Data Science Lab Assignment 9

Name: P Manohar Rao

Roll No: 197158

Section: CSE-A

1. Implement Principal Component Analysis from scratch in Python for the following dataset and show the following steps below.
   a. Dataset: https://archive.ics.uci.edu/ml/datasets/iris
   b. Scale the dataset.
   c. Calculate the covariance matrix for the features in the dataset.
   d. Calculate the eigenvalues and eigenvectors for the covariance matrix.
   e. Sort eigenvalues and their corresponding eigenvectors.
   f. Plot the principal components and percentage of explained variances.
   g. Choose first k eigen vectors.
   h. Transform the original matrix.

```
# Implementation PCA
import pandas as pd
import numpy as np
from math import sqrt
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
```

```
# PART A
df = pd.read_csv("/content/Iris.csv")
df
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa

```
# variable used
# Independent variable = All col - {Species} = X
# Dependent variable = Species = Y

X, Y = df.drop(columns = ["Id", "Species"]), df['Species']
# Scaling
scaler = StandardScaler()
X = scaler.fit_transform(X)
print("Independent Features  Shape : ",X.shape)

for i in range(len(Y)):
  if Y.iat[i] == "Iris-setosa":
    Y.iat[i] = 0
  if Y.iat[i] == "Iris-versicolor":
    Y.iat[i] = 1
  if Y.iat[i] == "Iris-virginica":
    Y.iat[i] = 2
print("Dependent Feature Shape : ",Y.shape)
```

```
Independent Features  Shape :  (150, 4)
Dependent Feature Shape :  (150,)
```

```
#Calculating the covariance matrix for the features in the dataset
def get_covariance_matrix(x):
    n = x.shape[0]
    C = np.dot(x.T, x) / (n)
    return C

Cov = get_covariance_matrix(X)

print(Cov.shape)
print(Cov)
```

```
(4, 4)
[[ 1.         -0.10936925  0.87175416  0.81795363]
 [-0.10936925  1.         -0.4205161  -0.35654409]
 [ 0.87175416 -0.4205161   1.          0.9627571 ]
 [ 0.81795363 -0.35654409  0.9627571   1.        ]]
```

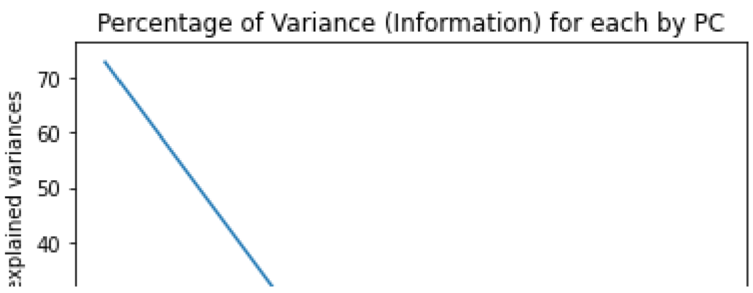
```
# Calculating the eigenvalues and eigenvectors for the covariance matrix
def get_eigenvectors(C):
    # calculating eigenvalues & eigenvectors of covariance matrix 'C'
    eigenvalues, eigenvectors = np.linalg.eig(C)
    print("Eigen Values : ",eigenvalues)
    print("Eigen Vector :\n",eigenvectors)
    # sort eigenvalues descending and select columns based on n_components
    n_cols = np.flip(np.argsort(eigenvalues))
    selected_vectors = eigenvectors[:, n_cols]
    return np.flip(np.sort(eigenvalues)),selected_vectors

eigenvalues,eigenvectors = get_eigenvectors(Cov)

print("\nSorted Eigen Values : ",eigenvalues)
print("Sorted Eigen Vector :\n",eigenvectors)
```

```
Eigen Values :  [2.91081808 0.92122093 0.14735328 0.02060771]
Eigen Vector :
 [[ 0.52237162 -0.37231836 -0.72101681  0.26199559]
 [-0.26335492 -0.92555649  0.24203288 -0.12413481]
 [ 0.58125401 -0.02109478  0.14089226 -0.80115427]
 [ 0.56561105 -0.06541577  0.6338014   0.52354627]]

Sorted Eigen Values :  [2.91081808 0.92122093 0.14735328 0.02060771]
Sorted Eigen Vector :
 [[ 0.52237162 -0.37231836 -0.72101681  0.26199559]
 [-0.26335492 -0.92555649  0.24203288 -0.12413481]
 [ 0.58125401 -0.02109478  0.14089226 -0.80115427]
 [ 0.56561105 -0.06541577  0.6338014   0.52354627]]
```

```
#Plot the principal components and percentage of explained variances.
explained_variances = []
for i in range(len(eigenvalues)):
    explained_variances.append(eigenvalues[i] / np.sum(eigenvalues))

print("Sum of  Explained Variance :",np.sum(explained_variances))
print("Explained Variance :", explained_variances)
```

```
Sum of  Explained Variance : 1.0
Explained Variance : [0.7277045209380139, 0.23030523267680594, 0.036838319576273884,
```

```
#Plot the principal components and percentage of explained variances.
import matplotlib.pyplot as plt
x = [1,2,3,4]
q = []
for i in explained_variances:
    q.append(i*100)
plt.plot(x,q)
plt.xlabel('Principal components')
plt.ylabel('Percentage of explained variances')
plt.title('Percentage of Variance (Information) for each by PC')
plt.show()
```



```
#consider only the features upto where the pov adds to <95% here 2
k = 2
eisub = eigenvectors[:,0:k]
eisub
```

```
     array([[ 0.52237162, -0.37231836],
            [-0.26335492, -0.92555649],
            [ 0.58125401, -0.02109478],
            [ 0.56561105, -0.06541577]])
```

```
X_transformed = np.dot(eisub.T,X.T).T
print("Transformed X :\n", np.around(X_transformed, 2))
```

```
      [ 1.04  1.39]
      [ 0.07  0.21]
      [ 0.28  1.33]
      [ 0.27  1.12]
      [ 0.62 -0.03]
      [ 0.33  0.99]
      [-0.37  2.02]
      [ 0.28  0.85]
      [ 0.09  0.17]
      [ 0.22  0.38]
      [ 0.57  0.15]
      [-0.46  1.54]
      [ 0.25  0.6 ]
      [ 1.85 -0.87]
      [ 1.15  0.7 ]
      [ 2.21 -0.55]
      [ 1.44  0.05]
      [ 1.87 -0.29]
      [ 2.75 -0.79]
      [ 0.36  1.56]
      [ 2.3  -0.41]
      [ 2.    0.72]
      [ 2.27 -1.92]
      [ 1.37 -0.69]
      [ 1.6   0.43]
      [ 1.88 -0.41]
      [ 1.25  1.17]
      [ 1.46  0.44]
      [ 1.59 -0.68]
      [ 1.47 -0.25]
      [ 2.44 -2.56]
      [ 3.31  0.  ]
      [ 1.25  1.72]
      [ 2.04 -0.91]
      [ 0.97  0.57]
      [ 2.9  -0.4 ]
      [ 1.33  0.49]
```

```
[ 1.7  -1.01]
[ 1.96 -1.  ]
[ 1.17  0.32]
[ 1.02 -0.07]
[ 1.79  0.19]
[ 1.86 -0.56]
[ 2.44 -0.25]
[ 2.32 -2.63]
[ 1.86  0.18]
[ 1.11  0.3 ]
[ 1.2   0.82]
[ 2.8  -0.84]
[ 1.58 -1.07]
[ 1.35 -0.42]
[ 0.92 -0.02]
[ 1.85 -0.67]
[ 2.02 -0.61]
[ 1.9  -0.69]
[ 1.15  0.7 ]
[ 2.04 -0.86]
[ 2.   -1.05]
[ 1.87 -0.38]
```

2. Implement PCA and Logistic Regression for the following dataset by performing the required steps.
   a. Dataset: https://www.kaggle.com/datasets/dileep070/heart-disease-prediction-using-logistic-regression
   b. Loading the dataset
   c. Scale the dataset
   d. Select the principal components
   e. Build the Logistic regression model with the transformed dataset.

```
#importing the dataset
import pandas as pd
import numpy as np
df = pd.read_csv(r"/content/framingham.csv")
#drop nan values
df = df.dropna()
df
```

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prev
0	1	39	4.0	0	0.0	0.0	0	
1	0	46	2.0	0	0.0	0.0	0	
2	1	48	1.0	1	20.0	0.0	0	
3	0	61	3.0	1	30.0	0.0	0	
4	0	46	3.0	1	23.0	0.0	0	

```
# variable used
# Independent variable = All col - {TenYearCHD} = X
# Dependent variable = TenYearCHD = Y

X, Y = df.drop(columns = ["TenYearCHD"]), df['TenYearCHD']
# Scaling
scaler = StandardScaler()
X = scaler.fit_transform(X)
print("Independent Features  Shape : ",X.shape)

print("Dependent Feature Shape : ",Y.shape)
```

```
Independent Features  Shape :  (3656, 15)
Dependent Feature Shape :  (3656,)
```

```
#Calculating the covariance matrix for the features in the dataset
def get_covariance_matrix(x):
    n = x.shape[0]
    C = np.dot(x.T, x) / (n)
    return C

Cov = get_covariance_matrix(X)

print(Cov.shape)
print(np.around(Cov,3))
```

```
(15, 15)
[[ 1.    -0.024  0.018  0.207  0.331 -0.052 -0.002  0.001  0.014 -0.07
  -0.045  0.052  0.073 -0.115  0.003]
 [-0.024  1.    -0.159 -0.211 -0.189  0.135  0.051  0.307  0.109  0.268
   0.389  0.209  0.137 -0.003  0.118]
 [ 0.018 -0.159  1.     0.025  0.014 -0.014 -0.03  -0.079 -0.04  -0.013
  -0.125 -0.059 -0.137 -0.064 -0.032]
 [ 0.207 -0.211  0.025  1.     0.774 -0.052 -0.038 -0.108 -0.042 -0.051
  -0.134 -0.116 -0.16   0.05  -0.053]
 [ 0.331 -0.189  0.014  0.774  1.    -0.046 -0.036 -0.07  -0.037 -0.03
  -0.095 -0.057 -0.087  0.064 -0.054]
 [-0.052  0.135 -0.014 -0.052 -0.046  1.     0.113  0.263  0.049  0.094
   0.271  0.2    0.106  0.013  0.054]
 [-0.002  0.051 -0.03  -0.038 -0.036  0.113  1.     0.066  0.01   0.013
   0.061  0.056  0.036 -0.017  0.016]
 [ 0.001  0.307 -0.079 -0.108 -0.07   0.263  0.066  1.     0.081  0.167
   0.698  0.618  0.303  0.147  0.087]
 [ 0.014  0.109 -0.04  -0.042 -0.037  0.049  0.01   0.081  1.     0.048
   0.103  0.051  0.089  0.061  0.615]
 [-0.07   0.268 -0.013 -0.051 -0.03   0.094  0.013  0.167  0.048  1.
```

```
  0.22   0.175  0.121  0.093  0.05 ]
 [-0.045  0.389 -0.125 -0.134 -0.095  0.271  0.061  0.698  0.103  0.22
   1.     0.787  0.331  0.185  0.135]
 [ 0.052  0.209 -0.059 -0.116 -0.057  0.2    0.056  0.618  0.051  0.175
   0.787  1.     0.386  0.179  0.064]
 [ 0.073  0.137 -0.137 -0.16  -0.087  0.106  0.036  0.303  0.089  0.121
   0.331  0.386  1.     0.074  0.084]
 [-0.115 -0.003 -0.064  0.05   0.064  0.013 -0.017  0.147  0.061  0.093
   0.185  0.179  0.074  1.     0.097]
 [ 0.003  0.118 -0.032 -0.053 -0.054  0.054  0.016  0.087  0.615  0.05
   0.135  0.064  0.084  0.097  1.   ]]
```

```
# Calculating the eigenvalues and eigenvectors for the covariance matrix
def get_eigenvectors(C):
    # calculating eigenvalues & eigenvectors of covariance matrix 'C'
    eigenvalues, eigenvectors = np.linalg.eig(C)
    print("Eigen Values : ", np.around(eigenvalues, 3))
    print("Eigen Vector :\n", np.around(eigenvectors,3))

    # sort eigenvalues descending and select columns based on n_components
    n_cols = np.flip(np.argsort(eigenvalues))
    selected_vectors = eigenvectors[:, n_cols]
    return np.flip(np.sort(eigenvalues)),selected_vectors

eigenvalues,eigenvectors = get_eigenvectors(Cov)

print("\nSorted Eigen Values : ", np.around(eigenvalues,3))
print("Sorted Eigen Vector :\n", np.around(eigenvectors,3))
```

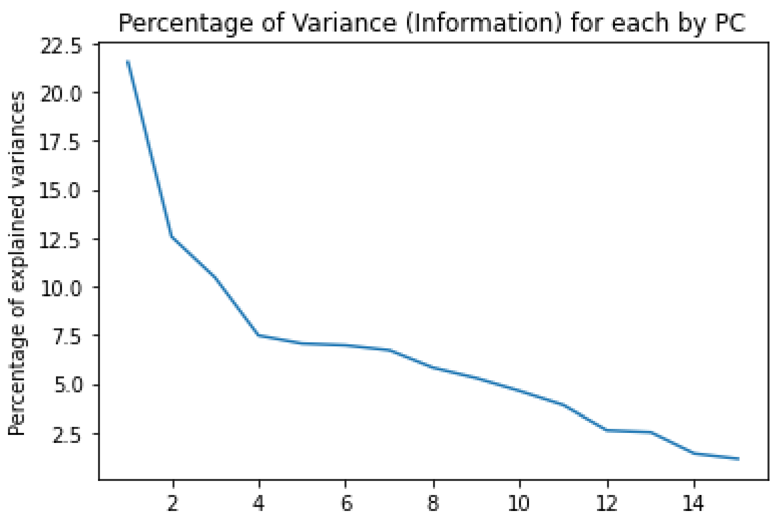
```
Eigen Values :  [3.229 1.884 1.568 0.172 0.212 0.376 0.391 0.584 0.692 0.791 0.872
 1.007 1.058 1.045]
Eigen Vector :
 [[ 0.055 -0.364  0.043 -0.077 -0.125 -0.024 -0.022  0.273  0.595 -0.007
   0.182 -0.528  0.171  0.263 -0.027]
 [-0.295  0.096  0.026  0.151  0.001  0.092  0.06  -0.593  0.25  -0.334
  -0.077 -0.122  0.244 -0.01   0.512]
 [ 0.107  0.017 -0.031 -0.036  0.003  0.014 -0.003 -0.309  0.056 -0.05
   0.292  0.004  0.546 -0.417 -0.573]
 [ 0.2   -0.588  0.051  0.024 -0.681 -0.005  0.024 -0.178 -0.241 -0.033
  -0.096  0.11  -0.017 -0.137  0.122]
 [ 0.169 -0.633  0.043  0.008  0.719  0.02   0.014 -0.134 -0.118  0.01
  -0.03   0.035  0.007 -0.065  0.113]
 [-0.209 -0.043 -0.053  0.037 -0.002  0.051  0.046  0.015  0.301  0.493
  -0.539 -0.147 -0.089 -0.54  -0.027]
 [-0.068  0.017 -0.027 -0.009  0.002  0.006 -0.004 -0.008 -0.06  -0.16
   0.531 -0.342 -0.541 -0.507  0.124]
 [-0.43  -0.16  -0.121  0.147  0.01  -0.583 -0.571  0.065 -0.082 -0.212
  -0.106 -0.012  0.022 -0.029 -0.135]
 [-0.135  0.005  0.685 -0.011 -0.006  0.487 -0.506  0.044 -0.094  0.005
  -0.024 -0.053  0.016 -0.021 -0.062]
 [-0.195 -0.019 -0.009 -0.006 -0.017 -0.048 -0.031  0.376 -0.104  0.338
   0.373  0.225  0.46  -0.204  0.499]
 [-0.481 -0.151 -0.104 -0.763 -0.006  0.168  0.201  0.076 -0.114 -0.22
  -0.083  0.04   0.025 -0.001 -0.088]
 [-0.436 -0.195 -0.157  0.598 -0.014  0.376  0.316  0.214 -0.13  -0.118
   0.055  0.005  0.012  0.083 -0.247]
 [-0.286 -0.063 -0.03  -0.054 -0.037 -0.058 -0.022 -0.467 -0.176  0.627
```

```
   0.275 -0.171 -0.113  0.366 -0.107]
 [-0.128 -0.138  0.073 -0.001 -0.026  0.017 -0.049 -0.099  0.57   0.022
   0.238  0.686 -0.294  0.014 -0.089]
 [-0.148  0.011  0.68   0.054  0.017 -0.484  0.516  0.037 -0.036 -0.029
  -0.01  -0.017  0.004 -0.036 -0.078]]

Sorted Eigen Values :  [3.229 1.884 1.568 1.12  1.058 1.045 1.007 0.872 0.791 0.69
 0.376 0.212 0.172]
Sorted Eigen Vector :
 [[ 0.055 -0.364  0.043 -0.528  0.263 -0.027  0.171  0.182 -0.007  0.595
   0.273 -0.022 -0.024 -0.125 -0.077]
 [-0.295  0.096  0.026 -0.122 -0.01   0.512  0.244 -0.077 -0.334  0.25
  -0.593  0.06   0.092  0.001  0.151]
 [ 0.107  0.017 -0.031  0.004 -0.417 -0.573  0.546  0.292 -0.05   0.056
  -0.309 -0.003  0.014  0.003 -0.036]
 [ 0.2   -0.588  0.051  0.11  -0.137  0.122 -0.017 -0.096 -0.033 -0.241
  -0.178  0.024 -0.005 -0.681  0.024]
 [ 0.169 -0.633  0.043  0.035 -0.065  0.113  0.007 -0.03   0.01  -0.118
  -0.134  0.014  0.02   0.719  0.008]
 [-0.209 -0.043 -0.053 -0.147 -0.54  -0.027 -0.089 -0.539  0.493  0.301
   0.015  0.046  0.051 -0.002  0.037]
 [-0.068  0.017 -0.027 -0.342 -0.507  0.124 -0.541  0.531 -0.16  -0.06
  -0.008 -0.004  0.006  0.002 -0.009]
 [-0.43  -0.16  -0.121 -0.012 -0.029 -0.135  0.022 -0.106 -0.212 -0.082
   0.065 -0.571 -0.583  0.01   0.147]
 [-0.135  0.005  0.685 -0.053 -0.021 -0.062  0.016 -0.024  0.005 -0.094
   0.044 -0.506  0.487 -0.006 -0.011]
 [-0.195 -0.019 -0.009  0.225 -0.204  0.499  0.46   0.373  0.338 -0.104
```

```
#Plot the principal components and percentage of explained variances.
explained_variances = []
for i in range(len(eigenvalues)):
    explained_variances.append(eigenvalues[i] / np.sum(eigenvalues))

print("Sum of  Explained Variance :",np.sum(explained_variances))
print("Explained Variance :", np.around(explained_variances, 3))
```

```
Sum of  Explained Variance : 1.0
Explained Variance : [0.215 0.126 0.105 0.075 0.071 0.07  0.067 0.058 0.053 0.046 0.
 0.025 0.014 0.011]
```

```
#Plot the principal components and percentage of explained variances.
import matplotlib.pyplot as plt
x = [i for i in range(1,16)]
q = []
for i in explained_variances:
    q.append(i*100)
plt.plot(x,q)
plt.xlabel('Principal components')
plt.ylabel('Percentage of explained variances')
plt.title('Percentage of Variance (Information) for each by PC')
plt.show()
```



```
#consider only the features upto where the pov adds to <95%
k = 0
sum = 0
while sum < 95:
    sum = sum + q[k]
    k = k+1
k = k-1
print("k : ",k)
eisub = eigenvectors[:,0:k]
eisub
```

