ML LAB FILE

SUBMITTED TO – DEEPIKA VARSHNEY

SUBMITTED BY- VARUN KUMAR
2K19/IT/140

INDEX

LAB-1

In [1]:

```
a = int(input("Enter first  number:  "))
b = int(input("Enter second number: "))
sum = a+b
print(sum)
```

```
Enter first number: 25
Enter second number: 34
59
```

In [6]:

```
num = int(input("Enter a number: "))
if (num % 2) == 0:
    print("%d is Even number" % num)
else:
    print("%d is Odd number" % num)
```

```
Enter a number: 23
23 is Odd number
```

In [9]:

```
def changelistelitems(mylist):
    mylist = [13,2,4,56]
    print("values inside the function: ",mylist)
    return;

mylist = [17,29,15,3]
changelistelitems(mylist)
print("values outside the function: ",mylist)
```

```
values  inside the  function:   [13, 2, 4, 56]
values outside  the  function:   [17, 29, 15, 3]
```

In [17]:

```
my_dict = {'name': 'Aman', 'age': 26,'Job': 'Architect'}

print("name:",my_dict['name'])
print("age:",my_dict['age'])
print("job:",my_dict['Job'])
```

```
name: Aman
age: 26
job: Architect
```

In [16]:

```
first_str = input("enter first string : ")
second_str = input("enter the second string : ")
str3 = first_str + second_str
print(str3)

print(str3.lower())
print(str3.upper())

print(type(str3))
fruit = 'mango'
st = 'go'
pos = fruit.find(st)
print("position of "+st+" in",fruit,"is: ",pos)
fullname = 'Tanmay kumar'
str4 = fullname.replace('kumar','arora')
print(str4)
```

```
enter first string :  Hello
enter the second string : World
HelloWorld
helloworld
HELLOWORLD
<class 'str'>
position of go in mango is:  3
Tanmay arora
```

LAB - 2

In [1]:

```
import numpy as np
```

In [2]:

```
#Create 1-D , 2-D  array
A=np.array([1,2,3,4,5,6])   # 1D array
AA=np.array([[1,2,3],[4,5,6]])   # 2D array
AAA=np.array([[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]],[[1,2,3],[4,5,6]]]) #3D array

print("The Dimensions of array A are : ", A.ndim)
print("The Dimensions of array AA are : ", AA.ndim)
print("The Dimensions of array AAA are : ", AAA.ndim)
print()
print("The Shape of A is : ", A.shape)
print("The Shape of AA is : ", AA.shape)
print("The Shape of AAA is : ", AAA.shape)
print()
print("The datatype of A is : ", A.dtype)
print()
```

```
The Dimensions of array  A are :  1
The Dimensions of array AA are :  2
The Dimensions of array AAA are :  3

The  Shape  of A is :  (6,)
The Shape of AA  is :  (2, 3)
The Shape of AAA is :  (3, 2, 3)

The  datatype of A  is :  int32
```

In [3]:

```
print(AA)

#RESHAPE THE ARRAY
BB=AA.reshape((3,2))
print(BB)
print()
```

```
[[1 2 3]
 [4 5 6]]
[[1 2]
 [3 4]
 [5 6]]
```

In [4]:

```
#OPERATIONS ON NUMPY
NEW_A = np.array([6,5,4,3,2,1])
NEW_A = NEW_A * 3
print(NEW_A)

print('   ------------ ')

print([1,2,3]*3)

print(" -----")

AA[0][1]=10
print(AA)
print()
print(BB)
```

```
[18 15 12  9  6  3]
---------------------
[1, 2, 3, 1, 2, 3, 1, 2, 3]
-------
[[ 1 10  3]
 [ 4  5  6]]

[[ 1 10]
 [ 3  4]
 [ 5  6]]
```

