1) Find S

```python
import pandas as pd

num_attributes = 6
a = []
df = pd.read_excel("E:\me\sem5\AIML\lab\prog1.csv.xlsx", header=None)
a = df.values.tolist()

for row in a:
    print(row)

print("The initial value of hypothesis\n")
hypothesis = ['0'] * num_attributes
print(hypothesis)

for j in range(0, num_attributes):
    hypothesis[j] = a[0][j]

print("Find S: Finding a maximally 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 instance no:{0} the hypothesis is:".format(i), hypothesis)

print("\nThe Maximally Specific Hypothesis for a given training Example:\n")
print(hypothesis)

```

2) Candidate elimination

```
import pandas as pd

df = pd.read_csv('/home/sahyadri/4SF21CS164/Prog2 dataset.csv')

df = df.drop(['slno'], axis=1)

concepts = df.iloc[:, :-1].values
target = df.iloc[:, -1].values

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

    general_h = [row for row in general_h if '?' not in row]

    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print(f"Final S: {s_final}")
print(f"Final G: {g_final}")
```

3) ID3

4) Backpropogation-ANN

```
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 = 1000
learning_rate = 0.6
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(f"Input:\n{x}")
print(f"Actual Output:\n{y}")
print(f"Predicted output:\n{output}")
```

5) KNN

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets

iris = datasets.load_iris()
print("Iris Data set loaded...")
x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.1)

for i in range(len(iris.target_names)):
    print("Label", i , "-", str(iris.target_names[i]))

clf = KNeighborsClassifier(n_neighbors=2)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)

print("Results of Classification using K-nn with K=1 ")

for r in range(0, len(x_test)):
    print("Sample:", str(x_test[r]), "Actual-label:", str(y_test[r]), "Predicted-label:", str(y_pred[r]))

print("Classification Accuracy:", clf.score(x_test, y_test))
```

acbbbdbd

6) Naïve Bayes

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import datasets
from sklearn.metrics import accuracy_score, classification_report

iris = datasets.load_iris()

x_train,x_test, y_train, y_test = train_test_split(iris.data, iris.target,test_size=0.2)

clf = GaussianNB()
clf.fit(X_train, y_train)
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Acbdd

7) KMeans

8)Locally weighted regression

```
import numpy as np
import matplotlib.pyplot as plt

def local_regression(x0, x, y, tau):
    x0 = [1, x0]
    x = [[1, i] for i in x]
    x = np.asarray(x)
    w = (x.T) * np.exp(np.sum((x - x0) ** 2, axis=1) / (-2 * tau))
    beta = np.linalg.pinv(w @ x) @ w @ y @ x0
    return beta

def draw(tau):
    prediction = [local_regression(x0, x, y, tau) for x0 in domain]
    plt.plot(x, y, 'o', color='black')
    plt.plot(domain, prediction, color='red')
    plt.show()

x = np.linspace(-3, 3, num=1000)
domain = x
y = np.log(np.abs(x ** 2 - 1) + .5)

draw(10)
draw(0.1)
draw(0.01)
draw(0.001)
```