Implementing Gradient Descent from Scratch

In [92]:

```
# importing the flight dataset from the folder and the required libraries
import pandas as pd
import seaborn as sns
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt

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

Out[92]:

	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

In [93]:

```
# dropping the flight column

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

In [94]:

```
# 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[94]:

	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 [95]:
```
# 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"]
df["dur_stop_"]=np.log(df["duration"])*df["stops"]

df
```

Out[95]:

	airline	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price	stop_class_	dur_stop_
0	1	1	1	1	1	1	1	2.17	1	5953	1	0.774727
1	1	1	2	1	2	1	1	2.33	1	5953	1	0.845868
2	2	1	2	1	3	1	1	2.17	1	5956	1	0.774727
3	3	1	3	1	4	1	1	2.25	1	5955	1	0.81093
4	3	1	3	1	2	1	1	2.33	1	5955	1	0.845868
...	...	...	...	...	...	...	...	...	...	...	...	...
300148	3	6	3	2	5	4	2	10.08	49	69265	4	4.621107
300149	3	6	4	2	1	4	2	10.42	49	77105	4	4.687454
300150	3	6	2	2	1	4	2	13.83	49	79099	4	5.25368
300151	3	6	2	2	5	4	2	10.00	49	81585	4	4.60517
300152	3	6	3	2	5	4	2	10.08	49	81585	4	4.621107

300153 rows × 12 columns

In [96]:
```
# converting different datatype like object to int for defined columns

cat_col=("duration","dur_stop_","stop_class_","airline","source_city","departure_time","stops","arrival_time","destination_ci

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

Out[96]:
```
airline             int32
source_city         int32
departure_time      int32
stops               int32
arrival_time        int32
destination_city    int32
class               int32
duration            int32
days_left           int64
price               int64
stop_class_         int32
dur_stop_           int32
dtype: object
```

In [97]:
```
# 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 [98]:

```
# 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[98]:

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

300153 rows × 12 columns

In [99]:

```
#df.drop(columns=["arrival_time","destination_city"],inplace=True)
```

In [100]:

```
# Taking Price at the target variable
# defining the train and test split function

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 [101]:

```
# 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)
```

```
(239751, 11)
(239751,)
(60402, 11)
(60402,)
```

In [102]:
```
class Ridge_GD:

    def __init__(self, itr, learning_rate,lamda):
        self.learning_rate = learning_rate
        self.itr = itr
        self.lamda=lamda
        self.weights = None
        self.losss = []
        self.we = []

    def loss(self,test_x,test_y):
        predicted=self.predict(test_x)
        mse=.5*np.mean((test_y-predicted)**2) + ((self.lamda/2)*(np.dot(self.weights.T,self.weights)))
        return mse


    def gradient_descent(self,x_train,train_y, y_predicted):
        delta = y_predicted- train_y
        dW=(2*(np.dot(x_train.T,delta)) + (2*self.lamda*self.weights))/x_train.shape[0]
        return (dW)

    def fit(self, x_train, train_y):
        self.weights=np.ones(x_train.shape[1])
        for i in range(self.itr):
            z = np.dot(x_train,self.weights.T)
            y_predicted= z
            dW= self.gradient_descent(x_train,train_y, y_predicted)
            self.weights=self.weights-(self.learning_rate*dW)
            loss=self.loss(x_train, train_y)

            self.losss.append(loss)
            print(f"For Iteration {i} the Loss is {round(self.losss[i],4)}.")
            self.we.append(self.weights)

    def predict(self, x_train):
        z=np.dot(x_train,self.weights.T)
        y_predicted=z


        return(y_predicted)
```

In [103]:
```
from datetime import datetime
start_time = datetime.now()

model_Ridgegd=Ridge_GD(lamda=.01,itr=4000, learning_rate=.001)
model_Ridgegd.fit(train_x,train_y)

# predicting the test_y
model_Ridgegd.predict(test_x)

# getting the test error
model_Ridgegd.loss(test_x,test_y)

end_time = datetime.now()
```

```
For Iteration 0 the Loss is 255.5171.
For Iteration 1 the Loss is 174.2093.
For Iteration 2 the Loss is 118.7978.
For Iteration 3 the Loss is 81.0347.
For Iteration 4 the Loss is 55.2989.
For Iteration 5 the Loss is 37.7596.
For Iteration 6 the Loss is 25.8063.
For Iteration 7 the Loss is 17.6598.
For Iteration 8 the Loss is 12.1076.
For Iteration 9 the Loss is 8.3234.
For Iteration 10 the Loss is 5.7442.
For Iteration 11 the Loss is 3.9861.
For Iteration 12 the Loss is 2.7876.
For Iteration 13 the Loss is 1.9705.
For Iteration 14 the Loss is 1.4133.
For Iteration 15 the Loss is 1.0332.
For Iteration 16 the Loss is 0.7738.
For Iteration 17 the Loss is 0.5967.
For Iteration 18 the Loss is 0.4757.
For Iteration 19 the Loss is 0.3929
```

In [104]:
```
# time taken in fitting the model

print('Time taken to fit the model: {}'.format(end_time - start_time))
```

```
Time taken to fit the model: 0:04:50.940519
```

In [105]:

```
# getting the train error
model_Ridgegd.loss(train_x,train_y)
```

Out[105]: 0.021059718296063495

In [106]:

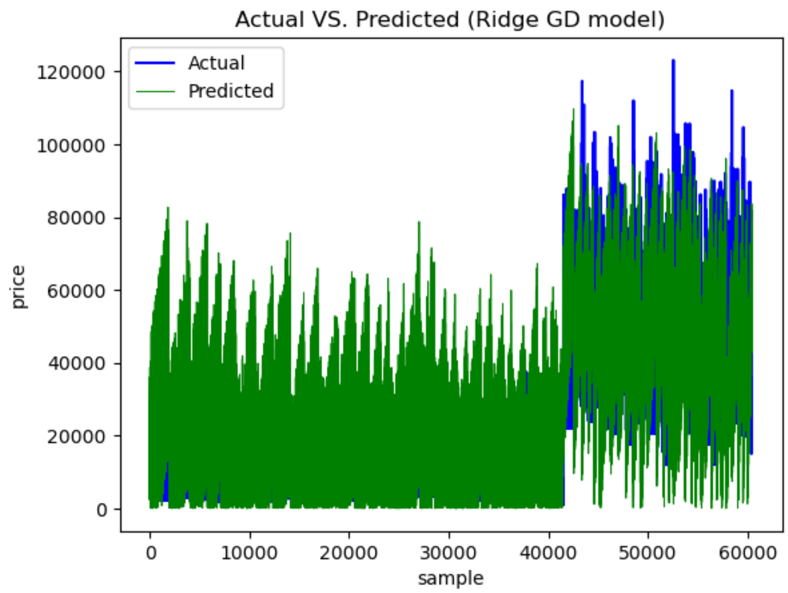
```
# getting the train error
model_Ridgegd.loss(test_x,test_y)
```

Out[106]: 0.020927402577616338

https://stackoverflow.com/questions/1557571/how-do-i-get-time-of-a-python-programs-execution (https://stackoverflow.com/questions/1557571/how-do-i-get-time-of-a-python-programs-execution)

In [107]:

```
import matplotlib.pyplot as plt

plt.plot(np.array(denorm(test_y)),label = "Actual",c = "b")
plt.plot(np.abs(denorm(model_Ridgegd.predict(test_x))),label = "Predicted",c = "g",linewidth=.7)
plt.xlabel("sample")
plt.ylabel("price")
plt.title("Actual VS. Predicted (Ridge GD model)")
plt.legend()
plt.show()
```



In [39]:

```
# saving the Ridge regression Using Gradient Descent model as the pickle file

import pickle
with open('Hasan_Hussain_assignment1_bonus_RidgeGD', 'wb') as files:
    pickle.dump(model_Ridgegd, files)
```

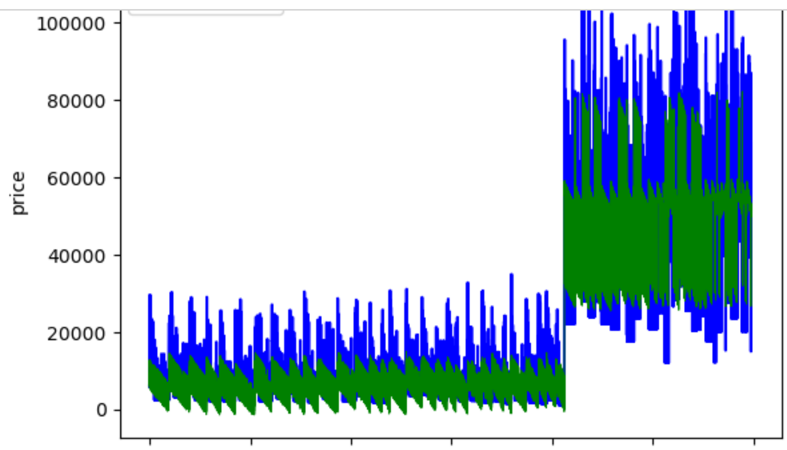
Elastic Net Regularization from Scratch

In [40]:
```
class Elastic:

    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))) + ((self.alpha/2)*(np.s
        return mse
```

In [116]:
```
elastic_loss={}
for i in [.000001,.00001,.001,.1,1]:
    model_Elastic=Elastic(alpha=i)
    model_Elastic.ols(train_x,train_y)
    model_Elastic.predict(test_x)
    elastic_loss[i]=model_Elastic.ols_los(test_x,test_y)
print(f"Best MSE is {min(elastic_loss.values())} for lambda value {min(elastic_loss.keys())}")
```

Best MSE is 0.0012868250723234856 for lambda value 1e-06

In [49]:
```
plt.plot(np.array(denorm(test_y)),label = "Actual",c = "b")
plt.plot(denorm(model_Elastic.predict(test_x)),label = "Predicted",c = "g",linewidth=.7)
plt.xlabel("sample")
plt.ylabel("price")
plt.title("Actual VS. Predicted (Elastic model)")
plt.legend()
plt.show()
```



In [117]:
```
# saving the Elastic Net regression model as the pickle file

import pickle
with open('Hasan_Hussain_assignment1_bonus_elastic', 'wb') as files:
    pickle.dump(model_Elastic, files)
```

Resouces

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

In [ ]: