classifying-penguins

February 13, 2025

1 Classifying Penguins with Decision Trees

Load the penguins dataset - as before, we'll use Seaborn dataset:

[1]:
```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
penguins_df = sns.load_dataset("penguins")
penguins_df.head()
```

[2]:
```
  species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0  Adelie  Torgersen            39.1           18.7              181.0
1  Adelie  Torgersen            39.5           17.4              186.0
2  Adelie  Torgersen            40.3           18.0              195.0
3  Adelie  Torgersen             NaN            NaN                NaN
4  Adelie  Torgersen            36.7           19.3              193.0

   body_mass_g     sex
0       3750.0    Male
1       3800.0  Female
2       3250.0  Female
3          NaN     NaN
4       3450.0  Female
```

[3]:
```
len(penguins_df)
```

[3]:
```
344
```

[4]:
```
penguins_df.describe().T
```

[4]:
```
                   count         mean         std     min       25%      50%  \
bill_length_mm     342.0    43.921930    5.459584    32.1    39.225    44.45
bill_depth_mm      342.0    17.151170    1.974793    13.1    15.600    17.30
flipper_length_mm  342.0   200.915205   14.061714   172.0   190.000   197.00
body_mass_g        342.0  4201.754386  801.954536  2700.0  3550.000  4050.00

                      75%     max
```

```
bill_length_mm       48.5    59.6
bill_depth_mm        18.7    21.5
flipper_length_mm   213.0   231.0
body_mass_g        4750.0  6300.0
```

[5]:
```
penguins_df.isnull().sum()
```

[5]:
```
species               0
island                0
bill_length_mm        2
bill_depth_mm         2
flipper_length_mm     2
body_mass_g           2
sex                  11
dtype: int64
```

We've spotted some missing values. For this example we'll just get rid of rows with missing values.

[6]:
```
penguins_df = penguins_df.dropna()
```

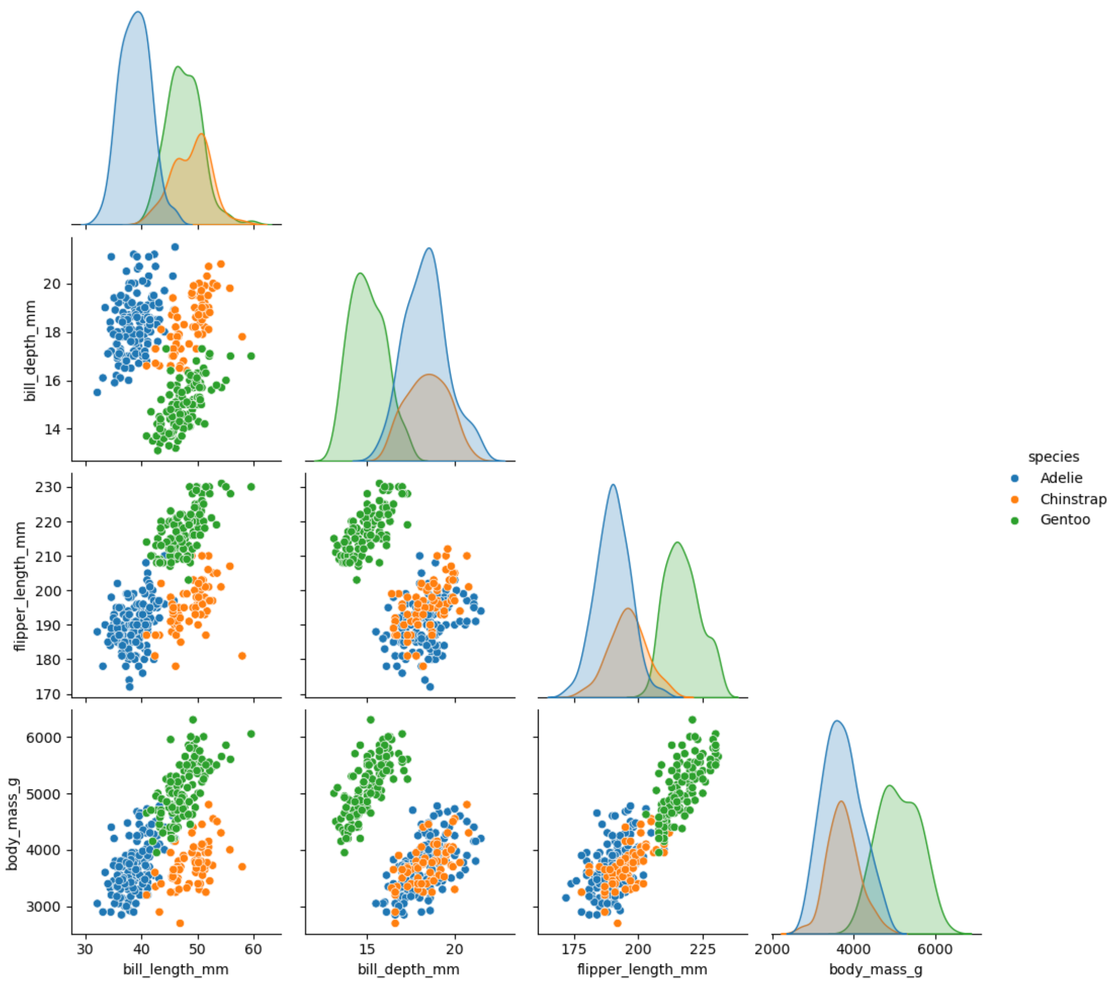
[7]:
```
len(penguins_df)
```

[7]:
```
333
```

Let's get a quick visual on the relationship between all pairs of numeric columns.

[8]:
```
sns.pairplot(data=penguins_df,
             hue="species",
             corner=True)
```

[8]:
```
<seaborn.axisgrid.PairGrid at 0x2bb88d25eb0>
```



1.1 Using a Decision Tree Classifier to Predict Penguin Species

[9]:
```
penguins_df.head()
```

[9]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	\
0	Adelie	Torgersen	39.1	18.7	181.0	
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	
4	Adelie	Torgersen	36.7	19.3	193.0	
5	Adelie	Torgersen	39.3	20.6	190.0	

	body_mass_g	sex
0	3750.0	Male
1	3800.0	Female
2	3250.0	Female
4	3450.0	Female

```
5       3650.0    Male
```

1.1.1 Encode categorical variables

The species, island, and sex variables are strings. This will cause a problem for our decision tree classifier which **works with numeric variables**. To solve this, we'll need to encode those variables.

See: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html#sklearn.prepro https://scikit-learn.org/stable/modules/preprocessing_targets.html and https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html#sklearn.preprocessing.OrdinalEr

The species is our label, so we'll use a LabelEncoder for that.

[10]:
```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
```

Let's first make a quick DataFrame with two columns: the original penguin species and it's new encoded value.

[11]:
```
encoding_df = pd.DataFrame({"species": penguins_df["species"],
                            "species_encoded": label_encoder.
 ↪fit_transform(penguins_df["species"])})
```

[12]:
```
encoding_df.sample(10)
```

[12]:
```
       species  species_encoded
190  Chinstrap                1
172  Chinstrap                1
195  Chinstrap                1
34      Adelie                0
78      Adelie                0
208  Chinstrap                1
156  Chinstrap                1
293     Gentoo                2
119     Adelie                0
95      Adelie                0
```

Now we'll replace the original text based penguin species label with it's numerically encoded version which will be compatible with our machine learning models.

[13]:
```
penguins_df['species'] = label_encoder.fit_transform(penguins_df['species'])
```

We can convert the male/female values in the sex column using an OrdinalEncoder. This will change them to 0 and 1.

[14]:
```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
```

Again, let's make a quick DataFrame to show the encoded values.

[15]:
```
encoded_df = pd.DataFrame({"sex": penguins_df["sex"]})
encoded_df[['encoded_value']] = ordinal_encoder.
 ↪fit_transform(encoded_df[['sex']])

encoded_df.head()
```

[15]:
```
      sex  encoded_value
0    Male            1.0
1  Female            0.0
2  Female            0.0
4  Female            0.0
5    Male            1.0
```

Now we'll go ahead and update the penguins sex column with the numerically encoded values.

[16]:
```
penguins_df[['sex']] = ordinal_encoder.fit_transform(penguins_df[['sex']])
```

[17]:
```
penguins_df.head()
```

[17]:
```
   species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0        0  Torgersen            39.1           18.7              181.0
1        0  Torgersen            39.5           17.4              186.0
2        0  Torgersen            40.3           18.0              195.0
4        0  Torgersen            36.7           19.3              193.0
5        0  Torgersen            39.3           20.6              190.0

   body_mass_g  sex
0       3750.0  1.0
1       3800.0  0.0
2       3250.0  0.0
4       3450.0  0.0
5       3650.0  1.0
```

It makes more sense to use a OneHotEncoder for our island column. As we'll see, this will make our decision tree more readable as we'll avoid questions like "Is the island less than 2?". Instead, we'll get three columns for island with a 1 in the column that corresponds to the original island value, and 0s for the other two.

[18]:
```
from sklearn.preprocessing import OneHotEncoder
```

[19]:
```
one_hot_encoder = OneHotEncoder()

island_encoded = one_hot_encoder.fit_transform(penguins_df[['island']])
```

The OneHotEncoder has given us a sparse matrix.

[20]:
```
island_encoded
```

[20]:
```
<333x3 sparse matrix of type '<class 'numpy.float64'>'
        with 333 stored elements in Compressed Sparse Row format>
```

Before converting this to a Pandas DataFrame, we need to convert it to a dense array.

[21]:
```
island_encoded = island_encoded.toarray()

island_encoded
```

[21]:
```
array([[0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
```

```
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
```

```
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 0., 1.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
```

```
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
```

```
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[0., 1., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
```

```
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
```

```
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
[1., 0., 0.],
```

```
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.]])
```

Now we can convert the one-hot encoded 'island' result to a DataFrame.

[22]:
```
island_encoded_df = pd.DataFrame(
    island_encoded, columns=one_hot_encoder.get_feature_names_out(['island'])
)

island_encoded_df
```

[22]:
```
     island_Biscoe  island_Dream  island_Torgersen
0              0.0           0.0               1.0
1              0.0           0.0               1.0
2              0.0           0.0               1.0
3              0.0           0.0               1.0
4              0.0           0.0               1.0
..             …             …               …
328            1.0           0.0               0.0
329            1.0           0.0               0.0
330            1.0           0.0               0.0
331            1.0           0.0               0.0
332            1.0           0.0               0.0

[333 rows x 3 columns]
```

We now bring the new one-hot encoded 'island' columns into the original DataFrame and drop the redundant 'island' column.

[23]:
```
penguins_df = pd.concat([penguins_df, island_encoded_df],
                        axis=1,  # axis=1 for columns
                        join='inner')  # without this we'll introduce NaNs
```

[24]:
```
penguins_df = penguins_df.drop(columns=['island'])
```

Now when we check out our DataFrame, we'll see that species, island, and sex are now encoded with numeric values.

[25]:
```
penguins_df.sample(15)
```

[25]:

	species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
209	1	49.3	19.9	203.0	4050.0
99	0	43.2	18.5	192.0	4100.0
309	2	52.1	17.0	230.0	5550.0
144	0	37.3	16.8	192.0	3000.0
27	0	40.5	17.9	187.0	3200.0
295	2	48.6	16.0	230.0	5800.0
136	0	35.6	17.5	191.0	3175.0
20	0	37.8	18.3	174.0	3400.0
259	2	48.7	15.7	208.0	5350.0
254	2	49.1	14.8	220.0	5150.0
306	2	43.4	14.4	218.0	4600.0
202	1	48.1	16.4	199.0	3325.0
325	2	46.8	16.1	215.0	5500.0
284	2	45.8	14.2	219.0	4700.0
56	0	39.0	17.5	186.0	3550.0

	sex	island_Biscoe	island_Dream	island_Torgersen
209	1.0	0.0	1.0	0.0
99	1.0	1.0	0.0	0.0
309	1.0	1.0	0.0	0.0
144	0.0	0.0	1.0	0.0
27	0.0	0.0	1.0	0.0
295	1.0	1.0	0.0	0.0
136	0.0	0.0	1.0	0.0
20	0.0	1.0	0.0	0.0
259	1.0	1.0	0.0	0.0
254	0.0	1.0	0.0	0.0
306	0.0	1.0	0.0	0.0
202	0.0	0.0	1.0	0.0
325	1.0	1.0	0.0	0.0
284	0.0	1.0	0.0	0.0
56	0.0	1.0	0.0	0.0

1.2 Training and test data

1.2.1 Define X and y

We want to be able to predict a penguin's species from the observations about that penguin - the "features". Our target variable is species, which is conventionally called y. The features are conventionally called X (upper case).

[26]:
```
X = penguins_df.drop('species', axis=1)
y = penguins_df['species']
```

[27]:
```
X.head()
```

[27]:

```
   bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g  sex  \
0            39.1           18.7              181.0       3750.0  1.0
1            39.5           17.4              186.0       3800.0  0.0
2            40.3           18.0              195.0       3250.0  0.0
4            36.7           19.3              193.0       3450.0  0.0
5            39.3           20.6              190.0       3650.0  1.0

   island_Biscoe  island_Dream  island_Torgersen
0            0.0           0.0               1.0
1            0.0           0.0               1.0
2            0.0           0.0               1.0
4            0.0           0.0               1.0
5            0.0           0.0               1.0
```

[28]:

```
y.head()
```

[28]:

```
0    0
1    0
2    0
4    0
5    0
Name: species, dtype: int32
```

1.2.2 Split the dataset into training and testing sets

We are going to build our decision tree using the penguin data, but we'll set aside some of that data so that we can test how accurate our decision tree is on examples that weren't usef in its construction. For this we can use the train_test_split function.

[29]:

```
from sklearn.model_selection import train_test_split
```

[30]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.3,
    random_state=42)
```

[31]:

```
X_train.head()
```

[31]:

```
     bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g  sex  \
268            44.9           13.3              213.0       5100.0  0.0
87             36.9           18.6              189.0       3500.0  0.0
154            51.3           19.2              193.0       3650.0  1.0
44             37.0           16.9              185.0       3000.0  0.0
229            46.8           15.4              215.0       5150.0  1.0

     island_Biscoe  island_Dream  island_Torgersen
268            1.0           0.0               0.0
```

```
87            0.0          1.0               0.0
154           0.0          1.0               0.0
44            1.0          0.0               0.0
229           1.0          0.0               0.0
```

[32]:
```
y_train.head()
```

[32]:
```
268    2
87     0
154    1
44     0
229    2
Name: species, dtype: int32
```

[33]:
```
y_train.isnull().sum()
```

[33]:
```
0
```

[34]:
```
X_test.head()
```

[34]:
```
     bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g  sex  \
138            37.0           16.5              185.0       3400.0  0.0
114            39.6           20.7              191.0       3900.0  0.0
143            40.7           17.0              190.0       3725.0  1.0
14             34.6           21.1              198.0       4400.0  1.0
186            49.7           18.6              195.0       3600.0  1.0

