Experiment No:1
Title: Implementation of PCA.
Name:Adityaraj Hari Ingavale
Roll No:08
Batch:A1

```
# PCA implementation using Library.
```

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
```

```
irisdata=load_iris()
x=irisdata.data
print(x)
```

```
 [6.2 2.2 4.5 1.5]
 [5.6 2.5 3.9 1.1]
 [5.9 3.2 4.8 1.8]
 [6.1 2.8 4.  1.3]
 [6.3 2.5 4.9 1.5]
 [6.1 2.8 4.7 1.2]
 [6.4 2.9 4.3 1.3]
 [6.6 3.  4.4 1.4]
 [6.8 2.8 4.8 1.4]
 [6.7 3.  5.  1.7]
 [6.  2.9 4.5 1.5]
 [5.7 2.6 3.5 1. ]
 [5.5 2.4 3.8 1.1]
 [5.5 2.4 3.7 1. ]
 [5.8 2.7 3.9 1.2]
 [6.  2.7 5.1 1.6]
 [5.4 3.  4.5 1.5]
 [6.  3.4 4.5 1.6]
 [6.7 3.1 4.7 1.5]
 [6.3 2.3 4.4 1.3]
 [5.6 3.  4.1 1.3]
 [5.5 2.5 4.  1.3]
 [5.5 2.6 4.4 1.2]
 [6.1 3.  4.6 1.4]
 [5.8 2.6 4.  1.2]
 [5.  2.3 3.3 1. ]
 [5.6 2.7 4.2 1.3]
 [5.7 3.  4.2 1.2]
 [5.7 2.9 4.2 1.3]
 [6.2 2.9 4.3 1.3]
 [5.1 2.5 3.  1.1]
 [5.7 2.8 4.1 1.3]
 [6.3 3.3 6.  2.5]
 [5.8 2.7 5.1 1.9]
 [7.1 3.  5.9 2.1]
 [6.3 2.9 5.6 1.8]
 [6.5 3.  5.8 2.2]
 [7.6 3.  6.6 2.1]
 [4.9 2.5 4.5 1.7]
 [7.3 2.9 6.3 1.8]
 [6.7 2.5 5.8 1.8]
 [7.2 3.6 6.1 2.5]
 [6.5 3.2 5.1 2. ]
 [6.4 2.7 5.3 1.9]
 [6.8 3.  5.5 2.1]
 [5.7 2.5 5.  2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
```

```
pca=PCA(n_components=2)
```

```
result=pca.fit_transform(x)
print(result)
```

```
[[-2.68412563  0.31939725]
 [-2.71414169 -0.17700123]
 [-2.88899057 -0.14494943]
 [-2.74534286 -0.31829898]
 [-2.72871654  0.32675451]
 [-2.28085963  0.74133045]
 [-2.82053775 -0.08946138]
 [-2.62614497  0.16338496]
 [-2.88638273 -0.57831175]
 [-2.6727558  -0.11377425]
 [-2.50694709  0.6450689 ]
 [-2.61275523  0.01472994]
 [-2.78610927 -0.235112  ]
 [-3.22380374 -0.51139459]
 [-2.64475039  1.17876464]
 [-2.38603903  1.33806233]
 [-2.62352788  0.81067951]
 [-2.64829671  0.31184914]
 [-2.19982032  0.87283904]
 [-2.5879864   0.51356031]
 [-2.31025622  0.39134594]
 [-2.54370523  0.43299606]
 [-3.21593942  0.13346807]
 [-2.30273318  0.09870885]
 [-2.35575405 -0.03728186]
 [-2.50666891 -0.14601688]
 [-2.46882007  0.13095149]
 [-2.56231991  0.36771886]
 [-2.63953472  0.31203998]
 [-2.63198939 -0.19696122]
 [-2.58739848 -0.20431849]
 [-2.4099325   0.41092426]
 [-2.64886233  0.81336382]
 [-2.59873675  1.09314576]
 [-2.63692688 -0.12132235]
 [-2.86624165  0.06936447]
 [-2.62523805  0.59937002]
 [-2.80068412  0.26864374]
 [-2.98050204 -0.48795834]
 [-2.59000631  0.22904384]
 [-2.77010243  0.26352753]
 [-2.84936871 -0.94096057]
 [-2.99740655 -0.34192606]
 [-2.40561449  0.18887143]
 [-2.20948924  0.43666314]
 [-2.71445143 -0.2502082 ]
 [-2.53814826  0.50377114]
 [-2.83946217 -0.22794557]
 [-2.54308575  0.57941002]
 [-2.70335978  0.10770608]
 [ 1.28482569  0.68516047]
 [ 0.93248853  0.31833364]
 [ 1.46430232  0.50426282]
 [ 0.18331772 -0.82795901]
 [ 1.08810326  0.07459068]
 [ 0.64166908 -0.41824687]
 [ 1.09506066  0.28346827]
 [-0.74912267 -1.00489096]
```

```
# PCA implementation without using Library.
```

```
import numpy as np
```

```
input=np.array([[1.5,2.4],[2.3,5.2],[0.5,0.7],[1.2,1.9],[1.6,1.4]])
```

```
input
```

```
array([[1.5, 2.4],
       [2.3, 5.2],
       [0.5, 0.7],
       [1.2, 1.9],
       [1.6, 1.4]])
```

```
mean_values=input.mean(axis=0)
```

```
print(mean_values)
```

```
[1.42 2.32]
```

```
zeromean=input-mean_values
```

```
print(zeromean)
```

```
[[ 0.08  0.08]
 [ 0.88  2.88]
 [-0.92 -1.62]
 [-0.22 -0.42]
 [ 0.18 -0.92]]
```

```
cov=np.cov(zeromean.T)
```

```
print(cov)
```

```
[[0.427  0.9895]
 [0.9895 2.987 ]]
```

```
eigval,eigvect=np.linalg.eig(cov)
```

```
print(eigval)
```

```
[0.08912786 3.32487214]
```

```
id=eigval.argsort()[::-1]
```

```
eigvect=eigvect[:,id]
```

```
print(eigvect)
```

```
[[-0.32313879 -0.94635159]
 [-0.94635159  0.32313879]]
```

```
newigvect=eigvect.T
```

```
print(newigvect)
```

```
[[-0.32313879 -0.94635159]
 [-0.94635159  0.32313879]]
```

```
newzeromean=zeromean.T
```

```
newzeromean
```

```
array([[ 0.08,  0.88, -0.92, -0.22,  0.18],
       [ 0.08,  2.88, -1.62, -0.42, -0.92]])
```

```
finaldata=newigvect.dot(newzeromean)
```

```
print(finaldata.T)
```

```
[[-0.10155923 -0.04985702]
 [-3.0098547   0.09785032]
 [ 1.83037725  0.34715862]
```

```
         [ 0.4685582   0.07247906]
         [ 0.81247848 -0.46763097]]
```

● ✕

```
Experiment No:2
Title: Implementation of linear regression.
Name:Adityaraj Hari Ingavale
Roll No:08
Batch:A1
```

```
# LR implementation using Library.
```

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

```
x=np.array([[1],[2],[3],[4],[5]])
y=np.array([[3],[4],[2],[4],[5]])
```

```
model=LinearRegression().fit(x,y)
```

```
model.score(x,y)
```

```
     0.3076923076923075
```

```
print(model.coef_)
print(model.intercept_)
```

```
     [[0.4]]
     [2.4]
```

```
print(model.predict([[8]]))
```

```
     [[5.6]]
```

```
# LR implementation without using Library.
```

```
import numpy as np
```

```
x=np.array([[1],[2],[3],[4],[5]])
y=np.array([[3],[4],[2],[4],[5]])
```

```
xmean=np.mean(x)
```

```
ymean=np.mean(y)
```

```
xmval=x-xmean
```

```
ymval=y-ymean
```

```
sqxmval=(xmval**2)

xs=sum(sqxmval)

mul=xmval*ymval

m=sum(mul)/xs

c=ymean-(m*xmean)

xnew =8

ynew=(m*xnew)+c

ynew
```

```
     array([5.6])
```

✓ 0s completed at 10:54 PM ● ×

```
Experiment No:3
Title: Implementation of Logistic regression.
Name:Adityaraj·Hari·Ingavale
Roll·No:08
Batch:A1
```

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
```

```
input=load_breast_cancer()
input
```

```
{'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) dataset
 'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'data_module': 'sklearn.datasets.data',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'mean area
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension'], dtype='<U23'),
 'filename': 'breast_cancer.csv',
 'frame': None,
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'target_names': array(['malignant', 'benign'], dtype='<U9')}
```

```
print(input['DESCR'])
```

```
             - fractal dimension ("coastline approximation" - 1)

             The mean, standard error, and "worst" or largest (mean of the three
             worst/largest values) of these features were computed for each image,
             resulting in 30 features.  For instance, field 0 is Mean Radius, field
             10 is Radius SE, field 20 is Worst Radius.

             - class:
                     - WDBC-Malignant
                     - WDBC-Benign

         :Summary Statistics:

         ===================================== ====== ======
                                                Min    Max
         ===================================== ====== ======
         radius (mean):                        6.981  28.11
         texture (mean):                       9.71   39.28
         perimeter (mean):                     43.79  188.5
         area (mean):                          143.5  2501.0
         smoothness (mean):                    0.053  0.163
         compactness (mean):                   0.019  0.345
         concavity (mean):                     0.0    0.427
         concave points (mean):                0.0    0.201
         symmetry (mean):                      0.106  0.304
         fractal dimension (mean):             0.05   0.097
         radius (standard error):              0.112  2.873
         texture (standard error):             0.36   4.885
         perimeter (standard error):           0.757  21.98
         area (standard error):                6.802  542.2
         smoothness (standard error):          0.002  0.031
         compactness (standard error):         0.002  0.135
         concavity (standard error):           0.0    0.396
         concave points (standard error):      0.0    0.053
         symmetry (standard error):            0.008  0.079
         fractal dimension (standard error):   0.001  0.03
         radius (worst):                       7.93   36.04
         texture (worst):                      12.02  49.54
         perimeter (worst):                    50.41  251.2
         area (worst):                         185.2  4254.0
         smoothness (worst):                   0.071  0.223
         compactness (worst):                  0.027  1.058
         concavity (worst):                    0.0    1.252
         concave points (worst):               0.0    0.291
         symmetry (worst):                     0.156  0.664
         fractal dimension (worst):            0.055  0.208
```

```
        ===================================== ====== ======

        :Missing Attribute Values: None

        :Class Distribution: 212 - Malignant, 357 - Benign

        :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

        :Donor: Nick Street

        :Date: November, 1995

    This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
```

```
x=input.data
x
```

```
    array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
            1.189e-01],
           [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
            8.902e-02],
           [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
            8.758e-02],
           ...,
           [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
            7.820e-02],
           [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
            1.240e-01],
           [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
            7.039e-02]])
```

```
y=input.target
y
```

```
    array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
           0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
           1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
           1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
           1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
           0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
           1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
           0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
           1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
           1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
           0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
           0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
           1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
           1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
           1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
           1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
           1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
           1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

```
model=LogisticRegression().fit(x,y)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
test=model.predict([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,0, 0
```

```
test
```

```
array([0])
```

Experiment No:4
Title: Implementation of KNN Algorithm.
Name:Adityaraj Hari Ingavale
Roll No:08
Batch:A1

```
import numpy as np
import pandas as pd
```

```
dataset = pd.read_csv('/content/Social_Network_Ads.csv')
```

```
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state =
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test, y_pred)
```

```
cm
```

```
array([[55,  3],
       [ 1, 21]])
```

```
ac
```

```
0.95
```

```
Experiment No:5
Title: Implementation of Decision Tree.
Name:Adityaraj Hari Ingavale
Roll No:08
Batch:A1
```

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
pima = pd.read_csv("/content/pima-indians-diabetes.csv", header=None)
```

```
pima.head()
```

