University of Toronto Scarborough
Department of Computer and Mathematical Sciences
STAC32 (K. Butler), Midterm Exam
October 21, 2022

Aids allowed (on paper, no computers):

- My lecture overheads (slides)
- Any notes that you have taken in this course
- Your marked assignments
- My assignment solutions
- Non-programmable, non-communicating calculator

This exam has 8 numbered pages of questions plus this cover page.
In addition, you have an additional booklet of Figures to refer to during the exam.
The maximum marks available for each part of each question are shown next to the question part.
If you need more space, use the last page of the exam. Anything written on the back of the page will not be graded.
You may assume throughout this exam that the code shown in Figure 1 of the booklet of Figures has already been run.

1. The questions here are all about reading in data from files.

   (a) [3] A data file is laid out as shown in Figure 2 (in the booklet of Figures). This file is stored in a file `biscuit1.txt` in the same folder as your R Studio project. What code will read these data values into a dataframe called `test1` and display that dataframe?

My answer:

This is aligned columns, so `read_table` will do it:

```
test1 <- read_table("biscuit1.txt")
```

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

```
test1
```

```
## # A tibble: 6 x 2
##   treatment height
##   <chr>      <dbl>
## 1 low         10.1
## 2 low          9.7
## 3 medium       4.8
## 4 medium      11.3
## 5 high        10.3
## 6 high         9.8
```

Two for the first line of code, one for the second. Don't forget to display what you read in!

Three points is a bit generous, but it's the first part of the first question, so all being well you should be able to get off to a fast start. Minus one per error. You might help yourself by using the words "aligned columns" in your answer, even if something else goes wrong.

   (b) [3] A second data file is shown in Figure 3. This file is stored in `biscuit2.txt` in the same folder as your R Studio project. Explain briefly why the code below will read in the data from the file. (You may treat it as given that the data read in will be stored in a dataframe called `test2`.)

```
test2 <- read_delim("biscuit2.txt", " ")
```

My answer:

Points to make:

- the data values are separated by *exactly* one space, so `read_delim` is appropriate. Two points. ("The data values are separated by spaces" is not precise enough; this does not rule out the situation of (a), where `read_delim` will definitely *not* work. One point for this kind of thing.)

- the second input indicates what the data values are separated by. One point.

To illustrate that it does in fact work:

```
test2 <- read_delim("biscuit2.txt", " ")
```

```
## Rows: 6 Columns: 2
## -- Column specification ---------------------------------------------------------
## Delimiter: " "
## chr (1): treatment
## dbl (1): length
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
test2
```

```
## # A tibble: 6 x 2
##   treatment length
##   <chr>      <dbl>
## 1 low          6.1
## 2 low          6.5
## 3 medium       8.7
## 4 medium      10.3
## 5 high         3.1
## 6 high        13.8
```

(c) [2] Explain briefly why the code shown in Figure 4 also works.

My answer:

If you don't say what single character separates the data values, R will try to guess. In this case, it guessed successfully, deducing that the data values were separated by single spaces. You know it did this because of the `Delimiter: " "` in the output of `read_delim` in Figure 4. If it had inferred the character separating the data values to be, say, a semicolon, you would have seen in the same place `Delimiter: ";"` instead.

One point for saying that `read_delim` guessed or inferred the right separator from what was in the data, and the second for saying how you know that it guessed the right thing.

(d) [4] A collaborator gives you a spreadsheet, stored in `precious.xlsx`, as shown in Figure 5, containing data that you would like to read into R. The spreadsheet is a "workbook" containing two sheets, `Sheet1` containing some notes, and `Sheet2` containing the data you want. You have saved the file in the same folder as your current R Studio project.

What are *two* distinct ways you might read the data in? Describe what you would do, or give code for doing it, as appropriate.

My answer:

The easiest approach is to save the sheet `Sheet2` with the data in it as a CSV (one point), and read it in with `read_csv` (one point):

```
precious <- read_csv("precious.csv")
```

```
## Rows: 3 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (2): x, y
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
precious
```

```
## # A tibble: 3 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     2
## 2     3     4
## 3     5     6
```

making sure to save your read-in dataframe somewhere.

I only looked at the result to show that it worked. This is the recommended way to share datafiles: save the spreadsheet as a CSV and then share that. (Hence, when you have been doing assignments, the datafiles have mostly been CSVs.)

The other way is to read the sheet in directly with `read_excel`. This lives in package `readxl`, which is one of the packages loaded in Figure 1, so you don't need to repeat the `library` line from there:

```
precious2 <- read_excel("precious.xlsx", sheet = "Sheet2")
precious2
```

```
## # A tibble: 3 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     2
## 2     3     4
## 3     5     6
```

Two points for this. Only one if you forget or mess up the `sheet` piece.

Extra: you might have been wondering where the names `biscuit1` and `biscuit2` came from. The exceedingly tortuous route by which that came about is this:

- this file is all about reading files.
- the town where I grew up (in England) is called Reading (but pronounced like “redding”).
- the soccer team there is now known as the Royals (because of Windsor Castle being in the same county), but they used to be called the Biscuitmen, because there was a biscuit (cookie) factory in the town, up until recently.

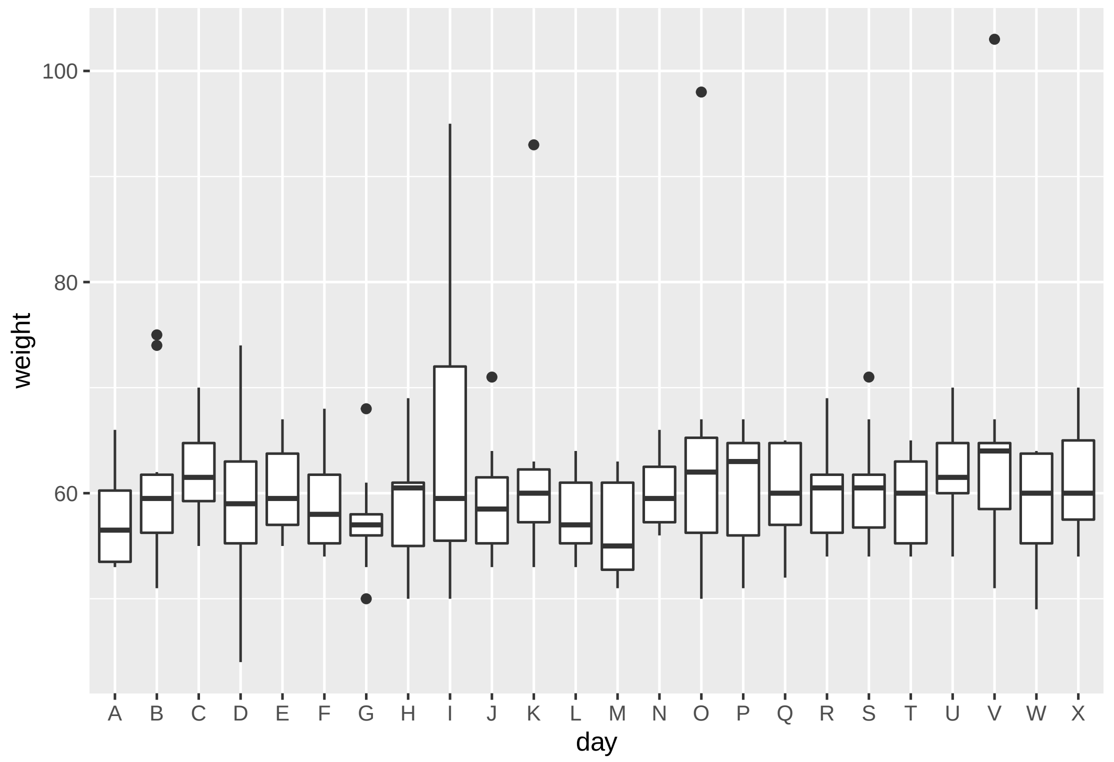
2. A chicken farmer keeps track of the weights of eggs (in grams) produced by their chickens for each of 24 days (labelled days A through X in the data). On each day, 10 eggs were randomly selected and weighed. Some of the data are shown in Figure 6.

   (a) [2] A graph is shown in Figure 7. What code was used to draw this graph?

My answer:

This is a boxplot (though admittedly one with rather a lot of groups). The categorical variable `day` is the `x`, and the quantitative variable `weight` is the `y`, hence:

```
ggplot(eggs, aes(x = day, y = weight)) + geom_boxplot()
```



If you are not comfortable with doing it this way around, think about what plot *you* would draw with this dataframe, and then decide whether that would (or would not) come out like this. (If it wouldn't, think about how you would fix it so it does.)

   (b) [4] Considering Figure 7, what code would compute a suitable measure of centre and spread of egg weight for each day? Justify your choices briefly.

My answer:

The boxplots indicate some outliers (mostly at the upper end), which could distort the mean and SD, so we should use the median and inter-quartile range. One point for a properly justified preference of median/IQR over mean/SD.

The code is a group-by (days) and summarize (whatever two summaries you chose). You can call the summaries whatever you like, but you need to call the functions `median` and `IQR` to find the summaries:

```
eggs %>%
  group_by(day) %>%
  summarize(med_weight = median(weight), iqr_weight = IQR(weight))
```

```
## # A tibble: 24 x 3
##    day   med_weight iqr_weight
##    <chr>      <dbl>      <dbl>
##  1 A           56.5       6.75
##  2 B           59.5       5.5
##  3 C           61.5       5.5
##  4 D           59         7.75
##  5 E           59.5       6.75
##  6 F           58         6.5
##  7 G           57         2
##  8 H           60.5       6
##  9 I           59.5      16.5
## 10 J           58.5       6.25
## # ... with 14 more rows
```

One point for the group-by, and one each for a summary of centre and a summary of spread consistent with your justification. (That means that if you decide that mean and SD are the way to go, you won't get the first point, but you'll get all the others.)