     island_Biscoe  island_Dream  island_Torgersen
138            0.0           1.0               0.0
114            0.0           0.0               1.0
143            0.0           1.0               0.0
14             0.0           0.0               1.0
186            0.0           1.0               0.0
```

[35]:
```
y_test.head()
```

[35]:
```
138    0
114    0
143    0
14     0
186    1
Name: species, dtype: int32
```

1.3 Create our decision tree

Scikit-learn makes this very easy for us with the DecisionTreeClassifier and its fit method.

[36]:
```
from sklearn.tree import DecisionTreeClassifier
```

[37]:
```
clf = DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
```

Now we use the classifier to predict species in the test set and calculate accuracy

[38]:
```
y_pred = clf.predict(X_test)
y_pred
```

[38]:
```
array([0, 0, 0, 0, 1, 0, 2, 1, 2, 0, 1, 1, 2, 0, 1, 2, 2, 0, 0, 0, 2, 2,
       0, 0, 0, 1, 1, 0, 0, 2, 0, 1, 2, 1, 2, 1, 0, 0, 0, 1, 0, 0, 2, 2,
       0, 2, 1, 2, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 2, 0, 0, 2, 0,
       1, 0, 0, 1, 2, 2, 2, 2, 2, 2, 0, 0, 1, 2, 0, 2, 1, 1, 2, 2, 0, 0,
       2, 0, 1, 0, 0, 2, 2, 0, 1, 1])
```

We now test the accuracy of our decision tree using the test set.

[39]:
```
from sklearn.metrics import accuracy_score
```

[40]:
```
accuracy_score(y_test, y_pred)
```

[40]:
```
0.9795918367346939
```

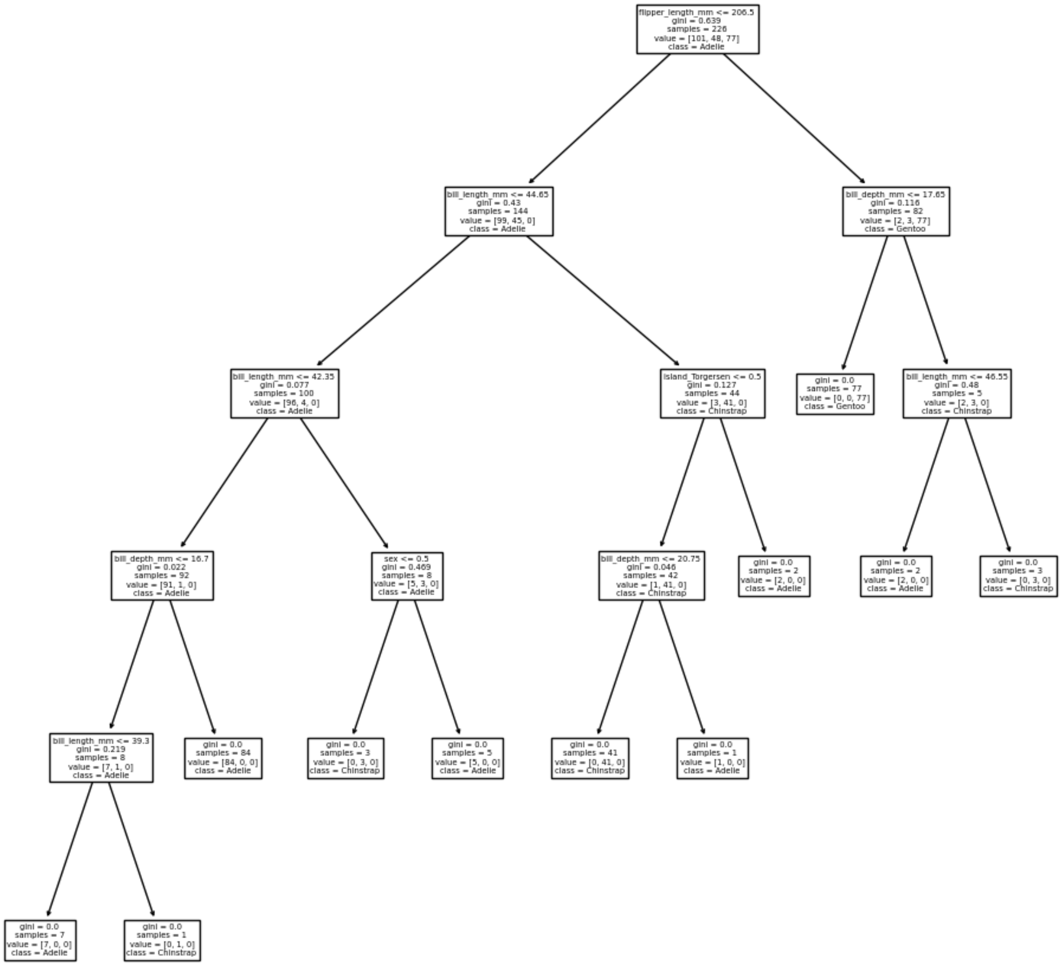
Visualise the decision tree

[41]:
```
from sklearn.tree import plot_tree
```

[42]:
```
plt.figure(figsize=(12, 12))  # set plot size (denoted in inches)
plot_tree(clf,
          feature_names=list(X.columns),
          class_names=['Adelie', 'Chinstrap', 'Gentoo'])
```

[42]:
```
[Text(0.6527777777777778, 0.9166666666666666, 'flipper_length_mm <= 206.5\ngini
= 0.639\nsamples = 226\nvalue = [101, 48, 77]\nclass = Adelie'),
 Text(0.4722222222222222, 0.75, 'bill_length_mm <= 44.65\ngini = 0.43\nsamples =
144\nvalue = [99, 45, 0]\nclass = Adelie'),
 Text(0.2777777777777778, 0.5833333333333334, 'bill_length_mm <= 42.35\ngini =
0.077\nsamples = 100\nvalue = [96, 4, 0]\nclass = Adelie'),
 Text(0.16666666666666666, 0.4166666666666667, 'bill_depth_mm <= 16.7\ngini =
0.022\nsamples = 92\nvalue = [91, 1, 0]\nclass = Adelie'),
 Text(0.1111111111111111, 0.25, 'bill_length_mm <= 39.3\ngini = 0.219\nsamples =
8\nvalue = [7, 1, 0]\nclass = Adelie'),
 Text(0.05555555555555555, 0.08333333333333333, 'gini = 0.0\nsamples = 7\nvalue
= [7, 0, 0]\nclass = Adelie'),
 Text(0.16666666666666666, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1, 0]\nclass = Chinstrap'),
 Text(0.2222222222222222, 0.25, 'gini = 0.0\nsamples = 84\nvalue = [84, 0,
0]\nclass = Adelie'),
 Text(0.3888888888888889, 0.4166666666666667, 'sex <= 0.5\ngini = 0.469\nsamples
= 8\nvalue = [5, 3, 0]\nclass = Adelie'),
```

