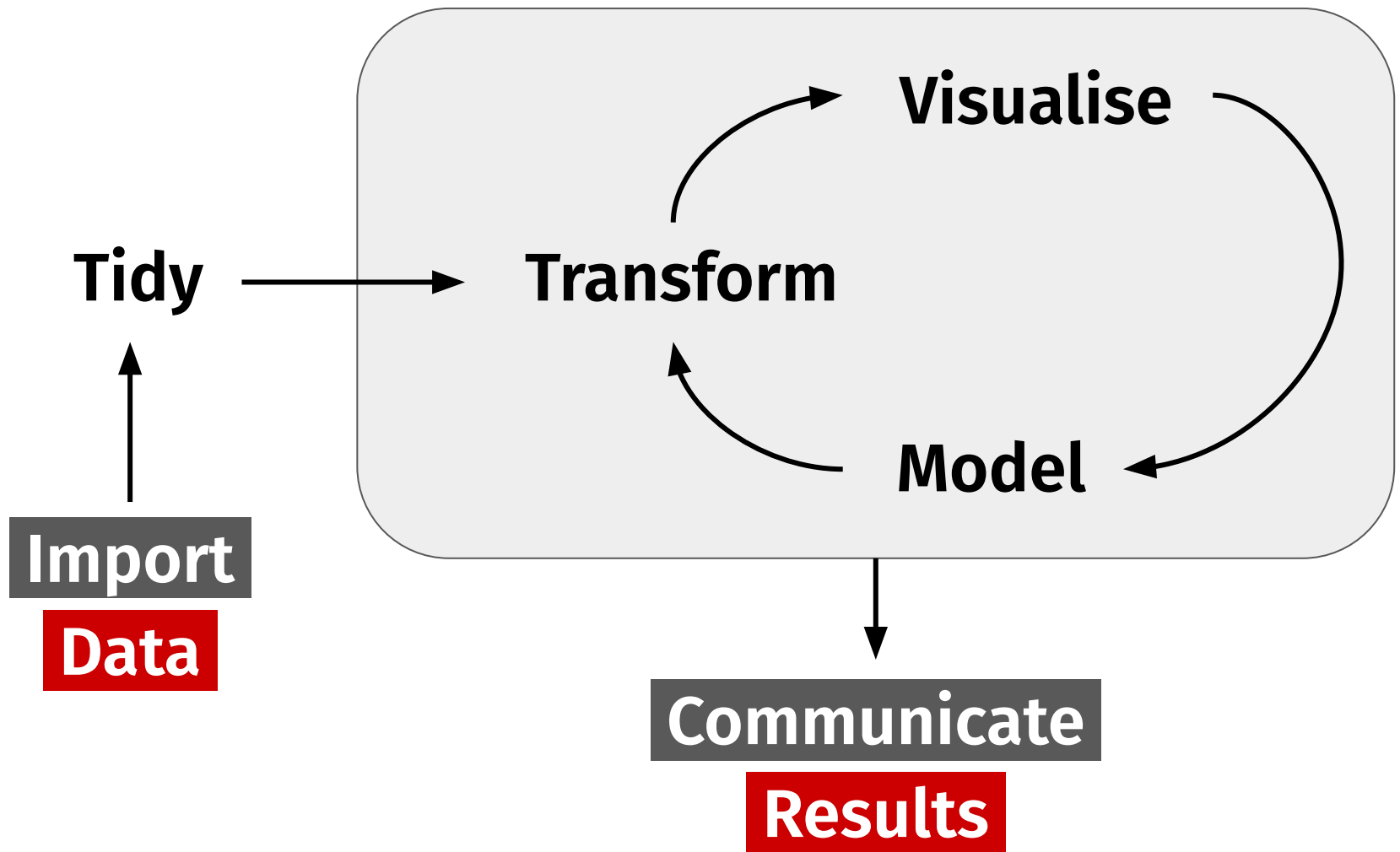





Data Management

QUANTI 2 · Session 4

François Briatte



Source Grolemund and Wickham

A woman with dark hair and a surprised expression is sitting on a stone ledge in a jungle. She is holding a long wooden staff. The background is filled with lush green foliage and a stone wall. Overlaid on the image are three black text boxes with white text.

90% of data analysis
is data cleaning

(and merging datasets)

Data cleaning a.k.a. messy inputs

e.g. removing extra characters, filtering out undesired (incomplete, corrupt...) data rows

e.g. open-ended fields (locations, professions, religions...) in survey data, commercial databases

Shown right: dataset from Kazakhstan containing self-reported knowledge of software by job-seekers

Source: [Sergiy Radyakin](#)

Photoshop	Lotus Notes	Corel Draw
photo shop	lotusnotes	carel draw
photo-shop	lotus-notes	orel draw
foto shop	lotusnotus	coreldraw
fotoshop	lotus notus	corldraw
foto-shop	lotus-notus	corel-draw
photo shop	lotes-notes	corel draw
photo-shop	lotesnotes	coral draw
photoshop	lotes notes	corral draw

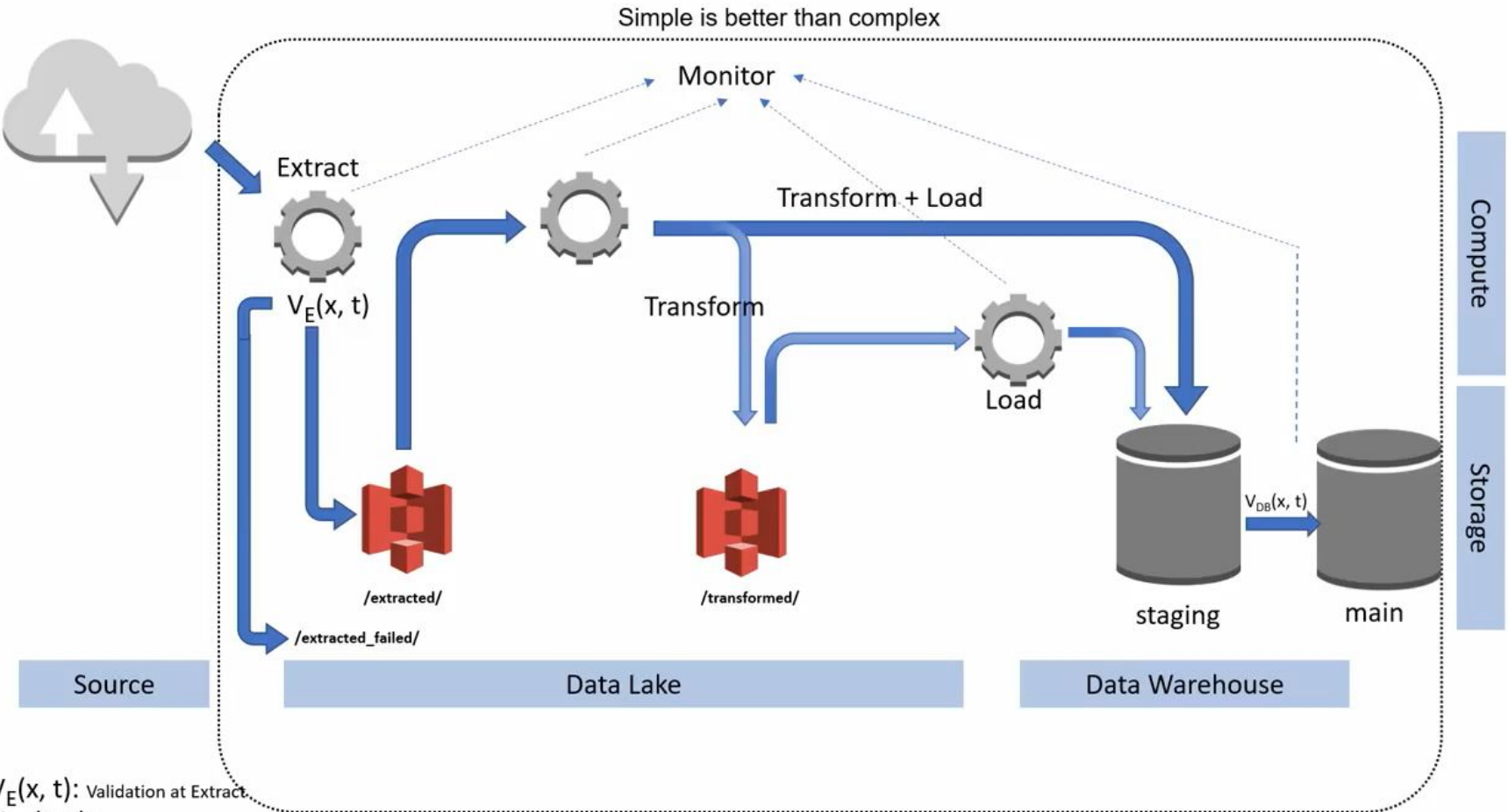
Data management beyond cleaning

- **Data I/O** Reading/Writing various inputs/outputs
- **Reshaping** Transforming the dataset structure
- **Recoding** Transforming values (e.g. regrouping)
- **Subsetting** Filtering out observations from the dataset
- **Aggregating** Summarising values by groups (categories)

More complex operations (e.g. real-time, distributed data)

~ **Data Integration/Engineering**

Extract, Transform, Load (ETL)



$V_E(x, t)$: Validation at Extract

$V_{DB}(x, t)$: Validation at DB

How it fits in job taxonomies

Data Engineer vs Data Scientist

Data Engineer

- Develop scalable data architecture
- Streamline data acquisition
- Set up processes to bring together data
- Clean corrupt data
- Well versed in cloud technology

Data Scientist

- Mining data for patterns
- Statistical modeling
- Predictive models using machine learning
- Monitor business processes
- Clean outliers in data

Relevant professional skill set



- **Relational Data Base/Stream Management Systems (RDBMS/RDSMS) for data **warehousing****
e.g. **SQL** and variants, **Amazon RedShift**, **Apache Kafka**
involves learning data models/**schemas** and new **DSLs**
- **Building fast, reproducible data **pipelines**/streams**
required for (scientific, legal) reproducibility
might involve using some form of **version control**
might involve **thinking of data like we think of code**
e.g. **idempotency**, data partitions as **immutable** objects

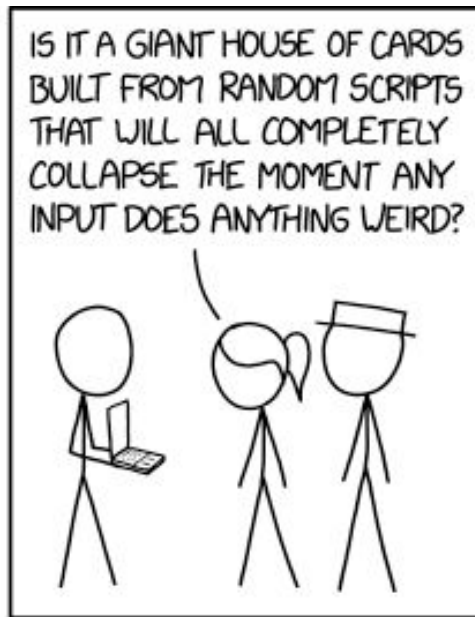
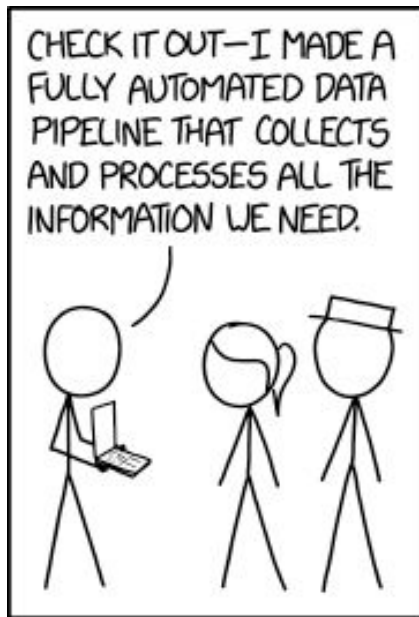
More prosaically for us

- **Accessing** the data (sources, datasets)
- **Understanding** the data (codebooks, variables/values)
- **Reading/Writing** the data (formats, object types)
- **Manipulating** the data (basic principles, packages)
- **Coding** as much of the entire operation as possible

- Accepting that data input and coding are often **messy**
- Expecting that **things will break** badly and often

condom use is astoundingly high, close to 85 per cent. This is worth passing on to my colleagues over beer and jazz. I'm already out of my chair, just about to shut down my computer, when my eye falls on the scrappy code sheet. Wait, these data are raw, they are still coded 1 = yes, 2 = no.