This is a longer summary than we have had before, because there are more groups (days), but it works the same.

(c) [2] The column `day_number` contains the same information as `day`, except that the days are now numbered rather than indicated by letters. Would the plot of Figure 7 still work using `day_number`, instead of `day`? Explain briefly, and, if necessary, indicate how you would change your code of (a) to make the plot work using `day_number`.

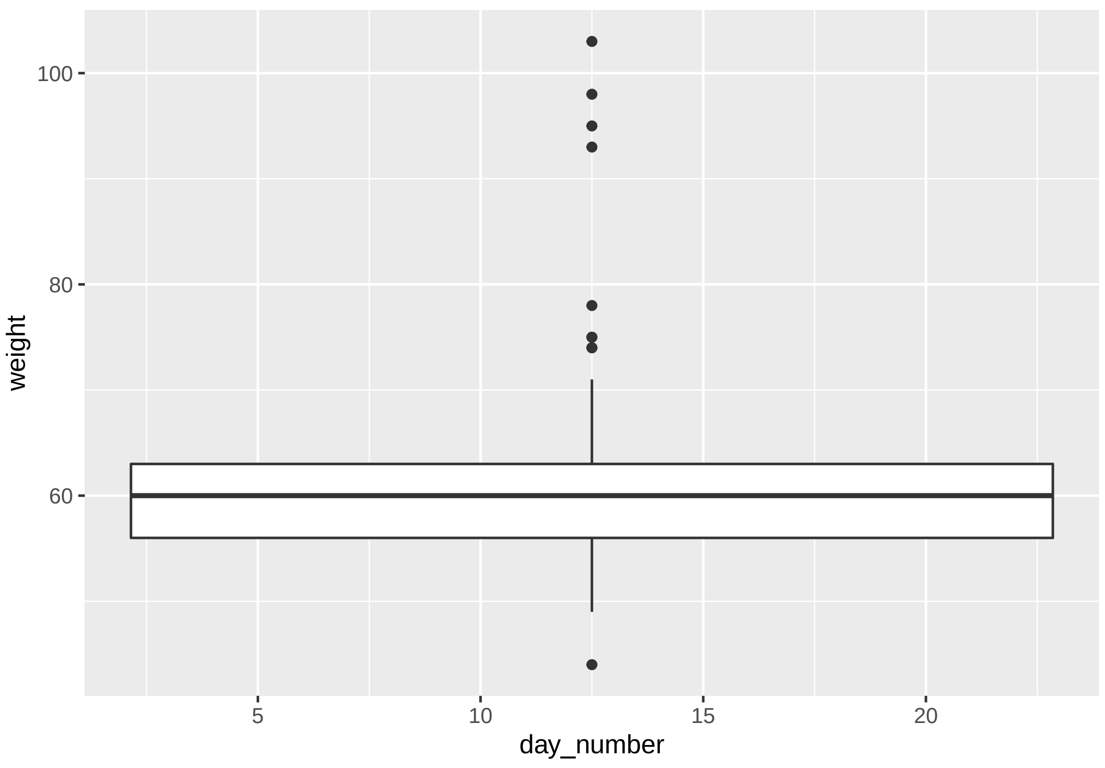
My answer:

No, it would not work, because the boxplot needs to have the groups indicated by something categorical, and by using `day_number` it will see this variable as quantitative (it looks like a number). One point.

This is what will happen:

```
ggplot(eggs, aes(x = day_number, y = weight)) + geom_boxplot()
```

```
## Warning: Continuous x aesthetic -- did you forget aes(group=...)?
```

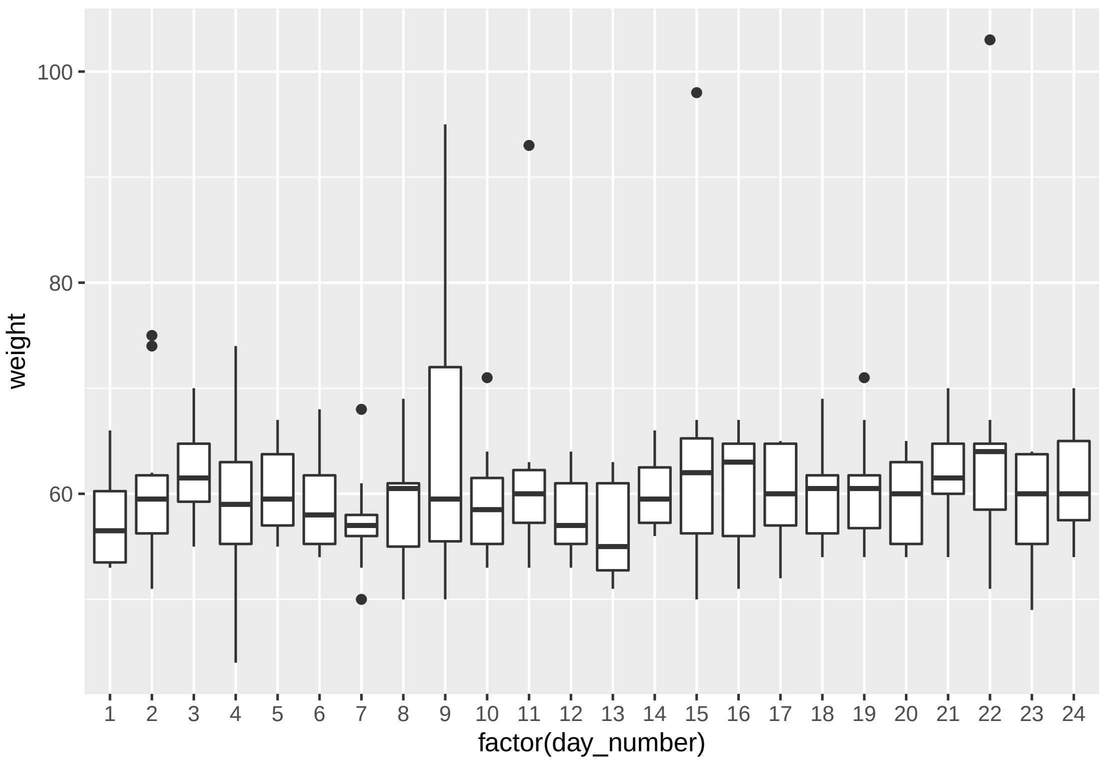


I don't expect you to know what will happen, but I *do* expect you to know that it will not work and why.

If you care, now that you are reading this: `ggplot` simply ignores the `x` if it doesn't think it is categorical, so that it is not actually an error at all (you get a graph of some kind, which is actually a boxplot of all the weights regardless of day), but you *do* get a warning that this is probably not what you wanted.

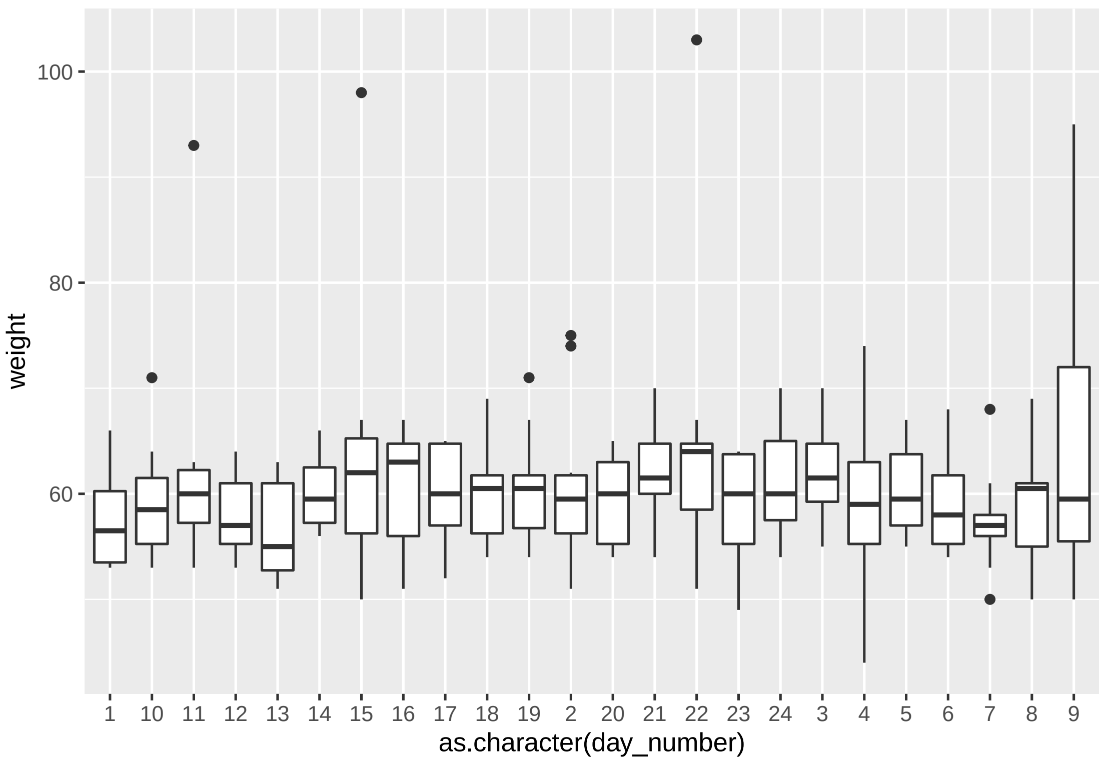
To fix it, turn `day_number` into something categorical. Either make it a `factor` (easiest):

```
ggplot(eggs, aes(x = factor(day_number), y = weight)) + geom_boxplot()
```



or make it text, with the same result:

