Task

- Download balance-scale dataset from: https://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balance-scale.data
- Explore it.
- How many features and samples are there?

```
import pandas as pd
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balance-scale.dat
```

```
data.head(10)
```

	Class	Left-Weight	Left-Distance	Right-Weight	Right-Distance
0	B	1	1	1	1
1	R	1	1	1	2
2	R	1	1	1	3
3	R	1	1	1	4
4	R	1	1	1	5
5	R	1	1	2	1
6	R	1	1	2	2
7	R	1	1	2	3
8	R	1	1	2	4
9	R	1	1	2	5

```
data.shape
```

```
(625, 5)
```

Task

- Divide the data into X and y.
- How many classes are there?
- What are the possible values that all the features can have?
- Plot the distribution of the possible classes.

```
X = data.iloc[:, 1:]
y = data.iloc[:, 0]
```

```
print(X.shape)
print(y.shape)
```

```
(625, 4)
(625,)
```

```
set(y)
```

```
{'B', 'L', 'R'}
```

```
set(X.iloc[:,0])
```

```
{1, 2, 3, 4, 5}
```

```
set(X.iloc[:,1])
```

```
{1, 2, 3, 4, 5}
```

```
set(X.iloc[:,2])
```

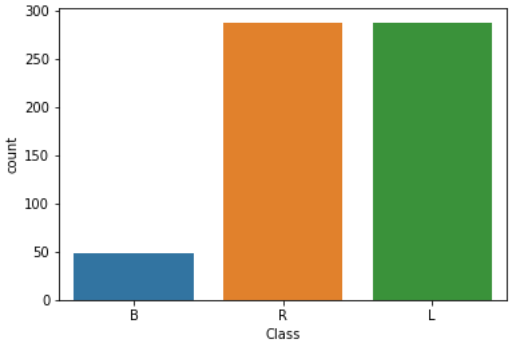
```
{1, 2, 3, 4, 5}
```

```
set(X.iloc[:,3])
```

```
{1, 2, 3, 4, 5}
```

```
import seaborn as sns
sns.countplot(y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49ee684790>
```



Task

- Divide X and y into train and test with test_size = 0.2

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
print(X_train.shape)
print(X_test.shape)
```

```
(500, 4)
(125, 4)
```

Task

- Train a DecisionTreeClassifier model with random_state = 0 and default values.
- What are the train and test accuracy scores?

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(random_state = 0)
```

```
dtc.fit(X_train, y_train)
train_score = dtc.score(X_train, y_train)
test_score = dtc.score(X_test, y_test)
print(train_score)
print(test_score)
```

```
1.0
0.752
```

Task

- WHat is the depth of the tree?
- How many nodes are there in the tree?
- Plot the first 2 levels of the tree.

```
dtc.get_depth()
```

```
11
```

```
dtc.tree_.max_depth
```

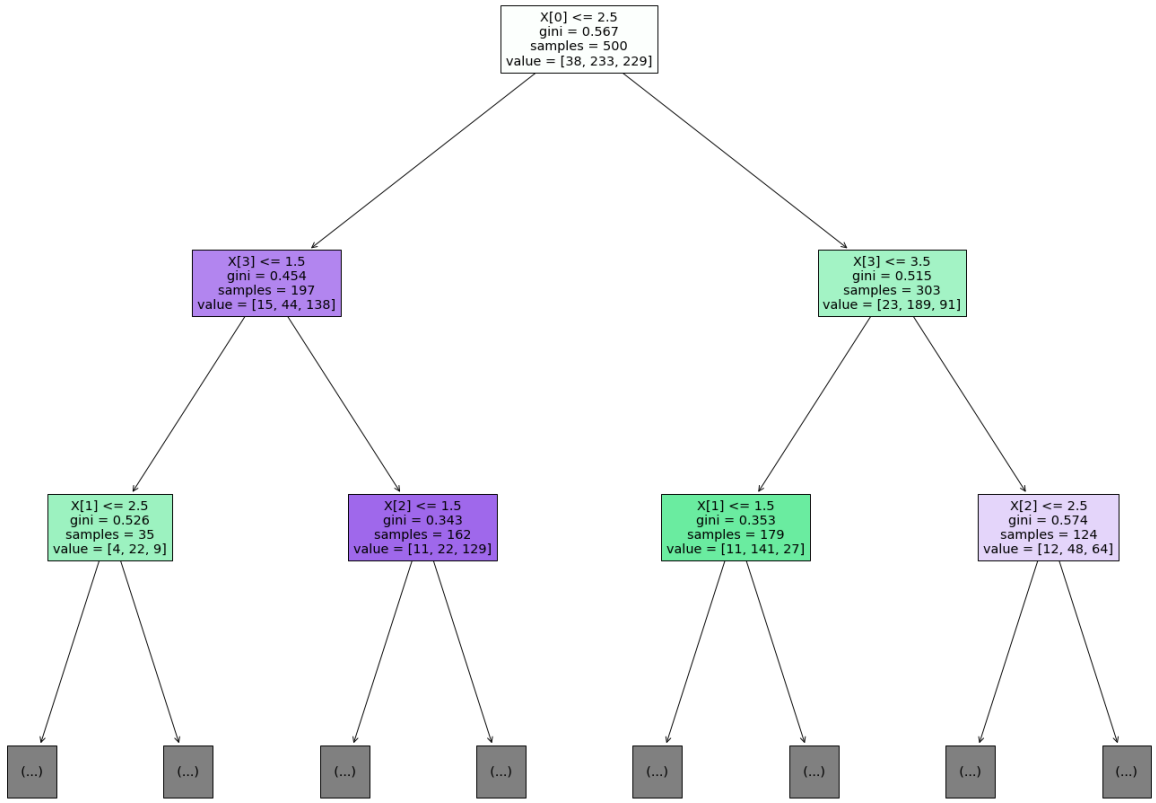
```
11
```

```
dtc.tree_.node_count
```

```
251
```

