Data_thon_Assignment

March 24, 2022

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

1 Data thon data

[ ]:
```
# Loading the Data set
df = pd.read_csv("Train.csv")
df.head()
```

[ ]:

	origin	destination	airline	refundable	baggage_weight	baggage_pieces	flight_number	purchace_date	departure_date	arival_date	departure_time	arival_time	purchase_time
0	x	y	gamma	1	0.000000	0	c-2	2020-12-31	2021-01-10	2021-01-10	05:00:00	07:00:00	09:46:17
1	x	y	gamma	1	0.711111	1	c-2	2020-12-31	2021-01-10	2021-01-10	05:00:00	07:00:00	09:46:17
2	x	y	gamma	1	0.711111	1	c-4	2020-12-31	2021-01-10	2021-01-10	11:00:00	13:00:00	09:46:17
3	x	y	gamma	1	0.711111	2	c-4	2020-12-31	2021-01-10	2021-01-10	11:00:00	13:00:00	09:46:17
4	x	y	beta	0	0.444444	0	b-69	2020-12-31	2021-01-25	2021-01-25	11:00:00	12:55:00	09:46:18

1.1 Replace the strings to number of airline type

[ ]:
```
df = df.replace('gamma',0)
df = df.replace('beta',1)
df = df.replace('alpha',2)
df = df.replace('omega',3)
```

[ ]:
```
# loading the target values just to make it easy for interpretation by adding␣
 ↪along in same dataframe
y = pd.read_csv('y_train.csv')
y.head()
```

[ ]:
```
   Unnamed: 0   target
0           0   7400.0
1           1   8650.0
2           2   9150.0
3           3  10400.0
4           4   8697.0
```

[ ]:
```
# Droping the unnamed coloumn/index
y = y.drop("Unnamed: 0",axis=1)
```

[ ]:
```
# Adding the target coloumn to the taining data frame
df['price'] = y
```

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

[ ]:
```
  origin destination  airline  refundable  baggage_weight  baggage_pieces  \
0      x           y        0           1        0.000000               0
1      x           y        0           1        0.711111               1
2      x           y        0           1        0.711111               1
3      x           y        0           1        0.711111               2
4      x           y        1           0        0.444444               0

  flight_number purchace_date departure_date arival_date departure_time  \
0           c-2    2020-12-31     2021-01-10  2021-01-10       05:00:00
1           c-2    2020-12-31     2021-01-10  2021-01-10       05:00:00
2           c-4    2020-12-31     2021-01-10  2021-01-10       11:00:00
3           c-4    2020-12-31     2021-01-10  2021-01-10       11:00:00
4          b-69    2020-12-31     2021-01-25  2021-01-25       11:00:00

  arival_time purchase_time    price
0    07:00:00      09:46:17   7400.0
1    07:00:00      09:46:17   8650.0
2    13:00:00      09:46:17   9150.0
3    13:00:00      09:46:17  10400.0
4    12:55:00      09:46:18   8697.0
```

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

[ ]:
```
          origin destination  airline  refundable  baggage_weight  \
21776585      x           y        2           1        0.444444
21776586      x           y        2           1        0.333333
21776587      x           y        3           1        0.444444
21776588      x           y        3           1        0.444444
21776589      x           y        3           1        0.444444

          baggage_pieces flight_number purchace_date departure_date  \
21776585               1           a-7    2021-08-31     2021-09-03
21776586               1           a-9    2021-08-31     2021-09-03
21776587               1           d-1    2021-08-31     2021-09-03
21776588               1           d-3    2021-08-31     2021-09-03
21776589               1           d-5    2021-08-31     2021-09-03

          arival_date departure_time arival_time purchase_time   price
21776585  2021-09-03       10:00:00    12:00:00      23:29:18  8381.0
21776586  2021-09-03       13:40:00    15:40:00      23:29:18  9045.0
21776587  2021-09-03       04:40:00    06:40:00      23:29:18  6155.0
21776588  2021-09-03       10:35:00    12:35:00      23:29:18  6155.0
21776589  2021-09-03       17:05:00    19:05:00      23:29:18  6605.0
```

[ ]:
```
#Converting the data type from object to datetime
df['purchace_date'] = pd.to_datetime(df['purchace_date'])
df.head()
```

[ ]:
```
  origin destination  airline  refundable  baggage_weight  baggage_pieces  \
0      x           y        0           1        0.000000               0
1      x           y        0           1        0.711111               1
2      x           y        0           1        0.711111               1
3      x           y        0           1        0.711111               2
4      x           y        1           0        0.444444               0

  flight_number purchace_date departure_date arival_date departure_time  \
0           c-2    2020-12-31     2021-01-10  2021-01-10       05:00:00
1           c-2    2020-12-31     2021-01-10  2021-01-10       05:00:00
2           c-4    2020-12-31     2021-01-10  2021-01-10       11:00:00
3           c-4    2020-12-31     2021-01-10  2021-01-10       11:00:00
4          b-69    2020-12-31     2021-01-25  2021-01-25       11:00:00

  arival_time purchase_time    price
0    07:00:00      09:46:17   7400.0
1    07:00:00      09:46:17   8650.0
2    13:00:00      09:46:17   9150.0
3    13:00:00      09:46:17  10400.0
4    12:55:00      09:46:18   8697.0
```

[ ]:
```
#Converting the data type from object to datetime
df['departure_date'] = pd.to_datetime(df['departure_date'])
df.head()
```

[ ]:
```
  origin destination  airline  refundable  baggage_weight  baggage_pieces  \
0      x           y        0           1        0.000000               0
1      x           y        0           1        0.711111               1
2      x           y        0           1        0.711111               1
3      x           y        0           1        0.711111               2
4      x           y        1           0        0.444444               0

  flight_number purchace_date departure_date arival_date departure_time  \
0           c-2    2020-12-31     2021-01-10  2021-01-10       05:00:00
1           c-2    2020-12-31     2021-01-10  2021-01-10       05:00:00
2           c-4    2020-12-31     2021-01-10  2021-01-10       11:00:00
3           c-4    2020-12-31     2021-01-10  2021-01-10       11:00:00
4          b-69    2020-12-31     2021-01-25  2021-01-25       11:00:00

  arival_time purchase_time    price
0    07:00:00      09:46:17   7400.0
1    07:00:00      09:46:17   8650.0
2    13:00:00      09:46:17   9150.0
3    13:00:00      09:46:17  10400.0
4    12:55:00      09:46:18   8697.0
```

[ ]:
```
# To find the number of days the user bought the ticket before departure
a = df['departure_date']-df['purchace_date']
a
```

[ ]:
```
0          10 days
1          10 days
2          10 days
3          10 days
4          25 days
             ...
21776585    3 days
21776586    3 days
21776587    3 days
21776588    3 days
21776589    3 days
Length: 21776590, dtype: timedelta64[ns]
```

[ ]:
```
# To add no of days to the data frame for further prediction
s = a.dt.days
s
```

[ ]:
```
0           10
1           10
2           10
3           10
4           25
            ..
21776585     3
21776586     3
21776587     3
21776588     3
21776589     3
Length: 21776590, dtype: int64
```

[ ]:
```
# Appending the data frame with new coloumn
df['purchase_days_before_daprture'] = s
df.head(6)
```

[ ]:
```
  origin destination  airline  refundable  baggage_weight  baggage_pieces  \
0      x           y        0           1        0.000000               0
1      x           y        0           1        0.711111               1
2      x           y        0           1        0.711111               1
3      x           y        0           1        0.711111               2
4      x           y        1           0        0.444444               0
5      x           y        1           0        0.444444               0

  flight_number purchace_date departure_date arival_date departure_time  \
0           c-2    2020-12-31     2021-01-10  2021-01-10       05:00:00
1           c-2    2020-12-31     2021-01-10  2021-01-10       05:00:00
2           c-4    2020-12-31     2021-01-10  2021-01-10       11:00:00
3           c-4    2020-12-31     2021-01-10  2021-01-10       11:00:00
4          b-69    2020-12-31     2021-01-25  2021-01-25       11:00:00
5           b-1    2020-12-31     2021-01-25  2021-01-25       05:00:00

  arival_time purchase_time    price  purchase_days_before_daprture
0    07:00:00      09:46:17   7400.0                             10
1    07:00:00      09:46:17   8650.0                             10
2    13:00:00      09:46:17   9150.0                             10
3    13:00:00      09:46:17  10400.0                             10
4    12:55:00      09:46:18   8697.0                             25
5    06:55:00      09:46:18   8697.0                             25
```

[ ]:
```
# Just for fun Type conversion
d= pd.to_datetime(df['departure_time'])
d
```

[ ]:
```
0          2022-03-24 05:00:00
1          2022-03-24 05:00:00
```

```
2          2022-03-24 11:00:00
3          2022-03-24 11:00:00
4          2022-03-24 11:00:00
                   …
21776585   2022-03-24 10:00:00
21776586   2022-03-24 13:40:00
21776587   2022-03-24 04:40:00
21776588   2022-03-24 10:35:00
21776589   2022-03-24 17:05:00
Name: departure_time, Length: 21776590, dtype: datetime64[ns]
```

[ ]:
```
# Just for fun Type conversion
e= pd.to_datetime(df['arival_time'])
e
```

[ ]:
```
0          2022-03-24 07:00:00
1          2022-03-24 07:00:00
2          2022-03-24 13:00:00
3          2022-03-24 13:00:00
4          2022-03-24 12:55:00
                   …
21776585   2022-03-24 12:00:00
21776586   2022-03-24 15:40:00
21776587   2022-03-24 06:40:00
21776588   2022-03-24 12:35:00
21776589   2022-03-24 19:05:00
Name: arival_time, Length: 21776590, dtype: datetime64[ns]
```

[ ]:
```
# To find the Journey duration its just for practice
f = e-d
f
```

[ ]:
```
0          0 days 02:00:00
1          0 days 02:00:00
2          0 days 02:00:00
3          0 days 02:00:00
4          0 days 01:55:00
                   …
21776585   0 days 02:00:00
21776586   0 days 02:00:00
21776587   0 days 02:00:00
21776588   0 days 02:00:00
21776589   0 days 02:00:00
Length: 21776590, dtype: timedelta64[ns]
```

2 Yahan sy nechy kal k Assignmnet hai Ammar bhai

[ ]:
```
thon=␣
 ↪df[['purchase_days_before_daprture','airline','baggage_weight','baggage_pieces','price']]
thon.head()
```

[ ]:
```
   purchase_days_before_daprture  airline  baggage_weight  baggage_pieces  \
0                             10        0        0.000000               0
1                             10        0        0.711111               1
2                             10        0        0.711111               1
3                             10        0        0.711111               2
4                             25        1        0.444444               0

     price
0   7400.0
1   8650.0
2   9150.0
3  10400.0
4   8697.0
```

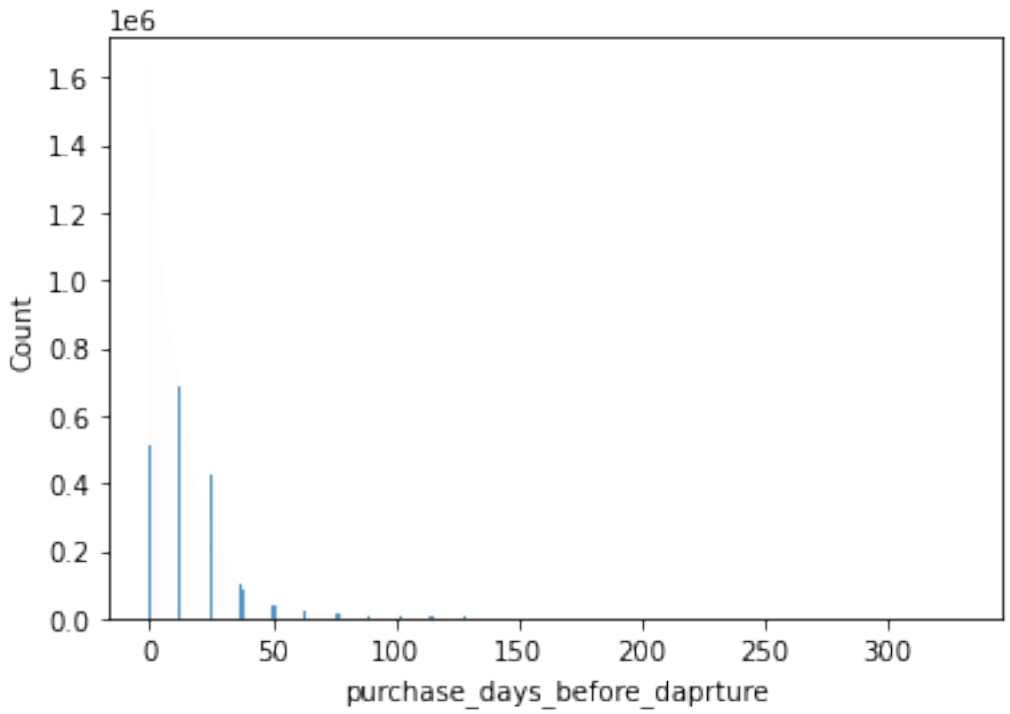
[ ]:
```
thon.tail()
```

[ ]:
```
          purchase_days_before_daprture  airline  baggage_weight  \
21776585                              3        2        0.444444
21776586                              3        2        0.333333
21776587                              3        3        0.444444
21776588                              3        3        0.444444
21776589                              3        3        0.444444

          baggage_pieces   price
21776585               1  8381.0
21776586               1  9045.0
21776587               1  6155.0
21776588               1  6155.0
21776589               1  6605.0
```

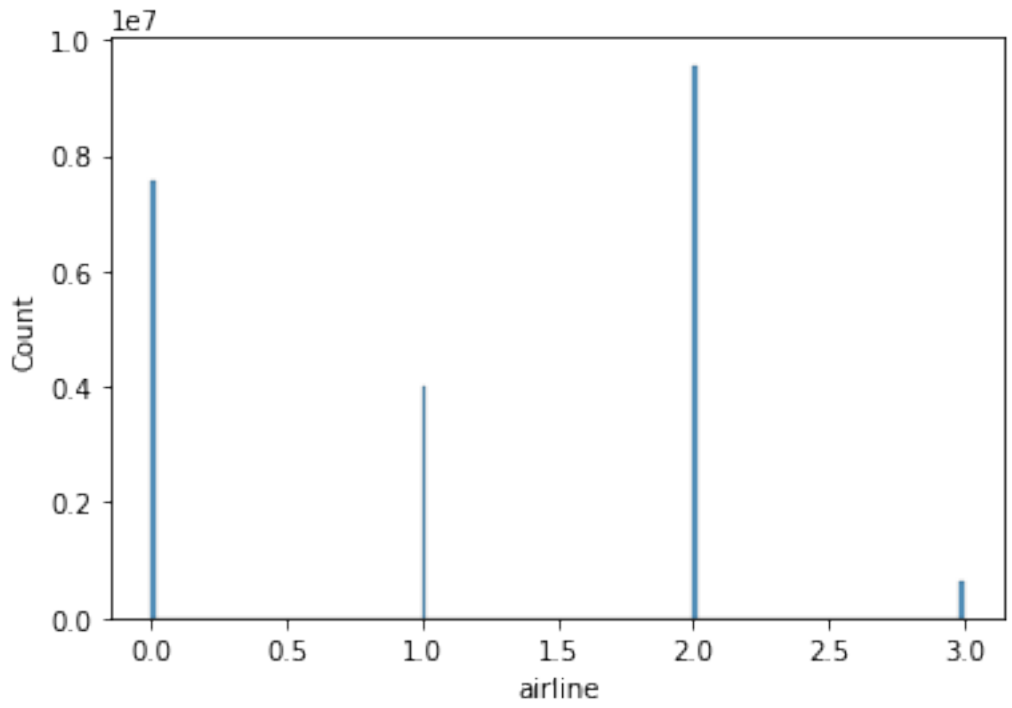
[ ]:
```
# Histogram test
sns.histplot(thon['purchase_days_before_daprture'])
```

[ ]:
```
<AxesSubplot:xlabel='purchase_days_before_daprture', ylabel='Count'>
```



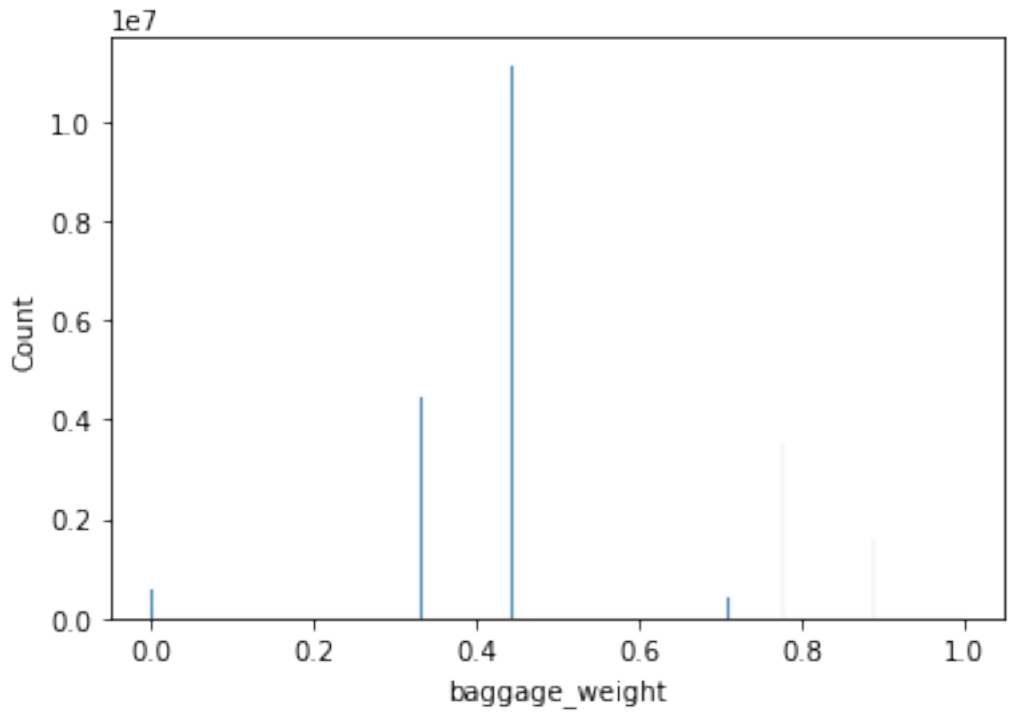
[ ]:
```
# Histogram test
sns.histplot(thon['airline'])
```

[ ]:
```
<AxesSubplot:xlabel='airline', ylabel='Count'>
```



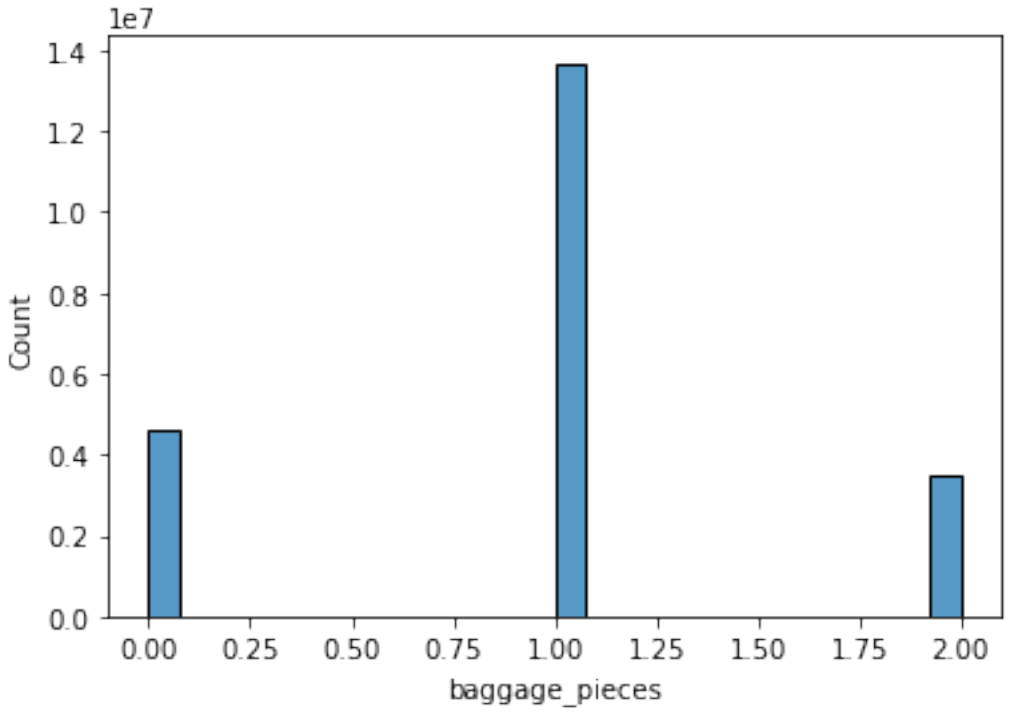
[ ]:
```
# Histogram test
sns.histplot(thon['baggage_weight'])
```

[ ]:
```
<AxesSubplot:xlabel='baggage_weight', ylabel='Count'>
```



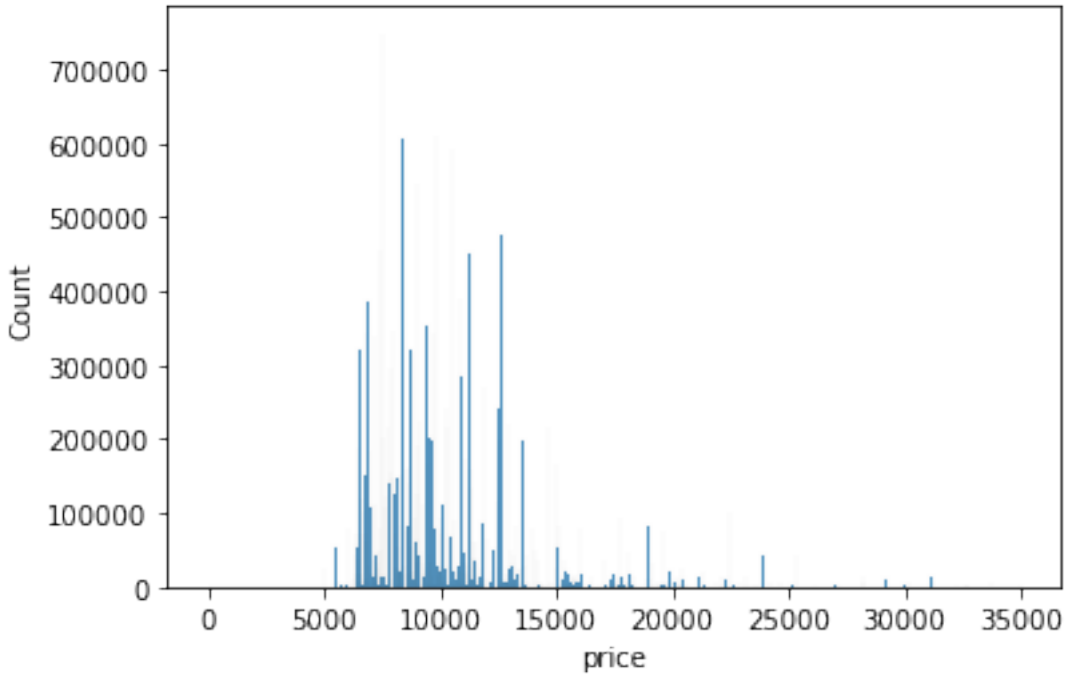
[ ]:
```
# Histogram test
sns.histplot(thon['baggage_pieces'])
```

[ ]:
```
<AxesSubplot:xlabel='baggage_pieces', ylabel='Count'>
```



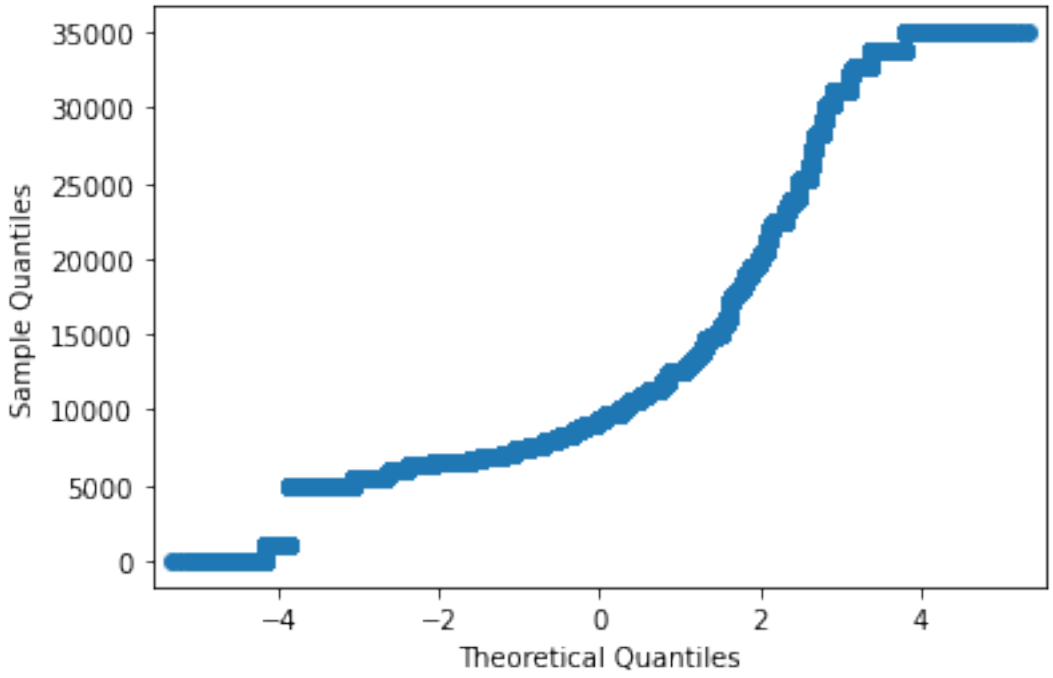
[ ]:
```
# Histogram test
sns.histplot(thon['price'])
```

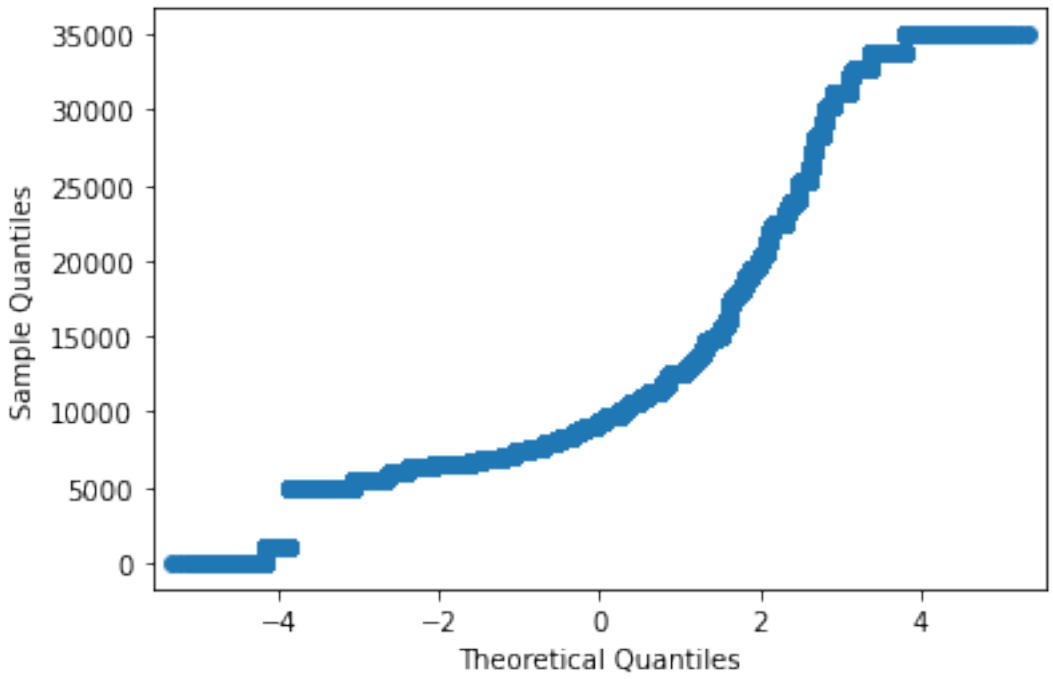
[ ]:
```
<AxesSubplot:xlabel='price', ylabel='Count'>
```



