INDEX

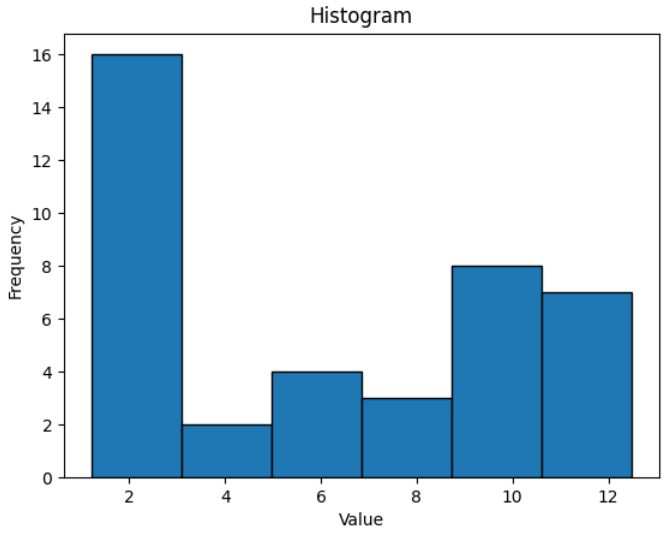
S. NO.	PROGRAM	DATE	SIGNATURE
1	Write Python Program to Plot Frequency Polygon, Ogive , Histogram of the Unemployment Rates Data provided to you.	8-9-2023	
2	Write Python Program to generate some random data. Plot Box plots using this random data.	15-9-2023	
3	Write Python Program to generate data from Binomial distribution and show the pdf plot for various of n and p.	22-9-2023	
4	Write Python Program to generate Poisson distribution and show the pdf plot for various values of Lambda.	29-9-2023	
5	Write Python Program to generate data from Normal distribution and show the pdf plot for various of μ, σ	6-10-2023	
6	Write Python Program to generate exponential distribution and show the pdf plot for various values of Lambda.	13-10-2023	
7	Write a Python Program to import and export data using Pandas. Demonstrate various data pre-processing techniques.	20-10-2023	
8	Write a Python Program to implement Simple and Multiple Linear Regression.	27-10-2023	
9	Write a Python Program to implement Logistic Regression on a given dataset.	3-11-2023	
10	Write a Python Program to implement Decision tree on a given dataset.	10-11-2023	
11	Write a Python Program to implement K-Means clustering algorithm.	17-11-2023	
12	Write a program to generate Stem and Leaf plot.	24-11-2023	
13	Write a program for ANOVA test.	1-12-2023	
14	Write a program for z-testing and t-testing.	8-12-2023	

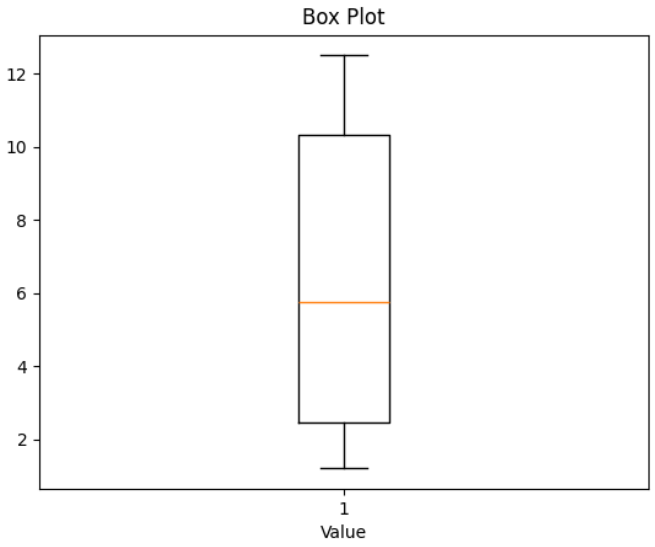
PROGRAM -1

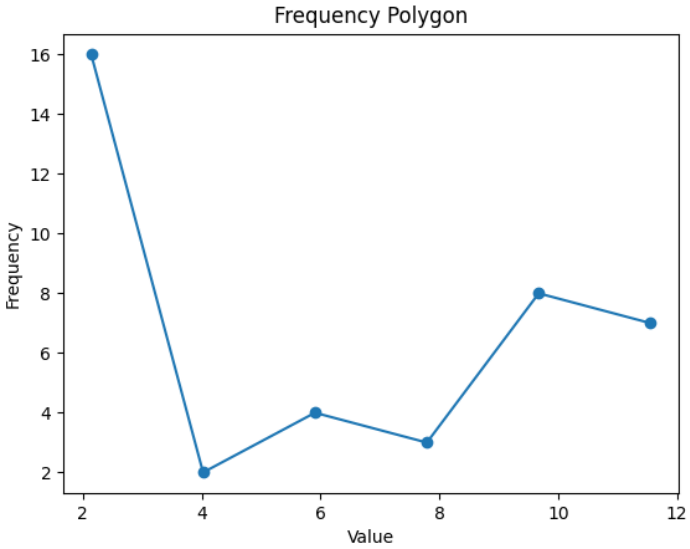
Write Python Program to Plot Frequency Polygon, Ogive , Histogram of the Unemployment Rates Data provided to you.

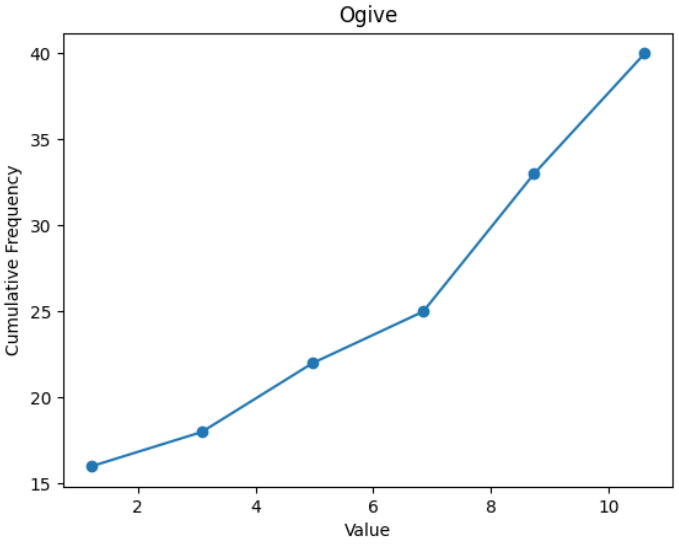
```
import matplotlib.pyplot as plt
import numpy as np

# Sample data - each value represents a bin and it's number of occurrences the frequency
data = [1.6, 2.1, 4.2, 8.6, 9.6, 1.5, 2.7, 4.6, 10, 10.4, 1.2, 2.3, 5.2, 10.5,
        11.8, 1.4, 2.5, 5.4, 10.6, 12.3, 1.6, 2.8,
        6.1, 10.8, 11.8, 1.2, 2.9, 6.5, 10.3, 12.5, 1.6, 2.8, 7.6,
        9.6, 12.4, 1.6, 2.9, 8.3, 9.1, 11.8, ]

# HISTOGRAM
plt.hist(data, bins=6, edgecolor='black')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()

# data = [65, 72, 78, 82, 88, 90, 94, 98, 100, 105, 110, 115, 120, 125]
# Create a box plot
plt.boxplot(data)
plt.xlabel('Value')
plt.title('Box Plot')
plt.show()

# Create a histogram to obtain the frequency data
hist, bin_edges = np.histogram(data, bins=6, range=(min(data), max(data)))
print(hist)
print(bin_edges)

# Calculate midpoints of each bin
bin_midpoints = (bin_edges[1:] + bin_edges[:-1]) / 2

# FREQUENCY POLYGON
plt.plot(bin_midpoints, hist, marker='o')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Frequency Polygon')
plt.show()
```

```
# Calculate cumulative frequencies
cumulative_freq = np.cumsum(hist)

# Plot the OGIVE
plt.plot(bin_edges[:-1], cumulative_freq, marker='o', linestyle='-')
plt.xlabel('Value')
plt.ylabel('Cumulative Frequency')
plt.title('Ogive')
plt.show()
```





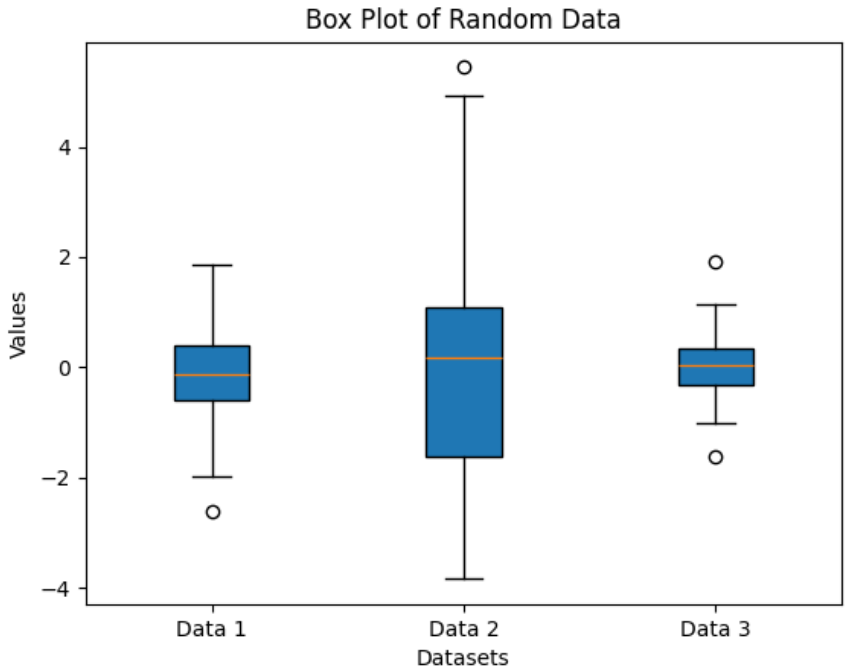




PROGRAM -2

Write Python Program to generate some random data. Plot Box plots using this random data.

