Iris_Flower_classification_Machine_Learning_Data_Science

project by Anand Bharti (IIT Bhubaneswar).

In [1]:

```
import pandas as pd
from sklearn.datasets import load_iris
```

Data Importing

In [2]:

```
iris = load_iris()
```

In [3]:

```
feature_name=iris.feature_names
```

In [4]:

```
species=iris.target_names
```

Data Cleaning and Data Formatting

In [5]:

```
df=pd.DataFrame(iris.data,columns=iris.feature_names)
df_i=df
df.head()
```

Out[5]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [6]:

```
df['target']=iris.target

df.head()
```

Out[6]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [7]:

```
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])
df.head()
```

Out[7]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

In [8]:

```
df[df.target==1].head()
```

Out[8]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

In [9]:

```
df[df.target==2].head()
```

Out[9]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica

DATA VISUALIZATION

In [10]:

```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:

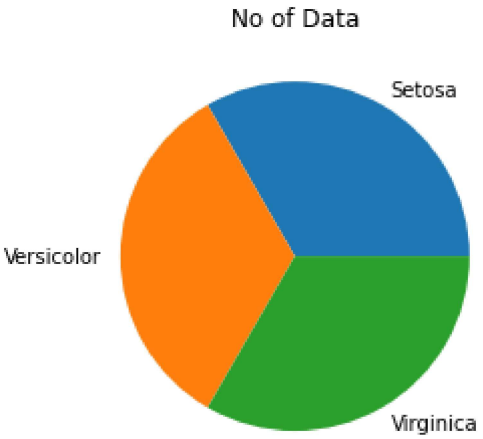
```

```

In [11]:

```
length = [len(df[df.target==0]),len(df[df.target==1]),len(df[df.target==2])]
plt.title("No of Data")
plt.pie(length , labels=["Setosa", "Versicolor", "Virginica", ])
```

Out[11]:

```
([<matplotlib.patches.Wedge at 0x7faa58c5cf90>,
  <matplotlib.patches.Wedge at 0x7faa58c72690>,
  <matplotlib.patches.Wedge at 0x7faa58c72bd0>],
 [Text(0.5499999702695115, 0.9526279613277875, 'Setosa'),
  Text(-1.0999999999999954, -1.0298943258065002e-07, 'Versicolor'),
  Text(0.5500001486524352, -0.9526278583383436, 'Virginica')])
```



In [12]:

```
df_setosa= df[:50]
df_versicolor = df[50:100]
df_virginica = df[100:]
```

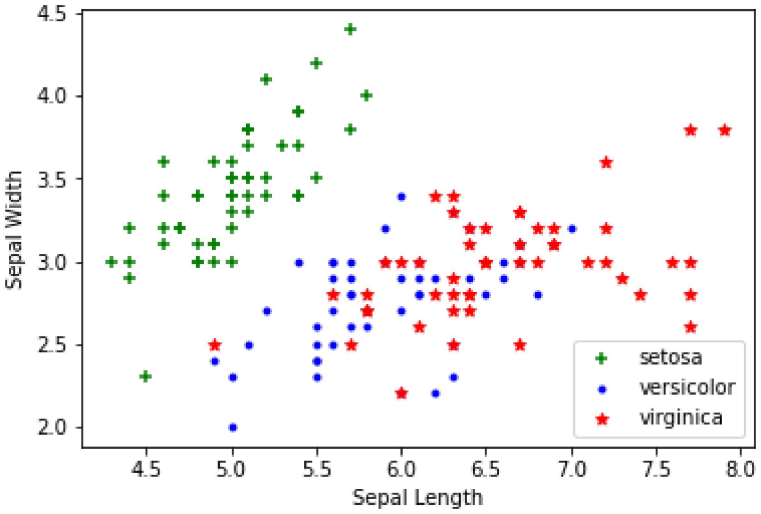
Scatter Plot of with different parameter

In [13]:

```
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df_setosa['sepal length (cm)'], df_setosa['sepal width (cm)'],color="green",marker='+')
plt.scatter(df_versicolor['sepal length (cm)'], df_versicolor['sepal width (cm)'],color="blue",marker='.')
plt.scatter(df_virginica['sepal length (cm)'],df_virginica['sepal width (cm)'],color="red",marker='*')
plt.legend(["setosa", "versicolor", "virginica"], loc ="lower right")
```

Out[13]:

<matplotlib.legend.Legend at 0x7faa58b59510>

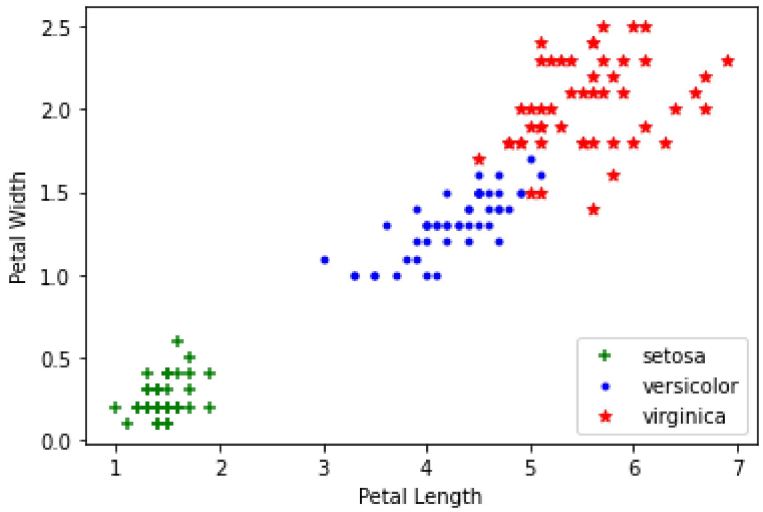


In [14]:

```
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df_setosa['petal length (cm)'], df_setosa['petal width (cm)'],color="green",marker='+')
plt.scatter(df_versicolor['petal length (cm)'], df_versicolor['petal width (cm)'],color="blue",marker='.')
plt.scatter(df_virginica['petal length (cm)'], df_virginica['petal width (cm)'],color="red",marker='*')
plt.legend(["setosa","versicolor", "virginica"], loc ="lower right")
```

Out[14]:

<matplotlib.legend.Legend at 0x7faa58a442d0>

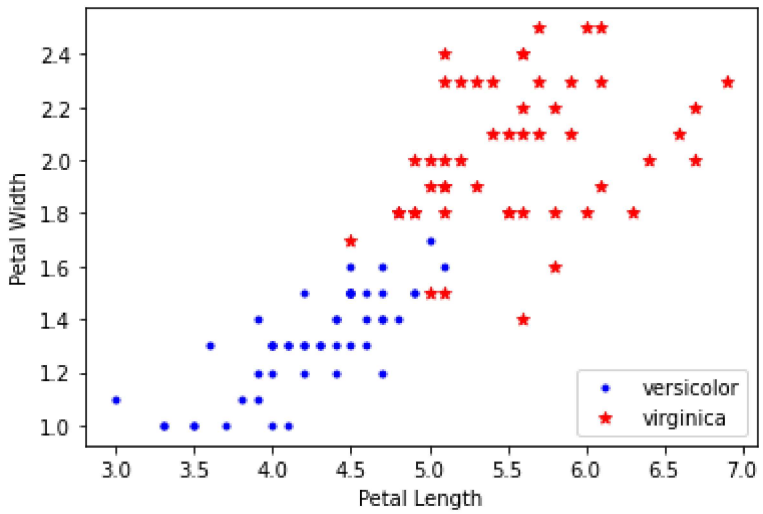


In [15]:

```
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df_versicolor['petal length (cm)'], df_versicolor['petal width (cm)'],color="blue",marker='.')
plt.scatter(df_virginica['petal length (cm)'], df_virginica['petal width (cm)'],color="red",marker='*')
plt.legend(["versicolor", "virginica"], loc ="lower right")
```

Out[15]:

<matplotlib.legend.Legend at 0x7faa58c026d0>



In [ ]:

In [ ]:

In [ ]:

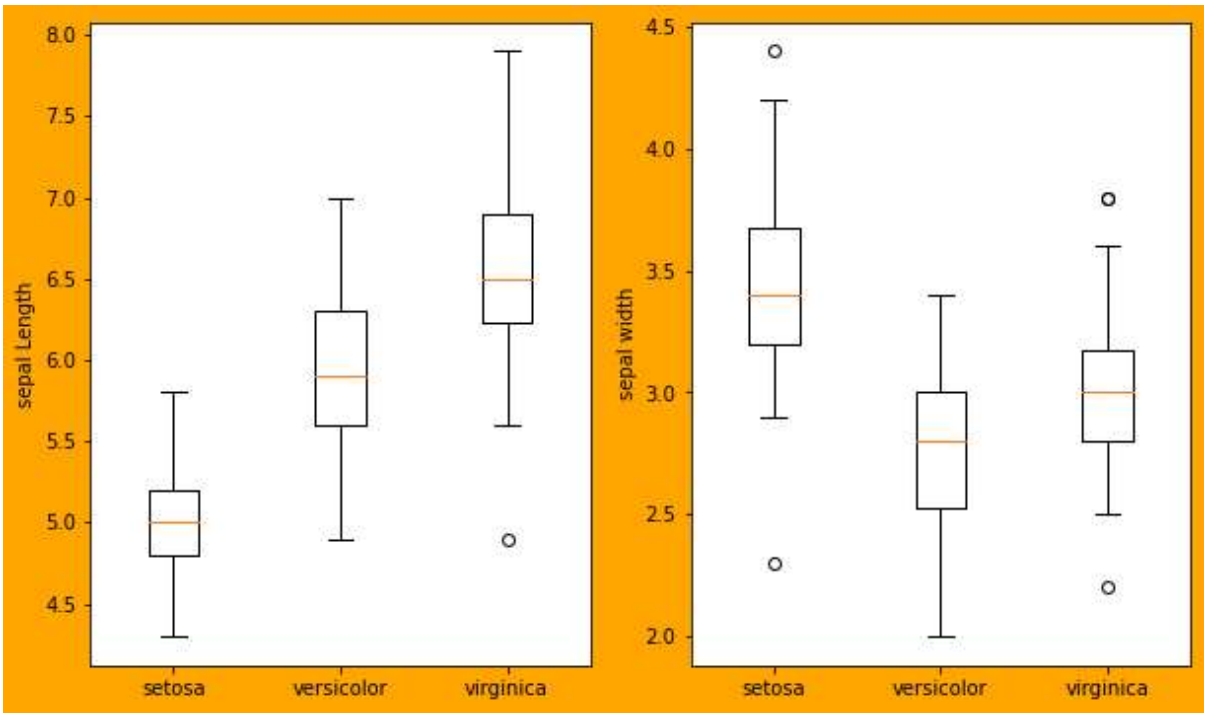
Box Plot of with different parameter

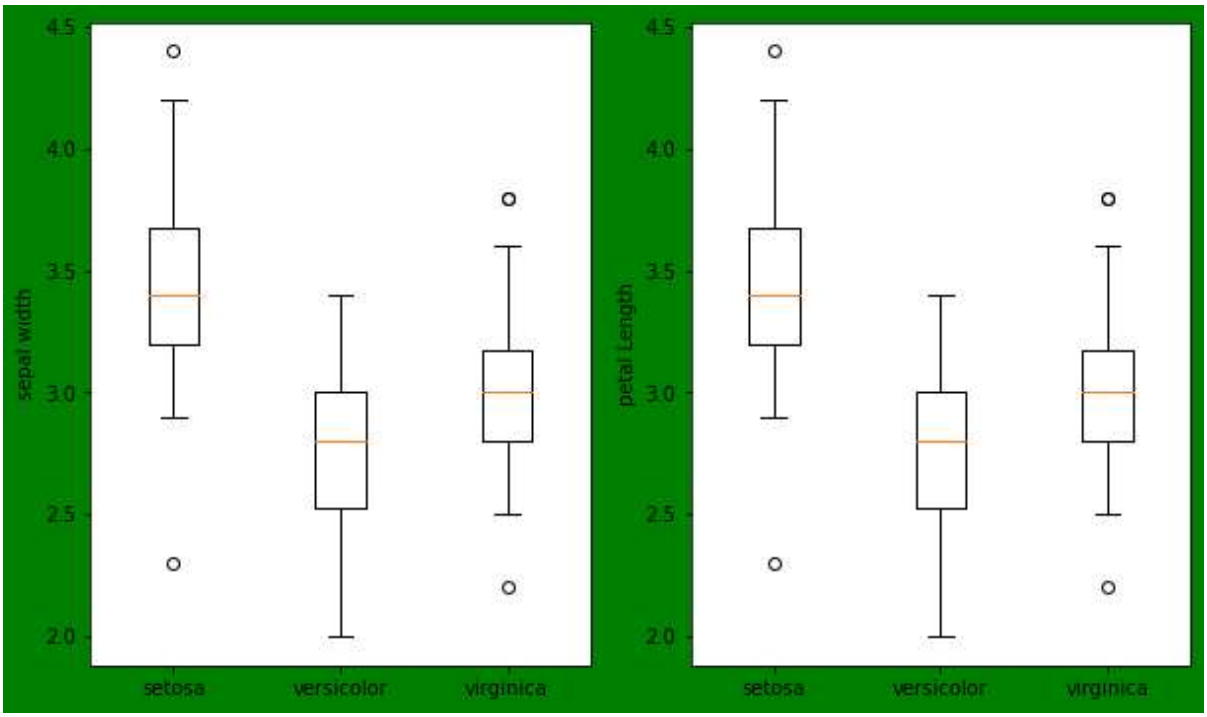
In [16]:

```
df_setosa_s = df_setosa['sepal length (cm)']
df_versicolor_s= df_versicolor['sepal length (cm)']
df_virginica_s = df_virginica['sepal length (cm)']
plt.figure(1,figsize=[10.0,6.0],facecolor="orange")
plt.subplot(1,2,1)
plt.ylabel('sepal Length')
plt.boxplot([df_setosa_s,df_versicolor_s,df_virginica_s], labels=['setosa', 'versicolor','virginica'])

#----------------------------------------
df_setosa_s = df_setosa['sepal width (cm)']
df_versicolor_s= df_versicolor['sepal width (cm)']
df_virginica_s = df_virginica['sepal width (cm)']
plt.subplot(1,2,2)
plt.ylabel('sepal width')
plt.boxplot([df_setosa_s,df_versicolor_s,df_virginica_s], labels=['setosa', 'versicolor','virginica'])

#----------------------------------------

df_setosa_s = df_setosa['sepal width (cm)']
df_versicolor_s= df_versicolor['sepal width (cm)']
df_virginica_s = df_virginica['sepal width (cm)']
plt.figure(2,figsize=[10.0,6.0],facecolor="green")
plt.subplot(1,2,1)
plt.ylabel('sepal width')
plt.boxplot([df_setosa_s,df_versicolor_s,df_virginica_s], labels=['setosa', 'versicolor','virginica'])

#----------------------------------------

df_setosa_p = df_setosa['petal length (cm)']
df_versicolor_p= df_versicolor['petal length (cm)']
df_virginica_p = df_virginica['petal length (cm)']
plt.subplot(1,2,2)
plt.ylabel('petal Length')
plt.boxplot([df_setosa_s,df_versicolor_s,df_virginica_s], labels=['setosa', 'versicolor','virginica'])
```

Out[16]:

{'whiskers': [<matplotlib.lines.Line2D at 0x7faa587bb7d0>,
<matplotlib.lines.Line2D at 0x7faa587bbc10>,
<matplotlib.lines.Line2D at 0x7faa58753650>,
<matplotlib.lines.Line2D at 0x7faa58753a50>,
<matplotlib.lines.Line2D at 0x7faa58768450>,
<matplotlib.lines.Line2D at 0x7faa587688d0>],
'caps': [<matplotlib.lines.Line2D at 0x7faa58746090>,
<matplotlib.lines.Line2D at 0x7faa587464d0>,
<matplotlib.lines.Line2D at 0x7faa58753e90>,
<matplotlib.lines.Line2D at 0x7faa5875f310>,
<matplotlib.lines.Line2D at 0x7faa58768d10>,
<matplotlib.lines.Line2D at 0x7faa587741d0>],
'boxes': [<matplotlib.lines.Line2D at 0x7faa587bb510>,
<matplotlib.lines.Line2D at 0x7faa58753210>,
<matplotlib.lines.Line2D at 0x7faa5875ff90>],
'medians': [<matplotlib.lines.Line2D at 0x7faa58746950>,
<matplotlib.lines.Line2D at 0x7faa5875f750>,
<matplotlib.lines.Line2D at 0x7faa58774610>],
'fliers': [<matplotlib.lines.Line2D at 0x7faa58746d90>,
<matplotlib.lines.Line2D at 0x7faa5875fb90>,
<matplotlib.lines.Line2D at 0x7faa58774a50>],
'means': []}





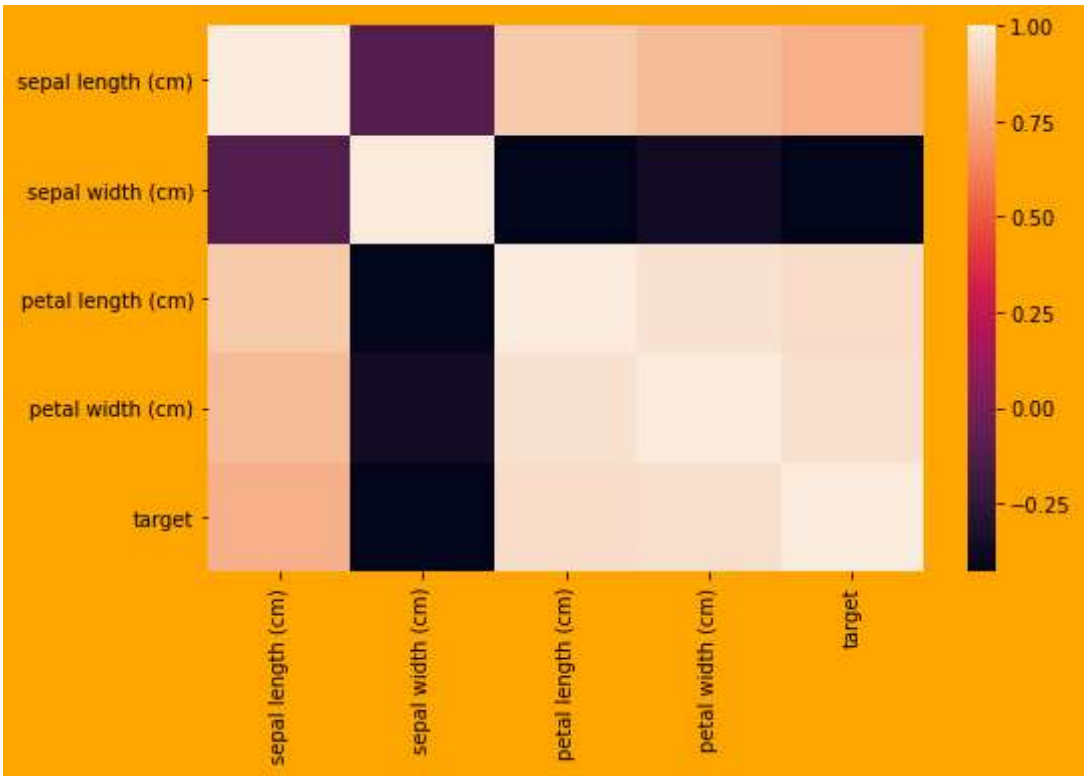
Correlation heat plot

In [17]:

