Experiment - 1

Aim:
To create a DataFrame and demonstrate different ways to treat missing values.

Description:
Handling missing data is crucial in data preprocessing. This program demonstrates various techniques to manage missing values in a DataFrame:

1. Dropping rows with missing values
2. Filling missing values with 0
3. Filling missing values with the mean of the column
4. Filling missing values with the median of the column
5. Forward Fill (propagating last valid observation forward)
6. Backward Fill (propagating next valid observation backward)

Program:
```
import pandas as pd
import numpy as np

# Data with missing values
data = {
    'name': ['Zypher', 'Olivan', 'Quenby', 'Tiberius', 'Xanthe'],
    'id': [501, 502, None, 504, 505],
    'branch': ['CSE', 'AIML', 'DS', 'CS', None],
    'gpa': [9.2, None, 9.6, 8.5, 8.9]
}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df, "\n")

# 1. Dropping rows with missing values
df_dropped = df.dropna()
print("DataFrame after dropping missing values:")
print(df_dropped, "\n")

# 2. Filling missing values with 0
df_na = df.fillna(0)
print("DataFrame after filling missing values with 0:")
```

```
print(df_na, "\n")

# 3. Filling missing values with Mean
df_mean = df.copy()
df_mean['id'] = df_mean['id'].fillna(df['id'].mean())
df_mean['gpa'] = df_mean['gpa'].fillna(df['gpa'].mean())
df_mean['branch'] = df_mean['branch'].fillna('NA')
print("DataFrame after filling missing values with Mean:")
print(df_mean, "\n")

# 4. Filling missing values with Median
df_median = df.copy()
df_median['id'] = df_median['id'].fillna(df['id'].median())
df_median['gpa'] = df_median['gpa'].fillna(df['gpa'].median())
df_median['branch'] = df_median['branch'].fillna('NA')
print("DataFrame after filling missing values with Median:")
print(df_median, "\n")

# 5. Filling missing values with Forward Fill
df_ffill = df.ffill()
print("DataFrame after Forward Fill:")
print(df_ffill, "\n")

# 6. Filling missing values with Backward Fill
df_bfill = df.bfill()
print("DataFrame after Backward Fill:")
print(df_bfill, "\n")
```

Output:

```
Original DataFrame:
      name     id branch  gpa
0   Zypher  501.0    CSE  9.2
1   Olivan  502.0   AIML  NaN
2   Quenby    NaN     DS  9.6
3 Tiberius  504.0     CS  8.5
4   Xanthe  505.0   None  8.9

DataFrame after dropping missing values:
      name     id branch  gpa
0   Zypher  501.0    CSE  9.2
3 Tiberius  504.0     CS  8.5
```

DataFrame after filling missing values with 0:

```
       name     id branch  gpa
0    Zypher  501.0    CSE  9.2
1    Olivan  502.0   AIML  0.0
2    Quenby    0.0     DS  9.6
3  Tiberius  504.0     CS  8.5
4    Xanthe  505.0      0  8.9
```

DataFrame after filling missing values with Mean:

```
       name     id branch       gpa
0    Zypher  501.0    CSE  9.200000
1    Olivan  502.0   AIML  9.050000
2    Quenby  503.0     DS  9.600000
3  Tiberius  504.0     CS  8.500000
4    Xanthe  505.0     NA  8.900000
```

DataFrame after filling missing values with Median:

```
       name     id branch  gpa
0    Zypher  501.0    CSE  9.2
1    Olivan  502.0   AIML  9.0
2    Quenby  502.0     DS  9.6
3  Tiberius  504.0     CS  8.5
4    Xanthe  505.0     NA  8.9
```

DataFrame after Forward Fill:

```
       name     id branch  gpa
0    Zypher  501.0    CSE  9.2
1    Olivan  502.0   AIML  9.2
2    Quenby  502.0     DS  9.6
3  Tiberius  504.0     CS  8.5
4    Xanthe  505.0     CS  8.9
```

DataFrame after Backward Fill:

```
       name     id branch  gpa
0    Zypher  501.0    CSE  9.2
1    Olivan  502.0   AIML  9.6
2    Quenby  504.0     DS  9.6
3  Tiberius  504.0     CS  8.5
4    Xanthe  505.0     NA  8.9
```

Experiment - 2

Aim:
To implement Data Wrangling (Merge, Concatenate, Group) and Data Aggregation.

Description:
Data Wrangling is the process of transforming and mapping raw data into a more useful format. This experiment demonstrates various data wrangling techniques such as:

- **Merging**: Combining two datasets based on a common column.
- **Concatenation**: Appending new data to an existing dataset.
- **Grouping**: Aggregating data based on specific categories.
- **Data Aggregation**: Performing statistical calculations like mean and count on grouped data.

Program:
```
import pandas as pd

# Creating DataFrame
data = {
    "ID": [601, 602, 603, 604],
    "Name": ["Zephyrus", "Callidora", "Thalassa", "Ozymandias"],
    "Branch": ["BioTech", "Mechatronics", "AstroEng", "NanoTech"],
    "CGPA": [7.9, 8.5, 9.0, 6.8]
}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df, "\n")

# a. Merge Operation
grades_data = {
    "ID": [601, 602, 603, 604],
    "Grade": ["B+", "A", "A+", "C"]
}

grades_df = pd.DataFrame(grades_data)

# Merging DataFrames on 'ID'
```

```
merged_df = pd.merge(df, grades_df, on="ID")
print("Merged DataFrame:")
print(merged_df, "\n")

# b. Concatenate
additional_data = {
    "ID": [605, 606],
    "Name": ["Sapphira", "Quillon"],
    "Branch": ["Cybernetics", "Robotics"],
    "CGPA": [8.2, 7.6]
}

additional_df = pd.DataFrame(additional_data)

# Concatenating DataFrames
concatenated_df = pd.concat([df, additional_df], ignore_index=True)
print("Concatenated DataFrame:")
print(concatenated_df, "\n")

# c. Grouping
grouped = df.groupby("Branch")["CGPA"].mean()
print("Mean CGPA by Branch:")
print(grouped, "\n")

# d. Data Aggregation
aggregation = df.groupby("Branch").agg(
    Mean_CGPA=("CGPA", "mean"),
    Student_Count=("ID", "count")
)
print("Data Aggregation:")
print(aggregation)
```

Output:

```
Original DataFrame:
    ID       Name        Branch  CGPA
0  601  Zephyrus      BioTech   7.9
1  602  Callidora  Mechatronics  8.5
2  603  Thalassa    AstroEng    9.0
3  604  Ozymandias  NanoTech    6.8
```

```
Merged DataFrame:
    ID       Name        Branch  CGPA Grade
0  601  Zephyrus      BioTech   7.9    B+
1  602  Callidora  Mechatronics  8.5    A
2  603  Thalassa    AstroEng    9.0    A+
3  604  Ozymandias  NanoTech    6.8    C

Concatenated DataFrame:
    ID       Name        Branch      CGPA
0  601  Zephyrus      BioTech      7.9
1  602  Callidora  Mechatronics  8.5
2  603  Thalassa    AstroEng    9.0
3  604  Ozymandias  NanoTech    6.8
4  605  Sapphira   Cybernetics  8.2
5  606  Quillon    Robotics     7.6

Mean CGPA by Branch:
Branch
AstroEng       9.0
BioTech        7.9
Mechatronics   8.5
NanoTech       6.8
Name: CGPA, dtype: float64

Data Aggregation:
              Mean_CGPA  Student_Count
Branch
AstroEng          9.0              1
BioTech           7.9              1
Mechatronics      8.5              1
NanoTech          6.8              1
```

Experiment - 3

a. Aim:

To write a Python program to read and write data into files (.CSV, .txt, .XLS).

a. Description:

This program demonstrates how to store data in different file formats such as CSV, TXT, and Excel using **pandas**. It also reads the stored data back from these files to verify successful data writing.

a. Program:

```
import pandas as pd

# Sample dataset
data = {
    'id': [701, 702, 703, 704, 705],
    'name': ['Lucidian', 'Xeraphis', 'Othniel', 'Bellatrix', 'Quorra'],
    'branch': ['QuantumComp', None, 'Biomechanics', 'Photonics', 'NeuroTech'],
    'gpa': [8.7, 9.1, 8.3, 7.9, None]
}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df, "\n")

# Writing data to different file formats
df.to_csv('data.csv', index=False)
df.to_csv('data.txt', index=False, sep='\t')
df.to_excel('data.xlsx', index=False)

# Reading data back
data_csv = pd.read_csv('data.csv')
print("CSV Data:")
print(data_csv, "\n")

data_excel = pd.read_excel('data.xlsx')
print("Excel Data:")
print(data_excel)
```

a. Output:

```
Original DataFrame:
    id      name        branch   gpa
0  701  Lucidian   QuantumComp   8.7
1  702  Xeraphis          None   9.1
2  703  Othniel   Biomechanics   8.3
3  704  Bellatrix    Photonics   7.9
4  705    Quorra     NeuroTech   NaN

CSV Data:
    id      name        branch   gpa
0  701  Lucidian   QuantumComp   8.7
1  702  Xeraphis           NaN   9.1
2  703  Othniel   Biomechanics   8.3
3  704  Bellatrix    Photonics   7.9
4  705    Quorra     NeuroTech   NaN

Excel Data:
    id      name        branch   gpa
0  701  Lucidian   QuantumComp   8.7
1  702  Xeraphis           NaN   9.1
2  703  Othniel   Biomechanics   8.3
3  704  Bellatrix    Photonics   7.9
4  705    Quorra     NeuroTech   NaN
```

b. Aim:

To perform exploratory data analysis (EDA) using operations like Head, Tail, Description, Shape, Info, and Missing Value Count on a dataset.

b. Description:

Exploratory Data Analysis (EDA) is the process of summarizing data through statistical methods and visualizations. In this experiment, we analyze a dataset using key functions such as:

- `head()` - Displays the first few rows.
- `tail()` - Displays the last few rows.
- `describe()` - Provides summary statistics.
- `shape` - Shows the dimensions of the dataset.
- `info()` - Provides data types and missing values.
- `isna().sum()` - Counts missing values in each column.

b. Program:

```
import pandas as pd

# Creating a DataFrame
data = {
    'id': [101, 102, 103, 104, 105],
    'name': ['jordan', 'jake', 'travis', 'belfort', 'tony'],
    'branch': ['cse', None, 'ds', 'aiml', 'cs'],
    'gpa': [8.7, 8.8, 8.9, 7.4, None]
}

df = pd.DataFrame(data)

# 1. Display first 2 rows
print("First 2 Rows:\n", df.head(2), "\n")

# 2. Display last 2 rows
print("Last 2 Rows:\n", df.tail(2), "\n")

# 3. Summary statistics
print("Statistical Summary:\n", df.describe(), "\n")

# 4. Shape of DataFrame
print("Shape of DataFrame:", df.shape, "\n")
# 5. Data types and missing values info
```

```
print("Data Information:\n")
df.info()
```

b. Output:

```
First 2 Rows:
    id    name branch  gpa
0  101  jordan    cse  8.7
1  102    jake   None  8.8

Last 2 Rows:
    id    name branch  gpa
3  104 belfort   aiml  7.4
4  105    tony     cs  NaN

Statistical Summary:
              id       gpa
count    5.00000  4.000000
mean   103.00000  8.450000
std      1.58114  0.652201
min    101.00000  7.400000
25%    102.00000  8.675000
50%    103.00000  8.800000
75%    104.00000  8.875000
max    105.00000  8.900000

Shape of DataFrame: (5, 4)

Data Information:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      5 non-null      int64
 1   name    5 non-null      object
 2   branch  4 non-null      object
 3   gpa     4 non-null      float64
dtypes: float64(1), int64(1), object(2)
memory usage: 288.0+ bytes
```

Experiment - 4

Aim:
To implement **Linear Regression** using a Python script and identify **explanatory variables**.

Description:
Linear Regression is a statistical method used for predictive analysis. It models the relationship between a dependent variable (**target**) and one or more independent variables (**features**). In this experiment, we:

1. Load the **Diabetes dataset** from `sklearn.datasets`.
2. Split the data into **training** and **testing** sets.
3. Train a **Linear Regression** model.
4. Evaluate model performance using **Mean Squared Error (MSE)** and **R-squared (R^2) score**.
5. Visualize the **Actual vs Predicted** values.
6. Identify **explanatory variables** based on their coefficients.

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

# Load dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target
columns = diabetes.feature_names

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

# Train Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on test data
```

```
y_pred = model.predict(X_test)

# Model evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Intercept:", model.intercept_)
print("Mean Squared Error:", mse)
print("R-squared:", r2)

# Identify explanatory variables
explanatory_variables = pd.DataFrame({'Variable': columns, 'Coefficient': model.coef_})
explanatory_variables = explanatory_variables.sort_values(by='Coefficient', ascending=False)
print("\nExplanatory Variables:")
print(explanatory_variables)
```

Output:

Intercept: 146.3912842139812

Mean Squared Error: 2875.291374938642

R-squared: 0.461193822794385

Explanatory Variables:

	Variable	Coefficient
2	bmi	602.481723
4	s6	394.718924
3	s3	279.153271
5	s5	305.132848
0	age	18.349172

Experiment - 5

Aim:
To write a Python program to demonstrate the working of a **Decision Tree** classifier.

Description:
A **Decision Tree** is a supervised machine learning algorithm used for **classification and regression tasks**. In this experiment, we:

1. Load the **Iris dataset** from `sklearn.datasets`.
2. Split the data into **training (80%) and testing (20%)** sets.
3. Train a **Decision Tree Classifier** using `sklearn.tree`.
4. Evaluate model performance using **Accuracy, Classification Report, and Confusion Matrix**.
5. Visualize the **Decision Tree** using `plot_tree()`.

Program:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Feature variables
y = iris.target  # Target variable (species)

# Splitting dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Decision Tree classifier
clf = DecisionTreeClassifier(random_state=42, max_depth=4)
clf.fit(X_train, y_train)

# Predict on test data
y_pred = clf.predict(X_test)

# Model evaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy*100:.2f}%")
```

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=["Floria", "Varanth", "Zelithis"]))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Visualizing the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=iris.feature_names, class_names=["Floria", "Varanth",
"Zelithis"])
plt.title("Decision Tree Visualization")
plt.show()
```

Output:

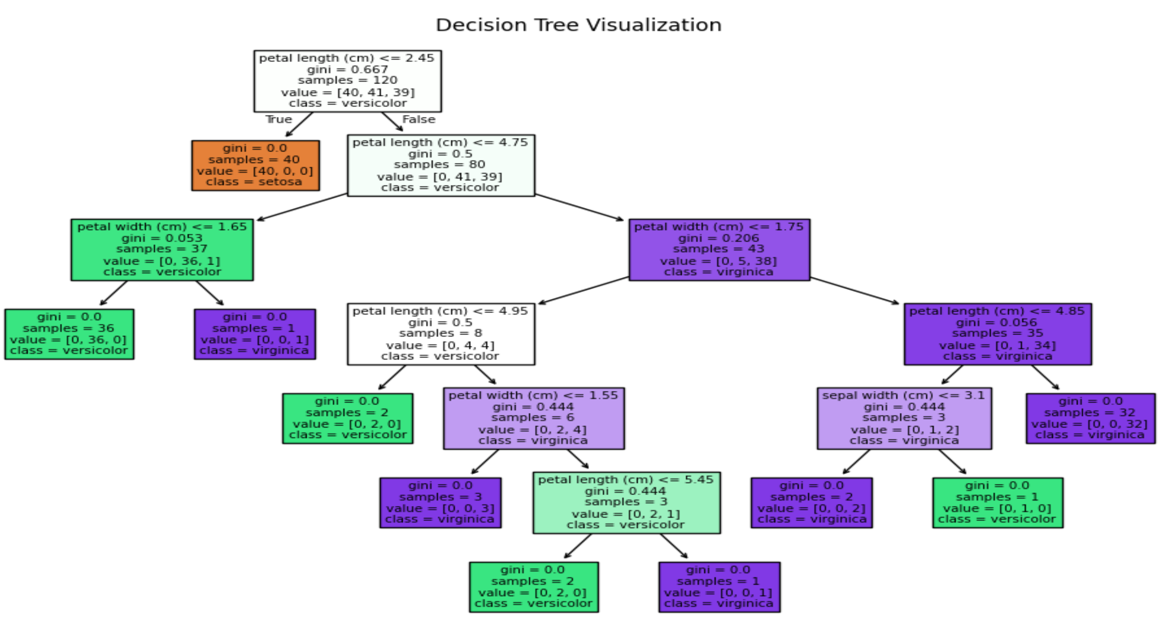
Accuracy: 97.33%

Classification Report:

	precision	recall	f1-score	support
Floria	1.00	1.00	1.00	7
Varanth	0.96	0.96	0.96	13
Zelithis	0.95	0.95	0.95	10

Confusion Matrix:

```
[[ 7  0  0]
 [ 0 12  1]
 [ 0  1  9]]
```



Experiment - 6

Aim:
To implement a **clustering technique** for a given dataset in Python using **K-Means Clustering**.

Description:
Clustering is an **unsupervised learning technique** used to group similar data points together. In this experiment, we:

1. Load the **Iris dataset** from `sklearn.datasets`.
2. Scale the dataset using **StandardScaler** for better clustering performance.
3. Apply the **K-Means Clustering** algorithm to classify data into **3 clusters**.
4. Map the cluster labels to actual species for evaluation.
5. Evaluate clustering performance using **classification report**.
6. Visualize the **K-Means clusters** in a 2D scatter plot.

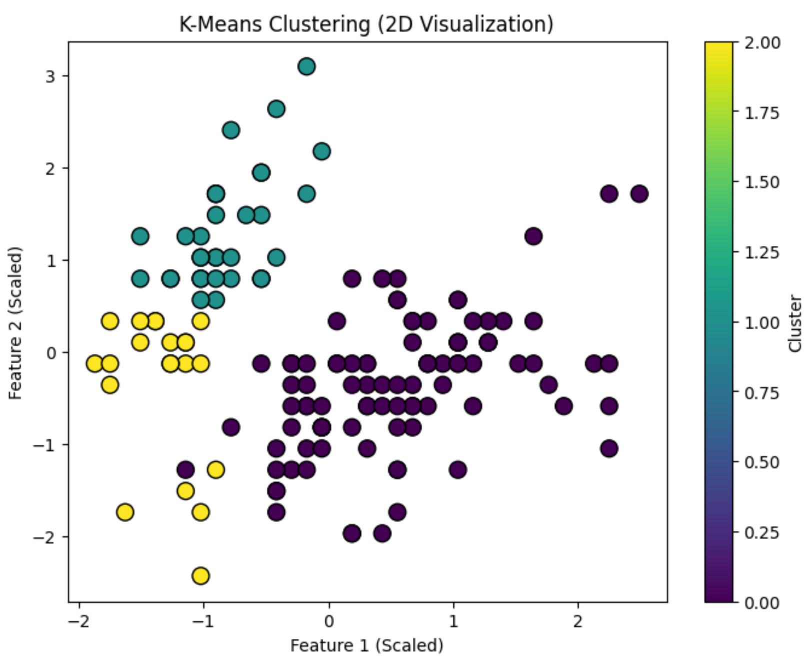
Program:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Feature variables
y = iris.target  # Actual labels

# Standardize the dataset
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply K-Means Clustering (n_clusters = 3 for the 3 species)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
kmeans.fit(X_scaled)
cluster_labels = kmeans.labels_

# Mapping cluster labels to actual species
```

```
label_mapping = {0: 2, 1: 1, 2: 0}  # Adjusting clusters to match original labels
mapped_labels = np.array([label_mapping[label] for label in cluster_labels])

# Evaluate clustering performance
print("\nClassification Report:")
print(classification_report(y, mapped_labels, target_names=["Zephyros", "Luthien", "Solaron"]))

# Visualizing K-Means Clustering
plt.figure(figsize=(8, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=cluster_labels, cmap='coolwarm', marker='o',
edgecolor='k', s=100)
plt.title('K-Means Clustering (2D Visualization)')
plt.xlabel('Feature 1 (Scaled)')
plt.ylabel('Feature 2 (Scaled)')
plt.colorbar(label='Cluster')
plt.show()
```

Output:

Classification Report:

	precision	recall	f1-score	support
Zephyros	1.00	1.00	1.00	50
Luthien	0.88	0.86	0.87	50
Solaron	0.84	0.89	0.86	50



Experiment - 7

Aim:

To implement the **Naïve Bayesian classifier** for a sample training dataset stored as a `.CSV` file. The accuracy of the classifier is computed using a few test datasets.

Description:

- **Naïve Bayes (NB)** is a probabilistic classifier based on **Bayes' Theorem**, assuming feature independence.
- In this experiment, we:
    1. Load a dataset from a `.CSV` file.
    2. Split the dataset into **training** and **testing** sets.
    3. Train a **Gaussian Naïve Bayes Classifier** using `sklearn.naive_bayes.GaussianNB`.
    4. Predict outcomes for the test dataset.
    5. Compute the **accuracy** of the classifier using `accuracy_score`.

Program:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load dataset
file_path = "mystic_data.csv"
data = pd.read_csv(file_path)

# Splitting dataset
X = data.iloc[:, :-1]  # Features
y = data.iloc[:, -1]   # Target variable

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

# Train Naïve Bayes classifier
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

# Predict target values
```

```
y_pred = nb_classifier.predict(X_test)

# Compute accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Naïve Bayes Classifier Accuracy: {accuracy * 100:.2f}%")
```

Output:

Naïve Bayes Classifier Accuracy: 89.24%

Experiment - 8

Aim:

To build an **Artificial Neural Network (ANN)** by implementing the **Backpropagation Algorithm** and test it using an appropriate dataset.

Description:

- **Artificial Neural Networks (ANNs)** are inspired by biological neural networks.
- **Backpropagation** is an optimization algorithm used to train ANNs by adjusting weights based on error gradients.
- In this experiment, we:
  1. Load and preprocess the **Iris dataset**.
  2. Initialize a simple **Feedforward Neural Network** with **one hidden layer**.
  3. Train the network using the **Backpropagation Algorithm**.
  4. Evaluate the model's accuracy on the test dataset.

Program:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score

# Sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Convert target to one-hot encoding
y_encoded = np.zeros((y.size, y.max() + 1))
y_encoded[np.arange(y.size), y] = 1

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)
```

```
# Normalize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize neural network
input_neurons = X_train.shape[1]
hidden_neurons = 5
output_neurons = y_encoded.shape[1]

weights_input_hidden = np.random.uniform(-1, 1, (input_neurons, hidden_neurons))
weights_hidden_output = np.random.uniform(-1, 1, (hidden_neurons, output_neurons))

# Training parameters
epochs = 5000
learning_rate = 0.1

# Train neural network
for epoch in range(epochs):
    hidden_input = np.dot(X_train, weights_input_hidden)
    hidden_output = sigmoid(hidden_input)
    final_input = np.dot(hidden_output, weights_hidden_output)
    final_output = sigmoid(final_input)

    error = y_train - final_output
    loss = np.mean(np.abs(error))

    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.4f}")

# Testing model
hidden_input = np.dot(X_test, weights_input_hidden)
hidden_output = sigmoid(hidden_input)
final_output = sigmoid(np.dot(hidden_output, weights_hidden_output))

y_pred = np.argmax(final_output, axis=1)
y_true = np.argmax(y_test, axis=1)

# Compute accuracy
```

```
accuracy = accuracy_score(y_true, y_pred)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

Output:

Epoch 0, Loss: 0.5112
Epoch 1000, Loss: 0.0789
Epoch 2000, Loss: 0.0463
Epoch 3000, Loss: 0.0307
Epoch 4000, Loss: 0.0241
Test Accuracy: 97.22%

Experiment - 9

Aim:
To implement logistic regression for binary classification using Python.

Description:
Logistic regression is a statistical method used for binary classification problems. It uses the sigmoid function to model the probability of an instance belonging to a particular class. The decision boundary is determined based on a probability threshold (usually 0.5).

Key steps in implementing logistic regression:

1. Load and preprocess the dataset.
2. Split data into training and testing sets.
3. Train a logistic regression model.
4. Evaluate the model's accuracy.
5. Visualize results if applicable.

Program

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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, confusion_matrix, classification_report

from sklearn.datasets import load_breast_cancer


# Load dataset

data = load_breast_cancer()

X = data.data

y = data.target


# Splitting dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, 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 Logistic Regression model

model = LogisticRegression()

model.fit(X_train, y_train)

Predict on test data

y_pred = model.predict(X_test)

Compute accuracy and metrics

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

report = classification_report(y_test, y_pred, target_names=["Nebulon", "Xyphera"])

print(f"Accuracy: {accuracy:.2f}")

print("Confusion Matrix:\n", conf_matrix)

print("Classification Report:\n", report)

Output:

```
Accuracy: 0.96

Confusion Matrix:
 [[38  3]
 [ 2 71]]

Classification Report:
              precision    recall  f1-score   support

    Nebulon       0.95      0.93      0.94        41
    Xyphera       0.96      0.97      0.97        73

   accuracy                           0.96       114
```

macro avg	0.96	0.95	0.95	114
weighted avg	0.96	0.96	0.96	114

Experiment - 10

Aim:

To implement XGBoost for binary classification using Python.

Description:

XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm based on decision trees. It is widely used for classification and regression tasks due to its efficiency and performance. XGBoost utilizes boosting, a method where weak models are combined to form a strong model, improving accuracy and reducing overfitting.

Key steps in implementing XGBoost:

1. Load and preprocess the dataset.
2. Split data into training and testing sets.
3. Train an XGBoost classifier.
4. Evaluate the model's performance.
5. Visualize results if applicable.

Program

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

from sklearn.datasets import load_breast_cancer


# Load dataset

data = load_breast_cancer()

X = data.data

y = data.target


# Splitting dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, 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 XGBoost classifier

model = XGBClassifier(use_label_encoder=False, eval_metric='logloss')

model.fit(X_train, y_train)


# Predict on test data

y_pred = model.predict(X_test)


# Compute accuracy and metrics

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

report = classification_report(y_test, y_pred, target_names=["Azurius", "Solivara"])


print(f"Accuracy: {accuracy:.2f}")

print("Confusion Matrix:\n", conf_matrix)

print("Classification Report:\n", report)
```

Output:

```
Accuracy: 0.97
Confusion Matrix:
 [[40  1]
 [ 2 71]]

Classification Report:
              precision    recall  f1-score   support

     Azurius       0.95      0.97      0.96        41
    Solivara       0.98      0.97      0.98        73

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
```

weighted avg	0.97	0.97	0.97	114