RMarkdown Driven Development (RmdDD)

Emily Riederer

@emilyriederer

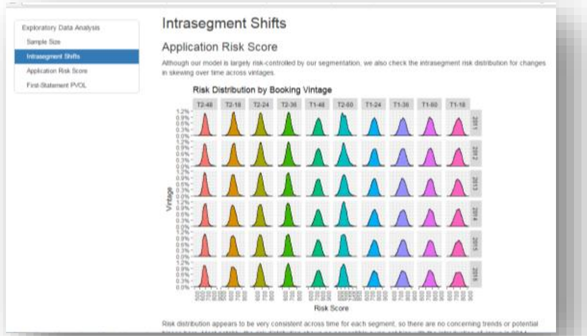
tinyurl.com/rmddd
tinyurl.com/rmddd-appendix

Code notebooks such as RMarkdown and Jupyter facilitate interactive data exploration and persistent document creation with literate programming

~~~
---
title: "My Analysis"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE)
```

```{r pkg-load}
library(dplyr)
library(tidyr)
library(survival)
library(ggfortify)
```

```{r data-load}
outcomes_df <- readr::read_csv('outcomes.csv')
```

## Introduction

In this analysis, we report the…

```{r all-the-good-code}
~~~

Dashboards

Analysis Reports



Intrasegment Shifts

Application Risk Score

Risk Distribution by Booking Vintage

Websites

projmgr

Try before you buy!

Installation

Slides

Test execution matches intent

Each analysis depends on a latent tool custom-fit to your domain-specific workflow

~~~
---
title: "My Analysis"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE)
~~~

```{r pkg-load}
library(dplyr)
library(tidyr)
library(survival)
library(ggfortify)
```

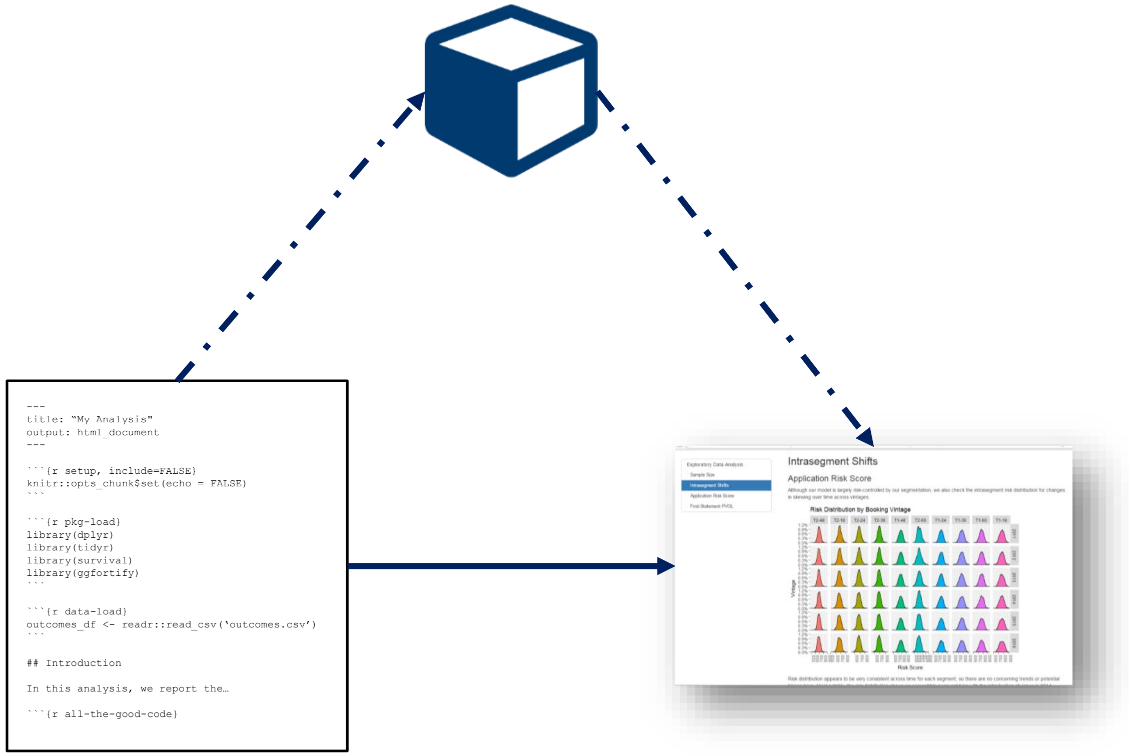
```{r data-load}
outcomes_df <- readr::read_csv('outcomes.csv')
```

Introduction

In this analysis, we report the…

```{r all-the-good-code}
```

library(myperfectpackage)



~~~
---
title: "My Analysis"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE)
~~~

```{r pkg-load}
library(dplyr)
library(tidyr)
library(survival)
library(ggfortify)
```

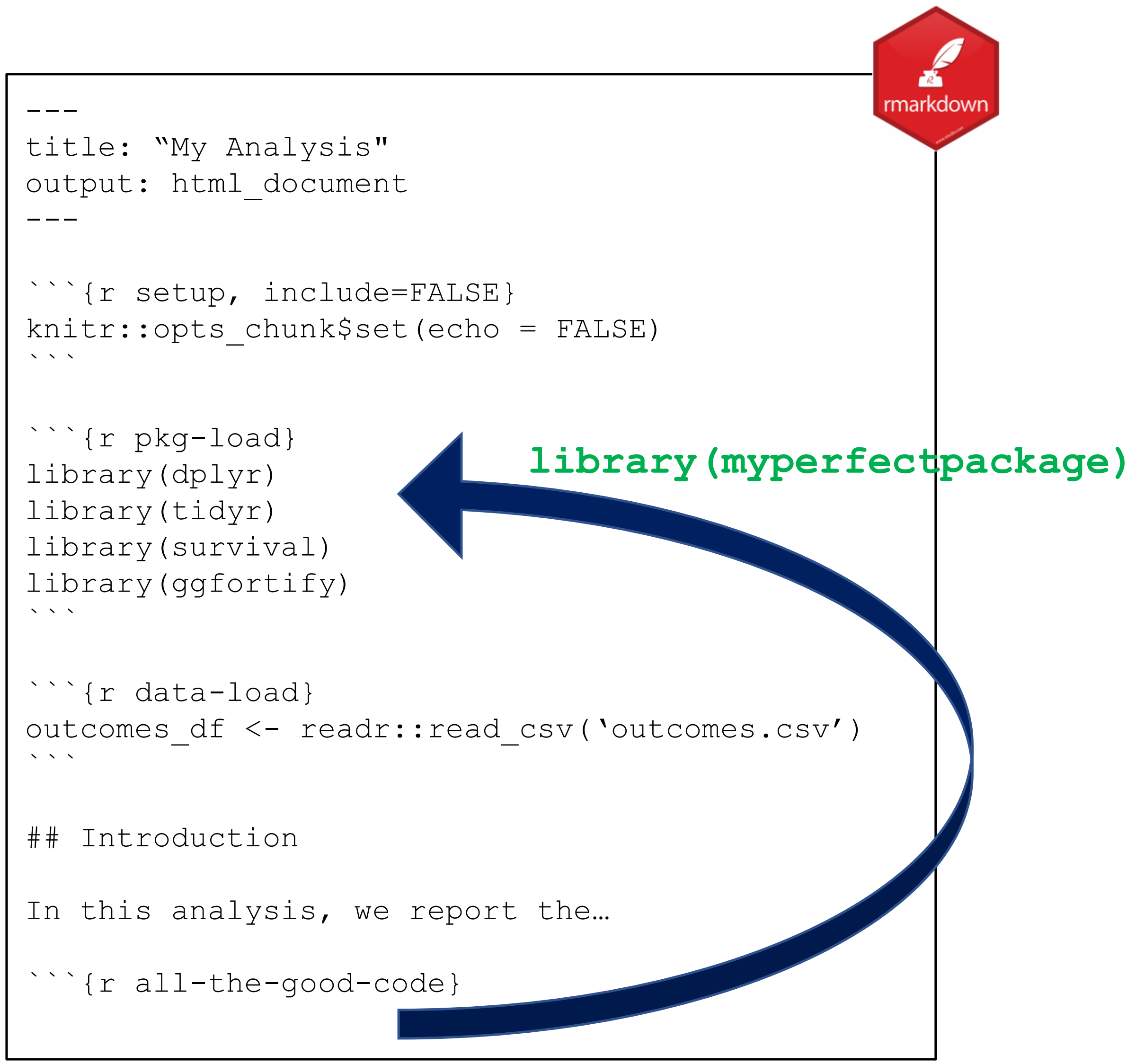
```{r data-load}
outcomes_df <- readr::read_csv('outcomes.csv')
```

Introduction

In this analysis, we report the…

```{r all-the-good-code}
```

Each analysis depends on a latent tool custom-fit to your domain-specific workflow



~~~
---
title: “My Analysis"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE)
~~~

```{r pkg-load}
library(dplyr)
library(tidyr)
library(survival)
library(ggfortify)
```

```{r data-load}
outcomes_df <- readr::read_csv(‘outcomes.csv’)
```

Introduction

In this analysis, we report the…

```{r all-the-good-code}
```

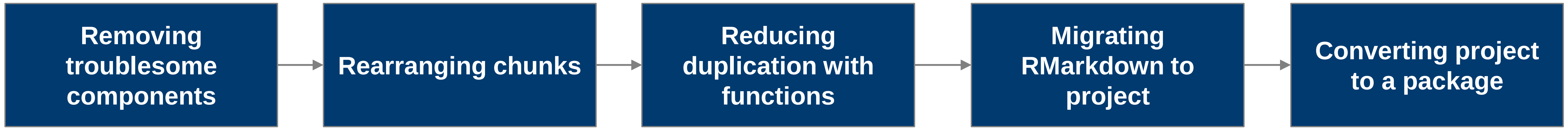
Design

- ✓ Understanding of requirements
- ✓ Sane workflow
- ✓ Complete & compelling example

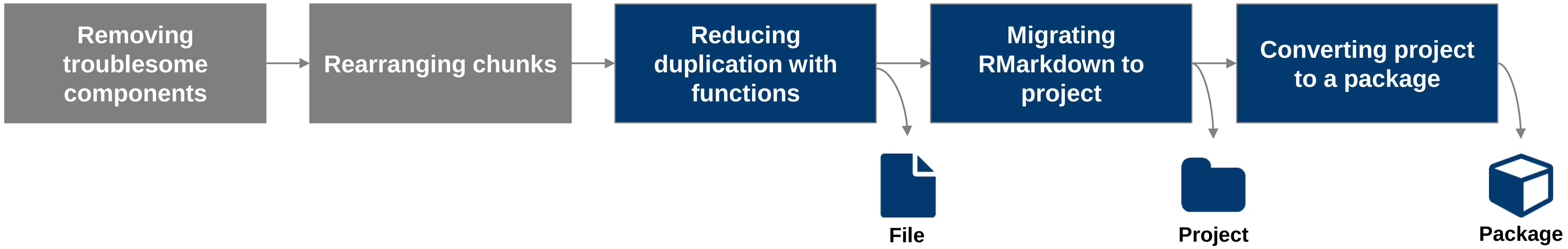
Development

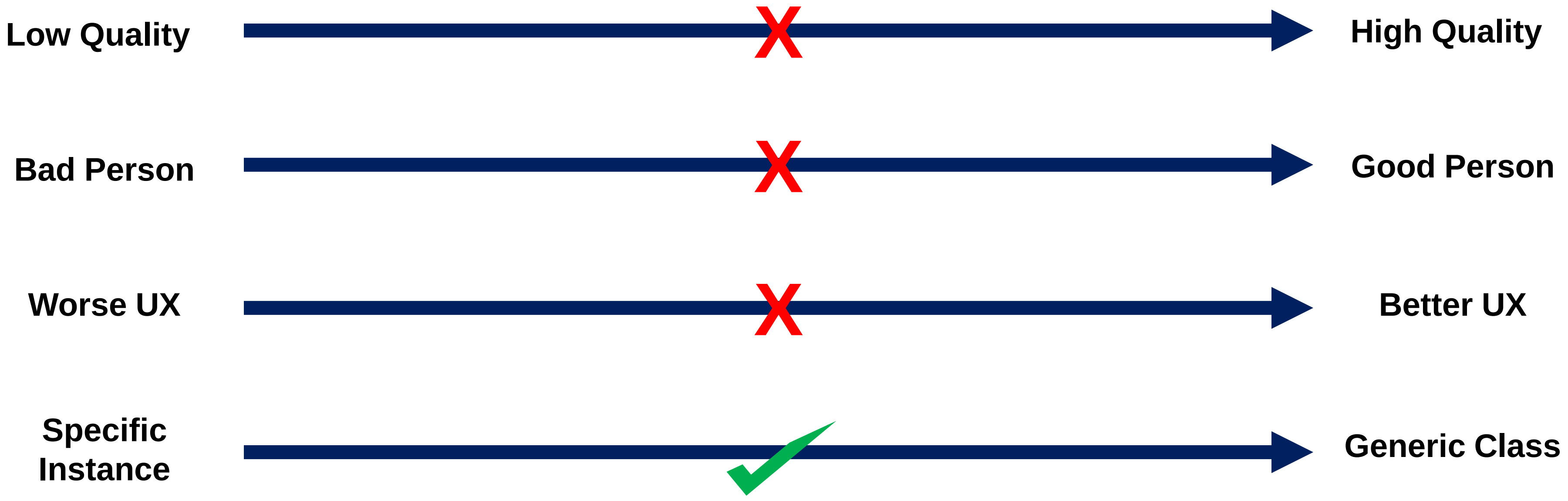
- ✓ Curated set of related libraries
- ✓ Working and “tested” code

RMarkdown Driven Development (RmdDD) has five main steps

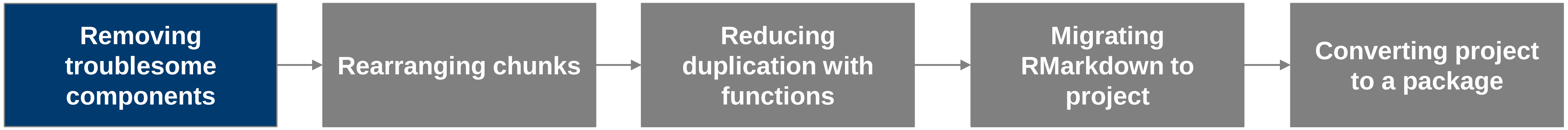

Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package

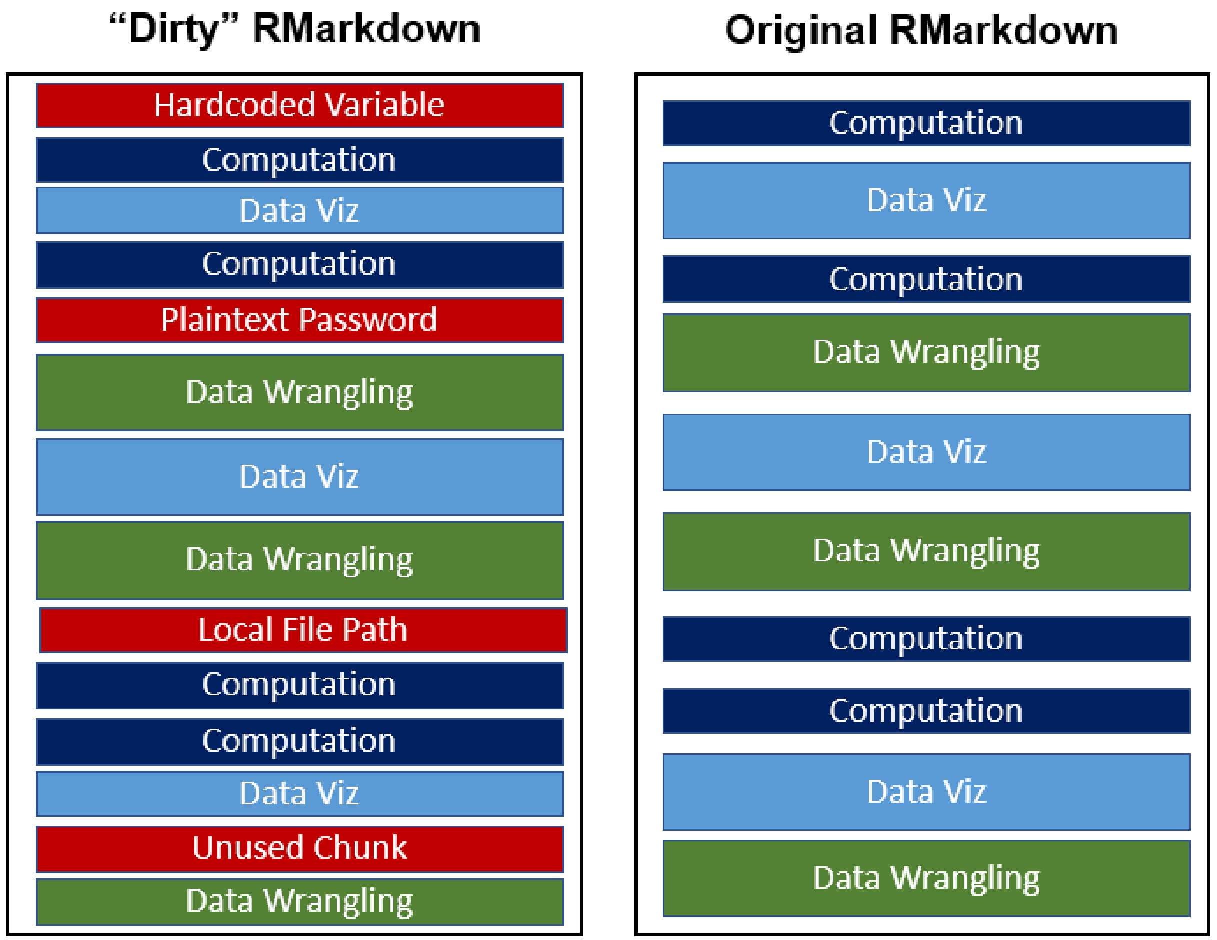
RmdDD has multiple endpoints, so you can take the right exit ramp for your destination


Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package
File
Project
Package

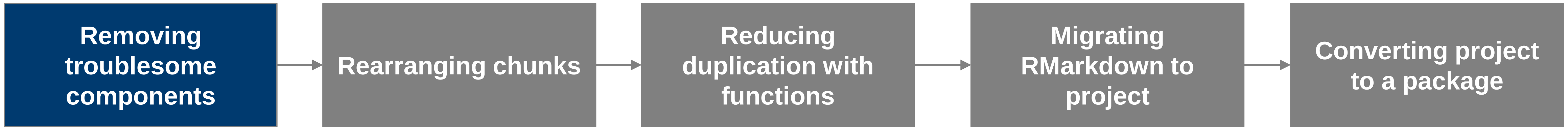


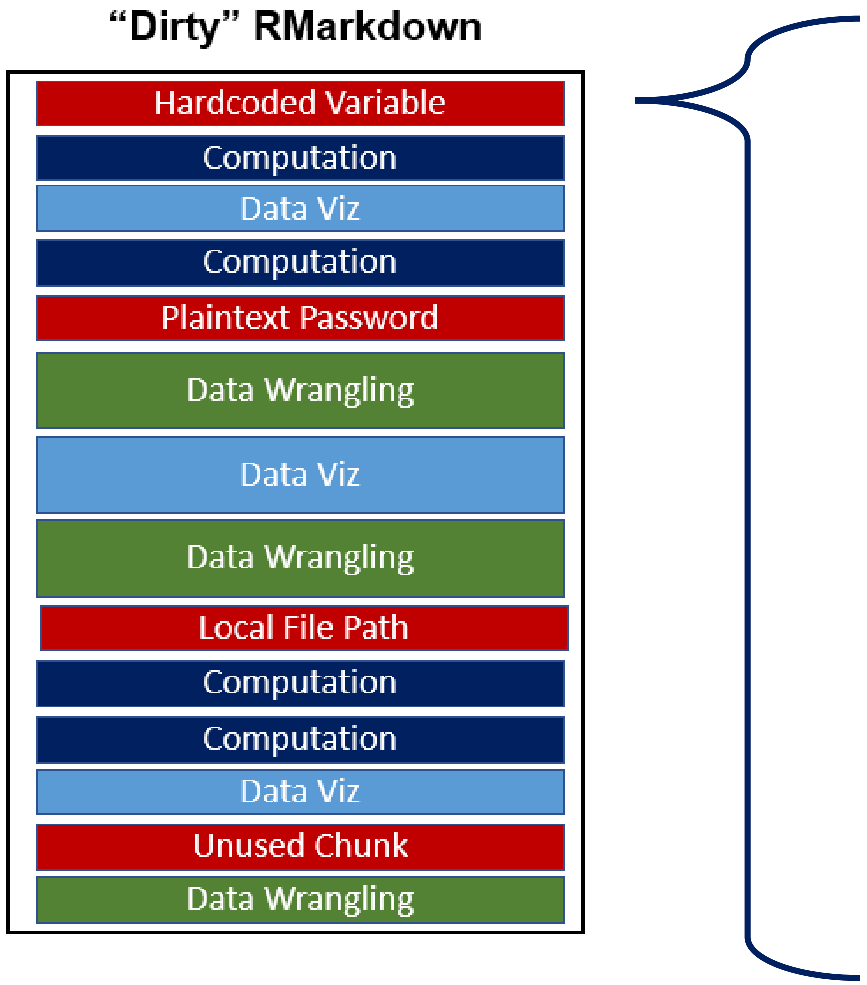
Eliminate clutter to make your own code more trustworthy for its initial use


Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package



Parameters can protect the integrity of your analysis and your credentials

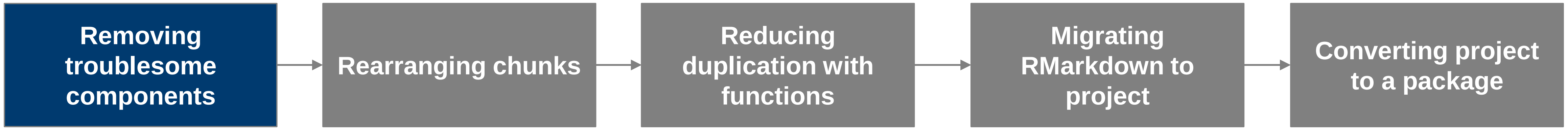

Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package

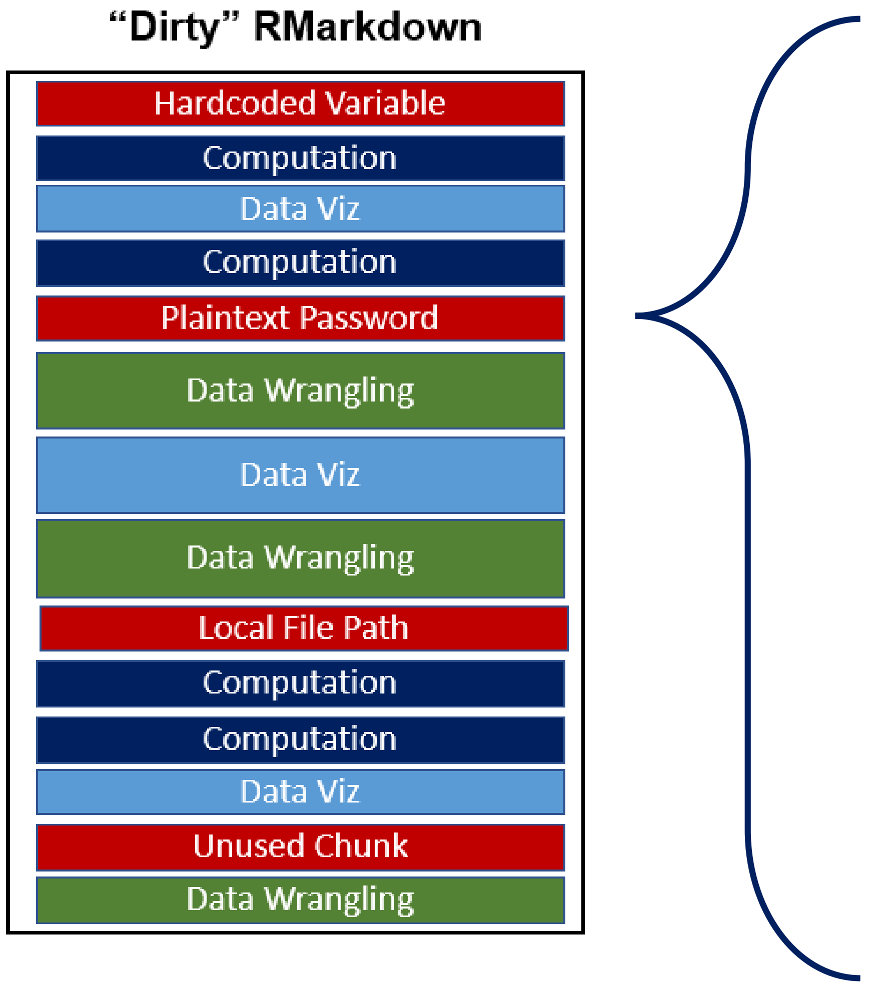


~~~
---
title: “My Analysis"
output: html_document
---

{{package loads, data loads, etc.}}

```{r}
data_lastyr <- data %>%
  filter(between(date, ‘2018-01-01’, ‘2018-12-31’))
```
~~~

~~~
---
title: “My Analysis"
output: html_document
params:
  start: ‘2018-01-01’
  end: ‘2018-12-31’
---

{{package loads, data loads, etc.}}

```{r}
data_lastyr <- data %>%
  filter(between(date, params$start, params$end))
```
~~~

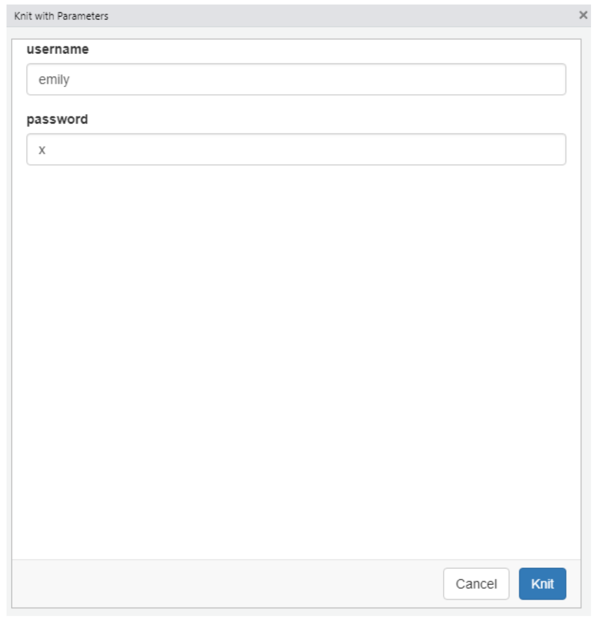
Parameters can protect the integrity of your analysis and your credentials


Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package



~~~
---
title: "My Analysis"
output: html_document
params:
  username: emily
  password: x
---

{{package loads, data loads, etc.}}

```{r}
con <-
  connect_to_database(
    username = params$username,
    password = params$password
  )
```
~~~

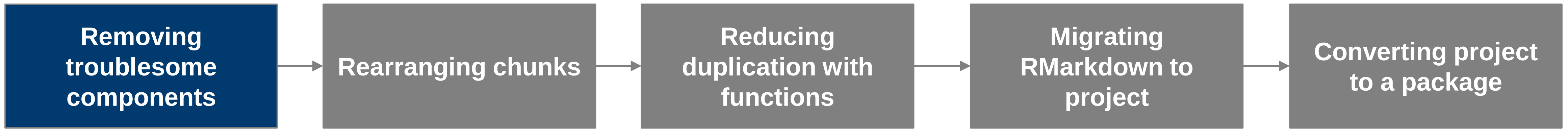
RStudio: Knit > Knit with Parameters...

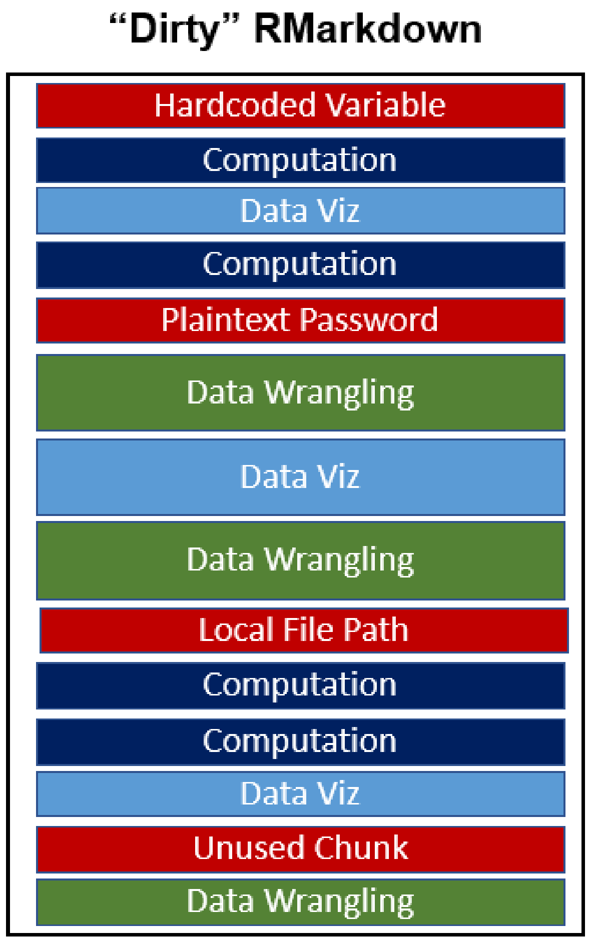


username: emily

password: x

Local file paths nearly guarantee that your project will not work on someone else's machine


Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package



Not resilient to any file structure change:

```
data <- readRDS('C:\Users\me\Desktop\my-project\data\my-data.rds')
```

Resilient to movement *of* working directory:

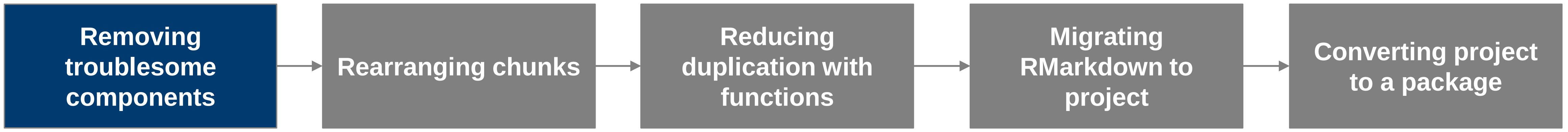
```
data <- readRDS('data\my-data.rds')
```

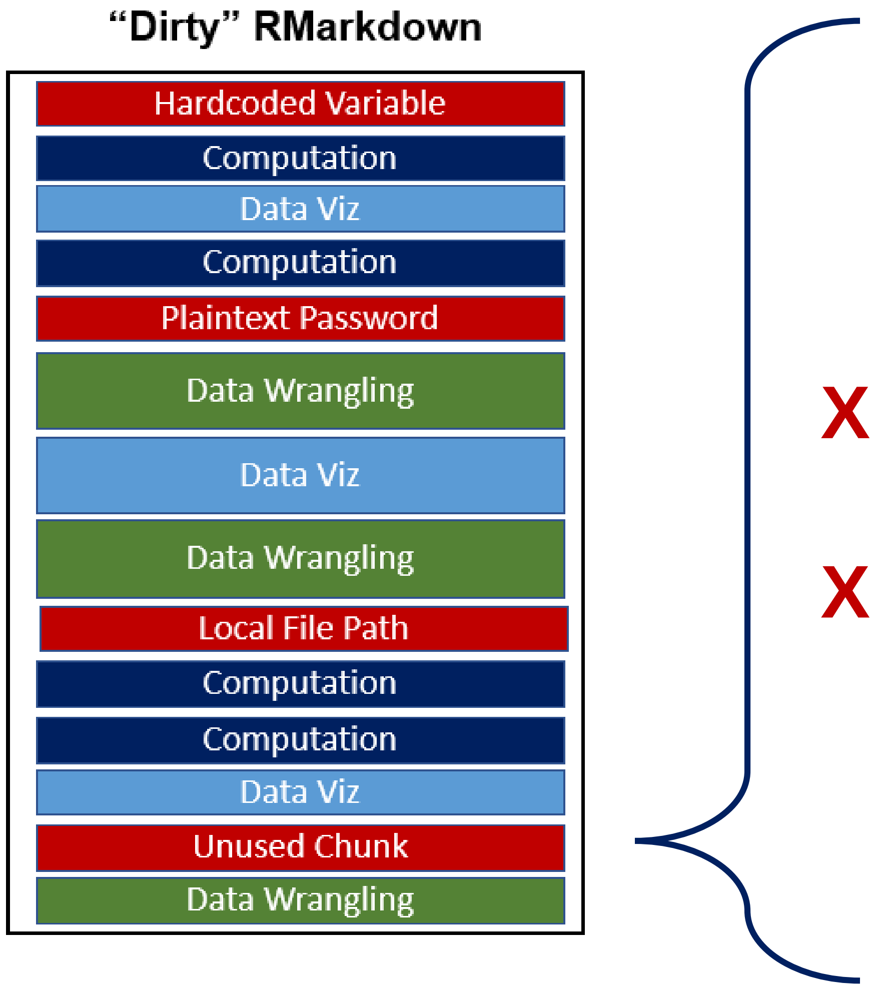
Resilient to movement of Rmd *within* working directory or across OS:

```
data <- readRDS(here::here('data', 'my-data.rds'))
```

here

Don’t let your script become a junk drawer

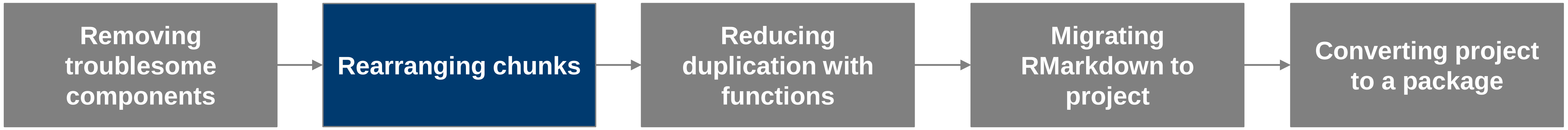

Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package

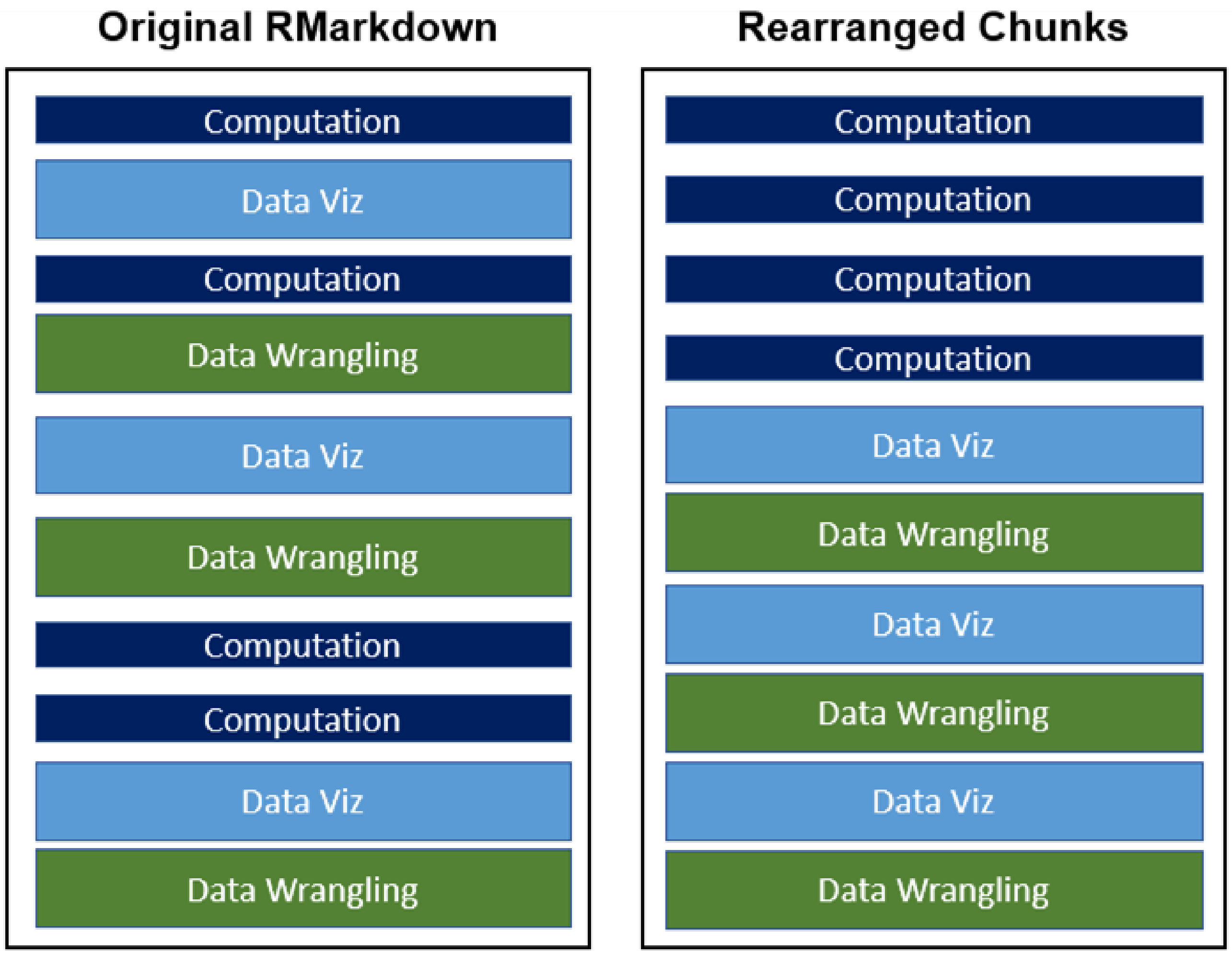


X Unused package loads

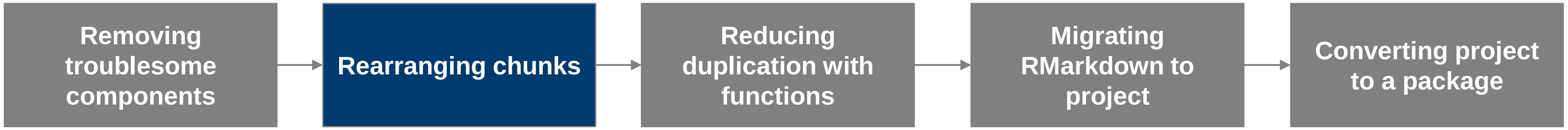
X Unsuccessful coding experiments

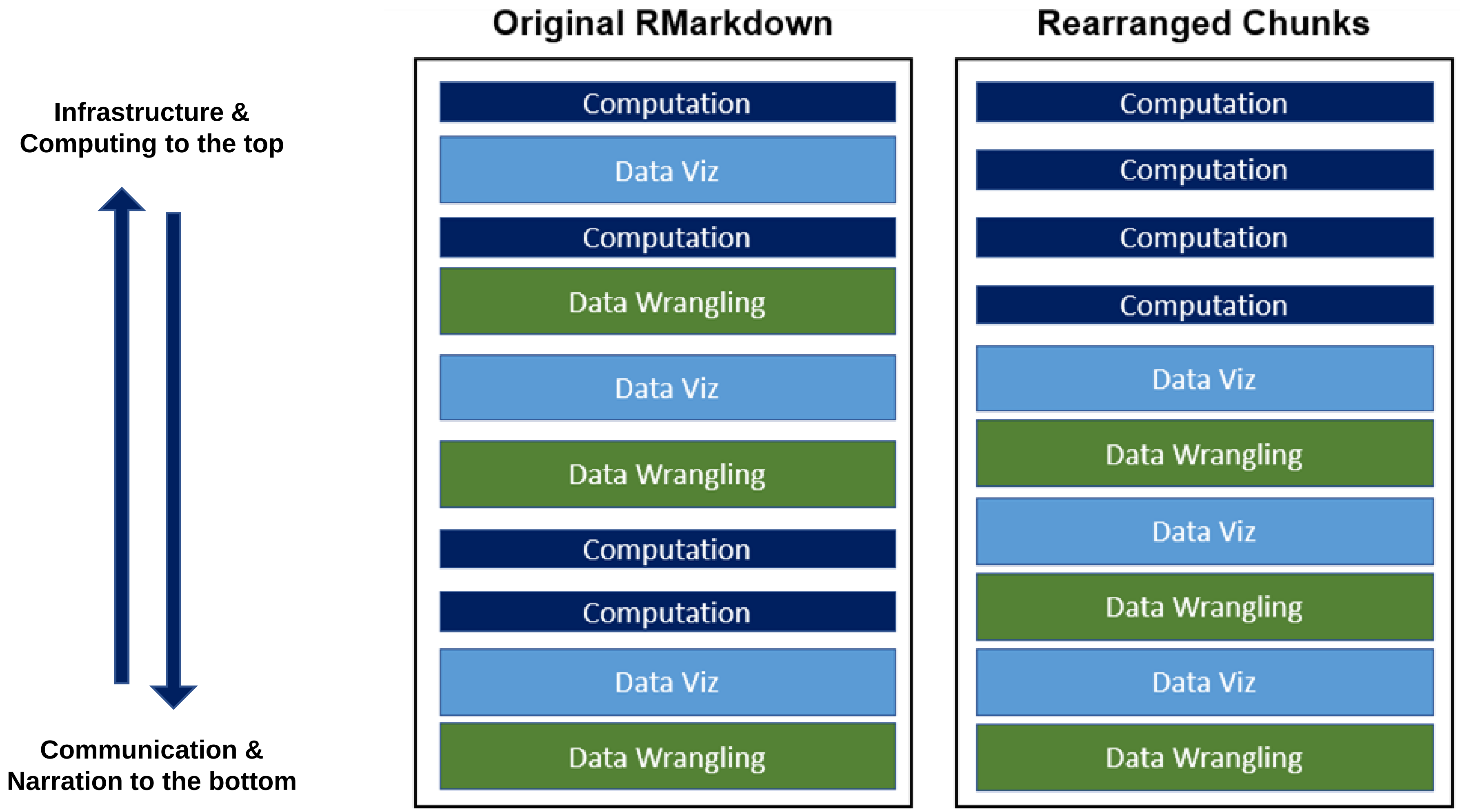
RMarkdown is (too) good at capturing our non-linear thought processes


Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package



Clustering quantitative and narrative components makes both easier to iterate on


Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package



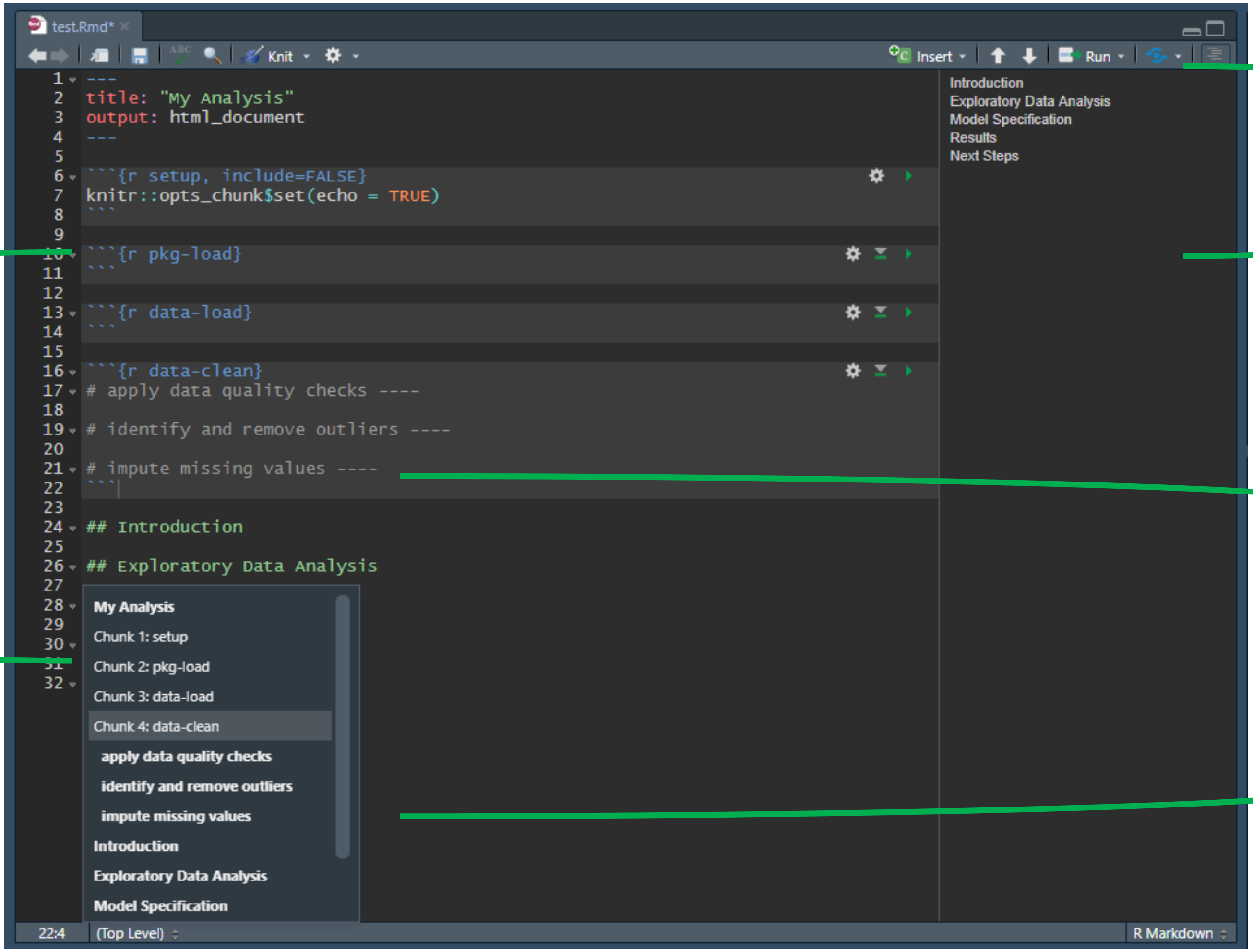
- Clear dependencies
- Frontloaded errors

- Increased likelihood of noticing repeated code

- Consolidated story
- Easier for non-coder to contribute

Enhance the navigability of your file in RStudio with chunk names and special comments

Removing troublesome components → **Rearranging chunks** → Reducing duplication with functions → Migrating RMarkdown to project → Converting project to a package



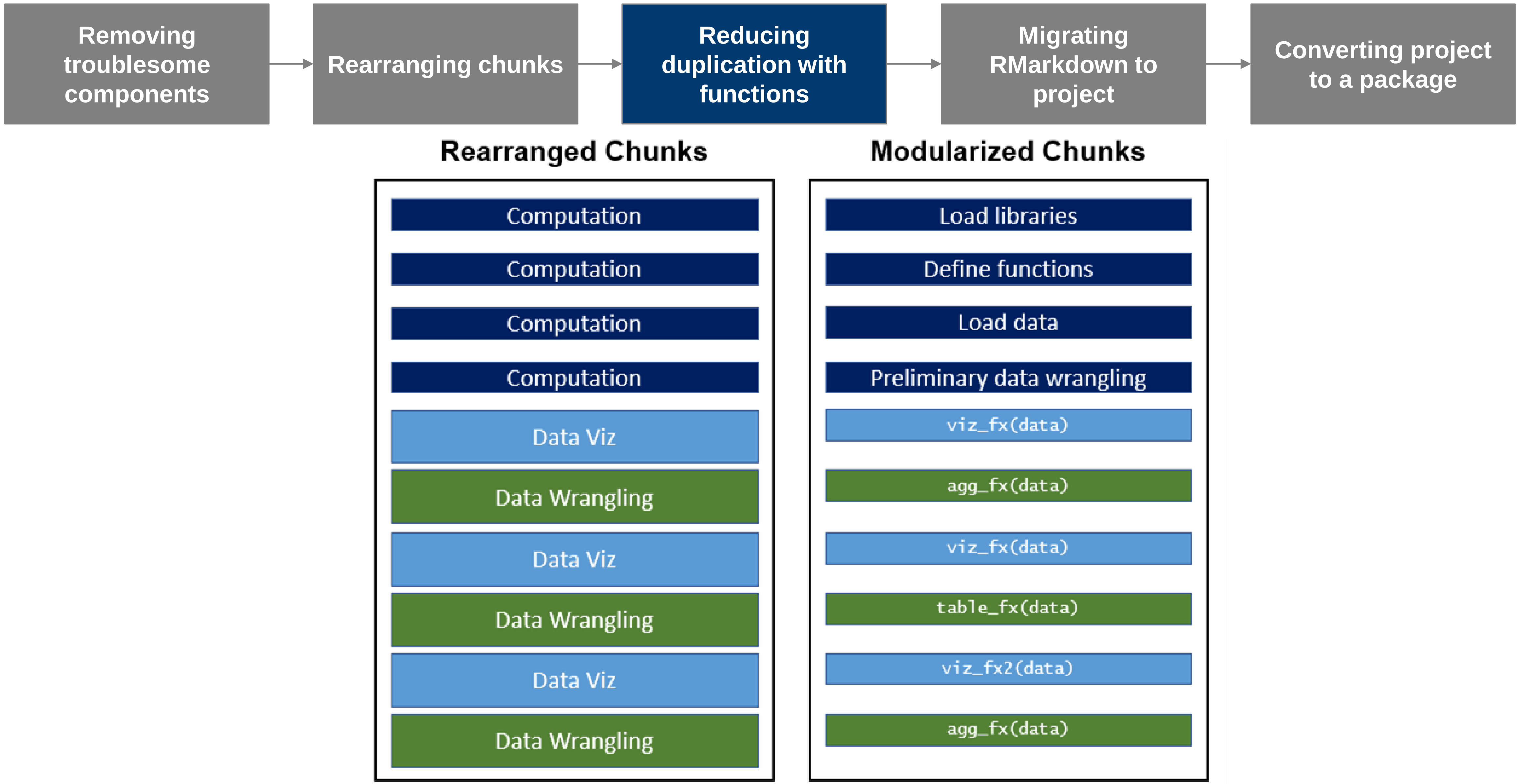
~~~
---
title: "My Analysis"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

```{r pkg-load}
```

```{r data-load}
```

```{r data-clean}
# apply data quality checks ----

# identify and remove outliers ----

# impute missing values ----
```

## Introduction

## Exploratory Data Analysis
~~~

Expandable TOC allows you to jump to your Markdown headers (#)

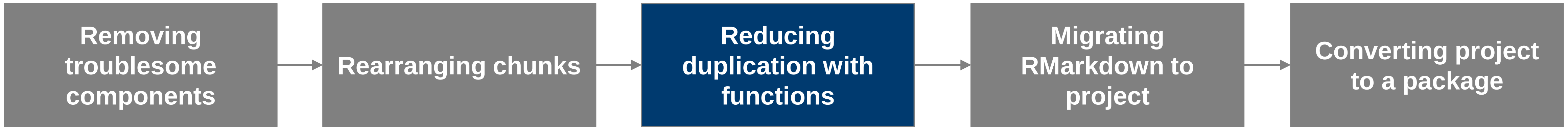
Named chunks create bookmark on nav bar and encourage semantically grouped chunks

Comments followed by four dashes create expand/contract button in margin and bookmark on nav bar

Writing functions eliminates duplication and increases code readability



Writing functions eliminates duplication and increases code readability


Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package

~~~
## Exploratory Analysis

```{r}
ggplot(data, aes(x,y)) + geom_point()
```
```{r}
ggplot(data, aes(x,z)) + geom_point()
```
```{r}
ggplot(data, aes(x,w)) + geom_point()
```
~~~

~~~
```{r fx-viz-scatter-x}

viz_scatter_x <- function(data, vbl) {
  ggplot(
    data = data,
    mapping = aes(x = x, y = {{vbl}}) +
  geom_point()
}

```


## Exploratory Analysis

```{r viz-scatter-x }
viz_scatter_x(data, y)
viz_scatter_x(data, z)
viz_scatter_x(data, w)
```
~~~

roxygen2 function documentation can give your script a package-like understandability

Removing troublesome components → Rearranging chunks → **Reducing duplication with functions** → Migrating RMarkdown to project → Converting project to a package

~~~
```{r fx-viz-scatter-x}

#' Scatterplot of variable versus x
#'
#' @param data Dataset to plot. Must contain variable named x
#' @param vbl Name of variable to plot on y axis
#'
#' @return ggplot2 object
#' @import ggplot2
#' @export

viz_scatter_x <- function(data, vbl) {
  ggplot(
    data = data,
    mapping = aes(x = x, y = {{vbl}}) +
  geom_point()
}

```
~~~

RStudio: Ctrl + Alt + Shift + R for skeleton

Get a virtual second pair of eyes on your polished single-file RMarkdown

Removing troublesome components → Rearranging chunks → **Reducing duplication with functions** → Migrating RMarkdown to project → Converting project to a package

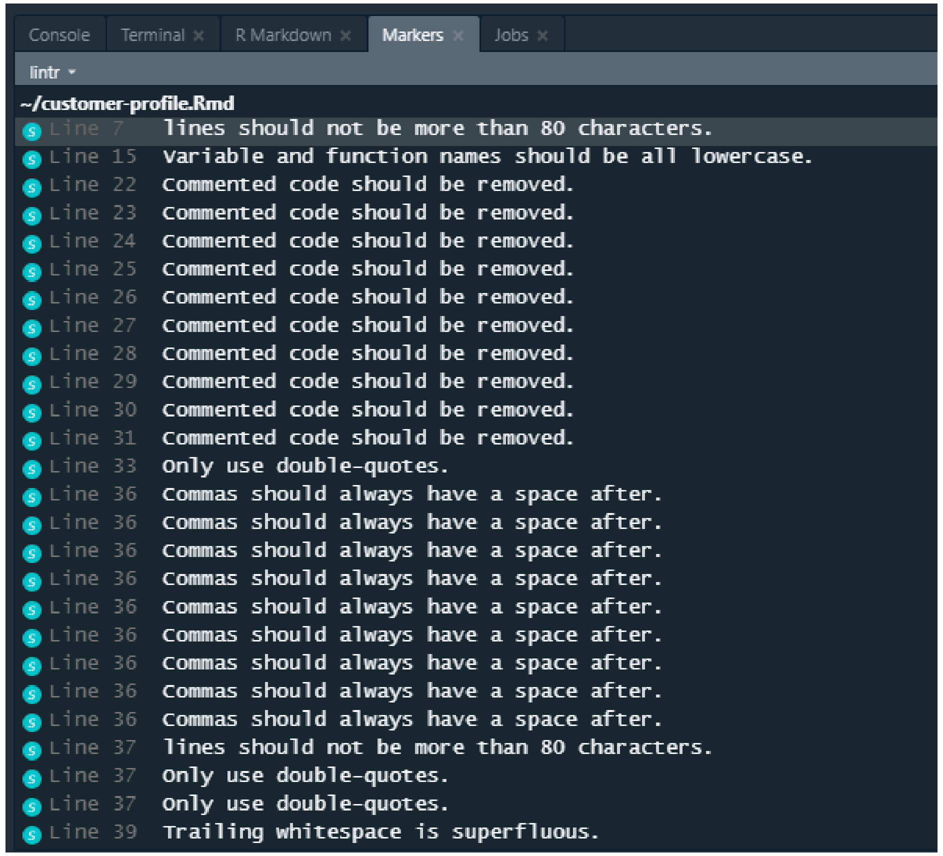
Automatically find areas of improvement with `lintr`, `styler`, and `spelling`

lintr — Analyses code and points out potential style violations

styler — Automatically reformats code with built-in style guides

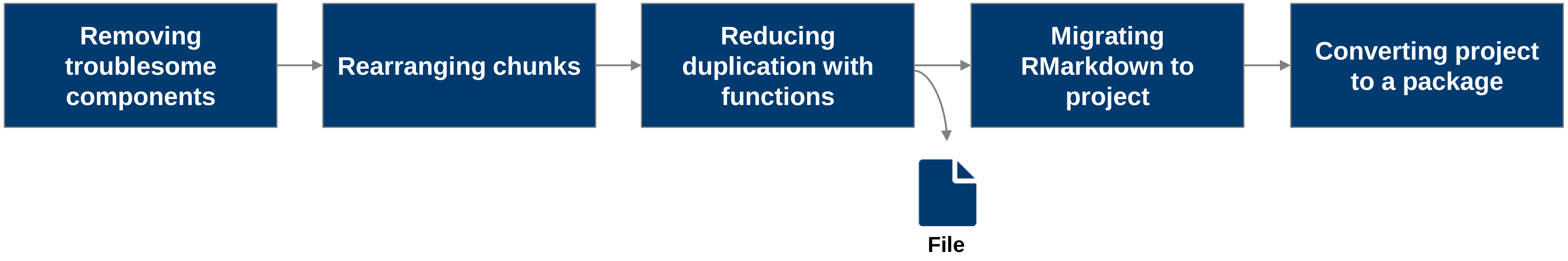
spelling — Highlight typos in code

```
> lintr::lint('my-analysis.Rmd')
```



```
~/customer-profile.Rmd
Line 7   lines should not be more than 80 characters.
Line 15  Variable and function names should be all lowercase.
Line 22  Commented code should be removed.
Line 23  Commented code should be removed.
Line 24  Commented code should be removed.
Line 25  Commented code should be removed.
Line 26  Commented code should be removed.
Line 27  Commented code should be removed.
Line 28  Commented code should be removed.
Line 29  Commented code should be removed.
Line 30  Commented code should be removed.
Line 31  Commented code should be removed.
Line 33  Only use double-quotes.
Line 36  Commas should always have a space after.
Line 36  Commas should always have a space after.
Line 36  Commas should always have a space after.
Line 36  Commas should always have a space after.
Line 36  Commas should always have a space after.
Line 36  Commas should always have a space after.
Line 36  Commas should always have a space after.
Line 36  Commas should always have a space after.
Line 36  Commas should always have a space after.
Line 37  lines should not be more than 80 characters.
Line 37  Only use double-quotes.
Line 37  Only use double-quotes.
Line 39  Trailing whitespace is superfluous.
```

A polished single-file RMarkdown can be a very practical end-state for maximum portability


Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package
File

Standalone File

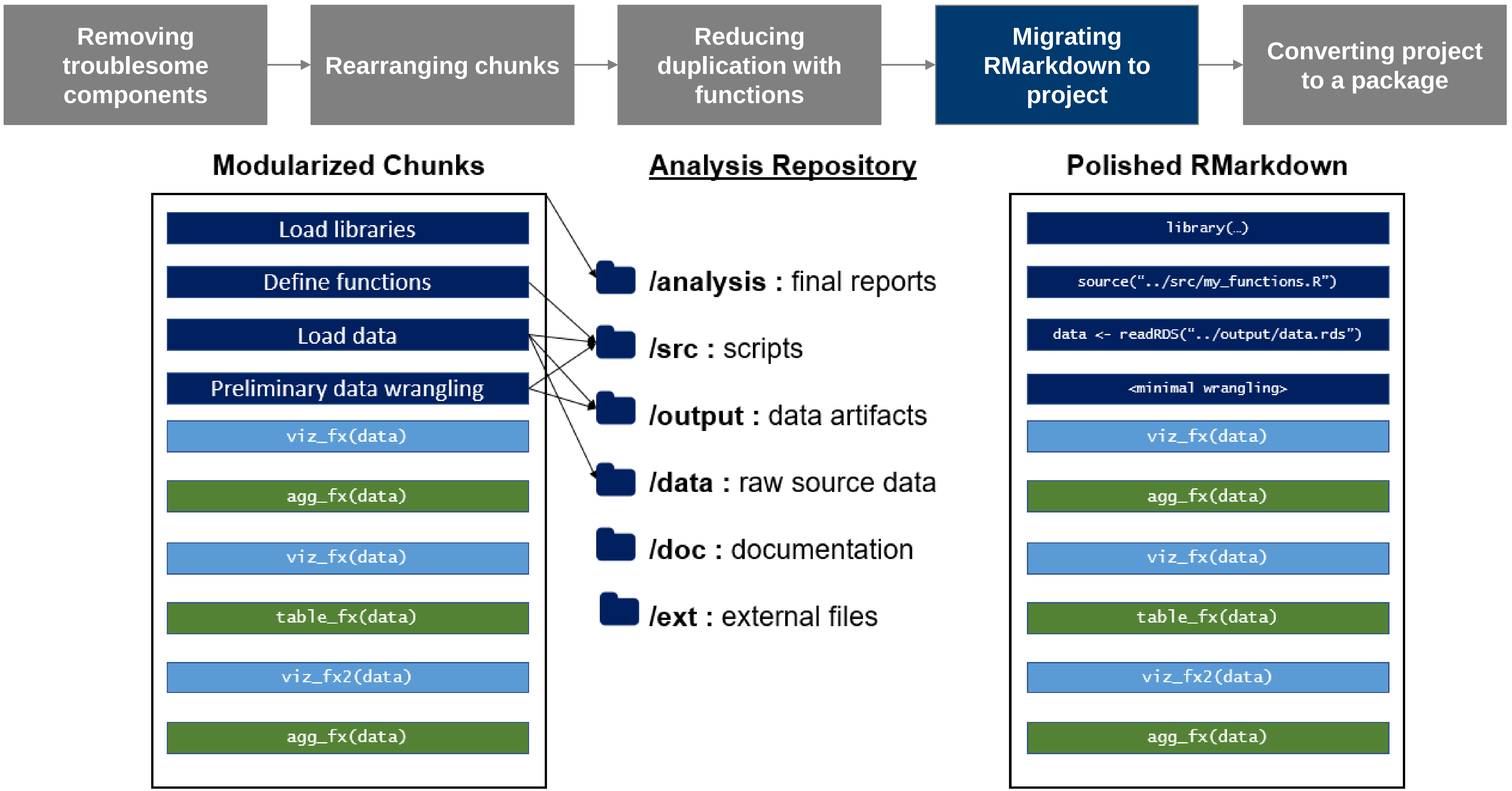
Benefits

- Portable without formal repository
- Easy to compare versions with diffs without formal version control
- One-push execution / refresh

Pitfalls

- Can be lengthy, monolithic, and intimidating
- Potentially slow to run and relies on RMarkdown to play role of job scheduler
- Enables antipatterns (e.g. not saving artifacts)

Projects modularize components and make it easy to access individual project assets



```
library(…)
source("../src/my_functions.R")
data <- readRDS("../output/data.rds")
<minimal wrangling>
viz_fx(data)
agg_fx(data)
viz_fx(data)
table_fx(data)
viz_fx2(data)
agg_fx(data)
```

The `source()` function enables us to execute R code from another script

Removing troublesome components → Rearranging chunks → Reducing duplication with functions → **Migrating RMarkdown to project** → Converting project to a package

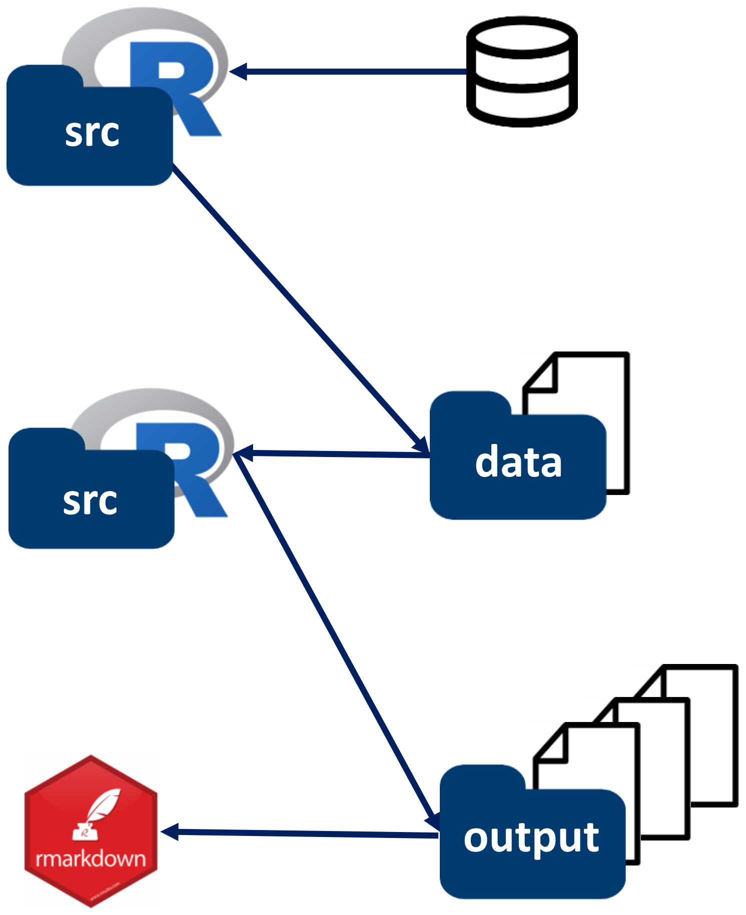
~~~
```{r fx-viz-scatter-x}
#' Scatterplot of variable versus x
#'
#' @param data Dataset to plot. Must contain variable named x
#' @param vbl Name of variable to plot on y axis
#'
#' @return ggplot2 object
#' @import ggplot2
#' @export
viz_scatter_x <- function(data, vbl) {
  ggplot(
    data = data,
    mapping = aes(x = x, y = {{vbl}}) +
  geom_point()
}
~~~
Exploratory Analysis

```{r viz-scatter-x }
viz_scatter_x(data, y)
viz_scatter_x(data, z)
viz_scatter_x(data, w)
```
```

```
```{r load-fx}
source(here('src', 'viz-scatter-x.R'))
```
Exploratory Analysis

```{r viz-scatter-x }
viz_scatter_x(data, y)
viz_scatter_x(data, z)
viz_scatter_x(data, w)
```
```

```
#' Scatterplot of variable versus x
#'
#' @param data Dataset to plot. Must contain variable named x
#' @param vbl Name of variable to plot on y axis
#'
#' @return ggplot2 object
#' @import ggplot2
#' @export
viz_scatter_x <- function(data, vbl) {
  ggplot(
    data = data,
    mapping = aes(x = x, y = {{vbl}}) +
  geom_point()
}
```
```

## Pre-processing data decreases external system dependencies and knitting time

Removing troublesome components → Rearranging chunks → Reducing duplication with functions → **Migrating RMarkdown to project** → Converting project to a package



Load data outside of Rmd to eliminate dependence on API, Database, etc. being ‘up’ when need to knit

Store ‘raw’ data for posterity and reproducibility

Store analytical artifacts (e.g. lean models, aggregate data) to read in to final report

## There are many tools to help make a project, but consistency is the key!

Removing troublesome components → Rearranging chunks → Reducing duplication with functions → **Migrating RMarkdown to project** → Converting project to a package

### Standardized File Structure

usethis

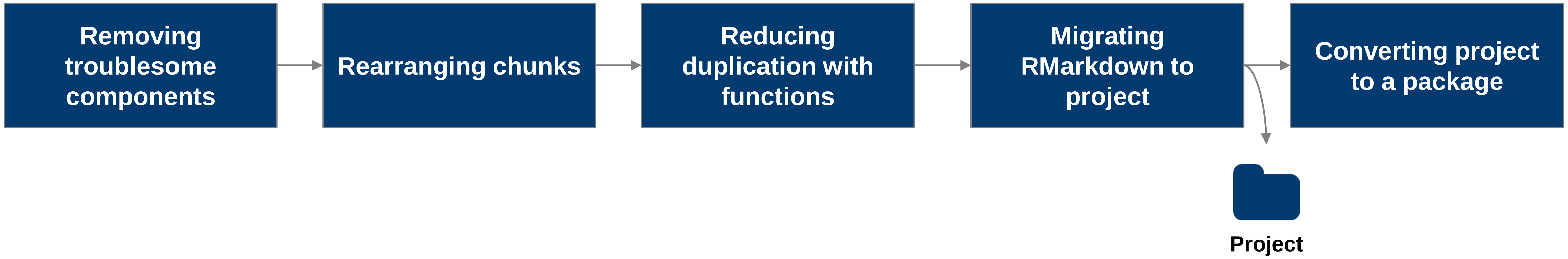
Project Template

starters

### Dependency Management

renv

# R projects preserve problem-specific context while making it easy to reapply components

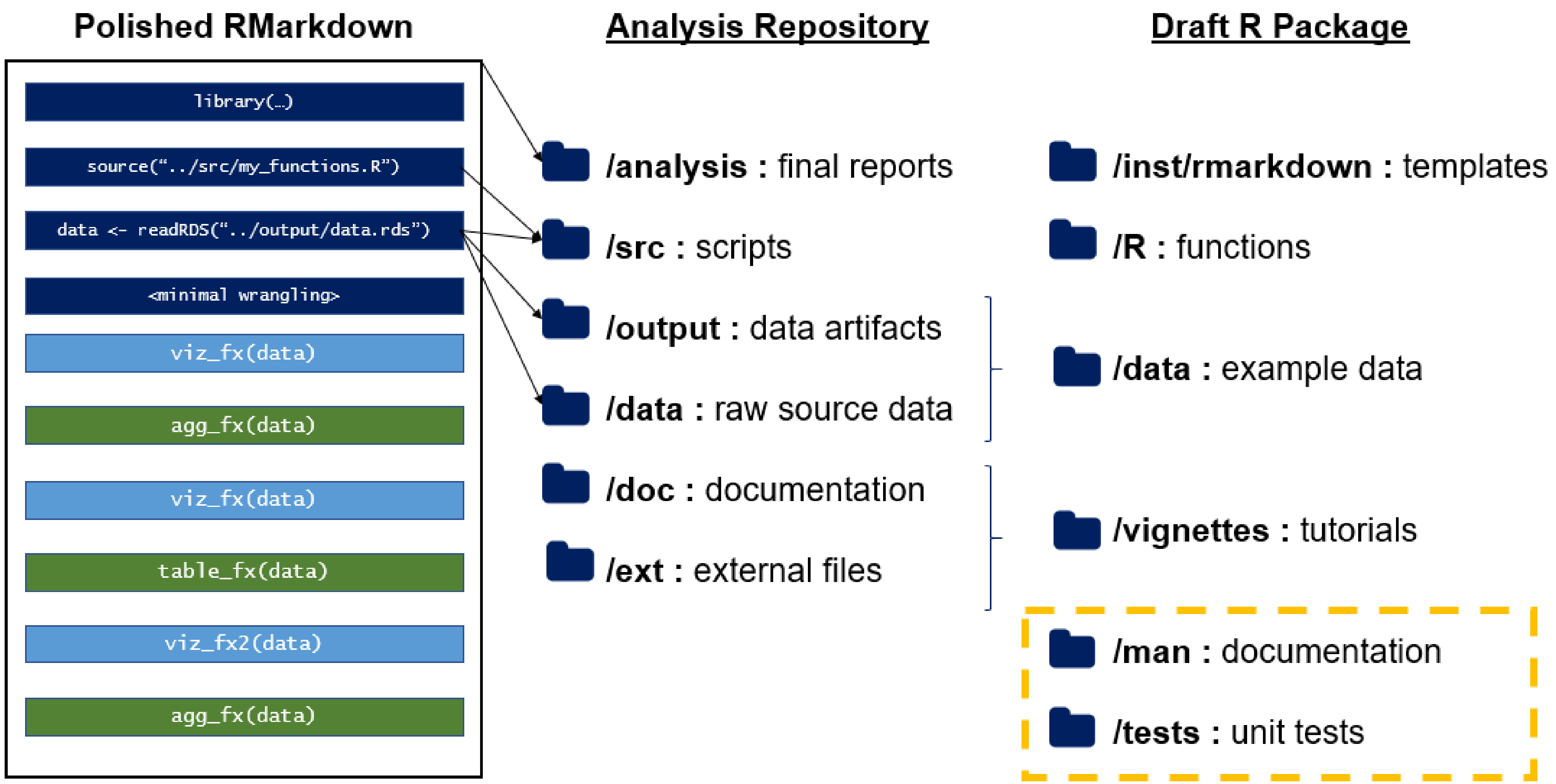

Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package
Project

Project

| | Benefits | Pitfalls |
|---|---|---|
| Project | • Flexible to extract small proportion of functionality or modify at will<br>• Preserves problem-specific context (when desirable) | • The line between analysis and code may be unclear<br>• Can't make full use of developer tools |

# There is a near one-to-one mapping between the components of a project and a package

Removing troublesome components → Rearranging chunks → Reducing duplication with functions → Migrating RMarkdown to project → **Converting project to a package**



```
library(…)
source("../src/my_functions.R")
data <- readRDS("../output/data.rds")
<minimal wrangling>
viz_fx(data)
agg_fx(data)
viz_fx(data)
table_fx(data)
viz_fx2(data)
agg_fx(data)
```

Developer tools exist to help us create everything we need – and more!

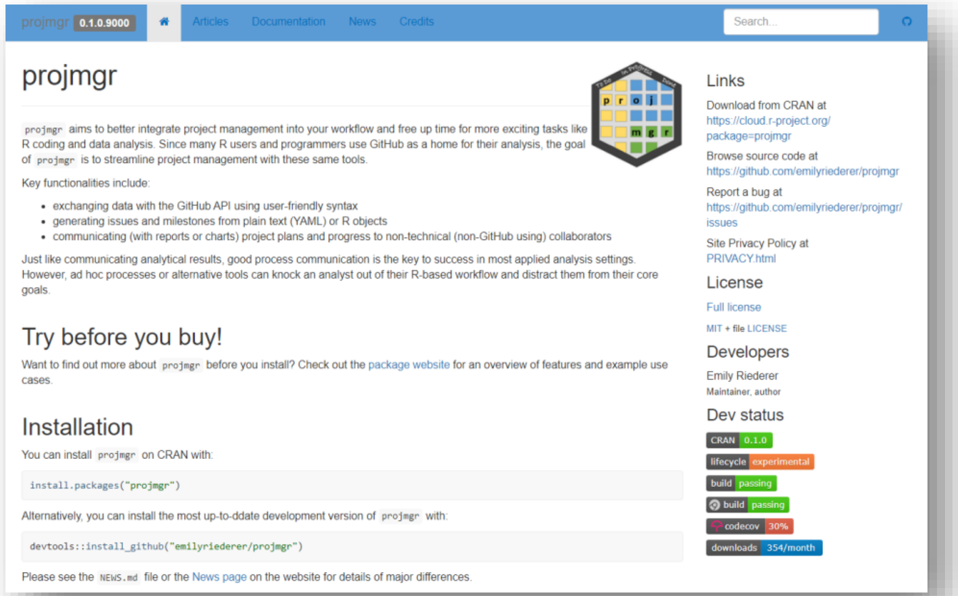
Removing troublesome components → Rearranging chunks → Reducing duplication with functions → Migrating RMarkdown to project → **Converting project to a package** → Package

Sets up all of the folders and configuration files to ensure your package assets are put in the right place

Autogenerates documentation (`man/` folder) from your roxygen2 function comments

Provides high level interface for writing and running unit tests

Renders a polished, user-friendly website from package metadata



projmgr

`projmgr` aims to better integrate project management into your workflow and free up time for more exciting tasks like R coding and data analysis. Since many R users and programmers use GitHub as a home for their analysis, the goal of `projmgr` is to streamline project management with these same tools.

Key functionalities include:

- exchanging data with the GitHub API using user-friendly syntax
- generating issues and milestones from plain text (YAML) or R objects
- communicating (with reports or charts) project plans and progress to non-technical (non-GitHub using) collaborators

Just like communicating analytical results, good process communication is the key to success in most applied analysis settings. However, ad hoc processes or alternative tools can knock an analyst out of their R-based workflow and distract them from their core goals.

Try before you buy!

Want to find out more about `projmgr` before you install? Check out the package website for an overview of features and example use cases.

Installation

You can install `projmgr` on CRAN with:

```
install.packages("projmgr")
```

Alternatively, you can install the most up-to-ddate development version of `projmgr` with:

```
devtools::install_github("emilyriederer/projmgr")
```

Please see the `NEWS.md` file or the News page on the website for details of major differences.

Links

Download from CRAN at https://cloud.r-project.org/package=projmgr

Browse source code at https://github.com/emilyriederer/projmgr

Report a bug at https://github.com/emilyriederer/projmgr/issues

Site Privacy Policy at PRIVACY.html

License

Full license

MIT + file LICENSE

Developers

Emily Riederer
Maintainer, author

Dev status

CRAN 0.1.0
lifecycle experimental
build passing
build passing
codecov 30%
downloads 354/month

Different stopping points optimize for recreation versus extension of your work

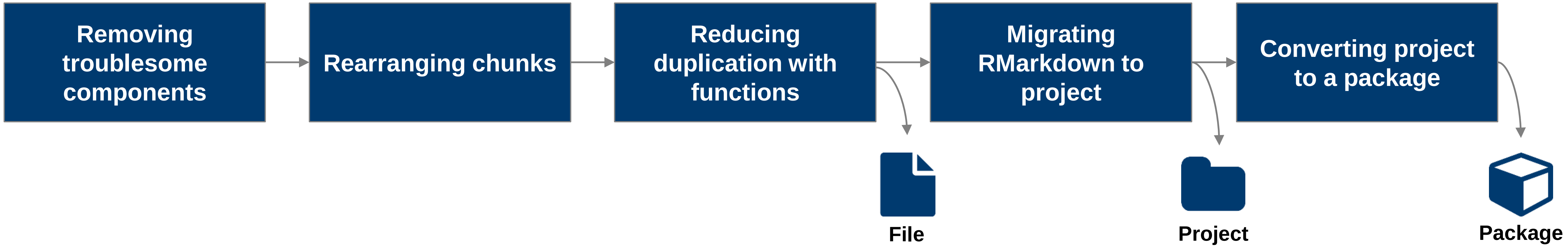
	Benefits	Pitfalls
Standalone File	• Portable without formal repository • Easy to compare versions with diffs without formal version control • One-push execution / refresh	• Can be lengthy, monolithic, and intimidating • Potentially slow to run and relies on RMarkdown to play role of job scheduler • Enables antipatterns (e.g. not saving artifacts)
Project	• Flexible to extract small proportion of functionality or modify at will • Preserves problem-specific context (when desirable)	• The line between analysis and code may be unclear • Can't make full use of developer tools
Package	• Formal mechanisms for distributing at scale (e.g. CRAN) • Familiar format for others to learn and use	• May be too narrowly focused and inflexible if built towards specific project • Potentially more challenging to extract specific features from for interactive use

Specific Instance

↕

Generic Class

No matter what path you chose, your RMarkdown analysis is closer to a sustainable and empathetic data product than you may think!


Removing troublesome components
Rearranging chunks
Reducing duplication with functions
Migrating RMarkdown to project
Converting project to a package
File
Project
Package

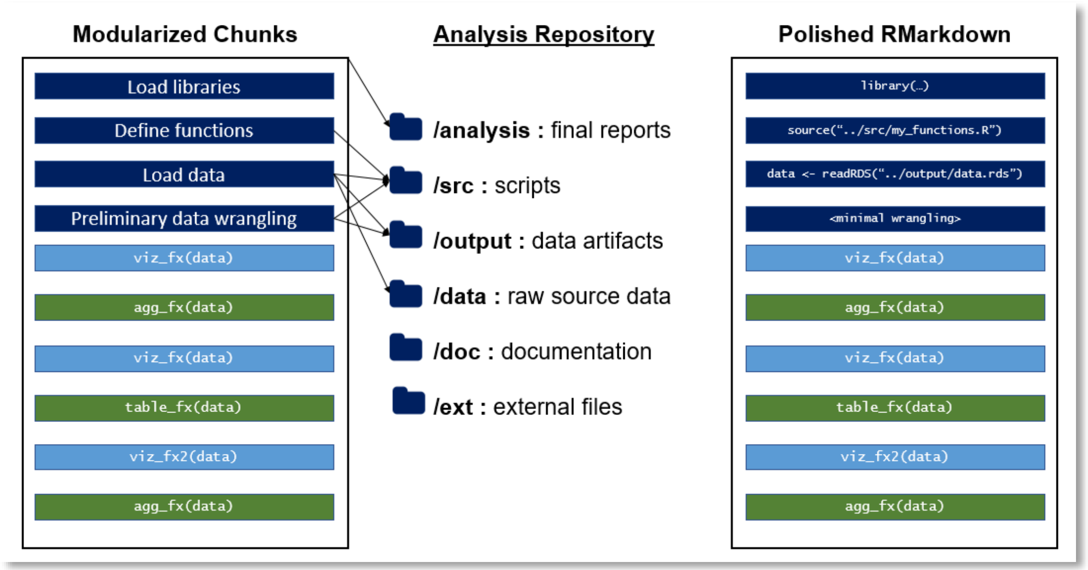
Emily Riederer

@emilyriederer

tiny.cc/rmddd

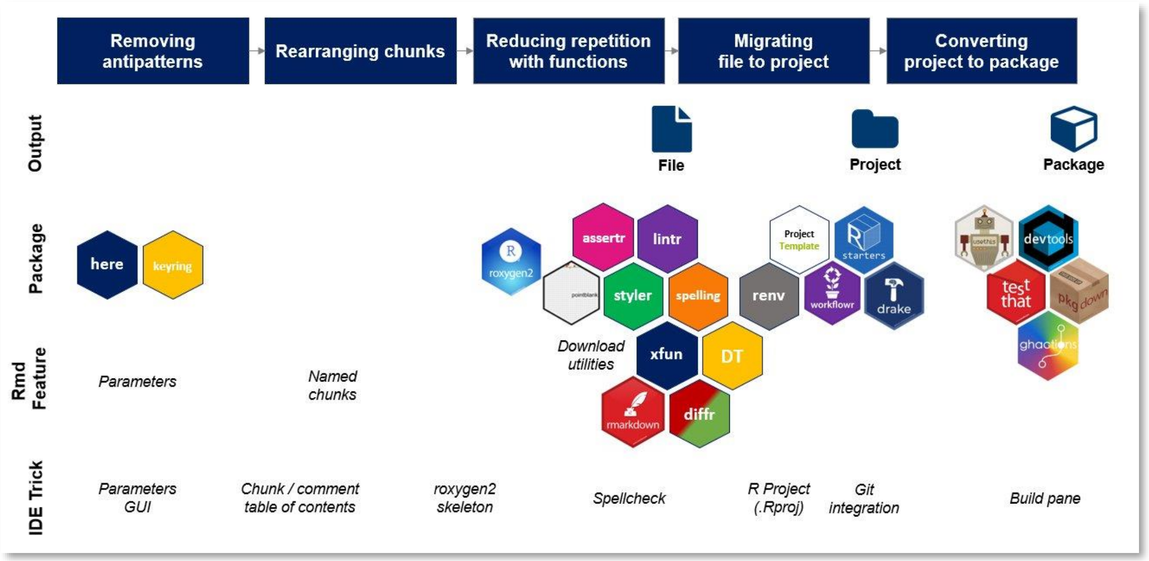
Please get in touch or see related blog posts for more details

RMarkdown Driven Development



tinyurl.com/rmddd

Technical Appendix



tinyurl.com/rmddd-appendix

Emily Riederer

@emilyriederer