# Machine Learning in R: Workshop Series

## Structuring Data Projects

Simon Schölzel

*Research Team Berens*

2020-11-02 (updated: 2020-10-29)

# Agenda

**1 Learning Objectives**

**2 Introduction to RStudio Projects (`.Rproj`)**

**3 R Markdown Documents (`.Rmd`)**

> 3.1 Document Structure I: Header
> 3.2 Document Structure II: Text Body
> 3.3 Document Structure III: Code Chunks & Output (*incl. live demo*)
> 3.4 Generating High-Quality Reports

**4 Interactive Data Science Environments**

> 4.1 Introduction to Computational Notebooks
> 4.2 R Markdown Documents vs. R Notebooks (*incl. live demo*)

**5 `renv`: Project Environments**

> 5.1 Library Paths
> 5.2 Features of `renv`
> 5.3 The `renv` Workflow
> 5.4 Additional Remarks

# 1 Learning Objectives 💡

This workshop should support you in optimizing your overall data science and machine learning workflow by setting up an RStudio project, conducting your analyses in an interactive notebook environment and maintaining local project libraries (so-called *virtual environments*).

More specifically, after this workshop you will

- be familiar with the main features of RStudio projects (`.Rproj`) as well as the differences between absolute and relative directory paths,
- know your way around R Markdown documents (`.Rmd`) and embrace R Notebooks as your primary tool for doing interactive data science and machine learning in `R`,
- see the value in using local project environments to prevent dependency errors, promote collaboration, and ensure reproducibility.

# 2 Introduction to RStudio Projects

# 2 Introduction to RStudio Projects (`.Rproj`)

**What is an RStudio project?**
RStudio projects offer you a way to structure your data science project and improve your workflow:

- Keep all the files related to your data science project in a single place.
- Stop working on one or multiple `R scripts` and return later to continue where you left.
- Structure your project or analysis in a modular manner.
- Easily share your whole project with collaborators.

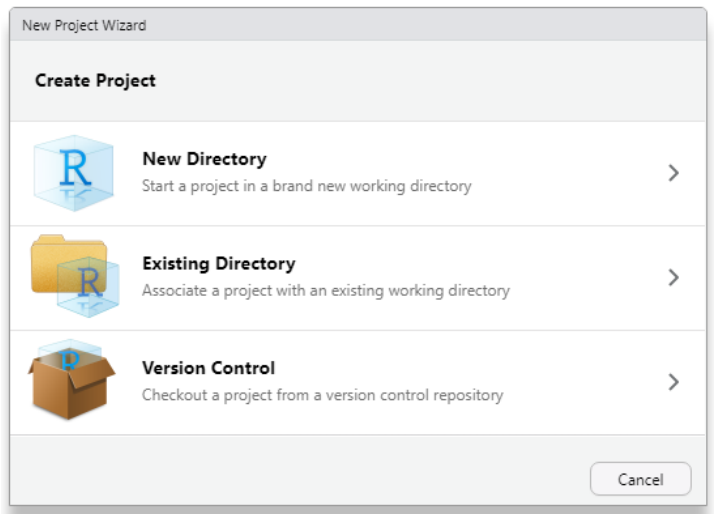
# 2 Introduction to RStudio Projects (`.Rproj`)

**What is an RStudio project?**
RStudio projects offer you a way to structure your data science project and improve your workflow:

- Keep all the files related to your data science project in a single place.
- Stop working on one or multiple `R scripts` and return later to continue where you left.
- Structure your project or analysis in a modular manner.
- Easily share your whole project with collaborators.

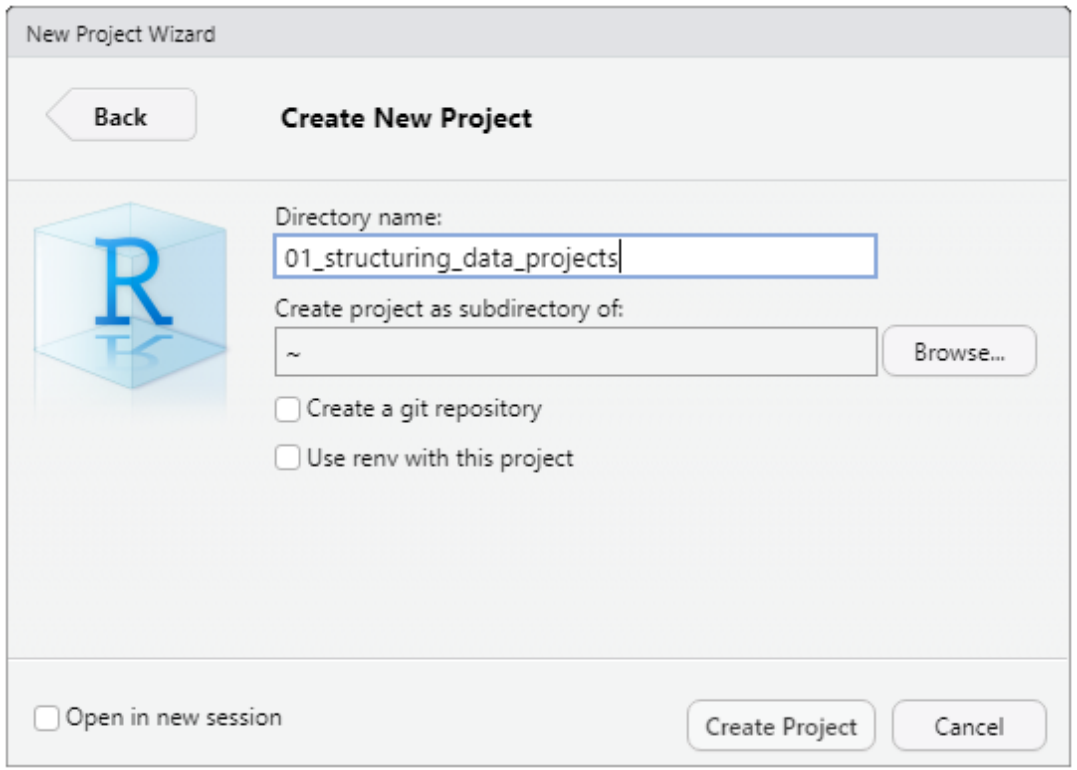
**How do I create an RStudio project?**
RStudio provides three ways of creating an `.Rproj` (*File -> New Project...*):

1. As a new file directory
2. As part of an existing file directory
3. By cloning a version control repository (e.g., from GitHub)



# 2 Introduction to RStudio Projects (`.Rproj`)

Here, I am creating a brand new directory, called `01_structuring_data_projects` in which I create this presentation:



# 2 Introduction to RStudio Projects (`.Rproj`)

**What happens when a new project is created?**

- RStudio creates a `.Rproj` file within the specified directory. This file contains certain project options and enables to open the RStudio project from your machine's filesystem.
- RStudio creates a hidden directory (`.Rproj.user`) where project-related temporary files (e.g. auto-saved documents) are stored.
- RStudio loads the `.Rproj` file and displays the project name on the far right side of the main toolbar.

# 2 Introduction to RStudio Projects (`.Rproj`)

**What happens when a new project is created?**

- RStudio creates a `.Rproj` file within the specified directory. This file contains certain project options and enables to open the RStudio project from your machine's filesystem.
- RStudio creates a hidden directory (`.Rproj.user`) where project-related temporary files (e.g. auto-saved documents) are stored.
- RStudio loads the `.Rproj` file and displays the project name on the far right side of the main toolbar.

**What happens when a new RStudio Project is opened?**

- A fresh `R` session is started.
- The `.RData` file in the project directory is loaded (depending on the project options).
- The `.Rhistory` file in the project directory is loaded into the *RStudio history pane*.
- Previously edited scripts are restored.
- The current *working directory* is set to the project directory (as previously specified, see slide 6).
- ...

# 2 Introduction to RStudio Projects (`.Rproj`)

Let's have a look at the project directory for this presentation (maintained as an `.Rproj`):

```
abs_path <- getwd()
abs_path
```

```
> [1] "C:/Users/.../workshops/01_structuring_data_projects"
```

This path is referred to as an **absolute path**. What files do we find under this path?

```
dir(abs_path)
```

```
>  [1] "01_rstudio_projects.Rmd"        "02_r_notebooks.Rmd"             "03_project_environments.Rmd"
>  [4] "custom"                         "demo"                           "img"
>  [7] "index.html"                     "index.Rmd"                      "libs"
> [10] "README.md"                      "renv"                           "renv.lock"
> [13] "slides"                         "structuring_data_projects.Rproj"
```

This is basically all the stuff required for rendering this presentation. Put differently: With an `.Rproj`, I organize all the files related to the project in one folder (e.g., script files, data sets, figures, manuscripts, etc.).

# 2 Introduction to RStudio Projects (`.Rproj`)

Instead of using `dir(abs_path)`, you may also use `dir(".")`. When specifying a path on your computer, the single dot ( `.` ) reflects a **relative path**, i.e. a path relative to your current working directory. Hence, it again refers to:

```
> [1] "C:/Users/.../workshops/01_structuring_data_projects"
```

Now, let's assume you want to access `data.txt` in the subfolder `data`, i.e. you search for this file under:

```
> [1] "C:/Users/.../workshops/01_structuring_data_projects/data"
```

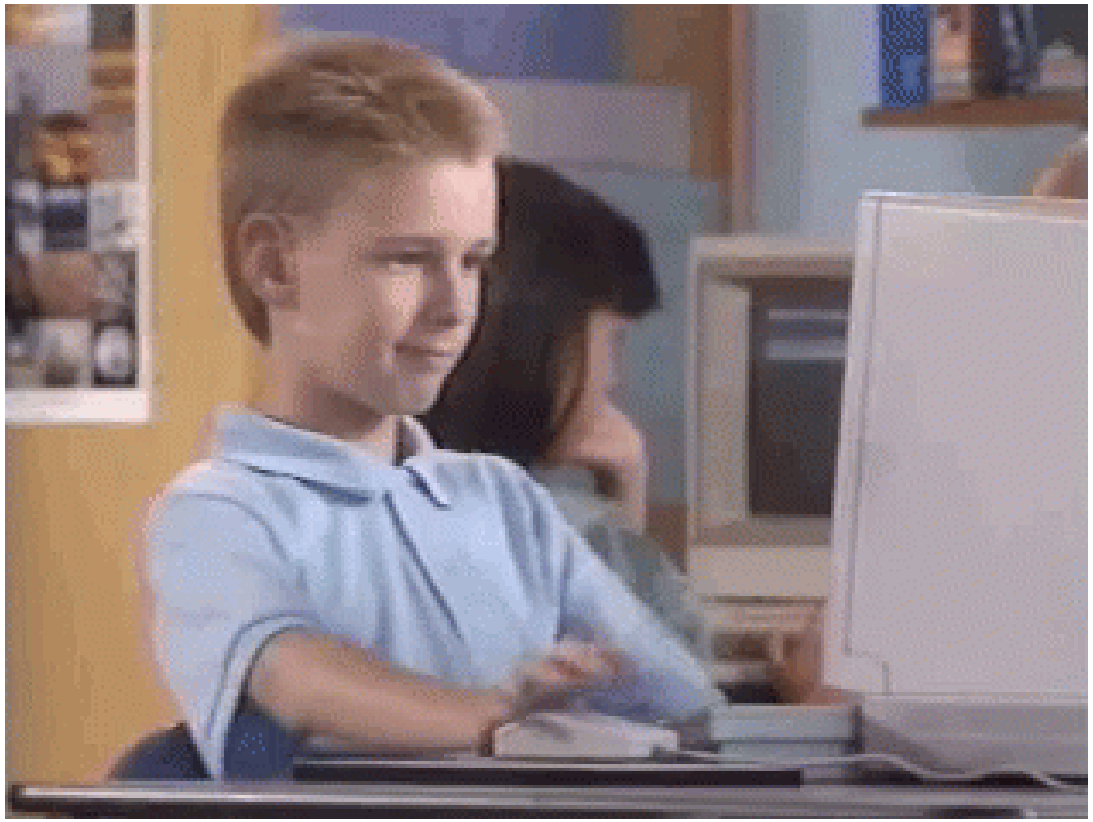
What is the corresponding relative path when reading in the data?

```
read.delim("./data/data.txt", header = T, sep = ",")
```

The use of relative paths makes it very easy to navigate your project or share it with collaborators without having to adjust every path manually!

*Note: If you want to step one folder up in the folder hierarchy, you may use two dots ( `..` ). When writing paths on Windows, use `/` instead of `\` , otherwise `R` interprets it as an escape character.*

# 2 Introduction to RStudio Projects (`.Rproj`)



# 2 Introduction to RStudio Projects (`.Rproj`)

**Project options:** Access project options within RStudio by navigating to *Tools -> Project Options...* .

- **Save workspace to `.RData` on exit:** Stores your current `global environment` (workspace) in an `.RData`-file in your project directory when closing your RStudio project.

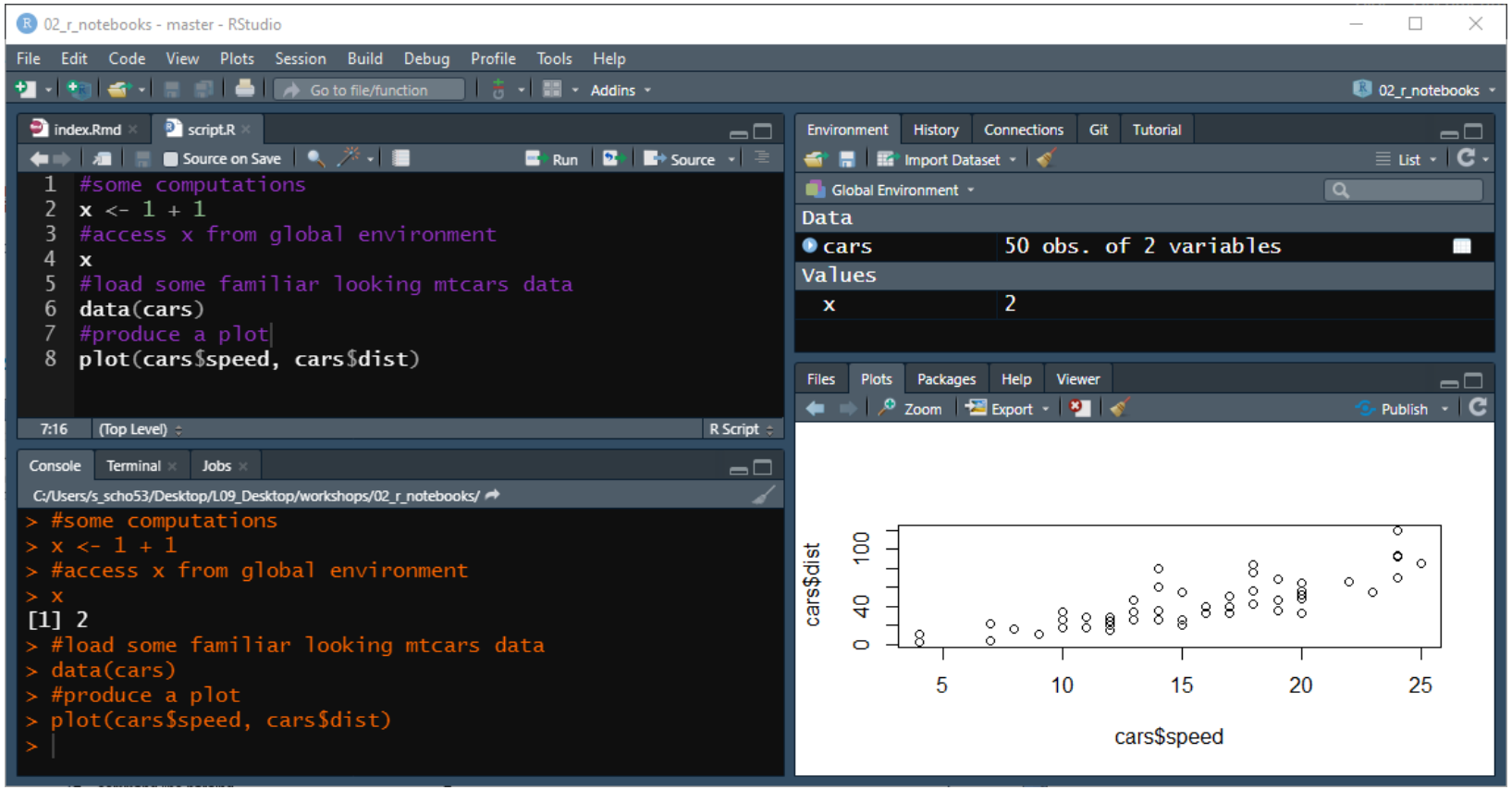
  *Note: Not recommended! You should always be able to recreate your global environment from your scripts. Hence, I would set this behavior of Rstudio to "never" (Tools -> Global Options...).*

- **Restore `.RData` into workspace at startup:** Loads your previously saved workspace from the `.RData` file into the `global environment`.

- **Always save history:** Writes your code history (see history pane) into an `.Rhistory` file located in your project directory.

# 3 R Markdown Documents

# 3 The Story So Far: Rscripts (`.R`)



# 3 R Markdown Documents (`.Rmd`)


**The `rmarkdown` package:**

- Markdown-based authoring framework for data science
- Embed `R` Code and plain text
- Generate high-quality reports

**Document elements:**

- Header
- Text body
- Code & output

*Note: The different document elements apply to R Markdown Documents (`.Rmd`-files) and do not appear in common R Scripts (`.R`-files).*


What is R Markdown? from RStudio, Inc. on Vimeo.

# 3 R Markdown Documents (`.Rmd`)


## 3.1 The `YAML` Header

The **header** appears at the top of your `.Rmd` document. It contains a bunch of metadata as well as formatting options for your final report. The syntax for the header is based on YAML.

```
---
title: "Rmarkdown Demo"    #Title/headline of your markdown report
author: John Doe           #Name of the author
date: 2020-11-02           #Date of the report
output:                    #YAML options specifying the output
  html_document:               #Output format
    toc: true                  #Table of contents (toc)
    toc_float: true            #Floating toc on the left side of the document
    toc_depth: 2               #Depth of headers to generate toc
---
```

# 3 R Markdown Documents (`.Rmd`)


## 3.2 The Text Body

```
You can write plain text,

in *italics* or in **bold**,

and use S^uperscript^ or S~ubscript~.

You can format text as `fancy R code`,

and even embed LaTeX equations:
$$
x_{1/2} = -\frac{p}{2}\pm \sqrt{(\frac{p}{2})^2-q}
$$
```

You can write plain text,

in *italics* or in **bold**,

and use S^uperscript^ or S~ubscript~.

You can format text as `fancy R code`,

and even embed LaTeX equations:

$$x_{1/2} = -\frac{p}{2} \pm \sqrt{(\frac{p}{2})^2 - q}$$

# 3 R Markdown Documents (`.Rmd`)


## 3.2 The Text Body

```
# You can use headers (H1)
```

# You can use headers (H1)

```
## You can use headers (H2)
```

## You can use headers (H2)

```
### You can use headers (H3)
```

### You can use headers (H2)

# 3 R Markdown Documents (`.Rmd`)


## 3.2 The Text Body

```
> You can highlight quotes,
```

> You can highlight quotes,

```
or include images: ![](url),
```

or include images:

,

```
embed [Hyperlinks](url),
```

embed Hyperlinks,

# 3 R Markdown Documents (`.Rmd`)


## 3.2 The Text Body

```
- You can
- create
  - an unordered
  - list
    - of items

1. Or even
2. better
3. you create an ordered list
  1. which should
  2. also look
    1. familiar!
```

```
And finally, since we are in an academic context
you might also want to use some footnotes.[^1]

[^1]: Insert footnote
```

And finally, since we are in an academic context you might also want to use some footnotes.[1]

[1] Insert footnote.

# 3 R Markdown Documents (`.Rmd`)


## 3.3 Code Chunks & Output

**Code Chunks:** Self-contained part of your `R` code for which, upon code evaluation, the output of your code is shown directly below the chunk instead of only in your console or the RStudio viewer or the plots pane.

````
```{r fibonacci}
fib <- function(n) {
  x <-  numeric(n)
  x[1:2] <-  c(1,1)
  for(i in 3:n) {
    x[i] = x[i-1] + x[i-2]
  }
  return(x)
}

fib(10)
```
````

```
 [1]  1  1  2  3  5  8 13 21 34 55
```

**Creating code chunks**:
Using the *insert* button in the RStudio interface 

, or via the keyboard shortcut: *Ctrl + Alt + I*.

*Note: In fact, code chunks even allow you to write and evaluate code in other languages, such as Python, SQL or C++.*

# 3 R Markdown Documents (`.Rmd`)


## 3.3 Live Demo

02:00

# 3 R Markdown Documents (`.Rmd`)


## 3.3 Code Chunks & Output

**Chunk Options:** Can be set as boolean (`True`/`False`) arguments in the chunk header `{r ...}`.

| Argument | Description |
|---|---|
| include=F | Code and Output does not appear in the final report (yet, code is still evaluated). |
| eval=F | Output does not appear in the final report as the code is not evaluated. |
| result=F | Output does not appear in the final report but the code is evaluated. |
| echo=F | Code does not appear in the final report. |
| message=F | Messages generated by the code do not appear in the final report. |
| warning=F | Warnings generated by the code do not appear in the final report. |

# 3 R Markdown Documents (`.Rmd`)


## 3.4 Generating High-Quality Reports

Render the `.Rmd` document into your desired output format by using `knitr::render()` command from the `knitr` package or hit the *Knit* button in your RStudio toolbar.

# 3 R Markdown Documents (`.Rmd`)


## 3.4 Generating High-Quality Reports

Render the `.Rmd` document into your desired output format by using `knitr::render()` command from the `knitr` package or hit the *Knit* button in your RStudio toolbar.


**Workflow in the background:**

1. `knitr` takes your `.Rmd`-file and converts it into a plain markdown file (`.md`).
2. The `.md`-file is processed by `pandoc` to be converted into the final output format (e.g., PDF, HTML, Word).

*Note: `knitr` can only convert your final `.Rmd`-file if your code is free of errors as `knitr` tries to run your entire code as part of the conversion workflow!*

# 3 R Markdown Documents (`.Rmd`)


# 4 Interactive Data Science Environments

# 4 Interactive Data Science Environments


## 4.1 Introduction to Computational Notebooks

**Computational notebooks** are virtual environments used for Literate Programming. They combine functionalities of word processors (e.g., MS Word) and the programming language kernel (e.g., the R shell which you find in the RStudio console pane).

# 4 Interactive Data Science Environments


## 4.1 Introduction to Computational Notebooks

**Computational notebooks** are virtual environments used for Literate Programming. They combine functionalities of word processors (e.g., MS Word) and the programming language kernel (e.g., the R shell which you find in the RStudio console pane).

**Features:**

- Interactivity
- Iteration
- Sharing & Communication
- Transparency
- Reproducibility


# 4 Interactive Data Science Environments

## 4.1 Introduction to Computational Notebooks

This sounds pretty familiar after learning about R Markdown docs right?
So what is R Notebook all about?

# 4 Interactive Data Science Environments


## 4.2 R Markdown Documents vs. R Notebooks

Primarily, both formats are based on `.Rmd`-files:

- An R Markdown document can be used as a computational notebook,
- and an R Notebook can be rendered to other R Markdown document types (e.g., PDF, HTML, Word).

A notebook (in the `R` universe) can be viewed as a special version of an Markdown document.

# 4 Interactive Data Science Environments


## 4.2 R Markdown Documents vs. R Notebooks

Primarily, both formats are based on `.Rmd`-files:

- An R Markdown document can be used as a computational notebook,
- and an R Notebook can be rendered to other R Markdown document types (e.g., PDF, HTML, Word).

A notebook (in the `R` universe) can be viewed as a special version of an Markdown document.

The use case is determined by your goal: generating high-quality reports/documents with R Markdown documents vs. doing data science / machine learning using R Notebooks.

**R Markdown documents:**

- Execution of the whole code (batch execution)
- Requires code to be free of errors

**R Notebooks:**

- Execution of the code interactively
- Preview is simply a snapshot of your notebook (see `.nb.html`-file)

# 4 Interactive Data Science Environments


## 4.2 Live Demo

02:00

# 5 Project Environments

# 5 `renv`: Project Environments

## 5.1 Library Paths

**Question:** Where does R store my installed packages respectively loads them from?

# 5 `renv`: Project Environments

## 5.1 Library Paths

**Question:** Where does R store my installed packages respectively loads them from?

**Answer:** Your user-specific *active library path*, i.e. on your disk.

# 5 `renv`: Project Environments

## 5.1 Library Paths

**Question:** Where does R store my installed packages respectively loads them from?

**Answer:** Your user-specific *active library path*, i.e. on your disk.

```
.libPaths()
```

```
> [1] "C:/Users/.../Documents/R/R-4.0.2/library"
```

# 5 `renv`: Project Environments

## 5.1 Library Paths

**Question:** Where does R store my installed packages respectively loads them from?

**Answer:** Your user-specific *active library path*, i.e. on your disk.

```
.libPaths()
```

```
> [1] "C:/Users/.../Documents/R/R-4.0.2/library"
```

Packages are stored and loaded from the root installation folder (here for Windows operating systems). `R` crawls this path when loading (`library()`) or downloading (`install.packages()`) packages.

```
find.package("splines")
```

```
> [1] "C:/Users/.../Documents/R/R-4.0.2/library/splines"
```

# 5 `renv`: Project Environments

## 5.1 Library Paths

By default, this path is accessed each time you run a new `R` session or create a new `.Rproj`. Yet, the required version of a given package may vary across projects (e.g., depending on the project's start date):

- Project A requires `dplyr 1.0.0`,
- Project B requires `dplyr 0.8.1`, and
- Project C requires the dev version of `dplyr`.

According to what you have learned, the three projects share the same mutual library path (`C:/Users/.../Documents/R/R-4.0.2/library`). An upgrade or downgrade of the package version increases the chance that one of the projects does not work seamlessly any longer.

# 5 `renv`: Project Environments


## 5.2 Features of `renv`

`R` package for initializing and managing local project environments, i.e. collections of R packages utilized within an *.Rproj*.

**Isolated projects:**

- One local library of R packages per `.Rproj`
- Upgrade/downgrade packages without breaking other projects

**Portable projects:**

- Save the state of your project library in a `lockfile`
- Share the `lockfile` with others and ensure library compatibility

**Reproducible projects:**

- Use `renv::snapshot()` to save the state of your project library
- Use `use renv::restore()` to restore the state of your project library as specified in the lockfile

# 5 `renv`: Project Environments


## 5.3 The `renv` Workflow

**Step 1:**
Set up `renv` by running `renv::init()` to initialize your local environment with a private library of `R` packages specific to your project.

**Step 2:**
Write some code, install, uninstall and update some packages...

**Step 3:**
Use `renv::snapshot()` to save the current state of your project library to the so-called *lockfile*.

**Step 4:**
Write some code, install, uninstall and update some packages...

**Step 5:**
Use `renv::restore()` to revert to the library state encoded in your *lockfile* in case you run into problems while working with updated packages.

# 5 `renv`: Project Environments


## 5.3 The `renv` Workflow: `renv::init()`

```
> install.packages("renv")
> renv::init()
```

# 5 `renv`: Project Environments


## 5.3 The `renv` Workflow: `renv::init()`

```
> install.packages("renv")
> renv::init()
```

Under the hood, `renv::init()` establishes the infrastructure to run your project-specific library. More specifically, the following files are written to and used by your `renv`-enabled projects:

| File | Usage |
|---|---|
| .Rprofile | Used to activate renv whenever a new R session is launched within the project |
| renv.lock | The lockfile, defining the state of your project library |
| renv/activate.R | The activation script run by the project .Rprofile |
| renv/library | Your local project library |

# 5 `renv`: Project Environments


## 5.3 The `renv` Workflow: `renv.lock`

The **lockfile** stores all information necessary to re-install the required packages, e.g., when sharing the project with a collaborator, archiving a project or putting code into productive use (*deployment*). Packages can be downloaded from: `CRAN`, `Bioconductor`, `GitHub`, `Gitlab`, `Bitbucket`.

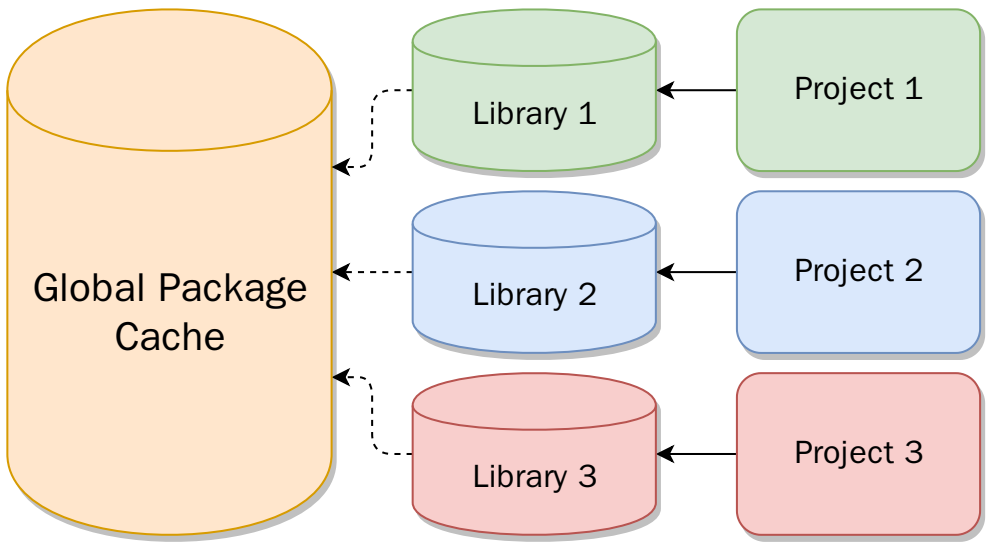
**Exemplary lockfile entry in JSON format:**

```
"renv": {
  "Package": "renv",
  "Version": "0.11.0",
  "Source": "Repository",
  "Repository": "CRAN",
  "Hash": "1c3ef87cbb81c23ac96797781ec7aecc"
}
```

# 5 `renv`: Project Environments


## 5.3 The `renv` Workflow: `renv/library`

`renv` links your project to a **global package cache** where all your downloaded packages are stored by `renv`. You can think of it as a repository that contains all package versions ever installed on your machine.




This approach avoids redundancies, if the same package version is used by multiple projects.

In this case, the installation of the respective package is stored in the global cache with `renv` simply creating a link to this cache within your project. This is what your local **project library** under the path `renv/library` is all about.

Picture by Kevin Ushey.

# 5 `renv`: Project Environments


## 5.3 The `renv` Workflow: `renv/library`

Finally, if you now check your active library path (`.libPaths()`), you can observe a new project-specific path (as well as an additional system library path which you need to consider further at this point).

```
.libPaths()
```

```
> [1] "C:/Users/.../01_structuring_data_projects/renv/library/R-4.0/x86_64-w64-mingw32"
> [2] "C:/Users/.../AppData/Local/Temp/Rtmp0aVyVr/renv-system-library"
```

# 5 `renv`: Project Environments


## 5.3 The `renv` Workflow: `renv::snapshot()`

After having successfully set up our project-specific library, let's consider the two other main features of the `renv` package: `renv::snapshot()` and `renv::restore()`.

**`renv::snapshot()`** updates the lockfile by writing the current state of the project library to the lockfile.

```
renv::snapshot()
```

# 5 `renv`: Project Environments


## 5.3 The `renv` Workflow: `renv::snapshot()`

After having successfully set up our project-specific library, let's consider the two other main features of the `renv` package: `renv::snapshot()` and `renv::restore()`.

**`renv::snapshot()`** updates the lockfile by writing the current state of the project library to the lockfile.

```
renv::snapshot()
```

```
> The following package(s) will be updated in the lockfile:
>
> # CRAN ===============================
> - glue        [1.4.0 -> 1.4.1]
> - jsonlite    [1.6.0 -> 1.7.0]
> - rmarkdown   [* -> 2.3]
>
> Do you want to proceed? [y/N]: y
>  * Lockfile written to 'C:/Users/.../01_structuring_data_projects/renv.lock'.
```

# 5 `renv`: Project Environments


## 5.3 The `renv` Workflow: `renv::restore()`

After having successfully set up our project-specific library, let's consider the two other main features of the `renv` package: `renv::snapshot()` and `renv::restore()`.

**renv::restore()** updates the project library by restoring the state of the project library as specified in the lockfile.

```
renv::restore()
```

# 5 `renv`: Project Environments


## 5.3 The `renv` Workflow: `renv::restore()`

After having successfully set up our project-specific library, let's consider the two other main features of the `renv` package: `renv::snapshot()` and `renv::restore()`.

**renv::restore()** updates the project library by restoring the state of the project library as specified in the lockfile.

```
renv::restore()
```

```
> The following package(s) will be updated:
>
> # CRAN ===============================
> - rmarkdown   [* -> 2.3]
> - stringr     [* -> 1.4.0]
>
> Do you want to proceed? [y/N]: y
> Installing stringr [1.4.0] ... OK (copied cache)
> Installing rmarkdown [2.3] ... OK (copied cache)
```

# 5 `renv`: Project Environments


## 5.4 Additional Remarks

A project is always bound to a particular version of `renv`. Updating the current version of the `renv` package should be done using the `renv::upgrade()` command.

When setting up `renv` for an existing project, it crawls all project scripts and searches for packages that are mentioned in your code via `renv::dependencies()`. For this presentation, I relied, among others, on the following packages (note that packages are only identified if explicitly mentioned in the code, i.e. by name):

```
head(renv::dependencies(), 5)
```

```
> Finding R package dependencies ... Done!

>                                       Source          Package
> 1 C:/Users/s_scho53/.../01_rstudio_projects.Rmd        rmarkdown
> 2 C:/Users/s_scho53/.../01_rstudio_projects.Rmd            knitr
> 3 C:/Users/s_scho53/.../01_rstudio_projects.Rmd            knitr
> 4 C:/Users/s_scho53/.../01_rstudio_projects.Rmd            knitr
> 5      C:/Users/s_scho53/.../02_r_notebooks.Rmd        rmarkdown
```

# Additional Resources

**RStudio Support (2020):** Using Projects. URL: https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects.

**Wickham, H./Grolemund, G. (2017):** R for Data Science: Visualize, Model, Transform, Tidy, and Import Data. URL: https://r4ds.had.co.nz/workflow-projects.html, chapter 8 (Workflow: projects).

---

**RStudio (2020):** Introduction to `Rmarkdown`. URL: https://rmarkdown.rstudio.com/lesson-1.html.

**RStudio (2020):** `Rmarkdown::cheat sheet`. URL: https://raw.githubusercontent.com/rstudio/cheatsheets/master/rmarkdown-2.0.pdf.

**Xie, Y./Allaire, J.J./Grolemund, G. (2020):** R Markdown: The Definitive Guide. URL: https://bookdown.org/yihui/rmarkdown/, chapter 2 (Basics) and 3 (Documents).

---

**Ushey, K. (2020):** `renv`: Project Environments for R. URL: https://kevinushey-2020-rstudio-conf.netlify.app/slides.html#1.

**Ushey, K. (2020):** Introduction to `renv`. URL: https://rstudio.github.io/renv/articles/renv.html.