Week 2: Practical Exercise

Calum Webb

09/09/2022

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

In R Markdown you need to style your writing in plain text - it will then read this plain text and format it however you want it to be formatted. For example, headings always begin with a # sign like above. The more # signs, the smaller the heading. **Bold** text is text with two asterixes or **underscores** around it and *italicised* text is text with only one asterix or *underscore* around it.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
print("Hello SMI606!")
```

```
## [1] "Hello SMI606!"
```

Your RMarkdown file will only knit successfully if there are no errors in your code, so you will need to wait to try it until the end of the worksheet (or test it out using the week-2-answers.Rmd file)!

You can run the code in the chunks just like you would in a normal R script: just highlight it and click run or press control/command and enter while your typing cursor is on it.

Introduction

By the end of this tutorial, you will have practiced:

- Using the `mean()`, `median()`, `mfv()`, and `sd()` functions to calculate descriptive statistics (measures of central tendency and dispersion) in `R`
- Using the convenience functions `summary()`, `tabyl()`, and `skim()` to generate multiple descriptive statistics at once
- Programming a custom `summarise()` function with your chosen descriptive statistics
- Plotting histograms and bar charts for visualising continuous and nominal/ordinal variables in both base `R` and in `ggplot2`

You will also be revisiting and reinforcing some of the learning from week one on loading packages, reading in data, and using functions.

Load Relevant Packages

Exercise

Start by writing the code to load the `tidyverse` and `modeest` packages from the library. Remember that the `tidyverse` contains a lot of additional functions that we can use and the `modeest` package can be used for calculating the mode for a given variable.

Remember to load a package to take it down from the `library()`, and remember that you might have to install any missing packages with `install.packages()` (e.g. `install.packages('tidyverse')`) in your console if you get an error message!

```
# For example, this is how we would load the tidyverse package
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.1     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
# Write the equivalent code below to load the "modeest" package for calculating modes
library(modeest)
```

Load data for these exercises

For this week's exercises we will be using a subset of data from the Health Survey for England (2011) teaching dataset, which is provided as part of Quantitative Social Science Data with `R`.

This data is stored within a folder called 'data' in the working directory. It is called `hse.rds`.

Exercise

Write a line of `R` code that allows you to *read* in the data and assign it to the `hse_data` object. Remember that this file type is `.rds` and we use the `read_*()` functions to read in data.

```
# Finish the line of code below so that it successfully reads in the
# data called "hse.rds" that is found in the data directory.
hse_data <- read_rds("hse.rds")
```

Summary Statistics: Measures of Central Tendency

Using the `mean()` function

The first measure of central tendency we will explore using in `R` is the mean. The mean can be calculated through the inbuilt base function `mean()`.

Try running the mean function below to get the mean equivalised income (income adjusted for household size) for households in the Health Survey for England.

```
mean(hse_data$eqvinc)
```

```
## [1] NA
```

What do we get? Is this what we would expect? Why/why not?

`na.rm` and explicit removal of missing data.

In many functions in `R` you are required to explicitly state that you wish to remove missing data when calculating a statistic like a mean. While this might seem counter intuitive as there is always likely some missing data in your dataset, it can actually encourage good practice and help you be aware of the errors that might come from not recognising how much of your data is missing.

We explicitly state that we wish to remove missing values from our data before calculating the mean by adding the argument: `na.rm = TRUE` — which can be read as missing (na) remove (rm) equals true. In R functions arguments are seperated by commas within the function's brackets.

Exercise

Add the `na.rm = TRUE` argument to the `mean()` function to calculate the mean.

```
mean(hse_data$eqvinc, na.rm = TRUE)
```

```
## [1] 33274.13
```

Using the `median()` function

The `median()` function operates in an almost identical fashion to the `mean()` function, and also requires `na.rm = TRUE` to be manually specified to calculate medians for vectors with missing data.

Example

```
median(hse_data$eqvinc, na.rm = TRUE)
```

```
## [1] 23442.62
```

Exercise

Use the `median()` function to get the median for the variable `porfv` — 'portions of fruit and vegetables eaten the day prior to the survey':

