```
import csv

# Load data from CSV
with open('lab1a.csv', 'r') as f:
    reader = csv.reader(f)
    your_list = list(reader)

if your_list:
    num_attributes = len(your_list[0]) - 1
    # Exclude the class label
    h = ['0'] * num_attributes
    print(h)

for i in your_list:
    print(i)

    # Check if the instance is positive
    if i[-1] == "Yes":
        j = 0

        # Iterate through attributes of the instance (excluding the class label)
        for x in i[:-1]:
            # Update hypothesis
            if x != h[j] and h[j] == '0':
                h[j] = x
            elif x != h[j] and h[j] != '0':
                h[j] = '?'
            j += 1
        print(h)

print("Most specific hypothesis is")
print(h)
```

```
            print(f"{indent}{n.parent}: {n.parent_val} {decision}")
        for child in n.childs:
            if child:
                print_tree(child, level + 1)

    # Load the dataset

    # Example usage
    df = pd.read_csv('Play Tennis.csv')  # Read the dataset from CSV
    dt = DecisionTree(df, 'Play', 'Yes')  # Initialize the DecisionTree object
    dt.update_nodes()  # Build the decision tree
    print_tree(dt)  # Print the decision tree
```

```
import math
import pandas as pd
from operator import itemgetter

class DecisionTree:
    def __init__(self, df, target, positive, parent_val=None, parent=None):
        self.data = df
        self.target = target
        self.positive = positive
        self.parent_val = parent_val
        self.parent = parent
        self.childs = []
        self.decision = None

    def _get_entropy(self, data):
        p = sum(data[self.target] == self.positive)
        n = data.shape[0] - p
        p_ratio = p / (p + n)
        n_ratio = 1 - p_ratio
        entropy_p = -p_ratio * math.log2(p_ratio) if p_ratio != 0 else 0
        entropy_n = -n_ratio * math.log2(n_ratio) if n_ratio != 0 else 0
        return entropy_p + entropy_n

    def _get_gain(self, feat):
        avg_info = 0
        for val in self.data[feat].unique():
            subset = self.data[self.data[feat] == val]
            avg_info += self._get_entropy(subset) * len(subset) / self.data.shape[0]
        return self._get_entropy(self.data) - avg_info

    def _get_splitter(self):
        self.splitter = max(self.gains, key=itemgetter(1))[0]
    def update_nodes(self):
        self.features = [col for col in self.data.columns if col != self.target]
        self.entropy = self._get_entropy(self.data)
        if self.entropy != 0:
            self.gains = [(feat, self._get_gain(feat)) for feat in self.features]
            self._get_splitter()
            residual_columns = [k for k in self.data.columns if k != self.splitter]
            for val in self.data[self.splitter].unique():
                df_tmp = self.data[self.data[self.splitter] == val][residual_columns]
                tmp_node = DecisionTree(df_tmp, self.target, self.positive, val, self.splitter)
                tmp_node.update_nodes()
                self.childs.append(tmp_node)
        else:
            self.decision = self.data[self.target].iloc[0]

def print_tree(n, level=0):
    if n.parent is not None:
        indent = "  " * level
        decision = f"(Decision: {n.decision})" if n.decision is not None else ""
    Dept of ISE
```

```
    from sklearn.datasets import load_iris
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.model_selection import train_test_split
    import numpy as np

    dataset=load_iris()
    #print(dataset)
    X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_state=0)

    kn=KNeighborsClassifier(n_neighbors=1)
    kn.fit(X_train,y_train)

    for i in range(len(X_test)):
        x=X_test[i]
        x_new=np.array([x])
        prediction=kn.predict(x_new)

    print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",prediction,dataset["target_names"][predicti
    on])
    print(kn.score(X_test,y_test))
```

```
import csv

# Open and read the CSV file
with open("lab1a.csv") as f:
    csv_file = csv.reader(f)
    data = list(csv_file)

# Initialize specific and general hypotheses
specific = data[1][:-1]  # Initial specific hypothesis from the first positive example
general = [['?' for _ in range(len(specific))] for _ in range(len(specific))]  # General hypothesis

# Process each example in the data
for i in data:
    if i[-1] == "Yes":  # Positive example
        for j in range(len(specific)):
            if i[j] != specific[j]:
                specific[j] = "?"
                general[j][j] = "?"
    elif i[-1] == "No":  # Negative example
        for j in range(len(specific)):
            if i[j] != specific[j]:
                general[j][j] = specific[j]
            else:
                general[j][j] = "?"

    # Print steps of the Candidate Elimination Algorithm
    print("\nStep " + str(data.index(i) + 1) + " of Candidate Elimination Algorithm")
    print("Specific hypothesis: ", specific)
    print("General hypothesis: ", general)

# Collect the general hypotheses
gh = []  # gh = general Hypothesis
for i in general:
    for j in i:
        if j != '?':
            gh.append(i)
            break

# Final hypotheses
print("\nFinal Specific hypothesis:\n", specific)
print("\nFinal General hypothesis:\n", gh)
```

```
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
data = pd.read_csv('Play Tennis.csv')
print("The first 5 values of data is :\n",data.head())
X = data.iloc[:,:-1]
print("\nThe First 5 values of train data is\n",X.head())
y = data.iloc[:,-1]
print("\nThe first 5 values of Train output is\n",y.head())
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)
le_temperature = LabelEncoder()
X.Temperature = le_temperature.fit_transform(X.Temperature)
le_humidity = LabelEncoder()
X.Humidity = le_humidity.fit_transform(X.Humidity)
le_windy = LabelEncoder()
X.Windy = le_windy.fit_transform(X.Windy)
print("\nNow the Train data is :\n",X.head())
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)
classifier = GaussianNB()
classifier.fit(X_train,y_train)
from sklearn.metrics import accuracy_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

```
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-")
```

```
import pandas as pd
from sklearn.model_selection import train_test_split from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Step 1: Dataset Preparation data = {
'Weight': [150, 200, 250, 180, 300, 220],
'Color': ['Red', 'Red', 'Orange', 'Orange', 'Red', 'Orange'],
'Label': ['Apple', 'Apple', 'Orange', 'Orange', 'Apple', 'Orange']
}

df = pd.DataFrame(data)

# Step 2: Feature Extraction
df['Color'] = df['Color'].map({'Red': 0, 'Orange': 1})

# Step 3: Data Split
X = df[['Weight', 'Color']] y = df['Label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 4: Model Training svm = SVC(kernel='linear') svm.fit(X_train, y_train)

# Step 5: Model Evaluation y_pred = svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred) print("Accuracy:", accuracy)

# Step 6: Prediction new_data = {
'Weight': [190],
'Color': [0]
}

new_df = pd.DataFrame(new_data) new_prediction = svm.predict(new_df) print("New prediction:", new_prediction)
```

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset=load_iris()
# print(dataset)

X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
# print(X)

plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

# K-PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')

# GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```