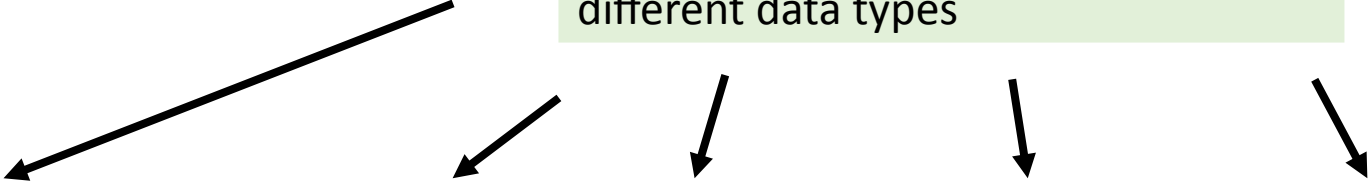


Tabular data



What is tabular data?

Unlike arrays, each column can represent another type of value, with different data types



Date (index)	Wind speed	Wind direction	Rain fall (mm)	Hours of sun
7.3.2024	7.1	N	0.0	10
8.3.2024	0.3	NW	2.1	2
9.3.2024	1.1	SE	0.3	5

Subject ID (index)	Condition ID	Presentation nr	Response time (ms)	Response
VM	732	2	28	LEFT
VM	732	3	41	RIGHT
PB	665	1	73	LEFT

What is tabular data?

Column and rows have meaningful labels (indices) that are attached to the data for each operation

Date (index)	Wind speed	Wind direction	Rain fall (mm)	Hours of sun
7.3.2024	7.1	N	0.0	10
8.3.2024	0.3	NW	2.1	2
9.3.2024	1.1	SE	0.3	5

Subject ID (index)	Condition ID	Presentation nr	Response time (ms)	Response
VM	732	2	28	LEFT
VM	732	3	41	RIGHT
PB	665	1	73	LEFT

Spreadsheets and databases rule the world!



Ariel Fischman holds the Guinness World Record for owning the most spreadsheet software (over 500!)

Many tools to handle tabular data

- Python tools
 - pandas: in-memory tabular data
 - dask: on-disk tabular data
- SQL databases
 - Optimized for retrieving rows (tree data structure for index)
 - Transactional: groups of operations are either all executed, or none
- Columnar DBs, Spark, Hadoop
 - Optimized for operations on columns
 - Ideal for data science tasks
 - Operations can be automatically distributed over multiple machines

Tabular data ideas and operations are universal for all tabular data tools

Pandas `df.groupby('condition_id')['response_time'].mean()`

dask `df.groupby('condition_id')['response_time'].mean()`

PySpark `df.groupby('condition_id').avg('response_time')`

SQL `SELECT condition_id,
 AVG(response_time) AS avg_response_time
FROM df
GROUP BY condition_id;`

TABULAR DATA OPERATIONS

Common operations on tabular data

- Tabular data has additional needs compared to arrays. Understanding how to vectorize these operations is critical for handling them
- Combine information across tables (**join, anti-join**)
 - **Join**: e.g., combine table with experiments results with table with experiments metadata (date, location, experimenter, free-form notes, ...)
 - **Anti-join**: e.g. student compiles list of outliers, exclude them from the table of experiments to analyze
- Summary tables (**split-apply-combine**)
 - E.g., compute average measurement and standard deviation by experimental condition and treatment dosage
- **Window functions** to vectorize complex computations over groups
 - E.g., compute some operation considering more than one row at the time

Joins

Join operations: combining informations from multiple tables

subject_id	condition_id	response_time	response
312	A1	0.12	LEFT
312	A2	0.37	LEFT
312	C2	0.68	LEFT
313	A1	0.07	RIGHT
313	B1	0.08	RIGHT
314	A2	0.29	LEFT
314	B1	0.14	RIGHT
314	C2	0.73	RIGHT

+

	orientation	duration	surround	stimulus_type
A1	0	0.1	FULL	LINES
A2	0	0.01	NONE	DOTS
B1	45	0.1	NONE	PLAID
B2	45	0.01	FULL	PLAID
C1	90	0.2	FULL	WIGGLES

=

subject_id	condition_id	response_time	response	orientation	duration	surround	stimulus_type
312	A1	0.12	LEFT	0.0	0.10	FULL	LINES
312	A2	0.37	LEFT	0.0	0.01	NONE	DOTS
312	C2	0.68	LEFT	NaN	NaN	NaN	NaN
313	A1	0.07	RIGHT	0.0	0.10	FULL	LINES
313	B1	0.08	RIGHT	45.0	0.10	NONE	PLAID
314	A2	0.29	LEFT	0.0	0.01	NONE	DOTS
314	B1	0.14	RIGHT	45.0	0.10	NONE	PLAID
314	C2	0.73	RIGHT	NaN	NaN	NaN	NaN

Join operations



Live Coding

notebooks/tabular_data/
020_join_operations.ipynb

Join operations – Main points

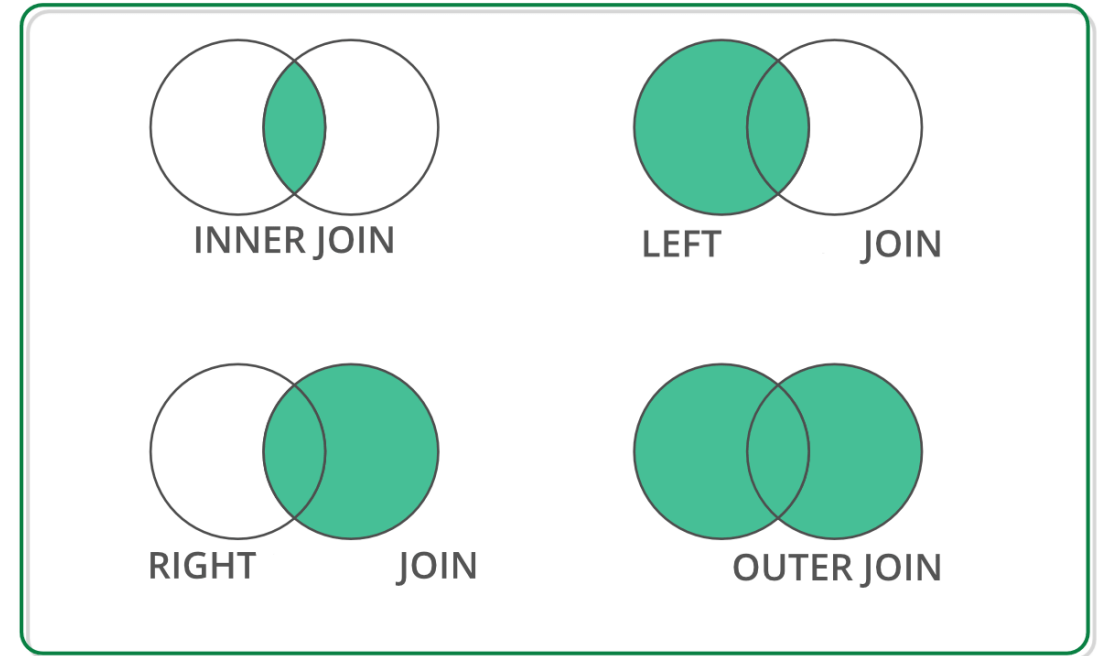


Live Coding

notebooks/tabular_data/
020_join_operations.ipynb

Join operations can be used to combine two tables using the values of one or more columns

- Different types of join:
 - left/right: keep all the column values that are present in the first/second table
 - inner: keep the column values common to both tables (intersection)
 - outer: keep all the column values of both tables
- Anti-joins can be used to exclude the values that are present in one, but not the other table (filtering based on arbitrary criteria)



Example of tabular data: “Predimed” study

- Research question: Does Mediterranean diet help prevent cardiovascular diseases, such as heart attacks and strokes?
- Longitudinal study following > 7000 Spanish participants
- Three different diets:
 - Mediterranean diet + extra virgin olive oil
 - Mediterranean diet + nuts
 - Control diet (low fat food items)



- [Estruch et al. \(2018\). New England Journal of Medicine](#)

Example of tabular data: “Predimed” study

	patient-id	location-id	sex	age	smoke	bmi	waist	wth	htn	diab	hyperchol	famhist	hormo	p14	toevent	event
0	436	4	Male	58	Former	33.53	122	0.753086	No	No	Yes	No	No	10	5.374401	Yes
1	1130	4	Male	77	Current	31.05	119	0.730061	Yes	Yes	No	No	No	10	6.097194	No
2	1131	4	Female	72	Former	30.86	106	0.654321	No	Yes	No	Yes	No	8	5.946612	No
3	1132	4	Male	71	Former	27.68	118	0.694118	Yes	No	Yes	No	No	8	2.907598	Yes
4	1111	2	Female	79	Never	35.94	129	0.806250	Yes	No	Yes	No	No	9	4.761123	No

	location-id	patient-id	group
0	2	885	MedDiet + VOO
1	1	182	MedDiet + Nuts
2	1	971	MedDiet + Nuts
3	2	691	MedDiet + Nuts
4	2	632	Control

[Estruch et al. \(2018\). New England Journal of Medicine](#)

Hands-on



Exercise

`exercises/tabular_join`

- Use joins to add the experimental information to the data (who followed which diet)
- Use anti-joins to correct for attrition (people who self-reported not following the diet)
- Submit a PR on `git.aspp.school/ASPP/2025-plovdiv-data`

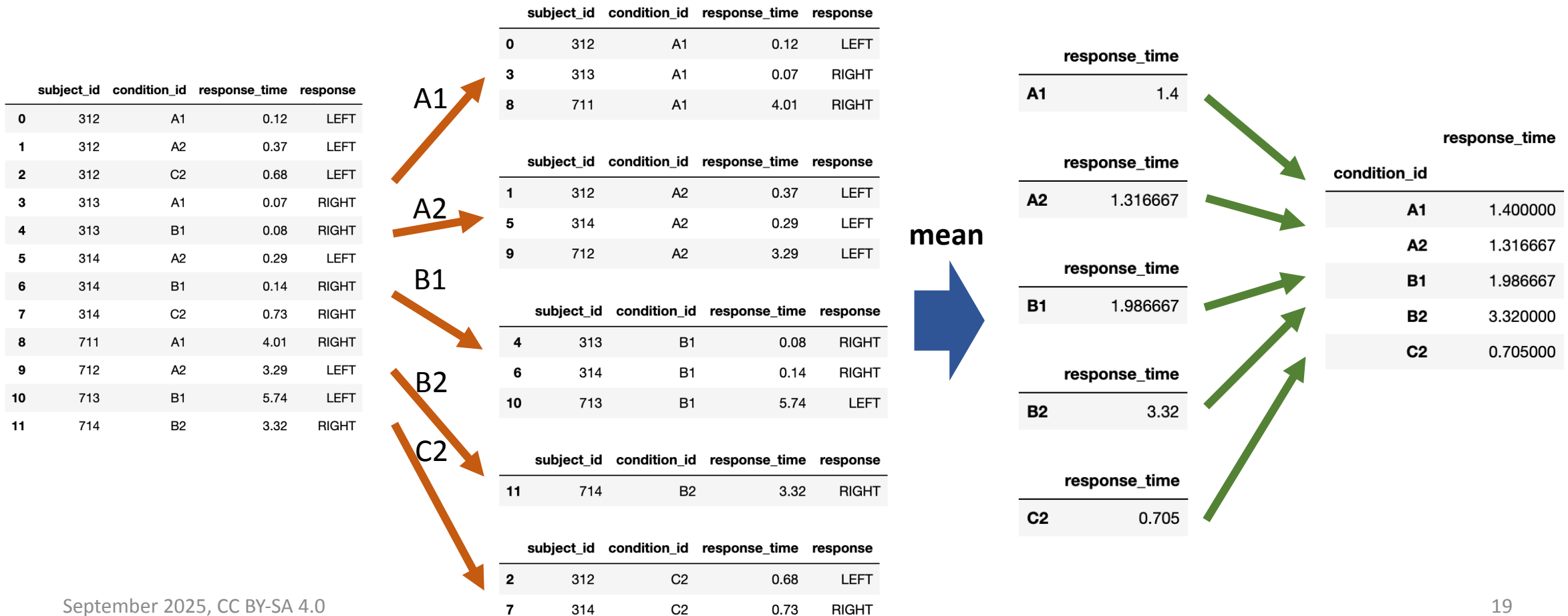
Split-apply-combine

The basic structure of most numerical analyses

split

apply

combine



Split-apply-combine operations



Live Coding

notebooks/030_tabular_data/
030_split-apply-combine.ipynb

Split-apply-combine operations



Live Coding

notebooks/030_tabular_data/
030_split-apply-combine.ipynb

Main points:

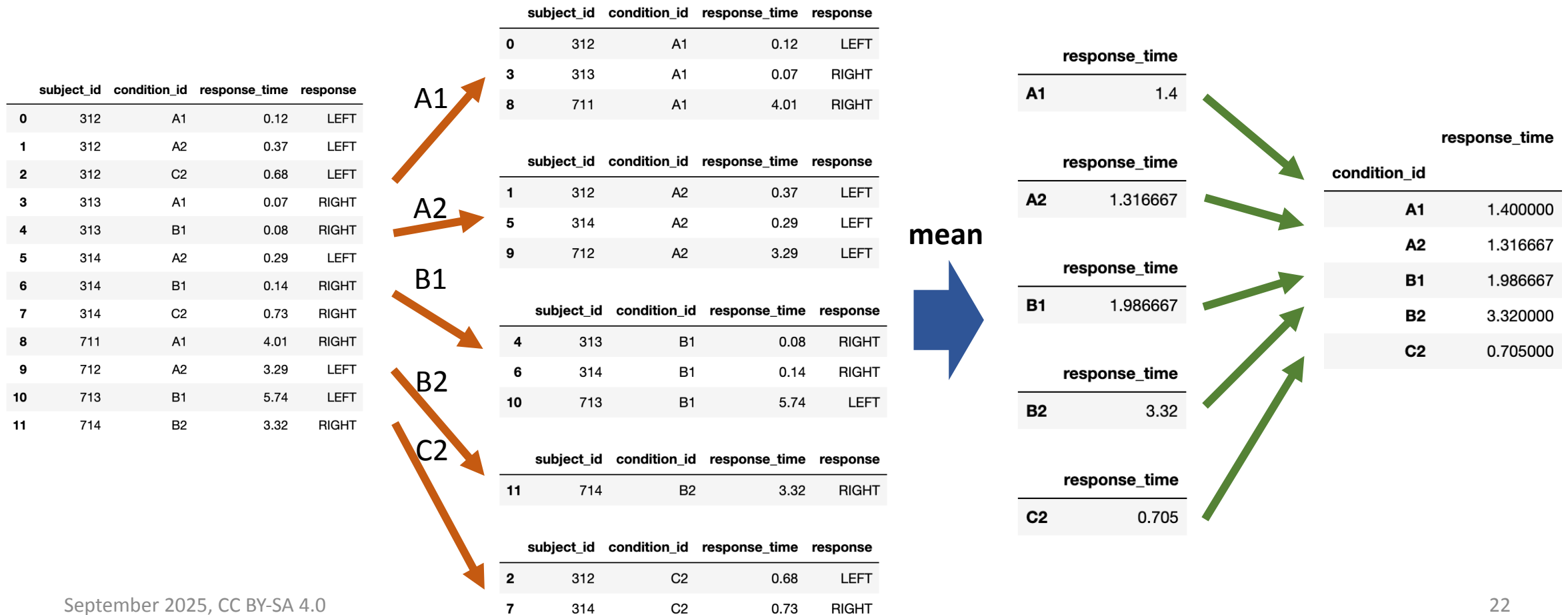
- Tabular data tools have a way to vectorize the standard split-apply-combine operations, using a “group-by” command
- In addition, Pandas has got a “pivot-table” command that can be used to simplify the creation of more complex summary tables

```
df.groupby('condition_id')['response_time'].mean()
```

split

apply

combine




```
data.pivot_table(
    index='condition_id', columns='response',
    values='response_time', aggfunc='mean',
)
```

split

apply

combine

	subject_id	condition_id	response_time	response
0	312	A1	0.12	LEFT
1	312	A2	0.37	LEFT
2	312	C2	0.68	LEFT
3	313	A1	0.07	RIGHT
4	313	B1	0.08	RIGHT
5	314	A2	0.29	LEFT
6	314	B1	0.14	RIGHT
7	314	C2	0.73	RIGHT
8	711	A1	4.01	RIGHT
9	712	A2	3.29	LEFT
10	713	B1	5.74	LEFT
11	714	B2	3.32	RIGHT



	response	LEFT	RIGHT
condition_id			
A1	0.12	2.04	
A2	1.32	NaN	
B1	5.74	0.11	
B2	NaN	3.32	
C2	0.68	0.73	

Hands-on



Exercise

exercises/
tabular_split_apply_combine

- Compute summary statistics to answer the question:
Did the Mediterranean diet help prevent cardiovascular events?
- Submit a PR on
`git.aspp.school/ASPP/2025-plovdiv-data`



- Compute some summary tables for the WHO tuberculosis data

Males 15-24 years

↙ ↘

	country	year	sp_m_014	sp_m_1524	sp_m_2534	...	sp_f_2534	sp_f_3544	sp_f_4554	sp_f_5564	sp_f_65
rownames											
5551	San Marino	2009	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
642	Belarus	2009	0.0	66.0	173.0	...	52.0	52.0	41.0	25.0	68.0
7234	Zimbabwe	2007	138.0	500.0	3693.0	...	3311.0	0.0	553.0	213.0	90.0
3471	Kuwait	2008	0.0	18.0	90.0	...	47.0	27.0	7.0	5.0	6.0
3336	Jordan	2009	1.0	5.0	15.0	...	14.0	8.0	3.0	7.0	12.0
2689	Grenada	2008	NaN	1.0	NaN	...	NaN	NaN	NaN	NaN	NaN
634	Belarus	2001	2.0	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN

Tidy Data

Same data, different organization

Which one is best for data analysis?

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

Same data, different organization
Which one is best for data analysis?

What do we want?
We want data to be in a natural format, such that data analysis is easy

John Smith
treatmenta
treatmentb

	treatmenta	treatmentb
	—	2
Jane Doe	16	11
Mary Johnson	3	1

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

Tidy data

In tidy data:

1. Each variable forms a column
2. Each observation forms a row
3. Each type of observational unit forms a table

Variables (or features, attributes)

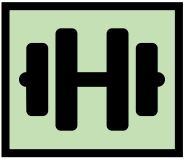
Observations
(or samples)

Subject ID	Condition ID	Trial nr	Response time (ms)	Response
VM	732	2	28	LEFT
VM	732	3	41	RIGHT
PB	665	1	73	LEFT

Variables increase when new types of measurements are introduced

Observations increase when new units (dates, subjects, ...) are measured

Hands-on



Identify variables, observations, and values.

What would a tidy version look like?

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

Table 11: Original weather dataset. There is a column for each possible day in the month. Columns d9 to d31 have been omitted to conserve space.

Hands-on



Identify variables, observations, and values.

What would a tidy version look like?

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

Table 11: Original weather dataset. There is a column for each possible day in the month. Columns d9 to d31 have been omitted to conserve space.

id	date	tmax	tmin
MX17004	2010-01-30	27.8	14.5
MX17004	2010-02-02	27.3	14.4
MX17004	2010-02-03	24.1	14.4
MX17004	2010-02-11	29.7	13.4
MX17004	2010-02-23	29.9	10.7
MX17004	2010-03-05	32.1	14.2
MX17004	2010-03-10	34.5	16.8
MX17004	2010-03-16	31.1	17.6
MX17004	2010-04-27	36.3	16.7
MX17004	2010-05-27	33.2	18.2

(b) Tidy data

Messy data

Variables are
stored in both
rows and columns

city	type	date	temperature
Bilbao	tmax	2024-07-03	34
Bilbao	tmin	2024-07-03	25
Bordeaux	tmax	2024-03-21	29
Bordeaux	tmin	2024-03-21	23
Berlin	tmax	2021-08-16	21
Berlin	tmin	2021-08-16	14
Heraklion	tmax	2021-09-01	30
Heraklion	tmin	2021-09-01	23

Column headers
are values, not
variable names

subject	date	A	B
PB	2024-07-03	0.12	0.19
VM	2024-03-21	0.37	0.41
TZ	2021-08-16	0.68	0.73
LS	2021-09-01	0.07	0.08
ZS	2023-11-11	0.08	0.16

*“Tidy datasets are all alike but
every messy dataset is messy in its
own way”*
– Hadley Wickham

Some variables
are stored in the
file names

2024-01_prices_DE.csv
2024-01_prices_FR.csv
2024-02_prices_DE.csv
2024-02_prices_FR.csv

Multiple variables
are stored in one
column

country	year	variable	cases
Angola	2000	sp_m_014	186.0
Angola	2001	sp_m_014	230.0
Angola	2002	sp_m_014	435.0
Angola	2003	sp_m_014	409.0
Angola	2004	sp_m_014	554.0
Angola	2005	sp_m_014	520.0
Angola	2006	sp_m_014	540.0

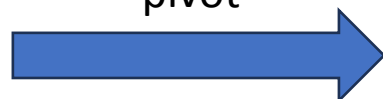
The life-changing magic of tidying up data

Pivoting – we know this one

city	type	date	temperature
Bilbao	tmax	2024-07-03	34
Bilbao	tmin	2024-07-03	25
Bordeaux	tmax	2024-03-21	29
Bordeaux	tmin	2024-03-21	23
Berlin	tmax	2021-08-16	21
Berlin	tmin	2021-08-16	14
Heraklion	tmax	2021-09-01	30
Heraklion	tmin	2021-09-01	23

The “type” column is
storing variable names

pivot



	city	date	tmax	tmin
	Berlin	2021-08-16	21	14
	Bilbao	2024-07-03	34	25
	Bordeaux	2024-03-21	29	23
	Heraklion	2021-09-01	30	23

```
df.pivot_table(  
    index=['city', 'date'], columns='type',  
    values='temperature', aggfunc='max',  
)
```



The life-changing magic of tidying up data

Melting – it's kind of the opposite of pivoting

The treatment values are stored as a column name

subject	date	A	B
PB	2024-07-03	0.12	0.19
VM	2024-03-21	0.37	0.41
TZ	2021-08-16	0.68	0.73
LS	2021-09-01	0.07	0.08
ZS	2023-11-11	0.08	0.16

melt



subject	date	variable	response_time
PB	2024-07-03	A	0.12
VM	2024-03-21	A	0.37
TZ	2021-08-16	A	0.68
LS	2021-09-01	A	0.07
ZS	2023-11-11	A	0.08
PB	2024-07-03	B	0.19
VM	2024-03-21	B	0.41
TZ	2021-08-16	B	0.73
LS	2021-09-01	B	0.08
ZS	2023-11-11	B	0.16



```
pd.melt(data, id_vars=['subject', 'date'], value_name='response_time')
```

Split the columns in (A) “id_vars” and (B) non-“id_vars”. The column names in (B) are used as new values in a new column “variable”. The values in columns (B) go into a new column, “response_time”.

The life-changing magic of tidying up data

`pd.concat` – add together tables with the same variables (columns)

Some variables
are stored in the
file names

```
2024-01_prices_DE.csv  
2024-01_prices_FR.csv  
2024-02_prices_DE.csv  
2024-02_prices_FR.csv
```

```
tables = []  
for filename in filenames:  
    # Parse filename  
    year_month, _, country = filename[:-4].split('_')  
    # Read table and add columns for the variables  
    df = pd.read_csv(filename)  
    # Add the variables that were in the filename  
    df['year_month'] = year_month  
    df['country'] = country  
    # Store table  
    tables.append(df)  
  
# Create complete table  
tidy_df = pd.concat(tables)
```



Hands-on



Exercise

exercises/tabular_tidy_data

- Tidy up the data set in the tuberculosis exercise and compute the summary stats
- Submit a PR on `git.aspp.school/ASPP/2025-plovdiv-data`

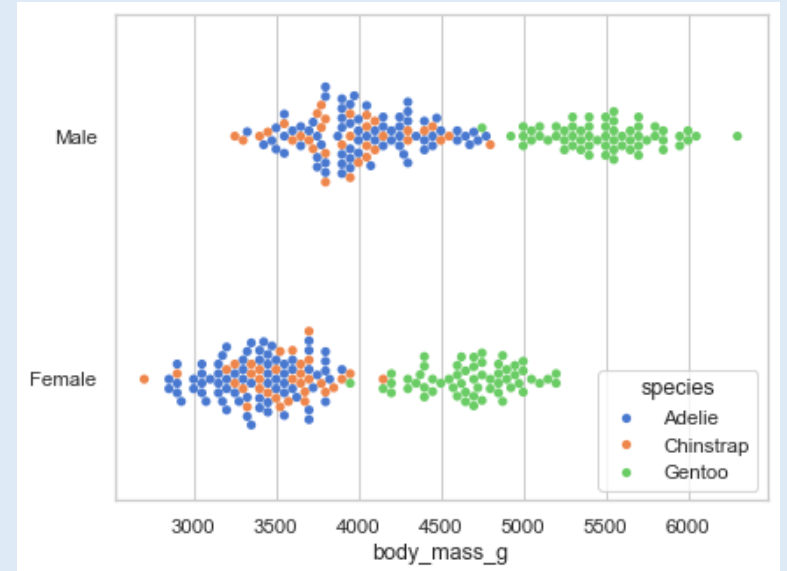
Multiple variables
are stored in one
column

country	year	variable	cases
Angola	2000	sp_m_014	186.0
Angola	2001	sp_m_014	230.0
Angola	2002	sp_m_014	435.0
Angola	2003	sp_m_014	409.0
Angola	2004	sp_m_014	554.0
Angola	2005	sp_m_014	520.0
Angola	2006	sp_m_014	540.0

Why is tidy data good?

- Many analyses require a simple sequence of steps:
 - Filter by individual variables to discard data that is not needed
 - Group and summarize
 - Re-arrange (e.g. sort)
 - Visualize
- Joining tidy tables is easy!
- One can write generic code that takes tidy data as input.
For example, **seaborn** relies on tidy data to make complex plots

```
sns.swarmplot(  
    data=df,  
    x="body_mass_g",  
    y="sex",  
    hue="species",  
)
```



Window functions

Window functions: grouped row-by-row operations

- “Window functions” are a kind of split-apply-combine operation, but instead of aggregating the data in a group and returning one value per group, they return one value per row
- Examples: ranking all entries in a group; computing the distance between timestamps per group; number the rows by group in chronological order
- In Pandas, most of these operations can be performed with a combination of sorting and grouping-by

Window functions



Live Coding

notebooks/tabular_data/
040_window_functions.ipynb

Window functions



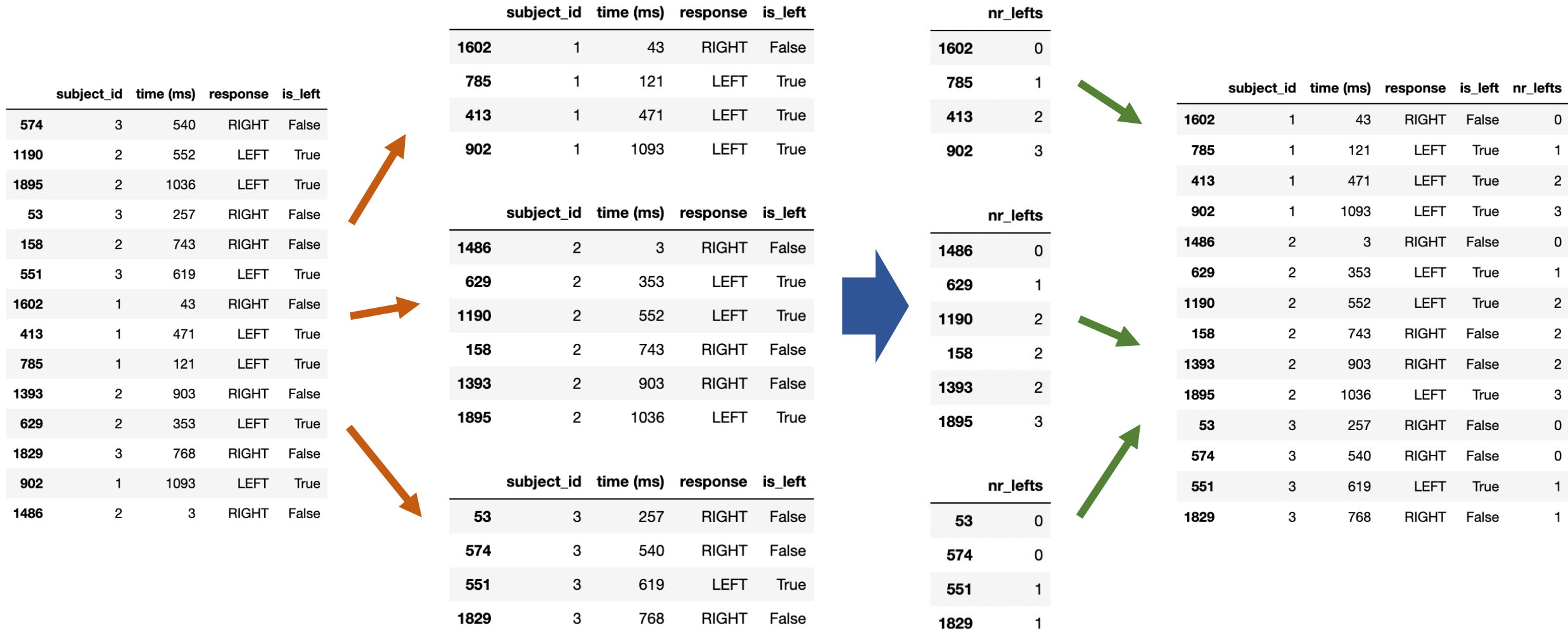
Live Coding

notebooks/tabular_data/
040_window_functions.ipynb

- Main points:
 - Window functions perform row-by-row operations on grouped data
 - They are an advanced way of avoiding for loops with tabular data
 - In Pandas, they can be achieved with a combo of sorting and grouping-by

Window functions operations

```
df['nr_lefts'] = df.sort_values('time (ms)').groupby('subject_id')['is_left'].cumsum()
```



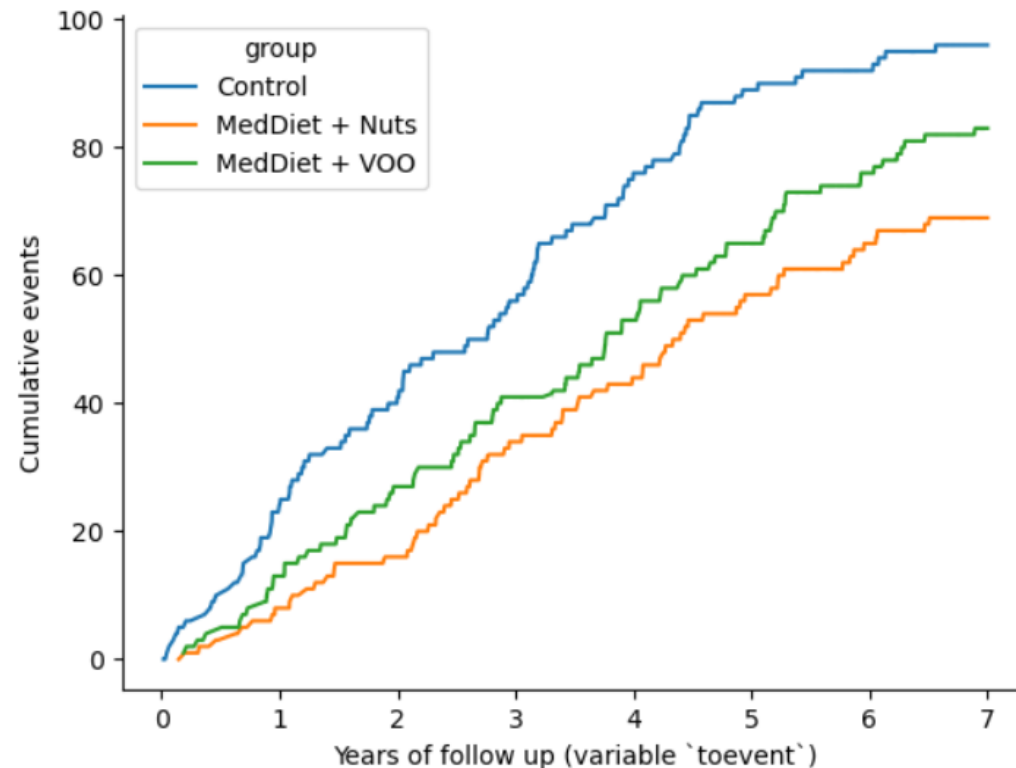
Hands-on



Exercise

exercises/
tabular_window_functions

- Compute the cumulative number of cases across time, per diet group
- Submit a PR on `git.aspp.school/ASPP/2025-plovdiv-data`

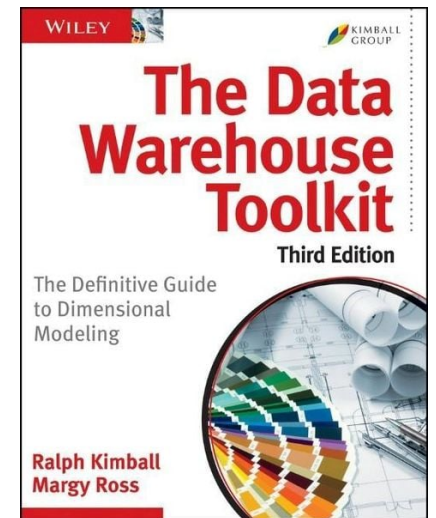
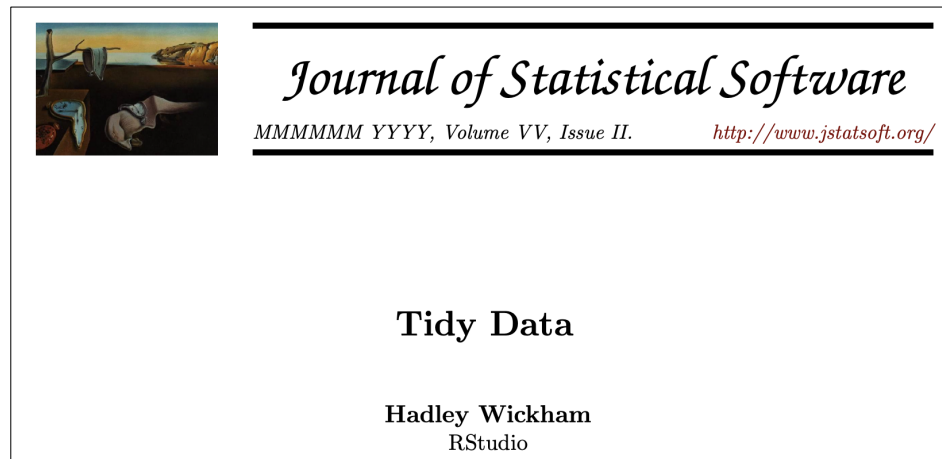
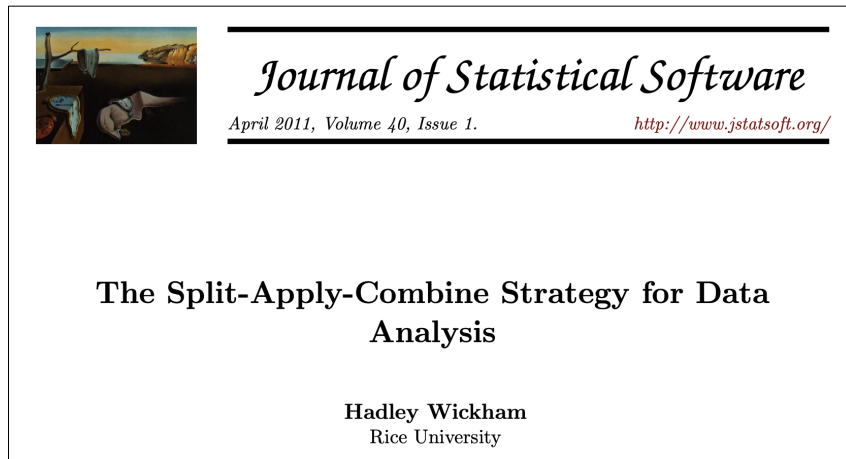


Global summary

- There are many different data structures, each specialized in efficiently processing one type of data
- Code performance grows differently with data size: Big-O
- NumPy array efficiently store data in a C-native memory block, interpreted as an array using some metadata
- NumPy operations that only need to change the metadata do so, creating a view of the same memory block. These operations are $O(1)$!
- Tabular data can also be vectorized using joins, anti-joins, split-apply-combine operations, and window functions
- For these operations to be efficient and painless, data should be stored in a tidy data format

What we didn't talk about

- Other data structures: graphs, trees, priority queues, ...
- Options for working with large data on disk / remotely (instead of in-memory)
- Best practices in data handling: versioning, lineage, sharing
- Organizing a complex data set in multiple tables
- ... and a lot more!



Thank you!



Data organization

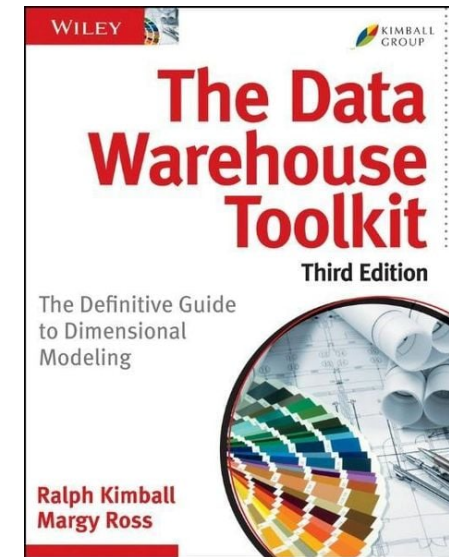
- Data organization concepts:
 - tidy data
 - normalized data (star organization)
 - data science friendly data (denormalized)

Organizing multiple tables

- Dimension vs fact tables
- De-normalization (but for data analysis flat tables are more convenient)

id	artist	track	time
1	2 Pac	Baby Don't Cry	4:22
2	2Ge+her	The Hardest Part Of ...	3:15
3	3 Doors Down	Kryptonite	3:53
4	3 Doors Down	Loser	4:24
5	504 Boyz	Wobble Wobble	3:35
6	98~0	Give Me Just One Nig...	3:24
7	A*Teens	Dancing Queen	3:44
8	Aaliyah	I Don't Wanna	4:15
9	Aaliyah	Try Again	4:03
10	Adams, Yolanda	Open My Heart	5:30
11	Adkins, Trace	More	3:05
12	Aguilera, Christina	Come On Over Baby	3:38
13	Aguilera, Christina	I Turn To You	4:00
14	Aguilera, Christina	What A Girl Wants	3:18
15	Alice DeeJay	Better Off Alone	6:50

id	date	rank
1	2000-02-26	87
1	2000-03-04	82
1	2000-03-11	72
1	2000-03-18	77
1	2000-03-25	87
1	2000-04-01	94
1	2000-04-08	99
2	2000-09-02	91
2	2000-09-09	87
2	2000-09-16	92
3	2000-04-08	81
3	2000-04-15	70
3	2000-04-22	68
3	2000-04-29	67
3	2000-05-06	66



Dealing with changes in the data

- Recommendations:
 - NEVER overwrite a data file. Treat data files as immutable
 - Use versioning for changes in the data file, and load the latest version for new analyses, old versions to reproduce previous results
 - (pond is a library I'm working on to automatize this process)
- Like in computer code:
 - Adding new columns / rows is generally ok
 - Deleting/changing a column is not! Code will break! Add a new column instead