VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS

Grade A++ Accredited Institution by NAAC

NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC; Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE An ISO 9001:2015 Certified Institution

SCHOOL OF ENGINEERING & TECHNOLOGY

B.Tech Programme: CSE

Course Title: Supervised Deep Learning

Course Code: ML 463P

Submitted To:

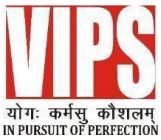
Ms. Priyanka
Assistant Professor

Submitted By:

Name: Ishaann Jain

Enrollment No: 06117702722

Branch & Section: CSE B



VISION OF INSTITUTE

To be an educational institute that empowers the field of engineering to build a sustainable future by providing quality education with innovative practices that supports people, planet and profit.

MISSION OF INSTITUTE

To groom the future engineers by providing value-based education and awakening students' curiosity, nurturing creativity and building capabilities to enable them to make significant contributions to the world.

EXPERIMENT 1

AIM: Linear regression: Implement linear regression on a dataset and evaluate the model's performance.

THEORY:

Linear Regression is a fundamental statistical and machine learning algorithm used to model the relationship between a dependent variable (Y) and one or more independent variables (X). When there is only one independent variable, it is called Simple Linear Regression. The goal of Simple Linear Regression is to find the best-fitting straight line, also known as the regression line, that minimizes the difference between the actual and predicted values.

The equation of a simple linear regression model is:

$Y = wX + b$

Where:
Y = Dependent variable (target/output)
X = Independent variable (input)
w = weight
b = bias (constant term)

The values of w and b are determined using the Least Squares Method, which minimizes the sum of squared differences between actual and predicted values.

Algorithm:

The working of Simple Linear Regression involves the following steps:

1. **Data Collection –** Gather historical data containing input-output pairs.
2. **Data Preprocessing –** Handle missing values, remove outliers, and normalize data if necessary.
3. **Splitting Data –** Divide the dataset into training and testing sets to evaluate model performance.
4. **Model Training –** Fit the Simple Linear Regression model to the training data using the Least Squares Method.
5. **Prediction –** Use the trained model to predict new values.

6. **Performance Evaluation –** Measure model accuracy using metrics like Mean Squared Error (MSE) and R-squared (R^2) score.

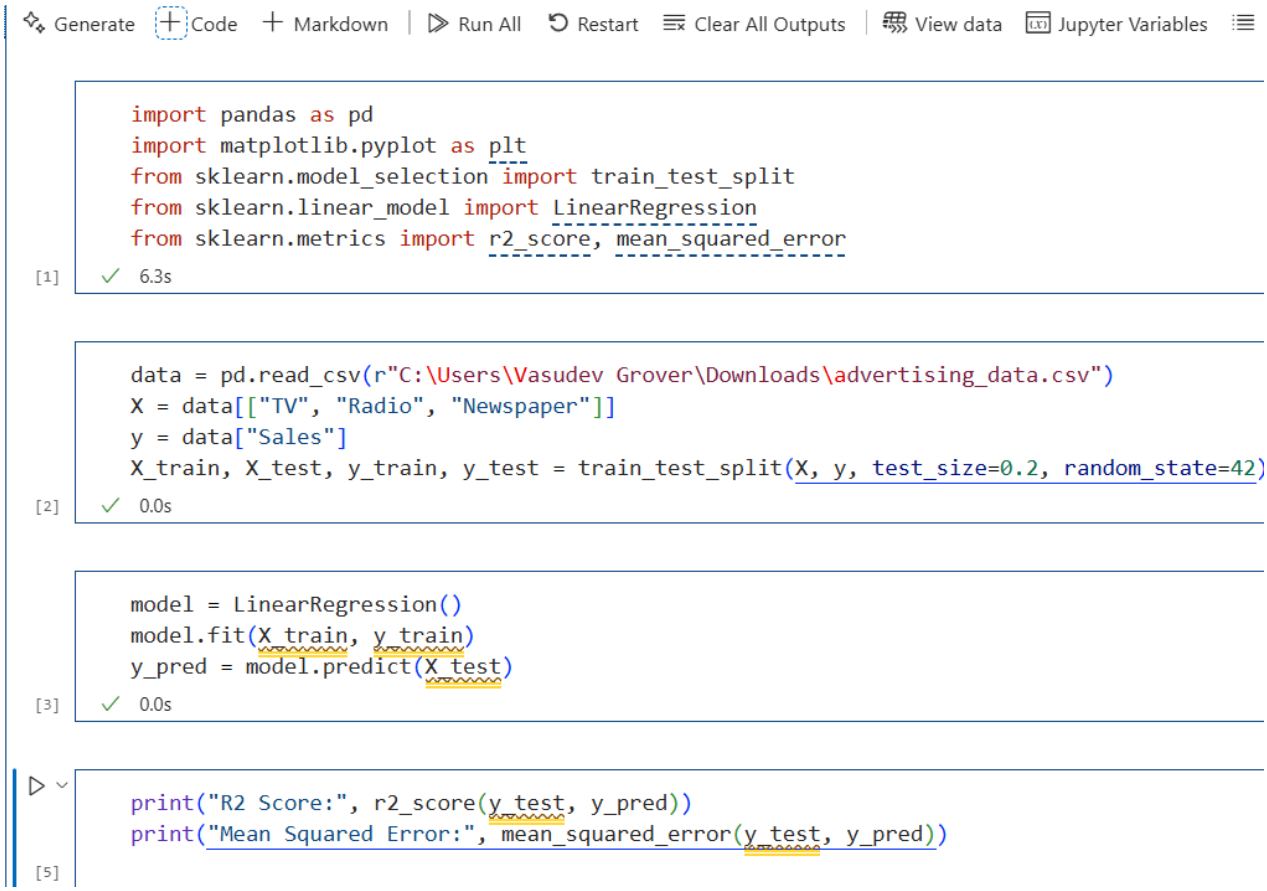
ABOUT THE DATASET:

The dataset used in this experiment is the **Advertising Dataset**, which contains information about advertising expenditures across different media and the corresponding sales. It consists of **50 rows** (observations) and **4 columns** (features).

Attributes

1. **TV** – Advertising budget spent on TV (in thousands of dollars).
2. **Radio** – Advertising budget spent on Radio (in thousands of dollars).
3. **Newspaper** – Advertising budget spent on Newspaper (in thousands of dollars).
4. **Sales** – The dependent variable, representing the sales generated (in thousands of units).

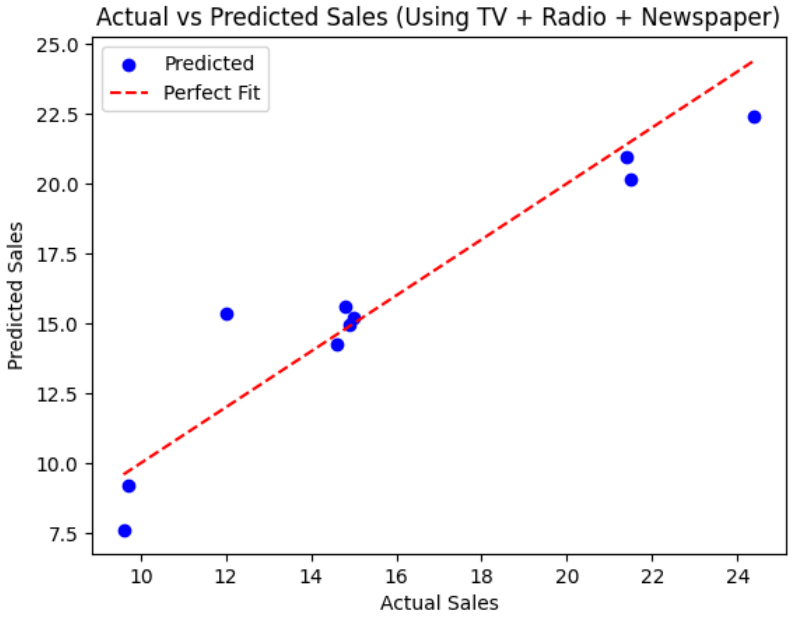
SOURCE CODE:



```
plt.scatter(y_test, y_pred, color="blue", label="Predicted")
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()],
         color="red", linestyle="--", label="Perfect Fit")
plt.xlabel("Actual Sales")
plt.ylabel("Predicted Sales")
plt.title("Actual vs Predicted Sales (TV + Radio + Newspaper)")
plt.legend()
plt.show()
```

OUTPUT:

```
R2 Score: 0.9028868775738176
Mean Squared Error: 2.2509753534042387
```



LEARNING OUTCOME:

EXPERIMENT 2

AIM: Logistic Regression: Implement logistic regression on a binary classification dataset and evaluate the model's performance.

THEORY:

Logistic Regression is a classification algorithm used in Machine Learning to predict categorical outcomes. It is mainly used for **binary classification**, where the target variable has two possible values (e.g., Yes/No, 0/1, True/False). Instead of fitting a straight line as in **Linear Regression**, it applies the **sigmoid function** to map predicted values between **0 and 1**.

Mathematical Formulation:

The logistic function (sigmoid function) is given by:

$$\sigma(z) = \frac{1}{1 + e^{(-z)}}$$

where z is the linear combination of input features:

$z = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n$

The output represents the probability of belonging to a particular class. A threshold (usually 0.5) determines the final classification.

Algorithm:

- **Data Collection** – Load and analyze the dataset.
- **Preprocessing** – Handle missing values, normalize features, and split the dataset.
- **Model Training** – Apply logistic regression using Gradient Descent to optimize weights.
- **Prediction** – Classify new observations based on computed probabilities.
- **Evaluation** – Assess model performance using metrics like Accuracy, Precision, Recall, and F1-score.

ABOUT THE DATASET:

The dataset used in this experiment is the **PIMA Indians Diabetes Dataset**, a well-known medical dataset for diabetes prediction. It contains **768 rows** (patient

records) and **9 columns** (attributes). The dataset is often used for binary classification, where the goal is to predict whether a patient has diabetes (**Outcome = 1**) or not (**Outcome = 0**).

Attributes:

1. **Pregnancies** – Number of times the patient has been pregnant.
2. **Glucose** – Plasma glucose concentration (2 hours after an oral glucose tolerance test).
3. **BloodPressure** – Diastolic blood pressure (mm Hg).
4. **SkinThickness** – Triceps skinfold thickness (mm).
5. **Insulin** – 2-hour serum insulin level (mu U/ml).
6. **BMI** – Body Mass Index, calculated as weight (kg) / height (m^2).
7. **DiabetesPedigreeFunction** – A score representing the genetic relationship of diabetes in the family.
8. **Age** – Age of the patient (in years).
9. **Outcome** – Target variable:
    - **0** → Patient does not have diabetes
    - **1** → Patient has diabetes

SOURCE CODE:

```
[9]
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
```

```
[10]
df = pd.read_csv(r"C:\Users\Vasudev Grover\Downloads\diabetes_datasets.csv")
print(df.head())
```

```
...    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   Diabetespedigreefunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

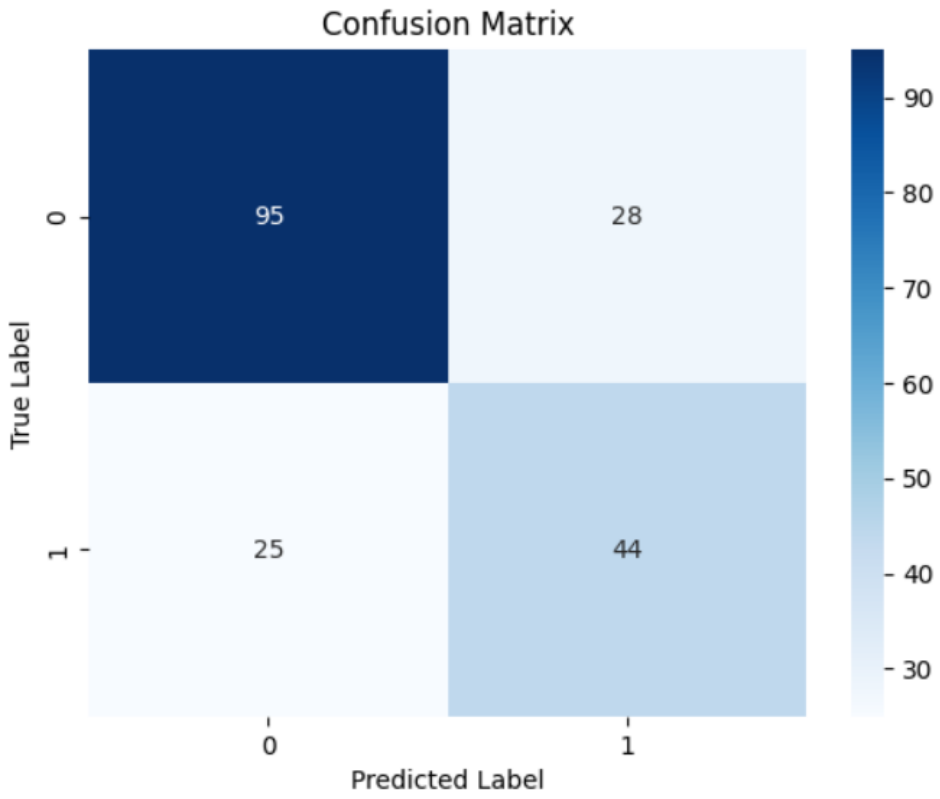
```
X = df.drop('Outcome',axis = 1)
y = df["Outcome"]
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.25, random_state = 42)
```

```
model = LogisticRegression(max_iter = 1000)
model.fit(X_train, y_train)
prediction = model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, prediction)
conf_matrix = confusion_matrix(y_test, prediction)
print(f'Accuracy: {accuracy * 100:.2f}%')
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

OUTPUT:

```
Accuracy: 72.40%
```



LEARNING OUTCOME:

EXPERIMENT 3

AIM: k-Nearest Neighbors (k-NN): Implement k-NN algorithm on a dataset and evaluate the model's performance.

THEORY:

The k-Nearest Neighbours (k-NN) algorithm is a supervised learning technique used for both classification and regression problems. It is one of the simplest machine learning algorithms, based on the principle of similarity. The fundamental idea behind k-NN is that data points with similar features tend to belong to the same category.
k-NN is a lazy learning algorithm, meaning it does not create an explicit model during the training phase. Instead, it stores the training data and classifies new instances by finding the k closest training examples.

Key Characteristics of k-NN:

- **Instance-based Learning:** The algorithm memorizes training data rather than learning a general model.
- **Non-parametric:** No assumption is made about the underlying data distribution.
- **Versatile:** Can be used for both classification and regression tasks.
- **Computational Complexity:** Increases with dataset size, as it requires searching for nearest neighbors.

Algorithm:

1. Choose the value of k (the number of nearest neighbours to consider).
2. Calculate the distance between the test sample and all training samples using a distance metric such as:

- Euclidean Distance (most common):

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Manhattan Distance
- Minkowski Distance

3. Find the k nearest neighbours (smallest distances).
4. Assign the class label based on majority voting from the k-nearest neighbours.
5. Return the predicted class.

ABOUT THE DATASET:

The Iris dataset is a well-known dataset used for classification tasks. It contains 150 samples of iris flowers from three species:

- Setosa
- Versicolor
- Virginica

Each sample has four features:

1. Sepal Length (cm)
2. Sepal Width (cm)
3. Petal Length (cm)
4. Petal Width (cm)

Target Variable:

- Species (Categorical: Setosa, Versicolor, Virginica)

Dataset Source:

The dataset is originally from Fisher's Iris dataset and is widely used for pattern recognition in Machine Learning.

SOURCE CODE:

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

dataset = load_iris()

X = dataset.data
y = dataset.target

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=41)

model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%\n')

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

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d",
            xticklabels=dataset.target_names,
            yticklabels=dataset.target_names)
plt.title("Confusion Matrix - KNN on Iris Dataset")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

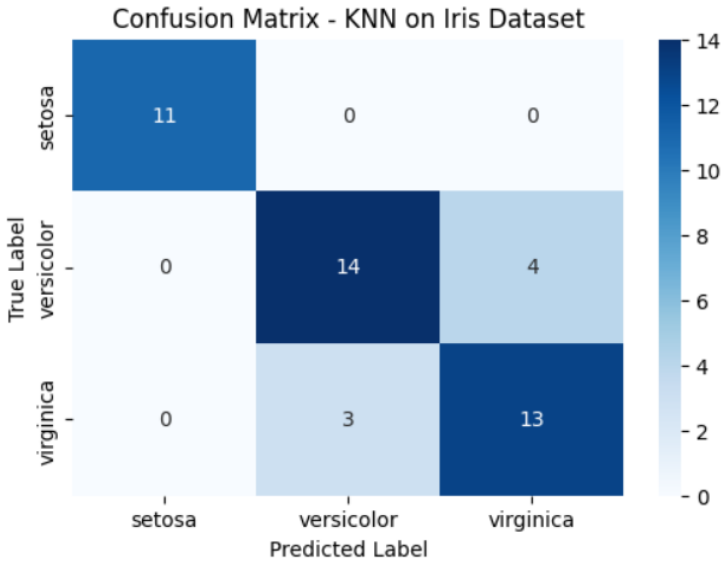
OUTPUT:

```
Accuracy: 84.44%

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       0.82      0.78      0.80        18
           2       0.76      0.81      0.79        16

    accuracy                           0.84        45
   macro avg       0.86      0.86      0.86        45
weighted avg       0.85      0.84      0.84        45
```



LEARNING OUTCOME:

EXPERIMENT 4

AIM: Decision Trees: Implement decision trees on a dataset and evaluate the model's performance.

THEORY:

A Decision Tree is a supervised machine learning algorithm used for classification and regression tasks. It is a tree-like structure where:

- Each internal node represents a decision based on an attribute.
- Each branch represents the outcome of the decision.
- Each leaf node represents a class label (in classification problems).

The Decision Tree is constructed using recursive partitioning, where the dataset is split into subsets based on the best attribute at each level.

Advantages of Decision Trees:

1. Easy to interpret and visualize.
2. Works well with both numerical and categorical data.
3. Requires minimal data preprocessing (no need for feature scaling).
4. Can handle multi-class classification problems effectively.

Disadvantages of Decision Trees:

1. Prone to overfitting, especially with deep trees.
2. Small changes in data can lead to different tree structures.
3. Greedy approach may not always lead to the optimal tree.

Algorithm:

1) Compute Entropy: The entropy of the dataset is calculated to measure uncertainty.

$$H(S) = -\sum p_i \log_2 p_i$$

2) Calculate Information Gain:

Information Gain is used to decide the best attribute for splitting.

$$IG(S, A) = H(S) - \sum \left(\frac{|S_v|}{|S|} \right) H(S_v)$$

3) Split on the Best Attribute:

The attribute with the highest Information Gain is chosen for the next split.

4) Repeat Until Stopping Condition:

- If all samples belong to the same class → Stop
- If no more attributes left → Stop

If dataset is empty → Stop

ABOUT THE DATASET:

The Iris dataset is a well-known dataset used for classification tasks. It contains 150 samples of iris flowers from three species:

- Setosa
- Versicolor
- Virginica

Each sample has four features:

1. Sepal Length (cm)
2. Sepal Width (cm)
3. Petal Length (cm)
4. Petal Width (cm)

Target Variable:

- Species (Categorical: Setosa, Versicolor, Virginica)

Dataset Source:

The dataset is originally from Fisher's Iris dataset and is widely used for pattern recognition in Machine Learning.

SOURCE CODE:

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

```
dataset = pd.read_csv("Iris.csv")
X = dataset[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"]]
y = dataset["Species"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
model = DecisionTreeClassifier(criterion="entropy")
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print(classification_report(y_test, y_pred))
```

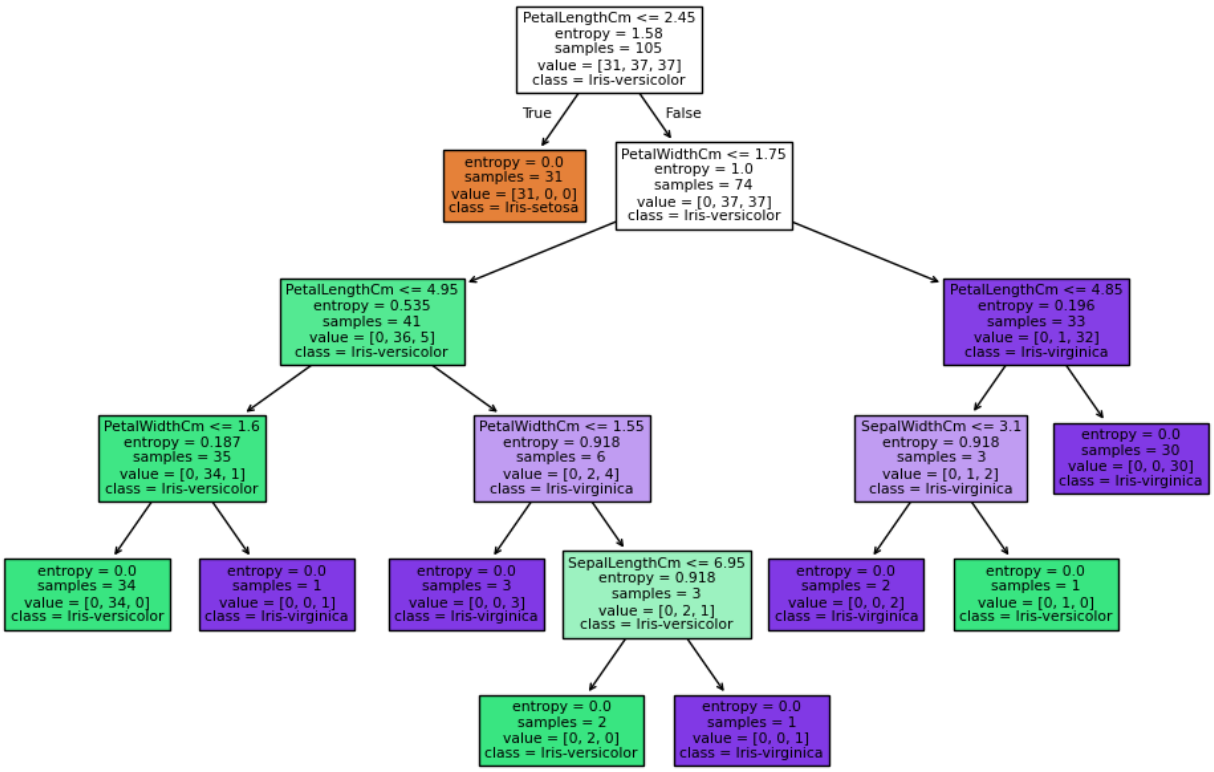
```
plt.figure(figsize=(12,8))
plot_tree(model, feature_names=X.columns, class_names=model.classes_, filled=True)
plt.show()
```

OUTPUT:

```
Accuracy: 100.00%
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        19
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       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
```



LEARNING OUTCOME:

EXPERIMENT 5 (a)

AIM: Random Forest: Implement random forest algorithm on a dataset and evaluate the model's performance.

THEORY:

A Random Forest is a supervised machine learning algorithm used for classification and regression tasks. It is an ensemble learning method that constructs multiple decision trees and combines their outputs to improve predictive performance and reduce overfitting.

Advantages of Random Forest:

- High accuracy due to ensemble of multiple trees.
- Reduces overfitting compared to a single decision tree.
- Can handle large datasets with higher dimensionality.
- Provides feature importance for better interpretability.

Disadvantages of Random Forest:

- Computationally intensive for large datasets.
- Less interpretable than a single decision tree.
- Training time increases with the number of trees.

Algorithm:

1. Create Bootstrap Samples: Random subsets of the dataset are created by sampling with replacement. Each subset is used to train a separate decision tree.
2. Select Random Features at Each Split: At each node of the decision tree, a random subset of features is considered for splitting. This ensures diversity among the trees.
3. Build Decision Trees: Each decision tree is grown fully without pruning using the selected features from step 2.
4. Aggregate Predictions: For classification, the final output is determined by majority voting across all trees. For regression, the final output is the average prediction of all trees.

5. Repeat Until Stopping Condition:

- If the maximum number of trees is reached → Stop.
- If no further improvement in performance → Stop.

ABOUT THE DATASET:

The Iris dataset is a well-known dataset used for classification tasks. It contains 150 samples of iris flowers from three species:

- Setosa
- Versicolor
- Virginica

Each sample has four features:

1. Sepal Length (cm)
2. Sepal Width (cm)
3. Petal Length (cm)
4. Petal Width (cm)

Target Variable:

- Species (Categorical: Setosa, Versicolor, Virginica)

Dataset Source:

The dataset is originally from Fisher's Iris dataset and is widely used for pattern recognition in Machine Learning.

SOURCE CODE:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, accuracy_score, classification_report, confusion_matrix
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
```

```
df = load_iris()
X = df.data
y = df.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state=42, stratify = y)
```

```
model = RandomForestClassifier()
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix\n",cm)
```

```
plt.figure(figsize=(20,10))
plot_tree(model.estimators_[0],
          feature_names = df.feature_names,
          class_names = df.target_names,
          filled = True,
          rounded = True,
          fontsize = 10)
plt.show()
```

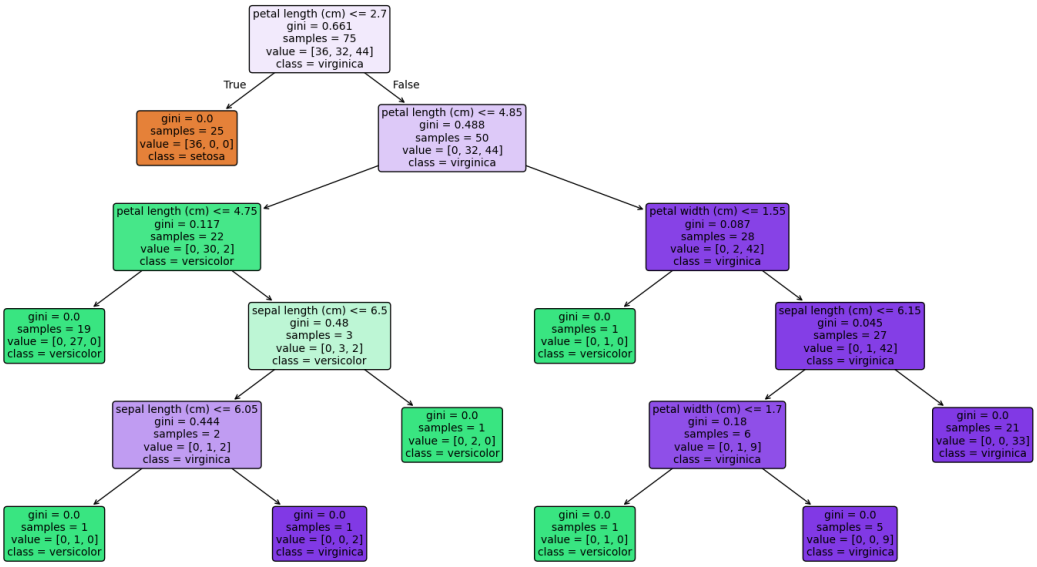
OUTPUT:

```
Accuracy: 94.74%
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.92      0.92      0.92        13
           2       0.92      0.92      0.92        13

    accuracy                           0.95        38
   macro avg       0.95      0.95      0.95        38
weighted avg       0.95      0.95      0.95        38

Confusion matrix
 [[12  0  0]
 [ 0 12  1]
 [ 0  1 12]]
```



LEARNING OUTCOME:

EXPERIMENT 5 (b)

AIM: Random Forest: Implement random forest algorithm on a dataset and evaluate the model's performance.

THEORY:

A Random Forest is a supervised machine learning algorithm used for classification and regression tasks. It is an ensemble learning method that constructs multiple decision trees and combines their outputs to improve predictive performance and reduce overfitting.

Advantages of Random Forest:

- High accuracy due to ensemble of multiple trees.
- Reduces overfitting compared to a single decision tree.
- Can handle large datasets with higher dimensionality.
- Provides feature importance for better interpretability.

Disadvantages of Random Forest:

- Computationally intensive for large datasets.
- Less interpretable than a single decision tree.
- Training time increases with the number of trees.

Algorithm:

1. Create Bootstrap Samples: Random subsets of the dataset are created by sampling with replacement. Each subset is used to train a separate decision tree.
2. Select Random Features at Each Split: At each node of the decision tree, a random subset of features is considered for splitting. This ensures diversity among the trees.
3. Build Decision Trees: Each decision tree is grown fully without pruning using the selected features from step 2.
4. Aggregate Predictions: For classification, the final output is determined by majority voting across all trees. For regression, the final output is the average prediction of all trees.
5. Repeat Until Stopping Condition:

- If the maximum number of trees is reached → Stop.
- If no further improvement in performance → Stop.

ABOUT THE DATASET:

Source: sklearn.datasets.fetch_california_housing (based on 1990 California census)

Purpose: Predict median house value in California districts using demographic and geographical features.

Number of Samples: 20,640

Number of Features: 8 (all numerical)

Features:

- MedInc – Median income (in $10k)
- HouseAge – Median house age (years)
- AveRooms – Average rooms per household
- AveBedrms – Average bedrooms per household
- Population – Population of the block
- AveOccup – Average household size
- Latitude – Latitude of the block
- Longitude – Longitude of the block

Target:

- MedHouseVal – Median house value (in $100k)

Type: Regression dataset, no missing values, all numerical.

Use: Evaluate regression models like Linear Regression, Decision Trees, and Random Forest.

SOURCE CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.datasets import fetch_california_housing
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

✓ 8.6s

```
housing = fetch_california_housing()
X = pd.DataFrame(housing.data, columns=housing.feature_names)
y = pd.Series(housing.target)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

✓ 0.0s

```
rf_regressor = RandomForestRegressor(n_estimators = 100, random_state = 42)
rf_regressor.fit(X_train, y_train)
y_pred = rf_regressor.predict(X_test)
```

✓ 10.7s

```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("\nModel Evaluation:")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
```

✓ 0.0s

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.xlabel("Actual Housing Prices ($100,000s)")
plt.ylabel("Predicted Housing Prices ($100,000s)")
plt.title("Actual vs. Predicted Housing Prices")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.show()
```

✓ 0.1s

OUTPUT:

```
Model Evaluation:
Mean Squared Error (MSE): 0.2554
Root Mean Squared Error (RMSE): 0.5053
```

LEARNING OUTCOME:

06117702722