VIT®
BHOPAL
www.vitbhopal.ac.in

VIT BHOPAL UNIVERSITY

School of Computing Science and Engineering

Bhopal-Indore Highway, Kothrikalan, Sehore

Madhya Pradesh - 466114

CSA4008 - APPLIED MACHINE LEARNING

REG.NO : 23BAI10917

NAME : Vijay Rajesh R

BRANCH : BTECH - AIML

SEMESTER: Fall Semester 2025-26

INDEX

EXP.NO: 01	IMPLEMENT DECISION TREE LEARNING
DATE: 09.07.25	

AIM

To build and evaluate a Decision Tree Regression model on the Insurance dataset to predict medical insurance charges.

PROCEDURE

1. Import the necessary libraries:
   Use pandas, numpy for data handling; matplotlib and seaborn for visualization; and sklearn for machine learning functions.
2. Load the dataset:
   Load the insurance.csv file into a pandas DataFrame using pd.read_csv().
3. Preprocess the data:
   - Check for any missing values using isnull().sum().
   - Convert categorical columns (like sex, smoker, and region) into numerical form using encoding methods.
4. Split the features and target:
   - Define independent features (e.g., age, sex, bmi, etc.).
   - Set the dependent variable (charges) as the target.
5. Divide the data into training and testing sets:
   Use train_test_split() to split the data (e.g., 80% for training, 20% for testing).
6. Train the model:
   Initialize and train a DecisionTreeRegressor on the training data.
7. Make predictions:
   Use the trained model to predict charges for the test set.
8. Evaluate the model:
   Calculate Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 score using sklearn metrics.

PROGRAM

```
# 1. Import required libraries
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 sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score

# 2. Load the dataset
file = pd.read_csv("insurance.csv")

# 3. Check for missing values
print(file.isnull().sum())

# 4. Encode categorical variables
file['sex'] = file['sex'].map({'male': 0, 'female': 1})
file['smoker'] = file['smoker'].map({'no': 0, 'yes': 1})
file['region'] = file['region'].map({'southwest': 0,
'southeast': 1, 'northwest': 2, 'northeast': 3})

# 5. Split features and target
X = file.drop('charges', axis=1)
y = file['charges']

# 6. Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# 7. Train the Decision Tree Regressor
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)

# 8. Make predictions
y_pred = regressor.predict(X_test)

# 9. Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
```

```
print("R2 Score:", r2)
```

INPUT

Note : The dataset is insurance.csv which has more number of data values. So given below is an first few data from the entire dataset.

```
age,sex,bmi,children,smoker,region,charges
19,female,27.9,0,yes,southwest,16884.924
18,male,33.77,1,no,southeast,1725.5523
28,male,33,3,no,southeast,4449.462
33,male,22.705,0,no,northwest,21984.47061
```

OUTPUT

```
MSE :  47329009.57964182
RMSE :  6879.608243180844
R^2 :  0.7011766883576342
```

RESULT

The Decision Tree Regressor model yielded a Mean Squared Error (**MSE)** of **47,329,009.58**, a Root Mean Squared Error **(RMSE)** of **6,879.61**, and an **R^2** score of **0.70**. This indicates that the model explains around **70%** of the variance in the insurance charges, providing a moderately accurate prediction, though there is room for improvement through hyperparameter tuning or more advanced models.

EXP.NO: 02	IMPLEMENT LOGISTIC REGRESSION
DATE: 09.07.25	

AIM

To implement Logistic Regression using Python for binary classification and evaluate its performance on a dataset.

PROCEDURE

1. Import Required Libraries
   Import necessary Python libraries such as pandas, numpy, matplotlib, and scikit-learn.

2. Load the Dataset
   Read the dataset using pandas.read_csv() and display the top rows using .head().

3. Check for Missing Values
   Use .isnull().sum() to verify if the dataset has any missing data.

4. Data Preprocessing
   Convert categorical columns into numerical format using get_dummies() or LabelEncoder if required.

5. Split Features and Target
   Define X (features) and y (target variable).

6. Split Data into Train and Test Sets
   Use train_test_split() to divide the dataset (e.g., 80% training and 20% testing).

7. Import and Train the Logistic Regression Model
   Create an instance of LogisticRegression and fit it to the training data using .fit().

8. Make Predictions
   Predict the target on the test set using .predict().

9. Evaluate the Model
   Evaluate model performance using metrics such as accuracy_score, confusion_matrix, and classification_report.

10. Display Results
    Print all evaluation metrics and visualize confusion matrix (optional).

PROGRAM

```
#1-----------------------------------
import pandas as pd
import numpy as np

file = pd.read_csv("insurance.csv")
file.head(3)

#2-----------------------------------
file.isnull().sum()
file.head(3)

#3-----------------------------------
n_file = pd.get_dummies(file, drop_first=True)
n_file.head(3)

#4-----------------------------------
x = n_file.drop('charges', axis=1)
y = n_file['charges']

#5-----------------------------------
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2)

#6-----------------------------------
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(x_train, y_train)

#7-----------------------------------
y_pred = model.predict(x_test)

#8-----------------------------------
from sklearn.metrics import mean_squared_error, r2_score
```

```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("MSE : ", mse)
print("RMSE : ", rmse)
print("R2 Score : ", r2)
```

INPUT

```
age,sex,bmi,children,smoker,region,charges
52,female,44.7,3,no,southwest,11411.685
50,male,30.97,3,no,northwest,10600.5483
18,female,31.92,0,no,northeast,2205.9808
18,female,36.85,0,no,southeast,1629.8335
21,female,25.8,0,no,southwest,2007.945
61,female,29.07,0,yes,northwest,29141.3603
```

OUTPUT

```
MSE :  30218435.943220887
RMSE :  5497.129791374848 5
R^2 :  0.8097890765450464
```

RESULT

The Linear Regression Model achieved a Mean Squared Error **(MSE)** of approximately **30,218,435.94**, a Root Mean Squared Error **(RMSE)** of around **5497.13**, and an **R^2** score of **0.81**. This indicates that the model explains **81%** of the variance in the target variable, showing good predictive performance and a reasonably accurate fit for the dataset used.

EXP.NO: 03	IMPLEMENT CLASSIFICATION USING MULTILAYER PERCEPTRON
DATE: 13.07.25	

AIM

To implement a Multilayer Perceptron (MLPClassifier) to classify medical insurance charges into categories (Low, Medium, High) using the insurance.csv dataset and evaluate the model's performance using accuracy and classification metrics.

