Experiment 3

8 January

Download the dataset 'Book1.csv' from moodle. This dataset has information regarding the house price and many features depending on it. Open the CSV file and see the different features and the target variable Y (house price) also. ¶

In [36]:
```
## Importing the necessary libraries
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
##  Load the dataset to a dataframe.
df = pd.read_csv(r'C:\Users\Batch1\Documents\Downloads\TK\Book1.csv')
df
```

Out[36]:

	price	area	bedrooms	bathrooms	stories	parking	furnishingstatus
0	13300000	7420	4	2	3	2	furnished
1	12250000	8960	4	4	4	3	furnished
2	12250000	9960	3	2	2	2	semi-furnished
3	12215000	7500	4	2	2	3	furnished
4	11410000	7420	4	1	2	2	furnished
...	...	...	...	...	...	...	...
244	4550000	5320	3	1	2	0	semi-furnished
245	4550000	5360	3	1	2	2	unfurnished
246	4550000	3520	3	1	1	0	semi-furnished
247	4550000	8400	4	1	4	3	unfurnished
248	4543000	4100	2	2	1	0	semi-furnished

249 rows × 7 columns

Drop the furnishing status column and then we will be left out with 5 features (X1, X2, X3, X4, X5) to predict the house price (Y)

In [37]:
```
df.drop('furnishingstatus',axis = 1,inplace = True)
df
```

Out[37]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2
...	...	...	...	...	...	...
244	4550000	5320	3	1	2	0
245	4550000	5360	3	1	2	2
246	4550000	3520	3	1	1	0
247	4550000	8400	4	1	4	3
248	4543000	4100	2	2	1	0

249 rows × 6 columns

Use MinMaxScaler() to scale the data to 0 to 1 range.

In [38]:
```
MM = preprocessing.MinMaxScaler()
x = MM.fit_transform(df)
print(x)
```

```
[[1.00000000e+00 3.56776557e-01 5.00000000e-01 3.33333333e-01
  6.66666667e-01 6.66666667e-01]
 [8.80095923e-01 4.69597070e-01 5.00000000e-01 1.00000000e+00
  1.00000000e+00 1.00000000e+00]
 [8.80095923e-01 5.42857143e-01 2.50000000e-01 3.33333333e-01
  3.33333333e-01 6.66666667e-01]
 ...
 [7.99360512e-04 7.10622711e-02 2.50000000e-01 0.00000000e+00
  0.00000000e+00 0.00000000e+00]
 [7.99360512e-04 4.28571429e-01 5.00000000e-01 0.00000000e+00
  1.00000000e+00 1.00000000e+00]
 [0.00000000e+00 1.13553114e-01 0.00000000e+00 3.33333333e-01
  0.00000000e+00 0.00000000e+00]]
```

Split the data into training and testing sets using appropriate functions. Use a 80:20 split.

In [39]:
```
X = x[:,1:]
Y = x[:,0]
```

In [40]:
```
X.shape
```

Out[40]:
```
(249, 5)
```

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

Out[41]:
```
(249,)
```

In [46]:
```
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size = 0.2, randc
```

Also, use the inbuilt LinearRegression class and create an object of this class and fit the model using training data and check for the values of the parameters of your model. If you print the intercept and coefficients as you did in the previous lab you will get the model parameters θ0, θ1, ..., θ5

In [47]:
```
model = LinearRegression()
model.fit(x_train,y_train)
print("optimized m:",model.coef_)
print("optimized n:",model.intercept_)
```

```
optimized m: [0.32033532 0.09007501 0.3199135  0.13202285 0.14958391]
optimized n: -0.06121183786718576
```

In [48]:
```
Y_pred = model.predict(x_test)

# Evaluating the model performance using Mean Squared Error
mse = mean_squared_error(y_test, Y_pred)
print(f"Mean Squared Error on test data: {mse}")
```

```
Mean Squared Error on test data: 0.020077937566470735
```

Q2

Gradient descent algorithm for multiple regression

In [49]:
```
def gd(X, Y, m, n, L):
    Dm = np.zeros(X.shape[1])
    Dn = 0
    m_len = len(X)

    for i in range(m_len):
        prediction_error = np.dot(X[i], m) + n - Y[i]
        Dm += (2 / m_len) * prediction_error * X[i]
        Dn += (2 / m_len) * prediction_error
    m = m - L * Dm
    n = n - L * Dn
    return m, n
m = np.zeros(x_train.shape[1])
n = 0
L = 0.3
epochs = 1000

for epoch in range(epochs):
    m, n = gd(x_train, y_train, m, n, L)

print(f"Optimized m: {m}")
print(f"Optimized n (intercept): {n}")

# Making predictions using the learned parameters on the test data
Y_pred = np.dot(x_test, m) + n

# Evaluating the model performance using Mean Squared Error
mse = mean_squared_error(y_test, Y_pred)
print(f"Mean Squared Error on test data: {mse}")
```

```
Optimized m: [0.32033458 0.09007431 0.31991389 0.13202283 0.14958395]
Optimized n (intercept): -0.06121148456591009
Mean Squared Error on test data: 0.020077925354191114
```

Alternate approach

In [50]:
```
z = np.ones(len(x_train))
new_X_train = np.concatenate((np.array(z)[:, np.newaxis], x_train), axis=1)
print(new_X_train)
```

```
[[1.         0.05054945 0.5        0.         0.66666667 0.33333333]
 [1.         0.15018315 0.25       0.33333333 0.33333333 0.66666667]
 [1.         0.27472527 0.25       0.         0.         0.66666667]
 ...
 [1.         0.08424908 0.5        0.         0.33333333 0.        ]
 [1.         0.2967033  0.25       0.         1.         1.        ]
 [1.         0.42857143 0.25       0.         0.33333333 0.66666667]]
```

In [51]:
```
def gd(data,yt,parameters,lrate):
    slopes = np.zeros(6)
    for j in range(len(data)):
        for k in range(6):
            slopes[k]+= (1/len(data))*((data[j]*parameters).sum()-yt[j])*da
    parameters = parameters-lrate*slopes
    return parameters
parameters = np.zeros(6)
lrate = 0.9
iter_value = 1500
for i in range(iter_value):
    parameters = gd(new_X_train,y_train,parameters,lrate)
print(parameters)
```

```
[-0.06121184  0.32033532  0.09007501  0.3199135   0.13202285  0.14958391]
```

Q3

Implement the Linear regression problem what we attempted using the housepricedata set in the last lab using stochastic gradient descent and mini batch gradient descent and present the results and plots of your model. Also find the testing error in both the cases. (Use the learning rate to be 0.5 and Hint: you can use random module and random.sample function can be used to generate say 30 (a mini batch) of indices and the derivative can be calculated for only those data points and summed. which can be used to update the model parameters values.

In [142]:
```
## loading the dataset
data = pd.read_csv(r'C:\Users\Batch1\Documents\Downloads\TK\Training_set_he
data
```

Out[142]:

	Height	Weight
0	127.8296	67.63371
1	123.4114	65.95421
2	134.4043	66.14316
3	155.9981	73.45251
4	136.1354	69.30943
...	...	...
194	135.2500	68.41222
195	109.5143	66.49607
196	139.6043	67.84894
197	134.3672	67.27839
198	130.3869	68.48742

199 rows × 2 columns

In [143]:
```
#  Use MinMaxScaler() to scale the data to 0 to 1 range.
MM = preprocessing.MinMaxScaler()
x = MM.fit_transform(data)
```

In [144]:
```
X = x[:,0]
Y = x[:,1]
```

In [145]:
```
# splitting the data into train and test

x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size = 0.2,random
```

In [147]:
```
# Creating linear regression model and fitting the data
reg = LinearRegression()
reg.fit(x_train.reshape(-1,1),y_train)
m = reg.intercept_
n = reg.coef_
print(m,n)
# Making predictions using the learned parameters on the test data
Y_pred = reg.predict(x_test.reshape(-1,1))

# Evaluating the model performance using Mean Squared Error
mse = mean_squared_error(y_test, Y_pred)
print(f"Mean Squared Error on test data: {mse}")
```

```
0.3187097588547204 [0.42917335]
Mean Squared Error on test data: 0.028299747771813315
```

In [98]:
```
def gd(xt,yt,mb,nb,L):
    Dm=0
    Dn=0
    for i in range(len(xt)):
        Dm=Dm+(2/len(xt))*((mb*xt[i]+nb-yt[i])*xt[i])
        Dn=Dn+(2/len(xt))*(mb*xt[i]+nb-yt[i])
    mb=mb-L*Dm
    nb=nb-L*Dn
    return mb,nb
m=0
n=0
L=0.5
epochs=900
for i in range(epochs):
    m,n=gd(x_train,y_train,m,n,L)
print(m,n)
```

```
0.42917326798919886 0.3187098016324925
```

Stochastic gradient descent

In [157]:
```
def sgd(x,y,m,n,L):
    ## randomly shuffling the data
    i = np.random.randint(len(x))
    dm = 2*(m*x[i]+n - y[i])*x[i]
    dn = 2*(m*x[i]+n - y[i])
    m -= L*dm
    n -= L*dn
    return m,n

m = 0
n = 0
L = 0.4
epochs = 1200

for epoch in range(epochs):
    m, n = sgd(x_train, y_train, m, n, L)

print(f"Optimized m: {m}")
print(f"Optimized n (intercept): {n}")

# Making predictions using the learned parameters on the test data
Y_pred = x_test*m + n

# Evaluating the model performance using Mean Squared Error
mse = mean_squared_error(y_test, Y_pred)
print(f"Mean Squared Error on test data: {mse}")
```

```
Optimized m: 0.4688999790461973
Optimized n (intercept): 0.6935169644013384
Mean Squared Error on test data: 0.16480703058510277
```

In [ ]: