EDA

April 2, 2022

Student Name : Sartaj Ahmed Salman

Email: s2140019@edu.cc.uec.ac.jp

Phd Student At UEC Tokyo, Japan

Address: From Skardu, Pakistan

1 Exploratory Data Analysis (EDA)

EDA and its 10 Important steps

```
[ ]: # Import Data set
     import pandas as pd
     import numpy as np
     import seaborn as sns
     # data set
     df = pd.read_csv('Sample.csv')
     df1 = sns.load_dataset('tips')
     df2 = sns.load_dataset('titanic')
```

```
[ ]: # Step One is to check the shape of the Data set
     print(df.shape)
     row,column = df.shape
     print("Number of rows ", row)
     print("Number of columns ", column)
```

```
(30000, 5)
Number of rows  30000
Number of columns  5
```

```
[ ]: # Step 2 is to look the data-structure the detail info containing instances and␣
      ↪serires
     df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 5 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
```

```
 0   purchase_days_before_daprture  30000 non-null  int64
 1   airline                        30000 non-null  object
 2   baggage_weight                 30000 non-null  float64
 3   baggage_pieces                 30000 non-null  int64
 4   price                          30000 non-null  float64
dtypes: float64(2), int64(2), object(1)
memory usage: 1.1+ MB
```

```
[ ]: df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 16 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   891 non-null    int64
 1   survived     891 non-null    int64
 2   pclass       891 non-null    int64
 3   sex          891 non-null    object
 4   age          714 non-null    float64
 5   sibsp        891 non-null    int64
 6   parch        891 non-null    int64
 7   fare         891 non-null    float64
 8   embarked     889 non-null    object
 9   class        891 non-null    category
 10  who          891 non-null    object
 11  adult_male   891 non-null    bool
 12  deck         203 non-null    category
 13  embark_town  889 non-null    object
 14  alive        891 non-null    object
 15  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(5), object(5)
memory usage: 87.6+ KB
```

```
[ ]: # Step 3 is to look for the Missing values
     df.isnull().sum()
```

```
[ ]: purchase_days_before_daprture    0
     airline                          0
     baggage_weight                   0
     baggage_pieces                   0
     price                            0
     dtype: int64
```

```
[ ]: df2.isnull().sum()
```

```
[ ]: Unnamed: 0       0
     survived         0
```

```
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

[ ]:
```
# Calculate the percentage of missing values
df2.isnull().sum()/df2.shape[0]*100
```

[ ]:
```
Unnamed: 0      0.000000
survived        0.000000
pclass          0.000000
sex             0.000000
age            19.865320
sibsp           0.000000
parch           0.000000
fare            0.000000
embarked        0.224467
class           0.000000
who             0.000000
adult_male      0.000000
deck           77.216611
embark_town     0.224467
alive           0.000000
alone           0.000000
dtype: float64
```

[ ]:
```
# Step 4 is to Split the data we can also add new columns or do some feature␣
 ↪engineering
city = pd.DataFrame(np.array([['lahore, pakistan', 30],['tokyo, japan',␣
 ↪20],['berlin, germany', 45]]),columns=["address","participants"])
```

[ ]:
```
city
```

[ ]:
```
            address participants
0  lahore, pakistan           30
1      tokyo, japan           20
```

```
2   berlin, germany           45
```

```
[ ]: city[['city','country']] = city['address'].str.split(', ',expand=True)
```

```
[ ]: city
```

```
[ ]:             address participants    city   country
     0  lahore, pakistan           30  lahore  pakistan
     1      tokyo, japan           20   tokyo     japan
     2   berlin, germany           45  berlin   germany
```

```
[ ]: city.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   address       3 non-null      object
 1   participants  3 non-null      object
 2   city          3 non-null      object
 3   country       3 non-null      object
dtypes: object(4)
memory usage: 224.0+ bytes
```

```
[ ]: # Step 5 is to do the type casting conversion of data type
     # first to int
     city['participants'] = city['participants'].astype('int')
     city.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   address       3 non-null      object
 1   participants  3 non-null      int32
 2   city          3 non-null      object
 3   country       3 non-null      object
dtypes: int32(1), object(3)
memory usage: 212.0+ bytes
```

2 *To convert an obejct into string we need to write string instead of str solution of the question*

```
[ ]: # first to str
     city['city'] = city['city'].astype('string')
     city.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   address       3 non-null      object
 1   participants  3 non-null      int32
 2   city          3 non-null      string
 3   country       3 non-null      object
dtypes: int32(1), object(2), string(1)
memory usage: 212.0+ bytes
```

```
[ ]: # Step 6 is to look for the data set summary
     df2.describe()
```

```
[ ]:        Unnamed: 0    survived      pclass         age       sibsp       parch  \
     count  891.000000  891.000000  891.000000  714.000000  891.000000  891.000000
     mean   445.000000    0.383838    2.308642   29.699118    0.523008    0.381594
     std    257.353842    0.486592    0.836071   14.526497    1.102743    0.806057
     min      0.000000    0.000000    1.000000    0.420000    0.000000    0.000000
     25%    222.500000    0.000000    2.000000   20.125000    0.000000    0.000000
     50%    445.000000    0.000000    3.000000   28.000000    0.000000    0.000000
     75%    667.500000    1.000000    3.000000   38.000000    1.000000    0.000000
     max    890.000000    1.000000    3.000000   80.000000    8.000000    6.000000

                  fare
     count  891.000000
     mean    32.204208
     std     49.693429
     min      0.000000
     25%      7.910400
     50%     14.454200
     75%     31.000000
     max    512.329200
```

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

```
[ ]:        purchase_days_before_daprture  baggage_weight  baggage_pieces  \
     count                   30000.000000    30000.000000    30000.000000
     mean                       15.589133        0.505014        0.947567
```

```
std                          18.949462        0.197538        0.605444
min                           0.000000        0.000000        0.000000
25%                           4.000000        0.444444        1.000000
50%                          10.000000        0.444444        1.000000
75%                          20.000000        0.711111        1.000000
max                         279.000000        1.000000        2.000000

              price
count  30000.000000
mean   10148.610833
std     3455.986201
min     1000.000000
25%     7796.000000
50%     9403.000000
75%    11245.000000
max    35000.000000
```

```
[ ]: # Step 7 is to count the number of values in the any specific coloum of the data setr
     df['price'].value_counts()
```

```
[ ]: 7524.0     844
     10545.0    819
     12645.0    680
     9045.0     668
     11245.0    579
               …
     9535.0       1
     7068.0       1
     10262.0      1
     9857.0       1
     10795.0      1
     Name: price, Length: 987, dtype: int64
```

```
[ ]: df['airline'].value_counts()
```

```
[ ]: alpha    13145
     gamma    10399
     beta      5525
     omega      931
     Name: airline, dtype: int64
```

```
[ ]: df['airline'].unique()
```

```
[ ]: array(['alpha', 'beta', 'gamma', 'omega'], dtype=object)
```

[ ]:

```
# Deal with duplicate
a = df.sample(30)
```

[ ]:

```
# deal with duplicates and null values (mean replacment, median replacment)
a[a.airline == "alpha"]
# we can remove duplicates,
```

[ ]:

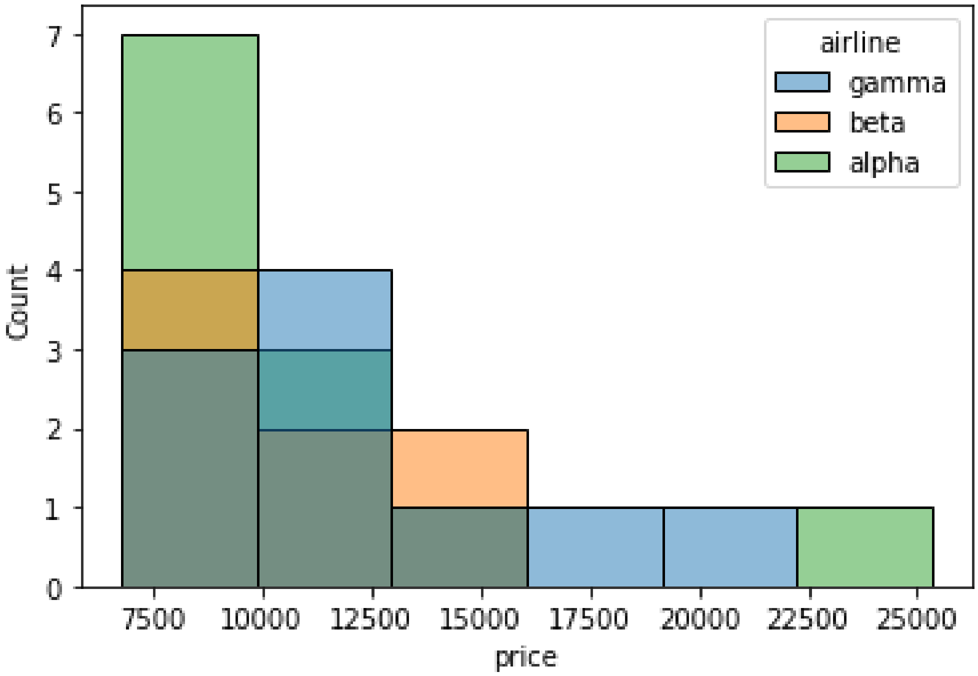
```
       purchase_days_before_daprture airline  baggage_weight  baggage_pieces  \
28097                              3   alpha        0.777778               1
12734                              1   alpha        0.333333               1
18212                             88   alpha        0.333333               1
27076                              1   alpha        0.444444               1
8855                               1   alpha        0.333333               1
2565                               5   alpha        0.777778               1
25552                             38   alpha        0.777778               1
8362                              74   alpha        0.777778               1
22421                              7   alpha        0.444444               1
26245                              9   alpha        0.777778               1
2593                              20   alpha        0.333333               1
17335                             35   alpha        0.333333               1