In [5]:

```
C = BB.copy()
print(C)
print("------ ")

# IN COPY A DEEP COPY IS CREATED SO CHANGING BB not reflect in the BB
C[0][0]=33
print(C)
print()
print(BB)
```

```
[[ 1 10]
 [ 3  4]
 [ 5  6]]
-------
[[33 10]
 [ 3  4]
 [ 5  6]]

[[ 1 10]
 [ 3  4]
 [ 5  6]]
```

In [6]:

```
#printing SPECIFIC INDEX
print(A[np.array([1,2])])
print()

print("THE ODD ELEMENTS IN A are :", A[A%2!=0])
```

```
[2 3]

THE ODD ELEMENTS IN A are : [1 3 5]
```

In [7]:

```
# CLIP IN NUMPY
D = A.copy()
print(D)
# ALL less than 2 become 2 all greater than 4 become  4
print(D.clip(2,4))
```

```
[1 2 3 4 5 6]
[2 2 3 4 4 4]
```

In [14]:

```
a=np.array([[1,2,3],[4,5,6],[7,8,9]],dtype="f")
b=np.array([[1,2,3],[4,5,6]])
print(b.resize(3,3)); print(b)
print("a matrix \n", a);print("*********")
print("b matrix \n", b);print("*********")
print(a+b); print("*********")
print(a-b); print("*********")
print(a*b); print("*********") #Here values mul takes place
print(a+2);  print("*********")
print(a-2); print("*********")
print(a*5); print("*********")
print(a/2); print("*********")
print(a.dot(b)); print("*********") #here matrix mul takes place
print(np.average(a)); print("*********")
print(np.std(a)); print("*********")
print(np.var(a)); print("*********")          # variance
print(np.linalg.det(a)); print("*********")    # determinent
print(np.linalg.det(b))
```

```
None
[[1 2 3]
 [4 5 6]
 [0 0 0]]
a matrix
 [[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
*********
b matrix
 [[1 2 3]
 [4 5 6]
 [0 0 0]]
*********
[[ 2.  4.  6.]
 [ 8. 10. 12.]
 [ 7.  8.  9.]]
*********
[[0. 0. 0.]
 [0. 0. 0.]
 [7. 8. 9.]]
*********
[[ 1.  4.  9.]
 [16. 25. 36.]
 [ 0.  0.  0.]]
*********
[[ 3.  4.  5.]
 [ 6.  7.  8.]
 [ 9. 10. 11.]]
*********
[[-1.  0.  1.]
 [ 2.  3.  4.]
 [ 5.  6.  7.]]
*********
[[ 5. 10. 15.]
 [20. 25. 30.]
 [35. 40. 45.]]
*********
[[0.5 1.  1.5]
```

```
 [2.   2.5 3. ]
 [3.5 4.   4.5]]
*********
[[ 9. 12. 15.]
 [24.  33. 42.]
 [39. 54. 69.]]
*********
5.0
*********
2.5819888
*********
6.6666665
*********
-9.516197e-16
*********
0.0
```

In [15]:

```
a = np.array([1, 2, 3, 4])
b = np.array([5, 2, 2, 4])
c = np.array([1, 2, 3, 4])
print(a == b)
print(a > b)

#array-wise comparisions
print(np.array_equal(a,b))
print(np.array_equal(a,c))
```

```
[False  True  False   True]
[False  False  True  False]
False
True
```

In [16]:

```
k=np.random.randint(10,15,size=(4,5), dtype="int64"); print(k)
```

```
[[12  10  13  14  14]
 [13  13  10  14  12]
 [14  11  13  13  14]
 [12  10  10  14  12]]
```

In [17]:

```
g=np.identity(3,dtype="float64"); print(g)
```

```
[[1.  0.  0.]
 [0.  1.  0.]
 [0.  0.  1.]]
```

LAB-3

In [ ]:

```
# importing the dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:

```
dataset = pd.read_csv('Salary_Data.csv')
dataset.head()
```

Out[18]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

In [ ]:

```
# data preprocessing
X = dataset.iloc[:,:-1].values #independent variable array
y = dataset.iloc[:,1].values #dependent variable vector
```

In [ ]:

```
# splitting the dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=1/3,random_state=0)
```

In [ ]:

```
# fitting the regression model
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train,y_train) #actually produces the linear eqn for the data
```

Out[21]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

In [ ]:

```
# predicting the test set results
y_pred = regressor.predict(X_test)
y_pred

y_test
print('Coefficients: \n', regressor.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(y_test, y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y_test, y_pred))
```

```
Coefficients:
 [9345.94244312]
Mean squared error: 21026037.33
Coefficient of determination: 0.97
```

In [ ]:

```
#visualizing the results
#plot for the TRAIN

plt.scatter(X_train, y_train, color='red') # plotting the observation line
plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regression line
plt.title("Salary vs Experience (Training set)") # stating the title of the graph

plt.xlabel("Years of experience") # adding the name of x-axis
plt.ylabel("Salaries") # adding the name of y-axis
plt.show() # specifies end of graph
```

```python
#plot for the TEST

plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regression line
plt.title("Salary vs Experience (Testing set)")

plt.xlabel("Years of experience")
plt.ylabel("Salaries")
plt.show()
```

LAB-4

In [53]:

```
import cv2
from google.colab.patches import cv2_imshow
import numpy as np
import matplotlib.pyplot as plt
```

In [54]:

```
#read image
image = cv2.imread("nature.jpg")
image = cv2.resize(image, (0, 0), None, .25, .25)
cv2_imshow(image)
cv2.waitKey(0) #waitkey

grey = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
grey_3_channel = cv2.cvtColor(grey, cv2.COLOR_GRAY2BGR)
cv2_imshow(grey_3_channel)
cv2.destroyAllWindows()
```

In [55]:

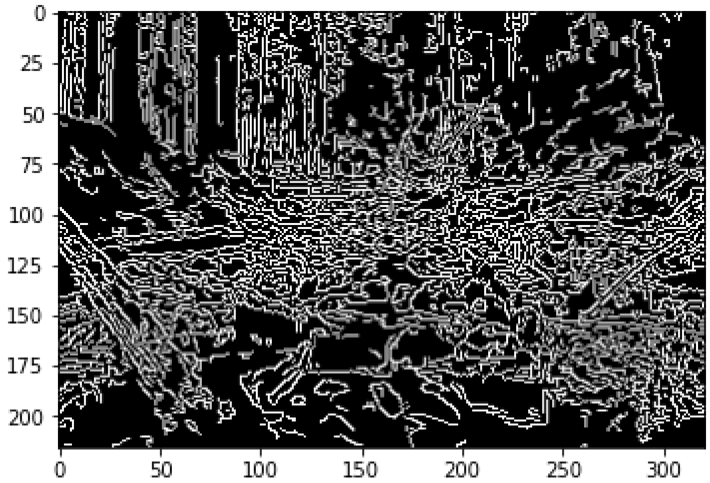
```
#blur and change color
img_0 = cv2.blur(image, ksize = (7, 7))
img_1 = cv2.medianBlur(image, 7)
img_2 = cv2.bilateralFilter(image, 7, 75, 75)
img_3 = cv2.cvtColor(image,cv2.COLOR_RGB2BGR)

img_stack = np.hstack((image, img_0))
numpy_horizontal = np.hstack((img_2, img_1))
numpy_horizontal_concat = np.concatenate((grey_3_channel,img_3), axis=1)

cv2_imshow(img_stack)
cv2_imshow(numpy_horizontal)
cv2_imshow(numpy_horizontal_concat)
```

In [56]:

```
#edge detection
edgeimg = cv2.Canny(img , 150,250)
plt.imshow(cv2.cvtColor(edgeimg , cv2.COLOR_BGR2RGB))
```

Out[56]:

<matplotlib.image.AxesImage at 0x7f875776f510>



In [57]:

```
#save  img
from google.colab import files
status = cv2.imwrite('savedfig1.png',image)
files.download('savedfig1.png')
print(status)
```

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

True

LAB-5

In [52]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [53]:

```
def mean_squared_error(y_true, y_predicted):

    # Calculating the loss or cost
    cost = np.sum((y_true-y_predicted)**2) / len(y_true)
    return cost
```

In [54]:

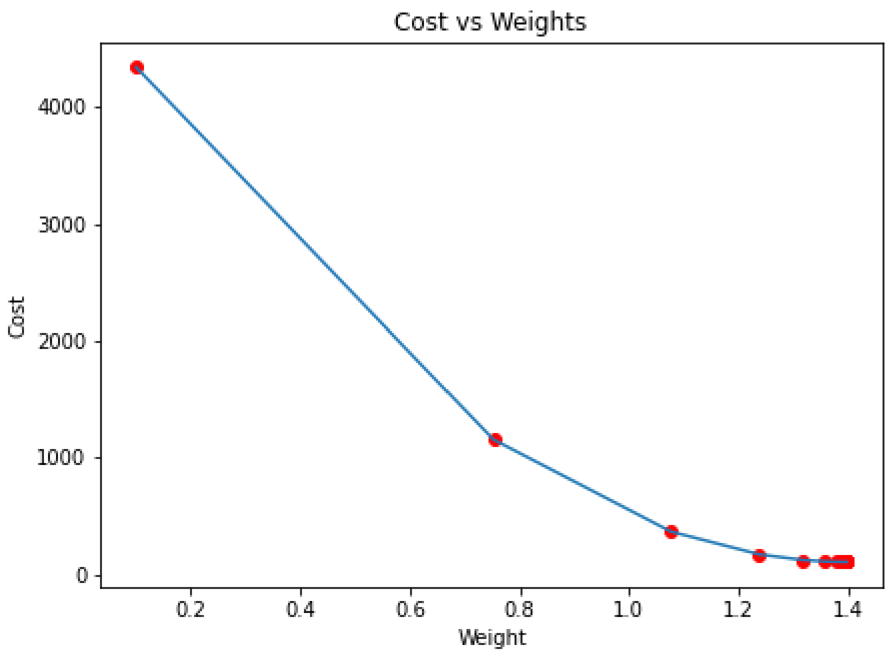
```
# Gradient Descent Function
def gradient_descent(x, y,stopping_threshold = 1e-6):

    # Initializing weight, bias, learning rate and iterations
    current_weight = 0.1
    current_bias = 0.01
    n = float(len(x))
    iterations  = 1000
    learning_rate = 0.0001
    costs = []
    weights = []
    previous_cost = None

    # Estimation of optimal parameters
    for i in range(iterations):

        # Making predictions
        y_predicted = (current_weight * x) + current_bias

        # Calculationg the current cost
        current_cost = mean_squared_error(y, y_predicted)

        # If the change in cost is less than or equal to
        # stopping_threshold we stop the gradient  descent
        if previous_cost and abs(previous_cost-current_cost)<=stopping_threshold:
            break

        previous_cost = current_cost

        costs.append(current_cost)
        weights.append(current_weight)

        # Calculating the gradients
        weight_derivative  =  -(2/n)  *  sum(x  *  (y-y_predicted))
        bias_derivative  =  -(2/n)  *  sum(y-y_predicted)

        # Updating weights and bias
        current_weight = current_weight - (learning_rate * weight_derivative)
        current_bias = current_bias - (learning_rate * bias_derivative)

        # Printing the parameters for each 1000th iteration
        print(f"Iteration {i+1}: Cost {current_cost}, Weight {current_weight}, Bias {curren


    # Visualizing the weights and cost at for all iterations
    plt.figure(figsize = (7,5))
    plt.plot(weights, costs)
    plt.scatter(weights, costs, marker='o', color='red')
    plt.title("Cost vs Weights")
    plt.ylabel("Cost")
    plt.xlabel("Weight")
    plt.show()

    return current_weight, current_bias
```

In [55]:

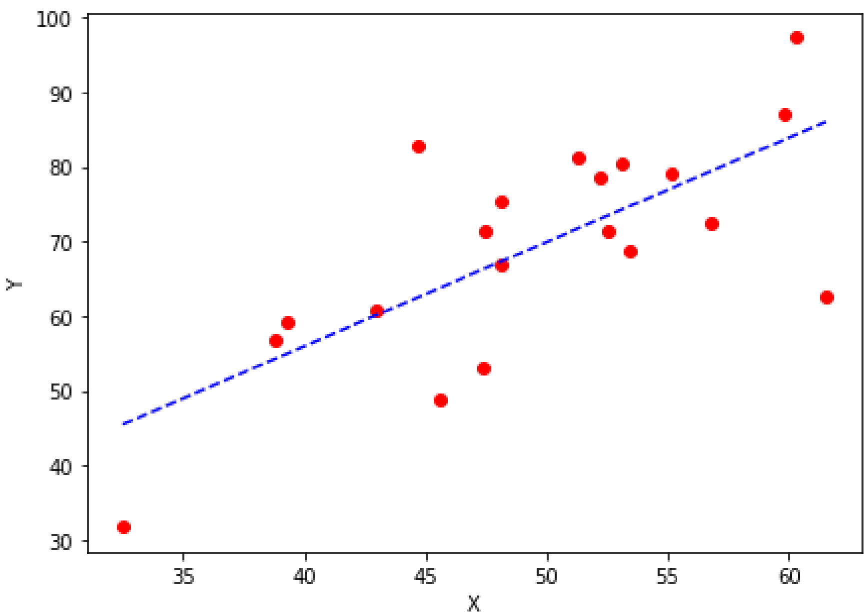
```
def main():

    X = np.array([32.50234527, 53.42680403, 61.53035803, 47.47563963, 59.81320787,
           55.14218841,  52.21179669, 39.29956669, 48.10504169, 52.55001444,
           47.41973014,  51.35163488, 48.1640495 , 56.76847072, 53.12720806,
           42.95588857,  44.68719623, 60.29732685, 45.61864377, 38.81681754])
    Y = np.array([31.70700585, 68.77759598, 62.5623823 , 71.54663223, 87.23092513,
           79.21151327,  78.64121305, 59.17148932, 75.3312423 , 71.30087989,
           53.16567715,  81.37824676, 67.00892325, 72.39287043, 80.43619216,
           60.72360244,  82.89250373, 97.37989686, 48.84715332, 56.87721319])

    # Estimating weight and bias using gradient descent
    estimated_weight, estimated_bias = gradient_descent(X, Y)
    print(f"Estimated Weight: {estimated_weight}\nEstimated Bias: {estimated_bias}")

    # Making predictions using estimated parameters
    Y_pred = estimated_weight*X + estimated_bias

    # Plotting the regression line
    plt.figure(figsize = (7,5))
    plt.scatter(X, Y, marker='o', color='red')
    plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='blue',markerfacecolor='re
             markersize=8,linestyle='dashed')
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.show()

if name ==" main ":
    main()
```

Iteration 1: Cost 4341.581076655404, Weight 0.7523677867747861, Bias 0.02
2872567657090004
Iteration 2: Cost 1153.8034948189297, Weight 1.0769142488207037, Bias 0.0
29275874245869263
Iteration 3: Cost 364.84034437488197, Weight 1.238372880710946, Bias 0.03
246078816219145
Iteration 4: Cost 169.57488534877945, Weight 1.3186969483171411, Bias 0.0
3404458391752937
Iteration 5: Cost 121.24740709445219, Weight 1.3586573817068328, Bias 0.0
348318394052224
Iteration 6: Cost 109.28653512036976, Weight 1.378537310794286, Bias 0.03
522282407974981
Iteration 7: Cost 106.32626357237136, Weight 1.3884273898382435, Bias 0.0
35416667991185626
Iteration 8: Cost 105.59360730982864, Weight 1.393347618404841, Bias 0.03
551243634772583
Iteration 9: Cost 105.41227758385966, Weight 1.3957953960677647, Bias 0.0
35559413099589196
Iteration 10: Cost 105.3673991346956, Weight 1.3970131541357187, Bias 0.0
35582116519067265
Iteration 11: Cost 105.35629187140164, Weight 1.3976189897578792, Bias 0.
0355927442013912
Iteration 12: Cost 105.35354285159053, Weight 1.3979204000837004, Bias 0.
035597364328187095
Iteration 13: Cost 105.35286246609958, Weight 1.398070361869423, Bias 0.0
35598995759119456
Iteration 14: Cost 105.35269405994492, Weight 1.3981449795211902, Bias 0.

03559914034667647
Iteration 15: Cost 105.35265236678075, Weight 1.3981821142198554, Bias 0.
03559854524744932
Iteration 16: Cost 105.35264203461277, Weight 1.3982006015242023, Bias 0.
03559758216436901
Iteration 17: Cost 105.35263946417538, Weight 1.398209811936331, Bias 0.0
35596436015925874



Estimated Weight: 1.398209811936331
Estimated Bias: 0.035596436015925874



LAB-6

In [2]:

```
import pandas as pd
# load dataset
ds = pd.read_csv("diabetes.csv")
```

In [3]:

```
ds.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

In [4]:

```
cols = ['Pregnancies', 'Glucose',  'BloodPressure',  'Insulin','BMI','DiabetesPedigreeFunctio
X = ds[cols] # Features
y = ds.Outcome # Target variable
```

In [5]:

```
import sklearn
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=2)
```

In [6]:

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=400)

# fit the model with data
model.fit(X_train,y_train)

y_pred=model.predict(X_test)
```

In [7]:

```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[7]:

```
array([[138,  17],
       [ 40,  36]])
```

In [8]:

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy:  0.7532467532467533
Precision: 0.6792452830188679
Recall: 0.47368421052631576
```

LAB-7

In [1]:

```
from numpy.lib.npyio import load
from sklearn.datasets import load_iris
```

In [2]:

```
iris=load_iris()
X=iris.data
y=iris.target
```

In [3]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [4]:

```
from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
gnb.fit(X_train,y_train)
y_pred=gnb.predict(X_test)
```

In [5]:

```
from sklearn import metrics
print("ACCURACY=",metrics.accuracy_score(y_test,y_pred)*100)
```

```
ACCURACY=  96.0
```

LAB-8

In [43]:

```
import numpy as np
import pandas as  pd
from sklearn import metrics
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

In [44]:

```
df = datasets.load_breast_cancer()

X =  df.data
Y = df.target

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = 1
```

In [46]:

```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [47]:

```
svm = SVC(kernel = 'linear')
svm.fit(x_train , y_train)
```

Out[47]:

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None,coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True,
    tol=0.001, verbose=False)
```

In [48]:

```
y_pred = svm.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy:  0.9883040935672515
Precision: 0.9818181818181818
Recall: 1.0
```

LAB-9

In [65]:

```
import pandas as  pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn import tree
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
import graphviz
```

In [66]:

```
iris=load_iris()

print(iris.feature_names)
print(iris.target_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
['setosa' 'versicolor' 'virginica']
```

In [67]:

```
#Spilitting the dataset
removed =[0,50,100]
new_target = np.delete(iris.target,removed)
new_data = np.delete(iris.data,removed, axis=0)
```

In [68]:

```
#train classifier
clf = tree.DecisionTreeClassifier() # defining decision tree classifier
clf = clf.fit(new_data,new_target) # train data on new data and new target
prediction = clf.predict(iris.data[removed]) #  assign removed data as input

print("Original Labels",iris.target[removed])
print("Labels Predicted",prediction)
```

