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

Code:

```
data = input("Enter numbers separated by spaces: ").split()
data = [float(x) if '.' in x else int(x) for x in data]
sorted_data = sorted(data)
n = len(data)

# Mean
mean = sum(data) / n

#  Median
if n % 2 == 0:
    median = (sorted_data[n//2 - 1] + sorted_data[n//2]) / 2
else:
    median = sorted_data[n//2]

# Mode
frequency = {}
for item in data:
    frequency[item] = frequency.get(item, 0) + 1

max_freq = max(frequency.values())
mode = [key for key, val in frequency.items() if val == max_freq]

if len(mode) == len(frequency):
    mode_result = "No mode"
else:
    mode_result = mode[0] if len(mode) == 1 else mode

#  Variance
mean_diff_squares = [(x - mean) ** 2 for x in data]
variance = sum(mean_diff_squares) / (n - 1)  # Sample variance

# Standard Deviation
std_deviation = variance ** 0.5

print("\nMeasures of Central Tendency:")
print(f"Mean   : {mean}")
print(f"Median : {median}")
print(f"Mode   : {mode_result}")

print("\n Measures of Dispersion: ")
```

```
print(f"Variance          : {variance}")
print(f"Standard Deviation: {std_deviation}")
```

Output:

```
Enter numbers separated by spaces:  2 4 3 5 3 6 7 3 8 2 9 6 3 5 6

Measures of Central Tendency:
Mean   : 4.8
Median : 5
Mode   : 3

 Measures of Dispersion:
Variance          : 4.742857142857143
Standard Deviation: 2.177810171446801
```

2. Write a python program to implement Find S algorithm

Code:

```
#Find S
import pandas as pd

# Load the dataset
dataframe = pd.read_csv('enjoysport.csv')

# Convert dataframe to a list of rows
dataset = dataframe.values.tolist()

def Find_S(dataset):
    # Initialize the hypothesis with '0's (no constraints)
    hypothesis = ['0'] * (len(dataset[0]) - 1)

    # Loop through each row in the dataset
    for row in dataset:
        if row[-1] == 'Yes':  # Focus on rows where the last column is 'Yes'
            for i in range(len(hypothesis)):
                if hypothesis[i] == '0':  # If hypothesis hasn't been updated
                    hypothesis[i] = row[i]
                elif hypothesis[i] != row[i]:  # If mismatch, set '?' as a wildcard
                    hypothesis[i] = '?'

    return hypothesis

# Find the hypothesis and display the result
output_hypothesis = Find_S(dataset)
print("Output Hypothesis:", output_hypothesis)
```

enjoySport.csv

```
Sky,AirTemp,Humidity,Wind,Water,Forecast,EnjoySport
Sunny,Warm,Normal,Strong,Warm,Same,Yes
Sunny,Warm,High,Strong,Warm,Same,Yes
Rainy,Cold,High,Strong,Warm,Change,No
Sunny,Warm,High,Strong,Cool,Change,Yes
```

Output:

```
Output Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

3. Write a python program to implement Candidate Elimination algorithm

Code:

```
import pandas as pd
data = pd.read_csv("enjoysport.csv")
concepts = data.iloc[:, :-1].values
target = data.iloc[:, -1].values

def candidate_elimination(concepts, target):
    # Step 1: Initialize S and G
    S = list(concepts[0])  # most specific hypothesis
    G = [['?' for _ in range(len(S))]]  # most general hypothesis

    for i, val in enumerate(concepts):
        if target[i] == "Yes":
            # General → Specific: Remove inconsistent hypotheses from G
            G = [g for g in G if all(g[j] == '?' or g[j] == val[j] for j in range(len(S)))]
            # Update S
            for j in range(len(S)):
                if S[j] != val[j]:
                    S[j] = '?'
        else:
            # Specific → General: Specialize G
            new_G = []
            for g in G:
                for j in range(len(S)):
                    if g[j] == '?':
                        if S[j] != val[j]:
                            new_hypo = list(g)
                            new_hypo[j] = S[j]
                            if new_hypo not in new_G:
                                new_G.append(new_hypo)
            G = new_G

    return S, G

S_final, G_final = candidate_elimination(concepts, target)
print("Final Specific Hypothesis (S):", S_final)
print("Final General Hypotheses (G):", G_final)
```

Output:

```
Final Specific Hypothesis (S): ['Sunny', 'Warm', '?', 'Strong', '?', '?']
Final General Hypotheses (G): [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

