Data Workflow

The Restaurant at the End of the tidyverse

Frank Loesche

16 March 2017

Synopsis

I was asked for *life-saving tips to make your life easier when using the computer*

- Workshop with Bodo Winter: stats OK, coding difficult
- Coding Club
  - different levels of knowledge, specific topics
  - scripting, based on personal preferences
- Observations
  - mixture of copy & past from google etc
  - unstructured workflow

Gap

- consistent workflow from raw data to publication

Framework

Reproducible (science)

1. document every step
2. avoid (analytical) discontinuity
3. never copy & paste
4. automate where possible (**DRY**, **D**ont **R**epeat **Y**ourself)
5. consider everything not saved in a file as lost
6. single source of truth (only one copy of data files)

tidyverse

> *The tidyverse is a collection of R packages that share common philosophies and are designed to work together.* [1]

collection of packages:

ggplot, tibble, tidyr, dplyr, haven, readr, string, forcats, broom, purrr...

common "philosophy" / grammar

Basic Idea: **what** %>% **how** -> **result**

```
df %>% filter(Age > 25) %>% select(Name) -> drink
```

[1] from http://tidyverse.org

Workflow

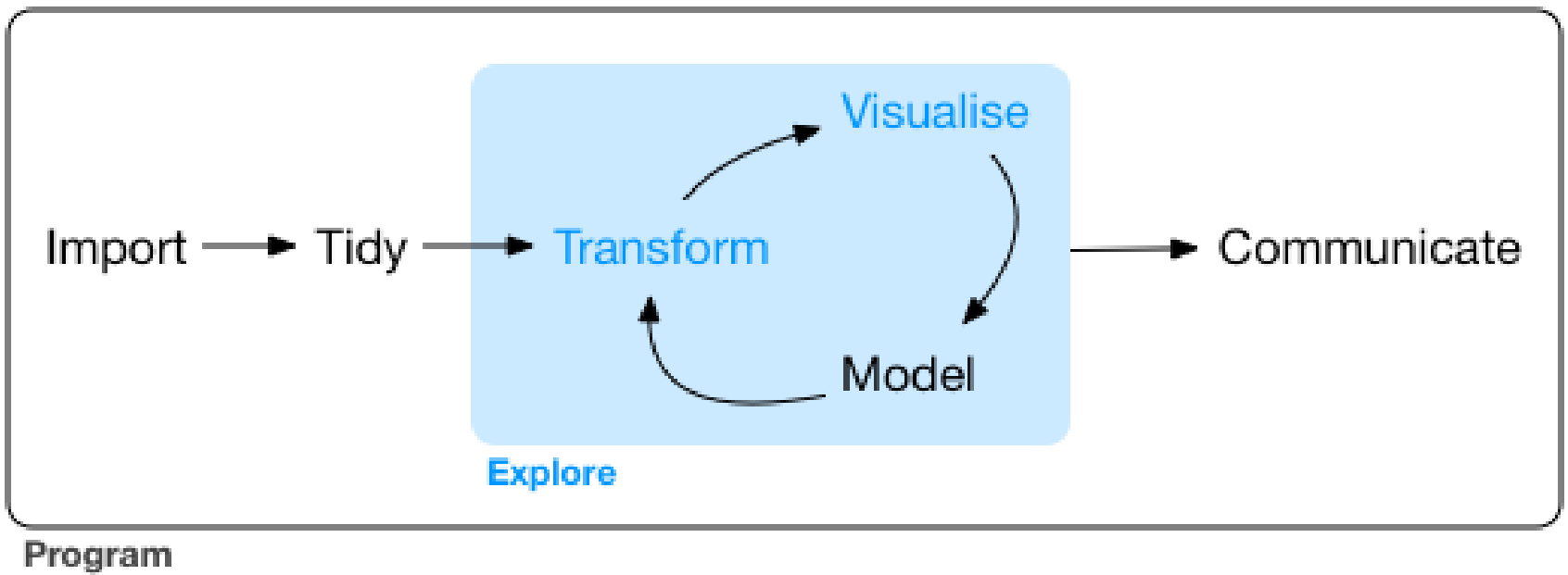


Figure 1: Workflow for data exploration [2]

[2] from *R for Data Science*, http://r4ds.had.co.nz

Import

Goal

- have *some* data in R (data frame, tibble)

Problem

- 1000s of input sources

Challenge

- data might update at any time during the data exploration

Import

Data Sources

- experimental data
- questionnaires

File formats

- CSV, TSV... (readr)
- SAS, SPSS, Stat (haven)
- MATLAB
- Excel (readxl)
- data bases, SQL

Import

File Structure

At the beginning:

```
~frank/OHBDS/
             /data/raw-data.csv
             /Journal.Rmd
             /functions.R
```

- `data/*`: input
- `Journal.Rmd`: human centric document, also see previous workshop
- `functions.R`: functional units, code centric

Import

Load data

Import CSV as most people know it [3]

```
data <- read.csv("data/raw-data.csv")
# Sanity check 1, should show ~30000 rows
nrow(data)
```

```
## [1] 29544
```

Yay!

[3] the next few slides are illustrating how work with data in a base R environment. Continue reading at "Import - the tidyverse way" if you don't need a refresher.

Debug

Where did it go wrong?

- revisit data

```
1|Q1  Q2  Q3  Q4...E1  E2  E3...others
2|4 1   5   1   4   2   5... 5423  2313...
```

Result

- has no comma, as in Comma Separated Values (CSV)
- has "tabs" as in Tab Separated Values (TSV)

Refactor code

- working snippet of code into functions
- move `import()` to `functions.R`

```
import <- function() {
  data <- read.csv("data/raw-data.csv", sep = "\t")
  return(data)
}
raw_data <- import()
```

Journal.Rmd

```
source("functions.R")
raw_data <- import()
```

Conclusion

```
raw_data <- import()
```

Good

- `raw_data` describes what is in the data
- `import()` describes what the function does
- only one run of `Journal.Rmd` to restore current R environment

Potential problems

- generic function names (e.g `import()`, `arrange()`, `filter()`) overwrites existing ones

Solution

- avoid name conflicts with prefix, eg `fl_import()`
- or be polite: `please_import()` :-)

Tidy - the 2nd step in data analysis

What is tidy data?

- meet semantics of your data (aggregation level)
- one observation per row
- one column per variable

example `raw_data`

- does not have Participant ID, depends on current order

Aim

- additional Column “Participant” with “PID1” . . . “PID29544”

Recap

- data imported into `raw_data`
- `temp_data` has additional column "Participant" with unique IDs

Good

- writing functions for each step
  - reduces complexity
  - separation of concerns (coding details in `functions.R`, statistical data analysis in `Journal.Rmd`)
- sensible names for variable and functions
  - saves writing verbose documentation
  - "code as prose"

Recap

Bad?

- lots of different syntax
  - `read.csv` requires argument `sep=` for TSV files
  - `for` loop *feels* wrong
  - is `data[column, row]`, `data[row, column]` or `data$row[column]` correct?
  - wasn't there a function for renaming columns?
  - where is the documentation about data manipulation when I need it?
- slow
  - `please_add_participant_id()` already takes 6 seconds, just to add 30000 strings
  - how about more complex manipulation?
  - how about different aggregation level?
- Why haven't I mentioned tidyverse yet?

Import the tidyverse way

- many different libraries
- syntax very similar
- learn once, apply often

Different Libraries:

- `haven` [4]
  - for SASS, SPSS, Stata
- `readxl` [5]
  - for Excel files
- `readr` [6]
  - for *rectangular* data, eg. csv, tsv, fixed width files...
  - string stays string (not factor)

---

[4] http://haven.tidyverse.org/
[5] http://readxl.tidyverse.org/
[6] http://readr.tidyverse.org/

Import - example

functions.R

```
please_import <- function() {
  read.csv("data/raw-data.csv", sep = "\t")
}
```

to

```
please_import <- function() {
  library(readr)
  read_tsv("data/raw-data.csv")
}
```

Import: ✓ (using tidyverse)

Tidy

`dplyr` & `tidyr` – data manipulation

- good documentation in data wrangling cheatsheet!
- some example functions to follow

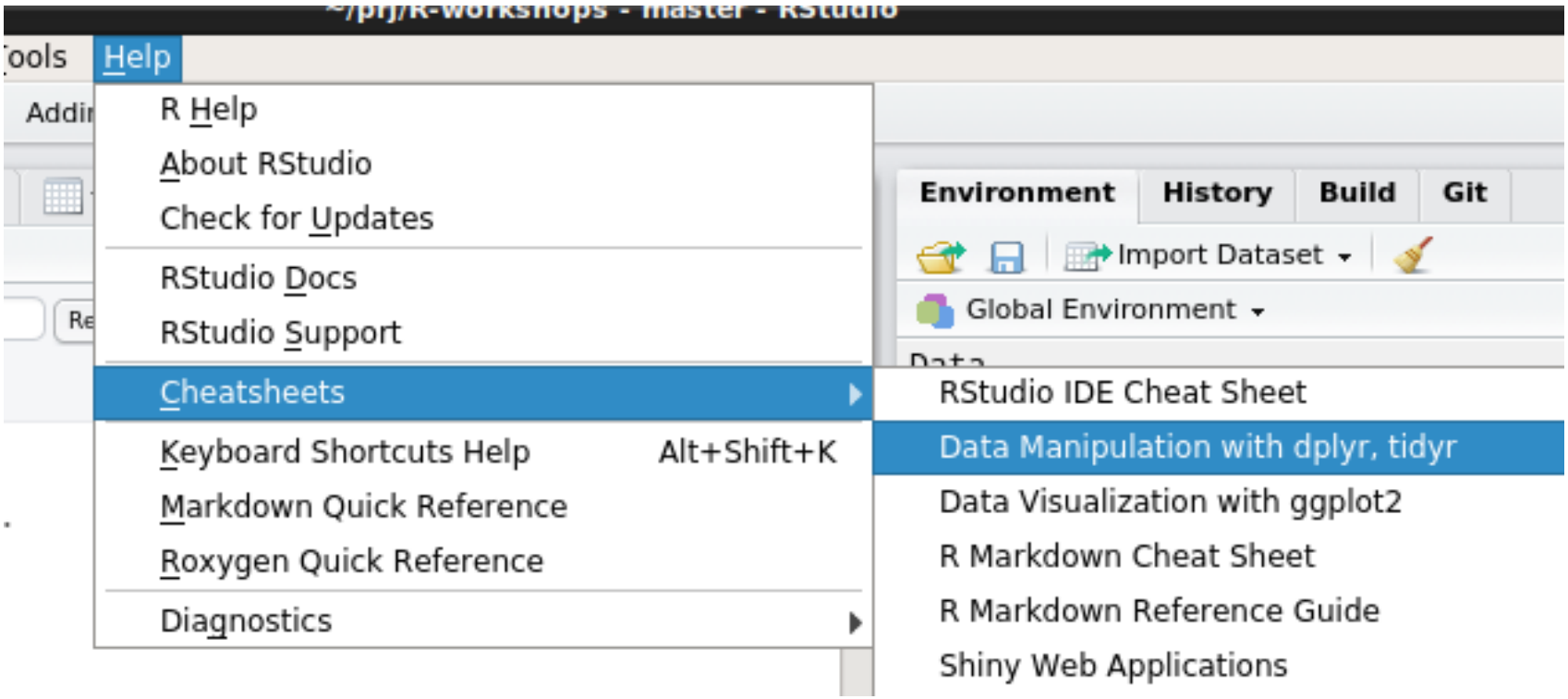


Figure 2: Your new best friend

%>% - pass data between functions through the pipe [7]

- `data %>% do_something()` is the same as `do_something(data)`
- better readability, with "verbs"
  - `data %>% sqrt() %>% add(5) %>% arrange()`
  - `arrange(add(sqrt(data), 5))`
- implicitely last verb always `print()` unless data is assigned
  - `data %>% sqrt()` is the same as `data %>% sqrt() %>% print()`
  - `df %>% filter(Age > 25) %>% select(Name) -> drink` doesn't print anything
  - `df %>% filter(Age > 25) %>% select(Name)` prints something

[7] https://github.com/tidyverse/magrittr

`filter()` - filter for an observation

```
temp_data %>% filter(Participant == "PID17")
```

```
## # A tibble: 1 × 47
##      Q1    Q2    Q3    Q4    Q5    Q6    Q7    Q8
##   <int> <int> <int> <int> <int> <int> <int> <int>
## 1     1     1     3     4     2     2     2     3
## # ... with 39 more variables: Q9 <int>,
## #   Q10 <int>, Q11 <int>, Q12 <int>, Q13 <int>,
## #   Q14 <int>, Q15 <int>, Q16 <int>, Q17 <int>,
## #   Q18 <int>, Q19 <int>, Q20 <int>, E1 <int>,
## #   E2 <int>, E3 <int>, E4 <int>, E5 <int>,
## #   E6 <int>, E7 <int>, E8 <int>, E9 <int>,
## #   E10 <int>, E11 <int>, E12 <int>, E13 <int>,
## #   E14 <int>, E15 <int>, E16 <int>, E17 <int>,
## #   E18 <int>, E19 <int>, E20 <int>,
## #   country <chr>, introelapse <int>,
## #   testelapse <int>, wrapupelapse <int>,
```

select() - select a column

```
temp_data %>%
  filter(Participant == "PID17") %>%
  select(Participant, country) -> p17country
print(p17country)
```

```
## # A tibble: 1 × 2
##   Participant country
##         <chr>   <chr>
## 1       PID17      HU
```

mutate() - add columns

```
temp_data %>%
  mutate(mean_Q = (Q1 + Q4) / 2) %>%
  filter(Participant == "PID17") %>%
  select(Participant, country, Q1, Q4, mean_Q)
```

```
## # A tibble: 1 × 5
##   Participant country    Q1    Q4 mean_Q
##         <chr>   <chr> <int> <int>  <dbl>
## 1       PID17      HU     1     4    2.5
```

summarise() – aggregate

- summarise (eg. 30000 observations) into one observation

```
temp_data %>%
  summarise(
    mean_Q1 = mean(Q1),
    meanQ2 = mean(Q2),
    elements = n()
  )
```

```
## # A tibble: 1 × 3
##   mean_Q1 meanQ2 elements
##     <dbl>  <dbl>    <int>
## 1   2.926  2.608    29544
```

Better additional column

functions.R

```
please_add_participant_id <- function(data_in) {
  data_out <- data_in
  for (i in 1:nrow(data_out)) {
    data_out[i, "Participant"] <- paste0("PID", i)
  }
  return(data_out)
}
```

to

```
please_add_participant_id <- function(data_in) {
  data_in %>%
    mutate(Participant = paste0("PID", 1:n()))
}
```

recap

- tidyverse much faster
  - old version: 5 seconds for 30000 observations
  - new version: 0.015 seconds for 30000 observations
- easy to read
- Ceci n'est pas un pipe
- next step: tidy data

```
temp_data <- please_import() %>%
  please_add_participant_id()
```

Current status

functions.R

```
please_import <- function() {
  read_tsv("data/raw-data.csv")
}
please_add_participant_id <- function(data_in) {
  data_in %>%
    mutate(Participant = paste0("PID", 1:n()))
}
```

Journal.Rmd

```
library(tidyverse)
source("functions.R")
temp_data <- please_import() %>%
  please_add_participant_id()
```

tidy

goal (reminder: what is *tidy* data)

- one column per variable
- one observation per row
- represent semantic structure of the data

observation

- Q1. . . Q20 are answers to different questions, E1. . . E20 the according reaction times
- each question is one observation (answer, response time)
- country, introelapse etc are observations per participant

Suggestion

- 2 aggregation levels: Participant data, Question data

tidy participant data

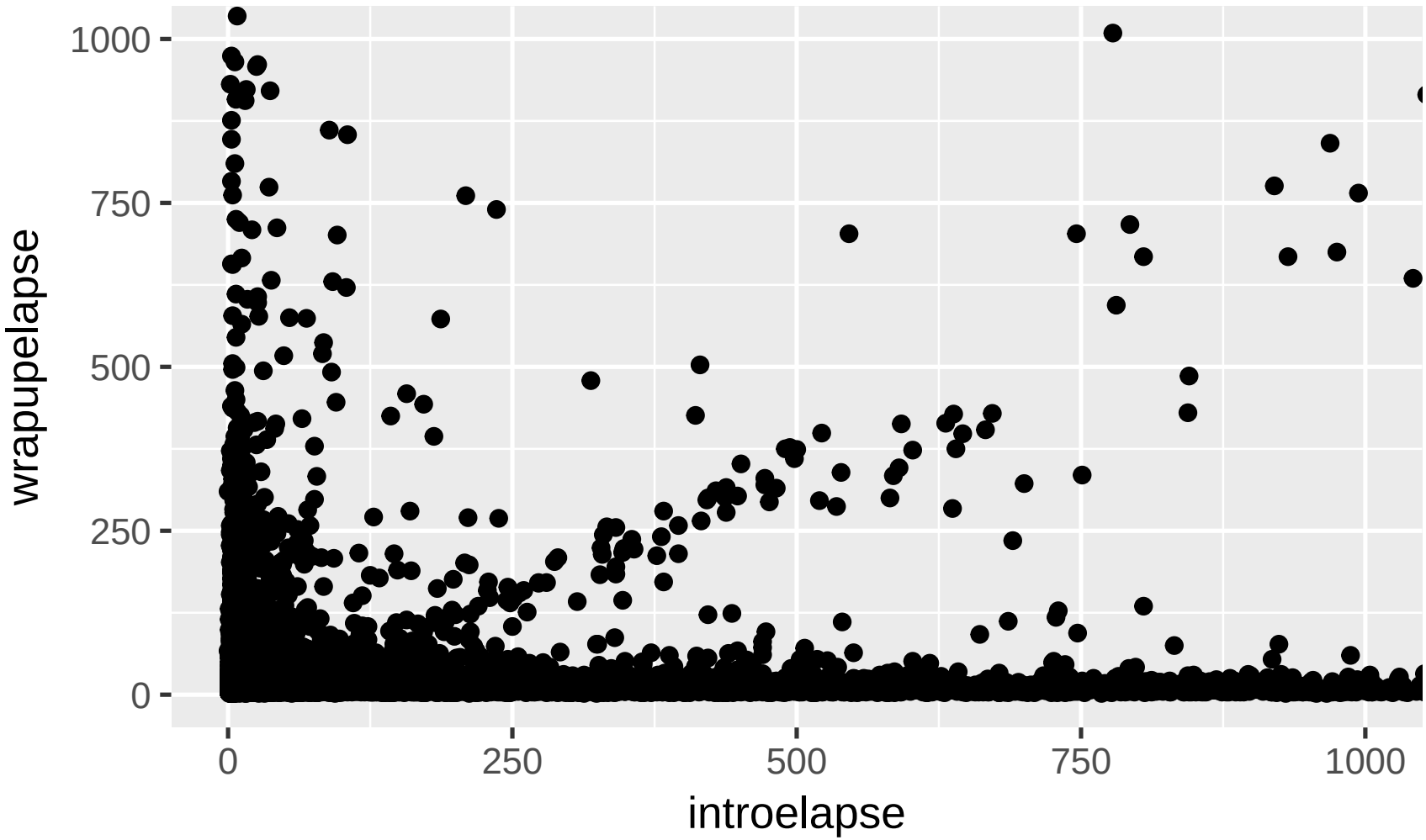
functions.R

```
please_extract_participant_data <-
  function(data_in){
    data_in %>%
      select( -starts_with("Q")) %>%
      select( -starts_with("E")) %>%
      select(Participant, everything())
  }
```

Journal.Rmd

```
participant_data <- temp_data %>%
  please_extract_participant_data()
```

```r
ggplot(participant_data) +
  geom_point(aes(introelapse, wrapupelapse)) +
  coord_cartesian(
    xlim = c(0, 1000),
    ylim = c(0, 1000))
```



tidy question data (extract)

functions.R

```
please_extract_questions_data <-
  function(data_in){
    data_in %>%
      select(Participant,
             starts_with("Q"),
             starts_with("E"))
  }
```

tidy question data (wide to long)

functions.R

```
please_tidy_question_data <-
  function(data_in){
    data_in %>%
      gather(variable, value, -Participant) %>%
      mutate(
        question = parse_number(variable),
        variable = substr(variable, 1, 1)) %>%
      spread(variable, value) %>%
      rename(Answer = Q, Reaction_time = E)
  }
```

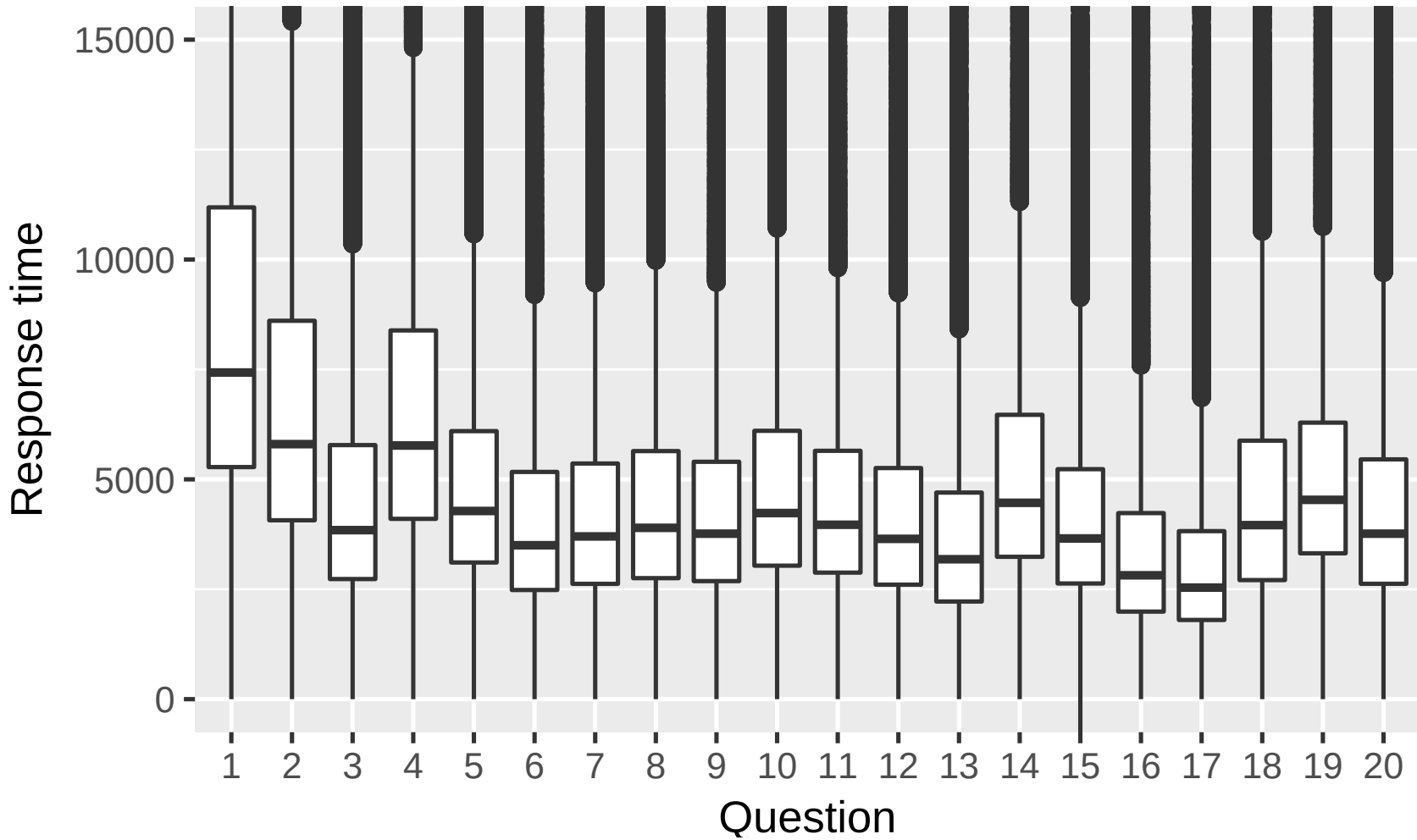
Documentation

Journal.Rmd

The only thing that needs to go in the Journal is a collection of functions whose names describe what is happening with the data. The details of implementation is *hidden* inside `functions.R`

```
tidy_questions <-
  please_import() %>%
  please_add_participant_id() %>%
  please_extract_questions_data() %>%
  please_tidy_question_data()
```

```
ggplot(tidy_questions ) +
  geom_boxplot(aes(factor(question), Reaction_time)) +
  labs(x = "Question", y = "Response time") +
  coord_cartesian(ylim = c(0, 15000))
```



between aggregation levels

```
combined_data_on_question_level <-
  full_join(
    participant_data,
    tidy_questions,
    by = "Participant"
  )
print(combined_data)
```

between aggregation level

```
combined_data_on_participant_level <-
  tidy_questions %>%
    group_by(Participant) %>%
    summarise(
      mean_answer = mean(Answer),
      mean_reaction_time = mean(Reaction_time)) %>%
    left_join( participant_data , by = "Participant")
```

This slide is intentionally left blank for your data manipulation and plots :-)

Recap

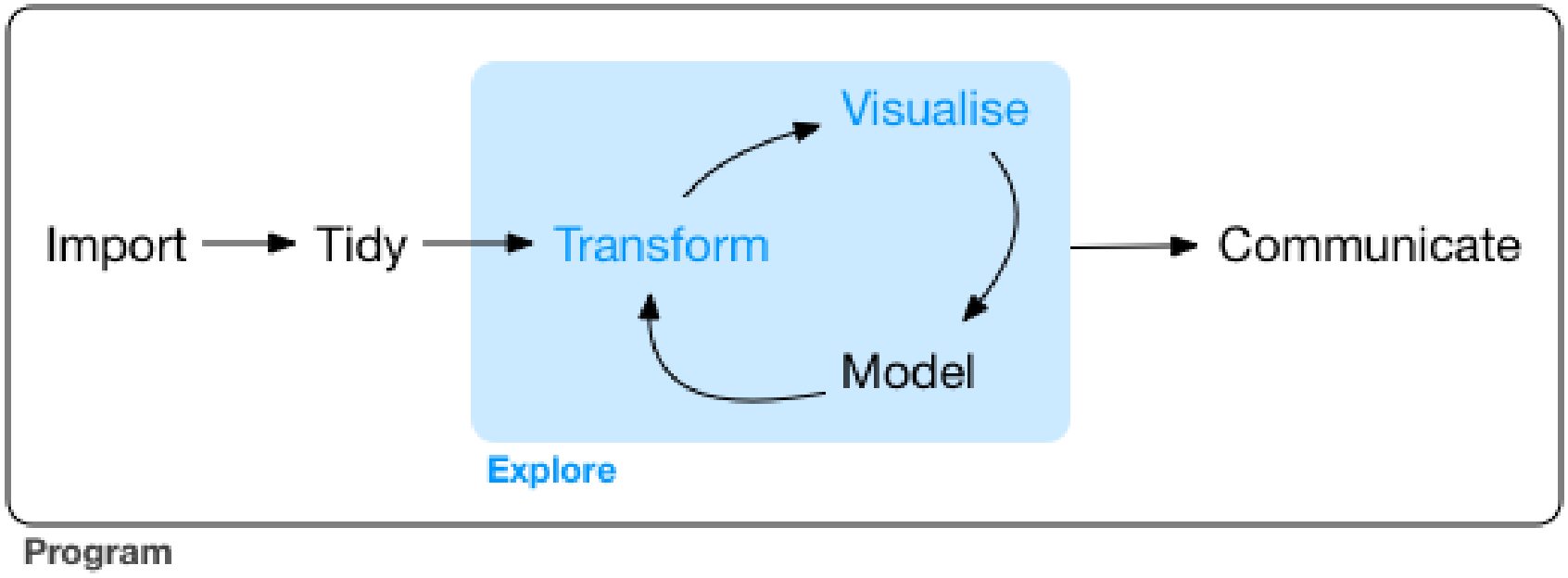


Figure 3: Workflow for data exploration

- this document shows: import, tidy, (transform, visualise)
- programming style: separate levels of concern (code vs analysis)

Thanks

More examples at http://github.com/floesche/R-workshops

Also have a look at the previous workshop
"Raw Text to Camera-Ready"

Sorry, there was no restaurant. But you should be able to understand this much better now:

```
df %>%
  filter(Age > 25) %>%
  select(Name) -> drink
```