Day 2: Data Visualization in R

FSU Summer Methods School

Therese Anders

5/7/2019

Contents

1 Why data wrangling

This workshop focuses on data visualization. However, in practice, data visualization is only the last part in a long stream of data gathering, cleaning, wrangling, and analysis.

`ggplot2` is the most powerful when we have "tidy" data. There are three rules for tidy data, based on Hadley Wickham's R for Data Science.

1. "Each variable must have its own column."
2. "Each observation must have its own row."
3. "Each value must have its own cell."

If the data are in a tidy format, we can pass separate variables to separate aesthetics and create layered displays of multiple variables. Thus an important component of creating interesting data visualizations is to

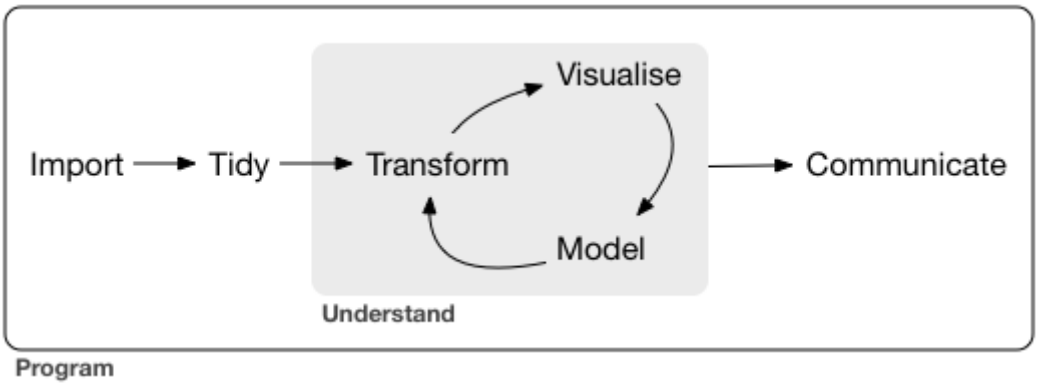


Figure 1: https://d33wubrfki0l68.cloudfront.net/571b056757d68e6df81a3e3853f54d3c76ad6efc/32d37/diagrams/data-science.png

get the data to be in the right format. We will also learn a number of new data visualization tools as part of the data wrangling section, including

- Bar charts
- Error bars on plots

RStudio offers a great Data wrangling cheat sheet you should take a look at.

2 Introduction to `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 ...
```

2.1 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.

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

```
##  [1] "Month"     "DayOfWeek" "DepTime"   "ArrDelay"  "TailNum"
##  [6] "Airline"   "Time"      "Origin"    "Dest"      "Distance"
## [11] "TaxiIn"    "TaxiOut"   "Cancelled"
```

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

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

2.2 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 DepTime ArrDelay TailNum Airline Time Origin Dest
## 1     1         1    1916        2  N76522      CO  227    IAH  LAX
## 2     1         1     747        5  N67134      CO  229    IAH  LAX
## 3     1         1    1433       14  N73283      CO  236    IAH  LAX
## 4     1         1    1750        6  N34282      CO  211    IAH  ONT
## 5     1         1     917       15  N76515      CO  243    IAH  SNA
## 6     1         1    1550        8  N76502      CO  226    IAH  LAX
##   Distance TaxiIn TaxiOut
## 1     1379      8      20
## 2     1379     11      17
## 3     1379     10      27
## 4     1334      5      17
## 5     1347      6      35
## 6     1379     13      15
```

