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:

Ajay Mittur

1BM18CS006

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
Mar-June 2021

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 Ajay Mittur (1BM18CS006) during the 6th Semester Mar-June-2021.

Signature of the Faculty Incharge:

NAME OF THE FACULTY:

Department of Computer Science and Engineering
B.M.S. College of Engineering, Bangalore

Contents

Program 1: Find S Algorithm

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)
```

Program 2: Candidate Elimination Algorithm

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)
```

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', '?', '?', '?', '?']])
```

Program 3: Decision Tree (ID3)

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 [ ]:

Program 4: Naïve Bayes Classifier

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 = {}
```

```python
    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]:

```python
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 [ ]:

Program 5: Bayesian Network

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 [ ]:

Program 6: K-Means (Iris Dataset)

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

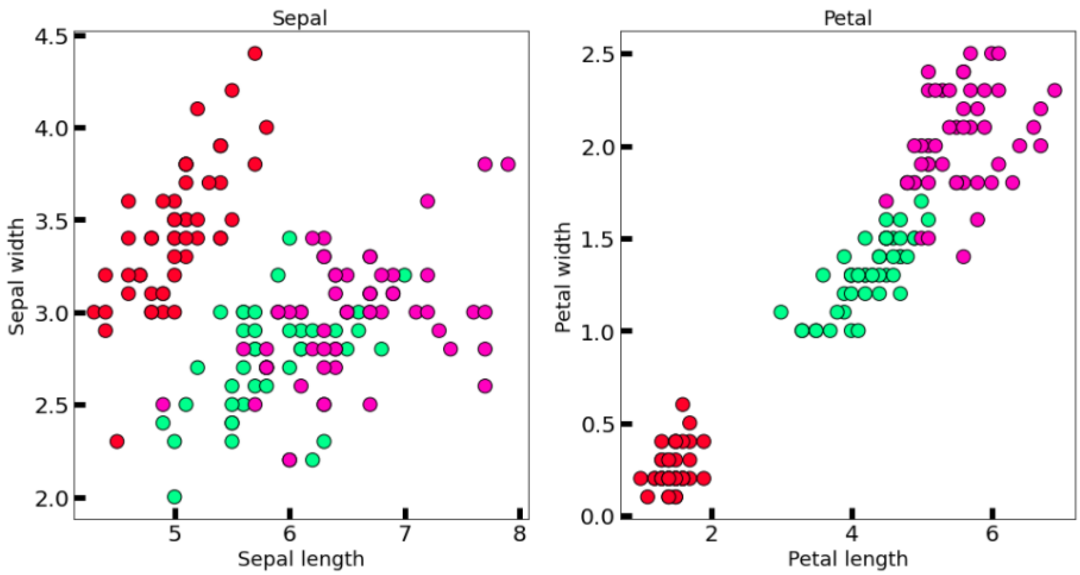
In [2]:
```
iris = datasets.load_iris()
iris
```

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]),
 '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    :Num
ber of Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4 numeric, predictive attributes and t
he class\n    :Attribute Information:\n        - sepal length in cm\n        - sepal width in cm\n        - petal leng
th in cm\n        - petal width in cm\n        - class:\n                - Iris-Setosa\n                - Iris-Versico
lour\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    :Mis
sing Attribute Values: None\n    :Class Distribution: 33.3% for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Do
nor: 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\n
Machine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known database to be found i
n the\npattern recognition literature.  Fisher\'s paper is a classic in the field and\nis referenced frequently to thi
s day.  (See Duda & Hart, for example.)  The\ndata set contains 3 classes of 50 instances each, where each class refer
s 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 problem
s"\n     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistics" (John W
iley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n     (Q327.D83) Joh
n Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A Ne
w System\n     Structure and Classification Rule for Recognition in Partially Exposed\n     Environments".  IEEE Trans
actions on Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The R
educed 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 dat
a.\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)
```

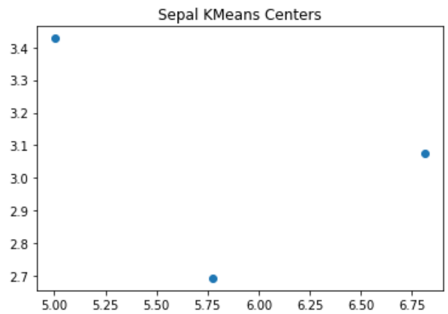
Analysis

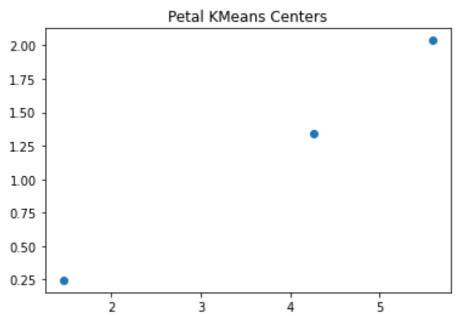
In [5]:
```
def plot_centers(sepal_centers, petal_centers):
    plt.scatter([point[0] for point in sepal_centers], [point[1] for point in sepal_centers])
    plt.title('Sepal KMeans Centers')
    plt.show()
```

```
    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)
```

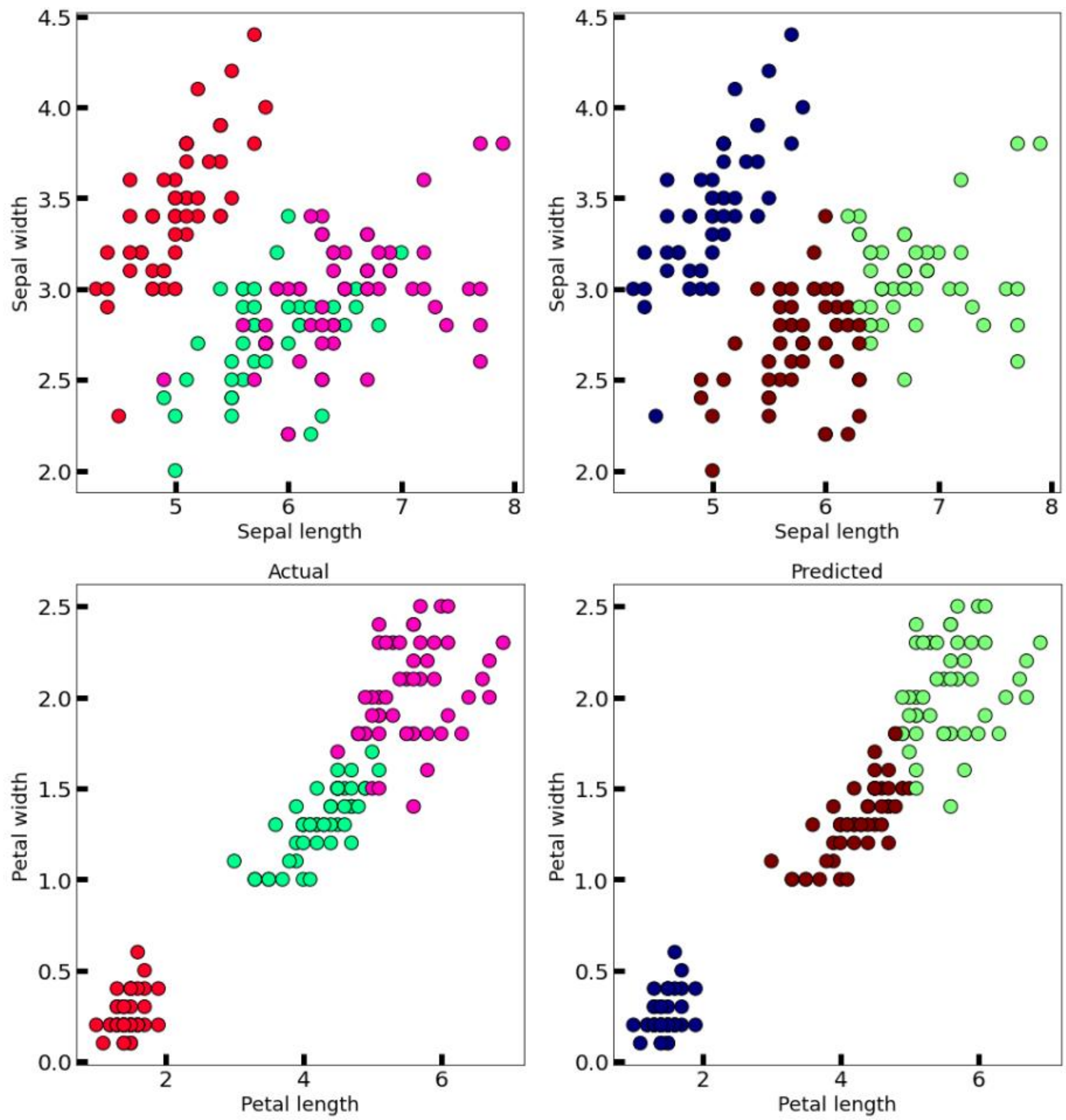




In [7]:
```python
sepal_labels = model_sepal.labels_
petal_labels = model_petal.labels_
print(sepal_labels, petal_labels, sep='\n')
```

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

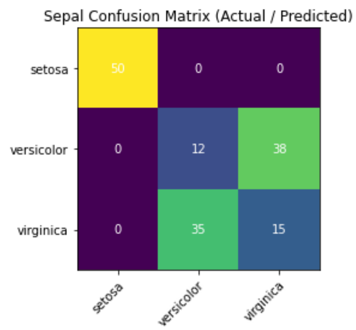
In [8]:
```python
plot_actualvpredicted(sepal_X, y, sepal_labels, 'Sepal')
plot_actualvpredicted(petal_X, y, petal_labels, 'Petal')
```


Sepal width
Sepal length
Actual
Predicted
Petal width
Petal length

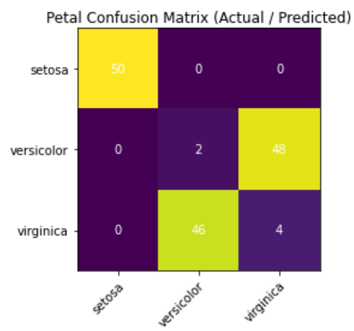
In [9]:
```
sepal_accuracy = accuracy_score(y, sepal_labels)
petal_accuracy = accuracy_score(y, petal_labels)

