-----------First Program-------------

```
import csv
a = []
with open('finds.csv') as csfile:
    reader = csv.reader(csfile)
    for row in reader:
        a.append(row)
        print(row)
num_attributes = len(a[0]) - 1
print(["?"] * num_attributes)
print(["0"] * num_attributes)
hypothesis = a[0][:-1]
for i in range(len(a)):
    if a[i][num_attributes] == "Yes":
        for j in range(num_attributes):
            if a[i][j] != hypothesis[j]:
                hypothesis[j] = '?'
    print(i + 1, hypothesis)
print(hypothesis)
```

---------finds.csv

```
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
```

----------Second Program--------------

```
# Program 2
import numpy as np
import pandas as pd
# Loading Data from a CSV File
data = pd.DataFrame(data=pd.read_csv('Training_examples.csv'))
# Separating concept features from Target
concepts = np.array(data.iloc[:, 0:-1])
target = np.array(data.iloc[:, -1])
```

```
def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "No":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

Training_examples.csv

```
Sky,Air,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
```

-----------Program 3-----------------

```
import pandas as pd
import numpy as np
# Import the dataset and define the feature as well as the target datasets/columns
dataset = pd.read_csv('playtennis.csv', names=['outlook', 'temperature', 'humidity', 'wind', 'class'])
attributes = ('Outlook', 'Temperature', 'Humidity', 'Wind', 'PlayTennis')
```

```
def entropy(target_col):
    elements, counts = np.unique(target_col, return_counts=True)
    entropy = np.sum([(-counts[i] / np.sum(counts)) * np.log2(counts[i] / np.sum(counts)) for i in range(len(elements))])
    return entropy
def InfoGain(data, split_attribute_name, target_name="class"):
    total_entropy = entropy(data[target_name])
    vals, counts = np.unique(data[split_attribute_name], return_counts=True)
    Weighted_Entropy = np.sum([(counts[i] / np.sum(counts)) * entropy(data.where(data[split_attribute_name] == vals[i]).dropna()[target_name]) for i in range(len(vals))])
    Information_Gain = total_entropy - Weighted_Entropy
    return Information_Gain
def ID3(data, originaldata, features, target_attribute_name="class", parent_node_class=None):
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]
    elif len(data) == 0:
        return np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originaldata[target_attribute_name], return_counts=True)[1])]
    elif len(features) == 0:
        return parent_node_class
    else:
        parent_node_class = np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attribute_name], return_counts=True)[1])]
        item_values = [InfoGain(data, feature, target_attribute_name) for feature in features]
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]
        tree = {best_feature: {}}
        features = [i for i in features if i != best_feature]
        for value in np.unique(data[best_feature]):
            value = value
            sub_data = data.where(data[best_feature] == value).dropna()
            subtree = ID3(sub_data, dataset, features, target_attribute_name, parent_node_class)
            tree[best_feature][value] = subtree
        return tree
```

```
def predict(query, tree, default=1):
    for key in list(query.keys()):
        if key in list(tree.keys()):
            try:
                result = tree[key][query[key]]
            except:
                return default
            result = tree[key][query[key]]
            if isinstance(result, dict):
                return predict(query, result)
            else:
                return result
def train_test_split(dataset):
    training_data = dataset.iloc[:14].reset_index(drop=True)
    return training_data
def test(data, tree):
    queries = data.iloc[:, :-1].to_dict(orient="records")
    predicted = pd.DataFrame(columns=["predicted"])
    for i in range(len(data)):
        predicted.loc[i, "predicted"] = predict(queries[i], tree, 1.0)
    print('The prediction accuracy is: ', (np.sum(predicted["predicted"] == data["class"]) / len(data)) * 100, '%')
XX = train_test_split(dataset)
training_data = XX
tree = ID3(training_data, training_data, training_data.columns[:-1])
print('Display Tree:', tree)
print('len=', len(training_data))
test(training_data, tree)
```

.csv

