Lecture 9

- More on Pipe Operator %>%
- Long Lines with %>%
- Short Lines with %>%
- **dplyr** Cheatsheets
- Exercises

More on Pipe Operator: %>%

- %>% should always have a space before it, and it should usually be followed by a new line. After the first step, each line should be indented by two spaces. This structure makes it easier to add new steps (or rearrange existing steps) and harder to overlook a step.

```
#Good Sample Code
iris %>%
  group_by(Species) %>%
  summarize_if(is.numeric, mean) %>%
  ungroup() %>%
  gather(measure, value, -Species) %>%
  arrange(value)
```

```
#Bad Sample Code
iris %>% group_by(Species) %>% summarize_all(mean) %>%
ungroup %>% gather(measure, value, -Species) %>%
arrange(value)
```

Long Lines with %>%

- If the arguments to a function do not all fit on one line, put each argument on its own line and indent:

```
iris %>%
  group_by(Species) %>%
  summarise(
    Sepal.Length = mean(Sepal.Length),
    Sepal.Width = mean(Sepal.Width),
    Species = n_distinct(Species)
  )
```

Short Lines with %>%

- A one-step pipe can stay on one line, but unless you plan to expand it later on, you should consider rewriting it to a regular function call.

```
# Good Sample Codes
iris %>% arrange(Species)

iris %>%
  arrange(Species)

arrange(iris, Species)
```

Short Lines with %>%

- Sometimes it is useful to include a short pipe as an argument to a function in a longer pipe.

```
# Good Sample Code
x %>%
  select(a, b, w) %>%
  left_join(y %>% select(a, b, v), by = c("a", "b"))


# Better Sample Code
x_join <- x %>% select(a, b, w)
y_join <- y %>% select(a, b, v)
left_join(x_join, y_join, by = c("a", "b"))
```

Assignments with %>%

- There are three acceptable forms of assignment:

1. Variable name and assignment on separate lines:

```
iris_long <-
  iris %>%
  gather(measure, value, -Species) %>%
  arrange(-value)
```

2. Variable name and assignment on the same line:

```
iris_long <- iris %>%
  gather(measure, value, -Species) %>%
  arrange(-value)
```

3. Assignment at the end of the pipe with ->:

```
Iris %>%
      gather(measure, value, -Species) %>%
      arrange(-value) ->
      iris_long
```

Split-apply-combine Data Analysis

- Many data analysis tasks can be approached using the “split-apply-combine” paradigm:
  - split the data into groups
  - apply some analysis to each group
  - then combine the results using summarize

- R functions operating on vectors that contains missing data will return NA. It’s a way to make sure that users know they have missing data, and make a conscious decision on how to deal with it.

dplyr Cheatsheets

- https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

Exercises

1. Refer back to the lists of useful mutate and filtering functions. Describe how each operation changes when you combine it with grouping.

2. Which plane (tailnum) has the worst on-time record?

3. What time of day should you fly if you want to avoid delays as much as possible?

4. For each destination, compute the total minutes of delay. For each flight, compute the proportion of the total delay for its destination.

5. Delays are typically temporally correlated: even once the problem that caused the initial delay has been resolved, later flights are delayed to allow earlier flights to leave. Using lag(), explore how the delay of a flight is related to the delay of the immediately preceding flight.

6. Look at each destination. Can you find flights that are suspiciously fast? (i.e. flights that represent a potential data entry error). Compute the air time of a flight relative to the shortest flight to that destination. Which flights were most delayed in the air?

7. Find all destinations that are flown by at least two carriers. Use that information to rank the carriers.

8. For each plane, count the number of flights before the first delay of greater than 1 hour.