Assignment for April 16

Byung-Deuk Woo

04/16/2019

## Set up “nycflights13” and “tidyverse”

library(nycflights13)  
library(tidyverse)  
nycflights13::flights  
View(flights)

# 5.2 Exercises

# 1.1~1.7

filter(flights, arr\_delay >= 120)  
filter(flights, dest == "IAH" | dest == "HOU")  
filter(flights, carrier %in% c("AA", "DL", "UA"))  
filter(flights, month >= 7, month <= 9)  
filter(flights, arr\_delay > 120, dep\_delay <= 0)  
filter(flights, dep\_delay >= 60, dep\_delay - arr\_delay > 30)  
filter(flights, dep\_time <= 600 | dep\_time == 2400)

# 2

filter(flights, between(month, 7, 9))

# 3

filter(flights, is.na(dep\_time))

# 4

NA ^ 0 == 1 since for all numeric values x^0 = 1. NA | TRUE is TRUE because the value of the missing TRUE or FALSE, x or TRUE is TURE for all values of x. Why is FALSE & NA not missing? It is becuase the value of the missing element matters in NA | FALSE and NA & TURE. It is becuase x multiplied by infinite is undefined.

# 5.3 Exercises

# 1

arrange(flights, desc(is.na(dep\_time)), dep\_time)

# 2

arrange(flights, desc(dep\_delay))

# 3

arrange(flights, air\_time)

# 4

arrange(flights, desc(distance))

# 5.4 Exercises

# 1

select(flights, "dep\_time", "dep\_delay", "arr\_time", "arr\_delay")

# 2

It does not show an error becuase it deals with duplicated variables.

select(flights, year, month, day, year, year)

# 3

It selects variables with a character vetor rather than unquoted variable name arguments. It is helpful because it helps us to generate character vectors with variables names.

vars <- c("year", "month", "day", "dep\_delay", "arr\_delay")  
select(flights, one\_of(vars))

# 4

select(flights, contains("TIME"))

# 5.5 Exercises

# 1

1504 %/% 100  
1504 %% 100  
1504 %/% 100 \* 60 + 1504 %% 100  
  
flights\_times <- mutate(flights,  
 dep\_time\_mins = (dep\_time %/% 100 \* 60 + dep\_time %% 100) %% 1440,  
 sched\_dep\_time\_mins = (sched\_dep\_time %/% 100 \* 60 +  
 sched\_dep\_time %% 100) %% 1440  
)  
  
select(  
 flights\_times, dep\_time, dep\_time\_mins, sched\_dep\_time,  
 sched\_dep\_time\_mins  
)

# 2

air\_time will be equal to arr\_time - dep\_time

flights\_airtime <-  
 mutate(flights,  
 dep\_time = (dep\_time %/% 100 \* 60 + dep\_time %% 100) %% 1440,  
 arr\_time = (arr\_time %/% 100 \* 60 + arr\_time %% 100) %% 1440,  
 air\_time\_diff = air\_time - arr\_time + dep\_time  
 )  
  
nrow(filter(flights\_airtime, air\_time\_diff != 0))

The above commands show that arr\_time != arr\_time - dep\_time. It is because time zone. Moreover, it is related to the codings of those three variables.

# 3

It can be expected that dep\_time - sched\_dep\_time = dep\_delay.

# 4

rankme <- tibble(  
 x = c(10, 5, 1, 5, 5)  
)  
  
arrange(rankme, x)

# 5.6 Exercises

# 1

The one thing that we have to pick the most important variable is related to our theory. If we worry about airplane crash or missing, arrival delay is more important than departure delay.

# 2

not\_cancelled <- flights %>%  
 filter(!is.na(dep\_delay), !is.na(arr\_delay))  
  
not\_cancelled %>%  
 count(dest)  
   
not\_cancelled %>%  
 group\_by(dest) %>%  
 summarise(n = length(dest))  
  
not\_cancelled %>%  
 count(tailnum, wt = distance)  
  
not\_cancelled %>%  
 group\_by(tailnum) %>%  
 summarise(n = sum(distance))  
  
not\_cancelled %>%  
 group\_by(tailnum) %>%  
 tally(distance)

# 3

For example, flight cannot arrive if it did not depart. The most important one is arr\_delay because there are severe bad senarios.

# 4

cancelled\_delayed <-  
 flights %>%  
 mutate(cancelled = (is.na(arr\_delay) | is.na(dep\_delay))) %>%  
 group\_by(month, day) %>%  
 summarise(  
 prop\_cancelled = mean(cancelled),  
 avg\_dep\_delay = mean(dep\_delay, na.rm = TRUE)  
 )  
  
ggplot(cancelled\_delayed, aes(x = avg\_dep\_delay, prop\_cancelled)) +  
 geom\_point() +  
 geom\_smooth()

Yes, there is a pattern. When avg\_dep\_delay increases prop\_cancelled increases.

# 5.7.1 Exercises

# 1

Mutate and filtering functions operate within each group rather than over the entire data frame.

# 2

flights %>%  
 filter(!is.na(tailnum)) %>%  
 mutate(on\_time = !is.na(arr\_time) & (arr\_delay <= 0)) %>%  
 select(tailnum, on\_time, arr\_time, arr\_delay) %>%  
 group\_by(tailnum) %>%  
 summarise(on\_time = mean(on\_time), n = n()) %>%  
 filter(min\_rank(on\_time) == 1)  
  
quantile(count(flights, tailnum)$n)  
  
flights %>%  
 filter(!is.na(tailnum)) %>%  
 mutate(on\_time = !is.na(arr\_time) & (arr\_delay <= 0)) %>%  
 select(tailnum, on\_time, arr\_time, arr\_delay) %>%  
 group\_by(tailnum) %>%  
 summarise(on\_time = mean(on\_time), n = n()) %>%  
 filter(n >= 20) %>%  
 filter(min\_rank(on\_time) == 1)  
  
flights %>%  
 group\_by(tailnum) %>%  
 summarise(arr\_delay = mean(arr\_delay), n = n()) %>%  
 filter(n >= 20) %>%  
 filter(min\_rank(desc(arr\_delay)) == 1)

The above commands deal with “proportion of lights not delayed or cancelled” and “mean arrival delay”.

# 3

flights %>%  
 group\_by(hour) %>%  
 summarise(arr\_delay = mean(arr\_delay, na.rm = TRUE)) %>%  
 arrange(arr\_delay)

Through the above command, we can see which time has lowest expected delay.

# 4

flights %>%  
 filter(arr\_delay > 0) %>%  
 group\_by(dest) %>%  
 mutate(  
 arr\_delay\_total = sum(arr\_delay),  
 arr\_delay\_prop = arr\_delay / arr\_delay\_total  
 ) %>%  
 select(  
 dest, month, day, dep\_time, carrier, flight,  
 arr\_delay, arr\_delay\_prop  
 ) %>%  
 arrange(dest, desc(arr\_delay\_prop))

# 12.2 Exercises

# 1

table1  
table2  
table3  
table4a  
table4b

Table 1 shows the country year data. Table 2 shows the country year variable data. Table 3 shows the country year and rate (cases/population) data. Table 4a shows the cases. Table 4b shows the values of population.

# 2

t2\_cases <- filter(table2, type == "cases") %>%  
 rename(cases = count) %>%  
 arrange(country, year)  
t2\_population <- filter(table2, type == "population") %>%  
 rename(population = count) %>%  
 arrange(country, year)  
  
t2\_cases\_per\_cap <- tibble(  
 year = t2\_cases$year,  
 country = t2\_cases$country,  
 cases = t2\_cases$cases,  
 population = t2\_population$population  
) %>%  
 mutate(cases\_per\_cap = (cases / population) \* 10000) %>%  
 select(country, year, cases\_per\_cap)  
  
2\_cases\_per\_cap <- t2\_cases\_per\_cap %>%  
 mutate(type = "cases\_per\_cap") %>%  
 rename(count = cases\_per\_cap)  
  
bind\_rows(table2, t2\_cases\_per\_cap) %>%  
 arrange(country, year, type, count)  
  
table4c <-  
 tibble(  
 country = table4a$country,  
 `1999` = table4a[["1999"]] / table4b[["1999"]] \* 10000,  
 `2000` = table4a[["2000"]] / table4b[["2000"]] \* 10000  
 )

The table 4c shows the cases per capita.

# 3

table2 %>%  
 filter(type == "cases") %>%  
 ggplot(aes(year, count)) +  
 geom\_line(aes(group = country), colour = "grey50") +  
 geom\_point(aes(colour = country)) +  
 scale\_x\_continuous(breaks = unique(table2$year)) +  
 ylab("cases")

# 12.3 Exercises

# 1

It is because when we use gather() command, it discards the original column types. However, spread() function does not know the original data types of the variables.

# 2

It fails becuase the column names of 1999 and 2000 are not non-syntactic variable names.

# 3

It fails because name and key columns do not uniquely identify rows.

people <- tribble(  
 ~name, ~key, ~value,  
 #-----------------|--------|------  
 "Phillip Woods", "age", 45,  
 "Phillip Woods", "height", 186,  
 "Phillip Woods", "age", 50,  
 "Jessica Cordero", "age", 37,  
 "Jessica Cordero", "height", 156  
)  
glimpse(people)  
spread(people, key, value)  
  
people2 <- people %>%  
 group\_by(name, key) %>%  
 mutate(obs = row\_number())  
people2  
spread(people2, key, value)  
people %>%  
 distinct(name, key, .keep\_all = TRUE) %>%  
 spread(key, value)

The above commands can solve the problem.

# 4

In order to tidy the preg tibble, we have to use gather(). Sex, pregnant, and count are the variables in the data.

# 12.4 Exercises

# 1

extra tells separate() what to do if there are too many pieces, and the fill tells it what to do if there aren’t enought. separate() basically drops extra values with a warning. Below are the examples.

tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%  
 separate(x, c("one", "two", "three"))  
  
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%  
 separate(x, c("one", "two", "three"), extra = "drop")  
  
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%  
 separate(x, c("one", "two", "three"), extra = "merge")  
  
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%  
 separate(x, c("one", "two", "three"))  
  
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%  
 separate(x, c("one", "two", "three"), fill = "right")  
  
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%  
 separate(x, c("one", "two", "three"), fill = "left")

# 2

remove discards input columns in the result data frame.

# 3

separate function splits a column into multiple columns by separator. extract uses a regular expression to specify groups in character vector and split that single character vector into multiple columns.

# 12.5 Exercises

# 1

Fill in spread() sets the value to replace NAs. On the other hand, fill in complete() sets a value to replace NAs but it is named lists. Therefore, it allows for different values for different variables.

# 2

direction commands with fill() determines whether NA values should be replace by the previous non-missing value.

# 12.6 Exercises

# 1

There are zeros and there are missing values too. We can check this by using codes below.

who1 %>%  
 filter(cases == 0) %>%  
 nrow()  
  
gather(who, new\_sp\_m014:newrel\_f65, key = "key", value = "cases") %>%  
 group\_by(country, year) %>%  
 mutate(prop\_missing = sum(is.na(cases)) / n()) %>%  
 filter(prop\_missing > 0, prop\_missing < 1)  
  
nrow(who)  
who %>%  
 complete(country, year) %>%  
 nrow()

Moreover, we can use the below commands for checking what implicit missing values are.

anti\_join(complete(who, country, year), who, by = c("country", "year")) %>%  
 select(country, year) %>%  
 group\_by(country) %>%  
 # so I can make better sense of the years  
 summarise(min\_year = min(year), max\_year = max(year))

0 means no cases in this dataset. Explicit missing values are used to missing data for country and year. Implicit missing values represent missing data when a country does not exist in that year.

# 2

who3a <- who1 %>%  
 separate(key, c("new", "type", "sexage"), sep = "\_")  
filter(who3a, new == "newrel") %>% head()

When we neglect the mutate() step, there will be a warning message “too few values”.

# 3

select(who3, country, iso2, iso3) %>%  
 distinct() %>%  
 group\_by(country) %>%  
 filter(n() > 1)

We can use the above commands for confirming the claim.

# 4

who5 %>%  
 group\_by(country, year, sex) %>%  
 filter(year > 1995) %>%  
 summarise(cases = sum(cases)) %>%  
 unite(country\_sex, country, sex, remove = FALSE) %>%  
 ggplot(aes(x = year, y = cases, group = country\_sex, colour = sex)) +  
 geom\_line()

The above commands makes me possible to see the visualization of the data with time, sex, and cases.

For doing this homework, I refer the follow site. “<http://jrnold.github.io>”