Student Name:Dhaniswar B.K. And StudentId:NP03A190318

USE of Multivariate Regression on DIABETES Dataset

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

importing pandas library ,numpy and pyplot library

In [2]:
```
Data = pd.read_csv("diabetes.csv")
Data
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

reading csv file using pandas and converting header to numeric form to apply multivariate linear Regression

In [3]:
```
Data = pd.read_csv("diabetes.csv",header=None)
Data
```

Out[3]:

	0	1	2	3	4	5	6	7	8
0	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
...	...	...	...	...	...	...	...	...	...
764	10	101	76	48	180	32.9	0.171	63	0
765	2	122	70	27	0	36.8	0.34	27	0
766	5	121	72	23	112	26.2	0.245	30	0
767	1	126	60	0	0	30.1	0.349	47	1
768	1	93	70	31	0	30.4	0.315	23	0

769 rows × 9 columns

#loading data from pandas and making header none and gisplaying data

In [4]:
```
Data[0] = Data[0].replace("Pregnancies",1)
Data[1] = Data[1].replace("Glucose",1)
Data[2] = Data[2].replace("BloodPressure",1)
Data[3] = Data[3].replace("SkinThickness",1)
Data[4] = Data[4].replace("Insulin",1)
Data[5] = Data[5].replace("BMI",1)
Data[6] = Data[6].replace("DiabetesPedigreeFunction",1)
Data[7] = Data[7].replace("Age",1)
Data[8] = Data[8].replace("Outcome",1)
Data
```

Out[4]:

	0	1	2	3	4	5	6	7	8
0	1	1	1	1	1	1	1	1	1

	0	1	2	3	4	5	6	7	8
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
...	...	...	...	...	...	...	...	...	...
764	10	101	76	48	180	32.9	0.171	63	0
765	2	122	70	27	0	36.8	0.34	27	0
766	5	121	72	23	112	26.2	0.245	30	0
767	1	126	60	0	0	30.1	0.349	47	1
768	1	93	70	31	0	30.4	0.315	23	0

769 rows × 9 columns

replacing string value with numeric value 0 using replace function

In [5]:

```
X = Data.drop(columns=8).astype('float64')
X
```

Out[5]:

	0	1	2	3	4	5	6	7
0	1.0	1.0	1.0	1.0	1.0	1.0	1.000	1.0
1	6.0	148.0	72.0	35.0	0.0	33.6	0.627	50.0
2	1.0	85.0	66.0	29.0	0.0	26.6	0.351	31.0
3	8.0	183.0	64.0	0.0	0.0	23.3	0.672	32.0
4	1.0	89.0	66.0	23.0	94.0	28.1	0.167	21.0
...	...	...	...	...	...	...	...	...
764	10.0	101.0	76.0	48.0	180.0	32.9	0.171	63.0
765	2.0	122.0	70.0	27.0	0.0	36.8	0.340	27.0
766	5.0	121.0	72.0	23.0	112.0	26.2	0.245	30.0

	0	1	2	3	4	5	6	7
767	1.0	126.0	60.0	0.0	0.0	30.1	0.349	47.0
768	1.0	93.0	70.0	31.0	0.0	30.4	0.315	23.0

769 rows × 8 columns

dividing input data using drop function and repacing string value with float using astype function

In [6]:
```
y = Data.iloc[:,8:].astype('float64')
y
```

Out[6]:

	8
0	1.0
1	1.0
2	0.0
3	1.0
4	0.0
...	...
764	0.0
765	0.0
766	0.0
767	1.0
768	0.0

769 rows × 1 columns

dividing output data using drop function and repacing string value with float using astype function

In [7]:
```
theta = np.array([0]*len(X.columns))
```

initializing theta value initially 0 using numpy dot array

In [8]:
```
m = len(Data)
```

```
m
```

Out[8]: 769

#length of training set

In [9]:
```
def hypothesis(theta, X):
    return theta*X
```

Creating hypotheis function it is a hypothesis function it gives hypothetic value which is theta*X

In [10]:
```
def computeCost(X,y, theta):
    y1 = hypothesis(theta,X)
    y1 = np.sum(y1, axis = 1)
    return sum(np.sqrt((y1-y)**2))/(2*m)
```

computing costfunction where y1 is hypothesis value and y is actual value And applying RMSE

In [11]:
```
def gradientDescent(X,y,theta,alpha,i):
    J = [] #cost function in each iterations
    k = 0
    while k < i:
        y1 = hypothesis(theta, X)
        y1 = np.sum(y1,axis=1)
        for C in range(0, len(X.columns)):
            theta[C] = theta[C] - alpha*(sum((y1-y)*X.iloc[:,C])/len(X))
        j = computeCost(X, y, theta)
        J.append(j)
        k += 1
    return J, j, theta
```

optimizing bias tearm and it gives optimize theta by differentiation multivariate is used for output in iteration

In [16]:
```
J, j, theta = gradientDescent(X, y, theta, 0.5, 1000)
```

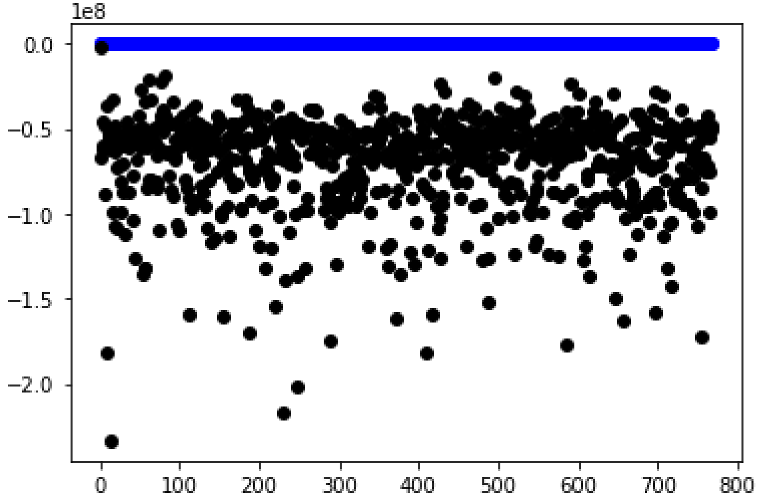
Call gradient descent where 0.5 is value of alpha and 1000 is value of iteration our target is difference between predicted and hypothetical value should be nearly equal to zero

In [17]:
```
y_hat = hypothesis(theta, X)
y_hat = np.sum(y_hat, axis=1)
```

call hypothesis it means predict output and theta = optimize theta and X input where y_hat is predicted data

In [18]:

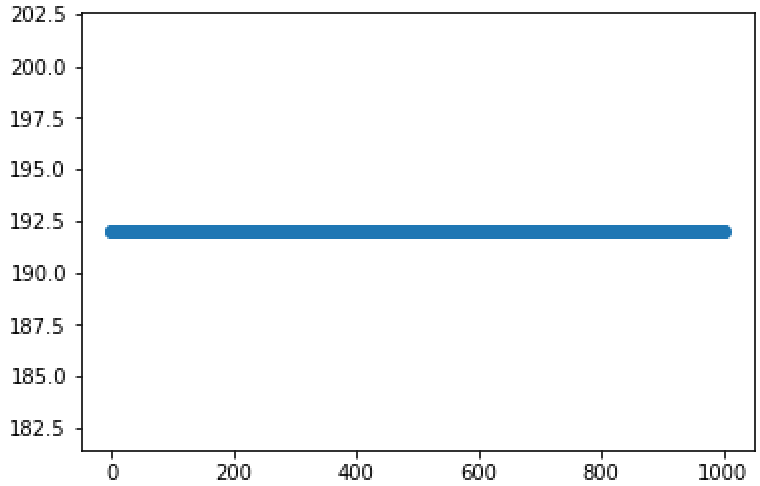
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure()
plt.scatter(x=list(range(0, 769)),y= y, color="blue")
plt.scatter(x=list(range(0, 769)),y= y_hat, color="black")
plt.show()
```



showing the distane between y and y_hat it meand distance between original data and predicted data where black data is predicted and blue data is origional

In [20]:

```
plt.figure()
plt.scatter(x=list(range(0, 1000)), y=J)
plt.show()
```



platting the cost function where J is a cost function

In [64]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.01, random_state = 95)
```

importing linear regression And importing train_test_split from sklearn to split data into trannig and testing set

In [65]:
```
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

creating Linear Regression, Run the logistic Regression And calculating the predicted value by model

In [66]:
```
model.fit(X_train, y_train)
```

Out[66]:
```
LinearRegression()
```

In [67]:
```
model.score(X_test,y_test)
```

Out[67]:
```
0.7119361973183576
```

Measuring the total Accuracy of the module

In [68]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

importing mean_squared_error and mean_absolute_error to claculate MSE, MAE and RMSE

In [69]:

```
# np.mean((y_test - predictions)**2)
mse = mean_squared_error( y_test, predictions) # calculating MSE
mse
```

Out[69]: 0.07201595067041063

In [70]:

```
mae = mean_absolute_error(y_test, predictions) # calculating MAE
mae
```

Out[70]: 0.19570865608559376

In [71]:

```
rmse = np.sqrt(mse)
```

In [72]:

```
rmse #calculating MAE
```

Out[72]: 0.2683578779734454