Practical – 3

Aim: Implement and demonstrate the FIND-S Algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file

- **Code:**

```
import pandas as pd
df = pd.read_csv("/ws.csv")
print("Columns in ws.csv:", df.columns)
features = df.columns[:-1]
target = df.columns[-1]
df[features] = df[features].astype(str)
df[target] = df[target].astype(str)
positive_example = df[df[target] == 'Yes'][features].values
def find_s(example):
  if len(example) == 0:
    return "No positive examples found."
  hypothesis = example[0].copy()
  print("Initial hypothesis:", hypothesis)
  for i, instance in enumerate(example[1: ]):
      for j in range(len(hypothesis)):
        if hypothesis[j] != instance[j]:
          hypothesis[j] = '?'
      print(f"Hypothesis after example {i+2}:", hypothesis)
  return hypothesis
hypothesis = find_s(positive_example)
print("Most specific hypothesis:", hypothesis)
```

- **Output:**

```
Columns in ws.csv: Index(['Sunny', 'Warm', 'Normal', 'Strong', 'Warm.1', 'Same', 'Yes'], dtype='object')
Initial hypothesis: ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
Hypothesis after example 2: ['Sunny' 'Warm' 'High' 'Strong' '?' '?']
Most specific hypothesis: ['Sunny' 'Warm' 'High' 'Strong' '?' '?']
```

Practical – 4

Aim: Write a Python program to implement Simple Linear Regression

- How many total observations in data?
- How many independent variables?
- Which is a dependent variable?
- Quantify the goodness of your model and discuss steps taken for improvement (RMSE, SSE, R2Score).

- **Code:**

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

df = pd.read_csv("/Salary_Data.csv")
df.head()
df.describe()
X = df.iloc[:, :1]  # independent
y = df.iloc[:, 1:]  # dependent
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred_test = regressor.predict(X_test)
y_pred_train = regressor.predict(X_train)
rmse = np.sqrt(mean_squared_error(y_test, y_pred_test))
sse = np.sum((y_test - y_pred_test) ** 2)
r2 = r2_score(y_test, y_pred_test)
print(f"\nModel Evaluation:")
print(f"RMSE: {rmse:.4f}")
print(f"SSE: {sse.iloc[0]:.4f}")
print(f"R² Score: {r2:.4f}")
plt.scatter(X_train, y_train, color = 'green')
plt.plot(X_train, y_pred_train, color = 'firebrick')
plt.title('Salary vs Experience (Training Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend(['X_train/Pred(y_test)', 'X_train/y_train'], title = 'Sal/Exp', loc='best',
facecolor='lightblue')
plt.box(False)
plt.show()
```

- **Output:**

```
Model Evaluation:
RMSE: 3580.9792
SSE: 76940473.7888
R² Score: 0.9882
```

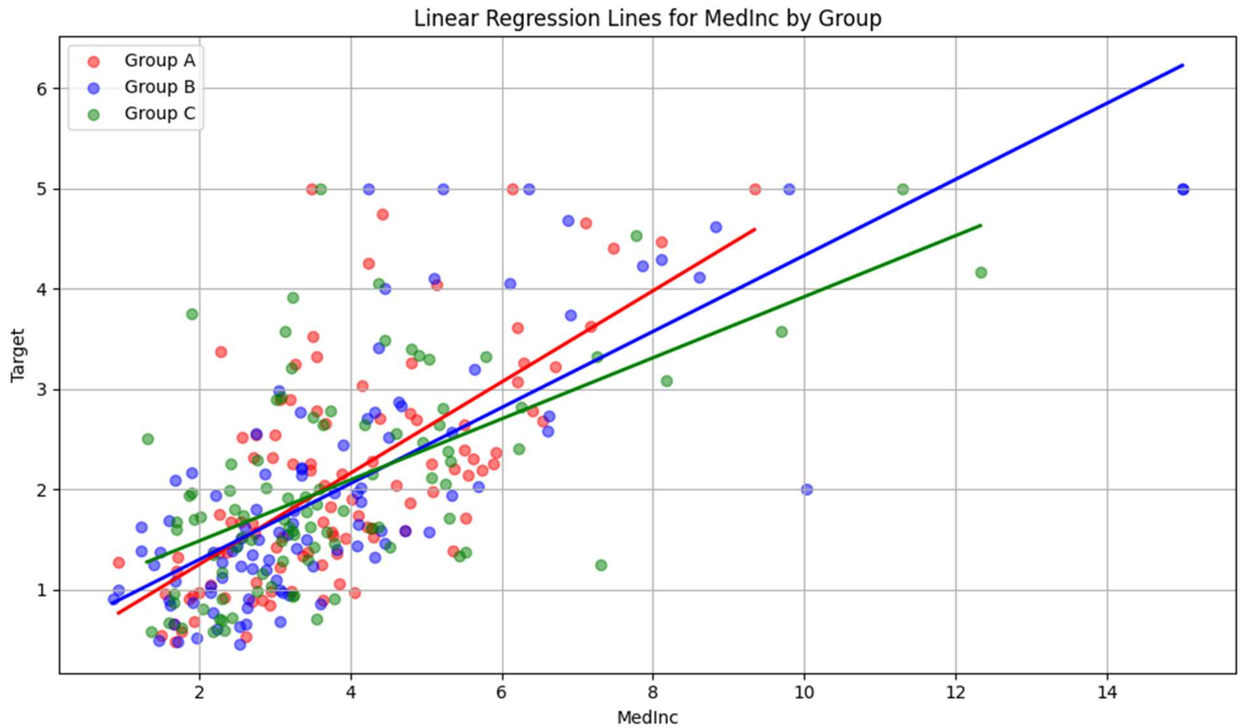
Practical – 5

Aim: Implementation of Multiple Linear Regression for House Price Prediction using sklearn.

- **Code:**

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

from sklearn.linear_model import LinearRegression
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
df = pd.DataFrame(housing.data, columns=housing.feature_names)
df['Target'] = housing.target
df = df.sample(300, random_state=42).reset_index(drop=True)
df['Group'] = np.random.choice(['A', 'B', 'C'], size=len(df))
x_feature = 'MedInc'
y_feature = 'Target'
plt.figure(figsize=(10, 6))
colors = {'A': 'red', 'B': 'blue', 'C': 'green'}
for group in df['Group'].unique():
    subset = df[df['Group'] == group]
    X = pd.DataFrame(subset[[x_feature]])
    y = subset[y_feature]
    model = LinearRegression()
    model.fit(X, y)
    x_line = np.linspace(X[x_feature].min(), X[x_feature].max(), 100).reshape(-1, 1)
    y_line = model.predict(pd.DataFrame(x_line, columns=[x_feature]))
    plt.scatter(X, y, alpha=0.5, label=f'Group {group}', color=colors[group])
    plt.plot(x_line, y_line, color=colors[group], linewidth=2)
plt.xlabel(x_feature)
plt.ylabel(y_feature)
plt.title(f"Linear Regression Lines for {x_feature} by Group")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

- **Output:**


Linear Regression Lines for MedInc by Group
Group A
Group B
Group C
Target
MedInc
1
2
3
4
5
6
2
4
6
8
10
12
14

Practical – 6

Aim: Two Class Classification (Logistic Regression)

- How many total observations in data?
- How many independent variables?
- Which is a dependent variable?
- Implement logistic function.
- Implement Log-loss function.
- Quantify the goodness of your model and discuss steps taken for improvement (Accuracy, Confusion matrices, F-measure).

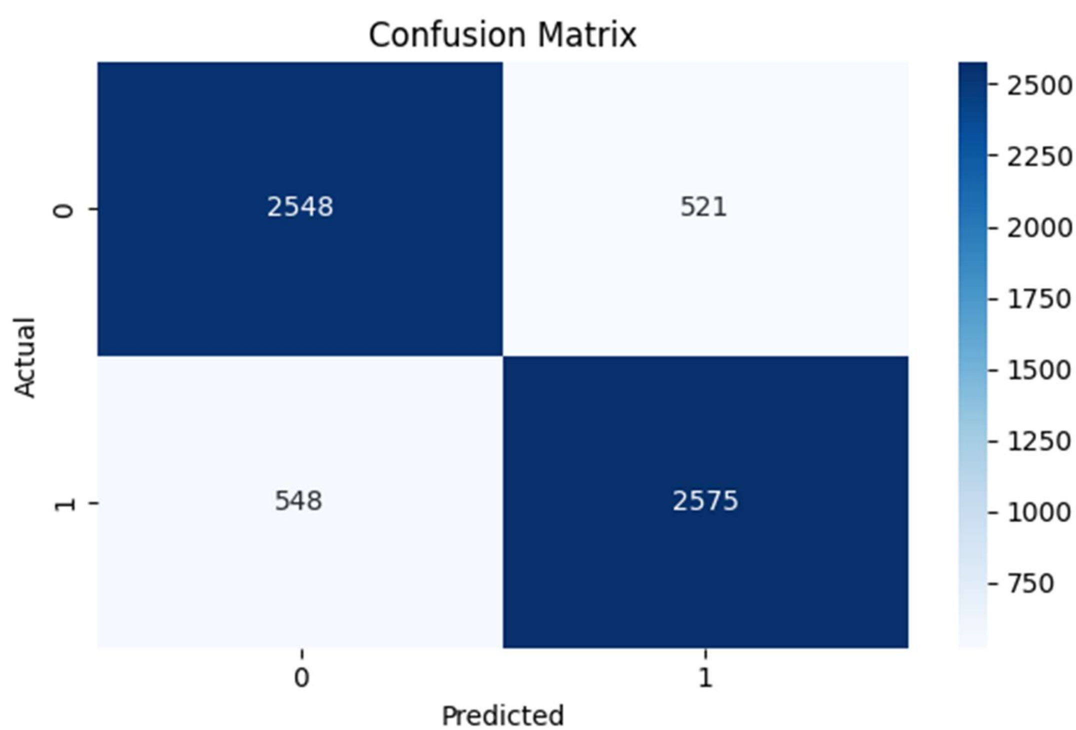
- **Code:**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score, confusion_matrix, f1_score, log_loss, classification_report
)
housing = fetch_california_housing()
df = pd.DataFrame(housing.data, columns=housing.feature_names)
df['Target'] = housing.target
median_value = df['Target'].median()
df['Target_Binary'] = (df['Target'] > median_value).astype(int)
X = df.drop(columns=['Target', 'Target_Binary'])
y = df['Target_Binary']
print("Total observations:", len(df))
print("Number of independent variables:", X.shape[1])
print("Dependent variable: Target_Binary")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_pred_prob = model.predict_proba(X_test)[:, 1]
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
logloss = log_loss(y_test, y_pred_prob)
print("\nAccuracy:", accuracy)
print("F1 Score:", f1)
print("Log Loss (sklearn):", logloss)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
def logistic(z):
    return 1 / (1 + np.exp(-z))
def log_loss_manual(y_true, y_prob):
```

```
    eps = 1e-15
    y_prob = np.clip(y_prob, eps, 1 - eps)
    return -np.mean(y_true * np.log(y_prob) + (1 - y_true) * np.log(1 - y_prob))
manual_logloss = log_loss_manual(y_test.values, y_pred_prob)
print("Manual Log Loss:", manual_logloss)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

- **Output:**

```
Total observations: 20640
Number of independent variables: 8
Dependent variable: Target_Binary

Accuracy: 0.8273578811369509
F1 Score: 0.8281074127673259
Log Loss (sklearn): 0.38525281617941143

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.83      0.83      3069
           1       0.83      0.82      0.83      3123

    accuracy                           0.83      6192
   macro avg       0.83      0.83      0.83      6192
weighted avg       0.83      0.83      0.83      6192

Manual Log Loss: 0.38525281617941143
```



Practical – 7

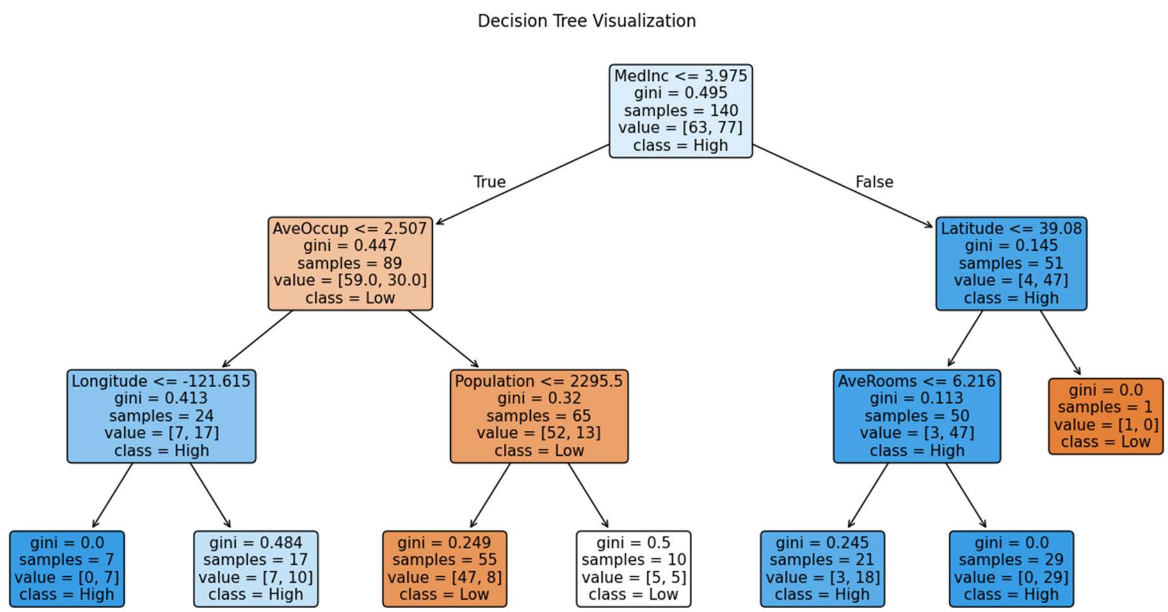
Aim: Implementation of Decision tree using sklearn and its parameter tuning.

- **Code:**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

housing = fetch_california_housing()
df = pd.DataFrame(housing.data, columns=housing.feature_names)
df['Target'] = housing.target
df = df.sample(n=200, random_state=42).reset_index(drop=True)
df['Target_Binary'] = (df['Target'] > df['Target'].median()).astype(int)
X = df.drop(columns=['Target', 'Target_Binary'])
y = df['Target_Binary']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
clf = DecisionTreeClassifier(random_state=42)
param_grid = {
    'max_depth': [3, 5, None],
    'min_samples_split': [2, 10],
    'criterion': ['gini', 'entropy']
}
grid = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)
best_tree = grid.best_estimator_
print("Best Parameters:", grid.best_params_)
y_pred = best_tree.predict(X_test)
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print("\nAccuracy:", acc)
print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
plt.figure(figsize=(16, 8))
plot_tree(best_tree, feature_names=X.columns, class_names=["Low", "High"], filled=True,
rounded=True)
plt.title("Decision Tree Visualization")
plt.show()
```

- **Output:**

```
Best Parameters: {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 2}

Accuracy: 0.75

Confusion Matrix:
 [[25 12]
 [ 3 20]]

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.68      0.77        37
           1       0.62      0.87      0.73        23

    accuracy                           0.75        60
   macro avg       0.76      0.77      0.75        60
weighted avg       0.79      0.75      0.75        60
```



Practical – 9

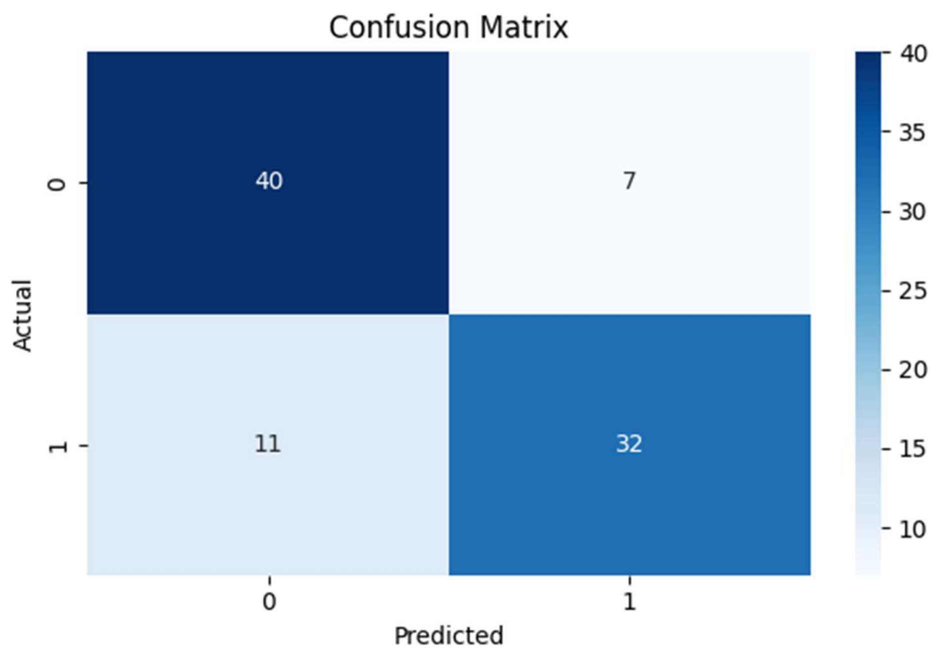
Aim: Write a program to implement Random Forest Algorithm

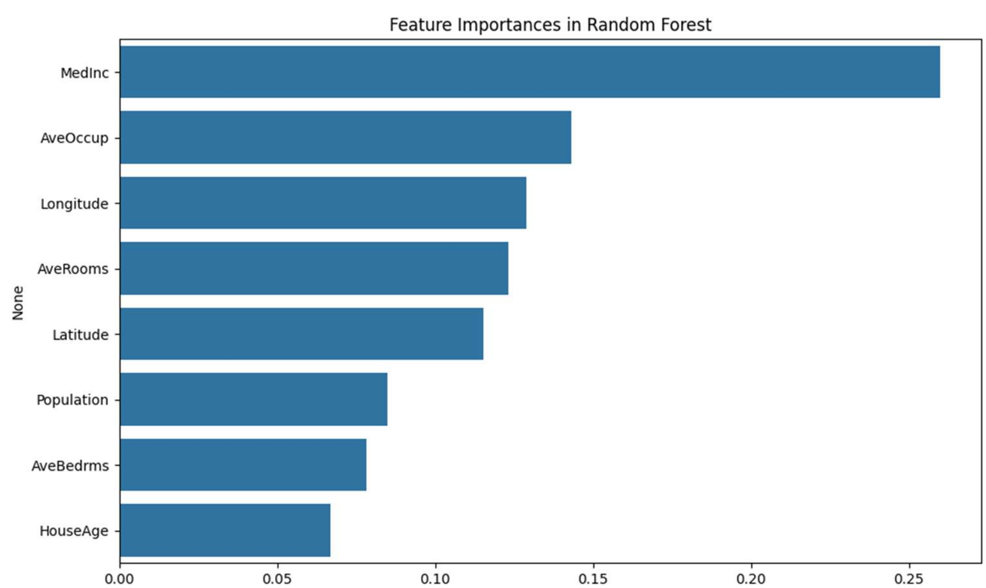
- **Code:**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

housing = fetch_california_housing()
df = pd.DataFrame(housing.data, columns=housing.feature_names)
df['Target'] = housing.target
df = df.sample(n=300, random_state=42).reset_index(drop=True)
median_value = df['Target'].median()
df['Target_Binary'] = (df['Target'] > median_value).astype(int)
X = df.drop(columns=['Target', 'Target_Binary'])
y = df['Target_Binary']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
rf = RandomForestClassifier(
    n_estimators=100,
    max_depth=None,
    random_state=42
)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print("Accuracy:", acc)
print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
importances = rf.feature_importances_
feature_names = X.columns
sorted_idx = np.argsort(importances)[::-1]
plt.figure(figsize=(10, 6))
sns.barplot(x=importances[sorted_idx], y=feature_names[sorted_idx])
plt.title("Feature Importances in Random Forest")
plt.tight_layout()
plt.show()
```

- **Output:**

```
Confusion Matrix:
 [[40  7]
 [11 32]]

Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.85      0.82        47
           1       0.82      0.74      0.78        43

    accuracy                           0.80        90
   macro avg       0.80      0.80      0.80        90
weighted avg       0.80      0.80      0.80        90
```





Practical – 10

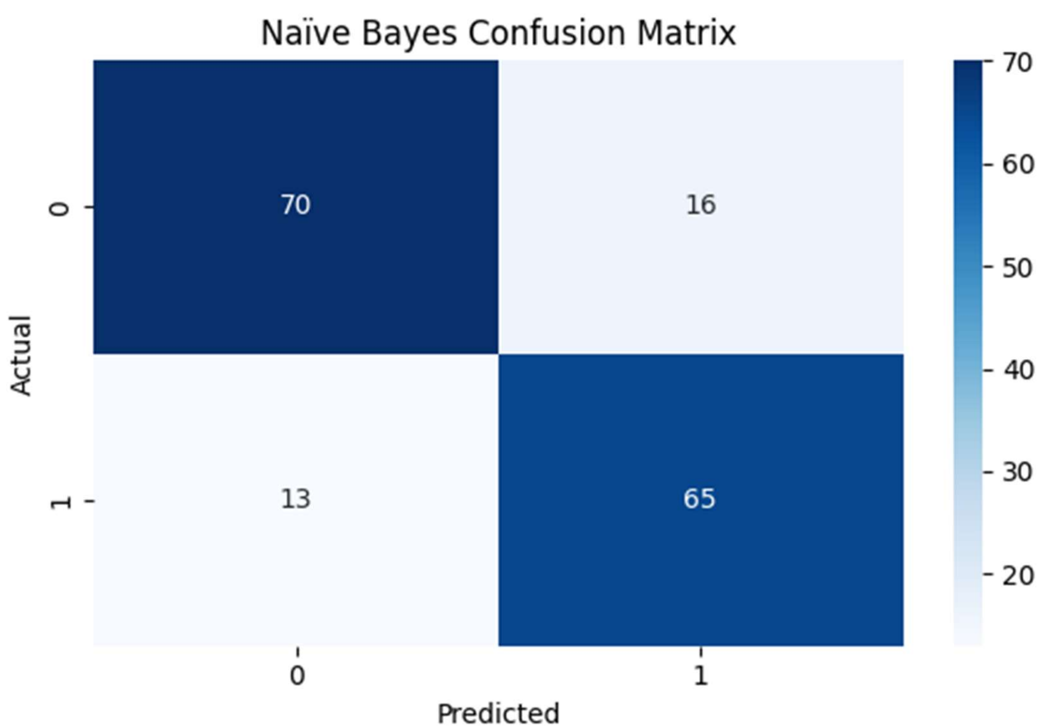
Aim: Write a program to implement Random Forest Algorithm

- **Code:**

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("/content/drive/MyDrive/temp/1_Housing.csv")
# converting data into binaries 1 for high and 0 for low
median_price = df['price'].median()
df['price_category'] = (df['price'] > median_price).astype(int)
df.drop(columns=['price'], inplace=True)
categorical_cols = df.select_dtypes(include='object').columns
le = LabelEncoder()
for col in categorical_cols:
    df[col] = le.fit_transform(df[col])
X = df.drop(columns=['price_category'])
y = df['price_category']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Naïve Bayes Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

- **Output:**

```
Accuracy: 0.823170731707317

Confusion Matrix:
 [[70 16]
 [13 65]]

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.81      0.83        86
           1       0.80      0.83      0.82        78

    accuracy                           0.82       164
   macro avg       0.82      0.82      0.82       164
weighted avg       0.82      0.82      0.82       164
```



Practical – 11

Aim: Write a program to implement OR, AND gate using Perceptron with learning rules.

- **Code:**

```
import numpy as np

def step_function(x):
    return 1 if x >= 0 else 0
def train_perceptron(inputs, labels, learning_rate=0.1, iterations=5):
    weights = np.zeros(inputs.shape[1])
    bias = 0
    for iteration in range(iterations):
        print(f"\nIteration no: {iteration+1}")
        for x, label in zip(inputs, labels):
            z = np.dot(x, weights) + bias
            y_pred = step_function(z)
            error = label - y_pred
            weights += learning_rate * error * x
            bias += learning_rate * error
            print(f"Input: {x}, Pred: {y_pred}, Error: {error}, Updated Weights: {weights}, Bias: {bias}")
    return weights, bias
def predict(inputs, weights, bias):
    return [step_function(np.dot(x, weights) + bias) for x in inputs]
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
or_labels = np.array([0,1,1,1])
and_labels = np.array([0,0,0,1])
# OR Gate
print("Training Perceptron for OR Gate")
or_weights, or_bias = train_perceptron(inputs, or_labels)
or_outputs = predict(inputs, or_weights, or_bias)
print("\nFinal OR Predictions:")
for i, o in zip(inputs, or_outputs):
    print(f"Input: {i}, Output: {o}")
# AND Gate
print("\n\nTraining Perceptron for AND Gate")
and_weights, and_bias = train_perceptron(inputs, and_labels)
and_outputs = predict(inputs, and_weights, and_bias)
print("\nFinal AND Predictions:")
for i, o in zip(inputs, and_outputs):
    print(f"Input: {i}, Output: {o}")
```

- **Output:**

```
Training Perceptron for OR Gate

Iteration no: 1
Input: [0 0], Pred: 1, Error: -1, Updated Weights: [0. 0.], Bias: -0.1
Input: [0 1], Pred: 0, Error: 1, Updated Weights: [0.  0.1], Bias: 0.0
Input: [1 0], Pred: 1, Error: 0, Updated Weights: [0.  0.1], Bias: 0.0
Input: [1 1], Pred: 1, Error: 0, Updated Weights: [0.  0.1], Bias: 0.0

Iteration no: 2
Input: [0 0], Pred: 1, Error: -1, Updated Weights: [0.  0.1], Bias: -0.1
Input: [0 1], Pred: 1, Error: 0, Updated Weights: [0.  0.1], Bias: -0.1
Input: [1 0], Pred: 0, Error: 1, Updated Weights: [0.1 0.1], Bias: 0.0
Input: [1 1], Pred: 1, Error: 0, Updated Weights: [0.1 0.1], Bias: 0.0

Iteration no: 3
Input: [0 0], Pred: 1, Error: -1, Updated Weights: [0.1 0.1], Bias: -0.1
Input: [0 1], Pred: 1, Error: 0, Updated Weights: [0.1 0.1], Bias: -0.1
Input: [1 0], Pred: 1, Error: 0, Updated Weights: [0.1 0.1], Bias: -0.1
Input: [1 1], Pred: 1, Error: 0, Updated Weights: [0.1 0.1], Bias: -0.1

Iteration no: 4
Input: [0 0], Pred: 0, Error: 0, Updated Weights: [0.1 0.1], Bias: -0.1
Input: [0 1], Pred: 1, Error: 0, Updated Weights: [0.1 0.1], Bias: -0.1
Input: [1 0], Pred: 1, Error: 0, Updated Weights: [0.1 0.1], Bias: -0.1
Input: [1 1], Pred: 1, Error: 0, Updated Weights: [0.1 0.1], Bias: -0.1

Iteration no: 5
Input: [0 0], Pred: 0, Error: 0, Updated Weights: [0.1 0.1], Bias: -0.1
Input: [0 1], Pred: 1, Error: 0, Updated Weights: [0.1 0.1], Bias: -0.1
Input: [1 0], Pred: 1, Error: 0, Updated Weights: [0.1 0.1], Bias: -0.1
Input: [1 1], Pred: 1, Error: 0, Updated Weights: [0.1 0.1], Bias: -0.1

Final OR Predictions:
Input: [0 0], Output: 0
Input: [0 1], Output: 1
Input: [1 0], Output: 1
Input: [1 1], Output: 1
```

```
Training Perceptron for AND Gate

Iteration no: 1
Input: [0 0], Pred: 1, Error: -1, Updated Weights: [0. 0.], Bias: -0.1
Input: [0 1], Pred: 0, Error: 0, Updated Weights: [0. 0.], Bias: -0.1
Input: [1 0], Pred: 0, Error: 0, Updated Weights: [0. 0.], Bias: -0.1
Input: [1 1], Pred: 0, Error: 1, Updated Weights: [0.1 0.1], Bias: 0.0

Iteration no: 2
Input: [0 0], Pred: 1, Error: -1, Updated Weights: [0.1 0.1], Bias: -0.1
Input: [0 1], Pred: 1, Error: -1, Updated Weights: [0.1 0. ], Bias: -0.2
Input: [1 0], Pred: 0, Error: 0, Updated Weights: [0.1 0. ], Bias: -0.2
Input: [1 1], Pred: 0, Error: 1, Updated Weights: [0.2 0.1], Bias: -0.1

Iteration no: 3
Input: [0 0], Pred: 0, Error: 0, Updated Weights: [0.2 0.1], Bias: -0.1
Input: [0 1], Pred: 1, Error: -1, Updated Weights: [0.2 0. ], Bias: -0.2
Input: [1 0], Pred: 1, Error: -1, Updated Weights: [0.1 0. ], Bias: -0.30000000000000004
Input: [1 1], Pred: 0, Error: 1, Updated Weights: [0.2 0.1], Bias: -0.20000000000000004

Iteration no: 4
Input: [0 0], Pred: 0, Error: 0, Updated Weights: [0.2 0.1], Bias: -0.20000000000000004
Input: [0 1], Pred: 0, Error: 0, Updated Weights: [0.2 0.1], Bias: -0.20000000000000004
Input: [1 0], Pred: 0, Error: 0, Updated Weights: [0.2 0.1], Bias: -0.20000000000000004
Input: [1 1], Pred: 1, Error: 0, Updated Weights: [0.2 0.1], Bias: -0.20000000000000004

Iteration no: 5
Input: [0 0], Pred: 0, Error: 0, Updated Weights: [0.2 0.1], Bias: -0.20000000000000004
Input: [0 1], Pred: 0, Error: 0, Updated Weights: [0.2 0.1], Bias: -0.20000000000000004
Input: [1 0], Pred: 0, Error: 0, Updated Weights: [0.2 0.1], Bias: -0.20000000000000004
Input: [1 1], Pred: 1, Error: 0, Updated Weights: [0.2 0.1], Bias: -0.20000000000000004

Final AND Predictions:
Input: [0 0], Output: 0
Input: [0 1], Output: 0
Input: [1 0], Output: 0
Input: [1 1], Output: 1
```

Practical – 12

Aim: Build an Artificial Neural Network by implementing the Backpropagation Algorithm.

- **Code:**

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def sigmoid_derivative(x):
    return x * (1 - x)
def mse(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [1]])
np.random.seed(42)
input_size = 2
hidden_size = 4
output_size = 1
W1 = np.random.randn(input_size, hidden_size)
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size)
b2 = np.zeros((1, output_size))
learning_rate = 0.1
epochs = 10000
for epoch in range(epochs):
    # Forward pass
    z1 = np.dot(X, W1) + b1
    a1 = sigmoid(z1)
    z2 = np.dot(a1, W2) + b2
    a2 = sigmoid(z2)
    # Backward pass
    error = y - a2
    d_a2 = error * sigmoid_derivative(a2)
    d_W2 = np.dot(a1.T, d_a2)
    d_b2 = np.sum(d_a2, axis=0, keepdims=True)
    d_a1 = np.dot(d_a2, W2.T) * sigmoid_derivative(a1)
    d_W1 = np.dot(X.T, d_a1)
    d_b1 = np.sum(d_a1, axis=0, keepdims=True)
    # update weights
    W2 += learning_rate * d_W2
    b2 += learning_rate * d_b2
    W1 += learning_rate * d_W1
    b1 += learning_rate * d_b1
    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {mse(y, a2):.6f}")
print("\nFinal outputs after training: ")
print(np.round(a2))
```

- **Output:**

```
Epoch 0, Loss: 0.387145
Epoch 1000, Loss: 0.026790
Epoch 2000, Loss: 0.005534
Epoch 3000, Loss: 0.002624
Epoch 4000, Loss: 0.001640
Epoch 5000, Loss: 0.001168
Epoch 6000, Loss: 0.000897
Epoch 7000, Loss: 0.000723
Epoch 8000, Loss: 0.000602
Epoch 9000, Loss: 0.000515

Final outputs after training:
[[0.]
 [1.]
 [1.]
 [1.]]
```