1. **Write a python program to compute Central Tendency Measures: Mean, Median, Mode Measure of Dispersion: Variance, Standard Deviation.**

**Program:**

def calculate\_measures(data):

    print("data: ",data)

    n= len(data)

    mean = sum(data)/n

*#median*

    sorted\_data = sorted(data)

    if n%2==0:

        median = (sorted\_data[n//2 - 1]+sorted\_data[n//2])/2

    else:

        median = sorted\_data[n//2]

*#mode*

    freq = {}

    for value in data:

        if value in freq:

            freq[value]+=1

        else:

            freq[value]=1

    max\_freq = max(freq.values())

    modes = [key for key, val in freq.items() if val==max\_freq]

    mode = modes[0] if len(modes)==1 else "No unique values"

    s=0

    for x in data:

        s=s+(x-mean)\*\*2

    variance = s/len(data)

    std\_dev= variance\*\*0.5

    print(f"Mean   : {mean}")

    print(f"Median : {median}")

    print(f"Mode   : {mode}")

    print("\n--- Dispersion Measures ---")

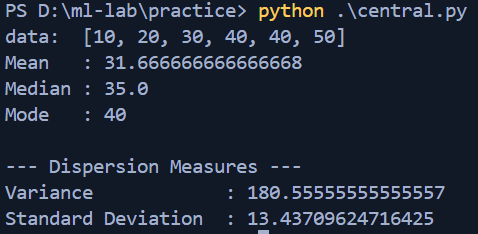
    print(f"Variance            : {variance}")

    print(f"Standard Deviation  : {std\_dev}")

data = [10,20,30,40,40,50]

calculate\_measures(data)

**Output:**



1. **Study of Python basic libraries such as Statistics, Math, NumPy and SciPy.**

**Program:**

import statistics as stat

import math

import numpy as np

from scipy import constants,linalg

def stat\_demo():

    data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,1]

    print("statistics\n")

    print("mean: ",stat.mean(data))

    print("mode:",stat.mode(data))

    print("median:",stat.median(data))

    print("variance:",stat.variance(data))

    print("standard deviation:",stat.stdev(data))

def math\_demo():

    print("\nmath\n")

    number=25

    print(f"Square root of {number} is: {math.sqrt(number)}")

*# Power*

    print(f"{number} raised to power 2 is: {math.pow(number, 2)}")

*# Factorial*

    print(f"Factorial of 5 is: {math.factorial(5)}")

*# GCD*

    print(f"GCD of 48 and 18 is: {math.gcd(48, 18)}")

*# Trigonometry*

    angle = 45  *# degrees*

    radians = math.radians(angle)

    print(f"Cosine of {angle} degrees is: {math.cos(radians)}")

    print(f"Sine of {angle} degrees is: {math.sin(radians)}")

    print(f"Tangent of {angle} degrees is: {math.tan(radians)}")

*# Logarithm*

    print(f"Log base 10 of 1000 is: {math.log10(1000)}")

*# Constants*

    print(f"Value of pi: {math.pi}")

    print(f"Value of e: {math.e}")

def numpy\_demo():

    print("\nNumPy Demo\n")

    arr = np.array([1, 2, 3, 4, 5])

    print("1D Array:", arr)

    print("Shape of array:", arr.shape)

    print("Data type of array:", arr.dtype)

    print("Number of dimensions:", arr.ndim)

    print("Total number of elements:", arr.size)

    print("Sum of all elements:", arr.sum())

    b = np.array([[1, 2], [3, 4]])

    print("\n2D Array:\n", b)

    print("Shape of 2D array:", b.shape)

    print("Element at row 0, column 1 of 2D array:", b[0, 1])

    brr = np.array([8, 9, 10])

    print("\nAnother 1D Array:", brr)

    newarr = np.concatenate((arr, brr))

    print("Concatenated array (arr + brr):", newarr)

    split\_arrays = np.array\_split(newarr, 3)

    print("Array split into 3 parts:")

    for i, part in enumerate(split\_arrays, start=1):

        print(f"  Part {i}: {part}")

def scipy\_demo():

    a= np.array([[1,2],[4,5]])

    inv = linalg.inv(a)

    det = linalg.det(a)

    print("inverse:",inv)

    print("determinant:",det)

*# System of equations: 2x + 3y = 5 and 4x + y = 6*

    A\_eq = np.array([[2, 3], [4, 1]])  *# Coefficient matrix*

    b\_eq = np.array([5, 6])            *# Right-hand side vector*

    values = linalg.solve(A\_eq,b\_eq)

    print(values)

