INDEX

1. For a given set of training data examples stored in a .CSV file, implement and demonstrate the <u>Candidate-Elimination</u> algorithm to output a description of the set of all hypotheses consistent with the training examples.

```
import csv

a = []

print("\n The Given Training Data Set")

with open('enjoysport.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        a.append (row)
        print(row)

num_attributes = len(a[0])-1

print("\n The initial value of hypothesis: ")

S = ['0'] * num_attributes

G = ['?'] * num_attributes

print ("\n The most specific hypothesis S0 : [0,0,0,0,0,0]")

print (" \n The most general hypothesis G0 : [?,?,?,?,?,?]")

for j in range(0,num_attributes):
    S[j]=a[0][j]

print("\n Candidate Elimination algorithm Hypotheses Version Space
Computation\n")

temp=[]

for i in range(0,len(a)):
    if a[i][num_attributes]=='Yes':
        for j in range(0,num_attributes):
            if a[i][j]!=S[j]:
                S[j]='?'
        for j in range(0,num_attributes):
            for k in range(1,len(temp)):
                if temp[k][j]!='?' and temp[k][j]!=S[j]:
                    del temp[k]
```

```
        print("-------------------------------------------------------------------
----------- ")
        print(" For Training Example No :{0} the hypothesis is S{0}
".format(i+1),S)
        if (len(temp)==0):
            print(" For Training Example No :{0} the hypothesis is G{0}
".format(i+1),G)
        else:
            print(" For  Positive Training Example No :{0} the hypothesis is
G{0}".format(i+1),temp)
    if a[i][num_attributes]=='No':
        for j in range(0,num_attributes):
            if S[j] != a[i][j] and S[j]!= '?':
                G[j]=S[j]
                temp.append(G)
                G = ['?'] * num_attributes

        print("-------------------------------------------------------------------
----------- ")

        print(" For Training Example No :{0} the hypothesis is S{0}
".format(i+1),S)

        print(" For Training Example No :{0} the hypothesis is
G{0}".format(i+1),temp)
```

Output

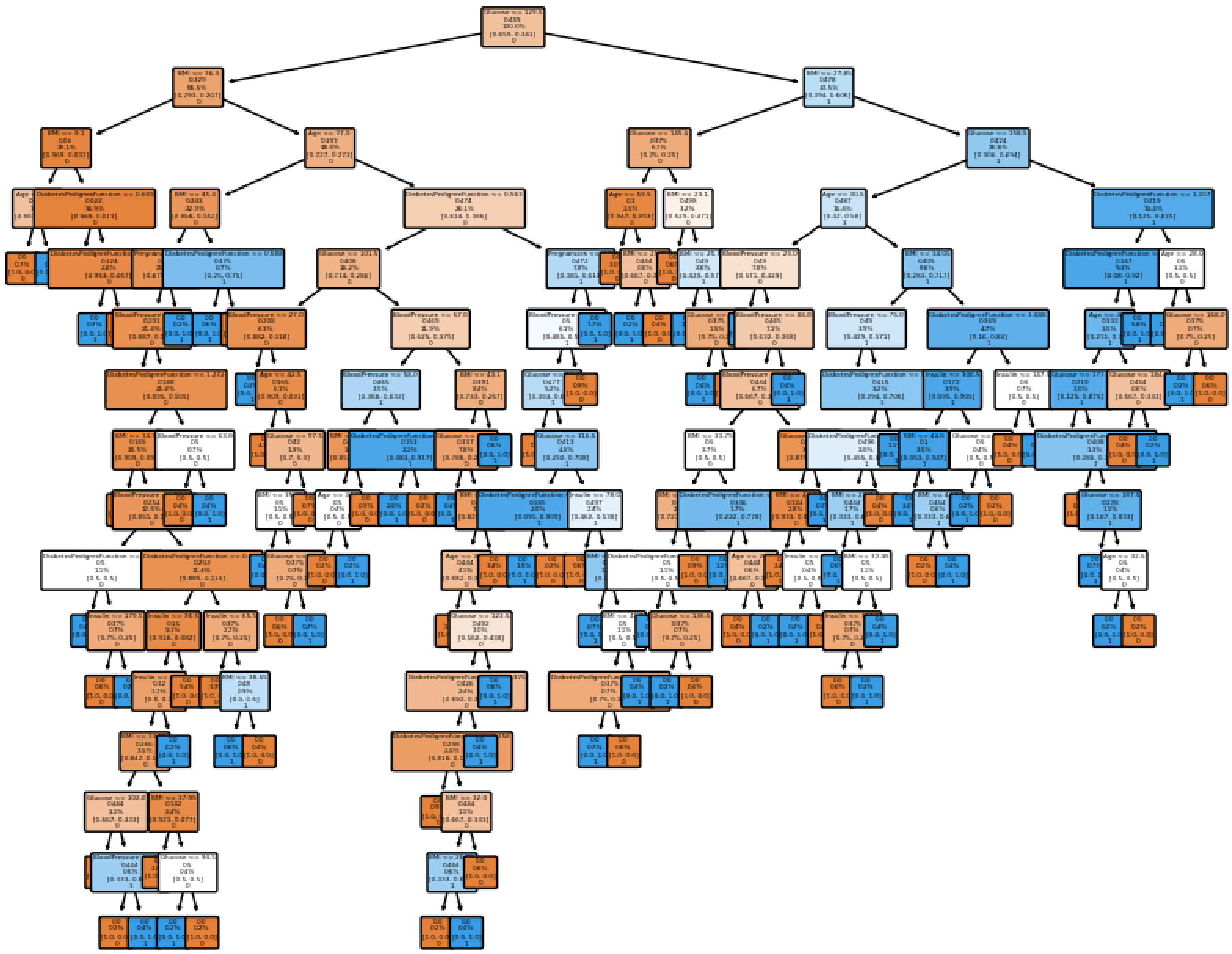
```
The Given Training Data Set
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'Yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'Yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'No']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'Yes']

 The initial value of hypothesis:

 The most specific hypothesis S0 : [0,0,0,0,0,0]

 The most general hypothesis G0 : [?,?,?,?,?,?]

 Candidate Elimination algorithm Hypotheses Version Space Computation

-----------------------------------------------------------------------------
 For Training Example No :1 the hypothesis is S1  ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
 For Training Example No :1 the hypothesis is G1  ['?', '?', '?', '?', '?', '?']
-----------------------------------------------------------------------------
 For Training Example No :2 the hypothesis is S2  ['sunny', 'warm', '?', 'strong', 'warm', 'same']
 For Training Example No :2 the hypothesis is G2  ['?', '?', '?', '?', '?', '?']
-----------------------------------------------------------------------------
 For Training Example No :3 the hypothesis is S3  ['sunny', 'warm', '?', 'strong', 'warm', 'same']
 For Training Example No :3 the hypothesis is G3 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
-----------------------------------------------------------------------------
 For Training Example No :4 the hypothesis is S4  ['sunny', 'warm', '?', 'strong', '?', '?']
 For  Positive Training Example No :4 the hypothesis is G4 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

2. Demonstrate the working of the <u>Decision Tree</u>. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
import pandas as pd # Load libraries
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split
function
from sklearn import metrics #Import scikit-learn metrics module for accuracy
calculation
from sklearn import tree #Import scikit-learn tree module for plotting
import matplotlib.pyplot as plt #Import matplotlib for plotting

# load dataset
pima = pd.read_csv("diabetes.csv")
pima.columns

#split dataset in features and target variable
feature_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age', 'Glucose',
'BloodPressure', 'DiabetesPedigreeFunction']
X = pima[feature_cols] # Features
y = pima.Outcome # Target variable

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=1) # 70% training and 30% test

# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

plt.figure(figsize=(10,8))
tree.plot_tree(clf, feature_names=feature_cols, class_names=['0','1'], label=all,
filled=True, proportion=True, rounded=True, fontsize=3)
print('Number of correct predictions are')
diff = list(y_pred - y_test)
print(diff.count(0)," out of ",len(diff))
```

Output

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')

Accuracy: 0.683982683982684
```



3. Build an <u>Artificial Neural Network</u> by testing the same using appropriate data sets.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

data = pd.read_csv('digit-recognizer/train.csv')

data = np.array(data)
m, n = data.shape
np.random.shuffle(data) # shuffle before splitting into dev and training sets

data_dev = data[0:1000].T
Y_dev = data_dev[0]
X_dev = data_dev[1:n]
X_dev = X_dev / 255.

data_train = data[1000:m].T
Y_train = data_train[0]
X_train = data_train[1:n]
X_train = X_train / 255.
_,m_train = X_train.shape

Y_train

def init_params():
    W1 = np.random.rand(10, 784) - 0.5
    b1 = np.random.rand(10, 1) - 0.5
    W2 = np.random.rand(10, 10) - 0.5
    b2 = np.random.rand(10, 1) - 0.5
    return W1, b1, W2, b2

def ReLU(Z):
    return np.maximum(Z, 0)

def softmax(Z):
    A = np.exp(Z) / sum(np.exp(Z))
    return A

def forward_prop(W1, b1, W2, b2, X):
    Z1 = W1.dot(X) + b1
    A1 = ReLU(Z1)
    Z2 = W2.dot(A1) + b2
    A2 = softmax(Z2)
```

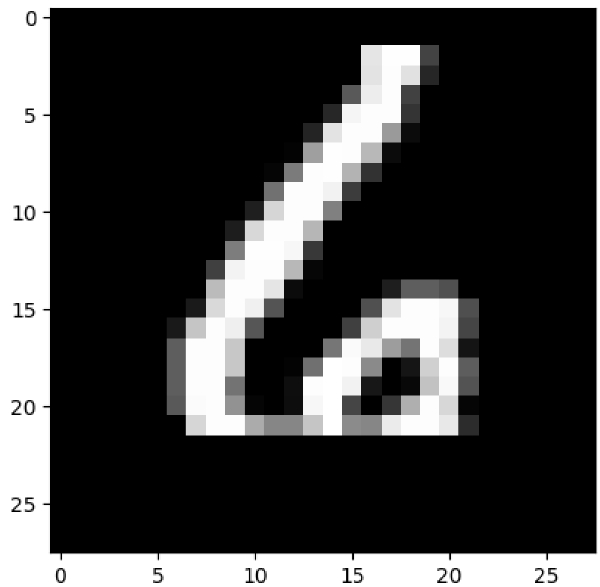
```
    return Z1, A1, Z2, A2

def ReLU_deriv(Z):
    return Z > 0

def one_hot(Y):
    one_hot_Y = np.zeros((Y.size, Y.max() + 1))
    one_hot_Y[np.arange(Y.size), Y] = 1
    one_hot_Y = one_hot_Y.T
    return one_hot_Y

def backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y):
    one_hot_Y = one_hot(Y)
    dZ2 = A2 - one_hot_Y
    dW2 = 1 / m * dZ2.dot(A1.T)
    db2 = 1 / m * np.sum(dZ2)
    dZ1 = W2.T.dot(dZ2) * ReLU_deriv(Z1)
    dW1 = 1 / m * dZ1.dot(X.T)
    db1 = 1 / m * np.sum(dZ1)
    return dW1, db1, dW2, db2

def update_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha):
    W1 = W1 - alpha * dW1
    b1 = b1 - alpha * db1
    W2 = W2 - alpha * dW2
    b2 = b2 - alpha * db2
    return W1, b1, W2, b2

def get_predictions(A2):
    return np.argmax(A2, 0)

def get_accuracy(predictions, Y):
    print(predictions, Y)
    return np.sum(predictions == Y) / Y.size

def gradient_descent(X, Y, alpha, iterations):
    W1, b1, W2, b2 = init_params()
    for i in range(iterations):
        Z1, A1, Z2, A2 = forward_prop(W1, b1, W2, b2, X)
        dW1, db1, dW2, db2 = backward_prop(Z1, A1, Z2, A2, W1, W2, X, Y)
        W1, b1, W2, b2 = update_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha)
        if i % 10 == 0:
            print("Iteration: ", i)
            predictions = get_predictions(A2)
            print(get_accuracy(predictions, Y))
```

```
    return W1, b1, W2, b2
W1, b1, W2, b2 = gradient_descent(X_train, Y_train, 0.10, 500)

def make_predictions(X, W1, b1, W2, b2):
    _, _, _, A2 = forward_prop(W1, b1, W2, b2, X)
    predictions = get_predictions(A2)
    return predictions

def test_prediction(index, W1, b1, W2, b2):
    current_image = X_train[:, index, None]
    prediction = make_predictions(X_train[:, index, None], W1, b1, W2, b2)
    label = Y_train[index]
    print("Prediction: ", prediction)
    print("Label: ", label)

    current_image = current_image.reshape((28, 28)) * 255
    plt.gray()
    plt.imshow(current_image, interpolation='nearest')
    plt.show()

test_prediction(0, W1, b1, W2, b2)
test_prediction(1, W1, b1, W2, b2)
test_prediction(2, W1, b1, W2, b2)
test_prediction(3, W1, b1, W2, b2)

dev_predictions = make_predictions(X_dev, W1, b1, W2, b2)
get_accuracy(dev_predictions, Y_dev)
```

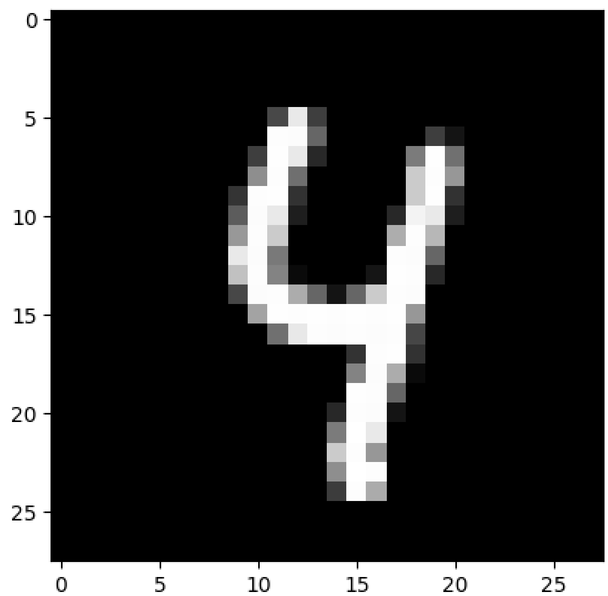
Output

```
array([6, 4, 7, ..., 4, 1, 0], dtype=int64)

Output exceeds the size limit. Open the full output data in a text editor
Iteration:  0
[6 2 6 ... 6 6 6] [6 4 7 ... 4 1 0]
0.09565853658536586
Iteration:  10
[4 8 9 ... 8 9 1] [6 4 7 ... 4 1 0]
0.1646829268292683
Iteration:  20
[4 8 9 ... 8 6 0] [6 4 7 ... 4 1 0]
0.2276829268292683
Iteration:  30
[4 8 9 ... 8 6 0] [6 4 7 ... 4 1 0]
0.2939512195121951
Iteration:  40
[4 4 9 ... 8 6 0] [6 4 7 ... 4 1 0]
0.3631951219512195
Iteration:  50
[4 4 3 ... 8 6 0] [6 4 7 ... 4 1 0]
0.4305121951219512
Iteration:  60
[6 4 3 ... 7 6 0] [6 4 7 ... 4 1 0]
0.48236585365853657
Iteration:  70
[6 4 3 ... 9 6 0] [6 4 7 ... 4 1 0]
0.5249512195121951
Iteration:  80
...
0.8472439024390244
Iteration:  490
[6 4 2 ... 9 8 0] [6 4 7 ... 4 1 0]
0.8479756097560975
```

```
Prediction:  [6]
Label:  6
```



```
Prediction:  [4]
Label:  4
```



```
[8 2 8 6 3 9 5 3 5 6 1 4 6 0 9 0 7 5 1 1 9 3 3 3 1 2 1 4 2 7 4 8 9 4 6 7 1
 7 9 7 5 1 1 3 9 1 1 9 1 5 2 4 8 4 3 6 8 6 6 9 8 9 7 6 9 1 8 1 6 4 4 6 5 3
 6 5 1 3 2 5 9 1 9 7 6 3 4 3 4 3 5 6 2 2 2 6 9 6 5 5 8 1 5 7 0 9 6 5 0 2 2
 3 5 5 0 8 6 7 2 2 1 9 8 9 9 2 1 2 8 4 8 6 0 8 8 0 5 7 7 4 9 3 5 1 9 6 2 1
 9 9 5 4 1 1 2 0 2 7 3 1 6 5 7 3 6 4 0 2 4 1 3 1 5 9 0 6 9 9 8 7 8 6 7 3 1
 2 2 2 4 5 8 4 3 1 0 2 9 1 0 4 4 2 0 8 4 8 3 5 0 9 2 2 0 1 9 7 2 6 2 0 7 1
 8 6 8 1 9 8 9 4 9 5 1 1 4 5 4 9 2 1 3 0 8 9 2 6 8 3 8 1 4 0 2 4 0 8 3 7 8
 3 6 2 8 3 1 8 7 9 4 7 7 4 8 9 3 0 7 4 0 5 4 3 7 2 6 1 2 0 5 0 6 9 0 6 6 1
 1 8 7 3 4 1 3 4 6 5 2 0 3 6 4 7 1 6 4 9 5 7 8 8 6 1 2 2 4 4 9 2 3 6 6 0 6
 9 1 8 2 2 8 9 9 5 7 8 2 9 9 7 6 0 7 1 6 6 0 6 6 7 6 2 6 3 8 2 6 0 3 8 0 1
 2 0 3 0 4 1 0 7 1 2 4 6 3 4 8 3 7 2 8 4 8 6 2 1 0 3 2 9 6 0 7 6 6 0 4 9 3
 6 1 8 4 5 6 0 9 4 0 7 6 7 5 9 4 7 6 9 2 2 5 5 6 2 6 4 7 5 2 6 3 3 4 3 3 4
 5 2 8 6 2 7 7 1 2 7 7 1 7 3 3 0 1 1 0 8 7 4 7 8 9 5 1 5 2 6 9 8 7 7 3 1 8
 4 6 0 3 4 2 5 2 7 6 5 1 6 2 5 8 7 4 1 3 1 7 3 9 4 0 8 9 5 8 1 8 8 3 3 1 9
 3 8 2 7 4 8 8 5 6 3 2 4 8 3 7 0 0 5 6 7 1 0 8 2 8 5 8 8 2 4 3 1 7 5 2 6 1
 6 7 8 5 9 6 3 6 2 7 2 1 0 4 1 3 6 0 6 1 0 7 7 6 9 0 8 2 0 4 3 6 7 3 1 4 0
 9 6 3 7 1 1 3 5 8 2 6 8 8 4 2 8 7 2 8 0 9 9 4 7 7 6 5 1 1 8 2 3 6 1 3 8 6
 4 1 7 8 1 2 2 7 3 4 2 3 5 6 9 4 6 0 1 7 1 1 2 6 4 7 1 7 7 8 8 9 8 3 0 4 8
 8 3 4 4 4 1 7 2 6 2 1 9 7 4 7 7 4 5 2 4 1 7 1 5 0 2 8 2 4 5 8 9 4 0 0 1 8
 7 7 9 7 1 7 7 2 9 4 4 7 2 0 9 1 6 6 2 0 2 2 4 8 1 8 2 4 0 5 5 3 6 8 9 0 6
 1 2 6 1 8 5 9 0 7 6 4 0 6 2 4 3 1 5 0 9 7 7 3 6 4 8 5 2 8 7 4 8 3 9 9 1 6
 1 2 1 5 7 7 7 8 3 9 4 7 0 5 4 9 6 4 2 1 0 3 9 7 9 9 4 4 9 6 4 6 4 5 4 6 2
 3 5 9 6 8 4 9 0 9 6 1 3 8 0 0 7 4 1 5 5 2 1 0 7 5 0 8 5 0 6 7 1 0 3 8 6 7
 0 5 7 7 5 5 5 4 0 2 8 6 1 9 8 4 3 9 4 6 3 0 5 2 2 0 6 9 1 8 4 3 8 4 8 1 7
 2 3 7 3 1 8 1 6 5 3 4 0 9 8 7 3 6 6 8 9 3 1 5 3 1 6 7 9 2 6 1 0 0 3 9 1 8
...
 2 3 7 3 1 3 1 6 5 3 4 0 9 8 7 5 6 6 2 9 5 1 5 3 1 3 7 9 2 6 1 0 0 3 7 1 8
 3 6 0 6 0 9 9 5 9 1 4 8 7 8 1 4 7 0 3 7 4 2 1 1 6 4 7 4 5 1 6 7 0 5 1 5 1
 6 1 1 0 9 8 8 5 6 6 2 7 7 5 1 0 2 0 8 7 0 4 4 6 1 8 1 7 3 1 7 2 1 6 0 9 7
 1]
```

```
0.843
```