4. Write a Python program to implement Simple Linear Regression

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Step 1: Load and preprocess CSV
df = pd.read_csv("lr.csv")  # Your CSV file name
df.dropna(inplace=True)  # Remove missing rows
df = df.select_dtypes(include=["float64", "int64"])  # Keep only numeric columns

# Ensure we have exactly 2 columns
if df.shape[1] != 2:
    raise ValueError("CSV must contain exactly two numeric columns (X and Y)")

# Step 2: Extract features and labels
X = df.iloc[:, 0].values.reshape(-1, 1)  # Features (independent variable)
Y = df.iloc[:, 1].values                # Labels (dependent variable)

# Step 3: Fit the model
model = LinearRegression()
model.fit(X, Y)

slope = model.coef_[0]
intercept = model.intercept_

# Step 4: Predict values
Y_pred = model.predict(X)

# Step 5: Display results
print("=== Simple Linear Regression ===")
print(f"Slope (m): {slope}")
print(f"Intercept (c): {intercept}")

# Step 6: Plot
plt.scatter(X, Y, color='blue', label='Actual Data')
plt.plot(X, Y_pred, color='red', linewidth=2, label='Regression Line')
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Simple Linear Regression")
plt.legend()
plt.grid(True)
plt.show()
```

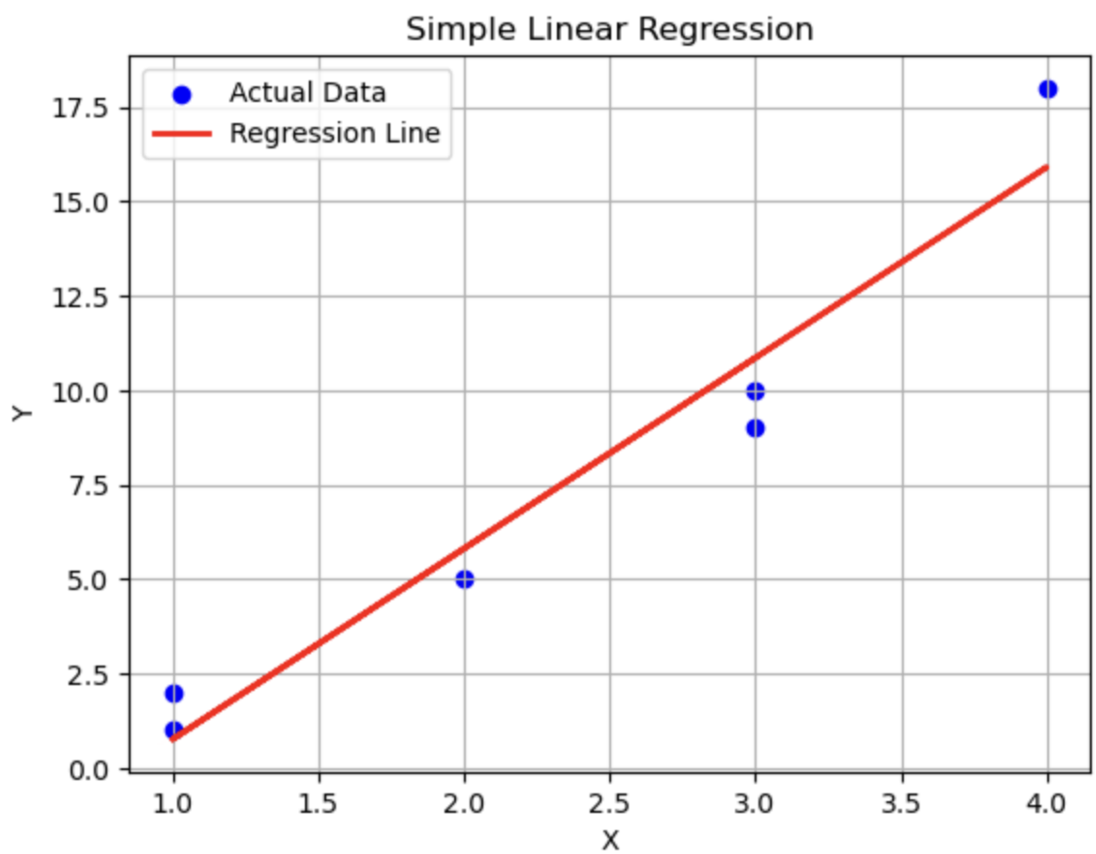
lr.csv

lr

x	y
1	2
2	5
3	9
4	18
1	1
3	10

Output:

