Part 2

Step 1,2

In [1580]:
```
# importing the flight dataset from the folder and the required libraries
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

df=pd.read_csv(r"C:\Users\hasan\Downloads\datasets\datasets\flight_price_prediction.csv")
df
```

Out[1580]:

	Unnamed: 0	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price
0	0	SpiceJet	SG-8709	Delhi	Evening	zero	Night	Mumbai	Economy	2.17	1	5953
1	1	SpiceJet	SG-8157	Delhi	Early_Morning	zero	Morning	Mumbai	Economy	2.33	1	5953
2	2	AirAsia	I5-764	Delhi	Early_Morning	zero	Early_Morning	Mumbai	Economy	2.17	1	5956
3	3	Vistara	UK-995	Delhi	Morning	zero	Afternoon	Mumbai	Economy	2.25	1	5955
4	4	Vistara	UK-963	Delhi	Morning	zero	Morning	Mumbai	Economy	2.33	1	5955
...	...	...	...	...	...	...	...	...	...	...	...	...
300148	300148	Vistara	UK-822	Chennai	Morning	one	Evening	Hyderabad	Business	10.08	49	69265
300149	300149	Vistara	UK-826	Chennai	Afternoon	one	Night	Hyderabad	Business	10.42	49	77105
300150	300150	Vistara	UK-832	Chennai	Early_Morning	one	Night	Hyderabad	Business	13.83	49	79099
300151	300151	Vistara	UK-828	Chennai	Early_Morning	one	Evening	Hyderabad	Business	10.00	49	81585
300152	300152	Vistara	UK-822	Chennai	Morning	one	Evening	Hyderabad	Business	10.08	49	81585

300153 rows × 12 columns

Step 3

In [1530]:
```
# getting the basic statistics of the dataset

df.describe()
```

Out[1530]:

	Unnamed: 0	duration	days_left	price
count	300153.000000	300153.000000	300153.000000	300153.000000
mean	150076.000000	12.221021	26.004751	20889.660523
std	86646.852011	7.191997	13.561004	22697.767366
min	0.000000	0.830000	1.000000	1105.000000
25%	75038.000000	6.830000	15.000000	4783.000000
50%	150076.000000	11.250000	26.000000	7425.000000
75%	225114.000000	16.170000	38.000000	42521.000000
max	300152.000000	49.830000	49.000000	123071.000000

In [1460]:
```
#checking if there is any null value in the datset

df.isna().sum()
```

Out[1460]:
```
Unnamed: 0          0
airline             0
flight              0
source_city         0
departure_time      0
stops               0
arrival_time        0
destination_city    0
class               0
duration            0
days_left           0
price               0
dtype: int64
```

In [1461]:
```
df.dtypes
```

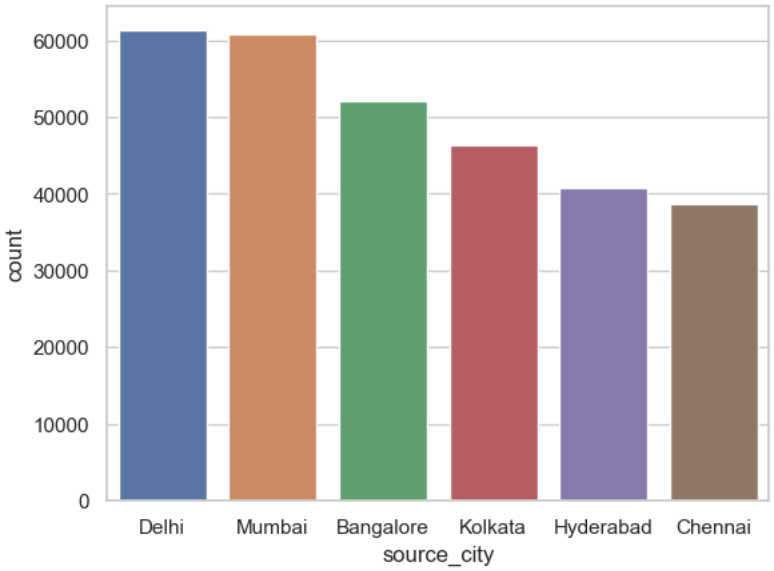
Out[1461]:
```
Unnamed: 0            int64
airline              object
flight               object
source_city          object
departure_time       object
stops                object
arrival_time         object
destination_city     object
class                object
duration            float64
days_left             int64
price                 int64
dtype: object
```

