Data Wrangling in R - Data Wrangling - 2

One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

Objective	Complete
Demonstrate installing a package and loading a library	
Define the six functions that provide verbs for the language of data manipulation, from the package dplyr	

Packages and datasets in R

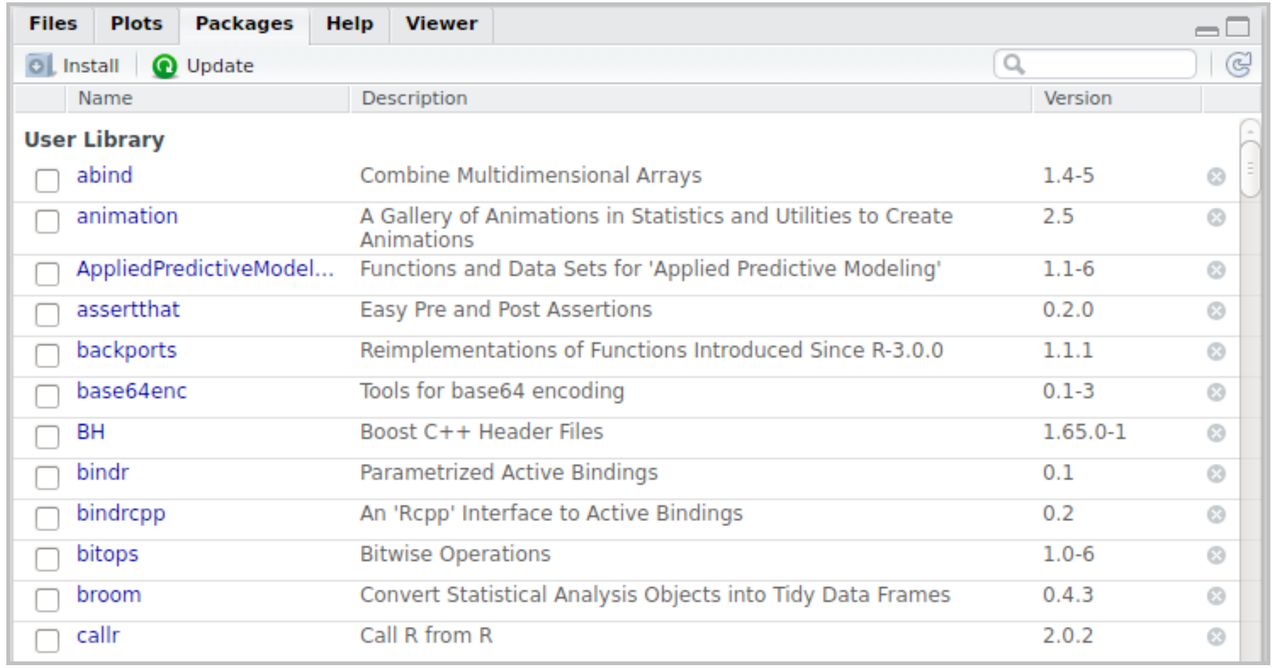
- We've now spent some time wrangling the CMP dataset by:
  - creating a subset of the data
  - identifying missing values (NAs)
  - imputing a new value (the mean) to replace missing values
- To continue practicing data wrangling, we are going install a series of packages intended to help with data wrangling tasks, known as the `tidyverse`
- We will also prepare to use some datasets available directly within R as packages

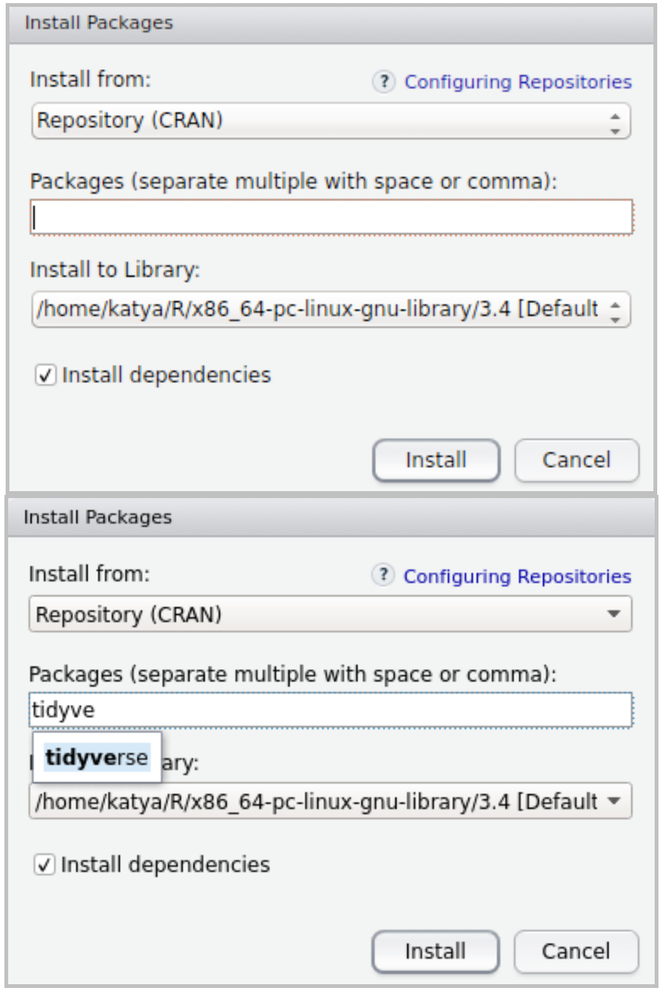
Packages in RStudio

- R packages are an extension of the R programming language
- They are collections of compiled code, sample data, and documentation
- They are typically installed from the CRAN (the Comprehensive R Archive Network)
- Learn more about 'base' and 'add-on' packages in R ***here***
- The package we'll be using for the remainder of the course is called the `tidyverse`

Installing packages: package explorer

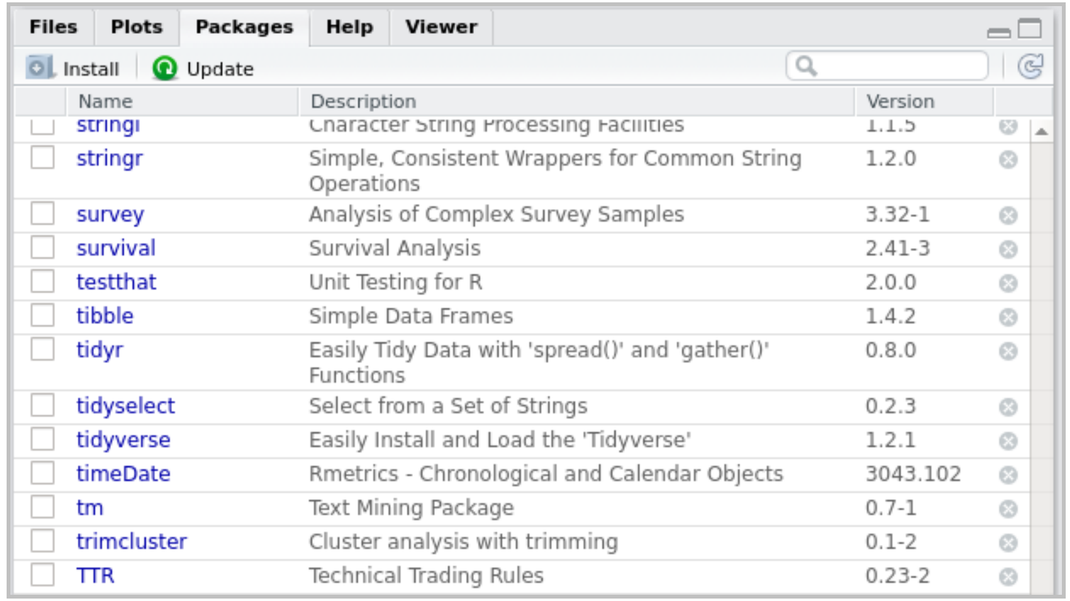
- RStudio has a built-in package manager in the bottom right pane to help us install packages
- Click on **`Packages`** tab in the bottom-right pane
- Click **`Install`** button next to `Update`
- Type package name in the box and install



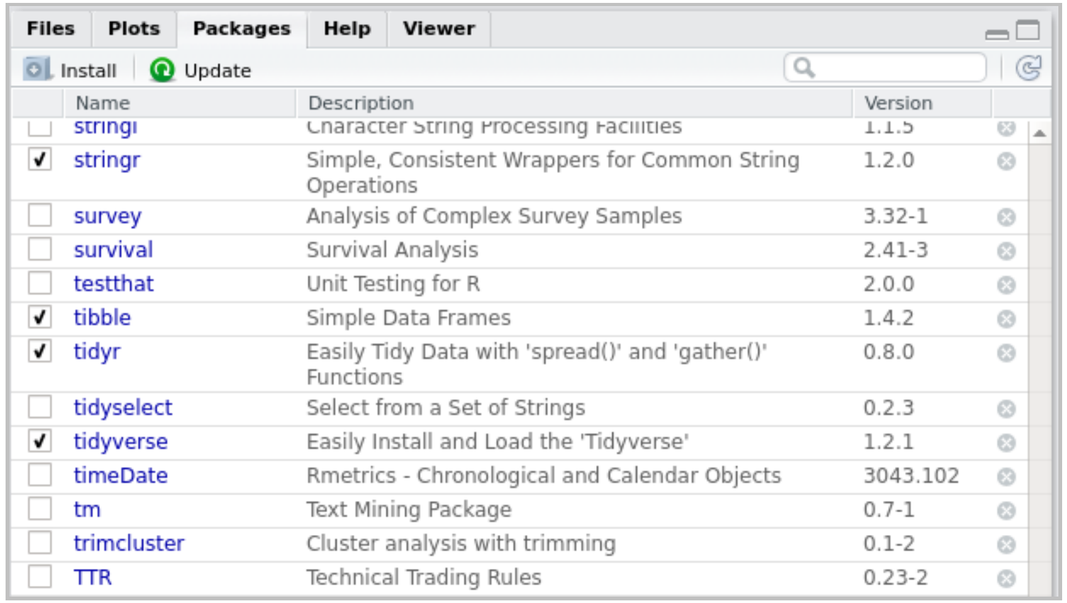


Installing packages: package explorer

- The installed package should appear in the list of packages in the package explorer



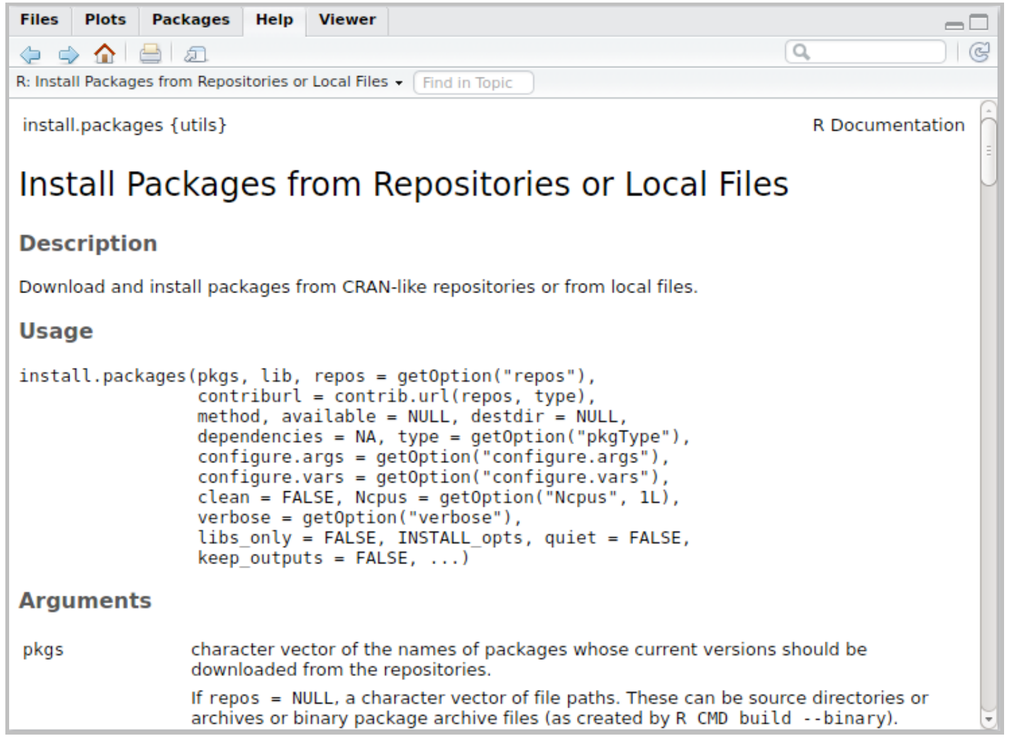
- To load the package into R's environment, check the box next to the name of your desired package



Installing packages

- If the function we would like to use comes from a `package`, we need to install the `package` first
- In addition to installing `packages` with package explorer as we introduced earlier, we can also use the function `install.packages()`
- For this function, we need to provide a single required argument: a character string corresponding to the `package name`

```
# Install package
?install.packages
```



Installing packages example

- Here is an example of how we install packages with function **`install.packages()`**
- You can always check the detailed documentation of a package with `help = "package name"`

```
install.packages("tidyverse")    #<- Install package
```

Documentation for package 'tidyv...

```
                Information on package 'tidyverse'

Description:

Package:            tidyverse
Title:              Easily Install and Load the 'Tidyverse'
Version:            1.2.1
Authors@R:          c( person("Hadley", "Wickham", , "hadley@rstudio.com", role =
                    c("aut", "cre")), person("RStudio", role = c("cph", "fnd")) )
Description:        The 'tidyverse' is a set of packages that work in harmony because
                    they share common data representations and 'API' design. This package
                    is designed to make it easy to install and load multiple 'tidyverse'
                    packages in a single step. Learn more about the 'tidyverse' at
                    <https://tidyverse.org>.
License:            GPL-3 | file LICENSE
URL:                http://tidyverse.tidyverse.org,
                    https://github.com/tidyverse/tidyverse
```

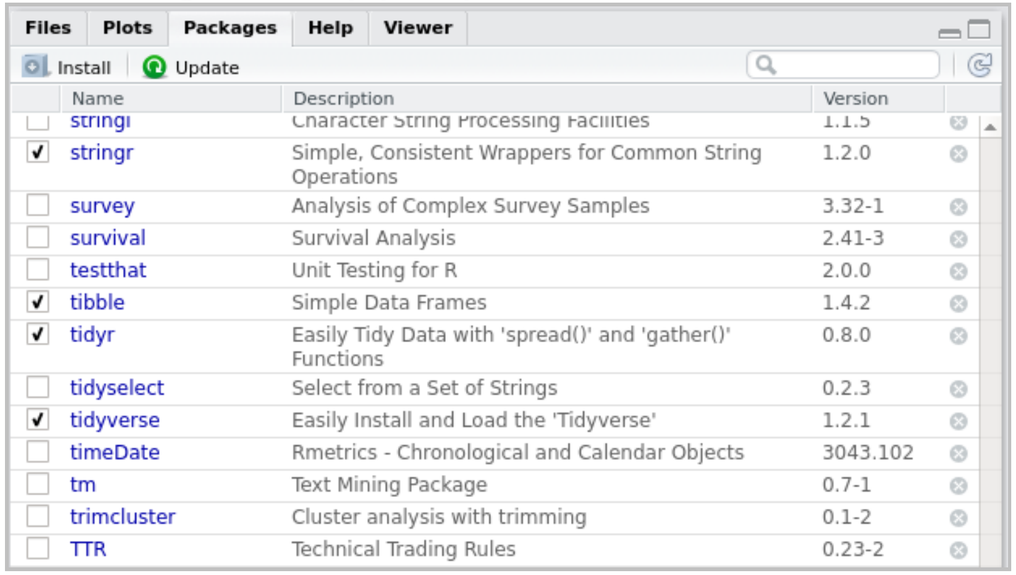
Loading packages in RStudio

- Once installed, it is **important** that your desired package is properly loaded in R.
- You can do this two different ways:

- The first option uses the `library()` command

```
library(tidyverse)
library(help = "tidyverse")      #<- View package
documentation.
```

- The second option involves locating the package name in the **Packages** pane and checking the appropriate box



Detaching packages in RStudio

- Some packages are very large and consume quite a bit of available memory when loaded
- You may want to use `detach()` after usage to free up your computer's memory

```
detach(package:tidyverse,unload=TRUE)
```

- You can also uncheck the box by the package's name on the **Packages** pane
- **Note: once detached you can not use any of the packages' function until reloaded**

Built-in R Datasets

- R comes with several built-in data packages, so that you always have sample data available for exploring how new packages and functions work
  - Titanic: Survival of passengers on the Titanic
  - iris: Edgar Anderson's Iris Data
  - mtcars: Motor Trend Car Road Tests

Loading Built-in Datasets

- Loading built-in datasets takes the form of installing a package rather than reading in an external file
- Let's now install and load the `nycflights13` package

```
install.packages("nycflights13", repos='http://cran.us.r-project.org')
```

```
Updating HTML index of packages in '.Library'
```

```
Making 'packages.html' ... done
```

```
library(nycflights13)
```

- The `nycflights13` package contains the following five datasets:
  - `flights`: all flights that departed from NYC in 2013
  - `weather`: hourly meteorological data for each airport
  - `planes`: construction information about each plane
  - `airports`: airport names and locations
  - `airlines`: translation between two letter carrier codes and names

Data transformation with tidyverse

- When you are given messy data, your goal is to transform it into a usable format
- To do this, you may need the help from multiple **packages** that can be found within the universe of ***`tidyverse`***
- Some core packages in `tidyverse` are:
  - `ggplot2`
  - `dplyr`
  - `tidyr`
- In this module, we will introduce few functions in `dplyr` that can be used to manipulate data

A little more about tidyverse

- Packages in the tidyverse change fairly frequently
- You can see if updates are available, and optionally install them, by running the following code

```
tidyverse_update()
```

- Like we noted previously, there are many libraries within the `tidyverse` package

- **The packages we will focus on help you wrangle and manipulate data quickly and efficiently**

Data transformation

- **`dplyr`** is an essential library within the tidyverse universe
- It will be the tool we use for transforming our data by filtering, aggregating, and summarizing
- Before starting this lesson, understand that dplyr does **overwrite** some `base R` packages such as `filter` and `lag`
- Even functions with exactly the same name can be of different usage and syntax when belonging to different packages
- If you have loaded dplyr and want to use the base version of the package, you will have to type in the full name:
  - for example `stats::filter` and `stats::lag`

Framework of dplyr

- The framework of `dplyr` is as follows:
  - The first argument is the **original dataframe**
  - The next arguments describe **what to do with the original dataframe**, using the six key `dplyr` functions
  - The final result is a **new, transformed dataframe**

Basics of dplyr

- Here are the six key `dplyr` functions we will discuss for the remainder of the course:

Function	Use Case	Data Type
`filter`	Pick observations by their value	All data types
`arrange`	Reorder the rows	All data types
`select`	Pick variables by their names	All data types
`mutate`	Create new variables with functions of existing variables	All data types
`summarize`	Collapse many values down to a single summary	All data types
`group_by`	Allows the above functions to operate on a dataset group by group	All data types

Knowledge check

Module completion checklist

Objective	Complete
Demonstrate installing a package and loading a library	✔
Define the six functions that provide verbs for the language of data manipulation, from the package dplyr	✔

Congratulations on completing this module!