

## Python Programming for Machine Learning

**Name: NAVEEN.A**

**Class: ECE 'C'**

**Roll Number: 220801135**

Sl.No	Date	Name of the experiment	Signature
1.	16/2/2024	Calculating values of random data using NumPy for mathematical formulas 1)Euclidean distance between two points 2) Dot Product of two Vectors 3)Solving a System of Linear Equations	
2.	23/2/2024	Write a simple Python code to generate random values and then compute their sigmoid and tanh (hyperbolic tangent) values using NumPy. Plot the values.	
3.	2/3/2024	simple Python program using pandas that creates a DataFrame, performs some basic operations, and prints the result.	
4.	23/3/2024	Store and Load Excel / CSV files.	
5.	28/3/2024	Data Visualization	
6.	6/4/2024	Time Series	
7.	12/4/2024	Linear regression model to predict the signal strength	
8.	26/4/2024	A component is defective or not based on Voltage and Current	
9.	3/5/2024	Decision tree classifier to predict signal quality based on transmitter, signal strength, and frequency	
10.	11/5/2024	k-NN classifier to predict signal quality based on distance from the transmitter, signal strength, and frequency	
11.	17/5/2024	Study of Artificial Neural Network (ANN) and Simple Program in ANN	
12.	24/5/2024	Study Of Support Vector Machine and and Simple Program in SVM	

**Ex.no 1**

**Calculating values of random data using NumPy for  
Date: 16.2.2024 mathematical formulas**

**220801135**

**Aim:**

Calculating values of random data using NumPy for mathematical formulas

1)Euclidean distance between two points 2) Dot Product of two Vectors 3)Solving a System of Linear Equations

**Program:**

```
#euclidean distance between 2 points
point1=np.array([3,2])
point2=np.array([1,1])
d=((point1[0]-point2[0])**2)+((point1[1]-point2[1])**2)
print(math.sqrt(d))

#dot product
def dot(x,y):
    dot_prod=x.dot(y)
    print(dot_prod)
dot(point1, point2)

#Linear equation
a=np.array([[4,3], [5, 9]])
b=np.array([2,1])
print(np.linalg.solve(a,b))
```

**Output:**

```
In [31]: #euclidean distance between 2 points
point1=np.array([3,2])
point2=np.array([1,1])
d=((point1[0]-point2[0])**2)+((point1[1]-point2[1])**2)
print(math.sqrt(d))
2.23606797749979

In [36]: #euclidean
def euclidean(x,y):
    dist=np.sum((x-y)**2)
    print(math.sqrt(dist))
euclidean(point1,point2)
2.23606797749979

In [38]: #dot product
def dot(x,y):
    dot_prod=x.dot(y)
    print(dot_prod)
dot(point1,point2)
5

In [41]: #Linear equation
a = np.array([[4, 3], [5, 9]])
b = np.array([2,1])
print(np.linalg.solve(a,b))
[ 0.71428571 -0.28571429]
```

**Result:**

Thus the program has been done and executed and output has been verified successfully.

**Ex.no 2**  
**Date: 23/2/2024**

## **Sigmoid and Tanh**

**220801135**

### **Aim:**

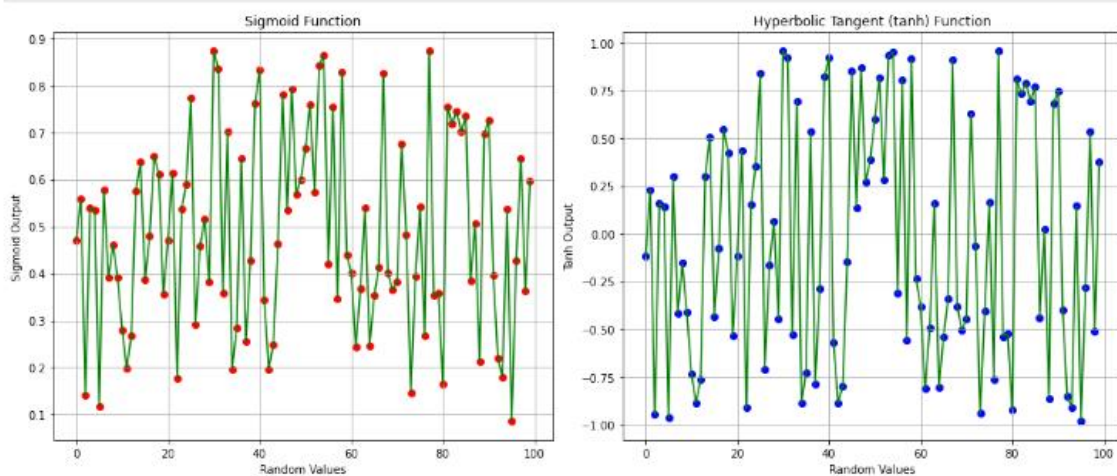
Write a simple Python code to generate random values and then compute their sigmoid and tanh (hyperbolic tangent) values using NumPy. Plot the values.

### **Program:**

```
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
def tanh(x):  
    return np.tanh(x)  
  
random_values=np.random.randn(100)  
sigmoid_values=sigmoid(random_values)  
tanh_values=tanh(random_values)  
  
#plotting  
indices=np.arange(len(random_values))  
plt.figure(figsize=(14, 6))  
plt.subplot(1,2,1)  
plt.scatter(indices, sigmoid_values, color='r', label='Sigmoid')  
plt.plot(indices,sigmoid_values,'g',linestyle='--')  
plt.title('Sigmoid Function')  
plt.xlabel('Random Values')  
plt.ylabel('Sigmoid Output')  
plt.grid()  
plt.subplot(1,2,2)  
plt.scatter(indices, tanh_values, color='b', label='Tanh')  
plt.plot(indices,tanh_values,'g',linestyle='--')  
plt.title('Hyperbolic Tangent (tanh) Function')  
plt.xlabel('Random Values')  
plt.ylabel('Tanh Output')  
plt.grid()  
plt.tight_layout()  
plt.show()
```

## Output:

```
j> def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
def tanh(x):  
    return np.tanh(x)  
random_values=np.random.randn(100)  
sigmoid_values=sigmoid(random_values)  
tanh_values=tanh(random_values)  
#plotting  
indices=np.arange(len(random_values))  
  
plt.figure(figsize=(14, 6))  
plt.subplot(1,2,1)  
plt.scatter(indices, sigmoid_values, color='r', label='Sigmoid')  
plt.plot(indices,sigmoid_values,'g',linestyle='-')  
plt.title('Sigmoid Function')  
plt.xlabel('Random Values')  
plt.ylabel('Sigmoid Output')  
plt.grid()  
  
plt.subplot(1,2,2)  
plt.scatter(indices, tanh_values, color='b', label='Tanh')  
plt.plot(indices,tanh_values,'g',linestyle='-')  
plt.title('Hyperbolic Tangent (tanh) Function')  
plt.xlabel('Random Values')  
plt.ylabel('Tanh Output')  
plt.grid()  
  
plt.tight_layout()  
plt.show()
```



## Result:

Thus the program has been done and executed and output has been verified successfully.

**Ex.no 3**

**Simple Program using Pandas**

**220801135**

**Date: 2/3/2024**

**Aim:**

Simple Python program using pandas that creates a DataFrame, performs some basic operations, and prints the result.

**Steps:**

1. Imports the pandas library as pd.
2. Creates two lists: data containing fruit names and prices containing their corresponding prices.
3. Zips these lists together and creates a DataFrame named fruits\_df with columns named 'Fruit' and 'Price'
4. Uses info() to get information about the DataFrame, including data types and number of entries.
5. Prints the entire DataFrame using to\_string().
6. Calculates descriptive statistics (mean, standard deviation, etc.) for the 'Price' column and prints the results.

**Program Code:**

```
import pandas as pd

# Create a list of data
data = ["Apple", "Banana", "Cherry", "Orange", "Grape"]
prices = [1.25, 0.79, 2.00, 1.50, 0.99]

# Create a DataFrame
fruits_df = pd.DataFrame(list(zip(data, prices)), columns=['Fruit', 'Price'])

# Get basic information about the DataFrame
print(fruits_df.info())

# Print the DataFrame
print(fruits_df.to_string())

# Get descriptive statistics of the 'Price' column
print(fruits_df['Price'].describe())
```

## Output:

```
[2] import pandas as pd
```



```
# Create a list of data
```

```
data = ["Apple", "Banana", "Cherry", "Orange", "Grape"]
```

```
prices = [1.25, 0.79, 2.00, 1.50, 0.99]
```

```
[7] # Create a DataFrame
```

```
fruits_df = pd.DataFrame(list(zip(data, prices)), columns = ['Fruit', 'Price'])
```

```
fruits_df
```



	Fruit	Price
0	Apple	1.25
1	Banana	0.79
2	Cherry	2.00
3	Orange	1.50
4	Grape	0.99



✓  
0s [8] # Get basic information about the DataFrame  
print(fruits\_df.info())

↵ <class 'pandas.core.frame.DataFrame'>  
RangeIndex: 5 entries, 0 to 4  
Data columns (total 2 columns):  
# Column Non-Null Count Dtype  
--- ---  
0 Fruit 5 non-null object  
1 Price 5 non-null float64  
dtypes: float64(1), object(1)  
memory usage: 208.0+ bytes  
None

✓  
0s [9] # Get descriptive statistics of the 'Price' column  
print(fruits\_df['Price'].describe())

↵ count 5.000000  
mean 1.306000  
std 0.471307  
min 0.790000  
25% 0.990000  
50% 1.250000  
75% 1.500000  
max 2.000000  
Name: Price, dtype: float64

## Result:

Thus the program has been done and executed and output has been verified successfully.