         price
28097  10799.0
12734   6810.0
18212   9798.0
27076   9799.0
8855    7524.0
2565   13277.0
25552  12645.0
8362    6785.0
22421  25345.0
26245   7900.0
2593   12945.0
17335   7796.0
```

[ ]:

```
# Step 9 is to check the normality
sns.histplot(a, x='price',hue='airline')
```

[ ]:

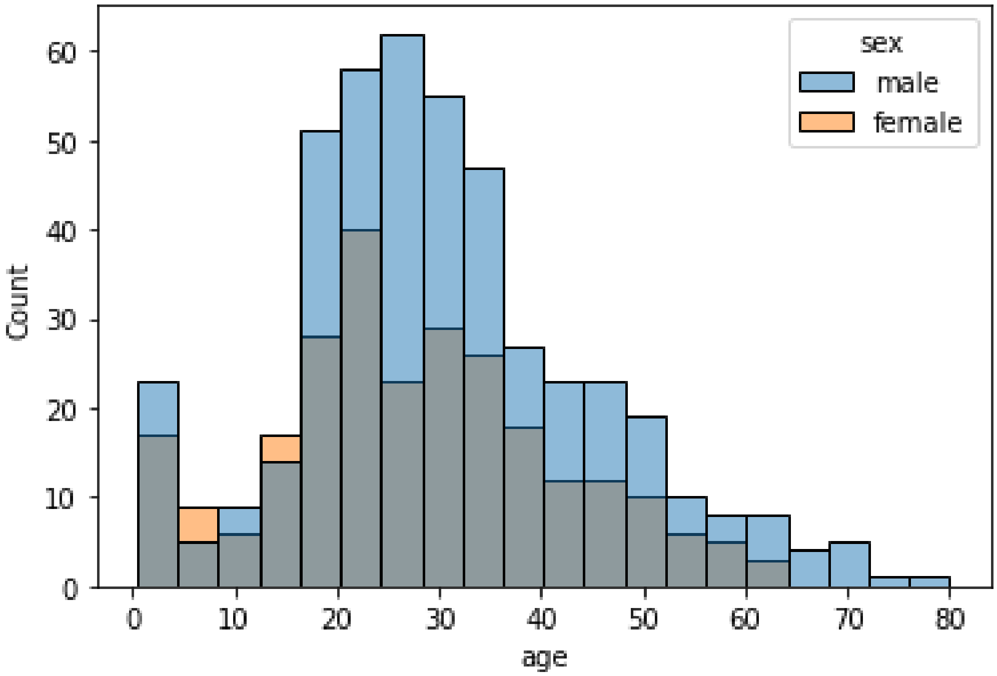
```
<AxesSubplot:xlabel='price', ylabel='Count'>
```



3 *Hisplot of two categorical values*

