In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import math
import random
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler,MinMaxScaler
```

In [3]:
```
location=r"C:\Users\HP\Desktop\assignment 1\23CS60R57\logistics\Pumpkin_Seeds_Dataset
df = pd.read_excel(r"Pumpkin_Seeds_Dataset.xlsx",sheet_name='Pumpkin_Seeds_Dataset')
```

In [4]:
```
df.head()
df.insert(0, 'bias', '1')
```

In [5]:
```
df['Class'].replace(['Ürgüp Sivrisi','Çerçevelik'],[0, 1], inplace=True)
df.head()
```

Out[5]:

	bias	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Convex_Area	Equiv_Diameter	Ecce
0	1	56276	888.242	326.1485	220.2388	56831	267.6805	
1	1	76631	1068.146	417.1932	234.2289	77280	312.3614	
2	1	71623	1082.987	435.8328	211.0457	72663	301.9822	
3	1	66458	992.051	381.5638	222.5322	67118	290.8899	
4	1	66107	998.146	383.8883	220.4545	67117	290.1207	

In [6]:
```
data=np.array(df,dtype=float)
scaler=StandardScaler()
X=data[:,:13]
Y=data[:,-1]
X=scaler.fit_transform(X)
X[:,0]+=1
```

In [7]:
```
X_train, X_temp, y_train, y_temp = train_test_split(X, Y, test_size=0.5, random_state
X_validation, X_test, y_validation, y_test = train_test_split(X_temp, y_temp, test_si
print(len(X_train))
print(len(X_validation))
print(len(X_test))
```

```
1250
750
500
```

In [8]:
```
def sigmoid(y_predict):
    try:
        ans = math.exp(-1*y_predict)
    except OverflowError:
        ans = float('inf')
    return 1/(1+ans)
```

In [9]:
```
def h(x,theta):
    y=np.dot(x,theta)
    return sigmoid(y)
```

In [10]:
```
def maximum_liklihood(y_predict,y):
    loss=(y[i]*log(y_predict[i]))+(1-y[i])*log(1-y_predict[i])
    return loss
```

In [11]:
```
def accuracy(y,y_p):
    l=len(y)
    acc=0
    for i in range(l):
        acc=acc+(y[i]*(y_p[i]))+(1-y[i])*(1-(y_p[i]))
    return acc/l
def precision(y,y_p):
    l=len(y)
    tp=0
    fp=0
    for i in range(l):
        tp+=y[i]*y_p[i]
        fp+=y_p[i]*(1-y[i])
    return tp/(tp+fp)
def recall(y,y_p):
    l=len(y)
    tp=0
    fn=0
    for i in range(l):
        tp=tp+y[i]*y_p[i]
        fn=fn+(1-y_p[i])*(y[i])
    return tp/(tp+fn)
```

In [12]:
```
def prediction(x,theta):
    y=np.zeros((x.shape[0],1))
    l=x.shape[0]
    for i in range(l):
        y[i]=round(h(x[i],theta))
    return y
```

In [13]:
```
def gradient_ascent(x,y,theta,learning_rate,epochs):
    l=len(y)
    for _ in range(epochs):
        for i in range(l):
            y_predicted=h(x[i],theta)
            if(y[i]==0):
                y_predicted=1-y_predicted
            loss=(y[i]-y_predicted)*learning_rate*x[i]
            theta=theta+loss
    return theta
```

In [14]:
```
theta=np.zeros(X_train.shape[1])
print(theta)
theta=gradient_ascent(X_train,y_train,theta,0.0001,300)
print(theta)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[-11.62031407  -3.39857845  -7.8694635  -11.22719766   7.74231978
  -3.34224159  -3.2408463  -12.90892479  -3.05102842   4.9310491
  13.2551307  -14.51005648  14.05064974]
```

In [15]:
```
predicted_y=prediction(X_test,theta)
print("Accuracy " + str(accuracy(y_test,predicted_y)))
print("Precission "+ str(precision(y_test,predicted_y)))
print("Recall "+str(recall(y_test,predicted_y)))
```

```
Accuracy [0.842]
Precission [0.86026201]
Recall [0.80737705]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: