```
# Library yang digunakan
from sklearn.datasets import load_breast_cancer
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.semi_supervised import SelfTrainingClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split
```

```
#Data Collection
#Load Data
data = load_breast_cancer()

#Show Core Information
print('Feature   :', data.feature_names)
print('Class     :', data.target_names)
print('Size Data :', len(data.target))
```

```
Feature   : ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
Class     : ['malignant' 'benign']
Size Data : 569
```

```
# Transform ke pandas Data Frame
df = pd.DataFrame(data.data, columns = data.feature_names)
df['target'] = ['malignant' if target == 0 else 'benign' for target in data.target] # Tambahkan column target

# Tampilkan 5 data teratas
df.head()
```

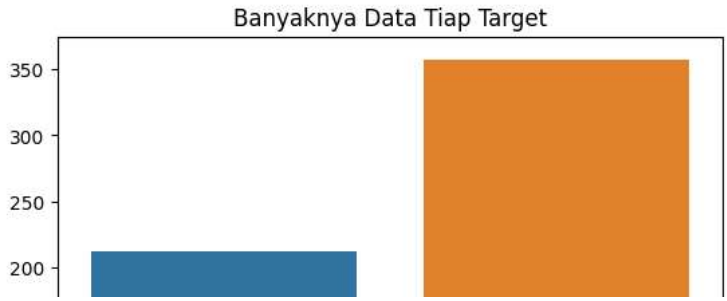
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883

5 rows × 31 columns

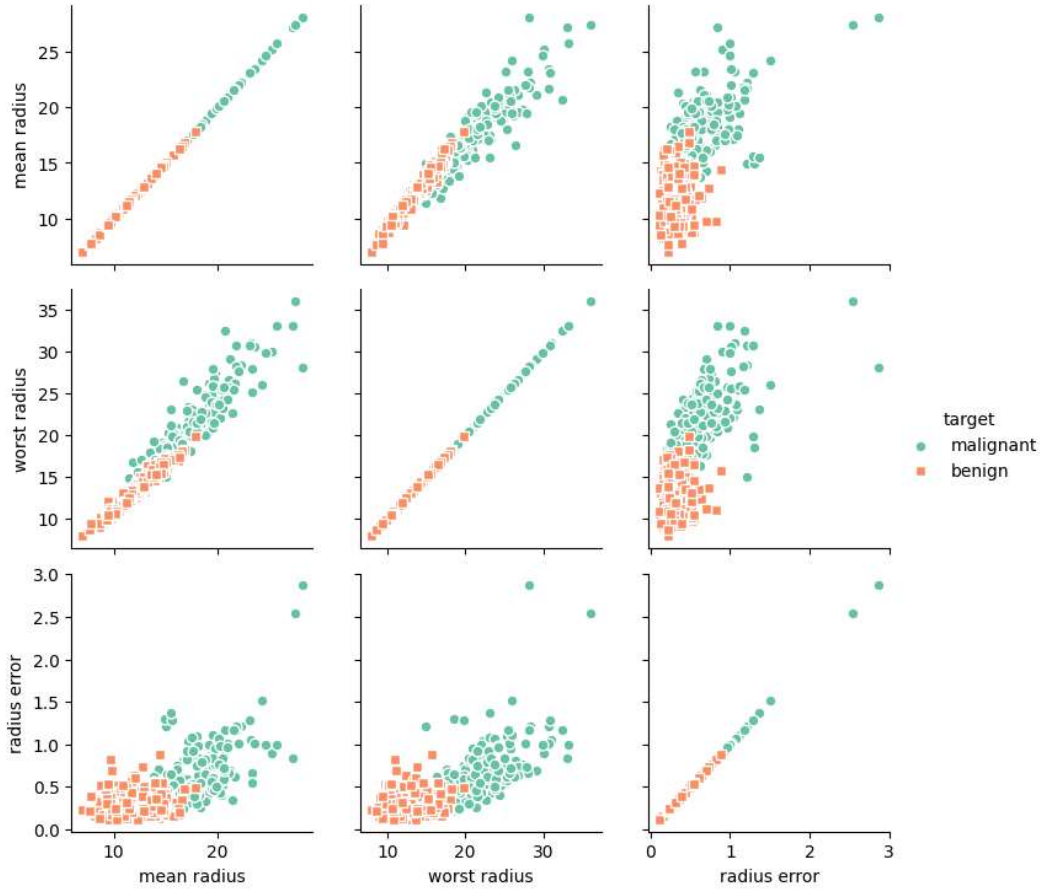
```
# Pisahkan data masing - masing target
malignant = df[df['target'] == 'malignant']
