Prepare the dataset

Import the necessary library:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
```

Import the iris dataset and separating it to testing and training set:

```
iris = load_iris()
iris_data = iris.data
iris_label = iris.target
X_train, X_test, y_train, y_test = train_test_split(iris_data, iris_label, test_size=0.20)
```

Training and Predictions

To train a decision tree with maximum depth=2

```
#bulid the decision three with maximum depth=2
classifier = DecisionTreeClassifier(max_depth=2)
classifier.fit(X_train, y_train)
```

To make predictions on test data and evaluate the predictions, execute the following script:

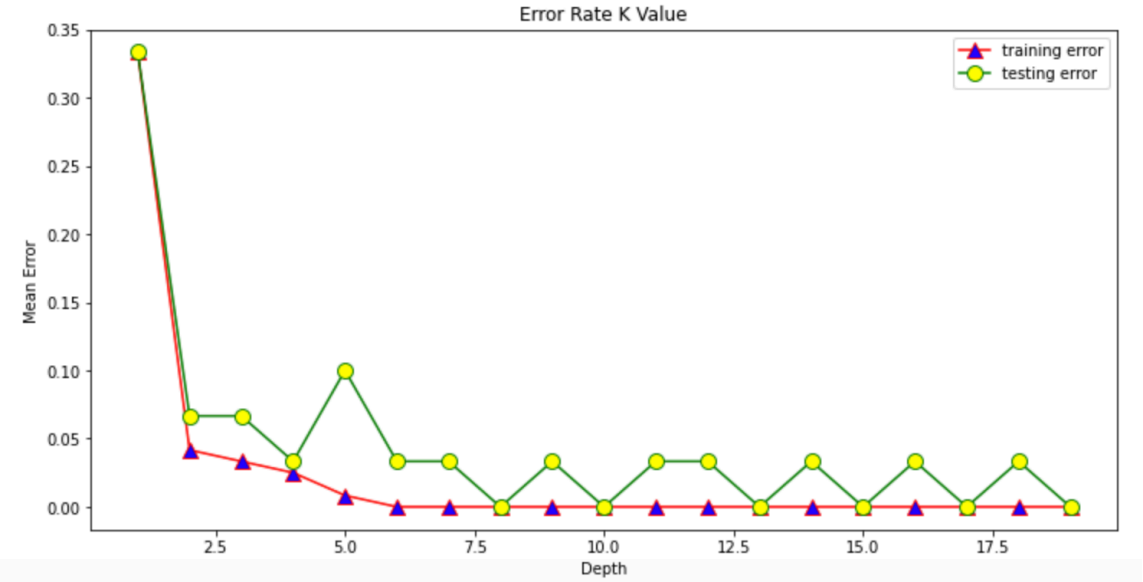
```
#make predictions
pred_test = classifier.predict(X_test)
#evaluate the prediction
accuracy_score(pred_test, y_test)
```

Try different maximum depth and report results:

```
depth_range=20
test_error = []
train_error=[]
# change the maximum depth from 1 to 20
for i in range(1, depth_range):
    # create the model
    classifier = DecisionTreeClassifier(max_depth=i)
    # fit the model
    classifier.fit(X_train, y_train)
    pred_test = classifier.predict(X_test)
    pred_train = classifier.predict(X_train)
    # calculate the error rate and save it
    test_error.append(1-accuracy_score(pred_test, y_test))
    train_error.append(1-accuracy_score(pred_train, y_train))
```

To plot the error values against maximum depth values, execute the following script to create the plot:

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, depth_range), train_error, color='red', marker='^',
         markerfacecolor='blue', markersize=10,label='training error')
plt.plot(range(1, depth_range), test_error, color='green',  marker='o',
         markerfacecolor='yellow', markersize=10,label='testing error')
plt.title('Error Rate K Value')
plt.xlabel('Depth')
plt.ylabel('Mean Error')
plt.legend()
plt.show()
```