```
=== Simple Linear Regression ===
Slope (m): 5.045454545454545
Intercept (c): -4.272727272727272
```



5.Write a Python program to implement Perceptron.

Code:

```
def perceptron_train(X, y, lr=0.1, epochs=10):
    weights = [0.0 for _ in range(len(X[0]) + 1)]  # +1 for bias
    for _ in range(epochs):
        for inputs, target in zip(X, y):
            summation = weights[0] + sum(w * x for w, x in zip(weights[1:], inputs))
            prediction = 1 if summation >= 0 else 0
            error = target - prediction
            weights[0] += lr * error  # bias update
            for i in range(len(inputs)):
                weights[i+1] += lr * error * inputs[i]
    return weights

def perceptron_predict(X, weights):
    return [1 if weights[0] + sum(w * x for w, x in zip(weights[1:], row)) >= 0 else 0 for row in X]

# Input data for AND and OR gates
X = [[0,0], [0,1], [1,0], [1,1]]

# ---------- AND Gate ----------
y_and = [0, 0, 0, 1]
w_and = perceptron_train(X, y_and)
pred_and = perceptron_predict(X, w_and)
print("AND Gate Prediction:", pred_and)

# ---------- OR Gate ----------
y_or = [0, 1, 1, 1]
w_or = perceptron_train(X, y_or)
pred_or = perceptron_predict(X, w_or)
print("OR Gate Prediction:", pred_or)
```

Output:

```
AND Gate Prediction: [0, 0, 0, 1]
OR Gate Prediction: [0, 1, 1, 1]
```

6.Write a Python program to implement Multiple Linear Regression

Code:

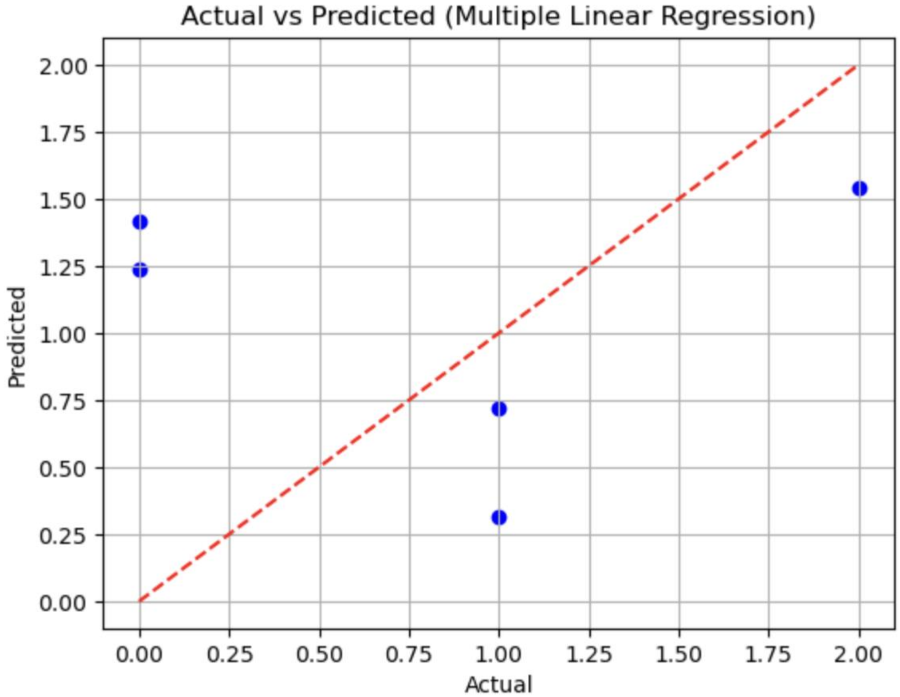
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import LabelEncoder, StandardScaler


# Load CSV data
df = pd.read_csv('knn.csv')  # Replace with your CSV file name

# Select features and target
X = df[['Weight(x2)', 'Height(y2)']]  # Replace with actual column names
y = df['Class']                  # Replace with your target column
y = LabelEncoder().fit_transform(df['Class'])

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

# Create and train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluation
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("R² Score:", r2_score(y_test, y_pred))
print("MSE:", mean_squared_error(y_test, y_pred))

# Optional: Plot actual vs predicted
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')  # ideal line
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs Predicted (Multiple Linear Regression)")
plt.grid(True)
plt.show()
```

