Assignment 10

Instructions: Make an R Notebook and in it answer the question or questions below. When you are done, hand in on Quercus the *output* from Previewing (or Knitting) your Notebook, probably an `html` or `pdf` file. An `html` file is easier for the grader to deal with. Do *not* hand in the Notebook itself. You want to show that you can (i) write code that will answer the questions, (ii) run that code and get some sensible output, (iii) write some words that show you know what is going on and that reflect your conclusions about the data. Your goal is to convince the grader that you *understand* what you are doing: not only doing the right thing, but making it clear that you know *why* it's the right thing.

Do *not* expect to get help on this assignment. The purpose of the assignments is for you to see how much *you* have understood. You will find that you also learn something from grappling with the assignments. The time to get help is after you watch the lectures and work through the problems from PASIAS, via tutorial and the discussion board, that is *before* you start work on the assignment. The only reasons to contact the instructor while working on an assignment are to report (i) something missing like a data file that cannot possibly be read, (ii) something *beyond your control* that makes it impossible to finish the assignment in time after you have started it.

There is a time limit on this assignment (you will see Quercus counting down the time remaining).

1. A student keeps track of what time they go to bed and what time they get up in the morning. They also have an app on their phone that measures the number of hours they were asleep during that time. The data for one week are in http://ritsokiguess.site/STAC33/sleeping.csv, in the 24-hour clock.

   (a) Read in and display the data. What type of things are each of your columns?

Solution:

The usual, to start:

```
my_url <- "http://ritsokiguess.site/STAC33/sleeping.csv"
sleep <- read_csv(my_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   bed.time = col_datetime(format = ""),
##   rise.time = col_datetime(format = ""),
##   sleep.time = col_double()
## )
```

```
sleep
```

```
## # A tibble: 7 x 3
##   bed.time            rise.time           sleep.time
##   <dttm>              <dttm>                   <dbl>
## 1 2013-09-01 23:05:24 2013-09-02 08:03:29       6.74
## 2 2013-09-02 22:51:09 2013-09-03 07:34:21       7.92
## 3 2013-09-04 00:09:16 2013-09-04 07:45:06       7.01
## 4 2013-09-04 23:43:31 2013-09-05 07:07:17       6.23
```

```
## 5 2013-09-06 00:17:41 2013-09-06 08:17:13       6.34
## 6 2013-09-06 22:42:27 2013-09-07 06:52:11       7.42
## 7 2013-09-08 00:22:27 2013-09-08 07:15:19       6.45
```

On mine, the sleep time is an ordinary decimal number, but the two times are something called `dttm`, which I can guess means date-time. In your notebook, you might see `S3:POSIXct`, and you probably don't know what that is (although you can guess).[1]

If you search for this, you'll find some links to the help files, but a bit further down is this, which says it in a few words: "These objects store the number of seconds (for `POSIXct`) … since January 1st 1970 at midnight."[2]

Make the claim that the first two columns are genuine date-times, and if they are labelled `S3:POSIXct` for you, say how you know. That is to say, they may look like pieces of text laid out as date-times, but they are *actual* date-times stored internally as seconds since Jan 1 1970 and displayed nicely. Thus we *do not* need to use `ymd_hms` or anything similar to deal with them.

(b) Work out the fractional number of hours that the student was in bed each of these nights. (They may not have been asleep this whole time.) Your result needs to be a *number* since we will be doing some calculations with it shortly.

Solution:

Since these are genuine date-times, you can take the difference, but the unit is not predictable. Internally, these are stored as a number of *seconds*, but it displays a "nice" unit:

```
sleep %>% mutate(in_bed = rise.time - bed.time)
```

```
## # A tibble: 7 x 4
##   bed.time            rise.time           sleep.time in_bed
##   <dttm>              <dttm>                   <dbl> <drtn>
## 1 2013-09-01 23:05:24 2013-09-02 08:03:29       6.74 8.968056 hours
## 2 2013-09-02 22:51:09 2013-09-03 07:34:21       7.92 8.720000 hours
## 3 2013-09-04 00:09:16 2013-09-04 07:45:06       7.01 7.597222 hours
## 4 2013-09-04 23:43:31 2013-09-05 07:07:17       6.23 7.396111 hours
## 5 2013-09-06 00:17:41 2013-09-06 08:17:13       6.34 7.992222 hours
## 6 2013-09-06 22:42:27 2013-09-07 06:52:11       7.42 8.162222 hours
## 7 2013-09-08 00:22:27 2013-09-08 07:15:19       6.45 6.881111 hours
```

In this case, we did get a number of hours, but in the next part, we are going to do a calculation like this:

```
sleep %>% mutate(in_bed = rise.time - bed.time) %>%
    mutate(ratio = sleep.time / in_bed)
```

```
## Error: Problem with `mutate()` input `ratio`.
## x second argument of / cannot be a "difftime" object
## i Input `ratio` is `sleep.time/in_bed`.
```

and this doesn't work because you can't divide a number by a time. (What would its units be?) So we have to turn `in_bed` into a number, and to do that we can divide by the number of seconds in an hour:

```
sleep %>% mutate(in_bed = (rise.time - bed.time) / dhours(1))
```

```
## # A tibble: 7 x 4
##   bed.time            rise.time           sleep.time in_bed
##   <dttm>              <dttm>                   <dbl>  <dbl>
## 1 2013-09-01 23:05:24 2013-09-02 08:03:29       6.74   8.97
## 2 2013-09-02 22:51:09 2013-09-03 07:34:21       7.92   8.72
## 3 2013-09-04 00:09:16 2013-09-04 07:45:06       7.01   7.60
## 4 2013-09-04 23:43:31 2013-09-05 07:07:17       6.23   7.40
## 5 2013-09-06 00:17:41 2013-09-06 08:17:13       6.34   7.99
## 6 2013-09-06 22:42:27 2013-09-07 06:52:11       7.42   8.16
## 7 2013-09-08 00:22:27 2013-09-08 07:15:19       6.45   6.88
```

This is now correctly a (decimal) number.

(c) The student is concerned with something they call "sleep efficiency". This is the percentage of time in bed spent sleeping. Work out the student's sleep efficiency for the seven nights in this dataframe. Which night was the student's sleep efficiency greatest?

Solution:

Divide the sleep time by the in-bed time and multiply by 100. To answer the last part of the question, you might think of sorting these in descending order as well:

```
sleep %>% mutate(in_bed = (rise.time - bed.time) / dhours(1)) %>%
    mutate(efficiency = sleep.time / in_bed * 100) %>%
    arrange(desc(efficiency))
```

```
## # A tibble: 7 x 5
##   bed.time            rise.time           sleep.time in_bed efficiency
##   <dttm>              <dttm>                   <dbl>  <dbl>      <dbl>
## 1 2013-09-08 00:22:27 2013-09-08 07:15:19       6.45   6.88       93.7
## 2 2013-09-04 00:09:16 2013-09-04 07:45:06       7.01   7.60       92.3
## 3 2013-09-06 22:42:27 2013-09-07 06:52:11       7.42   8.16       90.9
## 4 2013-09-02 22:51:09 2013-09-03 07:34:21       7.92   8.72       90.8
## 5 2013-09-04 23:43:31 2013-09-05 07:07:17       6.23   7.40       84.2
## 6 2013-09-06 00:17:41 2013-09-06 08:17:13       6.34   7.99       79.3
## 7 2013-09-01 23:05:24 2013-09-02 08:03:29       6.74   8.97       75.2
```

The night of September 8. This was the night the student went to bed the latest, but they were asleep almost all the time they were in bed.

(d) Display the time spent in bed each night as a number of hours, minutes and seconds.

Solution:

The idea here is to display the time between going to bed and getting up as an interval, using %--%, and then turn that into a period:

```
sleep %>% mutate(in_bed_hms = as.period(bed.time %--% rise.time))
```

```
## # A tibble: 7 x 4
##   bed.time            rise.time           sleep.time in_bed_hms
```

```
##   <dttm>              <dttm>                <dbl> <Period>
## 1 2013-09-01 23:05:24 2013-09-02 08:03:29    6.74 8H 58M 5S
## 2 2013-09-02 22:51:09 2013-09-03 07:34:21    7.92 8H 43M 12S
## 3 2013-09-04 00:09:16 2013-09-04 07:45:06    7.01 7H 35M 50S
## 4 2013-09-04 23:43:31 2013-09-05 07:07:17    6.23 7H 23M 46S
## 5 2013-09-06 00:17:41 2013-09-06 08:17:13    6.34 7H 59M 32S
## 6 2013-09-06 22:42:27 2013-09-07 06:52:11    7.42 8H 9M 44S
## 7 2013-09-08 00:22:27 2013-09-08 07:15:19    6.45 6H 52M 52S
```

(e) Make a graph of what time the student went to bed each night. Bear in mind that you only need the times, not the dates, and that you want a graph that is informative, showing appropriately the distribution of times the student went to bed.

Solution:

If you just pull out the times, some of them will be at the end of the day and some will be at the beginning. Extracting the hours, minutes and seconds is one way:[3]

```
sleep %>% mutate(h = hour(bed.time), m = minute(bed.time), s = second(bed.time))
```

```
## # A tibble: 7 x 6
##   bed.time            rise.time           sleep.time     h     m     s
##   <dttm>              <dttm>                   <dbl> <int> <int> <dbl>
## 1 2013-09-01 23:05:24 2013-09-02 08:03:29       6.74    23     5    24
## 2 2013-09-02 22:51:09 2013-09-03 07:34:21       7.92    22    51     9
## 3 2013-09-04 00:09:16 2013-09-04 07:45:06       7.01     0     9    16
## 4 2013-09-04 23:43:31 2013-09-05 07:07:17       6.23    23    43    31
## 5 2013-09-06 00:17:41 2013-09-06 08:17:13       6.34     0    17    41
## 6 2013-09-06 22:42:27 2013-09-07 06:52:11       7.42    22    42    27
## 7 2013-09-08 00:22:27 2013-09-08 07:15:19       6.45     0    22    27
```

