1. Find S Code

```
import pandas as pd
import numpy as np
H = [0] * 6
df = pd.read_csv("finds.csv", header=None)
attributes = df.iloc[:, :-1].values
target = df.iloc[:, -1].values
for i in range(len(target)):
    if target[i] == "Yes":
        for j in range(len(attributes[i])):
            if H[j] == 0:
                H[j] = attributes[i][j]
            elif H[j] != attributes[i][j]:
                H[j] = '?'
print(H)
```

2. Candidate elimination

```
import pandas as pd
df=pd.read_csv("finds.csv",sep=",",header=None)
#Intialize S and G
S=[0,0,0,0,0,0]
G=list()
for i in range(len(df.columns)-1):
    G.append(['?','?','?','?','?','?'])
#Read samples
for i in range(len(df)):
    for j in range(len(df.columns)-1):
        if df.iloc[i,-1]=="Yes":
            if S[j]==0:
                S[j]=df.iloc[i,j]
            elif df.iloc[i,j]!=S[j]:
                S[j]="?"
            if G[j][j]!='?' and S[j]=='?':
                G[j][j]='?'
        else:
            if df.iloc[i,j]!=S[j] and S[j]!='?':
                G[j][j]=S[j]
print(S)
```

```
print(G)
```

```
3.Decision Tree Classifier
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn import tree
df = pd.read_csv("decision_tree1.csv")
df = df.apply(lambda col: col.astype('category').cat.codes if
col.dtype == 'object' else col)
x = df.iloc[:, :-1]
y = df.iloc[:, -1]
clf = DecisionTreeClassifier(criterion="entropy").fit(x, y)
plt.figure(figsize=(6,4))
tree.plot_tree(clf, feature_names=x.columns, filled=True)
plt.show()
print(f"Number of features in training data: {x.shape[1]}")
new_data = [[1, 0, 0, 1]]
print(clf.predict(new_data))
```

```
4. Naive Bayes
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
df=pd.read_csv("iris1.csv")
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
y=y.astype('category')
y=y.cat.codes
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3)
g=GaussianNB()
g.fit(xtrain,ytrain)
ypred=g.predict(xtest)
print(accuracy_score(ytest,ypred))
```

```
5.Bayesian Belief Network
import pandas as pd
import numpy as np
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
df = pd.read_csv("Medical Dataset.csv").replace("?", np.nan)
model = BayesianModel([
    ('age', 'heartdisease'),
    ('sex', 'heartdisease'),
    ('exang', 'heartdisease'),
    ('cp', 'heartdisease'),
    ('heartdisease', 'restecg'),
    ('heartdisease', 'chol')
])
model.fit(df, estimator=MaximumLikelihoodEstimator)
infer = VariableElimination(model)
result = infer.query(variables=['heartdisease'], evidence={'restecg':
1})
print(result)
```

```
6. EM Algorithm
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
df=pd.read_csv('Iris.csv')
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
colormap=np.array(["red","green","blue"])
y=y.astype('category')
y=y.cat.codes
gm=GaussianMixture(n_components=3)
gm.fit(x)
gmc=gm.predict(x)
km=KMeans(n_clusters=3)
km.fit(x)
kmc=km.predict(x)
import matplotlib.pyplot as plt
```

```
plt.subplot(1,3,1)
plt.scatter(x.iloc[:,0],x.iloc[:,1],c=colormap[y],s=40)
plt.subplot(1,3,2)
plt.scatter(x.iloc[:,0],x.iloc[:,1],c=colormap[gmc],s=40)
plt.subplot(1,3,3)
plt.scatter(x.iloc[:,0],x.iloc[:,1],c=colormap[kmc],s=40)
plt.show()
```

```
7. KNN Algorithm
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
df=pd.read_csv('Iris.csv')
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
y=y.astype('category')
y=y.cat.codes
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3)
kn=KNeighborsClassifier(n_neighbors=3)
kn.fit(xtrain,ytrain)
ypred=kn.predict(xtest)
i=0
for label in ytest:
    if label==ypred[i]:
        print('Correct',label)
    else:
        print('Incorrect',label,ypred[i])
```

```
8. Regression
import numpy as np
import matplotlib.pyplot as plt
def locally_weighted_regression(x_query, X, y, tau=0.1):
    X = np.array(X)
    y = np.array(y)
    x_query = np.array(x_query)
    kernel_weights = np.exp(-(X - x_query)**2 / (2 * tau**2))
```

```
    W = np.diag(kernel_weights)
    X_design = np.vstack([X, np.ones_like(X)]).T
    theta = np.linalg.inv(X_design.T @ W @ X_design) @ (X_design.T @
W @ y)
    y_query = np.array([x_query, 1]).T @ theta
    return y_query
np.random.seed(0)
X = np.linspace(0, 10, 100)
y = np.sin(X) + np.random.normal(0, 0.1, X.shape)
x_queries = np.linspace(0, 10, 100)
y_pred = [locally_weighted_regression(x, X, y, tau=0.5) for x in
x_queries]
plt.figure(figsize=(6, 4))
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(x_queries, y_pred, color='red', label='LWR Fit (tau=0.5)')
plt.title('Locally Weighted Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.show()
```

```
9. Support Vector Machine
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
iris=pd.read_csv("Iris.csv")
x=iris.iloc[:,:-1]
y=iris.iloc[:,-1]
y=y.astype('category')
y=y.cat.codes
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3)
svmclf=make_pipeline(StandardScaler(),LinearSVC(C=15))
svmclf.fit(xtrain,ytrain)
ypred=(svmclf.predict(xtest)) acc=accuracy_score(ytest,ypred)
print(acc)
```

```
10. Back Propagation
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=5
lr=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))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
for i in range(epoch):
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+bout
    output = sigmoid(outinp)
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    print ("-----------Epoch-", i+1, "Starts----------")
```

```
    print("Input: \n" + str(X))
    print("Actual Output: \n" + str(y))
    print("Predicted Output: \n" ,output)
    print ("-----------Epoch-", i+1, "Ends----------\n")
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```