EXPERIMENT – 1

AIM : Write a program to implement Linear Regression using Scikit-Learn

CODE :

```
from sklearn.linear_model import LinearRegression

import numpy as np

X = np.array([[1], [2], [3], [4], [5]])

y = np.array([2, 4, 6, 8, 10])

model = LinearRegression()

model.fit(X, y)

pred = model.predict([[6]])

print("Predicted value for 6:", pred[0])
```

OUTPUT:

```
Predicted value for 6: 12.0
```

EXPERIMENT - 2

AIM: Logistic Regression Classification

CODE :

```
from sklearn.linear_model import LogisticRegression

X = [[1], [2], [3], [4]]

y = [0, 0, 1, 1]

model = LogisticRegression()

model.fit(X, y)

print('Prediction for 2.5:', model.predict([[2.5]])[0])
```

OUTPUT:

```
Prediction for 2.5: 0
```

EXPERIMENT - 3

AIM: Write a program to find the K-Nearest Neighbors (KNN)

CODE:

```
from sklearn.neighbors import KNeighborsClassifier

X = [[1,2], [2,3], [3,4], [6,7]]

y = [0, 0, 1, 1]

model = KNeighborsClassifier(n_neighbors=3)

model.fit(X, y)

print('Class for [4,5]:', model.predict([[4,5]])[0])
```

OUTPUT:

```
Class for [4,5]: 1
```

EXPERIMENT – 4

AIM: Write a program to implement Decision Tree Classifier

CODE:

```
from sklearn.tree import DecisionTreeClassifier

X = [[0,0], [1,1], [2,2], [3,3]]

y = [0, 0, 1, 1]

model = DecisionTreeClassifier()

model.fit(X, y)

print('Prediction for [1.5,1.5]:', model.predict([[1.5,1.5]])[0])
```

OUTPUT:

```
Prediction for [1.5,1.5]: 0
```

EXPERIMENT – 5

AIM: Write a program to implement K - Means Clustering

CODE:

```
from sklearn.cluster import KMeans

import numpy as np

X = np.array([[1,2], [1,4], [1,0], [10,2], [10,4], [10,0]])

kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

print('Cluster centers:', kmeans.cluster_centers_)
```

OUTPUT:

```
Cluster centers: [[ 1. 2.] [10. 2.]]
```

EXPERIMENT – 6

AIM: Write a program to implement Naive Bayes Classification

CODE:

```
from sklearn.naive_bayes import GaussianNB

X = [[1,2], [2,3], [3,4], [4,5]]

y = [0, 0, 1, 1]

model = GaussianNB()

model.fit(X, y)

print('Prediction for [2,2]:', model.predict([[2,2]])[0])
```

OUTPUT:

```
Prediction for [2,2]: 0
```

EXPERIMENT – 7

AIM: Write a program to implement PCA (Principal Component Analysis) on Iris Dataset

CODE:

```
from sklearn.datasets import load_iris

from sklearn.decomposition import PCA

import pandas as pd

iris = load_iris()

X = iris.data

y = iris.target

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X)


print('Original shape:', X.shape)

print('Transformed shape:', X_pca.shape)

print('First 5 PCA values:\n', X_pca[:5])
```

OUTPUT:

```
Original shape: (150, 4) Transformed shape: (150, 2) First 5 PCA values: [[ -2.68412563

0.31939725] [ -2.71414169 -0.17700123] [ -2.88899057 -0.14494943] [ -2.74534286 -
0.31829898] [

-2.72871654 0.32675451]]
```

EXPERIMENT – 8

AIM: Write a program to implement DBSCAN Clustering Algorithm

CODE:

```
from sklearn.cluster import DBSCAN

import numpy as np

# Sample data with two dense regions

X = np.array([[1,2], [1,4], [1,0], [10,2], [10,4], [10,0]])

dbscan = DBSCAN(eps=1.5, min_samples=2).fit(X)

print('Labels:', dbscan.labels_) # -1 means noise
```

OUTPUT:

```
Labels: [0 0 0 1 1 1]
```

EXPERIMENT – 9

AIM: Write a program to implement K-Medoids Clustering Algorithm (K-Medoid)

CODE:

```
# K-Medoids requires scikit-learn-extra: pip install scikit-learn-extra

from sklearn_extra.cluster import KMedoids

import numpy as np

X = np.array([[1,2], [1,4], [1,0], [10,2], [10,4], [10,0]])

kmedoids = KMedoids(n_clusters=2, random_state=0).fit(X)

print('Medoid indices:', kmedoids.medoid_indices_)

print('Labels:', kmedoids.labels_)
```

OUTPUT:

```
Medoid indices: [0 3] Labels: [0 0 0 1 1 1]
```

EXPERIMENT – 10

AIM: Program 10: LDA (Linear Discriminant Analysis) on Iris Dataset

CODE:

```
from sklearn.datasets import load_iris
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
iris = load_iris()
X = iris.data
y = iris.target
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X, y)
print('Original shape:', X.shape)
print('Transformed shape:', X_lda.shape)
print('First 5 LDA values:\n', X_lda[:5])
```

OUTPUT:

```
Original shape: (150, 4) Transformed shape: (150, 2) First 5 LDA values: [[ 8.06179978
0.30042062] [ 7.12868772 0.78666043] [ 7.48982894 0.26538449] [ 6.81320057
0.67063107] [8.13230933 -0.51446253]]
```