05
NOV
2020

ASSIGNMENT 1

APPLYING GRADIENT DESCENT ALGORITHM TO LINEAR REGRESSION
PROBLEM WITH VARIOUS LEARNING RATES (STEP SIZE) AND STUDYING ITS EFFECTS

SUBJECT : DEEP LEARNING [IT702]

SHANKARANARAYAN N | 203IT001 | M.TECH(RESEARCH) IT

ASSIGNMENT 1
SUBJECT: IT702 - DEEP LEARNING

AIM:

- TASK1: To apply linear regression with gradient descent to predict the insurance rate.
- TASK2: To visualize the dataset
- TASK3: To exploring how different learning rate values affect the gradient descent

NOTATIONS:

- X : Denoted the training data
- y : Denotes the expected result
- theta : Denotes the unknown variables (*Our aim is to find an optimum theta*)
- Predicted : Predicted value
- J[a] : Change in cost function stores as a list item

ENVIRONMENT:

- The entire assignment is executed in Google Colab environment.
- The dataset is stored and retrieved from Google Drive.

DATASET:

- **SOURCE :** Kaggle
- **COLUMNS:**
    - LIFE EXPECTANCY : Denotes the average life expectancy of the person
    - AGE : Age of the person
    - SEX : Gender (1-Female | 0-Male)
    - BMI : Body mass index of the person
    - CHILDREN : Number of children
    - SMOKER : Smoker or not (1-Yes | 0-No)
    - REGION : Location (Here only 4 specific locations were takes with 0-3 donating each of them)
    - BMI : Average BMI for last 10 years
    - CHARGES : Insurance charge (Data to be predicted)

-THE TRAINING DATA CONTAINS A TOTAL OF 1336 DATASETS-

DATASET DESCCRIPTION:

```
data = pd.read_csv('data4.csv')
data.head()
```

	lifeexp	age	sex	bmi	children	smoker	region	BMI	charges
0	36.3	35	1	35.860	2	0	1	44.2	5836.52040
1	45.3	28	1	25.935	1	0	2	15.4	4133.64165
2	45.3	40	0	29.900	2	0	0	12.6	6600.36100
3	45.6	18	1	33.880	0	0	1	17.2	11482.63485
4	45.6	19	1	30.590	2	0	2	16.8	24059.68019

```
data.describe()
```

	lifeexp	age	sex	bmi	children	smoker	region	BMI	charges
count	1336.000000	1336.000000	1336.00000	1336.000000	1336.000000	1336.000000	1336.000000	1336.000000	1336.000000
mean	69.750225	39.193114	0.49476	30.673177	1.095060	0.205090	1.483533	38.067515	13276.019562
std	9.665135	14.049996	0.50016	6.094841	1.205769	0.403918	1.105532	19.474377	12117.796317
min	36.300000	18.000000	0.00000	15.960000	0.000000	0.000000	0.000000	1.400000	1121.873900
25%	62.875000	26.750000	0.00000	26.308750	0.000000	0.000000	1.000000	19.400000	4733.635288
50%	72.650000	39.000000	0.00000	30.400000	1.000000	0.000000	1.000000	44.400000	9382.033000
75%	76.800000	51.000000	1.00000	34.700000	2.000000	0.000000	2.000000	55.900000	16687.364100
max	89.000000	64.000000	1.00000	53.130000	5.000000	1.000000	3.000000	67.000000	63770.428010

OBSERVATIONS : TASK 1
[PREDICTION]

- The following is the result of applying linear regression with gradient descent to predict the insurance charges/rate.

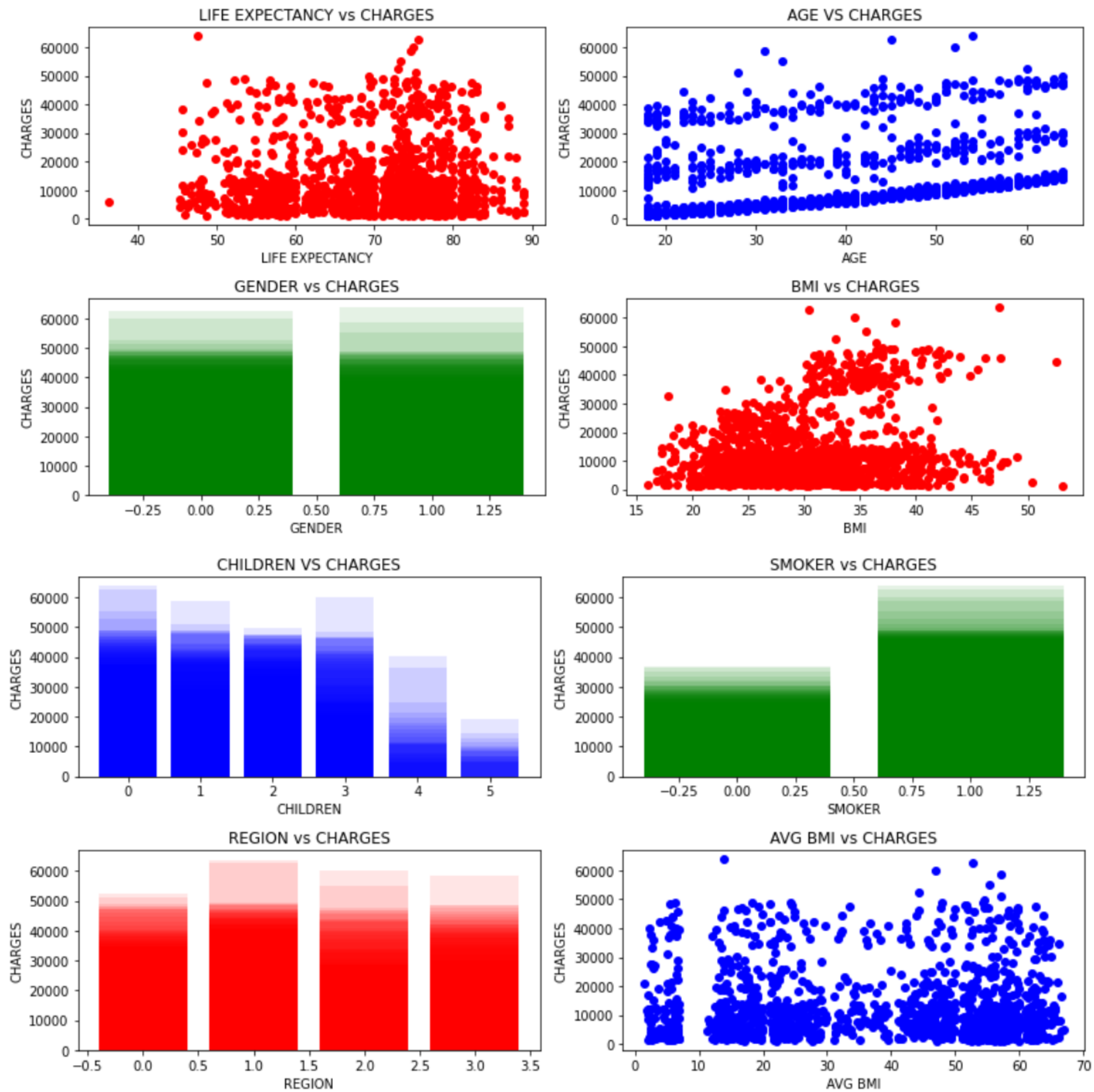
PREDICTION WITH LINEAR REGRESSION

```
#PREDICTION
p=[100,15,1,20,0,1,2,25] #Dummy input
pdt=np.dot(p,theta) #Predicted result
print(pdt) #Printing
```

```
[99293.23409259]
```

OBSERVATIONS : TASK 2
[DATA VISUALIZATION]

- The dataset is visualized using MATPLOTLIB
- As the data is multidimensional a single graph cannot represent it.



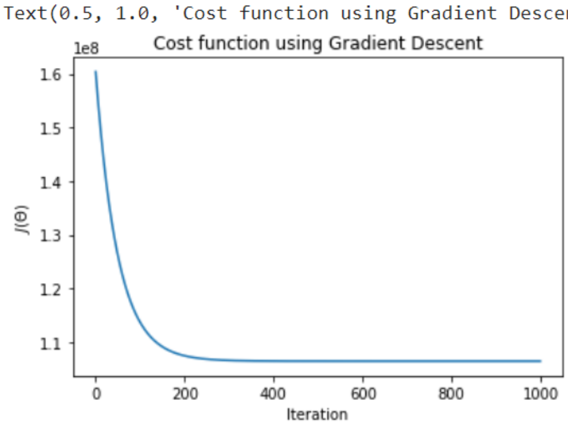
OBSERVATIONS : TASK 3
[EXPLORING HOW DIFFERENT LEARNING RATE VALUES AFFECT THE GRADIENT DESCENT]

- To explore this we have taken 4 cases
  - **CASE 1 :** THETA REACHES OPTIMUM VALUE
    [LEARNING RATE=0.01] [ITERATIONS=1000]
  - **CASE 2 :** THETA REACHES OPTIMUM VALUE AFTER A LONG TIME
    [LEARNING RATE=0.01] [ITERATIONS=5000]
  - **CASE 3 :** THETA REACHES OPTIMUM VALUE VERY SOON
    [LEARNING RATE=0.1] [ITERATIONS=1000]
  - **CASE 4 :** ABNORMAL REACTION
    [LEARNING RATE=10] [ITERATIONS=1000]

CASE 1 : THETA REACHES OPTIMUM VALUE [LEARNING RATE=0.01] [ITERATIONS=1000]

- When the learning rate and iteration are in optimum setting the cost function decreases gradually.
- It can be noted that the cost function J[] decreases steadily with every iteration and after a certain iteration x it remains the same.