```
[ ]: # Step 9 is to check the normality
     sns.histplot(df2, x='age',hue='sex')
```

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



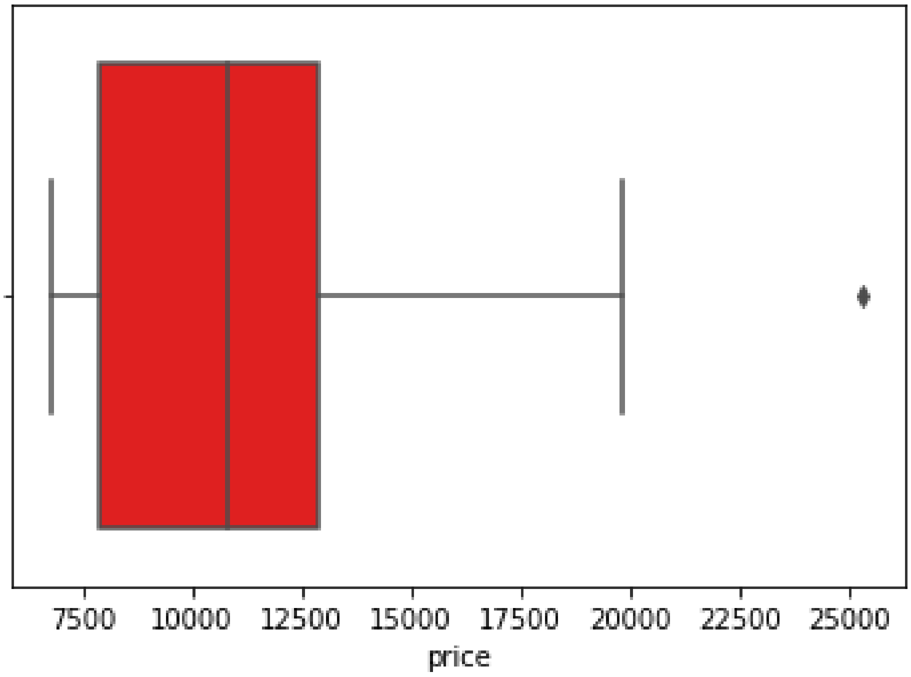
```
[ ]: # also we can measure skewness and kurtosis of the column
     a['price'].agg(['skew','kurtosis']).transpose()
```

```
[ ]: skew        1.623522
     kurtosis    3.585570
     Name: price, dtype: float64
```

```
[ ]: # we can also make boxplots
     sns.boxplot(a['price'],color='red')
```

```
C:\Users\Sartaj\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

```
[ ]: <AxesSubplot:xlabel='price'>
```

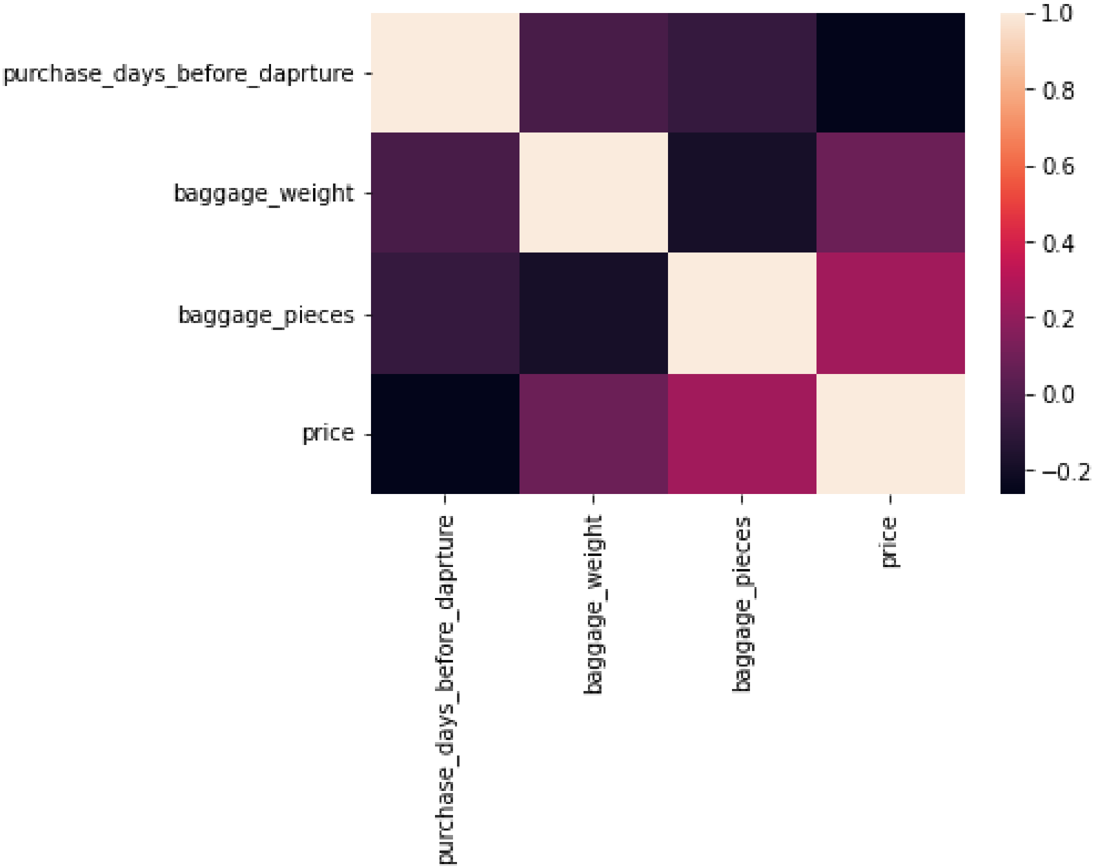


```
[ ]: # Step 10 is to draw the corellation
     corr = a.corr(method='pearson')
     corr
```

```
[ ]:                               purchase_days_before_daprture  baggage_weight  \
     purchase_days_before_daprture                       1.000000       -0.022514
     baggage_weight                                     -0.022514        1.000000
     baggage_pieces                                     -0.084014       -0.186334
     price                                              -0.262808        0.083882

                                    baggage_pieces     price
     purchase_days_before_daprture       -0.084014 -0.262808
     baggage_weight                      -0.186334  0.083882
     baggage_pieces                       1.000000  0.242578
     price                                0.242578  1.000000
```

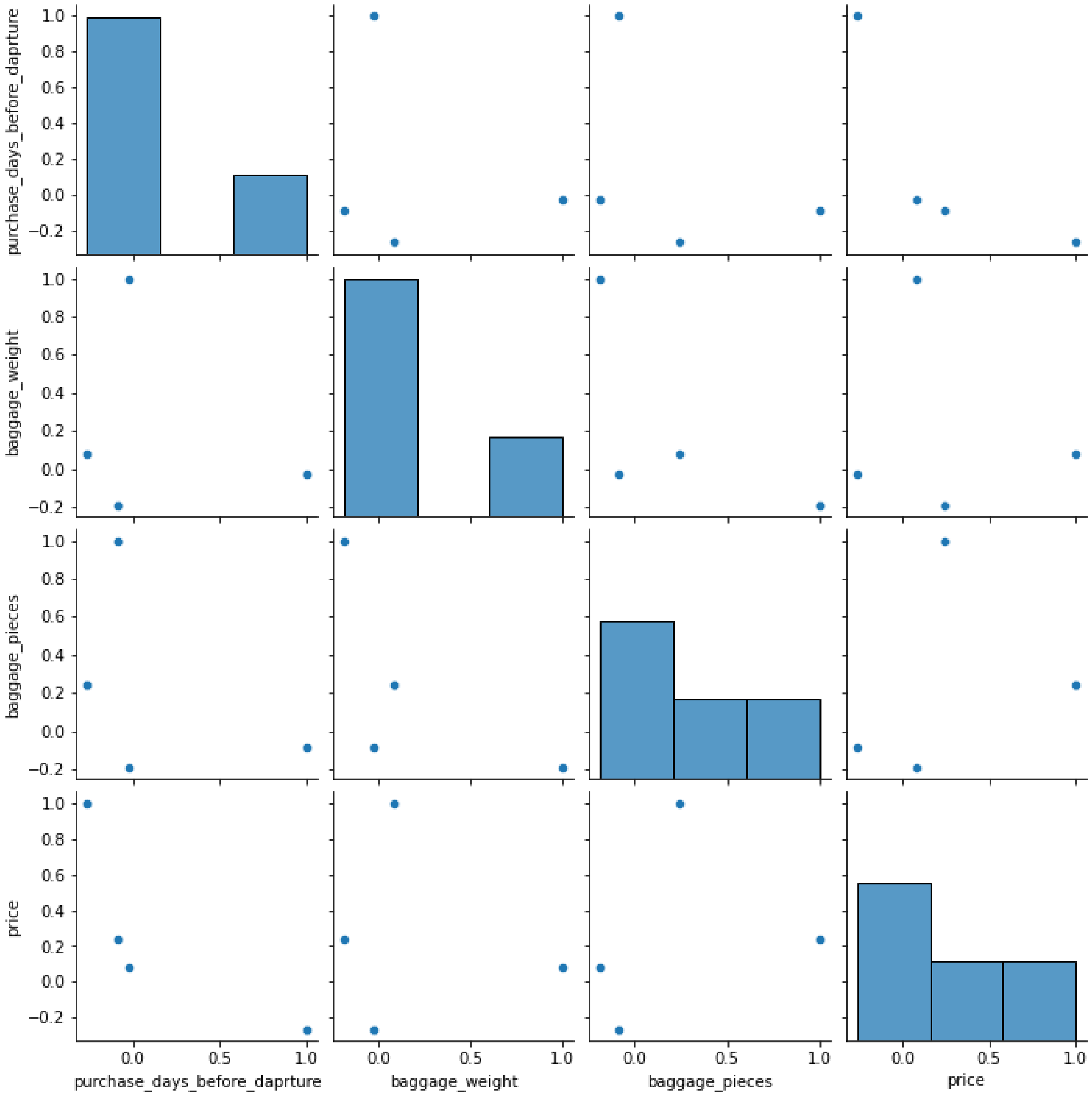
```
[ ]: sns.heatmap(corr)
```

```
[ ]: <AxesSubplot:>
```



```
[ ]: sns.pairplot(corr)
```

```
[ ]: <seaborn.axisgrid.PairGrid at 0x28997152610>
```

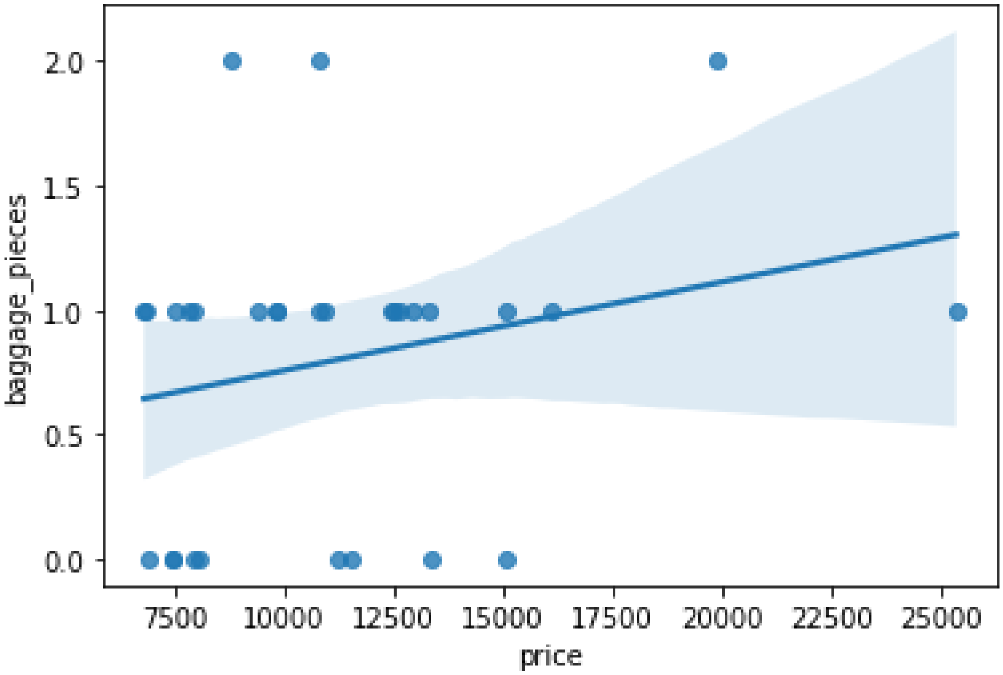


[ ]:
```
corr.style.background_gradient('coolwarm')
```

[ ]:
```
<pandas.io.formats.style.Styler at 0x28997977700>
```

[ ]:
```
# regplot to check the positive correlation
sns.regplot(a['price'],a['baggage_pieces'],data=a)
```

```
C:\Users\Sartaj\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
[ ]: <AxesSubplot:xlabel='price', ylabel='baggage_pieces'>
```


2.0
1.5
1.0
0.5
0.0
baggage_pieces
7500
10000
12500
15000
17500
20000
22500
25000
price

```
[ ]:
```