Introduction to `teal`

Stefan Thoma

1/19/23

Table of contents

Preface

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 this chapter should give you a good understanding of how the more developed application works.

What will you 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

- link to other manuals for teal

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.

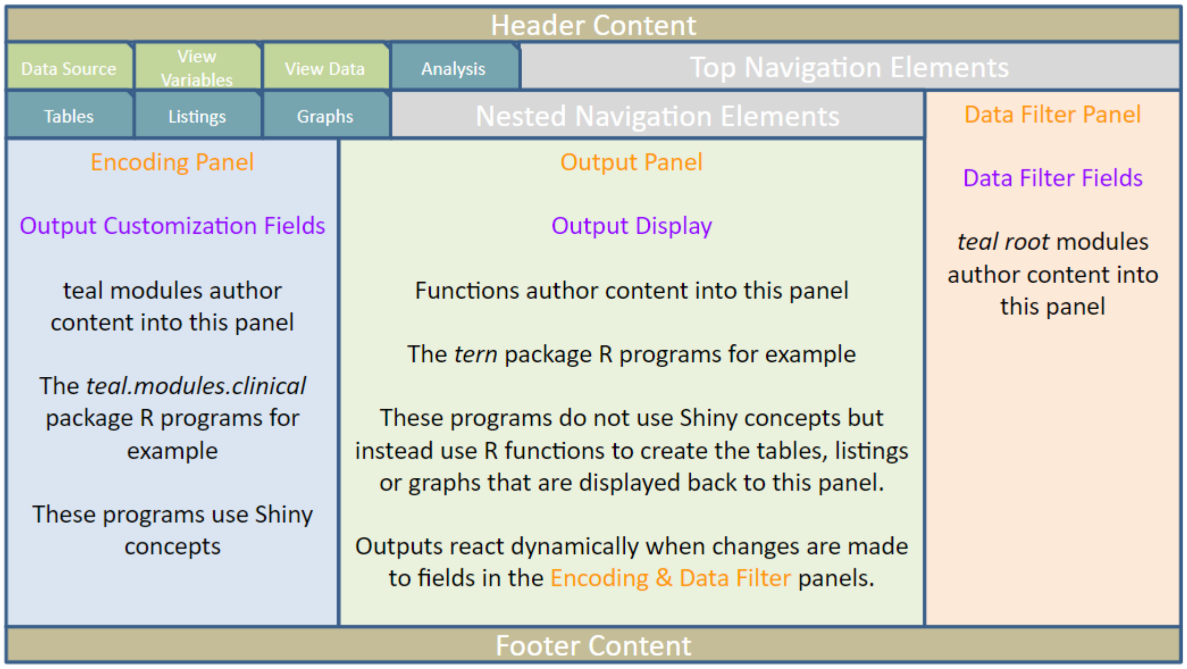
- The `Left Panel`, referred to as the encoding panel, includes elements used to adjust the output interactively.

- The `Center Panel`, referred to as the output panel, where the tables and visualizations are displayed.

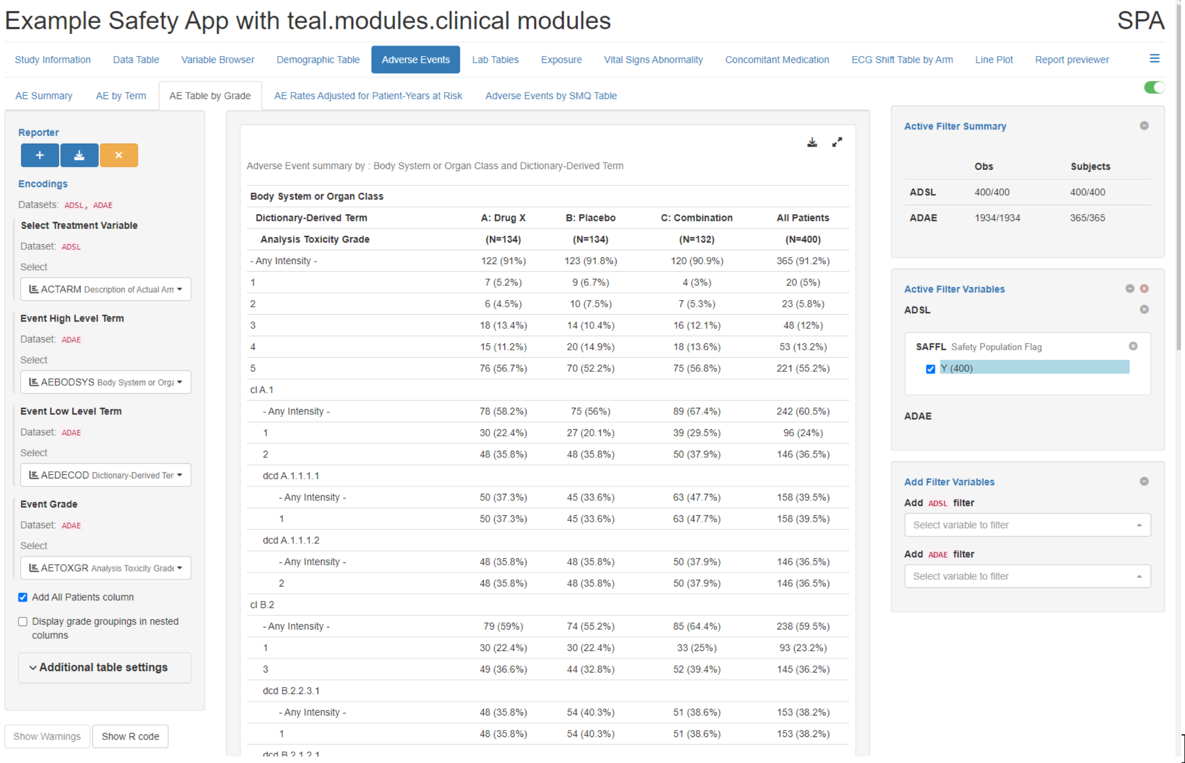
- The `Right Panel`, referred to as the data filter panel, includes elements used to subset the data interactively.

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.


Header Content
Data Source
View Variables
View Data
Analysis
Top Navigation Elements
Tables
Listings
Graphs
Nested Navigation Elements
Data Filter Panel
Data Filter Fields
teal root modules author content into this panel
Encoding Panel
Output Customization Fields
teal modules author content into this panel
The teal.modules.clinical package R programs for example
These programs use Shiny concepts
Output Panel
Output Display
Functions author content into this panel
The tern package R programs for example
These programs do not use Shiny concepts but instead use R functions to create the tables, listings or graphs that are displayed back to this panel.
Outputs react dynamically when changes are made to fields in the Encoding & Data Filter panels.
Footer Content

(a) diagram



Example Safety App with teal.modules.clinical modules

Datasets: ADSL, ADAE

Select Treatment Variable
Dataset: ADSL
Select: ACTARM Description of Actual Arm

Event High Level Term
Dataset: ADAE
Select: AEBODSYS Body System or Orga

Event Low Level Term
Dataset: ADAE
Select: AEDECOD Dictionary-Derived Ter

Event Grade
Dataset: ADAE
Select: AETOXGR Analysis Toxicity Grade

[✓] Add All Patients column
[ ] Display grade groupings in nested columns

Adverse Event summary by : Body System or Organ Class and Dictionary-Derived Term

Body System or Organ Class				
Dictionary-Derived Term	A: Drug X	B: Placebo	C: Combination	All Patients
Analysis Toxicity Grade	(N=134)	(N=134)	(N=132)	(N=400)
- Any Intensity -	122 (91%)	123 (91.8%)	120 (90.9%)	365 (91.2%)
1	7 (5.2%)	9 (6.7%)	4 (3%)	20 (5%)
2	6 (4.5%)	10 (7.5%)	7 (5.3%)	23 (5.8%)
3	18 (13.4%)	14 (10.4%)	16 (12.1%)	48 (12%)
4	15 (11.2%)	20 (14.9%)	18 (13.6%)	53 (13.2%)
5	76 (56.7%)	70 (52.2%)	75 (56.8%)	221 (55.2%)
cl A.1				
- Any Intensity -	78 (58.2%)	75 (56%)	89 (67.4%)	242 (60.5%)
1	30 (22.4%)	27 (20.1%)	39 (29.5%)	96 (24%)
2	48 (35.8%)	48 (35.8%)	50 (37.9%)	146 (36.5%)
dcd A.1.1.1.1				
- Any Intensity -	50 (37.3%)	45 (33.6%)	63 (47.7%)	158 (39.5%)
1	50 (37.3%)	45 (33.6%)	63 (47.7%)	158 (39.5%)
dcd A.1.1.1.2				
- Any Intensity -	48 (35.8%)	48 (35.8%)	50 (37.9%)	146 (36.5%)
2	48 (35.8%)	48 (35.8%)	50 (37.9%)	146 (36.5%)
cl B.2				
- Any Intensity -	79 (59%)	74 (55.2%)	85 (64.4%)	238 (59.5%)
1	30 (22.4%)	30 (22.4%)	33 (25%)	93 (23.2%)
3	49 (36.6%)	44 (32.8%)	52 (39.4%)	145 (36.2%)
dcd B.2.2.3.1				
- Any Intensity -	48 (35.8%)	54 (40.3%)	51 (38.6%)	153 (38.2%)
1	48 (35.8%)	54 (40.3%)	51 (38.6%)	153 (38.2%)
dcd B.2.1.2.1				

Active Filter Summary

	Obs	Subjects
ADSL	400/400	400/400
ADAE	1934/1934	365/365

Active Filter Variables

ADSL

SAFFL Safety Population Flag
[✓] Y (400)

ADAE

Add ADSL filter
Add ADAE filter

](genentech.shinya

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

- r output
- shopping cart

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

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

```r
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")
```

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

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

```r
ADTTE <- random.cdisc.data::cadtte
```

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

```r
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"),
  id = character(0)
)
```

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

`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.

