STAC32

Assignment 1

Due Thursday September 12 at 11:59pm

The question(s) at the end without solutions, only one here, are an assignment: you need to do these questions yourself and hand them in (instructions below). These are due on the date shown above. An assignment handed in after the deadline is late, and may or may not be accepted (see course outline). I am usually OK with a few minutes late, but not more than that. Don't take that risk.

My solutions to the assignment questions will be available when everyone has handed in their assignment. These are *my* solutions; you can have something different and still be correct.

The grader will have 100-odd assignments to mark, so it is vitally important that you make your assignment *easy* for the grader to deal with. Your answers *must be easy* for the grader to find. A simple structure is, for each part of each question in order, to put these three things:

- Your *code*
- Your *output*
- Your *answers and explanation*

When you use an R Notebook with your comments below the output, and "preview" the result, the HTML (or Word) file that comes out is in this format. This is why I talk about R Notebooks in class. You can do it differently, but then you have extra work to organize things.

You are reminded that work handed in with your name on it must be *entirely your own work*. It is as if you have signed your name under it. If it was done wholly or partly by someone else, *you have committed an academic offence*, and you can expect to be asked to explain yourself. The same applies if you allow someone else to copy your work. The graders will be watching out for assignments that look suspiciously similar to each other (or to my solutions). Besides which, if you do not do your own assignments, you *will* do badly on the exams, because the struggle to figure things out for yourself is an important part of the learning process.

Assignments are to be handed in on Quercus. See `https://www.utsc.utoronto.ca/~butler/c32/quercus1.nb.html` for instructions on handing in assignments in Quercus. *Allow yourself time to figure out what you need to do.*

As with any other course involving software, *there are no extensions due to failure to access software*. It is *your* responsibility to make sure that you allow yourself enough time to get connected to R, and to get any packages installed and working. (This is more of an issue if you are using R Studio Cloud, which might be busy and therefore slow; if you install R and R Studio on your own computer, you won't be fighting with other people for resources, but you will need to get everything set up.)

Once you are sitting in front of R Studio (either on `rstudio.cloud` or on your own computer), you would do well to make a new notebook, create a code chunk, in it put

```
library(tidyverse)

## -- Attaching packages --------------------------------- tidyverse 1.2.1 --
## v ggplot2 3.2.0          v purrr   0.3.2
## v tibble  2.1.3          v dplyr   0.8.0.1
## v tidyr   0.8.3.9000     v stringr 1.4.0
## v readr   1.3.1          v forcats 0.3.0

## -- Conflicts ------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

and run it (like I just did). If you don't get output like I did, instead getting an error like "no such package", you need to install the Tidyverse first, like this:

```
install.packages("tidyverse")
```

and then you can try again. If at any point R Studio invites you to install any other packages, let it do so.

1. Work through chapter 4 of PASIAS, along with whatever of chapter 3 that you haven't worked through yet.

   Hand the next question in, in which you will learn some astronomy:

2. The planet Saturn has five satellites ("moons") that rotate around the planet. With so many satellites, it is quite common to see one of them passing in front of another. There are two kinds of events observed here: "eclipses" and "occultations", a total of 19 events in all. I give definitions below. The events were observed for one month. For each event, the date was recorded, along with the type of event, and the percentage of light lost by the eclipsed or occulted satellite.

   Definitions:

   - An **occultation** occurs when a larger object (from the point of view of the observer) passes in front of a smaller one. From the Earth, an occultation might occur when the moon (which looks large) passes in front of a star (which looks small because it is so far away).
   - An **eclipse** occurs when an object passes in front of one that is about the same size (as it looks to the observer), so that the light from the larger object is never completely blocked. For example, from the Earth, the moon looks about the same size as the sun, so that when the moon passes in front of the sun, it is an eclipse. Occasionally, there is a "total eclipse" in which the moon blocks out all the light from the sun for a few minutes. If you are in the right place on Earth, and there are no clouds, you can experience one of these.
   - The other possibility (of which there were none in our data set) is that a visibly smaller object passes in front of a larger one. This is called a **transit**. (There were none in our data because Saturn's satellites are of similar sizes, so that the one passing in front will look bigger or about the same size as the one passing behind.) For example, the planet Venus is tiny compared to the sun, so if Venus is directly between us and the sun, it is a transit.

   References: `https://en.wikipedia.org/wiki/Eclipse`, `https://en.wikipedia.org/wiki/Occultation`.

   The data are in `http://www.utsc.utoronto.ca/~butler/assgt_data/saturn.txt`.

   (a) (2 marks) Read in and display the data. Do you have the right number of rows?

   Solution: Take a look at the data: values separated by single spaces, so `read_delim` is the thing. The second input is what separates the data values, here a space:

```
url="http://www.utsc.utoronto.ca/~butler/assgt_data/saturn.txt"
saturn=read_delim(url, " ")

## Parsed with column specification:
## cols(
##   date = col_date(format = ""),
##   event = col_character(),
##   light_loss = col_double()
## )

saturn

## # A tibble: 19 x 3
##    date       event   light_loss
##    <date>     <chr>        <dbl>
##  1 1995-08-02 eclipse         65
##  2 1995-08-04 eclipse         61
##  3 1995-08-05 occult           1
##  4 1995-08-06 eclipse         56
##  5 1995-08-08 eclipse         46
##  6 1995-08-08 occult           2
##  7 1995-08-09 occult           9
##  8 1995-08-11 occult           5
##  9 1995-08-12 occult          39
## 10 1995-08-14 occult           1
## 11 1995-08-14 eclipse        100
## 12 1995-08-15 occult           5
## 13 1995-08-15 occult           4
## 14 1995-08-16 occult          13
## 15 1995-08-20 occult          11
## 16 1995-08-23 occult           3
## 17 1995-08-23 occult          20
## 18 1995-08-25 occult          20
## 19 1995-08-28 occult          12
```

I said in the question that 19 events were recorded, so there should be (and are) 19 rows. You might only see the first ten rows, with buttons to click to see the rest, but there will be an indication *somewhere* that there are 19 rows altogether.

Call the data frame whatever you like (and use that name for the rest of the question). I like to use a name that reflects what's in it, but it is your choice. (My preference is not to use something like `mydata`, but if you can do it and not get confused by it, go for it.)

(b) (3 marks) What kind of values do each of your three variables have? Explain briefly.

Solution: Look at the tops of the columns of your display of the data frame. `date` is dates, `event` is text, and light loss is numerical.

I was not asking about types of variable (that is coming up); it is pretty clear that light loss is quantitative and `event` is categorical, but I don't know what you would call `date`. (I suggest a way of handling this later.)

If you used something like `read.table`, this won't have happened:

```
saturn2=read.table(url, header=T)
saturn2
```

```
##          date   event light_loss
## 1  1995-08-02 eclipse         65
## 2  1995-08-04 eclipse         61
## 3  1995-08-05  occult          1
## 4  1995-08-06 eclipse         56
## 5  1995-08-08 eclipse         46
## 6  1995-08-08  occult          2
## 7  1995-08-09  occult          9
## 8  1995-08-11  occult          5
## 9  1995-08-12  occult         39
## 10 1995-08-14  occult          1
## 11 1995-08-14 eclipse        100
## 12 1995-08-15  occult          5
## 13 1995-08-15  occult          4
## 14 1995-08-16  occult         13
## 15 1995-08-20  occult         11
## 16 1995-08-23  occult          3
## 17 1995-08-23  occult         20
## 18 1995-08-25  occult         20
## 19 1995-08-28  occult         12
```

It looks all right, but:

```
glimpse(saturn2)
```

```
## Observations: 19
## Variables: 3
## $ date       <fct> 1995-08-02, 1995-08-04, 1995-08-05, 1995-08-06, 199...
## $ event      <fct> eclipse, eclipse, occult, eclipse, eclipse, occult,...
## $ light_loss <int> 65, 61, 1, 56, 46, 2, 9, 5, 39, 1, 100, 5, 4, 13, 1...
```

What happened is that the text variables got turned into `factor`s. This is all right for `event` (since it really is a categorical variable), but not for `date`, which is, if anything, quantitative. If you leave it like this, your plot later with the dates on it will look terrible. This is a good reason *not* to use the `read` things with dots in them. When you use the `read_` things with an underscore in them, you know exactly what you're getting:

- numbers are read in as numbers
- text is read in as text
- if it looks like a date (in year-month-day format) it will be turned into a date.

(By the way, this is also a good reason to *always* record dates as year-month-day.)

If you insist on using `read.table`, you have some more work to do to bash things into shape:

```
saturn3=read.table(url, header=T, stringsAsFactors=F)
saturn3 %>% mutate(date=as.Date(date)) -> saturn3
saturn3
```

```
##          date   event light_loss
## 1  1995-08-02 eclipse         65
## 2  1995-08-04 eclipse         61
## 3  1995-08-05  occult          1
## 4  1995-08-06 eclipse         56
## 5  1995-08-08 eclipse         46
## 6  1995-08-08  occult          2
## 7  1995-08-09  occult          9
## 8  1995-08-11  occult          5
## 9  1995-08-12  occult         39
## 10 1995-08-14  occult          1
## 11 1995-08-14 eclipse        100
## 12 1995-08-15  occult          5
## 13 1995-08-15  occult          4
## 14 1995-08-16  occult         13
## 15 1995-08-20  occult         11
## 16 1995-08-23  occult          3
## 17 1995-08-23  occult         20
## 18 1995-08-25  occult         20
## 19 1995-08-28  occult         12
```

and

```
glimpse(saturn3)
```

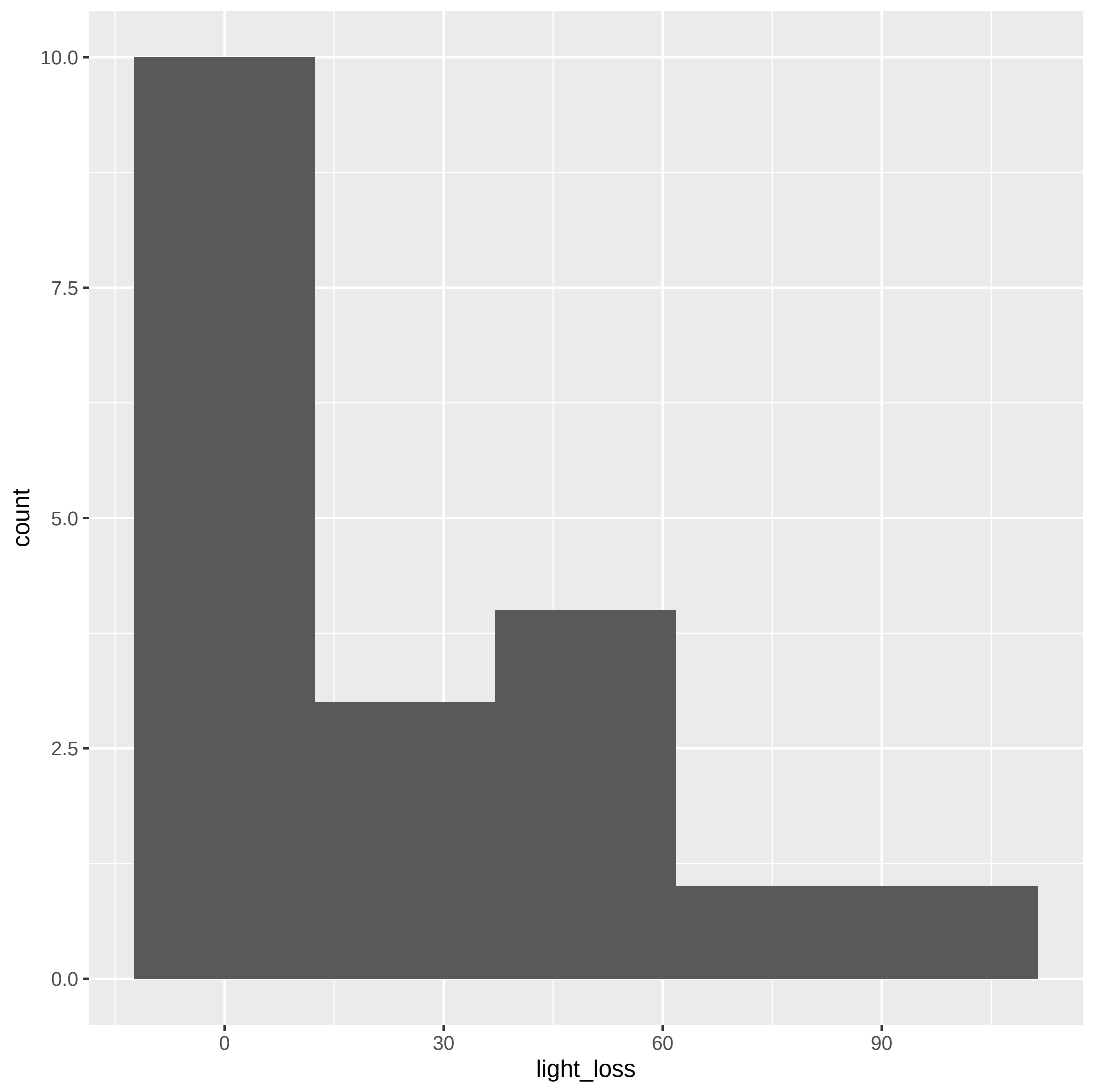
```
## Observations: 19
## Variables: 3
## $ date       <date> 1995-08-02, 1995-08-04, 1995-08-05, 1995-08-06, 19...
## $ event      <chr> "eclipse", "eclipse", "occult", "eclipse", "eclipse...
## $ light_loss <int> 65, 61, 1, 56, 46, 2, 9, 5, 39, 1, 100, 5, 4, 13, 1...
```

The first line keeps text as text, and the second line turns the (now-text) dates into dates that are real dates.

(c) (2 marks) Make a suitable plot of the light loss values (ignoring the other two variables). Comment briefly on the shape of the distribution.

Solution: One quantitative variable, so a histogram is the obvious choice. Remember that for a histogram, you need a number of bins (the default 30 is way too many). I think something like 5 bins will show the shape:

```
ggplot(saturn, aes(x=light_loss)) + geom_histogram(bins=5)
```

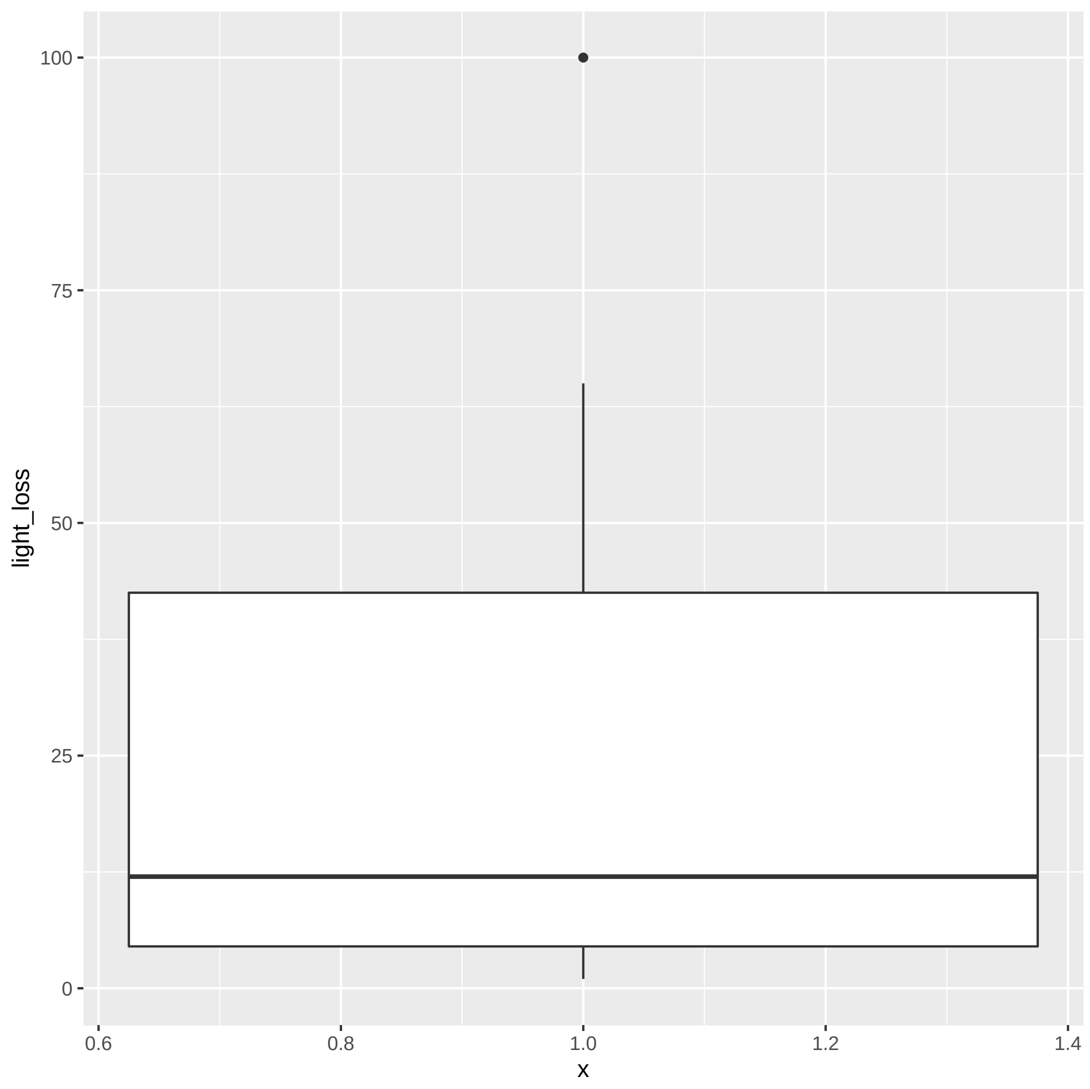


This is skewed to the right, because the long tail is to the right. (There are a lot of small light loss values, and a few very big ones.)

I think anything up to about 10 bins is OK (more than 10 definitely will not display the shape in a way you can interpret).

The other possibility is a one-group boxplot. This does not require a call about the number of bins, but *does* require you to remember that the grouping variable has to be something like a 1:

```
ggplot(saturn, aes(x=1, y=light_loss)) + geom_boxplot()
```

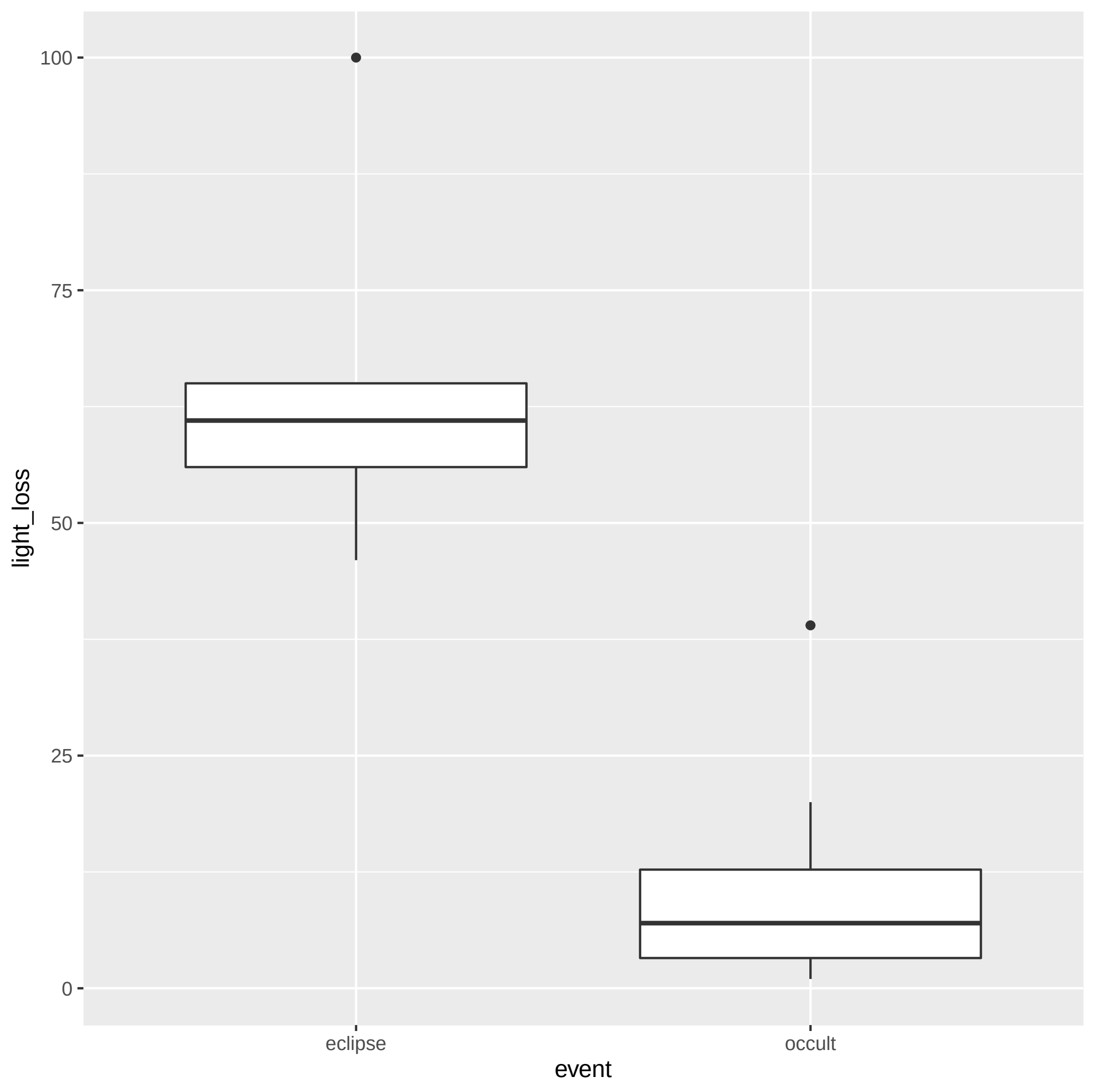


This also shows the right-skewness, with a long upper whisker and an upper outlier.

(d) (3 marks) Is the light loss clearly different for the two types of event? Make a graph that will enable you to assess this, and describe what you see.

Solution: One quantitative variable (light loss) and one categorical variable, event type, so the obvious thing is a boxplot. (For variety, I was trying to steer you away from a boxplot in the previous part.)

```
ggplot(saturn, aes(x=event, y=light_loss)) + geom_boxplot()
```



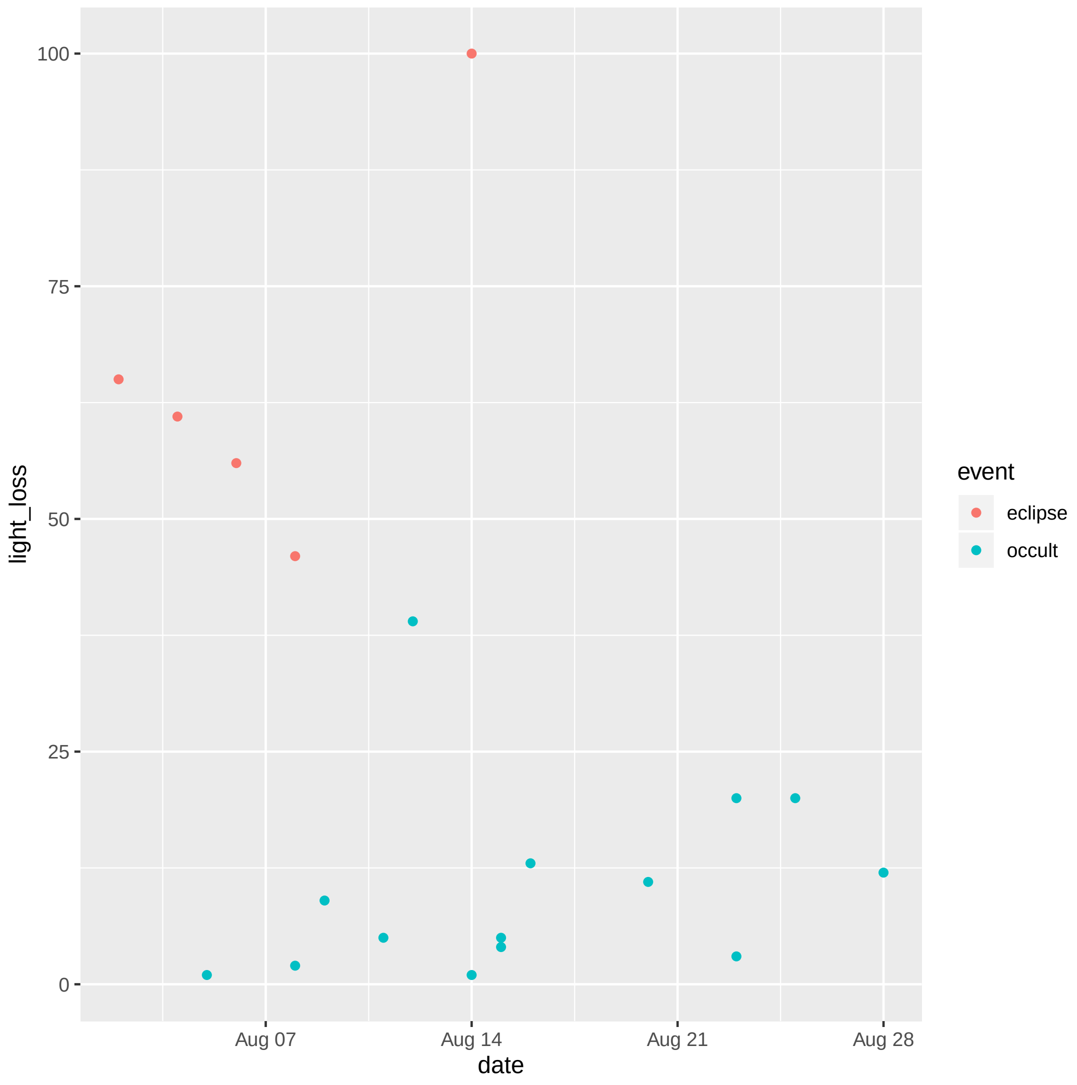
The two groups of observations are completely distinct (no overlap at all). Even the upper outlier in light loss for the occultations is less than all the light loss values for the eclipses.

Said briefer: the eclipses have clearly greater light loss than the occultations.

(e) (4 marks) Find a way to plot all three variables, treating date as quantitative. How does your conclusion of the previous part show up on this plot? Explain briefly.

Solution: Three variables, two quantitative and one categorical. This suggests a scatter plot with the categorical variable distinguished by something like colour. With a scatterplot you need to make some kind of call about what should be x and what should be y; the traditional thing is to have time on the x-axis, or you can reason that the light loss is an outcome and thus deserves to be y:

```
ggplot(saturn, aes(x=date, y=light_loss, colour=event)) +
    geom_point()
```



The light loss was higher for eclipses than for occultations, which shows up here as the red dots being higher than the blue ones.

Extra 1: it actually does make some kind of sense to treat dates as quantitative. Dates are "interval" (if not "ratio"): you can talk about the difference between two dates, but you can't multiply or divide them. The difference between two dates is a number of days (say), and you can also add a number of days to a date and get another date. This is enough to make dates a quantitative thing, and suitable to go on a scatterplot. (You can see that on my scatterplot the x-axis scale makes perfect sense, with the tick marks being seven days apart). What happens behind the scenes is that R represents dates as the number of days since January 1, 1970 (the so-called "Unix Epoch"), so that a date really is a quantitative thing. Time can be represented as a number of seconds after midnight, in the same kind of way, and date-and-times, like the date and time of the first lecture in this course, are represented as a (large) number of seconds since January 1, 1970.

To illustrate the dates thing:

```
mydates <- tibble(date_text = c("1966-04-13", "1970-01-01", "2001-09-11", "2019-09-03"))
mydates
```

```
## # A tibble: 4 x 1
##   date_text
##   <chr>
## 1 1966-04-13
## 2 1970-01-01
## 3 2001-09-11
## 4 2019-09-03
```

These are dates formatted as text in "ISO format", with the year first. Then I can do the following:

```
mydates %>% mutate(date_date = as.Date(date_text),
                   date_number= as.numeric(date_date))
```

```
## # A tibble: 4 x 3
##   date_text  date_date  date_number
##   <chr>      <date>           <dbl>
## 1 1966-04-13 1966-04-13       -1359
## 2 1970-01-01 1970-01-01           0
## 3 2001-09-11 2001-09-11       11576
## 4 2019-09-03 2019-09-03       18142
```

The second column is what R knows of as an actual date (what I gave you in the Saturn data), and the last column is the date as a number. You see that January 1, 1970 is indeed "day zero". Dates before that, like the first one, are negative numbers. The first day of classes in Fall 2019 is almost 50 years after the zero date, so should have a day number a bit less than

```
50*365.25
```

```
## [1] 18262.5
```

and does.

A date-time looks like this (we will see later a better way of doing the first `mutate`):

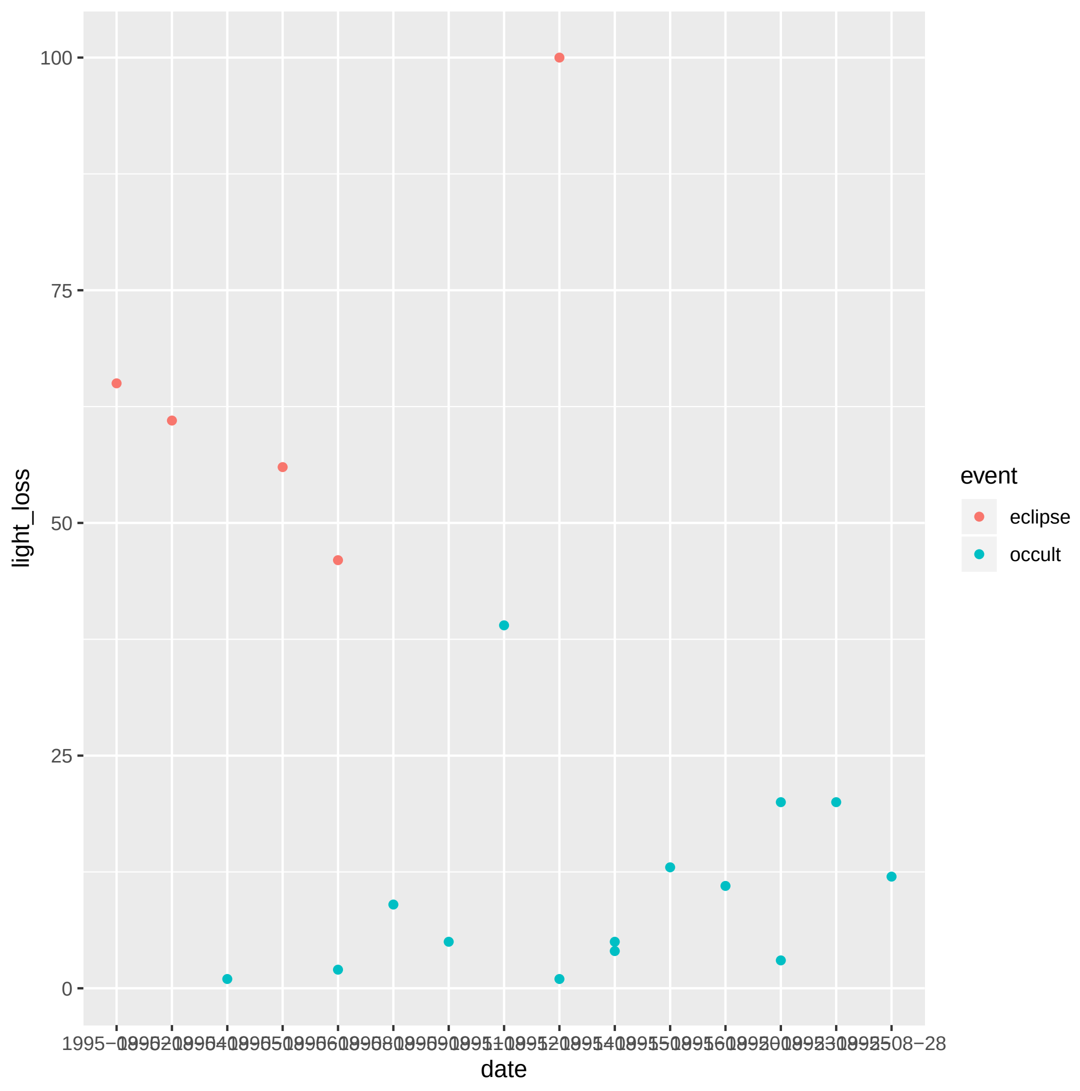
```
tibble(dt_text = "2019-09-03 13:00:00") %>%
  mutate(dt_dt=as.POSIXct(dt_text, tz="America/Toronto")) %>%
  mutate(dt_number=as.numeric(dt_dt))
```

```
## # A tibble: 1 x 3
##   dt_text             dt_dt                dt_number
##   <chr>               <dttm>                   <dbl>
## 1 2019-09-03 13:00:00 2019-09-03 13:00:00 1567530000
```

I was not kidding about the large number of seconds!

Extra 2: because `read_delim` handled the dates as dates, the x-axis is formatted nicely. If you were foolish enough to use `read.table`, this happens:

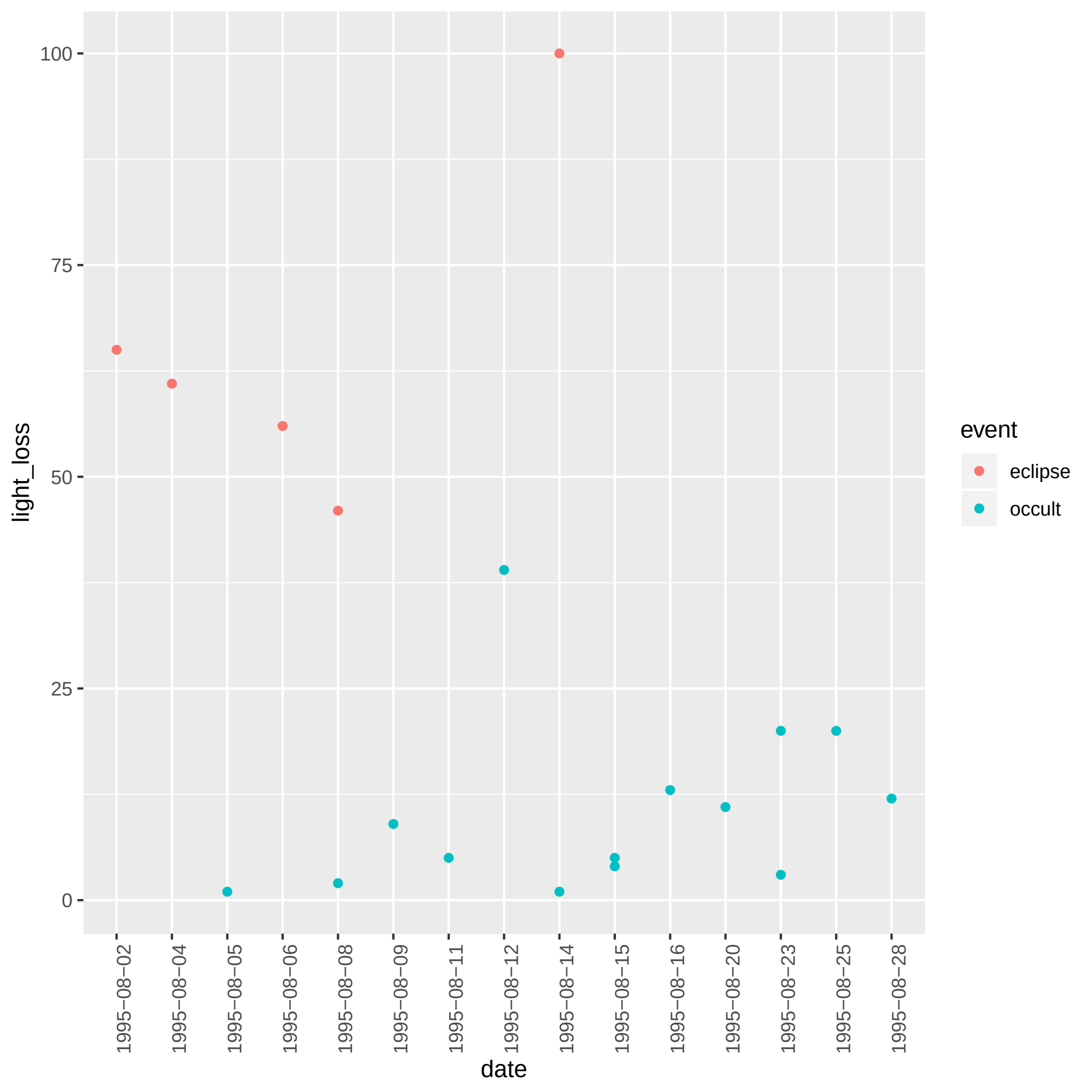
```
ggplot(saturn2, aes(x=date, y=light_loss, colour=event)) +
    geom_point()
```



The illegible stuff on the x-axis is all the different dates, printed individually (what happens on a scatterplot when you have a factor, that is, a categorical variable, on the x-axis).

If you really want to do it this way, you can, but you'll need to figure out how to make the dates readable. One way is to make them vertical rather than horizontal, which you'll have to find out how to do, thus:

```r
ggplot(saturn2, aes(x=date, y=light_loss, colour=event)) +
    geom_point() + theme(axis.text.x = element_text(angle=90))
```



This is still not quite right, though, because this graph makes it look as if the dates are equally spaced, which they are not (sometimes there are three or four days between events). The right way to do it is to have the dates as dates, as we did it before.