Importing the libraries

In [1]:
```
import pandas as pd
import numpy as np
import sklearn as sk
from numpy import log,dot,exp,shape
import matplotlib.pyplot as plt
```

Importing the dataset

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

In [3]:
```
dataset
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

768 rows × 9 columns

Setting the feature and target variable

In [4]:
```
x=dataset.iloc[:,:-1].values
y=dataset.iloc[:,-1].values
```

In [5]:
```
x
```

Out[5]:
```
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.
]])
```

In [6]:
```
y
```

Out[6]:
```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1,
       0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
       1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
       0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1,
       0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0,
       1, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0,
       1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0,
       1, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1,
       0, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0,
       0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0,
       1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
```

```
0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1,
1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0])
```

Filling the missing data

In [7]:
```
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(missing_values=np.nan,strategy='mean')
imputer.fit(x)
x=imputer.transform(x)
```

In [8]:
```
x
```

Out[8]:
```
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.
]])
```

Splitting the dataset into training set and test set(test data set 20%)

In [9]:
```
x_train=x[:615,:]
x_test=x[615:,:]
y_train=y[:615]
y_test=y[615:]
```

In [10]:
```
x_train
```

Out[10]:
```
array([[  6.   , 148.   ,  72.   , ...,  33.6  ,   0.627,  50.   ],
       [  1.   ,  85.   ,  66.   , ...,  26.6  ,   0.351,  31.   ],
       [  8.   , 183.   ,  64.   , ...,  23.3  ,   0.672,  32.   ],
       ...,
       [  7.   , 168.   ,  88.   , ...,  38.2  ,   0.787,  40.   ],
       [  6.   , 105.   ,  80.   , ...,  32.5  ,   0.878,  26.   ],
       [ 11.   , 138.   ,  74.   , ...,  36.1  ,   0.557,  50.
]])
```

In [11]:
```
x_test
```

Out[11]:
```
array([[  3.   , 106.   ,  72.   , ...,  25.8  ,   0.207,  27.   ],
       [  6.   , 117.   ,  96.   , ...,  28.7  ,   0.157,  30.   ],
       [  2.   ,  68.   ,  62.   , ...,  20.1  ,   0.257,  23.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,  26.2  ,   0.245,  30.   ],
       [  1.   , 126.   ,  60.   , ...,  30.1  ,   0.349,  47.   ],
       [  1.   ,  93.   ,  70.   , ...,  30.4  ,   0.315,  23.   
]])
```

In [12]:
```
y_train
```

Out[12]:
```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1,
0, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1,
0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0,
1, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0,
1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0,
0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0,
1, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1,
0, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0,
       1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0,
1])
```

In [13]:
```
y_test
```

Out[13]:
```
array([0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
0, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0,
1, 1,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1,
0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0,
0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1,
0])
```

Feature Scaling

In [14]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
```

In [15]:
```
x_train
```

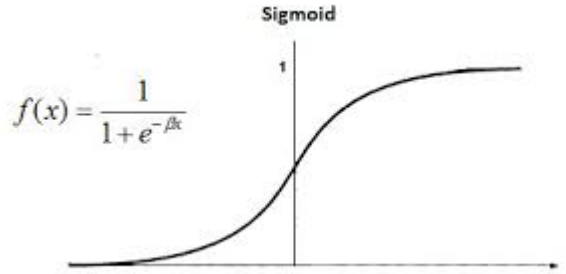
Out[15]:
```
array([[ 0.65218839,  0.84277421,  0.17715826, ...,  0.21380506,
         0.43319642,  1.42490668],
       [-0.83887428, -1.086291  , -0.12931806, ..., -0.66513955,
        -0.38742276, -0.1889379 ],
       [ 1.24861345,  1.9144771 , -0.23147683, ..., -1.07949916,
         0.56699303, -0.10399871],
       ...,
       [ 0.95040092,  1.45517586,  0.99442844, ...,  0.79139724,
         0.90891769,  0.5755148 ],
       [ 0.65218839, -0.47388934,  0.58579335, ...,  0.0756852 ,
         1.17948416, -0.61363385],
       [ 2.14325106,  0.53657338,  0.27931703, ...,  0.52771386,
         0.22506837,  1.42490668]])
```

In [16]:
```
x_test
```

Out[16]:
```
array([[-0.24244921, -0.44326926,  0.17715826, ..., -0.76559036,
        -0.81557189, -0.52869466],
       [ 0.65218839, -0.10644835,  1.40306353, ..., -0.40145617,
        -0.96423479, -0.27387709],
       [-0.54066175, -1.6068324 , -0.3336356 , ..., -1.48130241,
        -0.666909  , -0.86845141],
       ...,
       [ 0.35397585,  0.01603198,  0.17715826, ..., -0.71536496,
        -0.70258809, -0.27387709],
       [-0.83887428,  0.16913239, -0.43579437, ..., -0.22566724,
        -0.39336927,  1.17008912],
       [-0.83887428, -0.84133034,  0.07499949, ..., -0.18799819,
        -0.49446004, -0.86845141]])
```

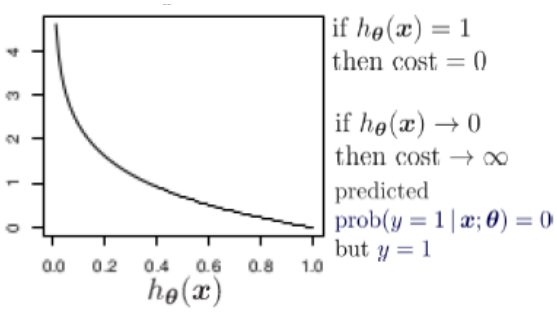
Logistic Regression Class

In [17]:
```
#defining the logistic regression class
class LogisticRegression:

    #sigmoid function used for calculating the hypothesis
    def sigmoid(self,z):
            sig = 1/(1+exp(-z))
            return sig

    def initialize(self,X):
            weights = np.zeros((shape(X)[1]+1,1))
            X = np.c_[np.ones((shape(X)[0],1)),X]
            return weights,X

    #training the model
    def fit(self,X,y,alpha=0.001,iter=400):
            weights,X = self.initialize(X)
            def cost(theta):
                z = dot(X,theta)
                cost0 = y.T.dot(log(self.sigmoid(z)))
                cost1 = (1-y).T.dot(log(1-self.sigmoid(z)))
                cost = -((cost1 + cost0))/len(y)
                return cost
            cost_list = np.zeros(iter,)
            for i in range(iter):
                weights = weights - alpha*dot(X.T,self.sigmoid(dot(X,
                cost_list[i] = cost(weights)
            self.weights = weights
            return cost_list

    #for result prediction
    def predict(self,X):
            z = dot(self.initialize(X)[1],self.weights)
            lis = []
            for i in self.sigmoid(z):
                if i>0.5:
                    lis.append(1)
                else:
                    lis.append(0)
            return lis
```



$$\text{cost}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}),\ y\right) = \begin{cases} -\log(h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 1 \\ -\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 0 \end{cases}$$

if $y = 1$ if $y = 0$



Model Training

In [18]:
```
#object creation of the LogisticRegression class
classifier= LogisticRegression()
```

In [19]:
```
#fitting our model on training set
model= classifier.fit(x_train,y_train)
```

Model Prediction

In [20]:
```
#predicting the rersults on test set
y_pred = classifier.predict(x_test)
#y_pred here is a list
```

In [21]:
```
#converting the pandas series to a list
y_test=y_test.tolist()
```

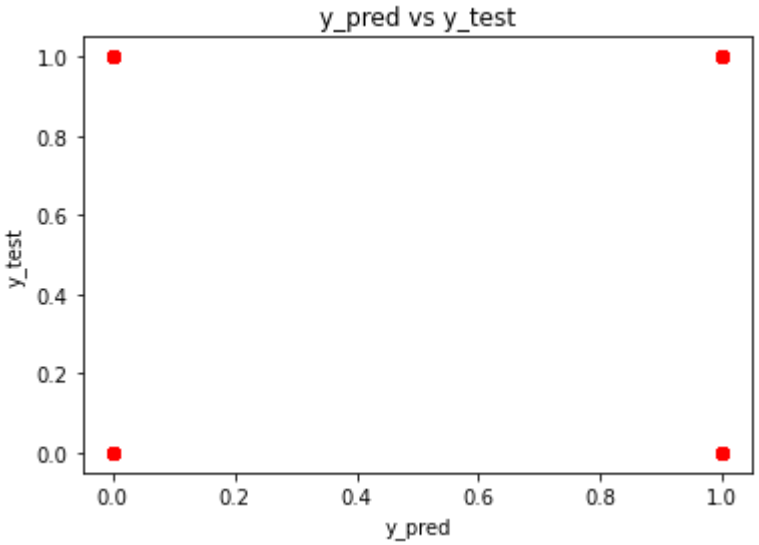
In [22]:
```
y_pred
```

Out[22]:
```
[0,
 0,
 0,
 1,
 0,
 0,
 0,
 1,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
```

In [23]:
```
y_test
```

Out[23]:
```
[0,
 0,
 0,
 1,
 1,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 1,
 0,
 0,
 0,
```

In [24]:
```
#plotting to analyse the predictied and actual values
plt.scatter(y_pred,y_test,color='red') #plotting the actual data
plt.title('y_pred vs y_test')
plt.xlabel('y_pred')
plt.ylabel("y_test")
plt.show()
```



Defining F1_score

In [25]:
```
def F1_score(y,y_pred):
    tp,tn,fp,fn = 0,0,0,0
    for i in range(len(y)):
        if y[i] == 1 and y_pred[i] == 1:
            tp += 1
        elif y[i] == 1 and y_pred[i] == 0:
            fp += 1
        elif y[i] == 0 and y_pred[i] == 1:
            fn += 1
        elif y[i] == 0 and y_pred[i] == 0:
            tn += 1
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    f1_score = 2*precision*recall/(precision+recall)
    return f1_score
```

F1 is calculated as follows:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

where:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

In "macro" F1 a separate F1 score is calculated for each `species` value and then averaged.

F1_Score Metric value

In [26]:
```
#calculating the f1_score for checking the accuracy
f1_score=F1_score(y_test,y_pred)
```

In [27]:
```
f1_score
```

Out[27]:
```
0.6236559139784946
```

Defining The Accuracy

In [28]:
```
def accuracy(y,y_pred):
    tp,tn,fp,fn=0,0,0,0
    for i in range(len(y)):
        if y[i]==1 and y_pred[i]==1:
            tp+=1
        elif y[i]==1 and y_pred[i]==0:
            fp+=1
        elif y[i]==0 and y_pred[i]==1:
            fn+=1
        elif y[i]==0 and y_pred[i]==0:
            tn+=1
    accuracy_score=(tp+tn)/(tp+tn+fp+fn)
    return accuracy_score
```

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy Metric Value

In [29]:
```
#calculating the accuracy
accuracy=accuracy(y_test,y_pred)
accuracy
```

Out[29]:
```
0.7712418300653595
```