

# Estimating Associations

Michele Coscia

First Year Project #1

February 15<sup>th</sup>, 2022

# Lecture Plan

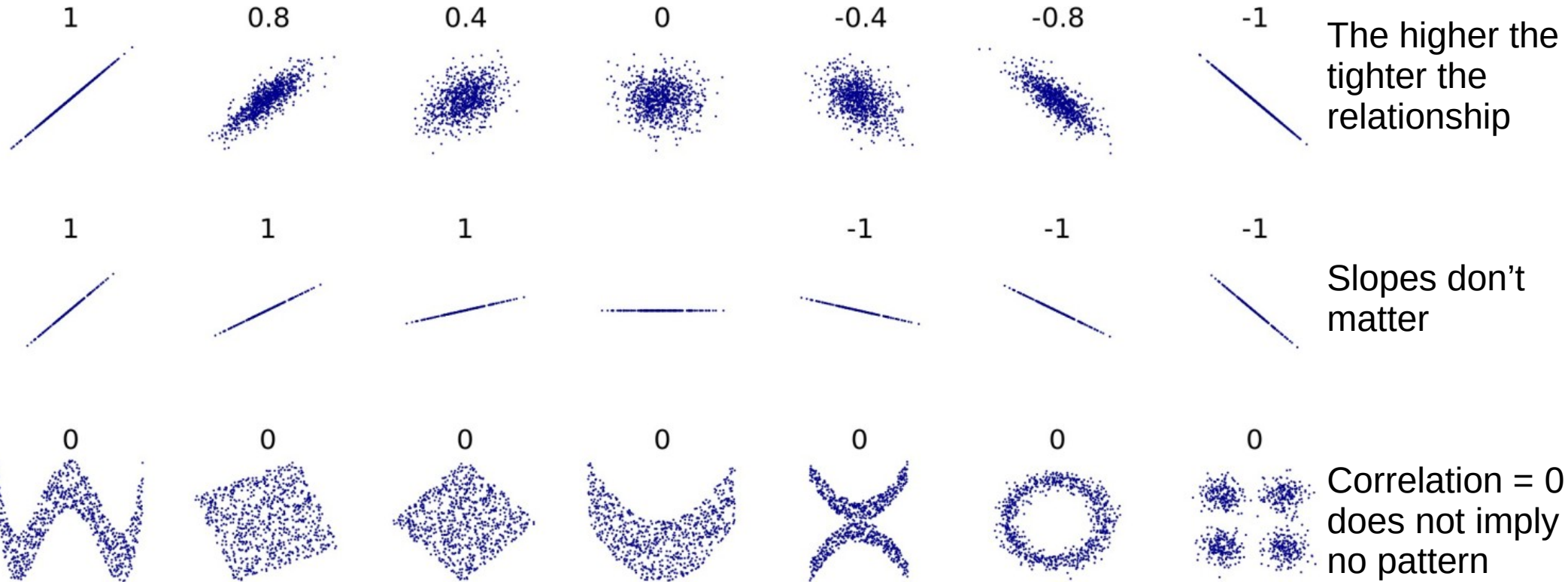
- 1) (February 8<sup>th</sup>) Intro
- 2) (February 10<sup>th</sup>) Geospatial Basics
- 3) (Today) Estimating Associations**
- 4) (February 17<sup>th</sup>) Multivariate Regression
- 5) (February 22<sup>nd</sup>) Interventions
- 6) (February 24<sup>th</sup>) Project Run Through
- 7) (March 1<sup>st</sup>) Q&A – Open Supervision
- 8) (March 3<sup>rd</sup>) Q&A – Open Supervision

# Outline

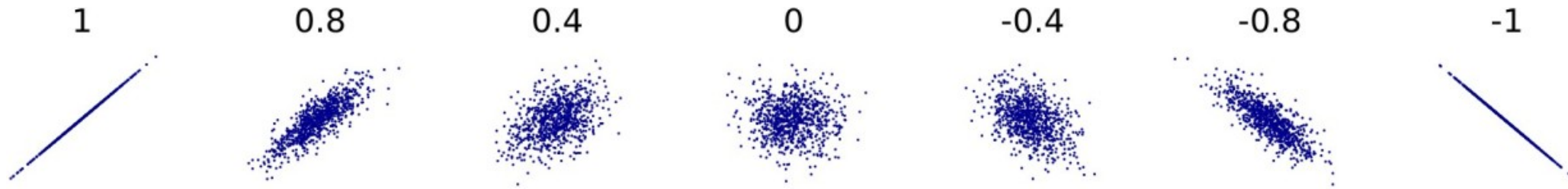
- Pearson & Spearman correlations
- Non-linear associations
- P-values
- Multiple Hypotheses Testing

Pearson

# Pearson Correlation



# The higher the tighter the relationship



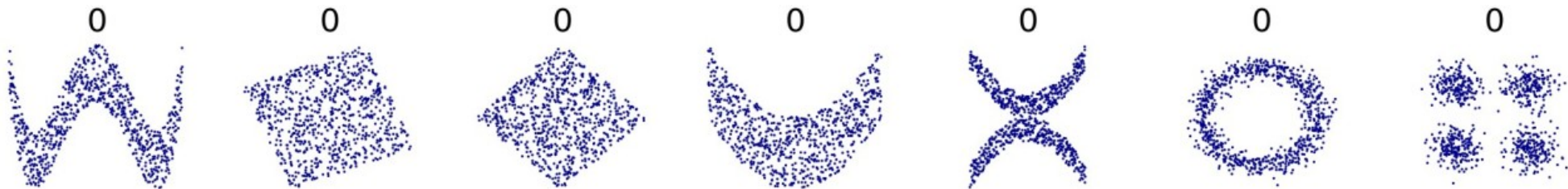
- 1 = Every time variable X increases, so does variable Y
- -1 = Every time variable X increases, variable Y decreases
- 0 = Knowing that variable X increases tells you nothing about variable Y
- Million \$\$\$ question: is 0.4 a high or low correlation?

# Slopes don't matter



- Correlation strength  $\neq$  Scale of the effect
- Significance  $\neq$  Scale of the effect
- E.g. X & Y are two raters and Y always rates half the score than X
- The correlation is the relation between X and Y if they were to be standardized

# The curse of linearity



- There are TONS of interesting relationships that are not linear
- Pearson (and linear regression) are blind to this
- In this case, Pearson is an underestimation of the relation
- These examples are extreme, but most likely?  
Diminishing returns and/or skewed data



# Let's give it a try

The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a directory named `/notebooks/` with several files, including `pr01_e03.ipynb` which is currently selected. The code editor shows the following code:

```
[1]: # As usual, importing the libraries we need
import json
import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy.stats import pearsonr, spearmanr

[4]: # Clean the corona data
corona_df = pd.read_csv("../data/raw/corona/dk_corona.csv", sep = "\t")

with open("../data/raw/metadata/dk_metadata.json", 'r') as f:
    country_metadata = json.load(f)

region_map = {country_metadata["country_metadata"][i]["covid_region_code"]: country_metadata["country_metadata"][i]["iso3166-2_code"] for i in range(len(country_metadata["country_metadata"]))}
corona_df["region"] = corona_df["region_code"].map(region_map)

corona_df
```

The output of the code is a preview of the `corona_df` DataFrame, showing 1760 rows and 4 columns: `date`, `region_code`, `hospitalized_addition`, and `region`. The preview shows the first 5 rows and the last 5 rows of the DataFrame.

	date	region_code	hospitalized_addition	region
0	2020-03-01	Capital Region of Denmark	1	DK-84
1	2020-03-02	Capital Region of Denmark	0	DK-84
2	2020-03-03	Capital Region of Denmark	1	DK-84
3	2020-03-04	Capital Region of Denmark	0	DK-84
4	2020-03-05	Capital Region of Denmark	1	DK-84
...	...	...	...	...
1755	2021-02-11	North Denmark Region	1	DK-81
1756	2021-02-12	North Denmark Region	1	DK-81
1757	2021-02-13	North Denmark Region	1	DK-81
1758	2021-02-14	North Denmark Region	1	DK-81
1759	2021-02-15	North Denmark Region	2	DK-81

1760 rows x 4 columns

```
[6]: # Clean the weather data
weather_df = pd.read_csv("../data/raw/weather/weather.csv")

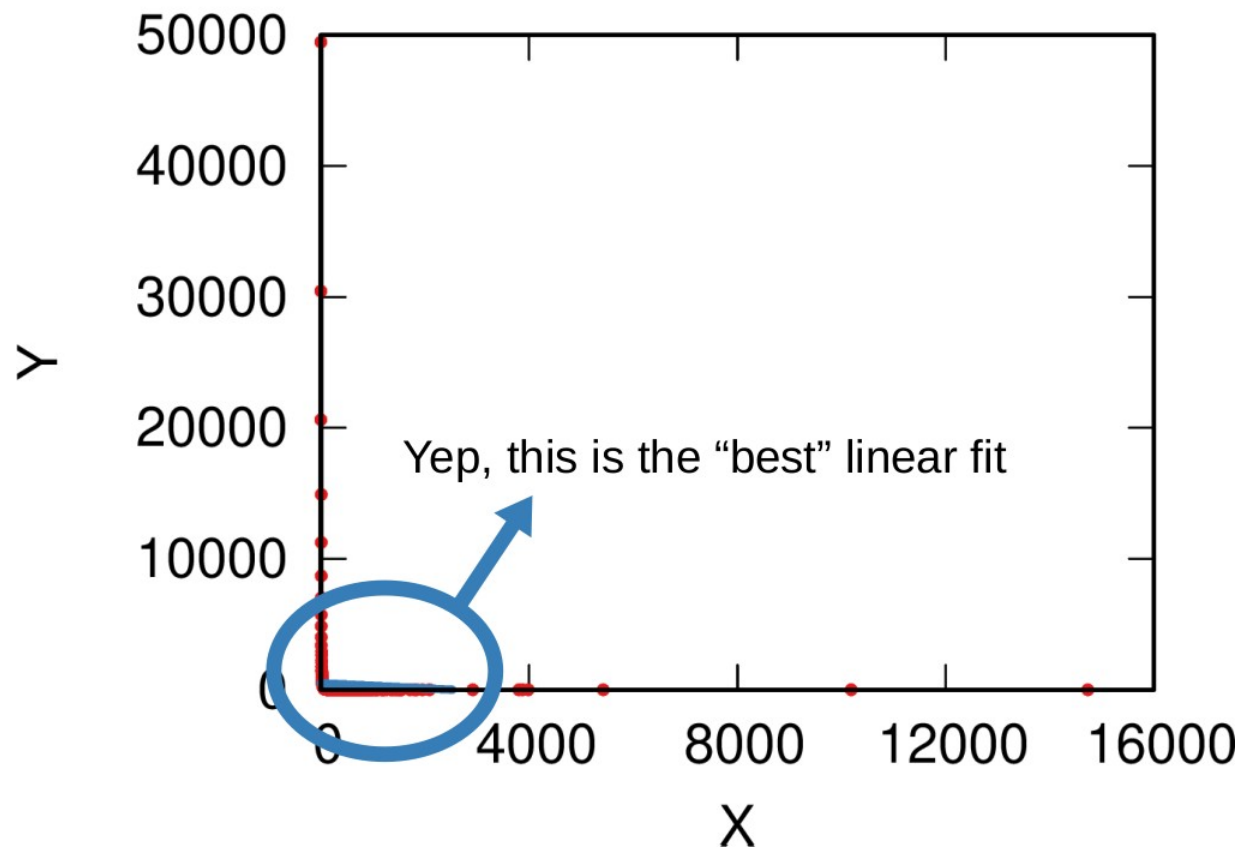
weather_df["TemperatureAboveGround"] = weather_df["TemperatureAboveGround"] - 273.15
weather_df = weather_df[weather_df["iso3166-2"].str.startswith("DK")]

weather_df
```

The status bar at the bottom indicates the notebook is in `Simple` mode, using `Python 3 (ipykernel)`, and the current cell is `pr01_e03.ipynb`.

# Non-linear Associations

# What if your data looks like this?

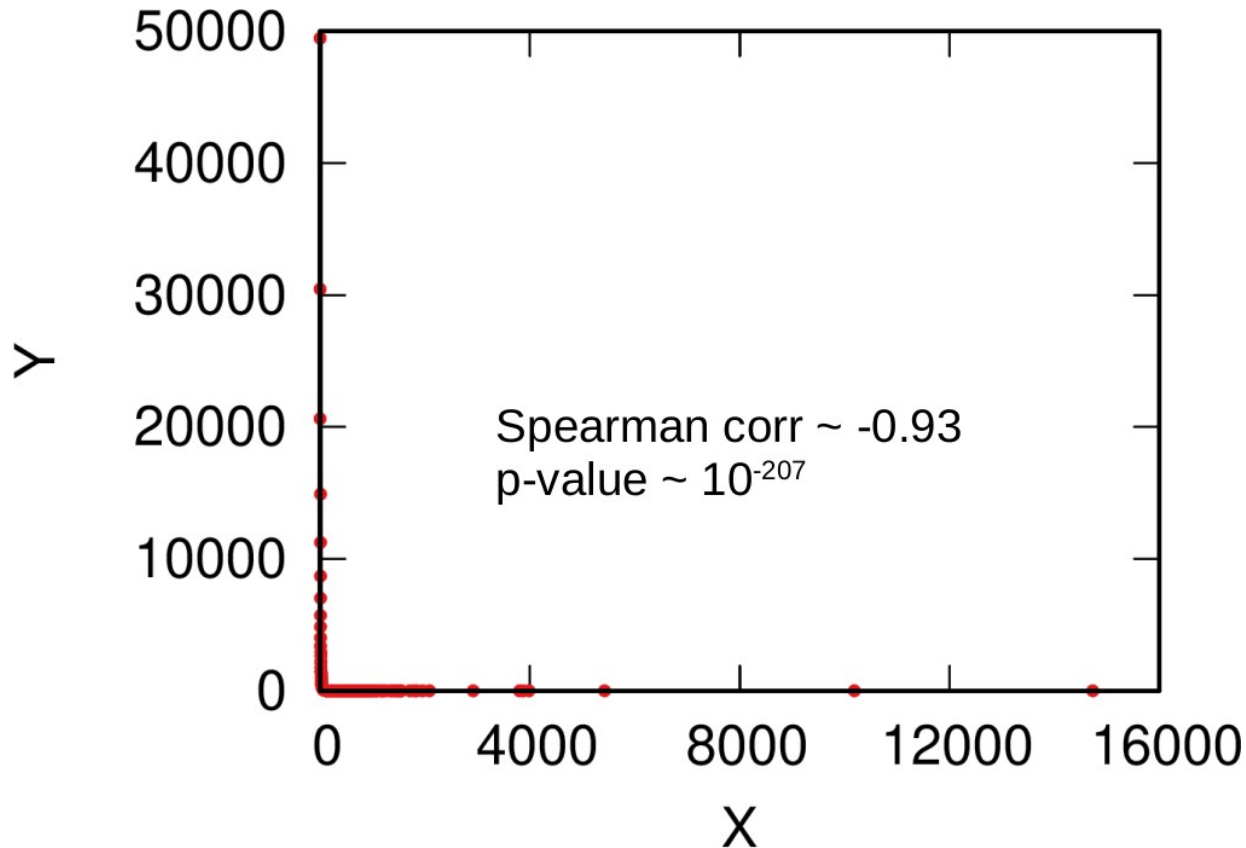


Pearson  
correlation  $\sim -0.05$

p-value  $\sim 0.2$

Ugh!

# Two Solutions

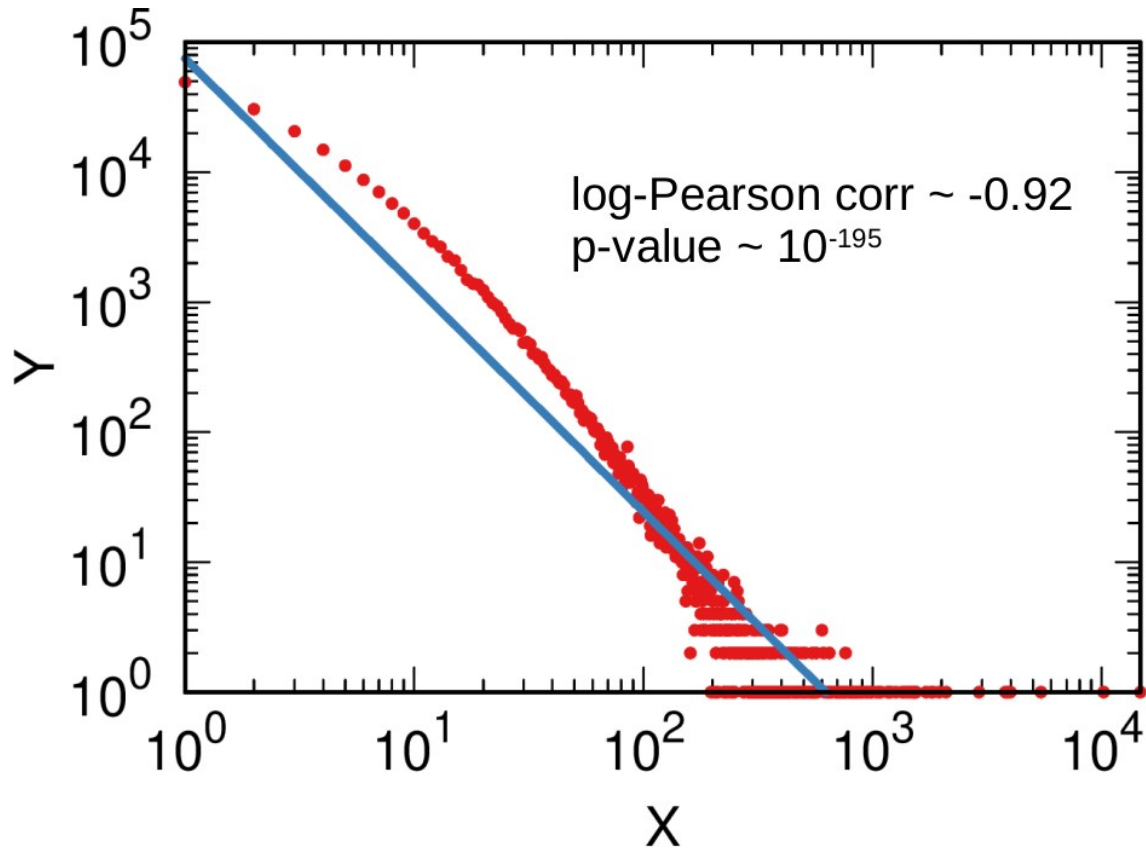


Solution #1: Spearman rank correlation

The values of X & Y don't matter

We only care that the n-th ranked value of X is also ranked n-th in Y

# Two Solutions



Solution #2: get rid of skewedness

Your best friend: the logarithm

We don't care about the precise values, only about the order of magnitude

# Let's try them out

The screenshot shows a Jupyter Notebook interface with a file browser on the left and a code editor on the right. The file browser shows a list of notebooks: pr01\_e01.i..., pr01\_e02.i..., pr01\_e03.i..., and pr02\_e99.i.... The code editor shows two code cells. The first cell, labeled [16]:, contains a loop that iterates over variables in a DataFrame and prints the Spearman rank correlation and p-value for each variable. The output shows results for RelativeHumiditySurface, SolarRadiation, Surfacepressure, TemperatureAboveGround, Totalprecipitation, UVIndex, and WindSpeed. The second cell, labeled [19]:, contains a loop that iterates over variables in a DataFrame and prints the log-transformed Pearson correlation and p-value for each variable. The output shows results for RelativeHumiditySurface, SolarRadiation, Surfacepressure, TemperatureAboveGround, Totalprecipitation, UVIndex, and WindSpeed.

```
[16]: # Now we do the same for the Spearman rank correlation
for var in Xs:
    corr, pvalue = spearmanr(df["hospitalized_addition"], df[var])
    print(f"{var}\n{corr:.3f}\t{pvalue}\t{pvalue < significance_threshold}\n")

RelativeHumiditySurface
0.295  9.41696780258754e-37  True

SolarRadiation
-0.507  1.0422712265565598e-115  True

Surfacepressure
0.023  0.3361348582493121  False

TemperatureAboveGround
-0.589  1.116912821679257e-164  True

Totalprecipitation
0.011  0.6487617370604414  False

UVIndex
-0.666  9.598316772732291e-226  True

WindSpeed
0.073  0.0021929360954572085  True

[19]: # And a thir dtime, for a log-transformed Pearson correlation, not the plus one,
# because you can't take the log of zero
for var in Xs:
    corr, pvalue = pearsonr(np.log(df["hospitalized_addition"] + 1), df[var])
    print(f"{var}\n{corr:.3f}\t{pvalue}\t{pvalue < significance_threshold}\n")

RelativeHumiditySurface
0.258  3.4511079816205213e-28  True

SolarRadiation
-0.459  1.2554270741551475e-92  True

Surfacepressure
0.002  0.9405622280596322  False

TemperatureAboveGround
-0.555  4.651616618011463e-143  True

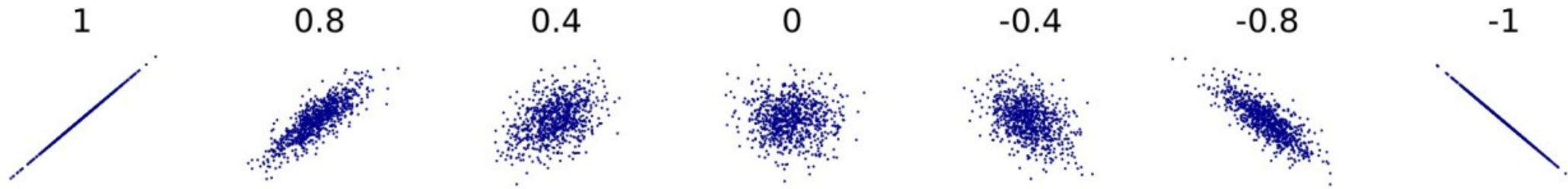
Totalprecipitation
-0.036  0.12654233015725455  False

UVIndex
-0.626  2.074355631569998e-192  True
```

Simple 0 2 Python 3 (ipykernel) | Idle Mode: Command Ln 1, Col 29 pr01\_e03.ipynb

p-values

# Significance: Basics



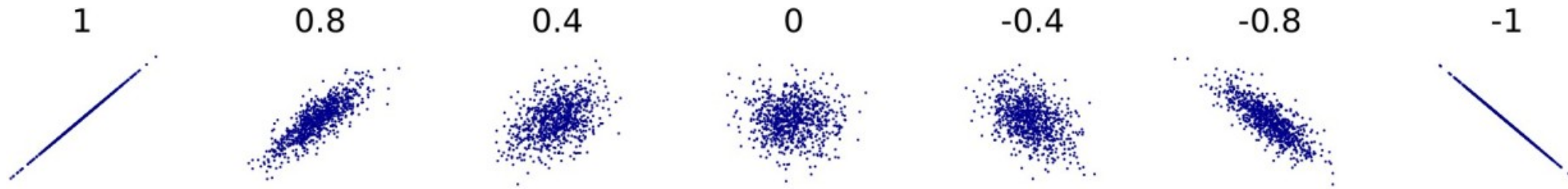
- p-value, jargon version: the probability of obtaining test results at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct.
- Translation: how likely it is to see this correlation value if there is no relation between X and Y



# P-values: Rules of Thumb

- 0.05: loose, used in social science (weak, noisy effects)
- 0.01: social science with many observations
- 0.001: better benchmark for big data science
- 5 sigma ( $3 \times 10^{-7}$ ): physics

# Significance: Basics



- The higher the coefficient, the less likely the null hypothesis can generate the data
- More observations can strengthen your confidence even for low correlations
- (We'll come back to this later)

# What are we doing?

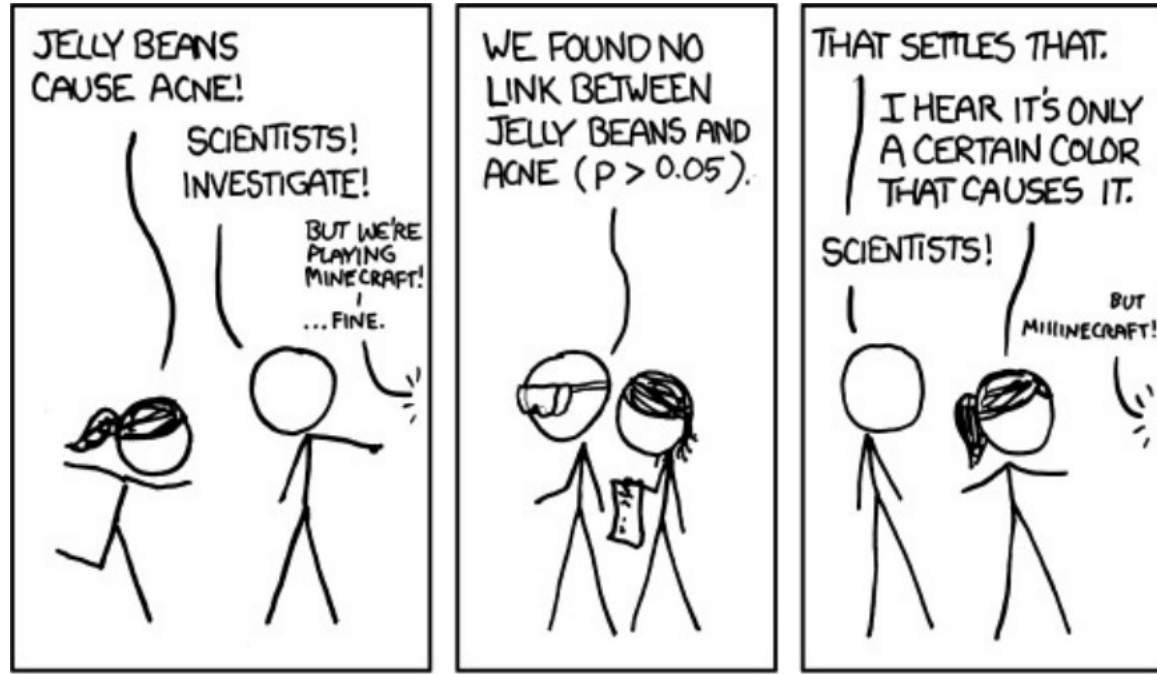
- $h_1$ : there is a linear relationship between  $X$  and  $Y$
- $h_0$ : there is not a linear relationship between  $X$  and  $Y$
- $h_0$  is the null hypothesis

# Skewed Data

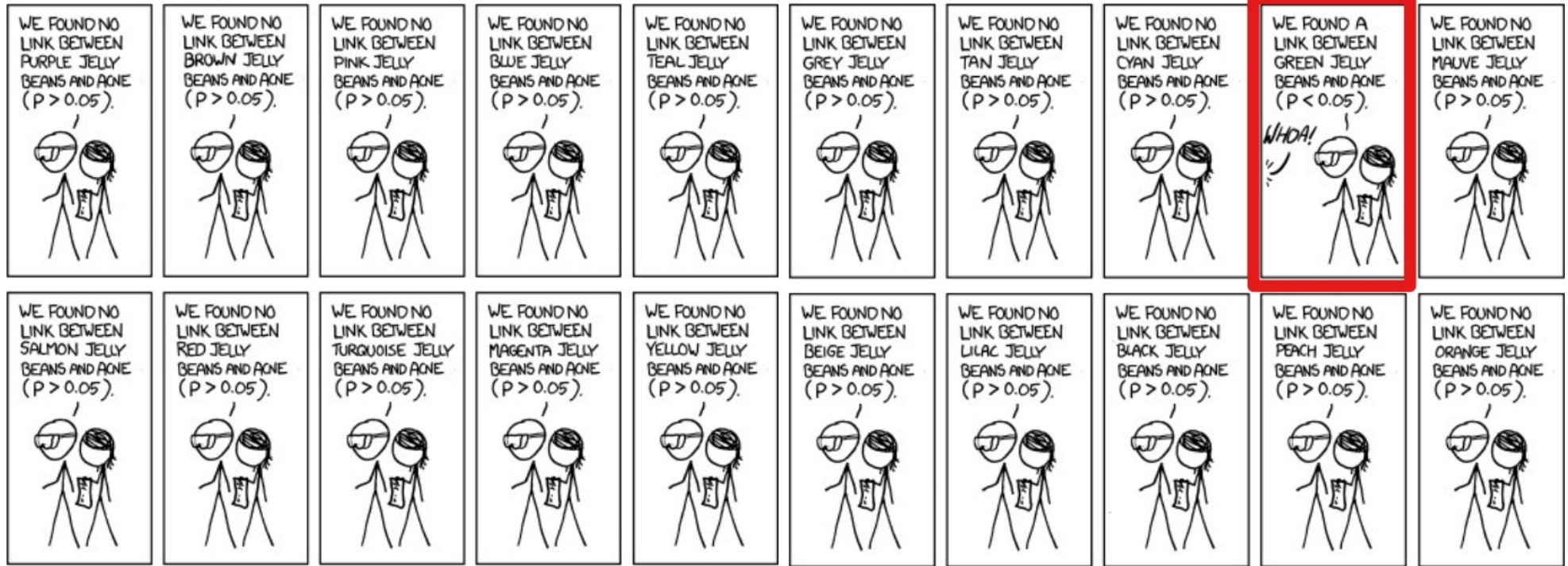
- Inflated p-values because  **$h_0$**  is correct!
- But we really care about a **different**  $h_1$ !
- i.e. there is a **non-linear** relationship between X and Y
- Spearman & log-log make different  $h_1$ s

# Multiple Hypotheses Testing

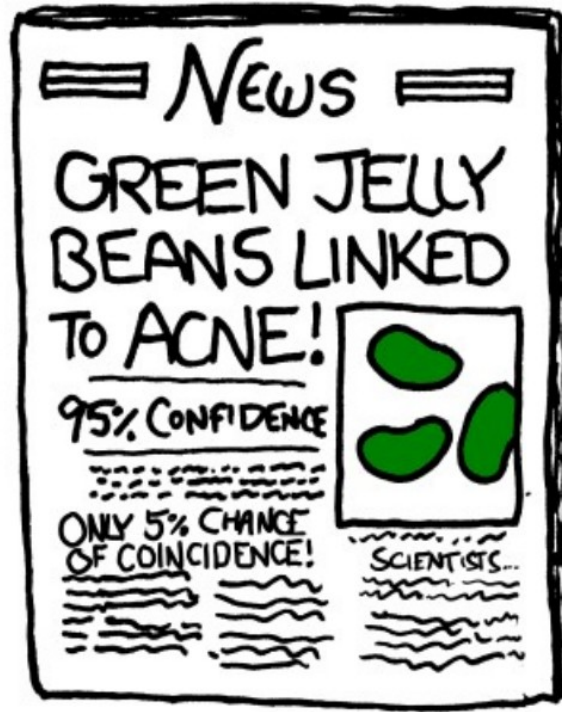
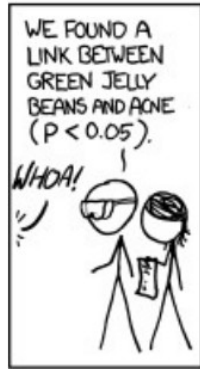
# A rule of thumb for not fooling yourself with p-values



# A rule of thumb for not fooling yourself with p-values



# A rule of thumb for not fooling yourself with p-values



Right?

**WRONG!**

If you run  $X$  tests, you expect one of them to have  $1/X$  p-value by pure chance!

(That's literally what "p-value" means!)



# Bonferroni Correction

- If 20 tests will generate a p-value  $\sim 0.05$  by chance...
- ...and 0.05 is the p-value threshold I chose to determine significance...
- Then my real threshold should be lower than 0.05!
  - Your  $h_0$  is that **none** of  $h_1$  to  $h_{20}$  is true!
- Specifically it should be  $0.05/20$

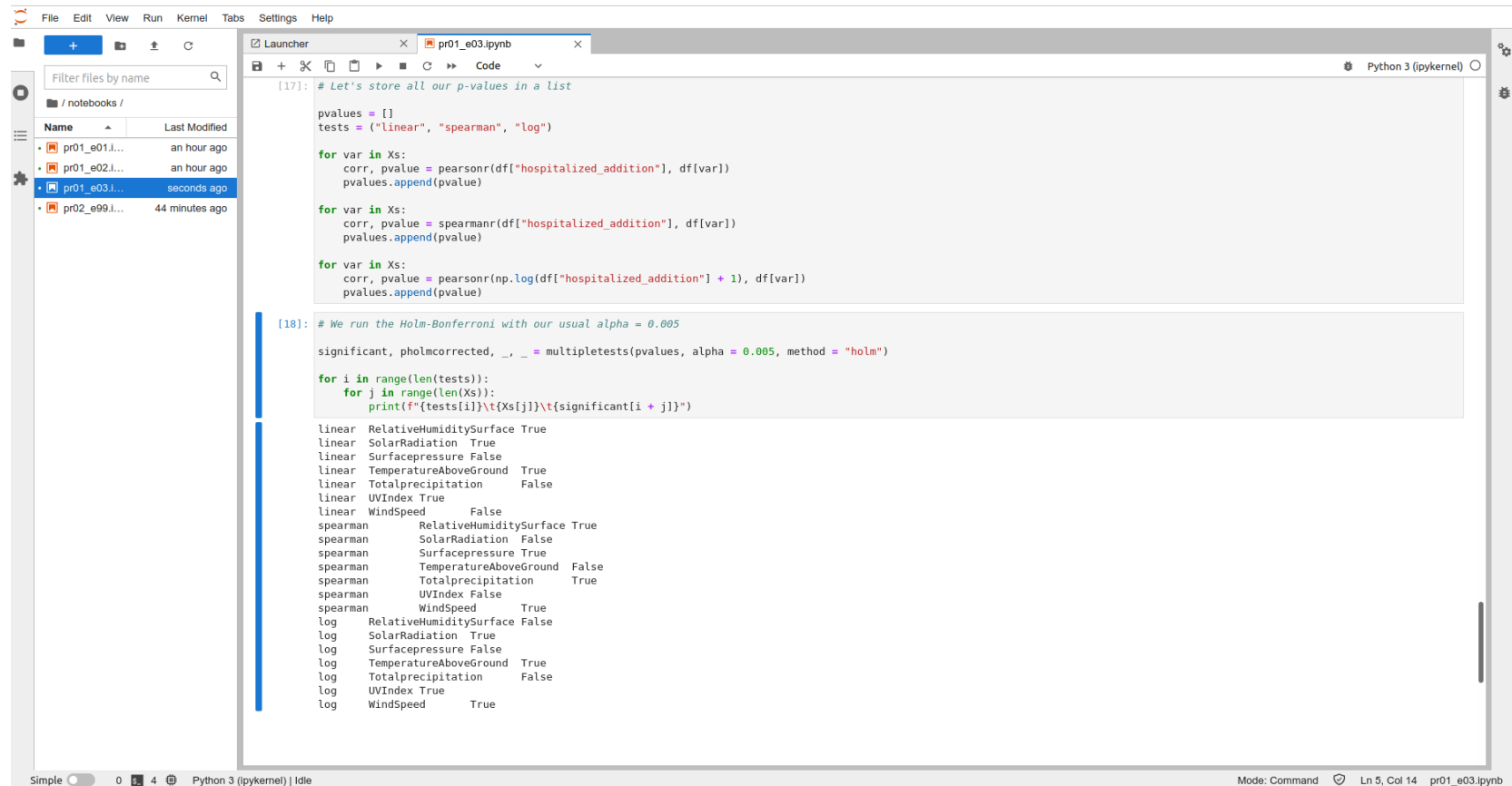
# But!

- What I said applied if your questions are **independent**
- In your case they are **not**!
- So Bonferroni is too strict
- Alternatives?

# Holm-Bonferroni

- Sort your p-values in ascending order
- First p-value  $\rightarrow p_1 < \alpha / N$
- Second p-value  $\rightarrow p_2 < \alpha / (N - 1)$ 
  - Because we already know  $p_1$  passes!
- And so on...

# Let's get our hands dirty...



File Edit View Run Kernel Tabs Settings Help

Launcher pr01\_e03.ipynb Python 3 (ipykernel)

```
[17]: # Let's store all our p-values in a list

pvalues = []
tests = ("linear", "spearman", "log")

for var in Xs:
    corr, pvalue = pearsonr(df["hospitalized_addition"], df[var])
    pvalues.append(pvalue)

for var in Xs:
    corr, pvalue = spearmanr(df["hospitalized_addition"], df[var])
    pvalues.append(pvalue)

for var in Xs:
    corr, pvalue = pearsonr(np.log(df["hospitalized_addition"] + 1), df[var])
    pvalues.append(pvalue)

[18]: # We run the Holm-Bonferroni with our usual alpha = 0.005

significant, holmcorrected, _, _ = multipletests(pvalues, alpha = 0.005, method = "holm")

for i in range(len(tests)):
    for j in range(len(Xs)):
        print(f"{tests[i]}\t{Xs[j]}\t{significant[i + j]}")

linear RelativeHumiditySurface True
linear SolarRadiation True
linear Surfacepressure False
linear TemperatureAboveGround True
linear Totalprecipitation False
linear UVIndex True
linear WindSpeed False
spearman RelativeHumiditySurface True
spearman SolarRadiation False
spearman Surfacepressure True
spearman TemperatureAboveGround False
spearman Totalprecipitation True
spearman UVIndex False
spearman WindSpeed True
log RelativeHumiditySurface False
log SolarRadiation True
log Surfacepressure False
log TemperatureAboveGround True
log Totalprecipitation False
log UVIndex True
log WindSpeed True
```

Simple 0 4 Python 3 (ipykernel) | Idle Mode: Command Ln 5, Col 14 pr01\_e03.ipynb

Q&A