Midterm Exam

# 12.2.1 Exercises

## (1) 12.2.1 Exercise 2

Compute the rate for table2, and table4a + table4b. You will need to perform four operations:

#Using table2  
  
#Step 1 - Extract the number of TB cases per country per year.  
step1 <- table2 %>% spread(key = "type", value = "count") %>% select(country, year, cases)  
step1

## # A tibble: 6 × 3  
## country year cases  
## \* <chr> <int> <int>  
## 1 Afghanistan 1999 745  
## 2 Afghanistan 2000 2666  
## 3 Brazil 1999 37737  
## 4 Brazil 2000 80488  
## 5 China 1999 212258  
## 6 China 2000 213766

#Step2 - Extract the matching population per country per year.  
step2 <- table2 %>% spread(key = "type", value = "count") %>% select(country, year, population)  
step2

## # A tibble: 6 × 3  
## country year population  
## \* <chr> <int> <int>  
## 1 Afghanistan 1999 19987071  
## 2 Afghanistan 2000 20595360  
## 3 Brazil 1999 172006362  
## 4 Brazil 2000 174504898  
## 5 China 1999 1272915272  
## 6 China 2000 1280428583

#Step3 - Divide cases by population, and multiply by 10000.  
step3 <- rate <- (select(step1, rate = cases) / select(step2, population) \* 10000)  
step3

## rate  
## 1 0.372741  
## 2 1.294466  
## 3 2.193930  
## 4 4.612363  
## 5 1.667495  
## 6 1.669488

#Final - Store back in the appropriate place.  
final <- step1  
final["country"] <- step1["country"]  
final["year"] <- step1["year"]  
final["rate"] <- step3["rate"]  
final

## # A tibble: 6 × 4  
## country year cases rate  
## \* <chr> <int> <int> <dbl>  
## 1 Afghanistan 1999 745 0.372741  
## 2 Afghanistan 2000 2666 1.294466  
## 3 Brazil 1999 37737 2.193930  
## 4 Brazil 2000 80488 4.612363  
## 5 China 1999 212258 1.667495  
## 6 China 2000 213766 1.669488

#Using table4a + table4b  
  
#Step 1 - Extract the number of TB cases per country per year.  
step1 <- table4a %>% gather(`1999`, `2000`, key = "year", value = "cases")  
step1

## # A tibble: 6 × 3  
## country year cases  
## <chr> <chr> <int>  
## 1 Afghanistan 1999 745  
## 2 Brazil 1999 37737  
## 3 China 1999 212258  
## 4 Afghanistan 2000 2666  
## 5 Brazil 2000 80488  
## 6 China 2000 213766

#Step 2 - Extract the matching population per country per year.  
step2 <- table4b %>% gather(`1999`, `2000`, key = "year", value = "population")  
step2

## # A tibble: 6 × 3  
## country year population  
## <chr> <chr> <int>  
## 1 Afghanistan 1999 19987071  
## 2 Brazil 1999 172006362  
## 3 China 1999 1272915272  
## 4 Afghanistan 2000 20595360  
## 5 Brazil 2000 174504898  
## 6 China 2000 1280428583

#Step 3 - Divide cases by population, and multiply by 10000.  
step3 <- rate <- (select(step1, rate = cases) / select(step2, population) \* 10000)  
step3

## rate  
## 1 0.372741  
## 2 2.193930  
## 3 1.667495  
## 4 1.294466  
## 5 4.612363  
## 6 1.669488

#Final - Store back in the appropriate place.  
final["country"] <- step1["country"]  
final["year"] <- step1["year"]  
final["rate"] <- step3["rate"]  
final

## # A tibble: 6 × 4  
## country year cases rate  
## \* <chr> <chr> <int> <dbl>  
## 1 Afghanistan 1999 745 0.372741  
## 2 Brazil 1999 2666 2.193930  
## 3 China 1999 37737 1.667495  
## 4 Afghanistan 2000 80488 1.294466  
## 5 Brazil 2000 212258 4.612363  
## 6 China 2000 213766 1.669488

Which representation is easiest to work with? Which is hardest? Why?

Which is hardest is a matter of opinion in this case. I would say both are about equally as difficult. One requires the use of gather to complete whereas the other requires the use of spread to complete. I would rank both at an equal difficulty based on this observation. The two functions gather and spread are close to opposites. We will address why they are not quite symmetrical in the next question.

# 12.3.3 Exercises

## (2) 12.3.3 Exercise 1

Why are gather() and spread() not perfectly symmetrical? Carefully consider the following example:

stocks <- tibble(  
 year = c(2015, 2015, 2016, 2016),  
 half = c( 1, 2, 1, 2),  
 return = c(1.88, 0.59, 0.92, 0.17)  
)  
stocks %>%   
 spread(year, return) %>%   
 gather("year", "return", `2015`:`2016`)

## # A tibble: 4 × 3  
## half year return  
## <dbl> <chr> <dbl>  
## 1 1 2015 1.88  
## 2 2 2015 0.59  
## 3 1 2016 0.92  
## 4 2 2016 0.17

(Hint: look at the variable types and think about column names.)

Both spread() and gather() have a convert argument. What does it do?

datatypes seem to persist with spread probably because they are using the datatypes that are already present whereas they do not with gather probably because it is creating a new column that will need a datatype. With gather, the column names can also be set to the name that the user prefers because they are creating a new column, and with spread, it is automatically set to the values of the row specified to be spread.

The convert argument automatically converts it to the appropriate value (logical, integer, numeric, complex, or factor).

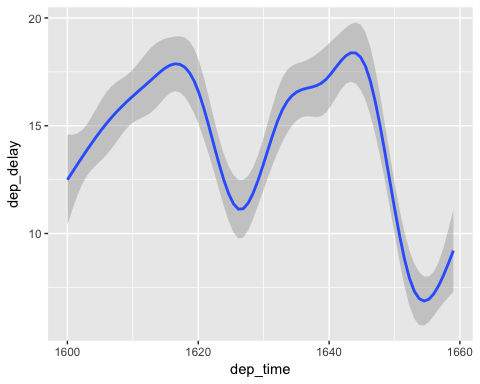
# 16.3.4 Exercises

## (3) 16.3.4 Exercise 2

Compare dep\_time, sched\_dep\_time and dep\_delay. I recommend looking at the distributions over an hour. Are they consistent? Explain your findings.

flights %>% filter(dep\_time >= 1600 & dep\_time <= 1660) %>%  
ggplot(aes(x=dep\_time, y=dep\_delay)) + geom\_smooth()

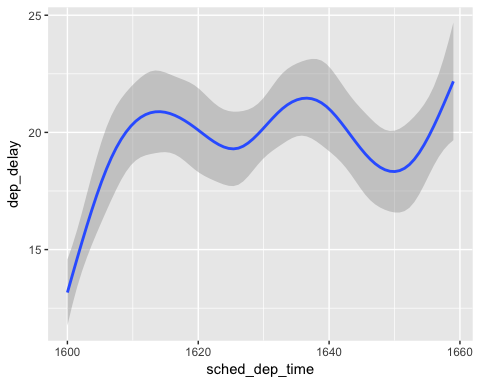
## `geom\_smooth()` using method = 'gam'



flights %>% filter(sched\_dep\_time >= 1600 & sched\_dep\_time <= 1660) %>%  
ggplot(aes(x=sched\_dep\_time, y=dep\_delay)) + geom\_smooth()

## `geom\_smooth()` using method = 'gam'

## Warning: Removed 840 rows containing non-finite values (stat\_smooth).



As we can see these graphs are similar. Schedule departure time relates to departure delay in a similar but not quite as radical fashion as actual departure times over the 4PM - 5PM time frame. This is probably because as scheduled flights become late, these flights will leave later and exaggerate the departure delay of making the graph more pronounced for departure time and departure delay when compared to scheduled departure time and departure delay.

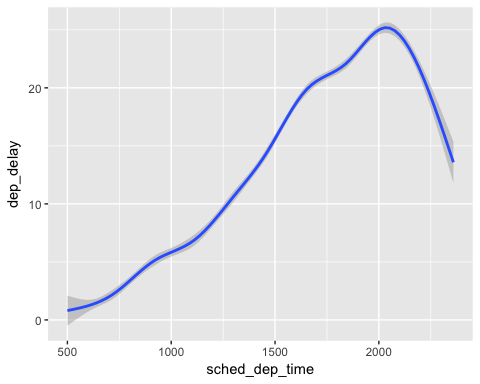
## (4) 16.3.4 Exercise 4

How does the average departure delay change over the course of a day? Should you use dep\_time or sched\_dep\_time? Why?

ggplot(flights, aes(x=sched\_dep\_time, y=dep\_delay)) + geom\_smooth()

## `geom\_smooth()` using method = 'gam'

## Warning: Removed 8255 rows containing non-finite values (stat\_smooth).

 Departure delay average changes over the course of the day rising steadily from scheduled departure times starting at 5AM until 8PM. Then departure delays start declining at a similar pace until midnight. We use sched\_dep\_delay because this is the time that flights are supposed to leave so it does not involve the dep\_delay that has already happened in dep\_time column. Flights that are supposed to leave at the times above had the departure delays depicted above. We can actually calculate departure delay by taking the absolute value of the subtraction of scheduled departure time from departure time. This, again, is why we use scheduled departure time rather than departure time in this example.

## (5) 16.3.4 Exercise 5

On what day of the week should you leave if you want to minimize the chance of a departure delay?

devtools::install\_github("hadley/lubridate")

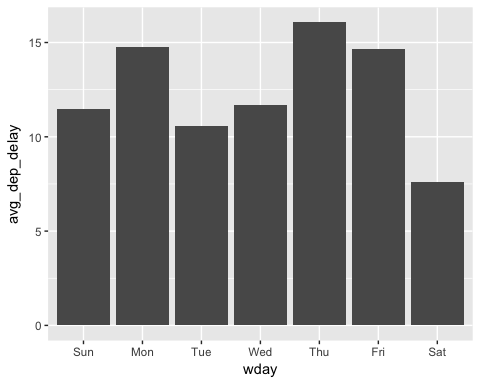
## Skipping install of 'lubridate' from a github remote, the SHA1 (c5ddda2a) has not changed since last install.  
## Use `force = TRUE` to force installation

library(lubridate)

##   
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':  
##   
## date

# transform times into date-time format  
make\_datetime\_100 <- function(year, month, day, time) {  
 make\_datetime(year, month, day, time %/% 100, time %% 100)  
}  
  
flights\_dt <- flights %>%   
 # Remove flights that never departed or never arrived  
 filter(!is.na(dep\_time), !is.na(arr\_time)) %>%   
 # Convert times into date-times  
 mutate(  
 dep\_time = make\_datetime\_100(year, month, day, dep\_time)  
 )  
  
#group by week day and calculate average departure delay for each week day.  
flights\_dt %>%   
 mutate(wday = wday(dep\_time, label = TRUE)) %>%   
 group\_by(wday) %>%  
 summarise(avg\_dep\_delay = mean(dep\_delay, na.rm=TRUE), n = n()) %>%  
 ggplot(aes(x = wday, y=avg\_dep\_delay)) +  
 geom\_col()



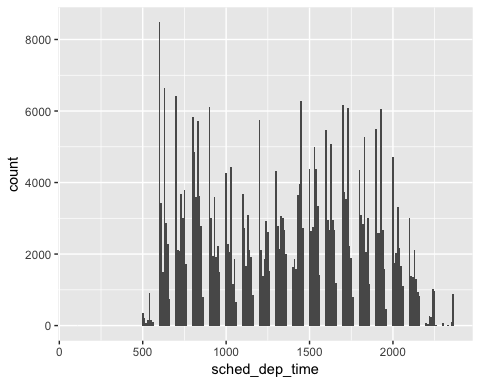
Saturday would be the day that one would want to travel to minimize the chance of departure delay. There are also less flights on the weekend which may be playing a factor in this.

## (6) 16.3.4 Exercise 6

What makes the distribution of diamondssched\_dep\_time similar?

In both of these distributions across their respective datasets, carat and shced\_dep\_time have the similarity of being concentrated at certain intervals. Carat seems to be concentrated at half carat and whole carat intervals as well as below .3 carat. Scheduled departure times seems to be concentrated at intervals of every hour mark as well as slightly less at half hour marks and even less and quarter hour marks. The spaces in the flights graph represent the times that are impossible such as 1670, since this would be the same as 1710. This shows the flight distribution well since the first new line from left to right after a break will be on the hour and is also some of the tallest since that is where most of the flights are distributed.

flights %>% ggplot(aes(sched\_dep\_time)) + geom\_histogram(binwidth=10)



diamonds %>% ggplot(aes(carat)) + geom\_histogram(binwidth = .1)