[ ]:
```
from statsmodels.graphics.gofplots import qqplot
qqplot(thon['price'])
```

[ ]:

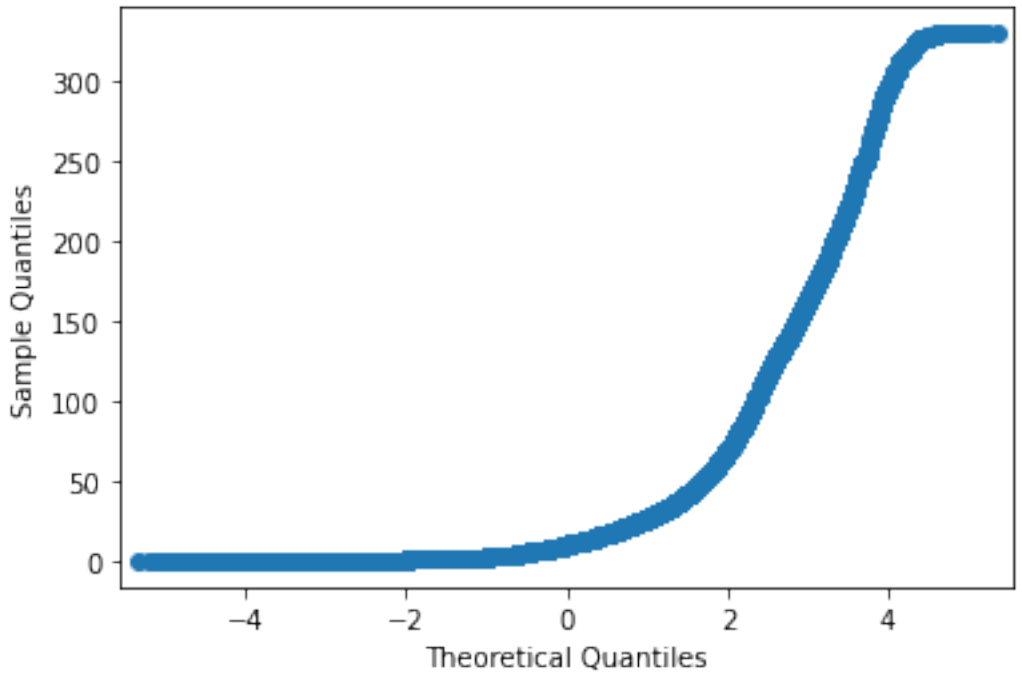


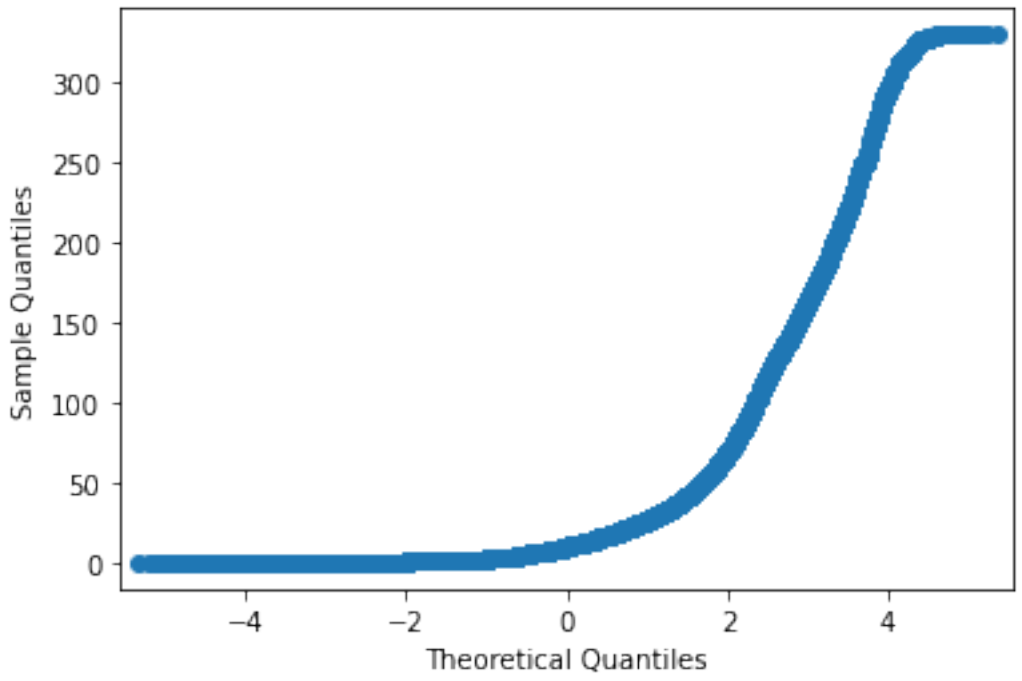


3 Ammar bhai is me agr hm first kuch rows ko remove krain to ya normal hojya ga price ?

[ ]:
```
qqplot(thon['purchase_days_before_daprture'])
```

[ ]:





4 Is say ek or chezz k pata chala k agr hmri number of values > 5000 ho to maybe possible hai k p-value accurate na aya.

[ ]:
```
# Shapiro wilk test
from scipy.stats import shapiro
stats , p_value = shapiro(thon['price'])
print("Stats = ",stats)
print("P_value = ", p_value)
if p_value > 0.5:
    print("Data is probabily normal or Guassian")
else:
    print("Data is probabily not Guassian")
```

```
Stats =  0.7644489407539368
P_value =  0.0
Data is probabily not Guassian

C:\Users\Sartaj\anaconda3\lib\site-packages\scipy\stats\morestats.py:1760:
UserWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

5 Answer of the question asked by Ammar bhai When to use D Agostino's test is :

If the data contains repeated values it is recommended to use Agostino's test otherwise mostly used test shapiro wilk test is recommended.

[ ]:
```
# D Agostino's K-squared test
from scipy.stats import normaltest
stats , p_value = normaltest(thon['price'])
print("Stats = ",stats)
print("P_value = ", p_value)
if p_value > 0.5:
    print("Data is probabily normal or Guassian")
else:
    print("Data is probabily not Guassian")
```

```
Stats =  10071130.773025015
P_value =  0.0
Data is probabily not Guassian
```

[ ]:
```
# Shapiro wilk test for purchase_days_before_daprture
from scipy.stats import shapiro
stats , p_value = shapiro(thon['purchase_days_before_daprture'])
print("Stats = ",stats)
print("P_value = ", p_value)
if p_value > 0.5:
    print("Data is probabily normal or Guassian")
else:
    print("Data is probabily not Guassian")
```

```
