SPEC Lab REU R Resources: Data Management I with tidyverse

Alix Ziff and Benjamin A.T. Graham

Summer 2020

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.

Review: Vectors, Objects, & Operators

We will start by using data from the World Development Indicators. Specifically, we look at different indicators for the energy consumption of all countries in the WDI dataset for the past 25 years. First, install the packages we will need – you only need to do this once.

```
library(foreign)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------- ti
```

```
## v ggplot2 3.3.1     v purrr   0.3.3
## v tibble  2.1.3     v stringr 1.4.0
## v tidyr   1.0.2     v forcats 0.4.0
## v readr   1.3.1
```

```
## Warning: package 'ggplot2' was built under R version 3.6.2
```

```
## -- Conflicts ------------------------------------------------------------------------------------ tidyverse
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(hflights)
```

###Vectors 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)
```

###Sequencing 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
```

###Subset Sequencing Print all the values in world.pop except for 2000

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

The | is the symbol for "or"; == is exactly equal to; != is "not equal"

```
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 simpler?

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

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

##Data Management Let's load the data we will use today. Because it is from STATA (in .dta format), we need to use the `foreign` package.

```
getwd()
library(foreign)
hflights <-tibble::as_tibble(hflights)
```

Let's take a look!

```
str(hflights)
```

Helpful Hint: click the hflights variable in the Environment tab in R Studio to see it in table view.

###Selecting & Subsetting Start by selecting a subset of renamed variables. We're going to drop the variables we don't want by selecting the onces we want to keep.

```
data <- hflights %>%
  select(Month, #keep the month variable
         DayOfWeek, #keep the day of week variable
         Airline = UniqueCarrier, #rename the UniqueCarrier variable to Airline
         Time = ActualElapsedTime, #rename
         Origin:Cancelled) #add only the columns "Origin" through "Cancelled"
```

How does the dropped data look?

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

```
str(data)
```

###Arranging and Changing Data Create an object listing all LA-area aiports. We want to filter for flights with Los Angeles-based destinations.

```
la_airports <- c("LAX", "ONT", "SNA", "BUR", "LGB") #filters for the flights that
#have destinations of LAX, ONT, SNA, BUR, LGB
```

Now, we'll add these airports as an object to our data. This will change the number of rows, not the columns.

```
la_flights <- data %>%
  filter(Dest %in% la_airports)
la_flights_alt <- data %>%
  filter(Dest == c("LAX", "BUR"))
```

Let's take a look. Yikes! Thats a lot of data! How about we use the `head()` command to display the first few rows.

```
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.

###Changing and Re-Arranging Data Create a data frame with all variables that contain the word "Time".

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

###mutate() Use the mutate function to add variables that combine data from other variables

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

###ifelse () Use the ifelse() function to create a Weekend variable that codes whether the flight is on a weekend or not then filter by this variable. 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
```

###summarise() Use the summarise() function to create a column AverageTime with missing values removed.

```
weekday_time <- la_flights %>%
  group_by(DayOfWeek) %>%
  summarise(AverageTime = mean(Time, na.rm = T),
            MaxTaxiTotal = max(TaxiTotal, na.rm = T))
#so they don't get counted in the mean calculation)
```

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 for AverageTime
```

###n() Count all the observations grouped by Airline.

```
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.
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))
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 conta
  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
```

##And we're off! Let's display our new dataframe! Before we finish, make sure to check your working directory, create your own copy of your code using write.csv(), and save this walk-through work locally on your device!