4. Implement the <u>Naïve Bayesian Classifier</u> for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

X

y

model = GaussianNB()
y_pred = model.fit(X_train, y_train).predict(X_test)
y_pred

y_test

diff = y_pred - y_test
diff

print('Number of incorrect classifications are: ',len(y_pred) -
list(diff).count(0))
```

Output

```
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],
...
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 0,
       0])

array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 0,
       0])

array([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])

Number of incorrect classifications are:  0
```

5. Implement Find S Algorithm.

```
import csv

def read_csv(filename):
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        traindata = []
        for row in datareader:
            traindata.append(row)
        print(traindata)
    return traindata

attributes = ['Sky', 'Temp', 'Humidity', 'Wind', 'Water', 'Forecast']
print("Attributes: ",attributes)
num_attributes = len(attributes)

dataset = read_csv("EnjoySport.csv")

dataset

len(dataset)

hypothesis = ['0']*len(attributes)
hypothesis

print("Initial Hypothesis: ",hypothesis)

for j in range(0, num_attributes):
    hypothesis[j] = dataset[0][j]

print("Find S: Finding a Maximally Specific Hypothesis")
for i in range(len(dataset)):
    if dataset[i][num_attributes] == "Yes":
        for j in range(num_attributes):
            if dataset[i][j] != hypothesis[j]:
                hypothesis[j] = "?"
    else:
        continue

i

print("Hypothesis")
print(hypothesis)
```

Output

```
Attributes:  ['Sky', 'Temp', 'Humidity', 'Wind', 'Water', 'Forecast']
[['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'Yes'], ['sunny', 'warm',
'high', 'strong', 'warm', 'same', 'Yes'], ['rainy', 'cold', 'high', 'strong',
'warm', 'change', 'No'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change',
'Yes']]

Initial Hypothesis:  ['0', '0', '0', '0', '0', '0']

Find S: Finding a Maximally Specific Hypothesis

Hypothesis
['sunny', 'warm', '?', 'strong', '?', '?']
```

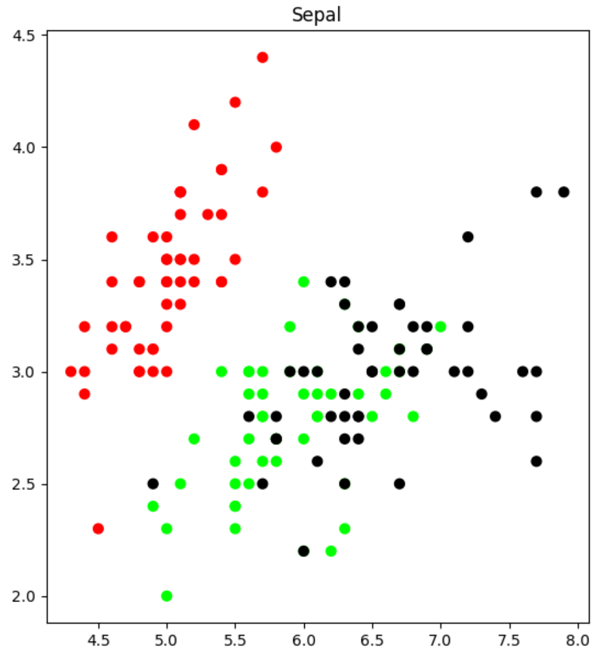
6. Apply <u>K-Means Clustering</u> algorithm to cluster data stored in a .CSV file.

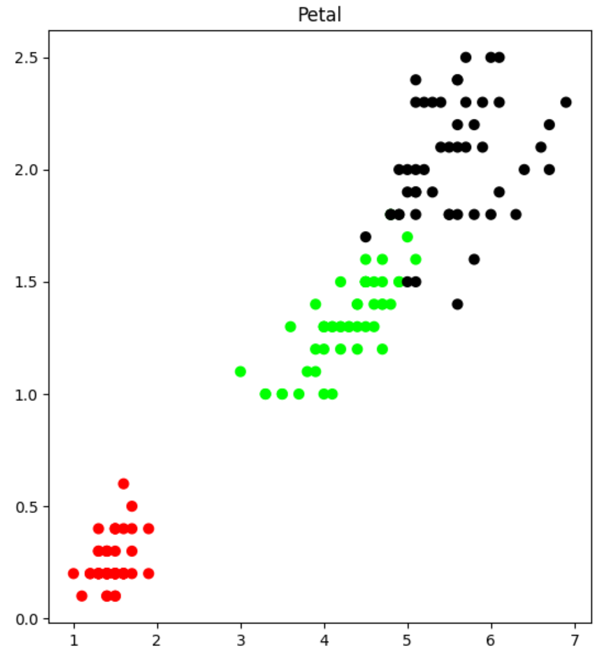
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

# import some data to play with
iris = datasets.load_iris()

# print("\n IRIS DATA :",iris.data);
# print("\n IRIS FEATURES :\n",iris.feature_names)
# print("\n IRIS TARGET :\n",iris.target)
# print("\n IRIS TARGET NAMES:\n",iris.target_names)

# Store the inputs as a Pandas Dataframe and set the column names
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

# Set the size of the plot
plt.figure(figsize=(14,7))

# Create a colormap
colormap = np.array(['red', 'lime', 'black'])

# Plot Sepal
plt.subplot(1, 2, 1)
plt.scatter(X.Sepal_Length,X.Sepal_Width, c=colormap[y.Targets], s=40)
plt.title('Sepal')

plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length,X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Petal')

# K Means Cluster
model = KMeans(n_clusters=3, n_init='auto')
model.fit(X)

model.labels_
```

```python
# View the results
# Set the size of the plot
plt.figure(figsize=(14,7))

# Create a colormap
colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')

# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
```

Output

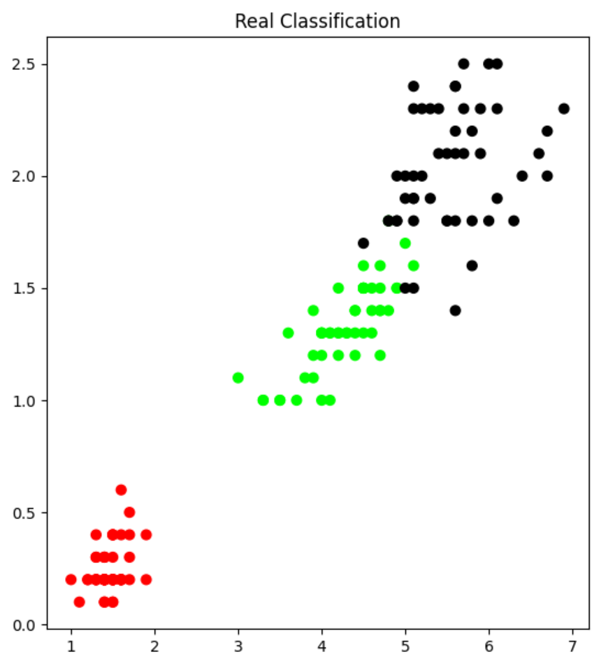
Before K Means Clustering

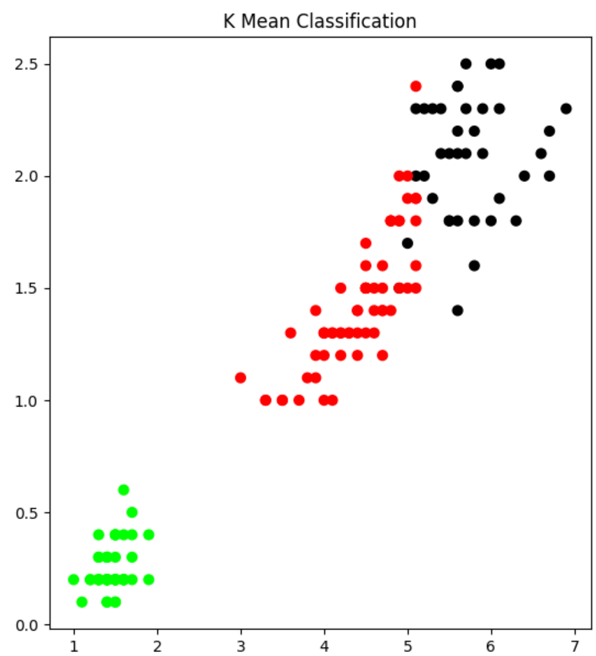




After K Means Clustering

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 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, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2,
       2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0])
```





7. Implement <u>k-Nearest Neighbour</u> algorithm to classify the iris data set. Print both correct and wrong predictions.

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np

df = pd.read_csv('iris.csv')
df.head()
df['variety'].unique()

x = df[['sepal.length','sepal.width','petal.length','petal.width']]
y = df['variety'].replace({'Setosa':1, 'Versicolor':2, 'Virginica':3 })
train_x, test_x , train_y, test_y=
train_test_split(x,y,test_size=0.1,random_state=3, shuffle=True)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(train_x,train_y)
pred_y = knn.predict(test_x)
print('The prediction is',pred_y)

print('Actual expected results are:', list(test_y))

print('Number of incorrect classifications are')
diff = np.array(pred_y) - np.array(test_y)
print(len(diff) - list(diff).count(0))
```

