Case study – Decision Tree Classifier

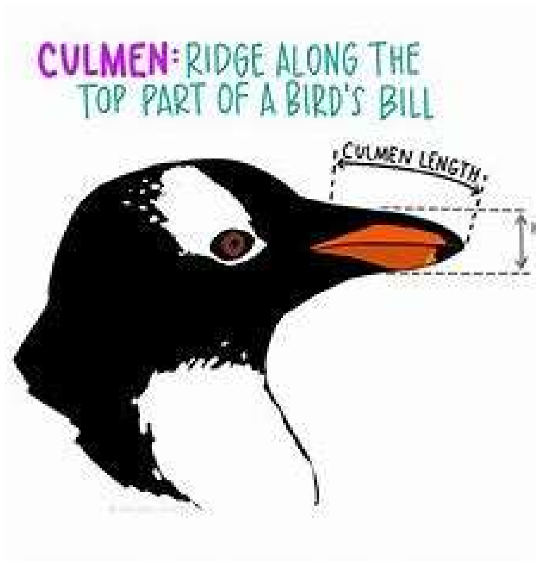
This case study will focus on Classification Logical Regression (CLR). CLR is a type of analysis where the model is trying to classify the output into more than two possible outcomes. For example, if we are analyzing ten type of roses. Based on flower characteristics identify what kind of rose we have.

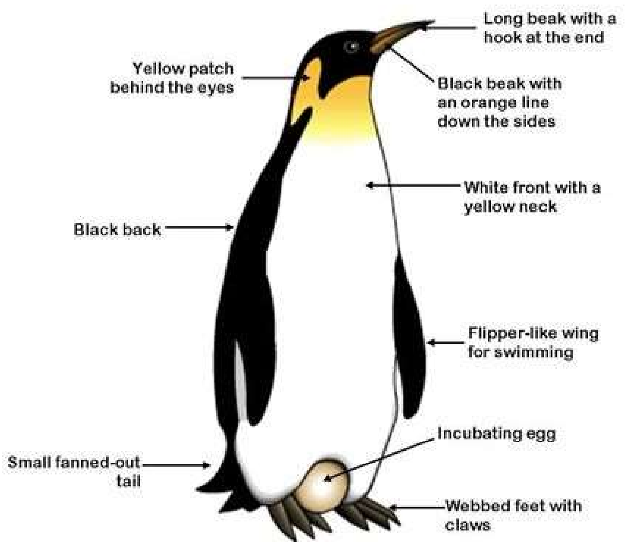
Problem Introduction.

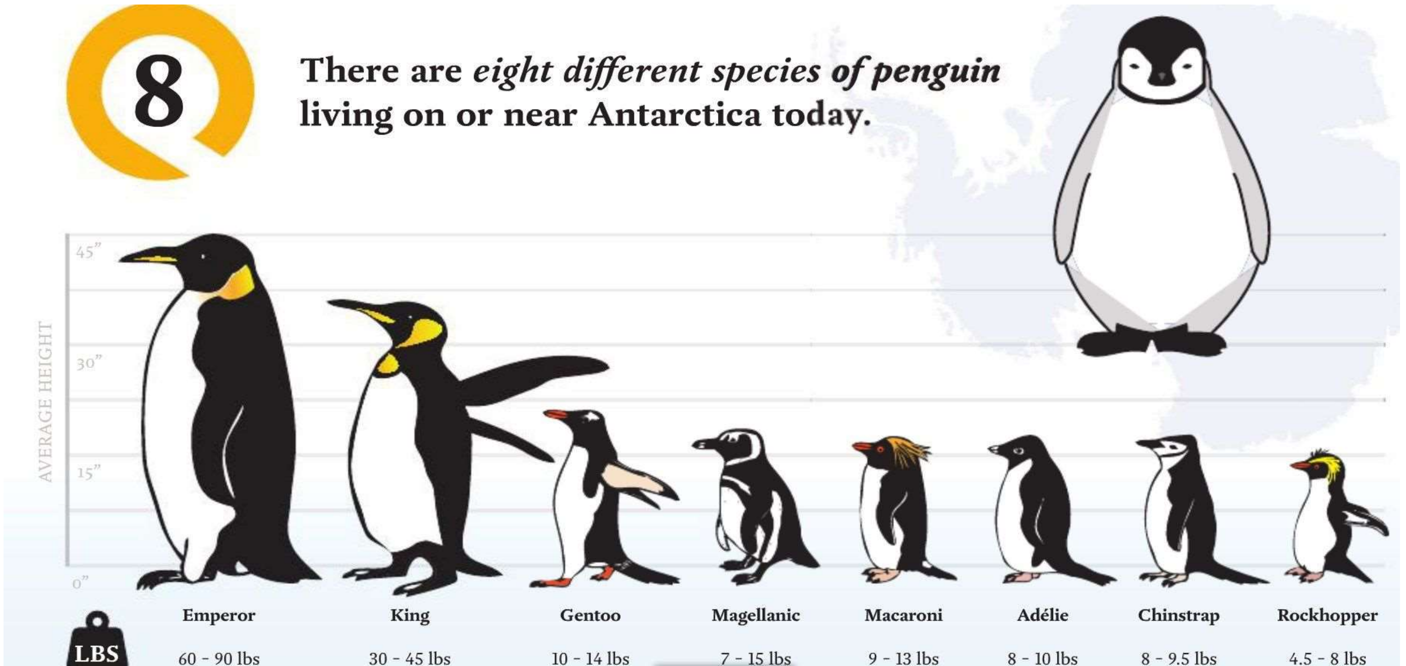
The data set is about three type of penguins. It was collected in Antarctica. Biologist caught different kind of penguins and measured their culmen length in millimeters, culmen depth in millimeters, flipper length in millimeters, body mass in grams, specified their gender and the location of the island each penguin was caught. Then they identified each penguin by its type – Chinstrap, Adelie or Gentoo.

Objective of the Model

Build a model that can predict the penguin type based on their physical characteristics.




Long beak with a hook at the end
Yellow patch behind the eyes
Black beak with an orange line down the sides
White front with a yellow neck
Black back
Flipper-like wing for swimming
Incubating egg
Small fanned-out tail
Webbed feet with claws



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df=pd.read_csv('penguins.csv')
```

Data Exploration

```
df.head()
```

The table bellow shows five rows of the data.

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	MALE
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	FEMALE

Getting some summary info about the data.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   culmen_length_mm   342 non-null    float64
 3   culmen_depth_mm    342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                334 non-null    object
dtypes: float64(4), object(3)
```

Altogether there are 344 rows, however it can be seen that this data set is missing some values for certain rows. Sex has 334 out of 344 so missing 10, and four others are missing 2.

```
df.isna().sum()
```

```
species               0
island                0
culmen_length_mm      2
culmen_depth_mm       2
flipper_length_mm     2
body_mass_g           2
sex                  10
dtype: int64
```

It is recommended working on data that have all the correct value types at the columns. Hence, we will drop the columns with missing data.

```
df = df.dropna()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 334 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   species            334 non-null    object 
 1   island             334 non-null    object 
 2   culmen_length_mm   334 non-null    float64
 3   culmen_depth_mm    334 non-null    float64
 4   flipper_length_mm  334 non-null    float64
 5   body_mass_g        334 non-null    float64
 6   sex                334 non-null    object 
dtypes: float64(4), object(3)
```

We end up dropping 10 rows.

Because not all data are numbers. It is recommended to check the composition of all non-numerical values. Sometimes it is possible to have a typo. Which will mislead the algorithm.

In how many distinct island penguins are found?

```
df['island'].unique()
```

```
array(['Torgersen', 'Biscoe', 'Dream'], dtype=object)
```

How many penguin types are in the target label?

```
df['species'].unique()
```

```
array(['Adelie', 'Chinstrap', 'Gentoo'], dtype=object)
```

What is the gender of the penguin?

```
df['sex'].unique()
```

```
array(['MALE', 'FEMALE', '.'], dtype=object)
```

The sex column besides male and female has also dot(.) in one or more rows. We will drop these rows as well.

```
df = df[df['sex']!='.']
```

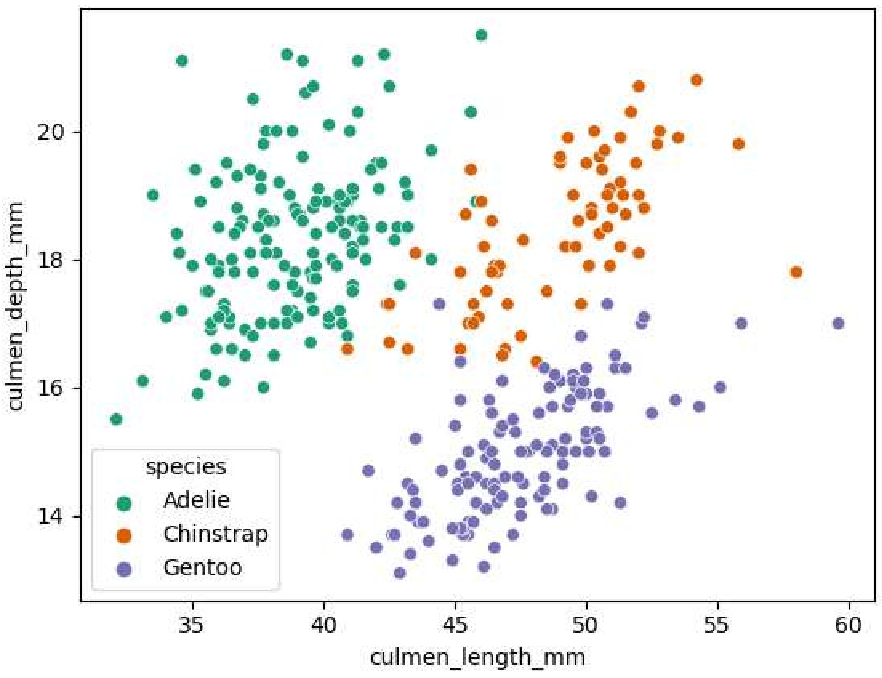
df.info()

```
<class 'pandas.core.frame.DataFrame'>
Index: 333 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   species            333 non-null    object 
 1   island             333 non-null    object 
 2   culmen_length_mm   333 non-null    float64
 3   culmen_depth_mm    333 non-null    float64
 4   flipper_length_mm  333 non-null    float64
 5   body_mass_g        333 non-null    float64
 6   sex                333 non-null    object 
dtypes: float64(4), object(3)
```

Only one row was deleted.

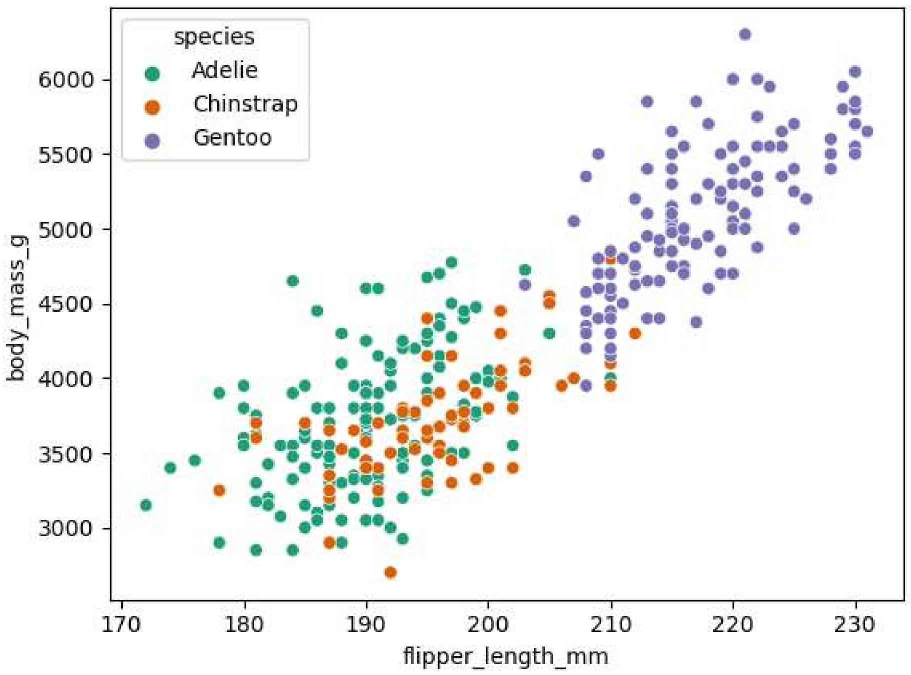
Data visualizations.

sns.scatterplot(x='culmen_length_mm',y='culmen_depth_mm',data=df,hue='species',palette='Dark2')



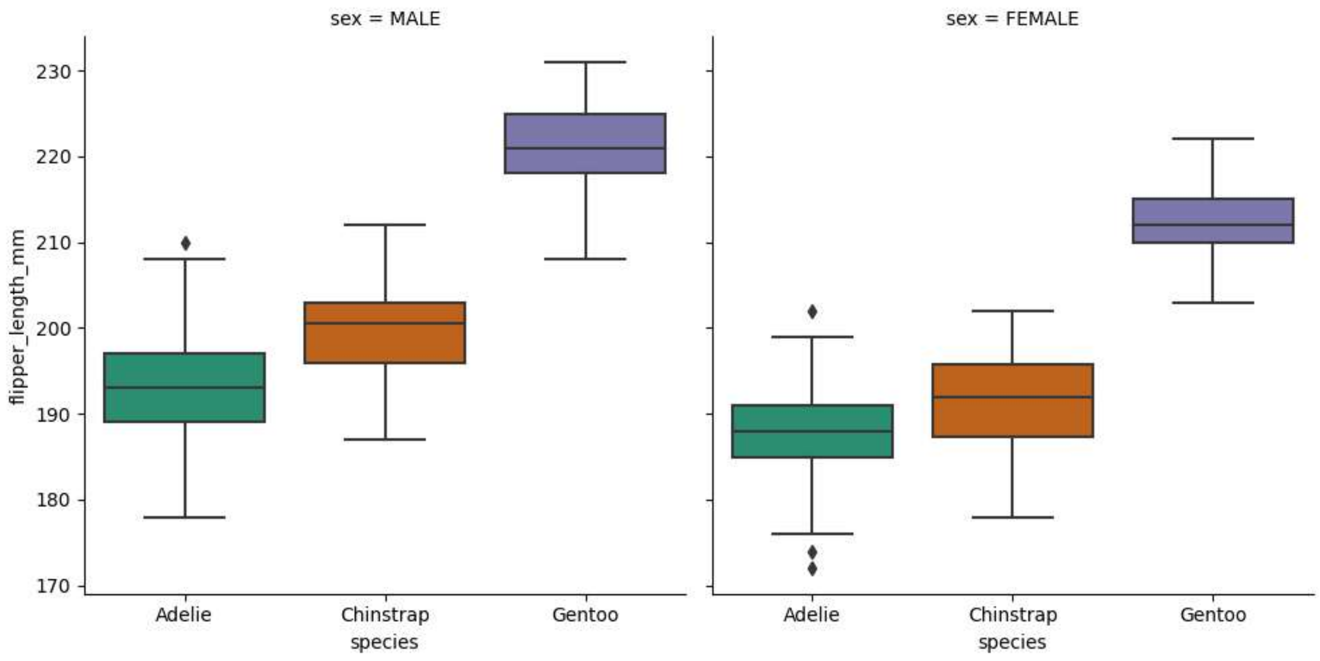
We can see a clear separation of the species if we compare their culmen depth and length.
Gentoo has much smaller culmen depth.

sns.scatterplot(x='flipper_length_mm',y='body_mass_g',data=df,hue='species',palette='Dark2')



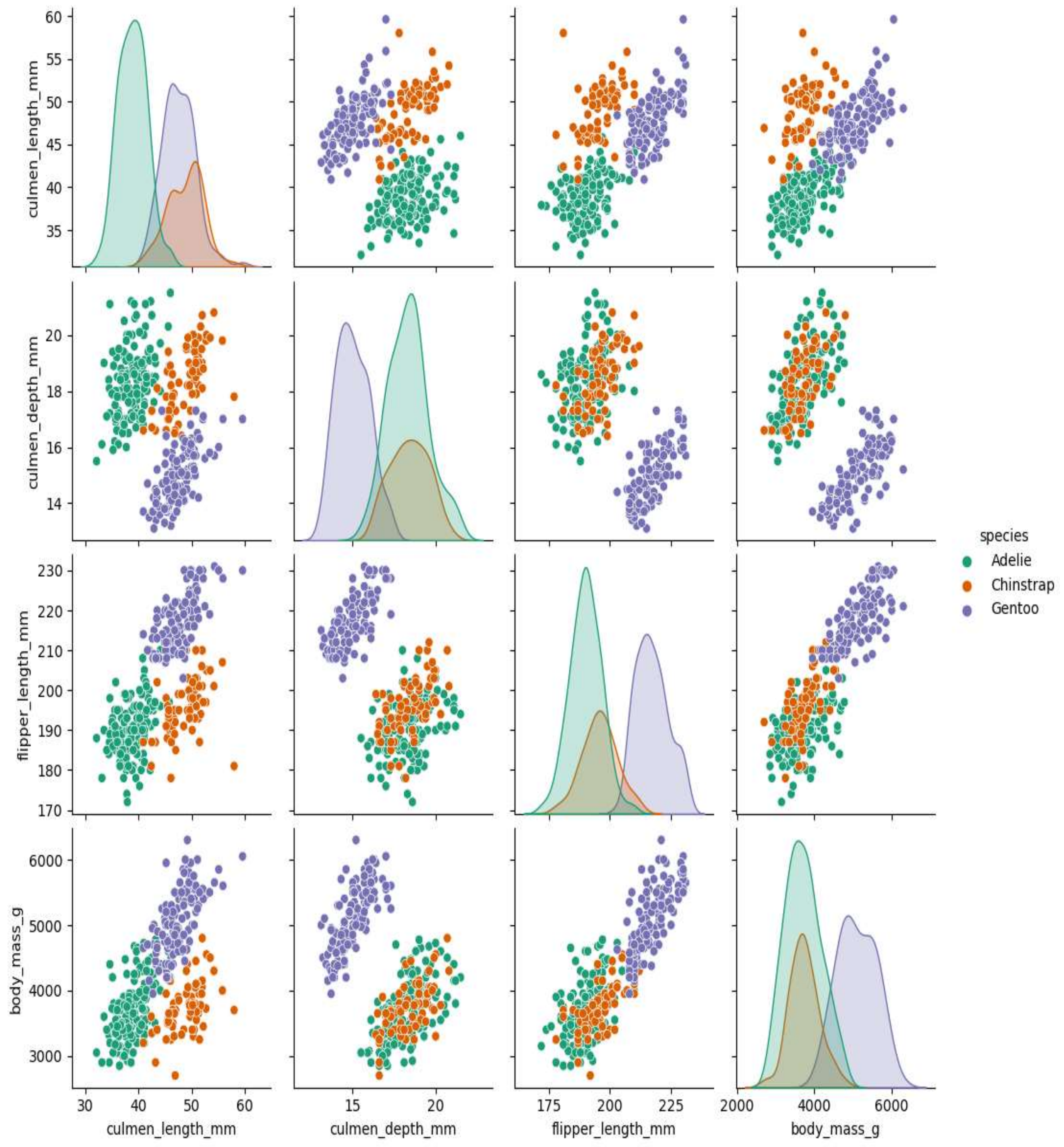
In this visualization it is very obvious that Gentoo are much bigger than the other two. They weight more and their flipper length is much longer. Addelie and Chinstrap have similar sizes.

sns.catplot(x='species',y='flipper_length_mm',data=df,kind='box',col='sex',palette='Dark2')



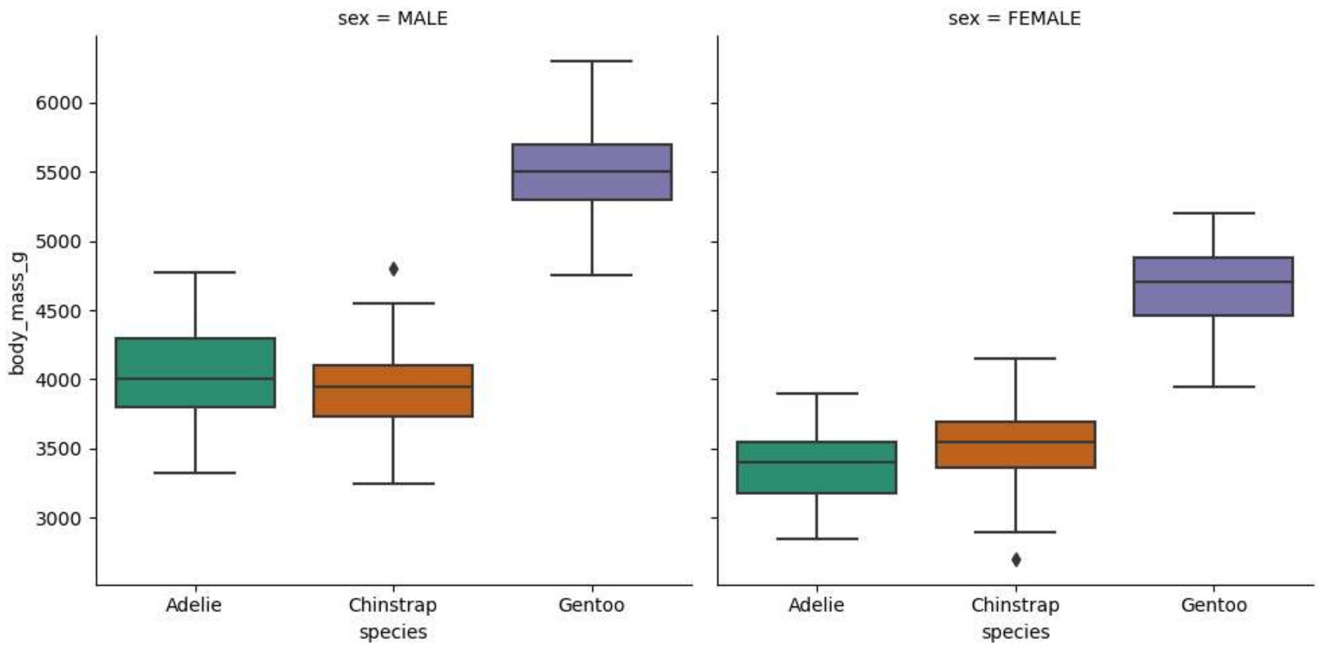
On average Chinstrap have longer flipper length than Adelie. Gentoo has the longest.

sns.pairplot(df,hue='species',palette='Dark2')

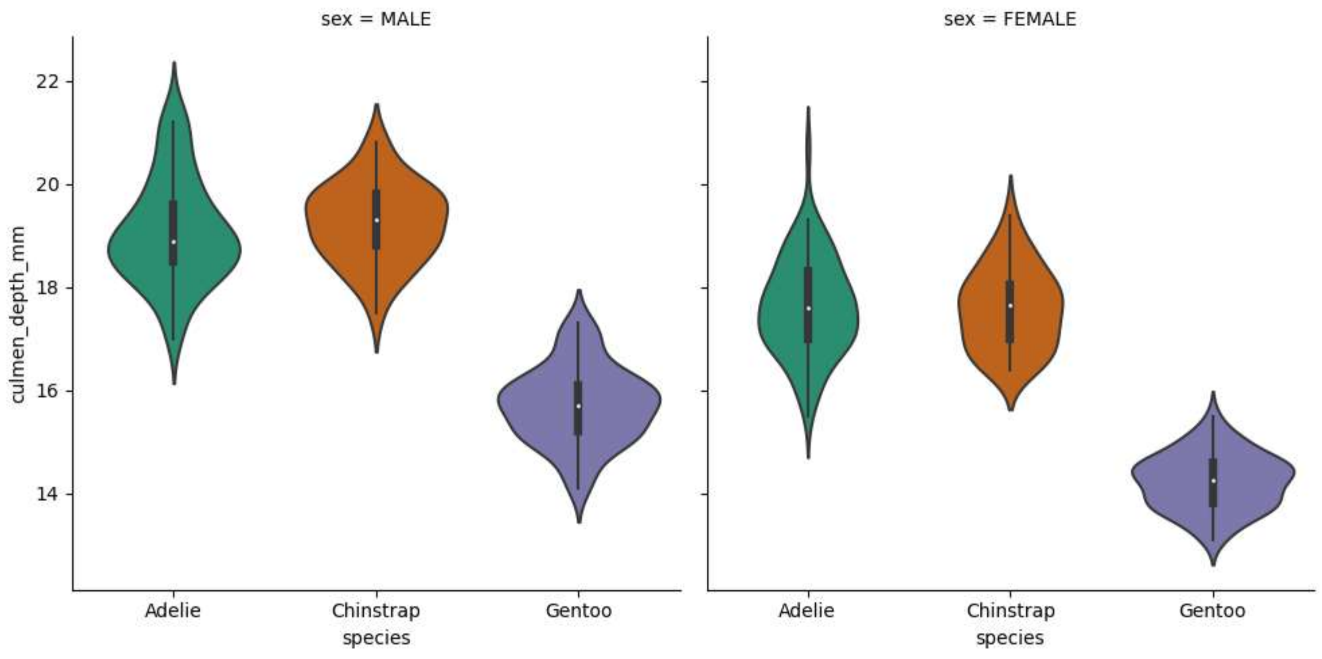


The paiplot confirms all together the conclusion we have made in the previous plots. It also show us the distribution of the features.

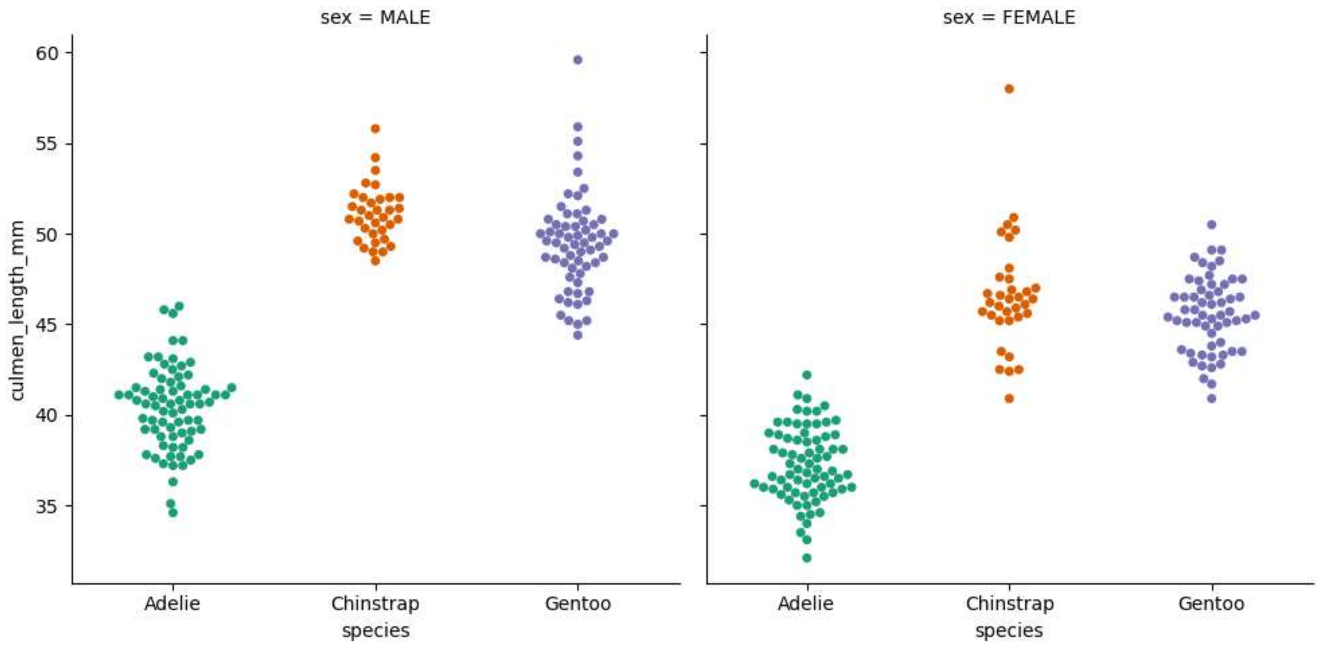
sns.catplot(x='species',y='body_mass_g',data=df,kind='box',col='sex',palette='Dark2')



sns.catplot(x='species',y='culmen_depth_mm',data=df,kind='violin',col='sex',palette='Dark2')



sns.catplot(x='species',y='culmen_length_mm',data=df,kind='swarm',col='sex',palette='Dark2')



This data set has some non-numerical data. All data should be turned into numerical value before model building begins. For that purpose we use the one-hot encoding system which converts values into zero or one. One hot encoding is a technique used in machine learning to represent categorical variables as binary vectors. It is particularly useful when dealing with categorical data that cannot be naturally ordered or compared mathematically.

pd.get_dummies(df)

Data sets with one-hot encoding becomes very large.

X = pd.get_dummies(df.drop('species',axis=1),drop_first=True)

	culmen_length_mm	**culmen_depth_mm**	**flipper_length_mm**	**body_mass_g**	**island_Dream**	**island_Torgersen**	**sex_MALE**
0	39.1	18.7	181.0	3750.0	False	True	True
1	39.5	17.4	186.0	3800.0	False	True	False
2	40.3	18.0	195.0	3250.0	False	True	False
4	36.7	19.3	193.0	3450.0	False	True	False
5	39.3	20.6	190.0	3650.0	False	True	True
...	...	...	...	...	...	...	...
338	47.2	13.7	214.0	4925.0	False	False	False
340	46.8	14.3	215.0	4850.0	False	False	False
341	50.4	15.7	222.0	5750.0	False	False	True
342	45.2	14.8	212.0	5200.0	False	False	False
343	49.9	16.1	213.0	5400.0	False	False	True

y = df['species']

```
0      Adelie
1      Adelie
2      Adelie
4      Adelie
5      Adelie
        ...
338    Gentoo
340    Gentoo
341    Gentoo
342    Gentoo
343    Gentoo
```

Like all other models the data set should be split into training and testing parts.

from sklearn.model_selection import train_test_split

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

The method that we will use is based on the decision tree method. Since the objective is to figure out three different types, it is a classification problem. Very common question is how the algorithm is classifying using logical regression methods. What happens is that, the way the problem is approached is the algorithm will pick one of the label types and classify it against the rest. Then pick the second type label and compare and try to classify against the rest this time including the first label. And so on, until all types in the target are compared against the rest.

from sklearn.tree import DecisionTreeClassifier

Create an instance of a model.

model = DecisionTreeClassifier()

model.fit(X_train,y_train)

Predict the penguin type using the left aside testing set.

base_pred = model.predict(X_test)

Evaluating Model performance.

from sklearn.metrics import confusion_matrix,classification_report

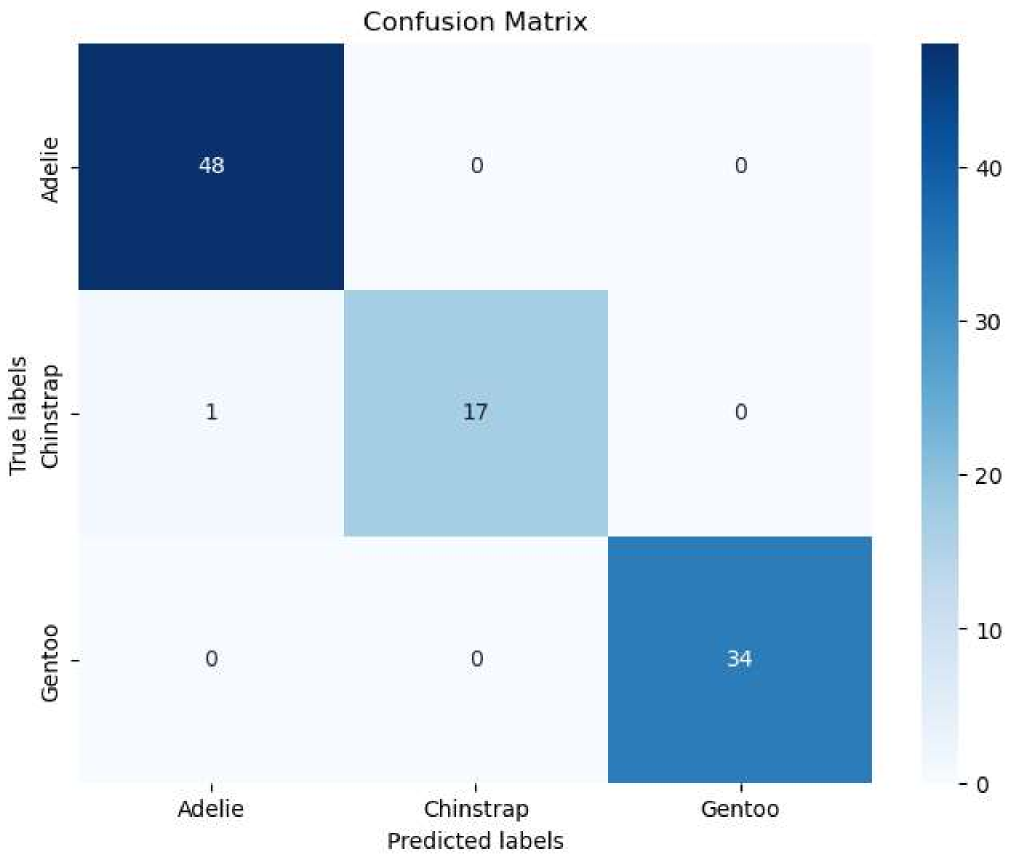
confusion_matrix(y_test,base_pred)

```
array([[48,  0,  0],
       [ 1, 17,  0],
       [ 0,  0, 34]], dtype=int64)
```

The subject of this case study is to show how to explore data and build Decision Tree Model. The logical regression case study provided some information about the meaning of the confusion matrix and accuracy metrics such as precision, recall and f1-score. More can be found `in this link.`

It is highly recommended plotting the matrix. Which helps to understand the result much easier.

```
def plot_confusion_matrix(y_test,y_pred, labels):
    cm = confusion_matrix(y_test,base_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.title('Confusion Matrix')
    plt.show()
labels=['Adelie', 'Chinstrap','Gentoo']
plot_confusion_matrix(y_test,base_pred,labels)
```



Only one Penguin was misclassified. One Chinstrap was classified as Adelie. Every other prediction was accurate.

```
print(classification_report(y_test,base_pred))
```

```
              precision    recall  f1-score   support

      Adelie       0.98      1.00      0.99        48
   Chinstrap       1.00      0.94      0.97        18
      Gentoo       1.00      1.00      1.00        34

    accuracy                           0.99       100
   macro avg       0.99      0.98      0.99       100
weighted avg       0.99      0.99      0.99       100
```

Which features are more important than others?

model.feature_importances_

```
([0.3575592 , 0.08249491, 0.49930306, 0.        , 0.03779222,
       0.        , 0.0228506 ])
```

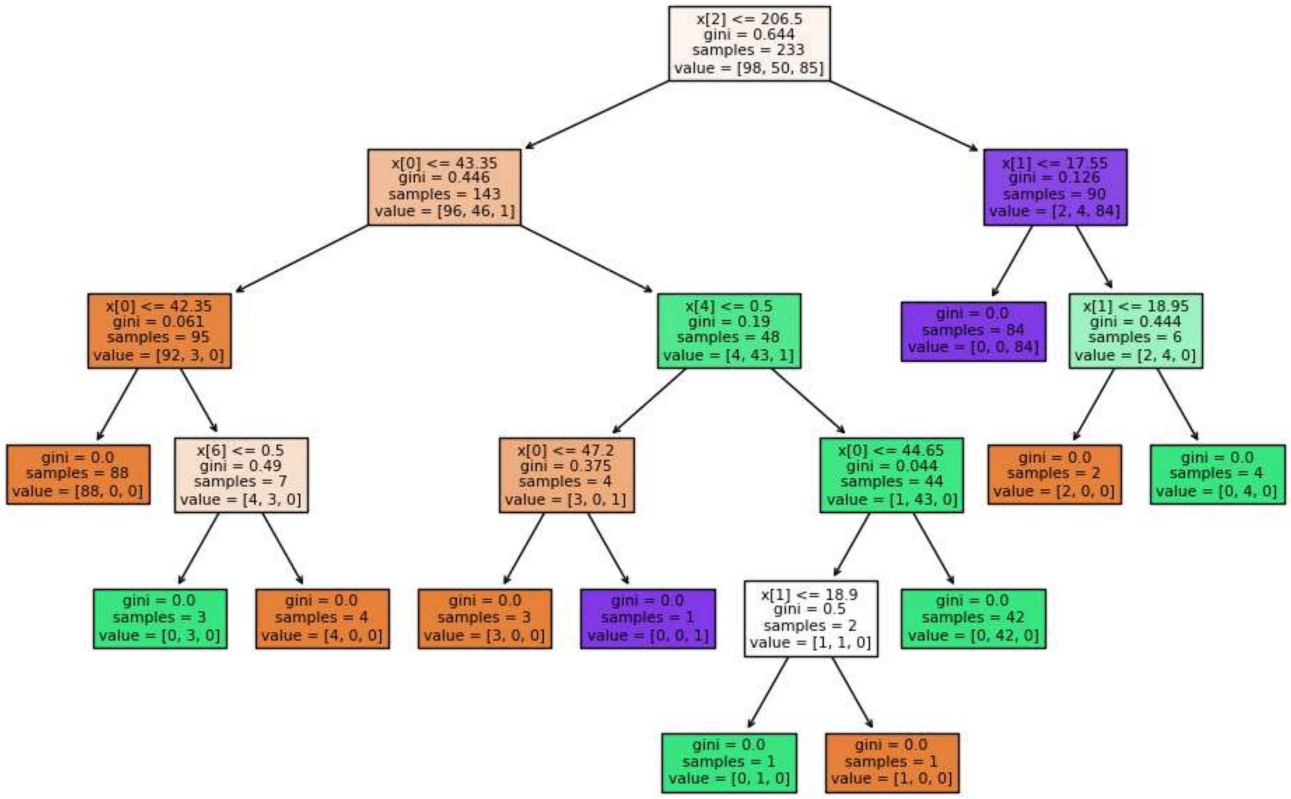
pd.DataFrame(index=X.columns,data=model.feature_importances_,columns=['Feature Importance'])

	Feature Importance
culmen_length_mm	0.357559
culmen_depth_mm	0.082495
flipper_length_mm	0.499303
body_mass_g	0.000000
island_Dream	0.037792
island_Torgersen	0.000000
sex_MALE	0.022851

Flipper length is the most important in classifying. After that comes culmen length.

Visualization of the tree.

from sklearn.tree import plot_tree
plt.figure(figsize=(12,8))
plot_tree(model,filled=True);



First, the algorithm checks x[2] which is the flipper length, then based on this measure separates the data into two. For the data with flipper length bigger than 206.5 mm it check their culmen depth. For the data with flipper length less than 206.5 mm checks x[o], their culmen length. And so on.

Even thought the model is pretty accurate, we can adjust some hyper parameters to see how the model will perform with other parameters.

For that purpose lets write a helper function.

```
def report_model(model):
    global model_peds
    model_preds = model.predict(X_test)
    print(classification_report(y_test,model_preds))
    print('\n')
    plt.figure(figsize=(12,8),dpi=150)
    plot_tree(model,filled=True,feature_names=X.columns)
    return global model_peds
```

Decision Tree Modeling using Entropy

```
entropy_tree = DecisionTreeClassifier(criterion='entropy')
entropy_tree.fit(X_train,y_train)
```

report_model(entropy_tree)

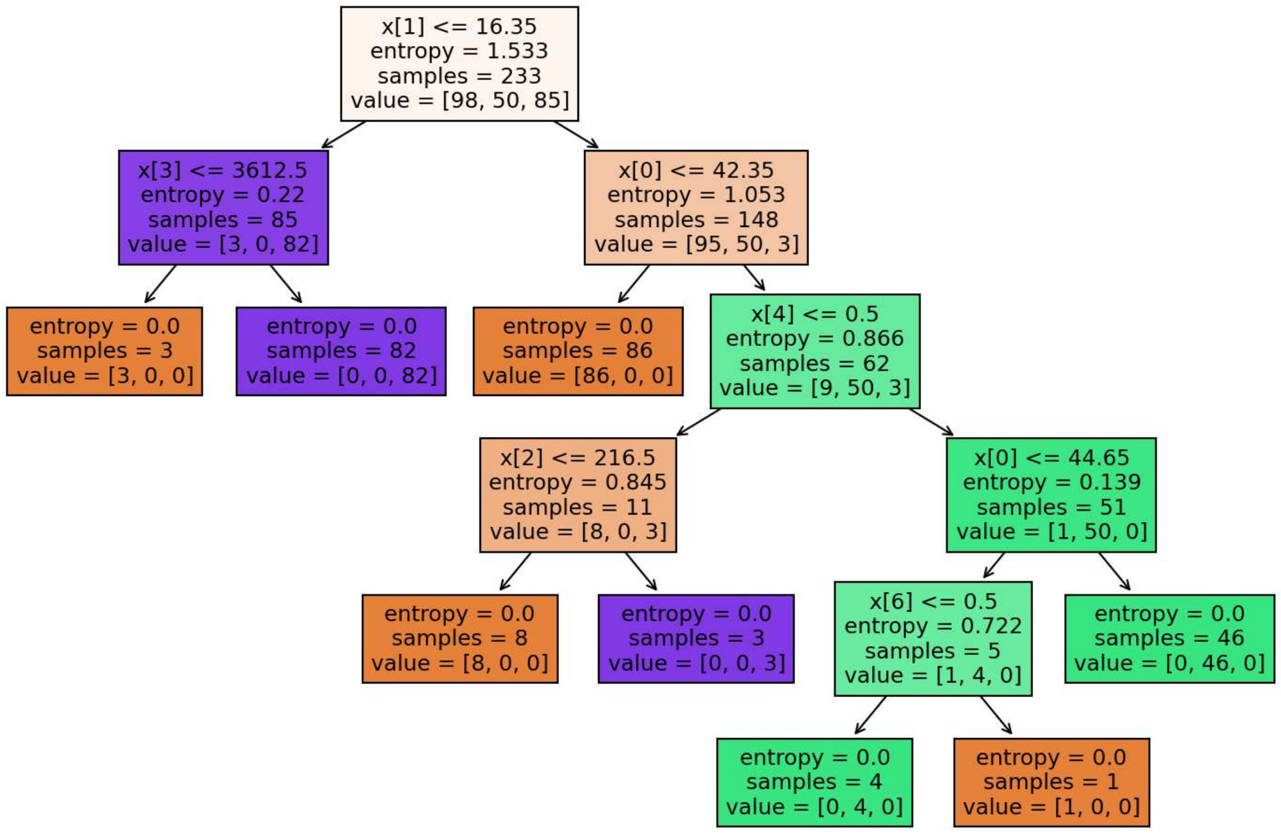
```
                precision    recall  f1-score   support

      Adelie       0.98      1.00      0.99        48
   Chinstrap       1.00      0.94      0.97        18
      Gentoo       1.00      1.00      1.00        34

    accuracy                           0.99       100
   macro avg       0.99      0.98      0.99       100
weighted avg       0.99      0.99      0.99       100
```

confusion_matrix(y_test,model_preds)

```
array([[47,  1,  0],
       [ 1, 17,  0],
       [ 0,  0, 34]], dtype=int64)
```

This model is doing slightly worse than the previous. It misclassified two penguins.

plt.figure(figsize=(12,8))
plot_tree(entropy_tree,filled=True);



entropy_tree.feature_importances_

```
array([0.30614921, 0.51098274, 0.07846864, 0.        , 0.10439941,
       0.        , 0.        ])
```

pd.DataFrame(index=X.columns,data=entropy_tree.feature_importances_,columns=['Feature Importance'])

	Feature Importance
culmen_length_mm	0.306149
culmen_depth_mm	0.510983
flipper_length_mm	0.078469
body_mass_g	0.000000
island_Dream	0.104399
island_Torgersen	0.000000
sex_MALE	0.000000

This model is checking first x[1] which is culment depth. If it is less than 16.35 mm it is checking the body mass. If it is longer than 16.35 mm it is checking x[o] which is the culmen length. And so on. Clearly the two models have different approaches. And different features have different importance in both models.

Prediction.

What type of penguin is it? It is time to check our model with data from the data set. For simplicity lets pick the first row in the data set (index=0).

X = pd.get_dummies(df.drop('species',axis=1),drop_first=True)

y = df['species']

	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	island_Dream	island_Torgersen	sex_MALE
0	39.1	18.7	181.0	3750.0	False	True	True
1	39.5	17.4	186.0	3800.0	False	True	False
2	40.3	18.0	195.0	3250.0	False	True	False
4	36.7	19.3	193.0	3450.0	False	True	False
5	39.3	20.6	190.0	3650.0	False	True	True
...	...	...	...	...	...	...	...
338	47.2	13.7	214.0	4925.0	False	False	False
340	46.8	14.3	215.0	4850.0	False	False	False
341	50.4	15.7	222.0	5750.0	False	False	True
342	45.2	14.8	212.0	5200.0	False	False	False
343	49.9	16.1	213.0	5400.0	False	False	True

```
0      Adelie
1      Adelie
2      Adelie
4      Adelie
```

```
5       Adelie
         ...
338     Gentoo
340     Gentoo
341     Gentoo
342     Gentoo
343     Gentoo
```

```
ptype1 = [[39.1, 18.7, 181, 3750, False, True, True]]

prediction1 = model.predict(ptype1)

prediction1[0]
```

'Adelie' – Correct !!

Now let’s try the last row(index 343).

ptype = [[49.9, 16.1, 213, 5400, False, False, True]]

prediction = model.predict(ptype)

prediction[0]

'Gentoo' – Correct