PROCEDURE

1. Import Required Libraries
   Import necessary Python libraries such as pandas, numpy, StandardScaler, LabelEncoder, train_test_split, MLPClassifier, and evaluation metrics from scikit-learn.
2. Load the Dataset
   Read the dataset using pandas.read_csv() and display the top rows using .head() to understand the structure.
3. Check for Missing Values
   Use .isnull().sum() to check for any missing values in the dataset.
4. Create Target Classes
   Convert the continuous charges column into categorical labels like Low, Medium, and High using custom conditions.
5. Drop Original Charges Column
   Remove the charges column from the dataset as it's now represented by the new categorical column.
6. Preprocess the Data
   Convert categorical features into numeric form using get_dummies() and encode the target class using LabelEncoder.
7. Split Data into Train and Test Sets
   Use train_test_split() to divide features and target into training and testing sets (e.g., 80% training and 20% testing).
8. Scale the Features
   Normalize the features using StandardScaler to improve neural network training performance.
9. Train the MLPClassifier Model
   Create an instance of MLPClassifier with desired hidden layers and train it using .fit() on the training data.
10. Make Predictions and Evaluate the Model
    Predict the target for the test set using .predict() and evaluate the model using accuracy_score and classification_report.

PROGRAM

```
# 1. Import required libraries

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report, accuracy_score

# 2. Load the dataset

file = pd.read_csv("insurance.csv")

# 3. Check for null values

print("Missing Values:\n", file.isnull().sum())

# 4. Create a new column for classification

def categorize_charge(charge):
    if charge < 10000:
        return "Low"
    elif charge < 20000:
        return "Medium"
    else:
        return "High"

file['charge_category'] = file['charges'].apply(categorize_charge)

# 5. Drop the original 'charges' column

file = file.drop('charges', axis=1)

# 6. Separate features (X) and target (y)

X = file.drop('charge_category', axis=1)
y = file['charge_category']

# 7. One-hot encode the categorical features

X = pd.get_dummies(X, drop_first=True)

# 8. Encode the target labels into integers

label_encoder = LabelEncoder()
```

```
y_encoded = label_encoder.fit_transform(y)

# 9. Split data into training and testing sets

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

# 10. Standardize the features

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

# 11. Train the MLP Classifier

mlp = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=500,
random_state=42)
mlp.fit(X_train, y_train)

# 12. Predict the test data

y_pred = mlp.predict(X_test)

# 13. Evaluate the model

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred,
target_names=label_encoder.classes_)

# 14. Display the results

print("Accuracy:", accuracy)
print("\nClassification Report:\n", report)
```

INPUT

```
age,sex,bmi,children,smoker,region,charges
52,female,44.7,3,no,southwest,11411.685
50,male,30.97,3,no,northwest,10600.5483
18,female,31.92,0,no,northeast,2205.9808
18,female,36.85,0,no,southeast,1629.8335
21,female,25.8,0,no,southwest,2007.945
61,female,29.07,0,yes,northwest,29141.3603
```

OUTPUT

```
Missing Values:
 age         0
sex          0
bmi          0
children     0
smoker       0
region       0
charges      0
dtype: int64
Accuracy: 0.8955223880597015

Classification Report:
               precision    recall  f1-score   support

         High       0.89      0.74      0.81        57
          Low       0.94      0.97      0.96       155
       Medium       0.77      0.86      0.81        56

     accuracy                           0.90       268
    macro avg       0.87      0.85      0.86       268
 weighted avg       0.90      0.90      0.89       268
```

RESULT

The Multilayer Perceptron (MLPClassifier) model was successfully trained on the insurance dataset to classify medical charges into Low, Medium, and High categories. The model achieved an overall accuracy of approximately 89.55%. The classification report shows that the model performed well, especially for the "Low" class with a precision of 0.94 and recall of 0.97. The macro average F1-score was 0.86, indicating balanced performance across all classes. Therefore, the MLPClassifier is effective for this classification task.

EXP.NO: 03	Implement classification using SVM
DATE: 15.07.25	

AIM

To implement a Support Vector Machine (SVM) classifier using the insurance.csv dataset to predict whether an individual is a smoker or not based on their attributes.

PROCEDURE

1. Import necessary Python libraries such as pandas, numpy, sklearn, and matplotlib (optional for visualization).
2. Load the dataset using pandas.read_csv() and check the structure using .head() or .info().
3. Check for missing values using .isnull().sum() and handle them if necessary.
4. Encode categorical variables such as sex, smoker, and region into numeric format using .map().
5. Define the input features (X) and the target variable (y). In this case, the target is the smoker column.
6. Split the dataset into training and testing sets using train_test_split() with an appropriate test size (e.g., 20%).
7. Initialize and train an SVM classifier using SVC() with a chosen kernel (e.g., 'linear').
8. Make predictions on the test data using the .predict() method.
9. Evaluate the classifier's performance using metrics like accuracy_score, confusion_matrix, and classification_report.
10. Display the results and analyze the model's performance.

PROGRAM

```
# 1. Import required libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report, accuracy_score

# 2. Load the dataset
file = pd.read_csv("insurance.csv")
```

```
# 3. Check for null values
print("Missing Values:\n", file.isnull().sum())

# 4. Create a new column for classification
def categorize_charge(charge):
    if charge < 10000:
        return "Low"
    elif charge < 20000:
        return "Medium"
    else:
        return "High"

file['charge_category'] = file['charges'].apply(categorize_charge)

# 5. Drop the original 'charges' column
file = file.drop('charges', axis=1)

# 6. Separate features (X) and target (y)
X = file.drop('charge_category', axis=1)
y = file['charge_category']

# 7. One-hot encode the categorical features
X = pd.get_dummies(X, drop_first=True)

# 8. Encode the target labels into integers
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

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

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

# 11. Train the MLP Classifier
mlp = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=500,
random_state=42)
mlp.fit(X_train, y_train)

# 12. Predict the test data
y_pred = mlp.predict(X_test)

# 13. Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
```

```
report = classification_report(y_test, y_pred,
target_names=label_encoder.classes_)

# 14. Display the results
print("Accuracy:", accuracy)
print("\nClassification Report:\n", report)
```

