MM20B007 Tutorial 6

```
# Import libraries and load the dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_curve, auc
from sklearn.datasets import make_moons
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.model_selection import cross_val_score

url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/breast-
cancer-wisconsin/wdbc.data"
names = ["id", "diagnosis", "radius_mean", "texture_mean",
"perimeter_mean", "area_mean", "smoothness_mean", "compactness_mean",
"concavity_mean", "concave points_mean", "symmetry_mean",
"fractal_dimension_mean", "radius_se", "texture_se", "perimeter_se",
"area_se", "smoothness_se", "compactness_se", "concavity_se", "concave
points_se", "symmetry_se", "fractal_dimension_se",
"radius_worst", "texture_worst", "perimeter_worst", "area_worst",
"smoothness_worst", "compactness_worst", "concavity_worst",
"concave points_worst", "symmetry_worst", "fractal_dimension_worst"]

data = pd.read_csv(url, names=names)
print('Shape of give data: ', data.shape)
# Encode the diagnosis
data.head()
```

```
Shape of give data:  (569, 32)
```

```
         id diagnosis  radius_mean  texture_mean  perimeter_mean
area_mean  \
0    842302         M        17.99         10.38          122.80
1001.0
1    842517         M        20.57         17.77          132.90
1326.0
2  84300903         M        19.69         21.25          130.00
1203.0
3  84348301         M        11.42         20.38           77.58
386.1
4  84358402         M        20.29         14.34          135.10
1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave
```

```
points_mean  \
0           0.11840           0.27760          0.3001
0.14710
1           0.08474           0.07864          0.0869
0.07017
2           0.10960           0.15990          0.1974
0.12790
3           0.14250           0.28390          0.2414
0.10520
4           0.10030           0.13280          0.1980
0.10430

   ...  radius_worst  texture_worst  perimeter_worst  area_worst  \
0  ...         25.38          17.33           184.60      2019.0
1  ...         24.99          23.41           158.80      1956.0
2  ...         23.57          25.53           152.50      1709.0
3  ...         14.91          26.50            98.87       567.7
4  ...         22.54          16.67           152.20      1575.0

   smoothness_worst  compactness_worst  concavity_worst  concave
points_worst  \
0            0.1622             0.6656           0.7119
0.2654
1            0.1238             0.1866           0.2416
0.1860
2            0.1444             0.4245           0.4504
0.2430
3            0.2098             0.8663           0.6869
0.2575
4            0.1374             0.2050           0.4000
0.1625

   symmetry_worst  fractal_dimension_worst
0          0.4601                  0.11890
1          0.2750                  0.08902
2          0.3613                  0.08758
3          0.6638                  0.17300
4          0.2364                  0.07678

[5 rows x 32 columns]
```

```python
# There is no missing value in this dataset, But how will you handle
the missing values in dataset?
# Check for missing values
missing_values = data.isnull().sum()
# print(missing_values)
data = data.dropna() # Drop rows with missing values
print('Shape of data after handling missing values: ', data.shape)
```

```
Shape of data after handling missing values:  (569, 32)
```

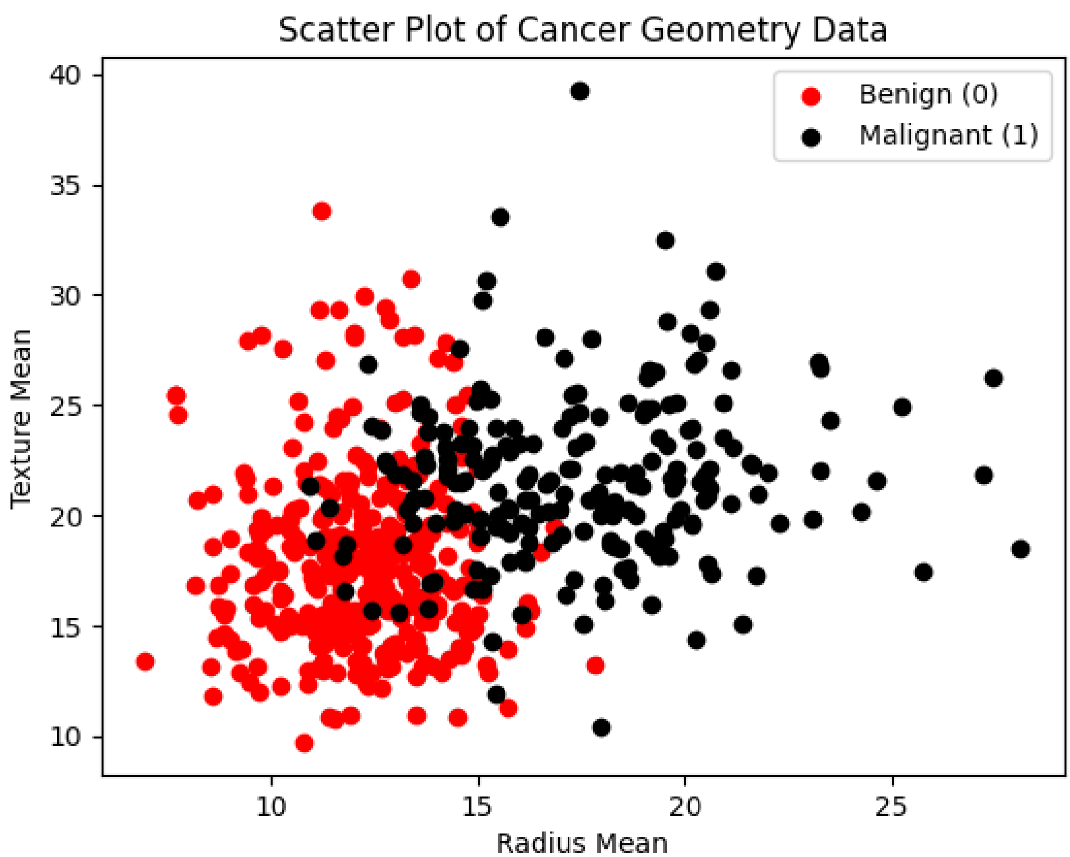
```python
# Encode labels
data["diagnosis"] = data["diagnosis"].map({"B": 0, "M": 1})
data.head()
```

```
         id  diagnosis  radius_mean  texture_mean  perimeter_mean
area_mean  \
0    842302          1        17.99         10.38          122.80
1001.0
1    842517          1        20.57         17.77          132.90
1326.0
2  84300903          1        19.69         21.25          130.00
1203.0
3  84348301          1        11.42         20.38           77.58
386.1
4  84358402          1        20.29         14.34          135.10
1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \
0          0.11840           0.27760          0.3001
0.14710
1          0.08474           0.07864          0.0869
0.07017
2          0.10960           0.15990          0.1974
0.12790
3          0.14250           0.28390          0.2414
0.10520
4          0.10030           0.13280          0.1980
0.10430

   ...  radius_worst  texture_worst  perimeter_worst  area_worst  \
0  ...         25.38          17.33           184.60      2019.0
1  ...         24.99          23.41           158.80      1956.0
2  ...         23.57          25.53           152.50      1709.0
3  ...         14.91          26.50            98.87       567.7
4  ...         22.54          16.67           152.20      1575.0

   smoothness_worst  compactness_worst  concavity_worst  concave
points_worst  \
0            0.1622             0.6656           0.7119
0.2654
1            0.1238             0.1866           0.2416
0.1860
2            0.1444             0.4245           0.4504
0.2430
3            0.2098             0.8663           0.6869
0.2575
4            0.1374             0.2050           0.4000
0.1625
```

```
   symmetry_worst  fractal_dimension_worst
0          0.4601                  0.11890
1          0.2750                  0.08902
2          0.3613                  0.08758
3          0.6638                  0.17300
4          0.2364                  0.07678

[5 rows x 32 columns]
```

Data visualization & Train-test split

```
X = data[['radius_mean', 'texture_mean']]
y = data['diagnosis']

# Separate data points based on the target variable (0 for Benign, 1
for Malignant)
benign_data = X[y == 0]
malignant_data = X[y == 1]
# Create a scatter plot
plt.scatter(benign_data['radius_mean'], benign_data['texture_mean'],
c='red', label='Benign (0)')
plt.scatter(malignant_data['radius_mean'],
malignant_data['texture_mean'], c='black', label='Malignant (1)')
# Add labels and a legend
plt.xlabel('Radius Mean')
plt.ylabel('Texture Mean')
plt.legend(loc='upper right')
# Show the plot
plt.title('Scatter Plot of Cancer Geometry Data')
plt.show()
# Split the data into training and testing sets (e.g., 80% training,
20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```



Decision Tree Classifier GINI creterion

Using K fold validation

```
val_score = {
    'recall': [],
    'precision': [],
    'accuracy': [],
    'roc_auc': [],
    'f1': []
}

mean_accuracy_scores = []
mean_precision_scores = []
mean_recall_scores = []
mean_roc_scores = []
mean_f1_scores = []

methods = ['recall', 'precision', 'accuracy', 'roc_auc', 'f1']
for i in range(1, 16):
  n = i
```

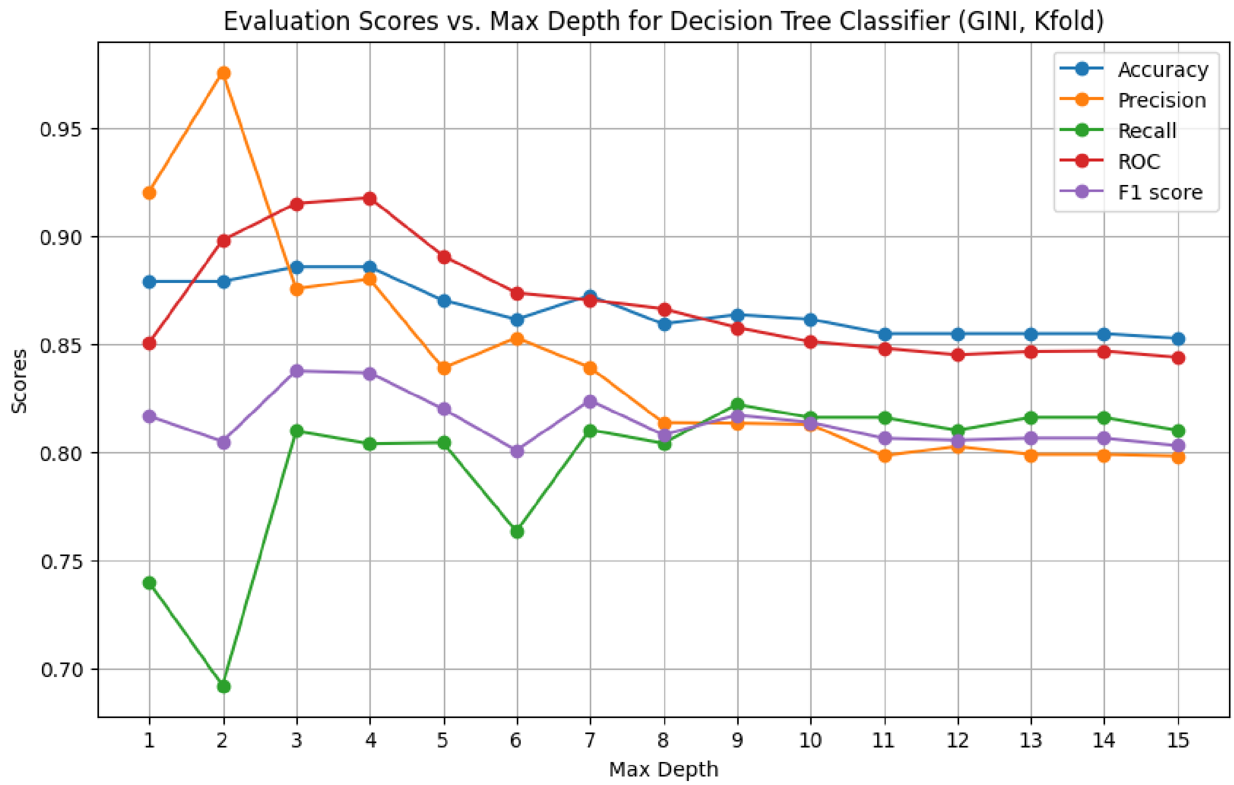
```
  # Create a decision tree classifier
  decision_tree_model_kfold_GINI =
DecisionTreeClassifier(criterion='gini', random_state=42, max_depth =
i)
  for metric in methods:
    c_val_score = cross_val_score(decision_tree_model_kfold_GINI,
X_train, y_train, cv = 5, scoring = metric)
    val_score[metric].append(c_val_score)

for i in range(1, 16):
  print(f'for max_depth = {i} the 5 fold cross validation scores are:
\n')
  mean_accuracy_scores.append(sum(val_score['accuracy'][i-1])/5)
  mean_precision_scores.append(sum(val_score['precision'][i-1])/5)
  mean_recall_scores.append(sum(val_score['recall'][i-1])/5)
  mean_roc_scores.append(sum(val_score['roc_auc'][i-1])/5)
  mean_f1_scores.append(sum(val_score['f1'][i-1])/5)
  for key in val_score.keys():
    print(f'{key} = {val_score[key][i-1]}')
  print('\n')


# Create a single plot to visualize the scores with respect to max
depth
plt.figure(figsize=(10, 6))
plt.plot(list(range(1, 16)), mean_accuracy_scores, label='Accuracy',
marker='o')
plt.plot(list(range(1, 16)), mean_precision_scores, label='Precision',
marker='o')
plt.plot(list(range(1, 16)), mean_recall_scores, label='Recall',
marker='o')
plt.plot(list(range(1, 16)), mean_roc_scores, label='ROC', marker='o')
plt.plot(list(range(1, 16)), mean_f1_scores, label='F1 score',
marker='o')


plt.xlabel('Max Depth')
plt.ylabel('Scores')
plt.title('Evaluation Scores vs. Max Depth for Decision Tree
Classifier (GINI, Kfold)')
plt.xticks(np.arange(1, 16))
plt.legend()
plt.grid(True)
plt.show()
```

```
for max_depth = 1 the 5 fold cross validation scores are:

recall = [0.75757576 0.64705882 0.91176471 0.64705882 0.73529412]
precision = [0.96153846 0.95652174 0.91176471 0.88       0.89285714]
accuracy = [0.9010989  0.85714286 0.93406593 0.83516484 0.86813187]
```

```
roc_auc = [0.87016719 0.81475748 0.92956656 0.79721362 0.84133127]
f1 = [0.84745763 0.77192982 0.91176471 0.74576271 0.80645161]


for max_depth = 2 the 5 fold cross validation scores are:

recall = [0.63636364 0.64705882 0.91176471 0.61764706 0.64705882]
precision = [1.         0.95652174 0.96875    0.95454545 1.        ]
accuracy = [0.86813187 0.85714286 0.95604396 0.84615385 0.86813187]
roc_auc = [0.88009404 0.86635707 0.9625903  0.89293086 0.88854489]
f1 = [0.77777778 0.77192982 0.93939394 0.75       0.78571429]


for max_depth = 3 the 5 fold cross validation scores are:

recall = [0.66666667 0.85294118 0.94117647 0.79411765 0.79411765]
precision = [0.91666667 0.85294118 0.86486486 0.9        0.84375   ]
accuracy = [0.85714286 0.89010989 0.92307692 0.89010989 0.86813187]
roc_auc = [0.88140021 0.90815273 0.96955624 0.9254386  0.89112487]
f1 = [0.77192982 0.85294118 0.90140845 0.84375    0.81818182]


for max_depth = 4 the 5 fold cross validation scores are:

recall = [0.66666667 0.82352941 0.94117647 0.79411765 0.79411765]
precision = [0.91666667 0.84848485 0.86486486 0.9        0.87096774]
accuracy = [0.85714286 0.87912088 0.92307692 0.89010989 0.87912088]
roc_auc = [0.87513062 0.90892673 0.96671827 0.92414861 0.91382869]
f1 = [0.77192982 0.8358209  0.90140845 0.84375    0.83076923]


for max_depth = 5 the 5 fold cross validation scores are:

recall = [0.75757576 0.82352941 0.94117647 0.76470588 0.73529412]
precision = [0.78125    0.84848485 0.86486486 0.86666667 0.83333333]
accuracy = [0.83516484 0.87912088 0.92307692 0.86813187 0.84615385]
roc_auc = [0.80956113 0.90247678 0.94220846 0.91021672 0.89009288]
f1 = [0.76923077 0.8358209  0.90140845 0.8125     0.78125   ]


for max_depth = 6 the 5 fold cross validation scores are:

recall = [0.75757576 0.64705882 0.94117647 0.73529412 0.73529412]
precision = [0.80645161 0.91666667 0.84210526 0.89285714 0.80645161]
accuracy = [0.84615385 0.84615385 0.91208791 0.86813187 0.83516484]
roc_auc = [0.79754441 0.87667699 0.93292054 0.89370485 0.86790506]
f1 = [0.78125    0.75862069 0.88888889 0.80645161 0.76923077]


for max_depth = 7 the 5 fold cross validation scores are:
```

```
recall = [0.75757576 0.82352941 0.94117647 0.79411765 0.73529412]
precision = [0.78125    0.82352941 0.91428571 0.84375    0.83333333]
accuracy = [0.83516484 0.86813187 0.94505495 0.86813187 0.84615385]
roc_auc = [0.78657262 0.88286894 0.94788442 0.86480908 0.87022704]
f1 = [0.76923077 0.82352941 0.92753623 0.81818182 0.78125   ]


for max_depth = 8 the 5 fold cross validation scores are:

recall = [0.6969697  0.85294118 0.94117647 0.79411765 0.73529412]
precision = [0.76666667 0.87878788 0.86486486 0.84375    0.71428571]
accuracy = [0.81318681 0.9010989  0.92307692 0.86813187 0.79120879]
roc_auc = [0.80329154 0.88880289 0.94865841 0.8619711  0.82894737]
f1 = [0.73015873 0.86567164 0.90140845 0.81818182 0.72463768]


for max_depth = 9 the 5 fold cross validation scores are:

recall = [0.75757576 0.85294118 0.94117647 0.79411765 0.76470588]
precision = [0.75757576 0.87878788 0.86486486 0.84375    0.72222222]
accuracy = [0.82417582 0.9010989  0.92307692 0.86813187 0.8021978 ]
roc_auc = [0.80590387 0.89138287 0.94762642 0.85319917 0.79050568]
f1 = [0.75757576 0.86567164 0.90140845 0.81818182 0.74285714]


for max_depth = 10 the 5 fold cross validation scores are:

recall = [0.72727273 0.85294118 0.94117647 0.79411765 0.76470588]
precision = [0.75       0.87878788 0.88888889 0.84375    0.7027027 ]
accuracy = [0.81318681 0.9010989  0.93406593 0.86813187 0.79120879]
roc_auc = [0.78996865 0.89344685 0.93343653 0.85319917 0.78586171]
f1 = [0.73846154 0.86567164 0.91428571 0.81818182 0.73239437]


for max_depth = 11 the 5 fold cross validation scores are:

recall = [0.72727273 0.85294118 0.94117647 0.79411765 0.76470588]
precision = [0.72727273 0.85294118 0.86486486 0.84375    0.7027027 ]
accuracy = [0.8021978  0.89010989 0.92307692 0.86813187 0.79120879]
roc_auc = [0.78605016 0.89009288 0.9254386  0.85319917 0.78586171]
f1 = [0.72727273 0.85294118 0.90140845 0.81818182 0.73239437]


for max_depth = 12 the 5 fold cross validation scores are:

recall = [0.72727273 0.82352941 0.94117647 0.79411765 0.76470588]
precision = [0.72727273 0.875      0.86486486 0.84375    0.7027027 ]
accuracy = [0.8021978  0.89010989 0.92307692 0.86813187 0.79120879]
roc_auc = [0.78605016 0.875129   0.9254386  0.85319917 0.78586171]
f1 = [0.72727273 0.84848485 0.90140845 0.81818182 0.73239437]
```

```
for max_depth = 13 the 5 fold cross validation scores are:

recall = [0.72727273 0.85294118 0.94117647 0.79411765 0.76470588]
precision = [0.72727273 0.87878788 0.84210526 0.84375    0.7027027 ]
accuracy = [0.8021978  0.9010989  0.91208791 0.86813187 0.79120879]
roc_auc = [0.78605016 0.89009288 0.91795666 0.85319917 0.78586171]
f1 = [0.72727273 0.86567164 0.88888889 0.81818182 0.73239437]


for max_depth = 14 the 5 fold cross validation scores are:

recall = [0.72727273 0.85294118 0.94117647 0.79411765 0.76470588]
precision = [0.72727273 0.87878788 0.84210526 0.84375    0.7027027 ]
accuracy = [0.8021978  0.9010989  0.91208791 0.86813187 0.79120879]
roc_auc = [0.78605016 0.89138287 0.91795666 0.85319917 0.78586171]
f1 = [0.72727273 0.86567164 0.88888889 0.81818182 0.73239437]


for max_depth = 15 the 5 fold cross validation scores are:

recall = [0.72727273 0.82352941 0.94117647 0.79411765 0.76470588]
precision = [0.72727273 0.875      0.84210526 0.84375    0.7027027 ]
accuracy = [0.8021978  0.89010989 0.91208791 0.86813187 0.79120879]
roc_auc = [0.78605016 0.87667699 0.91795666 0.85319917 0.78586171]
f1 = [0.72727273 0.84848485 0.88888889 0.81818182 0.73239437]
```



Train Validation split method

```
X_train_new, X_val, y_train_new, y_val = train_test_split(X_train,
y_train, train_size = 0.8, random_state = 0)

max_depths = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
auc_scores = []

for i in range(1, 16):
  # Create a decision tree classifier
  decision_tree_model_GINI = DecisionTreeClassifier(criterion='gini',
random_state=42, max_depth = i)
  # Train the model on the training data
  decision_tree_model_GINI.fit(X_train_new, y_train_new)
  # Make predictions on the test data
  print(f'for max_depth = {i}')
  y_pred_decision_tree = decision_tree_model_GINI.predict(X_val)
  accuracy_decision_tree = accuracy_score(y_val, y_pred_decision_tree)
  print(f'Decision Tree Accuracy: {accuracy_decision_tree:.2f}')
  precision_decision_tree = precision_score(y_val,
y_pred_decision_tree)
  print(f'Decision Tree Precision: {precision_decision_tree:.2f}')
```

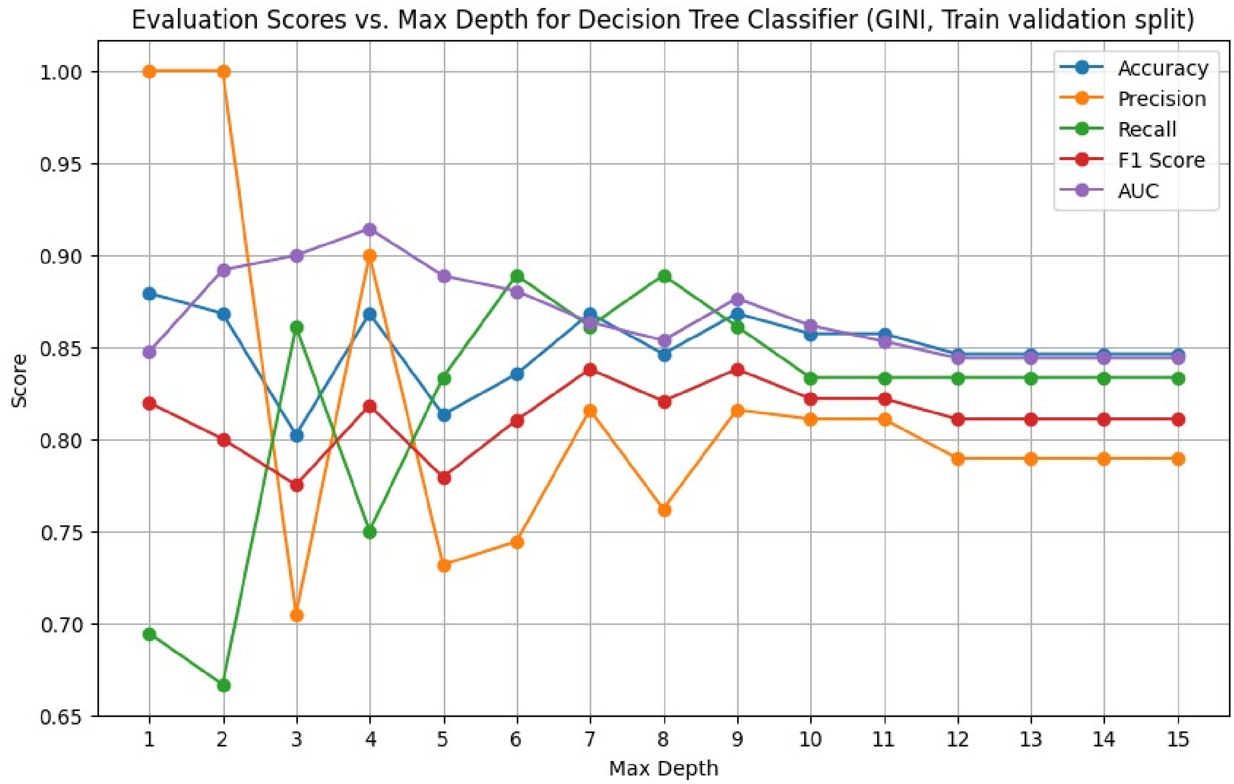
```
  recall_decision_tree = recall_score(y_val, y_pred_decision_tree)
  print(f'Decision Tree Recall: {recall_decision_tree:.2f}')
  f1_decision_tree = f1_score(y_val, y_pred_decision_tree)
  print(f'Decision Tree F1 Score: {f1_decision_tree:.2f}')


  # Get predicted probabilities for class 1 (Malignant) from the
decision tree model
  y_prob_decision_tree = decision_tree_model_GINI.predict_proba(X_val)
[:, 1]
  # Calculate ROC curve for the decision tree model
  fpr_decision_tree, tpr_decision_tree, _ = roc_curve(y_val,
y_prob_decision_tree)
  # Calculate AUC for the decision tree model
  roc_auc_decision_tree = auc(fpr_decision_tree, tpr_decision_tree)
  print(f'AUC - Decision Tree Classifier:
{roc_auc_decision_tree:.2f}')

print('************************************************************')

  max_depths.append(i)
  accuracy_scores.append(accuracy_decision_tree)
  precision_scores.append(precision_decision_tree)
  recall_scores.append(recall_decision_tree)
  f1_scores.append(f1_decision_tree)
  auc_scores.append(roc_auc_decision_tree)

# Create a single plot to visualize the scores with respect to max
depth
plt.figure(figsize=(10, 6))
plt.plot(max_depths, accuracy_scores, label='Accuracy', marker='o')
plt.plot(max_depths, precision_scores, label='Precision', marker='o')
plt.plot(max_depths, recall_scores, label='Recall', marker='o')
plt.plot(max_depths, f1_scores, label='F1 Score', marker='o')
plt.plot(max_depths, auc_scores, label='AUC', marker='o')

plt.xlabel('Max Depth')
plt.ylabel('Score')
plt.title('Evaluation Scores vs. Max Depth for Decision Tree
Classifier (GINI, Train validation split)')
plt.xticks(np.arange(1, 16))
plt.legend()
plt.grid(True)
plt.show()
```

```
for max_depth = 1
Decision Tree Accuracy: 0.88
Decision Tree Precision: 1.00
Decision Tree Recall: 0.69
Decision Tree F1 Score: 0.82
```

```
AUC - Decision Tree Classifier: 0.85
************************************************************
for max_depth = 2
Decision Tree Accuracy: 0.87
Decision Tree Precision: 1.00
Decision Tree Recall: 0.67
Decision Tree F1 Score: 0.80
AUC - Decision Tree Classifier: 0.89
************************************************************
for max_depth = 3
Decision Tree Accuracy: 0.80
Decision Tree Precision: 0.70
Decision Tree Recall: 0.86
Decision Tree F1 Score: 0.78
AUC - Decision Tree Classifier: 0.90
************************************************************
for max_depth = 4
Decision Tree Accuracy: 0.87
Decision Tree Precision: 0.90
Decision Tree Recall: 0.75
Decision Tree F1 Score: 0.82
AUC - Decision Tree Classifier: 0.91
************************************************************
for max_depth = 5
Decision Tree Accuracy: 0.81
Decision Tree Precision: 0.73
Decision Tree Recall: 0.83
Decision Tree F1 Score: 0.78
AUC - Decision Tree Classifier: 0.89
************************************************************
for max_depth = 6
Decision Tree Accuracy: 0.84
Decision Tree Precision: 0.74
Decision Tree Recall: 0.89
Decision Tree F1 Score: 0.81
AUC - Decision Tree Classifier: 0.88
************************************************************
for max_depth = 7
Decision Tree Accuracy: 0.87
Decision Tree Precision: 0.82
Decision Tree Recall: 0.86
Decision Tree F1 Score: 0.84
AUC - Decision Tree Classifier: 0.86
************************************************************
for max_depth = 8
Decision Tree Accuracy: 0.85
Decision Tree Precision: 0.76
Decision Tree Recall: 0.89
Decision Tree F1 Score: 0.82
```