Dataset:

Weight(x2)	Height(y2)	Class
51	167	Underweight
66	177	Normal
75	169	Overweight
69	176	Normal
50	173	Underweight
82	170	Overweight
65	172	Normal
58	180	Underweight
68	162	Overweight
63	165	Normal
52	174	Underweight

Output:

```
Coefficients: [-0.04585803  0.00802144]
Intercept: 2.5409440282439357
R² Score: -0.5339847960067876
MSE: 0.8590314857638012
```

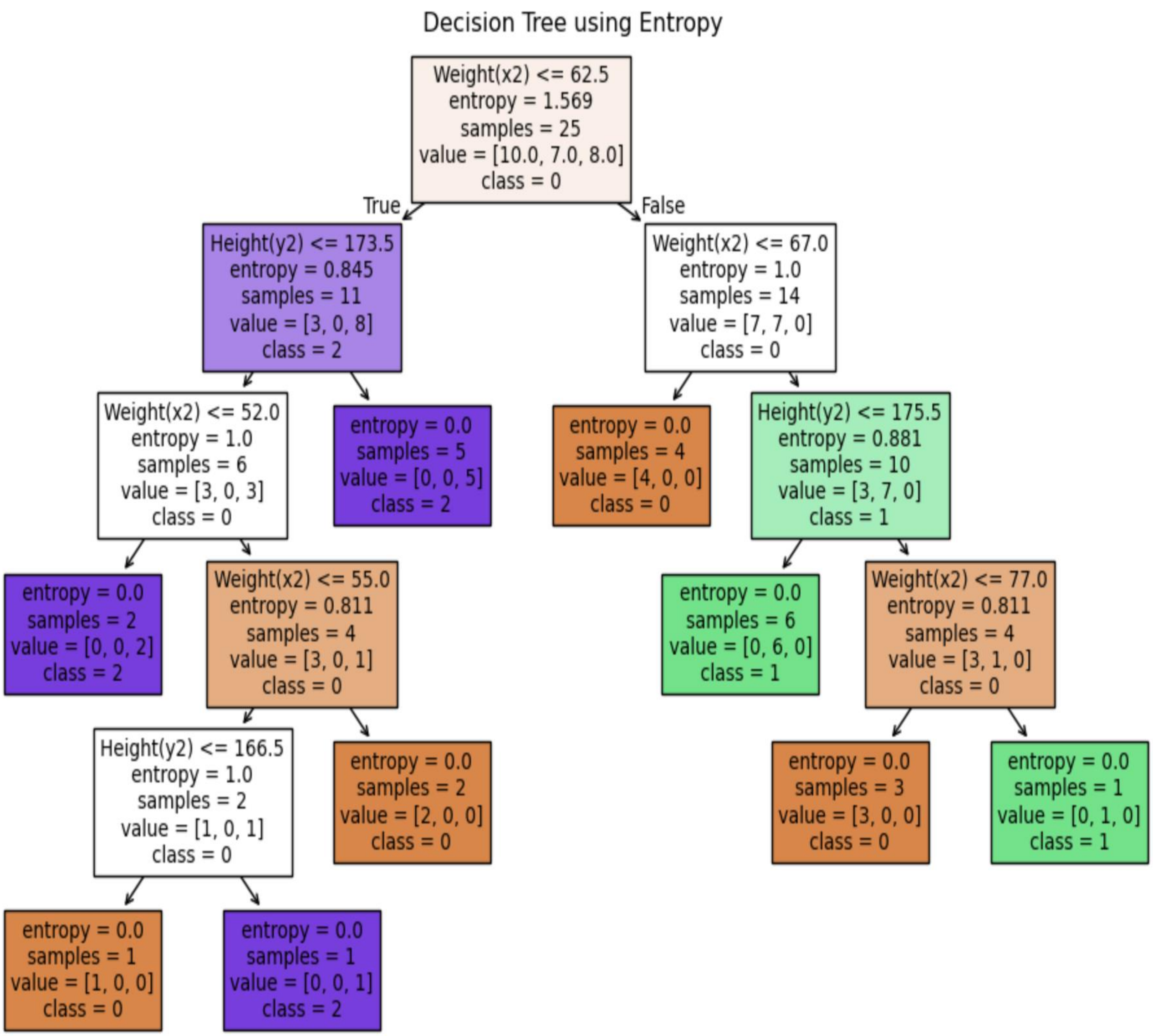


7.Decision tree using entropy(ID3)

Code:

```
#decision tree using entropy(ID3)
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('knn.csv')  # replace with your actual CSV file path
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Encode categorical features if any
for col in X.columns:
    if X[col].dtype == object:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col])

# Encode target if needed
if y.dtype == object:
    y = LabelEncoder().fit_transform(y)

clf = DecisionTreeClassifier(criterion='entropy')
clf = clf.fit(X, y)
plt.figure(figsize=(12, 8))
tree.plot_tree(clf, filled=True, feature_names=X.columns, class_names=[str(cls) for cls in
clf.classes_])
plt.title("Decision Tree using Entropy")
plt.show()
```

Output:

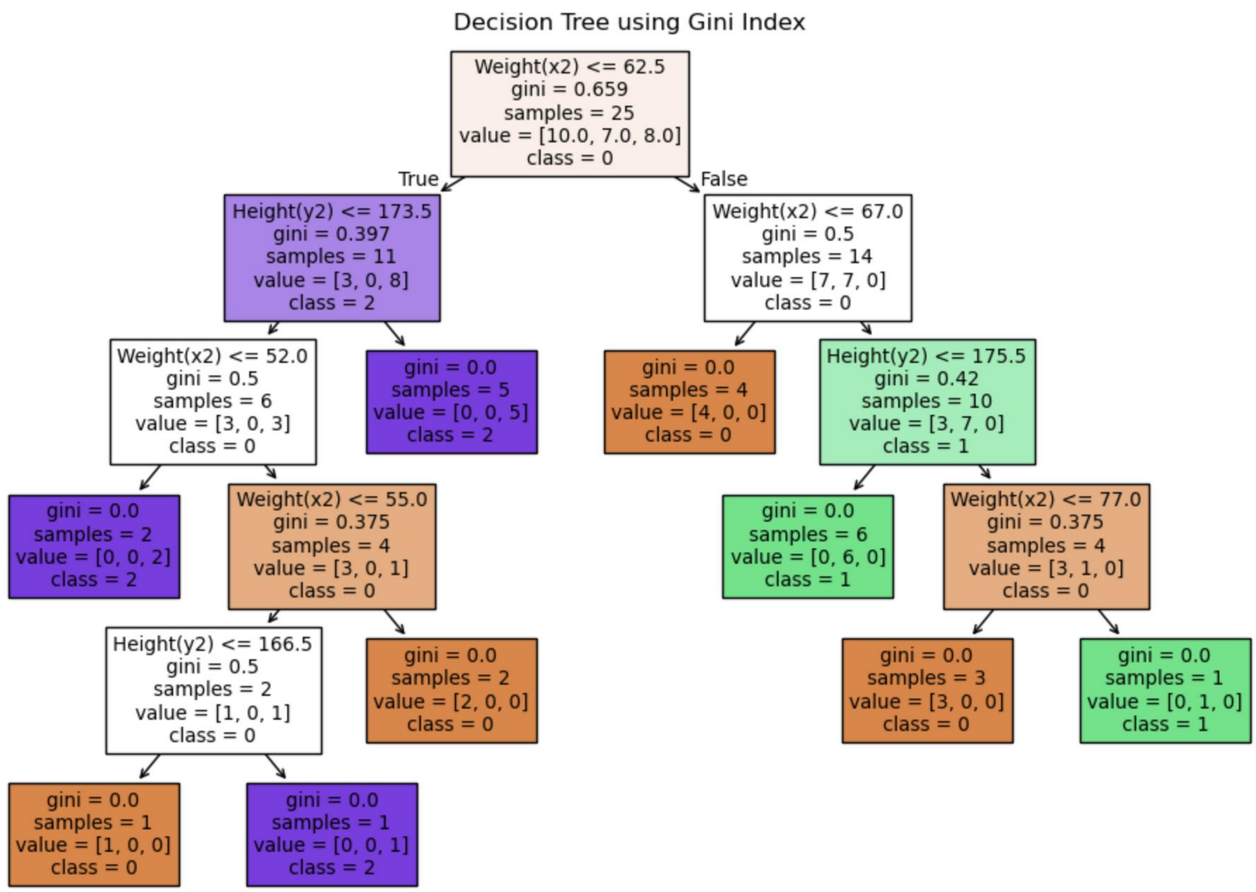


8.Decision tree using gini index

Code:

```
#decision tree using gini
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('knn.csv')  # replace with your actual CSV file path
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Encode categorical features if any
for col in X.columns:
    if X[col].dtype == object:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col])

# Encode target if needed
if y.dtype == object:
    y = LabelEncoder().fit_transform(y)

clf = DecisionTreeClassifier(criterion='gini')
clf = clf.fit(X, y)
plt.figure(figsize=(12, 8))
tree.plot_tree(clf, filled=True, feature_names=X.columns, class_names=[str(cls) for cls in
clf.classes_])
plt.title("Decision Tree using Gini Index")
plt.show()
```

Output:


Decision Tree using Gini Index
Weight(x2) <= 62.5
gini = 0.659
samples = 25
value = [10.0, 7.0, 8.0]
class = 0
True
False
Height(y2) <= 173.5
gini = 0.397
samples = 11
value = [3, 0, 8]
class = 2
Weight(x2) <= 67.0
gini = 0.5
samples = 14
value = [7, 7, 0]
class = 0
Weight(x2) <= 52.0
gini = 0.5
samples = 6
value = [3, 0, 3]
class = 0
gini = 0.0
samples = 5
value = [0, 0, 5]
class = 2
gini = 0.0
samples = 4
value = [4, 0, 0]
class = 0
Height(y2) <= 175.5
gini = 0.42
samples = 10
value = [3, 7, 0]
class = 1
gini = 0.0
samples = 2
value = [0, 0, 2]
class = 2
Weight(x2) <= 55.0
gini = 0.375
samples = 4
value = [3, 0, 1]
class = 0
gini = 0.0
samples = 6
value = [0, 6, 0]
class = 1
Weight(x2) <= 77.0
gini = 0.375
samples = 4
value = [3, 1, 0]
class = 0
Height(y2) <= 166.5
gini = 0.5
samples = 2
value = [1, 0, 1]
class = 0
gini = 0.0
samples = 2
value = [2, 0, 0]
class = 0
gini = 0.0
samples = 3
value = [3, 0, 0]
class = 0
gini = 0.0
samples = 1
value = [0, 1, 0]
class = 1
gini = 0.0
samples = 1
value = [1, 0, 0]
class = 0
gini = 0.0
samples = 1
value = [0, 0, 1]
class = 2

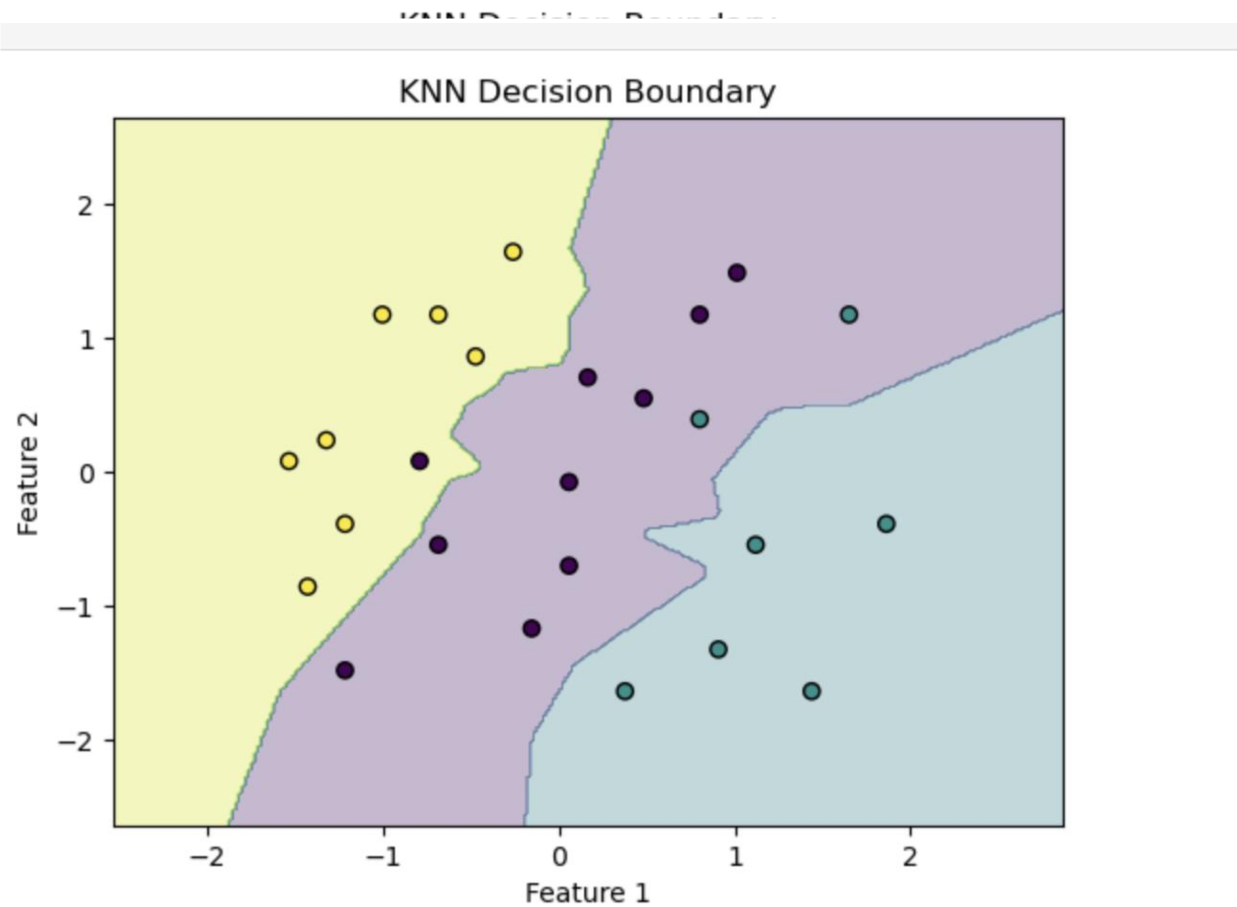
9. KNN Algorithm using sklearn

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# Load data
df = pd.read_csv('knn.csv')
X = df[['Weight(x2)', 'Height(y2)']]  # Replace with actual column names
y = LabelEncoder().fit_transform(df['Class'])

# Scale features
X = StandardScaler().fit_transform(X)

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

# Train KNN
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)

# Plot decision boundaries
x_min, x_max = X[:, 0].min()-1, X[:, 0].max()+1
y_min, y_max = X[:, 1].min()-1, X[:, 1].max()+1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 300),
                     np.linspace(y_min, y_max, 300))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

plt.contourf(xx, yy, Z, alpha=0.3)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k')
plt.title("KNN Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

Output:

```
Accuracy: 0.8

Classification Report:
              precision    recall  f1-score   support

           0       0.67      1.00      0.80         2
           1       1.00      0.50      0.67         2
           2       1.00      1.00      1.00         1

    accuracy                           0.80         5
   macro avg       0.89      0.83      0.82         5
weighted avg       0.87      0.80      0.79         5


Confusion Matrix:
 [[2 0 0]
 [1 1 0]
 [0 0 1]]
```



10. Kmeans clustering

Code:

```
#KMeans using any other dataset of our choice
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# 1. Load your dataset
data = pd.read_csv('knn.csv')  # Replace with your data source

# 2. Preprocess the data
# Select only numerical features (K-Means works with numerical data)
X = data.select_dtypes(include=[np.number])

# Handle missing values if any
X = X.dropna()  # or use imputation

# Scale the data (important for K-Means)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. Apply K-Means clustering
k = 3  # Number of clusters - you'll need to determine this
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X_scaled)

# 4. Get cluster assignments
cluster_labels = kmeans.labels_

# Add cluster labels back to original data
data['Cluster'] = cluster_labels

# 5. View cluster centers (in original feature space)
centers = scaler.inverse_transform(kmeans.cluster_centers_)
print("Cluster Centers:")
print(pd.DataFrame(centers, columns=X.columns))

# 6. Visualization
plt.figure(figsize=(12, 6))

# Plot 1: First two features
plt.subplot(1, 2, 1)
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=cluster_labels, cmap='viridis', alpha=0.6)
```

```
plt.scatter(centers[:, 0], centers[:, 1], s=200, c='red', marker='X')
plt.xlabel(X.columns[0])
plt.ylabel(X.columns[1])
plt.title('Cluster Visualization (Features 1 & 2)')
```

Output:

```
Cluster Centers:
   Weight(x2)  Height(y2)
0   72.000000  165.714286
1   53.428571  169.857143
2   66.636364  178.454545
```

```
: Text(0.5, 1.0, 'Cluster Visualization (Features 1 & 2)')
```