STAD29 / STA 1007 assignment 1

Due Tuesday Jan 14 at 11:59pm on Blackboard

Hand in the indicated questions. In preparation for the questions you hand in, it is worth your while to work through (or at least read through) the other questions as well.

What you hand in needs to include (i) your code, (ii) the output that your code produced and (iii) your comments on the output as asked for in the questions. The easiest way to get this is to use an R Notebook and preview the results (to HTML or Word or PDF) when you are done.

Hand in your work on Quercus. If you did STAC32 last fall, it's the same procedure. A reminder is here: `https://www.utsc.utoronto.ca/~butler/c32/quercus1.nb.html`

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 grader 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.

Before you start, you'll need this:

```
library(tidyverse)

## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

library(lubridate)

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

1. Work through, or at least read, Chapter 15 of PASIAS: `http://ritsokiguess.site/pasias/`

   Hand the next one in:

2. Manchester United is one of the most famous soccer clubs in England and indeed the world. Information about the players in the current squad (as at December 10, 2019) is here: `http://ritsokiguess.site/STAD29/manu.csv` (it's a CSV file). We are going to learn something about the ages of the players.

   (a) (2 marks) Read in the file and display some of the resulting data frame.

Solution: The usual thing:

```
my_url="http://ritsokiguess.site/STAD29/manu.csv"
man_united=read_csv(my_url)
## Parsed with column specification:
## cols(
##   number = col_double(),
##   name = col_character(),
##   nationality = col_character(),
##   date_of_birth = col_character(),
##   age = col_double(),
##   country_of_birth = col_character(),
##   place_of_birth = col_character(),
##   position = col_character(),
##   height = col_character(),
##   weight = col_character(),
##   foot = col_character()
## )
man_united
## # A tibble: 29 x 11
##    number name  nationality date_of_birth   age country_of_birth place_of_birth
##     <dbl> <chr> <chr>       <chr>         <dbl> <chr>            <chr>
##  1      1 Davi~ Spain       7 November 1~    29 Spain            Madrid
##  2     13 Lee ~ England     27 January 1~    36 England          Hemel Hempste~
##  3     22 Serg~ Argentina   22 February ~    32 Argentina        Bernardo de I~
##  4      2 Vict~ Sweden      17 July 1994     25 Sweden           Västerås
##  5      3 Eric~ Côte d'Ivo~ 12 April 1994    25 Côte d'Ivoire    Bingerville
##  6      4 Phil~ England     21 February ~    27 England          Preston
##  7      5 Harr~ England     5 March 1993     26 England          Sheffield
##  8     16 Faus~ Argentina   20 March 1990    29 Argentina        La Plata
##  9     18 Ashl~ England     9 July 1985      34 England          Stevenage
## 10     20 José~ Portugal    18 March 1999    20 Portugal         Braga
## # ... with 19 more rows, and 4 more variables: position <chr>, height <chr>,
## #   weight <chr>, foot <chr>
```

You'll most likely see more detail than that.

(b) (3 marks) What kind of thing is the column `date_of_birth`? Create a new column that contains the players' dates of birth as actual R dates, and display the old dates of birth alongside your new column (or at least the first few rows of them). Save your updated data frame.

Solution: The dates of birth are currently text, as evidenced by the `chr` at the top of the column, or the `col_character` in the output from `read_csv`.

To make these into proper R dates, we need to use something from `lubridate`. To figure out what, look at the dates as given: they are a day number, a month name and then a year, so that `dmy` will do the job. (It doesn't matter that the month is a name; R has a thing called a `locale` that tells it what language it is working in, and it will get things like month names from there.)

Thus:

```
man_united %>%
    mutate(date_of_birth_date = dmy(date_of_birth)) -> mu2
mu2 %>% select(starts_with("date"))
## # A tibble: 29 x 2
##    date_of_birth    date_of_birth_date
##    <chr>            <date>
##  1 7 November 1990  1990-11-07
##  2 27 January 1983  1983-01-27
##  3 22 February 1987 1987-02-22
##  4 17 July 1994     1994-07-17
##  5 12 April 1994    1994-04-12
##  6 21 February 1992 1992-02-21
##  7 5 March 1993     1993-03-05
##  8 20 March 1990    1990-03-20
##  9 9 July 1985      1985-07-09
## 10 18 March 1999    1999-03-18
## # ... with 19 more rows
```

You have a lot of freedom here in terms of choices for things. I thought it was easier to save the new data frame first, then look at the appropriate columns, but if you have something else that works, it's good.

The new column is correctly a `date`, and is displayed in ISO format with the year first, but you can check that, at least for the ones you see, the dates in both formats are the same.

Extra: the dates of birth are read in as text because they are not in the format year-month-day where everything is numbers, which is the only format that will get turned into dates automatically.

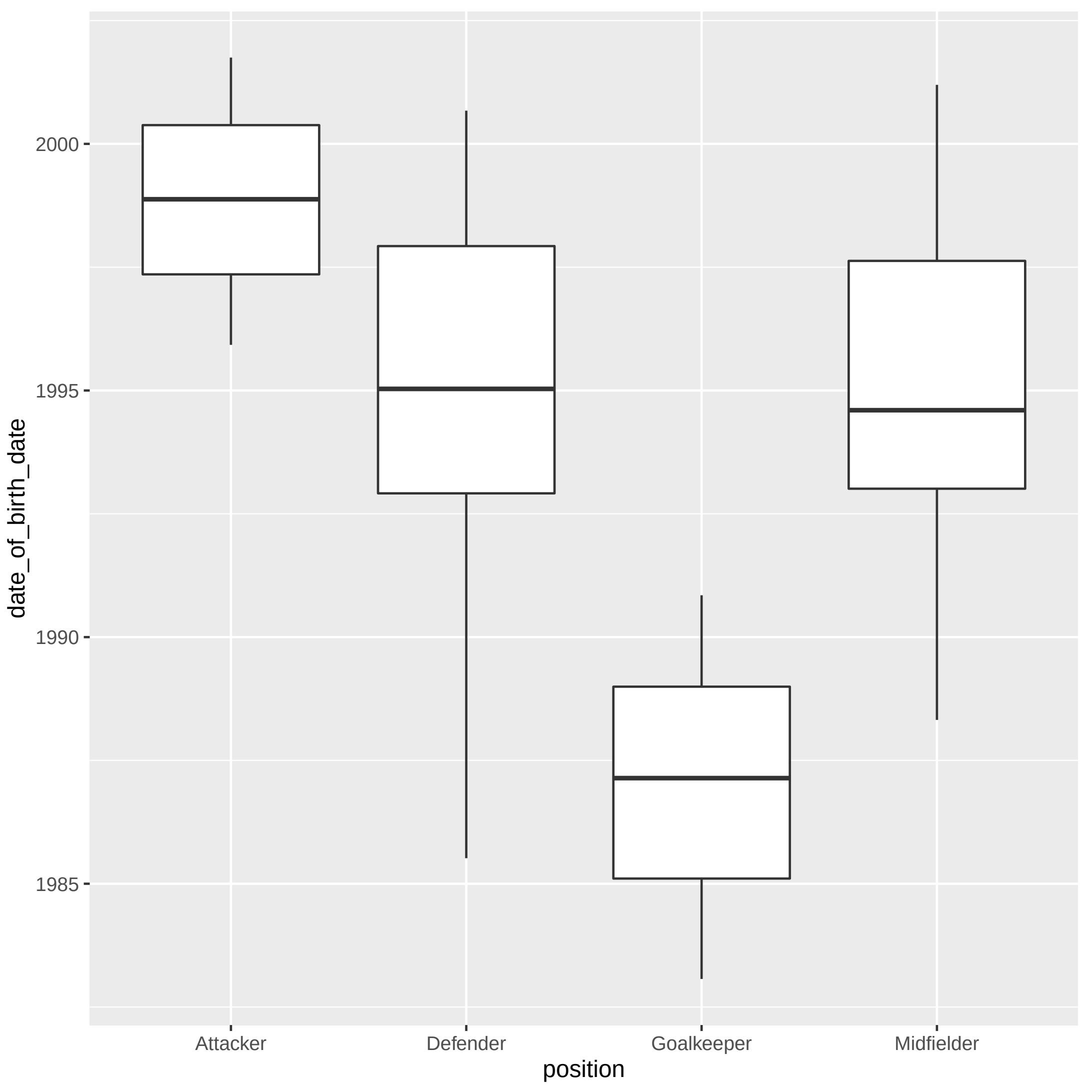
(c) (2 marks) Treating the new column of dates as quantitative, make a suitable plot of these with the players' `position` on the field. What do you see on the quantitative axis?

Solution: The positions are text, and if you look at the data, they are one of these:

```
mu2 %>% count(position)
## # A tibble: 4 x 2
##   position       n
##   <chr>      <int>
## 1 Attacker       4
## 2 Defender      12
## 3 Goalkeeper     3
## 4 Midfielder    10
```

So it makes sense to treat `position` as categorical, and hence to draw a boxplot. You might be curious about what will happen:

```
ggplot(mu2, aes(x=position, y=date_of_birth_date)) + geom_boxplot()
```



The *y*-axis is marked off in years, that is, on a date scale. (If the players had all been born in the same year, you would have seen months here as well.)

(d) (2 marks) Is there a position where the players tend to be older? Explain briefly.

Solution: The oldest players will have had the earliest birth dates, that is, they will be at the bottom of the plot. Thus, the goalkeepers tend to be older than the players in other positions.

If you know anything about soccer, you'll know that it is generally good to have an experienced goalkeeper, someone who knows how to be in the right place so that they don't have to move very far (or who knows what an opposing attacker is likely to do).

In this data set, the attackers are youngest, perhaps because you want someone fast playing up front.

(e) (3 marks) (This is to prepare you for the next thing.) Work out how many years old *you* are by using something like `as.Date("1966-04-13")` to turn your birth date into an R date, create a `period` from the interval from it to now (use `Sys.Date()` to get today's date), and pull out the number of (completed) years. (If you don't want to share your birth date, use any other date. I'm not checking.) Make sure to have the right number of brackets in the right places.

Solution: Here's the one-step solution, using the date above (which is my brother's birthday):

```
year(as.period(as.Date("1966-04-13") %--% Sys.Date()))
## [1] 53
```

If you don't like that, do it in steps, putting brackets around the appropriate things, and copying and pasting each line into the next one:

```
as.Date("1966-04-13")
## [1] "1966-04-13"
as.period(as.Date("1966-04-13") %--% Sys.Date())
## [1] "53y 9m 2d 0H 0M 0S"
year(as.period(as.Date("1966-04-13") %--% Sys.Date()))
## [1] 53
```

These are, respectively:

- my brother's birthday as a date
- the time between then and now as a period (note how it displays)
- the number of *years* in that. My brother is (at the time of writing) 53 years old.

I realize that this also works as a pipe:

```
as.Date("1966-04-13") %--% Sys.Date() %>%
    as.period() %>%
    year()
## [1] 53
```

but it's better for what we'll be doing (in a moment) to cope with all the brackets.

(f) (3 marks) Go back to the Manchester United players. Calculate a new column containing the age, in completed years, of each player as measured today (thus, for example, a player who is currently 29 years and some number of days old should be counted as 29 years old, even if the number of days is something like 364). Display your new column side by side with the one called `age`. (This uses the same technique that you used to calculate your own age in the previous part, except that you don't need `as.Date` because you converted the birth dates into R `Dates` in an earlier part.)

Solution: The reason for doing the previous part was to give you hints for this one.
I am just going to display my result, since I am going to add to my pipeline in a moment:

```
mu2 %>% mutate(
          my_age=year(as.period(date_of_birth_date %--% Sys.Date()))
        ) %>%
    select(age, my_age)

## # A tibble: 29 x 2
##       age my_age
##     <dbl>  <int>
##  1    29     29
##  2    36     36
##  3    32     32
##  4    25     25
##  5    25     25
##  6    27     27
##  7    26     26
##  8    29     29
##  9    34     34
## 10    20     20
## # ... with 19 more rows
```

(g) (3 marks) Display the names and (original) birth dates of the players whose age (in the original data frame) and whose age (as you calculated it) are different. What do these players have in common?

Solution: I am also displaying the players' ages (as in the database and as I calculated) for checking, but you don't need to do that.

Adding to my pipeline:

```
mu2 %>% mutate(
          my_age=year(as.period(date_of_birth_date %--% Sys.Date()))
        ) %>%
    filter(age != my_age) %>%
    select(name, age, my_age, date_of_birth, date_of_birth_date)

## # A tibble: 3 x 5
##   name                          age my_age date_of_birth    date_of_birth_date
##   <chr>                       <dbl>  <int> <chr>            <date>
## 1 Timothy Evans Fosu-Mensah     21     22 2 January 1998   1998-01-02
## 2 Jesse Lingard                 26     27 15 December 1992 1992-12-15
## 3 Andreas Hoelgebaum Pereira    23     24 1 January 1996   1996-01-01
```

The answer you get here depends on when precisely you run this, but the idea is that the original `age`s were calculated when I originally downloaded this data set (Dec 10, 2019), and the players whose ages were different are the ones who have had a birthday since then. (So your answer might be different depending on exactly how many players had birthdays between December 10 and the date you ran this.) "Their birthdays are between December 10 and now" is the answer I want.

Extra: I had a hell of a time with this, which goes to show how sticky dates are. I originally thought that what you did was to subtract the dates, obtaining in the jargon a "duration", and then you divided by `dyears(1)`. This doesn't *quite* work, as I (eventually) realized, because years are not all the same number of days (leap years). This way works out the number of days there are between the two dates you have and then gets the number of years between them *allowing for leap years*, so the answer is correct. (Despite the warning.) Doing it the way I originally thought with `dyears(1)` gave me some problems I couldn't resolve at first. I wrote this originally on December 11, and this way thought that Jesse Lingard's birthday (December 15) had already happened and he was already 27 years old, and when I did January 20, 2020 it thought that Lee Grant's birthday (January 27) had already happened. So I delved a bit further, and ended up rewriting the lecture notes. A couple of times.

3. Work through, or at least read, problems 13.12, 13.13, 13.14, and 13.18 in PASIAS: `http://ritsokiguess.site/pasias/`

   Hand the next one in:

4. In the sport of baseball, the pitcher has a very important role: to stop the batters of the other team from scoring runs, and thereby helping their team to win the game. One way that a pitcher can get a batter out is called a "strikeout"; this, roughly speaking, is done by throwing three accurate pitches that the batter cannot hit. We would suspect that a pitcher who has more strikeouts would also help their team to win more games, but is that actually true?

   The data in `http://ritsokiguess.site/STAD29/pitchers.txt` is from 40 pitchers in Major League Baseball in the 2011 season. Pitchers earn a win for themselves each time they help their team win a game. Technical definition here.

   (a) (2 marks) Read in the (space-delimited) data and confirm that you have the right number of rows and the right columns (the column `sos` contains the number of strikeouts).

Solution: Nothing new here at all. I even told you what kind of data file it was:

```
my_url <- "http://ritsokiguess.site/STAD29/pitchers.txt"
pitchers <- read_delim(my_url, " ")
## Parsed with column specification:
## cols(
##   wins = col_double(),
##   sos = col_double()
## )
pitchers
## # A tibble: 40 x 2
##     wins   sos
##    <dbl> <dbl>
##  1    21   248
##  2    12   158
##  3    19   220
##  4    13   191
##  5    17   238
##  6    16   158
##  7    24   250
##  8     8   134
##  9    18   198
## 10    10   197
## # ... with 30 more rows
```

As ever, call the data frame whatever you like.

Indeed, I have 40 pitchers (rows), and columns containing a number of wins and a number of strikeouts.

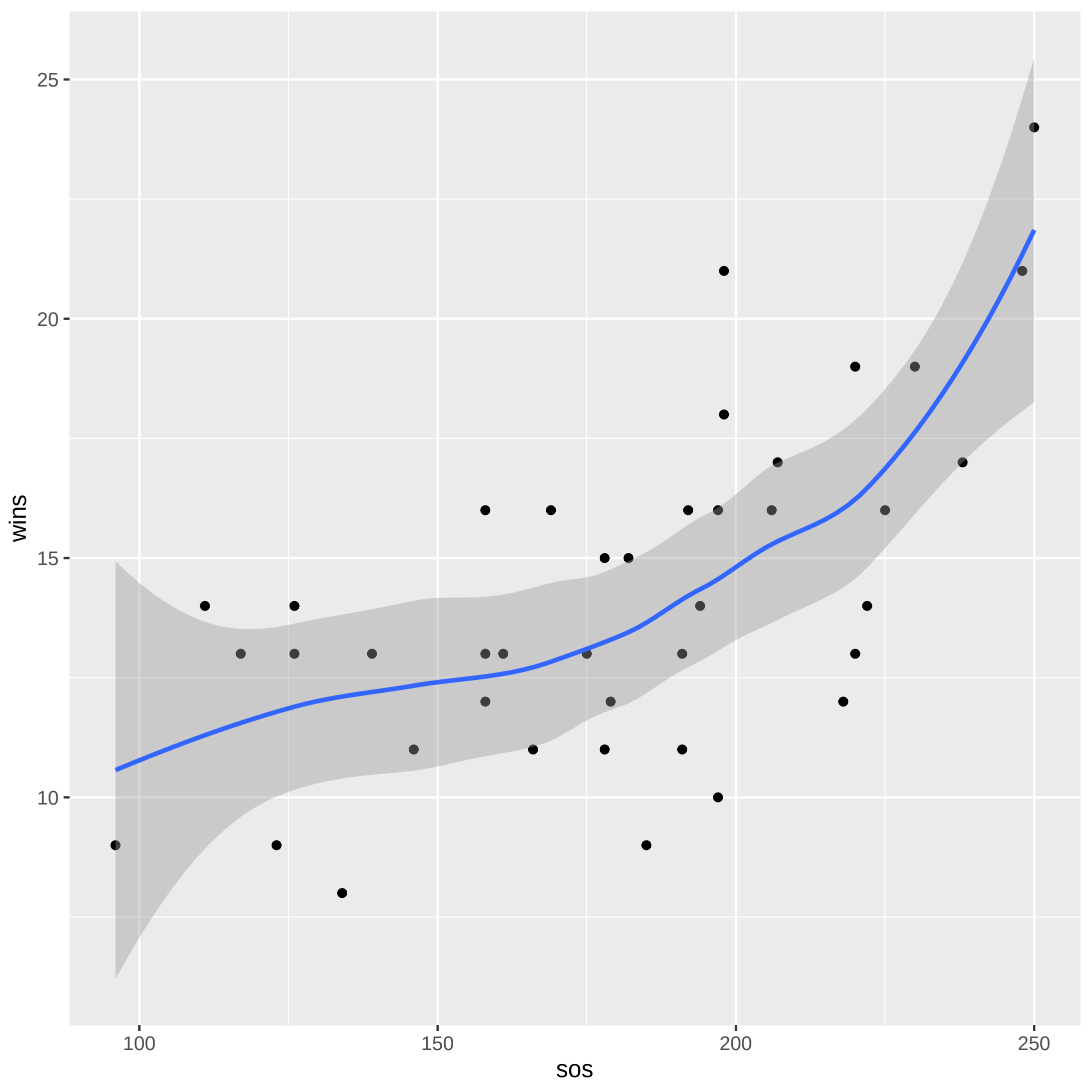
   (b) (2 marks) Fit a linear regression (predicting wins from strikeouts), and display the output.

Solution: Nothing very challenging here, I hope:

```
pitchers.1 <- lm(wins~sos, data=pitchers)
summary(pitchers.1)
##
## Call:
## lm(formula = wins ~ sos, data = pitchers)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.4479 -1.9829 -0.0056  2.1047  5.8755
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.98386    2.13331   1.867   0.0696 .
## sos          0.05656    0.01158   4.885  1.9e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.814 on 38 degrees of freedom
## Multiple R-squared:  0.3857,Adjusted R-squared:  0.3695
## F-statistic: 23.86 on 1 and 38 DF,  p-value: 1.903e-05
```

Extra: we are assuming a linear relationship here. I didn't ask you to check that, since the question will be long enough already, but:

```
ggplot(pitchers, aes(x=sos, y=wins)) + geom_point() + geom_smooth()
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



I would say there definitely is an upward trend on the scatterplot. Is it approximately linear? Make a call. I see something of an upward-opening curve, with (especially) the pitchers with the highest numbers of strikeouts (say, above 225) having more wins than you would expect if the relationship were straight.

For yourself, think about (i) whether you think the trend is up, down or non-existent (and if you like, how strong you think it is), and (ii) if you think it's a curve, something about what kind of curve it is.

In the grand scheme of things, if you think this relationship is a curve, then you ought to *fit* a curve, and base your confidence/prediction intervals on that.

(c) (3 marks) Obtain confidence intervals for the mean number of wins for pitchers that have 125 and 175 strikeouts, based on your regression of the previous part.

Solution: Remember the steps: create a data frame (which I, by habit, call `new`) containing the values you want to predict for. Then predict for them, getting the right kind of interval. Then display the results side by side with the values they're predictions for:

```
new <- tibble(sos=c(125, 175))
p <- predict(pitchers.1, new, interval="c")
cbind(new, p)
```

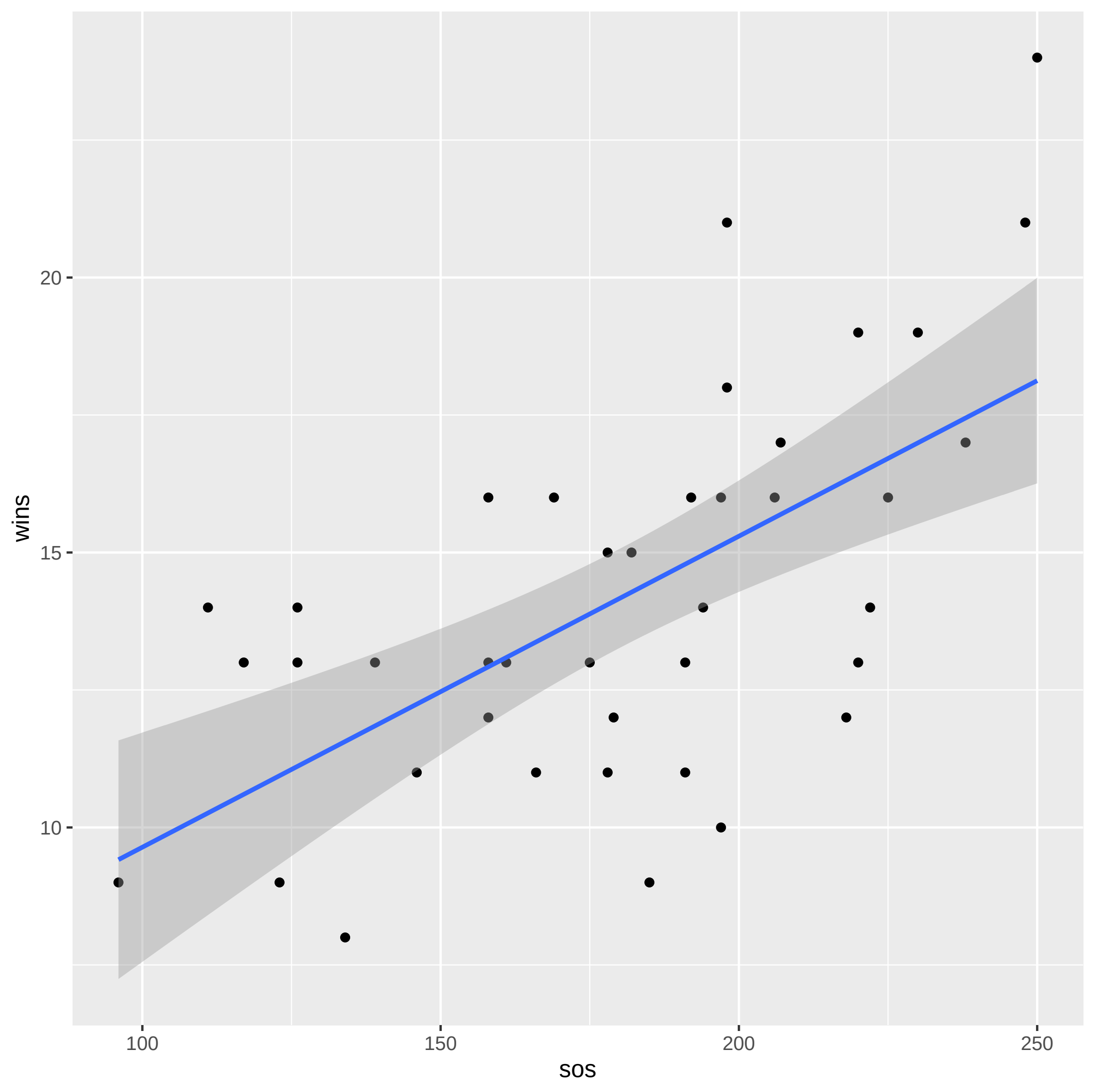
```
##   sos      fit       lwr      upr
## 1 125 11.05417  9.478053 12.63028
## 2 175 13.88229 12.973493 14.79108
```

(d) (2 marks) Which one of your intervals in the previous part is longer? Explain briefly how that makes sense.

Solution: The interval for 125 strikeouts is longer, over 3 wins long as compared to under 2 wins. This is because 125 is on the low side for a number of strikeouts (among these pitchers), and so there are not so many "nearby" pitchers to guide the prediction. (You can also say that 125 is "extreme" or "far from the mean".) For 175 strikeouts, on the other hand, this is pretty much in the middle of the pitchers, so there is a lot of nearby stuff to base a prediction on.

Extra: if you draw a scatterplot with the regression line on it, you get the customary grey envelope:

```
ggplot(pitchers, aes(x=sos, y=wins)) + geom_point() +
    geom_smooth(method="lm")
```



The grey envelope *is* the confidence interval for the mean response, so if you check how far up and down it goes at 125 and 175 strikeouts, you should get the same results, to the accuracy of the graph, as your `predict` did. Try it and see. You can see that 175 strikeouts is about where the interval is at its shortest, and if I'd asked you to do it for 100 strikeouts instead of 125, it would have been longer still.

If you happen to like formulas, the exact formula is at `https://newonlinecourses.science.psu.edu/stat462/node/126/`, near the top. The formula is complicated. Note that x_h is the value of x you're predicting for. The big square root controls how far the confidence interval will go up and down from the prediction, and the whole square root will be bigger if $(x_h - \bar{x})^2$ is bigger: that is to say, if the value of x you're predicting for is further from the mean of all the x-values. For our data, the mean number of strikeouts is between 175 and 200, and so the confidence interval for the mean number of wins is shortest there:

```
pitchers %>% summarize(mean_sos = mean(sos))
## # A tibble: 1 x 1
##   mean_sos
##      <dbl>
## 1     180.
```

(e) (2 marks) Find the prediction intervals for the numbers of wins for new pitchers (ones not in this data set) who have 125 and 175 strikeouts.

Solution: This is actually almost the same as two parts back, just changing a `c` to a `p`. The `new` data frame is the same as before (exactly), so you can re-use the one you had instead of defining it again:

```
new <- tibble(sos=c(125, 175))
p <- predict(pitchers.1, new, interval="p")
cbind(new, p)
##   sos      fit      lwr      upr
## 1 125 11.05417 5.143849 16.96448
## 2 175 13.88229 8.113960 19.65062
```

(f) (2 marks) For a pitcher or pitchers with 175 strikeouts, compare the lengths of the confidence and prediction intervals. Which one is longer? Explain briefly why that happened.

Solution: The confidence interval for the mean number of wins at 175 strikeouts is from 13.0 to 14.8 wins, and the prediction interval for the same thing is 8.1 to 19.7 wins. The prediction interval is much longer.

The reason for the difference is down to variability. If you estimate the mean number of wins for *all* pitchers who had 175 strikeouts, there are going to be some pitchers who had more wins than you would expect and some that had fewer. When you take the average, the unusually-high and unusually-low numbers of wins will cancel out and the *mean* number of wins will be quite predictable, especially if you have a lot of data.

On the other hand, if you are predicting a number of wins for a *single* pitcher, you might happen to get a pitcher with a lot more or fewer wins than you'd expect, and that has to be accounted for in the prediction interval. As you see here, knowing that this new pitcher has 175 strikeouts doesn't tell you much about *that* pitcher's likely number of wins, because individual pitchers have a lot of variability.

More discussion: there are two sources of variability to be concerned about here: (i) how much variability do we have in our estimation of where the line goes, and (ii) how much variability in number of wins do pitchers with the same number of strikeouts have? If you have a reasonable amount of data, (i) will be small, and the more data you have, the smaller it will be. On the other hand, (ii) you have no control over. If you look back at the scatterplot, you'd suspect that (ii) will be largish here.

The confidence interval for the mean response only uses (i), so you can make it smaller by having more data. The prediction interval is a combination of (i) and (ii), so, no matter how much data you have, if (ii) is large, you are, to put it not too politely, screwed.

Extra: that was where I decided to end the question, but there is some more exploration about that apparent curve on the scatterplot you drew back at the beginning. Since there is only one x-variable here, you could transform the x by adding a squared term, or the y by something like Box-Cox. (There is some kind of justification for the latter lurking in the background because the data are counts, or precisely the number of wins is a count, and a response variable that is a count often responds well to something like a square root transformation.)

Anyway, let's begin by adding a squared term:

```
pitchers.2 <- lm(wins~sos+I(sos^2), data=pitchers)
summary(pitchers.2)
```

```
##
## Call:
## lm(formula = wins ~ sos + I(sos^2), data = pitchers)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.6896 -1.8725  0.6367  1.6219  6.3501
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 19.7630781  7.7238338   2.559   0.0147 *
## sos         -0.1325271  0.0899513  -1.473   0.1491
## I(sos^2)     0.0005389  0.0002544   2.118   0.0409 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.693 on 37 degrees of freedom
## Multiple R-squared:  0.4521,Adjusted R-squared:  0.4225
## F-statistic: 15.27 on 2 and 37 DF,  p-value: 1.463e-05
```

The squared term is actually *just* significant, so it looks as if it was worth adding. (It is also positive, which makes sense given that it looks if anything like an upward-opening parabola with a minimum.) How do the confidence intervals for the mean response compare? The code is mostly copy-and-paste. I'm going to add a prediction for 225 strikeouts, because I suspect it's at the top where the action happens.

Here's the original linear one, slightly modified to include the new prediction:

```
new <- tibble(sos=c(125, 175, 225))
p <- predict(pitchers.1, new, interval="c")
cbind(new, p)
```

```
##   sos      fit       lwr      upr
## 1 125 11.05417  9.478053 12.63028
## 2 175 13.88229 12.973493 14.79108
## 3 225 16.71041 15.326447 18.09438
```

and then all we do is change the model and keep everything else the same:

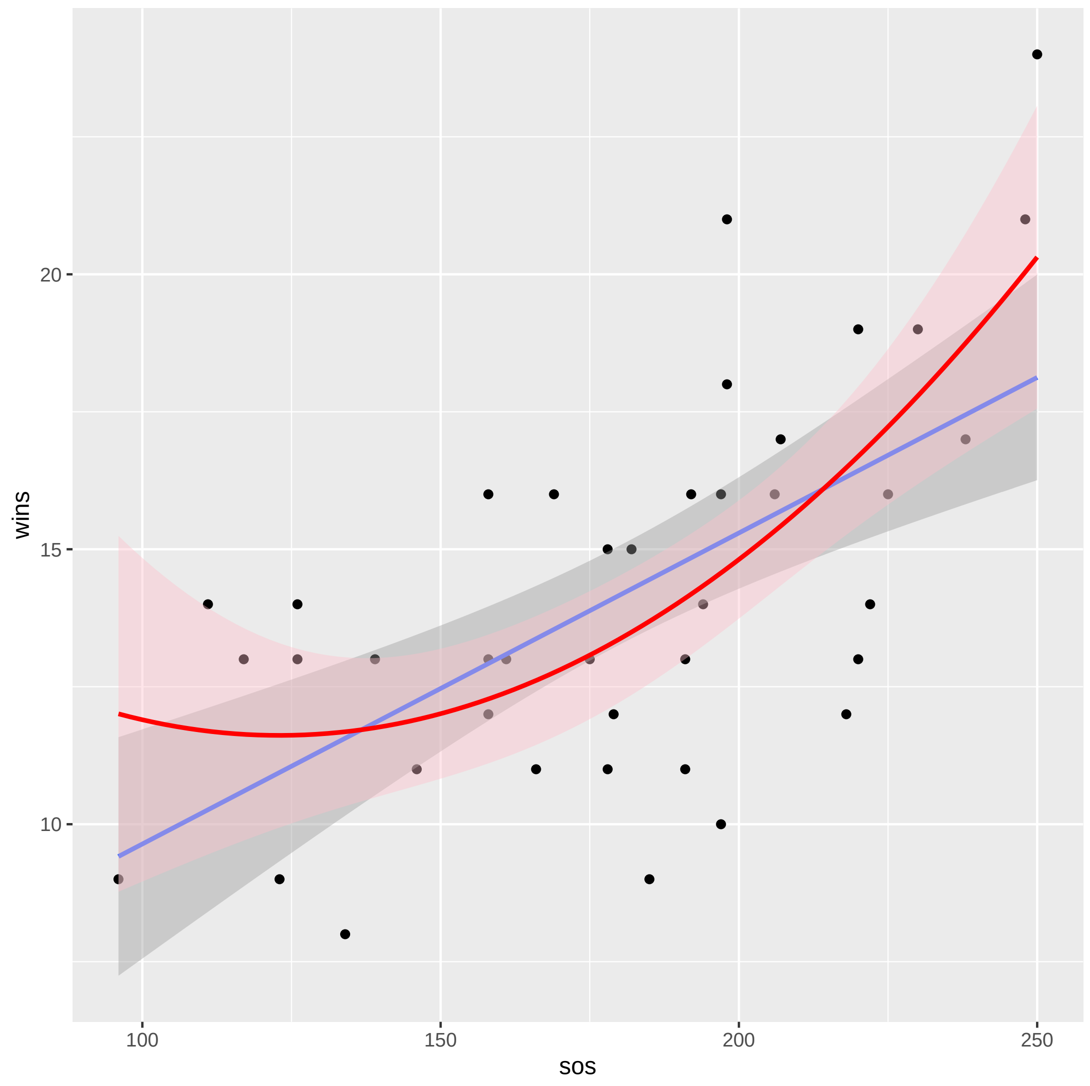
```
p <- predict(pitchers.2, new, interval="c")
cbind(new, p)
```

```
##   sos      fit      lwr      upr
## 1 125 11.61758 10.01452 13.22065
## 2 175 13.07480 11.91101 14.23860
## 3 225 17.22655 15.81190 18.64119
```

This says that the interval is *higher* for 125 strikeouts, *lower* for 175, and *higher* again for 225. I think this makes sense because the way the curve appears to go is to increase more slowly at the beginning and faster later.

I was curious about how you do smooths with parabolas on a graph. It turns out that `geom_smooth` has a `formula` option where you can put in something like what you would put into an `lm`, thus:

```r
ggplot(pitchers, aes(x=sos, y=wins)) + geom_point() +
    geom_smooth(method="lm") +
    geom_smooth(method="lm", formula = y~x+I(x^2), colour="red", fill="pink")
```



I'm a genius. Well, no, the `ggplot` hive mind is collectively a genius. I just know how to use Google. In the `formula` for the second smooth, you have to use the names `y` and `x`; the form of the formula is the same as for the second regression, but the actual things in it are `y` and `x` instead of the things that they stand for (you will get an error if you try to use `sos` and `wins` here. I did.) In `geom_smooth`, you can also configure the look of the smooth; the line itself is controlled by `colour`, and the colour of the *envelope* is controlled by `fill`. I thought a pink envelope would go nicely with a red smooth.[1] Compare the use of `colour` and `fill` when drawing boxplots: there, `colour` colours the outline, but `fill` colours the whole box.

The confidence intervals for the mean response from the two models are the grey envelope top and bottom for the line, and the pink envelope top and bottom for the curve. At 125 and 225, the pink envelope is higher, but at 175 it is lower, in agreement with our calculations.

The other way to go is to transform the number of wins. `boxcox`, which we use, lives in `MASS`. A problem is that `MASS` has a function called `select`, and if you want to use the `tidyverse` `select` (which actually lives in a package called `dplyr`), how is R going to know which `select` to use? What it actually does is to use the one in the *last* package loaded (without telling you), and if you don't know that and work around it, you can end up with error messages from the *wrong* `select` that are nearly impossible to debug!

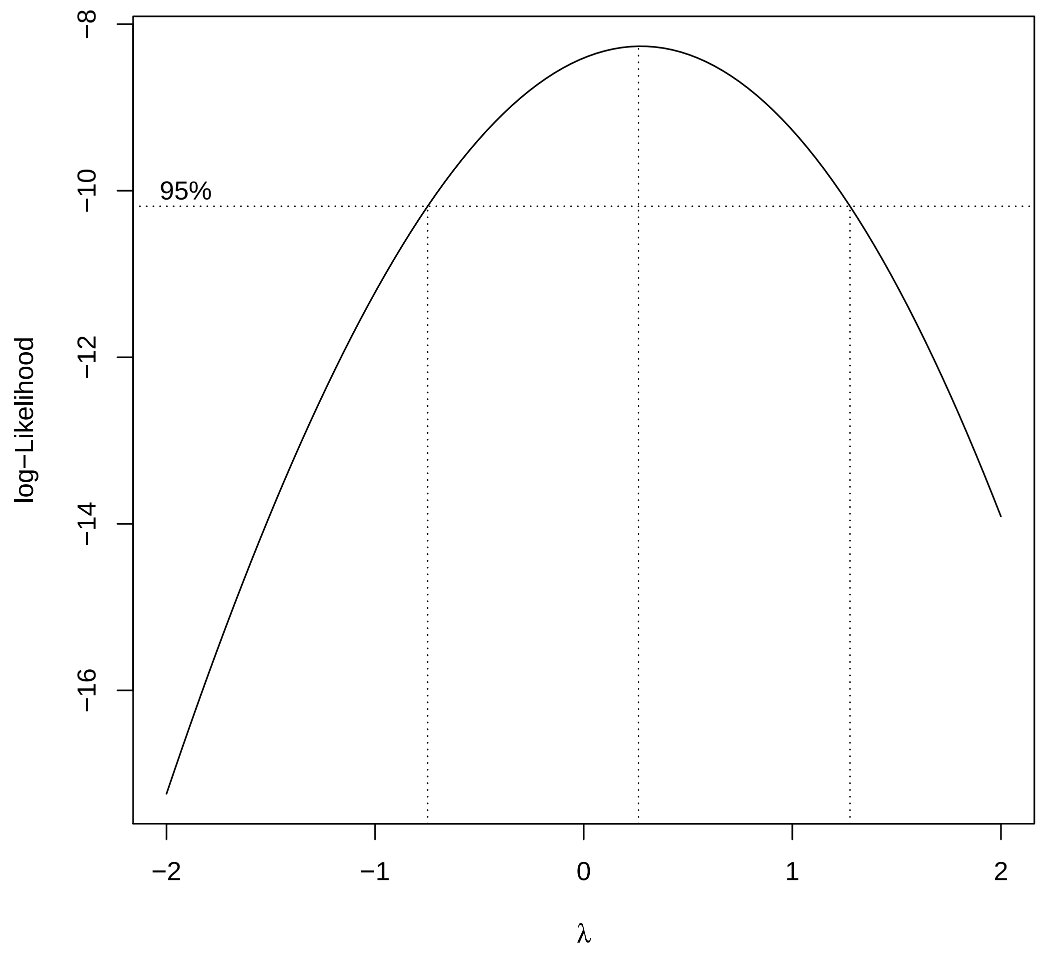
So, before we start playing with `boxcox`, let's load `MASS` and also another package called `conflicted` (which you'll probably have to install first):

```
library(MASS)
##
## Attaching package: 'MASS'
## The following object is masked from 'package:dplyr':
##
##     select
library(conflicted)
```

The warning message below `library(MASS)` means "you can no longer directly get at `select` from `tidyverse`". But `conflicted` will help us with that.

Let's run `boxcox` and see what happens:

```
boxcox(wins~sos, data=pitchers)
```

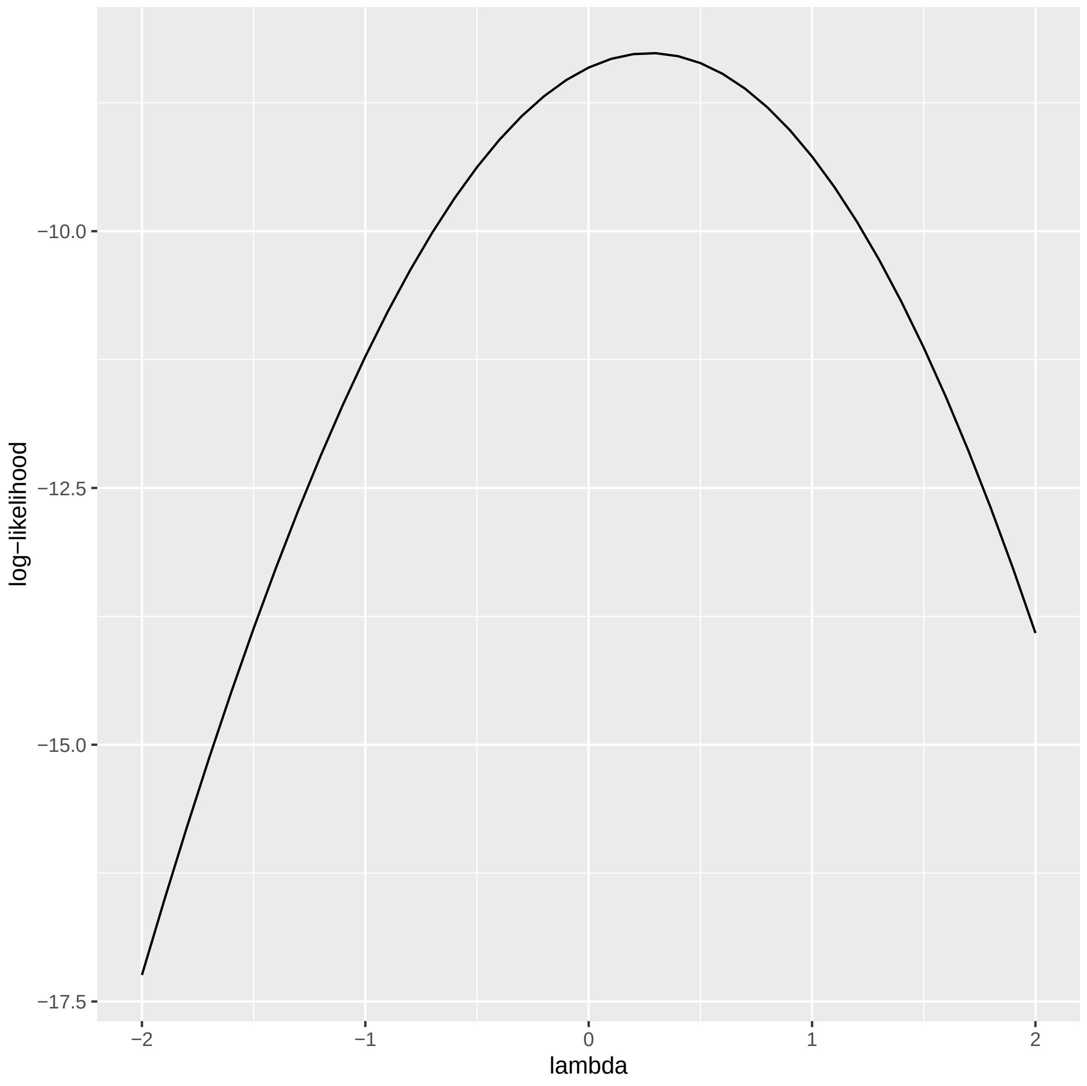


Aside (to this extra): this is a "base R plot". If you save the output from `boxcox` in a variable, you get something you can plot using `ggplot`, should you prefer to do that:

```
pitchers_boxcox <- boxcox(wins~sos, data=pitchers, plotit=F)
glimpse(pitchers_boxcox)
## List of 2
##  $ x: num [1:41] -2 -1.9 -1.8 -1.7 -1.6 -1.5 -1.4 -1.3 -1.2 -1.1 ...
##  $ y: num [1:41] -17.2 -16.5 -15.8 -15.1 -14.5 ...
```

I turned off the plot (with `plotit=F`) since we are going to make a new one. You see that the output from `boxcox` contains a thing called `x` and a thing called `y`, so, making a `tibble`, joining the points with lines, and making the axis labels something sensible:

```
as_tibble(pitchers_boxcox) -> d
d %>%
    ggplot(aes(x=x, y=y)) + geom_line() +
    xlab("lambda") + ylab("log-likelihood") -> g
g
```



(I'm saving this graph because I'm going to add something to it in a minute.)

But this doesn't have the confidence interval, which helps us to judge whether we should do a transformation at all. So we need to work that out first. There's some theory (based on a thing called the "likelihood ratio test") that says you make a 95% CI for a parameter by taking all the values that have a log-likelihood within 1.92 of the maximum, so let's work that out first.

I need to say that I mean the `filter` in `tidyverse` (not the base one that has to do with time series). See below for where this came from:

```
conflict_prefer("filter", "dplyr")
## [conflicted] Will prefer dplyr::filter over any other package
```

and then:

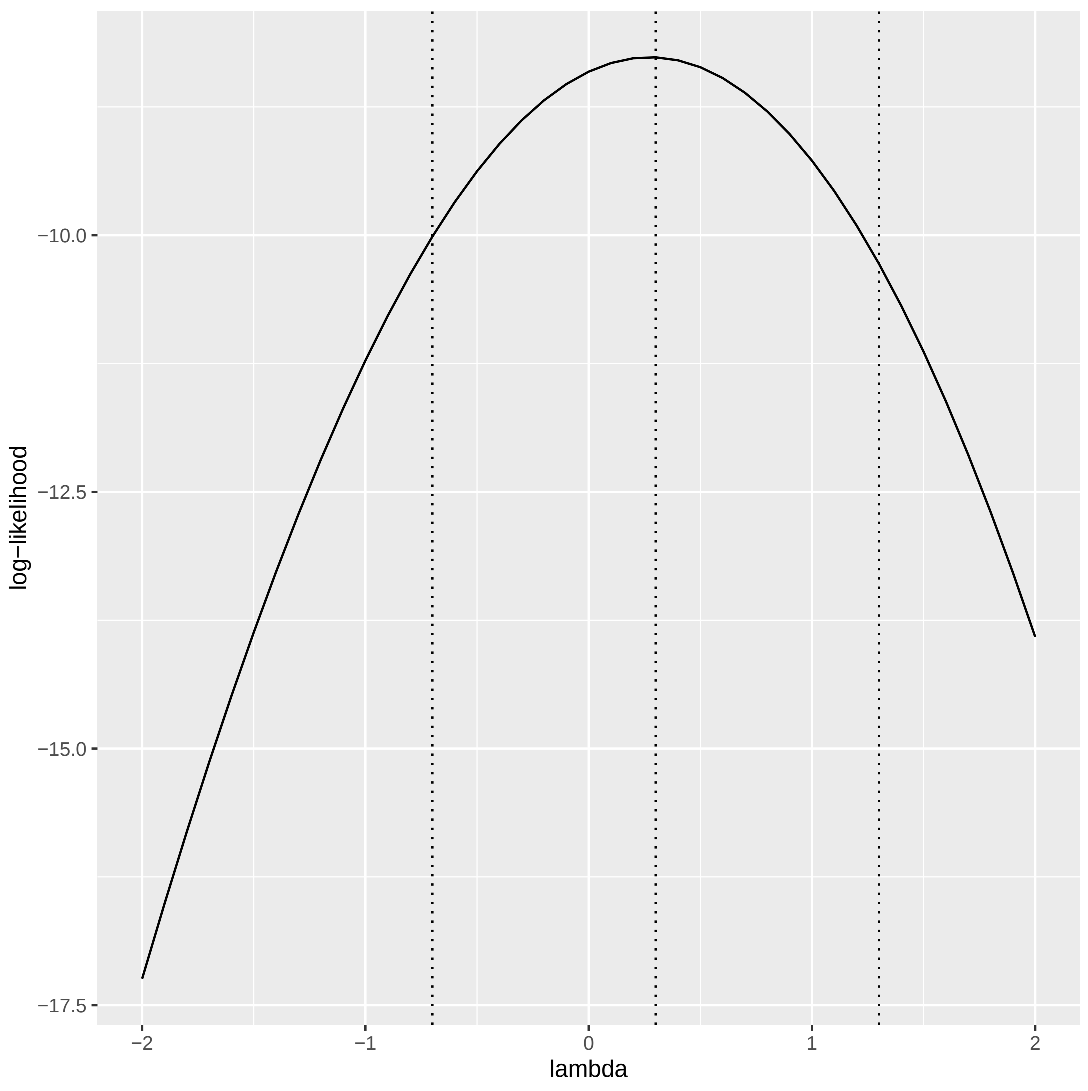
```
d %>% filter(y==max(y)) -> maxlik
maxlik
## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1   0.3 -8.27
```

And then, individually, the values of lambda whose likelihood is closest to the maximum minus 1.92 and are (i) less and (ii) greater than the lambda at the maximum.

```
d %>% filter(x<maxlik$x) %>%
    mutate(off = abs(maxlik$y-y-1.92)) %>%
    filter(off==min(off)) -> d1
d1
## # A tibble: 1 x 3
##       x     y   off
##   <dbl> <dbl> <dbl>
## 1  -0.7 -10.0 0.174
d %>% filter(x>maxlik$x) %>%
    mutate(off = abs(maxlik$y-y-1.92)) %>%
    filter(off==min(off)) -> d2
d2
## # A tibble: 1 x 3
##       x     y    off
##   <dbl> <dbl>  <dbl>
## 1   1.3 -10.3 0.0905
```

So the best value of λ is 0.3, and the confidence interval goes from -0.7 to 1.3. Let's add those to our graph:

```
g + geom_vline(xintercept=maxlik$x, linetype="dotted") +
    geom_vline(xintercept=d1$x, linetype="dotted") +
    geom_vline(xintercept=d2$x, linetype="dotted")
```



This is more or less the same as we had before. The two dotted lines don't hit the curve at exactly the same height because we only observed a certain number of different λ values. We could improve that by working out the log-likelihood at more different λ values, or by interpolating. But that's probably going too far.

Back to where we were:

"Do nothing" $\lambda = 1$ is within the confidence interval for λ. So we would be entirely justified in saying that there is no need for a transformation. However, square root ($\lambda = 0.5$) is also in the interval, so let's give that a try and see what it does:[2]

```
pitchers.3 <- lm(sqrt(wins)~sos, data=pitchers)
summary(pitchers.3)
##
## Call:
## lm(formula = sqrt(wins) ~ sos, data = pitchers)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.77239 -0.27381  0.02056  0.28239  0.71465
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.412825   0.281117   8.583 1.99e-10 ***
## sos         0.007349   0.001526   4.816 2.35e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3708 on 38 degrees of freedom
## Multiple R-squared:  0.379,Adjusted R-squared:  0.3627
## F-statistic: 23.19 on 1 and 38 DF,  p-value: 2.354e-05
```

Note that you can define transformations in the `lm`. Or, you could define a new column in the data frame:[3]

```
pitchers %>% mutate(sqrt_wins=sqrt(wins)) -> pitchers_trans
pitchers_trans %>% select(sos, sqrt_wins)
## Error: [conflicted] `select` found in 2 packages.
## Either pick the one you want with `::`
## * MASS::select
## * dplyr::select
## Or declare a preference with `conflict_prefer()`
## * conflict_prefer("select", "MASS")
## * conflict_prefer("select", "dplyr")
```

Wait, what? Nothing wrong with our `select`!

This is where `conflicted` shows its value. It recognizes that there is a select in `dplyr` (the `tidyverse` one that we want) and also one in `MASS` that we don't, and gives us an opportunity to say which one we prefer. If we didn't use `conflicted`, R would choose one of the `select`s for us (without saying that it had done so), and in fact in this case it would have picked the wrong one, which is the impenetrable error message I mentioned earlier.

Anyway, `conflicted` is very informative about what to do, so I think it is well worth the work up front to be clear about things (and make sure you get what you want). There are basically two approaches: you can either say which one you want with a package name and two colons (this is helpful if you might want to use both), or you can say which one you want `select` to mean henceforth (better if you only want to use one). The nice thing about this error message is that you can copy and paste the line you want. In our case, we always want `select` to mean the `tidyverse` one, so:

```
conflict_prefer("select", "dplyr")
## [conflicted] Will prefer dplyr::select over any other package
pitchers_trans %>% select(sos, sqrt_wins)
## # A tibble: 40 x 2
##       sos sqrt_wins
##     <dbl>     <dbl>
##  1   248       4.58
##  2   158       3.46
##  3   220       4.36
##  4   191       3.61
##  5   238       4.12
##  6   158       4
##  7   250       4.90
##  8   134       2.83
##  9   198       4.24
## 10   197       3.16
## # ... with 30 more rows
```

and now we are good. The approved way to use `conflicted` is to have a section up top with all your packages in it, including `library(conflicted)`, and as `conflicted` discovers any repeated function names, put a `conflict_prefer` line in that section as well, below the `library(conflicted)`. Or, skip the `conflict_prefer` line and use package name plus two colons plus function name in your code.

Thus this works (even without the `conflict_prefer`):

```
pitchers_trans %>% dplyr::select(sos, sqrt_wins)
## # A tibble: 40 x 2
##       sos sqrt_wins
##     <dbl>     <dbl>
##  1   248       4.58
##  2   158       3.46
##  3   220       4.36
##  4   191       3.61
##  5   238       4.12
##  6   158       4
##  7   250       4.90
##  8   134       2.83
##  9   198       4.24
## 10   197       3.16
## # ... with 30 more rows
```

and this doesn't:

```
pitchers_trans %>% MASS::select(sos, sqrt_wins)
## Error in MASS::select(., sos, sqrt_wins): unused arguments (sos, sqrt_wins)
```

Then:

```
pitchers.4 <- lm(sqrt_wins~sos, data=pitchers_trans)
summary(pitchers.4)
```

```
##
## Call:
## lm(formula = sqrt_wins ~ sos, data = pitchers_trans)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.77239 -0.27381  0.02056  0.28239  0.71465
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.412825   0.281117   8.583 1.99e-10 ***
## sos         0.007349   0.001526   4.816 2.35e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3708 on 38 degrees of freedom
## Multiple R-squared:  0.379,Adjusted R-squared:  0.3627
## F-statistic: 23.19 on 1 and 38 DF,  p-value: 2.354e-05
```

with the same results as before.

So, how do the confidence intervals for the mean response compare?

```
new
```

```
## # A tibble: 3 x 1
##     sos
##   <dbl>
## 1   125
## 2   175
## 3   225
```

```
p <- predict(pitchers.3, new, interval="c")
cbind(new, p)
```

```
##   sos      fit      lwr      upr
## 1 125 3.331451 3.123758 3.539144
## 2 175 3.698901 3.579144 3.818658
## 3 225 4.066352 3.883979 4.248724
```

Only this is not right: `p` is now a predicted *square root* of number of wins, so we have to undo the square root everywhere:

```
p
##        fit      lwr      upr
## 1 3.331451 3.123758 3.539144
## 2 3.698901 3.579144 3.818658
## 3 4.066352 3.883979 4.248724
p^2
##        fit       lwr      upr
## 1 11.09857  9.757865 12.52554
## 2 13.68187 12.810275 14.58215
## 3 16.53522 15.085293 18.05166
cbind(new, p^2)
##   sos      fit       lwr      upr
## 1 125 11.09857  9.757865 12.52554
## 2 175 13.68187 12.810275 14.58215
## 3 225 16.53522 15.085293 18.05166
```

We got lucky here: `p` contains the predictions and the endpoints of the CIs, all of which we wanted to un-transform, and R understands squaring something to mean squaring each element of the something, so it all works.[4]

Let's compare that with the original regression, which I have to do again since I didn't save it before:

```
p <- predict(pitchers.1, new, interval="c")
cbind(new, p)
##   sos      fit       lwr      upr
## 1 125 11.05417  9.478053 12.63028
## 2 175 13.88229 12.973493 14.79108
## 3 225 16.71041 15.326447 18.09438
```

This is not quite the same as adding strikeouts-squared: the middle interval is lower as before, but the first interval is *shorter* and the third interval is lower as well (and also a bit longer). (In the grand scheme of things, there is not much difference between the intervals from the square-rooted wins and those from the original regression.)

You might reasonably prefer taking the square root of the number of wins (over adding strikeouts-squared) because, with a positive slope as here, as strikeouts increases, wins will always increase, which makes sense from a baseball point of view. If you look back at the strikeouts-squared predictions (the red line on the graph earlier), you'll see at the very left it actually goes up again, which makes no baseball sense.

Hand in your answers to the questions with marks attached to them (questions 2 and 4, here). I want to see your code, output and comments; a good way to get those is to use an R Notebook and Preview it (producing an HTML or PDF or Word file that you hand in).

Notes

[1]If you're colour-blind, you might find it difficult to distinguish everything. If that's you, let me know and we'll work on something you can actually see.

[2]So is $\lambda = 0$ or log, which you could also try. The data are not very revealing about what transformation would be good.

[3]This is a slightly artificial use of `select`, but I wanted to show you what happened.

[4]This would work with any transformation, for example `exp` to undo `log`.