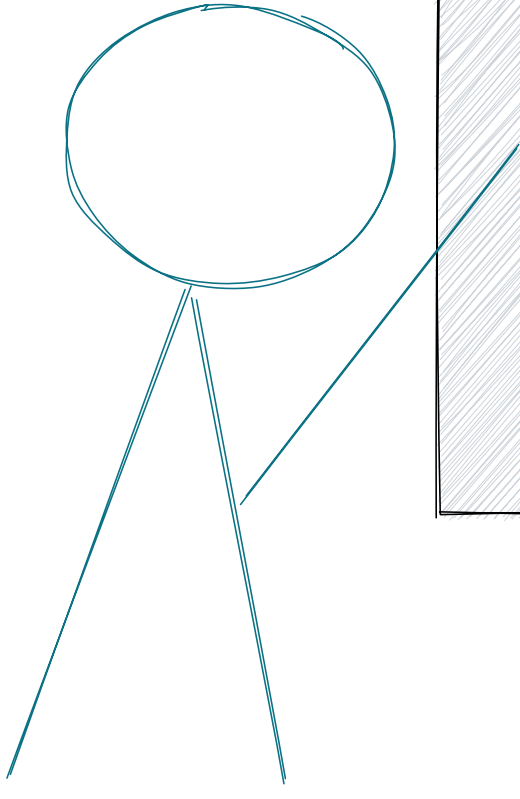


Simple linear regression

EBA3500: Data analysis with programming

Lecturer: Jonas Moss
Contact: jonas.moss@bi.no

Office hours:

What is "linearity"?

lin·ear

['lɪnɪə]

ADJECTIVE

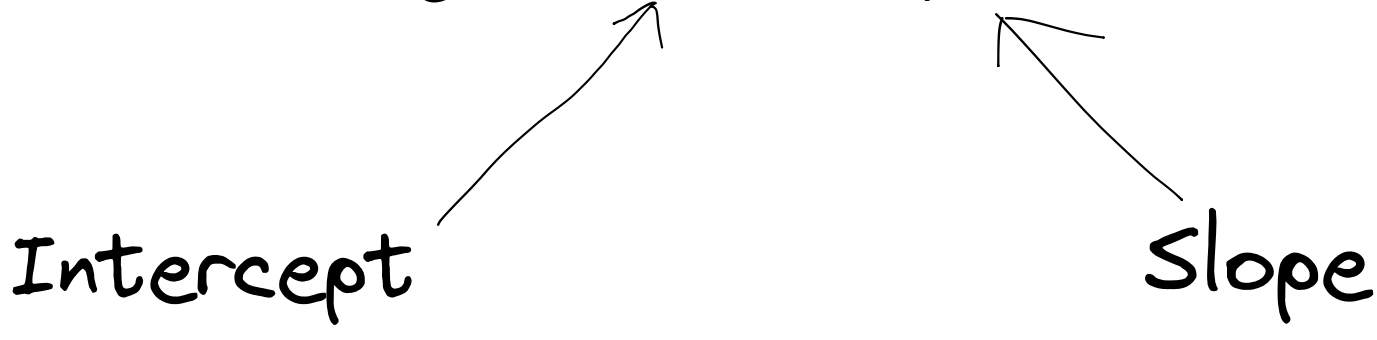
1. arranged in or extending along a straight or nearly straight line:
   "linear arrangements" · [More]
   synonyms: unswerving · undeviating · direct · as straight as an arrow · uncurving · unbending

$$\vec{y} = A\vec{x} + b$$

For us, linearity means this!

Linearity in two dimensions

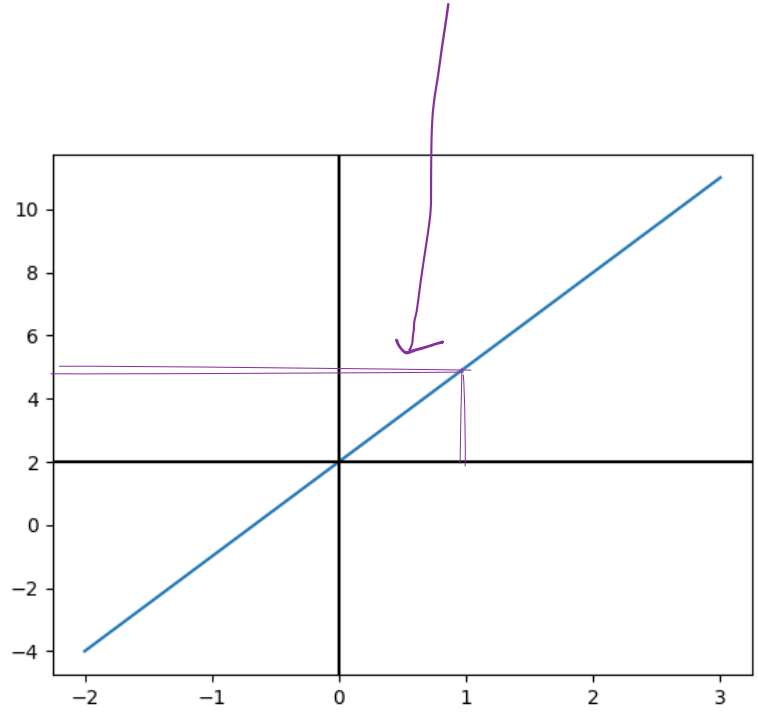
$$y = \alpha + \beta x$$



```
import numpy as np
import matplotlib.pylab as plt
x = np.linspace(-2, 3, 1000)
plt.plot(x, 2 + 3 * x)
plt.axvline(x=0, color = "black")
plt.axhline(y=2, color = "black")
plt.show()
```

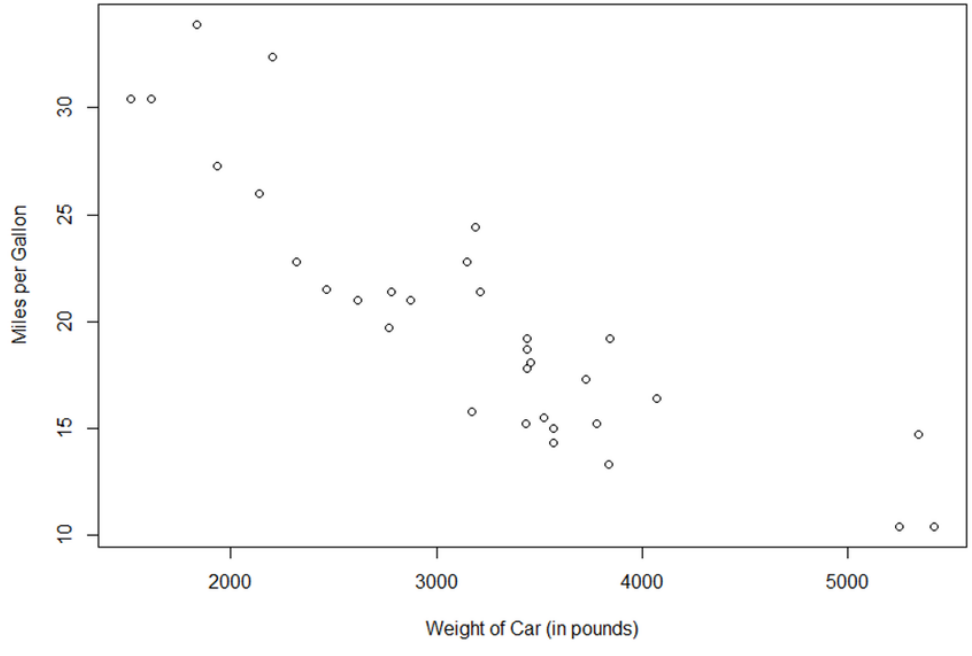
The intercept is 2

Increasing x by one increases y by 3!
Slope = 3.



A scatterplot

- "Weight of Car" is on the x-axis
  - feature
  - independent variable
  - explanatory variable
  - regressor
  - covariate
  - predictor
- "Miles per gallon" on the y-axis.
  - target
  - outcome
  - dependent variable
  - response

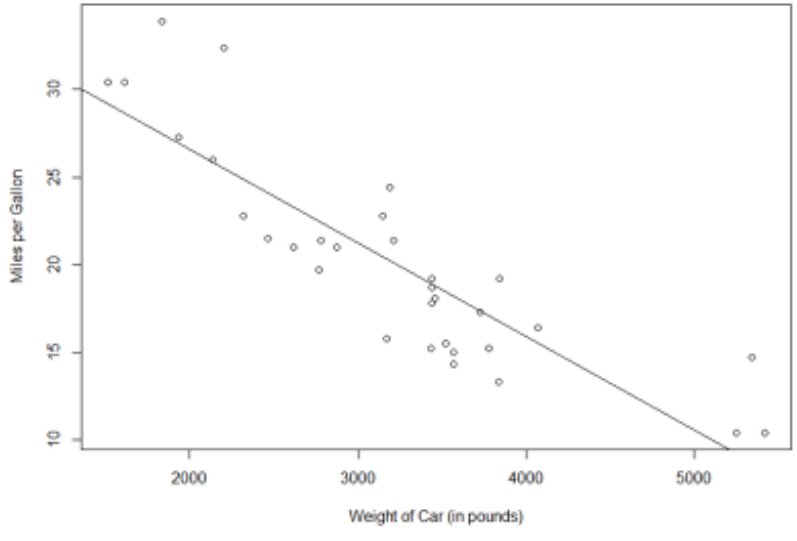


All of these words are often used! Try to remember them!

Linear regression

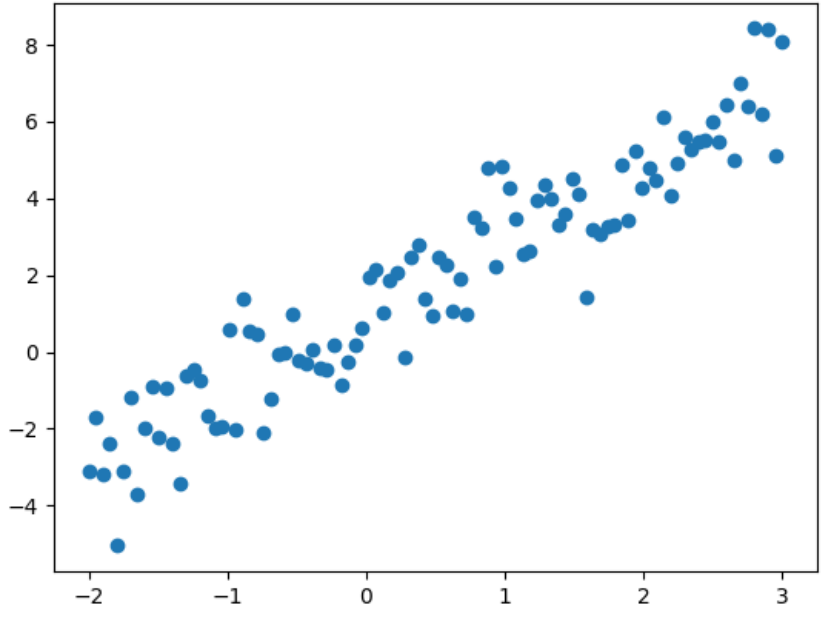
- $y = a + bx$
- a is the intercept
- b is the slope
- Here:
  - $a = 37$
  - $b = -5.3$

Wikipedia: "**Linearity** is the property of a mathematical relationship (*function*) that can be graphically represented as a straight line."



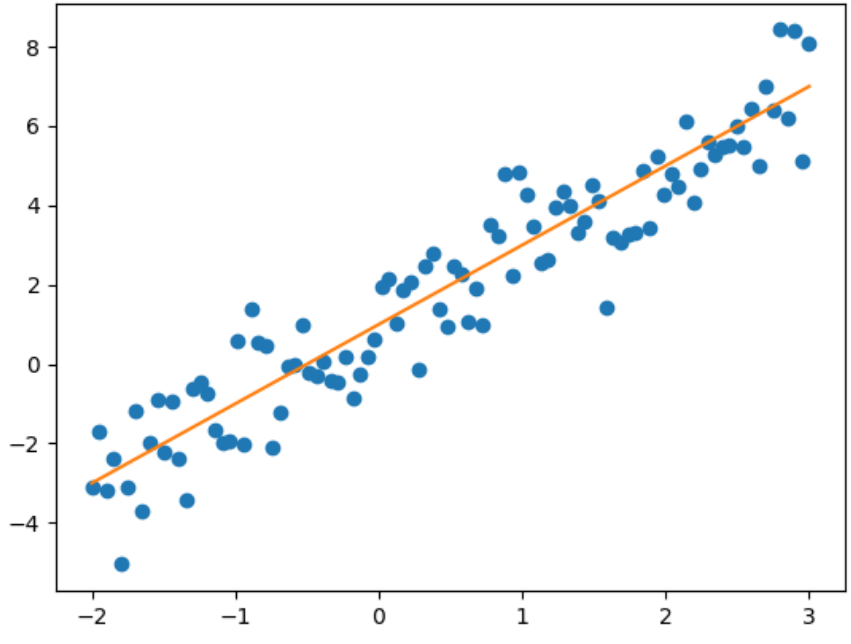
```
rng = np.random.default_rng(313)
x = np.linspace(-2, 3, 100)
y = 1 + 2 * x + rng.normal(0, 1, 100)
plt.plot(x, y, "o")
plt.show()
```

Not exactly on a line,
but pretty close!



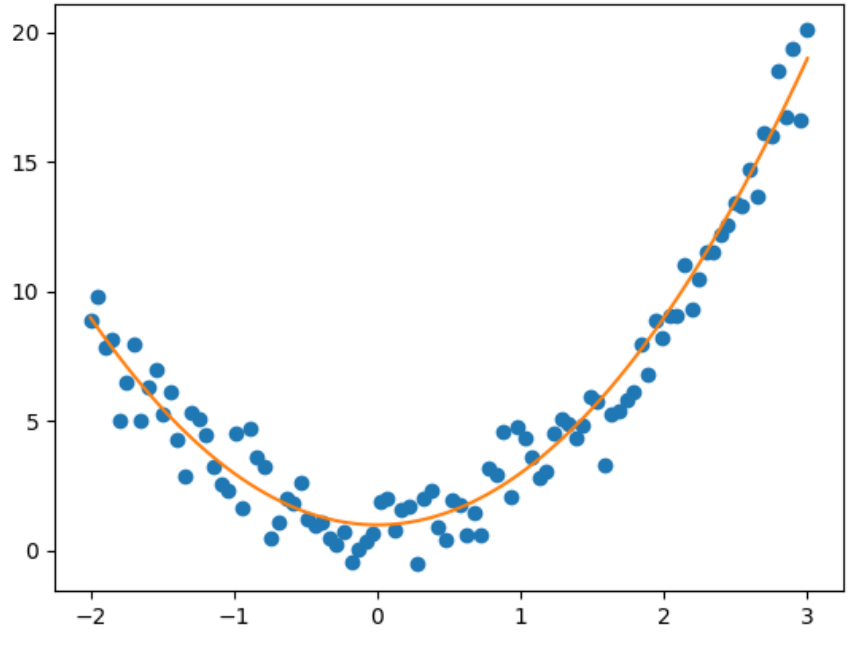
```
rng = np.random.default_rng(313)
x = np.linspace(-2, 3, 100)
y = 1 + 2 * x + rng.normal(0, 1, 100)
plt.plot(x, y, "o")
plt.plot(x, 1 + 2 * x)
plt.show()
```

Yeah, let's add a line!



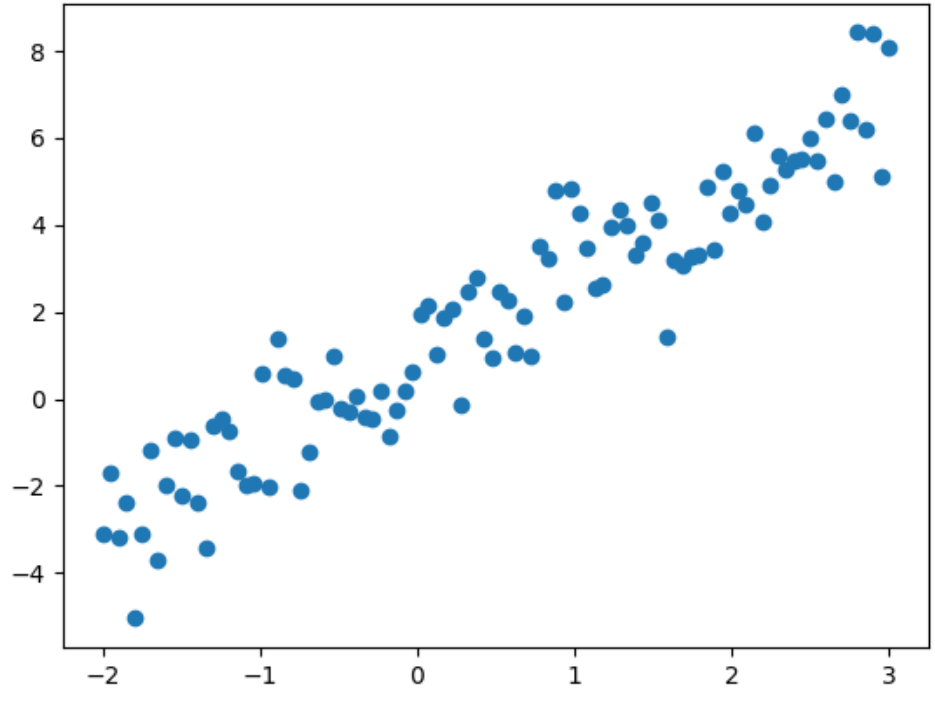
Possible with any function, really.

```
rng = np.random.default_rng(313)
x = np.linspace(-2, 3, 100)
y = 1 + 2 * x ** 2 + rng.normal(0, 1, 100)
plt.plot(x, y, "o")
plt.plot(x, 1 + 2 * x ** 2)
plt.show()
```

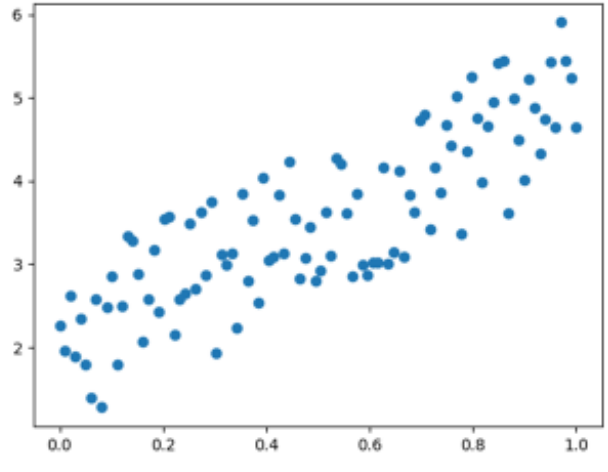


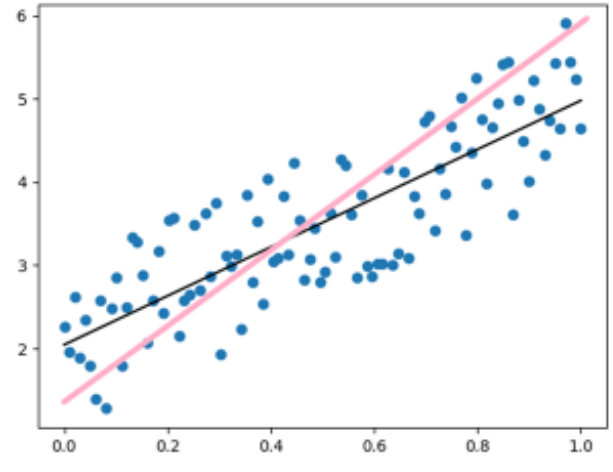
Like quadratics!

How you can you recover the true line from the data?



How do we get from A to B?

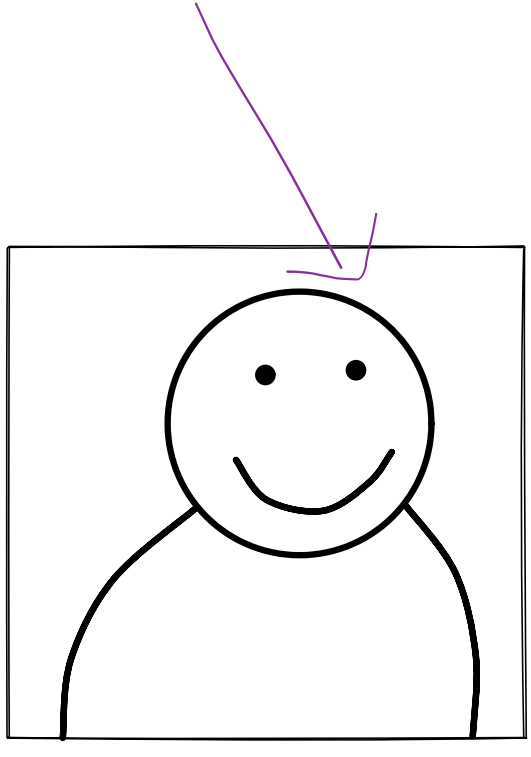




You can eyeball the line!

There is a *true linear relationship*.

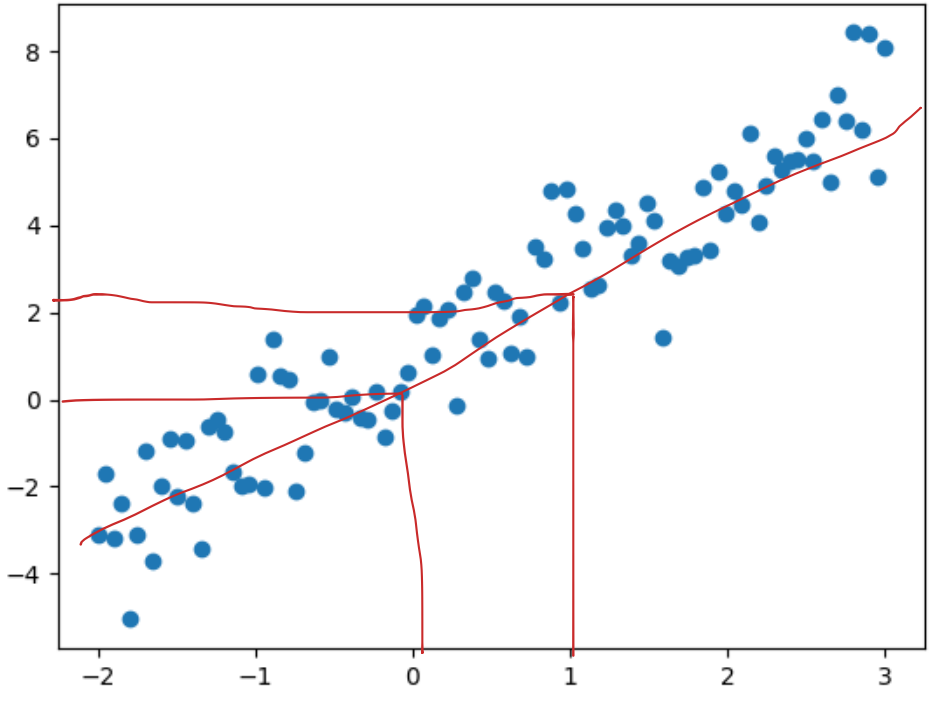
This is Bob.

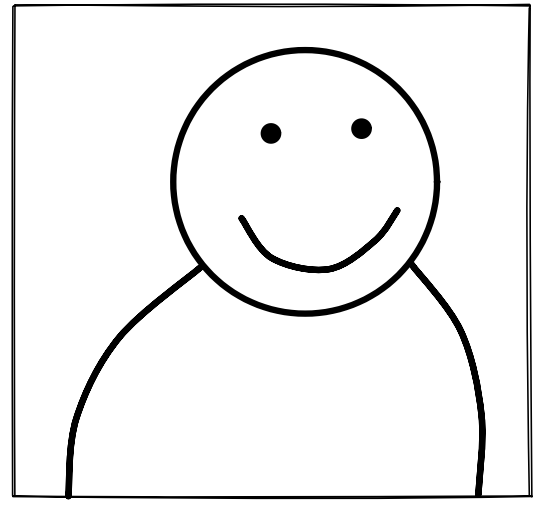


I just wanna draw a line!

(Cuz I'm artsy and lazy. ;))

"The intercept is approximately 0 and the slope is approximately 2."





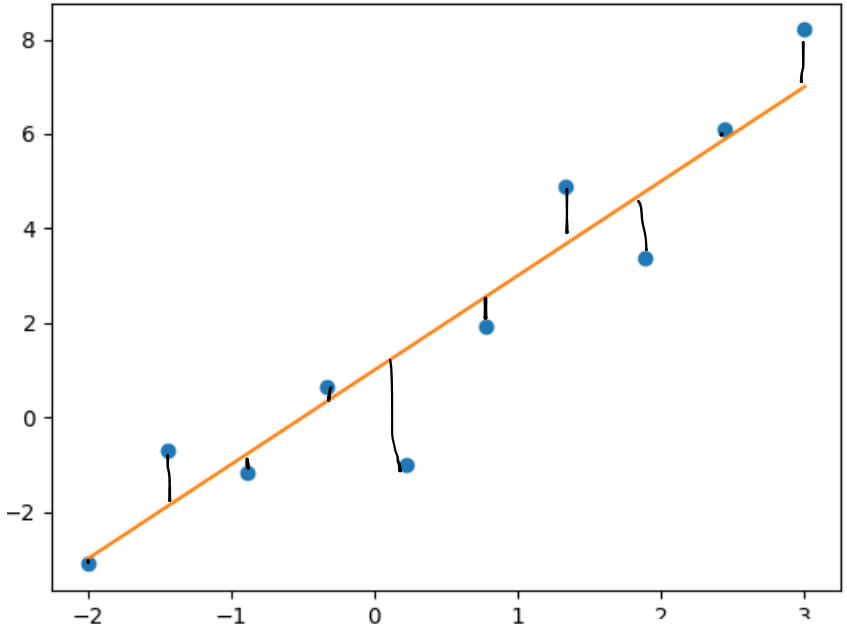
Feel free to do this on your spare time!
Whatever floats your boat.

Idea:

Minimize a "distance measure" between $\alpha+\beta x$ and the data y.

(Oh golly, this is an idea you should take seriously.)

Here we measure the absolute distance.



We use the true line right now.

```
np.abs(1 + 2*x - y).sum()
```

Does the true line minimize the distance?

```
def dist(x, y, a, b):
    """Calculate the total distance between a + bx and y."""
    return np.abs(a + b*x - y).sum()
```

```
bb = np.linspace(-3, 3, 1000)
aa = np.linspace(-3, 3, 1000)

minimal = np.inf
for a in aa:
    for b in bb:
        current = dist(x, y, a, b)
        if current < minimal:
            minimal = current
            a_solution, b_solution = a, b
```

Then a_solution, b_solution, are minimizers of the distance!

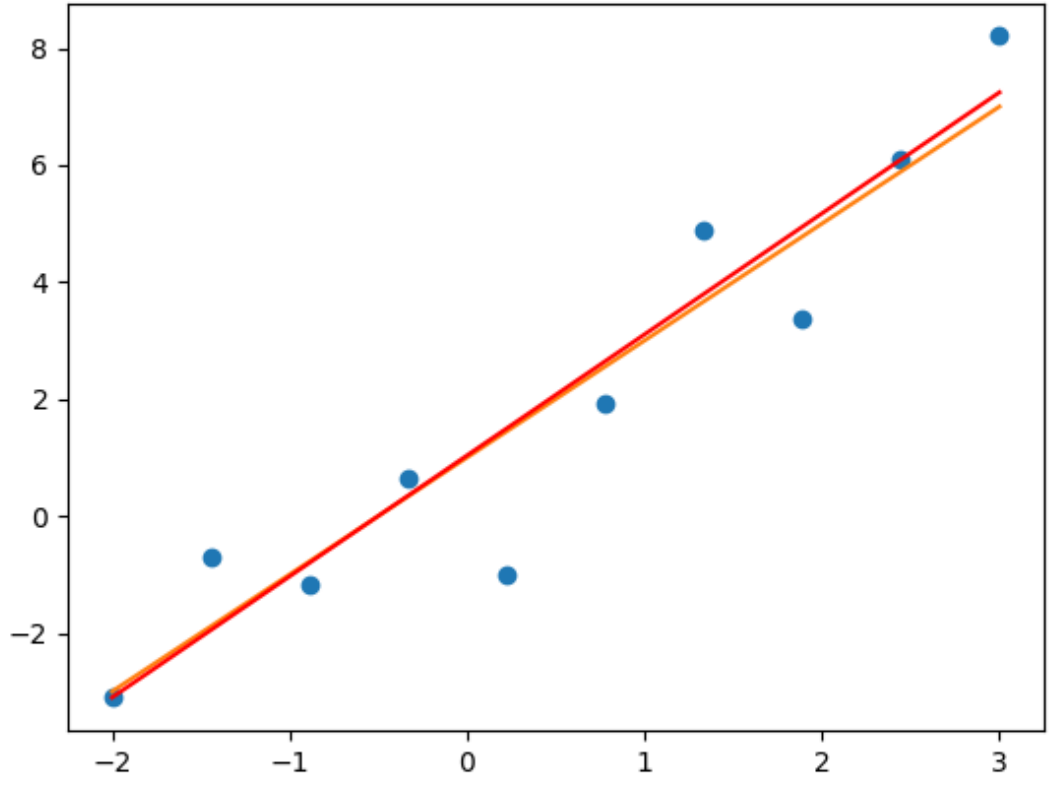
```
>>> a_solution
1.036036036036357
>>> b_solution
2.069069069069694
```

This is not 1 and 2!

But really close, right?

Let's plot it again!

```
rng = np.random.default_rng(313)
x = np.linspace(-2, 3, 10)
y = 1 + 2 * x + rng.normal(0, 1, 10)
plt.plot(x, y, "o")
plt.plot(x, 1 + 2 * x)
plt.plot(x, a_solution + b_solution * x, color = "red")
plt.show()
```



The distance minimizer did a great job!

Don't you think?

We have minimized the absolute distance!

$$(\hat{\alpha}, \hat{\beta}) = \underbrace{\text{argmin}_{\alpha,\beta}} \sum \underbrace{|y - (\alpha + \beta x)|}$$

These are estimators.

Finds the parameters that minimizes what's on the left

This is the absolute value distance

This minimizer is called
"least absolute deviations".

Estimating the parameters describing the relationsship between a response y and covariates x is called regression!

But actually, the
absolute value distance isn't used very much!

$$(\hat{\alpha}, \hat{\beta}) = \text{argmin}_{\alpha,\beta} \sum |y - (\alpha + \beta x)|$$

Special case of:

$$(\hat{\alpha}, \hat{\beta}) = \text{argmin}_{\alpha,\beta} \sum d(y, \alpha + \beta x)$$

a distance function!

$$(\hat{\alpha}, \hat{\beta}) = \mathbf{argmin}_{\alpha,\beta} \sum d(y, \alpha + \beta x)$$

can be loads of things

But most satisfy the following:

$$d(x, y) \geq 0, \quad d(x, x) = 0$$

$$(\hat{\alpha}, \hat{\beta}) = \operatorname{argmin}_{\alpha,\beta} \sum d(y, \alpha + \beta x)$$

$$d(x, y) = |x - y|$$ Absolute value loss

$$d(x, y) = (x - y)^2$$ Quadratic loss

$$d(x, y) = \begin{cases} \frac{1}{2}(y - x)^2, & |x - y| \le \delta, \\ \delta\left(|x - y| - \frac{1}{2}\delta\right), & \text{otherwise.} \end{cases}$$ Huber loss

But there are others! Infinitely many of them.

Almost ALL linear regression uses the quadratic distance function.

$$d(x, y) = (x - y)^2$$

$$(\hat{\alpha}, \hat{\beta}) = \operatorname{argmin}_{\alpha,\beta} \sum_{i=1}^{n} (y_i - (\alpha + \beta x_i))^2$$

But why?

1. Linear regression with quadratic weights, called ordinary linear regression (OLS) is easy to work with mathematically and computationally inexpensive.
No other distance come close.

---

2. The estimators estimate the conditional expectation $E(Y|x)$ provided that $E(Y|x) = \alpha + \beta x$. Other estimators typically estimate something slightly different.

---

3. The estimates are unique when $n > 1$.
The absolute value distance estimates are not.

---

4. The estimates are the "best linear unbiased estimator" and "efficient under normality". This means we can expect the estimators to be the best. Roughly.

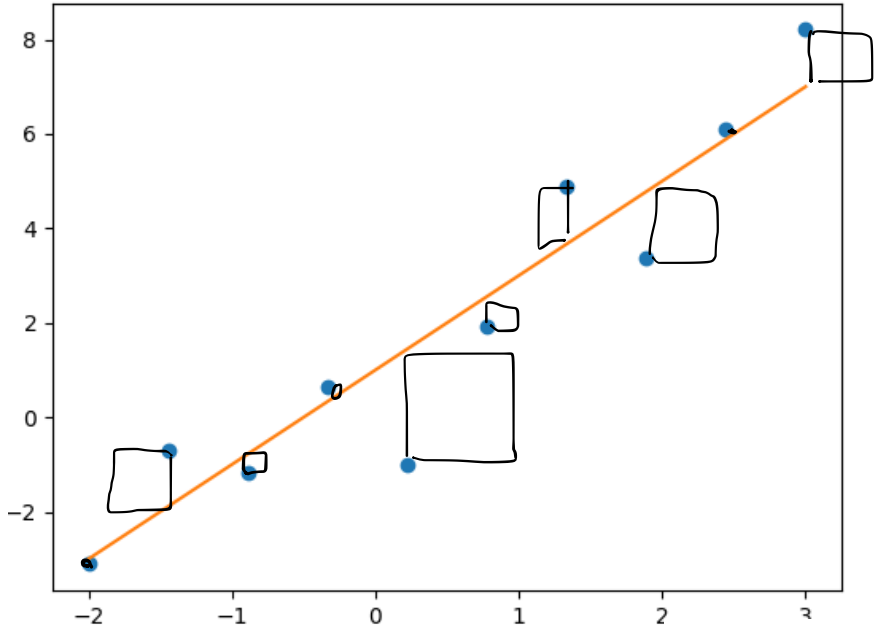
Easy to calculate? Yes. They have formulas!

$$\widehat{\alpha} = \bar{y} - (\widehat{\beta}\,\bar{x}),$$

$$\widehat{\beta} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

(Exercise.)

Sum up all the squares!



We use the true line right now.

```
((a + b*x - y)**2).sum()
```

$$\widehat{\alpha} = \bar{y} - (\widehat{\beta}\,\bar{x}),$$

$$\widehat{\beta} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

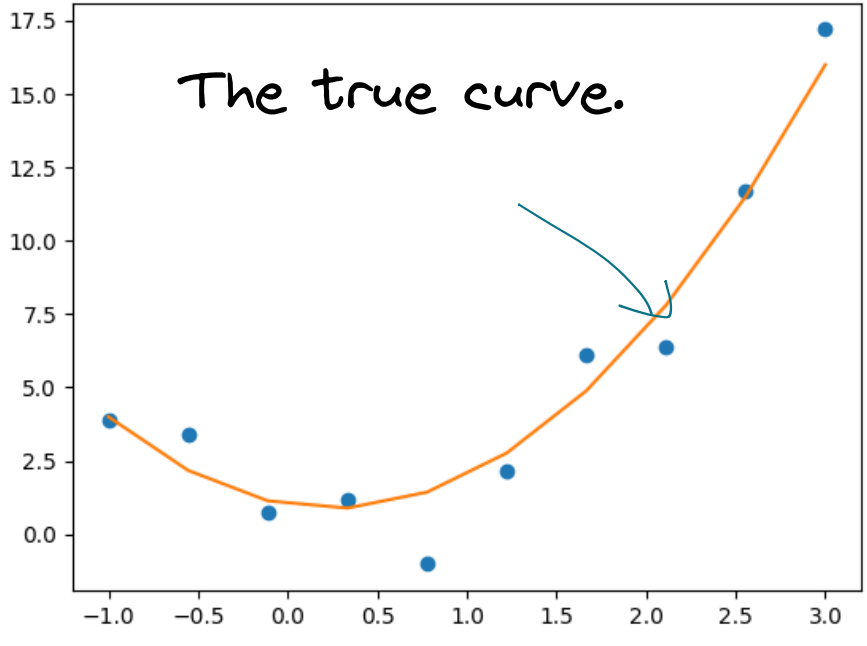
Remeber, the true parameters are 1 and 2!

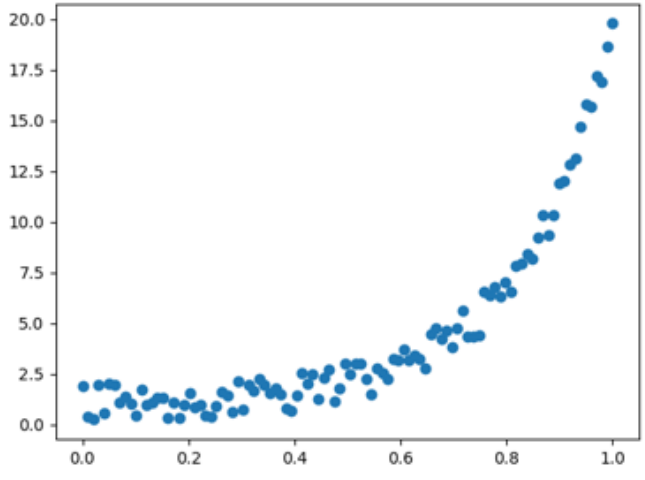
```
def ols(y, x):
    """ Returns the parameters a, b estimated by simple OLS."""
    y_bar = y.mean()
    x_bar = x.mean()
    beta = ((x - x_bar) * (y - y_bar)).sum() / ((x - x_bar)**2).sum()
    return (y_bar - beta * x_bar, beta)

ols(y, x)
```

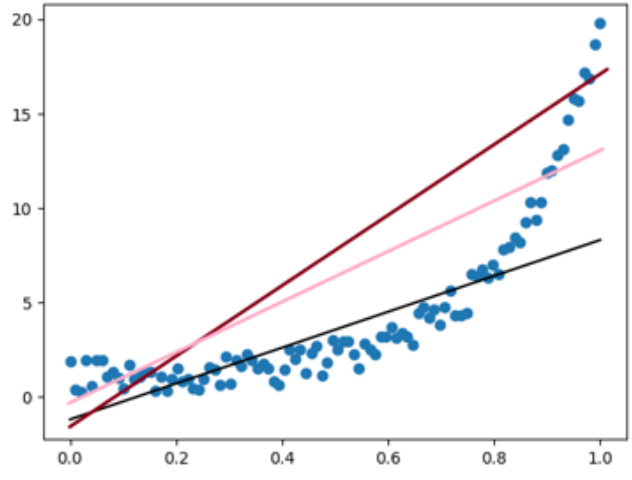
```
>>> ols(y, x)
(0.894462984791285, 2.0469643636640087)
```

```
rng = np.random.default_rng(313)
x = np.linspace(-1, 3, 10)
y = 1 + 2 * x ** 2 - x + rng.normal(0, 1, 10)
plt.plot(x, y, "o")
plt.plot(x, 1 + 2 * x ** 2 - x)
plt.show()
```

What happens when the linear model is false?





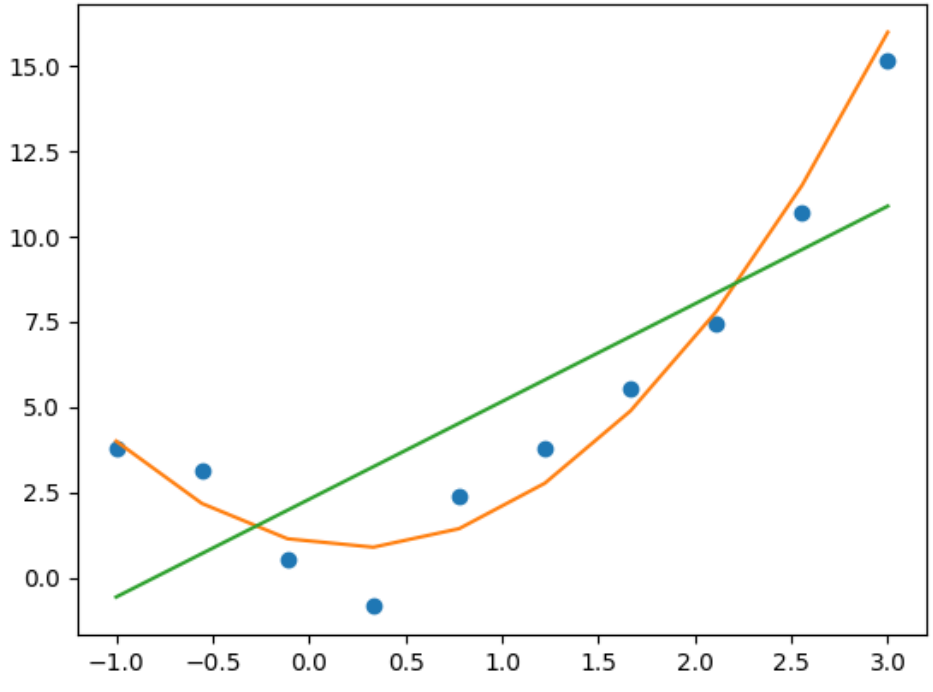


Do the lines on the left make sense?

Running linear regression still makes sense! And often works well!

```
>>> a, b = ols(y, x)
>>> a
2.118498971302539
>>> b
3.0587054545800103
```

```
plt.plot(x, y, "o")
plt.plot(x, 1 + 2 * x ** 2 - x)
plt.plot(x, a + b*x)
plt.show()
```



The estimates for the intercept and slope
are called
the least false values
when the linear model isn't true!

Which is NEVER the case in reality.

(But might be approximately the case.)

What does the esimtators $\hat{\alpha}$ and $\hat{\beta}$ estimate?

- If the linear model is true, i.e., the data is generated according to $\alpha + \beta x$ + some error with mean 0, then they are consistent for α and β!

- If not, which is virtually always the case, they are consistent for the intercept and slope of the best-fitting line. And best-fitting is defined as:

$$(\alpha, \beta) = \text{argmin}_{\alpha,\beta} E[Y - (\alpha + \beta X)]^2$$

$$(\alpha, \beta) = \text{argmin}_{\alpha,\beta} E[Y - (\alpha + \beta X)]^2$$

These parameters minimize the expected squared distance between X and Y.

Thus $\hat{\alpha}$ $\hat{\beta}$ are consistent estimators of the

α and β that generates the smallest EXPECTED squared distance between Y and α+βx!

It's not the same with other distances!

$$\text{argmin}_{\alpha,\beta} E[Y - (\alpha + \beta X)]^2$$
$$\neq \text{argmin}_{\alpha,\beta} E[|Y - (\alpha + \beta X)|]$$

In general, unless the linear model is true and the error is symmetric around 0.

What can you use this for?

Many things.

But first of all -- prediction!

How do we predict?

If the linear model

$$y = \alpha + \beta x + \epsilon, \; E(\epsilon \mid x) = 0$$

is true, then

$$\hat{y}(x) = E(Y \mid x) = \alpha + \beta x$$

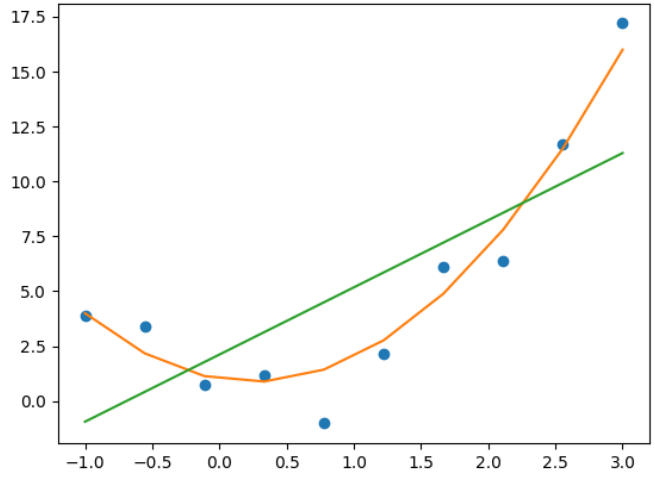
is the best possible prediction you can make, as measured by the quadratic loss.

But we don't know α and β!

So we use their estimates instead.

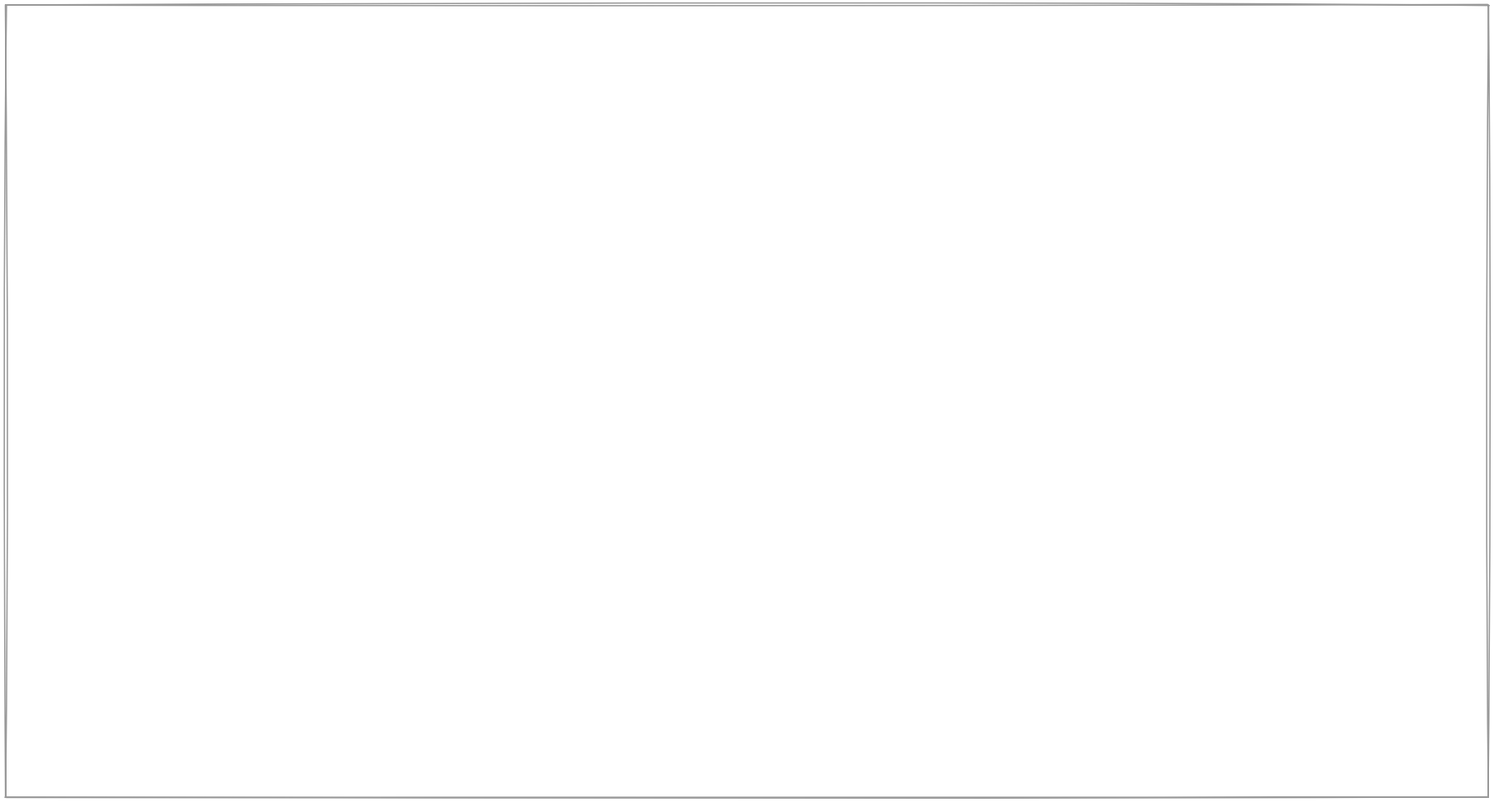
But we don't know α and β!

So we use their estimates instead.




```
>>> ols(y, x)
(2.118498971302539, 3.0587054545800103)
```

And our prediction, for any x_i, is simply the points on the green line!



statsmodels

We will mainly use **statsmodels** in this course. This package does a lot!

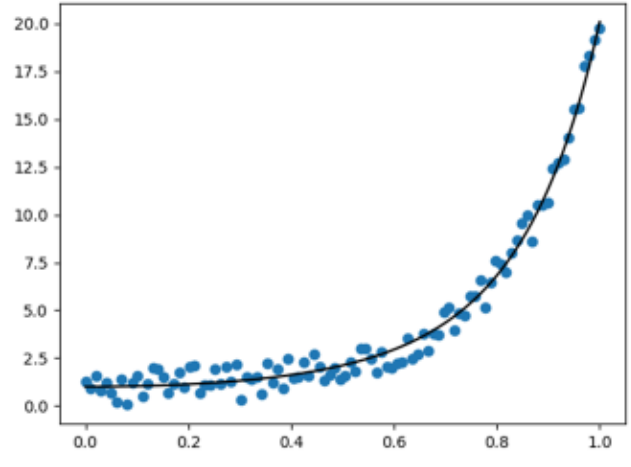
May also use **scikit-learn** and **scipy**.

statsmodels is inspired by the R ecosystem.

statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. An extensive list of result statistics are available for each estimator. The results are tested against existing statistical packages to ensure that they are correct. The package is released under the open source Modified BSD (3-clause) license. The online documentation is hosted at statsmodels.org.

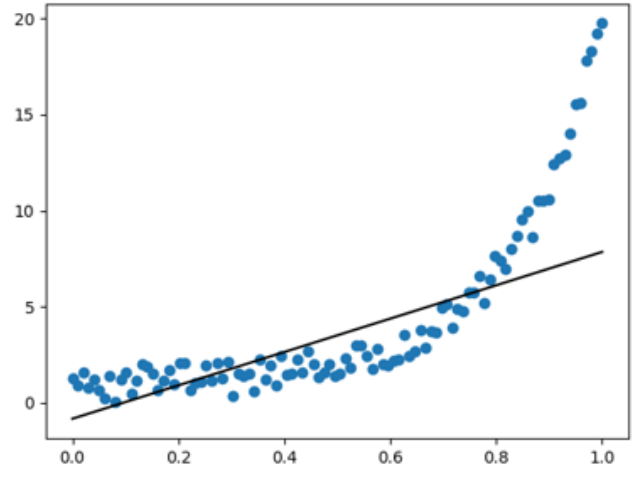
$$y = e^{3x^2} + u$$

```python
# Prototype 2
import numpy as np
import matplotlib.pyplot as plt
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)

plt.scatter(x, y)
plt.plot(x, np.exp(3 * x ** 2), color = "black")
plt.show()
```



LAD with statsmodels

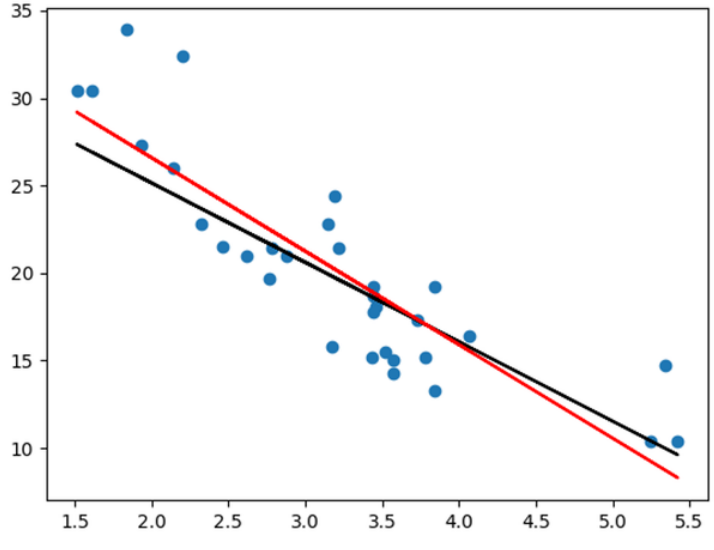
```
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import pandas as pd
import matplotlib.pyplot as plt

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

plt.scatter(x, y)
plt.plot(x, res_lad.params[0] + x * res_lad.params[1], color = "black")
plt.show()
```



```
res_lad = smf.quantreg("mpg ~ wt", data = mtcars).fit(q = 0.5)
res_ls = smf.ols("mpg ~ wt", data = mtcars).fit()

plt.scatter(mtcars["wt"], mtcars["mpg"])
plt.plot(mtcars["wt"], res_lad.params[0] + mtcars["wt"] * res_lad.params[1], color = "black")
plt.plot(mtcars["wt"], res_ls.params[0] + mtcars["wt"] * res_ls.params[1], color = "red")
plt.show()
```

Least squares too!

```
>>> res_lad.params
Intercept    34.232244
wt           -4.539476
dtype: float64
>>> res_ls.params
Intercept    37.285126
wt           -5.344472
dtype: float64
```



Not a huge difference between these fits!

How do you measure
how well you can predict Y from X?

Remember the idea of population vs sample value.

Even if the true relationsship between x and y isn't linear, there is a best-fitting line.

$$\min_{(\alpha,\beta)} E[(Y - \alpha - \beta X)^2]$$

If we don't know X, what is the best we can do?

$$\min_{\mu} E[(Y - \mu)^2]$$

The predictive power of a linear model, with the quadratic distance, is very frequently understood using

$$R^2 = 1 - \frac{\min_{(\alpha,\beta)} E[(Y - \alpha - \beta X)^2]}{\min_{\mu} E[(Y - \mu)^2]}$$

"How much better can I predict Y given X than without it?"

Define the covariance as

$$\mathrm{Cov}(X, Y) = E(XY) - E(X)E(Y)$$

And the correlation as:

$$\mathrm{Cor}(X, Y) = \frac{\mathrm{Cov}(X, Y)}{\mathrm{sd}X \cdot \mathrm{sd}Y}$$

Define the covariance as

$$\mathrm{Cov}(X, Y) = E(XY) - E(X)E(Y)$$

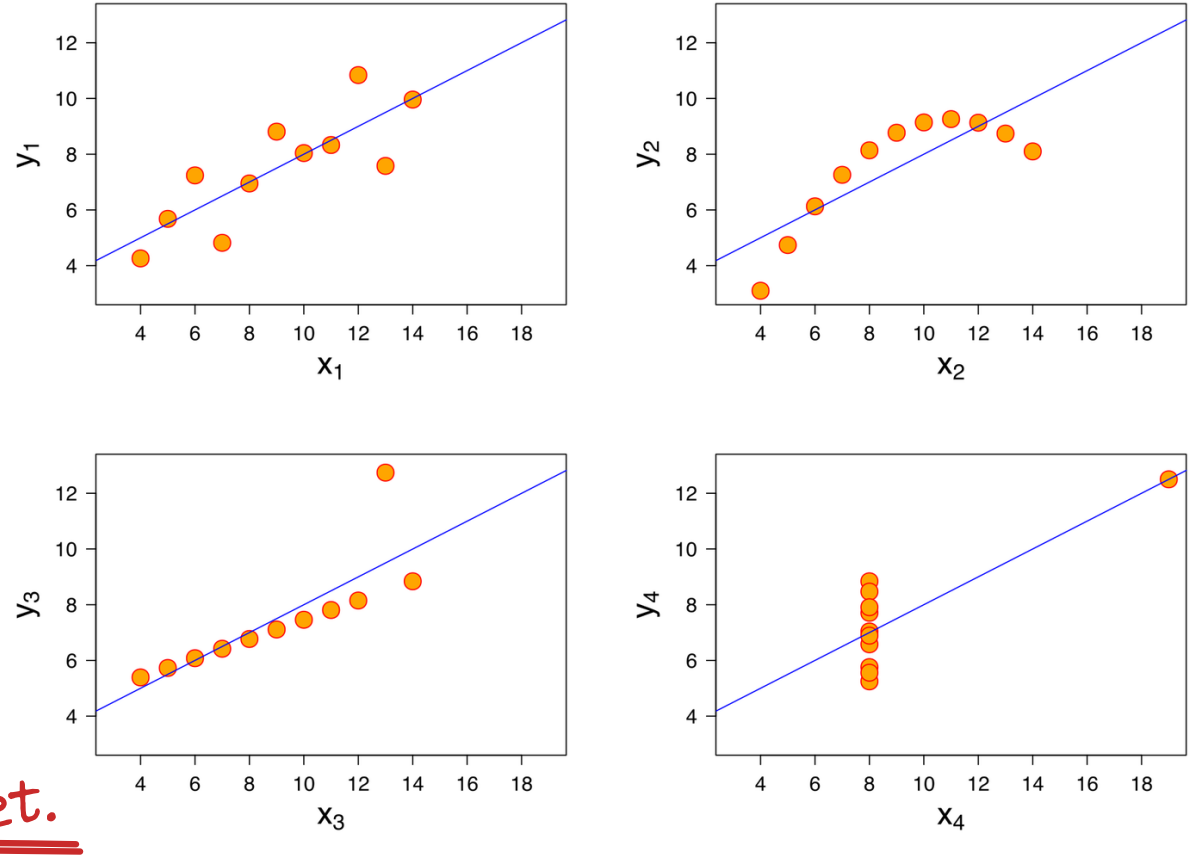
And the correlation as:

$$\mathrm{Cor}(X, Y) = \frac{\mathrm{Cov}(X, Y)}{\mathrm{sd}X \cdot \mathrm{sd}Y}$$

Important properties of correlation:

1. If X, Y are independent, then $\mathrm{Cor}(X, Y) = 0$.
2. If $X = a + bY$, then $\mathrm{Cor}(X, Y) = 1$.
3. If $X = a - bY$, then $\mathrm{Cor}(X, Y) = -1$.
4. If $\mathrm{Cor}(X, Y) = 0$, you can't conclude that X, Y are independent.
5. The correlation is *symmetric*, $\mathrm{Cor}(X, Y) = \mathrm{Cor}(Y, X)$.

All have correlation 0.67.



Anscombe's quartet.

Gain intuition from

http://guessthecorrelation.com/

The correlation is the most popular and frequently used measure of linear relationsship!

But why? What does it mean?

$$\mathrm{Cor}(X, Y) = \frac{\mathrm{Cov}(X, Y)}{\mathrm{sd}X \cdot \mathrm{sd}Y}$$

Remember the definition

$$R^2 = 1 - \frac{\min_{(\alpha,\beta)} E[(Y - \alpha - \beta X)^2]}{\min_{\mu} E[(Y - \mu)^2]}$$

Then we have:

$$\mathrm{Cor}(X, Y) = \mathrm{sign}(\beta) \cdot \sqrt{R^2}$$

(Exercise!)

Summing up

- Linear regression fits a linear line to a scatterplot
- This line is not unique, but depends on the distance function.
- The most common distance function is the squared distance.
  - But it does not have to be the best choice!
  - Doesn't matter much if the true relationsship is linear.
- The true relationsship does not have to be linear for linear regression to be used.