To build a decision tree with min number of leave node records = 10, using the following script:

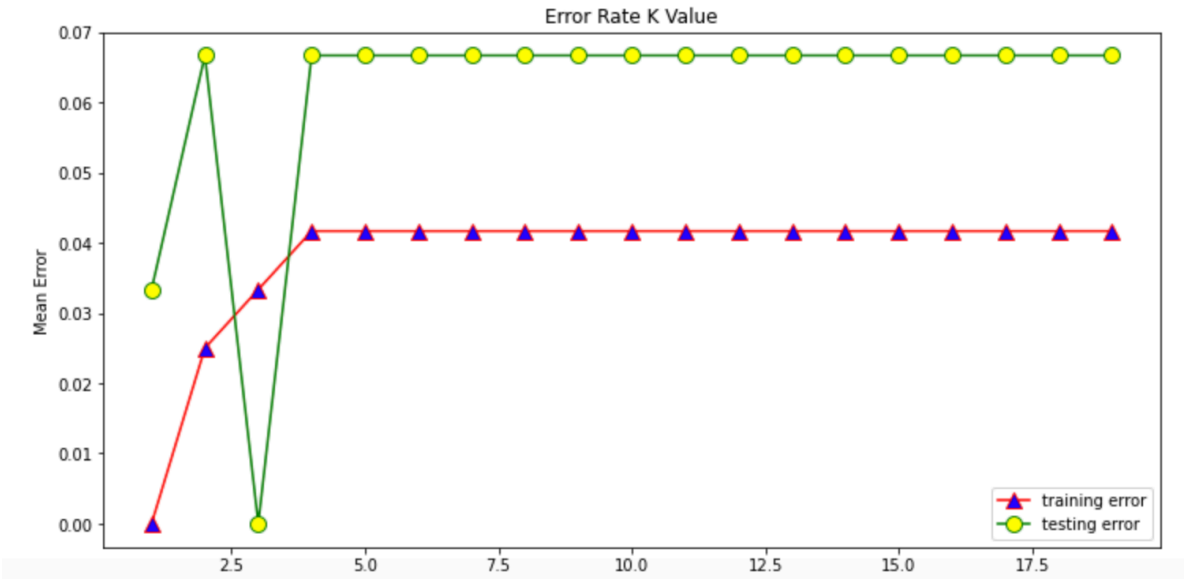
```
classifier = DecisionTreeClassifier(min_samples_leaf=10)
# fit the model
classifier.fit(X_train, y_train)
```

To train decision trees with min number of leave node records from 1 to 20 and classify training and testing data:

```
sample_leaf_range = 20
test_error = []
train_error=[]
# change the maximum depth from 1 to 20
for i in range(1, depth_range):
    # create the model
    classifier = DecisionTreeClassifier(min_samples_leaf=i)
    # fit the model
    classifier.fit(X_train, y_train)
    pred_test = classifier.predict(X_test)
    pred_train = classifier.predict(X_train)
    # calculate the error rate and save it
    test_error.append(1-accuracy_score(pred_test, y_test))
    train_error.append(1-accuracy_score(pred_train, y_train))
```

To plot the testing and training error against leave node records, using the flowing script:

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, depth_range), train_error, color='red', marker='^',
         markerfacecolor='blue', markersize=10,label='training error')
plt.plot(range(1, depth_range), test_error, color='green',  marker='o',
         markerfacecolor='yellow', markersize=10,label='testing error')
plt.title('Error Rate K Value')
plt.xlabel('Depth')
plt.ylabel('Mean Error')
plt.legend()
plt.show()
```



To add noise vectors in training and testing data, using the following script:

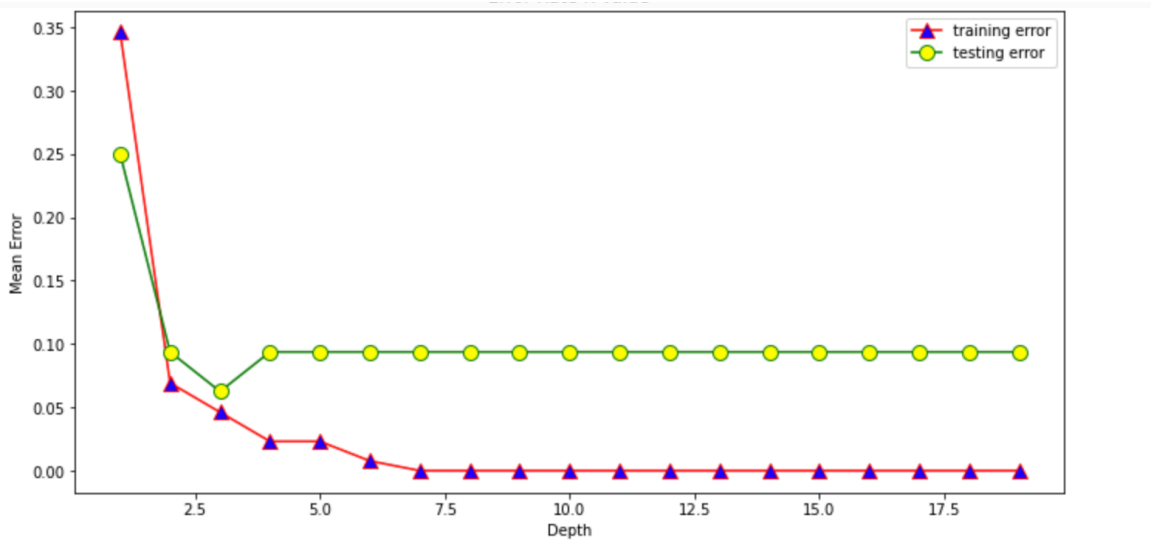
```
#creat 10 noise vectors to training data
train_noise = 10*np.random.rand(10,X_train.shape[1])
train_noiselable = np.random.randint(3, size=(10,))
#create 2 noise vectors to testing data
test_noise = 10*np.random.rand(2,X_test.shape[1])
test_noiselable = np.random.randint(3, size=(2,))
#add the data and lable to train and testing data
X_train = np.concatenate((X_train, train_noise), axis=0)
y_train = np.concatenate((y_train,train_noiselable),axis=0)
X_test = np.concatenate((X_test,test_noise),axis=0)
y_test = np.concatenate((y_test,test_noiselable),axis=0)
```

Then you can use the modified data to train and test the classifier.

```
depth_range=20
test_error = []
train_error=[]
# change the maximum depth from 1 to 20
for i in range(1, depth_range):
    # create the model
    classifier = DecisionTreeClassifier(max_depth=i)
    # fit the model
    classifier.fit(X_train, y_train)
    pred_test = classifier.predict(X_test)
    pred_train = classifier.predict(X_train)
    # calculate the error rate and save it
    test_error.append(1-accuracy_score(pred_test, y_test))
    train_error.append(1-accuracy_score(pred_train, y_train))

plt.figure(figsize=(12, 6))

plt.plot(range(1, depth_range), train_error, color='red', marker='^',
         markerfacecolor='blue', markersize=10,label='training error')
plt.plot(range(1, depth_range), test_error, color='green',  marker='o',
         markerfacecolor='yellow', markersize=10,label='testing error')
plt.title('Error Rate K Value')
plt.xlabel('Depth')
plt.ylabel('Mean Error')
plt.legend()
plt.show()
```



Pruning

Apply Minimal Cost-Complexity Pruning on decision trees, set the ccp_alpha bigger than 0. For example apply Minimal Cost-Complexity Pruning on a tree with min number of leave node records = 2, the ccp_alpha is set to 0.01:

```
classifier = DecisionTreeClassifier(min_samples_leaf=2,ccp_alpha=0.01)
```