A tale of two variables

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

Swedish motor insurance data

- Each row represents one geographic region in Sweden.
- There are 63 rows.

n_claims	total_payment_sek
108	392.5
19	46.2
13	15.7
124	422.2
40	119.4
...	...

Descriptive statistics

```
import pandas as pd
print(swedish_motor_insurance.mean())
```

```
n_claims             22.904762
total_payment_sek    98.187302
dtype: float64
```

```
print(swedish_motor_insurance['n_claims'].corr(swedish_motor_insurance['total_payment_sek']))
```

```
0.9128782350234068
```

What is regression?

- Statistical models to explore the relationship a response variable and some explanatory variables.
- Given values of explanatory variables, you can predict the values of the response variable.

n_claims	total_payment_sek
108	3925
19	462
13	157
124	4222
40	1194
200	**???**

Jargon

Response variable (a.k.a. dependent variable)

The variable that you want to predict.

Explanatory variables (a.k.a. independent variables)

The variables that explain how the response variable will change.

Linear regression and logistic regression

Linear regression

- The response variable is numeric.

Logistic regression

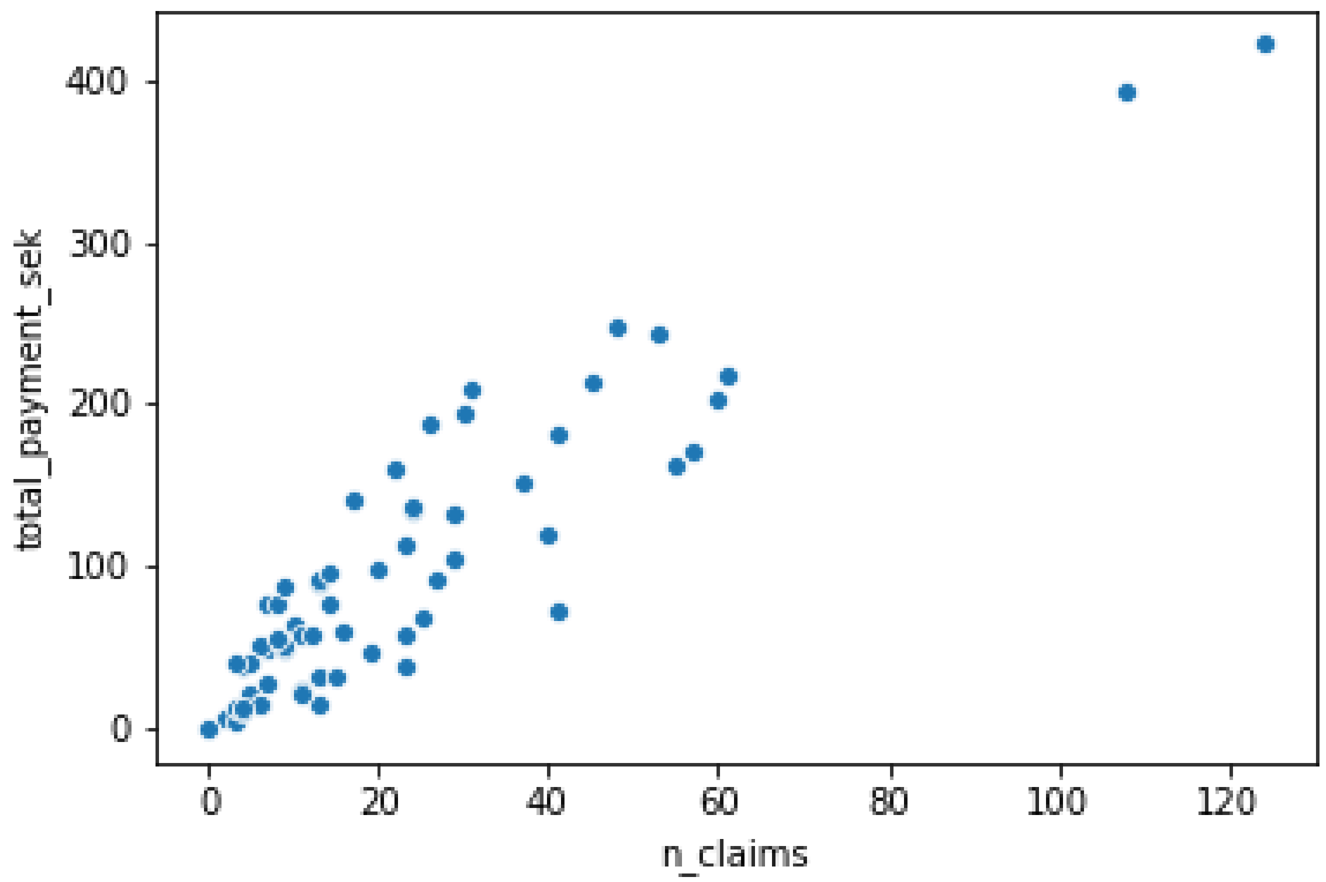
- The response variable is logical.

Simple linear/logistic regression

- There is only one explanatory variable.

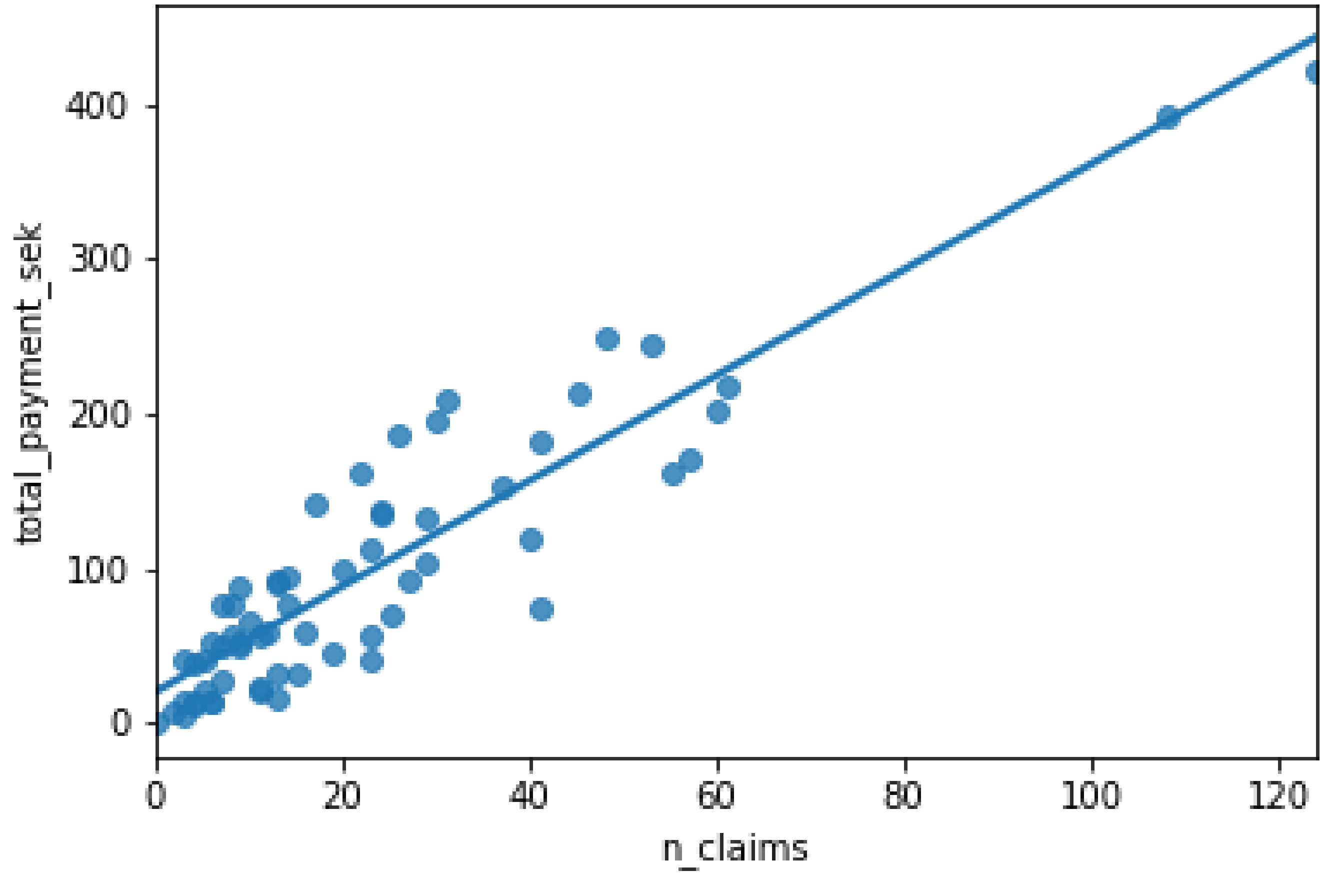
Visualizing pairs of variables

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

sns.scatterplot(x="n_claims",
                y="total_payment_sek",
                data=swedish_motor_insurance)

plt.show()
```



Adding a linear trend line

```
sns.regplot(x="n_claims",
            y="total_payment_sek",
            data=swedish_motor_insurance,
            ci=None)
```



Course flow

Chapter 1

Visualizing and fitting linear regression models.

Chapter 2

Making predictions from linear regression models and understanding model coefficients.

Chapter 3

Assessing the quality of the linear regression model.

Chapter 4

Same again, but with logistic regression models

Python packages for regression

`statsmodels`

- Optimized for insight (focus in this course)

`scikit-learn`

- Optimized for prediction (focus in other DataCamp courses)

Let's practice!

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Fitting a linear regression

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

Straight lines are defined by two things

Intercept

The y value at the point when x is zero.

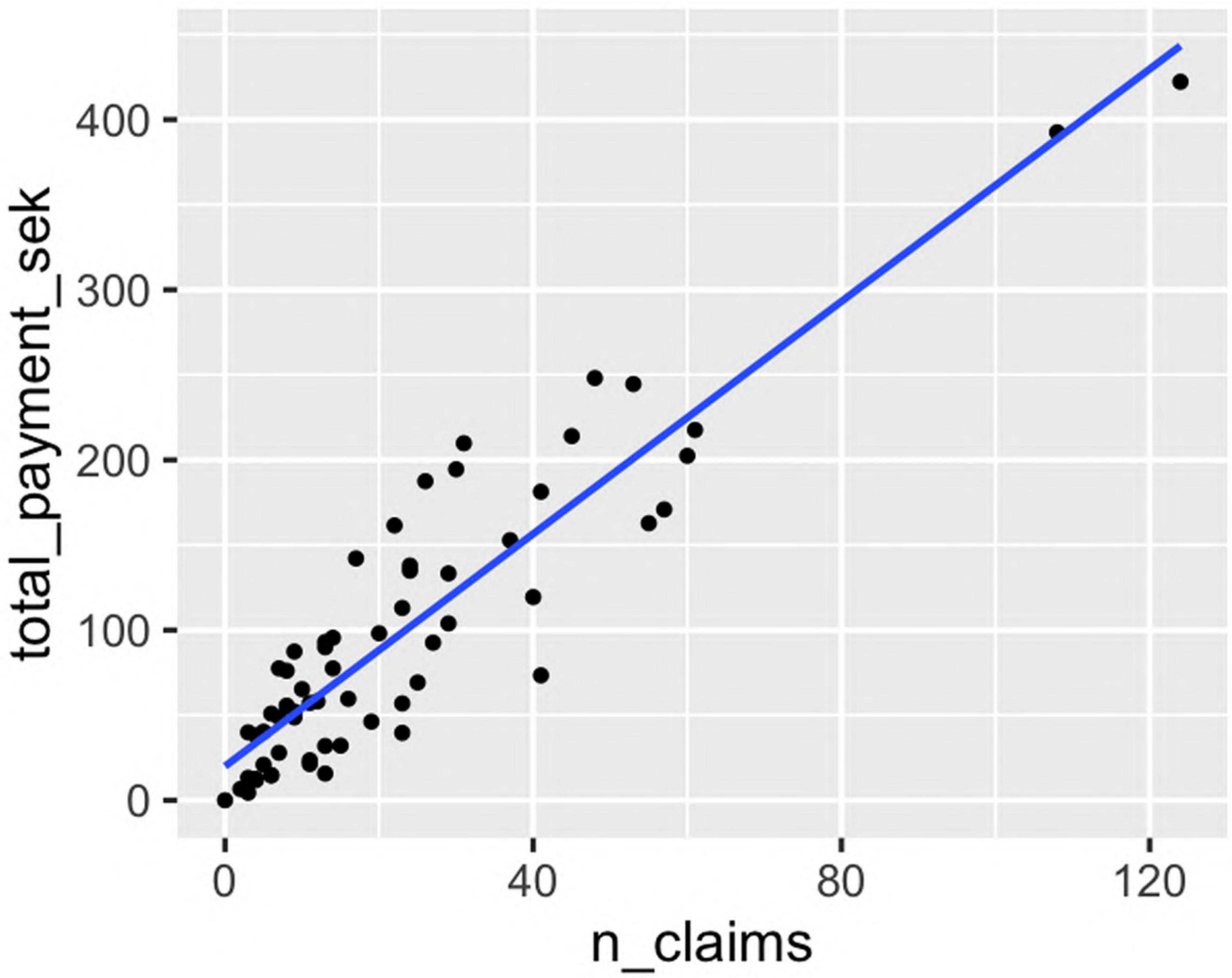
Slope

The amount the y value increases if you increase x by one.

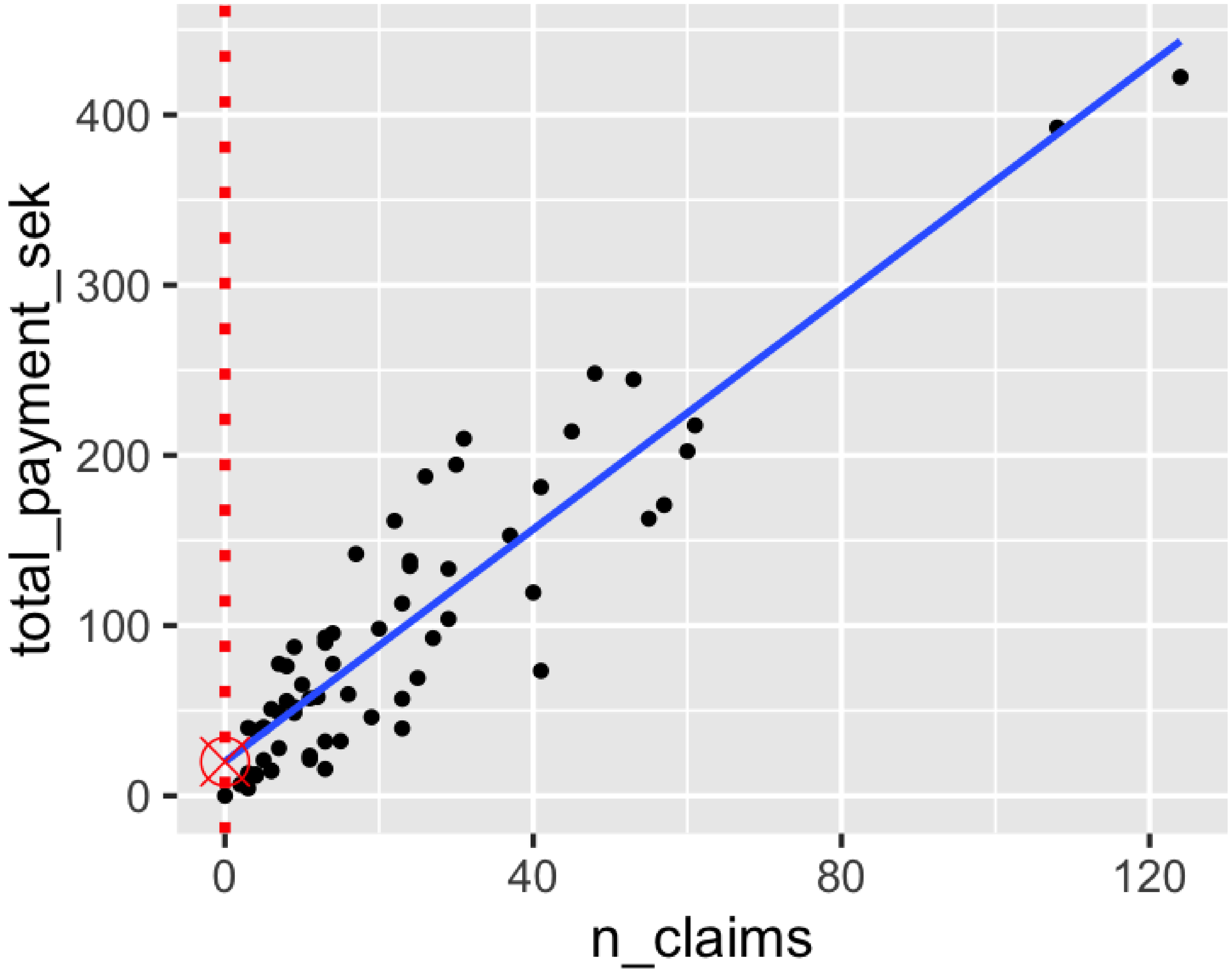
Equation

$y = \text{intercept} + \text{slope} * x$

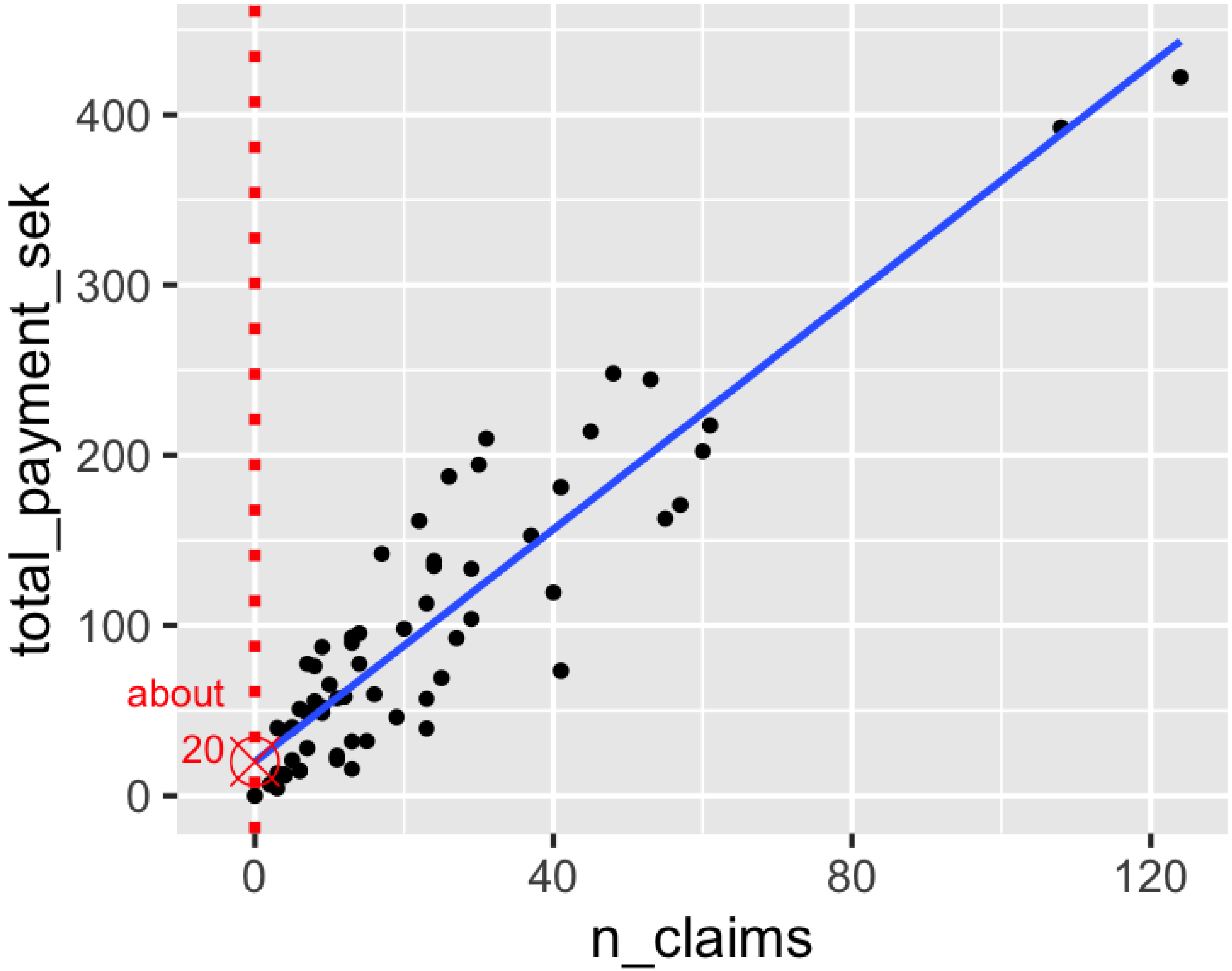
Estimating the intercept



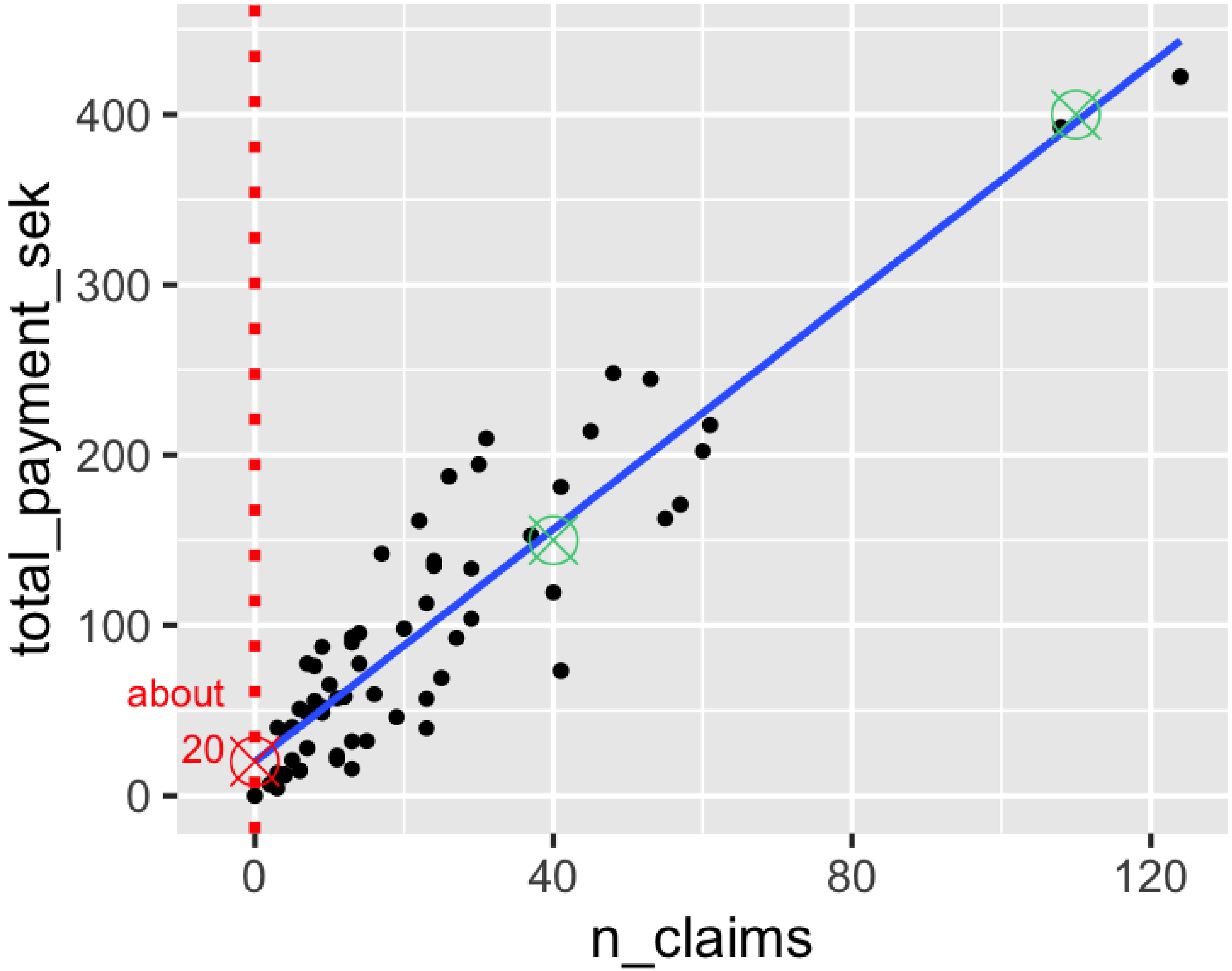
Estimating the intercept



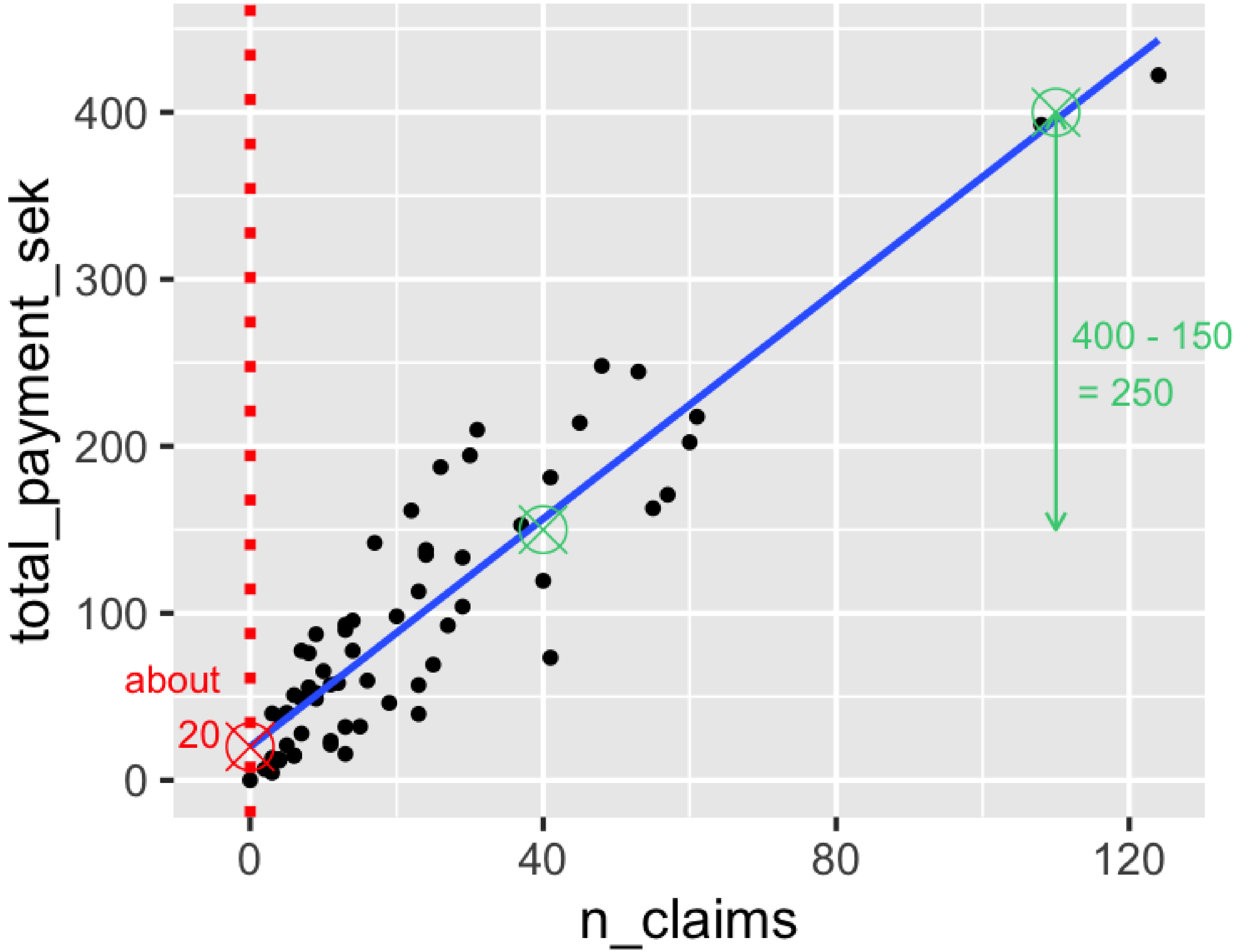
Estimating the intercept



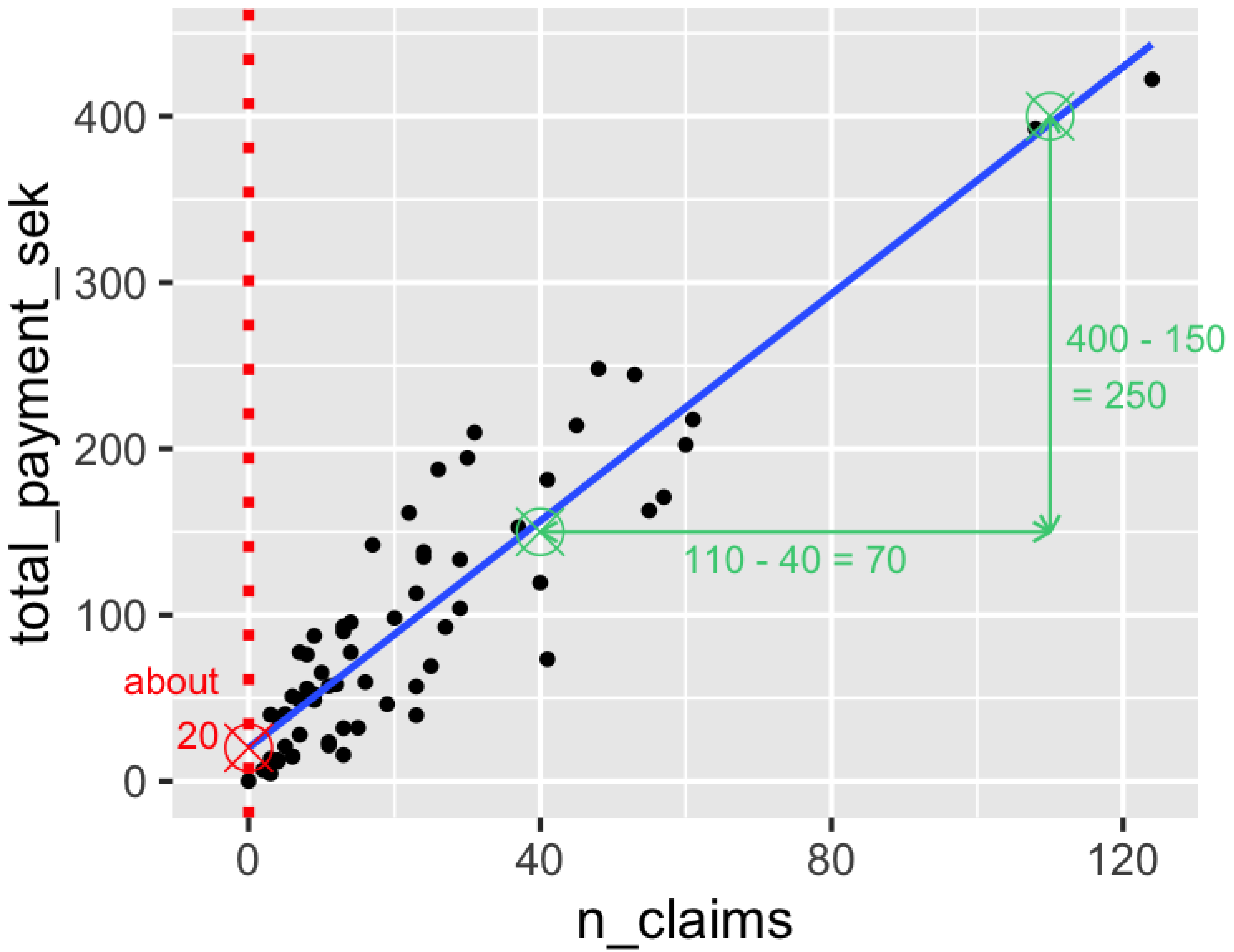
Estimating the slope



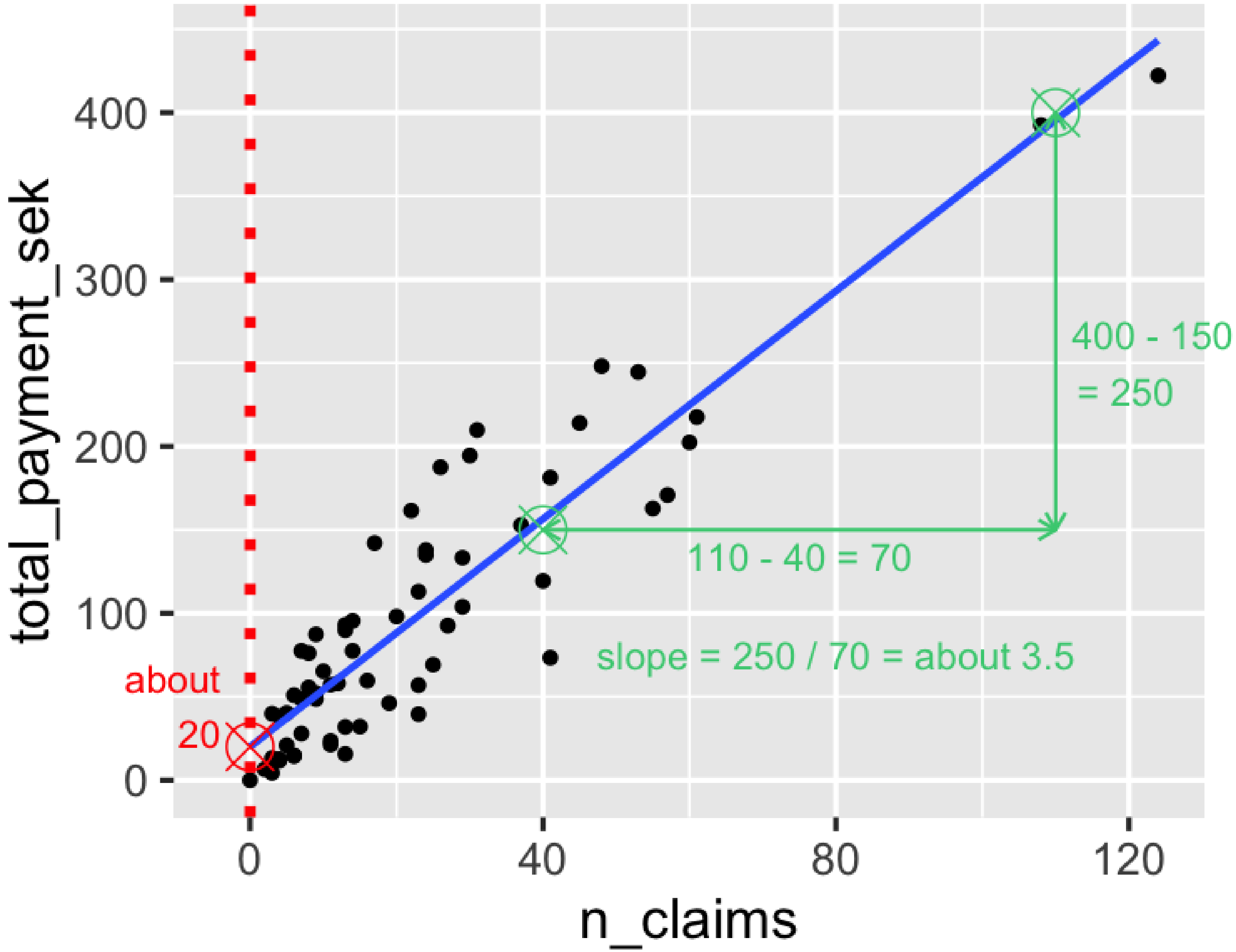
Estimating the slope



Estimating the slope



Estimating the slope



Running a model

```
from statsmodels.formula.api import ols
mdl_payment_vs_claims = ols("total_payment_sek ~ n_claims",
                            data=swedish_motor_insurance)

mdl_payment_vs_claims = mdl_payment_vs_claims.fit()
print(mdl_payment_vs_claims.params)
```

```
Intercept    19.994486
n_claims      3.413824
dtype: float64
```

Interpreting the model coefficients

```
Intercept    19.994486
n_claims      3.413824
dtype: float64
```

Equation

$$\text{total\_payment\_sek} = 19.99 + 3.41 * \text{n\_claims}$$

Let's practice!

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Categorical explanatory variables

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

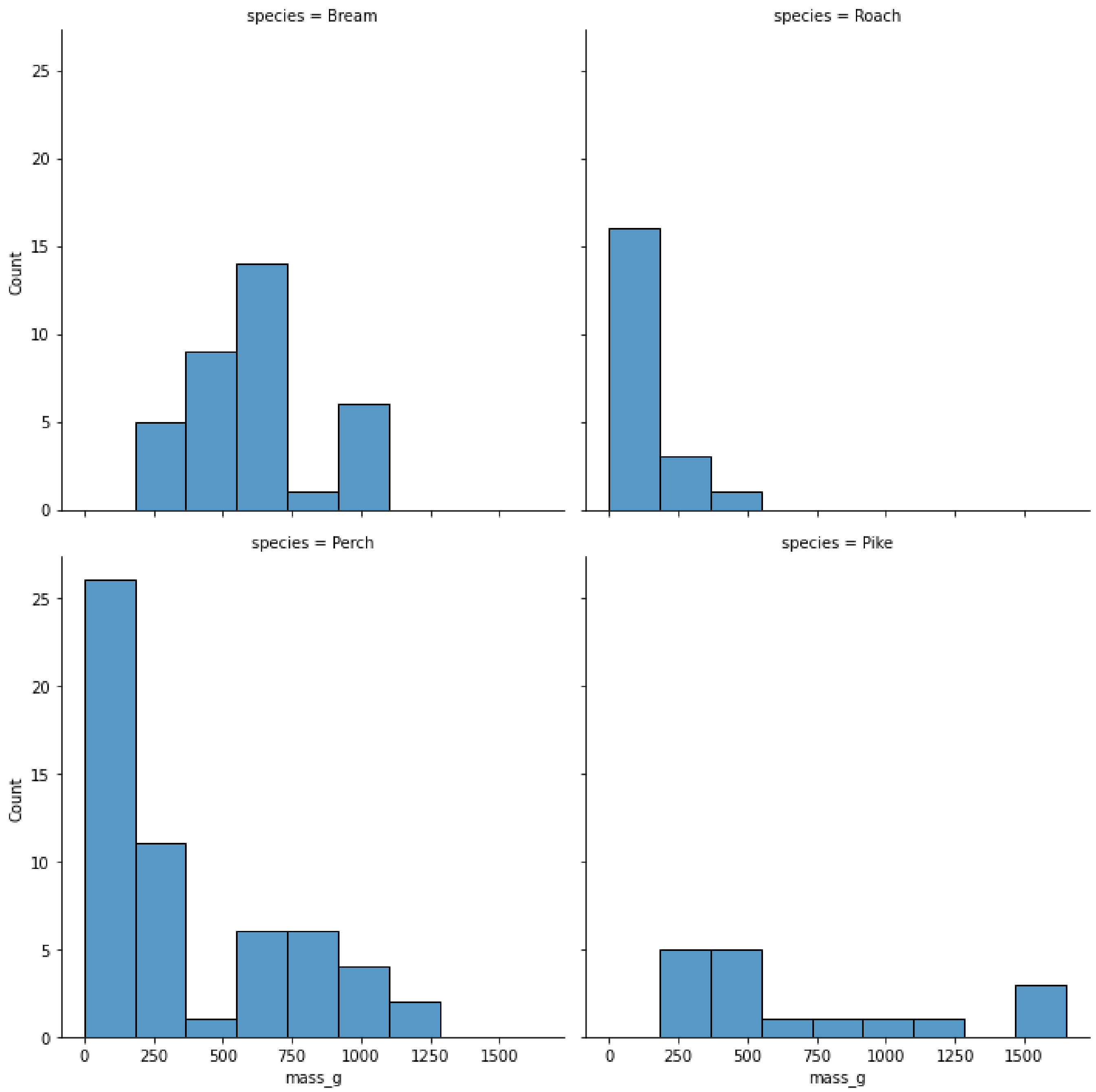
Fish dataset

- Each row represents one fish.
- There are 128 rows in the dataset.
- There are 4 species of fish:
  - Common Bream
  - European Perch
  - Northern Pike
  - Common Roach

species	mass_g
Bream	242.0
Perch	5.9
Pike	200.0
Roach	40.0
...	...

Visualizing 1 numeric and 1 categorical variable

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

sns.displot(data=fish,
            x="mass_g",
            col="species",
            col_wrap=2,
            bins=9)

plt.show()
```



Summary statistics: mean mass by species

```
summary_stats = fish.groupby("species")["mass_g"].mean()
print(summary_stats)
```

```
species
Bream    617.828571
Perch    382.239286
Pike     718.705882
Roach    152.050000
Name: mass_g, dtype: float64
```

Linear regression

```python
from statsmodels.formula.api import ols
mdl_mass_vs_species = ols("mass_g ~ species", data=fish).fit()
print(mdl_mass_vs_species.params)
```

```
Intercept            617.828571
species[T.Perch]    -235.589286
species[T.Pike]      100.877311
species[T.Roach]    -465.778571
```

Model with or without an intercept

From previous slide, model with intercept

```
mdl_mass_vs_species = ols(
  "mass_g ~ species", data=fish).fit()
print(mdl_mass_vs_species.params)
```

```
Intercept            617.828571
species[T.Perch]    -235.589286
species[T.Pike]      100.877311
species[T.Roach]    -465.778571
```

The coefficients are relative to the intercept: $617.83 - 235.59 = 382.24!$

Model without an intercept

```
mdl_mass_vs_species = ols(
  "mass_g ~ species + 0", data=fish).fit()
print(mdl_mass_vs_species.params)
```

```
species[Bream]    617.828571
species[Perch]    382.239286
species[Pike]     718.705882
species[Roach]    152.050000
```

In case of a single, categorical variable, coefficients are the means.

Let's practice!

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Making predictions

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

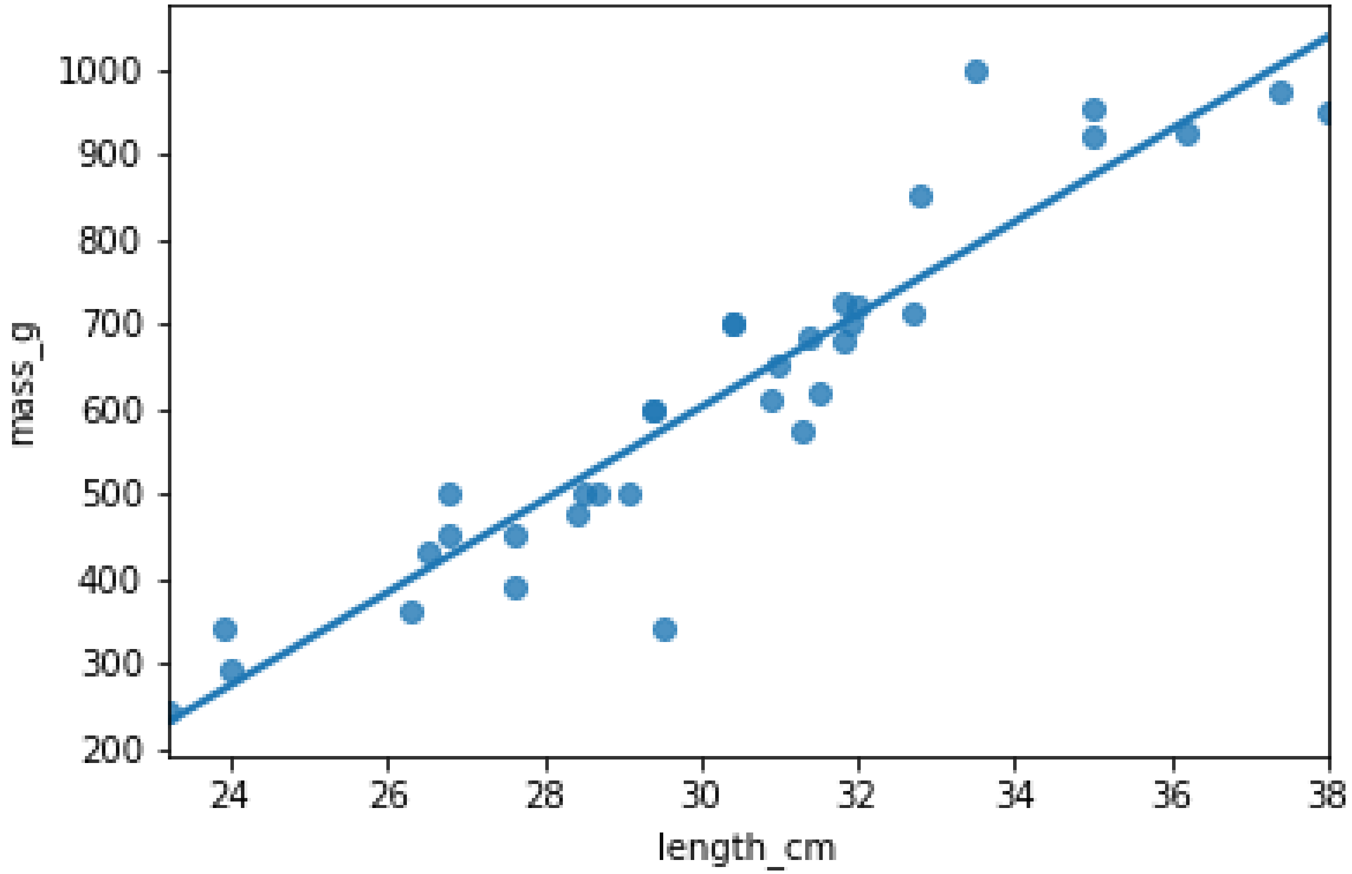
The fish dataset: bream

```
bream = fish[fish["species"] == "Bream"]
print(bream.head())
```

```
  species  mass_g  length_cm
0   Bream   242.0       23.2
1   Bream   290.0       24.0
2   Bream   340.0       23.9
3   Bream   363.0       26.3
4   Bream   430.0       26.5
```

Plotting mass vs. length

```
sns.regplot(x="length_cm",
            y="mass_g",
            data=bream,
            ci=None)

plt.show()
```



Running the model

```
mdl_mass_vs_length = ols("mass_g ~ length_cm", data=bream).fit()
print(mdl_mass_vs_length.params)
```

```
Intercept   -1035.347565
length_cm      54.549981
dtype: float64
```

Data on explanatory values to predict

> If I set the explanatory variables to these values,
> what value would the response variable have?

```
explanatory_data = pd.DataFrame({"length_cm": np.arange(20, 41)})
```

```
    length_cm
0          20
1          21
2          22
3          23
4          24
5          25
     ...
```

Call predict()

```
print(mdl_mass_vs_length.predict(explanatory_data))
```

```
0       55.652054
1      110.202035
2      164.752015
3      219.301996
4      273.851977
    ...
16     928.451749
17     983.001730
18    1037.551710
19    1092.101691
20    1146.651672
Length: 21, dtype: float64
```

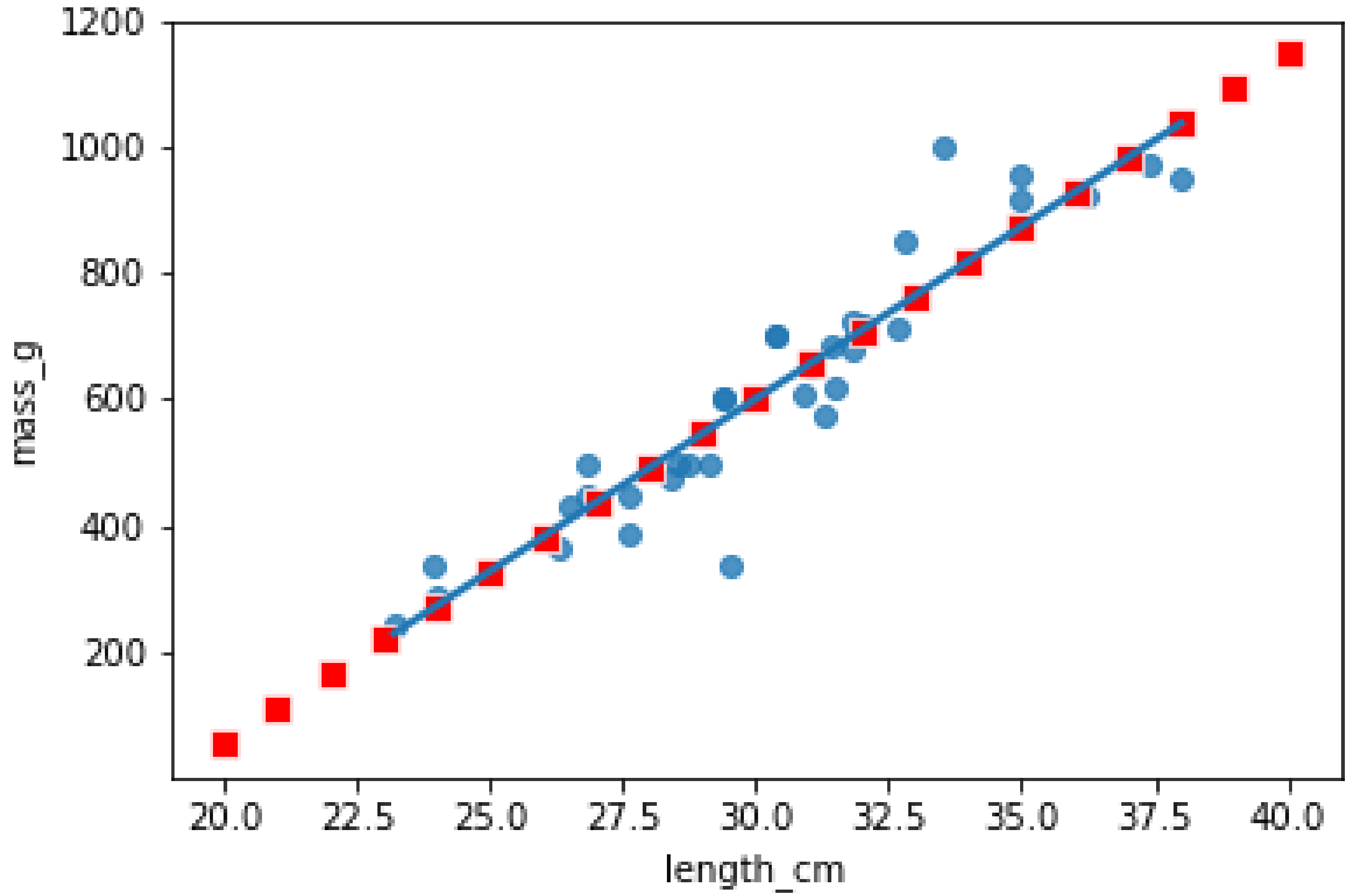
Predicting inside a DataFrame

```
explanatory_data = pd.DataFrame(
  {"length_cm": np.arange(20, 41)}
)
prediction_data = explanatory_data.assign(
    mass_g=mdl_mass_vs_length.predict(explanatory_data)
)
print(prediction_data)
```

```
    length_cm       mass_g
0          20    55.652054
1          21   110.202035
2          22   164.752015
3          23   219.301996
4          24   273.851977
..        ...          ...
16         36   928.451749
17         37   983.001730
18         38  1037.551710
19         39  1092.101691
20         40  1146.651672
```

Showing predictions

```
import matplotlib.pyplot as plt
import seaborn as sns
fig = plt.figure()
sns.regplot(x="length_cm",
            y="mass_g",
            ci=None,
            data=bream,)
sns.scatterplot(x="length_cm",
                y="mass_g",
                data=prediction_data,
                color="red",
                marker="s")
plt.show()
```

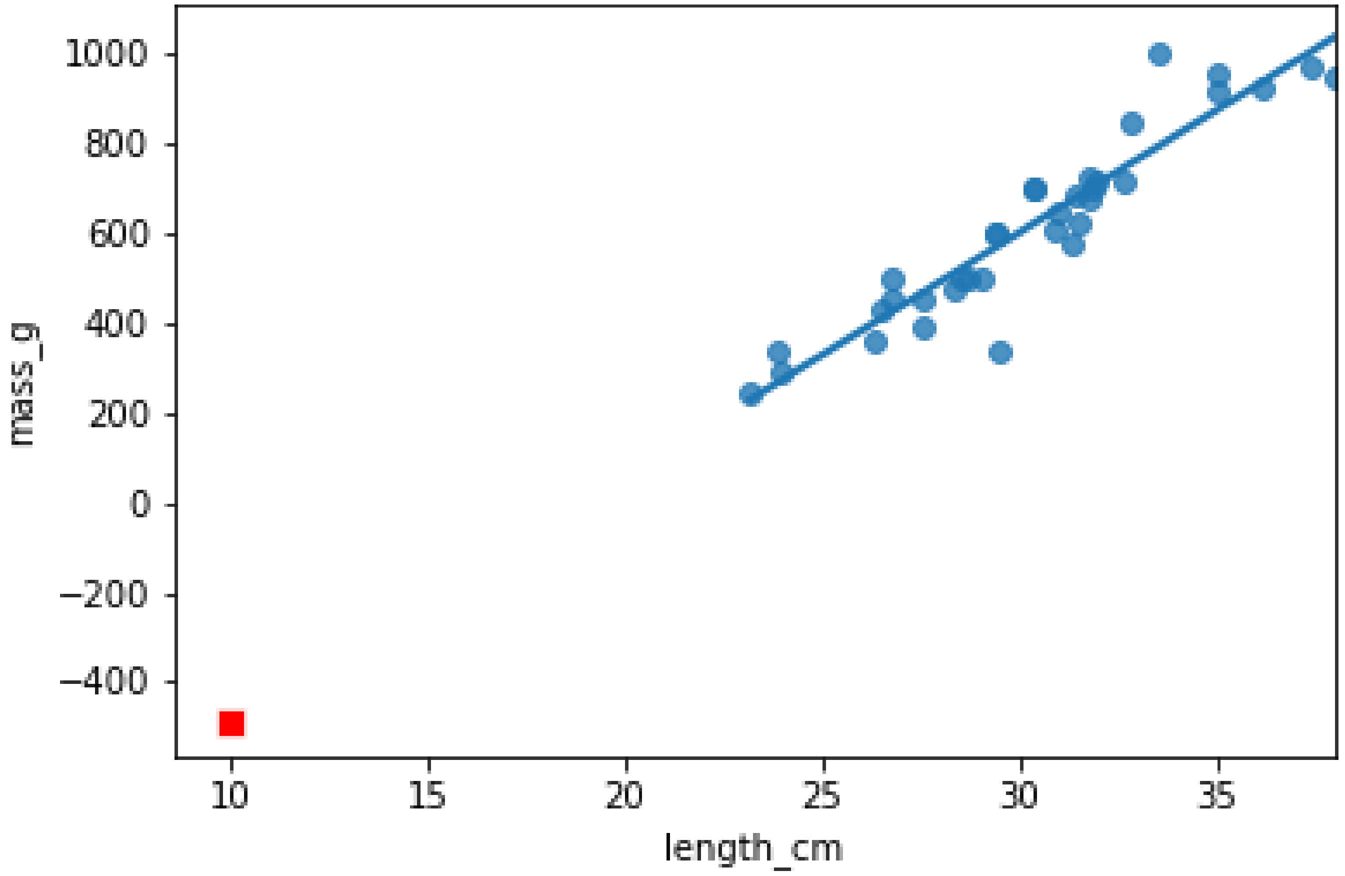


Extrapolating

Extrapolating means making predictions outside the range of observed data.

```
little_bream = pd.DataFrame({"length_cm": [10]})

pred_little_bream = little_bream.assign(
    mass_g=mdl_mass_vs_length.predict(little_bream))

print(pred_little_bream)
```

```
   length_cm      mass_g
0         10 -489.847756
```



Let's practice!

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Working with model objects

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

.params attribute

```
from statsmodels.formula.api import ols
mdl_mass_vs_length = ols("mass_g ~ length_cm", data = bream).fit()
print(mdl_mass_vs_length.params)
```

```
Intercept   -1035.347565
length_cm      54.549981
dtype: float64
```

.fittedvalues attribute

Fitted values: predictions on the original dataset

```
print(mdl_mass_vs_length.fittedvalues)
```

or equivalently

```
explanatory_data = bream["length_cm"]

print(mdl_mass_vs_length.predict(explanatory_data))
```

```
0       230.211993
1       273.851977
2       268.396979
3       399.316934
4       410.226930
    ...
30      873.901768
31      873.901768
32      939.361745
33     1004.821722
34     1037.551710
Length: 35, dtype: float64
```

.resid attribute

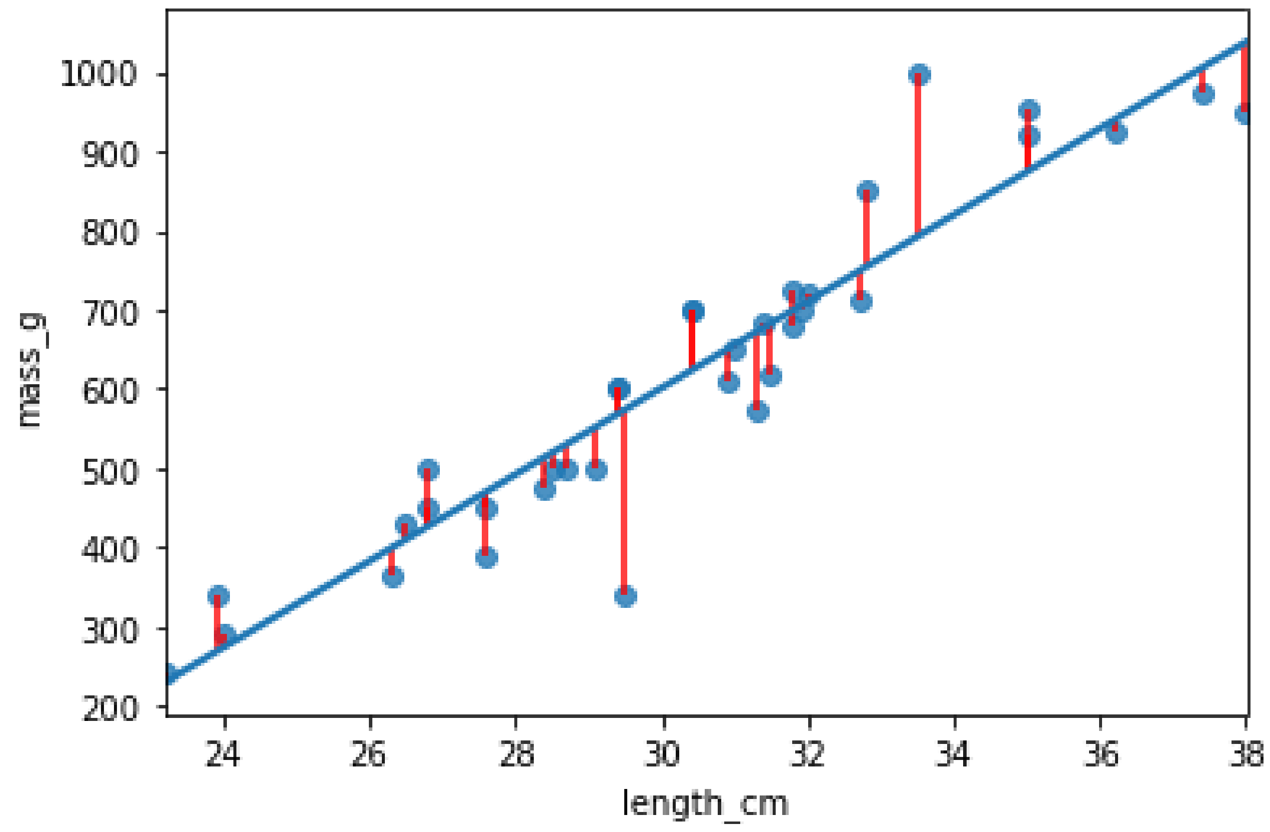
Residuals: actual response values minus predicted response values

```
print(mdl_mass_vs_length.resid)
```

or equivalently

```
print(bream["mass_g"] - mdl_mass_vs_length.fittedvalues)
```

```
0    11.788007
1    16.148023
2    71.603021
3   -36.316934
4    19.773070
    ...
```



.summary()

```
mdl_mass_vs_length.summary()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 mass_g   R-squared:                       0.878
Model:                            OLS   Adj. R-squared:                  0.874
Method:                 Least Squares   F-statistic:                     237.6
Date:                Thu, 29 Oct 2020   Prob (F-statistic):           1.22e-16
Time:                        13:23:21   Log-Likelihood:                -199.35
No. Observations:                  35   AIC:                             402.7
Df Residuals:                      33   BIC:                             405.8
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
<-----------------------------------------------------------------------------
Intercept  -1035.3476    107.973     -9.589      0.000   -1255.020    -815.676
length_cm     54.5500      3.539     15.415      0.000      47.350      61.750
==============================================================================
Omnibus:                        7.314   Durbin-Watson:                   1.478
Prob(Omnibus):                  0.026   Jarque-Bera (JB):               10.857
Skew:                          -0.252   Prob(JB):                      0.00439
Kurtosis:                       5.682   Cond. No.                         263.
```

```
                 OLS Regression Results
=============================================================================
Dep. Variable:                mass_g   R-squared:                       0.878
Model:                           OLS   Adj. R-squared:                  0.874
Method:                Least Squares   F-statistic:                     237.6
Date:               Thu, 29 Oct 2020   Prob (F-statistic):           1.22e-16
Time:                       13:23:21   Log-Likelihood:                -199.35
No. Observations:                 35   AIC:                             402.7
Df Residuals:                     33   BIC:                             405.8
Df Model:                          1
Covariance Type:           nonrobust
```

```
                 coef    std err          t      P>|t|      [0.025      0.975]
<------------------------------------------------------------------------------
Intercept  -1035.3476    107.973     -9.589      0.000   -1255.020    -815.676
length_cm     54.5500      3.539     15.415      0.000      47.350      61.750
==============================================================================
Omnibus:                        7.314   Durbin-Watson:                   1.478
Prob(Omnibus):                  0.026   Jarque-Bera (JB):               10.857
Skew:                          -0.252   Prob(JB):                      0.00439
Kurtosis:                       5.682   Cond. No.                         263.
```

Let's practice!

Regression to the mean

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

The concept

- Response value = fitted value + residual
- "The stuff you explained" + "the stuff you couldn't explain"
- Residuals exist due to problems in the model *and* fundamental randomness
- Extreme cases are often due to randomness
- *Regression to the mean* means extreme cases don't persist over time

Pearson's father son dataset

- 1078 father/son pairs
- Do tall fathers have tall sons?

father_height_cm	son_height_cm
165.2	151.8
160.7	160.6
165.0	160.9
167.0	159.5
155.3	163.3
...	...

[1] Adapted from https://www.rdocumentation.org/packages/UsingR/topics/father.son

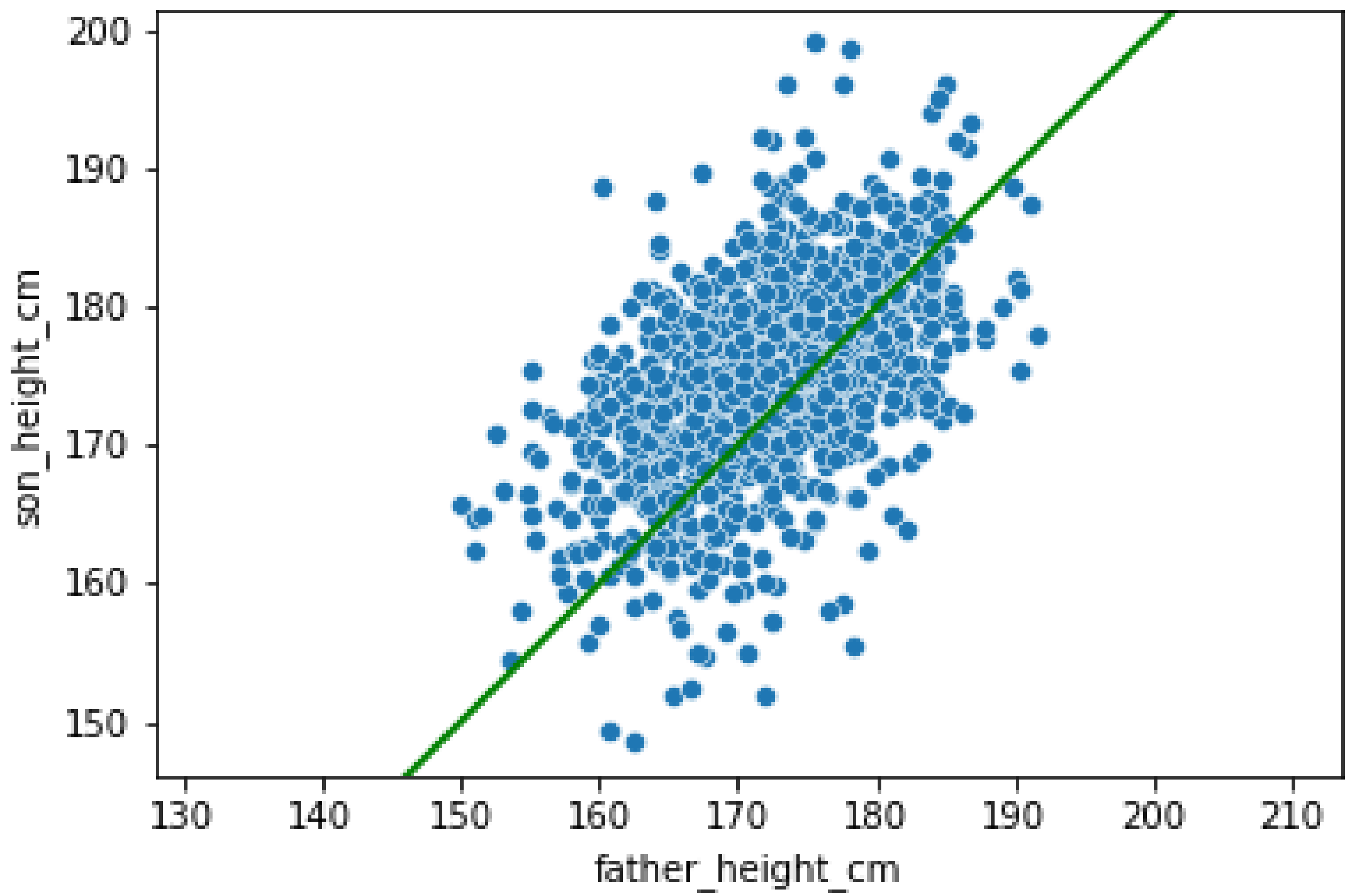
Scatter plot

```
fig = plt.figure()
```

```
sns.scatterplot(x="father_height_cm",
                y="son_height_cm",
                data=father_son)
```

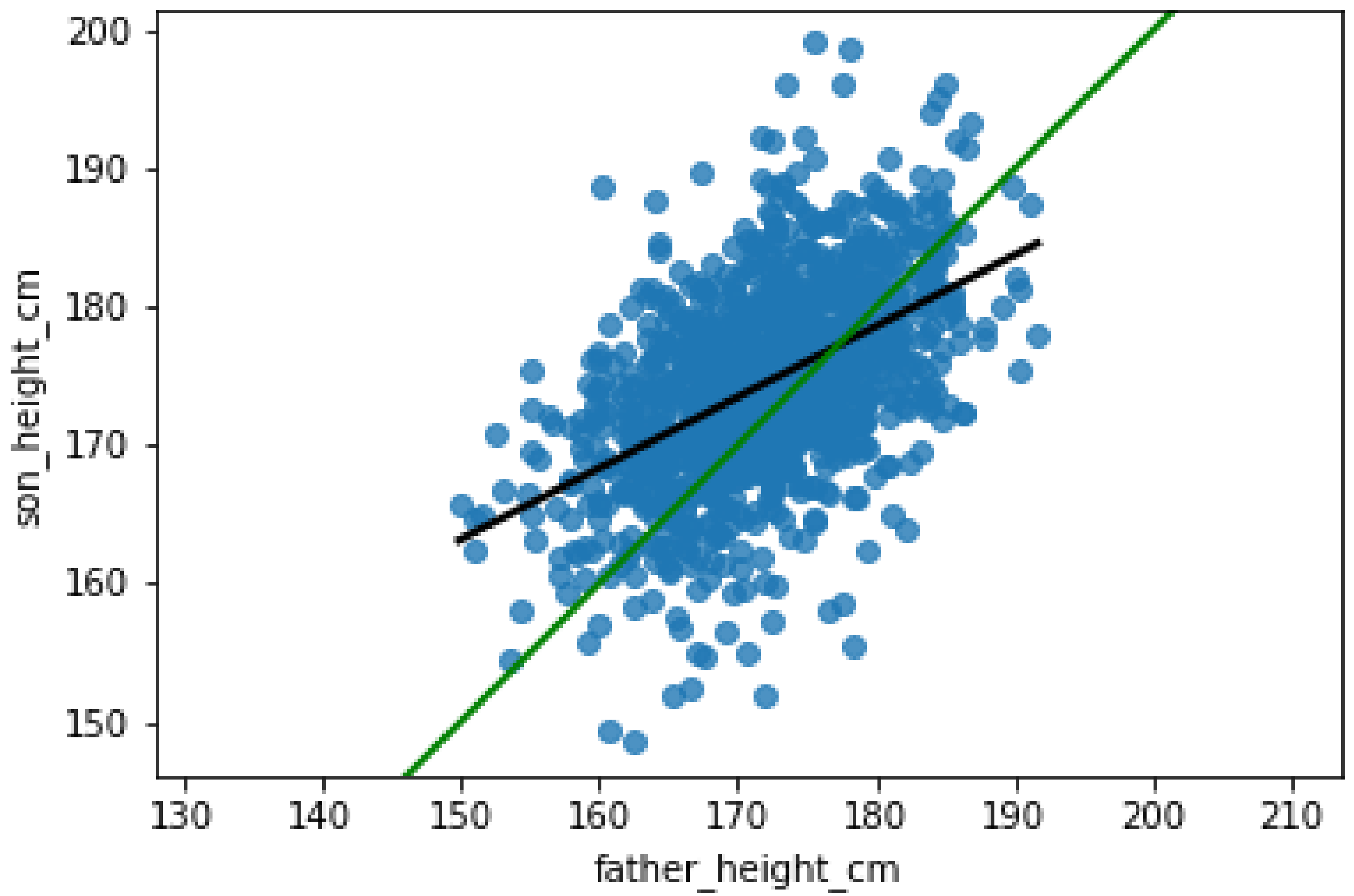
```
plt.axline(xy1=(150, 150),
           slope=1,
           linewidth=2,
           color="green")
```

```
plt.axis("equal")
plt.show()
```



Adding a regression line

```
fig = plt.figure()

sns.regplot(x="father_height_cm",
            y="son_height_cm",
            data=father_son,
            ci = None,
            line_kws={"color": "black"})

plt.axline(xy1 = (150, 150),
           slope=1,
           linewidth=2,
           color="green")

plt.axis("equal")
plt.show()
```



Running a regression

```
mdl_son_vs_father = ols("son_height_cm ~ father_height_cm",
                        data = father_son).fit()
print(mdl_son_vs_father.params)
```

```
Intercept           86.071975
father_height_cm     0.514093
dtype: float64
```

Making predictions

```
really_tall_father = pd.DataFrame(
  {"father_height_cm": [190]})

mdl_son_vs_father.predict(
  really_tall_father)
```

```
183.7
```

```
really_short_father = pd.DataFrame(
  {"father_height_cm": [150]})

mdl_son_vs_father.predict(
  really_short_father)
```

```
163.2
```

Let's practice!

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Transforming variables

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

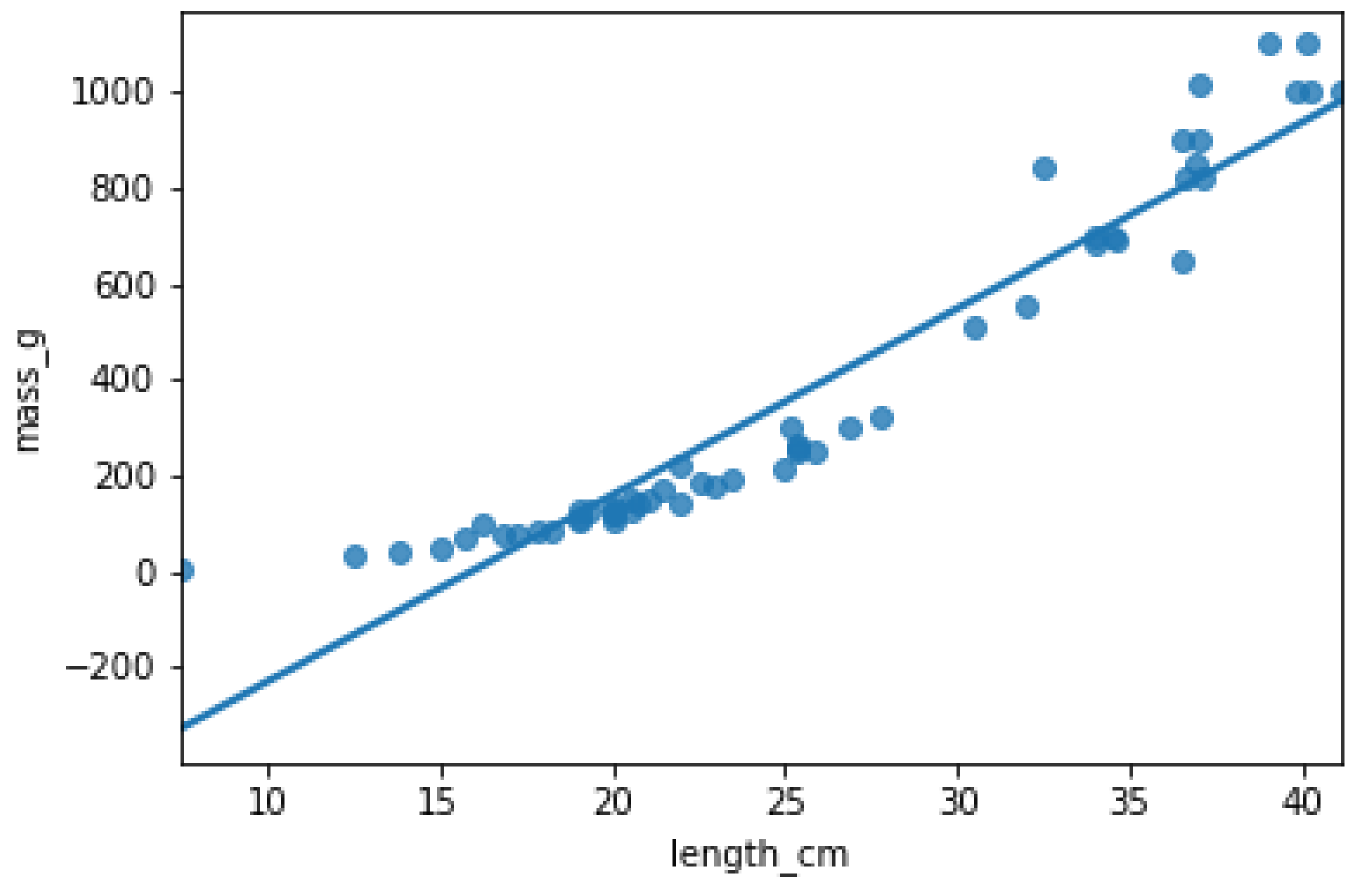
Perch dataset

```
perch = fish[fish["species"] == "Perch"]
print(perch.head())
```

```
   species  mass_g  length_cm
55   Perch     5.9        7.5
56   Perch    32.0       12.5
57   Perch    40.0       13.8
58   Perch    51.5       15.0
59   Perch    70.0       15.7
```

It's not a linear relationship

```
sns.regplot(x="length_cm",
            y="mass_g",
            data=perch,
            ci=None)

plt.show()
```

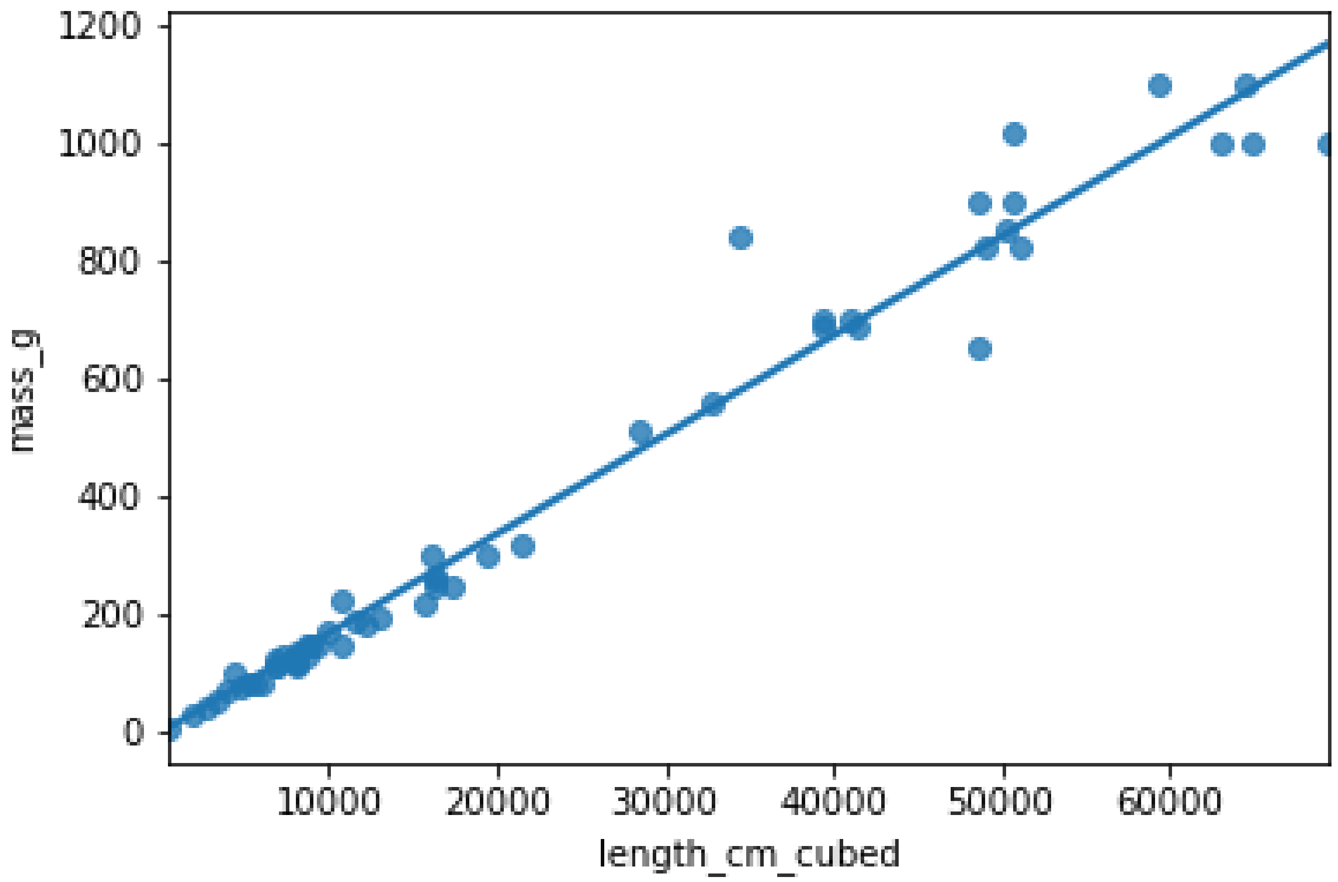


Bream vs. perch

Plotting mass vs. length cubed

```
perch["length_cm_cubed"] = perch["length_cm"] ** 3
```

```
sns.regplot(x="length_cm_cubed",
            y="mass_g",
            data=perch,
            ci=None)
plt.show()
```



Modeling mass vs. length cubed

```
perch["length_cm_cubed"] = perch["length_cm"] ** 3


mdl_perch = ols("mass_g ~ length_cm_cubed", data=perch).fit()
mdl_perch.params
```

```
Intercept          -0.117478
length_cm_cubed     0.016796
dtype: float64
```

Predicting mass vs. length cubed

```
explanatory_data = pd.DataFrame({"length_cm_cubed": np.arange(10, 41, 5) ** 3,
                                 "length_cm": np.arange(10, 41, 5)})
```

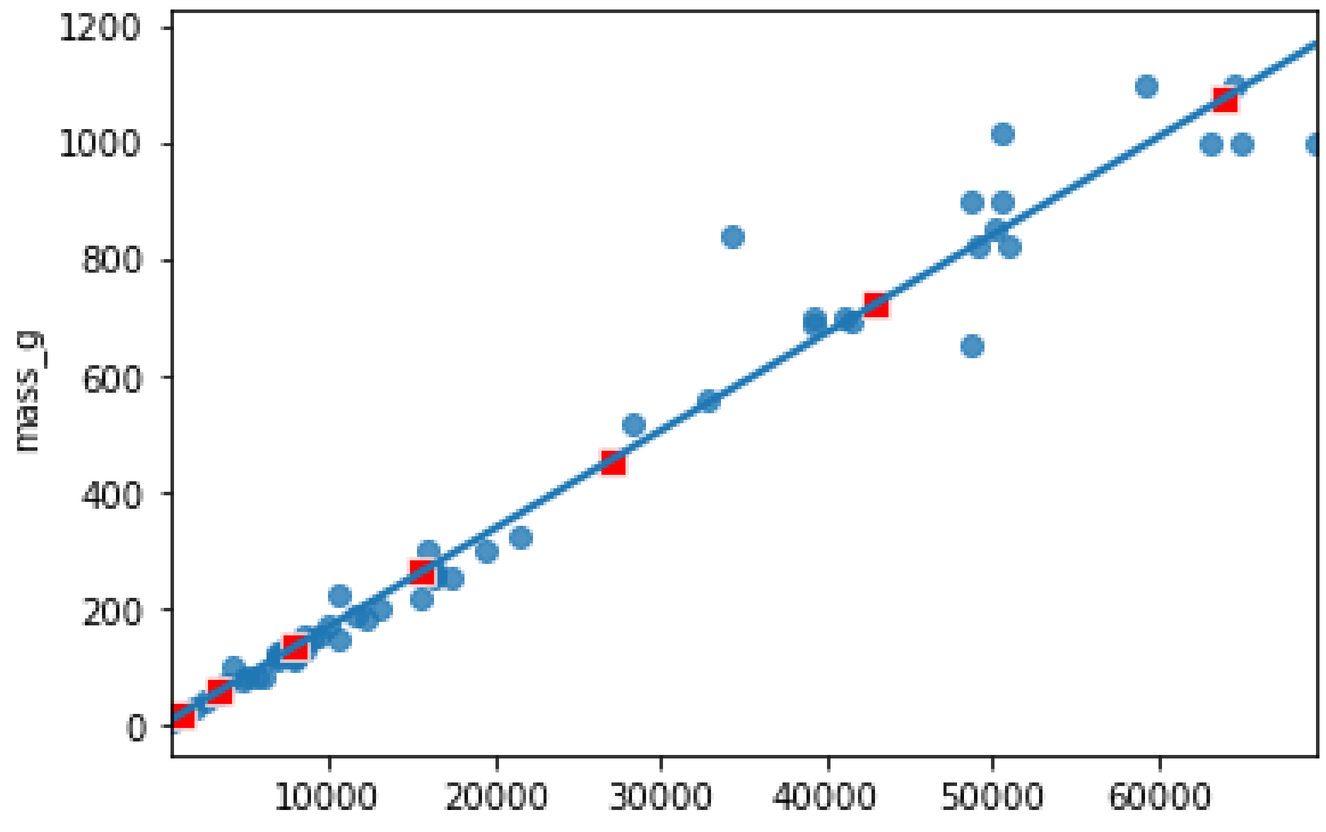
```
prediction_data = explanatory_data.assign(
  mass_g=mdl_perch.predict(explanatory_data))
print(prediction_data)
```