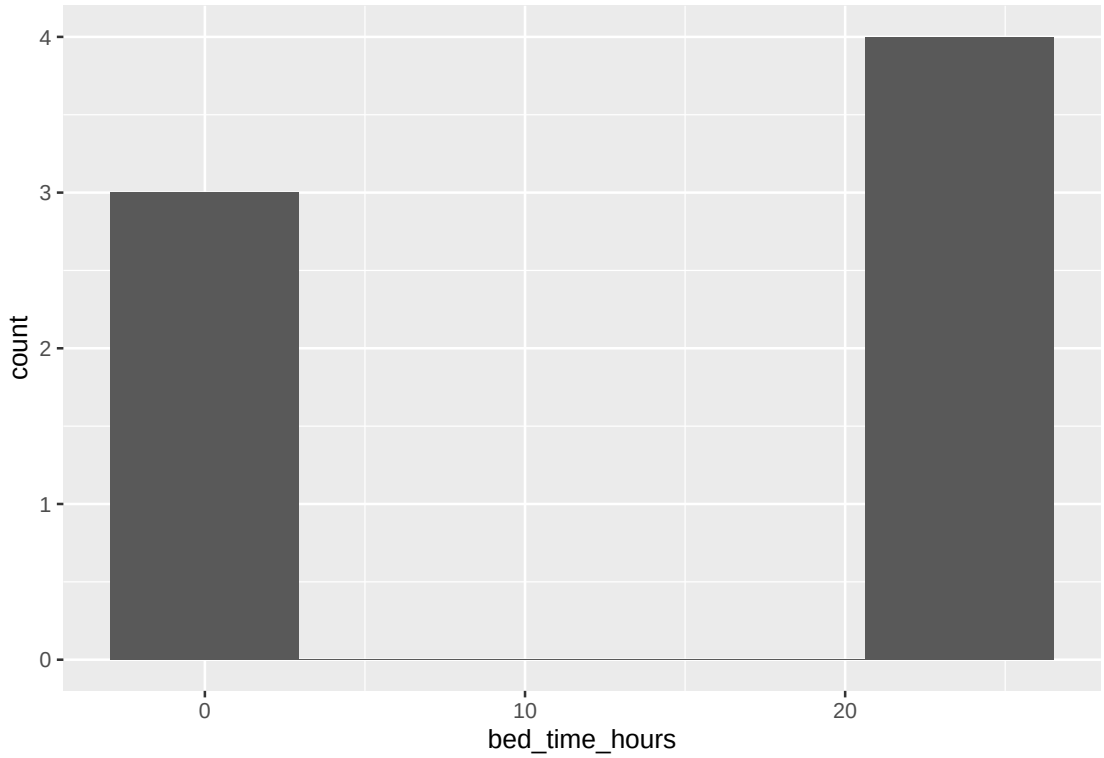
You could convert these into fractional hours to make a histogram of:

```
sleep %>% mutate(h = hour(bed.time), m = minute(bed.time), s = second(bed.time)) %>%
    mutate(bed_time_hours = h + m / 60 + s / (60*60))
```

```
## # A tibble: 7 x 7
##   bed.time            rise.time           sleep.time     h     m     s
##   <dttm>              <dttm>                   <dbl> <int> <int> <dbl>
## 1 2013-09-01 23:05:24 2013-09-02 08:03:29       6.74    23     5    24
## 2 2013-09-02 22:51:09 2013-09-03 07:34:21       7.92    22    51     9
## 3 2013-09-04 00:09:16 2013-09-04 07:45:06       7.01     0     9    16
## 4 2013-09-04 23:43:31 2013-09-05 07:07:17       6.23    23    43    31
## 5 2013-09-06 00:17:41 2013-09-06 08:17:13       6.34     0    17    41
## 6 2013-09-06 22:42:27 2013-09-07 06:52:11       7.42    22    42    27
## 7 2013-09-08 00:22:27 2013-09-08 07:15:19       6.45     0    22    27
## # ... with 1 more variable: bed_time_hours <dbl>
```

but if you make a histogram of these, this is what you get:

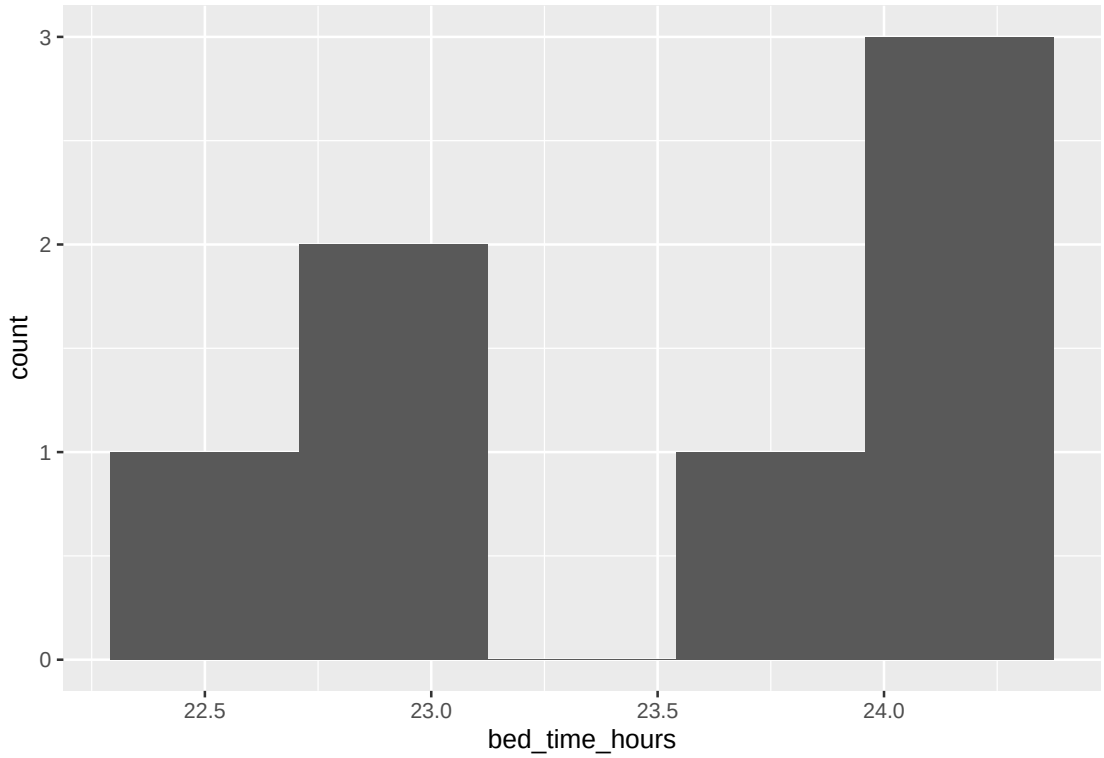
```
sleep %>% mutate(h = hour(bed.time), m = minute(bed.time), s = second(bed.time)) %>%
    mutate(bed_time_hours = h + m / 60 + s / (60*60)) %>%
    ggplot(aes(x = bed_time_hours)) + geom_histogram(bins = 5)
```



but this makes no sense because the bedtimes after midnight are on the end of the previous day, not the beginning of the next one!

With that in mind, let's move the bedtimes that are, say, before 3:00am to the end of the previous day by adding 24 to them before we make the graph:

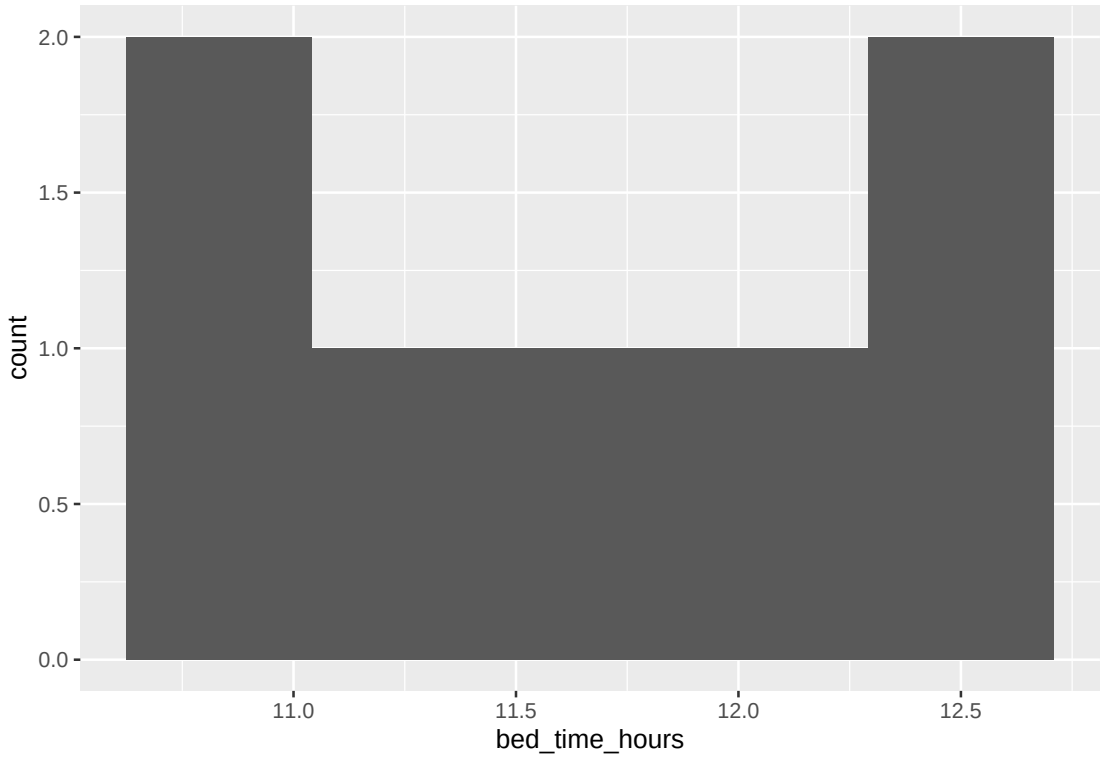
```
sleep %>% mutate(h = hour(bed.time), m = minute(bed.time), s = second(bed.time)) %>%
    mutate(bed_time_hours = h + m / 60 + s / (60*60)) %>%
    mutate(bed_time_hours = ifelse(bed_time_hours < 3, bed_time_hours + 24, bed_time_hours)) %
    ggplot(aes(x = bed_time_hours)) + geom_histogram(bins = 5)
```



This gives a sense of where the bedtimes are. If you're used to reading the 24-hour clock, you'll know that 23 is 11:00pm, and you'll have a sense that some of the bedtimes were 11 or a bit earlier and some were around midnight. (I like the 24-hour clock.) There are only 7 observations, so the graph you get won't look very nice as a histogram, but at least this one says something about when the student went to bed, in a way that puts times just after midnight next to times just before. You should give some thought about the number of bins; with only 7 observations, even 5 bins is pushing it, but this looked nicer to me than 4 bins.

If you're more used to the 12-hour clock, you'll want to convert the times to something between 10 and 12. You can do this with an `ifelse` as above, subtracting 12 from the ones before midnight and adding 12 to the ones after. Or you can recognize this as modulo arithmetic (the clock is a classic case: what is 10:00pm plus 3 hours?) A little thought will reveal that subtracting (or adding) 12 hours and taking the result modulo 24 would do it: the pre-midnight bedtimes will get turned into a number like 10 or 11, and the post-midnight ones to 12 and a bit. R has a modulo operator, which is `%%` (cite your source: mine was this):

```
sleep %>% mutate(h = hour(bed.time), m = minute(bed.time), s = second(bed.time)) %>%
    mutate(bed_time_hours = h + m / 60 + s / (60*60)) %>%
    mutate(bed_time_hours = (bed_time_hours - 12) %% 24) %>%
    ggplot(aes(x = bed_time_hours)) + geom_histogram(bins = 5)
```



and you might find the x-scale of that easier to cope with. (The bins have come out differently, for some reason.)

I think the best graph uses the fact that date-times plot nicely, so if we keep them as date-times, the x-scale will look nice. The problem is that they are times on *different* days. What if we faked it up so that they were all on the *same* day (or, at least, consecutive days, to account for the ones after midnight)?

Let's look at our dataframe again:

```
sleep
```

```
## # A tibble: 7 x 3
##   bed.time            rise.time           sleep.time
##   <dttm>              <dttm>                   <dbl>
## 1 2013-09-01 23:05:24 2013-09-02 08:03:29       6.74
## 2 2013-09-02 22:51:09 2013-09-03 07:34:21       7.92
## 3 2013-09-04 00:09:16 2013-09-04 07:45:06       7.01
## 4 2013-09-04 23:43:31 2013-09-05 07:07:17       6.23
## 5 2013-09-06 00:17:41 2013-09-06 08:17:13       6.34
## 6 2013-09-06 22:42:27 2013-09-07 06:52:11       7.42
## 7 2013-09-08 00:22:27 2013-09-08 07:15:19       6.45
```

The `rise.time` values are all a.m., and on consecutive days, so if we subtract consecutive numbers of days from the `bed.time`s, we'll put them all on appropriate days too:

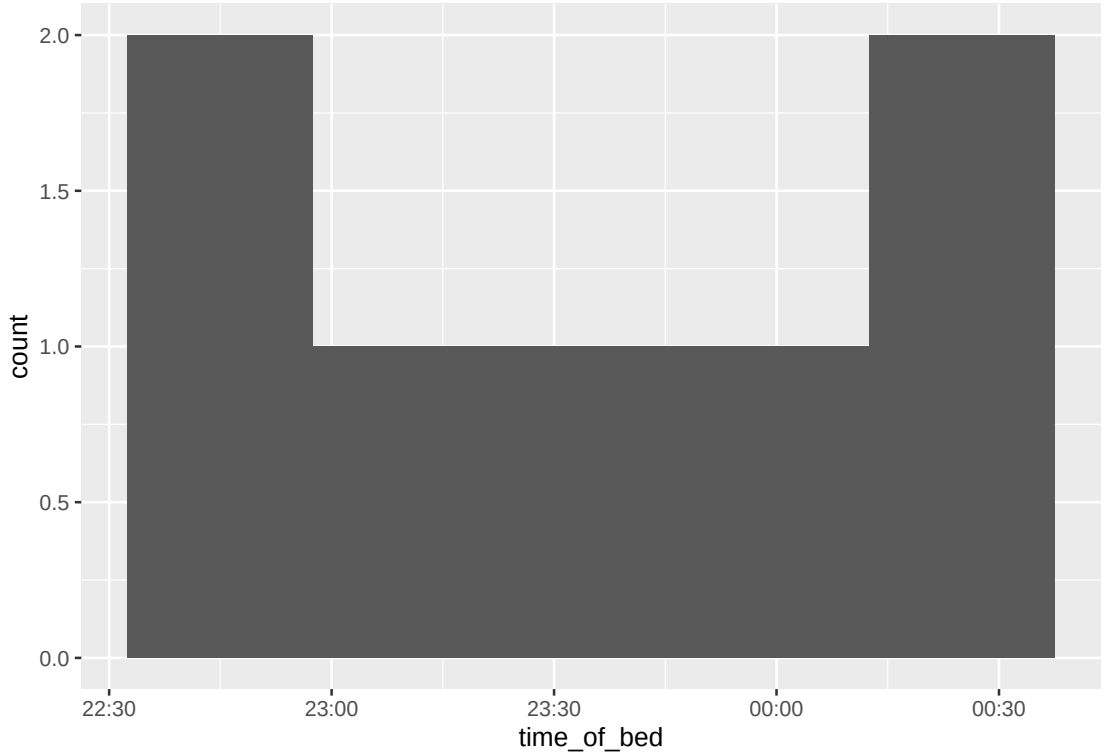
```
sleep %>% mutate(time_of_bed = bed.time - ddays(0:6))
```

```
## # A tibble: 7 x 4
##   bed.time            rise.time           sleep.time time_of_bed
```

```
##   <dttm>              <dttm>                <dbl> <dttm>
## 1 2013-09-01 23:05:24 2013-09-02 08:03:29    6.74 2013-09-01 23:05:24
## 2 2013-09-02 22:51:09 2013-09-03 07:34:21    7.92 2013-09-01 22:51:09
## 3 2013-09-04 00:09:16 2013-09-04 07:45:06    7.01 2013-09-02 00:09:16
## 4 2013-09-04 23:43:31 2013-09-05 07:07:17    6.23 2013-09-01 23:43:31
## 5 2013-09-06 00:17:41 2013-09-06 08:17:13    6.34 2013-09-02 00:17:41
## 6 2013-09-06 22:42:27 2013-09-07 06:52:11    7.42 2013-09-01 22:42:27
## 7 2013-09-08 00:22:27 2013-09-08 07:15:19    6.45 2013-09-02 00:22:27
```

These are all around the midnight at the end of September 1, so some of them are in the early hours of September 2. Now, if we make a histogram of *those*:

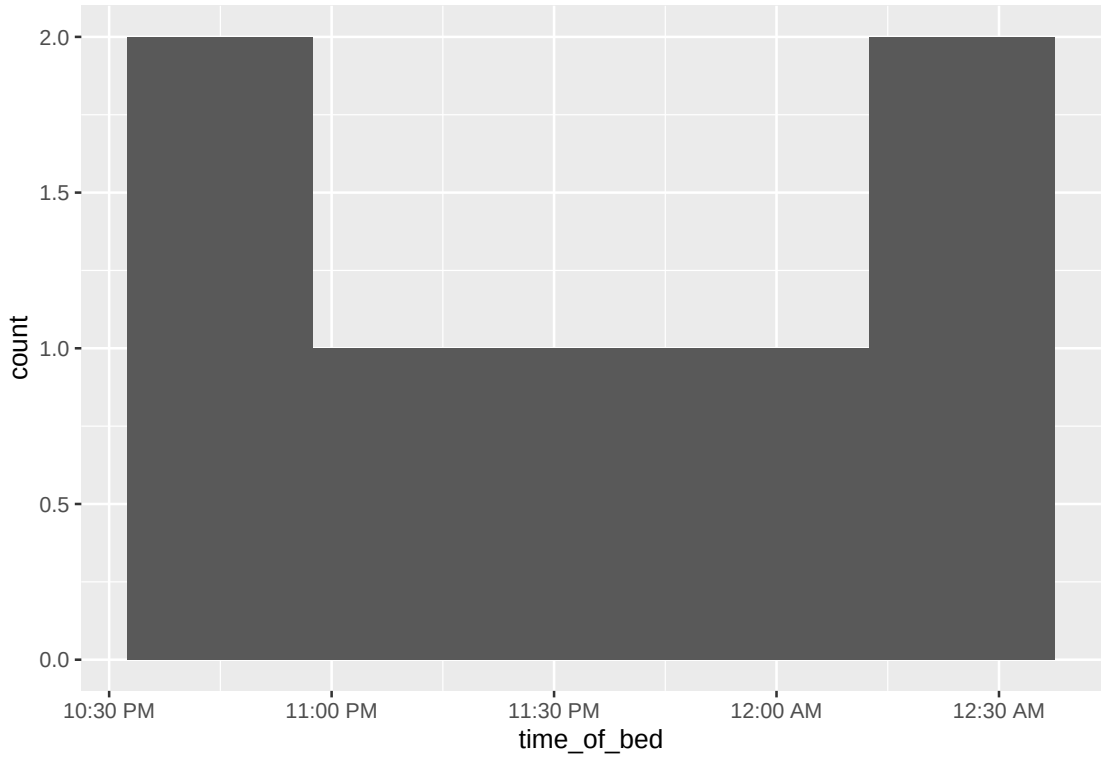
```
sleep %>% mutate(time_of_bed = bed.time - ddays(0:6)) %>%
    ggplot(aes(x = time_of_bed)) + geom_histogram(bins = 5)
```



Now the x-axis formatting looks like a time, and spills seamlessly into the next day. (There was no real range of dates, so the formatting is of the times only.)

One more embellishment, idea from here:

```
sleep %>% mutate(time_of_bed = bed.time - ddays(0:6)) %>%
    ggplot(aes(x = time_of_bed)) + geom_histogram(bins = 5) +
    scale_x_datetime(date_labels = "%l:%M %p")
```



The `scale_x` and `scale_y` functions customize the x and y axes respectively. Inside `date_labels` go some codes that say what time units you want to display: in this case, the 12-hour hours, the minutes, and whether the time is AM or PM. The codes come from a function called `strftime`, and a full list is here. Alternatively, you can look up the help for R's function of the same name with `?strftime`.[4]

There is a second question on the next page.

2. Earlier, we investigated some data on predicting the height of a person from the length of their foot. The data were in http://ritsokiguess.site/STAC32/heightfoot.csv.

   (a) Read in and display (some of) the data.

Solution:

Copy what you did before:

```
my_url <- "http://ritsokiguess.site/STAC32/heightfoot.csv"
hf <- read_csv(my_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   height = col_double(),
##   foot = col_double()
## )
```

```
hf
```

```
## # A tibble: 33 x 2
##    height  foot
##     <dbl> <dbl>
##  1   66.5  27
##  2   73.5  29
##  3   70    25.5
##  4   71    27.9
##  5   73    27
##  6   71    26
##  7   71    29
##  8   69.5  27
##  9   73    29
## 10   71    27
## # ... with 23 more rows
```

   (b) In STAC67, you learn (or will learn) the matrix formulation of the least squares estimates of intercept and slope. This produces a vector $\hat{\beta}$ containing estimates of both the intercept and the slope, from the formula

$$\hat{\beta} = (X^T X)^{-1} X^T y,$$

   where:

   - X is a matrix containing a column of 1s followed by all the columns of explanatory variables
   - X^T denotes the (matrix) transpose of X
   - M^{-1} denotes the inverse of the matrix M
   - y denotes the column of response variable values.

   Use the formula above to obtain the least squares estimates of intercepts and slope for this regression, using R's vector-matrix algebra. Hint: you are advised to do the calculation in steps, or else it will be very hard to read, and hard for the grader to check that it is correct.

Solution:

There is some setup first: we have to get hold of X and y from the data as a matrix and a vector respectively. I would use tidyverse ideas to do this, and then turn them into a matrix at the end, which I think is best. Don't forget to create a column of 1s to make the first column of X

```
hf %>% mutate(one=1) %>%
  select(one, foot) %>%
  as.matrix() -> X
head(X)
```

```
##      one foot
## [1,]   1 27.0
## [2,]   1 29.0
## [3,]   1 25.5
## [4,]   1 27.9
## [5,]   1 27.0
## [6,]   1 26.0
```

(`head` displays the first six rows, or else you'll be displaying all 33, which is too many.)

Another approach is this:

```
X <- cbind(1, hf$foot)
head(X)
```

```
##      [,1] [,2]
## [1,]    1 27.0
## [2,]    1 29.0
## [3,]    1 25.5
## [4,]    1 27.9
## [5,]    1 27.0
## [6,]    1 26.0
```

Note that the recycling rules mean that a column with only one value in it will be repeated to the length of the other one, and so this is better than working out how many observations there are and repeating `1` that many times.

The choice here is whether to use tidyverse stuff and turn into a matrix at the end, or make a matrix at the start (which is what `cbind` from base R is doing). I don't believe you've seen that in this course, so you ought to cite your source if you go that way.

The simplest choice for making y is this:

```
y <- hf$height
y
```

```
##  [1] 66.5 73.5 70.0 71.0 73.0 71.0 71.0 69.5 73.0 71.0 69.0 69.0 73.0 75.0 73.0
## [16] 72.0 69.0 68.0 72.5 78.0 79.0 71.0 74.0 66.0 71.0 71.0 71.0 84.0 77.0 72.0
## [31] 70.0 76.0 68.0
```

This also works:

```
hf %>% select(height) %>% pull(height)
```

```
##  [1] 66.5 73.5 70.0 71.0 73.0 71.0 71.0 69.5 73.0 71.0 69.0 69.0 73.0 75.0 73.0
## [16] 72.0 69.0 68.0 72.5 78.0 79.0 71.0 74.0 66.0 71.0 71.0 71.0 84.0 77.0 72.0
```

```
## [31] 70.0 76.0 68.0
```

(remembering that you don't want to have anything that's a dataframe), or this:

```
hf %>% select(height) %>% as.matrix() -> yy
head(yy)
```

```
##      height
## [1,]   66.5
## [2,]   73.5
## [3,]   70.0
## [4,]   71.0
## [5,]   73.0
## [6,]   71.0
```

remembering that a (column) vector and a 1-column matrix are the same thing as R is concerned.

Now we want to construct some things. I would go at it this way, rather than trying to do everything at once (if you do, you will either get lost now, or in six months when you try to figure out what you did):

```
Xt <- t(X) # X-transpose
XtX <- Xt %*% X
XtXi <- solve(XtX)
Xty <- Xt %*% y
XtXi %*% Xty
```

```
##           [,1]
## [1,] 34.336335
## [2,]  1.359062
```

The intercept is 34.33 and the slope is 1.36.

These compute, respectively, X^T, X^TX, the inverse of that, X^Ty and $\hat{\beta}$. Expect credit for laying out your calculation clearly.

Extra: the value of this formula is that it applies no matter whether you have one x-variable, as here (or in the windmill data), or whether you have a lot (as in the asphalt data). In either case, $\hat{\beta}$ contains the estimate of the intercept followed by all the slope estimates, however many there are. There are also matrix formulas that tell you how the slopes or the residuals will change if you remove one observation or one explanatory variable, so that something like `step` will work very efficiently, and calculations for leverages likewise.

(c) Verify that your calculation is correct by running the regression.

Solution:

The usual `lm`:

```
hf.1 <- lm(height ~ foot, data = hf)
summary(hf.1)
```

```
##
## Call:
## lm(formula = height ~ foot, data = hf)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7491 -1.3901 -0.0310  0.8918 12.9690
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  34.3363     9.9541   3.449 0.001640 **
## foot          1.3591     0.3581   3.795 0.000643 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.102 on 31 degrees of freedom
## Multiple R-squared:  0.3173, Adjusted R-squared:  0.2952
## F-statistic: 14.41 on 1 and 31 DF,  p-value: 0.0006428
```

The same.

Extra: in this "well-conditioned" case,[5] it makes no difference, but if X^TX is almost singular, so that it almost doesn't have an inverse (for example, some of your explanatory variables are highly correlated with each other), you can get into trouble. Regression calculations in practice use something more sophisticated like the singular value decomposition of X^TX to diagnose whether X^TX is actually singular or almost so, which from a numerical point of view is almost as bad, and to produce a sensible answer in that case.

I guess I should try to make up one where it struggles. Let me do one with two x's that are strongly correlated:

```
d <- tribble(
  ~x1, ~x2, ~y,
  10, 20, 55,
  11, 19.0001, 60,
  12, 17.9999, 61,
  13, 17.0001, 64,
  14, 15.9998, 66,
  15, 15.0001, 67
)
d
```

```
## # A tibble: 6 x 3
##      x1    x2     y
##   <dbl> <dbl> <dbl>
## 1    10  20      55
## 2    11  19.0    60
## 3    12  18.0    61
## 4    13  17.0    64
## 5    14  16.0    66
## 6    15  15.0    67
```

x2 is almost exactly equal to 30 minus x1. What's the right answer?

```
d.1 <- lm(y ~ x1 + x2, data = d)
summary(d.1)
```

```
##
```

```
## Call:
## lm(formula = y ~ x1 + x2, data = d)
##
## Residuals:
##       1       2       3       4       5       6
## -1.37530  1.26859  0.03118  0.63549  0.43765 -0.99760
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -11837.3   138685.6  -0.085    0.937
## x1             398.0     4622.9   0.086    0.937
## x2             395.7     4622.8   0.086    0.937
##
## Residual standard error: 1.303 on 3 degrees of freedom
## Multiple R-squared:  0.9485, Adjusted R-squared:  0.9141
## F-statistic: 27.61 on 2 and 3 DF,  p-value: 0.0117
```

```
coef(d.1)
```

```
## (Intercept)          x1          x2
## -11837.2938    398.0000    395.6835
```

You should be right away suspicious here: the R-squared is high, but neither of the explanatory variables are significant! (This actually means that you can remove one of them, either one.) The standard errors are also suspiciously large, never a good sign. If you've done C67, you might be thinking about variance inflation factors here:

```
library(car)
```

```
## Loading required package: carData
```

```
vif(d.1)
```

```
##        x1        x2
## 220326265 220326265
```

These are both huge (greater than 5 or 10 or whatever guideline you use), indicating that they are highly correlated with each other (as we know they are).

All right, how does the matrix algebra work? This is just the same as before:

```
d %>% mutate(one=1) %>%
  select(one, starts_with("x")) %>%
  as.matrix() -> X
head(X)
```

```
##      one x1      x2
## [1,]   1 10 20.0000
## [2,]   1 11 19.0001
## [3,]   1 12 17.9999
## [4,]   1 13 17.0001
## [5,]   1 14 15.9998
## [6,]   1 15 15.0001
```

and then

```
y <- d$y
Xt <- t(X) # X-transpose
XtX <- Xt %*% X
XtXi <- solve(XtX)
Xty <- Xt %*% y
XtXi %*% Xty
```

```
##             [,1]
## one -11837.2051
## x1     397.9970
## x2     395.6805
```

These answers are actually noticeably different from the right answers (with a few more decimals here):

```
coef(d.1)
```

```
## (Intercept)          x1          x2
## -11837.2938    398.0000    395.6835
```

One way of finding out how nearly singular X^TX is is to look at its eigenvalues. You'll recall that a singular matrix has one or more zero eigenvalues:

```
eigen(XtX)
```

```
## eigen() decomposition
## $values
## [1] 2.781956e+03 3.404456e+01 8.805997e-11
##
## $vectors
##             [,1]        [,2]        [,3]
## [1,] -0.04643281 -0.00782885  0.99889074
## [2,] -0.57893109 -0.81469635 -0.03329647
## [3,] -0.81405331  0.57983495 -0.03329628
```

The third eigenvalue is 8.8×10^{-11}, which is very close to zero, especially compared to the other two, which are 34 and over two *thousand*. This is a very nearly singular matrix, and hence $(X^TX)^{-1}$ is very close to not existing at all, and *that* would mean that you couldn't even compute the intercept and slope estimates, never mind hope to get close to the right answer.

Notes

1. Which one you see will depend on which type of output you are looking at. In your notebook you will probably see the POSIXCT thing, but in your HTML or PDF output it will probably say "dttm".

2. The piece in quotes comes word-for-word from the source: it is exactly what the author said. Except that the author also talks about dates, which don't concern us here, so I removed that bit and replaced it with the three dots, called an "ellipsis", to show that the author said some extra stuff that I didn't quote. I checked that what remained does actually still capture what the author said. Extra-in-endnote: an ellipsis is not to be confused with the conic section called an ellipse, and these three dots are not to be confused with the three dots in an R function, where they mean "anything else that was passed in to the function". Both uses of the three dots capture the idea of "something was missed out".

3. Make sure you use "hour" and not "hours" as I did the first time. That computes the total number of hours between the zero date of Jan 1, 1970 and the time given, and so is way too large to be an answer here!

4. Confusingly, uppercase I and lowercase l not only look the same, but they also both display the 12-hour hour. The former adds a zero to the front if the hour is a single digit, and the latter does not. All the hours here have two digits, though, so it comes out the same whichever you use.

5. Which in this case means that the column of 1s and the actual x values are not strongly correlated, which means that the x-values vary enough.