SPEC Lab REU R Resources: Data Management I with tidyverse

Benjamin A.T. Graham and Gaea Morales

Summer 2022

Viewing, Arranging, & Changing Data

This walkthrough work provides an introduction to data management in R using the tidyverse and dplyr packages. We will start with some review and then work with data on Flights out of Houston.

First, set up the header on your R script. Use # at the beginning of a line of code to write notes and comments. My header looks like this:

```
####################################
#Data Management I Walkthrough Script
#Ben Graham
#Last updated: September 17, 2021
#Lot's of cool stuff in the dplyr() package.
#Creating variables, subsetting data, summarizing data ####################################
```

Now, install the packages we will need – you only need to do this once. You can use the Tools dropdown menu and select "Install Packages", which will allow you to search for and install each of the packages below.

Once they are installed, you use the `library()` command to load the package for use.

```
library(dplyr)
library(tidyverse)
library(hflights)
```

Review: Vectors, Objects, & Operators

R thinks in Vectors. So let's start by making a vector. c() means concatenate, or "stick together in a series". We will make a vector called "world.pop" (sound familiar?)

```
world.pop <- c(2525, 3026, 3691, 4449, 5321, 6128, 6916)
```

Generating a sequence

The syntax for generating a function is: `seq(a, b, c)`, where a = start, b = end, c = how much to increment by.

1. Create a vector called "newvector" from 1 to 10 counting by 1s

```
newvector <- seq(1, 10, 1)
```

2. Create a vector called "year" with values from 1950-2010, counting by 10s. Have a go!

```
year <- seq(1950, 2010, 10)
```

3. Now, let's assign the year vector as the names of the population vector, because each of the values in our initial vector is actually (roughly) the world population in millions, every 10 years from 1950-2010.

```
names(world.pop) <- year

#print all the values in the world.pop vector
print(world.pop)
```

```
## 1950 1960 1970 1980 1990 2000 2010
## 2525 3026 3691 4449 5321 6128 6916
```

```
#alternatively
world.pop
```

```
## 1950 1960 1970 1980 1990 2000 2010
## 2525 3026 3691 4449 5321 6128 6916
```

A little playing around with our vectors.

As we go through this training, we will learn how to subset dataframes using filter() in the dplyr package, but for now let's play with the subset() function.

Print all the values in world.pop except for 2000, and subset(a, b), where a = object and b = rule.

Helpful Hints: | is the symbol for "or" == is a test for equality, it can be read as "is exactly equal to" != is a test for inequality, "is not equal to"

```
smallworld <- subset(world.pop, year < 2000 | year == 2010)
print(smallworld)
```

```
## 1950 1960 1970 1980 1990 2010
## 2525 3026 3691 4449 5321 6916
```

There are always many ways to do things in R. Can you write this a bit more simply?

```
smallworld2 <- subset(world.pop, year != 2000)
print(smallworld2)
```

```
## 1950 1960 1970 1980 1990 2010
## 2525 3026 3691 4449 5321 6916
```

```
# Or

world.pop
```

```
## 1950 1960 1970 1980 1990 2000 2010
## 2525 3026 3691 4449 5321 6128 6916
```

```
print(world.pop[c(-6, -5)]) #removes columns 6 and 5
```

```
## 1950 1960 1970 1980 2010
## 2525 3026 3691 4449 6916
```

Data Management

Let's load the data we will use today. We are going to use the `hflights` dataset that is preloaded in R. The `table_df()` command just loads in this data as a dataframe.

```
hflights <-tbl_df(hflights)
```

```
## Warning: `tbl_df()` was deprecated in dplyr 1.0.0.
## Please use `tibble::as_tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

Let's take a look!

```
str(hflights)
```

```
## tibble [227,496 x 21] (S3: tbl_df/tbl/data.frame)
##  $ Year             : int [1:227496] 2011 2011 2011 2011 2011 2011 2011 2011 2011 2011 ...
##  $ Month            : int [1:227496] 1 1 1 1 1 1 1 1 1 1 ...
##  $ DayofMonth       : int [1:227496] 1 2 3 4 5 6 7 8 9 10 ...
##  $ DayOfWeek        : int [1:227496] 6 7 1 2 3 4 5 6 7 1 ...
##  $ DepTime          : int [1:227496] 1400 1401 1352 1403 1405 1359 1359 1355 1443 1443 ...
##  $ ArrTime          : int [1:227496] 1500 1501 1502 1513 1507 1503 1509 1454 1554 1553 ...
##  $ UniqueCarrier    : chr [1:227496] "AA" "AA" "AA" "AA" ...
##  $ FlightNum        : int [1:227496] 428 428 428 428 428 428 428 428 428 428 ...
##  $ TailNum          : chr [1:227496] "N576AA" "N557AA" "N541AA" "N403AA" ...
##  $ ActualElapsedTime: int [1:227496] 60 60 70 70 62 64 70 59 71 70 ...
##  $ AirTime          : int [1:227496] 40 45 48 39 44 45 43 40 41 45 ...
##  $ ArrDelay         : int [1:227496] -10 -9 -8 3 -3 -7 -1 -16 44 43 ...
##  $ DepDelay         : int [1:227496] 0 1 -8 3 5 -1 -1 -5 43 43 ...
##  $ Origin           : chr [1:227496] "IAH" "IAH" "IAH" "IAH" ...
##  $ Dest             : chr [1:227496] "DFW" "DFW" "DFW" "DFW" ...
##  $ Distance         : int [1:227496] 224 224 224 224 224 224 224 224 224 224 ...
##  $ TaxiIn           : int [1:227496] 7 6 5 9 9 6 12 7 8 6 ...
##  $ TaxiOut          : int [1:227496] 13 9 17 22 9 13 15 12 22 19 ...
##  $ Cancelled        : int [1:227496] 0 0 0 0 0 0 0 0 0 0 ...
##  $ CancellationCode : chr [1:227496] "" "" "" "" ...
##  $ Diverted         : int [1:227496] 0 0 0 0 0 0 0 0 0 0 ...
```

Helpful Hint: Click the `hflights` object in the Environment tab in R Studio to see it in table view. Or type `View(hflights)` into your R script.

Using select() to subset dataframes by choosing columns (i.e. variables)

Start by selecting a subset of variables and renaming them. We're going to drop the variables we don't want by selecting the ones we want to keep. This will reduce the number of columns (i.e. variables) in our dataframe while leaving the number of rows (i.e. observations) the same.

```
data <- hflights %>%
  select(Month, #keep the month variable
         DayOfWeek, #keep the day of week variable
         Airline = UniqueCarrier, #rename the UniqueCarrier variable
         # to Airline and keep it
         Time = ActualElapsedTime, #rename and keep the elapsed time variable
         Origin:Cancelled) #also keep all the variables "Origin" through "Cancelled"
```

We can use select() to identify the variables we don't want, using a - sign to denote the variables we want to drop.

```
data <- data %>%
  select(-Cancelled)
str(data)
```

```
## tibble [227,496 x 9] (S3: tbl_df/tbl/data.frame)
##  $ Month    : int [1:227496] 1 1 1 1 1 1 1 1 1 1 ...
##  $ DayOfWeek: int [1:227496] 6 7 1 2 3 4 5 6 7 1 ...
##  $ Airline  : chr [1:227496] "AA" "AA" "AA" "AA" ...
##  $ Time     : int [1:227496] 60 60 70 70 62 64 70 59 71 70 ...
##  $ Origin   : chr [1:227496] "IAH" "IAH" "IAH" "IAH" ...
##  $ Dest     : chr [1:227496] "DFW" "DFW" "DFW" "DFW" ...
##  $ Distance : int [1:227496] 224 224 224 224 224 224 224 224 224 224 ...
##  $ TaxiIn   : int [1:227496] 7 6 5 9 9 6 12 7 8 6 ...
##  $ TaxiOut  : int [1:227496] 13 9 17 22 9 13 15 12 22 19 ...
```

We can also create a dataframe with all variables that contain the word "Time".

```
testframe <- hflights %>%
  select(contains("Time"))
head(testframe)
```

```
## # A tibble: 6 x 4
##   DepTime ArrTime ActualElapsedTime AirTime
##     <int>   <int>             <int>   <int>
## 1    1400    1500                60      40
## 2    1401    1501                60      45
## 3    1352    1502                70      48
## 4    1403    1513                70      39
## 5    1405    1507                62      44
## 6    1359    1503                64      45
```

Using filter() to subset dataframes by choosing rows (i.e. observations)

First, let's create a simple vector object that lists all LA-area aiports.

```
la_airports <- c("LAX", "ONT", "SNA", "BUR", "LGB")
          #filters for the flights that have LA airports
la_airports #to see what that vector looks like...
```

```
## [1] "LAX" "ONT" "SNA" "BUR" "LGB"
```

Now, we want to use filter() to create a new, smaller dataframe that contains only flights with LA-area destinations. This will reduce the number of rows (i.e. observations) in our dataframe. The number of columns (i.e., variables) will remain the same.

```
la_flights <- data %>%
  filter(Dest %in% la_airports)

la_flights_alt <- data %>%
filter(Dest == c("LAX", "BUR")) # la_flights_alt only contains flights into
                                  # LAX and Burbank
```

Let's use the head() command to display the first few rows of these new, smaller, dataframes.

```
head(la_flights)
```

```
## # A tibble: 6 x 9
##   Month DayOfWeek Airline  Time Origin Dest  Distance TaxiIn TaxiOut
##   <int>     <int> <chr>   <int> <chr>  <chr>    <int>  <int>   <int>
## 1     1         1 CO        227 IAH    LAX       1379      8      20
## 2     1         1 CO        229 IAH    LAX       1379     11      17
## 3     1         1 CO        236 IAH    LAX       1379     10      27
## 4     1         1 CO        211 IAH    ONT       1334      5      17
## 5     1         1 CO        243 IAH    SNA       1347      6      35
## 6     1         1 CO        226 IAH    LAX       1379     13      15
```

```
head(la_flights_alt)
```

```
## # A tibble: 6 x 9
##   Month DayOfWeek Airline  Time Origin Dest  Distance TaxiIn TaxiOut
##   <int>     <int> <chr>   <int> <chr>  <chr>    <int>  <int>   <int>
## 1     1         1 CO        227 IAH    LAX       1379      8      20
## 2     1         1 CO        236 IAH    LAX       1379     10      27
## 3     1         1 CO        220 IAH    LAX       1379      7      12
## 4     1         1 CO        236 IAH    LAX       1379      8      33
## 5     1         1 CO        253 IAH    LAX       1379     11      30
## 6     1         1 CO        229 IAH    LAX       1379     15      14
```

Helpful Hints: In addition to the head() command which gives you a preview, we can use `View()` to see all our data.

mutate() to create and/or transform variables

We can use the `mutate()` function to add variables to our dataframe. We often use this command to create new variables that combine or transform information already present in other variables in our dataframe. For example, we can use `mutate()` to create a variable with the natural log of GDP in a dataframe that already has a GDP variable.

Here, we will combine information on time taxiing in and time taxiing out to create a variable, TaxiTotal, that captures the total time spent taxiing for each flight. We can also create a variable, TaxiProp that captures time spent taxiing as a share of total flight time.

```
la_flights <- la_flights %>%
  mutate(TaxiTotal = TaxiIn + TaxiOut,
         TaxiProp = TaxiTotal/Time) #now TaxiTotal and TaxiProp variables have been added
```

ifelse() for adding conditions with mutate()

`ifelse()` is a very versatile helper function. Here we use it along with `mutate()` to create a "Weekend" variable that codes whether each flight is on a weekend. We then use the `filter()` function to keep only the weekend flights. **Remember:** ifelse(condition, if TRUE [do something], if FALSE [do something]).

```
la_flights <- la_flights %>%
      # create a weekend variable: if the DayofWeek is 1 or 7 (Sunday or Saturday),
      # then assign with a value of 1, else it is a weekday, and assign it 0
  mutate(Weekend = ifelse(DayOfWeek %in% c(1,7), 1, 0)) %>%
  filter(Weekend == 1) # filter for all the Weekend = 1 rows
                       # (those flights will be on a weekend)
```

group_by() and summarise() for summarising variables by observations

We will cover the combination of `group_by()` and `summarise()` more in Data Management 3 because they can take a bit of time to wrap your head around.

Let's say I hate taxiing and I want to know which days of the week have the shortest taxi times. Once we use `group_by()` to denote "DayofWeek" as the grouping variable, I can use the `summarise()` function to create a new variable "AverageTime" that has the average taxi time for each day of the week.

Note that `summarise()` radically changes my dataframe. Instead of one observation for each individual flight, I now have just one observation for each day of the week, with summary information about the average and maximum taxi times on those days.

```
weekday_time <- la_flights %>%
  group_by(DayOfWeek) %>%
  summarise(AverageTime = mean(Time, na.rm = T),
            MaxTaxiTotal = max(TaxiTotal, na.rm = T)) # na.rm tells R to ignore
                                                      # missing values
```

For legibility, let's reorder the output in ascending order using `arrange()`, with "MaxTaxiTotal" as a tie breaker.

```r
weekday_time <- la_flights %>%
  group_by(DayOfWeek) %>%
  summarise(AverageTime = mean(Time, na.rm = T),
            MaxTaxiTotal = max(TaxiTotal, na.rm = T)) %>%
  arrange(AverageTime, MaxTaxiTotal) # Default is ascending order, though you can specify
                                     # descending order using the desc() operator
```

n()

We can also pair `group_by()` with `n()` to count the number of flights each airline has in our dataframe.

```r
carriers <- la_flights %>%
  group_by(Airline) %>%
  summarise(NoFlights = n()) %>% # n() tells dplyr to count all the observations
                                 #for the groups specified in group_by().
  arrange(desc(NoFlights)) #desc() for descending order.

# We can take a look at the data
head(carriers)
```

```
## # A tibble: 3 x 2
##   Airline NoFlights
##   <chr>       <int>
## 1 CO           1943
## 2 WN            393
## 3 MQ            228
```

Putting it all together with piping %>%

We want a clean line of code without comments. We can use piping to make all these operations happen at once. Let's give it a go!

```r
carriers_new <- hflights %>%
  select(Month,
         DayOfWeek,
         Airline = UniqueCarrier,
         Time = ActualElapsedTime,
         Origin:TaxiOut) %>%
  filter(Dest %in% c("LAX", "ONT", "SNA", "BUR", "LGB")) %>%
  mutate(TaxiTotal = TaxiIn + TaxiOut,
         TaxiProp = TaxiTotal/Time,
         Weekend = ifelse(DayOfWeek %in% c(1,7), 1, 0)) %>%
  filter(Weekend == 1) %>%
  group_by(Airline) %>%
  summarise(NoFlights = n()) %>%
  arrange(desc(NoFlights))
```

Breaking it Down

```
carriers_new <- hflights %>%
  #select the variables you want to work with select
  select(Month,
         DayOfWeek,
         Airline = UniqueCarrier,
         Time = ActualElapsedTime,
         Origin:TaxiOut) %>%

  #then filter by the desired destinations (keeps only rows you want/specify)
  filter(Dest %in% c("LAX", "ONT", "SNA", "BUR", "LGB")) %>%

  #add variables whose values are the result of operations between other variables
  mutate(TaxiTotal = TaxiIn + TaxiOut,
         TaxiProp = TaxiTotal/Time,
         Weekend = ifelse(DayOfWeek %in% c(1,7), 1, 0)) %>%

  #get only the rows whose weekend variable has a value of 1
  filter(Weekend == 1) %>%

  group_by(Airline) %>%
  summarise(NoFlights = n()) %>% #adds a variable called NoFlights that counts the number
                                 #corresponding to the group_by value (which is Airline)
                                 #(counts the number of flights that each airline has)
  arrange(desc(NoFlights)) #arrange in descending order
```

Saving the carriers_new dataframe that you have created

The default is for R to save any ouput to your working directory, so you will want to specify the full filepath you want to save the file to. Because this object is a single dataframe, we will save it in the `.rds` format. If you want to save multiple dataframes to a single data file you can use `.RData`.

In my case, I will use:

```
saveRDS(carriers_new, file = "/Volumes/GoogleDrive/My Drive/SPEC Summer/
        REU 2022/training output/carriers_new_BEN.rds")
```

```
## Error in gzfile(file, mode): cannot open the connection
```

And we're off!

When you have made it through this walkthrough, we highly recommend saving your R script somewhere you can find it later. Your own annotations on your own code is a priceless reference for future you as you move into applying these skills (and circle back to apply them again a few years from now or whenever.)