```
 Text(0.3333333333333333, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 3,
0]\nclass = Chinstrap'),
 Text(0.4444444444444444, 0.25, 'gini = 0.0\nsamples = 5\nvalue = [5, 0,
0]\nclass = Adelie'),
 Text(0.6666666666666666, 0.5833333333333334, 'island_Torgersen <= 0.5\ngini =
0.127\nsamples = 44\nvalue = [3, 41, 0]\nclass = Chinstrap'),
 Text(0.6111111111111112, 0.4166666666666667, 'bill_depth_mm <= 20.75\ngini =
0.046\nsamples = 42\nvalue = [1, 41, 0]\nclass = Chinstrap'),
 Text(0.5555555555555556, 0.25, 'gini = 0.0\nsamples = 41\nvalue = [0, 41,
0]\nclass = Chinstrap'),
 Text(0.6666666666666666, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [1, 0,
0]\nclass = Adelie'),
 Text(0.7222222222222222, 0.4166666666666667, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0, 0]\nclass = Adelie'),
 Text(0.8333333333333334, 0.75, 'bill_depth_mm <= 17.65\ngini = 0.116\nsamples =
82\nvalue = [2, 3, 77]\nclass = Gentoo'),
 Text(0.7777777777777778, 0.5833333333333334, 'gini = 0.0\nsamples = 77\nvalue =
[0, 0, 77]\nclass = Gentoo'),
 Text(0.8888888888888888, 0.5833333333333334, 'bill_length_mm <= 46.55\ngini =
0.48\nsamples = 5\nvalue = [2, 3, 0]\nclass = Chinstrap'),
 Text(0.8333333333333334, 0.4166666666666667, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0, 0]\nclass = Adelie'),
 Text(0.9444444444444444, 0.4166666666666667, 'gini = 0.0\nsamples = 3\nvalue =
[0, 3, 0]\nclass = Chinstrap')]
```



We dropped 11 rows with missing values in the sex column, but it turns out that this column wasn't used. Maybe we could just drop the whole column and keep the rows that had data in the other columns. This would give us a little more data and might help to improve accuracy further.

1.3.1 Gini impurity

[43]:
```python
from collections import Counter

def gini_impurity(values):
    """
    Calculate the Gini impurity for a list of values.

    Args:
```

```
    values (list): A list of categorical values.

    Returns:
    float: The Gini impurity of the list.
    """
    counts = Counter(values)
    total = len(values)
    impurity = 1.0

    for label in counts:
        prob_of_label = counts[label] / total
        impurity -= prob_of_label ** 2
    return impurity
```

[44]:
```
labels = ['Gentoo', 'Gentoo', 'Adelie', 'Chinstrap']
#labels = ['Gentoo', 'Gentoo', 'Adelie', 'Adelie']
#labels = ['Gentoo', 'Gentoo', 'Gentoo', 'Gentoo']

gini_impurity(labels)
```

[44]: 0.625