United Kingdom	2010-12	2010-12	20.3
4	0	84029E	2763	6	2010-12-01 08:26:00	3.39	17850.0	35	6.0	3.39	United Kingdom	2010-12	2010-12	20.3

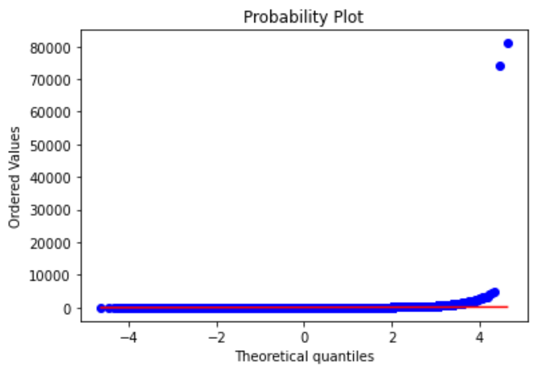
In [68]:
```
df.skew()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_6492\1665899112.py:1: FutureWarning: Dropping of nuisance columns in DataFrame redu
ctions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns befo
re calling the reduction.
  df.skew()
```

Out[68]:
```
InvoiceNo       -0.121364
Description     -0.131316
Quantity       404.976947
UnitPrice      201.536500
CustomerID       0.034254
Country         -3.026945
box_Quantity     3.740180
box_price        2.434586
Total          445.798678
dtype: float64
```

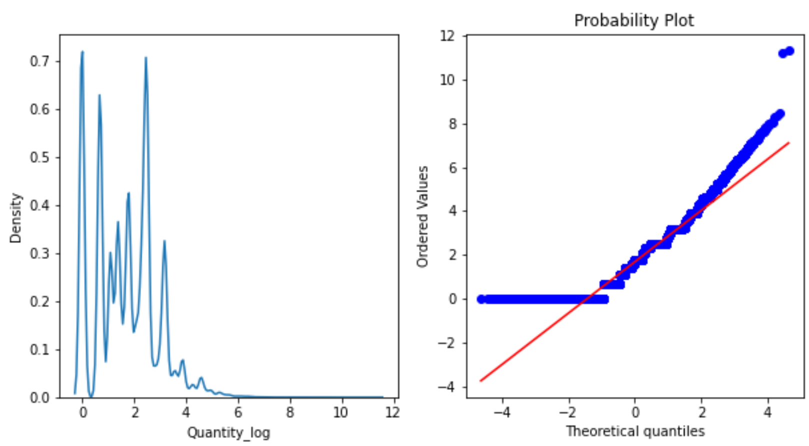
In [69]:
```
#importing necessary libraries
import scipy.stats as stats
import pylab
stats.probplot(df.Quantity,plot=pylab)
```

Out[69]:
```
((array([-4.63473223, -4.44780338, -4.34651948, ...,  4.34651948,
          4.44780338,  4.63473223]),
  array([    1,     1,     1, ...,  4800, 74215, 80995], dtype=int64)),
 (20.410068015830216, 13.244192191975415, 0.11239649091315367))
```

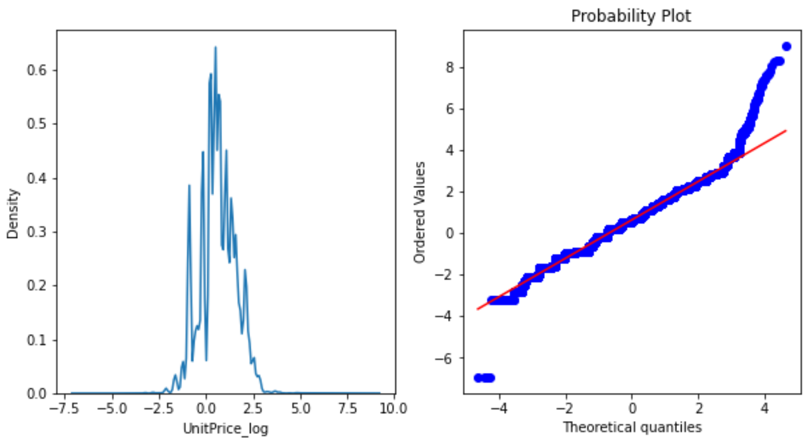


In [70]:
```
#function to return plots for the feature
def normality(df,feature):
    plt.figure(figsize=(10,5))
    plt.subplot(1,2,1)
    sns.kdeplot(df[feature])
    plt.subplot(1,2,2)
    stats.probplot(df[feature],plot=pylab)
    plt.show()
```

In [71]:
```
# A)performing logarithmic transformation on the feature
df['Quantity_log']=np.log(df['Quantity'])
normality(df,'Quantity_log')
```

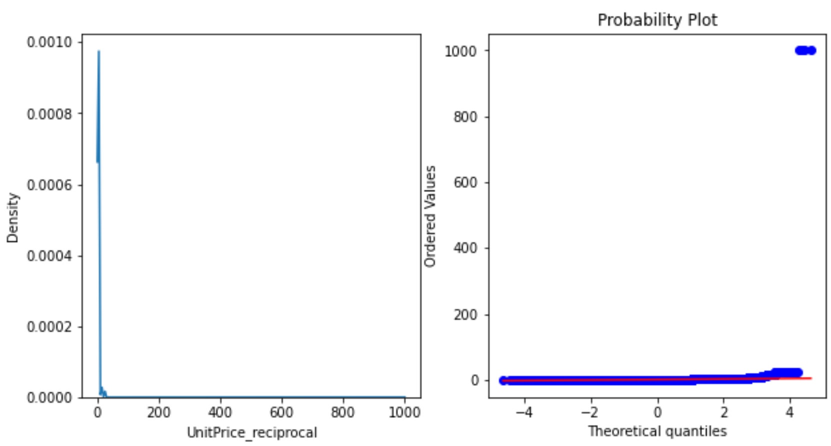


In [72]:
```
#performing Logarithmic transformation on the feature
df['UnitPrice_log']=np.log(df['UnitPrice'])
normality(df,'UnitPrice_log')
```

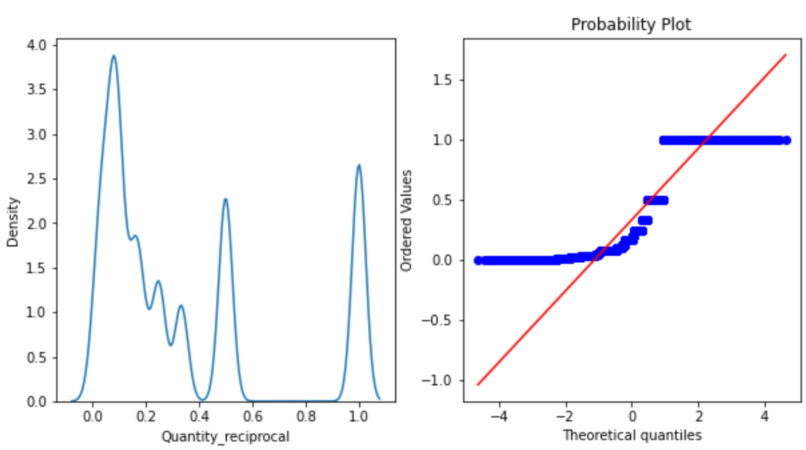


In [73]:
```
## B) Reciprocal Method
```

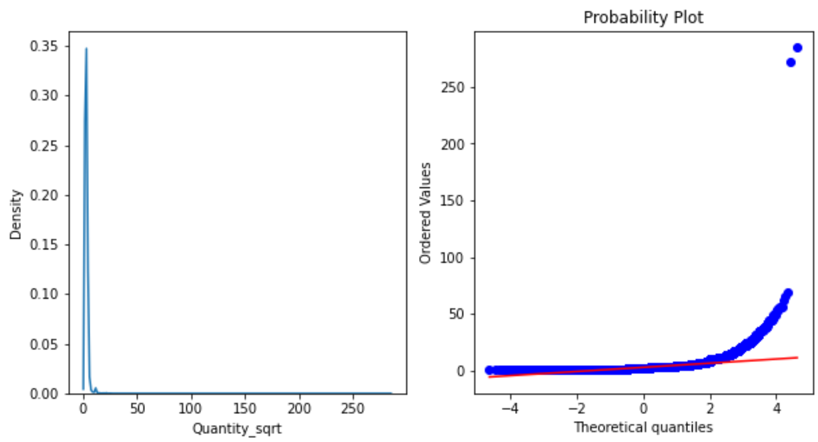
In [74]:
```
df['UnitPrice_reciprocal']=1/df['UnitPrice']
normality(df,'UnitPrice_reciprocal')
```



