```
#Experiment : 1
import random
import csv
attributes = [["Sunny", "Rainy"], ["Warm", "Cold"], ["Normal", "High"], ["Strong", "Weak"], ["Warm", "Cool"], ["Same", "Change"]]
num_attributes = len(attributes)
print("The most general hypothesis: ['?', '?', '?', '?', '?', '?']")
print("The most general hypothesis: ['0', '0', '0', '0', '0', '0']")
a = []
print("The given training dataset: ")
file_path  = "C:/Users/kkndc/Downloads/sample_restaurants.restaurants.csv"
with open(file_path, 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        a.append(row)
print(row)
print("The initial value of hypothesis: ")
hypothesis = ['0'] * num_attributes
print(hypothesis)
for j in range(0, num_attributes):
    hypothesis[j] = a[0][j]
print("FIND-S: Finding a Maximality Specific Hypothesis")
for i in range(0, len(a)):
    if a[i][num_attributes] == "yes":
        for j in range(0, num_attributes):
            if a[i][j] != hypothesis[j]:
                hypothesis[j] = '?'
            else:
                hypothesis[j] = a[i][j]
    print("For training example no: {0} the hypothesis is ".format(i), hypothesis)
print("The Maximally Specific Hypothesis for a given training examples:")
print(hypothesis)


# Experiment 2

import numpy as np
import pandas as pd

data = pd.DataFrame(pd.read_csv('/content/enjoysport.csv'))
concepts = np.array(data.iloc[:, :-1])
target = np.array(data.iloc[:, -1])

print("Concepts:")
print(concepts)
print("\nTarget:")
print(target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?" for _ in range(len(specific_h))] for _ in range(len(specific_h))]

    print("\nInitialization:")
    print("Specific Hypothesis (S):", specific_h)
    print("General Hypothesis (G):", general_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] = '?'

        elif 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] = '?'

        print(f"\nStep {i+1}: Updated Hypotheses")
        print("Specific Hypothesis (S):", specific_h)
        print("General Hypothesis (G):", general_h)

    general_h = [hyp for hyp in general_h if hyp != ['?'] * len(specific_h)]

    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("\nFinal Specific Hypothesis (S):")
print(s_final)
print("\nFinal General Hypothesis (G):")
print(g_final)




# Experiment 3
import pandas as pd
import math

def id3(df, target_attribute_name, attribute_names, default_class=None):
    if len(set(df[target_attribute_name])) == 1:
        return df[target_attribute_name].iloc[0]
    elif len(attribute_names) == 0:
        return default_class
    else:
        gains = {attribute_name: information_gain(df, attribute_name, target_attribute_name) for attribute_name in attribute_names}
        best_attribute = max(gains, key=gains.get)
        tree = {best_attribute: {}}
        remaining_attributes = [attr for attr in attribute_names if attr != best_attribute]

        for value in df[best_attribute].unique():
            subset = df[df[best_attribute] == value]
            subtree = id3(subset, target_attribute_name, remaining_attributes, default_class)
            tree[best_attribute][value] = subtree

        return tree

def entropy(probs):
    return sum([-prob * math.log(prob, 2) for prob in probs if prob != 0])

def entropy_of_list(a_list):
    total_instances = len(a_list)
    class_counts = a_list.value_counts()
    probs = class_counts / total_instances
    return entropy(probs)

def information_gain(df, split_attribute_name, target_attribute_name):
    total_entropy = entropy_of_list(df[target_attribute_name])
    subset_entropy = df.groupby(split_attribute_name)[target_attribute_name].apply(entropy_of_list)
    subset_sizes = df.groupby(split_attribute_name).size()
    weighted_entropy = (subset_entropy * subset_sizes / len(df)).sum()
    return total_entropy - weighted_entropy

df = pd.read_csv('/content/id3.csv')

attribute_names = list(df.columns)
target_attribute_name = 'Answer'
attribute_names.remove(target_attribute_name)

tree = id3(df, target_attribute_name, attribute_names)

print("Decision Tree:")
print(tree)




# Experiment 4
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
IceCream=pd.read_csv('/content/IceCreamData.csv')
print(IceCream)



# Divide the data into “Attributes” and “labels”
X = IceCream[['Temperature']]
y = IceCream['Revenue']
# Split 80% of the data to the training set while 20% of the data to test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
# Create a Linear Regression model and fit it
regressor =LinearRegression(fit_intercept=True)
regressor.fit(X_train,y_train)
print('Linear Model Coeff (m) =' , regressor.coef_)
print('Linear Model Coeff (b) =' , regressor.intercept_)
# Predicting the data
y_predict=regressor.predict(X_test)
print(y_predict)
```

```
# Scatter plot on Training Data
plt.scatter(X_train,y_train,color='blue')
plt.plot(X_train,regressor.predict(X_train),color='red')
plt.ylabel('Revenue [$]')
plt.xlabel('Temperatur [degC]')
plt.title('Revenue Generated vs. Temperature @Ice Cream Stand (Training)')


# Scatter plot on Testing Data
plt.scatter(X_test,y_test,color='blue')
plt.plot(X_test,regressor.predict(X_test),color='red')
plt.ylabel('Revenue [$]')
plt.xlabel('Temperatur [degC]')
plt.title('Revenue Generated vs. Temperature @Ice Cream Stand (Training)')



# Prediction the revenve using Temperature Value directly
print('---------0---------')
Temp = -0
Revenue = regressor.predict([[Temp]])
print(Revenue)
print('--------35----------')
Temp = 35
Revenue = regressor.predict([[Temp]])
print(Revenue)
print('--------55----------')
Temp = 55
Revenue = regressor.predict([[Temp]])
print(Revenue)




# Experiment 5
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('/content/Social_Network_Ads.csv')
print(dataset)


X = dataset.iloc[:, [2,3]].values
y = dataset.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)



y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()


from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
# Experiment 6
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score
import pandas as pd

dataset = load_breast_cancer(as_frame=True)
X = dataset['data']
y = dataset['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

ss_train = StandardScaler()
X_train = ss_train.fit_transform(X_train)

ss_test = StandardScaler()
X_test = ss_test.fit_transform(X_test)

models = {
    'Logistic Regression': LogisticRegression(),
    'Support Vector Machines': LinearSVC(),
    'Decision Trees': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
    'K-Nearest Neighbor': KNeighborsClassifier()
}

accuracy, precision, recall = {}, {}, {}

for key in models.keys():
    models[key].fit(X_train, y_train)
    predictions = models[key].predict(X_test)

    accuracy[key] = accuracy_score(y_test, predictions)
    precision[key] = precision_score(y_test, predictions)
    recall[key] = recall_score(y_test, predictions)

df_model = pd.DataFrame(index=models.keys(), columns=['Accuracy', 'Precision', 'Recall'])
df_model['Accuracy'] = accuracy.values()
df_model['Precision'] = precision.values()
df_model['Recall'] = recall.values()

ax = df_model.plot.barh()
ax.legend(
    ncol=len(models.keys()),
    bbox_to_anchor=(0, 1),
    loc="lower left",
    prop={'size': 14}
)

plt.tight_layout()
plt.show()
```

```
from sklearn.metrics import confusion_matrix
```

```
cm=confusion_matrix(y_test,predictions)
TN,FP,FN,TP=confusion_matrix(y_test,predictions).ravel()
print("True Positive(TP):",TP)
print("False Positive(FP):",FP)
print("True Negative(TN):",TN)
print("False Negative(FN):",FN)
accuracy=(TP+TN)/(TP+FP+TN+FN)
print("Accuracy of the binary classifier = {:0.3f}".format(accuracy))


# Experiment 7
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

# Load dataset
iris_data = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")

# Remove duplicates
iris_data_no_duplicates = iris_data.drop_duplicates()

# Split dataset into features and target variable
X = iris_data_no_duplicates.drop(columns=['species'])
y = iris_data_no_duplicates['species']

# Convert categorical labels into numerical values
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Logistic Regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Compute Bias (1 - Accuracy)
bias = 1 - accuracy_score(y_test, y_pred)

# Compute Variance using Cross-Validation
cv_scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
variance = np.var(cv_scores)

# Print results
print("Bias (1 - Accuracy):", bias)
print("Variance (Cross-validation variance):", variance)
print("Cross-validation Accuracy:", cv_scores.mean())



# Experiment 8
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# Sample employee data
data = {
    'Employee id': [10, 20, 15, 25, 30],
    'Gender': ['M', 'F', 'F', 'M', 'F'],
    'Remarks': ['Good', 'Nice', 'Good', 'Great', 'Nice'],
}

# Create DataFrame
df = pd.DataFrame(data)
print(f"Employee data:\n{df}\n")

# Extract categorical columns
categorical_columns = df.select_dtypes(include=['object']).columns.tolist()

# Initialize OneHotEncoder
encoder = OneHotEncoder(sparse_output=False)

# Apply one-hot encoding
one_hot_encoded = encoder.fit_transform(df[categorical_columns])

# Create DataFrame with encoded columns
one_hot_df = pd.DataFrame(one_hot_encoded, columns=encoder.get_feature_names_out(categorical_columns))
```

```
# Concatenate original and encoded data
df_encoded = pd.concat([df, one_hot_df], axis=1)

# Drop original categorical columns
df_encoded = df_encoded.drop(categorical_columns, axis=1)

# Print encoded data
print(f"Encoded Employee data:\n{df_encoded}")



# Experiment 9
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Sample data
data = {
    'Color': ['Red', 'Blue', 'Green', 'Red', 'Green'],
    'Size': ['Small', 'Large', 'Medium', 'Medium', 'Small'],
    'Shape': ['Circle', 'Square', 'Triangle', 'Circle', 'Square'],
    'Label': ['A', 'B', 'C', 'A', 'B']
}

# Create DataFrame
df = pd.DataFrame(data)
print("Original dataset:")
print(df)

# Initialize LabelEncoder
label_encoder = LabelEncoder()
df_encoded = df.copy()

# Apply Label Encoding to categorical columns
for col in df.columns:
    if df[col].dtype == 'object':  # Ensure encoding is applied only to categorical columns
        df_encoded[col] = label_encoder.fit_transform(df[col])

print("\nAfter Label Encoding:")
print(df_encoded)



# Experiment 10
import numpy as np

# Input and output datasets
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)

# Feature scaling (Normalizing inputs)
X = X / np.amax(X, axis=0)  # Normalize X
y = y / 100  # Normalize y

# Activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def derivatives_sigmoid(x):
    return x * (1 - x)

# Neural Network parameters
epoch = 5000      # Number of iterations
lr = 0.1          # Learning rate
input_neurons = 2  # Number of input neurons
hidden_neurons = 3 # Number of hidden neurons
output_neurons = 1 # Number of output neurons

# Weight and bias initialization (random values)
wh = np.random.uniform(size=(input_neurons, hidden_neurons))
bh = np.random.uniform(size=(1, hidden_neurons))
wout = np.random.uniform(size=(hidden_neurons, output_neurons))
bout = np.random.uniform(size=(1, output_neurons))

# Training process
for i in range(epoch):
    # Forward Propagation
    hidden_layer_input = np.dot(X, wh) + bh
    hidden_layer_activation = sigmoid(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_activation, wout) + bout
    output = sigmoid(output_layer_input)

    # Backpropagation
    error_output = y - output
    output_gradient = derivatives_sigmoid(output)
    d_output = error_output * output_gradient

    error_hidden = d_output.dot(wout.T)
    hidden_gradient = derivatives_sigmoid(hidden_layer_activation)
    d_hidden = error_hidden * hidden_gradient

    # Updating Weights and Biases
    wout += hidden_layer_activation.T.dot(d_output) * lr
    wh += X.T.dot(d_hidden) * lr

# Final output
print("Input: \n", X)
print("Actual Output: \n", y)
print("Predicted Output: \n", output)



# Experiment 11
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets

# Load the iris dataset
iris = datasets.load_iris()

# Splitting dataset into training and testing sets (10% test data)
x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.1, random_state=42)

# Displaying label names
print("Iris Flower Labels:")
for i in range(len(iris.target_names)):
    print(f"Label {i} - {iris.target_names[i]}")

# K-NN Classifier (Using K=2)
k = 2  # Number of neighbors
classifier = KNeighborsClassifier(n_neighbors=k)
classifier.fit(x_train, y_train)

# Predicting the test set
y_pred = classifier.predict(x_test)

# Displaying Classification Results
print(f"\nResults of Classification using K-NN with K={k}:")
for i in range(len(x_test)):
    print(f"Sample: {x_test[i]} | Actual Label: {y_test[i]} ({iris.target_names[y_test[i]]}) | "
          f"Predicted Label: {y_pred[i]} ({iris.target_names[y_pred[i]]})")

# Classification Accuracy
accuracy = classifier.score(x_test, y_test)
print(f"\nClassification Accuracy: {accuracy:.2f}")



# Experiment 12
from math import ceil
import numpy as np
import matplotlib.pyplot as plt

def lowess(x, y, f, iterations):
    n = len(x)
    r = int(ceil(f * n))  # Number of nearest neighbors
    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]

    # Compute weights
    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
    w = (1 - w ** 3) ** 3  # Apply tricube weight function

    yest = np.zeros(n)  # Smoothed values
    delta = np.ones(n)  # Robustness weights

    for _ 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 = np.linalg.solve(A, b)  # Solve for beta
            yest[i] = beta[0] + beta[1] * x[i]
        residuals = y - yest
        s = np.median(np.abs(residuals))  # Median absolute deviation
        delta = np.clip(residuals / (6.0 * s), -1, 1)
        delta = (1 - delta ** 2) ** 2  # Apply robustness function

    return yest

# Generate sample data
n = 100
x = np.linspace(0, 2 * np.pi, n)
y = np.sin(x) + 0.3 * np.random.randn(n)  # Add noise

# Apply LOWESS smoothing
f = 0.25  # Smoothing factor
iterations = 3
yest = lowess(x, y, f, iterations)

# Plot results
plt.figure(figsize=(8, 5))
plt.scatter(x, y, color='red', label="Noisy Data", alpha=0.6)
plt.plot(x, yest, color='blue', linewidth=2, label="LOWESS Smoothed Curve")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("LOWESS Smoothing on Noisy Sinusoidal Data")
plt.legend()
plt.show()
```