Assignment 3

Instructions (same as for STAC32): Make an R Notebook and in it answer the question below. When you are done, hand in on Quercus the *output* from Previewing (or Knitting) your Notebook. 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 reason to contact the instructor while working on the assignments is to report something missing like a data file that cannot be read.

1. In 1985, a survey was carried out in El Salvador. 3185 married women were asked about their current use of contraception, classified as "sterilization", "other" or "none". Each woman's age was also recorded. This was originally done as an age group, but for you the age given is the middle of the age group, which we will treat as quantitative. (We are thus assuming that all the women in the 25-29 age group were actually exactly 27.5 years old, which of course they are not.)

   The data are in http://ritsokiguess.site/STAD29/el_salvador.csv. There are three columns, age, contraception method and the number of surveyed women of that age (group) who used that contraception method.

   (a) Read in and display the data. Is there one woman or more than one woman per row of the dataset?

   Solution:

   As usual:

```
my_url <- "http://ritsokiguess.site/STAD29/el_salvador.csv"
el_salvador <- read_csv(my_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   age = col_double(),
##   contraception = col_character(),
##   count = col_double()
## )
```

```
el_salvador
```

```
## # A tibble: 21 x 3
##      age contraception count
##    <dbl> <chr>         <dbl>
##  1  17.5 Ster              3
##  2  17.5 Other            61
```

```
##  3  17.5 None            232
##  4  22.5 Ster             80
##  5  22.5 Other           137
##  6  22.5 None            400
##  7  27.5 Ster            216
##  8  27.5 Other           131
##  9  27.5 None            301
## 10  32.5 Ster            268
## # ... with 11 more rows
```

There were seven age groups and three contraception methods, so 21 rows in total.

It is important at this stage to think here about how many women are represented on each row of the dataset: more than one, as many as are in the `count` column. So this is like, in lecture, the coal-miners or the variation of brand-preferences that had the data on 65 lines, and, looking ahead, to make it work you will need a `weights=` on the model-fitting line.

Extra: there is of course a data-tidying story here. I copy-pasted the data from my source and it looked like this after a little bit of editing:

```
txt <- "
Age Ster Other None All
15-19 3 61 232 296
20-24 80 137 400 617
25-29 216 131 301 648
30-34 268 76 203 547
35-39 197 50 188 435
40-44 150 24 164 338
45-49 91 10 183 284
"
```

I decided to store it as a piece of text, since I knew that a piece of text works just the same as a file for reading-in purposes. These values are separated by single spaces, so:

```
els <- read_delim(txt, " ")
els
```

```
## # A tibble: 7 x 5
##   Age    Ster Other  None   All
##   <chr> <dbl> <dbl> <dbl> <dbl>
## 1 15-19     3    61   232   296
## 2 20-24    80   137   400   617
## 3 25-29   216   131   301   648
## 4 30-34   268    76   203   547
## 5 35-39   197    50   188   435
## 6 40-44   150    24   164   338
## 7 45-49    91    10   183   284
```

There was actually an extra header, Contraception, spanning the Ster through None columns, but these things are a pain to deal with, so I edited it out.

There is a package called `datapasta` that lets you paste things straight into a dataframe (it recognizes what you have pasted), but I have had mixed success with that, so I did it this way.

This you will recognize as wide format (we will want the frequencies all in one column), so you can expect that we have a `pivot_longer` in our future. But there are some other things:

- the age groups are two numeric ages separated by a dash
- we want to work out the *middle* of each age group
- we don't need (or, really, even want) those totals in the `All` column.

Getting the ends of the age ranges is a job for `separate`; a dash (minus sign) is one of the things `separate` knows about, so:

```
els %>% separate(Age, into=c("age_lower", "age_upper"), convert=TRUE)
```

```
## # A tibble: 7 x 6
##   age_lower age_upper  Ster Other  None   All
##       <int>     <int> <dbl> <dbl> <dbl> <dbl>
## 1        15        19     3    61   232   296
## 2        20        24    80   137   400   617
## 3        25        29   216   131   301   648
## 4        30        34   268    76   203   547
## 5        35        39   197    50   188   435
## 6        40        44   150    24   164   338
## 7        45        49    91    10   183   284
```

The first time, my two age columns came out as text, but I realized that I was going to be doing arithmetic with them, so I made sure they were numbers. `convert` turns the separated things into whatever they look like, here numbers.

For what I was going to have you do, I didn't need the ages at the ends of the intervals, but rather the midpoints. This seems like just averaging the two `age` columns. But: the 15-19 age group includes everyone from those who just turned 15 to those a day short of 20, so the midpoint is at half of $15 + 19$ *plus one*:

```
els %>% separate(Age, into=c("age_lower", "age_upper"), convert=TRUE) %>%
  mutate(age_mid = (age_lower + age_upper + 1) / 2)
```

```
## # A tibble: 7 x 7
##   age_lower age_upper  Ster Other  None   All age_mid
##       <int>     <int> <dbl> <dbl> <dbl> <dbl>   <dbl>
## 1        15        19     3    61   232   296    17.5
## 2        20        24    80   137   400   617    22.5
## 3        25        29   216   131   301   648    27.5
## 4        30        34   268    76   203   547    32.5
## 5        35        39   197    50   188   435    37.5
## 6        40        44   150    24   164   338    42.5
## 7        45        49    91    10   183   284    47.5
```

Next, keep what we want, the `age_mid` column, which we'll call `age` now,[1] and the frequencies except for `All`:

```
els %>% separate(Age, into=c("age_lower", "age_upper"), convert=TRUE) %>%
  mutate(age_mid = (age_lower + age_upper + 1) / 2) %>%
  select(age=age_mid, Ster:None)
```

```
## # A tibble: 7 x 4
##     age  Ster Other  None
##   <dbl> <dbl> <dbl> <dbl>
## 1  17.5     3    61   232
## 2  22.5    80   137   400
```

```
## 3  27.5   216   131   301
## 4  32.5   268    76   203
## 5  37.5   197    50   188
## 6  42.5   150    24   164
## 7  47.5    91    10   183
```

and *now* we get to do our `pivot_longer`:

```
els %>% separate(Age, into=c("age_lower", "age_upper"), convert=TRUE) %>%
  mutate(age_mid = (age_lower + age_upper + 1) / 2) %>%
  select(age=age_mid, Ster:None) %>%
  pivot_longer(-age, names_to="contraception", values_to="count") -> el_salvador
el_salvador
```

```
## # A tibble: 21 x 3
##      age contraception count
##    <dbl> <chr>         <dbl>
##  1  17.5 Ster              3
##  2  17.5 Other            61
##  3  17.5 None            232
##  4  22.5 Ster             80
##  5  22.5 Other           137
##  6  22.5 None            400
##  7  27.5 Ster            216
##  8  27.5 Other           131
##  9  27.5 None            301
## 10  32.5 Ster            268
## # ... with 11 more rows
```

and this is what I saved for you.

(b) What kind of response variable do we have? What makes this a multinomial model? Explain briefly.

Solution:

The response variable is the kind of contraception, in the `contraception` column. This is categorical with three categories. The categories are just labels; they are not ordered in any way. Hence a multinomial model is called for, to predict the probability of a woman using each of the kinds of contraception given how old she is.

In short:

- categorical response
- three categories (more than two)
- categories not ordered (just labels).

(c) Fit a suitable multinomial model, predicting contraception from age *and* age-squared, and display the output.

Solution:

Load the `nnet` package (installing it first if needed), and then use `multinom`, which goes like `polr`. You can have or not have a `factor()` around `contraception`; it works either with or

without.[2]

```
el_salvador.1 <- multinom(contraception ~ age + I(age^2),
                          weights = count, data = el_salvador)
```

```
## # weights:  12 (6 variable)
## initial  value 3477.107894
## iter  10 value 2883.272291
## final  value 2883.136389
## converged
```

```
summary(el_salvador.1)
```

```
## Call:
## multinom(formula = contraception ~ age + I(age^2), data = el_salvador,
##     weights = count)
##
## Coefficients:
##       (Intercept)       age     I(age^2)
## Other   -4.549838 0.2640771 -0.004758144
## Ster   -12.618330 0.7097288 -0.009732882
##
## Std. Errors:
##        (Intercept)         age     I(age^2)
## Other 0.0004850755 0.007198081 0.0002166951
## Ster  0.0003634951 0.006137073 0.0001616953
##
## Residual Deviance: 5766.273
## AIC: 5778.273
```

As with an ordinal response model, it is not (yet) clear what this means.

You *need* the `weights=` part. This is how the model knows how many women are represented by each line of the data file. Without it, any results you get further down will be *nonsense*, and you will need to be able to see that they are nonsense. (Without the `weights`, the assumption is that each row of the dataframe corresponds to *one* woman, which it very much does not.) Catch this before you hand the assignment in, when you are reading through it to check that it makes sense.

(d) Demonstrate that the model with age-squared in it fits significantly better than one with just age.

Solution:

The most obvious thing to try is `drop1` on the model that contains age-squared:

```
drop1(el_salvador.1, test = "Chisq")
```

```
## trying - age
```

```
## Error in if (trace) {: argument is not interpretable as logical
```

This is that mysterious one that does not work for reasons that I do not understand. Hint: if you want to demonstrate this for yourself in an R Markdown document (or R Notebook), the fact that this step produces an error may stop the whole thing from knitting. If this happens to you, you can edit the top line of this code chunk to say `r, error=TRUE` inside the curly

brackets, which will show this error and also continue to process the whole document.

For even more unfathomable reasons, `step` *does* work:[3]

```
step(el_salvador.1, test = "Chisq")
```

```
## Start:  AIC=5778.27
## contraception ~ age + I(age^2)
##
## trying - age
## # weights:  9 (4 variable)
## initial  value 3477.107894
## iter  10 value 3038.597253
## iter  10 value 3038.597253
## final  value 3038.597253
## converged
## trying - I(age^2)
## # weights:  9 (4 variable)
## initial  value 3477.107894
## iter  10 value 3019.887931
## iter  10 value 3019.887930
## final  value 3019.887930
## converged
##            Df      AIC
## <none>      6 5778.273
## - I(age^2)  4 6047.776
## - age       4 6085.195
```

```
## Call:
## multinom(formula = contraception ~ age + I(age^2), data = el_salvador,
##     weights = count)
##
## Coefficients:
##       (Intercept)       age     I(age^2)
## Other   -4.549838 0.2640771 -0.004758144
## Ster   -12.618330 0.7097288 -0.009732882
##
## Residual Deviance: 5766.273
## AIC: 5778.273
```

Taking out nothing is the best thing to do, meaning that age-squared needs to stay in the model, meaning that a model with it in is significantly better than one without it. If you can get this to work *and* explain why it indicates that, that's good.

A perhaps simpler way conceptually, though requiring a bit more work, is to use `anova`. This means comparing a model with age-squared to a model with just age, which you *don't have yet*, so you have to fit that first:

```
el_salvador.2 <- update(el_salvador.1, .~. - I(age^2))
```

```
## # weights:  9 (4 variable)
## initial  value 3477.107894
## iter  10 value 3019.887931
## iter  10 value 3019.887930
```

```
## final  value 3019.887930
## converged
```

```
anova(el_salvador.2, el_salvador.1, test = "Chisq")
```

```
## Likelihood ratio tests of Multinomial Models
##
## Response: contraception
##             Model Resid. df Resid. Dev   Test    Df LR stat. Pr(Chi)
## 1             age        38   6039.776
## 2 age + I(age^2)        36   5766.273 1 vs 2     2 273.5031       0
```

`anova` compares the fit of *any* two models; they don't have to be regressions. Strictly speaking, it's not an analysis of variance but a likelihood ratio test. The null hypothesis is that the two models fit equally well; here, that is resoundingly rejected, and thus the model with the squared term fits significantly better.

A tactical issue: I got the model with just `age` by using `update`, which I think is better than getting it by writing another `multinom`, because then you'd need to write `data=` and `weights=` again. Or you could copy, paste, and edit, I guess.

The `anova`, I think, is the best way to answer this question. Here, it is the other way around from usual; normally you'd fit the model with just `age` *first*, see that there is something wrong with it, and then add age-squared. That is how I originally conceived this question, but then it went up to part (k), and I figured you would like me better if I shortened it some!

(e) For ages between 17.5 and 47.5 (inclusive) in steps of 5, obtain predicted probabilities that a woman of that age will use each of the contraceptives, and display each of the predictions appropriately.

Solution:

The obvious way to think about this is to make a new data frame with the required ages in it, run `predict`, then use `cbind` to display the predictions beside the ages they are predictions for. (Just displaying the predictions is pointless until we know what they are predictions for.) That goes like this:

```
new <- tibble(age = seq(17.5, 47.5, 5))
new
```

```
## # A tibble: 7 x 1
##     age
##   <dbl>
## 1  17.5
## 2  22.5
## 3  27.5
## 4  32.5
## 5  37.5
## 6  42.5
## 7  47.5
```

You remember, I hope, that `seq` does "from here to there in steps of that". Then feed this into `predict`:

```
p1 <- predict(el_salvador.1, new, type = "probs")
```

and display side-by-side:

```
pred1 <- cbind(new, p1)
pred1
```

```
##    age      None      Other       Ster
## 1 17.5 0.7741180 0.19365022 0.03223174
## 2 22.5 0.6376070 0.23062839 0.13176458
## 3 27.5 0.4895363 0.20182346 0.30864026
## 4 32.5 0.3917329 0.14510492 0.46316214
## 5 37.5 0.3809132 0.09993131 0.51915546
## 6 42.5 0.4734423 0.06934326 0.45721446
## 7 47.5 0.6733663 0.04340388 0.28322983
```

I gave this a name, because we will be using it again later (to make a graph out of). I also gave the predictions here a number 1, because I will have a number 2 later. (It is likely that you won't, in which case a name without a number is fine.)

I didn't ask for comment since we are going to make a graph shortly, but all the predicted probabilities seem to fluctuate with age (they seem to go down and up or up and down). This is not a kind of fluctuation we've seen much of before, and it is because we have age-squared in the model. Later on I compare what happens if you have *without* age-squared, and the predictions look quite different.

Another way you might go (which needs some care) is to recognize that these are the same ages as in the original dataframe, and therefore you can run `predict` without making a `new` dataframe:

```
p1a <- predict(el_salvador.1, type = "probs")
cbind(el_salvador, p1a)
```

```
##     age contraception count      None      Other       Ster
## 1  17.5          Ster     3 0.7741180 0.19365022 0.03223174
## 2  17.5         Other    61 0.7741180 0.19365022 0.03223174
## 3  17.5          None   232 0.7741180 0.19365022 0.03223174
## 4  22.5          Ster    80 0.6376070 0.23062839 0.13176458
## 5  22.5         Other   137 0.6376070 0.23062839 0.13176458
## 6  22.5          None   400 0.6376070 0.23062839 0.13176458
## 7  27.5          Ster   216 0.4895363 0.20182346 0.30864026
## 8  27.5         Other   131 0.4895363 0.20182346 0.30864026
## 9  27.5          None   301 0.4895363 0.20182346 0.30864026
## 10 32.5          Ster   268 0.3917329 0.14510492 0.46316214
## 11 32.5         Other    76 0.3917329 0.14510492 0.46316214
## 12 32.5          None   203 0.3917329 0.14510492 0.46316214
## 13 37.5          Ster   197 0.3809132 0.09993131 0.51915546
## 14 37.5         Other    50 0.3809132 0.09993131 0.51915546
## 15 37.5          None   188 0.3809132 0.09993131 0.51915546
## 16 42.5          Ster   150 0.4734423 0.06934326 0.45721446
## 17 42.5         Other    24 0.4734423 0.06934326 0.45721446
## 18 42.5          None   164 0.4734423 0.06934326 0.45721446
## 19 47.5          Ster    91 0.6733663 0.04340388 0.28322983
## 20 47.5         Other    10 0.6733663 0.04340388 0.28322983
## 21 47.5          None   183 0.6733663 0.04340388 0.28322983
```

Giving this as your answer reveals that you are *not thinking*. There are three rows with each age, and the predictions for each of them are the same. Also, there is no value in showing the `contraception` and `count` columns, because those are not part of the predictions. So, use `distinct` like this to get one prediction for each age, keeping all the other columns, and then get rid of the ones you don't want:

```
cbind(el_salvador, p1a) %>%
  distinct(age, .keep_all = TRUE) %>%
  select(age, None:Ster)
```

```
##     age      None      Other       Ster
## 1  17.5 0.7741180 0.19365022 0.03223174
## 4  22.5 0.6376070 0.23062839 0.13176458
## 7  27.5 0.4895363 0.20182346 0.30864026
## 10 32.5 0.3917329 0.14510492 0.46316214
## 13 37.5 0.3809132 0.09993131 0.51915546
## 16 42.5 0.4734423 0.06934326 0.45721446
## 19 47.5 0.6733663 0.04340388 0.28322983
```

This last is a good answer, but you need to get all the way to it. Here is another way of doing the same thing, perhaps simpler:

```
el_salvador %>%
  filter(contraception == "Ster") -> new1
p1b <- predict(el_salvador.1, new1, type = "probs")
cbind(new1, p1b) %>%
  select(age, None:Ster) -> preds1b
preds1b
```

```
##    age      None      Other       Ster
## 1 17.5 0.7741180 0.19365022 0.03223174
## 2 22.5 0.6376070 0.23062839 0.13176458
## 3 27.5 0.4895363 0.20182346 0.30864026
## 4 32.5 0.3917329 0.14510492 0.46316214
## 5 37.5 0.3809132 0.09993131 0.51915546
## 6 42.5 0.4734423 0.06934326 0.45721446
## 7 47.5 0.6733663 0.04340388 0.28322983
```

The strategy here is to grab only ages with *one* of the contraception methods (it doesn't matter which one you pick), having recognized that each age appears once in the original data with each contraception method. Then predict for each of those ages, put the predictions next to the ages they are for, and tidy up the resulting dataframe by removing the columns you don't need (best).

(f) Make a dataframe that shows the *observed proportion* of women of each age that use each type of contraception. (This has nothing to do with any fitted model, so start with the original data.)

Solution:

Start with the original data frame:

```
el_salvador
```

```
## # A tibble: 21 x 3
```

```
##      age contraception count
##    <dbl> <chr>         <dbl>
##  1  17.5 Ster              3
##  2  17.5 Other            61
##  3  17.5 None            232
##  4  22.5 Ster             80
##  5  22.5 Other           137
##  6  22.5 None            400
##  7  27.5 Ster            216
##  8  27.5 Other           131
##  9  27.5 None            301
## 10  32.5 Ster            268
## # ... with 11 more rows
```

We want a `count` divided by a `sum(count)` for each `age`. "For each age" ought to suggest to you that you should group by age first:

```
el_salvador %>%
  group_by(age) %>%
  mutate(proportion = count / sum(count)) -> observed
observed
```

```
## # A tibble: 21 x 4
## # Groups:   age [7]
##      age contraception count proportion
##    <dbl> <chr>         <dbl>      <dbl>
##  1  17.5 Ster              3     0.0101
##  2  17.5 Other            61     0.206
##  3  17.5 None            232     0.784
##  4  22.5 Ster             80     0.130
##  5  22.5 Other           137     0.222
##  6  22.5 None            400     0.648
##  7  27.5 Ster            216     0.333
##  8  27.5 Other           131     0.202
##  9  27.5 None            301     0.465
## 10  32.5 Ster            268     0.490
## # ... with 11 more rows
```

You can do a quick check that the three proportions for each age seem to add up to 1 (it looks if they do). You might have to experiment with what to put in the `group_by` (just `age`). Also note that this is not a `summarize`: you want a proportion for *every* row of the original dataframe. If you do `summarize`, you will lose some columns (the ones in neither the `group_by` nor the `summarize`) that you really want to keep:

```
el_salvador %>%
  group_by(age) %>%
  summarize(proportion = count / sum(count))
```

```
## `summarise()` has grouped output by 'age'. You can override using the `.groups` argument.
```

```
## # A tibble: 21 x 2
## # Groups:   age [7]
##      age proportion
##    <dbl>      <dbl>
```

```
##  1  17.5     0.0101
##  2  17.5     0.206
##  3  17.5     0.784
##  4  22.5     0.130
##  5  22.5     0.222
##  6  22.5     0.648
##  7  27.5     0.333
##  8  27.5     0.202
##  9  27.5     0.465
## 10  32.5     0.490
## # ... with 11 more rows
```

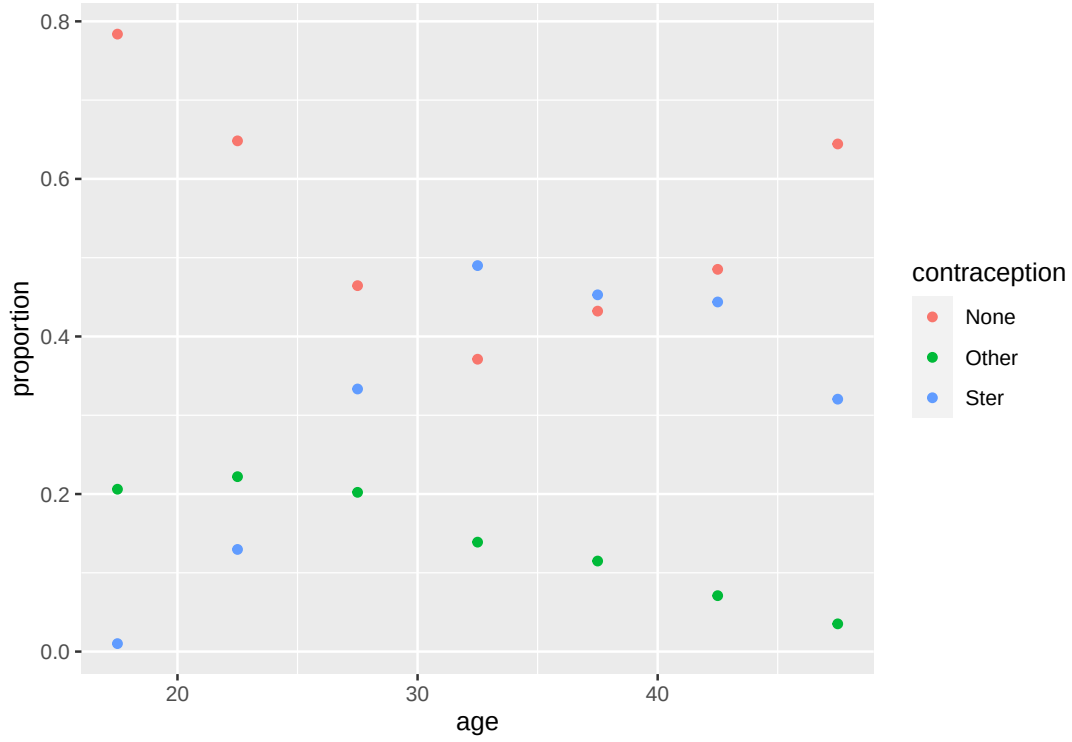
specifically, the names of the contraception methods.

(g) Make a suitable graph that contains both the observed proportions and the predicted probabilities, one that you can use to see how well the model fits the data. Pay some attention to the way the predicted probabilities are laid out.

Solution:

This will require a bit of thinking, since we will be using both the observed proportions from the previous part (plotted as points) and the predicted probabilities, that I called `preds1`, *which are in the wrong format*. Let's get the easiest stuff out of the way first:

```
ggplot(observed, aes(x=age, y=proportion, colour=contraception)) + geom_point()
```



To this we will be adding the predictions, which currently look like this:

```
pred1
```

```
##    age      None      Other       Ster
## 1 17.5 0.7741180 0.19365022 0.03223174
## 2 22.5 0.6376070 0.23062839 0.13176458
## 3 27.5 0.4895363 0.20182346 0.30864026
## 4 32.5 0.3917329 0.14510492 0.46316214
## 5 37.5 0.3809132 0.09993131 0.51915546
## 6 42.5 0.4734423 0.06934326 0.45721446
## 7 47.5 0.6733663 0.04340388 0.28322983
```

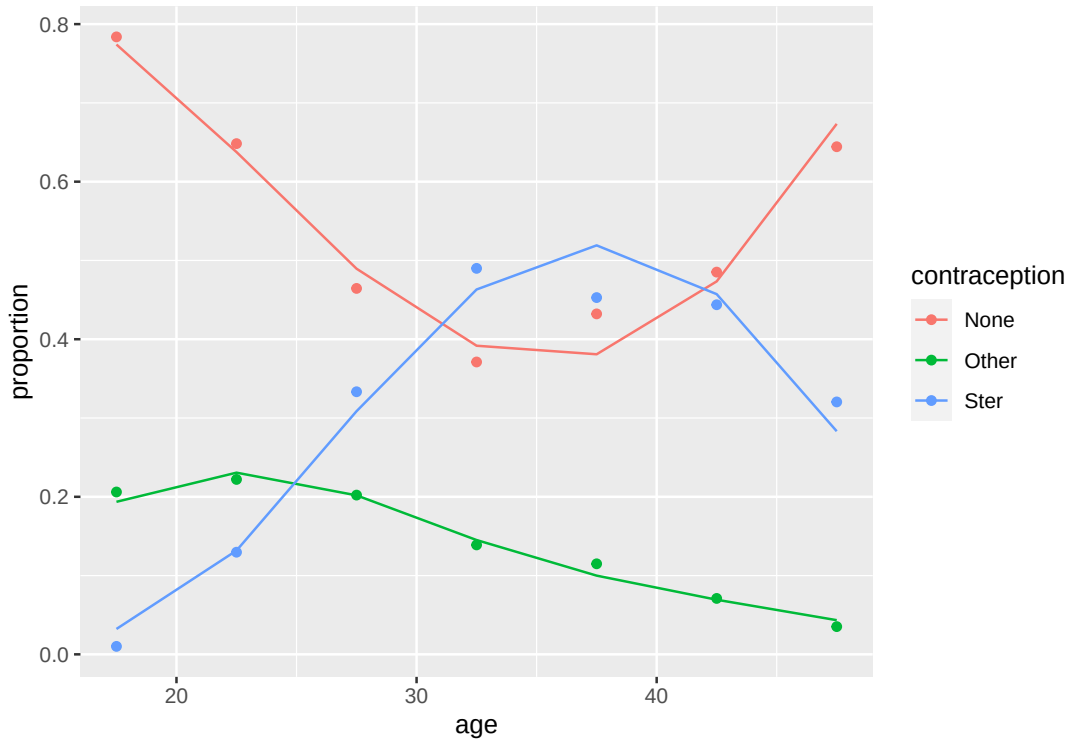
This is wide, and we want longer:

```
pred1 %>% pivot_longer(-age, names_to = "contraception", values_to = "prob") -> pred1_long
pred1_long
```

```
## # A tibble: 21 x 3
##      age contraception   prob
##    <dbl> <chr>          <dbl>
##  1  17.5 None          0.774
##  2  17.5 Other         0.194
##  3  17.5 Ster          0.0322
##  4  22.5 None          0.638
##  5  22.5 Other         0.231
##  6  22.5 Ster          0.132
##  7  27.5 None          0.490
##  8  27.5 Other         0.202
##  9  27.5 Ster          0.309
## 10  32.5 None          0.392
## # ... with 11 more rows
```

I gave this a name too, since I want to add it to the plot as lines. This means adding a `geom_line` but with a `data=`, since we are using a different dataframe. In the `aes`, you need to give anything that is different from what is in the `ggplot`. In my case, that is only the `y`, which is now `prob`; in my `pivot_longer`, I made sure the `contraception` had the same name as before (best). Thus:

```
ggplot(observed, aes(x=age, y=proportion, colour=contraception)) + geom_point() +
  geom_line(data = pred1_long, aes(y=prob))
```



(h) Comment briefly on how well the data fits the model, and the effect of including the age-squared term in the regression.

Solution:

It looks as if the fitted curves correspond well to the up-and-down or down-and-up pattern of the data; the curves appear to be close to the points.

In the other examples we've seen, most of the predicted probabilities keep going down or keep going up. That is what happens when you do *not* have a squared term; the shape has to be that kind of thing. It's linear only in the log-odds, not in the probabilities, so the trends are not linear, but they tend not to bend like these ones do. That's the consequence of having age-squared in the model; the data seem to bend like this, so we also need a model that can include those bends, such as this one.

Extra: the way I originally wrote this question, I had you fit a model with just `age` first:

```
el_salvador.2 <- multinom(contraception ~ age,
                          weights = count, data = el_salvador)
```

```
## # weights:  9 (4 variable)
## initial  value 3477.107894
## iter  10 value 3019.887931
## iter  10 value 3019.887930
## final  value 3019.887930
## converged
```

```
summary(el_salvador.2)
```

```
## Call:
## multinom(formula = contraception ~ age, data = el_salvador, weights = count)
##
## Coefficients:
##       (Intercept)         age
## Other  -0.1517456 -0.03711584
## Ster   -2.2363358  0.05342966
##
## Std. Errors:
##       (Intercept)         age
## Other   0.1900505 0.006459679
## Ster    0.1588330 0.004680629
##
## Residual Deviance: 6039.776
## AIC: 6047.776
```

(or do it as I did before with `update`).

Then we can get predictions out of this model:

```
new
```

```
## # A tibble: 7 x 1
##     age
##   <dbl>
## 1  17.5
## 2  22.5
## 3  27.5
## 4  32.5
## 5  37.5
## 6  42.5
## 7  47.5
```

```
p2 <- predict(el_salvador.2, new, type = "probs")
pred2 <- cbind(new, p2)
pred2
```

```
##    age      None      Other      Ster
## 1 17.5 0.5810810 0.26076439 0.1581547
## 2 22.5 0.5786133 0.21567707 0.2057097
## 3 27.5 0.5636953 0.17452754 0.2617772
## 4 32.5 0.5365439 0.13798403 0.3254720
## 5 37.5 0.4985081 0.10648777 0.3950041
## 6 42.5 0.4519864 0.08019677 0.4678168
## 7 47.5 0.4001020 0.05896661 0.5409314
```

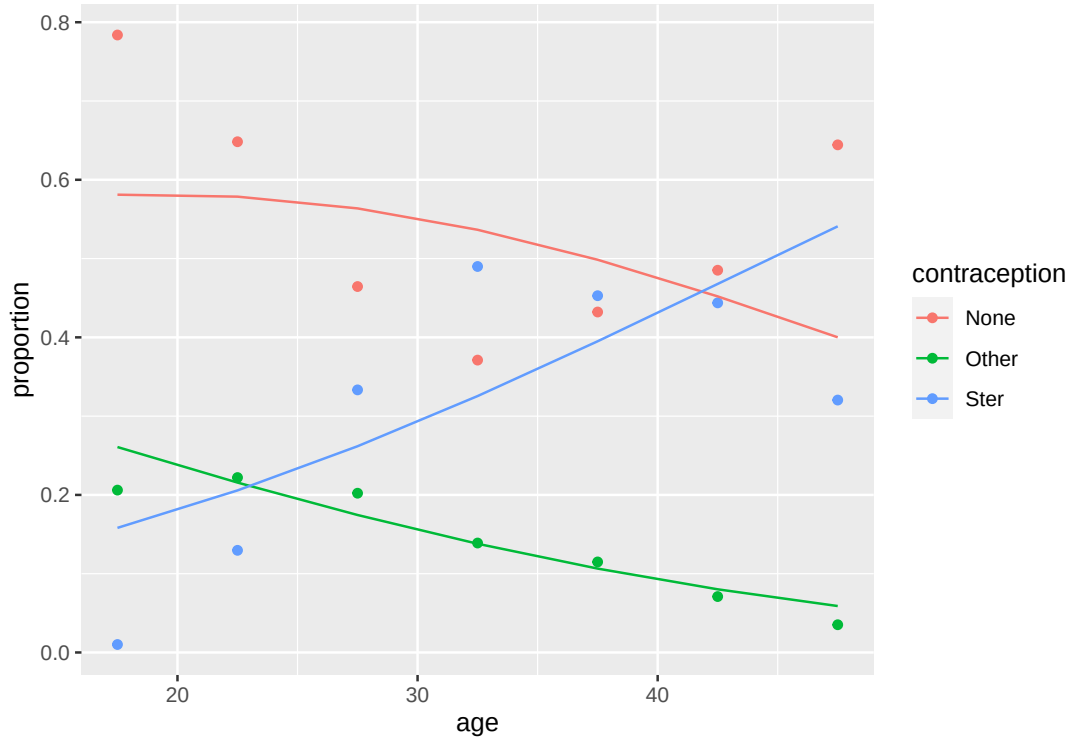
and then we can plot *these* predictions with the observed proportions in the same way you did:

```
pred2 %>%
  pivot_longer(-age, names_to = "contraception", values_to = "prob") -> pred2_long
pred2_long
```

```
## # A tibble: 21 x 3
```

```
##      age contraception  prob
##    <dbl> <chr>         <dbl>
##  1  17.5 None          0.581
##  2  17.5 Other         0.261
##  3  17.5 Ster          0.158
##  4  22.5 None          0.579
##  5  22.5 Other         0.216
##  6  22.5 Ster          0.206
##  7  27.5 None          0.564
##  8  27.5 Other         0.175
##  9  27.5 Ster          0.262
## 10  32.5 None          0.537
## # ... with 11 more rows
```

```
ggplot(observed, aes(x = age, y = proportion, colour = contraception)) +
  geom_point() +
  geom_line(data = pred2_long, aes(y=prob))
```



This is clearly a worse fit, since it needs to bend more. The green curve is mostly near the points, but the red and blue ones don't get the shapes right at all.

I was originally going to have you discover this for yourselves, and *then* add age-squared, but then you would have had to draw two graphs and do some repetitive stuff and some extra thinking, so I decided to cut it back.

One more thing, though: here's the summary of the linear model:

```
summary(el_salvador.2)
```

```
## Call:
## multinom(formula = contraception ~ age, data = el_salvador, weights = count)
##
## Coefficients:
##       (Intercept)         age
## Other  -0.1517456 -0.03711584
## Ster   -2.2363358  0.05342966
##
## Std. Errors:
##       (Intercept)         age
## Other   0.1900505 0.006459679
## Ster    0.1588330 0.004680629
##
## Residual Deviance: 6039.776
## AIC: 6047.776
```

The three response categories are in alphabetical order, with `None` as the baseline, so its intercept and slope are zero. None of the women have an age near zero, so the intercepts are not worth looking at. `Ster` has the only positive slope of 0.053, so as age goes up, the probability of `Ster` is the only one going up. `Other` has the most negative slope, so you would expect the probability of `Other` to be going down fastest, but the green and red curves seem to be going down at about the same rate. The difficulty in interpreting this comes from the fact that this model is based on log-odds rather than probabilities directly. Unfortunately, `predict` won't give us log-odds for these models, so we have to reason it out: for `Other`, the predicted probabilities are getting down near zero, so the log-odds is changing faster than for `None`, even though the probabilities themselves are changing at about the same rate.

2. The data in http://ritsokiguess.site/STAD29/six-mp.csv are survival times (in weeks) of cancer patients given an experimental drug 6-MP or a control. (Each patient was randomly allocated to one of the treatments.) Our aim is to see whether 6-MP is effective at prolonging life. The `status` column indicates whether a patient was dead or still alive at last observation.

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

Solution:

The usual:

```
my_url <- "http://ritsokiguess.site/STAD29/six-mp.csv"
leukemia <- read_csv(my_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   treatment = col_character(),
##   survival_time = col_double(),
##   status = col_character()
## )
```

```
leukemia
```

```
## # A tibble: 42 x 3
##    treatment survival_time status
```

```
##    <chr>             <dbl> <chr>
##  1 6-MP                  6 dead
##  2 6-MP                  6 dead
##  3 6-MP                  6 dead
##  4 6-MP                  6 living
##  5 6-MP                  7 dead
##  6 6-MP                  9 living
##  7 6-MP                 10 dead
##  8 6-MP                 10 living
##  9 6-MP                 11 living
## 10 6-MP                 13 dead
## # ... with 32 more rows
```

Extra: this is not how the data came to me, and *of course* I'm going to tell you the story of how the data set got to what you see above from what I received. This is what I got:

```
treatment survival_time
6-MP      6,6,6,6*,7,9*,10,10*,11*,13,16,17*,19*,20*,22,23,25*,32*,32*,34*,35*
control   1,1,2,2,3,4,4,5,5,8,8,8,8,11,11,12,12,15,17,22,23
```

The first thing I had to do with this is to get it into R. Fortunately for me, the *only* spaces in this data file are in between the treatment names and the survival times, so `read_table` will work for this:

```
my_url <- "http://ritsokiguess.site/STAD29/mp6.txt"
leukemia0 <- read_table(my_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   treatment = col_character(),
##   survival_time = col_character()
## )
```

```
leukemia0
```

```
## # A tibble: 2 x 2
##   treatment survival_time
##   <chr>     <chr>
## 1 6-MP      6,6,6,6*,7,9*,10,10*,11*,13,16,17*,19*,20*,22,23,25*,32*,32*,34*,35*
## 2 control   1,1,2,2,3,4,4,5,5,8,8,8,8,11,11,12,12,15,17,22,23
```

This looks like `separate`, but that would be a lot of work, because a lot of values need separating. There are, at least, the same number of values in each row:

```
leukemia %>% count(treatment)
```

```
## # A tibble: 2 x 2
##   treatment     n
## * <chr>     <int>
## 1 6-MP         21
## 2 control      21
```

but we might not even have been that lucky. In any case, do you want to make up 21 new column names, and then get rid of them? I thought not. So, the way to go is `separate_rows`:

```
leukemia0 %>% separate_rows(survival_time, sep=",")
```

```
## # A tibble: 42 x 2
##    treatment survival_time
##    <chr>     <chr>
##  1 6-MP      6
##  2 6-MP      6
##  3 6-MP      6
##  4 6-MP      6*
##  5 6-MP      7
##  6 6-MP      9*
##  7 6-MP      10
##  8 6-MP      10*
##  9 6-MP      11*
## 10 6-MP      13
## # ... with 32 more rows
```

If you scroll down, you'll see that there are representatives of both treatments. But we still have some work to do: some of the survival times have asterisks by them, which means they are censored observations (these patients were never observed to die). We need a separate column with an indication of whether the patient died or not, and we also need the length of time each patient was observed for, as a number.

Each survival time might have an asterisk or it might not. This makes it difficult to use `separate`, since we might need to pull off the last character, or we might not, depending on what it contains. For this kind of task, I'd rather pull out `stringr` (part of the `tidyverse`) and go hunting for things. The function to use here is `str_detect`: this answers the question "is this piece of text in that piece of text", giving an answer of yes or no. It takes two inputs: the first one is the text you're looking in, and the second input is what you are hoping to find in it.[4] This is usually paired with `mutate`, to do something with whether or not it matches. This is how you create a column containing TRUE or FALSE depending on whether the survival time contains an asterisk or not:

```
leukemia0 %>% separate_rows(survival_time, sep=",") %>%
  mutate(matches = str_detect(survival_time, fixed("*")))
```

```
## # A tibble: 42 x 3
##    treatment survival_time matches
##    <chr>     <chr>         <lgl>
##  1 6-MP      6             FALSE
##  2 6-MP      6             FALSE
##  3 6-MP      6             FALSE
##  4 6-MP      6*            TRUE
##  5 6-MP      7             FALSE
##  6 6-MP      9*            TRUE
##  7 6-MP      10            FALSE
##  8 6-MP      10*           TRUE
##  9 6-MP      11*           TRUE
## 10 6-MP      13            FALSE
## # ... with 32 more rows
```

There is an annoying technicality here: the default form of second input to `str_detect` is a "regular expression", and in one of those, * has a special meaning,[5] so by using `fixed` I said

to gimme TRUE if it matches a literal *. Normally you won't have to worry about `fixed`, or about "escaping" the special characters, which is another way around this one.

Anyway, this is not really what I wanted. I wanted a more meaningful label that said what happened to each patient. But we have the machinery to make this happen now:

```
leukemia0 %>% separate_rows(survival_time, sep=",") %>%
  mutate(status = ifelse(str_detect(survival_time, fixed("*")), "living", "dead"))
```

```
## # A tibble: 42 x 3
##    treatment survival_time status
##    <chr>     <chr>         <chr>
##  1 6-MP      6             dead
##  2 6-MP      6             dead
##  3 6-MP      6             dead
##  4 6-MP      6*            living
##  5 6-MP      7             dead
##  6 6-MP      9*            living
##  7 6-MP      10            dead
##  8 6-MP      10*           living
##  9 6-MP      11*           living
## 10 6-MP      13            dead
## # ... with 32 more rows
```

`ifelse` starts with something that is TRUE or FALSE (here, our `str_detect`). If it's true, the next thing is the value (for `status`, here), and if it's false, the last thing is, like `=IF` in a spreadsheet.[6]

The last step is to make those survival times, which still have asterisks attached to some of them and are text (look at the top of the column) into genuine numbers. This is less of a struggle than you might think, since there is `parse_number` that pulls out anything that looks like a number:

```
leukemia0 %>% separate_rows(survival_time, sep=",") %>%
  mutate(status = ifelse(str_detect(survival_time, fixed("*")), "living", "dead")) %>%
  mutate(survival_time = parse_number(survival_time))
```

```
## # A tibble: 42 x 3
##    treatment survival_time status
##    <chr>              <dbl> <chr>
##  1 6-MP                   6 dead
##  2 6-MP                   6 dead
##  3 6-MP                   6 dead
##  4 6-MP                   6 living
##  5 6-MP                   7 dead
##  6 6-MP                   9 living
##  7 6-MP                  10 dead
##  8 6-MP                  10 living
##  9 6-MP                  11 living
## 10 6-MP                  13 dead
## # ... with 32 more rows
```

and that's the data frame I saved for you.

(b) In the context of this dataset, explain briefly what a "censored observation" is, and give an example

of one.

Solution:

A censored observation is one for which the event of interest was never observed to happen. The event has to be something that you can tell when it happens, thus it could be "death", since that happens at a particular moment, but it could not be "is alive" since that is a continuous state and you can't tell when it happens. Here, the people for whom `status` is `living` are the censored ones: the ones that were never observed to die.

For an example, pull out a patient whose status is `living`, such as the fourth one in the dataframe, who was observed for 6 weeks, was still alive at that time, and we have no idea what happened to them after that.

Extra: a standard notation for a censored observation is to put a `*` next to the number, which is what the original authors did (and we had to disentangle when I organized the data for you.)

(c) Create and display a suitable response variable, called `y`, for a Cox model for these data. How do you know which observations are censored?

Solution:

This is `Surv`, from package `survival`, which you have of course installed and loaded. The smoothest way to create the response variable is to put it in the dataframe with a `mutate`. This used not to work, but now it does (so it's the way I'd recommend, but I have to remember to go back and change my lecture notes):

```
leukemia %>% mutate(y = Surv(survival_time, status=="dead")) -> leukemia
```

This looks weird when I display the dataframe here, but when you do it it will look like my `yy` below.

The other way (that I used to do because it was the only way that worked) went like this. I like to use `with` for this, since you need to refer to the data frame more than once, but you can do it with dollar signs if that is your preference:

```
yy <- with(leukemia, Surv(survival_time, status=="dead"))
yy
```

```
##  [1]  6   6   6   6+  7   9+ 10  10+ 11+ 13  16  17+ 19+ 20+ 22  23  25+ 32+ 32+
## [20] 34+ 35+  1   1   2   2   3   4   4   5   5   8   8   8   8  11  11  12  12
## [39] 15  17  22  23
```

I called this one `yy` so as not to get it confused with the other one, but if you do it this way, call it `y` so that however you got your response, it's called `y`.

For the last point, note that the censored observations (the ones still living) are marked with a `+` here. This checks; for example, the first censored observation here is the 4th one in the dataframe, which is indeed the first one for whom `status` is `living`.

Extra: it used not to work because the thing I called `yy`, though it looks like a vector (it has a way of displaying that makes it look like that), is actually *two* columns wide:

```
print.default(yy)
```

```
##       time status
##  [1,]    6      1
```

```
##  [2,]    6      1
##  [3,]    6      1
##  [4,]    6      0
##  [5,]    7      1
##  [6,]    9      0
##  [7,]   10      1
##  [8,]   10      0
##  [9,]   11      0
## [10,]   13      1
## [11,]   16      1
## [12,]   17      0
## [13,]   19      0
## [14,]   20      0
## [15,]   22      1
## [16,]   23      1
## [17,]   25      0
## [18,]   32      0
## [19,]   32      0
## [20,]   34      0
## [21,]   35      0
## [22,]    1      1
## [23,]    1      1
## [24,]    2      1
## [25,]    2      1
## [26,]    3      1
## [27,]    4      1
## [28,]    4      1
## [29,]    5      1
## [30,]    5      1
## [31,]    8      1
## [32,]    8      1
## [33,]    8      1
## [34,]    8      1
## [35,]   11      1
## [36,]   11      1
## [37,]   12      1
## [38,]   12      1
## [39,]   15      1
## [40,]   17      1
## [41,]   22      1
## [42,]   23      1
## attr(,"type")
## [1] "right"
## attr(,"class")
## [1] "Surv"
```

(`print.default` displays something without any special properties.)

(d) Fit a Cox proportional hazards model and display the output.

Solution:

This:

```
leukemia.1 <- coxph(y~treatment, data = leukemia)
summary(leukemia.1)
```

```
## Call:
## coxph(formula = y ~ treatment, data = leukemia)
##
##   n= 42, number of events= 30
##
##                     coef exp(coef) se(coef)     z Pr(>|z|)
## treatmentcontrol 1.5721    4.8169   0.4124 3.812 0.000138 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##                  exp(coef) exp(-coef) lower .95 upper .95
## treatmentcontrol     4.817     0.2076     2.147     10.81
##
## Concordance= 0.69  (se = 0.041 )
## Likelihood ratio test= 16.35  on 1 df,   p=5e-05
## Wald test            = 14.53  on 1 df,   p=1e-04
## Score (logrank) test = 17.25  on 1 df,   p=3e-05
```

Extra: when you give a modelling function a `data=`, it will first look in the workspace for the variables it needs (so that if I had had a `y` not attached to a dataframe, it would have used that), and if it doesn't find what it needs there, it will look in the dataframe given on `data=`. So it works whichever way you do it.

(e) Do the two treatments differ significantly? If so, which one is better? Explain briefly.

Solution:

The P-value for treatment, next to `treatmentcontrol` is 0.0001, very small, so the two treatments differ.

There are only two, so this is a reasonable way to compare them. If you want to be safe, run drop1 on your fitted model:

```
drop1(leukemia.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## y ~ treatment
##           Df    AIC    LRT  Pr(>Chi)
## <none>       172.02
## treatment  1 186.37 16.352 5.261e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

For this problem, this is entirely optional.

To see which treatment is the best, go back to the summary output. The other treatment 6-MP

is the baseline (for computers, numbers are before letters alphabetically). Compared to that, the control treatment has a positive coefficient, which means that the hazard of event (death) is higher for the control treatment (and thus lower for the 6-MP treatment). That is to say, death is more likely to happen sooner for the control group.

Two things to get:

- the positive coefficient for control compared to 6-MP
- what that means in terms of the likelihood of death. (You might find this backwards to what you're expecting: if the event is a bad thing, as it is here, a more positive coefficient is *worse* and a less positive one is *better*.)

(f) Plot the estimated survival curves for each treatment.

Solution:

Steps:

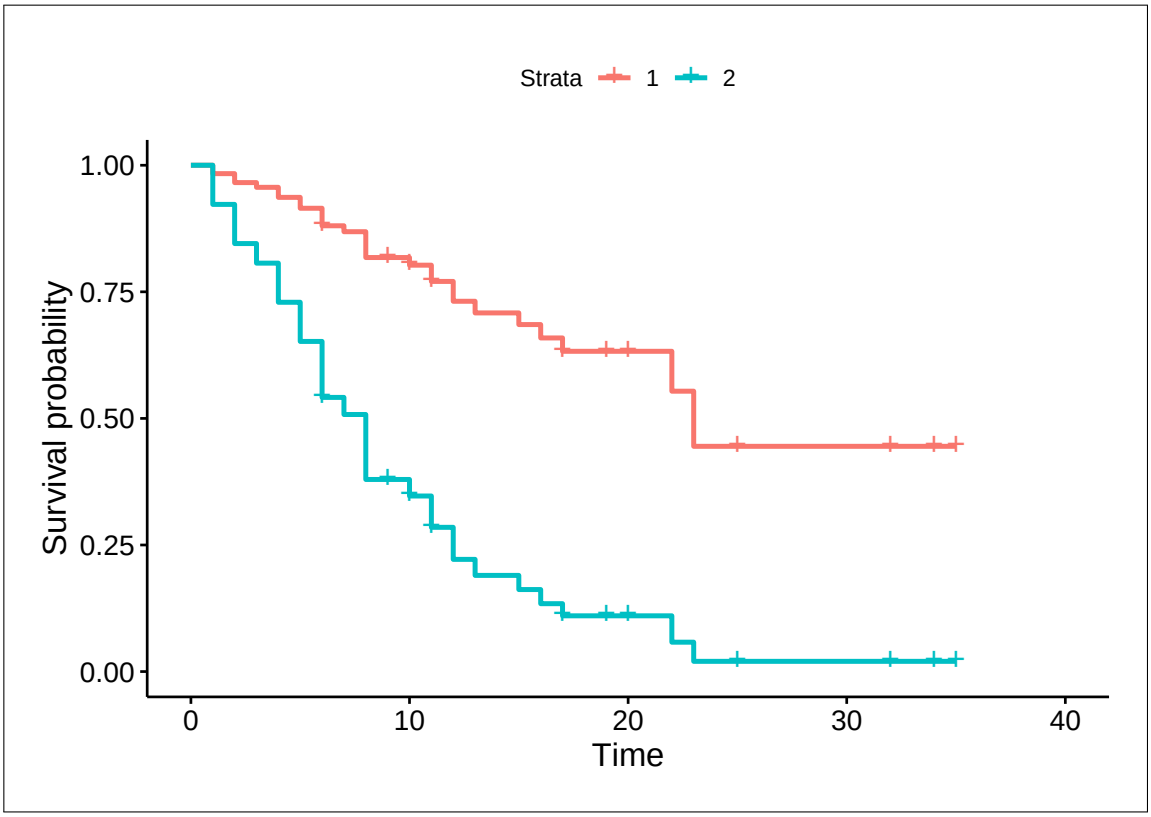
- make a dataframe of new values to predict for (the two treatments, since there is nothing else in the model)
- run `survfit`, using as input the fitted model, the new data, *and* the original data. If you name them, they can be in any order.
- run `ggsurvplot`, with or without the confidence intervals. (I like without, since I find it cluttered with, but this one especially is fine either way.)

Make the new data frame however you like. I counted the treatments, which would guarantee getting both of them. It's also fine to leave the counts in `new`, since they will just be ignored. Or, build it yourself, typing in the two treatment names. If it works, it's good.

```
leukemia %>% count(treatment) %>%
  select(treatment) -> new
new
```

```
## # A tibble: 2 x 1
##   treatment
##   <chr>
## 1 6-MP
## 2 control
```

```
s <- survfit(leukemia.1, newdata = new, data = leukemia)
ggsurvplot(s, conf.int = FALSE)
```



(g) Which treatment is better? Explain briefly. (This should be consistent with what you discovered before about the better treatment, so the credit here is for your explanation.)

Solution:

The event, death, is bad, so being more likely to survive longer is better. Hence, the higher curve is better. For me, this is stratum 1. Which treatment is that? Well, look at your `new` values to predict for:

```
new
```

```
## # A tibble: 2 x 1
##   treatment
##   <chr>
## 1 6-MP
## 2 control
```

For me, stratum 1 is 6-MP (row 1; if your `new` had control first, then your stratum *2* would be better.) So 6-MP is better, as I found before; the new drug really is better than placebo.

Two things:

- explain why up-and-right is better here (death is bad)
- explain which actual treatment goes with your better stratum.

Notes

1. Did you know that you can rename things in *select*? You do now. New name equals old name, inside the *select*. There is also a *rename* that works the same way.

2. I was worried that you would need it, in which case I would have needed to clue you in to that, but I tried it and it works without as well, so you can do it either way. If you are tempted to ask "do I need *factor* here?", the answer is "try it and see". On *glm*, you do need it if the response is not a genuine factor or 0–1, but here you don't.

3. Even though "step" *uses* "drop1". I have no idea.

4. Thus, the second input is shorter than the first one.

5. One or more of the preceding character, to be precise.

6. If you use Excel, you might be used to IF inside IF for more than two choices, but don't use that illegibility in R, because we have *case_when*, which is much clearer.