```
median(hse_data$porfv, na.rm = TRUE)
```

```
## [1] 3.333333
```

How many portions of fruit and vegetables does the average person eat? Enter your answer to two decimal places below.

–

3.33

–

Using the `mvf()` function (from `modeest`)

There is no base function for calculating the mode (or most frequent value) in `R`, but the package `modeest` that we loaded earlier contains a function for doing so called `mvf()`.

Example

```
mfv(hse_data$eqvinc, na.rm = TRUE)
```

```
## argument 'na.rm' is soft-deprecated, please start using 'na_rm' instead
```

```
## [1] 10655.74
```

Using the `sd()` function for standard deviation

`sd()` is used in `R` for calculating standard deviation. The `sd()` function follows the same structure as the `mean()`, `median()`, and `mfv()` functions — using this information, get the standard deviation for the equivalised income variable by calling the `sd()` function below.

Exercise

No clues here! Using what you've learned from the last exercises, try using the `sd()` command to get the standard deviation of the equivalised income variable `eqvinc` — remember, you can bring up the documentation for the `sd()` function by typing `?sd` into your `R` console.

```
sd(hse_data$eqvinc, na.rm = TRUE)
```

```
## [1] 30347.75
```

Generating quantiles with `quantile()`

Before we move onto convenience functions for generating entire tables of descriptive statistics, we will also cover how to generate quantiles. As you saw in the lecture, standard deviation is often a poor way of summarising the distribution of a variable if that variable is not normally distributed. We can use a 'non-parametric' method of summarising the data using 'quantiles' instead; the `quantile()` function in `R` can be used to calculate these.

By default, the `quantile()` function will return the "0th" percentile (the minimum value); the 25th percentile (the lower bound of a 'quartile' - split into four parts - range); the 50th percentile (which is equivalent to the median); the 75th percentile (the upper bound of a 'quartile'); and the "100th" percentile (the maximum value). Here is an example of using the quantile function with the `eqvinc` variable.

Example

```
quantile(hse_data$eqvinc, na.rm = TRUE)
```

```
##          0%         25%         50%         75%        100%
##    271.0843  14300.0000  23442.6230  43624.1611 262295.0820
```

If we want to set our own custom quantile ranges we can use the optional `probs` argument within the quantile function. For example, if we wanted a quantile range that spanned the central 95% of the data range, similar to a 2 times standard deviation range, we could ask for the 2.5th percentile and the 97.5th percentile (because there are 95 percentiles between 2.5 and 97.5 with the 50th percentile being in the middle).

`quantile()` requires the ranges to be specified as a probability (between 0 and 1), so the equivalent probability for 2.5 per cent would be 0.025 and 0.975 for 97.5% (if unsure, just divide the percentage form by 100: $97.5 \ / \ 100 = 0.975$)

Example

```
quantile(hse_data$eqvinc, probs = c(0.025, 0.975), na.rm = TRUE)
```

```
##     2.5%    97.5%
##   4062.5 115000.0
```

Exercise

Now it's your turn to write a quantile function. This time, I want you to get a 90 per cent quantile range for the `wtval` variable in the data (a person's weight in kg in the Health Survey England data).

```
quantile(hse_data$wtval, probs = c(0.05, 0.95), na.rm = TRUE)
```

```
##    5%   95%
##  53.4 108.6
```

`summary()` for quick descriptive statistics

Okay! Now that we have gone through the difficult part of how to calculate these statistics using their underlying functions I can introduce you to some convenience functions for quickly getting summary statistics.

The first is `R`'s inbuilt `summary()` function. `summary()` is a context-sensitive function that operates differently depending on what kind of variable or object is passed to it. For example, if you give it a numeric variable it will print is minimum value, its 1st Quartile (25th percentile) value, its median, its mean, its 3rd quartile (75th percentile), and its maximum value.

Exercise

Use the `summary()` function to get descriptive statistics for the age of the `hse_data` sample using the `hse_data$age` variable.

```
summary(hse_data$age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17.00   35.00   49.00   50.07   64.00  100.00
```

What is the mean age of the sample to two decimal places (e.g. 10.09)? Type the numeric value below.

_

50.07

_

`summary()` for nominal/ordinal variables

When `summary()` detects that it has been passed a nominal or oridinal variable (based on whether it's received a factor or string variable rather than numeric - see last week's R exercises if you're unsure what these mean), it will provide a frequency count of all of the values in the dataset.

Exercise

Try this out by running `summary()` on the `cigs` variable within the `hse_data` dataset.