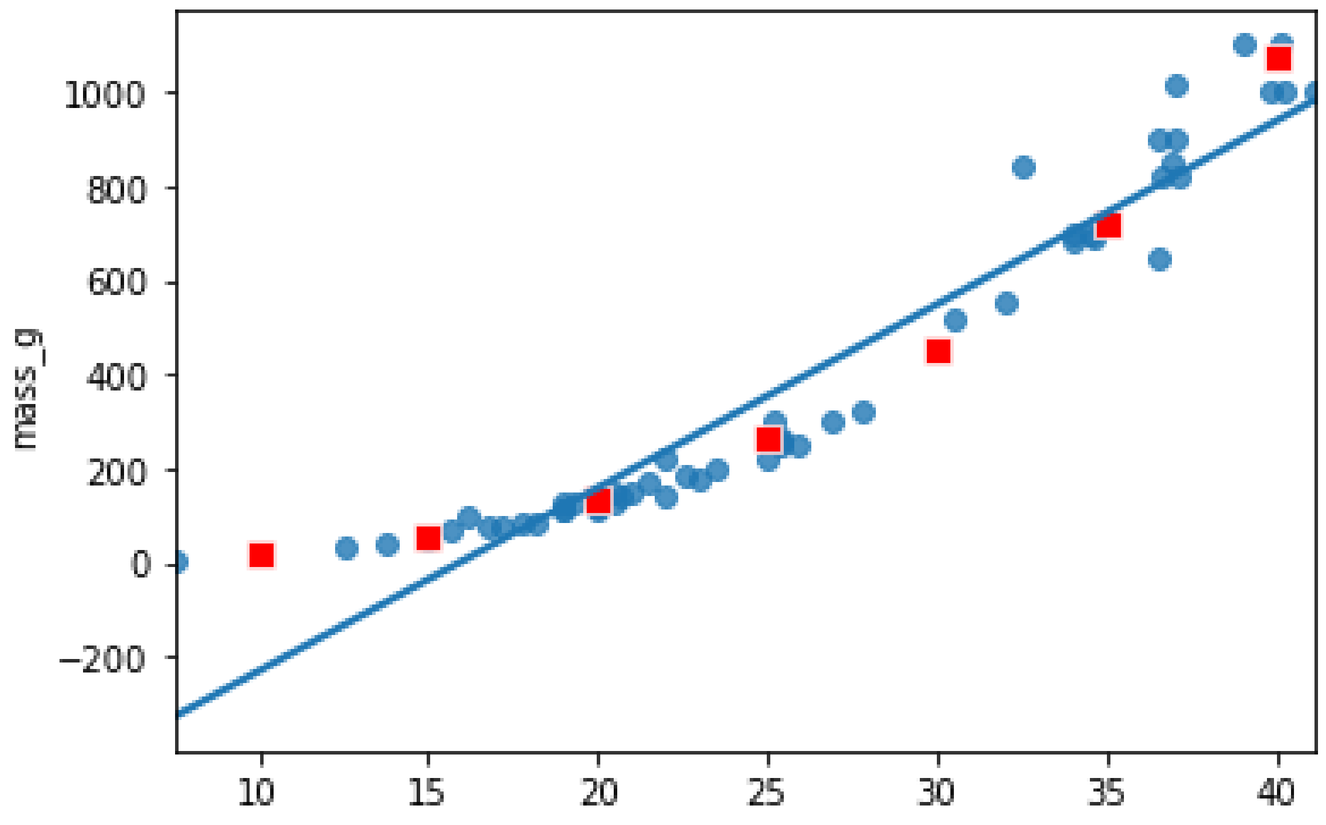
```
   length_cm_cubed  length_cm       mass_g
0             1000         10    16.678135
1             3375         15    56.567717
2             8000         20   134.247429
3            15625         25   262.313982
4            27000         30   453.364084
5            42875         35   719.994447
6            64000         40  1074.801781
```

Plotting mass vs. length cubed

```
fig = plt.figure()
sns.regplot(x="length_cm_cubed", y="mass_g",
            data=perch, ci=None)
sns.scatterplot(data=prediction_data,
                x="length_cm_cubed", y="mass_g",
                color="red", marker="s")
```

```
fig = plt.figure()
sns.regplot(x="length_cm", y="mass_g",
            data=perch, ci=None)
sns.scatterplot(data=prediction_data,
                x="length_cm", y="mass_g",
                color="red", marker="s")
```





Facebook advertising dataset

How advertising works

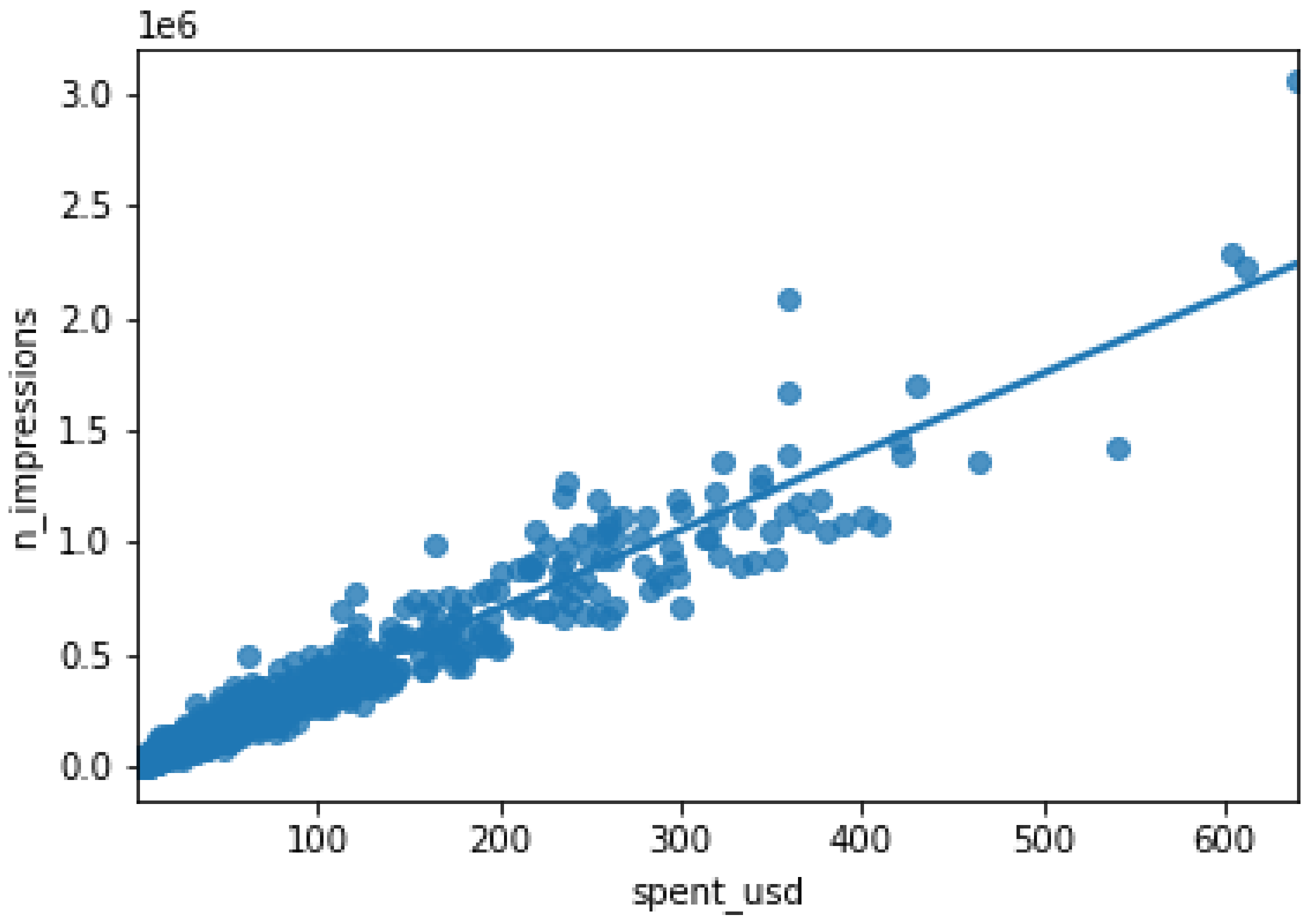
1. Pay Facebook to shows ads.
2. People see the ads ("impressions").
3. Some people who see it, click it.

- 936 rows
- Each row represents 1 advert

spent_usd	n_impressions	n_clicks
1.43	7350	1
1.82	17861	2
1.25	4259	1
1.29	4133	1
4.77	15615	3
...	...	...

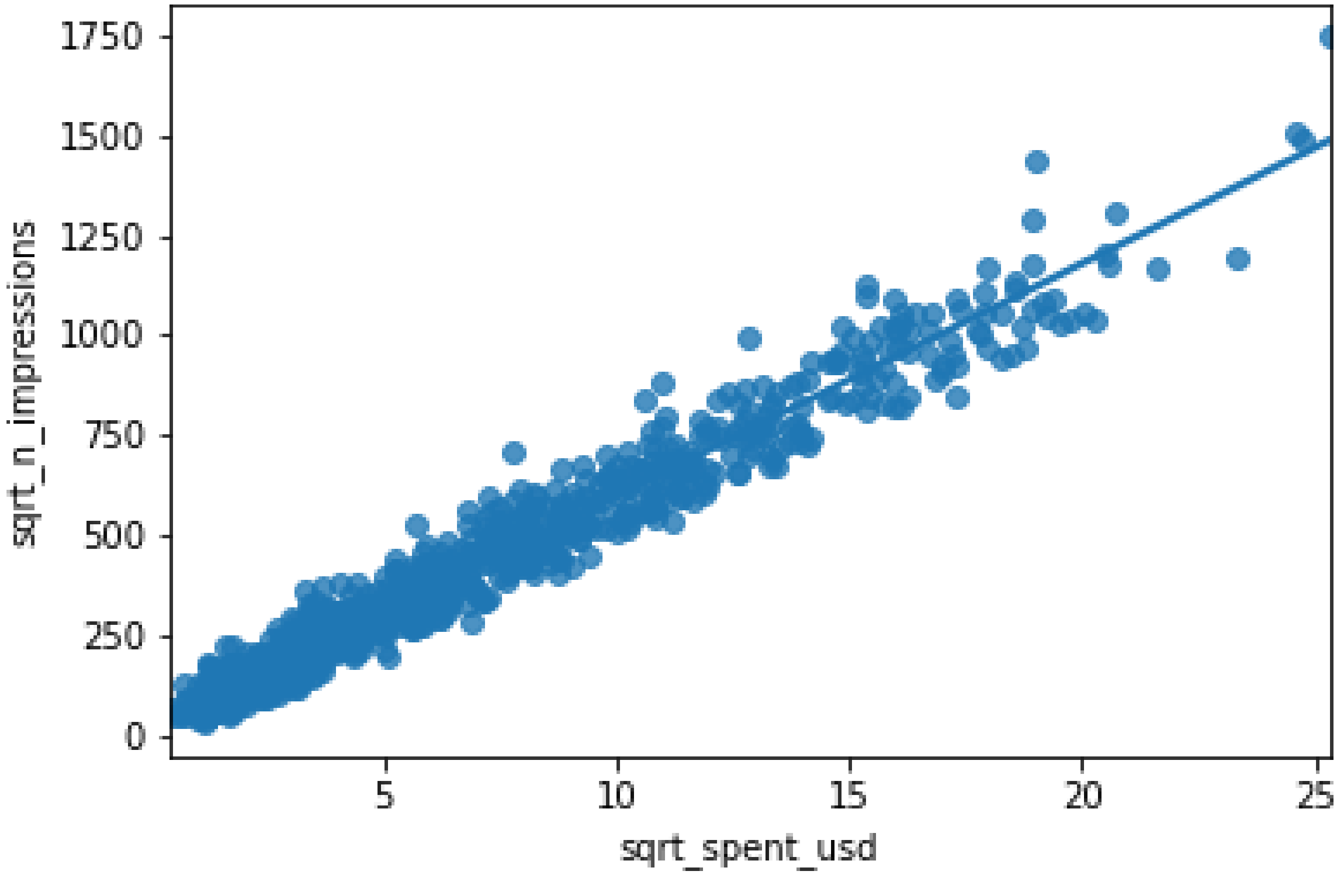
Plot is cramped

```
sns.regplot(x="spent_usd",
            y="n_impressions",
            data=ad_conversion,
            ci=None)
```



Square root vs square root

```
ad_conversion["sqrt_spent_usd"] = np.sqrt(
  ad_conversion["spent_usd"])

ad_conversion["sqrt_n_impressions"] = np.sqrt(
  ad_conversion["n_impressions"])

sns.regplot(x="sqrt_spent_usd",
            y="sqrt_n_impressions",
            data=ad_conversion,
            ci=None)
```



Modeling and predicting

```
mdl_ad = ols("sqrt_n_impressions ~ sqrt_spent_usd", data=ad_conversion).fit()
```

```
explanatory_data = pd.DataFrame({"sqrt_spent_usd": np.sqrt(np.arange(0, 601, 100)),
                                 "spent_usd": np.arange(0, 601, 100)})
```

```
prediction_data = explanatory_data.assign(sqrt_n_impressions=mdl_ad.predict(explanatory_data),
                          n_impressions=mdl_ad.predict(explanatory_data) ** 2)
print(prediction_data)
```

```
   sqrt_spent_usd  spent_usd  sqrt_n_impressions  n_impressions
0        0.000000          0           15.319713   2.346936e+02
1       10.000000        100          597.736582   3.572890e+05
2       14.142136        200          838.981547   7.038900e+05
3       17.320508        300         1024.095320   1.048771e+06
4       20.000000        400         1180.153450   1.392762e+06
5       22.360680        500         1317.643422   1.736184e+06
6       24.494897        600         1441.943858   2.079202e+06
```

Let's practice!

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

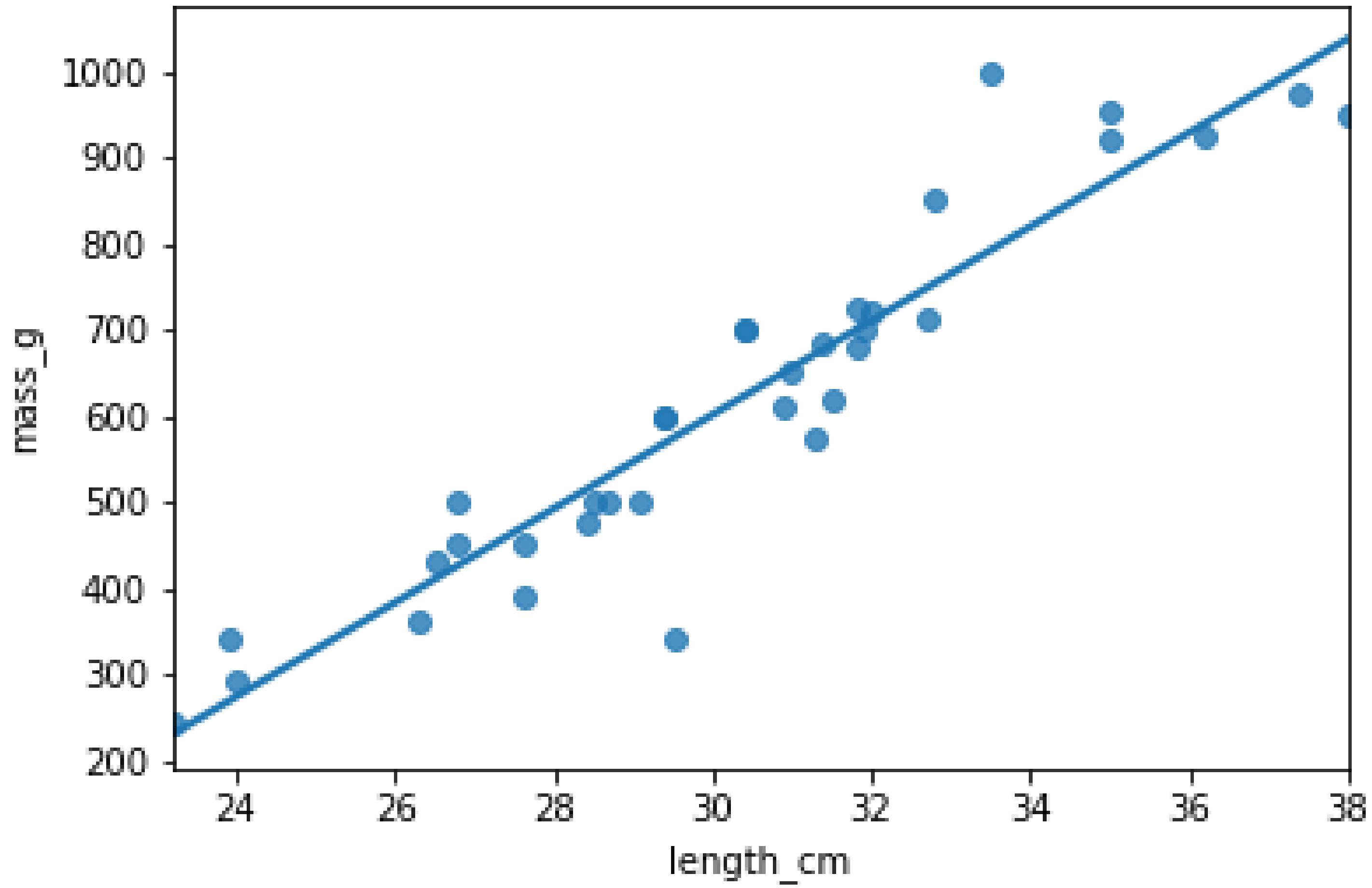
Quantifying model fit

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

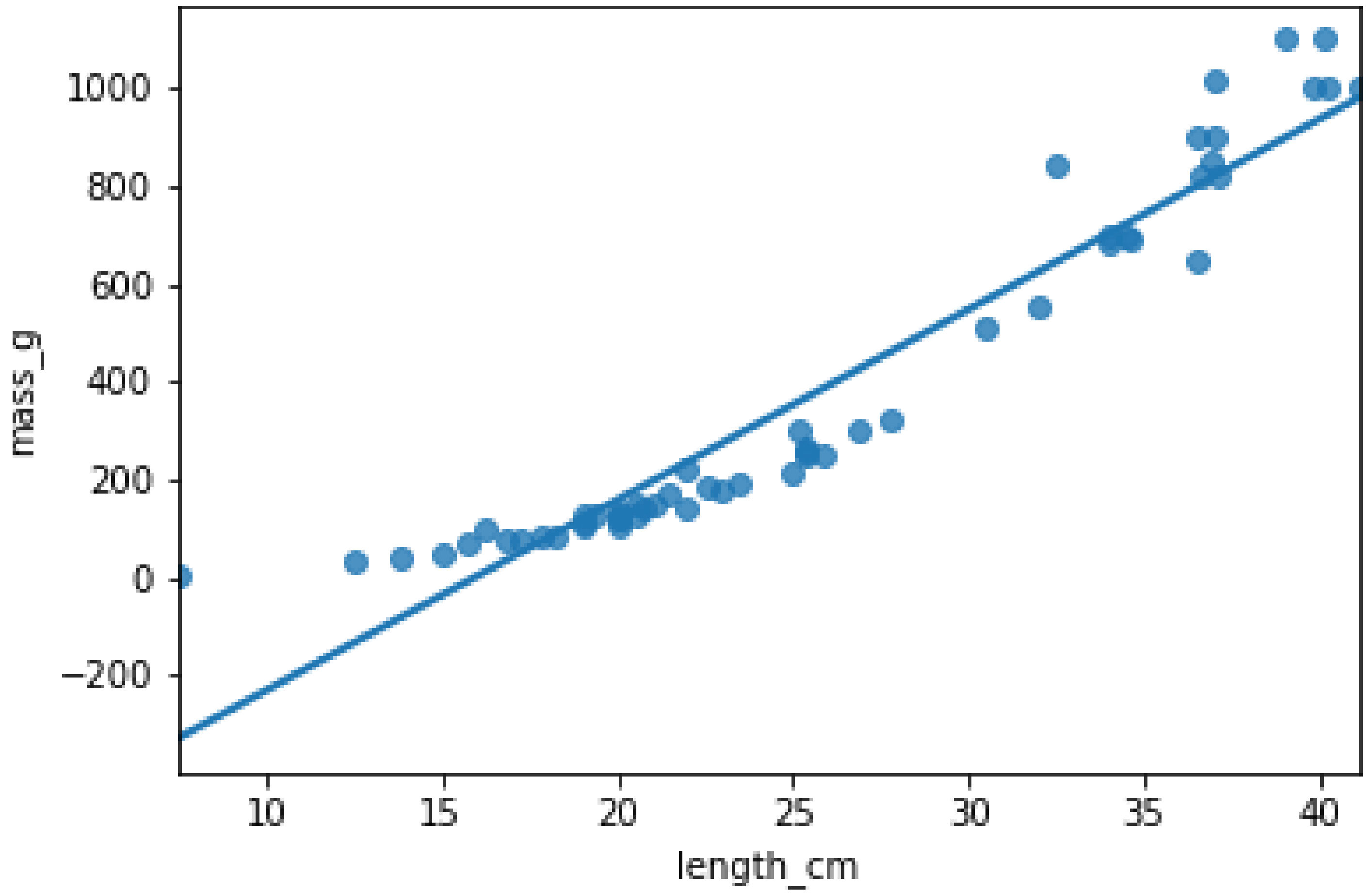
Maarten Van den Broeck
Content Developer at DataCamp

Bream and perch models

Bream



Perch



Coefficient of determination

Sometimes called "r-squared" or "R-squared".

> The proportion of the variance in the response variable that is predictable from the explanatory variable

- `1` means a perfect fit
- `0` means the worst possible fit

.summary()

Look at the value titled "R-Squared"

```
mdl_bream = ols("mass_g ~ length_cm", data=bream).fit()
```

```
print(mdl_bream.summary())
```

```
# Some lines of output omitted


                            OLS Regression Results
Dep. Variable:                  mass_g   R-squared:                       0.878
Model:                             OLS   Adj. R-squared:                  0.874
Method:                  Least Squares   F-statistic:                     237.6
```

.rsquared attribute

```
print(mdl_bream.rsquared)
```

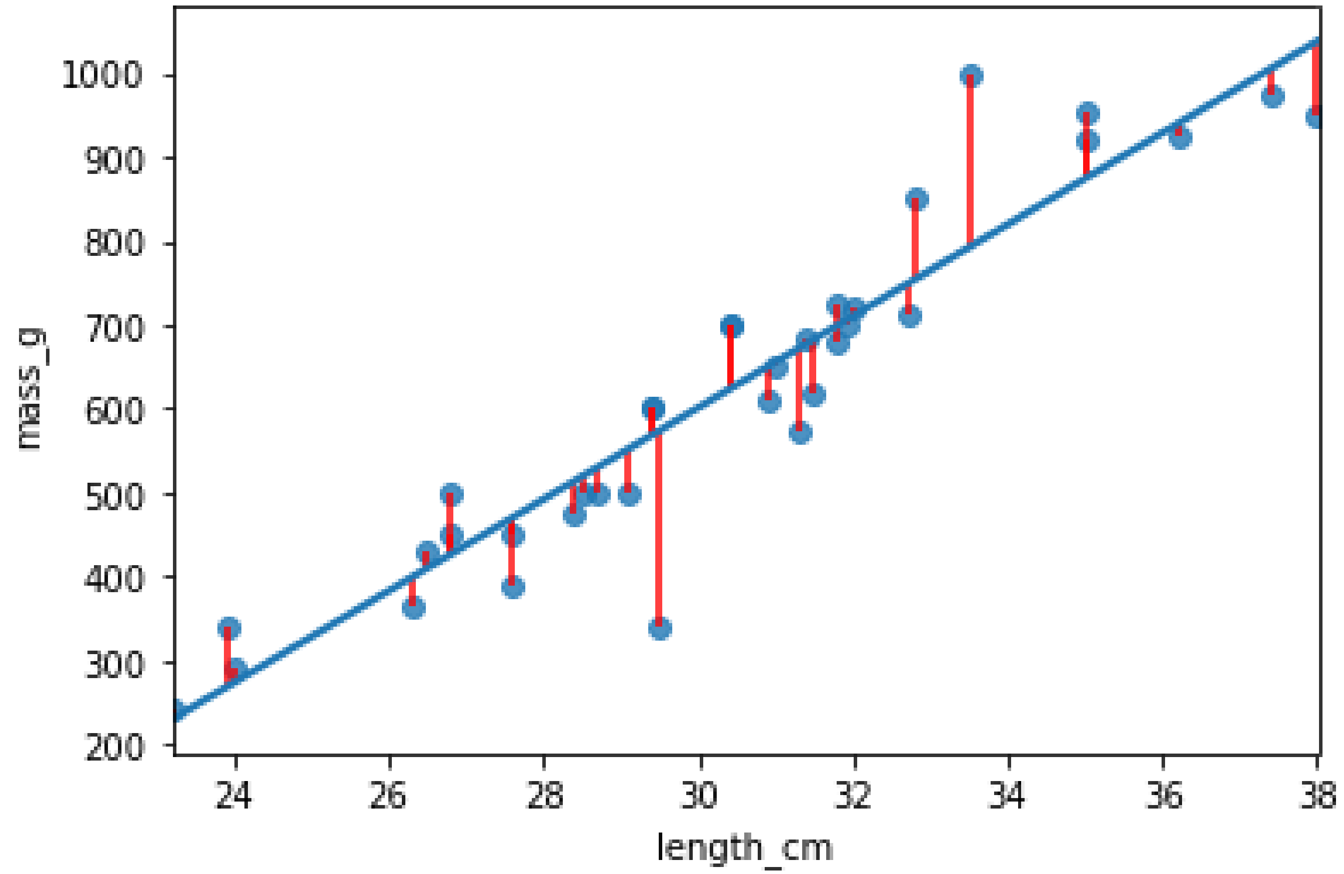
```
0.8780627095147174
```

It's just correlation squared

```
coeff_determination = bream["length_cm"].corr(bream["mass_g"]) ** 2
print(coeff_determination)
```

```
0.8780627095147173
```

Residual standard error (RSE)



- A "typical" difference between a prediction and an observed response
- It has the same unit as the response variable.
- MSE = RSE^2

.mse_resid attribute

```
mse = mdl_bream.mse_resid
print('mse: ', mse)
```

```
mse:  5498.555084973521
```

```
rse = np.sqrt(mse)
print("rse: ", rse)
```

```
rse:  74.15224261594197
```

Calculating RSE: residuals squared

```
residuals_sq = mdl_bream.resid ** 2

print("residuals sq: \n", residuals_sq)
```

```
residuals sq:
 0      138.957118
1      260.758635
2     5126.992578
3     1318.919660
4      390.974309
       
30    2125.047026
31    6576.923291
32     206.259713
33     889.335096
34    7665.302003
Length: 35, dtype: float64
```

Calculating RSE: sum of residuals squared

```
residuals_sq = mdl_bream.resid ** 2

resid_sum_of_sq = sum(residuals_sq)

print("resid sum of sq :",
      resid_sum_of_sq)
```

```
resid sum of sq : 181452.31780412616
```

Calculating RSE: degrees of freedom

```
residuals_sq = mdl_bream.resid ** 2

resid_sum_of_sq = sum(residuals_sq)

deg_freedom = len(bream.index) - 2

print("deg freedom: ", deg_freedom)
```

```
deg freedom:  33
```

Degrees of freedom equals the number of observations minus the number of model coefficients.

Calculating RSE: square root of ratio

```
residuals_sq = mdl_bream.resid ** 2

resid_sum_of_sq = sum(residuals_sq)

deg_freedom = len(bream.index) - 2

rse = np.sqrt(resid_sum_of_sq/deg_freedom)

print("rse :", rse)
```

```
rse : 74.15224261594197
```

Interpreting RSE

`mdl_bream` has an RSE of `74`.

> The difference between predicted bream masses and observed bream masses is typically about 74g.

Root-mean-square error (RMSE)

```
residuals_sq = mdl_bream.resid ** 2

resid_sum_of_sq = sum(residuals_sq)

deg_freedom = len(bream.index) - 2

rse = np.sqrt(resid_sum_of_sq/deg_freedom)

print("rse :", rse)
```

```
rse : 74.15224261594197
```

```
residuals_sq = mdl_bream.resid ** 2

resid_sum_of_sq = sum(residuals_sq)

n_obs = len(bream.index)

rmse = np.sqrt(resid_sum_of_sq/n_obs)

print("rmse :", rmse)
```

```
rmse : 72.00244396727619
```

Let's practice!

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Visualizing model fit

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

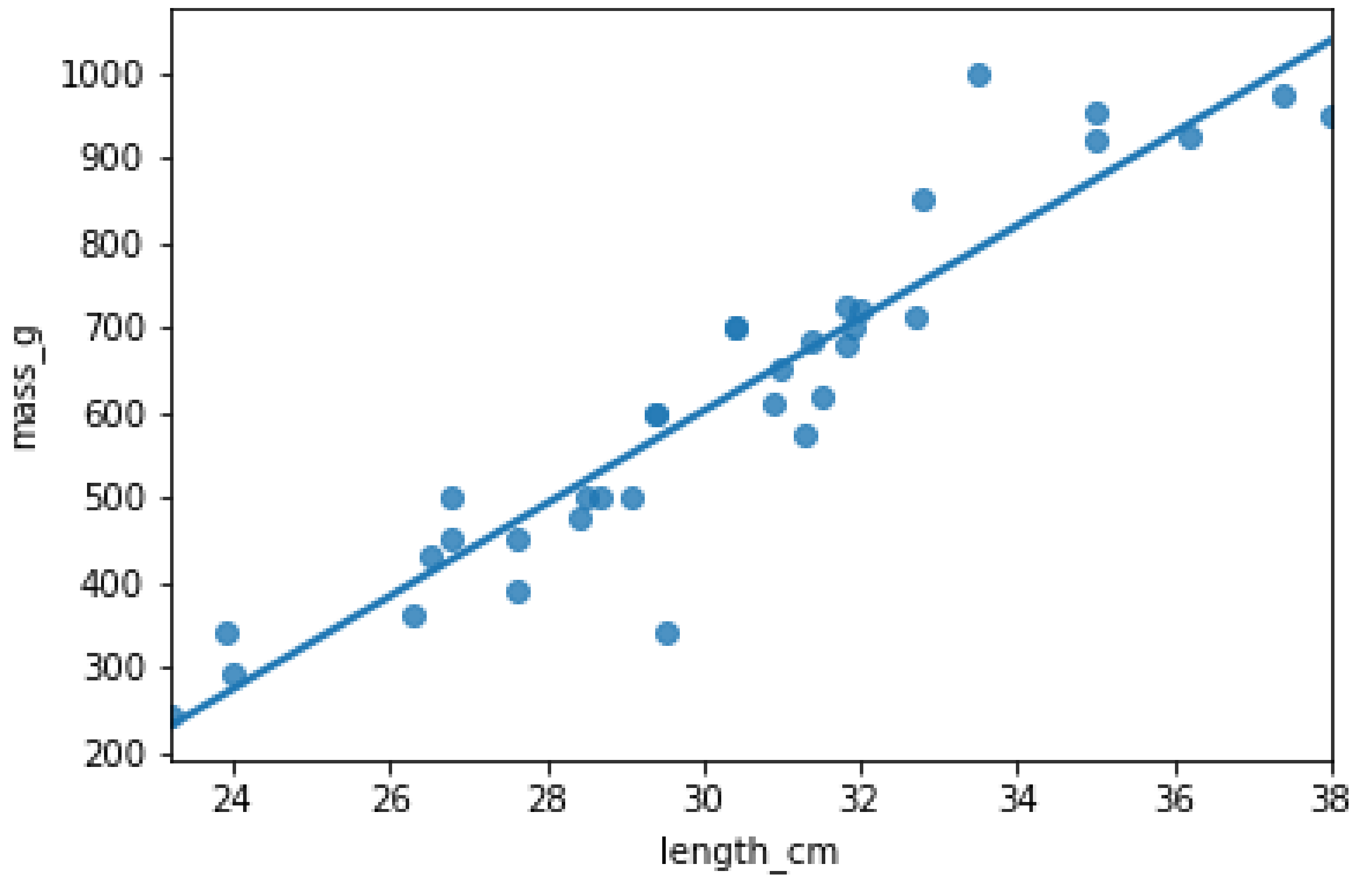
Residual properties of a good fit

- Residuals are normally distributed
- The mean of the residuals is zero

Bream and perch again

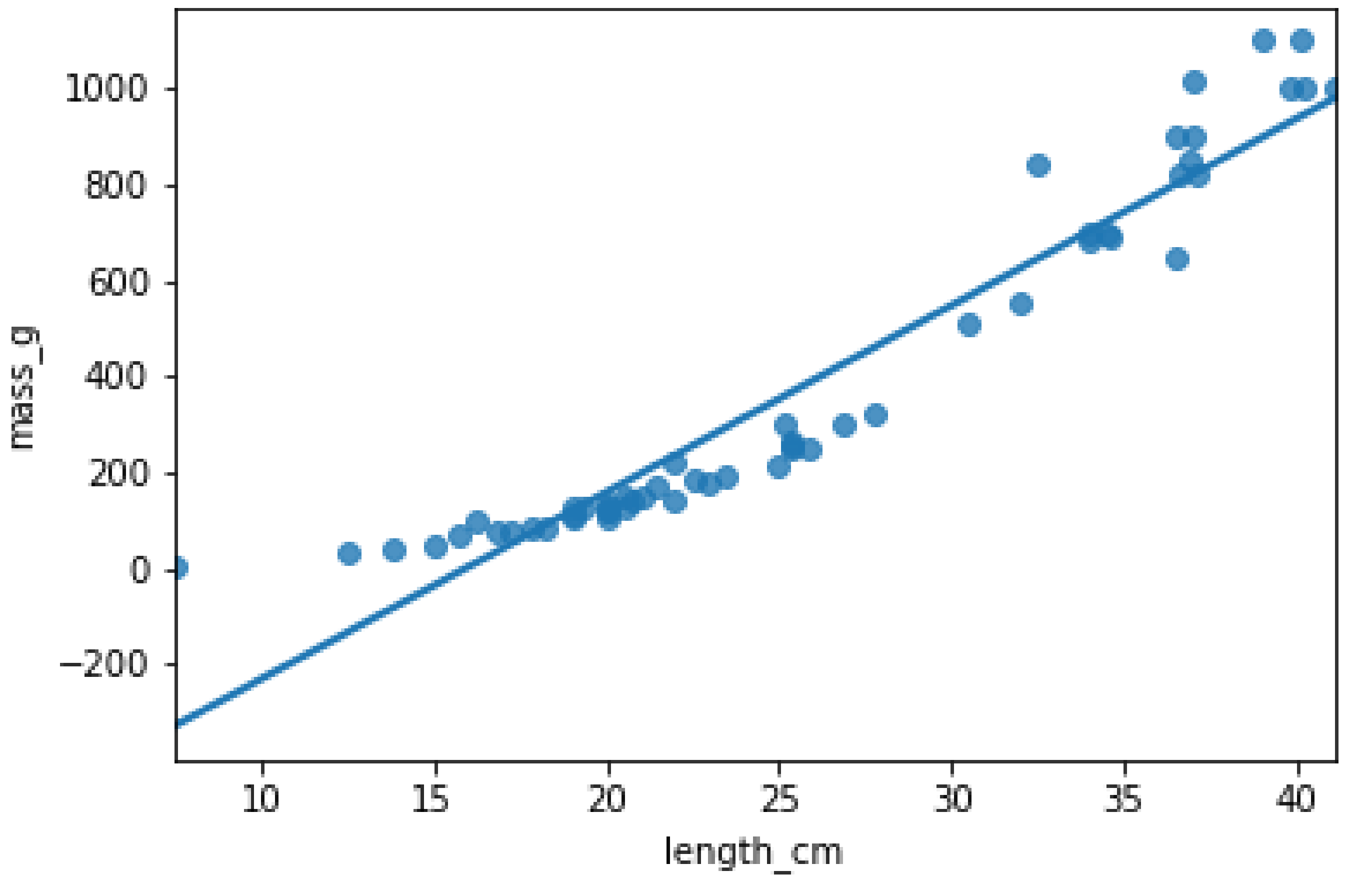
Bream: the "good" model

```
mdl_bream = ols("mass_g ~ length_cm", data=bream).fit()
```



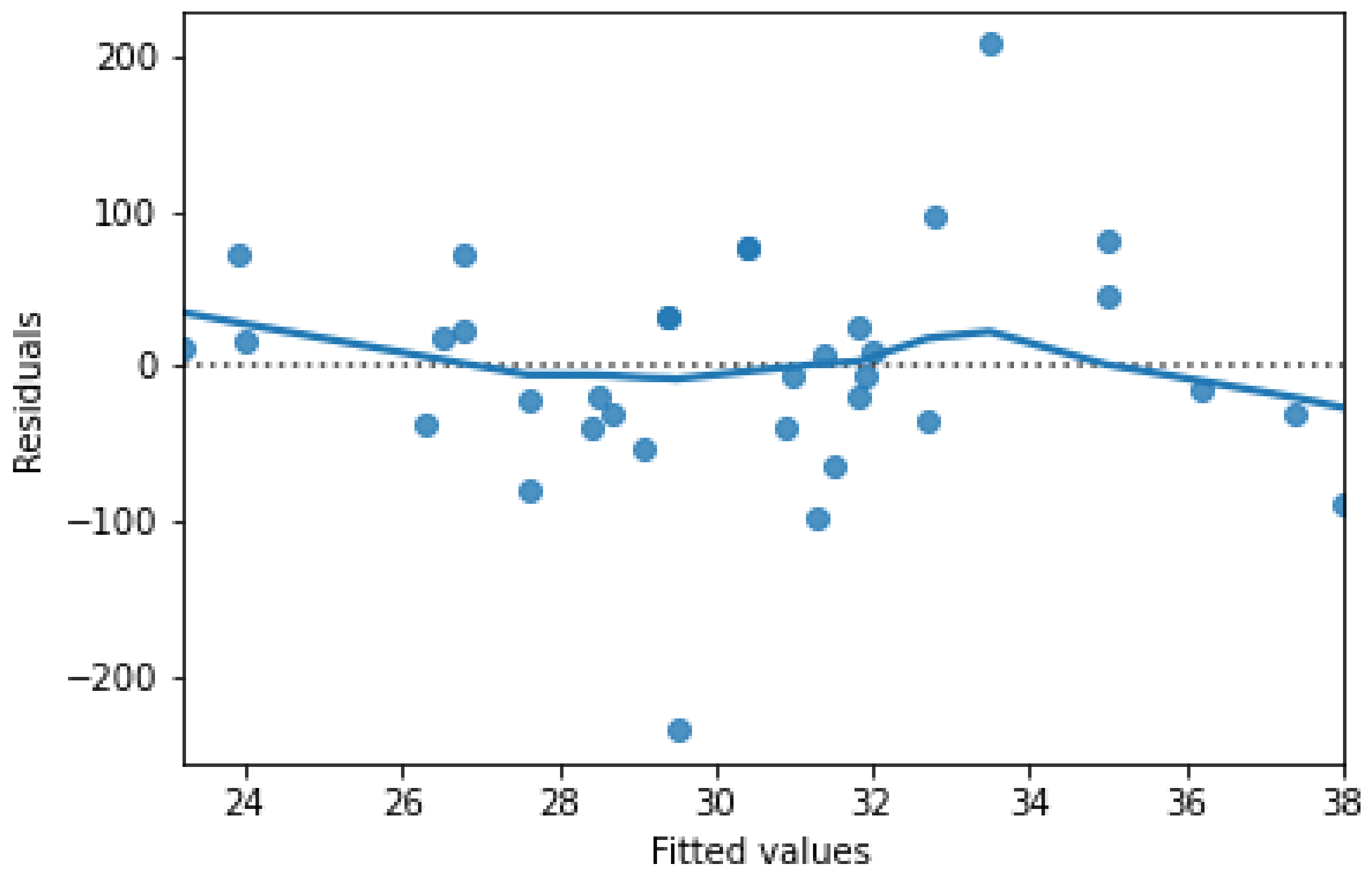
Perch: the "bad" model

```
mdl_perch = ols("mass_g ~ length_cm", data=perch).fit()
```

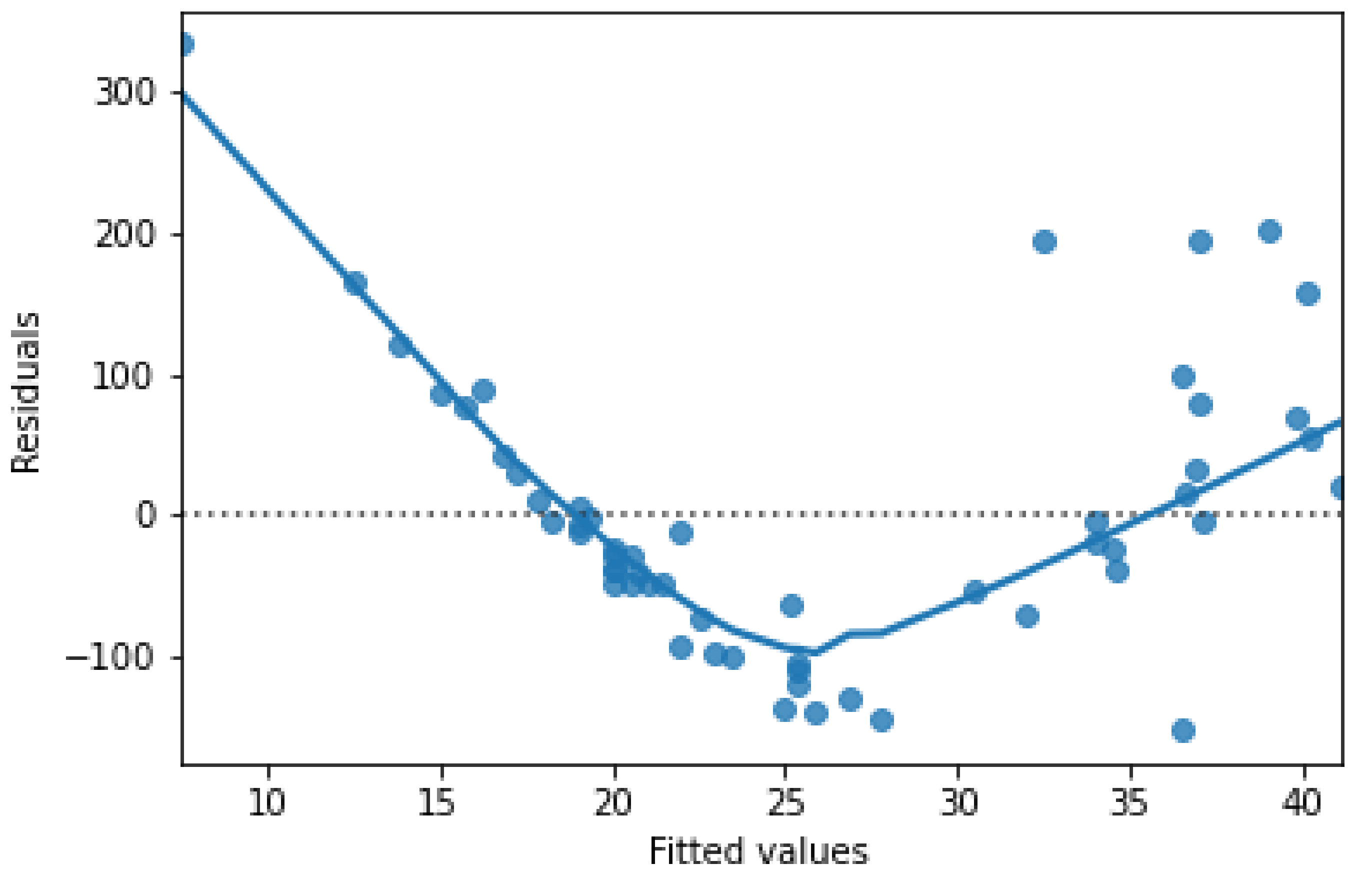


Residuals vs. fitted

Bream

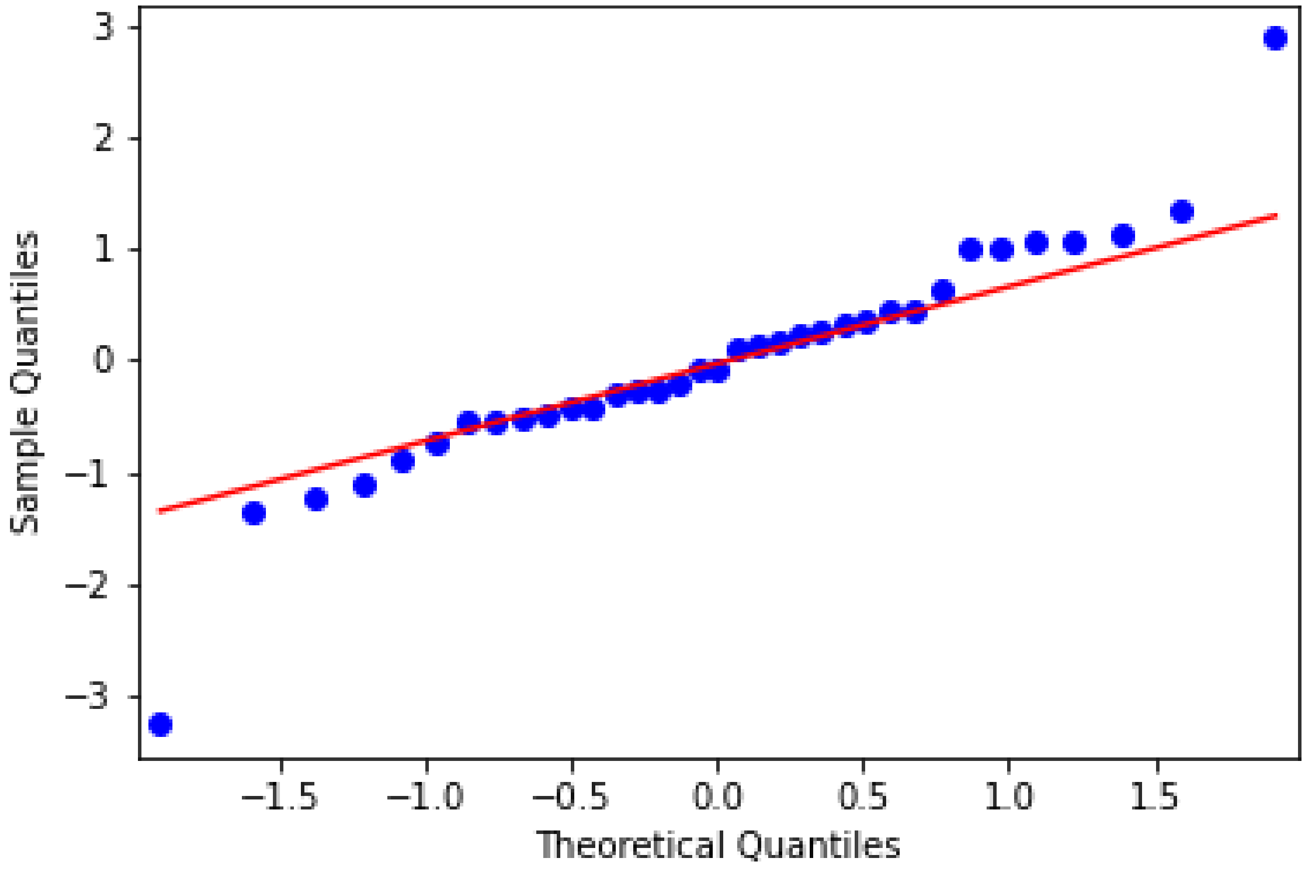


Perch

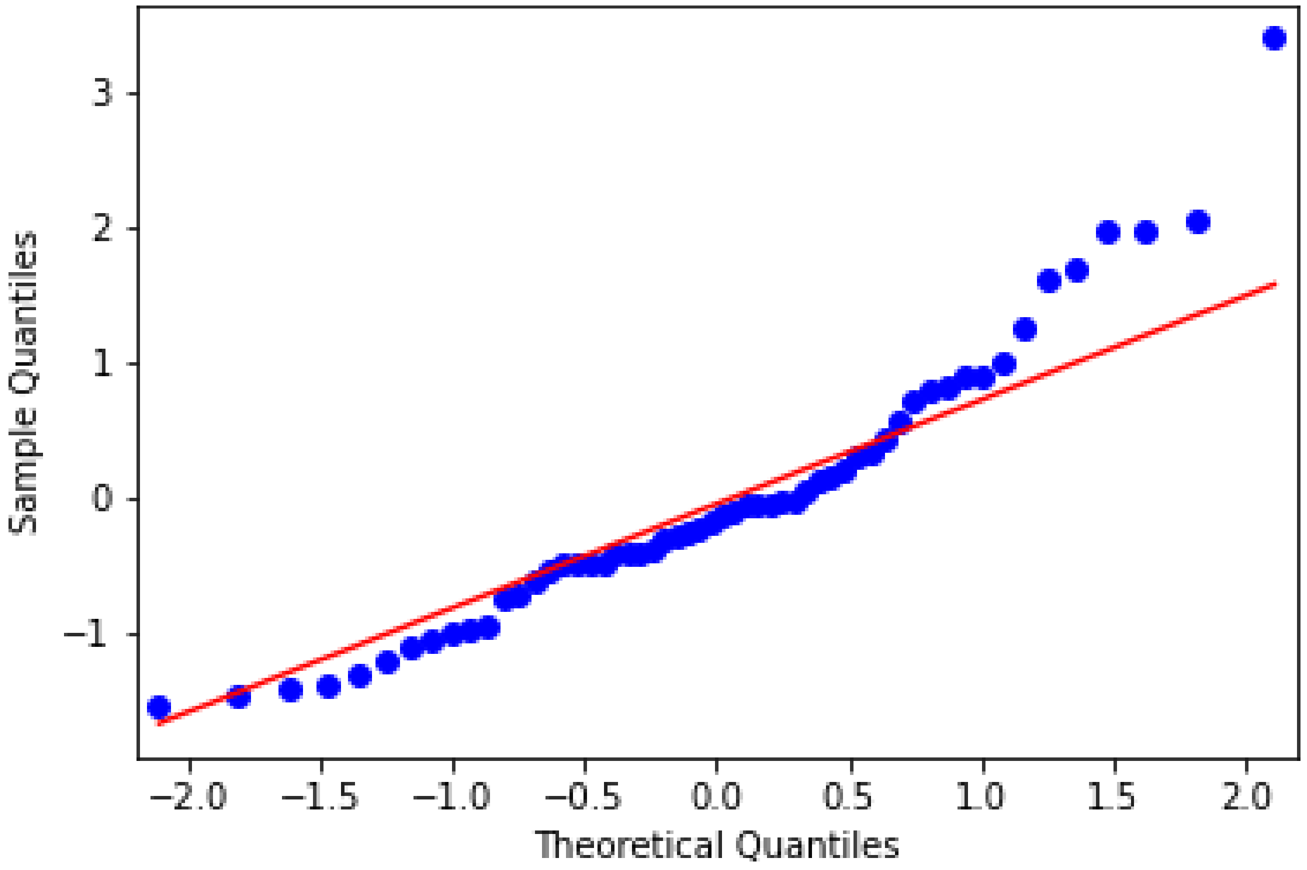


Q-Q plot

Bream

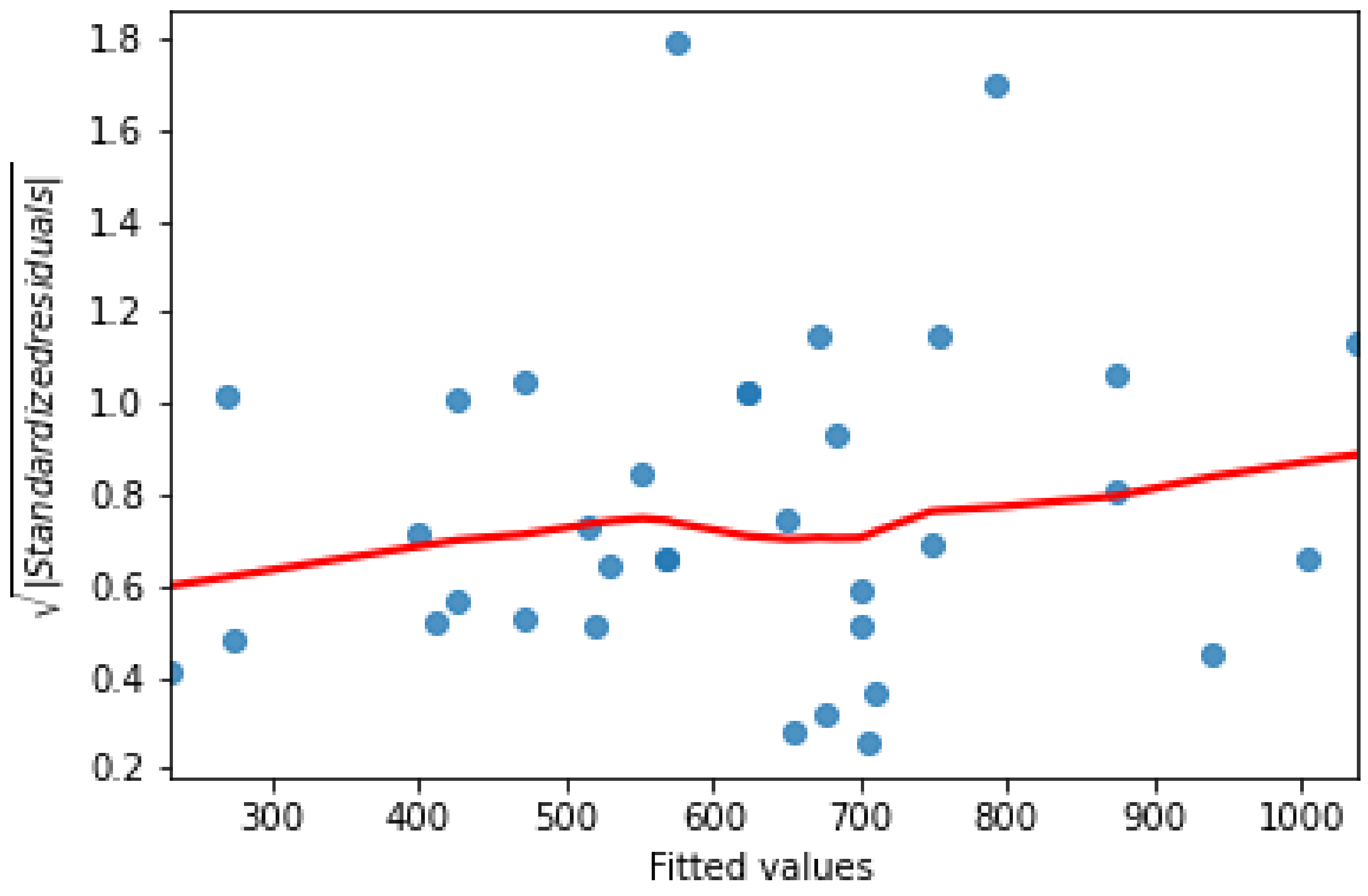


Perch

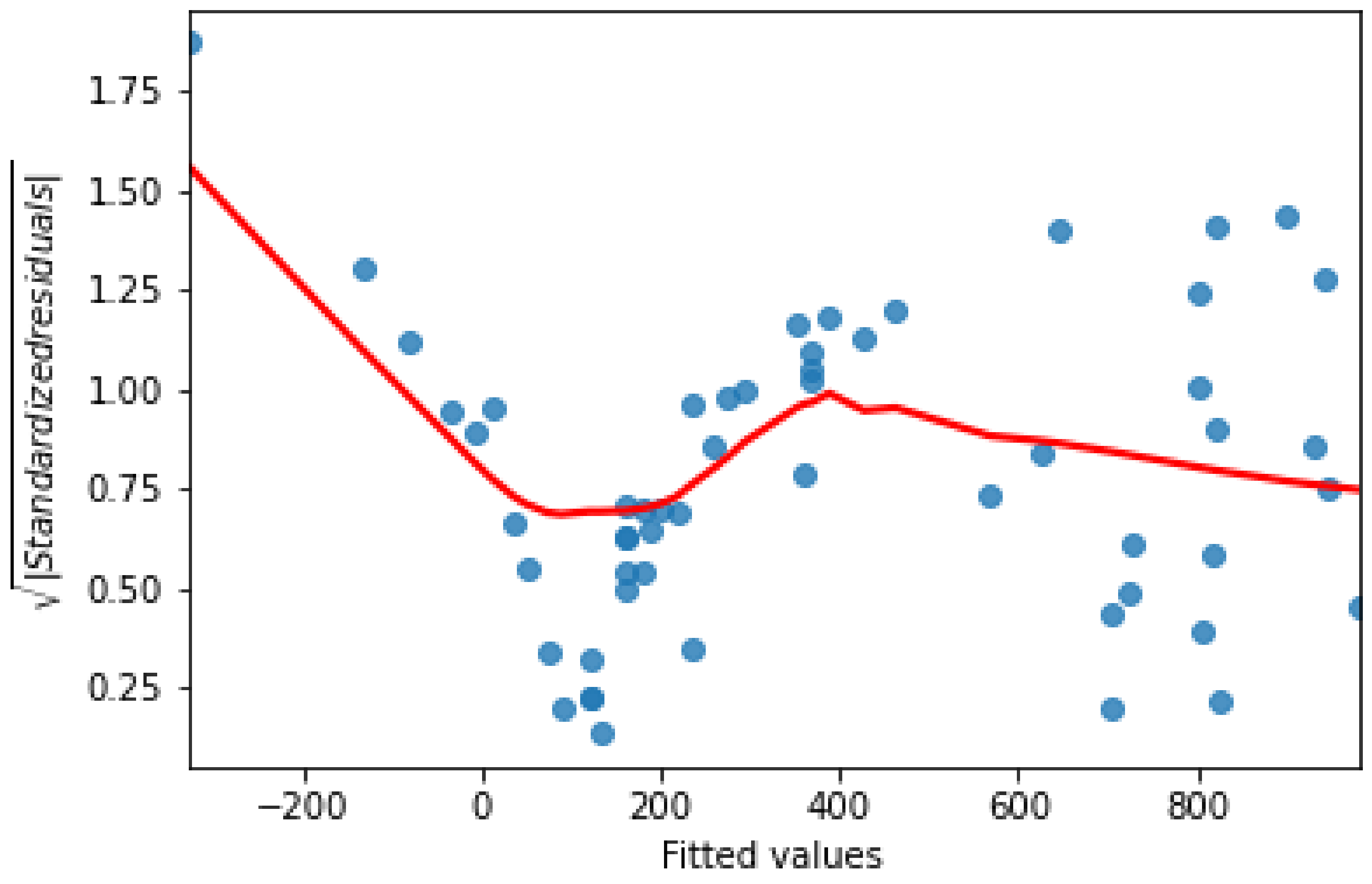


Scale-location plot

Bream

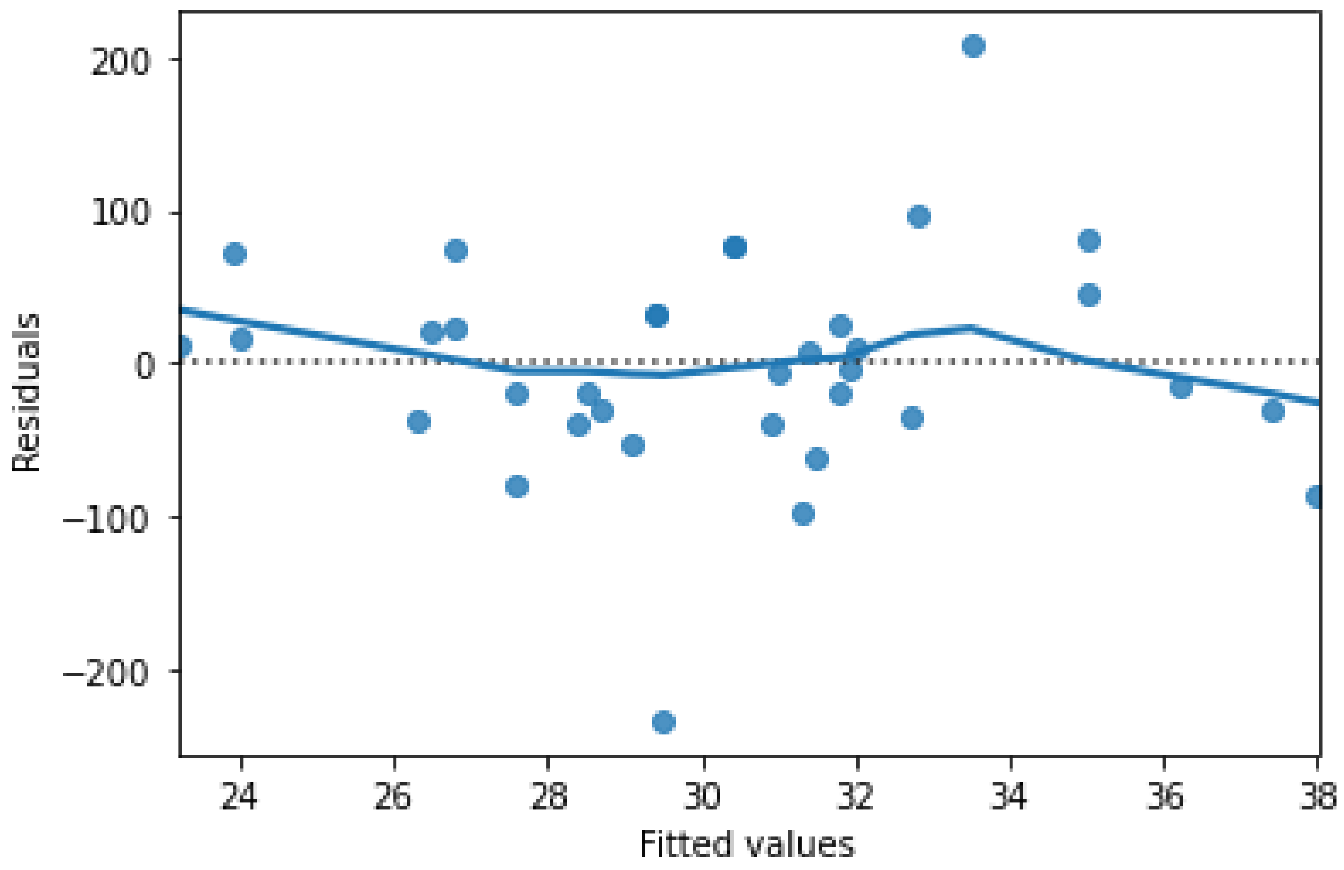


Perch



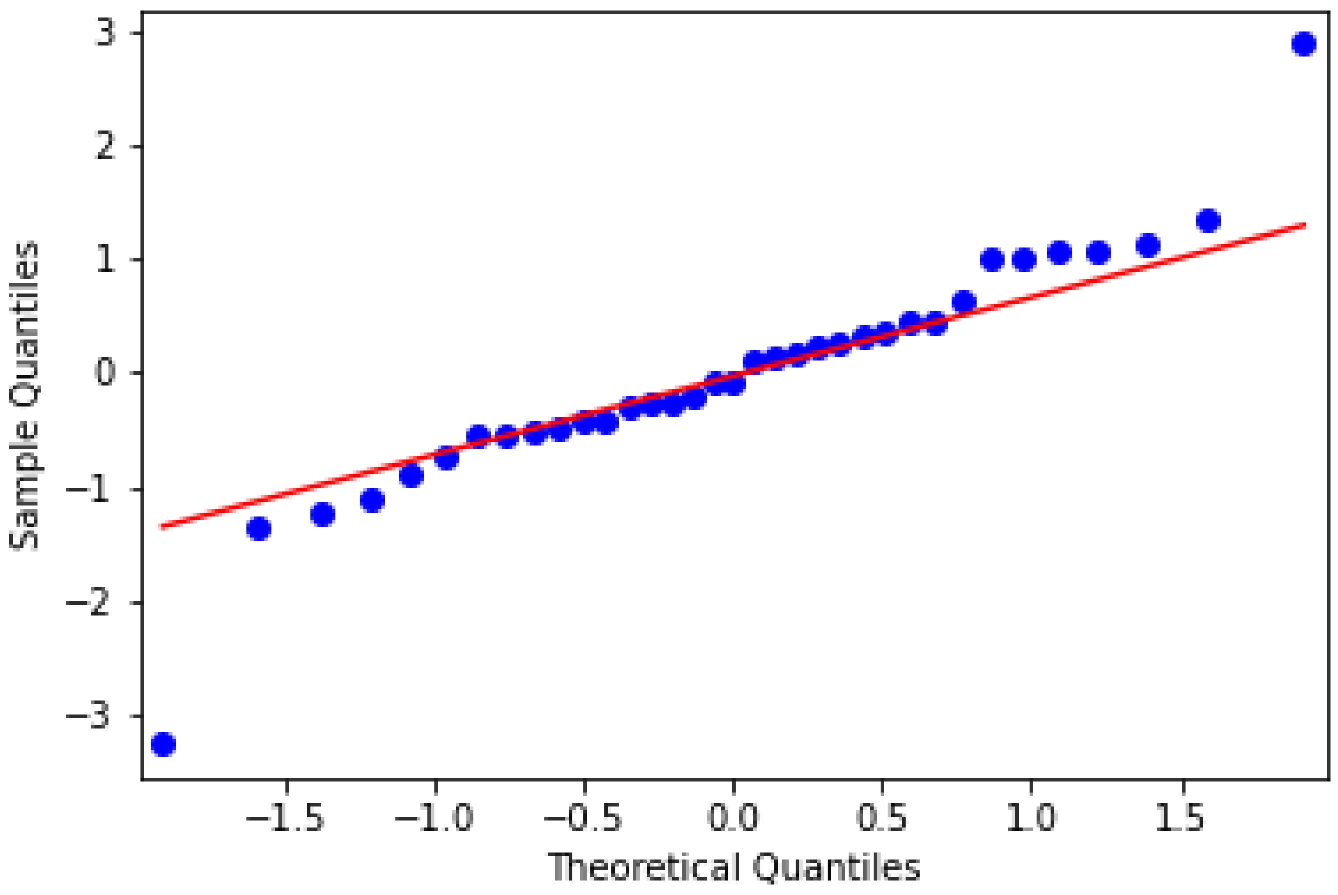
residplot()

```
sns.residplot(x="length_cm", y="mass_g", data=bream, lowess=True)
plt.xlabel("Fitted values")
plt.ylabel("Residuals")
```



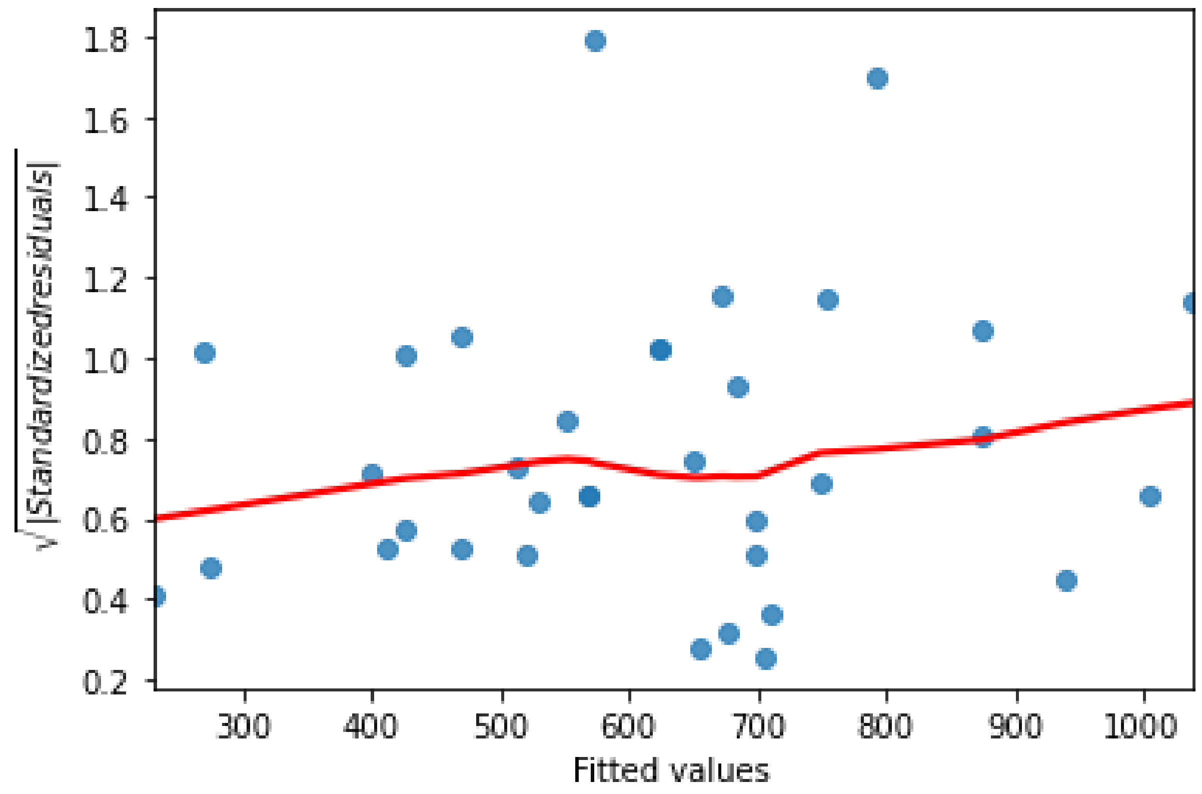
qqplot()

```
from statsmodels.api import qqplot
qqplot(data=mdl_bream.resid, fit=True, line="45")
```



Scale-location plot

```
model_norm_residuals_bream = mdl_bream.get_influence().resid_studentized_internal
model_norm_residuals_abs_sqrt_bream = np.sqrt(np.abs(model_norm_residuals_bream))
sns.regplot(x=mdl_bream.fittedvalues, y=model_norm_residuals_abs_sqrt_bream, ci=None, lowess=True)
plt.xlabel("Fitted values")
plt.ylabel("Sqrt of abs val of stdized residuals")
```



Let's practice!

Outliers, leverage, and influence

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

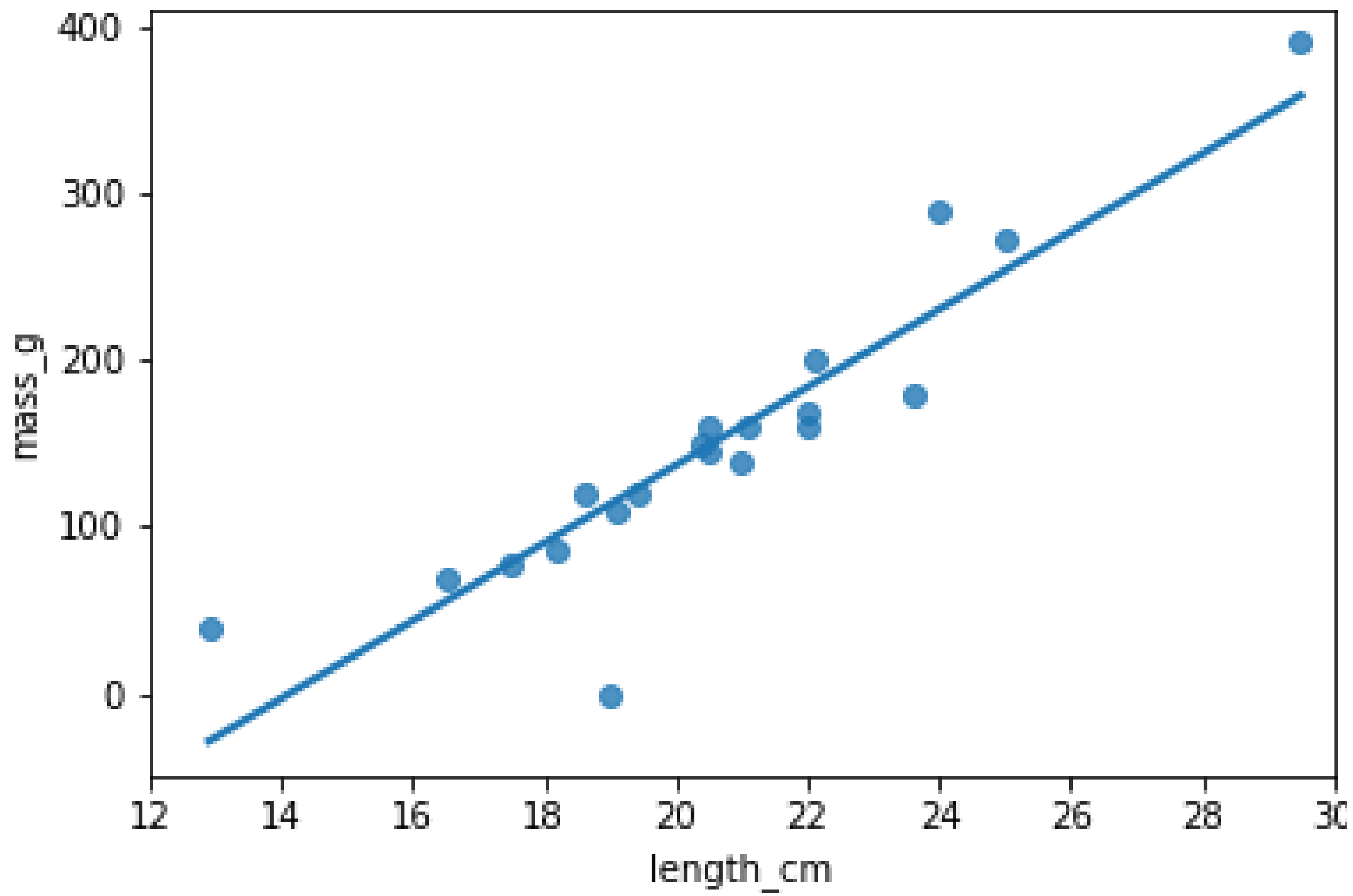
Roach dataset

```
roach = fish[fish['species'] == "Roach"]
print(roach.head())
```

```
   species  mass_g  length_cm
35   Roach    40.0       12.9
36   Roach    69.0       16.5
37   Roach    78.0       17.5
38   Roach    87.0       18.2
39   Roach   120.0       18.6
```

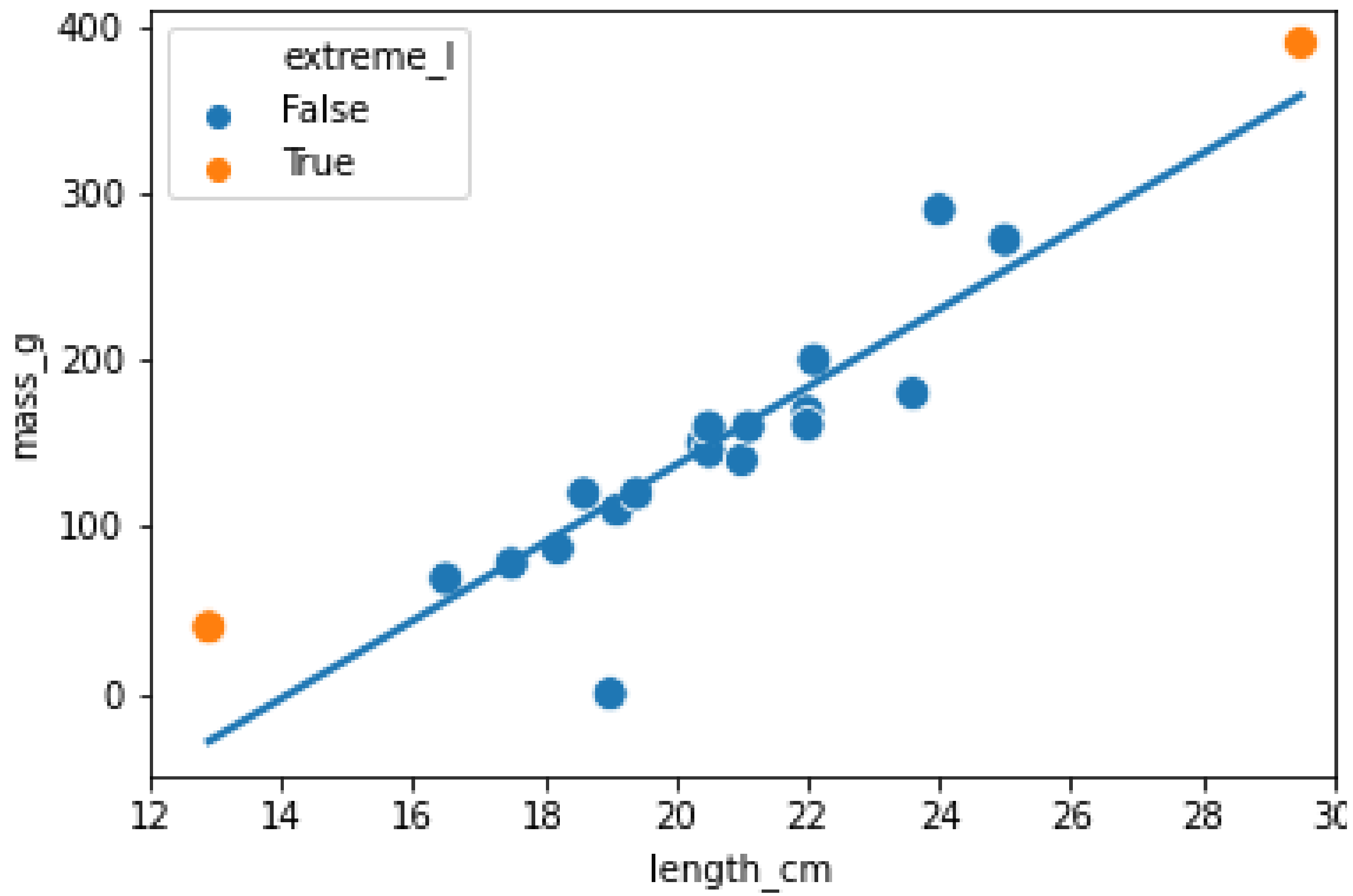
Which points are outliers?

```
sns.regplot(x="length_cm",
            y="mass_g",
            data=roach,
            ci=None)
plt.show()
```



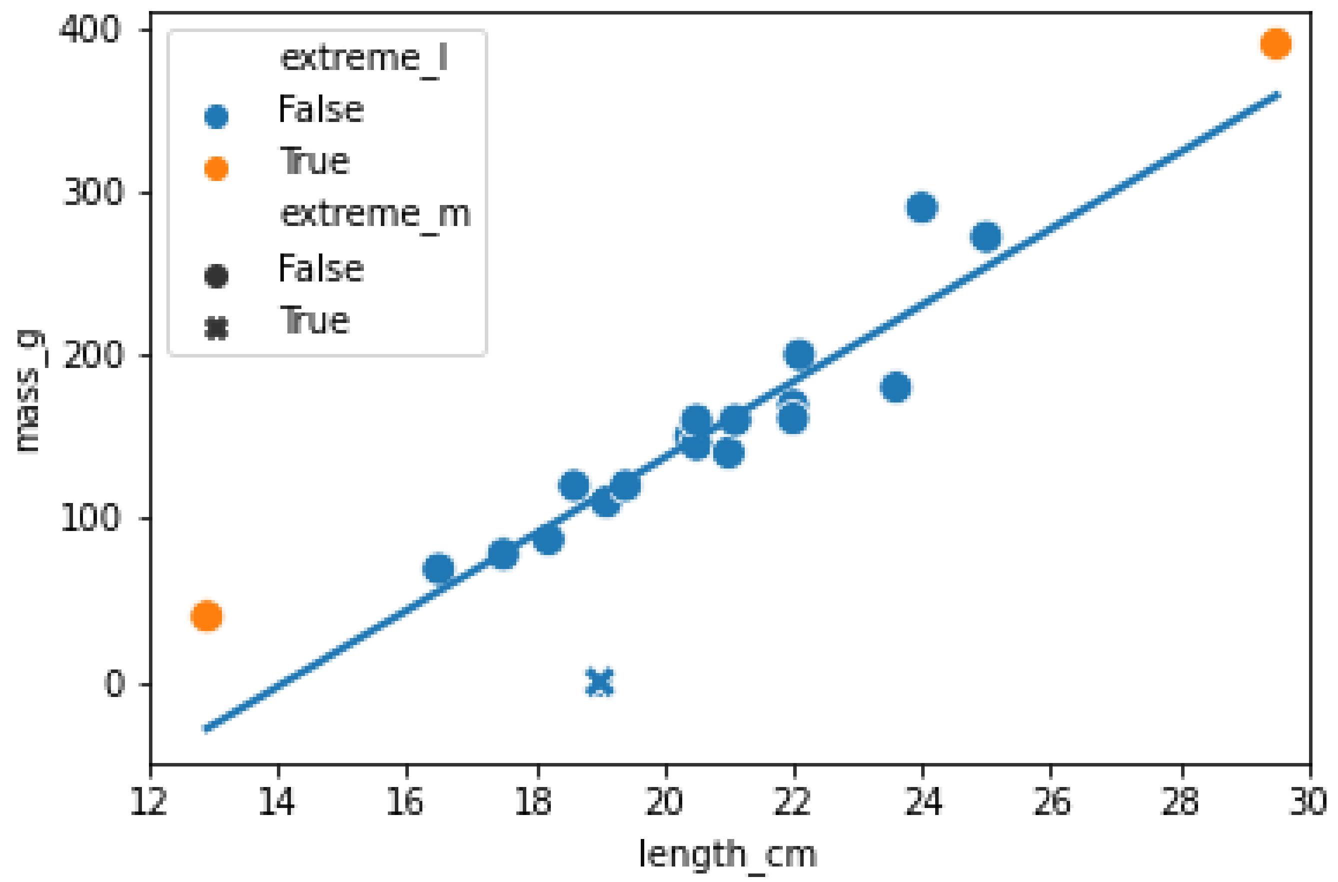
Extreme explanatory values

```
roach["extreme_l"] = ((roach["length_cm"] < 15) |
                      (roach["length_cm"] > 26))

fig = plt.figure()
sns.regplot(x="length_cm",
            y="mass_g",
            data=roach,
            ci=None)

sns.scatterplot(x="length_cm",
                y="mass_g",
                hue="extreme_l",
                data=roach)
```



Response values away from the regression line

```
roach["extreme_m"] = roach["mass_g"] < 1

fig = plt.figure()
sns.regplot(x="length_cm",
            y="mass_g",
            data=roach,
            ci=None)

sns.scatterplot(x="length_cm",
                y="mass_g",
                hue="extreme_l",
                style="extreme_m",
                data=roach)
```



Leverage and influence

Leverage is a measure of how extreme the explanatory variable values are.

Influence measures how much the model would change if you left the observation out of the dataset when modeling.

.get_influence() and .summary_frame()

```
mdl_roach = ols("mass_g ~ length_cm", data=roach).fit()
summary_roach = mdl_roach.get_influence().summary_frame()
roach["leverage"] = summary_roach["hat_diag"]

print(roach.head())
```

```
   species  mass_g  length_cm  leverage
35   Roach    40.0       12.9  0.313729
36   Roach    69.0       16.5  0.125538
37   Roach    78.0       17.5  0.093487
38   Roach    87.0       18.2  0.076283
39   Roach   120.0       18.6  0.068387
```

Cook's distance

Cook's distance is the most common measure of influence.

```
roach["cooks_dist"] = summary_roach["cooks_d"]
print(roach.head())
```

```
   species  mass_g  length_cm  leverage  cooks_dist
35   Roach    40.0       12.9  0.313729    1.074015
36   Roach    69.0       16.5  0.125538    0.010429
37   Roach    78.0       17.5  0.093487    0.000020
38   Roach    87.0       18.2  0.076283    0.001980
39   Roach   120.0       18.6  0.068387    0.006610
```

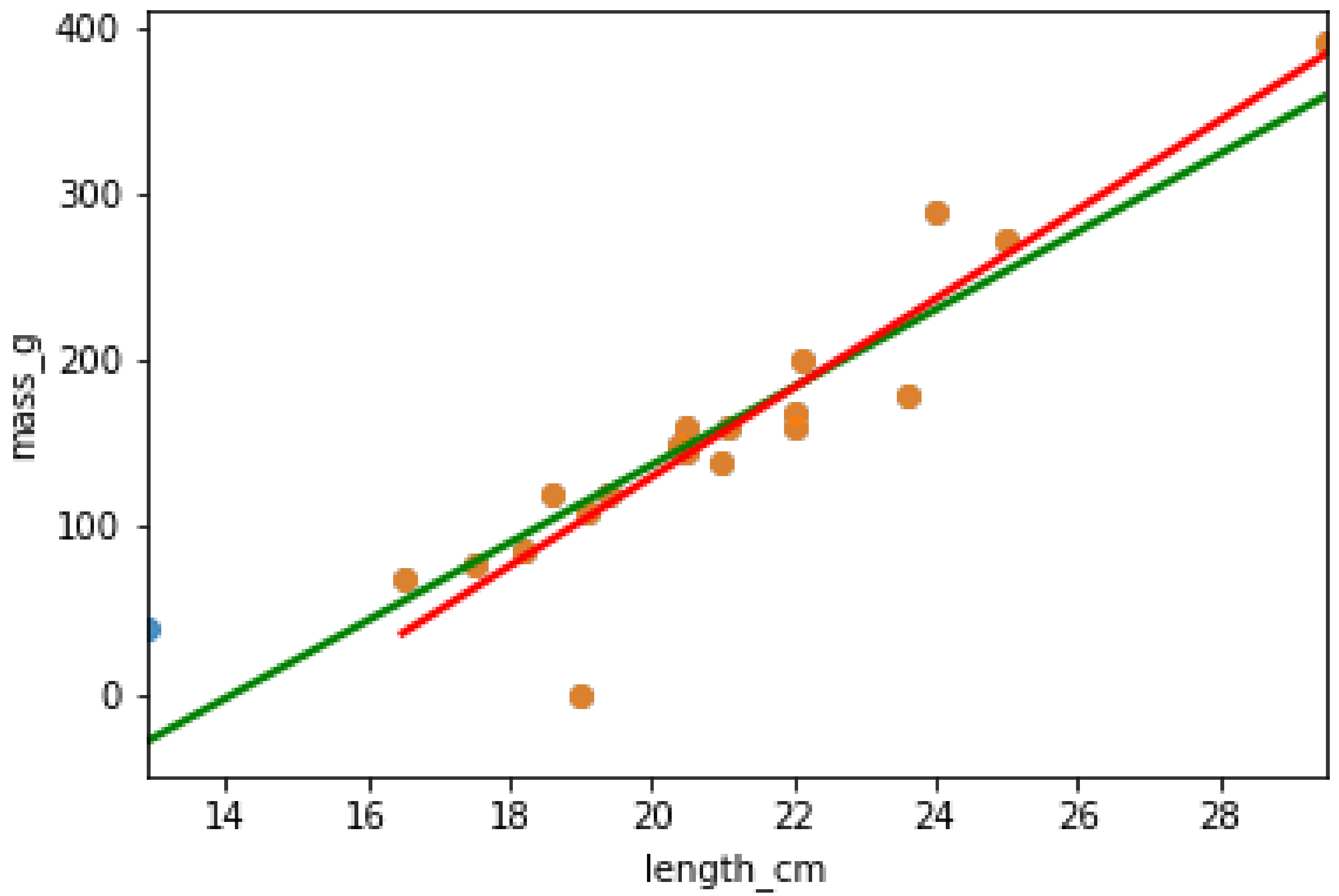
Most influential roaches

```
print(roach.sort_values("cooks_dist", ascending = False))
```

```
   species  mass_g  length_cm  leverage  cooks_dist
35   Roach    40.0       12.9  0.313729    1.074015 # really short roach
54   Roach   390.0       29.5  0.394740    0.365782 # really long roach
40   Roach     0.0       19.0  0.061897    0.311852 # roach with zero mass
52   Roach   290.0       24.0  0.099488    0.150064
51   Roach   180.0       23.6  0.088391    0.061209
..     ...     ...        ...       ...         ...
43   Roach   150.0       20.4  0.050264    0.000257
44   Roach   145.0       20.5  0.050092    0.000256
42   Roach   120.0       19.4  0.056815    0.000199
47   Roach   160.0       21.1  0.050910    0.000137
37   Roach    78.0       17.5  0.093487    0.000020
```

Removing the most influential roach

```
roach_not_short = roach[roach["length_cm"] != 12.9]

sns.regplot(x="length_cm",
            y="mass_g",
            data=roach,
            ci=None,
            line_kws={"color": "green"})

sns.regplot(x="length_cm",
            y="mass_g",
            data=roach_not_short,
            ci=None,
            line_kws={"color": "red"})
```



Let's practice!

Why you need logistic regression

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

Bank churn dataset

has_churned	time_since_first_purchase	time_since_last_purchase
0	0.3993247	-0.5158691
1	-0.4297957	0.6780654
0	3.7383122	0.4082544
0	0.6032289	-0.6990435
...	...	...
response	*length of relationship*	*recency of activity*

[1] https://www.rdocumentation.org/packages/bayesQR/topics/Churn

Churn vs. recency: a linear model

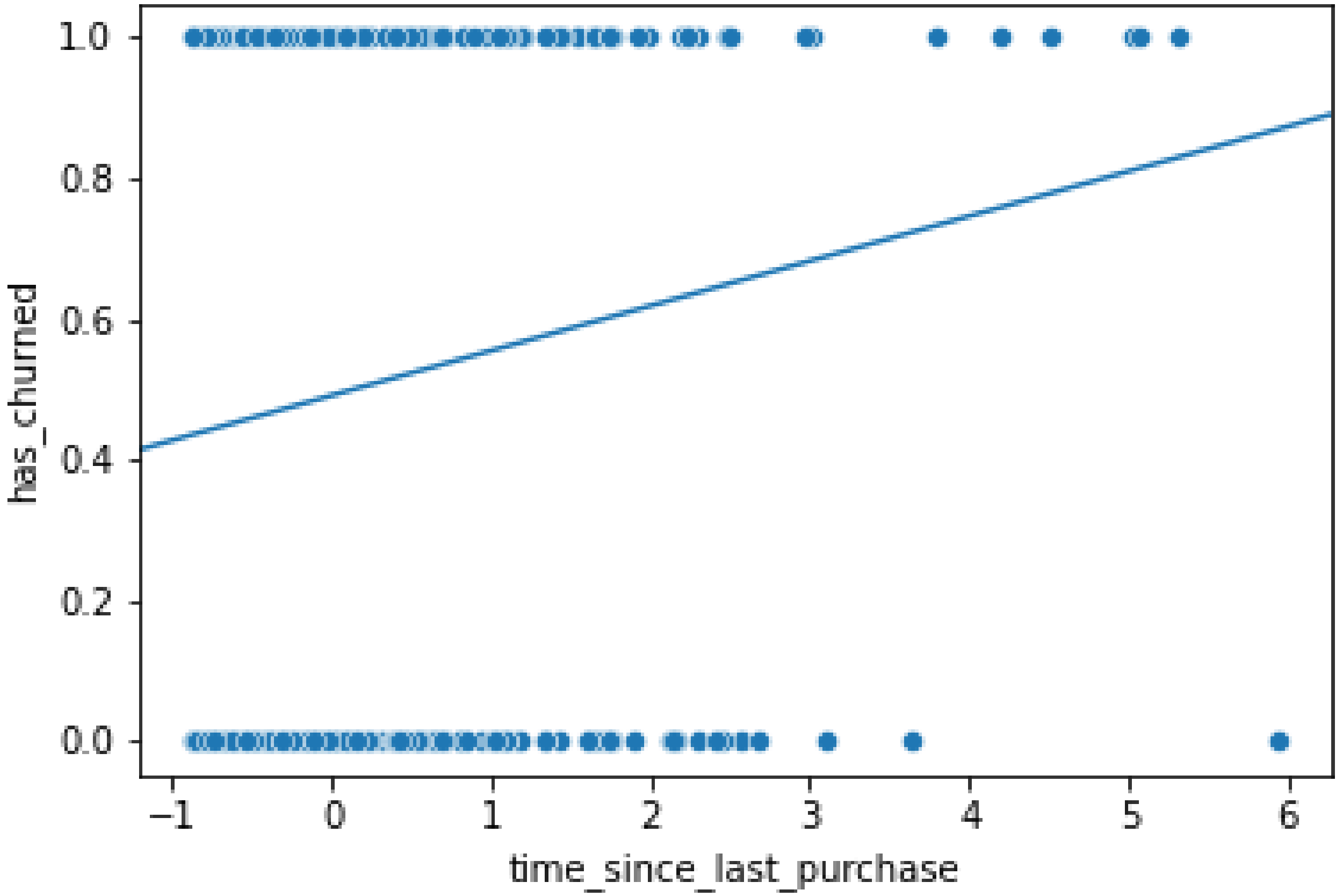
```
mdl_churn_vs_recency_lm = ols("has_churned ~ time_since_last_purchase",
                              data=churn).fit()

print(mdl_churn_vs_recency_lm.params)
```

```
Intercept                   0.490780
time_since_last_purchase    0.063783
dtype: float64
```

```
intercept, slope = mdl_churn_vs_recency_lm.params
```

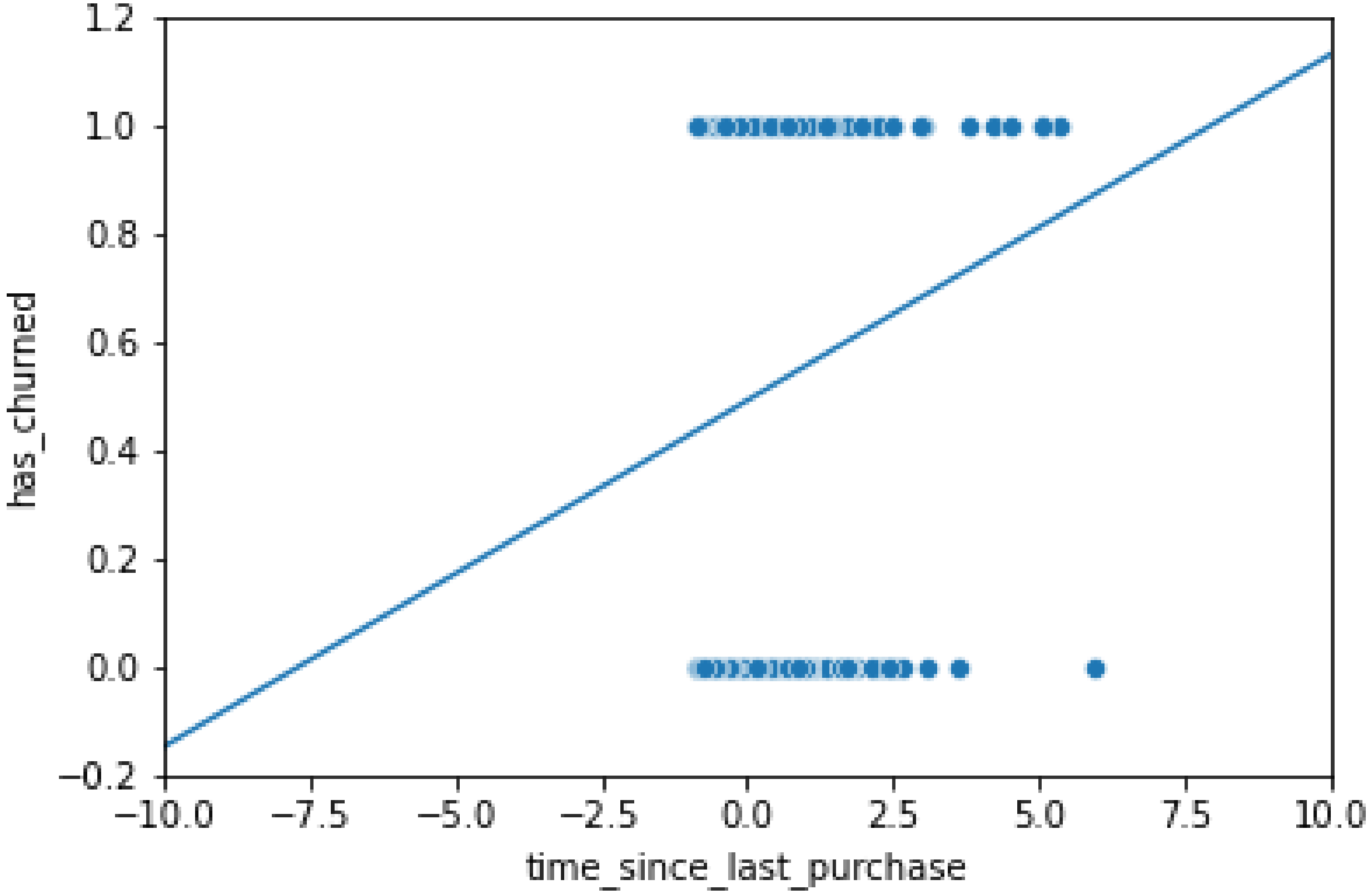
Visualizing the linear model

```
sns.scatterplot(x="time_since_last_purchase",
                y="has_churned",
                data=churn)

plt.axline(xy1=(0, intercept),
           slope=slope)

plt.show()
```



Zooming out

```
sns.scatterplot(x="time_since_last_purchase",
                y="has_churned",
                data=churn)

plt.axline(xy1=(0,intercept),
           slope=slope)

plt.xlim(-10, 10)
plt.ylim(-0.2, 1.2)
plt.show()
```



What is logistic regression?

- Another type of generalized linear model.
- Used when the response variable is logical.
- The responses follow logistic (S-shaped) curve.

Logistic regression using logit()

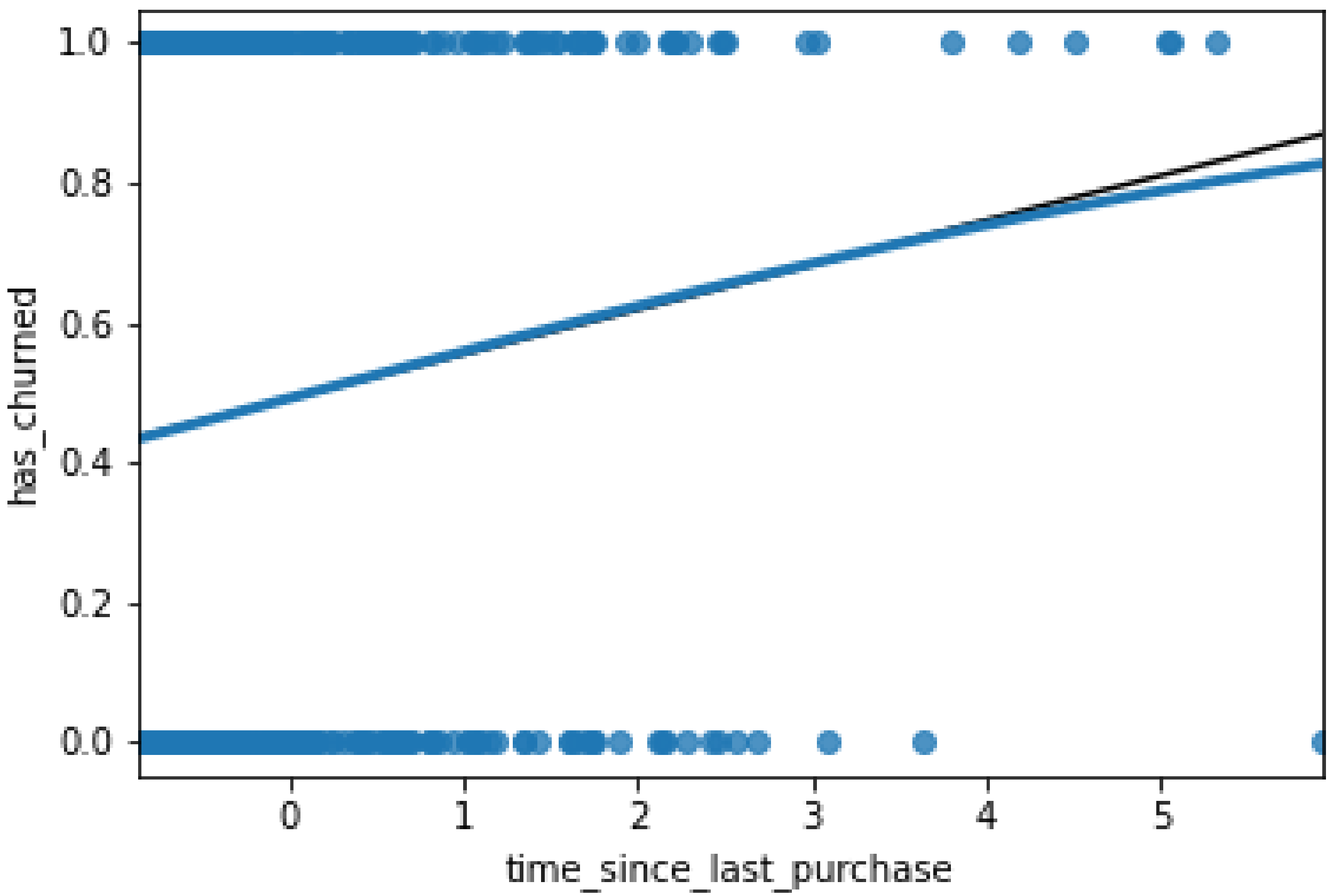
```
from statsmodels.formula.api import logit
mdl_churn_vs_recency_logit = logit("has_churned ~ time_since_last_purchase",
                                   data=churn).fit()
print(mdl_churn_vs_recency_logit.params)
```

```
Intercept                  -0.035019
time_since_last_purchase    0.269215
dtype: float64
```

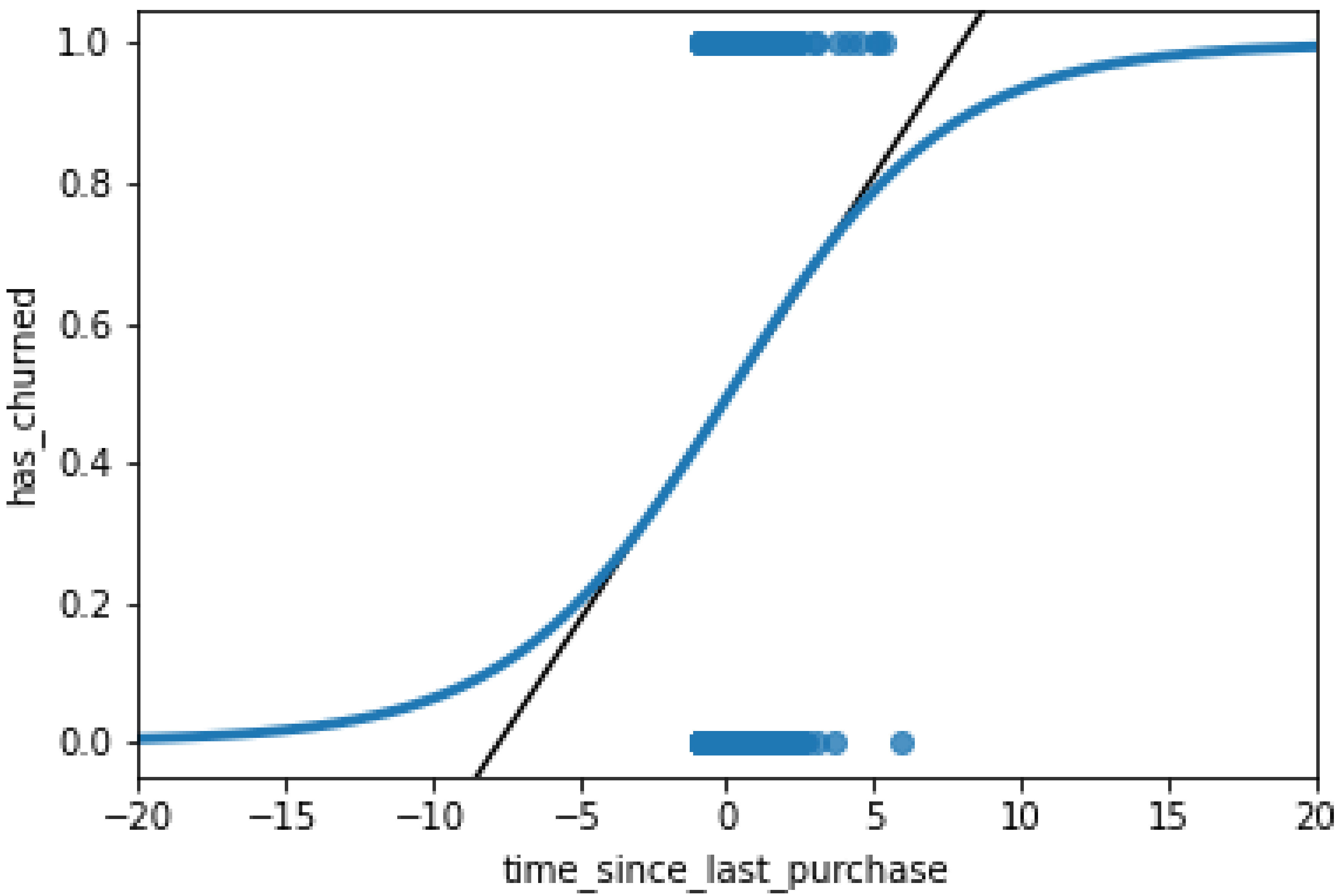
Visualizing the logistic model

```
sns.regplot(x="time_since_last_purchase",
            y="has_churned",
            data=churn,
            ci=None,
            logistic=True)
```

```
plt.axline(xy1=(0,intercept),
           slope=slope,
           color="black")

plt.show()
```



Zooming out



1.0
0.8
0.6
0.4
0.2
0.0
has_churned
−20
−15
−10
−5
0
5
10
15
20
time_since_last_purchase

Let's practice!

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

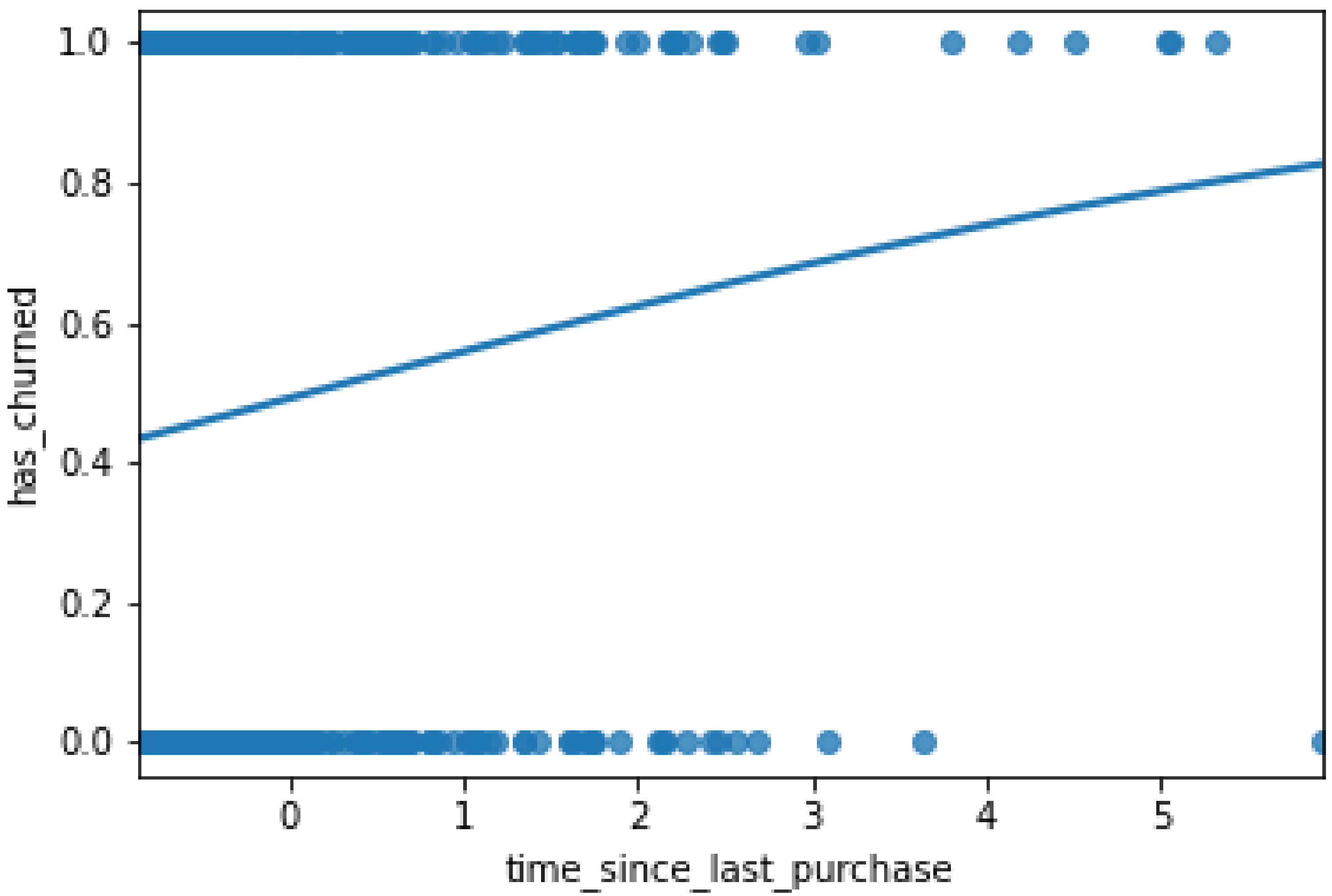
Predictions and odds ratios

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

The regplot() predictions

```
sns.regplot(x="time_since_last_purchase",
            y="has_churned",
            data=churn,
            ci=None,
            logistic=True)

plt.show()
```

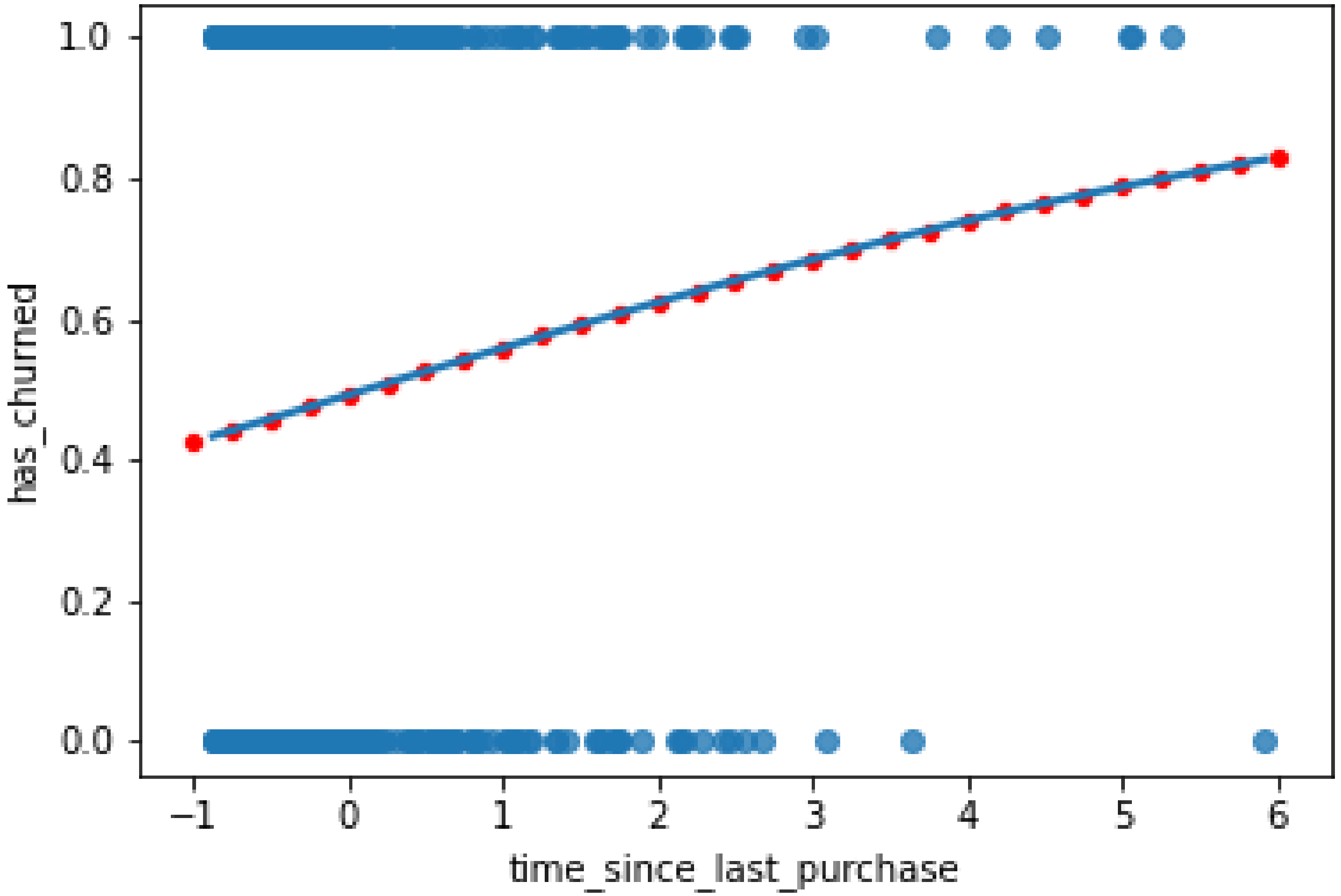


Making predictions

```
mdl_recency = logit("has_churned ~ time_since_last_purchase",
                    data = churn).fit()

explanatory_data = pd.DataFrame(
  {"time_since_last_purchase": np.arange(-1, 6.25, 0.25)})

prediction_data = explanatory_data.assign(
    has_churned = mdl_recency.predict(explanatory_data))
```

Adding point predictions

```
sns.regplot(x="time_since_last_purchase",
            y="has_churned",
            data=churn,
            ci=None,
            logistic=True)

sns.scatterplot(x="time_since_last_purchase",
                y="has_churned",
                data=prediction_data,
                color="red")

plt.show()
```

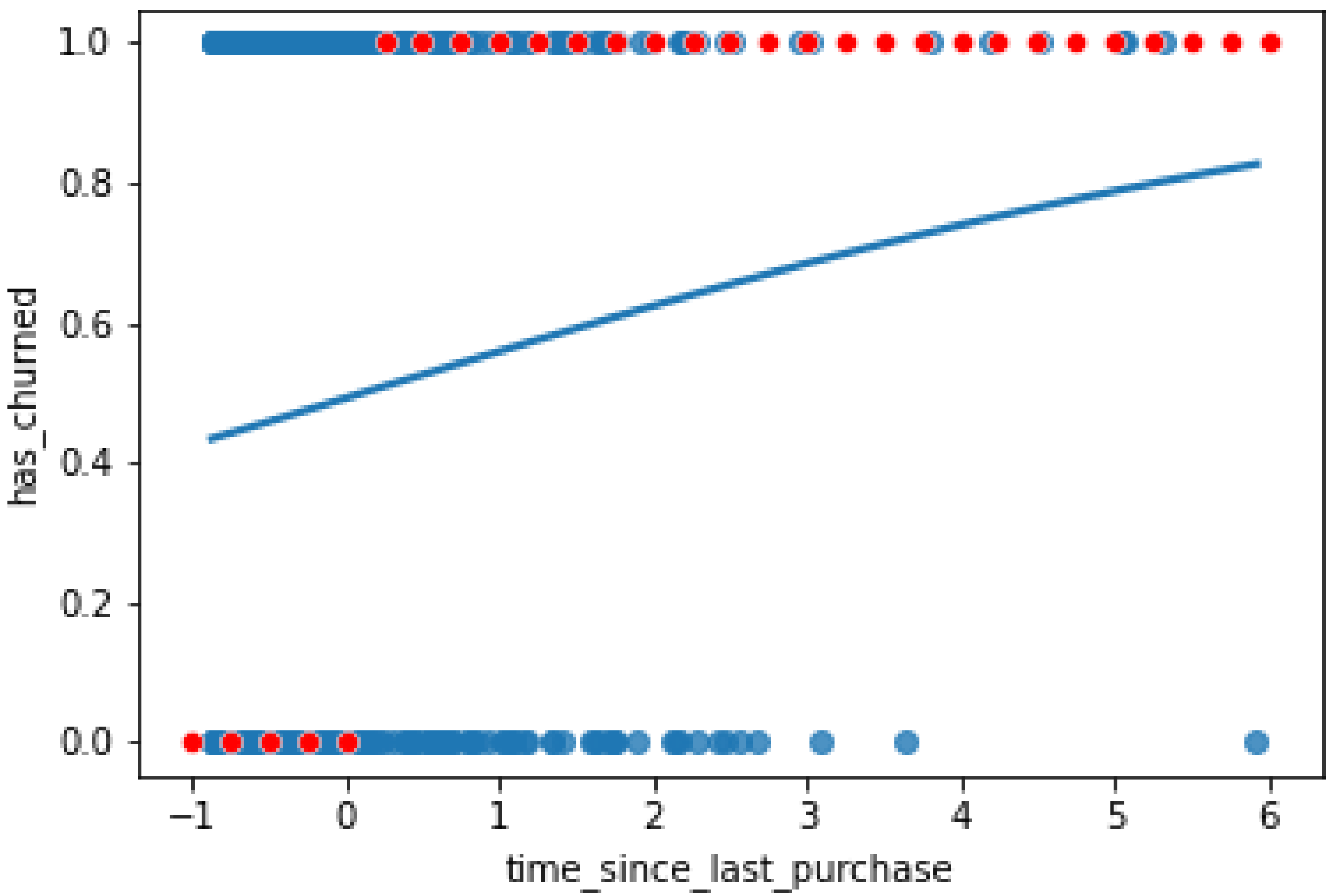


Getting the most likely outcome

```
prediction_data = explanatory_data.assign(
    has_churned = mdl_recency.predict(explanatory_data))
prediction_data["most_likely_outcome"] = np.round(prediction_data["has_churned"])
```

Visualizing most likely outcome

```
sns.regplot(x="time_since_last_purchase",
            y="has_churned",
            data=churn,
            ci=None,
            logistic=True)

sns.scatterplot(x="time_since_last_purchase",
                y="most_likely_outcome",
                data=prediction_data,
                color="red")

plt.show()
```

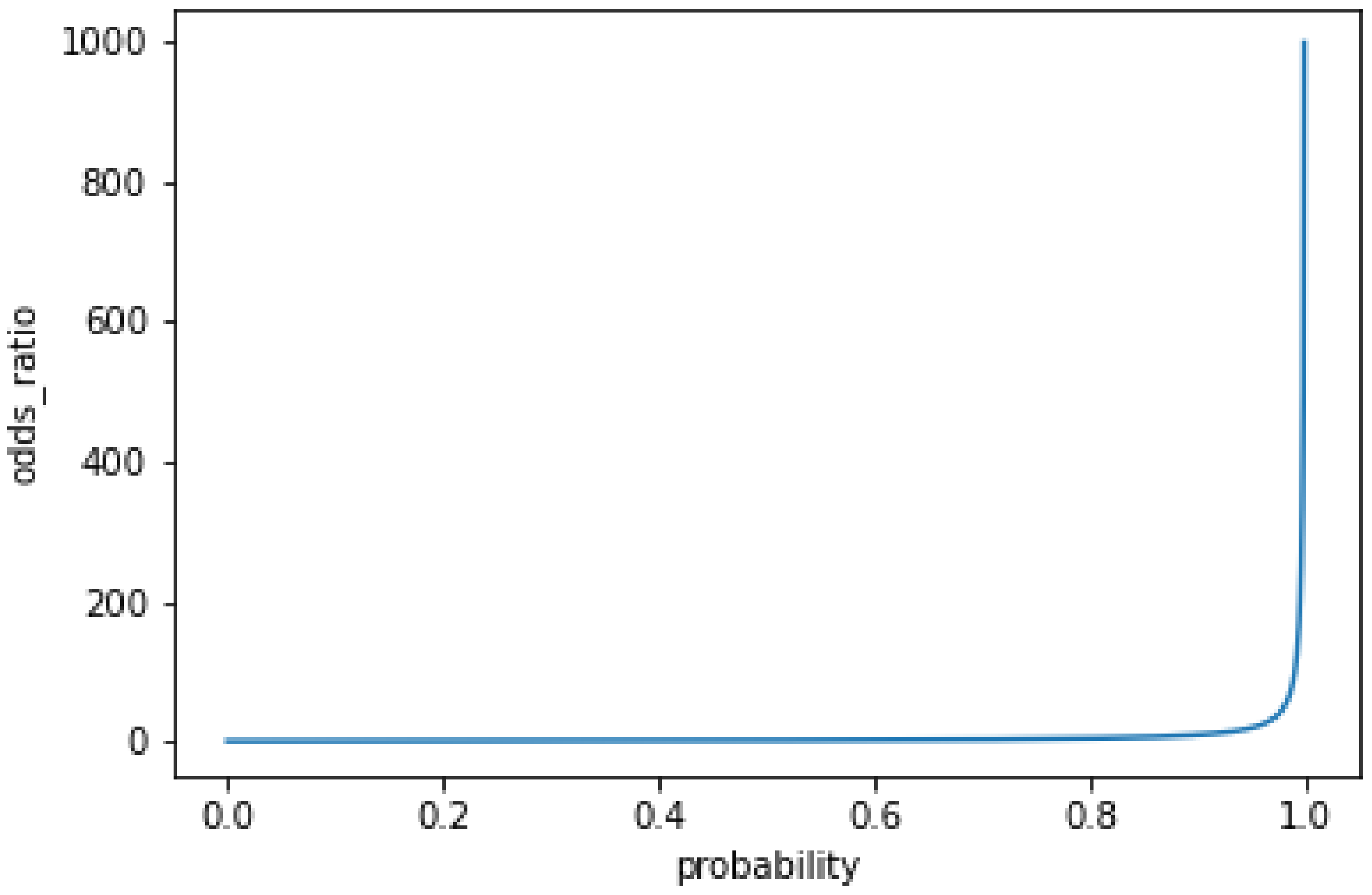


Odds ratios

Odds ratio is the probability of something happening divided by the probability that it doesn't.

$$\text{odds\_ratio} = \frac{\text{probability}}{(1 - \text{probability})}$$

$$\text{odds\_ratio} = \frac{0.25}{(1 - 0.25)} = \frac{1}{3}$$

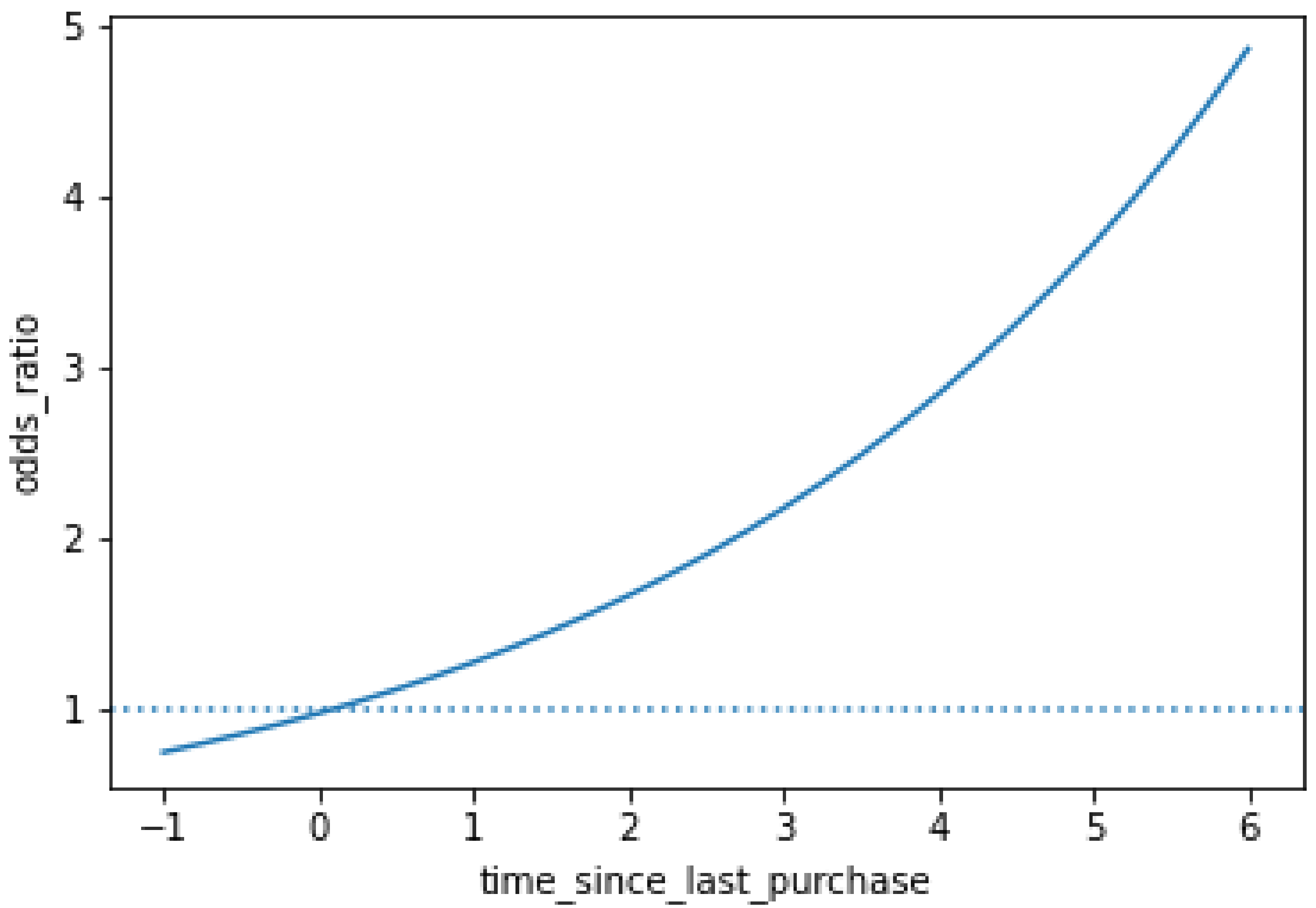


Calculating odds ratio

```
prediction_data["odds_ratio"] = prediction_data["has_churned"] /
                                (1 - prediction_data["has_churned"])
```

Visualizing odds ratio

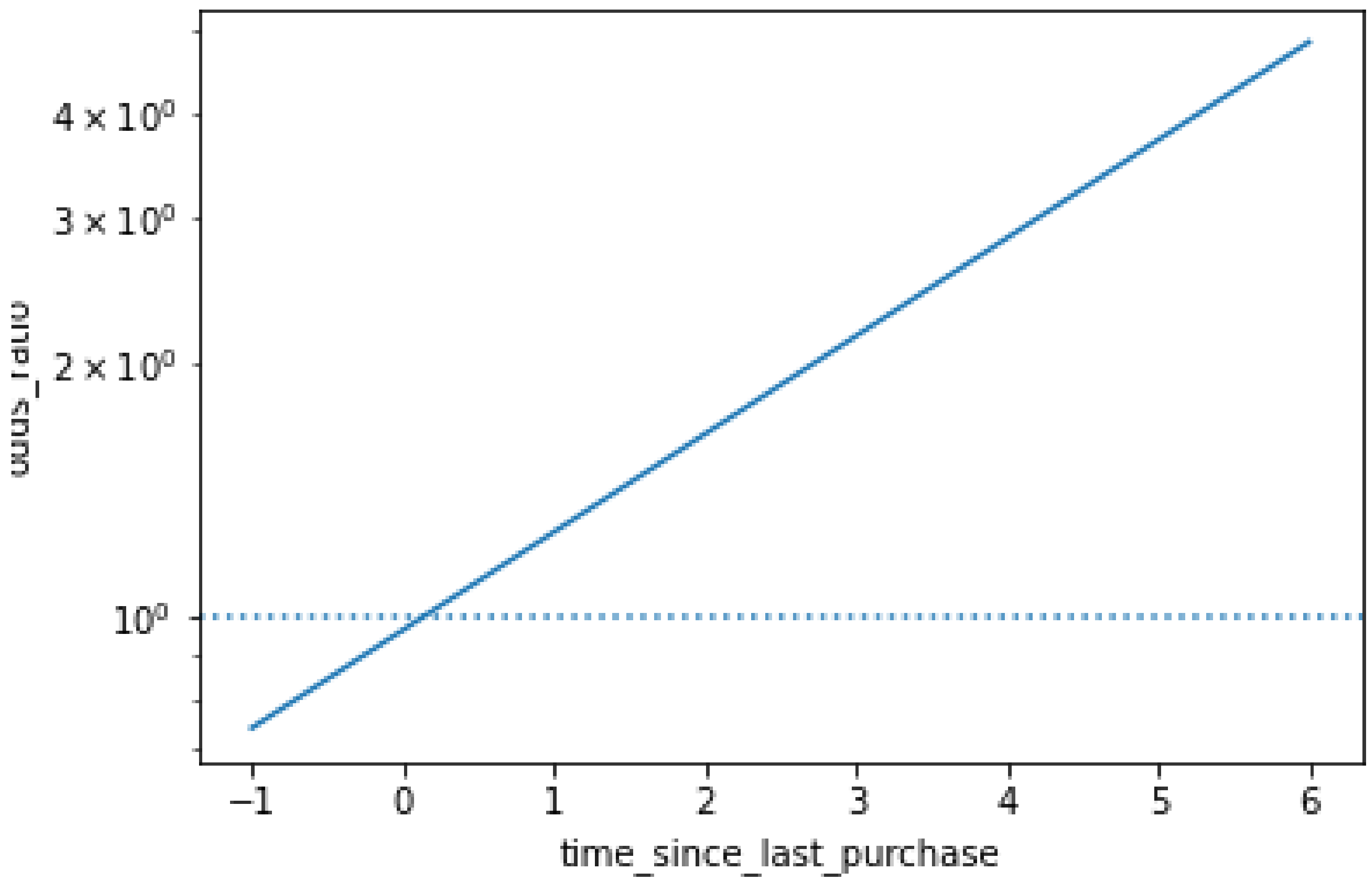
```
sns.lineplot(x="time_since_last_purchase",
             y="odds_ratio",
             data=prediction_data)

plt.axhline(y=1,
            linestyle="dotted")

plt.show()
```



Visualizing log odds ratio

```
sns.lineplot(x="time_since_last_purchase",
             y="odds_ratio",
             data=prediction_data)


plt.axhline(y=1,
            linestyle="dotted")
```

```
plt.yscale("log")


plt.show()
```



Calculating log odds ratio

```
prediction_data["log_odds_ratio"] = np.log(prediction_data["odds_ratio"])
```

All predictions together

time_since_last_prchs	has_churned	most_likely_rspns	odds_ratio	log_odds_ratio
0	0.491	0	0.966	-0.035
2	0.623	1	1.654	0.503
4	0.739	1	2.834	1.042
6	0.829	1	4.856	1.580
...	...	...	...	...

Comparing scales

Scale	Are values easy to interpret?	Are changes easy to interpret?	Is precise?
Probability	✔	✘	✔
Most likely outcome	✔✔	✔	✘
Odds ratio	✔	✘	✔
Log odds ratio	✘	✔	✔

Let's practice!

Quantifying logistic regression fit

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

The four outcomes

	predicted false	predicted true
actual false	correct	false positive
actual true	false negative	correct

Confusion matrix: counts of outcomes

```
actual_response = churn["has_churned"]
```

```
predicted_response = np.round(mdl_recency.predict())
```

```
outcomes = pd.DataFrame({"actual_response": actual_response,
                         "predicted_response": predicted_response})
```

```
print(outcomes.value_counts(sort=False))
```

```
actual_response  predicted_response
0                0.0                   141
                 1.0                    59
1                0.0                   111
                 1.0                    89
```

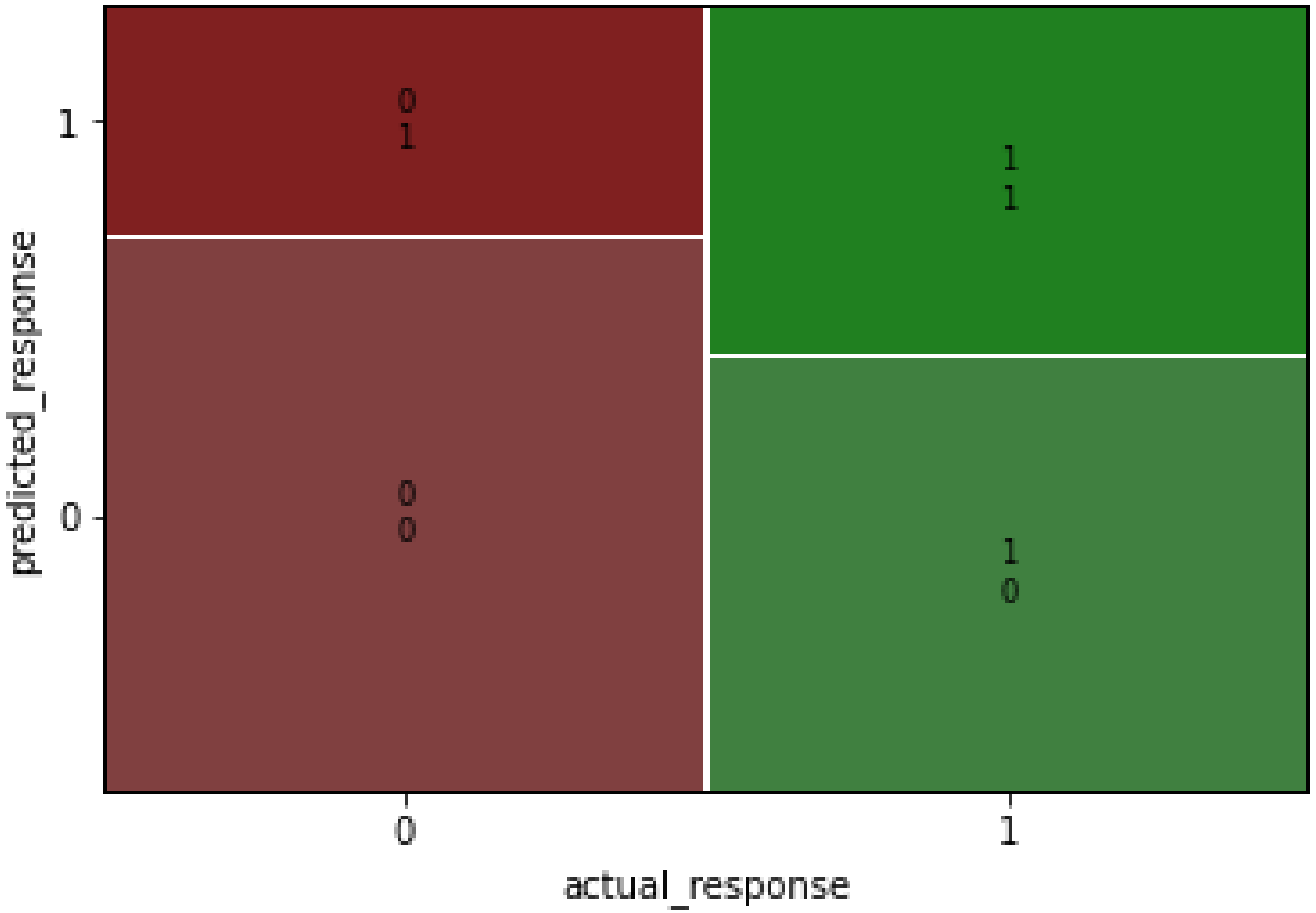
Visualizing the confusion matrix

```
conf_matrix = mdl_recency.pred_table()

print(conf_matrix)
```

```
[[141.          59.]
 [111.          89.]]
```

true negative	**false positive**
false negative	**true positive**

```
from statsmodels.graphics.mosaicplot
import mosaic

mosaic(conf_matrix)
```



Accuracy

Accuracy is the proportion of correct predictions.

$$\text{accuracy} = \frac{TN + TP}{TN + FN + FP + TP}$$

```
[[141.,  59.],
 [111.,  89.]]
```

```
TN = conf_matrix[0,0]
TP = conf_matrix[1,1]
FN = conf_matrix[1,0]
FP = conf_matrix[0,1]
```

```
acc = (TN + TP) / (TN + TP + FN + FP)
print(acc)
```

```
0.575
```

Sensitivity

Sensitivity is the proportion of true positives.

$$\text{sensitivity} = \frac{TP}{FN + TP}$$

```
[[141.,  59.],
 [111.,  89.]]
```

```
TN = conf_matrix[0,0]
TP = conf_matrix[1,1]
FN = conf_matrix[1,0]
FP = conf_matrix[0,1]
```

```
sens = TP / (FN + TP)
print(sens)
```

```
0.445
```

Specificity

Specificity is the proportion of true negatives.

$$\text{specificity} = \frac{TN}{TN + FP}$$

```
[[141.,  59.],
 [111.,  89.]]
```

```
TN = conf_matrix[0,0]
TP = conf_matrix[1,1]
FN = conf_matrix[1,0]
FP = conf_matrix[0,1]
```

```
spec = TN / (TN + FP)
print(spec)
```

```
0.705
```

Let's practice!

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Congratulations

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

Maarten Van den Broeck
Content Developer at DataCamp

You learned things

Chapter 1

- Fit a simple linear regression
- Interpret coefficients

Chapter 2

- Make predictions
- Regression to the mean
- Transforming variables

Chapter 3

- Quantifying model fit
- Outlier, leverage, and influence

Chapter 4

- Fit a simple logistic regression
- Make predictions
- Get performance from confusion matrix

Multiple explanatory variables

Intermediate Regression with statsmodels in Python

Unlocking advanced skills

- Generalized Linear Models in Python
- Introduction to Predictive Analytics in Python
- Linear Classifiers in Python

Happy learning!

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON