EBA3500 LECTURE 4

INFERENCE, TRANSFORMATIONS, CORRELATION

JONAS MOSS

INFERENCE: REGRESSION FROM A STATISTICIAN'S VIEWPOINT

WHERE DOES THE DATA COME FROM?

Data generating process

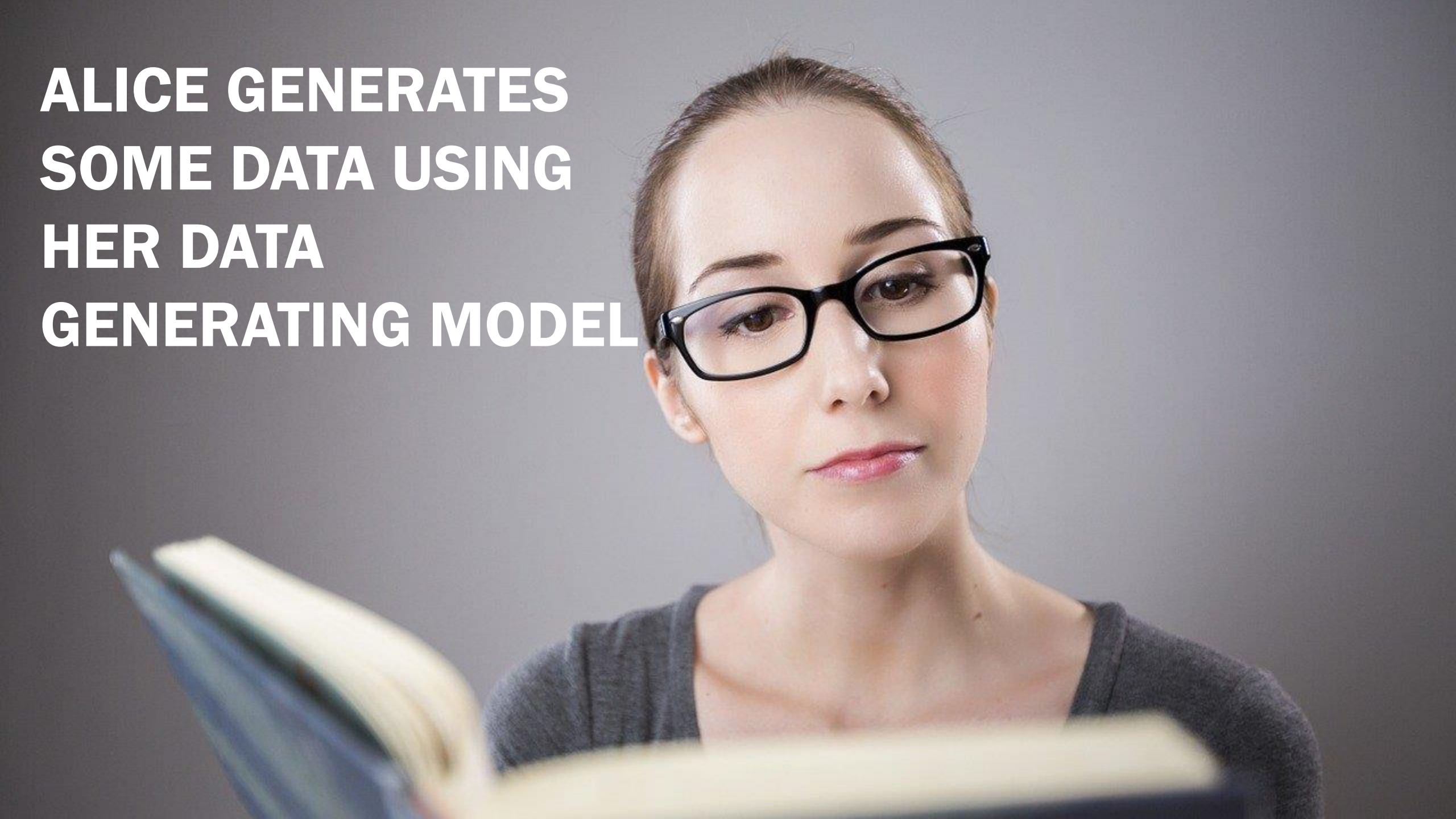
Data generating model

DATA GENERATING PROCESS – THINK SIMULATION!

```
import numpy as np

rng = np.random.default_rng(seed = 313)

# One model
x = np.linspace(0, 1, num = 100)
y = np.exp(3 * x ** 2) + rng.uniform(-1, 1, 100)

# Another model
x = np.linspace(0, 1, num = 100)
y = 3 + 2 * x + rng.uniform(-1, 1, 100)
```

THE CASE OF ALICE AND BOB

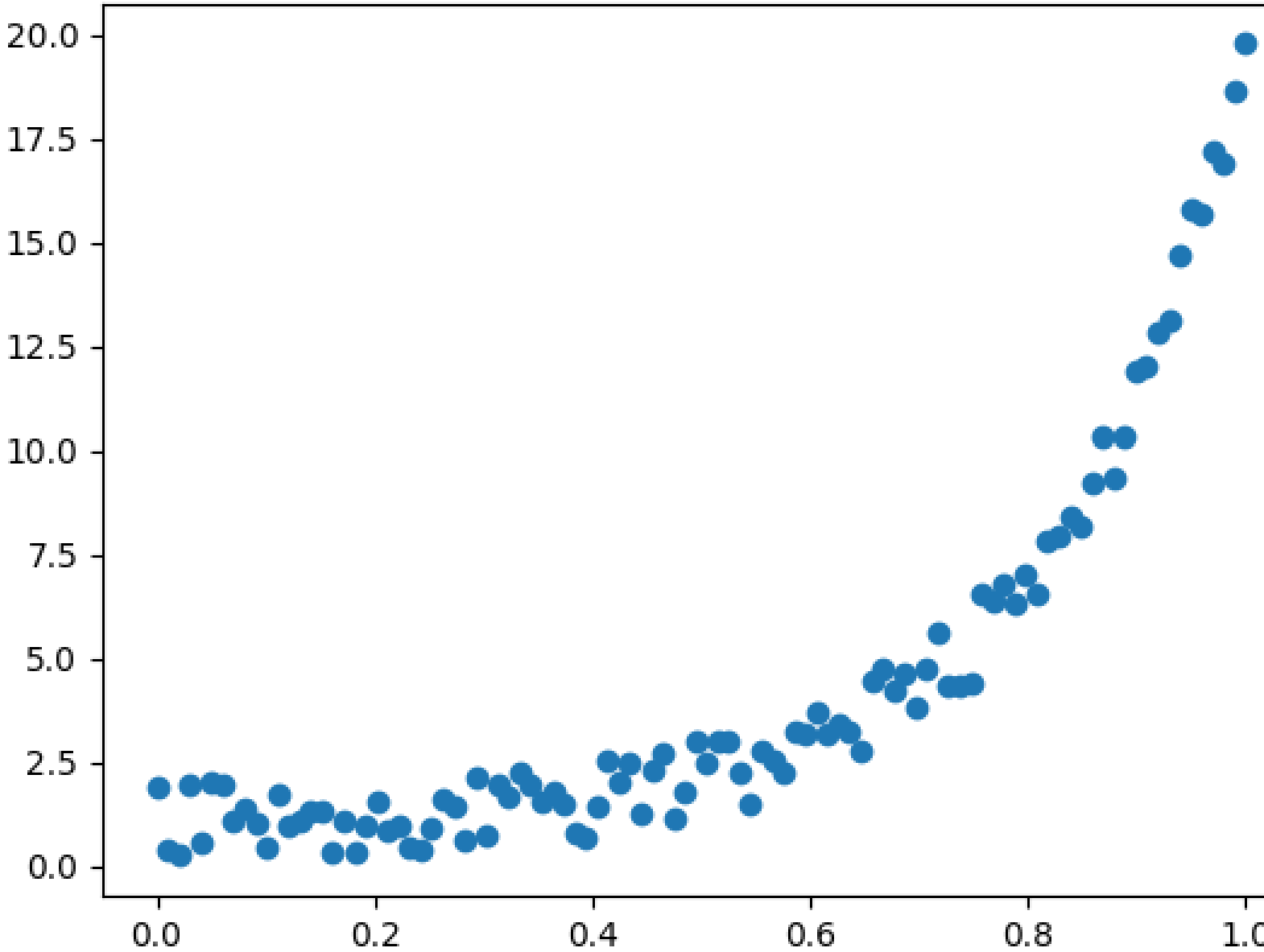
ALICE GENERATES SOME DATA USING HER DATA GENERATING MODEL



BOB DOESN'T KNOW THE DATA GENERATION PROCESS.

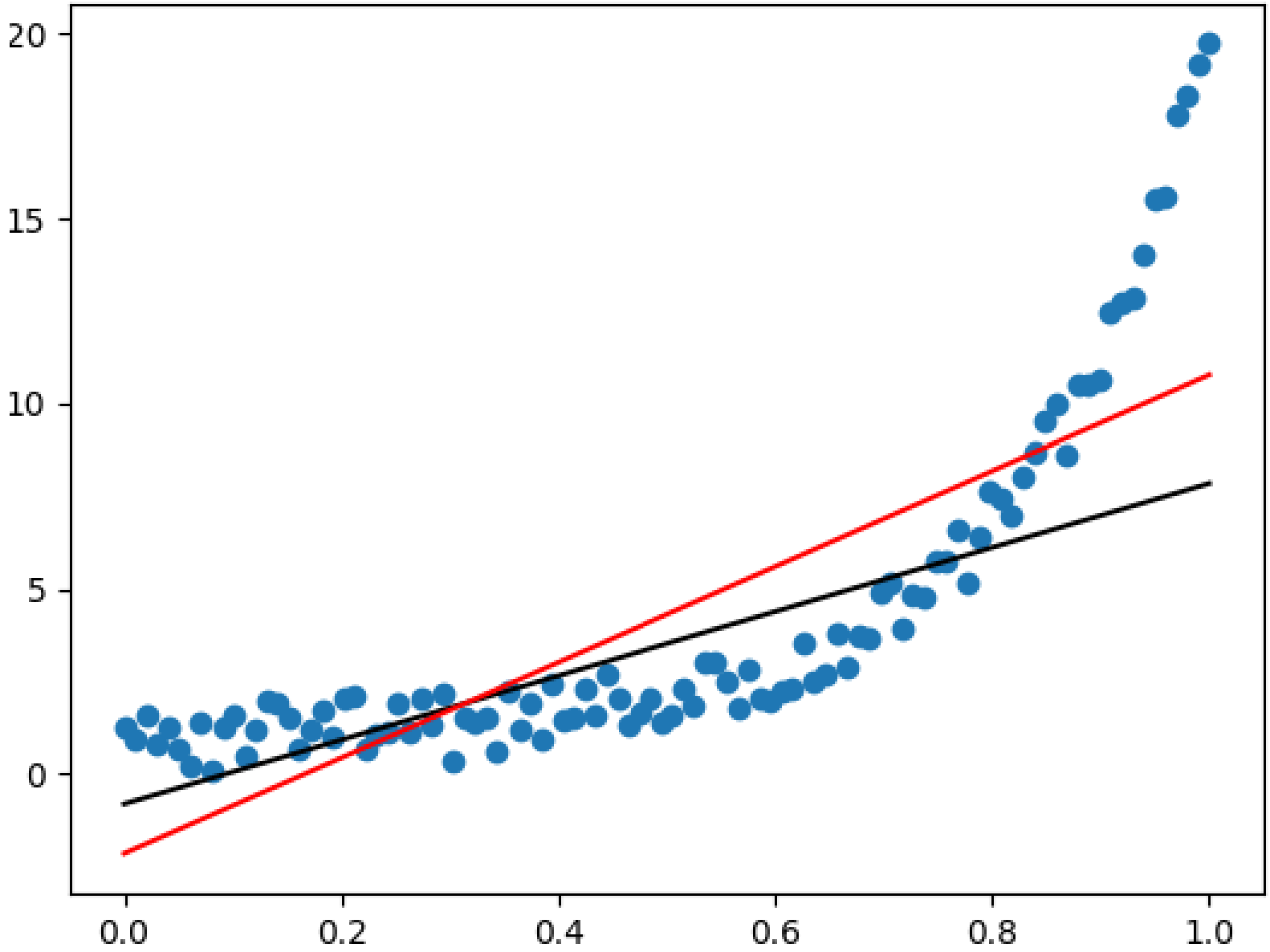
HE MUST GUESS A MODEL FOR THE DATA!

ALICE GENERATES SOME NON-LINEAR DATA



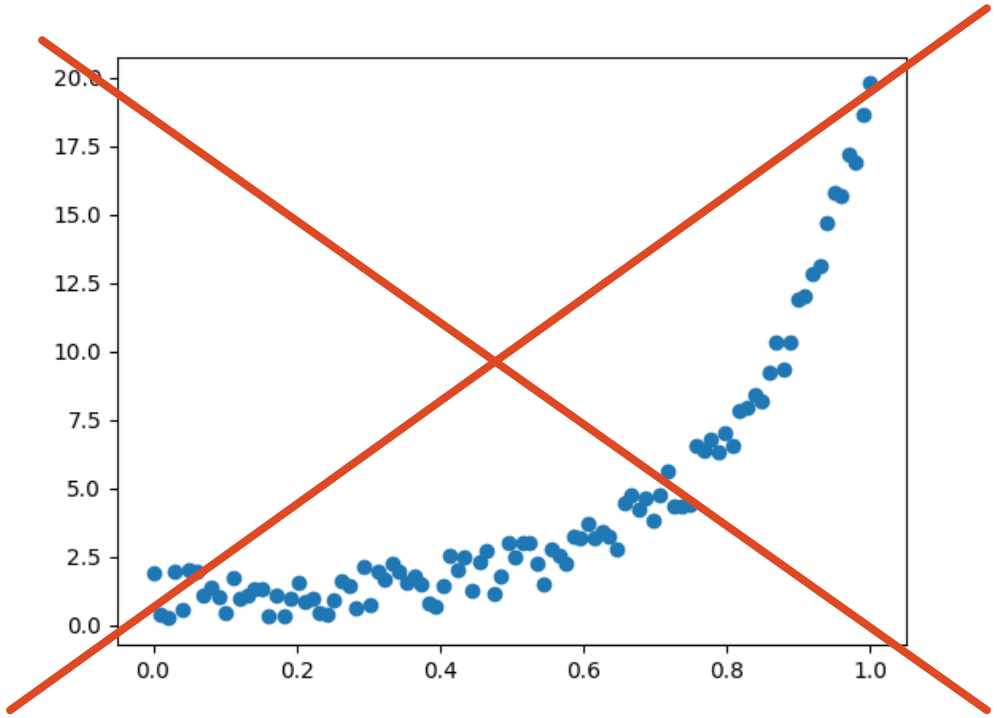
BUT CREDULOUS BOB DOESN'T CARE – HE BELIEVES IN THE LINEAR MODEL!

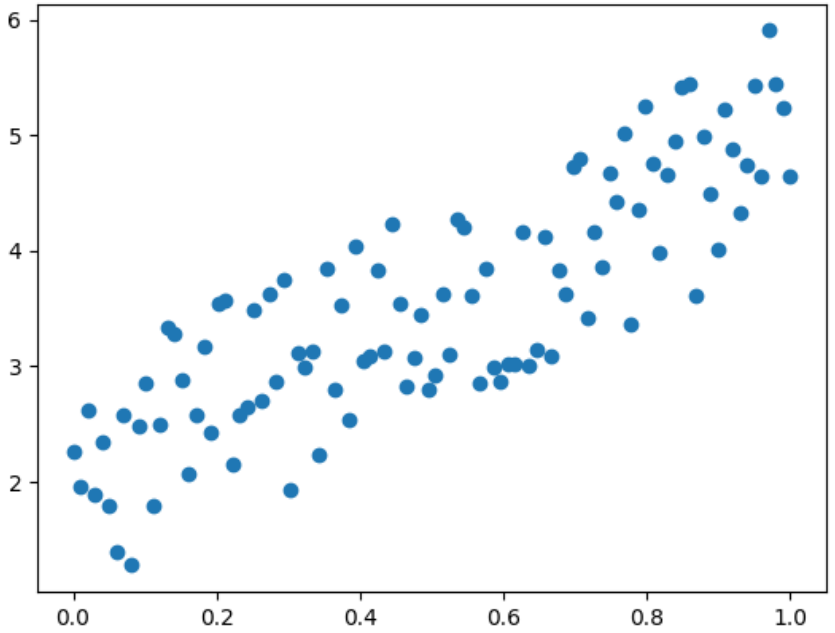
THAT'S WHY HE ENDS UP WITH THIS THING!



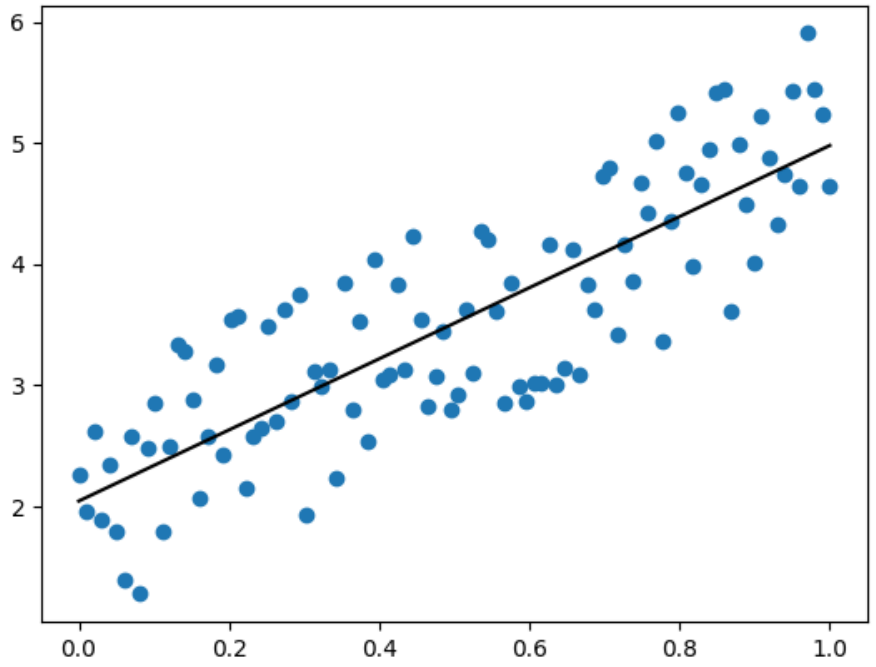
FOUNDATION OF STATISTICS: BE LIKE CREDULOUS BOB! JUST BELIEVE THAT THE MODEL IS TRUE!

THAT «THING» ON THE LEFT CAN'T POSSIBLY HAPPEN!!!





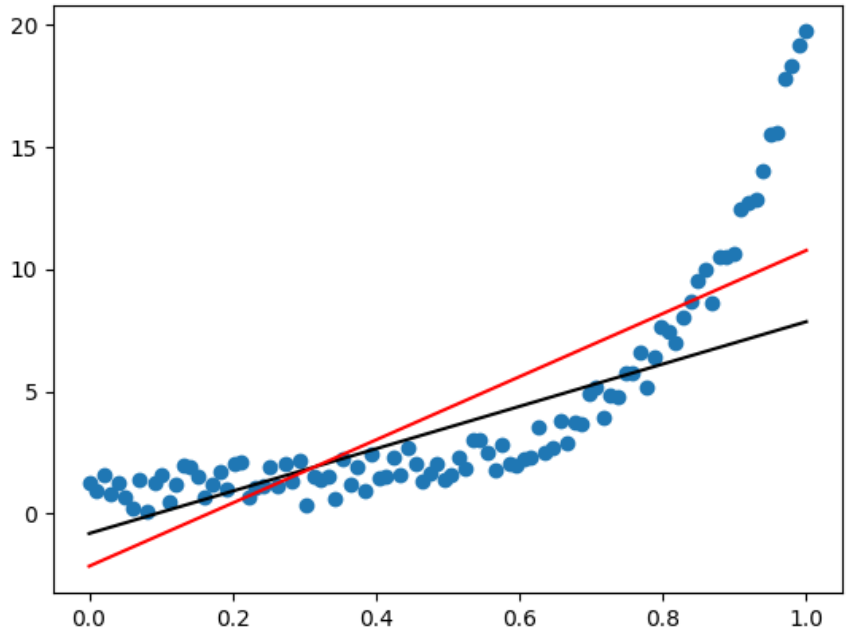
THE TRUE LINEAR MODEL



- Suppose we know $y_i = a + bx_i + u_i$.
- Here every u_i satisfies some extra conditions:
  1. u_i has the same distribution for every i,
  2. u_i is independent of x_i,
  3. all u_is are independent of each other.
  4. u_i has mean 0.

- These are the ***conditions for linear regression!***
- (There are slight variations of these; some people add that u_i is normal, some people only want u_i to be uncorrelated.)
- These conditions make most sense for least squares regression.
- u_i has median 0 would be best for least absolute deviations.

NOT AT ALL COMPATIBLE WITH THE NON-LINEAR MODEL



- We need to put the non-linear part into the error!
- But that's not compatible with condition (2), that the error should be independent of x_i!

```
import numpy as np
import matplotlib.pyplot as plt
rng = np.random.default_rng(seed = 313)

x = np.linspace(0, 1, num = 100)
v = rng.uniform(-1, 1, 100)
u = np.exp(3 * x ** 2) - (2 + 3 * x) + v
y = 2 + 3 * x + u
```

«ASSUMING THE LINEAR MODEL IS TRUE, THEN… »

- This is usually called an "assumption".
- Would be better to say "Provided the model is true, then …“
- Or just say “I believe the model is true, and therefore”.

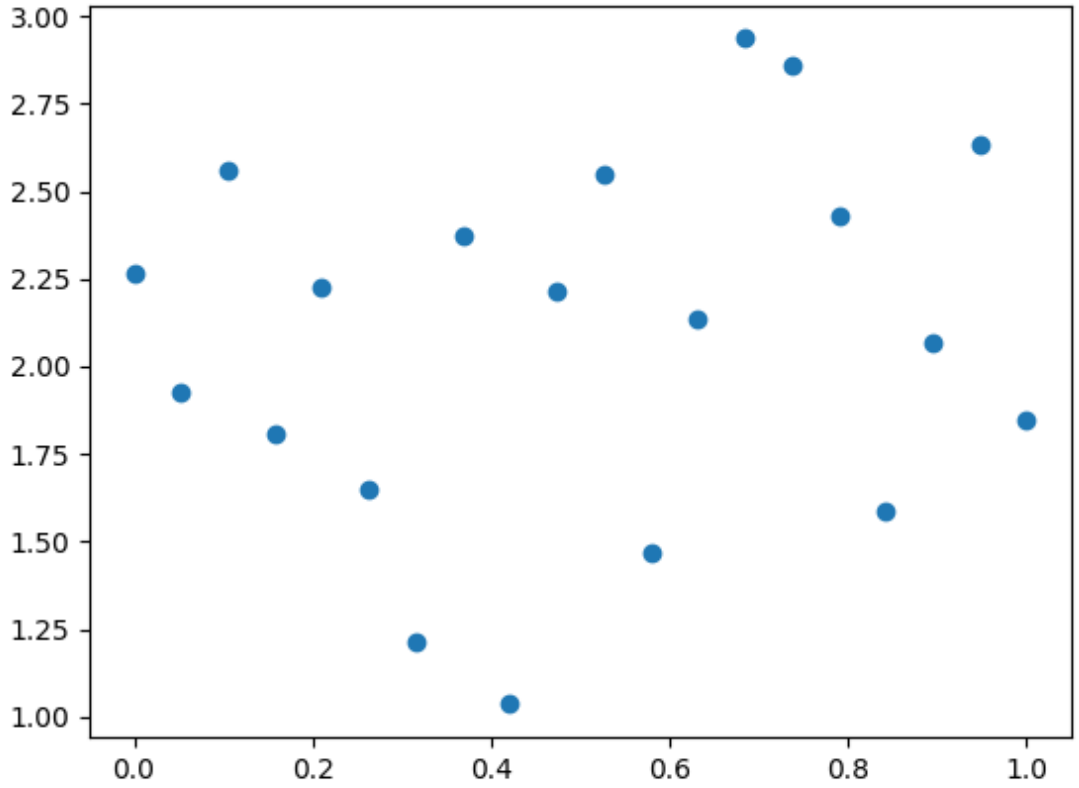

MORAL OF THE STORY
THERE IS ONE TRUE REGRESSION LINE!

NOW WE CAN ASK QUESTIONS ABOUT A AND B!

- **Alice:** I made some data from a linear model. Here it is! I promise that the slope is different from 0, I promise I promise!
- **Bob:** Oh yeah?! We'll see about that!!

ALICE'S IS LYING! (UNBEKNOWNST TO BOB)

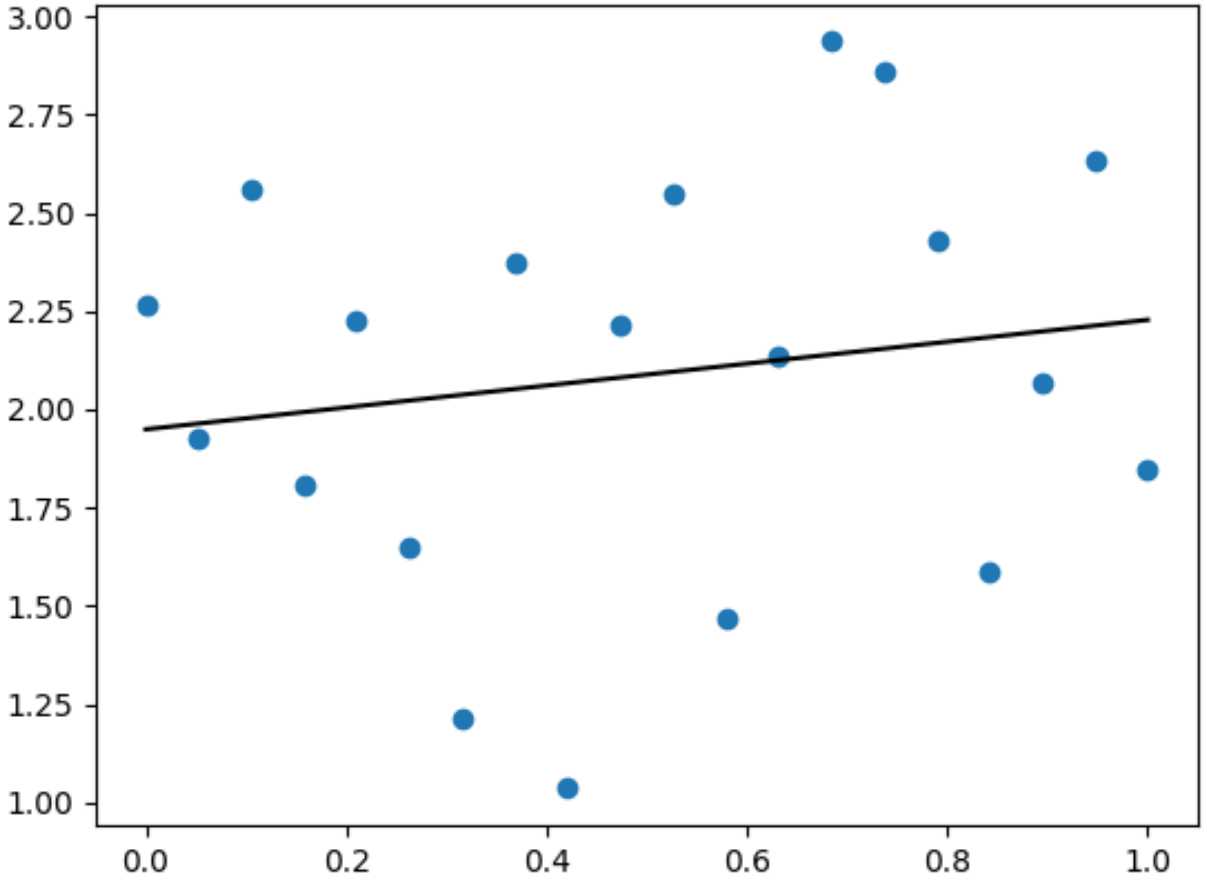
```
import numpy as np
import matplotlib.pyplot as plt
rng = np.random.default_rng(seed = 313)

num = 20
x = np.linspace(0, 1, num = 20)
u = rng.uniform(-1, 1, 20)
y = 2 + 0 * x + u

plt.scatter(x, y)
plt.show()
```



BOB FITS A REGRESSION LINE!

```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
import pandas as pd
res_ls = smf.ols("y ~ x", data = pd.DataFrame([y, x]).T).fit()
res_ls.params
plt.scatter(x, y)
plt.plot(x, res_ls.params[0] + x * res_ls.params[1], color = "black")
plt.show()
res_ls.params
>>> res_ls.params
Intercept    1.949194
x            0.278595
dtype: float64
```

HMMM… IS THE SLOPE 0.2785? OR MAYBE IT'S JUST 0!



BUT BOB KNOWS WHAT TO DO!

HE'S A STATISTICIAN!

P-VALUES

- "[The] *p*-value is the probability of obtaining test results at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct." Wikipedia
  - Don't simplify this definition!
  - Don't formulate it in your own words!
  - If asked for a definition, give this one!
- Don't get yourself in trouble over the definition of a p-value!

H_0: The true slope is $a = 0$.
H_a: The true slope isn't $a = 0$!

$$p = P(T \geq t \mid H_0)$$

«IF MY NULL-HYPOTHESIS, THAT ALICE IS LYING, IS TRUE, ARE THESE RESULTS PROBABLE?»

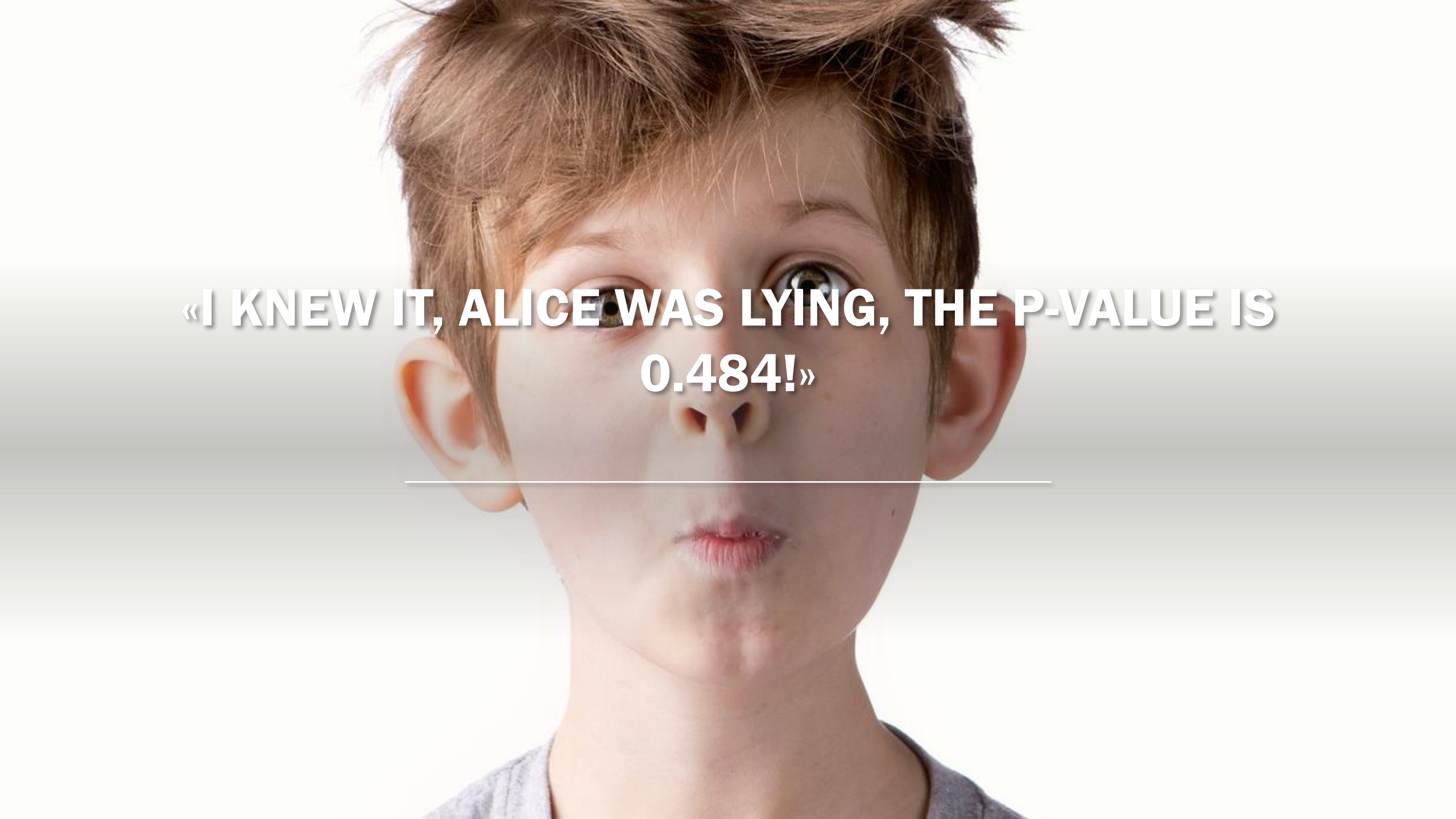
```python
# p-value
res = smf.ols("y ~ x", data = pd.DataFrame([y, x]).T).fit()
res.summary()
```

```
>>> res.summary()
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.028
Model:                            OLS   Adj. R-squared:                 -0.026
Method:                 Least Squares   F-statistic:                    0.5106
Date:                Tue, 07 Sep 2021   Prob (F-statistic):              0.484
Time:                        10:39:02   Log-Likelihood:                -14.596
No. Observations:                  20   AIC:                             33.19
Df Residuals:                      18   BIC:                             35.18
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      1.9492      0.228      8.548      0.000       1.470       2.428
x              0.2786      0.390      0.715      0.484      -0.541       1.098
==============================================================================
Omnibus:                        1.104   Durbin-Watson:                   2.135
Prob(Omnibus):                  0.576   Jarque-Bera (JB):                0.992
Skew:                          -0.373   Prob(JB):                        0.609
Kurtosis:                       2.205   Cond. No.                         4.18
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
"""
```

REMEMBER, A BIG P-VALUE IS CONSISTENT WITH THE NULL-HYPOTHESIS.

A SMALL P-VALUE IS NOT.

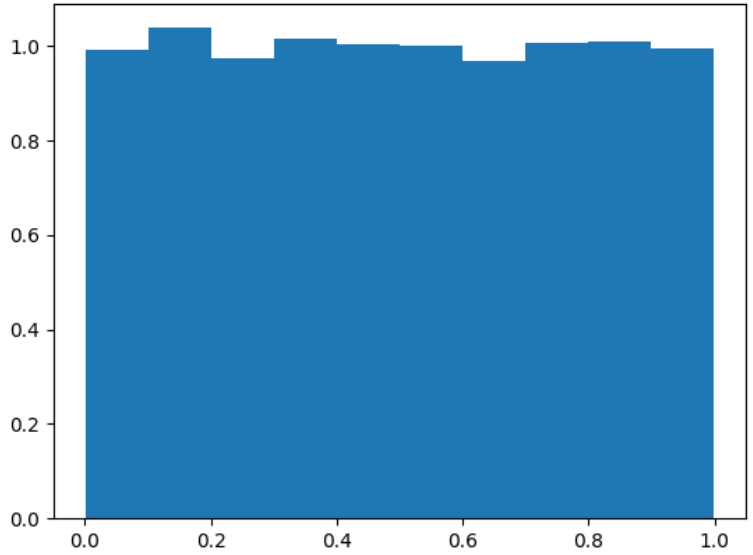
“SMALL” IS LESS THAN 0.05.



«I KNEW IT, ALICE WAS LYING, THE P-VALUE IS 0.484!»

THE DISTRIBUTION OF THE P-VALUES IS UNIFORM UNDER THE NULL-HYPOTHESIS!

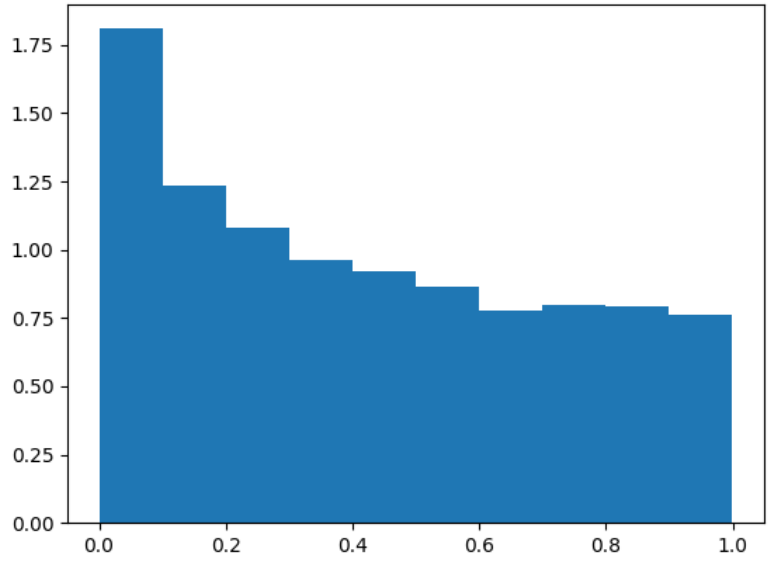
```
# Simulation.
pvalues = np.zeros(n_reps)
num = 20

for i in range(n_reps):
    # Simulate stuff.
    x = np.linspace(0, 1, num)
    u = rng.normal(0, np.sqrt(res_ls.scale), num)
    y = res_ls.params[0] + 0 * x + u

    # Fit
    res_sim = smf.ols("y ~ x", data = pd.DataFrame([y, x]).T).fit()
    pvalues[i] = res_sim.pvalues[1]

plt.hist(pvalues, density = True)
plt.show()
```



THE POINT: ALL P-VALUES ARE EQUALLY LIKELY WHEN THE NULL-HYPOTHESIS IS TRUE!

BUT SKEWED TO THE RIGHT IF THE NULL-HYPOTHESIS IS FALSE

```
# Simulation 2.
n_reps = 10000
pvalues = np.zeros(n_reps)
num = 20

for i in range(n_reps):
    # Simulate stuff.
    x = np.linspace(0, 1, num)
    u = rng.normal(0, np.sqrt(res_ls.scale), num)
    y = res_ls.params[0] + res_ls.params[1] * x + u

    # Fit
    res_sim = smf.ols("y ~ x", data = pd.DataFrame([y, x]).T).fit()
    pvalues[i] = res_sim.pvalues[1]

plt.hist(pvalues, density = True)
plt.show()
```



SMALL P-VALUES ARE MORE LIKELY WHEN THE NULL-HYPOTHESIS IS FALSE!

BOB CAN CALCULATE CONFIDENCE INTERVALS TOO!

- Make 95% confidence intervals CI_a and CI_b for a and b,
  - The probability that CI_a contains the true a is 0.95.
  - The probability that CI_b contains the true b is 0.95.
- You can use other probabilities too, but you knew that.
- Remember that the true intercept a and slope b are unambiguously defined!

```python
# p-value
res = smf.ols("y ~ x", data = pd.DataFrame([y, x]).T).fit()
res.summary()
```

```
>>> res.summary()
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.028
Model:                            OLS   Adj. R-squared:                 -0.026
Method:                 Least Squares   F-statistic:                    0.5106
Date:                Tue, 07 Sep 2021   Prob (F-statistic):              0.484
Time:                        10:39:02   Log-Likelihood:                -14.596
No. Observations:                  20   AIC:                             33.19
Df Residuals:                      18   BIC:                             35.18
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      1.9492      0.228      8.548      0.000       1.470       2.428
x              0.2786      0.390      0.715      0.484      -0.541       1.098
==============================================================================
Omnibus:                        1.104   Durbin-Watson:                   2.135
Prob(Omnibus):                  0.576   Jarque-Bera (JB):                0.992
Skew:                          -0.373   Prob(JB):                        0.609
Kurtosis:                       2.205   Cond. No.                         4.18
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
"""
```

These p-values are based on t-tests.

They are *exact* when the linear model is true and the residuals are normal.

Are *good* even when this isn't the case.

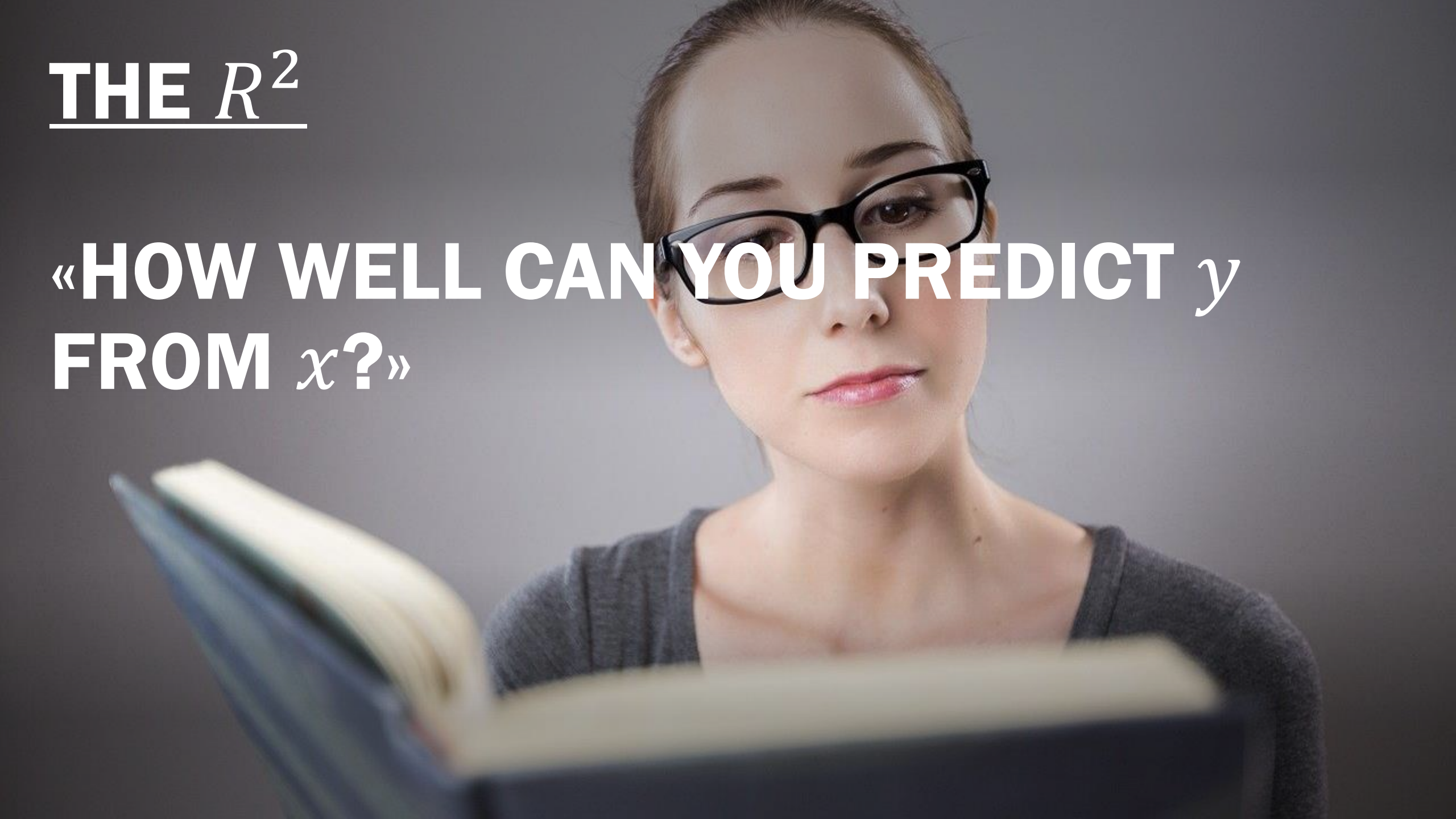
$$t = \frac{\widehat{\beta} - \beta}{s_{\widehat{\beta}}} \sim t_{n-2},$$

where

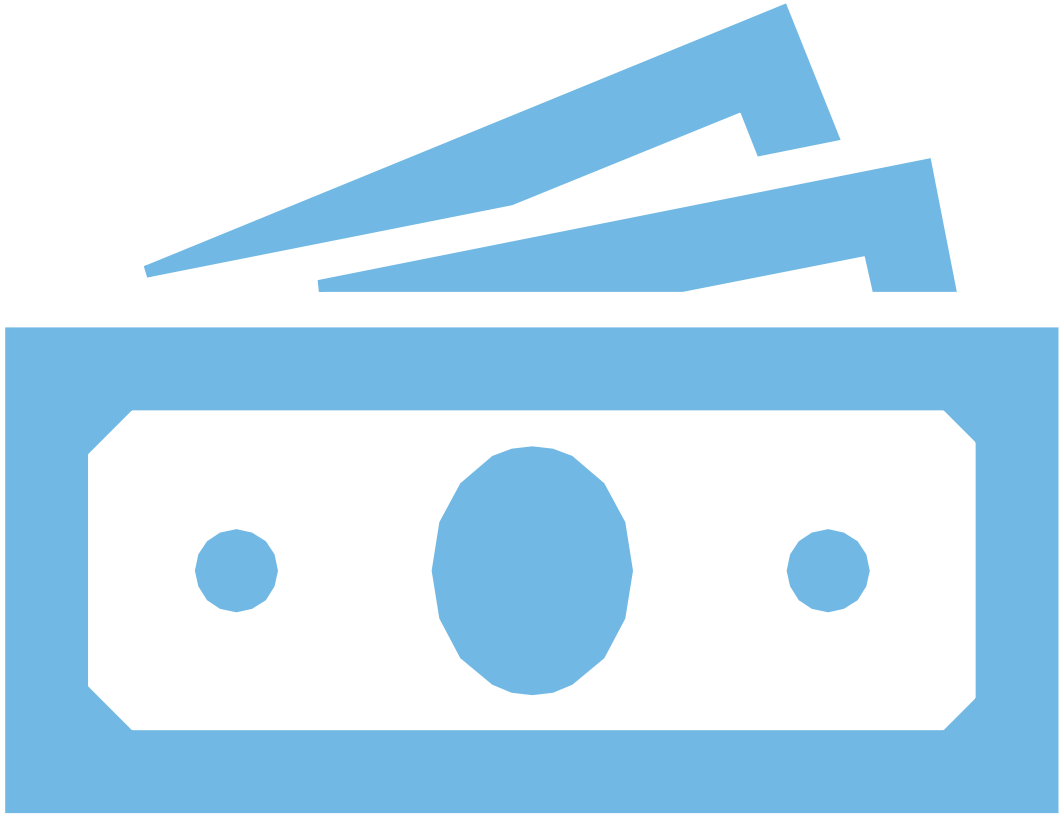
$$s_{\widehat{\beta}} = \sqrt{\frac{\frac{1}{n-2}\sum_{i=1}^{n} \hat{\varepsilon}_i^{\,2}}{\sum_{i=1}^{n}(x_i - \bar{x})^2}}$$

is the *standard error* of the estimator $\widehat{\beta}$.

THE R^2



«HOW WELL CAN YOU PREDICT y FROM x?»



HOW DO WE QUANTIFY THIS?

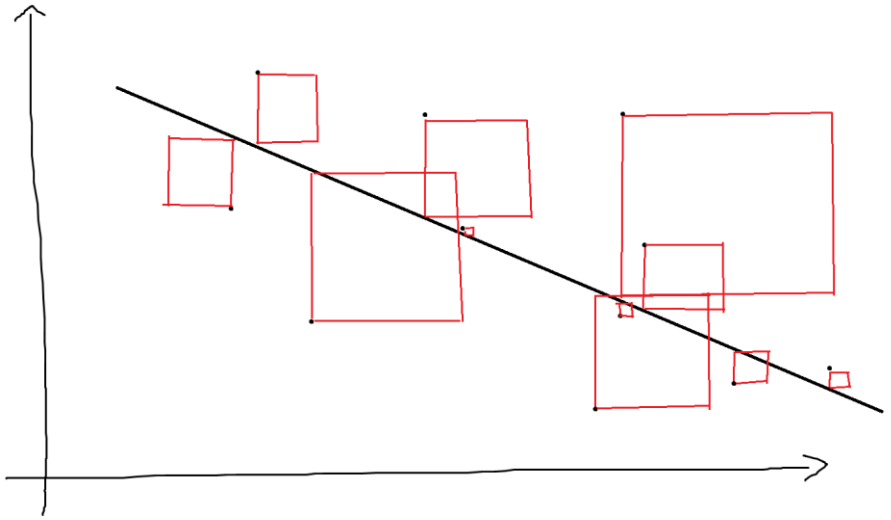
- Recall that a, b in $y = a + bx$ are estimated using a distance function.
- Minimize $\sum d(y_i, a + bx_i)$
- How about $\min_m \sum d(y_i, m)$? That's the best we could do if we know only y_i!
- We will use that as our baseline.

COMPARE THE BEST $\sum d(y_i, a + bx_i)$ AND THE BEST $\sum d(y_i, m)$!

WHAT m MINIMIZES $\sum d(y_i, m)$?

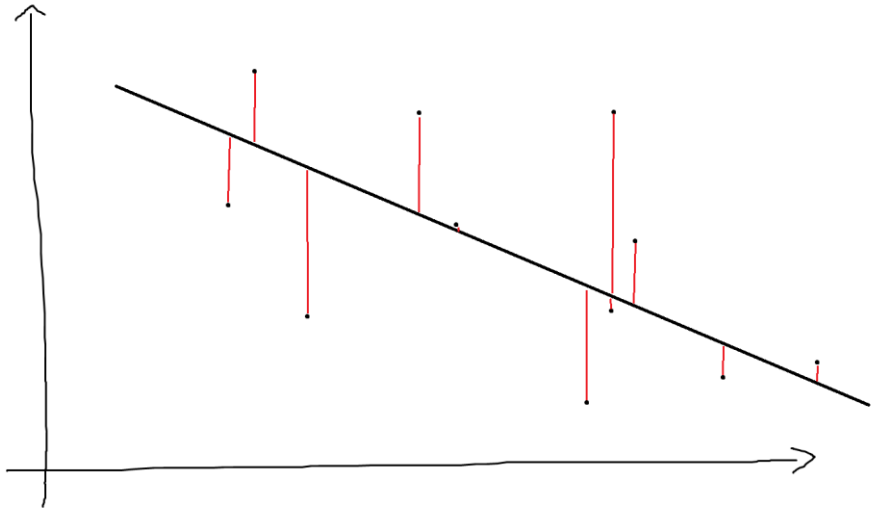
Least squares

- **Solution:** The mean $\overline{y}$!
- **Value:** $\sum(y_i - \overline{y})^2$



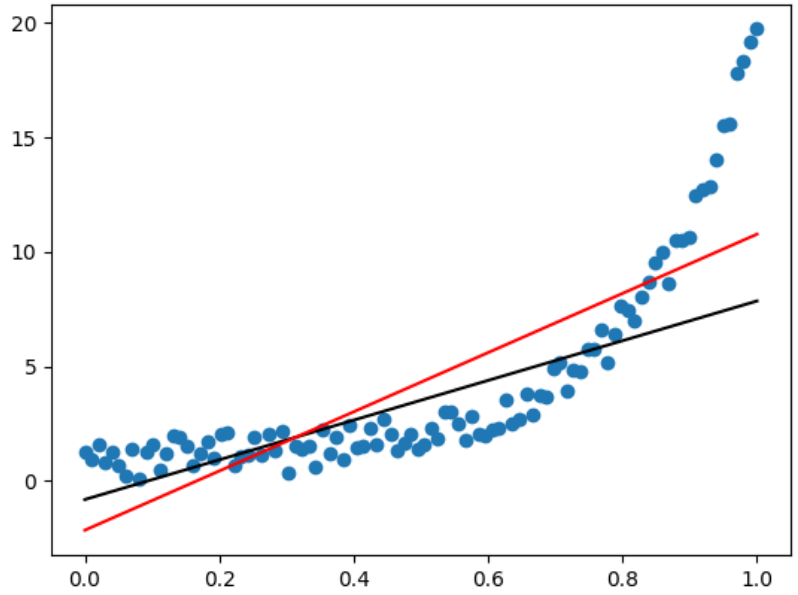
Least absolute deviations

- **Solution:** The median of $\{y_i\}$!
- **Value:** $\sum|y_i - med(y_i)|$



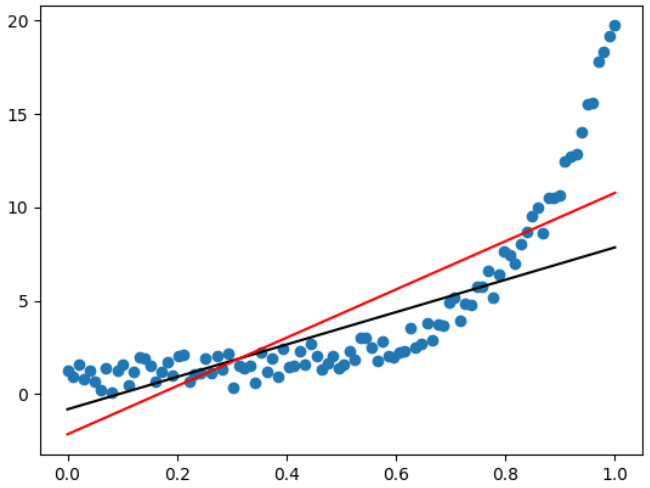
A SCALE PROBLEM

- $D = \sum(y_i - m)^2 - \sum(y_i - (a + bx_i))^2$
- $\sum(y_i - m)^2 = 2168$ and $\sum\left(y_i - (a + bx_i)\right)^2 = 747$
- $\sum(y_i - m)^2 - \sum\left(y_i - (a + bx_i)\right)^2 = 1421!$
- Impossible to interpret!




```
res_ls = smf.ols("y ~ x", data = pd.DataFrame([y, x]).T).fit()
lower = np.sum((y - np.mean(y)) ** 2)
# 2168.736140549201
upper = np.sum((y - (res_ls.params[0] + x * res_ls.params[1])) ** 2)
# 747.2527517297256
lower - upper
# 1421.4833888194755
```

THE FIX

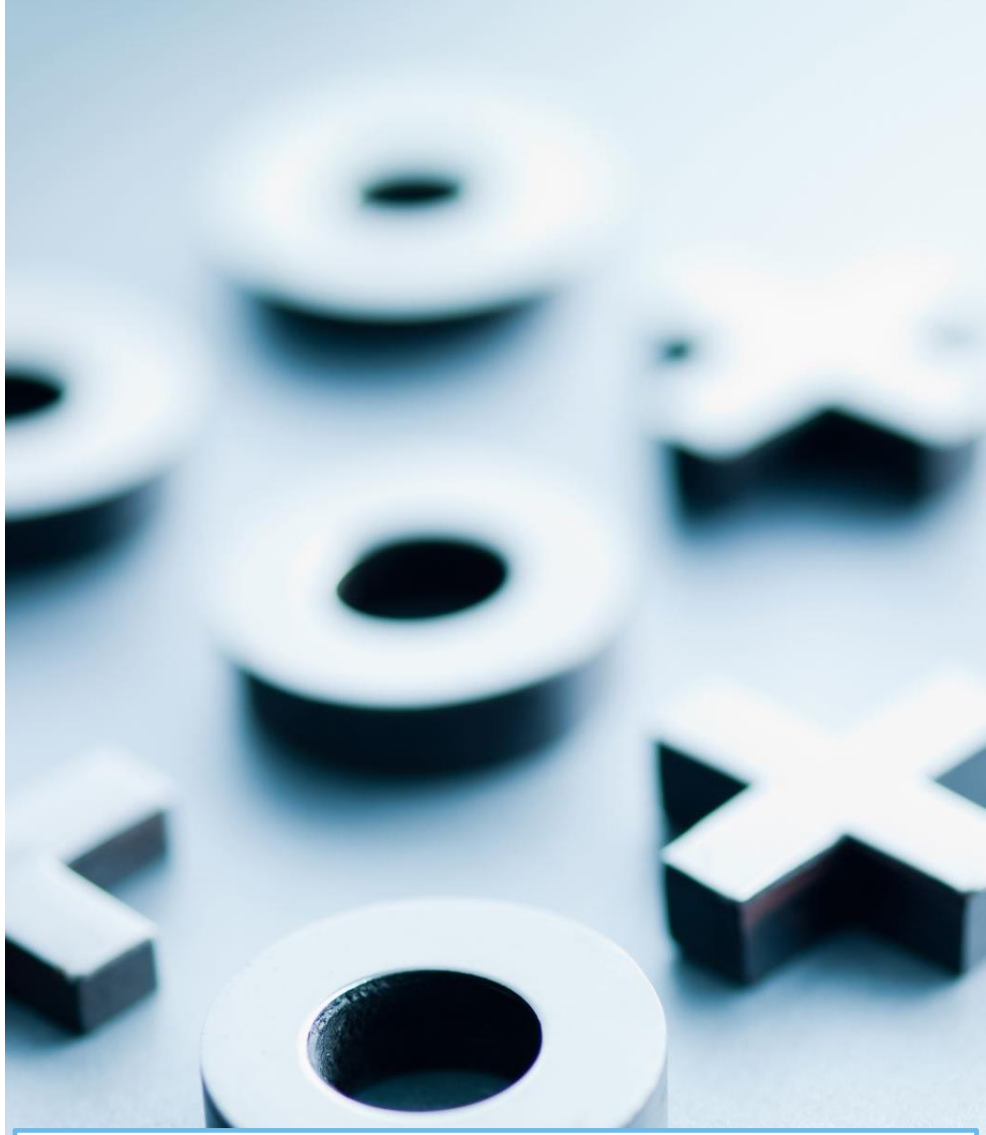



```
rng = np.random.default_rng(seed = 313)
x = np.linspace(0, 1, num = 100)
y = np.exp(3 * x ** 2) + rng.uniform(-1, 1, 100)

res_ls = smf.ols("y ~ x", data = pd.DataFrame([y, x]).T).fit()
lower = np.sum((y - np.mean(y)) ** 2)
upper = np.sum((y - (res_ls.params[0] + x * res_ls.params[1])) ** 2)
lower - upper # 1421.4833888194755
1 - upper / lower # 0.6554432151711667
```

- Fixed by dividing D by $\sum(y_i - m)^2$
- $R^2 = \frac{\sum(y-m)^2 - \sum(y-a+bx_i)^2}{\sum(y-m)^2} = 1 - \frac{\sum(y-(a+bx_i))^2}{\sum(y-m)^2}$

HOW MUCH BETTER CAN I PREDICT Y USING X THAN WITHOUT IT?

- $R^2 = \frac{\sum(y-m)^2 - \sum(y-a+bx_i)^2}{\sum(y-m)^2} = 1 - \frac{\sum(y-(a+bx_i))^2}{\sum(y-m)^2}$
- And in general: $\mathrm{R}^2 = 1 - \frac{\sum d(y_i, a+bx_i)}{\sum d(y_i, m)}$
- Since $0 \leq \sum d(y_i, a + bx_i) \leq \sum d(y_i, m)$, we have that $1 \geq R^2 \geq 0$.
- Bigger is better.

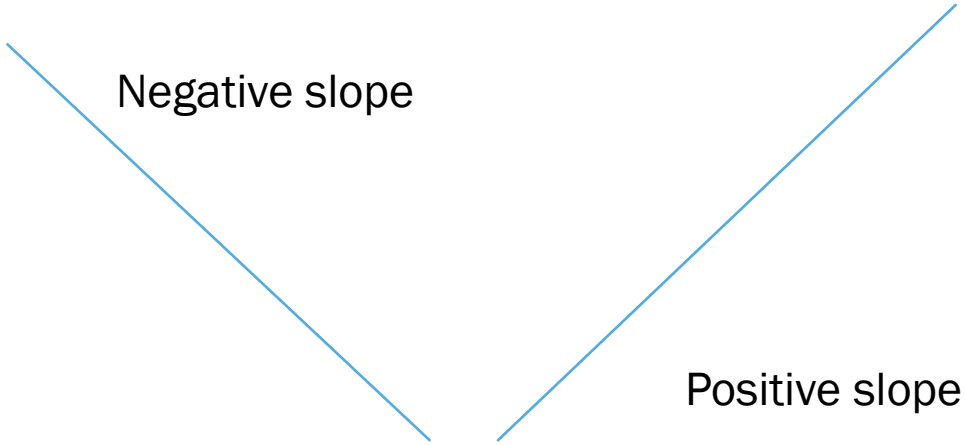


Important: When people talk about R^2, they talk about least squares almost always!

$$1 - \frac{\sum d(y_i, a + bx_i)}{\sum d(y_i, m)}$$

Korn, E. L., & Simon, R. (1991). Explained Residual Variation, Explained Risk, and Goodness of Fit. *The American Statistician*, *45*(3), 201–206. https://doi.org/10.1080/00031305.1991.10475802

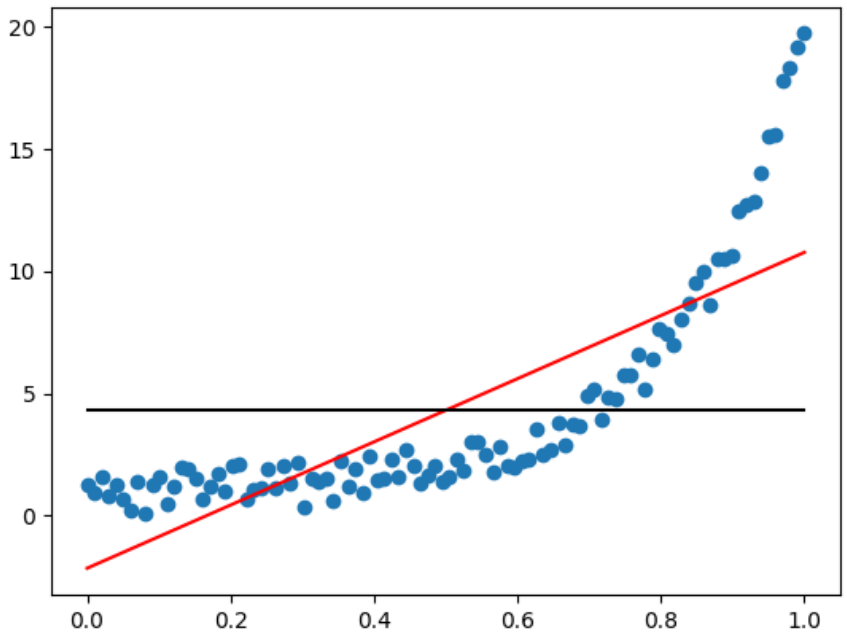
THE CORRELATION



- $\mathrm{Cor}(x,y) = \frac{\mathrm{Cov}(x,y)}{\mathrm{sd}(x)\mathrm{sd}(y)} = \frac{\sum(x_i-\overline{x})(y_i-\overline{y})}{[\sum(x_i-\overline{x})^2]^{\frac{1}{2}}[\sum(y_i-\overline{y})^2]^{\frac{1}{2}}}$
- The sign of the slope is positive if it points upwards.
- Negative if it points downwards.
- Denote it by $\mathrm{sign}(b)$.

$$\mathrm{Cor}(x,y) = \mathrm{sign}(b) \cdot \sqrt{R^2}$$
$$R^2 = \mathrm{Cor}^2(x,y)$$

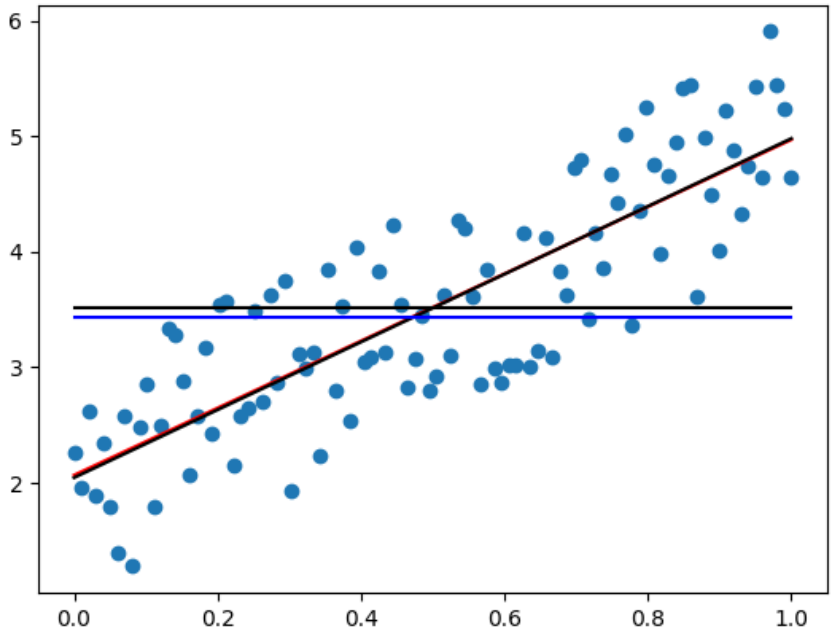
HAS NOTHING TO DO WITH THE REAL SHAPE OF THE REGRESSION MODEL!




```python
# rsq prototype 2
import numpy as np
import statsmodels.formula.api as smf
import pandas as pd
import matplotlib.pyplot as plt

res_ls = smf.ols("y ~ x", data = pd.DataFrame([y, x]).T).fit()
lower = np.sum((y - np.mean(y)) ** 2)
# 2168.736140549201
upper = np.sum((y - (res_ls.params[0] + x * res_ls.params[1])) ** 2)
# 747.2527517297256
1 - upper / lower # 0.6554432151711667
res_ls.rsquared # 0.6554432151711667
np.corrcoef(x, y)[0, 1]**2 # 0.6554432151711672

np.mean(y) # 4.304984335161987
plt.scatter(x, y)
plt.plot(x, res_ls.params[0] + x * res_ls.params[1], color = "red")
plt.plot(x, np.mean(y) + 0 * x , color = "black")
plt.show()
```

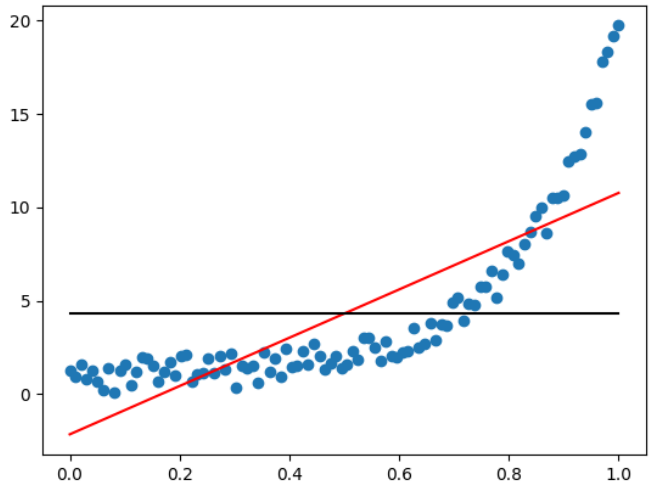
WHAT ABOUT LEAST ABSOLUTE DEVIATIONS?

```
# rsq prototype 1
import numpy as np
import statsmodels.formula.api as smf
import pandas as pd
import matplotlib.pyplot as plt

rng = np.random.default_rng(seed = 313)
x = np.linspace(0, 1, num = 100)
y = 2 + 3 * x + rng.uniform(-1, 1, 100)

res_ls = smf.ols("y ~ x", data = pd.DataFrame([y, x]).T).fit()
res_lad = smf.quantreg("y ~ x", data = pd.DataFrame([y, x]).T).fit(q = 0.5)

lower = np.sum(abs(y - np.median(y)))
upper = np.sum(abs(y - (res_lad.params[0] + x * res_lad.params[1])))
1 - upper / lower # 0.4282332871090164
res_ls.rsquared # 0.6888118053315937

plt.scatter(x, y)
plt.plot(x, res_ls.params[0] + x * res_ls.params[1], color = "red")
plt.plot(x, res_lad.params[0] + x * res_lad.params[1], color = "black")
plt.plot(x, np.mean(y) + 0 * x , color = "black")
plt.plot(x, np.median(y) + 0 * x , color = "blue")
plt.show()
```



BUT WHAT'S THE BEST PREDICTION I CAN DO OF Y GIVEN X, PERIOD?!

- You're not Bob – you don't believe that the graph below is linear!



«BOB, I'M GIVING YOU SOME DATA ON THE FORM $e^{ax^b} + u$.

I'M DEAD SERIOUS THIS TIME!»

WHAT SHOULD BOB DO?

MINIMIZE $\sum d(y_i, e^{ax_i^b})$ **OVER** a **AND** b**!**

WE WILL USE CURVE_FIT FROM SCIPY.OPTIMIZE

```
>>> from scipy.optimize import curve_fit
>>> help(curve_fit)
Help on function curve_fit in module scipy.optimize.minpack:

curve_fit(f, xdata, ydata, p0=None, sigma=None,
absolute_sigma=False, check_finite=True, bounds=(-inf, inf),
method=None, jac=None, **kwargs)
    Use non-linear least squares to fit a function, f, to data.
```

You must use a computer routine; there's typically no closed form solution!

This is a least squares procedure!

USING THE FUNCTION

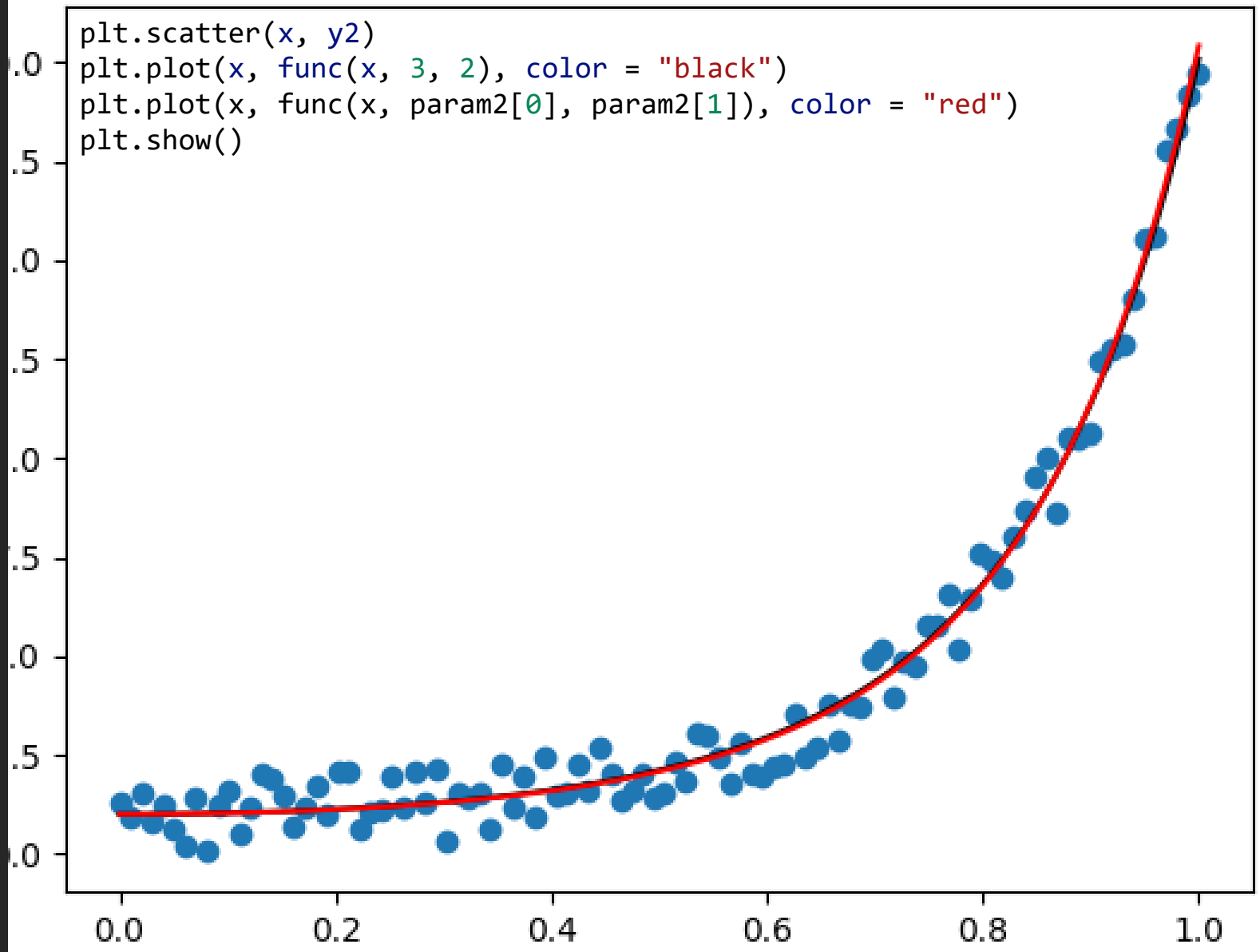
```
from scipy.optimize import curve_fit


rng = np.random.default_rng(seed = 313)
x = sort(np.linspace(0, 1, num = 100))
u = rng.uniform(-1, 1, 100)
y1 = 2 + 3 * x + u # prototype 1
y2 = np.exp(3 * x ** 2) + u # prototype 2

# The function Alice gave Bob!
func = lambda x, a, b: np.exp(a * x ** b)

>>> curve_fit(func, x, y2)
(array([3.01678888, 2.04035717]),
array([[0.00019554, 0.00040825],
       [0.00040825, 0.00209514]]))
```

- The first component, `([3.01678888, 2.04035717])`, are out a and b.
- The second is the variance-covariance matrix, which may be used for inference.

- The function is a function of the data x and as many parameters as we want.

- Notice how well it approximates the correct parameters, 3 and 2!

CAN YOU SPOT THE BLACK LINE?



AND WE CAN CALCULATE THE NON-LINEAR R^2_{nls}

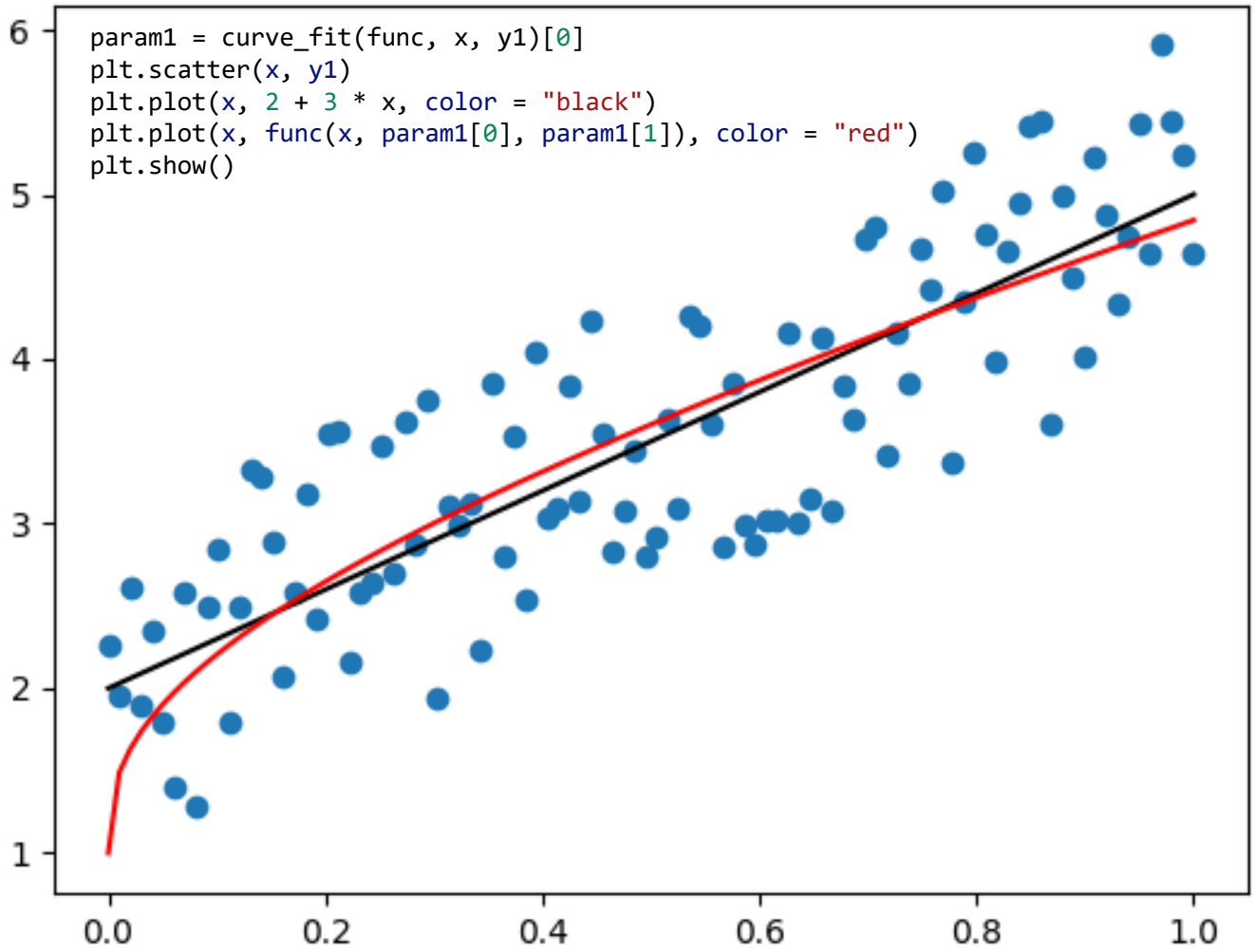
```
lower = np.sum((y2 - np.mean(y2)) ** 2) # 2168.736140549201
upper = np.sum((y2 - func(x, param2[0], param2[1])) ** 2) # 32.016818192175435
1 - upper / lower
```

```
>>> 1 - upper / lower
0.9852371076436861

>>> np.corrcoef(x, y2)[1, 0] ** 2
0.6554432151711672
```

Much more impressive than 0.66 from the linear model!

(Such a high R^2_{nls} is quite rare though.)

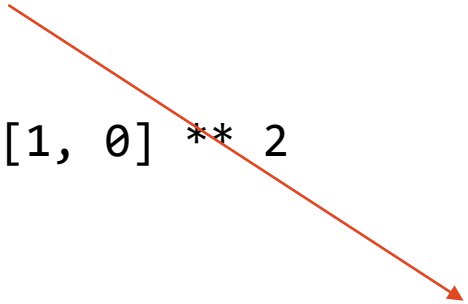
WHAT ABOUT THE LINEAR MODEL?



WE CAN CALCULATE THE NON-LINEAR R^2

```
lower = np.sum((y2 - np.mean(y2)) ** 2) # 2168.736140549201
upper = np.sum((y2 - func(x, param2[0], param2[1])) ** 2) # 32.016818192175435
1 - upper / lower
```

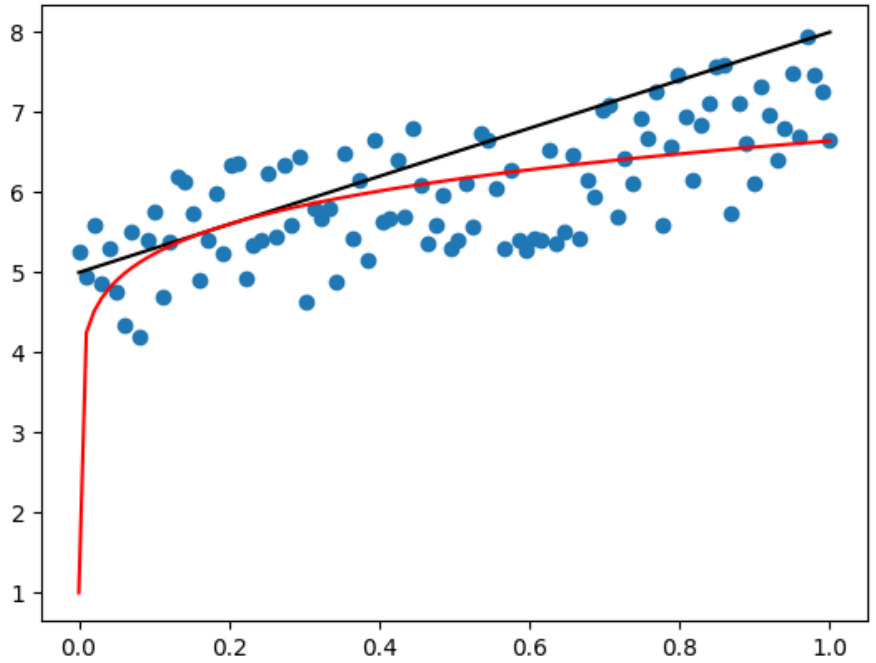
```
>>> 1 - upper / lower
0.9852371076436861

>>> np.corrcoef(x, y2)[1, 0] ** 2
0.6554432151711672
```



The flexibility of the model allowed it's R^2 to be bigger!

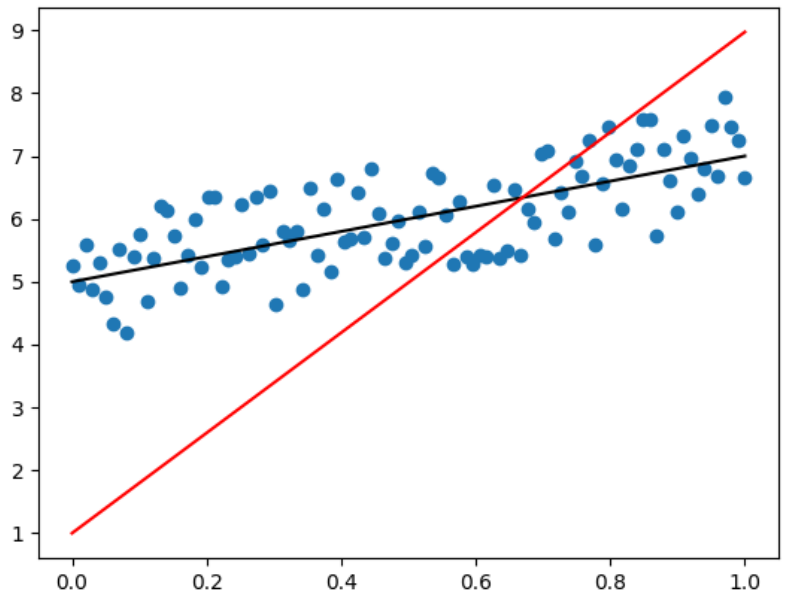

IS THIS ALWAYS THE CASE?

BAD NON-LINEAR LEAST SQUARES




```
rng = np.random.default_rng(seed = 313)
x = sort(np.linspace(0, 1, num = 100))
u = rng.uniform(-1, 1, 100)
y = 5 + 2 * x + u
param = curve_fit(func, x, y)[0]

plt.scatter(x, y)
plt.plot(x, 5 + 3 * x, color = "black")
plt.plot(x, func(x, param[0], param[1]), color = "red")
plt.show()

lower = np.sum((y - np.mean(y)) ** 2)
upper = np.sum((y - func(x, param[0], param[1])) ** 2)

>>> 1 - upper / lower
0.12462613272055834
>>> np.corrcoef(x, y)[1, 0] ** 2
0.4875571269866591
```

HORRIBLE LEAST SQUARES




```
rng = np.random.default_rng(seed = 313)
x = sort(np.linspace(0, 1, num = 100))
u = rng.uniform(-1, 1, 100)
y = 5 + 2 * x + u

func = lambda x, a, b: a*x + np.exp(b*x)
param = curve_fit(func, x, y)[0]

plt.scatter(x, y)
plt.plot(x, 5 + 2 * x, color = "black")
plt.plot(x, func(x, param[0], param[1]), color = "red")
plt.show()

lower = np.sum((y - np.mean(y)) ** 2)
upper = np.sum((y - func(x, param[0], param[1])) ** 2)

>>> 1 - upper / lower
-6.153148272325556
>>> np.corrcoef(x, y)[1, 0] ** 2
0.4875571269866591
```

THE NON-LINEAR ESTIMATOR DOES *MUCH* WORSE THAN THE MEAN. HENCE $R^2_{nls} = -6$!

CAN YOU MAKE $1 \geq R^2_{nls} \geq 0$ ALWAYS?

- Make sure the function you optimize can be equal to any constant c!
- Why is this the case?
  - $\sum d(y_i, ax_i + e^{bx_i})$ may be larger than $\sum d(y_i, m)$
  - But $\sum d(y_i, c + ax_i + e^{bx_i}) \leq \sum d(y_i, m)$.
  - For if $a = 0$ and $b = 0$ and $c = m - 1$, then $c + ax_i + e^{bx_i} = m$.

WORKS ON *PARAMETERIZED* FUNCTIONS

- Let $f(x; a_1, a_2, \dots, a_p)$ be a parameterized function.
  - $e^{a_1 x} + e^{a_2 x^2} + a_3 x,$
  - $a_1 + a_2 x$
  - $a_1 x^2 + a_2 x^3 + e^{a_3 x}$
- x is data
- $a_1, a_2, \dots, a_p$ are parameters modiying how the function works.
- We know the data but we don't know the parameters!

PROS OF NON-LINEAR LEAST SQUARES

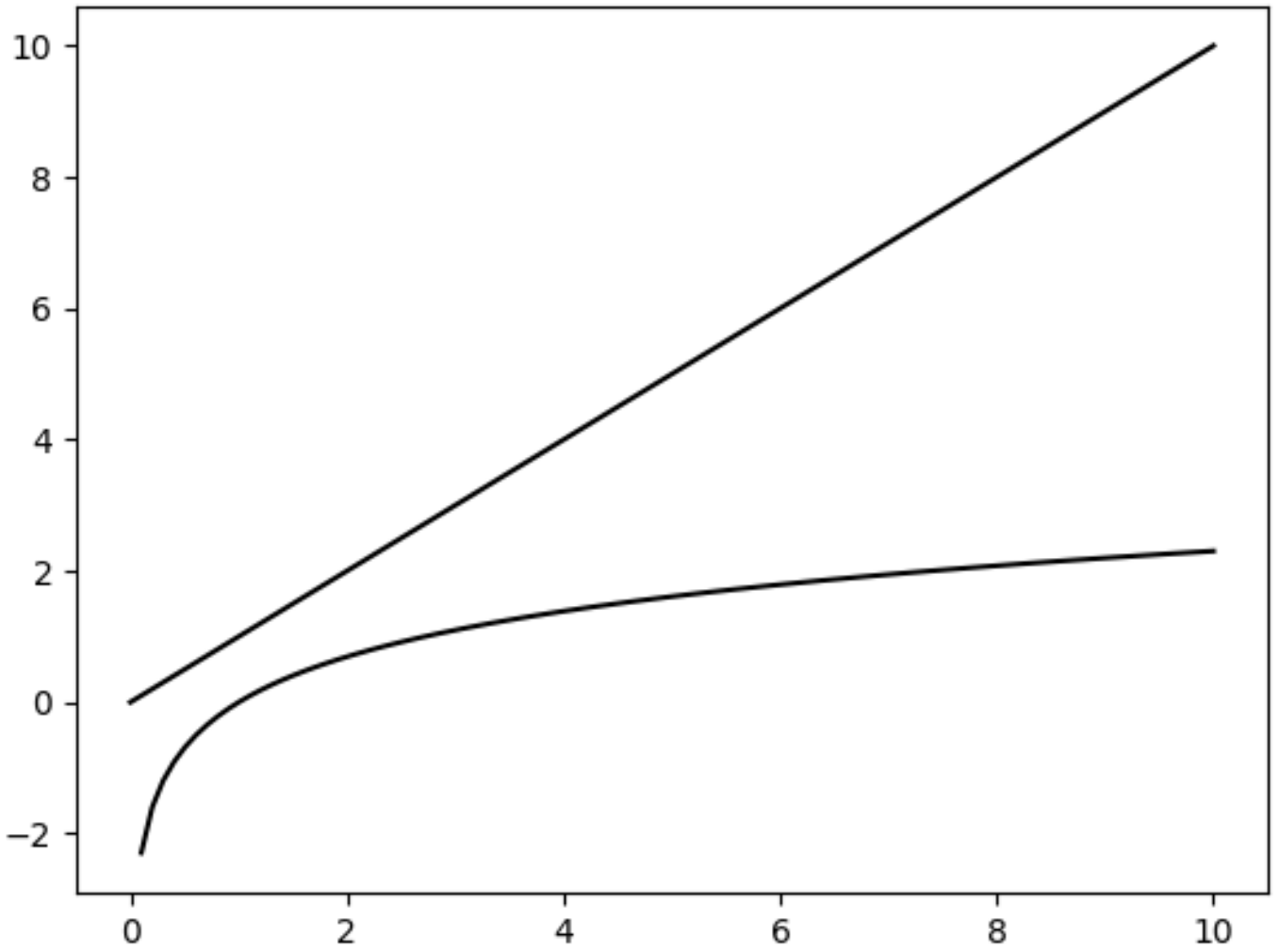
- Can use any function shape!
- Sometimes we know the shape of a function.
  - Especially in chemistry, physics, pharmacology.
  - Real sciences!
- Is easy to understand and use.
- Inference is easy enough; we won't cover it though.

CONS OF NON-LINEAR LEAST SQUARES

- Slower!
- Unique solution? Any solution? Hard to say! (Not that important to practically minded people!)
- Numerically not well-behaved; may have to transform data. May act unexpectedly!
- Not well-known.
- You need to guess or know the functional form. See here for examples.

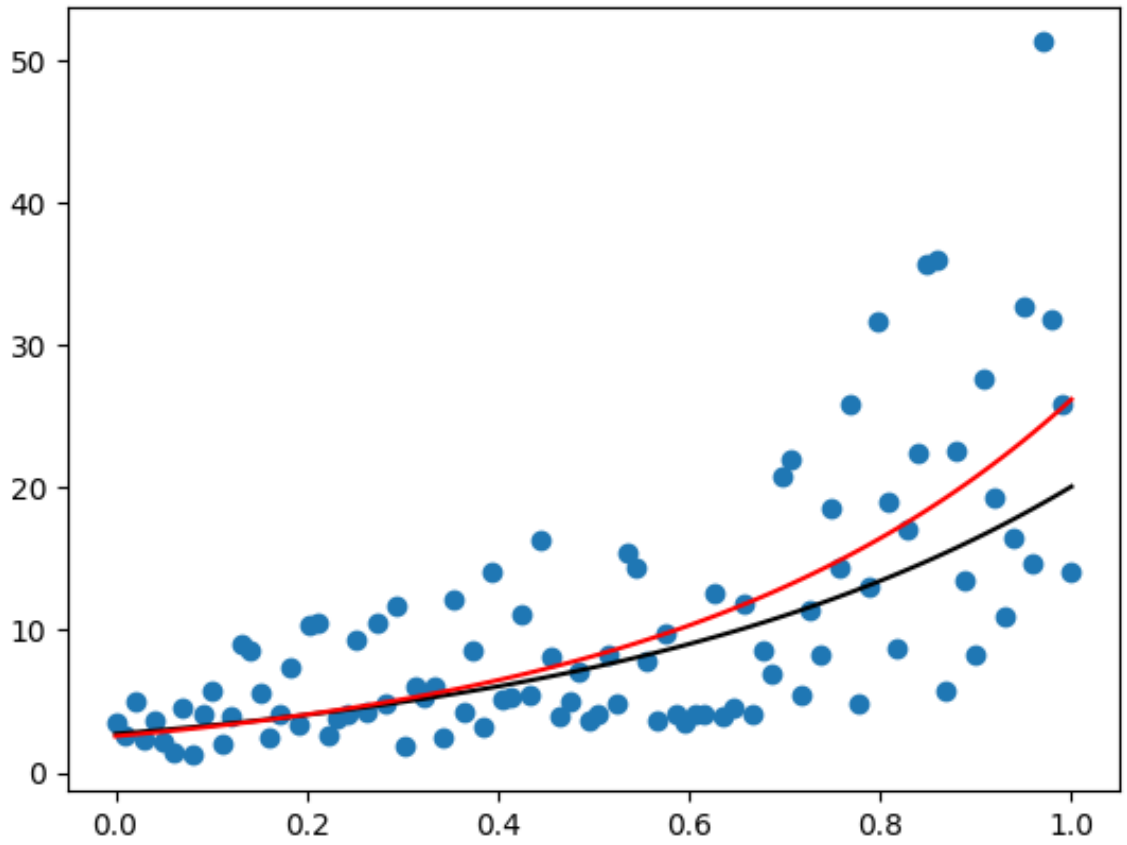
THE LOGARITHM

- Inverse of the exponential function.
- $\log e^x = e^{\log x} = x$
- Grows very slowly!
- Graph shows $\log x$ and x.



$$y = e^{1+2x+u}$$

```
rng = np.random.default_rng(seed = 313)
x = np.linspace(0, 1, num = 100)
u = rng.uniform(-1, 1, 100)
y = np.exp(1 + 2 * x + u)

func = lambda x, a, b: np.exp(a + b * x)
param = curve_fit(func, x, y)[0]

plt.scatter(x, y)
plt.plot(x, func(x, 1, 2), color = "black")
plt.plot(x, func(x, *param), color = "red")
plt.show()
```

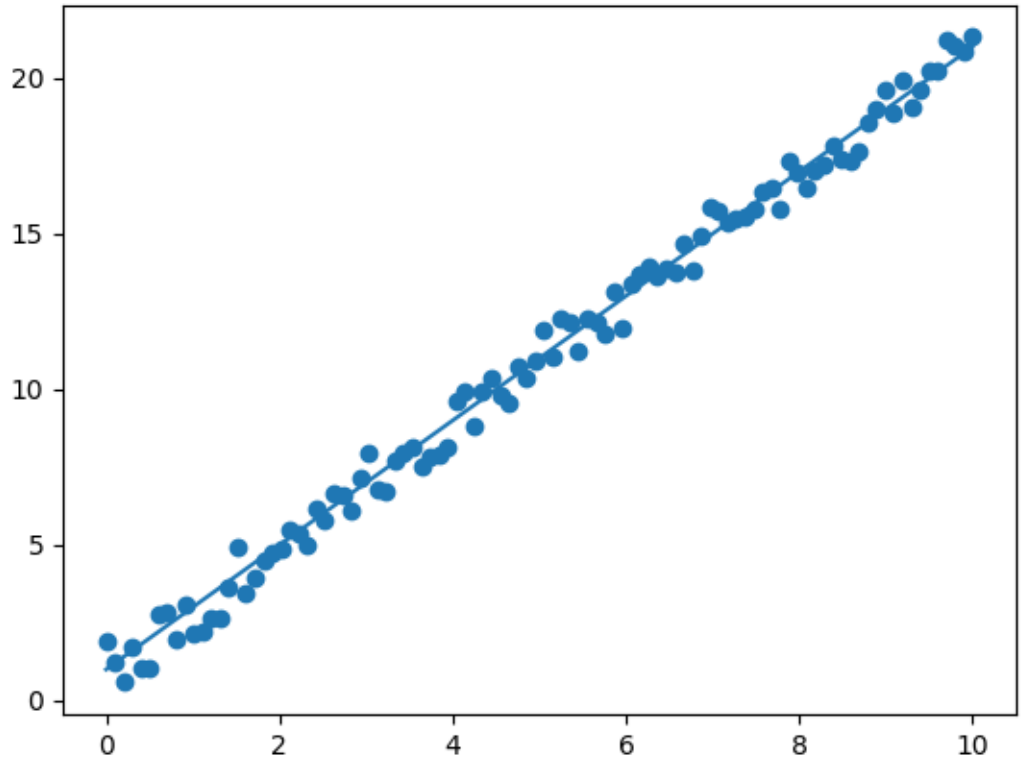


Multiplicative errors!

WHAT ABOUT THIS ONE?

$$\log y = 1 + 2x + u$$

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, num = 100)
u = rng.uniform(-1, 1, 100)
y = np.exp(1 + 2 * x + u)

plt.scatter(x, np.log(y))
plt.plot(x, 1 + 2 * x)
plt.show()
```



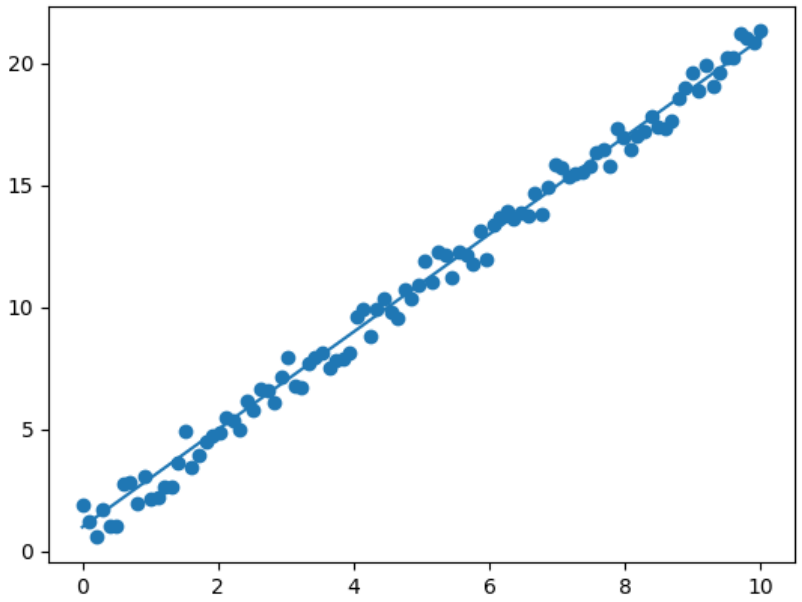

DO WE WANT TO TALK ABOUT e^{1+2x+u} OR $1+2x+u$?

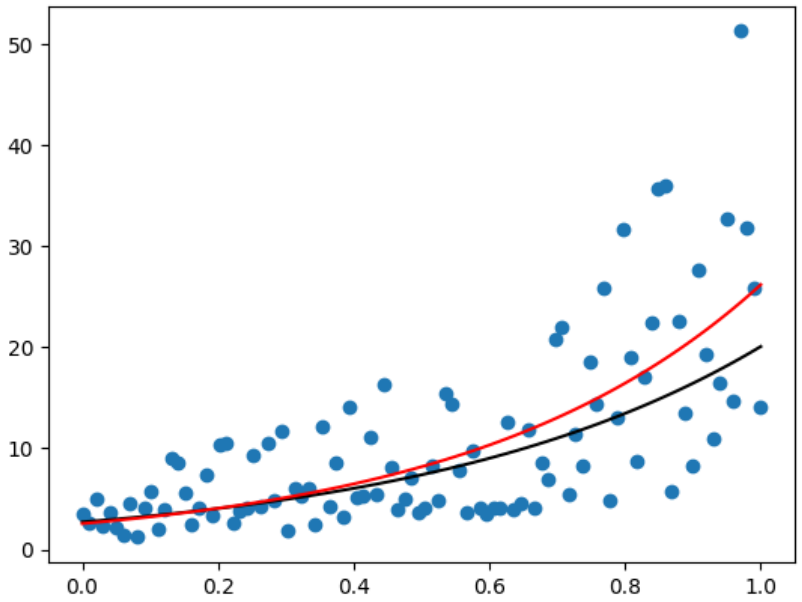
DIFFERENCE IN R^2

```
rng = np.random.default_rng(seed = 313)
x = np.linspace(0, 1, num = 100)
u = rng.uniform(-1, 1, 100)
y = np.exp(1 + 2 * x + u)

func = lambda x, a, b: np.exp(a + b * x)
param = curve_fit(func, x, y)[0]

lower = np.sum((y - np.mean(y)) ** 2)
upper = np.sum((y - func(x, param[0], param[1])) ** 2)
1 - upper / lower
np.corrcoef(x, np.log(y))[1, 0] ** 2
```

```
>>> 1 - upper / lower
0.491781632329627
>>> np.corrcoef(x, np.log(y))[1, 0] ** 2
0.487557126986659
```

Why? More variability before taking logs!





THE SCALE IS LARGER ON THE NON-TRANSFORMED PLOT! SO THE MEAN EXPLAINS LESS.

DIFFERENCE IN PARAMETER ESTIMATES

```
func = lambda x, a, b: np.exp(a + b * x)
param = curve_fit(func, x, y)[0]
>>> param
array([1.37297845, 1.77035421])

res_ls = smf.ols("np.log(y) ~ x", data = pd.DataFrame([y, x]).T).fit()
>>> res_ls.params
Intercept    1.079184
x            1.999232
dtype: float64
```

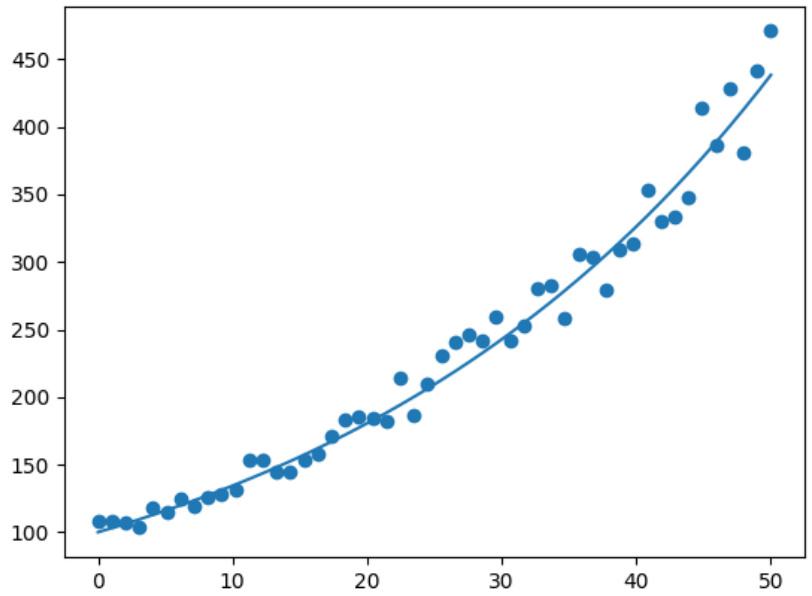
The parameter estimates are closer to 1 and 2 in the log-transformed model. This is true in general.

WHY DOES THIS HAPPEN? COMPLICATED REASONS!

BUT WITHOUT THE TRANSFORMATION, THE ESTIMATION WILL PUT TOO MUCH WEIGHT THE OBSERVATIONS WITH HIGH x.

I may tell you why the logarithmic transform works better later on. The keyword is «efficiency».

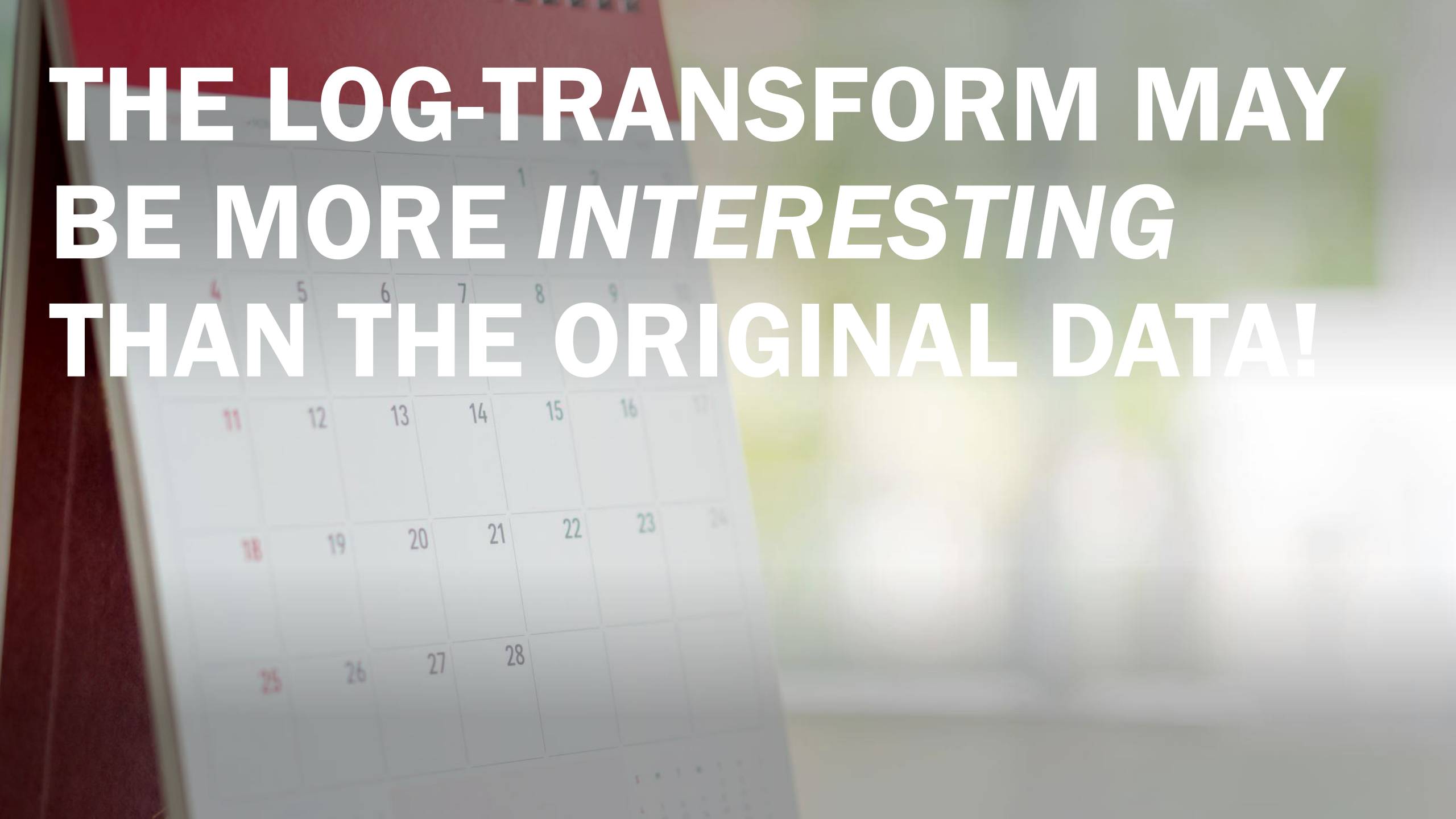
WHEN DOES IT HAPPEN? MORE OFTEN THAN YOU'D THINK!



- Think about interest rates and compound interest.
- Suppose the interest rate is 1.03.
- If you invest 100 NOK today, you'll have $y = 100 \cdot 1.03^x$ in x years.
- But there is some random noise in the data, so that have $y = 100 \cdot 1.03^x \cdot v$
- Taking logs yields $\log y = \log 100 + x \log 1.03 + \log v$

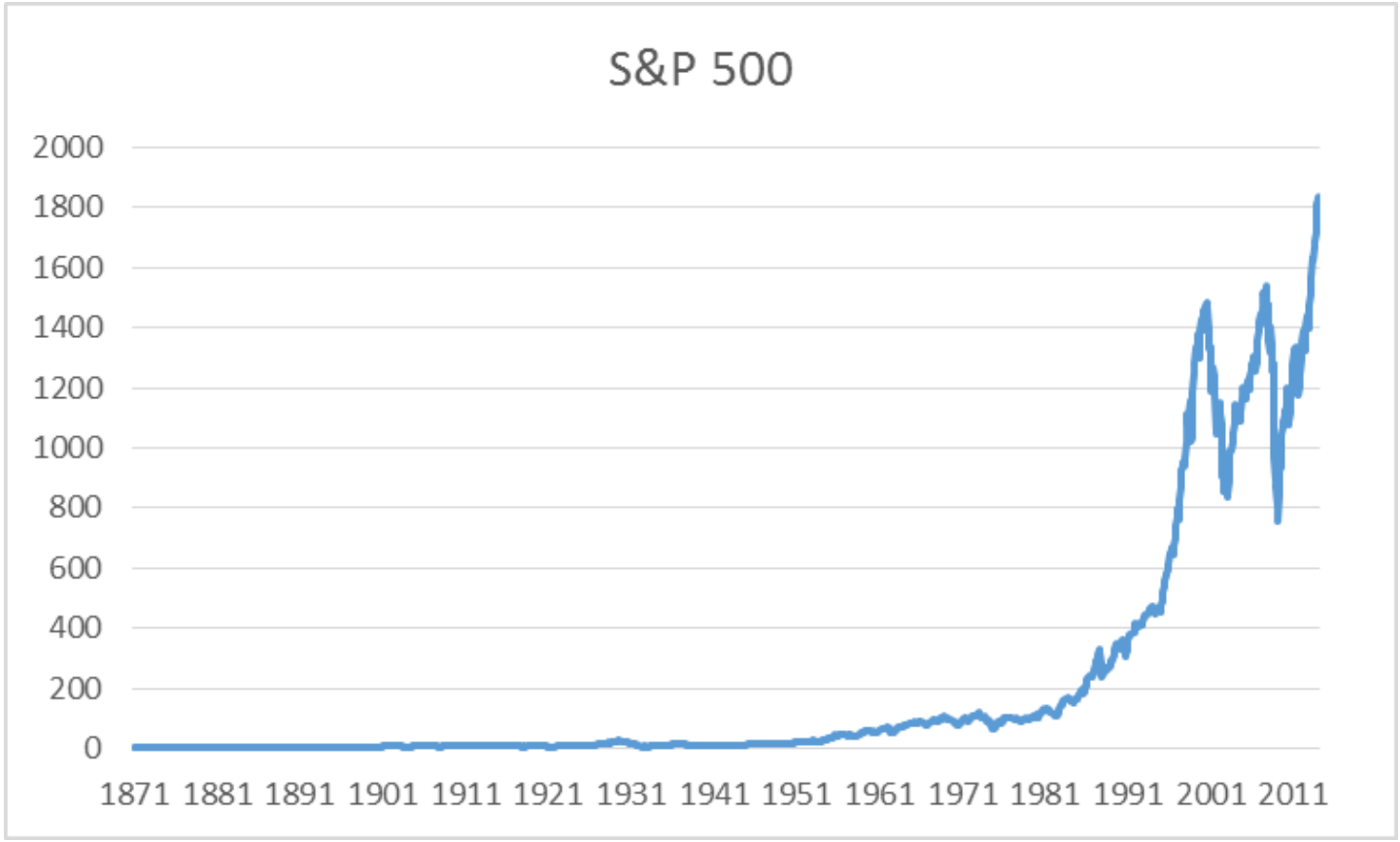
YOU WOULD WANT TO LOG-TRANSFORM IF YOU WANT TO ESTIMATE THE INTEREST RATE.

WHY? THE INTEREST RATE IS A PARAMETER. AND YOU WILL ESTIMATE IT BETTER WITH THE LOG-TRANSFORM.


THE LOG-TRANSFORM MAY BE MORE *INTERESTING* THAN THE ORIGINAL DATA!

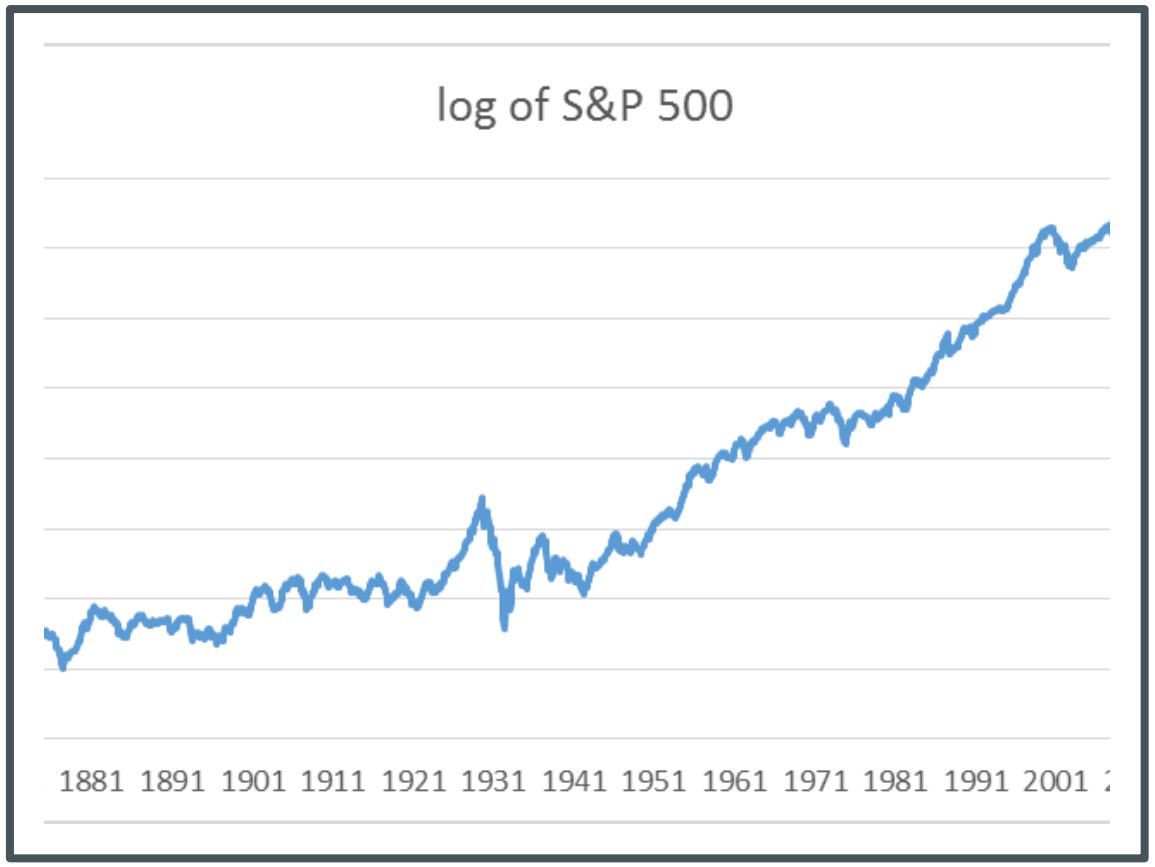
S&P500

- Source: https://econbrowser.com/archives/2014/02/use-of-logarithms-in-economics (James Hamilton)



"Plotted on this scale, one can see nothing in the first century, whereas the most recent decade appears insanely volatile."

LOGARITHM OF S&P500



- "On the other hand, if you plot these same data on a log scale, a vertical move of 0.01 corresponds to a 1% change at any point in the figure. Plotted this way, it's clear that, in percentage terms, the recent volatility of stock prices is actually modest relative to what happened in the Great Depression in the 1930's."

LINEAR MODELS ARE MUCH MORE FAMILIAR!

LINEAR MODEL WILL HAVE MORE PRECISE ESTIMATES TOO, PROVIDED THE EXPONENTIAL MODEL IS TRUE!

AND MAY BE MORE INTERESTING.

IF YOU’RE NOT SURE, TRANSFORM.


BYE BYE