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
```



Program 7: EM vs K-Means (Iris Dataset)

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

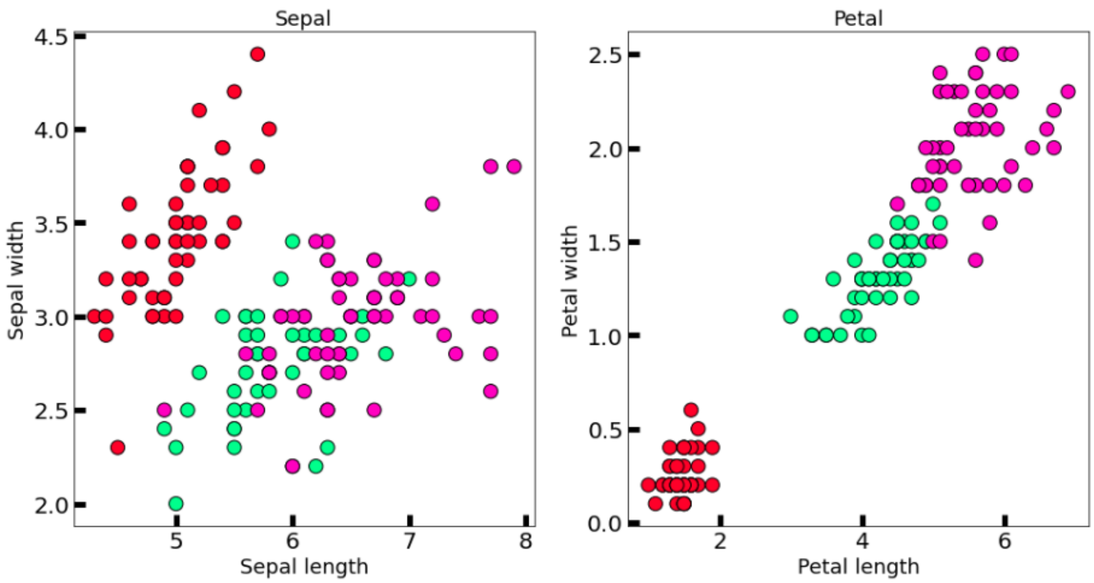
In [2]:
```
data = pd.read_csv('../input/iris/Iris.csv')
data.head()
```

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

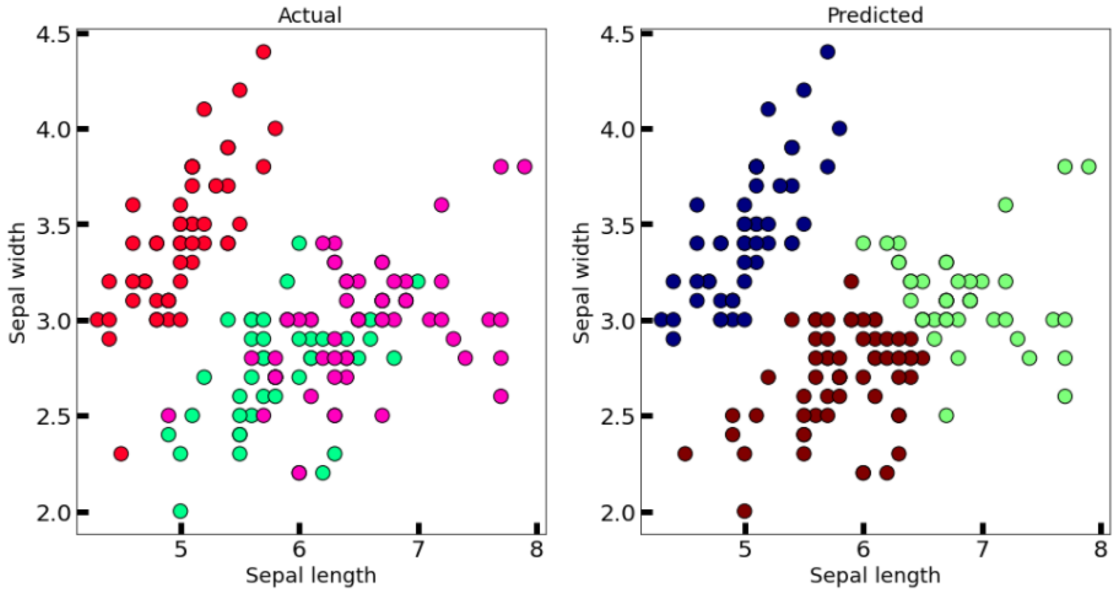
In [5]:
```
model_sepal = GaussianMixture(n_components=3)
sepal_labels = model_sepal.fit_predict(sepal_X)
model_petal = GaussianMixture(n_components=3)
petal_labels = model_petal.fit_predict(petal_X)
```

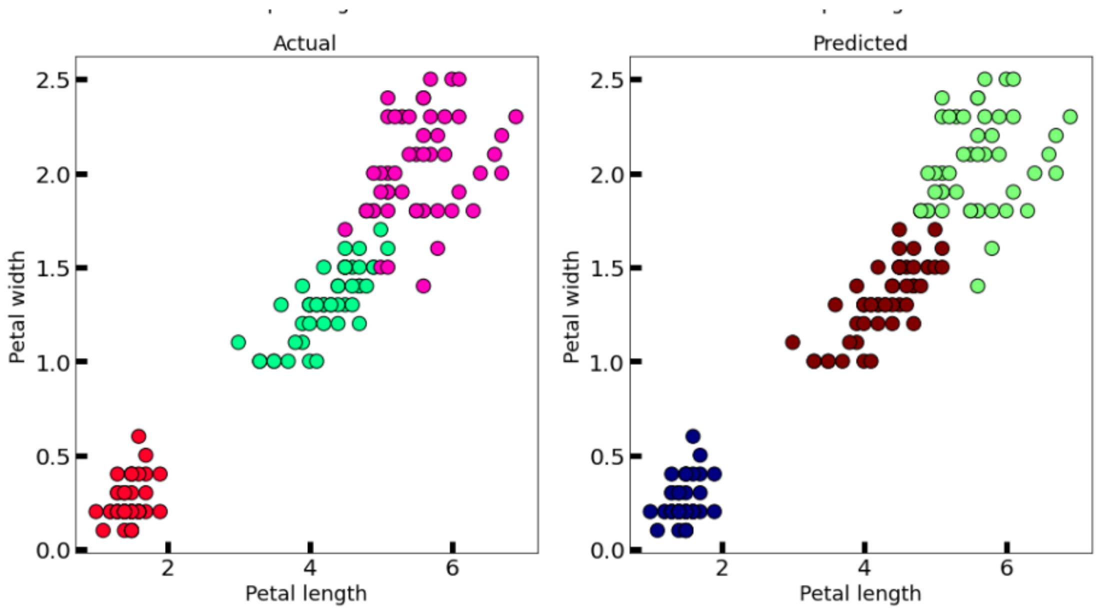
Analysis

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()
```

```python
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]:
```python
plot_actualvpredicted(sepal_X, y, sepal_labels, 'Sepal')
plot_actualvpredicted(petal_X, y, petal_labels, 'Petal')
```

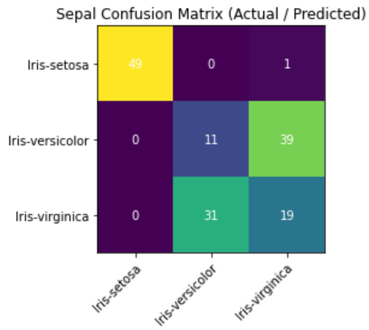




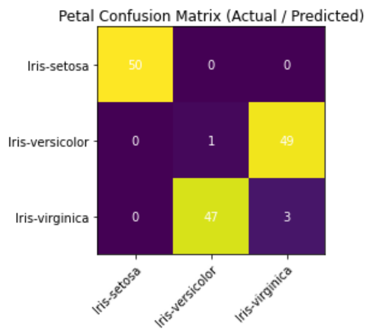
In [8]:
```
sepal_accuracy = accuracy_score(y, sepal_labels)
petal_accuracy = accuracy_score(y, petal_labels)

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 [ ]:

Program 8: KNN (Iris Dataset)

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

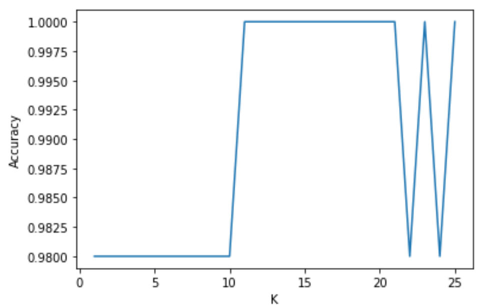
In [5]:
```
X = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
sepal_X = data[['SepalLengthCm', 'SepalWidthCm']]
petal_X = data[['PetalLengthCm', 'PetalWidthCm']]
y = data['Species'].astype("category").cat.codes
num_cat = y.nunique()
categories = data['Species'].astype("category").cat.categories
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 [ ]:

Program 9: Linear Regression

In [1]:
```python
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]:
```python
df = pd.read_csv('../input/housesalesprediction/kc_house_data.csv')
```

In [3]:
```python
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]:
```python
X = np.array(df['sqft_living']).reshape(-1, 1)
y = df['price']
```

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

Model

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

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

Analysis

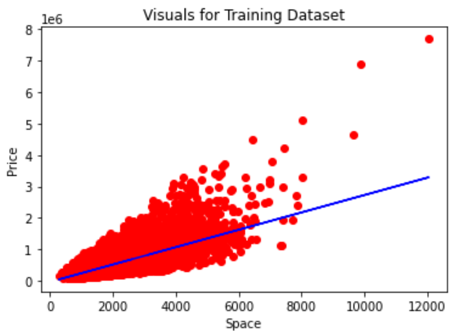
In [7]:
```python
y_pred = model.predict(X_test)
print(f"Mean Squared Error: {mean_squared_error(y_pred, y_test)}")
```

```
Mean Squared Error: 76715223988.35832
```

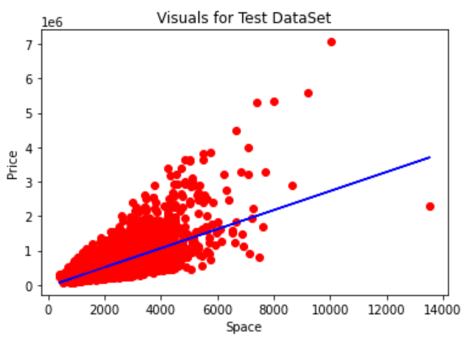
In [8]:
```python
#Visualizing the training Test Results
plt.scatter(X_train, y_train, color= 'red')
```

```
Mean Squared Error: [illegible]
```

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 [ ]:
```
```

Program 10: Locally Weighted Regression

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 traiining 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)
```

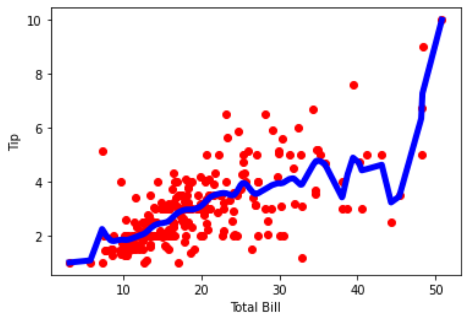
```
(244, 2)
```

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 [ ]: