Quantitative Big Imaging - Statistics

Anders Kaestner

Apr 13, 2022

CONTENTS

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

CHAPTER

ONE

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
import matplotlib.patches as patches
%matplotlib inline

plt.rcParams["figure.figsize"] = (8, 8)
plt.rcParams["figure.dpi"] = 100
plt.rcParams["font.size"] = 12
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*, **Chapter 3**
- John C. Russ, *The Image Processing Handbook*,(Boca Raton, CRC Press)
- Gregory J. Privitera, *Statistics for the Behavioral Sciences* **Chapter 8**
- Leland and Wilkinson, *Grammar of Graphics*

1.1.2 Papers / Sites

- 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 with John Oliver: Scientific Studies
- Credibility Crisis

1.1.4 Further material

Slides

- How to solve NLP problems
- Kieran Healy, Data Visualization - A practical introduction
- P-Values with Puppies

Model Evaluation

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

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 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

1.4 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*
  - The three V’s: __v__olume, __v__elocity, __v__ariety
  - scalable, fast, easy to customize

So what is “big” imaging?

1.4.1 Doing analyses in a disciplined manner

- fixed, well-defined, steps
- easy to regenerate results
- no *magic*
- documentation

1.4.2 Advantages of a diciplined workflow

Having everything automated

- 100 samples is as easy as 1 sample
- Some intitial extra effort pays off

Being able to adapt and reuse analyses

- one really well working script
- modify parameters to address e.g.
  - different types of cells
  - different regions

1.5 Objectives

1. Scientific Studies all try to get to a single number
- Make sure this number is describing the structure well (earlier lectures)
- 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?

1.6 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 (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
- and almost as large number of parameters to *tune*

How do we do something meaningful with it?

CHAPTER

TWO

CORRELATION AND CAUSATION

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

Correlation

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

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.

Observational

Controlled

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

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 baking time and bubble size

2.1.2 Controlled

The controlled experiments are designed to explore specific aspects of a population or phenomenon. To achieve this, you want to introduce differences between different groups and keep one as reference. The reference group can be the unmodified samples or samples prepared with an wellknown process.

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

STATISTICAL EXPERIMENTS - 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.

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 assumtions, in particular not the second one.

A different question is about collections of random variables:

What 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 Real experiment

1. Each measurement tells us about:
    - our sample,
    - our instrument,
    - *and* our analysis
2. 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.

CHAPTER

FOUR

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.

It covers:

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

4.1 Let's load the Iris Dataset

Fisher, The Use of Multiple Measurements in Taxonomic Problems, 1936

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)
```

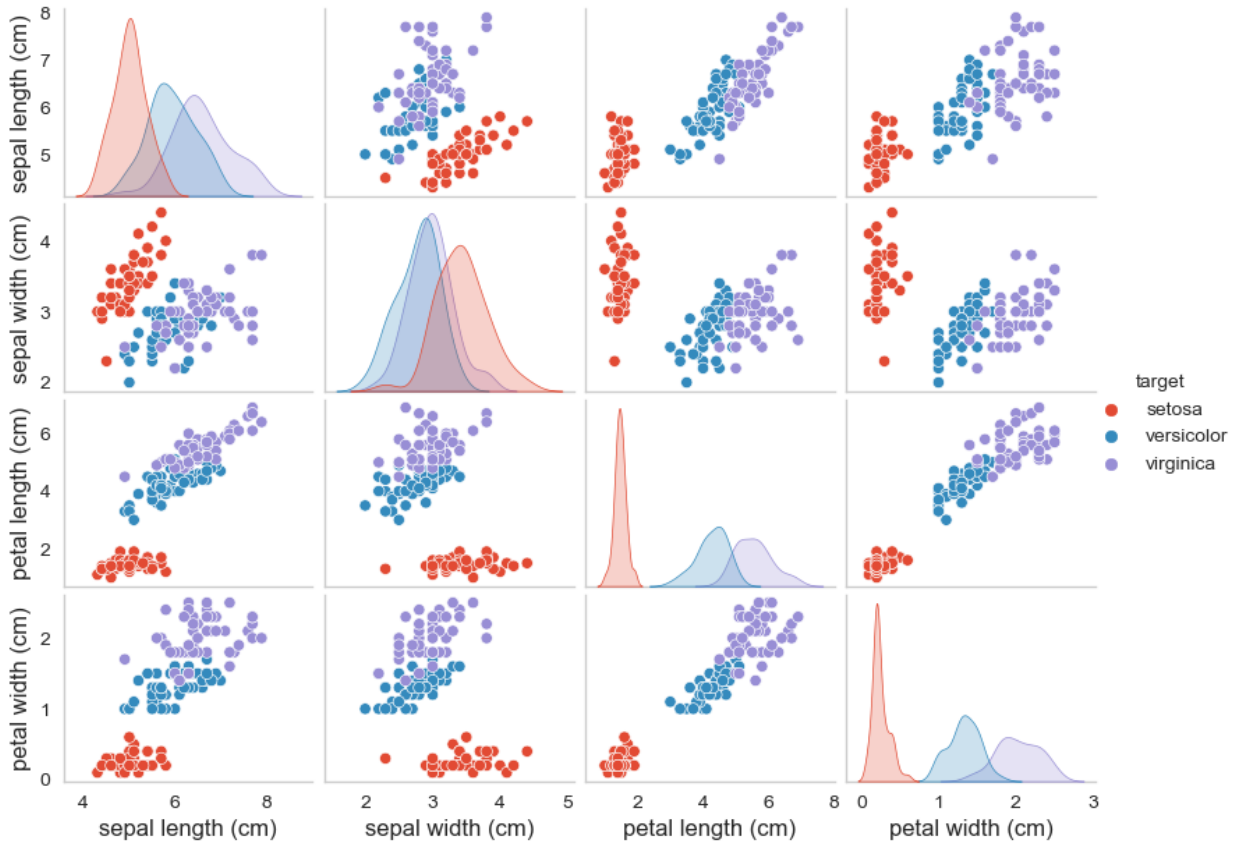
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
113                5.7               2.5                5.0               2.0
112                6.8               3.0                5.5               2.1
121                5.6               2.8                4.9               2.0
87                 6.3               2.3                4.4               1.3
139                6.9               3.1                5.4               2.1

         target
113   virginica
112   virginica
121   virginica
87   versicolor
139   virginica
```

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.

```
p=sns.pairplot(iris_df, hue='target');
plt.gcf().set_size_inches(9, 6)
```



In the plot, we clearly see that one species (setosa) in general has other flower leaf 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 Group similarity

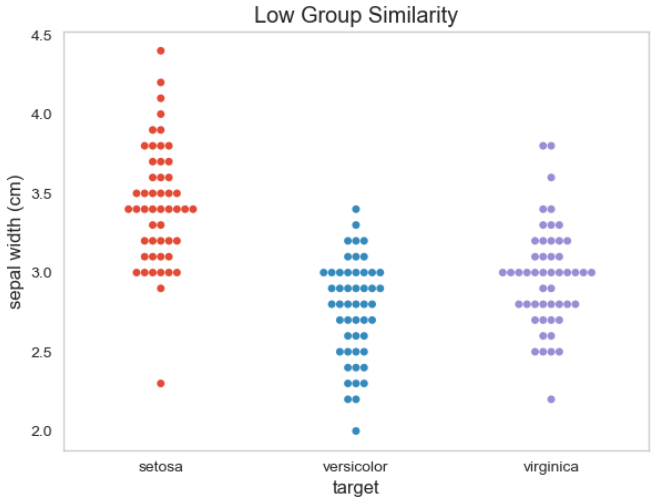
How well are groups separated in a study

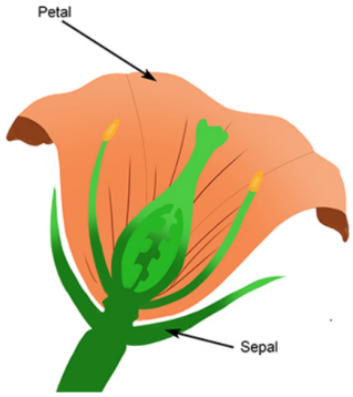
- Low group similarity - overlapping histograms, harder to separate
- High group similarity - separated histograms, easier to separate

4.3.2 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');
```

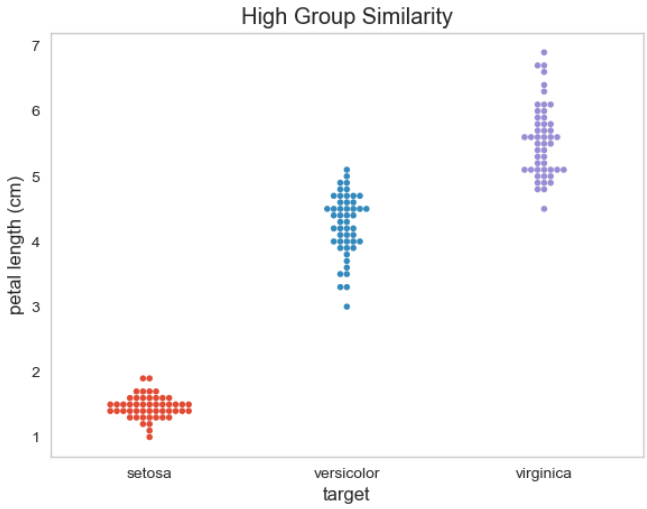


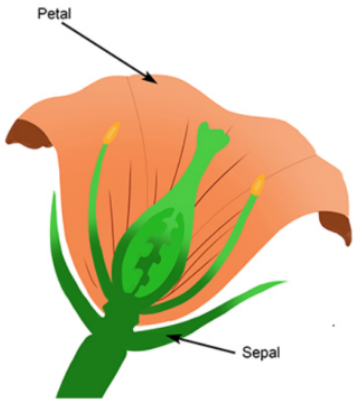


4.3.3 Petal length

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

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

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

When compute the ICC for sepal width and petal length, we see that the ICC confirms our first qualitative assessment about the group similiarity.

```
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()
    print('{0}: S_w={1:0.02f}, S_a={2:0.2f}'.format(value_name,S_w,S_a))
    return S_a/(S_a+S_w)
```

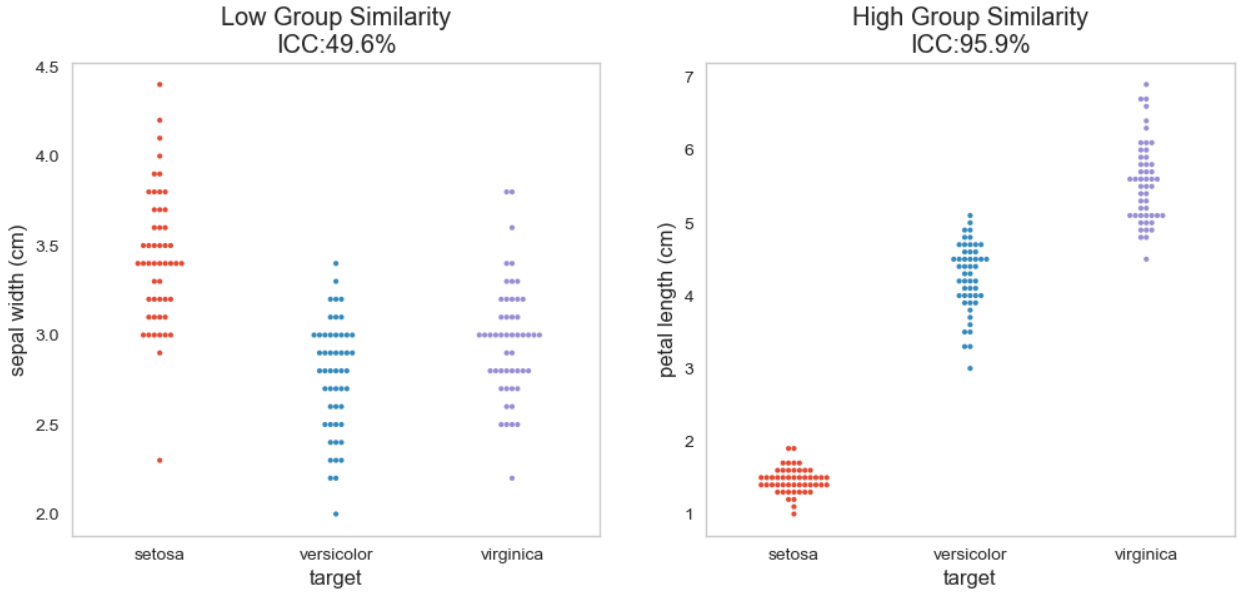
(continues on next page)

(continued from previous page)

```
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)
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)));
```

```
sepal width (cm): S_w=0.12, S_a=0.11
petal length (cm): S_w=0.19, S_a=4.37
```



CHAPTER

FIVE

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 $\mathcal{H}_0$
    - Usually that both groups are the same
3. 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 Outcomes for decision making

With error probabilites:

- α - probability of Type I errors / significance level
- β - probability of Type II errors

From Privitera 2017

5.2 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 ($\mathcal{H}_0$): 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?

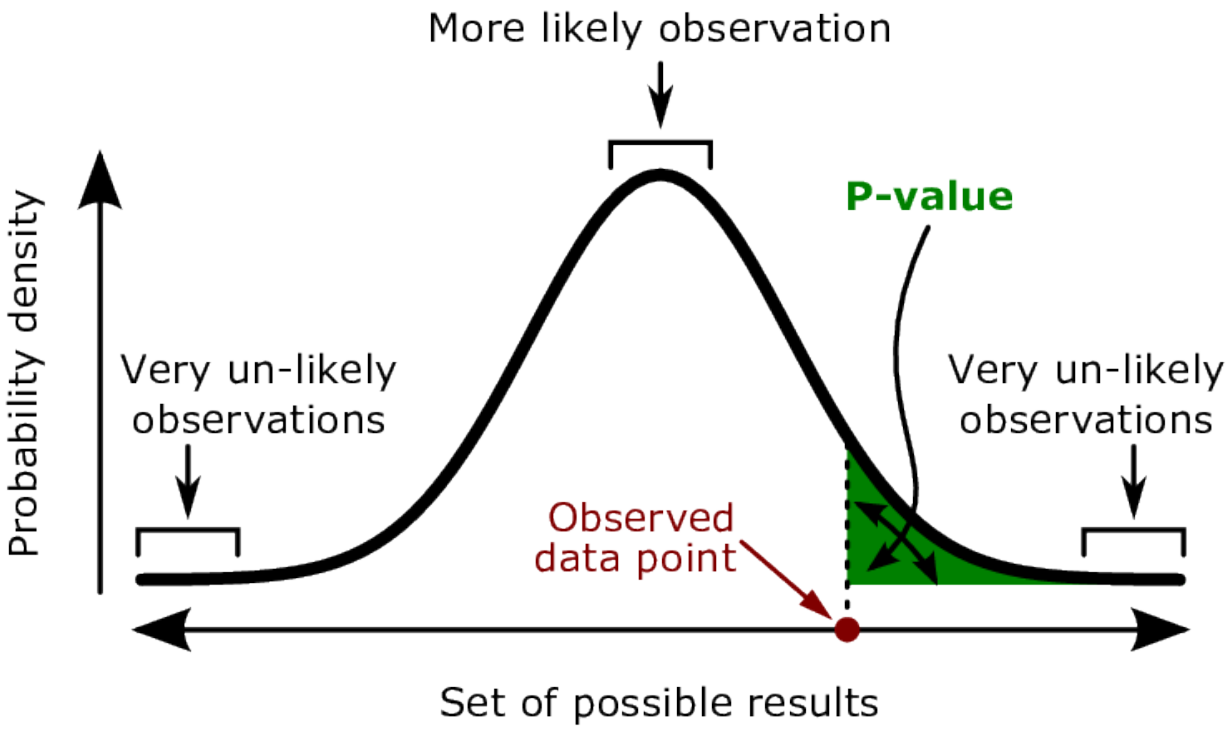


Fig. 5.1: Explaining p-value with the normal distribution.

5.3 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

5.3.1 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

5.3.2 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, $\mathcal{H}_0$)

- Assume unbiased observations
- Assume normal distribution

CHAPTER

SIX

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.
- Null-hypothesis ($\mathcal{H}_0$): the coin is fair
- 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        1                  50.0%
1        4                   6.2%
2        5                   3.1%
```

6.2 Probability with many experiments

Let N friends make 5 tosses…

$$P = \overbrace{1 - (\underbrace{1 - 0.5^{N_{Tosses}}}_{\text{Get 5 heads}})^{N_{Friends}}}^{\text{Not getting 5 heads}}$$

```
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          1                  3.1%
1         10                 27.2%
2         20                 47.0%
3         40                 71.9%
4         80                 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

6.3.1 Let's look at multiple observations

We make five random variables with ten observations of

$$var\_i \in \mathcal{U}(-1, 1)$$

and make two groups '1' and '2'

```
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(
```

(continues on next page)

(continued from previous page)

```
        np.random.uniform(-1, 1, size=(rows, cols)),
        columns=['Var_{:02d}'.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
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

6.3.2 Compute p-values for the table

The Student-t test is computed using the python function

```
scipy.stats import ttest_ind

ttest_ind(var_i[Group==1],var_i[Group==2])
```

This is a two-sided test for the null hypothesis that two independent samples have identical average (expected) values. This test assumes that the populations have identical variances by default.

Here, we compute the p-values for the table we just created. The variables with p-values less than 0.05 are marked with yellow.

In the following example we compare the two groups of each random variable to determine if they are significantly different.

Essentially $ttest\_ind(var_i[Group == 1], var_i[Group == 2])$

We expect the two parts to be the same as all values are generated using the same random generator.

```
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}})
```

(continues on next page)

(continued from previous page)

```
show_significant(all_ttest(rand_df))
```

```
<pandas.io.formats.style.Styler at 0x15d8d6160>
```

6.3.3 A larger table

Now, let's create a larger table with 150 rows and 20 independent variables.

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

```
<pandas.io.formats.style.Styler at 0x15fd923a0>
```

6.3.4 Repeating the measurements

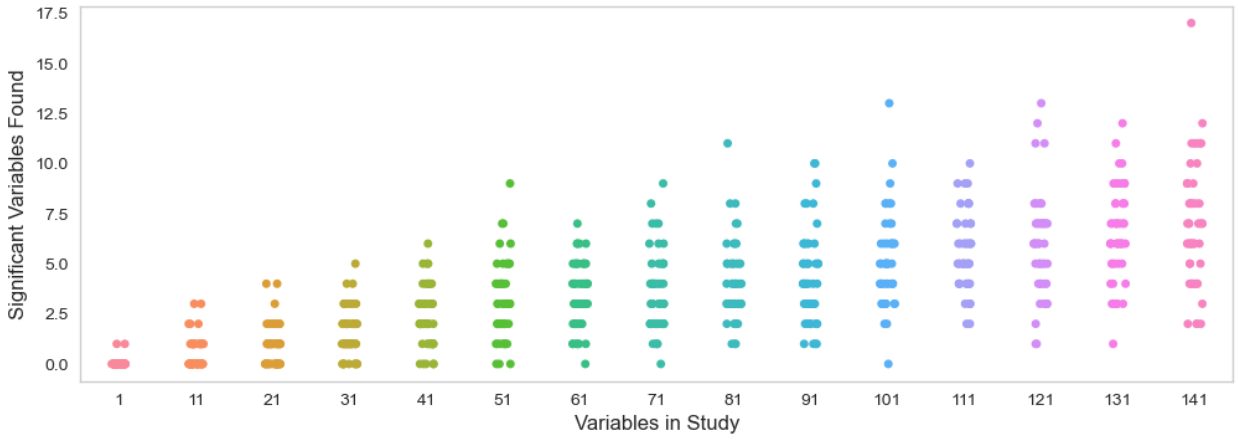
We saw with the coin tossing that the probability to detect the event we are looking for increased with the number of repeated independent measurements (friends tossing coins).

Let's see what happens when we do the the same with our table of experiments. First, we must generate the data. We will try using tables with 1 to 150 variable and 100 observations. Each measurement will be repeated 50 times.

```
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)
```

```
  0%|          | 0/15 [00:00<?, ?it/s]
```

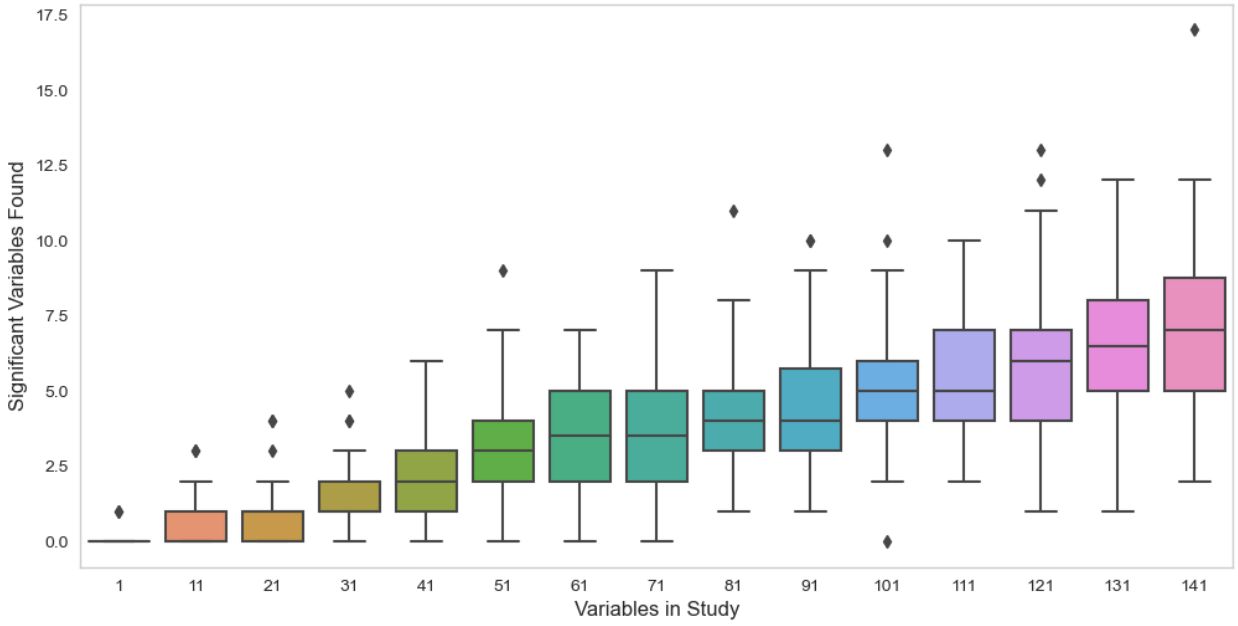
```
fig,ax = plt.subplots(1,1,figsize=(12,4))
sns.stripplot(data=var_found_df, x='Variables in Study', y='Significant Variables␣
↪Found');
```



6.3.5 Visualize the results differently

The strip plot we just used gets cluttered when we have too many observations. A different way to show the results is to use a boxplot.

```
plt.figure(figsize=(12,6))
sns.boxplot(data=var_found_df,
            x='Variables in Study', y='Significant Variables Found');
```



6.4 Multiple Testing Bias: Correction

We saw that increasing the number of tests also increases the probability of detection. This is missleading and needs to be corrected.

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

This comes from the familywise error $$\bar{\alpha} = 1 - \left(1 - \alpha_{\text{per comparison}}\right)^{m}$$

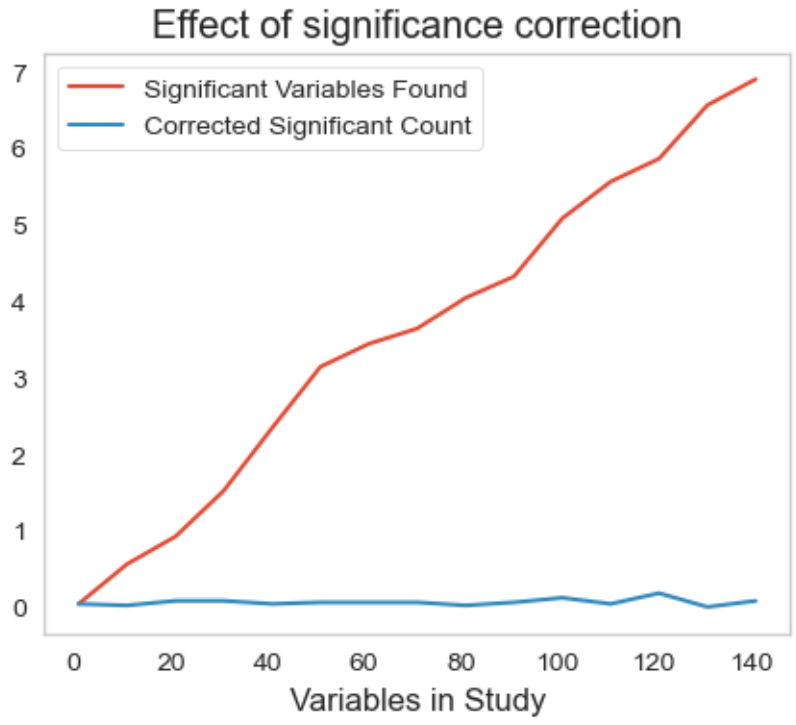
where m is the number of hypotheses tested. Then, with Boole's inequality we have

$$\bar{\alpha} \leq m \cdot \alpha_{\text{per comparison}}$$

Which leads to the Boniferroni correction.

```
fig,ax=plt.subplots(1,figsize=(5,4))
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'
],ax=ax);
plt.title('Effect of significance correction');
```



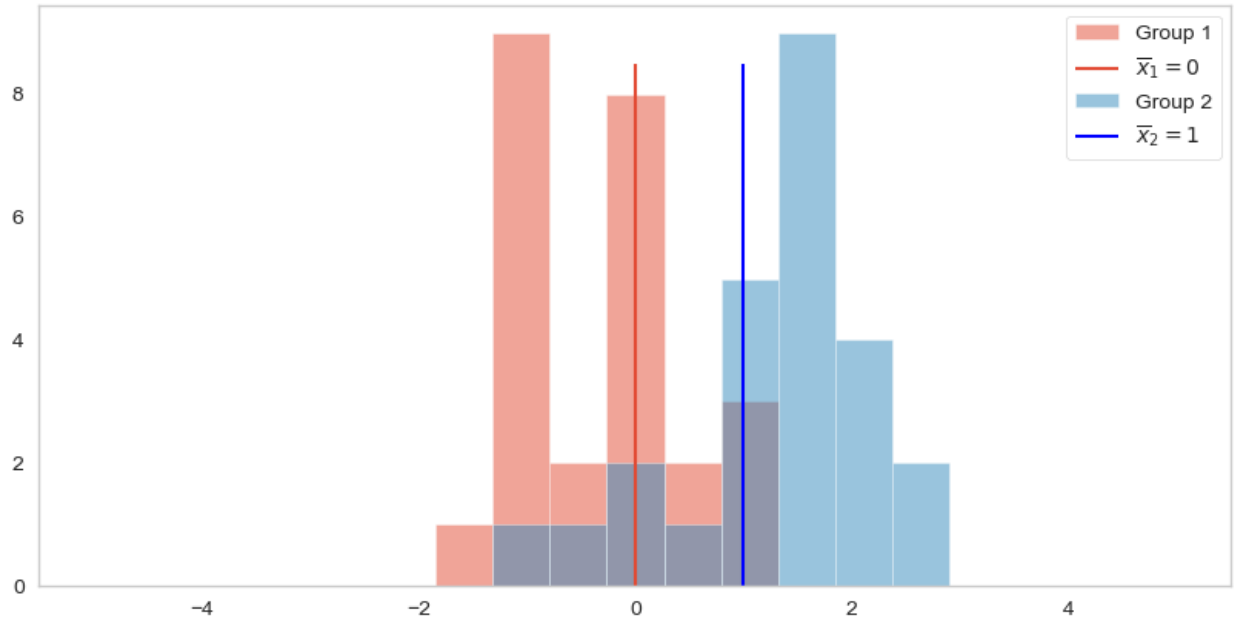
6.4.1 It this correction factor sufficient?

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))
ax1.hist(table_df.query('Group==1')['Really Different Var'], np.linspace(-5, 5, 20),␣
 ↪label='Group 1', alpha=0.5);
ax1.vlines(0,ymin=0,ymax=8.5,label='$\overline{x}_1=0$')
ax1.hist(table_df.query('Group==2')['Really Different Var'], np.linspace(-5, 5, 20),␣
 ↪label='Group 2', alpha=0.5);
ax1.vlines(1,ymin=0,ymax=8.5,color='blue',label='$\overline{x}_2=1$')
ax1.legend();
```



6.4.2 Run many tests

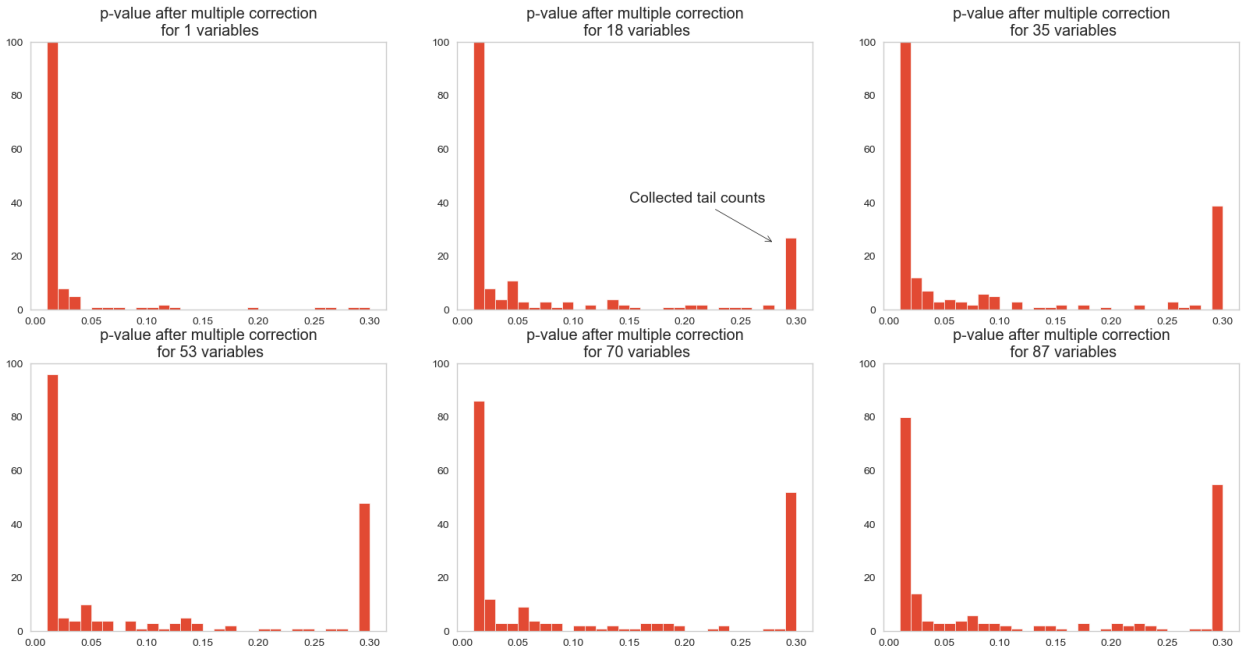
We run 200 tests with two variables with $\mathcal{N}(0,1)$ and $\mathcal{N}(1,1)$ and compute the p-values for each test.

```
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]
```

When we look at the histograms of p-values scale by the number of variables in the test we see that there is a greater probability to accept the null-hypothesis.

```
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_ylim(0, 100)
    c_ax.set_title('p-value after multiple correction\n for {} variables'.format(var_
 ↪count))


m_axs[0,1].annotate("Collected tail counts", xy=(0.28, 25), xytext=(0.15, 40),␣
 ↪arrowprops=dict(arrowstyle="->",color="black"),fontsize=14);
```

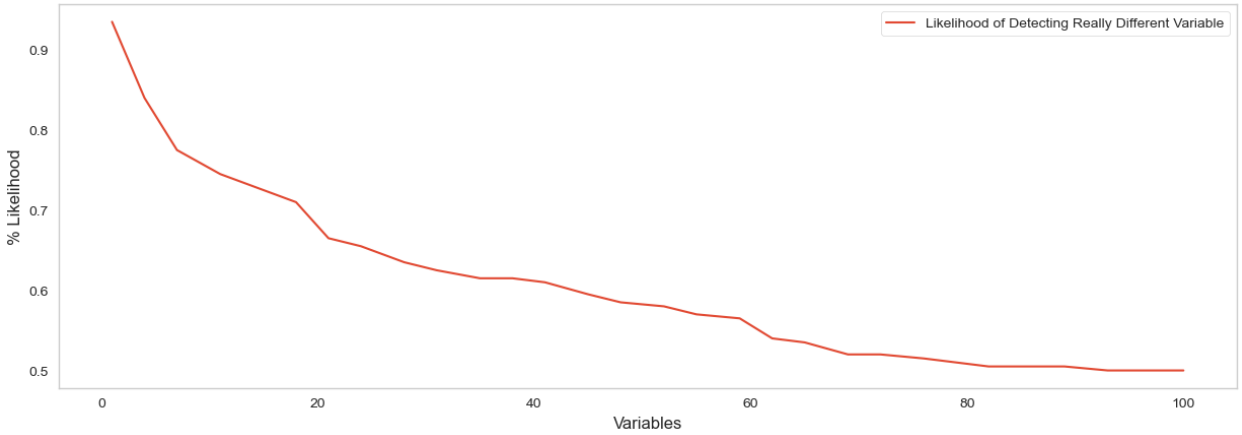


6.4.3 The likelihood to find a different variable

The scaling by the number of variables means that we are less likely to reject the null hypothesis. So, what is the likelihood?

We count the number of the p-values less than 0.05 to compute the likelihood of detecting a really different variable.

```
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');
```



Here, we see that the likelihood is very low for many varibles. A reason in that we are working on a limited number of samples and split these on an incresing number of varibles.

CHAPTER

SEVEN

PREDICTING AND VALIDATING - MAIN CATEGORIES

There are plenty machine-learning techniques available. Each one dedicated to a specific type of problem and data collection.

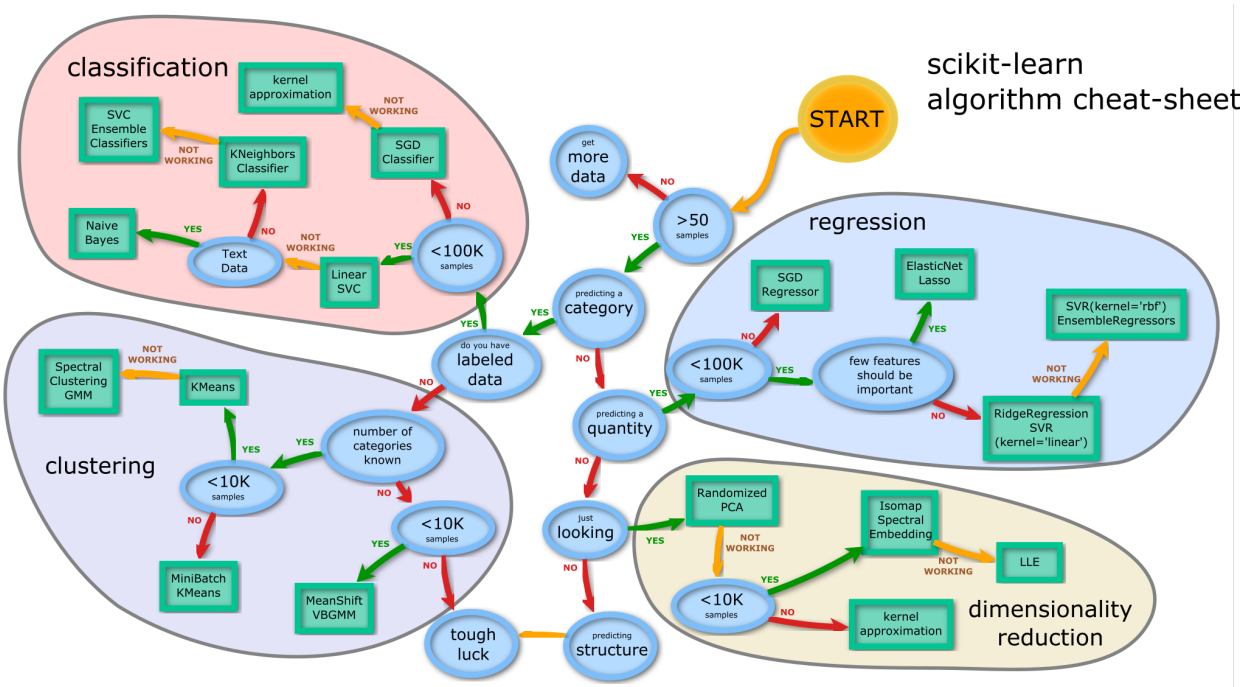


Fig. 7.1: A cheat sheet to identify the best machine learning technique for your problem.

`Fig <number> fig_mlcheatsheet` provides a guide to find the correct method 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

7.1 Overview

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

7.1.1 The input could be

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

7.1.2 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)

7.1.3 Overfitting

The most serious problem with machine learning and such approachs 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.

Training a model like:

```
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?!
```

This 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.

7.2 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**

7.3 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
```

CHAPTER

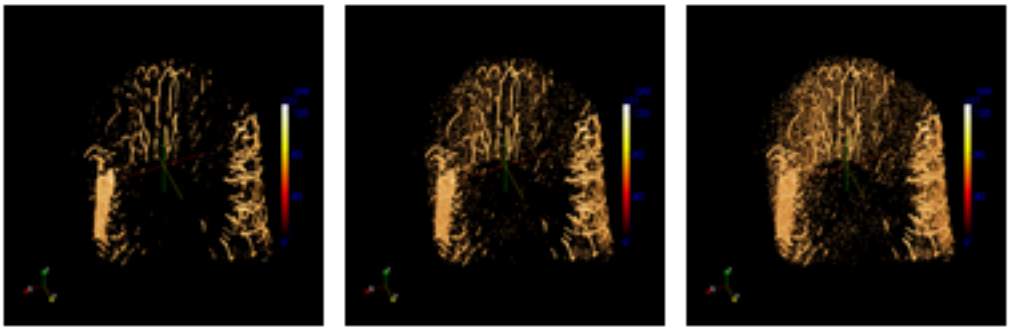
EIGHT

QUALITATIVE VS QUANTITATIVE

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

8.1 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?



8.2 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.

8.3 Parameters

We have a workflow to analyze *shape* and *thickness* of items in an image:

```
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 0x16898b250>
```

Three parameters can be controlled:

- Gaussian filter: σ and *neighborhood size*
- Threshold level

8.3.1 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.

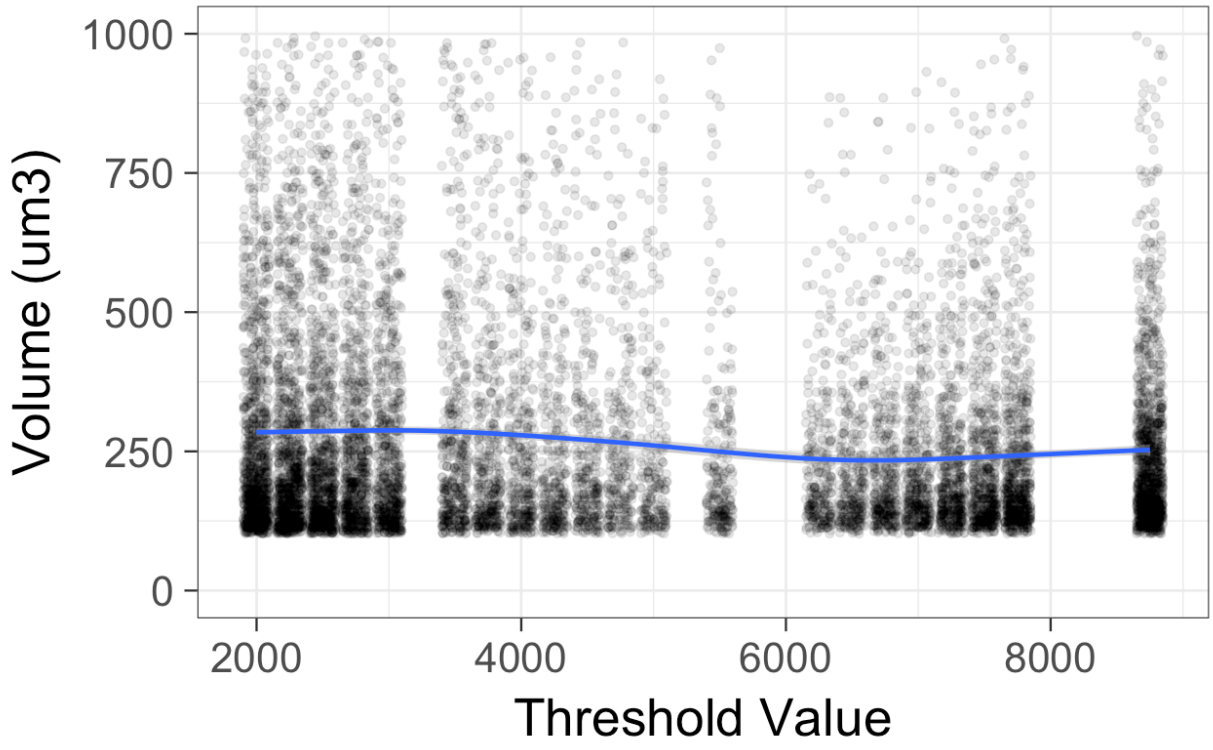


Fig. 8.1: Volume measurements for different thresholds

8.3.2 Is it always the same?

The jittered scatter plot in fig 8.1 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.

8.4 Sensitivity

8.4.1 Control system theory

Sensitivity is defined as

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

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

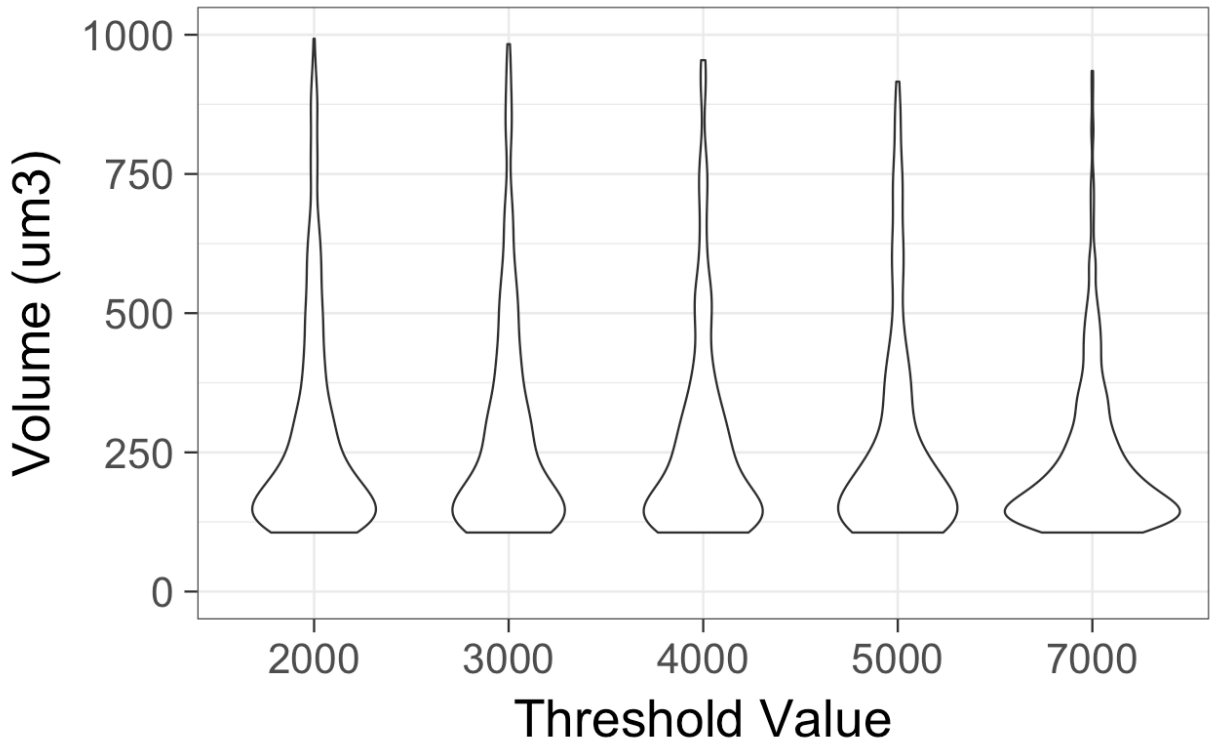


Fig. 8.2: A violin plot of the volume data.

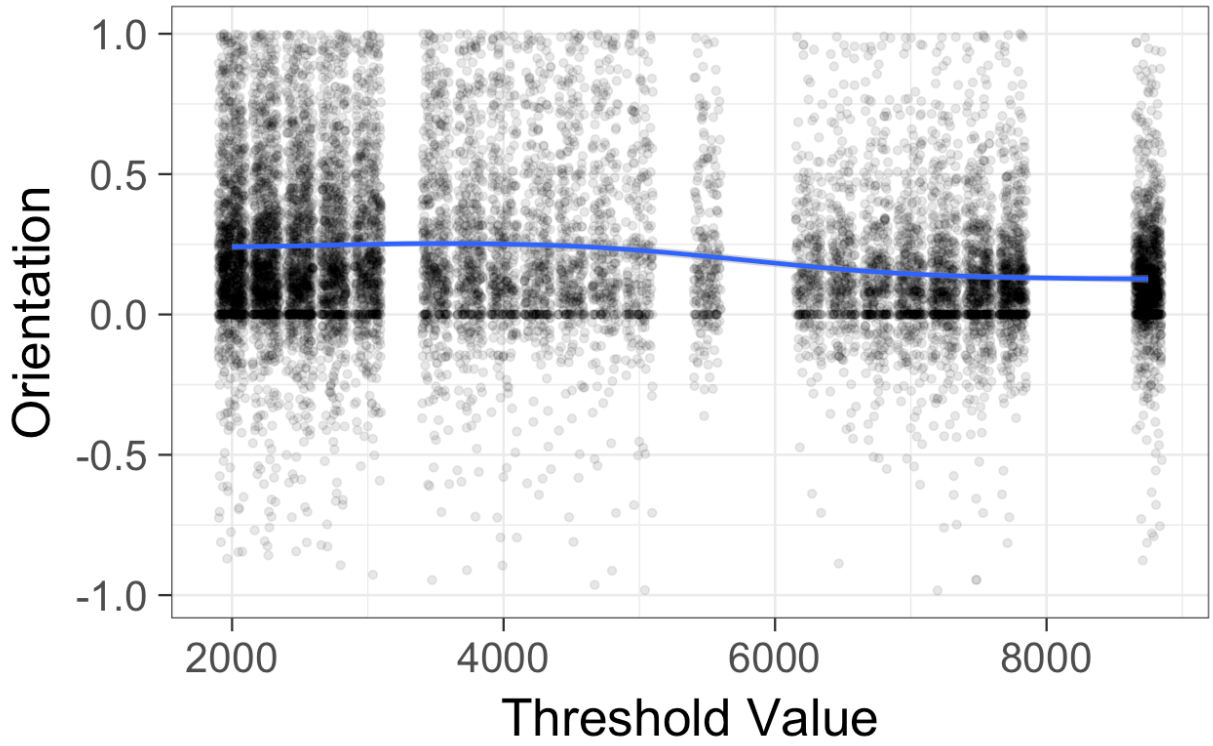


Fig. 8.3: Scatter plot of the orientation of the items.

8.4.2 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

$\rightarrow$ the sensitivity becomes volume per intensity!

8.4.3 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.

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

8.5 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

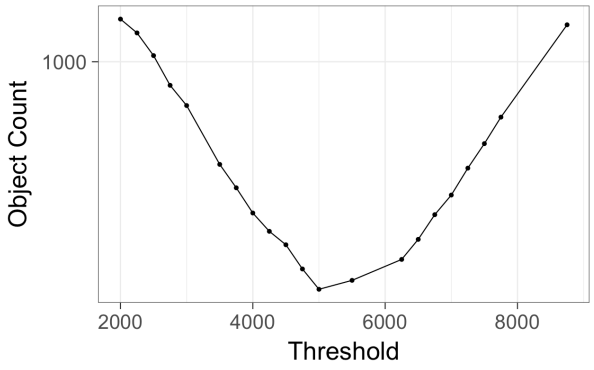


Fig. 8.4: Sensitivity measurement to measure how sensitive the object count is to the choice of the threshold.

8.6 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.

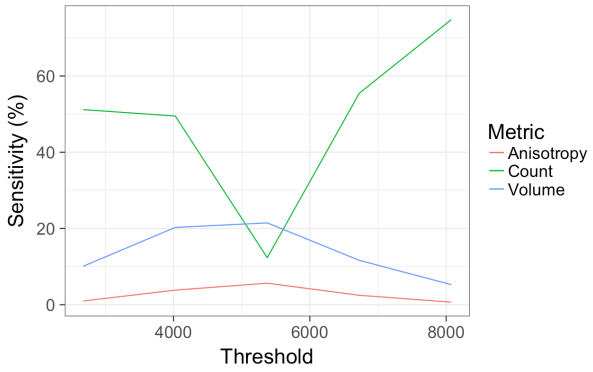


Fig. 8.5: Sensitivity comparison for different metrics (Anisotropy, Count, Volume) to the choice of the threshold.

8.6.1 A contradiction?

We see in `Fig <number> fig_comparesensitivity` 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.

8.7 Reproducibility

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

8.7.1 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

8.7.2 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

8.8 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.

8.8.1 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)
```

8.8.2 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);..."`

CHAPTER

NINE

DATA FRAMES - MANAGING FEATURE TABLES

9.1 Our workflow

- Image analysis
- Feature selection and analysis
- Presentation

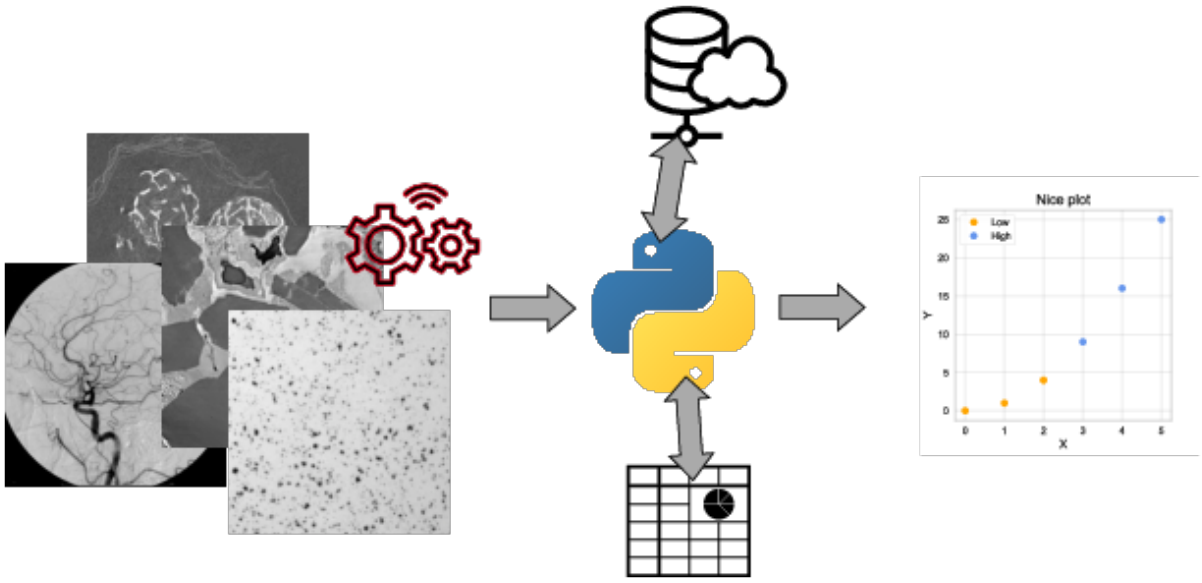


Fig. 9.1: Different destinations of the processed data.

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

9.2 How do we store the features?

9.2.1 Python offers different options

- Arrays per feature
- List of data structures/dictionaries

9.2.2 Operations on the feature data

- Counting
- Statistics
- Selections
- Transforms
- Visualization

9.2.3 Problem

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

- Time consuming
- Little flexibility
- Error prone

9.3 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 artihmetics
- Compute statistics
- Read and store into files and databases

Pandas documentation Getting started with Pandas

9.4 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
```

9.4.1 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)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [23], in <cell line: 1>()
----> 1 pheno = pd.read_csv('../../Exercises/10-Statistics_DataFrames/phenoTable.csv')
      2 pheno.sample(5)

File ~/miniforge3/lib/python3.9/site-packages/pandas/util/_decorators.py:311, in␣
 ↪deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args, **kwargs)
    305 if len(args) > num_allow_args:
    306     warnings.warn(
    307         msg.format(arguments=arguments),
    308         FutureWarning,
    309         stacklevel=stacklevel,
    310     )
--> 311 return func(*args, **kwargs)

File ~/miniforge3/lib/python3.9/site-packages/pandas/io/parsers/readers.py:586, in␣
 ↪read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols,␣
 ↪squeeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_
 ↪values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na,␣
 ↪na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_
 ↪col, date_parser, dayfirst, cache_dates, iterator, chunksize, compression,␣
 ↪thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar,␣
 ↪comment, encoding, encoding_errors, dialect, error_bad_lines, warn_bad_lines, on_
 ↪bad_lines, delim_whitespace, low_memory, memory_map, float_precision, storage_
 ↪options)
    571 kwds_defaults = _refine_defaults_read(
    572     dialect,
    573     delimiter,
   (...)
    582     defaults={"delimiter": ","},
    583 )
    584 kwds.update(kwds_defaults)
--> 586 return _read(filepath_or_buffer, kwds)
```

(continues on next page)

(continued from previous page)

```
File ~/miniforge3/lib/python3.9/site-packages/pandas/io/parsers/readers.py:482, in _
↪read(filepath_or_buffer, kwds)
    479 _validate_names(kwds.get("names", None))
    481 # Create the parser.
--> 482 parser = TextFileReader(filepath_or_buffer, **kwds)
    484 if chunksize or iterator:
    485     return parser

File ~/miniforge3/lib/python3.9/site-packages/pandas/io/parsers/readers.py:811, in␣
↪TextFileReader.__init__(self, f, engine, **kwds)
    808 if "has_index_names" in kwds:
    809     self.options["has_index_names"] = kwds["has_index_names"]
--> 811 self._engine = self._make_engine(self.engine)

File ~/miniforge3/lib/python3.9/site-packages/pandas/io/parsers/readers.py:1040, in␣
↪TextFileReader._make_engine(self, engine)
   1036     raise ValueError(
   1037         f"Unknown engine: {engine} (valid options are {mapping.keys()})"
   1038     )
   1039 # error: Too many arguments for "ParserBase"
-> 1040 return mapping[engine](self.f, **self.options)

File ~/miniforge3/lib/python3.9/site-packages/pandas/io/parsers/c_parser_wrapper.
↪py:51, in CParserWrapper.__init__(self, src, **kwds)
     48 kwds["usecols"] = self.usecols
     50 # open handles
---> 51 self._open_handles(src, kwds)
     52 assert self.handles is not None
     54 # Have to pass int, would break tests using TextReader directly otherwise :(

File ~/miniforge3/lib/python3.9/site-packages/pandas/io/parsers/base_parser.py:222,␣
↪in ParserBase._open_handles(self, src, kwds)
    218 def _open_handles(self, src: FilePathOrBuffer, kwds: dict[str, Any]) -> None:
    219     """
    220     Let the readers open IOHandles after they are done with their potential␣
↪raises.
    221     """
--> 222     self.handles = get_handle(
    223         src,
    224         "r",
    225         encoding=kwds.get("encoding", None),
    226         compression=kwds.get("compression", None),
    227         memory_map=kwds.get("memory_map", False),
    228         storage_options=kwds.get("storage_options", None),
    229         errors=kwds.get("encoding_errors", "strict"),
    230     )

File ~/miniforge3/lib/python3.9/site-packages/pandas/io/common.py:702, in get_
↪handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors,␣
↪storage_options)
    697 elif isinstance(handle, str):
    698     # Check whether the filename is to be opened in binary mode.
    699     # Binary mode does not support 'encoding' and 'newline'.
    700     if ioargs.encoding and "b" not in ioargs.mode:
    701         # Encoding
--> 702         handle = open(
    703             handle,
```

(continues on next page)

(continued from previous page)

```
    704             ioargs.mode,
    705             encoding=ioargs.encoding,
    706             errors=errors,
    707             newline="",
    708         )
    709     else:
    710         # Binary mode
    711         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: '../../Exercises/10-
 ↪Statistics_DataFrames/phenoTable.csv'
```

Saving works similarly:

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

9.4.2 Create a data frame using dicts

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)})
print(dl)
```

```
[{'position': 0, 'sine': 0.0}, {'position': 1, 'sine': 0.8414709848078965}, {'position
 ↪': 2, 'sine': 0.9092974268256817}, {'position': 3, 'sine': 0.1411200080598672}, {
 ↪'position': 4, 'sine': -0.7568024953079282}, {'position': 5, 'sine': -0.
 ↪9589242746631385}, {'position': 6, 'sine': -0.27941549819892586}, {'position': 7,
 ↪'sine': 0.6569865987187891}, {'position': 8, 'sine': 0.9893582466233818}, {'position
 ↪': 9, 'sine': 0.4121184852417566}]
```

```
df = pd.DataFrame(dl)

df.sample(5)
```

```
   position  sine
7         7  0.66
5         5 -0.96
0         0  0.00
8         8  0.99
3         3  0.14
```

9.4.3 An alternative...

You may already have numpy arrays with the data.

```
x    = np.linspace(0,2*np.pi,100)
sine = np.sin(x)
dd   = {'position' : x, 'sine' : sine}

ddf  = pd.DataFrame(dd)
ddf.head(5)
```

```
   position  sine
0      0.00  0.00
1      0.06  0.06
2      0.13  0.13
3      0.19  0.19
4      0.25  0.25
```

9.5 Working with columns

9.5.1 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['positive'] = 0<df['sine']
df.head()
```

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

9.5.2 Select some rows with content filtering

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

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

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

```
   hej  hopp
0    1     4
1    2     5
2    3     6
```

9.6 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
positive    0.60
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
```

9.7 Statistics of filtered data

Compute the standard deviation for all columns with the rows have 0<sum

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

```
0.5297967209946162
```

9.7.1 Statistics using grouping

Filtering works well for few categories, but for many categories some smarter tricks can be used:

```
# compute category statistics
group_name='positive'
data_agg = df.groupby(group_name).mean()

data_agg
```

```
          position  sine  cosine   sum
positive
False         3.75 -0.50    0.40 -0.10
True          5.00  0.66   -0.19  0.46
```

Too cluttered?

We may not be interested in the all columns, maybe want to add std dev.

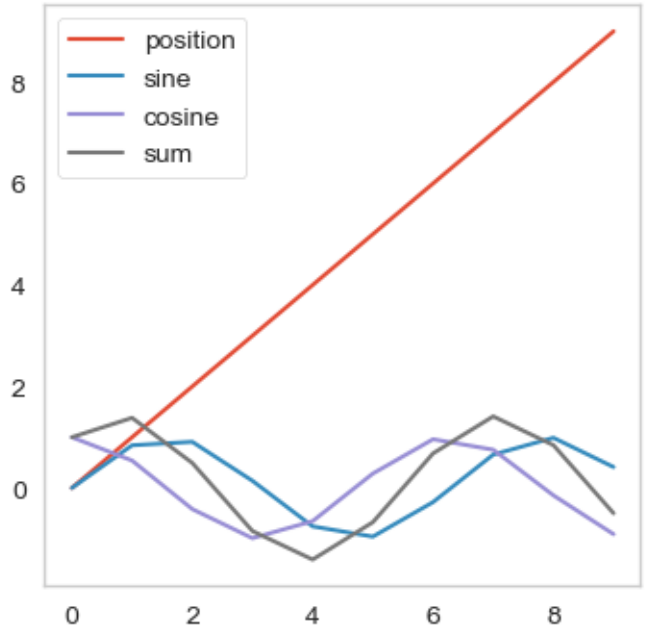
```
# compute category statistics
group_name='positive'
value_name='sine'
data_agg = df.groupby(group_name).agg({value_name: ['mean', 'var']}).reset_index()
data_agg.columns = data_agg.columns.get_level_values(1)

data_agg
```

```
          mean   var
0  False -0.50  0.19
1   True  0.66  0.11
```

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);
```



9.7.2 Selective plotting

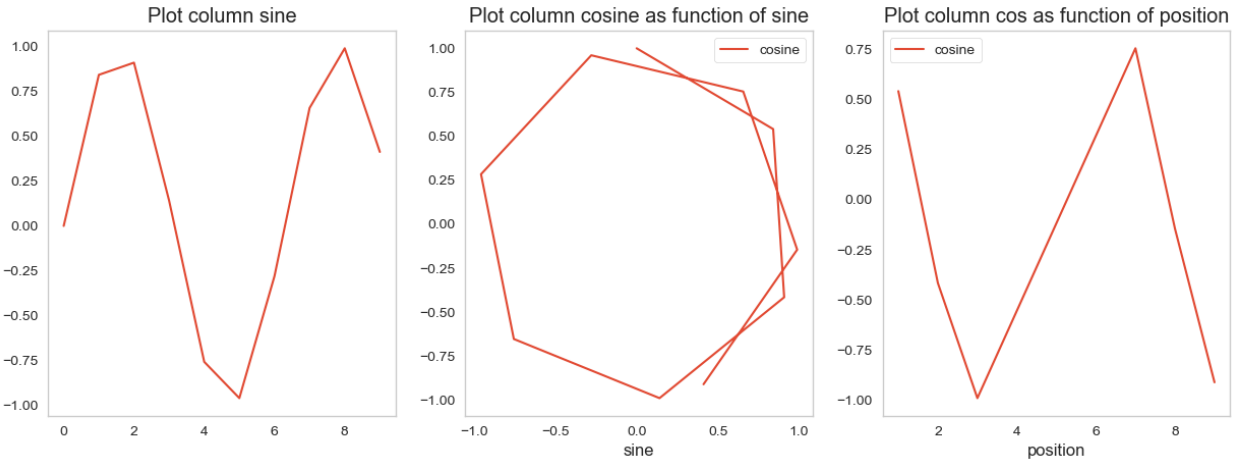
We mostly don't want to plot all columns at once

```
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");
```

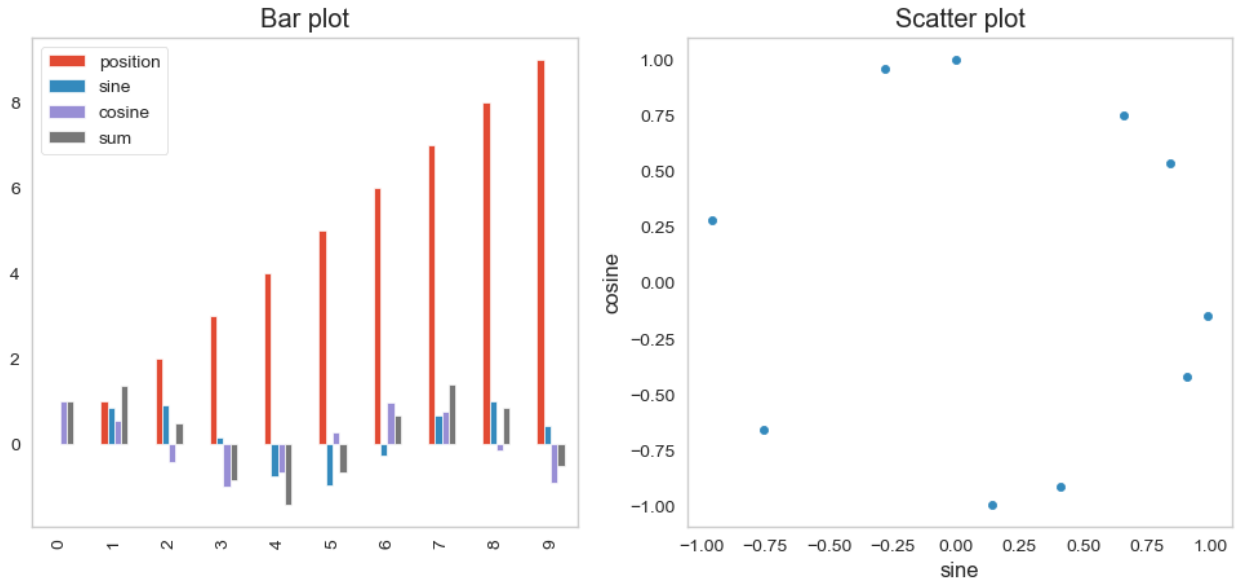


9.7.3 Different plotting styles

```
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

9.8 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

9.8.1 Concatenating frames

Add more rows with the same categories

Concatenation adds rows of two data frames with the same columns.

```
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
```

Concatenating this way has the disadvantage that the rows will be maintained from the original data frames. This can be avoided if you add an index vector when you concatenate.

```
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
```

(continues on next page)

(continued from previous page)

```
5  A5  B5  C5  D5
6  A6  B6  C6  D6
7  A7  B7  C7  D7
```

9.8.2 Merging data frames

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

Merging is when you add columns to the data frame.

```
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])

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
```

9.9 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
```

9.10 When are pandas data frames useful?

- Pandas is useful for **large data** - as long as it fits in the local memory.
- Really **big data** needs other options, e.g. dask data frames

9.10.1 How about big quantitative imaging?

You have seen the levels of reduction we produce when we apply different step in our processing chain. A good example is the sand grain segmentation last week. This in this example we started with a 3D grey-level image and ended up with a table with information about the individual grains, i.e. a reduction from an image with 500x500x300x16bit to a table 1600x8x32bit.

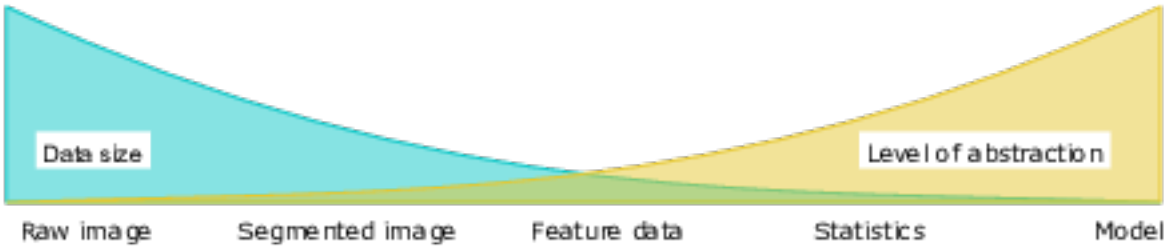


Fig. 9.2: The data size decreases radicall when we increase the level of abstraction.

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

An image is in general a chunk of memory which can be stored in any framework (matrix, 1D vector, column in a data frame). Now, if you add features to the image like coordinates, then you add a new column with the same length for each feature. This may be ok for a single image but if you would store a series of images, you would rapidliy fill the memory beyond the limitations of your system.

CHAPTER

TEN

PRESENTING THE RESULTS - BRINGING OUT THE MESSAGE

In the end you will want to present your results

Fig. 10.1: Different ways to present your data.

10.1 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:

1. **Understanding and exploration**

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

1. **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

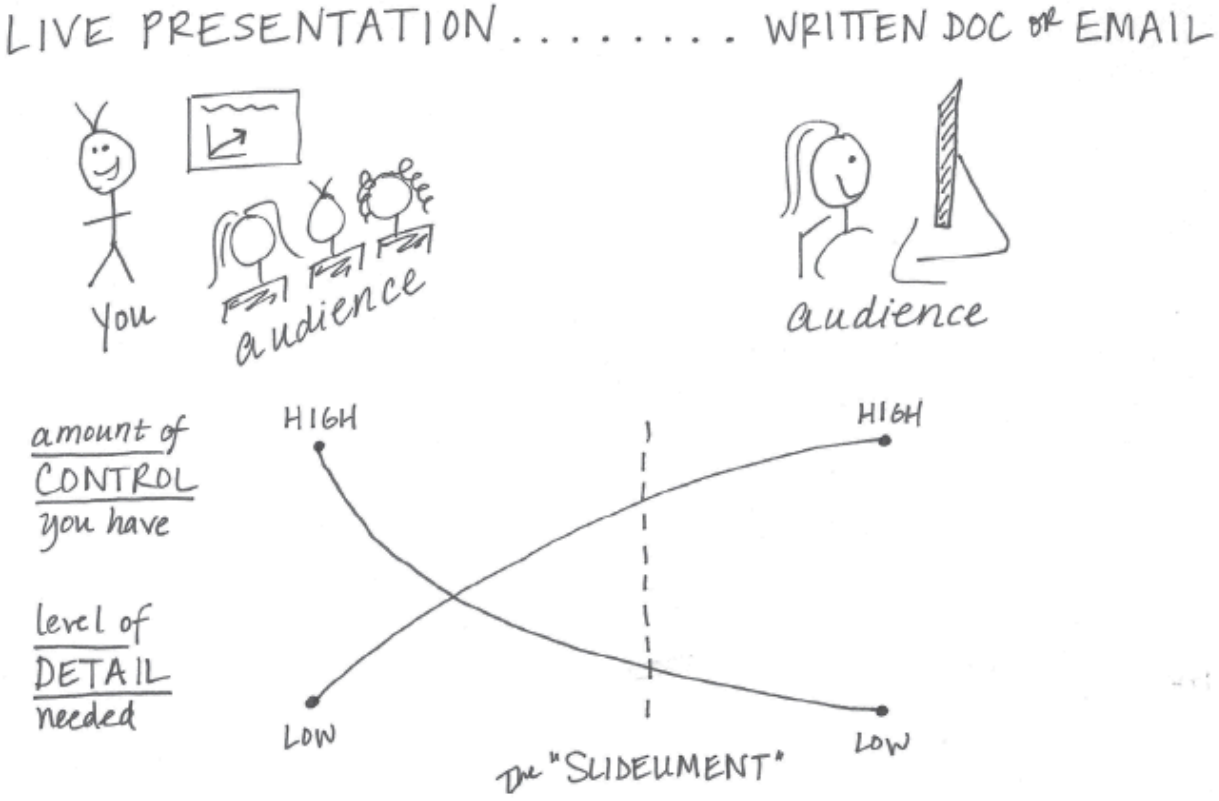


Fig. 10.2: The level of detail in a presentation depends on the medium it is presented Knaflic 2015.

10.2 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

xkcd

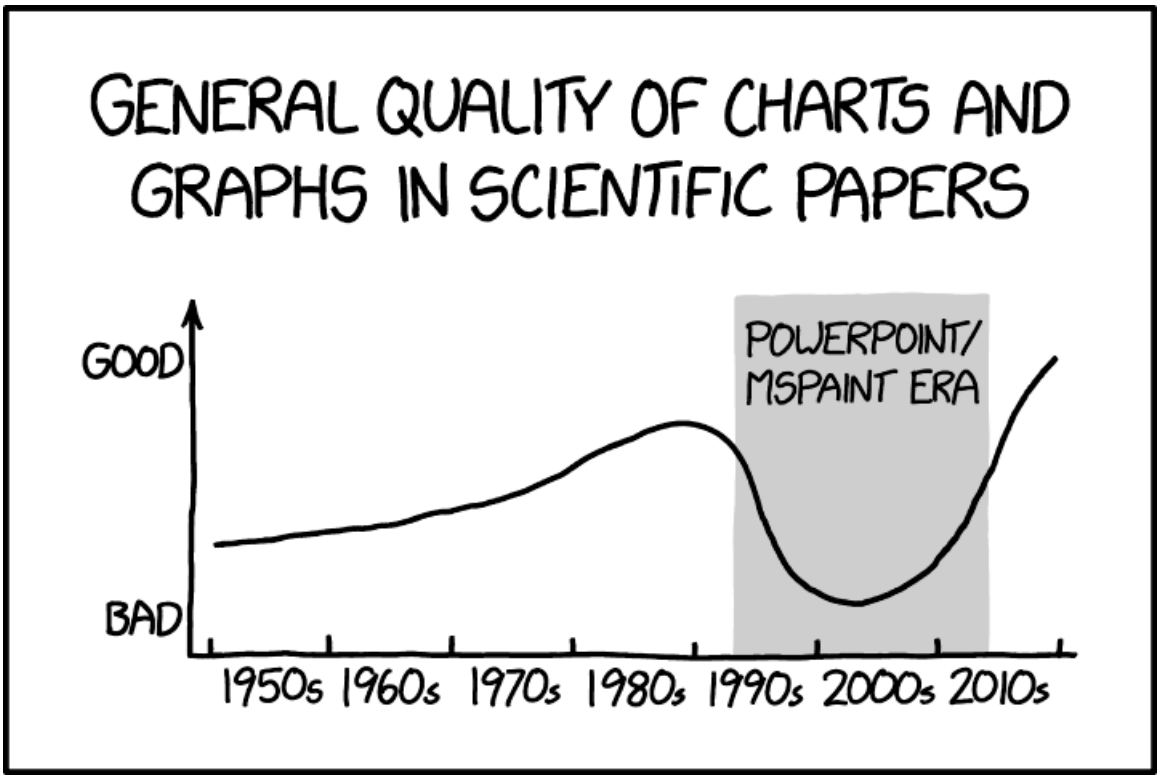


Fig. 10.3: This cartoon from XKCD highlights a problem with the access to software making it too easy to produce a graph or illustration.

10.2.1 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 predigestend before it is ready to any audience.

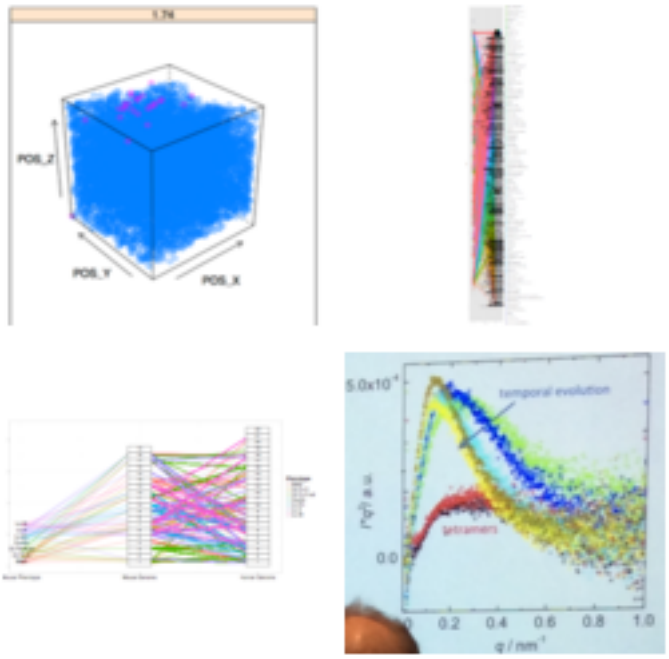


Fig. 10.4: Four examples of how *not* to present your data.

10.3 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 you 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.

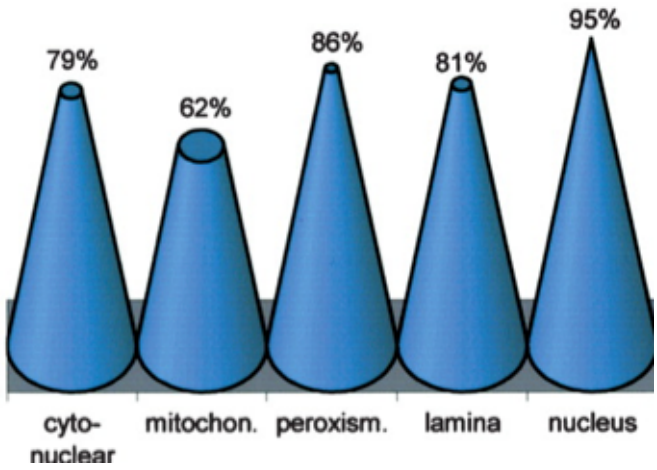
10.3.1 Some literature

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

10.3.2 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

10.4 Grammar of Graphics

10.4.1 What is a grammar?

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

10.4.2 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.

10.4.3 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

10.4.4 Grammar

Separate the graph into its component parts

Construct graphics by focusing on each portion independently.

10.4.5 Figure decorations

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

It may seem unnescessary 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 offical…

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 - tellning what information you see on each axis.
- Figure title - if you use multiples 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.

10.4.6 Color maps revisited

Choosing the right color is a science.

Crameri, F., et al. (2020)

The choice depends on the type of data you want to present and how humans percieve different colors. Some combinations make it easier to highlight relevant features in the images. Still, you have to be cautious not put too much a priori information into the color map.

Visualization toolboxes provide a great collection of colormaps as we have seen several times already in this course. There are however cases when you have to define your own color map. An example is the colormap we created last week to be able to identify each item in a watershed segmented image.

10.5 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
import numpy as np
# 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();
```

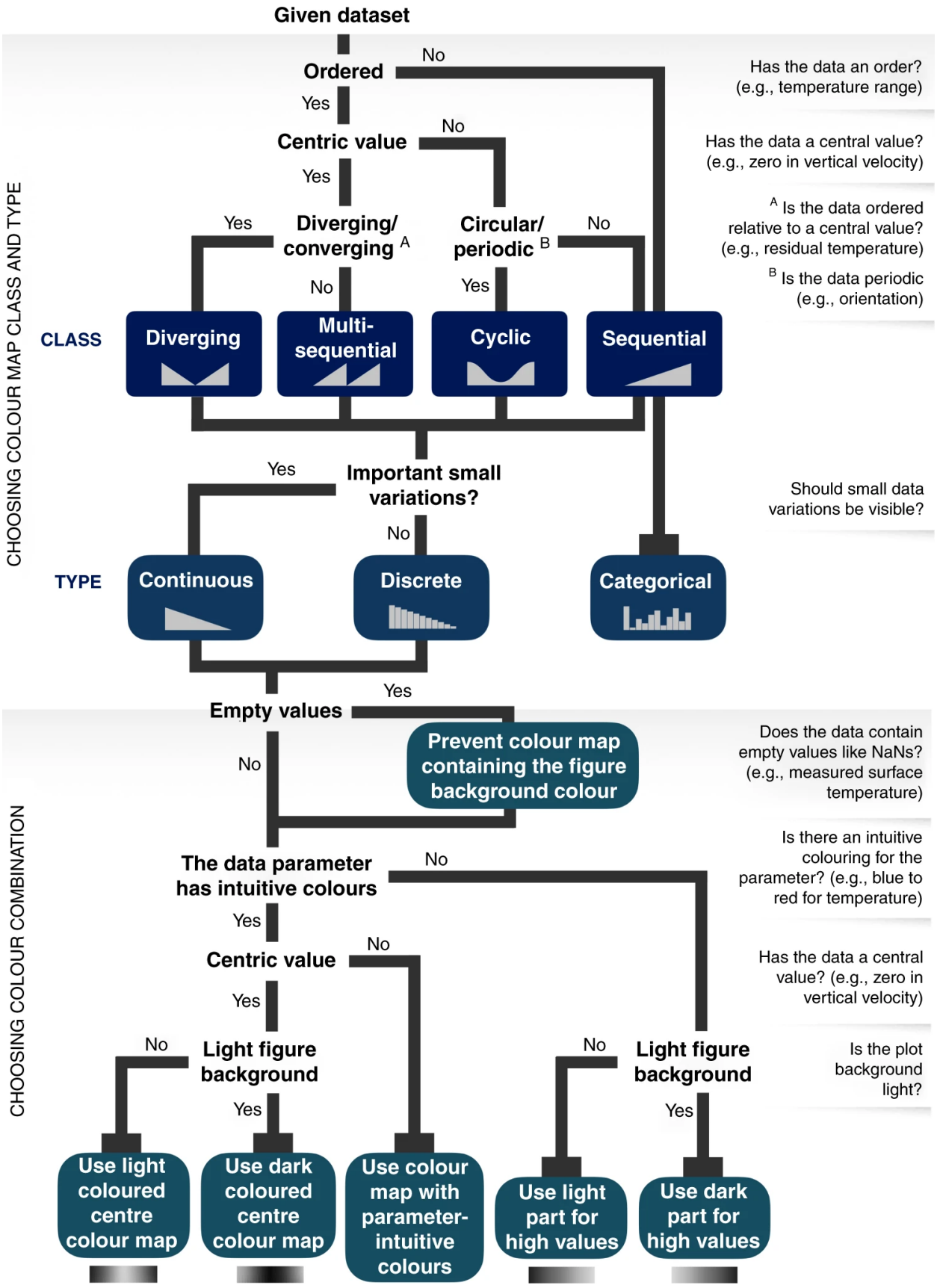
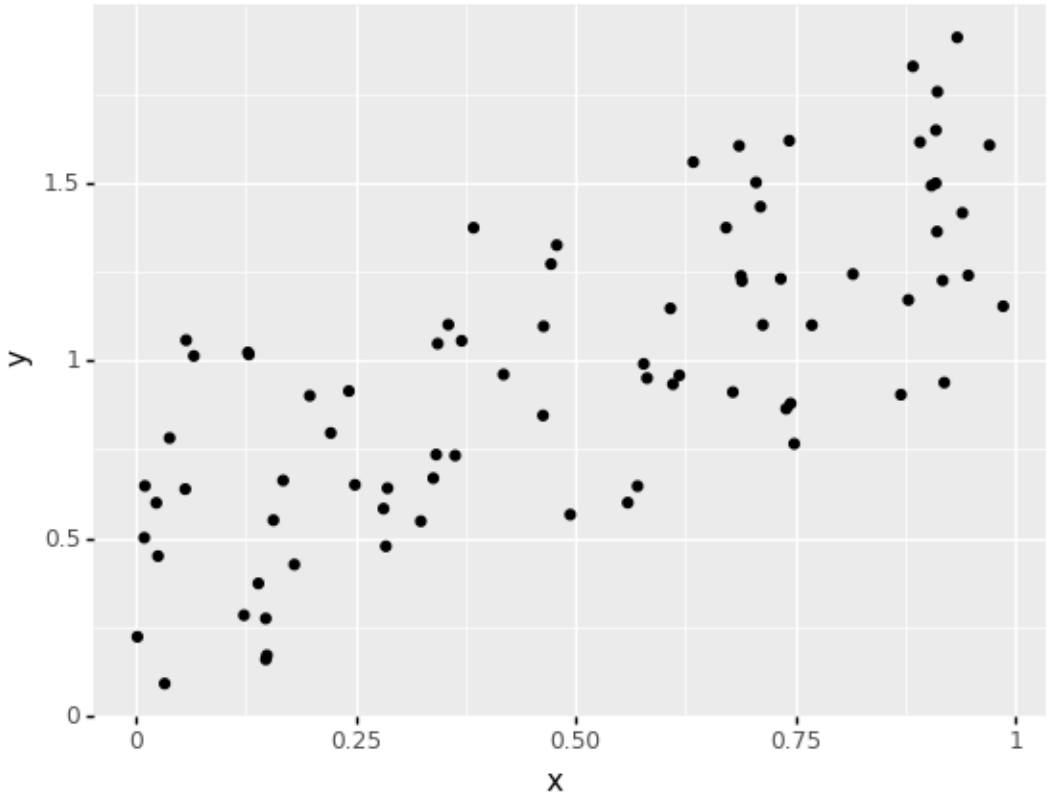


Fig. 10.5: Crameri et al. developed this decision flow chart to help you decide which type of color map is best suited for your data.

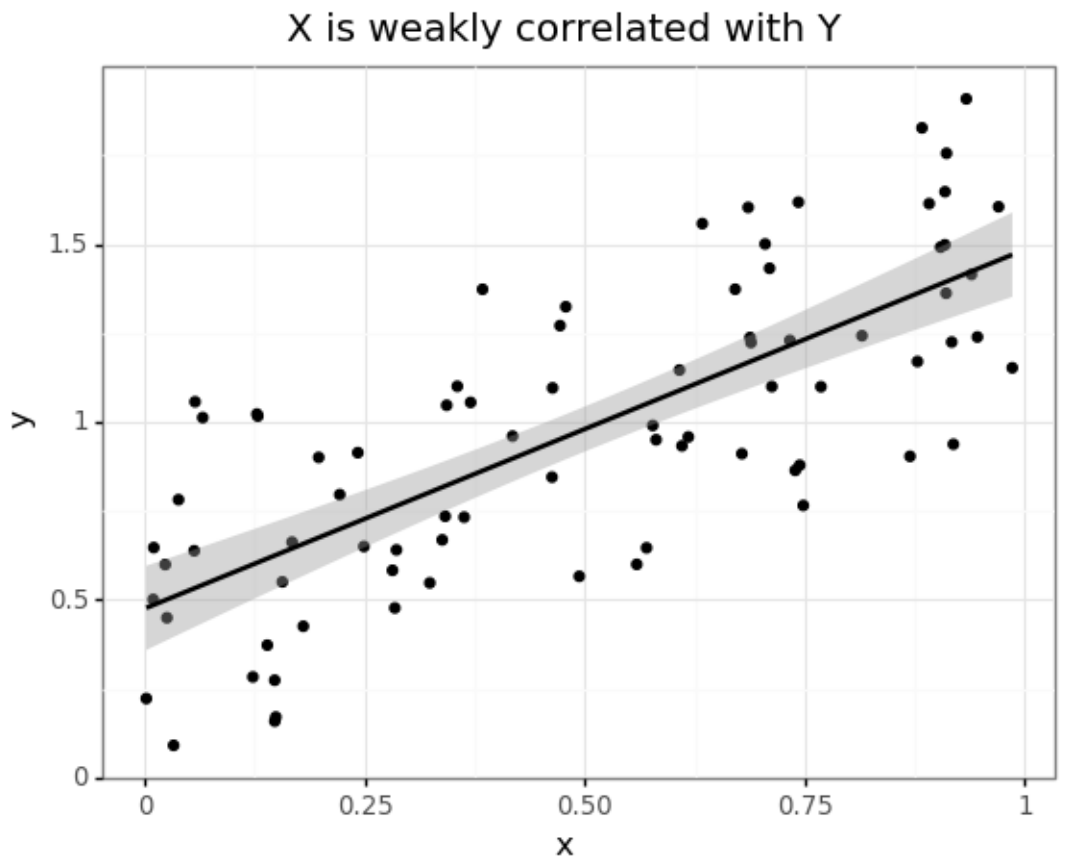



```
<ggplot: (8768353912197)>
```

10.5.1 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(12) );
```



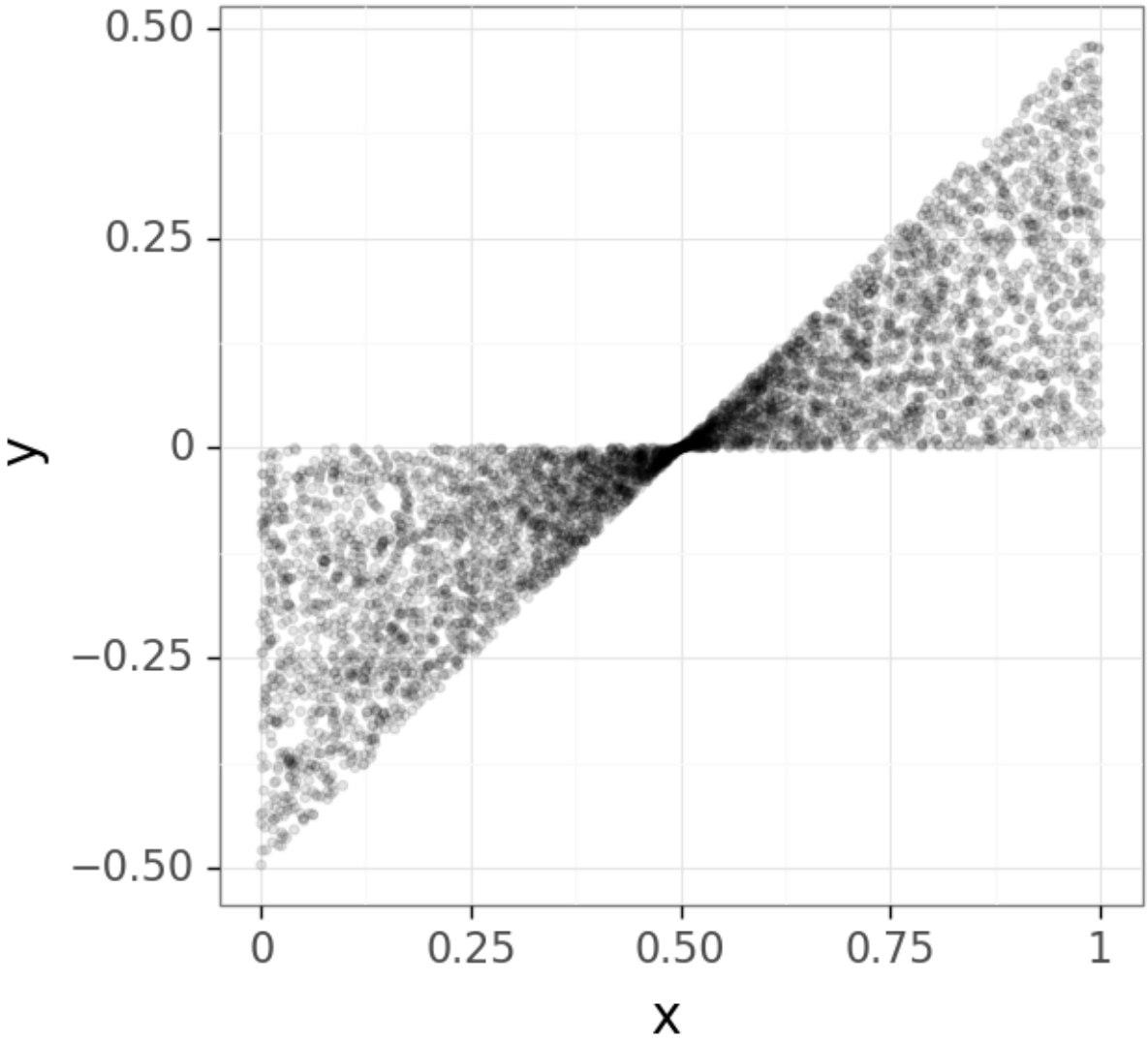

```
<ggplot: (8768353974538)>
```

10.5.2 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( alpha = 0.1 )
 + coord_equal()
 + theme_bw(20));
```




```
<ggplot: (8768369430200)>
```

We have earlier used transparency to better visualize dense scatter plots. You can see the effect by setting the alpha parameter to geom_point. Using transparency is a qualitative way of showing higher density in the data.

10.5.3 Reduce the data

Filter and reduce information until it is extremely simple

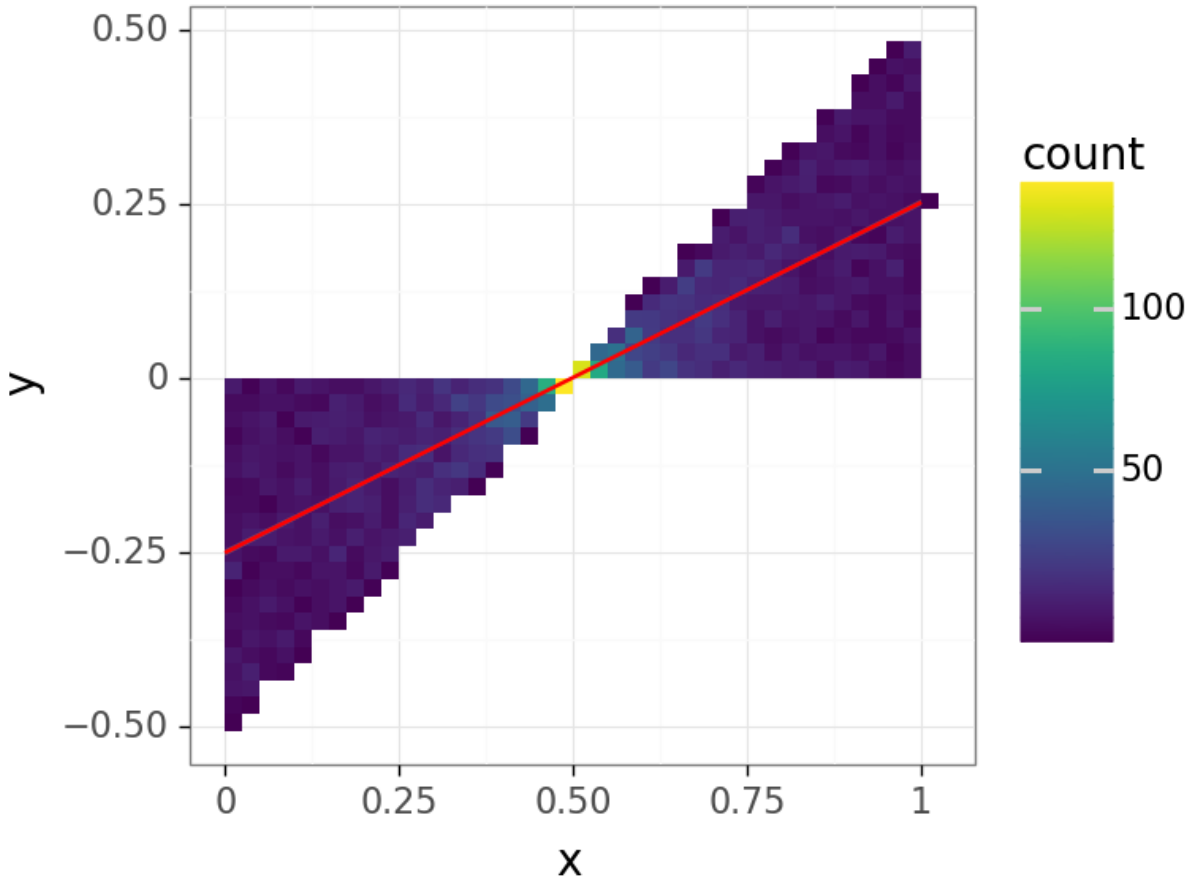
In this plot we create a density count view of the data by downsampling the grid and count the amount of points in each bin. It is related to a histogram but it count in space instead of in the intensity levels.

```
(ggplot(df,aes(x='x',y='y'))
 + stat_bin_2d(bins=40)
 + geom_smooth(method="lm",color='red')
 + coord_equal()
 + theme_bw(20)
```

(continues on next page)

(continued from previous page)

```
 + guides(color='F')
);
```

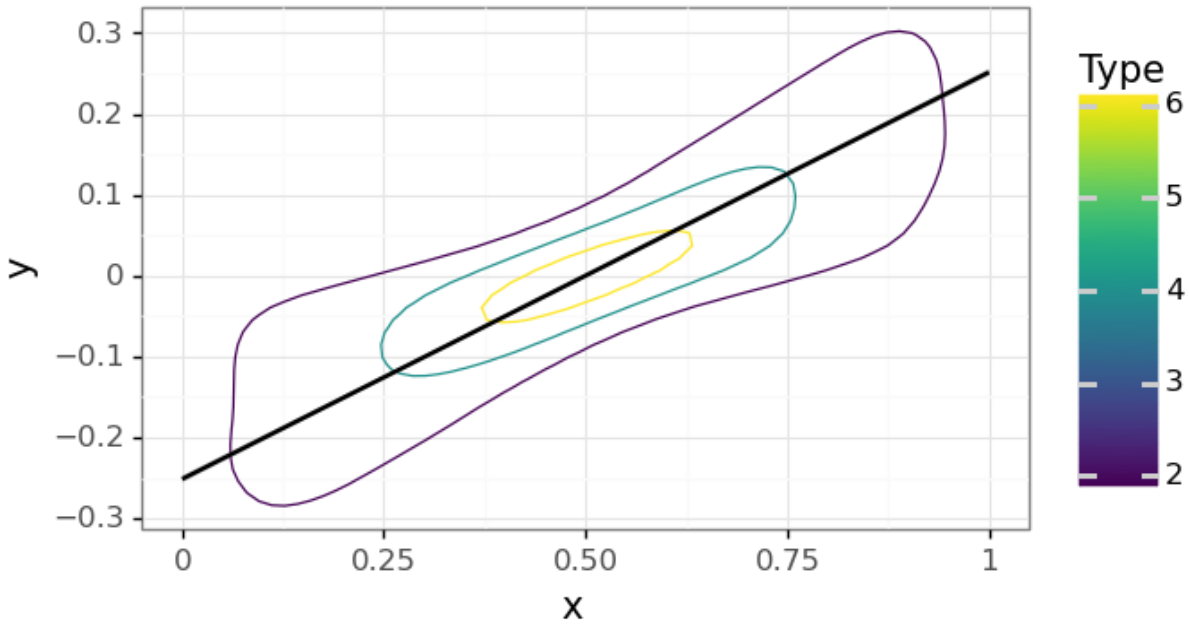



```
<ggplot: (8768335630873)>
```

Using this kind of plot allows us to measure how many points there are in each bin and thus we are now going towards a quantitative plot. The colorbar on the side helps us to interpret the colors.

10.5.4 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: (8768369635994)>
```

10.6 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

ELEVEN

SUMMARY

11.1 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

11.2 Data frames

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

11.3 Visualization

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

11.4 Old slides to transfer from R to python

old lecture 2017 old lecture 2019