# R and Tidyverse

SYS 6018 | Spring 2022

Rintro.pdf

## Contents

# 1 Technical Requirements

- Working and updated version of R and RStudio
  - Update packages as well
- Install R packages: `tidyverse` and `nycflights13`
- Course Webpage: https://mdporter.github.io/SYS6018

# 2 Introduction to R

## 2.1 Getting Help

- A good source of basic data analysis using R is found in the free book R for Data Science.
- Web search, especially *stackoverflow.com* and *stats.stackexchange.com*
- Troubleshooting/Debugging.
  - Check one line of code at a time.
  - Use scripts
  - Make sure it works in plain R before incorporating into Rmd

## 2.2 RStudio

- Install R and RStudio
- Make use of *Projects* in RStudio

## 2.3 Using R Packages

It takes two steps to use the functions and data in an R package

1. Install the package
   - i.e., download the package to your computer
   - this only needs to be done one time
   - `install.packages()`
2. Load the package
   - i.e., tell R to look for the package functions and/or data
   - this needs to be done every time R is started (and you want to use the package)
   - `library()`

### 2.3.1 Note on `tidyverse` package

- The `tidyverse` package https://www.tidyverse.org/packages/ is really just a wrapper to load several related R packages
  - `ggplot2` for graphics
  - `dplyr` for data manipulation
  - `tidyr` for getting data into tidy form
  - `readr` for loading in data
  - `tibble` for improved data frames
  - `purrr` for functional programming

  - `stringr` for string manipulation
  - `forcats` for categorical/factor data
- This provides a nice shortcut to load all of these packages with `library(tidyverse)` instead of each separately:

```
#- the hard way
library(ggplot2)
library(dplyr)
library(tidyr)
library(readr)
library(tibble)
library(purrr)
library(stringr)
library(forcats)
```

```
#- the easy way
library(tidyverse)
```

## 2.4 RMarkdown

- Homework will be submitted in Rmd and (html) format
- When you *knit* a Rmd, it:
  1. starts a new instance of R (clean environment)
  2. in the current directory
- Any data or code must first be put into the Rmd file
  - The Rmd won't know about anything in another script or in your R environment
  - Any `source()` or data paths are relative to the current directory of the Rmd
- A homework template will be provided for each homework
  - This will automatically apply a custom format if you have the `R6018` package installed

## 2.5 Graphics with the `ggplot2` package

The `ggplot2` package is an approach to creating graphics for data analysis.

- See https://ggplot2.tidyverse.org/
- Keep the ggplot2 cheatsheet handy

## 2.6 Data Transformation with the `dplyr` package

- See https://dplyr.tidyverse.org/
- Keep the dplyr cheatsheet handy

### 2.6.1 single table verbs

1. `filter()`: find/keep certain rows
   - alternative to `base::subset()`
   - `slice()` to keep by row number

    - helper functions: `between()`: numeric values in a range
2. `arrange()`: reorder rows
    - alternative to `base::order()`
    - helper functions: `desc()` to use descending order
3. `select()`: find/keep certain columns
    - helper functions: `starts_with()`, `ends_with()`, `matches()`, `contains()`, `?select`
4. `mutate()`: add/create new variables
    - alternative to `base::transform()`
    - `transmute()`: only return new variables
5. `summarize()`: produce summary statistics
    - don't confuse with `summary()`
    - most useful when data is *grouped*

### 2.6.2 Chaining/Pipes

- Multiple operations can be chained together with the *pipe* operator, `%>%`, (pronounced as *then*). Technically, it performs `x %>% f(y) -> f(x, y)`. This lets you focus on the verbs, or actions you are performing.

```
x = c(1:5, NA)
x %>% mean(na.rm=TRUE)
#> [1] 3
mean(x, na.rm=TRUE)
#> [1] 3
```

**Your Turn #1**

1. Load the `nycflights13` package, which contains airline on-time data for all flights departing NYC in 2013. Also includes useful 'metadata' on airlines, airports, weather, and planes.
2. Load the `tidyverse` package
3. Using the `flights` data,
    - find all flights that were less than 1000 miles (`distance`)
    - Keep only the columns: `dep_delay`, `arr_delay`, `origin`, `dest`, `air_time`, and `distance`
    - Add the $Z$-score for departure delays
    - Convert the departure and arrival delays into hours
    - Calculate the average flight speed (in mph)
    - order by average flight speed (fastest to slowest)
    - return the first 12 rows

### 2.6.3 Other useful `dplyr` functions

- `distinct()`: retain unique/distinct rows
- `slice_sample()`: select random rows
- `slice_min/slice_max()`: select rows with smallest/highest values
- `mutate()/add_column()` add new column in particular position
- `coalesce(x, y)` replaces the `NA` in `x` with `y`

```
x = c(1, 2, NA, 5, 5, NA)
coalesce(x, 0)          # replace NA with 0
#> [1] 1 2 0 5 5 0
```

## 2.7 Groupwise operations

### 2.7.1 Split - Apply - Combine

The dplyr operations are more powerful when they can be used with grouping variables. Split - Apply - Combine.