Stats =  0.6838458180427551
P_value =  0.0
Data is probabily not Guassian

C:\Users\Sartaj\anaconda3\lib\site-packages\scipy\stats\morestats.py:1760:
UserWarning: p-value may not be accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")
```

[ ]:
```
# D Agostino's K-squared test purchase_days_before_daprture
from scipy.stats import normaltest
stats , p_value = normaltest(thon['purchase_days_before_daprture'])
print("Stats = ",stats)
print("P_value = ", p_value)
if p_value > 0.5:
    print("Data is probabily normal or Guassian")
else:
    print("Data is probabily not Guassian")
```

```
Stats =  17590790.676418208
P_value =  0.0
```

```
Data is probabily not Guassian
```

6 Anderson-Darling test

[ ]:
```
from scipy.stats import anderson

result = (anderson(thon['price'], dist='norm'))
print('Statistic: %.3f' % result.statistic)
for i in range(len(result.critical_values)):
        sl, cv = result.significance_level[i], result.critical_values[i]
        if result.statistic < result.critical_values[i]:
                print('%.3f: %.3f, data looks normal (fail to reject H0)' %␣
 ↪(sl, cv))
        else:
                print('%.3f: %.3f, data does not look normal (reject H0)' %␣
 ↪(sl, cv))
```

```
Statistic: 872668.221
15.000: 0.576, data does not look normal (reject H0)
10.000: 0.656, data does not look normal (reject H0)
5.000: 0.787, data does not look normal (reject H0)
2.500: 0.918, data does not look normal (reject H0)
1.000: 1.092, data does not look normal (reject H0)
```

[ ]:
```
from scipy.stats import anderson

result = (anderson(thon['purchase_days_before_daprture'], dist='norm'))

print(f"A-D statistic: {result[0]}")
print(f"Critical values: {result[1]}")
print(f"Significance levels: {result[2]}")
```

```
A-D statistic: 1629886.5941748247
Critical values: [0.576 0.656 0.787 0.918 1.092]
Significance levels: [15.  10.   5.   2.5  1. ]
```