```
# 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
```

```
#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()
```

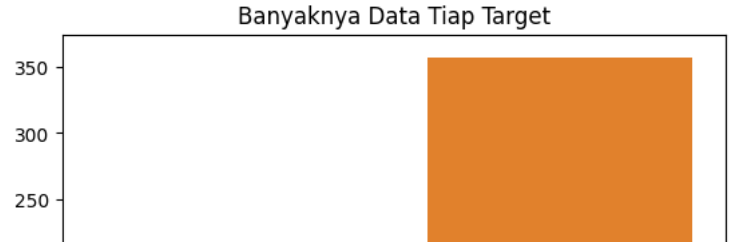
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	cor pc
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.

5 rows × 31 columns

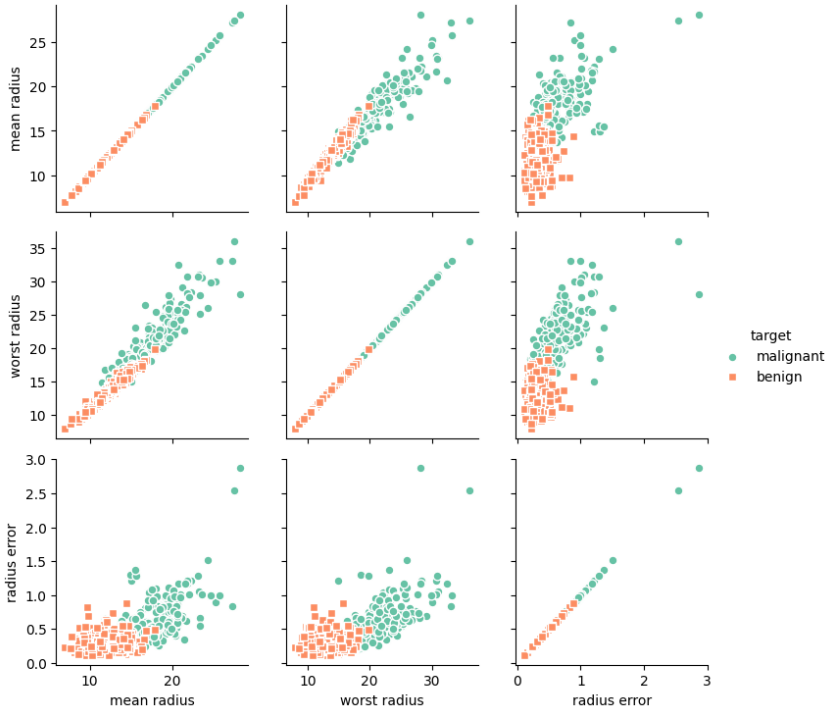
```
# Pisahkan data masing - masing target
malignant = df[df['target'] == 'malignant']
benign = df[df['target'] == 'benign']
```

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

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

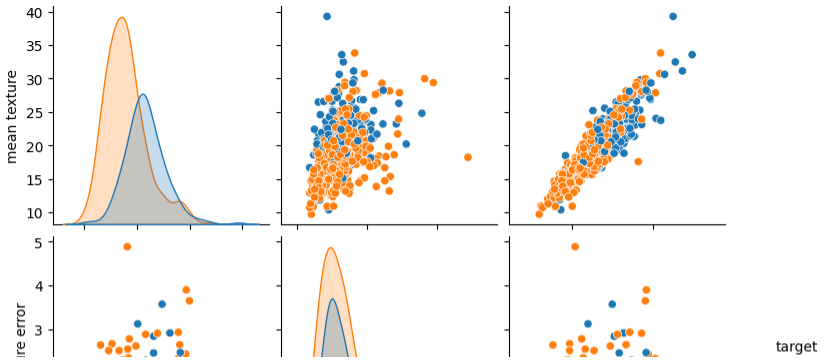



```
g = sns.PairGrid(df[['mean radius', 'worst radius', 'radius error', 'target']], hue="target", palette="Set2", hue_kws={"marke
g = g.map(plt.scatter, linewidths=1, edgecolor="w", s=40)
g = g.add_legend()
```



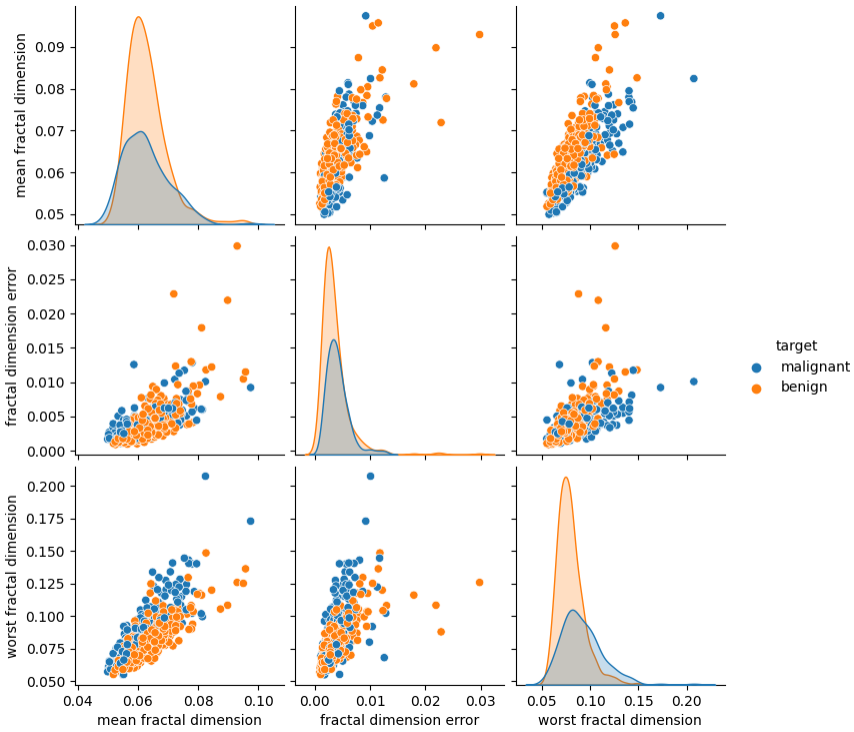

```
sns.pairplot(df[['mean texture', 'texture error', 'worst texture', 'target']], hue='target')
```

```
<seaborn.axisgrid.PairGrid at 0x7f6218b74910>
```




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

```
<seaborn.axisgrid.PairGrid at 0x7f6218b74fd0>
```

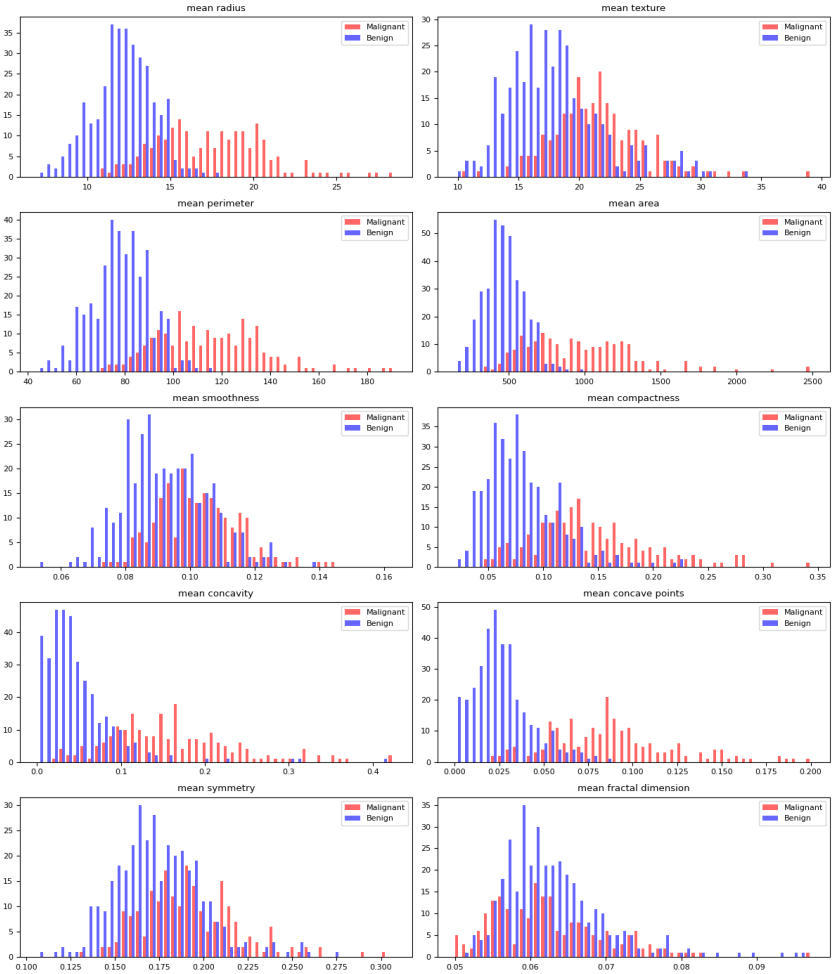



```
mean_column = list(df.columns[0:10])


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'], c
    graph.legend(loc='upper right')
    graph.set_title(mean_column[idx])
plt.tight_layout()
```



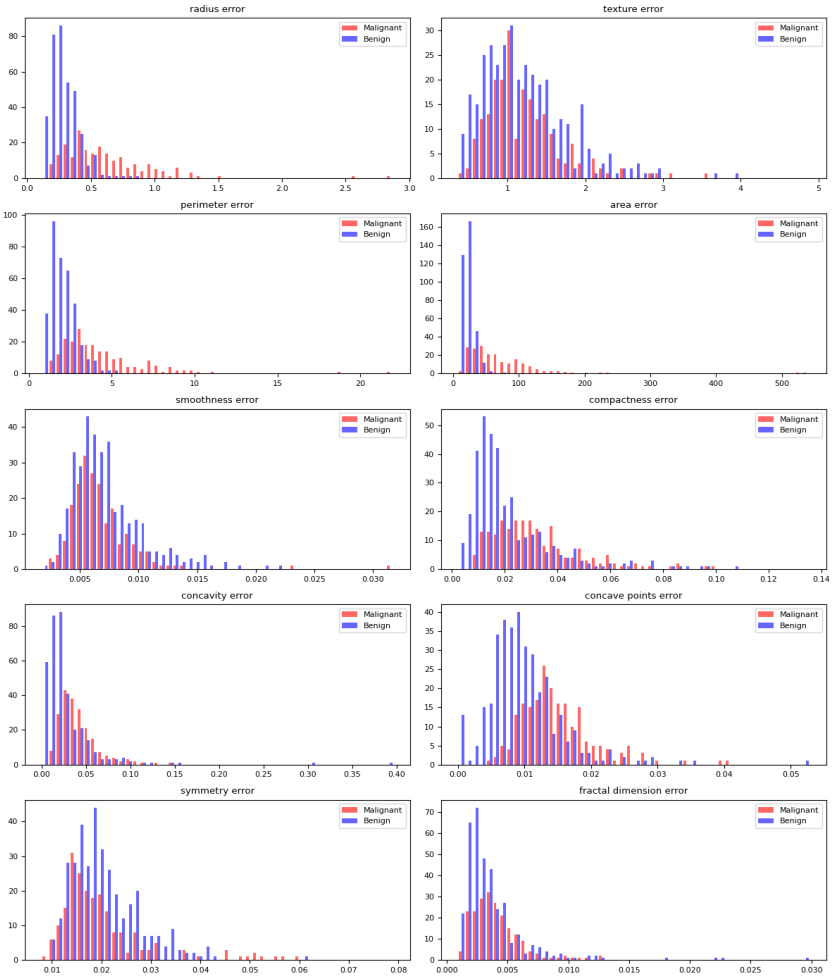

```
error_column = list(df.columns[10:20])


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'],
    graph.legend(loc='upper right')
```

```
    graph.set_title(error_column[idx])
plt.tight_layout()
```

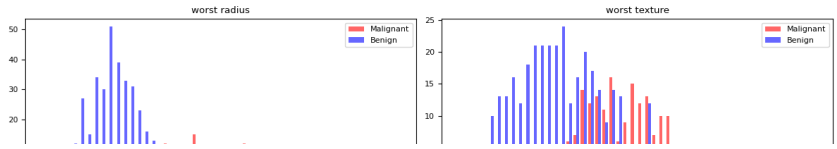



```
worst_column = list(df.columns[20:30])
```

```
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'],
    graph.legend(loc='upper right')
    graph.set_title(worst_column[idx])
plt.tight_layout()
```

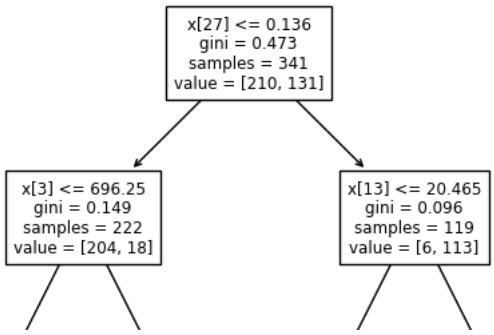



```
# Pisahkan features dengan targets
x = df.iloc[:, 0:30] # features
y = df.iloc[:, -1] # targets
```

```
# 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)
```

```
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[3] <= 696.25\ngini = 0.149\nsamples = 222\nvalue =
[204, 18]'),
 Text(0.125, 0.16666666666666666, 'gini = 0.029\nsamples = 203\nvalue =
[200, 3]'),
 Text(0.375, 0.16666666666666666, 'gini = 0.332\nsamples = 19\nvalue = [4,
15]'),
 Text(0.75, 0.5, 'x[13] <= 20.465\ngini = 0.096\nsamples = 119\nvalue =
[6, 113]'),
 Text(0.625, 0.16666666666666666, 'gini = 0.469\nsamples = 8\nvalue = [5,
3]'),
 Text(0.875, 0.16666666666666666, 'gini = 0.018\nsamples = 111\nvalue =
[1, 110]')]
```




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

```
Akurasi : 0.9078947368421053
```

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

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

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

```
Akurasi : 0.9429824561403509
```

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

```
/usr/local/lib/python3.9/dist-packages/sklearn/semi_supervised/_self_traini
  warnings.warn("y contains no unlabeled samples", UserWarning)
```

```
SelfTrainingClassifier(base_estimator=DecisionTreeClassifier())
```

```
svc = DecisionTreeClassifier(random_state = 0, max_depth = 2)
self_training = SelfTrainingClassifier(svc)
```