STAC33

At-home Final Exam

Due on Thursday April 9 at 1:00 pm EDT. Accessability students may submit the exam later, according to their accommodations. It is your responsibility to submit the exam no later than it is due. Unauthorized late exams may be subject to a penalty.

Answer the four questions below. Hand in your code, output and explanations, by creating an R Notebook and Previewing it to make an HTML file, exactly as you have done for assignments. Hand this in on Quercus, and check, by downloading your file and looking at it, that you have handed in what you intended to hand in.

This exam is open book and open Internet. That means that you may use information from any website, but you may not seek or obtain help from any other person (for example, by asking a question yourself on a forum).

In this exam, "display" means displaying enough of the results to show that you have the right idea. You do not need to display *all* the results. For example, displaying ten lines of a dataframe is enough.

If you have questions, make your best effort to answer them yourself. If you are not clear about what you need to do, make your best assumption about what I am asking you to do, and *state that assumption clearly*. If you get stuck on an early part of a question, and therefore cannot do the later parts, bear in mind that there are no surprises in the early parts of any of the questions, and you may simply need to come back to it later. Get as close as you can to answering the question as asked, or at least getting *an* answer, even if you do it a different way.

If you absolutely must, email me at `ken.butler@utoronto.ca`, but only do so if something is truly impossible, such as a data file that cannot be accessed. If it can be accessed but you cannot read it in, that is *your* problem to solve, and do not expect any help solving it. I will get to emails as soon as I can, but that may not be quickly.

To begin (you need to do this as well):

```
library(tidyverse)

## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.0       v purrr   0.3.3
## v tibble  3.0.0       v dplyr   0.8.99.9002
## v tidyr   1.0.2       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.5.0

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

library(broom)
```

If these give errors, make sure you have installed these packages first.

The questions follow. There are 4 pages in total.

1. The United Kingdom has an official Government Wine Cellar, and the people in charge of it keep track of the number of bottles of wines and spirits used from it for official government functions (for example, when a Member of Parliament is hosting a distinguished guest). The data are in `http://ritsokiguess.site/STAC33/winecellar.csv` as a `.csv` file. It has four columns:

   - the country or region from which the wine comes
   - its vintage (the year in which it was made)
   - the name of the wine
   - the total number of bottles of that wine consumed (over the year for which the data were collected).

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

   Solution: The usual:

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

```
## Parsed with column specification:
## cols(
##   country_region = col_character(),
##   vintage = col_character(),
##   product_name = col_character(),
##   consumption = col_double()
## )
```

```
winecellar
```

```
## # A tibble: 199 x 4
##    country_region vintage product_name                                 consumption
##    <chr>          <chr>   <chr>                                              <dbl>
##  1 ALSACE         2007    Alsace Pinot Gris Les Elements Dom. Bott ~            5
##  2 ALSACE         2007    Riesling Domaine Marcel Deiss                         4
##  3 ALSACE         2008    Pinot Gris Domaine Rolly Gassmann                     8
##  4 ALSACE         2010    Gewurztraminer Les Princes Abbes Dom Sch~            17
##  5 AUSTRALIA      1995    Lindemanns Pyrus                                      2
##  6 AUSTRALIA      1999    Eileen Hardy Shiraz                                   1
##  7 AUSTRALIA      2002    D'Arenberg The Coppermine Road                        1
##  8 AUSTRALIA      2002    Giaconda Shiraz Warner Vineyard                       3
##  9 AUSTRALIA      2002    Katnook Prodicy Shiraz Coonawarra                     3
## 10 AUSTRALIA      2008    Moss Wood Chardonnay Margaret River                  10
## # ... with 189 more rows
```

   (b) (2 marks) Display only the column that contains the name of the wine.

   Solution: This:

```
winecellar %>% select(product_name)

## # A tibble: 199 x 1
##    product_name
##    <chr>
##  1 Alsace Pinot Gris Les Elements Dom. Bott Geyl 2007
##  2 Riesling Domaine Marcel Deiss
##  3 Pinot Gris Domaine Rolly Gassmann
##  4 Gewurtztraminer Les Princes Abbes Dom Schlumberger
##  5 Lindemanns Pyrus
##  6 Eileen Hardy Shiraz
##  7 D'Arenberg The Coppermine Road
##  8 Giaconda Shiraz Warner Vineyard
##  9 Katnook Prodicy Shiraz Coonawarra
## 10 Moss Wood Chardonnay Margaret River
## # ... with 189 more rows
```

(c) (3 marks) Display all the information about the wines that come from the Loire region, or as much of it as will display for you. To do this, note how the country/region names are specified. (In case it matters to you, the Loire is a river in France; the area around the river is also called Loire.)

Solution: Choosing rows is `filter`, and the countries and regions are ALL CAPS, thus:

```
winecellar %>% filter(country_region=="LOIRE")

## # A tibble: 4 x 4
##   country_region vintage product_name                     consumption
##   <chr>          <chr>   <chr>                                  <dbl>
## 1 LOIRE          2002    Pouilly =-Fume Chateau de Tracey           4
## 2 LOIRE          2006    Sancerre Roland Tissiere                   3
## 3 LOIRE          2010    Sancerre Nuance Vincent Pinard             4
## 4 LOIRE          2010    Sancerre Dom Vacheron                      6
```

There are four of them.

To not match on case, you'll need something like this:

```
winecellar %>%
    filter(str_detect(country_region, regex("Loire", ignore_case=T)))

## # A tibble: 4 x 4
##   country_region vintage product_name                     consumption
##   <chr>          <chr>   <chr>                                  <dbl>
## 1 LOIRE          2002    Pouilly =-Fume Chateau de Tracey           4
## 2 LOIRE          2006    Sancerre Roland Tissiere                   3
## 3 LOIRE          2010    Sancerre Nuance Vincent Pinard             4
## 4 LOIRE          2010    Sancerre Dom Vacheron                      6
```

Or you can convert both `country_region` and `Loire` to lowercase first:

```
winecellar %>% filter(tolower(country_region)==tolower("Loire"))

## # A tibble: 4 x 4
##   country_region vintage product_name                         consumption
##   <chr>          <chr>   <chr>                                      <dbl>
## 1 LOIRE          2002    Pouilly =-Fume Chateau de Tracey               4
## 2 LOIRE          2006    Sancerre Roland Tissiere                       3
## 3 LOIRE          2010    Sancerre Nuance Vincent Pinard                 4
## 4 LOIRE          2010    Sancerre Dom Vacheron                          6
```

There are undoubtedly other ways. Find one that works. I'm expecting you to Google this and find something.

(d) (2 marks) How many different wines are there of each vintage? (Displaying the first few is enough.)

Solution: This is a simple `count`:

```
winecellar %>% count(vintage)

## # A tibble: 40 x 2
##    vintage     n
##  * <chr>   <int>
##  1 1921        1
##  2 1948        1
##  3 1960        1
##  4 1967        1
##  5 1975        1
##  6 1977        3
##  7 1979        1
##  8 1981        1
##  9 1982        1
## 10 1983        3
## # ... with 30 more rows
```

(e) (3 marks) How many *bottles* of wine were consumed of each vintage? (Again, displaying the first few is enough.)

Solution: This means summing up the `consumption` values for each `vintage`, which can be done like this:

```
winecellar %>% count(vintage, wt=consumption)
```

```
## # A tibble: 40 x 2
##    vintage     n
##  * <chr>   <dbl>
##  1 1921     0.75
##  2 1948     0.5
##  3 1960     0.25
##  4 1967     1
##  5 1975     2
##  6 1977    12
##  7 1979     1
##  8 1981     1
##  9 1982     1
## 10 1983    12
## # ... with 30 more rows
```

I don't know whether we've seen one of those; you may have to search for ideas. Another way is this:

```
winecellar %>% group_by(vintage) %>%
    summarize(total=sum(consumption))
```

```
## # A tibble: 40 x 2
##    vintage total
##  * <chr>   <dbl>
##  1 1921     0.75
##  2 1948     0.5
##  3 1960     0.25
##  4 1967     1
##  5 1975     2
##  6 1977    12
##  7 1979     1
##  8 1981     1
##  9 1982     1
## 10 1983    12
## # ... with 30 more rows
```

Either is good, or anything else you can make work.

Extra: the fractional bottles are of spirits:

```
winecellar %>% filter(vintage<1965)
```

```
## # A tibble: 3 x 4
##   country_region vintage product_name           consumption
##   <chr>          <chr>   <chr>                        <dbl>
## 1 COGNAC         1921    Martell                       0.75
## 2 COGNAC         1960    Hine                          0.25
## 3 COGNAC         1948    Frapin Grand Champagne        0.5
```

Brandy, to be precise. Look it up in Wikipedia.

(f) (2 marks) Which vintage had the most bottles consumed, and how many bottles was that? (Note: if your preferred way of doing it returns you 0 rows and no answer, find another way.)

Solution: Find a way to display just the maximum, or the top few. The first way I thought of was to sort them in descending order. Add the `arrange` onto the end of whatever you did in the previous part:

```
winecellar %>% group_by(vintage) %>%
    summarize(total=sum(consumption)) %>%
    arrange(desc(total))

## # A tibble: 40 x 2
##    vintage  total
##    <chr>    <dbl>
##  1 2014/15 1637
##  2 2016     836
##  3 2010     407
##  4 2000     112
##  5 1990      92
##  6 2013      87
##  7 N/A       86.2
##  8 2006      83
##  9 2007      73
## 10 2008      65
## # ... with 30 more rows
```

2014/15, with 1637 bottles consumed. Why there are two years combined into one, I don't know. *State the vintage and the number of bottles.* A look at the data reveals a separate year 2014, but not a separate 2015.

Alternatively, find the row(s) where `total` is equal to the maximum `total`:

```
winecellar %>% count(vintage, wt=consumption) %>%
  filter(n==max(n))

## # A tibble: 1 x 2
##   vintage     n
##   <chr>   <dbl>
## 1 2014/15  1637
```

This ought to work, but appears not to:

```
winecellar %>% group_by(vintage) %>%
    summarize(total=sum(consumption)) %>%
    filter(total==max(total))

## # A tibble: 0 x 2
## # ... with 2 variables: vintage <chr>, total <dbl>
```

This, however, does work:

```
winecellar %>% group_by(vintage) %>%
    summarize(total=sum(consumption)) %>%
    top_n(1)

## Selecting by total

## # A tibble: 1 x 2
##   vintage total
##   <chr>   <dbl>
## 1 2014/15  1637
```

(g) (3 marks) How many bottles of wine from Australia were consumed? I do not want to see total consumption from any other countries or regions.

Solution: I can think of two approaches, both involving `filter`:

- grab just the Australian wines, and then total up the consumption of those
- find the total consumption of wine in all the countries, and pull out Australia from the results.

The first way gives:

```
winecellar %>% filter(country_region=="AUSTRALIA") %>%
  summarize(total=sum(consumption))

## # A tibble: 1 x 1
##   total
##   <dbl>
## 1    25
```

and the second way gives:

```
winecellar %>% group_by(country_region) %>%
  summarize(total=sum(consumption)) %>%
  filter(country_region=="AUSTRALIA")

## # A tibble: 1 x 2
##   country_region total
##   <chr>          <dbl>
## 1 AUSTRALIA         25
```

25 bottles, either way.

2. Back in the days before Internet banking, you had two choices to find out your current balance and recent transactions: you could go into a bank, wait in line and talk to a teller, or you could do your banking over the phone with an automated system (like voicemail). A bank did a survey to find out how satisfied people were with the two systems, and whether the satisfaction depended on the time of day. There were $2 \times 3 = 6$ possible combinations of "contact" (phone or teller) and time of day (morning, afternoon, evening).

The bank called four customers who had recently used each of the combinations of contact and time of day to access the bank (thus they called $4 \times 6 = 24$ people altogether). In the call, the bank asked each customer to rate their satisfaction with their experience, on a scale of 1 (worst) to 10 (best). The data are in `http://ritsokiguess.site/STAC33/bank.csv`.

(a) (1 mark) Read in and display the data.

Solution: This is a `.csv`:

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

```
## Parsed with column specification:
## cols(
##   within = col_double(),
##   morning_phone = col_double(),
##   morning_teller = col_double(),
##   afternoon_phone = col_double(),
##   afternoon_teller = col_double(),
##   evening_phone = col_double(),
##   evening_teller = col_double()
## )
```

```
bank
```

```
## # A tibble: 4 x 7
##   within morning_phone morning_teller afternoon_phone afternoon_teller
##    <dbl>         <dbl>          <dbl>           <dbl>            <dbl>
## 1      1             6              8               3                9
## 2      2             5              7               5               10
## 3      3             8              9               6                6
## 4      4             4              9               5                8
## # ... with 2 more variables: evening_phone <dbl>, evening_teller <dbl>
```

Extra: four rows and seven columns: the six treatment combinations, and the column called `within` that says what number customer each row is within their treatment combination.

(b) (2 marks) Explain briefly how the data are not tidy (that is to say, the data are currently not suitable for making a graph or doing a statistical analysis).

Solution: All the numbers are satisfaction scores; they need to be all in one column with some kind of identification of which contact and time of day they belong to.

Or, the column names are *levels* of categorical variables (actually, combinations of levels) rather than being categorical variables themselves. For this one, you might call the time-of-day and contact combinations `treatment` and say that the column names as they now stand are levels of that categorical variable.

Or, talk about what tidy data would look like here, and how what you have is not it.

(c) (4 marks) Starting from the data you read in, create a tidy data set. There are two steps (or, one more complicated one). Save your tidy data frame and display (some of) it.

Solution: Probably the first thing you think of is `pivot_longer`. Try that first and see what it does here:

```
bank %>% pivot_longer(-within, names_to="treatment", values_to="satisfaction")
```

```
## # A tibble: 24 x 3
##    within treatment        satisfaction
##     <dbl> <chr>                   <dbl>
##  1      1 morning_phone               6
##  2      1 morning_teller              8
##  3      1 afternoon_phone             3
##  4      1 afternoon_teller            9
##  5      1 evening_phone               5
##  6      1 evening_teller              9
##  7      2 morning_phone               5
##  8      2 morning_teller              7
##  9      2 afternoon_phone             5
## 10      2 afternoon_teller           10
## # ... with 14 more rows
```

The treatment column encodes two things. Use `separate` to split these into two columns:

```
bank %>% pivot_longer(-within, names_to="treatment",
                      values_to="satisfaction") %>%
    separate(treatment, into=c("time_of_day", "contact"), sep="_") -> bank_tidy
bank_tidy
```

```
## # A tibble: 24 x 4
##    within time_of_day contact satisfaction
##     <dbl> <chr>       <chr>          <dbl>
##  1      1 morning     phone              6
##  2      1 morning     teller             8
##  3      1 afternoon   phone              3
##  4      1 afternoon   teller             9
##  5      1 evening     phone              5
##  6      1 evening     teller             9
##  7      2 morning     phone              5
##  8      2 morning     teller             7
##  9      2 afternoon   phone              5
## 10      2 afternoon   teller            10
## # ... with 14 more rows
```

Two points for each thing: the `pivot_longer` and the `separate`.

I said there was a one-step solution. This is the "advanced" version of `pivot_longer`:

```
bank %>% pivot_longer(-within,
                      names_to=c("time_of_day", "contact"),
                      values_to="satisfaction",
                      names_sep="_")

## # A tibble: 24 x 4
##    within time_of_day contact satisfaction
##     <dbl> <chr>       <chr>          <dbl>
##  1      1 morning     phone              6
##  2      1 morning     teller             8
##  3      1 afternoon   phone              3
##  4      1 afternoon   teller             9
##  5      1 evening     phone              5
##  6      1 evening     teller             9
##  7      2 morning     phone              5
##  8      2 morning     teller             7
##  9      2 afternoon   phone              5
## 10      2 afternoon   teller            10
## # ... with 14 more rows
```

It will probably work without the `names_sep`, since `_` is one of the things that the pivot functions recognize as a separator automatically. The full four if you successfully do it this way.

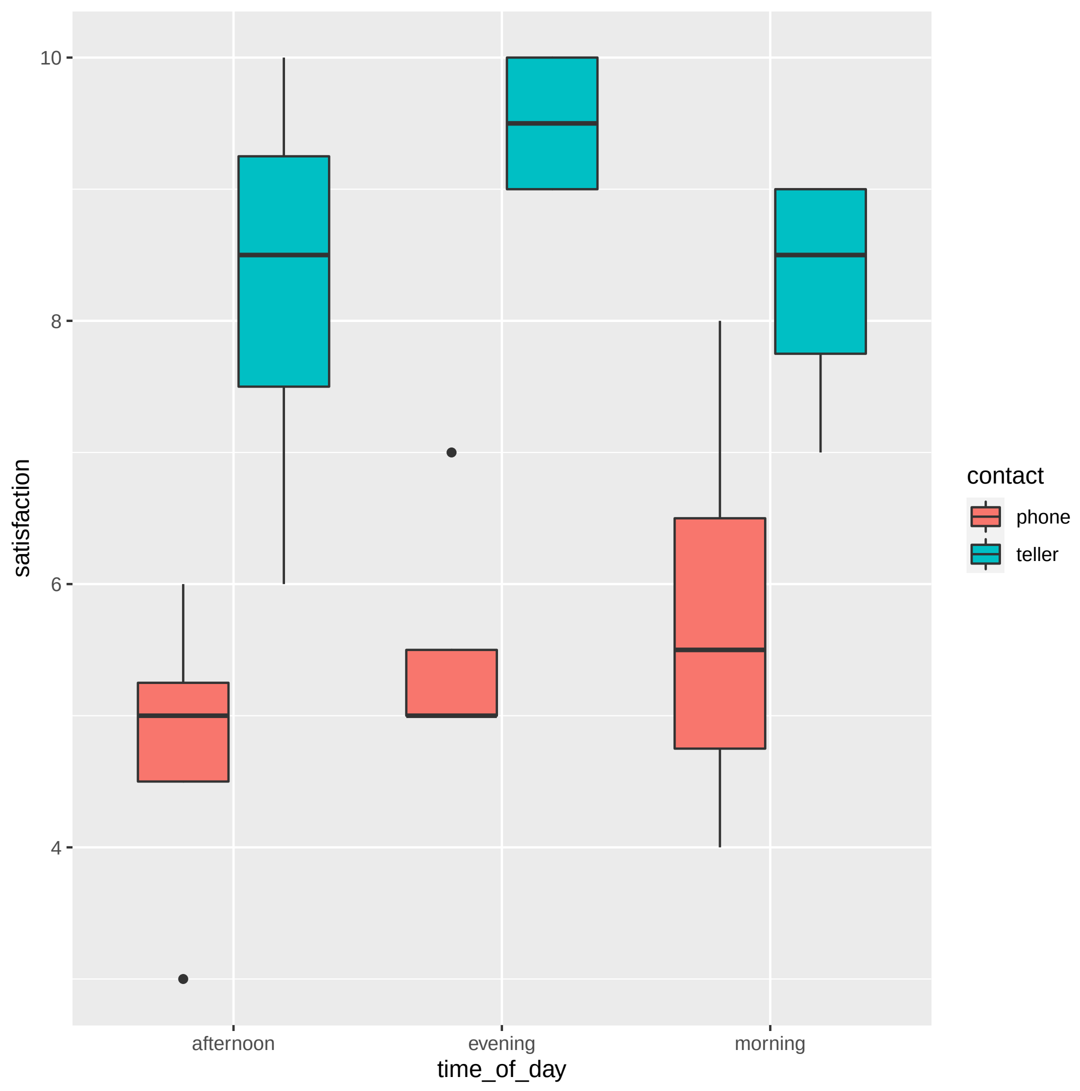
The logic here is that by supplying two things to `names_to`, `pivot_longer` will also `separate` them, to save you having to do it by hand.

(d) (3 marks) Make a suitable graph using your tidy data frame.

Solution: This was why I said to save it earlier.

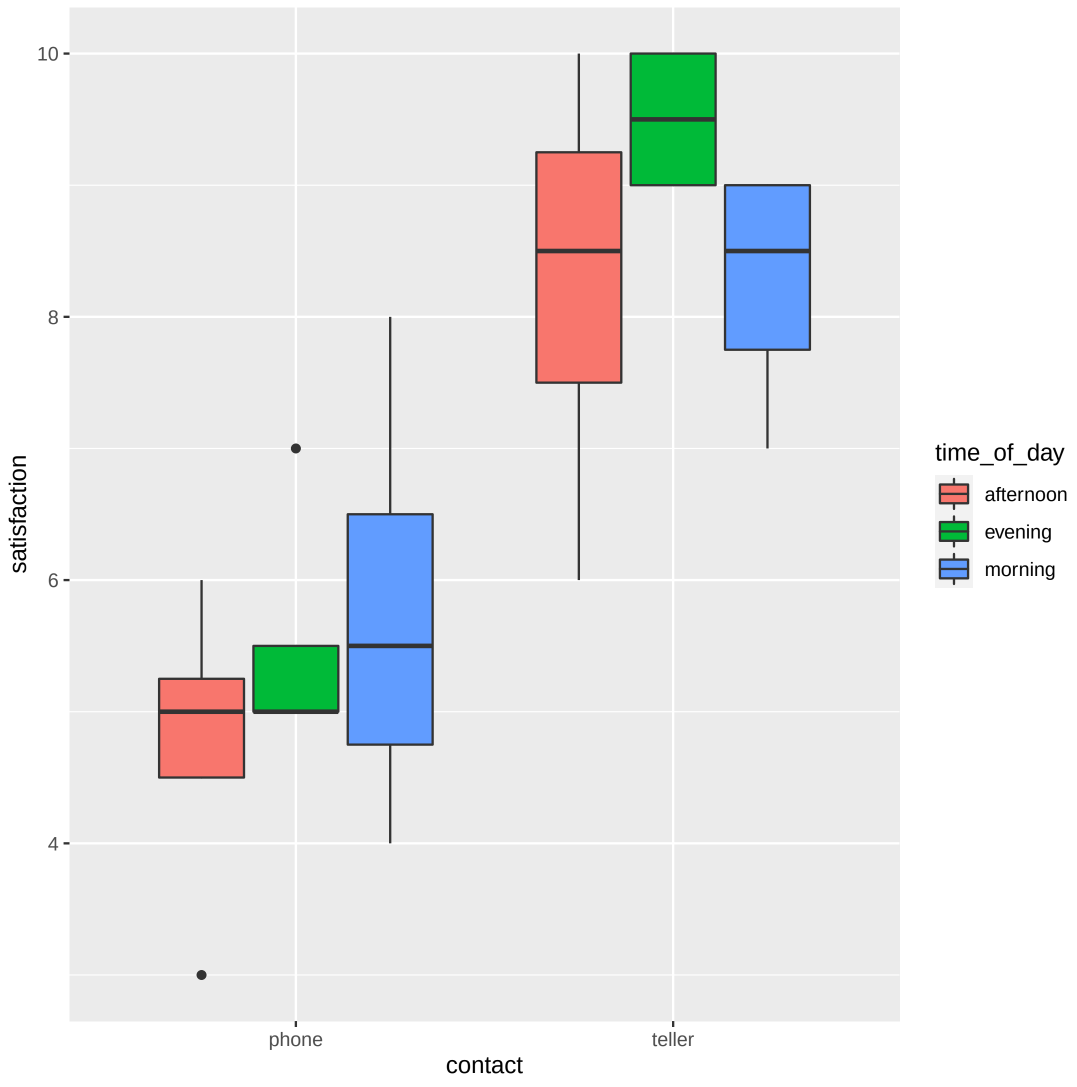
Go back to your list of graphs, and find that with two categorical variables and one quantitative one, what you want is a grouped boxplot, thus:

```
ggplot(bank_tidy, aes(x=time_of_day, fill=contact, y=satisfaction)) +
    geom_boxplot()
```



Three points for that. I prefer having more levels on the x-axis and fewer colours, but for what you're doing next, it's perfectly good to do this (and therefore three points also):

```
ggplot(bank_tidy, aes(fill=time_of_day, x=contact, y=satisfaction)) +
    geom_boxplot()
```



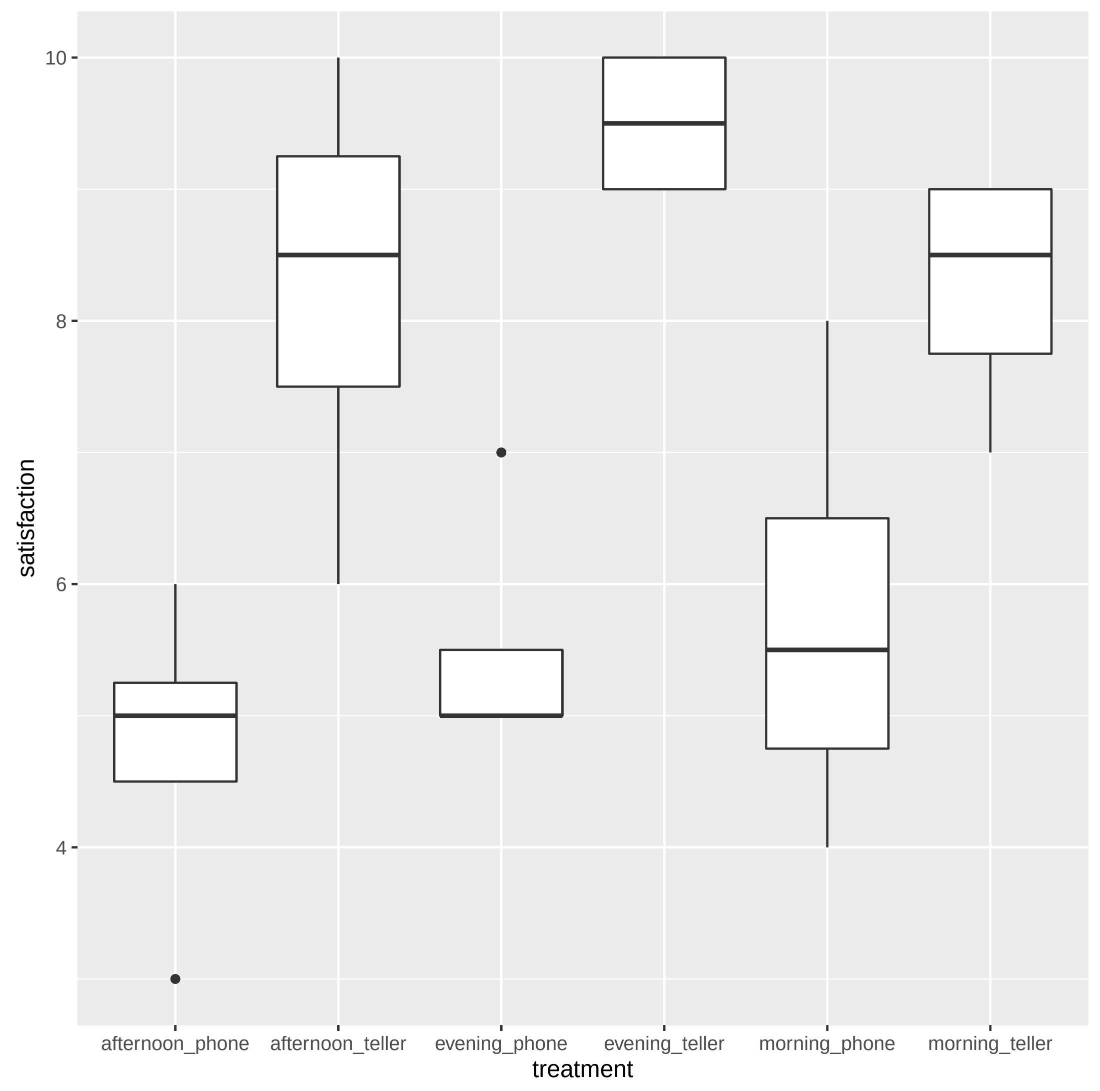
Also, if you only got halfway with your tidying, you would have ended up with something like this:

```
bank %>% pivot_longer(-within, names_to="treatment",
                      values_to="satisfaction")
```

```
## # A tibble: 24 x 3
##    within treatment        satisfaction
##     <dbl> <chr>                   <dbl>
##  1      1 morning_phone               6
##  2      1 morning_teller              8
##  3      1 afternoon_phone             3
##  4      1 afternoon_teller            9
##  5      1 evening_phone               5
##  6      1 evening_teller              9
##  7      2 morning_phone               5
##  8      2 morning_teller              7
##  9      2 afternoon_phone             5
## 10      2 afternoon_teller           10
## # ... with 14 more rows
```

From this, you have *one* categorical variable `treatment` and one quantitative one `satisfaction`, so this is an ordinary boxplot:

```
bank %>% pivot_longer(-within, names_to="treatment",
                      values_to="satisfaction") %>%
    ggplot(aes(x=treatment, y=satisfaction)) + geom_boxplot()
```



This is a logical deduction from what you obtained in the previous part, but you have made the plot easier to draw. Two points.

(e) (2 marks) What does your plot tell you about customers' preferences for contact method (phone or teller)? Is that different for different times of day? Explain briefly.

Solution: I know I promised to simplify my question parts, and this one has two questions in it again. But the two parts are linked.

On my preferred boxplot (adapt for yours), the three blue boxes for `teller` are all noticeably above the corresponding red ones for `phone`. Thus, customers prefer teller over phone at all times of the day; if you prefer, the preference for teller over phone does not depend (much) on the time of day.

If you had one of the other plots in the previous part, you will have to work a bit harder to get to this point, but you should be able to manage it. On the second variant of the grouped boxplots, for example, you can see that the three different-coloured boxplots for `teller` are clearly higher than the ones for `phone`.

If you want to, you can say a bit more about how the preference for teller over phone is largest in the evening (the satisfaction for teller is highest in the evening), or that the preference for teller over phone is (slightly) less in the morning, because the satisfaction with phone is (slightly) higher in the morning.

One thing to avoid is comparing the three *times of day* for each *contact method*. That is exactly the opposite of what I asked about. (There is, in any case, not much difference in time of day for each contact method. The interesting thing here is that people like seeing a teller much more than using the phone-banking system.)

Extra: the idea of comparing teller with phone at *each* time of day is known as "simple effects" (of contact for each time of day). This is exactly conditioning on one categorical variable (time of day) and looking at the effect of the other one (contact method). Here, the simple effects are all about the same, which is an indication that there is no *interaction* between contact method and time of day, which is another way of saying that the effect of contact method is the same for all times of day. Thus it makes sense to talk about *the* effect of contact method (for all times of day) and *the* effect of time of day (for both contact methods).

Some of you will recognize this as a two-way ANOVA (which we don't do in this class). The first thing you do is to fit a model with an interaction and test that. We don't expect a significant interaction here (for reasons discussed above). I use `aov` here since I am planning to use Tukey later:

```
bank.1 <- aov(satisfaction~contact*time_of_day, data=bank_tidy)
summary(bank.1)
```

```
##                     Df Sum Sq Mean Sq F value  Pr(>F)
## contact              1  66.67   66.67  41.379 4.7e-06 ***
## time_of_day          2   4.00    2.00   1.241   0.313
## contact:time_of_day  2   2.33    1.17   0.724   0.498
## Residuals           18  29.00    1.61
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Having found that the interaction is not significant, we *look no further*. We should take out the non-significant interaction and re-fit, to get better tests for the other things (in the same way that in regression, we remove non-significant terms first to get a better, easier-to-explain model):

```
bank.2 <- aov(satisfaction~contact+time_of_day, data=bank_tidy)
summary(bank.2)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## contact      1  66.67   66.67  42.553 2.34e-06 ***
## time_of_day  2   4.00    2.00   1.277    0.301
## Residuals   20  31.33    1.57
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

There is a strongly significant effect of contact method and no significant effect of time of day. If we run Tukey now, we get this:

```
TukeyHSD(bank.2)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = satisfaction ~ contact + time_of_day, data = bank_tidy)
##
## $contact
##                   diff      lwr     upr   p adj
## teller-phone 3.333333 2.267426 4.39924 2.3e-06
##
## $time_of_day
##                   diff        lwr      upr     p adj
## evening-afternoon  1.0 -0.5833455 2.583346 0.2696098
## morning-afternoon  0.5 -1.0833455 2.083346 0.7079522
## morning-evening   -0.5 -2.0833455 1.083346 0.7079522
```

The Tukey doesn't really tell us very much. `contact` was significant, but there are only two contact methods, so we already knew they were different (and, from the plot, people liked seeing a teller more than using the phone-banking system). There was no significant effect of time of day, so the disciplined statistician in you will not even look at the Tukey for that; even if you do, you'll see no significant differences at all.

Some of you may have been worried about the outliers on the boxplots, and its implication for normality and equal variances (which we technically need here too). But, bear in mind that there are only four observations per treatment combination, so the distributions won't look very normal even if they actually are (and their spreads won't look very equal either). In addition to that, since this is a two-way ANOVA, we don't obviously have anything like Mood's median test available to us (more discussion below). I think the results are so clear that the actual P-values could be off by a bit and we'd still be safe, so I'm going to call this good enough.

Another way to assess the normality assumption is to fit the model as an `lm` and look at its residuals. This is potentially better, since we look at one set of 24 residuals all together, which ought to behave better than six sets of four residuals:

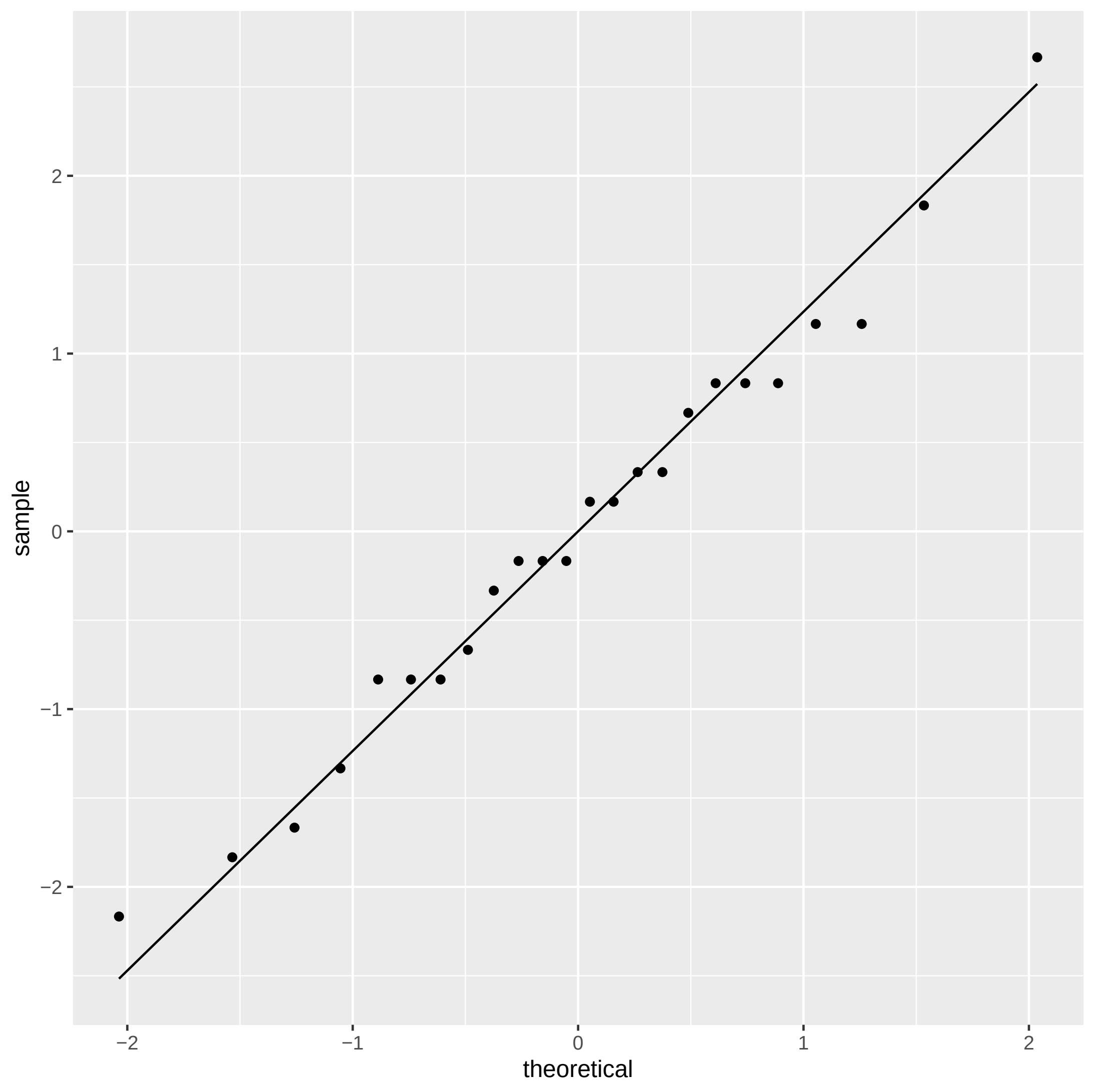
```
bank.2=lm(satisfaction~contact+time_of_day, data=bank_tidy)
drop1(bank.2, test="F")
```

```
## Single term deletions
##
## Model:
## satisfaction ~ contact + time_of_day
##             Df Sum of Sq    RSS    AIC F value    Pr(>F)
## <none>                   31.333 14.399
## contact      1    66.667 98.000 39.766 42.5532 2.337e-06 ***
## time_of_day  2     4.000 35.333 13.283  1.2766    0.3008
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

`drop1` is better than `summary` since we want to test for an effect of *all* the levels of the categorical variables. The tests are the same as in the `aov`.

Now, treat this as a regression and look at the normal quantile plot of the residuals:

```
ggplot(bank.2, aes(sample=.resid)) + stat_qq() + stat_qq_line()
```



That looks completely acceptable.

If you want to make Mood's median test work, one way is to turn our two factors `contact` and `time_of_day` into one, in the fashion of my `treatment` above, and then run Mood's median test on that:

```
library(smmr)
bank %>% pivot_longer(-within, names_to="treatment",
                      values_to="satisfaction") -> d
median_test(d, satisfaction, treatment)
```

```
## $table
##                    above
## group               above below
##   afternoon_phone       0     4
##   afternoon_teller      3     1
##   evening_phone         0     3
##   evening_teller        4     0
##   morning_phone         1     3
##   morning_teller        3     0
##
## $test
##        what        value
## 1 statistic 16.000000000
## 2        df  5.000000000
## 3   P-value  0.006844074
```

and then

```
pairwise_median_test(d, satisfaction, treatment)
```

```
## # A tibble: 15 x 4
##    g1               g2               p_value adj_p_value
##    <chr>            <chr>              <dbl>       <dbl>
##  1 afternoon_phone  afternoon_teller 0.0143       0.215
##  2 afternoon_phone  evening_phone    0.386        1
##  3 afternoon_phone  evening_teller   0.00468      0.0702
##  4 afternoon_phone  morning_phone    0.709        1
##  5 afternoon_phone  morning_teller   0.00468      0.0702
##  6 afternoon_teller evening_phone    0.157        1
##  7 afternoon_teller evening_teller   0.136        1
##  8 afternoon_teller morning_phone    0.157        1
##  9 afternoon_teller morning_teller   1            1
## 10 evening_phone    evening_teller   0.00468      0.0702
## 11 evening_phone    morning_phone    0.505        1
## 12 evening_phone    morning_teller   0.0143       0.215
## 13 evening_teller   morning_phone    0.00468      0.0702
## 14 evening_teller   morning_teller   0.0455       0.683
## 15 morning_phone    morning_teller   0.157        1
```

The overall Mood's median test is significant, though probably with a warning (out of `chisq.test`) that the expected frequencies are too small. The pairwise median tests, though, have *no* significant results! Partly this is because, with six treatment combinations, there are $\binom{6}{2} = 15$ pairs of treatments, and the Bonferroni adjustment says that, since we are doing 15 simultaneous tests, we have to multiply each P-value by 15. Also, partly this is because we have small frequencies in each table. Basically, all of this is not to be taken too seriously.

The ones that are closest to being significant are all comparisons between teller and phone at the same or different times.

I think the "simple effects" idea here is really like this, but comparing each contact method for morning, then for afternoon, then for evening. This would be more focused, since we are not comparing different times of day, and the Bonferroni adjustment would only be to multiply by 3 rather than 15, since we are only doing three simultaneous tests. That would give something like this:

Comparison	P-value	Adjusted P-value
Morning phone vs. teller	0.157	0.471
Afternoon phone vs. teller	0.0143	0.0429
Evening phone vs. teller	0.0047	0.0141

By limiting which comparisons we are making, we do get some significant ones now, though, curiously, morning does not show a significant difference. The boxplots do show some overlap for morning, and with the small frequencies, this is probably enough to stop the Mood's median test from being significant.

(f) (1 mark) The data as I originally found it was as shown in `http://ritsokiguess.site/STAC33/bank_original.csv`, with the four numeric values for each treatment group separated by colons. Read in and display this arrangement of the data.

Solution: As usual, though the file looks rather odd:

```
my_url="http://ritsokiguess.site/STAC33/bank_original.csv"
bank_different <- read_csv(my_url)

## Parsed with column specification:
## cols(
##   timeofday = col_character(),
##   phone = col_character(),
##   teller = col_character()
## )

bank_different

## # A tibble: 3 x 3
##   timeofday phone   teller
##   <chr>     <chr>   <chr>
## 1 morning   6:5:8:4 8:7:9:9
## 2 afternoon 3:5:6:5 9:10:6:8
## 3 evening   5:5:7:5 9:10:10:9
```

(g) (4 marks) Find a way to get this format of data tidy.

Solution: There is a slick way, which is called `separate_rows`. First, get those colon-separated lists of numbers into one column. Then `separate_rows` puts each thing separated by a colon (in `sep`) in its own row, copying the other rows as necessary:

```
bank_different %>%
    pivot_longer(-timeofday, names_to="contact", values_to="satisfaction") %>%
    separate_rows(satisfaction, sep=":", convert=T)
```

```
## # A tibble: 24 x 3
##    timeofday contact satisfaction
##    <chr>     <chr>          <int>
##  1 morning   phone              6
##  2 morning   phone              5
##  3 morning   phone              8
##  4 morning   phone              4
##  5 morning   teller             8
##  6 morning   teller             7
##  7 morning   teller             9
##  8 morning   teller             9
##  9 afternoon phone              3
## 10 afternoon phone              5
## # ... with 14 more rows
```

The `convert=T` says to take the values that were originally text and convert them into what they look like (numbers).

If you didn't track that down, there is still hope. Do the `pivot_longer` again, but use `separate` to physically separate the four values one by one:

```
bank_different %>%
    pivot_longer(-timeofday, names_to="contact", values_to="satisfaction") %>%
    separate(satisfaction, into=c("rep1", "rep2", "rep3", "rep4"),
             convert=T, sep=":")
```

```
## # A tibble: 6 x 6
##   timeofday contact  rep1  rep2  rep3  rep4
##   <chr>     <chr>   <int> <int> <int> <int>
## 1 morning   phone       6     5     8     4
## 2 morning   teller      8     7     9     9
## 3 afternoon phone       3     5     6     5
## 4 afternoon teller      9    10     6     8
## 5 evening   phone       5     5     7     5
## 6 evening   teller      9    10    10     9
```

I called these `rep`-something since they are the four replicated values within each treatment combination. Now pivot these longer again to get them all in one column:

```
bank_different %>%
    pivot_longer(-timeofday, names_to="contact", values_to="satisfaction") %>%
    separate(satisfaction, into=c("rep1", "rep2", "rep3", "rep4"),
             convert=T, sep=":") %>%
    pivot_longer(starts_with("rep"), names_to="rep", values_to="satisfaction")
```

```
## # A tibble: 24 x 4
##    timeofday contact rep   satisfaction
##    <chr>     <chr>   <chr>        <int>
##  1 morning   phone   rep1             6
##  2 morning   phone   rep2             5
##  3 morning   phone   rep3             8
##  4 morning   phone   rep4             4
##  5 morning   teller  rep1             8
##  6 morning   teller  rep2             7
##  7 morning   teller  rep3             9
##  8 morning   teller  rep4             9
##  9 afternoon phone   rep1             3
## 10 afternoon phone   rep2             5
## # ... with 14 more rows
```

We have gained an extra column `rep` that plays the same role as the `within` I had earlier. Note that I had a column `satisfaction`, lost it, and then got another one back!

The `convert=T` is necessary because the satisfaction scores are numbers, but the values combined and separated by colons are text. Expect the grader to check that your final values are numbers.

3. All Greens is a franchise store that sells house plants and lawn and garden supplies. Each All Greens store is owned and managed by private individuals. Some friends have asked you to go into business with them to open a new All Greens store in the suburbs of San Diego (in California). The national franchise operator sends you information about 27 All Greens stores in California, as follows:

   - Annual net sales (in thousands of dollars)
   - Floor display area in store (thousands of square feet)
   - Value of store inventory (in thousands of dollars)
   - Size of sales district (thousands of households living within a 5-mile radius of the store)
   - Number of competing stores in sales district.

   The data file is at `http://ritsokiguess.site/STAC33/allgreen.txt`.

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

   Solution: You might guess (or better, check by looking at the file) that the data values are separated by spaces, and so they are:

```
my_url <- "http://ritsokiguess.site/STAC33/allgreen.txt"
allgreen <- read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   sales = col_double(),
##   display = col_double(),
##   inventory = col_double(),
##   advertising = col_double(),
##   sales_size = col_double(),
##   competing = col_double()
## )
```

```
allgreen
```

```
## # A tibble: 27 x 6
##    sales display inventory advertising sales_size competing
##    <dbl>   <dbl>     <dbl>       <dbl>      <dbl>     <dbl>
##  1   231     3         294         8.2        8.2        11
##  2   156     2.2       232         6.9        4.1        12
##  3    10     0.5       149         3          4.3        15
##  4   519     5.5       600        12         16.3         1
##  5   437     4.4       567        10.5       14.1         5
##  6   487     4.8       571        11.8       12.7         4
##  7   299     3.1       512         8.1       10.1        10
##  8   195     2.5       347         7.7        8.4        12
##  9    20     1.2       212         3.3        2.1        15
## 10    68     0.6       102         4.9        4.7         8
## # ... with 17 more rows
```

There is another column `advertising` that I forgot to mention, which is the size of the advertising budget.

(b) (3 marks) Our aim is to predict net sales, and to see which of the other variables help to predict it. Fit a linear regression predicting net sales from all of the other variables, and display the results.

Solution: `lm`, as ever, with the explanatory variables separated by plus signs:

```
allgreen.1 <- lm(sales ~ display + inventory + advertising +
                    sales_size + competing, data=allgreen)
```

If it is too long for one line on your display, split the line somewhere. R will recognize the line as incomplete wherever you split it, because it is waiting for a close-bracket.

The `summary` is what I want:

```
summary(allgreen.1)
```

```
##
## Call:
## lm(formula = sales ~ display + inventory + advertising + sales_size +
##     competing, data = allgreen)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -25.795 -10.297  -4.299   5.002  40.938
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -13.75392   30.91717  -0.445 0.660970
## display      16.41732    3.65879   4.487 0.000203 ***
## inventory     0.17594    0.05955   2.954 0.007572 **
## advertising  11.43898    2.61496   4.374 0.000265 ***
## sales_size   13.21008    1.80256   7.328 3.26e-07 ***
## competing    -5.54914    1.75254  -3.166 0.004653 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 18.25 on 21 degrees of freedom
## Multiple R-squared:  0.9927,Adjusted R-squared:  0.991
## F-statistic: 571.7 on 5 and 21 DF,  p-value: < 2.2e-16
```

Equivalently, from `broom`:

```
glance(allgreen.1)
```

```
## # A tibble: 1 x 11
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <int>  <dbl> <dbl> <dbl>
## 1     0.993         0.991  18.3      572. 1.09e-21     6  -113.  241.  250.
## # ... with 2 more variables: deviance <dbl>, df.residual <int>
```

```
tidy(allgreen.1)
```

```
## # A tibble: 6 x 5
##   term        estimate std.error statistic     p.value
##   <chr>          <dbl>     <dbl>     <dbl>       <dbl>
## 1 (Intercept)  -13.8     30.9       -0.445 0.661
## 2 display       16.4      3.66       4.49  0.000203
## 3 inventory      0.176    0.0596     2.95  0.00757
## 4 advertising   11.4      2.61       4.37  0.000265
## 5 sales_size    13.2      1.80       7.33  0.000000326
## 6 competing     -5.55     1.75      -3.17  0.00465
```

That's good too.

I intended you to include `advertising`, but you could reasonably omit it, with these results:

```r
allgreen.1a <- lm(sales ~ display + inventory +
                    sales_size + competing, data=allgreen)
summary(allgreen.1a)
```

```
##
## Call:
## lm(formula = sales ~ display + inventory + sales_size + competing,
##     data = allgreen)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -32.984 -17.147  -6.136  11.660  45.783
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 61.56880   34.68351   1.775 0.089710 .
## display     14.10483    4.89002   2.884 0.008608 **
## inventory    0.36016    0.05688   6.332 2.26e-06 ***
## sales_size  11.52151    2.37822   4.845 7.68e-05 ***
## competing   -9.43145    2.04117  -4.621 0.000133 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 24.65 on 22 degrees of freedom
## Multiple R-squared:  0.9861,Adjusted R-squared:  0.9835
## F-statistic: 389.1 on 4 and 22 DF,  p-value: < 2.2e-16
```

I suspect the impact on other parts is minimal.

(c) (2 marks) Would you consider modifying this regression model at all? Explain briefly why or why not.

Solution: No, because all the explanatory variables are significant in this regression. (Or, if your regression came out different from mine, say the appropriate thing for your regression.)

The intercept is not up for grabs and has to stay.

There is a largish clue that we are not going to modify this regression, in that none of the other parts of this question talk about what you might do.

Talking about R-squared is not helpful because you could have a high R-squared but still have some x-variables that need to be removed. For example, I could have added an x that was entirely random numbers to this regression; R-squared would still have been very high, but the x we added should not have been.

You could also reasonably take this as an invitation to look at something like plots of sales against everything else and note that these look straight, so that you need all the x's but you don't need to transform them anyhow.

(d) (2 marks) Explain briefly why the sign (positive or negative) of the Estimate for `competing` makes practical sense.

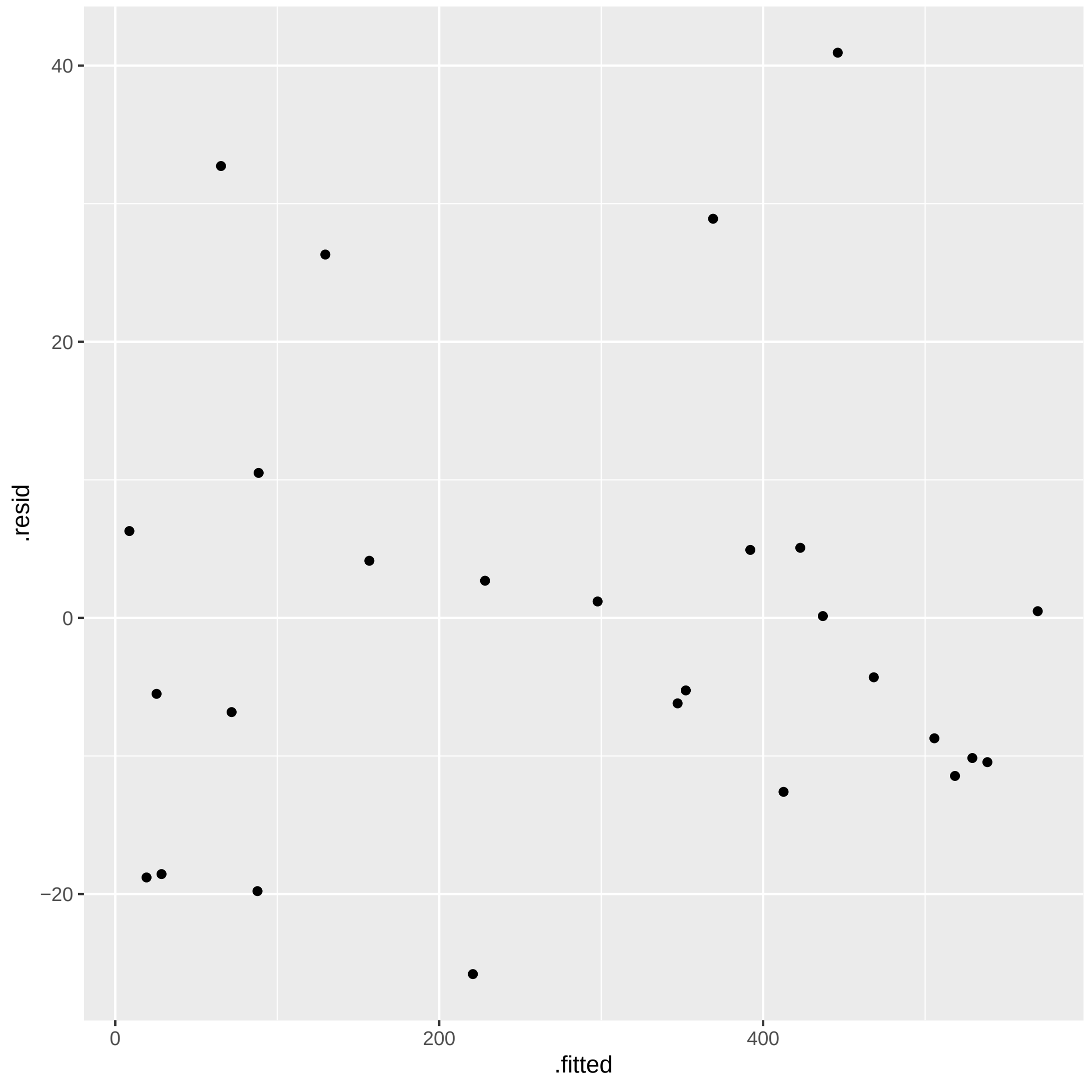
Solution: My Estimate is -5.55, negative (-9.43, also negative, if you omitted `advertising`). This means that if you increase the number of competing scores by 1, leaving everything else the same (or, if you compare two stores that are identical except that one of them has one more competing store than the other), the net sales is predicted to go *down* by 5.55 thousand dollars. One point.

This makes practical sense because if a store has more competition, it will have smaller sales than one that has less nearby competition. (The same number of customers will be spread among more stores, so the sales at any one would be expected to be less.) The other point.

(e) (2 marks) Make a plot of the residuals against the fitted values.

Solution: The usual thing, treating the fitted model object as if it were a data frame:

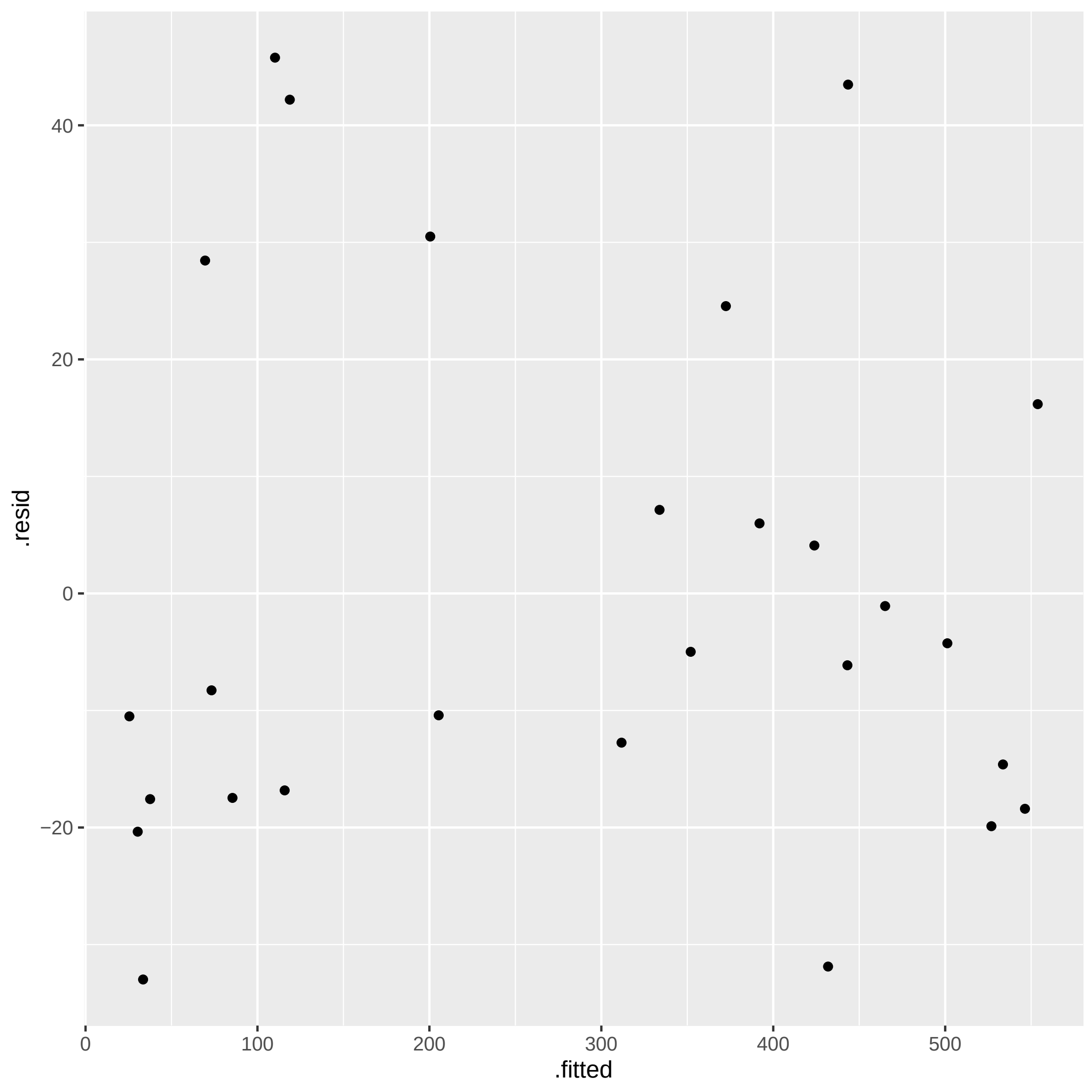
```
ggplot(allgreen.1, aes(x=.fitted, y=.resid)) + geom_point()
```



I think it's better *not* to have a smooth trend, though put one on there if you want (but don't take it too seriously in the next part). See discussion at the end of the final part.

Alternatively:

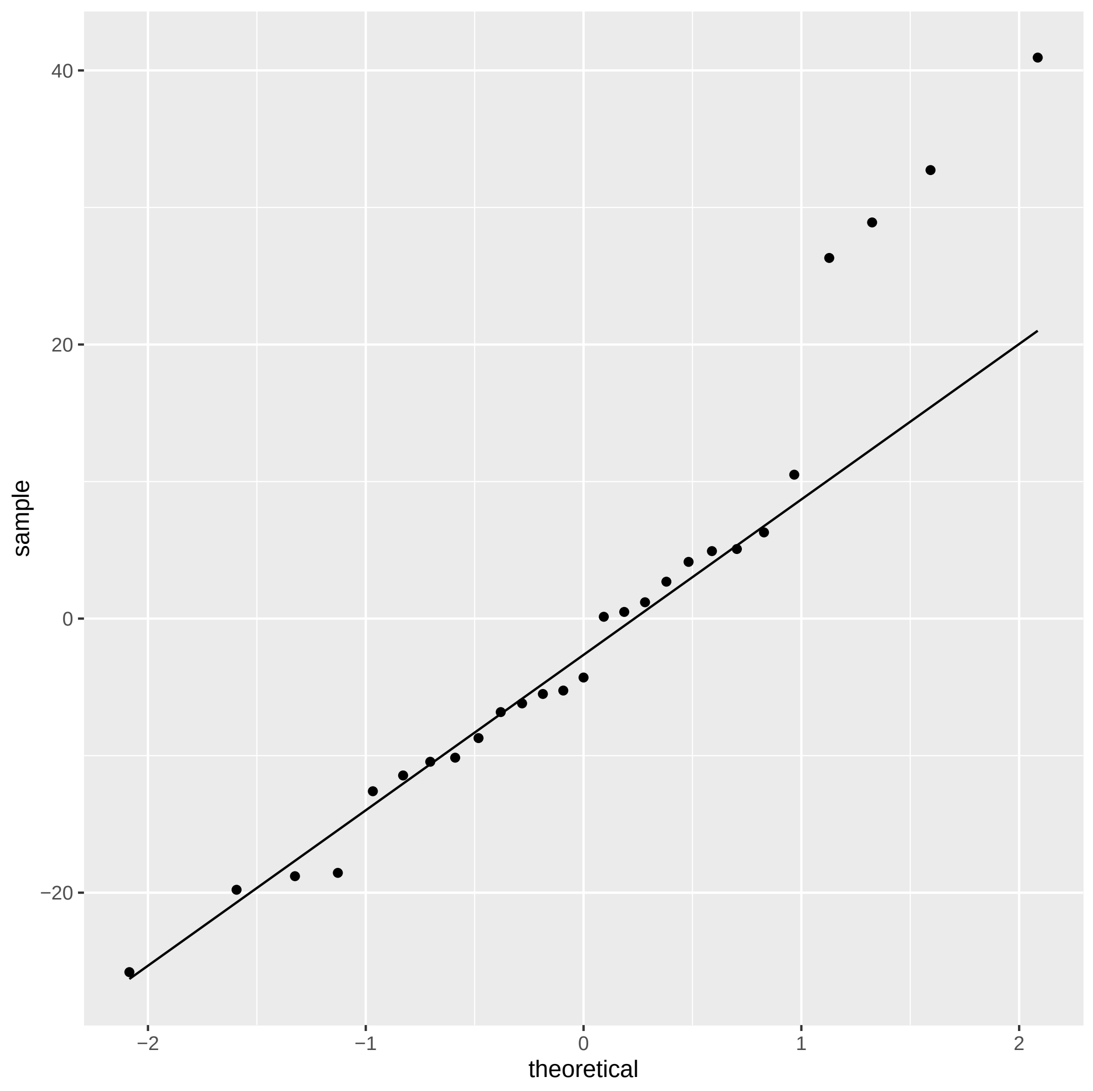
```
ggplot(allgreen.1a, aes(x=.fitted, y=.resid)) + geom_point()
```



Use something we learned in this course, ie. *not* base `plot`. (Using `plot` will cost you a half point, and if this is your thinking, you probably won't figure out (g).)

Extra: I didn't ask about the normality of the residuals. You might reasonably be concerned about that given that the residuals seem to go further up than they do down:

```
ggplot(allgreen.1, aes(sample=.resid)) + stat_qq() + stat_qq_line()
```



Those highest four residuals look a bit too high.

(f) (2 marks) Do you see any problems on your plot of residuals against fitted values? Explain briefly.

Solution: This, for my money, is a classic random formless scatter. There are no trends or curves that I see, or any evidence of fanning-out. Argue for something different if you want, though you'll need to make it convincing.

(g) (6 marks) Create a *single* `ggplot` showing the residuals against each of the explanatory variables.

Solution: This requires some thinking. The smoothest way is to use `augment` from `broom` to make a data frame with data and regression stuff in it:

```
allgreen.1 %>% augment(allgreen)
```

```
## # A tibble: 27 x 13
##    sales display inventory advertising sales_size competing .fitted .se.fit
##    <dbl>   <dbl>     <dbl>       <dbl>      <dbl>     <dbl>   <dbl>   <dbl>
##  1   231     3         294         8.2        8.2        11   228.     8.14
##  2   156     2.2       232         6.9        4.1        12   130.     7.22
##  3    10     0.5       149         3          4.3        15    28.6    7.24
##  4   519     5.5       600        12         16.3         1   529.     6.18
##  5   437     4.4       567        10.5       14.1         5   437.     5.68
##  6   487     4.8       571        11.8       12.7         4   446.     5.37
##  7   299     3.1       512         8.1       10.1        10   298.     8.56
##  8   195     2.5       347         7.7        8.4        12   221.     7.11
##  9    20     1.2       212         3.3        2.1        15    25.5    7.63
## 10    68     0.6       102         4.9        4.7         8    87.8    9.92
## # ... with 17 more rows, and 5 more variables: .resid <dbl>, .hat <dbl>,
## #   .sigma <dbl>, .cooksd <dbl>, .std.resid <dbl>
```

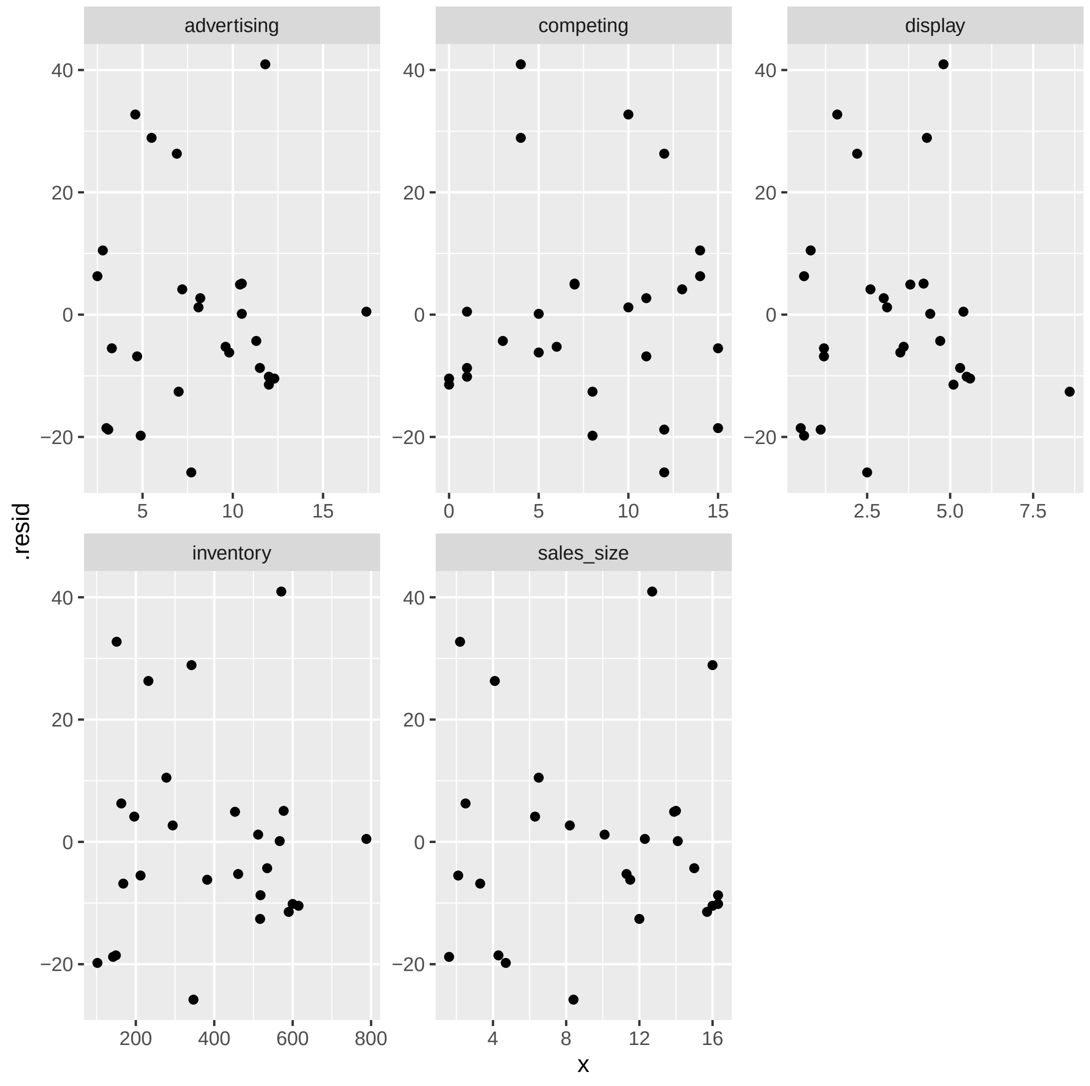
then pivot this longer to get all the x-values in one column, labelled by which x-variable they were:

```
allgreen.1 %>% augment(allgreen) %>%
    pivot_longer(display:competing, names_to="xname", values_to="x")
```

```
## # A tibble: 135 x 10
##    sales .fitted .se.fit .resid  .hat .sigma .cooksd .std.resid xname           x
##    <dbl>   <dbl>   <dbl>  <dbl> <dbl>  <dbl>   <dbl>      <dbl> <chr>       <dbl>
##  1   231    228.    8.14   2.69 0.199   18.7 0.00113      0.165 display       3
##  2   231    228.    8.14   2.69 0.199   18.7 0.00113      0.165 inventory   294
##  3   231    228.    8.14   2.69 0.199   18.7 0.00113      0.165 advertisi~    8.2
##  4   231    228.    8.14   2.69 0.199   18.7 0.00113      0.165 sales_size    8.2
##  5   231    228.    8.14   2.69 0.199   18.7 0.00113      0.165 competing    11
##  6   156    130.    7.22  26.3  0.156   17.6 0.0762       1.57  display       2.2
##  7   156    130.    7.22  26.3  0.156   17.6 0.0762       1.57  inventory   232
##  8   156    130.    7.22  26.3  0.156   17.6 0.0762       1.57  advertisi~    6.9
##  9   156    130.    7.22  26.3  0.156   17.6 0.0762       1.57  sales_size    4.1
## 10   156    130.    7.22  26.3  0.156   17.6 0.0762       1.57  competing    12
## # ... with 125 more rows
```

and then plot these, facetting on `xname` (or whatever you called it), remembering the `scales="free"` because the x-variables are on different scales:

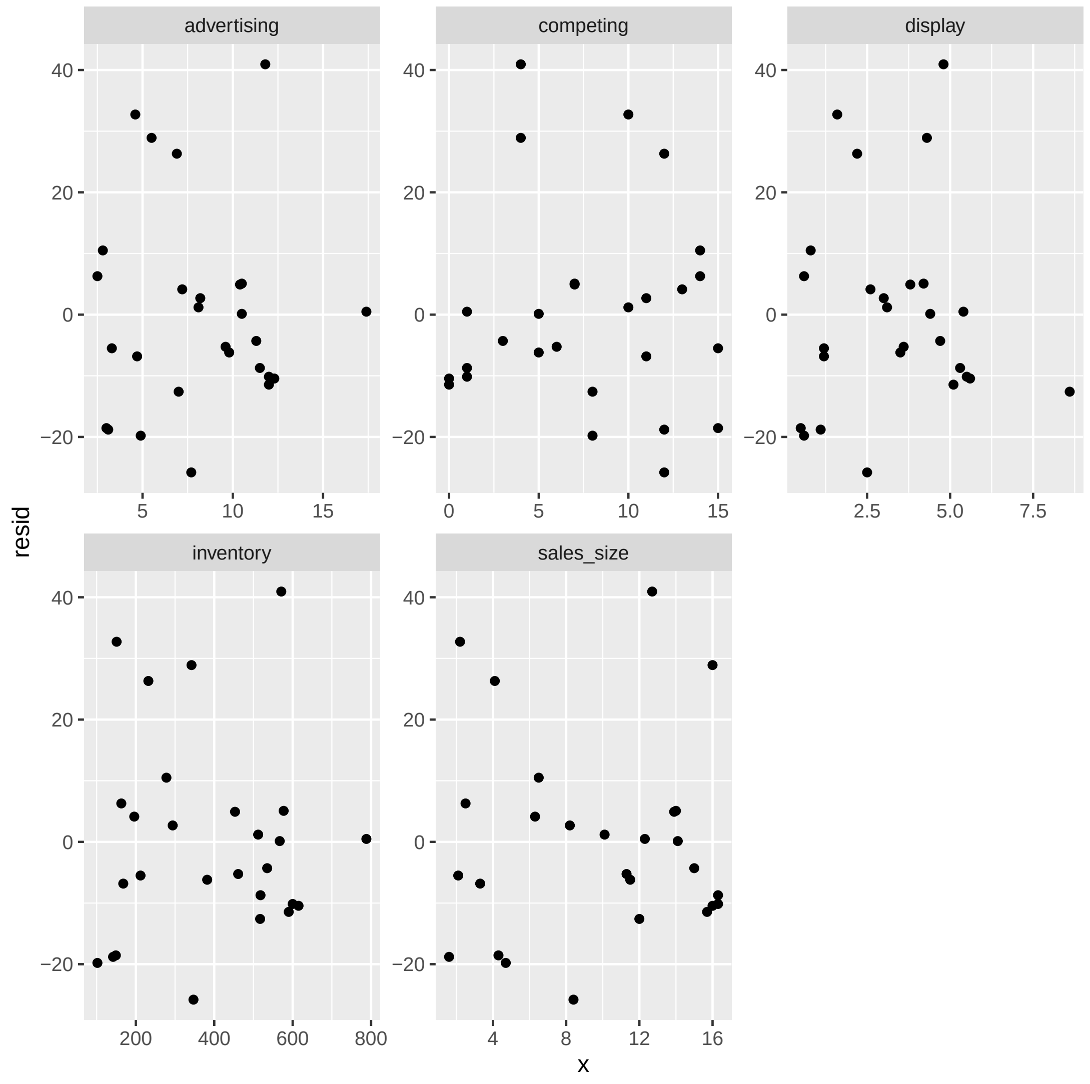
```
allgreen.1 %>% augment(allgreen) %>%
    pivot_longer(display:competing, names_to="xname", values_to="x") %>%
    ggplot(aes(x=x, y=.resid)) + geom_point() +
    facet_wrap(~xname, scales="free")
```



You don't have to do it in steps like I did; the pipeline I had at the end, or something like it, is what I want. Two points for each of: augmenting with the data, pivoting longer, and making the plot.

If you don't remember `augment`, then add the residuals to the original data, something like this. The fitted model is not an actual data frame, so you have to get the residuals out like this:

```
allgreen %>% mutate(resid=resid(allgreen.1)) %>%
    pivot_longer(display:competing, names_to="xname", values_to="x") %>%
    ggplot(aes(x=x, y=resid)) + geom_point() +
    facet_wrap(~xname, scales="free")
```



There are other possibilities for getting hold of the residuals; if you find one that works, you are good.

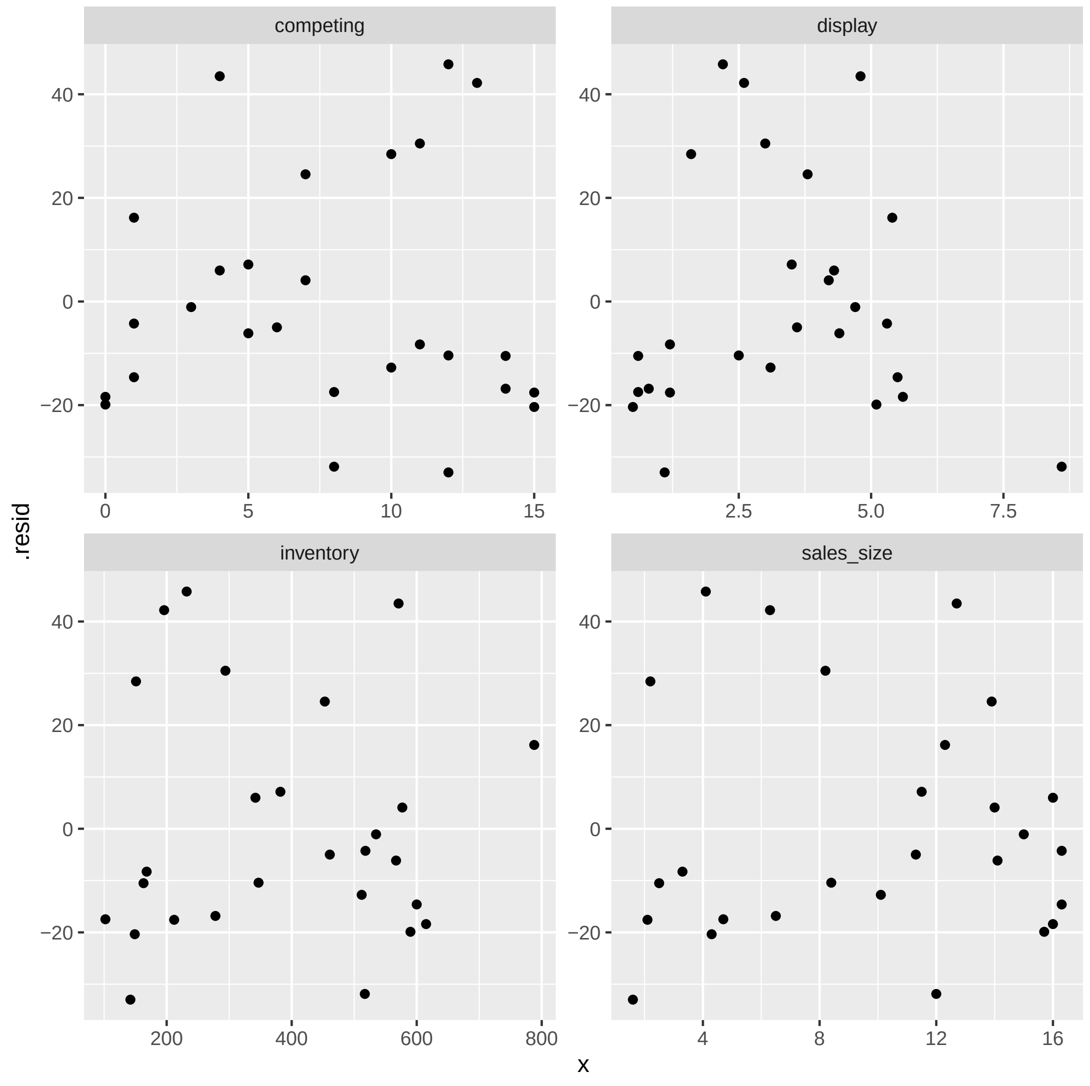
This time the `y` of the `ggplot` is whatever you called that column of residuals.

Again, add a `geom_smooth` if you like; if you do, see the discussion at the end of the final part (in the Extra).

If you get stuck, churn out plots of residuals against xs one by one, making five graphs. Three points for that.

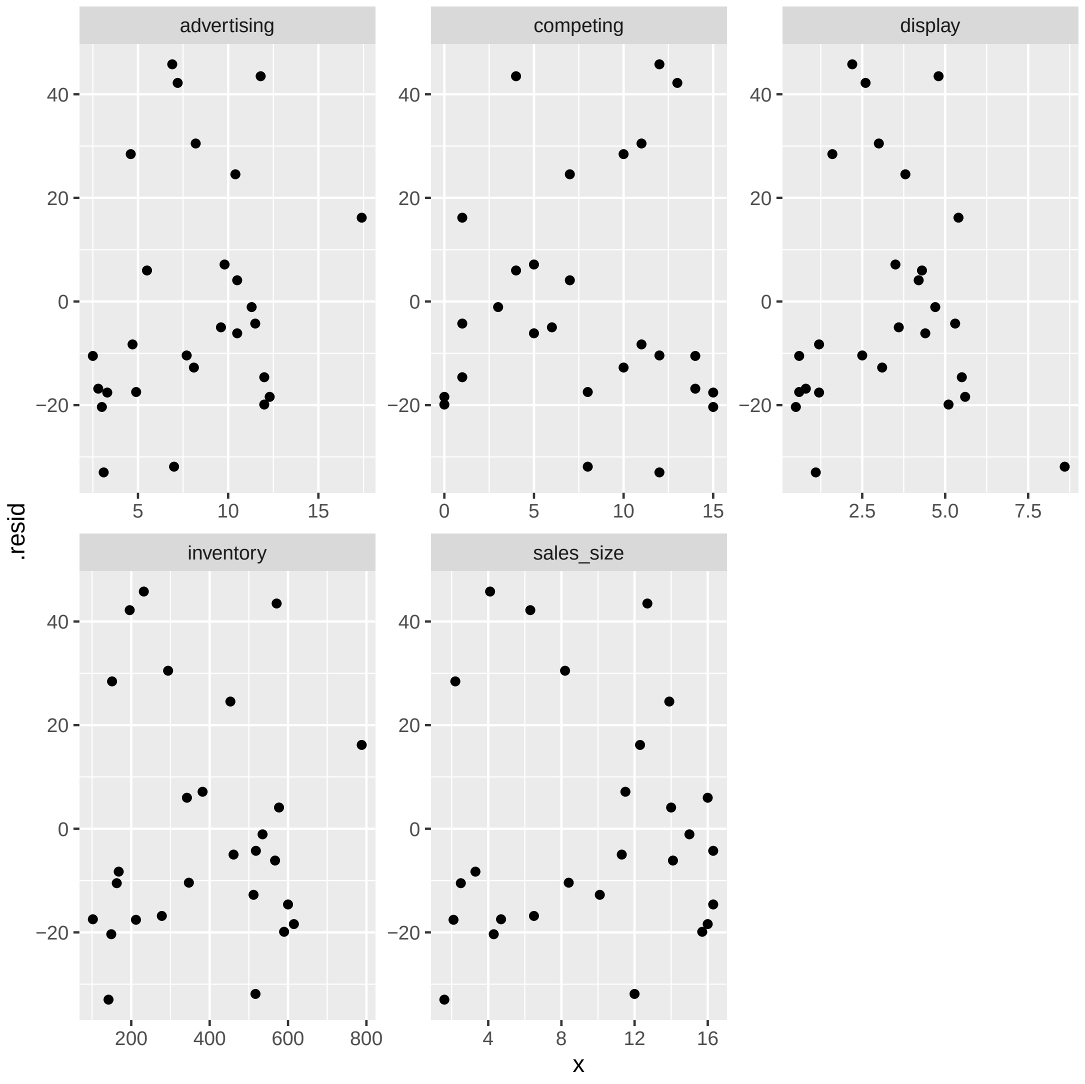
Again, if you left out `advertising`:

```
allgreen.1a %>% augment(allgreen) %>%
    pivot_longer(c(display, inventory, sales_size, competing), names_to="xname", values_to="x"
    ggplot(aes(x=x, y=.resid)) + geom_point() +
    facet_wrap(~xname, scales="free")
```



There is, in all honesty, no harm in including `advertising` here as well, since there might be a relationship with `advertising` that you want to know about. For this, I'm good with any of these:

```
allgreen.1a %>% augment(allgreen) %>%
    pivot_longer(display:competing, names_to="xname", values_to="x") %>%
    ggplot(aes(x=x, y=.resid)) + geom_point() +
    facet_wrap(~xname, scales="free")
```



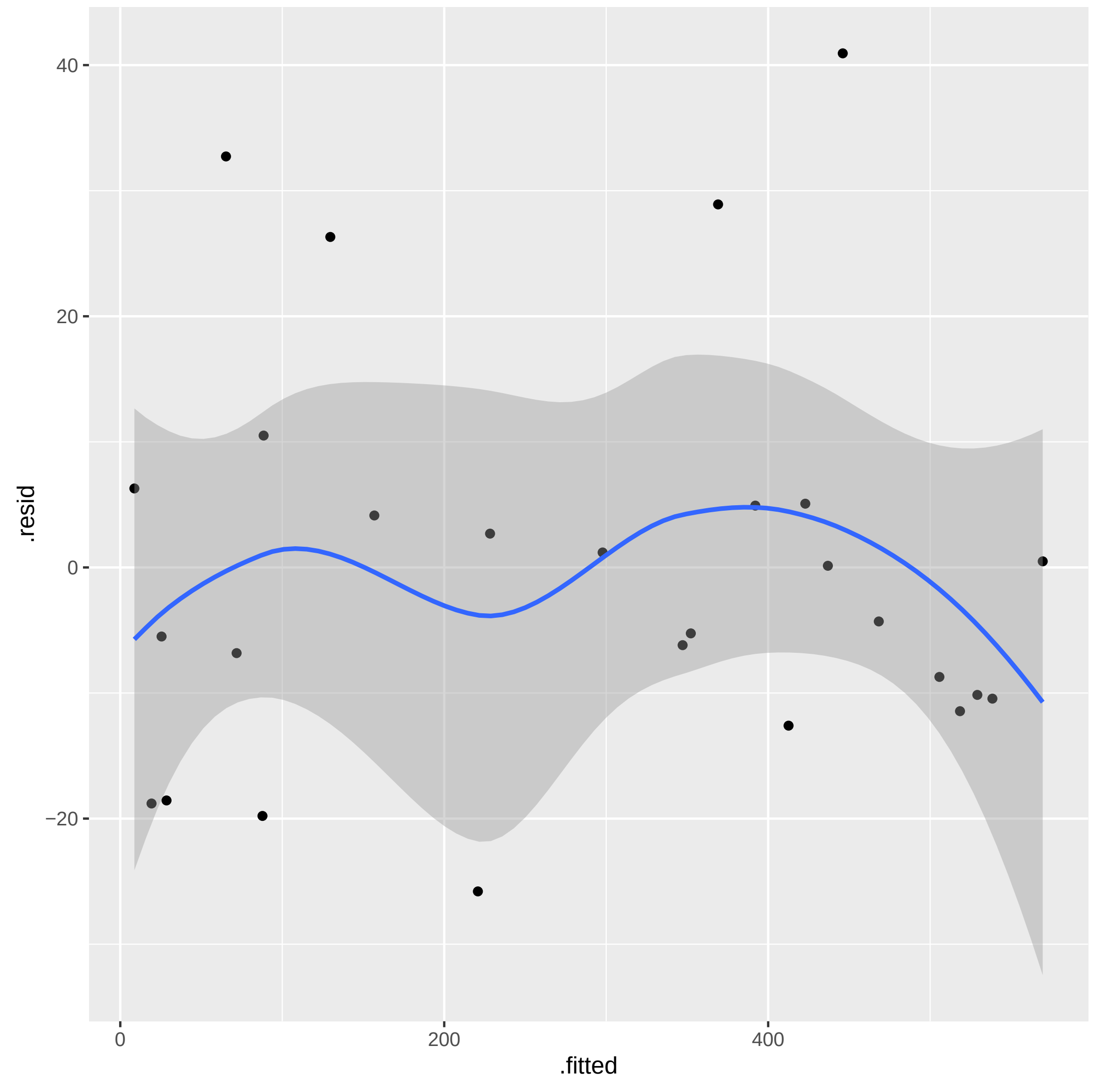
(h) (2 marks) Comment briefly on any problems you see in your plots of the previous part.

Solution: I really don't see any problems with any of these; they all look like random patternless scatters to me. If that's what you see as well, say that. If you see anything else, make the case for it. I think you will have to be rather convincing to make the case for something other than "all good" (but see next paragraph). If you want to talk about outliers, you need to be able to say that the outlying points are a long way from the rest of the data.

If you omitted `advertising`, you might have seen an upward trend with `advertising` with the residuals, which is an indication that you really ought to have included it in the first place, and `display` might have acquired a curved trend.

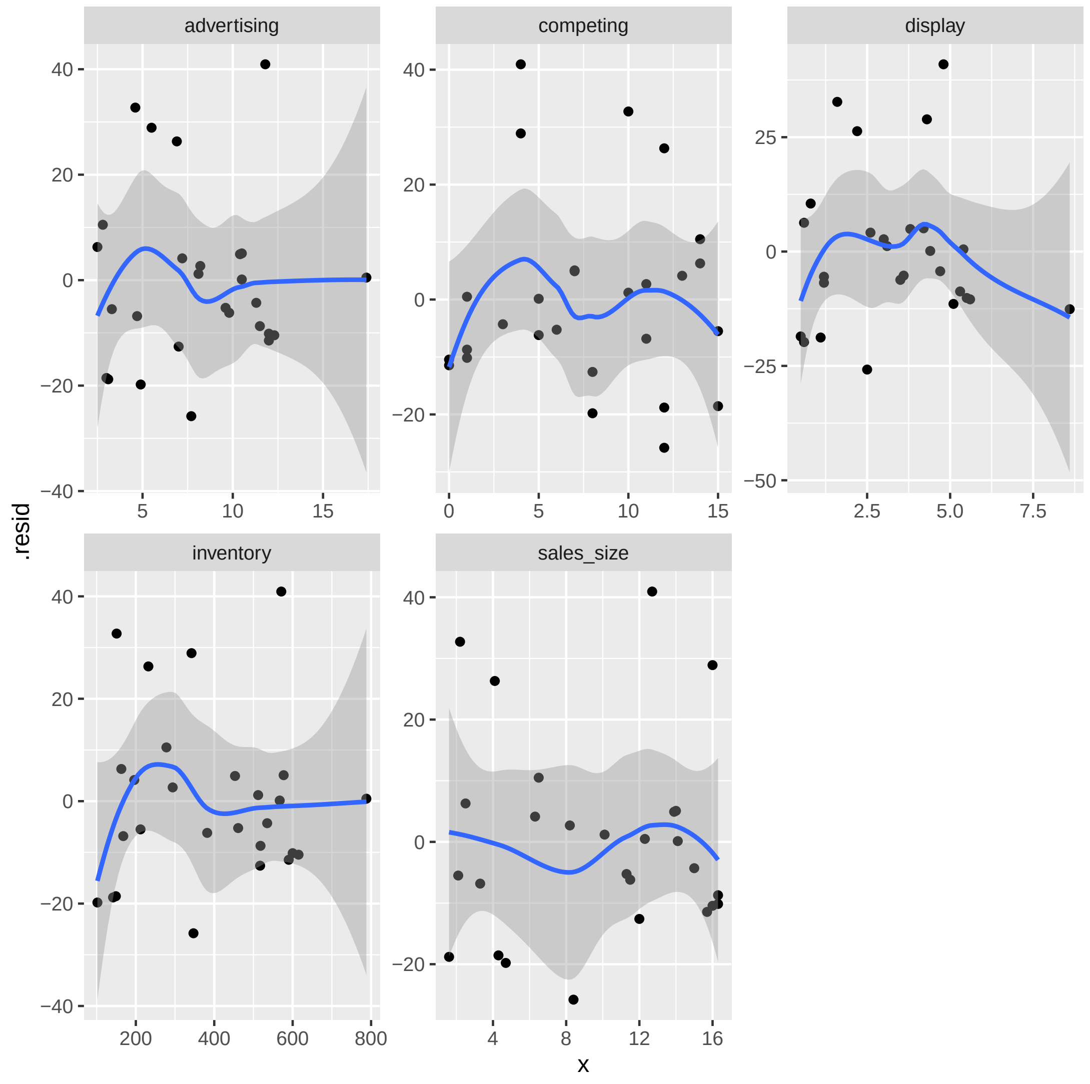
Extra: I was curious what happens if you add a smooth trend to these:

```
ggplot(allgreen.1, aes(x=.fitted, y=.resid)) + geom_point() + geom_smooth()

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



and

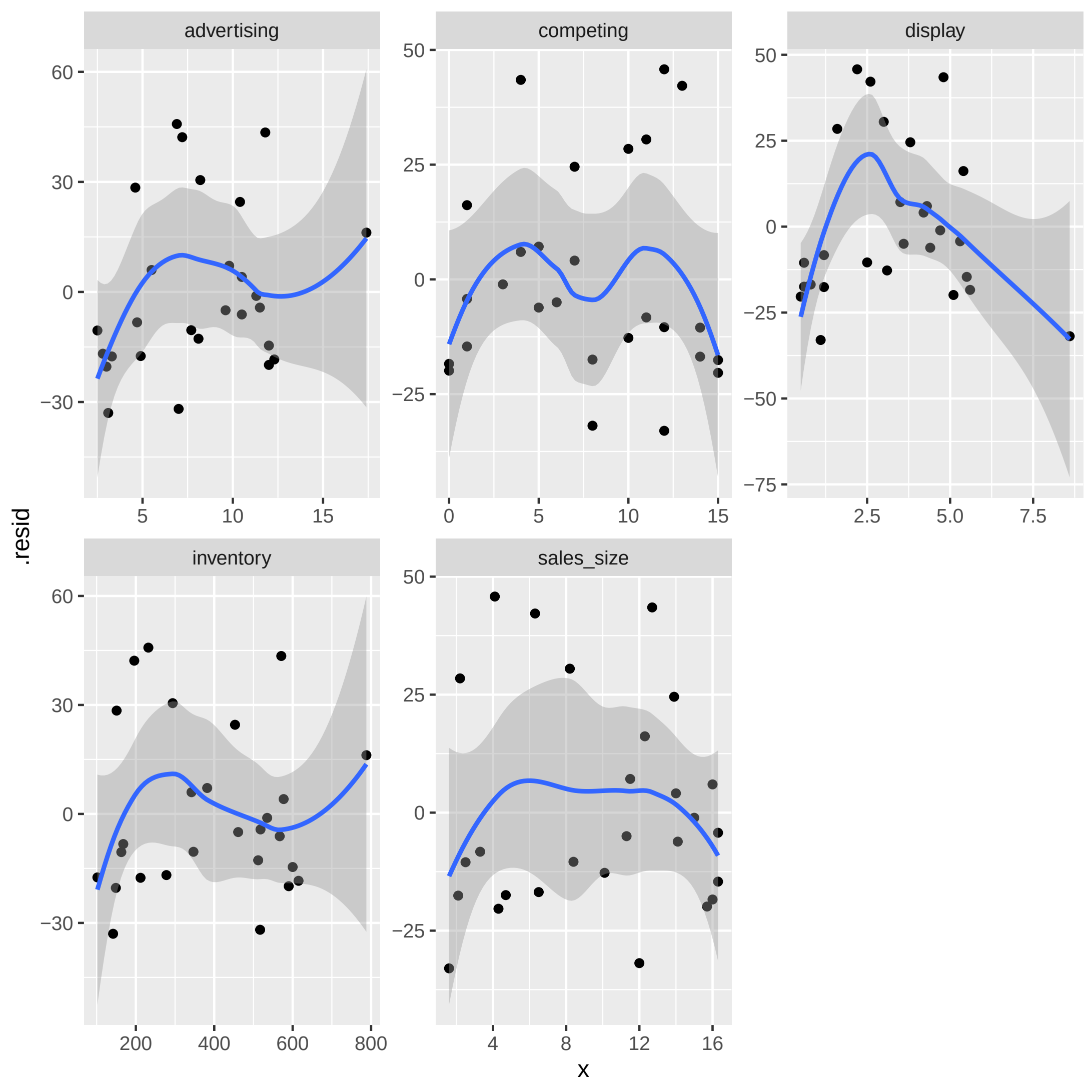
```
allgreen.1 %>% augment(allgreen) %>%
    pivot_longer(display:competing, names_to="xname", values_to="x") %>%
    ggplot(aes(x=x, y=.resid)) + geom_point() + geom_smooth() +
    facet_wrap(~xname, scales="free")

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



To my mind, all of those smooth trends fall firmly in the category of "inconsequential wiggles", for two reasons: (i) 0 is inside the grey envelopes all the way across on all of these, and (ii) the points are a long way from the smooth trends. (i) says it is believable that the mean residual is zero all the way across, and for (ii) if we really had something like a curved trend in any of those, the points would follow that curved trend a lot more closely.

Or:

```
allgreen.1a %>% augment(allgreen) %>%
    pivot_longer(display:competing, names_to="xname", values_to="x") %>%
    ggplot(aes(x=x, y=.resid)) + geom_point() + geom_smooth() +
    facet_wrap(~xname, scales="free")

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



with maybe an increasing trend for `advertising`, and a curved one for `display`.

My error (in not mentioning `advertising`) made this question a lot more awkward to grade.

4. A sample of 100 values is taken on a variable z. The data are at `http://ritsokiguess.site/STAC33/zz.csv`. We are interested in estimating the variance of z with 95% confidence.

   (a) (1 mark) Read in and display (some of) the data.

Solution: As ever:

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

## Parsed with column specification:
## cols(
##   z = col_double()
## )

zz

## # A tibble: 100 x 1
##        z
##    <dbl>
##  1  9.90
##  2  2.17
##  3  6.16
##  4  2.09
##  5  6.26
##  6  1.53
##  7  5.74
##  8  2.92
##  9 16.7
## 10  1.94
## # ... with 90 more rows
```

To avoid confusing yourself, call the data frame something other than `z`.

(b) (2 marks) For normally-distributed populations, $(n-1)s^2/\sigma^2$ has a chi-squared distribution with $n-1$ degrees of freedom, where n is the sample size, s^2 is the sample variance, and σ^2 is the population variance. This means that a 95% confidence interval for σ^2 is given by the following, where a and b are defined below:

$$\left[\frac{(n-1)s^2}{b}, \frac{(n-1)s^2}{a}\right].$$

a and b are defined as follows: let W be a chi-squared random variable with $n-1$ degrees of freedom; then a is such that $P(W \le a) = 0.025$ and b is such that $P(W \le b) = 0.975$. Use `qchisq` to find the values of a and b appropriate for this data set. (You will have to find out how `qchisq` works.)

Solution: `qchisq` is the "inverse cdf" of the chi-squared distribution; that is, it is exactly what we want to find a and b. It has two inputs: the probability whose inverse CDF we want, and the degrees of freedom, here $n-1 = 100-1 = 99$.

Hence

```
a <- qchisq(0.025, 99)
a

## [1] 73.36108

b <- qchisq(0.975, 99)
b

## [1] 128.422
```

(c) (2 marks) Obtain a 95% confidence interval for σ^2 from this data set, using the formula above. Note that `var` finds the variance of a sample.

Solution: The limits are found thus (or by some equivalent means):

```
v <- var(zz$z)
top <- (100-1)*v
top/b

## [1] 9.888232

top/a

## [1] 17.30981
```

This looks backwards, but to find the lower limit, you have to divide by a *bigger* number.

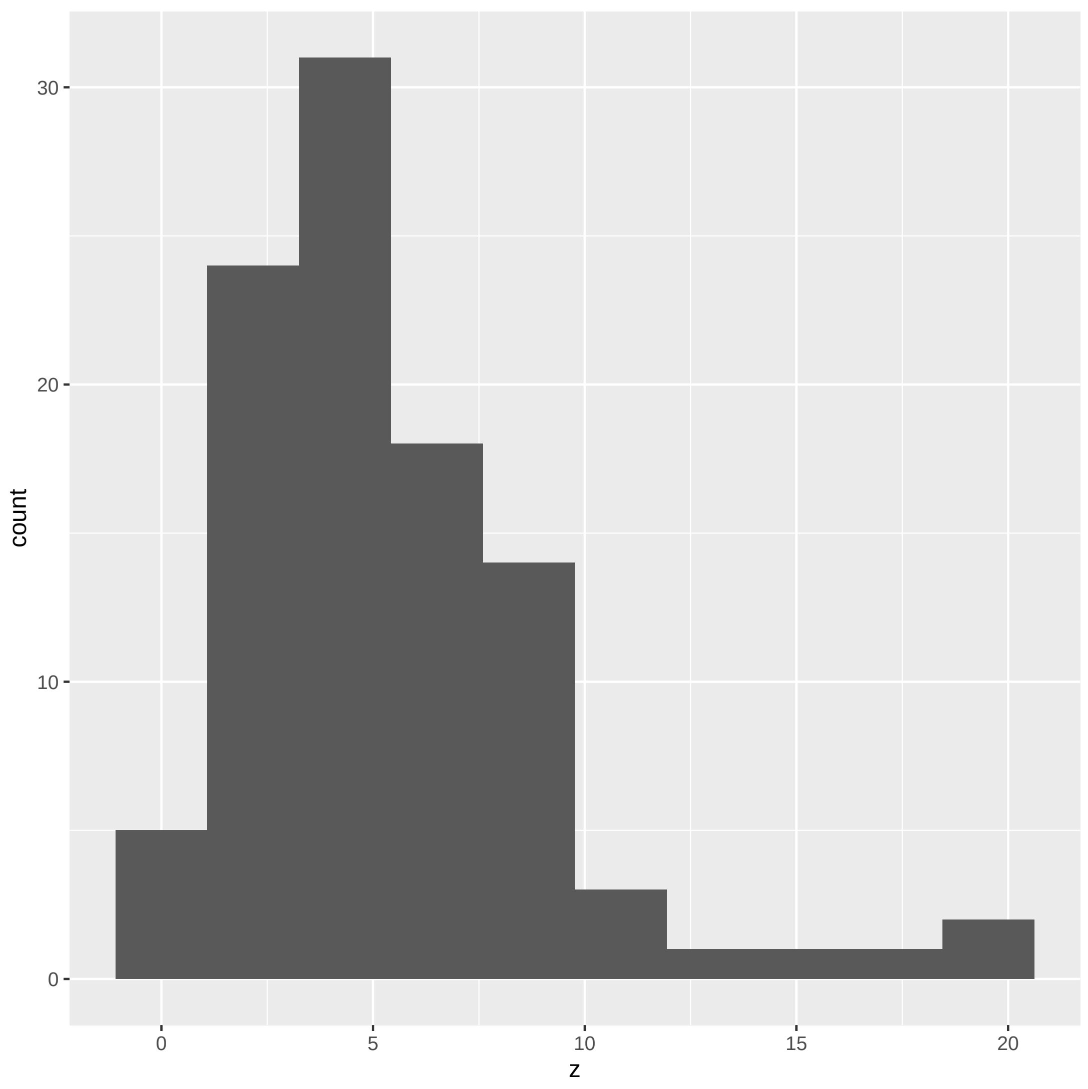
This you most likely saw in B57.

If you couldn't find a and b in the previous part, make up some values to show that you know what to do here. Any values will do.

(d) (2 marks) Obtain a suitable plot of `z`.

Solution: One quantitative variable, so a histogram:

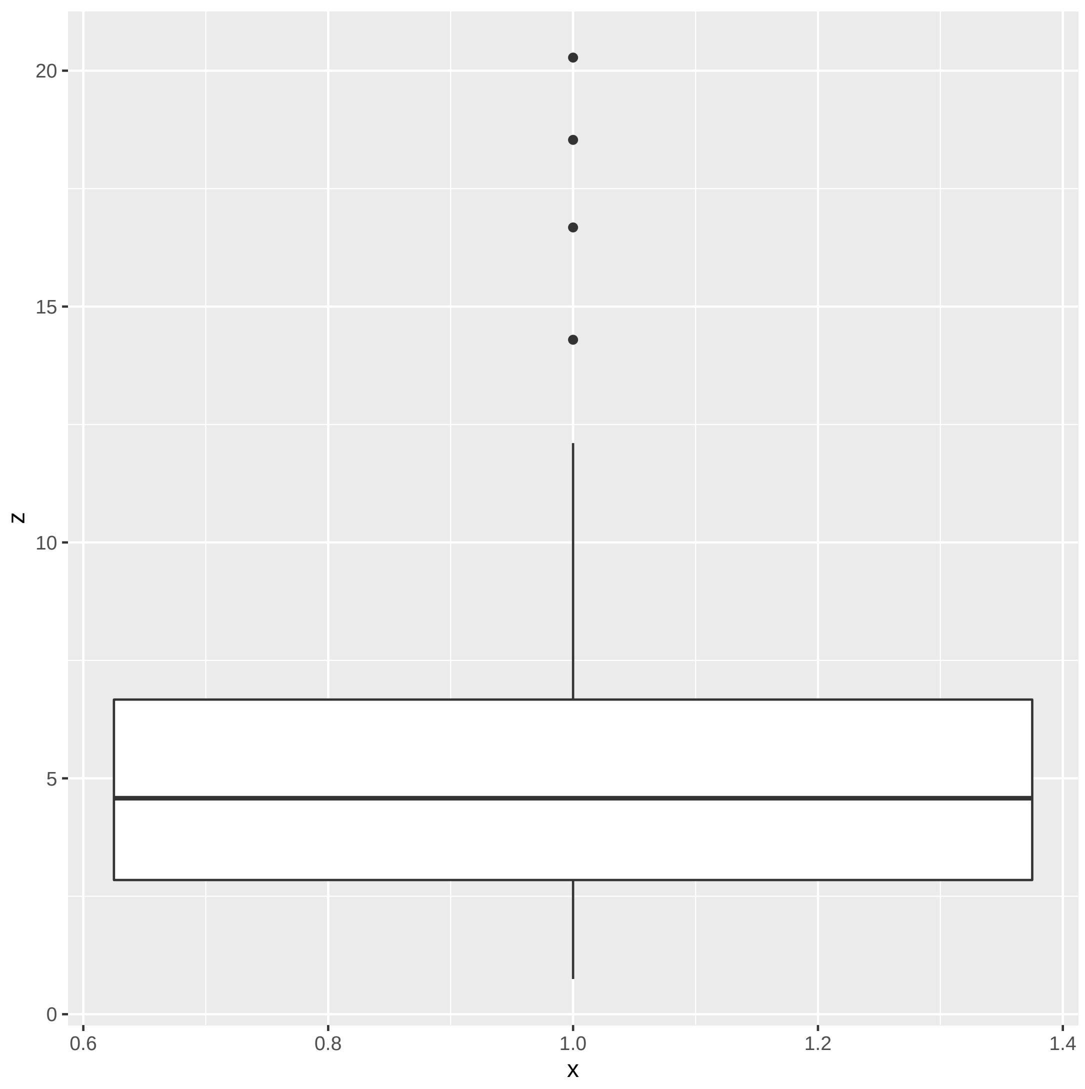
```
ggplot(zz, aes(x=z)) + geom_histogram(bins=10)
```



Pick a number of bins, though I am not especially choosy about how many. My take is that 10 is about as high as you should go. For example, Sturges' rule says 8 bins, since $2^7 = 128$.

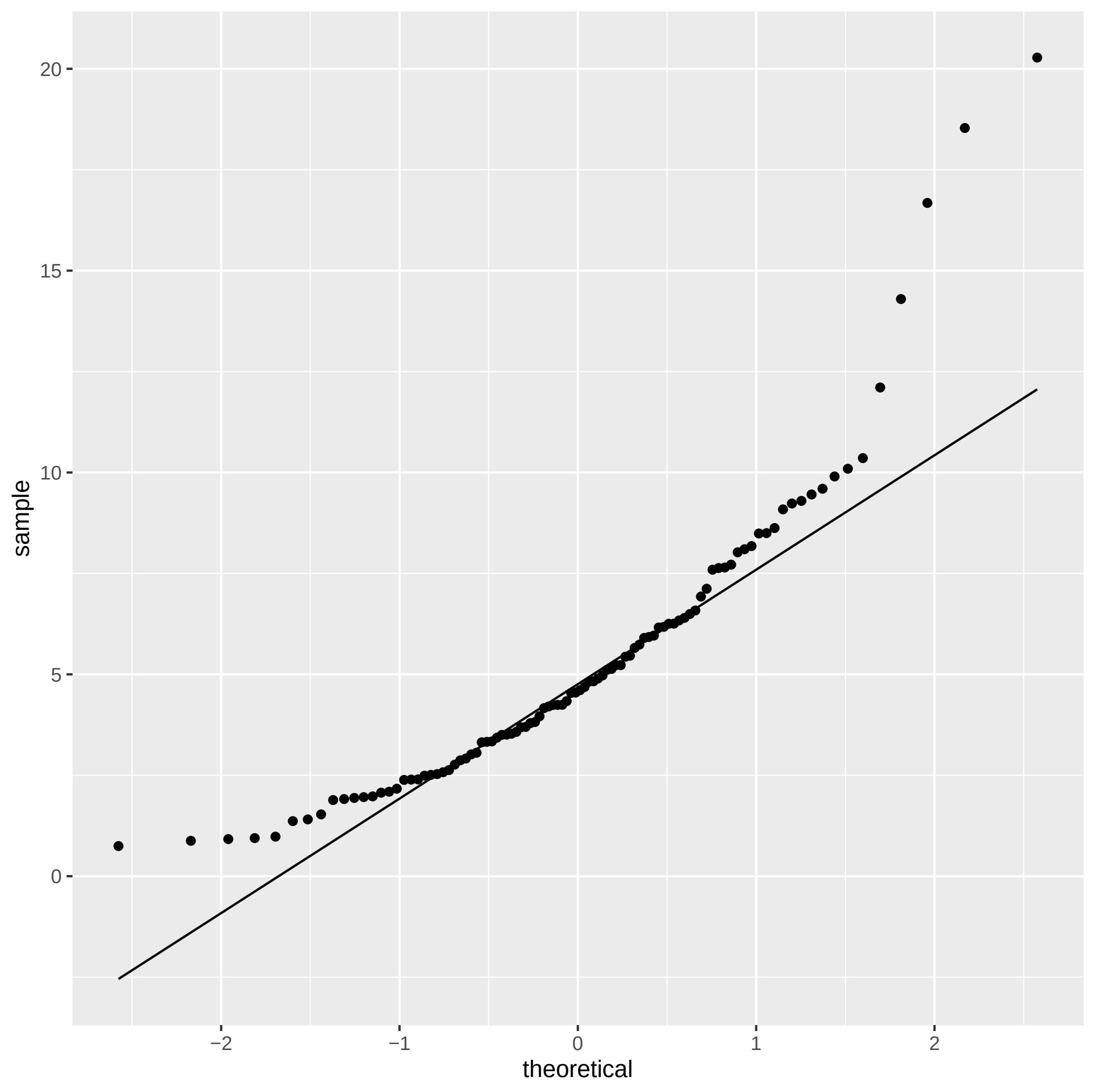
If you insist, a one-group boxplot also works:

```
ggplot(zz, aes(x=1, y=z)) + geom_boxplot()
```



Or, looking ahead, a normal quantile plot would also be the thing:

```
ggplot(zz, aes(sample=z)) + stat_qq() + stat_qq_line()
```



I don't ask for comment here (that's coming up), though if you make the relevant comments here or in the next part, that's OK. In your mind, you probably ought to be thinking "skewed to the right" and wondering how that's going to play into the next part.

(e) (2 marks) Explain briefly why you would have some doubts about the confidence interval you calculated in part (c).

Solution: Two things: the chi-squared-based interval assumes a normal population, and `z` is skewed to the right, not normal.

"The values of `z` are not normal" is only one point. For the second, you have to say *why it matters* that they are not normal.

Extra: a t confidence interval for the *mean* would be perfectly all right from data like this, because the sample size is large enough for the central limit theorem to help, a lot. But there is no central limit theorem for variances, and so we have to insist more firmly on the normality in that case.

(f) (4 marks) Obtain a 95% bootstrap percentile confidence interval for the population variance.

Solution: Giving you no clues here. What you have to do to take a bootstrap sample from this data set is to obtain a sample of size 100 with replacement from the sample, calculate its variance, then repeat "many" times. When you have done that, take the 2.5 and 97.5 percentiles of the results, like this:

```
rerun(1000, sample(zz$z, replace=T)) %>% map_dbl(~var(.)) -> vars
quantile(vars, c(0.025, 0.975))

##      2.5%     97.5%
##  7.112684 19.379134
```

1000 was my "large"; use anything bigger than about 100. I used a different value from my sample size to make it clear that I knew the difference.

(g) (3 marks) Compare your two confidence intervals from parts (c) and (f), and explain briefly why you prefer the one from part (f). (If you could not solve (f), say why you would *expect* it to be better.)

Solution: My answer from (c) was:

```
top/b

## [1] 9.888232

top/a

## [1] 17.30981
```

The answer you get in (f) will depend on the particular random numbers you got, but I think it is likely that the interval will be longer than the one in (c). Describe what you see. One point. (You might find that both limits are higher in (f), or something like that.)

An obvious thing to say about why you prefer (f) is that the data weren't anything like normal, so (c) will be bad. One point.

For the last point, say something about how the bootstrap interval (f) *does not* assume normality, so that the interval in (f) is actually good. This is a stronger conclusion than saying (c) is bad and stopping there.

This is a familiar theme in this course: yes, we would like a shorter confidence interval, all else equal, but it's better to have a longer interval like (f) that we can trust, compared to a shorter one like (c) that we don't trust at all. In these kinds of cases, all else is very much not equal at all.

Extra: I wasn't surprised that the bootstrap interval came out longer. With a right-skewed distribution, it is possible to get a few very large values in the bootstrap sample, and this would make that bootstrap sample have a large variance. So the variability of the sampling distribution of the variance (ouch!) would be larger than you'd expect from a normal population, and that plays out in the bootstrap CI for the variance being longer.

If you were unable to get the bootstrap interval, say why you would *expect* it to be better than the normal-theory one. There are marks here for a sensible discussion of this.