EXPERIMENT-1

WEEK -1: MATRIX OPERATIONS

AIM: To perform various matrix operations.

a) Create multi-dimensional arrays and find its shape and dimension

b) Create a matrix full of zeros and ones

c) Reshape and flatten data in the array

d) Append data vertically and horizontally

e) Apply indexing and slicing on array

f) Use statistical functions on array-Min, Max, Mean, Median and Standard Deviation

RESOURCES:

a) Python 3.7.0
b) Install: pip installer, Pandas library

PROCEDURE:

1. Create: Open a new file in Python shell, write a program and save the program with .py extension. 2. Execute: Goto Run->Run module(F5)

PROGRAM LOGIC:

a) Create multi-dimensional arrays and find its shape and dimension

```
Import numpy as np
#creationofmulti-dimensionalarray
a=np.array ([[1,2,3],[2,3,4],[3,4,5]])

 #shape

 b=a.shape
print("shape:",a.shape)
```

#dimensio
n c=a.ndim
print("dimensions:",a.ndim)

OUTPUT:

shape:(3,3)
dimensions:2

b) Create a matrix full of zeros and ones

#matrixfullofzeros

z=np.zeros((2,2))

print("zeros:",z)

#matrixfullofones

o=np.ones((2,2))
print("ones:",o)

OUTPUT:

zeros:[[0.0.]

a. 0.]]
ones:[
[1. 1.]
b. 1.]]

c) Reshape and flatten data in the array

#matrixreshape

```
a=np.array([[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7]])
b=a.reshape(4,2,2)
print("reshape:",b)
```

#matrix

```
flattenc=a.flatten()
print("flatten: ",c)
```

OUTPUT:

```
reshape:[[[12]
[34]]

[[23]
[45]]

[[34]
[56]]

[[45]
[6 7]]]
flatten:[1234234534564567]
```

Regd. No

d) Append data vertically and horizontally

#Appending data vertically

```
x=np.array([[10,20],[80,90]])
y=np.array([[30,40],[60,70]])
v=np.vstack((x,y))
print("vertically:",v)
```

#Appendingdatahorizontally

```
h=np.hstack((x,y))
```

```
print("horizontally:",h
)
```

OUTPUT:

vertically:[[1020]

[80 90]

[3040] [6070]]

horizontally:[[10203040]

[809060 70]]

e) Apply indexing and slicing on array

#indexing

```
a=np.array([[1,2,3,4],[
[2,3,4,5],[3,4,5,6],[4,5,6,7]])t
emp=a[[0,1,2,3],[1,1,1,1]]
```

```
print('indexing',temp)
#slicing i=a[:4,::2] print('slicing',i)
```

OUTPUT:

indexing[2 3 4 5]

slicing[[1 3]

[2 4]

[3 5]

[4 6]]

f) Use statistical functions on array-Min, Max, Mean, Median and Standard Deviation

```
#minforfindingminimumofanarray
a=np.array([[1,3,-1,4],[3,-2,1,4]])
b=a.min()
print('minimum'
',b)
#maxforfindingmaximumofanarray

c=a.max() print('maximum',c)
a=np.array([1,2,3,4,5])
d=a.mean ()
print('mean:',d)

#median
```

```
e=np.median( a)
    print('median:',e)

    #standarddeviation
f=a.std()
```

```
print('standarddeviation',f)
```

OUTPUT:

minimum:-2

maximum 4

mean:3.0

median:3.0

standarddeviation1.4142135623730951

EXPERIMENT -5

WEEK - 5: DATA PREPROCESSING-HANDLING MISSING VALUES

Write a python program to input missing values with various techniques on given dataset.

a) Remove rows/attributes
b) Replace with mean or mode
c) Write a python program to perform transformation of data using Discretization (Binning) and normalization(Min Max Scale or Max Ab sScaler) on given dataset.

```
import pandas as pd

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import KBinsDiscretizer, MinMaxScaler, MaxAbsScaler
```

 # Load your dataset

```
data = {
   'A': [1, 2, None, 4, 5],
   'B': [None, 2, 3, None, 5],
   'C': [1, 2, 3, 4, 5]
}
df = pd.DataFrame(data)
```

 # Display the original dataframe

```
print("Original DataFrame:") print(df)
```

 # Handling missing values

```
def handle_missing_values(df, method='mean'): if method ==
   'remove':
```

```
df = df.dropna()
```

 # Remove rows with missing valueselse:
if method == 'mean':

imputer = SimpleImputer(strategy='mean') elif method

== 'mode': imputer =

SimpleImputer(strategy='most_frequent') df[df.columns]

= imputer.fit_transform(df[df.columns])

return df

 # Apply missing value handling function

 # You can change the method parameter to 'remove', 'mean', or 'mode'

df_handled = handle_missing_values(df, method='mean')

 # Display the dataframe after handling missing values

print("\nDataFrame After Handling Missing Values:")

print(df_handled)

 # Data Transformation def data_transformation(df,

method='binning'): if method == 'binning':

est = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')

df[df.columns] = est.fit_transform(df[df.columns])

elif method == 'normalization': scaler

= MinMaxScaler()

df[df.columns]

You can also use MaxAbsScaler for MaxAbs

normalization scaler.fit_transform(df[df.columns]) return df

 # Apply data transformation function

 # You can change the method parameter to 'binning' or 'normalization'

df_transformed = data_transformation(df_handled, method='normalization')

 # Display the dataframe after data transformation

print("\nDataFrame After Data Transformation:")

print(df_transformed)

OUTPUT:

Original DataFrame:
A B C
0 1.0 NaN 1
1 2.0 2.0 2
2 NaN 3.0 3
3 4.0 NaN 4
4 5.0 5.0 5

DataFrame After Handling Missing Values: A
B C
0 1.0 3.333333 1.0 1
2.0 2.000000 2.0
2 3.0 3.000000 3.0
3 4.0 3.333333 4.0
4 5.0 5.000000 5.0

DataFrame After Data Transformation: A
B C

0 0.44444 0.0
0.00 4 0

1 0.00000 0.2
0.25 0 5

2 0.33333 0.5
0.50 3 0

3 0.44444 0.7
0.75 4 5

4 1.00000 1.0
1.00 0 0

[ ]

EXPERIMENT – 7

WEEK -7: CLASSIFICATION-LOGISTIC REGRESSION

LOGISTIC REGRESSION

A statistical model for binary classification is called logistic regression. Using the sigmoid function, it forecasts the likelihood that an instance will belong to a particular class, guaranteeing results between 0 and 1. To minimize the log loss, the model computes a linear combination of input characteristics, transforms it using the sigmoid, and then optimizes its coefficients using methods like gradient descent. These coefficients establishthe decision boundary that divides the classes. Because of its ease of use, interpretability, and versatility across multiple domains, Logistic Regression is widely used in machine learning for problems that involve binary outcomes. Overfitting can be avoided by implementing regularization.

SOURCE CODE :

 # Import necessary libraries

Import numpy as np import

pandas as pd import

matplotlib.pyplot as plt

import seaborn as sns fromsklearn.datasets import load_diabetes from

sklearn.model_selection import train_test_split from sklearn.preprocessing

import StandardScaler from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report,

confusion_matrix, roc_curve, auc

 # Load the diabetes dataset

diabetes = load_diabetes()

X, y = diabetes.data, diabetes.target

 # Convert the target variable to binary (1 for diabetes, 0 for no diabetes) y_binary
= (y >np.median(y)).astype(int)
 # Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y_binary,

test_size=0.2, random_state=42)

 # Standardize features

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train) X_test scaler.transform(X_test)

 # Train the Logistic Regression model

model = LogisticRegression() model.fit(X_train, y_train)

 # Evaluate the model

y_pred = model.predict(X_test) accuracy =

accuracy_score(y_test, y_pred) print("Accuracy:

{:.2f}%".format(accuracy * 100))

 # evaluate the model

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred)) **#**

Visualize the decision boundary with accuracy information

plt.figure(figsize=(8, 6))sns.scatterplot(x=X_test[:, 2], y=X_test[:, 8],

hue=y_test, palette={ 0: 'blue', 1: 'red'}, marker='o')

plt.xlabel("BMI")

plt.ylabel("Age")

plt.title("Logistic Regression Decision Boundary\nAccuracy: {:.2f}%".format(accuracy

* 100))

plt.legend(title="Diabetes", loc="upper right")

plt.show()

 # Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(

X, y_binary, test_size=0.2, random_state=42)

 # Standardize features

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train) X_test = scaler.transform(X_test)

 # Train the Logistic Regression model
model = LogisticRegression() model.fit(X_train,
y_train)

 # Evaluate the model
y_pred = model.predict(X_test) accuracy =
accuracy_score(y_test, y_pred) print("Accuracy:
{:.2f}%".format(accuracy * 100))

 # evaluate the model

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred))

 # Visualize the decision boundary with accuracy information
plt.figure(figsize=(8, 6))

sns.scatterplot(x=X_test[:, 2], y=X_test[:, 8],
hue=y_test, palette={0: 'blue', 1: 'red'}, marker='o')
plt.xlabel("BMI") plt.ylabel("Age")

plt.title("Logistic Regression Decision Boundary\nAccuracy: {:.2f}%".format(accuracy *
100))plt.legend(title="Diabetes", loc="upper right") plt.show()

Accuracy: 73.03% Confusion
Matrix:[[36 13][11 29]]

Classification Report:

precision recall f1-score support

0 0.77 0.73 0.75 49
0.69 0.72 0.71 40

89 8

0.73 macroavg 0.73

0.73 0.73 9

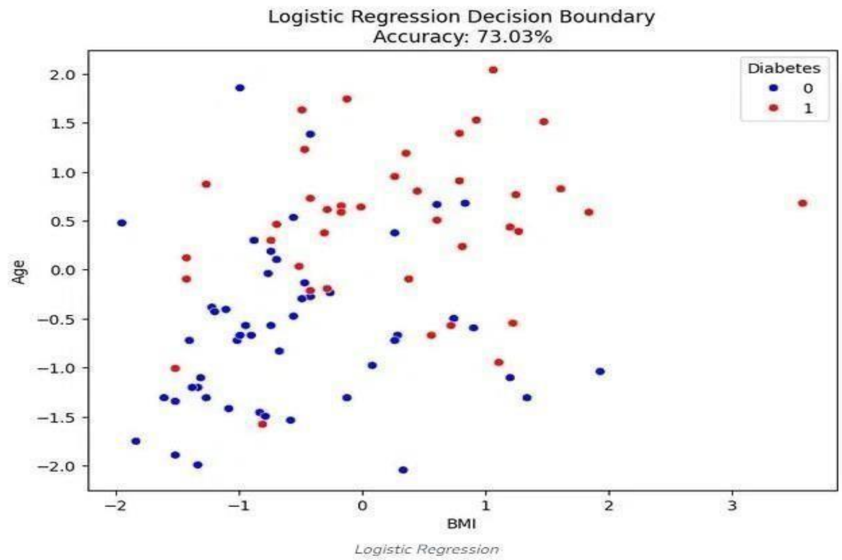
0.7

weighteda 0.7 0.73 3

vg 3

89

Output:



EXPERIMENT – 8

WEEK -8 : CLASSIFICATION-KNN ALGORITHM

K-NEAREST NEIGHBOR ALGORITHM:

This algorithm is used to solve the classification model problems. K-nearest neighbor or K-NN algorithm basically creates an imaginary boundary to classifythe data. When new data points come in, the algorithm will try to predict that to the nearest of the boundary line.

Therefore, larger k value means smother curves of separation resulting in less complex models. Whereas, smaller k value tends to overfit the data and resultingin complex models.

Note: It's very important to have the right k-value when analyzing the datasetto avoid overfitting and algorithm we fit the historical data (or train the model) and predict the future.

Here in the example shown above, we are creating a plot to see the k-value forwhich we have high accuracy.

Note: This is a technique which is not used industry-wide to choose the correct value of n-neighbors. Instead, we do hyperparameter tuning to choosethe value that gives the best performance. We will be covering this in future posts

<u>Source code:</u>

 # Import necessary modules from sklearn.neighbors

import kNeighborsClassifier from sklearn.model_selection
import train_test_splitfrom sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as pltirisData = load_iris()

 # Create feature and target arrays

X = irisData.data y
= irisData.target

 # Split into training and test set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)

neighbors = np.arange(1, 9) train_accuracy = np.empty(len(neighbors)) test_accuracy
= np.empty(len(neighbors))