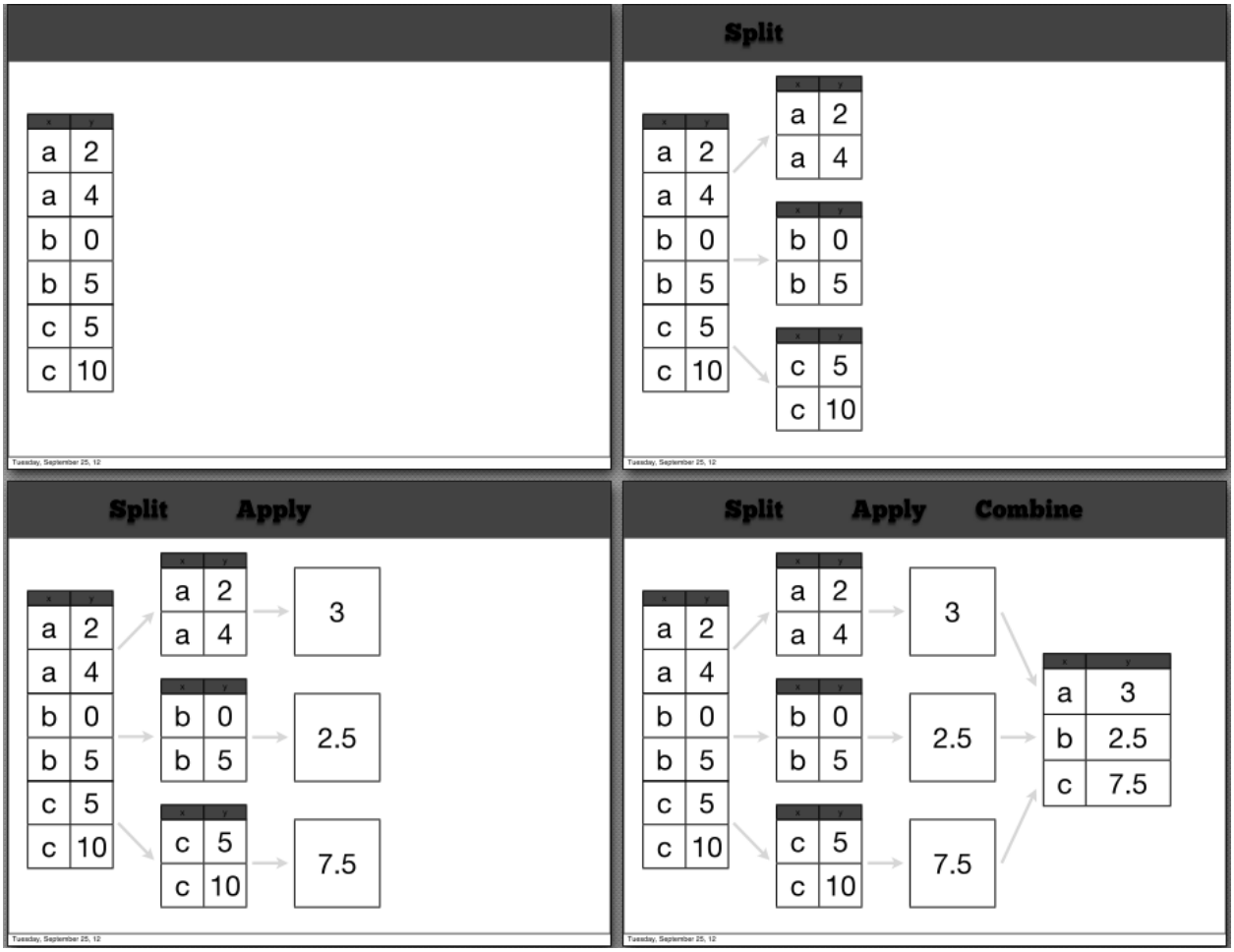



Image from Hadley Wickham UseR tutorial June 2014 http://www.dropbox.com/sh/i8qnluwmuieicxc/AAAgt9tIKoIm7WZKIyK25lh6a

### 2.7.2 `group_by()`

First use the `group_by()` function to group the data (determines how to split), then apply function(s) to each group using the `summarise()` function. Note: grouping should to be applied on discrete variables (categorical, factor, or maybe integer valued columns).

```
flights %>%
  group_by(origin, dest) %>%   # group by both origin and dest
  summarize(max.delay = max(arr_delay, na.rm=TRUE),
```

```
            avg.delay = mean(arr_delay, na.rm=TRUE),
            min.delay = min(arr_delay, na.rm=TRUE),
            count = n() )        # n() gives the group count
#> Warning in max(arr_delay, na.rm = TRUE): no non-missing arguments to max;
#> returning -Inf
#> Warning in min(arr_delay, na.rm = TRUE): no non-missing arguments to min;
#> returning Inf
#> # A tibble: 224 x 6
#>   origin dest  max.delay avg.delay min.delay count
#>   <chr>  <chr>     <dbl>     <dbl>     <dbl> <int>
#> 1 EWR    ALB         328    14.4         -34   439
#> 2 EWR    ANC          39    -2.5         -47     8
#> 3 EWR    ATL         796    13.2         -39  5022
#> 4 EWR    AUS         349    -0.474       -59   968
#> 5 EWR    AVL         228     8.80        -26   265
#> 6 EWR    BDL         266     7.05        -43   443
#> # ... with 218 more rows
```

- `count(...)` is a shortcut for `group_by(...) %>% summarize(n=n())`
- `ungroup()` removes the grouping

### 2.7.3 Grouped Mutate and Filter

- When data is *grouped*, `mutate()` and `filter()` operate on each group independently

```
#- proportion of carrier at each dest
flights %>%
  count(dest, carrier) %>%
  group_by(dest) %>%                    # group by dest
  mutate(total=sum(n), p=n/sum(n)) %>% # grouped mutate sum(n) is by group
  arrange(desc(total), -p)              # arrange by most freq dest and prop
#> # A tibble: 314 x 5
#>   dest  carrier     n total        p
#>   <chr> <chr>   <int> <int>    <dbl>
#> 1 ORD   UA       6984 17283 0.404
#> 2 ORD   AA       6059 17283 0.351
#> 3 ORD   MQ       2276 17283 0.132
#> 4 ORD   9E       1056 17283 0.0611
#> 5 ORD   B6        905 17283 0.0524
#> 6 ORD   EV          2 17283 0.000116
#> # ... with 308 more rows
```

## 2.8 Data Importing

### 2.8.1 `readr` package

- See https://readr.tidyverse.org/
- Keep the data import cheatsheet handy

```
## Load data from course website
library(tidyverse)

#: specify path to url
data.dir = 'https://mdporter.github.io/SYS6018/data/'
url = file.path(data.dir, 'crashes16.csv') # the crashes 16 data set
```

```
#: load directly from web
crashes = read_csv(url)
crashes
#> # A tibble: 456 x 2
#>    mile   time
#>   <dbl>  <dbl>
#> 1   87  6.62
#> 2  118  6.70
#> 3  120  0.0549
#> 4   90  0.206
#> 5  124. 0.726
#> 6  118  3.88
#> # ... with 450 more rows
```

```
## Download data first, then load into R

#: specify path to url
data.dir = 'https://mdporter.github.io/SYS6018/data/'
url = file.path(data.dir, 'crashes16.csv') # the crashes 16 data set

#: download file
save.path = "data/crashes16.csv"  # can be relative path!
download.file(url, save.path)

#: load data from hard drive
library(tidyverse)
crashes = read_csv(save.path)
```

### 2.8.2 `readxl` package

- See https://readxl.tidyverse.org/ for importing excel files

## 2.9 Tidy Data with the `tidyr` package

- https://tidyr.tidyverse.org/
- Keep the tidy data cheatsheet handy.

