Case study definition

Here is our pengouins database:

In [1]:
```
import pandas as pd
df = pd.read_csv(r'../3__matplotlib/data.csv')
df.head(5)                # show only the first five rows
```

Out[1]:

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

Let's search for a linear dependance of the pengouin mass (`'body_mass_g'` , M) in the following variables:

- `'bill_length_mm'` , L
- `'bill_depth_mm'` , D
- `flipper_length_mm'` , F

The linear expression is the following:

$$M = \alpha_1 L + \alpha_2 D + \alpha_3 F + \beta$$

In [2]:
```
M = 'body_mass_g'
L = 'bill_length_mm'
D = 'bill_depth_mm'
F = 'flipper_length_mm'
```

Code

The `scikit-learn` package includes a `LinearRegression` class to solve this problem;

Removing `nan` values

In [3]:

```
cond = df[[M, L, D, F]].isna().any(axis=1)   # every row where
                                             # at least one value is nan
df = df[~cond]                               # keep the other rows,
                                             # '~' does the negation of `cond`
```

Performing the regression

A `LinearRegression` instance is first created. Then it's `fit` method is called directly using the columns of the `DataFrame:

In [4]:
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
_ = model.fit(df[[L, D, F]], df[M])
```

Results are **stored in the `model` instance**. Let's read the coefficients:

In [5]:
```
alpha_1, alpha_2, alpha_3 = model.coef_.round(1)
beta = model.intercept_.round(1)
print(alpha_1, alpha_2, alpha_3, beta)
```

```
4.2 20.0 50.3 -6424.8
```

The regression quality (determination coefficient) is given by `score`:

In [6]:
```
model.score(df[[L, D, F]], df[M])
```

Out[6]:
```
0.7614704841272493
```

Using as a predictor

Using the fit coefficient, one can now go the reverse way: define the mass M from the 3 other variables (L, D, F). This is done using `predict`:

In [7]:
```
import numpy as np
L_predict = [40, 45]
D_predict = [15, 20]
F_predict = [200, 275]

# `T` takes the transpose of the array, because data must be column-wise
to_predict = np.array([L_predict, D_predict, F_predict]).T
model.predict(to_predict)
```

```
/home/nerotb/Python/3.12/lib/python3.12/site-packages/sklearn/base.py:493: U
serWarning: X does not have valid feature names, but LinearRegression was fi
tted with feature names
  warnings.warn(
```

Out[7]:
```
array([4096.29544534, 7987.54383621])
```