**Data Transformation with dplyr**

Visualization is an important tool for insight generation, but it is rare that you get the data in exactly the right form you need. Often you’ll need to create some new variables or summaries, or maybe you just want to rename the variables or reorder the observations in order to make the data a little easier to work with. You’ll learn how

to do all that how to transform your data using the **dplyr** package and a new dataset

on flights departing New York City in 2013.

**Prerequisites**

To install any packages : **packages.install(“nycflights13”)**

Library(nycflights13)

Library(tidyverse)

**nycflights13**

To explore the flights data manipulation verbs to dplyr, we’ll use nycfkights13::flights. The data frame comes with 336,776 flight that departed from New York City in 2013.

* Flights

**dplyr Basics**

* Pick observations by their values (filter())
* Reorder the rows (arrange()).
* Pick Variables by their names (select()).
* Create new variables with functions of existing variables (mutate()).
* Collapse many value down in a single summary (summary()).

**Filter Rows with filter()**

**filter()** allows you to subset observations based on their values. The first argument is the name of the data frame. The second and subsequent arguments are the expressions that filter the data frame. For example, we can select all flights on January 1st with:

* Filter(flights, month ==1, day==1)
* Jan1 <- filter(flights,month ==1,day ==1)
* (dec25 <- filter(flights, month ==12, day==25))

**Exercises**

**1. Find all flights that:**

**a**. Had an arrival delay of two or more hours

**library**(nycflights13)

**library**(tidyverse)

#glimpse(flights)

**filter**(flights, arr\_delay > 119 & arr\_delay < 121)

**b**. Flew to Houston (IAH or HOU)

**filter**(flights, dest == 'HOU' | dest == 'IAH')

**c**. Were operated by United, American, or Delta

**filter**(flights, carrier == "UA" | carrier == 'AA' | carrier == 'DA')

**d**. Departed in summer (July, August, and September)

**filter**(flights, month == 7 | month == 8 | month == 9)

**e**. Arrived more than two hours late, but didn’t leave late

**filter**(flights,arr\_delay > 120 & dep\_delay <= 0)

**f**. Were delayed by at least an hour, but made up over 30 minutes in flight

**filter**(flights, dep\_delay > 60 & air\_time > 30)

**g**. Departed between midnight and 6 a.m. (inclusive)

**filter**(flights, hour >= 0 & hour <= 6)

**2.How many flights have a missing dep\_time? What other variables**

**are missing?**

**filter**(flights, **is.na**(dep\_time) )

**Arrange Rows with arrange()**

**arrange()** works similarly to **filter()** except that instead of selecting rows, it changes their order. It takes a data frame and a set of column names (or more complicated expressions) to order by. If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns:

* arrange(flights,year,month,day)

**desc()** to arrange in descending order

* arrange(flights, desc(arr\_delay))

**Select Columns with select()**

It’s not uncommon to get datasets with hundreds or even thousands of variables. In this case, the first challenge is often narrowing in on the variables you’re actually interested in. **select()** allows you to rapidly zoom in on a useful subset using operations based on the names of the variables. **select()** is not terribly useful with the flight data because we only have 19 variables, but you can still get the general idea.

* select(flights,year,month,day)
* select(flights, year:day)
* select(flights, -(year:day))

select() can be used to rename variables, but it’s rarely useful because it drops all of the variables not explicitly mentioned. Instead, use rename(), which is a variant of select()that keeps all the variables that aren’t explicitly mentioned:

* rename(flights,tail\_num = tailnum)

Another option is to use select() in conjunction with the everything() helper. This is useful if you have a handful of variables you’d like to move to the start of the data frame:

* select(flights,time\_hour,air\_time,everything())

**Exercises**

**1.** What happens if you include the name of a variable multiple

times in a select() call?

**select**(flights, dep\_time, dep\_time)

**2.**What does the one\_of() function do? Why might it be helpful in conjunction with this vector?

vars <- **c**("year", "month", "day", "dep\_delay", "arr\_delay")

**select**(flights, **one\_of**(vars) )

**Add New Variables with mutate()**

Besides selecting sets of existing columns, it’s often useful to add new columns that are functions of existing columns. That’s the job of mutate(). mutate() always adds new columns at the end of your dataset so we’ll start by creating a narrower dataset so we can see the new variables. Remember that when you’re in RStudio, the easiest way to see

all the columns is View():

* flights\_sml <-select(flights,year:day, ends\_with(“delay”), distance,air\_time)

mutate(flights\_sml, gain = arr\_delay – dep\_delay, speed = distance /air\_time \*60)

If you only want to keep the new variables, use transmute():

* transmute(flights, gain = arr\_delay – dep\_delay,hours= air\_time /60,gain\_per\_hour = gain/hours)

**Grouped Summaries with summarize()**

The last key verb is **summarize()**. It collapses a data frame to a single row:

* summarize(flights, delay = mean(dep\_delay,na.rm = TRUE))

(We’ll come back to what that na.rm = TRUE means very shortly.)

summarize() is not terribly useful unless we pair it with group\_by().

This changes the unit of analysis from the complete dataset to individual

groups.

**Combining Multiple Operations with the Pipe**

Imagine that we want to explore the relationship between the distance and average delay for each location. Using what you know about **dplyr**, you might write code like this:

* by\_dest <- groupby(flights,dest)

delay <- summarize(by\_dest, count = n(),

dist = mean(distance, na.rm = TRUE),

delay = mean(arr\_delay,na.rm = TRUE)

)

Delay <- filter(delay,count >20, dest !=”HNL”)

Ggplot(data = delay, mapping = aes(x =dist, y=delay))+

Geom\_point(aes(size = count), alpha = 1/3) +

Geom\_smooth(se = FALSE)