

Introduction to {box} Package

 [MarEichler/rladies_box_intro](https://github.com/MarEichler/rladies_box_intro)

 [rladies_box_intro/presentation/presentation.pdf](https://github.com/MarEichler/rladies_box_intro/presentation/presentation.pdf)

MARTHA YVONNE EICHLERSMITH

R-Ladies Twin Cities

March 10, 2022

Goal of this talk

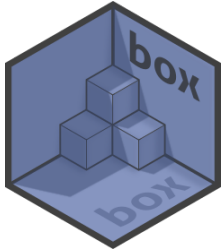
Introduce the `{box}` package and show how it can be used to write modularized/organized code and manage the name space.

Disclaimer: I have been using this package for about 6 months and I'm still learning new things everyday! Please ask as many questions as you want, but I may not know the answer.

Packages Used in the Making of this Presentation/Examples

- `{box}`
- `{cowplot}`: layout multiple plots using `plot_grid()`
- `{glue}`: used to combine strings and R code (similar to `paste/paste0`)
- `{tidyverse}` specifically `{dplyr}`, `{ggplot2}`, `{stringr}`, and `{tidyr}`

{box}



- {box} is a relatively new package
- first version released on GitHub in July 2017
- first version on CRAN in February 2021
- Current version 1.1.0 released in September 2021

Why use {box}?

- Clean-up/Manage name space
- Modular-ize code without having to write full package
 - purposeful and organized code
 - if change code within box module, importation of box module doesn't change

Box modules are like mini-packages: help re-use or share code without the much larger hurdle of developing a whole R package

- re-use code for yourself if doing similar actions frequently
- share code internally with peers/colleague (create consistency)
- share externally

Why use {box}: Name Space

Name space: group of items (data frames, functions, vectors, lists, etc.) that each have a unique name.

```
> library(tidyverse)
-- Attaching packages ----- tidyverse 1.3.1 --
v ggplot2 3.3.5    v purrr  0.3.4
v tibble  3.1.4    v dplyr 1.0.7
v tidyr   1.1.3    v stringr 1.4.0
v readr   2.0.1    v forcats 0.5.1
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
> |
```

- {dplyr} and {stats} use the same name for certain functions, CONFLICT in the namespace
- If want to use the {stats} function, have to refer to the package: stats::filter()

R Environment: Global Environment

The top screenshot shows the RStudio interface with the Global Environment pane empty, displaying "Environment is empty".

The bottom screenshot shows the RStudio interface with the Global Environment pane populated with the following variables:

Data	
1	List of 4

Values	
a	"some name"
A	"SOME NAME"

Functions	
myf	function (x)

The console shows the following code being executed:

```
> a <- "some name"
> l <- list("some", "sort", "of", "list")
> myf <- function(x){toupper(x)}
> myf(a)
[1] "SOME NAME"
> A <- myf(a)
> |
```

Why use {box}: Modular-ize

- put code in to 'modules' or sections
- can have each module for a specific situation or group of functions
- other coding languages used in data science have this capability as well
 - C++ with namespace
 - Python with modules/packages
 - JavaScript with node modules

Quick Example: Different Ways of Coding

- data set with both character and numeric variables
- two types of plots
- want to create 2 of each type of plot using different variables

Note: this is a simplified version of the example in the EX_LONG folder of the repository, using mpg data from the {ggplot2}

A: Frequency Bar Plot

Character or Discrete Numeric Variables

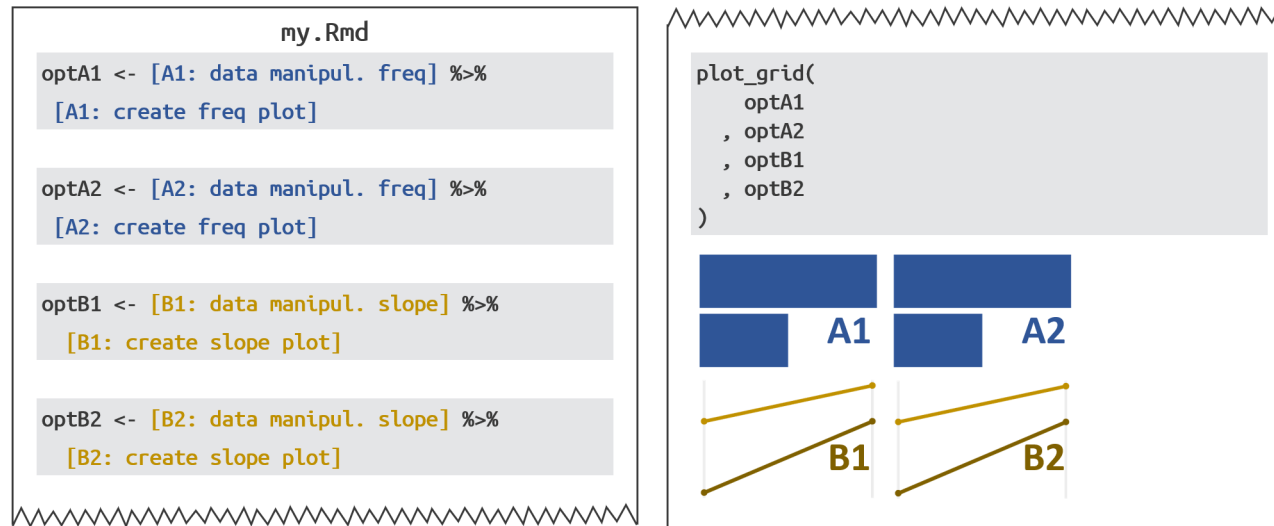


B: Slope Plot

Mean of Continuous Numeric Variable at Two Points



Option 1: Write Out Code

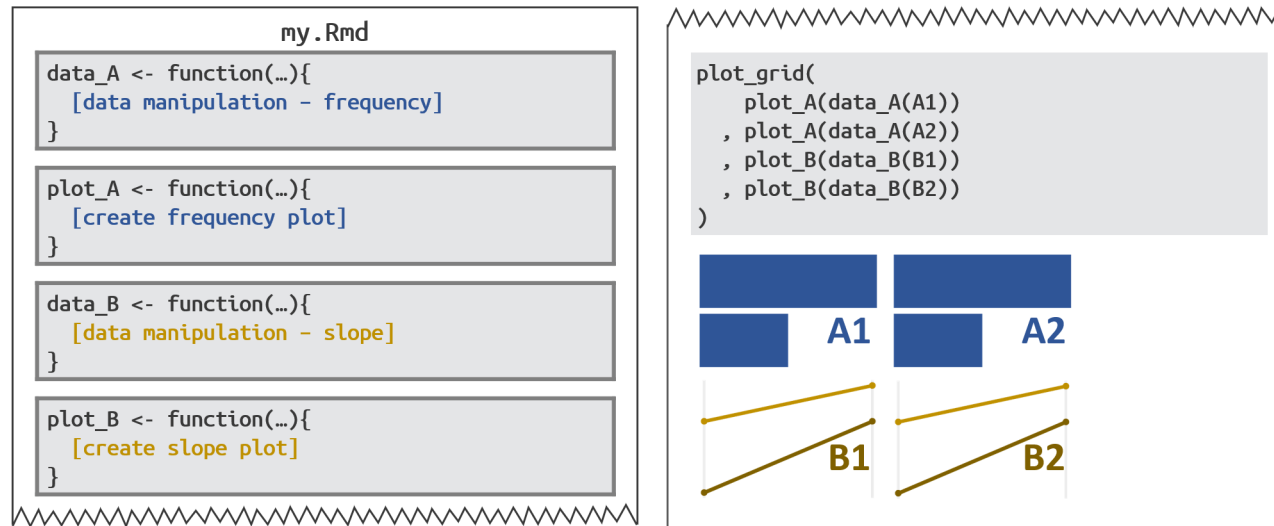


Create plots with different variables \Rightarrow more code (write or copy/paste)

More situations/dimensions \Rightarrow additional unstructured code:

- more mess and less organization
- more name space usage
- bad practice to continue to repeat code

Option 2: Multiple Functions

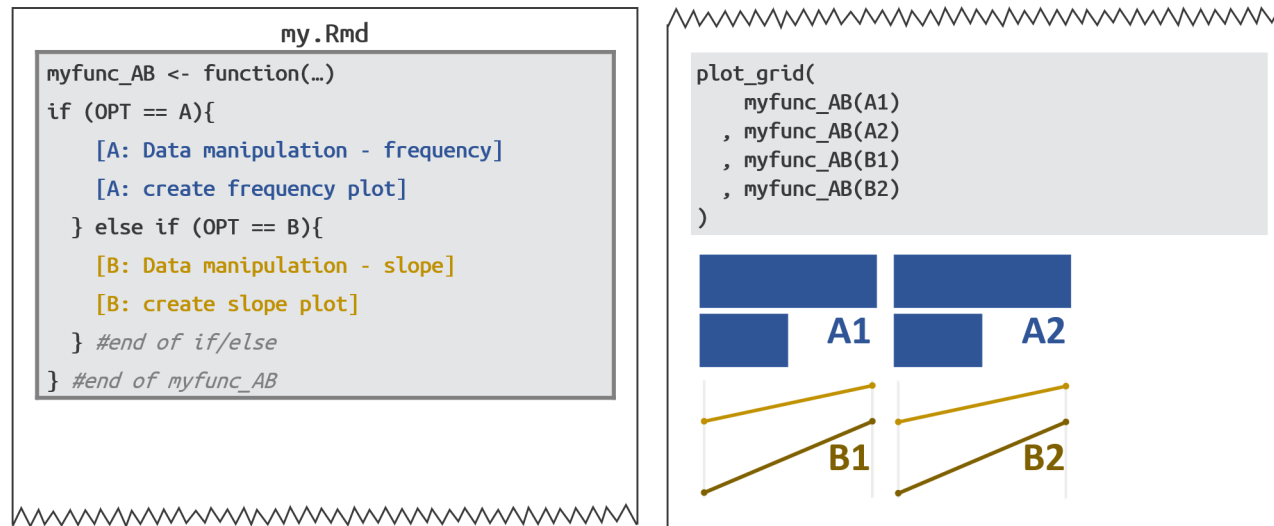


Create plots with different variables \Rightarrow can easily use functions

More situations/dimensions \Rightarrow more functions:

- more name space usage
- difficult to maintain and keep consistency (i.e. formatting across plots)

Option 3: One Long Function



Create plots with different variables \Rightarrow can easily use function

More situations/dimensions \Rightarrow Longer function

- difficult to maintain because of large size
- difficult to test (one part of the function breaks, entire function won't work)

Option 4: ✨ Box Modules ✨

```
Abox.R

#' Data Manipulation for A
createdata <- function(...){
  [data manipulation - frequency]
}

#' Plot Function for A
#' @export
createplot <- function(...)
  [create frequency plot]
}
```

```
Bbox.R

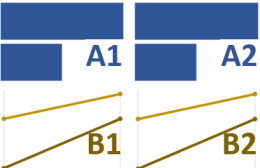
#' Data Manipulation for B
createdata <- function(...){
  [data manipulation - slope]
}

#' Plot Function for B
#' @export
createplot <- function(...)
  [create slope plot]
}
```

```
my.Rmd

box::use(
  myboxes/Abox #no ext.
  myboxes/Bbox )

plot_grid(
  Abox$createplot(A1)
  , Abox$createplot(A2)
  , Bbox$createplot(B1)
  , Bbox$createplot(B2)
)
```



















- Only box modules are in the name space
- Can use any **exported** functions
- Refer to other box modules within a specific modules

Create plots with different variables ⇒ use functions from modules

More situations/dimensions ⇒ add/amend box modules

- Test within a specific box module
- Break up long functions into multiple functions but only export the final function
- Much smaller usage of name space
- Easier to share with internally/externally

Comparison

Item	Option 1 Write Out Code	Option 2 Multiple Functions	Option 3 One Long Function	Option 4 Box Modules
Namespace Usage	 Constantly have to create unique names for objects	 Only use name space for specific functions and/or dependencies	 Only use name space for specific function	 Only use name space for box modules
Testing Code	 Testing code as written but shouldn't repeat code multiple times	 Functions are sectioned so can test at each level	 Difficult to test individual sections once function get extremely long	 All testing can be done within the module
Share/Re-Use	 Have to piece through which code to keep and then amend to new program	 Put functions together and document dependencies	 One function but may be challenging to amend to a different situation or different machine	 Just send the box modules .R files
Additional Dimensions	 New code either added to existing code or new section, bad practice if will use multiple times	 Manageable to difficult depending on how much each of the functions and their dependencies need to change	 Difficult given that the function is so large, adding a new dimension would be challenging	 Adjust easily within the box module or create a new box module (depending on situation)

Let's Learn how to use {box}

Box: File/Folder Structure

- Box modules are created using .R files
- Keep all modules in specific folder
- Folder can be named anything (often see 'box' or 'modules')
- How you call box modules (pathways) will depend on the structure of folders
- similar to including images or sourcing another file

Examples of folder structures:

```
EX_LONG
|
|---myboxes #folder with modules
|   |
|   |---Abox.R
|   |---admin.R
|   |---Bbox.R
|
|---my.Rmd #rmd file using box modules
```

```
EX_SHORT
|
|---box #folder with module
|   |
|   |---greet.R
|
|---my.R #r file using box module
```

Box: External Packages/Modules

- when starting your box module will need to use package (e.g. {ggplot2}, {dplyr}, etc.)
- may also want to use other box modules as well
- Usually load packages using `library()` command
- Need to load packages and functions with `box::use()`, NOT `library()`

`box::use` is a **universal import declaration**. It works for packages just as well as for modules. In fact, 'box' completely replaces the base R `library` and `require` functions. `box::use` is more explicit, more flexible, and less error-prone than `library`. At its simplest, it provides a direct replacement

- klmr.me/box/#loading-code

Box: `box::use()`

Use Box Module

```
box::use(myboxes/Abox)
```

- `myboxes`: folder where box module file is located
- `Abox`: name of box module to load (*without* the `.R` extension)

Use External Packages

```
box::use(dplyr[...])
```

- Imports `{dplyr}` and attached all exported names
- Similar to `library(dplyr)`
- Use any `{dplyr}` functions without specifying the package name
- If just use `box::use(dplyr)`, would not be able to use functions without calling package (i.e. `dplyr::filter`)

```
box::use(dplyr[filter, select])
```

- Imports `{dplyr}` and attaches the names `dplyr::filter` and `dplyr::select`
- Use `filter()` and `select()` from `{dplyr}`
- Would **not** be able to use `mutate()`
- Could use `dplyr::mutate()`

Box: Write Your Functions/Code with Roxygen

- Can use anything within box modules within functions
- Only need to export the final functions you want to use outside the module
- Use **Roxygen**
 - Roxygen is a way to document code, it's used in the development of R packages
 - `#'`: before each line with information on item
 - `@param`: specifies variable inputs (each input has own line)
 - `@return`: specify output (if function)
 - `@export`: export item (REQUIRED if using outside of module)
 - `@examples`: examples of use
 - can include as many or as few as you want
 - best practice to include all but I often don't, c'est la vie
 - best practices are a *journey* not a destination

Roxygen Skeleton

- Windows/Linux: Ctrl+Shift+Alt+R
- Mac: Option+Shift+Command+R
- Need to have cursor within a function

```
#' Title
#'  
#\' @param [param]  
#'  
#\' @return  
#\' @export  
#'  
#\' @examples
```

Box: Use Module Functions (EX_SHORT)

Write Code

File: box/greet.R

```
box::use(  
  glue[glue]  
  , stringr[str_to_title]  
)  
  
#' not exported, used WITHIN only  
greeting <- "Hello"  
  
#' Greet Someone when Given a Name  
#' @param name A character string  
#' @return Greeting with input name  
#' @export  
#' @examples  
#' say_hello("Martha")  
say_hello <- function(name){  
  to_use <- str_to_title(name)  
  glue("{greeting}, {to_use}")  
}
```

Import Module

File: my.R
Pathway to module file,
do not include .R extension

```
box::use(  
  box/greet  
)
```

Use Module!

File: my.R
Use \$ to use exported functions
(similar to :: for packages)

```
greet$say_hello("Martha")
```

```
## Hello, Martha
```

Box: Tips

Use Module after Changes

If you make changes to box module, need to **reload** module in order to changes to be in effect

```
box::reload(Abox) #just include name, not full pathway/folder
```

Restart R Session (Shift+Ctrl+F10) and then re-run all code (including `box::use()`)

Pathways

Use a box module **within** the same folder

```
#within Abox.R file  
box::use( ./admin)
```

```
EX_LONG  
|  
|---myboxes #folder with modules  
| |  
| |---Abox.R  
| |---admin.R  
| |---Bbox.R  
|  
|---my.Rmd #rmd file using box modules
```

Use a box module **up a directory**

```
#within reports/my.Rmd file  
box::use( ../myboxes/admin)
```

```
EX_LONG_REPORT_FOLDER  
|  
|---myboxes #folder with modules  
| |  
| |---Abox.R  
| |---admin.R  
| |---Bbox.R  
|  
|---reports #rmd file using box modules  
|  
|---my.Rmd
```