Karanjot Singh

Q1. Convert .txt file to .csv file.

Q2. Name the first column as Marks 1, second as Marks 2 and third column as admission. Consider 0 as not admitted and 1 as admitted.

In [80]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#df = pd.read_csv(r'university_admission.txt')
#df.to_csv('university_admission.csv')
data = pd.read_csv(r"university_admission.txt",sep = ',',names = ['Marks1','Marks2','Ad
mission'])
data.to_csv('university_admission.csv')
data.head()
```

Out[80]:

	Marks1	Marks2	Admission
0	34.623660	78.024693	0
1	30.286711	43.894998	0
2	35.847409	72.902198	0
3	60.182599	86.308552	1
4	79.032736	75.344376	1

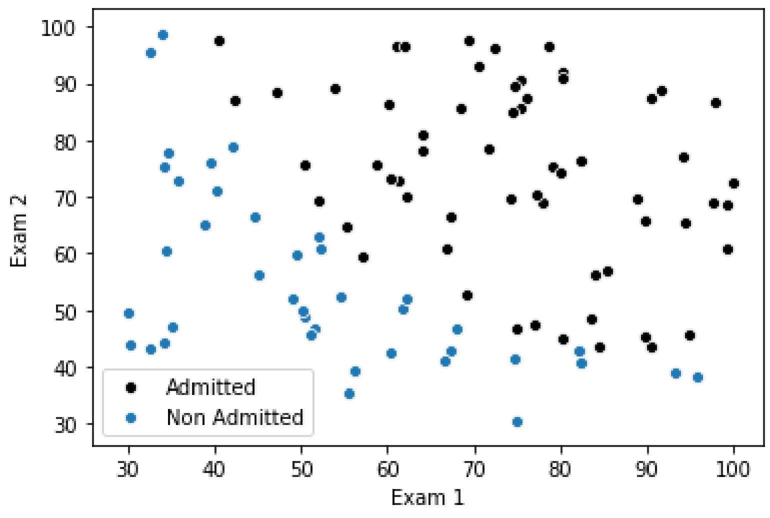
3. Plot scatter graph between exam 1 and exam 2. The marker should be x for not admitted and + for admitted.

In [81]:

```
X=data.iloc[:,:-1].values
y=data.iloc[:,-1].values
pos,neg = (y==1).reshape(100,1),(y==0).reshape(100,1)
#print(X[neg[:,0],1])
sns.scatterplot(X[pos[:,0],0],X[pos[:,0],1],color = 'Black')
sns.scatterplot(X[neg[:,0],0],X[neg[:,0],1])
plt.xlabel("Exam 1")
plt.ylabel("Exam 2")
plt.legend(['Admitted','Non Admitted'],loc = 0)
#print(neg)
```

Out[81]:

```
<matplotlib.legend.Legend at 0x2afdac58688>
```



Q4. Calculate Sigmoid Function.

In [82]:

```
def sigmoid(s):
    return 1/(1+np.exp(-s))

sigmoid(2)
```

Out[82]:

```
0.8807970779778823
```

Q5. Find the cost function and gradient decent.

In [83]:

```
def costFunction(theta, X, y):
    m=len(y)
    predictions = sigmoid(np.dot(X,theta))
    error = (-y * np.log(predictions)) - ((1-y)*np.log(1-predictions))
    cost = 1/m * sum(error)
    grad = 1/m * np.dot(X.transpose(),(predictions - y))
    return cost[0] , grad
def featureNormalization(X):
    mean = np.mean(X,axis = 0)
    std = np.std(X,axis = 0)
    X_normalized = (X-mean)/std
    return X_normalized,mean,std
```

In [84]:

```
m , n = X.shape[0], X.shape[1]
X, X_mean, X_std = featureNormalization(X)
X= np.append(np.ones((m,1)),X,axis=1)
y= y.reshape(m,1)
initial_theta = np.zeros((n+1,1))
cost, grad = costFunction(initial_theta,X,y)
print("Cost of initial theta is:",cost)
print("Gradient at initial theta (zeros):",grad)
```

```
Cost of initial theta is: 0.693147180559946
Gradient at initial theta (zeros): [[-0.1       ]
 [-0.28122914]
 [-0.25098615]]
```

In [103]:

```
def gradientDescent(X,y,theta,alpha,num_iters):
    m = len(y)
    j_history = []
    for i in range(num_iters):
        cost,grad = costFunction(theta,X,y)
        theta = theta-(alpha*grad)
        j_history.append(cost)
    return theta,j_history
theta,j_history = gradientDescent(X,y,initial_theta,1,400)
print("theta optimixed:\n",theta)
print("Cost of theta:",j_history[-1])
#print(j_history)
```

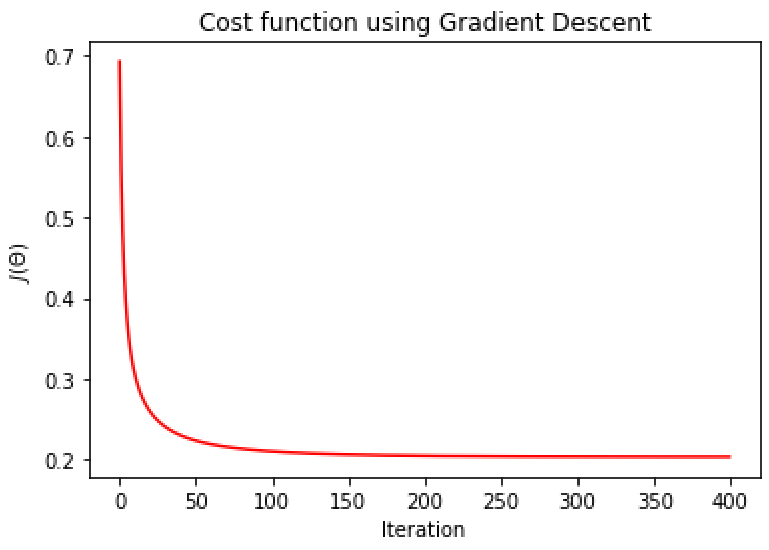
```
theta optimixed:
 [[1.65947664]
 [3.8670477 ]
 [3.60347302]]
Cost of theta: 0.20360044248226664
```

In [115]:

```
plt.plot(j_history,color = 'Red')
plt.xlabel('Iteration')
plt.ylabel('$J(\Theta)$')
plt.title('Cost function using Gradient Descent')
```

Out[115]:

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



In [93]:

```
def classifierPredict(theta,X):
    predictions = X.dot(theta)
    return predictions>0
```

Q6. Then Find the probability of getting admitted or not by user entry of marks.

In [117]:

```
x_test = np.array([45,85])
x_test = (x_test - X_mean)/X_std
x_test = np.append(np.ones(1),x_test)
prob = sigmoid(x_test.dot(theta))
print("For a student with scores 45 and 85, we predict an admission probability of",pro
b[0])
```

```
For a student with scores 45 and 85, we predict an admission probability o
f 0.7677628875792492
```

In [96]:

```
p = classifierPredict(theta,X)
print("Training Accuracy:",sum(p==y)[0],"%")
```

```
Training Accuracy: 89 %
```