TABLE OF CONTENTS

Practical - 1
Implement any arithmetic, logical operation using Numpy library

```
# Numpy arithmetic and logical operations
import numpy as np

# Create two numpy arrays
array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([5, 4, 3, 2, 1])

# Perform arithmetic operations
sum_array = np.add(array1, array2)
diff_array = np.subtract(array1, array2)
prod_array = np.multiply(array1, array2)
div_array = np.divide(array1, array2)

# Perform logical operations
greater_than = np.greater(array1, array2)
equal_to = np.equal(array1, array2)

# Print the results
print("Sum of arrays:", sum_array)
print("Difference of arrays:", diff_array)
print("Product of arrays:", prod_array)
print("Division of arrays:", div_array)
print("Array1 greater than Array2:", greater_than)
print("Array1 equal to Array2:", equal_to)
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                    Python

PS C:\Users\mhdas\Desktop\Minor Project> & C:/Users/mhdas/AppData/Local/Programs/Python/Python313
/python.exe "c:/Users/mhdas/Desktop/AI Practical/practical_1.py"
Sum of arrays: [6 6 6 6 6]
Difference of arrays: [-4 -2  0  2  4]
Product of arrays: [5 8 9 8 5]
Division of arrays: [0.2 0.5 1.  2.  5. ]
Array1 greater than Array2: [False False False  True  True]
Array1 equal to Array2: [False False  True False False]
PS C:\Users\mhdas\Desktop\Minor Project>
```

Practical – 2

Implement any 10 functionalities using Pandas library (including statistical function)

```
import pandas as pd
# Read the iris dataset
df = pd.read_csv(r"C:\Users\mhdas\Desktop\iris.csv")

# 1. Display the first few rows of the DataFrame
print("First few rows of the DataFrame:")
print(df.head())
# 2. Display the last few rows of the DataFrame
print("\nLast few rows of the DataFrame:")
print(df.tail())
# 3. Display DataFrame shape
print("\nShape of the DataFrame:")
print(df.shape)
# 4. Display DataFrame info
print("\nDataFrame info:")
print(df.info())
# 5. Check for null values
print("\nNull values in DataFrame:")
print(df.isnull())
# 6. Display the summary statistics of the DataFrame
print("\nSummary statistics of the DataFrame:")
print(df.describe())
# 7. Calculate the mean of each column
print("\nMean of each column:")
print(df.select_dtypes(include='number').mean())
# 8. Calculate the median of each column
print("\nMedian of each column:")
print(df.select_dtypes(include='number').median())  # Only select numeric columns
# 9. Calculate the standard deviation of each column
print("\nStandard deviation of each column:")
print(df.select_dtypes(include='number').std())
# 10. Display value counts of species
print("\nCount of each species:")
print(df['species'].value_counts())
# 11. Group by species and calculate mean
print("\nMean values for each species:")
print(df.groupby('species').mean())
# 12. Sort values by sepal length
print("\nSorted by sepal length:")
print(df.sort_values('sepal_length', ascending=False).head())
# 13. Calculate percentage of each species
print("\nPercentage of each species:")
print(df['species'].value_counts(normalize=True) * 100)
```

Output:

```
First few rows of the DataFrame:
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa

Last few rows of the DataFrame:
     sepal_length  sepal_width  petal_length  petal_width    species
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

Shape of the DataFrame:
(150, 5)

DataFrame info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None

Null values in DataFrame:
     sepal_length  sepal_width  petal_length  petal_width  species
0           False        False         False        False    False
1           False        False         False        False    False
2           False        False         False        False    False
3           False        False         False        False    False
4           False        False         False        False    False
..            ...          ...           ...          ...      ...
145         False        False         False        False    False
146         False        False         False        False    False
147         False        False         False        False    False
148         False        False         False        False    False
```

```
Summary statistics of the DataFrame:
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000

Mean of each column:
sepal_length    5.843333
sepal_width     3.054000
petal_length    3.758667
petal_width     1.198667
dtype: float64

Median of each column:
sepal_length    5.80
sepal_width     3.00
petal_length    4.35
petal_width     1.30
dtype: float64

Standard deviation of each column:
sepal_length    0.828066
sepal_width     0.433594
petal_length    1.764420
petal_width     0.763161
dtype: float64

Count of each species:
species
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64

Mean values for each species:
            sepal_length  sepal_width  petal_length  petal_width
species
setosa             5.006        3.418         1.464        0.244
versicolor         5.936        2.770         4.260        1.326
virginica          6.588        2.974         5.552        2.026
```

Ctrl+K to generate a command

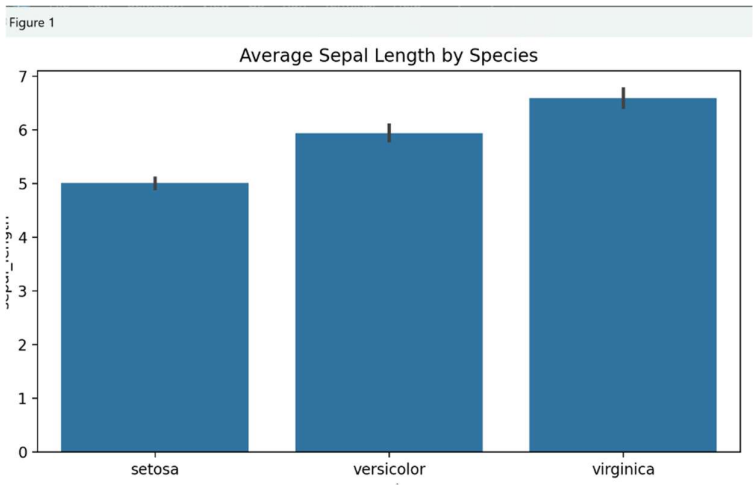
```
Sorted by sepal length:
     sepal_length  sepal_width  petal_length  petal_width    species
131           7.9          3.8           6.4          2.0  virginica
122           7.7          2.8           6.7          2.0  virginica
118           7.7          2.6           6.9          2.3  virginica
117           7.7          3.8           6.7          2.2  virginica
135           7.7          3.0           6.1          2.3  virginica

Percentage of each species:
species
setosa        33.333333
versicolor    33.333333
virginica     33.333333
Name: proportion, dtype: float64
```

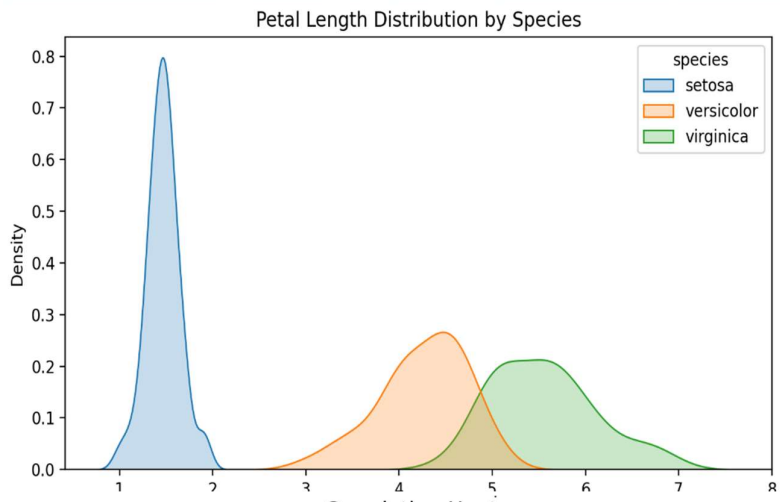
Practical -3

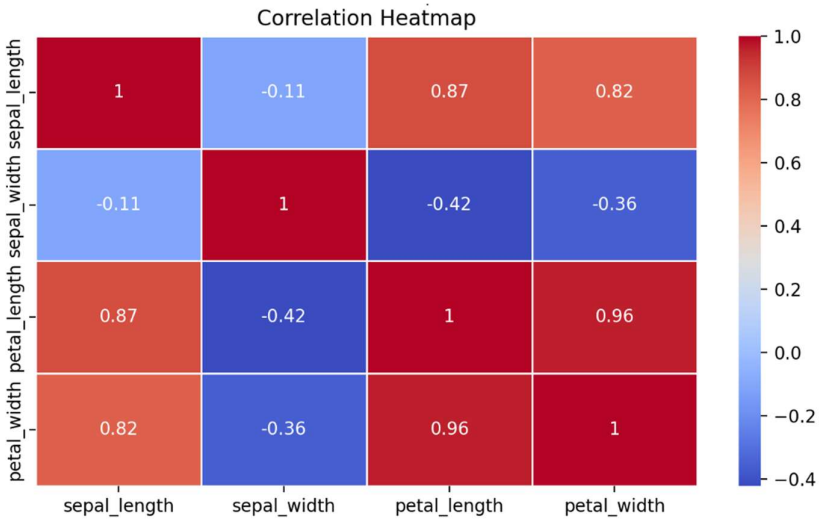
Perform various plot including barplot, density plot, pairwise plot, heat map using matplotlib library

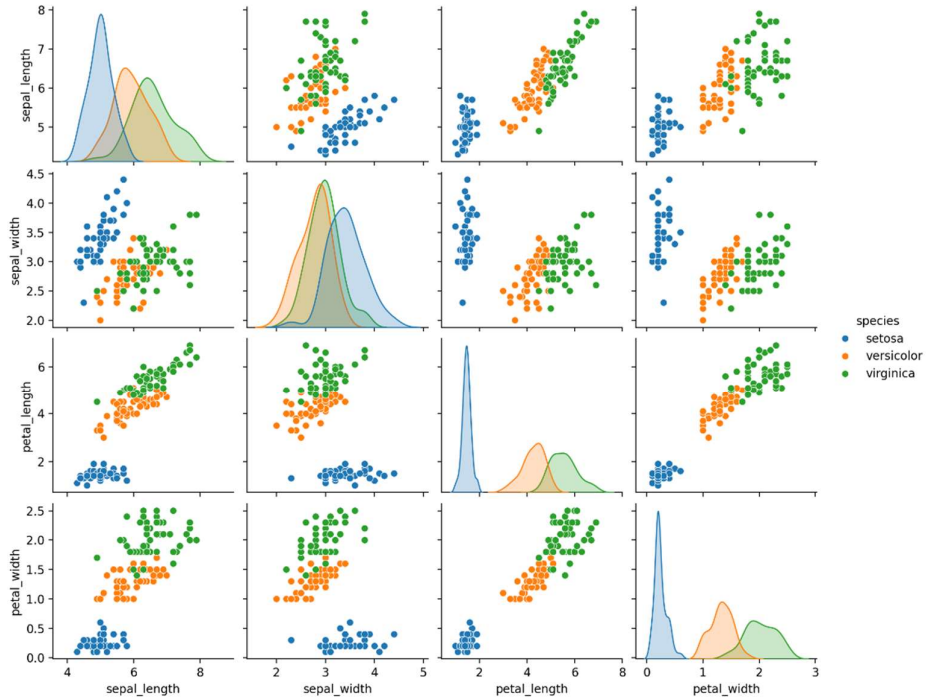
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Read the iris dataset
df = pd.read_csv(r"C:\Users\mhdas\Desktop\iris.csv")

# Create a figure with multiple subplots
plt.figure(figsize=(15, 10))

# 1. Bar plot
plt.subplot(2, 2, 1)
sns.barplot(x='species', y='sepal_length', data=df)
plt.title('Average Sepal Length by Species')

# 2. Density plot
plt.subplot(2, 2, 2)
sns.kdeplot(data=df, x='petal_length', hue='species', shade=True)
plt.title('Petal Length Distribution by Species')

# 3. Heat map
plt.subplot(2, 2, 3)
correlation = df.select_dtypes(include=['float64', 'int64']).corr()
sns.heatmap(correlation, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')

plt.tight_layout()
plt.show()

# Pairplot needs to be separate as it creates its own figure
sns.pairplot(df, hue='species')
plt.show()
```

Output:




Petal Length Distribution by Species
species
setosa
versicolor
virginica
Density


Correlation Heatmap
sepal_length
sepal_width
petal_length
petal_width
1
-0.11
0.87
0.82
-0.11
1
-0.42
-0.36
0.87
-0.42
1
0.96
0.82
-0.36
0.96
1


species
setosa
versicolor
virginica
sepal_length
sepal_width
petal_length
petal_width

Practical -4

Write a python program to fit a linear regression for any real-time dataset.

```
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, mean_absolute_error

# Read the dataset
df = pd.read_csv(r"C:\Users\mhdas\Desktop\iris.csv")

# Select features and target variable
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df['sepal_length']  # Assuming we want to predict sepal_length

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

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

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Mean Absolute Error:", mae)
```

Output :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                    Python
PS C:\Users\mhdas\Desktop\Minor Project> & C:/Users/mhdas/AppData/Local/Programs/Python/Pyt
/python.exe "c:/Users/mhdas/Desktop/AI Practical/practical_4.py"
Mean Squared Error: 5.434375124855859e-31
R-squared: 1.0
Mean Absolute Error: 6.118562446823085e-16
PS C:\Users\mhdas\Desktop\Minor Project>
```

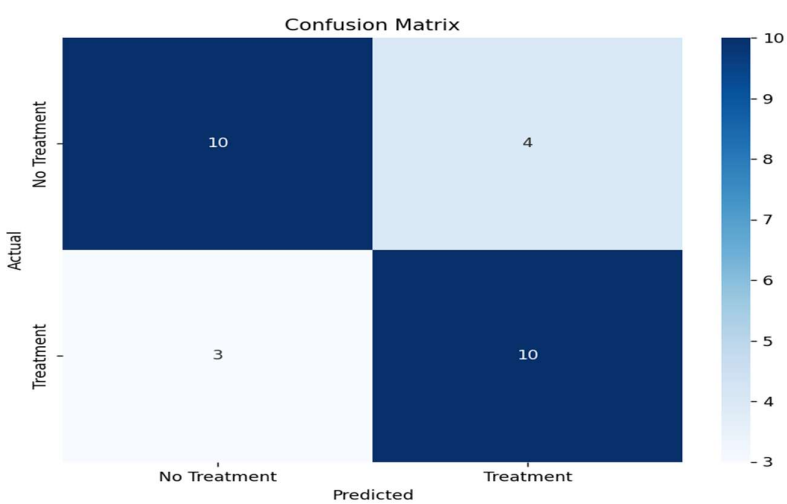
Practical -5
Implement k-nn for binary classification task.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the Cryotherapy dataset
dataset_path = r'C:\Users\VISHNULAL\Downloads\archive (1)\Cryotherapy.xlsx'
data = pd.read_excel(dataset_path)

# Step 2: Display column names and preview data
print("Column names:", data.columns)
print("\nFirst few rows:\n", data.head())

# Step 3: Define features (X) and target (y)
X = data.drop(columns=['Result_of_Treatment'])
y = data['Result_of_Treatment']

# Step 4: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 5: Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 6: Train a k-NN classifier
k = 3  # Set k to 3
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Step 7: Make predictions
y_pred = knn.predict(X_test)

# Step 8: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy)

# Calculate single value metrics
precision = precision_score(y_test, y_pred, average='binary')
recall = recall_score(y_test, y_pred, average='binary')
f1 = f1_score(y_test, y_pred, average='binary')

print("\nPrecision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
# Step 9: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", conf_matrix)

# Step 10: Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No Treatment', 'Treatment'],
yticklabels=['No Treatment', 'Treatment'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```

OUTPUT

```
Column names: Index(['sex', 'age', 'Time', 'Number_of_Warts', 'Type', 'Area',
       'Result_of_Treatment'],
      dtype='object')

First few rows:
    sex  age   Time  Number_of_Warts  Type  Area  Result_of_Treatment
0    1   35  12.00                5     1   100                    0
1    1   29   7.00                5     1    96                    1
2    1   50   8.00                1     3   132                    0
3    1   32  11.75                7     3   750                    0
4    1   67   9.25                1     1    42                    0

Accuracy: 0.7407407407407407

Precision: 0.7142857142857143
Recall: 0.7692307692307693
F1 Score: 0.7407407407407407
```



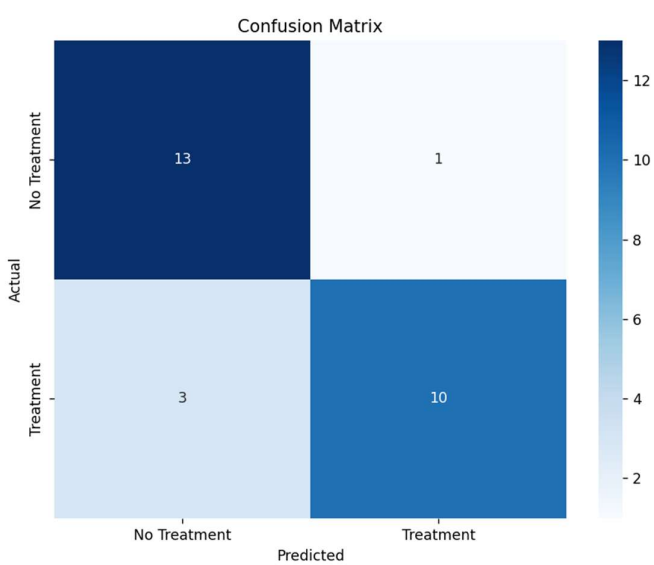
Practical-6
Implement logistic regression for binary classification task.

```
import pandas as pd
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, precision_score, recall_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the Cryotherapy dataset
dataset_path = r'C:\Users\VISHNULAL\Downloads\archive (1)\Cryotherapy.xlsx'
data = pd.read_excel(dataset_path)

# Step 2: Display column names and preview data
print("Column names:", data.columns)
print("\nFirst few rows:\n", data.head())

# Step 3: Define features (X) and target (y)
X = data.drop(columns=['Result_of_Treatment'])
y = data['Result_of_Treatment']

# Step 4: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 5: Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 6: Train a Logistic Regression classifier
log_reg = LogisticRegression()  # Create a logistic regression model
log_reg.fit(X_train, y_train)  # Fit the model to the training data

# Step 7: Make predictions
y_pred = log_reg.predict(X_test)

# Step 8: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy)

# Calculate single value metrics
precision = precision_score(y_test, y_pred, average='binary')
recall = recall_score(y_test, y_pred, average='binary')
f1 = f1_score(y_test, y_pred, average='binary')

print("\nPrecision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

# Step 9: Confusion Matrix
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", conf_matrix)

# Step 10: Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No Treatment', 'Treatment'],
yticklabels=['No Treatment', 'Treatment'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```

OUTPUT

```
Column names: Index(['sex', 'age', 'Time', 'Number_of_Warts', 'Type', 'Area',
       'Result_of_Treatment'],
      dtype='object')

First few rows:
    sex  age   Time  Number_of_Warts  Type  Area  Result_of_Treatment
0    1   35  12.00                5     1   100                    0
1    1   29   7.00                5     1    96                    1
2    1   50   8.00                1     3   132                    0
3    1   32  11.75                7     3   750                    0
4    1   67   9.25                1     1    42                    0

Accuracy: 0.8518518518518519

Precision: 0.9090909090909091
Recall: 0.7692307692307693
F1 Score: 0.8333333333333334
```



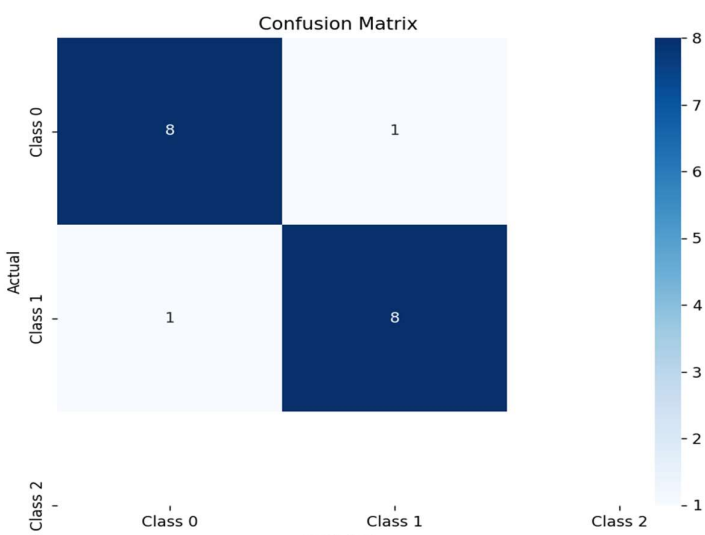
Practical-7
Implement decision tree binary & multi-class classification task.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the Cryotherapy dataset
dataset_path = r'C:\Users\VISHNULAL\Downloads\archive (1)\Cryotherapy.xlsx'
data = pd.read_excel(dataset_path)

# Step 2: Display column names and preview data
print("Column names:", data.columns)
print("\nFirst few rows:\n", data.head())

# Step 3: Define features (X) and target (y)
X = data.drop(columns=['Result_of_Treatment'])
y = data['Result_of_Treatment']

# Step 4: Split the dataset 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)

# Step 5: Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 6: Train a Decision Tree classifier
decision_tree = DecisionTreeClassifier()  # Create a decision tree model
decision_tree.fit(X_train, y_train)  # Fit the model to the training data

# Step 7: Make predictions
y_pred = decision_tree.predict(X_test)

# Step 8: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy)

# Calculate single value metrics for multi-class
precision = precision_score(y_test, y_pred, average='weighted')  # Adjusted for multi-class
recall = recall_score(y_test, y_pred, average='weighted')       # Adjusted for multi-class
f1 = f1_score(y_test, y_pred, average='weighted')              # Adjusted for multi-class

print("\nPrecision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

# Step 9: Confusion Matrix
```

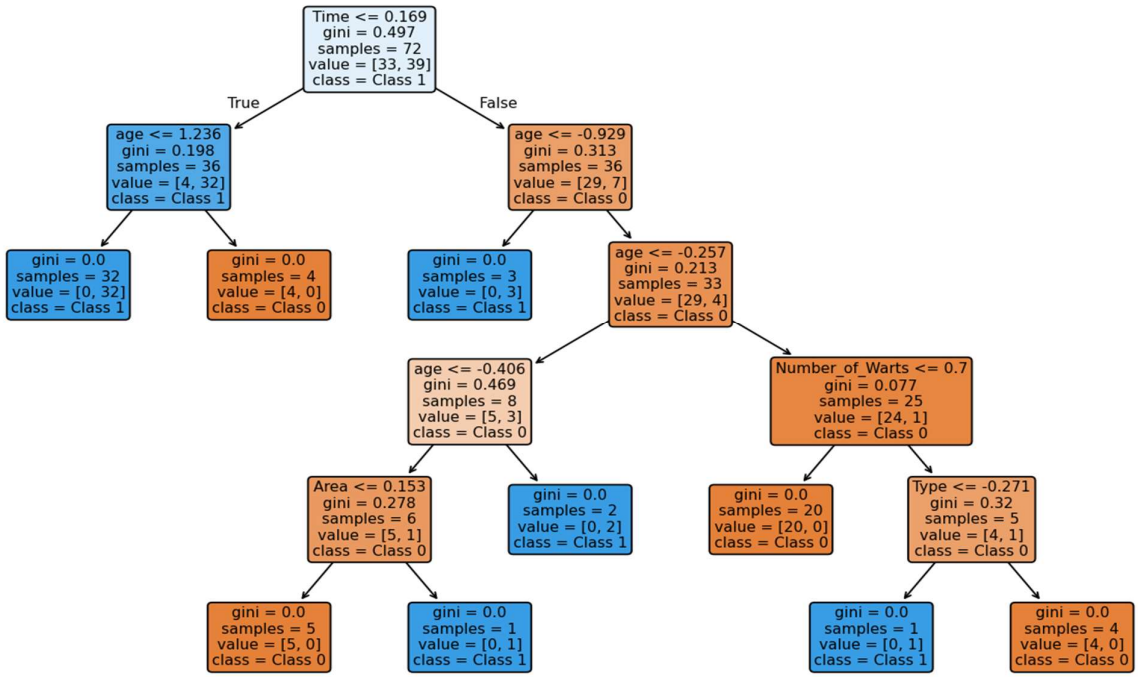
```
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", conf_matrix)

# Step 10: Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1', 'Class 2'],
yticklabels=['Class 0', 'Class 1', 'Class 2'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()

# Step 11: Decision Tree Visualization
plt.figure(figsize=(12, 8))
plot_tree(decision_tree, filled=True, feature_names=X.columns, class_names=['Class 0', 'Class 1', 'Class 2'],
rounded=True)
plt.title('Decision Tree Visualization')
plt.show()
```

OUTPUT

```
Column names: Index(['sex', 'age', 'Time', 'Number_of_Warts', 'Type', 'Area',
       'Result_of_Treatment'],
      dtype='object')

First few rows:
    sex  age   Time  Number_of_Warts  Type  Area  Result_of_Treatment
0    1   35  12.00                5     1   100                    0
1    1   29   7.00                5     1    96                    1
2    1   50   8.00                1     3   132                    0
3    1   32  11.75                7     3   750                    0
4    1   67   9.25                1     1    42                    0

Accuracy: 0.8888888888888888

Precision: 0.8888888888888888
Recall: 0.8888888888888888
F1 Score: 0.8888888888888888
```



Decision Tree Visualization



Practical-8
Implement k-means clustering for real-time application.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load the Cryotherapy dataset
dataset_path = r'C:\Users\VISHNULAL\Downloads\archive (1)\Cryotherapy.xlsx'
data = pd.read_excel(dataset_path)
# Step 2: Display column names and preview data
print("Column names:", data.columns)
print("\nFirst few rows:\n", data.head())
# Step 3: Define features (X) and target (y)
X = data.drop(columns=['Result_of_Treatment'])
y = data['Result_of_Treatment']

# Step 4: Split the dataset 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)

# Step 5: Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 6: Train a Decision Tree classifier
decision_tree = DecisionTreeClassifier()  # Create a decision tree model
decision_tree.fit(X_train, y_train)  # Fit the model to the training data

# Step 7: Make predictions
y_pred = decision_tree.predict(X_test)

# Step 8: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy)
# Calculate single value metrics for multi-class
precision = precision_score(y_test, y_pred, average='weighted')  # Adjusted for multi-class
recall = recall_score(y_test, y_pred, average='weighted')       # Adjusted for multi-class
f1 = f1_score(y_test, y_pred, average='weighted')              # Adjusted for multi-class

print("\nPrecision:", precision)
print("Recall:", recall)
```

```
print("F1 Score:", f1)

# Step 9: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", conf_matrix)

# Step 10: Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1', 'Class
2'], yticklabels=['Class 0', 'Class 1', 'Class 2'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
# Step 3: Define features (X) for clustering
X = data.drop(columns=['Result_of_Treatment'])  # Exclude the target variable
# Step 4: Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Step 5: Determine the optimal number of clusters using the Elbow Method
inertia = []
k_values = range(1, 11)
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Step 6: Plot the Elbow Method
plt.figure(figsize=(8, 5))
plt.plot(k_values, inertia, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.xticks(k_values)
plt.grid()
plt.show()

# Step 7: Fit K-Means with the optimal number of clusters (e.g., k=3)
optimal_k = 3  # You can change this based on the Elbow Method result
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(X_scaled)

# Step 8: Add cluster labels to the original data
data['Cluster'] = kmeans.labels_
# Step 9: Visualize the clusters (using the first two features for simplicity)
plt.figure(figsize=(10, 6))
sns.scatterplot(data=data, x=data.columns[0], y=data.columns[1], hue='Cluster', palette='viridis',
style='Cluster', s=100)
```

```
plt.title('K-Means Clustering Results')
plt.xlabel(data.columns[0])
plt.ylabel(data.columns[1])
plt.legend(title='Cluster')
plt.show()
```

OUTPUT