```
# 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

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	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perimeter_worst	area_
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	

5 rows × 32 columns

```
# 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)
```

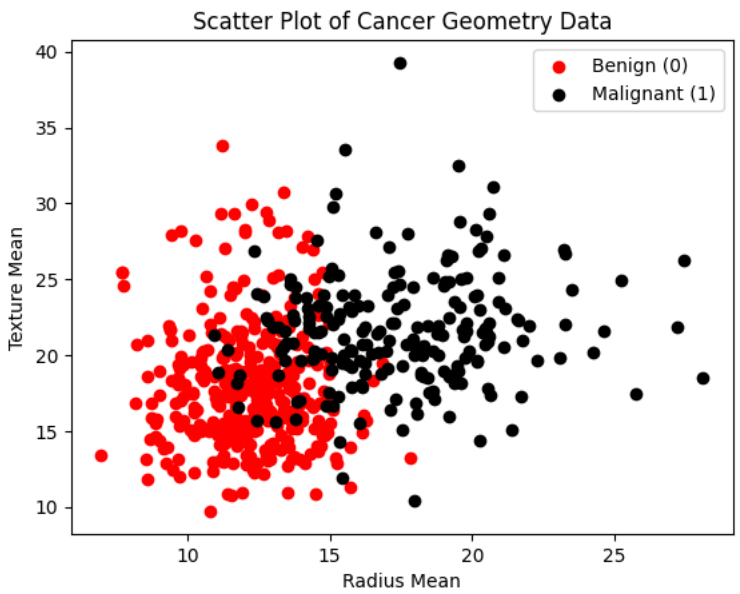
```
# Encode labels
data["diagnosis"] = data["diagnosis"].map({"B": 0, "M": 1})
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perimeter_worst	area_
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	

5 rows × 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)
```

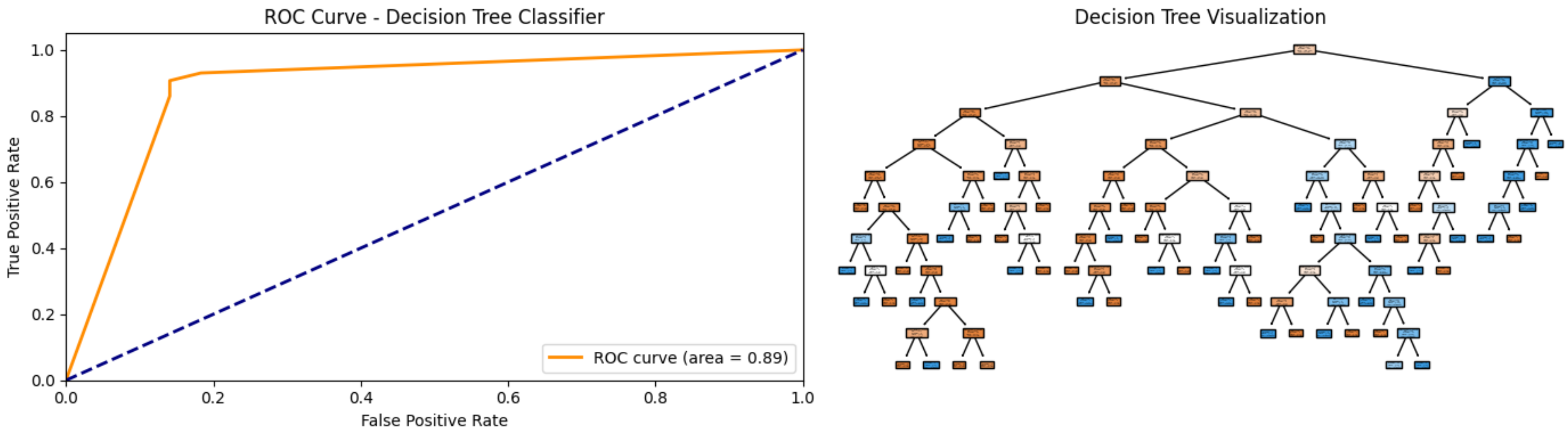


Decision Tree Classifier GINI creterion

```
# Create a decision tree classifier
decision_tree_model_GINI = DecisionTreeClassifier(criterion='gini', random_state=42)
# Train the model on the training data
decision_tree_model_GINI.fit(X_train, y_train)
# Make predictions on the test data
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}')
```

```
Decision Tree Accuracy: 0.88
Decision Tree Precision: 0.80
Decision Tree Recall: 0.91
Decision Tree F1 Score: 0.85
AUC - Decision Tree Classifier: 0.88
```

```
# Plot ROC curve for the decision tree model
plt.figure(figsize=(14, 4))

# Subplot for ROC curve
plt.subplot(121)
plt.plot(fpr_decision_tree, tpr_decision_tree, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc_decision_tree)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree Classifier')
plt.legend(loc='lower right')

# Subplot for decision tree visualization
plt.subplot(122)
plot_tree(decision_tree_model_GINI, filled=True, feature_names=['Radius Mean', 'Texture Mean'], class_names=['Benign', 'Malignant'])
plt.title("Decision Tree Visualization")

plt.tight_layout()  # Ensure proper spacing between subplots
plt.show()
```



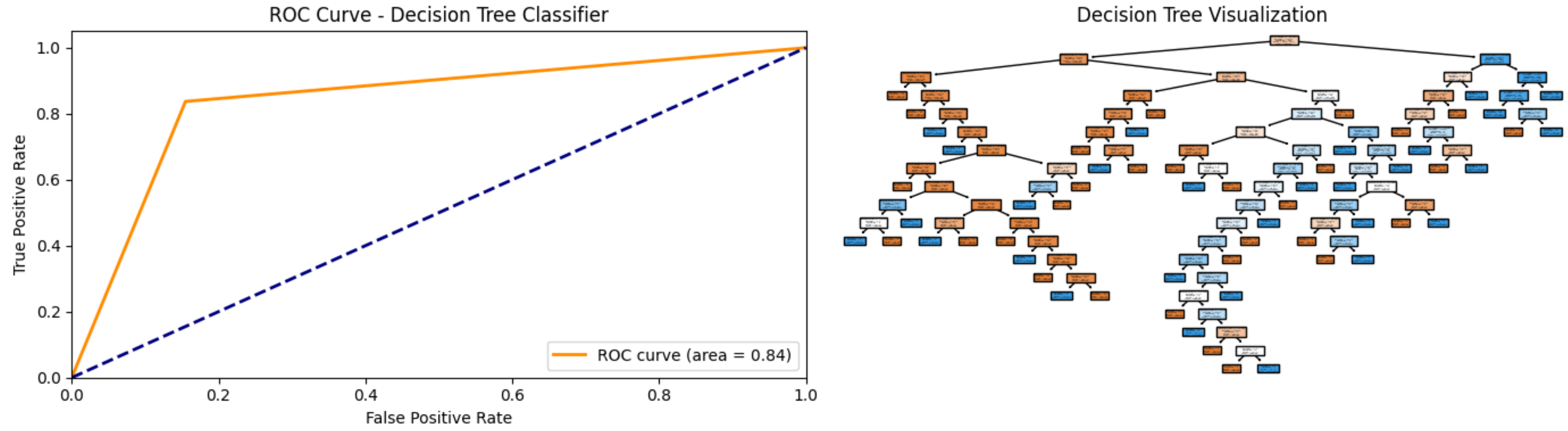
Decision Tree Classifier Entropy creterion

```
# Create a decision tree classifier
decision_tree_model_entropy = DecisionTreeClassifier(criterion='entropy', random_state=42)
# Train the model on the training data
decision_tree_model_entropy.fit(X_train, y_train)
# Make predictions on the test data
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)
print(f'AUC - Decision Tree Classifier: {roc_auc_decision_tree:.2f}')
```

```
Decision Tree Accuracy: 0.84
Decision Tree Precision: 0.77
Decision Tree Recall: 0.84
Decision Tree F1 Score: 0.80
AUC - Decision Tree Classifier: 0.84
```

```
# Plot ROC curve for the decision tree model
plt.figure(figsize=(14, 4))

# Subplot for ROC curve
plt.subplot(121)
plt.plot(fpr_decision_tree, tpr_decision_tree, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc_decision_tree)
```

```
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree Classifier')
plt.legend(loc='lower right')

# Subplot for decision tree visualization
plt.subplot(122)
plot_tree(decision_tree_model_entropy, filled=True, feature_names=['Radius Mean', 'Texture Mean'], class_names=['Benign', 'Malignant'])
plt.title("Decision Tree Visualization")

plt.tight_layout()  # Ensure proper spacing between subplots
plt.show()
```

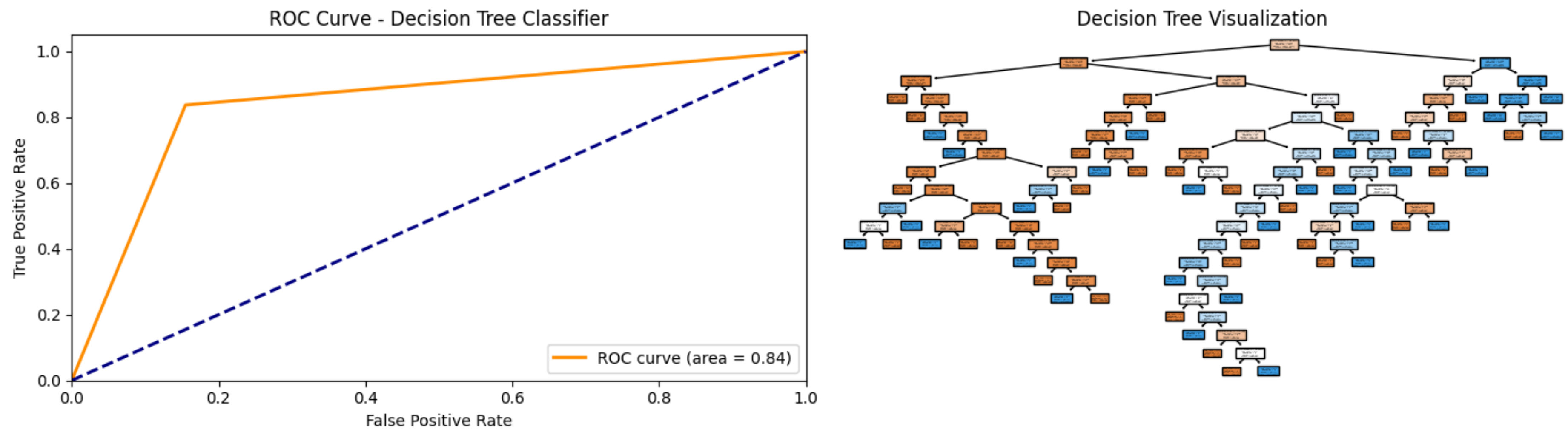


Decision Tree Classifier Log loss creterion

```
# Create a decision tree classifier
decision_tree_model_log_loss = DecisionTreeClassifier(criterion='log_loss', random_state=42)
# Train the model on the training data
decision_tree_model_log_loss.fit(X_train, y_train)
# Make predictions on the test data
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}')
```

```
Decision Tree Accuracy: 0.84
Decision Tree Precision: 0.77
Decision Tree Recall: 0.84
Decision Tree F1 Score: 0.80
AUC - Decision Tree Classifier: 0.84
```

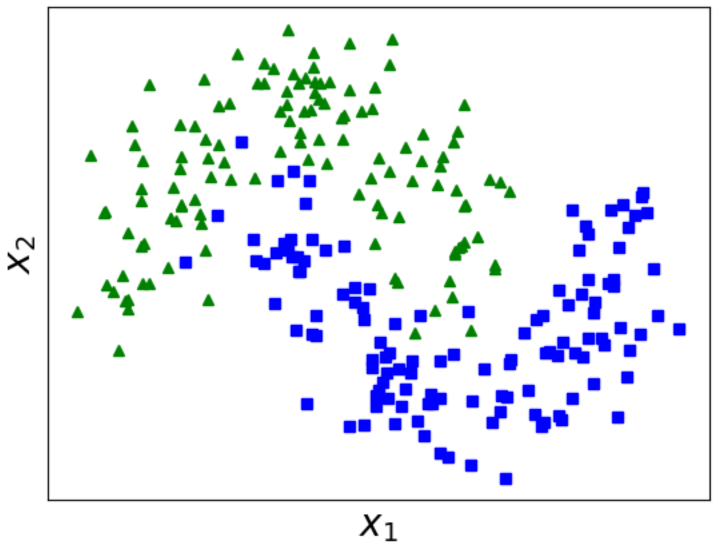
```
# Plot ROC curve for the decision tree model
plt.figure(figsize=(14, 4))

# Subplot for ROC curve
plt.subplot(121)
plt.plot(fpr_decision_tree, tpr_decision_tree, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc_decision_tree)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree Classifier')
plt.legend(loc='lower right')

# Subplot for decision tree visualization
plt.subplot(122)
plot_tree(decision_tree_model_log_loss, filled=True, feature_names=['Radius Mean', 'Texture Mean'], class_names=['Benign', 'Malignant'])
plt.title("Decision Tree Visualization")

plt.tight_layout()  # Ensure proper spacing between subplots
plt.show()
```



Hyper-parameters in Decision Trees

```
X,y = make_moons(n_samples=250,noise=0.2,random_state=42)
```

```
# plotting the data
plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs")
plt.plot(X[:, 0][y==0], X[:, 1][y==0], "g^")
plt.xlabel(r"$x_1$", fontsize=20)
plt.ylabel(r"$x_2$", fontsize=20)
plt.xticks([])
plt.yticks([])
plt.show()
```



Max-depth

maximum depth of the decision tree formed

```
depth_1_clf = DecisionTreeClassifier(max_depth=10 ).fit(X,y)
depth_2_clf = DecisionTreeClassifier(max_depth=3 ).fit(X,y)
```

```
# Plotting the decision boundary
ax = plt.subplot(1, 2, 1)
ax.set_title('Max Depth = 10')
plt.tight_layout(h_pad=5, w_pad=5, pad=2.5)
DecisionBoundaryDisplay.from_estimator(depth_1_clf, X, cmap=plt.cm.RdYlBu, response_method="predict", ax=ax, xlabel=r"$x_1$",ylabel=r"$x_2$")

# Plot the training points
for i, color in zip(range(2), 'rb'):
  idx = np.where(y == i)
  plt.scatter(X[idx, 0], X[idx, 1], c=color, cmap=plt.cm.RdBu, edgecolor="black", s=15,)


ax = plt.subplot(1, 2, 2)
ax.set_title('Max Depth = 3')
plt.tight_layout(h_pad=5, w_pad=5, pad=2.5)
DecisionBoundaryDisplay.from_estimator(depth_2_clf, X, cmap=plt.cm.RdYlBu, response_method="predict", ax=ax, xlabel=r"$x_1$",ylabel=r"$x_2$")

# Plot the training points
for i, color in zip(range(2), 'rb'):
  idx = np.where(y == i)
  plt.scatter(X[idx, 0], X[idx, 1], c=color, cmap=plt.cm.RdBu, edgecolor="black", s=15,)
```

```
<ipython-input-73-5136fc7c09f3>:10: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(X[idx, 0], X[idx, 1], c=color, cmap=plt.cm.RdBu, edgecolor="black", s=15,)
<ipython-input-73-5136fc7c09f3>:21: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(X[idx, 0], X[idx, 1], c=color, cmap=plt.cm.RdBu, edgecolor="black", s=15,)
```