Assignment 7

Instructions (the same as for Assignment 1): Make an R Notebook and in it answer the two questions below (one Notebook for both questions). 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. 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.

You have 3 hours to complete this assignment after you start it.

My solutions to this assignment, with extra discussion, will be available after everyone has handed in their assignment.

1. How does the consumption of electricity depend on temperature? To find out, a short-term study was carried out by a utility company based in a certain area. For a period of two years, the average monthly temperature was recorded (in degrees Fahrenheit), the mean daily demand for electricity per household (in kilowatt hours), and the cost per kilowatt hour of electricity for that year (8 cents for the first year and 10 cents for the second, which it will be easiest to treat as categorical).

   The data were laid out in an odd way, as shown in http://ritsokiguess.site/STAC32/utils.txt, in aligned columns: the twelve months of temperature were laid out on *two* lines for the first year, then the twelve months of consumption for the first year on the next two lines, and then four more lines for the second year laid out the same way. Thus the temperature of 31 in the first line goes with the consumption of 55 in the *third* line, and the last measurements for that year are the 78 at the end of the second line (temperature) and 73 at the end of the fourth line (consumption). Lines 5 through 8 of the data file are the same thing for the second year (when electricity was more expensive).

   The data seem to have been laid out in order of temperature, rather than in order of months, which I would have thought would make more sense. But this is what we have.

   (a) Read in and display the data file, bearing in mind that it has *no column names*.

Solution:

That means `col_names = FALSE` when reading in. I gave this a "disposable" name, saving the good name `utils` for the tidy version:

```r
my_url <- "http://ritsokiguess.site/STAC32/utils.txt"
utils0 <- read_table(my_url, col_names = FALSE)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_character(),
##   X2 = col_character(),
##   X3 = col_double(),
##   X4 = col_double(),
##   X5 = col_double(),
##   X6 = col_double(),
##   X7 = col_double(),
##   X8 = col_double()
## )
```

```
utils0
```

```
## # A tibble: 8 x 8
##   X1      X2             X3    X4    X5    X6    X7    X8
##   <chr>   <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 8cents  temperature    31    34    39    42    47    56
## 2 8cents  temperature    62    66    68    71    75    78
## 3 8cents  consumption    55    49    46    47    40    43
## 4 8cents  consumption    41    46    44    51    62    73
## 5 10cents temperature    32    36    39    42    48    56
## 6 10cents temperature    62    66    68    72    75    79
## 7 10cents consumption    50    44    42    42    36    40
## 8 10cents consumption    39    44    40    44    50    55
```

The columns have acquired names `X1` through `X8`. It doesn't really matter what these names *are*, but as we will see shortly, it matters that they *have* names.

(b) Arrange these data tidily, so that there is a column of price (per kilowatt hour), a column of temperatures, and a column of temperatures. Describe your process, including why you got list-columns (if you did) and what you did about them (if necessary).

Solution:

This question is asking about your process as well as your answer, so I think it's best to build a pipeline one step at a time (which corresponds in any case to how you would figure out what to do). The first step seems to be to make longer, for example getting all those numbers in one column. I'm not quite sure what to call the new columns, so I'll make up some names and figure things out later:

```
utils0 %>% pivot_longer(X3:X8, names_to = "col", values_to = "value")
```

```
## # A tibble: 48 x 4
##    X1     X2          col   value
##    <chr>  <chr>       <chr> <dbl>
##  1 8cents temperature X3       31
##  2 8cents temperature X4       34
##  3 8cents temperature X5       39
##  4 8cents temperature X6       42
##  5 8cents temperature X7       47
##  6 8cents temperature X8       56
##  7 8cents temperature X3       62
##  8 8cents temperature X4       66
##  9 8cents temperature X5       68
## 10 8cents temperature X6       71
## # ... with 38 more rows
```

If you scroll down, `X2` has consumptions as well as temperatures, so we need to get that straightened out.

This, so far, is actually a lot like the weather one in lecture (where we had a max and a min temperature), and the solution is the same: follow up with a `pivot_wider` to get the temperatures and consumptions in their own columns:

```
utils0 %>% pivot_longer(X3:X8, names_to = "col", values_to = "value") %>%
  pivot_wider(names_from = X2, values_from = value)
```

```
## Warning: Values are not uniquely identified; output will contain list-cols.
## * Use `values_fn = list` to suppress this warning.
## * Use `values_fn = length` to identify where the duplicates arise
## * Use `values_fn = {summary_fun}` to summarise duplicates
```

```
## # A tibble: 12 x 4
##    X1      col   temperature consumption
##    <chr>   <chr> <list>      <list>
##  1 8cents  X3    <dbl [2]>   <dbl [2]>
##  2 8cents  X4    <dbl [2]>   <dbl [2]>
##  3 8cents  X5    <dbl [2]>   <dbl [2]>
##  4 8cents  X6    <dbl [2]>   <dbl [2]>
##  5 8cents  X7    <dbl [2]>   <dbl [2]>
##  6 8cents  X8    <dbl [2]>   <dbl [2]>
##  7 10cents X3    <dbl [2]>   <dbl [2]>
##  8 10cents X4    <dbl [2]>   <dbl [2]>
##  9 10cents X5    <dbl [2]>   <dbl [2]>
## 10 10cents X6    <dbl [2]>   <dbl [2]>
## 11 10cents X7    <dbl [2]>   <dbl [2]>
## 12 10cents X8    <dbl [2]>   <dbl [2]>
```

Except that it didn't appear to work. Although it actually did. These are list-columns. I actually recorded lecture 14a to help you with this. (See also the discussion in Extra 3 of the last part of the writers question on Assignment 5, and the word "list" at the top of `temperature` and `consumption`), and each cell holds *two* numbers instead of the one you were probably expecting.

Why did that happen? The warning above the output is a clue. Something is going to be "not uniquely identified". Think about how `pivot_wider` works. It has to decide which *row* and *column* of the wider dataframe to put each value in. The column comes from the `names_from`: temperature or consumption. So that's not a problem. The row comes from the combination of the other columns not named in the `pivot_wider`: that means the ones called `X1` and `col`. (Another way to see that is the columns in the result from the `pivot_wider` that *do not* have values in them: not `temperature` or `consumption`, the other two.)

If you look back at the things in `col`, they go from `X3` to `X8`, so there are six of them. There are two values in `X1`, so there are $2 \times 6 = 12$ combinations of the two, and so 12 rows in the wider dataframe. This has two columns, and thus $12 \times 2 = 24$ cells altogether. But there were 48 values in the longer dataframe (go back and look: it has 48 rows), so there isn't enough room for all of them here.

If you go back and look at the longer dataframe, you'll see, for example, that there are two `temperature` values that go with an `X1` of 8 cents and a col of `X3`, so that they will both have to be jammed into one cell of the wider dataframe.

The resolution of the list-columns here is the same as in the one about the writers: `unnest` them, and then you can ignore the warning:

```
utils0 %>% pivot_longer(X3:X8, names_to = "col", values_to = "value") %>%
  pivot_wider(names_from = X2, values_from = value) %>%
  unnest(c(temperature, consumption)) -> utils
```

```
## Warning: Values are not uniquely identified; output will contain list-cols.
## * Use `values_fn = list` to suppress this warning.
## * Use `values_fn = length` to identify where the duplicates arise
## * Use `values_fn = {summary_fun}` to summarise duplicates
```

```
utils
```

```
## # A tibble: 24 x 4
##    X1     col   temperature consumption
##    <chr>  <chr>       <dbl>       <dbl>
##  1 8cents X3             31          55
##  2 8cents X3             62          41
##  3 8cents X4             34          49
##  4 8cents X4             66          46
##  5 8cents X5             39          46
##  6 8cents X5             68          44
##  7 8cents X6             42          47
##  8 8cents X6             71          51
##  9 8cents X7             47          40
## 10 8cents X7             75          62
## # ... with 14 more rows
```

There were 24 months of data, and a temperature and consumption for each, so this is now tidy and I can give it a proper name.

Extra: if you got to here and got scared:

```
utils0 %>% pivot_longer(X3:X8, names_to = "col", values_to = "value") %>%
  pivot_wider(names_from = X2, values_from = value)
```

```
## Warning: Values are not uniquely identified; output will contain list-cols.
## * Use `values_fn = list` to suppress this warning.
## * Use `values_fn = length` to identify where the duplicates arise
## * Use `values_fn = {summary_fun}` to summarise duplicates
```

```
## # A tibble: 12 x 4
##    X1      col   temperature consumption
##    <chr>   <chr> <list>      <list>
##  1 8cents  X3    <dbl [2]>   <dbl [2]>
##  2 8cents  X4    <dbl [2]>   <dbl [2]>
##  3 8cents  X5    <dbl [2]>   <dbl [2]>
##  4 8cents  X6    <dbl [2]>   <dbl [2]>
##  5 8cents  X7    <dbl [2]>   <dbl [2]>
##  6 8cents  X8    <dbl [2]>   <dbl [2]>
##  7 10cents X3    <dbl [2]>   <dbl [2]>
##  8 10cents X4    <dbl [2]>   <dbl [2]>
##  9 10cents X5    <dbl [2]>   <dbl [2]>
## 10 10cents X6    <dbl [2]>   <dbl [2]>
## 11 10cents X7    <dbl [2]>   <dbl [2]>
## 12 10cents X8    <dbl [2]>   <dbl [2]>
```

which is an entirely reasonable reaction, you might have asked yourself how you could have prevented this from happening. The problem, as discussed earlier, is with the rows, and that the `X1-col` combinations repeat. Let's go back to "longer":

```
utils0 %>% pivot_longer(X3:X8, names_to = "col", values_to = "value")
```

```
## # A tibble: 48 x 4
##    X1     X2          col   value
##    <chr>  <chr>       <chr> <dbl>
##  1 8cents temperature X3       31
##  2 8cents temperature X4       34
##  3 8cents temperature X5       39
##  4 8cents temperature X6       42
##  5 8cents temperature X7       47
##  6 8cents temperature X8       56
##  7 8cents temperature X3       62
##  8 8cents temperature X4       66
##  9 8cents temperature X5       68
## 10 8cents temperature X6       71
## # ... with 38 more rows
```

Rows 1 and 7, 2 and 8, etc, are "replicates" in that they have the same `X1` and `col` values but different temperatures. This is because they come from the same column in the original layout of the data (the 31 and the 62 are underneath each other). This means that the first six rows are "replicate 1" and the next six are "replicate 2". Scrolling down, we then get to 8 cents and consumption, and we need to do the same again. So if we make a column that has 1s and 2s in the right places (six 1s, six 2s, repeat), we should then have unique rows for the `pivot_wider`.

```
utils0 %>% pivot_longer(X3:X8, names_to = "col", values_to = "value") %>%
  mutate(replicate = rep(1:2, each = 6, length.out = 48))
```

```
## # A tibble: 48 x 5
##    X1     X2          col   value replicate
##    <chr>  <chr>       <chr> <dbl>     <int>
##  1 8cents temperature X3       31         1
##  2 8cents temperature X4       34         1
##  3 8cents temperature X5       39         1
##  4 8cents temperature X6       42         1
##  5 8cents temperature X7       47         1
##  6 8cents temperature X8       56         1
##  7 8cents temperature X3       62         2
##  8 8cents temperature X4       66         2
##  9 8cents temperature X5       68         2
## 10 8cents temperature X6       71         2
## # ... with 38 more rows
```

`rep` does repeats like this: something to repeat (the numbers 1 through 2), how many times to repeat each one (six times), and how long the final thing has to be (48, since there were 48 rows in the longer dataframe).

Then, this time, if we do the `pivot_wider`, it should give us something tidy:

```
utils0 %>% pivot_longer(X3:X8, names_to = "col", values_to = "value") %>%
  mutate(replicate = rep(1:2, each = 6, length.out = 48)) %>%
  pivot_wider(names_from = X2, values_from = value)
```

```
## # A tibble: 24 x 5
##    X1     col   replicate temperature consumption
##    <chr>  <chr>     <int>       <dbl>       <dbl>
##  1 8cents X3            1          31          55
##  2 8cents X4            1          34          49
##  3 8cents X5            1          39          46
##  4 8cents X6            1          42          47
##  5 8cents X7            1          47          40
##  6 8cents X8            1          56          43
##  7 8cents X3            2          62          41
##  8 8cents X4            2          66          46
##  9 8cents X5            2          68          44
## 10 8cents X6            2          71          51
## # ... with 14 more rows
```

and so it does, with 24 rows for the 24 months.

Another, perhaps easier, way to think about this (you might find it easier, anyway) is to go back to the original dataframe and make the `replicate` there:

```
utils0
```

```
## # A tibble: 8 x 8
##   X1      X2             X3    X4    X5    X6    X7    X8
##   <chr>   <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 8cents  temperature    31    34    39    42    47    56
## 2 8cents  temperature    62    66    68    71    75    78
## 3 8cents  consumption    55    49    46    47    40    43
## 4 8cents  consumption    41    46    44    51    62    73
## 5 10cents temperature    32    36    39    42    48    56
## 6 10cents temperature    62    66    68    72    75    79
## 7 10cents consumption    50    44    42    42    36    40
## 8 10cents consumption    39    44    40    44    50    55
```

The first two rows are replicates (both 8 cents and temperature), then the third and fourth, and so on. So setting a `replicate` column as 1, 2, 1, 2 etc should do it, and this is short enough to type directly. Do this *first*, then the `pivot_longer`, then the `pivot_wider` as we did before, and we should end up with something tidy:

```
utils0 %>% mutate(replicate = c(1,2,1,2,1,2,1,2)) %>%
  pivot_longer(X3:X8, names_to = "col", values_to = "value") %>%
  pivot_wider(names_from = X2, values_from = value) %>%
  unnest(c(temperature, consumption))
```

```
## # A tibble: 24 x 5
##    X1     replicate col   temperature consumption
##    <chr>      <dbl> <chr>       <dbl>       <dbl>
##  1 8cents         1 X3             31          55
##  2 8cents         1 X4             34          49
##   3 8cents          1 X5              39           46
##   4 8cents          1 X6              42           47
##   5 8cents          1 X7              47           40
##   6 8cents          1 X8              56           43
##   7 8cents          2 X3              62           41
##   8 8cents          2 X4              66           46
##   9 8cents          2 X5              68           44
##  10 8cents          2 X6              71           51
## # ... with 14 more rows
```

If you check this, you'll see that `replicate` gets turned into the same thing in the longer dataframe that we had earlier, so you can do it either way.

The moral of the story is that when you are planning to do a pivot-wider, you ought to devote some attention to which *rows* things are going into. Sometimes you can get away with just doing it and it works, but thinking about rows is how to diagnose it when it doesn't. (The ideas below also appear in Lecture 14a.) Here's another mini-example where the observations are matched pairs but they come to us long, like two-sample data:

```
d <- tribble(
  ~obs, ~y, ~time,
  1, 10, "pre",
  2, 13, "post",
  3, 12, "pre",
  4, 14, "post",
  5, 13, "pre",
  6, 15, "post"
)
d %>% pivot_wider(names_from = time, values_from = y)
```

```
## # A tibble: 6 x 3
##     obs   pre  post
##   <dbl> <dbl> <dbl>
## 1     1    10    NA
## 2     2    NA    13
## 3     3    12    NA
## 4     4    NA    14
## 5     5    13    NA
## 6     6    NA    15
```

Oh. The columns are all right, but the rows certainly are not.

The problem is that the only thing left after `y` and `time` have been used in the `pivot_wider` is the column `obs`, and there are six values there, so there are six rows. This is, in a way, the opposite of the problem we had before; now, there is *not enough* data to fill the twelve cells of the wider dataframe. For example, there is no `pre` measurement in the row where `obs` is 2, so this cell of the wider dataframe is empty: it has a missing value in it.

The problem is that the `obs` column numbered the six observations 1 through 6, but really they are three groups of two observations on three people, so instead of `obs` we need a column called `person` that shows which observations are the matched pairs, like this:

```
d <- tribble(
  ~person, ~y, ~time,
  1, 10, "pre",
  1, 13, "post",
  2, 12, "pre",
  2, 14, "post",
  3, 13, "pre",
  3, 15, "post"
)
```

Now there are going to be three rows with a pre and a post in each:

```
d %>% pivot_wider(names_from = time, values_from = y)
```

```
## # A tibble: 3 x 3
##   person   pre  post
##    <dbl> <dbl> <dbl>
## 1      1    10    13
## 2      2    12    14
## 3      3    13    15
```

`pivot_wider` requires more thinking than `pivot_longer`, and when it does something mysterious, that's when you need to have some understanding of how it works, so that you can fix things up.

(c) Make a suitable graph of temperature, consumption and price in your tidy dataframe. Add smooth trends if appropriate. If you were unable to get the data tidy, use my tidy version in http://ritsokiguess.site/STAC32/utils_tidy.csv.

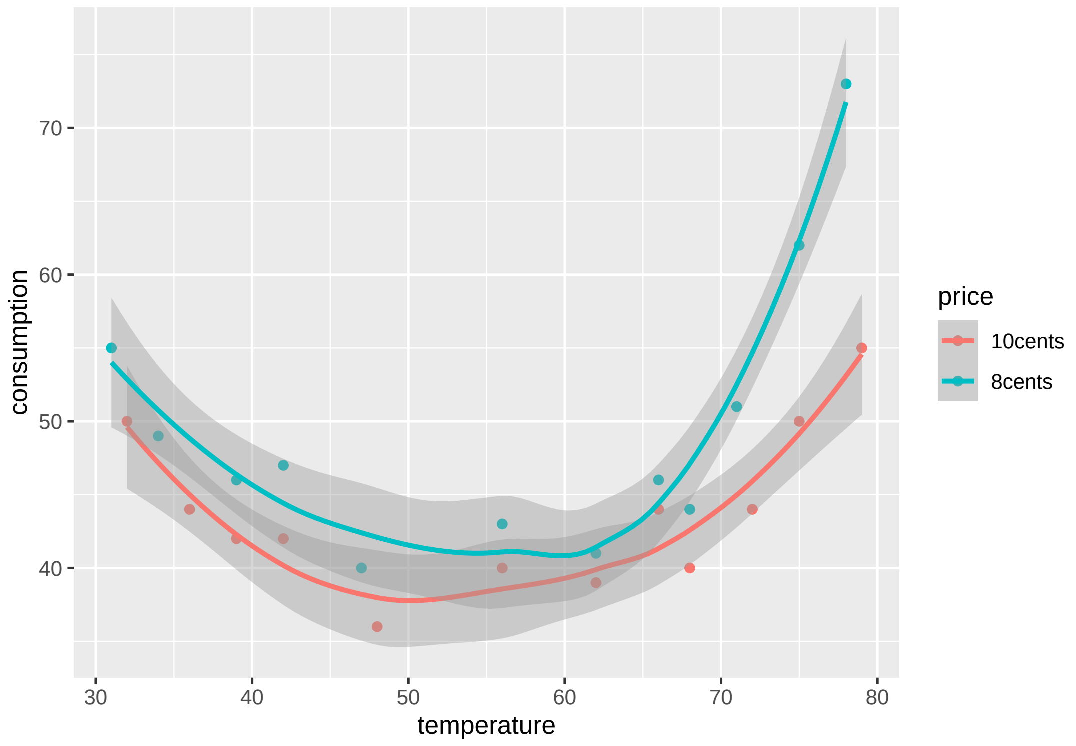
Solution:

I said earlier to treat price (rather badly labelled as `X1`) as categorical, so we have two quantitative variables and one categorical. This suggests a scatterplot with the two prices distinguished by colours. (We don't have a mechanism for making three-dimensional plots, and in any case if you have a quantitative variable with not that many distinct different values, you can often treat that as categorical, such as price here.)

Before we make a graph, though, we should rename `X1`. The way you might think of is to create a new column with the same values as `X1`, but a new name.[1] Like this. Consumption is the outcome, so it goes on the y-axis:

```
utils %>%
  mutate(price = X1) %>%
  ggplot(aes(x = temperature, y = consumption, colour = price)) +
    geom_point() + geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



I said smooth trends rather than lines, because you don't know until you draw the graph whether the trends *are* lines. If they're not, there's not much point in drawing lines through them. These ones are rather clearly curves, which we take up in the next part.

If you fail to rename `X1`, that's what will appear on the legend, and the first thing your reader would ask is "what is `X1`?" When writing, you need to think of your reader, since they are (in the real world) paying you for your work.

Extra: there is an official `rename` also. I haven't used that in class, so if you discover this, make sure to say where you found out about it from:

```
utils %>%
  rename(price = X1)
```

```
## # A tibble: 24 x 4
##    price  col   temperature consumption
##    <chr>  <chr>       <dbl>       <dbl>
##  1 8cents X3             31          55
##  2 8cents X3             62          41
##  3 8cents X4             34          49
##  4 8cents X4             66          46
##  5 8cents X5             39          46
##  6 8cents X5             68          44
##  7 8cents X6             42          47
##  8 8cents X6             71          51
##  9 8cents X7             47          40
## 10 8cents X7             75          62
## # ... with 14 more rows
```

The syntax is "new name equals old one". I used to think it was something like "take the column called `X1` and rename it to `price`", but as you see, that's exactly backwards. The English-language version is "create a new column called `price` from the column previously called `X1`".

(d) What patterns or trends do you see on your graph? Do they make practical sense? There are two things I would like you to comment on.

Solution:

The two things are:

- the relationships are both curves, going down and then up again.
- the blue curve is above the red one.

If the temperature is low (30 degrees F is just below freezing), people will need to heat their houses, and the electricity consumption to do this is reflected in the curves being higher at the left. (Not all people have electric heating, but at least some people do, and electric heating uses a lot of electricity.) When the temperature is high, people will turn on the air-conditioning (which is usually electric), and that explains the sharp increase in consumption at high temperatures. In between is the zone where the house stays a good temperature without heating or cooling.

So why is the blue curve above the red one? This is saying that when electricity is cheaper, people will use more of it. (This seems to be particularly true when the temperature is high; people might "crank" the air-conditioning if it doesn't cost too much to run.) Conversely, if electricity is more expensive, people might be more thoughtful about what temperature to turn on the heating or AC.

2. Dolphins and other large marine mammals are at the top of the marine food chain, and so if there is any heavy metal pollution in the sea, it will find its way into the dolphins. The study we look at is of the concentration of mercury. This is expected to be different in males and females because the mercury in a female is transferred to her offspring during gestation and nursing. In this study, there were 28 males and 17 females of various ages. There are three columns in the data file:

   - `mercury`, the concentration in the liver, in micrograms per gram
   - `age` in years
   - `sex` of the dolphin, male or female.

   The data are in http://ritsokiguess.site/STAC32/dolphins.csv as a CSV file. This question appears to have a lot of parts, but most of them ought not to take you too long.

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

Solution:

Exactly the usual:

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

```
## Parsed with column specification:
## cols(
##   mercury = col_double(),
##   age = col_double(),
##   sex = col_character()
## )
```

```
dolphins
```

```
## # A tibble: 45 x 3
##    mercury   age sex
##      <dbl> <dbl> <chr>
##  1    1.7   0.21 male
##  2    1.72  0.33 male
##  3    8.8   2    male
##  4    5.9   2.2  male
##  5  101     8.5  male
##  6   85.4  11.5  male
##  7  118    11.5  male
##  8  183    13.5  male
##  9  168    16.5  male
## 10  218    16.5  male
## # ... with 35 more rows
```

There are indeed 45 dolphins, with the males listed first and the females listed at the end (as you can check).

(b) Display only the two columns `mercury` and `sex`.

Solution:

Two ways to do it: either list both columns:

```
dolphins %>% select(mercury, sex)
```

```
## # A tibble: 45 x 2
##    mercury sex
##      <dbl> <chr>
##  1    1.7  male
##  2    1.72 male
##  3    8.8  male
##  4    5.9  male
##  5  101    male
##  6   85.4  male
##  7  118    male
##  8  183    male
##  9  168    male
## 10  218    male
## # ... with 35 more rows
```

or say "everything but age":

```
dolphins %>% select(-age)
```

```
## # A tibble: 45 x 2
##    mercury sex
##      <dbl> <chr>
##  1    1.7  male
##  2    1.72 male
##  3    8.8  male
##  4    5.9  male
##  5  101    male
##  6   85.4  male
##  7  118    male
##  8  183    male
##  9  168    male
## 10  218    male
## # ... with 35 more rows
```

The second one is easier coding, so is better.

Note that `mercury:sex` won't do it, because that will get you `age` as well ("`mercury` through `sex`").

(c) Display all the columns whose names have exactly three characters, without naming any columns.

Solution:

Evidently this is `age` and `sex`. We are selecting columns on the basis of their names, so we will need a select-helper. The one that works is `matches` with a regular expression. This is like the one in the lecture notes where we selected height and weight by selecting columns whose names had two letters and the second one was "t". In this case, the three letters can be anything, so it's three dots between the start of text and end of text:

```
dolphins %>%
  select(matches("^...$"))
```

```
## # A tibble: 45 x 2
##      age sex
##    <dbl> <chr>
##  1  0.21 male
##  2  0.33 male
##  3  2    male
##  4  2.2  male
##  5  8.5  male
##  6 11.5  male
##  7 11.5  male
##  8 13.5  male
##  9 16.5  male
## 10 16.5  male
## # ... with 35 more rows
```

(d) Display only the mercury levels for the females.

Solution:

This is selecting rows *and* columns, so you'll have a `select` and a `filter`. If you do the `select` first, you'll only have the mercury levels left, and you won't know which dolphins are female. So do the `filter` first:

```r
dolphins %>% filter(sex == "female") %>%
  select(mercury)
```

```
## # A tibble: 17 x 1
##    mercury
##      <dbl>
##  1    2.5
##  2    9.35
##  3    4.01
##  4   29.8
##  5   45.3
##  6  101
##  7  135
##  8  142
##  9  180
## 10  174
## 11  247
## 12  223
## 13  167
## 14  157
## 15  177
## 16  475
## 17  342
```

(e) What is the mean mercury concentration for all the dolphins whose age is less than 15?

Solution:

Grab only the dolphins with age less than 15 first, and then work out their mean `mercury` with `summarize`:

```r
dolphins %>% filter(age<15) %>%
  summarize(m = mean(mercury))
```

```
## # A tibble: 1 x 1
##       m
##   <dbl>
## 1  55.5
```

55.5 (micrograms per gram), as you should say.

Another way is to define a new column that indicates whether the age is less than 15 or not, and do group-by and summarize:

```r
dolphins %>%
  mutate(young = ifelse(age<15, "yes", "no")) %>%
  group_by(young) %>%
  summarize(m = mean(mercury))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   young     m
##   <chr> <dbl>
## 1 no    280.
## 2 yes    55.5
```

Here, you *definitely* need to extract the right number out of the output. You can also define groups by logical condition directly, which saves you a step:

```
dolphins %>%
  group_by(age<15) %>%
  summarize(m = mean(mercury))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   `age < 15`     m
##   <lgl>      <dbl>
## 1 FALSE      280.
## 2 TRUE        55.5
```

Doing it this way, you are probably getting some suspicions about the relationship between age and mercury concentration, but that's coming up.

(f) What is the mean mercury concentration for all the dolphins whose age is greater than 25?

Solution:

The same three choices of method:

```
dolphins %>% filter(age>25) %>%
  summarize(m = mean(mercury))
```

```
## # A tibble: 1 x 1
##       m
##   <dbl>
## 1  309.
```

309.2 (micrograms per gram), as you should say.

Or define a new column:

```
dolphins %>%
  mutate(young = ifelse(age>25, "yes", "no")) %>%
  group_by(young) %>%
  summarize(m = mean(mercury))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   young     m
##   <chr> <dbl>
## 1 no     142.
## 2 yes    309.
```

Or:

```
dolphins %>%
  group_by(age>25) %>%
  summarize(m = mean(mercury))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 2 x 2
##   `age > 25`     m
##   <lgl>      <dbl>
## 1 FALSE       142.
## 2 TRUE        309.
```

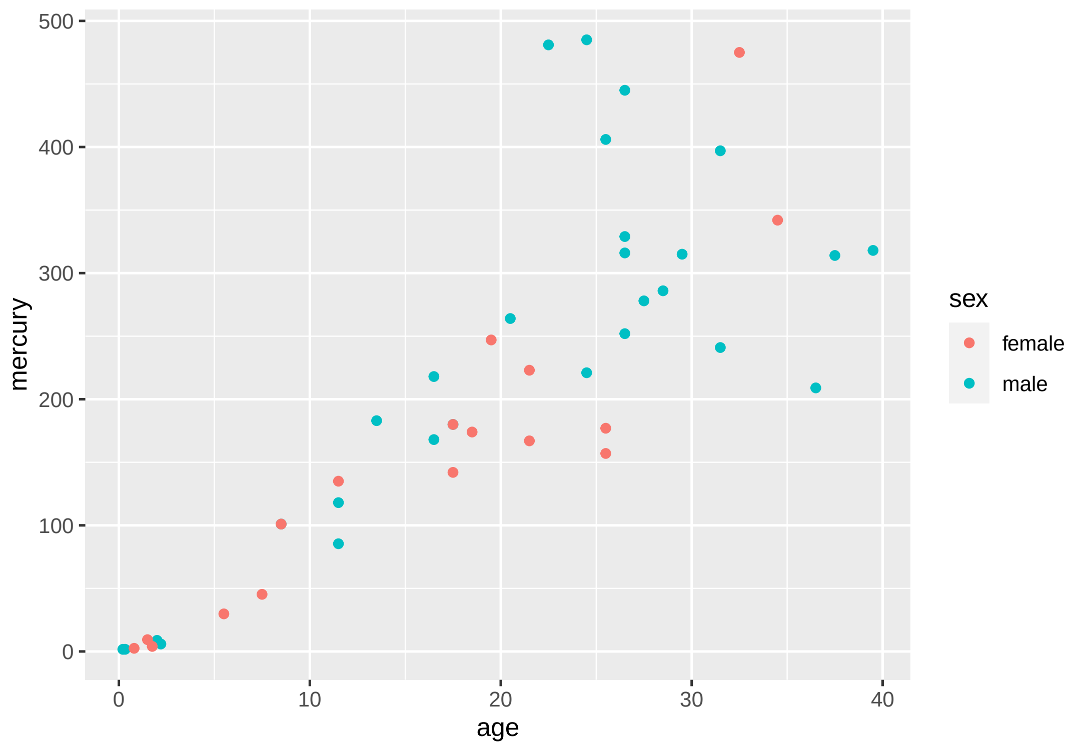
In any of these ways, 309.2 micrograms per gram.

(g) Make a suitable graph of these data (all three columns).

Solution:

This one ought to be a gimme, but I am using the result in the next part. With two quantitative variables and one categorical, a scatterplot with the sexes indicated by colour:

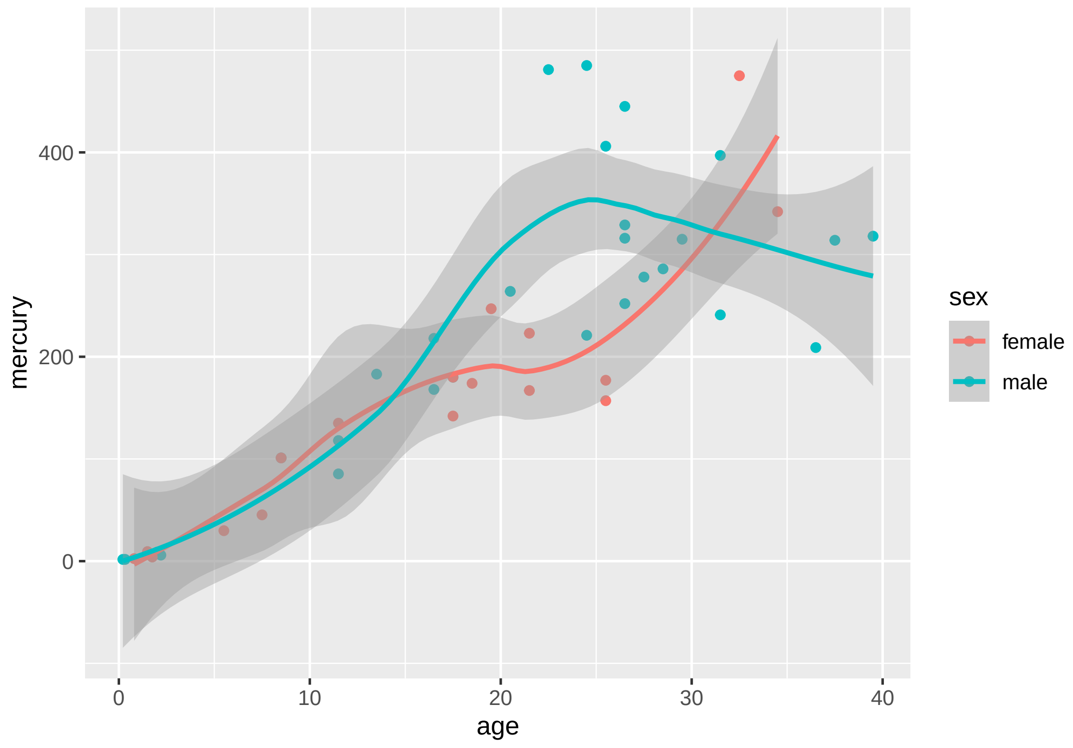
```
ggplot(dolphins, aes(x = age, y = mercury, colour = sex)) + geom_point()
```



I think this is clearer without regression lines; the upward trend is clear enough, and it is not clear that the trends are linear (look at the males of the largest ages). Add a smooth trend if you like:

```
ggplot(dolphins, aes(x = age, y = mercury, colour = sex)) + geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Mercury concentration increases with age for the females, and also does with age for the males up to about age 25, after which it levels off or decreases.

(h) Explain briefly how your graph and your calculations of mean mercury concentration are telling a similar story.

Solution:

I wanted to get to a point where we were saying something interesting about the data, rather than just playing with `select` and friends.

The graph is showing an increasing trend of mercury concentration with age (at least, for all but the oldest males). The mean calculations are showing that the younger dolphins have a small mercury concentration on average, and the older dolphins have a much larger mean. These are both saying that mercury concentration is increasing with age, so they are consistent.

Extra:

I tried to make this fairly obvious for you by choice of age groups to compare, and this pattern holds for both males and females, so that the mean calculations didn't need to depend on `sex`. But there is no great difficulty in making it work by sex as well – insert a `group_by` before the `summarize`:

```
dolphins %>% filter(age<15) %>%
  group_by(sex) %>%
  summarize(m = mean(mercury))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## # A tibble: 2 x 2
##   sex        m
##   <chr>  <dbl>
## 1 female  46.7
## 2 male    63.2
```

and

```
dolphins %>% filter(age>25) %>%
  group_by(sex) %>%
  summarize(m = mean(mercury))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   sex        m
##   <chr>  <dbl>
## 1 female  288.
## 2 male    316.
```

The male means are slightly bigger in the two cases, though the difference between the sexes is small compared to the age effect. (I ran a regression predicting mercury concentration from age and sex, and the sex effect wasn't even significant.)

A second extra: what if you wanted to divide age up into age groups for comparison of mercury concentration? This isn't always a good idea (for reasons I get into below), but if you want to do it, `cut` is your friend:

```
dolphins %>% mutate(age_group = cut(age, breaks=c(0,15,25,60))) -> dolphinsx
dolphinsx
```

```
## # A tibble: 45 x 4
##    mercury   age sex   age_group
##      <dbl> <dbl> <chr> <fct>
##  1    1.7   0.21 male  (0,15]
##  2    1.72  0.33 male  (0,15]
##  3    8.8   2    male  (0,15]
##  4    5.9   2.2  male  (0,15]
##  5  101     8.5  male  (0,15]
##  6   85.4  11.5  male  (0,15]
##  7  118    11.5  male  (0,15]
##  8  183    13.5  male  (0,15]
##  9  168    16.5  male  (15,25]
## 10  218    16.5  male  (15,25]
## # ... with 35 more rows
```

Note that `cut` needs input `breaks` whose first value is lower than all the ages and whose last value is higher than all the ages. The age group intervals are what mathematicians would call "half-open": the upper limit is included in the interval and the lower limit is not. The age of 15 is thus in the 0-15 interval and not in 15-25.

Now we can work out mean mercury concentration for each age group:

```
dolphinsx %>% group_by(age_group) %>%
  summarize(m = mean(mercury))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 2
##   age_group     m
##   <fct>     <dbl>
## 1 (0,15]     55.5
## 2 (15,25]   242.
## 3 (25,60]   309.
```

or even do it by sex as well:

```
dolphinsx %>% group_by(age_group, sex) %>%
  summarize(m = mean(mercury))
```

```
## `summarise()` regrouping output by 'age_group' (override with `.groups` argument)
```

```
## # A tibble: 6 x 3
## # Groups:   age_group [3]
##   age_group sex        m
##   <fct>     <chr>  <dbl>
## 1 (0,15]    female  46.7
## 2 (0,15]    male    63.2
## 3 (15,25]   female 189.
## 4 (15,25]   male   288.
## 5 (25,60]   female 288.
## 6 (25,60]   male   316.
```

Note that there is still no evidence here of the levelling off of the male mercury levels, which happened after age 25, and there was quite a lot of variability there as well.

My take is that if you have numerical data, that's what you should use, rather than breaking it up into categories. Those dolphins in the 25-60 age group, we don't know much about their actual ages, and we've seen that age and mercury concentration are closely related, so we are throwing away information by categorizing. Some people break things up into categories because that's what they know how to analyze, but that is never a good reason for you: it means that you need to go learn how to handle the original quantitative data!

Notes

1. This actually creates a copy of the original column, so if you look you now have two columns with the same thing in them, one with a bad name and one with a good one.