Processing tables with Python

Marcelo L. Zoccoler

With materials from

Robert Haase, PoL – TU Dresden

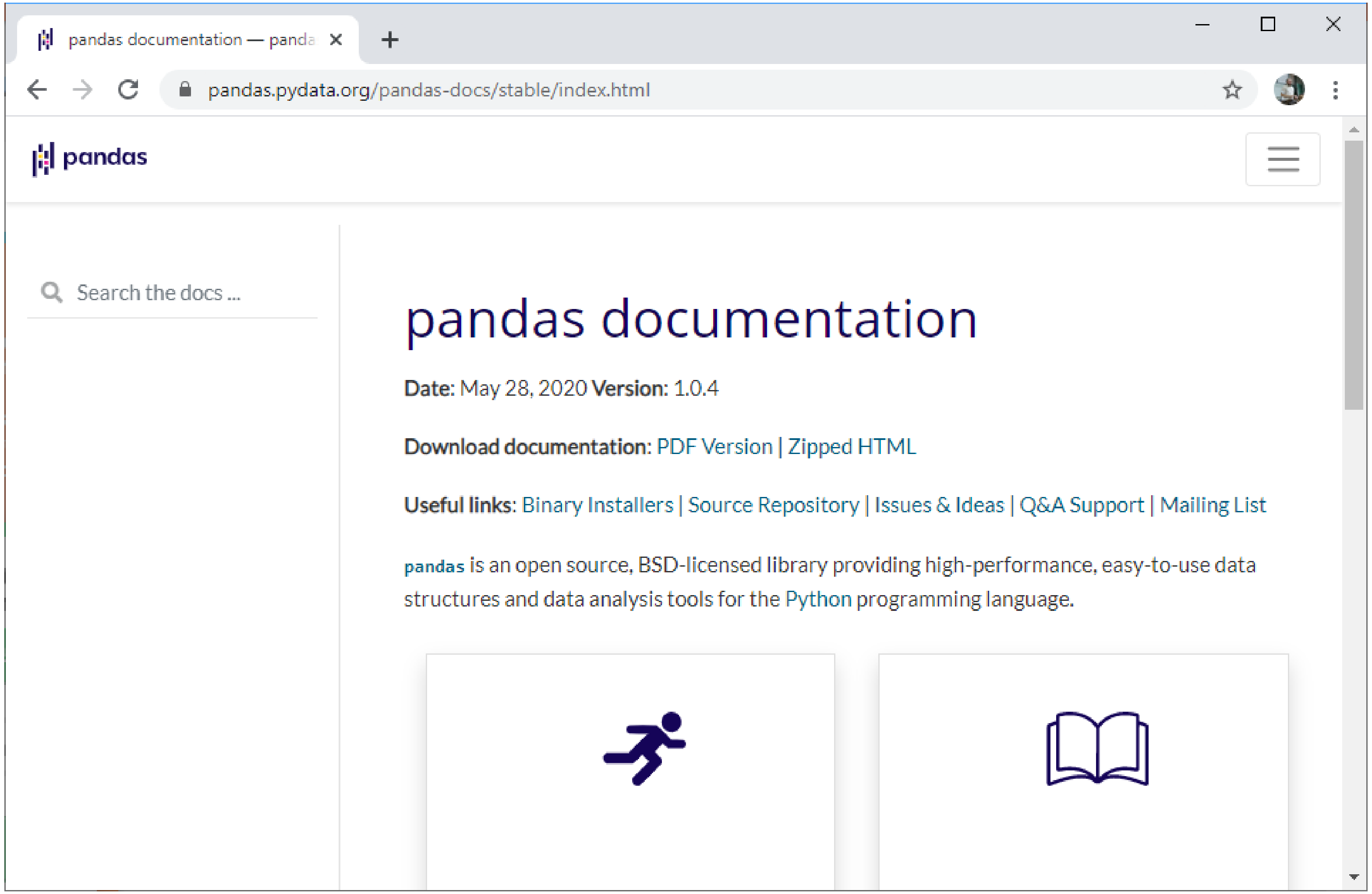
October 2022

@zoccolermarcelo

Pandas

- pandas is a library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

```
conda install pandas
```



Processing tables with pandas

- Typical use-case:
  - You get data from a colleague in form of a table
  - You get a table as output of a function and save it to disk

  - Using pandas, you can analyze it in python.

- Loading a table in python using pandas:

```
import pandas as pd

data_frame = pd.read_csv("Measurements_ImageJ.csv", delimiter=',')
data_frame
```

		Area	Mean	Circ.	AR	Round	Solidity
0	1	2610	96.920	0.773	1.289	0.776	1.0
1	2	2100	90.114	0.660	2.333	0.429	1.0
2	3	27	110.222	0.108	27.000	0.037	1.0

Display just the 5 first rows of a table:

```
data_frame.head(5)
```

Display just the 5 last rows of a table:

```
data_frame.tail(5)
```

Processing tables with pandas

- Typical use-case:
  - You get data from a colleague in form of a table
  - You get a table as output of a function and save it t
  - Using pandas, you can analyze it in python.

How do we get a row from the table?

		Area	Mean	Circ.	AR	Round	Solidity
0	1	2610	96.920	0.773	1.289	0.776	1.0
1	2	2100	90.114	0.660	2.333	0.429	1.0
2	3	27	110.222	0.108	27.000	0.037	1.0

```
data_frame.iloc[0,:]
```

```
data_frame.loc[0,:]
```

- Accessing a column

```
data_frame["Mean"]
```

```
0      96.920
1      90.114
2     110.222
Name: Mean, dtype: float64
```

- Determining mean of a column

```
import numpy as np
np.mean(data_frame["Mean"])
```

```
99.08533333333332
```

- Accessing an individual cell

```
data_frame["Mean"][0]
```

```
1.2890000000000001
```

```
data_frame.loc[0, "Mean"]
```

```
data_frame.iloc[0, 2]
```

Processing tables with pandas

- Creating tables with pandas

- Creating a new table

```
header = ['A', 'B', 'C']

data = [
    [1, 2, 3],  # this will later be colum A
    [4, 5, 6],  #                          B
    [7, 8, 9]   #                          C
]

# convert the data and header arrays in a pandas data frame
data_frame = pd.DataFrame(data, header)

# show it
data_frame
```

	0	1	2
A	1	2	3
B	4	5	6
C	7	8	9

```
cities_dict = {'City': ['Tokyo', 'Delhi', 'Shangai', 'Sao Paulo', 'Mexico City'],
               'Country': ['Japan', 'India', 'China', 'Brasil', 'Mexico'],
               'Population': [13515271, 16753235, 24183000, 12252023, 9209944],
               'Area_km2': [2191, 1484, 6341, 1521, 1485]}
```

```
cities = pd.DataFrame(cities_dict)
```

- *Rotate* a table

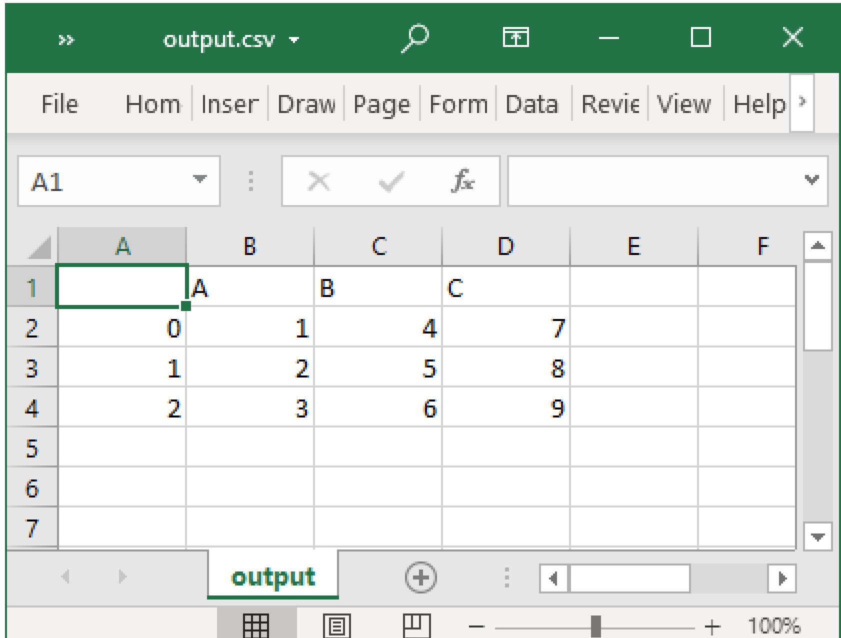
```
# rotate/flip it
data_frame = data_frame.transpose()

# show it
data_frame
```

	A	B	C
0	1	4	7
1	2	5	8
2	3	6	9

- Save it to disc

```
# save a dataframe to a CSV
data_frame.to_csv("output.csv")
```



	A	B	C	D	E	F
1		A	B	C		
2	0	1	4	7		
3	1	2	5	8		
4	2	3	6	9		
5						
6						
7						

Selecting rows and columns

	City	Country	Population	Area_km2
0	Tokyo	Japan	13515271	2191
1	Delhi	India	16753235	1484
2	Shanghai	China	24183000	6341
3	Sao Paulo	Brazil	12252023	1521
4	Mexico City	Mexico	9209944	1485

- Selecting columns

```
cities[['City', 'Country']]
```

	City	Country
0	Tokyo	Japan
1	Delhi	India
2	Shanghai	China
3	Sao Paulo	Brazil
4	Mexico City	Mexico

- Selecting rows

```
cities['Area_km2'] > 2000
```

```
0     True
1    False
2     True
3    False
4    False
Name: Area_km2, dtype: bool
```

```
cities[cities['Area_km2'] > 2000]
```

	City	Country	Population	Area_km2
0	Tokyo	Japan	13515271	2191
2	Shanghai	China	24183000	6341

Combining similar tables

- The big art in data science is the ability of combining information from multiple sources to gain new insights.
- If tables have the same columns

```
countries1
```

	Country	Population
0	Japan	127202192
1	India	1352642280
2	China	1427647786

```
countries2
```

	Country	Population
0	Brazil	209489323
1	Mexico	126190788

```
countries1['Survey ID'] = 26
countries1
```

	Country	Population	Survey ID
0	Japan	127202192	26
1	India	1352642280	26
2	China	1427647786	26

```
countries2['Survey ID'] = 73
countries2
```

	Country	Population	Survey ID
0	Brazil	209489323	73
1	Mexico	126190788	73

	Country	Population	Survey ID
0	Japan	127202192	26
1	India	1352642280	26
2	China	1427647786	26
0	Brazil	209489323	73
1	Mexico	126190788	73

```
pd.concat([countries1, countries2])
```

	Country	Population
0	Japan	127202192
1	India	1352642280
2	China	1427647786
0	Brazil	209489323
1	Mexico	126190788

Combining different tables

- The big art in data science is the ability of combining information from multiple sources to gain new insights.

	Country	Population
0	Japan	127202192
1	India	1352642280
2	China	1427647786
3	Brazil	209469323
4	Mexico	126190788

	City	Country	Population	Area_km2
0	Tokyo	Japan	13515271	2191
1	Delhi	India	16753235	1484
2	Shanghai	China	24183000	6341
3	Sao Paulo	Brazil	12252023	1521
4	Mexico City	Mexico	9209944	1485

```
combined = countries.merge(cities, on='Country', suffixes=['_country', '_city'])
combined
```

	Country	Population_country	City	Population_city	Area_km2
0	Japan	127202192	Tokyo	13515271	2191
1	India	1352642280	Delhi	16753235	1484
2	China	1427647786	Shanghai	24183000	6341
3	Brazil	209469323	Sao Paulo	12252023	1521
4	Mexico	126190788	Mexico City	9209944	1485

```
# compute ratio
combined['City_Country_population_ratio'] = combined['Population_city'] / combined['Population_country']

# only show selected columns
combined[['City', 'City_Country_population_ratio']]
```

	City	City_Country_population_ratio
0	Tokyo	0.106250
1	Delhi	0.012386
2	Shanghai	0.016939
3	Sao Paulo	0.058491
4	Mexico City	0.072984

Handling NaNs

- Sometimes tables may contains NaNs (Not a Number). These values may come from missing experimental data or from missing data when merging tables.

- They can introduce errors to calculations with tables.

- The easiest way to drop them is to use the ".dropna" method. This will drop any rows that contain NaN.

```
data_no_nan = data.dropna(how="any")
data_no_nan
```

- But be careful, do not drop NaNs carelessly. Try to investigate first why they are there. Also, you may accidentally discard useful data from other columns.

Tidy-Data

- Tidy data frames follow the rules:
  - Each variable is a column.
  - Each observation is a row.
  - Each type of observation has its own separate data frame.

Which of these data is tidy?

	Before		After	
	channel_1	channel_2	channel_1	channel_2
0	13.250000	21.000000	15.137984	42.022776
1	44.954545	24.318182	43.328836	48.661610
2	13.590909	18.772727	11.685995	37.926184
3	85.032258	19.741935	86.031461	40.396353

```
df['intensity_mean'] > 200
```

```
df['intensity_mean'] > 200
```

	time	label	intensity_mean	area
0	0	1	233.5	20
1	0	2	403.0	40
2	0	3	255.3	30
3	1	1	244.5	20
4	1	2	402.0	40
5	1	3	256.7	30
6	2	1	278.9	20
7	2	2	401.2	40
8	2	3	255.1	30

Tidy-Data

- Tidy data frames follow the rules:
  - Each variable is a column.
  - Each observation is a row.
  - Each type of observation has its own separate data frame.

Using pd.melt may help tidying data

	Before		After	
	channel_1	channel_2	channel_1	channel_2
0	13.250000	21.000000	15.137984	42.022776
1	44.954545	24.318182	43.328836	48.661610
2	13.590909	18.772727	11.685995	37.926184
3	85.032258	19.741935	86.031461	40.396353

```
df.melt()
```

	variable_0	variable_1	value
0	Before	channel_1	13.250000
1	Before	channel_1	44.954545
2	Before	channel_1	13.590909
3	Before	channel_1	85.032258
4	Before	channel_1	10.731707
...	...	...	...
99	After	channel_2	73.286439
100	After	channel_2	145.900739

Split-Apply-Combine

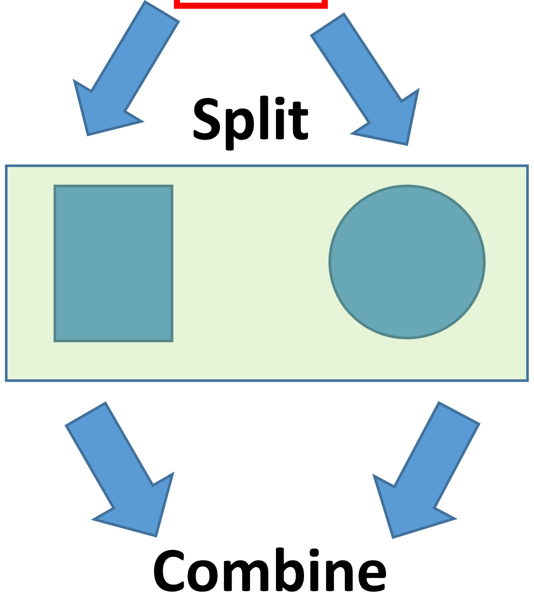
	area	intensity_mean	major_axis_length	minor_axis_length	aspect_ratio	file_name	round
0	139	96.546763	17.504104	10.292770	1.700621	20P1_POS0010_D_1UL	False
1	360	86.613889	35.746808	14.983124	2.385805	20P1_POS0010_D_1UL	False
2	43	91.488372	12.967884	4.351573	2.980045	20P1_POS0010_D_1UL	False
3	140	73.742857	18.940508	10.314404	1.836316	20P1_POS0010_D_1UL	False
4	144	89.375000	13.639308	13.458532	1.013432	20P1_POS0010_D_1UL	True

- compute the median "intensity_mean“
- of round objects

```
grouped = df.groupby('round')
```

Apply (calculate median):

```
df_median = grouped.median()
```




```
df_median.reset_index()
```

	round	area	intensity_mean	major_axis_length	minor_axis_length	aspect_ratio
0	False	270.0	92.788345	21.459495	15.858324	1.412849
1	True	291.0	100.256000	20.155547	18.352287	1.101700

Descriptive [Bio-] statistics

Marcelo L. Zoccoler

With material from

Robert Haase, PoL – TU Dresden

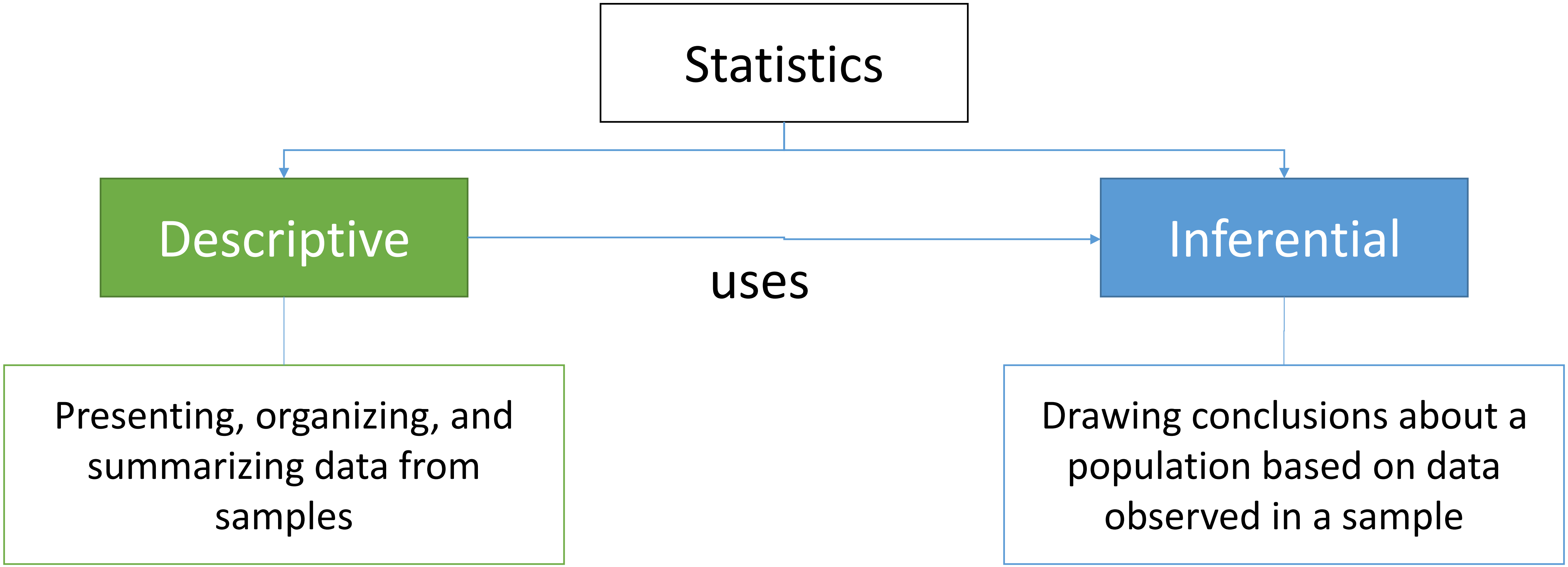
Aldo Acevedo Toledo, Biotec, TU Dresden

Martin J. Bland and Douglas G. Altman

May 2022

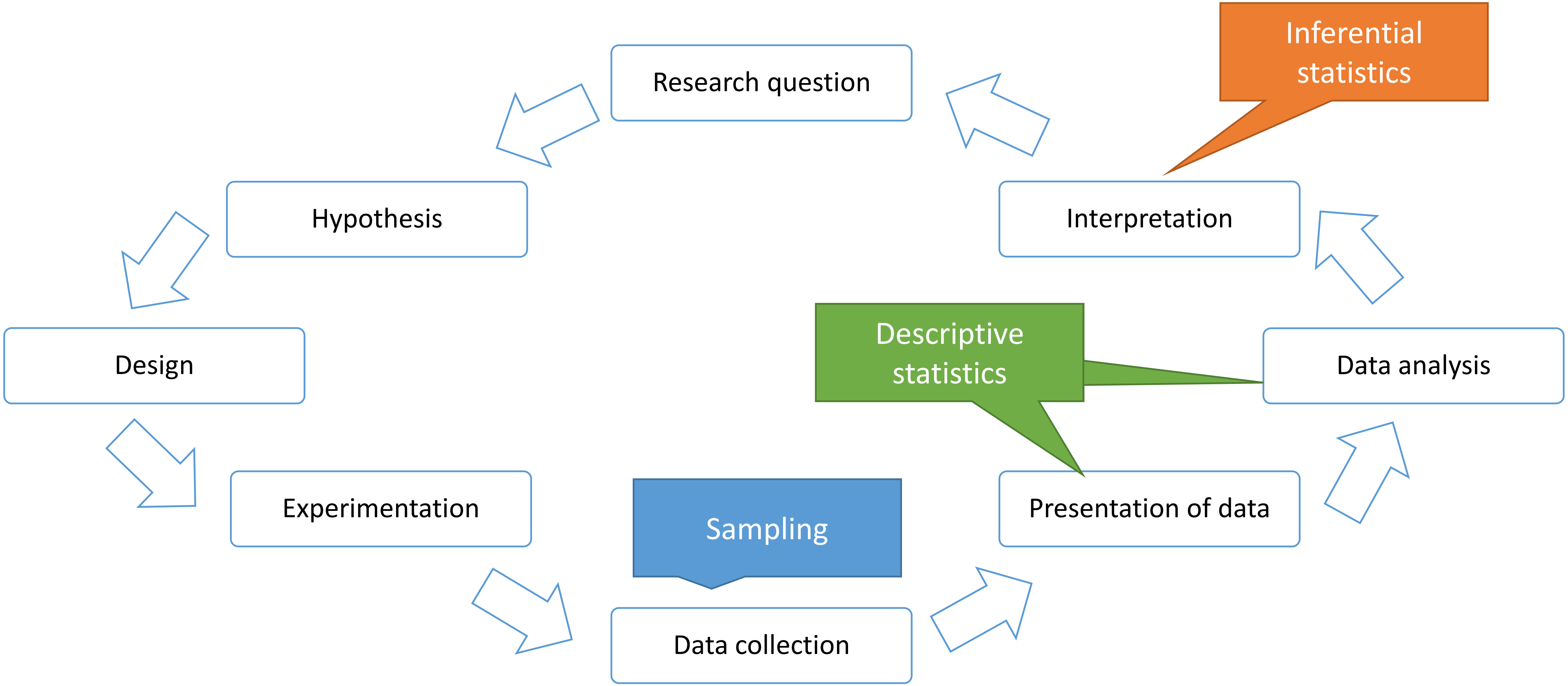
@zoccolermarcelo

Descriptive statistics



Slide adapted from
Aldo Acevedo Toledo, Biotec, TU Dresden

Research workflow



@zoccolermarcelo

Slide adapted from
Aldo Acevedo Toledo, Biotec, TU Dresden

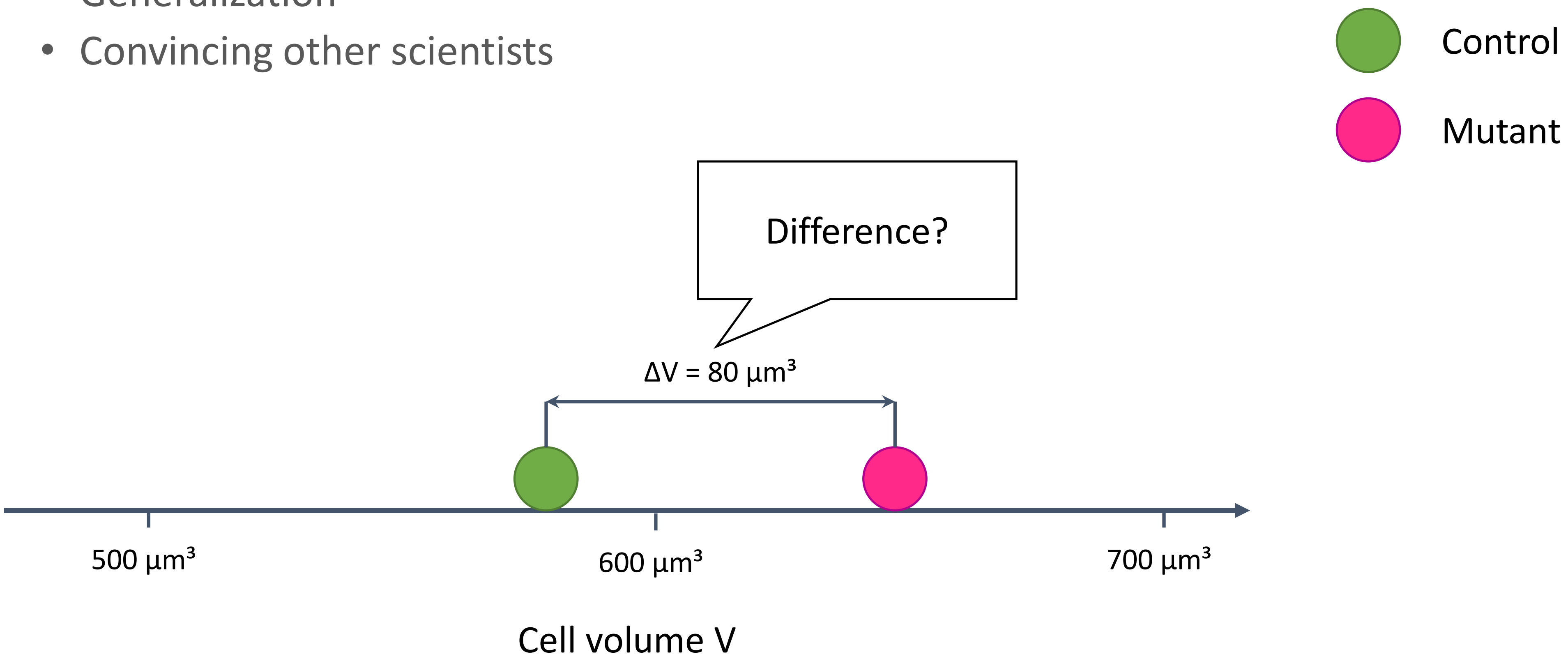
Sampling

Convenience Sampling	• Select the most accessible and available subjects in target population • Inexpensive, less time consuming, but sample is nearly always non-representative of target population
Random Sampling	• Select subjects at random from the target population • Need to identify all in target population first • Provides representative sample frequently
Systematic Sampling	• Identify all in target population • Select every n^{th} item as a subject
Stratified Sampling	• Identify important sub-groups in your target population. Sample from these groups randomly or by convenience • Ensures that important sub-groups are included in sample • May not be representative

Slide adapted from
Aldo Acevedo Toledo, Biotec, TU Dresden

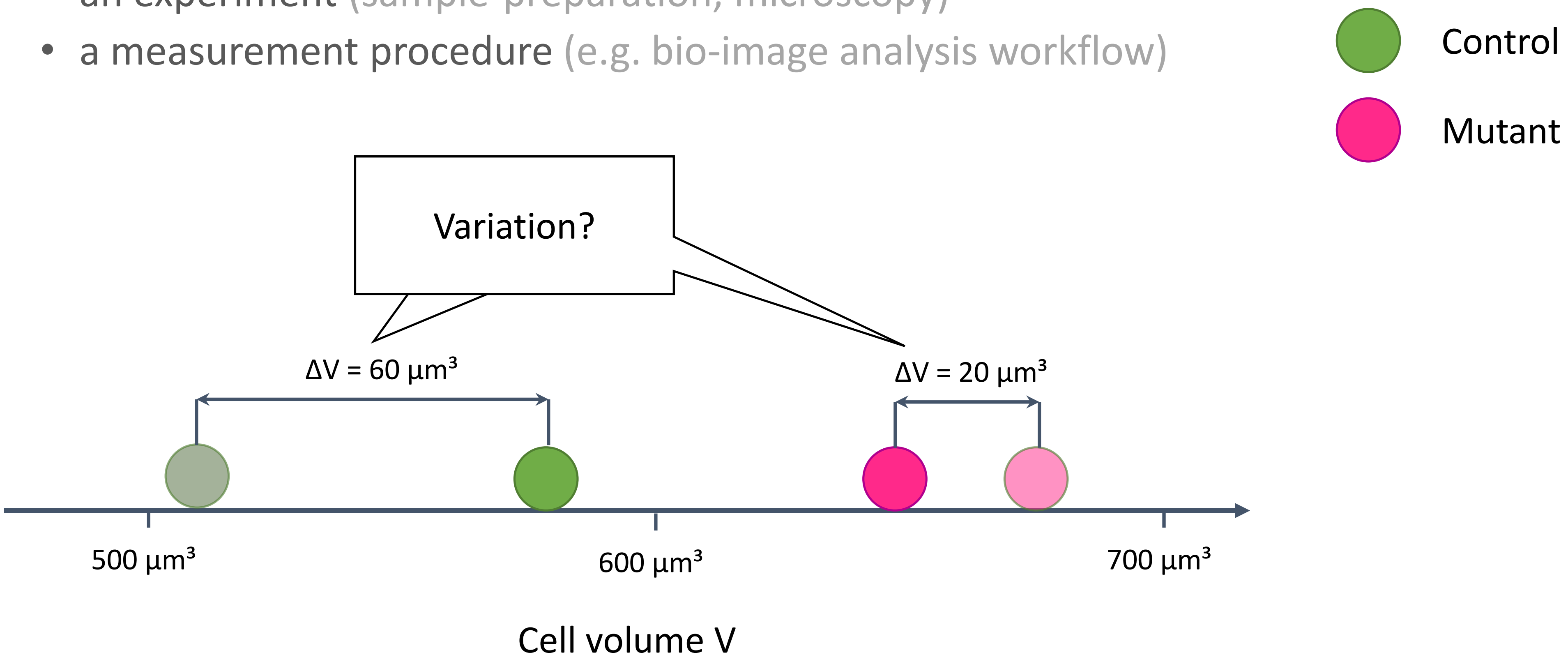
Sampling

- Raw individual measurements are limited regarding
  - Interpretability
  - Generalization
  - Convincing other scientists



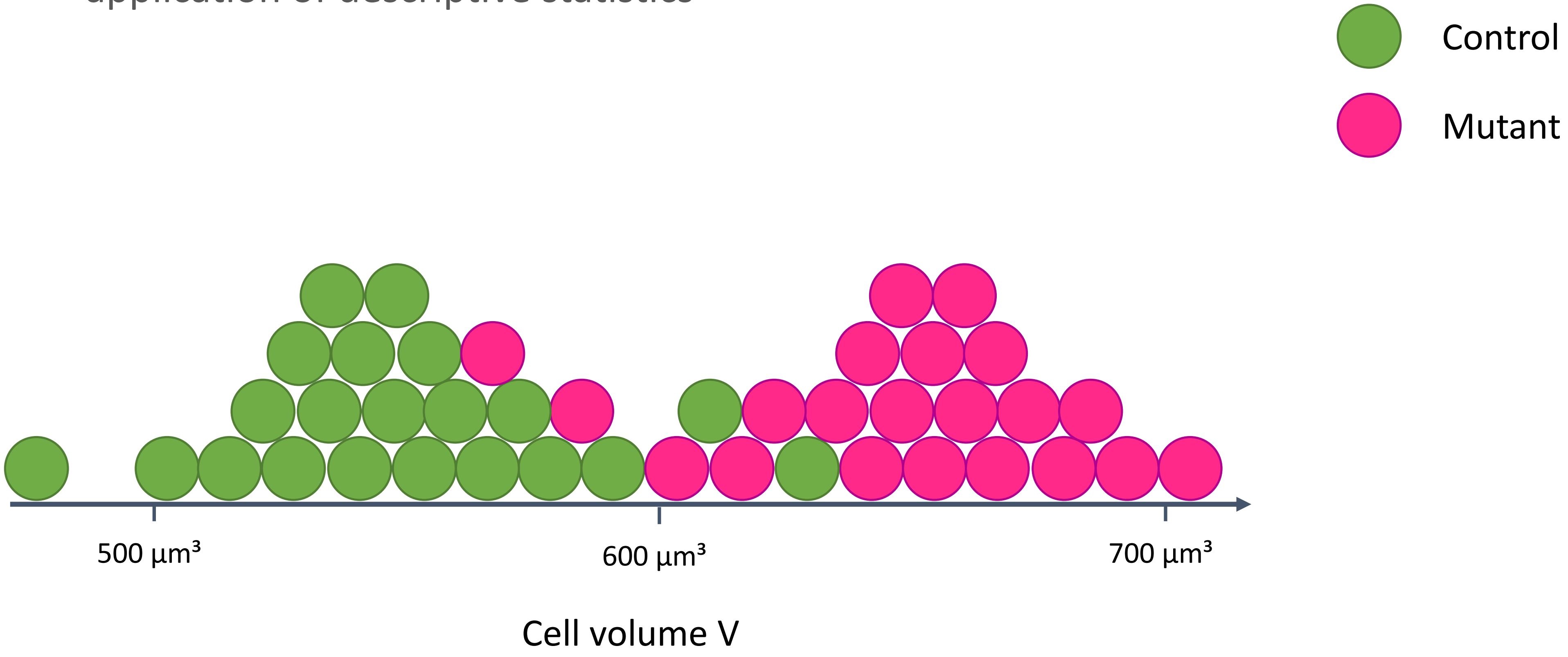
Sampling

- Raw individual measurements are result of
  - a stochastic process (cell life-cycle, biology)
  - an experiment (sample-preparation, microscopy)
  - a measurement procedure (e.g. bio-image analysis workflow)



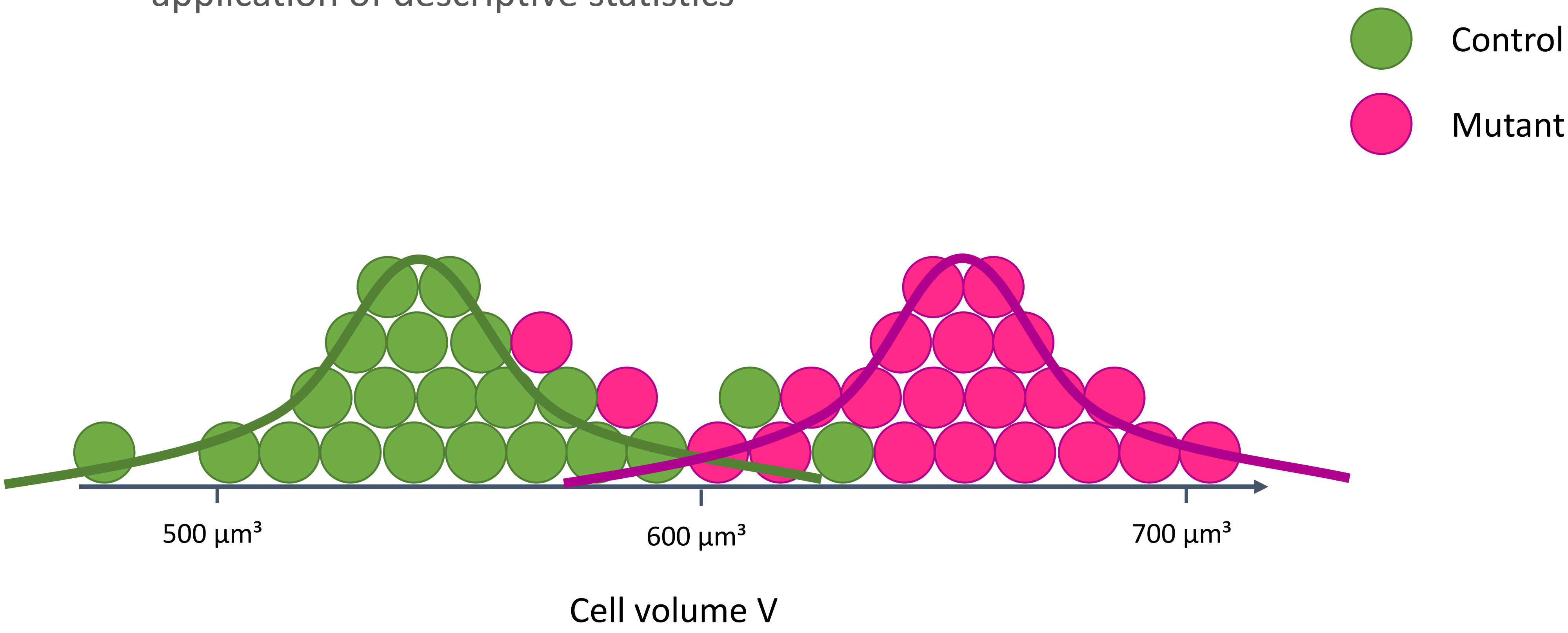
Sampling

- Repetitive experiments allow
  - a closer view
  - application of descriptive statistics



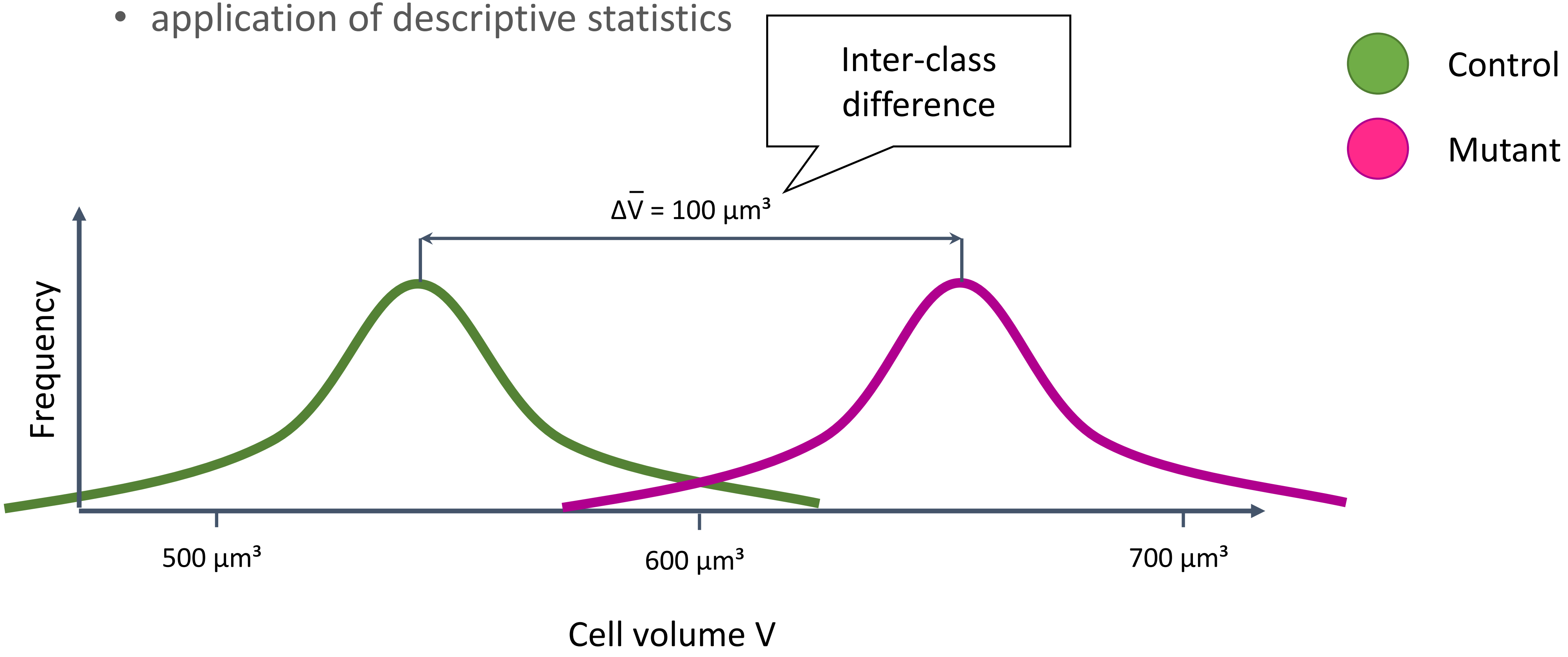
Sampling

- Repetitive experiments allow
  - a closer view
  - application of descriptive statistics



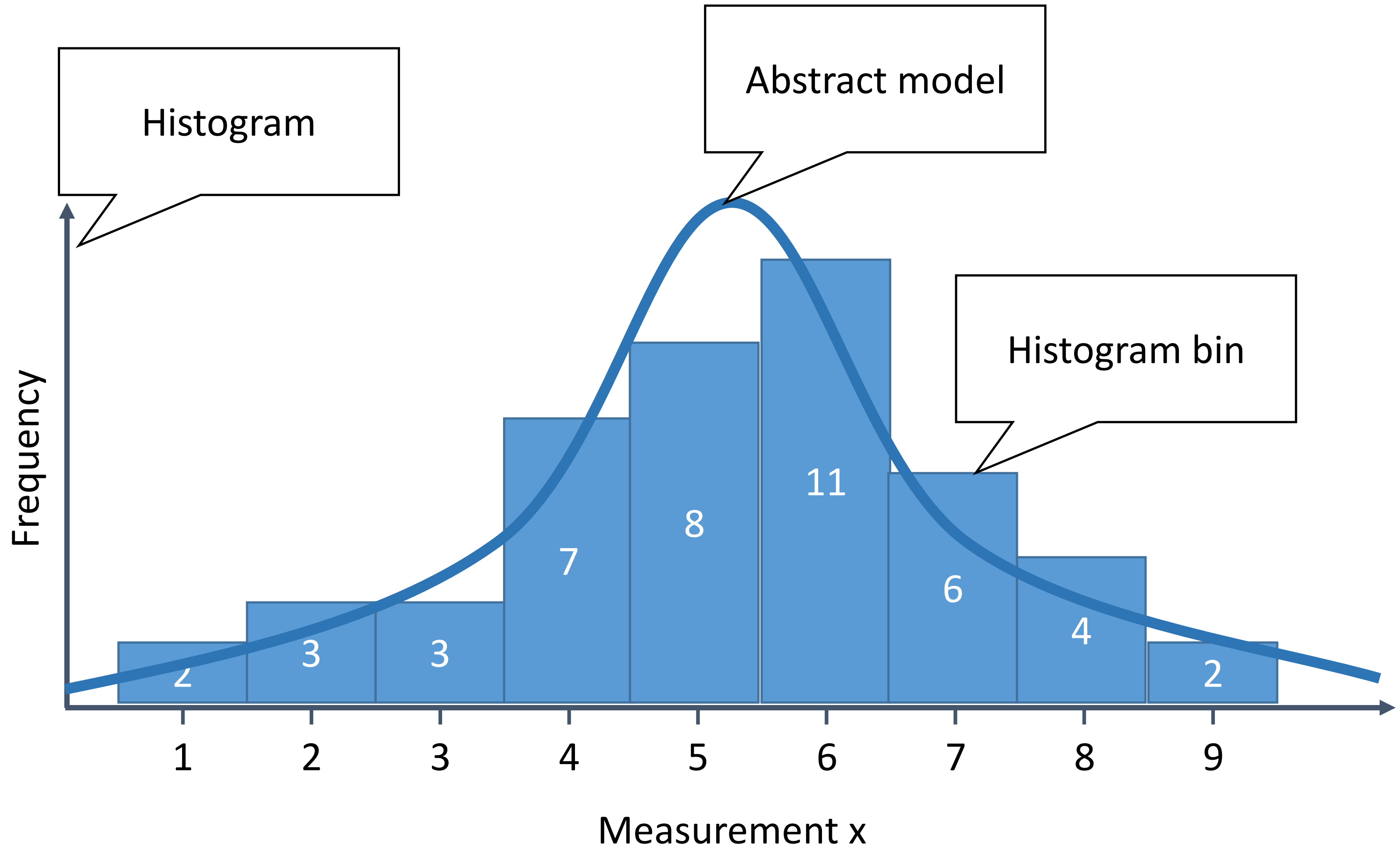
Descriptive statistics

- Repetitive experiments allow
  - a closer view
  - application of descriptive statistics



Descriptive statistics

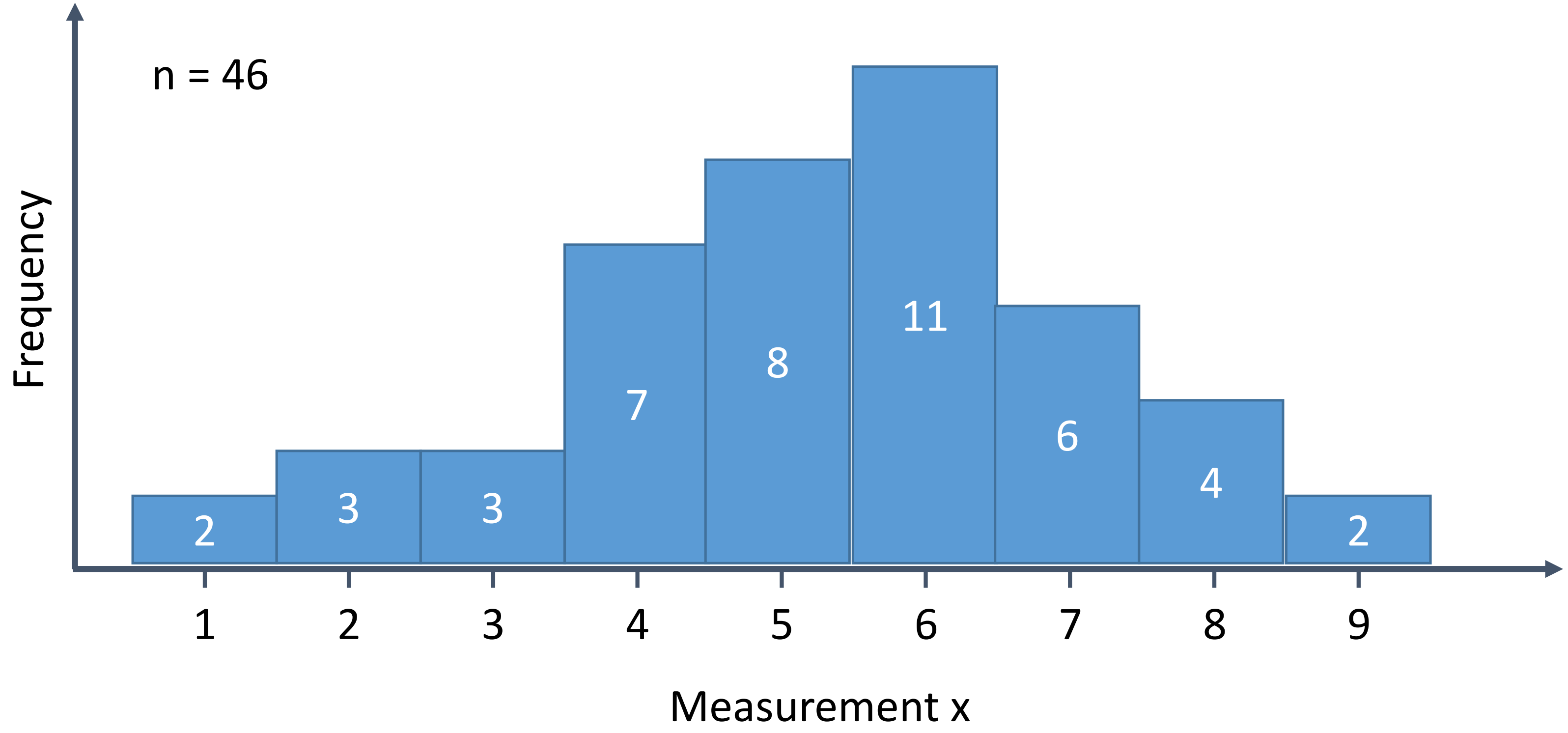
- Descriptive statistics enables summarizing data
  - Abstract models
  - Histograms



@zoccolermarcelo

Measures of central tendency

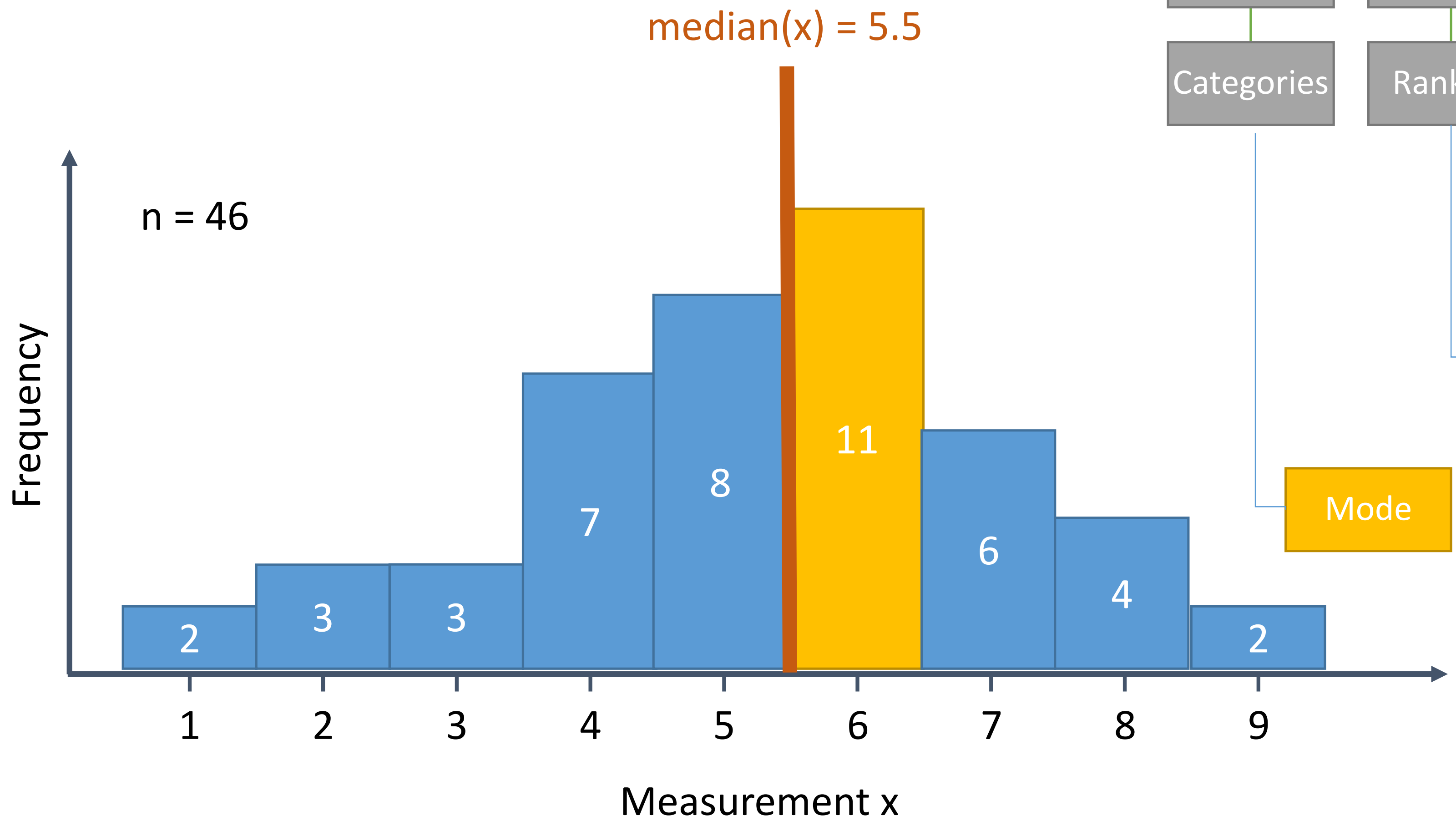
- “Where” in parameter space are my samples located?



@zoccolermarcelo

Measures of central tendency

- “Where” in parameter space are my samples located?



Measures of central tendency

Nominal	Ordinal	Interval
Categories	Ranking	On a scale
Mode	Median	Mean

Mean $\bar{x} = \sum \frac{x}{n}$

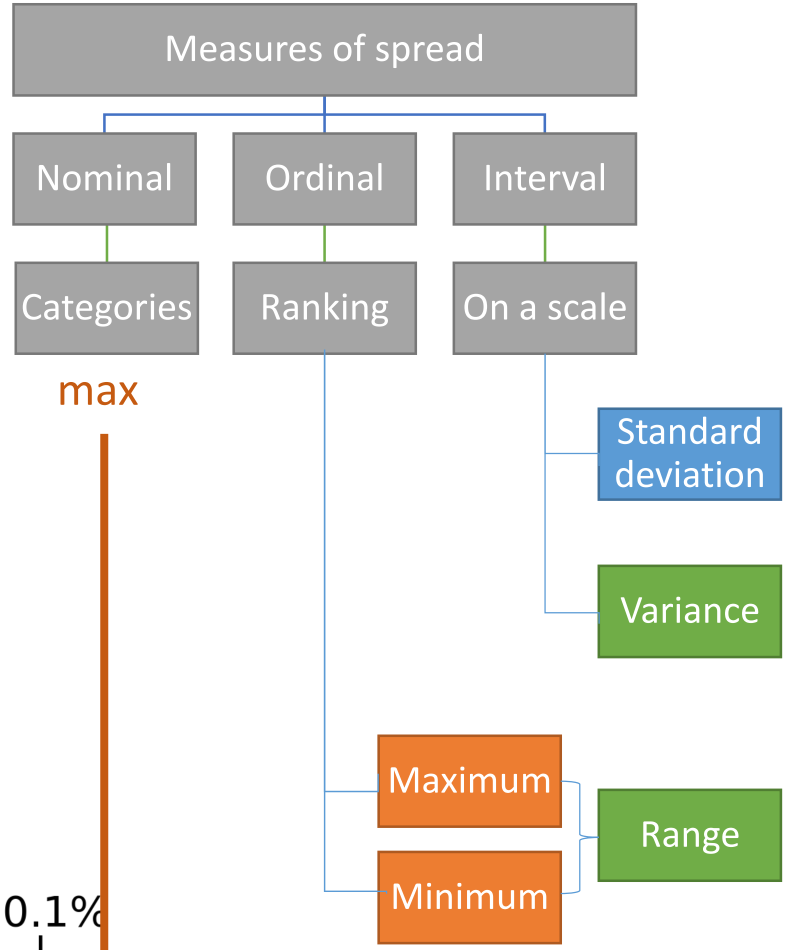
Median: x separating all samples in two groups of the same size

Mode: The most popular bin

Slide adapted from
Aldo Acevedo Toledo, Biotec, TU Dresden

Measures of spread

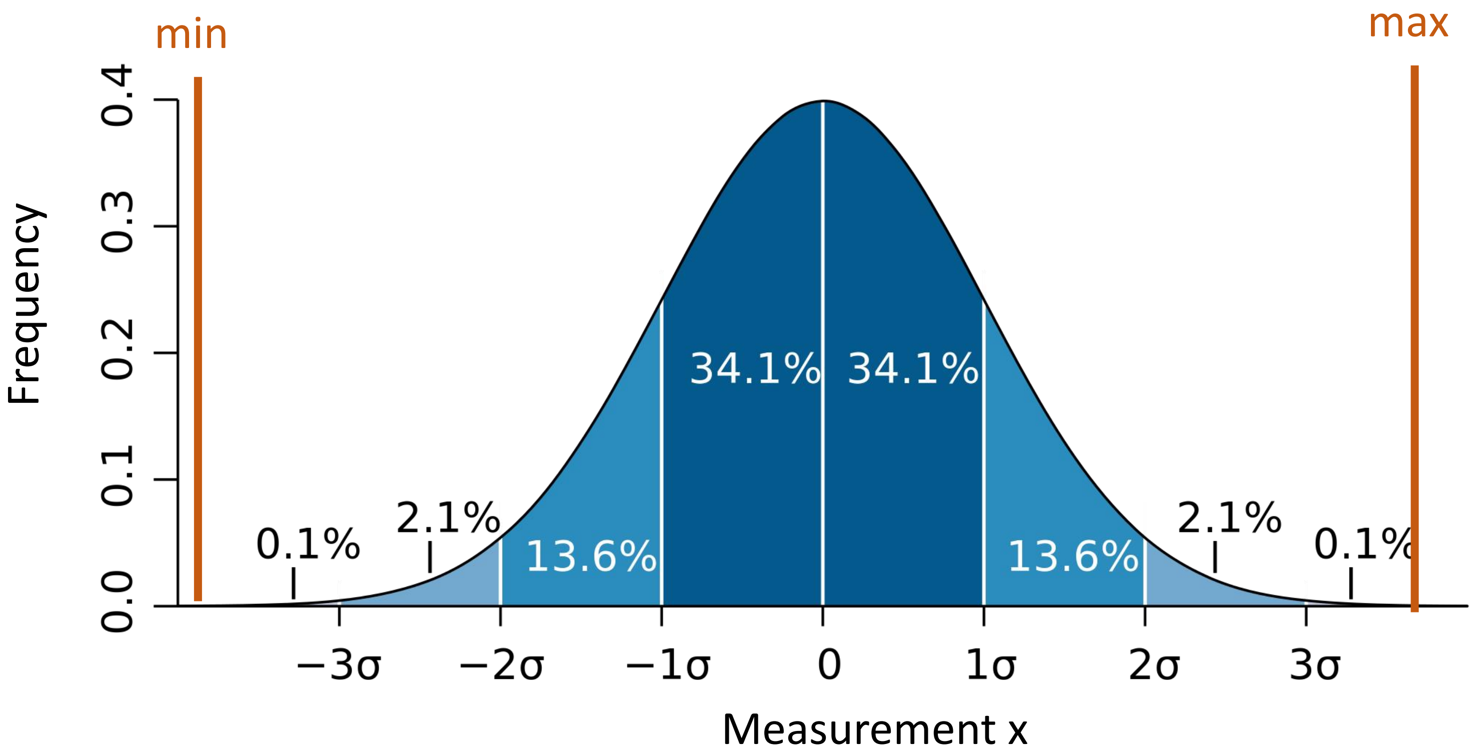
- How do my samples vary in parameters space?



$$\bar{x} = \sum \frac{x}{n}$$

$$\sigma = \sqrt{\sum \frac{(x - \bar{x})^2}{n}}$$

$$Var(x) = \sigma^2$$

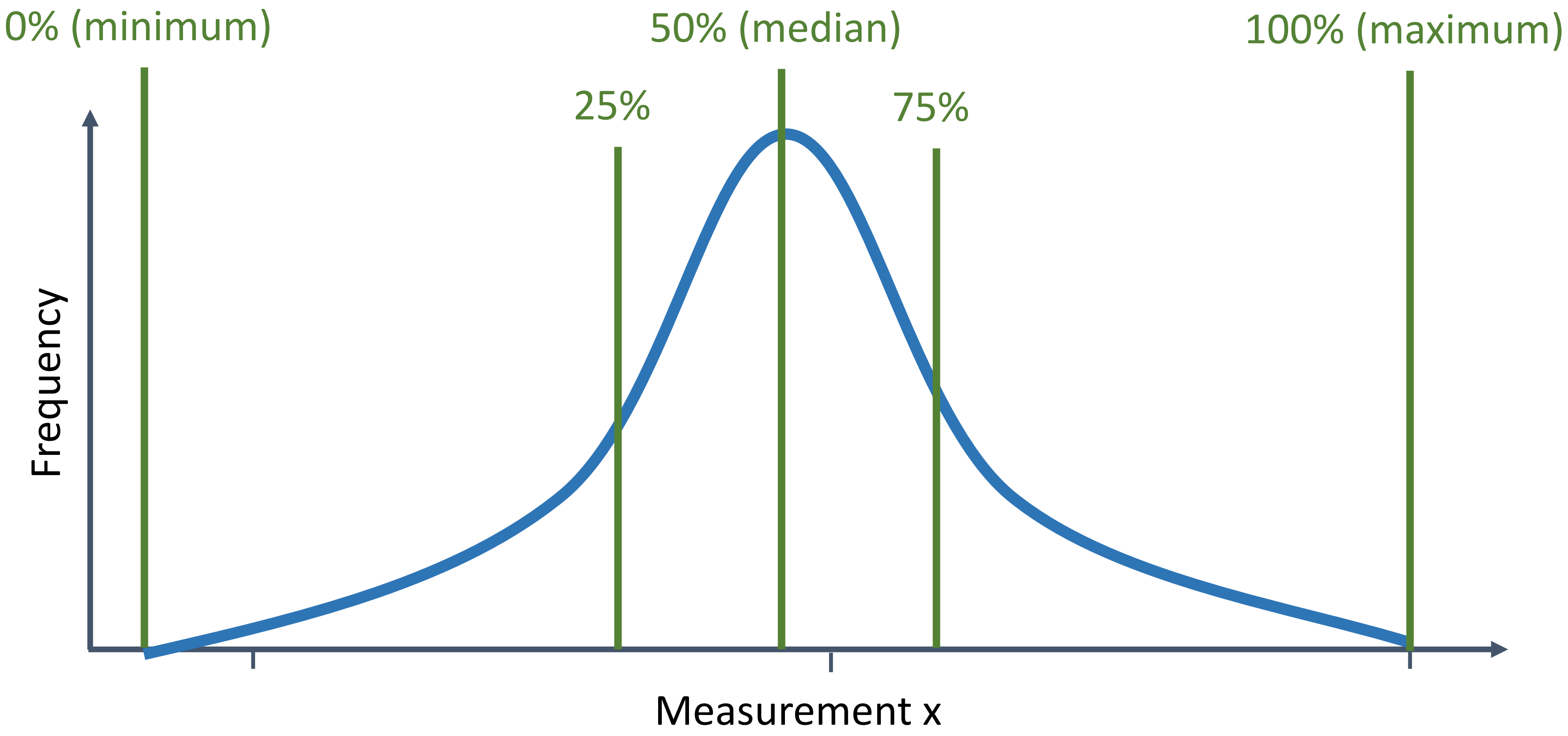


Slide adapted from: Aldo Acevedo Toledo, Biotec, TU Dresden
Graph adapted from: M. W. Toews - Own work, based (in concept) on figure by Jeremy Kemp, on 2005-02-09, CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=1903871

@zoccolermarcelo

Measures of spread

- Percentiles
  - The value under which a given percentage of our samples lie



Measures of spread

- Full width at half maximum (FWHM)