

**J.N.T.U.H UNIVERSITY COLLEGE OF ENGINEERING
SCIENCE AND TECHNOLOGY HYDERABAD,
KUKATPALLY, HYDERABAD – 500085**



CERTIFICATE

This is to certify that **KOLLURU ANUDEEPIKA** of CSE(Regular) III
year, II Semester bearing with Hall-Ticket number **22011A0538**
has fulfilled her **MACHINE LEARNING LAB** record for the academic year
2024-2025.

Signature of the HOD

Signature of the Staff

Date of Examination :

Internal Examiner

External Examiner

INDEX

S.NO	NAME OF THE PROGRAM	PAGE NO
1	Write a python program to compute Central Tendency Measures: Mean, Median, Mode Measure of Dispersion: Variance, Standard Deviation	1
2	Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy	3
3	Study of Python Libraries for ML application such as Pandas and Matplotlib	5
4	Write a Python program to implement Simple Linear Regression	7
5	Implementation of Multiple Linear Regression for House Price Prediction using sklearn	9
6	Implementation of Decision tree using sklearn and its parameter tuning	11
7	Implementation of KNN using sklearn	13
8	Implementation of Logistic Regression using sklearn	15
9	Implementation of K-Means Clustering	17

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

```
def calculate_mean(data):
    return sum(data) / len(data)

def calculate_median(data):
    sorted_data = sorted(data)
    n = len(sorted_data)
    mid = n // 2
    if n % 2 == 1:
        return sorted_data[mid]
    else:
        return (sorted_data[mid - 1] + sorted_data[mid]) / 2

def calculate_mode(data):
    frequency = {}
    for num in data:
        frequency[num] = frequency.get(num, 0) + 1

    max_freq = max(frequency.values())
    modes = [k for k, v in frequency.items() if v == max_freq]

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

def calculate_variance(data):
    mean = calculate_mean(data)
    n = len(data)
    return sum((x - mean) ** 2 for x in data) / (n - 1) # Sample variance

def calculate_std_dev(data):
    variance = calculate_variance(data)
    return variance ** 0.5

def compute_statistics(data):
    if len(data) == 0:
        print("Data list is empty.")
    return
```

```
mean = calculate_mean(data)
median = calculate_median(data)
mode = calculate_mode(data)
variance = calculate_variance(data)
std_dev = calculate_std_dev(data)

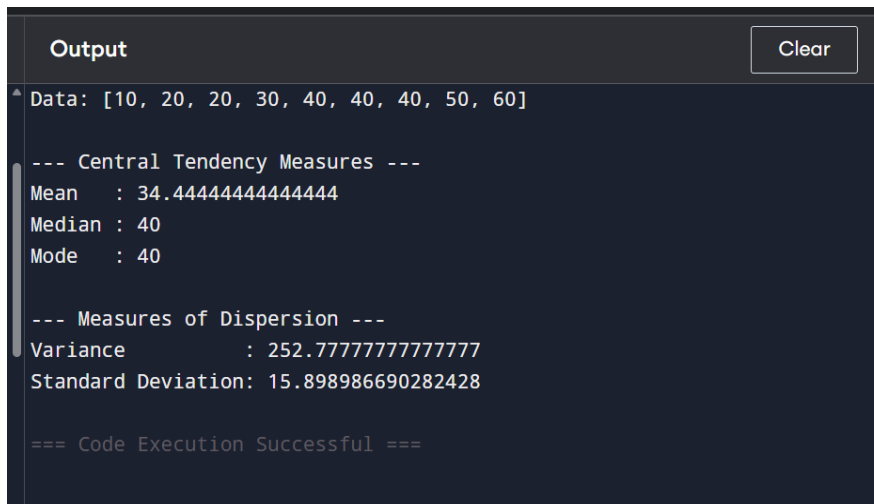
print("Data:", data)
print("\n--- Central Tendency Measures ---")
print(f"Mean : {mean}")
print(f"Median : {median}")
print(f"Mode : {mode}")

print("\n--- Measures of Dispersion ---")
print(f"Variance : {variance}")
print(f"Standard Deviation: {std_dev}")
```

Example usage

```
data = [10, 20, 20, 30, 40, 40, 40, 50, 60]
compute_statistics(data)
```

Output:



```
Output
Data: [10, 20, 20, 30, 40, 40, 40, 50, 60]

--- Central Tendency Measures ---
Mean : 34.44444444444444
Median : 40
Mode : 40

--- Measures of Dispersion ---
Variance : 252.77777777777777
Standard Deviation: 15.898986690282428

=== Code Execution Successful ===
```

2. Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy

Import libraries

import statistics

import math

import numpy as np

from scipy import stats, integrate, optimize

Sample Data

data = [10, 20, 20, 30, 40, 40, 50, 60]

print("=== Using statistics module ===")

print(f"Mean : {statistics.mean(data)}")

print(f"Median : {statistics.median(data)}")

print(f"Mode : {statistics.mode(data)}")

print(f"Variance: {statistics.variance(data)}")

print(f"Standard Deviation: {statistics.stdev(data)}")

print("\n=== Using math module ===")

num = 5

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

print(f"Square root of 25 : {math.sqrt(25)}")

print(f"Power 2^3 : {math.pow(2, 3)}")

print(f"Value of Pi : {math.pi}")

print(f"Sine of 90 degrees : {math.sin(math.radians(90))}")

print("\n=== Using numpy module ===")

np_data = np.array(data)

print(f"Numpy Array : {np_data}")

print(f"Mean using NumPy : {np.mean(np_data)}")

print(f"Standard Deviation : {np.std(np_data)}")

print(f"Variance : {np.var(np_data)}")

print(f"Max value : {np.max(np_data)}")

print(f"Min value : {np.min(np_data)}")

print(f"Array of even numbers : {np.arange(2, 11, 2)}")

print("\n=== Using scipy module ===")

Function: $f(x) = (x + 1)^2$

$f = \text{lambda } x: (x + 1)**2$

Minimize starting from $x = 0$

$\text{result} = \text{optimize.minimize}(f, x0=0)$

$\text{print}(f\text{"Minimum value: \{result.fun\}"})$

$\text{print}(f\text{"At } x = \{\text{result.x}[0]\}"})$

Normal distribution probability density at $x = 0$

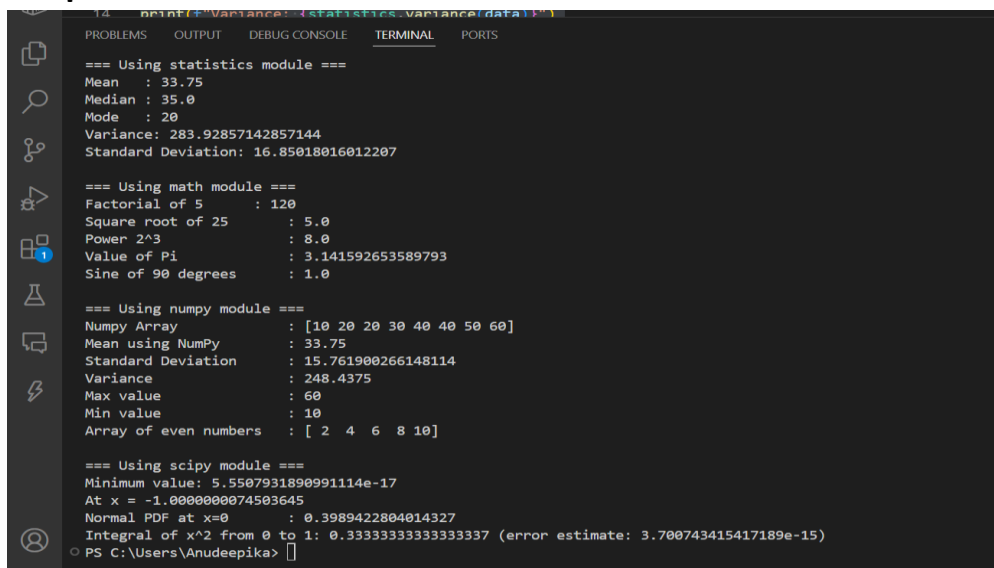
$\text{print}(f\text{"Normal PDF at } x=0 \quad : \{\text{stats.norm.pdf}(0, \text{loc}=0, \text{scale}=1)\}"})$

Integration example: $\int x^2 dx$ from 0 to 1

$\text{result, error} = \text{integrate.quad}(\text{lambda } x: x**2, 0, 1)$

$\text{print}(f\text{"Integral of } x^2 \text{ from 0 to 1: \{result\} (error estimate: \{error\})}"})$

Output:



```

print(f"Variance: {statistics.variance(data)}")

=== Using statistics module ===
Mean      : 33.75
Median    : 35.0
Mode      : 20
Variance   : 283.92857142857144
Standard Deviation: 16.85018016012207

=== Using math module ===
Factorial of 5      : 120
Square root of 25   : 5.0
Power 2^3           : 8.0
Value of Pi         : 3.141592653589793
Sine of 90 degrees  : 1.0

=== Using numpy module ===
Numpy Array        : [10 20 20 30 40 50 60]
Mean using NumPy    : 33.75
Standard Deviation  : 15.761900266148114
Variance            : 248.4375
Max value           : 60
Min value           : 10
Array of even numbers : [ 2  4  6  8 10]

=== Using scipy module ===
Minimum value: 5.5507931890991114e-17
At x = -1.0000000074503645
Normal PDF at x=0      : 0.3989422804014327
Integral of x^2 from 0 to 1: 0.3333333333333333 (error estimate: 3.700743415417189e-15)
PS C:\Users\Anudeepika>

```

3. Study of Python Libraries for ML application such as Pandas and Matplotlib

```

import pandas as pd
import matplotlib.pyplot as plt

# Sample Data (Cleaned)
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [24, 27, 22, 32, 29],
    'Maths': [85, 78, 90, 66, 95],
    'Science': [88, 76, 85, 80, 92],
    'City': ['Delhi', 'Mumbai', 'Delhi', 'Chennai', 'Mumbai']
}

df = pd.DataFrame(data)

# Basic Data Checks
print("=== Head ===")
print(df.head())

print("\n=== Info ===")
df.info()

print("\n=== Describe ===")
print(df.describe())

print("\n=== Rows 1 to 3 ===")
print(df[1:4])

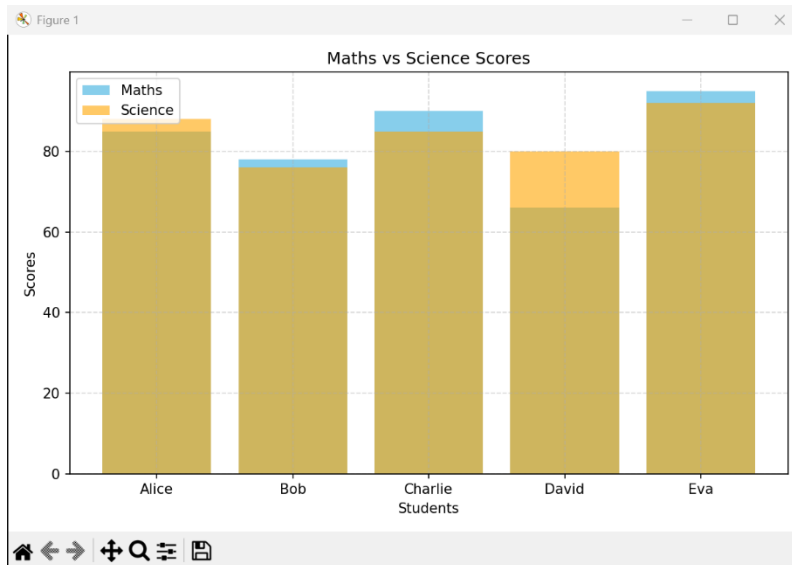
print("\n=== Sorted by Age ===")
print(df.sort_values(by='Age'))

# Bar Chart for Maths vs Science Scores
plt.figure(figsize=(8, 5))
plt.bar(df['Name'], df['Maths'], color='skyblue', label='Maths')
plt.bar(df['Name'], df['Science'], color='orange', alpha=0.6, label='Science')

plt.title("Maths vs Science Scores")
plt.xlabel("Students")
plt.ylabel("Scores")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()

```

plt.show()



```

import pandas as pd

df = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [24, 27, 22, 32, 29],
    'Maths': [85, 78, 90, 66, 95],
    'Science': [88, 76, 85, 80, 92],
    'City': ['Delhi', 'Mumbai', 'Delhi', 'Chennai', 'Mumbai']
})

print(df.head())
print(df.info())
print(df.describe())

print(df[1:4])
print(df.sort_values('Age'))

