Q3. (To Explore Decision Tree Algorithm)

For the given 'Iris' dataset, create the Decision Tree classifier and visualize it graphically. The purpose is if we feed any new data to this classifier, it would be able to predict the right class accordingly.

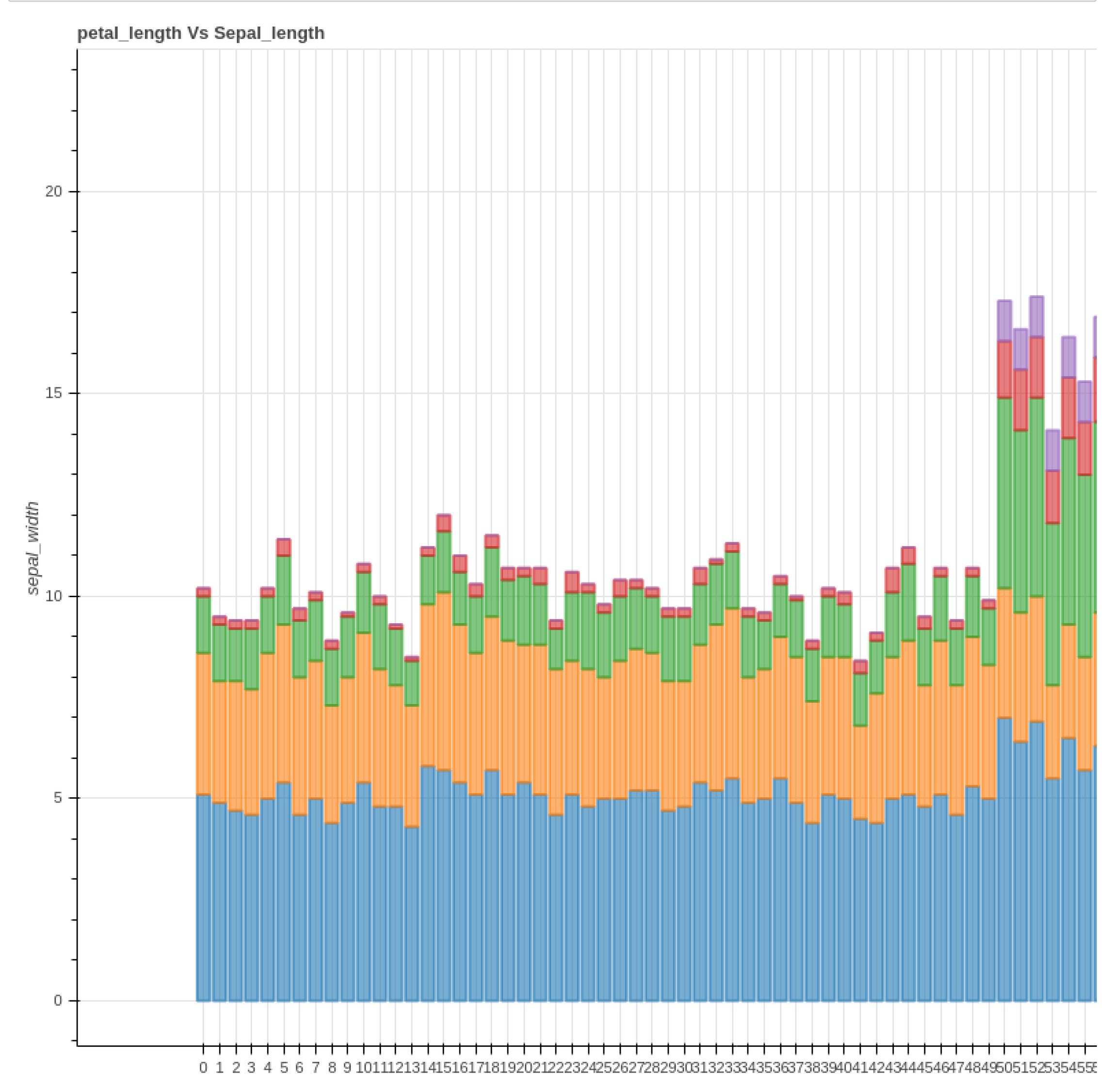
In [11]:
```python
import numpy as np
import pandas as pd

import pandas_bokeh
import seaborn as sns
import matplotlib.pyplot as plt
pandas_bokeh.output_notebook()
#pd.set_option('plotting.backend', 'pandas_bokeh')
# Create Bokeh-Table with DataFrame:
from bokeh.models.widgets import DataTable, TableColumn
from bokeh.models import ColumnDataSource


# Import the needed matplotlib functionality for scatter plot visualization.
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.datasets import load_iris
# Import the model and an additional visualization tool.
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

BokehJS 2.0.2 successfully loaded.

In [12]:
```python
# Load the iris dataset from scikit-learn (note the use of from [library] import [function]
 above)
iris = load_iris()