When my data are nice and tidy, with the standard 0 = no, 1 = yes codes, my 'yes' row is always at the bottom of the table, so that's where my key percentage is. Because my data weren't coded, the results were 'upside down' for me, and I had read them wrong. Not 20 per cent buying sex but 80 per cent—the highest we have ever seen anywhere in the world. Not 85 per cent using condoms but 85 per cent having unprotected sex—right at the bottom of the global league table.* I slumped back down into my chair, my head in my hands. I admit I wanted some bad news. But I never conceived of anything this bad. Be careful what you wish for.



related: [“Covid: how Excel may have caused loss of 16,000 test results in England”](#)

Dataset/file formats

[< Zurück](#)

European Parliament Election Study 2019, Voter Study

[Schmitt, Hermann](#); [Hobolt, Sara B.](#); [van der Brug, Wouter](#) 

GESIS Datenarchiv, Köln. ZA7581 Datenfile Version 1.0.0, <https://doi.org/10.4232/1.13473>

Primärforscher/ Wissenschaftlicher Beirat, Institution: Schmitt, Hermann - MZES, University of Mannheim | Hobolt, Sara B. - London School of Economics | van der Brug, Wouter - University of Amsterdam | Popa, Sebastian Adrian - Newcastle University & MZES, University of Mannheim

Herausgeber: GESIS Data Archive

Studiennummer: ZA7581

Aktuelle Version: 1.0.0, 2020-05-06, <https://doi.org/10.4232/1.13473>

DOI: 10.4232/1.13473

Publikationsjahr: 2020

Erhebungszeitraum: 14.06.2019 - 07.11.2019

[Date](#)[Frag](#)[And](#)[Zitie](#)

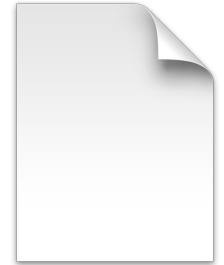
What's new in Twitter API v2

- Condensed default Tweet and user objects
- A fields query parameter to request desired object fields
- Poll, place, media metadata available to request within the Tweet payload
- [Metrics](#) and [annotations](#) available in the Tweet payload
- A new `conversation_id` field to help you track Tweets included in a conversation

The default [Tweet object](#) that is delivered with our endpoints is limited to just the `id` and `text` fields.

```
1  {
2    "data": {
3      "id": "1212092627178287104",
4      "text": "These launches would not be possible without the fe
```

Kinds of data structures



- Rectangular a.k.a. 'flat' **tabular** datasets
 - CSV / TSV** — comma / tab -separated (**not that great**)
 - XLSX** — recent Microsoft Excel format (open)
 - XLS** — old Microsoft Excel format **closed, avoid**
- Semantic a.k.a. markup languages
 - HTML / XML** — `<attribute>value</attribute>`
 - JSON** — `"basics" : { "sex" : "F", "age" : 20 }`
- Databases (DBs) — *see end of slides*

Version control on a **plain text** dataset

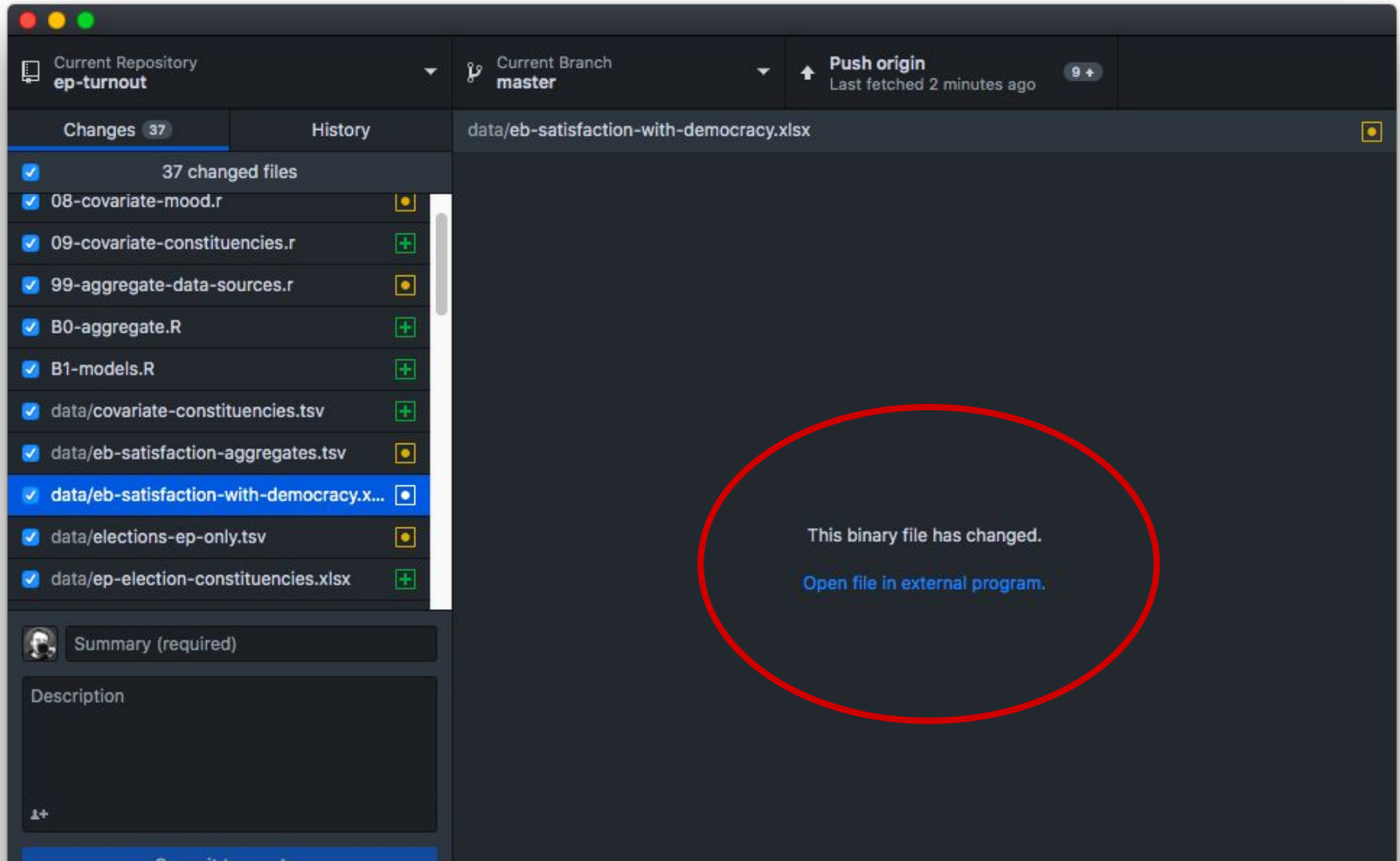
The screenshot shows a version control interface with the following components:

- Current Repository:** ep-turnout
- Current Branch:** master
- Push origin:** Never fetched
- Changes:** 37 changed files
- History:** A list of files with checkboxes and icons for each.
- Diff View:** A table showing the changes for the selected file, `data/eb-satisfaction-aggregates.tsv`.

The diff view table is as follows:

Line	Line	Country	Year	Count	Hash
132	144	LVA	2014	7	0.3677700114620074
133	145	LVA	2019	10	0.5108114390999395
	146	+MLT	1979	1	0.6620278330019881
134	147	MLT	2004	0	NA
135	148	MLT	2009	4	0.49125948103792416
136	149	MLT	2014	7	0.5511228170626968
@@ -140,10 +153,11 @@					
		NLD	1984	8	0.5567897081083634
140	153	NLD	1989	10	0.6096628620401133
141	154	NLD	1994	10	0.6895644973331541
142	155	NLD	1999	2	0.7119350135974796
143		-NLD	2004	8	0.694861220970008
	156	+NLD	2004	9	0.7116631332369571
144	157	NLD	2009	4	0.7429051935831186
145	158	NLD	2014	7	0.7416267765711541
146	159	NLD	2019	10	0.7839015732685193
	160	+POL	1987	1	0.6599999999999999
147	161	POL	2004	0	NA
148	162	POL	2009	4	0.36269194194194193
149	163	POL	2014	7	0.5070740687883545
@@ -153,17 +167,20 @@					
		PRT	1989	4	0.5638926426426426
153	167	PRT	1994	10	0.6382052554052554
154	168	PRT	1999	2	0.522032155490322
155	169	PRT	2004	8	0.4049642187974381
	170	+PRT	2007	1	0.6785714285714286
156	171	PRT	2009	4	0.3688137688137688

Less control on **binary files** e.g. Excel, Stata, SPSS



Memo to Reinhart and Rogoff: I think it's best to admit your errors and go on from there

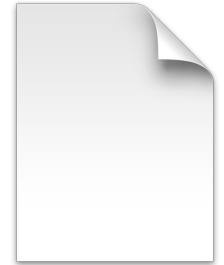
Posted by Andrew on 16 April 2013, 10:53 pm

	B	C	I	J	K	L	M
2			Real GDP growth				
3			Debt/GDP				
4	Country	Coverage	30 or less	30 to 60	60 to 90	90 or above	30 or less
26			3.7	3.0	3.5	1.7	5.5
27	Minimum		1.6	0.3	1.3	1.8	0.8
28	Maximum		5.4	4.9	10.2	3.6	13.3
29							
30	US	1946-2009	n.a.	3.4	3.3	-2.0	n.a.
31	UK	1946-2009	n.a.	2.4	2.5	2.4	n.a.
32	Sweden	1946-2009	3.6	2.9	2.7	n.a.	6.3
33	Spain	1946-2009	1.5	3.4	4.2	n.a.	9.9
34	Portugal	1952-2009	4.8	2.5	0.3	n.a.	7.9
35	New Zealand	1948-2009	2.5	2.9	3.9	-7.9	2.6
36	Netherlands	1956-2009	4.1	2.7	1.1	n.a.	6.4
37	Norway	1947-2009	3.4	5.1	n.a.	n.a.	5.4
38	Japan	1946-2009	7.0	4.0	1.0	0.7	7.0
39	Italy	1951-2009	5.4	2.1	1.8	1.0	5.6
40	Ireland	1948-2009	4.4	4.5	4.0	2.4	2.9
41	Greece	1970-2009	4.0	0.3	2.7	2.9	13.3
42	Germany	1946-2009	3.9	0.9	n.a.	n.a.	3.2
43	France	1949-2009	4.9	2.7	3.0	n.a.	5.2
44	Finland	1946-2009	3.8	2.4	2.5	n.a.	7.0
45	Denmark	1950-2009	3.5	1.7	2.3	n.a.	5.6
46	Canada	1951-2009	1.9	3.6	4.1	n.a.	2.2
47	Belgium	1947-2009	n.a.	4.2	3.1	2.6	n.a.
48	Austria	1948-2009	5.2	3.3	-3.8	n.a.	5.7
49	Australia	1951-2009	3.2	4.9	4.0	n.a.	5.9

another issue

with spreadsheet data – errors in user-invisible manipulations

Other things to consider



- **Common closed formats** used in social science
 - DTA (Stata) — many versions, non-retrocompatible
 - SAV / POR (SPSS) — common for surveys (labelled data)
- **Encoding issues** e.g. accents showing up as ❖
 - UTF-8 — gold standard, successor of...
 - ASCII, ISO-8859-1 (Latin-1) — oldschool **avoid**
- **APIs**, like Web scraping, require writing up queries, with e.g. SPARQL, and learning how to read the 'response'

Tidy data principles



R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

Overview

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

dplyr provides [alternatives to base R functions that apply to data frames](#)
it relies on the [tibble](#) format, which are enhanced data frames

Tidy-data-oriented packages

See the [tidyverse](#) series, which also includes [ggplot2](#)

- **Data I/O** with [readr](#), [readxl](#), [haven](#)
 - Used internally by [rio](#)
 - Fall back to [foreign](#) or other packages if need be
- **Dataset manipulation** with [dplyr](#)
 - Related dataset object type: [tibble](#)
 - Similar object type for time series: [tsibble](#)
- **Model results** manipulation with [broom](#) and [tidymodels](#)

Like families, tidy datasets are all alike but every messy dataset is messy in its own way.

Hadley Wickham

country	year	cases	population
Afghanistan	1999	3675	19987071
Afghanistan	2000	3666	20095360
Brazil	1999	36737	172006362
Brazil	2000	80488	174004898
China	1999	212258	1272915272
China	2000	213766	128042583

variables

country	year	cases	population
Afghanistan	1999	3675	19987071
Afghanistan	2000	3666	20095360
Brazil	1999	36737	172006362
Brazil	2000	80488	174004898
China	1999	212258	1272915272
China	2000	213766	128042583

observations

country	year	cases	population
Afghanistan	99	75	987071
Afghanistan	00	66	095360
Brazil	99	737	006362
Brazil	00	488	004898
China	99	258	915272
China	00	766	42583

values

In a tidy data set:



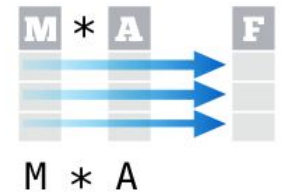
Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



This format also allows grouped operations via **split-apply-combine**

'Long' v. 'wide' data

Country	gdp1980	gdp1990	gdp2000	...
AA	100	107	80	...
BB	100	NA	NA	...
CC	100	102	NA	...
DD	100	103	105	...

Country	Variable	Year	Value
AA	GDP	1980	100
AA	GDP	1990	107
AA	GDP	2000	80
BB	GDP	1980	100
BB	GDP	1990	NA
BB	GDP	2000	NA
CC	GDP	1980	100
CC	GDP	1990	102
CC	GDP	2000	NA
DD	GDP	1980	100
DD	GDP	1990	103
DD	GDP	2000	105
...

Above: **wide format**, often found in official datasets (e.g. OECD time series)

Left: **long format**, less human-readable but much easier to manipulate

Fictional data representing relative GDP growth rates, base year 1980

Reshaping (pivoting)

The diagram illustrates the process of pivoting a wide table into a long table. The wide table on the right has columns for 'country', '1999', and '2000'. The long table on the left has columns for 'country', 'year', and 'cases'. Arrows show that the 'country' column in the long table is populated from the 'country' column of the wide table, the 'year' column is populated from the '1999' and '2000' columns, and the 'cases' column is populated from the corresponding values in the '1999' and '2000' columns.


country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

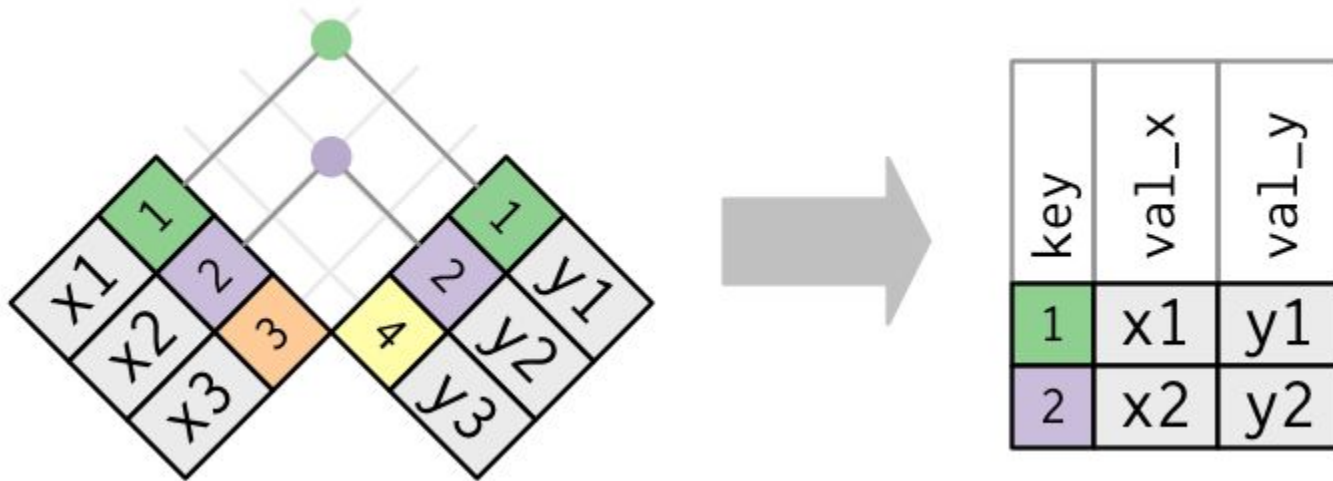
Splitting (separating)

country	year	rate
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

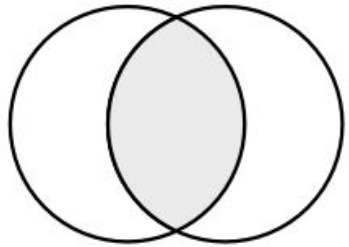


country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

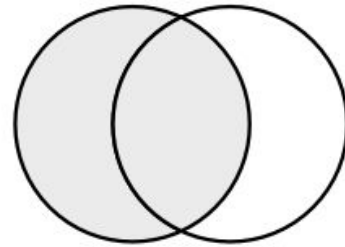
Merging (joining)



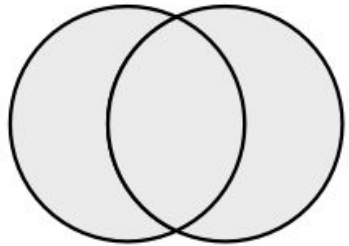
Merging (joining)



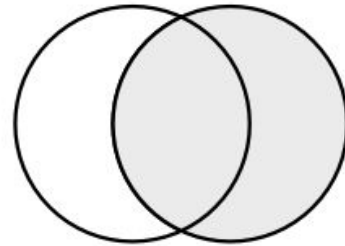
`inner_join(x, y)`



`left_join(x, y)`



`full_join(x, y)`



`right_join(x, y)`

Practice session

Voter Study 2019

[Home](#) \ [European Election Studies](#) \ [EES 2019 Study](#) \ [Voter Study 2019](#)

Research Design

The 2019 European Election Study (EES) Voter Study is a post-election study, conducted in all member states after the elections to the European Parliament were held between 23 and 26 May 2019.

The EES 2019 Voters Study was generously funded by the [Volkswagen Foundation](#). Additional support was provided from the [MZES](#) at the University of Mannheim and the [Amsterdam Center for European Studies](#) at the University of Amsterdam supported planning and realization of the study. The survey was conducted by Gallup International. For the first time in the history of EES, the data collection was

Q10 We have a number of parties in <country> each of which would like to get your vote. How probable is it that you will ever vote for the following parties? Please answer on a scale where 0 means "not at all probable" and 10 means "very probable".

[PARTY LIST C, PTV list up to 10 parties]

<Source: EES2014 QPP12>

	0 not at all probable	1	2	3	4	5	6	7	8	9	10 very proba- ble	Don't know the party
Party 1	0	1	2	3	4	5	6	7	8	9	10	98
Party 2	0	1	2	3	4	5	6	7	8	9	10	98
Party 3	0	1	2	3	4	5	6	7	8	9	10	98
Party 4	0	1	2	3	4	5	6	7	8	9	10	98
Party 5	0	1	2	3	4	5	6	7	8	9	10	98
Party 6	0	1	2	3	4	5	6	7	8	9	10	98
Party 7	0	1	2	3	4	5	6	7	8	9	10	98
Party 8	0	1	2	3	4	5	6	7	8	9	10	98
Party 9	0	1	2	3	4	5	6	7	8	9	10	98
Party 10	0	1	2	3	4	5	6	7	8	9	10	98

Useful resources

Data Management in R: A Guide for Social Scientists

Elff, Martin. 2020. *Data Management with R: A Guide for Social Scientists*. London: SAGE Publications.

This page provides material to accompany my recent book *Data Management in R: A Guide for Social Scientists*, which is being published by *Sage Publications*. The material is organised into different pages each corresponding to a chapter of the book:

- [Introduction](#)
- [Building Blocks of Data](#)
- [Data Frames and their Management](#)
- [Data Tables and the “Tidyverse”](#)
- [Social Science Surveys](#)
- [Data from Complex Samples](#)
- [Dates, Times, and Time Series](#)
- [Spatial and Geographical data](#)
- [Text as Data](#)

Data Import :: CHEAT SHEET



R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.



The front side of this sheet shows how to read text files into R with **readr**.



The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save **x**, an R object, to **path**, a file path, as:

Comma delimited file

`write_csv(x, path, na = "NA", append = FALSE, col_names = !append)`

File with arbitrary delimiter

`write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append)`

CSV for excel

`write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append)`

String to file

`write_file(x, path, append = FALSE)`

String vector to file, one element per line

`write_lines(x, path, na = "NA", append = FALSE)`

Object to RDS file

`write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...)`

Tab delimited files

`write_tsv(x, path, na = "NA", append = FALSE, col_names = !append)`

Read Tabular Data - These functions share the common arguments:

```
read_(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"),
quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,
n_max), progress = interactive())
```

```
a,b,c
1,2,3
4,5,NA
```

A	B	C
1	2	3
4	5	NA

Comma Delimited Files

`read_csv("file.csv")`

To make file.csv run:

`write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")`

```
a;b;c
1;2;3
4;5;NA
```

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files

`read_csv2("file2.csv")`

`write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")`

```
a|b|c
1|2|3
4|5|NA
```

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter

`read_delim("file.txt", delim = "|")`

`write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")`

```
a b c
1 2 3
4 5 NA
```

A	B	C
1	2	3
4	5	NA

Fixed Width Files

`read_fwf("file.fwf", col_positions = c(1, 3, 5))`

`write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")`

Tab Delimited Files

`read_tsv("file.tsv")` Also `read_table()`.

`write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")`

USEFUL ARGUMENTS

```
a,b,c
1,2,3
4,5,NA
```

Example file

`write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")`
`f <- "file.csv"`

1	2	3
4	5	NA

Skip lines

`read_csv(f, skip = 1)`

A	B	C
1	2	3
4	5	NA

No header

`read_csv(f, col_names = FALSE)`

A	B	C
1	2	3

Read in a subset

`read_csv(f, n_max = 1)`

x	y	z
1	2	3
4	5	NA

Provide header

`read_csv(f, col_names = c("x", "y", "z"))`

A	B	C
NA	2	3
4	5	NA

Missing Values

`read_csv(f, na = c("1", ""))`

Read Non-Tabular Data

Read a file into a single string

`read_file(file, locale = default_locale())`

Read each line into its own string

`read_lines(file, skip = 0, n_max = -1L, na = character(), locale = default_locale(), progress = interactive())`

Read Apache style log files

`read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())`

Read a file into a raw vector

`read_file_raw(file)`

Read each line into a raw vector

`read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive())`

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer
sex is a character
earn is a double (numeric)

1. Use `problems()` to diagnose problems.

`x <- read_csv("file.csv"); problems(x)`

2. Use a `col_` function to guide parsing.

- `col_guess()` - the default
- `col_character()`
- `col_double()`, `col_euro_double()`
- `col_datetime(format = "")` Also `col_date(format = "")`, `col_time(format = "")`
- `col_factor(levels, ordered = FALSE)`
- `col_integer()`
- `col_logical()`
- `col_number()`, `col_numeric()`
- `col_skip()`

```
x <- read_csv("file.csv", col_types = cols(
  A = col_double(),
  B = col_logical(),
  C = col_factor()))
```

3. Else, read in as character vectors then parse with a `parse_` function.

- `parse_guess()`
- `parse_character()`
- `parse_datetime()` Also `parse_date()` and `parse_time()`
- `parse_double()`
- `parse_factor()`
- `parse_integer()`
- `parse_logical()`
- `parse_number()`

`x$A <- parse_number(x$A)`



Data Transformation with dplyr : : CHEAT SHEET



dplyr functions work with pipes and expect **tidy data**. In tidy data:



Each **variable** is in its own **column**



Each **observation**, or **case**, is in its own **row**



x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function



summarise(.data, ...)
Compute table of summaries.
`summarise(mtcars, avg = mean(mpg))`



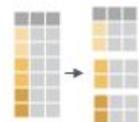
count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in ... Also **tally()**.
`count(iris, Species)`

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.



`mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))`

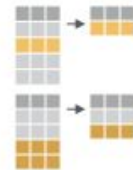
group_by(.data, ..., add = FALSE)
Returns copy of table grouped by ...
`g_iris <- group_by(iris, Species)`

ungroup(x, ...)
Returns ungrouped copy of table.
`ungroup(g_iris)`

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.



filter(.data, ...) Extract rows that meet logical criteria. `filter(iris, Sepal.Length > 7)`



distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
`distinct(iris, Species)`



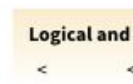
sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
`sample_frac(iris, 0.5, replace = TRUE)`



sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. `sample_n(iris, 10, replace = TRUE)`



slice(.data, ...) Select rows by position.
`slice(iris, 10:15)`



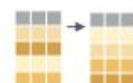
top_n(x, n, wt) Select and order top n entries (by group if grouped data). `top_n(iris, 5, Sepal.Width)`

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%		xor()
>	>=	!is.na()	!	&	

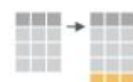
See **?base::logic** and **?Comparison** for help.

ARRANGE CASES



arrange(.data, ...) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
`arrange(mtcars, mpg)`
`arrange(mtcars, desc(mpg))`

ADD CASES

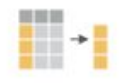


add_row(.data, ..., .before = NULL, .after = NULL)
Add one or more rows to a table.
`add_row(faithful, eruptions = 1, waiting = 1)`

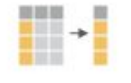
Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.



pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
`pull(iris, Sepal.Length)`



select(.data, ...)
Extract columns as a table. Also **select_if()**.
`select(iris, Sepal.Length, Species)`

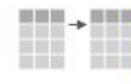
Use these helpers with select(),
e.g. `select(iris, starts_with("Sepal"))`

contains(match)	num_range(prefix, range)	; e.g. <code>mpg:cyl</code>
ends_with(match)	one_of(...)	, e.g. <code>-Species</code>
matches(match)	starts_with(match)	

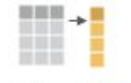
MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

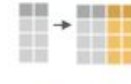
vectorized function



mutate(.data, ...)
Compute new column(s).
`mutate(mtcars, gpm = 1/mpg)`



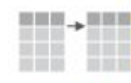
transmute(.data, ...)
Compute new column(s), drop others.
`transmute(mtcars, gpm = 1/mpg)`



mutate_all(.tbl, .funs, ...) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
`mutate_all(faithful, funs(log(.), log2(.)))`
`mutate_if(iris, is.numeric, funs(log(.)))`



mutate_at(.tbl, .cols, .funs, ...) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for **select()**.
`mutate_at(iris, vars(-Species), funs(log(.)))`



add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. `add_column(mtcars, new = 1:32)`



rename(.data, ...) Rename columns.
`rename(iris, Length = Sepal.Length)`



String manipulation with stringr : : CHEAT SHEET

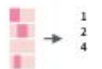


The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

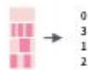
Detect Matches



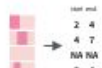
str_detect(string, pattern) Detect the presence of a pattern match in a string.
`str_detect(fruit, "a")`



str_which(string, pattern) Find the indexes of strings that contain a pattern match.
`str_which(fruit, "a")`



str_count(string, pattern) Count the number of matches in a string.
`str_count(fruit, "a")`



str_locate(string, pattern) Locate the positions of pattern matches in a string. Also **str_locate_all**.
`str_locate(fruit, "a")`

Subset Strings



str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector.
`str_sub(fruit, 1, 3)`; `str_sub(fruit, -2)`



str_subset(string, pattern) Return only the strings that contain a pattern match.
`str_subset(fruit, "b")`

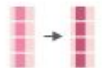


str_extract(string, pattern) Return the first pattern match found in each string, as a vector. Also **str_extract_all** to return every pattern match.
`str_extract(fruit, "[aeiou]")`



str_match(string, pattern) Return the first pattern match found in each string, as a matrix with a column for each () group in pattern. Also **str_match_all**.
`str_match(sentences, "[a]the ([^]+)")`

Mutate Strings



str_sub() <- value. Replace substrings by identifying the substrings with `str_sub()` and assigning into the results.
`str_sub(fruit, 1, 3) <- "str"`



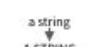
str_replace(string, pattern, replacement) Replace the first matched pattern in each string.
`str_replace(fruit, "a", "-")`



str_replace_all(string, pattern, replacement) Replace all matched patterns in each string.
`str_replace_all(fruit, "a", "-")`



str_to_lower(string, locale = "en")¹ Convert strings to lower case.
`str_to_lower(sentences)`

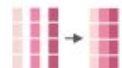


str_to_upper(string, locale = "en")¹ Convert strings to upper case.
`str_to_upper(sentences)`



str_to_title(string, locale = "en")¹ Convert strings to title case.
`str_to_title(sentences)`

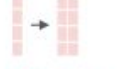
Join and Split



str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string.
`str_c(letters, LETTERS)`



str_c(..., sep = "", collapse = NULL) Collapse a vector of strings into a single string.
`str_c(letters, collapse = "")`



str_dup(string, times) Repeat strings times times.
`str_dup(fruit, times = 2)`



str_split_fixed(string, pattern, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings.
`str_split_fixed(fruit, " ", n=2)`



str_glue(..., sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate.
`str_glue("Pi is {pi}")`



str_glue_data(x, ..., sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate.
`str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")`

Manage Lengths



str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters).
`str_length(fruit)`



str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width.
`str_pad(fruit, 17)`



str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis.
`str_trunc(fruit, 3)`



str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string.
`str_trim(fruit)`

Order Strings



str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...) ¹ Return the vector of indexes that sorts a character vector.
`x[str_order(x)]`



str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...) ¹ Sort a character vector.
`str_sort(x)`

Helpers



str_conv(string, encoding) Override the encoding of a string.
`str_conv(fruit, "ISO-8859-1")`



str_view(string, pattern, match = NA) View HTML rendering of first regex match in each string.
`str_view(fruit, "[aeiou]")`



str_view_all(string, pattern, match = NA) View HTML rendering of all regex matches.
`str_view_all(fruit, "[aeiou]")`

str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs.
`str_wrap(sentences, 20)`

¹ See bit.ly/ISO639-1 for a complete list of locales.

Full-on treatments of the topic

Books

- Baumer *et al.*, *Modern Data Science with R*
- Grolemund and Wickham, *R for Data Science*
- Fogarty, *Quantitative Social Science Data with R*

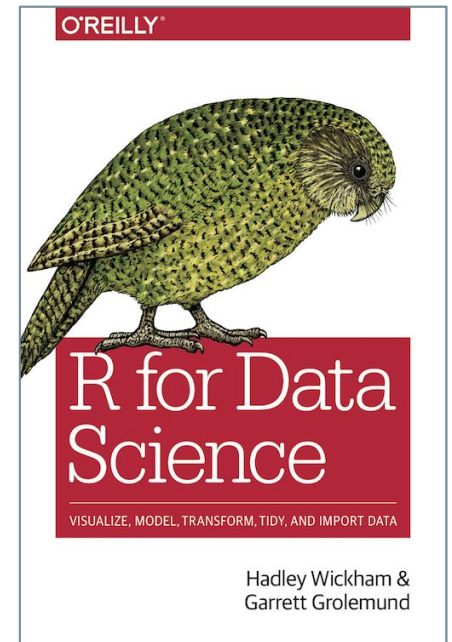
Courses

- **Data for Data Scientists** (LSE)
- **Data Science for Economists** (University of Oregon)
- **Introduction to Data Science** (Duke)

Canonical reading

Grolemund and Wickham, *R for Data Science*

- ch. 5 on **data transformation** with `dp1yr`
- ch. 6 on **writing workflows** (scripts)
- ch. 9–16 on **data wrangling**
 - see esp. ch. 12 (tidy data)
 - also covers strings, factors, dates

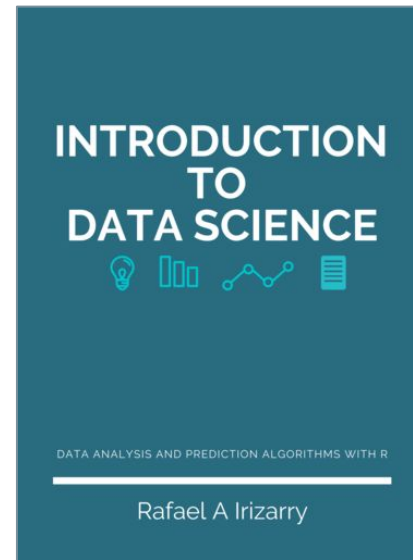


Note that, for very large datasets, there are faster ways to perform some of the operations covered here, using e.g. `data.table` or `collapse`.

Similar treatments

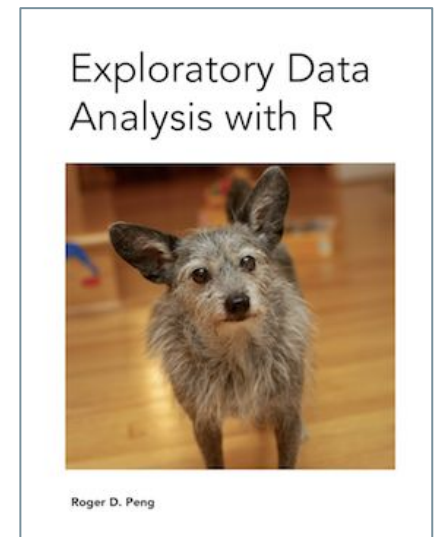
Irizarry, *Introduction to Data Science*

- ch. 4 on the **tidyverse**
- ch. 5 on **importing data**



Peng, *Exploratory Data Analysis with R*

- ch. 3 on **data managements** with **dplyr**
- ch. 4 · **good data analysis principles**



Videos to watch it happen

- **RStudio** (Grolemund, Wickham) has a series of videos on [Data Wrangling with R and the Tidyverse](#)
See esp. the [‘dplyr’](#) and [‘two datasets’](#) episodes
- **Hadley Wickham** has an excellent [tidy data tutorial](#) that makes use of some the packages covered today
See also his [Managing Many Models](#) tutorial
- **David Robinson** films [hour-long examples](#) of how to resolve [Tidy Tuesday](#) challenges with *dp1yr et al.*
See also his [Ten Tremendous Tricks in the Tidyverse](#)

Various links

- Look at e.g. [Quantitative Social Science Data](#) for an overview of what kind of data already exist
- For lots of example code that does data wrangling, take a look at [how Party Facts is built](#)
- On how to use surveys with complex weighting designs, see the wonderful [Analyze Survey Data for Free](#)
- Read [Automated Data Collection with R](#) if you are interested in Web scraping (and text mining)
 - See also Irizarry, [Introduction to Data Science](#), ch. 23, for a shorter overview of Web scraping

Activity / Homework

For those who want a **complex** problem

1. Download the **'European Mood' Indicator** by Guinaudeau and Schnatterer, and load it in R
2. Reshape the data in order to get the following variables: **year, semester, country, mood**
3. Can you spot what must be a **data entry error** in one of the variables? Write the code to fix it
4. Use **ggplot2** to reproduce as closely as possible the **plot on the next slide** (anticipating our next session together)



it shouldn't be too hard to guess what the blue dot markers stand for...

For those who prefer a **simpler** problem

1. Complete Steps 1–3 and **forget about Step 4**, which is the harder (most time-consuming) step
2. **Create a ‘decade’ variable** measuring, for each country, the *mean* and *s.d.* of EU mood in each decade since 1970
3. **Create a ‘delta’ variable** measuring, for each country, the *change in EU mood from one decade to the next*

Thanks for your work! I will provide solutions to both problem sets at the beginning of Session 5.

Thanks for your attention

See you again for

Data Visualization

(Session 5)



Bonus: Databases

User information					User activity			
user_id	country	agecat	gender	ppacat	url_id	user_id	timestamp	action
user1	DE	35-44	F	NULL	url1	user1	1483247834	like
user2	US	25-34	M	-1	url2	user1	1483266874	click
user3	BR	25-34	F	NULL	url1	user2	1483276812	share
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

JOIN ON user_id

GROUP BY (PRIVATE) SUM $\left(\sum [1 + N(0, \sigma^2)] | \mathbf{X} \right)$



url_id	year-month	country	agecat	gender	ppacat	view	click	share	like	...
url1	2017-01	DE	35-44	F	NULL	548	44	-4	68	⋮
url1	2017-01	US	25-34	M	-1	4736	199	111	152	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Breakdown Table

url_id	clean_url	share_title	...	hate_speech	top_country
url1	www.url1.com/hi.html	Hello	...	0	DE
url2	www.url2.com/hi.html	Hi	...	4	US
⋮	⋮	⋮	⋮	⋮	⋮

URL Attributes Table

Accessible data

Databases (DBs)



- Databases are **tables** with **relational schemes**
 - Rely on shared 'id' column between tables
 - Optimised for speed with large datasets
 - Requires learning a **DB query language**
- Two possible ways to go
 - **SQL** — **kinda** row-oriented, many variants
 - **MongoDB** — *column*-oriented, fast
- For an efficient introduction, see Grolemund and Wickham, *R for Data Science*, ch. 13 (relational data)

See Databases using R for and dbplyr

- **ODBC/DBI-driven (best practices)**

- Microsoft SQL Server
- **MySQL**
- Oracle
- **PostgreSQL**
- **SQLite**

SQL

see github.com/r-dbi
and [Jim Hester's slides](#)

- **MonetDB**

- MonetDB.R
- **MonetDBLite**



see the [tutorials](#)

see also
[bigrquery](#)



Google BigQuery

see also
[solrium](#)



and many other DB drivers