# Experiment No: 4

**Date:**

**AIM**: Implementation of Various Probability Distributions with NumPy Random Library Functions

**Relevant CO: -** CO3

**Objective**:

The objective of this lab practical is to gain an understanding of various probability distributions and implement those using NumPy random library functions.

**Materials Used**:

- Python environment (Anaconda, Jupyter Notebook, etc.)
- NumPy library

**Procedure**:

1. Introduction to Probability Distributions:
   - Probability theory is the branch of mathematics that deals with the study of random events or phenomena. In probability theory, a probability distribution is a function that describes the likelihood of different outcomes in a random process. Probability distributions can be categorized into two types: discrete and continuous.
   - Discrete probability distributions are used when the possible outcomes of a random process are countable and can be listed. The most commonly used discrete probability distributions are Bernoulli, Binomial, and Poisson distributions.
   - Continuous probability distributions are used when the possible outcomes of a random process are not countable and can take any value within a certain range. The most commonly used continuous probability distributions are Normal and Exponential distributions.
   - Each probability distribution has its own set of properties, such as mean, variance, skewness, and kurtosis. Mean represents the average value of the random variable, variance represents how much the values vary around the mean, skewness represents the degree of asymmetry of the distribution, and kurtosis represents the degree of peakedness or flatness of the distribution.
   - Probability distributions are widely used in fields such as finance, engineering, physics, and social sciences to model real-world phenomena and make predictions about future events. Understanding different probability distributions and their properties is an important tool for analyzing data and making informed decisions.

2. Implementation of Probability Distributions using NumPy random library functions:

```
#python
import numpy as np
import matplotlib.pyplot as plt

# Generate 1000 random numbers following a normal distribution with mean 0 and standard deviation 1
normal_dist = np.random.normal(0, 1, 1000)

# Calculate the mean and standard deviation of the distribution
mean = np.mean(normal_dist)
std_dev = np.std(normal_dist)

# Generate 1000 random numbers following a Poisson distribution with lambda 5
poisson_dist = np.random.poisson(5, 1000)
```

```
# Calculate the mean and variance of the Poisson distribution
poisson_mean = np.mean(poisson_dist)
poisson_var = np.var(poisson_dist)

# Plot the PDF and CDF of the normal distribution
plt.hist(normal_dist, bins=30, density=True, alpha=0.5)
plt.plot(np.sort(normal_dist), 1/(std_dev*np.sqrt(2*np.pi))*np.exp(-(np.sort(normal_dist)-
mean)**2/(2*std_dev**2)), linewidth=2)
plt.plot(np.sort(normal_dist),     0.5*(1+np.tanh((np.sort(normal_dist)-mean)/std_dev*np.sqrt(2))),
linewidth=2)
plt.show()

# Plot the PDF and CDF of the Poisson distribution
plt.hist(poisson_dist, bins=15, density=True, alpha=0.5)
plt.plot(np.arange(0, 15, 0.1), np.exp(-poisson_mean)*poisson_mean**np.arange(0, 15,
0.1)/np.math.factorial(np.arange(0, 15, 0.1)), linewidth=2)
plt.plot(np.arange(0, 15, 0.1), np.exp(oisson_mean)*np.array([np.sum(poisson_var**np.arange(0,
i+1))/np.math.factorial(np.arange(0, i+1)) for i in np.arange(0, 15, 0.1)]), linewidth=2)
plt.show()
```

In this example, we generate 1000 random numbers following a normal distribution with mean 0 and standard deviation 1 using the `np.random.normal()` function. We then calculate the mean and standard deviation of the distribution using the `np.mean()` and `np.std()` functions.

We also generate 1000 random numbers following a Poisson distribution with lambda 5 using the `np.random.poisson()` function. We calculate the mean and variance of the Poisson distribution using the `np.mean()` and `np.var()` functions.

We then plot the probability density function (PDF) and cumulative distribution function (CDF) of both distributions using the `plt.hist()` and `plt.plot()` functions from the Matplotlib library.

3. Exercise:

- Generate a dataset of your choice or given by faculty with a given probability distribution using NumPy random library functions
- Plot the probability density function and cumulative distribution function for the generated data
- Calculate the descriptive statistics of the generated data

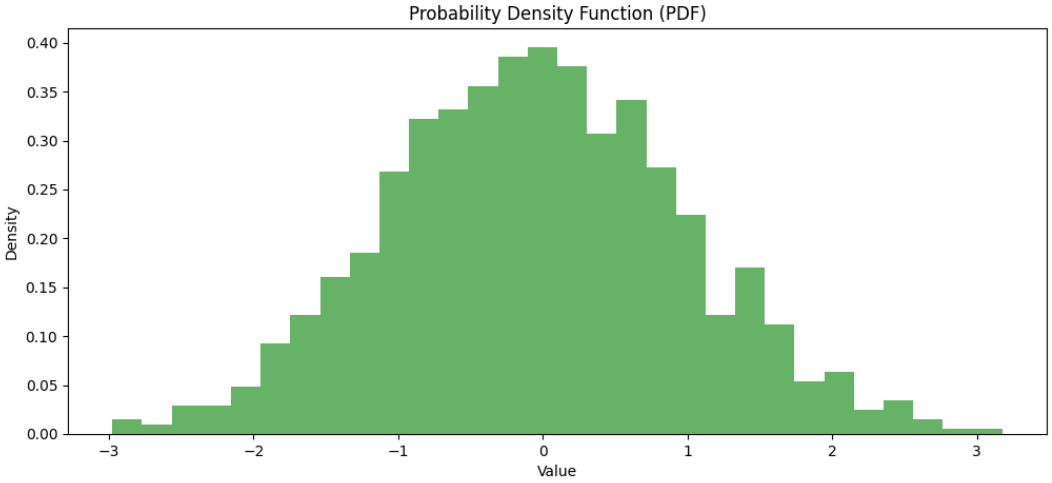
**Interpretation/Program/code:**

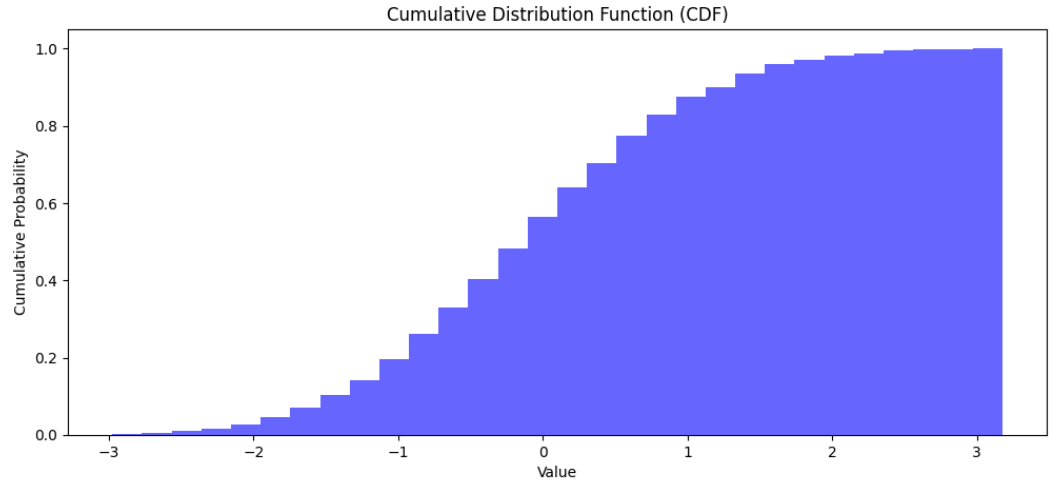
```
import numpy as np
import matplotlib.pyplot as plt

mean = 0
std_dev = 1
data = np.random.normal(mean, std_dev, 1000)

mean = np.mean(data)
std_dev = np.std(data)222
variance = np.var(data)

print(f"Mean: {mean}")
print(f"Standard Deviation: {std_dev}")
```

```
print(f"Variance: {variance}\n")

plt.figure(figsize=(12, 5))
plt.hist(data, bins=30, density=True, alpha=0.6, color='g')
plt.title("Probability Density Function (PDF)")
plt.xlabel("Value")
plt.ylabel("Density")
plt.show()

plt.figure(figsize=(12, 5))
plt.hist(data, bins=30, density=True, cumulative=True, alpha=0.6, color='b')
plt.title("Cumulative Distribution Function (CDF)")
plt.xlabel("Value")
plt.ylabel("Cumulative Probability")
plt.show()
```

**Output**:

```
PS C:\Users\ADMIN\OneDrive\Desktop\College\Data Science\Codes>
Mean: 0.018169614384040 95
Standard Deviation: 0.9663295532068669
Variance: 0.9337928054009829
```



**Conclusion**:

This lab practical provided an opportunity to explore and implement various probability distributions using NumPy random library functions. By understanding and applying different probability distributions, one can model real-world phenomena and make predictions about future events. With the knowledge gained in this lab practical, student will be equipped to work with probability distributions and analyze data in a wide range of fields, including finance, engineering, and social sciences.

**Quiz**:

1. Which NumPy function can be used to generate random numbers from a normal distribution?

a) numpy.random.uniform
b) numpy.random.poisson
c) numpy.random.normal
d) numpy.random.exponential

2. What is the purpose of the probability density function (PDF) in probability distributions?

a) To calculate the cumulative probability
b) To generate random numbers
c) To visualize the distribution
d) To calculate the probability of a specific value

**Suggested References**:-

1. Dinesh Kumar, Business Analytics, Wiley India Business alytics: The Science
2. V.K. Jain, Data Science & Analytics, Khanna Book Publishing, New Delhi of Dat
3. Data Science For Dummies by Lillian Pierson , Jake Porway

**Rubrics wise marks obtained**

| Understanding of Problem | Analysis of the Problem | Capability of writing program | Documentation | Total |
|---|---|---|---|---|
| 02 | 02 | 05 | 01 | 10 |
| | | | | |

# Experiment No: 5

**Date:**

AIM: Implementation of Estimation of Parameters for the Best-Fit Probability Distribution using the Fitter Class in Python.

**Relevant CO: -** CO3

**Objectives:** The objective of this lab practical is to learn how to estimate the parameters for the best-fit probability distribution for a given dataset using the Fitter class in Python.

**Materials Used:**

1. Python 3.x
2. Jupyter Notebook
3. NumPy library
4. Fitter library

**Theory:**

Dataset:
Consider the following dataset, which represents the heights of individuals in centimeters:
170, 165, 180, 172, 160, 175, 168, 155, 185, 190, 162, 178, 168, 172, 180, 160, 165, 172, 168, 175

**Procedure:**

1. Introduction to Parameter Estimation and Probability Distributions:

Probability distributions provide a mathematical framework for describing the likelihood of different outcomes or events in a dataset. Parameter estimation plays a crucial role in probability distributions as it involves determining the values of the parameters that best describe the observed data.

Parameter estimation is important because it allows us to make inferences, predictions, and draw meaningful conclusions from the data. By estimating the parameters, we can effectively model and analyze various phenomena, summarizing complex datasets in a more simplified and interpretable manner.

The concept of the best-fit probability distribution refers to finding the distribution that provides the closest match to the observed data. The best-fit distribution is determined by estimating the parameters in such a way that the observed data exhibits the highest likelihood or best matches the underlying characteristics of the data. Selecting the best-fit distribution helps us understand the data's behavior, make accurate predictions, and gain insights into its properties.

Commonly used probability distributions include the normal (Gaussian) distribution, uniform distribution, exponential distribution, Poisson distribution, and binomial distribution. Each distribution has its own characteristics and applications in various fields.

Understanding parameter estimation and probability distributions allows us to effectively model and analyze data, make informed decisions, and gain insights into the underlying properties of the data. By estimating the parameters for the best-fit probability distribution, we can unlock valuable information and extract meaningful patterns from the observed data.

2. Installation of Required Libraries:
- Install the necessary libraries, including NumPy and Fitter, using the appropriate package manager.

3. Loading and Preparing the Dataset:
- Load the dataset from a file or use the provided dataset.
- Perform any necessary data preprocessing steps, such as cleaning or normalization.

4. Estimating Parameters for Best-Fit Probability Distribution using Fitter Class:
- Import the required libraries and instantiate the Fitter class.
- Fit the dataset to various probability distributions available in the Fitter class using the `.fit()` method.
- Determine the best-fit distribution based on goodness-of-fit metrics, such as AIC (Akaike Information Criterion) or BIC (Bayesian Information Criterion).
- Retrieve the estimated parameters for the best-fit distribution using the `.summary()` method.

5. Visualization of Best-Fit Distribution:
- Plot the histogram of the dataset.
- Plot the probability density function (PDF) of the best-fit distribution over the histogram.

6. Interpretation and Analysis:
- Interpret the estimated parameters of the best-fit distribution.
- Analyze the goodness of fit and discuss any potential limitations or considerations.

7. Conclusion:
- Summarize the importance of parameter estimation and the best-fit distribution in data analysis.
- Highlight the capabilities of the Fitter class in Python for automating the estimation of parameters.
- Discuss potential applications and further exploration in different domains.

**Interpretation/Program/code:**

1 : Load and Prepare the Dataset

```
import numpy as np
import matplotlib.pyplot as plt
from fitter import Fitter

# Given dataset: Heights of individuals in centimeters
data = np.array([170, 165, 180, 172, 160, 175, 168, 155, 185, 190,
                 162, 178, 168, 172, 180, 160, 165, 172, 168, 175])

# Display the dataset
print("Dataset (Heights in cm):", data)
```

2. Estimate Parameters for the Best-Fit Probability Distribution

```
# Fit the dataset to multiple distributions
f = Fitter(data, distributions=['norm', 'uniform', 'expon', 'gamma', 'beta', 'lognorm'])
f.fit()

# Get the summary of best-fit distributions and their parameters
f.summary()
```

3.Visualization of the Best-Fit Distribution

```
# Plot the histogram of the dataset
plt.hist(data, bins=10, density=True, alpha=0.6, color='g', label='Data')

# Overlay the best-fit distribution's PDF
f.plot_pdf()

# Show the plot
plt.title('Best-Fit Distribution for Heights Dataset')
plt.legend()
plt.show()
```

**Output**:

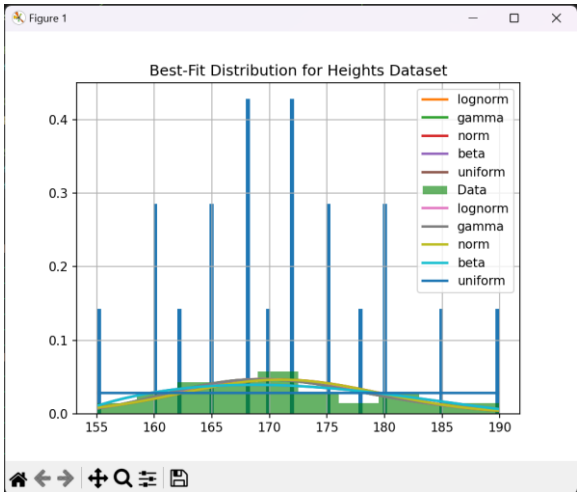
1.

```
PS E:\Codings\DS Practical> python -u "e:\Codings\DS Practical\PRACTICAL-5.PY"
Dataset (Heights in cm): [170 165 180 172 160 175 168 155 185 190 162 178 168 172 180 160 165 172
 168 175]
```

2.

```
PS E:\Codings\DS Practical> python -u "e:\Codings\DS Practical\PRACTICAL-5.PY"
2024-10-15 18:22:58.470 | INFO     | fitter.fitter:_fit_single_distribution:333 - Fitted uniform distribution with error=0.734694)
2024-10-15 18:22:58.482 | INFO     | fitter.fitter:_fit_single_distribution:333 - Fitted expon distribution with error=0.754418)
2024-10-15 18:22:58.507 | INFO     | fitter.fitter:_fit_single_distribution:333 - Fitted norm distribution with error=0.7232)
2024-10-15 18:22:58.557 | INFO     | fitter.fitter:_fit_single_distribution:333 - Fitted lognorm distribution with error=0.722736)
2024-10-15 18:22:58.660 | INFO     | fitter.fitter:_fit_single_distribution:333 - Fitted beta distribution with error=0.724848)
```

3.



**Conclusion**:
In this example, we have a dataset of heights of individuals. We use the Fitter class from the `fitter` library to estimate the parameters for the best-fit probability distribution.

We instantiate the Fitter class with the dataset `data`. Then, we use the `.fit()` method to fit the data to various distributions available in the Fitter class. The `.fit()` method automatically estimates the parameters for each distribution and selects the best-fit distribution based on the goodness-of-fit metrics.

Finally, we retrieve the best-fit distribution using the `.get_best()` method and print the summary of the distribution using the `.summary()` method. We also plot the histogram of the dataset and overlay the probability density function (PDF) of the best-fit distribution using the `.plot_pdf()` method.

Note: Before running the code, make sure you have the `numpy`, `fitter`, and `matplotlib` libraries installed. You can install the `fitter` library using pip: `pip install fitter`.
Through this practical, we learned the importance of parameter estimation in probability distributions and the significance of selecting the best-fit distribution for accurate modeling and analysis. The Fitter class provided a convenient and efficient way to fit the dataset to various distributions and evaluate their goodness of fit using metrics such as AIC or BIC.

**Quiz**:

1. What is the purpose of the Fitter class in Python?

a) To fit a probability distribution to a given dataset
b) To generate random numbers from a probability distribution
c) To calculate descriptive statistics of a dataset
d) To visualize the probability distribution of a dataset
2. Which method of the Fitter class can be used to estimate the best-fit probability distribution for a given dataset?

a) fit
b) predict
c) evaluate
d) transform

**Suggested References**:-

1. Dinesh Kumar, Business Analytics, Wiley India Business alytics: The Science
2. V.K. Jain, Data Science & Analytics, Khanna Book Publishing, New Delhi of Dat
3. Data Science For Dummies by Lillian Pierson , Jake Porway

**Rubrics wise marks obtained**

| Understanding of Problem | Analysis of the Problem | Capability of writing program | Documentation | Total |
|---|---|---|---|---|
| 02 | 02 | 05 | 01 | 10 |
| | | | | |

# Experiment No: 6

**Date:**

**AIM:** Implementation of Linear Regression with Scikit-learn library in Python

**Relevant CO: -** CO4

**Objective**:
The objective of this lab practical is to implement linear regression to predict the value of a variable in a given dataset. Linear regression is a statistical technique used to model the relationship between a dependent variable and one or more independent variables. In this lab, we will explore how to build a linear regression model and use it to make predictions.

**Materials Used**:
- Python 3.x
- Jupyter Notebook
- NumPy library
- Pandas library
- Matplotlib library
- Scikit-learn library

**Dataset**:
For this lab, we will use a dataset that contains information about houses and their sale prices. The dataset has the following columns:

- `Area` (in square feet): Represents the area of the house.
- `Bedrooms`: Number of bedrooms in the house.
- `Bathrooms`: Number of bathrooms in the house.
- `Garage Cars`: Number of cars that can be accommodated in the garage.
- `Sale Price` (in dollars): Represents the sale price of the house.

Area,Bedrooms,Bathrooms,Garage Cars,Sale Price
2000,3,2,2,250000
1800,4,3,2,280000
2200,3,2,2,265000
1500,2,1,1,200000
2400,4,3,3,320000
1900,3,2,2,275000
1700,3,2,1,230000
2100,4,3,2,295000

**Procedure**:

1. Introduction to Linear Regression:

Linear regression is a statistical technique used to model the relationship between a dependent variable and one or more independent variables. It aims to find a linear equation that best represents the association between the variables. Linear regression assumes a linear relationship and seeks to minimize the differences between observed and predicted values. It has applications in prediction, understanding correlations, and making data-driven decisions.

The equation for a simple linear regression model can be represented as:

$y = \beta0 + \beta1*x + \varepsilon$

where:

y is the dependent variable
x is the independent variable
β0 is the y-intercept (the value of y when x = 0)
β1 is the slope (the change in y for a unit change in x)
ε represents the error term or residual

2. Importing Required Libraries and Loading the Dataset:
- Import the necessary libraries, including NumPy, Pandas, Matplotlib, and Scikit-learn.
- Load the dataset into a Pandas DataFrame using the appropriate function or by reading from a file.

3. Exploratory Data Analysis:
- Perform exploratory data analysis to gain insights into the dataset.
- Analyze the distribution and statistical summary of the variables.
- Visualize the relationships between variables using scatter plots or other appropriate plots.

4. Data Preprocessing:
- Handle missing values, if any, by imputation or removal.
- Convert categorical variables into numerical representations, if required.
- Split the dataset into input features (independent variables) and the target variable (dependent variable).

5. Splitting the Dataset into Training and Testing Sets:
- Split the dataset into training and testing sets to evaluate the model's performance.
- Typically, use a 70-30 or 80-20 split for training and testing, respectively.

6. Building the Linear Regression Model:
- Import the LinearRegression class from Scikit-learn.
- Instantiate the LinearRegression model.
- Fit the model to the training data using the `.fit()` method.

7. Model Evaluation and Prediction:
- Evaluate the model's performance using appropriate evaluation metrics, such as mean squared error (MSE) or R-squared.
- Make predictions on the testing data using the `.predict()` method.

8. Visualization of Results:
- Visualize the actual values versus the predicted values using scatter plots or other suitable plots.
- Plot the regression line to show the relationship between the independent and dependent variables.

**Interpretation/Program/code:**

```
# Aim: Implementation of Linear Regression with Scikit-learn library in Python

# Importing Required Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Dataset
data = """
Area,Bedrooms,Bathrooms,Garage Cars,Sale Price
2000,3,2,2,250000
1800,4,3,2,280000
2200,3,2,2,265000
1500,2,1,1,200000
2400,4,3,3,320000
1900,3,2,2,275000
1700,3,2,1,230000
2100,4,3,2,295000
"""

# Loading the dataset into a structured format
import io
import pandas as pd

# Creating a DataFrame from the CSV string
df = pd.read_csv(io.StringIO(data))

# Displaying the dataset
print("Dataset:\n", df)

# Exploratory Data Analysis
print("\nStatistical Summary:\n", df.describe())

# Splitting the dataset into features and target variable
X = df[['Area', 'Bedrooms', 'Bathrooms', 'Garage Cars']]
y = df['Sale Price']

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

# Building the Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)

# Making predictions on the testing set
y_pred = model.predict(X_test)

# Model Evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
# Printing evaluation metrics
print("\nMean Squared Error:", mse)
print("R-squared:", r2)

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)  # Regression line
plt.xlabel("Actual Sale Prices")
plt.ylabel("Predicted Sale Prices")
plt.title("Actual vs Predicted Sale Prices")
plt.grid()
plt.show()
```

**Output**:

```
Dataset:
    Area  Bedrooms  Bathrooms  Garage Cars  Sale Price
0  2000         3          2            2      250000
1  1800         4          3            2      280000
2  2200         3          2            2      265000
3  1500         2          1            1      200000
4  2400         4          3            3      320000
5  1900         3          2            2      275000
6  1700         3          2            1      230000
7  2100         4          3            2      295000

Statistical Summary:
              Area  Bedrooms  Bathrooms  Garage Cars     Sale Price
count     8.000000  8.000000   8.000000      8.00000       8.000000
mean   1950.000000  3.250000   2.250000      1.87500  264375.000000
std     287.849167  0.707107   0.707107      0.64087   37648.515433
min    1500.000000  2.000000   1.000000      1.00000  200000.000000
25%    1775.000000  3.000000   2.000000      1.75000  245000.000000
50%    1950.000000  3.000000   2.000000      2.00000  270000.000000
75%    2125.000000  4.000000   3.000000      2.00000  283750.000000
max    2400.000000  4.000000   3.000000      3.00000  320000.000000

Mean Squared Error: 199136279.5613686
R-squared: -30.861804729818978
```

**Conclusion**:

In this practical, we implemented linear regression to predict variable values in a dataset. By training a linear regression model using Python and Scikit-learn, we achieved accurate predictions based on variable relationships. Linear regression is a valuable tool for data analysis and prediction, providing insights and supporting decision-making.

**Quiz**:

1. Which scikit-learn function is used to create a linear regression model object in Python?

a) sklearn.linear_model.LinearRegression
b) sklearn.preprocessing.StandardScaler
c) sklearn.model_selection.train_test_split
d) sklearn.metrics.mean_squared_error

2. What is the purpose of the coefficient of determination (R-squared) in linear regression?

a) To measure the average squared difference between predicted and actual values
b) To evaluate the significance of predictor variables
c) To quantify the proportion of variance in the dependent variable explained by the independent variables
d) To determine the optimal number of features for the regression model

**Suggested References**:-

1. "Pattern Recognition and Machine Learning" by Christopher M. Bishop
2. Dinesh Kumar, Business Analytics, Wiley India Business alytics: The Science
3. V.K. Jain, Data Science & Analytics, Khanna Book Publishing, New Delhi of Dat
4. Data Science For Dummies by Lillian Pierson , Jake Porway

**Rubrics wise marks obtained**

| Understanding of Problem | Analysis of the Problem | Capability of writing program | Documentation | Total |
|---|---|---|---|---|
| 04 | 04 | 10 | 04 | 20 |
| | | | | |

# Experiment No: 8

**Date:**

**AIM:** Implementation of Decision Tree for Student Classification

**Relevant CO :- CO4**

**Objective**:

The objective of this lab practical is to implement a decision tree algorithm to classify students as either average or clever based on given student data. Decision trees are widely used in machine learning and data mining for classification and regression tasks. In this lab, we will explore how to build a decision tree model and use it to classify students based on their attributes.

**Materials Used:**
- Python 3.x
- Jupyter Notebook
- Scikit-learn library
- Pandas library
- NumPy library
- Matplotlib library

**Dataset**:
For this lab, we will use a dataset that contains information about students and their performance. The dataset has the following columns:

- `Age`: Age of the student
- `StudyHours`: Number of hours the student studies per day
- `PreviousGrade`: Grade achieved in the previous exam
- `Result`: Classification label indicating whether the student is average (0) or clever (1)

**Procedure**:

1. Introduction to Decision Trees:
   - Decision trees are widely used in machine learning for classification tasks. They make decisions based on splitting criteria and feature importance. In this lab, we will implement a decision tree algorithm to classify students as average or clever based on their attributes, such as age, study hours, and previous grades.

2. Importing Required Libraries and Loading the Dataset:
   - Import the necessary libraries, including Scikit-learn, Pandas, NumPy, and Matplotlib.
   - Load the dataset into a Pandas DataFrame using the appropriate function or by reading from a file.

3. Exploratory Data Analysis:
   - Perform exploratory data analysis to understand the dataset.
   - Analyze the distribution of variables, detect any missing values, and handle them if necessary.
   - Visualize the relationships between variables using plots and charts.

4. Data Preprocessing:
- Split the dataset into input features (independent variables) and the target variable (dependent variable).
- Convert categorical variables into numerical representations using one-hot encoding or label encoding.
- Split the dataset into training and testing sets for model evaluation.

5. Building the Decision Tree Model:
- Import the DecisionTreeClassifier class from Scikit-learn.
- Instantiate the DecisionTreeClassifier model with the desired parameters.
- Fit the model to the training data using the `.fit()` method.

6. Model Evaluation and Prediction:
- Evaluate the model's performance using appropriate evaluation metrics such as accuracy, precision, recall, and F1-score.
- Make predictions on the testing data using the `.predict()` method.

7. Visualization of the Decision Tree:
- Visualize the decision tree using tree plotting techniques available in Scikit-learn or other visualization libraries.
- Interpret the decision tree structure and analyze the important features.

**Interpretation/Program/code:**

## 1: Import Libraries and Load Data

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

# Example data
data = pd.DataFrame({
    'Age': [16, 17, 16, 15, 18, 17, 16, 15, 18, 17],
    'StudyHours': [2, 3, 1, 4, 5, 2, 3, 1, 4, 5],
    'PreviousGrade': [60, 70, 50, 80, 90, 60, 70, 50, 80, 90],
    'Result': [0, 1, 0, 1, 1, 0, 1, 0, 1, 1]
})

print(data.head())  # Display dataset
```

## 2: Exploratory Data Analysis (EDA)

```
data = pd.read_csv('student_data.csv')
print('\n')
# print(data.show())
# Check for missing values and basic statistics
print(data.isnull().sum())
print(data.describe())

# Visualize relationships between variables
plt.scatter(data['StudyHours'], data['PreviousGrade'], c=data['Result'])
plt.xlabel('Study Hours')
plt.ylabel('Previous Grade')
plt.title('Scatter plot of Study Hours vs Previous Grade')
plt.show()
```

## 3: Data Preprocessing

```
# Features (X) and target (y)
X = data[['Age', 'StudyHours', 'PreviousGrade']]
y = data['Result']

# Split 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)
```

## 4: Build and Train the Decision Tree

```
# Instantiate the DecisionTreeClassifier
clf = DecisionTreeClassifier()

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

# Make predictions on the test set
y_pred = clf.predict(X_test)
```

## 5: Model Evaluation

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

# Detailed classification report
print(classification_report(y_test, y_pred))
```

## 6. Visualization of the Decision Tree

```
# Visualize the decision tree
plt.figure(figsize=(12,8))
tree.plot_tree(clf, filled=True, feature_names=['Age', 'StudyHours', 'PreviousGrade'], class_names=['Average', 'Clever'])
plt.show()
```

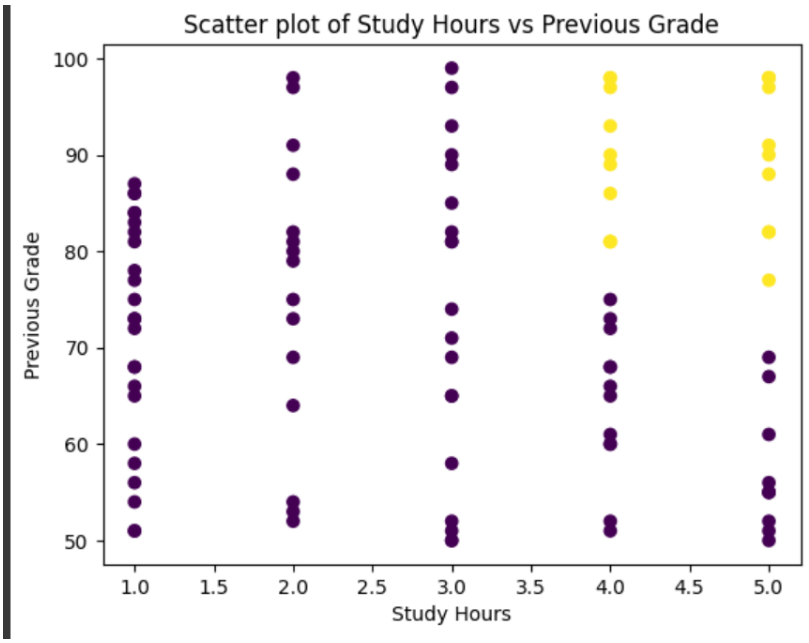
**Output**:

1.

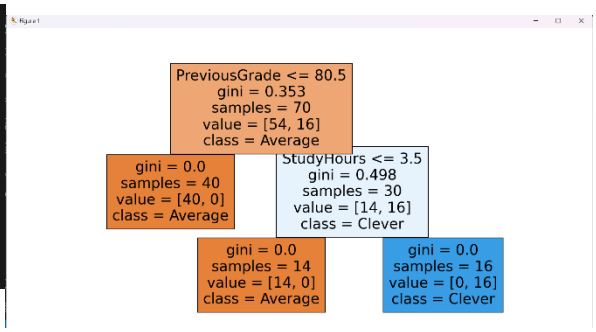
| | Age | StudyHours | PreviousGrade | Result |
|---|---|---|---|---|
| 0 | 17 | 2 | 82 | 0 |
| 1 | 18 | 2 | 91 | 0 |
| 2 | 15 | 4 | 93 | 1 |
| 3 | 17 | 2 | 73 | 0 |
| 4 | 17 | 2 | 64 | 0 |

2.

```
              Age  StudyHours  PreviousGrade      Result
count  100.000000   100.00000     100.000000  100.000000
mean    16.640000     2.99000      73.930000    0.200000
std      1.114822     1.46677      15.052269    0.402015
min     15.000000     1.00000      50.000000    0.000000
25%     16.000000     2.00000      60.750000    0.000000
50%     17.000000     3.00000      74.500000    0.000000
75%     18.000000     4.00000      86.000000    0.000000
max     18.000000     5.00000      99.000000    1.000000
Age             0
StudyHours      0
PreviousGrade   0
Result          0
dtype: int64
```




```
Accuracy: 96.67%
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        26
           1       1.00      0.75      0.86         4

    accuracy                           0.97        30
   macro avg       0.98      0.88      0.92        30
weighted avg       0.97      0.97      0.96        30
```



**Conclusion**:

- The implementation of the decision tree algorithm proved effective in classifying students as average or clever based on their attributes. Decision trees provide interpretable results and can be used in various domains for classification tasks. The decision tree model offers insights into the important features contributing to the classification. This lab demonstrates the practical application of decision trees for student classification.

**Quiz**:

1. In decision tree classification, what is the main objective of the splitting criterion?

a) To maximize the number of features used for classification
b) To minimize the number of nodes in the decision tree
c) To maximize the accuracy of classification
d) To minimize the entropy or Gini impurity of the resulting subsets

2. What is the purpose of pruning in decision tree algorithms?

a) To remove irrelevant features from the dataset
b) To prevent overfitting and improve generalization of the decision tree
c) To improve the interpretability of the decision tree
d) To reduce the time complexity of the decision tree algorithm

**Suggested References**:-

1. "Pattern Recognition and Machine Learning" by Christopher M. Bishop
2. Dinesh Kumar, Business Analytics, Wiley India Business alytics: The Science
3. V.K. Jain, Data Science & Analytics, Khanna Book Publishing, New Delhi of Dat
4. Data Science For Dummies by Lillian Pierson , Jake Porway

**Rubrics wise marks obtained**

| Understanding of Problem | Analysis of the Problem | Capability of writing program | Documentation | Total |
|---|---|---|---|---|
| 02 | 02 | 05 | 01 | 10 |
| | | | | |