In [1581]:
```
# dropping the flight column

df.drop("flight",axis=1,inplace=True)
```

Step 4

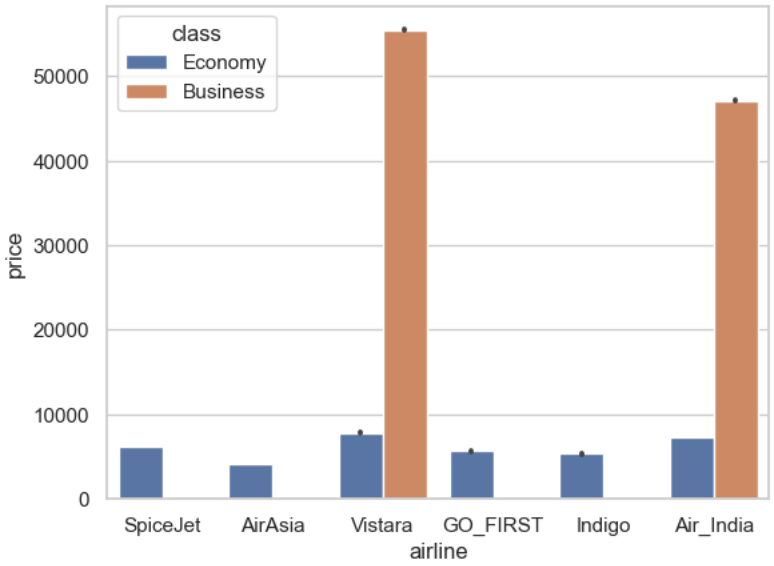
In [1438]:
```
# plotting a count plot for number of flights from different cities

sns.set(style = 'whitegrid')
sns.countplot(df["source_city"])
plt.show()
```

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



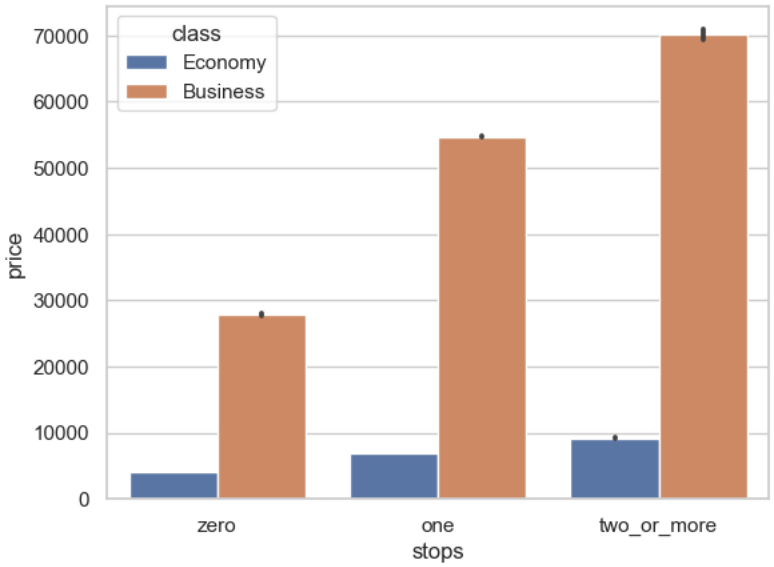
In [1398]:
```
# Plot 1
# plotting a Price for different airlines on different classes

sns.barplot(df.airline,df['price'],hue=df["class"])
plt.show()
```

```
C:\Users\hasan\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 exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
```



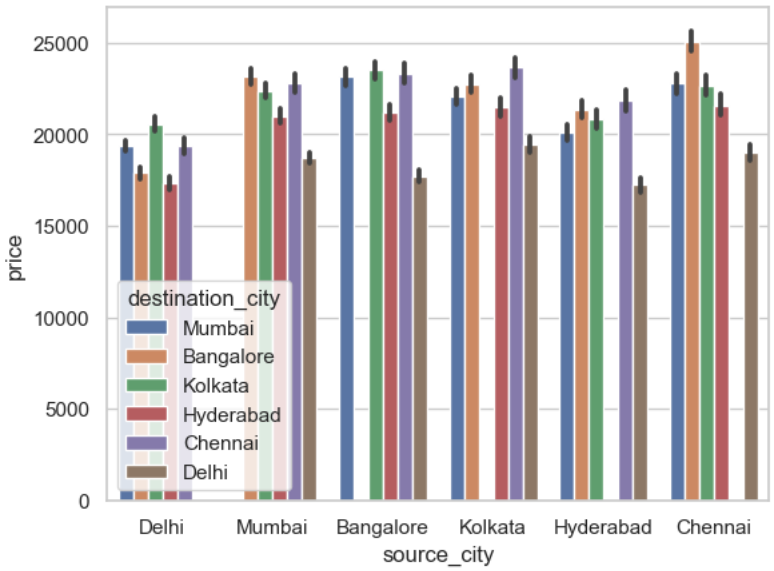
In [1399]:
```
# plotting a Price for different number of stops on different classes

sns.barplot(df.stops,df['price'],hue=df["class"])
plt.show()
```

```
C:\Users\hasan\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 exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
```



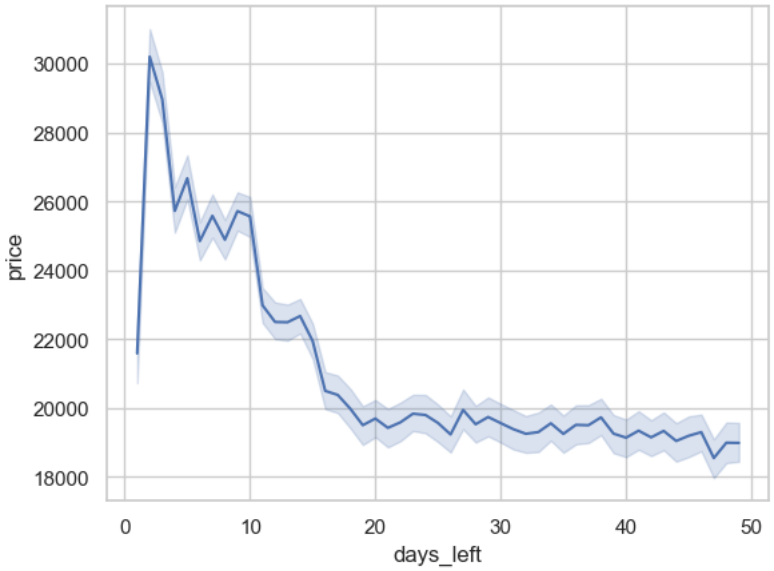
In [1491]:
```
# Plot 2
# plotting a Price for different source and destination

sns.barplot(df.source_city,df['price'],hue=df["destination_city"])
plt.show()
```

```
C:\Users\hasan\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 exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [1400]:
```
# Plot 3
# plotting a Price for days_left

sns.lineplot(df.days_left,df['price'])
plt.show()
```

```
C:\Users\hasan\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 exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [1567]:
```
# Pre-processing all the day_left value >20 to 20 from the result of graph above

df.loc[df["days_left"]>20, "days_left"] = 20
df
```

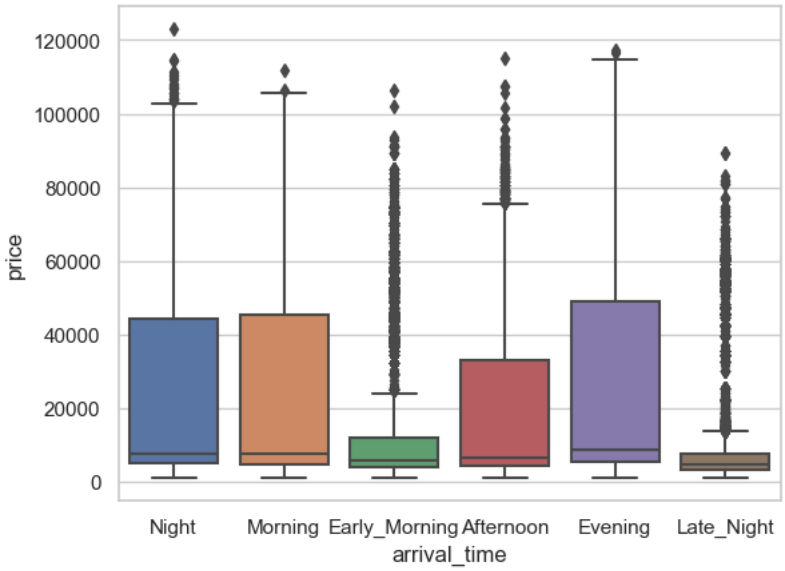
Out[1567]:

	Unnamed: 0	airline	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price
0	0	SpiceJet	Delhi	Evening	zero	Night	Mumbai	Economy	2.17	1	5953
1	1	SpiceJet	Delhi	Early_Morning	zero	Morning	Mumbai	Economy	2.33	1	5953
2	2	AirAsia	Delhi	Early_Morning	zero	Early_Morning	Mumbai	Economy	2.17	1	5956
3	3	Vistara	Delhi	Morning	zero	Afternoon	Mumbai	Economy	2.25	1	5955
4	4	Vistara	Delhi	Morning	zero	Morning	Mumbai	Economy	2.33	1	5955
...	...	...	...	...	...	...	...	...	...	...	...
300148	300148	Vistara	Chennai	Morning	one	Evening	Hyderabad	Business	10.08	20	69265
300149	300149	Vistara	Chennai	Afternoon	one	Night	Hyderabad	Business	10.42	20	77105
300150	300150	Vistara	Chennai	Early_Morning	one	Night	Hyderabad	Business	13.83	20	79099
300151	300151	Vistara	Chennai	Early_Morning	one	Evening	Hyderabad	Business	10.00	20	81585
300152	300152	Vistara	Chennai	Morning	one	Evening	Hyderabad	Business	10.08	20	81585

300153 rows × 11 columns

In [1403]:
```
# Plot 4
# plotting a box plot of Price based on arrival time

sns.boxplot(df.arrival_time,df['price'])
plt.show()
```

```
C:\Users\hasan\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 exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [1404]:
```
# checking the unique values of the categorical column

print(df.airline.unique())
print(df.source_city.unique())
print(df.departure_time.unique())
print(df.stops.unique())
print(df.arrival_time.unique())
print(df.destination_city.unique())
print(df["class"].unique())
```

```
['SpiceJet' 'AirAsia' 'Vistara' 'GO_FIRST' 'Indigo' 'Air_India']
['Delhi' 'Mumbai' 'Bangalore' 'Kolkata' 'Hyderabad' 'Chennai']
['Evening' 'Early_Morning' 'Morning' 'Afternoon' 'Night' 'Late_Night']
['zero' 'one' 'two_or_more']
['Night' 'Morning' 'Early_Morning' 'Afternoon' 'Evening' 'Late_Night']
['Mumbai' 'Bangalore' 'Kolkata' 'Hyderabad' 'Chennai' 'Delhi']
['Economy' 'Business']
```

In [1582]:
```
# performing the label encoding for all the categorical column

cat_col=("airline","source_city","departure_time","stops","arrival_time","destination_city","class")
for i in cat_col:
    unique_value=df[i].unique()

    n=1
    for j in unique_value:
        df.loc[df[i]==j, i] = n
        n+=1
df.drop("Unnamed: 0",axis=1,inplace=True)

df
```

Out[1582]:

	airline	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price
0	1	1	1	1	1	1	1	2.17	1	5953
1	1	1	2	1	2	1	1	2.33	1	5953
2	2	1	2	1	3	1	1	2.17	1	5956
3	3	1	3	1	4	1	1	2.25	1	5955
4	3	1	3	1	2	1	1	2.33	1	5955
...	...	...	...	...	...	...	...	...	...	...
300148	3	6	3	2	5	4	2	10.08	49	69265
300149	3	6	4	2	1	4	2	10.42	49	77105
300150	3	6	2	2	1	4	2	13.83	49	79099
300151	3	6	2	2	5	4	2	10.00	49	81585
300152	3	6	3	2	5	4	2	10.08	49	81585

300153 rows × 10 columns

In [1583]:
```
cat_col=("duration","airline","source_city","departure_time","stops","arrival_time","destination_city","class")

for i in df.columns:
    df[i] = df[i].astype("int")
df.dtypes
```

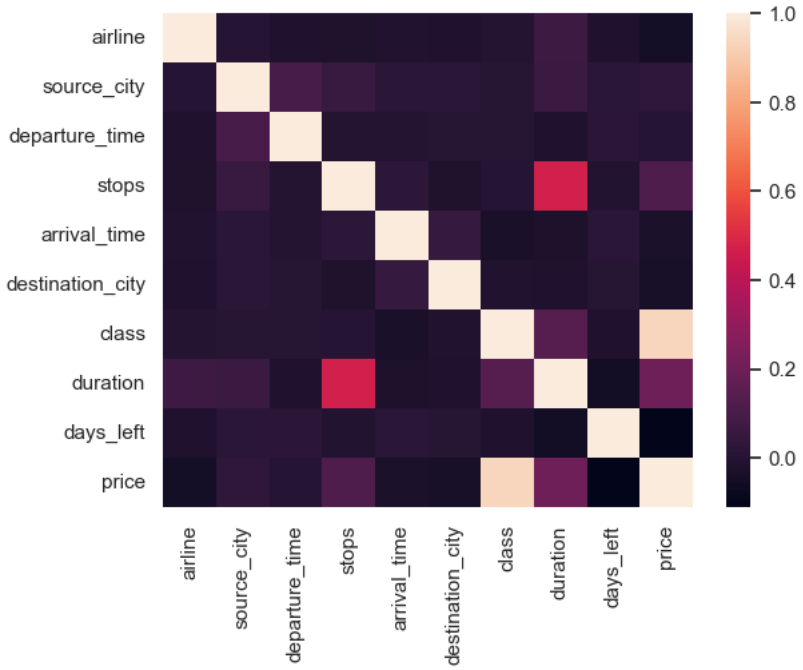
Out[1583]:
```
airline             int32
source_city         int32
departure_time      int32
stops               int32
arrival_time        int32
destination_city    int32
class               int32
duration            int32
days_left           int32
price               int32
dtype: object
```

In [1584]:
```
df.corr()
```

Out[1584]:

	airline	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price
airline	1.000000	-0.000563	-0.017736	-0.019418	-0.013514	-0.018620	-0.001687	0.065316	-0.010377	-0.052792
source_city	-0.000563	1.000000	0.101526	0.057762	0.015313	0.012365	0.006292	0.062685	0.008704	0.026990
departure_time	-0.017736	0.101526	1.000000	-0.003628	-0.005741	0.003258	0.007133	-0.015779	0.010374	0.002522
stops	-0.019418	0.057762	-0.003628	1.000000	0.022885	-0.022383	0.001027	0.467745	-0.008540	0.119648
arrival_time	-0.013514	0.015313	-0.005741	0.022885	1.000000	0.050441	-0.032293	-0.025124	0.005739	-0.031176
destination_city	-0.018620	0.012365	0.003258	-0.022383	0.050441	1.000000	-0.013714	-0.017191	0.005919	-0.036952
class	-0.001687	0.006292	0.007133	0.001027	-0.032293	-0.013714	1.000000	0.139375	-0.013039	0.937860
duration	0.065316	0.062685	-0.015779	0.467745	-0.025124	-0.017191	0.139375	1.000000	-0.039206	0.205079
days_left	-0.010377	0.008704	0.010374	-0.008540	0.005739	0.005919	-0.013039	-0.039206	1.000000	-0.091949
price	-0.052792	0.026990	0.002522	0.119648	-0.031176	-0.036952	0.937860	0.205079	-0.091949	1.000000

In [1556]:
```
# Plot 5
# plotting the corelation of different features

