Lesson 2 of 5
Import, Clean, Transform

Intro to R workshop, LU Skills School

Instructor:

Christopher Swader
LU Sociology (Assoc. Prof) and LMU (Munich, Researcher)

Teaching Assistant: Maximilian Hornung (MS programme in Social Scientific Data Analysis, LU)

Slides adapted from and inspired by:
Irina Vartanova, Institute for Future Studies, Stockholm

Today’s agenda

- Importing Data
- Exporting Data
- Cleaning Data
- Selecting Columns/Variables
- Creating new variables
- Filtering Rows/Cases
- Resorting Cases
- The pipe operator %>%
- Converting data between long and short formats

- Use the link to download the files we will be using today: https://github.com/ChristopherSwader/R_introduction
- Download the Day2 folder.

Reminder

- When you open Rstudio, be sure to clear your environment.
- Set your working directory, either through the menu or through the script.
- Run all of your code in your current script and save regularly.

Tidy data

A data set is **tidy** if:

1. Each **variable** is in its own **column**
2. Each **case** is in its own **row**
3. Each **value** is in its own **cell**

gss-moral-issues.csv

Time series for US public opinion on 81 moral issues 1972 - 2018.

Abortion and approval of premarital sex in the USA

Abortion: ”Do you think it should be possible for a pregnant woman to obtain a legal abortion if the woman wants it for any reason?” (*abany*)

Premarital sex: ”There's been a lot of discussion about the way morals and attitudes about sex are changing in this country. If a man and woman have sex relations before marriage -- do you think it is wrong?” (*premarsx*)

```
moral_issues
```

issue <chr>	year <dbl>	n <dbl>	prop <dbl>	prop_lib <dbl>	prop_cons <dbl>
abdefect	1972	1526	0.78571429	NA	NA
abhlth	1972	1539	0.86939571	NA	NA
abnomore	1972	1528	0.39725131	NA	NA
abpoor	1972	1507	0.48772395	NA	NA
abrape	1972	1512	0.79100529	NA	NA
absingle	1972	1502	0.43541944	NA	NA
busing	1972	1544	0.20272021	NA	NA
colath	1972	1538	0.41872562	NA	NA
colcom	1972	1497	0.34602538	NA	NA

Import data

gss-moral-issues.csv

```
issue,year,n,prop
abdefect,1972,1526,0.7857142857142857
abhlth,1972,1539,0.8693957115009746
abnomore,1972,1528,0.3972513089005235 7
abpoor,1972,1507,0.4877239548772396
abrape,1972,1512,0.791005291005291
absingle,1972,1502,0.4354194407456724 6
busing,1972,1544,0.202720207253886
colath,1972,1538,0.4187256176853056
colcom,1972,1497,0.3460253841015364
```

Import

You can click through menus, but...

Click Import Dataset From Text (readr)…

Import

Not reproducible!

Click Import Dataset From Text (readr)…

Import Text Data

File/URL:

~/Dropbox/projects/R course/R intro 2020/Day3_Transform/gss-moral-issues.csv

Browse...

Data Preview:

issue (character)	year (double)	n (double)	prop (double)	prop_lib (logical)	prop_cons (logical)
abdefect	1972	1526	0.78571429	NA	NA
abhlth	1972	1539	0.86939571	NA	NA
abnomore	1972	1528	0.39725131	NA	NA
abpoor	1972	1507	0.48772395	NA	NA
abrape	1972	1512	0.79100529	NA	NA
absingle	1972	1502	0.43541944	NA	NA

column 5: boolean

Previewing first 50 entries.

Import Options:

Name: gss_moral_issues

Skip: 0

First Row as Names

Trim Spaces

Open Data Viewer

Delimiter: Comma

Quotes: Default

Locale: Configure...

Escape: None

Comment: Default

NA: Default

Code Preview:

```
library(readr)
gss_moral_issues <-
read_csv("Day3_Transform
/gss-moral-issues.csv")
View(gss_moral_issues)
```

Reading rectangular data using readr

Import

Cancel

The Code Equivalent

Your Turn

Import the **gss-moral-issues.csv** data set as **moral_issues**. Then copy the import code into your R script (not your console).

Run it.

From the Working directory...

```
moral_issues <- read_csv("gss-moral-issues.csv")
```

Error in file(file, "rt") :
cannot open the connection
In addition: Warning message:
In file(file, "rt") :
  cannot open file 'test.csv': No such file or directory

Common error. *R* has been given the wrong path!

What is your working directory?
getwd()

Set your working directory path?
setwd()

SPSS or Stata data

https://haven.tidyverse.org/index.html

```
read_sav("file-name.sav")
```

```
read_dta("file-name.dta")
```

View

- View() your new data.frame!
- Or head() to view the first 5 rows in the console

gss_moral_issues

Filter

	issue	year	n	prop	prop_lib	prop_cons
1	abdefect	1972	1526	0.78571429	NA	NA
2	abhlth	1972	1539	0.86939571	NA	NA
3	abnomore	1972	1528	0.39725131	NA	NA
4	abpoor	1972	1507	0.48772395	NA	NA
5	abrape	1972	1512	0.79100529	NA	NA
6	absingle	1972	1502	0.43541944	NA	NA
7	busing	1972	1544	0.20272021	NA	NA
8	colath	1972	1538	0.41872562	NA	NA
9	colcom	1972	1497	0.34602538	NA	NA
10	fework	1972	1577	0.65377299	NA	NA
11	gunlaw	1972	1562	0.72407170	NA	NA
12	libath	1972	1552	0.62822165	NA	NA
13	libcom	1972	1532	0.55548303	NA	NA
14	premarsx	1972	1537	0.51528953	NA	NA
15	racfew	1972	1338	0.92825112	NA	NA

```r
head(moral_issues)
```

```
> head(moral_issues)
     issue year    n      prop prop_lib prop_cons
1 abdefect 1972 1526 0.7857143       NA        NA
2   abhlth 1972 1539 0.8693957       NA        NA
3 abnomore 1972 1528 0.3972513       NA        NA
4   abpoor 1972 1507 0.4877240       NA        NA
5   abrape 1972 1512 0.7910053       NA        NA
6 absingle 1972 1502 0.4354194       NA        NA
>
```

Choose your data.frame type

- Convert your data object into the type of data.frame you want to use.
- Check what kind it is with *str(df)*
- Then convert if needed. e.g. df ← *as.tibble(df)*

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"

A	B	C
1	2	3
4	5	NA

No header
read_csv(f, **col_names = FALSE**)

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header
read_csv(f, **col_names = c("x", "y", "z")**)

1	2	3
4	5	NA

Skip lines
read_csv(f, **skip = 1**)

A	B	C
1	2	3

Read in a subset
read_csv(f, **n_max = 1**)

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)

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at tidyverse.org • readr 1.1.0 • tibble 1.2.12 • tidyr 0.6.0 • Updated: 2019–08

Export data

write_csv()

Saves data set as a csv on your computer.

```
write_csv(moral_issues, path = "moral_issues.csv")
```

Table to save

Here it is not a full path, which defaults to your working directory

You can also save your data in any other format. E.g. excel, spss, stata.

I would recommend **csv** while you work in R.

If you need to call up your data extremely quickly, consider *saveRDS()*. R's format.

Clean data

Type.convert

```
moral_issues_converted <-type.convert( moral_issues )
```

This function will automatically convert each of your variables to a uniform format. Be careful to check before and after to see what has changed.

e.g. How to isolate just these cases/rows?

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583
fehome	1996	1884	0.836518
libath	1996	1853	0.7053427
marblk	1996	577	0.4038128
polescap	1996	1819	0.2820231
spkrac	1996	1878	0.6198083

issue	year	n	prop
marblk	1990	940	0.1670213
marblk	1996	577	0.4038128
…			

... or how to work with just 2 columns?

Isolating data

select() - extract **variables**
filter() - extract **cases**
arrange() - reorder **cases**

These functions are loaded when you load tidyverse: *library(tidyverse)*.

Or if you don't want the whole tidyverse, *library(dplyr)*

I will also provide you with non-tidyverse equivalents of these commands, written in base R, using square brackets indexing *(because it is very useful for more complex programming)*

dplyr

a grammar of data manipulation

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**

pipes
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)
ends_with(match)
matches(match)
num_range(prefix, range)
one_of(…)
starts_with(match)
:, e.g. mpg:cyl
-, e.g. -Species

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)

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2017-03

Selecting

columns/variables

select()

Extract columns by name.

```
select(.data, …)
```

data frame to transform

name(s) of columns to extract
(or a select helper function)

Extract columns by name: select()

```
select(moral_issues, issue, prop)
```

```
moral_issues
```

issue	year	n	prop
abdefect	1974	1440	0.8514
abhlth	1974	1452	0.9242
abnomore	1974	1411	0.4692
abpoor	1974	1417	0.5476
abrape	1974	1420	0.8648
absingle	1974	1420	0.5007
busing	1972	1544	0.2027202073
colath	1972	1538	0.41872

issue	prop
abdefect	0.8514
abhlth	0.9242
abnomore	0.4692
abpoor	0.5476
abrape	0.8648
absingle	0.5007
busing	0.2027202073
colath	0.41872

The square brackets equivalent of this is:

moral_issues[,c("issue", "prop")]

This is my personal preference.

select()

You can also rename variables when you select them

```
select(moral_issues, moral_issue = issue, proportion = prop)
```

moral_issues

issue	year	n	prop
abdefect	1974	1440	0.8514
abhlth	1974	1452	0.9242
abnomore	1974	1411	0.4692
abpoor	1974	1417	0.5476
abrape	1974	1420	0.8648
absingle	1974	1420	0.5007

→

moral_issue	proportion
abdefect	0.8514
abhlth	0.9242
abnomore	0.4692
abpoor	0.5476
abrape	0.8648
absingle	0.5007

Your turn

Select just the ”issue” and ”n” columns of the data into a new object called new_moral_issues

```
new_moral_issues ← select(moral_issues, n)
```

```
# A tibble: 1,651 × 2
   issue        n
   <chr>    <dbl>
 1 abdefect  1526
 2 abhlth    1539
 3 abnomore  1528
 4 abpoor    1507
 5 abrape    1512
 6 absingle  1502
 7 busing    1544
 8 colath    1538
 9 colcom    1497
10 fework    1577
# ℹ 1,641 more rows
# ℹ Use `print(n = ...)` to see more rows
```

select() helpers

: - Select range of columns

```
select(mpg, cty:class)
```

- - Select every column but

```
select(mpg, -c(cty, hwy))
```

starts_with() - Select columns that start with…

```
select(mpg, starts_with("c"))
```

ends_with() - Select columns that end with…

```
select(mpg, ends_with("y"))
```

select() helpers

contains() - Select columns whose names contain…

```
select(mpg, contains("d"))
```

matches() - Select columns whose names match a '*regular expression*'

```
select(mpg, matches("^.{4}$"))
```

one_of() - Select columns whose names are one of a set

```
select(mpg, one_of(c("fl", "fuel", "Fuel"))
```

num_range() - Select columns named in prefix, number style

```
select(mpg, num_range("x", 1:5))
```

select() helpers

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

contains(match**)**	**num_range(**prefix, range**)**	**:**, e.g. mpg:cyl
ends_with(match**)**	**one_of(…)**	**-**, e.g, -Species
matches(match**)**	**starts_with(**match**)**	

Quiz

Which of these is NOT a way to select the **prop, prop_lib** and **prop_cons** columns together?

```
select(moral_issues, -c(issue, year, n))

select(moral_issues, prop:prop_cons)

select(moral_issues, starts_with("prop"))

select(moral_issues, ends_with("prop"))
```

Quiz

Which of these is NOT a way to select the **prop, prop_lib** and **prop_cons** columns together?

```
select(moral_issues, -c(issue, year, n))

select(moral_issues, prop:prop_cons)

select(moral_issues, starts_with("prop"))

select(moral_issues, ends_with("prop"))
```

select() and $ do the same thing (select columns), but they output different formats: data.frame vs. vector.

You can easily convert between them.

select()

Extract columns by name.

```
select(moral_issues, n)
```

moral_issues

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583

→

n
890
881
940
854
892
1236

Still a dataframe

Notice Rstudio's nice autofill feature after you add the $ sign, allowing you to choose between available options.

$

Extract column contents as a vector.

```
moral_issues$n
```

moral_issues

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583

→ `1526 1539 1528 1507 …`

$

Extract column contents as a vector.

```
moral_issues$n
```

data frame

$

column name (no quotes)

pull()

Pipe friendly version of $

```
moral_issues %>% pull(n)
```

moral_issues

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583

→ `1526 1539 1528 1507 …`

Create new columns/variables

mutate()

Create new columns.

```
moral_issues %>%
  mutate(percent = round(prop*100, 2))
```

moral_issues

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583

→

issue	year	n	prop	percent
fehome	1990	890	0.8213483	88.66
libath	1990	881	0.6912599	75.77
marblk	1990	940	0.1670213	27.70
polescap	1990	854	0.2236534	25.11
spkrac	1990	892	0.6423767	67.98
fehire	1996	1236	0.6480583	72.78

mutate()

Create new columns.

```
moral_issues %>%
  mutate(percent = round(prop*100, 2), nper = round(percent))
```

moral_issues

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583

→

issue	year	n	prop	percent	nper
fehome	1990	890	0.8213483	88.66	89
libath	1990	881	0.6912599	75.77	76
marblk	1990	940	0.1670213	27.70	28
polescap	1990	854	0.2236534	25.11	25
spkrac	1990	892	0.6423767	67.98	68
fehire	1996	1236	0.6480583	72.78	73

Your Turn

Use **`mutate()`** *to calculate the difference between liberal and conservative public opinion and store it as a new* **`polr`** *variable.*

Which issue-years are the most polarised?

```r
moral_issues <- moral_issues %>%
  mutate(polr = prop_lib - prop_cons) %>%
  arrange(desc(polr))
moral_issues
```

```
     issue year    n      prop  prop_lib prop_cons      polr
1 helppoor 2018  827 0.5816203 0.8509091 0.2720588 0.5788503
2  helpblk 2018  986 0.4290061 0.7225806 0.1722054 0.5503752
3  marhomo 2008 1166 0.4545455 0.7602524 0.2237197 0.5365327
4 helpsick 2012  887 0.6899662 0.9000000 0.3937282 0.5062718
5 helppoor 2010  743 0.5074024 0.7607656 0.2579505 0.5028150
6 helppoor 2012  714 0.5000000 0.7560976 0.2555556 0.5005420
```

Filtering

for cases

filter()

Extract rows that meet logical criteria.

```
filter(.data, … )
```

data frame to transform

one or more logical tests (filter returns each row for which the test is TRUE)

filter()

Extract rows that meet logical criteria.

```
filter(moral_issues, issue == "marblk")
```

moral_issues

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583

→

issue	year	n	prop
marblk	1990	940	0.1670213
marblk	1996	577	0.4038128
…			

filter()

Extract rows that meet logical criteria.

```
filter(moral_issues, issue == "marblk")
```

moral_issues

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583

== tests if equal
(returns TRUE or FALSE)

Logical tests

?Comparison

`x < y`	Less than
`x > y`	Greater than
`x == y`	Equal to
`x <= y`	Less than or equal to
`x >= y`	Greater than or equal to
`x != y`	Not equal to
`x %in% y`	Group membership
`is.na(x)`	Is NA
`!is.na(x)`	Is not NA

Tip… you can multipy a logical vector by 1 to convert it to a dummy vector of zeroes and ones.

```
x <- c(T, F, T, T, T)
x*1
```

The square bracket equivalent to filter.
it uses these logical tests too.

```
moral_issues[moral_issues$issue=="marblk",] Notice i put my
true-false code before the comma to indicate that I am
selecting for rows instead of columns.


Those True-False vectors are actually called Boolean masks and
are really handy for programming in general, especially for
filtering, selecting.


e.g. you can also invert the mask by selecting every issue
except "marblk"

moral_issues[!moral_issues$issue=="marblk",]

Or you can combine multiple logical statements with AND (&) or
OR(|)
```

%in%

- %in% is really handy for checking if a given row (or column) are within a set.
- e.g. you can select when the issue is ”marblk” or ”libath” or ”fehome”

```
my_issues ← c(”marblk”, ”libath”, ”fehome”)
moral_issues[moral_issues$issue %in% my_issues,]
```

Quiz

What does NA stand for?

1 **"1"** **"one"** **NA**

Quiz

What does NA stand for?

1 **"1"** **"one"** **NA**

Missing Value (Not Available)

Quiz

What is the result?

1 == 1

Quiz

What is the result?

1 == 1

TRUE

Quiz

What is the result?

is.na(NA)

TRUE

Your Turn

*Use **moral_issues,** the logical operators ,and filter() or [,] to find:*

- *All of the rows for "abany" issue where **prop** is greater than or equal to 0.40*

```
filter(moral_issues,issue=="abany" ,  prop>=.40)
  issue  year     n  prop prop_lib prop_cons
  <chr> <dbl> <dbl> <dbl>    <dbl>     <dbl>
1 abany  1980  1406 0.411    0.497     0.373
2 abany  1989   989 0.403    0.554     0.328
3 abany  1990   877 0.434    0.584     0.311
```

```
moral_issues[moral_issues$issue=="abany" & moral_issues$prop>=.40,]
```

common mistakes

1. Using = instead of ==

```
filter(moral_issues, issue = "marblk")
```

```
filter(moral_issues, issue == "marblk")
```

2. Forgetting quotes

```
filter(moral_issues, issue == marblk)
```

```
filter(moral_issues, issue == "marblk")
```

3. Using == instead of is.na

```
filter(moral_issues, prop_lib == NA)
```

```
filter(moral_issues,is.na(prop_lib))
```

filter()

Extract rows that meet *every* logical criteria.

```
filter(moral_issues, issue == "marblk", year == 2018)
```

moral_issues

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583

→

issue	year	n	prop
marblk	1990	940	0.1670213

filter()

Extract rows that meet *every* logical criteria.

```
filter(moral_issues, issue == "marblk" & year == 2018)
```

moral_issues

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583

→

issue	year	n	prop
marblk	1990	940	0.1670213

Reminder, don't change moral_issues itself. Assign the filtered dataset as a new object. That way you have a backup and can doublecheck it worked.

Boolean operators

?base::Logic

`a & b`	and
`a | b`	or
`xor(a,b)`	exactly or
`!b`	not

```
x >= 2 & x < 3
```

↓

```
TRUE & TRUE
```

↓

```
TRUE
```

Your Turn

Use Boolean operators to alter the code below to return only the rows that contain either "abany" or "premarsx" issue in 2018

filter(moral_issues, issue == "abany" | issue == "premarsx")

```
filter(moral_issues, (issue == "marblk" | issue ==
"affrmact") & year == 2018)

#   issue     year     n  prop prop_lib prop_cons
1 affrmact  2018  1447 0.250    0.426     0.144
2 marblk    2018   793 0.768    0.859     0.650
```

Use parentheses to group statements together

Two more common mistakes

4. Collapsing multiple tests in one

```
filter(moral_issues, 1990 < year < 2000)
```

```
filter(moral_issues, year > 1990, year < 2000)
```

5. Stringing together too many tests (when you could use %in%)

```
filter(moral_issues, issue == "marblk"|issue == "affrmact")
```

```
filter(moral_issues, issue %in% c("marblk", "affrmact"))
```

```
filter(moral_issues, (issue == "marblk" | issue == "affrmact") &
                     year == 2018)

filter(moral_issues, issue %in% c("marblk", "affrmact") &
                     year == 2018)

#   issue     year     n  prop prop_lib prop_cons
1 affrmact  2018  1447 0.250    0.426     0.144
2 marblk    2018   793 0.768    0.859     0.650
```

Resorting your rows

arrange()

Order rows from smallest to largest values.

```
arrange(.data, …)
```

data frame to transform

one or more columns to order by (additional columns will be used as tie breakers)

The square brackets equivalent of this is:
moral_issues[order(moral_issues$prop),]

arrange()

Order rows from smallest to largest values.

```
arrange(moral_issues, prop)
```

moral_issues

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583

→

issue	year	n	prop
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583
libath	1990	881	0.6912599
fehome	1990	890	0.8213483

The square brackets equivalent of this is:
moral_issues[order(moral_issues$prop, decreasing=TRUE),]

desc()

Changes ordering to largest to smallest.

```
arrange(moral_issues, desc(prop))
```

moral_issues

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
marblk	1990	940	0.1670213
polescap	1990	854	0.2236534
spkrac	1990	892	0.6423767
fehire	1996	1236	0.6480583

→

issue	year	n	prop
fehome	1990	890	0.8213483
libath	1990	881	0.6912599
fehire	1996	1236	0.6480583
spkrac	1990	892	0.6423767
polescap	1990	854	0.2236534
marblk	1990	940	0.1670213

Your turn

What is the smallest value of ***prop****?*

What is the largest?

```
arrange(moral_issues, prop)
#    issue     year     n   prop prop_lib prop_cons
# 1 polattak  1975  1133 0.0150  0.00990    0.0173
# 2 pornlaw   1994  1984 0.0297  0.0417     0.0263
# 3 polattak  1973  1119 0.0304 NA         NA
# 4 pornlaw   2000  1865 0.0316  0.0524     0.0264
# 5 pornlaw   2012  1323 0.0325  0.0339     0.0362
# 6 pornlaw   1993  1040 0.0327  0.0474     0.0291
```

Base R equivalent… and more simple
min(moral_issues$prop). Also try *summary(moral_issues$prop)*

```
arrange(moral_issues, desc(prop))
#    issue     year     n  prop prop_lib prop_cons
# 1 hitmarch  1993  1032 0.978    0.974     0.981
# 2 hitmarch  1994   505 0.976    0.976     0.989
# 3 hitmarch  1988   976 0.975    0.984     0.968
# 4 hitmarch  1989  1012 0.973    0.958     0.971
# 5 hitmarch  1991   996 0.971    0.969     0.972
# 6 racfew    1996   980 0.970    0.969     0.974
# 7 hitmarch  1975  1062 0.970    0.983     0.981
```

Base R equivalent:
max(moral_issues$prop)

”the pipe”
%>%

Example Coding Steps

```
issues_2018 <- filter(moral_issues, year == 2018)
issues_2018 <- select(issues_2018, issue, prop)
issues_2018 <- arrange(issues_2018, desc(prop))
issues_2018
```

1. Filter cases with data from 2018 from moral_issues.
2. Select the name and prop columns from the result.
3. Arrange those columns so that the most popular issues appear near the top.

Iteration Style

```
issues_2018 <- filter(moral_issues, year == 2018)
issues_2018 <- select(issues_2018, issue, prop)
issues_2018 <- arrange(issues_2018, desc(prop))
issues_2018
```

Nested Parentheses Style

```
arrange(select(filter(moral_issues, year == 2018), issue,
prop), desc(prop))
```

The pipe operator %>%

`moral_issues  select(_____, …)`

Passes result on left into first argument of function on right.

So, for example, these do the same thing.

```
filter(moral_issues, year == 2018)
moral_issues %>% filter(year == 2018)
```

Pipes

```
moral_issues
issues_2018 <- filter(moral_issues, year == 2018)
issues_2018 <- select(issues_2018, issue, prop)
issues_2018 <- arrange(issues_2018, desc(prop))
issues_2018
```

Iterative style

```
issues_2018 <-
   moral_issues %>%
  filter(year == 2018) %>%
  select(issue, prop) %>%
  arrange(desc(prop))
```

Pipe Style

Shortcut to type %> %

Cmd + Shift + M (Mac)

Ctrl + Shift + M (Windows)

Your Turn

*Use **%>%** to write a sequence of functions that:*

1. *Using moral_issues, filter the rows with data from 1974, then…*
2. *Selects the **issue** and **prop** columns, then…*
3. *Arranges the results so that the issues with the lowest values of public opinion are near the top.*

```r
moral_issues %>%
    filter(year == 1974) %>%
    select(issue, prop) %>%
    arrange(prop)

#    issue     prop
# 1 xmarsex  0.140
# 2 busing   0.209
# 3 homosex  0.217
# 4 cappun   0.335
# 5 colath   0.433
# 6 colcom   0.444
```

Recap

Extract variables with **select()**

Or *[, x]*

Extract cases with **filter()**

Or *[x,]*

Arrange cases, with **arrange()**.

Or *[order(),]*

tomorrow Make tables of summaries with **summarise()**.

Or *aggregate()*

Make new variables, with **mutate()**.

Or df$new <- *x*

Converting

data between long and short formats

Fake pollution data

02-Transform-Data.Rmd

Preview Insert Run

```
---
title: "Transform Data"
output: html_notebook
---

```{r setup}
library(tidyverse)
library(babynames)

# Toy datasets to use

pollution <- tribble(
       ~city,    ~size,
  "New York", "large",
  "New York", "small",
    "London", "large",
    "London", "small",
   "Beijing", "large",
   "Beijing", "small",
)

band <- tribble(
    ~name,      ~band,
  "Mick",   "Stones",
  "John",  "Beatles",
  "Paul",  "Beatles"
)

instrument <- tribble(
     ~name,    ~plays,
   "John", "guitar",
   "Paul",    "bass",
  "Keith", "guitar"
)
```

Your Turn 1
```

34:2 Chunk 1: setup R Markdown

```
pollution <- tribble(
       ~city,    ~size, ~amount,
  "New York", "large",      23,
  "New York", "small",      14,
    "London", "large",      22,
    "London", "small",      16,
   "Beijing", "large",      121,
   "Beijing", "small",      56
)
```

Long to short conversion

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	large	small

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	large	small
New York	23	

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	large	small
New York	23	14

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	large	small
New York	23	14
London	22	

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	large	small
New York	23	14
London	22	16

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	large	small
New York	23	14
London	22	16
Beijing	121	

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	large	small
New York	23	14
London	22	16
Beijing	121	56

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	large	small
New York	23	14
London	22	16
Beijing	121	56

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

pivot_wider()

city	large	small
New York	23	14
London	22	16
Beijing	121	56

	1	2
city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

Here, column 1 becomes the new variables, and column 2 becomes the new values for that variable .

If column 2 had more than 2 values, the table would become much wider

city	large	small
New York	23	14
London	22	16
Beijing	121	56

names_from (new column names)

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

argument names_from: 'the values of this column will be converted into new variable names

city	large	small
New York	23	14
London	22	16
Beijing	121	56

names_from **values_from** (new cells)

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

argument values_from: ’the values of this column will be converted into values in these new variables’

city	large	small
New York	23	14
London	22	16
Beijing	121	56

pivot_wider()

```
pollution %>% pivot_wider(names_from = size, values_from = amount)
```

data frame to reshape

column to use for keys (becomes new column names)

column to use for values (becomes new column cells)

```
pollution %>% pivot_wider(size, amount)
```

```
      city  size amount
1 New York large     23
2 New York small     14
3   London large     22
4   London small     16
5  Beijing large    121
6  Beijing small     56
```

```
      city large small
1  Beijing   121    56
2   London    22    16
3 New York    23    14
```

Your turn

Use pivot_wider() to create a table with only the abany and premarsx variables in a wide format, with separate columns for each year.

```
moral_issues_wide <-
moral_issues %>%
  select(issue, year, prop) %>%
  filter(issue=="abany"|issue=="premarsx") %>%
  pivot_wider(names_from =   year, values_from =  prop)
View(moral_issues_wide)
```

	issue	1972	1974	1975	1977	1978	1980	1982	1983	1984	1985	1986	1987
1	premarsx	0.5152895	0.5430371	0.5676244	0.5948683	0.5903614	NA	0.6254181	0.6271621	NA	0.6322537	0.6329825	NA
2	abany	NA	NA	NA	0.3772819	0.3328841	0.4110953	0.3852273	0.3425743	0.3859155	0.3693450	NA	0.3924855

Fake data: # of members of national sociological associations

03-Tidy-Data.Rmd

```r
---
title: "Tidy Data"
output: html_notebook
---

```{r setup}
library(tidyverse)
library(babynames)

# Toy data
cases <- tribble(
  ~Country, ~"2011", ~"2012", ~"2
      "FR",    7000,    6900,
      "DE",    5800,    6000,
      "US",   15000,   14000,   1
)

pollution <- tribble(
       ~city, ~size, ~amount,
  "New York", "large",      23,
  "New York", "small",      14,
    "London", "large",      22,
    "London", "small",      16,
   "Beijing", "large",     121,
   "Beijing", "small",     121
)

x <- tribble(
  ~x1, ~x2,
  "A",   1,
  "B",  NA,
  "C",  NA,
  "D",   3,
  "E",  NA
)
```

1:1 Tidy Data R Markdown

```r
cases <- tribble(
  ~Country, ~"2011", ~"2012", ~"2013",
      "FR",    7000,    6900,    7000,
      "DE",    5800,    6000,    6200,
      "US",   15000,   14000,   13000
)
```
```

What are the variables?

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

What are the types of information?

- Country
- Year
- Count

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000
DE	2011	5800

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

pivot_longer()

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

1 2 3 4

cols argument

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

argument names_to: 'the names of these columns' will be converted into values in a variable called x

names_to (former column names)

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Argument values_to: 'the values in these columns' will be converted into values in a variable called x'

names_to **values_to** (former cells)

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

pivot_longer()

```
cases %>% pivot_longer(2:4, names_to = "year", values_to = "n")
```

data frame to reshape

numeric indexes of columns to collapse (or names)

name of the new key column (a character string)

name of the new value column (a character string)

pivot_longer()

```
cases %>% pivot_longer(2:4, names_to = "year", values_to = "n")
```

numeric indexes of original columns

	2	3	4
Country <chr>	**2011** <dbl>	**2012** <dbl>	**2013** <dbl>
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

pivot_longer()

```
cases %>%
  pivot_longer(c("2011", "2012", "2013"), names_to = "year", values_to = "n")
```

Column names

	2011	2012	2013
Country <chr>	**2011** <dbl>	**2012** <dbl>	**2013** <dbl>
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

pivot_longer()

```
cases %>% pivot_longer(-Country, names_to = "year", values_to = "n")
```

Everything except…

	Not Country	Not Country	Not Country
Country <chr>	**2011** <dbl>	**2012** <dbl>	**2013** <dbl>
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Your turn

Take **moral_issues_wide** and reorganise the data frame with **pivot_longer()** to look like this**:**

issue <chr>	year <dbl>	polviews <chr>	prop <dbl>
abdefect	1972	prop_lib	*NA*
abdefect	1972	prop_cons	*NA*
abhlth	1972	prop_lib	*NA*
abhlth	1972	prop_cons	*NA*
abnomore	1972	prop_lib	*NA*
abnomore	1972	prop_cons	*NA*

```
moral_issues_wide %>%
  pivot_longer(2:31,
               names_to = "year",
               values_to = "prop")
```

```
# A tibble: 60 × 3
   issue    year    prop
   <chr>    <chr>  <dbl>
 1 premarsx 1972   0.515
 2 premarsx 1974   0.543
 3 premarsx 1975   0.568
 4 premarsx 1977   0.595
 5 premarsx 1978   0.590
```