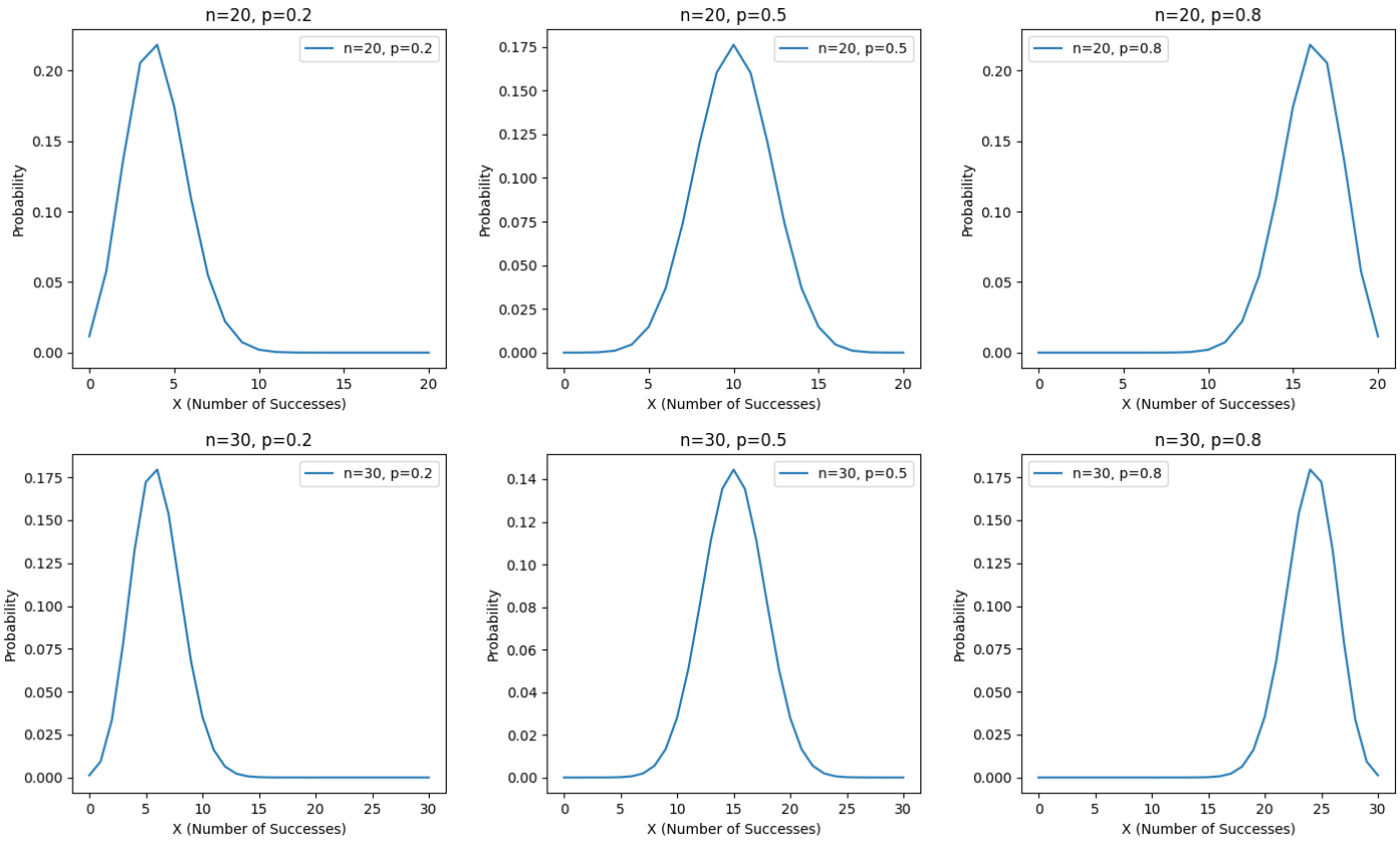
```
import numpy as np
import matplotlib.pyplot as plt
# Set a random seed for reproducibility
np.random.seed(42)
# Generate random data
data1 = np.random.normal(loc=0, scale=1, size=100)
data2 = np.random.normal(loc=0, scale=2, size=100)
data3 = np.random.normal(loc=0, scale=0.5, size=100)

# Create a list of data for box plots
data_to_plot = [data1, data2, data3]
# print(data_to_plot)

# Create a box plot
plt.boxplot(data_to_plot, vert=True, patch_artist=True)

# Add labels and title
plt.xticks([1, 2, 3], ['Data 1', 'Data 2', 'Data 3'])
plt.xlabel('Datasets')
plt.ylabel('Values')
plt.title('Box Plot of Random Data')
plt.show()
```



PROGRAM -3

Write Python Program to generate data from Binomial distribution and show the pdf plot for various of n and p.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom
ns = [20,30]
ps = [0.2, 0.5, 0.8]
plt.figure(figsize=(12, 8))
plot_index = 1
for n in ns:
    for p in ps:
        x = np.arange(0, n+1)
        pmf = binom.pmf(x, n, p)
        plt.subplot(len(ns), len(ps), plot_index)
        plt.plot(x, pmf, label=f'n={n}, p={p}')
        plt.title(f'n={n}, p={p}')
        plt.xlabel('X (Number of Successes)')
        plt.ylabel('Probability')
        plt.legend()
        plot_index += 1
plt.tight_layout()
plt.show()
```

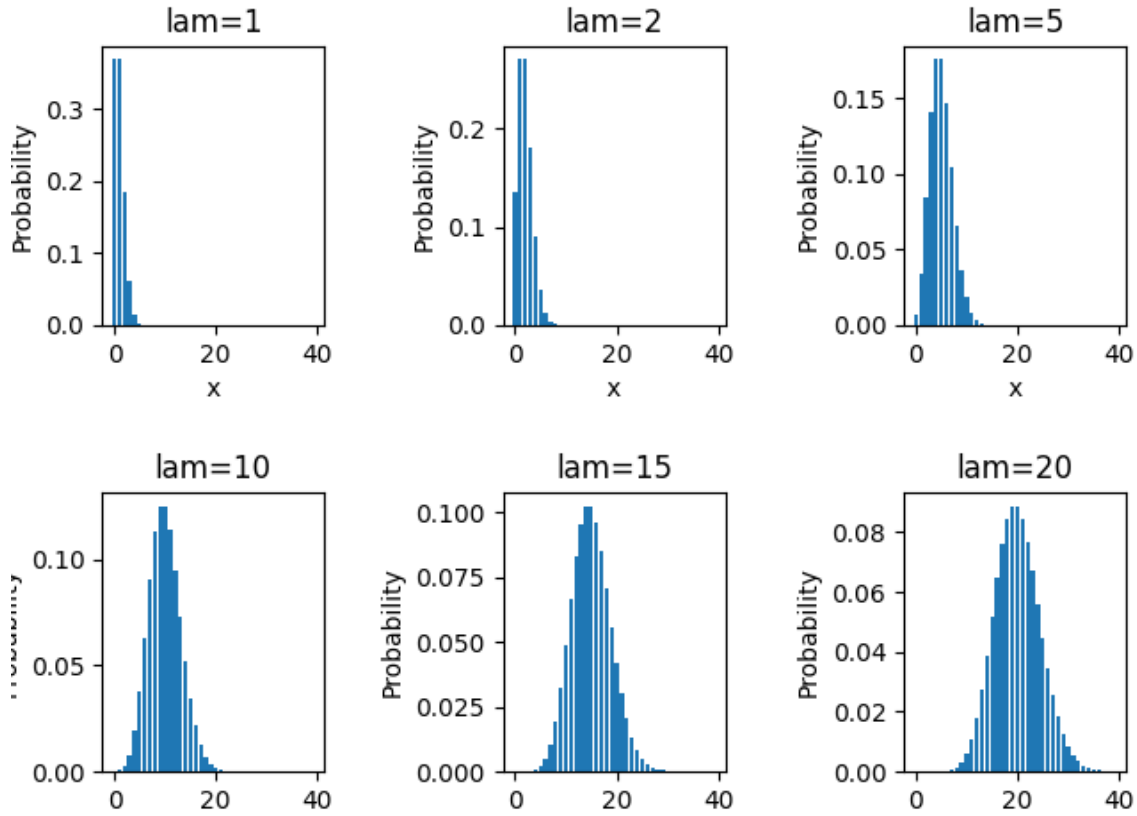


PROGRAM - 4

Write Python Program to generate Poisson distribution and show the pdf plot for various values of Lambda.

```
import numpy as np
import math
import scipy.special
import matplotlib.pyplot as plt


def poisson_dist(x, lam):
    u = []
    z = []
    for i in range(len(x)):
        u.append(math.factorial(i))
        z.append(lam ** i)
        # print(prob_density)
    q = np.zeros(len(x))
    p = np.zeros(len(x))
    q = np.array(u)
    p = np.array(z)
    wz = p / q
    prob_density = wz * np.exp(-lam)
    return prob_density

lam = 1
x = np.arange(0, 40, 1)
print(x)

result = poisson_dist(x, lam)

fig, axs = plt.subplots(2, 3)

axs[0, 0].set_title('lam=1')
axs[0, 0].bar(x, result)
axs[0, 0].set_title('lam=1')
axs[0, 0].set_xlabel('x')
axs[0, 0].set_ylabel('Probability')

lam = 2
result = poisson_dist(x, lam)

axs[0, 1].set_title('lam=2')
```

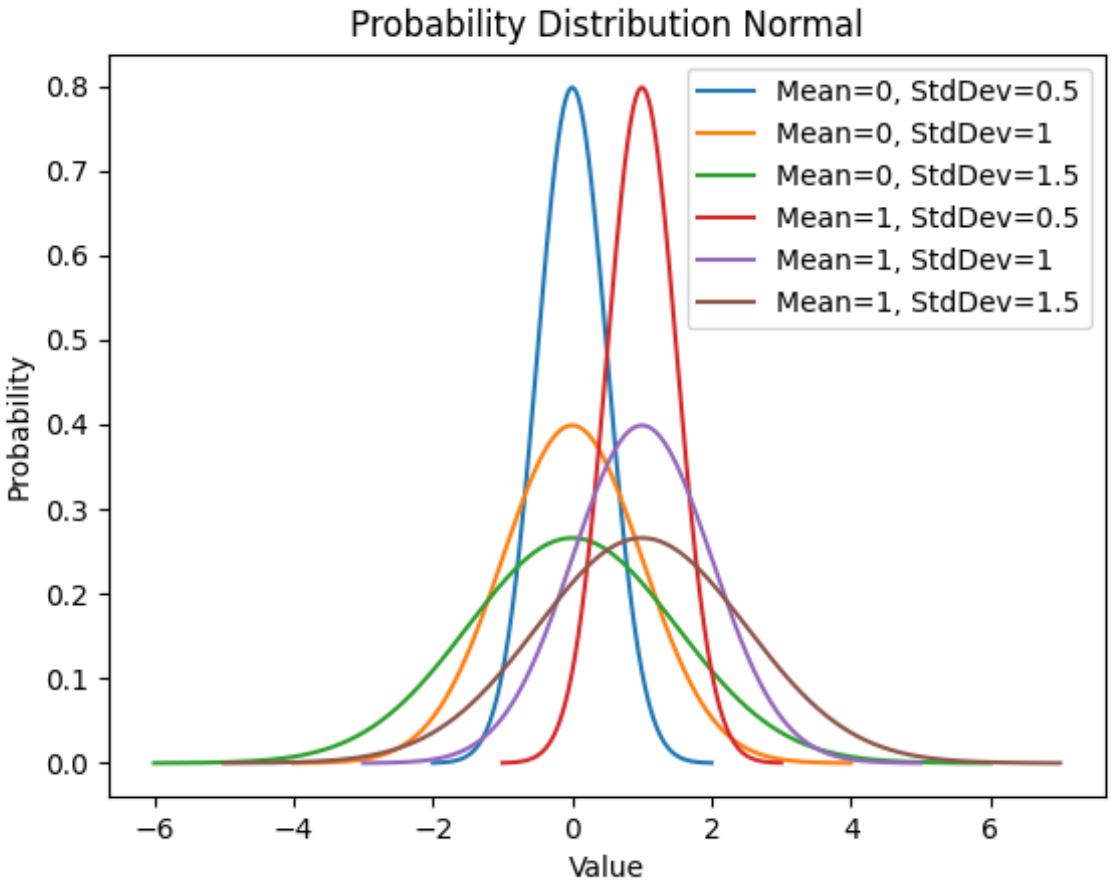
```
axs[0, 1].bar(x, result)
axs[0, 1].set_title('lam=2')
axs[0, 1].set_xlabel('x')
axs[0, 1].set_ylabel('Probability')