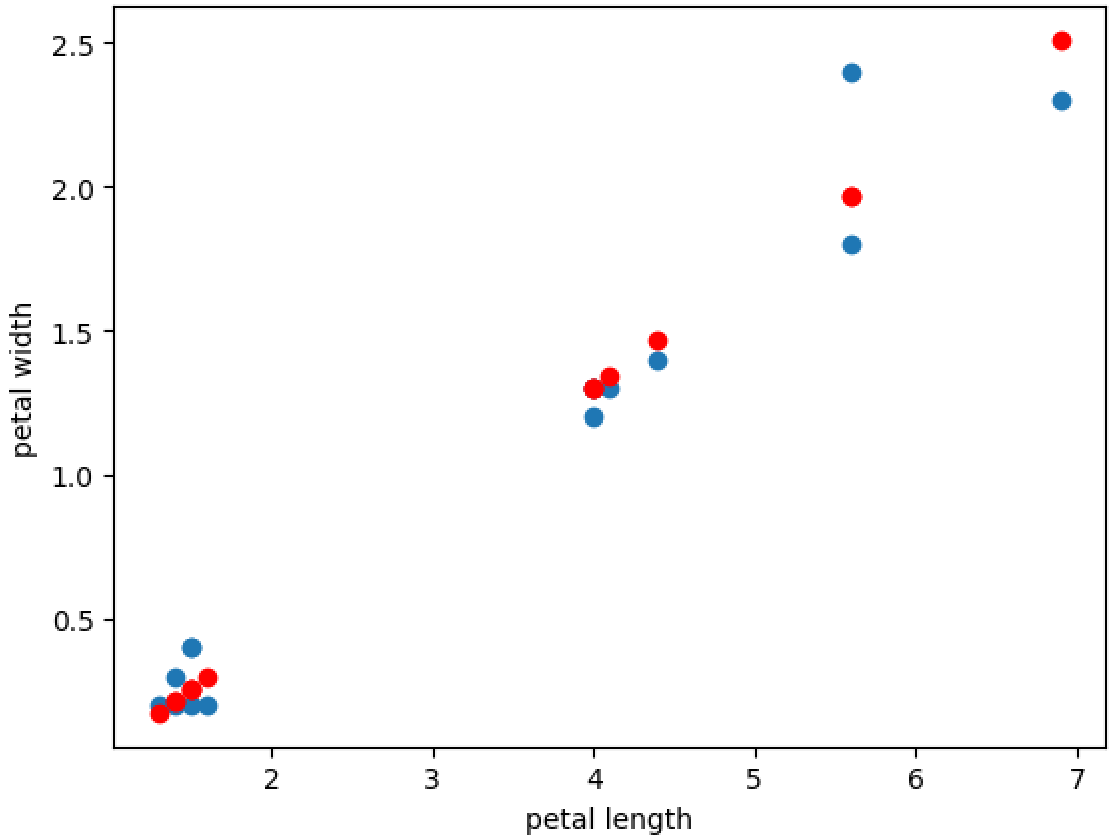
Output

```
array(['Setosa', 'Versicolor', 'Virginica'], dtype=object)

The prediction is [1 1 1 1 1 3 2 1 3 2 2 1 2 2 3]

Actual expected results are: [1, 1, 1, 1, 1, 3, 2, 1, 3, 2, 2, 1, 2, 2, 3]

Number of incorrect classifications are
0
```

8. Implement the parametric <u>Linear Regression</u> algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import linear_model

df = pd.read_csv('iris.csv')

newdf = df[['petal.length','petal.width']]
train_x, test_x , train_y, test_y=
train_test_split(newdf['petal.length'],newdf['petal.width'],test_size=0.1,random_
state=3, shuffle=True)
train_x = np.array(train_x).reshape(-1,1)
test_x = np.array(test_x).reshape(-1,1)
train_y = np.array(train_y).reshape(-1,1)
test_y = np.array(test_y).reshape(-1,1)

lr = linear_model.LinearRegression()
lr.fit(train_x,train_y)

import matplotlib.pyplot as plt
plt.scatter(test_x,test_y)
y_predict = lr.predict(test_x)
plt.scatter(test_x,y_predict,color = 'red')
plt.xlabel("petal length")
plt.ylabel("petal width")

from sklearn.metrics import mean_squared_error
sd = (10/4)**0.5
se = sd/(5**0.5)
print('Standard Error ',se )
print('Mean Square Error ', mean_squared_error(test_y,y_predict))
```

Output




```
Standard Error  0.7071067811865476
Mean Square Error  0.022523547640619812
```