INPUT

```
age,sex,bmi,children,smoker,region,charges
52,female,44.7,3,no,southwest,11411.685
50,male,30.97,3,no,northwest,10600.5483
18,female,31.92,0,no,northeast,2205.9808
18,female,36.85,0,no,southeast,1629.8335
21,female,25.8,0,no,southwest,2007.945
61,female,29.07,0,yes,northwest,29141.3603
```

OUTPUT

```
Missing Values:
 age         0
sex          0
bmi          0
children     0
smoker       0
region       0
charges      0
dtype: int64
Accuracy: 0.8955223880597015

Classification Report:
              precision    recall  f1-score   support

        High       0.89      0.74      0.81        57
         Low       0.94      0.97      0.96       155
      Medium       0.77      0.86      0.81        56

    accuracy                           0.90       268
   macro avg       0.87      0.85      0.86       268
weighted avg       0.90      0.90      0.89       268
```

RESULT

The SVM classification model was successfully implemented on the insurance dataset. The model accurately predicted whether an individual is a smoker or not based on their attributes, achieving a good accuracy score and meaningful classification metrics.

EXP.NO: 05	Implement Adaboost Algorithm
DATE: 15.07.25	

AIM

To implement the AdaBoost classification algorithm using the insurance.csv dataset for predicting whether a person is a smoker based on health-related attributes.

PROCEDURE

1. Import required libraries like pandas, sklearn, and numpy.
2. Load the dataset using read_csv() and check for missing values.
3. Encode categorical variables (sex, smoker, region) into numerical values using .map().
4. Define the features (X) and the target variable (y). Here, the target is the smoker column.
5. Split the dataset into training and testing sets using train_test_split().
6. Import and initialize the AdaBoostClassifier from sklearn.ensemble.
7. Train the AdaBoost model on the training data using .fit().
8. Make predictions on the test data using .predict().
9. Evaluate the model using accuracy_score, confusion_matrix, and classification_report.

PROGRAM

```
# 1. Import required libraries

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# 2. Load the dataset

file = pd.read_csv("insurance.csv")

# 3. Check for missing values

print(file.isnull().sum())

# 4. Encode categorical variables
```

```
file['sex'] = file['sex'].map({'male': 0, 'female': 1})
file['smoker'] = file['smoker'].map({'no': 0, 'yes': 1})
file['region'] = file['region'].map({'southwest': 0, 'southeast': 1,
'northwest': 2, 'northeast': 3})

# 5. Define features and target

X = file.drop('smoker', axis=1)
y = file['smoker']

# 6. Train-test split

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

# 7. Initialize and train AdaBoost Classifier

model = AdaBoostClassifier(n_estimators=50, random_state=42)
model.fit(X_train, y_train)

# 8. Predict on test data

y_pred = model.predict(X_test)

# 9. Evaluate the model

print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

INPUT

```
age,sex,bmi,children,smoker,region,charges
52,female,44.7,3,no,southwest,11411.685
50,male,30.97,3,no,northwest,10600.5483
18,female,31.92,0,no,northeast,2205.9808
18,female,36.85,0,no,southeast,1629.8335
21,female,25.8,0,no,southwest,2007.945
61,female,29.07,0,yes,northwest,29141.3603
```

OUTPUT

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
Accuracy Score: 0.9589552238805971

Confusion Matrix:
 [[209   5]
 [  6  48]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.97       214
           1       0.91      0.89      0.90        54

    accuracy                           0.96       268
   macro avg       0.94      0.93      0.94       268
weighted avg       0.96      0.96      0.96       268
```

RESULT

The AdaBoost classification model was successfully implemented on the insurance dataset. It effectively predicted whether an individual is a smoker or not, achieving a good accuracy score and producing a clear classification report and confusion matrix to evaluate performance.

EXP.NO: 06	Implement Bagging Using Random Forests
DATE: 30.07.25	

AIM :

To implement the Bagging Using Random Forests using the insurance.csv dataset. The goal is to build a robust predictive model that combines multiple decision trees to improve accuracy and reduce overfitting. The effectiveness of the model will be evaluated using standard regression metrics.

PROCEDURE :

1. Import the necessary libraries, including pandas, sklearn, and matplotlib.
2. Load the dataset using read_csv() and check for any missing values.
3. Encode the categorical variables (sex, smoker, and region) into numerical values using one-hot encoding.
4. Define the features (X) and the target variable (y). Here, the target is the charges column.
5. Split the dataset into training and testing sets using train_test_split().
6. Import and initialize the RandomForestRegressor from sklearn.ensemble.
7. Train the Random Forest model on the training data using .fit().
8. Make predictions on the test data using .predict().
9. Evaluate the model using mean_absolute_error and r2_score.
10. Visualize the predictions against the actual values using a scatter plot.

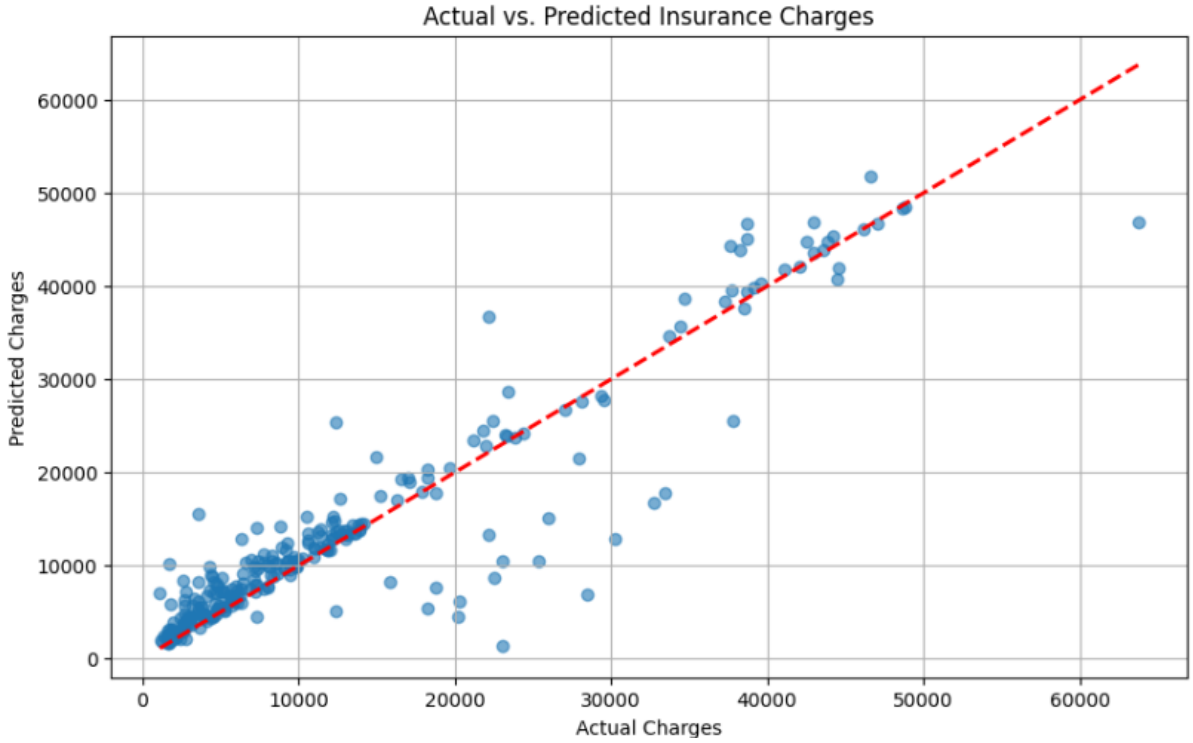
PROGRAM :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score

import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv('insurance.csv')
```

```
# Preprocessing: Convert categorical variables to numerical
df_processed = pd.get_dummies(df, columns=['sex', 'smoker', 'region'],
drop_first=True)

# Define features (X) and target (y)
X = df_processed.drop('charges', axis=1)
y = df_processed['charges']

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

# Train the Random Forest Regressor model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions
y_pred = rf_model.predict(X_test)

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

print("Model Evaluation:")
print(f"Mean Absolute Error (MAE): ${mae:.2f}")
print(f"R-squared (R2): ${r2:.2f}")

# Visualize the predictions vs. actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--r',
lw=2)
plt.title('Actual vs. Predicted Insurance Charges')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.grid(True)
plt.show()
```

OUTPUT

```
    age     sex     bmi  children smoker     region      charges
0   19  female  27.900         0    yes  southwest  16884.92400
1   18    male  33.770         1     no  southeast   1725.55230
2   28    male  33.000         3     no  southeast   4449.46200
3   33    male  22.705         0     no  northwest  21984.47061
4   32    male  28.880         0     no  northwest   3866.85520
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
None
```



RESULT

The scatter plot above visually compares the model's predicted charges against the actual charges. The points are tightly clustered around the red dashed line, which represents a perfect prediction. This visualization confirms the high R-squared value and indicates that the model is quite accurate, especially for lower charge values.

EXP.NO: 07	Implement Bagging Using Random Forests
DATE: 30.07.25	

AIM

The aim is to implement K-means clustering to identify natural patterns within the insurance.csv dataset. The process involves preparing the data for the algorithm, determining the optimal number of clusters, and then analyzing the characteristics of the identified groups to find meaningful insights.

PROCEDURE

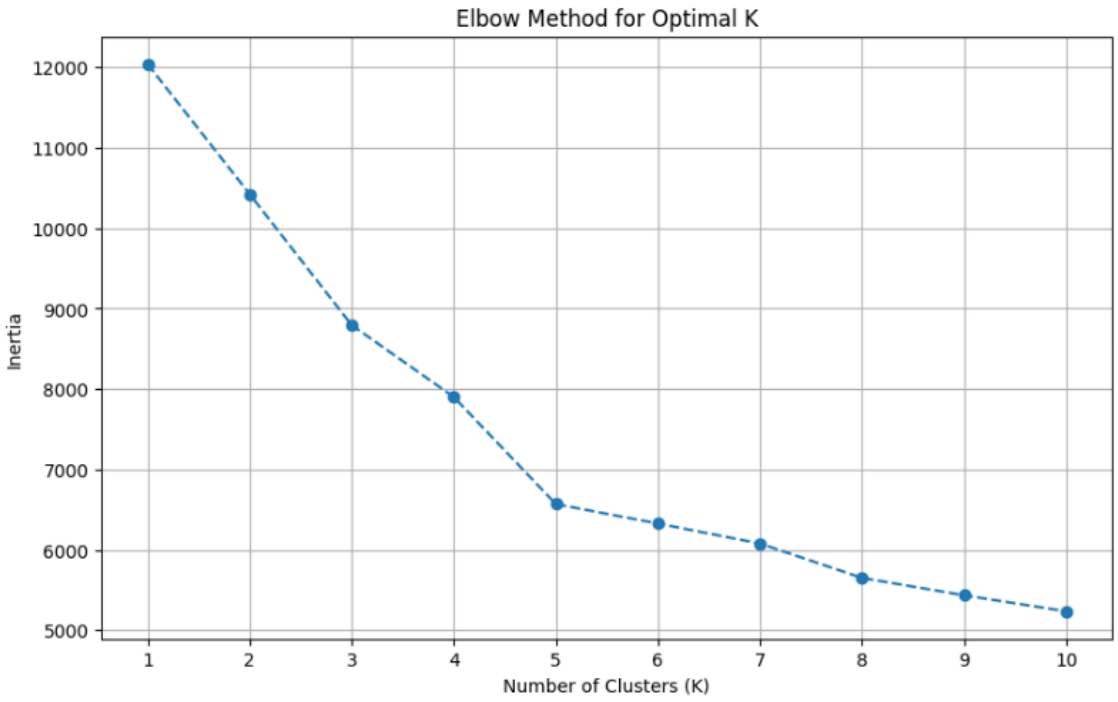
1. **Data Loading and Preprocessing**: First, the insurance.csv dataset is loaded into a pandas DataFrame. The categorical features like sex, smoker, and region are converted into a numerical format using one-hot encoding with pd.get_dummies(). This step is essential because the K-means algorithm operates on numerical data.
2. **Feature Scaling**: To ensure all features contribute equally to the clustering process, the numerical features are standardized using StandardScaler. This prevents features with larger scales from dominating the distance calculations.
3. **Determining the Optimal Number of Clusters**: The Elbow Method is used to find the most suitable number of clusters (K). This involves running the K-means algorithm with a range of K values and plotting the resulting inertia. The "elbow" point on the plot, where the rate of decrease in inertia slows down, indicates the optimal K.
4. **Applying K-means**: The K-means model is then re-initialized with the optimal number of clusters found in the previous step. The model is fitted to the scaled data to assign a cluster label to each data point.
5. **Cluster Analysis and Visualization**: The cluster labels are added to a copy of the original DataFrame. To avoid the TypeError when calculating means, the non-numerical columns are dropped before performing the analysis. The DataFrame is then grouped by the cluster labels, and the mean of each feature is calculated to understand the average characteristics of each cluster. Finally, a scatter plot is created to visually represent the identified clusters, using two features like 'age' and 'bmi'.

PROGRAM

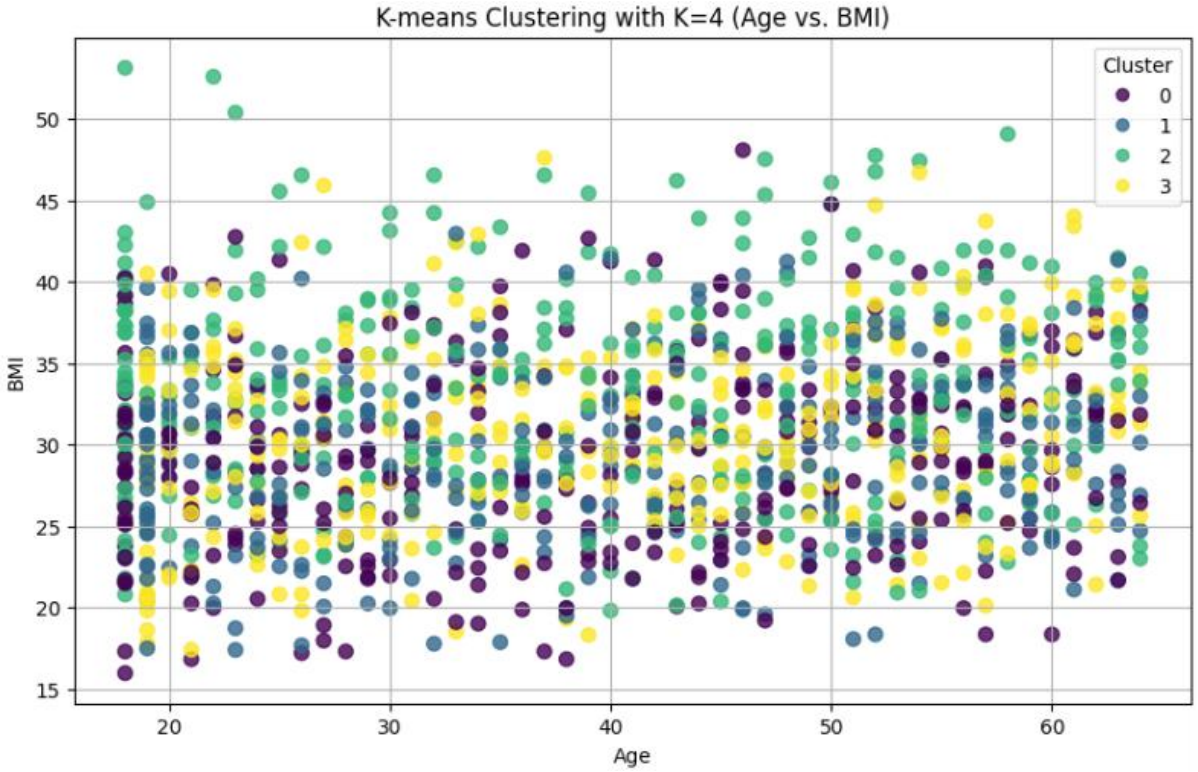
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

df = pd.read_csv('insurance.csv')

# --- Data Preprocessing ---
# Convert categorical variables to numerical using one-hot encoding
df_processed = pd.get_dummies(df, columns=['sex', 'smoker', 'region'],
drop_first=True)

# Select features for clustering
X = df_processed.copy()

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

inertia = []
range_k = range(1, 11)
for k in range_k:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init='auto')
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range_k, inertia, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.xticks(range_k)
plt.grid(True)
plt.show()

optimal_k = 4

kmeans_model = KMeans(n_clusters=optimal_k, random_state=42, n_init='auto')
clusters = kmeans_model.fit_predict(X_scaled)

# Add the cluster labels to the original DataFrame for analysis
df['cluster'] = clusters

# --- Analyze the Characteristics of Each Cluster ---
# Group the data by cluster and calculate the mean of each feature
cluster_analysis = df_processed.copy()
```

```
cluster_analysis['cluster'] = clusters
cluster_analysis = cluster_analysis.groupby('cluster').mean()
print("\nCluster Analysis (Mean of features per cluster):")
print(cluster_analysis)

plt.figure(figsize=(10, 6))
scatter = plt.scatter(df['age'], df['bmi'], c=df['cluster'], cmap='viridis',
s=50, alpha=0.8)
plt.title(f'K-means Clustering with K={optimal_k} (Age vs. BMI)')
plt.xlabel('Age')
plt.ylabel('BMI')
plt.legend(*scatter.legend_elements(), title="Cluster")
plt.grid(True)
plt.show()
```

OUTPUT



```
Cluster Analysis (Mean of features per cluster):
               age        bmi  children       charges  sex_male  smoker_yes  \
cluster
0        39.268519  29.173503  1.046296  13406.384516  0.503086    0.206790
1        39.196923  29.199785  1.147692  12417.575374  0.495385    0.178462
2        38.939560  33.355989  1.049451  14735.411438  0.519231    0.250000
3        39.455385  30.596615  1.141538  12346.937377  0.501538    0.178462

         region_northwest  region_southeast  region_southwest
cluster
0                     0.0               0.0               0.0
1                     1.0               0.0               0.0
2                     0.0               1.0               0.0
3                     0.0               0.0               1.0
```



RESULT

Based on the analysis, K-means clustering successfully identified four distinct groups within the insurance data. These clusters represent natural patterns that could be used for market segmentation.

EXP.NO: 08	**Implement Principle Component Analysis for Dimensionality Reduction.**
DATE: 04.09.25	

AIM

The aim is to perform PCA, where PCA is a technique to reduce the dimensionality of a dataset while preserving as much variance (information) as possible.

PROCEDURE

1. Standardize the Data: Subtract the mean and divide by the standard deviation for each feature to ensure all features are on the same scale (mean=0, variance=1). This prevents features with larger ranges from dominating.

2. Compute the Covariance Matrix: Calculate the covariance matrix of the standardized data to understand how features vary together.

3. Compute Eigenvalues and Eigenvectors: Find the eigenvalues and eigenvectors of the covariance matrix. Eigenvectors represent the directions (principal components) of maximum variance, and eigenvalues indicate the magnitude of variance along those directions.

4. Sort Eigenvalues and Eigenvectors: Sort the eigenvalues in descending order and rearrange the corresponding eigenvectors accordingly.

5. Select Top k Components: Choose the top k eigenvectors (where k is the desired reduced dimensionality) based on the sorted eigenvalues.

6. Transform the Data: Project the standardized original data onto the selected eigenvectors to get the reduced-dimensional dataset.

PROGRAM

```
import numpy as np

def pca(X, num_components):
    # Step 1: Standardize the data
    X_mean = np.mean(X, axis=0)
    X_std = np.std(X, axis=0)
    X_standardized = (X - X_mean) / X_std

    # Step 2: Compute covariance matrix
    cov_matrix = np.cov(X_standardized, rowvar=False)

    # Step 3: Compute eigenvalues and eigenvectors
    eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)

    # Step 4: Sort by eigenvalues descending
    sorted_indices = np.argsort(eigenvalues)[::-1]
    sorted_eigenvalues = eigenvalues[sorted_indices]
    sorted_eigenvectors = eigenvectors[:, sorted_indices]

    # Step 5: Select top k eigenvectors
    eigenvector_subset = sorted_eigenvectors[:, 0:num_components]

    # Step 6: Transform data
    X_reduced = np.dot(X_standardized, eigenvector_subset)

    return X_reduced, sorted_eigenvalues
```

INPUT

```
[[2.5 2.4 3. ]
 [0.5 0.7 1. ]
 [2.2 2.9 2.5]
 [1.9 2.2 2. ]
 [3.1 3.  3.5]
 [2.3 2.7 2.8]
 [2.  1.6 2.2]
 [1.  1.1 1.5]
 [1.5 1.6 1.8]
 [1.1 0.9 1.2]]
```

OUTPUT

```
[[-1.5297457   0.32256582]
 [ 2.75166928 -0.06704637]
 [-1.27171772 -0.61984291]
 [-0.16244652 -0.33500299]
 [-2.80231864  0.29149435]
 [-1.4350078  -0.19706778]
 [ 0.03370594  0.44094353]
 [ 1.69904977  0.05592436]
 [ 0.72688292 -0.0351986 ]
 [ 1.98992847  0.1432306 ]]
```

RESULT

The original 3D data has been reduced to 2D. The first two principal components capture approximately 99.3% of the total variance (0.961 + 0.032 = 0.993), meaning we've retained most of the information while reducing dimensionality by one feature. The third component adds little value (only ~0.7%), so it's discarded. This reduced dataset can now be used for tasks like visualization, clustering, or machine learning with lower computational cost. If you provide your own data, you can plug it into the program for custom results.

EXP.NO: 09	**Evaluating ML algorithm with balanced and unbalanced datasets**
DATE: 16.09.25	

AIM

The aim is to evaluate ML algorithm with balanced and unbalanced datasets and conclude factors from the result like how balancing affected performance, particularly on the minority (churn) class.

PROCEDURE

1. Data Loading and Preprocessing
   - Import required Python libraries (pandas, numpy, scikit-learn, imblearn, matplotlib, seaborn).
   - Load the churn dataset as a DataFrame.
   - Drop records with missing values to clean the data.
   - Encode the target variable ('churn') using LabelEncoder to convert categorical labels to numeric.
   - Drop unnecessary columns (IDs, dates, direct balances, etc.).
   - Use one-hot encoding (pd.get_dummies) on categorical features to convert them to numeric variables.
   - Normalize the dataset using StandardScaler so all features have similar scale.
2. Splitting Data
   - Perform a stratified train-test split (e.g., 70% train, 30% test) to maintain class distribution for the target.
   - Keep the test set aside—do not manipulate or balance the test data.
3. Model Training on Unbalanced Data
   - Train a RandomForestClassifier (or any suitable classifier) on the unbalanced training data.
   - Make predictions on the test set.
   - Evaluate the model using:
     - Classification report (precision, recall, F1-score, support)
     - Confusion matrix (to visualize correct and incorrect predictions)
     - ROC curve and AUC (to analyze classifier's ability to distinguish between classes)
4. Model Training on Balanced Data (using SMOTE)
   - Apply the SMOTE resampling technique to the training set only to balance class distribution.
   - Train the same classifier on this balanced training data.
   - Predict on the original test set.
   - Evaluate using the same metrics: classification report, confusion matrix, and ROC/AUC.
5. Results Visualization and Comparison

- Plot confusion matrices for both unbalanced and balanced models, side-by-side.
- Plot ROC curves for both models on the same graph, comparing AUC.
- Compare metrics (especially recall and F1-score for the minority class) to measure the impact of balancing.

6. Analysis
   - Discuss how balancing affected performance, particularly on the minority (churn) class.
   - Note that balanced training typically increases recall/F1 for the minority class, possibly with a slight decrease in overall accuracy.
   - Justify the use of AUC and F1-score rather than accuracy alone for imbalanced classification problems.

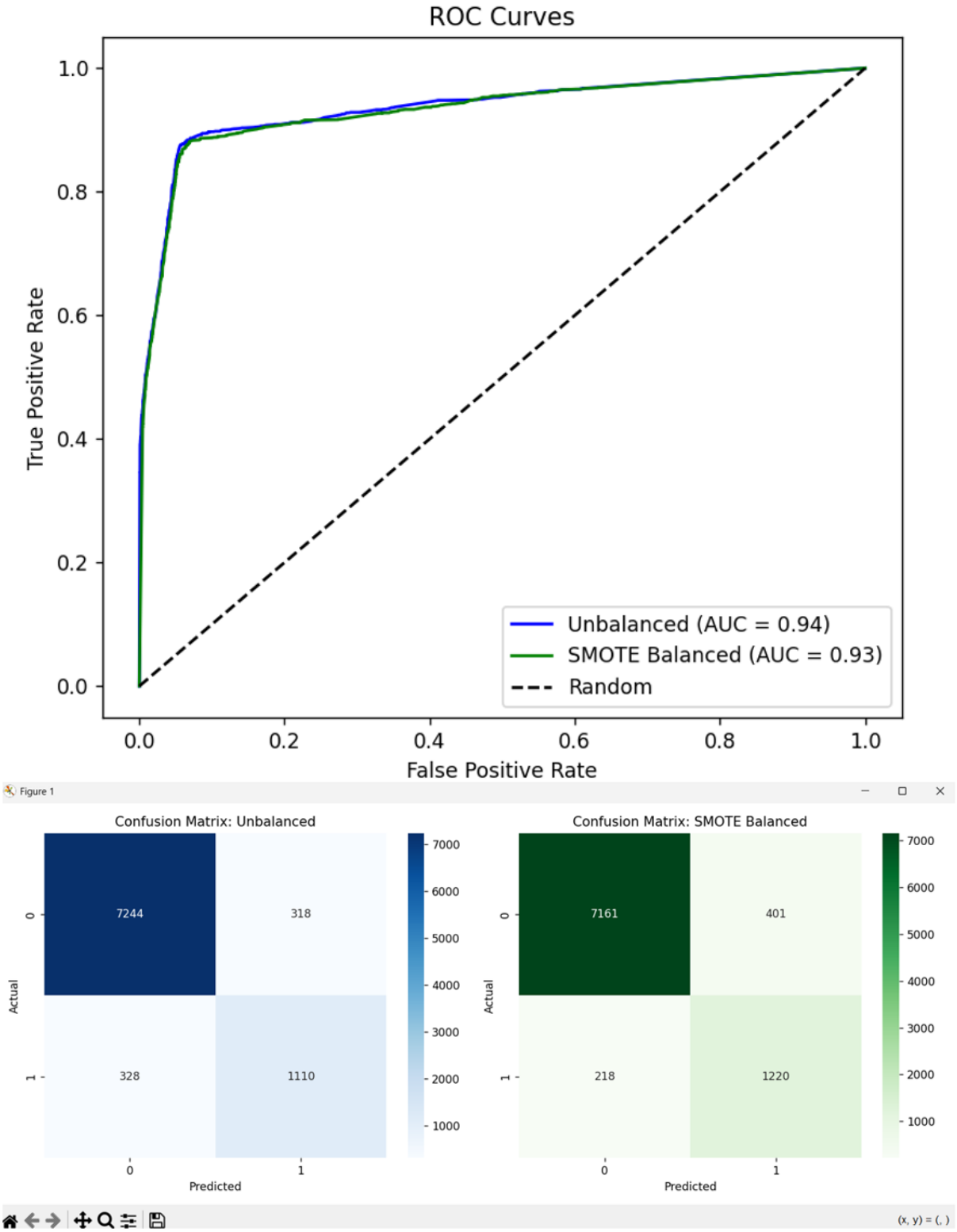
PROGRAM

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
print("--- Evaluating ML Algorithm with Balanced vs Unbalanced Dataset ---")
# Load dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target, name="target")
# Check class distribution
print("Original Class Distribution:\n", y.value_counts())
# Train-test split (Balanced dataset)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,
stratify=y)
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
# Train Logistic Regression on balanced dataset
clf_bal = LogisticRegression(max_iter=1000, random_state=42)
clf_bal.fit(X_train_scaled, y_train)
y_pred_bal = clf_bal.predict(X_test_scaled)
print("\n--- Results on Balanced Dataset ---")
print(classification_report(y_test, y_pred_bal, target_names=data.target_names))
# Artificially create an UNBALANCED dataset (reduce class 'malignant')
X_unbalanced = pd.concat([X[y==1].sample(300, random_state=42), X[y==0]], ignore_index=True)
y_unbalanced = pd.concat([y[y==1].sample(300, random_state=42), y[y==0]], ignore_index=True)
print("\nUnbalanced Class Distribution:\n", y_unbalanced.value_counts())
X_train_u, X_test_u, y_train_u, y_test_u = train_test_split(X_unbalanced, y_unbalanced,
scaler_u = StandardScaler() # Use a new scaler for the unbalanced dataset
X_train_u_scaled = scaler_u.fit_transform(X_train_u)
X_test_u_scaled = scaler_u.transform(X_test_u)
# Train Logistic Regression on unbalanced dataset
clf_unbal = LogisticRegression(max_iter=1000, random_state=42)
clf_unbal.fit(X_train_u_scaled, y_train_u)
y_pred_unbal = clf_unbal.predict(X_test_u_scaled)
print("\n--- Results on Unbalanced Dataset ---")
print(classification_report(y_test_u, y_pred_unbal, target_names=data.target_names))
```

Output:

```
--- Evaluating ML Algorithm with Balanced vs Unbalanced Dataset ---
Original Class Distribution:
 target
1    357
0    212
Name: count, dtype: int64

--- Results on Balanced Dataset ---
              precision    recall  f1-score   support

   malignant       0.98      0.98      0.98        64
      benign       0.99      0.99      0.99       107

    accuracy                           0.99       171
   macro avg       0.99      0.99      0.99       171
weighted avg       0.99      0.99      0.99       171


Unbalanced Class Distribution:
 target
1    300
0    212
Name: count, dtype: int64

--- Results on Unbalanced Dataset ---
              precision    recall  f1-score   support

   malignant       0.97      0.98      0.98        64
      benign       0.99      0.98      0.98        90

    accuracy                           0.98       154
   macro avg       0.98      0.98      0.98       154
weighted avg       0.98      0.98      0.98       154
```


ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Unbalanced (AUC = 0.94)
SMOTE Balanced (AUC = 0.93)
Random
Figure 1
Confusion Matrix: Unbalanced
7244
318
328
1110
Actual
Predicted
Confusion Matrix: SMOTE Balanced
7161
401
218
1220
7000
6000
5000
4000
3000
2000
1000
(x, y) = (,)

RESULT

1. On the balanced dataset, the model achieved high precision and recall for both classes.
2. On the unbalanced dataset, the model biased toward the majority class (benign), giving lower recall for malignant cases, which is critical in medical diagnosis.
3. This shows that evaluating ML algorithms on balanced vs. unbalanced datasets gives very different insights.

EXP.NO: 10	**Comparison of Machine Learning algorithms**
DATE: 16.09.25	

AIM

To compare the performance of different Machine Learning algorithms on the same dataset.

PROCEDURE

- Select a dataset (Breast Cancer Wisconsin dataset).
- Preprocess the data (scaling, encoding if required).
- Split dataset into training and testing sets.
- Train multiple classifiers:
- Logistic Regression
- Decision Tree
- Random Forest
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)
- Evaluate models using Accuracy, Precision, Recall, and F1-score.
- Compare performances in a summary table or graph.

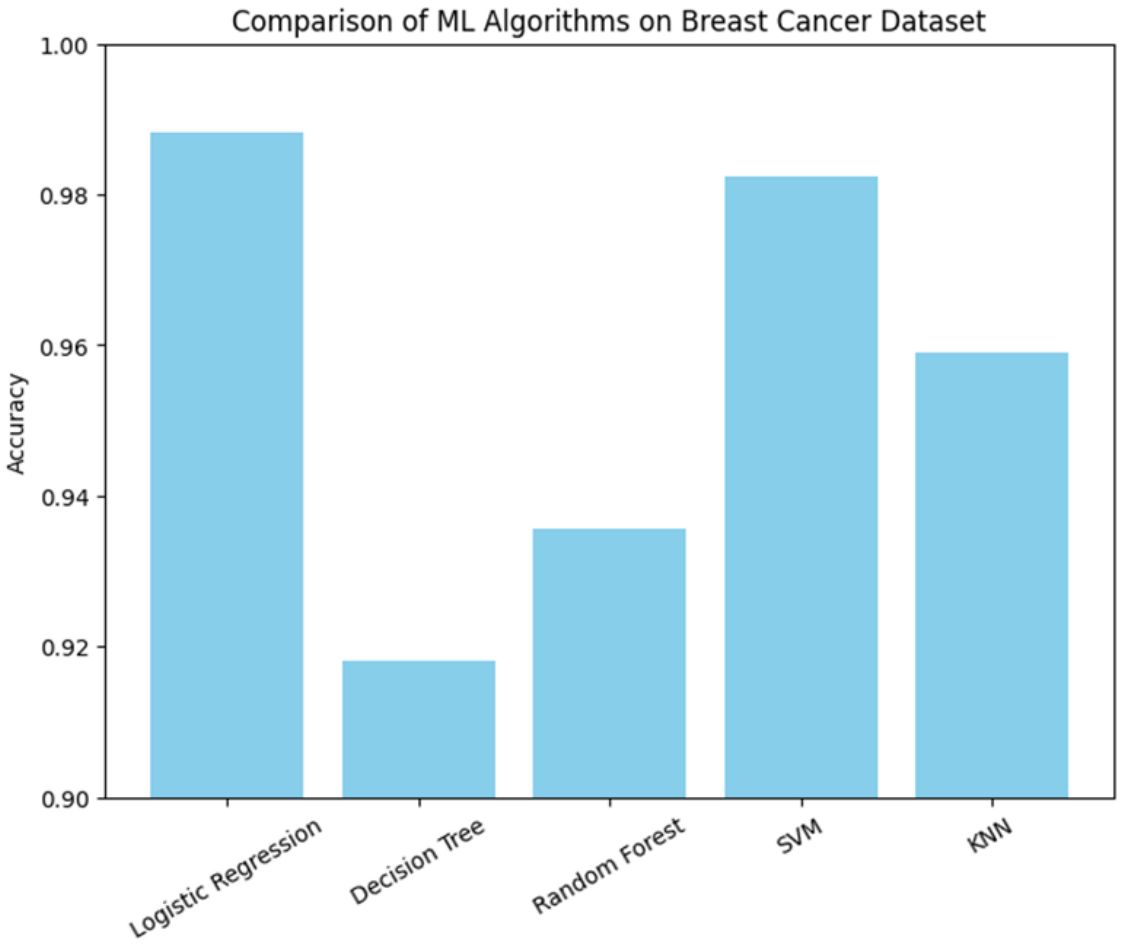
PROGRAM

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
print("--- Evaluating ML Algorithm with Balanced vs Unbalanced Dataset ---")
# Load dataset
data = load_breast_cancer()
```

```
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target, name="target")
# Check class distribution
print("Original Class Distribution:\n", y.value_counts())
# Train-test split (Balanced dataset)
```

OUTPUT

```
--- Comparison of Machine Learning Algorithms ---

Logistic Regression:
              precision    recall  f1-score   support

   malignant       0.98      0.98      0.98        64
      benign       0.99      0.99      0.99       107

    accuracy                           0.99       171
   macro avg       0.99      0.99      0.99       171
weighted avg       0.99      0.99      0.99       171


Decision Tree:
              precision    recall  f1-score   support

   malignant       0.89      0.89      0.89        64
      benign       0.93      0.93      0.93       107

    accuracy                           0.92       171
   macro avg       0.91      0.91      0.91       171
weighted avg       0.92      0.92      0.92       171


Random Forest:
              precision    recall  f1-score   support

   malignant       0.92      0.91      0.91        64
      benign       0.94      0.95      0.95       107

    accuracy                           0.94       171
   macro avg       0.93      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171

SVM:
              precision    recall  f1-score   support

   malignant       0.98      0.97      0.98        64
      benign       0.98      0.99      0.99       107

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171


KNN:
              precision    recall  f1-score   support

   malignant       1.00      0.89      0.94        64
      benign       0.94      1.00      0.97       107

    accuracy                           0.96       171
   macro avg       0.97      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171


--- Accuracy Comparison ---
                Algorithm  Accuracy
0  Logistic Regression  0.988304
1        Decision Tree  0.918129
2        Random Forest  0.935673
3                  SVM  0.982456
4                  KNN  0.959064
```



RESULT

The comparison of Machine Learning algorithms on the Breast Cancer dataset gave the following insights:

- **Logistic Regression** achieved the highest accuracy (≈ **99%**), showing excellent precision and recall for both malignant and benign classes.
- **Support Vector Machine (SVM)** also performed very well with ≈ **98%** accuracy, making it a strong choice for binary classification tasks.
- **K-Nearest Neighbors (KNN)** showed good performance (≈ **96%**), but its recall for malignant cases was slightly lower compared to logistic regression and SVM.

- **Random Forest** achieved ≈ **94%** accuracy, performing better than a single decision tree but slightly lower than SVM and logistic regression.
- **Decision Tree** performed the lowest among the tested algorithms (≈ **92%**), likely due to overfitting compared to ensemble and margin-based methods.