Portfolio assignment 11

20 min: Do a Numerical VS Categorical bivariate analysis on the penguins dataset.

- Choose one of the categorical columns: species, island or sex
- use .groupby('').mean() too look at the means of the numerical columns. Does it look like there is a difference between categories?
- Use the seaborn barplot to plot the mean and confidence. Create this plot for each of the numerical columns (bill_length_mm bill_depth_mm, flipper_length_mm, body_mass_g)
- For each of the plots, write a conclusion: Is there a statistically significant difference for this numerical column for each category?
- Optional: Repeat this proces for the other two categorical columns

In [ ]:

```
import seaborn as sns
penguins = sns.load_dataset("penguins")
```

In [ ]:

```
penguins.head()
```

Out[ ]:

	species	island	bill_length_mm	bill_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

In [ ]:

```
penguins.groupby("island").mean()
```

Out[ ]:

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
island				
Biscoe	45.257485	15.874850	209.706587	4716.017964
Dream	44.167742	18.344355	193.072581	3712.903226
Torgersen	38.950980	18.429412	191.196078	3706.372549

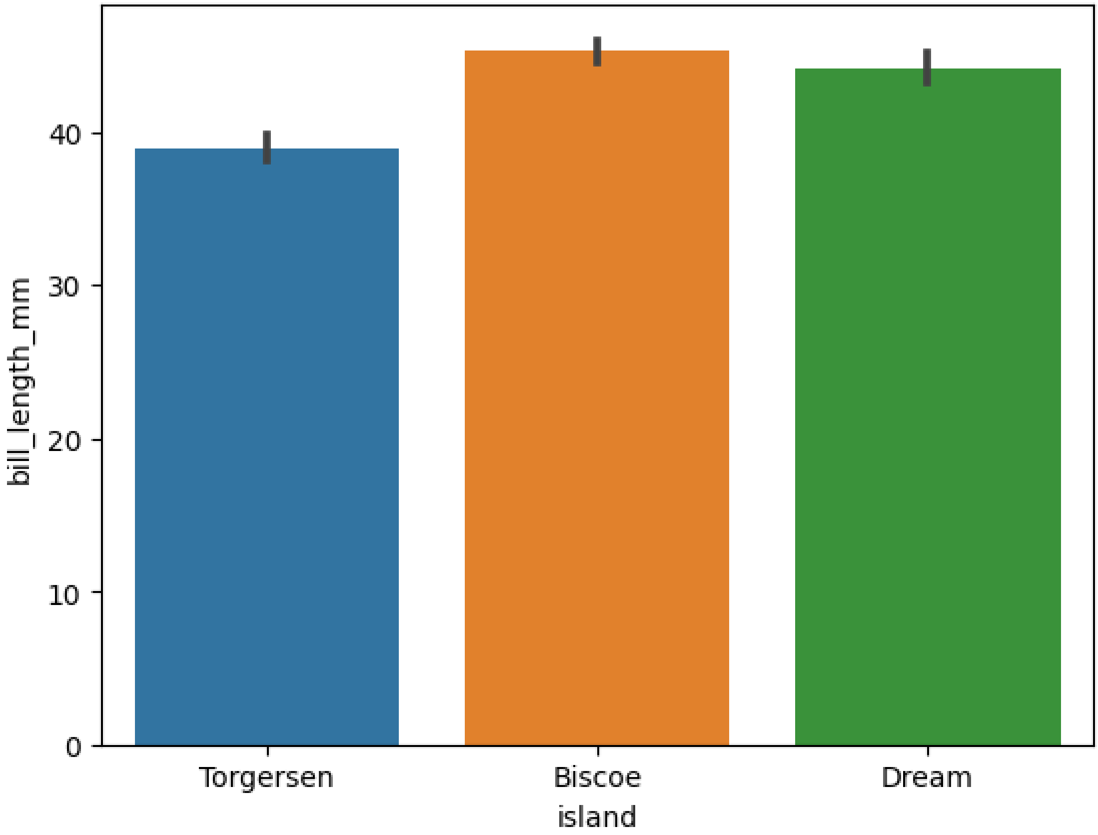
Ja, er is verschil aanwezig in bijvoorbeeld de lengte van de vleugel van de pinguins op verschillende eilanden.

In [ ]:

```
sns.barplot(x="island", y="bill_length_mm", data=penguins)
```

Out[ ]:

```
<AxesSubplot: xlabel='island', ylabel='bill_length_mm'>
```



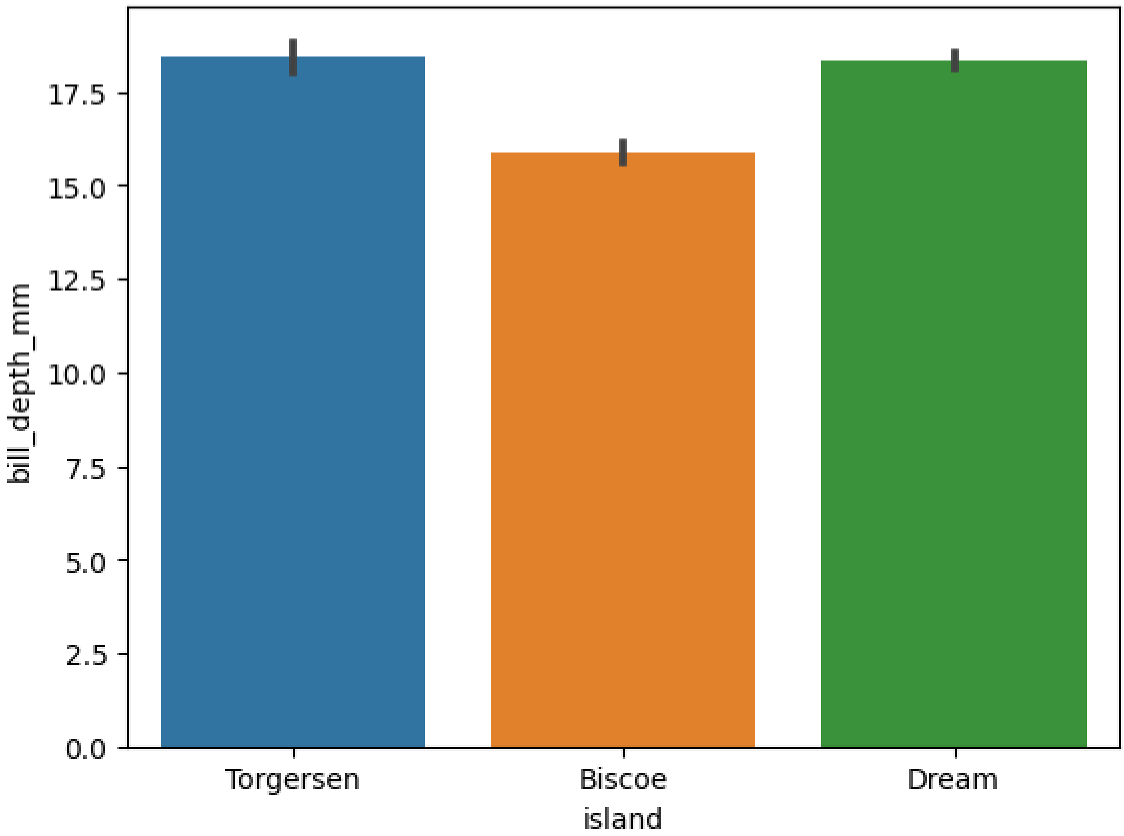
Alleen Torgersen heeft een significant verband met de snavellengte.

In [ ]:

```
sns.barplot(x="island", y="bill_depth_mm", data=penguins)
```

Out[ ]:

```
<AxesSubplot: xlabel='island', ylabel='bill_depth_mm'>
```

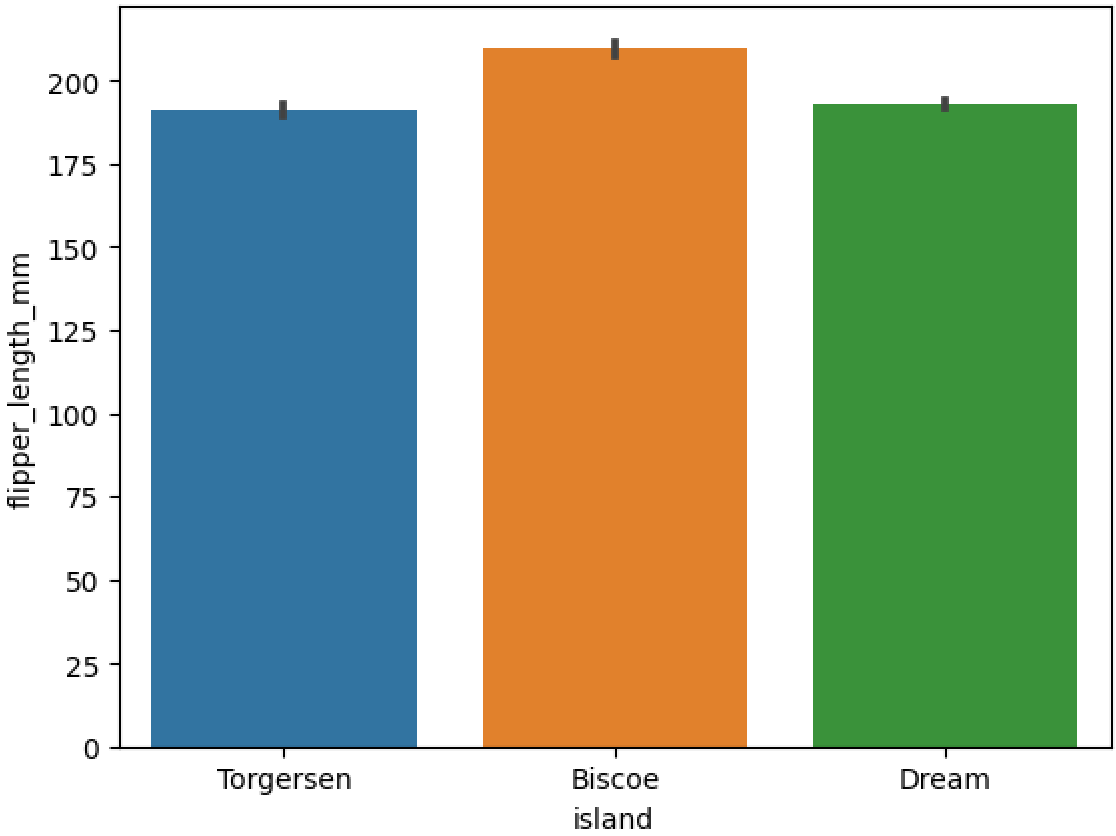


Alleen Biscoe heeft een significant verband met de snaveldiepte.

In [ ]:

```
sns.barplot(x="island", y="flipper_length_mm", data=penguins)
```

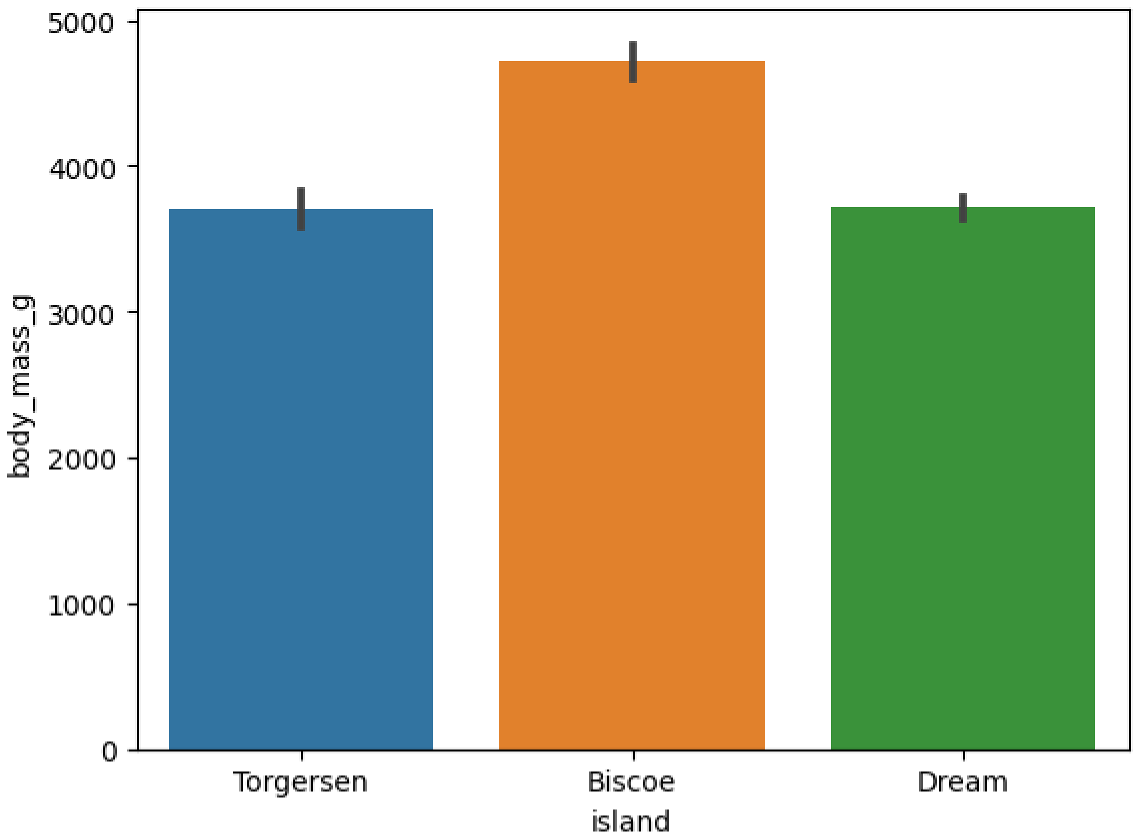
Out[ ]:
```
<AxesSubplot: xlabel='island', ylabel='flipper_length_mm'>
```



Biscoe heeft een significant verband met de vleugellengte.

In [ ]:
```
sns.barplot(x="island", y="body_mass_g", data=penguins)
```

Out[ ]:
```
<AxesSubplot: xlabel='island', ylabel='body_mass_g'>
```



Biscoe heeft een significant verband met het gewicht.

Portfolio assignment 12

30 min: Perform a bivariate analysis on at least 3 combinations of a numerical column with a categorical column in the dataset that you chose in portfolio assignment 4. Use *.groupby('columnname').mean()* to calculate the means. Is there a difference between categories? Then use seaborn barplots to check if there is a statistically significant difference.

In [ ]:

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

In [ ]:

```
pokemon = pd.read_csv("../pokemon.csv", sep=",")
pokemon.head()
```

Out[ ]:

	abilities	against_bug	against_dark	against_dragon	against_electric	against_fairy	against_fight	agai
0	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	2.0
1	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	2.0
2	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	2.0
3	['Blaze', 'Solar Power']	0.5	1.0	1.0	1.0	0.5	1.0	0.5
4	['Blaze', 'Solar Power']	0.5	1.0	1.0	1.0	0.5	1.0	0.5

5 rows × 41 columns

In [ ]:

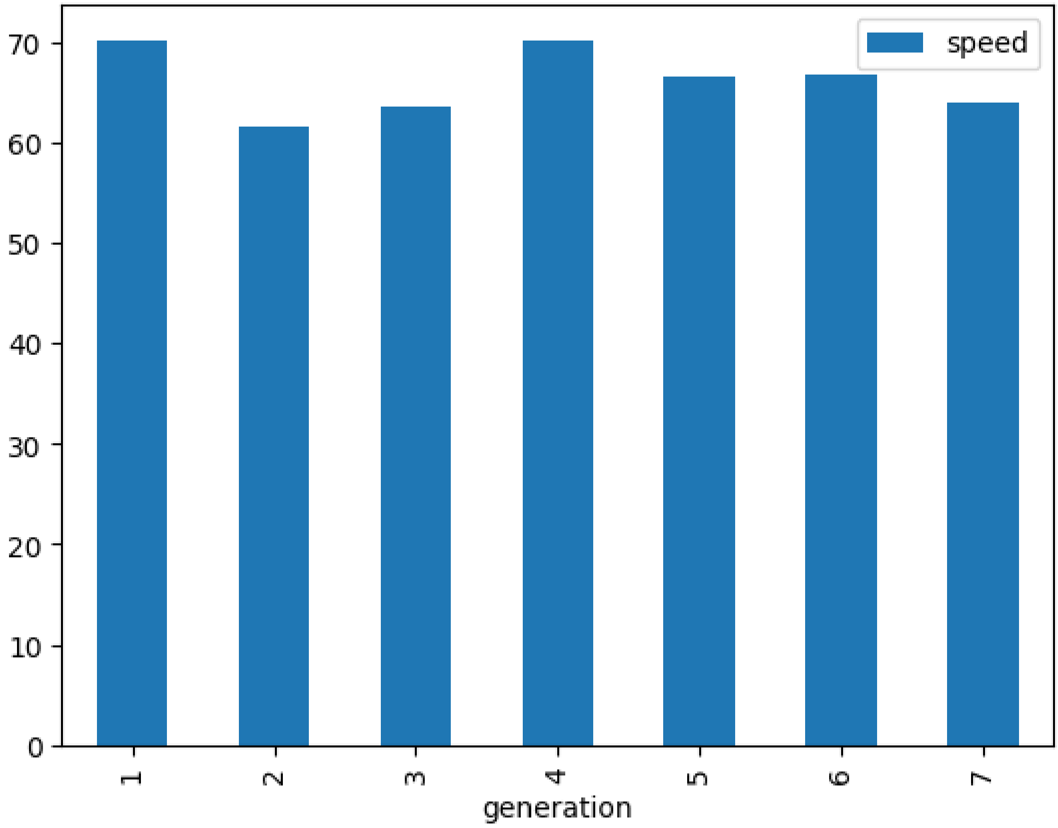
```
generation = pokemon.groupby('generation').mean()
generation["generation"] = generation.index
```

In [ ]:

```
generation.plot(x="generation", y="speed", kind="bar")
```

Out[ ]:

```
<AxesSubplot: xlabel='generation'>
```

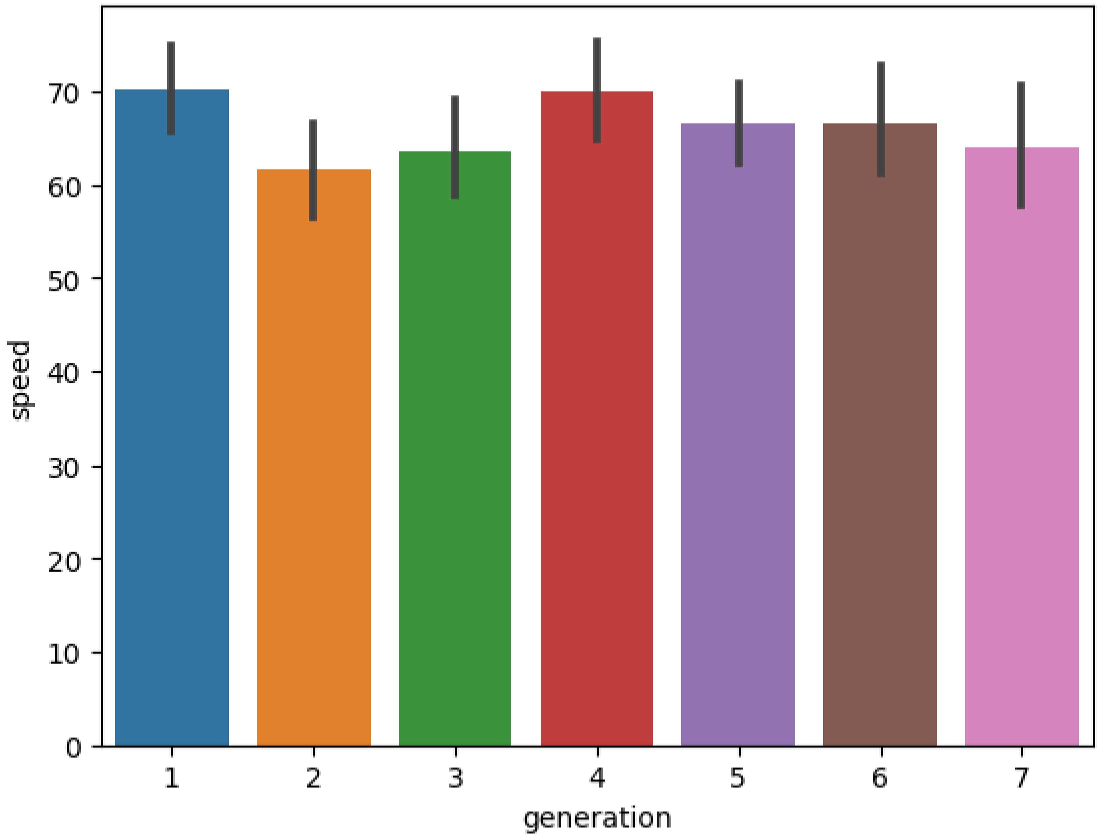


In [ ]:

```
sns.barplot(x="generation", y="speed", data=pokemon)
```

Out[ ]:

```
<AxesSubplot: xlabel='generation', ylabel='speed'>
```

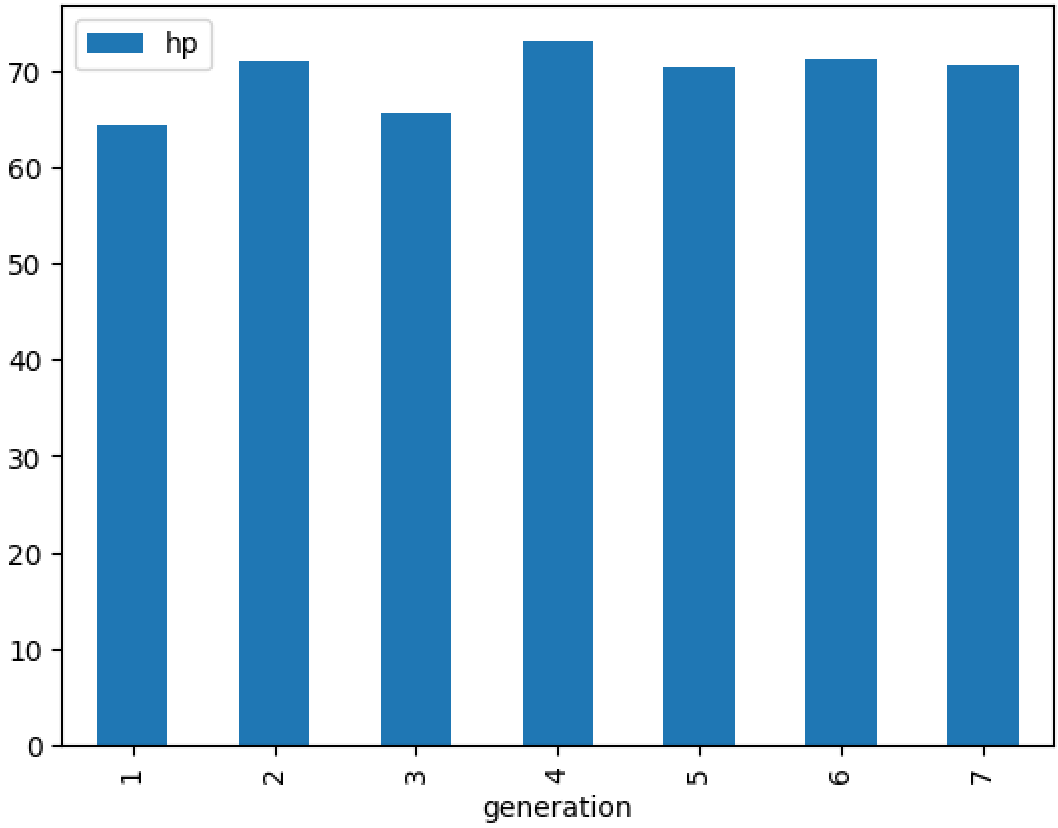


In [ ]:

```
generation.plot(x="generation", y="hp", kind="bar")
```

Out[ ]:

```
<AxesSubplot: xlabel='generation'>
```

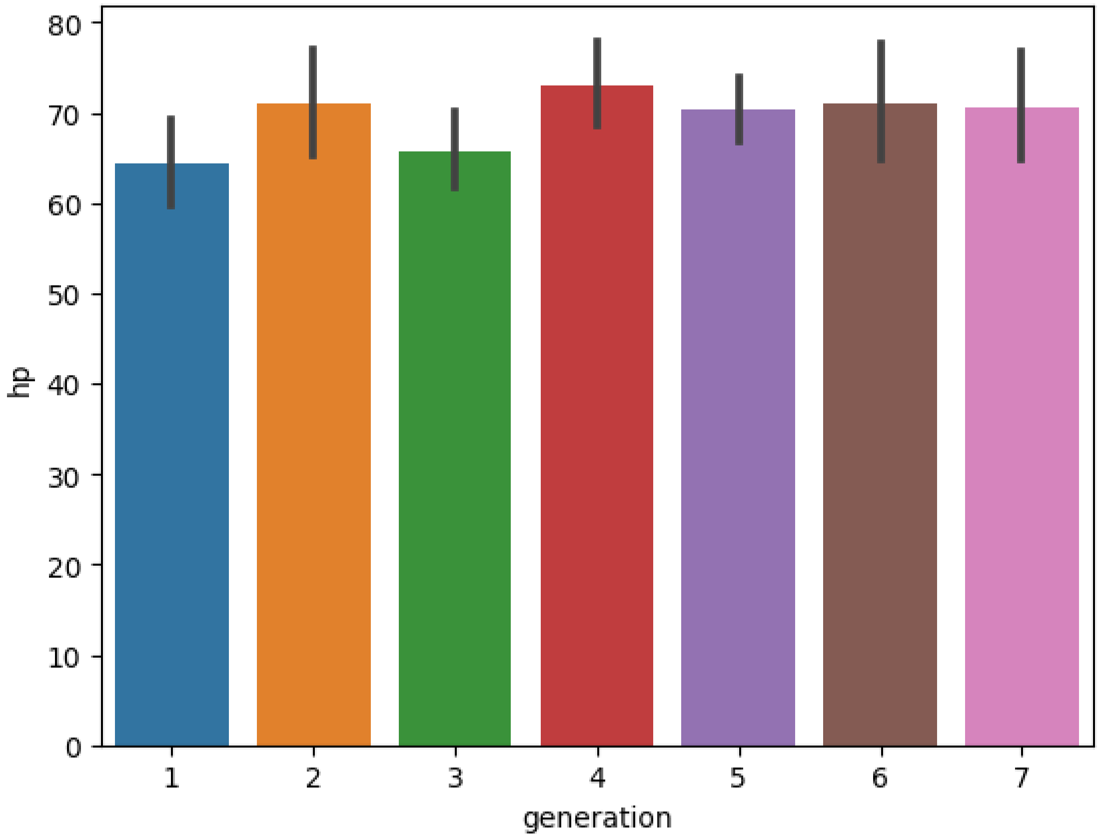


In [ ]:

```
sns.barplot(x="generation", y="hp", data=pokemon)
```

Out[ ]:

```
<AxesSubplot: xlabel='generation', ylabel='hp'>
```

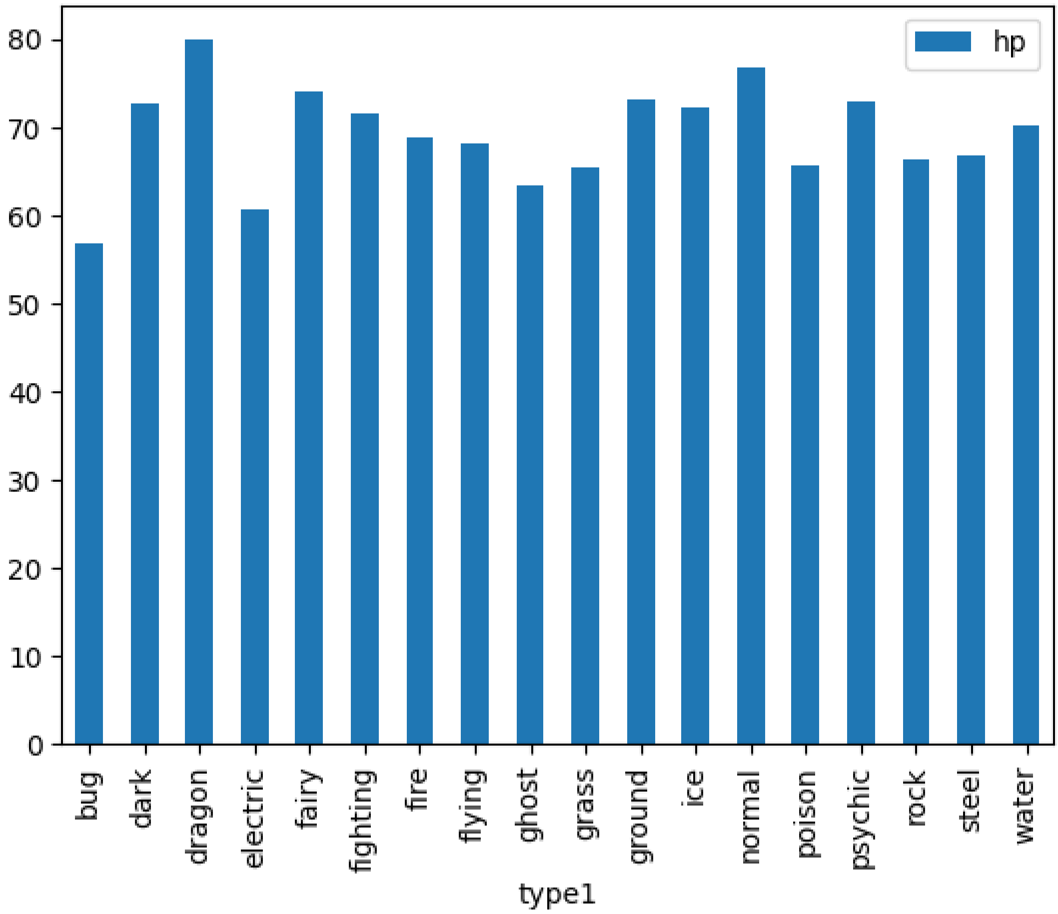


In [ ]:

```
type1 = pokemon.groupby('type1').mean()
type1["type1"] = type1.index

type1.plot(x="type1", y="hp", kind="bar")
```

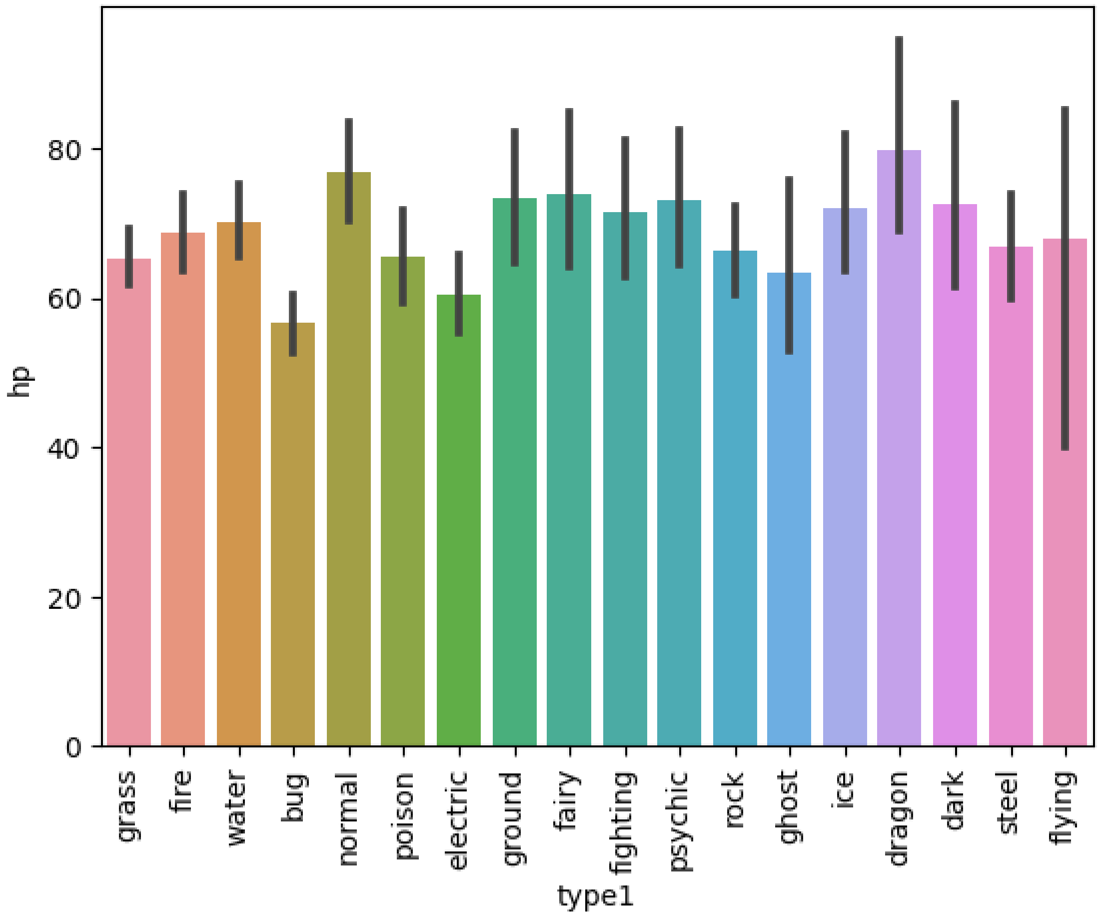
Out[ ]:

```
<AxesSubplot: xlabel='type1'>
```



In [ ]:

```
sns.barplot(x="type1", y="hp", data=pokemon)
plt.xticks(rotation=90)
plt.show()
```



Alleen bij bovenstaande chart is er een significant verschil aanwezig bij het type bug.

Portfolio assignment 13

10 min: Do a bivariate analysis on the penguins dataset for the following combination of columns:

- species VS sex
- island VS sex

For this bivariate analysis, at least perform the following tasks:

- Do you expect their to be a correlation between the two columns?
- Create a contingency table. Do you observe different ratios between categories here?
- Create a bar plot for this contingency table. Do you observe different ratios between categories here?
- Do a chi-squared test. What does the result say? What's the chance of there being a correlation between the two columns?

In [ ]:

```python
import seaborn as sns
penguins = sns.load_dataset("penguins")

penguins.head()
```

Out[ ]:

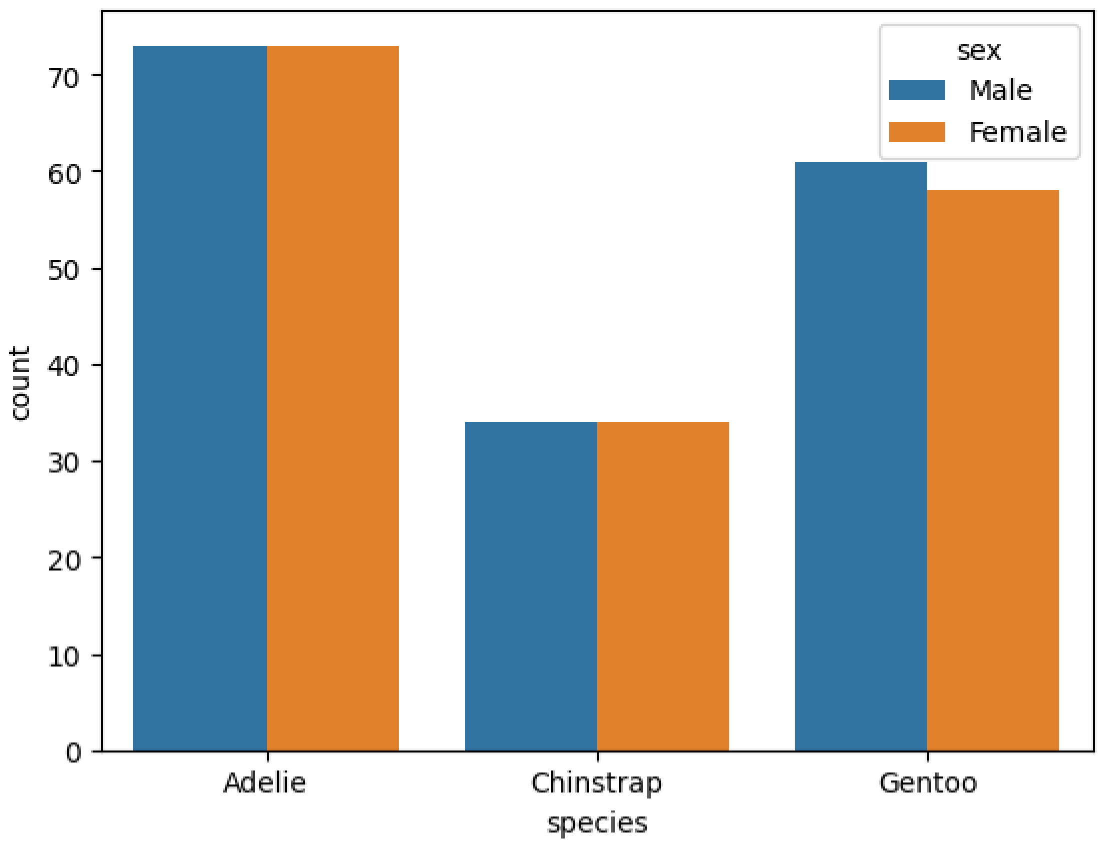
	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female

In [ ]:

```python
# species VS sex
sns.countplot(x='species', hue='sex', data=penguins)
```

Out[ ]:

```
<AxesSubplot: xlabel='species', ylabel='count'>
```



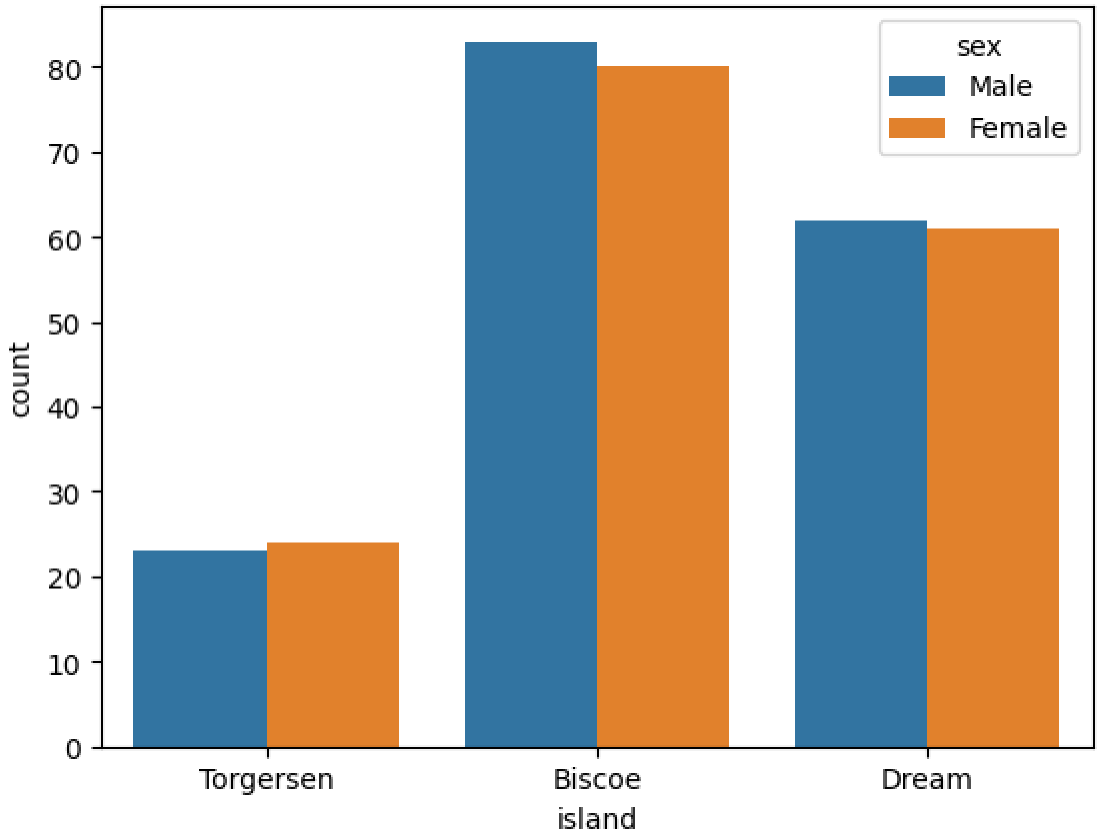
Er zit geen correlatie tussen species en sex. Alle soorten zijn gelijkmatig verdeeld op basis van het type sex.

In [ ]:

```
# island VS sex
sns.countplot(x='island', hue='sex', data=penguins)
```

Out[ ]:

```
<AxesSubplot: xlabel='island', ylabel='count'>
```



Er zit geen correlatie tussen island en sex. Alle eilanden zijn gelijkmatig verdeeld op basis van het type sex.

In [ ]:

```
from scipy.stats import chi2_contingency

def create_contingency_table(dataset, column1, column2):
    return dataset.groupby([column1, column2]).size().unstack(column1, fill_value=0)

def check_correlation(dataset, column1, column2):
    contingency_table = create_contingency_table(dataset, column1, column2)
    chi2 = chi2_contingency(contingency_table)
    p_value = chi2[1]
    odds_of_correlation = 1 - p_value
    print(f"The odds of a correlation between {column1} and {column2} is {odds_of_correlatio
    print("This percentage needs to be at least 95% for a significant correlation.")
```

In [ ]:

```
penguinsContingencyTable = create_contingency_table(penguins, 'species','sex')

penguinsContingencyTable
```

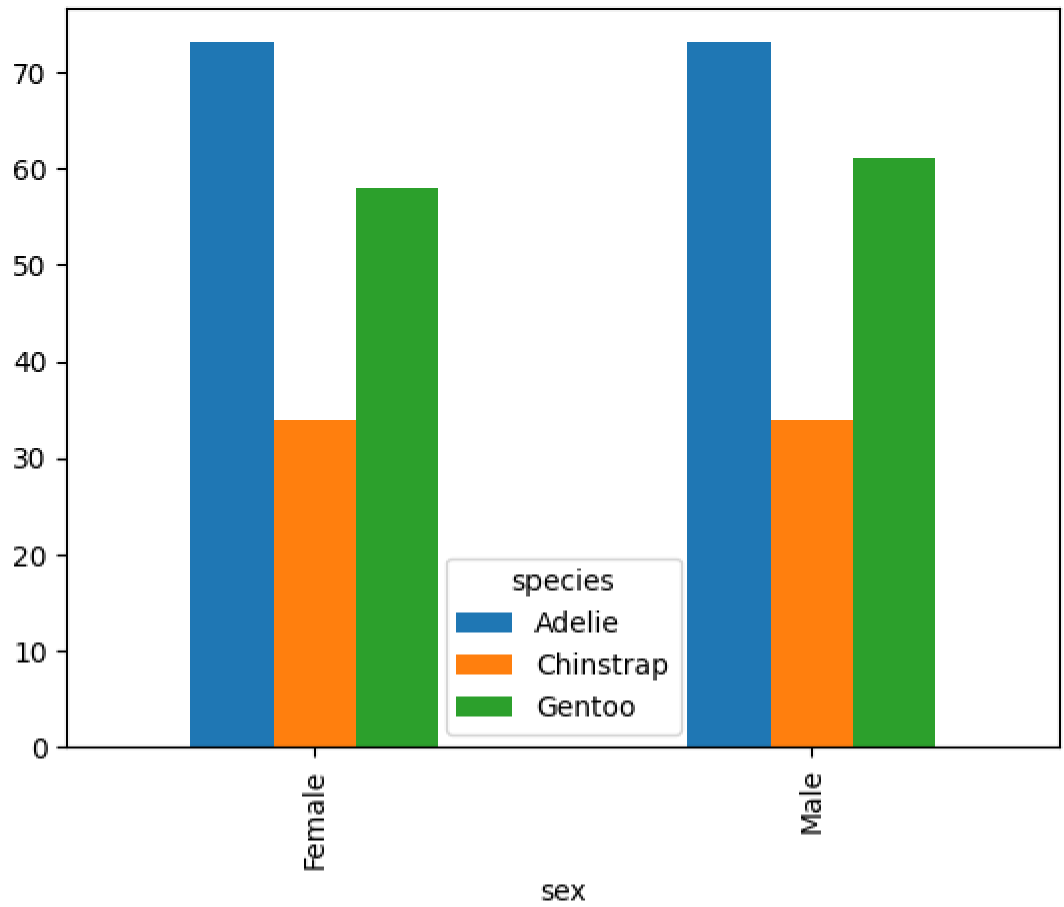
Out[ ]:

species	Adelie	Chinstrap	Gentoo
sex			
Female	73	34	58
Male	73	34	61

In [ ]:

```
penguinsContingencyTable.plot(kind='bar')
```

Out[ ]:
```
<AxesSubplot: xlabel='sex'>
```



Je ziet geen (grote) verschillen in de kolom sex op basis van de soort.

In [ ]:
```
check_correlation(penguins, 'species', 'sex')
```

```
The odds of a correlation between species and sex is 2.4010631023415385% (Based on a p value o
This percentage needs to be at least 95% for a significant correlation.
```

De Chi2 test bevestigt dat er geen grote verschillen aanwezig zijn.

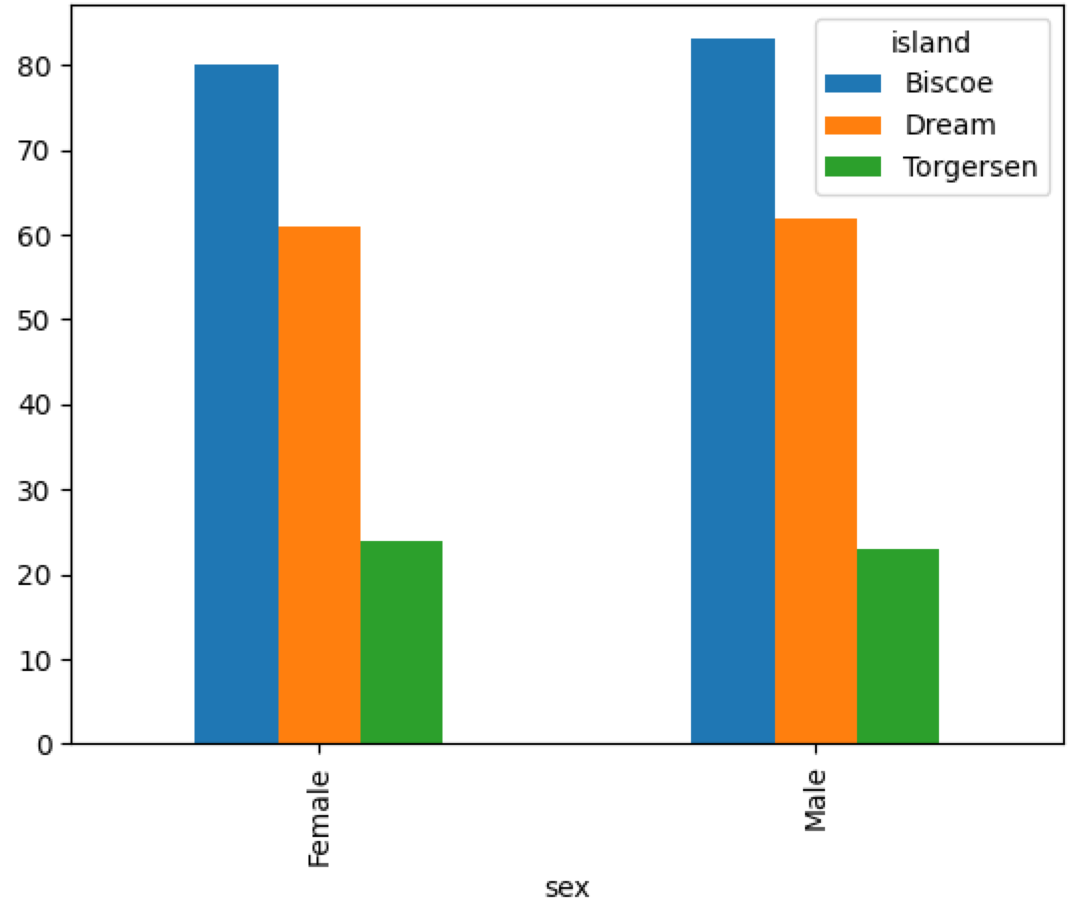
In [ ]:
```
penguinsContingencyTable = create_contingency_table(penguins, 'island','sex')

penguinsContingencyTable
```

Out[ ]:

island	Biscoe	Dream	Torgersen
sex			
Female	80	61	24
Male	83	62	23

In [ ]:
```
penguinsContingencyTable.plot(kind='bar')
```

Out[ ]:
```
<AxesSubplot: xlabel='sex'>
```



Je ziet geen (grote) verschillen in de kolom sex op de verschillende eilanden.

In [ ]:

```
check_correlation(penguins, 'island', 'sex')
```

```
The odds of a correlation between island and sex is 2.8388770718934975% (Based on a p value of
This percentage needs to be at least 95% for a significant correlation.
```

De Chi2 test bevestigt dat er geen grote verschillen aanwezig zijn.

Portfolio assignment 14

Perform a bivariate analysis on at least 1 combination of 2 columns with categorical data in the dataset that you chose in portfolio assignment 4.

- Do you expect their to be a correlation between the two columns?
- Create a contingency table. Do you observe different ratios between categories here?
- Create a bar plot for this contingency table. Do you observe different ratios between categories here?
- Do a chi-squared test. What does the result say? What's the chance of there being a correlation between the two columns?

In [ ]:

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

pokemon = pd.read_csv("../pokemon.csv", sep=",")
pokemon.head()
```

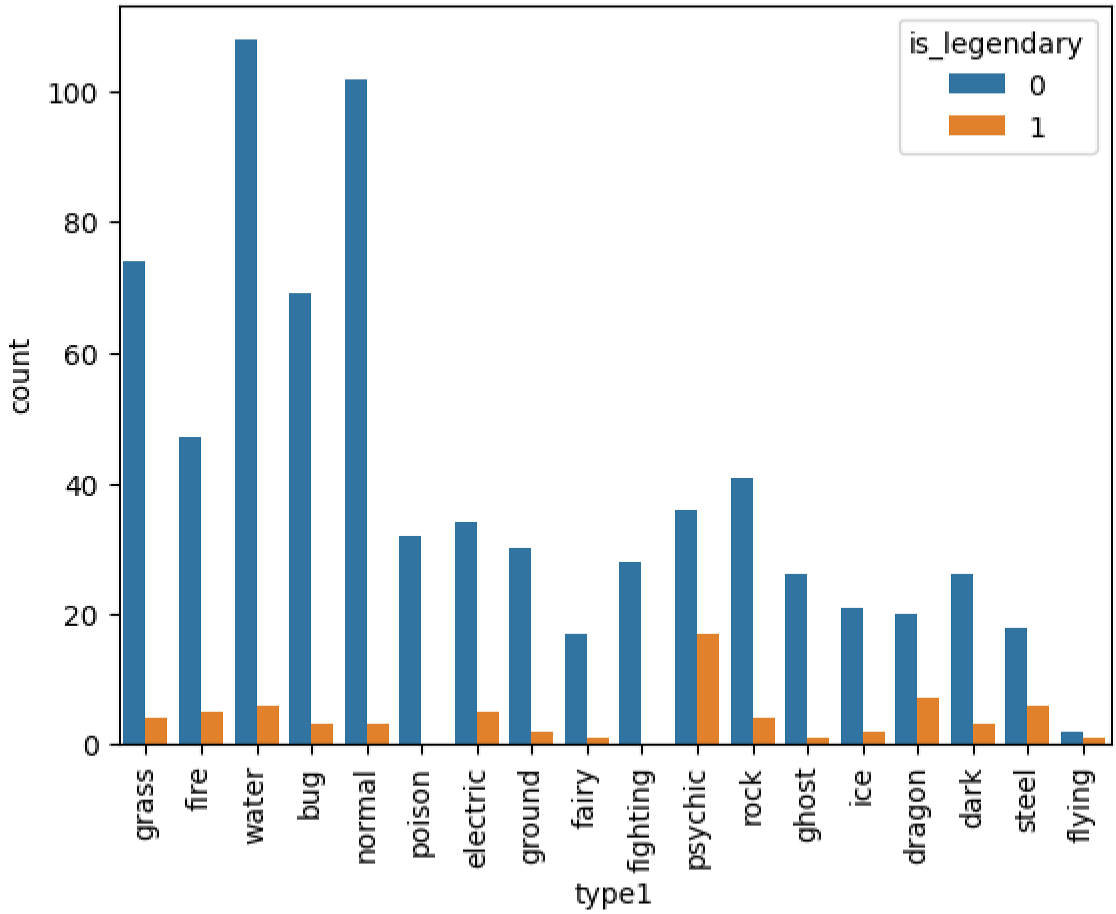
Out[ ]:

	abilities	against_bug	against_dark	against_dragon	against_electric	against_fairy	against_fight	agai
0	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	2.0
1	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	2.0
2	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	2.0
3	['Blaze', 'Solar Power']	0.5	1.0	1.0	1.0	0.5	1.0	0.5
4	['Blaze', 'Solar Power']	0.5	1.0	1.0	1.0	0.5	1.0	0.5

5 rows × 41 columns

In [ ]:

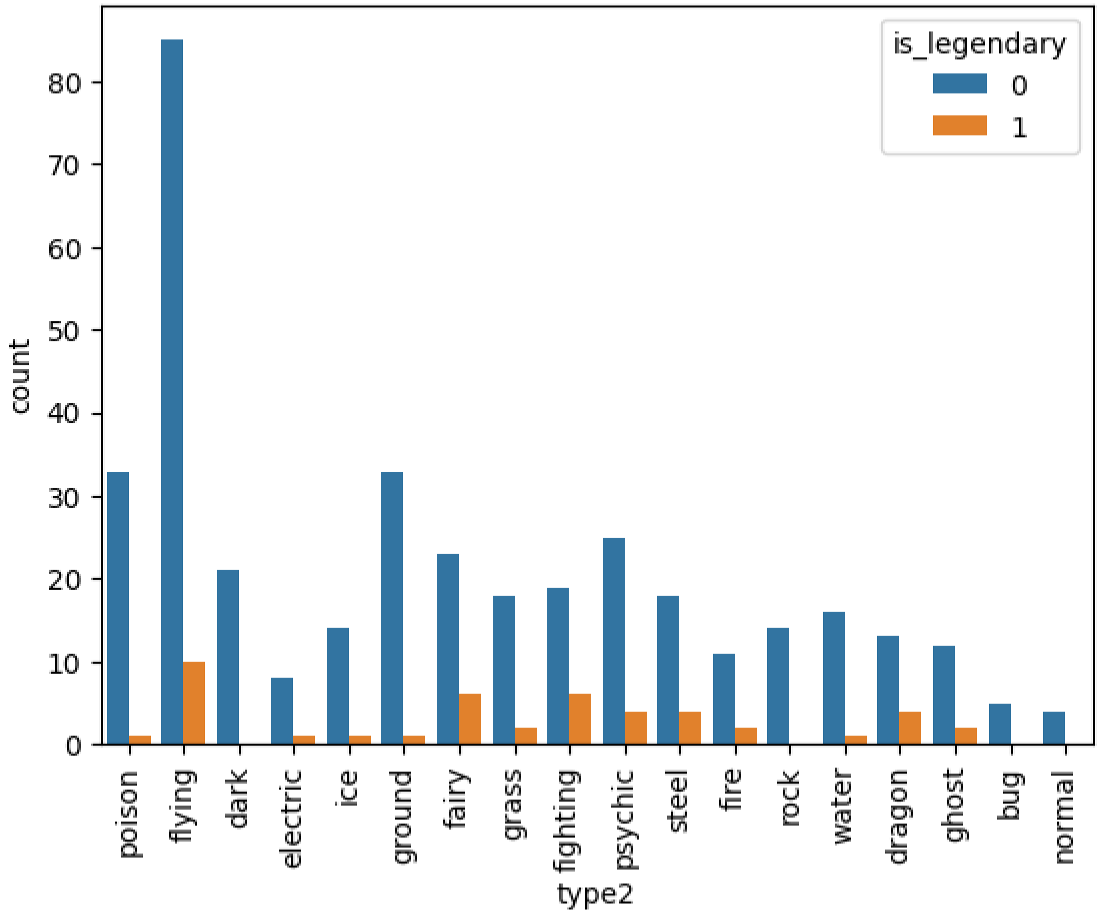
```
# type1 VS legendary
sns.countplot(x='type1', hue='is_legendary', data=pokemon)
plt.xticks(rotation=90)
plt.show()
```



Er lijkt geen correlatie tussen type1 en legendary te zitten. Wel heb je bij sommige types uitschieters. Of dit nu echt een verband is weet ik niet precies.

In [ ]:

```
# type2 VS legendary
sns.countplot(x='type2', hue='is_legendary', data=pokemon)
plt.xticks(rotation=90)
plt.show()
```



Er lijkt geen correlatie tussen type2 en legendary te zitten. Bij hoger pieken van niet-legendary heb je ook een hoge piek van wel-legendary.

In [ ]:

```
from scipy.stats import chi2_contingency

def create_contingency_table(dataset, column1, column2):
    return dataset.groupby([column1, column2]).size().unstack(column1, fill_value=0)

def check_correlation(dataset, column1, column2):
    contingency_table = create_contingency_table(dataset, column1, column2)
    chi2 = chi2_contingency(contingency_table)
    p_value = chi2[1]
    odds_of_correlation = 1 - p_value
    print(f"The odds of a correlation between {column1} and {column2} is {odds_of_correlatio
    print("This percentage needs to be at least 95% for a significant correlation.")
```

In [ ]:

```
contingencyTable = create_contingency_table(pokemon, 'is_legendary','type1')

contingencyTable
```

Out[ ]:

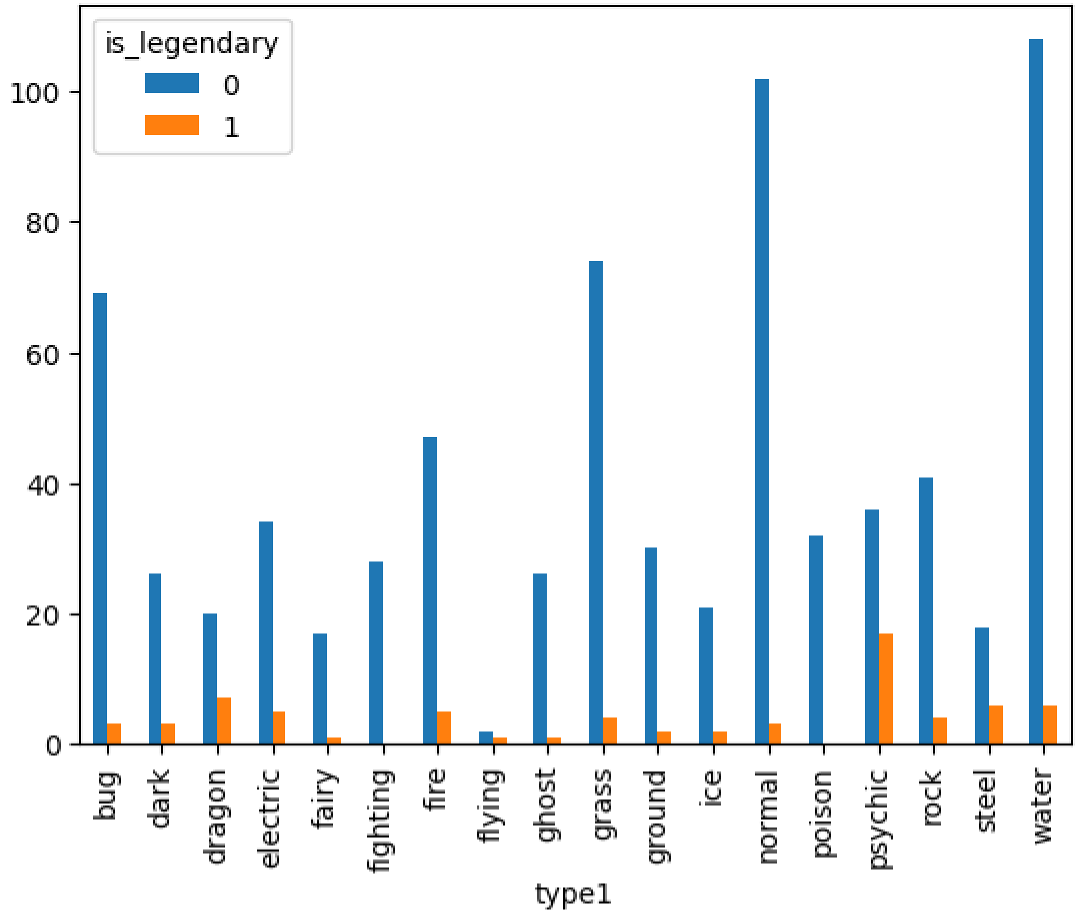
is_legendary	0	1
type1		
bug	69	3
dark	26	3
dragon	20	7
electric	34	5
fairy	17	1
fighting	28	0
fire	47	5
flying	2	1
ghost	26	1
grass	74	4
ground	30	2
ice	21	2
normal	102	3
poison	32	0
psychic	36	17
rock	41	4
steel	18	6
water	108	6

In [ ]:

