SPEC Lab REU R Resources: Data Management I with tidyverse

Summer 2021

Viewing, Arranging, & Changing Data

This workshop 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: April 10, 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

function seq(a, b, c) a = start, b = end, c = how much to increment by

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

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

```
newvector <- seq(1, 10, 1)
year <- seq(1950, 2010, 10)
```

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)
#alternatively
world.pop
```

###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

subset(a, b), where a = object and b = rule

`|` 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)
```

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)



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

##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)
```

Let's take a look!

```
str(hflights)
```

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", as we read from l
```

We can also 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)
```

Or we can create data frame with all variables that contain the word "Time".

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

###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 destinations of

#to see what that vector looks like...
la_airports
```

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)
head(la_flights_alt)
```

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

###mutate()

We can use the `mutate()` function to add variables to our dataframe. Often we use this command to create new variables that combine or transform information already present in other variables in our dataframe, like creating a variable for 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 to our datafr
```

###ifelse()

`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 or. 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 the Weekend
  #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 Saturday or Sun
```

###group_by() and summarise()

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 and just
```

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

```
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 if
```

###n()

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

```
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.
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
head(carriers)
```

##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!

```
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))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
head(carriers_new)
```

###Breaking it down

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

  #then filter by the desired destinations (removes any rows that don't contain your chosen airports)
  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
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

##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.)