SPEC Lab REU R Resources: Data Management I with tidyverse

Benjamin A.T. Graham and Gaea Morales

Summer 2024

This walkthrough introduces data management in R using the `tidyverse` and `dplyr` packages. After reviewing the foundational concepts covered in Module 1, we will guide you through key data management techniques, focusing on applying functions, such as filtering, selecting, and summarizing data, and using piping (`%>%`), which will be central to your future work in R.

Initial Setup

Let's begin by setting up your workspace and loading the necessary packages to manage and analyze our dataset. First, you'll want to set up a header in your R script to include information such as the script's purpose, who wrote it, and the last update. Use `#` to add comments that won't affect the code's execution. My header looks like this:

```
##########################################################
# Data Management I Walkthrough Script
# Ben Graham
# Last updated: September 5, 2024
# Lot's of cool stuff in the dplyr() package.
# Creating variables, subsetting data, summarizing data
##########################################################
```

Next, set your working directory to the folder where you'll store the project files using the `setwd()` function. As we discussed in the Organization for Collaboration Module, this step is essential for maintaining personal organization and enhancing teamwork by ensuring that all collaborators can easily navigate the project's file structure. Replace `"YourFolderPath"` in the example below with the actual path to your data.

```
# Set working directory
#setwd("YourFolderPath")
```

Finally, we need to install and load the required libraries: `dplyr`, `tidyverse`, and `hflights`. You can use the `Tools` dropdown menu and select `Install Packages` or run the `install.packages()` function to install the libraries. You only need to install these packages once, and then use the `library()` command to load the packages for use.

We're going to use the `hflights` dataset, which is a pre-packaged dataset that contains information on flights out of Houston. As a reminder, pre-packaged datasets come with R, so you don't need to import them from external files—they are available for immediate use.

```
# Load required libraries
library(dplyr)
library(tidyverse)
library(hflights)
```

Review: Vectors, Objects, & Operators

Remember that R thinks in vectors, so let's start by creating one. Using `c()` (concatenate), we'll make a vector named `world.pop` that contains population data for different years (sound familiar?).

```r
# Create 'world.pop' vector
world.pop <- c(2525, 3026, 3691, 4449, 5321, 6128, 6916)
```

Sequences

Next, we'll use the `seq()` function to generate a sequence of numbers, which will be useful for organizing data. Its syntax, `seq(a, b, c)`, requires three parameters: `a` is the starting point, `b` is the end point, and `c` is the increment by which to count. For example, create a vector called `newvector` that includes numbers from 1 to 10, incrementing by 1.

```r
# Create vector `newvector` with numbers from 1 to 10, incrementing by 1
newvector <- seq(1, 10, 1)
```

Also, let's create a `year` vector containing decades from 1950 to 2010 and assign these years as names for the `world.pop` vector.

```r
# Create vector `year` with numbers from 1950 to 2010, counting by decades
year <- seq(1950, 2010, 10)

# Assign the 'year' vector as names to the 'world.pop' vector
names(world.pop) <- year

# Display 'world.pop' vector
world.pop
```

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

Subsetting Vectors

Let's begin by familiarizing ourselves with the `subset()` function for manipulating vectors. The `subset()` function is used as follows: `subset(a, b)`, where `a` is the vector or dataframe to operate on, and `b` is the condition applied to extract specific parts of the vector or dataframe.

To practice, let's exclude the year 2000 from the `world.pop` vector.

Helpful Hints: `|` is the symbol for "or"; `==` tests for equality (read as "is exactly equal to"); and `!=` tests for inequality (read as "is not equal to").

```r
# Subset world.pop to exclude the year 2000
smallworld <- subset(world.pop, year != 2000)

print(smallworld)
```

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

Also, you can directly manipulate vectors without using the `subset()` function. Here's how you can remove specific positions from the `world.pop` vector, such as the values in the 5th and 6th positions, which correspond to the years close to 2000.

```r
# Remove the 6th and 5th elements from the vector
world.pop
```

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

```
print(world.pop[c(-6, -5)])
```

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

Data Management with `dplyr`

Now, let's move into data management. We will use the `tbl_df()` command to load the `hflights` data as a dataframe.

```
# Load 'hflights' data
hflights <-tbl_df(hflights)
```

Let's take a look at the structure of this dataframe!

Helpful Hint: To see the data you can 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.

```
# Inspect structure of the data
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 ...
```

Select Columns with `select()`

We often need to focus on specific variables within a dataframe. The `select()` function allows us to keep or rename the columns we want to work with, reducing unnecessary clutter in our data.

This will reduce the number of columns (i.e., variables) in our dataframe while leaving the number of rows (i.e., observations) the same.

```
# Select and rename specific variables
data <- hflights %>%
  select(Month,
         DayOfWeek,
         Airline = UniqueCarrier,
         ## Rename 'UniqueCarrier' variable to 'Airline' and keep it
         Time = ActualElapsedTime,
         ## Rename 'ActualElapsedTime' to 'Time'
         Origin:Cancelled)
         ## Keep all the variables from "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.

```
# Select all variables except for 'Cancelled'
data <- data %>%
  select(-Cancelled)

# Inspect structure of the data
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`.

Helpful Hint: `head()` command which gives you a preview (usually the first six elements), we can use `View()` to see all our data.

```
# Select 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
```

Filter Rows with `filter()`

Now, let's explore how to filter rows based on specific conditions using the `filter()` function. This will allow us to narrow down our data to only include the rows that meet certain criteria.

Let's filter the dataset to create a new, smaller dataframe that contains only flights with LA-area destinations.

```r
# Create a vector that lists all LA-area airports
la_airports <- c("LAX", "ONT", "SNA", "BUR", "LGB")

# Filter flights to LA-area airports
la_flights <- data %>%
  filter(Dest %in% la_airports)

# Display results
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
```

Creating and Transforming Variables with `mutate()`

Now that we have filtered our dataset to focus on LA-area flights, let's enhance our analysis by creating new variables. The `mutate()` function lets us create new variables or transform existing ones. For example, we can create a `TaxiTotal` variable representing total taxi time for each flight by combining taxi-in and taxi-out times.

```r
# Create 'TaxiTotal' variable
la_flights <- la_flights %>%
  mutate(TaxiTotal = TaxiIn + TaxiOut,
         TaxiProp = TaxiTotal/Time)
```

We can also create a variable, `TaxiProp`, that captures time spent taxing as a share of total flight time.

```r
# Create 'TaxiTotal' variable
la_flights <- la_flights %>%
  mutate(TaxiProp = TaxiTotal/Time)
```

Using `ifelse()` to add conditions to `mutate()`

Building on our previous use of `mutate()` to create and transform variables, we can further enhance our data by using the `ifelse()` function to add a conditional logic. The structure of `ifelse()` in R is:

`ifelse(condition, value_if_TRUE, value_if_FALSE)`

Here, we use `ifelse()` with `mutate()` to create a `Weekend` variable indicating if a flight occurred on a weekend, followed by the `filter()` function to keep only the weekend flights.

```r
# Create 'Weekend' variable
la_flights <- la_flights %>%
  mutate(Weekend = ifelse(DayOfWeek %in% c(1, 7), 1, 0)) %>%
  ## Create a Weekend variable: assign 1 if DayOfWeek is 1 or 7 (Sunday or
  ## Saturday), else assign 0
  filter(Weekend == 1)
  ## Filter for all the Weekend = 1 rows (those flights will be on a weekend)
```

Grouping and Summarizing Data

We are going to take a step further and learn how to summarize our data using the `group_by()` and `summarise()` functions. 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, but for now it's important to know that the combination of `group_by()` and `summarise()` helps us aggregate data by specific variables.

For instance, let's say we want to know which days of the week have the shortest taxi times. We can use `group_by()` to specify `DayOfWeek` as the grouping variable and `summarise()` to create a new variable `AverageTime` that contains the average taxi time for each day of the week.

Note: `summarise()` radically changes the structure of our dataframe. Instead of one observation per flight, we will now have one observation per day of the week, containing summary statistics about the average and maximum taxi times.

```r
# Create 'AverageTime' variable
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 better readability, let's reorder the output in ascending order using `arrange()`, with `MaxTaxiTotal` as a tie breaker.

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

Using `n()` to count observations

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

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

head(carriers)
```

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

These steps illustrate how to efficiently manage and analyze data using various functions in the `dplyr` package, providing a solid foundation for more advanced data manipulation techniques.

Putting it all together with piping %>%

Let's consolidate everything we've learned into a single, clean line of code using the piping operator `%>%`. This allows us to perform multiple operations in a streamlined manner. Let's give it a go!

```r
# Put all thr steps together with %>%
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

To understand each step, let's break down the code with detailed comments:

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

  # Filter by the desired destinations (keeps only specified rows)
  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 where the Weekend variable has a value of 1
  filter(Weekend == 1) %>%

  # Group by Airline
  group_by(Airline) %>%

  # Summarise to add a variable called NoFlights that counts the number of
  # flights per airline
  summarise(NoFlights = n()) %>%

  # Arrange in descending order
  arrange(desc(NoFlights))
```

Save Your Work

The default is for R to save any output to your working directory. If you want to save this R script in a different location, 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:

```r
# Save the new dataset
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!

This walkthrough has introduced key data management techniques in R using `dplyr`. By mastering these functions and the piping operator, you'll be well-prepared to handle more complex data tasks in future modules. Don't forget to save and annotate your R script for future reference—you'll thank yourself later!