R Tutorial at the WZB

10 - Record linkage

Markus Konrad

January 17, 2019

Today's schedule

1. Review of last week's tasks
2. Record linkage: Combining data sets
3. Reproducible workflows with RStudio

Review of last week's tasks

Solution for tasks #8

now online on https://wzbsocialsciencecenter.github.io/wzb_r_tutorial/

Record linkage

What is record linkage?

Record linkage or data joining is the process of **combining observations** in a data set \(A\) with observations in a data set \(B\) according to some **matching criteria**. Most of the time, a matching criterion is a **common identifier**.

A

id	x1
1	0.2875775
2	0.7883051
3	0.4089769
4	0.8830174
5	0.9404673

B

id	x2
1	0
2	5
3	9
4	6
5	5

A and B

id	x1	x2
1	0.2875775	0
2	0.7883051	5
3	0.4089769	9
4	0.8830174	6
5	0.9404673	5

\(A\) and \(B\) are joined by common identifier "id".

If you have multiple data sets that can be combined, you have **relational data**.

A practical example

Data from a repeated measures experiment:

```
##   person_id test_type score
## 1         1       pre     2
## 2         1      post     0
## 3         2       pre     3
## 4         2      post    10
## 5         3       pre     9
## 6         3      post     7
```

We also have "meta data" about each participant:

```
##   id age smoker
## 1  1  23   TRUE
## 2  3  42   TRUE
## 3  4  20  FALSE
```

We combine the data using a **left join** with criterion `person_id = id`:

```
##   person_id test_type score age smoker
## 1         1       pre     2  23   TRUE
## 2         1      post     0  23   TRUE
## 3         2       pre     3  NA     NA
## 4         2      post    10  NA     NA
## 5         3       pre     9  42   TRUE
## 6         3      post     7  42   TRUE
```

Notice how we introduced NAs, because participant ID 2 is missing in the "meta data". Also participant ID 4 does not appear in the result.

Combining data sets with dplyr

There are several functions in the package dplyr (contained in tidyverse) for combining data sets:

- joins: `left_join`, `right_join`, `inner_join`, `full_join`, `semi_join`, `anti_join`
- set operations: `intersect`, `union`, `setdiff`
- row or column binding: `bind_rows`, `bind_cols`

We'll have a look at the most common operations.

Data joining with dplyr

There are six join operations (of which three are commonly used).

All join operations have three parameters in common:

- a left hand side data set `a`
- a right hand side data set `b`
- a match criterion `by`

The type of join operation determines which rows and values are retained.

Left and right joins

scores

id	test_type	score
1	pre	10
1	post	5
2	pre	7
2	post	6
3	pre	1
3	post	9

personaldata

id	age	smoker
1	23	TRUE
3	42	TRUE
4	20	FALSE

`left_join(a, b, by = <criterion>)`: **always retains rows on the "left side"** and fills up non-matching rows with NAs.

```
left_join(scores, personaldata, by = c('id'))
```

```
##   id test_type score age smoker
## 1  1       pre    10  23   TRUE
## 2  1      post     5  23   TRUE
## 3  2       pre     7  NA     NA
## 4  2      post     6  NA     NA
## 5  3       pre     1  42   TRUE
## 6  3      post     9  42   TRUE
```

Left and right joins

scores

id	test_type	score
1	pre	10
1	post	5
2	pre	7
2	post	6
3	pre	1
3	post	9

personaldata

id	age	smoker
1	23	TRUE
3	42	TRUE
4	20	FALSE

`right_join(a, b, by = <criterion>)`: **always retains rows on the "right side"** and fills up non-matching rows with NAs.

How many rows do you expect for a right join between `scores` and `personaldata`?

```
right_join(scores, personaldata, by = c('id'))
```

```
##   id test_type score age smoker
## 1  1       pre    10  23   TRUE
## 2  1      post     5  23   TRUE
## 3  3       pre     1  42   TRUE
## 4  3      post     9  42   TRUE
## 5  4      <NA>    NA  20  FALSE
```

WZB Wissenschaftszentrum Berlin für Sozialforschung

Left and right joins

You can always transform a left join to a right join and vice versa.

Which of these statements are equivalent?

- `left_join(a, b, by = 'id')` and `right_join(a, b, by = 'id')`
- `right_join(b, a, by = 'id')` and `left_join(a, b, by = 'id')`
- `left_join(b, a, by = 'id')` and `right_join(a, b, by = 'id')`

Specifying match criteria

Match criteria are specified with parameter by.

experiments

group	test_type	mean_score
treat_A	pre	6.405068
treat_A	post	9.942698
treat_B	pre	6.557058
treat_B	post	7.085305
ctrl	pre	5.440660
ctrl	post	5.941420

session_info

group	test_type	n	lab
treat_A	pre	11	a
treat_A	post	14	b
treat_B	pre	12	b
treat_B	post	10	b
ctrl	pre	13	a
ctrl	post	12	a

Parameter by is a character vector with all columns that must match:

```
left_join(experiments, session_info, by = c('group', 'test_type'))
```

```
##     group test_type mean_score  n lab
## 1 treat_A       pre   6.405068 11   a
## 2 treat_A      post   9.942698 14   b
## 3 treat_B       pre   6.557058 12   b
## 4 treat_B      post   7.085305 10   b
## 5    ctrl       pre   5.440660 13   a
## 6    ctrl      post   5.941420 12   a
```

Specifying match criteria

Parameter `by` can be a named character vector like `c('x' = 'y')`. This will match `a.x` to `b.y` (`x` of left-hand side to `y` of right-hand side).

This time, the `scores` data set has an ID column named `person_id`:

scores

person_id	test_type	score
1	pre	2
1	post	0
2	pre	3
2	post	10
3	pre	9
3	post	7

personaldata

id	age	smoker
1	23	TRUE
3	42	TRUE
4	20	FALSE

We have to consider that when specifying the matching criterion:

```
left_join(scores, personaldata, by = c('person_id' = 'id'))
```

```
##   person_id test_type score age smoker
## 1         1       pre     2  23   TRUE
## 2         1      post     0  23   TRUE
## 3         2       pre     3  NA     NA
## 4         2      post    10  NA     NA
## 5         3       pre     9  42   TRUE
## 6         3      post     7  42   TRUE
```

Inner join

scores

person_id	test_type	score
1	pre	2
1	post	0
2	pre	3
2	post	10
3	pre	9
3	post	7

personaldata

id	age	smoker
1	23	TRUE
3	42	TRUE
4	20	FALSE

`inner_join(a, b, by = <criterion>)`: only retains rows that match on both sides.

How many rows do you expect for an inner join between `scores` and `personaldata`?

```
inner_join(scores, personaldata, by = c('person_id' = 'id'))
```

```
##   person_id test_type score age smoker
## 1         1       pre     2  23   TRUE
## 2         1      post     0  23   TRUE
## 3         3       pre     9  42   TRUE
## 4         3      post     7  42   TRUE
```

Full join

scores

person_id	test_type	score
1	pre	2
1	post	0
2	pre	3
2	post	10
3	pre	9
3	post	7

personaldata

id	age	smoker
1	23	TRUE
3	42	TRUE
4	20	FALSE

`full_join(a, b, by = <criterion>)`: **retains all rows for both sides** and fills up non-matching rows with NAs.

How many rows do you expect for a full join between `scores` and `personaldata`?

```
full_join(scores, personaldata, by = c('person_id' = 'id'))
```

```
##   person_id test_type score age smoker
## 1         1       pre     2  23   TRUE
## 2         1      post     0  23   TRUE
## 3         2       pre     3  NA     NA
## 4         2      post    10  NA     NA
## 5         3       pre     9  42   TRUE
## 6         3      post     7  42   TRUE
## 7         4      <NA>    NA  20  FALSE
```

WZB Wissenschaftszentrum Berlin für Sozialforschung

Semi join

scores

person_id	test_type	score
1	pre	2
1	post	0
2	pre	3
2	post	10
3	pre	9
3	post	7

personaldata

id	age	smoker
1	23	TRUE
3	42	TRUE
4	20	FALSE

`semi_join(a, b, by = <criterion>)`: A semi join is a filtering join. It returns **all observations of `a` where the criterion matches**.

```
semi_join(scores, personaldata, by = c('person_id' = 'id'))
```

```
##   person_id test_type score
## 1         1       pre     2
## 2         1      post     0
## 3         3       pre     9
## 4         3      post     7
```

→ "return all scores for which we have personal data"

Anti join

scores

person_id	test_type	score
1	pre	2
1	post	0
2	pre	3
2	post	10
3	pre	9
3	post	7

personaldata

id	age	smoker
1	23	TRUE
3	42	TRUE
4	20	FALSE

`anti_join(a, b, by = <criterion>)`: An anti join is the inverse of a semi join. It returns **all observations of `a` where the criterion does not match**.

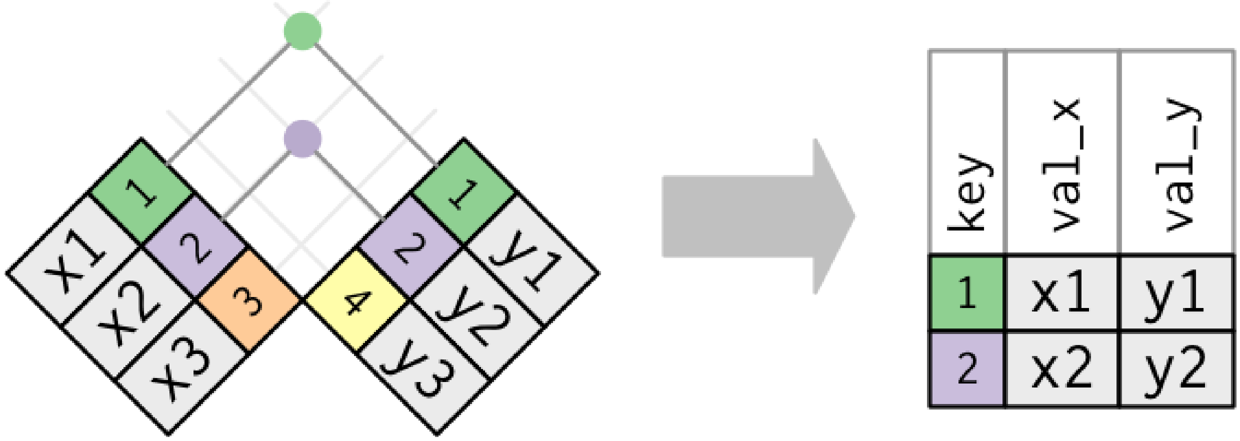
```
anti_join(scores, personaldata, by = c('person_id' = 'id'))
```

```
##   person_id test_type score
## 1         2       pre     3
## 2         2      post    10
```

→ "return all scores for which we have no personal data"

Summary of data joins

An inner join matches keys that appear in both data sets and returns the combined observations:

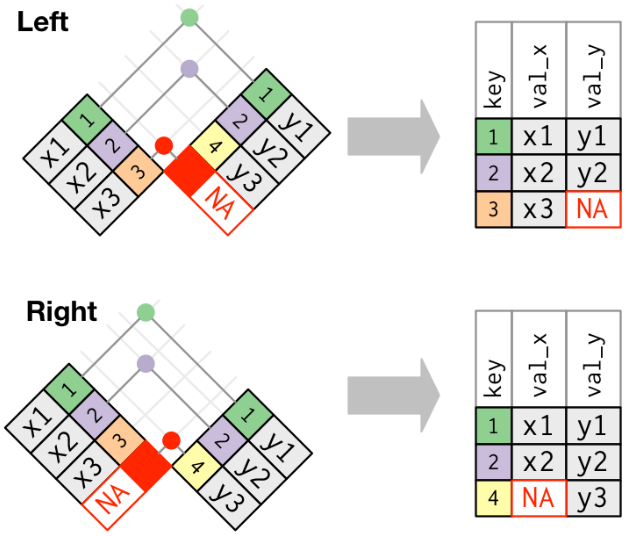


key	val_x	val_y
1	x1	y1
2	x2	y2

source: Grolemund, Wickham 2017: R for Data Science

Summary of data joins

Left and right outer joins keep all observations on the left-hand or right-hand side data sets respectively. Unmatched rows are filled up with NAs:



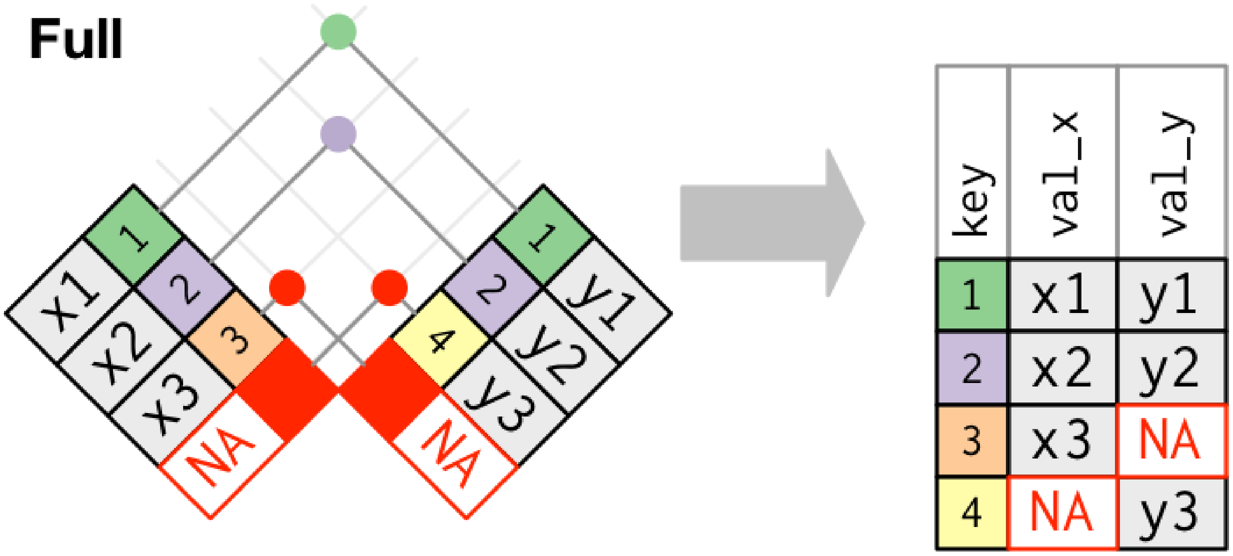
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

source: Grolemund, Wickham 2017: R for Data Science

Summary of data joins

A full outer join keeps all observations of both data sets. Unmatched rows are filled up with NAs:

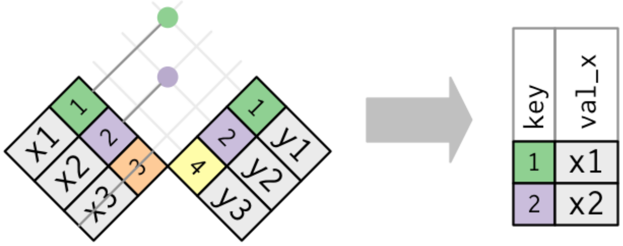


key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

source: Grolemund, Wickham 2017: R for Data Science

Summary of data joins

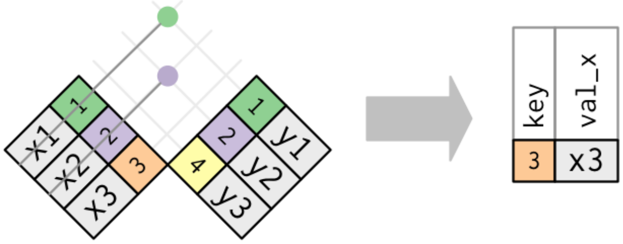
A semi join filters the left-hand data set to return only those observations, that match with the right-hand data set:



key	val_x
1	x1
2	x2

source: Grolemund, Wickham 2017: R for Data Science

An anti join is the inverse of a semi join:



key	val_x
3	x3

source: Grolemund, Wickham 2017: R for Data Science

Set operations

Set operations as defined in the dplyr package operate on data frames. They translate to the same operations as you know from maths:

- `intersect(A, B)`: \(A \cap B\)
  (observations that occur in both \(A\) and \(B\))
- `union(A, B)`: \(A \cup B\)
  (observations that occur in either \(A\) or \(B\))
- `setdiff(A, B)`: \(A \setminus B\)
  (observations that occur in \(A\) minus those that occur in \(B\))

There is no "match criterion". **All values** in a row are taken into account to match observations.

Set operations

Some examples:

A

x	y
1	1
2	1
3	2

B

x	y
1	1
2	0
3	2

```
intersect(A, B)
```

```
##   x y
## 1 1 1
## 2 3 2
```

```
union(A, B)
```

```
##   x y
## 1 2 0
## 2 3 2
## 3 2 1
## 4 1 1
```

```
setdiff(A, B)
```

```
##   x y
## 1 2 1
```

Common mistakes

1. Forgetting to specify **by**

```
left_join(scores, personaldata)
## Error: `by` required, because the data sources have no common variables
```

If there are common variables, they are used for matching by default, which is probably not what you want.

→ **always** specify `by`!

2. Not using a (named) character vector for **by**

```
left_join(scores, personaldata, by = c(person_id = id))
## Error: `by` must specify variables to join by
```

→ quotes are missing (`c('person_id' = 'id')`)

```
left_join(scores, personaldata, by = ('person_id' = 'id'))
## Error: `by` can't contain join column `id` which is missing from LHS
```

→ the little `c()` is missing to denote a vector

Common mistakes

3. Comparing the wrong types

```
A <- data.frame(id = 1:3, y = c(1, 1, 2))
B <- data.frame(id = as.factor(1:3), y = c(1, 0, 2))
inner_join(A, B, by = 'id')
## Error: Can't join on 'id' x 'id' because of incompatible types (integer /
```

Record linkage challenges

- Many things can go wrong (wrong join operation, wrong matching, etc.):
  → check numbers of rows
  → check for NAs
  → check samples
- Which type of join should I use?
  → think about which data must be retained and which data match can introduce NAs

Record linkage challenges

- Often you need to combine more than two datasets:
  → combine data sets one by one
  → check at each step
- Sometimes there's no easy way of matching observations (no common identifiers):
  → think of other strategies (other data sets, fuzzy matching, semi-automated matching)

Reproducible workflows with RStudio

Reproducible workflows

What is a reproducible workflow?

- allows **anybody** who meets your **software requirements** to reproduce your results **without programming trouble**
- **anybody** can also be your "future you"

Hence you need to provide:

- software requirements
- raw data*
- software / scripts that perform the data wrangling and analyses
- documentation

* subject to many limitations due to privacy and other considerations

Why?

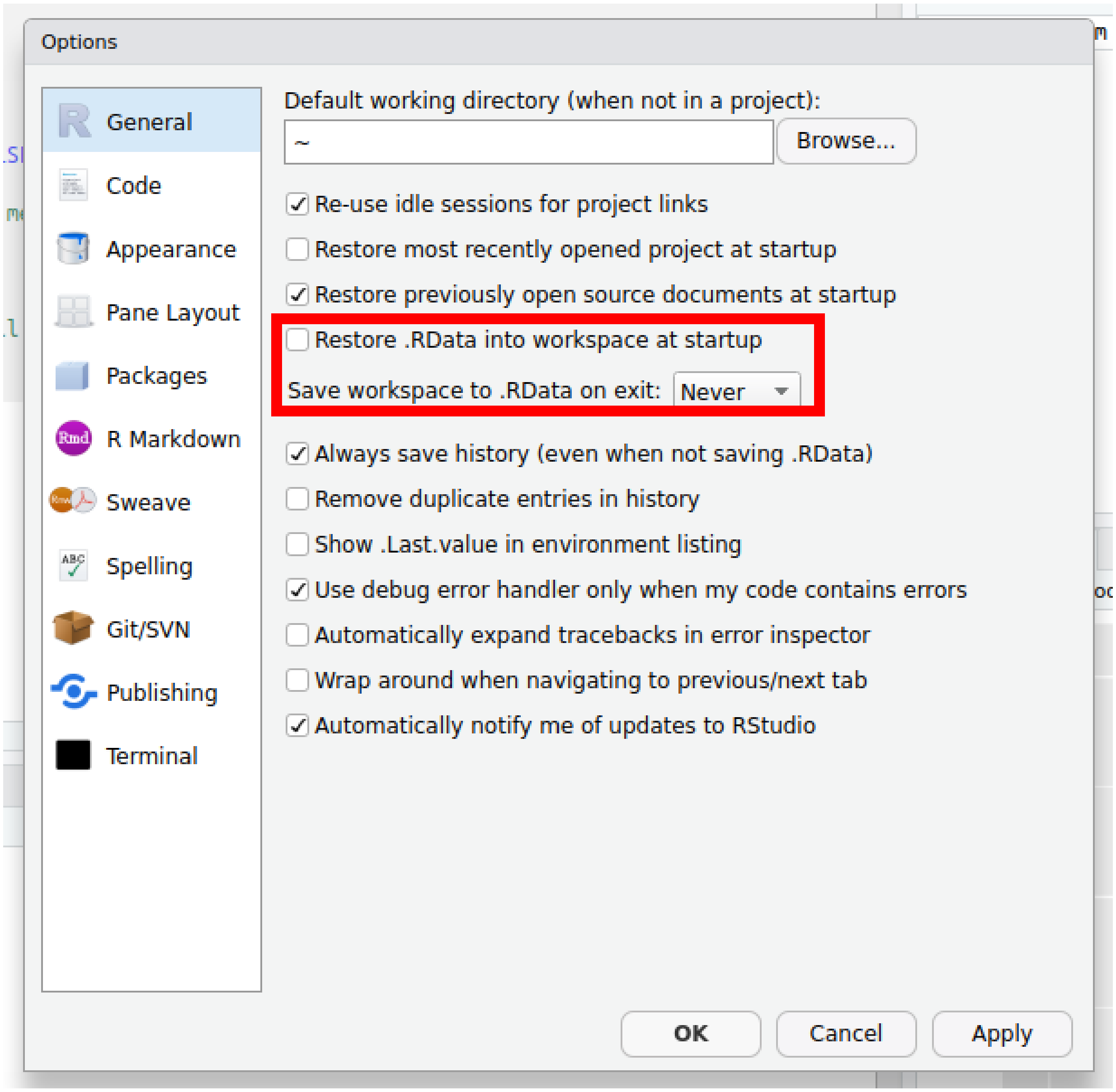
- makes sure your scripts run and are understandable when:
    - you change your computer
    - your "future you" comes back to the project after months or years
    - your collegues work with your code
    - another researcher tries to reproduce your results
- enforces transparency in the research process
- helps to find mistakes in your code
- helps in collaborative work

How?

- **set up RStudio** correctly (next slide)
- use **RStudio projects** for anything that is not just "playing around"
- **never use `setwd()`** in your scripts
- write **clean code**:
    - structure your code with indentations
    - use meaningful object and variable names
- **document** your code

Setting up RStudio

Go to Tools > Global options… → **Never** save workspace to file and never restore it on startup



Default working directory (when not in a project): ~

☑ Re-use idle sessions for project links

□ Restore most recently opened project at startup

☑ Restore previously open source documents at startup

□ Restore .RData into workspace at startup

Save workspace to .RData on exit: Never

☑ Always save history (even when not saving .RData)

□ Remove duplicate entries in history

□ Show .Last.value in environment listing

☑ Use debug error handler only when my code contains errors

□ Automatically expand tracebacks in error inspector

□ Wrap around when navigating to previous/next tab

☑ Automatically notify me of updates to RStudio

WZB Wissenschaftszentrum Berlin für Sozialforschung

Setting up RStudio

Go to Tools > Global options… → **Never** save workspace to file and never restore it on startup

Why?

Because:

- your scripts **should run without workspace**
- you don't need a workspace – the important stuff is the scripts and the data
- you don't share a workspace – you share scripts and data
- "Restart R session" will not reload data: you can start afresh
- use "Restart R session" (`CTRL+SHIFT+F10`) **often** to check if scripts work when loaded afresh

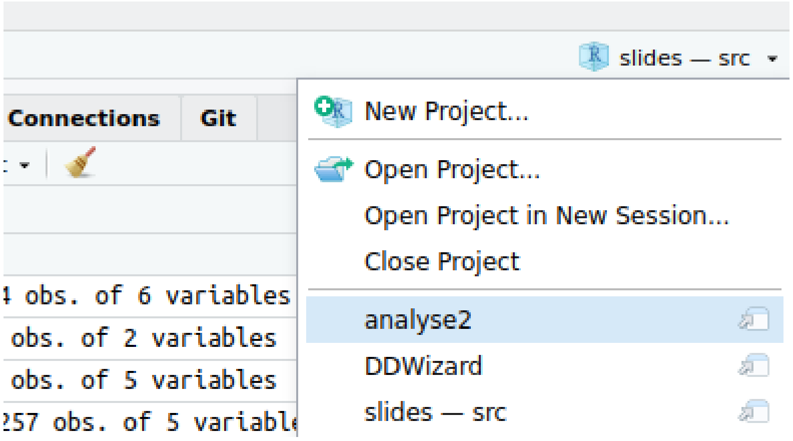
Working with RStudio projects

RStudio supports projects (File > New project …). Use them!

Why?

Because:

- loading a project will set the working directory (no need to `setwd()`!) to the root of the project
- projects can be shared (RStudio creates a `myproject.Rproj` file)
- you can easily switch between projects (top right corner)



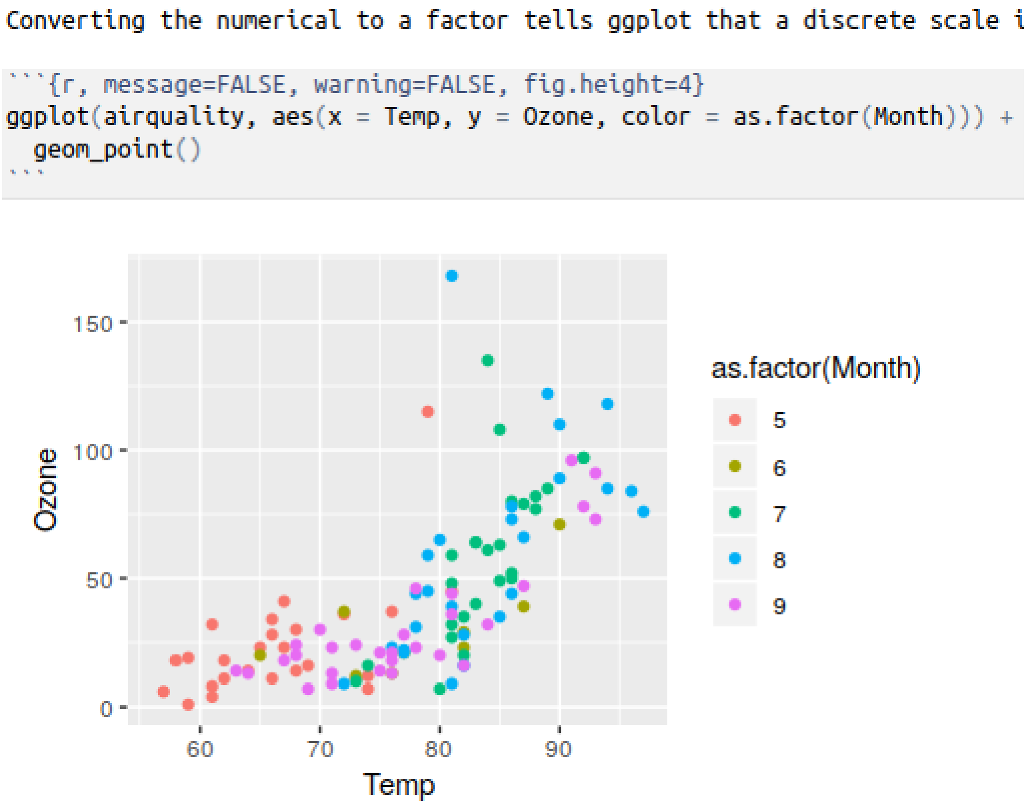
```
4 obs. of 6 variables
 obs. of 2 variables
 obs. of 5 variables
257 obs. of 5 variable
```

Working with RMarkdown

You may have noticed that the solutions to the tasks involve a **mixture of code, plots and prose**. They're **RMarkdown documents**:

- allow to write text documents with sections ("chunks") of executable R code
- output can be saved as PDF, HTML, Word, etc.

My presentations are actually RMarkdown documents:



Converting the numerical to a factor tells ggplot that a discrete scale i

~~~
```{r, message=FALSE, warning=FALSE, fig.height=4}
ggplot(airquality, aes(x = Temp, y = Ozone, color = as.factor(Month))) +
  geom_point()
```
~~~

Working with RMarkdown

Good for:

- **summaries** of your analyses / outcomes of a project
- exploratory data analysis
- small exercises

Not good for:

- complex, long running analyses
- complex programming code

→ use R scripts for that

Working with RMarkdown

Create a new RMarkdown document (short: Rmd) in RStudio with File > New file > RMarkdown ….

→ creates a sample document with explanations and links to RMarkdown documentation

Make sure that Rmd documents are **reproducible**: Select Run > Restart R and Run All Chunks.

Free book: R Markdown: The Definite Guide (Xie et al.)

WZB Wissenschaftszentrum Berlin für Sozialforschung

Concluding examples

Bad practice:

```r
setwd('C:/Research/Super Interesting Project/Analysis & more/')
X <- read.csv('cat_research_data.csv',stringsAsFactors =FALSE,
col.names= c('weight', 'age', 'length_tail'))
m <- mean(X$var2)
if (m> 8) {
print('fat cats!')
if(m > 12) {print('super fat cats!')}
}
S <- X[X$var2 > m,]
library(ggplot2)
qplot(S$var2, S$var3)
```

Concluding examples

Better practice:

```r
# super important cat research script (no need for setwd() -- RStudio projec
# author, date
library(ggplot2)  # put libraries on top

# load the data
catdata <- read.csv('cat_research_data.csv',
                    stringsAsFactors = FALSE,
                    col.names = c('weight', 'age', 'length_tail'))

# calculate mean weight and create subset with obs. where weight > mean weig
mean_weight <- mean(catdata$weight)
above_mean <- catdata[catdata$weight > mean_weight,]

if (mean_weight > 8) {  # just to show nested indentation
  print('fat cats!')
  if(mean_weight > 12) {
    print('super fat cats!')
  }
}

# scatter plot of above mean data with weight against length of cat's tail
qplot(above_mean$weight, above_mean$length_tail)
```

Tasks

See dedicated tasks sheet on the tutorial website.