 # Loop over K values for i, k in enumerate(neighbors):

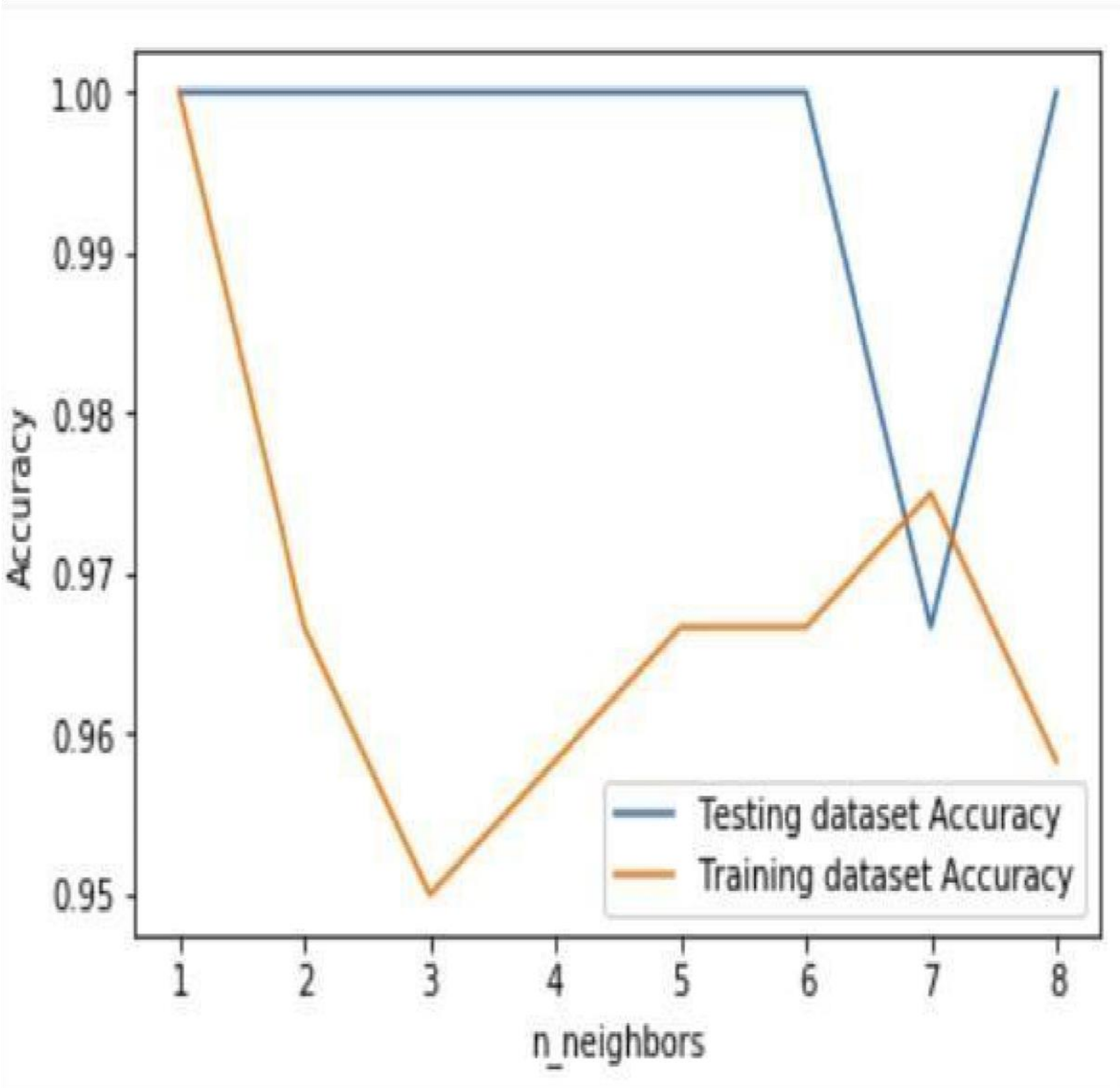
knn = KNeighborsClassifier(n_neighbors=k)knn.fit(X_train, y_train)

 # Compute training and test data accuracy
train_accuracy[i] = knn.score(X_train, y_train)test_accuracy[i] = knn.score(X_test, y_test)

 # Generate plot

plt.plot(neighbors, test_accuracy, label = 'Testing datasetAccuracy')
plt.plot(neighbors, train_accuracy, label = 'Training datasetAccuracy')
plt.legend()
plt.xlabel('n_neighbors')plt.ylabel('Accuracy') plt.show()

OUTPUT:


1.00
0.99
0.98
0.97
0.96
0.95
Accuracy
1
2
3
4
5
6
7
8
n_neighbors
Testing dataset Accuracy
Training dataset Accuracy