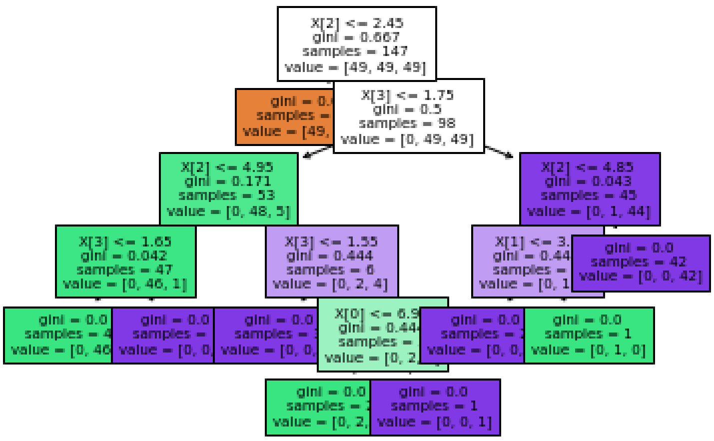
```
Original Labels [0 1 2]
Labels Predicted [0 1 2]
```

In [69]:

```python
tree.plot_tree(clf,filled=True,fontsize=7)
```

Out[69]:

```
[Text(167.4, 199.32, 'X[2] <= 2.45\ngini = 0.667\nsamples = 147\nvalue = [4
9, 49, 49]'),
 Text(141.64615384615385, 163.07999999999998, 'gini = 0.0\nsamples = 49\nval
ue = [49, 0, 0]'),
 Text(193.15384615384616, 163.07999999999998, 'X[3] <= 1.75\ngini = 0.5\nsam
ples = 98\nvalue = [0, 49, 49]'),
 Text(103.01538461538462, 126.83999999999999, 'X[2] <= 4.95\ngini = 0.171\ns
amples = 53\nvalue = [0, 48, 5]'),
 Text(51.50769230769231, 90.6, 'X[3] <= 1.65\ngini = 0.042\nsamples = 47\nva
lue = [0, 46, 1]'),
 Text(25.753846153846155, 54.359999999999985, 'gini = 0.0\nsamples = 46\nval
ue = [0, 46, 0]'),
 Text(77.26153846153846, 54.359999999999985, 'gini = 0.0\nsamples = 1\nvalue
= [0, 0, 1]'),
 Text(154.52307692307693, 90.6, 'X[3] <= 1.55\ngini = 0.444\nsamples = 6\nva
lue = [0, 2, 4]'),
 Text(128.76923076923077, 54.359999999999985, 'gini = 0.0\nsamples = 3\nvalu
e = [0, 0, 3]'),
 Text(180.27692307692308, 54.359999999999985, 'X[0] <= 6.95\ngini = 0.444\ns
amples = 3\nvalue = [0, 2, 1]'),
 Text(154.52307692307693, 18.119999999999976, 'gini = 0.0\nsamples = 2\nvalu
e = [0, 2, 0]'),
 Text(206.03076923076924, 18.119999999999976, 'gini = 0.0\nsamples = 1\nvalu
e = [0, 0, 1]'),
 Text(283.2923076923077, 126.83999999999999, 'X[2] <= 4.85\ngini = 0.043\nsa
mples = 45\nvalue = [0, 1, 44]'),
 Text(257.53846153846155, 90.6, 'X[1] <= 3.1\ngini = 0.444\nsamples = 3\nval
ue = [0, 1, 2]'),
 Text(231.7846153846154, 54.359999999999985, 'gini = 0.0\nsamples = 2\nvalue
= [0, 0, 2]'),
 Text(283.2923076923077, 54.359999999999985, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1, 0]'),
 Text(309.04615384615386, 90.6, 'gini = 0.0\nsamples = 42\nvalue = [0, 0, 4
2]')]
```



In [70]:

```
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,filled=True, rounded=True,
                special_characters=True,feature_names=iris.feature_names,
                class_names=iris.target_names)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('iris.png')
Image(graph.create_png())
```

Out[70]: