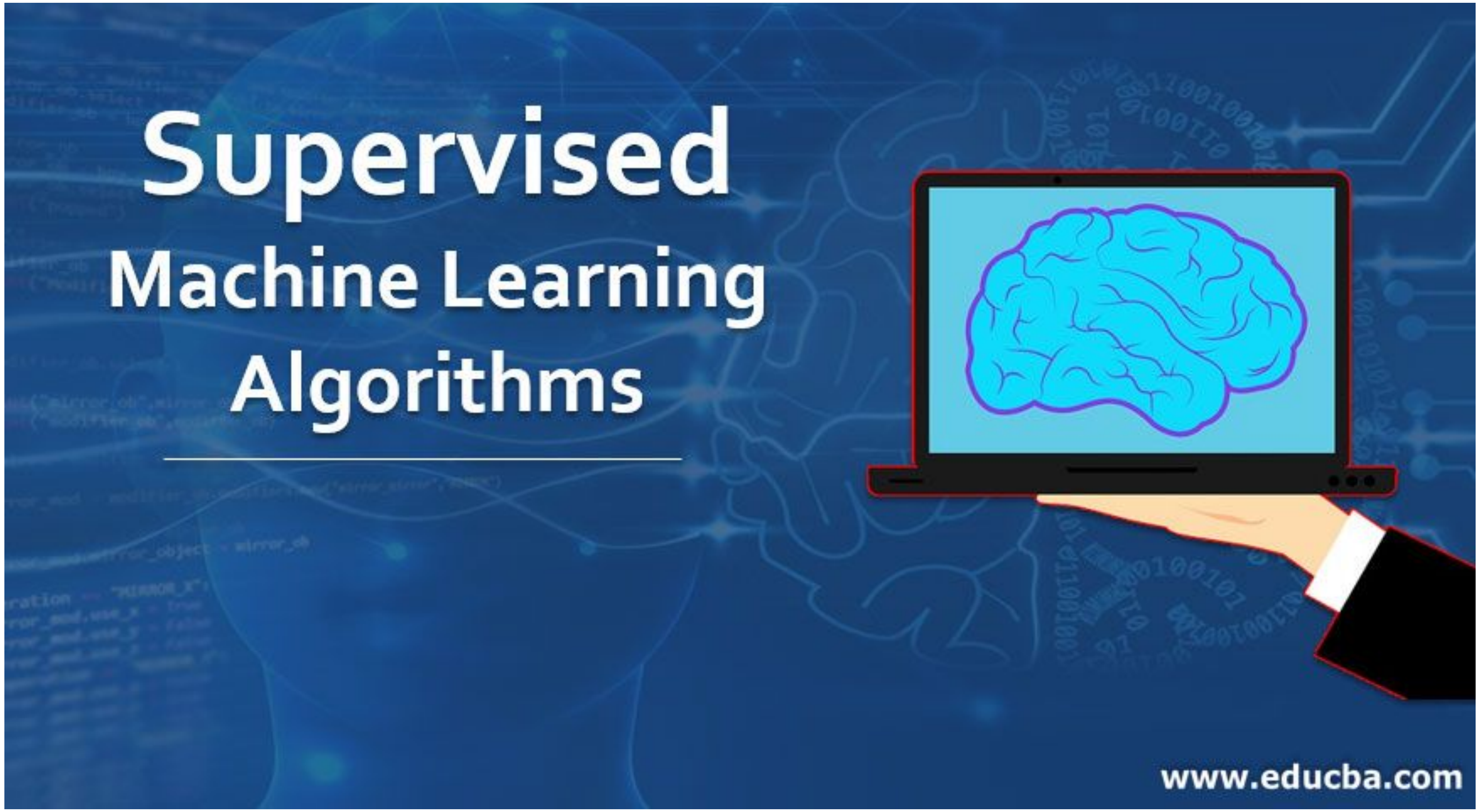


ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

Last lecture reminder

We learned about:

- Decision Trees - Theory
- Decision Trees - Terminology
- Decision Trees - Gini Impurity introduction
- Decision Trees - Gini Impurity calculation formula
- Decision Trees - Gini Impurity simple example

Decision Tree - Gini Impurity Usage

In the previous lecture we learned what is Gini Impurity and how to calculate it for each node in the tree. Once we understand that we can use the Gini Impurity value to decide where is the best places to cut the tree.

Our goal is minimizing the Gini impurity at the leaf nodes. The leaf nodes in a decision tree are the final categories / classes that a data point belongs to after passing through all decisions (nodes) in the tree.

The Gini impurity is a metric that quantifies the "purity" of a group of instances at a node. A low Gini impurity indicates a "purified" node, meaning most of the instances at that node belong to the same class. The lower the Gini impurity, the purer the node.

Decision Tree - Gini Impurity Usage

Why do we want to do this?

In classification tasks, our main goal is separating observations from classes as effectively as possible. "Effectively" means correctly assigning as many observations to their respective classes. If we're able to achieve a lower Gini Impurity at the leaf nodes, it implies that we've managed to create decisions that group instances correctly within their classes, hence, separating the classes effectively.

When instances in a class at a leaf node are more homogenous (pure), it becomes easier to make accurate predictions about future instances and the overall performance of our decision tree model improves.

In summary, by minimizing the Gini impurity at each decision stage in our tree, we're optimizing the decision tree to make the most accurate classifications possible, particularly at the leaf nodes (final decisions).

Gini Impurity With Multiple Features

Until now we saw how to calculate Gini Impurity and construct a decision tree for binary categorical label and only one binary categorical feature.

This was a very basic example because we didn’t have a choice for the root node in the tree (only 1 feature) and we didn’t have to decide how to effectively cut the tree (only 2 categories in the feature).

Let’s now see an example of a dataset with continuous data feature:

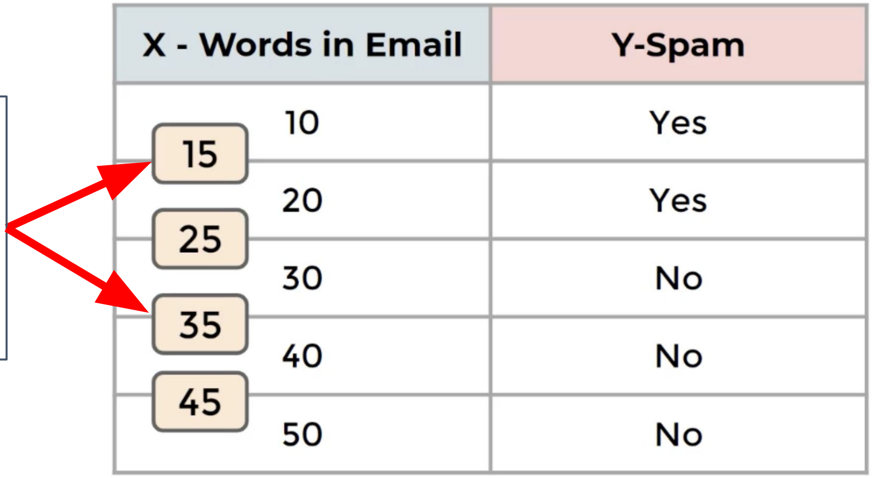
For this example we will use a given dataset providing data about emails with amount of words and if that email was a spam or not. Same as in the previous example, we want to predict according the given data if new emails will be spam or not.

X - Words in Email	Y-Spam
10	Yes
20	Yes
30	No
40	No
50	No

Gini Impurity With Multiple Features

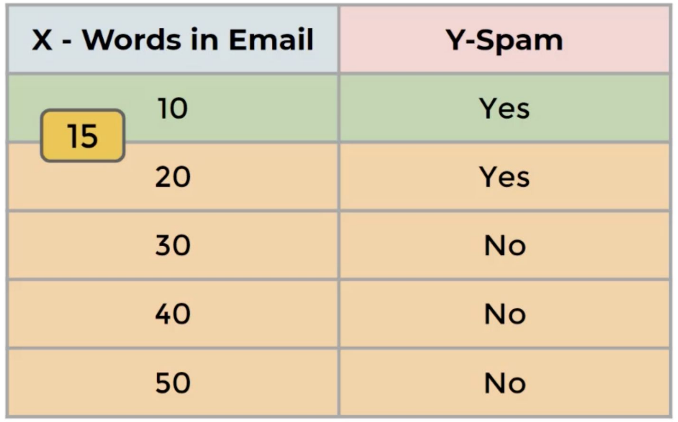
What we can do is calculate the average amount of words between each two categories and split the data accordingly.

Because we can get emails to predict with number of words between those categories, we want to take the average amount between each 2 categories and split accordingly

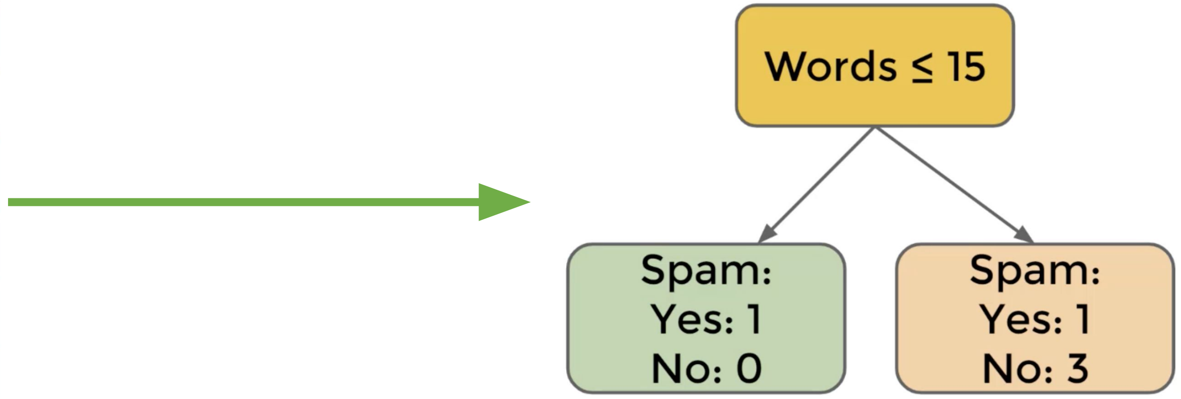


X - Words in Email	Y-Spam
10	Yes
20	Yes
30	No
40	No
50	No

So if we will choose to split according to Words <= 15 we will get:

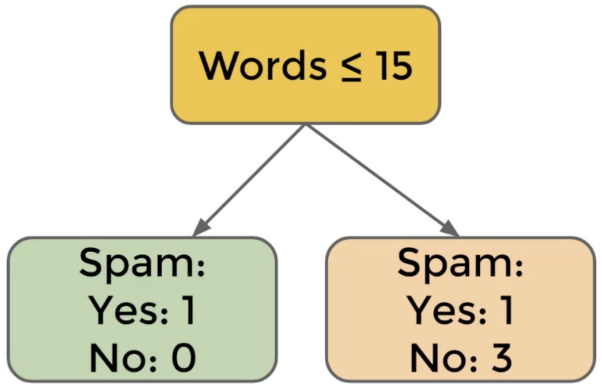


X - Words in Email	Y-Spam
10	Yes
20	Yes
30	No
40	No
50	No



Gini Impurity With Multiple Features

We can also calculate the Gini Impurity according to the split we chose:



$$G(Q) = (1/5)(0+0) + (4/5)((1/4)(1-1/4)+(3/4)(1-3/4)$$

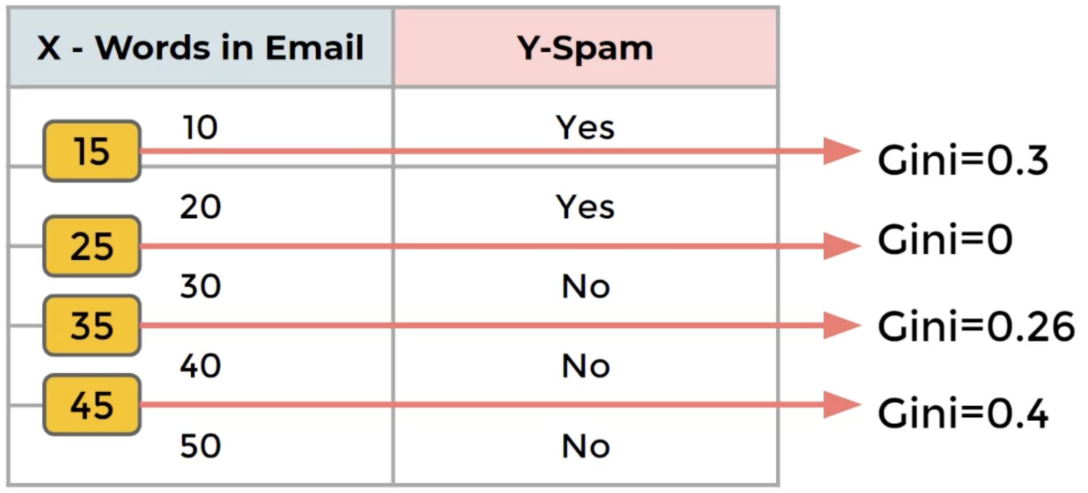
Now we need to ask ourselves if splitting the dataset according to Words <= 15 is the optimal choice or should we split our data on a different category (for example, Words <= 25, Words <= 35 …)

In order to determine what is the optimal category to split our data on we need to calculate for each category the Gini Impurity and choosing the one that minimize the Gini Impurity value.

Gini Impurity With Multiple Features

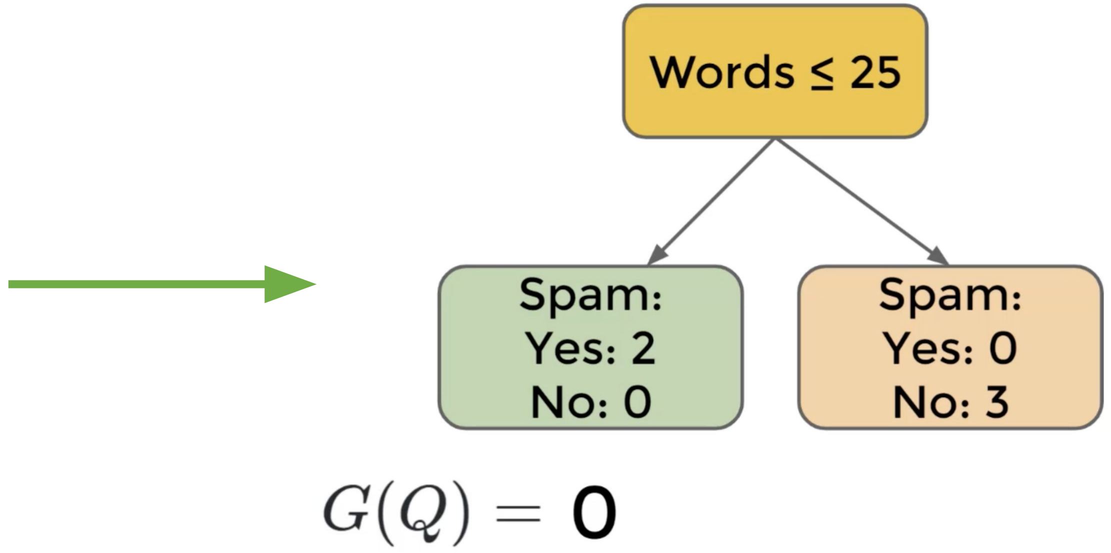
Let's calculate for each category its Gini Impurity value:

The calculation is done the same as we did with Words <= 15, we split our data as we have only this category and ignoring the other categories.



X - Words in Email	Y-Spam
10	Yes
20	Yes
30	No
40	No
50	No

15
25
35
45
Gini=0.3
Gini=0
Gini=0.26
Gini=0.4



$$G(Q) = 0$$

We can see that splitting the data on Words <= 25 provide the minimum amount of Gini Impurity so this will be the optimal category to split our data on.

Class Exercise - Gini Impurity Multiple Features

Instructions:

You have a simple dataset of animals in a zoo. They are categorized based on two features: whether they are mammals or not (Mammal or No Mammal), and whether they have fewer than four legs (L<4 or L>=4).

Your task is to classify if the animal is a pet (Pet) or not a pet (No Pet).

- Decide according to Gini Impurity how to split the data between the different features
- Construct a decision tree based on your feature split choice
- Calculate for each leaf node the Gini Impurity value
- Calculate the entire tree Gini Impurity value
- Run a simulation with the following unknown data and predict according to your decision tree if the animal is Pet or No Pet.

  Unknown animal → Mammal = No Mammal, Less than 4 legs = Yes

Class Exercise - Gini Impurity Multiple Features

Provided Data Set:

Animal	Is Mammal	Less Than 4 Legs (L<4)	Is Pet
1	Mammal	Yes	Pet
2	Mammal	Yes	Pet
3	No Mammal	Yes	No Pet
4	Mammal	No	No Pet
5	No Mammal	No	No Pet
6	No Mammal	Yes	Pet
7	Mammal	Yes	Pet
8	Mammal	Yes	No Pet
9	Mammal	No	Pet
10	No Mammal	No	No Pet

Class Exercise Solution -
Gini Impurity Multiple Features

Decision Tree - Python Example

Let’s now run an actual decision tree model with Python and Scikit-Learn library.

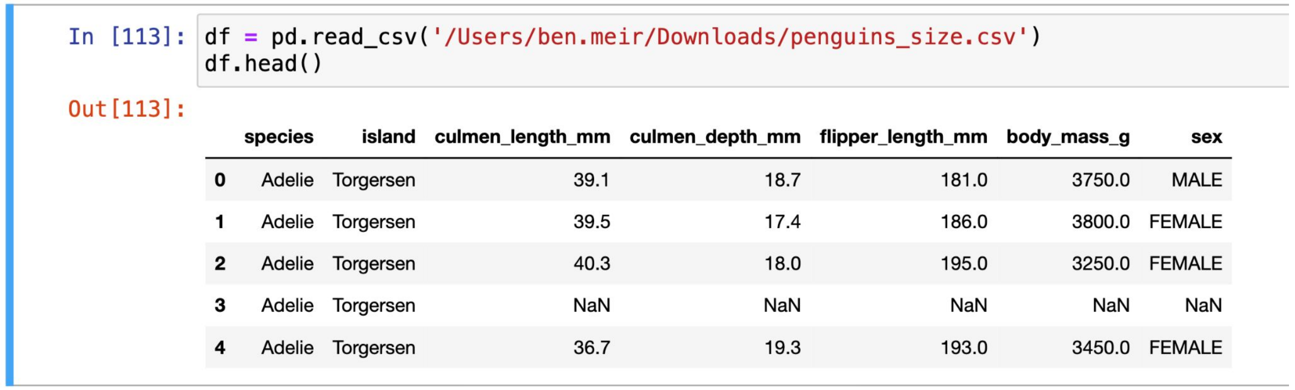
For this example we will use the **‘penguins_size.csv’** file.

The file contain data about different penguins species like body mass, sex, island, flipper length etc…

Our job is to use a decision tree model in order to predict the penguin species

(‘Adelie’ / ‘Chinstrap’ / ‘Gentoo’)

At first we will generate a default decision tree and then we will explore the different model parameters and how they affect the model performance and accuracy.



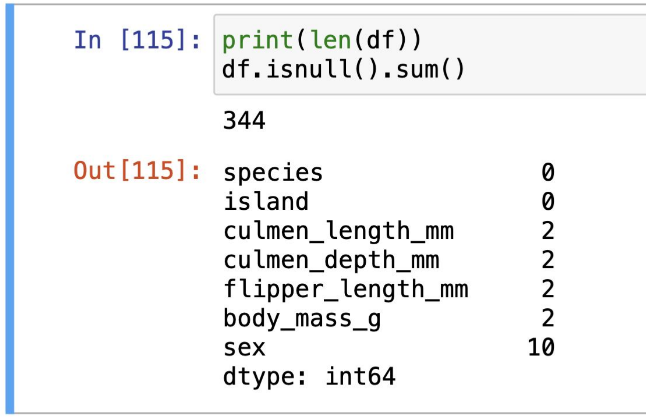
```
In [113]: df = pd.read_csv('/Users/ben.meir/Downloads/penguins_size.csv')
          df.head()
Out[113]:
```

	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

Decision Tree - Python Example

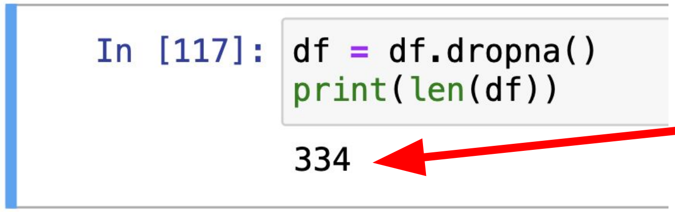
Let's first perform some data preparation because we have columns with missing data and with unrecognized data as well.

First, let's find out what are the columns with missing data:



```
In [115]: print(len(df))
          df.isnull().sum()

          344

Out[115]: species               0
          island                0
          culmen_length_mm      2
          culmen_depth_mm       2
          flipper_length_mm     2
          body_mass_g           2
          sex                  10
          dtype: int64
```

We can see that out of 344 rows only couple of rows has missing data so we can consider just drop them all out and not try to fill the missing values.

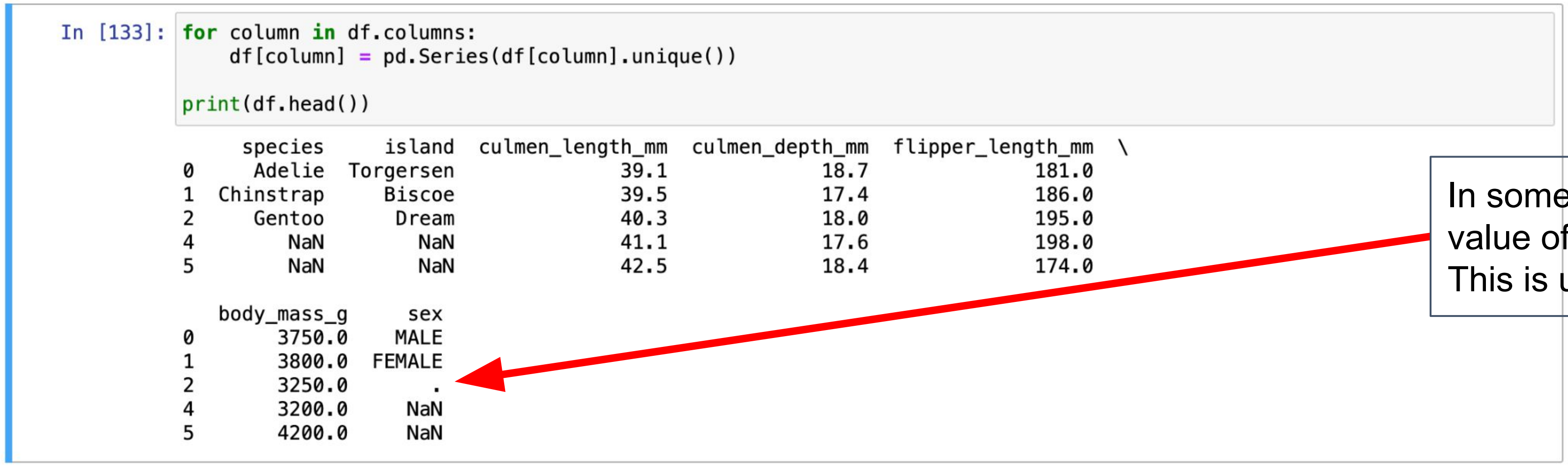


```
In [117]: df = df.dropna()
          print(len(df))

          334
```

We removed all missing data and our df size reduced by 10 rows from 344 to 334

Decision Tree - Python Example

As mentioned, we also need to handle some unrecognized data values. The way we can handle it is first print a dataframe with all unique values from all columns.

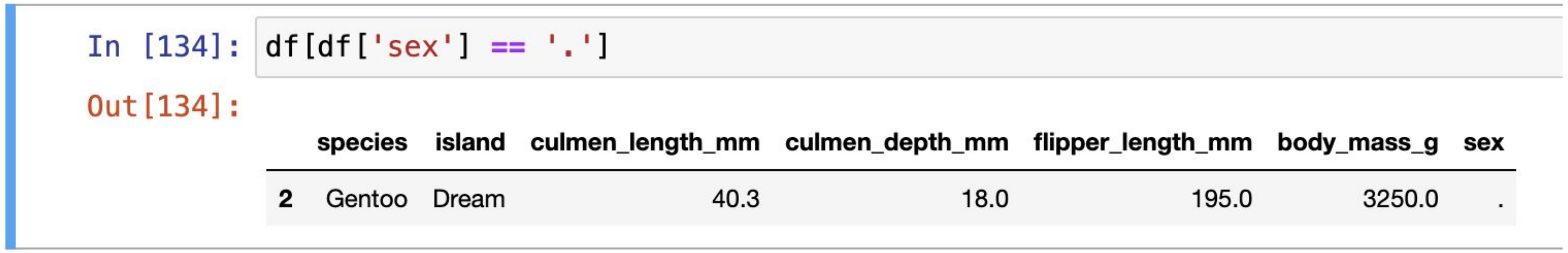


```
In [133]: for column in df.columns:
              df[column] = pd.Series(df[column].unique())

          print(df.head())
```

```
       species     island  culmen_length_mm  culmen_depth_mm  flipper_length_mm  \
0       Adelie  Torgersen              39.1             18.7              181.0
1    Chinstrap     Biscoe              39.5             17.4              186.0
2       Gentoo      Dream              40.3             18.0              195.0
4          NaN        NaN              41.1             17.6              198.0
5          NaN        NaN              42.5             18.4              174.0

   body_mass_g     sex
0       3750.0    MALE
1       3800.0  FEMALE
2       3250.0       .
4       3200.0     NaN
5       4200.0     NaN
```

In some rows we have '.' as the 'sex' value of the penguin.
This is unrecognized value.

Now, let's filter all the rows that contain '.' as the 'sex' column value:



```
In [134]: df[df['sex'] == '.']
```

Out[134]:

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
2	Gentoo	Dream	40.3	18.0	195.0	3250.0	.

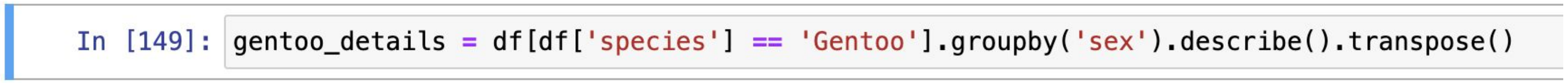
Decision Tree - Python Example

How should we handle the row with the unrecognized value?

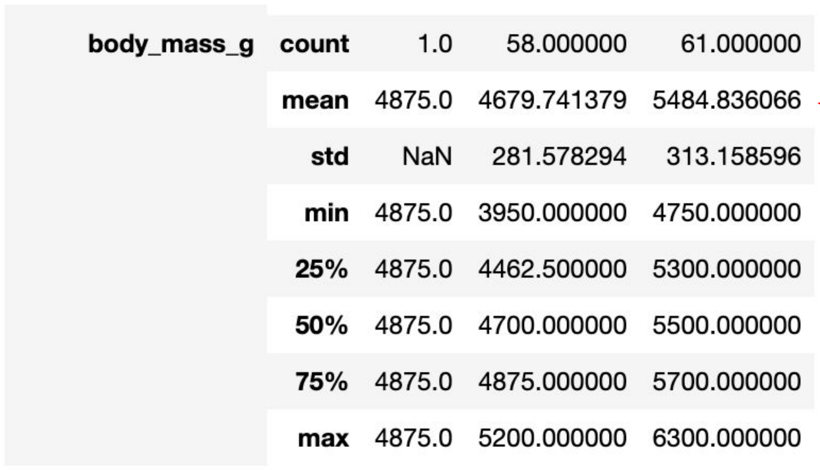
We have 2 options: or to remove it completely or to try and fill the value manually.

Let’s go with the second approach and try to figure out based on the other row data what should most likely be the ‘sex’ value of the penguin.

In order to do it we first need to compare the row data with the other dataframe data:



```
In [149]: gentoo_details = df[df['species'] == 'Gentoo'].groupby('sex').describe().transpose()
```

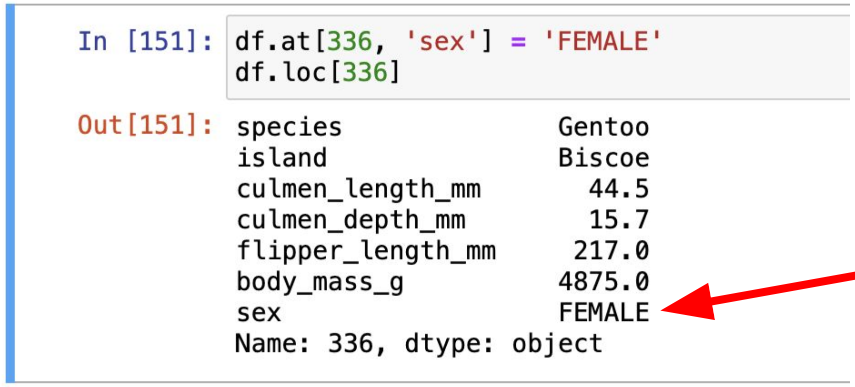


```
body_mass_g  count     1.0    58.000000    61.000000
              mean  4875.0  4679.741379  5484.836066
               std     NaN   281.578294   313.158596
               min  4875.0  3950.000000  4750.000000
               25%  4875.0  4462.500000  5300.000000
               50%  4875.0  4700.000000  5500.000000
               75%  4875.0  4875.000000  5700.000000
               max  4875.0  5200.000000  6300.000000
```

We can see that the penguin body mass value is 4875 and it can fit both to ‘male’ and ‘female’ but according to the comparison it's more likely a body mass of a female penguin.

Decision Tree - Python Example

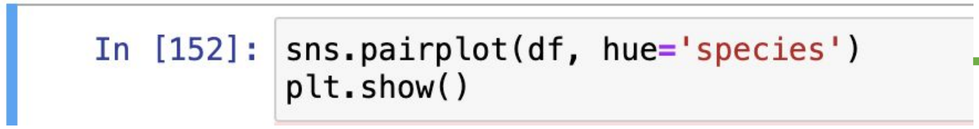
Now, let’s change manually the value for the ‘sex’ column to ‘female’:



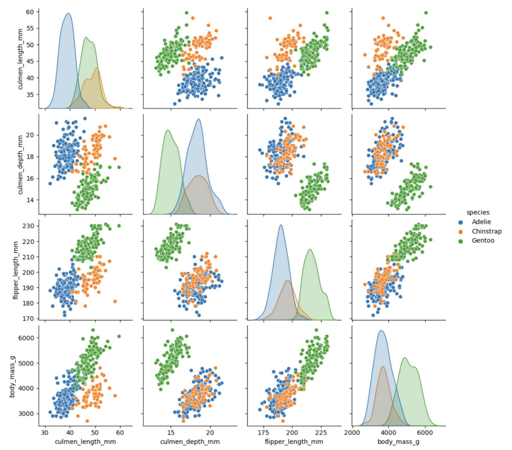
```
In [151]: df.at[336, 'sex'] = 'FEMALE'
          df.loc[336]

Out[151]: species                Gentoo
          island                 Biscoe
          culmen_length_mm         44.5
          culmen_depth_mm          15.7
          flipper_length_mm       217.0
          body_mass_g            4875.0
          sex                    FEMALE
          Name: 336, dtype: object
```

The value of the column is now ‘FEMALE’

Once we finish with the data preparation we can also generate a scatterplot to explore the different data features and the correlation between those feature and the penguin species (label).



```
In [152]: sns.pairplot(df, hue='species')
          plt.show()
```



ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

Decision Tree - Python Example

When dealing with decision trees in Scikit-Learn there are couple of things we need to understand:

- **No need for feature scaling** → Because in decision tree we only compare data according to a specific feature each time and not combining features like in other models, we don't need to make sure all feature units are in the same scale so feature scaling is redundant.

- **String values need to be changed to boolean / numbers** → Scikit-Learn tree algorithms doesn't know how to handle with string values so we need to transform those values in our dataframe into boolean values or number values instead.
  For example → A 'sex' column with 'Male' and 'Female' values can transform to a new column called 'is_male' with value True (or 1) for 'sex' = 'Male' and value False (or 0) for 'sex' = 'Female'.

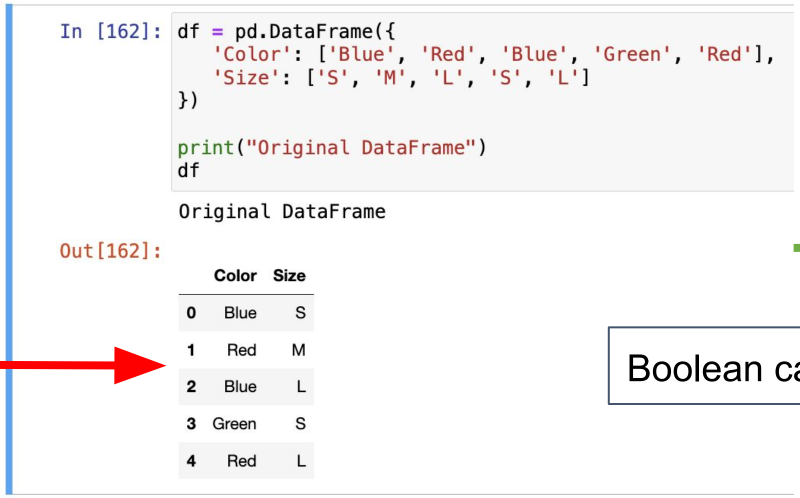
Decision Tree - Python Example

In our dataset we need to change the ‘sex’ column and the ‘island’ column which both of them are string values. In order to do it easily we can use the Pandas **get_dummies()** method.

pd.get_dummies() → A method used to convert categorical variables into dummy or indicator variables (also known as one-hot encoding). Each category becomes a new column in the data frame and is marked by 1 or 0 (True/False) whether that category is applicable for that row.

For example →

String categorical data

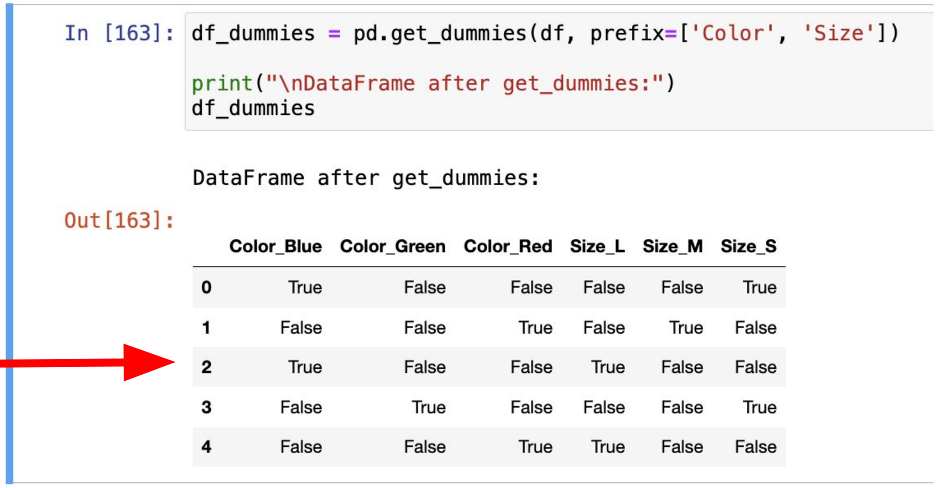


```
In [162]: df = pd.DataFrame({
              'Color': ['Blue', 'Red', 'Blue', 'Green', 'Red'],
              'Size': ['S', 'M', 'L', 'S', 'L']
          })

          print("Original DataFrame")
          df

          Original DataFrame
```

Out[162]:

	Color	Size
0	Blue	S
1	Red	M
2	Blue	L
3	Green	S
4	Red	L

Boolean categorical data



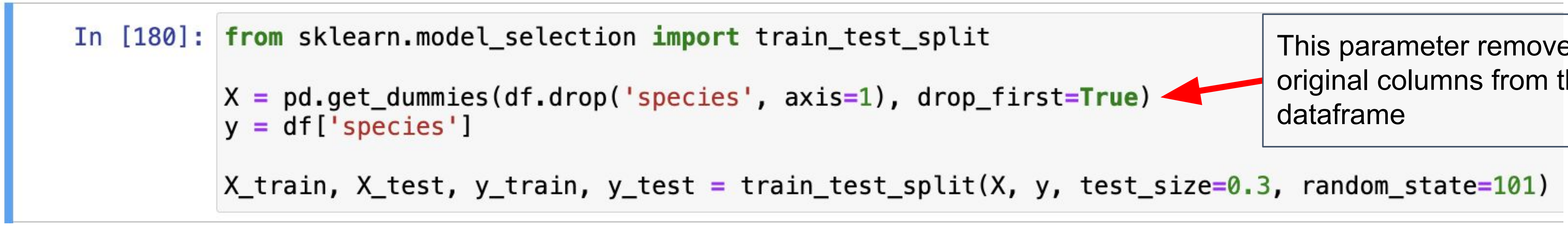
```
In [163]: df_dummies = pd.get_dummies(df, prefix=['Color', 'Size'])

          print("\nDataFrame after get_dummies:")
          df_dummies

          DataFrame after get_dummies:
```

Out[163]:

	Color_Blue	Color_Green	Color_Red	Size_L	Size_M	Size_S
0	True	False	False	False	False	True
1	False	False	True	False	True	False
2	True	False	False	True	False	False
3	False	True	False	False	False	True
4	False	False	True	True	False	False

Decision Tree - Python Example

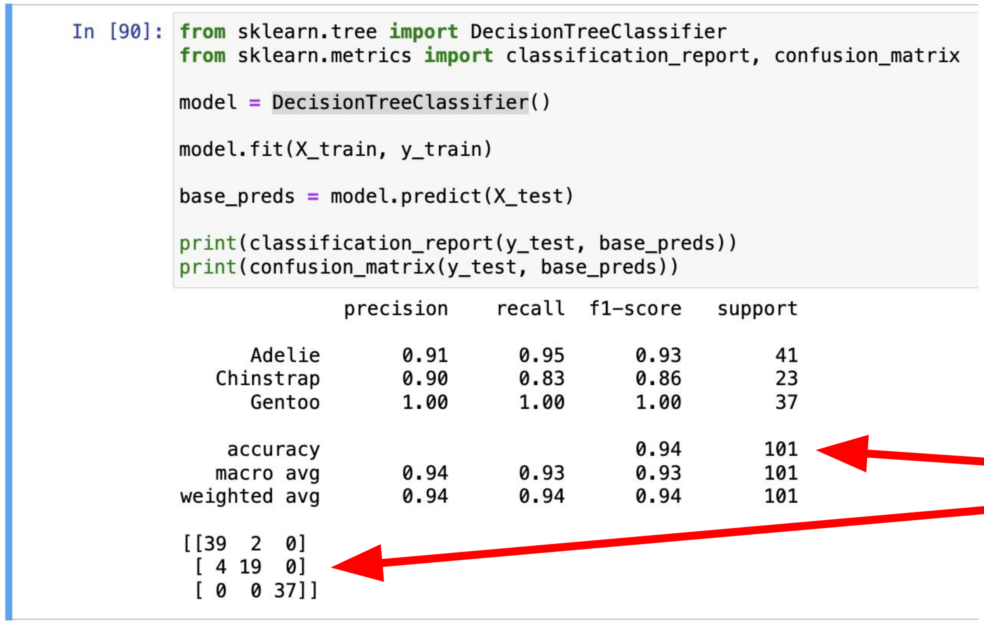
Let’s use the pd.get_dummies() method and build the model features and label datasets:



```
In [180]: from sklearn.model_selection import train_test_split

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

          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

This parameter remove the original columns from the result dataframe

For the model training we will use the **DecisionTreeClassifier()** model with the default parameters.



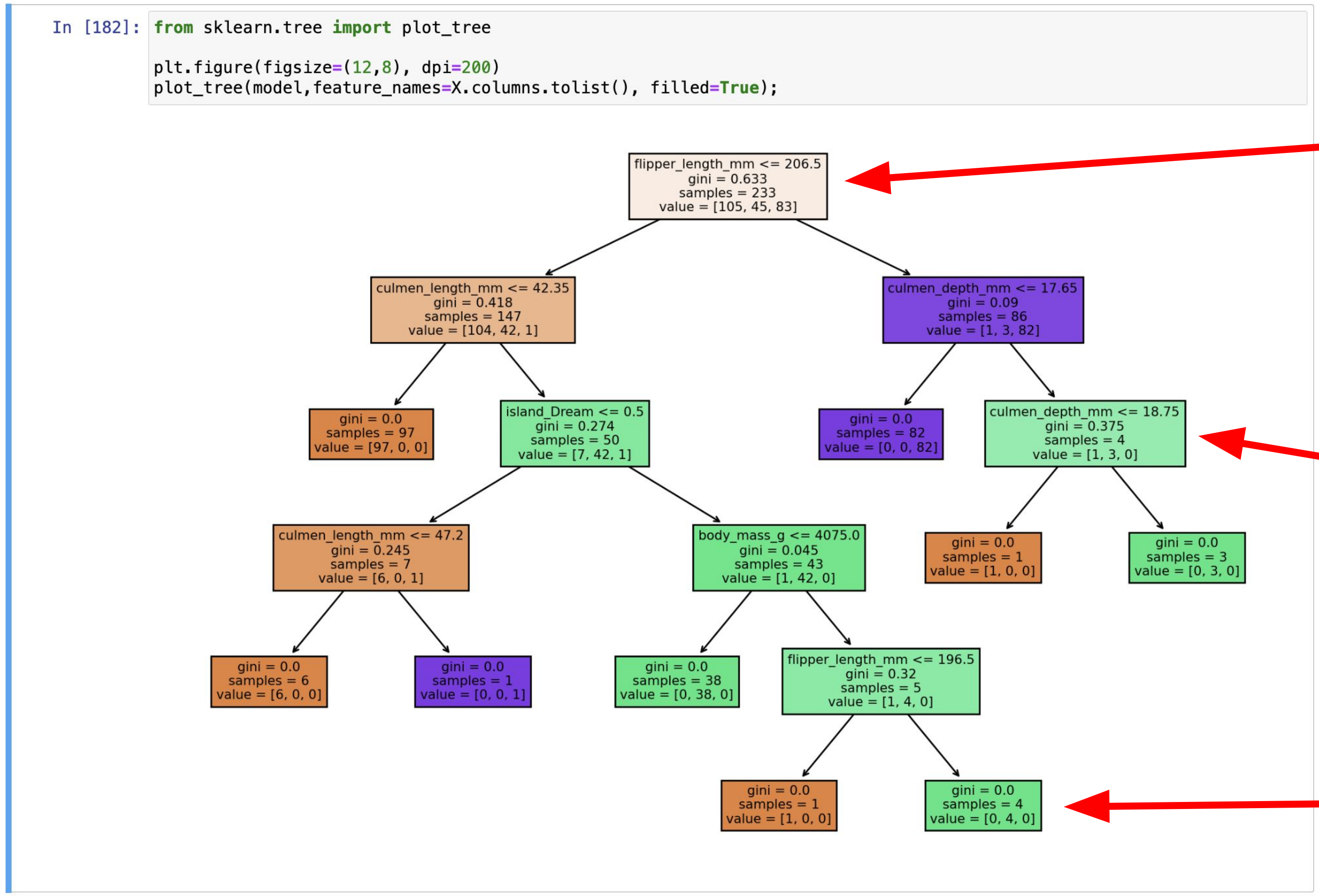
```
In [90]: from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import classification_report, confusion_matrix

         model = DecisionTreeClassifier()

         model.fit(X_train, y_train)

         base_preds = model.predict(X_test)

         print(classification_report(y_test, base_preds))
         print(confusion_matrix(y_test, base_preds))
```

```
              precision    recall  f1-score   support

      Adelie       0.91      0.95      0.93        41
   Chinstrap       0.90      0.83      0.86        23
      Gentoo       1.00      1.00      1.00        37

    accuracy                           0.94       101
   macro avg       0.94      0.93      0.93       101
weighted avg       0.94      0.94      0.94       101

[[39  2  0]
 [ 4 19  0]
 [ 0  0 37]]
```

Our model misclassified 6 pingunses out of 101
Representing a model accuracy of 94%

Decision Tree - Python Example

In order to visualize the tree our model constructed we can use the ‘**plot_tree**’ functionality.



```
In [182]: from sklearn.tree import plot_tree

          plt.figure(figsize=(12,8), dpi=200)
          plot_tree(model,feature_names=X.columns.tolist(), filled=True);
```

flipper_length_mm <= 206.5 / gini = 0.633 / samples = 233 / value = [105, 45, 83]
culmen_length_mm <= 42.35 / gini = 0.418 / samples = 147 / value = [104, 42, 1]
culmen_depth_mm <= 17.65 / gini = 0.09 / samples = 86 / value = [1, 3, 82]
gini = 0.0 / samples = 97 / value = [97, 0, 0]
island_Dream <= 0.5 / gini = 0.274 / samples = 50 / value = [7, 42, 1]
gini = 0.0 / samples = 82 / value = [0, 0, 82]
culmen_depth_mm <= 18.75 / gini = 0.375 / samples = 4 / value = [1, 3, 0]
culmen_length_mm <= 47.2 / gini = 0.245 / samples = 7 / value = [6, 0, 1]
body_mass_g <= 4075.0 / gini = 0.045 / samples = 43 / value = [1, 42, 0]
gini = 0.0 / samples = 1 / value = [1, 0, 0]
gini = 0.0 / samples = 3 / value = [0, 3, 0]
gini = 0.0 / samples = 6 / value = [6, 0, 0]
gini = 0.0 / samples = 1 / value = [0, 0, 1]
gini = 0.0 / samples = 38 / value = [0, 38, 0]
flipper_length_mm <= 196.5 / gini = 0.32 / samples = 5 / value = [1, 4, 0]
gini = 0.0 / samples = 1 / value = [1, 0, 0]
gini = 0.0 / samples = 4 / value = [0, 4, 0]

Our model selected the ‘flipper_length_mm’ feature to be the root of the tree.

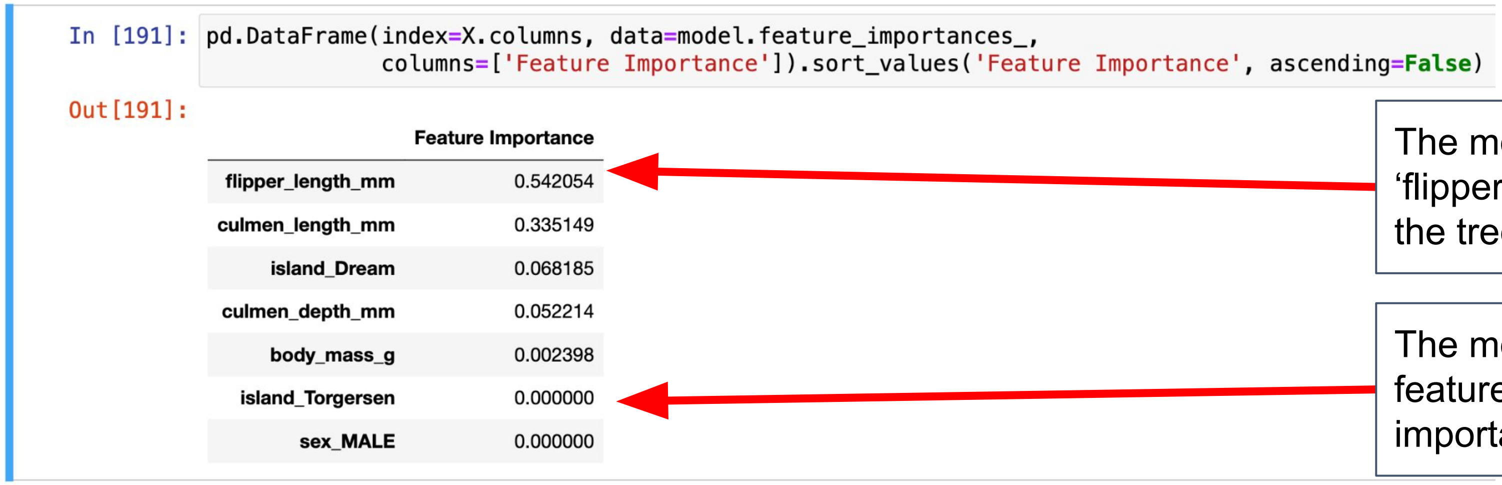
Inside each node we can see the following (according to this order)

- Split condition
- Gini Impurity value
- Samples → How many observations were relevant to this node
- Value → How many from the observations belong to each label class

Each color represent a different label category (Adelie / Chinstrap / Gentoo)

Decision Tree - Python Example

Python also provide us an easy way to see the level of importance each feature contribute to the model classification decision.



