# Practical : 1

**Aim:** Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

**Data Set:**

| Example | Sky | AirTemp | Humidity | Wind | Water | Forecast | EnjoySport |
|---|---|---|---|---|---|---|---|
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High | Strong | Warm | Change | No |
| 4 | Sunny | Warm | High | Strong | Cool | Change | Yes |

In [1]:
```
import pandas as pd
import numpy as np

#to read the data in the csv file
data = pd.read_csv("lab1.csv")
print(data)
```

```
     Sky AirTemp Humidity    Wind Water Forecast EnjoySport
0  Sunny    Warm   Normal  Strong  Warm     Same        Yes
1  Sunny    Warm     High  Strong  Warm     Same        Yes
2  Rainy    Cold     High  Strong  Warm   Change         No
3  Sunny    Warm     High  Strong  Cool   Change        Yes
```

In [2]:
```
#making an array of all the attributes
d = np.array(data)[:,:-1]
print("The attributes are: ",d)
```

```
The attributes are:  [['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

In [3]:
```
#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print(" The target is: ",target)
```

```
 The target is:  ['Yes' 'Yes' 'No' 'Yes']
```

In [4]:
```
#training function to implement find-s algorithm
def train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis
```

In [5]:
```
print(" The final hypothesis is:",train(d,target))
```

```
 The final hypothesis is: ['Sunny' 'Warm' '?' 'Strong' '?' '?']
```

# Practical : 2

**Aim:** For a given set of training data examples stored in a .CSV file, implement and demonstratethe Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

**Data Set:**

| Example | Sky | AirTemp | Humidity | Wind | Water | Forecast | EnjoySport |
|---|---|---|---|---|---|---|---|
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High | Strong | Warm | Change | No |
| 4 | Sunny | Warm | High | Strong | Cool | Change | Yes |

In [1]:
```
import pandas as pd
import numpy as np
data=pd.read_csv('lab1.csv')
print(data)
```

```
     Sky AirTemp Humidity    Wind Water Forecast EnjoySport
0  Sunny    Warm   Normal  Strong  Warm     Same        Yes
1  Sunny    Warm     High  Strong  Warm     Same        Yes
2  Rainy    Cold     High  Strong  Warm   Change         No
3  Sunny    Warm     High  Strong  Cool   Change        Yes
```

In [2]:
```
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
print(target)
```

```
['Yes' 'Yes' 'No' 'Yes']
```

In [3]:
```
print(concepts)
```

```
[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

In [4]:
```
def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))] for i 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] = '?'

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
```

In [5]:
```
s_final, g_final = learn(concepts, target)

print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

```
Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']
Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

# Practical : 3

**Aim:** Write a program to demonstrate the working of the decision tree-based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

**Data Set:**

| Day | *Outlook* | *Temperature* | *Humidity* | *Wind* | *PlayTennis* |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Weak | Yes |
| D11 | Sunny | Mild | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

In [1]:
```
import pandas as pd
import numpy as np
data=pd.read_csv('PlayTennis.csv')
data
```

Out[1]:

| | PlayTennis | Outlook | Temperature | Humidity | Wind |
|---|---|---|---|---|---|
| 0 | No | Sunny | Hot | High | Weak |
| 1 | No | Sunny | Hot | High | Strong |
| 2 | Yes | Overcast | Hot | High | Weak |
| 3 | Yes | Rain | Mild | High | Weak |
| 4 | Yes | Rain | Cool | Normal | Weak |
| 5 | No | Rain | Cool | Normal | Strong |
| 6 | Yes | Overcast | Cool | Normal | Strong |
| 7 | No | Sunny | Mild | High | Weak |
| 8 | Yes | Sunny | Cool | Normal | Weak |
| 9 | Yes | Rain | Mild | Normal | Weak |
| 10 | Yes | Sunny | Mild | Normal | Strong |
| 11 | Yes | Overcast | Mild | High | Strong |
| 12 | Yes | Overcast | Hot | Normal | Weak |
| 13 | No | Rain | Mild | High | Strong |

In [2]:
```
def entropy(probs):
    import math
    return sum( [-prob*math.log(prob, 2) for prob in probs] )
```

In [3]:
```
#Function to calulate the entropy of the given Data Sets/List with respect to target attributes
def entropy_of_list(a_list):
    #print("A-list",a_list)
    from collections import Counter
    cnt = Counter(x for x in a_list)   # Counter calculates the propotion of class
   # print("\nClasses:",cnt)
    #print("No and Yes Classes:",a_list.name,cnt)
    num_instances = len(a_list)*1.0   # = 14
   # print("\n Number of Instances of the Current Sub Class is {0}:".format(num_instances ))
    probs = [x / num_instances for x in cnt.values()]  # x means no of YES/NO
##    print("\n Classes:",min(cnt),max(cnt))
 #   print(" \n Probabilities of Class {0} is {1}:".format(min(cnt),min(probs)))
 #   print(" \n Probabilities of Class {0} is {1}:".format(max(cnt),max(probs)))
    return entropy(probs) # Call Entropy :

# The initial entropy of the YES/NO attribute for our dataset.
print("\n  INPUT DATA SET FOR ENTROPY CALCULATION:\n", data['PlayTennis'])

total_entropy = entropy_of_list(data['PlayTennis'])

print("\n Total Entropy of PlayTennis Data Set:",total_entropy)
```

```
  INPUT DATA SET FOR ENTROPY CALCULATION:
 0      No
1       No
2      Yes
3      Yes
4      Yes
5       No
6      Yes
7       No
8      Yes
9      Yes
10     Yes
11     Yes
12     Yes
13      No
Name: PlayTennis, dtype: object

 Total Entropy of PlayTennis Data Set: 0.9402859586706309
```

In [4]:
```
def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
 #   print("Information Gain Calculation of ",split_attribute_name)
    '''
    Takes a DataFrame of attributes, and quantifies the entropy of a target
    attribute after performing a split along the values of another attribute.
    '''
    # Split Data by Possible Vals of Attribute:
    df_split = df.groupby(split_attribute_name)
   # for name,group in df_split:
    #    print("Name:\n",name)
     #   print("Group:\n",group)

    # Calculate Entropy for Target Attribute, as well as
    # Proportion of Obs in Each Data-Split
    nobs = len(df.index) * 1.0
   # print("NOBS",nobs)
    df_agg_ent = df_split.agg({target_attribute_name : [entropy_of_list, lambda x: len(x)/nobs] })[target_attribute_name]
    #print([target_attribute_name])
    #print(" Entropy List ",entropy_of_list)
    #print("DFAGGENT",df_agg_ent)
    df_agg_ent.columns = ['Entropy', 'PropObservations']
    #if trace: # helps understand what fxn is doing:
     #   print(df_agg_ent)

    # Calculate Information Gain:
    new_entropy = sum( df_agg_ent['Entropy'] * df_agg_ent['PropObservations'] )
    old_entropy = entropy_of_list(df[target_attribute_name])
    return old_entropy - new_entropy


print('Info-gain for Outlook is :'+str( information_gain(data, 'Outlook', 'PlayTennis')),"\n")
print('\n Info-gain for Humidity is: ' + str( information_gain(data, 'Humidity', 'PlayTennis')),"\n")
print('\n Info-gain for Wind is:' + str( information_gain(data, 'Wind', 'PlayTennis')),"\n")
print('\n Info-gain for Temperature is:' + str( information_gain(data, 'Temperature','PlayTennis')),"\n")
```

```
Info-gain for Outlook is :0.2467498197744391


 Info-gain for Humidity is: 0.15183550136234136


 Info-gain for Wind is:0.04812703040826927


 Info-gain for Temperature is:0.029222565658954647
```

In [5]:
```python
def id3(df, target_attribute_name, attribute_names, default_class=None):

    ## Tally target attribute:
    from collections import Counter
    cnt = Counter(x for x in df[target_attribute_name])# class of YES /NO

    ## First check: Is this split of the dataset homogeneous?
    if len(cnt) == 1:
        return next(iter(cnt))  # next input data set, or raises StopIteration when EOF is hit.

    ## Second check: Is this split of the dataset empty?
    # if yes, return a default value
    elif df.empty or (not attribute_names):
        return default_class  # Return None for Empty Data Set

    ## Otherwise: This dataset is ready to be devied up!
    else:
        # Get Default Value for next recursive call of this function:
        default_class = max(cnt.keys()) #No of YES and NO Class
        # Compute the Information Gain of the attributes:
        gainz = [information_gain(df, attr, target_attribute_name) for attr in attribute_names] #
        index_of_max = gainz.index(max(gainz)) # Index of Best Attribute
        # Choose Best Attribute to split on:
        best_attr = attribute_names[index_of_max]

        # Create an empty tree, to be populated in a moment
        tree = {best_attr:{}} # Iniiate the tree with best attribute as a node
        remaining_attribute_names = [i for i in attribute_names if i != best_attr]

        # Split dataset
        # On each split, recursively call this algorithm.
        # populate the empty tree with subtrees, which
        # are the result of the recursive call
        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset,
                        target_attribute_name,
                        remaining_attribute_names,
                        default_class)
            tree[best_attr][attr_val] = subtree
        return tree
```

In [6]:
```python
# Get Predictor Names (all but 'class')
attribute_names = list(data.columns)
print("List of Attributes:", attribute_names)
attribute_names.remove('PlayTennis') #Remove the class attribute
print("Predicting Attributes:", attribute_names)
```

```
List of Attributes: ['PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind']
Predicting Attributes: ['Outlook', 'Temperature', 'Humidity', 'Wind']
```

In [7]:
```python
# Run Algorithm:
from pprint import pprint
tree = id3(data,'PlayTennis',attribute_names)
print("\n\nThe Resultant Decision Tree is :\n")
#print(tree)
pprint(tree)
attribute = next(iter(tree))
print("Best Attribute :\n",attribute)
print("Tree Keys:\n",tree[attribute].keys())
```

```


The Resultant Decision Tree is :

{'Outlook': {'Overcast': 'Yes',
             'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
Best Attribute :
 Outlook
Tree Keys:
 dict_keys(['Overcast', 'Rain', 'Sunny'])
```

# Practical : 4

**Aim:** Write a program to demonstrate the working of the decision tree-based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
In [1]: 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) # maximum of X array longitudinally
y = y/100
#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=5000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer

#weight and bias initialization
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))

#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
#Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)

#Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)
#how much hidden layer wts contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad
# dotproduct of nextlayererror and currentlayerop
    wout += hlayer_act.T.dot(d_output) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89409392]
 [0.87998279]
 [0.89511991]]
```

# Practical : 5

**Aim:** Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

## Data Set:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Outlook | Temperat | Humidity | Windy | PlayTennis |
| 2 | Sunny | Hot | High | FALSE | No |
| 3 | Sunny | Hot | High | TRUE | No |
| 4 | Overcast | Hot | High | FALSE | Yes |
| 5 | Rainy | Mild | High | FALSE | Yes |
| 6 | Rainy | Cool | Normal | FALSE | Yes |
| 7 | Rainy | Cool | Normal | TRUE | No |
| 8 | Overcast | Cool | Normal | TRUE | Yes |
| 9 | Sunny | Mild | High | FALSE | No |
| 10 | Sunny | Cool | Normal | FALSE | Yes |
| 11 | Rainy | Mild | Normal | FALSE | Yes |
| 12 | Sunny | Mild | Normal | TRUE | Yes |
| 13 | Overcast | Mild | High | TRUE | Yes |
| 14 | Overcast | Hot | Normal | FALSE | Yes |
| 15 | Rainy | Mild | High | TRUE | No |

In [1]:

```
# import necessary libarities
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

# load data from CSV
data = pd.read_csv('pr5.csv')
print("THe first 5 values of data is :\n",data.head())

# obtain Train data and Train output
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())

# Convert then in numbers
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())
```

```python
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))
```

```
THe first 5 values of data is :
    Outlook Temperature Humidity  Windy PlayTennis
0     Sunny         Hot     High  False         No
1     Sunny         Hot     High   True         No
2  Overcast         Hot     High  False        Yes
3     Rainy        Mild     High  False        Yes
4     Rainy        Cool   Normal  False        Yes

The First 5 values of train data is
    Outlook Temperature Humidity  Windy
0     Sunny         Hot     High  False
1     Sunny         Hot     High   True
2  Overcast         Hot     High  False
3     Rainy        Mild     High  False
4     Rainy        Cool   Normal  False

The first 5 values of Train output is
 0     No
1     No
2    Yes
3    Yes
4    Yes
Name: PlayTennis, dtype: object

Now the Train data is :
   Outlook  Temperature  Humidity  Windy
0        2            1         0      0
1        2            1         0      1
2        0            1         0      0
3        1            2         0      0
4        1            0         1      0

Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Accuracy is: 0.6666666666666666
```

# Practical : 6

**Aim:** Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

## Data Set:

| | A | B |
|---|---|---|
| 1 | I love this sandwich | pos |
| 2 | This is an amazing place | pos |
| 3 | I feel very good about these beers | pos |
| 4 | This is my best work | pos |
| 5 | What an awesome view | pos |
| 6 | I do not like this restaurant | neg |
| 7 | I am tired of this stuff | neg |
| 8 | I can't deal with this | neg |
| 9 | He is my sworn enemy | neg |
| 10 | My boss is horrible | neg |
| 11 | This is an awesome place | pos |
| 12 | I do not like the taste of this juice | neg |
| 13 | I love to dance | pos |
| 14 | I am sick and tired of this place | neg |
| 15 | What a great holiday | pos |
| 16 | That is a bad locality to stay | neg |
| 17 | We will have good fun tomorrow | pos |
| 18 | I went to my enemy's house today | neg |

[2]:
```
import pandas as pd
msg = pd.read_csv('pr6.csv', names=['message', 'label'])
print("Total Instances of Dataset: ", msg.shape[0])
msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})
```

```
Total Instances of Dataset:  18
```

[3]:
```
X = msg.message
y = msg.labelnum
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y)
from sklearn.feature_extraction.text import CountVectorizer

count_v = CountVectorizer()
Xtrain_dm = count_v.fit_transform(Xtrain)
Xtest_dm = count_v.transform(Xtest)
```

[5]:
```
df = pd.DataFrame(Xtrain_dm.toarray(), columns=count_v.get_feature_names_out())
print(df[0:5])
```

```
   about  am  an  and  awesome  bad  beers  best  boss  do  ...  to  today  \
0      0   0   0    0        0    0      0     0     0   0  ...   0      0
1      0   0   0    0        0    1      0     0     0   0  ...   1      0
2      1   0   0    0        0    0      1     0     0   0  ...   0      0
3      0   0   0    0        0    0      0     0     0   0  ...   0      0
4      0   1   0    1        0    0      0     0     0   0  ...   0      0

   tomorrow  very  view  we  went  what  will  work
0         1     0     0   1     0     0     1     0
1         0     0     0   0     0     0     0     0
2         0     1     0   0     0     0     0     0
3         0     0     0   0     0     0     0     0
4         0     0     0   0     0     0     0     0

[5 rows x 46 columns]
```

[6]:
```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(Xtrain_dm, ytrain)
pred = clf.predict(Xtest_dm)
```

[9]:
```
for doc, p in zip(Xtest, pred):
    p = 'pos' if p == 1 else 'neg'
    print("%s -> %s" % (doc, p))
```

```
I can't deal with this -> neg
I do not like the taste of this juice -> neg
I love to dance -> neg
This is an amazing place -> pos
What a great holiday -> pos
```

[10]:
```
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))
```

```
Accuracy Metrics:

Accuracy:  0.8
Recall:  0.6666666666666666
Precision:  1.0
Confusion Matrix:
 [[2 0]
 [1 2]]
```

# Practical : 7

**Aim:** Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

**Data Set:**

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | heartdisease |
| 2 | 63 | 1 | 1 | 145 | 233 | 1 | 2 | 150 | 0 | 2.3 | 3 | 0 | 6 | 0 |
| 3 | 67 | 1 | 4 | 160 | 286 | 0 | 2 | 108 | 1 | 1.5 | 2 | 3 | 3 | 2 |
| 4 | 67 | 1 | 4 | 120 | 229 | 0 | 2 | 129 | 1 | 2.6 | 2 | 2 | 7 | 1 |
| 5 | 37 | 1 | 3 | 130 | 250 | 0 | 0 | 187 | 0 | 3.5 | 3 | 0 | 3 | 0 |
| 6 | 41 | 0 | 2 | 130 | 204 | 0 | 2 | 172 | 0 | 1.4 | 1 | 0 | 3 | 0 |
| 7 | 56 | 1 | 2 | 120 | 236 | 0 | 0 | 178 | 0 | 0.8 | 1 | 0 | 3 | 0 |
| 8 | 62 | 0 | 4 | 140 | 268 | 0 | 2 | 160 | 0 | 3.6 | 3 | 2 | 3 | 3 |
| 9 | 57 | 0 | 4 | 120 | 354 | 0 | 0 | 163 | 1 | 0.6 | 1 | 0 | 3 | 0 |
| 10 | 63 | 1 | 4 | 130 | 254 | 0 | 2 | 147 | 0 | 1.4 | 2 | 1 | 7 | 2 |
| 11 | 53 | 1 | 4 | 140 | 203 | 1 | 2 | 155 | 1 | 3.1 | 3 | 0 | 7 | 1 |
| 12 | 57 | 1 | 4 | 140 | 192 | 0 | 0 | 148 | 0 | 0.4 | 2 | 0 | 6 | 0 |
| 13 | 56 | 0 | 2 | 140 | 294 | 0 | 2 | 153 | 0 | 1.3 | 2 | 0 | 3 | 0 |
| 14 | 56 | 1 | 3 | 130 | 256 | 1 | 2 | 142 | 1 | 0.6 | 2 | 1 | 6 | 2 |
| 15 | 44 | 1 | 2 | 120 | 263 | 0 | 0 | 173 | 0 | 0 | 1 | 0 | 7 | 0 |
| 16 | 52 | 1 | 3 | 172 | 199 | 1 | 0 | 162 | 0 | 0.5 | 1 | 0 | 7 | 0 |
| 17 | 57 | 1 | 3 | 150 | 168 | 0 | 0 | 174 | 0 | 1.6 | 1 | 0 | 3 | 0 |
| 18 | 48 | 1 | 2 | 110 | 229 | 0 | 0 | 168 | 0 | 1 | 3 | 0 | 7 | 1 |
| 19 | 54 | 1 | 4 | 140 | 239 | 0 | 0 | 160 | 0 | 1.2 | 1 | 0 | 3 | 0 |
| 20 | 48 | 0 | 3 | 130 | 275 | 0 | 0 | 139 | 0 | 0.2 | 1 | 0 | 3 | 0 |
| 21 | 49 | 1 | 2 | 130 | 266 | 0 | 0 | 171 | 0 | 0.6 | 1 | 0 | 3 | 0 |
| 22 | 64 | 1 | 1 | 110 | 211 | 0 | 2 | 144 | 1 | 1.8 | 2 | 0 | 3 | 0 |
| 23 | 58 | 0 | 1 | 150 | 283 | 1 | 2 | 162 | 0 | 1 | 1 | 0 | 3 | 0 |
| 24 | 58 | 1 | 2 | 120 | 284 | 0 | 2 | 160 | 0 | 1.8 | 2 | 0 | 3 | 1 |
| 25 | 58 | 1 | 3 | 132 | 224 | 0 | 2 | 173 | 0 | 3.2 | 1 | 2 | 7 | 3 |
| 26 | 60 | 1 | 4 | 130 | 206 | 0 | 2 | 132 | 1 | 2.4 | 2 | 2 | 7 | 4 |
| 27 | 50 | 0 | 3 | 120 | 219 | 0 | 0 | 158 | 0 | 1.6 | 2 | 0 | 3 | 0 |
| 28 | 58 | 0 | 3 | 120 | 340 | 0 | 0 | 172 | 0 | 0 | 1 | 0 | 3 | 0 |
| 29 | 66 | 0 | 1 | 150 | 226 | 0 | 0 | 114 | 0 | 2.6 | 3 | 0 | 3 | 0 |
| 30 | 43 | 1 | 4 | 150 | 247 | 0 | 0 | 171 | 0 | 1.5 | 1 | 0 | 3 | 0 |

heart.csv

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination

heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())
```

```
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   1       145   233    1        2      150      0      2.3      3
1   67    1   4       160   286    0        2      108      1      1.5      2
2   67    1   4       120   229    0        2      129      1      2.6      2
3   37    1   3       130   250    0        0      187      0      3.5      3
4   41    0   2       130   204    0        2      172      0      1.4      1

  ca thal  heartdisease
0  0    6             0
1  3    3             2
2  2    7             1
3  0    3             0
4  0    3             0
```

```
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

```
 Attributes and datatypes
age                 int64
sex                 int64
cp                  int64
trestbps            int64
chol                int64
fbs                 int64
restecg             int64
thalach             int64
exang               int64
oldpeak           float64
slope               int64
ca                 object
thal               object
heartdisease        int64
dtype: object
```

```
model= BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),
 ('heartdisease','restecg'),('heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
```

```
Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg
+----------------+--------------------+
| heartdisease   |   phi(heartdisease) |
+================+====================+
| heartdisease(0) |             0.1016 |
+----------------+--------------------+
| heartdisease(1) |             0.0000 |
+----------------+--------------------+
| heartdisease(2) |             0.2361 |
+----------------+--------------------+
| heartdisease(3) |             0.2017 |
+----------------+--------------------+
| heartdisease(4) |             0.4605 |
+----------------+--------------------+
```

```
print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

```
 2. Probability of HeartDisease given evidence= cp
+----------------+--------------------+
| heartdisease   |   phi(heartdisease) |
+================+====================+
| heartdisease(0) |             0.3742 |
+----------------+--------------------+
| heartdisease(1) |             0.2018 |
+----------------+--------------------+
| heartdisease(2) |             0.1375 |
+----------------+--------------------+
| heartdisease(3) |             0.1541 |
+----------------+--------------------+
| heartdisease(4) |             0.1323 |
+----------------+--------------------+
```

# Practical : 8

**Aim:** Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

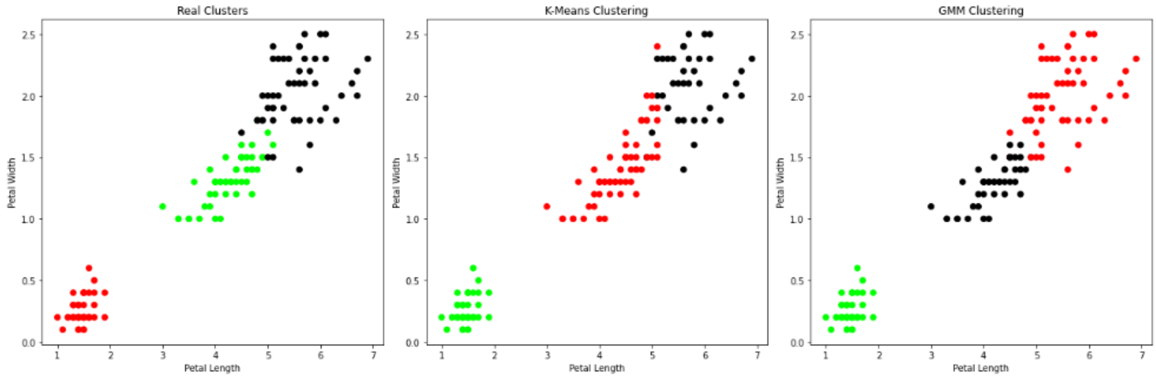
**Data Set:**

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Id | SepalLeng | SepalWidt | PetalLeng | PetalWidt | Species |
| 2 | 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 3 | 2 | 4.9 | 3 | 1.4 | 0.2 | Iris-setosa |
| 4 | 3 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 5 | 4 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 6 | 5 | 5 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| 7 | 6 | 5.4 | 3.9 | 1.7 | 0.4 | Iris-setosa |
| 8 | 7 | 4.6 | 3.4 | 1.4 | 0.3 | Iris-setosa |
| 9 | 8 | 5 | 3.4 | 1.5 | 0.2 | Iris-setosa |
| 10 | 9 | 4.4 | 2.9 | 1.4 | 0.2 | Iris-setosa |
| 11 | 10 | 4.9 | 3.1 | 1.5 | 0.1 | Iris-setosa |
| 12 | 11 | 5.4 | 3.7 | 1.5 | 0.2 | Iris-setosa |
| 13 | 12 | 4.8 | 3.4 | 1.6 | 0.2 | Iris-setosa |
| 14 | 13 | 4.8 | 3 | 1.4 | 0.1 | Iris-setosa |
| 15 | 14 | 4.3 | 3 | 1.1 | 0.1 | Iris-setosa |
| 16 | 15 | 5.8 | 4 | 1.2 | 0.2 | Iris-setosa |
| 17 | 16 | 5.7 | 4.4 | 1.5 | 0.4 | Iris-setosa |
| 18 | 17 | 5.4 | 3.9 | 1.3 | 0.4 | Iris-setosa |
| 19 | 18 | 5.1 | 3.5 | 1.4 | 0.3 | Iris-setosa |
| 20 | 19 | 5.7 | 3.8 | 1.7 | 0.3 | Iris-setosa |
| 21 | 20 | 5.1 | 3.8 | 1.5 | 0.3 | Iris-setosa |
| 22 | 21 | 5.4 | 3.4 | 1.7 | 0.2 | Iris-setosa |
| 23 | 22 | 5.1 | 3.7 | 1.5 | 0.4 | Iris-setosa |
| 24 | 23 | 4.6 | 3.6 | 1 | 0.2 | Iris-setosa |
| 25 | 24 | 5.1 | 3.3 | 1.7 | 0.5 | Iris-setosa |
| 26 | 25 | 4.8 | 3.4 | 1.9 | 0.2 | Iris-setosa |
| 27 | 26 | 5 | 3 | 1.6 | 0.2 | Iris-setosa |
| 28 | 27 | 5 | 3.4 | 1.6 | 0.4 | Iris-setosa |
| 29 | 28 | 5.2 | 3.5 | 1.5 | 0.2 | Iris-setosa |
| 30 | 29 | 5.2 | 3.4 | 1.4 | 0.2 | Iris-setosa |

iris

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

# Load the Iris dataset
iris = datasets.load_iris()
X = pd.DataFrame(iris.data, columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(iris.target, columns=['Targets'])

# KMeans clustering
kmeans_model = KMeans(n_clusters=3, random_state=42)
kmeans_model.fit(X)

# Standardize the data
scaler = preprocessing.StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

# Gaussian Mixture Model (GMM) clustering
gmm = GaussianMixture(n_components=3, random_state=42)  # Set n_components to match the number of true clusters
gmm.fit(X_scaled_df)

# Visualization
plt.figure(figsize=(18, 6))
colormap = np.array(['red', 'lime', 'black'])
```

```
# Real clusters
plt.subplot(1, 3, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# K-Means clustering
plt.subplot(1, 3, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[kmeans_model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# GMM clustering
plt.subplot(1, 3, 3)
gmm_labels = gmm.predict(X_scaled_df)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_labels], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

plt.tight_layout()
plt.show()

# Observation
print('Observation: The GMM using EM algorithm-based clustering matched the true labels more closely than KMeans.')
```




```
Observation: The GMM using EM algorithm-based clustering matched the true labels more closely than KM
eans.
```

# Practical : 9

**Aim:** Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

**Dataset:**

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Id | SepalLeng | SepalWidt | PetalLeng | PetalWidt | Species |
| 2 | 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 3 | 2 | 4.9 | 3 | 1.4 | 0.2 | Iris-setosa |
| 4 | 3 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 5 | 4 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 6 | 5 | 5 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| 7 | 6 | 5.4 | 3.9 | 1.7 | 0.4 | Iris-setosa |
| 8 | 7 | 4.6 | 3.4 | 1.4 | 0.3 | Iris-setosa |
| 9 | 8 | 5 | 3.4 | 1.5 | 0.2 | Iris-setosa |
| 10 | 9 | 4.4 | 2.9 | 1.4 | 0.2 | Iris-setosa |
| 11 | 10 | 4.9 | 3.1 | 1.5 | 0.1 | Iris-setosa |
| 12 | 11 | 5.4 | 3.7 | 1.5 | 0.2 | Iris-setosa |
| 13 | 12 | 4.8 | 3.4 | 1.6 | 0.2 | Iris-setosa |
| 14 | 13 | 4.8 | 3 | 1.4 | 0.1 | Iris-setosa |
| 15 | 14 | 4.3 | 3 | 1.1 | 0.1 | Iris-setosa |
| 16 | 15 | 5.8 | 4 | 1.2 | 0.2 | Iris-setosa |
| 17 | 16 | 5.7 | 4.4 | 1.5 | 0.4 | Iris-setosa |
| 18 | 17 | 5.4 | 3.9 | 1.3 | 0.4 | Iris-setosa |
| 19 | 18 | 5.1 | 3.5 | 1.4 | 0.3 | Iris-setosa |
| 20 | 19 | 5.7 | 3.8 | 1.7 | 0.3 | Iris-setosa |
| 21 | 20 | 5.1 | 3.8 | 1.5 | 0.3 | Iris-setosa |
| 22 | 21 | 5.4 | 3.4 | 1.7 | 0.2 | Iris-setosa |
| 23 | 22 | 5.1 | 3.7 | 1.5 | 0.4 | Iris-setosa |
| 24 | 23 | 4.6 | 3.6 | 1 | 0.2 | Iris-setosa |
| 25 | 24 | 5.1 | 3.3 | 1.7 | 0.5 | Iris-setosa |
| 26 | 25 | 4.8 | 3.4 | 1.9 | 0.2 | Iris-setosa |
| 27 | 26 | 5 | 3 | 1.6 | 0.2 | Iris-setosa |
| 28 | 27 | 5 | 3.4 | 1.6 | 0.4 | Iris-setosa |
| 29 | 28 | 5.2 | 3.5 | 1.5 | 0.2 | Iris-setosa |
| 30 | 29 | 5.2 | 3.4 | 1.4 | 0.2 | Iris-setosa |

iris

```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Display the features and target classes
print('Features (sepal-length, sepal-width, petal-length, petal-width):')
print(X)
print('Class: 0 - Iris-Setosa, 1 - Iris-Versicolor, 2 - Iris-Virginica')
print(y)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and fit the K-Nearest Neighbors classifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = classifier.predict(X_test)

# Display the confusion matrix and classification report
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))

print('Accuracy Metrics:')
print(classification_report(y_test, y_pred))
```

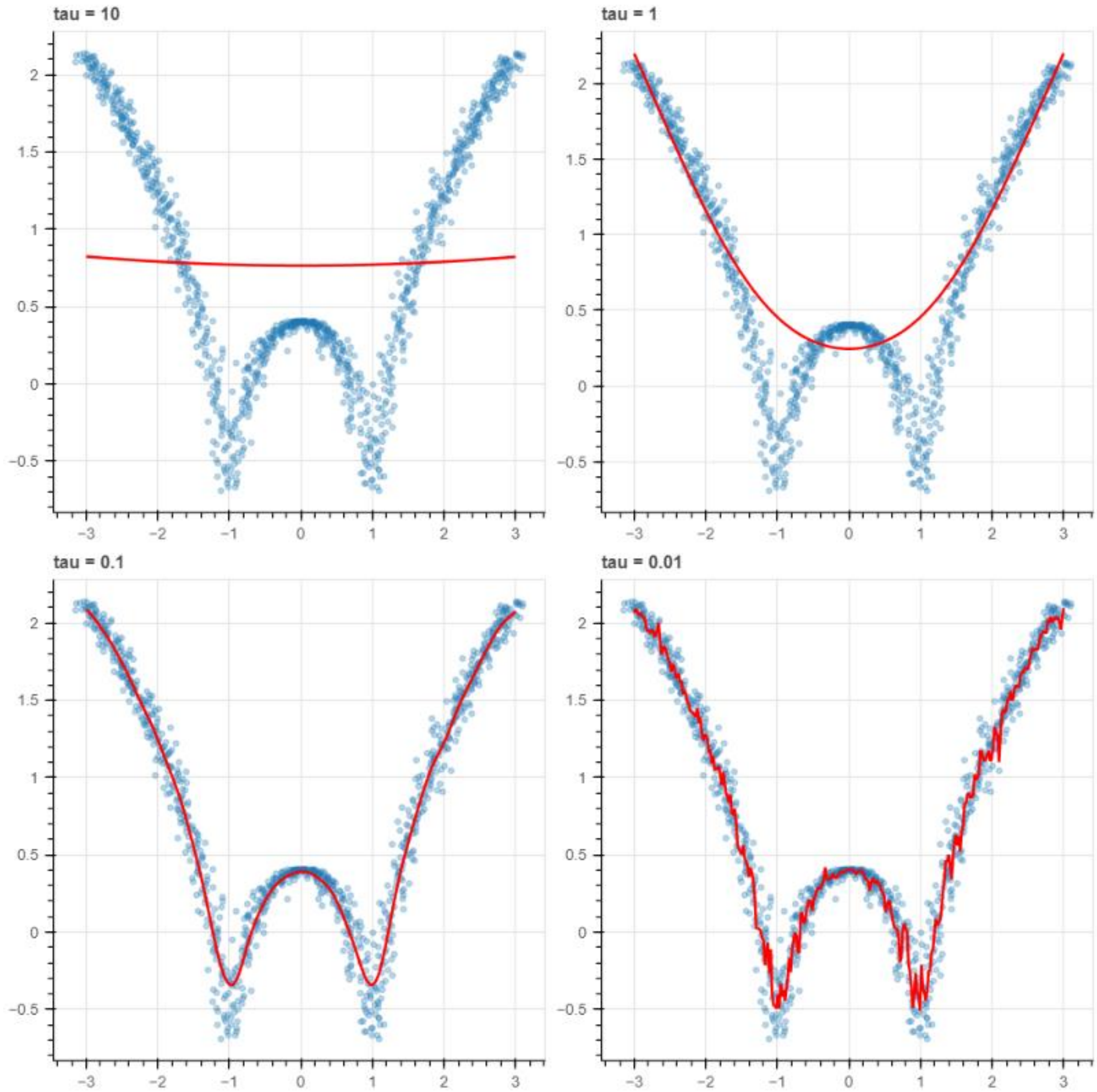
```
Features (sepal-length, sepal-width, petal-length, petal-width):
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
```

```
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
Class: 0 - Iris-Setosa, 1 - Iris-Versicolor, 2 - Iris-Virginica
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

# Practical : 10

**Aim:** Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

[1]:
```python
import numpy as np
from bokeh.plotting import figure, show
from bokeh.layouts import gridplot

# Local Weighted Regression (LWR) function
def local_regression(x0, X, Y, tau):
    # Add a bias term to x0
    x0 = np.r_[1, x0]

    # Add bias term to X
    X = np.c_[np.ones(len(X)), X]

    # Calculate the kernelized weight for each point
    xw = X.T * radial_kernel(x0, X, tau)  # XTranspose * Weight (W)

    # Compute beta (the weight vector)
    beta = np.linalg.pinv(xw @ X) @ xw @ Y  # Using pseudo-inverse for stability

    # Predict the value
    return x0 @ beta  # Predict the value using dot product
# Radial kernel function (Gaussian)
def radial_kernel(x0, X, tau):
    # Calculate radial kernel (Gaussian weights)
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))

# Generate dataset
n = 1000
X = np.linspace(-3, 3, num=n)
Y = np.log(np.abs(X ** 2 - 1) + 0.5)

# Add jitter to X to simulate noise
X += np.random.normal(scale=0.1, size=n)

# Define the domain for prediction
domain = np.linspace(-3, 3, num=300)

# Function to plot the locally weighted regression
def plot_lwr(tau):
    # Make predictions using the local regression model
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]

    # Create plot
    plot = figure(width=400, height=400)
    plot.title.text = 'tau = %g' % tau  # Display tau value in the title

    # Scatter plot of the original data
    plot.scatter(X, Y, alpha=0.3)

    # Line plot of the predicted regression
    plot.line(domain, prediction, line_width=2, color='red')

    return plot

# Display the plots with different tau values in a grid layout
show(gridplot([[plot_lwr(10.0), plot_lwr(1.0)], [plot_lwr(0.1), plot_lwr(0.01)]]))
```


tau = 10
tau = 1
tau = 0.1
tau = 0.01

## → tips.csv

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | total_bill | tip | sex | smoker | day | time | size |
| 2 | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| 3 | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| 4 | 21.01 | 3.5 | Male | No | Sun | Dinner | 3 |
| 5 | 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| 6 | 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |
| 7 | 25.29 | 4.71 | Male | No | Sun | Dinner | 4 |
| 8 | 8.77 | 2 | Male | No | Sun | Dinner | 2 |
| 9 | 26.88 | 3.12 | Male | No | Sun | Dinner | 4 |
| 10 | 15.04 | 1.96 | Male | No | Sun | Dinner | 2 |
| 11 | 14.78 | 3.23 | Male | No | Sun | Dinner | 2 |
| 12 | 10.27 | 1.71 | Male | No | Sun | Dinner | 2 |
| 13 | 35.26 | 5 | Female | No | Sun | Dinner | 4 |
| 14 | 15.42 | 1.57 | Male | No | Sun | Dinner | 2 |
| 15 | 18.43 | 3 | Male | No | Sun | Dinner | 4 |
| 16 | 14.83 | 3.02 | Female | No | Sun | Dinner | 2 |
| 17 | 21.58 | 3.92 | Male | No | Sun | Dinner | 2 |
| 18 | 10.33 | 1.67 | Female | No | Sun | Dinner | 3 |
| 19 | 16.29 | 3.71 | Male | No | Sun | Dinner | 3 |

## → Output:

```
The Data Set (10 Samples) X:
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y:
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
 2.11015444 2.10584249 2.10152068]
Normalized (10 Samples) X:
 [-2.94973156 -2.85391642 -2.91528377 -2.72415615 -3.02368489 -2.86973494
 -2.93495497 -3.10473025 -2.84580848]
Xo Domain Space (10 Samples):
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```