Date: 23/3/2024

**Aim:**

To store (save) and load data from Excel and CSV files using pandas.

**Steps:****To Store:**

- import pandas as pd.
- Create a sample DataFrame df.
- Use the to\_csv function to save the DataFrame to a CSV file.
- 'people.csv' is the filename.
- index=True (default) saves the row index as a column. Set it to False to skip it.

**To Load:**

- Import pandas as pd.
- Use read\_csv to load data from a CSV file.
- Use read\_excel to load data from an Excel file. By default, it reads the first sheet.
- Specify the sheet name with the sheet\_name argument for loading data from a specific
- Sheet.

**Program Code:****To store:**

```
import pandas as pd
```

```
# Sample data
```

```
data = {"Name": ["Alice", "Bob", "Charlie"], "Age": [25, 30, 22]}
```

```
df = pd.DataFrame(data)
```

```
# Save to CSV file (with index)
```

```
df.to_csv("people.csv", index=True)
```

```
# Save to CSV file (without index)
```

```
df.to_csv("people_no_index.csv", index=False)
```

**To Load:**

```
# Load CSV data (assuming it has a header row)
```

```
df_csv = pd.read_csv('people.csv')
```

```
print(df_csv)
```



### Output:

```
[10] import pandas as pd

# Sample data
data = {"Name": ["Alice", "Bob", "Charlie"], "Age": [25, 30, 22]}
df = pd.DataFrame(data)

# Save to CSV file (with index)
df.to_csv("people.csv", index=True)

# Save to CSV file (without index)
df.to_csv("people_no_index.csv", index=False)
```



```
# Load CSV data (assuming it has a header row)
df_csv = pd.read_csv("people.csv")
print(df_csv)
```



	Unnamed: 0	Name	Age
0	0	Alice	25
1	1	Bob	30
2	2	Charlie	22

### Result:

Thus the program has been done and executed and output has been verified successfully.

**Aim:**

To visualize the given data using the matplotlib library in python

**Algorithm:**

- Import the matplotlib.pyplot library for plotting.
- Prepare Data
- Use the plt.plot() function to create a line plot with cities on the x-axis and temperatures on the y-axis.
- Customize the plot by adding markers and setting the line style
- Add Labels and Title
- Use plt.show() to display the plot.

**Program:**

```
import matplotlib.pyplot as plt
import pandas as pd # Optional for data manipulation

# Sample data (replace with your data or use pandas to read a CSV)
temperatures = [15, 18, 22, 20, 17, 24, 21, 19]
cities = ["New York", "Los Angeles", "Chicago", "Denver", "Seattle", "Miami",
"Houston", "San Francisco"]

# Line plot
plt.plot(cities, temperatures, marker='o', linestyle='--') # Customize
markers and line style
# Labels and title
plt.xlabel("City")
plt.ylabel("Temperature (°C)")
plt.title("Average Temperatures in Major US Cities")

# Display the plot
plt.xticks(rotation=45) # Rotate city names for better readability
(optional)
plt.grid(True) # Add gridlines (optional)
plt.show()
```