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

```
##   Month DayOfWeek DepTime ArrDelay TailNum Airline Time Origin Dest
## 1     1         1    1916        2  N76522      CO  227    IAH  LAX
## 2     1         1    1433       14  N73283      CO  236    IAH  LAX
## 3     1         1    2107        7  N73270      CO  220    IAH  LAX
## 4     1         1     920        5  N77867      CO  236    IAH  LAX
## 5     1         1    1325       32  N26210      CO  253    IAH  LAX
## 6     1         1    1749        6  N73860      CO  229    IAH  LAX
##   Distance TaxiIn TaxiOut
## 1     1379      8      20
## 2     1379     10      27
## 3     1379      7      12
## 4     1379      8      33
## 5     1379     11      30
## 6     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` ...

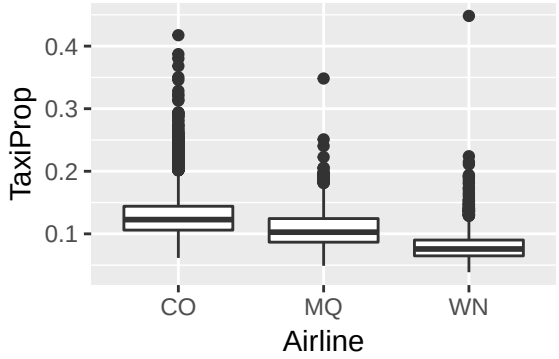
2.3 Introducting `mutate()`

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 <- data %>%
  filter(Dest %in% airports) %>%
  mutate(TaxiTotal = TaxiIn + TaxiOut,
         TaxiProp = TaxiTotal/Time)
```

We can the graph the average proportion of taxi time per airline.

```
library(ggplot2)
ggplot(la_flights,
       aes(x = Airline,
           y = TaxiProp)) +
  geom_boxplot()
```



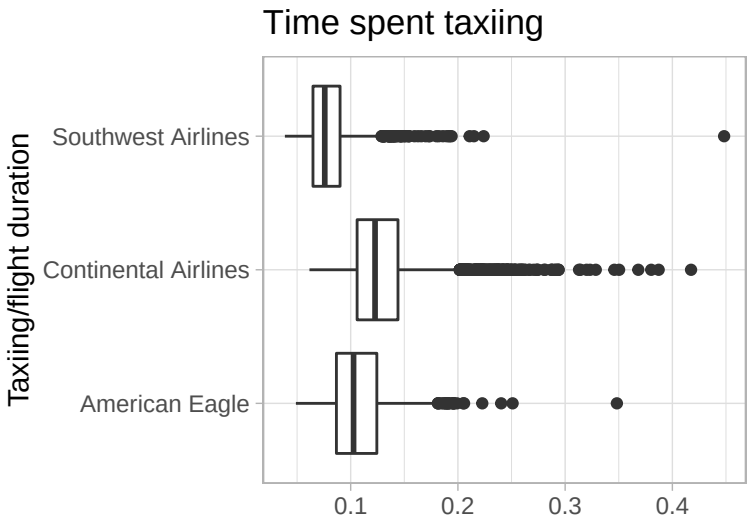
There is only three airlines flying to LA out of Houston. Lets create a new variable with the airline name using the `case_when()` function to make the graph more informative.

```
table(la_flights$Airline)
```

```
##
##   CO   MQ   WN
## 6471  810 1396
```

```
la_flights <- data %>%
  filter(Dest %in% airports) %>%
  mutate(TaxiTotal = TaxiIn + TaxiOut,
         TaxiProp = TaxiTotal/Time,
         AirlineName = case_when(
           Airline == "CO" ~ "Continental Airlines",
           Airline == "MQ" ~ "American Eagle",
           Airline == "WN" ~ "Southwest Airlines"
         ))
ggplot(la_flights,
       aes(x = AirlineName,
           y = TaxiProp)) +
  geom_boxplot() +
  coord_flip() +
  labs(title = "Time spent taxiing",
       x = "Taxiing/flight duration",
```

```
       y = "") +
  theme_light()
```



2.4 Introducting `summarise()` and `arrange()`

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

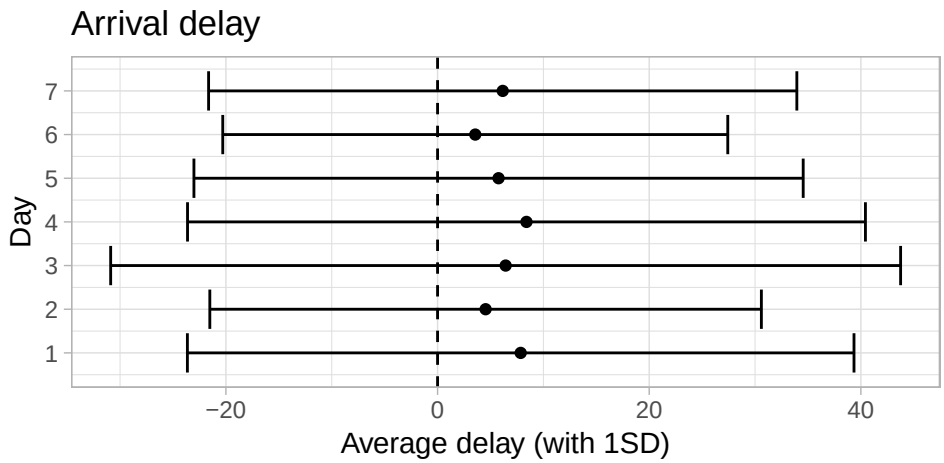
First, in a simple example, lets compute the average delay from Houston to Los Angeles by each day of the week. Note that the arrival delay variable is given in minutes. Also, I want to know what standard deviation of the delay is for each day of the weak. Note, that because there are missing values, we need to tell `R` what to do with them.

```
la_flights_delay <- la_flights %>%
  group_by(DayOfWeek) %>%
  summarise(av_delay = mean(ArrDelay, na.rm = T),
            sd_delay = sd(ArrDelay, na.rm = T))
```

We can use error bars to show the standard deviation of the delay time for each day of the weak. I add a line to denote no delay using the `geom_hline()` aesthetic.

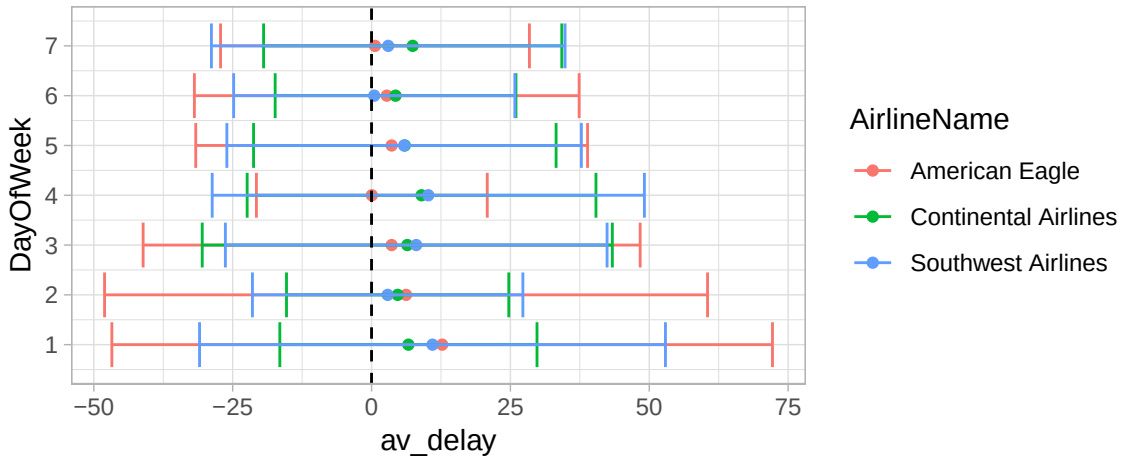
```
ggplot(la_flights_delay,
       aes(x = DayOfWeek,
           y = av_delay,
           ymin = av_delay - sd_delay,
           ymax = av_delay + sd_delay)) +
  geom_point() +
  geom_errorbar() +
  geom_hline(yintercept = 0,
             linetype = "dashed") +

  # Making the graph prettier
  scale_x_continuous(breaks = seq(1,7)) +
  theme_light() +
  labs(y = "Average delay (with 1SD)",
       x = "Day",
       title = "Arrival delay") +
```

```
  coord_flip()
```



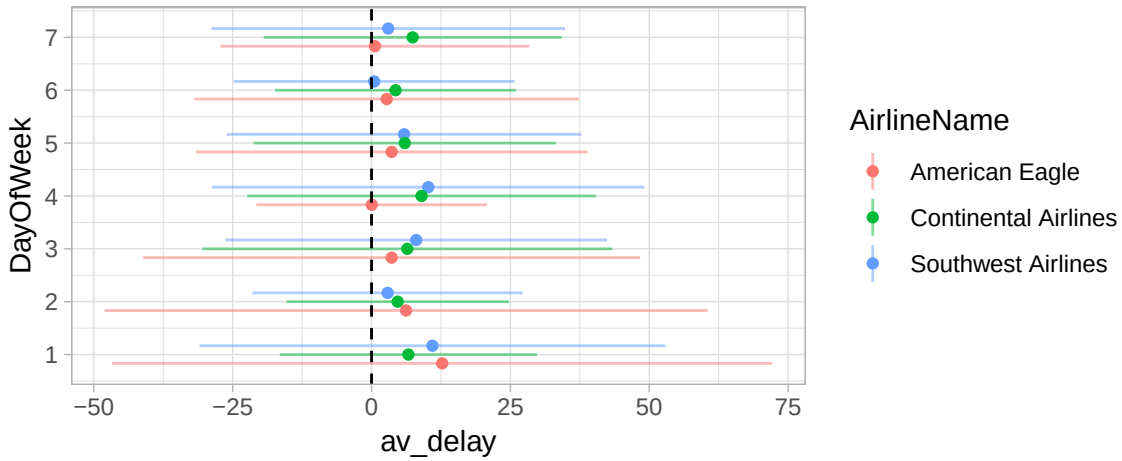
Suppose, I wanted to know whether some airlines have on average shorter arrival delays than others. We can add the airline to the `group_by()` function to compute the mean and standard deviation of arrival delay per day and airline.

```
la_flights_delay_airline <- la_flights %>%
  group_by(DayOfWeek, AirlineName) %>%
  summarise(av_delay = mean(ArrDelay, na.rm = T),
            sd_delay = sd(ArrDelay, na.rm = T))

# Plotting it
ggplot(la_flights_delay_airline,
       aes(x = DayOfWeek,
           y = av_delay,
           ymin = av_delay - sd_delay,
           ymax = av_delay + sd_delay,
           color = AirlineName)) +
  geom_point() +
  geom_errorbar() +
  geom_hline(yintercept = 0,
             linetype = "dashed") +

  # Making graph prettier
  theme_light() +
  coord_flip() +
  scale_x_continuous(breaks = seq(1,7))
```



To de-clutter the graph, I below, I use the `geom_linerange()` aesthetic rather than `geom_errorbar()`. I can use the `position = dodge` command within the `geom_point()` and `geom_linerange()` aesthetic to display the values for each airline next to each other, instead on top of each other. Note that I could have used `position = dodge` with `geom_errorbar()` as well.

```
ggplot(la_flights_delay_airline,
       aes(x = DayOfWeek,
           y = av_delay,
           ymin = av_delay - sd_delay,
           ymax = av_delay + sd_delay,
           color = AirlineName)) +
  geom_point(position = position_dodge(width = 0.5)) +
  geom_linerange(position = position_dodge(width = 0.5),
                 alpha = 0.5) +
  geom_hline(yintercept = 0,
             linetype = "dashed") +

  # Making graph prettier
  theme_light() +
  coord_flip() +
  scale_x_continuous(breaks = seq(1,7))
```



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

```
carriers <- la_flights %>%
  group_by(AirlineName) %>%
  summarise(NoFlights = n()) %>%
  arrange(desc(NoFlights)) #desc() for descending order.
carriers
```

```
## # A tibble: 3 x 2
##   AirlineName          NoFlights
##   <chr>                    <int>
## 1 Continental Airlines      6471
## 2 Southwest Airlines        1396
## 3 American Eagle             810
```

2.5 Introduction to `tidyr`

3 Displaying regression results

san_andreas

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

avenge <- avengers
```