lam = 5
result = poisson_dist(x, lam)

axs[0, 2].set_title('lam=5')
axs[0, 2].bar(x, result)
axs[0, 2].set_xlabel('x')
axs[0, 2].set_ylabel('Probability')

lam = 10
result = poisson_dist(x, lam)

axs[1, 0].set_title('lam=10')
axs[1, 0].bar(x, result)
axs[1, 0].set_xlabel('x')
axs[1, 0].set_ylabel('Probability')

lam = 15
result = poisson_dist(x, lam)
axs[1, 1].set_title('lam=15')
axs[1, 1].bar(x, result)
axs[1, 1].set_xlabel('x')
axs[1, 1].set_ylabel('Probability')

lam = 20
result = poisson_dist(x, lam)

axs[1, 2].set_title('lam=20')
axs[1, 2].bar(x, result)
axs[1, 2].set_xlabel('x')
axs[1, 2].set_ylabel('Probability')

fig.subplots_adjust(left=0.08, right=0.98, bottom=0.05, top=0.9,
                    hspace=0.6, wspace=0.8)

plt.show()
```


lam=1
lam=2
lam=5
lam=10
lam=15
lam=20
Probability
x
0.0
0.1
0.2
0.3
0.00
0.05
0.10
0.15
0.000
0.025
0.050
0.075
0.100
0.02
0.04
0.06
0.08
0
20
40

PROGRAM - 5

Write Python Program to generate data from Normal distribution and show the pdf plot for various of μ, σ

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm
def normal_dist(x, mean, sd):
    y = sd * np.sqrt(2 * np.pi)
    z = 1 / y
    prob_density = z * np.exp(-0.5 * ((x - mean) / sd) ** 2)
    return prob_density

def plot_normal_distribution(mu, sigma):
    x_range = np.linspace(mu - 4*sigma, mu + 4*sigma, 1000)
    y = norm.pdf(x_range, mu, sigma)
    plt.plot(x_range, y, label=f'Mean={mu}, StdDev={sigma}')

# Values for mean (μ) and standard deviation (σ)
mu_values = [0, 1]
sigma_values = [0.5, 1, 1.5]

# Plot normal distributions for various μ and σ
for mu in mu_values:
    for sigma in sigma_values:
        x = np.random.normal(mu, sigma, 10000)
        result = normal_dist(x, mu, sigma)
        plot_normal_distribution(mu, sigma)

# Add labels and title
plt.xlabel('Value')
plt.ylabel('Probability')
plt.title('Probability Distribution Normal')

# Show legend
plt.legend()

# Show the plot
plt.show()
```

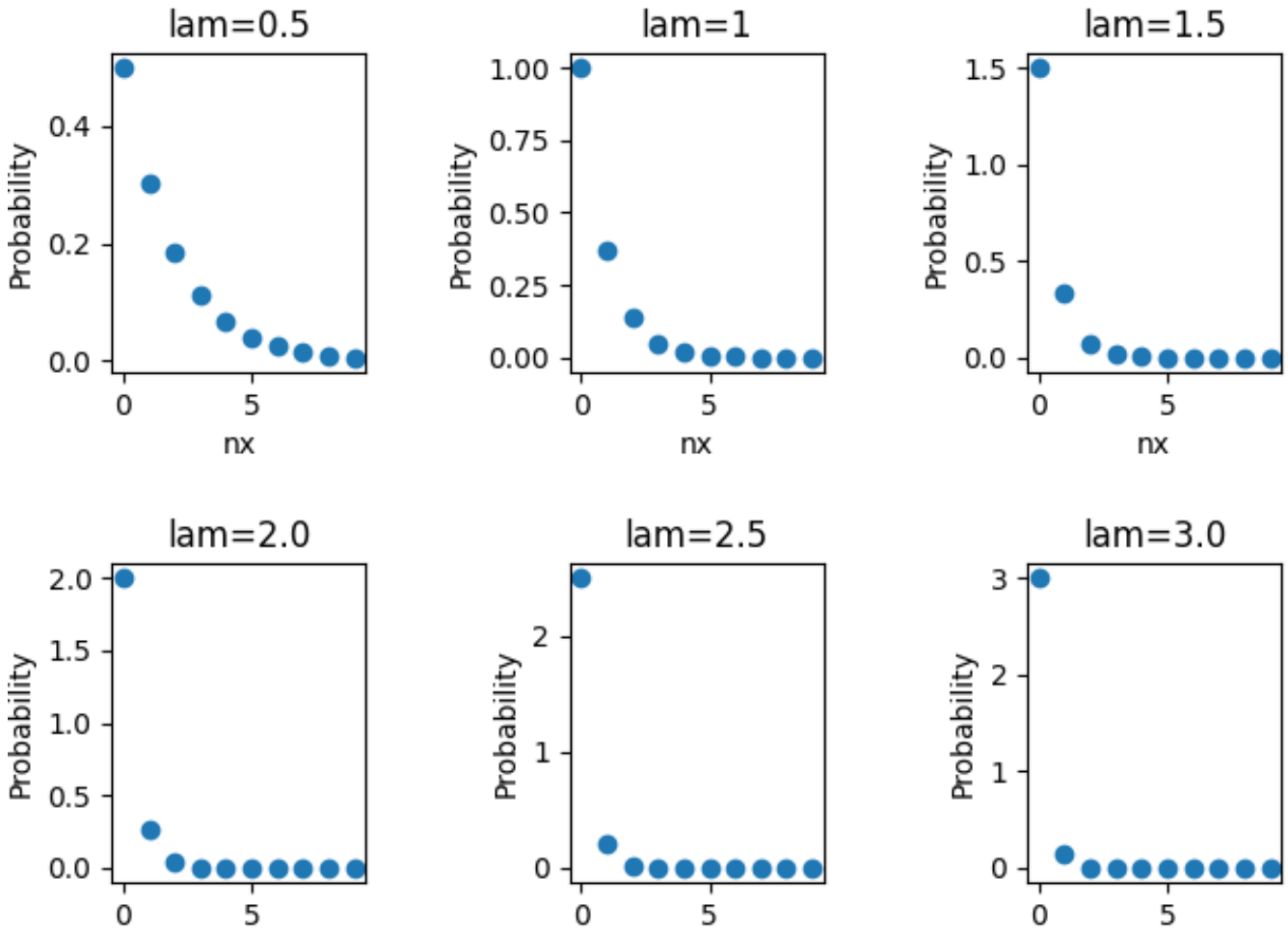

Probability Distribution Normal
Mean=0, StdDev=0.5
Mean=0, StdDev=1
Mean=0, StdDev=1.5
Mean=1, StdDev=0.5
Mean=1, StdDev=1
Mean=1, StdDev=1.5
Probability
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
−6
−4
−2
0
2
4
6
Value

PROGRAM - 6

Write Python Program to generate exponential distribution and show the pdf plot for various values of Lambda.

```
import numpy as np
import math
import scipy.special
import matplotlib.pyplot as plt

def exp_dist(nx, lam):
    nz = []
    for i in range(len(nx)):
        nz.append(-lam * i)

    pq = np.zeros(len(nx))
    pq = np.array(nz)
    prob_density = lam * np.exp(pq)
    return prob_density

lam = 0.5
nx = np.arange(0, 10, 1)
# print(nx)

result = exp_dist(nx, lam)
# print(result)

fig, axs = plt.subplots(2, 3)

axs[0, 0].set_title('lam=0.5')
axs[0, 0].scatter(nx, result)
axs[0, 0].set_xlabel('nx')
axs[0, 0].set_ylabel('Probability')

lam = 1.0
result = exp_dist(nx, lam)
axs[0, 1].set_title('lam=1')
axs[0, 1].scatter(nx, result)
axs[0, 1].set_xlabel('nx')
axs[0, 1].set_ylabel('Probability')

lam = 1.5
result = exp_dist(nx, lam)
axs[0, 2].set_title('lam=1.5')
```

```
axs[0, 2].scatter(nx, result)
axs[0, 2].set_xlabel('nx')
axs[0, 2].set_ylabel('Probability')

lam = 2.0
result = exp_dist(nx, lam)
axs[1, 0].set_title('lam=2.0')
axs[1, 0].scatter(nx, result)
axs[1, 0].set_xlabel('nx')
axs[1, 0].set_ylabel('Probability')

lam = 2.5
result = exp_dist(nx, lam)
axs[1, 1].set_title('lam=2.5')
axs[1, 1].scatter(nx, result)
axs[1, 1].set_xlabel('nx')
axs[1, 1].set_ylabel('Probability')

lam = 3.0
result = exp_dist(nx, lam)
axs[1, 2].set_title('lam=3.0')
axs[1, 2].scatter(nx, result)
axs[1, 2].set_xlabel('nx')
axs[1, 2].set_ylabel('Probability')

fig.subplots_adjust(left=0.08, right=0.98, bottom=0.05, top=0.9,
                    hspace=0.6, wspace=0.8)

plt.show()
```


lam=0.5
lam=1
lam=1.5
lam=2.0
lam=2.5
lam=3.0
Probability
nx
0.0
0.2
0.4
0.00
0.25
0.50
0.75
1.00
0.5
1.0
1.5
0
1
2
3
0
5

