```
In [1]: import numpy as np
        import pandas as pd
        from sklearn import datasets
        from sklearn import tree
        from sklearn import model_selection
        from sklearn import metrics
```

Problem 1

```
In [2]: iris = datasets.load_iris()
        iris
```

```
Out[2]: {'data': 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],
               [5.4, 3.9, 1.3, 0.4],
               [5.1, 3.5, 1.4, 0.3],
               [5.7, 3.8, 1.7, 0.3],
               [5.1, 3.8, 1.5, 0.3],
               [5.4, 3.4, 1.7, 0.2],
               [5.1, 3.7, 1.5, 0.4],
               [4.6, 3.6, 1. , 0.2],
               [5.1, 3.3, 1.7, 0.5],
               [4.8, 3.4, 1.9, 0.2],
               [5. , 3. , 1.6, 0.2],
               [5. , 3.4, 1.6, 0.4],
               [5.2, 3.5, 1.5, 0.2],
               [5.2, 3.4, 1.4, 0.2],
               [4.7, 3.2, 1.6, 0.2],
               [4.8, 3.1, 1.6, 0.2],
               [5.4, 3.4, 1.5, 0.4],
               [5.2, 4.1, 1.5, 0.1],
               [5.5, 4.2, 1.4, 0.2],
               [4.9, 3.1, 1.5, 0.2],
               [5. , 3.2, 1.2, 0.2],
               [5.5, 3.5, 1.3, 0.2],
               [4.9, 3.6, 1.4, 0.1],
               [4.4, 3. , 1.3, 0.2],
               [5.1, 3.4, 1.5, 0.2],
               [5. , 3.5, 1.3, 0.3],
               [4.5, 2.3, 1.3, 0.3],
               [4.4, 3.2, 1.3, 0.2],
               [5. , 3.5, 1.6, 0.6],
               [5.1, 3.8, 1.9, 0.4],
               [4.8, 3. , 1.4, 0.3],
               [5.1, 3.8, 1.6, 0.2],
               [4.6, 3.2, 1.4, 0.2],
               [5.3, 3.7, 1.5, 0.2],
               [5. , 3.3, 1.4, 0.2],
               [7. , 3.2, 4.7, 1.4],
               [6.4, 3.2, 4.5, 1.5],
               [6.9, 3.1, 4.9, 1.5],
               [5.5, 2.3, 4. , 1.3],
               [6.5, 2.8, 4.6, 1.5],
               [5.7, 2.8, 4.5, 1.3],
               [6.3, 3.3, 4.7, 1.6],
               [4.9, 2.4, 3.3, 1. ],
               [6.6, 2.9, 4.6, 1.3],
               [5.2, 2.7, 3.9, 1.4],
               [5. , 2. , 3.5, 1. ],
               [5.9, 3. , 4.2, 1.5],
               [6. , 2.2, 4. , 1. ],
               [6.1, 2.9, 4.7, 1.4],
               [5.6, 2.9, 3.6, 1.3],
               [6.7, 3.1, 4.4, 1.4],
               [5.6, 3. , 4.5, 1.5],
               [5.8, 2.7, 4.1, 1. ],
               [6.2, 2.2, 4.5, 1.5],
               [5.6, 2.5, 3.9, 1.1],
               [5.9, 3.2, 4.8, 1.8],
               [6.1, 2.8, 4. , 1.3],
               [6.3, 2.5, 4.9, 1.5],
               [6.1, 2.8, 4.7, 1.2],
               [6.4, 2.9, 4.3, 1.3],
               [6.6, 3. , 4.4, 1.4],
               [6.8, 2.8, 4.8, 1.4],
               [6.7, 3. , 5. , 1.7],
               [6. , 2.9, 4.5, 1.5],
               [5.7, 2.6, 3.5, 1. ],
               [5.5, 2.4, 3.8, 1.1],
               [5.5, 2.4, 3.7, 1. ],
               [5.8, 2.7, 3.9, 1.2],
               [6. , 2.7, 5.1, 1.6],
               [5.4, 3. , 4.5, 1.5],
               [6. , 3.4, 4.5, 1.6],
               [6.7, 3.1, 4.7, 1.5],
               [6.3, 2.3, 4.4, 1.3],
               [5.6, 3. , 4.1, 1.3],
               [5.5, 2.5, 4. , 1.3],
               [5.5, 2.6, 4.4, 1.2],
               [6.1, 3. , 4.6, 1.4],
               [5.8, 2.6, 4. , 1.2],
               [5. , 2.3, 3.3, 1. ],
               [5.6, 2.7, 4.2, 1.3],
               [5.7, 3. , 4.2, 1.2],
               [5.7, 2.9, 4.2, 1.3],
               [6.2, 2.9, 4.3, 1.3],
               [5.1, 2.5, 3. , 1.1],
               [5.7, 2.8, 4.1, 1.3],
               [6.3, 3.3, 6. , 2.5],
               [5.8, 2.7, 5.1, 1.9],
               [7.1, 3. , 5.9, 2.1],
               [6.3, 2.9, 5.6, 1.8],
               [6.5, 3. , 5.8, 2.2],
               [7.6, 3. , 6.6, 2.1],
               [4.9, 2.5, 4.5, 1.7],
               [7.3, 2.9, 6.3, 1.8],
               [6.7, 2.5, 5.8, 1.8],
               [7.2, 3.6, 6.1, 2.5],
               [6.5, 3.2, 5.1, 2. ],
               [6.4, 2.7, 5.3, 1.9],
               [6.8, 3. , 5.5, 2.1],
               [5.7, 2.5, 5. , 2. ],
               [5.8, 2.8, 5.1, 2.4],
               [6.4, 3.2, 5.3, 2.3],
               [6.5, 3. , 5.5, 1.8],
               [7.7, 3.8, 6.7, 2.2],
               [7.7, 2.6, 6.9, 2.3],
               [6. , 2.2, 5. , 1.5],
               [6.9, 3.2, 5.7, 2.3],
               [5.6, 2.8, 4.9, 2. ],
               [7.7, 2.8, 6.7, 2. ],
               [6.3, 2.7, 4.9, 1.8],
               [6.7, 3.3, 5.7, 2.1],
               [7.2, 3.2, 6. , 1.8],
               [6.2, 2.8, 4.8, 1.8],
               [6.1, 3. , 4.9, 1.8],
               [6.4, 2.8, 5.6, 2.1],
               [7.2, 3. , 5.8, 1.6],
               [7.4, 2.8, 6.1, 1.9],
               [7.9, 3.8, 6.4, 2. ],
               [6.4, 2.8, 5.6, 2.2],
               [6.3, 2.8, 5.1, 1.5],
               [6.1, 2.6, 5.6, 1.4],
               [7.7, 3. , 6.1, 2.3],
               [6.3, 3.4, 5.6, 2.4],
               [6.4, 3.1, 5.5, 1.8],
               [6. , 3. , 4.8, 1.8],
               [6.9, 3.1, 5.4, 2.1],
               [6.7, 3.1, 5.6, 2.4],
               [6.9, 3.1, 5.1, 2.3],
               [5.8, 2.7, 5.1, 1.9],
               [6.8, 3.2, 5.9, 2.3],
               [6.7, 3.3, 5.7, 2.5],
               [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]]),
 'target': 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]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Characteristi
cs:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n    :Number of Attributes: 4 nu
meric, predictive attributes and the class\n    :Attribute Information:\n        - sepal length in cm
\n        - sepal width in cm\n        - petal length in cm\n        - petal width in cm\n        - c
lass:\n                - Iris-Setosa\n                - Iris-Versicolour\n                - Iris-Virg
inica\n                \n    :Summary Statistics:\n\n    ============== ==== ==== ======= ===== =====
===============\n                    Min  Max   Mean    SD   Class Correlation\n    ============== ==
== ==== ======= ===== ====================\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n
sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76   1.76    0.949
0  (high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ============== ==== ==
== ======= ===== ====================\n\n    :Missing Attribute Values: None\n    :Class Distributio
n: 33.3% for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PL
U@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A. Fisher.
The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\n
Machine Learning Repository, which has two wrong data points.\n\nThis is perhaps the best known datab
ase to be found in the\npattern recognition literature.  Fisher\'s paper is a classic in the field an
d\nis referenced frequently to this day.  (See Duda & Hart, for example.)  The\ndata set contains 3 c
lasses of 50 instances each, where each class refers to a\ntype of iris plant.  One class is linearly
separable from the other 2; the\nlatter are NOT linearly separable from each other.\n\n.. topic:: Ref
erences\n\n   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n     Annual Eu
genics, 7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistics" (John Wi
ley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n
  (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nos
ing Around the Neighborhood: A New System\n     Structure and Classification Rule for Recognition in
Partially Exposed\n     Environments".  IEEE Transactions on Pattern Analysis and Machine\n     Intel
ligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IE
EE Transactions\n     on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings,
54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3 classes in the dat
a.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'C:\\anaconda3\\lib\\site-packages\\sklearn\\datasets\\data\\iris.csv'}
```

```
In [3]: iris_df = pd.DataFrame(data=iris.data, columns=["SepalLength", "SepalWidth", "PetalLength", "PetalWidt
        h"])
        iris_df
```

Out[3]:

	SepalLength	SepalWidth	PetalLength	PetalWidth
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
...	...	...	...	...
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

150 rows × 4 columns

```
In [4]: # Split the data between test and train with 20% of the test size
        X = iris.data
        Y = iris.target
        X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X,Y, test_size=0.2, random_state=0)
```

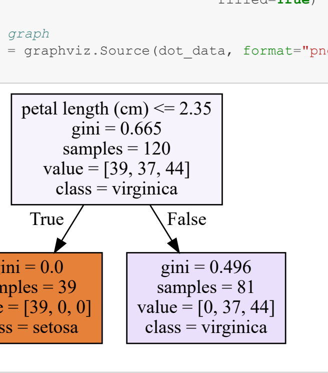
```
In [5]: # Calculate Precision, Recall and F1 Score
        max_depth_range = list(range(1, 6))
        print("\n")
        for depth in max_depth_range:
            clf = tree.DecisionTreeClassifier(min_samples_leaf=2, min_samples_split=5, max_depth = depth)
            clf.fit(X_train, Y_train)
            expected = Y_test
            predicted = clf.predict(X_test)
            print("******************************* Max depth = {} ********************************".format(dep
        th))
            print("\n")
            print("Precision, Recall and F1 Score with macro = {}".format(metrics.precision_recall_fscore_suppo
        rt(expected,predicted,average='macro')))
            print("Precision, Recall and F1 Score with micro = {}".format(metrics.precision_recall_fscore_suppo
        rt(expected,predicted,average='micro')))
            print("Precision, Recall and F1 Score with weighted = {}".format(metrics.precision_recall_fscore_su
        pport(expected,predicted,average='weighted')))
            print("Accuracy = {}".format(metrics.accuracy_score(expected,predicted)))
            print("\n")
            print(metrics.classification_report(expected,predicted,target_names=iris.target_names))
            print("\n")
            print("Confusion matrix :")
            print(metrics.confusion_matrix(expected, predicted))
            print("\n")
            print("Recall = {}".format(metrics.recall_score(expected,predicted,average=None)))
            print("Precision = {}".format(metrics.precision_score(expected,predicted,average=None)))
            print("F1 score = {}".format(metrics.f1_score(expected,predicted,average=None)))
            print("\n")
```

```
******************************* Max depth = 1 ********************************


Precision, Recall and F1 Score with macro = (0.4385964912280702, 0.6666666666666666, 0.49333333333333
335, None)
Precision, Recall and F1 Score with micro = (0.5666666666666667, 0.5666666666666667, 0.56666666666666
67, None)
Precision, Recall and F1 Score with weighted = (0.4298245614035088, 0.5666666666666667, 0.46266666666
66666, None)
Accuracy = 0.5666666666666667


              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       0.00      0.00      0.00        13
   virginica       0.32      1.00      0.48         6

    accuracy                           0.57        30
   macro avg       0.44      0.67      0.49        30
weighted avg       0.43      0.57      0.46        30



Confusion matrix :
[[11  0  0]
 [ 0  0 13]
 [ 0  0  6]]


Recall = [1. 0. 1.]
Precision = [1.         0.         0.31578947]
F1 score = [1.   0.   0.48]


******************************* Max depth = 2 ********************************


Precision, Recall and F1 Score with macro = (0.9761904761904763, 0.9444444444444445, 0.95735129068462
4, None)
Precision, Recall and F1 Score with micro = (0.9666666666666667, 0.9666666666666667, 0.96666666666666
67, None)
Precision, Recall and F1 Score with weighted = (0.9690476190476189, 0.9666666666666667, 0.96576879910
21324, None)
Accuracy = 0.9666666666666667


              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       0.93      1.00      0.96        13
   virginica       1.00      0.83      0.91         6

    accuracy                           0.97        30
   macro avg       0.98      0.94      0.96        30
weighted avg       0.97      0.97      0.97        30



Confusion matrix :
[[11  0  0]
 [ 0 13  0]
 [ 0  1  5]]


Recall = [1.         1.         0.83333333]
Precision = [1.         0.92857143 1.        ]
F1 score = [1.         0.96296296 0.90909091]


******************************* Max depth = 3 ********************************


Precision, Recall and F1 Score with macro = (0.9761904761904763, 0.9444444444444445, 0.95735129068462
4, None)
Precision, Recall and F1 Score with micro = (0.9666666666666667, 0.9666666666666667, 0.96666666666666
67, None)
Precision, Recall and F1 Score with weighted = (0.9690476190476189, 0.9666666666666667, 0.96576879910
21324, None)
Accuracy = 0.9666666666666667


              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       0.93      1.00      0.96        13
   virginica       1.00      0.83      0.91         6

    accuracy                           0.97        30
   macro avg       0.98      0.94      0.96        30
weighted avg       0.97      0.97      0.97        30



Confusion matrix :
[[11  0  0]
 [ 0 13  0]
 [ 0  1  5]]


Recall = [1.         1.         0.83333333]
Precision = [1.         0.92857143 1.        ]
F1 score = [1.         0.96296296 0.90909091]


******************************* Max depth = 4 ********************************


Precision, Recall and F1 Score with macro = (0.9761904761904763, 0.9444444444444445, 0.95735129068462
4, None)
Precision, Recall and F1 Score with micro = (0.9666666666666667, 0.9666666666666667, 0.96666666666666
67, None)
Precision, Recall and F1 Score with weighted = (0.9690476190476189, 0.9666666666666667, 0.96576879910
21324, None)
Accuracy = 0.9666666666666667


              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       0.93      1.00      0.96        13
   virginica       1.00      0.83      0.91         6

    accuracy                           0.97        30
   macro avg       0.98      0.94      0.96        30
weighted avg       0.97      0.97      0.97        30



Confusion matrix :
[[11  0  0]
 [ 0 13  0]
 [ 0  1  5]]


Recall = [1.         1.         0.83333333]
Precision = [1.         0.92857143 1.        ]
F1 score = [1.         0.96296296 0.90909091]


******************************* Max depth = 5 ********************************


Precision, Recall and F1 Score with macro = (0.9761904761904763, 0.9444444444444445, 0.95735129068462
4, None)
Precision, Recall and F1 Score with micro = (0.9666666666666667, 0.9666666666666667, 0.96666666666666
67, None)
Precision, Recall and F1 Score with weighted = (0.9690476190476189, 0.9666666666666667, 0.96576879910
21324, None)
Accuracy = 0.9666666666666667


              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       0.93      1.00      0.96        13
   virginica       1.00      0.83      0.91         6

    accuracy                           0.97        30
   macro avg       0.98      0.94      0.96        30
weighted avg       0.97      0.97      0.97        30



Confusion matrix :
[[11  0  0]
 [ 0 13  0]
 [ 0  1  5]]


Recall = [1.         1.         0.83333333]
Precision = [1.         0.92857143 1.        ]
F1 score = [1.         0.96296296 0.90909091]
```

```
C:\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_
division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning: Preci
sion is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` par
ameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

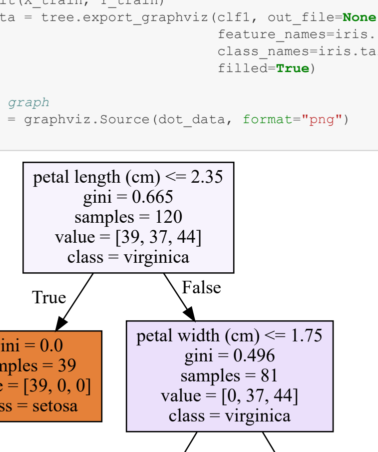
```
In [6]: import graphviz
        clf1 = tree.DecisionTreeClassifier(min_samples_leaf=2, min_samples_split=5, max_depth = 1)
        clf1.fit(X_train, Y_train)
        dot_data = tree.export_graphviz(clf1, out_file=None,
                                        feature_names=iris.feature_names,
                                        class_names=iris.target_names,
                                        filled=True)

        # Draw graph
        graph1 = graphviz.Source(dot_data, format="png")
        graph1
```

Out[6]:



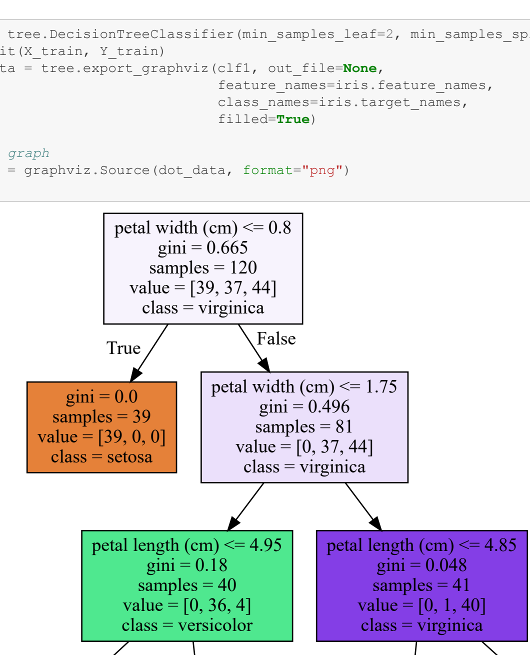

```
In [7]: clf1 = tree.DecisionTreeClassifier(min_samples_leaf=2, min_samples_split=5, max_depth = 2)
        clf1.fit(X_train, Y_train)
        dot_data = tree.export_graphviz(clf1, out_file=None,
                                        feature_names=iris.feature_names,
                                        class_names=iris.target_names,
                                        filled=True)

        # Draw graph
        graph1 = graphviz.Source(dot_data, format="png")
        graph1
```

Out[7]:



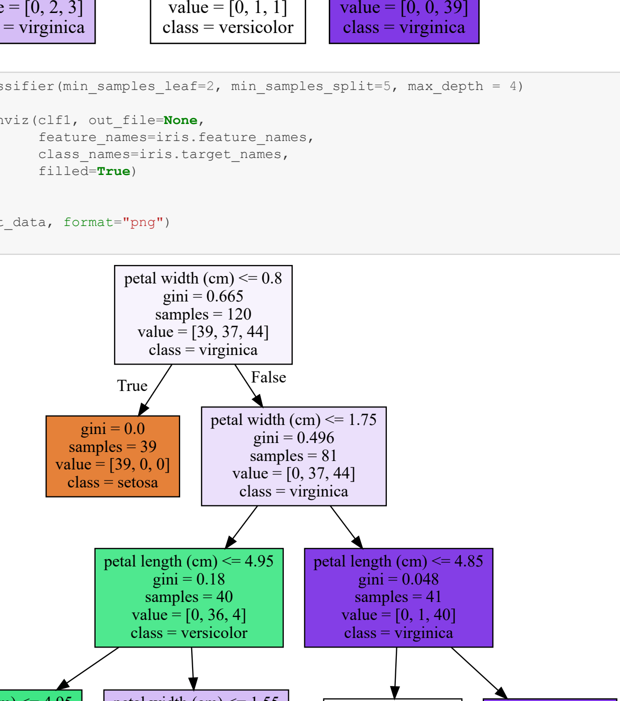

```
In [8]: clf1 = tree.DecisionTreeClassifier(min_samples_leaf=2, min_samples_split=5, max_depth = 3)
        clf1.fit(X_train, Y_train)
        dot_data = tree.export_graphviz(clf1, out_file=None,
                                        feature_names=iris.feature_names,
                                        class_names=iris.target_names,
                                        filled=True)

        # Draw graph
        graph1 = graphviz.Source(dot_data, format="png")
        graph1
```

Out[8]:



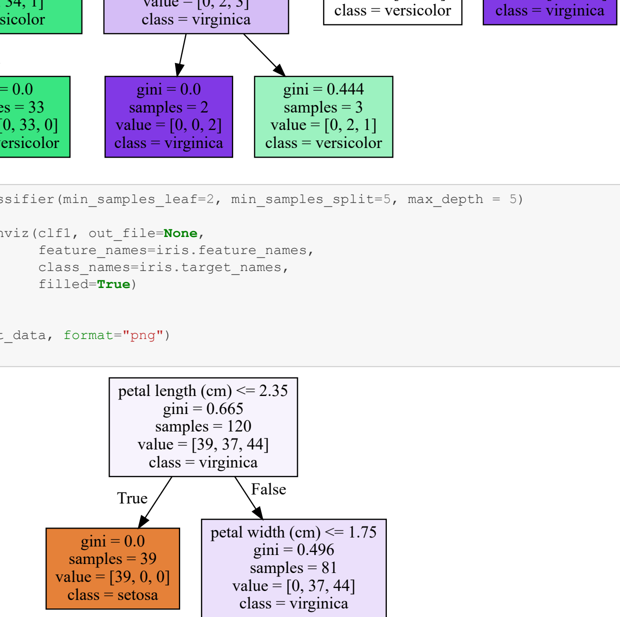

```
In [9]: clf1 = tree.DecisionTreeClassifier(min_samples_leaf=2, min_samples_split=5, max_depth = 4)
        clf1.fit(X_train, Y_train)
        dot_data = tree.export_graphviz(clf1, out_file=None,
                                        feature_names=iris.feature_names,
                                        class_names=iris.target_names,
                                        filled=True)

        # Draw graph
        graph1 = graphviz.Source(dot_data, format="png")
        graph1
```

Out[9]:




```
In [10]: clf1 = tree.DecisionTreeClassifier(min_samples_leaf=2, min_samples_split=5, max_depth = 5)
         clf1.fit(X_train, Y_train)
         dot_data = tree.export_graphviz(clf1, out_file=None,
                                         feature_names=iris.feature_names,
                                         class_names=iris.target_names,
                                         filled=True)

         # Draw graph
         graph1 = graphviz.Source(dot_data, format="png")
         graph1
```

Out[10]:



Conclusion:

- At depth 2,3,4,5, we get a highest recall value because we get highest "true positive rate" at these depths. Recall is calculated as the number of correct positive predictions (TP) divided by the total number of positives (TP + FN).
- At depth 1, we get a lowest precision. Precision is calculated as the number of correct positive predictions (TP) divided by the total number of positive predictions (TP + FP).
- At depth 2,3,4,5, we get best F1 score. F-score is a harmonic mean of precision and recall.

The differnce beween macro/micro/weighted average

- average = micro says the function to compute f1 by considering total true positives(TP), false negatives(FN) and false positives(FP) (no matter of the prediction for each label in the dataset)
- average = macro says the function to compute f1 for each label, and returns the average without considering the proportion for each label in the dataset.
- average = weighted says the function to compute f1 for each label, and returns the average considering the proportion for each label in the dataset.

Problem 2

```
In [11]: bc_file = "breast-cancer-wisconsin.data"
         column_names = ["CodeNumber","ClumpThickness", "UniformityOfCellSize", "UniformityOfCellShape","Margina
         lAdhesion",
                         "SingleEpithelialCellSize", "BareNuclei", "BlandChromatin", "NormalNucleoli", "Mitoses"
         , "Class"]
         bc_df = pd.read_csv(bc_file, names=column_names)
         bc_df
```

Out[11]:

	CodeNumber	ClumpThickness	UniformityOfCellSize	UniformityOfCellShape	MarginalAdhesion	SingleEpithelialCellSize	BareNuclei
0	1000025	5	1	1	1	2	1
1	1002945	5	4	4	5	7	10
2	1015425	3	1	1	1	2	2
3	1016277	6	8	8	1	3	4
4	1017023	4	1	1	3	2	1
...	...	...	...	...	...	...	...
694	776715	3	1	1	1	3	2
695	841769	2	1	1	1	2	1
696	888820	5	10	10	3	7	3
697	897471	4	8	6	4	3	4
698	897471	4	8	8	5	4	5

699 rows × 11 columns

```
In [12]: bc_df = bc_df.drop('CodeNumber',axis=1)
         bc_df
```

Out[12]:

	ClumpThickness	UniformityOfCellSize	UniformityOfCellShape	MarginalAdhesion	SingleEpithelialCellSize	BareNuclei	BlandChromatin
0	5	1	1	1	2	1	3
1	5	4	4	5	7	10	3
2	3	1	1	1	2	2	3
3	6	8	8	1	3	4	3
4	4	1	1	3	2	1	3
...	...	...	...	...	...	...	...
694	3	1	1	1	3	2	1
695	2	1	1	1	2	1	1
696	5	10	10	3	7	3	8
697	4	8	6	4	3	4	10
698	4	8	8	5	4	5	10

699 rows × 10 columns

```
In [13]: for name in bc_df.columns.values:
             ques_index = bc_df[bc_df[name] =='?'].index
             nan_index = bc_df[bc_df[name] == np.nan].index
             bc_df.drop(ques_index, inplace = True)
             bc_df.drop(nan_index, inplace = True)

         bc_df
```

```
C:\anaconda3\lib\site-packages\pandas\core\ops\array_ops.py:253: FutureWarning: elementwise compariso
n failed; returning scalar instead, but in the future will perform elementwise comparison
  res_values = method(rvalues)
```

Out[13]:

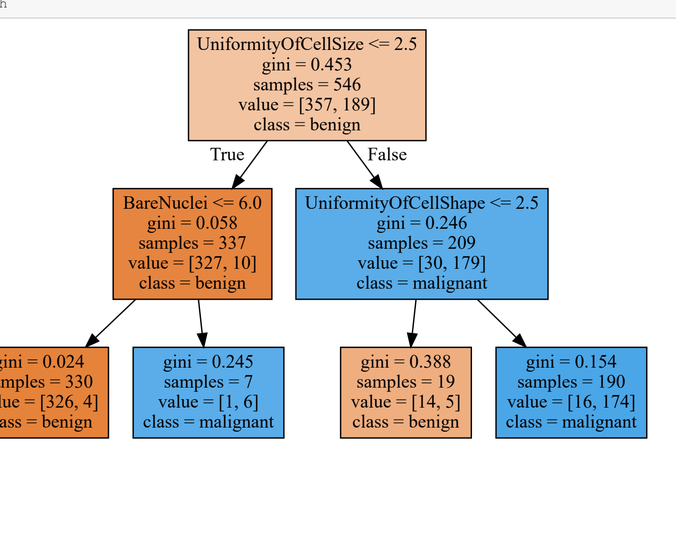
	ClumpThickness	UniformityOfCellSize	UniformityOfCellShape	MarginalAdhesion	SingleEpithelialCellSize	BareNuclei	BlandChromatin
0	5	1	1	1	2	1	3
1	5	4	4	5	7	10	3
2	3	1	1	1	2	2	3
3	6	8	8	1	3	4	3
4	4	1	1	3	2	1	3
...	...	...	...	...	...	...	...
694	3	1	1	1	3	2	1
695	2	1	1	1	2	1	1
696	5	10	10	3	7	3	8
697	4	8	6	4	3	4	10
698	4	8	8	5	4	5	10

683 rows × 10 columns

```
In [14]: feature = list(bc_df.columns[:9])
         x = bc_df[feature]
         y = bc_df["Class"]
         x_train, x_test, y_train, y_test = model_selection.train_test_split(x,
                                                                             y,
                                                                             random_state=0,
                                                                             test_size=0.2)
```

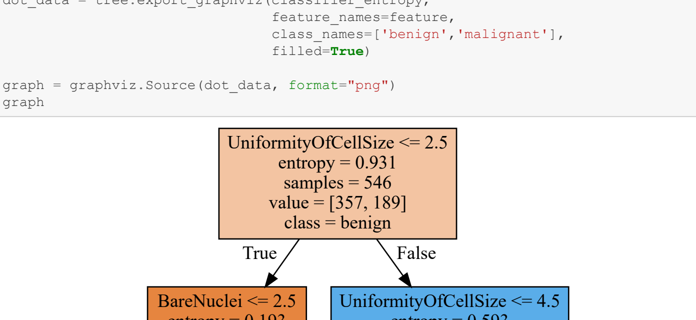
```
In [15]: # Graph with criterion="gini" and max_depth = 2
         classifier_gini = tree.DecisionTreeClassifier(criterion="gini", min_samples_leaf = 2, min_samples_split
          = 5, max_depth = 2)
         classifier_gini.fit(x_train, y_train)

         dot_data = tree.export_graphviz(classifier_gini,
                                         feature_names=feature,
                                         class_names=['benign','malignant'],
                                         filled=True)

         graph = graphviz.Source(dot_data, format="png")
         graph
```

Out[15]:



In [16]:
```python
# Graph with criterion="entropy" and max_depth = 2
classifier_entropy = tree.DecisionTreeClassifier(criterion="entropy", min_samples_leaf = 2, min_samples
_split = 5, max_depth = 2)
classifier_entropy.fit(x_train, y_train)

dot_data = tree.export_graphviz(classifier_entropy,
                                feature_names=feature,
                                class_names=['benign','malignant'],
                                filled=True)

graph = graphviz.Source(dot_data, format="png")
graph
```

Out[16]:



In [17]:
```python
classifier_gini = tree.DecisionTreeClassifier(criterion="gini", min_samples_leaf = 2, min_samples_split
 = 5, max_depth=1)
classifier_entropy = tree.DecisionTreeClassifier(criterion='entropy', min_samples_leaf = 2, min_samples
_split = 5, max_depth=1)
classifier_gini.fit(x_train, y_train)
classifier_entropy.fit(x_train, y_train)
pred_gini = classifier_gini.predict(x_test)
pred_entropy = classifier_entropy.predict(x_test)
print("Accuracy percent with gini criterion = {}".format(metrics.accuracy_score(y_test,pred_gini)*100))
print("Accuracy percent with entropy criterion = {}".format(metrics.accuracy_score(y_test,pred_entropy)
*100))
print("Misclassification Error percent with gini criterion = {}".format(100 - metrics.accuracy_score(y_
test,pred_gini)*100))
print("Misclassification Error percent with entropy criterion = {}".format(100 - metrics.accuracy_score
(y_test,pred_entropy)*100))
```

```
Accuracy percent with gini criterion = 92.7007299270073
Accuracy percent with entropy criterion = 92.7007299270073
Misclassification Error percent with gini criterion = 7.299270072992698
Misclassification Error percent with entropy criterion = 7.299270072992698
```

In [18]:
```python
cm = metrics.confusion_matrix(y_test, classifier_gini.predict(x_test))
print(cm)
TN, FP, FN, TP = metrics.confusion_matrix(y_test, classifier_gini.predict(x_test)).ravel()
```

```
[[79  8]
 [ 2 48]]
```

In [19]:
```python
print("TP = {}".format(TP))
print("FP = {}".format(FP))
print("FN = {}".format(FN))
print("TN = {}".format(TN))

# calculate accuracy
print((TP+TN)/float(TP+TN+FP+FN))
print(metrics.accuracy_score(y_test, classifier_gini.predict(x_test)))
```

```
TP = 48
FP = 8
FN = 2
TN = 79
0.927007299270073
0.927007299270073
```

We get same accuracy in both the case, mantually and using function.

In [20]:
```python
# calculate Misclassification Error
print((FP+FN)/float(TP+TN+FP+FN))
print(1 - metrics.accuracy_score(y_test, classifier_gini.predict(x_test)))
```

```
0.072992700729927
0.07299270072992703
```

We get same Misclassification Error in both the case, mantually and using function.

Conclusion:

- gini index at first split = 0.453
- entropy at first split = 0.931
- Misclassification Error at first split with gini criterion = 0.07299270072992698
- Misclassification Error at first split with entropy criterion = 0.07299270072992698
- Information gain

```
= parent entropy - average entropy of the children
= 0.931 - ((337/546)*0.193 + (209/546)*0.593)
= 0.585
```

- "Uniformity Of Cell Shape" feature is selected at first split.
- Decision boundary is <= 2.5

Problem 3

In [21]:
```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

In [22]:
```python
bc_file3 = "wdbc.data"
column_names3 = ["Id","Diagnosis",
                 "radius_mean","texture_mean","perimeter_mean","area_mean","smoothness_mean",
                 "compactness_mean","concavity_mean","concave points_mean","symmetry_mean","fractal_dim
ension_mean",
                 "radius_se","texture_se","perimeter_se","area_se","smoothness_se",
                 "compactness_se","concavity_se","concave points_se","symmetry_se","fractal_dimension_s
e",
                 "radius_worst","texture_worst","perimeter_worst","area_worst","smoothness_worst",
                 "compactness_worst","concavity_worst","concave points_worst","symmetry_worst","fractal
_dimension_worst"]

bc_df3 = pd.read_csv(bc_file3, names=column_names3)
bc_df3
```

Out[22]:

	Id	Diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_me
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.300
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.086
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.197
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.241
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.198
...	...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.243
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.144
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.092
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.351
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.000

569 rows × 32 columns

In [23]:
```python
bc_df3['Diagnosis'] = bc_df3["Diagnosis"].apply(lambda x: 1 if x == 'M' else 0)
bc_df3.drop(['Id'], axis=1, inplace=True)
bc_df3
```

Out[23]:

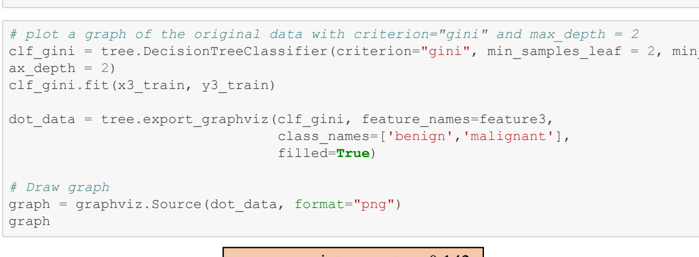
	Diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	cc points
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0
...	...	...	...	...	...	...	...	...	
564	1	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0
565	1	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0
566	1	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0
567	1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0
568	0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0

569 rows × 31 columns

In [24]:
```python
# split the original data
feature3 = list(bc_df3.columns[1:])
x3 = bc_df3[feature3]
y3 = bc_df3['Diagnosis']
x3_train, x3_test, y3_train, y3_test = model_selection.train_test_split(x3,
                                                                        y3,
                                                                        test_size=0.2,
                                                                        random_state=0)
```

In [25]:
```python
# plot a graph of the original data with criterion="gini" and max_depth = 2
clf_gini = tree.DecisionTreeClassifier(criterion="gini", min_samples_leaf = 2, min_samples_split = 5, m
ax_depth = 2)
clf_gini.fit(x3_train, y3_train)

dot_data = tree.export_graphviz(clf_gini, feature_names=feature3,
                                class_names=['benign','malignant'],
                                filled=True)

# Draw graph
graph = graphviz.Source(dot_data, format="png")
graph
```

Out[25]:



In [26]:
```python
e = y3_test
p = clf_gini.predict(x3_test)
```

In [27]:
```python
# Precision, Recall and F1 Score for the original data
print("Precision, Recall and F1 Score with macro = {}".format(metrics.precision_recall_fscore_support(e
,p,average='macro')))
print("Precision, Recall and F1 Score with micro = {}".format(metrics.precision_recall_fscore_support(e
,p,average='micro')))
print("Precision, Recall and F1 Score with weighted = {}".format(metrics.precision_recall_fscore_suppor
t(e,p,average='weighted')))
print(metrics.classification_report(e,p))
```

```
Precision, Recall and F1 Score with macro = (0.9671497584541062, 0.9606224198158145, 0.96355498721227
62, None)
Precision, Recall and F1 Score with micro = (0.9649122807017544, 0.9649122807017544, 0.96491228070175
44, None)
Precision, Recall and F1 Score with weighted = (0.9652851936604796, 0.9649122807017544, 0.96478889038
45291, None)
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        67
           1       0.98      0.94      0.96        47

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

In [28]:
```python
cm = metrics.confusion_matrix(e,p)
print(cm)
TN, FP, FN, TP = metrics.confusion_matrix(e,p).ravel()
```

```
[[66  1]
 [ 3 44]]
```

In [29]:
```python
print("TP = {}".format(TP))
print("FP = {}".format(FP))
print("FN = {}".format(FN))
print("TN = {}".format(TN))
print("TPR = {}".format(TP/float(TP+FP)))
print("FPR = {}".format(FP/float(TN+FP)))
```

```
TP = 44
FP = 1
FN = 3
TN = 66
TPR = 0.9777777777777777
FPR = 0.014925373134328358
```

In [30]:
```python
# Scale the data
scaler = StandardScaler()
scaler.fit(bc_df3.values)
scaled_data = scaler.transform(bc_df3.values)
```

In [31]:
```python
# Apply PCA for n_components=1
pca_1 = PCA(n_components=1)
pca_1.fit(scaled_data)
pca_1_transformed = pca_1.transform(scaled_data)
```

In [32]:
```python
pcadf_1 = pd.DataFrame(pca_1_transformed, columns = ['PC1'])
pcadf_1
```

Out[32]:

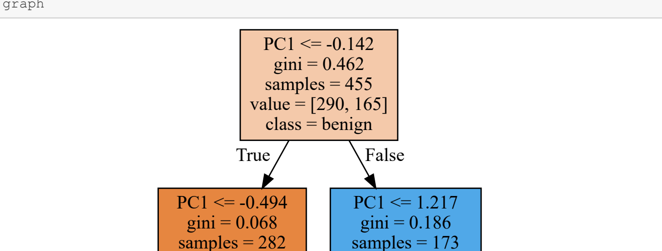
	PC1
0	9.225770
1	2.655802
2	5.892492
3	7.135401
4	4.129423
...	...
564	6.593983
565	4.024833
566	1.530077
567	10.405008
568	-5.504862

569 rows × 1 columns

In [33]:
```python
feature3 = list(pcadf_1.columns)
x3 = pcadf_1[feature3]
y3 = bc_df3['Diagnosis']

x3_train, x3_test, y3_train, y3_test = model_selection.train_test_split(x3,
                                                                        y3,
                                                                        test_size=0.2,
                                                                        random_state=0)
```

In [34]:
```python
clf_gini = tree.DecisionTreeClassifier(criterion="gini", min_samples_leaf = 2, min_samples_split = 5, m
ax_depth = 2)
clf_gini.fit(x3_train, y3_train)

dot_data = tree.export_graphviz(clf_gini, feature_names=feature3,
                                class_names=['benign','malignant'],
                                filled=True)

graph = graphviz.Source(dot_data, format="png")
graph
```

Out[34]:



In [35]:
```python
e = y3_test
p = clf_gini.predict(x3_test)
```

In [36]:
```python
print("Precision, Recall and F1 Score with macro = {}".format(metrics.precision_recall_fscore_support(e
,p,average='macro')))
print("Precision, Recall and F1 Score with micro = {}".format(metrics.precision_recall_fscore_support(e
,p,average='micro')))
print("Precision, Recall and F1 Score with weighted = {}".format(metrics.precision_recall_fscore_suppor
t(e,p,average='weighted')))
print(metrics.classification_report(e,p))
```

```
Precision, Recall and F1 Score with macro = (0.8812034739454093, 0.8902826294061607, 0.88395583744420
95, None)
Precision, Recall and F1 Score with micro = (0.8859649122807017, 0.8859649122807017, 0.88596491228070
17, None)
Precision, Recall and F1 Score with weighted = (0.890726350615994, 0.8859649122807017, 0.886634603892
8657, None)
              precision    recall  f1-score   support

           0       0.94      0.87      0.90        67
           1       0.83      0.91      0.87        47

    accuracy                           0.89       114
   macro avg       0.88      0.89      0.88       114
weighted avg       0.89      0.89      0.89       114
```

In [37]:
```python
cm = metrics.confusion_matrix(e,p)
print(cm)
TN, FP, FN, TP = metrics.confusion_matrix(e,p).ravel()
```

```
[[58  9]
 [ 4 43]]
```

In [38]:
```python
print("TP = {}".format(TP))
print("FP = {}".format(FP))
print("FN = {}".format(FN))
print("TN = {}".format(TN))
print("TPR = {}".format(TP/float(TP+FP)))
print("FPR = {}".format(FP/float(TN+FP)))
```

```
TP = 43
FP = 9
FN = 4
TN = 58
TPR = 0.8269230769230769
FPR = 0.13432835820895522
```

In [39]:
```python
pca_2 = PCA(n_components=2)
pca_2.fit(scaled_data)
pca_2_transformed = pca_2.transform(scaled_data)
```

In [40]:
```python
pcadf_2 = pd.DataFrame(pca_2_transformed, columns = ['PC1','PC2'])
pcadf_2
```

Out[40]:

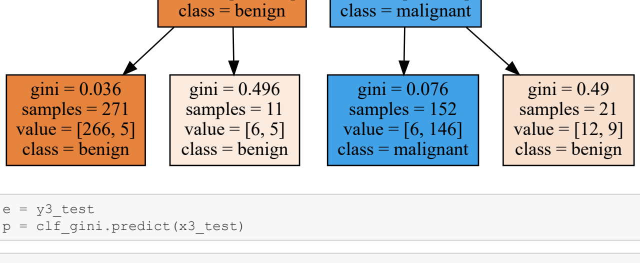
	PC1	PC2
0	9.225770	2.116196
1	2.655802	-3.784776
2	5.892492	-1.005579
3	7.135401	10.318716
4	4.129423	-1.905579
...	...	...
564	6.593983	-3.454947
565	4.024833	-3.556006
566	1.530077	-1.958871
567	10.405008	1.849078
568	-5.504862	-0.766348

569 rows × 2 columns

In [41]:
```python
feature3 = list(pcadf_2.columns)
x3 = pcadf_2[feature3]
y3 = bc_df3['Diagnosis']
x3_train, x3_test, y3_train, y3_test = model_selection.train_test_split(x3,
                                                                        y3,
                                                                        test_size=0.2,
                                                                        random_state=0)
```

In [42]:
```python
clf_gini = tree.DecisionTreeClassifier(criterion="gini", min_samples_leaf = 2, min_samples_split = 5, m
ax_depth = 2)
clf_gini.fit(x3_train, y3_train)

dot_data = tree.export_graphviz(clf_gini, feature_names=feature3,
                                class_names=['benign','malignant'],
                                filled=True)

graph = graphviz.Source(dot_data, format="png")
graph
```

Out[42]:



In [43]:
```python
e = y3_test
p = clf_gini.predict(x3_test)
```

In [44]:
```python
print("Precision, Recall and F1 Score with macro = {}".format(metrics.precision_recall_fscore_support(e
,p,average='macro')))
print("Precision, Recall and F1 Score with micro = {}".format(metrics.precision_recall_fscore_support(e
,p,average='micro')))
print("Precision, Recall and F1 Score with weighted = {}".format(metrics.precision_recall_fscore_suppor
t(e,p,average='weighted')))
print(metrics.classification_report(e,p))
```

```
Precision, Recall and F1 Score with macro = (0.8650895140664961, 0.8626548110511274, 0.86379928315412
19, None)
Precision, Recall and F1 Score with micro = (0.868421052631579, 0.868421052631579, 0.868421052631579,
None)
Precision, Recall and F1 Score with weighted = (0.8681181854892988, 0.868421052631579, 0.868200968370
7477, None)
              precision    recall  f1-score   support

           0       0.88      0.90      0.89        67
           1       0.85      0.83      0.84        47

    accuracy                           0.87       114
   macro avg       0.87      0.86      0.86       114
weighted avg       0.87      0.87      0.87       114
```

In [45]:
```python
cm = metrics.confusion_matrix(e,p)
print(cm)
TN, FP, FN, TP = metrics.confusion_matrix(e,p).ravel()
```

```
[[60  7]
 [ 8 39]]
```

In [46]:
```python
print("TP = {}".format(TP))
print("FP = {}".format(FP))
print("FN = {}".format(FN))
print("TN = {}".format(TN))
print("TPR = {}".format(TP/float(TP+FP)))
print("FPR = {}".format(FP/float(TN+FP)))
```

```
TP = 39
FP = 7
FN = 8
TN = 60
TPR = 0.8478260869565217
FPR = 0.1044776119402985
```

Conclusion :

For original data:

- Precision, Recall and F1 Score with macro = (0.9671497584541062, 0.9606224198158145, 0.9635549872122762, None)
- Precision, Recall and F1 Score with micro = (0.9649122807017544, 0.9649122807017544, 0.9649122807017544, None)
- Precision, Recall and F1 Score with weighted = (0.9652851936604796, 0.9649122807017544, 0.9647888903845291, None)
- TP = 44
- FP = 1
- FN = 3
- TN = 66
- TPR = 0.9777777777777777
- FPR = 0.014925373134328358

After applying PCA with n_component=1:

- Precision, Recall and F1 Score with macro = (0.8812034739454093, 0.8902826294061607, 0.8839558374442095, None)
- Precision, Recall and F1 Score with micro = (0.8859649122807017, 0.8859649122807017, 0.8859649122807017, None)
- Precision, Recall and F1 Score with weighted = (0.890726350615994, 0.8859649122807017, 0.8866346038928657, None)
- TP = 43
- FP = 9
- FN = 4
- TN = 58
- TPR = 0.8269230769230769
- FPR = 0.13432835820895522

After applying PCA with n_component=2:

- Precision, Recall and F1 Score with macro = (0.8650895140664961, 0.8626548110511274, 0.8637992831541219, None)
- Precision, Recall and F1 Score with micro = (0.868421052631579, 0.868421052631579, 0.868421052631579, None)
- Precision, Recall and F1 Score with weighted = (0.8681181854892988, 0.868421052631579, 0.8682009683707477, None)
- TP = 39
- FP = 7
- FN = 8
- TN = 60
- TPR = 0.8478260869565217
- FPR = 0.1044776119402985

Decision trees work with continuous variables same as PCA works.

For example, consider age as the target variable. So, you compute the variance of age and it comes out to be x. Next, decision tree looks at various splits and calculates the total weighted variance of each of these splits. It chooses the split which provides the minimum variance (Greedy approach).

TPR value is increasing and FPR value is decreasing as the number of components retained by PCA also increased.

Problem 4

In [47]:
```python
A = np.random.normal(5, 2, 500)
B = np.random.normal(-5, 2, 500)
```

In [48]:
```python
C1 = np.repeat(0, 500)
C2 = np.repeat(1, 500)
```

In [49]:
```python
dfA = pd.DataFrame(zip(A, C1), columns=["Feature", "Class"])
dfB = pd.DataFrame(zip(B, C2), columns=["Feature", "Class"])
```

In [50]:
```python
dfC = pd.concat([dfA, dfB])
dfC
```

Out[50]:

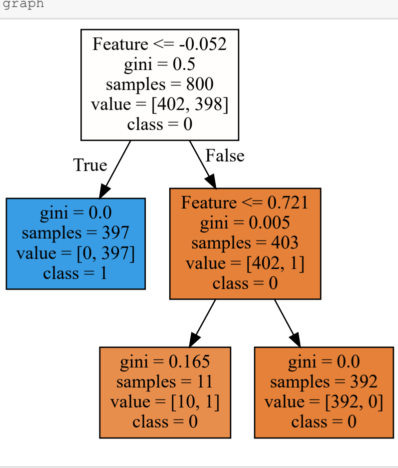
	Feature	Class
0	6.389558	0
1	3.656043	0
2	2.400999	0
3	2.856421	0
4	2.383818	0
...	...	...
495	-4.060165	1
496	-2.792871	1
497	-7.135381	1
498	-3.415122	1
499	-9.433261	1

1000 rows × 2 columns

In [51]:
```python
feature4 = list(dfC.columns[:1])
# feature4 = list(df.iloc[:, 0])
x4 = dfC[feature4]
y4 = dfC['Class']
x4_train, x4_test, y4_train, y4_test = model_selection.train_test_split(x4,
                                                                        y4,
                                                                        test_size=0.2,
                                                                        random_state=0)
```

In [52]:
```python
clfgini = tree.DecisionTreeClassifier(criterion="gini", max_depth = 2)
clfgini.fit(x4_train, y4_train)

dot_data = tree.export_graphviz(clfgini, feature_names=feature4,
                                class_names=['0','1'],
                                filled=True)

# Draw graph
graph = graphviz.Source(dot_data, format="png")
graph
```

Out[52]:



In [53]:
```python
dfC.describe()
```

Out[53]:

	Feature	Class
count	1000.000000	1000.00000
mean	-0.058940	0.50000
std	5.466306	0.50025
min	-10.300019	0.00000
25%	-5.173956	0.00000
50%	-0.052058	0.50000
75%	5.019240	1.00000
max	11.292439	1.00000

Conclusion:

Threshold value for the feature in the first split : <= -0.052

In [ ]: