

---

# **Quantitative Big Imaging - Statistics**

**Anders Kaestner**

**Apr 18, 2021**



# CONTENTS

<b>1 Statistics and Reproducibility</b>	<b>3</b>
1.1 Literature / Useful References . . . . .	3
1.2 Previously on QBI . . . . .	4
1.3 Quantitative “Big” Imaging . . . . .	5
1.4 Objectives . . . . .	5
1.5 What do we start with? . . . . .	6
1.6 Today’s outline . . . . .	6
<b>2 Correlation and Causation</b>	<b>7</b>
2.1 Observational or Controlled . . . . .	7
<b>3 Simple Model</b>	<b>9</b>
3.1 Magic / Biased Coin . . . . .	9
3.2 Experiment: Magic / Weighted Coin . . . . .	10
3.3 Simple Model to Reality . . . . .	10
<b>4 A more complicated model</b>	<b>11</b>
4.1 Let’s load the Iris data . . . . .	11
4.2 A first inspection of the data . . . . .	12
4.3 Comparing Groups: Intraclass Correlation Coefficient . . . . .	13
4.4 Making quantitative statements . . . . .	14
<b>5 Comparing Groups</b>	<b>17</b>
5.1 Loaded Coin example . . . . .	17
5.2 Comparing Groups: Student’s T Distribution . . . . .	17
<b>6 Multiple Testing Bias</b>	<b>19</b>
6.1 Probability with increasing number of tosses . . . . .	19
6.2 Probability with many experiments . . . . .	19
6.3 Multiple Testing Bias: Experiments . . . . .	20
<b>7 Multiple Testing Bias: Correction</b>	<b>23</b>
<b>8 Predicting and Validating - main categories</b>	<b>27</b>
8.1 Overview . . . . .	28
8.2 Validation . . . . .	29
8.3 Concrete Example: Classifying Flowers . . . . .	29
<b>9 Qualitative vs Quantitative</b>	<b>31</b>
9.1 Qualitative Assessment . . . . .	31
9.2 Quantitative Metrics . . . . .	31

9.3	Parameters . . . . .	32
9.4	Sensitivity . . . . .	34
9.5	Sensitivity: Real Measurements . . . . .	35
9.6	Sensitivity: compare more than one variable . . . . .	35
9.7	Reproducibility . . . . .	36
9.8	Reproducible Analysis . . . . .	36
<b>10</b>	<b>Data frames - managing feature tables</b>	<b>39</b>
10.1	Our workflow . . . . .	39
10.2	How do we store the features? . . . . .	40
10.3	Introducing data frames . . . . .	40
10.4	Create a data frame . . . . .	41
10.5	Working with columns . . . . .	42
10.6	Statistics with a data frame . . . . .	43
10.7	Statistics of filtered data . . . . .	44
10.8	Set operations with data frames . . . . .	46
10.9	Create new data frame from selected columns . . . . .	48
10.10	When are pandas data frames useful? . . . . .	48
<b>11</b>	<b>Presenting the results - bringing out the message</b>	<b>49</b>
11.1	Visualization . . . . .	49
11.2	Bad Graphs . . . . .	50
11.3	How to improve - Key Ideas . . . . .	52
11.4	Grammar of Graphics . . . . .	53
11.5	What is my message . . . . .	55
11.6	Common visualization packages for python . . . . .	60
<b>12</b>	<b>Summary</b>	<b>61</b>
12.1	Statistics . . . . .	61
12.2	Data frames . . . . .	61
12.3	Visualization . . . . .	61
<b>13</b>	<b>Old slides to transfer from R to python</b>	<b>63</b>

This is the lecture notes for the 8th lecture of the Quantitative big imaging class given during the spring semester 2021 at ETH Zurich, Switzerland.



## STATISTICS AND REPRODUCIBILITY

```
%load_ext autoreload
%autoreload 2
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import pandas as pd
%matplotlib inline

plt.rcParams["figure.figsize"] = (8, 8)
plt.rcParams["figure.dpi"] = 150
plt.rcParams["font.size"] = 14
plt.rcParams['font.family'] = ['sans-serif']
plt.rcParams['font.sans-serif'] = ['DejaVu Sans']
plt.style.use('ggplot')
sns.set_style("whitegrid", {'axes.grid': False})
```

## 1.1 Literature / Useful References

### 1.1.1 Books

- Jean Claude, *Morphometry with R*
- Online through ETHZ
- Chapter 3
- John C. Russ, *The Image Processing Handbook*,(Boca Raton, CRC Press)
- Available online within domain ethz.ch (or proxy.ethz.ch / public VPN)
- Hypothesis Testing Chapter
- Grammar of Graphics: Leland and Wilkinson - <http://www.springer.com/gp/book/9780387245447>

### **1.1.2 Papers / Sites**

- Matlab Unit Testing Documentation
- Databases Introduction
- Visualizing Genomic Data (General Visualization Techniques)
- NIMRod Parameter Studies
- M.E. Wolak, D.J. Fairbairn, Y.R. Paulsen (2012) Guidelines for Estimating Repeatability. Methods in Ecology and Evolution 3(1):129-137.
- David J.C. MacKay, [Bayesian Interpolation](#) (1991)

### **1.1.3 Videos / Podcasts**

- Google/Stanford Statistics Intro
- Last Week Tonight: Scientific Studies
- Credibility Crisis

### **1.1.4 Further material**

#### **Slides**

- How to solve NLP problems
- Data Visualization
- P-Values with Puppies

#### **Model Evaluation**

- Julia Evans - Recalling with Precision
- Stripe's Next Top Model

#### **Iris Dataset**

- The Iris dataset was used in Fisher's classic 1936 paper, [The Use of Multiple Measurements in Taxonomic Problems](#)

## **1.2 Previously on QBI ...**

- Image Enhancement
- Highlighting the contrast of interest in images
- Minimizing Noise
- Understanding image histograms
- Automatic Methods

- Component Labeling
- Single Shape Analysis
- Complicated Shapes

## 1.3 Quantitative “Big” Imaging

The course has covered imaging enough and there have been a few quantitative metrics, but “big” has not really entered.

What does **big** mean?

- Not just / even large
- it means being ready for *big data*
- volume, velocity, variety (3 V's)
- scalable, fast, easy to customize

So what is “big” imaging

### 1.3.1 Doing analyses in a disciplined manner

- fixed steps
- easy to regenerate results
- no *magic*
- documentation

#### Having everything automated

- 100 samples is as easy as 1 sample

#### Being able to adapt and reuse analyses

- one really well working script and modify parameters
- different types of cells
- different regions

## 1.4 Objectives

1. Scientific Studies all try to get to a single number
  - Make sure this number is describing the structure well (what we have covered before)
  - Making sure the number is meaningful (**today!**)
1. How do we compare the number from different samples and groups?
  - Within a sample or same type of samples

- Between samples
1. How do we compare different processing steps like filter choice, minimum volume, resolution, etc?
  2. How do we evaluate our parameter selection?
  3. How can we ensure our techniques do what they are supposed to do?
  4. How can we visualize so much data? Are there rules?

### 1.5 What do we start with?

Going back to our original cell image

1. We have been able to **get rid of the noise** in the image and **find all the cells** (lecture 2-4)
2. We have **analyzed the shape** of the cells using the shape tensor (lecture 5)
3. We even **separated cells joined together** using Watershed (lecture 6)
4. We have created even more **metrics characterizing the distribution** (lecture 7)

We have at least a few samples (or different regions), large number of metrics and an almost as large number of parameters to *tune*

**How do we do something meaningful with it?**

### 1.6 Today's outline

- Motivation (Why and How?)
- Scientific Goals
- Reproducibility
- Predicting and Validating
- Statistical metrics and results
- Parameterization
- Parameter sweep
- Sensitivity analysis
- Data frames
- Visualization

---

**CHAPTER  
TWO**

---

## **CORRELATION AND CAUSATION**

One of the most repeated criticisms of scientific work is that correlation and causation are confused.

### **1. Correlation**

- means a statistical relationship
- very easy to show (single calculation)

### **1. Causation**

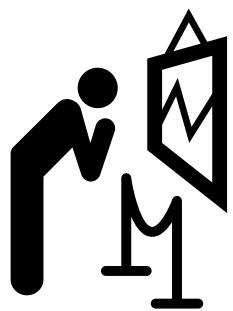
- implies there is a mechanism between A and B
- very difficult to show (impossible to prove)

## **2.1 Observational or Controlled**

There are two broad classes of data and scientific studies.

- Observational
- Controlled

Each type appears, but it is more likely to perform observational studies in the early stages of a project to gain an overview of the working field. From this study it you will make observations for more detailed studies which then are controlled.



Created by Luis Prado  
from the Noun Project

Fig. 2.1: In observational experiments you stand back and only observe what is happening.



Created by Alexis Lilly  
from the Noun Project

Fig. 2.2: In controlled experiments you prepare the samples for a specific purpose and control the environment.

### 2.1.1 Observational

In observational experiments you are not interfering with the observed phenomenon. You only make a selection of specimens or individuals that will be measured as they appear.

- Exploring large datasets looking for trends
- Population is random
- Not always hypothesis driven
- Rarely leads to causation

#### Examples of observational experiments

- We examined 100 people
  - the ones with blue eyes were on average 10cm taller
- In 100 cake samples
  - we found a 0.9 correlation between cooking time and bubble size

### 2.1.2 Controlled

- Most scientific studies fall into this category
- Specifics of the groups are controlled
- Can lead to causation

#### Examples of controlled experiments

- We examined 50 mice with gene XYZ off and 50 gene XYZ on and as the foot size increased by 10%
- We increased the temperature and the number of pores in the metal increased by 10%

---

CHAPTER  
**THREE**

---

## **SIMPLE MODEL**

It often convenient to start with a simplified models for your experiments where most uncertainties are reduced. In particular here in this lecture we chose a simple model for the demonstration

Since most of the experiments in science are usually

- specific,
- noisy,
- and often very complicated

and are not usually good teaching examples.

We go for a simple model...

### **3.1 Magic / Biased Coin**

Our model is the task to flip a coin and determine if it is a fair or loaded. The coin has two outcomes

- head
- or tail

You buy a *magic* coin at a shop.

#### **3.1.1 How many times do you need to flip it to prove it is not fair?**

Next step is to describe an experiment strategy. Some examples are:

- If I flip it 10 times and another person flips it 10 times, is that the same as 20 flips?
- If I flip it 10 times and then multiply the results by 10 is that the same as 100 flips?

As you already may have guessed, these are not the best assumptions, in particular not the second one.

A different question is about collections of random variables:

- If I buy 10 coins and want to know which ones are fair what do I do?



Fig. 3.1: Tossing a coin is a simple random process.

## 3.2 Experiment: Magic / Weighted Coin

1. Each coin represents a stochastic variable  $\mathcal{X}$  and each flip represents an observation  $\mathcal{X}_i$ .
2. The act of performing a coin flip  $\mathcal{F}$  is an observation  $\mathcal{X}_i = \mathcal{F}(\mathcal{X})$

We normally assume

1. A *fair* coin has an expected value of  $E(\mathcal{X}) = \frac{1}{2}$ :
  - 50% Heads,
  - 50% Tails
2. An *unbiased* flip(er) means *each flip is independent of the others*

$$P(\mathcal{F}_1(\mathcal{X}) \cdot \mathcal{F}_2(\mathcal{X})) = P(\mathcal{F}_1(\mathcal{X})) \cdot P(\mathcal{F}_2(\mathcal{X}))$$

- the expected value of the flip is the same as that of the coin

$$E\left(\prod_{i=0}^{\infty} \mathcal{F}_i(\mathcal{X})\right) = E(\mathcal{X})$$

## 3.3 Simple Model to Reality

### 3.3.1 Coin Flip

1. Each flip gives us a small piece of information about
  - the coin
  - *and* the flipper
1. More flips provides more information
  - **Random / Stochastic variations** in coin and flipper **cancel out**
  - **Systematic variations accumulate**

### 3.3.2 Simple Model to Reality - real experiment

1. Each measurement tells us about:
  - our sample,
  - our instrument,
  - and our analysis
1. More measurements provide more information:
  - Random / Stochastic variations in sample, instrument, and analysis cancel out
  - *Normally* the analysis has very little to no stochastic variation
  - Systematic variations accumulate

This is also the reason why we want many repeated observations in an experiment. Repetitions are however expensive, they require time to perform the experiment and more material for the specimens. Therefore, a pragmatic choice must be made that balances the cost versus a reasonable amount of observations.

## A MORE COMPLICATED MODEL

Coin flips are very simple and probably difficult to match to another experiment.

A very popular dataset for learning about such values beyond ‘coin-flips’ is called the Iris dataset which covers

- a number of measurements
- from different plants
- and the corresponding species.

### 4.1 Let's load the Iris data

The data set has information about dimensions of the flower anatomy for each of the three species. We load the data which is provided as a python dictionary and prepare a data frame for the table. You will get a more detailed introduction to pandas data frames later in this lecture.

```
data          = load_iris()
iris_df      = pd.DataFrame(data['data'], columns=data['feature_names'])
iris_df['target'] = data['target_names'][data['target']]
iris_df.sample(5)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	\
41	4.5	2.3	1.3	0.3	
125	7.2	3.2	6.0	1.8	
53	5.5	2.3	4.0	1.3	
40	5.0	3.5	1.3	0.3	
3	4.6	3.1	1.5	0.2	

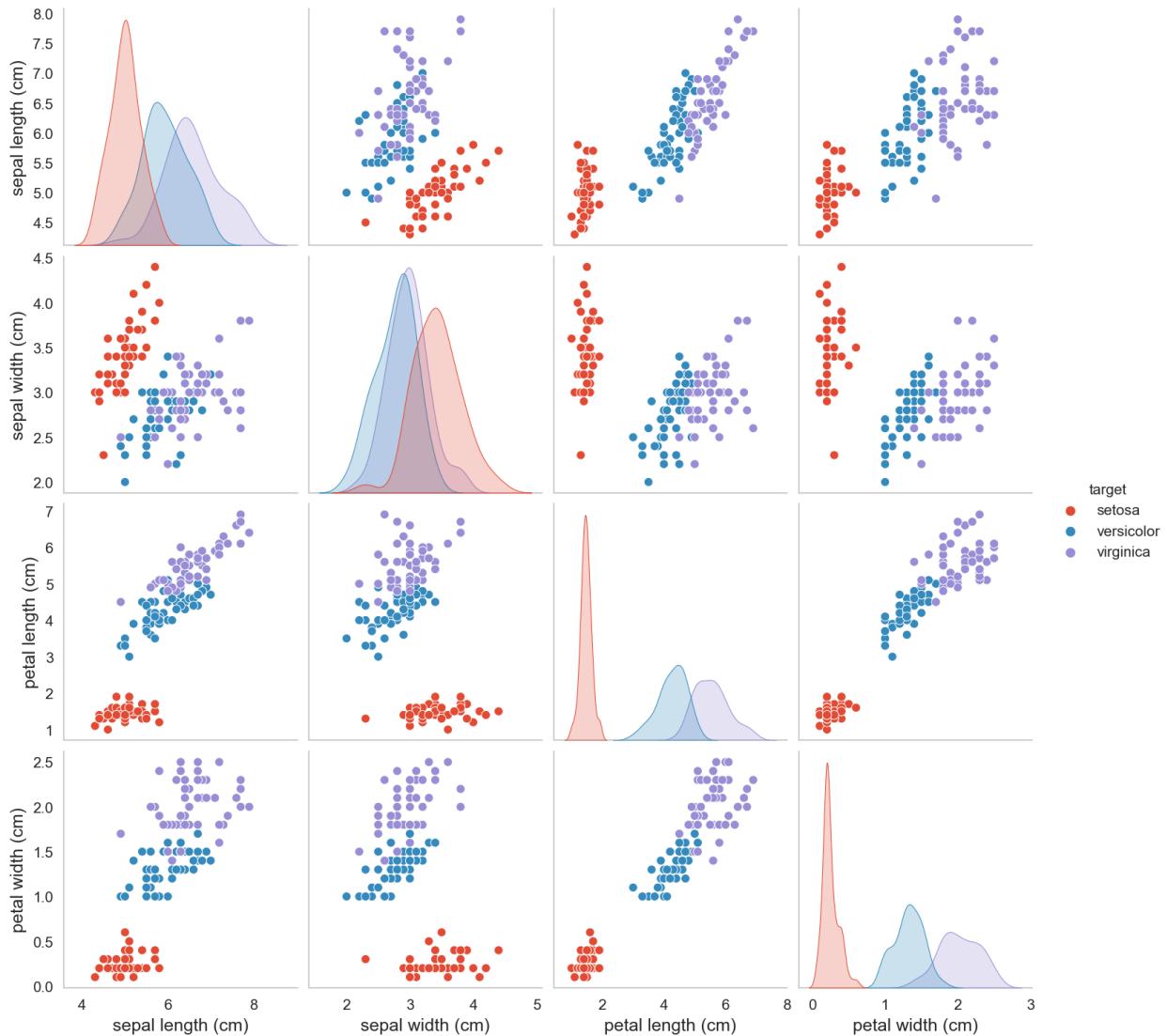
  

	target
41	setosa
125	virginica
53	versicolor
40	setosa
3	setosa

## 4.2 A first inspection of the data

We use a pair plot to inspect the table. Each target species is assigned a color to allow conclusions regarding clusters.

```
sns.pairplot(iris_df, hue='target');
```



In the plot, we clearly see that one species (setosa) in general has other flower dimensions than the other two.

## 4.3 Comparing Groups: Intraclass Correlation Coefficient

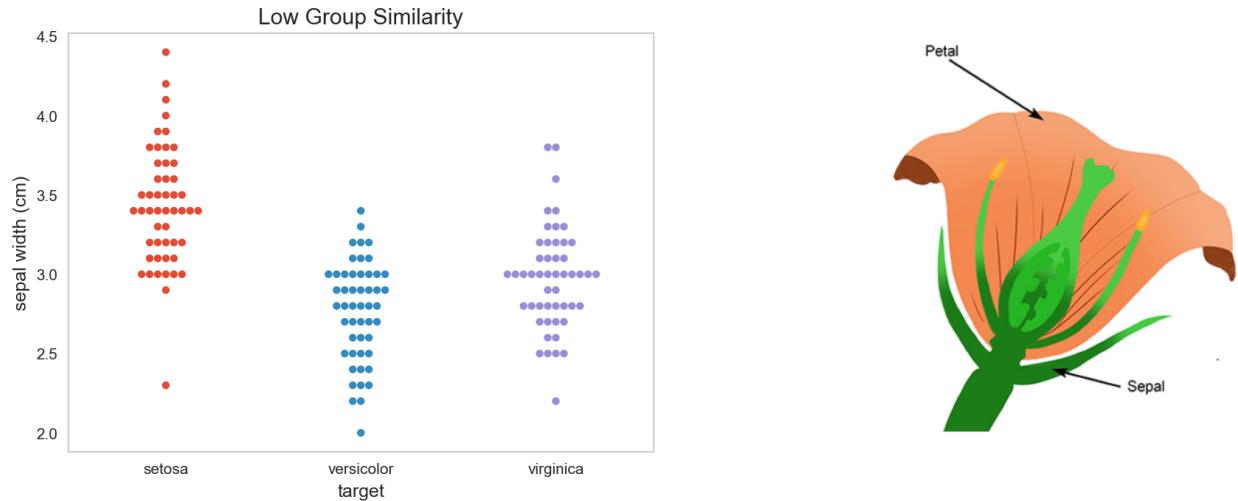
The intraclass correlation coefficient basically looking at

- how similar objects within a group are
- compared to the similarity between groups

### 4.3.1 Sepal width

Sepals are the green leaves of the flower bud. In this swarm plot we look at the width of the sepals and see that the variance of each class is about the same and also the the average width doesn't vary much. Under such conditions it is hard to separate the groups from each other and we are talking about a *low group similarity*.

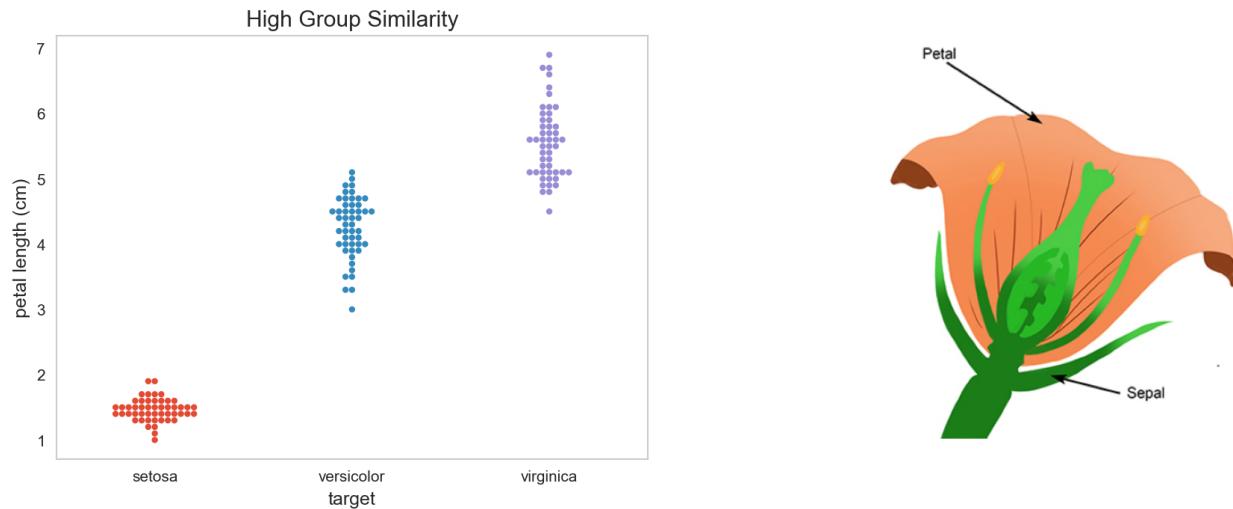
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
sns.swarmplot(data=iris_df, ax = ax1,
               x='target', y='sepal width (cm)'); ax1.set_title('Low Group Similarity
               ↵');
ax2.imshow(plt.imread('figures/FlowerAnatomy.png')); ax2.axis('off');
```



### 4.3.2 Petal length

Petals are the colourful and beautiful leaves of the flower. In this swarm plot of the petal length we see that the petals are more clustered and the averages are well separated from each other. This is a case we know is easy to separate the groups and we are talkning about data with a *high group similarity*.

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
g = sns.swarmplot(data=iris_df, ax=ax1,
                   x='target', y='petal length (cm)', size=4); g.set_title('High Group
                   ↵ Similarity');
ax2.imshow(plt.imread('figures/FlowerAnatomy.png')); ax2.axis('off');
```



## 4.4 Making quantitative statements

### 4.4.1 Intraclass Correlation Coefficient Definition

$$ICC = \frac{S_A^2}{S_A^2 + S_W^2}$$

where

- $S_A^2 = s[\mathbb{E}[x_{group}]]^2$  is the variance among groups or classes
- Estimate with the standard deviations of the mean values for each group
- $S_W^2 = \mathbb{E}[s[x_{group}]^2]$  is the variance within groups or classes.
- Estimate with the average of standard deviations for each group

#### Interpretation

$$ICC = \begin{cases} 1 & \text{means 100 percent of the variance is between classes} \\ 0 & \text{means 0 percent of the variance is between classes} \end{cases}$$

### 4.4.2 Intraclass Correlation Coefficient: Values

```
def icc_calc(value_name, group_name, data_df):
    data_agg = data_df.groupby(group_name).agg({value_name: ['mean', 'var']}).reset_index()
    data_agg.columns = data_agg.columns.get_level_values(1)
    S_w = data_agg['var'].mean()
    S_a = data_agg['mean'].var()
    return S_a/(S_a+S_w)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))
sns.swarmplot(data=iris_df, ax=ax1,
               x='target', y='sepal width (cm)', size=3)
```

(continues on next page)

(continued from previous page)

```

ax1.set_title('Low Group Similarity\nICC:{:2.1%}'.format(icc_calc('sepal width (cm)',  

    ↪'target', iris_df)));  
  

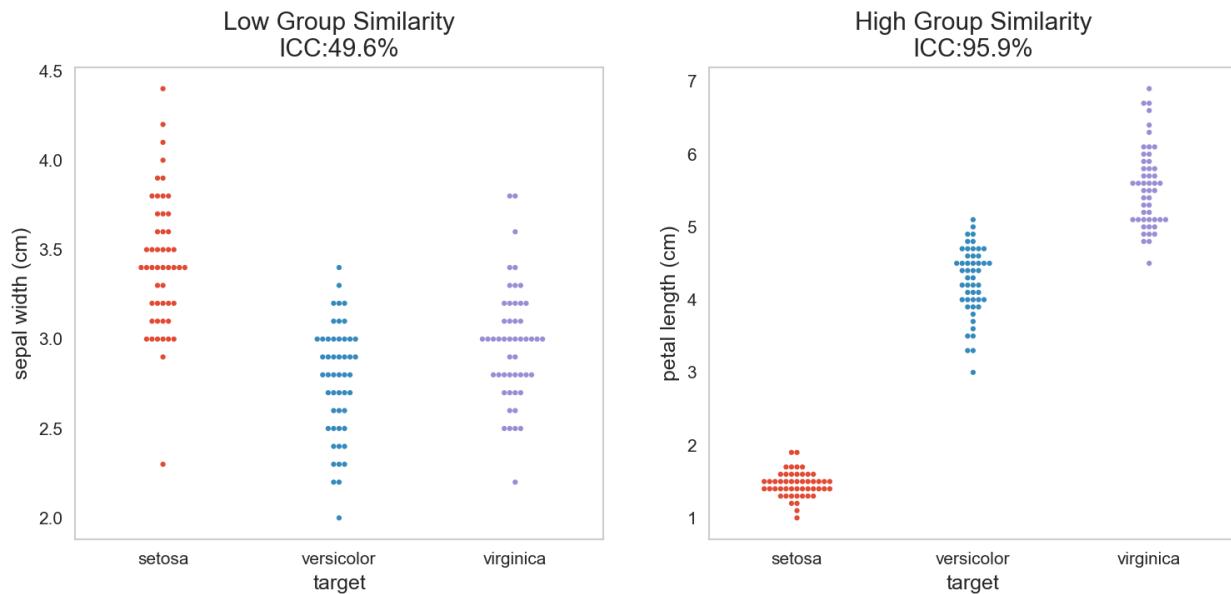
sns.swarmplot(data=iris_df,ax=ax2,  

    x='target', y='petal length (cm)',size=3)  

ax2.set_title('High Group Similarity\nICC:{:2.1%}'.format(icc_calc('petal length (cm)  

    ↪', 'target', iris_df)));

```





## COMPARING GROUPS

Once the reproducibility has been measured, it is possible to compare groups.

The idea is to make a test to assess the likelihood that two groups are the same given the data

1. List assumptions
2. Establish a null hypothesis
  - Usually that both groups are the same
1. Calculate the probability of the observations given the truth of the null hypothesis
  - Requires knowledge of probability distribution of the data
  - Modeling can be exceptionally complicated

### 5.1 Loaded Coin example

We have 1 coin from a magic shop.

- Our assumptions are:
- we flip and observe flips of coins accurately and independently
- the coin is invariant and always has the same expected value
- Our null hypothesis is the coin is unbiased  $E(\mathcal{X}) = 0.5$
- we can calculate the likelihood of a given observation given the number of flips ([p-value](#))

How good is good enough?

### 5.2 Comparing Groups: Student's T Distribution

- Since we do not usually know our distribution very well
- *or* have enough samples to create a sufficient probability model

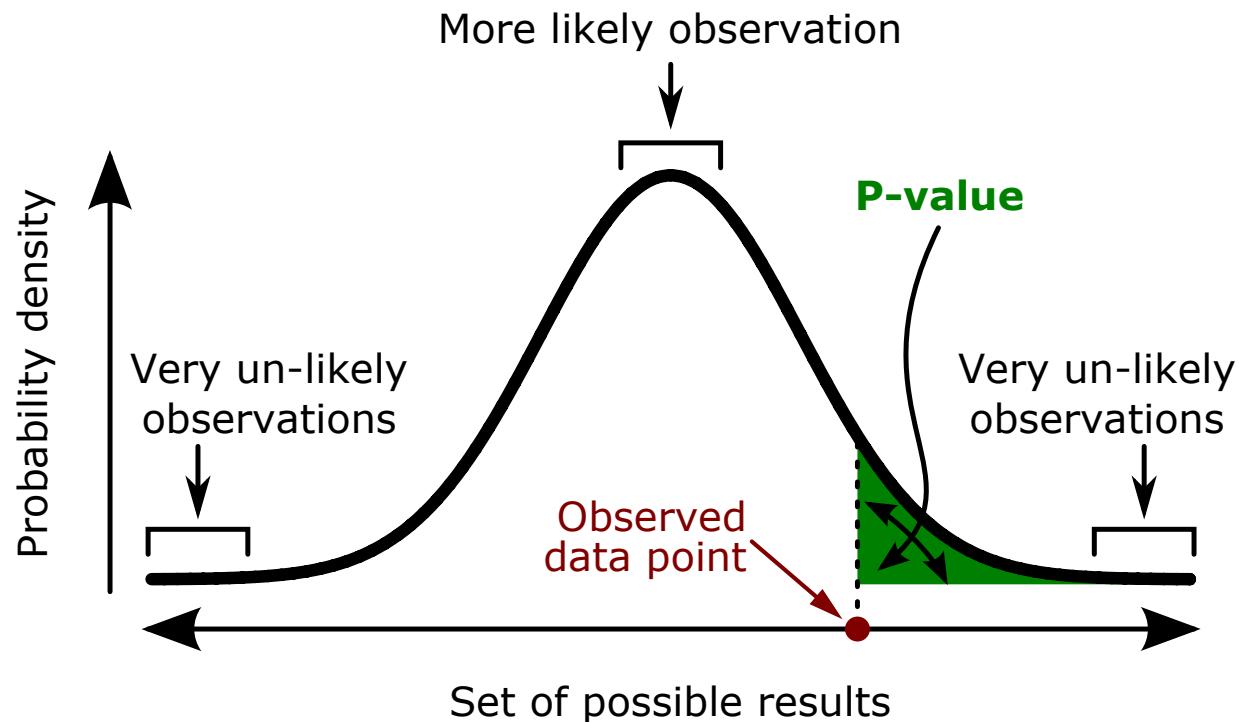


Fig. 5.1: Explaining p-value with the normal distribution.

### 5.2.1 Student T Distribution

We assume the distribution of our stochastic variable is normal (Gaussian) and the t-distribution provides an estimate for the mean of the underlying distribution based on few observations.

- We estimate the likelihood of our observed values assuming they are coming from random observations of a normal process

### 5.2.2 Student T-Test

Incorporates this distribution and provides an easy method for assessing the likelihood that the two given set of observations are coming from the same underlying process (null hypothesis)

- Assume unbiased observations
- Assume normal distribution

## MULTIPLE TESTING BIAS

Back to the magic coin, let's assume we are trying to publish a paper,

- we heard a p-value of  $< 0.05$  (5%) was good enough.
- That means if we get 5 heads we are good!

### 6.1 Probability with increasing number of tosses

$$P = \prod_i P(\mathcal{F}_i(\mathcal{X}))$$

```
import pandas as pd
from scipy.stats import ttest_ind
from IPython.display import display
all_heads_df = pd.DataFrame({'n_flips': [1, 4, 5]})
all_heads_df['Probability of # Heads'] = all_heads_df['n_flips'].map(
    lambda x: '{:2.1%}'.format(0.5**x))
display(all_heads_df)
```

n_flips	Probability of # Heads
0	50.0%
1	6.2%
2	3.1%

### 6.2 Probability with many experiments

Let N friends make 5 tosses...

```
friends_heads_df = pd.DataFrame({'n_friends': [1, 10, 20, 40, 80]})
friends_heads_df['Probability of 5 Heads'] = friends_heads_df['n_friends'].map(
    lambda n_friends: '{:2.1%}'.format((1-(1-0.5**5)**n_friends)))
display(friends_heads_df)
```

n_friends	Probability of 5 Heads
0	3.1%
1	27.2%
2	47.0%
3	71.9%
4	92.1%

Clearly this is not the case, otherwise we could keep flipping coins or ask all of our friends to flip until we got 5 heads and publish

The p-value is only meaningful when the experiment matches what we did.

- We didn't say the chance of getting 5 heads ever was < 5%
- We said is if we have
  - exactly 5 observations
  - and all of them are heads
  - the likelihood that a fair coin produced that result is <5%

There are many methods to correct.

Most just involve scaling  $p$ :

- The likelihood of a sequence of 5 heads in a row if you perform 10 flips is 5x higher.

## 6.3 Multiple Testing Bias: Experiments

This is very bad news for us. We have the ability to quantify all sorts of interesting metrics

- cell distance to other cells
- cell oblateness
- cell distribution oblateness

So lets throw them all into a magical statistics algorithm and push the **publish** button

With our  $p$  value of less than 0.05 and a study with 10 samples in each group, how does increasing the number of variables affect our result

```
import pandas as pd
import numpy as np
pd.set_option('precision', 2)
np.random.seed(2017)

def random_data_maker(rows, cols):
    data_df = pd.DataFrame(
        np.random.uniform(-1, 1, size=(rows, cols)),
        columns=['Var_{}'.format(c_col) for c_col in range(cols)])
    data_df['Group'] = [1]*((rows-rows//2)+[2]*((rows//2)))
    return data_df

rand_df = random_data_maker(10, 5)

rand_df
```

	Var_00	Var_01	Var_02	Var_03	Var_04	Group
0	-0.96	0.53	-0.10	-0.76	0.86	1
1	0.30	-0.72	-0.54	-0.55	-0.48	1
2	-0.77	0.26	-0.23	-0.37	0.26	1
3	-0.41	0.89	-0.70	-0.85	0.41	1
4	-0.86	-0.39	-0.34	-0.38	-0.12	1
5	0.53	-0.05	-0.99	0.40	0.26	2
6	-0.94	-0.83	0.41	-0.09	0.41	2

(continues on next page)

(continued from previous page)

7	0.86	-0.18	-0.92	0.24	-0.28	2
8	0.84	0.83	-0.46	-0.39	-0.97	2
9	0.08	0.34	-0.09	0.07	0.82	2

```
from scipy.stats import ttest_ind

def show_significant(in_df, cut_off=0.05):
    return in_df.sort_values('P-Value').style.apply(lambda x: ['background-color: yellow' if v<cut_off else '' for v in x])

def all_ttest(in_df):
    return pd.DataFrame(
        {'P-Value': {c_col: ttest_ind(
            a=in_df[in_df['Group'] == 1][c_col],
            b=in_df[in_df['Group'] == 2][c_col]
        ).pvalue
            for c_col in
            in_df.columns if 'Group' not in c_col)}})

show_significant(all_ttest(rand_df))
```

<pandas.io.formats.style.Styler at 0x7fd641a64940>

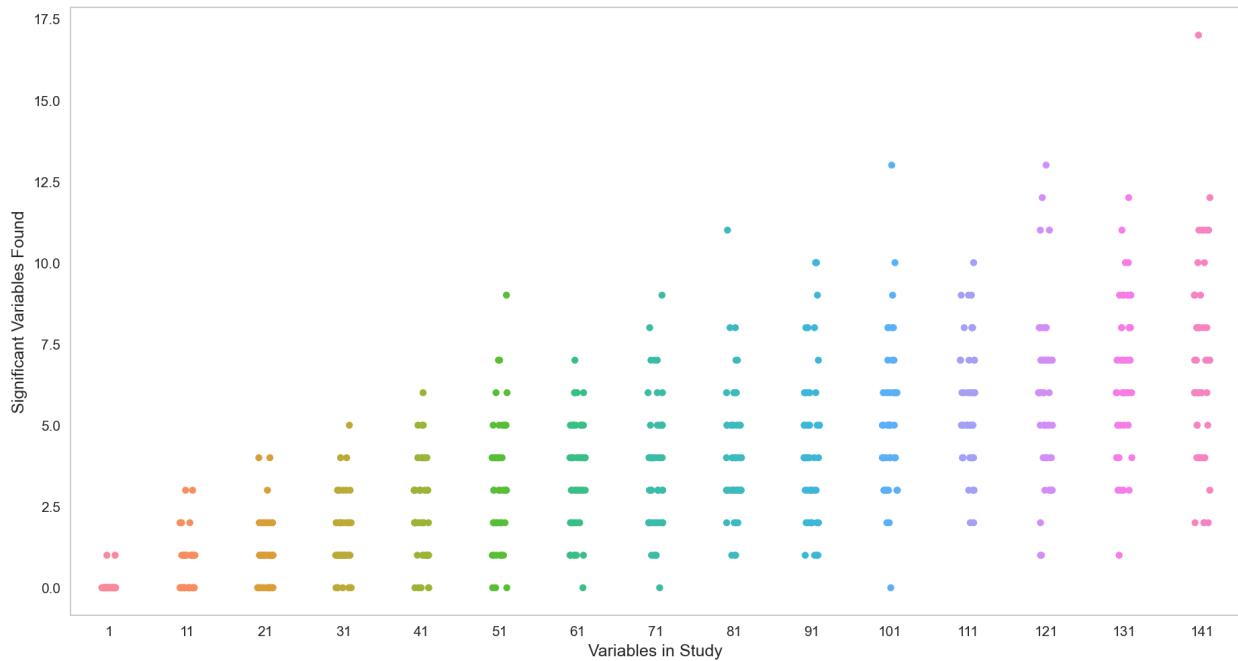
```
np.random.seed(2019)
show_significant(all_ttest(random_data_maker(150, 20)))
```

<pandas.io.formats.style.Styler at 0x7fd681314e50>

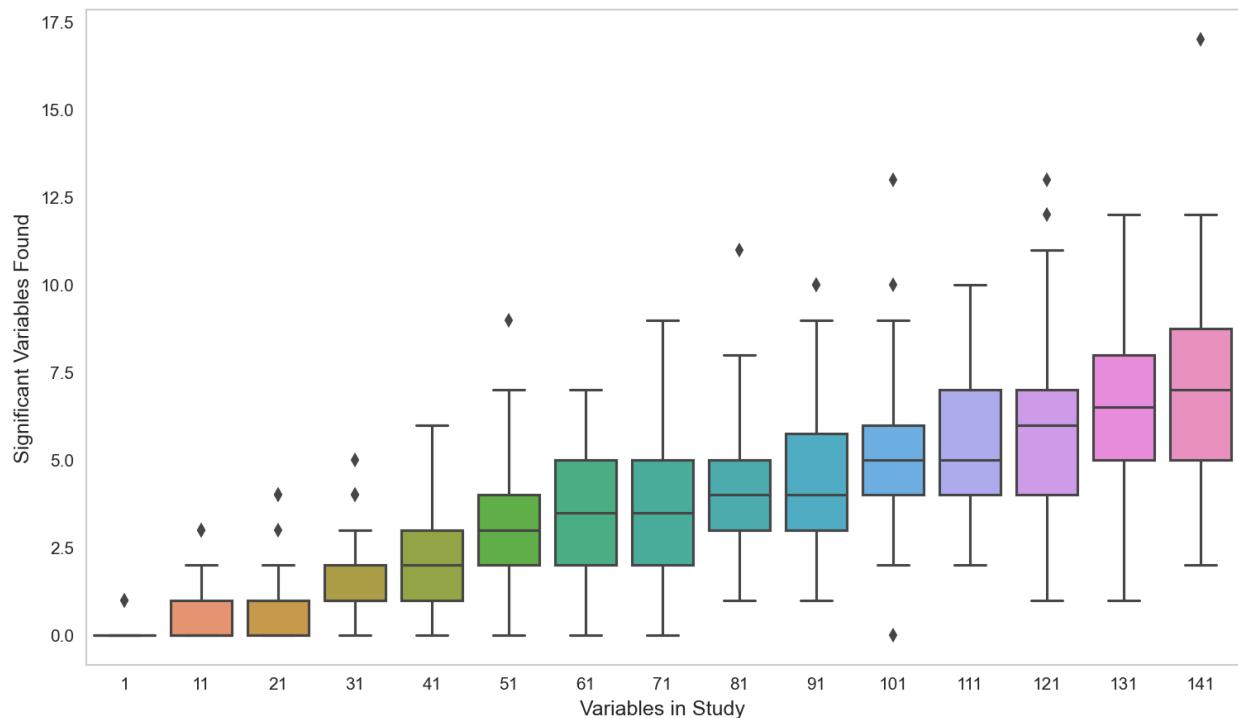
```
import seaborn as sns
from tqdm import notebook # progressbar
out_list = []
for n_vars in notebook.tqdm(range(1, 150, 10)):
    for _ in range(50):
        p_values = all_ttest(random_data_maker(100, n_vars)).values
        out_list += [{ 'Variables in Study': n_vars,
                      'Significant Variables Found': np.sum(p_values < 0.05),
                      'raw_values': p_values}]
var_found_df = pd.DataFrame(out_list)
```

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=15.0), HTML(value='')))

```
fig,ax = plt.subplots(1,1, figsize=(15,8))
sns.stripplot(data=var_found_df, x='Variables in Study', y='Significant Variables Found');
```



```
plt.figure(figsize=(12,7))
sns.boxplot(data=var_found_df,
             x='Variables in Study', y='Significant Variables Found');
```



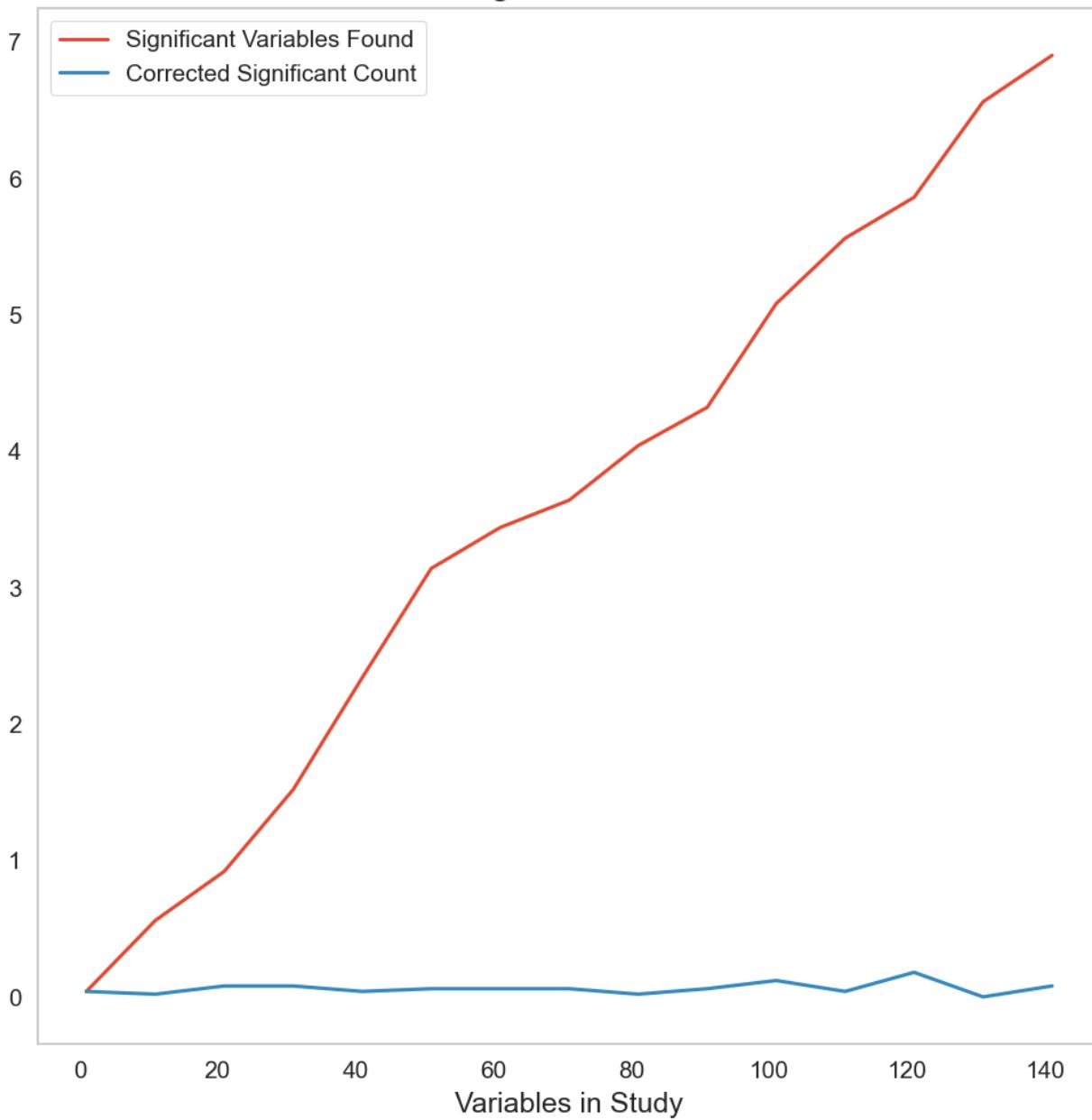
## MULTIPLE TESTING BIAS: CORRECTION

Using the simple correction factor (number of tests performed), we can make the significant findings constant again.  
 $p_{cutoff} = \frac{0.05}{\text{Number of Tests}}$

```
var_found_df['Corrected Significant Count'] = var_found_df['raw_values'].map(lambda p_
    ↪values:
        np.sum(p_
    ↪values<0.05/len(p_values)))

var_found_df.groupby('Variables in Study').agg('mean').reset_index().plot('Variables_
    ↪in Study', [
        'Significant Variables Found',
        'Corrected Significant Count'
]);
plt.title('Effect of significance correction');
```

### Effect of significance correction



So no harm done there we just add this correction factor right?

Well, what if we have exactly one variable with shift of 1.0 standard deviations from the other.

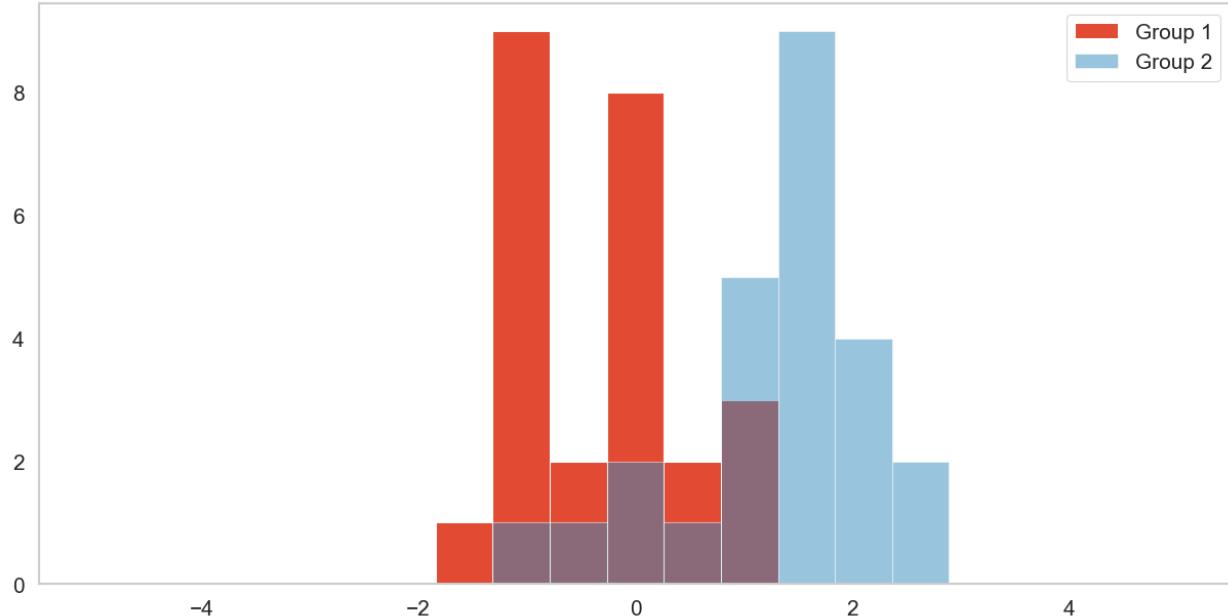
In a dataset where we check  $n$  variables?

```
table_df = random_data_maker(50, 10)
really_different_var = np.concatenate([
    np.random.normal(loc=0, scale=1.0, size=(table_df.shape[0]//2)),
    np.random.normal(loc=1, scale=1.0, size=(table_df.shape[0]//2))
])
table_df['Really Different Var'] = really_different_var
fig, ax1 = plt.subplots(1, 1, figsize=(10, 5))
```

(continues on next page)

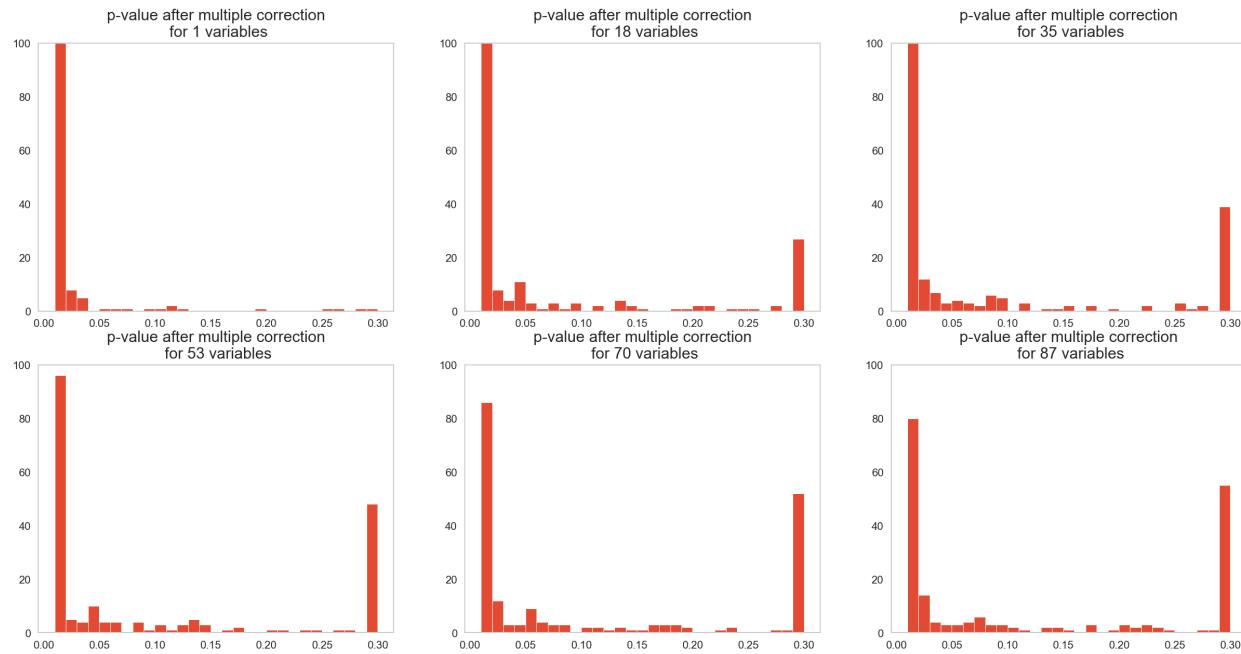
(continued from previous page)

```
ax1.hist(table_df.query('Group==1')['Really Different Var'], np.linspace(-5, 5, 20),  
        label='Group 1')  
ax1.hist(table_df.query('Group==2')['Really Different Var'], np.linspace(-5, 5, 20),  
        label='Group 2', alpha=0.5);  
ax1.legend();
```

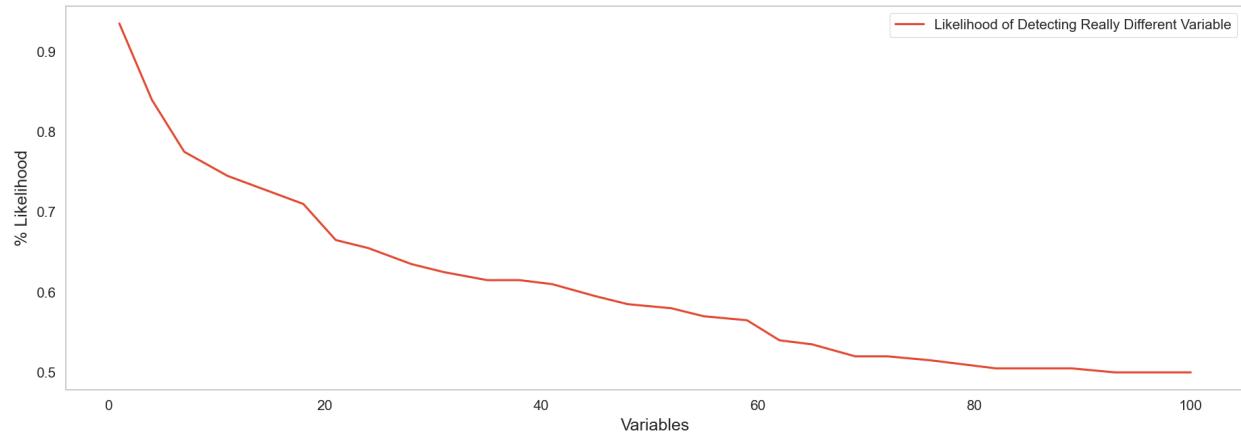


```
out_p_value = []  
for _ in range(200):  
    out_p_value += [ttest_ind(np.random.normal(loc=0, scale=1.0, size=(table_df.  
shape[0]//2)),  
                           np.random.normal(loc=1, scale=1.0, size=(table_df.shape[0]//2))).pvalue]
```

```
fig, m_axs = plt.subplots(2, 3, figsize=(20, 10))  
for c_ax, var_count in zip(m_axs.flatten(), np.linspace(1, 140, 9).astype(int)):  
    c_ax.hist(np.clip(np.array(out_p_value)*var_count, 0.01, 0.3), np.linspace(0.01,  
        0.3, 30))  
    c_ax.set_yscale('log')  
    c_ax.set_title('p-value after multiple correction\nfor {} variables'.format(var_  
count))
```



```
var_find_df = pd.DataFrame({'Variables': np.linspace(1, 100, 30).astype(int)})
var_find_df['Likelihood of Detecting Really Different Variable'] = var_find_df[
    ↪'Variables'].map(
    lambda var_count: np.mean(np.array(out_p_value)*var_count<0.05)
)
fig, ax1 = plt.subplots(1, 1, figsize=(15, 5))
var_find_df.plot('Variables', 'Likelihood of Detecting Really Different Variable', ↪
    ↪ax=ax1)
ax1.set_ylabel('% Likelihood');
```



## PREDICTING AND VALIDATING - MAIN CATEGORIES

There are plenty machine-learning techniques available. Each one dedicated to a specific type of problem and data collection. Fig <number> fig\_mlcheatsheet provides a guide to find the correct method for your problem.

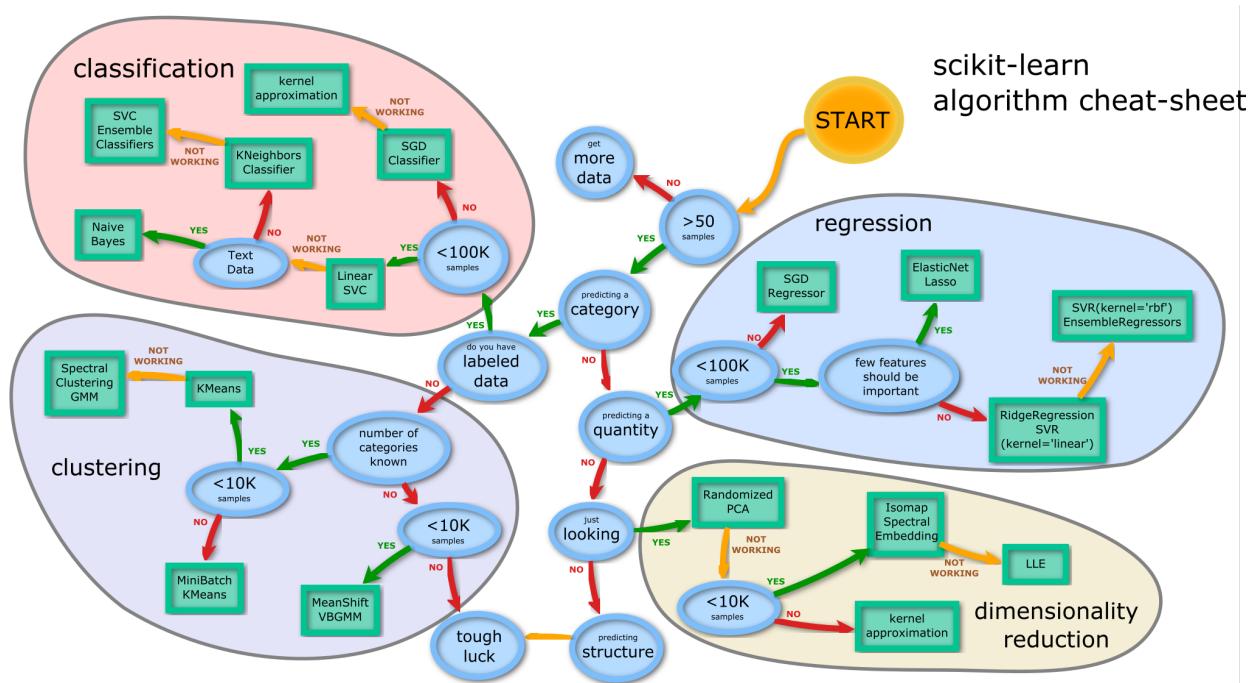


Fig. 8.1: A cheat sheet to identify the best machine learning technique for your problem.

A common task independent of which method you chose, it that you have to validate your processing workflow. This is important to be able to tell when and to what degree you can trust your workflow.

Borrowed from <http://peekaboo-vision.blogspot.ch/2013/01/machine-learning-cheat-sheet-for-scikit.html>

## 8.1 Overview

Basically all of these are ultimately functions which map inputs to outputs.

### 8.1.1 The input could be

- an image
- a point
- a feature vector
- or a multidimensional tensor

### 8.1.2 The output is

- a value (regression)
- a classification (classification)
- a group (clustering)
- a vector / matrix / tensor with *fewer* degrees of input / less noise as the original data (dimensionality reduction)

### 8.1.3 Overfitting

The most serious problem with machine learning and such approaches is overfitting your model to your data. Particularly as models get increasingly complex (random forest, neural networks, deep learning, ...), it becomes more and more difficult to apply common sense or even understand exactly what a model is doing and why a given answer is produced.

```
magic_classifier = {}  
# training  
magic_classifier['Dog'] = 'Animal'  
magic_classifier['Bob'] = 'Person'  
magic_classifier['Fish'] = 'Animal'
```

Now use this classifier, on the training data it works really well

```
magic_classifier['Dog'] == 'Animal' # true, 1/1 so far!  
magic_classifier['Bob'] == 'Person' # true, 2/2 still perfect!  
magic_classifier['Fish'] == 'Animal' # true, 3/3, wow!
```

On new data it doesn't work at all, it doesn't even execute.

```
magic_classifier['Octopus'] == 'Animal' # exception?! but it was working so well  
magic_classifier['Dan'] == 'Person' # exception?!
```

The above example appeared to be a perfect trainer for mapping names to animals or people, but it just memorized the inputs and reproduced them at the output and so didn't actually learn anything, it just copied.

## 8.2 Validation

Relevant for each of the categories, but applied in a slightly different way depending on the group.

The idea is to divide the dataset into groups called

- ideally training,
- validation,
- and testing.

The analysis is then

- developed on **training**
- iteratively validated on **validation**
- ultimately tested on **testing**

## 8.3 Concrete Example: Classifying Flowers

Here we return to the iris data set and try to automatically classify flowers

```
data = load_iris()
iris_df = pd.DataFrame(data['data'], columns=data['feature_names'])
iris_df['target'] = data['target_names'][data['target']]
iris_df.sample(5)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	\
86	6.7	3.1	4.7	1.5	
132	6.4	2.8	5.6	2.2	
47	4.6	3.2	1.4	0.2	
29	4.7	3.2	1.6	0.2	
62	6.0	2.2	4.0	1.0	
	target				
86	versicolor				
132	virginica				
47	setosa				
29	setosa				
62	versicolor				

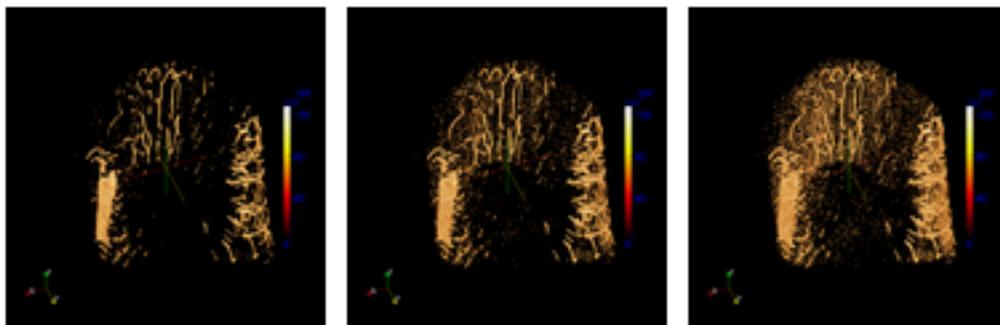


## QUALITATIVE VS QUANTITATIVE

Given the complexity of the tree, we need to do some pruning

### 9.1 Qualitative Assessment

- Evaluating metrics using visual feedback
- Compare with expectations from other independent techniques or approach
- Are there artifacts which are included in the output?
- Do the shapes look correct?
- Are they distributed as expected?
- Is their orientation meaningful?



### 9.2 Quantitative Metrics

With a quantitative approach, we can calculate

- the specific shape
- or distribution metrics on the sample

with each parameter and establish the relationship between

- parameter
- and metric.

## 9.3 Parameters

```
from graphviz import Digraph

dot = Digraph()

dot.node('Raw images',color='limegreen'),           dot.node('Gaussian filter', color=
˓→'lightblue')
dot.node('sigma=0.5', color='gray',shape='box'),   dot.node('3x3 Neighbors', color='gray'
˓→',shape='box')
dot.node('Threshold', color='lightblue'),          dot.node('100', color='gray',shape=
˓→'box')
dot.node('Thickness analysis',color='hotpink'),    dot.node('Shape analysis',color=
˓→'hotpink')
dot.node('Input',color='limegreen'),               dot.node('Functions', color='lightblue')
dot.node('Parameters', color='gray',shape='box'),  dot.node('Output',color='hotpink')

dot.edge('Raw images', 'Gaussian filter'),        dot.edge('sigma=0.5', 'Gaussian filter')
dot.edge('3x3 Neighbors', 'Gaussian filter'),    dot.edge('Gaussian filter','Threshold')
dot.edge('Threshold', 'Thickness analysis'),      dot.edge('Threshold', 'Shape analysis')
dot.edge('100','Threshold')
dot
```

```
<graphviz.dot.Digraph at 0x7fd68057cbe0>
```

### 9.3.1 Parameter Sweep

The way we do this is usually a parameter sweep which means

- taking one (or more) parameters
- and varying them between the reasonable bounds (judged qualitatively).

The outcome of a parameter sweep can look like in the figure below.

We can see that the volume is generally decreasing when the threshold increases. Still, it seems that the volume is not that sensitive to the choice of threshold. Variations in the order of about 50 voxels. That would correspond to a radius change from 6.3 to 6.7 for the equivalent spheres. On the other hand, this minor change could be the difference between separated or touching objects.

### 9.3.2 Is it always the same?

The jittered scatter plot in fig 9.1 makes it hard to see the distribution of the measurements. A violin plot as in the figure below is a histogram view of the data that allows stacking for different observations.

Now, what happens if we look at a different item metric like the orientation of the items.

Here, we see a similar trend as we saw with the volume.

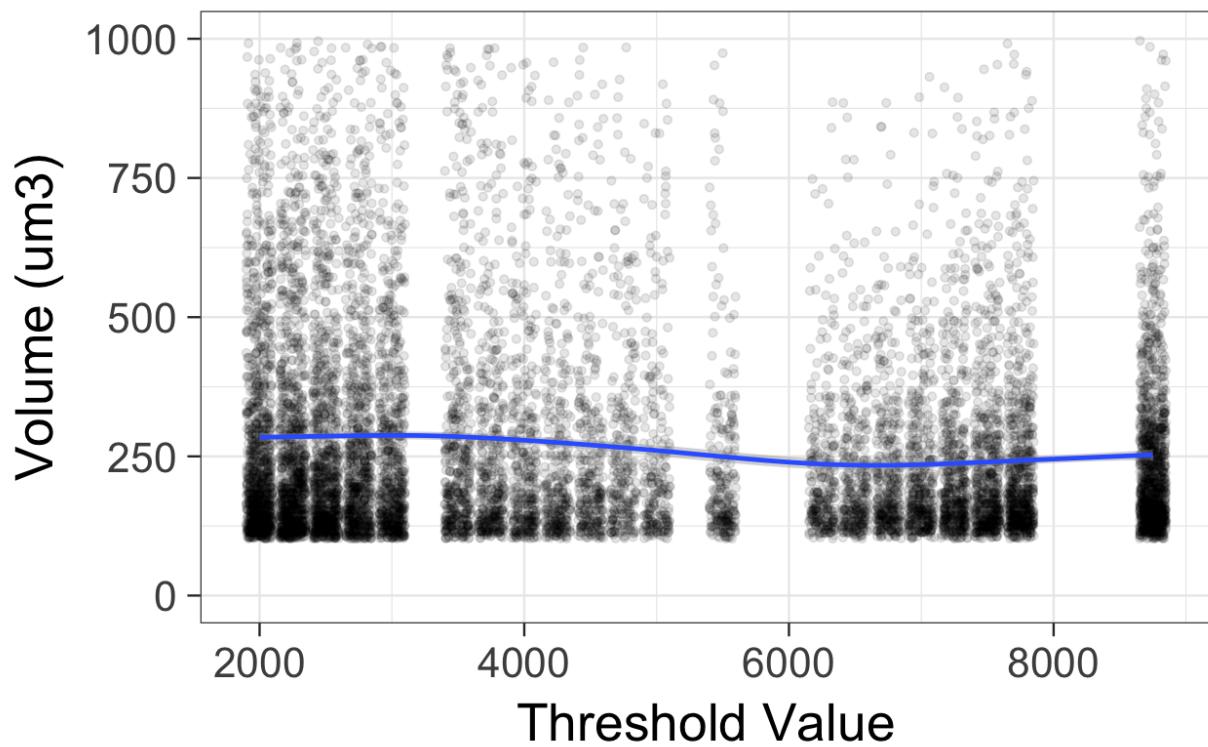


Fig. 9.1: Volume measurements for different thresholds

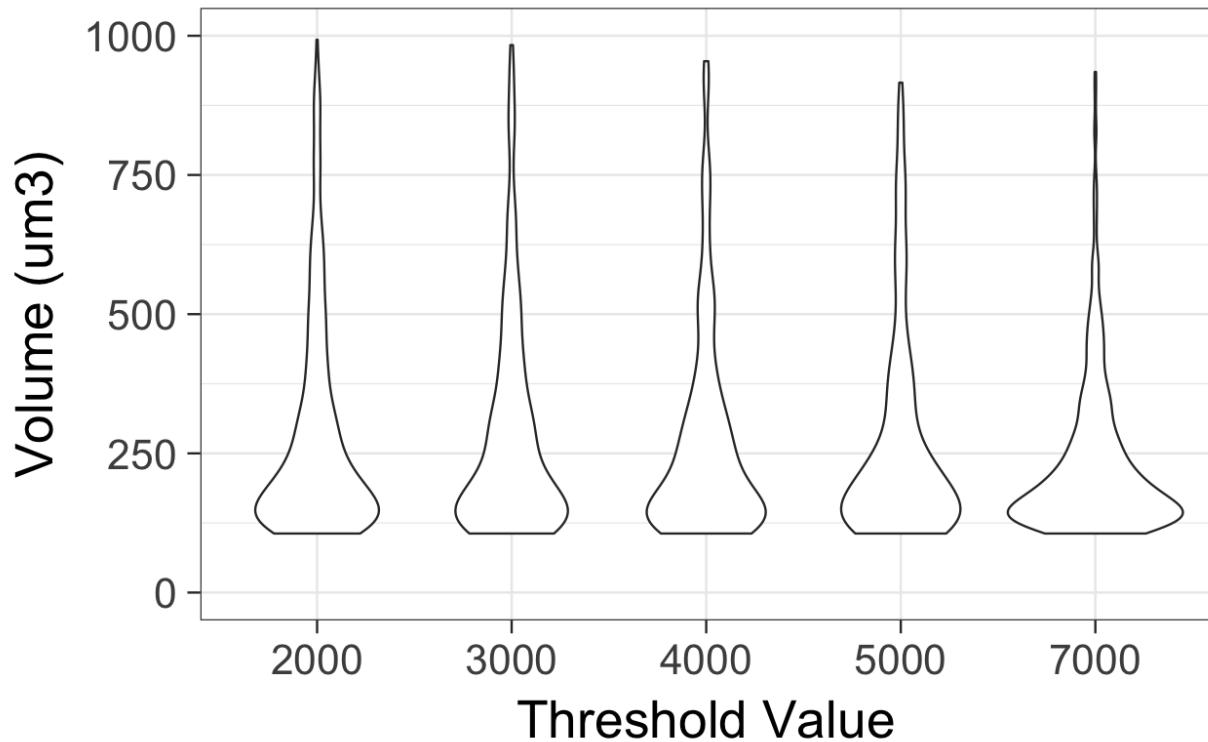


Fig. 9.2: A violin plot of the volume data.

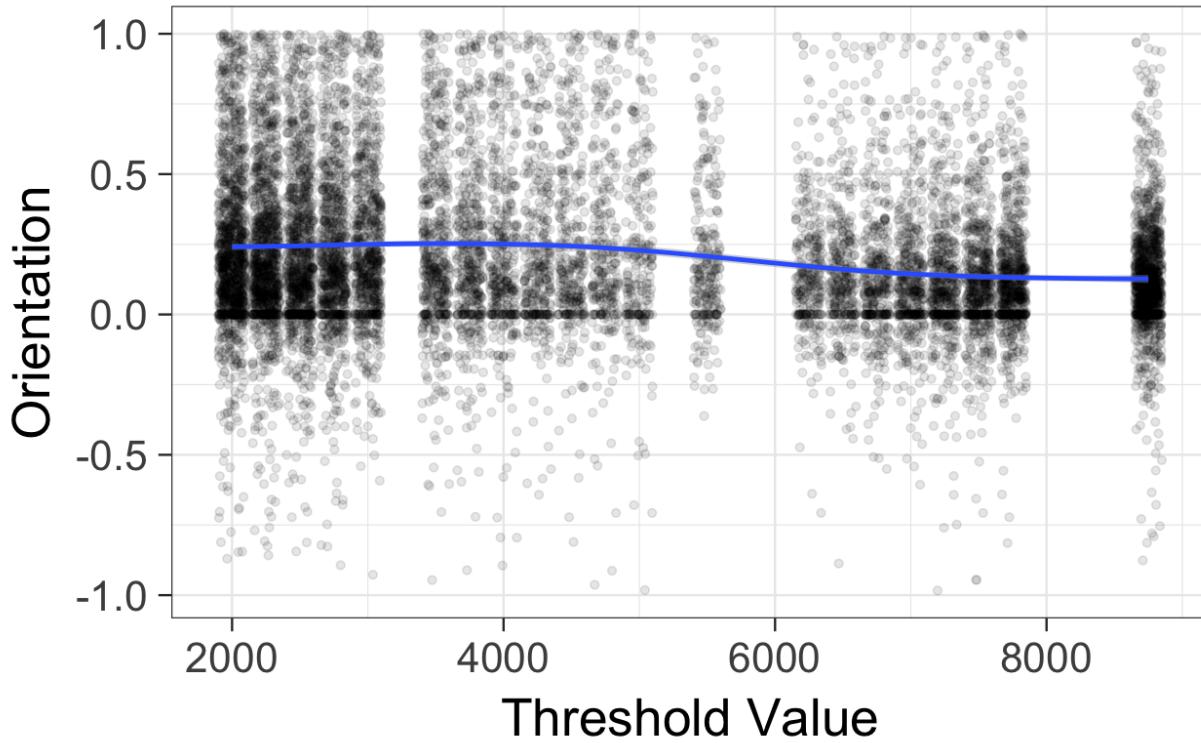


Fig. 9.3: Scatter plot of the orientation of the items.

## 9.4 Sensitivity

### 9.4.1 Control system theory

Sensitivity is defined as

- the change in the value of an output
- against the change in the input.

$$S = \frac{|\Delta \text{Metric}|}{|\Delta \text{Parameter}|}$$

### 9.4.2 Image processing

Such a strict definition is not particularly useful for image processing since

- a threshold has a unit of intensity and
  - a metric might be volume which has  $m^3$
- the sensitivity becomes volume per intensity!

### 9.4.3 Practical Sensitivity

A more common approach is to estimate the variation in this parameter between images or within a single image (automatic threshold methods can be useful for this) and define the sensitivity based on this variation.

It is also common to normalize it with the mean value so the result is a percentage.

$$S = \frac{\max(\text{Metric}) - \min(\text{Metric})}{\text{avg}(\text{Metric})}$$

## 9.5 Sensitivity: Real Measurements

In this graph it is magnitude of the slope. The steeper the slope the more the metric changes given a small change in the parameter

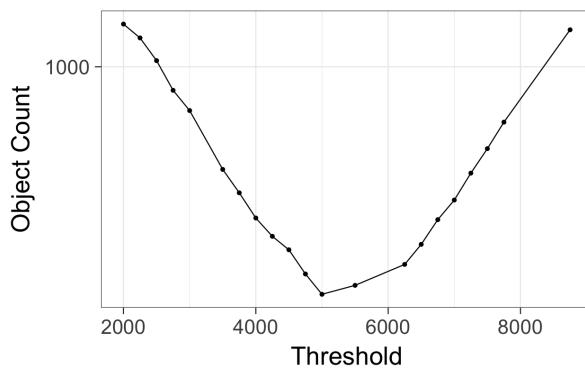


Fig. 9.4: Sensitivity measurement to measure how sensitive the object count is to the choice of the threshold.

## 9.6 Sensitivity: compare more than one variable

Comparing Different Variables we see that

- the best (lowest) value for the count sensitivity
- is the highest for the volume and anisotropy.

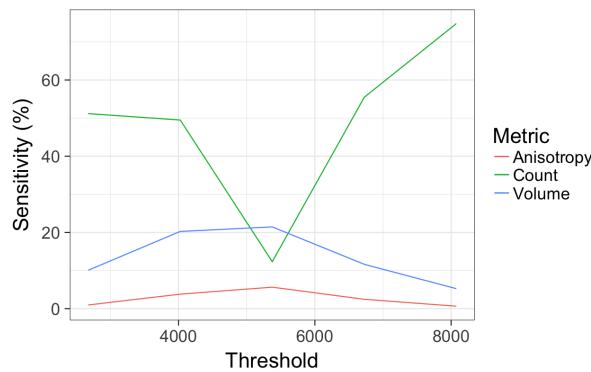


Fig. 9.5: Sensitivity comparison for different metrics (Anisotropy, Count, Volume) to the choice of the threshold.

### 9.6.1 A contradiction - Which metric is more important?

We see in Fig <number> fig\_comparisonsensitivity that two parameters with relatively low sensitivity variations behave the same while the last one (*count*) fluctuates a lot with the threshold choice. Which one we use to guide our segmentation ultimately depends on the objective of the investigation.

## 9.7 Reproducibility

A very broad topic with plenty of sub-areas and deeper meanings. We mean two things by reproducibility

### 9.7.1 Analysis

The process of going from images to numbers is detailed in a clear manner that **anyone**, **anywhere** could follow and get the exact (within some tolerance) same numbers from your samples

- No platform dependence
- No proprietary or “in house” algorithms
- No manual *clicking*, *tweaking*, or *copying*
- A single script to go from image to result

### 9.7.2 Measurement

Everything for analysis + taking a measurement several times (noise and exact alignment vary each time) does not change the statistics *significantly*

- No sensitivity to mounting or rotation
- No sensitivity to noise
- No dependence on exact illumination

## 9.8 Reproducible Analysis

Since we will need to perform the same analysis many times to understand how reproducible it is.

- Notebooks are good to develop and document analysis workflow.
- The basis for reproducible analysis are scripts and macros.

### 9.8.1 With python scripts

```
# #!/${PYTHONPATH}/python
import sys
from myAnalysis import analysisScript # some analysis script you implemented

imageFile = sys.argv[0] # File name from command line

threshold = 130
analysisScript(fname=imageFile, threshold = threshold)
```

### 9.8.2 or Matlab, ImageJ, or R

```
IMAGEFILE=$1
THRESHOLD=130
matlab -r "inImage=$IMAGEFILE; threshImage=inImage>$THRESHOLD; analysisScript;"
```

- **or** java -jar ij.jar -macro TestMacro.ijm blobs.tif
- **or** Rscript -e "library(plyr);..."



## DATA FRAMES - MANAGING FEATURE TABLES

### 10.1 Our workflow

- Image analysis
- Feature selection and analysis
- Presentation

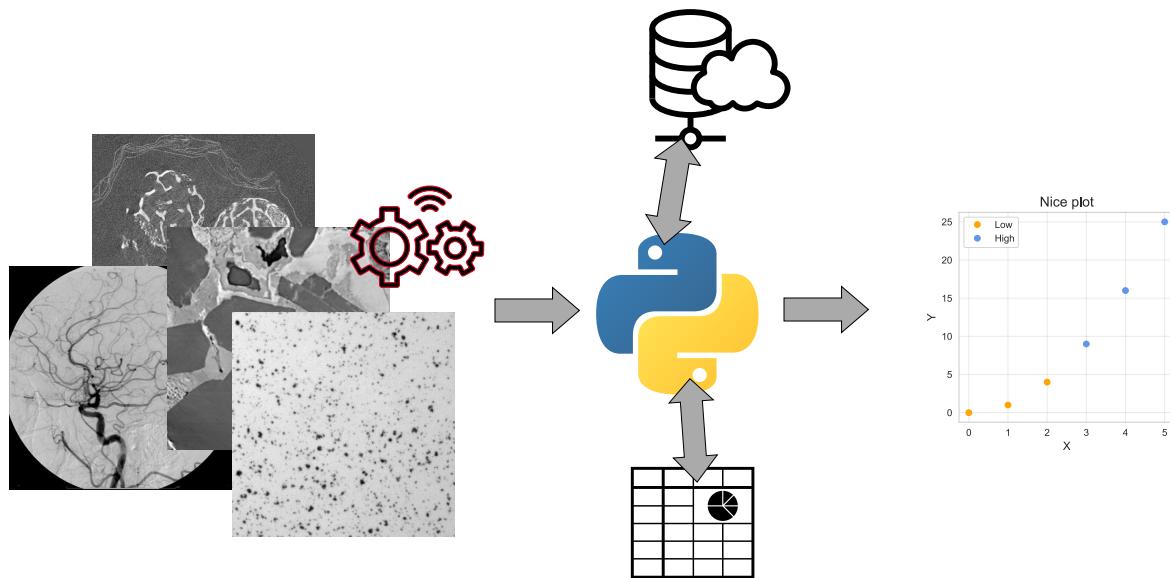


Fig. 10.1: Different destinations of the processed data.

**How do we store the features while working with the data?**

## 10.2 How do we store the features?

### 10.2.1 Python offers different options

- Arrays per feature
- List of data structures/dictionaries

### 10.2.2 Operations on the feature data

- Counting
- Statistics
- Selections
- Transforms
- Visualization

### 10.2.3 Problem

With custom storage we have to implement functions for each operation:

- Time consuming
- Little flexibility
- Error prone

## 10.3 Introducing data frames

We have already seen data frames in action but never formally introduced them...

**A data frame is**

- A data container
- Organized into columns and rows
- Has similarities to a spread sheet table
- Takes any data in the columns

**You can**

- Apply filters for selection
- Sort the rows
- Perform arithmetics
- Compute statistics
- Read and store into files and databases

Pandas documentation [Getting started with Pandas](#)

## 10.4 Create a data frame

There are different ways to create a data frame:

- From a dict
- From a data file
- From numpy arrays

First we have import pandas:

```
import pandas as pd
```

### 10.4.1 Create a data frame from a list of dicts

This is the very basic use case. We create the data frame as the features are produced.

```
dl = []
for i in np.arange(0,10) :
    dl.append({'position' : i, 'sine' : np.sin(i)})

df = pd.DataFrame(dl)

df.head()
```

	position	sine
0	0	0.00
1	1	0.84
2	2	0.91
3	3	0.14
4	4	-0.76

### 10.4.2 Read from a spreadsheet (csv)

Sometimes the features have been extracted elsewhere and stored in a file, e.g. CSV

```
pheno = pd.read_csv('.../Exercises/10-Statistics_DataFrames/phenoTable.csv')
pheno.sample(5)
```

	BMD	MECHANICS_STIFFNESS	CORT.DTO__C.TH	CORT.DTO__C.TH_SD	\	
440	0.03	69.34	0.19	0.02		
27	0.04	77.45	0.19	0.02		
311	0.03	71.76	0.16	0.02		
489	0.04	67.48	0.20	0.02		
403	0.04	74.65	0.19	0.02		
	CORT.MOM__J	CT.TH_RAD	CT.TH_RAD_STD	CANAL_VOLUME	CANAL_COUNT	\
440	0.16	86.27	21.46	54010.57	196.0	
27	0.19	78.40	11.36	59828.28	284.0	
311	0.16	76.91	21.67	72136.45	219.0	
489	0.19	84.27	15.01	27402.68	171.0	
403	0.18	90.61	24.34	92064.94	243.0	

(continues on next page)

(continued from previous page)

	CANAL_DENSITY	...	CANAL_THETA	CANAL_THETA.CV	CANAL_PCA1	\
440	223.50	...	52.05	0.41	216.13	
27	291.98	...	46.75	0.52	232.27	
311	261.51	...	56.69	0.35	241.97	
489	191.88	...	57.49	0.35	255.33	
403	263.96	...	46.41	0.51	208.09	
	CANAL_PCA1.CV	CANAL_PCA2	CANAL_PCA2.CV	CANAL_PCA3	CANAL_PCA3.CV	\
440	0.78	63.99	0.92	28.56	0.59	
27	0.75	76.10	0.74	38.04	0.73	
311	0.90	69.97	0.78	31.37	0.67	
489	1.14	65.88	0.92	32.74	0.63	
403	0.77	72.92	0.86	31.84	0.66	
	FEMALE	ID				
440	0	1278				
27	1	479				
311	0	1079				
489	0	1341				
403	0	1225				

[5 rows x 35 columns]

Saving works similarly:

```
pheno.to_csv('pheno2.csv')
```

## 10.5 Working with columns

### 10.5.1 Add a new column

When we start working on the data, we may need to add a column

```
df['cosine'] = np.cos(df['position'])
df['sum']    = df['sine']+df['cosine']
df.head()
```

	position	sine	cosine	sum
0	0	0.00	1.00	1.00
1	1	0.84	0.54	1.38
2	2	0.91	-0.42	0.49
3	3	0.14	-0.99	-0.85
4	4	-0.76	-0.65	-1.41

### 10.5.2 Select some rows with content filtering

```
df2=df[0<df['sine']]
df2
```

	position	sine	cosine	sum
1	1	0.84	0.54	1.38
2	2	0.91	-0.42	0.49
3	3	0.14	-0.99	-0.85
7	7	0.66	0.75	1.41
8	8	0.99	-0.15	0.84
9	9	0.41	-0.91	-0.50

### 10.5.3 Rename column titles

**Note:** Here is also a different way to create a data frame with dicts and lists.

```
df3 = pd.DataFrame({ "A": [1, 2, 3], "B": [4, 5, 6] })
df3.head()
```

	A	B
0	1	4
1	2	5
2	3	6

```
df3.rename(columns={ "A": "hej", "B": "hopp" })
```

	hej	hopp
0	1	4
1	2	5
2	3	6

## 10.6 Statistics with a data frame

You can easily compute statistics on a data frame:

- mean
- std
- min
- max
- median

```
df.mean()
```

position	4.50
sine	0.20
cosine	0.04
sum	0.24
dtype:	float64

```
df.describe()
```

	position	sine	cosine	sum
count	10.00	10.00	10.00	10.00
mean	4.50	0.20	0.04	0.24
std	3.03	0.69	0.77	1.01
min	0.00	-0.96	-0.99	-1.41
25%	2.25	-0.21	-0.59	-0.63
50%	4.50	0.28	0.07	0.59
75%	6.75	0.80	0.70	0.96
max	9.00	0.99	1.00	1.41

## 10.7 Statistics of filtered data

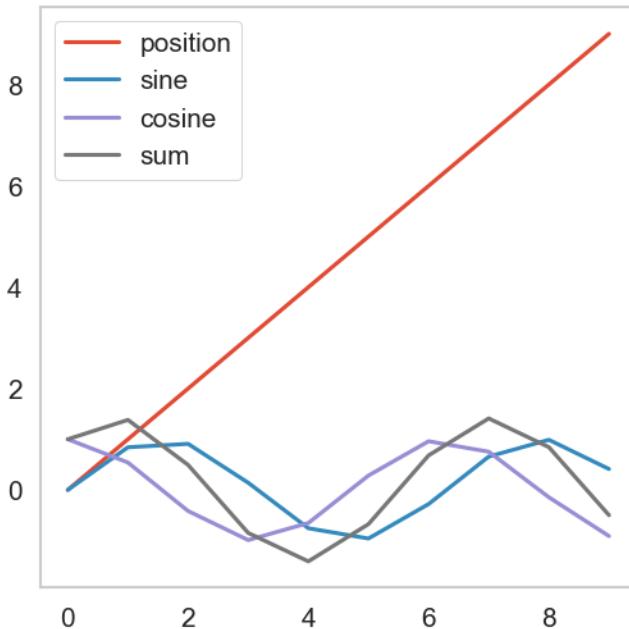
Compute the standard deviation for all columns with the rows have  $0 < \text{sum}$

```
df.head()  
df[0<df["sum"]][['sine']].std()
```

```
0.5297967209946162
```

## Visualizing the contents of a data frame In addition to all other plotting options, Pandas supports some basic plotting functionality

```
fig,ax=plt.subplots(1,1,figsize=(4,4))  
df.plot(ax=ax);
```



### 10.7.1 Selective plotting

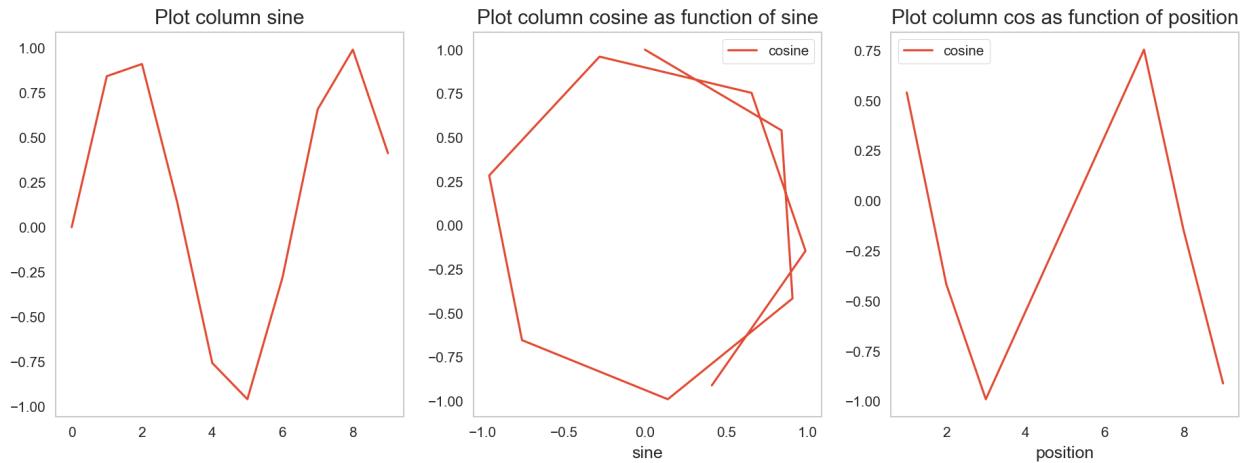
We mostly don't want to plot all columns at once

```
fig,ax=plt.subplots(1,3,figsize=(15,5))

df['sine'].plot(ax=ax[0]);
ax[0].set_title("Plot column sine");

df.plot(x='sine',y='cosine',ax=ax[1]);
ax[1].set_title("Plot column cosine as function of sine")

df[0<df['sine']].plot(x='position',y='cosine',ax=ax[2]);
ax[2].set_title("Plot column cos as function of position");
```

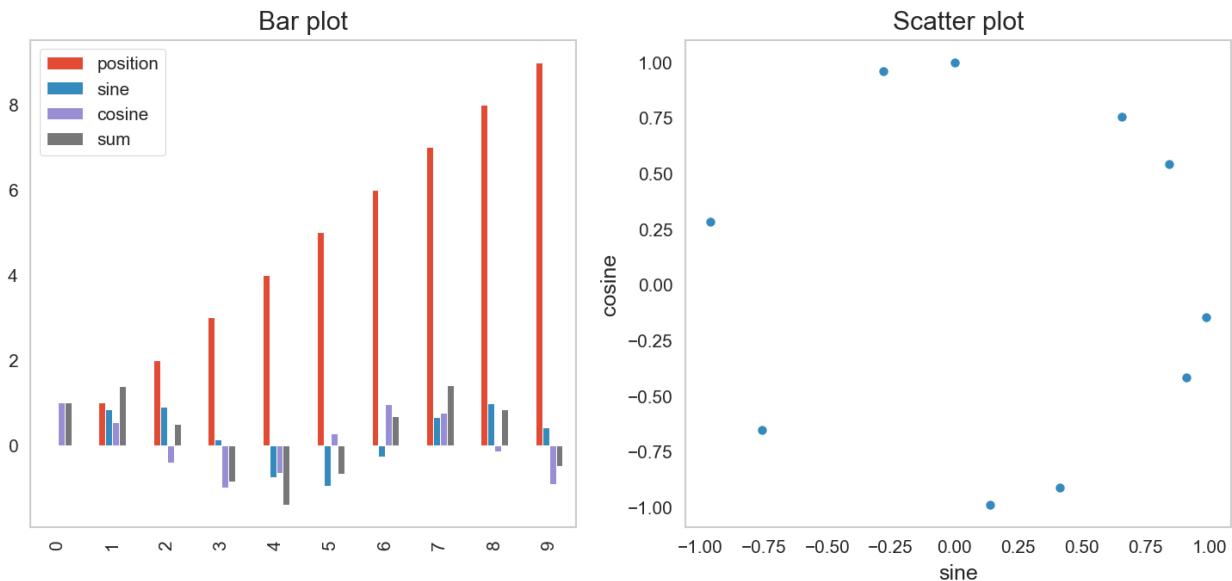


### 10.7.2 Different plotting styles

```
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(12,5))

df.plot(kind='bar',ax=ax1);
ax1.set_title('Bar plot');

df.plot(kind='scatter',x='sine',y='cosine',ax=ax2);
ax2.set_title('Scatter plot');
```



Further plotting options can be found on [pandas visualization documentation](#)

## 10.8 Set operations with data frames

In our work we may produce several data frames that needs to be merged:

- Image features
- Meta data
- Sensor logs
- etc.

[Merging frames topic in Pandas documentation](#)

### 10.8.1 Concatenating frames

#### Add more rows with the same categories

```
dfa = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']})

dfb = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']})

frames = [dfa, dfb]
result = pd.concat(frames)
result
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7

```
dfa = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                   index=[0, 1, 2, 3])

dfb = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                   index=[4, 5, 6, 7])

frames = [dfa, dfb]
result = pd.concat(frames)
result
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

## 10.8.2 Merging data frames

Add columns with new categories, at least one column in common.

```
dfa = pd.DataFrame({'id' : [1,2,3,4],
                    'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                   index=[0, 1, 2, 3])

dfb = pd.DataFrame({'id' : [1,2,3,4],
                    'X': ['B4', 'B5', 'B6', 'B7'],
                    'Y': ['C4', 'C5', 'C6', 'C7'],
                    'Z': ['D4', 'D5', 'D6', 'D7']},
                   index=[1, 2, 3, 4])
```

(continues on next page)

(continued from previous page)

```
result=pd.merge(dFA,dfB)
result
```

	id	A	B	C	D	X	Y	Z
0	1	A0	B0	C0	D0	B4	C4	D4
1	2	A1	B1	C1	D1	B5	C5	D5
2	3	A2	B2	C2	D2	B6	C6	D6
3	4	A3	B3	C3	D3	B7	C7	D7

## 10.9 Create new data frame from selected columns

```
new = result[['A','C','D']]
new.head()
```

	A	C	D
0	A0	C0	D0
1	A1	C1	D1
2	A2	C2	D2
3	A3	C3	D3

## 10.10 When are pandas data frames useful?

- Pandas is useful for *large* data - It must fit in the local memory.
- Really big data needs other options, e.g. dask data frames

### 10.10.1 How about big quantitative imaging?

Images can be stored in the data frame, but think twice!

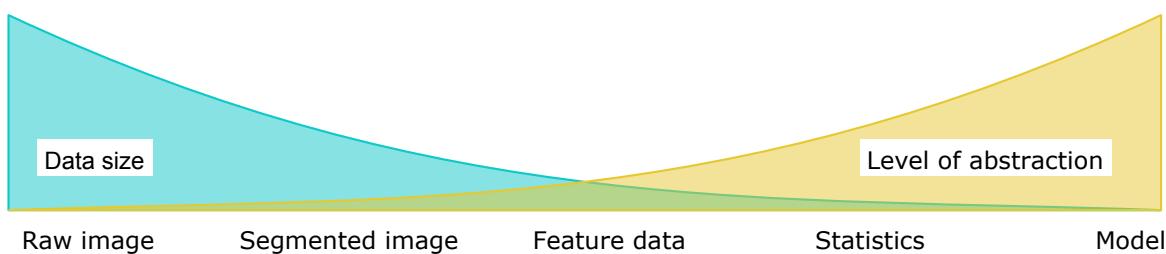


Fig. 10.2: The data size decreases radically when we increase the level of abstraction.

## PRESENTING THE RESULTS - BRINGING OUT THE MESSAGE

In the end you will want to present your results



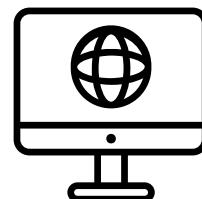
Discussions



Presentation



Publication



Web page

Fig. 11.1: Different ways to present your data.

### 11.1 Visualization

One of the biggest problems with *big* sciences is trying to visualize a lot of heterogeneous data.

- Tables are difficult to interpret
- 3D Visualizations are very difficult to compare visually
- Contradictory necessity of simple single value results and all of the data to look for trends and find problems

## Purpose of the visualization

You visualize your data for different reasons:

#### 11.1.1 Understanding and exploration

- Small and known audience (you and colleagues)
- High degree of understanding of specific topic.

### 11.1.2 Presenting your results

- Wider and sometimes unknown audience (reader of paper, person listening to presentation)
- At best general understanding of the topic.

from Knaflic 2015

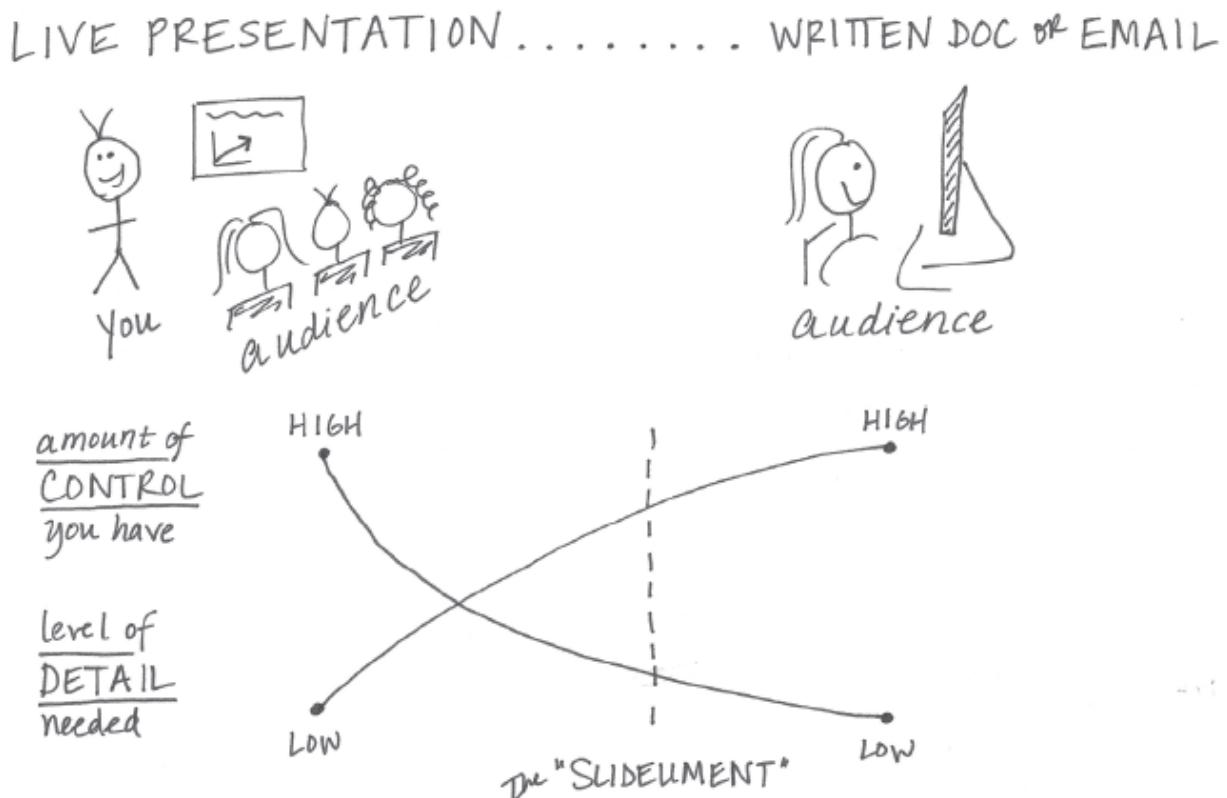


Fig. 11.2: The level of detail in a presentation depends on the medium it is presented Knaflic 2015.

## 11.2 Bad Graphs

There are too many graphs which say:

- *my data is very complicated*
- *I know how to use \_\_ toolbox in Python/Matlab/R/Mathematica*
- Most programs by default make poor plots
- Good visualizations takes time to produce

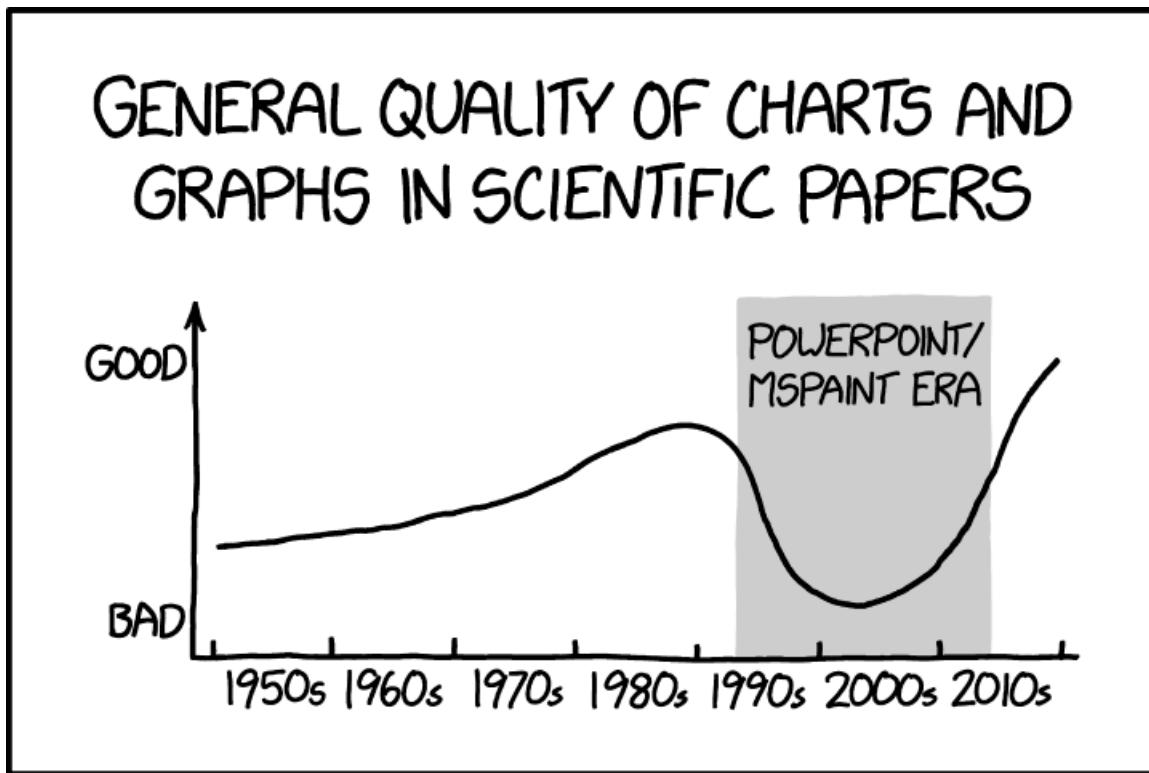


Fig. 11.3: This cartoon from XKCD highlights a problem with the access to software making it too easy to produce a graph or illustration.

### 11.2.1 Some bad examples

There are plenty examples on how you shouldn't present your data. The problem is in general that there is way too much information that needs to be pre digested before it is ready to any audience.

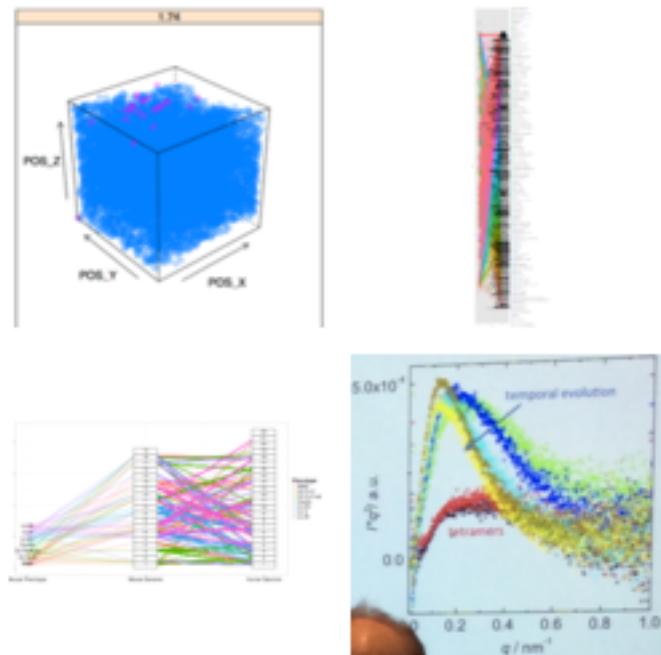


Fig. 11.4: Four examples of how *not* to present your data.

### 11.3 How to improve - Key Ideas

There is a need to consciously prepare your figures to bring your message to the audience in an understandable way. The first step is to ask yourself the following questions.

1. What is my message?
2. Does the graphic communicate it clearly?
3. Is a graphic representation really necessary?
  - Does every line / color serve a purpose?
  - Pretend ink is very expensive

Keep this in mind every time you create a figure and you will notice that after while you will have a tool set that makes it easier and faster to produce well thought figures that clearly brings out your message to your audience.

Personally, I always write scripts to produce each plot of a publication. This makes it easier to revise the manuscript in a reproducible and efficient manner. The first implementation may take longer, but the revision is done in no time.

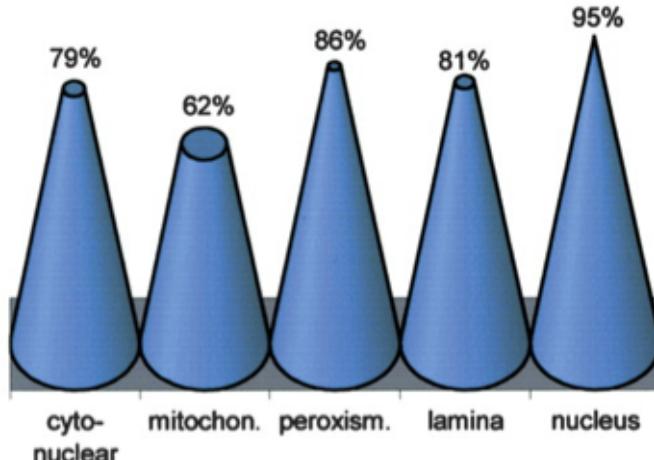
### 11.3.1 Some literature

If you want to read more about how to work with data visualization. I can recommend these:

- Knaflic, Storytelling with Data: A Data Visualization Guide for Business Professionals, 2015
- Few, Should data visualization always be beautiful?, 2012

### 11.3.2 Simple Rules

1. Never use 3D graphics when it can be avoided (unless you want to be deliberately misleading)



2. Pie charts can also be hard to interpret
3. Background color should almost always be white (not light gray)
4. Use color palettes adapted to human visual sensitivity
5. Use colors and transparency smart

## 11.4 Grammar of Graphics

### 11.4.1 What is a grammar?

- Set of rules for constructing and validating a sentence
- Specifies the relationship and order between the words constituting the sentence

### 11.4.2 How does grammar apply to graphics?

If we develop a consistent way of

- expressing graphics (sentences)
- in terms of elements (words) we can compose and decompose graphics easily

The most important modern work in graphical grammars is “The Grammar of Graphics” by Wilkinson, Anand, and Grossman (2005).

This work built on earlier work by Bertin (1983) and proposed a grammar that can be used to describe and construct a wide range of statistical graphics.

### **11.4.3 Grammar Explained**

Normally we think of plots in terms of some sort of data which is fed into a plot command that produces a picture

- In Excel you select a range and plot-type and click “Make”
  - In Matlab you run `plot (xdata, ydata, color/shape)`
1. These produces entire graphics (sentences) or at least phrases in one go and thus abstract away from the idea of grammar.
  2. If you spoke by finding entire sentences in a book it would be very ineffective, it is much better to build up word by word

### **11.4.4 Grammar**

Separate the graph into its component parts

Construct graphics by focusing on each portion independently.

### **11.4.5 Figure decorations**

Besides the data you also need to provide annotating items to the visualization.

It may seem unnecessary to list these annotations, but it happens too often that they are missing. This leaves the observers wondering about what they see in the figure. It is true that it takes a little more time to add annotation to your figure. Sometimes, you may think that the plot is only for your own understanding and you don't need to waste the time on making it complete. Still, in the next moment it finds its way to the presentation and then all of a sudden it is official...

Annotations are fundamental features of figures and available in any plotting library. In some cases you have to look a little longer to find them or write a little more code to use them, but they are there.

#### **Plots**

- Curve legend - telling what each curve represents.
- Axis labels - telling what information you see on each axis.
- Figure title - if you use multiple plots in the same figure.

#### **Images**

- Color bar - to tell how the colors are mapped to the values.
- Scale bar - to tell the size of the object in the image.

### 11.4.6 Color maps revisited

Choosing the right color is a science.

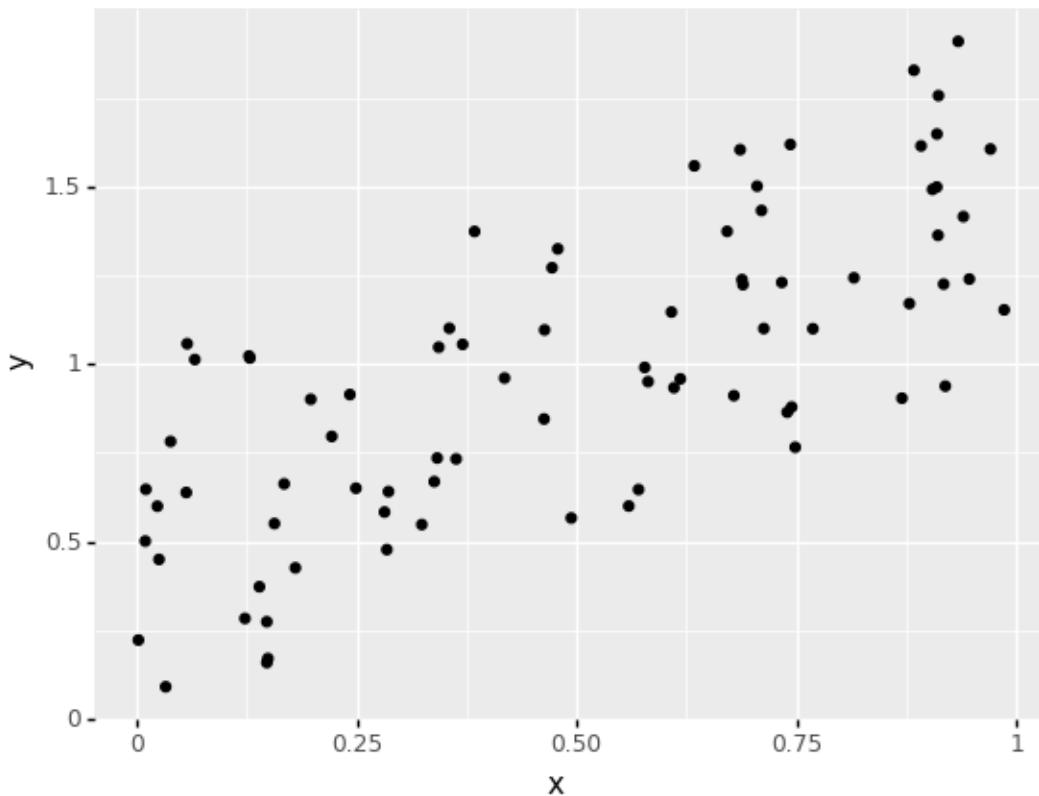
Crameri, F., et al. (2020)

## 11.5 What is my message

Plots to “show the results” or “get a feeling” are usually not good

```
from plotnine import *
from plotnine.data import *
import pandas as pd
# Some data
xd = np.random.rand(80)
yd = xd + np.random.rand(80)
zd = np.random.rand(80)

df = pd.DataFrame(dict(x=xd, y=yd, z=zd))
ggplot(df, aes(x='x', y='y')) + geom_point()
```



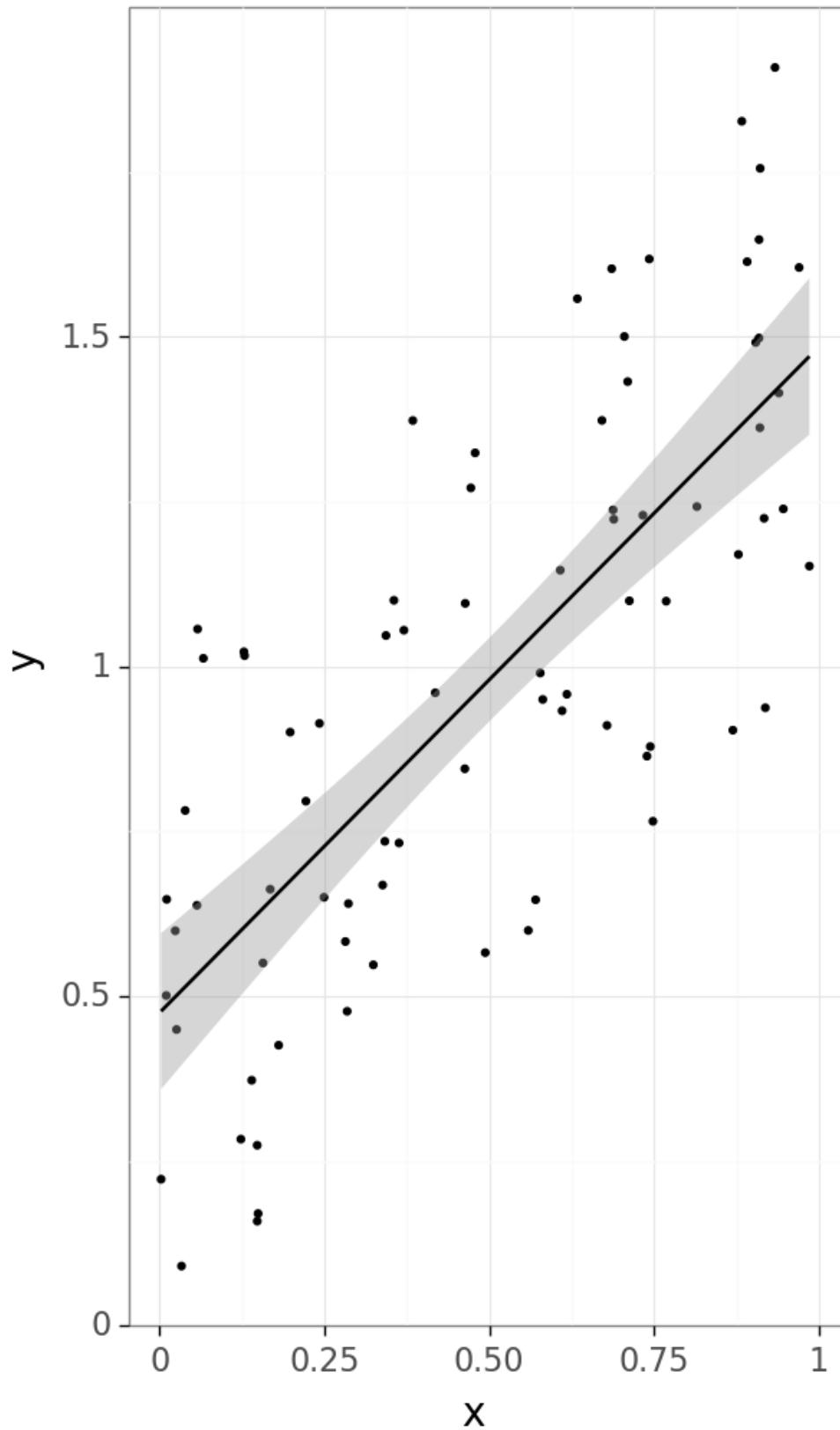
```
<ggplot: (8784921895476)>
```

### 11.5.1 Focus on a single, simple message

“X is a little bit correlated with Y”

```
(ggplot(df, aes(x='x', y='y'))  
+ geom_point()  
+ geom_smooth(method="lm")  
+ coord_equal()  
+ labs(title="X is weakly correlated with Y")  
+ theme_bw(20) )
```

X is weakly correlated with Y



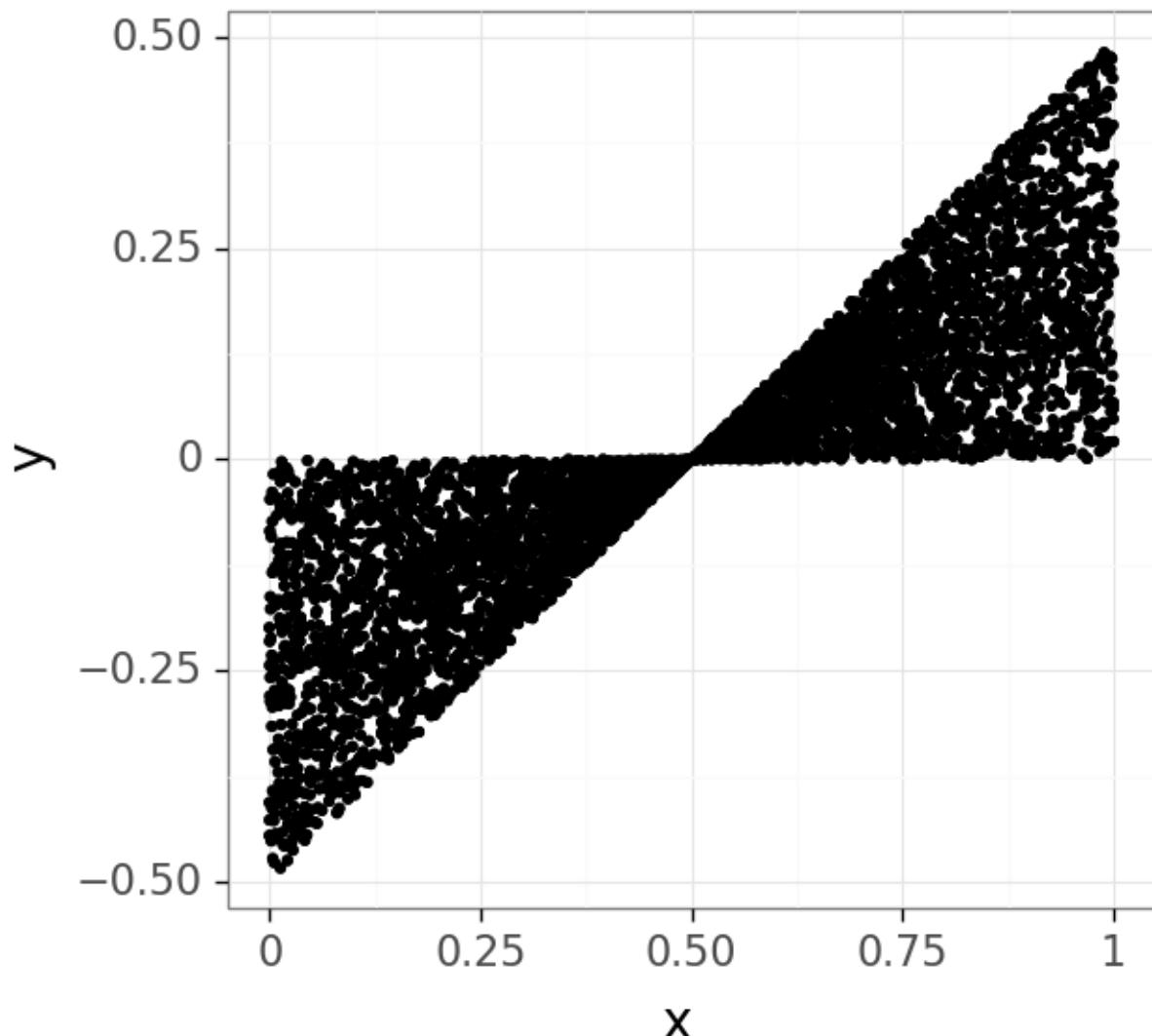
```
<ggplot: (8784921911818)>
```

### 11.5.2 Does my graphic communicate it clearly?

Too much data makes it very difficult to derive a clear message

```
xd = np.random.rand(5000)
yd = (xd-0.5)*np.random.rand(5000)

df = pd.DataFrame(dict(x=xd,y=yd))
(ggplot(df,aes(x='x',y='y'))
 + geom_point()
 + geom_point()
 + coord_equal()
 + theme_bw(20))
```

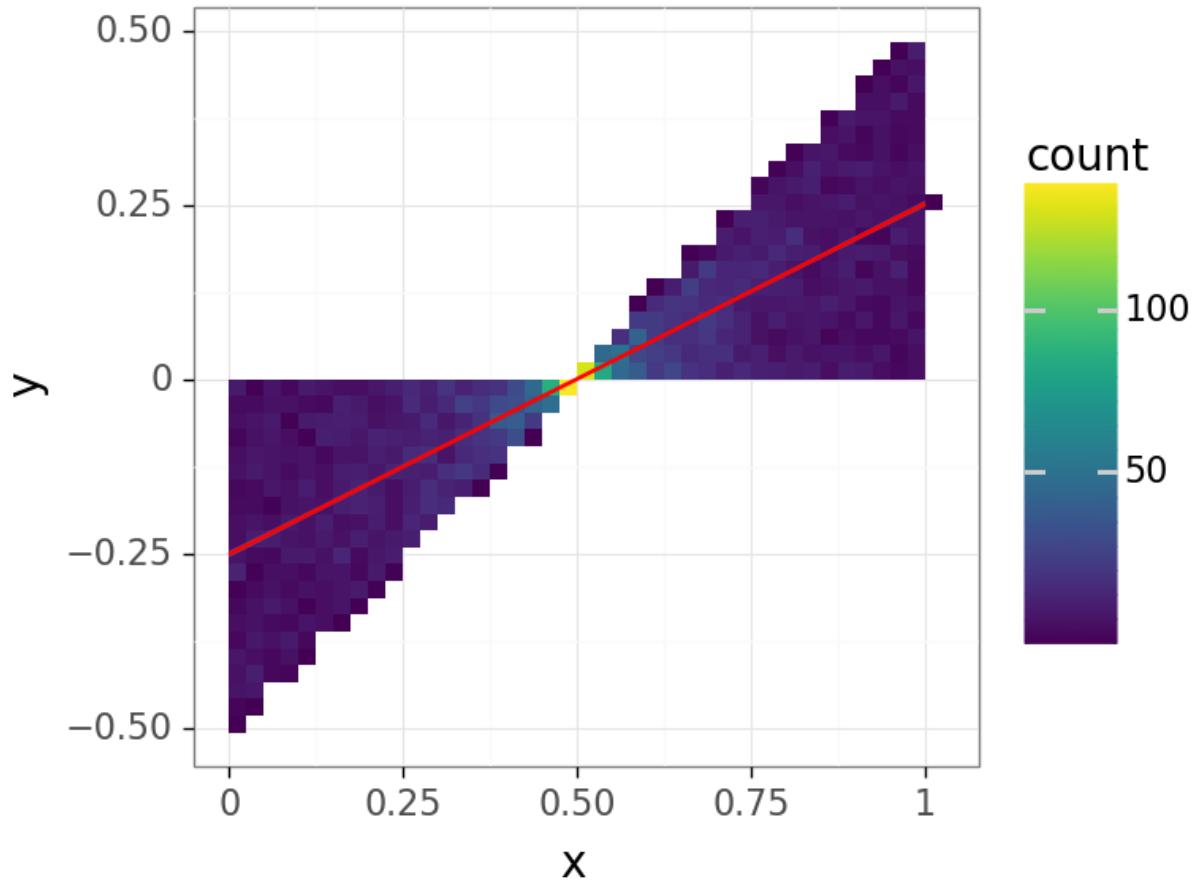


```
<ggplot: (8784921417521)>
```

### 11.5.3 Reduce the data

Filter and reduce information until it is extremely simple

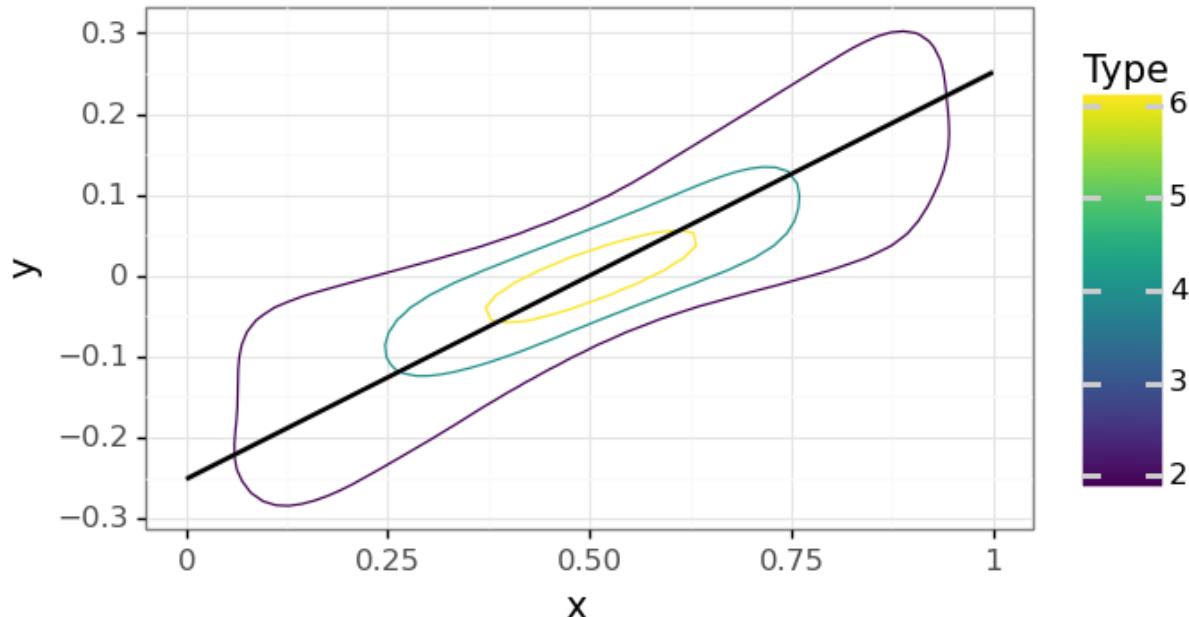
```
(ggplot(df, aes(x='x', y='y'))
+ stat_bin2d(bins=40)
+ geom_smooth(method="lm", color='red')
+ coord_equal()
+ theme_bw(20)
+ guides(color='F')
)
```



```
<ggplot: (8784887537210)>
```

### 11.5.4 Reduce even further

```
(ggplot(df, aes(x='x', y='y'))
+ geom_density_2d(aes(x='x', y='y', color='..level..'))
+ geom_smooth(method="lm")
+ coord_equal()
+ labs(color="Type")
+ theme_bw(15)
)
```



```
<ggplot: (8784887931697)>
```

## 11.6 Common visualization packages for python

- Matplotlib [Matplotlib 3.0 Cookbook](#) or ETHZ lib, code examples
- Plotly
- Seaborn
- ggplot R using the [ggplot2 library](#), which is ported to python.

A short summary of these packages can be found [here](#).

---

CHAPTER  
**TWELVE**

---

**SUMMARY**

## 12.1 Statistics

- I am not a statistician and is not a statistics course
- If you have questions or concerns
- Both ETHZ and Uni Zurich offer **free consultation** with real statisticians
- They are rarely bearers of good news - you allways need more data...
- Simulations (even simple ones) are very helpful
- Try and understand the tests you are performing

## 12.2 Data frames

- A tool for handling the mess of data storage and analysis
- A local database that allows filtering, arithmetics, plotting, etc.

## 12.3 Visualization

- Visualization is the crowning piece of your investigation - make it count!
- Many toolboxes can be used, choose the one that fits your needs.



---

CHAPTER  
**THIRTEEN**

---

## **OLD SLIDES TO TRANSFER FROM R TO PYTHON**

old lecture 2017 old lecture 2019