

Making predictions

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON



Maarten Van den Broeck

Content Developer at DataCamp

The fish dataset: bream

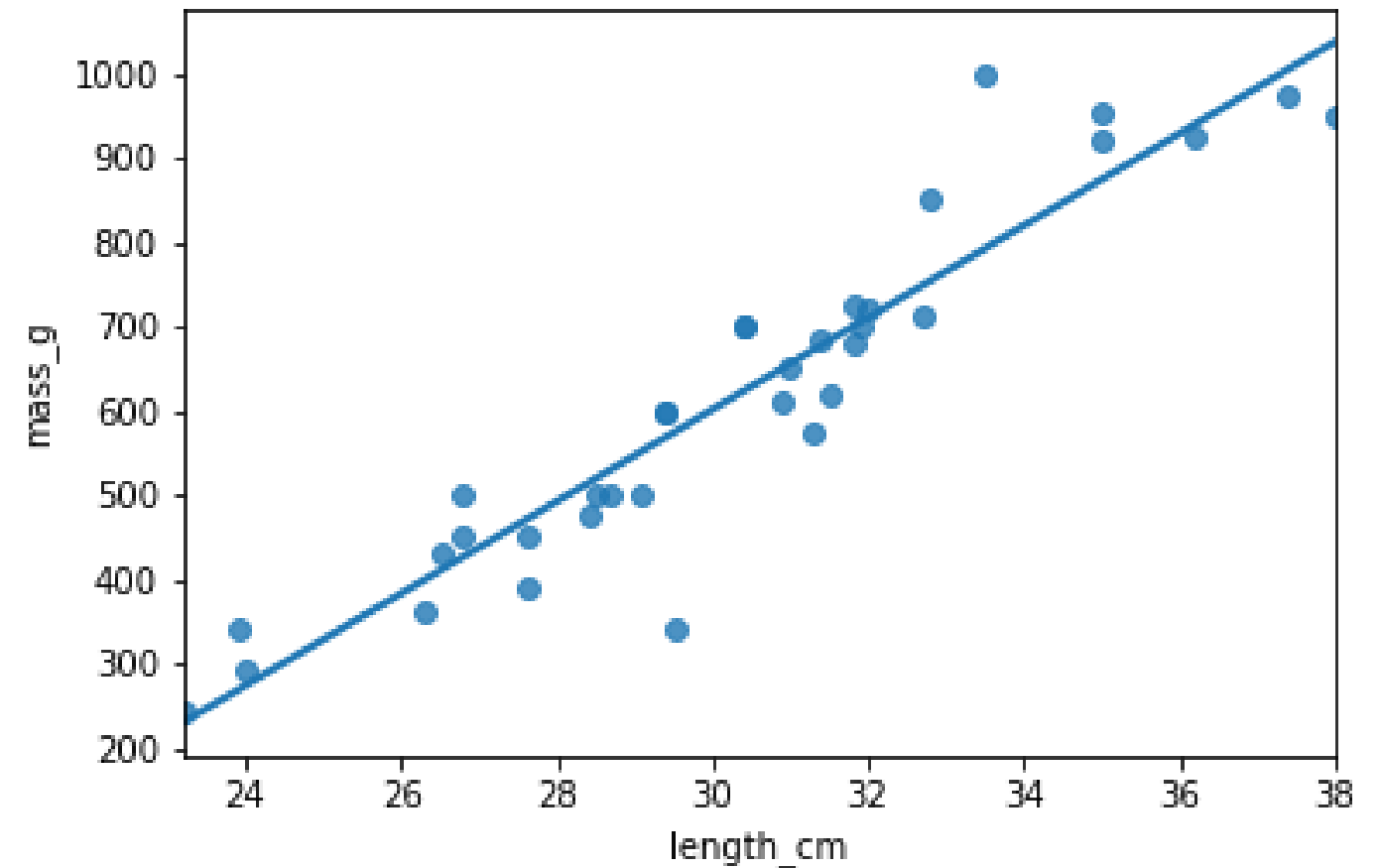
```
bream = fish[fish["species"] == "Bream"]  
print(bream.head())
```

	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?

create a new dataframe

```
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

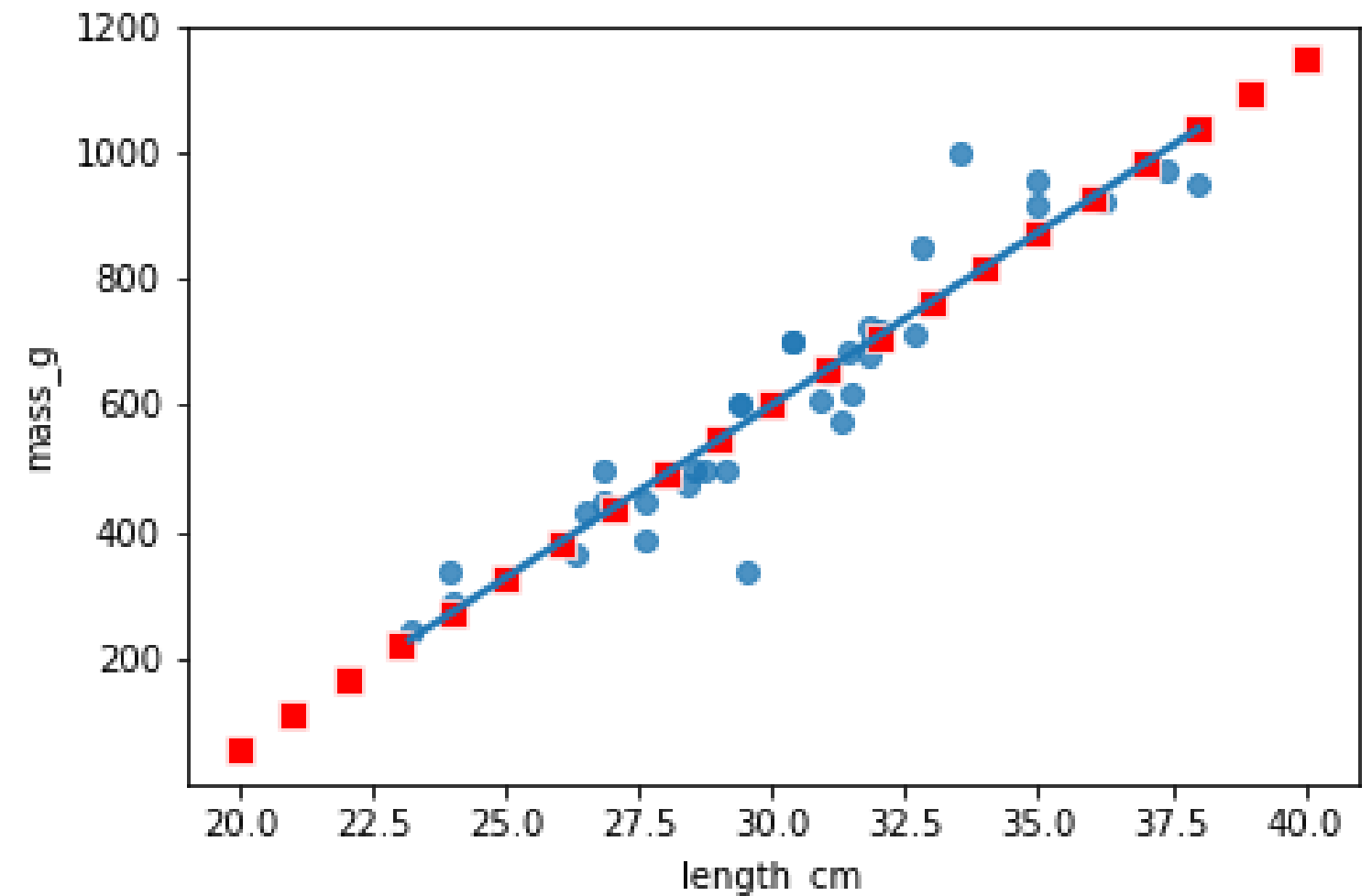
```
explanatory_data = pd.DataFrame(  
    {"length_cm": np.arange(20, 41)}  
)  
prediction_data = explanatory_data.assign(  
    mass_g=mdl_mass_vs_length.predict(explanatory_data)  
)  
print(prediction_data)
```

	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

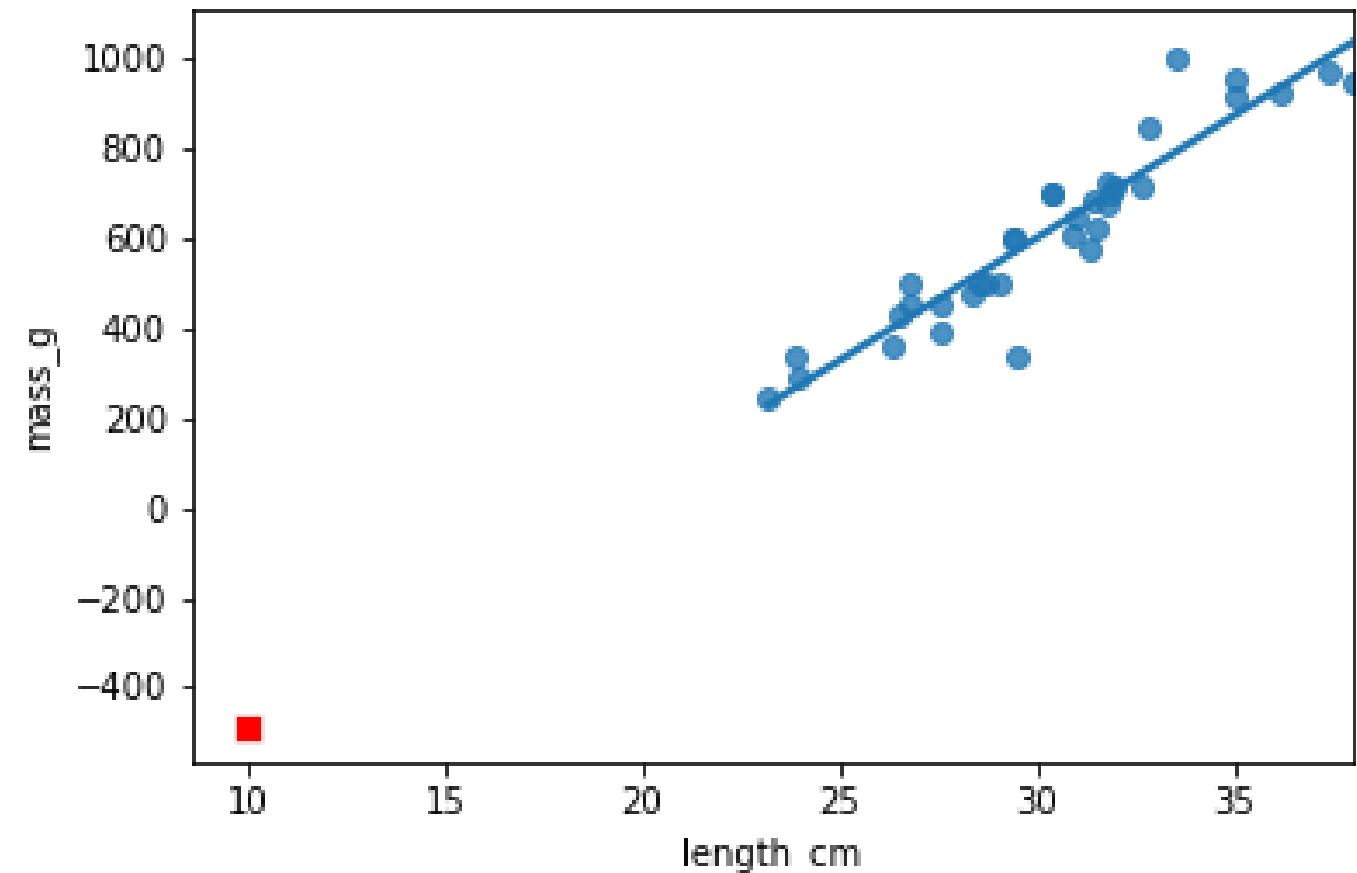
Extrapolating means making predictions outside the range of observed data.

```
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



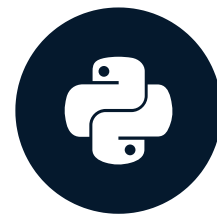
nonsensible to extrapolate

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

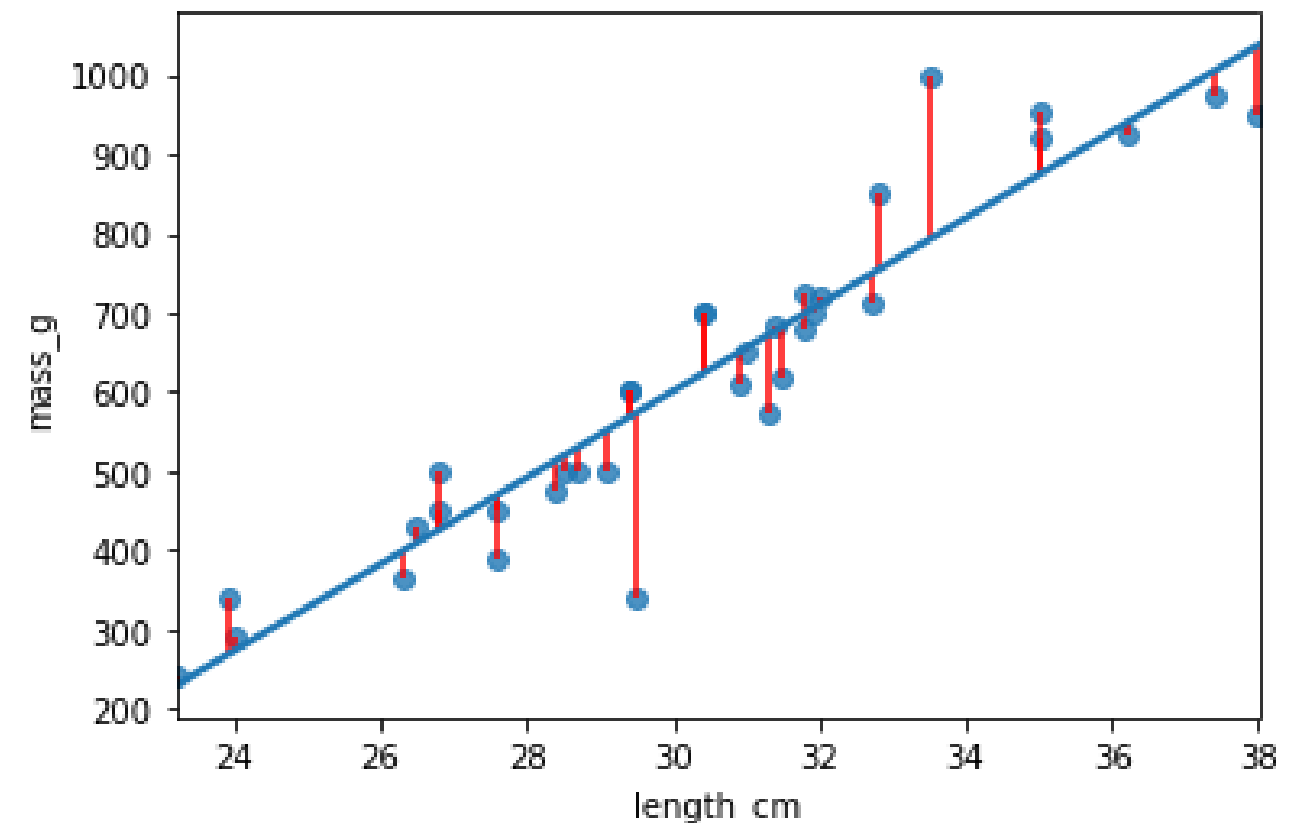
Residuals: actual response values minus predicted response values

```
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
```

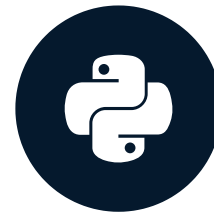

	coef	std err	t	P> t	[0.025	0.975]
<-----						
Intercept	-1035.3476	107.973	-9.589	0.000	-1255.020	-815.676
length_cm	54.5500	3.539	15.415	0.000	47.350	61.750
=====						
Omnibus:		7.314	Durbin-Watson:			1.478
Prob(Omnibus):		0.026	Jarque-Bera (JB):			10.857
Skew:		-0.252	Prob(JB):			0.00439
Kurtosis: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

Let's practice!

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON

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()
```

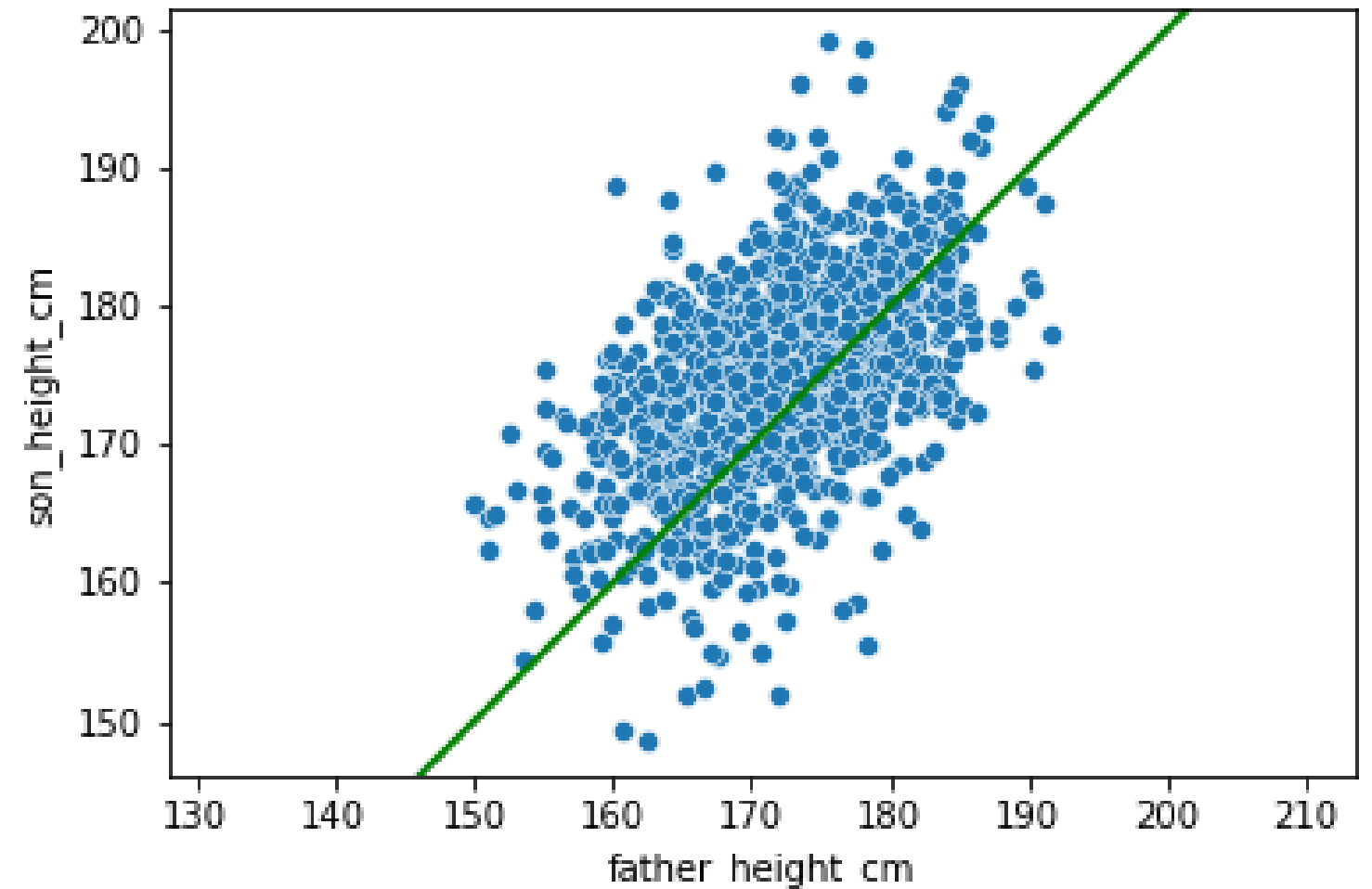
```
sns.scatterplot(x="father_height_cm",  
                y="son_height_cm",  
                data=father_son)
```

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

devdocs.io

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

should refer to point where the line passes through



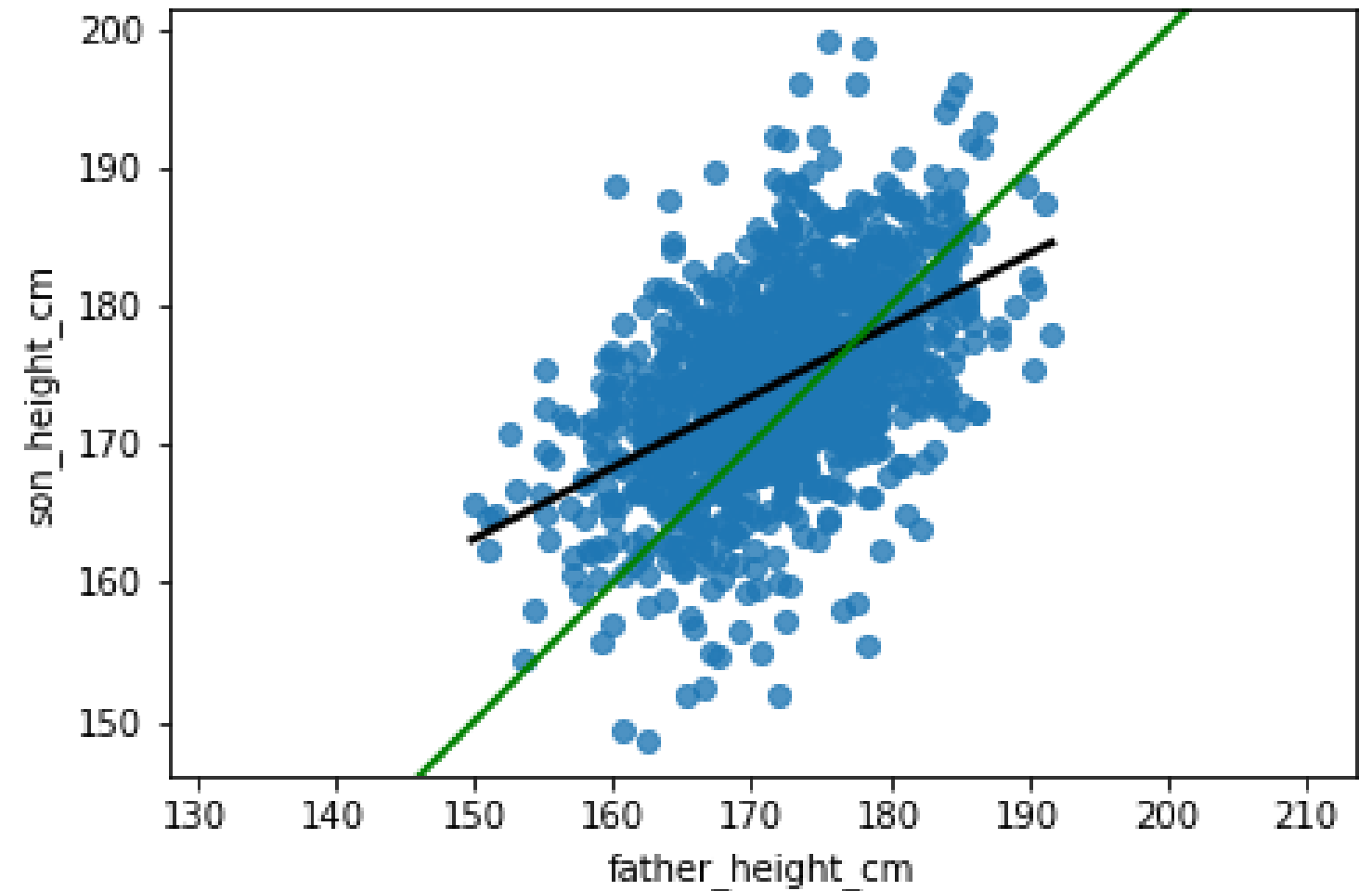
Adding a regression line

```
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

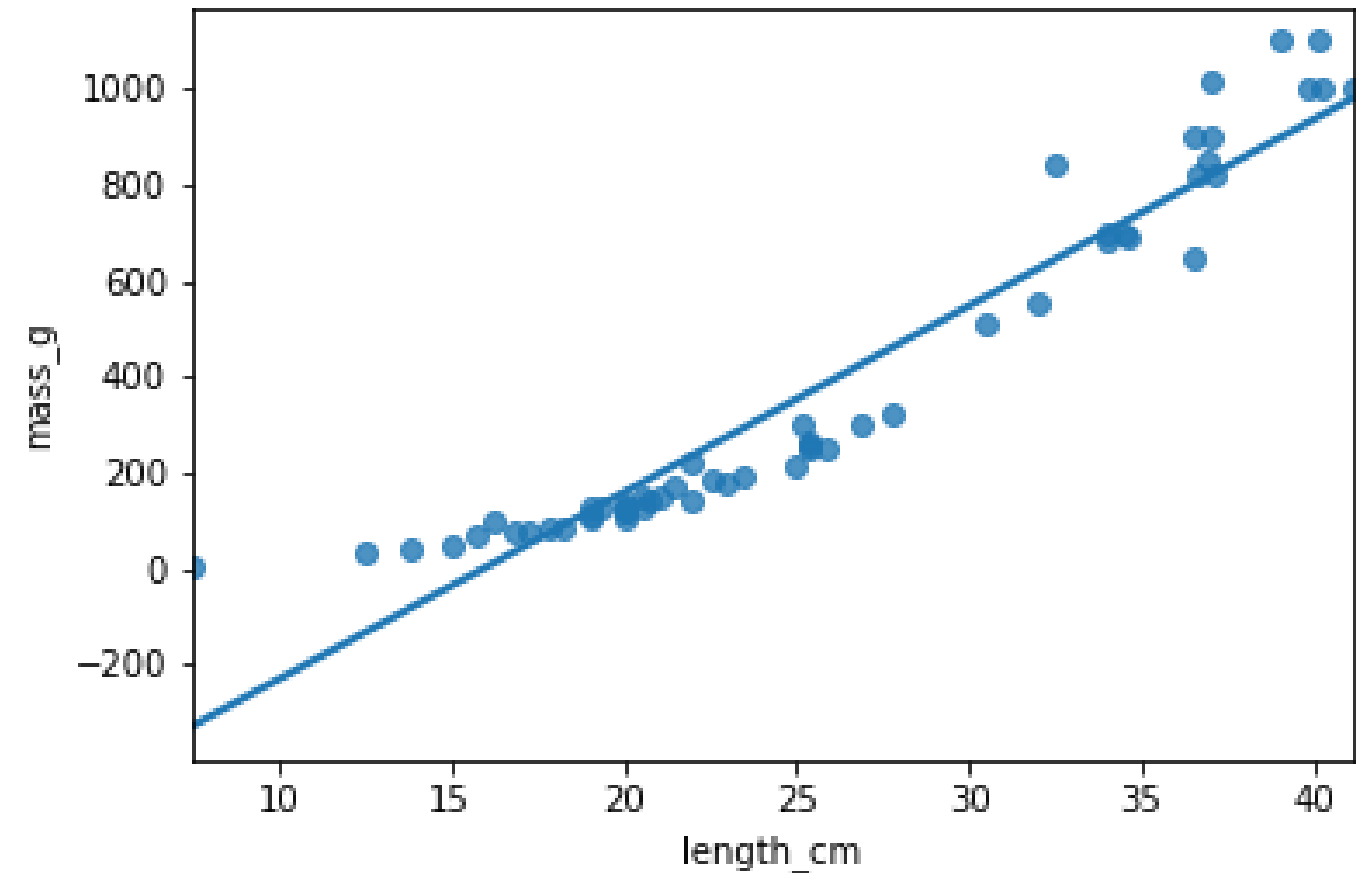
```
perch = fish[fish["species"] == "Perch"]  
print(perch.head())
```

	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()
```



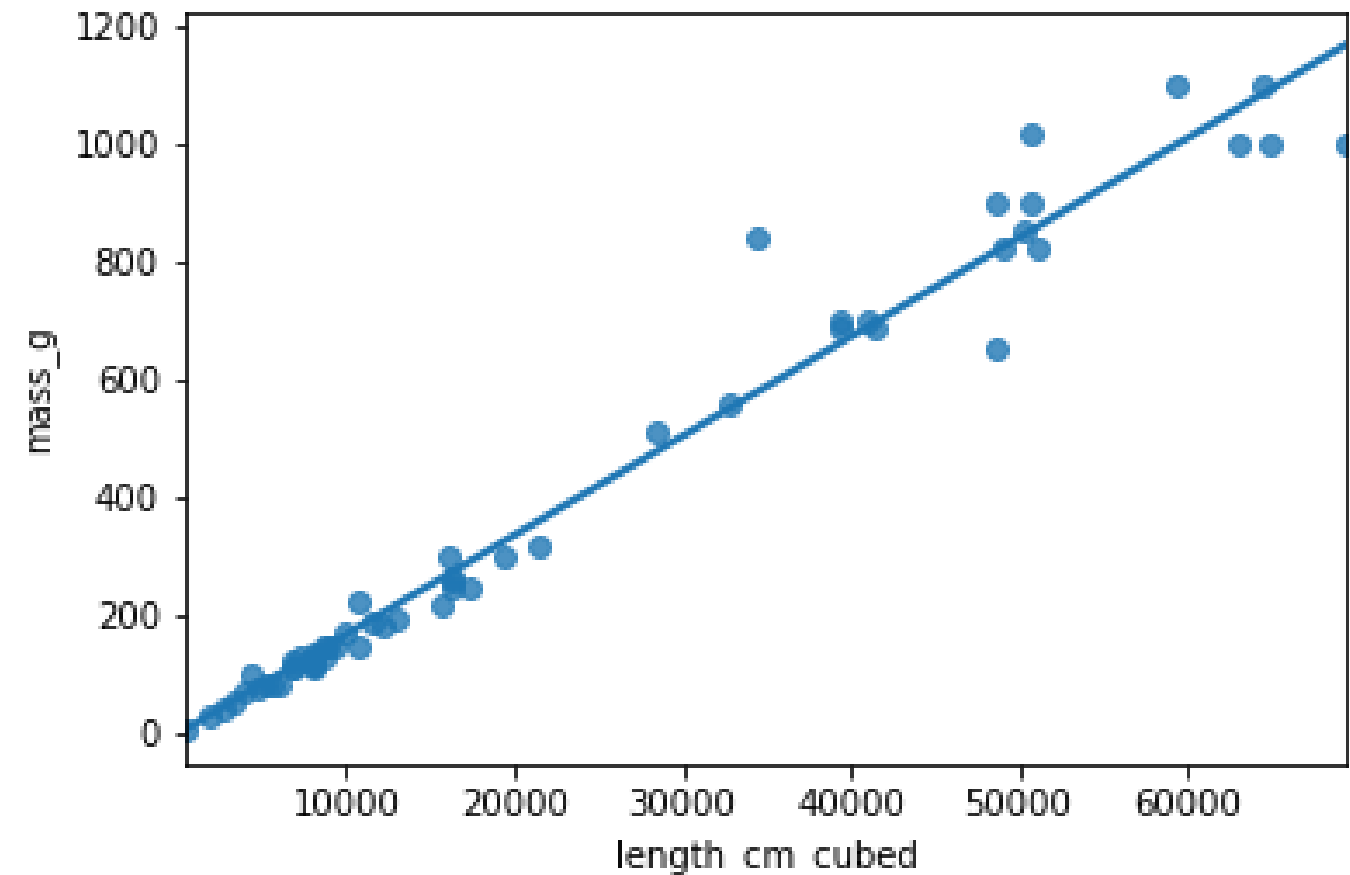
Bream vs. perch



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

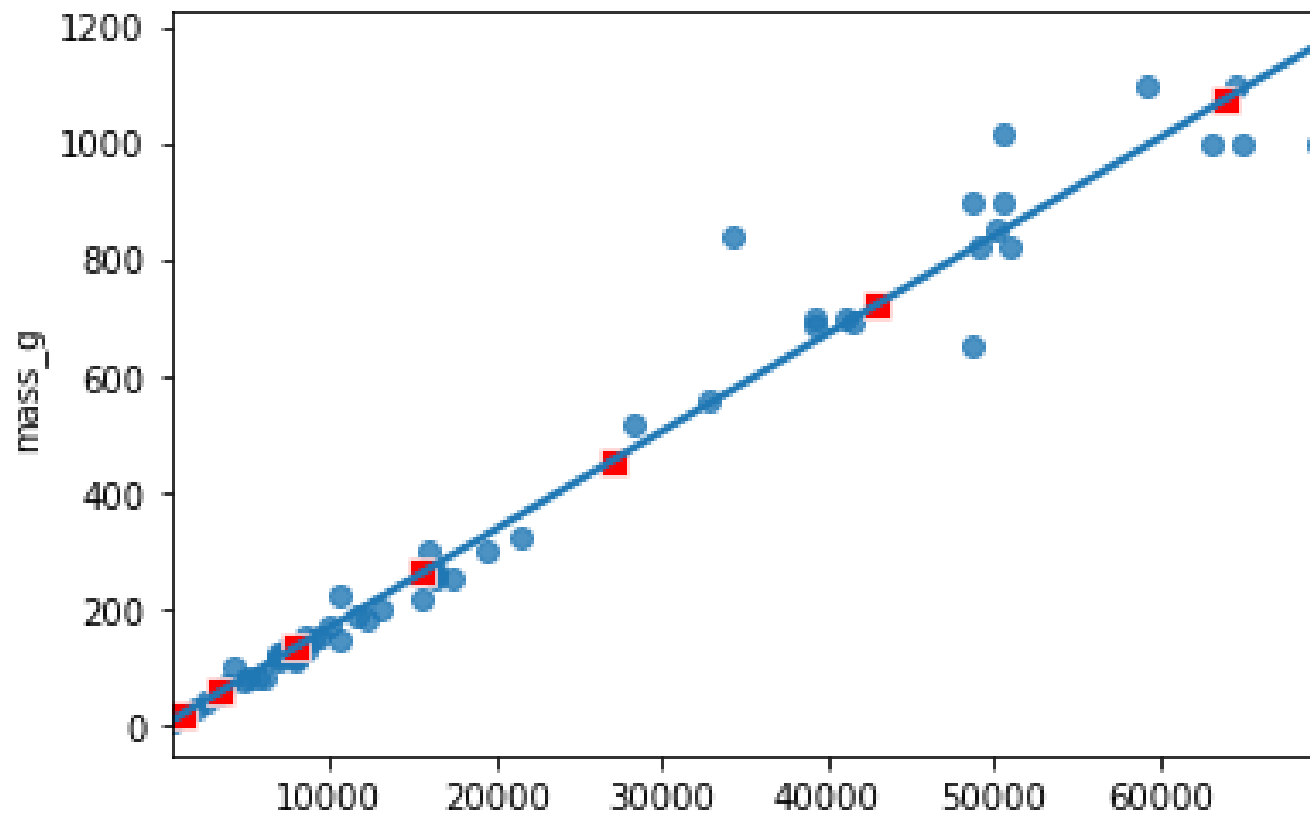
```
explanatory_data = pd.DataFrame({"length_cm_cubed": np.arange(10, 41, 5) ** 3,  
                                "length_cm": np.arange(10, 41, 5)})
```

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

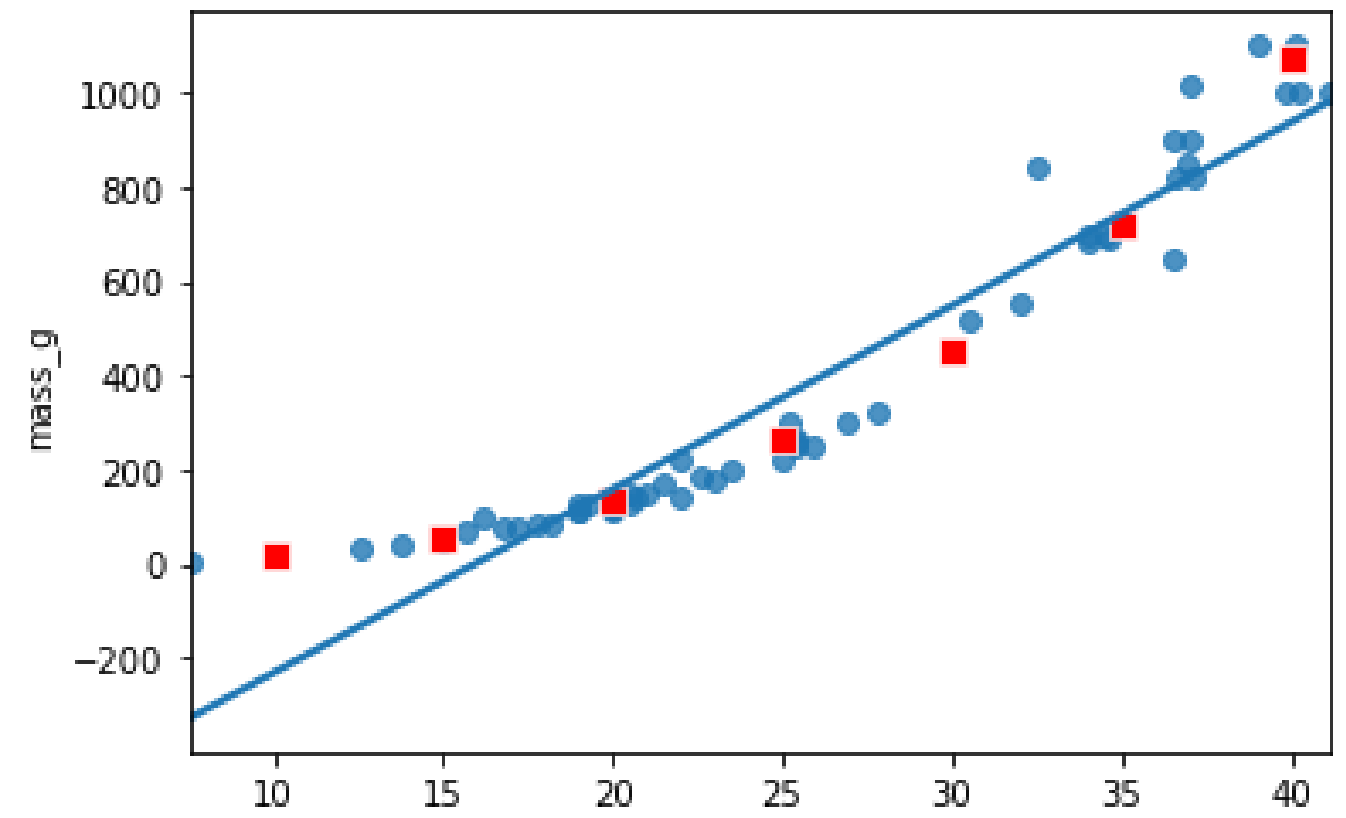
	length_cm_cubed	length_cm	mass_g
0	1000	10	16.678135
1	3375	15	56.567717
2	8000	20	134.247429
3	15625	25	262.313982
4	27000	30	453.364084
5	42875	35	719.994447
6	64000	40	1074.801781

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

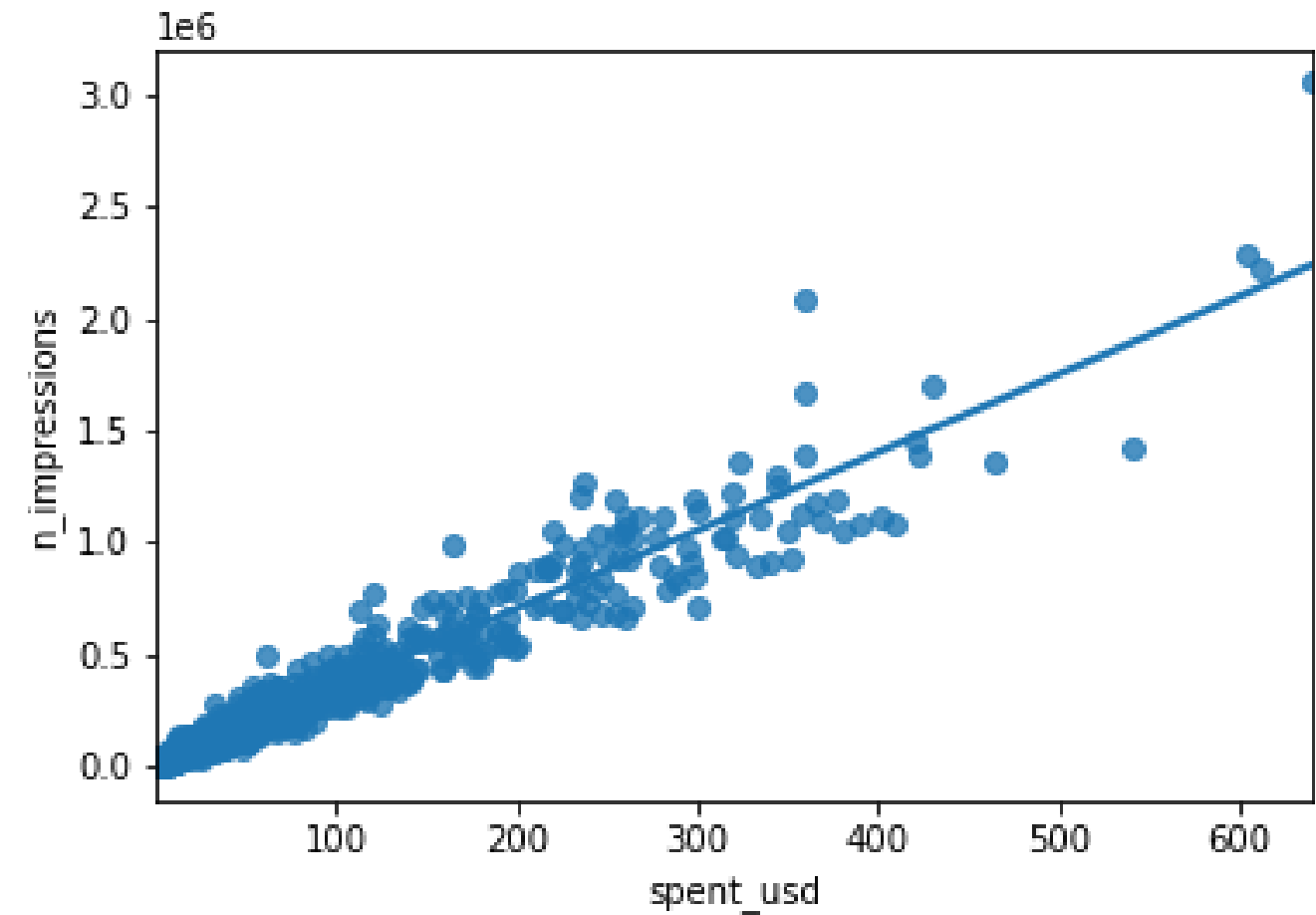
1. Pay Facebook to shows ads.
2. People see the ads ("impressions").
3. Some people who see it, click it.

- 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)
```

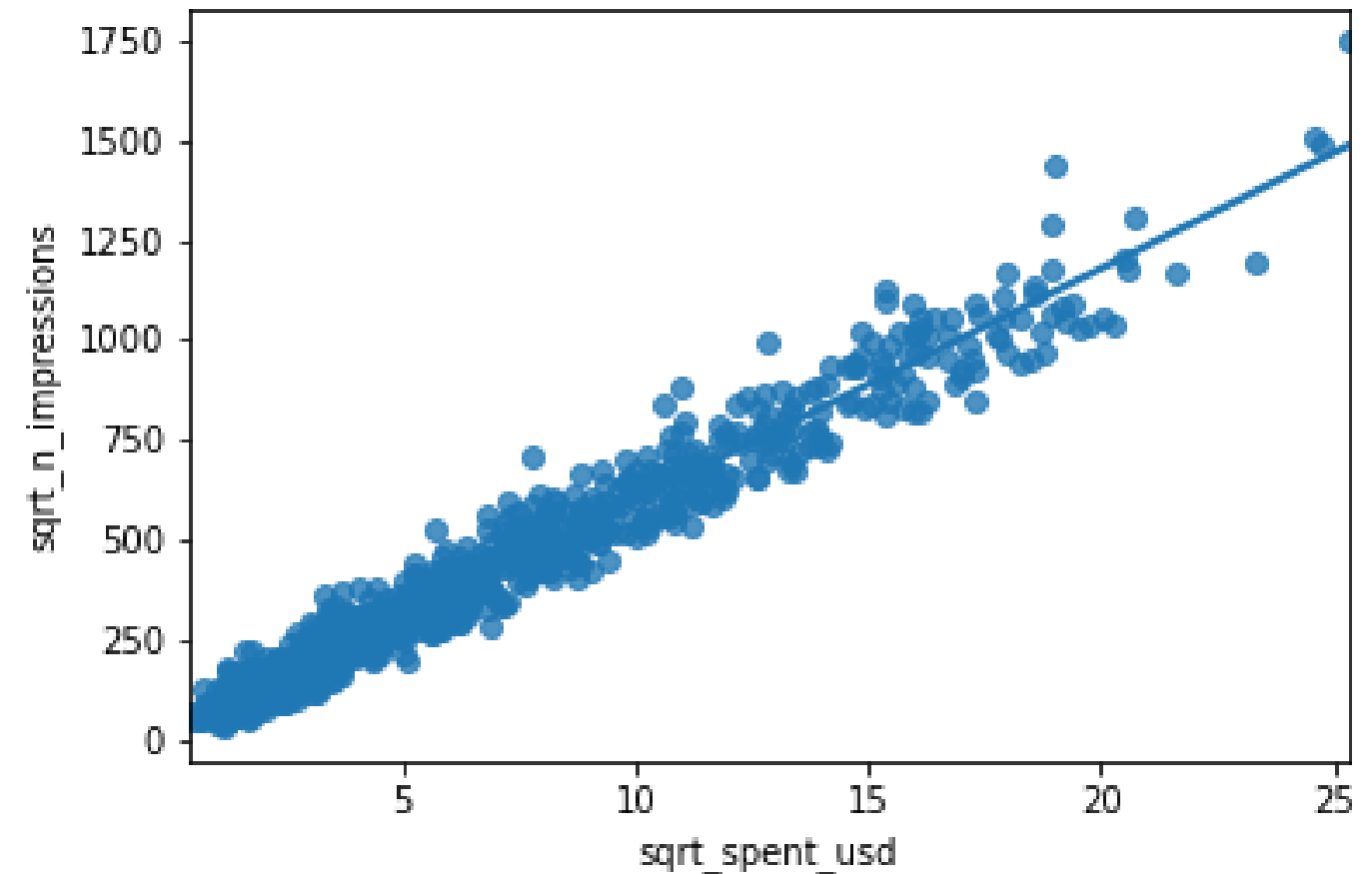


Square root vs square root

```
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)
```

back transformation

	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

• split-impressions

split-spent-uid

Model = "split-impressions" ~ split-spent-uid

- predicts split data

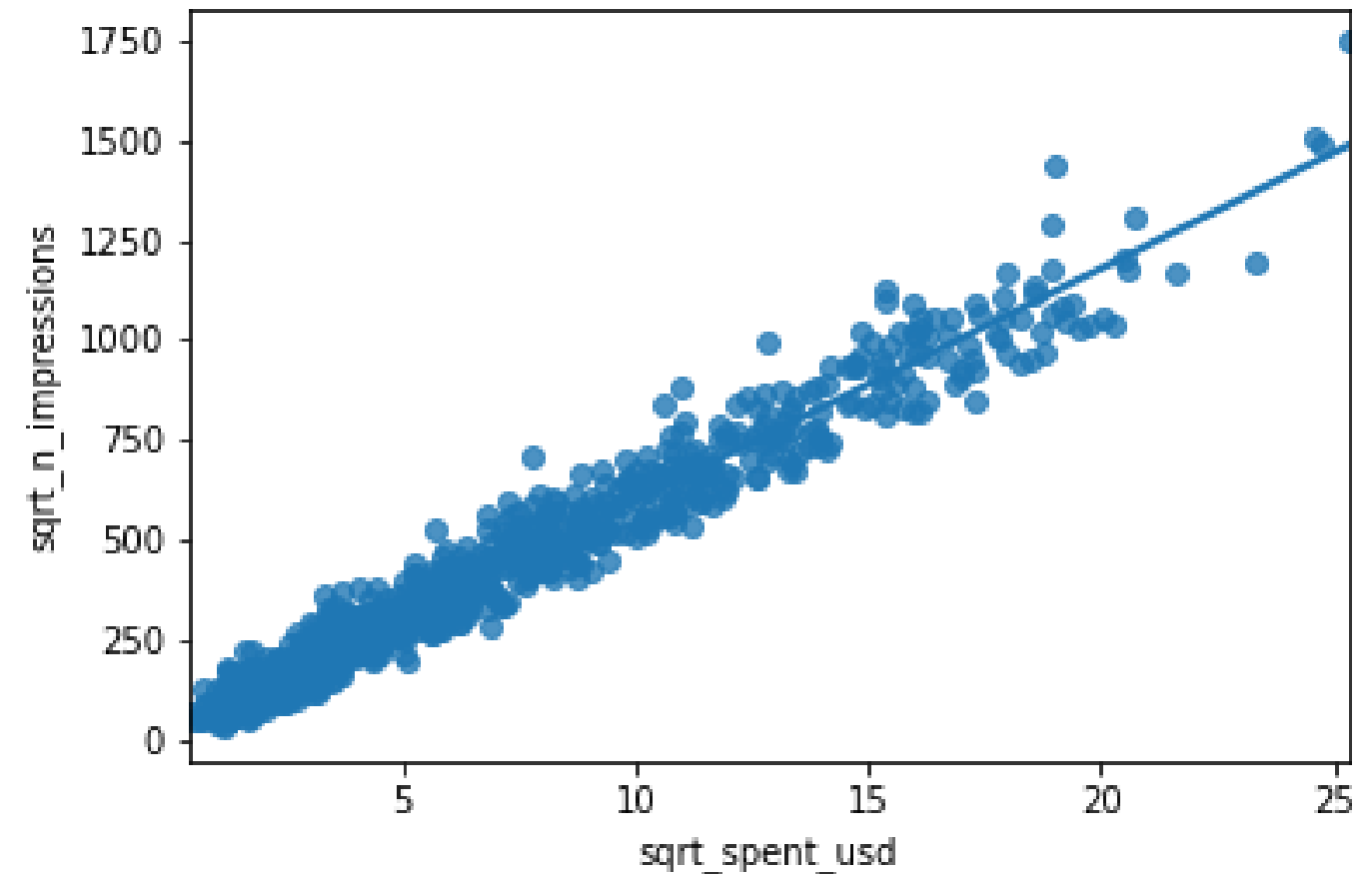
- then need to square that prediction.

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)
```



Let's practice!

INTRODUCTION TO REGRESSION WITH STATSMODELS IN PYTHON