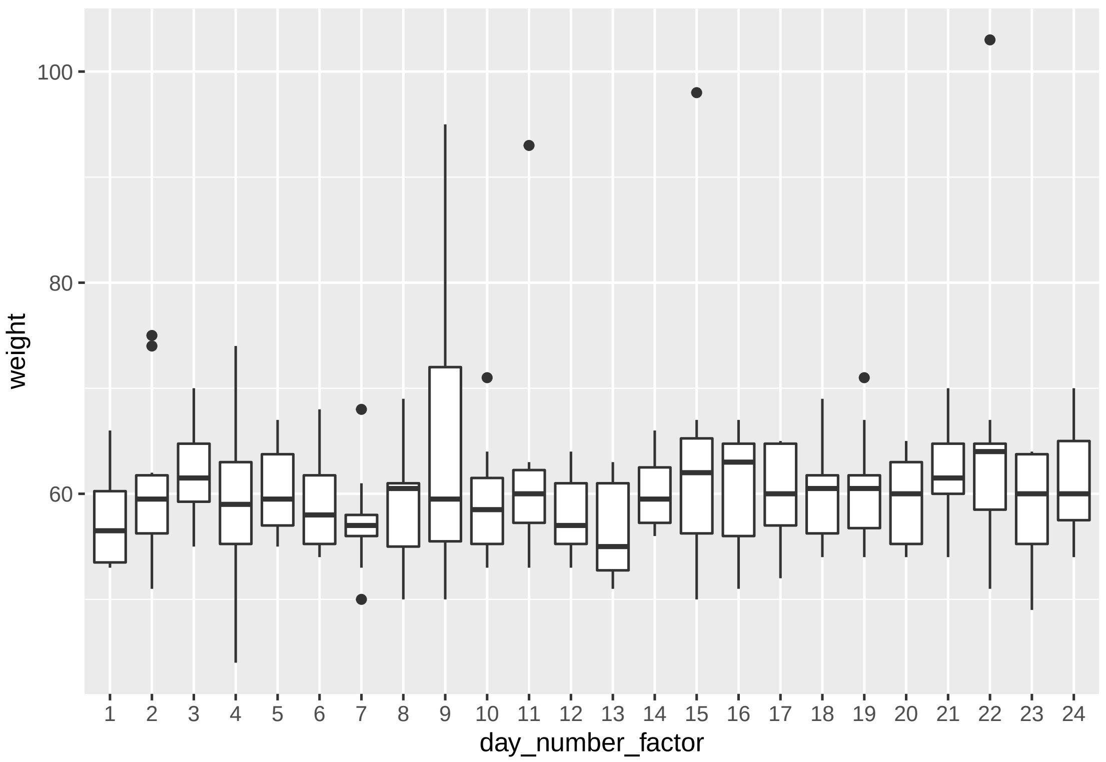
```
ggplot(eggs, aes(x = as.character(day_number), y = weight)) + geom_boxplot()
```



The other point for modifying `day_number` in an appropriate way. The most concise way to say it is "replace `day_number` by `factor(day_number)`".

It is also good to create a new column in your dataframe first, and then plot that. You might even argue that it is better, because then the axis label will look nicer:

```
eggs %>%
  mutate(day_number_factor = factor(day_number)) %>%
  ggplot(aes(x = day_number_factor, y = weight)) + geom_boxplot()
```



I asked for your changes from (a), so you don't have to write out the whole thing. Maybe write out the first line of code, then say "use `day_number_factor` in the graph instead of `day_number`."

Extra: data source is `goulden.eggs` from the `agridat` package:

```
library(agridat)
data("goulden.eggs")
head(goulden.eggs)
```

```
##   day weight
## 1   1     55
## 2   1     53
## 3   1     56
## 4   1     63
## 5   1     66
## 6   1     58
```

I made the day a genuine categorical variable, so that you wouldn't get confused, looking at the plot, and kept the original day number to use in the final part of the question:

```
goulden.eggs %>% mutate(day_number = day,
                        day = LETTERS[day]) -> eggs
head(eggs)

##   day weight day_number
## 1   A     55          1
## 2   A     53          1
## 3   A     56          1
## 4   A     63          1
## 5   A     66          1
## 6   A     58          1
```

3. Many cities require certain pet animals, such as cats and dogs, to be licensed. Seattle is one of those cities. The dataframe `seattlepets`, some of which is shown in Figure 8, contains information about all the pets licensed in Seattle over a certain time period, including the licence number, the date it was issued, what kind of animal it is (in `species`), what breed it is within `species` (`primary_breed`), the animal's name, and so on.

   In the questions below, what code would display the appropriate part of the dataframe, or summarize it in an appropriate way?

   (a) [3] Display only the animal names and kind of animal they are, and display the first ten rows of those.

   My answer:

   A `select` to display the columns, and a `slice` to display the rows, in either order:

```
seattlepets %>% select(animal_name, species) %>%
  slice(1:10)
```

```
## # A tibble: 10 x 2
##    animal_name species
##    <chr>       <chr>
##  1 Wall-E      Dog
##  2 Andre       Dog
##  3 Mac         Dog
##  4 Melb        Cat
##  5 Gingersnap  Cat
##  6 Cody        Dog
##  7 Millie      Dog
##  8 Sebastian   Cat
##  9 Madeline    Cat
## 10 Cleo        Cat
```

   (b) [2] How many animals are there of each species?

   My answer:

```
seattlepets %>% count(species)
```

```
## # A tibble: 4 x 2
##   species     n
##   <chr>   <int>
## 1 Cat     17294
## 2 Dog     35181
## 3 Goat       38
## 4 Pig         6
```

   Alternatively:

```
seattlepets %>% group_by(species) %>%
  summarize(species_count = n())
```

```
## # A tibble: 4 x 2
##   species species_count
##   <chr>           <int>
## 1 Cat             17294
## 2 Dog             35181
## 3 Goat               38
## 4 Pig                 6
```

using whatever name you like for the counts.

This was meant to be an easy one.

Fish, birds, and some other pets don't need to be registered, so they do not appear in this dataset.

(c) [2] Display only the pet pigs (you can display all the variables).

My answer:

Choosing rows, so `filter`:

```
seattlepets %>%
  filter(species == "Pig")
```

```
## # A tibble: 6 x 7
##   license_issue_date license_number animal_name species primary_breed
##   <date>             <chr>          <chr>       <chr>   <chr>
## 1 2018-07-27         731834         Millie      Pig     Pot-Bellied
## 2 2018-08-29         S146306        Othello     Pig     Pot-Bellied
## 3 2018-04-23         S116433        Atticus     Pig     Pot-Bellied
## 4 2018-04-10         139975         Darla       Pig     Pot Bellied
## 5 2018-08-29         S146305        Coconut     Pig     Pot-Bellied
## 6 2018-05-12         S141788        <NA>        Pig     Standard
## # ... with 2 more variables: secondary_breed <chr>, zip_code <chr>
```

The species have an initial Capital Letter, but I'm not expecting you to know that for pigs, so an initial lowercase p is fine also.

There are, as you might guess, not very many pet pigs.

(d) [3] For dogs (only), find the three most common names.

My answer:

Grab the dogs only, make a table of counts of names, and then find the three biggest frequencies (in the column `n` after you have `count`ed them).

```
seattlepets %>% filter(species == "Dog") %>%
  count(animal_name) %>%
  slice_max(n, n = 3)
```

```
## # A tibble: 3 x 2
##   animal_name     n
##   <chr>       <int>
## 1 Lucy          337
## 2 Charlie       306
## 3 Bella         249
```

or

```
seattlepets %>% filter(species == "Dog") %>%
  count(animal_name) %>%
  arrange(desc(n)) %>%
  slice(1:3)
```

```
## # A tibble: 3 x 2
##   animal_name     n
##   <chr>       <int>
## 1 Lucy          337
## 2 Charlie       306
## 3 Bella         249
```

Either way is good. The `filter` has to come first, because you don't want to count the names of animals other than dogs here.

One point each for the `filter`, the `count`, and finding the top three.

(e) [4] For cats (only), find the percentage of all cats that have a name that is included in the ten most popular cat names.

My answer:

There are probably many different ways you could tackle this. If it looks as if your code does the right thing, you will be good. It may help if you explain what you think you are doing as you go. (There may be partial credit if your intention is more accurate than your code.)

Points: one for finding the total number of cats, two for finding the top 10 most popular cat names, one for totalling those up and finding a percentage (or at least a proportion).

I did it this way. First find all cat names and their frequencies and save:

```
seattlepets %>% filter(species == "Cat") -> cats
cats %>%
  count(animal_name) -> cat_counts
cat_counts
```

```
## # A tibble: 7,027 x 2
##    animal_name         n
##    <chr>           <int>
##  1 "__"                1
##  2 "-"                 1
##  3 "'Alani"            1
##  4 "\"Mama\" Maya"     1
##  5 "\"Mo\""            1
##  6 "1"                 1
##  7 "2"                 1
##  8 "30 Weight"         1
##  9 "99"                1
## 10 "A.C."              1
## # ... with 7,017 more rows
```

I'm saving this dataframe because I am going to use it (at least) twice.

Part one: find the total number of cats. To do this, sum up the frequencies in `cat_counts`:

```
cat_counts %>% summarize(total_cats = sum(n)) %>%
  pull(total_cats) -> total
total
```

```
## [1] 17294
```

or count the number of rows in `cats`, which you can do like this (there are other ways):

```
cats %>% summarize(n_row = n())
```

```
## # A tibble: 1 x 1
##   n_row
##   <int>
## 1 17294
```

with the same result.

I'm going to use the total number of cats below, so I `pull`ed it out and saved it in a variable. If you prefer, stop after the `summarize` and (in your imagination) copy and paste the result below. "Call this result X" is a way of saying that you intend to use this number later.

Part two:

- find the top 10 cat names in the same way that we found the top three dog names,
- then total up their frequencies,
- then figure out what percent that is of the total.

You can `arrange` and then `slice` here as well:

```
cat_counts %>% slice_max(n, n = 10) %>%
  summarize(top_ten_percent = sum(n) / total * 100)
```

```
## # A tibble: 1 x 1
##   top_ten_percent
##             <dbl>
## 1            6.64
```

My `total` might be your X from earlier.

The most popular cat name is actually "missing"! (Or, people didn't enter the name on the form. I don't think Seattle has that many nameless cats.)

The point is that this, like any `tidyverse` output, is itself a dataframe, so if you want to do anything else with it, you do the exact same thing you would do with any dataframe. In our case, we want to know how many cats have names in the top 10 (ie. somewhere in the list of names in `animal_name`), and the number of those is the total of the values in column `n`. Hence

```
cat_counts %>% slice_max(n, n = 10) %>%
  summarize(top_ten_total = sum(n))
```

```
## # A tibble: 1 x 1
##   top_ten_total
##           <int>
## 1          1148
```

and then divide that by the total number of cats, which we found before, and turn into a percentage:

```
cat_counts %>% slice_max(n, n = 10) %>%
  summarize(top_ten_percent = sum(n) / total * 100)
```

```
## # A tibble: 1 x 1
##   top_ten_percent
##             <dbl>
## 1            6.64
```

I surreptitiously changed the name of the summary just above so that it actually described what the summary was.

Whichever way, the answer turns out to be 6.6%. There are a lot of different names people have for their cats, so the 10 most popular names are not a very big fraction of the total.

Another way: first find the top 10 cat names, as before, but then grab their names:

```
cat_counts %>% slice_max(n, n = 10) %>%
  pull(animal_name) -> top_ten_names
top_ten_names
```

```
##  [1] NA        "Luna"    "Lucy"    "Lily"    "Max"     "Bella"   "Charlie"
##  [8] "Oliver"  "Jack"    "Sophie"
```

Then go through the entire dataframe of cats and see whether the cat's name matches one of these, which you can do this way:

```
cats %>%
  filter(animal_name %in% top_ten_names)
```

```
## # A tibble: 1,148 x 7
##    license_issue_date license_number animal_name species primary_breed
##    <date>             <chr>          <chr>       <chr>   <chr>
##  1 2018-11-24         896415         <NA>        Cat     Domestic Medium Hair
##  2 2018-10-19         586531         Sophie      Cat     Siamese
##  3 2018-12-23         8004089        Oliver      Cat     American Shorthair
##  4 2018-10-29         732106         Lily        Cat     Domestic Shorthair
##  5 2018-11-25         895808         Max         Cat     Domestic Shorthair
##  6 2018-11-06         S125292        Jack        Cat     Domestic Shorthair
##  7 2018-12-09         8003651        Oliver      Cat     Domestic Shorthair
##  8 2018-12-03         S151150        Bella       Cat     Domestic Shorthair
##  9 2018-11-06         S124623        Lucy        Cat     American Shorthair
## 10 2018-10-20         715927         Max         Cat     Domestic Shorthair
## # ... with 1,138 more rows, and 2 more variables: secondary_breed <chr>,
## #   zip_code <chr>
```

and then see how many of those there are:

```
cats %>%
  filter(animal_name %in% top_ten_names) %>%
  summarize(rows = n())
```

```
## # A tibble: 1 x 1
##    rows
##   <int>
## 1  1148
```

with the same result. (I think you may have seen `%in%` in one of my Extras somewhere, or possibly in PASIAS).

(f) [2] It turns out that 6.6% of all the cats in Seattle have a name that is one of the most popular 10 cat names. (This is the answer your code of the previous part would have given if run on the actual data). Do you think that there are many different names people give their cats in Seattle, or only a few different names? Explain briefly. (There is no code required in this part.)

My answer:

The easiest way to tackle this is a what-if:

- if there are a lot of different cat names in Seattle, then the top 10 would be a small fraction of the total.
- If there were only a few different names, then the top 10 would be most of the total. (The extreme case of "only a few" is if the cat-owners of Seattle only gave their cats one of 10 different names; then the top 10 names would include *all* of the cats.)

The 6.6% seems to be a small fraction of all the cats (it was smaller than I expected), so we are in the first case: the top 10 are a small fraction of the total, so there must be a lot of different names people in Seattle give their cats.

Extra: how many? Well, I have the data, so I can tell you:

```
cat_counts %>%
  summarize(different_names = n())
```

```
## # A tibble: 1 x 1
##   different_names
##             <int>
## 1            7027
```

The 17,294 cats have 7027 different names between them. So there are *very many* different names that Seattle cat-owners have given their cats. (This is an explanation question; no code is required here.)

4. A researcher was interested in comparing ages of hockey defencemen and forwards, and looked at data from the 2019 Ottawa Senators hockey team. Some of the dataset is shown in Figure 9. The column `Forward` takes the value `yes` if the player in question is a forward, and `no` if the player is a defenceman. (This dataset does not include goaltenders.)

(a) [2] The researcher decided to use a t-test to compare the ages of forwards and defencemen. Looking at the information in Figure 10, how would you argue *in favour of* that decision?

My answer:

To use a t-test, the researcher needs to make the case that both groups (forwards and defencemen) have normal distributions of ages, given the sample sizes. The forwards' ages are slightly skewed to the left, but there are 18 of them, so the sample size appears large enough to overcome that. The ages of the defencemen are skewed more substantially to the right, and the sample size is only 8, so the researcher must have argued that even a sample size of 8 was large enough to overcome this skewness (or, equivalently, the researcher must have looked at the bootstrap sampling distribution of the sample mean and found that it was close to normal).

Asserting that the distributions of the ages within the two groups are "approximately normal" is not enough; even though there are no outliers, they are definitely skewed. The only way a t-test can work is if you consider the sample sizes to be "big enough". Max of 0.5 out of 2 if you assert this and stop.

Remember to answer the question as written, even if you disagree. (It's like being on the debate team: you don't have to agree with the point you are making, you just have to be able to articulate it persuasively.) But see below.

(A more sophisticated bootstrap than the ones we have done so far reveals that the difference in sample means between bootstrap samples of defencemen and forwards *does* in fact have very close to a normal distribution.)

I decided in the end to be more generous in the grading of this one: as long as you gave some thought to the shapes and sample sizes for the two distributions, and came to *a* conclusion about the t-test, even if it was different from mine, I was good. As long as you came to some kind of conclusion (that is to say, you made an honest attempt to answer the question), there was about a half point for consideration of each of shape and sample size for each of the two samples. For example, I decided that I was fine with you saying that a two sample t-test was no good because the sample of defencemen (8 observations) was too small to overcome the skewness in that distribution. The sample size for the forwards (18) was (in my estimation) large enough to overcome the small amount of skewness there (say this), but we need *both* groups to be normal enough given the sample sizes, and from this point of view, one of them is not. (Finding one group that is not normal enough given its sample size is actually enough, but you need to say why it's enough.)

The way we have looked at this, we have to consider the two samples separately, so that even though the combined sample size of $18 + 8 = 26$ looks large, this is not the relevant issue. A subtlety: the fact that the test statistic looks at the difference between the two sample means helps a bit more with the normality, which might be the reason that the more sophisticated bootstrap came out better than you might have guessed.

Extra: a hockey team (that is to say, an ice hockey team, not a field hockey team) has six players: a goaltender, two defencemen (D in the dataset), a centre (C), a left winger (LW), and a right winger (RW). A team will have several "lines" (sets of forwards and pairs of defencemen) that are frequently substituted for each other during a game.

(b) [3] A t-test is run in Figure 11. What code was used to run this t-test?

My answer:

We have been talking about comparing ages for the two groups of players defined by `Forward`. You can confirm that this was the case here by looking at the first line of output (below the title). The title tells you that this was a Welch test (rather that a pooled one), so you do not need `var.equal` in your code (or, I suppose, if you have it, it must be `FALSE`). Also, the alternative hypothesis in the output is two-sided, so you don't need an `alternative`. Hence:

```
t.test(Age ~ Forward, data = OttawaSenators2019)
```

```
##
##  Welch Two Sample t-test
##
## data:  Age by Forward
## t = -1.4615, df = 19.779, p-value = 0.1596
## alternative hypothesis: true difference in means between group no and group yes is not equa
## 95 percent confidence interval:
##  -4.4855683  0.7911238
## sample estimates:
##  mean in group no mean in group yes
##          23.87500          25.72222
```

Three points is rather generous here, so expect to lose one per mistake, with 1 point if you have any substantial part of it correct.

I didn't actually say the name of the dataset, but you can deduce it by looking at the table of counts under the boxplot, where I showed the code for getting that table; the dataframe's name must be the beginning of the pipeline there.

The test by default is two-sided, so the best style is to not specify the sidedness of the test. If you must specify an `alternative`, `two.sided` with a dot is the right one. But, for two reasons, this is not the best: (i) you don't need it, and including it makes it look as if you think it must be there, (ii) I didn't show you this in class. 2.5 for this.

I saw a few of these:

```
with(OttawaSenators2019, t.test(Age ~ Forward))
```

```
##
##  Welch Two Sample t-test
##
## data:  Age by Forward
## t = -1.4615, df = 19.779, p-value = 0.1596
## alternative hypothesis: true difference in means between group no and group yes is not equa
## 95 percent confidence interval:
##  -4.4855683  0.7911238
## sample estimates:
##  mean in group no mean in group yes
##          23.87500          25.72222
```

As you see, this works just fine, so full marks. The logic is that you can either specify the dataframe with `data=`, or by wrapping the whole thing in `with`.

(c) [2] What do you conclude from the *t*-test in Figure 11, in the context of the data?

My answer:

The null hypothesis here is that forwards and defencemen have equal mean ages (in the population from which these players were drawn, which might be "all players who could have played for the Ottawa Senators in 2019"). The P-value is 0.1596, so the null hypothesis is not rejected, and therefore there is no evidence that forwards and defencemen have different mean ages.

If you don't have a conclusion about mean *ages*, you will not get the second point, and you will probably have a "context?" comment from me. It is one thing to know how a hypothesis test works, but it is perhaps more important to be able to *apply* it to draw a conclusion about the data in front of you.

I was actually rather generous in giving 2 points for a conclusion that seemed to say the right thing for the right reason. But your hypothesis and conclusion should really at least hint at some kind of population (such as "all hockey players"). We have the entire Ottawa Senators 2019 roster, so we know the mean ages of forwards and defencemen there. The population that we are making inference about has to be *something* bigger than that, for a hypothesis test to be of value.

(d) [2] There are two kinds of *t*-test that might have been used here. Out of these two tests, was the one run here the better choice? Explain briefly.

My answer:

The two two-sample *t*-tests that might be appropriate here are the Welch test and the pooled test. The one used here was the Welch test. This does not assume equal spreads of ages in the two groups. The pooled test *does* assume equal spreads. Looking back at the boxplot in Figure 10, the spreads are definitely *not* equal, and therefore the Welch test as run was the better choice.

Make sure to say something about how you know the two groups (of forwards and defencemen) have different spreads of ages. You can refer to Figure 10 or to "the boxplot"; either is good. Saying that "we don't know" whether the spreads are equal or not will cost you a half point, because it is possible to know.

Strictly speaking, what the pooled test requires is that the two samples are drawn from *populations* with equal variances, but assessing the spreads (IQRs) in the boxplots is a reasonable way to do this.

Another way to put it is that the Welch test is not going to badly mislead you whether the variances are the same or different, which would be a (reasonable) argument that you should never use the pooled test, on the grounds that it can go badly wrong if the variances happen to be different. This is an easier argument to make, but it appears to be sound, and this way, you don't even have to refer to the boxplot!

Some people thought that the issue here was one-sided vs. two-sided. The way this question was set up, the issue of interest was a two-sided one (are there *any* differences in age between forwards and defencemen?), and there was no good way that one-sided would be better. (Especially not if you *look at the data* to decide whether to do a one-sided or two-sided test, which is a big no-no: the decision about the kind of alternative you use must be made before you look at the data, as an answer to the question "what am I trying to prove?".)

5. Figure 12 shows a power analysis. Use this Figure to answer the questions below, except where reference to another Figure is made.

   Explanations are not required, except where explicitly asked for, but may be worth partial credit if given and your answer is not correct. (An incorrect answer with no explanation is zero.) This is a long question, but answers should come quickly if you know what you are doing.

   (a) [2] What is the assumed true population mean?

   My answer:

   50 (the second number in the `rnorm`). This is a power analysis, so the true mean (50) and the null hypothesis mean (55) are *different*. Our aim will be to see how successful we are at (correctly) rejecting the null mean.

   (b) [2] What is the assumed population distribution? How do you know? (Your answer should contain the name of a probability distribution.)

   My answer:

   Normal, because of the use of `rnorm` to generate the random samples. The use of `t.test` later does not imply that the population is normal; it could be that the population is (say) slightly skewed, but in a way that a sample size of 15 would be large enough to overcome. One point for saying "normal", one for saying that it was because we used `rnorm`.

   Extra: the mean is 50, as we said before, and the SD is 9 (the third number in the `rnorm`).

   (c) [2] What is the null hypothesis being tested?

   My answer:

   Look in the `t.test` line, specifically at the `mu` there: $H_0 : \mu = 55$, where μ is the (hypothesized) population mean. Or, maybe clearer, write it out in words: "the population mean is equal to 55".

   I'll (generously) take "the mean is 55" or $\mu = 55$ without defining μ, but having a hypothesis with "sample mean" in it is definitely an error: the sample mean is what it is; the point of this stuff is to say something about a population mean that we don't know about for certain.

   If you thought the population mean was 55 above, and you gave an answer of 50 here, you might get a point here, if I thought that was a logical deduction from what you said before.

(d) [2] What is the alternative hypothesis being tested? Explain briefly.

My answer:

This is not explicitly stated, so we know it must be two-sided (the default), and therefore it is $H_a : \mu \neq 55$. (Said differently, there is no `alternative` given in the code, so we know it must be two-sided.)

If you said "sample mean" here and in (c), you should only get penalized for it once. Likewise, if you messed up (c) somehow and in my judgement that caused you not to get (d), you can get credit for (d).

(e) [2] What is the sample size in Figure 12?

My answer:

15: the first number in the `rnorm` is the sample size.

If you thought the answer was 1000, that's the number of *simulations*, each of which uses a new random sample of size 15. Saying something by way of explanation might get you some credit for an answer of 1000, but without explanation, that's a very fast zero.

(f) [2] Why do two of the code lines in Figure 12 have `list` in them? Explain briefly.

My answer:

The code inside the `list` produces more than a single number: a sample of 15 values the first time, and all the output from the *t*-test the second time. To store all of that in a single cell of the dataframe, we wrap it in `list` to create a list-column.

(g) [2] How do you know that this simulation is estimating power, and not the probability of making a Type I error? Explain briefly.

My answer:

We know that the null hypothesis is not true (the true mean is 50, and the null-hypothesis mean is 55: they are different), so we are indeed estimating the probability of rejecting the null hypothesis when it is *false*.

One (somewhat generous) point for some sensible general discussion of how power differs from type I error, but to get the second you need to say how you know the null is false *here*. This was the point of having you do (a) and (c): having decided that the true and null means were different earlier, you would be well-placed to tackle this one. (Something I often see is that people might have a good grasp of general principles, which is all fine and well, but the point of a course like this one is to be able to apply those general principles to the situation in front of you.)

To estimate the probability of a type I error, we would need the true mean and the null mean to be the *same*, for example the same `rnorm` line, but in the `t.test` line, have `mu = 50`. (The answer in that case ought to be very close to 0.05, since the simulated data really are normal.)

(h) [2] What is your estimate of the power of the test?

My answer:

$492/1000 = 0.492$.

Writing the calculation 492/1000 is enough.

(i) [2] A second power analysis is shown in Figure 13. How does this differ from the first power analysis, Figure 12, in the context of the data?

My answer:

It is using a sample size of 30 instead of a sample size of 15.

Saying that we are using `rnorm(30, 50, 5)` instead of `rnorm(15, 50, 5)` demonstrates very little understanding of what is actually being done here (or why you might want to do it). I really don't think this kind of answer is worth more than half a point.

I suppose there is something for "the power is bigger", but the point of the question is "what happened to make the power bigger?", namely, that the sample size increased.

(I tried to give the points for anything that got as far as "the sample size is bigger". Anything extra that you think about here will help you for the next part. Good thoughts to have here are noting that the larger sample size has the larger power, or even saying the sample mean is more likely to be closer to the (true) population mean if the sample size is larger, and therefore we should more easily be able to reject the incorrect null with a larger sample size.)

(j) [2] Suppose that you are aiming for a power of 0.80. What do Figures 12 and 13, taken together, tell you about the sample size you should use? Explain briefly.

My answer:

The power for a sample size of 15 is too small, and the power for a sample size of 30 is a bit too big, so we should use a sample size between 15 and 30. That's all you need to say. "Close to 30" is also all right, or "a sample size of 30 gives power close to 0.80", but saying 30 without qualifying it somehow is, as it were, not close enough.

If you want to go a bit further, you might say that the power for a sample size of 30 appears to be only a little bigger than 0.80, and so the sample size we should use is a little less than 30. Something like 25 would be a good guess, based on what we have seen here (or maybe bigger than 25).

Saying that a bigger sample size is better is something, but in the real world, there is a limit to how big a sample we can afford. That is why we aim for something like a power of 0.80 (or some other value that is not 1); this is our compromise between the cost of a bigger sample and its value to us. In this case, I could take a sample of size 100, but its power would be bigger than I need, so that I am wasting sampling effort by taking such a big sample: the gain in power beyond 0.80 is not worth the extra sampling cost to get it.

Extra: As some people noted, this is the kind of situation where `power.t.test` would work (sampling from a normal distribution, because I didn't want to give you anything more confusing than `rnorm`):

```
power.t.test(delta = 55-50, power = 0.80, sd = 9, type = "one.sample")
```

```
##
```

```
##      One-sample t test power calculation
##
##               n = 27.4131
##           delta = 5
##              sd = 9
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
```

This says that the sample size should be 28, which is entirely in agreement with what we just said. This is nice as an extra to *your* answer, but the main point should be that a sample size of 15 is too small and one of 30 is a little too big. (If you guessed at the right sample size in your answer, now you know how close your guess was!)

6. Thirty-one determinations of nickel content, in parts per million, were made from a Canadian syenite rock. The data, in column `nickel` of dataframe `abbey`, are shown in Figure 14. Our aim is to estimate the "average" nickel content of rock of this type, where "average" could be mean or median.

   (a) [2] What feature of Figure 15 would lead you to have doubts about using a t procedure (test or confidence interval) here? Explain briefly.

   My answer:

   This has an extreme upper outlier, suggesting that the distribution of the nickel determinations is very far from normal in shape. Something at least close to normality is needed for a t procedure, which this is not.

   I think this is really one outlier (an isolated value), rather than right skewness (which would be a feature of the distribution as a whole), but I'll accept either. In fact, the highest value is over 100, and the next highest is 34, so we most likely do have one outlier. A histogram with more bins might offer more insight about the rest of the distribution, but we don't have that here, so we have to make do with what we have.

   In any case, saying that the distribution is not normal is one point, and saying how you know, or in what way it is not normal, is the second point.

   (b) [1] What other feature of the data should also be considered in assessing the use of a t procedure? (No explanation needed.)

   My answer:

   The sample size. All that's needed for the point. The sample size here is 31. Talking about the sample size in (a) rather than here will also net you the point here.

   A larger sample size means that the non-normality of the data matters less. You might think that a sample size of 31 is big enough, but read on. (This goes to show that there is nothing magic about a sample of size 30.)

   What is needed here is some feature of the data that *does not* appear on the histogram. Discussions of shape, outliers etc. are what you needed to see in (a); here you want something distinct from that.

   (c) [4] What is the graph in Figure 16 a graph of? How does this graph *add* to your previous conclusion? What, therefore, is your overall conclusion about the validity of a t-procedure for these data? Explain briefly.

   My answer:

   The graph is a histogram of the bootstrap sampling distribution of the sample mean. (Strictly, a simulation of 1000 values from it, rather than the thing itself.) One point for a big enough fraction of that.

   Figure 16 is clearly not normal: it is skewed to the right. It has added to our conclusion by confirming that the sample size of 31 was in fact *not* big enough to overcome the effect of the outlier in Figure 15. Two points. The outlier is so distant from the rest of the data that it would take an even larger sample size for a t-procedure to be appropriate. The credit here is for building the bridge between your earlier conclusions and what you saw in Figure 16. Saying something like "the Central Limit Theorem does not apply here" is another way to imply that the sample size is not big enough (if it were, the bootstrap sampling distribution would be normal rather than skewed).

This means that the sampling distribution of the sample mean here is not close to normal, and therefore that a t procedure will not be appropriate. (This is my overall conclusion, one point.)

I saw some really good answers here. The very best of them got a "nice!" tag in Crowdmark which its recipients will see when they get their graded exams.

I think the best answers here say that the sampling distribution of the sample mean is *not* close enough to normal, bearing in mind that the bootstrap incorporates the sample size, but I was willing to be swayed by a well-made argument that the sampling distribution *is* in fact normal now and therefore the t-test is all right despite the nickel values themselves having a big outlier. I needed to be convinced that you had a good handle on the relevant issues though.

(d) [2] Two confidence intervals are shown in Figures 17 and 18. Which one do you prefer, and why? Your explanation should make it clear why one of them is bad and the other one is good.

My answer:

Figure 17 is a confidence interval for the mean, based on the t-distribution, which we previously said would not be appropriate (the nickel content values are not normal enough, even given the sample size). One point.

Figure 18 is a confidence interval for the median, based on a sign test. This does not assume a normal distribution for the nickel content values, and so is appropriate to use here. The other point.

For these, each point is for doing two things: (i) identifying what the interval is, or otherwise making it clear that you know, and (ii) saying how you know that it is good or bad.

Extra: the interval for the mean is much wider than the one for the median, because the one for the mean uses the sample SD, which will be greatly inflated by the outlier. This is one of those cases where the two intervals are very different, so it's important to use the right one. This is a useful point to make, but not as your only point: yes, it's nice to have a shorter interval, but say something about why the t-interval is wrong (distorted by the outlier). It's possible that the longer interval is actually more trustworthy unless you rule that out.

(e) [3] A hypothesis test is shown in Figure 19. What is the null hypothesis for this test? Suppose you are trying to detect any change from the null hypothesis. What do you conclude, in the context of the data?

My answer:

The null hypothesis would be that the population *median* (nickel content) is 15. One point. The word “median” needs to appear in your answer somewhere (if not here, then in your conclusion), and the word “mean” is definitely wrong.

Detecting “any change” means that the appropriate alternative hypothesis is two-sided, so that the appropriate P-value is 0.0107. Thus we reject the null hypothesis and conclude that the population median nickel content is in fact not 15. The other two points.

The question asks for a two-sided test, so make sure you don’t conclude that the median is *less* than 15. Perhaps it is worth saying here that you don’t get to choose your alternative hypothesis after looking at the data. This has to be done first, before looking at the data, when thinking about what you are trying to prove.

(f) [2] How might you have guessed from Figure 19, even without seeing the P-value, that the P-value would be small? Explain briefly.

My answer:

In the count of values above and below the null median (at the top of the Figure), there are 23 observations below the null median of 15 and only 8 above. One point for saying this much. To get the second, note that this is a much more uneven split than we would have expected if the null median were correct (the median were really 15), and therefore we would guess that the P-value should be small. (Imagine tossing 31 coins and getting only 8 tails.) Another way to say this is that if the P-value were going to be large, we would expect to see a roughly equal number of values above and below 15. Or, with so many sample values below 15, the (population) median ought to be less than 15.

If you look at Figure *18*, you’ll see that confidence interval for the median does not include 15. That tells you that the (two-sided) P-value will be less than 0.05. However, this is not the Figure I asked you to look at. One (rather generous) point for this, on the grounds that this is a sensible thought process (albeit not the one I was asking about).

Use the rest of this page if you need more space. Be sure to label any answers here with the question and part they belong to.

(there should probably be a figures title page here)

```
library(tidyverse)
library(readxl)
library(smmr)
```

Figure 1: Packages

```
  treatment height
   low       10.1
   low        9.7
   medium     4.8
   medium    11.3
   high      10.3
   high       9.8
```

Figure 2: Data file stored in `biscuit1.txt`

```
treatment length
low 6.1
low 6.5
medium 8.7
medium 10.3
high 3.1
high 13.8
```

Figure 3: Data file stored in `biscuit2.txt`

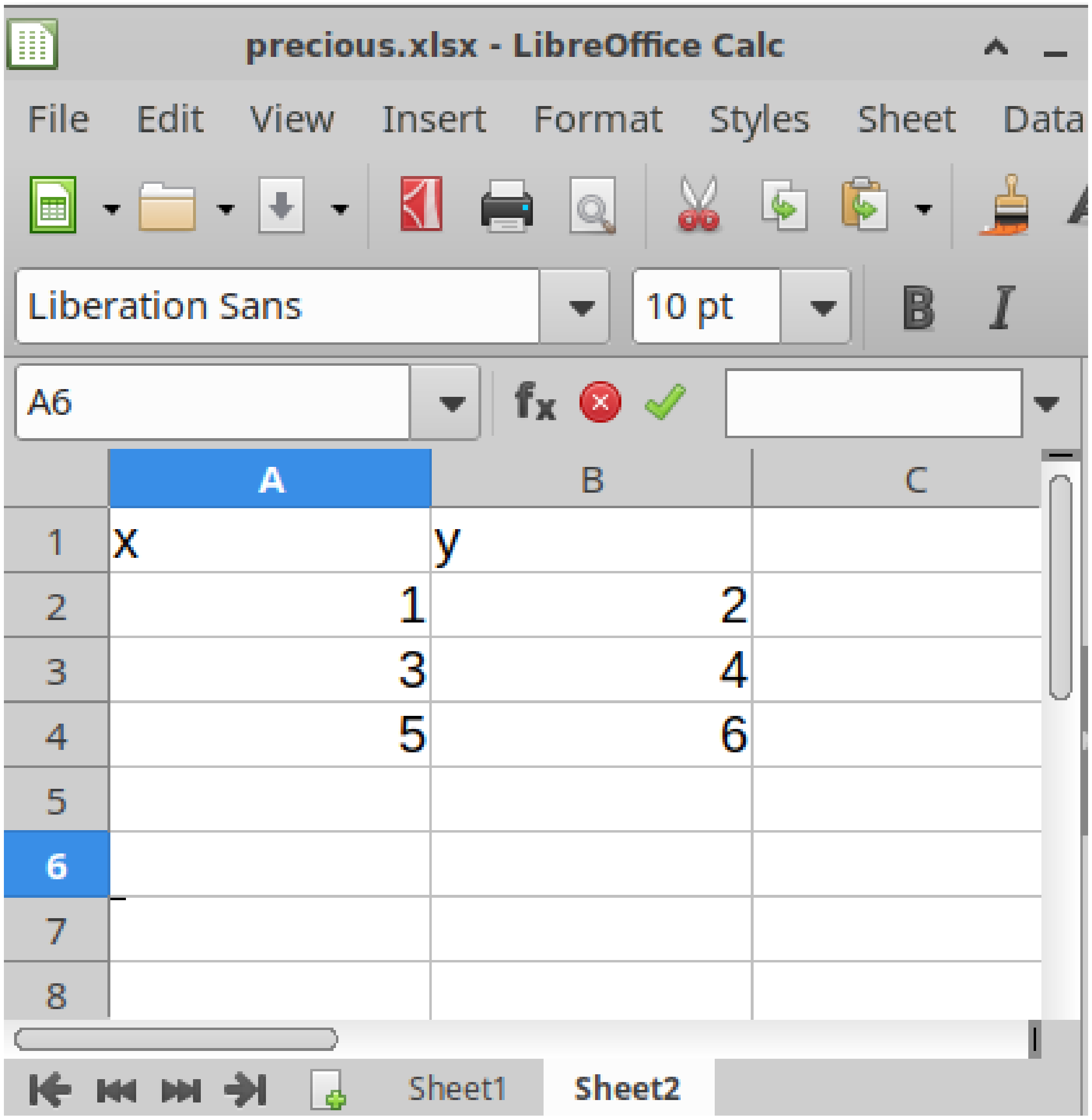
```
test3 <- read_delim("biscuit2.txt")
```

```
## Rows: 6 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: " "
## chr (1): treatment
## dbl (1): length
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The dataframe as read in:

```
## # A tibble: 6 x 2
##   treatment length
##   <chr>      <dbl>
## 1 low          6.1
## 2 low          6.5
## 3 medium       8.7
## 4 medium      10.3
## 5 high         3.1
## 6 high        13.8
```

Figure 4: Alternative way of reading data file stored in `biscuit2.txt`



	A	B	C
1	x	y	
2	1	2	
3	3	4	
4	5	6	
5			
6			
7			
8			

Figure 5: Spreadsheet to be read into R

```
eggs %>% slice(1:20)
```

```
## # A tibble: 20 x 3
##    day   weight day_number
##    <chr>  <dbl>      <dbl>
##  1 A         55          1
##  2 A         53          1
##  3 A         56          1
##  4 A         63          1
##  5 A         66          1
##  6 A         58          1
##  7 A         53          1
##  8 A         57          1
##  9 A         61          1
## 10 A         53          1
## 11 B         59          2
## 12 B         62          2
## 13 B         56          2
## 14 B         51          2
## 15 B         61          2
## 16 B         75          2
## 17 B         57          2
## 18 B         60          2
## 19 B         55          2
## 20 B         74          2
```

Figure 6: Egg weight data (first 20 rows)

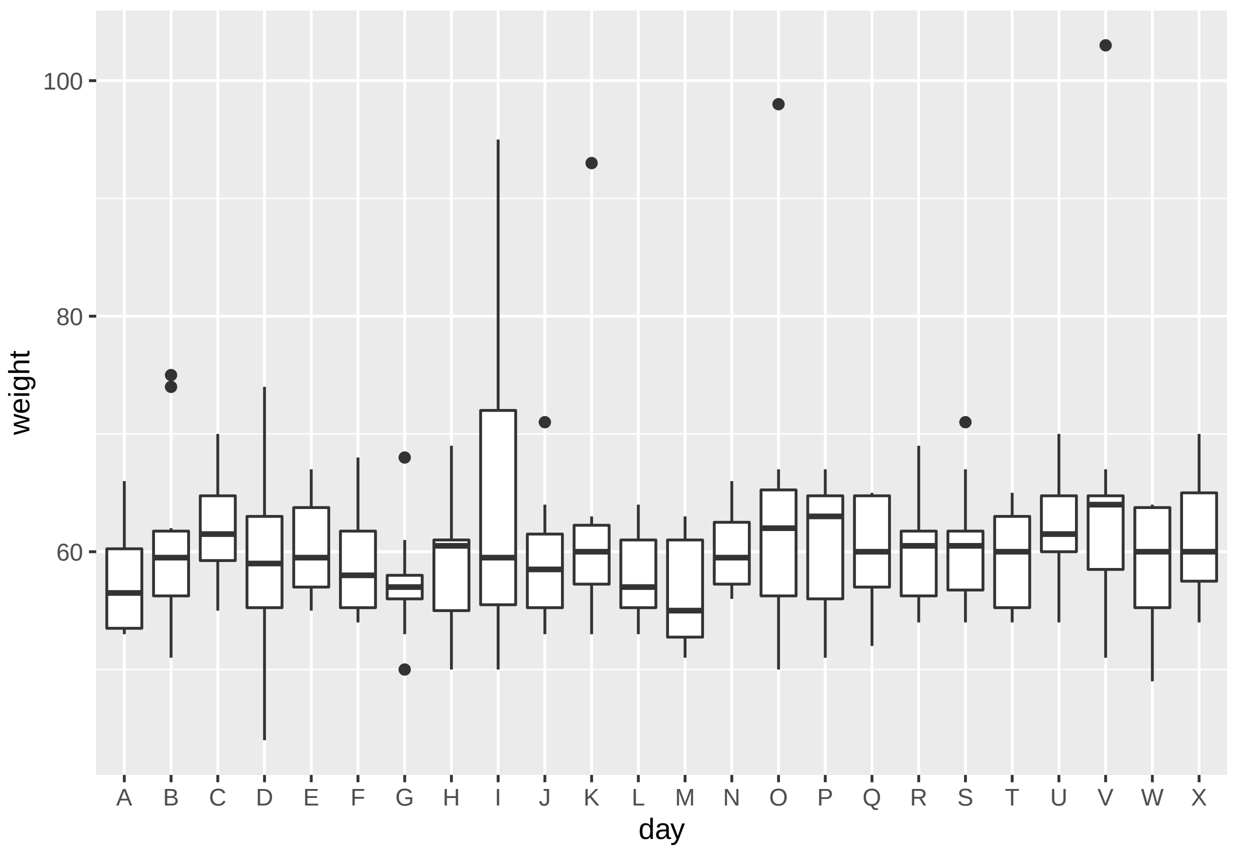


Figure 7: Graph of egg weight data

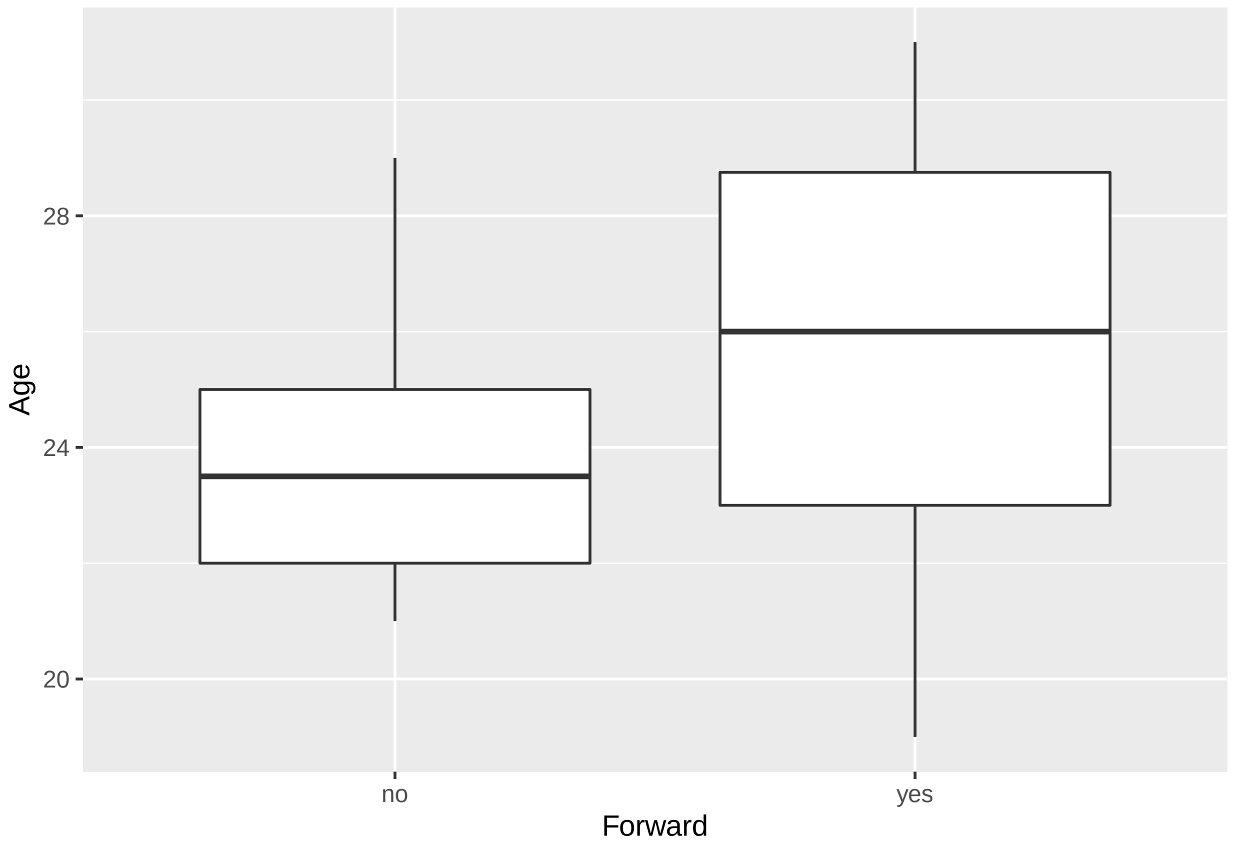
```
seattlepets
```

```
## # A tibble: 52,519 x 7
##    license_issue_date license_number animal_name species primary_breed
##    <date>             <chr>          <chr>       <chr>   <chr>
##  1 2018-11-16         8002756        Wall-E      Dog     Mixed Breed, Medium (u~
##  2 2018-11-11         S124529        Andre       Dog     Terrier, Jack Russell
##  3 2018-11-21         903793         Mac         Dog     Retriever, Labrador
##  4 2018-11-23         824666         Melb        Cat     Domestic Shorthair
##  5 2018-12-30         S119138        Gingersnap  Cat     Domestic Shorthair
##  6 2018-12-16         S138529        Cody        Dog     Retriever, Labrador
##  7 2017-10-04         580652         Millie      Dog     Terrier, Boston
##  8 2018-08-09         S142558        Sebastian   Cat     Domestic Shorthair
##  9 2018-08-20         S142546        Madeline    Cat     Domestic Shorthair
## 10 2018-12-08         S123830        Cleo        Cat     Domestic Shorthair
## # ... with 52,509 more rows, and 2 more variables: secondary_breed <chr>,
## #   zip_code <chr>
```

Figure 8: Seattle pets data (some)

```
## # A tibble: 26 x 8
##    Player          Position   Age Goals Assists PlusMinus PenMins Forward
##    <chr>           <chr>    <dbl> <dbl>   <dbl>     <dbl>   <dbl> <chr>
##  1 Chris Tierney   C           24     9      39       -22      26 yes
##  2 Magnus Paajarvi LW          27    11       8       -14       6 yes
##  3 Bobby Ryan      LW          31    15      27       -29      35 yes
##  4 Dylan DeMelo    D           25     4      18        -1      32 no
##  5 Cody Ceci       D           25     7      19       -22      18 no
##  6 Brady Tkachuk   LW          19    22      23       -10      75 yes
##  7 Colin White     C           22    14      27       -24      24 yes
##  8 Mikkel Boedker  RW          29     7      28       -23       6 yes
##  9 Thomas Chabot   D           22    14      41       -12      32 no
## 10 Zack Smith      C           30     9      19        -6      81 yes
## # ... with 16 more rows
```

Figure 9: Ottawa Senators data (some)



```
OttawaSenators2019 %>%
  count(Forward)
```

```
## # A tibble: 2 x 2
##   Forward     n
##   <chr>   <int>
## 1 no          8
## 2 yes        18
```

Figure 10: Boxplot by position and age, with other information

```
##
##  Welch Two Sample t-test
##
## data:  Age by Forward
## t = -1.4615, df = 19.779, p-value = 0.1596
## alternative hypothesis: true difference in means between group no and group yes is not equal to 0
## 95 percent confidence interval:
##  -4.4855683  0.7911238
## sample estimates:
##  mean in group no mean in group yes
##          23.87500          25.72222
```

Figure 11: *t*-test for Ottawa Senators data

```
tibble(sim = 1:1000) %>%
  rowwise() %>%
  mutate(my_sample = list(rnorm(15, 50, 9))) %>%
  mutate(t_test = list(t.test(my_sample, mu = 55))) %>%
  mutate(p_value = t_test$p.value) %>%
  count(p_value <= 0.05)
```

```
## # A tibble: 2 x 2
## # Rowwise:
##   `p_value <= 0.05`     n
##   <lgl>             <int>
## 1 FALSE               508
## 2 TRUE                492
```

Figure 12: A power analysis

```
tibble(sim = 1:1000) %>%
  rowwise() %>%
  mutate(my_sample = list(rnorm(30, 50, 9))) %>%
  mutate(t_test = list(t.test(my_sample, mu = 55))) %>%
  mutate(p_value = t_test$p.value) %>%
  count(p_value <= 0.05)
```

```
## # A tibble: 2 x 2
## # Rowwise:
##   `p_value <= 0.05`     n
##   <lgl>             <int>
## 1 FALSE               159
## 2 TRUE                841
```

Figure 13: A second power analysis

```
abbey
```

```
## # A tibble: 31 x 1
##    nickel
##     <dbl>
##  1    5.2
##  2    6.5
##  3    6.9
##  4    7
##  5    7
##  6    7
##  7    7.4
##  8    8
##  9    8
## 10    8
## # ... with 21 more rows
```

Figure 14: Nickel content data (some)

```
ggplot(abbey, aes(x = nickel)) + geom_histogram(bins = 10)
```

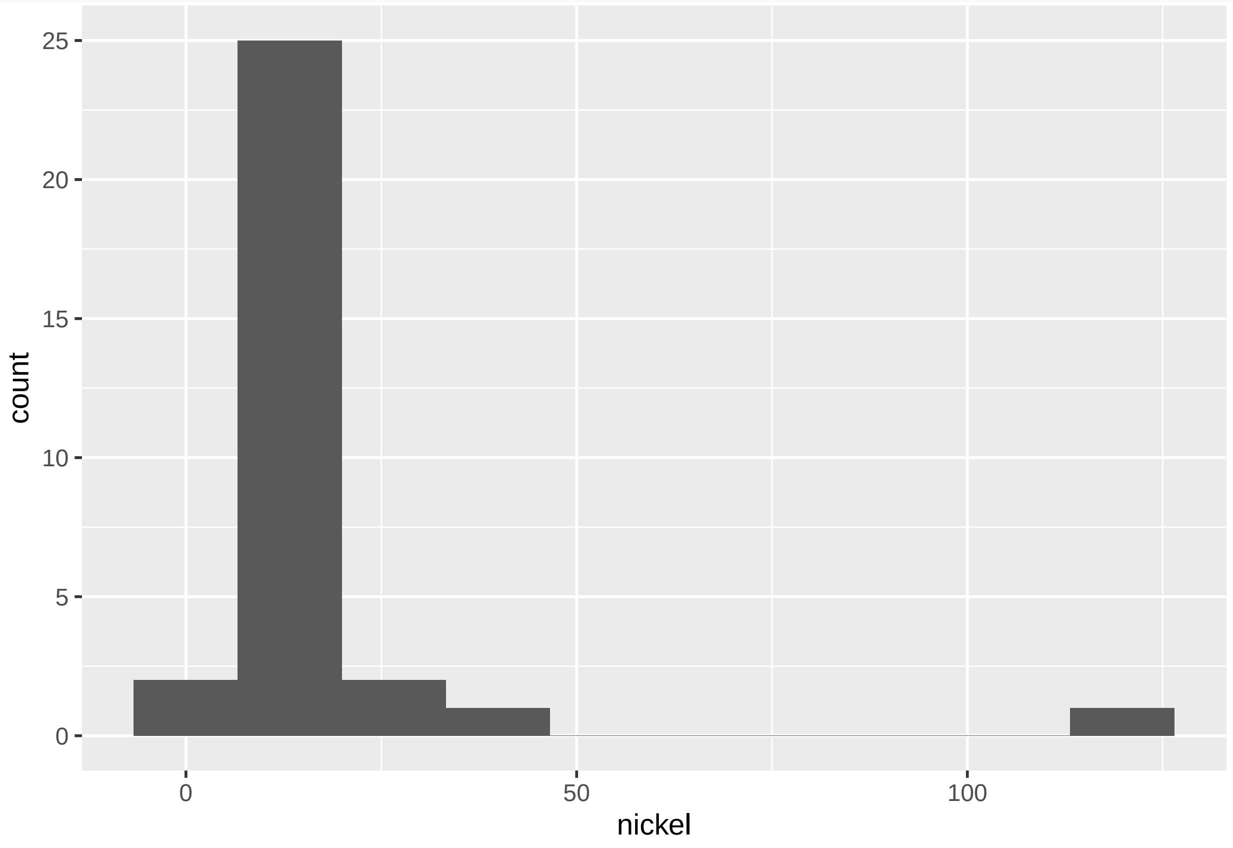


Figure 15: Nickel content histogram

```
tibble(sim = 1:1000) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(abbey$nickel, replace = TRUE))) %>%
  mutate(my_mean = mean(my_sample)) %>%
  ggplot(aes(x = my_mean)) + geom_histogram(bins = 10)
```

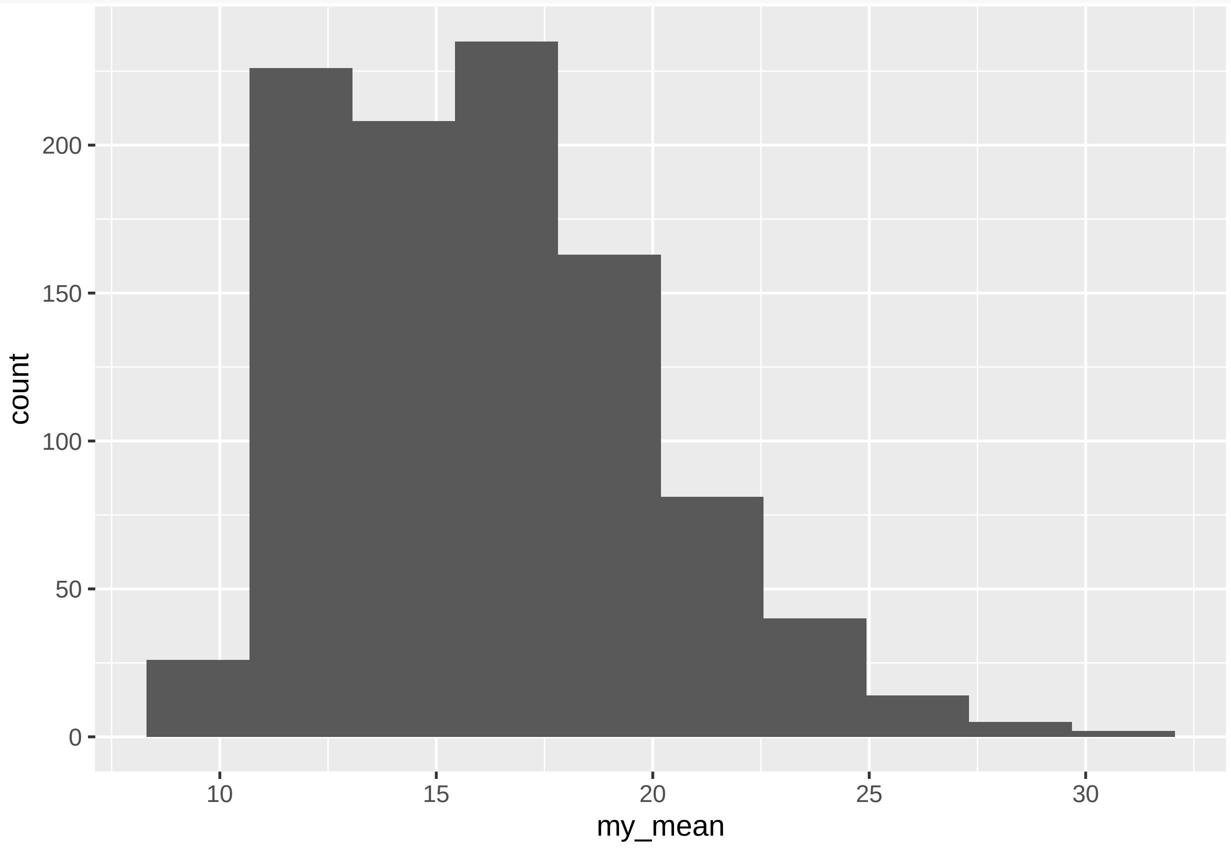


Figure 16: Nickel content further analysis

```
with(abbey, t.test(nickel))
```

```
##
##  One Sample t-test
##
## data:  nickel
## t = 4.1901, df = 30, p-value = 0.0002259
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##   8.204894 23.808009
## sample estimates:
## mean of x
##  16.00645
```

Figure 17: Nickel content confidence interval 1

```
ci_median(abbey, nickel)
```

```
## [1]  8.005078 13.997131
```

Figure 18: Nickel content confidence interval 2

```
sign_test(abbey, nickel, 15)
```

```
## $above_below
## below above
##    23     8
##
## $p_values
##   alternative    p_value
## 1       lower 0.00533692
## 2       upper 0.99833655
## 3   two-sided 0.01067384
```

Figure 19: A test