X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Categorical.from_codes(iris.target, iris.target_names)
data1 = pd.DataFrame(data= np.c_[iris['data'], iris['target']],columns= iris['feature_names'
] + ['target'])
print(data1)
```

```
     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
5                  5.4               3.9                1.7               0.4
6                  4.6               3.4                1.4               0.3
7                  5.0               3.4                1.5               0.2
8                  4.4               2.9                1.4               0.2
9                  4.9               3.1                1.5               0.1
10                 5.4               3.7                1.5               0.2
11                 4.8               3.4                1.6               0.2
12                 4.8               3.0                1.4               0.1
13                 4.3               3.0                1.1               0.1
14                 5.8               4.0                1.2               0.2
15                 5.7               4.4                1.5               0.4
16                 5.4               3.9                1.3               0.4
17                 5.1               3.5                1.4               0.3
18                 5.7               3.8                1.7               0.3
19                 5.1               3.8                1.5               0.3
20                 5.4               3.4                1.7               0.2
21                 5.1               3.7                1.5               0.4
22                 4.6               3.6                1.0               0.2
23                 5.1               3.3                1.7               0.5
24                 4.8               3.4                1.9               0.2
25                 5.0               3.0                1.6               0.2
26                 5.0               3.4                1.6               0.4
27                 5.2               3.5                1.5               0.2
28                 5.2               3.4                1.4               0.2
29                 4.7               3.2                1.6               0.2
..                 ...               ...                ...               ...
120                6.9               3.2                5.7               2.3
121                5.6               2.8                4.9               2.0
122                7.7               2.8                6.7               2.0
123                6.3               2.7                4.9               1.8
124                6.7               3.3                5.7               2.1
125                7.2               3.2                6.0               1.8
126                6.2               2.8                4.8               1.8
127                6.1               3.0                4.9               1.8
128                6.4               2.8                5.6               2.1
129                7.2               3.0                5.8               1.6
130                7.4               2.8                6.1               1.9
131                7.9               3.8                6.4               2.0
132                6.4               2.8                5.6               2.2
133                6.3               2.8                5.1               1.5
134                6.1               2.6                5.6               1.4
135                7.7               3.0                6.1               2.3
136                6.3               3.4                5.6               2.4
137                6.4               3.1                5.5               1.8
138                6.0               3.0                4.8               1.8
139                6.9               3.1                5.4               2.1
140                6.7               3.1                5.6               2.4
141                6.9               3.1                5.1               2.3
142                5.8               2.7                5.1               1.9
143                6.8               3.2                5.9               2.3
144                6.7               3.3                5.7               2.5
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

     target
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0
5       0.0
6       0.0
7       0.0
8       0.0
9       0.0
10      0.0
11      0.0
12      0.0
13      0.0
14      0.0
15      0.0
16      0.0
17      0.0
18      0.0
19      0.0
20      0.0
21      0.0
22      0.0
23      0.0
24      0.0
25      0.0
26      0.0
27      0.0
28      0.0
29      0.0
..      ...
120     2.0
121     2.0
122     2.0
123     2.0
124     2.0
125     2.0
126     2.0
127     2.0
128     2.0
129     2.0
130     2.0
131     2.0
132     2.0
133     2.0
134     2.0
135     2.0
136     2.0
137     2.0
138     2.0
139     2.0
140     2.0
141     2.0
142     2.0
143     2.0
144     2.0
145     2.0
146     2.0
147     2.0
148     2.0
149     2.0

[150 rows x 5 columns]
```

In [13]:
```python
data1.plot_bokeh.bar(xlabel="petal_length",ylabel="sepal_width",alpha=0.6,figsize=(2000,800
),title="petal_length Vs Sepal_length",category="species",stacked=True)
```



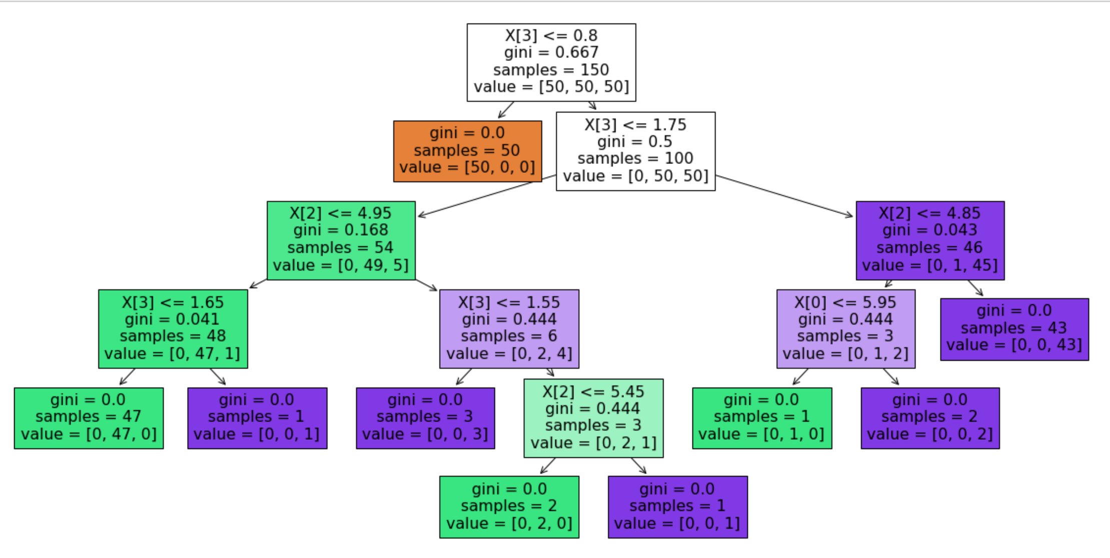
Out[13]: **Figure**(id = '1233', ...)

We know that Decision trees can handle categorical data, we still encode the targets in terms of digits (i.e. setosa=0, versicolor=1, virginica=2) in order to create a confusion matrix at a later point :

In [14]:
```python
y = pd.get_dummies(y)
print(y)
```

```
     setosa  versicolor  virginica
0         1           0          0
1         1           0          0
2         1           0          0
3         1           0          0
4         1           0          0
5         1           0          0
6         1           0          0
7         1           0          0
8         1           0          0
9         1           0          0
10        1           0          0
11        1           0          0
12        1           0          0
13        1           0          0
14        1           0          0
15        1           0          0
16        1           0          0
17        1           0          0
18        1           0          0
19        1           0          0
20        1           0          0
21        1           0          0
22        1           0          0
23        1           0          0
24        1           0          0
25        1           0          0
26        1           0          0
27        1           0          0
28        1           0          0
29        1           0          0
..      ...         ...        ...
120       0           0          1
121       0           0          1
122       0           0          1
123       0           0          1
124       0           0          1
125       0           0          1
126       0           0          1
127       0           0          1
128       0           0          1
129       0           0          1
130       0           0          1
131       0           0          1
132       0           0          1
133       0           0          1
134       0           0          1
135       0           0          1
136       0           0          1
137       0           0          1
138       0           0          1
139       0           0          1
140       0           0          1
141       0           0          1
142       0           0          1
143       0           0          1
144       0           0          1
145       0           0          1
146       0           0          1
147       0           0          1
148       0           0          1
149       0           0          1

[150 rows x 3 columns]
```

In [15]:
```python
# Apply the decision tree classifier model to the data using all four parameters at once.
model_all_params = DecisionTreeClassifier().fit(iris.data, iris.target)
# Prepare a plot figure with set size.
plt.figure(figsize = (20,10))
# Plot the decision tree, showing the decisive values and the improvements in Gini impurity
 along the way.
plot_tree(model_all_params,
          filled=True      )
# Display the tree plot figure.
plt.show()
```



As, this is a classification problem, so we make use of a confusion matrix to gauge the accuracy of our model.

In [16]:
```python
# Seperating the data into dependent and independent variables
X = data1.iloc[:, :-1].values
y = data1.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.6, random_state = 0)

classifier = DecisionTreeClassifier()

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
from sklearn.metrics import accuracy_score
print('Accuracy is:',accuracy_score(y_pred,y_test))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        26
         1.0       0.94      0.97      0.96        33
         2.0       0.97      0.94      0.95        31

    accuracy                           0.97        90
   macro avg       0.97      0.97      0.97        90
weighted avg       0.97      0.97      0.97        90

[[26  0  0]
 [ 0 32  1]
 [ 0  2 29]]
Accuracy is: 0.9666666666666667
```

□Now above we see that it giving approximately 97% which is good number in terms of prediction.

In [ ]: