

# Lab 4

## Stats 32: Introduction to R for Undergraduates

Harrison Li

4/11/2024

Note: The content of this lab is partially borrowed from Kenneth Tay's course materials in the Autumn 2019 iteration of this course.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## v dplyr      1.1.4      v readr      2.1.5
## v forcats    1.0.0      v stringr   1.5.1
## v ggplot2    3.4.4      v tibble    3.2.1
## v lubridate  1.9.3      v tidyr     1.3.1
## v purrr      1.0.2
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(nycflights13)
```

Recall the `Stanford_small` tibble we created in Lab 3, consisting of all flights in the `flights` tibble from the `nycflights13` package that went to SFO, SJC, or OAK, with an additional column for speed:

```
Stanford_small <- flights %>%
  filter(dest == "SFO" | dest == "SJC" | dest == "OAK") %>%
  select(month, carrier, origin, dest, air_time, distance) %>%
  mutate(speed = distance / air_time * 60)
```

### `group_by()`, `summarise()`

We'd like to learn some quantitative information about the flights.

For example, what was the mean/median air time for flights in our `Stanford_small` dataset? We can use the `summarise()` function to help us:

```
Stanford_small %>%
  summarise(mean_airtime = mean(air_time))
```

```
## # A tibble: 1 x 1
##   mean_airtime
##         <dbl>
## 1           NA
```

```
Stanford_small %>%
  summarise(median_airtime = median(air_time))
```

```
## # A tibble: 1 x 1
##   median_airtime
##         <dbl>
## 1             NA
```

The NAs are causing us trouble! We need to specify the `na.rm = TRUE` option to remove NAs from consideration:

```
Stanford_small %>%
  summarise(mean_airtime = mean(air_time, na.rm = TRUE),
            median_airtime = median(air_time, na.rm=TRUE))
```

```
## # A tibble: 1 x 2
##   mean_airtime median_airtime
##         <dbl>         <dbl>
## 1       346.           345
```

`summarise()` gives me a summary of the entire dataset. If I want summaries broken down for each possible value of a grouping variable, then I have to use `summarise()` in conjunction with `group_by()`. `group_by()` changes the unit of analysis from the whole dataset to individual groups. The following code groups the dataset by carrier, then computes the summary statistic for each group. We use `arrange()` to order in decreasing order of airtime.

```
Stanford_small %>%
  group_by(carrier) %>%
  summarise(mean_airtime = mean(air_time, na.rm = TRUE)) %>%
  arrange(desc(mean_airtime))
```

```
## # A tibble: 5 x 2
##   carrier mean_airtime
##   <chr>         <dbl>
## 1 AA           348.
## 2 VX           348.
## 3 DL           347.
## 4 B6           347.
## 5 UA           344.
```

I can also group by more than one variable. For example, if I wanted to count the number of flights for each carrier in each month, I could use the following code:

```
Stanford_small %>%
  group_by(month, carrier) %>%
  summarise(count = n())
```

```
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 60 x 3
## # Groups:   month [12]
##   month carrier count
##   <int> <chr>   <int>
## 1     1 AA      120
## 2     1 B6      121
## 3     1 DL      142
## 4     1 UA      422
## 5     1 VX      124
## 6     2 AA      108
## 7     2 B6      106
## 8     2 DL      127
```

```
## 9      2 UA      378
## 10     2 VX      104
## # i 50 more rows
```

1. Find the minimum, maximum, average, and standard deviation of air time for each (origin, dest) pair in `Stanford_small`.
2. Out of all the flights in the original `flights` tibble, on average, which origin airport had the longest delays on flights to SFO?

## Joins

We will illustrate joins using a very small and simple example.

`dplyr` comes loaded with three small tibbles: `band_members`, `band_instruments`, and `band_instruments2`:

```
band_members
```

```
## # A tibble: 3 x 2
##   name band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
band_instruments
```

```
## # A tibble: 3 x 2
##   name plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

```
band_instruments2
```

```
## # A tibble: 3 x 2
##   artist plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

Suppose we want to add instrument information to the `band_members` tibble. We can do this by left joining `band_members` onto `band_instruments`, using the common key variable `name`:

```
left_join(band_members, band_instruments, by="name")
```

```
## # A tibble: 3 x 3
##   name band  plays
##   <chr> <chr> <chr>
## 1 Mick  Stones <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
```

Notice we have one observation for each row in `band_members`. Since there is no entry for “Mick” in `band_instruments`, Mick’s instrument is filled in with NA.

3. Note that we get the same result as above if we exclude the `by=` argument to `left_join()` specifying the key variable:

```
left_join(band_members, band_instruments)
```

```
## Joining with `by = join_by(name)`  
## # A tibble: 3 x 3  
##   name band plays  
##   <chr> <chr> <chr>  
## 1 Mick Stones <NA>  
## 2 John Beatles guitar  
## 3 Paul Beatles bass
```

What is the key variable in this join? Why do we get the same result? Hint: Check the help page!

4. Return the same tibble by instead joining `band_members` onto `band_instruments2`. Note: `band_instruments` and `band_instruments2` have the same information, however they have different column names.

If we wanted to omit any rows where there was no match in the join, we could use `inner_join()`:

```
band_members %>%  
  inner_join(band_instruments)
```

```
## Joining with `by = join_by(name)`  
## # A tibble: 2 x 3  
##   name band plays  
##   <chr> <chr> <chr>  
## 1 John Beatles guitar  
## 2 Paul Beatles bass
```

Conversely, if we wanted an entry for any row that appears in either tibble, we can use `full_join()` (also known as an outer join):

```
band_members %>%  
  full_join(band_instruments)
```

```
## Joining with `by = join_by(name)`  
## # A tibble: 4 x 3  
##   name band plays  
##   <chr> <chr> <chr>  
## 1 Mick Stones <NA>  
## 2 John Beatles guitar  
## 3 Paul Beatles bass  
## 4 Keith <NA> guitar
```

5. Now do a right join of `band_members` onto `band_instruments`. Explain the difference, if any, from the left join.
6. If I swap the order of the two tibbles in `inner_join()`, does the result change? Why or why not?
7. Same as the previous question, but for `full_join()`.

Now we use `rbind()` to duplicate all the entries in `band_instruments()`:

```
band_instruments_duped <- rbind(band_instruments, band_instruments)  
band_instruments_duped
```

```
## # A tibble: 6 x 2  
##   name plays  
##   <chr> <chr>
```

```
## 1 John guitar
## 2 Paul bass
## 3 Keith guitar
## 4 John guitar
## 5 Paul bass
## 6 Keith guitar
```

Now our left joined tibble also has duplicate entries:

```
band_members %>%
  left_join(band_instruments_duped)
```

```
## Joining with `by = join_by(name)`

## # A tibble: 5 x 3
##   name band plays
##   <chr> <chr> <chr>
## 1 Mick Stones <NA>
## 2 John Beatles guitar
## 3 John Beatles guitar
## 4 Paul Beatles bass
## 5 Paul Beatles bass
```

Moral of the story: when you do a left join, you get *at least* one for each row in the first tibble. If there is more than one match, you will get all matches, in separate rows.

8. Use your knowledge to predict what would happen if we replaced `band_instruments` in the right join, inner join, and full outer join from above with `band_instruments_duped`. Check your answers with code.