B.M.S. COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

Lab Record

Machine Learning

Submitted in partial fulfillment for the 6th Semester Laboratory

Bachelor of Technology in
Computer Science and Engineering

Submitted by:

Rithesh M Rao

1BM20CS126

Department of Computer Science and Engineering B.M.S.
College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
Mar-July-2023

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Machine Learning (20CS6PCMAL) laboratory has been carried out by RITHESH M RAO (1BM20CS126) during the 6th Semester Mar-July-2023.

Signature of the Faculty Incharge:
Prof. Saritha N
Assistant Professor
Department of Computer Science and Engineering

B.M.S. College of Engineering, Bangalore

Contents

Lab Program	Unit #	Program Details
1	1	Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.
2	1	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
3	1	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.
4	3	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
5	3	Write a program to construct a Bayesian network considering training data. Use this model to make predictions.
6	3	Apply k-Means algorithm to cluster a set of data stored in a .CSV file.
7	3	Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.
8	4	Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.
9	4	Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.
10	4	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Program 1: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

In [ ]:
```
# Interactive

len_x = int(input('enter no. of samples: '))
x = []
attrib = input('enter attributes: ').split()
n_attrib = len(attrib)
for i in range(len_x):
    X = input(f'enter sample {i}: ').split()
    x.append(X)
y = []
for i in range(len_x):
    Y = input(f'output for sample {i} (true/false): ')
    Y = Y.lower()
    y.append(True if Y == 'true' else False)
```

In [4]:
```
# Read from csv

import pandas as pd
import numpy as np

data = pd.read_csv("data.csv")
print(data)
x = np.array(data)[:,:-1]
y = np.array(data)[:,-1]
len_x = len(x)
print(x)
print(y)
```

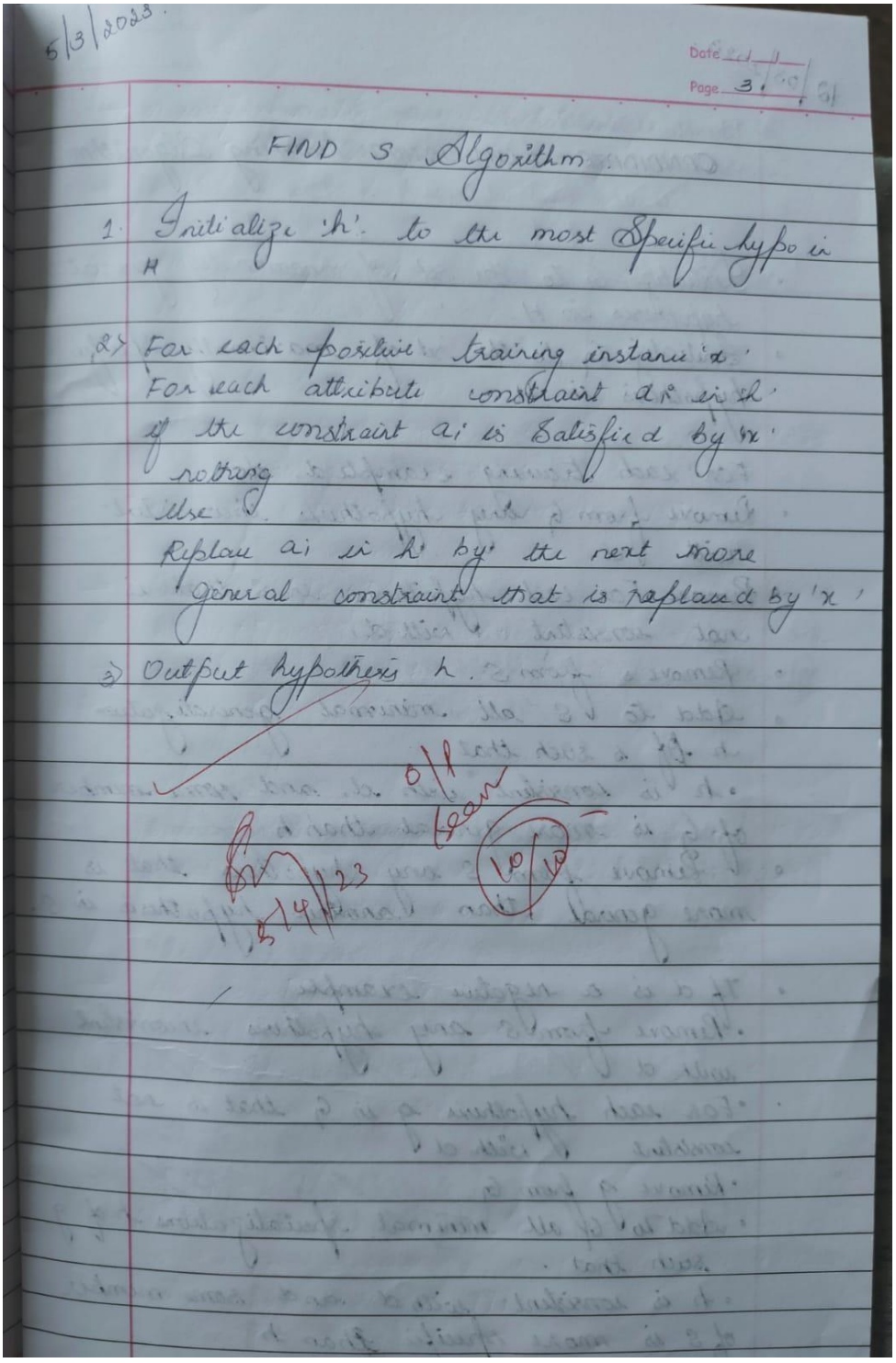
```
   temp       time is_holiday  result
0   hot  afternoon        yes    True
1   hot    morning         no   False
2  cold  afternoon        yes    True
[['hot' 'afternoon' 'yes']
 ['hot' 'morning' 'no']
 ['cold' 'afternoon' 'yes']]
[True False True]
```

In [7]:
```
def findSAlgo(x, y, len_x):
    h = x[0] # initial generalization

    for i in range(1, len_x):
        if not y[i]:
            continue

        for j, attrib in enumerate(x[i]):
            if h[j] != attrib:
                h[j] = '?'

    return h
```

In [8]:
```
findSAlgo(x, y, len_x)
```

Out[8]:
```
array(['?', 'afternoon', 'yes'], dtype=object)
```

ALGORITHM:



6/3/2023.

Date / /
Page 3.

FIND S Algorithm

1. Initialize 'h' to the most Specific hypo in H

2) For each positive training instance 'x'
For each attribute constraint a_i in 'h'
if the constraint a_i is Satisfied by 'x'
nothing
else
Replace a_i in 'h' by the next more general constraint that is satisfied by 'x'

3) Output hypothesis h.

5/4/23

0/1 learn

10/10

Program 2: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

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

In [2]:
```
data = pd.read_csv('../input/enjoysport/enjoysport.csv')
X = data.to_numpy()[:, :-1]
y = data.to_numpy()[:, -1]
```

In [3]:
```
def candidateElimination(X, y):
    n_attrib = len(X[0])
    specific_h = ['0' for _ in range(n_attrib)]
    general_h = [['?' for _ in range(n_attrib)] for _ in range(n_attrib)]

    print('================ Iteration 0 ================')
    print(f'Specific Boundary: {specific_h}')
    print(f'General Boundary: {general_h}')
    print()

    specific_h = X[0].copy()

    for i, x in enumerate(X):
        print(f'================ Iteration {i + 1} ================')
        print(f'Instance {i + 1}: {x} \t Target: {y[i]}')

        if y[i] == 'yes':
            for j in range(n_attrib):
                if x[j] != specific_h[j]:
                    specific_h[j] = '?'
                    general_h[j][j] = '?'
        else:
            for j in range(n_attrib):
                if x[j] != specific_h[j]:
                    general_h[j][j] = specific_h[j]
                else:
                    general_h[j][j] = '?'

        print(f'Specific Boundary: {specific_h}')
        print(f'General Boundary: {general_h}')
        print()

    general_h = [h for h in general_h if h != ['?' for _ in range(n_attrib)]]

    print('================ Result ================')
    print(f'Specific Boundary: {specific_h}')
    print(f'General Boundary: {general_h}')
    print()

    return list(specific_h), list(general_h)
```

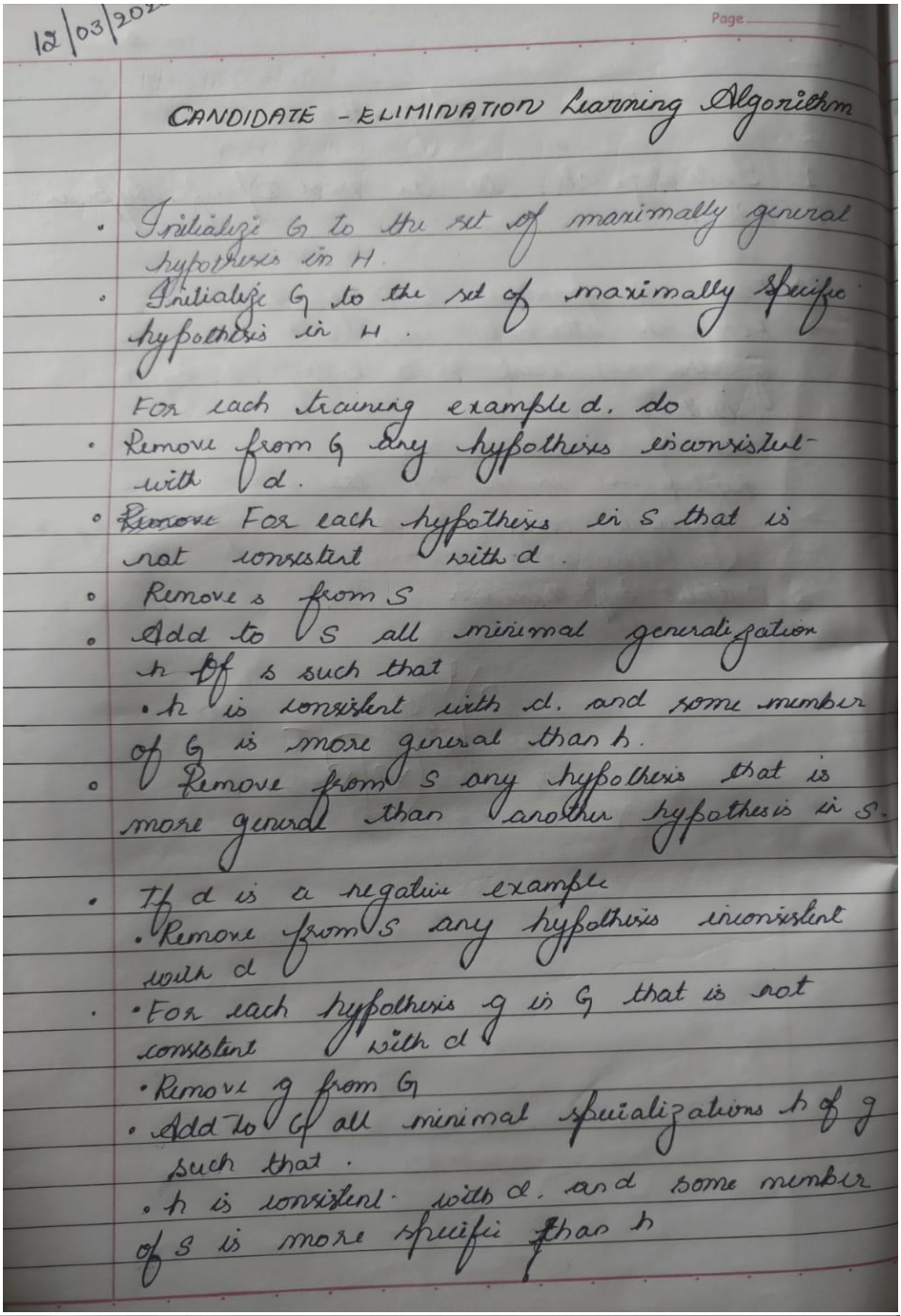
A

In [4]:
```
candidateElimination(X, y)
```

```
================ Iteration 0 ================
Specific Boundary: ['0', '0', '0', '0', '0', '0']
General Boundary: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

================ Iteration 1 ================
Instance 1: ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']     Target: yes
Specific Boundary: ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
General Boundary: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

================ Iteration 2 ================
Instance 2: ['sunny' 'warm' 'high' 'strong' 'warm' 'same']       Target: yes
Specific Boundary: ['sunny' 'warm' '?' 'strong' 'warm' 'same']
General Boundary: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

================ Iteration 3 ================
Instance 3: ['rainy' 'cold' 'high' 'strong' 'warm' 'change']     Target: no
Specific Boundary: ['sunny' 'warm' '?' 'strong' 'warm' 'same']
General Boundary: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

================ Iteration 4 ================
Instance 4: ['sunny' 'warm' 'high' 'strong' 'cool' 'change']     Target: yes
Specific Boundary: ['sunny' 'warm' '?' 'strong' '?' '?']
General Boundary: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

================ Result ================
Specific Boundary: ['sunny' 'warm' '?' 'strong' '?' '?']
General Boundary: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Out[4]:
```
(['sunny', 'warm', '?', 'strong', '?', '?'],
 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']])
```

ALGORITHM:

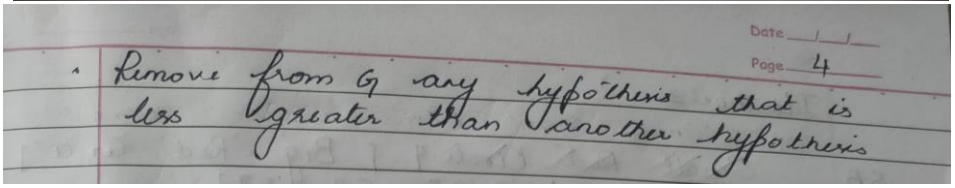


CANDIDATE - ELIMINATION Learning Algorithm

- Initialize G to the set of maximally general hypotheses in H.
- Initialize G to the set of maximally specific hypothesis in H.

For each training example d, do

- Remove from G any hypothesis inconsistent with d.
- ~~Remove~~ For each hypothesis in S that is not consistent with d.
  - Remove s from S
  - Add to S all minimal generalization h of s such that
    - h is consistent with d, and some member of G is more general than h.
  - Remove from S any hypothesis that is more general than another hypothesis in S.
- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
- For each hypothesis g in G that is not consistent with d
  - Remove g from G
  - Add to G all minimal specializations h of g such that.
    - h is consistent with d, and some member of S is more specific than h



Page 4

- Remove from G any hypothesis that is less greater than another hypothesis

Program 3: 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 [62]:
```
import numpy as np
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log
```

In [63]:
```
df = pd.read_csv('../input/playtennis/playtennis.csv')
df
```

Out[63]:

	outlook	temperature	humidity	wind	play
0	sunny	hot	high	weak	no
1	sunny	hot	high	strong	no
2	overcast	hot	high	weak	yes
3	rain	mild	high	weak	yes
4	rain	cool	normal	weak	yes
5	rain	cool	normal	strong	no
6	overcast	cool	normal	strong	yes
7	sunny	mild	high	weak	no
8	sunny	cool	normal	weak	yes
9	rain	mild	normal	weak	yes
10	sunny	mild	normal	strong	yes
11	overcast	mild	high	strong	yes
12	overcast	hot	normal	weak	yes
13	rain	mild	high	strong	no

In [64]:
```
def find_entropy(df):
    Class = df.keys()[-1]
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[Class])
        entropy += -fraction*np.log2(fraction)
    return entropy

def find_entropy_attribute(df,attribute):
    Class = df.keys()[-1]
    target_variables = df[Class].unique()
    variables = df[attribute].unique()
    entropy2 = 0
    for variable in variables:
        entropy = 0
        for target_variable in target_variables:
            num = len(df[attribute][df[attribute]==variable][df[Class] ==target_variable])
            den = len(df[attribute][df[attribute]==variable])
```

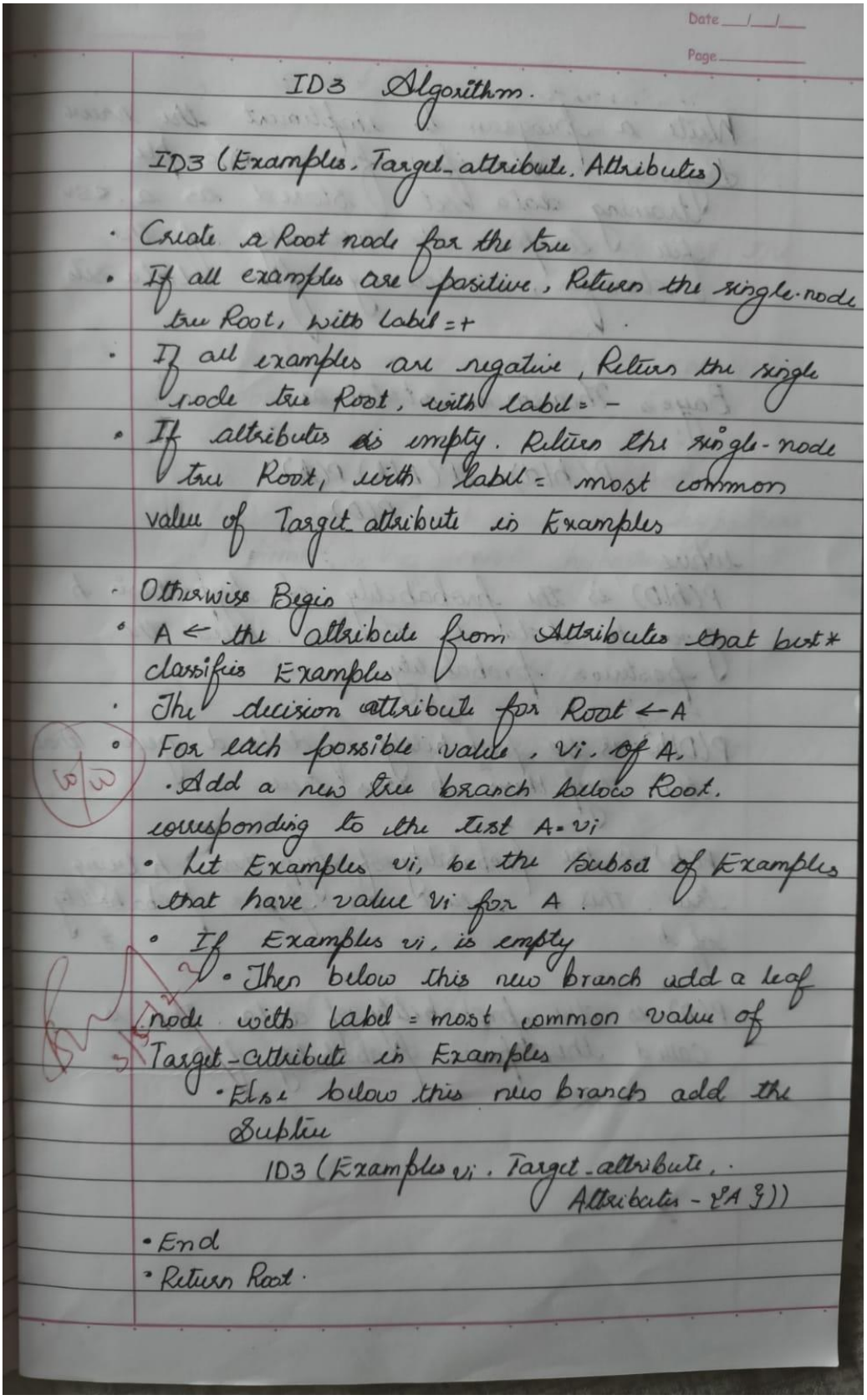
```
            fraction = num/(den+eps)
            entropy += -fraction*log(fraction+eps)
        fraction2 = den/len(df)
        entropy2 += -fraction2*entropy
    return abs(entropy2)


def find_winner(df):
    Entropy_att = []
    IG = []
    for key in df.keys()[:-1]:
        IG.append(find_entropy(df)-find_entropy_attribute(df,key))
    return df.keys()[:-1][np.argmax(IG)]

def get_subtable(df, node,value):
    return df[df[node] == value].reset_index(drop=True)

def buildTree(df,tree=None):
    Class = df.keys()[-1]
    node = find_winner(df)
    attValue = np.unique(df[node])
    if tree is None:
        tree={}
        tree[node] = {}

    for value in attValue:

        subtable = get_subtable(df,node,value)
        clValue,counts = np.unique(subtable['play'],return_counts=True)

        if len(counts)==1:
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable)

    return tree
```

In [65]:
```
t = buildTree(df)
t
```

Out[65]:
```
{'outlook': {'overcast': 'yes',
  'rain': {'wind': {'strong': 'no', 'weak': 'yes'}},
  'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}}}
```

In [ ]:

ALGORITHM:



Date __/__/__

Page _____

ID3 Algorithm.

ID3 (Examples, Target_attribute, Attributes)

- Create a Root node for the tree
- If all examples are positive, Return the single-node tree Root, with label = +
- If all examples are negative, Return the single node tree Root, with label = -
- If attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples

- Otherwise Begin
- A ← the attribute from Attributes that best* classifies Examples
- The decision attribute for Root ← A
- For each possible value, vi, of A,
  - Add a new tree branch below Root, corresponding to the test A = vi
  - Let Examples vi, be the subset of Examples that have value vi for A
  - If Examples vi, is empty
    - Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples
    - Else below this new branch add the Subtree

      ID3 (Examples vi, Target_attribute, Attributes - {A})

- End
- Return Root.

Program 4: 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.

In [1]:
```
import csv
import random
import math
```

In [2]:
```
def load_csv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset

def split_dataset(dataset, splitratio):
    trainsize = int(len(dataset) * splitratio);
    trainset = []
    copy = list(dataset);
    while len(trainset) < trainsize:
        index = random.randrange(len(copy));
        trainset.append(copy.pop(index))
    return [trainset, copy]

def separate_by_class(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated

def mean(numbers):
    return sum(numbers)/float(len(numbers))

def std_dev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)

def summarize(dataset):
    summaries = [(mean(attribute), std_dev(attribute)) for attribute in zip(*dataset)];
    del summaries[-1]
    return summaries

def summarize_by_class(dataset):
    separated = separate_by_class(dataset);
    summaries = {}
    for classvalue, instances in separated.items():
        summaries[classvalue] = summarize(instances)
    return summaries

def calculate_probability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculate_class_probabilities(summaries, inputvector):
    probabilities = {}
```

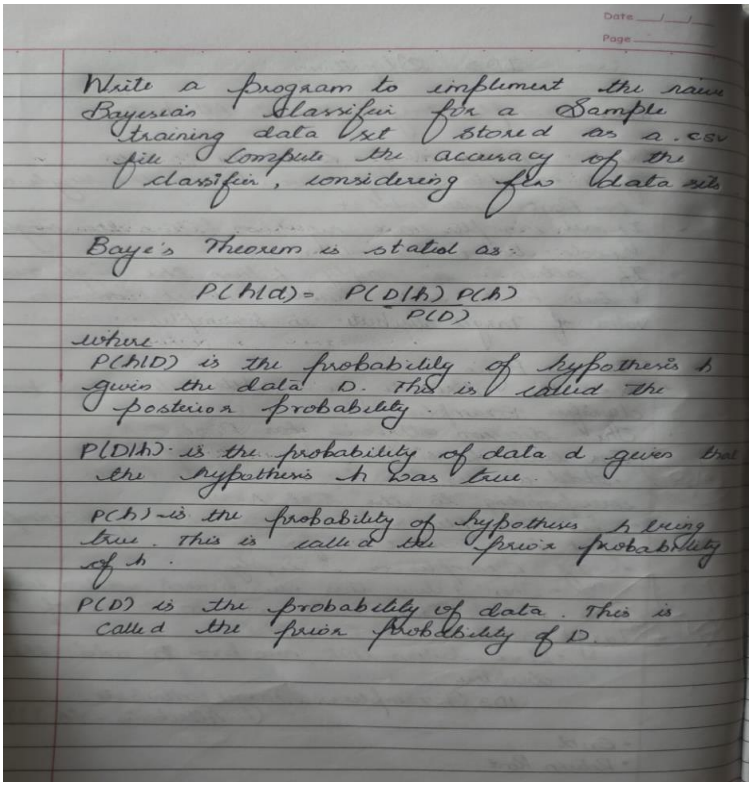
```
    for classvalue, classsummaries in summaries.items():
        probabilities[classvalue] = 1
    for i in range(len(classsummaries)):
        mean, stdev = classsummaries[i]
        x = inputvector[i]
        probabilities[classvalue] *= calculate_probability(x, mean, stdev)
    return probabilities

def predict(summaries, inputvector):
    probabilities = calculate_class_probabilities(summaries, inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classvalue
    return bestLabel

def get_predictions(summaries, testset):
    predictions = []
    for i in range(len(testset)):
        result = predict(summaries, testset[i])
        predictions.append(result)
    return predictions

def get_accuracy(testset, predictions):
    correct = 0
    for i in range(len(testset)):
        if testset[i][-1] == predictions[i]:
            correct += 1
    return (correct/float(len(testset))) * 100.0
```

In [3]:
```
splitratio = 0.67
dataset = load_csv('../input/pimaindiansdiabetescsv/pima-indians-diabetes.csv');

trainingset, testset = split_dataset(dataset, splitratio)

print(f'Split {len(dataset)} rows into train={len(trainingset)} and test={len(testset)} rows')

summaries = summarize_by_class(trainingset);
predictions = get_predictions(summaries, testset)
accuracy = get_accuracy(testset, predictions)

print(f'Accuracy of the classifier is :{accuracy}%')
```

```
Split 768 rows into train=514 and test=254 rows
Accuracy of the classifier is :64.96062992125984%
```

In [ ]:

ALGORITHM:



Write a program to implement the naive Bayesian classifier for a sample training data set stored as a .csv file. Compute the accuracy of the classifier, considering few test data sets.

Baye's Theorem is stated as:

$$P(h|d) = \frac{P(D|h)\,P(h)}{P(D)}$$

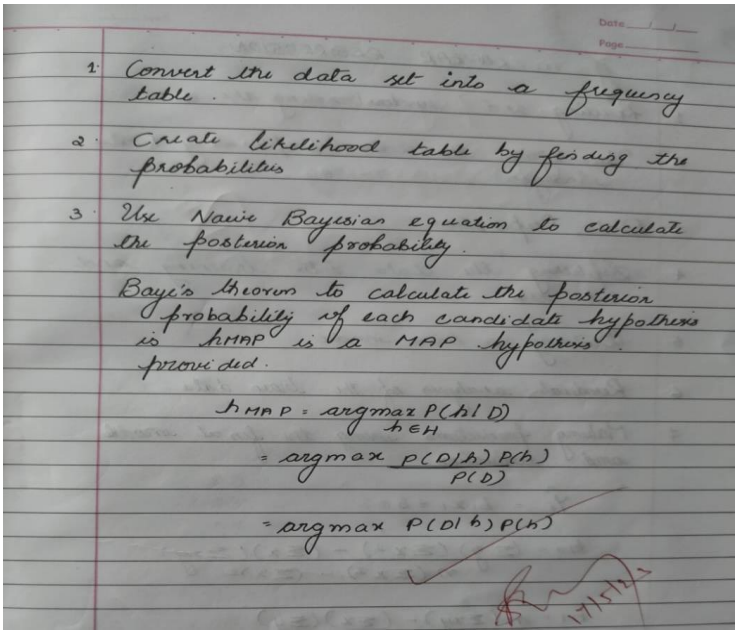
where

$P(h|D)$ is the probability of hypothesis h given the data D. This is called the posterior probability.

$P(D|h)$ is the probability of data d given that the hypothesis h was true.

$P(h)$ is the probability of hypothesis h being true. This is called the prior probability of h.

$P(D)$ is the probability of data. This is called the prior probability of D.



1. Convert the data set into a frequency table.
2. Create likelihood table by finding the probabilities
3. Use Naive Bayesian equation to calculate the posterior probability.

Baye's theorem to calculate the posterior probability of each candidate hypothesis is h_{MAP} is a MAP hypothesis provided.

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}}\, P(h|D)$$

$$= \underset{h}{\operatorname{argmax}} \frac{P(D|h)\,P(h)}{P(D)}$$

$$= \underset{h}{\operatorname{argmax}}\, P(D|h)\,P(h)$$

17/5/22

Program 5: Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

In [1]:
```
import numpy as np
import pandas as pd
import csv
!pip install pgmpy
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
```

```
Collecting pgmpy
  Downloading pgmpy-0.1.14-py3-none-any.whl (331 kB)
     |████████████████████████████████| 331 kB 3.0 MB/s
Requirement already satisfied: pandas in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.2.3)
Requirement already satisfied: scikit-learn in /opt/conda/lib/python3.7/site-packages (from pgmpy) (0.24.1)
Requirement already satisfied: scipy in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.5.4)
Requirement already satisfied: torch in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.7.0)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-packages (from pgmpy) (4.59.0)
Requirement already satisfied: joblib in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.0.1)
Requirement already satisfied: pyparsing in /opt/conda/lib/python3.7/site-packages (from pgmpy) (2.4.7)
Requirement already satisfied: statsmodels in /opt/conda/lib/python3.7/site-packages (from pgmpy) (0.12.2)
Requirement already satisfied: networkx in /opt/conda/lib/python3.7/site-packages (from pgmpy) (2.5)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.19.5)
Requirement already satisfied: decorator>=4.3.0 in /opt/conda/lib/python3.7/site-packages (from networkx->pgmpy) (4.4.
2)
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/conda/lib/python3.7/site-packages (from pandas->pgmpy)
(2.8.1)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.7/site-packages (from pandas->pgmpy) (2021.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-packages (from python-dateutil>=2.7.3->pandas
->pgmpy) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from scikit-learn->pgmp
y) (2.1.0)
Requirement already satisfied: patsy>=0.5 in /opt/conda/lib/python3.7/site-packages (from statsmodels->pgmpy) (0.5.1)
Requirement already satisfied: future in /opt/conda/lib/python3.7/site-packages (from torch->pgmpy) (0.18.2)
Requirement already satisfied: typing_extensions in /opt/conda/lib/python3.7/site-packages (from torch->pgmpy) (3.7.4.
3)
Requirement already satisfied: dataclasses in /opt/conda/lib/python3.7/site-packages (from torch->pgmpy) (0.6)
Installing collected packages: pgmpy
Successfully installed pgmpy-0.1.14
```

In [2]:
```
heartDisease = pd.read_csv('../input/heartdisease/heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
```

```
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

 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
```

In [3]:
```
model = BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),
('heartdisease','restecg'),('heartdisease','chol')])
```

In [4]:
```
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\nInferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
```

```
Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:
```

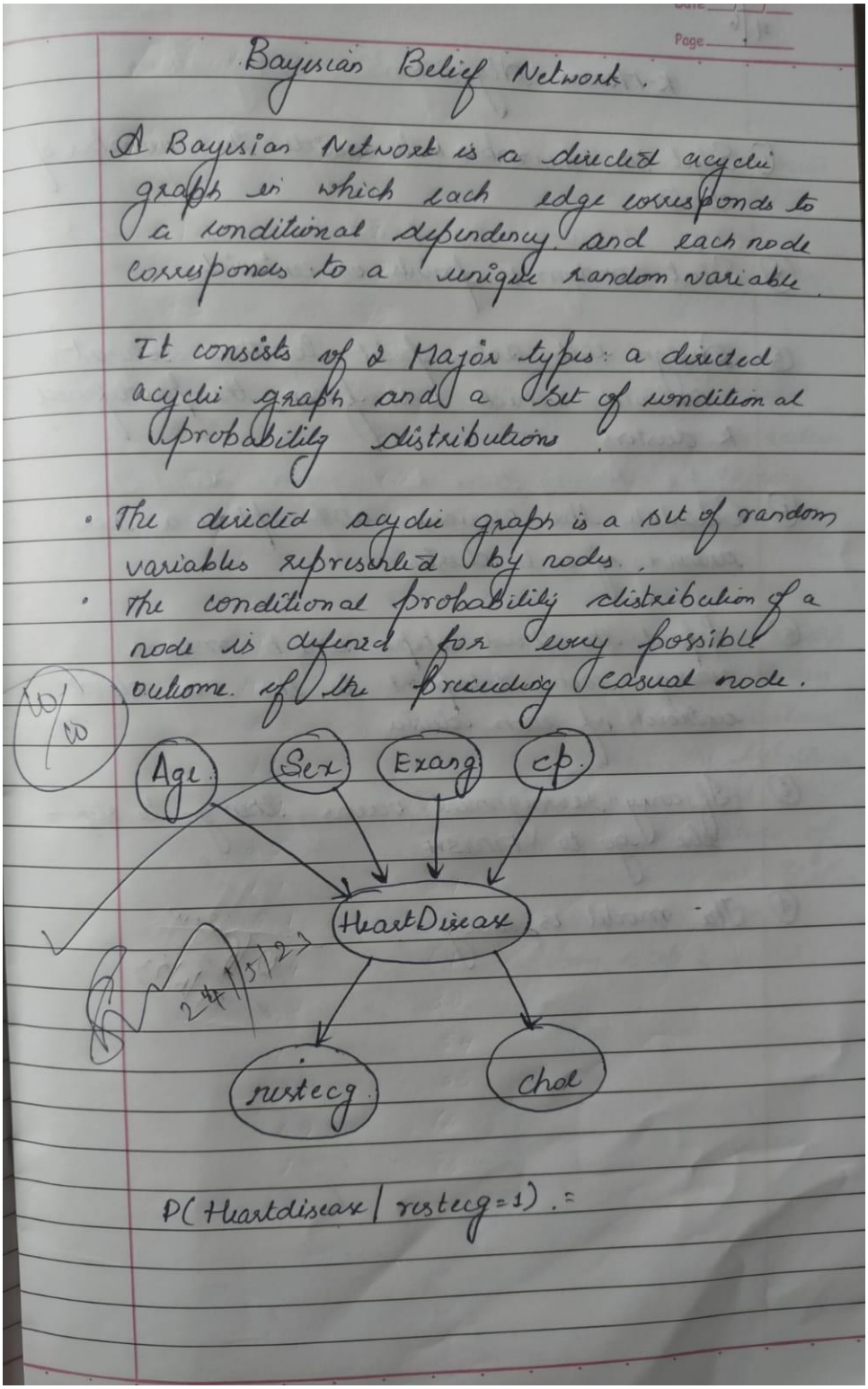
In [5]:
```
print('\n1.Probability of HeartDisease given evidence = restecg :')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)

print('\n2.Probability of HeartDisease given evidence = cp :')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

```
Finding Elimination Order: :   0%|          | 0/5 [00:00<?, ?it/s]
  0%|          | 0/5 [00:00<?, ?it/s]
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 480.98it/s]

Eliminating: age:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: sex:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: chol:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: exang: 100%|██████████| 5/5 [00:00<00:00, 92.85it/s]
```

```
1.Probability of HeartDisease given evidence = restecg :
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+

2.Probability of HeartDisease given evidence = cp :
Finding Elimination Order: :   0%|          | 0/5 [00:00<?, ?it/s]
  0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: age:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: sex:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: chol:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: exang:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: restecg: 100%|██████████| 5/5 [00:00<00:00, 227.41it/s]
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

In [ ]:

ALGORITHM:



Bayesian Belief Network.

A Bayesian Network is a directed acyclic graph in which each edge corresponds to a conditional dependency and each node corresponds to a unique random variable.

It consists of 2 major types: a directed acyclic graph and a set of conditional probability distributions.

- The directed acyclic graph is a set of random variables represented by nodes.
- The conditional probability distribution of a node is defined for every possible outcome of the preceding/casual node.

P(Heartdisease | restecg=1) =

Program 6: Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

In [1]:
```
from sklearn import datasets
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, confusion_matrix
```

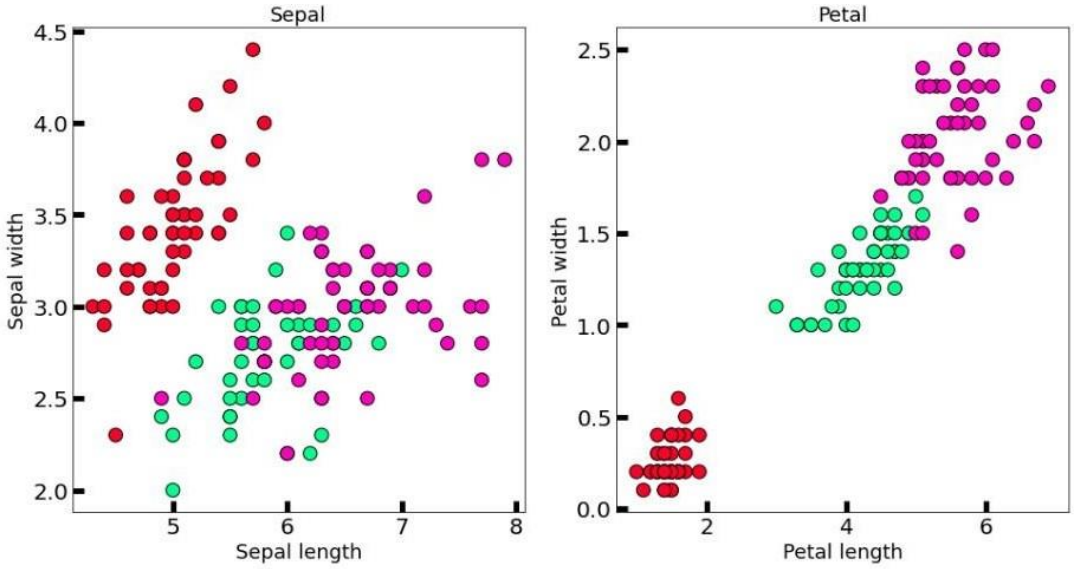
Dataset

In [2]:
```
iris = datasets.load_iris()
iris
```

Out[2]:
```
{'data': array([[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],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
        [5.2, 3.5, 1.5, 0.2],
        [5.2, 3.4, 1.4, 0.2],
        [4.7, 3.2, 1.6, 0.2],
        [4.8, 3.1, 1.6, 0.2],
        [5.4, 3.4, 1.5, 0.4],
        [5.2, 4.1, 1.5, 0.1],
        [5.5, 4.2, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.2],
        [5. , 3.2, 1.2, 0.2],
        [5.5, 3.5, 1.3, 0.2],
        [4.9, 3.6, 1.4, 0.1],
        [4.4, 3. , 1.3, 0.2],
        [5.1, 3.4, 1.5, 0.2],
```

```
       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, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4 numeric, predictive attributes and the class\n    :Attribute Information:\n        - sepal length in cm\n        - sepal width in cm\n        - petal length in cm\n        - petal width in cm\n        - class:\n                - Iris-Setosa\n                - Iris-Versicolour\n                - Iris-Virginica\n                \n    :Summary Statistics:\n\n    ============== ==== ==== ======= ===== ====================\n                    Min  Max   Mean    SD   Class Correlation\n    ============== ==== ==== ======= ===== ====================\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ============== ==== ==== ======= ===== ====================\n\n    :Missing Attribute Values: None\n    :Class Distribution: 33.3% for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found in the\npattern recognition literature.  Fisher\'s paper is a classic in the field and\nis referenced frequently to this day.  (See Duda & Hart, for example.)  The\ndata set contains 3 classes of 50 instances each, where each class refers to a\ntype of iris plant.  One class is linearly separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistics" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n     Structure and Classification Rule for Recognition in Partially Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n     on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3 classes in the data.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': '/opt/conda/lib/python3.7/site-packages/sklearn/datasets/data/iris.csv'}
```

In [3]:
```
sepal_X = iris.data[:, :2]
petal_X = iris.data[:, 2:]
y = iris.target
categories = len(iris.target_names)

fig, axes = plt.subplots(1, 2, figsize=(16,8))
axes[0].scatter(sepal_X[:, 0], sepal_X[:, 1], c=y, cmap='gist_rainbow', edgecolor='k', s=150)
axes[1].scatter(petal_X[:, 0], petal_X[:, 1], c=y, cmap='gist_rainbow', edgecolor='k', s=150)
axes[0].set_xlabel('Sepal length', fontsize=18)
axes[0].set_ylabel('Sepal width', fontsize=18)
axes[1].set_xlabel('Petal length', fontsize=18)
axes[1].set_ylabel('Petal width', fontsize=18)
axes[0].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
axes[1].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
axes[0].set_title('Sepal', fontsize=18)
axes[1].set_title('Petal', fontsize=18)
plt.show()
```



Model

In [4]:
```
model_sepal = KMeans(n_clusters=3)
model_sepal.fit(sepal_X)
model_petal = KMeans(n_clusters=3)
model_petal.fit(petal_X)
```

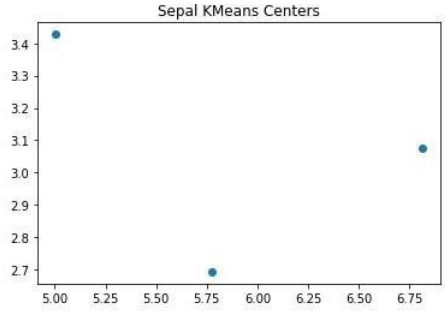
Out[4]:
```
KMeans(n_clusters=3)
```

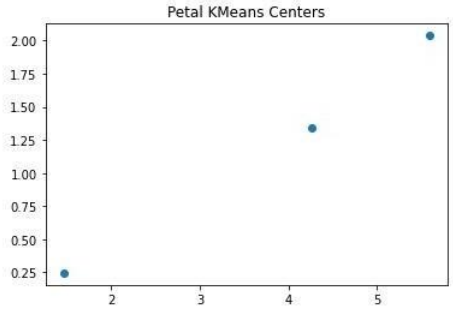
Analysis

In [5]:
```
def plot_centers(sepal_centers, petal_centers):
    plt.scatter([point[0] for point in sepal_centers], [point[1] for point in sepal_centers])
    plt.title('Sepal KMeans Centers')
    plt.show()
```

```
    plt.title( 'Petal KMeans Centers' )
    plt.show()

def plot_actualvpredicted(X, y, predicted, part):
    fig, axes = plt.subplots(1, 2, figsize=(16,8))
    axes[0].scatter(X[:, 0], X[:, 1], c=y, cmap='gist_rainbow', edgecolor='k', s=150)
    axes[1].scatter(X[:, 0], X[:, 1], c=predicted, cmap='jet', edgecolor='k', s=150)
    axes[0].set_xlabel(f'{part} length', fontsize=18)
    axes[0].set_ylabel(f'{part} width', fontsize=18)
    axes[1].set_xlabel(f'{part} length', fontsize=18)
    axes[1].set_ylabel(f'{part} width', fontsize=18)
    axes[0].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
    axes[1].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
    axes[0].set_title('Actual', fontsize=18)
    axes[1].set_title('Predicted', fontsize=18)
    plt.show()

def plot_confusion(accuracy, confusion, part):
    print(f'{part} Accuracy: {accuracy}')

    fig, ax = plt.subplots()
    im = ax.imshow(confusion)

    ax.set_xticks(range(categories))
    ax.set_yticks(range(categories))
    ax.set_xticklabels(iris.target_names)
    ax.set_yticklabels(iris.target_names)

    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    for i in range(categories):
        for j in range(categories):
            text = ax.text(j, i, confusion[i, j],
                           ha="center", va="center", color="w")

    ax.set_title(f"{part} Confusion Matrix (Actual / Predicted)")
    fig.tight_layout()
    plt.show()
```

In [6]:
```
sepal_centers = model_sepal.cluster_centers_
petal_centers = model_petal.cluster_centers_
plot_centers(sepal_centers, petal_centers)
```

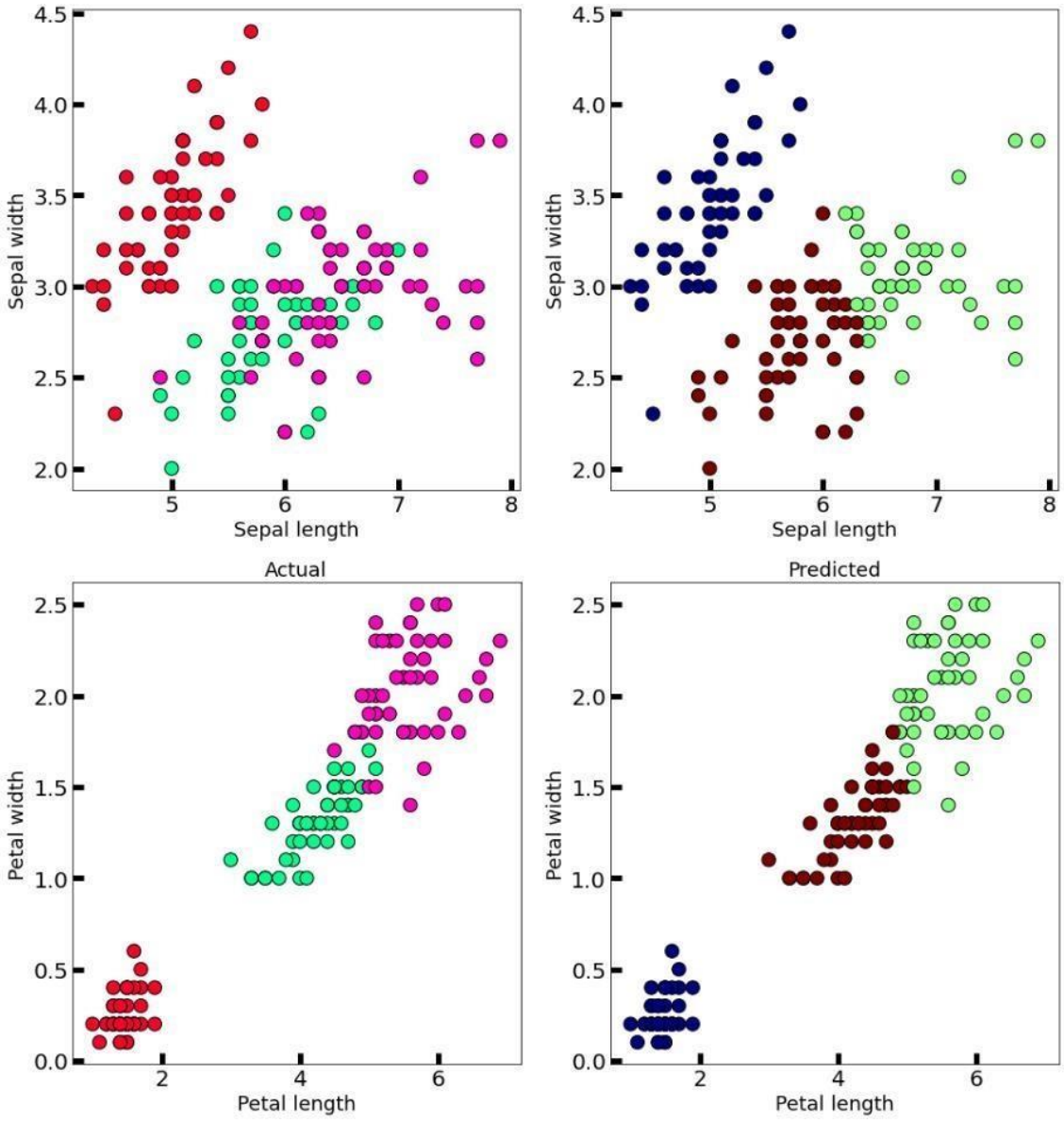




In [7]:
```
sepal_labels = model_sepal.labels_
petal_labels = model_petal.labels_
print(sepal_labels, petal_labels, sep='\n')
```

```
[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 0 1 1 1 2 1 2 1 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2
 1 1 1 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 1 1 1
 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 2 2 1 1 1 2 1 1 1 2 1 1 1 2 1
 1 2]
[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 0 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 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 2 1 1 1 1
 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1
 1 1]
```

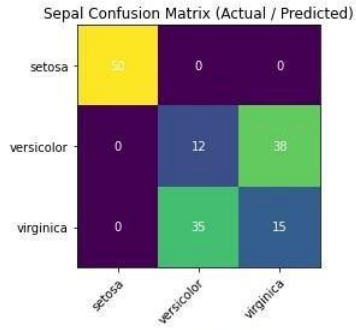
In [8]:
```
plot_actualvpredicted(sepal_X, y, sepal_labels, 'Sepal')
plot_actualvpredicted(petal_X, y, petal_labels, 'Petal')
```


Sepal width
Sepal length
Sepal width
Sepal length
Actual
Petal width
Petal length
Predicted
Petal width
Petal length

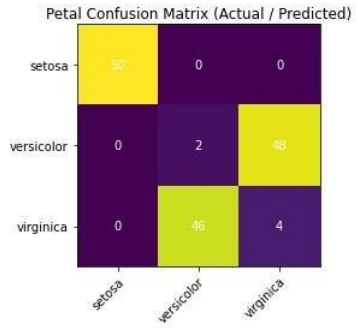
In [9]:
```
sepal_accuracy = accuracy_score(y, sepal_labels)
petal_accuracy = accuracy_score(y, petal_labels)

sepal_confusion = confusion_matrix(y, sepal_labels)
petal_confusion = confusion_matrix(y, petal_labels)
```

In [10]:
```
plot_confusion(sepal_accuracy, sepal_confusion, 'Sepal')
plot_confusion(petal_accuracy, petal_confusion, 'Petal')
```

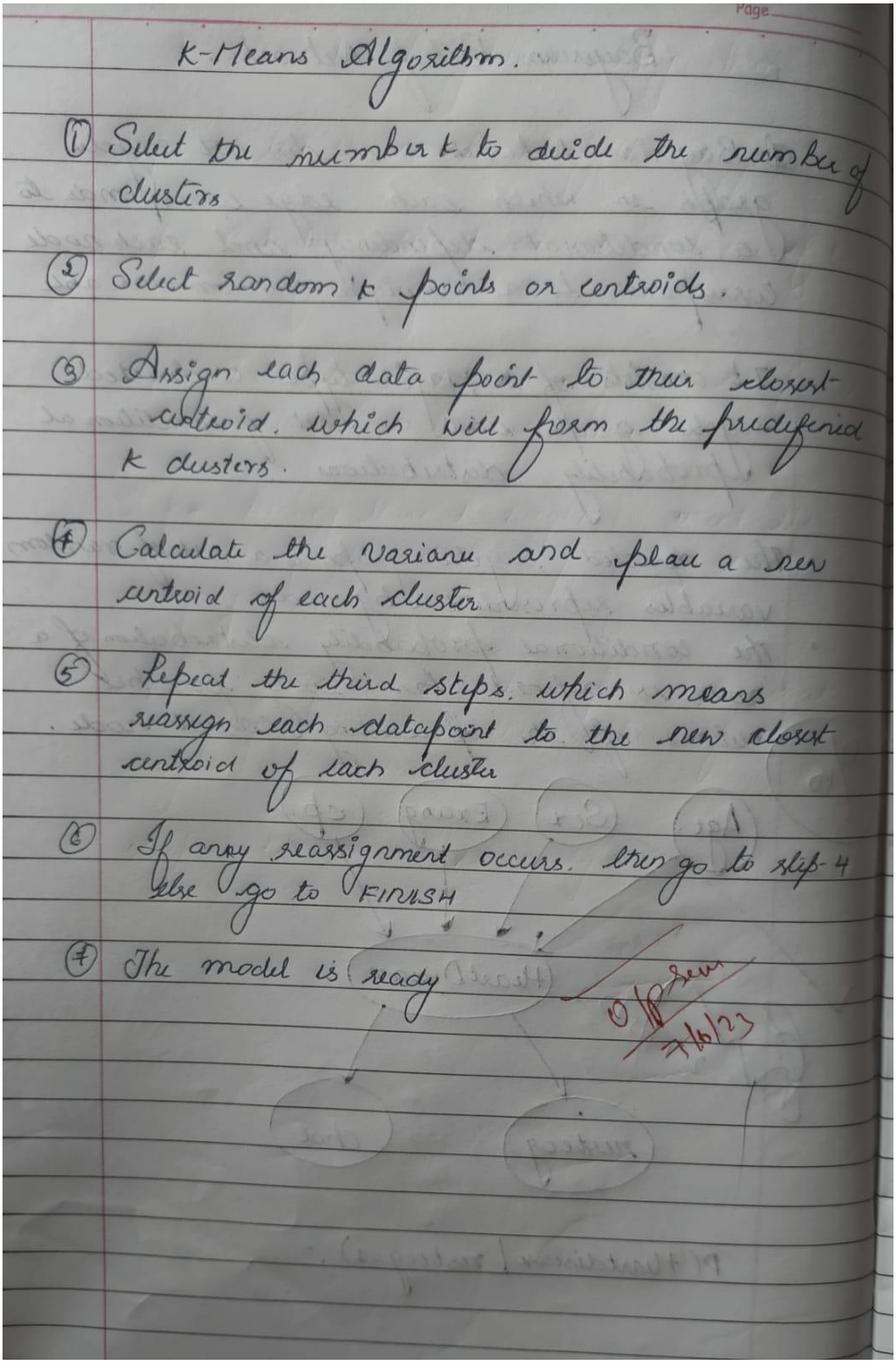
```
Sepal Accuracy: 0.5133333333333333
```




```
Petal Accuracy: 0.37333333333333335
```



ALGORITHM:



K-Means Algorithm.

1. Select the number k to decide the number of clusters
2. Select random k points or centroids.
3. Assign each data point to their closest centroid. which will form the predefined k clusters.
4. Calculate the variance and place a new centroid of each cluster
5. Repeat the third steps. which means reassign each datapoint to the new closest centroid of each cluster
6. If any reassignment occurs. then go to step-4 else go to FINISH
7. The model is ready

O/P seen 7/6/23

Program 7: Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.mixture import GaussianMixture
from sklearn.metrics import accuracy_score, confusion_matrix
```

Data

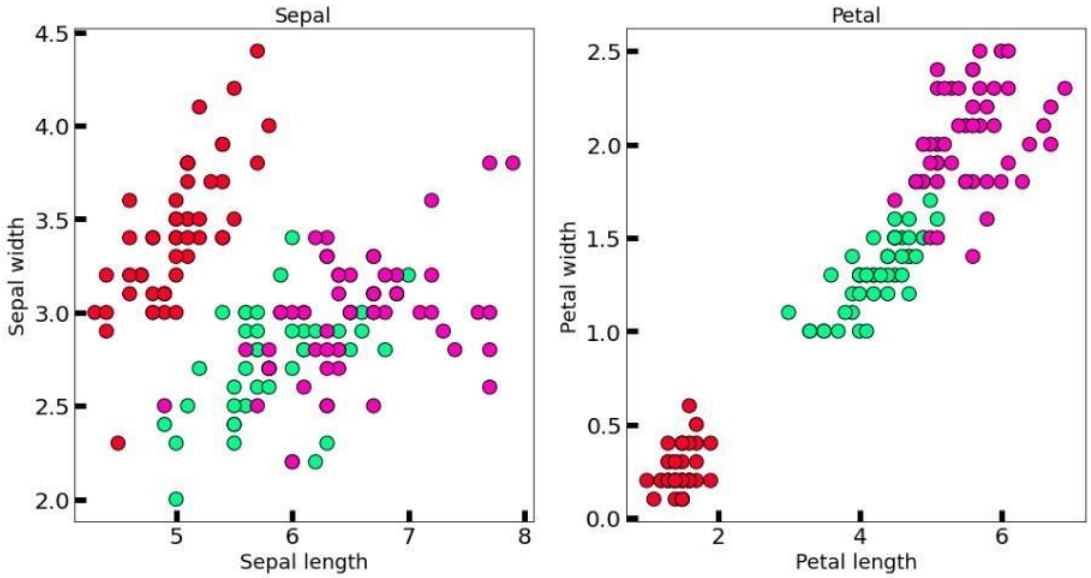
In [2]:
```
data = pd.read_csv('../input/iris/Iris.csv')
data.head()
```

Out[2]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [3]:
```
sepal_X = data[['SepalLengthCm', 'SepalWidthCm']]
petal_X = data[['PetalLengthCm', 'PetalWidthCm']]
y = data['Species'].astype("category").cat.codes
num_cat = y.nunique()
categories = data['Species'].astype("category").cat.categories
```

In [4]:
```
fig, axes = plt.subplots(1, 2, figsize=(16,8))
axes[0].scatter(sepal_X.SepalLengthCm, sepal_X.SepalWidthCm, c=y, cmap='gist_rainbow', edgecolor='k', s=150)
axes[1].scatter(petal_X.PetalLengthCm, petal_X.PetalWidthCm, c=y, cmap='gist_rainbow', edgecolor='k', s=150)
axes[0].set_xlabel('Sepal length', fontsize=18)
axes[0].set_ylabel('Sepal width', fontsize=18)
axes[1].set_xlabel('Petal length', fontsize=18)
axes[1].set_ylabel('Petal width', fontsize=18)
axes[0].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
axes[1].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
axes[0].set_title('Sepal', fontsize=18)
axes[1].set_title('Petal', fontsize=18)
plt.show()
```



Model

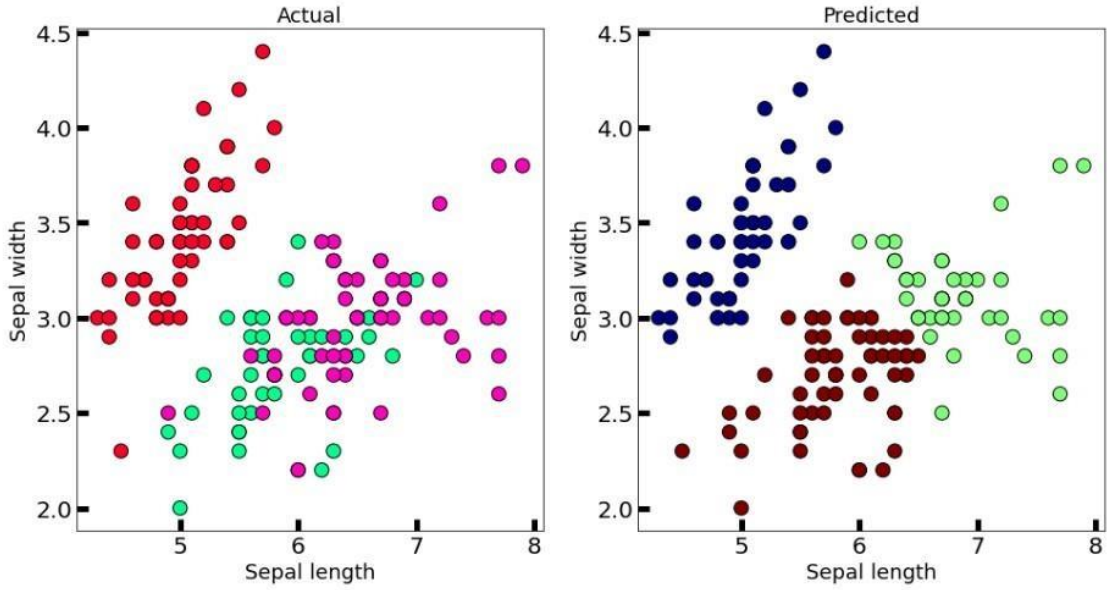
In [5]:
```
model_sepal = GaussianMixture(n_components=3)
sepal_labels = model_sepal.fit_predict(sepal_X)
model_petal = GaussianMixture(n_components=3)
petal_labels = model_petal.fit_predict(petal_X)
```

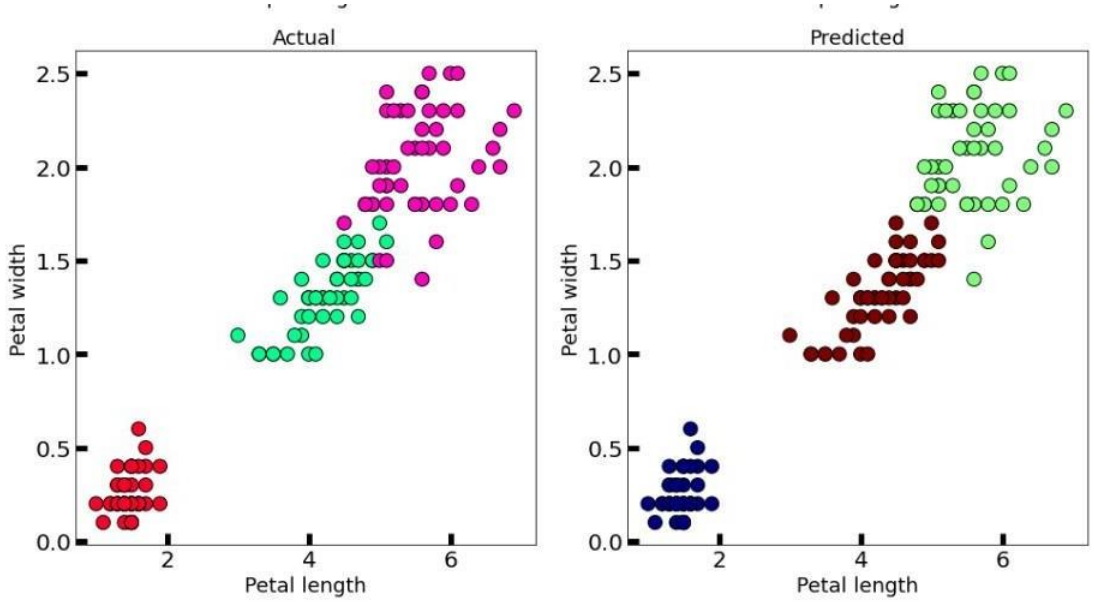
Analysis

In [6]:
```
def plot_actualvpredicted(X, y, predicted, part):
    fig, axes = plt.subplots(1, 2, figsize=(16,8))
    axes[0].scatter(X[f'{part}LengthCm'], X[f'{part}WidthCm'], c=y, cmap='gist_rainbow', edgecolor='k', s=150)
    axes[1].scatter(X[f'{part}LengthCm'], X[f'{part}WidthCm'], c=predicted, cmap='jet', edgecolor='k', s=150)
    axes[0].set_xlabel(f'{part} length', fontsize=18)
    axes[0].set_ylabel(f'{part} width', fontsize=18)
    axes[1].set_xlabel(f'{part} length', fontsize=18)
    axes[1].set_ylabel(f'{part} width', fontsize=18)
    axes[0].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
    axes[1].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
    axes[0].set_title('Actual', fontsize=18)
    axes[1].set_title('Predicted', fontsize=18)
    plt.show()
```

```
def plot_confusion(accuracy, confusion, part):
    print(f'{part} Accuracy: {accuracy}')

    fig, ax = plt.subplots()
    im = ax.imshow(confusion)

    ax.set_xticks(range(num_cat))
    ax.set_yticks(range(num_cat))
    ax.set_xticklabels(categories)
    ax.set_yticklabels(categories)

    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    for i in range(num_cat):
        for j in range(num_cat):
            text = ax.text(j, i, confusion[i, j],
                           ha="center", va="center", color="w")

    ax.set_title(f"{part} Confusion Matrix (Actual / Predicted)")
    fig.tight_layout()
    plt.show()
```

In [7]:
```
plot_actualvpredicted(sepal_X, y, sepal_labels, 'Sepal')
plot_actualvpredicted(petal_X, y, petal_labels, 'Petal')
```

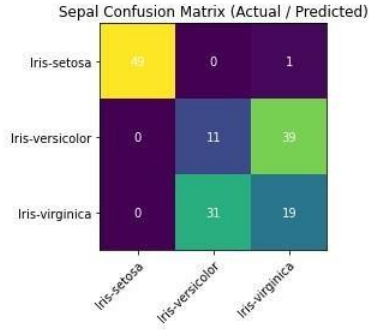




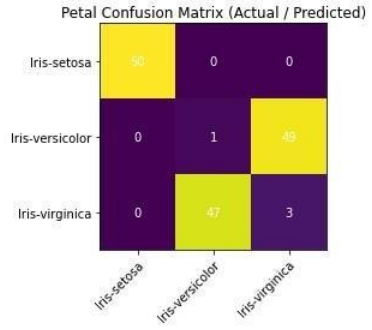
In [8]:
```
sepal_accuracy = accuracy_score(y, sepal_labels)
petal_accuracy = accuracy_score(y, petal_labels)

sepal_confusion = confusion_matrix(y, sepal_labels)
petal_confusion = confusion_matrix(y, petal_labels)
```

In [9]:
```
plot_confusion(sepal_accuracy, sepal_confusion, 'Sepal')
plot_confusion(petal_accuracy, petal_confusion, 'Petal')
```

```
Sepal Accuracy: 0.5266666666666666
```




```
Petal Accuracy: 0.36
```



Comparison with K-Means

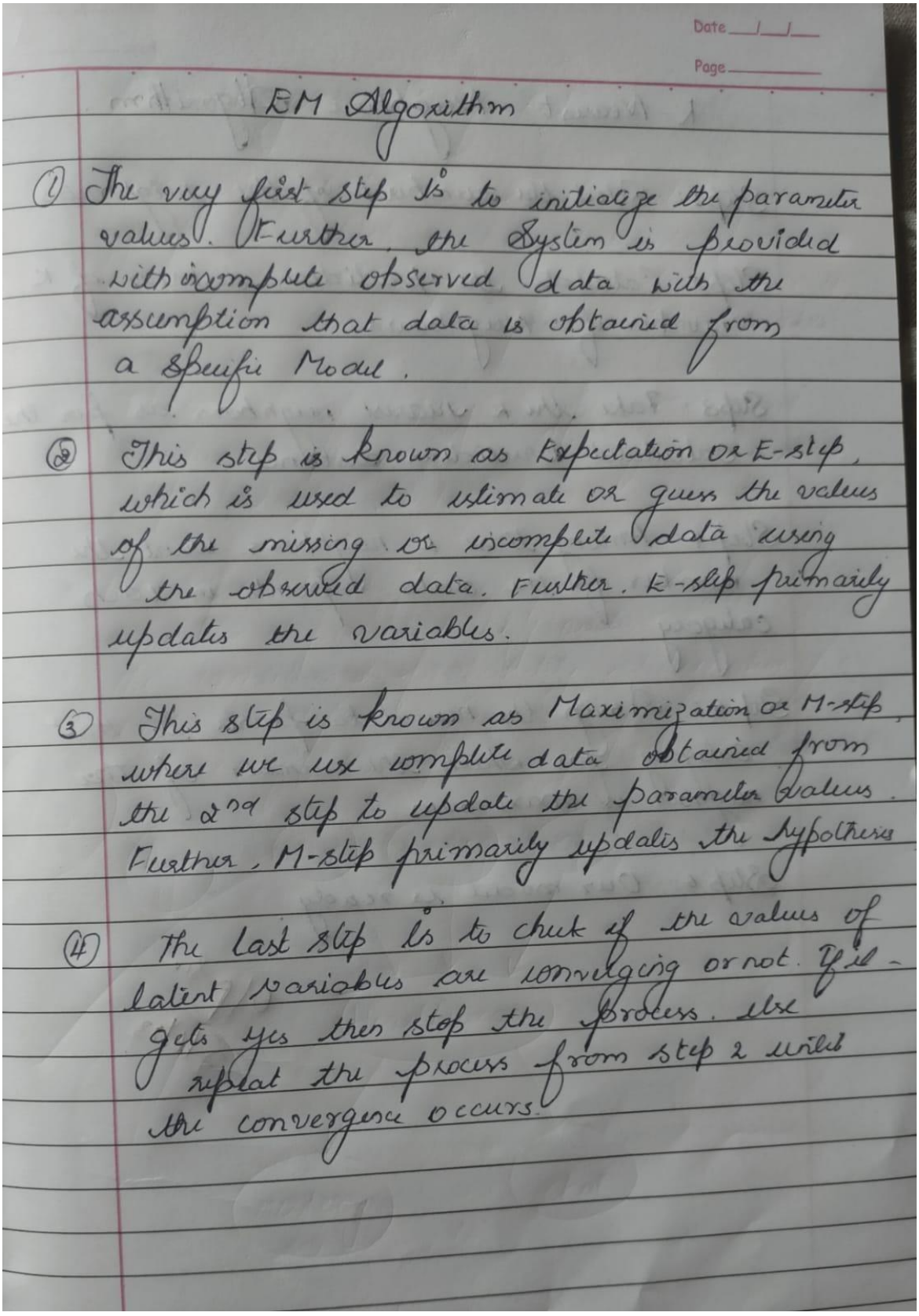
When we compare the results of the GaussianMixture model (uses EM algorithm) with that of K-Means clustering we observe that both give almost equal accuracies:

- K-Means: 0.5133333333333333 (*Sepal*) & 0.37333333333333335 (*Petal*)
- GaussianMixture: 0.5333333333333333 (*Sepal*) & 0.02 (*Petal*)

GaussianMixture performs slightly better when classified based on the *Sepal* length & width. Likewise K-Means performs slightly better when classified based on the *Petal* length & width.

In [ ]:
```
```

ALGORITHM:



Date __/__/__

Page _____

EM Algorithm

(1) The very first step is to initialize the parameter values. Further, the System is provided with incomplete observed data with the assumption that data is obtained from a Specific Model.

(2) This step is known as Expectation or E-step, which is used to estimate or guess the values of the missing or incomplete data using the observed data. Further, E-step primarily updates the variables.

(3) This step is known as Maximization or M-step, where we use complete data obtained from the 2nd step to update the parameter values. Further, M-step primarily updates the hypothesis.

(4) The last step is to check if the values of latent variables are converging or not. If it gets yes, then stop the process. Else repeat the process from step 2 until the convergence occurs.

Program 8: Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
```

Data

In [2]:
```
data = pd.read_csv('../input/iris/Iris.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [4]:
```
data.describe()
```

Out[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

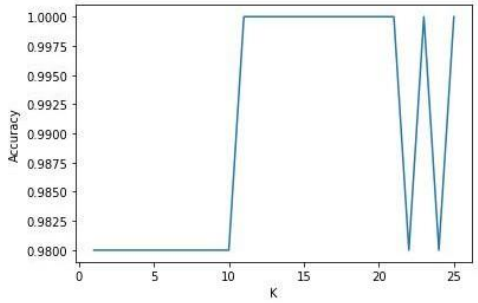
In [5]:
```
X = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
sepal_X = data[['SepalLengthCm', 'SepalWidthCm']]
petal_X = data[['PetalLengthCm', 'PetalWidthCm']]
y = data['Species'].astype("category").cat.codes
num_cat = y.nunique()
categories = data['Species'].astype("category").cat.categories
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Model

In [6]:
```
scores = {}
scores_list = []
for k in range(1, 26):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores[k] = accuracy_score(y_test, y_pred)
    scores_list.append(scores[k])
```

Analysis

In [7]:
```
plt.plot(range(1, 26), scores_list)
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.show()
```

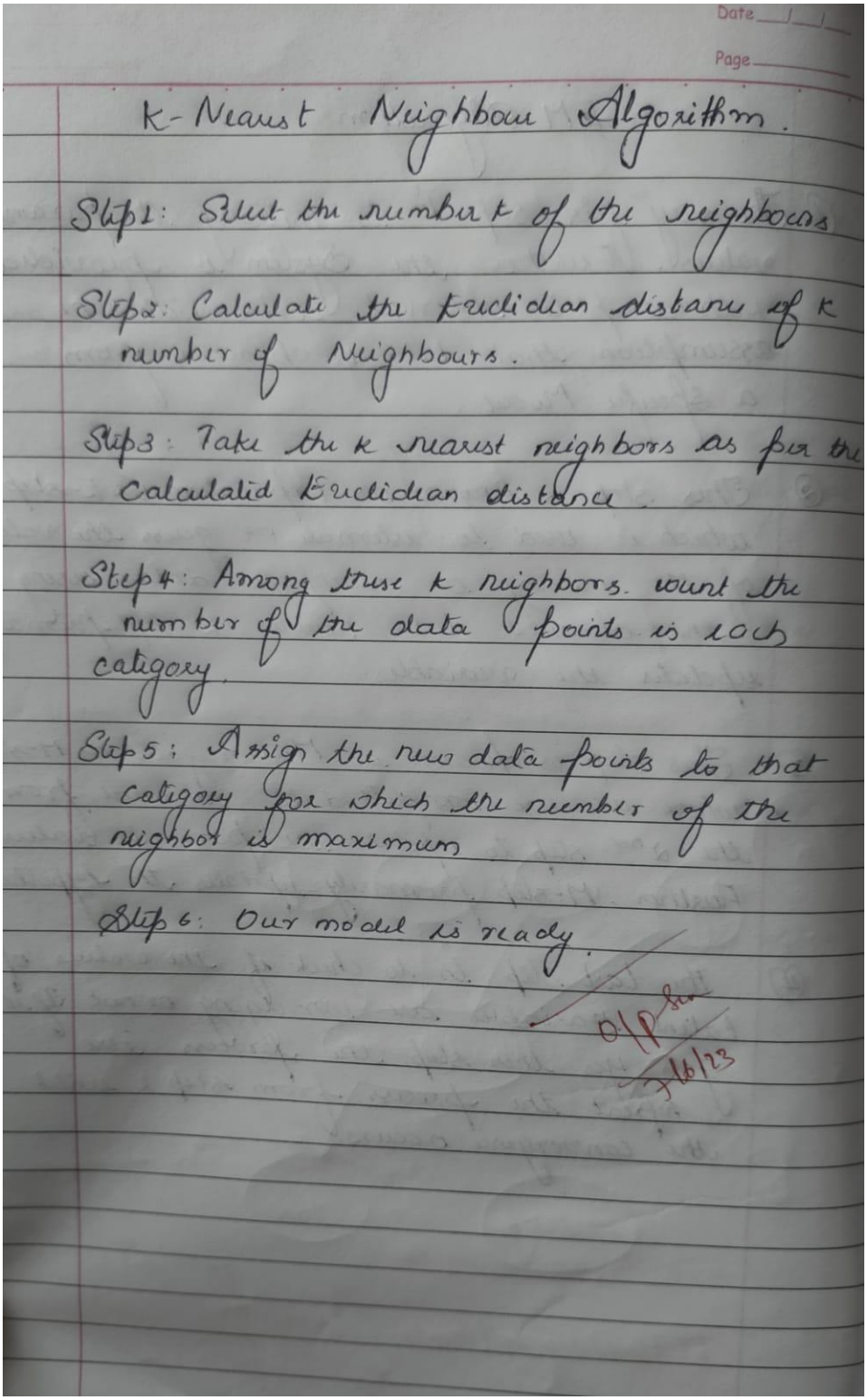


In [8]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
print(f"Correct Predictions: {accuracy_score(y_test, y_pred)}")
print(f"Wrong Predictions: {1 - accuracy_score(y_test, y_pred)}")
print("Accuracy Metrics: ")
print(classification_report(y_test,y_pred))
```

```
Confusion Matrix
[[19  0  0]
 [ 0 15  0]
 [ 0  1 15]]
Correct Predictions: 0.98
Wrong Predictions: 0.020000000000000018
Accuracy Metrics:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.94      1.00      0.97        15
           2       1.00      0.94      0.97        16

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50
```

In [ ]:

ALGORITHM:



Date / /

Page

K-Nearst Neighbour Algorithm.

Step1: Slect the number k of the neighbours

Step2: Calculate the Euclidian distance of K number of Neighbours.

Step3: Take the k nearst neighbors as per the calculalid Euclidian distance.

Step4: Among these k neighbors. count the number of the data points in each category.

Step5: Assign the new data points to that category for which the number of the neighbor is maximum

Step6: Our model is ready.

O/P

7/6/23

Program 9: Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

Data

In [2]:
```
df = pd.read_csv('../input/housesalesprediction/kc_house_data.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680

5 rows × 21 columns

In [4]:
```
X = np.array(df['sqft_living']).reshape(-1, 1)
y = df['price']
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Model

In [6]:
```
model = LinearRegression()
model.fit(X_train, y_train)
# model.fit(X, y)
```

Out[6]:
```
LinearRegression()
```

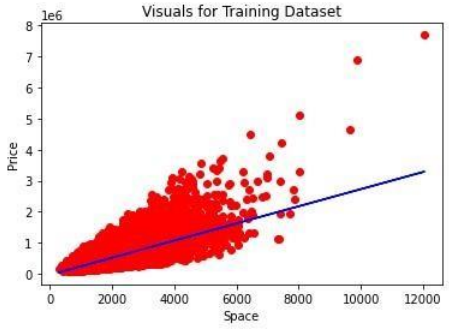
Analysis

In [7]:
```
y_pred = model.predict(X_test)
print(f"Mean Squared Error: {mean_squared_error(y_pred, y_test)}")
```

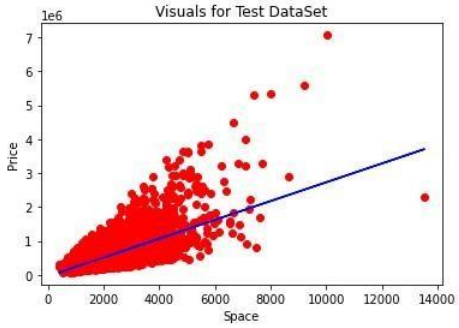
```
Mean Squared Error: 76715223988.35832
```

In [8]:
```
#Visualizing the training Test Results
plt.scatter(X_train, y_train, color= 'red')
```

In [8]:
```
#Visualizing the training Test Results
plt.scatter(X_train, y_train, color= 'red')
plt.plot(X_train, model.predict(X_train), color = 'blue')
plt.title ("Visuals for Training Dataset")
plt.xlabel("Space")
plt.ylabel("Price")
plt.show()
```

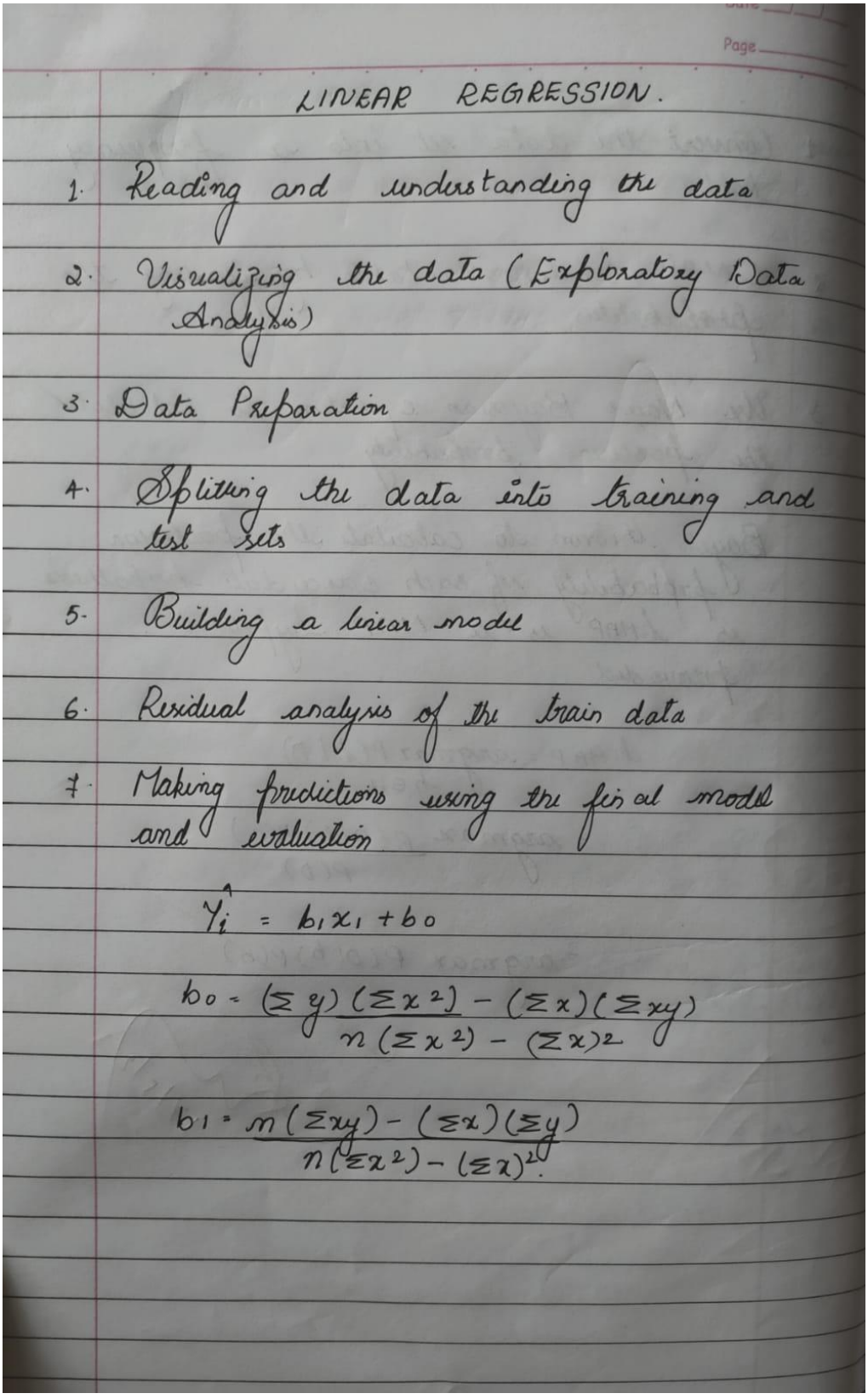


In [9]:
```
#Visualizing the Test Results
plt.scatter(X_test, y_test, color= 'red')
plt.plot(X_test, model.predict(X_test), color = 'blue')
plt.title("Visuals for Test DataSet")
plt.xlabel("Space")
plt.ylabel("Price")
plt.show()
```



In [ ]:

ALGORITHM:



LINEAR REGRESSION.

1. Reading and understanding the data
2. Visualizing the data (Exploratory Data Analysis)
3. Data Preparation
4. Splitting the data into training and test sets
5. Building a linear model
6. Residual analysis of the train data
7. Making predictions using the final model and evaluation

$$\hat{Y}_i = b_1 x_1 + b_0$$

$$b_0 = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$b_1 = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2}$$

Program 10: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Functions

In [2]:
```
# kernel smoothing function
def kernel(point, xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m)))

    for j in range(m):
        diff = point - X[j]
        weights[j, j] = np.exp(diff * diff.T / (-2.0 * k**2))

    return weights

# function to return local weight of eah training example
def localWeight(point, xmat, ymat, k):
    wt = kernel(point, xmat, k)
    W = (X.T * (wt*X)).I * (X.T * wt * ymat.T)
    return W

# root function that drives the algorithm
def localWeightRegression(xmat, ymat, k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)

    for i in range(m):
        ypred[i] = xmat[i] * localWeight(xmat[i], xmat, ymat, k)

    return ypred
```

Data

In [3]:
```
#import data
data = pd.read_csv('../input/tipsdata/tips.csv')

# place them in suitable data types
colA = np.array(data.total_bill)
colB = np.array(data.tip)

mcolA = np.mat(colA)
mcolB = np.mat(colB)

m = np.shape(mcolB)[1]
one = np.ones((1, m), dtype = int)

# horizontal stacking
X = np.hstack((one.T, mcolA.T))
print(X.shape)
```

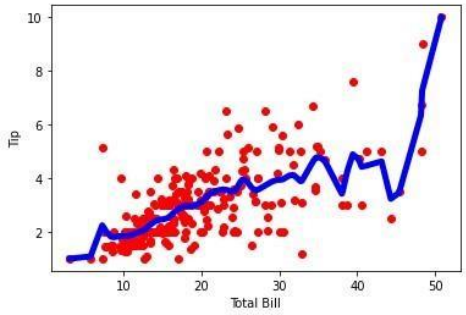
```
(244, 2)
```

Model

In [4]:
```
# predicting values using LWLR
ypred = localWeightRegression(X, mcolB, 0.8)
```

Analysis

In [5]:
```
# plotting the predicted graph
xsort = X.copy()
xsort.sort(axis=0)
plt.scatter(colA, colB, color='red')
plt.plot(xsort[:, 1], ypred[X[:, 1].argsort(0)], color='blue', linewidth=5)
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```



In [ ]:

ALGORITHM:

Date __/__/__

Page _____

Locally Weighted Regression Algorithm

1. Read the given data sample to X and the curve (linear or non linear) to Y

2. Set the value for Smoothening parameter or Free parameter say τ

3. Set the bias / Point of interset set x_0 which is a subset of X.

4. Determine the weight matrix using:

$$w(x, x_0) = e^{-\frac{(x-x_0)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using:

$$\hat{\beta}(x_0) = (X^T W X)^{-1} X^T W y$$

6. Predict = $x_0 * \beta$

o/p seen

3/6/23