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
plt.figure(figsize=(25,20))
plot_tree(dtc, max_depth = 2, filled = True)
```

```
[Text(0.5, 0.875, 'X[0] <= 2.5\ngini = 0.567\nsamples = 500\nvalue = [38, 233, 229]'),
 Text(0.25, 0.625, 'X[3] <= 1.5\ngini = 0.454\nsamples = 197\nvalue = [15, 44, 138]'),
 Text(0.125, 0.375, 'X[1] <= 2.5\ngini = 0.526\nsamples = 35\nvalue = [4, 22, 9]'),
 Text(0.0625, 0.125, '\n  (...)  \n'),
 Text(0.1875, 0.125, '\n  (...)  \n'),
 Text(0.375, 0.375, 'X[2] <= 1.5\ngini = 0.343\nsamples = 162\nvalue = [11, 22, 129]'),
 Text(0.3125, 0.125, '\n  (...)  \n'),
 Text(0.4375, 0.125, '\n  (...)  \n'),
 Text(0.75, 0.625, 'X[3] <= 3.5\ngini = 0.515\nsamples = 303\nvalue = [23, 189, 91]'),
 Text(0.625, 0.375, 'X[1] <= 1.5\ngini = 0.353\nsamples = 179\nvalue = [11, 141, 27]'),
 Text(0.5625, 0.125, '\n  (...)  \n'),
 Text(0.6875, 0.125, '\n  (...)  \n'),
 Text(0.875, 0.375, 'X[2] <= 2.5\ngini = 0.574\nsamples = 124\nvalue = [12, 48, 64]'),
 Text(0.8125, 0.125, '\n  (...)  \n'),
 Text(0.9375, 0.125, '\n  (...)  \n')]
```



Task

- Try all max_depth values between 1 and 15 and create those many classifiers.
- Store the train and test score on each classifier in two lists.

```
import numpy as np
max_depths = np.arange(1,16)
train_scores = []
test_scores = []
for max_depth in max_depths:
   dt = DecisionTreeClassifier(max_depth=max_depth, random_state = 0)
   dt.fit(X_train, y_train)
   train_scores.append(dt.score(X_train, y_train))
```

```
    test_scores.append(dt.score(X_test, y_test))

print(train_scores)
print(test_scores)
```

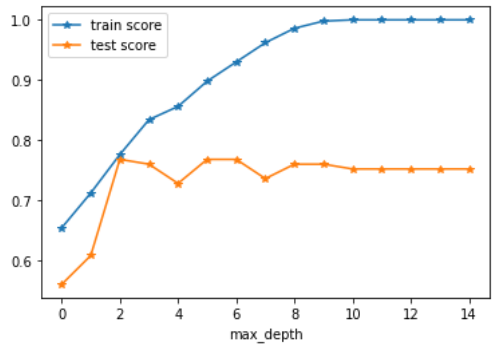
```
[0.654, 0.712, 0.776, 0.834, 0.856, 0.898, 0.93, 0.962, 0.986, 0.998, 1.0, 1.0, 1.0, 1.0, 1.0]
[0.56, 0.608, 0.768, 0.76, 0.728, 0.768, 0.768, 0.736, 0.76, 0.76, 0.752, 0.752, 0.752, 0.752, 0.752]
```

Task

- Plot train and test scores, with max_depth being on x_axis.
- At what value of max_depth are train and test scores comparable?

```
import matplotlib.pyplot as plt
plt.plot(train_scores, label = 'train score', marker = '*')
plt.plot(test_scores, label = 'test score', marker = '*')
plt.xlabel('max_depth')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f49eb167350>
```



Task

- Build a new model with the max_depth observed above (=3) and random_state = 0.
- Print the train and test scores
- Is there overfitting?
- How many instances of 'balanced' target in the test set are correctly predicted?

```
dtc1 = DecisionTreeClassifier(max_depth=3, random_state = 0)
```

```
dtc1.fit(X_train,y_train)
```

```
    DecisionTreeClassifier(max_depth=3, random_state=0)
```

```
train_score1 = dtc1.score(X_train, y_train)
test_score1 = dtc1.score(X_test, y_test)

print(train_score1)
print(test_score1)
```

```
    0.776
    0.768
```

```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(dtc1, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f49eb0e8610>
```

Task

- Build a new model with the ccp_alpha observed above (=0.02) and random_state = 0.
- Print the train and test scores
- Is there overfitting?
- How many instances of 'balanced' target in the test set are correctly predicted?

```
dtc2 = DecisionTreeClassifier(ccp_alpha = 0.02, random_state = 0)


dtc2.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.02, random_state=0)
```

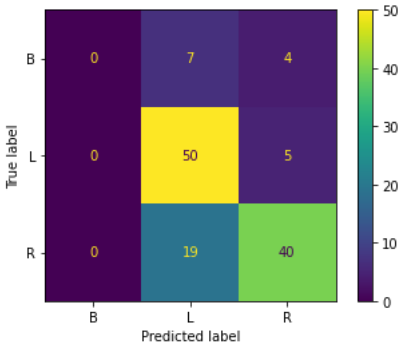
```
train_score2 = dtc2.score(X_train, y_train)
test_score2 = dtc2.score(X_test, y_test)

print(train_score2)
print(test_score2)
```

```
0.776
0.72
```

```
plot_confusion_matrix(dtc2, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f49ecb99d90>
```



Task

- Build a new model with the ccp_alpha observed above (=0.02) and random_state = 0 and class_weight to be balanced.
- Print the train and test scores
- Is there overfitting?
- How many instances of 'balanced' target in the test set are correctly predicted?

```
dtc3 = DecisionTreeClassifier(ccp_alpha = 0.02, class_weight = 'balanced', random_state = 0)


dtc3.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.02, class_weight='balanced', random_state=0)
```

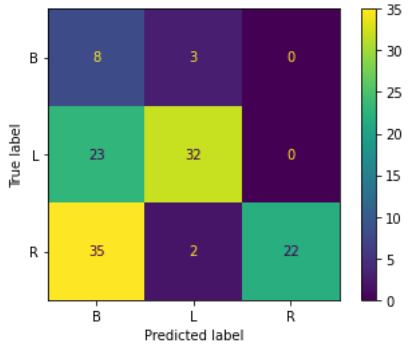
```
train_score3 = dtc3.score(X_train, y_train)
test_score3 = dtc3.score(X_test, y_test)

print(train_score3)
print(test_score3)
```

```
0.594
0.496
```

```
plot_confusion_matrix(dtc3, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f49ead0cd50>
```



Task

- Build a new model with the ccp_alpha observed above (=0.02) and random_state = 0 and class_weight to be {'B': 1, 'L': 0.1, 'R':0.1}.
- Print the train and test scores
- Is there overfitting?
- How many instances of 'balanced' target in the test set are correctly predicted?

```
dtc4 = DecisionTreeClassifier(ccp_alpha = 0.02, class_weight = {'B': 1, 'L': 0.1, 'R':0.1}, random_state = 0)
```

```
dtc4.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.02,
                       class_weight={'B': 1, 'L': 0.1, 'R': 0.1},
                       random_state=0)
```

```
train_score4 = dtc4.score(X_train, y_train)
test_score4 = dtc4.score(X_test, y_test)

print(train_score4)
print(test_score4)
```

```
0.598
0.44
```

```
plot_confusion_matrix(dtc4, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f49eac01710>
```