

WRANGLE

1. Import
2. Tidy
3. Transform
4. Tibble
   1. Functions to note: as\_tibble(), tibble()
5. Data Import
   1. **write\_csv**(challenge, "challenge-2.csv")  
      **read\_csv**("challenge-2.csv")
   2. **write\_rds**(challenge, "challenge.rds")  
      **read\_rds**("challenge.rds")
   3. **library**(feather)  
      **write\_feather**(challenge, "challenge.feather")  
      **read\_feather**("challenge.feather")
6. Tidy Data
   1. Use table datasets.
   2. Compute the rate for table2, and table4a + table4b. You will need to perform four operations:
      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 10000.
      4. Store back in the appropriate place.
      5. Which representation is easiest to work with? Which is hardest? Why?
   3. Spread and Gather
      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`)

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

* + 1. Both spread() and gather() have a convert argument. What does it do?
    2. Why does this code fail?

table4a %>%   
 **gather**(1999, 2000, key = "year", value = "cases")  
*#> Error in inds\_combine(.vars, ind\_list): Position must be between 0 and n*

* + 1. Why does spreading this tibble fail? How could you add a new column to fix the problem?

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  
)

* + 1. Tidy the simple tibble below. Do you need to spread or gather it? What are the variables?

preg <- **tribble**(  
 ~pregnant, ~male, ~female,  
 "yes", NA, 10,  
 "no", 20, 12  
)

* 1. Separate and Unite
     1. What do the extra and fill arguments do in separate()? Experiment with the various options for the following two toy datasets.
     2. **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,i")) %>%   
         **separate**(x, **c**("one", "two", "three"))
     3. Both unite() and separate() have a remove argument. What does it do? Why would you set it to FALSE?
     4. Compare and contrast separate() and extract(). Why are there three variations of separation (by position, by separator, and with groups), but only one unite?
  2. Fill()
     1. Compare and contrast the fill arguments to spread() and complete().
     2. What does the direction argument to fill() do?

1. Case Study on Tidy Data
   1. Tidy the who tibble.
      1. Hint (Gather, Mutate, Separate, Select, Separate)
2. Relational Data (**library**(nycflights13))
   1. Imagine you wanted to draw (approximately) the route each plane flies from its origin to its destination. What variables would you need? What tables would you need to combine?
   2. I forgot to draw the relationship between weather and airports. What is the relationship and how should it appear in the diagram?
   3. weather only contains information for the origin (NYC) airports. If it contained weather records for all airports in the USA, what additional relation would it define with flights?
   4. We know that some days of the year are “special”, and fewer people than usual fly on them. How might you represent that data as a data frame? What would be the primary keys of that table? How would it connect to the existing tables?
3. Date and Time : Functions to use from **library**(lubridate). Please explore all the functions.
   1. How does the distribution of flight times within a day change over the course of the year?
   2. Compare dep\_time, sched\_dep\_time and dep\_delay. Are they consistent? Explain your findings.
   3. Compare air\_time with the duration between the departure and arrival. Explain your findings. (Hint: consider the location of the airport.)
   4. How does the average delay time change over the course of a day? Should you use dep\_time or sched\_dep\_time? Why?
   5. On what day of the week should you leave if you want to minimise the chance of a delay?
   6. What makes the distribution of diamonds$carat andflights$sched\_dep\_time similar?
   7. Confirm my hypothesis that the early departures of flights in minutes 20-30 and 50-60 are caused by scheduled flights that leave early. Hint: create a binary variable that tells you whether or not a flight was delayed.
4. Strings **library**(stringr)
   1. In your own words, describe the difference between the sep and collapse arguments to str\_c().
   2. Use str\_length() and str\_sub() to extract the middle character from a string. What will you do if the string has an even number of characters?
   3. What does str\_wrap() do? When might you want to use it?
   4. What does str\_trim() do? What’s the opposite of str\_trim()?
5. Regular Expressions
   1. How would you match the literal string "$^$"?
   2. Given the corpus of common words in stringr::words, create regular expressions that find all words that:
      1. Start with “y”.
      2. End with “x”
      3. Are exactly three letters long. (Don’t cheat by using str\_length()!)
      4. Have seven letters or more.
   3. Since this list is long, you might want to use the match argument tostr\_view() to show only the matching or non-matching words.
   4. Create regular expressions to find all words that:
      1. Start with a vowel.
      2. That only contain consonants. (Hint: thinking about matching “not”-vowels.)
      3. End with ed, but not with eed.
      4. End with ing or ise.
   5. Empirically verify the rule “i before e except after c”.
   6. Is “q” always followed by a “u”?
   7. Write a regular expression that matches a word if it’s probably written in British English, not American English.
   8. Create a regular expression that will match telephone numbers as commonly written in your country.
   9. Describe the equivalents of ?, +, \* in {m,n} form.
   10. Describe in words what these regular expressions match: (read carefully to see if I’m using a regular expression or a string that defines a regular expression.)
       1. ^.\*$
       2. "\\{.+\\}"
       3. \d{4}-\d{2}-\d{2}
       4. "\\\\{4}"
   11. Create regular expressions to find all words that:
       1. Start with three consonants.
       2. Have three or more vowels in a row.
       3. Have two or more vowel-consonant pairs in a row.
   12. Solve the beginner regexp crosswords at<https://regexcrossword.com/challenges/beginner>.

**ANSWERS**

1. Compute the rate for table2, and table4a + table4b. You will need to perform four operations:
2. Extract the number of TB cases per country per year-

**table2 %>%**

**filter(type=="cases") %>%**

**group\_by(country) %>%**

**summarize(TB = sum(count))**

1. Extract the matching population per country per year.

**table2 %>% filter(type == "population")**

1. Divide cases by population, and multiply by 10000.

**tb2\_cases <- filter(table2, type == "cases")[["count"]]**

**tb2\_country <- filter(table2, type == "cases")[["country"]]**

**tb2\_year <- filter(table2, type == "cases")[["year"]]**

**tb2\_population <- filter(table2, type == "population")[["count"]]**

**table2\_new <- tibble(country = tb2\_country,**

**year = tb2\_year,**

**rate = tb2\_cases / tb2\_population)**

**table2\_new**

1. Store back in the appropriate place.

**I stored it in as table2\_new.**

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

**Working with table2 is easier than working with table4a & 4b. All the data is in table2 is separated in table4a & 4b. Due to this the operations are little bit complicated.**

1. Spread and Gather
2. 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`)

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

**The functions spread and gather are not perfectly symmetrical because column type information is not transferred between them. With gather, variable names are always converted to a character vector.**

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

**The convert argument tries to convert character vectors to the appropriate type. In the background this uses the type.convert function. "**

1. Why does this code fail?

table4a %>%   
gather(1999, 2000, key = "year", value = "cases")  
#> Error in inds\_combine(.vars, ind\_list): Position must be between 0 and n

**The code fails because the column names 1999 and 2000 are not standard and thus needs to be mentioned. The tidyverse functions will interpret 1999 and 2000 without estimates as looking for the 1999th and 2000th column of the data frame.**

1. Why does spreading this tibble fail? How could you add a new column to fix the problem?

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  
)

**spread(people, key, value) Error: Duplicate identifiers for rows (1, 3)**

**Spreading the data frame fails because there are two rows with “age” for “Phillip Woods”. I tried to solve this but I cldnt.**

1. Tidy the simple tibble below. Do you need to spread or gather it? What are the variables?

preg <- tribble(  
 ~pregnant, ~male, ~female,  
 "yes", NA, 10,  
 "no", 20, 12  
)

**It needs to gather male (logical values) and count (integer count of gender)**

**gather(preg, sex, count, male, female) %>%**

**mutate(pregnant = pregnant == "yes",**

**female = sex == "female") %>%**

**select(-sex)**

1. Separate and Unite
   * 1. What do the extra and fill arguments do in separate()? Experiment with the various options for the following two toy datasets.
     2. **tibble**(x = **c**("a,b,c", "d,e,f,g", "h,i,j")) %>%   
         **separate**(x, **c**("one", "two", "three"))

**## # A tibble: 3 x 3**

**## one two three**

**## \* <chr> <chr> <chr>**

**## 1 a b c**

**## 2 d e f,g**

**## 3 h i j**

**tibble**(x = **c**("a,b,c", "d,e", "f,g,i")) %>%   
 **separate**(x, **c**("one", "two", "three"))

**## # A tibble: 3 x 3**

**## one two three**

**## \* <chr> <chr> <chr>**

**## 1 a b c**

**## 2 <NA> d e**

**## 3 f g i**

* + 1. Both unite() and separate() have a remove argument. What does it do? Why would you set it to FALSE?

**I would set it to FALSE then if I want to create a new variable. But I have to keep the old one as it is.**

* + 1. Compare and contrast separate() and extract(). Why are there three variations of separation (by position, by separator, and with groups), but only one unite?

**The function extract uses a regular expression to find groups and split into columns. In unite it is unambiguous as it is many columns to one, and once the columns are specified, there is only one way to do it, the only choice is the separate. In separate, it is one to many, and there are multiple ways to split the character string.**

Fill()

* + 1. Compare and contrast the fill arguments to spread() and complete().

**In spread, the fill argument explicitly sets the value to replace NA’s. In complete, the fill argument also sets a value to replace NA’s but it is named list, allowing for different values for different variables. And both cases replace both implicit and explicit missing values.**

* + 1. What does the direction argument to fill() do?

**With fill, it determines whether NA values should be replaced by the previous non-missing value or the next non-missing value.**

1. Case Study on Tidy Data
   1. Tidy the who tibble.
      1. Hint (Gather, Mutate, Separate, Select, Separate)
2. Relational Data (**library**(nycflights13))
   1. Imagine you wanted to draw (approximately) the route each plane flies from its origin to its destination. What variables would you need? What tables would you need to combine?

**To get the route, I need the origin and dest variables in the flights table. And I would connect the flights table to the tailnum variable in the planes table to account for each plane. The route would be constructed from the latitude and longitude variables in the airports table.**

* 1. I forgot to draw the relationship between weather and airports. What is the relationship and how should it appear in the diagram?

**It appears as the tables weather and airports are connected by the airports (NYC) in the faa or origins variables in the tables, respectively.**

* 1. weather only contains information for the origin (NYC) airports. If it contained weather records for all airports in the USA, what additional relation would it define with flights?

**Year, month, day, hour, origin in weather would be matched to year, month, day, hour, dest in flight.**

* 1. We know that some days of the year are “special”, and fewer people than usual fly on them. How might you represent that data as a data frame? What would be the primary keys of that table? How would it connect to the existing tables?

**I would indicate the year, month and day variables to be keys and include other variables which would connect to tables.**

1. Date and Time : Functions to use from **library**(lubridate). Please explore all the functions.
   1. How does the distribution of flight times within a day change over the course of the year?

I got this from internet-

**make\_datetime\_100 <- function(year, month, day, time) {**

**make\_datetime(year, month, day, time %/% 100, time %% 100)**

**}**

**flights\_dt <- 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"))**

**We can show, over time in seconds and separated by each month, the departure time.**

**flights\_dt %>%**

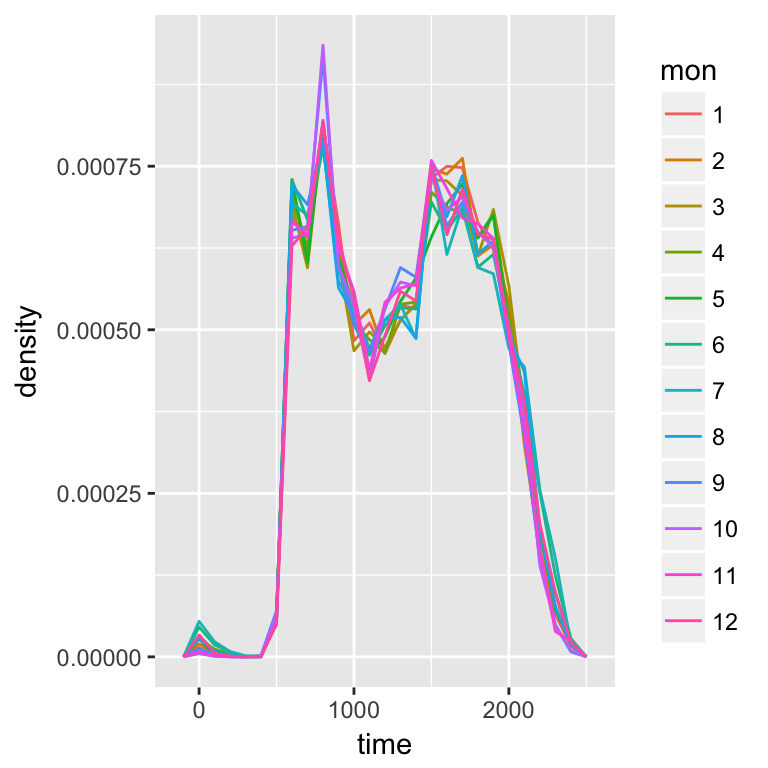
**mutate(time = hour(dep\_time) \* 100 + minute(dep\_time),**

**mon = as.factor(month**

**(dep\_time))) %>%**

**ggplot(aes(x = time, y = ..density.., group = mon, color = mon)) +**

**geom\_freqpoly(binwidth = 100)**

****

**The distribution looks the same from month to month.**

* 1. Compare dep\_time, sched\_dep\_time and dep\_delay. Are they consistent? Explain your findings.

**If they are consistent, then dep\_time = sched\_dep\_time + dep\_delay**

**flights\_dt %>%**

**mutate(dep\_time\_ = sched\_dep\_time + dep\_delay \* 60) %>%**

**filter(dep\_time\_ != dep\_time) %>%**

**select(dep\_time\_, dep\_time, sched\_dep\_time, dep\_delay)**

**## # A tibble: 1,205 x 4**

**## dep\_time\_ dep\_time sched\_dep\_time dep\_delay**

**## <dttm> <dttm> <dttm> <dbl>**

**## 1 2013-01-02 08:48:00 2013-01-01 08:48:00 2013-01-01 18:35:00 853**

**## 2 2013-01-03 00:42:00 2013-01-02 00:42:00 2013-01-02 23:59:00 43**

**## 3 2013-01-03 01:26:00 2013-01-02 01:26:00 2013-01-02 22:50:00 156**

**## 4 2013-01-04 00:32:00 2013-01-03 00:32:00 2013-01-03 23:59:00 33**

**## 5 2013-01-04 00:50:00 2013-01-03 00:50:00 2013-01-03 21:45:00 185**

**## 6 2013-01-04 02:35:00 2013-01-03 02:35:00 2013-01-03 23:59:00 156**

**## 7 2013-01-05 00:25:00 2013-01-04 00:25:00 2013-01-04 23:59:00 26**

**## 8 2013-01-05 01:06:00 2013-01-04 01:06:00 2013-01-04 22:45:00 141**

**## 9 2013-01-06 00:14:00 2013-01-05 00:14:00 2013-01-05 23:59:00 15**

**## 10 2013-01-06 00:37:00 2013-01-05 00:37:00 2013-01-05 22:30:00 127**

**## # ... with 1,195 more rows**

**There exist discrepancies. It looks like there are mistakes in the dates. These are flights in which the actual departure time is on the next day relative to the scheduled departure time. We forgot to account for this when creating the date-times. The code would have had to check if the departure time is less than the scheduled departure time. Alternatively, simply adding the delay time is more robust because it will automatically account for crossing into the next day.**

* 1. Compare air\_time with the duration between the departure and arrival. Explain your findings. (Hint: consider the location of the airport.)

**flights\_dt %>%**

**mutate(flight\_duration = as.numeric(arr\_time - dep\_time),**

**air\_time\_mins = air\_time,**

**diff = flight\_duration - air\_time\_mins) %>%**

**select(origin, dest, flight\_duration, air\_time\_mins, diff)**

**There seems to be a discrepancy in air time should be equal or less than flight duration.**

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

**flights\_dt %>%**

**mutate(sched\_dep\_hour = hour(sched\_dep\_time)) %>%**

**group\_by(sched\_dep\_hour) %>%**

**summarise(dep\_delay = mean(dep\_delay)) %>%**

**ggplot(aes(y = dep\_delay, x = sched\_dep\_hour)) +**

**geom\_point() +**

**geom\_smooth()**

* 1. On what day of the week should you leave if you want to minimise the chance of a delay?

**flights\_dt %>%**

**mutate(dow = wday(sched\_dep\_time)) %>%**

**group\_by(dow) %>%**

**summarise(dep\_delay = mean(dep\_delay),**

**arr\_delay = mean(arr\_delay, na.rm = TRUE),**

**tot\_delay = dep\_delay + arr\_delay)**

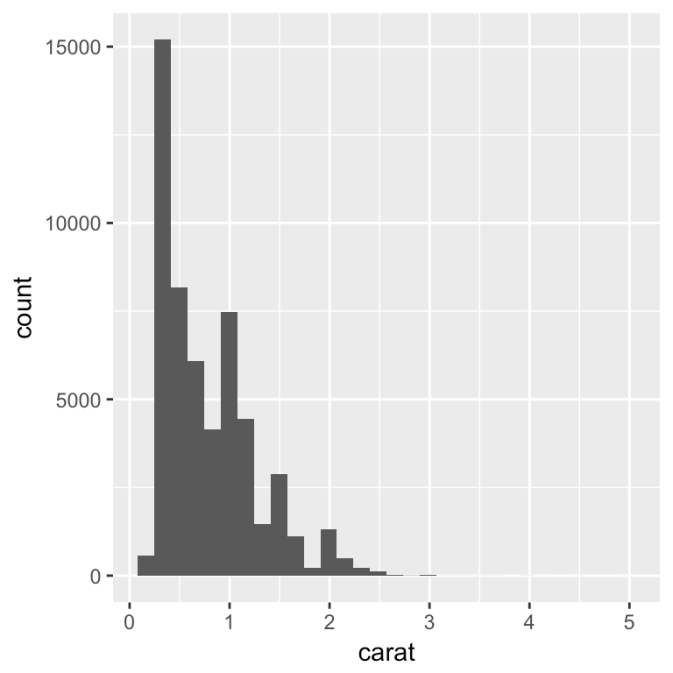
**SUNDAY**

* 1. What makes the distribution of diamonds$carat andflights$sched\_dep\_time similar?

**diamonds %>%**

**ggplot(aes(carat)) +**

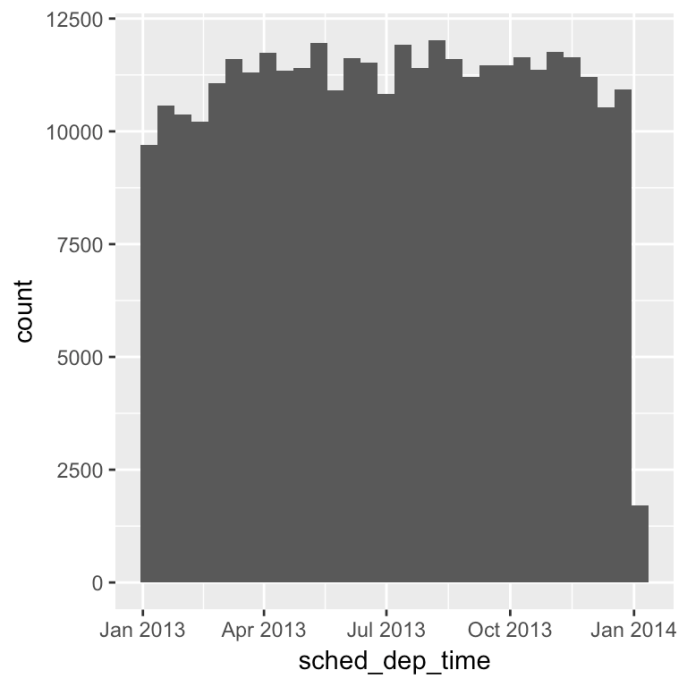
**geom\_histogram()**

****

**flights\_dt %>%**

**ggplot(aes(sched\_dep\_time)) +**

**geom\_histogram()**

****

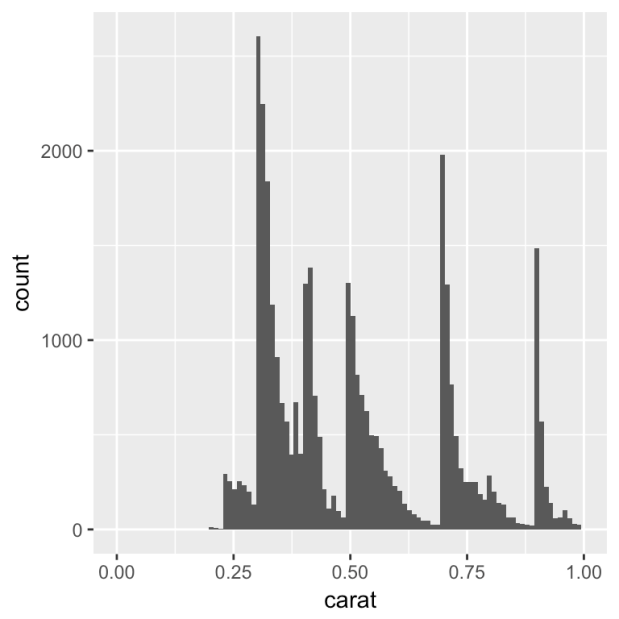
**Let’s look at more granularity,**

**diamonds %>%**

**ggplot(aes(carat)) +**

**geom\_histogram(bins=100) +**

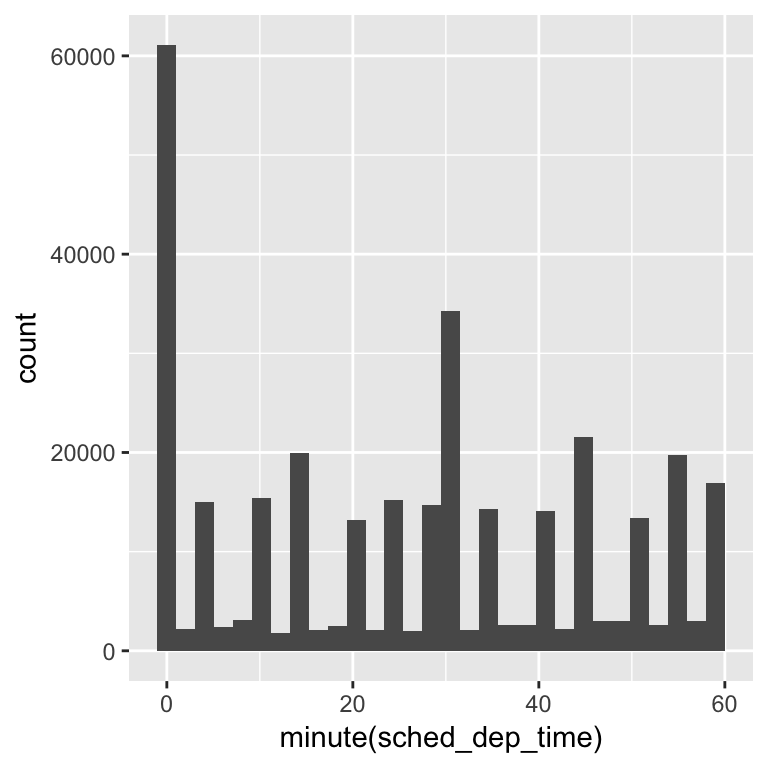
**xlim(0,1)**

****

**flights\_dt %>%**

**ggplot(aes(minute(sched\_dep\_time))) +**

**geom\_histogram()**

****

**Both distributions have multi-modal distributions, reflecting diamond carats at 1/3, 1/2, 2/3, etc and sched\_dep\_time of flights being close to every 5 minutes of the hour.**

* 1. Confirm my hypothesis that the early departures of flights in minutes 20-30 and 50-60 are caused by scheduled flights that leave early. Hint: create a binary variable that tells you whether or not a flight was delayed.

**flights\_dt %>%**

**mutate( wn2030or5060 = ifelse(**

**(minute(sched\_dep\_time)>=20 & minute(sched\_dep\_time)<=30) |**

**(minute(sched\_dep\_time)>=50 & minute(sched\_dep\_time)<=60),**

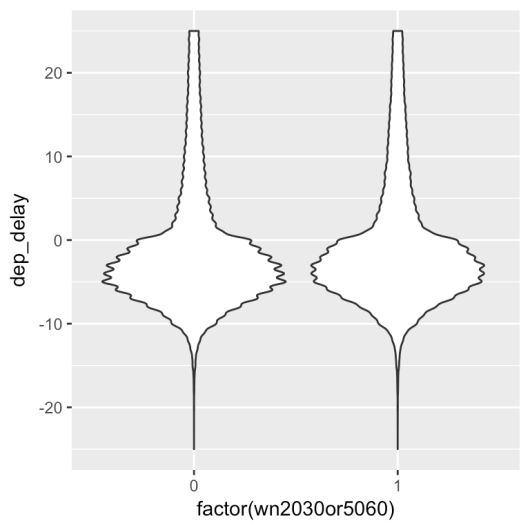
**1,0)) %>%**

**select(sched\_dep\_time,wn2030or5060,dep\_delay) %>%**

**ggplot() +**

**geom\_violin(aes(factor(wn2030or5060),dep\_delay)) +**

**ylim(-25,25)**

****

1. Strings **library**(stringr)
   1. In your own words, describe the difference between the sep and collapse arguments to str\_c().

**The collapse argument joins elements in a character vector while the sep argument joins elements from multiple character vectors.**

* 1. Use str\_length() and str\_sub() to extract the middle character from a string. What will you do if the string has an even number of characters?

**library(stringr)**

**x <- "Apple"**

**med <- ifelse(**

**str\_length(x) %% 2 == 0,**

**ceiling(str\_length(x) / 2),**

**floor(str\_length(x) / 2)**

**)**

**str\_sub(x,med,med)**

* 1. What does str\_wrap() do? When might you want to use it?

**This is useful for reformatting text so that it doesn’t run off the visible page or you want to indent.**

* 1. What does str\_trim() do? What’s the opposite of str\_trim()?

**str\_trim() trims whitespace from the beginning or end of a string. The opposite, str\_pad(), adds whitespace.**

1. Regular Expressions
   1. How would you match the literal string "$^$"?

**x <- '"$^$"'**

**str\_view(x,'"$^$"')**

**"$^$"**

**I think that matches it…**

* 1. Given the corpus of common words in stringr::words, create regular expressions that find all words that:
     1. Start with “y”.

**words <- stringr::words**

**str\_view(words,"^y",match=T)**

* + 1. End with “x”

**str\_view(words,"x$",match=T)**

* + 1. Are exactly three letters long. (Don’t cheat by using str\_length()!)

**str\_view(words,"\\b[[:alpha:]]{3}\\b",match = T)**

* + 1. Have seven letters or more.

**str\_match(words,"\\b[[:alpha:]]{7,}\\b")**

* 1. Since this list is long, you might want to use the match argument tostr\_view() to show only the matching or non-matching words.

**Yes it was a long list.**

* 1. Create regular expressions to find all words that:
     1. Start with a vowel.

**sub\_words <- sample(words,20)**

**str\_view(sub\_words,"^[aeiou]",match=T)**

* + 1. That only contain consonants. (Hint: thinking about matching “not”-vowels.)
    2. End with ed, but not with eed.

**str\_view(sub\_words,"[^e]ed$",match=T)**

* + 1. End with ing or ise.

**str\_view(sub\_words,"ing$|ise$",match=T)**

* 1. Empirically verify the rule “i before e except after c”.
     1. **str\_view(words,"cei",match=T)**
     2. **str\_view(words,"cie",match=T**)
     3. **str\_view(words,"iec",match=T)**
     4. **str\_view(words,"eic",match=T)**

**I think it’s only “iec” not “eic”.**

* 1. Is “q” always followed by a “u”?

**Yes**

**str\_view(words,"uq|qu",match=T)**

* 1. Write a regular expression that matches a word if it’s probably written in British English, not American English.

**str\_view(words,"colour",match=T)**

* 1. Create a regular expression that will match telephone numbers as commonly written in your country.
  2. Describe the equivalents of ?, +, \* in {m,n} form.

**? <- {0,1}**

**<- {1,}**

**<- {0.}**

* 1. Describe in words what these regular expressions match: (read carefully to see if I’m using a regular expression or a string that defines a regular expression.)
     1. ^.\*$

**This matches a 0 or more periods at the beginning of the string to the end.**

* + 1. "\\{.+\\}"

**This string of a regex that matches one or more periods.**

* + 1. \d{4}-\d{2}-\d{2}
    2. "\\\\{4}"

**This regex is in a string and matches 4 's**

* 1. Create regular expressions to find all words that:
     1. Start with three consonants.

**str\_view(words,"^[^aeiou]{3}.\*",match=T)**

* + 1. Have three or more vowels in a row.

**str\_view(words,"^[^aeiou]{3}.\*",match=T)**

* + 1. Have two or more vowel-consonant pairs in a row.

**str\_view(words,"([aeiou][^aeiou]){2,}.\*",match=T)**

Solve the beginner regexp crosswords at<https://regexcrossword.com/challenges/beginner>.