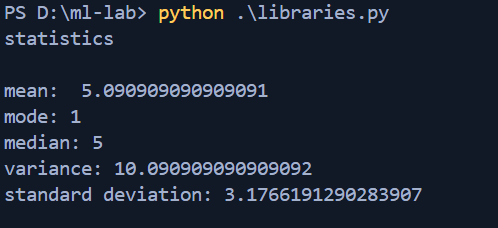
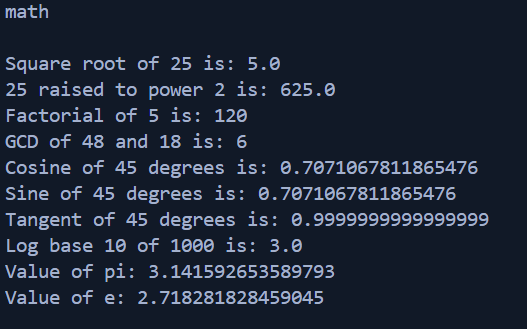
stat\_demo()

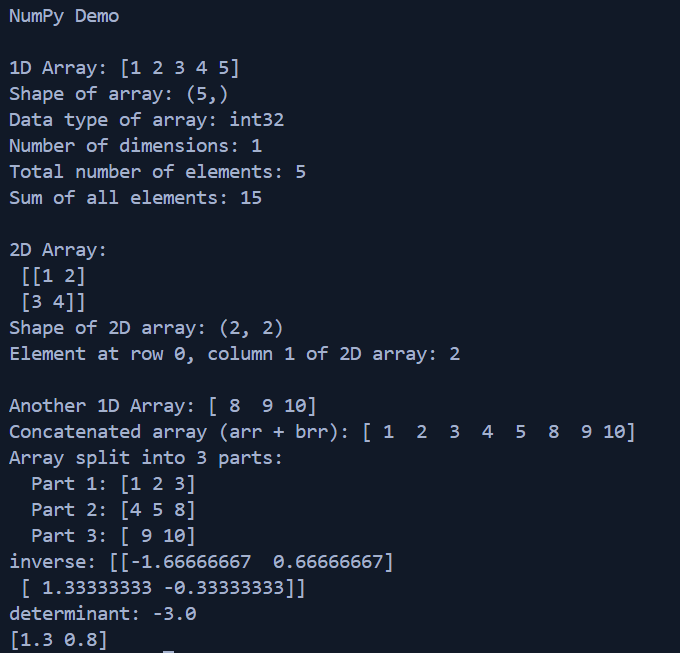
math\_demo()

numpy\_demo()

scipy\_demo()

**Output:**





**Study of python libraries for ML application such as Pandas and Matplotlib.**

**Pandas:**

* **Pandas** is an open-source Python library used for **data analysis, data cleaning, and data manipulation**.
* It provides powerful, flexible, and easy-to-use data structures like:
  + **Series**: One-dimensional labeled array
  + **DataFrame**: Two-dimensional table (like an Excel sheet or SQL table)

**Program:**

import pandas as pd

df = pd.read\_csv('./data/price.csv')

s = pd.Series([1, 2, 3, 4, 5])

obj = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}

series = pd.Series(obj)

print(series)

print(s)

print(df)

print(df.head())

data ={

    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],

    'age': [25, 30, 35, 40, 45],

}

student\_df = pd.DataFrame(data)

print(student\_df['name'])

print(student\_df.loc[0:3])

print(df.tail())

print(df.info())

new\_df = df.dropna() *# drop rows with missing values*

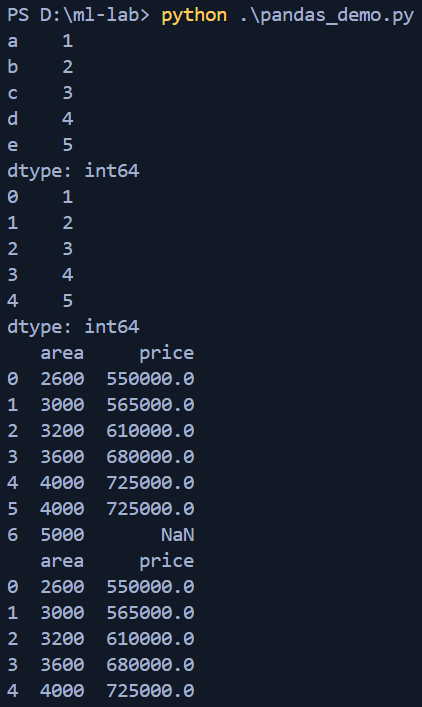
print(new\_df)

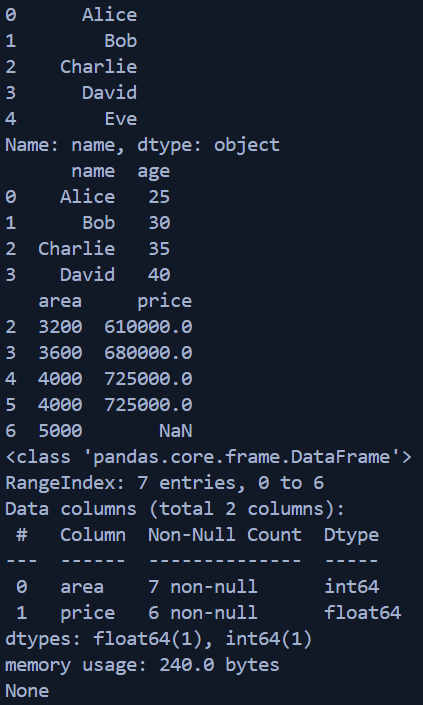
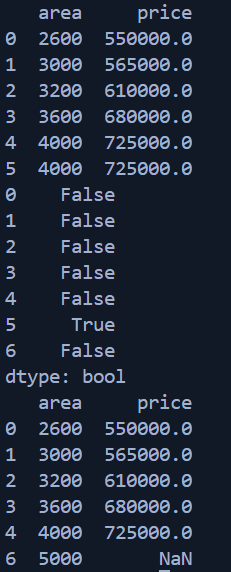
fillna = df.fillna(150000)*#fill with new value*

print(df.duplicated())

print(df.drop\_duplicates())

**Output:**



**Matplotlib:**

* **Matplotlib** is a Python library used for creating static, animated, and interactive visualizations.
* It supports a wide range of plots — including line, bar, scatter, histogram, pie, and more.
* The pyplot module provides a simple interface similar to MATLAB for quick plotting.
* You can customize every element of a plot — such as colors, labels, legends, and gridlines.
* It integrates well with NumPy, Pandas, and Jupyter for data analysis and visualization workflows.
* You can control every part of the plot: colors, line styles, axes, fonts, markers, legends, etc.
* Use plt.subplot() or plt.subplots() to create multiple plots in a grid layout.

**Program:**

import matplotlib.pyplot as plt

import numpy as np

x = np.array([1, 2, 3, 4, 5])

y = np.array([10, 20, 30, 40, 50])

xpoints = np.array([1, 2, 6, 8])

ypoints = np.array([3, 8, 1, 10])

plt.figure("Plot 1: Line with marker and red line")

plt.plot(xpoints, ypoints,'o:r',ms=10) *#ms is marker size*

plt.figure("Plot 2: Subplot 1 - Dashed red line")

plt.subplot(1,2,1) *#1 row 2 columns 1st plot*

plt.plot(xpoints, ypoints,ls='--',color='r') *#ls is line style*

plt.plot(y)

plt.figure("Plot 3: Subplot 2 - Line Plot")

plt.subplot(1,2,2) *#1 row 2 columns 2nd plot*

plt.plot(x, y)

plt.figure("Plot 4: Scatter Plot 1")

plt.scatter(x,y)

plt.figure("Plot 5: Scatter Plot 2")

plt.scatter(xpoints,ypoints,color='g')

plt.figure("Plot 6: Bar Chart")

plt.bar(x,y)

plt.figure("Plot 7: Pie Chart")

xpie = [10,20,30,40]

mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(xpie,labels=mylabels)

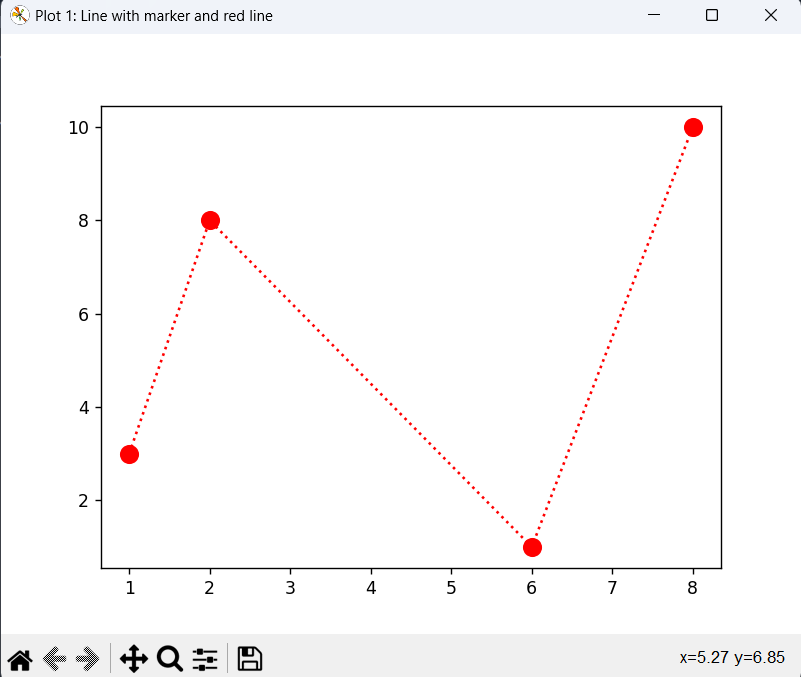
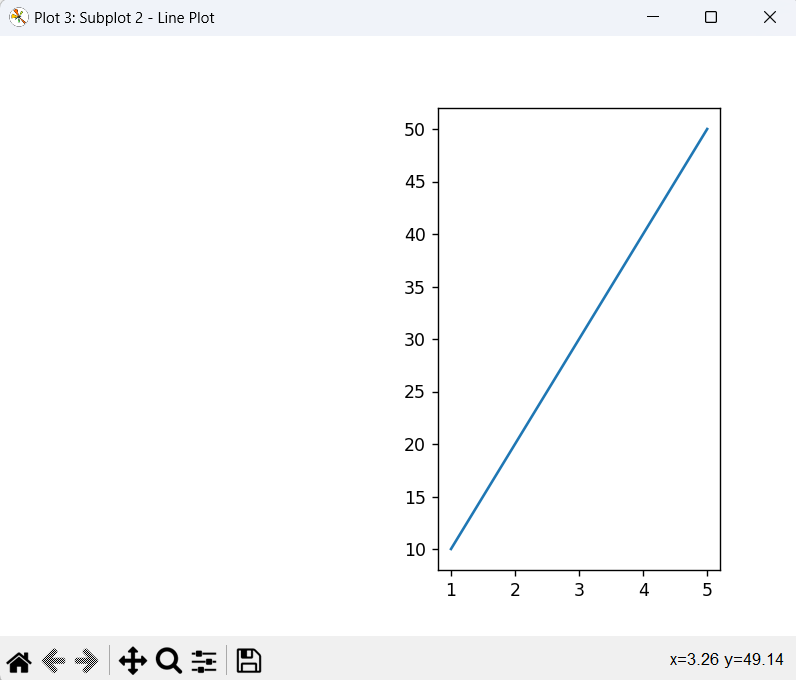
plt.xlabel('X-axis')

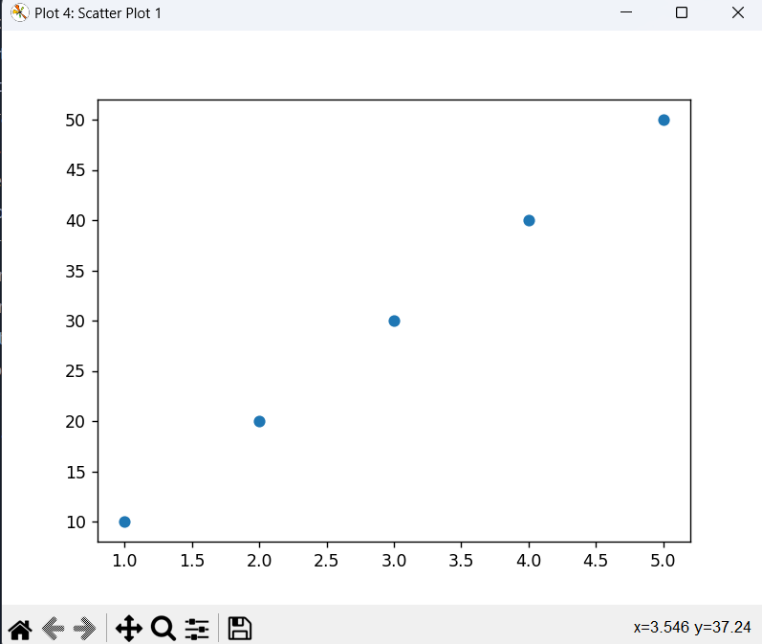
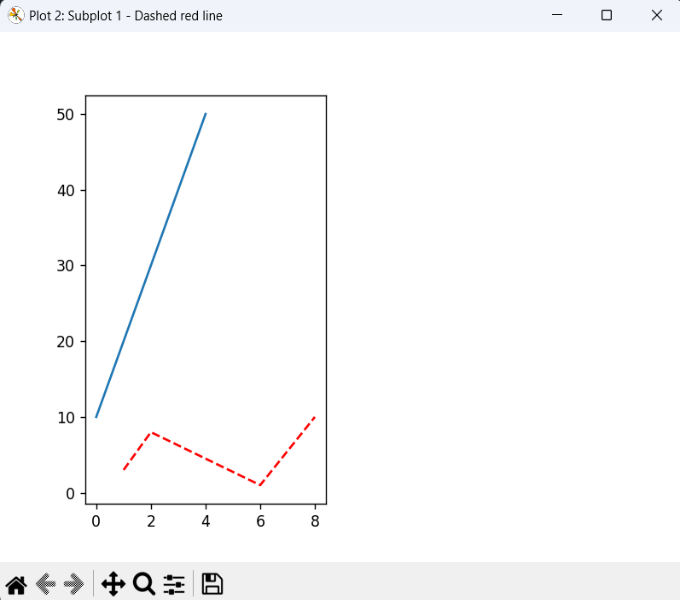
plt.ylabel('Y-axis')

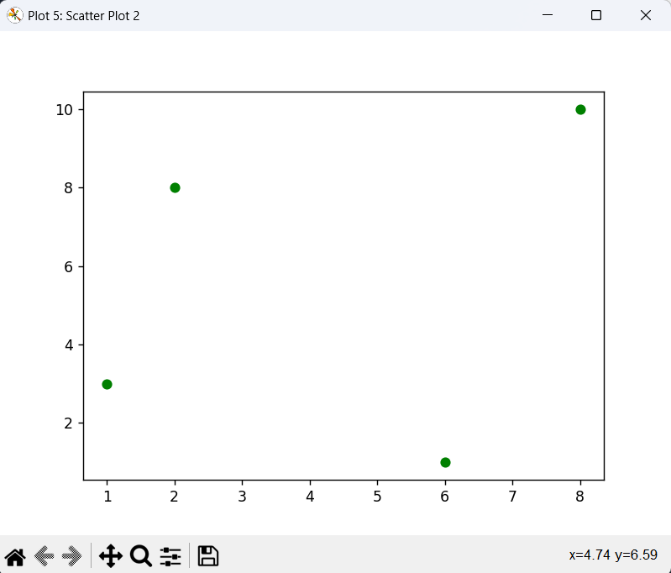
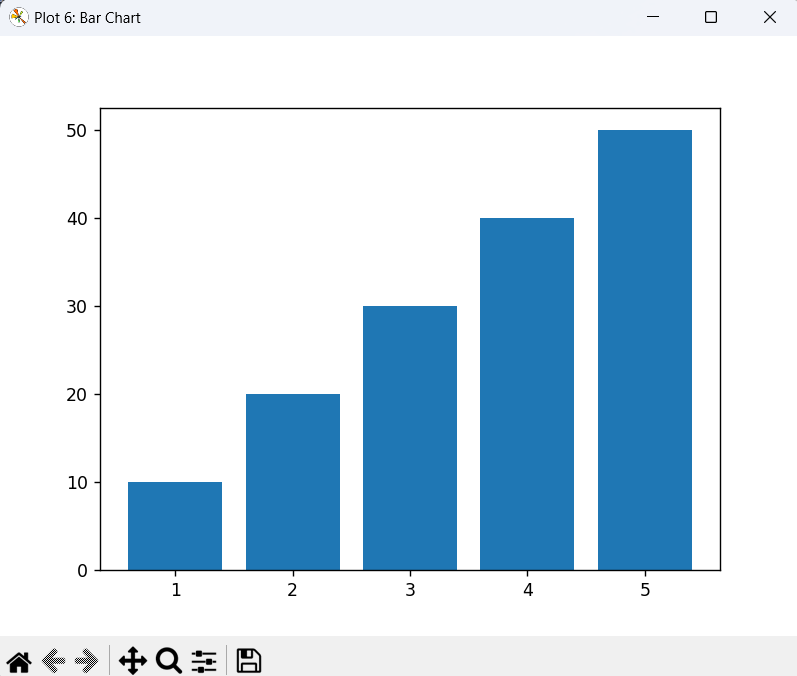
plt.title('Line Plot')

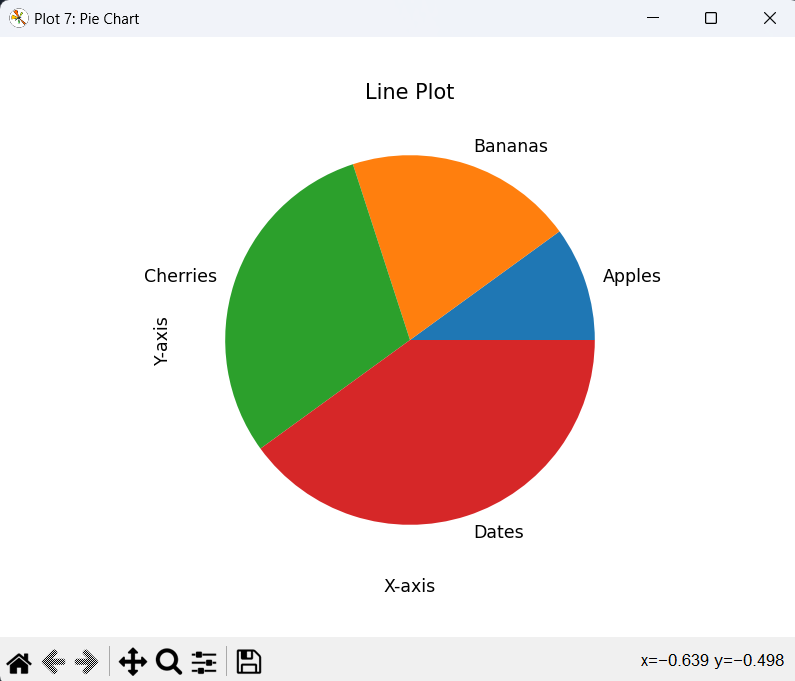
plt.show()

**Output:**







1. **Write a python program to implement simple linear regression.**

**Program:**

import numpy as np

import matplotlib.pyplot as plt

x= np.array([1,2,3,4,5])

y=np.array([30,35,40,45,50])

*# Means*

mean\_x = np.mean(x)

mean\_y = np.mean(y)

*# Initialize numerator and denominator*

numerator = 0

denominator = 0

*# Loop to calculate slope (b1)*

for i in range(len(x)):

    numerator += (x[i] - mean\_x) \* (y[i] - mean\_y)

    denominator += (x[i] - mean\_x) \*\* 2

b1 = numerator / denominator

b0 = mean\_y - b1 \* mean\_x

*# Output the coefficients*

print(f"Intercept (b0): {b0}")

print(f"Slope (b1): {b1}")

*# Predicting line*

y\_pred = b0 + b1 \* x

print(y\_pred)

*# Plotting*

plt.scatter(x, y, color='blue', label='Actual data')

plt.plot(x, y\_pred, color='red', label='Regression line')

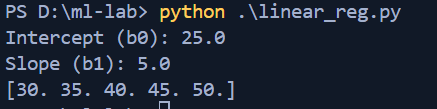
plt.xlabel("Square Feet")

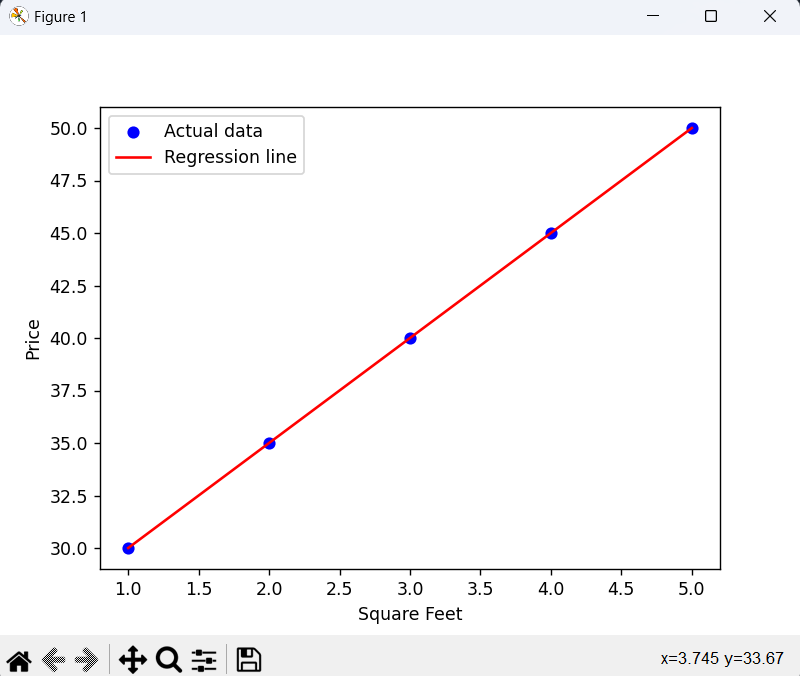
plt.ylabel("Price")

plt.legend()

plt.show()

**Output:**





1. **Implementation of Multiple Linear Regression for House Price Prediction using sklearn.**

**Data: price.csv**area,price

2600,550000

3000,565000

3200,610000

3600,680000

4000,725000

4000,725000

**Program:**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

df = pd.read\_csv("./data/price.csv")

x = df[["area"]]

y = df[["price"]]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=42)

model = LinearRegression()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

residuals = y\_test.values.flatten() - y\_pred.flatten()

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, residuals, alpha=0.7)

plt.axhline(y=0, color='red', linestyle='--')

plt.xlabel("Actual Price")

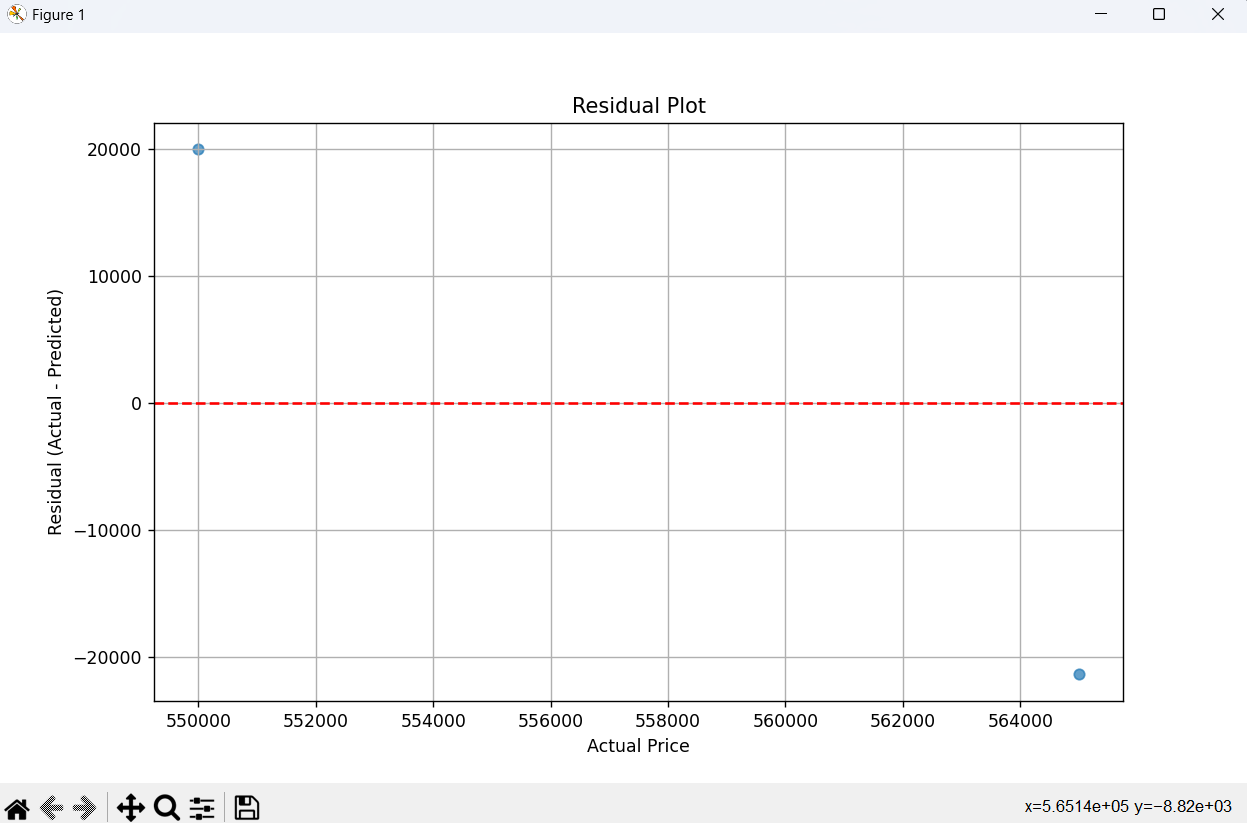
plt.ylabel("Residual (Actual - Predicted)")

plt.title("Residual Plot")

plt.grid(True)

plt.show()

**Output:**



1. **Implementation of decision tree using sklearn and its parameter tuning.**

**Program:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn import tree

X =np.array([

    [25, 40, 600],

    [45, 80, 700],

    [35, 60, 650],

    [50, 90, 720],

    [23, 30, 580],

    [40, 70, 680],

    [29, 50, 610],

])

y = np.array([0, 1, 1, 1, 0, 1, 0])

feature\_names = ["Age", "Income", "Credit Score"]

class\_names = ["No", "Yes"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

clf = DecisionTreeClassifier(random\_state=42)

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

accuracy = clf.score(X\_test, y\_test)

print(f"Accuracy: {accuracy \* 100:.2f}%")

plt.figure(figsize=(6, 6))

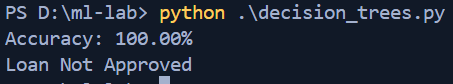
tree.plot\_tree(clf, filled=True, feature\_names=feature\_names, class\_names=class\_names)

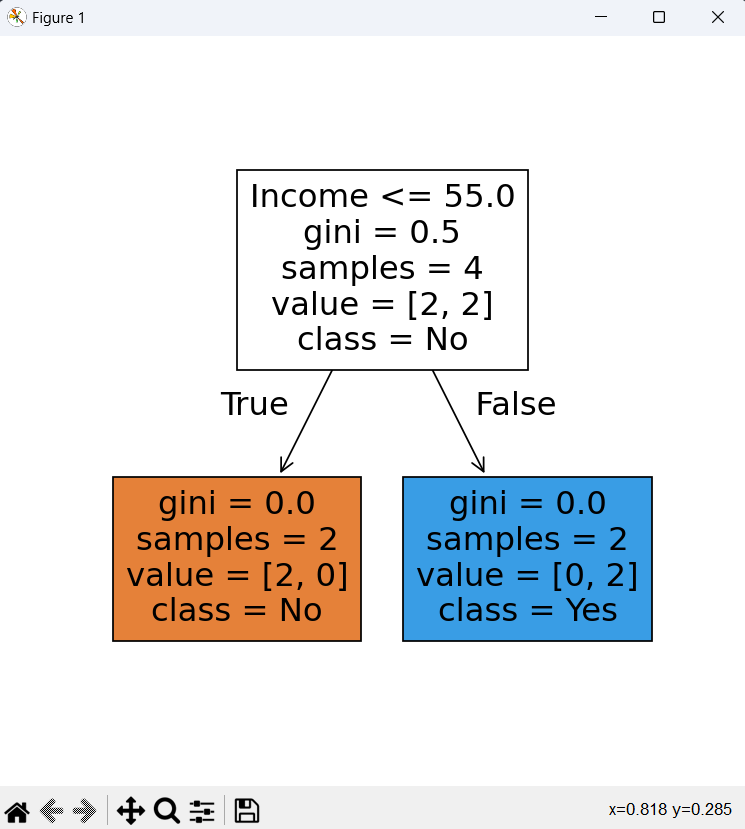
plt.show()

new\_applicant = [[30, 50, 620]]

prediction = clf.predict(new\_applicant)

print("Loan Approved" if prediction[0] == 1 else "Loan Not Approved")

**Output:**

**7.Implementation of KNN using sklearn.**

**Program:**

*# Step 1: Import Libraries*

import numpy as np

import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier

*# 2 classes (Class 0 and Class 1)*

X = np.array([

    [1, 2],

    [2, 3],

    [3, 1],

    [6, 5],

    [7, 7],

    [8, 6]

])

y = np.array([0, 0, 0, 1, 1, 1])   *# first 3 points are class 0, last 3 points are class 1*

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(X, y)

new\_point = np.array([[5, 5]])

prediction = knn.predict(new\_point)

print(f"Predicted class for {new\_point} is: {prediction[0]}")

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', label="Training Points")

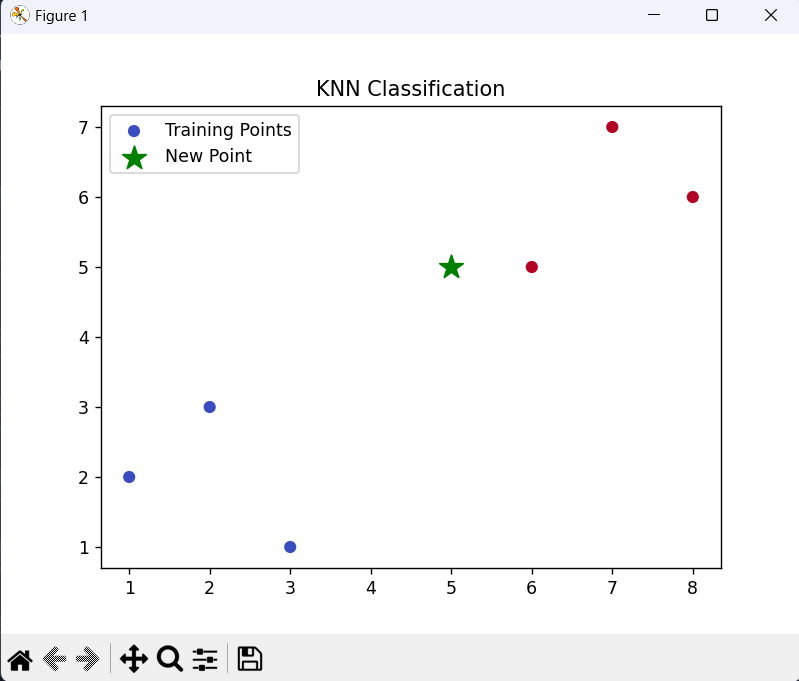
plt.scatter(new\_point[:, 0], new\_point[:, 1], c='green', marker='\*', s=200, label="New Point")

plt.title("KNN Classification")

plt.legend()

plt.show()

**Output:**





**8.Implementation of logistic regression using sklearn.**

**Program:**

import numpy as np

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, classification\_report

X = np.array([

    [1, 2],

    [2, 1],

    [3, 1],

    [4, 3],

    [5, 2],

    [6, 4],

    [7, 3],

    [8, 2],

    [9, 0],

    [10, 1]

])

y = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.3, random\_state=42

)

model = LogisticRegression()

model.fit(X\_train, y\_train)

print(X\_test)

y\_pred = model.predict([[3,8],[1,0],[11,5]])

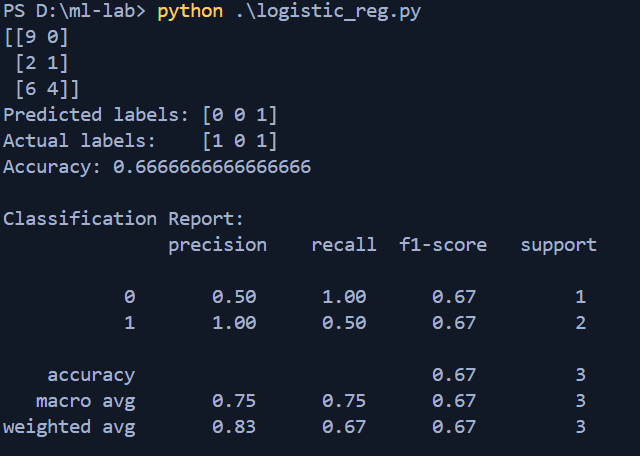
print("Predicted labels:", y\_pred)

print("Actual labels:   ", y\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

**Output:**



**9.Implementation of K-Means Clustering.**

**Program:**

*# Step 1: Import libraries*

from sklearn.cluster import KMeans

import numpy as np

import matplotlib.pyplot as plt

*# height and weight*

X = np.array([

    [160, 60],

    [170, 65],

    [165, 64],

    [155, 58],

    [180, 80],

    [175, 75],

    [185, 90],

    [190, 95],

])

kmeans = KMeans(n\_clusters=2, random\_state=42)

kmeans.fit(X)

labels = kmeans.labels\_      *# Which cluster each point belongs to*

centroids = kmeans.cluster\_centers\_

print("Cluster centers:")

print(centroids)

print("\nLabels:")

print(labels)

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')

plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', s=200, label='Centroids')

plt.xlabel('Height (cm)')

plt.ylabel('Weight (kg)')

plt.title('K-Means Clustering')

plt.legend()

plt.grid(True)

plt.show()

**Output:**