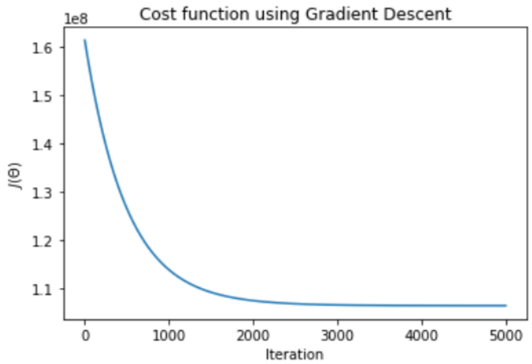
```
THETA [[ -39.16710942]
 [3611.20379199]
 [  59.17403198]
 [2024.52614773]
 [ 576.70026648]
 [9615.65967567]
 [ 389.46461201]
 [ -63.28204861]]

J[]= [160384142.38395026, 159298933.32161537, 158235794.299934, 157194270.50385696, 156173916.63942057, 155174296.7306091, 1541

Text(0.5, 1.0, 'Cost function using Gradient Descent')
```



CASE 2 : THETA REACHES OPTIMUM VALUE AFTER A LONG TIME **[LEARNING RATE=0.001] [ITERATIONS=5000]**

- When the learning rate is too small it take more iteration to reach the global minimum
- It can be observed that the difference between successive elements of J[] is very small.

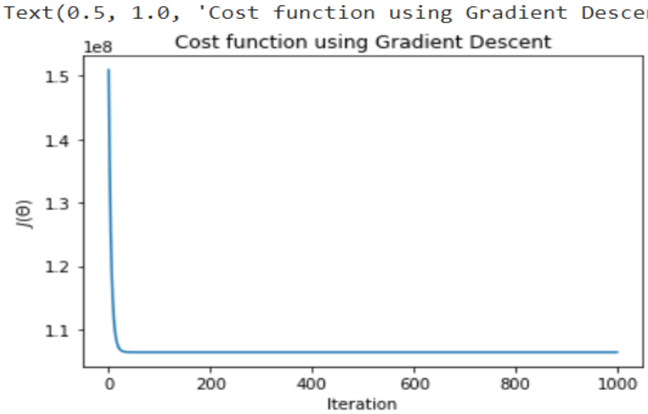
```
THETA [[ -21.91262189]
 [3585.81983011]
 [  32.19785611]
 [2016.28734457]
 [ 579.85312132]
 [9542.44930417]
 [ 379.24975278]
 [ -88.94323933]]

J[]= [161380599.34235644, 161269540.11747414, 161158707.88488817, 161048102.17437646, 160937722.51670656, 160827568

Text(0.5, 1.0, 'Cost function using Gradient Descent')
```



CASE 3 : THETA REACHES OPTIMUM VALUE VERY SOON **[LEARNING RATE=0.1] [ITERATIONS=1000]**

- When the learning rate is big it take very less iteration to reach the global minimum
- It can be observed that the difference between successive elements of J[] is very high.

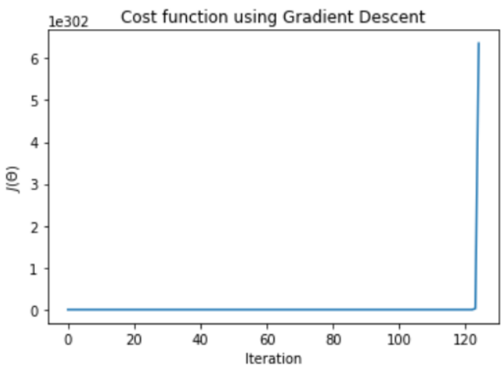
```
THETA [[ -43.53649201]
 [3611.10273984]
 [  59.76033399]
 [2024.74718401]
 [ 576.62880557]
 [9616.49569033]
 [ 389.78010869]
 [ -58.81670903]]

J[]= [150926773.27562746, 142408887.99302202, 135536895.24444416, 129989212.30326207, 125507903.

Text(0.5, 1.0, 'Cost function using Gradient Descent')
```



CASE 4 : ABNORMAL REACTION **[LEARNING RATE=10] [ITERATIONS=1000]**

- When the learning rate is too big we can observe a bounce in the constfunction values.
- It can be observed that the difference between successive elements of J[] is bouncing between higher and lower values.

```
THETA [[nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]
 [nan]]

J[]= [4740541452.063589, 417390277353.89526, 41555806711143.41, 4814009436265396.0, 6.80340643703186e+17, 1.1752515402670042e+2


/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:90: RuntimeWarning: overflow encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: RuntimeWarning: overflow encountered in square
  after removing the cwd from sys.path.
Text(0.5, 1.0, 'Cost function using Gradient Descent')
```



CONCLUSION : TASK 3

- Based on the above observations I conclude the following
  - High learning rate results in an anomaly where the global minimum is not reached.
  - Low learning rate take a lot of iteration to reach the global minimum

  So, we have to use an learning rate which requires an optimum/less amount of iterations to reach the global minimum.

LINK TO THE CODE :
https://colab.research.google.com/drive/1CebaQ6cV_9Ijz0EpAKKi6S0kx4GLIIxK?usp=sharing

ASS1_ipnb

November 5, 2020

1 *ASSIGNMENT 1*

SUBJECT : IT702 - DEEP LEARNING

SYBMITTED BY : SHANKARANARAYAN N, M.TECH(RESEARCH), DEPARTMENT OF INFORMATION TECHNOLOGY

- SOME LINES IN THE CODE ARE PURPOSELY OMITTED USING "#" TO MAKE THE PDF MORE APPEALING

---

2 *SOURCE CODE::*

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

THE FOLLOWING CODE IS USED TO CONNECT GOOGLE DRIVE AND GOOGLE COLAB

[ ]:
```
# Import PyDrive and associated libraries.
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

[ ]:
```
downloaded = drive.CreateFile({'id':'1-TZOOYbZEwKBIcJiPh4IbDjyymc27BMp'})
downloaded.GetContentFile('data4.csv')
```

DATA DESCTIPTION

[ ]:
```
data = pd.read_csv('data4.csv')
data.head()
```

[ ]:

	lifeexp	age	sex	bmi	children	smoker	region	BMI	charges
0	36.3	35	1	35.860	2	0	1	44.2	5836.52040
1	45.3	28	1	25.935	1	0	2	15.4	4133.64165
2	45.3	40	0	29.900	2	0	0	12.6	6600.36100
3	45.6	18	1	33.880	0	0	1	17.2	11482.63485
4	45.6	19	1	30.590	2	0	2	16.8	24059.68019

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

[ ]:

	lifeexp	age	…	BMI	charges
count	1336.000000	1336.000000	…	1336.000000	1336.000000
mean	69.750225	39.193114	…	38.067515	13276.019562
std	9.665135	14.049996	…	19.474377	12117.796317
min	36.300000	18.000000	…	1.400000	1121.873900
25%	62.875000	26.750000	…	19.400000	4733.635288
50%	72.650000	39.000000	…	44.400000	9382.033000
75%	76.800000	51.000000	…	55.900000	16687.364100
max	89.000000	64.000000	…	67.000000	63770.428010

[8 rows x 9 columns]

THE FOLLOWING CODE COMPUTES THE COST FUNCTION

[ ]:
```
def computeCost(X,y,theta):
    m=len(y)
    predictions=X.dot(theta)
    square_err=(predictions - y)**2
    return 1/(2*m) * np.sum(square_err)
```

THE FOLLOWING CODE COMPUTES THE GRADIENT DESCENT

[ ]:
```
def gradientDescent(X,y,theta,alpha,num_iters):
    m=len(y)
    J=[]
    for i in range(num_iters):
        predictions = X.dot(theta)
        error = np.dot(X.transpose(),(predictions -y))
        #error = np.sum(predictions -y)
        descent=alpha * 1/m * error
```

```
        theta-=descent
        J.append(computeCost(X,y,theta))
    return theta, J
```

THE FOLLOWING CODE COMPUTES FEATURE NORMALISATION

[ ]:
```
def featureNormalization(X):
    mean=np.mean(X,axis=0) #mean
    std=np.std(X,axis=0) #standard deviation
    X_norm = (X - mean)/std #notmalizing X
    return X_norm , mean , std
```

THE VALUE OF X AND Y IS INITIALIZED HERE

[ ]:
```
data_n2=data.values
m2=len(data_n2[:,-1])
X=data_n2[:,0:8].reshape(m2,8)
print("\t\nX BEFORE NORMALISATION\n",X)
X, mean_X, std_X = featureNormalization(X)
#X = np.append(np.ones((m2,1)),X,axis=1)
y=data_n2[:,-1].reshape(m2,1)
print("\t\nY VALUE\n",y)
#theta=np.zeros((9,1))
print("\t\nX AFTER NORMALISATION\n",X)
```

```
X BEFORE NORMALISATION
 [[36.3 35.   1.  …  0.   1.  44.2]
 [45.3 28.   1.  …  0.   2.  15.4]
 [45.3 40.   0.  …  0.   0.  12.6]
 …
 [89.  18.   0.  …  0.   1.  58.6]
 [89.  39.   1.  …  0.   3.  61.9]
 [89.  46.   0.  …  0.   2.  57.6]]

Y VALUE
 [[5836.5204 ]
 [4133.64165]
 [6600.361  ]
 …
 [2304.0022 ]
 [7986.47525]
 [9301.89355]]

X AFTER NORMALISATION
```

```
[[-3.46221247 -0.29855411  1.01053453 … -0.50794071 -0.43753965
   0.3150181 ]
 [-2.53068174 -0.79696146  1.01053453 … -0.50794071  0.46734112
  -1.16440195]
 [-2.53068174  0.05745115 -0.98957529 … -0.50794071 -1.34242043
  -1.30823445]
 …
 [ 1.9924175  -1.50897198 -0.98957529 … -0.50794071 -0.43753965
   1.05472813]
 [ 1.9924175  -0.0137499   1.01053453 … -0.50794071  1.37222189
   1.22424501]
 [ 1.9924175   0.48465745 -0.98957529 … -0.50794071  0.46734112
   1.00335937]]
```

VISUALIZING DATASET

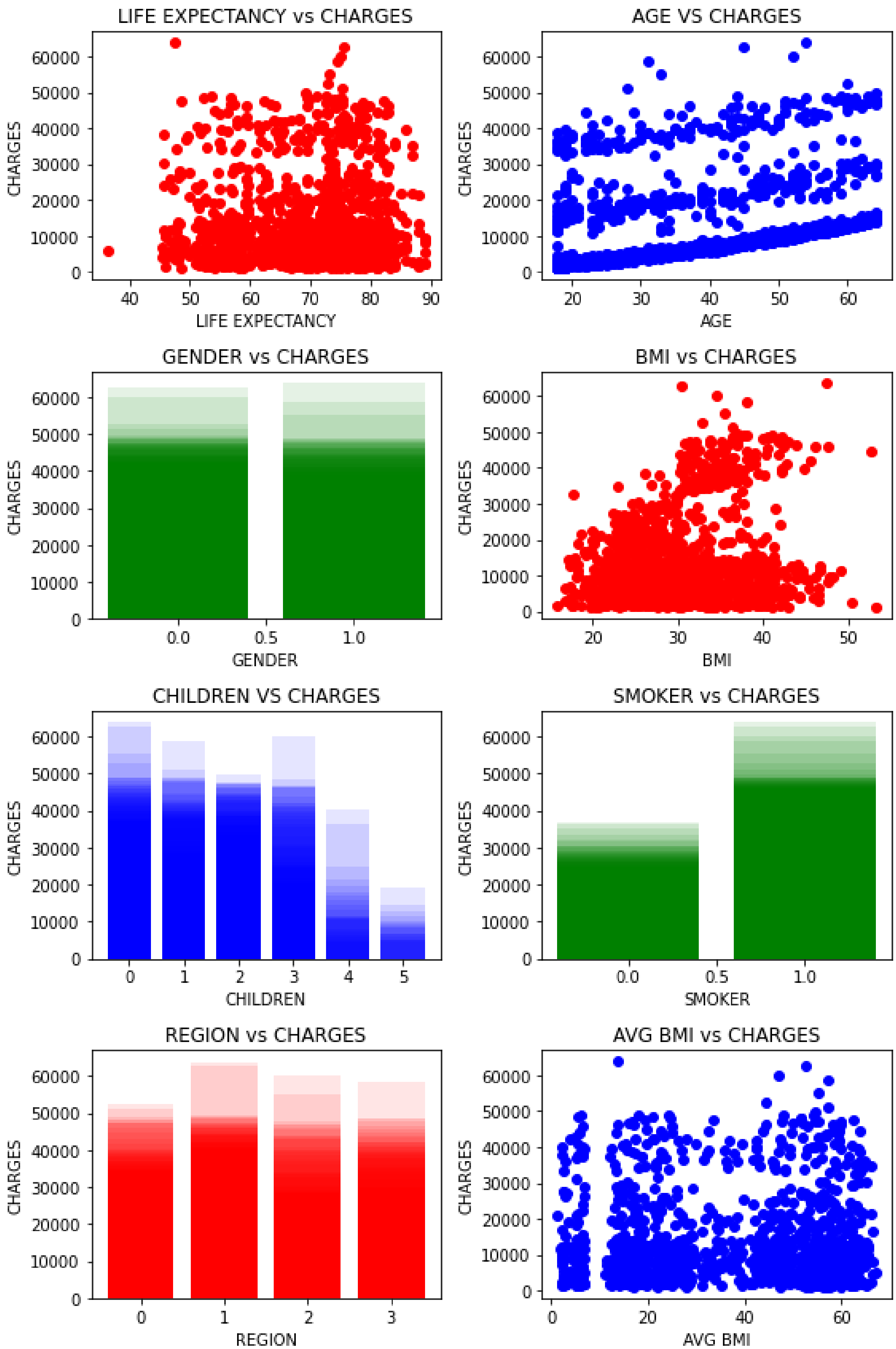
SPLITTING THE COLUMNS

[ ]:
```
a=data_n2[:,0].reshape(m2,1)
b=data_n2[:,1].reshape(m2,1)
c=data_n2[:,2].reshape(m2,1)
d=data_n2[:,3].reshape(m2,1)
e=data_n2[:,4].reshape(m2,1)
f=data_n2[:,5].reshape(m2,1)
g=data_n2[:,6].reshape(m2,1)
h=data_n2[:,7].reshape(m2,1)

g1=list(map(int,g))
c1=list(map(int,c))
e1=list(map(int,e))
f1=list(map(int,f))
y1=list(map(int, y))
```

PLOTTING DATA

[ ]:
```
fig, axes = plt.subplots(figsize=(8,12),nrows=4,ncols=2)
axes[0,0].scatter(a,y,color="r")
axes[0,0].set_xlabel("LIFE EXPECTANCY")
axes[0,0].set_ylabel("CHARGES")
axes[0,0].set_title("LIFE EXPECTANCY vs CHARGES")

axes[0,1].scatter(b,y,color="b")
axes[0,1].set_xlabel("AGE")
axes[0,1].set_ylabel("CHARGES")
axes[0,1].set_title("AGE VS CHARGES")

axes[1,0].bar(c1,y1,align='center',alpha=0.1,color="g")
axes[1,0].set_xlabel("GENDER")
```

```
axes[1,0].set_ylabel("CHARGES")
axes[1,0].set_title("GENDER vs CHARGES")

axes[1,1].scatter(d,y,color="r")
axes[1,1].set_xlabel("BMI")
axes[1,1].set_ylabel("CHARGES")
axes[1,1].set_title("BMI vs CHARGES")

axes[2,0].bar(e1,y1,align='center',alpha=0.1,color="b")
axes[2,0].set_xlabel("CHILDREN")
axes[2,0].set_ylabel("CHARGES")
axes[2,0].set_title("CHILDREN VS CHARGES")

axes[2,1].bar(f1,y1,align='center',alpha=0.1,color="g")
axes[2,1].set_xlabel("SMOKER")
axes[2,1].set_ylabel("CHARGES")
axes[2,1].set_title("SMOKER vs CHARGES")


axes[3,0].bar(g1,y1,align='center',alpha=0.1,color="r")
axes[3,0].set_xlabel("REGION")
axes[3,0].set_ylabel("CHARGES")
axes[3,0].set_title("REGION vs CHARGES")

axes[3,1].scatter(h,y,color="b")
axes[3,1].set_xlabel("AVG BMI")
axes[3,1].set_ylabel("CHARGES")
axes[3,1].set_title("AVG BMI vs CHARGES")

plt.tight_layout()
```


LIFE EXPECTANCY vs CHARGES
CHARGES
LIFE EXPECTANCY
AGE VS CHARGES
CHARGES
AGE
GENDER vs CHARGES
CHARGES
GENDER
BMI vs CHARGES
CHARGES
BMI
CHILDREN VS CHARGES
CHARGES
CHILDREN
SMOKER vs CHARGES
CHARGES
SMOKER
REGION vs CHARGES
CHARGES
REGION
AVG BMI vs CHARGES
CHARGES
AVG BMI

#HERE I AM DISCUSSING 4 DIFFERENT CASES COMPARING THE LEARNING RATE AND ITERATIONS IN GRADIENT DESCENT

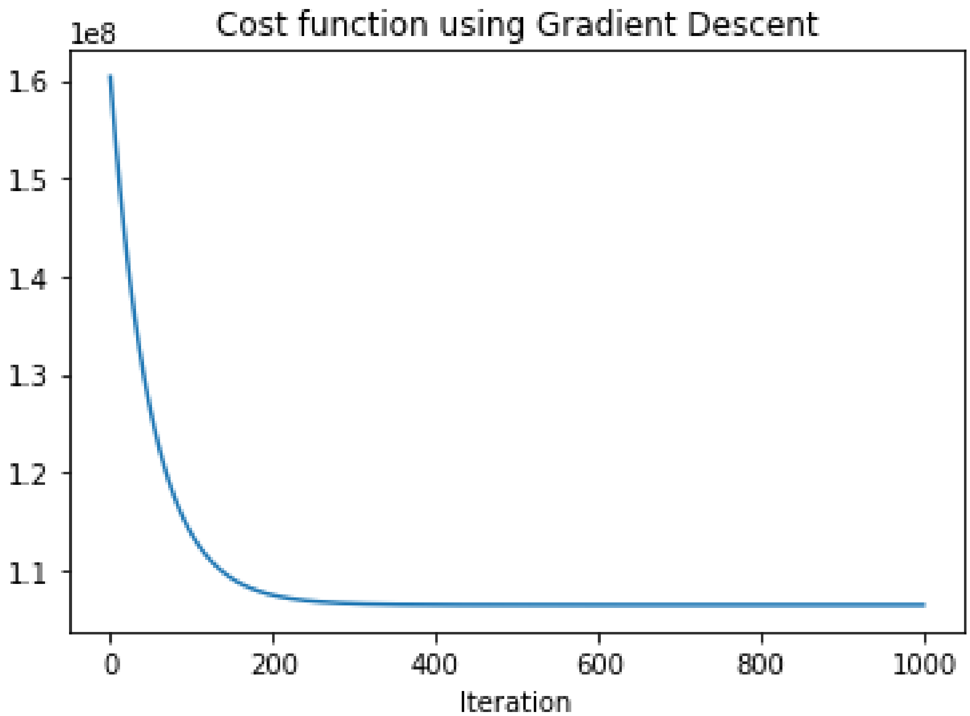
CASE 1 : THETA REACHES OPTIMUM VALUE [LEARNING RATE=0.01] [ITERATIONS=1000]

IN THIS CASE THE THETA REACHES ITS GLOBAL OPTIMUM WITH MINIMUM NUMBER OF STEPS AND ITERATION

- THETA is initialized to zero.
- Learningrate is initialized to 0.01
- Number of Iteration is initialized to 1000
- gradientDescent() function is invoked.

[141]:
```
#BEST CASE THETA REACHING OPTIMUM
theta=np.zeros((8,1))
theta, J = gradientDescent(X,y,theta,0.01,1000)
#print("THETA",theta)
#print("\nJ[]=",J)
print("\n")
plt.plot(J)
plt.xlabel("Iteration")
#plt.ylabel("$J(\Theta)$")
plt.title("Cost function using Gradient Descent")
```

[141]:
```
Text(0.5, 1.0, 'Cost function using Gradient Descent')
```

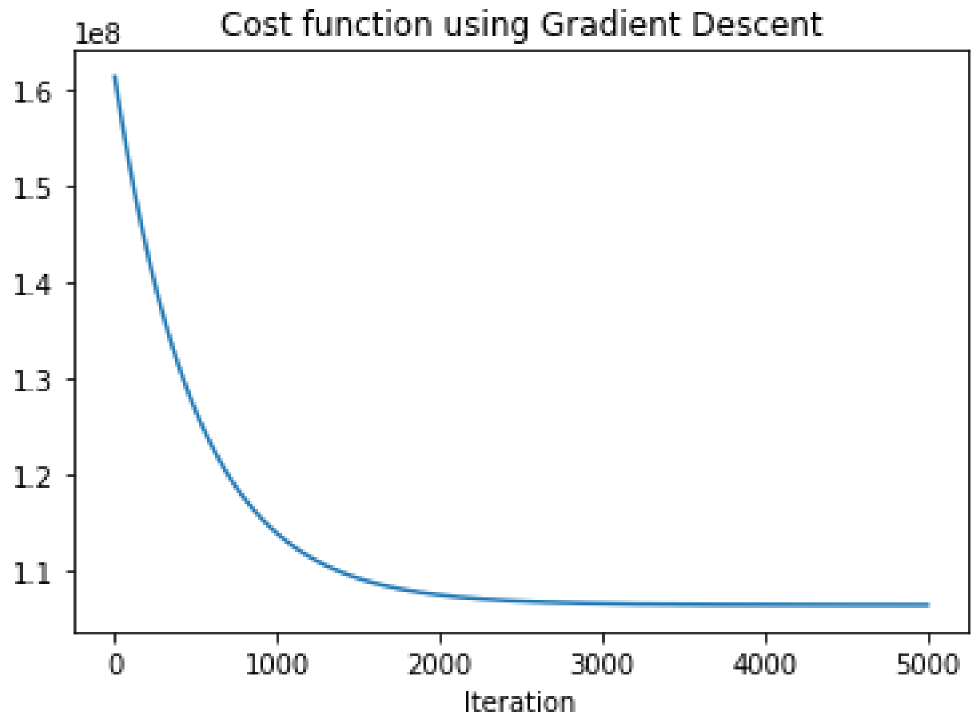


CASE 2 : THETA REACHES OPTIMUM VALUE AFTER A LONG TIME [LEARNING RATE=0.01] [ITERATIONS=5000]

- THETA is initialized to zero.
- Learningrate is initialized to 0.001
- Number of Iteration is initialized to 5000
- gradientDescent() function is invoked.

[140]:
```
theta=np.zeros((8,1)) #RESETTING
theta, J = gradientDescent(X,y,theta,0.001,5000)
#print("THETA",theta)
#print("\nJ[]=",J)
print("\n")
plt.plot(J)
plt.xlabel("Iteration")
#plt.ylabel("$J(\Theta)$")
plt.title("Cost function using Gradient Descent")
```

[140]:
```
Text(0.5, 1.0, 'Cost function using Gradient Descent')
```

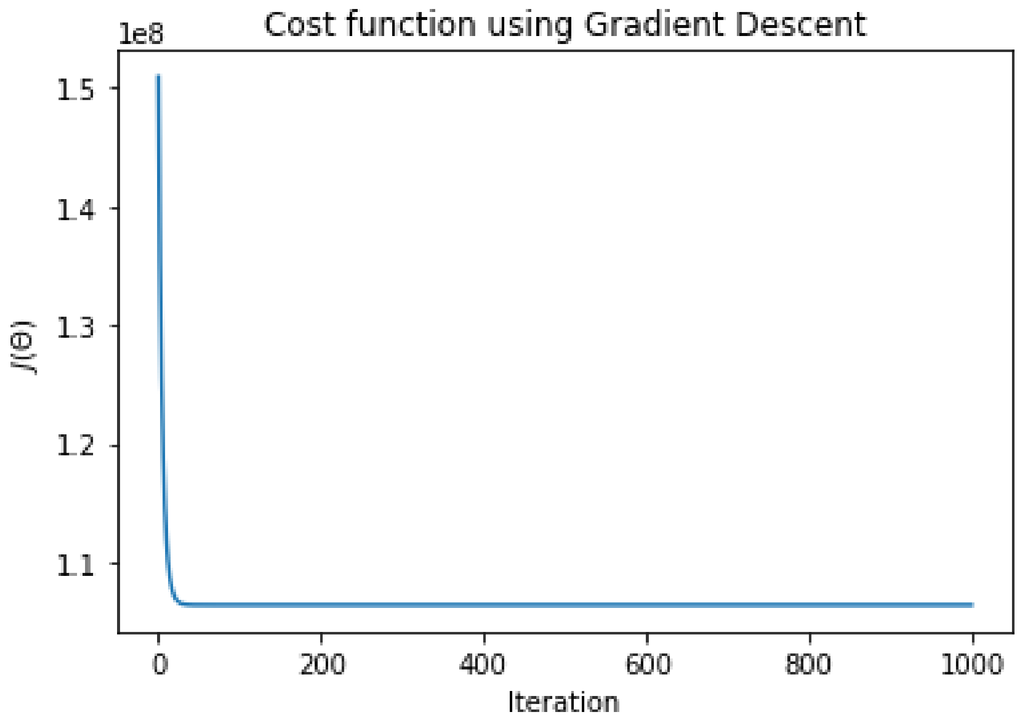


CASE 3 : THETA REACHES OPTIMUM VALUE VERY SOON [LEARNING RATE=0.1] [ITERATIONS=1000]

- THETA is initialized to zero.
- Learningrate is initialized to 0.1
- Number of Iteration is initialized to 1000
- gradientDescent() function is invoked.

[139]:
```
theta=np.zeros((8,1)) #RESETTING
theta, J = gradientDescent(X,y,theta,0.1,1000)
#print("THETA",theta)
#print("\nJ[]=",J)
print("\n")
plt.plot(J)
plt.xlabel("Iteration")
plt.ylabel("$J(\Theta)$")
plt.title("Cost function using Gradient Descent")
```

[139]:
```
Text(0.5, 1.0, 'Cost function using Gradient Descent')
```



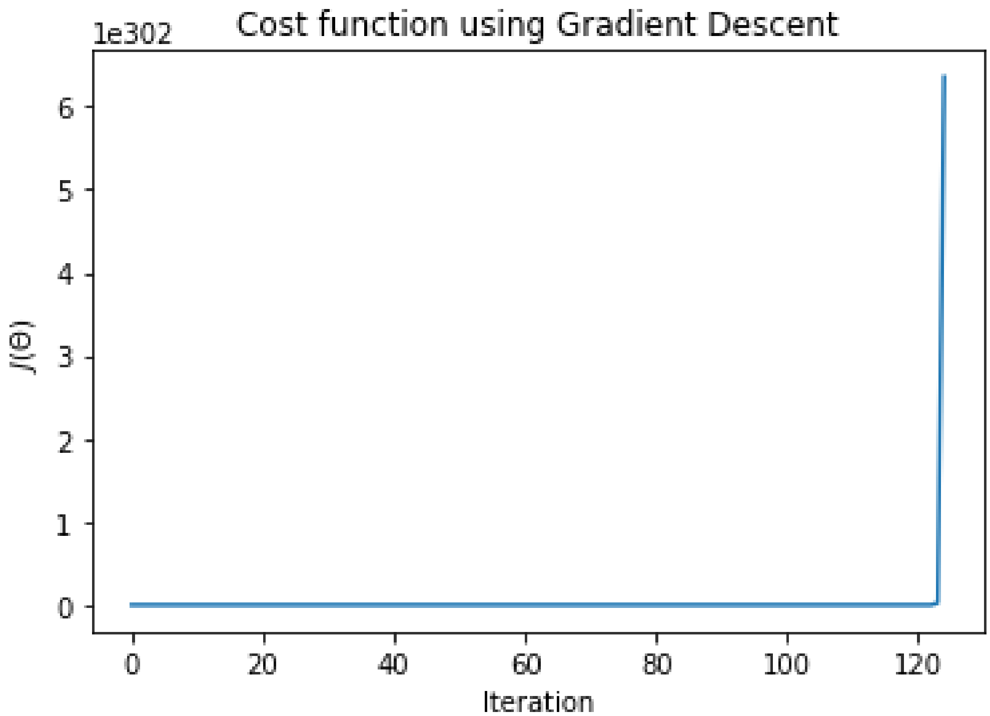
CASE 4 : ABNORMAL REACTION [LEARNING RATE=10] [ITERATIONS=1000]

- THETA is initialized to zero.
- Learningrate is initialized to 100
- Number of Iteration is initialized to 1000
- gradientDescent() function is invoked.

WE CAN NOTICE J[] BOUNCING BETWEEN HIGHER AND LOWER VALUES WITHOUT REACHING THE GLOBAL MINIMA

[138]:
```
theta=np.zeros((8,1)) #RESETTING
#theta, J = gradientDescent(X,y,theta,10,1000)
#print("THETA",theta)
#print("\nJ[]=",J)
print("\n")
plt.plot(J)
plt.xlabel("Iteration")
plt.ylabel("$J(\Theta)$")
plt.title("Cost function using Gradient Descent")
```

[138]:
```
Text(0.5, 1.0, 'Cost function using Gradient Descent')
```



PREDICTION WITH LINEAR REGRESSION

[130]:
```
#PREDICTION
p=[100,15,1,20,0,1,2,25] #Dummy input
pdt=np.dot(p,theta) #Predicted result
print(pdt) #Printing
```

```
[99293.23409259]
```