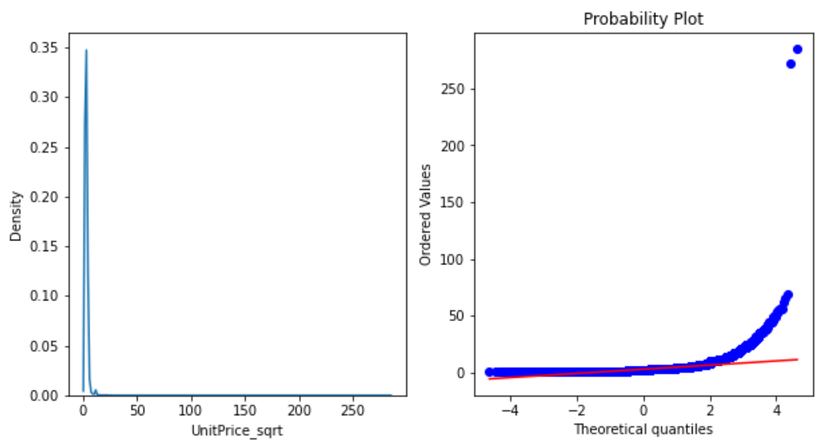
In [75]:
```
df['Quantity_reciprocal']=1/df['Quantity']
normality(df,'Quantity_reciprocal')
```



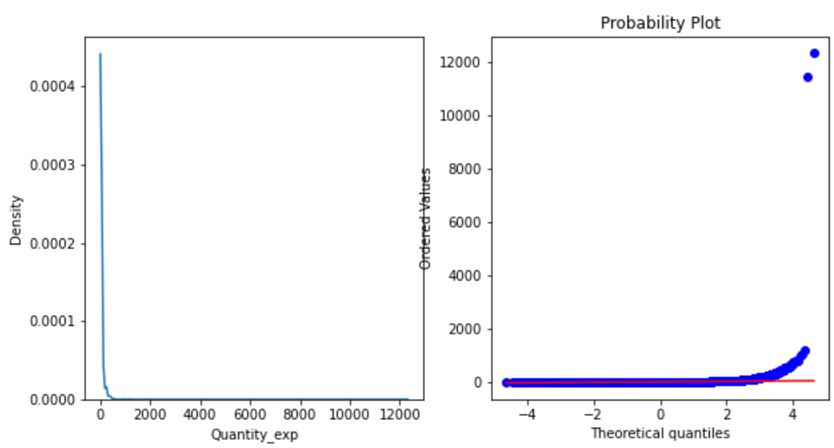
In [76]:
```
df['Quantity_sqrt']=np.sqrt(df.Quantity)
normality(df,'Quantity_sqrt')
```



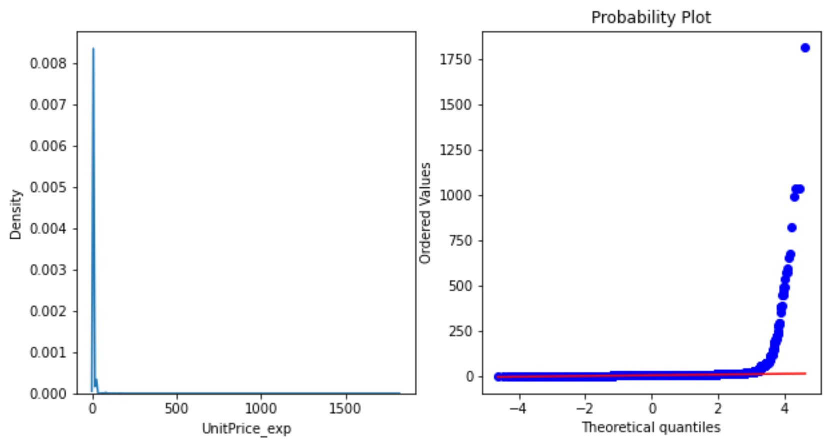
In [77]:
```
df['UnitPrice_sqrt']=np.sqrt(df.Quantity)
normality(df,'UnitPrice_sqrt')
```



In [78]:
```
df['Quantity_exp']=df.Quantity**(1/1.2)
normality(df,'Quantity_exp')
```



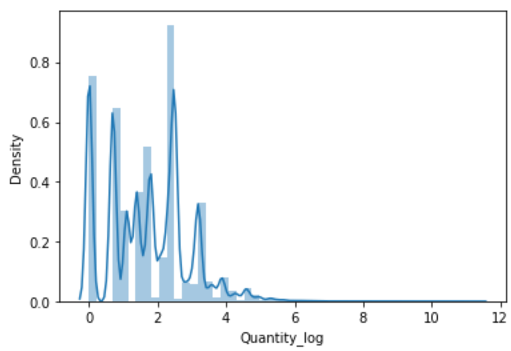
In [79]:
```
df['UnitPrice_exp']=df.UnitPrice**(1/1.2)
normality(df,'UnitPrice_exp')
```



In [80]:
```
sns.distplot(df['Quantity_log'])
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

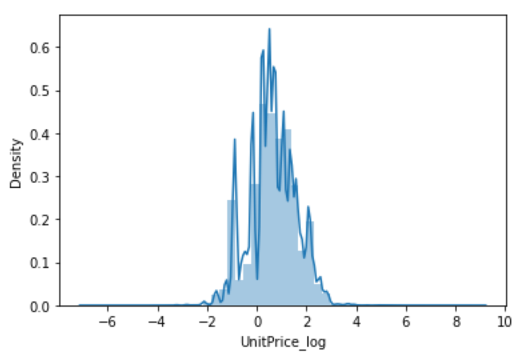
Out[80]: <AxesSubplot:xlabel='Quantity_log', ylabel='Density'>



In [81]:
```
sns.distplot(df['UnitPrice_log'])
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

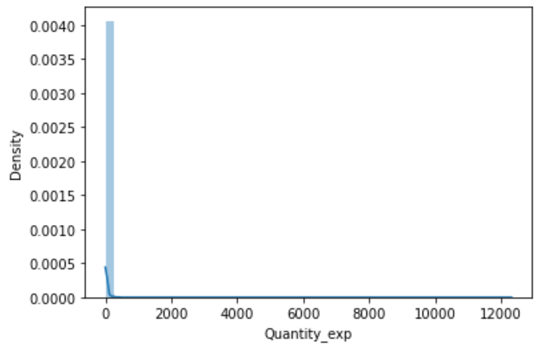
Out[81]: <AxesSubplot:xlabel='UnitPrice_log', ylabel='Density'>



In [82]:
```
sns.distplot(df['Quantity_exp'])
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

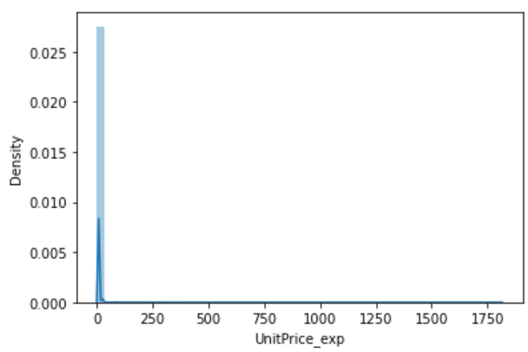
Out[82]: <AxesSubplot:xlabel='Quantity_exp', ylabel='Density'>



In [83]:
```
sns.distplot(df['UnitPrice_exp'])
```

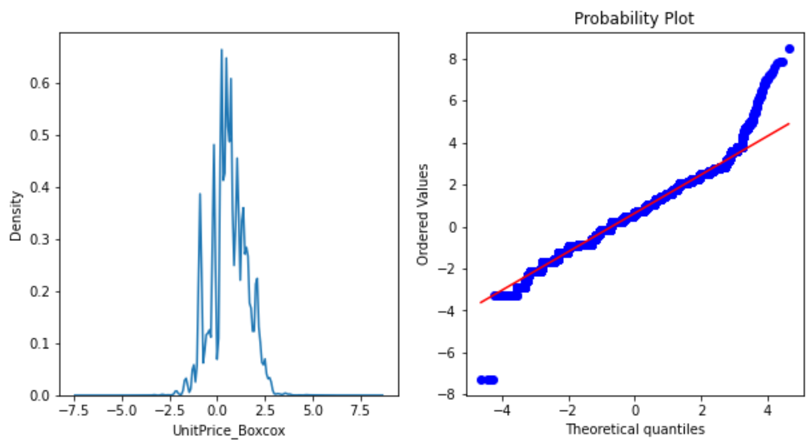
```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[83]: <AxesSubplot:xlabel='UnitPrice_exp', ylabel='Density'>

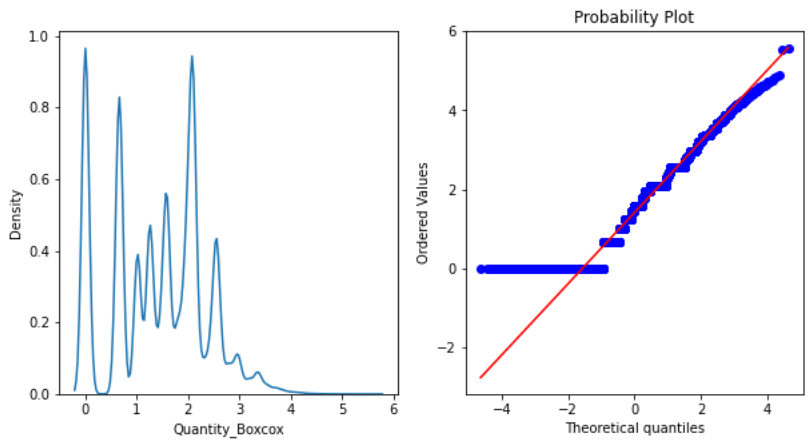


In [84]:
```
## 5) Box Cox Transformation
```

In [85]:
```
df['UnitPrice_Boxcox'],parameters=stats.boxcox(df['UnitPrice'])
normality(df,'UnitPrice_Boxcox')
```



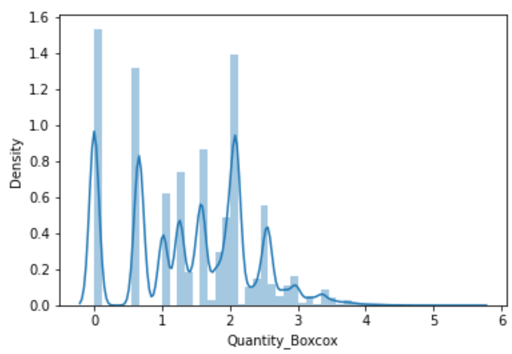
In [86]:
```
df['Quantity_Boxcox'],parameters=stats.boxcox(df['Quantity'])
normality(df,'Quantity_Boxcox')
```



In [87]:
```
sns.distplot(df['Quantity_Boxcox'])
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

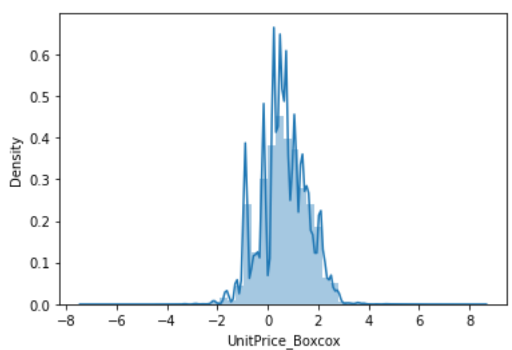
Out[87]: <AxesSubplot:xlabel='Quantity_Boxcox', ylabel='Density'>



In [88]:
```
sns.distplot(df['UnitPrice_Boxcox'])
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

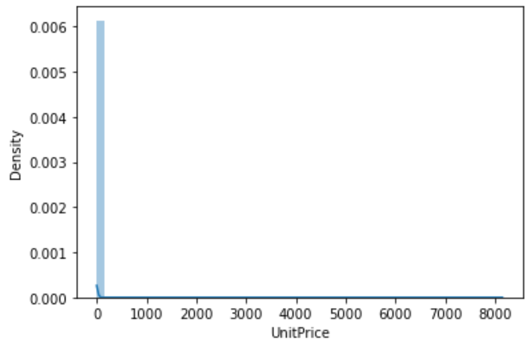
Out[88]: <AxesSubplot:xlabel='UnitPrice_Boxcox', ylabel='Density'>



In [89]:
```
sns.distplot(df['UnitPrice'])
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

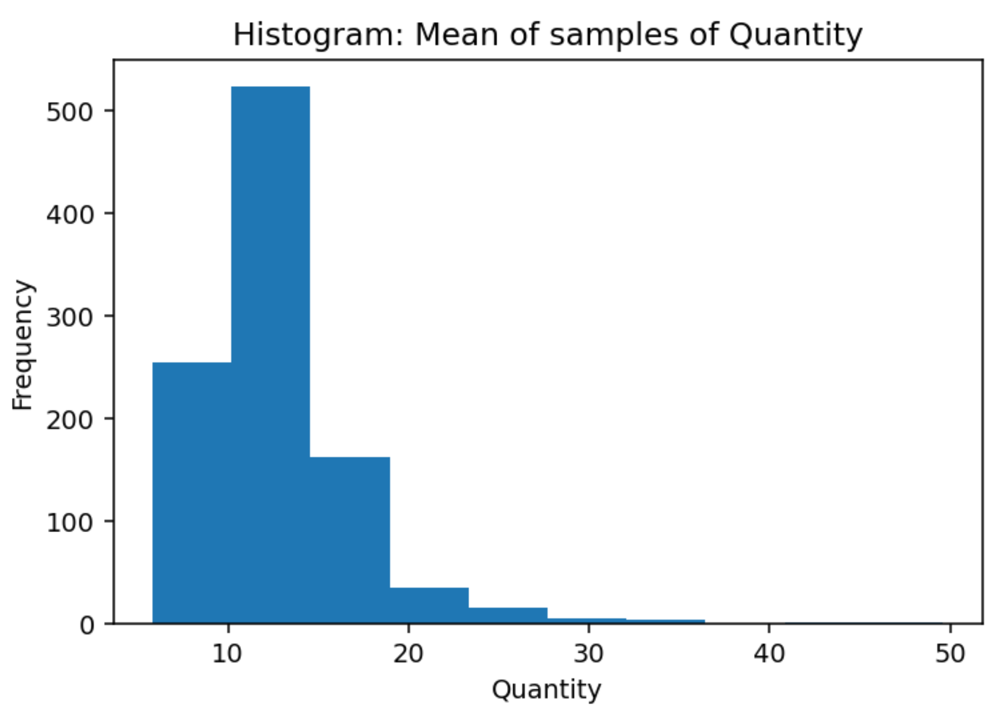
Out[89]:
```
<AxesSubplot:xlabel='UnitPrice', ylabel='Density'>
```



And, the variables with -0.5 < skewness < 0.5 are symmetric i.e normally distributed such as InvoiceNO,description,CustomerID are normally distributed

In [ ]:

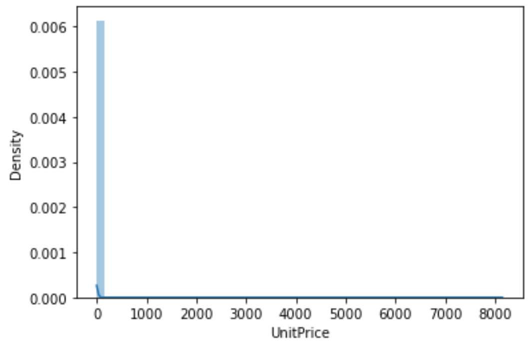
In [90]:
```
population=df['Quantity']
#Create a list
sampled_means = []
# For 1000 times:
for i in range(1000):
    # Take a random sample of 100 rows from the popoulation, take the mean of these rows,append to sampled_means
    sampled_means.append(population.sample(100).mean())
# plotting histogram
plt.figure(dpi = 140) #resolution of the figure
plt.hist(sampled_means)
plt.xlabel('Quantity')
plt.ylabel('Frequency')
plt.title("Histogram: Mean of samples of Quantity")
plt.show()
```



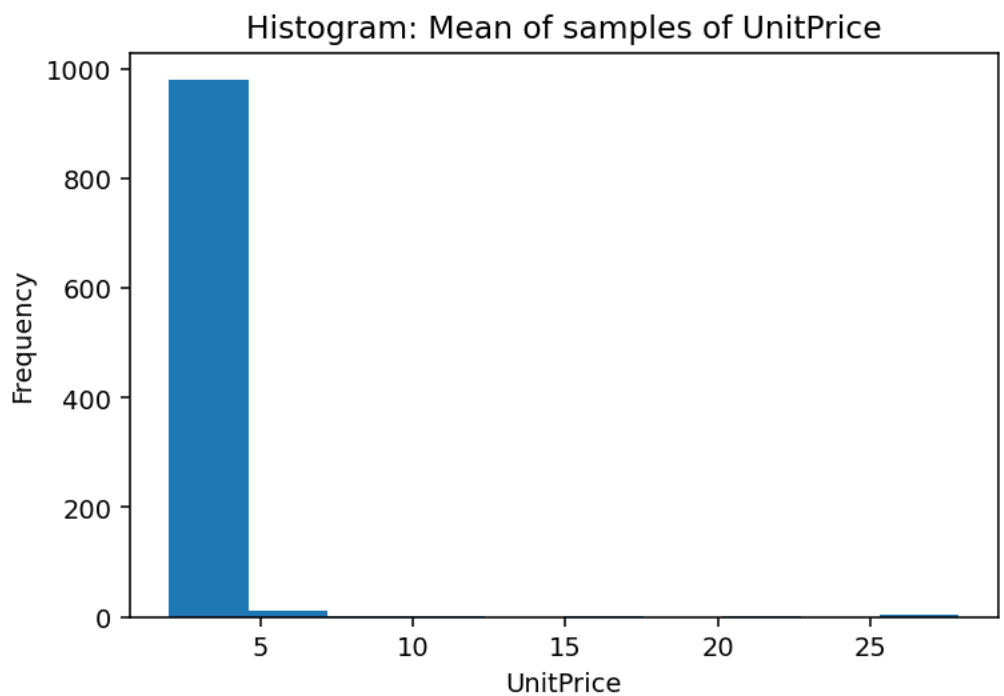
In [91]:
```
sns.distplot(df['UnitPrice'])
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fle
xibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[91]:
```
<AxesSubplot:xlabel='UnitPrice', ylabel='Density'>
```

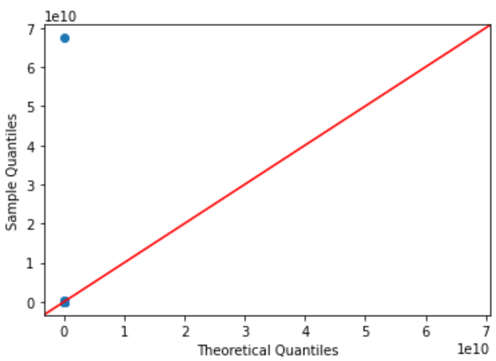


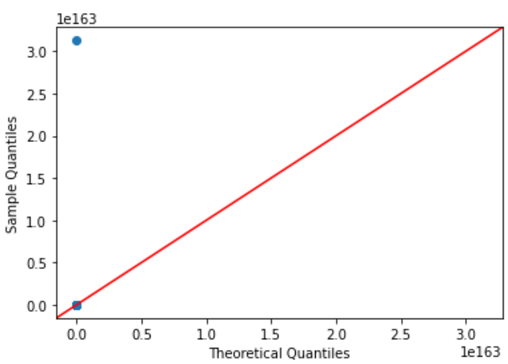
In [92]:
```
pop=df['UnitPrice']
#Create a list
sampled_means = []
# For 1000 times:
for i in range(1000):
    # Take a random sample of 100 rows from the popoulation, take the mean of these rows,append to sampled_means
    sampled_means.append(pop.sample(100).mean())
# plotting histogram
plt.figure(dpi = 140) #resolution of the figure
plt.hist(sampled_means)
plt.xlabel('UnitPrice')
plt.ylabel('Frequency')
plt.title("Histogram: Mean of samples of UnitPrice")
plt.show()
```



we will try by another method

In [93]:
```
import math
import numpy as np
from scipy.stats import lognorm
import statsmodels.api as sm
import matplotlib.pyplot as plt

#make this example reproducible
np.random.seed(1)

#generate dataset that contains 1000 log-normal distributed values
lognorm_dataset = lognorm.rvs(df['UnitPrice'][:1000])

#create Q-Q plot with 45-degree line added to plot
fig = sm.qqplot(lognorm_dataset, line='45')

plt.show()


#make this example reproducible
np.random.seed(1)

#generate dataset that contains 1000 log-normal distributed values
lognorm_dataset = lognorm.rvs(df['Quantity'][:1000])

#create Q-Q plot with 45-degree line added to plot
fig = sm.qqplot(lognorm_dataset, line='45')

plt.show()
```





In [94]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

In [95]:
```
df.head()
```

Out[95]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	box_Quantity	box_price	...	Quantity_log	UnitPrice_log	UnitPric
0	0	85123A	3698	6	2010-12-01 08:26:00	2.55	17850.0	35	6.0	2.55	...	1.791759	0.936093	
1	0	71053	3706	6	2010-12-01 08:26:00	3.39	17850.0	35	6.0	3.39	...	1.791759	1.220830	
2	0	84406B	858	8	2010-12-01 08:26:00	2.75	17850.0	35	8.0	2.75	...	2.079442	1.011601	
3	0	84029G	1804	6	2010-12-01 08:26:00	3.39	17850.0	35	6.0	3.39	...	1.791759	1.220830	
4	0	84029E	2763	6	2010-12-01 08:26:00	3.39	17850.0	35	6.0	3.39	...	1.791759	1.220830	

5 rows × 24 columns

In [ ]:
```
X=df.iloc[:, [3,5]][0:20000].values
```

In [ ]:
```
X.size
```

In [ ]:
```
scaled_data=sc.fit_transform(X)
```

In [ ]:
```
plt.scatter(df['Quantity'],df['UnitPrice'])
plt.xlabel('mean_dist_day')
plt.ylabel('mean_over_speed_perc')
```

Here we have applied standard scaler and we got scaled data now,hence we are going to apply k-means clustering now.

In [161]:
```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(scaled_data)
y_predicted
```

Out[161]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [162]:
```
cluster=np.array(y_predicted)
```

In [163]:
```
df.head()
```

Out[163]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	box_Quantity	box_price	...	UnitPrice_log	UnitPrice_reciprocal	(
0	0	85123A	3698	6	2010-12-01 08:26:00	2.55	17850.0	35	6.0	2.55	...	0.936093	0.392157	
1	0	71053	3706	6	2010-12-01 08:26:00	3.39	17850.0	35	6.0	3.39	...	1.220830	0.294985	
2	0	84406B	858	8	2010-12-01 08:26:00	2.75	17850.0	35	8.0	2.75	...	1.011601	0.363636	
3	0	84029G	1804	6	2010-12-01 08:26:00	3.39	17850.0	35	6.0	3.39	...	1.220830	0.294985	
4	0	84029E	2763	6	2010-12-01 08:26:00	3.39	17850.0	35	6.0	3.39	...	1.220830	0.294985	