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:

```r
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.

```r
# 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 = "ADSL_preproc.R"
)
```

The resulting R file looks something like this:

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

```r
ADSL_processed <- ADSL %>%
  cdisc_dataset("ADSL", .) %>%
  mutate_dataset(script = "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 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 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 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 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:

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

`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_data_table()`	`teal.modules.general`	Displays raw data tables
`tm_variable_browser()`	`teal.modules.general`	Let's you look at distributions of variables.
`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 suject 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] "forcats"               "stringr"               "dplyr"
 [4] "purrr"                 "readr"                 "tidyr"
 [7] "tibble"                "tidyverse"             "sparkline"
[10] "teal.modules.clinical" "tern"                  "rtables"
[13] "formatters"            "teal.modules.general"  "teal"
[16] "teal.transform"        "magrittr"              "shinyTree"
[19] "ggmosaic"              "ggplot2"               "scda"
[22] "teal.data"             "shiny"                 "random.cdisc.data"
[25] "stats"                 "graphics"              "grDevices"
[28] "utils"                 "datasets"              "methods"
[31] "base"
```

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

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

```
[1] '1.3.2'
```

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))))
```

2.6

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 Shiny tutorial? If so"),
                 tags$a(href = "https://docs.posit.co/resources/shiny/", "Click Here!")
))
```

Learn more about html tags on the shiny help page.

metadata has to be specified 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))))
# 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 Shiny tutorial? If so"),
                 tags$a(href = "https://docs.posit.co/resources/shiny/", "Click Here!")
)),
  footnotes = c("X" = "is the first footnote", "Y is the second footnote"),
  show_metadata = TRUE
)
```

2.7 `tm_data_table()`

```r
tm_data_table()
```

2.8 `tm_variable_browser`

```r
tm_variable_browser()
```

2.9 `tm_t_summary`

Demographics table

```r
tm_t_summary()
```

header

footer

id

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

```r
app <- teal::init(
  data = cdisc_data(
    cdisc_dataset("ADSL", ADSL) %>%
      mutate_dataset(script = "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 are used that have been created with the
      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 Shiny tutorial? If so")
                                         tags$a(href = "https://docs.posit.co/resources/sh
      )),
      show_metadata = TRUE
    ),
tm_data_table("Data Table"),
tm_variable_browser("Variable Browser"),
    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"
)
```