```
AUC - Decision Tree Classifier: 0.85
************************************************************
for max_depth = 9
Decision Tree Accuracy: 0.87
Decision Tree Precision: 0.82
Decision Tree Recall: 0.86
Decision Tree F1 Score: 0.84
AUC - Decision Tree Classifier: 0.88
************************************************************
for max_depth = 10
Decision Tree Accuracy: 0.86
Decision Tree Precision: 0.81
Decision Tree Recall: 0.83
Decision Tree F1 Score: 0.82
AUC - Decision Tree Classifier: 0.86
************************************************************
for max_depth = 11
Decision Tree Accuracy: 0.86
Decision Tree Precision: 0.81
Decision Tree Recall: 0.83
Decision Tree F1 Score: 0.82
AUC - Decision Tree Classifier: 0.85
************************************************************
for max_depth = 12
Decision Tree Accuracy: 0.85
Decision Tree Precision: 0.79
Decision Tree Recall: 0.83
Decision Tree F1 Score: 0.81
AUC - Decision Tree Classifier: 0.84
************************************************************
for max_depth = 13
Decision Tree Accuracy: 0.85
Decision Tree Precision: 0.79
Decision Tree Recall: 0.83
Decision Tree F1 Score: 0.81
AUC - Decision Tree Classifier: 0.84
************************************************************
for max_depth = 14
Decision Tree Accuracy: 0.85
Decision Tree Precision: 0.79
Decision Tree Recall: 0.83
Decision Tree F1 Score: 0.81
AUC - Decision Tree Classifier: 0.84
************************************************************
for max_depth = 15
Decision Tree Accuracy: 0.85
Decision Tree Precision: 0.79
Decision Tree Recall: 0.83
Decision Tree F1 Score: 0.81
```

```
AUC - Decision Tree Classifier: 0.84
************************************************************
```



Metrics for selection of most suitable max_depth parameter

1. For breast cancer prediction we should focus on reducing false negatives and we can tolerate false positives to an extent, this means we can priotize recall over precision.
2. For the other scores we can try to maximize them.

Based on the above metric, the most optimal value for max_depth is **9**. Decision Tree Accuracy: 0.87. Decision Tree Precision: 0.82. Decision Tree Recall: 0.86. Decision Tree F1 Score: 0.84. AUC - Decision Tree Classifier: 0.88.

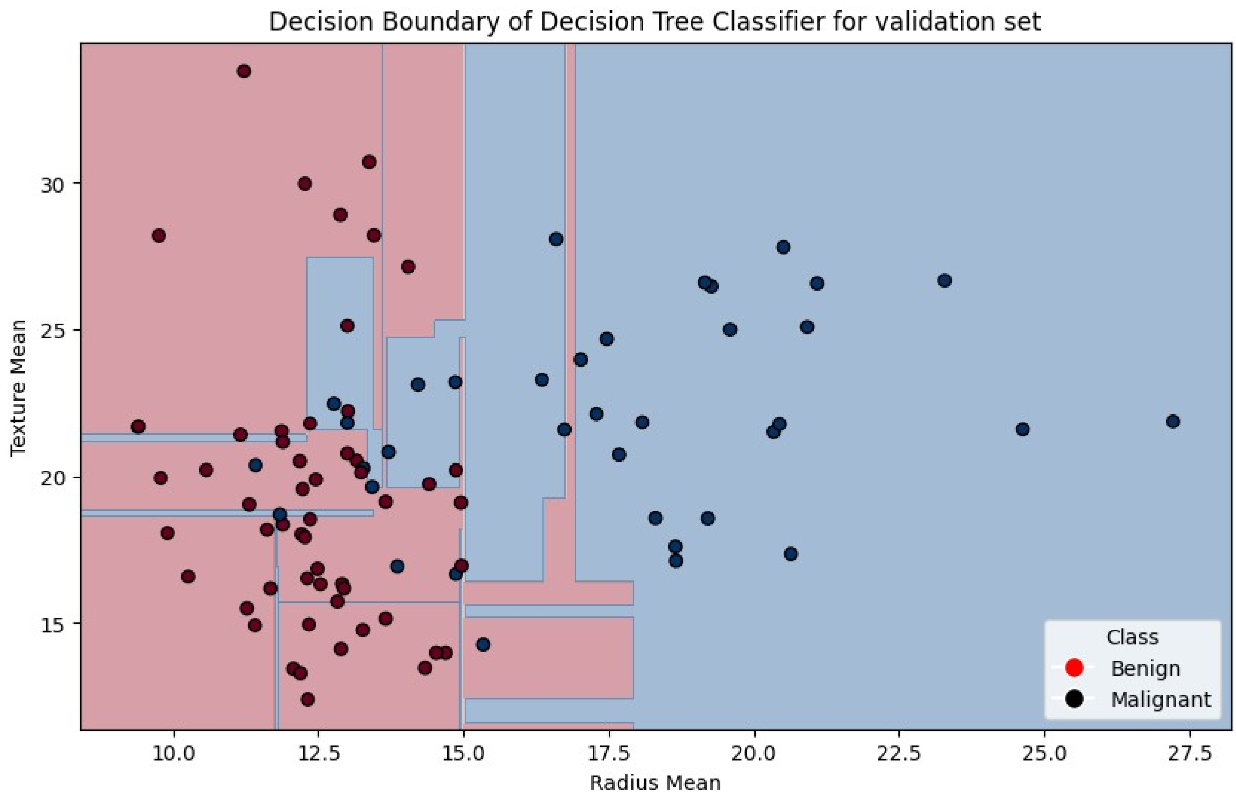
Observations

1. Here we can see that the values obtained from k fold are very close to what we obtained from train validation split.
2. The max_depth = 9 is the most optimal values as per the metric that we used earlier.

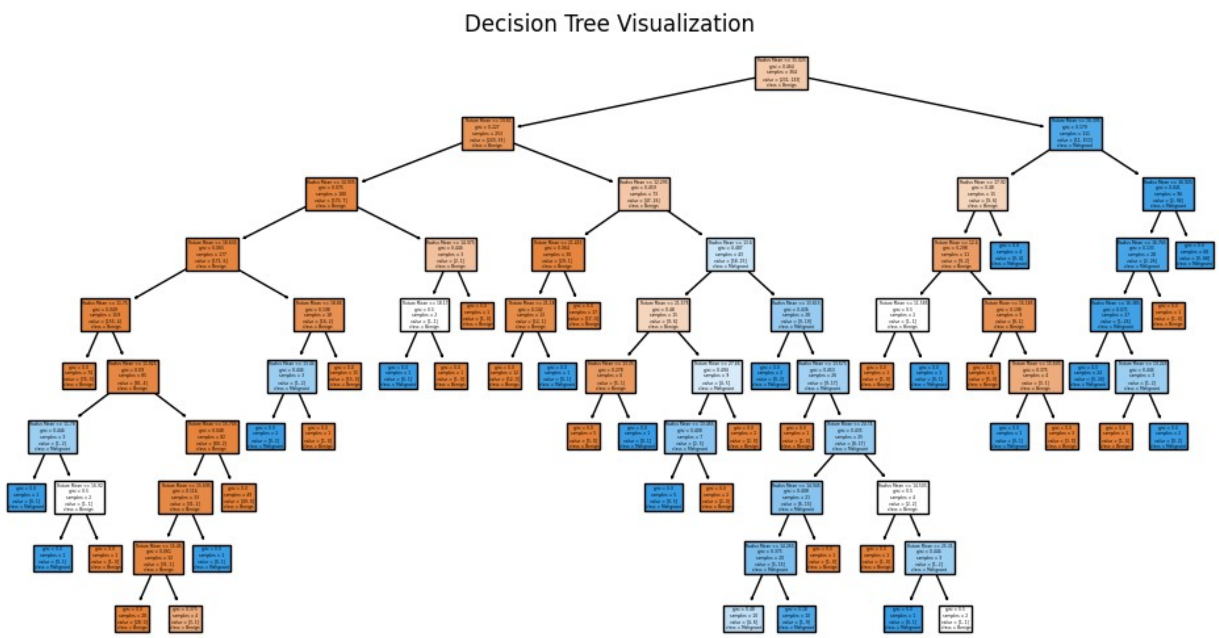
Decision Boundary of validation set

```
# Create a decision tree classifier
decision_tree_model_GINI_9 = DecisionTreeClassifier(criterion='gini',
random_state=42, max_depth = 9)
# Train the model on the training data
depth_max_cls = decision_tree_model_GINI_9.fit(X_train_new,
```

```
y_train_new)

# Define a mesh grid of points to plot the decision boundary
x_min, x_max = X_val.iloc[:, 0].min() - 1, X_val.iloc[:, 0].max() + 1
y_min, y_max = X_val.iloc[:, 1].min() - 1, X_val.iloc[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min,
y_max, 0.01))

# Use the classifier to predict the class labels for each point in the
mesh grid
Z = depth_max_cls.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.RdBu)

# Plot the data points
plt.scatter(X_val.iloc[:, 0], X_val.iloc[:, 1], c=y_val,
cmap=plt.cm.RdBu, edgecolor='k')
plt.xlabel('Radius Mean')
plt.ylabel('Texture Mean')
plt.title('Decision Boundary of Decision Tree Classifier for
validation set')

legend1 = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='red', markersize=10, label='Benign')
legend2 = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='black', markersize=10, label='Malignant')
plt.legend(handles=[legend1, legend2], title='Class', loc='lower
right')

plt.show()

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```




```
# Plot the decision tree
plt.figure(figsize=(12, 6))
plot_tree(decision_tree_model_GINI_9, filled=True,
feature_names=['Radius Mean', 'Texture Mean'], class_names=['Benign',
'Malignant'])
plt.title("Decision Tree Visualization")
plt.show()
```



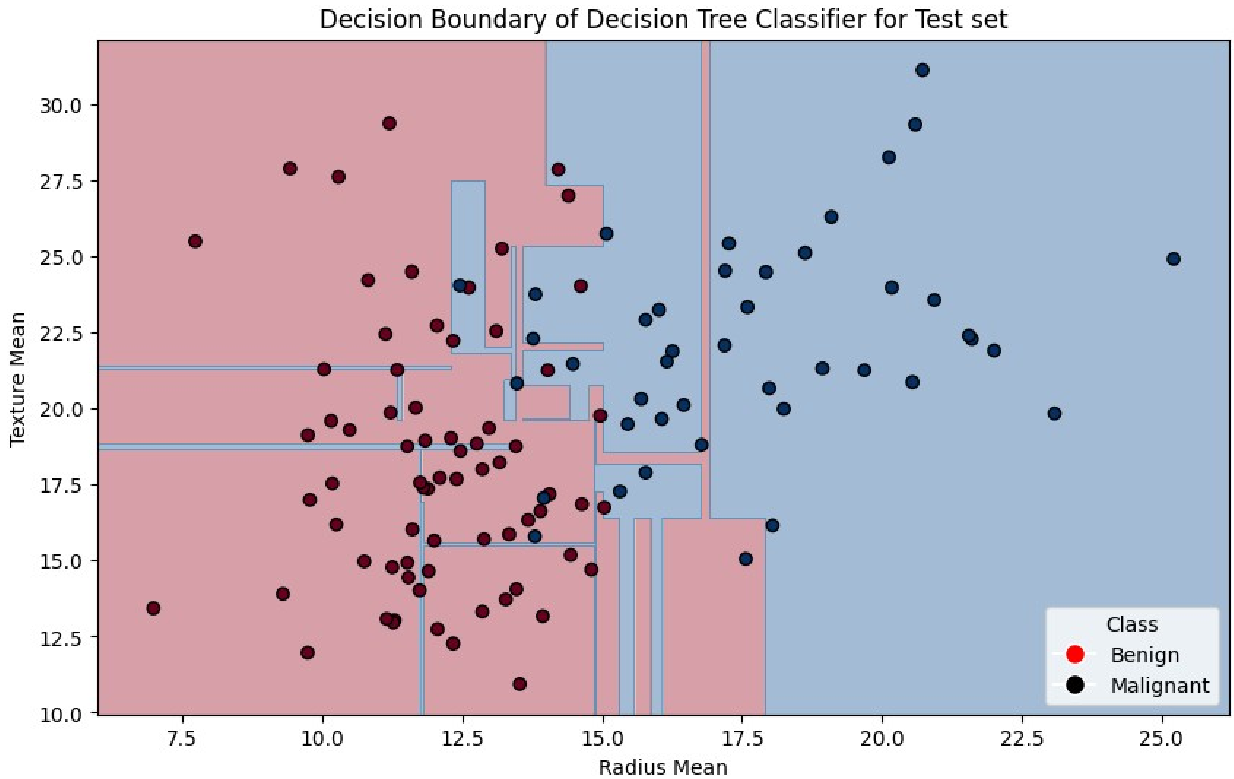
Test Error

```
# Create a decision tree classifier
decision_tree_model_GINI = DecisionTreeClassifier(criterion='gini',
random_state=42, max_depth = 9)
# Train the model on the training data
depth = decision_tree_model_GINI.fit(X_train, y_train)
# Make predictions on the test data
print(f'For test dataset using GINI creterion')
y_pred_decision_tree = decision_tree_model_GINI.predict(X_test)
accuracy_decision_tree = accuracy_score(y_test, y_pred_decision_tree)
print(f'Decision Tree Accuracy: {accuracy_decision_tree:.2f}')
precision_decision_tree = precision_score(y_test,
y_pred_decision_tree)
print(f'Decision Tree Precision: {precision_decision_tree:.2f}')
recall_decision_tree = recall_score(y_test, y_pred_decision_tree)
print(f'Decision Tree Recall: {recall_decision_tree:.2f}')
f1_decision_tree = f1_score(y_test, y_pred_decision_tree)
print(f'Decision Tree F1 Score: {f1_decision_tree:.2f}')

# Get predicted probabilities for class 1 (Malignant) from the
decision tree model
y_prob_decision_tree = decision_tree_model_GINI.predict_proba(X_test)
[:, 1]
# Calculate ROC curve for the decision tree model
fpr_decision_tree, tpr_decision_tree, _ = roc_curve(y_test,
y_prob_decision_tree)
# Calculate AUC for the decision tree model
roc_auc_decision_tree = auc(fpr_decision_tree, tpr_decision_tree)
print(f'AUC - Decision Tree Classifier: {roc_auc_decision_tree:.2f}')
```

```
print('************************************************************************************************')

# Define a mesh grid of points to plot the decision boundary
x_min, x_max = X_test.iloc[:, 0].min() - 1, X_test.iloc[:, 0].max() +
1
y_min, y_max = X_test.iloc[:, 1].min() - 1, X_test.iloc[:, 1].max() +
1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min,
y_max, 0.01))

# Use the classifier to predict the class labels for each point in the
mesh grid
Z = depth.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.RdBu)

# Plot the data points
plt.scatter(X_test.iloc[:, 0], X_test.iloc[:, 1], c=y_test,
cmap=plt.cm.RdBu, edgecolor='k')
plt.xlabel('Radius Mean')
plt.ylabel('Texture Mean')
plt.title('Decision Boundary of Decision Tree Classifier for Test
set')

legend1 = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='red', markersize=10, label='Benign')
legend2 = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='black', markersize=10, label='Malignant')
plt.legend(handles=[legend1, legend2], title='Class', loc='lower
right')

plt.show()
```

```
For test dataset using GINI creterion
Decision Tree Accuracy: 0.89
Decision Tree Precision: 0.81
Decision Tree Recall: 0.91
Decision Tree F1 Score: 0.86
AUC - Decision Tree Classifier: 0.90
************************************************************************************

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
```

```
DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```



Decision Tree Classifier Entropy creterion

Using K fold validation

```
val_score = {
    'recall': [],
    'precision': [],
    'accuracy': [],
    'roc_auc': [],
    'f1': []
}

mean_accuracy_scores = []
mean_precision_scores = []
mean_recall_scores = []
mean_roc_scores = []
```

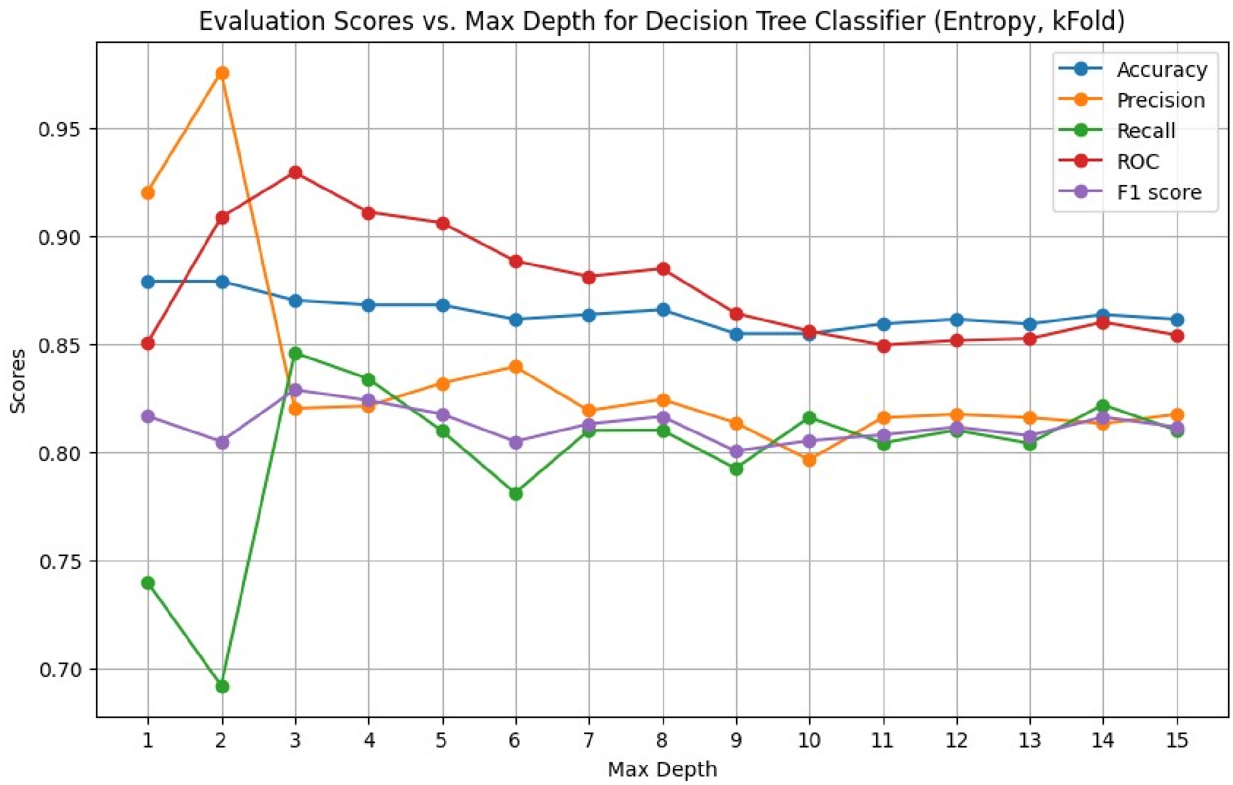
```
mean_f1_scores = []

methods = ['recall', 'precision', 'accuracy', 'roc_auc', 'f1']
for i in range(1, 16):
  n = i
  decision_tree_model_kfold_entropy =
DecisionTreeClassifier(criterion='entropy', random_state=42, max_depth
= i)
  for metric in methods:
    c_val_score = cross_val_score(decision_tree_model_kfold_entropy,
X_train, y_train, cv = 5, scoring = metric)
    val_score[metric].append(c_val_score)

for i in range(1, 16):
  print(f'for max_depth = {i} the 5 fold cross validation scores are:
\n')
  mean_accuracy_scores.append(sum(val_score['accuracy'][i-1])/5)
  mean_precision_scores.append(sum(val_score['precision'][i-1])/5)
  mean_recall_scores.append(sum(val_score['recall'][i-1])/5)
  mean_roc_scores.append(sum(val_score['roc_auc'][i-1])/5)
  mean_f1_scores.append(sum(val_score['f1'][i-1])/5)
  for key in val_score.keys():
    print(f'{key} = {val_score[key][i-1]}')
  print('\n')


# Create a single plot to visualize the scores with respect to max
depth
plt.figure(figsize=(10, 6))
plt.plot(max_depths, mean_accuracy_scores, label='Accuracy',
marker='o')
plt.plot(max_depths, mean_precision_scores, label='Precision',
marker='o')
plt.plot(max_depths, mean_recall_scores, label='Recall', marker='o')
plt.plot(max_depths, mean_roc_scores, label='ROC', marker='o')
plt.plot(max_depths, mean_f1_scores, label='F1 score', marker='o')


plt.xlabel('Max Depth')
plt.ylabel('Scores')
plt.title('Evaluation Scores vs. Max Depth for Decision Tree
Classifier (Entropy, kFold)')
plt.xticks(np.arange(1, 16))
plt.legend()
plt.grid(True)
plt.show()
```

```
for max_depth = 1 the 5 fold cross validation scores are:

recall = [0.75757576 0.64705882 0.91176471 0.64705882 0.73529412]
```

```
precision = [0.96153846 0.95652174 0.91176471 0.88       0.89285714]
accuracy = [0.9010989  0.85714286 0.93406593 0.83516484 0.86813187]
roc_auc = [0.87016719 0.81475748 0.92956656 0.79721362 0.84133127]
f1 = [0.84745763 0.77192982 0.91176471 0.74576271 0.80645161]


for max_depth = 2 the 5 fold cross validation scores are:

recall = [0.63636364 0.64705882 0.91176471 0.61764706 0.64705882]
precision = [1.         0.95652174 0.96875    0.95454545 1.        ]
accuracy = [0.86813187 0.85714286 0.95604396 0.84615385 0.86813187]
roc_auc = [0.93782654 0.86635707 0.95794634 0.89293086 0.88854489]
f1 = [0.77777778 0.77192982 0.93939394 0.75       0.78571429]


for max_depth = 3 the 5 fold cross validation scores are:

recall = [0.81818182 0.85294118 0.97058824 0.79411765 0.79411765]
precision = [0.77142857 0.85294118 0.73333333 0.9        0.84375   ]
accuracy = [0.84615385 0.89010989 0.85714286 0.89010989 0.86813187]
roc_auc = [0.94514107 0.90815273 0.96981424 0.92724458 0.89705882]
f1 = [0.79411765 0.85294118 0.83544304 0.84375    0.81818182]


for max_depth = 4 the 5 fold cross validation scores are:

recall = [0.78787879 0.82352941 0.97058824 0.79411765 0.79411765]
precision = [0.76470588 0.84848485 0.75       0.9        0.84375   ]
accuracy = [0.83516484 0.87912088 0.86813187 0.89010989 0.86813187]
roc_auc = [0.86050157 0.90892673 0.96955624 0.91847265 0.89834881]
f1 = [0.7761194  0.8358209  0.84615385 0.84375    0.81818182]


for max_depth = 5 the 5 fold cross validation scores are:

recall = [0.6969697  0.82352941 0.97058824 0.76470588 0.79411765]
precision = [0.82142857 0.84848485 0.80487805 0.86666667 0.81818182]
accuracy = [0.83516484 0.87912088 0.9010989  0.86813187 0.85714286]
roc_auc = [0.85475444 0.90247678 0.97316821 0.91718266 0.88364293]
f1 = [0.75409836 0.8358209  0.88       0.8125     0.80597015]


for max_depth = 6 the 5 fold cross validation scores are:

recall = [0.78787879 0.64705882 0.94117647 0.79411765 0.73529412]
precision = [0.78787879 0.91666667 0.86486486 0.87096774 0.75757576]
accuracy = [0.84615385 0.84615385 0.92307692 0.87912088 0.81318681]
roc_auc = [0.84770115 0.87719298 0.97729618 0.89628483 0.84391125]
f1 = [0.78787879 0.75862069 0.90140845 0.83076923 0.74626866]
```

```
for max_depth = 7 the 5 fold cross validation scores are:

recall = [0.6969697  0.85294118 0.97058824 0.79411765 0.73529412]
precision = [0.79310345 0.82857143 0.86842105 0.87096774 0.73529412]
accuracy = [0.82417582 0.87912088 0.93406593 0.87912088 0.8021978 ]
roc_auc = [0.85318704 0.88493292 0.96413829 0.88519092 0.81940144]
f1 = [0.74193548 0.84057971 0.91666667 0.83076923 0.73529412]


for max_depth = 8 the 5 fold cross validation scores are:

recall = [0.72727273 0.85294118 0.94117647 0.79411765 0.73529412]
precision = [0.75       0.87878788 0.86486486 0.87096774 0.75757576]
accuracy = [0.81318681 0.9010989  0.92307692 0.87912088 0.81318681]
roc_auc = [0.83855799 0.89009288 0.95433437 0.88854489 0.85319917]
f1 = [0.73846154 0.86567164 0.90140845 0.83076923 0.74626866]


for max_depth = 9 the 5 fold cross validation scores are:

recall = [0.72727273 0.73529412 0.97058824 0.79411765 0.73529412]
precision = [0.75       0.89285714 0.84615385 0.84375    0.73529412]
accuracy = [0.81318681 0.86813187 0.92307692 0.86813187 0.8021978 ]
roc_auc = [0.78944619 0.89009288 0.94814241 0.89009288 0.80366357]
f1 = [0.73846154 0.80645161 0.90410959 0.81818182 0.73529412]


for max_depth = 10 the 5 fold cross validation scores are:

recall = [0.72727273 0.82352941 0.97058824 0.82352941 0.73529412]
precision = [0.75       0.82352941 0.84615385 0.84848485 0.71428571]
accuracy = [0.81318681 0.86813187 0.92307692 0.87912088 0.79120879]
roc_auc = [0.78944619 0.87770898 0.94814241 0.86661507 0.79876161]
f1 = [0.73846154 0.82352941 0.90410959 0.8358209  0.72463768]


for max_depth = 11 the 5 fold cross validation scores are:

recall = [0.72727273 0.76470588 0.97058824 0.82352941 0.73529412]
precision = [0.77419355 0.89655172 0.84615385 0.84848485 0.71428571]
accuracy = [0.82417582 0.87912088 0.92307692 0.87912088 0.79120879]
roc_auc = [0.80433647 0.85190918 0.94814241 0.86790506 0.7752838 ]
f1 = [0.75       0.82539683 0.90410959 0.8358209  0.72463768]


for max_depth = 12 the 5 fold cross validation scores are:

recall = [0.72727273 0.76470588 0.97058824 0.82352941 0.76470588]
precision = [0.77419355 0.89655172 0.84615385 0.84848485 0.72222222]
accuracy = [0.82417582 0.87912088 0.92307692 0.87912088 0.8021978 ]
roc_auc = [0.79179728 0.85603715 0.94814241 0.86790506 0.79463364]
```

```
f1 = [0.75       0.82539683 0.90410959 0.8358209  0.74285714]


for max_depth = 13 the 5 fold cross validation scores are:

recall = [0.6969697  0.76470588 0.97058824 0.82352941 0.76470588]
precision = [0.76666667 0.89655172 0.84615385 0.84848485 0.72222222]
accuracy = [0.81318681 0.87912088 0.92307692 0.87912088 0.8021978 ]
roc_auc = [0.79623824 0.85603715 0.94814241 0.86790506 0.79463364]
f1 = [0.73015873 0.82539683 0.90410959 0.8358209  0.74285714]


for max_depth = 14 the 5 fold cross validation scores are:

recall = [0.72727273 0.82352941 0.97058824 0.82352941 0.76470588]
precision = [0.77419355 0.875      0.84615385 0.84848485 0.72222222]
accuracy = [0.82417582 0.89010989 0.92307692 0.87912088 0.8021978 ]
roc_auc = [0.81191223 0.87667699 0.9496904  0.86790506 0.79463364]
f1 = [0.75       0.84848485 0.90410959 0.8358209  0.74285714]


for max_depth = 15 the 5 fold cross validation scores are:

recall = [0.72727273 0.76470588 0.97058824 0.82352941 0.76470588]
precision = [0.77419355 0.89655172 0.84615385 0.84848485 0.72222222]
accuracy = [0.82417582 0.87912088 0.92307692 0.87912088 0.8021978 ]
roc_auc = [0.80329154 0.85603715 0.9496904  0.86790506 0.79463364]
f1 = [0.75       0.82539683 0.90410959 0.8358209  0.74285714]
```



Train Validation split method

```
X_train_new, X_val, y_train_new, y_val = train_test_split(X_train,
y_train, train_size = 0.8, random_state = 0)

max_depths = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
auc_scores = []

for i in range(1, 16):
  # Create a decision tree classifier
  decision_tree_model_GINI =
DecisionTreeClassifier(criterion='entropy', random_state=42, max_depth
= i)
  # Train the model on the training data
  decision_tree_model_GINI.fit(X_train_new, y_train_new)
  # Make predictions on the test data
  print(f'for max_depth = {i}')
  y_pred_decision_tree = decision_tree_model_GINI.predict(X_val)
  accuracy_decision_tree = accuracy_score(y_val, y_pred_decision_tree)
  print(f'Decision Tree Accuracy: {accuracy_decision_tree:.2f}')
  precision_decision_tree = precision_score(y_val,
y_pred_decision_tree)
```

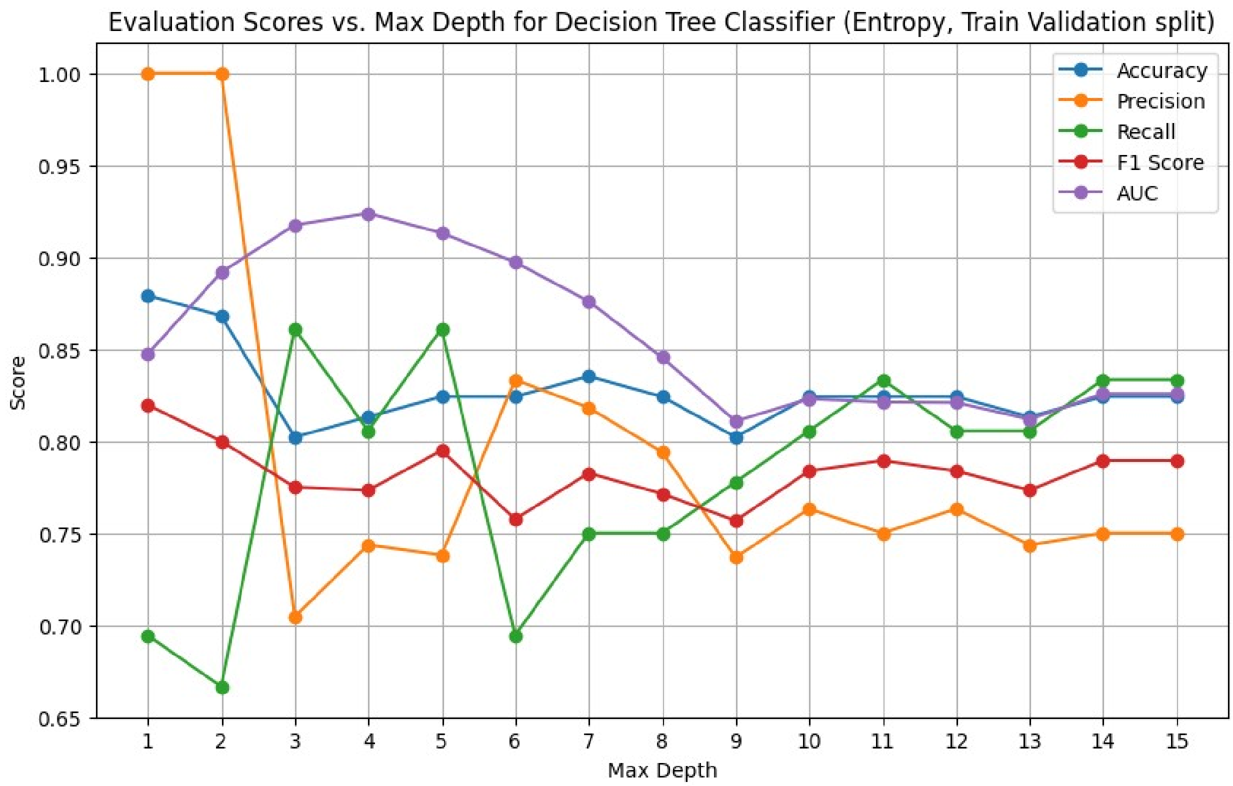
```
  print(f'Decision Tree Precision: {precision_decision_tree:.2f}')
  recall_decision_tree = recall_score(y_val, y_pred_decision_tree)
  print(f'Decision Tree Recall: {recall_decision_tree:.2f}')
  f1_decision_tree = f1_score(y_val, y_pred_decision_tree)
  print(f'Decision Tree F1 Score: {f1_decision_tree:.2f}')


  # Get predicted probabilities for class 1 (Malignant) from the
decision tree model
  y_prob_decision_tree = decision_tree_model_GINI.predict_proba(X_val)
[:, 1]
  # Calculate ROC curve for the decision tree model
  fpr_decision_tree, tpr_decision_tree, _ = roc_curve(y_val,
y_prob_decision_tree)
  # Calculate AUC for the decision tree model
  roc_auc_decision_tree = auc(fpr_decision_tree, tpr_decision_tree)
  print(f'AUC - Decision Tree Classifier:
{roc_auc_decision_tree:.2f}')

print('************************************************************')

  max_depths.append(i)
  accuracy_scores.append(accuracy_decision_tree)
  precision_scores.append(precision_decision_tree)
  recall_scores.append(recall_decision_tree)
  f1_scores.append(f1_decision_tree)
  auc_scores.append(roc_auc_decision_tree)

# Create a single plot to visualize the scores with respect to max
depth
plt.figure(figsize=(10, 6))
plt.plot(max_depths, accuracy_scores, label='Accuracy', marker='o')
plt.plot(max_depths, precision_scores, label='Precision', marker='o')
plt.plot(max_depths, recall_scores, label='Recall', marker='o')
plt.plot(max_depths, f1_scores, label='F1 Score', marker='o')
plt.plot(max_depths, auc_scores, label='AUC', marker='o')

plt.xlabel('Max Depth')
plt.ylabel('Score')
plt.title('Evaluation Scores vs. Max Depth for Decision Tree
Classifier (Entropy, Train Validation split)')
plt.xticks(np.arange(1, 16))
plt.legend()
plt.grid(True)
plt.show()
```

```
for max_depth = 1
Decision Tree Accuracy: 0.88
Decision Tree Precision: 1.00
Decision Tree Recall: 0.69
```

```
Decision Tree F1 Score: 0.82
AUC - Decision Tree Classifier: 0.85
************************************************************
for max_depth = 2
Decision Tree Accuracy: 0.87
Decision Tree Precision: 1.00
Decision Tree Recall: 0.67
Decision Tree F1 Score: 0.80
AUC - Decision Tree Classifier: 0.89
************************************************************
for max_depth = 3
Decision Tree Accuracy: 0.80
Decision Tree Precision: 0.70
Decision Tree Recall: 0.86
Decision Tree F1 Score: 0.78
AUC - Decision Tree Classifier: 0.92
************************************************************
for max_depth = 4
Decision Tree Accuracy: 0.81
Decision Tree Precision: 0.74
Decision Tree Recall: 0.81
Decision Tree F1 Score: 0.77
AUC - Decision Tree Classifier: 0.92
************************************************************
for max_depth = 5
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.74
Decision Tree Recall: 0.86
Decision Tree F1 Score: 0.79
AUC - Decision Tree Classifier: 0.91
************************************************************
for max_depth = 6
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.83
Decision Tree Recall: 0.69
Decision Tree F1 Score: 0.76
AUC - Decision Tree Classifier: 0.90
************************************************************
for max_depth = 7
Decision Tree Accuracy: 0.84
Decision Tree Precision: 0.82
Decision Tree Recall: 0.75
Decision Tree F1 Score: 0.78
AUC - Decision Tree Classifier: 0.88
************************************************************
for max_depth = 8
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.79
Decision Tree Recall: 0.75
```

```
Decision Tree F1 Score: 0.77
AUC - Decision Tree Classifier: 0.85
************************************************************
for max_depth = 9
Decision Tree Accuracy: 0.80
Decision Tree Precision: 0.74
Decision Tree Recall: 0.78
Decision Tree F1 Score: 0.76
AUC - Decision Tree Classifier: 0.81
************************************************************
for max_depth = 10
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.76
Decision Tree Recall: 0.81
Decision Tree F1 Score: 0.78
AUC - Decision Tree Classifier: 0.82
************************************************************
for max_depth = 11
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.75
Decision Tree Recall: 0.83
Decision Tree F1 Score: 0.79
AUC - Decision Tree Classifier: 0.82
************************************************************
for max_depth = 12
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.76
Decision Tree Recall: 0.81
Decision Tree F1 Score: 0.78
AUC - Decision Tree Classifier: 0.82
************************************************************
for max_depth = 13
Decision Tree Accuracy: 0.81
Decision Tree Precision: 0.74
Decision Tree Recall: 0.81
Decision Tree F1 Score: 0.77
AUC - Decision Tree Classifier: 0.81
************************************************************
for max_depth = 14
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.75
Decision Tree Recall: 0.83
Decision Tree F1 Score: 0.79
AUC - Decision Tree Classifier: 0.83
************************************************************
for max_depth = 15
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.75
Decision Tree Recall: 0.83
```

```
Decision Tree F1 Score: 0.79
AUC - Decision Tree Classifier: 0.83
************************************************************
```



Metrics for selection of most suitable max_depth parameter

1. For breast cancer prediction we should focus on reducing false negatives and we can tolerate false positives to an extent, this means we can priotize recall over precision.
2. For the other scores we can try to maximize them.

Based on the above metric, the most optimal value for max_depth is **5**. Decision Tree Accuracy: 0.82 Decision Tree Precision: 0.74 Decision Tree Recall: 0.86 Decision Tree F1 Score: 0.79 AUC - Decision Tree Classifier: 0.91

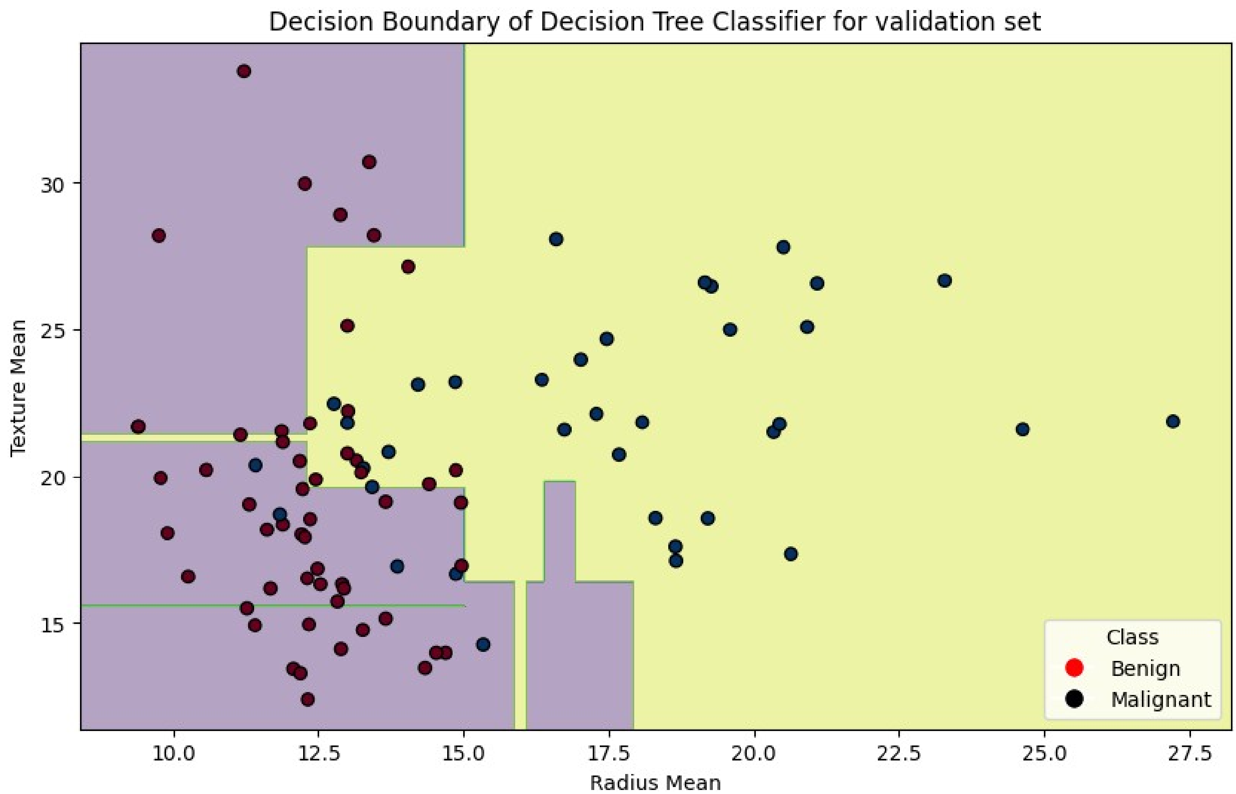
Observations

1. Here we can see that the values obtained from k fold are very close to what we obtained from train validation split.
2. Here max_depth = 5 is the most optimal values as per the metric that we used earlier.

```
# Create a decision tree classifier
decision_tree_model_entropy_5 = 
DecisionTreeClassifier(criterion='entropy', random_state=42, max_depth
= 5)
# Train the model on the training data
depth_max_cls = decision_tree_model_entropy_5.fit(X_train_new,
```

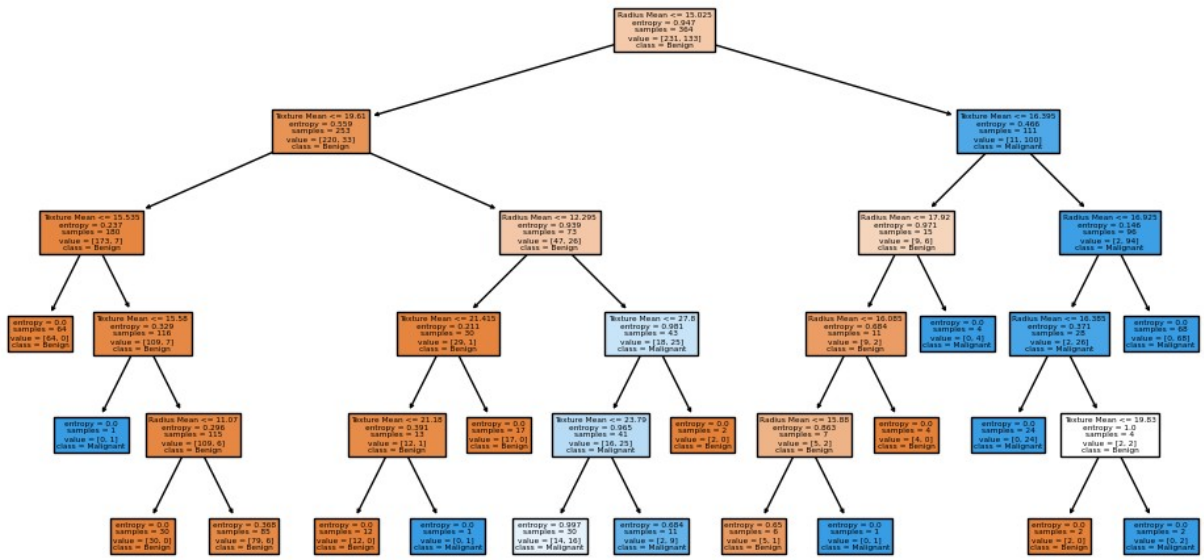
```
y_train_new)
# Define a mesh grid of points to plot the decision boundary
x_min, x_max = X_val.iloc[:, 0].min() - 1, X_val.iloc[:, 0].max() + 1
y_min, y_max = X_val.iloc[:, 1].min() - 1, X_val.iloc[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min,
y_max, 0.01))

# Use the classifier to predict the class labels for each point in the
mesh grid
Z = depth_max_cls.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.4)

# Plot the data points
plt.scatter(X_val.iloc[:, 0], X_val.iloc[:, 1], c=y_val,
cmap=plt.cm.RdBu, edgecolor='k')
plt.xlabel('Radius Mean')
plt.ylabel('Texture Mean')
plt.title('Decision Boundary of Decision Tree Classifier for
validation set')

legend1 = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='red', markersize=10, label='Benign')
legend2 = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='black', markersize=10, label='Malignant')
plt.legend(handles=[legend1, legend2], title='Class', loc='lower
right')

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```




```
# Plot the decision tree
plt.figure(figsize=(12, 6))
plot_tree(decision_tree_model_entropy_5, filled=True,
feature_names=['Radius Mean', 'Texture Mean'], class_names=['Benign',
'Malignant'])
plt.title("Decision Tree Visualization")
plt.show()
```

Decision Tree Visualization



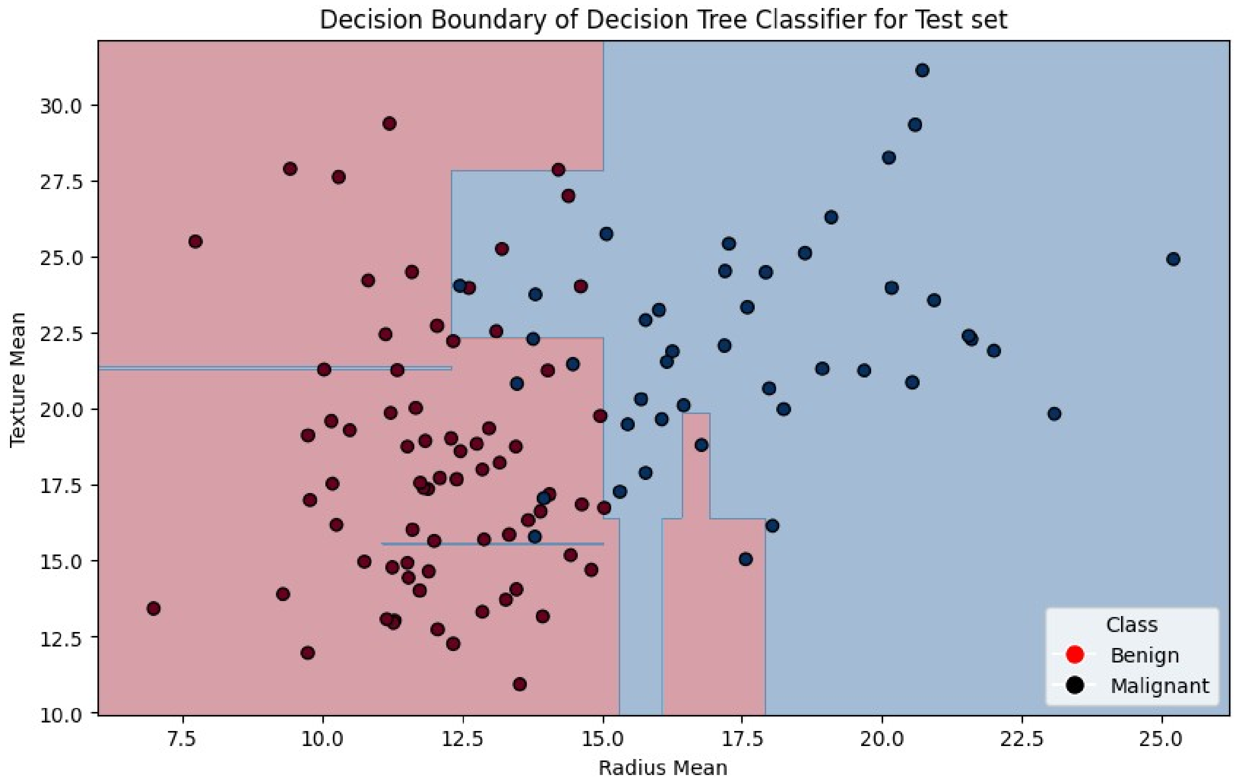
Test Error

```python
# Create a decision tree classifier
decision_tree_model_entropy = DecisionTreeClassifier(criterion='entropy', random_state=42, max_depth = 5)
# Train the model on the training data
depth = decision_tree_model_entropy.fit(X_train, y_train)
# Make predictions on the test data
print(f'For test dataset using Entropy criterion')
y_pred_decision_tree = decision_tree_model_entropy.predict(X_test)
accuracy_decision_tree = accuracy_score(y_test, y_pred_decision_tree)
print(f'Decision Tree Accuracy: {accuracy_decision_tree:.2f}')
precision_decision_tree = precision_score(y_test, y_pred_decision_tree)
print(f'Decision Tree Precision: {precision_decision_tree:.2f}')
recall_decision_tree = recall_score(y_test, y_pred_decision_tree)
print(f'Decision Tree Recall: {recall_decision_tree:.2f}')
f1_decision_tree = f1_score(y_test, y_pred_decision_tree)
print(f'Decision Tree F1 Score: {f1_decision_tree:.2f}')

# Get predicted probabilities for class 1 (Malignant) from the decision tree model
y_prob_decision_tree = decision_tree_model_entropy.predict_proba(X_test)[:, 1]
# Calculate ROC curve for the decision tree model
fpr_decision_tree, tpr_decision_tree, _ = roc_curve(y_test, y_prob_decision_tree)
# Calculate AUC for the decision tree model
roc_auc_decision_tree = auc(fpr_decision_tree, tpr_decision_tree)
```

```python
print(f'AUC - Decision Tree Classifier: {roc_auc_decision_tree:.2f}')
print('*****************************************************************
*****************************')


# Define a mesh grid of points to plot the decision boundary
x_min, x_max = X_test.iloc[:, 0].min() - 1, X_test.iloc[:, 0].max() +
1
y_min, y_max = X_test.iloc[:, 1].min() - 1, X_test.iloc[:, 1].max() +
1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min,
y_max, 0.01))

# Use the classifier to predict the class labels for each point in the
mesh grid
Z = depth.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.RdBu)

# Plot the data points
plt.scatter(X_test.iloc[:, 0], X_test.iloc[:, 1], c=y_test,
cmap=plt.cm.RdBu, edgecolor='k')
plt.xlabel('Radius Mean')
plt.ylabel('Texture Mean')
plt.title('Decision Boundary of Decision Tree Classifier for Test
set')

legend1 = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='red', markersize=10, label='Benign')
legend2 = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='black', markersize=10, label='Malignant')
plt.legend(handles=[legend1, legend2], title='Class', loc='lower
right')

plt.show()
```

```
For test dataset using Entropy criterion
Decision Tree Accuracy: 0.88
Decision Tree Precision: 0.84
Decision Tree Recall: 0.84
Decision Tree F1 Score: 0.84
AUC - Decision Tree Classifier: 0.93
**********************************************************************
*********************

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
```

```
DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```



Decision Tree Classifier Log loss creterion

Using K fold validation

```
val_score = {
    'recall': [],
    'precision': [],
    'accuracy': [],
    'roc_auc': [],
    'f1': []
}

mean_accuracy_scores = []
mean_precision_scores = []
mean_recall_scores = []
mean_roc_scores = []
```

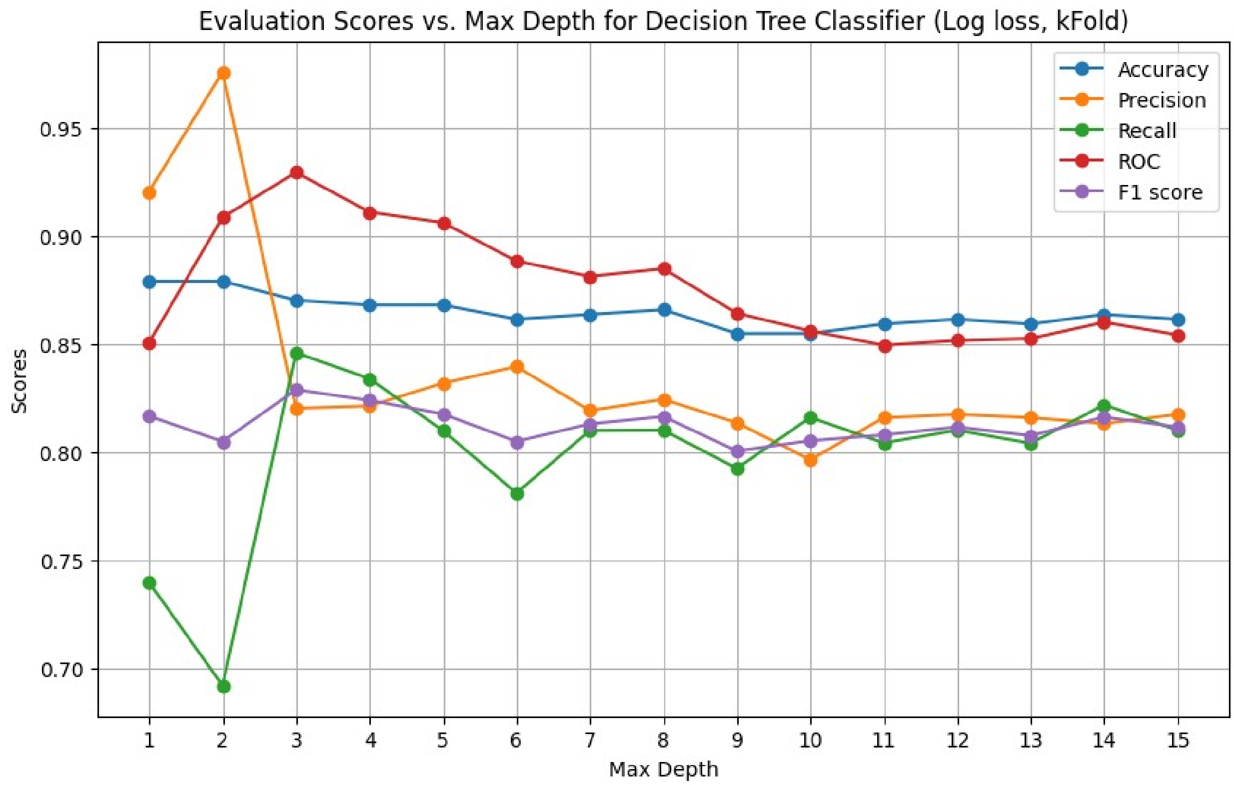
```
mean_f1_scores = []

methods = ['recall', 'precision', 'accuracy', 'roc_auc', 'f1']
for i in range(1, 16):
  n = i
  decision_tree_model_kfold_log_loss =
DecisionTreeClassifier(criterion='log_loss', random_state=42,
max_depth = i)
  for metric in methods:
    c_val_score = cross_val_score(decision_tree_model_kfold_log_loss,
X_train, y_train, cv = 5, scoring = metric)
    val_score[metric].append(c_val_score)

for i in range(1, 16):
  print(f'for max_depth = {i} the 5 fold cross validation scores are:
\n')
  mean_accuracy_scores.append(sum(val_score['accuracy'][i-1])/5)
  mean_precision_scores.append(sum(val_score['precision'][i-1])/5)
  mean_recall_scores.append(sum(val_score['recall'][i-1])/5)
  mean_roc_scores.append(sum(val_score['roc_auc'][i-1])/5)
  mean_f1_scores.append(sum(val_score['f1'][i-1])/5)
  for key in val_score.keys():
    print(f'{key} = {val_score[key][i-1]}')
  print('\n')


# Create a single plot to visualize the scores with respect to max
depth
plt.figure(figsize=(10, 6))
plt.plot(max_depths, mean_accuracy_scores, label='Accuracy',
marker='o')
plt.plot(max_depths, mean_precision_scores, label='Precision',
marker='o')
plt.plot(max_depths, mean_recall_scores, label='Recall', marker='o')
plt.plot(max_depths, mean_roc_scores, label='ROC', marker='o')
plt.plot(max_depths, mean_f1_scores, label='F1 score', marker='o')


plt.xlabel('Max Depth')
plt.ylabel('Scores')
plt.title('Evaluation Scores vs. Max Depth for Decision Tree
Classifier (Log loss, kFold)')
plt.xticks(np.arange(1, 16))
plt.legend()
plt.grid(True)
plt.show()
```

```
for max_depth = 1 the 5 fold cross validation scores are:

recall = [0.75757576 0.64705882 0.91176471 0.64705882 0.73529412]
```

```
precision = [0.96153846 0.95652174 0.91176471 0.88       0.89285714]
accuracy = [0.9010989  0.85714286 0.93406593 0.83516484 0.86813187]
roc_auc = [0.87016719 0.81475748 0.92956656 0.79721362 0.84133127]
f1 = [0.84745763 0.77192982 0.91176471 0.74576271 0.80645161]


for max_depth = 2 the 5 fold cross validation scores are:

recall = [0.63636364 0.64705882 0.91176471 0.61764706 0.64705882]
precision = [1.         0.95652174 0.96875    0.95454545 1.        ]
accuracy = [0.86813187 0.85714286 0.95604396 0.84615385 0.86813187]
roc_auc = [0.93782654 0.86635707 0.95794634 0.89293086 0.88854489]
f1 = [0.77777778 0.77192982 0.93939394 0.75       0.78571429]


for max_depth = 3 the 5 fold cross validation scores are:

recall = [0.81818182 0.85294118 0.97058824 0.79411765 0.79411765]
precision = [0.77142857 0.85294118 0.73333333 0.9        0.84375   ]
accuracy = [0.84615385 0.89010989 0.85714286 0.89010989 0.86813187]
roc_auc = [0.94514107 0.90815273 0.96981424 0.92724458 0.89705882]
f1 = [0.79411765 0.85294118 0.83544304 0.84375    0.81818182]


for max_depth = 4 the 5 fold cross validation scores are:

recall = [0.78787879 0.82352941 0.97058824 0.79411765 0.79411765]
precision = [0.76470588 0.84848485 0.75       0.9        0.84375   ]
accuracy = [0.83516484 0.87912088 0.86813187 0.89010989 0.86813187]
roc_auc = [0.86050157 0.90892673 0.96955624 0.91847265 0.89834881]
f1 = [0.7761194  0.8358209  0.84615385 0.84375    0.81818182]


for max_depth = 5 the 5 fold cross validation scores are:

recall = [0.6969697  0.82352941 0.97058824 0.76470588 0.79411765]
precision = [0.82142857 0.84848485 0.80487805 0.86666667 0.81818182]
accuracy = [0.83516484 0.87912088 0.9010989  0.86813187 0.85714286]
roc_auc = [0.85475444 0.90247678 0.97316821 0.91718266 0.88364293]
f1 = [0.75409836 0.8358209  0.88       0.8125     0.80597015]


for max_depth = 6 the 5 fold cross validation scores are:

recall = [0.78787879 0.64705882 0.94117647 0.79411765 0.73529412]
precision = [0.78787879 0.91666667 0.86486486 0.87096774 0.75757576]
accuracy = [0.84615385 0.84615385 0.92307692 0.87912088 0.81318681]
roc_auc = [0.84770115 0.87719298 0.97729618 0.89628483 0.84391125]
f1 = [0.78787879 0.75862069 0.90140845 0.83076923 0.74626866]
```

```
for max_depth = 7 the 5 fold cross validation scores are:

recall = [0.6969697  0.85294118 0.97058824 0.79411765 0.73529412]
precision = [0.79310345 0.82857143 0.86842105 0.87096774 0.73529412]
accuracy = [0.82417582 0.87912088 0.93406593 0.87912088 0.8021978 ]
roc_auc = [0.85318704 0.88493292 0.96413829 0.88519092 0.81940144]
f1 = [0.74193548 0.84057971 0.91666667 0.83076923 0.73529412]


for max_depth = 8 the 5 fold cross validation scores are:

recall = [0.72727273 0.85294118 0.94117647 0.79411765 0.73529412]
precision = [0.75       0.87878788 0.86486486 0.87096774 0.75757576]
accuracy = [0.81318681 0.9010989  0.92307692 0.87912088 0.81318681]
roc_auc = [0.83855799 0.89009288 0.95433437 0.88854489 0.85319917]
f1 = [0.73846154 0.86567164 0.90140845 0.83076923 0.74626866]


for max_depth = 9 the 5 fold cross validation scores are:

recall = [0.72727273 0.73529412 0.97058824 0.79411765 0.73529412]
precision = [0.75       0.89285714 0.84615385 0.84375    0.73529412]
accuracy = [0.81318681 0.86813187 0.92307692 0.86813187 0.8021978 ]
roc_auc = [0.78944619 0.89009288 0.94814241 0.89009288 0.80366357]
f1 = [0.73846154 0.80645161 0.90410959 0.81818182 0.73529412]


for max_depth = 10 the 5 fold cross validation scores are:

recall = [0.72727273 0.82352941 0.97058824 0.82352941 0.73529412]
precision = [0.75       0.82352941 0.84615385 0.84848485 0.71428571]
accuracy = [0.81318681 0.86813187 0.92307692 0.87912088 0.79120879]
roc_auc = [0.78944619 0.87770898 0.94814241 0.86661507 0.79876161]
f1 = [0.73846154 0.82352941 0.90410959 0.8358209  0.72463768]


for max_depth = 11 the 5 fold cross validation scores are:

recall = [0.72727273 0.76470588 0.97058824 0.82352941 0.73529412]
precision = [0.77419355 0.89655172 0.84615385 0.84848485 0.71428571]
accuracy = [0.82417582 0.87912088 0.92307692 0.87912088 0.79120879]
roc_auc = [0.80433647 0.85190918 0.94814241 0.86790506 0.7752838 ]
f1 = [0.75       0.82539683 0.90410959 0.8358209  0.72463768]


for max_depth = 12 the 5 fold cross validation scores are:

recall = [0.72727273 0.76470588 0.97058824 0.82352941 0.76470588]
precision = [0.77419355 0.89655172 0.84615385 0.84848485 0.72222222]
accuracy = [0.82417582 0.87912088 0.92307692 0.87912088 0.8021978 ]
roc_auc = [0.79179728 0.85603715 0.94814241 0.86790506 0.79463364]
```

```
f1 = [0.75       0.82539683 0.90410959 0.8358209  0.74285714]


for max_depth = 13 the 5 fold cross validation scores are:

recall = [0.6969697  0.76470588 0.97058824 0.82352941 0.76470588]
precision = [0.76666667 0.89655172 0.84615385 0.84848485 0.72222222]
accuracy = [0.81318681 0.87912088 0.92307692 0.87912088 0.8021978 ]
roc_auc = [0.79623824 0.85603715 0.94814241 0.86790506 0.79463364]
f1 = [0.73015873 0.82539683 0.90410959 0.8358209  0.74285714]


for max_depth = 14 the 5 fold cross validation scores are:

recall = [0.72727273 0.82352941 0.97058824 0.82352941 0.76470588]
precision = [0.77419355 0.875      0.84615385 0.84848485 0.72222222]
accuracy = [0.82417582 0.89010989 0.92307692 0.87912088 0.8021978 ]
roc_auc = [0.81191223 0.87667699 0.9496904  0.86790506 0.79463364]
f1 = [0.75       0.84848485 0.90410959 0.8358209  0.74285714]


for max_depth = 15 the 5 fold cross validation scores are:

recall = [0.72727273 0.76470588 0.97058824 0.82352941 0.76470588]
precision = [0.77419355 0.89655172 0.84615385 0.84848485 0.72222222]
accuracy = [0.82417582 0.87912088 0.92307692 0.87912088 0.8021978 ]
roc_auc = [0.80329154 0.85603715 0.9496904  0.86790506 0.79463364]
f1 = [0.75       0.82539683 0.90410959 0.8358209  0.74285714]
```



Train Validation split method

```
X_train_new, X_val, y_train_new, y_val = train_test_split(X_train,
y_train, train_size = 0.8, random_state = 0)

max_depths = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
auc_scores = []

for i in range(1, 16):
  # Create a decision tree classifier
  decision_tree_model_GINI =
DecisionTreeClassifier(criterion='log_loss', random_state=42,
max_depth = i)
  # Train the model on the training data
  decision_tree_model_GINI.fit(X_train_new, y_train_new)
  # Make predictions on the test data
  print(f'for max_depth = {i}')
  y_pred_decision_tree = decision_tree_model_GINI.predict(X_val)
  accuracy_decision_tree = accuracy_score(y_val, y_pred_decision_tree)
  print(f'Decision Tree Accuracy: {accuracy_decision_tree:.2f}')
  precision_decision_tree = precision_score(y_val,
y_pred_decision_tree)
```

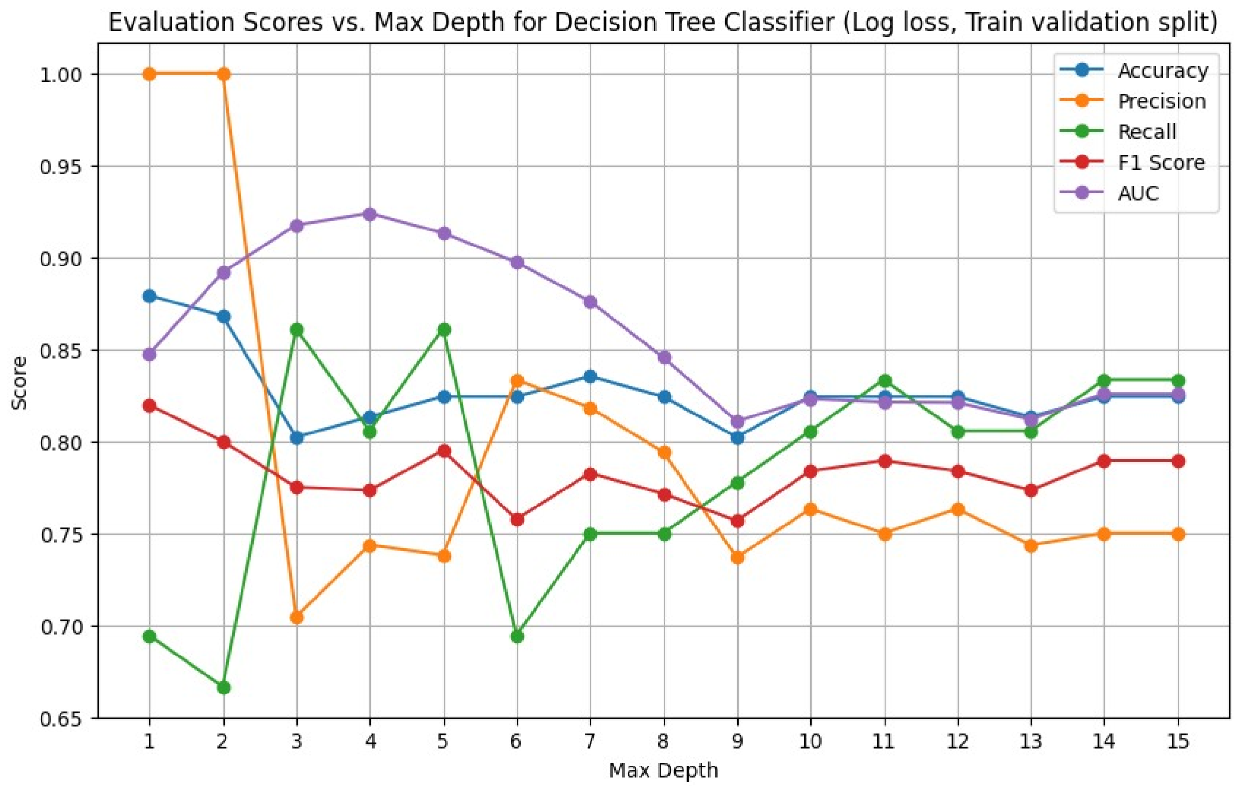
```
  print(f'Decision Tree Precision: {precision_decision_tree:.2f}')
  recall_decision_tree = recall_score(y_val, y_pred_decision_tree)
  print(f'Decision Tree Recall: {recall_decision_tree:.2f}')
  f1_decision_tree = f1_score(y_val, y_pred_decision_tree)
  print(f'Decision Tree F1 Score: {f1_decision_tree:.2f}')


  # Get predicted probabilities for class 1 (Malignant) from the
decision tree model
  y_prob_decision_tree = decision_tree_model_GINI.predict_proba(X_val)
[:, 1]
  # Calculate ROC curve for the decision tree model
  fpr_decision_tree, tpr_decision_tree, _ = roc_curve(y_val,
y_prob_decision_tree)
  # Calculate AUC for the decision tree model
  roc_auc_decision_tree = auc(fpr_decision_tree, tpr_decision_tree)
  print(f'AUC - Decision Tree Classifier:
{roc_auc_decision_tree:.2f}')

print('*************************************************************')

  max_depths.append(i)
  accuracy_scores.append(accuracy_decision_tree)
  precision_scores.append(precision_decision_tree)
  recall_scores.append(recall_decision_tree)
  f1_scores.append(f1_decision_tree)
  auc_scores.append(roc_auc_decision_tree)

# Create a single plot to visualize the scores with respect to max
depth
plt.figure(figsize=(10, 6))
plt.plot(max_depths, accuracy_scores, label='Accuracy', marker='o')
plt.plot(max_depths, precision_scores, label='Precision', marker='o')
plt.plot(max_depths, recall_scores, label='Recall', marker='o')
plt.plot(max_depths, f1_scores, label='F1 Score', marker='o')
plt.plot(max_depths, auc_scores, label='AUC', marker='o')

plt.xlabel('Max Depth')
plt.ylabel('Score')
plt.title('Evaluation Scores vs. Max Depth for Decision Tree
Classifier (Log loss, Train validation split)')
plt.xticks(np.arange(1, 16))
plt.legend()
plt.grid(True)
plt.show()
```

```
for max_depth = 1
Decision Tree Accuracy: 0.88
Decision Tree Precision: 1.00
Decision Tree Recall: 0.69
```

```
Decision Tree F1 Score: 0.82
AUC - Decision Tree Classifier: 0.85
************************************************************
for max_depth = 2
Decision Tree Accuracy: 0.87
Decision Tree Precision: 1.00
Decision Tree Recall: 0.67
Decision Tree F1 Score: 0.80
AUC - Decision Tree Classifier: 0.89
************************************************************
for max_depth = 3
Decision Tree Accuracy: 0.80
Decision Tree Precision: 0.70
Decision Tree Recall: 0.86
Decision Tree F1 Score: 0.78
AUC - Decision Tree Classifier: 0.92
************************************************************
for max_depth = 4
Decision Tree Accuracy: 0.81
Decision Tree Precision: 0.74
Decision Tree Recall: 0.81
Decision Tree F1 Score: 0.77
AUC - Decision Tree Classifier: 0.92
************************************************************
for max_depth = 5
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.74
Decision Tree Recall: 0.86
Decision Tree F1 Score: 0.79
AUC - Decision Tree Classifier: 0.91
************************************************************
for max_depth = 6
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.83
Decision Tree Recall: 0.69
Decision Tree F1 Score: 0.76
AUC - Decision Tree Classifier: 0.90
************************************************************
for max_depth = 7
Decision Tree Accuracy: 0.84
Decision Tree Precision: 0.82
Decision Tree Recall: 0.75
Decision Tree F1 Score: 0.78
AUC - Decision Tree Classifier: 0.88
************************************************************
for max_depth = 8
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.79
Decision Tree Recall: 0.75
```

```
Decision Tree F1 Score: 0.77
AUC - Decision Tree Classifier: 0.85
************************************************************
for max_depth = 9
Decision Tree Accuracy: 0.80
Decision Tree Precision: 0.74
Decision Tree Recall: 0.78
Decision Tree F1 Score: 0.76
AUC - Decision Tree Classifier: 0.81
************************************************************
for max_depth = 10
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.76
Decision Tree Recall: 0.81
Decision Tree F1 Score: 0.78
AUC - Decision Tree Classifier: 0.82
************************************************************
for max_depth = 11
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.75
Decision Tree Recall: 0.83
Decision Tree F1 Score: 0.79
AUC - Decision Tree Classifier: 0.82
************************************************************
for max_depth = 12
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.76
Decision Tree Recall: 0.81
Decision Tree F1 Score: 0.78
AUC - Decision Tree Classifier: 0.82
************************************************************
for max_depth = 13
Decision Tree Accuracy: 0.81
Decision Tree Precision: 0.74
Decision Tree Recall: 0.81
Decision Tree F1 Score: 0.77
AUC - Decision Tree Classifier: 0.81
************************************************************
for max_depth = 14
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.75
Decision Tree Recall: 0.83
Decision Tree F1 Score: 0.79
AUC - Decision Tree Classifier: 0.83
************************************************************
for max_depth = 15
Decision Tree Accuracy: 0.82
Decision Tree Precision: 0.75
Decision Tree Recall: 0.83
```

```
Decision Tree F1 Score: 0.79
AUC - Decision Tree Classifier: 0.83
************************************************************
```



Metrics for selection of most suitable max_depth parameter

1. For breast cancer prediction we should focus on reducing false negatives and we can tolerate false positives to an extent, this means we can priotize recall over precision.
2. For the other scores we can try to maximize them.

Based on the above metric, the most optimal value for max_depth is **5**. Decision Tree Accuracy: 0.82 Decision Tree Precision: 0.74 Decision Tree Recall: 0.86 Decision Tree F1 Score: 0.79 AUC - Decision Tree Classifier: 0.91

Decision Tree Accuracy: 0.80 Decision Tree Precision: 0.70 Decision Tree Recall: 0.86 Decision Tree F1 Score: 0.78 AUC - Decision Tree Classifier: 0.92

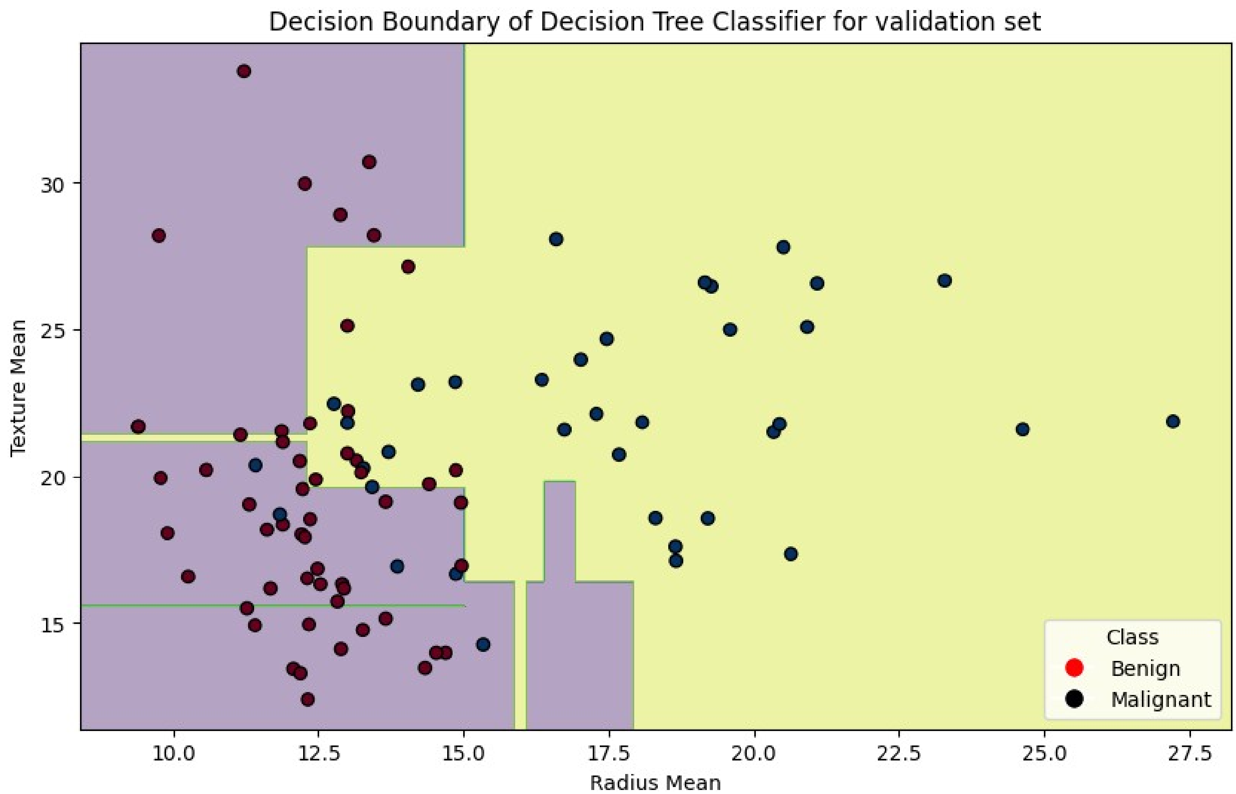
Observations

1. Here we can see that the values obtained from k fold and train validation split are very close to what we obtained from train validation split.
2. Here max_depth = 5 is the most optimal values as per the metric that we used earlier.

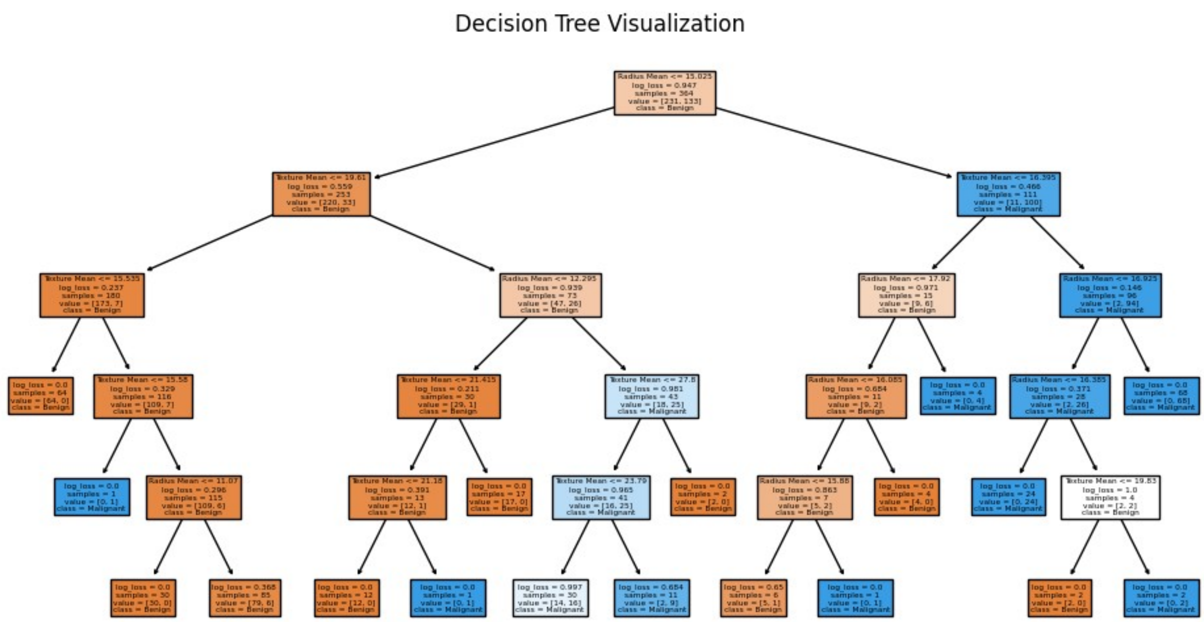
```
# Create a decision tree classifier
decision_tree_model_log_loss_5 =
DecisionTreeClassifier(criterion='log_loss', random_state=42,
```

```
max_depth = 5)
# Train the model on the training data
depth_max_cls = decision_tree_model_log_loss_5.fit(X_train_new,
y_train_new)

# Define a mesh grid of points to plot the decision boundary
x_min, x_max = X_val.iloc[:, 0].min() - 1, X_val.iloc[:, 0].max() + 1
y_min, y_max = X_val.iloc[:, 1].min() - 1, X_val.iloc[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min,
y_max, 0.01))

# Use the classifier to predict the class labels for each point in the
mesh grid
Z = depth_max_cls.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.4)

# Plot the data points
plt.scatter(X_val.iloc[:, 0], X_val.iloc[:, 1], c=y_val,
cmap=plt.cm.RdBu, edgecolor='k')
plt.xlabel('Radius Mean')
plt.ylabel('Texture Mean')
plt.title('Decision Boundary of Decision Tree Classifier for
validation set')

legend1 = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='red', markersize=10, label='Benign')
legend2 = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='black', markersize=10, label='Malignant')
plt.legend(handles=[legend1, legend2], title='Class', loc='lower
right')

plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```




```
# Plot the decision tree
plt.figure(figsize=(12, 6))
plot_tree(decision_tree_model_log_loss_5, filled=True,
feature_names=['Radius Mean', 'Texture Mean'], class_names=['Benign',
'Malignant'])
plt.title("Decision Tree Visualization")
plt.show()
```



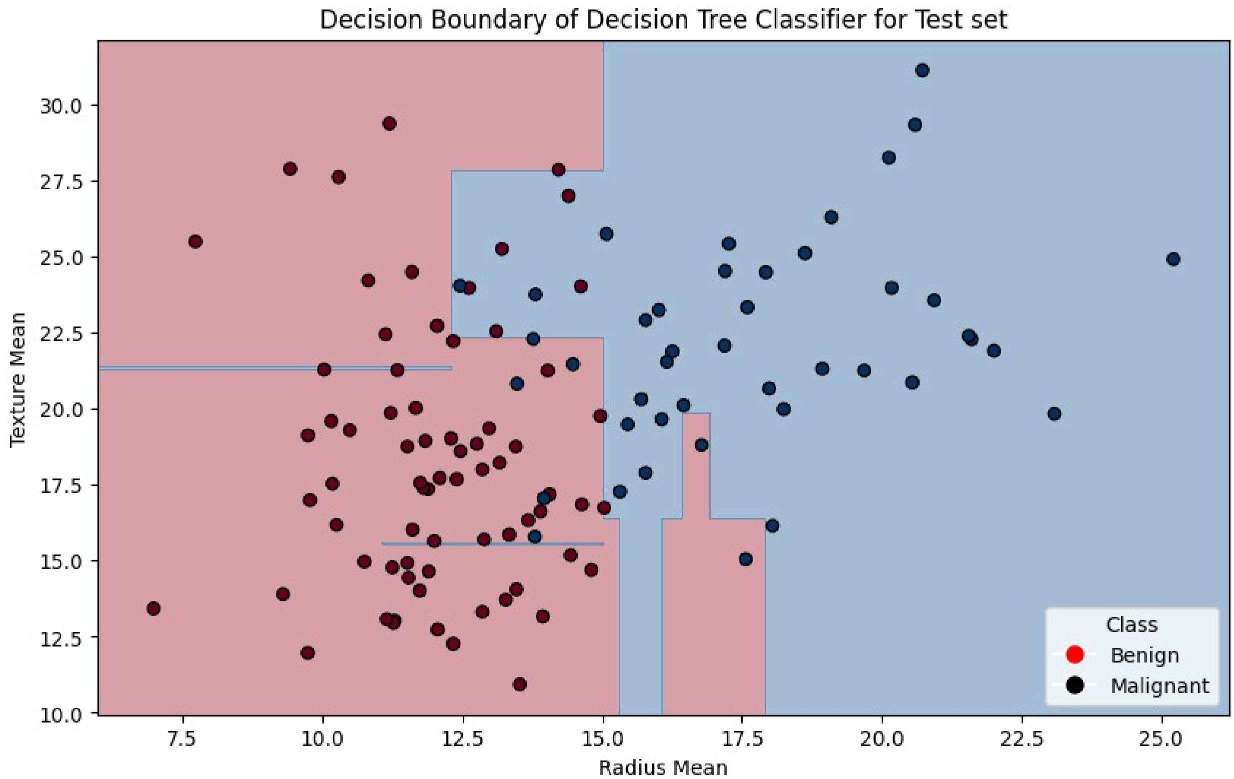
Test Error

```
# Create a decision tree classifier
decision_tree_model_log_loss =
DecisionTreeClassifier(criterion='log_loss', random_state=42,
max_depth = 5)
# Train the model on the training data
depth = decision_tree_model_log_loss.fit(X_train, y_train)
# Make predictions on the test data
print(f'For test dataset using Log Loss criterion')
y_pred_decision_tree = decision_tree_model_log_loss.predict(X_test)
accuracy_decision_tree = accuracy_score(y_test, y_pred_decision_tree)
print(f'Decision Tree Accuracy: {accuracy_decision_tree:.2f}')
precision_decision_tree = precision_score(y_test,
y_pred_decision_tree)
print(f'Decision Tree Precision: {precision_decision_tree:.2f}')
recall_decision_tree = recall_score(y_test, y_pred_decision_tree)
print(f'Decision Tree Recall: {recall_decision_tree:.2f}')
f1_decision_tree = f1_score(y_test, y_pred_decision_tree)
print(f'Decision Tree F1 Score: {f1_decision_tree:.2f}')

# Get predicted probabilities for class 1 (Malignant) from the
decision tree model
y_prob_decision_tree =
decision_tree_model_log_loss.predict_proba(X_test)[:, 1]
# Calculate ROC curve for the decision tree model
fpr_decision_tree, tpr_decision_tree, _ = roc_curve(y_test,
y_prob_decision_tree)
# Calculate AUC for the decision tree model
roc_auc_decision_tree = auc(fpr_decision_tree, tpr_decision_tree)
```

```
print(f'AUC - Decision Tree Classifier: {roc_auc_decision_tree:.2f}')
print('*************************************************************
****************************')


# Define a mesh grid of points to plot the decision boundary
x_min, x_max = X_test.iloc[:, 0].min() - 1, X_test.iloc[:, 0].max() +
1
y_min, y_max = X_test.iloc[:, 1].min() - 1, X_test.iloc[:, 1].max() +
1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min,
y_max, 0.01))

# Use the classifier to predict the class labels for each point in the
mesh grid
Z = depth.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.RdBu)

# Plot the data points
plt.scatter(X_test.iloc[:, 0], X_test.iloc[:, 1], c=y_test,
cmap=plt.cm.RdBu, edgecolor='k')
plt.xlabel('Radius Mean')
plt.ylabel('Texture Mean')
plt.title('Decision Boundary of Decision Tree Classifier for Test
set')

legend1 = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='red', markersize=10, label='Benign')
legend2 = plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='black', markersize=10, label='Malignant')
plt.legend(handles=[legend1, legend2], title='Class', loc='lower
right')

plt.show()
```

```
For test dataset using Log Loss criterion
Decision Tree Accuracy: 0.88
Decision Tree Precision: 0.84
Decision Tree Recall: 0.84
Decision Tree F1 Score: 0.84
AUC - Decision Tree Classifier: 0.93
**********************************************************************
*********************

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439:
UserWarning: X does not have valid feature names, but
```

```
DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```



Conclusion

For test dataset using GINI creterion

- Decision Tree Accuracy: 0.89
- Decision Tree Precision: 0.81
- Decision Tree Recall: 0.91
- Decision Tree F1 Score: 0.86
- AUC - Decision Tree Classifier: 0.90

For test dataset using Entropy criterion

- Decision Tree Accuracy: 0.88
- Decision Tree Precision: 0.84
- Decision Tree Recall: 0.84
- Decision Tree F1 Score: 0.84
- AUC - Decision Tree Classifier: 0.93

For test dataset using Log Loss criterion

- Decision Tree Accuracy: 0.88
- Decision Tree Precision: 0.84

- Decision Tree Recall: 0.84
- Decision Tree F1 Score: 0.84
- AUC - Decision Tree Classifier: 0.93

1. Interestingly, the Entropy and Log Loss scoring are the same; I could not understand why.
2. Between the GINI index and Entropy/log loss, I would go with GINI because, as per metric, we will prioritize recall over precision because, in the case of breast cancer, we want fewer false negative predictions, whereas false positives are not concerning.
3. The GINI criterion gives more recall than the other two, while other scorings are comparable.
4. Here are some considerations for using the Gini index over entropy for a breast cancer dataset:

- Sensitivity to Outliers: The Gini index tends to favor larger partitions with impurity, making it less sensitive to outliers. In a breast cancer dataset, outliers could represent unusual cases that are not indicative of the majority. Gini may perform better in such cases.

- Simplicity: Gini index calculations are often computationally simpler than entropy, making them faster to compute. This could be advantageous for larger datasets or when time efficiency is a concern.

- Interpretability: The Gini index's values are more intuitive to interpret, as they represent the probability of misclassifying a randomly chosen element from a set. In medical contexts like breast cancer diagnosis, interpretability can be crucial for understanding the decision tree's logic.