```
k :  12
array([[ 0.05467462, -0.36389546,  0.04310623, -0.52764457,  0.2633536 ,
        -0.02674147,  0.1710008 ,  0.18215532, -0.00682477,  0.59547196,
         0.27319947, -0.02152826],
       [-0.2948085 ,  0.09611887,  0.02580589, -0.12167369, -0.00957423,
         0.51175022,  0.24406496, -0.07690297, -0.33419181,  0.25049638,
        -0.59305266,  0.06048594],
       [ 0.10745757,  0.01691292, -0.03083472,  0.00370124, -0.41679434,
        -0.57256411,  0.54604146,  0.29162626, -0.04982949,  0.05607413,
        -0.30871516, -0.00327381],
       [ 0.19954607, -0.58809556,  0.05063284,  0.11046361, -0.13725217,
         0.12246109, -0.01739216, -0.09587546, -0.0331284 , -0.2408757 ,
        -0.17797681,  0.02360533],
       [ 0.16949064, -0.63266357,  0.04293371,  0.03510651, -0.06546762,
         0.11340509,  0.00674547, -0.03019626,  0.00958544, -0.11791034,
        -0.13422276,  0.01434833],
       [-0.20878548, -0.04274692, -0.05277178, -0.14740065, -0.5397049 ,
        -0.02742394, -0.08881837, -0.53877469,  0.49282497,  0.30104167,
         0.01465222,  0.04632822],
       [-0.0683295 ,  0.01711142, -0.02703162, -0.34191905, -0.50734591,
         0.12377503, -0.54106591,  0.5313677 , -0.16041293, -0.0604535 ,
        -0.00764496, -0.00376257],
       [-0.42983319, -0.16016912, -0.12075133, -0.01185563, -0.02934169,
        -0.13548953,  0.0218144 , -0.10551039, -0.21179998, -0.08207739,
         0.06472279, -0.57134014],
       [-0.1349491 ,  0.00518461,  0.68507362, -0.05322854, -0.02100371,
        -0.06155345,  0.0162699 , -0.0240159 ,  0.0046704 , -0.09393563,
         0.04416125, -0.50629998],
       [-0.19482989, -0.01945319, -0.00928059,  0.2248992 , -0.20370357,
         0.49901648,  0.45986309,  0.37332822,  0.33809926, -0.1035433 ,
         0.37554585, -0.03052304],
       [-0.48085117, -0.15084757, -0.10411601,  0.04039611, -0.00134662,
        -0.08781674,  0.02517891, -0.08267041, -0.22024669, -0.11351192,
         0.07575261,  0.20111351],
       [-0.43551138, -0.19486996, -0.15705885,  0.00506375,  0.08277814,
        -0.24743193,  0.01229394,  0.05509571, -0.11803206, -0.13019644,
         0.2140456 ,  0.31617457],
       [-0.28586198, -0.06280014, -0.02966469, -0.17113738,  0.36585677,
        -0.10742093, -0.11340778,  0.27479764,  0.62691929, -0.17578375,
        -0.46742639, -0.02157555],
       [-0.12800012, -0.13767643,  0.07318006,  0.68551491,  0.01424156,
        -0.0888955 , -0.29353338,  0.2378201 ,  0.02194129,  0.57007829,
        -0.09892185, -0.04859042],
       [-0.14774936,  0.0113193 ,  0.68034311, -0.01652586, -0.03640071,
        -0.07808647,  0.00421899, -0.01002707, -0.02909666, -0.03617561,
         0.03714895,  0.51575508]])
```

```
X_transformed = np.dot(eisub.T,X.T).T
print("Transformed X :\n", np.around(X_transformed, 2))
```

```
Transformed X :
 [[ 1.8   1.01  0.03 ...  1.35 -0.21 -0.36]
 [-0.01  1.29 -0.15 ...  0.56 -0.25  0.04]
 [ 1.01 -1.43 -0.14 ...  0.27  0.39  0.09]
 ...
 [-2.16 -1.22 -0.48 ... -0.69  1.6  -0.04]
 [ 2.15 -2.35 -0.17 ...  0.11 -0.28  0.18]
 [-0.17  1.53  0.62 ...  0.16  0.58  1.  ]]
```

```
# Split a dataset into 80/20 train/test set
from random import randrange

def train_test_split(x, y, split):
  x_train, y_train = list(),list()
  train_size = split * len(x)
  x_test, y_test = list(x), list(y)
  while len(x_train) < train_size:
    idx = randrange(len(x_test))
    x_train.append(x_test.pop(idx))
    y_train.append(y_test.pop(idx))
  return np.array(x_train), np.array(x_test), np.array(y_train), np.array(y_test)

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,0.8)

print("Independent Training Feature Shape :-",X_train.shape)
print("Independent Testing Feature Shape :-",X_test.shape)
print("Dependent Training Feature Shape :- ",Y_train.shape)
print("Dependent Testing Feature Shape :- ",Y_test.shape)
```

```
Independent Training Feature Shape :- (2925, 15)
Independent Testing Feature Shape :- (731, 15)
Dependent Training Feature Shape :-  (2925,)
Dependent Testing Feature Shape :-  (731,)
```

```
# Logistic Regression
# Implement the logic of the algorithm using Gradient Descent Function
# Estimate linear regression coefficients using stochastic gradient descent

from math import exp

# Make a prediction with coefficients

def sigmoid(z):
  return 1.0 / (1.0 + exp(-z))

def predict(row, coeff):
    y_pred = coeff[0]
    for i in range(len(row)):
        y_pred += coeff[i + 1] * row[i]
    return sigmoid(y_pred)

def Gradient_Descent(x_train, y_train, alpha, n_epoch):
    coef = [0.0 for i in range(len(x_train[0])+1)]
    for epoch in range(n_epoch):
        for i in range(len(x_train)):
            y_pred = predict(x_train[i], coef)
            error = y_train[i] - y_pred
            coef[0] = coef[0] + alpha * error * y_pred * (1.0 - y_pred)
            for j in range(len(x_train[i])):
                coef[j + 1] = coef[j + 1] + alpha * error * y_pred * (1.0 - y_pred) * x_tr
    return coef

alpha = 0.1
n_epoch = 100

# Finding coefficients
coef = Gradient_Descent(X_train, Y_train, alpha, n_epoch)
print(np.around(coef,4))
```

```
     [-2.0943  0.4743  0.5886 -0.0073 -0.0618  0.222   0.0098  0.2991  0.2057
      -0.1305  0.0049  0.4338 -0.0326  0.0109 -0.0418  0.2232]
```

```
# Predict the values using test data
Y_pred = []
for i in range(len(X_test)):
  y = predict(X_test[i],coef)
  Y_pred.append(y)

# print predicted value
print("Predicted Value for testing data")
print(np.around(Y_pred,3))


# To calculate LOSS
def LOG_LOSS(actual, predict):
  error = 0.0
  for i in range(len(actual)):
    pred_error_0 = actual[i] * np.log(predict[i])
    pred_error_1 = (1 - actual[i]) * np.log(1 - predict[i])
    error += pred_error_0 + pred_error_1
  mean_error = -error/float(len(actual))
  return mean_error

me = LOG_LOSS(Y_test, Y_pred)
print("\nMean Error :- ", me)
```

```
     0.111 0.039 0.027 0.022 0.235 0.028 0.051 0.425 0.033 0.056 0.184 0.13
     0.085 0.124 0.906 0.049 0.349 0.025 0.247 0.168 0.103 0.062 0.267 0.157
     0.281 0.067 0.101 0.158 0.025 0.056 0.031 0.074 0.014 0.037 0.077 0.076
     0.028 0.114 0.107 0.409 0.079 0.06  0.094 0.022 0.151 0.416 0.167 0.095
     0.12  0.211 0.085 0.067 0.291 0.228 0.056 0.068 0.031 0.155 0.019 0.028
     0.039 0.242 0.082 0.041 0.075 0.086 0.02  0.065 0.034 0.424 0.186 0.123
     0.024 0.049 0.071 0.122 0.019 0.123 0.21  0.078 0.07  0.126 0.303 0.143
     0.113 0.407 0.521 0.047 0.171 0.497 0.02  0.317 0.219 0.03  0.163 0.1
     0.08  0.045 0.607 0.096 0.041 0.074 0.108 0.544 0.249 0.096 0.259 0.237
     0.097 0.12  0.162 0.048 0.152 0.197 0.104 0.204 0.185 0.399 0.593 0.218
     0.095 0.052 0.068 0.043 0.016 0.048 0.26  0.219 0.07  0.117 0.069 0.031
     0.12  0.138 0.156 0.032 0.069 0.019 0.108 0.147 0.122 0.605 0.141 0.343
     0.074 0.187 0.083 0.03  0.306 0.028 0.298 0.52  0.041 0.165 0.228 0.316
     0.039 0.266 0.039 0.083 0.327 0.151 0.196 0.234 0.187 0.166 0.201 0.24
     0.036 0.366 0.614 0.252 0.29  0.152 0.034 0.062 0.229 0.022 0.174 0.018
     0.014 0.036 0.215 0.068 0.174 0.021 0.073 0.035 0.031 0.227 0.087 0.036
     0.221 0.082 0.037 0.069 0.024 0.102 0.246 0.089 0.048 0.069 0.166 0.071
     0.029 0.025 0.081 0.171 0.494 0.15  0.124 0.047 0.02  0.196 0.175 0.216
     0.482 0.045 0.046 0.035 0.199 0.018 0.162 0.156 0.091 0.025 0.4   0.034
     0.033 0.058 0.036 0.052 0.143 0.411 0.022 0.023 0.204 0.325 0.058 0.038
     0.03  0.477 0.239 0.072 0.183 0.075 0.192 0.288 0.017 0.218 0.017 0.073
     0.127 0.147 0.191 0.068 0.431 0.035 0.318 0.212 0.019 0.278 0.625 0.023
     0.18  0.188 0.39  0.245 0.275 0.196 0.055 0.119 0.039 0.099 0.197 0.046
     0.168 0.261 0.771 0.05  0.043 0.032 0.03  0.074 0.052 0.018 0.032 0.037
     0.068 0.04  0.243 0.345 0.027 0.23  0.079 0.015 0.122 0.092 0.024 0.063
     0.054 0.074 0.113 0.106 0.459 0.108 0.97  0.243 0.317 0.021 0.034 0.098
     0.079 0.103 0.301 0.019 0.192 0.063 0.097 0.104 0.265 0.122 0.028 0.018
     0.454 0.224 0.016 0.023 0.13  0.158 0.241 0.14  0.032 0.038 0.299 0.048
     0.075 0.016 0.021 0.141 0.221 0.028 0.028 0.057 0.314 0.507 0.057 0.039
     0.087 0.193 0.518 0.071 0.43  0.251 0.306 0.349 0.24  0.023 0.58  0.307
     0.432 0.022 0.021 0.145 0.159 0.166 0.237 0.154 0.067 0.204 0.025 0.133
     0.086 0.013 0.175 0.201 0.071 0.077 0.31  0.072 0.155 0.123 0.028 0.244
     0.023 0.331 0.036 0.031 0.126 0.038 0.083 0.035 0.028 0.129 0.318 0.025
     0.126 0.102 0.22  0.056 0.472 0.021 0.282 0.12  0.106 0.201 0.44  0.113
     0.229 0.349 0.141 0.025 0.094 0.04  0.127 0.022 0.021 0.387 0.16  0.015
     0.083 0.022 0.116 0.141 0.019 0.178 0.05  0.252 0.018 0.066 0.093 0.034
     0.048 0.028 0.06  0.378 0.154 0.381 0.037 0.199 0.111 0.134 0.058 0.28
     0.03  0.046 0.052 0.131 0.059 0.444 0.134 0.132 0.15  0.076 0.012 0.022
     0.091 0.073 0.039 0.049 0.065 0.021 0.183 0.349 0.204 0.023 0.072 0.269
     0.058 0.04  0.041 0.163 0.057 0.371 0.313 0.134 0.066 0.128 0.087 0.181
     0.154 0.078 0.014 0.082 0.02  0.021 0.121 0.257 0.032 0.037 0.127 0.308
     0.127 0.209 0.102 0.166 0.169 0.155 0.028 0.07  0.202 0.04  0.049 0.028
     0.142 0.686 0.086 0.09  0.047 0.111 0.084 0.229 0.328 0.233 0.029 0.453
     0.156 0.042 0.346 0.115 0.069 0.408 0.092 0.31  0.111 0.147 0.298 0.071
     0.151 0.251 0.135 0.247 0.063 0.1   0.019 0.042 0.118 0.35  0.176 0.313
     0.108 0.082 0.126 0.01  0.658 0.067 0.031 0.082 0.136 0.044 0.143 0.041
     0.067 0.045 0.032 0.028 0.078 0.539 0.041 0.077 0.173 0.025 0.427 0.016

     0.283 0.115 0.057 0.039 0.455 0.057 0.816 0.208 0.241 0.131 0.108 0.097
     0.112 0.287 0.077 0.164 0.092 0.211 0.489 0.19  0.095 0.101 0.245 0.04
     0.028 0.283 0.096 0.03  0.037 0.333 0.013 0.018 0.099 0.192 0.039 0.252
     0.028 0.021 0.269 0.267 0.038 0.087 0.084 0.056 0.119 0.09  0.367 0.048
     0.07  0.165 0.185 0.086 0.02  0.049 0.106 0.075 0.023 0.225 0.047 0.066
     0.033 0.02  0.253 0.053 0.298 0.066 0.034 0.103 0.47  0.07  0.112 0.104
```