1 to 5 of 5 entries Filter

index	0	1	2	3	4	5	6	7	8
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

```
X =pima.iloc[:,[1,2,3]].values
Y=pima.iloc[:,-1].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=1)
model = DecisionTreeClassifier()
model = model.fit(X_train,y_train)
y_pred = model.predict(X_test)
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6233766233766234
```

```
from sklearn.datasets import load_iris
from sklearn import tree
```

```
dataset=load_iris()
x=dataset.data
y=dataset.target
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
model=tree.DecisionTreeClassifier().fit(x,y)
```

```
tree.plot_tree(model)
```

```
model.predict([[2,2,2,2]])
```

```
array([0])
```

✓ 0s completed at 11:01 PM ● ×

Experiment No:6
Title: Implementation of Naive Bayesian classifier.
Name:Adityaraj Hari Ingavale
Roll No:08
Batch:A1

```
import numpy as np
from sklearn.naive_bayes import GaussianNB
x= np.array([[-1,-1],[-2,-1],[-3,-2],[1,1],[2,1],[3,2]])
y=np.array ([1,1,1,2,2,2])
clf=GaussianNB()
clf.fit(x,y)
print(clf.predict([[2,2]]))
```

```
[2]
```

```
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Ho
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
weather_encoded=le.fit_transform(weather)
print (weather_encoded)
```

```
[2 2 0 1 1 1 0 2 2 1 2 0 0 1]
```

```
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)
print ("Temp:", temp_encoded)
print ("Play:",label)
```

```
Temp: [1 1 1 2 0 0 0 2 0 2 2 2 1 2]
Play: [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

```
features=list(zip(weather_encoded,temp_encoded))
features
```

```
[(2, 1),
 (2, 1),
 (0, 1),
 (1, 2),
 (1, 0),
 (1, 0),
 (0, 0),
 (2, 2),
 (2, 0),
 (1, 2),
 (2, 2),
 (0, 2),
     (0, 1),
     (1, 2)]
```

```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
model.fit(features,label)
```

```
    GaussianNB()
```

```
predicted= model.predict([[0,2]])
print ("Predicted Value:", predicted)
```

```
    Predicted Value: [1]
```

```
Experiment No:7
Title: Implementation of Bayesian network.
Name:Adityaraj Hari Ingavale
Roll No:08
Batch:A1
```

```
pip install pomegranate
```

```
Collecting pomegranate
  Downloading pomegranate-0.14.8.tar.gz (4.3 MB)
     |████████████████████████████████| 4.3 MB 21.7 MB/s
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... done
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (fron
Requirement already satisfied: joblib>=0.9.0b4 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: numpy>=1.20.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.7/dist-package
Building wheels for collected packages: pomegranate
  Building wheel for pomegranate (PEP 517) ... done
  Created wheel for pomegranate: filename=pomegranate-0.14.8-cp37-cp37m-linux_x86_64
  Stored in directory: /root/.cache/pip/wheels/24/68/69/0eaab474ef1d65abedcd47de8a38a
Successfully built pomegranate
Installing collected packages: pomegranate
Successfully installed pomegranate-0.14.8
```

```
import math
from pomegranate import *
```

```
guest =DiscreteDistribution( { 'A': 1./3, 'B': 1./3, 'C': 1./3 } )
prize =DiscreteDistribution( { 'A': 1./3, 'B': 1./3, 'C': 1./3 } )
```

```
monty =ConditionalProbabilityTable(
[[ 'A', 'A', 'A', 0.0 ],
[ 'A', 'A', 'B', 0.5 ],
[ 'A', 'A', 'C', 0.5 ],
[ 'A', 'B', 'A', 0.0 ],
[ 'A', 'B', 'B', 0.0 ],
[ 'A', 'B', 'C', 1.0 ],
[ 'A', 'C', 'A', 0.0 ],
[ 'A', 'C', 'B', 1.0 ],
[ 'A', 'C', 'C', 0.0 ],
[ 'B', 'A', 'A', 0.0 ],
[ 'B', 'A', 'B', 0.0 ],
[ 'B', 'A', 'C', 1.0 ],
[ 'B', 'B', 'A', 0.5 ],
[ 'B', 'B', 'B', 0.0 ],
[ 'B', 'B', 'C', 0.5 ],
[ 'B', 'C', 'A', 1.0 ],
[ 'B', 'C', 'B', 0.0 ],
[ 'B', 'C', 'C', 0.0 ],
[ 'C', 'A', 'A', 0.0 ],
[ 'C', 'A', 'B', 1.0 ],
[ 'C', 'A', 'C', 0.0 ],
[ 'C', 'B', 'A', 1.0 ],
[ 'C', 'B', 'B', 0.0 ],
[ 'C', 'B', 'C', 0.0 ],
[ 'C', 'C', 'A', 0.5 ],
[ 'C', 'C', 'B', 0.5 ],
[ 'C', 'C', 'C', 0.0 ]], [guest, prize] )


d1 = State( guest, name="guest" )
d2 = State( prize, name="prize" )
d3 = State( monty, name="monty" )


network = BayesianNetwork( "Solving the Monty Hall Problem With Bayesian Networks" )
network.add_states(d1, d2, d3)
network.add_edge(d1, d3)
network.add_edge(d2, d3)
network.bake()


beliefs = network.predict_proba({ 'guest' : 'A' })
beliefs = map(str, beliefs)
print("\n".join( "{}\t{}".format( state.name, belief ) for state, belief in zip( network.s
```

```
guest   A
prize   {
    "class" : "Distribution",
    "dtype" : "str",
    "name" : "DiscreteDistribution",
    "parameters" : [
        {
            "A" : 0.3333333333333333,
            "B" : 0.3333333333333333,
            "C" : 0.3333333333333333
        }
    ],
    "frozen" : false
}
monty   {
    "class" : "Distribution",
    "dtype" : "str",
    "name" : "DiscreteDistribution",
    "parameters" : [
        {
            "B" : 0.49999999999999983,
            "A" : 0.0,
            "C" : 0.49999999999999983
        }
    ],
    "frozen" : false
}
```

```
beliefs = network.predict_proba({'guest' : 'A', 'monty' : 'B'})
print("\n".join( "{}\t{}".format( state.name, str(belief) ) for state, belief in zip( netw
```

```
guest   A
prize   {
    "class" : "Distribution",
    "dtype" : "str",
    "name" : "DiscreteDistribution",
    "parameters" : [
        {
            "A" : 0.3333333333333334,
            "B" : 0.0,
            "C" : 0.6666666666666664
        }
    ],
    "frozen" : false
}
monty   B
```

Experiment No:9
Title: Implementation of k-Means Algorithm.
Name:Adityaraj Hari Ingavale
Roll No:08
Batch:A1

```
import numpy as np
import pandas as pd
```

```
x = np.array([[110, 32, 22],[ 34, 334,45], [57, 34, 78]])
```

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters = 3, random_state = 1)
model.fit(x)
```

```
     KMeans(n_clusters=3, random_state=1)
```

```
print(model.cluster_centers_)
```

```
     [[ 34. 334.  45.]
      [ 57.  34.  78.]
      [110.  32.  22.]]
```

```
x_p = ([[10, 5, 32]])
```

```
import matplotlib.pyplot as plt
```

```
plt.scatter(x[:,[0]],x[:,[1]],x[:,[2]],c=model.labels_,cmap='rainbow')
```

Experiment No:8
Title: Implementation of GM Algorithm.
Name:Adityaraj Hari Ingavale
Roll No:08
Batch:A1

+ Code + Text

```
import numpy as np
import pandas as pd
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt
```

```
dataset = pd.read_csv("/content/pima-indians-diabetes.csv")
```

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

```
array([[ 85,  66,  29],
       [183,  64,   0],
       [ 89,  66,  23],
       ...,
       [121,  72,  23],
       [126,  60,   0],
       [ 93,  70,  31]])
```

```
model = GaussianMixture(n_components=3, random_state= 1).fit(x)
```

```
y = model.predict([[10, 5, 32]])
y
```

```
array([2])
```

```
print(model.means_)
print(model.weights_)
```

```
[[106.74045882  69.58243931  21.17694706]
 [154.77036428  78.94620956  21.79329546]
 [102.20956104   9.91831666   5.83815651]]
[0.64584798 0.2991461  0.05500592]
```

```
plt.scatter(x[:,[0]],x[:,[1]],x[:,[2]],c=model.weights_,cmap='rainbow')
```

✓ 4s completed at 11:17 PM ● ✕