## Output:

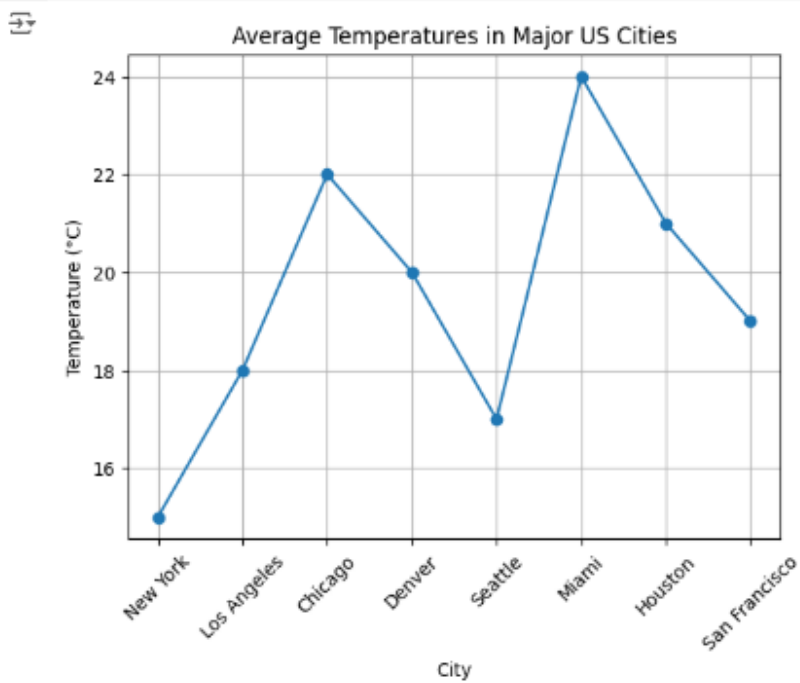
```
import matplotlib.pyplot as plt
import pandas as pd # Optional for data manipulation

# Sample data (replace with your data or use pandas to read a CSV)
temperatures = [15, 18, 22, 20, 17, 24, 21, 19]
cities = ["New York", "Los Angeles", "Chicago", "Denver", "Seattle", "Miami", "Houston", "San Francisco"]

# Line plot
plt.plot(cities, temperatures, marker='o', linestyle='--') # Customize markers and line style

# Labels and title
plt.xlabel("City")
plt.ylabel("Temperature (°C)")
plt.title("Average Temperatures in Major US Cities")

# Display the plot
plt.xticks(rotation=45) # Rotate city names for better readability (optional)
plt.grid(True) # Add gridlines (optional)
plt.show()
```



## Result:

Thus the program has been done and executed and output has been verified successfully.

**Ex.no: 7**  
**Date: 12/4/2024**

## **Time Series**

**220801135**

### **Aim:**

To write a python program to analyze time series data with the help of pandas and matplotlib.

### **Algorithm:**

- Import the pandas library for data manipulation
- Import the matplotlib.pyplot library for plotting.
- Create a dictionary data containing the date strings and corresponding values.
- Create a DataFrame df from the dictionary.
- Plot the Time Series:
- Add Labels and Title:
- Use plt.show() to display the plot.
- Calculate Daily Change (Optional):

### **Program:**

```
import pandas as pd
import matplotlib.pyplot as plt

data = {
    "Date": pd.to_datetime(["2023-01-01", "2023-02-01", "2023-03-01", "2023-04-01",
"2023-05-01"]),
    "Value": [100, 120, 135, 110, 145]
}

# Create DataFrame with Date as index
df = pd.DataFrame(data).set_index("Date")

# Plot the time series
plt.figure(figsize=(10, 6)) # Adjust figure size for better viewing
plt.plot(df["Value"], marker='o', linestyle='-')
plt.xlabel("Date")
plt.ylabel("Value")
plt.title("Time Series Data")
plt.grid(True)
plt.show()

# Calculate daily change (optional)
df["Daily Change"] = df["Value"].diff() # Calculate difference between consecutive
values

print(df["Daily Change"].describe())
```

### **Output:**

```

import pandas as pd
import matplotlib.pyplot as plt

# Sample time series data (replace with your actual data)
data = {
    "Date": pd.to_datetime(["2023-01-01", "2023-02-01", "2023-03-01", "2023-04-01", "2023-05-01"]),
    "Value": [100, 120, 135, 110, 145]
}

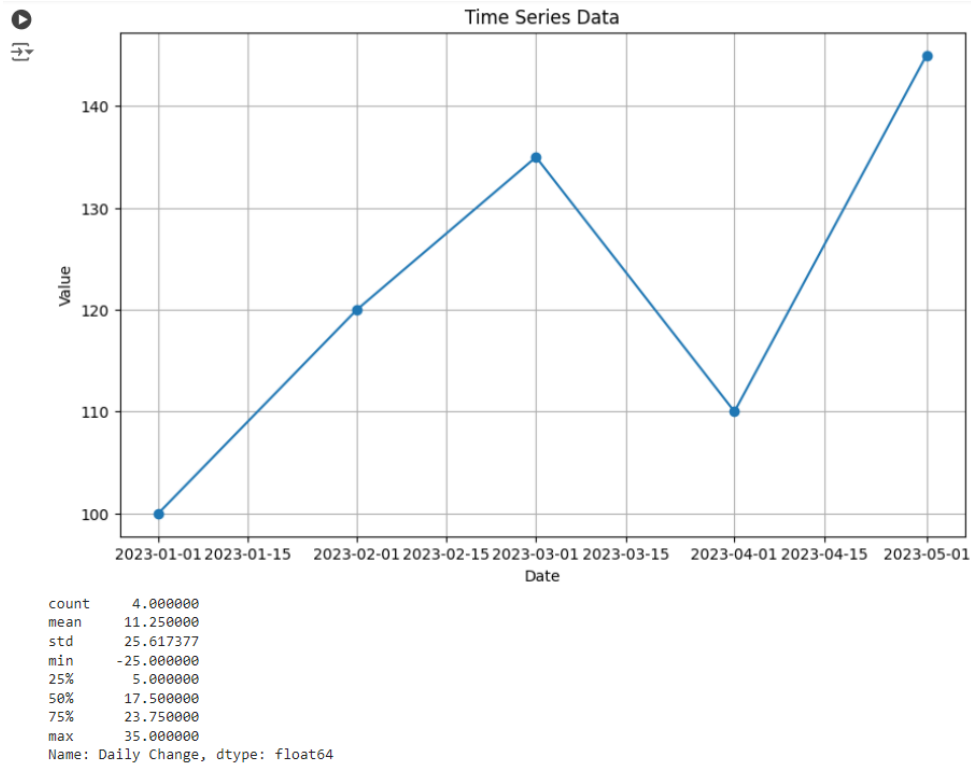
# Create DataFrame with Date as index
df = pd.DataFrame(data).set_index("Date")

# Plot the time series
plt.figure(figsize=(10, 6)) # Adjust figure size for better viewing
plt.plot(df["Value"], marker='o', linestyle='-')
plt.xlabel("Date")
plt.ylabel("Value")
plt.title("Time Series Data")
plt.grid(True)
plt.show()

# Calculate daily change (optional)
df["Daily Change"] = df["Value"].diff() # Calculate difference between consecutive values

# Print descriptive statistics of daily change (optional)
print(df["Daily Change"].describe())

```



## Result:

Thus the program has been done and executed and output has been verified successfully.

**Ex.no: 8**

**Linear regression model to predict the signal strength**

**220801135**

**Date: 26/4/2024**

**Aim:**

To develop a linear regression model to predict the signal strength based on the distance.

**Problem Statement:**

We have a dataset that records the signal strength (in dBm) at various distances (in meters) from a transmitter. The goal is to develop a linear regression model to predict the signal strength based on the distance.

**Algorithm:**

- Import the necessary libraries
- Prepare the dataset
- Split the dataset into training and testing sets a. Use train\_test\_split from sklearn.model\_selection to split X and y into training and testing sets
- Initialize the linear regression model
- Train the model on the training data.
- Make predictions on the testing data
- Evaluate the model
- Plot the results

**Program:**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Example dataset: Distance (meters) vs. Signal Strength (dBm)
data = {
    'Distance': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Signal_Strength': [-30, -35, -40, -45, -50, -55, -60, -65, -70, -75]
}

# Convert the data into a DataFrame
df = pd.DataFrame(data)

# Separate features and target variable
```

```
X = df[['Distance']].values # Feature: Distance
y = df['Signal_Strength'].values # Target: Signal Strength

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

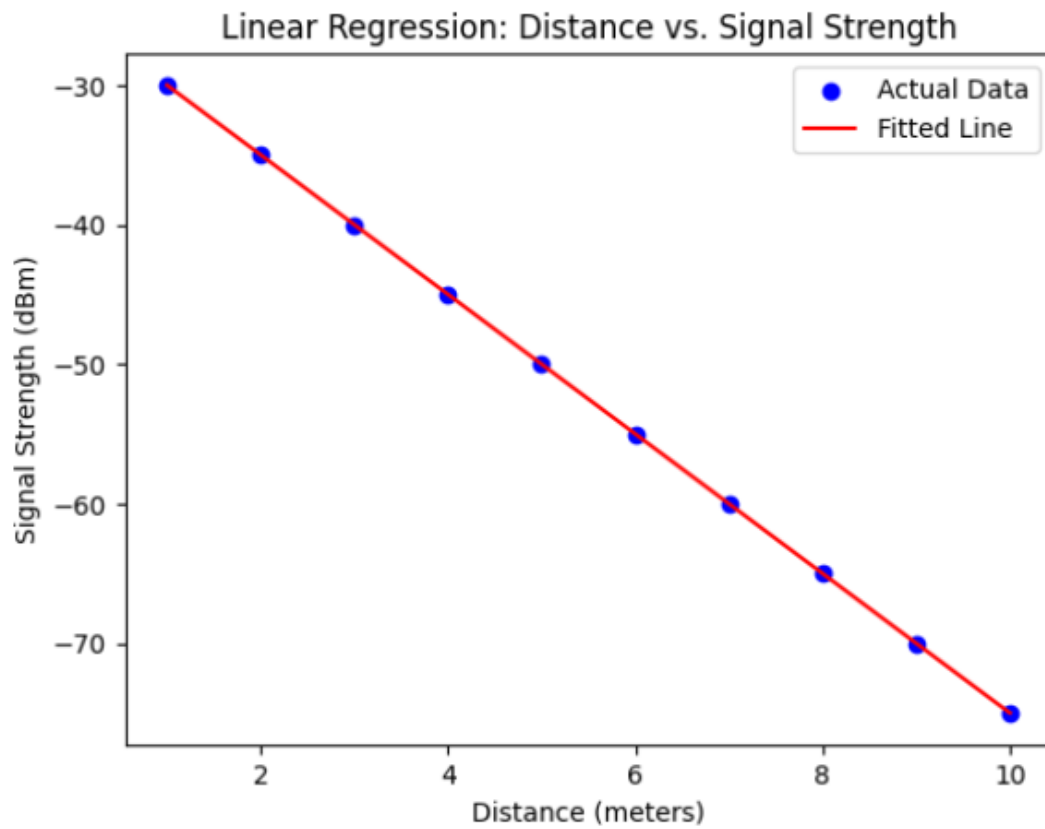
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse:.2f}')
print(f'R^2 Score: {r2:.2f}')

# Visualize the results
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, model.predict(X), color='red', label='Fitted Line')
plt.xlabel('Distance (meters)')
plt.ylabel('Signal Strength (dBm)')
plt.title('Linear Regression: Distance vs. Signal Strength')
plt.legend()
plt.show()
```

**Output:**

Mean Squared Error: 0.00  
R<sup>2</sup> Score: 1.00

**Result:**

Thus the program has been done and executed and output has been verified successfully.



**Ex.no: 9      Decision tree classifier to predict signal quality based on transmitter,      220801135**

**Date: 3/5/2024      signal strength, and frequency**

**Aim:**

Create a simple dataset to classify signal quality based on various parameters such as distance from the transmitter, signal strength, and frequency.

**Problem Statement:**

Dataset that records various parameters affecting the signal quality (Good or Bad). The goal is to develop a decision tree classifier to predict signal quality based on these parameters.

**Algorithm:**

1. Dataset:
  - We create a simple dataset with distance from the transmitter, signal strength, frequency, and corresponding signal quality (Good or Bad). The dataset is stored in a dictionary and then converted into a pandas DataFrame.
2. Data Preparation:
  - Separate the dataset into features (X) and the target variable (y).
  - Encode the target variable Signal\_Quality from categorical values ('Good', 'Bad') to numerical values using LabelEncoder.
3. Model Training:
  - Split the data into training and testing sets using train\_test\_split.
  - Create an instance of DecisionTreeClassifier and train the model on the training data using the fit method.
4. Prediction and Evaluation:
  - Use the trained model to make predictions on the test data.
  - Calculate the accuracy score and generate a classification report to evaluate the model's performance.
5. Visualization:
  - Visualize the decision tree using plot\_tree to understand how the model makes decisions based on the input features.

**Program:**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report
```

```

# Example dataset: Distance (meters), Signal Strength (dBm), Frequency (MHz) vs.
Signal Quality
data = {
    'Distance': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 3, 4, 5, 6],
    'Signal_Strength': [-30, -35, -40, -45, -50, -55, -60, -65, -70, -75, -33, -38, -
43, -48, -53],
    'Frequency': [850, 850, 850, 850, 850, 1900, 1900, 1900, 1900, 1900, 850, 850,
1900, 1900, 1900],
    'Signal_Quality': ['Good', 'Good', 'Good', 'Good', 'Bad', 'Bad', 'Bad', 'Bad',
'Bad', 'Bad', 'Good', 'Good', 'Bad', 'Bad', 'Bad']
}

# Convert the data into a DataFrame
df = pd.DataFrame(data)

# Separate features and target variable
X = df[['Distance', 'Signal_Strength', 'Frequency']].values # Features
y = df['Signal_Quality'].values # Target

# Encode the target variable
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y) # 'Good' -> 1, 'Bad' -> 0

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and train the decision tree classifier
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=['Bad', 'Good'])

print(f'Accuracy: {accuracy:.2f}')
print('Classification Report:')
print(report)

# Visualize the decision tree
plt.figure(figsize=(20,10))
plot_tree(model, feature_names=['Distance', 'Signal_Strength', 'Frequency'],
class_names=['Bad', 'Good'], filled=True)
plt.show()

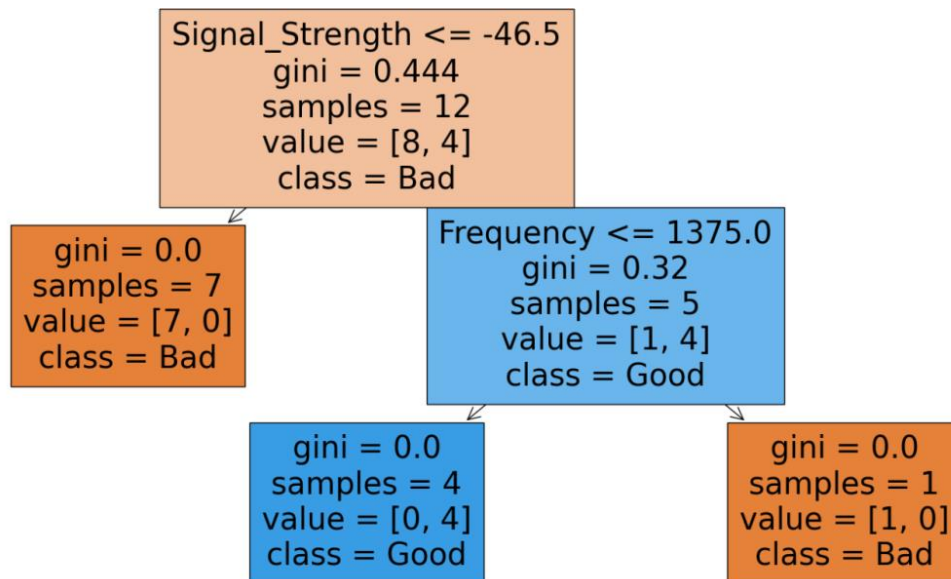
```

## Output:

Accuracy: 1.00

Classification Report:

	precision	recall	f1-score	support
Bad	1.00	1.00	1.00	1
Good	1.00	1.00	1.00	2
accuracy			1.00	3
macro avg	1.00	1.00	1.00	3
weighted avg	1.00	1.00	1.00	3



**Result:**

Thus the program has been done and executed and output has been verified successfully.

**Ex.no: 10** k-NN classifier to predict signal quality based on distance from the transmitter, signal strength, and frequency

**Date: 11/5/2024**

**220801135**

### **Aim:**

To classify signal quality based on various parameters such as distance from the transmitter, signal strength, and frequency.

### **Prerequisite:**

pip install numpy pandas scikit-learn matplotlib

### **Problem Statement**

A dataset that records various parameters affecting the signal quality (Good or Bad). The goal is to develop a k-NN classifier to predict signal quality based on these parameters.

### **Algorithm:**

- Import the necessary libraries
- Prepare the dataset
- Split the dataset into training and testing sets a. Use train\_test\_split from sklearn.model\_selection to split X and y into training and testing sets
- Initialize the KNN model
- Train the model on the training data.
- Make predictions on the testing data
- Evaluate the model
- Plot the results

### **Program:**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns

# Example dataset: Distance (meters), Signal Strength (dBm), Frequency (MHz) vs.
Signal Quality
data = {
    'Distance': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 3, 4, 5, 6],
    'Signal_Strength': [-30, -35, -40, -45, -50, -55, -60, -65, -70, -75, -33, -38, -
43, -48, -53],
```

```

    'Frequency': [850, 850, 850, 850, 850, 1900, 1900, 1900, 1900, 1900, 850, 850,
1900, 1900, 1900],
    'Signal_Quality': ['Good', 'Good', 'Good', 'Good', 'Bad', 'Bad', 'Bad', 'Bad',
'Bad', 'Bad', 'Good', 'Good', 'Bad', 'Bad', 'Bad']
}

# Convert the data into a DataFrame
df = pd.DataFrame(data)

# Separate features and target variable
X = df[['Distance', 'Signal_Strength', 'Frequency']].values # Features
y = df['Signal_Quality'].values # Target

# Encode the target variable
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y) # 'Good' -> 1, 'Bad' -> 0

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the k-NN classifier
k = 3 # Number of neighbors
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=['Bad', 'Good'])

print(f'Accuracy: {accuracy:.2f}')
print('Classification Report:')
print(report)

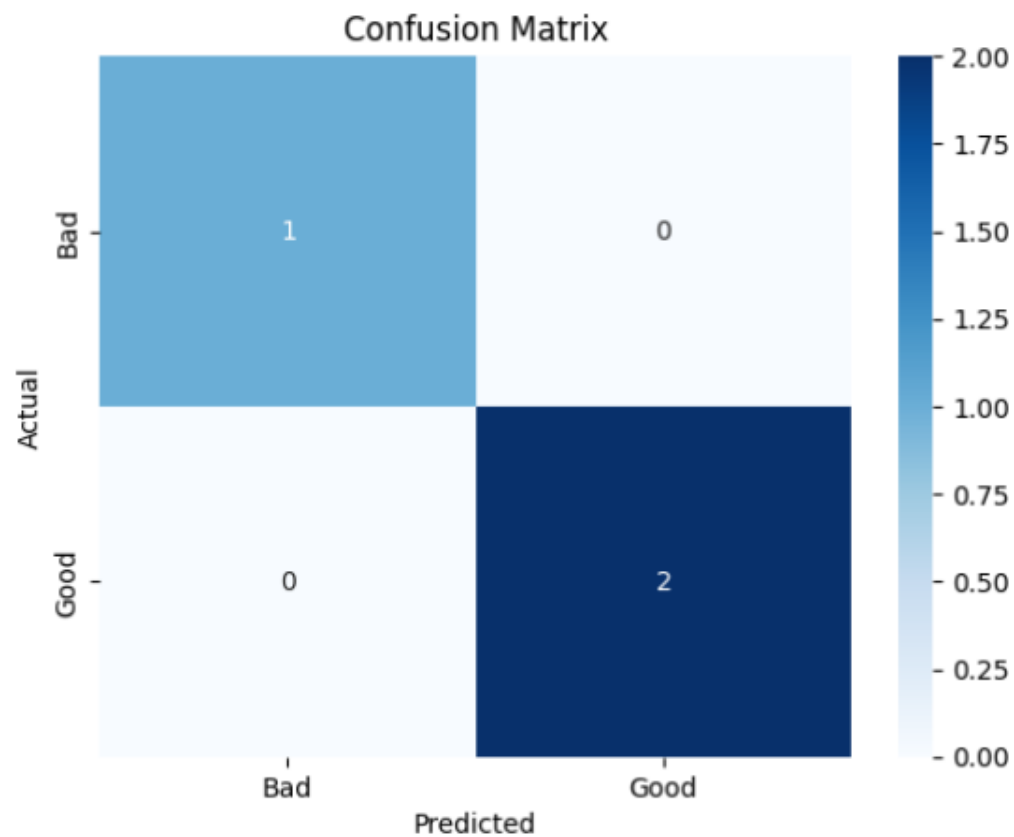
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Bad',
'Good'], yticklabels=['Bad', 'Good'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

```

```
plt.show()
```

### Output:

```
Accuracy: 1.00  
Classification Report:  
              precision    recall  f1-score   support  
  
   Bad           1.00        1.00        1.00         1  
   Good          1.00        1.00        1.00         2  
  
 accuracy                1.00         3  
 macro avg              1.00        1.00        1.00         3  
 weighted avg          1.00        1.00        1.00         3
```



### Result:

Thus the program has been done and executed and output has been verified successfully.

**Ex.no: 11**

**Study of Artificial Neural Network (ANN) and Simple Program in ANN 220801135**

**Date: 17/5/2024**

**Aim:**

To study Artificial Neural Network (ANN) using a simple program in ANN

**Prerequisite:**

pip install numpy scikit-learn

**Algorithm:**

- Import the necessary libraries
- Prepare the dataset
- Split the dataset into training and testing sets a. Use train\_test\_split from sklearn.model\_selection to split X and y into training and testing sets
- Initialize the neural network model
- Train the model on the training data.
- Make predictions on the testing data
- Evaluate the model
- Plot the results

**Program:**

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, accuracy_score

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

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



```
# We'll use a Multi-layer Perceptron classifier
mlp = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=42)
mlp.fit(X_train, y_train)

y_pred = mlp.predict(X_test)

print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

### Output:

Classification Report:

	Precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

Accuracy: 1.0

### Result:

Thus the program has been done and executed and output has been verified successfully.

**Date: 24/5/2024**

**Aim:**

To demonstrate the application of SVM for classification, showcasing its strengths in handling high-dimensional spaces and providing a clear understanding of its working mechanism.

**Prerequisite:**

pip install scikit-learn

**Algorithm:**

- Import the necessary libraries
- Prepare the dataset
- Split the dataset into training and testing sets a. Use train\_test\_split from sklearn.model\_selection to split X and y into training and testing sets
- Initialize SVC model
- Train the model on the training data.
- Make predictions on the testing data
- Evaluate the model
- 

**Program:**

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

clf = SVC(kernel='linear', C=1)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
```

```
print('Accuracy: {accuracy:.2f}')
```

### Output:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

clf = SVC(kernel='linear', C=1)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Accuracy: 1.00

### Result:

Thus the program has been done and executed and output has been verified successfully.