SPEC Lab R Workshop Series: Session 3

Therese Anders

1 Data wrangling with `dplyr`

1.1 Introduction

Data cleaning and reshaping is one of the tasks that we end up spending the most time on. Today, we will be introducing the `dplyr` library. Together with `stringr` (string operations using regular expressions) and `tidyr` these packages offer functionality for virtually any data cleaning and reshaping task in `R`.

For an overview of the most common functions inside `dplyr`, please refer to the RStudio data wrangling cheat sheet https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf.

1.2 Functions in `dplyr`

`dplyr` does not accept tables or vectors, just data frames (similar to `ggplot2`)! `dplyr` uses a strategy called "Split - Apply - Combine". Some of the key functions include:

- `select()`: Subset columns.
- `filter()`: Subset rows.
- `arrange()`: Reorders rows.
- `mutate()`: Add columns to existing data.
- `summarise()`: Summarizing data set.

First, lets dowload the package and call it using the `library()` function.

```
# install.packages("dplyr")
library(dplyr)
```

Today, we will be working with a data set from the `hflights` package. The data set contains all flights from the Houston IAH and HOU airports in 2011. Install the package `hflights`, load it into the library, extract the data frame into a new object called `raw` and inspect the data frame.

NOTE: The `::` operator specifies that we want to use the *object* `hflights` from the *package* `hflights`. In the case below, this explicit programming is not necessary. However, it is useful when functions or objects are contained in multiple packages to avoid confusion. A classic example is the `select()` function that is contained in a number of packages besides `dplyr`.

```
# install.packages("hflights")
library(hflights)
raw <- hflights::hflights
str(raw)
```

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

1.3 Using `select()` and introducing the Piping Operator `%>%`

Using the so-called **piping operator** will make the `R` code faster and more legible, because we are not saving every output in a separate data frame, but passing it on to a new function. First, let's use only a subsample of variables in the data frame, specifically the year of the flight, the airline, as well as the origin airport, the destination, and the distance between the airports.

Notice a couple of things in the code below:

- We can assign the output to a new data set.
- We use the piping operator to connect commands and create a single flow of operations.
- We can use the select function to rename variables.
- Instead of typing each variable, we can select sequences of variables.
- Note that the `everything()` command inside `select()` will select all variables.

```r
data <-  raw %>%
  dplyr::select(Month,
                DayOfWeek,
                Airline = UniqueCarrier, #Renaming the variable
                Time = ActualElapsedTime, #Renaming the variable
                Origin:Cancelled) #Selecting a number of columns.
```

Suppose, we didn't really want to select the `Cancelled` variable. We can use `select()` to drop variables.

```r
data <- data %>%
  dplyr::select(-Cancelled)
```

1.4 Introducting `filter()`

There are a number of key operations when manipulating observations (rows).

- `x < y`
- `x <= y`
- `x == y`
- `x != y`
- `x >= y`
- `x > y`
- `x %in% c(a,b,c)` is `TRUE` if `x` is in the vector `c(a, b, c)`.

Suppose, we wanted to filter all the flights that have their destination in the greater Los Angeles area, specifically Los Angeles (LAX), Ontario (ONT), John Wayne (SNA), Bob Hope (BUR), and Long Beach

(LGB) airports.

```
airports <- c("LAX", "ONT", "SNA", "BUR", "LGB")

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

Caution: The following command does not return the flights to LAX or ONT!

```
head(la_flights)
```

```
##   Month DayOfWeek Airline Time Origin Dest Distance TaxiIn TaxiOut
## 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
```

```
la_flights_alt <- data %>%
  filter(Dest == c("LAX", "ONT"))
head(la_flights_alt)
```

```
##   Month DayOfWeek Airline Time Origin Dest Distance TaxiIn TaxiOut
## 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
```

Why? We are basically returning all values for which the following is `TRUE` (using the correct output of the `la_flights` data frame:

`Dest[1] == LAX`

`Dest[2] == ONT`

`Dest[3] == LAX`

`Dest[4] == ONT` ...

1.5 Helper functions

`dplyr` has a number of helper functions–and that is where the magic lies. These can be used with either `select()` or `filter()`. Here are some useful functions:

- `starts_with()`
- `ends_with()`
- `contains()`
- `matches()`: Every name that matches "X", which can be a regular expression (we will talk about regular expressions in session 5).
- `one_of()`: Every name that appears in x, which should be a character vector.

For example, let's create a data frame with all variables that contain the word "Time".

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

```
##      DepTime ArrTime ActualElapsedTime AirTime
## 5424    1400    1500                60      40
## 5425    1401    1501                60      45
## 5426    1352    1502                70      48
## 5427    1403    1513                70      39
## 5428    1405    1507                62      44
## 5429    1359    1503                64      45
```

1.6 Introducting `mutate()`

Currently, we have two taxi time variables in our data set: `TaxiIn` and `TaxiOut`. I care about total taxi time, and want to add the two together. Also, people hate sitting in planes while it is not in the air. To see how much time is spent taxiing versus flying, we create a variable which measures the proportion of taxi time of total time of flight.

```
la_flights <- la_flights %>%
  mutate(TaxiTotal = TaxiIn + TaxiOut,
         TaxiProp = TaxiTotal/Time)
```

Suppose, I only wanted to fly on weekends. Therefore, I create another variable that codes whether the flight is on a weekend or not and filter the data by this variable. Here, we introduce the `ifelse()` function that is tremendously helpful in data wrangling exercises. The syntax of `ifelse()` is as follows:

`ifelse(condition, if TRUE this, if FALSE this)`.

```
la_flights <- la_flights %>%
  mutate(Weekend = ifelse(DayOfWeek %in% c(1,7), 1, 0)) %>%
  filter(Weekend == 1)
```

1.7 Introducting `summarise()` and `arrange()`

One of the most powerful `dplyr` features is the `summarise()` function, especially in combination with `group_by()`.

First, in a simple example, lets compute the average flight time from Houston to Los Angeles by each day of the week. Also, I want to know what the maximum total taxi time is for each day of the weak. Note, that because there are missing values, we need to tell `R` what to do with them.

```
weekday_time <- la_flights %>%
  group_by(DayOfWeek) %>%
  summarise(AverageTime = mean(Time, na.rm = T),
            MaxTaxiTotal = max(TaxiTotal, na.rm = T))
```

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
```

Now, suppose, I was flying from Houston to Los Angeles, and wanted to know which airline operates the most flights for this route before booking.

Here, we will be using the operator `n()` to tell dplyr to count all the observations for the groups specified in `group_by()`. After computing the result, I would like to arrange the output from highest number of flights, to lowest number.

```r
carriers <- la_flights %>%
  group_by(Airline) %>%
  summarise(NoFlights = n()) %>%
  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
```

So if I want to have the highest selection of flights, I should book with Continental Airlines (at least back in 2011).

1.8 Putting it all together: The power of piping

In this example, I am starting all the way from the original `hflights` data set to demonstrate the power of the piping operator.

```r
carriers_new <-  raw %>%
  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)
```

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

1.9 Saving files

Finally, lets save the output of our code. This time, we would like to save it in the .dta format. Remember to load the `foreign()` library before saving.

```r
library(foreign)
write.dta(carriers_new, "carriers_houston_la.dta")
```