Intro to Programming with R for Political Scientists

Session 4: Data Wrangling II

Markus Freitag

Geschwister Scholl Institute of Political Science, LMU

July 19, 2021

Overview

1. Intro + R-Studio and (Git)Hub
2. Base R & Tidyverse Basics
3. Data Wrangling I
4. **Data Wrangling II**
5. Data Viz
6. Writing Functions

Workflow

- Navigate to `Session Scripts > Session 3` and open `Session_3_script.R`.
- You will see a pre-formatted Script with all the steps I do on the slides.
- Explore as you follow.
- If you have a second monitor, great! If not, split your screen.

The Data

- We will use the **parlgov** database:

> ParlGov is a data infrastructure for political science and contains information for all EU and most OECD democracies (37 countries). The database combines approximately 1700 parties, 1000 elections (9400 results), and 1600 cabinets (3900 parties).

- It's relational, i.e. consists of different tables (parties, elections, cabinets) that can be **joined** using key variables. It can also be joined with the **partyfacts** dataset that provides id's for many other datasets (e.g. CLEA, ESS).

- It makes for pretty simple examples and hence we use it.

The Data

Let's import the election data:

```
parlgov_elec <- import("http://www.parlgov.org/static/data/development-cp1252/view_election.
```

And filter for German elections:

```
parlgov_elec_de <- parlgov_elec %>% # add, e.g., _de if we want to keep our original df
  filter(country_name_short == "DEU")
```

Dealing with factor variables: forcats

- Factor variables are useful, especially for plotting and modelling.
- With `factor_recode`, we can easily recode levels:

```
parlgov_elec_de %>%
  mutate(election_type = fct_recode(election_type, # Coerces the type automatically from chr
    Bundestagswahl = "parliament",
    Europawahl = "ep"
  )) %>%
  count(election_type)
```

```
##    election_type   n
## 1     Europawahl  82
## 2 Bundestagswahl 264
```

- With `fct_recorder` we can reorder factors (which will be useful for plotting factors).

Complex conditions: if_else and case_when

- Often, we also want to manipulate variables by means of complex conditions
- We will go deeper into control flow statements next week, but here is a sneak preview for data wrangling.
- Say we want to create a variable, "family", that puts parties into some party family based on some arbitrary cutoff of the time-invariant left_right position:

```
parlgov_elec_de <- parlgov_elec_de %>%
  mutate(family = if_else(left_right > 5, "right", "left"))
```

- Vectorised if: `if_else(condition, true, false)`.

Complex conditions: if_else and case_when

- A generalised version of `if_else` is `case_when`.
- This is **very** useful:

```
parlgov_elec_de <- parlgov_elec_de %>%
  mutate(family = case_when(
    left_right <= 2.5 ~ "left",
    left_right > 2.5 & left_right < 5 ~ "centre-left",
    left_right > 5 & left_right < 7.5 ~ "centre-right",
    left_right >= 7.5 ~ "right"))
```

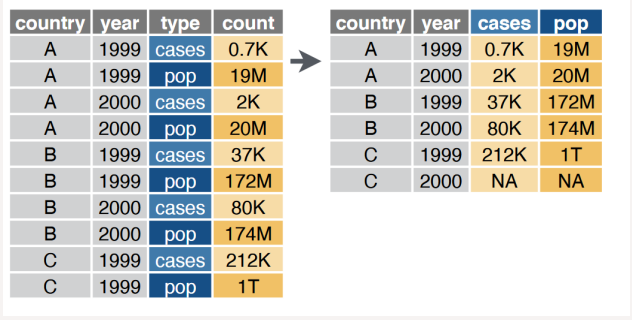
- Two-sided formula: LHS = logical test; RHS = value to assign if the test is `TRUE`.
- Values that do not fall into any of the conditions become `NA` which can be prevented by adding `TRUE ~ something` as the last argument.

tidyr: reshaping data

- Reshaping data is one of the key things you need to when cleaning/analysing data.
- **Two functions**:
  - `tidyr::pivot_wider/longer()` is for reshaping from long (wide) to wide (long).