```
contingencyTable.plot(kind='bar')
```

Out[ ]:

```
<AxesSubplot: xlabel='type1'>
```



Het type psychic zijn er veel legendary. Dit kan wel eens een correlatie hebben.

In [ ]:

```
check_correlation(pokemon, 'type1', 'is_legendary')
```

```
The odds of a correlation between type1 and is_legendary is 99.99999995467418% (Based on a p va
This percentage needs to be at least 95% for a significant correlation.
```

De Chi2 test bevestigt dat er een correlatie aanwezig is.

In [ ]:

```
contingencyTable = create_contingency_table(pokemon, 'is_legendary','type2')

contingencyTable
```

Out[ ]:

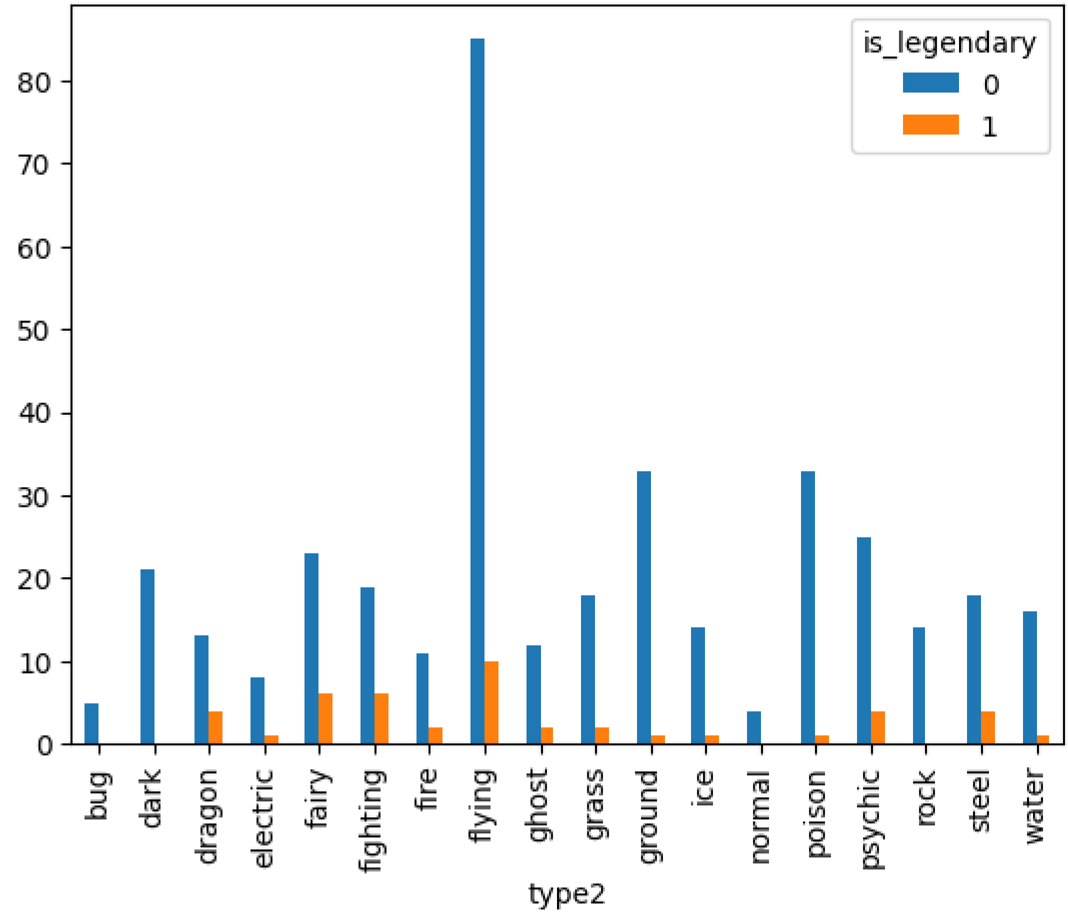
is_legendary	0	1
type2		
bug	5	0
dark	21	0
dragon	13	4
electric	8	1
fairy	23	6
fighting	19	6
fire	11	2
flying	85	10
ghost	12	2
grass	18	2
ground	33	1
ice	14	1
normal	4	0
poison	33	1
psychic	25	4
rock	14	0
steel	18	4
water	16	1

In [ ]:

```
contingencyTable.plot(kind='bar')
```

Out[ ]:

```
<AxesSubplot: xlabel='type2'>
```



Ik zie geen opmerkelijke correlatie. Bij hoge pieken heb je ook veel legendary. Dat lijkt mij logisch.

In [ ]:

```
check_correlation(pokemon, 'type2', 'is_legendary')
```

```
The odds of a correlation between type2 and is_legendary is 84.1298006223832% (Based on a p va
This percentage needs to be at least 95% for a significant correlation.
```

De Chi2 test bevestigt dat er geen grote correlatie aanwezig is.

Portfolio assignment 15

30 min: Train a decision tree to predict the species of a penguin based on their characteristics.

- Split the penguin dataset into a train (70%) and test (30%) set.
- Use the train set to fit a DecisionTreeClassifier. You are free to to choose which columns you want to use as feature variables and you are also free to choose the max_depth of the tree. **Note**: Some machine learning algorithms can not handle missing values. You will either need to
    - replace missing values (with the mean or most popular value). For replacing missing values you can use .fillna(\<value>) https://pandas.pydata.org/docs/reference/api/pandas.Series.fillna.html
    - remove rows with missing data. You can remove rows with missing data with .dropna() https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html
- Use your decision tree model to make predictions for both the train and test set.
- Calculate the accuracy for both the train set predictions and test set predictions.
- Is the accurracy different? Did you expect this difference?
- Use the plot_tree_classification function above to create a plot of the decision tree. Take a few minutes to analyse the decision tree. Do you understand the tree?

Optional: Perform the same tasks but try to predict the sex of the pinguin based on the other columns

In [ ]:

```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
```

In [ ]:

```
penguins = sns.load_dataset("penguins")
penguins.fillna(penguins.mean(), inplace=True)
penguins = penguins[penguins['sex'].notna()]

penguins_train, penguins_test = train_test_split(penguins, test_size=0.3, random_state=42, st
```

```
C:\Users\Jens\AppData\Local\Temp\ipykernel_4304\1888856847.py:2: FutureWarning: Dropping of nu
  penguins.fillna(penguins.mean(), inplace=True)
```

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:

```
from sklearn import tree
import graphviz

def plot_tree_classification(model, features, class_names):
    # Generate plot data
    dot_data = tree.export_graphviz(model, out_file=None,
                          feature_names=features,
                          class_names=class_names,
                          filled=True, rounded=True,
                          special_characters=True)

    # Turn into graph using graphviz
    graph = graphviz.Source(dot_data)

    # Write out a pdf
    graph.render("decision_tree")

    # Display in the notebook
    return graph
```

In [ ]:

```
def calculate_accuracy(predictions, actuals):
    if(len(predictions) != len(actuals)):
        raise Exception("The amount of predictions did not equal the amount of actuals")

    return (predictions == actuals).sum() / len(actuals)
```

Decision tree based on species

In [ ]:

```
features= ['bill_length_mm']
dt = DecisionTreeClassifier(max_depth = 2) # Increase max_depth to see effect in the plot
dt.fit(penguins_train[features], penguins_train['species'])
```

Out[ ]:

```
▸ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2)
```

In [ ]:

```
predictions = dt.predict(penguins_train[features])
predictions
```

Out[ ]:

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

In [ ]:

```
predictions = dt.predict(penguins_test[features])
predictions
```

Out[ ]:

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

In [ ]:

```
predictionsOnTrainset = dt.predict(penguins_train[features])
predictionsOnTestset = dt.predict(penguins_test[features])

accuracyTrain = calculate_accuracy(predictionsOnTrainset, penguins_train.species)
accuracyTest = calculate_accuracy(predictionsOnTestset, penguins_test.species)

print("Accuracy on training set " + str(accuracyTrain))
print("Accuracy on test set " + str(accuracyTest))
```

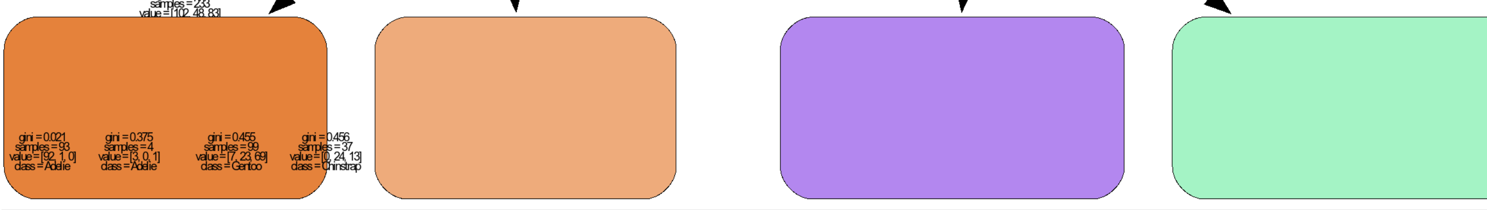
```
Accuracy on training set 0.8068669527896996
Accuracy on test set 0.7
```

The accuracy is different. This is because the model is based and optimalised on the trainings dataset.

In [ ]:

```
plot_tree_classification(dt, features, np.sort(penguins.species.unique()))
```

Out[ ]:



Decision tree based on sex

In [ ]:

```
features= ['bill_length_mm']
dt = DecisionTreeClassifier(max_depth = 2) # Increase max_depth to see effect in the plot
dt.fit(penguins_train[features], penguins_train['sex'])
```

Out[ ]:

```
▸ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2)
```

In [ ]:

```
predictions = dt.predict(penguins_train[features])
predictions
```

Out[ ]:

```
array(['Female', 'Male', 'Male', 'Male', 'Female', 'Female', 'Female',
       'Male', 'Female', 'Female', 'Female', 'Male', 'Female', 'Female',
       'Male', 'Female', 'Female', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Male', 'Female', 'Male', 'Male', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Male', 'Female', 'Male', 'Female',
       'Female', 'Male', 'Female', 'Female', 'Male', 'Male', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Female', 'Female', 'Male',
       'Female', 'Male', 'Male', 'Female', 'Male', 'Male', 'Female',
       'Male', 'Male', 'Female', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Male', 'Female', 'Female', 'Female', 'Male',
       'Female', 'Female', 'Female', 'Female', 'Male', 'Female', 'Female',
       'Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Female',
       'Female', 'Male', 'Female', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Male', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Male', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Male', 'Female', 'Female', 'Male',
       'Male', 'Female', 'Male', 'Male', 'Male', 'Female', 'Male', 'Male',
       'Male', 'Male', 'Female', 'Female', 'Female', 'Female', 'Male',
       'Female', 'Female', 'Male', 'Male', 'Male', 'Male', 'Female',
       'Male', 'Female', 'Female', 'Female', 'Female', 'Male', 'Female',
       'Female', 'Female', 'Male', 'Female', 'Male', 'Female', 'Female',
       'Female', 'Male', 'Male', 'Female', 'Female', 'Female', 'Male',
       'Male', 'Male', 'Female', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Male', 'Male', 'Female',
       'Male', 'Female', 'Male', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Male', 'Female', 'Male', 'Female', 'Female',
       'Male', 'Female', 'Male', 'Female', 'Female', 'Male', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Male', 'Female', 'Female',
       'Male', 'Female', 'Female', 'Female', 'Female', 'Female', 'Male',
       'Female', 'Female', 'Female', 'Female', 'Male', 'Female', 'Female',
       'Female', 'Female', 'Male', 'Male', 'Female', 'Female', 'Male',
       'Female', 'Female', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Female', 'Female', 'Male',
       'Female', 'Female'], dtype=object)
```

In [ ]:

```
predictions = dt.predict(penguins_test[features])
predictions
```

Out[ ]:

```
array(['Male', 'Male', 'Male', 'Female', 'Female', 'Male', 'Male',
       'Female', 'Female', 'Male', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Male', 'Female', 'Male',
       'Female', 'Female', 'Female', 'Female', 'Male', 'Female', 'Male',
       'Male', 'Male', 'Female', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Female', 'Male', 'Male',
       'Male', 'Female', 'Female', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Female', 'Female', 'Male',
       'Male', 'Female', 'Male', 'Male', 'Male', 'Male', 'Female',
       'Female', 'Female', 'Male', 'Female', 'Female', 'Female', 'Male',
       'Female', 'Female', 'Female', 'Male', 'Female', 'Female', 'Female',
       'Male', 'Male', 'Female', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Female', 'Male'], dtype=object)
```

In [ ]:

```
predictionsOnTrainset = dt.predict(penguins_train[features])
predictionsOnTestset = dt.predict(penguins_test[features])

accuracyTrain = calculate_accuracy(predictionsOnTrainset, penguins_train.sex)
accuracyTest = calculate_accuracy(predictionsOnTestset, penguins_test.sex)

print("Accuracy on training set " + str(accuracyTrain))
```

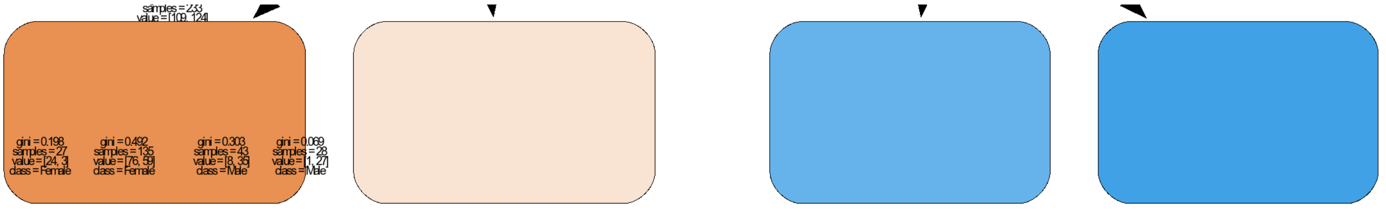
```
print("Accuracy on test set " + str(accuracyTest))
```

```
Accuracy on training set 0.6952789699570815
Accuracy on test set 0.69
```

In [ ]:

```
plot_tree_classification(dt, features, np.sort(penguins.sex.unique()))
```

Out[ ]:



Portfolio assignment 16

30 min: Train a decision tree to predict one of the categorical columns of your own dataset.

- Split your dataset into a train (70%) and test (30%) set.
- Use the train set to fit a DecisionTreeClassifier. You are free to to choose which columns you want to use as feature variables and you are also free to choose the max_depth of the tree.
- Use your decision tree model to make predictions for both the train and test set.
- Calculate the accuracy for both the train set predictions and test set predictions.
- Is the accurracy different? Did you expect this difference?
- Use the plot_tree function above to create a plot of the decision tree. Take a few minutes to analyse the decision tree. Do you understand the tree?

In [ ]:

```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
```

In [ ]:

```
pokemon = pd.read_csv("../pokemon.csv", sep=",")
# pokemon.head()
# penguins.fillna(penguins.mean(), inplace=True)
# penguins = penguins[penguins['sex'].notna()]

pokemon_train, pokemon_test = train_test_split(pokemon, test_size=0.3, random_state=42, strat
```

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:

```
from sklearn import tree
import graphviz

def plot_tree_classification(model, features, class_names):
    # Generate plot data
    dot_data = tree.export_graphviz(model, out_file=None,
                          feature_names=features,
                          class_names=class_names,
                          filled=True, rounded=True,
                          special_characters=True)

    # Turn into graph using graphviz
    graph = graphviz.Source(dot_data)

    # Write out a pdf
    graph.render("decision_tree")

    # Display in the notebook
    return graph
```

In [ ]:

```
def calculate_accuracy(predictions, actuals):
    if(len(predictions) != len(actuals)):
        raise Exception("The amount of predictions did not equal the amount of actuals")
```

```
    return (predictions == actuals).sum() / len(actuals)
```

Decision tree based on generation

In [ ]:

```
features= ['speed', 'hp', 'attack']
dt = DecisionTreeClassifier(max_depth = 3)
dt.fit(pokemon_train[features], pokemon_train['type1'])
```

Out[ ]:

```
▸ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

In [ ]:

```
predictions = dt.predict(pokemon_train[features])
predictions
```

Out[ ]:

```
array(['normal', 'normal', 'bug', 'bug', 'normal', 'normal', 'ice',
       'normal', 'normal', 'normal', 'water', 'normal', 'water', 'bug',
       'bug', 'normal', 'water', 'water', 'water', 'bug', 'normal', 'bug',
       'water', 'water', 'normal', 'normal', 'bug', 'bug', 'normal',
       'ice', 'normal', 'normal', 'normal', 'normal', 'water', 'normal',
       'normal', 'normal', 'normal', 'normal', 'normal', 'normal',
       'normal', 'normal', 'water', 'normal', 'normal', 'bug', 'normal',
       'bug', 'ice', 'water', 'water', 'normal', 'ice', 'water', 'normal',
       'water', 'normal', 'normal', 'water', 'normal', 'water', 'normal',
       'normal', 'normal', 'normal', 'bug', 'normal', 'water', 'normal',
       'normal', 'water', 'normal', 'normal', 'water', 'water', 'normal',
       'normal', 'normal', 'normal', 'normal', 'normal', 'bug', 'bug',
       'water', 'normal', 'normal', 'normal', 'normal', 'water', 'water',
       'normal', 'water', 'water', 'water', 'water', 'normal', 'normal',
       'normal', 'normal', 'normal', 'water', 'water', 'normal', 'normal',
       'bug', 'normal', 'normal', 'water', 'normal', 'bug', 'water',
       'normal', 'normal', 'water', 'psychic', 'water', 'ice', 'water',
       'normal', 'water', 'normal', 'water', 'normal', 'bug', 'normal',
       'normal', 'normal', 'normal', 'normal', 'normal', 'water', 'water',
       'bug', 'water', 'bug', 'water', 'normal', 'bug', 'normal', 'bug',
       'normal', 'water', 'normal', 'water', 'water', 'water', 'water',
       'water', 'bug', 'normal', 'bug', 'bug', 'psychic', 'normal', 'ice',
       'water', 'normal', 'normal', 'normal', 'normal', 'water', 'normal',
       'normal', 'normal', 'normal', 'bug', 'water', 'water', 'normal',
       'bug', 'normal', 'normal', 'bug', 'water', 'bug', 'normal',
       'normal', 'normal', 'water', 'bug', 'normal', 'water', 'normal',
       'water', 'normal', 'normal', 'water', 'normal', 'normal', 'normal',
       'normal', 'normal', 'water', 'bug', 'bug', 'psychic', 'water',
       'normal', 'normal', 'water', 'normal', 'water', 'water', 'water',
       'normal', 'bug', 'normal', 'normal', 'bug', 'bug', 'normal', 'bug',
       'normal', 'normal', 'water', 'normal', 'normal', 'normal', 'bug',
       'bug', 'normal', 'normal', 'normal', 'normal', 'water', 'water',
       'normal', 'water', 'normal', 'water', 'bug', 'normal', 'water',
       'normal', 'normal', 'normal', 'water', 'normal', 'normal', 'bug',
       'bug', 'normal', 'normal', 'water', 'water', 'bug', 'bug',
       'normal', 'normal', 'normal', 'normal', 'psychic', 'water',
       'normal', 'normal', 'normal', 'normal', 'water', 'normal',
       'normal', 'water', 'bug', 'water', 'normal', 'normal', 'normal',
       'bug', 'normal', 'normal', 'normal', 'water', 'water', 'water',
       'bug', 'bug', 'normal', 'water', 'normal', 'normal', 'bug',
       'normal', 'normal', 'bug', 'normal', 'bug', 'water', 'water',
       'water', 'water', 'normal', 'water', 'normal', 'normal', 'normal',
       'bug', 'normal', 'bug', 'water', 'water', 'normal', 'psychic',
       'normal', 'normal', 'water', 'water', 'water', 'normal', 'water',
       'normal', 'normal', 'normal', 'ice', 'normal', 'normal', 'normal',
       'water', 'normal', 'normal', 'water', 'water', 'normal', 'water',
       'normal', 'bug', 'normal', 'water', 'normal', 'water', 'normal',
       'normal', 'normal', 'normal', 'normal', 'normal', 'bug', 'normal',
       'normal', 'normal', 'normal', 'normal', 'normal', 'bug', 'bug',
       'normal', 'normal', 'normal', 'water', 'water', 'normal', 'bug',
       'bug', 'normal', 'normal', 'normal', 'normal', 'normal', 'bug',
       'bug', 'water', 'water', 'bug', 'bug', 'normal', 'normal',
       'normal', 'water', 'normal', 'water', 'normal', 'normal', 'water',
       'normal', 'normal', 'bug', 'bug', 'normal', 'water', 'normal',
       'bug', 'normal', 'normal', 'bug', 'bug', 'bug', 'water', 'normal',
       'bug', 'ice', 'water', 'bug', 'bug', 'water', 'normal', 'normal',
       'psychic', 'normal', 'water', 'bug', 'bug', 'normal', 'water',
       'normal', 'water', 'normal', 'ice', 'water', 'water', 'normal',
       'normal', 'bug', 'normal', 'normal', 'water', 'normal', 'bug',
       'normal', 'water', 'water', 'normal', 'normal', 'normal', 'ice',
       'bug', 'normal', 'normal', 'water', 'normal', 'bug', 'normal',
       'water', 'normal', 'water', 'bug', 'normal', 'water', 'water',
       'normal', 'water', 'normal', 'bug', 'normal', 'normal', 'water',
```

```
       'water', 'water', 'water', 'water', 'bug', 'bug', 'normal',
       'normal', 'water', 'bug', 'normal', 'normal', 'bug', 'normal',
       'water', 'normal', 'normal', 'normal', 'normal', 'water', 'normal',
       'normal', 'normal', 'normal', 'water', 'water', 'bug', 'water',
       'bug', 'normal', 'normal', 'water', 'normal', 'normal', 'normal',
       'normal', 'bug', 'normal', 'water', 'water', 'water', 'water',
       'water', 'normal', 'normal', 'normal', 'water', 'water', 'water',
       'normal', 'normal', 'water', 'normal', 'normal', 'normal',
       'normal', 'water', 'water', 'normal', 'bug', 'normal', 'water',
       'water', 'normal', 'normal', 'bug', 'normal', 'normal', 'water',
       'normal', 'psychic', 'normal', 'normal', 'water', 'normal',
       'normal', 'normal', 'water', 'water', 'normal', 'normal', 'bug',
       'normal', 'water', 'bug', 'normal', 'bug', 'normal', 'normal',
       'water', 'water', 'psychic', 'normal', 'water', 'normal', 'normal',
       'normal', 'normal', 'bug', 'water', 'normal', 'bug', 'normal',
       'water', 'normal', 'water', 'normal', 'water', 'normal', 'normal',
       'water', 'normal', 'water', 'normal', 'normal'], dtype=object)
```

In [ ]:

```
predictions = dt.predict(pokemon_test[features])
predictions
```

Out[ ]:

```
array(['normal', 'normal', 'water', 'normal', 'water', 'normal', 'bug',
       'water', 'water', 'normal', 'normal', 'normal', 'water', 'bug',
       'normal', 'water', 'ice', 'water', 'water', 'water', 'water',
       'water', 'normal', 'normal', 'water', 'water', 'bug', 'water',
       'water', 'bug', 'normal', 'normal', 'bug', 'water', 'water',
       'normal', 'normal', 'water', 'normal', 'normal', 'normal', 'water',
       'water', 'water', 'bug', 'normal', 'bug', 'water', 'normal',
       'normal', 'normal', 'normal', 'water', 'normal', 'normal', 'bug',
       'water', 'normal', 'normal', 'normal', 'normal', 'normal',
       'normal', 'normal', 'water', 'water', 'normal', 'normal', 'normal',
       'bug', 'normal', 'bug', 'bug', 'normal', 'water', 'normal', 'bug',
       'normal', 'normal', 'water', 'normal', 'bug', 'water', 'ice',
       'normal', 'normal', 'bug', 'normal', 'normal', 'water', 'normal',
       'bug', 'bug', 'water', 'normal', 'water', 'bug', 'normal',
       'normal', 'water', 'normal', 'water', 'bug', 'normal', 'normal',
       'normal', 'bug', 'normal', 'normal', 'normal', 'normal', 'bug',
       'normal', 'bug', 'water', 'bug', 'bug', 'water', 'bug', 'normal',
       'normal', 'water', 'water', 'normal', 'normal', 'water', 'water',
       'bug', 'normal', 'normal', 'normal', 'water', 'water', 'water',
       'normal', 'normal', 'normal', 'bug', 'normal', 'normal', 'water',
       'bug', 'normal', 'normal', 'water', 'normal', 'normal', 'water',
       'normal', 'normal', 'normal', 'normal', 'normal', 'water', 'water',
       'water', 'bug', 'normal', 'water', 'normal', 'bug', 'water',
       'normal', 'normal', 'normal', 'normal', 'normal', 'water',
       'normal', 'water', 'normal', 'psychic', 'water', 'bug', 'bug',
       'water', 'normal', 'normal', 'water', 'water', 'water', 'normal',
       'normal', 'normal', 'water', 'bug', 'normal', 'water', 'bug',
       'normal', 'ice', 'normal', 'bug', 'normal', 'bug', 'normal', 'bug',
       'normal', 'water', 'normal', 'normal', 'normal', 'bug', 'bug',
       'normal', 'normal', 'normal', 'bug', 'normal', 'water', 'water',
       'water', 'bug', 'normal', 'normal', 'water', 'water', 'normal',
       'normal', 'normal', 'bug', 'normal', 'bug', 'water', 'water',
       'normal', 'water', 'normal', 'normal', 'water', 'bug', 'normal',
       'water', 'normal', 'water', 'water', 'water', 'normal', 'normal',
       'normal', 'normal'], dtype=object)
```

In [ ]:

```
predictionsOnTrainset = dt.predict(pokemon_train[features])
predictionsOnTestset = dt.predict(pokemon_test[features])

accuracyTrain = calculate_accuracy(predictionsOnTrainset, pokemon_train.type1)
```

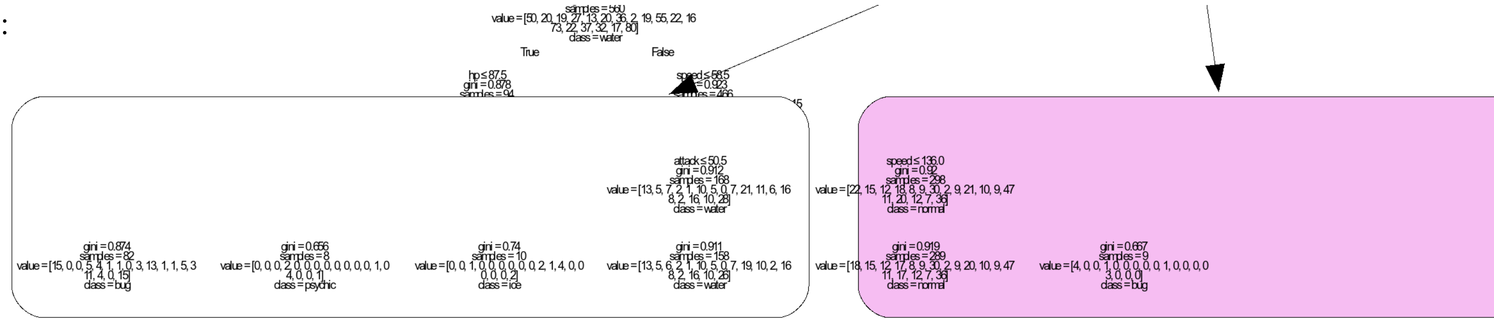
```
accuracyTest = calculate_accuracy(predictionsOnTestset, pokemon_test.type1)

print("Accuracy on training set " + str(accuracyTrain))
print("Accuracy on test set " + str(accuracyTest))
```

```
Accuracy on training set 0.18571428571428572
Accuracy on test set 0.14107883817427386
```

In [ ]:

```
plot_tree_classification(dt, features, np.sort(pokemon.type1.unique()))
```

Out[ ]:



Portfolio assignment 17

30 min: Train a decision tree to predict the body_mass_g of a penguin based on their characteristics.

- Split the penguin dataset into a train (70%) and test (30%) set.
- Use the train set to fit a DecisionTreeRegressor. You are free to to choose which columns you want to use as feature variables and you are also free to choose the max_depth of the tree. **Note**: Some machine learning algorithms can not handle missing values. You will either need to
  - replace missing values (with the mean or most popular value). For replacing missing values you can use .fillna(\<value>) https://pandas.pydata.org/docs/reference/api/pandas.Series.fillna.html
  - remove rows with missing data. You can remove rows with missing data with .dropna() https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html
- Use your decision tree model to make predictions for both the train and test set.
- Calculate the RMSE for both the train set predictions and test set predictions.
- Is the RMSE different? Did you expect this difference?
- Use the plot_tree_regression function above to create a plot of the decision tree. Take a few minutes to analyse the decision tree. Do you understand the tree?

In [ ]:

```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
```

In [ ]:

```
penguins = sns.load_dataset("penguins")
penguins.fillna(penguins.mean(), inplace=True)
penguins = penguins[penguins['sex'].notna()]

penguins_train, penguins_test = train_test_split(penguins, test_size=0.3, random_state=42) #

penguins.head()
```

```
C:\Users\Jens\AppData\Local\Temp\ipykernel_9500\145872236.py:2: FutureWarning: Dropping of nui
  penguins.fillna(penguins.mean(), inplace=True)
```

Out[ ]:

	species	island	bill_length_mm	bill_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
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
5	Adelie	Torgersen	39.3	20.6	190.0	3650.0	Male

In [ ]:

```
penguins.corr()
```

Out[ ]:

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
bill_length_mm	1.000000	-0.228626	0.653096	0.589451
bill_depth_mm	-0.228626	1.000000	-0.577792	-0.472016
flipper_length_mm	0.653096	-0.577792	1.000000	0.872979
body_mass_g	0.589451	-0.472016	0.872979	1.000000

In [ ]:

```
features= ['flipper_length_mm', 'bill_length_mm']
dt_regression = DecisionTreeRegressor(max_depth = 3) # Increase max_depth to see effect in th
dt_regression.fit(penguins_train[features], penguins_train['body_mass_g'])
```

Out[ ]:

```
▸ DecisionTreeRegressor
DecisionTreeRegressor(max_depth=3)
```

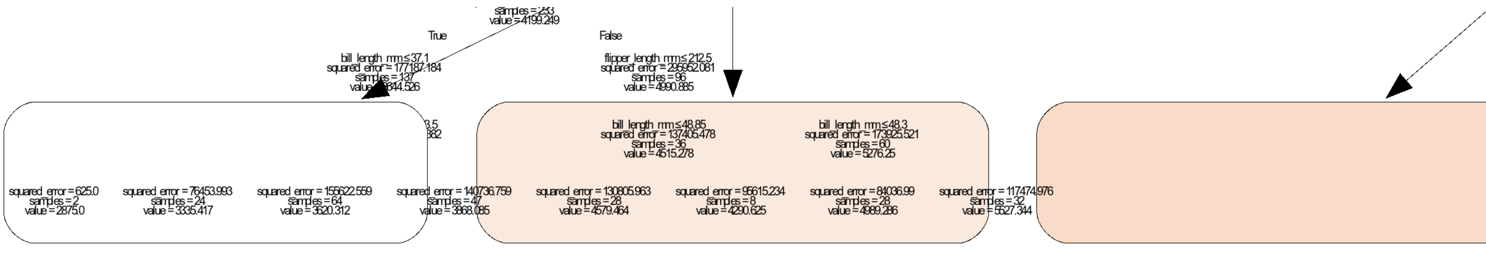
In [ ]:

```python
from sklearn import tree
import graphviz

def plot_tree_regression(model, features):
    # Generate plot data
    dot_data = tree.export_graphviz(model, out_file=None,
                          feature_names=features,
                          filled=True, rounded=True,
                          special_characters=True)

    # Turn into graph using graphviz
    graph = graphviz.Source(dot_data)

    # Write out a pdf
    graph.render("decision_tree")

    # Display in the notebook
    return graph
```

In [ ]:

```python
plot_tree_regression(dt_regression, features)
```

Out[ ]:



In [ ]:

```python
def calculate_rmse(predictions, actuals):
    if(len(predictions) != len(actuals)):
        raise Exception("The amount of predictions did not equal the amount of actuals")

    return (((predictions - actuals) ** 2).sum() / len(actuals)) ** (1/2)
```

In [ ]:

```
predictionsOnTrainset = dt_regression.predict(penguins_train[features])
predictionsOnTestset = dt_regression.predict(penguins_test[features])

rmseTrain = calculate_rmse(predictionsOnTrainset, penguins_train.body_mass_g)
rmseTest = calculate_rmse(predictionsOnTestset, penguins_test.body_mass_g)

print("RMSE on training set " + str(rmseTrain))
print("RMSE on test set " + str(rmseTest))
```

```
RMSE on training set 352.49168872939606
RMSE on test set 360.7449386097482
```

Portfolio assignment 18

30 min: Train a decision tree to predict one of the numerical columns of your own dataset.

- Split your dataset into a train (70%) and test (30%) set.
- Use the train set to fit a DecisionTreeRegressor. You are free to to choose which columns you want to use as feature variables and you are also free to choose the max_depth of the tree.
- Use your decision tree model to make predictions for both the train and test set.
- Calculate the RMSE for both the train set predictions and test set predictions.
- Is the accurracy different? Did you expect this difference?
- Use the plot_tree function above to create a plot of the decision tree. Take a few minutes to analyse the decision tree. Do you understand the tree?

In [ ]:

```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
```

In [ ]:

```
pokemon = pd.read_csv("../pokemon.csv", sep=",")
pokemon.fillna(pokemon.mean(), inplace=True)
penguins = pokemon[pokemon['type1'].notna()]

pokemon_train, pokemon_test = train_test_split(pokemon, test_size=0.3, random_state=42)

pokemon.head()
```

```
C:\Users\Jens\AppData\Local\Temp\ipykernel_20092\1556286363.py:2: FutureWarning: Dropping of n
  pokemon.fillna(pokemon.mean(), inplace=True)
```

Out[ ]:

	abilities	against_bug	against_dark	against_dragon	against_electric	against_fairy	against_fight	agai
0	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	2.0
1	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	2.0
2	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5	2.0
3	['Blaze', 'Solar Power']	0.5	1.0	1.0	1.0	0.5	1.0	0.5
4	['Blaze', 'Solar Power']	0.5	1.0	1.0	1.0	0.5	1.0	0.5

5 rows × 41 columns

In [ ]:

```
corr = pokemon.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Out[ ]:

	against_bug	against_dark	against_dragon	against_electric	against_fairy	against_fight	aga
against_bug	1.000000	0.230107	0.165430	-0.246943	0.239566	0.137902	0.2
against_dark	0.230107	1.000000	0.140830	-0.015830	-0.301354	-0.357981	0.0
against_dragon	0.165430	0.140830	1.000000	-0.108928	0.439705	0.035237	-0.
against_electric	-0.246943	-0.015830	-0.108928	1.000000	-0.089864	-0.102798	-0.
against_fairy	0.239566	-0.301354	0.439705	-0.089864	1.000000	0.157712	-0.
against_fight	0.137902	-0.357981	0.035237	-0.102798	0.157712	1.000000	-0.0
against_fire	0.202778	0.010527	-0.261570	-0.279029	-0.169489	-0.076480	1.0
against_flying	0.183343	-0.179697	0.064850	-0.111461	0.199862	-0.318941	0.5
against_ghost	0.129174	0.672337	-0.049941	-0.073031	-0.120806	-0.546982	0.0
against_grass	0.079197	-0.006533	-0.037135	0.056209	0.052899	0.269157	-0.
against_ground	-0.186841	-0.007660	-0.120042	-0.269444	-0.256504	0.358793	-0.
against_ice	0.148176	-0.010763	0.350048	-0.328531	0.273650	-0.220239	0.1
against_normal	0.215589	-0.413632	0.142035	0.076699	0.149488	-0.006997	-0.0
against_poison	0.354255	-0.236919	-0.210199	-0.015769	0.146464	-0.189798	0.1
against_psychic	-0.463272	-0.230415	0.100153	-0.017592	-0.145238	-0.264938	-0.0
against_rock	-0.210522	0.011963	0.090184	0.417261	-0.205444	-0.240964	0.1
against_steel	0.055504	-0.119758	-0.227697	-0.187543	0.130323	0.165066	0.1
against_water	-0.254732	-0.001976	-0.096549	-0.297600	-0.218937	0.205249	-0.
attack	-0.054175	-0.098849	0.138217	-0.104276	0.207526	0.149123	-0.0
base_egg_steps	0.062133	0.187220	0.164773	-0.061970	0.120594	-0.006359	-0.0
base_happiness	0.009994	0.024155	-0.151915	0.030411	-0.209323	-0.088722	0.0
base_total	-0.012398	0.065446	0.069766	-0.017137	0.098948	0.048629	-0.0
defense	-0.036474	0.048039	-0.023794	-0.072433	0.001655	0.150424	0.0
experience_growth	0.035717	-0.008391	0.172547	-0.041584	0.146370	0.010407	-0.0
height_m	-0.059781	0.018608	0.164448	0.003022	0.114993	0.058524	-0.0
hp	0.034897	0.010589	0.089721	-0.035354	0.129284	0.109425	-0.0
percentage_male	-0.044982	-0.079434	0.055214	0.049106	0.009831	0.045678	-0.
pokedex_number	0.004618	0.009066	0.000872	-0.068552	0.176651	0.018296	0.0
sp_attack	0.055352	0.170849	0.039739	0.022305	-0.010296	-0.118481	-0.0
sp_defense	-0.002342	0.132507	-0.047416	0.019193	0.002754	-0.044460	-0.0
speed	-0.043802	-0.000326	0.078123	0.111422	0.065401	-0.050495	-0.0
weight_kg	-0.031344	0.037634	0.125991	-0.101403	0.098210	0.159761	-0.0
generation	-0.001549	-0.016013	-0.025201	-0.063180	0.150801	0.000681	0.0
is_legendary	0.027864	0.136315	0.014844	-0.023151	0.050165	-0.059132	-0.0

In [ ]:

```
features= ['height_m', 'hp', 'base_total']
dt_regression = DecisionTreeRegressor(max_depth = 3) # Increase max_depth to see effect in th
dt_regression.fit(pokemon_train[features], pokemon_train['speed'])
```

Out[ ]:

```
▸ DecisionTreeRegressor
DecisionTreeRegressor(max_depth=3)
```

In [ ]:

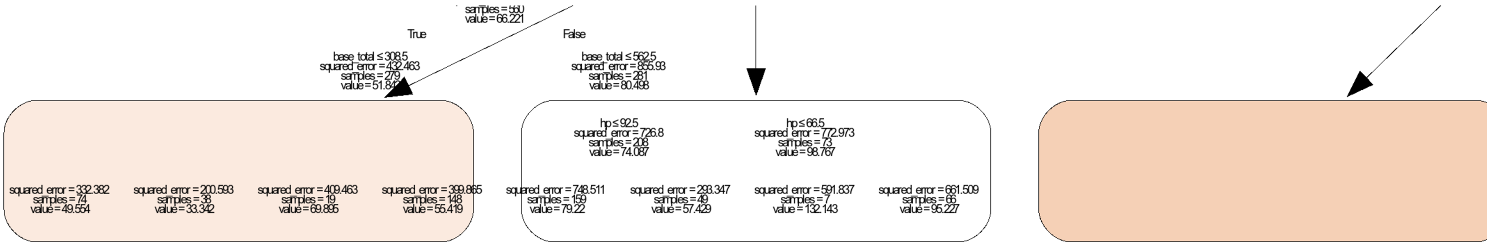
```
from sklearn import tree
import graphviz

def plot_tree_regression(model, features):
    # Generate plot data
    dot_data = tree.export_graphviz(model, out_file=None,
                          feature_names=features,
                          filled=True, rounded=True,
                          special_characters=True)
```

```python
    # Turn into graph using graphviz
    graph = graphviz.Source(dot_data)

    # Write out a pdf
    graph.render("decision_tree")

    # Display in the notebook
    return graph
```

In [ ]:

```python
plot_tree_regression(dt_regression, features)
```

Out[ ]:



In [ ]:

```python
def calculate_rmse(predictions, actuals):
    if(len(predictions) != len(actuals)):
        raise Exception("The amount of predictions did not equal the amount of actuals")

    return (((predictions - actuals) ** 2).sum() / len(actuals)) ** (1/2)
```

In [ ]:

```python
predictionsOnTrainset = dt_regression.predict(pokemon_train[features])
predictionsOnTestset = dt_regression.predict(pokemon_test[features])

rmseTrain = calculate_rmse(predictionsOnTrainset, pokemon_train.speed)
rmseTest = calculate_rmse(predictionsOnTestset, pokemon_test.speed)

print("RMSE on training set " + str(rmseTrain))
print("RMSE on test set " + str(rmseTest))
```

```
RMSE on training set 22.37538493545603
RMSE on test set 24.882053058123287
```

Portfolio assignment 19

30 min: Create a cluster model on the penguins dataset.

- Use the pairplot() function on the penguins dataset. Do you visually notice any clusters? How many clusters do you think there are?
- Use the KMeans algorithm to create a cluster model. Apply this model to the dataset to create an extra column 'cluster' just like we did for the iris dataset above.
  Note: Some machine learning algorithms can not handle missing values. You will either need to
  - replace missing values (with the mean or most popular value). For replacing missing values you can use .fillna(\<value>) https://pandas.pydata.org/docs/reference/api/pandas.Series.fillna.html
  - remove rows with missing data. You can remove rows with missing data with .dropna() https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html
- Calculate the Silhouette Coefficient for your clustering. Play around with the features and n_clusters to search for better results. Keep the cluster model with the highest Silhouette Coefficient.
- Use the pairplot(hue='cluster') function to observe how the model has clustered the data.
- We know the species of each penguin. Use a contingency table to reveal the relation between the cluster results and the species. Is there an exact match? Are there species which ended up in the same cluster? If so, what does it mean that they ended up in the same cluster?

In [ ]:

```
import pandas as pd
import seaborn as sns
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.metrics import pairwise_distances
```

In [ ]:

```
penguins = sns.load_dataset("penguins")
penguins.fillna(penguins.mean(), inplace=True)
penguins.head()
```

```
C:\Users\Jens\AppData\Local\Temp\ipykernel_8156\4094850315.py:2: FutureWarning: Dropping of nu
  penguins.fillna(penguins.mean(), inplace=True)
```

Out[ ]:

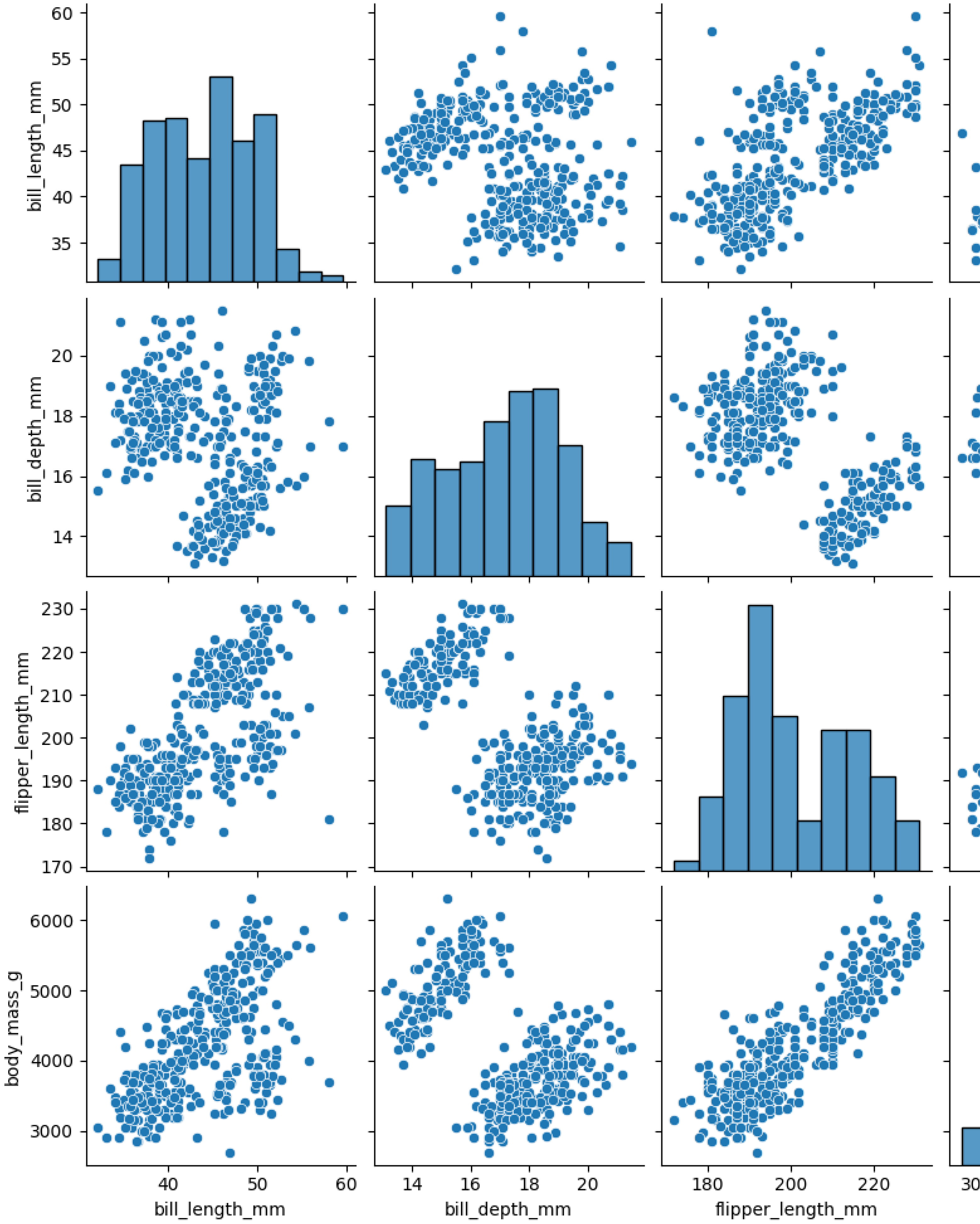
	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.10000	18.70000	181.000000	3750.000000	Male
1	Adelie	Torgersen	39.50000	17.40000	186.000000	3800.000000	Female
2	Adelie	Torgersen	40.30000	18.00000	195.000000	3250.000000	Female
3	Adelie	Torgersen	43.92193	17.15117	200.915205	4201.754386	NaN
4	Adelie	Torgersen	36.70000	19.30000	193.000000	3450.000000	Female

In [ ]:

```
sns.pairplot(penguins)
```

Out[ ]:

```
<seaborn.axisgrid.PairGrid at 0x228646e4220>
```



Ik zie 29 clusters.

In [ ]:

```
features = ['bill_length_mm','bill_depth_mm','flipper_length_mm', 'body_mass_g']

# cluster : coefficient
```

```
# 2 : 0.6270788983213472
# 3 : 0.5746583550492242
# 4 : 0.5536647391613351
# 5 : 0.5455535572866389

km = KMeans(n_clusters=2, random_state=80).fit(penguins[features])

metrics.silhouette_score(penguins[features], km.labels_, metric='euclidean')
```

```
C:\Users\Jens\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalC
  warnings.warn(
```

Out[ ]:

```
0.6270788983213472
```

In [ ]:

```
penguins['cluster'] = km.predict(penguins[features])
```

In [ ]:

```
penguins.head()
```

Out[ ]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	cluster
0	Adelie	Torgersen	39.10000	18.70000	181.000000	3750.000000	Male	0
1	Adelie	Torgersen	39.50000	17.40000	186.000000	3800.000000	Female	0
2	Adelie	Torgersen	40.30000	18.00000	195.000000	3250.000000	Female	0
3	Adelie	Torgersen	43.92193	17.15117	200.915205	4201.754386	NaN	0
4	Adelie	Torgersen	36.70000	19.30000	193.000000	3450.000000	Female	0

In [ ]:

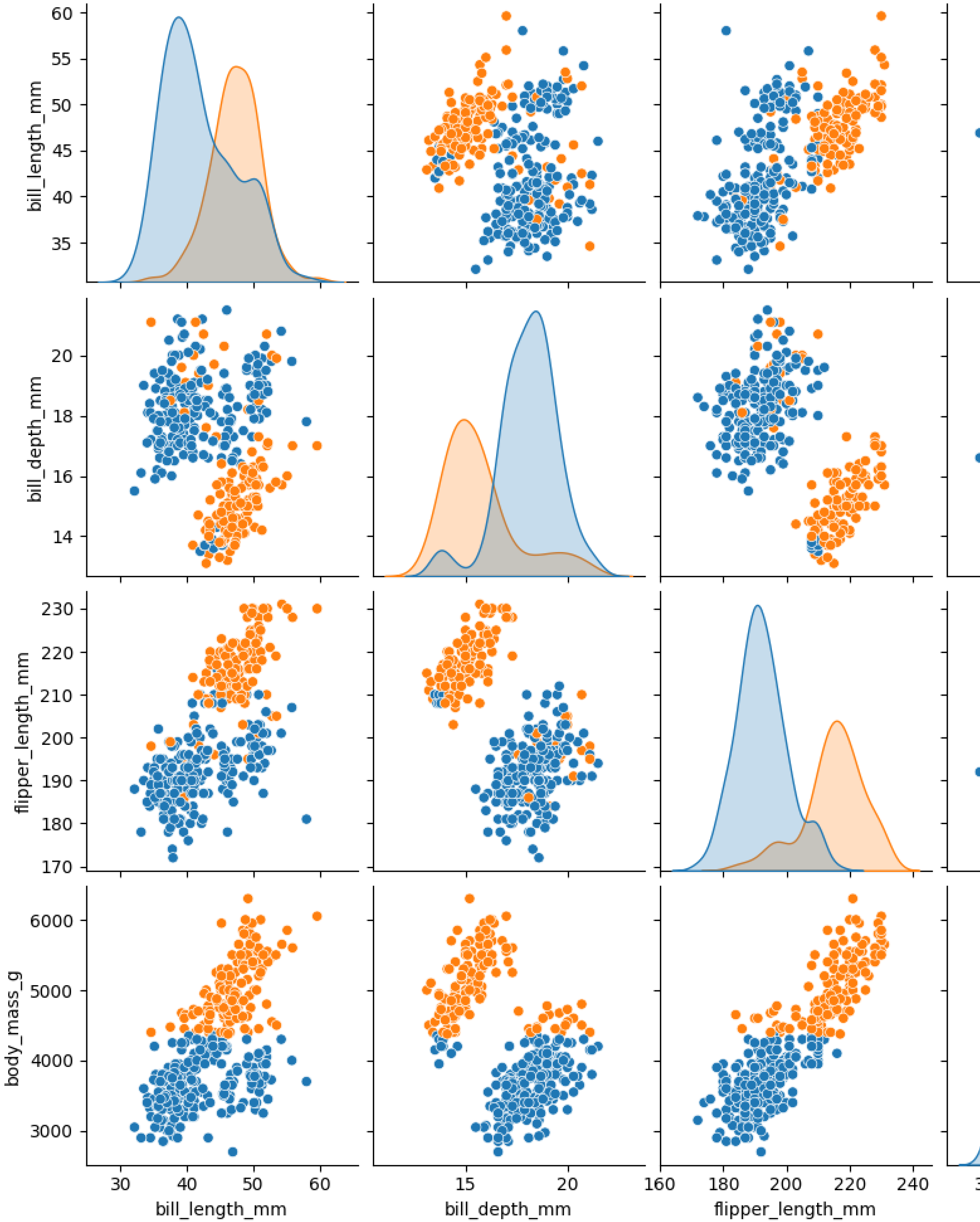
```
penguins.cluster.value_counts()
```

Out[ ]:

```
0    211
1    133
Name: cluster, dtype: int64
```

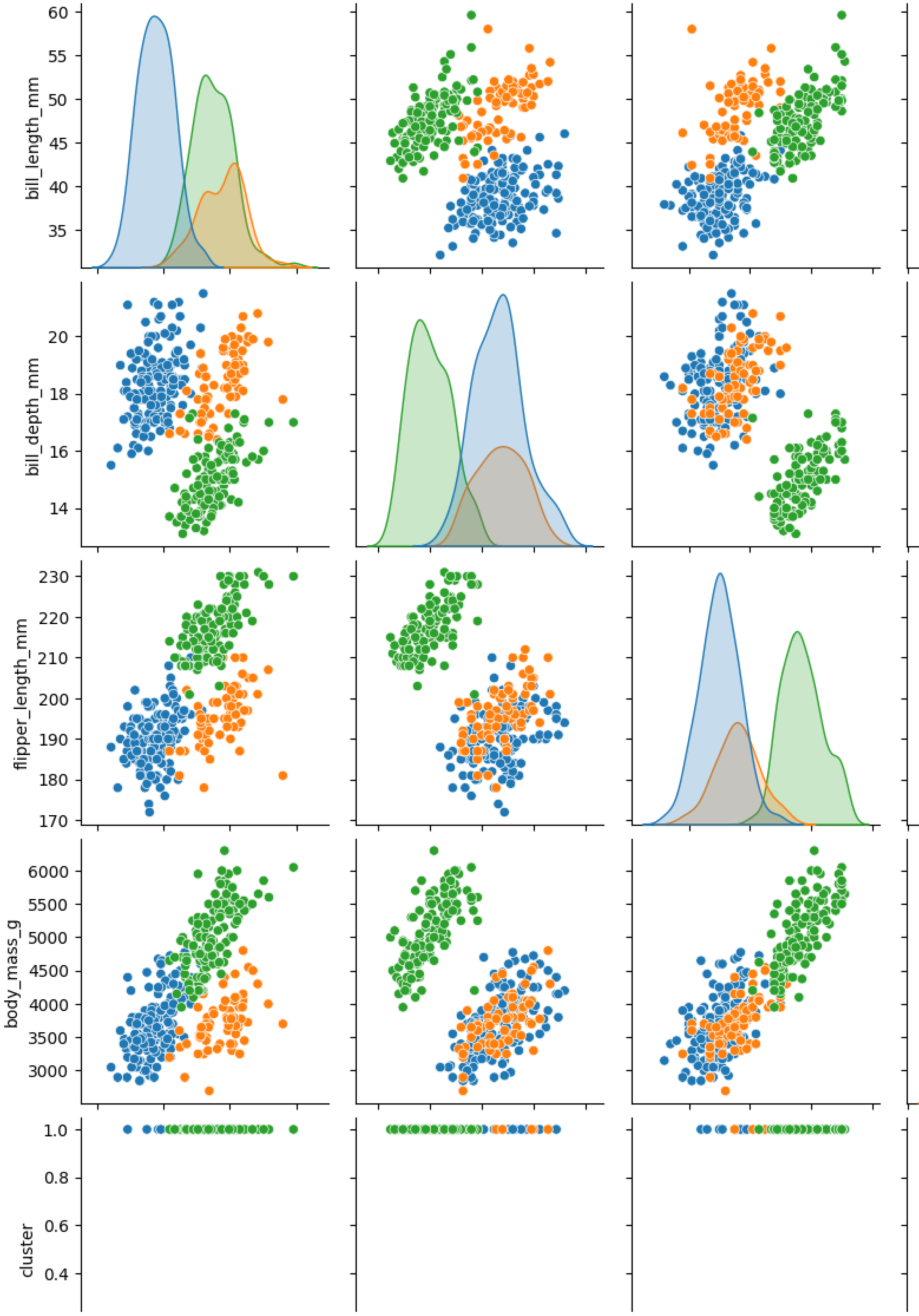
In [ ]:

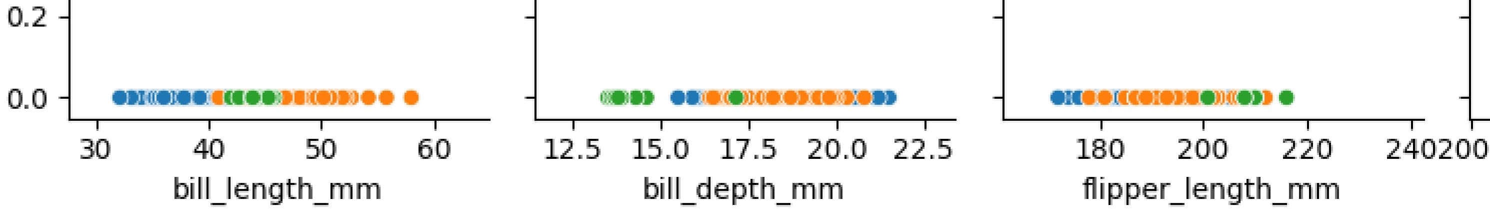
```
sns.pairplot(penguins, hue="cluster")
plt.show()
```



In [ ]:

```
sns.pairplot(penguins, hue="species")
plt.show()
```


bill_length_mm
bill_depth_mm
flipper_length_mm
body_mass_g
cluster



In [ ]:

```
from scipy.stats import chi2_contingency

def create_contingency_table(dataset, column1, column2):
    return dataset.groupby([column1, column2]).size().unstack(column1, fill_value=0)

def check_correlation(dataset, column1, column2):
    contingency_table = create_contingency_table(dataset, column1, column2)
    chi2 = chi2_contingency(contingency_table)
    p_value = chi2[1]
    odds_of_correlation = 1 - p_value
    print(f"The odds of a correlation between {column1} and {column2} is {odds_of_correlatio
    print("This percentage needs to be at least 95% for a significant correlation.")
```

In [ ]:

```
penguinsContingencyTable = create_contingency_table(penguins, 'species','cluster')

penguinsContingencyTable
```

Out[ ]:

species	Adelie	Chinstrap	Gentoo
cluster			
0	138	63	10
1	14	5	114

In [ ]:

```
penguinsContingencyTable.plot(kind='bar')
```

Out[ ]:

```
<AxesSubplot: xlabel='cluster'>
```