```

Output of the code:

```

=== Head ===
   Name  Age  Maths  Science  City
0  Alice   24    85     88  Delhi
1    Bob   27    78     76  Mumbai
2 Charlie   22    90     85  Delhi
3  David   32    66     80  Chennai
4    Eva   29    95     92  Mumbai

=== Info ===
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 5 columns):
#   Column  Non-Null Count  Dtype
---  -
0  Name    5 non-null        object
1  Age     5 non-null        int64
2  Maths   5 non-null        int64
3  Science 5 non-null        int64
4  City    5 non-null        object
dtypes: int64(3), object(2)
memory usage: 332.0+ bytes

=== Describe ===
      Age      Maths      Science
count  5.000000  5.000000  5.000000
mean   26.800000  82.800000  84.200000
std     3.962323  11.300442   6.340347
min    22.000000  66.000000  76.000000
25%    24.000000  78.000000  80.000000
50%    27.000000  85.000000  85.000000
75%    29.000000  90.000000  88.000000
max    32.000000  95.000000  92.000000

=== Rows 1 to 3 ===
   Name  Age  Maths  Science  City
1    Bob   27    78     76  Mumbai
2 Charlie   22    90     85  Delhi
3  David   32    66     80  Chennai

=== Sorted by Age ===
   Name  Age  Maths  Science  City
2  Charlie   22    90     85  Delhi
0  Alice    24    85     88  Delhi
1    Bob    27    78     76  Mumbai
4    Eva    29    95     92  Mumbai
3  David    32    66     80  Chennai

```


4. Write a Python program to implement Simple Linear Regression

```

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

n = int(input("Enter the number of data points: "))
X = []
y = []

print("Enter years of experience and salary (in $1000s), space-separated:")

for _ in range(n):
    exp, sal = map(float, input().split())
    X.append([exp])
    y.append(sal)

X = np.array(X)
y = np.array(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

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

y_pred = model.predict(X_test)

print("\nPredictions From test data:")
for i in range(len(X_test)):
    print(f"Experience: {X_test[i][0]} years -> Predicted Salary: ${y_pred[i]:.2f}")

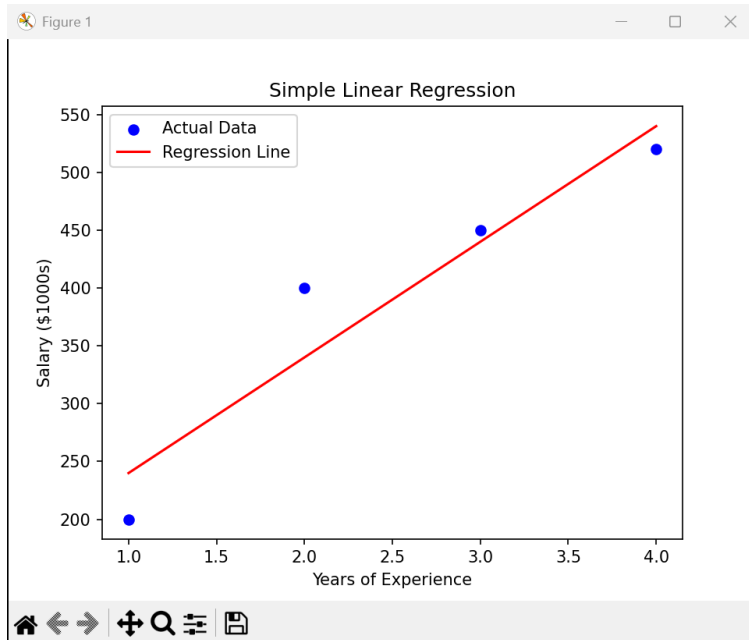
# Predict for dynamic input
new_exp = float(input("\nEnter years of experience to predict salary: "))
pred_salary = model.predict([[new_exp]])
print(f"Predicted Salary for {new_exp} years experience: ${pred_salary[0]:.2f}")

plt.scatter(X, y, color='blue', label='Actual Data')

X_sorted = np.sort(X, axis=0)
plt.plot(X_sorted, model.predict(X_sorted), color='red', label='Regression Line')
plt.xlabel("Years of Experience")

```

```
plt.ylabel("Salary ($1000s)")  
plt.title("Simple Linear Regression")  
plt.legend()  
plt.show()
```

Output:

```
PS C:\Users\Anudeepika\OneDrive\Desktop\Coding\pract> python hi.py  
Enter the number of data points: 4  
Enter years of experience and salary (in $1000s), space-separated:  
1 200  
2 400  
3 450  
4 520  
  
Predictions From test data:  
Experience: 3.0 years -> Predicted Salary: $440.00  
  
Enter years of experience to predict salary: 6  
Predicted Salary for 6.0 years experience: $740.00  
█
```

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

```

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# === Step 1: Training Data ===
X = np.array([
    [1000, 2, 1],
    [1500, 3, 2],
    [2000, 4, 3],
    [1200, 2, 2],
    [1800, 3, 2]
])
y = np.array([150000, 250000, 300000, 180000, 300000])

# === Step 2: Train the Model ===
model = LinearRegression()
model.fit(X, y)

# === Step 3: Predict for Training Data ===
y_pred = model.predict(X)

# === Step 4: Evaluate Model Accuracy ===
mae = mean_absolute_error(y, y_pred)
r2 = r2_score(y, y_pred)

# === Step 4: Predict for New Input ===
print("Enter new house details (Size_sqft Bedrooms Bathrooms):")
size, beds, baths = map(float, input().split())
new_house = np.array([[size, beds, baths]])
predicted_price = model.predict(new_house)

print(f"\nPredicted price for {int(size)} sqft, {int(beds)} bed, {int(baths)} bath house:
${predicted_price[0]:.2f}")

print(f"Mean Absolute Error (MAE): ${mae:.2f}")
print(f"R2 Score: {r2:.2f}")

# === Step 5: Plot Actual vs Predicted Prices ===
labels = [f"House {i+1}" for i in range(len(y))]

```

```

x = np.arange(len(labels))
bar_width = 0.35

plt.figure(figsize=(10, 6))
plt.bar(x - bar_width/2, y, width=bar_width, color='skyblue', label='Actual Price')
plt.bar(x + bar_width/2, y_pred, width=bar_width, color='orange', label='Predicted Price')

plt.xticks(x, labels)
plt.ylabel('Price ($)')
plt.title('Actual vs Predicted House Prices')
plt.legend()
plt.grid(True, axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()

```

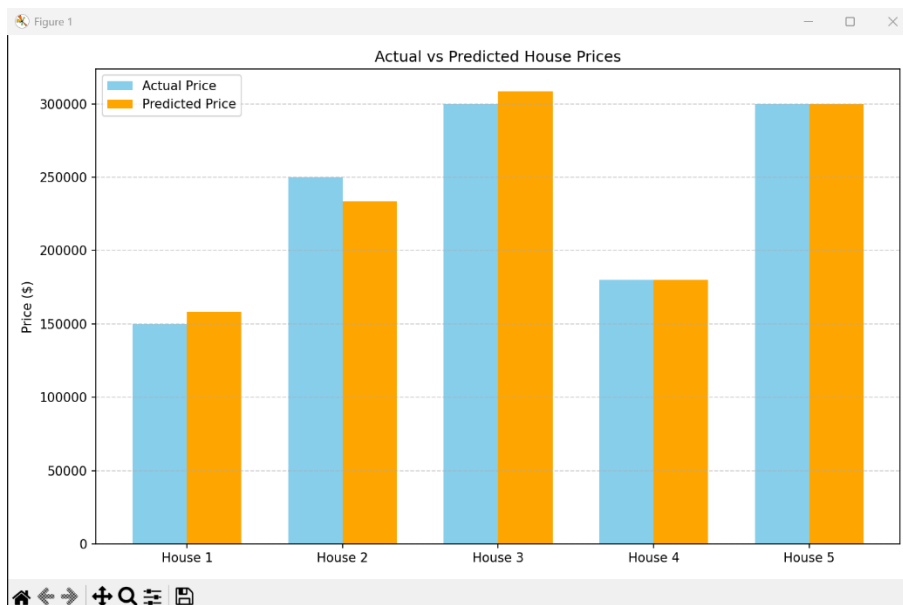
Output:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
1
R² Score: 0.98
PS C:\Users\Anudeepika\OneDrive\Desktop\Coding\pract> python hi.py
Enter new house details (Size_sqft Bedrooms Bathrooms):
1000 4 2

Predicted price for 1000 sqft, 4 bed, 2 bath house: $108888.89
Mean Absolute Error (MAE): $6666.67
R² Score: 0.98

```



6.Implementation of Decision tree using sklearn and its parameter tuning.

```

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

data = {
    'Age':      [22, 25, 28, 30, 34, 36, 40, 42, 44, 48, 50, 52, 55, 58, 60],
    'Salary':   [20000, 25000, 27000, 30000, 35000, 40000, 45000, 50000, 55000, 60000,
70000, 75000, 80000, 85000, 90000],
    'Experience': [1, 2, 3, 4, 5, 6, 7, 10, 11, 13, 14, 15, 16, 18, 20],
    'Education_Level': [1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4], # 1=HS, 2=UG, 3=PG, 4=PhD
    'Buy':       [0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1]
}

df = pd.DataFrame(data)

# 2. Train/Test split
X = df[['Age', 'Salary', 'Experience', 'Education_Level']]
y = df['Buy']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 3. Decision tree training
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)

params = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 4, 5, 6],
    'min_samples_split': [2, 3],
    'min_samples_leaf': [1, 2]
}

grid = GridSearchCV(DecisionTreeClassifier(random_state=42), params, cv=3)
grid.fit(X_train, y_train)
best_dt = grid.best_estimator_

y_pred_best = best_dt.predict(X_test)
print("\nBest Parameters:", grid.best_params_)
print("Accuracy (After Tuning):", accuracy_score(y_test, y_pred_best))

```

```
print("Classification Report (Tuned):\n", classification_report(y_test, y_pred_best))
```

5. Prediction point

```
test_point = pd.DataFrame({'Age': [35], 'Salary': [42000], 'Experience': [6], 'Education_Level': [2]})
```

```
prediction = best_dt.predict(test_point)
```

```
print(f"\nPrediction for point {test_point.values.tolist()[0]}: {'Buy' if prediction[0] == 1 else 'Not Buy'}")
```

6. Plot the decision tree

```
plt.figure(figsize=(14, 8))
```

```
plot_tree(
```

```
    best_dt,
```

```
    feature_names=['Age', 'Salary', 'Experience', 'Education_Level'],
```

```
    class_names=['Not Buy', 'Buy'],
```

```
    filled=True,
```

```
    rounded=True,
```

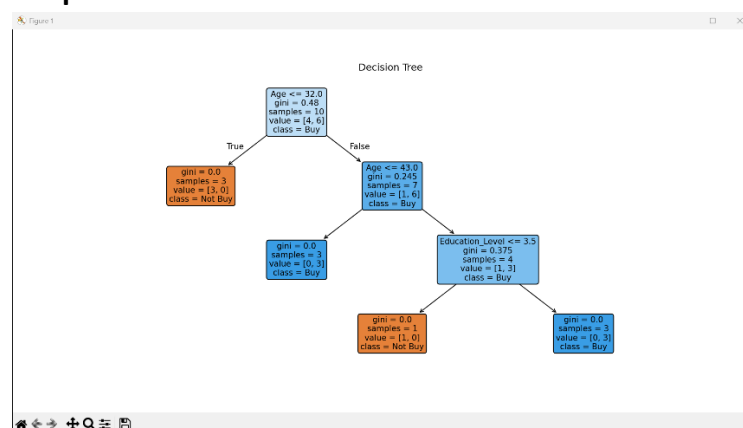
```
    fontsize=10
```

```
)
```

```
plt.title("Decision Tree ")
```

```
plt.show()
```

Output:



```
PS C:\Users\Anudeepika\OneDrive\Desktop\Coding\pract> python hi.py

Best Parameters: {'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}
Accuracy (After Tuning): 0.6
Classification Report (Tuned):
```

	precision	recall	f1-score	support
0	0.50	0.50	0.50	2
1	0.67	0.67	0.67	3
accuracy			0.60	5
macro avg	0.58	0.58	0.58	5
weighted avg	0.60	0.60	0.60	5

```

Prediction for point [35, 42000, 6, 2]: Buy

```

7.Implementation of KNN using sklearn

```

import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# Simple custom dataset: [Height, Weight]
X = np.array([
    [150, 50], [160, 55], [165, 60], [170, 65],
    [175, 70], [180, 75], [185, 80], [190, 85],
    [195, 90], [200, 95]
])
y = np.array([0, 0, 0, 1, 1, 1, 1, 1, 0, 0]) # 0 = Unfit, 1 = Fit

# Input K value
k = int(input("Enter value of k (neighbors): "))

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Train KNN model
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)

# Evaluate
y_pred = model.predict(X_test)
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Predict new point
print("\nEnter new data (Height Weight): ")
new_data = list(map(float, input().split()))
prediction = model.predict([new_data])[0]
print("Predicted class:", "Fit" if prediction == 1 else "Unfit")

# Plot data and prediction
colors = ['red' if label == 0 else 'blue' for label in y]
plt.scatter(X[:, 0], X[:, 1], c=colors, s=100, label='Data Points')
plt.scatter(new_data[0], new_data[1], c='green', s=150, marker='*', label='Your Input')
plt.xlabel("Height")
plt.ylabel("Weight")
plt.title("KNN Classification (Fit vs Unfit)")

```

```
plt.legend()
plt.grid(True)
plt.show()
```

Output:

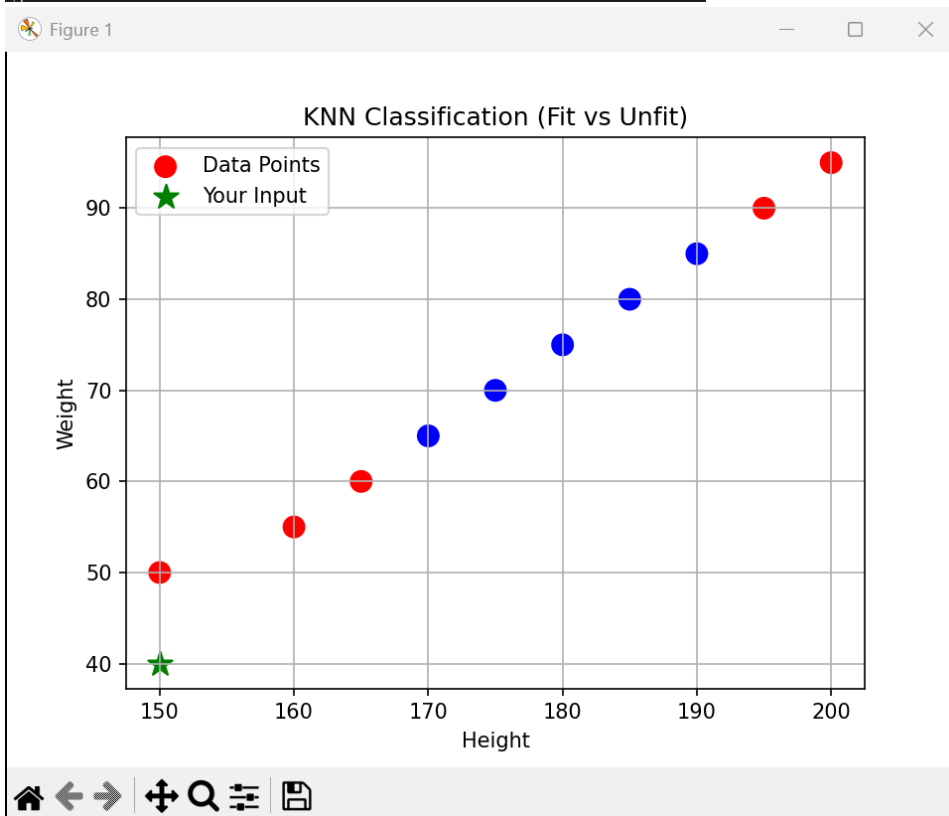
```
PS C:\Users\Anudeepika\OneDrive\Desktop\Coding\pract> python hi.py
Enter value of k (neighbors): 2

Classification Report:
              precision    recall  f1-score   support

     0           1.00       1.00       1.00         2
     1           1.00       1.00       1.00         1

 accuracy          1.00
 macro avg          1.00       1.00       1.00         3
 weighted avg       1.00       1.00       1.00         3

Enter new data (Height Weight):
150 40
Predicted class: Unfit
█
```



8.Implementation of Logistic Regression using sklearn

```

import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import numpy as np

# Custom dataset: Height vs Weight to classify Fitness (0 = Unfit, 1 = Fit)
X = np.array([
    [150, 50],
    [160, 55],
    [170, 65],
    [180, 75],
    [190, 85],
    [200, 95]
])
y = np.array([0, 0, 1, 1, 1, 0])

# Split into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Train logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

print("\nClassification Report:\n", classification_report(y_test, y_pred, zero_division=0))

# Predict a new point
print("\nEnter Height and Weight to classify:")
new_point = list(map(float, input().split()))
pred = model.predict([new_point])[0]
print("Predicted class:", "Fit" if pred == 1 else "Unfit")

# Plot data and prediction point
colors = ['red' if label == 0 else 'blue' for label in y]
plt.scatter(X[:, 0], X[:, 1], c=colors, s=100, label='Data')
plt.scatter(new_point[0], new_point[1], c='green', s=150, marker='*', label='Your Input')
plt.xlabel("Height")
plt.ylabel("Weight")

```

```
plt.title("Logistic Regression (6 Samples)")
plt.legend()
plt.grid(True)
plt.show()
```

Output:

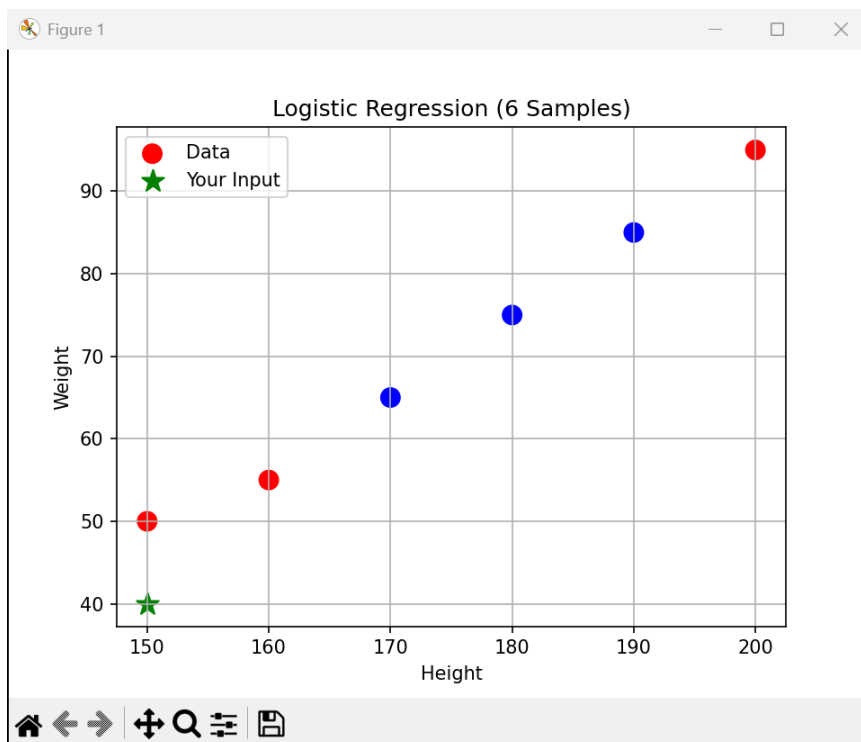
```
PS C:\Users\Anudeepika> cd C:\Users\Anudeepika\OneDrive\Desktop\Coding\pract
PS C:\Users\Anudeepika\OneDrive\Desktop\Coding\pract> python hi.py

Classification Report:
              precision    recall  f1-score   support

     0           0.00        0.00        0.00         1
     1           0.50        1.00        0.67         1

 accuracy          0.50         2
 macro avg          0.25        0.50        0.33         2
 weighted avg       0.25        0.50        0.33         2

Enter Height and Weight to classify:
150 40
Predicted class: Fit
```



9.Implementation of K-Means Clustering

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Custom data: [Height (cm), Weight (kg)]
X = np.array([
    [150, 50],
    [160, 55],
    [165, 60],
    [175, 70],
    [185, 80],
    [195, 90]
])

# Apply KMeans clustering (3 clusters)
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X)

# Get cluster labels and centroids
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

# Function to give meaning to clusters (manual logic)
def label_cluster(centroid):
    h, w = centroid
    if h < 165:
        return "Unfit"
    elif h < 180:
        return "Average"
    else:
        return "Fit"

# Take user input and predict
print("\nEnter Height and Weight to find your cluster:")
user_input = list(map(float, input().split()))
user_cluster = kmeans.predict([user_input])[0]
print(f"You belong to Cluster {user_cluster} → {label_cluster(centroids[user_cluster])}")

# Plotting
colors = ['red', 'green', 'blue']

```

```

plt.figure(figsize=(8, 6))
for i in range(len(X)):
    plt.scatter(X[i][0], X[i][1], color=colors[labels[i]], s=100)

# Plot centroids
for i, c in enumerate(centroids):
    plt.scatter(c[0], c[1], marker='X', color='black', s=200, label=f'Cluster {i}
({label_cluster(c)}')

# Plot user input
plt.scatter(user_input[0], user_input[1], color='purple', marker='*', s=200, label='Your Point')

plt.title("K-Means Clustering (Height vs Weight)")
plt.xlabel("Height (cm)")
plt.ylabel("Weight (kg)")
plt.legend()
plt.grid(True)
plt.show()

```

Output:

```

PS C:\Users\Anudeepika> python -u "c:\Users\Anudeepika\OneDrive\Deskt

Enter Height and Weight to find your cluster:
170 65
You belong to Cluster 0 → Average

```