### 2.9.1 Why Tidy Data?

- Tidy data (in form of a data frame) is usually the best form for analysis
  - some exceptions are for modeling (e.g., matrix manipulations and algorithms)
- For presentation of data (e.g., in tables), non-tidy form can often do better
- the functions in `tidyr` usually allow us to covert from non-tidy to tidy for analysis and also from tidy to non-tidy for presentation

### 2.9.2 Main `tidyr` functions

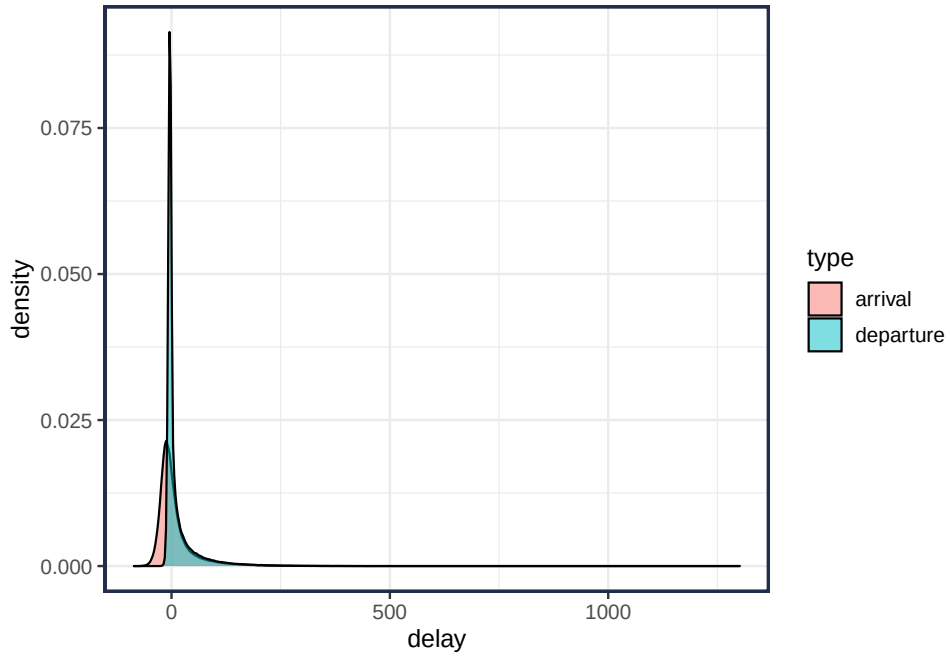
| function | description |
|---|---|
| `pivot_wider()/spread()` | Spreads a pair of key:value columns into a set of tidy columns |
| `pivot_longer()/gather()` | Gather takes multiple columns and collapses into key-value pairs, duplicating all other columns as needed. You use `pivot_longer()/gather()` when you notice that you have columns that are not variables |

| function | description |
|---|---|
| `separate()` | turns a single character column into multiple columns |
| `unite()` | paste together multiple columns into one (reverse of `separate()`) |

```
## Converting to longer format for grouped summaries and plotting
delays_long = flights %>%
  select(year, month, day, dep_delay, arr_delay) %>%
  pivot_longer(cols = c(dep_delay, arr_delay), names_to="type", values_to="delay") %>%
  mutate(type = ifelse(type == "dep_delay", "departure", "arrival"))
```

```
#: average delays of each type
delays_long %>%
  group_by(type) %>%
  summarize(avg.delay = mean(delay, na.rm=TRUE))
#> # A tibble: 2 x 2
#>   type      avg.delay
#>   <chr>         <dbl>
#> 1 arrival        6.90
#> 2 departure     12.6
```

```
#: plot histogram
delays_long %>%
  ggplot(aes(delay, fill=type)) +
  geom_density(alpha = .5)
```




## 2.10 Loops

We will make good use of iteration in this course. Suppose we want to compare the performance of two models over multiple subsets of a data set.

```
#: simulate fake data
set.seed(2022)
data = tibble(x = runif(100), y = rnorm(100))
```

```r
#: fake model output
model_1 <- function(x) rnorm(length(x), mean = -.5, sd = 1)
model_2 <- function(x) rnorm(length(x), mean = .5, sd = 2)
```

We can consider using a for loop

```r
n_subsets = 5

set.seed(876)
output = vector("list", n_subsets)  # initiate empty list
for(i in 1:n_subsets){

  #: sub-sample data (25 samples)
  data_sub = dplyr::slice_sample(data, n = 25)

  #: get output from the models
  yhat_1 = model_1(data_sub$x)
  yhat_2 = model_2(data_sub$x)

  #: score models (using MSE)
  perf_1 = mean( (yhat_1 - data_sub$y)^2 )
  perf_2 = mean( (yhat_2 - data_sub$y)^2 )

  #: save results
  output[[i]] = tibble(perf_1, perf_2, iter = i)
}

#: convert to tibble and summarize
bind_rows(output) %>%
  summarize(avg.diff = mean(perf_1 - perf_2), n=n())
#> # A tibble: 1 x 2
#>   avg.diff     n
#>      <dbl> <int>
#> 1    -1.51     5
```