Two main arguments:

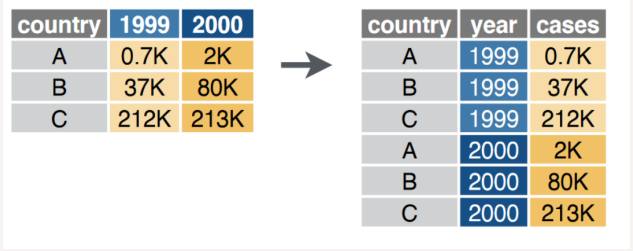
- `names_*` and `values_*`, where "*" is "to" for `pivot_wider()` and "from" for `pivot_longer()`.



country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	NA	NA

long → wide: `pivot_wider()`



country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

wide → long: `pivot_longer()`

tidyr: reshaping data

Example:

- Say, we want a table of the vote shares of all major parties for each post-WW2 parliamentary election.
- Where each row is an election:

```
wide <- parlgov_elec_de %>%
  filter(election_type == "parliament", vote_share >= 5, year(election_date) >= 1945) %>%
  select(election_date, party_name_short, vote_share) %>%
  pivot_wider(names_from = party_name_short, values_from = vote_share)
```

tidyr: reshaping data

Show 5 entries Search:

	election_date	SPD	CDU	FDP	CSU	KPD	GB/BHE	B90/Gru	PDS|Li	AfD
1	1949-08-14	29.2	25.2	11.9	5.8	5.7				
2	1953-09-06	28.8	36.4	9.5	8.8		5.9			
3	1957-09-15	31.8	39.7	7.7	10.5					
4	1961-09-17	36.2	35.8	12.8	9.6					
5	1965-09-19	39.3	38.2	9.5	9.6					

Showing 1 to 5 of 19 entries Previous 1 2 3 4 Next

tidyr: reshaping data

- We can revert back to long format:

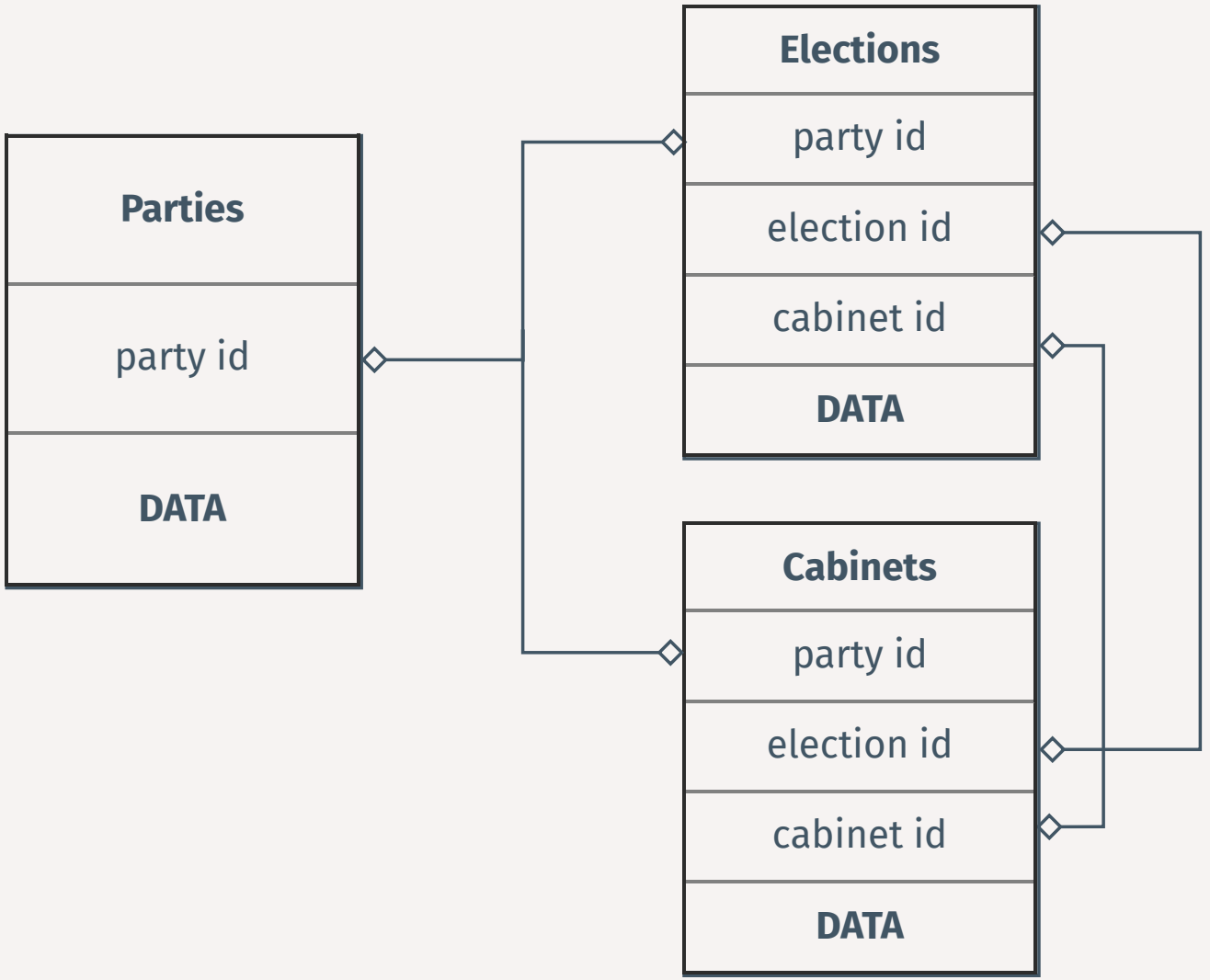
```
long <- wide %>%
  pivot_longer(!election_date, names_to = "party_name_short", values_to = "vote_share") %>%
  filter(is.na(vote_share) == FALSE) # alternatively, simply set values_drop_na to TRUE in p
head(long)
```

```
## # A tibble: 6 x 3
##   election_date party_name_short vote_share
##   <date>        <chr>                <dbl>
## 1 1949-08-14    SPD                   29.2
## 2 1949-08-14    CDU                   25.2
## 3 1949-08-14    FDP                   11.9
## 4 1949-08-14    CSU                    5.8
## 5 1949-08-14    KPD                    5.7
## 6 1953-09-06    SPD                   28.8
```

dplyr: joins

- Let's come back to the relational nature of our data...

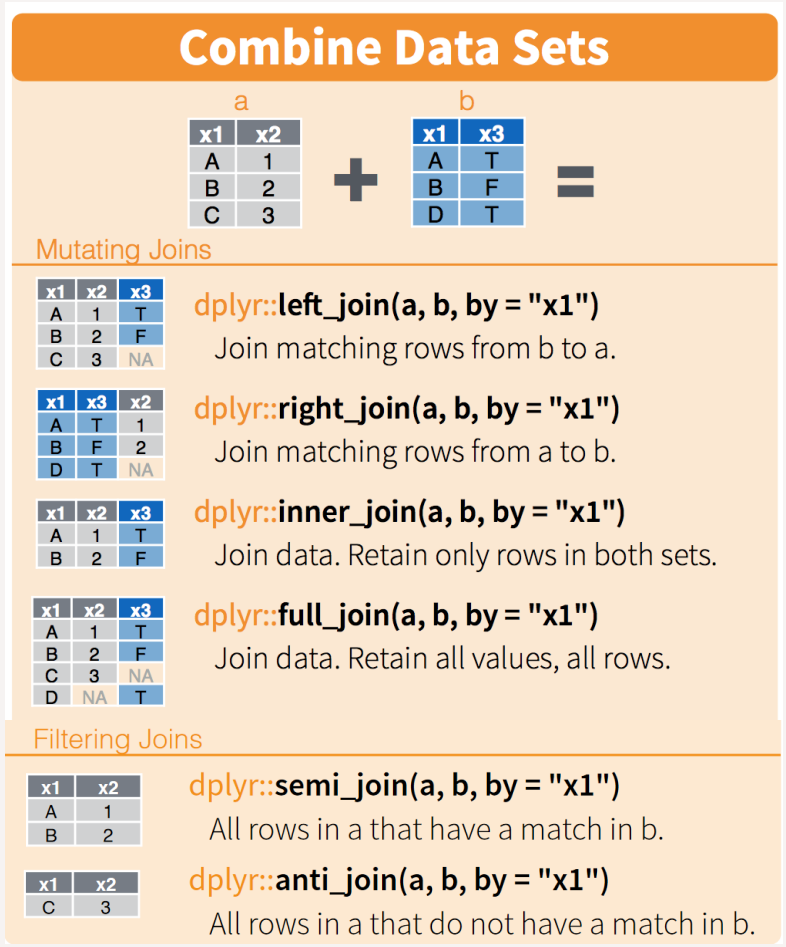
The Data


Parties
party id
DATA
Elections
party id
election id
cabinet id
DATA
Cabinets
party id
election id
cabinet id
DATA

dplyr: joins

- Let's come back to the **relational** nature of our data...
- Remember, it consists of different tables, each representing one "entity type" (c.f. the 3rd. point in Wickham's tidy data framework)
- Each table has a unique key, representing each row. This key variable is used to link/join tables
- **Suppose we want to join the party and the election table. How do we do that?**

dplyr: joins



Q Which join do we need?

dplyr: joins

We want a left join here...

```
parlgov_party <- rio::import("http://www.parlgov.org/static/data/development-utf-8/view_party.csv")

l_joined <- left_join(parlgov_elec_de, parlgov_party, by = "party_id")

head(l_joined)
```

```
##   country_name_short.x country_name.x election_type election_date vote_share
## 1                  DEU        Germany    parliament    1919-01-19      37.87
## 2                  DEU        Germany    parliament    1919-01-19      18.32
## 3                  DEU        Germany    parliament    1919-01-19      15.45
## 4                  DEU        Germany    parliament    1919-01-19      10.26
## 5                  DEU        Germany    parliament    1919-01-19       4.66
## 6                  DEU        Germany    parliament    1919-01-19       7.63
##   seats seats_total party_name_short.x
## 1   165         423                SPD
## 2    74         423                DDP
## 3    73         423                 DZ
## 4    41         423               DNVP
## 5    23         423                DVP
```

dplyr: joins

- There are multiple matching (by name) variables in both tables.
- Hence, we need to specify all keys, or let `dplyr` do its magic:

```
l_joined <- left_join(parlgov_elec_de, parlgov_party)

head(l_joined)
```

```
##   country_name_short country_name election_type election_date vote_share seats
## 1                DEU      Germany    parliament    1919-01-19      37.87   165
## 2                DEU      Germany    parliament    1919-01-19      18.32    74
## 3                DEU      Germany    parliament    1919-01-19      15.45    73
## 4                DEU      Germany    parliament    1919-01-19      10.26    41
## 5                DEU      Germany    parliament    1919-01-19       4.66    23
## 6                DEU      Germany    parliament    1919-01-19       7.63    22
##   seats_total party_name_short
## 1         423              SPD
## 2         423              DDP
## 3         423               DZ
## 4         423             DNVP
```

Alternative approaches

- For every tidyverse function ("verb"), there is, of course, **a base R way to do it**.
- There are alternatives.
- For instance, the **data.table** and **collapse** (also comes with fast versions of summary stats and models) package provide great and fast data wrangling alternatives.
- Data.table syntax is closer to the base R way of indexing/manipulating data frames. Some like that.
- Don't be dogmatic. Use whatever suits you and your context. Mix stuff.
- You can find a great comparison of `data.table` and `dplyr` **here**.

A glimpse at data.table

- Comes with its own interpretation of data frames, "data tables". Special structure to work faster.
- Looks similar to basic `df[]` indexing but with alot of twists.
- Three elements: **which observations/rows** COMMA **transformations or other functions** COMMA grouping.

Rough `dplyr` equivalent:

DT[`slice(); filter(); arrange()`, `select(); mutate()`, `group_by()`]

A glimpse at data.table

Example:

```
parlgov_elec_de %>% # add, e.g., _de if we
  filter(party_name_short == "SPD") %>%
  summarise(mean(vote_share, na.rm = T))
```

```
##   mean(vote_share, na.rm = T)
## 1                    31.12622
```

```
setDT(parlgov_elec_de)

parlgov_elec_de[party_name_short == "SPD",
```

```
## [1] 31.12622
```

Session 4 Problem Set/"Homework"

- We will not be able to do this task in class/break-out sessions.
- I highly recommend doing it at home - it's great practice!

If you like, send me your solution via mail and I will comment/give you feedback!

Next Up: Data Viz