2.10 Toy app

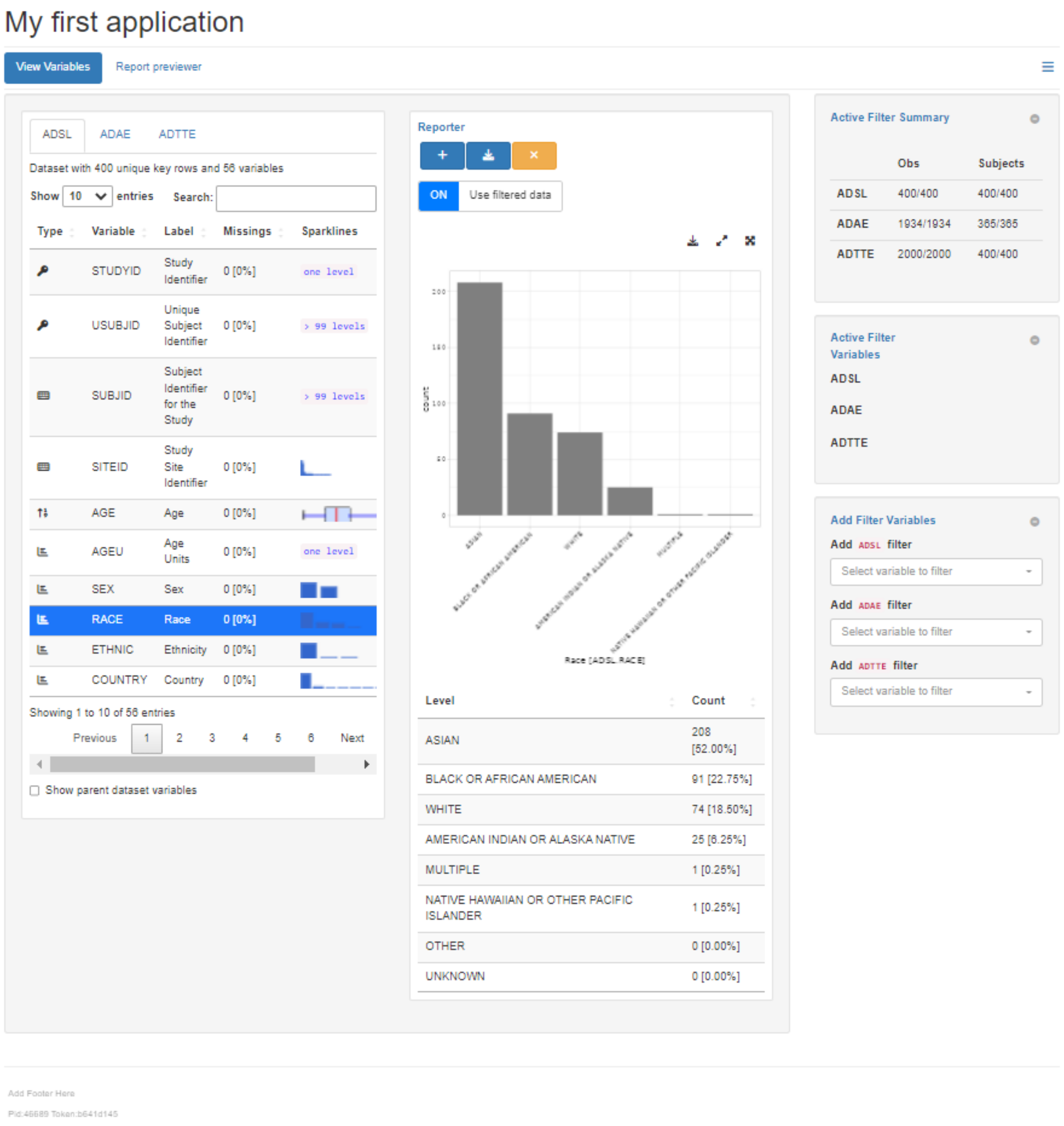
We have now gone through all steps to create your first teal app.

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

```r
## ---- 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 <- "ARM"

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 = "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 are used that have been created with the
      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 Shiny tutorial? If so")
                                          tags$a(href = "https://docs.posit.co/resources/sh
      )),
      show_metadata = TRUE
    ),
  tm_data_table("Data Table"),
  tm_variable_browser("Variable Browser"),
    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:

2.11 Production app



My first application

Dataset with 400 unique key rows and 56 variables

Show 10 entries Search:

Type	Variable	Label	Missings	Sparklines
	STUDYID	Study Identifier	0 [0%]	one level
	USUBJID	Unique Subject Identifier	0 [0%]	> 99 levels
	SUBJID	Subject Identifier for the Study	0 [0%]	> 99 levels
	SITEID	Study Site Identifier	0 [0%]	
	AGE	Age	0 [0%]	
	AGEU	Age Units	0 [0%]	one level
	SEX	Sex	0 [0%]	
	RACE	Race	0 [0%]	
	ETHNIC	Ethnicity	0 [0%]	
	COUNTRY	Country	0 [0%]	

Showing 1 to 10 of 56 entries

Previous 1 2 3 4 5 6 Next

Show parent dataset variables

Level	Count
ASIAN	208 [52.00%]
BLACK OR AFRICAN AMERICAN	91 [22.75%]
WHITE	74 [18.50%]
AMERICAN INDIAN OR ALASKA NATIVE	25 [6.25%]
MULTIPLE	1 [0.25%]
NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER	1 [0.25%]
OTHER	0 [0.00%]
UNKNOWN	0 [0.00%]

Active Filter Summary

	Obs	Subjects
ADSL	400/400	400/400
ADAE	1934/1934	365/365
ADTTE	2000/2000	400/400

Figure 2.1: Data App preview

3 efficacy

http://docs.roche.com/#/agile-R/2022_01_28/teal/sample_apps/sample-app-efficacy/

```r
sessionInfo()
```

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Big Sur ... 10.16

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
 [1] digest_0.6.30   lifecycle_1.0.3 jsonlite_1.8.3  magrittr_2.0.3
 [5] evaluate_0.18   rlang_1.0.6     stringi_1.7.8   cli_3.4.1
 [9] rstudioapi_0.14 vctrs_0.5.1     rmarkdown_2.18  tools_4.2.1
[13] stringr_1.5.0   glue_1.6.2      xfun_0.35       yaml_2.3.6
[17] fastmap_1.1.0   compiler_4.2.1  htmltools_0.5.3 knitr_1.41
```

4 Summary

In summary, this book has no content whatsoever.

```
1 + 1
```

```
[1] 2
```

References