sns.heatmap(df.corr())
plt.show()
```



In [1585]:
```
# Adding new feature to the dataset based on the previous data analysis which helped to improve the mse

df["stop_class_"]=df["stops"]*df["class"].astype("int")
#df["dur_stop_"]=np.log(df["duration"])*df["stops"].astype("int")

df
```

Out[1585]:

	airline	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price	stop_class_
0	1	1	1	1	1	1	1	2	1	5953	1
1	1	1	2	1	2	1	1	2	1	5953	1
2	2	1	2	1	3	1	1	2	1	5956	1
3	3	1	3	1	4	1	1	2	1	5955	1
4	3	1	3	1	2	1	1	2	1	5955	1
...	...	...	...	...	...	...	...	...	...	...	...
300148	3	6	3	2	5	4	2	10	49	69265	4
300149	3	6	4	2	1	4	2	10	49	77105	4
300150	3	6	2	2	1	4	2	13	49	79099	4
300151	3	6	2	2	5	4	2	10	49	81585	4
300152	3	6	3	2	5	4	2	10	49	81585	4

300153 rows × 11 columns

Step 5

In [1586]:
```
df.dtypes
```

Out[1586]:
```
airline             int32
source_city         int32
departure_time      int32
stops               int32
arrival_time        int32
destination_city    int32
class               int32
duration            int32
days_left           int32
price               int32
stop_class_         int32
dtype: object
```

In [1409]:
```
print(df.airline.unique())
print(df.source_city.unique())
print(df.departure_time.unique())
print(df.stops.unique())
print(df.arrival_time.unique())
print(df.destination_city.unique())
print(df["class"].unique())
```

```
[1 2 3 4 5 6]
[1 2 3 4 5 6]
[1 2 3 4 5 6]
[1 2 3]
[1 2 3 4 5 6]
[1 2 3 4 5 6]
[1 2]
```

In [1588]:
```
# defining the denormalization function for the price column to get the final number in the same range

price_max=df["price"].max()
price_min=df["price"].min()
def denorm(x):
    return (x * (price_max-price_min) + price_min)
```

In [1589]:
```
# Normalizing the required column

independen_variable=["duration","days_left","price"]
for i in independen_variable:
    df[i]=(df[i]-df[i].min())/(df[i].max()-df[i].min())
#df.drop(columns=["arrival_time","stops","destination_city","stop_class_"],inplace=True)
df
```

Out[1589]:

	airline	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price	stop_class_
0	1	1	1	1	1	1	1	0.040816	0.0	0.039749	1
1	1	1	2	1	2	1	1	0.040816	0.0	0.039749	1
2	2	1	2	1	3	1	1	0.040816	0.0	0.039773	1
3	3	1	3	1	4	1	1	0.040816	0.0	0.039765	1
4	3	1	3	1	2	1	1	0.040816	0.0	0.039765	1
...	...	...	...	...	...	...	...	...	...	...	...
300148	3	6	3	2	5	4	2	0.204082	1.0	0.558844	4
300149	3	6	4	2	1	4	2	0.204082	1.0	0.623124	4
300150	3	6	2	2	1	4	2	0.265306	1.0	0.639473	4
300151	3	6	2	2	5	4	2	0.204082	1.0	0.659856	4
300152	3	6	3	2	5	4	2	0.204082	1.0	0.659856	4

300153 rows × 11 columns

In [1604]:
```
#df.drop(columns=["stops"],inplace=True)
df.drop(columns=["duration"],inplace=True)
#df.drop(columns=["arrival_time","destination_city","departure_time"],inplace=True)
```

Step 6,7,8,9

In [1590]:
```
# Taking Price at the target variable
# defining the train and test split function

import numpy as np
def train_test_split(df):
    train_index = np.random.rand(len(df)) < 0.8
    train_data = df[train_index]
    test_data = df[~train_index]
    train_x=train_data.drop("price",axis=1)
    test_x=test_data.drop("price",axis=1)
    train_y=train_data["price"]
    test_y=test_data["price"]
    return(train_x,train_y,test_x,test_y)
```

In [1605]:
```
# printing the shape of train and test datasets

train_x,train_y,test_x,test_y=train_test_split(df)

print(train_x.shape)
print(train_y.shape)
print(test_x.shape)
print(test_y.shape)
```

```
(240016, 9)
(240016,)
(60137, 9)
(60137,)
```

Step 10, 11

In [1592]:
```
class LinearRegression:
    def __init__(self):
        self.weight = None


    def ols(self,train_x,train_y):
        fir=np.dot(train_x.T,train_x)
        fir=np.linalg.inv(fir)
        sec=np.dot(train_x.T,train_y)
        f_weight=np.dot(fir,sec)
        self.weight=f_weight
        return (self.weight)

    def predict(self,test_x):
        y_hat=np.dot(test_x,self.weight.T)
        return (y_hat)

    # find the mse
    def ols_los(self,test_x,test_y):
        predicted=self.predict(test_x)
        mse=.5*np.mean((test_y-predicted)**2)
        return mse
```

In [1606]:
```
# fitting the Linear Regression class defined above

model_linear=LinearRegression()
model_linear.ols(train_x,train_y)
model_linear.predict(test_x)
model_linear.ols_los(test_x,test_y)
```

Out[1606]:
```
0.0012743227144145297
```

In [1607]:
```
model_linear.ols_los(train_x,train_y)
```

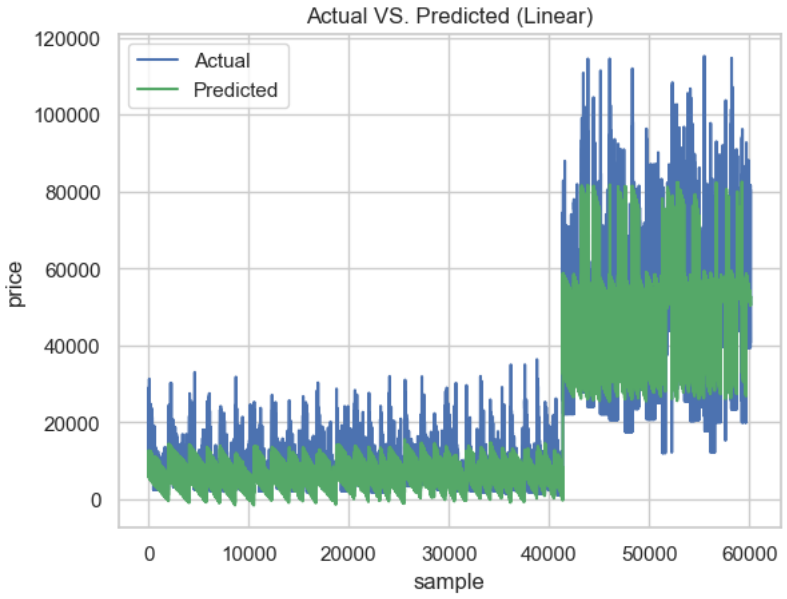
Out[1607]:
```
0.001279702284629127
```

In [1608]:
```
# printing the weight by OLS to different column

model_linear.weight
```

Out[1608]:
```
array([-0.00596723,  0.00143208, -0.00067929, -0.14369821, -0.00038464,
       -0.00198548,  0.05192788, -0.05104179,  0.16901337])
```

Step 12

In [1609]:
```
plt.plot(np.array(denorm(test_y)),label = "Actual",c = "b")
plt.plot(denorm(model_linear.predict(test_x)),label = "Predicted",c = "g")
plt.xlabel("sample")
plt.ylabel("price")
plt.title("Actual VS. Predicted (Linear)")
plt.legend()
plt.show()
```



In [1597]:
```
# saving the Linear regression model as the pickle file

import pickle
with open('Hasan_Hussain_assignment1_part_2', 'wb') as files:
    pickle.dump(model_linear, files)
```

Part 3(Ridge Regression)

In [1610]:
```
class RidgeRegression:

    def __init__(self, alpha):
        self.weight = None
        self.alpha= alpha

    def ols(self,train_x,train_y):
        I=np.identity(train_x.shape[1])
        fir=np.dot(train_x.T,train_x) + self.alpha*I
        fir=np.linalg.inv(fir)
        sec=np.dot(train_x.T,train_y)
        f_weight=np.dot(fir,sec)
        self.weight=f_weight
        return (self.weight)

    def predict(self,test_x):
        y_hat=np.dot(test_x,self.weight.T)
        return (y_hat)

    def ols_los(self,test_x,test_y):
        predicted=self.predict(test_x)
        mse=.5*np.mean((test_y-predicted)**2) + ((self.alpha/2)*(np.dot(self.weight.T,self.weight)))

        return mse
```

In [1611]:
```
Ridge_loss={}
for i in [.000001,.00001,.001,.1,1]:
    model_Ridge=RidgeRegression(alpha=i)
    model_Ridge.ols(train_x,train_y)
    model_Ridge.predict(test_x)

    # getting the test error
    Ridge_loss[i]=model_Ridge.ols_los(test_x,test_y)
print(f"Best MSE for Ridge reg is {min(Ridge_loss.values())} for lambda value {min(Ridge_loss.keys())}")
```

```
Best MSE for Ridge reg is 0.0012743499937520025 for lambda value 1e-06
```

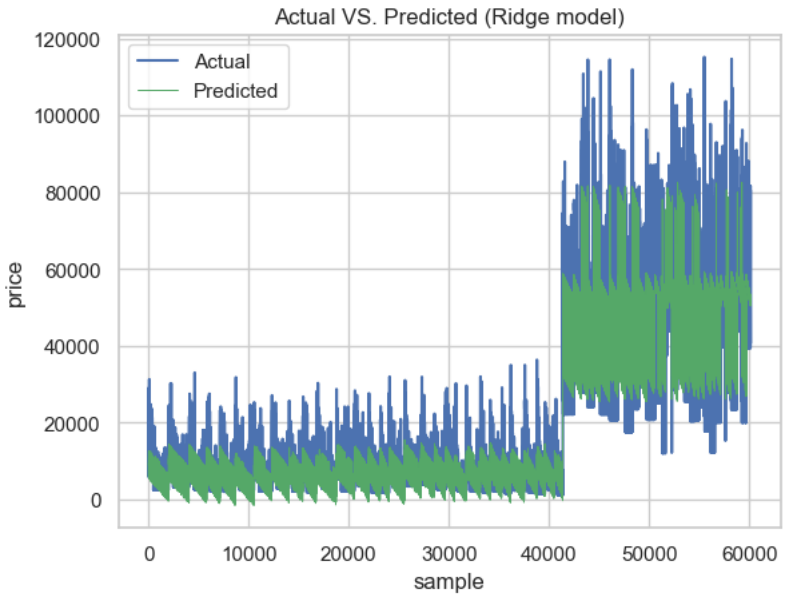
In [1612]:
```
model_Ridge.ols_los(train_x,train_y)
```

Out[1612]:
```
0.028557491313466073
```

In [1613]:
```
model_Ridge.weight
```

Out[1613]:
```
array([-0.00596796,  0.00143174, -0.0006798 , -0.14369164, -0.00038501,
       -0.00198588,  0.0519364 , -0.0510416 ,  0.1690072 ])
```

In [1614]:
```
plt.plot(np.array(denorm(test_y)),label = "Actual",c = "b")
plt.plot(denorm(model_Ridge.predict(test_x)),label = "Predicted",c = "g",linewidth=.7)
plt.xlabel("sample")
plt.ylabel("price")
plt.title("Actual VS. Predicted (Ridge model)")
plt.legend()
plt.show()
```



In [1615]:
```
# saving the Ridge regression model as the pickle file

import pickle
with open('Hasan_Hussain_assignment1_part_3', 'wb') as files:
    pickle.dump(model_Ridge, files)
```

Resources

https://www.kaggle.com/code/residentmario/ridge-regression-cost-function (https://www.kaggle.com/code/residentmario/ridge-regression-cost-function)

https://www.geeksforgeeks.org/implementation-of-elastic-net-regression-from-scratch/?ref=rp (https://www.geeksforgeeks.org/implementation-of-elastic-net-regression-from-scratch/?ref=rp)