```
0,0,0,0,0
0,0,0,1,0
1,0,0,0,1
2,1,0,0,1
2,2,1,0,1
2,2,1,1,0
```

```
1,2,1,1,1
0,1,0,0,0
0,2,1,0,1
2,1,1,0,1
0,1,1,1,1
1,1,0,1,1
```

```
1,0,1,0,1
2,1,0,1,0
0,1,1,1,1
1,1,1,1,1
```

```
-----------------# Program 4---------------

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X, axis=0)

y = y/100

def sigmoid(x):

    return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):

    return x * (1 - x)

epoch = 7000

learning_rate = 0.1

inputlayer_neurons = 2

hiddenlayer_neurons = 3

output_neurons = 1

wh = np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))

bh = np.random.uniform(size=(1, hiddenlayer_neurons))

wo = np.random.uniform(size=(hiddenlayer_neurons, output_neurons))

bo = np.random.uniform(size=(1, output_neurons))

for i in range(epoch):

    net_h = np.dot(X, wh) + bh

    sigma_h = sigmoid(net_h)

    net_o = np.dot(sigma_h, wo) + bo

    output = sigmoid(net_o)

    deltaK = (y - output) * derivatives_sigmoid(output)

    deltaH = deltaK.dot(wo.T) * derivatives_sigmoid(sigma_h)

    wo = wo + sigma_h.T.dot(deltaK) * learning_rate

    wh = wh + X.T.dot(deltaH) * learning_rate

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n", output)
```

------------Program 5-----------------

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
data = pd.read_csv('dataset5.csv')
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = MultinomialNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
precision = metrics.precision_score(y_test, y_pred, average='weighted', zero_division=1)
recall = metrics.recall_score(y_test, y_pred, average='weighted')
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

------------------dataset5.csv------

```
feature1,feature2,feature3,feature4,target
5.1,3.5,1.4,0.2,class1
4.9,3.0,1.4,0.2,class1
4.7,3.2,1.3,0.2,class1
4.6,3.1,1.5,0.2,class1
5.0,3.6,1.4,0.2,class1
5.4,3.9,1.7,0.4,class2
4.6,3.4,1.4,0.3,class2
5.0,3.4,1.5,0.2,class2
4.4,2.9,1.4,0.2,class2
4.9,3.1,1.5,0.1,class2
5.4,3.7,1.5,0.2,class3
4.8,3.4,1.6,0.2,class3
4.8,3.0,1.4,0.1,class3
4.3,3.0,1.1,0.1,class3
```

```
5.8,4.0,1.2,0.2,class3
```

```
----------------------#Program 6-----------------------
import pandas as pd
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
data = pd.read_csv("ds4.csv")
heart_disease = pd.DataFrame(data)
print(heart_disease)
model = BayesianModel([
   ('age', 'Lifestyle'),
   ('Gender', 'Lifestyle'),
   ('Family', 'heartdisease'),
   ('diet', 'cholestrol'),
   ('Lifestyle', 'diet'),
   ('cholestrol', 'heartdisease'),
   ('diet', 'cholestrol')])
model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)
HeartDisease_infer = VariableElimination(model)
print('For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4')
print('For Gender enter Male:0, Female:1')
print('For Family History enter Yes:1, No:0')
print('For Diet enter High:0, Medium:1')
print('for LifeStyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3')
print('for Cholesterol enter High:0, BorderLine:1, Normal:2')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={
   'age': int(input('Enter Age: ')),
   'Gender': int(input('Enter Gender: ')),
   'Family': int(input('Enter Family History: ')),
   'diet': int(input('Enter Diet: ')),
   'Lifestyle': int(input('Enter Lifestyle: ')),
   'cholestrol': int(input('Enter Cholestrol: '))})
print(q)
```

```
---ds4.csv
age,Gender,Family,diet,Lifestyle,cholestrol,heartdisease
```

```
0,0,1,1,3,0,1

0,1,1,1,3,0,1

1,0,0,0,2,1,1

4,0,1,1,3,2,0

3,1,1,0,0,2,0

2,0,1,1,1,0,1

4,0,1,0,2,0,1

0,0,1,1,3,0,1

3,1,1,0,0,2,0

1,1,0,0,0,2,1

4,1,0,1,2,0,1

4,0,1,1,3,2,0

2,1,0,0,0,0,0

2,0,1,1,1,0,1

3,1,1,0,0,1,0

0,0,1,0,0,2,1

1,1,0,1,2,1,1

3,1,1,1,0,1,0

4,0,1,1,3,2,0
```

```
----------------------------------#program 7

import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# Step 1: Load data from a CSV file

def load_data(csv_file):

    data = pd.read_csv(csv_file)

    return data

# Step 2: Apply K-Means clustering

def apply_kmeans(data, n_clusters):

    # K-Means algorithm

    kmeans = KMeans(n_clusters=n_clusters)

    kmeans.fit(data)

    # Step 3: Get the cluster labels

    labels = kmeans.labels_
    return labels, kmeans.cluster_centers_

# Step 4: Visualize the clusters

def visualize_clusters(data, labels, cluster_centers):

    plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c=labels, cmap='rainbow')

    # plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], s=300, c='black', marker='X')  # Cluster centers

    plt.title("K-Means Clustering")

    plt.xlabel("Feature 1")

    plt.ylabel("Feature 2")

    plt.show()

if __name__ == "__main__":

    # Load data from CSV (Assume the CSV file has two numeric columns)

    data = load_data('your_data.csv')

    # Specify the number of clusters

    n_clusters = 3

    # Apply K-Means algorithm

    labels, cluster_centers = apply_kmeans(data, n_clusters)

    # Visualize the clusters

    visualize_clusters(data, labels, cluster_centers)
```

--------------------your_data.csv

```
Feature1,Feature2
5.1,3.5
4.9,3.0
6.2,3.4
5.9,3.0
5.0,3.4
5.2,3.6
5.3,3.2
4.8,2.9
6.1,3.3
5.5,3.1
5.7,3.8
6.0,3.2
5.8,3.5
5.4,3.7
4.7,2.8
6.3,3.6
4.9,2.7
5.6,3.4
6.4,3.8
5.5,3.3
5.1,2.9
4.6,2.8
5.0,3.1
5.3,3.5
5.6,3.9
8.0,7.5
7.8,7.3
8.1,7.6
7.9,7.4
8.2,7.8
8.3,7.6
8.5,7.9
8.0,7.2
7.7,7.4
8.1,7.5
8.4,7.7
8.2,7.6
7.9,7.3
7.8,7.2
8.3,7.5
8.6,7.9
8.0,7.7
8.1,7.8
7.6,7.1
8.4,7.3
8.2,7.9
7.5,7.4
8.3,7.6
8.5,7.8
8.7,7.5
2.0,1.5
1.9,1.4
2.1,1.6
1.8,1.2
2.2,1.7
```

-------------------------------------8

```
# 1. Import Data
from sklearn.datasets import load_iris
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
# Load Iris dataset
iris = load_iris()
# Create a DataFrame from the iris dataset
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['species'] = iris.target
# Save the dataset to a CSV file
iris_df.to_csv('iris_data', index=False)
# Print some details
print("Feature Names:", iris.feature_names)
print("Iris Data:")
print(iris.data)
print("Target Names:", iris.target_names)
print("Target:", iris.target)
# 2. Split the data into Training and Test sets
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.25)
# 3. Build the KNN model
clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
# 4. Calculate accuracy on test data
accuracy = clf.score(X_test, y_test)
print("Accuracy =", accuracy)
# 5. Calculate predictions with the test data
predictions = clf.predict(X_test)
print("Predicted Data:")
print(predictions)
# Display the test labels
print("Test Data:")
print(y_test)
# 6. Identify misclassifications
```

```
misclassifications = predictions - y_test

print("Result of misclassifications:")

print(misclassifications)

# Calculate total misclassified samples

total_misclassified = sum(abs(misclassifications))

print('Total number of samples misclassified =', total_misclassified)

----------------------------9

from math import ceil

import numpy as np

from scipy import linalg

def lowess(x, y, f, iterations):

    n = len(x)

    r = int(ceil(f * n))

    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]

    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)

    w = (1 - w ** 3) ** 3

    yest = np.zeros(n)

    delta = np.ones(n)

    for iteration in range(iterations):

        for i in range(n):

            weights = delta * w[:, i]

            b = np.array([np.sum(weights * y), np.sum(weights * y * x)])

            A = np.array([[np.sum(weights), np.sum(weights * x)],[np.sum(weights * x), np.sum(weights * x * x)]])

            beta = linalg.solve(A, b)

            yest[i] = beta[0] + beta[1] * x[i]

        residuals = y - yest

        s = np.median(np.abs(residuals))

        delta = np.clip(residuals / (6.0 * s), -1, 1)

        delta = (1 - delta ** 2) ** 2

    return yest

import math

n = 100

x = np.linspace(0, 2 * math.pi, n)

y = np.sin(x) + 0.3 * np.random.randn(n)
```

```
f =0.25

iterations=3

yest = lowess(x, y, f, iterations)

import matplotlib.pyplot as plt

plt.plot(x,y,"r.")

plt.plot(x,yest,"b-")
```