Midterm Exam: Preston Robertson

# Fixing the flights data  
  
make\_datetime\_100 <- function(year, month, day, time) {  
 make\_datetime(year, month, day, time %/% 100, time %% 100)  
}  
  
  
flights\_fixed <- flights %>%  
 filter(!is.na(dep\_time), !is.na(arr\_time)) %>%  
 mutate(  
 dep\_time = make\_datetime\_100(year, month, day, dep\_time),  
 arr\_time = make\_datetime\_100(year, month, day, arr\_time),  
 sched\_dep\_time = make\_datetime\_100(year, month, day, sched\_dep\_time),  
 sched\_arr\_time = make\_datetime\_100(year, month, day, sched\_arr\_time)  
 ) %>%  
 select(origin, dest, ends\_with("delay"), ends\_with("time"))

All of these problems are from the *R for Data Science* book. The section numbers (e.g., “3.2.4 Exercises”) refer to sections in this book.

When solving these problems, you are allowed to use any method from the book or class, even if that method wasn’t yet covered when the exercise was presented in the book.

Because this is an exam, you need to do the work by yourself. Do not collaborate on this exam.

This exam is out of 60 points.

## 12.2.1 Exercises

### (1) 12.2.1 Exercise 2 (10 pts)

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

table1

# A tibble: 6 x 4  
 country year cases population  
 <chr> <int> <int> <int>  
1 Afghanistan 1999 745 19987071  
2 Afghanistan 2000 2666 20595360  
3 Brazil 1999 37737 172006362  
4 Brazil 2000 80488 174504898  
5 China 1999 212258 1272915272  
6 China 2000 213766 1280428583

table2 #Cases and Population are in separate rows

# A tibble: 12 x 4  
 country year type count  
 <chr> <int> <chr> <int>  
 1 Afghanistan 1999 cases 745  
 2 Afghanistan 1999 population 19987071  
 3 Afghanistan 2000 cases 2666  
 4 Afghanistan 2000 population 20595360  
 5 Brazil 1999 cases 37737  
 6 Brazil 1999 population 172006362  
 7 Brazil 2000 cases 80488  
 8 Brazil 2000 population 174504898  
 9 China 1999 cases 212258  
10 China 1999 population 1272915272  
11 China 2000 cases 213766  
12 China 2000 population 1280428583

table3

# A tibble: 6 x 3  
 country year rate   
\* <chr> <int> <chr>   
1 Afghanistan 1999 745/19987071   
2 Afghanistan 2000 2666/20595360   
3 Brazil 1999 37737/172006362   
4 Brazil 2000 80488/174504898   
5 China 1999 212258/1272915272  
6 China 2000 213766/1280428583

table4a # Cases and Population are in different tables

# A tibble: 3 x 3  
 country `1999` `2000`  
\* <chr> <int> <int>  
1 Afghanistan 745 2666  
2 Brazil 37737 80488  
3 China 212258 213766

table4b

# A tibble: 3 x 3  
 country `1999` `2000`  
\* <chr> <int> <int>  
1 Afghanistan 19987071 20595360  
2 Brazil 172006362 174504898  
3 China 1272915272 1280428583

1. Extract the number of TB cases per country per year.
2. Extract the matching population per country per year.
3. Divide cases by population, and multiply by 1000.
4. Store back in the appropriate place.

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

# Table 2  
  
# First step take out old values (parts 1 and 2)  
t2\_case <- filter(table2, type == "cases") %>%  
 rename(cases = count) %>%  
 arrange(country, year) # Keeping order  
t2\_pop <- filter(table2, type == "population") %>%  
 rename(population = count) %>%  
 arrange(country, year)  
  
# Step 2 calculate the rate  
t2\_rate <- tibble(year = t2\_case$year, country = t2\_case$country,cases = t2\_case$cases,  
 population = t2\_pop$population) %>%  
 mutate(rate = (cases / population) \* 1000) %>% #Doing part 3  
 select(country, year, rate)  
  
# Step 3 Combine the rate with table 2  
t2\_rate <- t2\_rate %>%  
 mutate(type = "rate") %>%  
 rename(count = rate) #So rate will go to the proper column  
bind\_rows(table2, t2\_rate) %>%  
 arrange(country, year, type, count)

# A tibble: 18 x 4  
 country year type count  
 <chr> <int> <chr> <dbl>  
 1 Afghanistan 1999 cases 7.45e+2  
 2 Afghanistan 1999 population 2.00e+7  
 3 Afghanistan 1999 rate 3.73e-2  
 4 Afghanistan 2000 cases 2.67e+3  
 5 Afghanistan 2000 population 2.06e+7  
 6 Afghanistan 2000 rate 1.29e-1  
 7 Brazil 1999 cases 3.77e+4  
 8 Brazil 1999 population 1.72e+8  
 9 Brazil 1999 rate 2.19e-1  
10 Brazil 2000 cases 8.05e+4  
11 Brazil 2000 population 1.75e+8  
12 Brazil 2000 rate 4.61e-1  
13 China 1999 cases 2.12e+5  
14 China 1999 population 1.27e+9  
15 China 1999 rate 1.67e-1  
16 China 2000 cases 2.14e+5  
17 China 2000 population 1.28e+9  
18 China 2000 rate 1.67e-1

# Table 4  
  
# Table 4 is already split so just make a new table and skip step 1 and step 3 above  
table4 <- tibble(  
 country = table4a$country,  
 `1999` = table4a[["1999"]] / table4b[["1999"]] \* 1000, # First value is table cases of 1999 and second is population of `1999  
 `2000` = table4a[["2000"]] / table4b[["2000"]] \* 1000  
 )  
  
table4

# A tibble: 3 x 3  
 country `1999` `2000`  
 <chr> <dbl> <dbl>  
1 Afghanistan 0.0373 0.129  
2 Brazil 0.219 0.461  
3 China 0.167 0.167

# Which is harder to work with? Easily table2 for me. The data sets that I normally work with do not work like a branching system where the previous columns work as dictionaries for the value column. It took me forever to think to take out the values and then do the math. It also was difficult to find out to plug them back in, I had to "jerry-rig" by just saying rate is count. Table4 allowed for simple operations already plugged into r. It is still inconvenient to call multiple different tables, but way more manageable than table2 for me.

## 12.3.3 Exercises

### (2) 12.3.3 Exercise 1 (10 pts)

Why are pivot\_longer() and pivot\_wider() not perfectly symmetrical? Carefully consider the following example:

(Hint: look at the variable types and think about column *names*.) pivot\_longer() has a names\_transform argument. What does it do?

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

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

stocks\_wrong <- stocks %>%   
 pivot\_wider(names\_from = year, values\_from = return) %>%   
 pivot\_longer(`2015`:`2016`, names\_to = "year", values\_to = "return")  
  
stocks\_wrong

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

# The pivot\_longer() function turned the year column from a numeric value (double) to a character value. This is because the 'names\_to' function inside pivot\_longer assumes all values to be a character.   
  
# The 'names\_transform' argument allows the user to change the character type from the 'names\_to' argument back to a numeric data type.  
  
stocks %>%  
 pivot\_wider(names\_from = year, values\_from = return)%>%  
 pivot\_longer(`2015`:`2016`, names\_to = "year", values\_to = "return", names\_transform = list(year = as.numeric))

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

## 16.3.4 Exercises

### (3) 16.3.4 Exercise 2 (10 pts)

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

Question3.1 <- flights\_fixed %>% select(contains('dep')) %>%  
 mutate(calculated\_delay = as.numeric(dep\_time - sched\_dep\_time) / 60) %>% #Calculating departure delay manually  
 filter(dep\_delay != calculated\_delay) #Filtering all the inconsistencies  
  
Question3.1

# A tibble: 1,205 x 4  
 dep\_delay dep\_time sched\_dep\_time calculated\_delay  
 <dbl> <dttm> <dttm> <dbl>  
 1 853 2013-01-01 08:48:00 2013-01-01 18:35:00 -587  
 2 43 2013-01-02 00:42:00 2013-01-02 23:59:00 -1397  
 3 156 2013-01-02 01:26:00 2013-01-02 22:50:00 -1284  
 4 33 2013-01-03 00:32:00 2013-01-03 23:59:00 -1407  
 5 185 2013-01-03 00:50:00 2013-01-03 21:45:00 -1255  
 6 156 2013-01-03 02:35:00 2013-01-03 23:59:00 -1284  
 7 26 2013-01-04 00:25:00 2013-01-04 23:59:00 -1414  
 8 141 2013-01-04 01:06:00 2013-01-04 22:45:00 -1299  
 9 15 2013-01-05 00:14:00 2013-01-05 23:59:00 -1425  
10 127 2013-01-05 00:37:00 2013-01-05 22:30:00 -1313  
# ... with 1,195 more rows

flights\_fixed %>% select(contains('dep')) %>%  
 mutate(cal\_delay = as.numeric(dep\_time - sched\_dep\_time) / 60) %>%  
 filter(dep\_delay != cal\_delay) %>%  
 mutate(dep\_time = update(dep\_time, mday = mday(dep\_time) + 1)) %>% #Adding one day to all of the previous table  
 mutate(cal\_delay = as.numeric(dep\_time - sched\_dep\_time)) %>% # Re-doing above process  
 filter(dep\_delay != cal\_delay)

# A tibble: 0 x 4  
# ... with 4 variables: dep\_delay <dbl>, dep\_time <dttm>,  
# sched\_dep\_time <dttm>, cal\_delay <dbl>

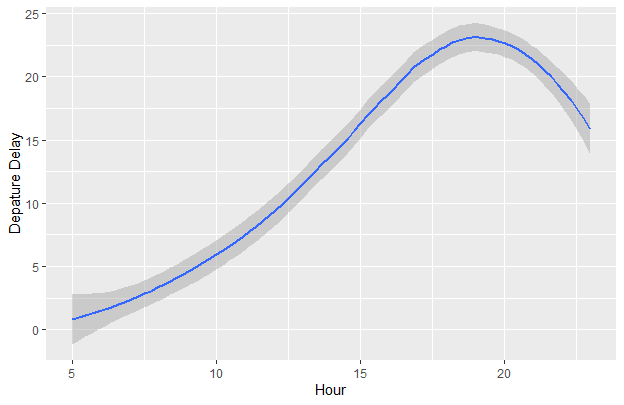
# Looking at the below table, you can see that the calculated delay shows that some flights actually leave earlier and were not delayed. However that was probably a mistake on the people recording the flights. To fix this we are going to add a day onto negative values

### (4) 16.3.4 Exercise 4 (10 pts)

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

# We should you scheduled departure time since that is what the delay is based on.  
# Looking at the graph, we can see a clear trend that the farther the day goes until around 7pm the delays add up. The delays are probably due to a mix of busyness during those times, plus other flights being delayed have a ripple effect on the airport.  
  
flights\_fixed %>%  
 mutate(hour = hour(sched\_dep\_time)) %>%  
 group\_by(hour) %>%  
 summarize(avg\_dep\_delay = mean(dep\_delay)) %>%  
 ggplot(mapping = aes(x = hour, y = avg\_dep\_delay)) +  
 geom\_smooth() +  
 labs(y = "Depature Delay", x = "Hour")

`geom\_smooth()` using method = 'loess' and formula 'y ~ x'

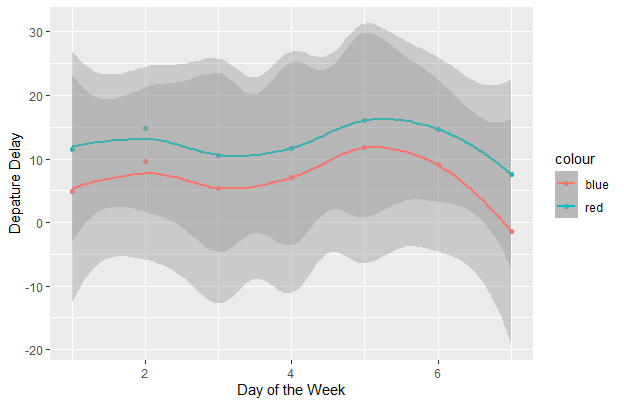


### (5) 16.3.4 Exercise 5 (10 pts)

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

flights\_days <- flights\_fixed %>%  
 mutate(days = wday(sched\_dep\_time)) %>% # Grouping into day of the week, Sunday is 1  
 group\_by(days) %>%  
 summarise(  
 dep\_delay = mean(dep\_delay), #Averaging all values into a single one.  
 arr\_delay = mean(arr\_delay, na.rm = TRUE)  
 )   
  
  
   
ggplot(flights\_days) +  
 geom\_point(mapping = aes(x = days, y = dep\_delay, colour = "red")) +  
 geom\_smooth(mapping = aes(x = days, y = dep\_delay, colour = "red")) +  
 geom\_point(mapping = aes(x = days, y = arr\_delay, colour = "blue")) +  
 geom\_smooth(mapping = aes(x = days, y = arr\_delay, colour = "blue"))+  
 labs(y = "Depature Delay", x = "Day of the Week")

`geom\_smooth()` using method = 'loess' and formula 'y ~ x'  
`geom\_smooth()` using method = 'loess' and formula 'y ~ x'

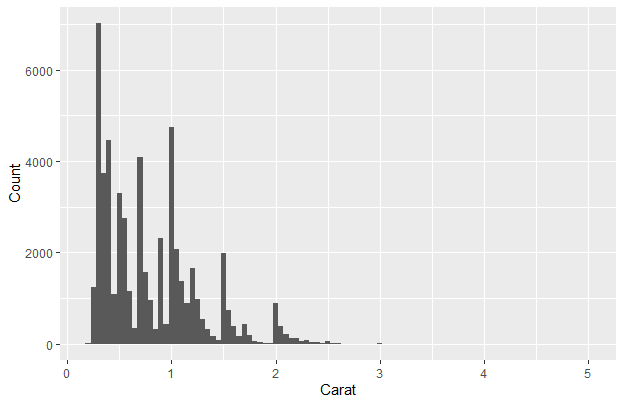


# Saturdays look like the bast day to leave if you are minimizing delays. This is due to lower average delays and even average arriving early on Saturdays.

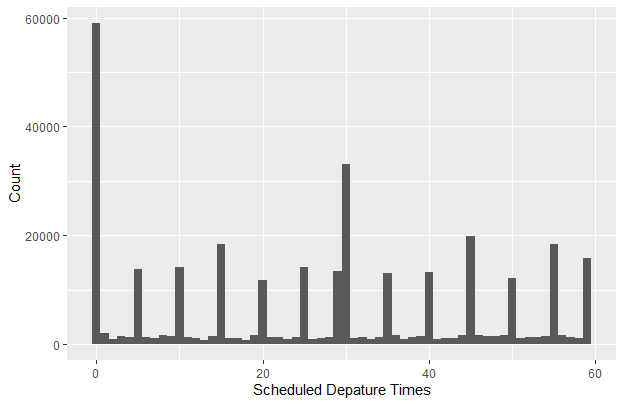
### (6) 16.3.4 Exercise 6 (10 pts)

What makes the distribution of diamonds$carat and flights$sched\_dep\_time similar?

# Not going to lie, it took me a while to figure out this problem but it makes so much sense now.  
  
ggplot(diamonds, aes(x = carat)) +  
 geom\_histogram(binwidth = .05) +   
 labs(y = "Count",x = "Carat")



ggplot(flights\_fixed, aes(x = minute(sched\_dep\_time))) +  
 geom\_histogram(binwidth = 1)+  
 labs(y = "Count",x = "Scheduled Depature Times")



# Looking at the two graphs, it is easy to see that both distributions of data have spikes. This is due to the nature of both data sets. Where each sculptor for the diamond has a goal for the size of diamond and wants no less than a whole number carat. The same way the flights want to leave in minutes ending in 5 or 0 since it is easier for people to understand.