benign = df[df['target'] == 'benign']
```

```
# Visualize banyaknya data tiap target
sns.barplot(x = ['malignant', 'benign'], y = [len(malignant), len(benign)])

plt.title('Banyaknya Data Tiap Target')
plt.show()
```

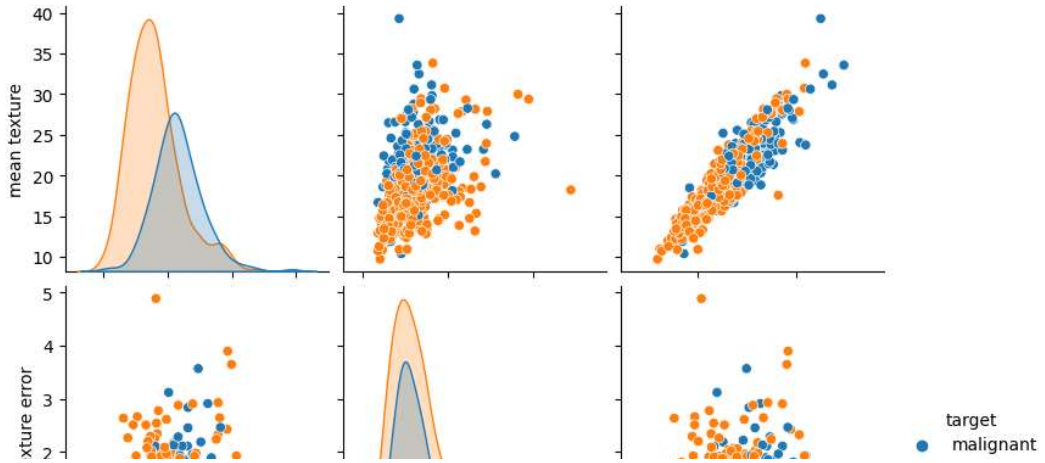



```
#Dataset terbagi menjadi tiga bagian, [mean, error, worst]
#Visualization beberapa feature dari masing - masing bagian
g = sns.PairGrid(df[['mean radius', 'worst radius', 'radius error', 'target']], hue="target", palette="Set2", hue_kws={"marker": ["o", "s
g = g.map(plt.scatter, linewidths=1, edgecolor="w", s=40)
g = g.add_legend()
```



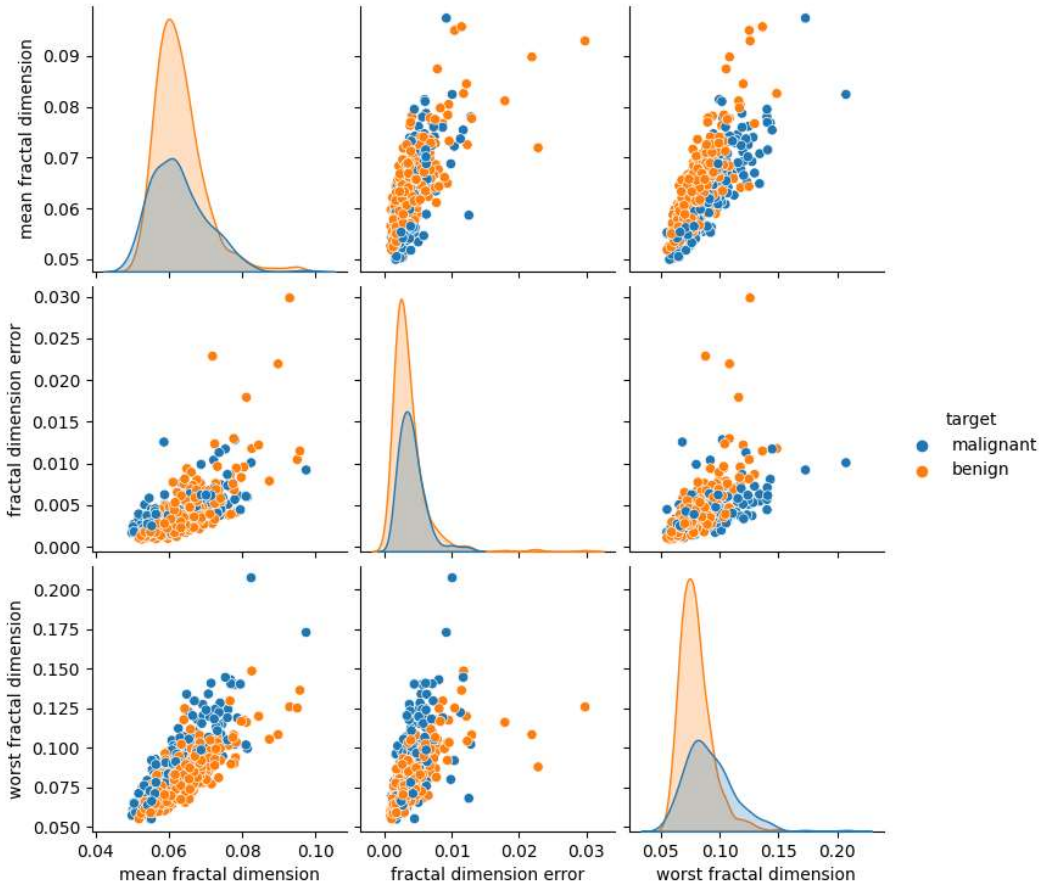

```
sns.pairplot(df[['mean texture', 'texture error', 'worst texture', 'target']], hue='target')
```

```
<seaborn.axisgrid.PairGrid at 0x7fc6db7b1a20>
```




```
sns.pairplot(df[['mean fractal dimension', 'fractal dimension error', 'worst fractal dimension', 'target']], hue='target')
```

```
<seaborn.axisgrid.PairGrid at 0x7fc6dbbf42e0>
```

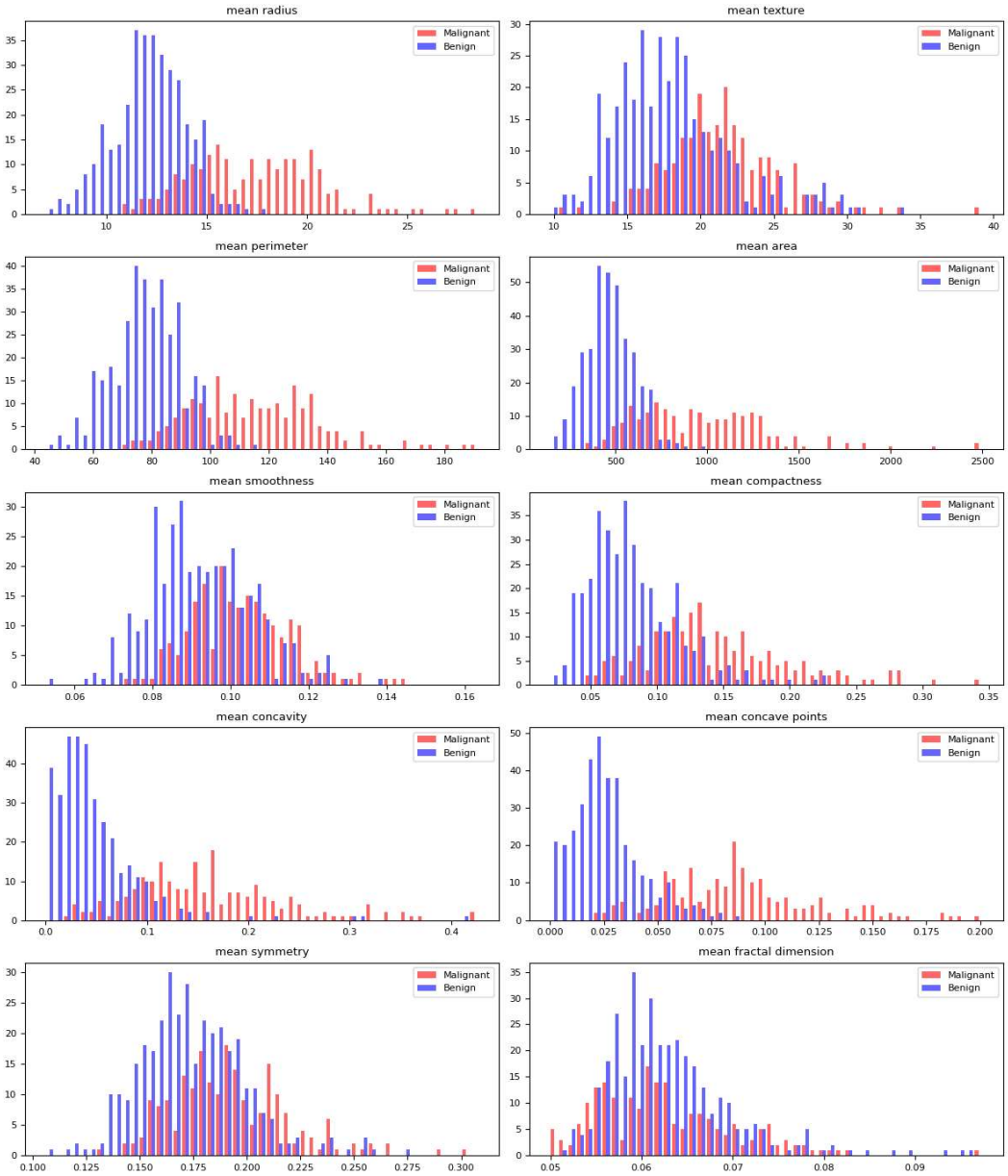



```
mean_column = list(df.columns[0:10])


plt.rcParams.update({'font.size': 8})
plot, graphs = plt.subplots(nrows=5, ncols=2, figsize=(12,14))
graphs = graphs.flatten()

for idx, graph in enumerate(graphs):
    graph.figure

    binwidth= (max(df[mean_column[idx]]) - min(df[mean_column[idx]]))/50
    bins = np.arange(min(df[mean_column[idx]]), max(df[mean_column[idx]]) + binwidth, binwidth)
    graph.hist([malignant[mean_column[idx]], benign[mean_column[idx]]], bins=bins, alpha=0.6, label=['Malignant','Benign'], color=['red',
    graph.legend(loc='upper right')
    graph.set_title(mean_column[idx])
plt.tight_layout()
```

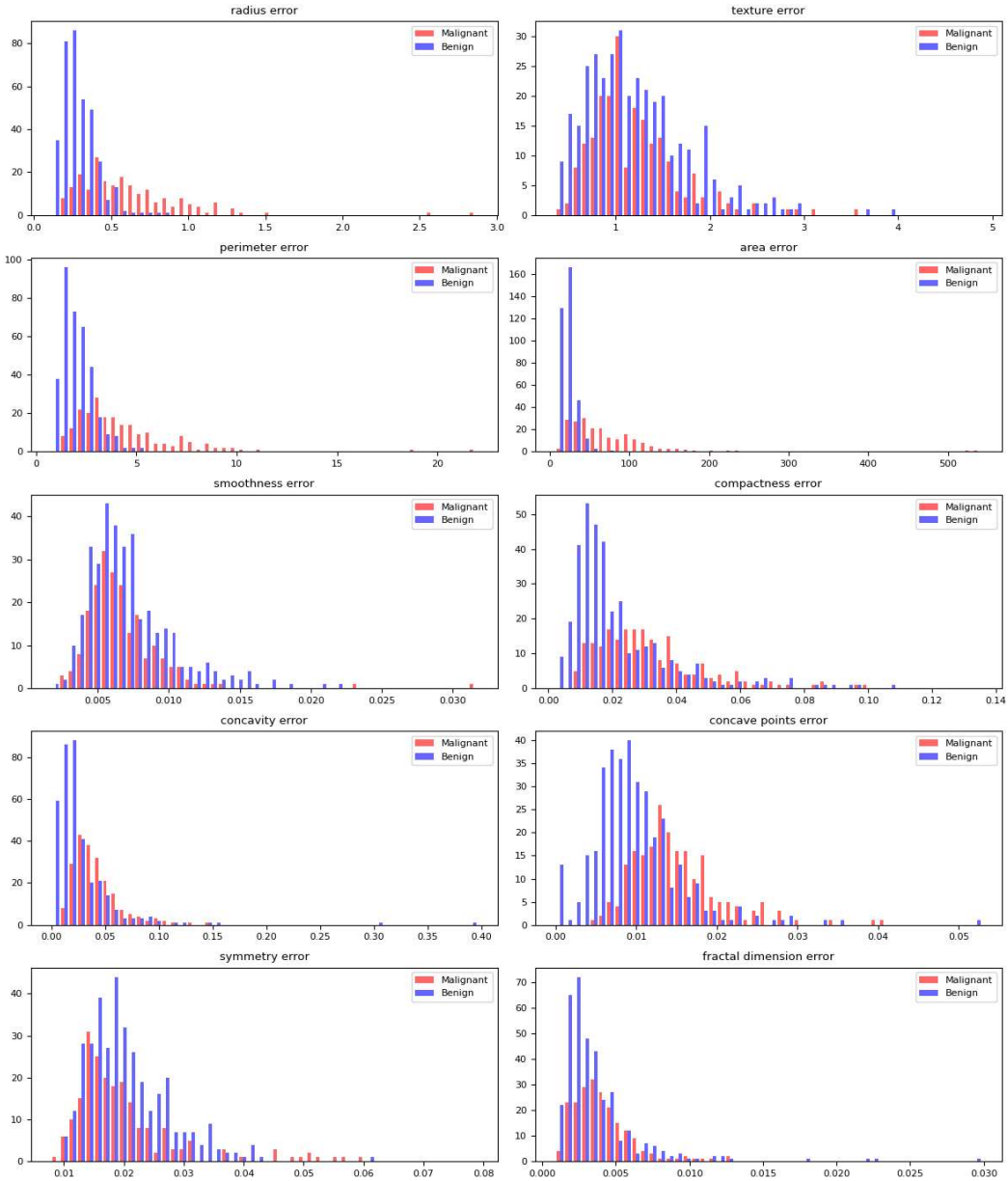



```
error_column = list(df.columns[10:20])


plt.rcParams.update({'font.size': 8})
plot, graphs = plt.subplots(nrows=5, ncols=2, figsize=(12,14))
graphs = graphs.flatten()

for idx, graph in enumerate(graphs):
    graph.figure
```

```
    binwidth= (max(df[error_column[idx]]) - min(df[error_column[idx]]))/50
    bins = np.arange(min(df[error_column[idx]]), max(df[error_column[idx]]) + binwidth, binwidth)
    graph.hist([malignant[error_column[idx]], benign[error_column[idx]]], bins=bins, alpha=0.6, label=['Malignant','Benign'], color=['rec
    graph.legend(loc='upper right')
    graph.set_title(error_column[idx])
plt.tight_layout()
```

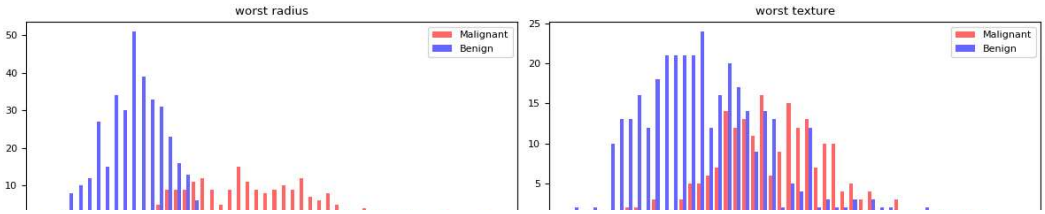


```
worst_column = list(df.columns[20:30])


plt.rcParams.update({'font.size': 8})
plot, graphs = plt.subplots(nrows=5, ncols=2, figsize=(12,14))
graphs = graphs.flatten()

for idx, graph in enumerate(graphs):
    graph.figure

    binwidth= (max(df[worst_column[idx]]) - min(df[worst_column[idx]]))/50
    bins = np.arange(min(df[worst_column[idx]]), max(df[worst_column[idx]]) + binwidth, binwidth)
    graph.hist([malignant[worst_column[idx]], benign[worst_column[idx]]], bins=bins, alpha=0.6, label=['Malignant','Benign'], color=['rec
    graph.legend(loc='upper right')
    graph.set_title(worst_column[idx])
plt.tight_layout()
```

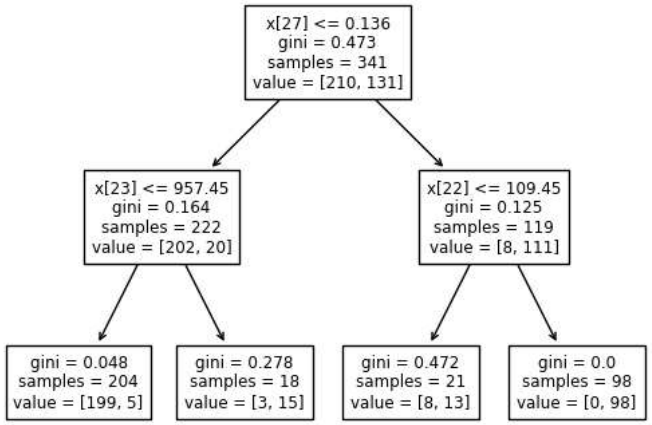



```
# Data Preparation
# Pisahkan features dengan targets
x = df.iloc[:, 0:30] # features
y = df.iloc[:, -1] # targets
```

```
# Split data dengan ketentuan 60% train 40% test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4)
```

```
# Model
# Decision Tree
decision_tree = DecisionTreeClassifier(random_state = 0, max_depth = 2) # max_depth 2 paling optimal dari 3, 5, dan default
decision_tree = decision_tree.fit(x_train, y_train) # train dtree
tree.plot_tree(decision_tree) # plot dtree
```

```
[Text(0.5, 0.8333333333333334, 'x[27] <= 0.136\ngini = 0.473\nsamples = 341\nvalue = [210, 131]'),
 Text(0.25, 0.5, 'x[23] <= 957.45\ngini = 0.164\nsamples = 222\nvalue = [202, 20]'),
 Text(0.125, 0.16666666666666666, 'gini = 0.048\nsamples = 204\nvalue = [199, 5]'),
 Text(0.375, 0.16666666666666666, 'gini = 0.278\nsamples = 18\nvalue = [3, 15]'),
 Text(0.75, 0.5, 'x[22] <= 109.45\ngini = 0.125\nsamples = 119\nvalue = [8, 111]'),
 Text(0.625, 0.16666666666666666, 'gini = 0.472\nsamples = 21\nvalue = [8, 13]'),
 Text(0.875, 0.16666666666666666, 'gini = 0.0\nsamples = 98\nvalue = [0, 98]')]
```




```
# Hitung score akurasi
score = decision_tree.score(x_test, y_test)
print('Akurasi :',score)
```

```
Akurasi : 0.9298245614035088
```

```
# Random Forest
random_forest = RandomForestClassifier(random_state = 0, max_depth = 4) # Optimal di max_depth 4
random_forest.fit(x_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=4, random_state=0)
```

```
# Hitung score akurasi
score = random_forest.score(x_test, y_test)
print('Akurasi :',score)
```

```
Akurasi : 0.9736842105263158
```

```
# Self Training
svc = DecisionTreeClassifier(random_state = 0, max_depth = 2)
self_training = SelfTrainingClassifier(svc)
self_training.fit(x_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/semi_supervised/_self_training.py:212: UserWarning: y c
  warnings.warn("y contains no unlabeled samples", UserWarning)
```

```
▸ SelfTrainingClassifier
▸ base_estimator: DecisionTreeClassifier
```

```
# Hitung score akurasi
score = self_training.score(x_test, y_test)
print('Akurasi :',score)
```

```
Akurasi : 0.9298245614035088
```