PROGRAM - 7

Write a Python Program to import and export data using Pandas. Demonstrate various data pre-processing techniques.

```
import pandas as pd

# Step 1: Import data
file_path = '/Users/mukulhooda/Desktop/College/3rd Year/Machine Learning-1/Lab File/Programs/titanic/train.csv '
titanic_data = pd.read_csv(file_path)

print()
print("Original Data:")
print(titanic_data.head())
print("\n")


#Data Exploration and Pre-processing
# 1: Check for missing values
print("Missing Values:")
print(titanic_data.isnull().sum())
print("\n")

# 2: Handling Missing Values
titanic_data.dropna(subset=['Age'], inplace=True)
titanic_data['Embarked'].fillna(titanic_data['Embarked'].mode()[0], inplace=True)

# Display data after handling missing values
print("Data after handling missing values:")
print(titanic_data.head())
print("\n")

# 3: Removing Duplicates
titanic_data.drop_duplicates(inplace=True)

# Data after removing duplicates
print("Data after removing duplicates:")
print(titanic_data.head())
print("\n")

# 4: Renaming Columns
titanic_data.rename(columns={'Pclass': 'PassengerClass', 'Survived': 'Survival'}, inplace=True)
```

```
# Data after renaming columns
print("Data after renaming columns:")
print(titanic_data.head())
print("\n")

# 5: Changing Data Types
titanic_data['Fare'] = titanic_data['Fare'].astype(int)  # Change data type of Fare to integer

# Data after changing data types
print("Data after changing data types:")
print(titanic_data.head())
print("\n")

# 6: Adding a New Column
titanic_data['FamilySize'] = titanic_data['SibSp'] + titanic_data['Parch'] + 1  # Add a new column for family size

# Data after adding a new column
print("Data after adding a new column:")
print(titanic_data.head())
print("\n")

# Export data
export_file_path = 'titanic_processed.csv'
titanic_data.to_csv(export_file_path, index=False)

print(f"Pre-processed data has been exported to {export_file_path}")
```

"/Users/mukulhooda/Desktop/College/3rd Year/Machine Learning-1/Lab File /Programs/venv/bin/python" /Users/mukulhooda/Desktop/College/3rd Year/ Machine Learning-1/Lab File/Programs/Program 7.py

Original Data:

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
0	1	0	3	...	7.2500	NaN	S
1	2	1	1	...	71.2833	C85	C
2	3	1	3	...	7.9250	NaN	S
3	4	1	1	...	53.1000	C123	S
4	5	0	3	...	8.0500	NaN	S

[5 rows x 12 columns]

Missing Values:

```
PassengerId     0
Survived        0
Pclass          0
Name            0
Sex             0
Age           177
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin         687
Embarked        2
dtype: int64
```

Data after handling missing values:

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
0	1	0	3	...	7.2500	NaN	S
1	2	1	1	...	71.2833	C85	C
2	3	1	3	...	7.9250	NaN	S
3	4	1	1	...	53.1000	C123	S
4	5	0	3	...	8.0500	NaN	S

[5 rows x 12 columns]

Data after removing duplicates:

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
0	1	0	3	...	7.2500	NaN	S
1	2	1	1	...	71.2833	C85	C
2	3	1	3	...	7.9250	NaN	S

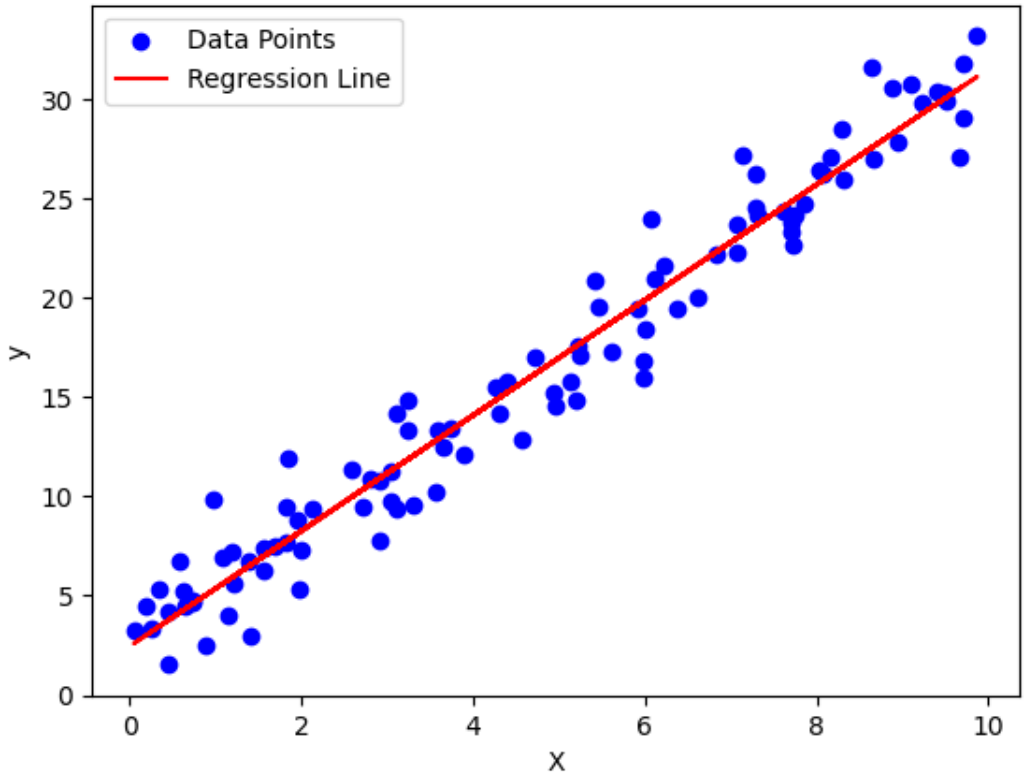
```
3            4         1       1 ...  53.1000  C123        S
4            5         0       3 ...   8.0500   NaN        S

[5 rows x 12 columns]


Data after renaming columns:
   PassengerId  Survival  PassengerClass  ...     Fare Cabin  Embarked
0            1         0               3 ...   7.2500   NaN         S
1            2         1               1 ...  71.2833   C85         C
2            3         1               3 ...   7.9250   NaN         S
3            4         1               1 ...  53.1000  C123         S
4            5         0               3 ...   8.0500   NaN         S

[5 rows x 12 columns]


Data after changing data types:
   PassengerId  Survival  PassengerClass  ... Fare Cabin  Embarked
0            1         0               3 ...    7   NaN         S
1            2         1               1 ...   71   C85         C
2            3         1               3 ...    7   NaN         S
3            4         1               1 ...   53  C123         S
4            5         0               3 ...    8   NaN         S

[5 rows x 12 columns]


Data after adding a new column:
   PassengerId  Survival  PassengerClass  ... Cabin Embarked  FamilySize
0            1         0               3 ...   NaN        S           2
1            2         1               1 ...   C85        C           2
2            3         1               3 ...   NaN        S           1
3            4         1               1 ...  C123        S           2
4            5         0               3 ...   NaN        S           1

[5 rows x 13 columns]


Pre-processed data has been exported to titanic_processed.csv

Process finished with exit code 0
```

PROGRAM - 8

Write a Python Program to implement Simple and Multiple Linear Regression.

Simple Linear Regression

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

def simple_linear_regression(X, y):
    # Calculate the mean of X and y
    mean_X = np.mean(X)
    mean_y = np.mean(y)

    # Calculate the slope (m) and y-intercept (b) using the least squares method
    numerator = np.sum((X - mean_X) * (y - mean_y))
    denominator = np.sum((X - mean_X) ** 2)

    slope = numerator / denominator
    intercept = mean_y - slope * mean_X

    return slope, intercept

def plot_regression_line(X, y, slope, intercept):
    plt.scatter(X, y, color='blue', label='Data Points')
    plt.plot(X, slope * X + intercept, color='red', label='Regression Line')
    plt.xlabel('X')
    plt.ylabel('y')
    plt.legend()
    plt.show()

# Generate some random data
np.random.seed(42)
X = np.random.rand(100, 1) * 10  # Random values for X
y = 3 * X + 2 + np.random.randn(100, 1) * 2  # Linear relationship with noise
slope, intercept = simple_linear_regression(X, y)

print("Slope:", slope)
print("Intercept:", intercept)

plot_regression_line(X, y, slope, intercept)
```

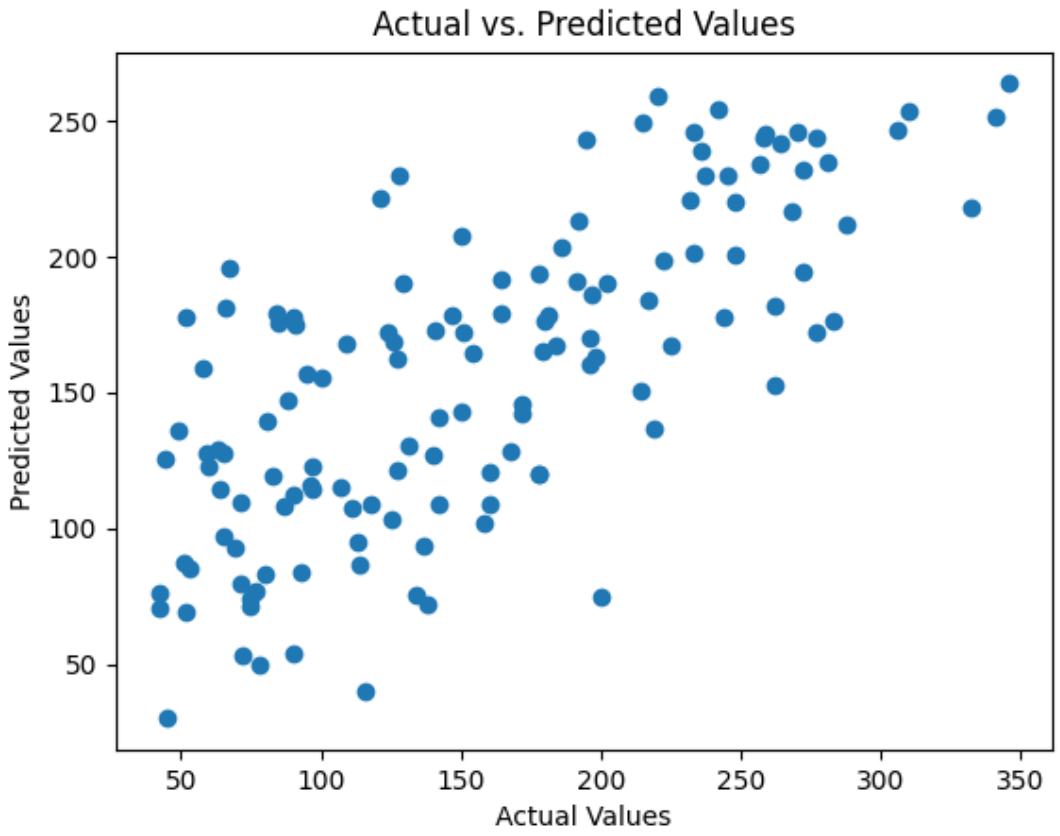


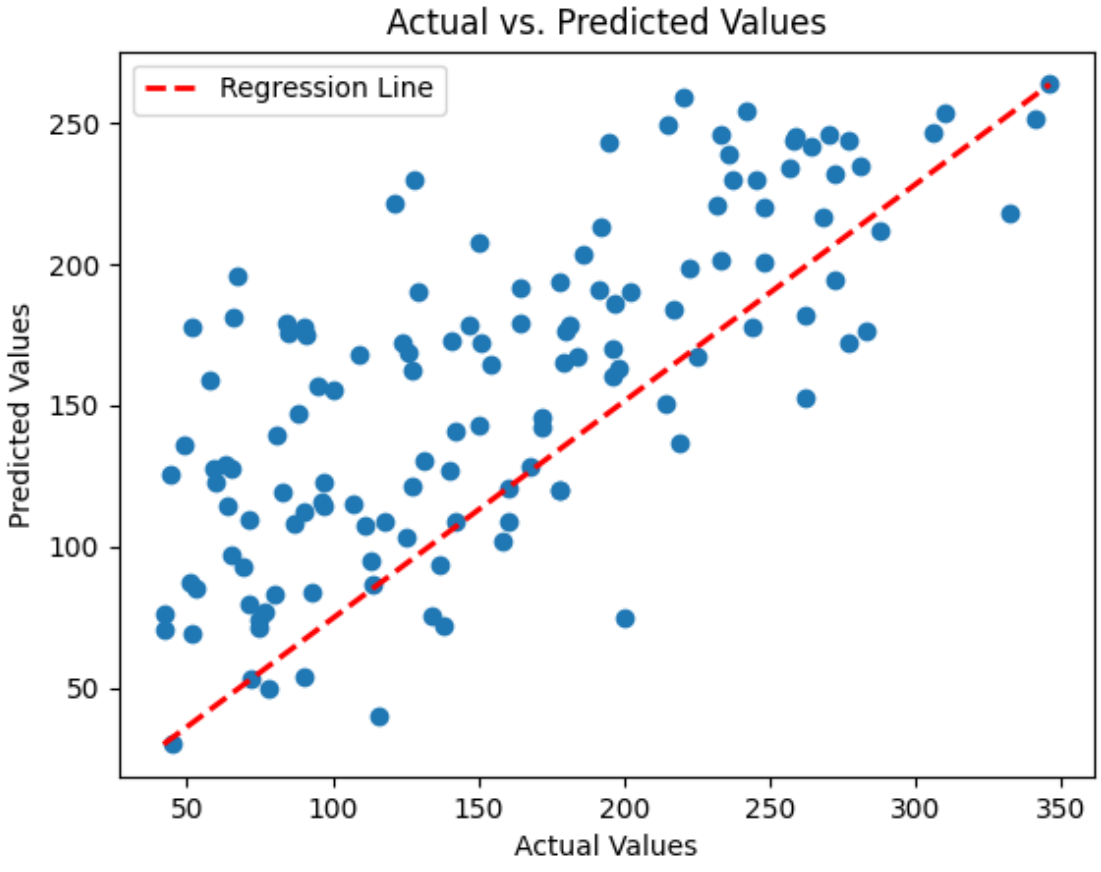
Multiple Linear Regression

```
# Importing libraries
from sklearn import datasets
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

print()
# Loading dataset
diabetes = datasets.load_diabetes()

# print(diabetes.DESCR)

X = diabetes.data
Y = diabetes.target

# print(X.shape, Y.shape)
```

```
# Viewing the data in the form of a dataframe
X_df = pd.DataFrame(X, columns=diabetes.feature_names)
X_df.describe()


# Implementing from scratch with our own functions

def add_bias_feature(X):
    return np.hstack((np.ones((X.shape[0], 1)), X))


def fit(X, y):
    X_with_bias = add_bias_feature(X)

                                                                    w e i g h t s          =
np.linalg.inv(X_with_bias.T.dot(X_with_bias)).dot(X_with_bias.T).dot(y)
    return weights


def predict(X, weights):
    X_with_bias = add_bias_feature(X)
    return np.dot(X_with_bias, weights)


def calculate_mse(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)


def calculate_r_squared(y_true, y_pred):
    mean_y = np.mean(y_true)
    ss_total = np.sum((y_true - mean_y) ** 2)
    ss_residual = np.sum((y_true - y_pred) ** 2)
    r_squared = 1 - (ss_residual / ss_total)
    return r_squared


def plotter(ax, y_true, y_pred):
    ax.plot([min(y_true), max(y_true)], [min(y_pred), max(y_pred)], linestyle='--',
            color='red', linewidth=2, label='Regression Line')



# 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)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)


# Train the linear regression model
weights = fit(X_train, y_train)


# Extract intercept and coefficients
intercept = weights[0]
coefficients = weights[1:]


# Make predictions on the test set
y_pred = predict(X_test, weights)


# # Calculate metrics
# mse = calculate_mse(y_test, y_pred)
# r_squared = calculate_r_squared(y_test, y_pred)


print(f"Coefficients: {coefficients}")
print(f"Intercept: {intercept}")
# print(f"Mean Squared Error: {mse}")
# print(f"R-squared (Coefficient of Determination): {r_squared}")


# Plotting predictions vs. actual values
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs. Predicted Values")
plt.show()

# Plotting predictions vs. actual values with regression line
fig, ax = plt.subplots()
ax.scatter(y_test, y_pred)
plotter(ax, y_test, y_pred)
ax.set_xlabel("Actual Values")
ax.set_ylabel("Predicted Values")
ax.set_title("Actual vs. Predicted Values")
ax.legend()
plt.show()
```


Actual vs. Predicted Values
Predicted Values
Actual Values
50
100
150
200
250
300
350


Actual vs. Predicted Values
Regression Line
Predicted Values
Actual Values
50
100
150
200
250
300
350

THEORY:

Linear Regression:

1. Definition:
Linear Regression is a statistical method used for modeling the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data.
The simplest form is simple linear regression, which involves only one independent variable, while multiple linear regression involves multiple independent variables.

2. Applications:
- Predicting sales based on advertising spending.
- Estimating house prices based on features like size and location.
- Analyzing the impact of variables on exam scores in education.

3. Key Concepts:
- Linear Equation:
The linear regression model assumes a linear relationship between the dependent variable (Y) and independent variable(s) (X).

Simple Linear Regression Equation:
$$ Y = \beta_0 + \beta_1 X + \epsilon $$

Where:
- Y is the dependent variable.
- X is the independent variable.
- β_0 is the y-intercept.
- β_1 is the slope.
- ϵ is the error term.

- Least Squares Method:
The coefficients (β_0 and β_1) are estimated using the least squares method, minimizing the sum of squared differences between the observed and predicted values.

- Assumptions:
Linear Regression makes several assumptions, including linearity, independence, homoscedasticity (constant variance), and normality of errors.

- Residuals:
Residuals represent the differences between observed and predicted values. A good linear regression model has residuals close to zero, indicating accurate predictions.

PROGRAM - 9

Write a Python Program to implement Logistic Regression on a given dataset.

```
import numpy as np
import pandas as pd
from sklearn import datasets, preprocessing
import matplotlib.pyplot as plt
import math
from sklearn.model_selection import train_test_split

# Loading dataset
iris = datasets.load_iris()

X, y = iris.data, (iris.target == 2).astype(int)

# Split the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, y,
                                    test_size=0.2,
                                    random_state=42)

# print(X_train.shape, Y_train.shape, X_test.shape, Y_test.shape)

def Sigmoid(x):
    return 1 / (1 + np.exp(-x))


def Cost(X_train, Y_train, m):
    cost_ = 0
    N = X_train.shape[0]
    for i in range(N):
        agg = (X_train[i] * m).sum()
        h = Sigmoid(agg)
        cost = -Y_train[i] * np.log(h) - (1 - Y_train[i]) * np.log(1 - h)
        cost_ += cost

    return cost_


def Step_Gradient(X_train, Y_train, lr, m):
    N = X_train.shape[0]
    slope_m = np.zeros(X_train.shape[1])
```

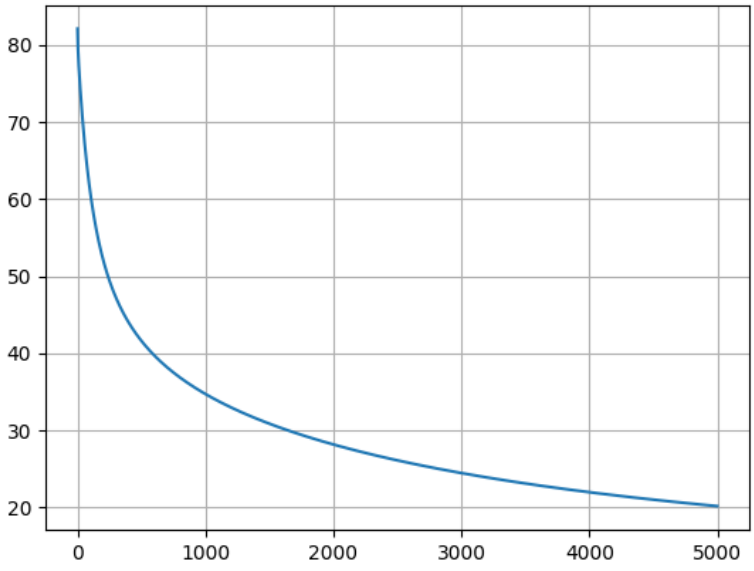
```
    for i in range(N):
        agg = (X_train[i] * m).sum()
        h = Sigmoid(agg)
        slope_m += (-1 / N) * (Y_train[i] - h) * X_train[i]

    m = m - lr * slope_m
    return m


def Fit(X_train, Y_train, epochs=100, lr=0.01):
    m = np.zeros(X_train.shape[1])
    cost_array = []
    unit = epochs // 100
    for i in range(epochs):
        m = Step_Gradient(X_train, Y_train, lr, m)
        cost_ = Cost(X_train, Y_train, m)
        cost_array.append(cost_)
        if i % unit == 0:
            print("Epoch:{}, Cost:{}".format(i, cost_))

    return m, cost_array


def Predict(X_test, m):
    y_pred = []
    N = X_test.shape[0]
    for i in range(N):
        agg = (X_test[i] * m).sum()
        h = Sigmoid(agg)
        if h >= 0.5:
            y_pred.append(1)
        else:
            y_pred.append(0)

    return np.array(y_pred)


def Accuracy(Y_test, Y_pred):
    correct = 0
    N = Y_test.shape[0]
    correct = (Y_test == Y_pred).sum()

    return (correct / N) * 100
```

```
m, cost_array = Fit(X_train, Y_train, 5000, 0.01)
print(m)
plt.plot(cost_array)
plt.grid()
plt.show()
Y_pred_train = Predict(X_train, m)
Accuracy(Y_train, Y_pred_train)
Y_pred_val = Predict(X_test, m)
```



```
"/Users/mukulhooda/Desktop/College/3rd Year/Machine Learning-1/Lab File
/Programs/venv/bin/python" /Users/mukulhooda/Desktop/College/3rd Year/
Machine Learning-1/Lab File/Programs/Program 9.py
Epoch:0, Cost:82.11348370439097
Epoch:50, Cost:68.3272022957023
Epoch:100, Cost:60.84657950473642
Epoch:150, Cost:55.7937306620496
Epoch:200, Cost:52.16485104037949
Epoch:250, Cost:49.4192239940126
Epoch:300, Cost:47.25187691322189
Epoch:350, Cost:45.48168233438514
Epoch:400, Cost:43.99567017185994
Epoch:450, Cost:42.72022870322254
Epoch:500, Cost:41.605519941087
Epoch:550, Cost:40.61665950231294
Epoch:600, Cost:39.728512779674325
Epoch:650, Cost:38.92250684151344
Epoch:700, Cost:38.18461018279281
Epoch:750, Cost:37.50401332059155
Epoch:800, Cost:36.872243562800676
Epoch:850, Cost:36.28255657050414
Epoch:900, Cost:35.72950900583452
Epoch:950, Cost:35.20865245304289
Epoch:1000, Cost:34.7163102952571
Epoch:1050, Cost:34.24941243901712
Epoch:1100, Cost:33.8053710917946
Epoch:1150, Cost:33.38198614450601
Epoch:1200, Cost:32.97737221942656
Epoch:1250, Cost:32.58990178877993
Epoch:1300, Cost:32.21816036333503
Epoch:1350, Cost:31.860910851140932
Epoch:1400, Cost:31.517064957855
Epoch:1450, Cost:31.185660047935198
Epoch:1500, Cost:30.865840279972378
Epoch:1550, Cost:30.556841116162797
Epoch:1600, Cost:30.257976516878387
Epoch:1650, Cost:29.968628288107112
Epoch:1700, Cost:29.688237167231126
Epoch:1750, Cost:29.41629532175042
Epoch:1800, Cost:29.15234000364875
Epoch:1850, Cost:28.895948154526646
Epoch:1900, Cost:28.646731797304824
Epoch:1950, Cost:28.404334082087683
Epoch:2000, Cost:28.168425878784394
Epoch:2050, Cost:27.93870282888746
Epoch:2100, Cost:27.714882784582976
Epoch:2150, Cost:27.49670357600652
```

Epoch:2200, Cost:27.283921057643006
Epoch:2250, Cost:27.07630739311522
Epoch:2300, Cost:26.873649544319917
Epoch:2350, Cost:26.675747936359738
Epoch:2400, Cost:26.482415274229552
Epoch:2450, Cost:26.293475490937738
Epoch:2500, Cost:26.10876280982496
Epoch:2550, Cost:25.928120906409003
Epoch:2600, Cost:25.751402157224696
Epoch:2650, Cost:25.578466964922985
Epoch:2700, Cost:25.40918315040213
Epoch:2750, Cost:25.243425404017827
Epoch:2800, Cost:25.08107478899645
Epoch:2850, Cost:24.922018291091742
Epoch:2900, Cost:24.766148409304538
Epoch:2950, Cost:24.613362783150897
Epoch:3000, Cost:24.463563852535334
Epoch:3050, Cost:24.316658546774892
Epoch:3100, Cost:24.17255799974244
Epoch:3150, Cost:24.03117728846153
Epoch:3200, Cost:23.892435192800637
Epoch:3250, Cost:23.756253974188052
Epoch:3300, Cost:23.622559171506527
Epoch:3350, Cost:23.491279412533963
Epoch:3400, Cost:23.36234623947779
Epoch:3450, Cost:23.235693947308448
Epoch:3500, Cost:23.11125943373735
Epoch:3550, Cost:22.988982059805736
Epoch:3600, Cost:22.868803520159155
Epoch:3650, Cost:22.75066772217678
Epoch:3700, Cost:22.63452067320886
Epoch:3750, Cost:22.520310375249508
Epoch:3800, Cost:22.407986726438313
Epoch:3850, Cost:22.29750142884248
Epoch:3900, Cost:22.188807902023452
Epoch:3950, Cost:22.08186120193829
Epoch:4000, Cost:21.976617944767842
Epoch:4050, Cost:21.873036235300226
Epoch:4100, Cost:21.7710755995325
Epoch:4150, Cost:21.67069692118186
Epoch:4200, Cost:21.57186238182561
Epoch:4250, Cost:21.474535404412595
Epoch:4300, Cost:21.378680599911213
Epoch:4350, Cost:21.28426371687773
Epoch:4400, Cost:21.191251593747484
Epoch:4450, Cost:21.099612113666833
Epoch:4500, Cost:21.009314161698416



EPOCHS VS COST

Epoch:4550, Cost:20.92032758424601
Epoch:4600, Cost:20.83262315055629
Epoch:4650, Cost:20.746172516166578
Epoch:4700, Cost:20.66094818817721
Epoch:4750, Cost:20.57692349223657
Epoch:4800, Cost:20.494072541134045
Epoch:4850, Cost:20.412370204905628
Epoch:4900, Cost:20.331792082361762
Epoch:4950, Cost:20.252314473954797

[-1.6617861 -1.57836576 2.40223979 1.9139939]

THOERY:
Logistic regression is a statistical method used for modeling the probability of a binary outcome, which can take one of two possible values (typically 0 or 1).
Logistic regression predicts the probability that a given instance belongs to a particular category.
It is widely used in various fields, such as medicine, biology, social sciences, and machine learning.

1. Sigmoid (Logistic) Function:
- The logistic regression model uses the logistic function (or sigmoid function) to transform a linear combination of input features into a probability between 0 and 1.
- The sigmoid function is defined as:
$$ P(Y=1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1X_1 + \beta_2X_2 + \ldots + \beta_kX_k)}} $$
- Here, $P(Y=1)$ is the probability that the dependent variable Y is equal to 1, given the values of the independent variables $X_1, X_2, \ldots, X_k$.
- $\beta_0, \beta_1, \ldots, \beta_k$ are the coefficients to be estimated.

2. Log Odds (Logit) Transformation:
- The logistic function is often expressed in terms of log odds (logit transformation), making the relationship linear in terms of the log odds.
- The log odds of the probability $P(Y=1)$ is given by:
$$ \text{logit}(P(Y=1)) = \log\left(\frac{P(Y=1)}{1 - P(Y=1)}\right) = \beta_0 + \beta_1X_1 + \beta_2X_2 + \ldots + \beta_kX_k $$

3.Maximum Likelihood Estimation (MLE):
- Logistic regression estimates the coefficients (β) by maximizing the likelihood function. The goal is to find the set of parameters that maximizes the likelihood of observing the given data.
- MLE is commonly used because it provides unbiased and efficient parameter estimates.

4.Interpretation of Coefficients:
- The coefficients (β) in logistic regression represent the change in the log odds of the outcome for a one-unit change in the corresponding independent variable, holding other variables constant.
- The sign and magnitude of the coefficients indicate the direction and strength of the relationship.

5. Threshold and Decision Boundary:

- A threshold is set (commonly 0.5) to classify the predicted probabilities into binary outcomes (0 or 1).
- The decision boundary is the point where the logistic function equals the threshold.

6. Assumptions and Considerations:

- Independence of observations is assumed.
- Linearity in log odds is assumed.
- Absence of perfect multicollinearity is important.
- The dependent variable should have a binary distribution.

7. Applications:

- Logistic regression is used for binary classification tasks, such as spam detection, disease diagnosis, and customer churn prediction.
- Extensions like multinomial logistic regression handle multiple categories.

PROGRAM - 10

Write a Python Program to implement a decision tree on a given data set

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
from pprint import pprint

sns.set_style("darkgrid")
df = pd.read_csv("/Users/mukulhooda/Desktop/College/3rd Year/Machine Learning-1/Lab File/Programs/Iris - Iris.csv")

df = df.drop("Id", axis=1)
df = df.rename(columns={"species": "label"})
df.head()


def train_test_split(df, test_size):
    if isinstance(test_size, float):
        test_size = round(test_size * len(df))
    indices = df.index.tolist()
    test_indices = random.sample(population=indices, k=test_size)
    test_df = df.loc[test_indices]
    train_df = df.drop(test_indices)
    return train_df, test_df

def check_purity(data):
    label_column = data[:, -1]
    unique_classes = np.unique(label_column)
    if len(unique_classes) == 1:
        return True
    else:
        return False


def classify_data(data):
    label_column = data[:, -1]
            unique_classes, counts_unique_classes = np.unique(label_column, return_counts=True)
```

```
    index = counts_unique_classes.argmax()
    classification = unique_classes[index]
    return classification


def get_potential_splits(data):
    potential_splits = {}
    _, n_columns = data.shape
    for column_index in range(n_columns - 1):
        potential_splits[column_index] = []
        values = data[:, column_index]
        unique_values = np.unique(values)
        for index in range(len(unique_values)):
            if index != 0:
                current_value = unique_values[index]
                previous_value = unique_values[index - 1]
                potential_split = (current_value + previous_value) / 2
                potential_splits[column_index].append(potential_split)
    return potential_splits


def split_data(data, split_column, split_value):
    split_column_values = data[:, split_column]
    data_below = data[split_column_values <= split_value]
    data_above = data[split_column_values > split_value]
    return data_below, data_above


def calculate_entropy(data):
    label_column = data[:, -1]
    _, counts = np.unique(label_column, return_counts=True)
    probabilities = counts / counts.sum()
    entropy = sum(probabilities * -np.log2(probabilities))
    return entropy


def calculate_overall_entropy(data_below, data_above):
    n = len(data_below) + len(data_above)
    p_data_below = len(data_below) / n
    p_data_above = len(data_above) / n
    overall_entropy = (p_data_below * calculate_entropy(data_below)
                       + p_data_above * calculate_entropy(data_above))
    return overall_entropy
```

```
def determine_best_split(data, potential_splits):
    overall_entropy = 9999
    for column_index in potential_splits:
        for value in potential_splits[column_index]:
                data_below, data_above = split_data(data, split_column=column_index,
split_value=value)
            current_overall_entropy = calculate_overall_entropy(data_below, data_above)
            if current_overall_entropy <= overall_entropy:
                overall_entropy = current_overall_entropy
                best_split_column = column_index
                best_split_value = value

    return best_split_column, best_split_value


sub_tree = {"question": ["yes_answer",
                         "no_answer"]}
example_tree = {"petal_width <= 0.8": ["Iris-setosa",
                                       {"petal_width <= 1.65": [{"petal_length <= 4.9": ["Iris-
versicolor",
                                                                 "Iris-virginica"]},
                                                                "Iris-virginica"]}]}


def decision_tree_algorithm(df, counter=0, min_samples=2, max_depth=5):
    # data preparations
    if counter == 0:
        global COLUMN_HEADERS
        COLUMN_HEADERS = df.columns
        data = df.values
    else:
        data = df
        # base cases
    if (check_purity(data)) or (len(data) < min_samples) or (counter == max_depth):
        classification = classify_data(data)
        return classification
    # recursive part
    else:
        counter += 1
        # helper functions
        potential_splits = get_potential_splits(data)
        split_column, split_value = determine_best_split(data, potential_splits)
        data_below, data_above = split_data(data, split_column, split_value)

        # instantiate sub-tree
```

```
        feature_name = COLUMN_HEADERS[split_column]
        question = "{} <= {}".format(feature_name, split_value)
        sub_tree = {question: []}

        # find answers (recursion)
            yes_answer = decision_tree_algorithm(data_below, counter, min_samples,
max_depth)
             no_answer = decision_tree_algorithm(data_above, counter, min_samples,
max_depth)

        # If the answers are the same, then there is no point in asking the question.
        # This could happen when the data is classified even though it is not pure
        # yet (min_samples or max_depth base cases).
        if yes_answer == no_answer:
            sub_tree = yes_answer
        else:
            sub_tree[question].append(yes_answer)
            sub_tree[question].append(no_answer)
        return sub_tree


def classify_example(example, tree):
    question = list(tree.keys())[0]
    feature_name, comparison_operator, value = question.split()

    # ask question
    if example[feature_name] <= float(value):
        answer = tree[question][0]
    else:
        answer = tree[question][1]

    # base case
    if not isinstance(answer, dict):
        return answer

    # recursive part
    else:
        residual_tree = answer
        return classify_example(example, residual_tree)


def calculate_accuracy(df, tree):
    df["classification"] = df.apply(classify_example, axis=1, args=(tree,))
    df["classification_correct"] = df["classification"] == df["label"]
    accuracy = df["classification_correct"].mean()
```

```
    return accuracy


train_df, test_df = train_test_split(df, test_size=20)
tree = decision_tree_algorithm(train_df, max_depth=3)
accuracy = calculate_accuracy(test_df, tree)
print()
pprint(tree)
print(accuracy)
```

File - Program 10

```
"/Users/mukulhooda/Desktop/College/3rd Year/Machine Learning-1/Lab File
/Programs/venv/bin/python" /Users/mukulhooda/Desktop/College/3rd Year/
Machine Learning-1/Lab File/Programs/Program 10.py

{'petal_width <= 0.8': ['Iris-setosa',
                        {'petal_width <= 1.75': [{'petal_length <= 4.95': ['Iris-versicolor',
                                                                           'Iris-virginica']},
                                                 'Iris-virginica']}]}
1.0

Process finished with exit code 0
```

THEORY:

Decision trees are a popular machine learning algorithm used for both classification and regression tasks.
They are a type of supervised learning algorithm that works by recursively partitioning the input space into regions and assigning a label or value to each region.

Decision trees are particularly useful for their interpretability and ease of visualization.

1. Tree Structure:
   - A decision tree is a tree-like structure where each internal node represents a decision based on a specific feature or attribute.
   - The edges or branches represent the outcome of the decision, leading to further nodes or leaves.
   - The leaves of the tree contain the final predicted label or value.

2. Node Splitting:
- The process of creating a decision tree involves recursively splitting nodes based on features that result in the best separation of data according to the target variable.
- The goal is to minimize impurity or maximize homogeneity within the resulting subsets.

3. Impurity Measures:
- Common impurity measures used in decision trees include Gini impurity, entropy, and mean squared error, depending on whether the task is classification or regression.
- Gini impurity measures the probability of misclassifying a randomly chosen element, while entropy measures the level of disorder in a set.

4. Decision Criteria:
- At each internal node, a decision tree uses a specific feature and a threshold to make decisions about the data.
- For example, in a binary classification task, a decision node might ask if a certain feature is greater than a threshold.

5. Stopping Criteria:
- The tree-building process continues until a stopping criterion is met, such as a specified maximum depth, a minimum number of samples in a leaf, or when further splits do not significantly improve the model.

6. Pruning:
- Pruning is a technique used to prevent overfitting by removing branches from the tree that do not contribute significantly to the predictive accuracy.
- Pruning can be based on cross-validation or other metrics.

7. Advantages:
- Decision trees are easy to understand and interpret, making them suitable for both technical and non-technical audiences.
- They can handle both numerical and categorical data.
- Decision trees implicitly perform feature selection by selecting the most informative features for splitting.

8. Disadvantages:
- Decision trees can be prone to overfitting, especially when the tree becomes too deep.
- They may not capture complex relationships in the data as well as more sophisticated algorithms.
- They can be sensitive to small variations in the data.

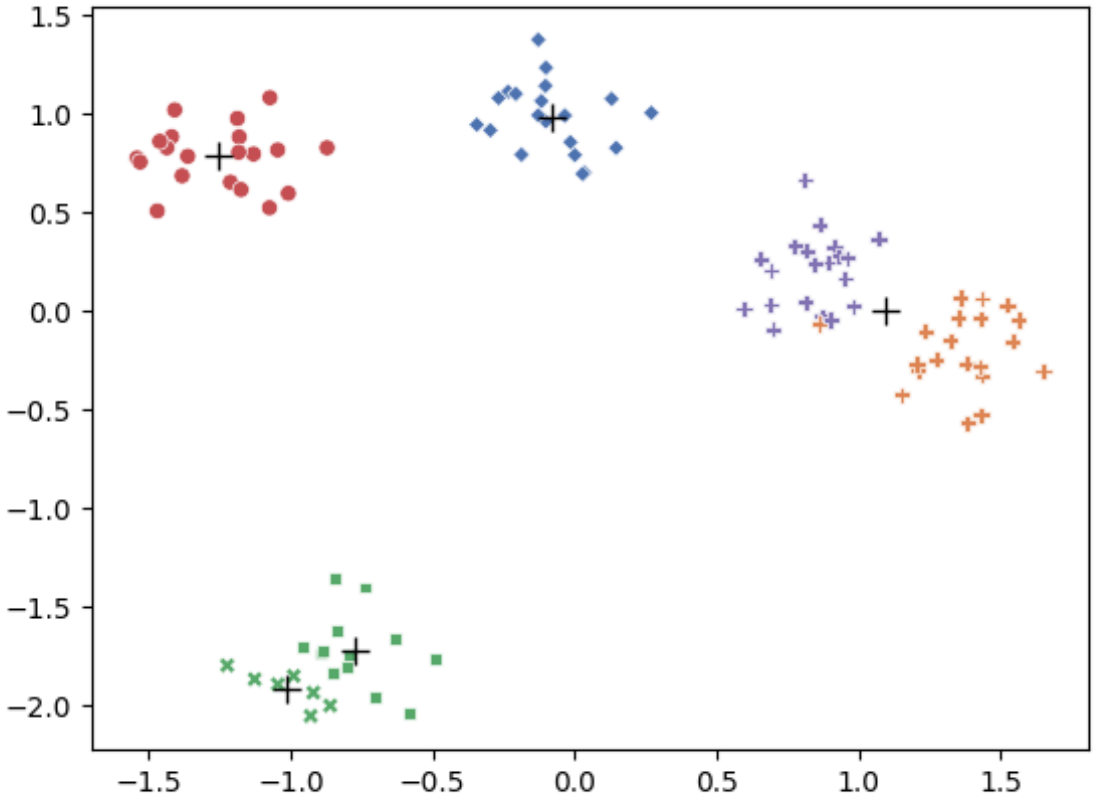
PROGRAM - 11

Write a Python Program to implement K-means clustering algorithm.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from numpy.random import uniform
from sklearn.datasets import make_blobs
import seaborn as sns
import random


def euclidean(point, data):
    return np.sqrt(np.sum((point - data) ** 2, axis=1))


class KMeans:
    def __init__(self, n_clusters=8, max_iter=300):
        self.n_clusters = n_clusters
        self.max_iter = max_iter

    def fit(self, X_train):
            # Initialize the centroids, using the "k-means++" method, where a random datapoint is selected as the first,
            # then the rest are initialized w/ probabilities proportional to their distances to the first
        # Pick a random point from train data for first centroid
        self.centroids = [random.choice(X_train)]
        for _ in range(self.n_clusters - 1):
            # Calculate distances from points to the centroids
                dists = np.sum([euclidean(centroid, X_train) for centroid in self.centroids], axis=0)
            # Normalize the distances
            dists /= np.sum(dists)
            # Choose remaining points based on their distances
            new_centroid_idx, = np.random.choice(range(len(X_train)), size=1, p=dists)
            self.centroids += [X_train[new_centroid_idx]]
        # This initial method of randomly selecting centroid starts is less effective
        # min_, max_ = np.min(X_train, axis=0), np.max(X_train, axis=0)
        # self.centroids = [uniform(min_, max_) for _ in range(self.n_clusters)]
        # Iterate, adjusting centroids until converged or until passed max_iter
        iteration = 0
        prev_centroids = None
```

```
            while np.not_equal(self.centroids, prev_centroids).any() and iteration < self.max_iter:
        # Sort each datapoint, assigning to nearest centroid
        sorted_points = [[] for _ in range(self.n_clusters)]
        for x in X_train:
            dists = euclidean(x, self.centroids)
            centroid_idx = np.argmin(dists)
            sorted_points[centroid_idx].append(x)
         # Push current centroids to previous, reassign centroids as mean of the points belonging to them
        prev_centroids = self.centroids
        self.centroids = [np.mean(cluster, axis=0) for cluster in sorted_points]
        for i, centroid in enumerate(self.centroids):
             if np.isnan(centroid).any():  # Catch any np.nans, resulting from a centroid having no points
                 self.centroids[i] = prev_centroids[i]
        iteration += 1

    def evaluate(self, X):
        centroids = []
        centroid_idxs = []
        for x in X:
            dists = euclidean(x, self.centroids)
            centroid_idx = np.argmin(dists)
            centroids.append(self.centroids[centroid_idx])
            centroid_idxs.append(centroid_idx)
        return centroids, centroid_idxs


# Create a dataset of 2D distributions
centers = 5
X_train, true_labels = make_blobs(n_samples=100, centers=centers, random_state=42)
X_train = StandardScaler().fit_transform(X_train)

# Fit centroids to dataset
kmeans = KMeans(n_clusters=centers)
kmeans.fit(X_train)

# View results
class_centers, classification = kmeans.evaluate(X_train)
sns.scatterplot(x=[X[0] for X in X_train],
                y=[X[1] for X in X_train],
                hue=true_labels,
                style=classification,
```

```
            palette="deep",
            legend=None
            )
plt.plot([x for x, _ in kmeans.centroids],
        [y for _, y in kmeans.centroids],
        'k+',
        markersize=10,
        )
plt.show()
```



THEORY:

K-Means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into groups, or clusters, based on similarity.

The objective is to group data points that are more similar to each other while being dissimilar to those in other clusters.

The algorithm is called "K-Means" because it involves dividing the data into K clusters.

1. Initialization:
   - Choose the number of clusters (\(K\)) that you want to identify in the data.
   - Randomly initialize the centroids of the \(K\) clusters.

2. Assignment:
   - Assign each data point to the cluster whose centroid is the closest in terms of Euclidean distance. The Euclidean distance between two points \(x\) and \(y\) in an \(n\)-dimensional space is given by:
   \[\text{distance}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \]

3. Update:
   - Recalculate the centroids of each cluster by taking the mean of all the data points assigned to that cluster.

4. Repeat:
   - Repeat the assignment and update steps until convergence. Convergence occurs when the centroids no longer change significantly between iterations.

The algorithm aims to minimize the within-cluster sum of squares, also known as inertia or distortion. Inertia is the sum of squared distances between each data point and its assigned cluster's centroid.

Key Points:
- K-Means is sensitive to the initial placement of centroids. Different initializations can result in different final clusters.

- The algorithm can be computationally efficient, but the number of clusters (\(K\)) needs to be specified a priori, which can be a challenge.

- K-Means assumes that clusters are spherical and equally sized, which may not be suitable for all types of data.

Applications:
- Customer segmentation in marketing.
- Anomaly detection by identifying data points that deviate significantly from the cluster centroids.

PROGRAM - 12

Write a program to generate Stem and leaf plot

```
import random

def generate_random_data(size, decimal_places=0):
    return [round(random.uniform(0, 100), decimal_places) for _ in range(size)]

def stem_and_leaf(data, decimal_places=0, sort_data=True):
    if decimal_places < 0:
        raise ValueError("Decimal places must be a non-negative integer.")

    if sort_data:
        data = sorted(data)

    stems = [int(x * 10 ** decimal_places // 10) for x in data]
    leaves = [int(x * 10 ** decimal_places % 10) for x in data]

    unique_stems = sorted(set(stems))

    stem_and_leaf_dict = {stem: [] for stem in unique_stems}

    for i in range(len(data)):
        stem_and_leaf_dict[stems[i]].append(leaves[i])

    for stem in unique_stems:
        leaves_str = ' '.join(map(str, sorted(stem_and_leaf_dict[stem])))
        print(f"{stem / 10 ** decimal_places: .{decimal_places}f} | {leaves_str}")


# Example usage with randomly generated data:
random_data = generate_random_data(15, decimal_places=2)
print()
# print("Random Data:", random_data)
for i in range(len(random_data)):
    print(random_data[i],end=" , ")

print("\nStem-and-Leaf Plot:")
stem_and_leaf(random_data, decimal_places=2)
```

"/Users/mukulhooda/Desktop/College/3rd Year/Machine Learning-1/Lab File /Programs/venv/bin/python" /Users/mukulhooda/Desktop/College/3rd Year/ Machine Learning-1/Lab File/Programs/Program 12.py

54.38 , 49.6 , 65.31 , 45.02 , 7.47 , 10.88 , 2.62 , 8.47 , 67.68 , 52.37 , 76.08 , 53. 79 , 93.48 , 63.49 , 39.74 ,
Stem-and-Leaf Plot:
0.26 | 2
0.74 | 7
0.84 | 7
1.08 | 8
3.97 | 4
4.50 | 2
4.96 | 0
5.23 | 7
5.37 | 9
5.43 | 8
6.34 | 9
6.53 | 1
6.76 | 8
7.60 | 8
9.34 | 8

Process finished with exit code 0

PROGRAM - 13

Write a program for ANOVA test

```
import numpy as np
import pandas as pd
def calculate_mean(data):
    return sum(data) / len(data)

def calculate_total_mean(data):
    total_sum = 0
    total_len = 0
    for group in data:
        total_sum += sum(group)
        total_len += len(group)
    return total_sum / total_len

def calculate_sum_of_squares(data):
    total_mean = calculate_total_mean(data)
    ss_total = 0
    for group in data:
        group_mean = calculate_mean(group)
        ss_group = sum((x - group_mean) ** 2 for x in group)
        ss_total += ss_group
    return ss_total, total_mean

def calculate_between_group_ss(data):
    ss_total, total_mean = calculate_sum_of_squares(data)
    ss_between = 0
    for group in data:
        group_mean = calculate_mean(group)
        ss_between += len(group) * ((group_mean - total_mean) ** 2)
    return ss_between

def calculate_within_group_ss(data):
    ss_total, _ = calculate_sum_of_squares(data)
    ss_between = calculate_between_group_ss(data)
    ss_within = ss_total - ss_between
    return ss_within

def calculate_f_statistic(data):
    k = len(data)
    n = sum(len(group) for group in data)
    df_between = k - 1
```

```
    df_within = n - k

    ss_between = calculate_between_group_ss(data)
    ss_within = calculate_within_group_ss(data)

    ms_between = ss_between / df_between
    ms_within = ss_within / df_within

    f_statistic = ms_between / ms_within
    return f_statistic, df_between, df_within

def critical_value(alpha, df_between, df_within):
    from scipy.stats import f
    return f.ppf(1 - alpha, df_between, df_within)

def anova(data, alpha):
    f_statistic, df_between, df_within = calculate_f_statistic(data)
    crit_val = critical_value(alpha, df_between, df_within)
    print(f"F-statistic: {f_statistic}")
    print(f"Critical value: {crit_val}")

    if f_statistic > crit_val:
        print("Reject the null hypothesis: There is a significant difference between group
means.")
    else:
            print("Fail to reject the null hypothesis: There is no significant difference
between group means.")

data = [
 [56, 60, 61, 53, 58],
 [72, 69, 65, 74, 70],
 [63, 59, 67, 66, 64]
    ]

alpha = 0.05

print()
anova(data, alpha)
```

File - Program 13

"/Users/mukulhooda/Desktop/College/3rd Year/Machine Learning-1/Lab File /Programs/venv/bin/python" /Users/mukulhooda/Desktop/College/3rd Year/ Machine Learning-1/Lab File/Programs/Program 13.py

F-statistic: -8.925696594427247
Critical value: 3.8852938346523933
Fail to reject the null hypothesis: There is no significant difference between group means.

Process finished with exit code 0

PROGRAM - 14

Write a program for z-testing and t-testing

```
import math
from scipy.stats import norm
from scipy.stats import t


def z_test(sample1, sample2, alpha=0.05):
    mean1 = sum(sample1) / len(sample1)
    mean2 = sum(sample2) / len(sample2)

    std_dev1 = math.sqrt(sum((x - mean1) ** 2 for x in sample1) / (len(sample1) - 1))
    std_dev2 = math.sqrt(sum((x - mean2) ** 2 for x in sample2) / (len(sample2) - 1))

      pooled_std_dev = math.sqrt((std_dev1 ** 2 / len(sample1)) + (std_dev2 ** 2 /
len(sample2)))

    z_score = (mean1 - mean2) / pooled_std_dev

    # Calculate the critical value for two-tailed test
    critical_value = norm.ppf(1 - alpha / 2)

    # Compare the z-score with the critical value
    if abs(z_score) > critical_value:
        print("Reject the null hypothesis")
    else:
        print("Fail to reject the null hypothesis")

def t_test(sample1, sample2, alpha=0.05):
    mean1 = sum(sample1) / len(sample1)
    mean2 = sum(sample2) / len(sample2)

    std_dev1 = math.sqrt(sum((x - mean1) ** 2 for x in sample1) / (len(sample1) - 1))
    std_dev2 = math.sqrt(sum((x - mean2) ** 2 for x in sample2) / (len(sample2) - 1))

    # Calculate the t-score
     t_score = (mean1 - mean2) / math.sqrt((std_dev1 ** 2 / len(sample1)) + (std_dev2
** 2 / len(sample2)))

    # Degrees of freedom
    df = len(sample1) + len(sample2) - 2
```

```
    # Calculate the critical value for two-tailed test
    critical_value = t.ppf(1 - alpha / 2, df)

    # Compare the t-score with the critical value
    if abs(t_score) > critical_value:
        print("Reject the null hypothesis")
    else:
        print("Fail to reject the null hypothesis")


# Example usage
sample_group1 = [25, 30, 35, 40, 45]
sample_group2 = [20, 25, 30, 35, 40]

print()
print('z-test: ')
z_test(sample_group1, sample_group2)


print()
print('t-test')
t_test(sample_group1, sample_group2)
```

File - Program 14

"/Users/mukulhooda/Desktop/College/3rd Year/Machine Learning-1/Lab File /Programs/venv/bin/python" /Users/mukulhooda/Desktop/College/3rd Year/ Machine Learning-1/Lab File/Programs/Program 14.py

z-test:
Fail to reject the null hypothesis

t-test
Fail to reject the null hypothesis

Process finished with exit code 0