5 rows × 25 columns

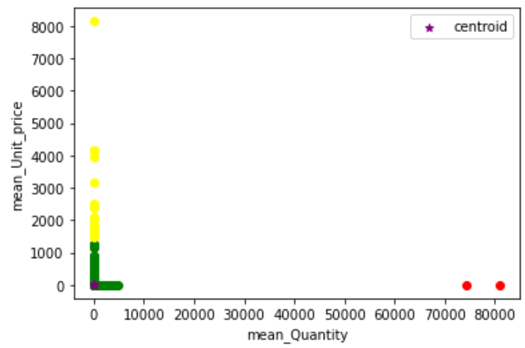
In [164]:
```
km.cluster_centers_
```

Out[164]:
```
array([[-1.96880937e-02, -1.55286530e-02],
       [ 3.04219894e+01, -4.13484996e-01],
       [-2.17681469e-01,  3.15591205e+01]])
```

Plot the custer centroid

In [165]:
```
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]

plt.scatter(df1['Quantity'],df1['UnitPrice'],color='green')
plt.scatter(df2['Quantity'],df2['UnitPrice'],color='red')
plt.scatter(df3['Quantity'],df3['UnitPrice'],color='yellow')

plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.xlabel('mean_Quantity')
plt.ylabel('mean_Unit_price')
plt.legend()
```

Out[165]: <matplotlib.legend.Legend at 0x1edf9265fd0>



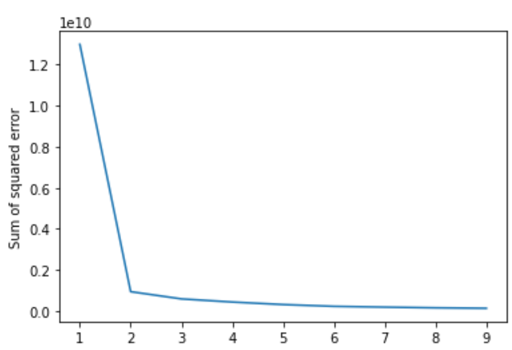
In [166]:
```
sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['Quantity','UnitPrice']])
    sse.append(km.inertia_)
```

In [167]:
```
sse
```

Out[167]:
```
[12984316701.835003,
 943293472.3465149,
 590481552.0372462,
 437954291.5964886,
 315266328.0075663,
 228491288.45711443,
 190999663.49099034,
 156276990.02616012,
 133243616.47008108]
```

In [168]:
```
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

Out[168]: [<matplotlib.lines.Line2D at 0x1edfc15ffa0>]



In [170]:
```
#conda install -c conda-forge kneed
```

In [173]:
```
### we found the above plot get abrupt change at 2 so,we can say there must be 2 cluster we have to follow
```

In [179]:
```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=2)
y_predict = km.fit_predict(scaled_data)
y_predict
```

Out[179]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

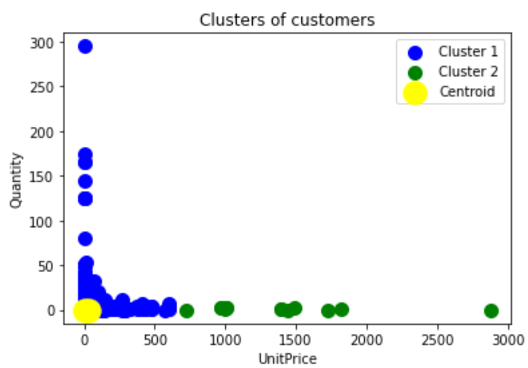
In [180]:
```
cluster=np.array(y_predict)
```

In [181]:
```
## we will visualize the clusters
```

In [182]:
```
km.cluster_centers_
```

Out[182]:
```
array([[-1.97871548e-02,  2.68940059e-04],
       [ 3.04219894e+01, -4.13484996e-01]])
```

In [183]:
```
plt.scatter(X[y_predict == 0, 0], X[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster
plt.scatter(X[y_predict == 1, 0], X[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster

plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')
plt.title('Clusters of customers')
plt.xlabel('UnitPrice')
plt.ylabel('Quantity')
plt.legend()
plt.show()
```



Cluster1 shows the product with lower unit price and less number of quantity sold out.

Cluster2 shows the product has a average Unit price but less number of quantity sold out.

Agglomerative Clustering

In [ ]:

In [ ]:

In [ ]: