Introduction to `teal`

Stefan Thoma

2023-03-01T00:00:00+00:00

Table of contents

Preface

This is a tutorial for creating `teal` applications.

Who is this for

This tutorial is aimed at data scientists / analysts who want to write their first `teal` application. Some R knowledge is required although no exposure to `shiny` is expected.

The `teal` package aims to make the creation of (teal-specific) shiny applications intuitive — without the need to understand how shiny works exactly. Shiny knowledge is useful once you want to either modify an application, or write your own teal module from scratch.

How is it structured

All chapters in this book follow a common structure: First, the learning goals of a chapter is defined. We then go through the step-by-step process of using specific `teal` modules. In each chapter we combine what we learned into one code chunk with which you can create a working teal application (*toy app*). As every `teal` app requires data and meta-data, the *toy app* created in the introduction will be the fundamental building block of the apps in the subsequent chapters. In the *toy apps* created in each chapter you should be able to understand every line of code. The code for a more developed app (but built for the same purpose) and a deployed instance of that app is linked at the end of each chapter. What you learned in the chapter should give you a good understanding of how the more developed application works.

What you will learn

You will learn the structure of `teal` applications and the structure of the corresponding code.

After this tutorial you should be able to create your own application for your data based on existing `teal` modules. This requires an awareness of — some — available modules, where to find them, their intent, how to use them, and where to find the rest. Further, you should be able to customize existing modules to suit your specific needs.

Where can you find more

The teal gallery repository on GitHub hosts the code of many demo `teal` applications as well as links to the apps. Feel free to check them out - or submit your own app.

1 Introduction

At the end of this chapter you should be able to

- understand the structure of `teal` apps
- understand the utility of `teal` apps
- determine whether `teal` apps are useful for *you*

1.1 Structure of `teal` apps

`teal` is a shiny-based modular framework. It provides an interactive user interface allowing users to customize tables and visualizations facilitating the exploration and analysis of clinical trial data.

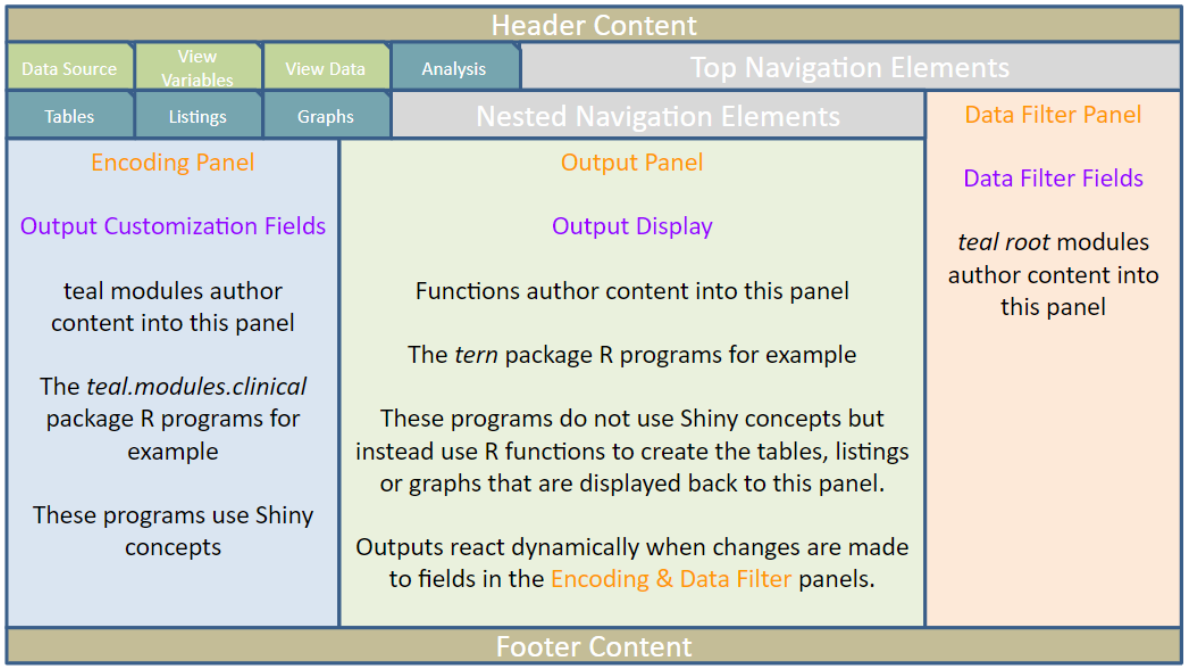
Dynamic analysis and display control, data filtering and code reproducibility are included in each module.

`teal` modules are laid out in three panels.

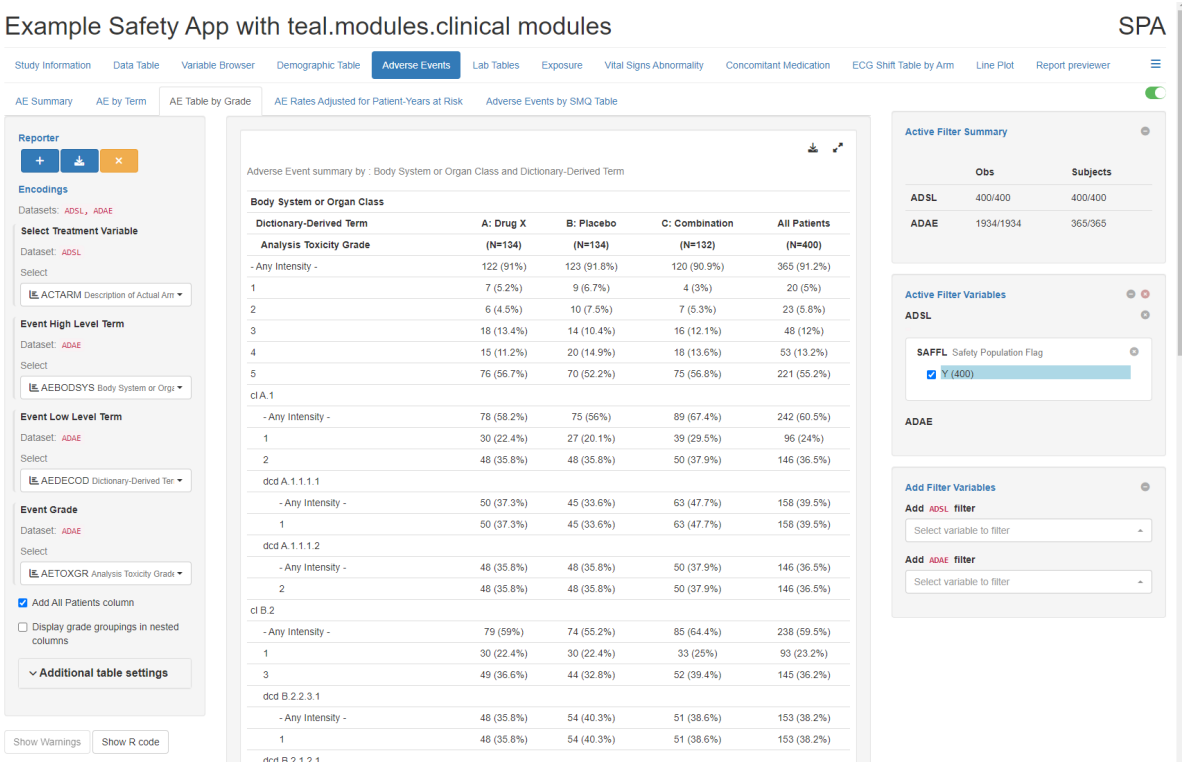
- The `Left Panel`, referred to as the encoding panel, includes elements used to adjust the output interactively.
- The `Center Panel`, referred to as the output panel, where the tables and visualizations are displayed.
- The `Right Panel`, referred to as the data filter panel, includes elements used to subset the data interactively.

The layout is pre-set within each module. However, user can decide overall tab order and nesting. See Figure 1.1 for the typical teal app structure.

`teal` is part of the pharmaverse and is particularly built upon tern and rtables.



(a) diagram



(b) example

Figure 1.1: typical teal app layout

1.2 Modules

`teal` modules are modular building blocks for your `teal` app. They implement specific app functionalities, such as outlier exploration, data visualizations, and survival analysis. Although modules can be created from scratch, many `teal` modules have already been released and can be found in the following packages:

- `teal.modules.general`: general modules for exploring relational/independent/CDISC data
- `teal.modules.clinical`: modules specific to CDISC data and clinical trial reporting
- `teal.modules.hermes`: modules for analyzing MultiAssayExperiment objects

1.3 Reproducibility

Most `teal` modules implement the `teal reportR` functionality. For these modules you can add any given output (i.e. a specific data table or graph) as a `card` to a `report`. Conceptually, this works like a shopping cart to which the app-user can add outputs generated during their stay on the application across all modules. For each `card`, `reportR` will produce a .pdf and a .html file in addition to a stand-alone R script that reproduces the specified output. This is really helpful to make `teal` outputs reproducible and comprehensible.

1.4 Conclusion / use cases

`teal` applications allow individuals with little or no programming experience to extract insights from data, making it accessible to a broad audience. The package is particularly aimed at statisticians and statistical programmers in the pharmaceutical industry working with standardized (CDISC) data-sets. However, it is not limited to standardized data-sets and could be used in other fields as well. One of the key features of teal is its ability to ensure reproducibility through the generation of R code for the selected outputs. This is an important consideration for maintaining the accuracy and reliability of results. At present, teal is not intended for regulatory submission, but it may have potential for this purpose in the future.

2 Data app

The App created at the end of this chapter will be the basic building block of all subsequent chapters in this book. The structure of the app code will remain consistent throughout the book — make sure you understand each line of code of the *toy* application at the end.

At the end of this chapter you should be able to

- understand the code structure of `teal` apps
- understand the workflow to create a `teal` app
- install and load the relevant packages
- write your own app that displays data

2.1 Your first app

As is generally the case when you start a project, you should have an approximate idea of what you would like to achieve. In this chapter, the aim is to create an app that visualizes variables of three connected study data-sets. The app should also display meta-data guiding users through the app itself.

In most applications where `teal` apps are useful, the above should be included. Therefore, all subsequent chapters are built upon the toy app created in this chapter.

2.2 Code structure

- First, you want to define the project metadata and load the required packages and the data. This can include the author name, the study name, the molecule name, etc.
- Then, we have a section where we can define parameters for the app. Parameters can be simple (e.g. `background_color <- "gray"`) but they can also be short code snippets that would otherwise often be repeated throughout the application, e.g. pre-selected variables of interest.
- After that, we define the app using the `init()` function. This function takes data, modules, and html input and returns a list (usually named `app`) containing the user interface (the `ui` object) and back end (the `server` object) of the application.

- In the final step we instantiate the app by calling `shinyApp(app$ui, app$server)`.

2.3 Setup

To get started, we first need to install the `teal` packages from GitHub. This works best if you have a GitHub account to which you are connected in your current R environment using a Personal Access Token (PAT).

> **Connect to GitHub**
>
> There are different ways to connect RStudio to GitHub. I find it easiest using the R-packages `credentials` and `usethis` In any case: You need a GitHub account.
>
> ```
> # first, you need to install `usethis`
> install.packages("usethis")
>
> # Add authentication information:
> # Use your GitHub user name and associated email address.
> usethis::use_git_config(
>   user.name = "StefanThoma",
>   user.email = "stefan.thoma@roche.com"
> )
>
> # feel free to write a more appropriate description.
> # default scopes of the token should work fine.
> usethis::create_github_token(description = "connect this session to github")
> # make sure to store your token in a save place, e.g. a password manager.
> # I currently use bitwarden, which is free and open source.
>
> # finally, add your token to the machine. Use:
> gitcreds::gitcreds_set()
> # And enter the created GH-Token in the R console when prompted
> ```

```
if (!require("remotes")) install.packages("remotes")

# install teal
remotes::install_github("insightsengineering/teal@*release")

# install teal.modules.general
remotes::install_github("insightsengineering/teal.modules.general@*release")
```

```
# install tern mmrm (required for teal.modules.clinical)
remotes::install_github("insightsengineering/tern.mmrm@*release")

# install teal.modules.clinical
remotes::install_github("insightsengineering/teal.modules.clinical@*release")

# install teal.modules.hermes
remotes::install_github("insightsengineering/teal.modules.hermes@*release")
```

In this tutorial we will be working with realistic looking but simulated data.

We first load two synthetic data sets. `random.cdisc.data::cadsl` is a Subject Level Analysis Dataset (ADSL) with one record (row) per subject. The unique identifier per subject is stored in the variable `USUBJID`. According to CDISC the main purpose of ADSL is to provide a "(…) source for denominators for populations of interest, stratification variables, and other important subject subgroups".

We can now load the three data-sets:

Data Structure for Adverse Event Analysis

Data-set name	Explanation	Purpose
ADSL	Analysis data-set subject level	Filtering
ADAE	Analysis data-set for adverse events	Adverse events
ADTTE	Analysis data-set for time to event	Efficacy

```
ADSL <- random.cdisc.data::cadsl
ADAE <- random.cdisc.data::cadae
ADTTE <- random.cdisc.data::cadtte
```

Let's look at the first three rows of the data-frames.

```
list(ADSL, ADAE, ADTTE) %>% map(head, 3)
```

```
[[1]]
# A tibble: 3 x 56
  STUDYID USUBJID     SUBJID SITEID   AGE AGEU  SEX   RACE  ETHNIC COUNTRY DTHFL
  <chr>   <chr>       <chr>  <chr>  <int> <fct> <fct> <fct> <fct>  <fct>   <fct>
1 AB12345 AB12345-CH~ id-128 CHN-3     32 YEARS M     ASIAN HISPA~ CHN     Y
2 AB12345 AB12345-CH~ id-262 CHN-15    35 YEARS M     BLAC~ NOT H~ CHN     N
3 AB12345 AB12345-RU~ id-378 RUS-3     30 YEARS F     ASIAN NOT H~ RUS     N
```

```
# ... with 45 more variables: INVID <chr>, INVNAM <chr>, ARM <fct>,
#   ARMCD <fct>, ACTARM <fct>, ACTARMCD <fct>, TRT01P <fct>, TRT01A <fct>,
#   TRT02P <fct>, TRT02A <fct>, REGION1 <fct>, STRATA1 <fct>, STRATA2 <fct>,
#   BMRKR1 <dbl>, BMRKR2 <fct>, ITTFL <fct>, SAFFL <fct>, BMEASIFL <fct>,
#   BEP01FL <fct>, AEWITHFL <fct>, RANDDT <date>, TRTSDTM <dttm>,
#   TRTEDTM <dttm>, TRT01SDTM <dttm>, TRT01EDTM <dttm>, TRT02SDTM <dttm>,
#   TRT02EDTM <dttm>, AP01SDTM <dttm>, AP01EDTM <dttm>, AP02SDTM <dttm>, ...

[[2]]
# A tibble: 3 x 92
  STUDYID USUBJID     SUBJID SITEID   AGE AGEU  SEX   RACE  ETHNIC COUNTRY DTHFL
  <chr>   <chr>       <chr>  <chr>  <int> <fct> <fct> <fct> <fct>  <fct>   <fct>
1 AB12345 AB12345-BR~ id-134 BRA-1     47 YEARS M     WHITE NOT H~ BRA     Y
2 AB12345 AB12345-BR~ id-134 BRA-1     47 YEARS M     WHITE NOT H~ BRA     Y
3 AB12345 AB12345-BR~ id-134 BRA-1     47 YEARS M     WHITE NOT H~ BRA     Y
# ... with 81 more variables: INVID <chr>, INVNAM <chr>, ARM <fct>,
#   ARMCD <fct>, ACTARM <fct>, ACTARMCD <fct>, TRT01P <fct>, TRT01A <fct>,
#   TRT02P <fct>, TRT02A <fct>, REGION1 <fct>, STRATA1 <fct>, STRATA2 <fct>,
#   BMRKR1 <dbl>, BMRKR2 <fct>, ITTFL <fct>, SAFFL <fct>, BMEASIFL <fct>,
#   BEP01FL <fct>, AEWITHFL <fct>, RANDDT <date>, TRTSDTM <dttm>,
#   TRTEDTM <dttm>, TRT01SDTM <dttm>, TRT01EDTM <dttm>, TRT02SDTM <dttm>,
#   TRT02EDTM <dttm>, AP01SDTM <dttm>, AP01EDTM <dttm>, AP02SDTM <dttm>, ...

[[3]]
# A tibble: 3 x 68
  STUDYID USUBJID     SUBJID SITEID   AGE AGEU  SEX   RACE  ETHNIC COUNTRY DTHFL
  <chr>   <chr>       <chr>  <chr>  <int> <fct> <fct> <fct> <fct>  <fct>   <fct>
1 AB12345 AB12345-BR~ id-105 BRA-1     38 YEARS M     BLAC~ HISPA~ BRA     N
2 AB12345 AB12345-BR~ id-105 BRA-1     38 YEARS M     BLAC~ HISPA~ BRA     N
3 AB12345 AB12345-BR~ id-105 BRA-1     38 YEARS M     BLAC~ HISPA~ BRA     N
# ... with 57 more variables: INVID <chr>, INVNAM <chr>, ARM <fct>,
#   ARMCD <fct>, ACTARM <fct>, ACTARMCD <fct>, TRT01P <fct>, TRT01A <fct>,
#   TRT02P <fct>, TRT02A <fct>, REGION1 <fct>, STRATA1 <fct>, STRATA2 <fct>,
#   BMRKR1 <dbl>, BMRKR2 <fct>, ITTFL <fct>, SAFFL <fct>, BMEASIFL <fct>,
#   BEP01FL <fct>, AEWITHFL <fct>, RANDDT <date>, TRTSDTM <dttm>,
#   TRTEDTM <dttm>, TRT01SDTM <dttm>, TRT01EDTM <dttm>, TRT02SDTM <dttm>,
#   TRT02EDTM <dttm>, AP01SDTM <dttm>, AP01EDTM <dttm>, AP02SDTM <dttm>, ...
```

2.4 `init()`

The most crucial function of the `teal` package(s) is the `teal::init()` function, which is structured as follows:

```
init(
  data,
  modules,
  title = NULL,
  filter = list(),
  header = tags$p("Add Title Here"),
  footer = tags$p("Add Footer Here"),
)
```

The `init()` function sets up a shiny app that consists of `teal` modules. Let's go through the arguments:

In `header` and `footer` we can specify html content to be displayed at the top (respectively at the bottom) of the application.

`data`

In the `data` argument we define one or more data-frames for the application. If more than one data-frames are specified, they should be combined as a list, e.g. `data = list(ADSL, ADTR)`. For ADaM data-sets, the package `teal.data` provides helper functions with which the `data` argument of `init()` can be specified, — the `cdisc_data()` and the `cdisc_dataset()` functions. They allow the teal app to know the merge key variables of the data-sets. The `cdisc_data`function returns an S6 object.

> **Note**
>
> `keys` are pre-specified variables in ADaM data to merge data-sets. Each type of data-set requires specific keys to be specified. The package `teal.data` automatically chooses the correct key variables based on the `dataname` for the following data-sets: ADSL, ADAE, ADEG, ADTTE, ADAETTE, ADCM, ADEX, ADLB, ADMH, ADQS, ADRS, ADSAFTTE, ADVS, ADDV, ADSUB, ADHY.
> If you want to work with other ADaM data-sets you must specify the keys manually. ADTR is a Tumor Results Analysis data-set where there is one record (row) per subject. As of now, this is not part of the default cdisc keys in the `teal.data` package, so we would have to specify them manually:

```
ADTR <- random.cdisc.data::cadtr


cdisc_data(
  cdisc_dataset("ADSL", ADSL),
  cdisc_dataset("ADTR", ADTR, keys = c("STUDYID", "USUBJID", "PARAMCD", "AVISIT"))
)
```

Note that all keys have to be specified.

Often, pre-processing of data is still necessary. Generally, it is recommended to collect all data pre-processing steps in a separate R-script, which can then directly be applied to the `cdisc_data()` output. One such pre-processing steps could be the creation of a new variable in the ADSL data-set which splits patients into two age groups where patients younger than 40 are `AGEGR = 0` and patients older than 40 are `AGEGR = 1`.

```
# you would write this in a separate file directly, without the writeLines command.
writeLines(
  text = "
ADSL <- ADSL %>%
  dplyr::mutate(AGEGR = if_else(AGE < 40, 0, 1, NA_real_))",
  con = "R/snippets/ADSL_preproc.R"
)
```

The resulting R file looks something like this:

```
ADSL <- ADSL %>%
  dplyr::mutate(AGEGR = if_else(AGE < 40, 0, 1, NA_real_))
```

```
ADSL_processed <- ADSL %>%
  cdisc_dataset("ADSL", .) %>%
  mutate_dataset(script = "R/snippets/ADSL_preproc.R")
# check if the new variable exists:
ADSL_processed$get_raw_data()$AGEGR
```

```
  [1] 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 [38] 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0
 [75] 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1
[112] 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0
[149] 0 1 1 0 1 0 0 1 0 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
[186] 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0
```

```
[223] 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0
[260] 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0
[297] 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0
[334] 1 0 0 1 1 1 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1
[371] 0 0 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1
```

In practice, we would not necessarily want to separately save the pre-processed `ADSL` in our environment. Instead, we would apply the pre-processing within the `data` argument of the `init()` function, e.g., like this:

```
app <- teal::init(
  data = cdisc_data(
    cdisc_dataset("ADSL", ADSL) %>%
      mutate_dataset(script = "R/snippets/ADSL_preproc.R"),
    cdisc_dataset("ADAE", ADAE),
    cdisc_dataset("ADTTE", ADTTE)
    # <<additional data-set code>>
  )
  # , <<additional init arguments code>>
)
```

`filter`

This text is still under construction.

```
filter <- list(ADSL = structure(list(AGE = list()), filterable = c("AGE", "SEX", "RACE")))


module(
  "ADSL AGE histogram",
  server = function(input, output, session, datasets) {
    output$hist <- renderPlot(
      hist(datasets$get_data("ADSL", filtered = TRUE)$AGE)
    )
  },
  ui = function(id, ...) {
    ns <- NS(id)
    plotOutput(ns("hist"))
  },
  filters = "ADSL"
)
```

```
+ ADSL AGE histogram
```

id

Not sure if this should be included.

modules

The `modules` argument takes pre-defined or custom `teal` modules as input. A module or various modules need to be wrapped within the `modules()` function to work.

In this chapter we look at four useful modules from the `teal.modules.general` and the `teal.modules.clinical` packages to look at the data.

Module	Package	Purpose
`tm_front_page()`	`teal.modules.general`	Creates a front page which should include project metadata and an application user guide
`tm_variable_browser()`	`teal.modules.general`	Let's you look at distributions of variables.
`tm_data_table()`	`teal.modules.general`	Displays raw data tables
`tm_t_summary()`	`teal.modules.clinical`	Demographics table

2.5 `tm_front_page()`

We can use this module to create an informative front page of our app. This will be the landing page for anyone clicking on the link to our application.

The front page allows you to set the context of the app and inform users how to use the app.

It can make sense to define inputs in the parameters section, especially for information that is subject to change or might get re-used. This improves code re-usability.

Package table

We can get all loaded packages with the command '.packages()':

```
print(.packages())
```

```
 [1] "lubridate"             "timechange"            "forcats"
 [4] "stringr"               "dplyr"                 "purrr"
 [7] "readr"                 "tidyr"                 "tibble"
[10] "tidyverse"             "sparkline"             "teal.modules.clinical"
[13] "tern"                  "rtables"               "formatters"
[16] "teal.modules.general"  "teal"                  "teal.transform"
```

```
[19] "magrittr"              "shinyTree"             "ggmosaic"
[22] "ggplot2"               "teal.data"             "shiny"
[25] "random.cdisc.data"     "stats"                 "graphics"
[28] "grDevices"             "datasets"              "utils"
[31] "methods"               "base"
```

Further, we can get the version number of each package via ‘ packageVersion()‘

```
packageVersion("tidyverse")
```

```
[1] '1.3.2.9000'
```

We can then create a data-frame listing all packages and versions:

```
(package_table <- tibble(
  package = .packages(),
  version = sapply(
    X = .packages(),
    FUN = function(x) as.character(packageVersion(x))
  )
))
```

```
# A tibble: 32 x 2
   package    version
   <chr>      <chr>
 1 lubridate  1.9.0
 2 timechange 0.2.0
 3 forcats    0.5.2
 4 stringr    1.5.0
 5 dplyr      1.0.10
 6 purrr      1.0.1
 7 readr      2.1.3
 8 tidyr      1.2.1
 9 tibble     3.1.8
10 tidyverse  1.3.2.9000
# ... with 22 more rows
```

You can supply additional html tags using the `additional_tags` argument. If you just want to paste some extra text or your input already includes html tags, use `HTML()`

```
# just some extra text
HTML("here is some extra text")
```

```
# a more complex html input
HTML('<div class="header" checked>
  <p>Ready to take the Shiny tutorial? If so</p>
  <a href="shiny.rstudio.com/tutorial">Click Here!</a>
</div>')
```

You can also use the `tagList()` to create html code on the spot:

```
tagList(tags$div(
  class = "header", checked = NA,
  tags$p("Ready to take the `teal` tutorial? If so"),
  tags$a(href = "https://stefanthoma.github.io/teal_intro/", "Click Here!")
))
```

You can also specify metadata of your data-sets in the data argument, e.g.

```
data <- cdisc_data(
  cdisc_dataset("ADSL", ADSL,
    code = "ADSL <- synthetic_cdisc_data(\"latest\")$adsl",
    metadata = list("Author" = "NEST team", "data_source" = "synthetic data")
  ),
  check = TRUE
)
```

```
# Parameters
molecule <- "Gargamelumab"
author <- "Stefan Thoma"
studyid <- "1509210419"

# Tables for the front page:
# used packages:
package_table <- tibble(
  package = .packages(),
  version = sapply(
    X = .packages(),
    FUN = function(x) as.character(packageVersion(x))
  )
)
```

```r
# any other tables you want to include.
color_table <- tibble(
  color = c("green", "red"),
  meaning = c("ready", "not ready")
)



tm_front_page(
  label = "Front page", # what the navigation item should be called
  header_text = c(
    "Molecule" = molecule,
    "Study ID" = studyid,
    "App Author" = author,
  ),
  tables = list(
    package_table,
    color_table
  ),
  additional_tags = tagList(tags$div(
    class = "header", checked = NA,
    tags$p("Ready to take the `teal` tutorial? If so"),
    tags$a(href = "https://stefanthoma.github.io/teal_intro/", "Click Here!")
  )),
  footnotes = c("X" = "is the first footnote", "Y is the second footnote"),
  show_metadata = TRUE # lets app users access the specified metadata.
)
```

2.6 `tm_data_table()` #sec-data_table

This module allows users to browse the data-sets . Users can chose one of the data-sets and select various variables of interest and scroll through the rows.

As always, you can define the tab title using the `label` argument. The `datasets_selected` arguments lets you specify the data-sets that can be selected – and their order of appearance. By default all data-sets are select-able.

You can specify which variables of a data-set are initially shown using the `variables_selected` argument, this argument defaults to the first six columns.

Formatting and behaviour of the data-table can be adjusted using the arguments `dt_args` and `dt_options`. Please consult the manual for more details.

`pre_output` and `post_output` let you add custom html before and after the inputs on the encoding panel. [[[[SEE PICTURE]]]]

```
library(teal.modules.general)
tm_data_table(
  label = "Data Table",
  variables_selected = list(
    ADSL = c("STUDYID", "USUBJID", "SUBJID", "SITEID", "AGE", "SEX"),
    ADTTE = c("USUBJID", "AGE", "SEX", "EOTSTT")
  ),
)
```

2.7 `tm_variable_browser`

The `tm_variable_browser` lets app users look at summaries and histograms of variables in the data-set. It also allows for filtering based on the inputs on the right hand side. This module does not require any additional inputs to its function to be useful. However, its arguments do allow for some level of customization.

The arguments `dataset_selected`, `label`, `pre_output`, and `post_output` are described in section **?@sec-data_table**.

The `ggplot2_args` argument allows for the customisation of the plots displayed in the middle section. `teal` expects a specific format of the input, namely `teal.widgets::ggplot2_args()`-output - which itself allows two kinds of inputs:

- `labs` (named list) where all fields have to match ggplot2::labs() arguments.
- theme (named list) where all fields have to match ggplot2::theme() arguments.

In the example below I add subtitles to all plots using the `labs` argument, and I remove plot grid- and borderlines for a more minimalistic look.

[[[Think about names for the different sections of the app]]]

```
tm_variable_browser(
  label = "Variable Browser",
  #                    datasets_selected = "ADSL",
  pre_output = shiny::h3("Custom title"),
  post_output = shiny::h3("Custom text after output"),
  ggplot2_args = teal.widgets::ggplot2_args(
    labs = list(subtitle = "Plot generated by Variable Browser Module"),
    themes = list(
```

```
      panel.border = element_blank(),
      panel.grid.major = element_blank(),
      panel.grid.minor = element_blank()
    )
  )
)
```

2.8 `tm_t_summary`

Although this module is from the `teal.modules.clinical` package it offers a useful addition to the standard data display repertoire. The module allows users to summarise variables across different grouping variables. This is the first module in this tutorial that is a bit more complex to set up.

We first need to specify which data-set we want to summarize using the `dataname` argument. We also have to specify which variables we want summarized (using the argument `summarize_vars`), and by which grouping variables we want to summarize them (using the argument `arm_var`). To specify these two variables we rely on the `choices_selected()` function.

For `arm_var`, we can specify the potential grouping variables using the `choices` argument in the `choices_selected()` function using `variable_choices()` — yet another helper function — as specified below. In the `selected` argument we can then specify the initial grouping variable. If two variable-names (as a vector) are supplied to the `selected` argument, the groups are then nested.

`summarize_vars` also needs to be specified using `choices_selected()` where we specify all variables which can be selected to be summarized by the user, and the initially selected variables.

```
# parameters
ADSL
demographic_variables <- c("SEX", "AGE", "RACE")
arm_vars <- c("ARM", "REGION1")

cs_arm_var <- choices_selected(
  choices = variable_choices(ADSL, subset = arm_vars),
  selected = "ARM"
)
```

```
# module
tm_t_summary(
  label = "Demographic Table",
  dataname = "ADSL",
  arm_var = cs_arm_var,
  summarize_vars = choices_selected(
    choices = variable_choices(ADSL, demographic_variables),
    selected = c("SEX", "AGE", "RACE")
  )
)
```

Combining everything above, our `init()` function looks like this:

```
app <- teal::init(
  data = cdisc_data(
    cdisc_dataset("ADSL", ADSL) %>%
      mutate_dataset(script = "snippets/ADSL_preproc.R"),
    cdisc_dataset("ADAE", ADAE),
    cdisc_dataset("ADTTE", ADTTE)
  ),
  modules = modules(
    tm_front_page(
      label = "Study Information",
      header_text =
        c("Info about data source" = "Random data from the `random.cdisc.data` package are
      tables =
        list(
          "packages used" = package_table,
          "color table" = color_table
        ),
      additional_tags = tagList(tags$div(
        class = "header", checked = NA,
        tags$p("Ready to take the `teal` tutorial? If so"),
        tags$a(href = "https://stefanthoma.github.io/teal_intro/", "Click Here!")
      )),
      show_metadata = TRUE
    ),
    tm_data_table(
      label = "Data Table",
      variables_selected = list(
        ADSL = c("STUDYID", "USUBJID", "SUBJID", "SITEID", "AGE", "SEX"),
        ADTTE = c("USUBJID", "AGE", "SEX", "EOTSTT")
```

```
      ),
    ),
    tm_variable_browser(
      label = "Variable Browser",
      pre_output = shiny::h3("Custom title"),
      post_output = shiny::h3("Custom text after output"),
      ggplot2_args = teal.widgets::ggplot2_args(
        labs = list(subtitle = "Plot generated by Variable Browser Module"),
        theme = list(
          panel.border = element_blank(),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank()
        )
      )
    ),
    tm_t_summary(
      label = "Demographic Table",
      dataname = "ADSL",
      arm_var = cs_arm_var,
      summarize_vars = choices_selected(
        choices = variable_choices(ADSL, demographic_variables),
        selected = c("SEX", "AGE", "RACE")
      )
    )
  ),
  header = "My first application",
  footer = "footer"
)
```

2.9 Toy app

We have now gone through all steps to create your first teal app.

```
# data app
## ---- load packages data app ----
library(random.cdisc.data) # this package provides the data used
library(teal.data)
library(teal.modules.general)
library(teal.modules.clinical)
library(sparkline)
```

```r
library(teal)
library(tidyverse)


## ---- load data data app ----
ADSL <- random.cdisc.data::cadsl
ADAE <- random.cdisc.data::cadae
ADTTE <- random.cdisc.data::cadtte

## ---- parameter data app ----
# metadata
author <- "Stefan Thoma"
molecule <- "hexadromedar"

# parameters
demographic_variables <- c("SEX", "AGE", "RACE")
arm_vars <- c("ARM", "REGION1")

cs_arm_var <- choices_selected(
  choices = variable_choices(ADSL, subset = arm_vars),
  selected = "ARM"
)

package_table <- tibble(
  package = .packages(),
  version = sapply(
    X = .packages(),
    FUN = function(x) as.character(packageVersion(x))
  )
)
# any other tables you want to include.
color_table <- tibble(
  color = c("green", "red"),
  meaning = c("ready", "not ready")
)


## ---- init data app ----
app <- teal::init(
  data = cdisc_data(
    cdisc_dataset("ADSL", ADSL) %>%
```

```
    mutate_dataset(script = "snippets/ADSL_preproc.R"),
  cdisc_dataset("ADAE", ADAE),
  cdisc_dataset("ADTTE", ADTTE)
),
modules = modules(
  tm_front_page(
    label = "Study Information",
    header_text =
      c("Info about data source" = "Random data from the `random.cdisc.data` package are
    tables =
      list(
        "packages used" = package_table,
        "color table" = color_table
      ),
    additional_tags = tagList(tags$div(
      class = "header", checked = NA,
      tags$p("Ready to take the `teal` tutorial? If so"),
      tags$a(href = "https://stefanthoma.github.io/teal_intro/", "Click Here!")
    )),
    show_metadata = TRUE
  ),
  tm_data_table(
    label = "Data Table",
    variables_selected = list(
      ADSL = c("STUDYID", "USUBJID", "SUBJID", "SITEID", "AGE", "SEX"),
      ADTTE = c("USUBJID", "AGE", "SEX", "EOTSTT")
    ),
  ),
  tm_variable_browser(
    label = "Variable Browser",
    pre_output = shiny::h3("Custom title"),
    post_output = shiny::h3("Custom text after output"),
    ggplot2_args = teal.widgets::ggplot2_args(
      labs = list(subtitle = "Plot generated by Variable Browser Module"),
      theme = list(
        panel.border = element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()
      )
    )
  ),
```

```
    tm_t_summary(
      label = "Demographic Table",
      dataname = "ADSL",
      arm_var = cs_arm_var,
      summarize_vars = choices_selected(
        choices = variable_choices(ADSL, demographic_variables),
        selected = c("SEX", "AGE", "RACE")
      )
    )
  ),
  header = "My first application",
  footer = "footer"
)


## ---- run data app ----
shinyApp(app$ui, app$server)
```

The output should look something like this:

2.10 Production app

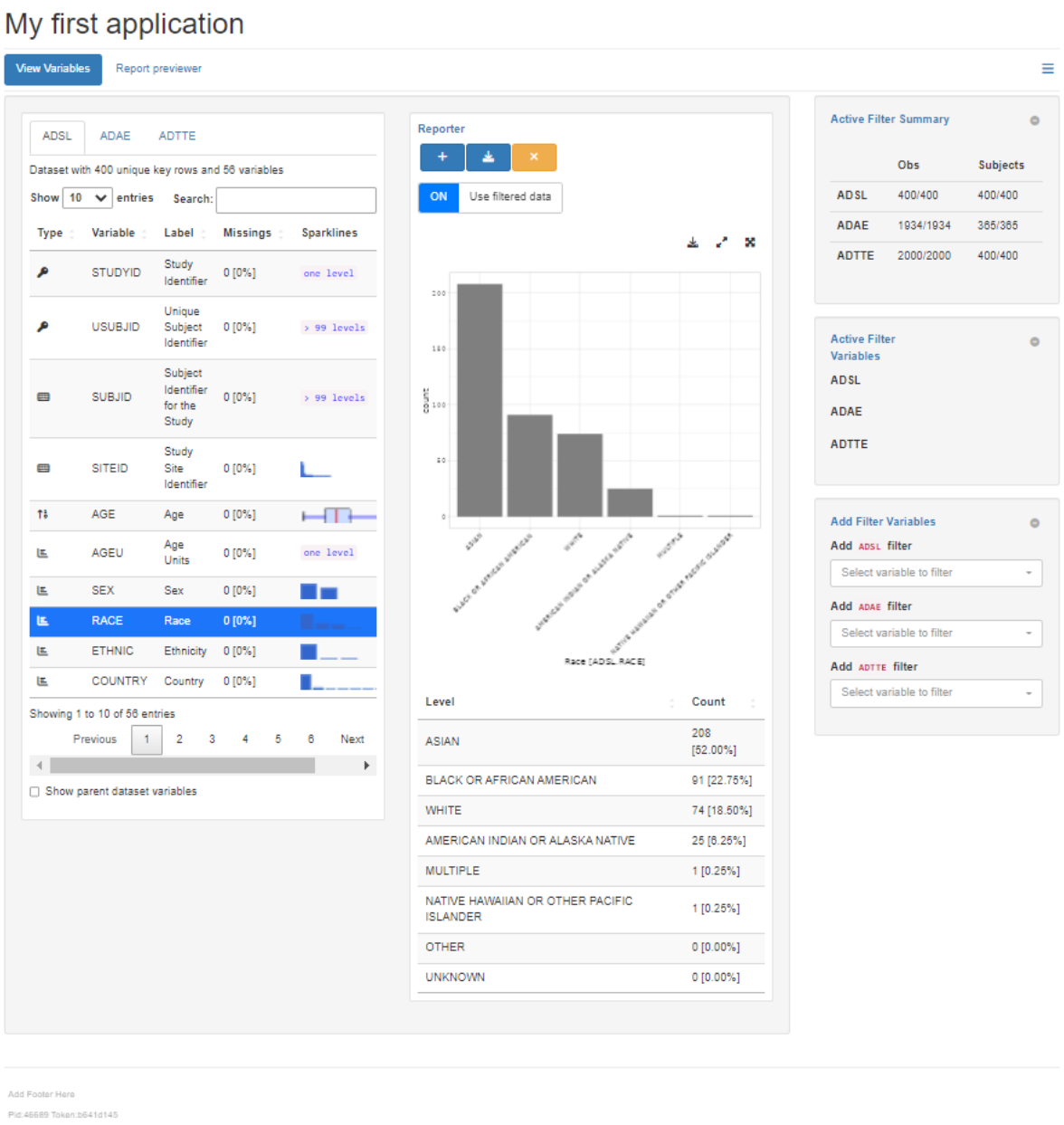


Figure 2.1: Data App preview

3 Efficacy app

This is a template file for new chapters. If you want to create a new chapter, simply copy this template, and adjust as needed.

At the end of this chapter you should be able to

- write your own chapter
- keep the chapters looking consistent
- especially from data.qmd onward

3.1 Setup

3.2 Modules

forest plot

kaplan meier

cox regressoin

time to event

maybe logistic regression

3.3 Toy app

3.4 Production app

Link to app:

4 Summary

In summary, this book has no content whatsoever.

```
1 + 1
```

```
[1] 2
```

References

Feedback

Loading...