```
import seaborn as sns
plt.figure(1,figsize=[8.0,5.0],facecolor="orange")
corr = df.corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
```

Out[17]:

<AxesSubplot:>



In [18]:

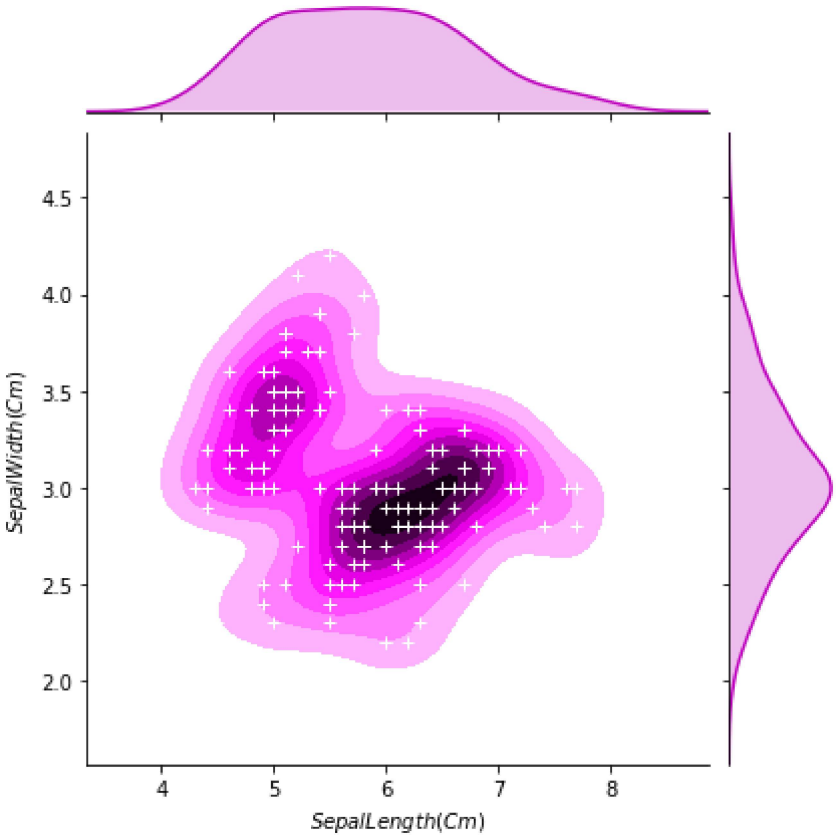
```
df.head()
```

Out[18]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

In [19]:

```
iris1 = df.drop(columns = ['target','flower_name'])
g = sns.jointplot(x="sepal length (cm)", y="sepal width (cm)", data=iris1, kind="kde", color="m")
g.plot_joint(plt.scatter, c="w", s=40, linewidth=1, marker="+")
g.ax_joint.collections[0].set_alpha(0)
g.set_axis_labels("$SepalLength(Cm)$", "$SepalWidth(Cm)$")
plt.show()
```

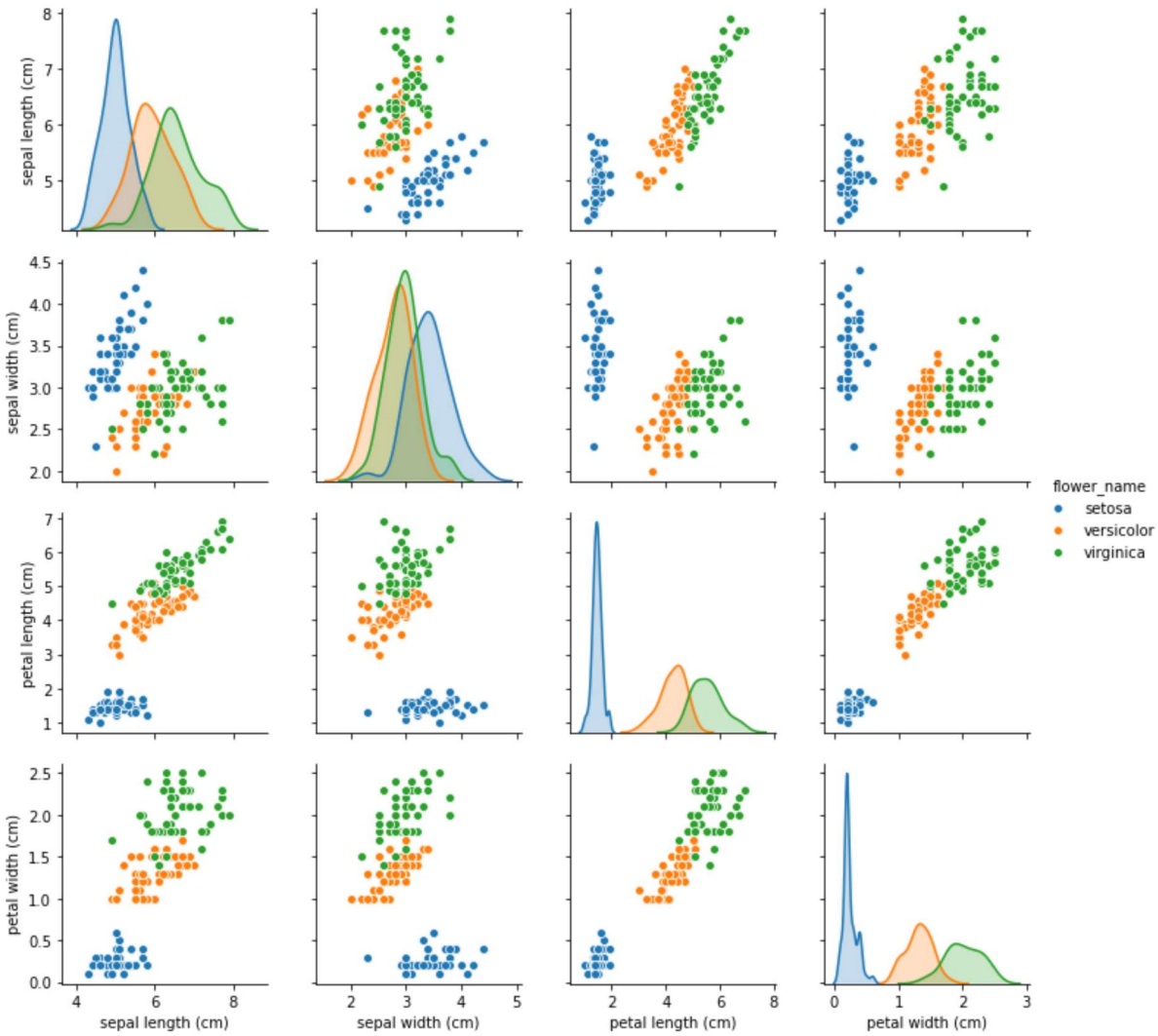


In [20]:

```
iris2=df.drop(columns = ['target'])
sns.pairplot(iris2, hue="flower_name")
```

Out[20]:

<seaborn.axisgrid.PairGrid at 0x7faa57d60990>



In [21]:

```
X = df.drop(['target','flower_name'], axis='columns')
y = df.target
```

In [22]:

```
from sklearn.model_selection import train_test_split
```

In [ ]:

```

```

In [23]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

In [24]:

```
len(X_train), len(X_test)
```

Out[24]:

(105, 45)

Support Vector Machine (SVM)

A Support Vector Machine is a supervised machine learning algorithm which can be used for both classification and regression problems. It follows a technique called the kernel trick to transform the data and based on these transformations, it finds an optimal boundary between the possible outputs.

In [25]:

```
from sklearn.svm import SVC
model = SVC(gamma='auto')
```

In [26]:

```
model.fit(X_train, y_train,)
```

Out[26]:

SVC(gamma='auto')

In [27]:

```
print('The accuracy of the SVM is',model.score(X_test, y_test),'with gamma and c value is Auto')
```

The accuracy of the SVM is 0.9555555555555556 with gamma and c value is Auto

'C' is a hypermeter which is set before the training model and used to control error.

'Gamma' is also a hypermeter which is set before the training model and used to

In [28]:

```
model_C = SVC(C=10,gamma='auto')
model_C.fit(X_train, y_train)
```

Out[28]:

SVC(C=10, gamma='auto')

In [29]:

```
model_C = SVC(C=1,gamma='auto')
model_C.fit(X_train, y_train)
print('The accuracy of the SVM is',model_C.score(X_test, y_test),'with gamma value is Auto and C value is 1')

model_C = SVC(C=10,gamma='auto')
model_C.fit(X_train, y_train)
print('The accuracy of the SVM is',model_C.score(X_test, y_test),'with gamma value is Auto and C value is 10')
```

The accuracy of the SVM is 0.9555555555555556 with gamma value is Auto and C value is 1
The accuracy of the SVM is 0.9555555555555556 with gamma value is Auto and C value is 10

In [ ]:

```

```

In [30]:

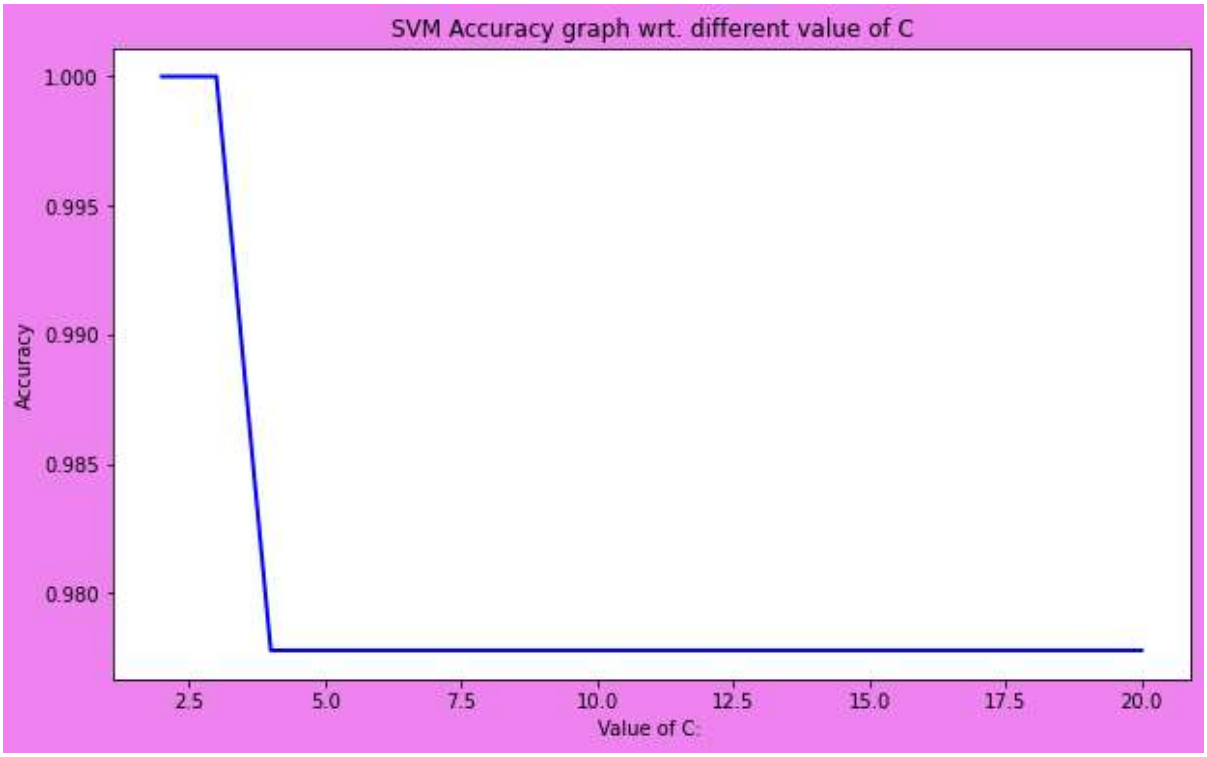
```
acc_SVM = [0] * 11
k=[2,3,4,5,6,7,8,9,10,15,20]
j=0;
for i in k:
    model_C_1 = SVC(C=i,gamma='auto')
    model_C_1.fit(X, y)
    acc_SVM[j]=model_C_1.score(X_test, y_test)
    j=j+1;
```

In [31]:

```
plt.figure(1,figsize=[10.0,6.0],facecolor="violet")
plt.ylabel('Accuracy')
plt.xlabel('Value of C: ')
plt.title('SVM Accuracy graph wrt. different value of C')
plt.plot(k,acc_SVM,color="blue",linewidth=2, markersize=12,)
```

Out[31]:

[<matplotlib.lines.Line2D at 0x7faa571242d0>]



In [32]:

```
model_g = SVC(gamma=10)
model_g.fit(X_train, y_train)
print('The accuracy of the SVM is',model_g.score(X_test, y_test),'with gamma value is 7 and C value is auto.')


model_g = SVC(gamma=1)
model_g.fit(X_train, y_train)
print('The accuracy of the SVM is',model_g.score(X_test, y_test),'with gamma value is 1 and C value is auto.')
```

The accuracy of the SVM is 0.9555555555555556 with gamma value is 7 and C value is auto.
The accuracy of the SVM is 0.9555555555555556 with gamma value is 1 and C value is auto.

In [33]:

```
model_linear_kernal = SVC(kernel='rbf',gamma='auto')
model_linear_kernal.fit(X_train, y_train)

print('The accuracy of the SVM is',model_linear_kernal.score(X_test, y_test),'kernal=rbf')
```

The accuracy of the SVM is 0.9555555555555556 kernal=rbf

K-Nearest Neighbors (KNN)

KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution. In other words, the model structure determined from the dataset.

In [34]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [35]:

```
neigh = KNeighborsClassifier(n_neighbors=2)
neigh.fit(X, y)
neigh.score(X_test, y_test)
print('The accuracy of the KNN is',neigh.score(X_test, y_test),'WHERE n_neighbors=2')
```

The accuracy of the KNN is 0.9777777777777777 WHERE n_neighbors=2

/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/base.py:442: UserWarn
ing: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  "X does not have valid feature names, but"
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/base.py:442: UserWarn
ing: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  "X does not have valid feature names, but"

In [36]:

```
acc = [0] * 5
k=[2,3,4,5,6]
for i in k:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X, y)
    acc[i-2]=neigh.score(X_test, y_test)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/base.py:442: UserWarn
ing: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  "X does not have valid feature names, but"
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/base.py:442: UserWarn
ing: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  "X does not have valid feature names, but"
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/base.py:442: UserWarn
ing: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  "X does not have valid feature names, but"
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/base.py:442: UserWarn
ing: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  "X does not have valid feature names, but"
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/base.py:442: UserWarn
ing: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  "X does not have valid feature names, but"
```

In [37]:

```
acc
```

Out[37]:

```
[0.9777777777777777,
 0.9555555555555556,
 0.9555555555555556,
 0.9555555555555556,
 0.9777777777777777]
```

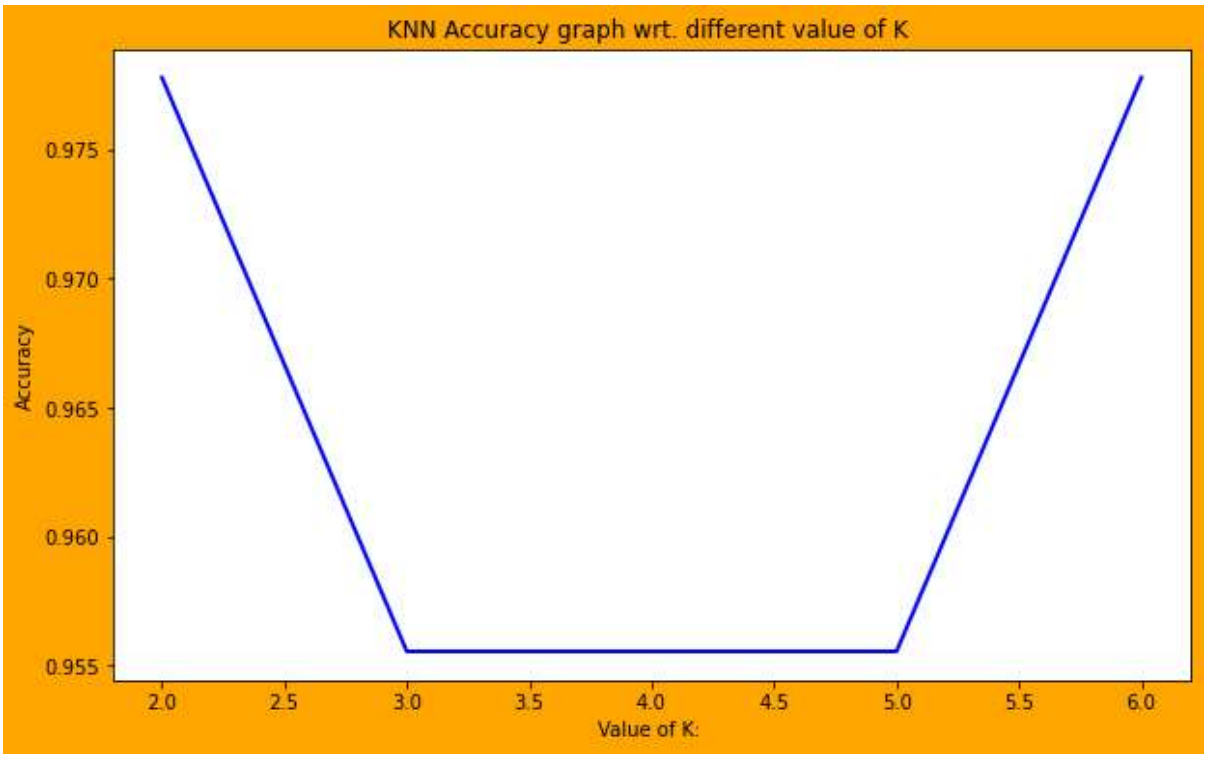
KNN accuracy plot for different value of n_neighbors

In [38]:

```
plt.figure(1,figsize=[10.0,6.0],facecolor="orange")
plt.ylabel('Accuracy')
plt.xlabel('Value of K: ')
plt.title('KNN Accuracy graph wrt. different value of K ')
plt.plot(k,acc,color="blue",linewidth=2, markersize=12,)
```

Out[38]:

[<matplotlib.lines.Line2D at 0x7faa5417a510>]



Decision Trees (DTs)

Treesare a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

In [39]:

```
pip install --upgrade sklearn
```

```
Requirement already satisfied: sklearn in /home/jupyterlab/conda/envs/python/lib/python3.7/site
-packages (0.0)
Requirement already satisfied: scikit-learn in /home/jupyterlab/conda/envs/python/lib/python3.7/
site-packages (from sklearn) (1.0)
Requirement already satisfied: joblib>=0.11 in /home/jupyterlab/conda/envs/python/lib/python3.7/
site-packages (from scikit-learn->sklearn) (1.0.1)
Requirement already satisfied: scipy>=1.1.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/
site-packages (from scikit-learn->sklearn) (1.7.1)
Requirement already satisfied: numpy>=1.14.6 in /home/jupyterlab/conda/envs/python/lib/python
3.7/site-packages (from scikit-learn->sklearn) (1.21.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /home/jupyterlab/conda/envs/python/lib/p
ython3.7/site-packages (from scikit-learn->sklearn) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [40]:

```
import sklearn
from sklearn.tree import DecisionTreeClassifier
import matplotlib.image as mpimg
from sklearn import tree
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
```

In [41]:

```
pip install --upgrade scikit-learn
```

```
Requirement already satisfied: scikit-learn in /home/jupyterlab/conda/envs/python/lib/python3.7/
site-packages (1.0)
Requirement already satisfied: joblib>=0.11 in /home/jupyterlab/conda/envs/python/lib/python3.7/
site-packages (from scikit-learn) (1.0.1)
Requirement already satisfied: scipy>=1.1.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/
site-packages (from scikit-learn) (1.7.1)
Requirement already satisfied: numpy>=1.14.6 in /home/jupyterlab/conda/envs/python/lib/python
3.7/site-packages (from scikit-learn) (1.21.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /home/jupyterlab/conda/envs/python/lib/p
ython3.7/site-packages (from scikit-learn) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [42]:

```
dtree=DecisionTreeClassifier(max_depth=3)
```

In [43]:

```
clf = dtree.fit(X,y)
```

In [44]:

```
predTree = clf.predict(X_test)
print("Accuracy with depth of 3:",metrics.accuracy_score(y_test, predTree))
```

Accuracy with depth of 3: 0.9555555555555556

In [45]:

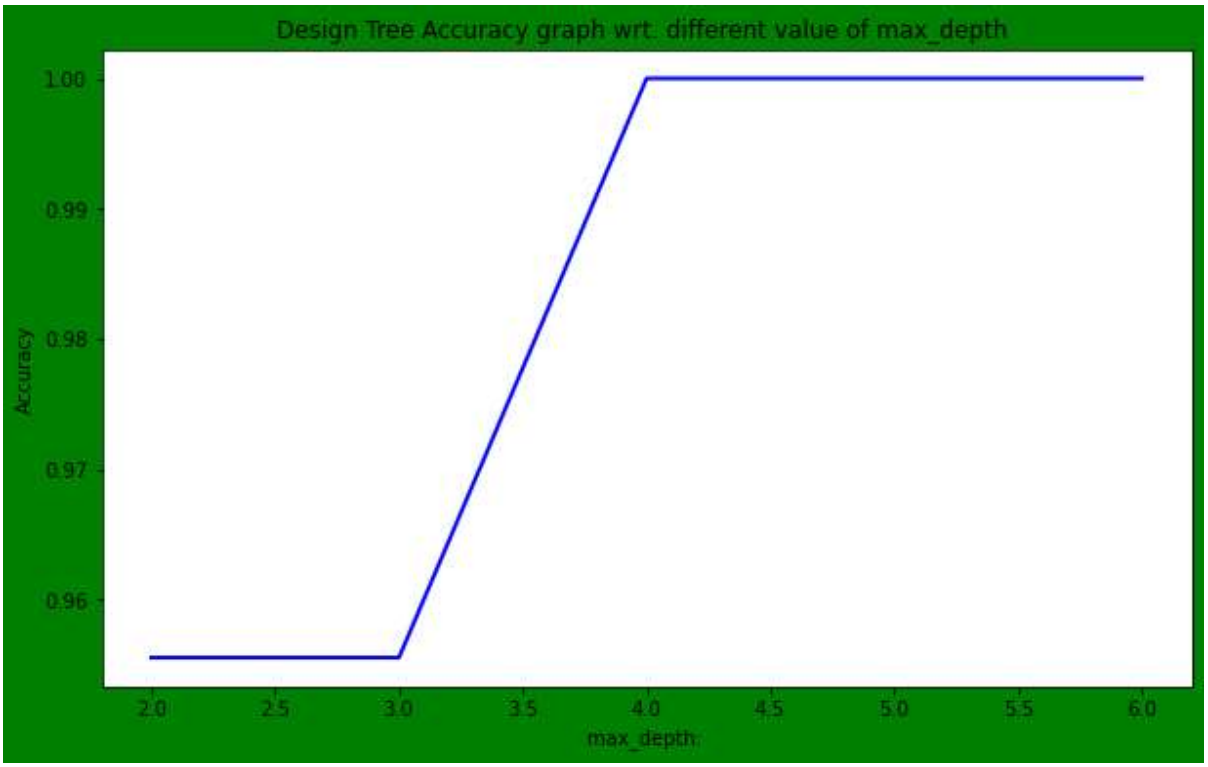
```
acc_dt = [0] * 5
k=[2,3,4,5,6]
for i in k:
    DT = dtree=DecisionTreeClassifier(max_depth=i)
    DT.fit(X, y)
    acc_dt[i-2]=DT.score(X_test, y_test)
```

In [46]:

```
plt.figure(1,figsize=[10.0,6.0],facecolor="green")
plt.ylabel('Accuracy')
plt.xlabel('max_depth: ')
plt.title('Design Tree Accuracy graph wrt. different value of max_depth ')
plt.plot(k,acc_dt,color="blue",linewidth=2, markersize=12,)
```

Out[46]:

[<matplotlib.lines.Line2D at 0x7faa540afa50>]



In [47]:

```
sklearn_version = sklearn.__version__
print(sklearn_version)
```

1.0

In [48]:

```
from sklearn.tree import plot_tree
```

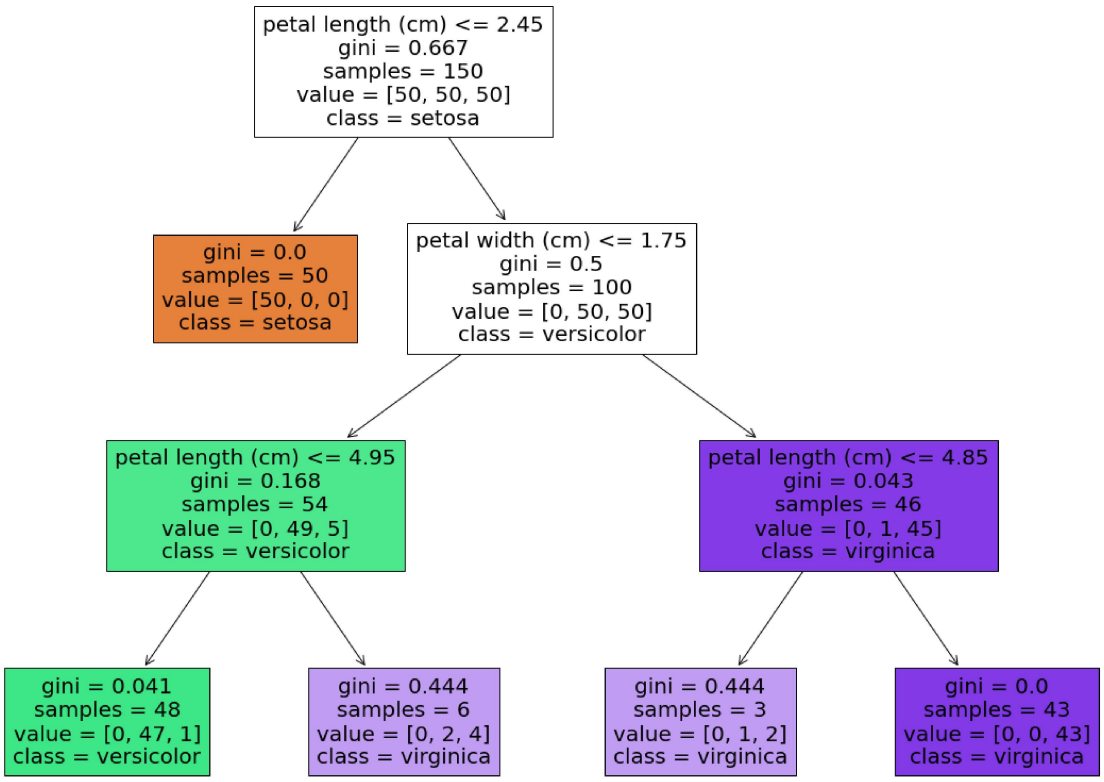
In [49]:

```
# fig = clf.fit(X,y)
# tree.plot_tree(clf)
# plt.show()
```

Design tree visualization using plot

In [50]:

```
fig = plt.figure(figsize=(20,15))
_ = tree.plot_tree(clf,
                   feature_names=iris.feature_names,
                   class_names=iris.target_names,
                   filled=True)
```



Logistic Regression

Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. Dichotomous means there are only two possible classes. For example, it can

In [51]:

```
from sklearn.linear_model import LogisticRegression
```

In [52]:

```
model = LogisticRegression()
model.fit(X,y)
prediction=model.predict(X_test)
print('The accuracy of the Logistic Regression is',metrics.accuracy_score(prediction,y_test))
```

The accuracy of the Logistic Regression is 0.9555555555555556

/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/linear_model/_logisti
c.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,

In [ ]:

```

```