```
In [191]: pd.DataFrame(index=X.columns, data=model.feature_importances_,
                       columns=['Feature Importance']).sort_values('Feature Importance', ascending=False)

Out[191]:
```

	Feature Importance
flipper_length_mm	0.542054
culmen_length_mm	0.335149
island_Dream	0.068185
culmen_depth_mm	0.052214
body_mass_g	0.002398
island_Torgersen	0.000000
sex_MALE	0.000000

The most important feature was the 'flipper_length_mm' which was selected as the tree root node

The model also didn't used some of the features at all and mark them with 0 importance.

Note: By default, the Scikit-Learn decision tree algorithm splits the nodes on the best features until it reaches a Gini Impurity of 0. As we learned, A Gini impurity of 0 indicates that the node is perfectly homogeneous and all its samples belong to the same class, making it an ideal leaf node. So when the model encounters a node with a Gini impurity of 0, it can accurately classify new data based on that attribute.

Decision Tree Parameters

As we saw, the default decision tree will keep splitting the data and keep extending the tree depth until reaching node with Gini Impurity value of 0.
In some cases <u>this can make our model to be overfitting with our data</u> and unnecessary complex our model.

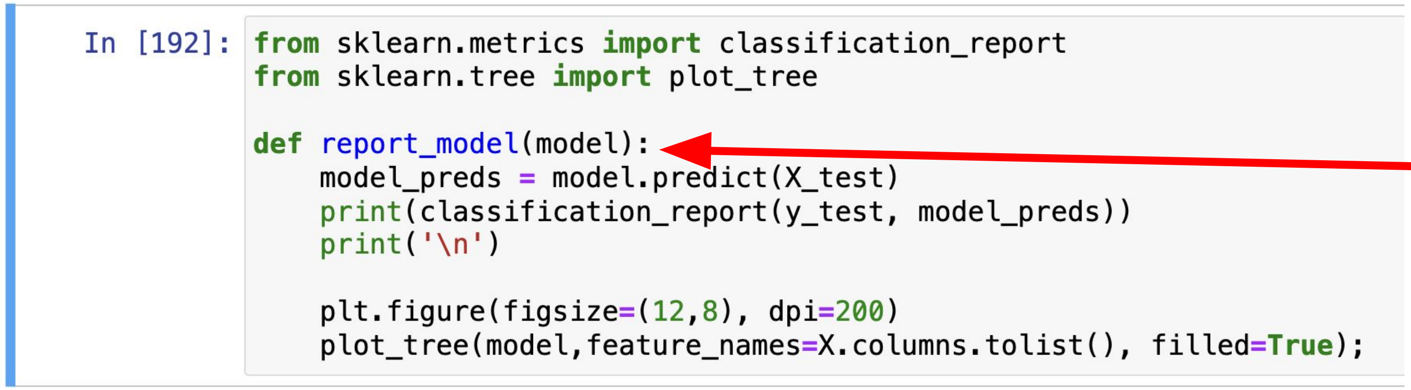
Scikit-Learn provide us number of model parameters that will help us preventing overfitting and over complexity of our model. Those parameters are:

- **\`max_depth\`** → Controls the maximum depth of the tree
- **\`min_samples_split\`** → The minimum number of samples required to split a node
- **\`min_samples_leaf\`** → The minimum number of samples required to be at a leaf node
- **\`max_features\`** → The number of features to consider when looking for the best split
- **\`max_leaf_nodes\`** → Best nodes are defined as relative reduction in impurity.

Decision Tree Parameters - Python Example

Let’s use our previous penguin example but now our model will also contained customized parameters.

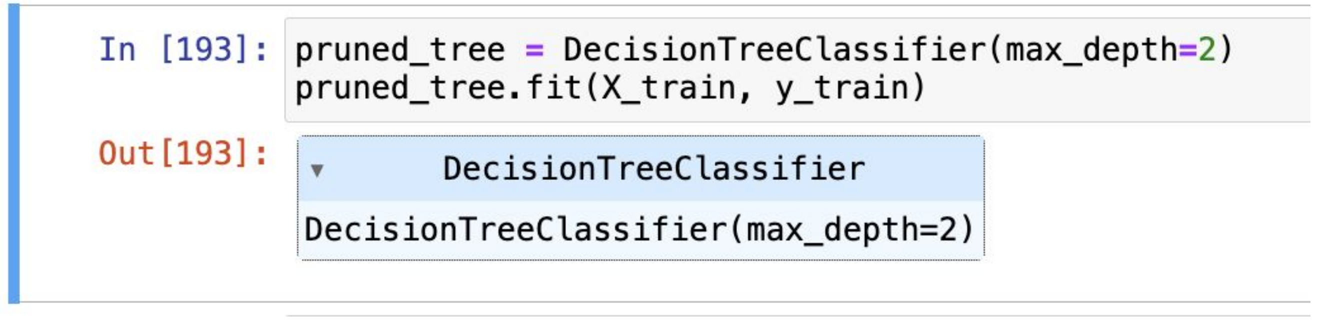
First, let’s create the following function so it will be easier for us to visualize each decision tree model:



```
In [192]: from sklearn.metrics import classification_report
          from sklearn.tree import plot_tree

          def report_model(model):
              model_preds = model.predict(X_test)
              print(classification_report(y_test, model_preds))
              print('\n')

              plt.figure(figsize=(12,8), dpi=200)
              plot_tree(model,feature_names=X.columns.tolist(), filled=True);
```

This function will help us visualize the decision tree model. By calling it and pass the model as parameter the function will print all the relevant metrics and the decision tree plot itself.

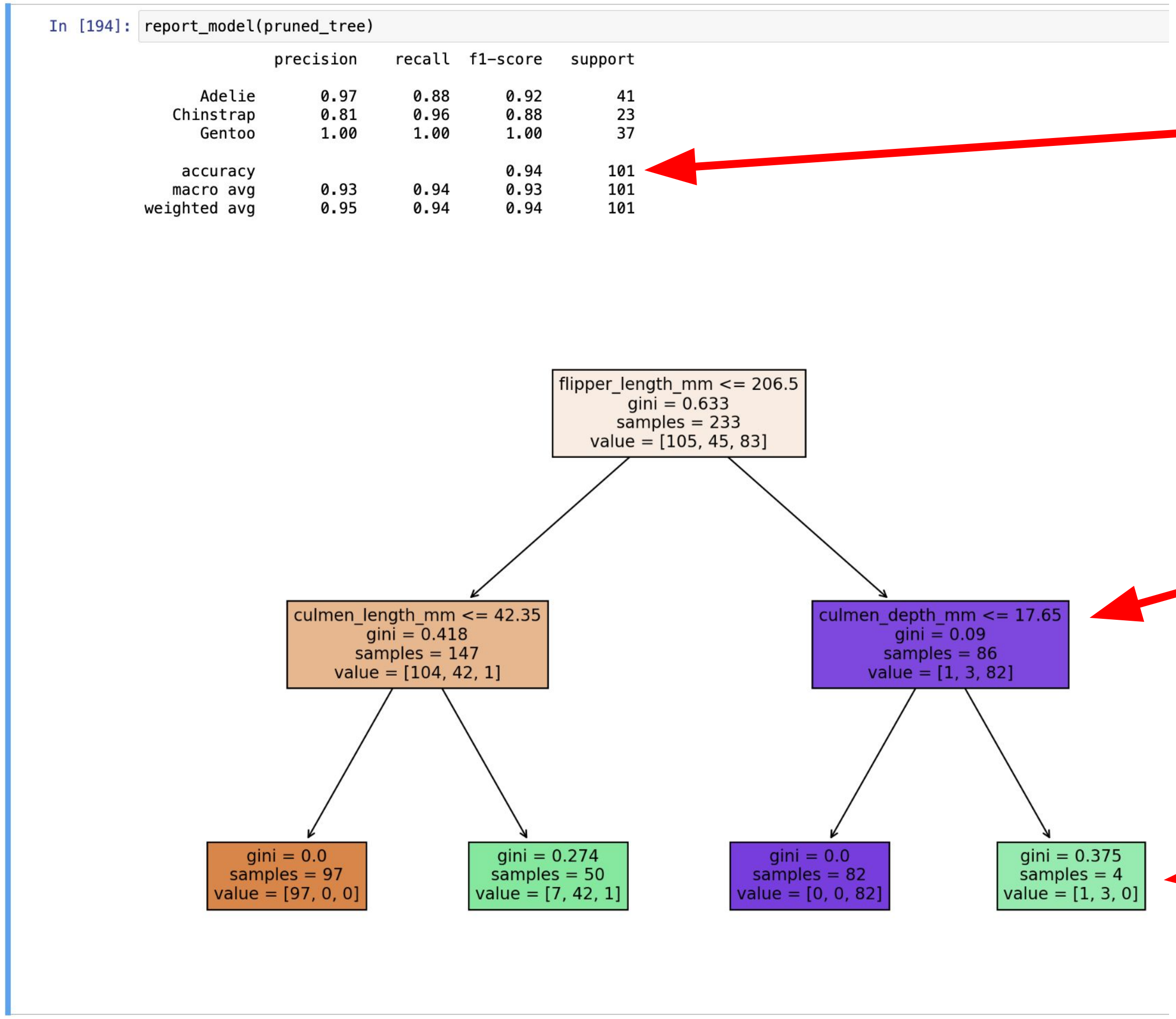
Now, let’s create model with **‘max_depth’ = 2** meaning we allowing our decision tree to be with depth of 2 from the root node.



```
In [193]: pruned_tree = DecisionTreeClassifier(max_depth=2)
          pruned_tree.fit(X_train, y_train)

Out[193]: ▾ DecisionTreeClassifier
          DecisionTreeClassifier(max_depth=2)
```

Decision Tree Parameters - Python Example

Let’s also call the report_model() function we wrote to visualize this decision tree:



```
In [194]: report_model(pruned_tree)
```

```
              precision    recall  f1-score   support

      Adelie       0.97      0.88      0.92        41
   Chinstrap       0.81      0.96      0.88        23
      Gentoo       1.00      1.00      1.00        37

    accuracy                           0.94       101
   macro avg       0.93      0.94      0.93       101
weighted avg       0.95      0.94      0.94       101
```

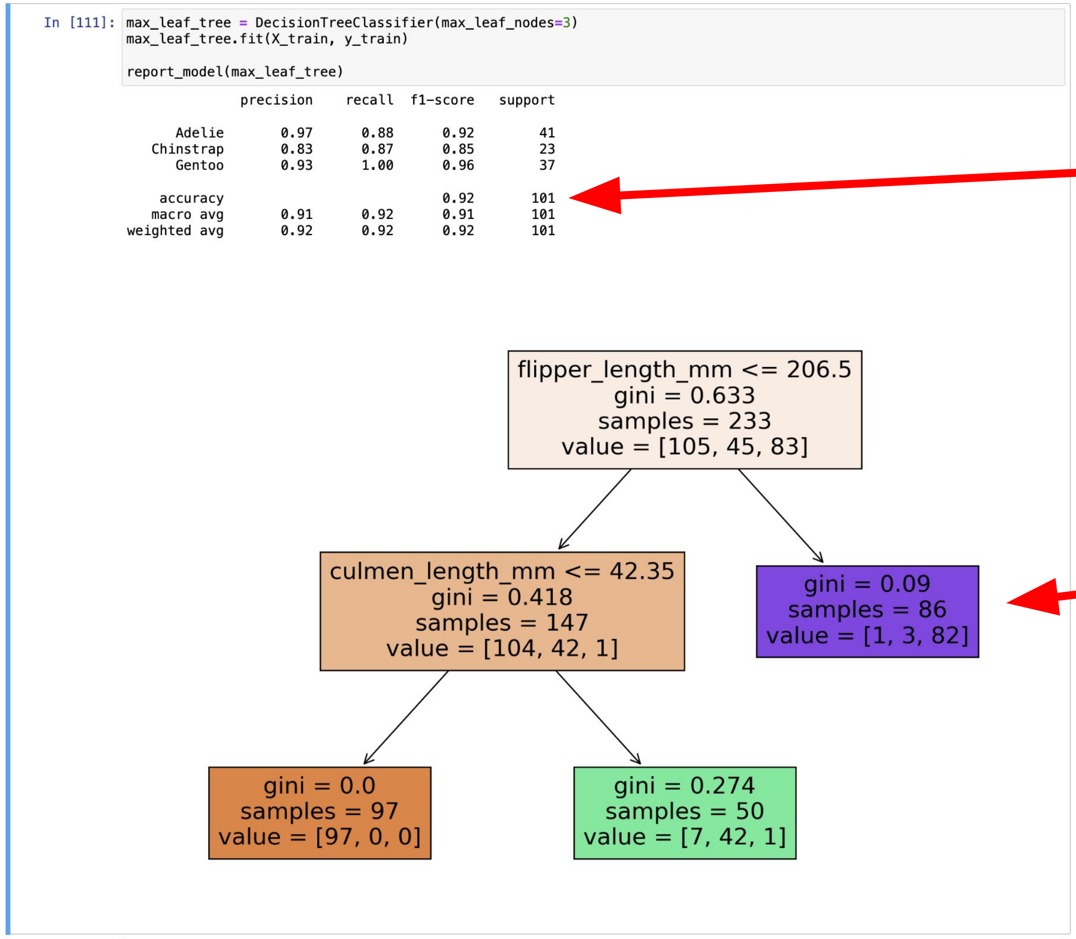
We got the same accuracy of 94% with much simpler decision tree

The decision tree is now of depth of 2 because of the max_depth parameter we provided

The Gini Impurity value of the leaf nodes is no longer always 0 as with the default model

Decision Tree Parameters - Python Example

Let’s also see another example with **‘max_leaf_nodes’ = 3** meaning we only allow the diction tree to contain maximum of 3 leaf nodes in total.



```
In [111]: max_leaf_tree = DecisionTreeClassifier(max_leaf_nodes=3)
          max_leaf_tree.fit(X_train, y_train)

          report_model(max_leaf_tree)
```

```
              precision    recall  f1-score   support

      Adelie       0.97      0.88      0.92        41
   Chinstrap       0.83      0.87      0.85        23
      Gentoo       0.93      1.00      0.96        37

    accuracy                           0.92       101
   macro avg       0.91      0.92      0.91       101
weighted avg       0.92      0.92      0.92       101
```

We got accuracy of 92% which is a little less than before (94%) but again our decision tree is much simpler.

Our decision tree now has only 3 leaf nodes in total because of the ‘max_leaf_nodes’ parameter we provided