finds

```
import pandas as pd
import numpy as np
df= pd.read_csv('en.csv')
df=df[df[df.columns[-1]]==1]
print(df)
d=np.array(df)
print(d)
def FindS(df):
  h=['pi']*(len(df.columns)-1)
  print(h)
  for i in d:
    if 'pi' in h:
      for j in range(len(h)):
        h[j]=i[j]
    else:
      for j in range(len(h)):
        if h[j]!=i[j]:
          h[j]='?'
    print(h)

FindS(df)
```

Candidate-Elimination

```
import pandas as pd
import numpy as np
p = pd.read_csv('en.csv')
print(p)
#p.drop(p.columns[0], axis=1, inplace=True)
n = np.array(p)
h = n[0]
G = []
l1=[]
for i in n:
  if i[len(h) - 1] == 1:
    for j in range(len(i)):
      if h[j] != i[j]:
        h[j] = '?'
  elif i[len(h) - 1] == 0:
    for j in range(len(i)-1):
      l = ['?'] * len(h)
      if h[j] != i[j] and h[j] != '?':
        l[j] = h[j]
        print(l)
        l1.append(l)
    G.append(l1)
print("Specific hypothesis",h)
print("\n")
print("General Hypothesis",G
```

Navie- Bayesian classifier

```
import pandas as pd
import numpy as np
from sklearn import metrics
df=pd.read_csv("pt.csv")
print(df)
from sklearn import preprocessing
string_int=preprocessing.LabelEncoder()
df=df.apply(string_int.fit_transform)
print(df)
x=df[['outlook','temp','humidity','wind']]
y=df['play']
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.1,random_state=1)
from sklearn.naive_bayes import GaussianNB
k=GaussianNB()
k.fit(xtrain,ytrain)
p=k.predict(xtest)
#initial accuracy
from sklearn.metrics import accuracy_score
print("accuracy is:",accuracy_score(p,ytest))

fp=k.predict([[1,1,0,1]])
print("the final label is:",fp)
```

Bayesian classifier model using Built-in Java classes /API

Text-Classifier

```
import pandas as pd
msg = pd.read_csv('text.csv', names=['message', 'label'])
msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})
print(msg)

x = msg.message
y = msg.labelnum
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(x, y)
print(Xtrain)
print(ytrain)

from sklearn.feature_extraction.text import CountVectorizer
count_v = CountVectorizer()
Xtrain_dm =count_v.fit_transform(Xtrain)
Xtest_dm = count_v.transform(Xtest)
print(count_v.get_feature_names_out())

from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(Xtrain_dm, ytrain)
pred = clf.predict(Xtest_dm)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,
precision_score, recall_score

print('Accuracy Metrics: \n')

print('Accuracy: ', accuracy_score(ytest, pred))

print('Recall: ', recall_score(ytest, pred))

print('Precision: ', precision_score(ytest, pred))

print('Confusion Matrix: \n', confusion_matrix(ytest, pred))
```

Bayesian Network – Heart using API

```
import pandas as pd

import numpy as np

h=pd.read_csv('heart.csv')

print(h)

h=h.replace('?',np.nan)

from pgmpy.models import BayesianNetwork

model=BayesianNetwork([('age','target'),('sex','target'),('trestbps','target'),('exa
ng','target'),('restecg','target')])

from pgmpy.estimators import MaximumLikelihoodEstimator

model.fit(h,estimator= MaximumLikelihoodEstimator)

from pgmpy.inference import VariableElimination

infer=VariableElimination(model)

q1=infer.query(variables=['target'],evidence={'restecg':2})

print(q1)
```

BB-prop

```
import numpy as np
# Define the sigmoid function
def sigmoid(x):
  return 1 / (1 + np.exp(-x))
# Derivative of the sigmoid function
def ds(x):
  return x * (1 - x)
# Input data
x = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
# Normalize input and output
x = x / np.amax(x, axis=0)
y = y / 100
# Parameters
epoch = 5
lr = 0.1
iln = 2
oln = 1
hln = 1
# Initialize weights and biases
wh = np.random.uniform(size=(iln, hln))
bh = np.random.uniform(size=(1, hln))
wout = np.random.uniform(size=(hln, oln))
bout = np.random.uniform(size=(1, oln))
# Training loop
```

```
for i in range(epoch):
# Forward pass
  hinp = np.dot(x, wh) + bh
  hlayer_act = sigmoid(hinp)
  oinp = np.dot(hlayer_act, wout)
  output = sigmoid(oinp)
  # Backpropagation
  eo = y - output
  o_grad = ds(output)
  d_output = eo * o_grad

  eh = d_output.dot(wout.T)
  h_grad = ds(hlayer_act)
  d_hidden = eh * h_grad
  # Update weights and biases
  wh += x.T.dot(d_hidden) * lr
  wout += hlayer_act.T.dot(d_output) * lr
  # Print outputs for each epoch
  print("Epoch:", i + 1)
  print("Expected Output:")
  print(y)
  print("Predicted Output:")
  print(output)
  print()
```

EM & K-Means

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt


# Load the Iris dataset

df = pd.read_csv("Iris.csv")

# Define features and labels

x = df[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm",
"PetalWidthCm"]]

label = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in df["Species"]]

# KMeans clustering

from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3).fit(x)

kmeans_colors = np.array(["blue", "yellow", "green"])

# Plot KMeans clusters

plt.scatter(df["PetalLengthCm"], df["PetalWidthCm"],
c=kmeans_colors[kmeans.labels_])

plt.title("KMeans Clustering")

plt.xlabel("Petal Length")

plt.ylabel("Petal Width")

plt.show()

# Gaussian Mixture Model (GMM)

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=3, random_state=0).fit(x)

gmm_prediction = gmm.predict(x)
```

```
# Plot GMM classification
plt.scatter(df["PetalLengthCm"], df["PetalWidthCm"],
c=kmeans_colors[gmm_prediction])
plt.title("GMM Classification")
plt.xlabel("Petal Length")
plt.ylabel("Petal Width")
plt.show()
# Calculate accuracies
from sklearn import metrics
accuracy_kmeans = metrics.accuracy_score(y, kmeans.labels_)
accuracy_gmm = metrics.accuracy_score(y, gmm_prediction)

print("Accuracy of KMeans:", accuracy_kmeans)
print("Accuracy of GMM:", accuracy_gmm)
```

KNN

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import numpy as np
i=pd.read_csv("Iris.csv")
print(i)
x=i[["SepalLengthCm","SepalWidthCm","PetalLengthCm","PetalWidthCm"]]
y=i["Species"]
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2)
k=KNeighborsClassifier(n_neighbors=1)
k.fit(xtrain,ytrain)
p=k.predict(xtest)
ytestt=np.array(ytest)
for i in range(len(ytestt)):
  print("the actual is:",ytestt[i]," ","the predicted is:",p[i])
```

Local Weighted regression

```
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-")
```