Introduction to modern R

tidyverse, tidymodels, and quarto

DS 6030 | Fall 2024

Rintro.pdf

Contents

1 Technical Requirements

- Course Webpage: https://mdporter.github.io/DS6030
- Working and updated version of R and RStudio
  - Update packages as well
  - See course syllabus for details
- Install R packages: `tidyverse` and `nycflights13`

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 2e.
- 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
- ChatGPT can do a decent job at troubleshooting.
- Post a question on Teams. A classmate or teaching staff will be able to help.

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
  - `lubridate` for dealing with dates and times
- This provides a nice shortcut to load all of these packages with `library(tidyverse)` instead of each separately:

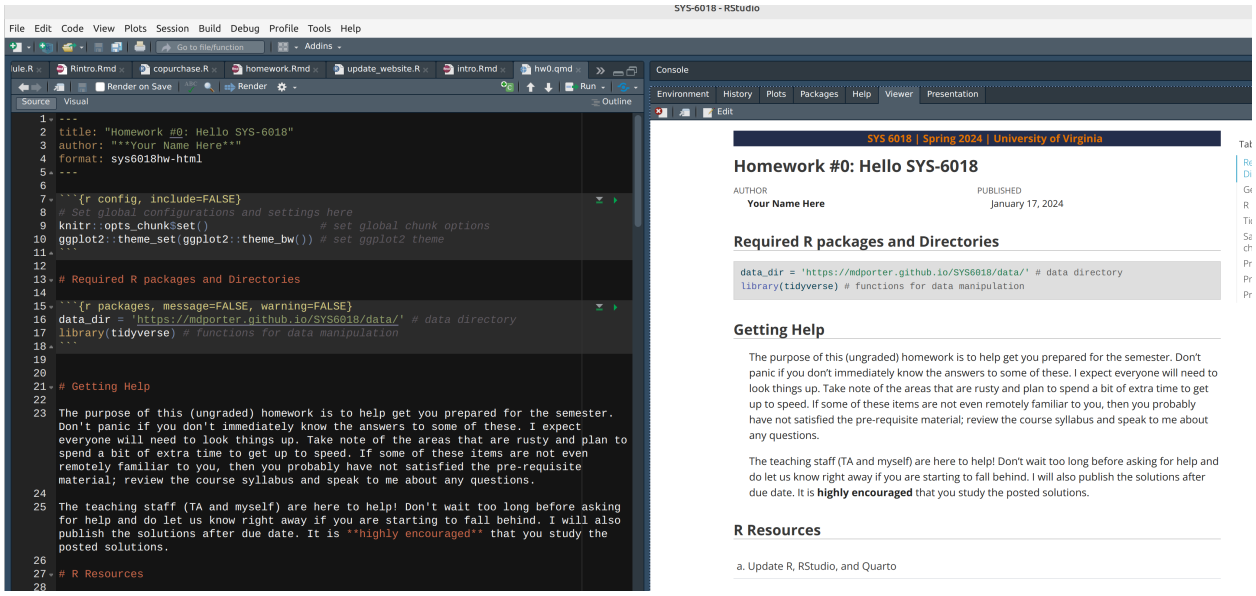
```r
#- the hard way
library(ggplot2)
library(dplyr)
library(tidyr)
library(readr)
library(tibble)
library(purrr)
library(stringr)
library(forcats)
library(lubridate)
```

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

2.4 Quarto

Quarto is the new RMarkdown.

- Homework will be submitted in .qmd and .html format
- When you *render* a .qmd, it:
  1. starts a new R kernel (clean environment)
  2. in the directory the .qmd is in (important for relative links)
  3. runs all the code blocks
  4. converts the results of the code and other text to plain markdown
  5. uses pandoc to create documents of the desired format (e.g., html, pdf, word, revealjs, beamer)
- Any data or code must first be put into the .qmd file
  - The .qmd won't know about anything in another script or in another R environment
  - Any `source()` or data paths are relative to the current directory of the qmd.
- A homework .qmd template will be provided for each homework. You will modify this template to create your homework solution.
- The course uses a special formatting to ensure all submissions look consistent and are thus easier to grade. The formatting is applied with a special quarto extension. Follow the instructions to install the extension.
- More quarto resources:
  - https://quarto.org/
  - quarto cheatsheet



~~~
---
title: "Homework #0: Hello SYS-6018"
author: "**Your Name Here**"
format: sys6018hw-html
---

```{r config, include=FALSE}
# Set global configurations and settings here
knitr::opts_chunk$set()                    # set global chunk options
ggplot2::theme_set(ggplot2::theme_bw()) # set ggplot2 theme
```

# Required R packages and Directories

```{r packages, message=FALSE, warning=FALSE}
data_dir = 'https://mdporter.github.io/SYS6018/data/' # data directory
library(tidyverse) # functions for data manipulation
```


# Getting Help

The purpose of this (ungraded) homework is to help get you prepared for the semester.
Don't panic if you don't immediately know the answers to some of these. I expect
everyone will need to look things up. Take note of the areas that are rusty and plan to
spend a bit of extra time to get up to speed. If some of these items are not even
remotely familiar to you, then you probably have not satisfied the pre-requisite
material; review the course syllabus and speak to me about any questions.

The teaching staff (TA and myself) are here to help! Don't wait too long before asking
for help and do let us know right away if you are starting to fall behind. I will also
publish the solutions after due date. It is **highly encouraged** that you study the
posted solutions.

# R Resources

~~~

SYS 6018 | Spring 2024 | University of Virginia

Homework #0: Hello SYS-6018

AUTHOR
Your Name Here

PUBLISHED
January 17, 2024

Required R packages and Directories

```
data_dir = 'https://mdporter.github.io/SYS6018/data/' # data directory
library(tidyverse) # functions for data manipulation
```

Getting Help

The purpose of this (ungraded) homework is to help get you prepared for the semester. Don't panic if you don't immediately know the answers to some of these. I expect everyone will need to look things up. Take note of the areas that are rusty and plan to spend a bit of extra time to get up to speed. If some of these items are not even remotely familiar to you, then you probably have not satisfied the pre-requisite material; review the course syllabus and speak to me about any questions.

The teaching staff (TA and myself) are here to help! Don't wait too long before asking for help and do let us know right away if you are starting to fall behind. I will also publish the solutions after due date. It is **highly encouraged** that you study the posted solutions.

R Resources

a. Update R, RStudio, and Quarto

Figure 1: Quarto Screen Shot

2.5 Graphics with the `ggplot2` package

The `ggplot2` package is an approach for 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.

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

- **Update:** newer versions of R have introduced a native pipe `|>`. This can be used instead of `%>%`.

```r
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`

```r
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.

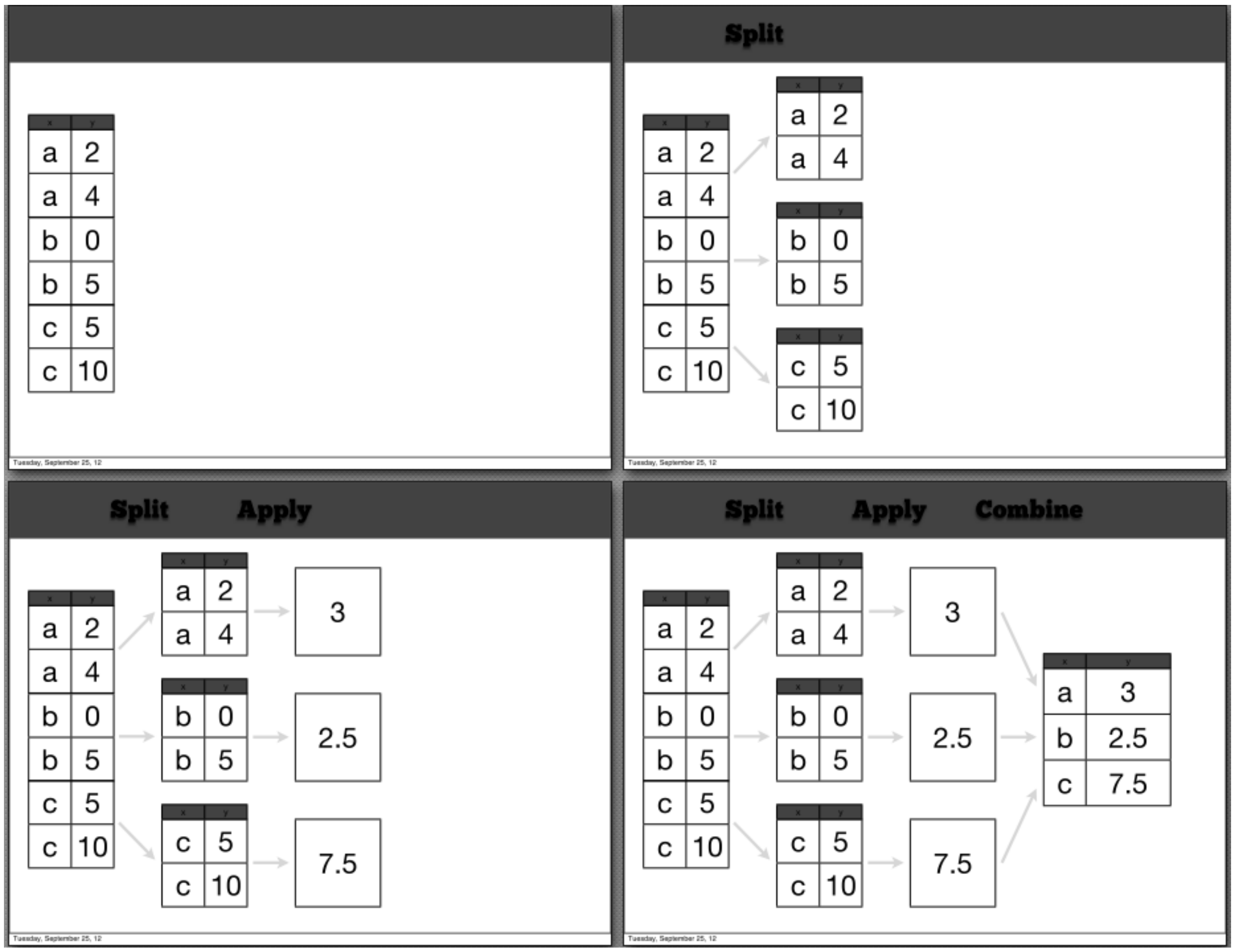


Image from Hadley Wickham UseR tutorial June 2014 http://www.dropbox.com/sh/i8qnluwmuieicxc/AAAgt9tIKoIm7WZKIyK25lh6a

These steps can be performed, at scale, with the MapReduce framework.

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).

```r
flights |>
  group_by(origin, dest) |>    # group by both origin and dest
  summarize(
    n_flights = n(), # n() gives the group count
    max_delay = max(arr_delay, na.rm=TRUE),
    avg_delay = mean(arr_delay, na.rm=TRUE),
    min_delay = min(arr_delay, na.rm=TRUE)
  )
#> Warning: There were 2 warnings in `summarize()`.
#> The first warning was:
#> i In argument: `max_delay = max(arr_delay, na.rm = TRUE)`.
#> i In group 41: `origin = "EWR"` and `dest = "LGA"`.
#> Caused by warning in `max()`:
#> ! no non-missing arguments to max; returning -Inf
#> i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
#> # A tibble: 224 x 6
#>   origin dest  n_flights max_delay avg_delay min_delay
#>   <chr>  <chr>     <int>     <dbl>     <dbl>     <dbl>
#> 1 EWR    ALB         439       328    14.4         -34
#> 2 EWR    ANC           8        39    -2.5         -47
#> 3 EWR    ATL        5022       796    13.2         -39
#> 4 EWR    AUS         968       349    -0.474       -59
#> 5 EWR    AVL         265       228     8.80        -26
#> 6 EWR    BDL         443       266     7.05        -43
#> # i 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),             # grouped mutate sum(n) is by group
      p = n/sum(n)
    ) |>
  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
#> # i 308 more rows
```

2.7.4 Update: grouping with `.by`

In newer versions of `dplyr`, many functions that were used with grouped data frames have been given an additional argument that allows grouping specification without the need to use `group_by()`.

```
#- proportion of carrier at each dest
flights |>
  count(dest, carrier) |>
    mutate(.by = dest,            # <-- specify grouping
      total = sum(n),             # grouped mutate sum(n) is by group
      p = n/sum(n)
    ) |>
  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
#> # i 308 more rows
```

```
flights |>
  summarize(.by = c(origin, dest),  # <-- specify grouping
    n_flights = n(), # n() gives the group count
    max_delay = max(arr_delay, na.rm=TRUE),
    avg_delay = mean(arr_delay, na.rm=TRUE),
    min_delay = min(arr_delay, na.rm=TRUE)
  )
#> # A tibble: 224 x 6
#>   origin dest  n_flights max_delay avg_delay min_delay
#>   <chr>  <chr>     <int>     <dbl>     <dbl>     <dbl>
#> 1 EWR    IAH        3973       374      5.41       -63
#> 2 LGA    IAH        2951       435      1.45       -59
#> 3 JFK    MIA        3314       614     -1.99       -64
#> 4 JFK    BQN         599       183      6.94       -32
#> 5 LGA    ATL       10263       895     11.3        -49
#> 6 EWR    ORD        6100      1109      9.00       -59
#> # i 218 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/teaching/data/' # path to course 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
#> # i 450 more rows
```

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

#: specify path to url
data_dir = 'https://mdporter.github.io/teaching/data/' # path to course 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
`separate()`	turns a single character column into multiple columns
`unite()`	paste together multiple columns into one (reverse of `separate()`)

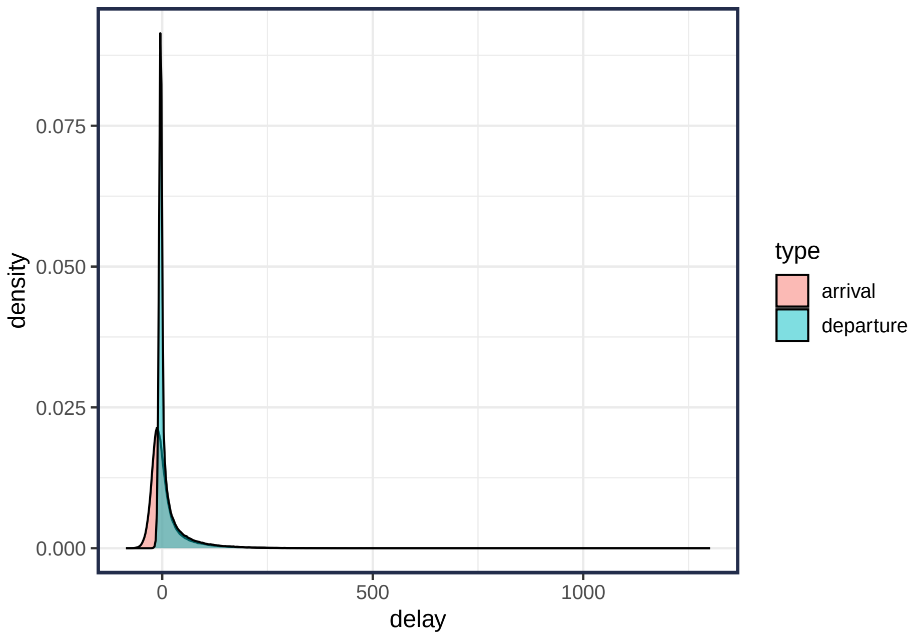
See R4DS (2e) Data Tidying for more details.

```
## 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"))
```

```r
delays_long
#> # A tibble: 673,552 x 5
#>    year month   day type       delay
#>   <int> <int> <int> <chr>      <dbl>
#> 1  2013     1     1 departure      2
#> 2  2013     1     1 arrival       11
#> 3  2013     1     1 departure      4
#> 4  2013     1     1 arrival       20
#> 5  2013     1     1 departure      2
#> 6  2013     1     1 arrival       33
#> # i 673,546 more rows
```

```r
#: 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
```

```r
#: plot density (kernel density estimation)
delays_long |>
  ggplot(aes(delay, fill=type)) +
  geom_density(alpha = .5)
```



2.10 Iteration

We will make good use of iteration in this course. Be sure to review R4DS (2e) Iteration for more details.

Suppose we want to compare the performance of two models using Monte Carlo cross-validations. Here we'll use the flights data to predict arrival delay (`arr_delay`).

```r
data = flights %>%
  select(arr_delay, dep_delay, sched_arr_time, carrier) %>%
  filter(!is.na(arr_delay))
```

We can consider using a for loop to assess predictive performance in each iteration.

```r
n_iters = 5

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

  #: train/test split (25 samples in test)
  ind_test  = sample(nrow(data), size = 25)
  data_test = data[ind_test, ]
  data_fit  = data[-ind_test, ]

  #: fit models
  lm_1 = lm(arr_delay ~ dep_delay, data = data_fit)
  lm_2 = lm(arr_delay ~ dep_delay + sched_arr_time, data = data_fit)

  #: get predictions from the models on test data
  yhat_1 = predict(lm_1, data_test)
  yhat_2 = predict(lm_2, data_test)

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

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

```r
#: convert to tibble and summarize
bind_rows(output) |>
  summarize(
    avg_perf_1 = mean(perf_1),          # avg MSE of model 1
    avg_perf_2 = mean(perf_2),          # avg MSE of model 2
    avg_diff = mean(perf_1 - perf_2),   # average MSE difference
    n = n()                             # number of iterations
  )
#> # A tibble: 1 x 4
#>   avg_perf_1 avg_perf_2 avg_diff     n
#>        <dbl>      <dbl>    <dbl> <int>
#> 1       266.       264.     1.95     5
```

Notice that every *for* loop requires three elements:

1. Initializing the **output** structure to store results.
   - In the above example, set the output to be a list with `n_iters` empty elements. The list is unnamed.
2. The sequence to iterate over.
   - In this example, indexes over 1 to `n_iters`. It is common to use `seq_len(n_iters)` or `seq_along(output)` instead of the explicit sequence.
3. The body of the loop.
   - In the body of the loop, do the stuff of interest.

2.10.1 Vectorization

R works best (i.e., fastest) when you use *vectorized* calculations. This means you should avoid loops whenever a vectorized function is available.

```r
x = 1:100

#: Find squared value
x_sq = x^2 # using vectorized power operator

x_sq_slow = numeric(length(x))
for(i in seq_along(x)) {
  x_sq_slow[i] = x[i]^2
}

#: replace all even values with -1
x_even = ifelse(x %% 2 == 0, -1, x)
```

Note: check out the `dplyr::case_when()` for more complex handling of multiple if-else statements.

2.10.2 Iterate over elements of a list or vector with `purrr::map()`

Sometimes there are no vectorized solutions and we have to loop. The `purrr` library provides a nice set of functions to help you make better loops. See R4DS The map functions

We saw above that every *for* loop requires three elements: 1. Initializing the output structure to store results. 2. The sequence to iterate over. 3. The body of the loop.

The Base R approach explicitly requires each step. Alternatively, the `purrr::map()` solution hides the output and sequence steps and let's you focus on the body. This has other advantages - cleaner and easier to understand code, and ability to more easily parallelize the operations (see https://furrr.futureverse.org/).

The first step is to make a function that does everything in the body; in this example it calculates the mean squared error (mse) of each predictive model:

```r
calculate_mse <- function(data){

  #: train/test split (25 samples in test)
  ind_test  = sample(nrow(data), size = 25)
  data_test = data[ind_test, ]
  data_fit  = data[-ind_test, ]

  #: fit models
  lm_1 = lm(arr_delay ~ dep_delay, data = data_fit)
  lm_2 = lm(arr_delay ~ dep_delay + sched_arr_time, data = data_fit)

  #: get predictions from the models on test data
  yhat_1 = predict(lm_1, data_test)
  yhat_2 = predict(lm_2, data_test)

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

  #: output results
  tibble(perf_1, perf_2)
}
```

Then we use the `map()` function to run the loop:

```
library(purrr)
set.seed(876)
map_df(1:n_iters, \(i) calculate_mse(data)) |>
  summarize(
    avg_perf_1 = mean(perf_1),          # avg MSE of model 1
    avg_perf_2 = mean(perf_2),          # avg MSE of model 2
    avg_diff = mean(perf_1 - perf_2),   # average MSE difference
    n = n()                             # number of iterations
  )
#> # A tibble: 1 x 4
#>   avg_perf_1 avg_perf_2 avg_diff     n
#>        <dbl>      <dbl>    <dbl> <int>
#> 1       266.       264.     1.95     5
```

3 Tidymodels Introduction

The `tidymodels` set of packages are an improvement of the `caret` package (and developed by the same authors).

- tidymodels website
- tidymodels book

3.1 Predictive Modeling

There are a few steps common to most predictive modeling pipelines:

1. Split data to create hold-out set(s).
2. Specify model configuration and tuning parameters.
3. Pre-process *training* data.
4. Fit model (i.e., estimate model parameters)
5. Pre-process hold-out/test data.
6. Predict values of hold-out/test data.
7. Evaluate predictions.

These, as well as more advanced steps, can be carried out with tidymodels. Here's an example of using ridge regression for predicting the arrival delay:

```
library(tidymodels)

# 1. Split the data
set.seed(456)
data_split = initial_split(data, prop = .99, strata = arr_delay)
training(data_split)
#> # A tibble: 324,071 x 4
#>   arr_delay dep_delay sched_arr_time carrier
#>       <dbl>     <dbl>          <int> <chr>
#> 1       -18        -1           1022 B6
#> 2       -25        -6            837 DL
#> 3       -17         0            915 UA
#> 4       -21         0           1100 B6
#> 5       -19        -1            740 WN
#> 6       -18        -3            940 UA
#> # i 324,065 more rows
testing(data_split)
```

```
#> # A tibble: 3,275 x 4
#>   arr_delay dep_delay sched_arr_time carrier
#>       <dbl>     <dbl>          <int> <chr>
#> 1        14        11            931 UA
#> 2        29        -2            947 UA
#> 3        14        -1            810 AA
#> 4       -12        -3           1035 AA
#> 5         6        -4           1220 AA
#> 6        -4        -2           1425 UA
#> # i 3,269 more rows
```

```
# 2. Model specification. Using lasso penalized linear regression.
# library(parsnip)
lasso = linear_reg(engine = "glmnet") %>%
  set_args(
    mixture = 1,       # 1 = lasso, 0 = ridge
    penalty = 1.1      # the strength of penalty (i.e., lambda)
  )
lasso
#> Linear Regression Model Specification (regression)
#>
#> Main Arguments:
#>   penalty = 1.1
#>   mixture = 1
#>
#> Computational engine: glmnet
```

```
# 3. Pre-process the training data. Note: we need to one-hot encode `carrier`
# library(recipe)

# a. create recipe (specify data types, outcome variable, and predictor variables)
# b. one-hot encode the categorical predictors
rec_lasso = recipe(data = training(data_split),
             formula = arr_delay ~ .) %>% # create pre-processing recipe
  step_dummy(all_nominal_predictors(), one_hot = TRUE) # one-hot encoding of categorical

# c. Fit or learn the pre-processing values; e.g., number of levels/dummy columns
rec_prep = rec_lasso %>% prep(training = training(data_split))  # "fit" the training data

# d. Create numeric data ready for modeling
data_train = rec_prep %>% bake(new_data = training(data_split))
head(data_train)
#> # A tibble: 6 x 19
#>   dep_delay sched_arr_time arr_delay carrier_X9E carrier_AA carrier_AS
#>       <dbl>          <int>     <dbl>       <dbl>      <dbl>      <dbl>
#> 1        -1           1022       -18           0          0          0
#> 2        -6            837       -25           0          0          0
#> 3         0            915       -17           0          0          0
#> 4         0           1100       -21           0          0          0
#> 5        -1            740       -19           0          0          0
#> 6        -3            940       -18           0          0          0
#> # i 13 more variables: carrier_B6 <dbl>, carrier_DL <dbl>, carrier_EV <dbl>,
#> #   carrier_F9 <dbl>, carrier_FL <dbl>, carrier_HA <dbl>, carrier_MQ <dbl>,
#> #   carrier_OO <dbl>, carrier_UA <dbl>, carrier_US <dbl>, carrier_VX <dbl>,
#> #   carrier_WN <dbl>, carrier_YV <dbl>
```

```
# 4. Fit model
lasso_hat = fit_xy(lasso, x = data_train %>% select(-arr_delay),
                   y = data_train %>% pull(arr_delay) )
```

```
#    Note: instead of training the recipe separately, this "workflow" combines
#          steps 3 and 4 to pre-process and fit the model all in one step
# library(workflows)
wf_hat = workflow(
  spec = lasso,
  preprocessor = rec_lasso
  ) %>%
  fit(training(data_split))
```

```
# 5. Pre-process *testing* data
data_test = rec_prep %>% bake(new_data = testing(data_split))
head(data_test)
#> # A tibble: 6 x 19
#>   dep_delay sched_arr_time arr_delay carrier_X9E carrier_AA carrier_AS
#>       <dbl>          <int>     <dbl>       <dbl>      <dbl>      <dbl>
#> 1        11            931        14           0          0          0
#> 2        -2            947        29           0          0          0
#> 3        -1            810        14           0          1          0
#> 4        -3           1035       -12           0          1          0
#> 5        -4           1220         6           0          1          0
#> 6        -2           1425        -4           0          0          0
#> # i 13 more variables: carrier_B6 <dbl>, carrier_DL <dbl>, carrier_EV <dbl>,
#> #   carrier_F9 <dbl>, carrier_FL <dbl>, carrier_HA <dbl>, carrier_MQ <dbl>,
#> #   carrier_OO <dbl>, carrier_UA <dbl>, carrier_US <dbl>, carrier_VX <dbl>,
#> #   carrier_WN <dbl>, carrier_YV <dbl>
```

```
# 6. Predict value of test data (set penalty again here)
y_hat = predict(lasso_hat, data_test, penalty = 1.1)
y_hat = predict(wf_hat, testing(data_split), penalty = 1.1)
head(y_hat)
#> # A tibble: 6 x 1
#>   .pred
#>   <dbl>
#> 1  4.93
#> 2 -7.96
#> 3 -6.67
#> 4 -8.66
#> 5 -9.65
#> 6 -7.96
```

```
# 7. Evaluate predictions
# library(yardstick)
RMSE = rmse_vec(
  truth = testing(data_split)$arr_delay,
  estimate = y_hat$.pred
  )
glue::glue("The test RMSE for the lasso model is: {round(RMSE, 2)}")
#> The test RMSE for the lasso model is: 17.79
```