```
summary(hse_data$cigs)
```

```
##       1. Non-Smoker   2. Under 10 a Day   3. 10 to 20 a Day 4. 20 or More a Day
##                6743                 607                 712                 369
##                NA's
##                  72
```

How many people in the `hse_data` sample smoke 20 or more cigarettes a day? Type the numeric value below.

_

369

_

`summary()` for entire dataset descriptives

Lastly, you can also use `summary()` to print descriptive statistics for every variable in a dataset by not including a specific variable/vector when you call it (no `$variable_name` after the dataset's name). This is generally a bad idea because some datasets can be huge, but you can do it.

Try this below by using `summary()` to output descriptive statistics for the entire `hse_data` dataset.

```r
summary(hse_data)
```

```
##     hserial           pserial
##  Min.   :1001011   Min.   :100101101
##  1st Qu.:1136066   1st Qu.:113606602
##  Median :1275151   Median :127515101
##  Mean   :1277856   Mean   :127785559
##  3rd Qu.:1421096   3rd Qu.:142109602
##  Max.   :1562161   Max.   :156216101
##
##                                                          tenureb
##  Buying it with the help of a mortgage or loan              :2980
##  Own it outright                                            :2848
##  Rent it                                                    :2523
##  Live here rent free (including rent free in relative s/frien:  76
##  Pay part rent and part mortgage (shared ownership)         :  47
##  (Other)                                                    :   0
##  NA's                                                       :  29
##                       sex            age
##  Refusal                :   0   Min.   : 17.00
##  Don't Know             :   0   1st Qu.: 35.00
##  Schedule not applicable:   0   Median : 49.00
##  Item not applicable    :   0   Mean   : 50.07
##  Male                   :3777   3rd Qu.: 64.00
##  Female                 :4726   Max.   :100.00
##
##                       topqual3                            econact
##  No qualification         :2009   In employment              :4616
##  NVQ4/NVQ5/Degree or equiv:2008   Retired                    :2265
##  NVQ2/GCE O Level equiv   :1744   Other economically inactive:1204
##  NVQ3/GCE A Level equiv   :1238   ILO unemployed             : 376
##  Higher ed below degree   : 945   Refused                    :   0
##  (Other)                  : 516   (Other)                    :   0
##  NA's                     :  43   NA's                       :  42
##                                          nssec8
##  Lower managerial and professional occupations :1880
##  Semi-routine occupations                      :1429
##  Intermediate occupations                      :1256
##  Routine occupations                           :1141
##  Higher managerial and professional occupations:1025
##  (Other)                                       :1586
##  NA's                                          : 186
##                                                 eth_origin
##  White - English/Welsh/Scottish/Northern Irish/British:7094
##  Any other white background                           : 414
##  Indian                                               : 203
##  Pakistani                                            : 138
##  African                                              : 117
##  (Other)                                              : 497
##  NA's                                                 :  40
##      eqvinc             htval           wtval           omsysval
##  Min.   :   271.1   Min.   :138.1   Min.   : 35.90   Min.   : -7.0
##  1st Qu.: 14300.0   1st Qu.:160.2   1st Qu.: 65.10   1st Qu.:114.5
##  Median : 23442.6   Median :167.1   Median : 75.90   Median :124.5
```

```
##  Mean   : 33274.1   Mean   :167.6   Mean   : 77.51   Mean   :125.6
##  3rd Qu.: 43624.2   3rd Qu.:174.5   3rd Qu.: 87.35   3rd Qu.:136.5
##  Max.   :262295.1   Max.   :202.5   Max.   :184.30   Max.   :203.5
##  NA's   :1781       NA's   :1324    NA's   :1392     NA's   :3779
##     omdiaval          porfv                          cigs
##  Min.   : -7.00   Min.   : 0.000   1. Non-Smoker      :6743
##  1st Qu.: 65.50   1st Qu.: 2.000   2. Under 10 a Day  : 607
##  Median : 72.50   Median : 3.333   3. 10 to 20 a Day  : 712
##  Mean   : 72.46   Mean   : 3.656   4. 20 or More a Day: 369
##  3rd Qu.: 79.50   3rd Qu.: 5.000   NA's               :  72
##  Max.   :122.50   Max.   :30.000
##  NA's   :3779     NA's   :10
##                health               marital     employed_lgc
##  1. Bad/Very Bad  : 620   0. Not Married:4000   Mode :logical
##  2. Fair          :1588   1. Married    :4501   FALSE:3845
##  3. Good/Very Good:6288   NA's          :   2   TRUE :4616
##  NA's             :   7                         NA's :42
##
##
##
```

`select()`ing a subset of variables to `summary()`

As we can see, this is quite a lot of information printed and some datasets have hundreds or even thousands of variables. This would make using `summary()` alone quite unpractical.

We can use the `select()` function from the `dplyr` package (included in the `tidyverse`) to select only certain variables.

First, we might need to remind ourselves what variables are in the dataset itself and what they are called. We can do this by running the `names()` function.

```
names(hse_data)
```

```
##  [1] "hserial"      "pserial"      "tenureb"      "sex"          "age"
##  [6] "topqual3"     "econact"      "nssec8"       "eth_origin"   "eqvinc"
## [11] "htval"        "wtval"        "omsysval"     "omdiaval"     "porfv"
## [16] "cigs"         "health"       "marital"      "employed_lgc"
```

Let's say we only wanted descriptive statistics for the `topqual3` and `econact` variables. We could achieve this by first running a `select()` function that retains only these variables before passing the result through to the `summary()` function.

Remember that piping the result through to an additional function is achieved using the `%>%` (pipe) operator (part of `tidyverse`), and that the result in a pipe is stored in a `.`.

Example

Remember that we will have needed to load the tidyverse library before we can use any of the functions from it, but we did this at the start of the document when we read in the data.

We can use select and pipes to get summary statistics using `summary()` statistics for only a few variables using the following code (you will need to either run the entire chunk using the green arrow, run the code from the first line using ctrl/cmd + enter, or highlight all of the code and press ctrl/cmd + enter to run everything in the pipeline):

```
hse_data %>% # Pipe the data through to the next argument
  select(topqual3, econact) %>% # Note variables are without quotation marks
  summary(.) # Run summary with the result (explicitly stated with the full stop)
```

```
##                       topqual3                           econact
##  No qualification         :2009   In employment              :4616
##  NVQ4/NVQ5/Degree or equiv:2008   Retired                    :2265
##  NVQ2/GCE O Level equiv   :1744   Other economically inactive:1204
##  NVQ3/GCE A Level equiv   :1238   ILO unemployed             : 376
##  Higher ed below degree   : 945   Refused                    :   0
##  (Other)                  : 516   (Other)                    :   0
##  NA's                     :  43   NA's                       :  42
```

Exercise

Using the above code as a template, write some valid R code that selects only the variables `eqvinc` (equivalised income), `porfv` (portions of fruit and vegetables eaten in the previous dat), and `marital` (marital status) — in that order — from the `hse_data` dataset and summarises them using descriptive statistics produced using the `summary()` function, using pipes as above.

```
hse_data %>%
  select(eqvinc, porfv, marital) %>%
  summary(.)
```

```
##      eqvinc             porfv                  marital
##  Min.   :   271.1   Min.   : 0.000   0. Not Married:4000
##  1st Qu.: 14300.0   1st Qu.: 2.000   1. Married    :4501
##  Median : 23442.6   Median : 3.333   NA's          :   2
##  Mean   : 33274.1   Mean   : 3.656
##  3rd Qu.: 43624.2   3rd Qu.: 5.000
##  Max.   :262295.1   Max.   :30.000
##  NA's   :1781       NA's   :10
```

Using `janitor`'s `tabyl()` for better categorical/ordinal variable summaries

You may have noticed that when you use `summary()` for descriptive statistics for categorical and ordinal variables you only receive the frequency counts and not the percentage of each response. This can be quite unhelpful for getting a sense of the distribution across responses in your data.

One package that adds a range of convenience functions to R is the `janitor` package.

Exercise

Write the code required to load the `janitor` package from the library in R. Remember, you will need to have installed `janitor` first using `install.packages("janitor")` in the console.

```
library(janitor)
```

```
##
## Attaching package: 'janitor'
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

Example

Use the `tabyl()` function in `janitor` to create a more detailed set of summary statistics including percentages for the `health` variable in the `hse_data` dataset.

```
hse_data %>%
  tabyl(health)
```

```
##             health    n      percent valid_percent
##    1. Bad/Very Bad  620 0.0729154416    0.07297552
##            2. Fair 1588 0.1867576150    0.18691149
##  3. Good/Very Good 6288 0.7395037046    0.74011299
##               <NA>    7 0.0008232389            NA
```

The `tabyl()` function produces four columns: the first column shows all of the unique values for the variable in the tabyl; the second column (`n`) shows the frequency counts for each response; the third — somewhat incorrectly named — column (`percent`) shows the proportion of responses that fall into each value, *including* missing (`NA`) values; the final column (`valid_percent`) shows the proportion of responses across all non-missing values.

A proportion can be converted into a percentage by multiplying it by 100 — e.g. the proportion of people saying that have bad or very bad health is 0.0729 but the percentage of people saying they have bad or very bad health is 7.29%. Alternative, you can add `adorn_pct_formatting()` to the janitor tabyl pipe.

```
hse_data %>%
  tabyl(health) %>%
  adorn_pct_formatting()
```

```
##             health    n percent valid_percent
##    1. Bad/Very Bad  620    7.3%          7.3%
##            2. Fair 1588   18.7%         18.7%
##  3. Good/Very Good 6288   74.0%         74.0%
##               <NA>    7    0.1%             -
```

Exercise

Use the `tabyl()` function to create univariate descriptive statistics for the `cigs` variable in the `hse_data` dataset. Use a pipe, as above, to pass the data to the `tabyl()` function.

```
hse_data %>%
  tabyl(cigs)
```

```
##                 cigs    n    percent valid_percent
##        1. Non-Smoker 6743 0.79301423    0.79978650
##    2. Under 10 a Day  607 0.07138657    0.07199620
##    3. 10 to 20 a Day  712 0.08373515    0.08445024
##  4. 20 or More a Day  369 0.04339645    0.04376705
##                 <NA>   72 0.00846760            NA
```

What percentage of people, excluding NAs from the sample, were non-smokers in the sample? Write your answer below.

–

80%

–

Using `skimr`'s `skim()` for better and multiple continuous variable summaries

Another useful package for summary statistics is `skimr` — it contains a function called `skim_without_charts()` which returns summary statistics for all variables.

The reason we use `skim_without_charts()` instead of `skim()` is that `skim()` produces mini-histograms using special characters - these special characters will cause R markdown knits to fail so it's generally better not to use them in Rmarkdown. You can always test out what `skim()` looks like using the console below.

Exercise: load the package `skimr`

```
library(skimr)
```

Example

Now that `skimr` is loaded we can use the `skim_without_charts()` function to get descriptive statistics for variables. We will continue using pipes (`%>%`) here for consistency.

```
hse_data %>%
  skim_without_charts(eqvinc, porfv)
```

Table 1: Data summary

Name	Piped data
Number of rows	8503
Number of columns	19
_____	
Column type frequency:	
numeric	2
_____	
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
eqvinc	1781	0.79	33274.13	30347.75	271.08	14300	23442.62	43624.16	262295.1
porfv	10	1.00	3.66	2.61	0.00	2	3.33	5.00	30.0

`skim_without_charts()` gives us some additional statistics that `summary()` does not provide. It also makes it easier to read descriptive statistics across a large number of variables at once. `skim_without_charts()` provides us with:

- `skim_variable` - the name of the variable being skimmed
- `n_missing` - the number of observations with missing data for this variable
- `complete_rate` - the proportion observations in the data *with* valid values for this variable
- `mean` - the arithmetic mean for the variable
- `sd` - the standard deviation for the variable
- `p0` - the "zeroth-percentile", in other words, the minimum value in the variable
- `p25` - the 25th percentile value
- `p50` - the 50th percentile value, which is equal to the median
- `p75` - the 75th percentile value
- `p100` - the "100th-percentile" value, in other others, the maximum value in the variable

Exercise

Try running the `skim_without_charts()` function on the variables `age`, `wtval` (weight in kg), and `eth_origin` (ethnic origin).

```
hse_data %>%
  skim_without_charts(age, wtval, eth_origin)
```

Table 3: Data summary

Name	Piped data
Number of rows	8503
Number of columns	19
Column type frequency:	
factor	1
numeric	2
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
eth_origin	40	1	FALSE	18	Whi: 7094, Any: 414, Ind: 203, Pak: 138

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
age	0	1.00	50.07	18.43	17.0	35.0	49.0	64.00	100.0
wtval	1392	0.84	77.51	17.27	35.9	65.1	75.9	87.35	184.3

Notice how `skim_without_charts()` can sometimes be less useful than `summary()` or `tabyl()` for categorical/ordinal variables like `eth_origin`, especially those with large numbers of categories.

Creating your own custom summaries using dplyr's `summarise()`

So, finally, what if none of `summary()`, `skim_without_charts()`, or `tabyl()` do what you want them to do? What if you want summaries of a variable with the mean, median, mode, and a 90% quantile? We can create our own summary statistics using `dplyr`'s `summarise()` function (or `summarize()` if you prefer international English).

`summarise()` requires manual specification of everything you wish it to calculate. It works similar to other functions in `dplyr`, including `mutate()`, but `mutate()` does not change the shape of the dataset, it only adds new columns. We will use `mutate()` in the future.

Example

The following code summarises the variable `eqvinc` with its mean, mode, median, and standard deviation using the `summarise()` function. Notice how all of the functions we've been using above are on the right hand side of the equals sign in each of the summarise arguments, with the name of the result written on the left hand side of the equals sign (e.g. the resulf of the calculation for the mean is assigned *within* the `summarise()` argument to the name `mean_eqvinc` — any name could have been chosen).

```
hse_data %>%
  summarise(
    mean_eqvinc = mean(eqvinc, na.rm = TRUE),
    mode_eqvinc = mfv(eqvinc, na.rm = TRUE),
    median_eqvinc = median(eqvinc, na.rm = TRUE),
    sd_eqvinc = sd(eqvinc, na.rm = TRUE)
  )
```

```
## argument 'na.rm' is soft-deprecated, please start using 'na_rm' instead
```

```
## # A tibble: 1 x 4
##   mean_eqvinc mode_eqvinc median_eqvinc sd_eqvinc
##         <dbl>       <dbl>         <dbl>     <dbl>
## 1      33274.      10656.        23443.    30348.
```

Exercise

Try adding a new column to the `summarise()` function (a new argument, below the line `sd_eqvinc = sd(eqvinc, na.rm = TRUE)`) that adds the `class()` of the variable `eqvinc`. Name this new summary variable `class_eqvinc`.

```
hse_data %>%
  summarise(
    mean_eqvinc = mean(eqvinc, na.rm = TRUE),
    mode_eqvinc = mfv(eqvinc, na.rm = TRUE),
    median_eqvinc = median(eqvinc, na.rm = TRUE),
    sd_eqvinc = sd(eqvinc, na.rm = TRUE),
    class_eqvinc = class(eqvinc)
  )
```

```
## argument 'na.rm' is soft-deprecated, please start using 'na_rm' instead
```

```
## # A tibble: 1 x 5
##   mean_eqvinc mode_eqvinc median_eqvinc sd_eqvinc class_eqvinc
##         <dbl>       <dbl>         <dbl>     <dbl> <chr>
## 1      33274.      10656.        23443.    30348. numeric
```

A note on packages

Okay, that's enough different packages for summary statistics. Now is a good time to pause and address all of the different ways of doing things in `R` and the many, many different packages and functions that can be used to reach the same results. This can feel overwhelming.

The key thing to remember is that there is no wrong way to reach some end goal in `R`, just lots of different ways of going about it. You don't need to know all of the different ways — just the ones that work best for you. The goal here is to introduce you to a small range of different ways to reach the same important statistical analyses in social science projects, not to have you retain every different way. You **will** forget how to do things, particularly if you've not done them in a while. This is natural and everyone who uses `R` — from professionals who created the thing to people just starting out — needs to Google how to do things from time to time, especially simple things! Do not be afraid to consult Google.

However, having some exposure to different ways of doing things in `R` can lead to some unintended benefits later down the line. For example, functions like `summarise()` can be a lot more useful than functions like `summary()`, `tabyl()`, or `skim()` when you are trying to aggregate a dataset up to a higher level (for example, if you have a dataset of millions of pupils and want to create a dataset for school results). However, if you had no awareness of tools like `summarise()` it might take you a long time to find out where to start to achieve this end goal.

So, basically — use as many packages and as many different functions as you wish. You should feel free to seek out new packages and functions that work better for you, don't be restricted to the packages we use in the class (we will, as your module leaders, generally, be able to figure out how things that are new to us work and how to troubleshoot 99% of issues you run into with them). Very often you will be introduced to new packages and functions when trying to troubleshoot error messages or search for how to achieve something in `R` using Google. There is nothing wrong with using packages and functions outside of the scope of the module, as long as they reach the same end goal — you will not be penalised in any way for doing this in your assessments.

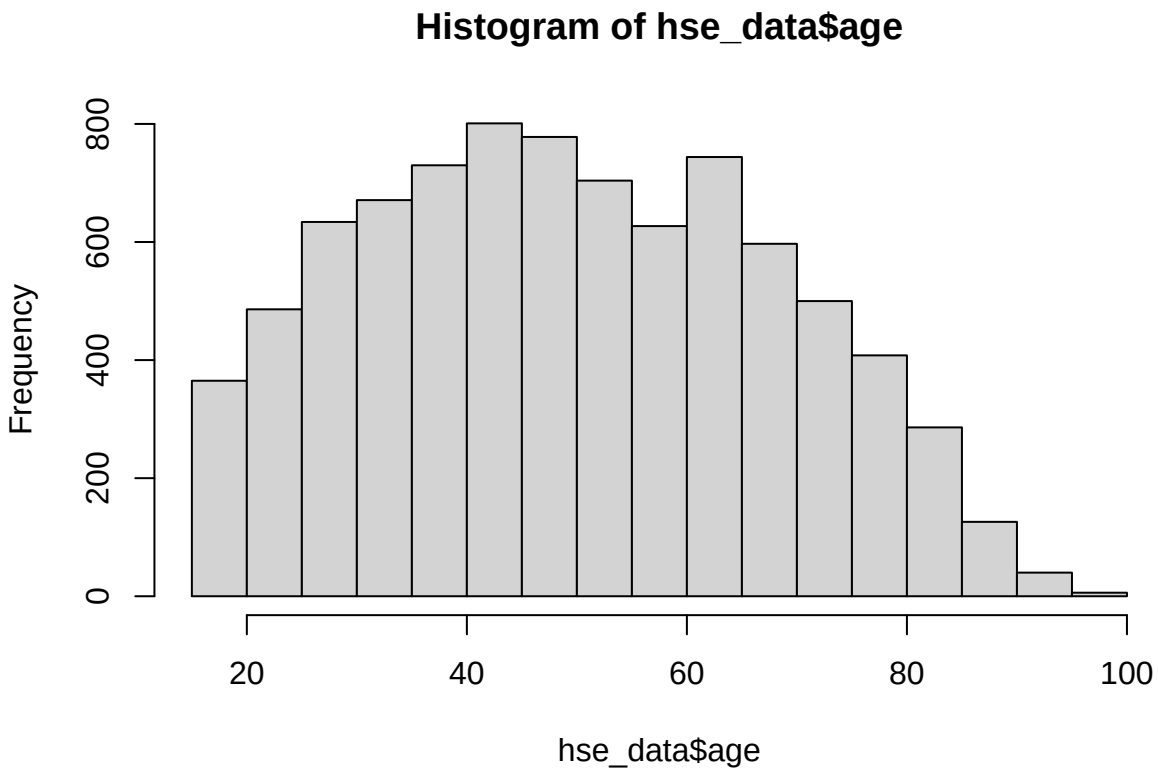
Okay, now let's wrap up with some exercises for plotting univariate data.

Plots for describing data: Base `R` Histogram

You can quickly produce a *histogram* with `R` by using the built in `hist()` function.

Example

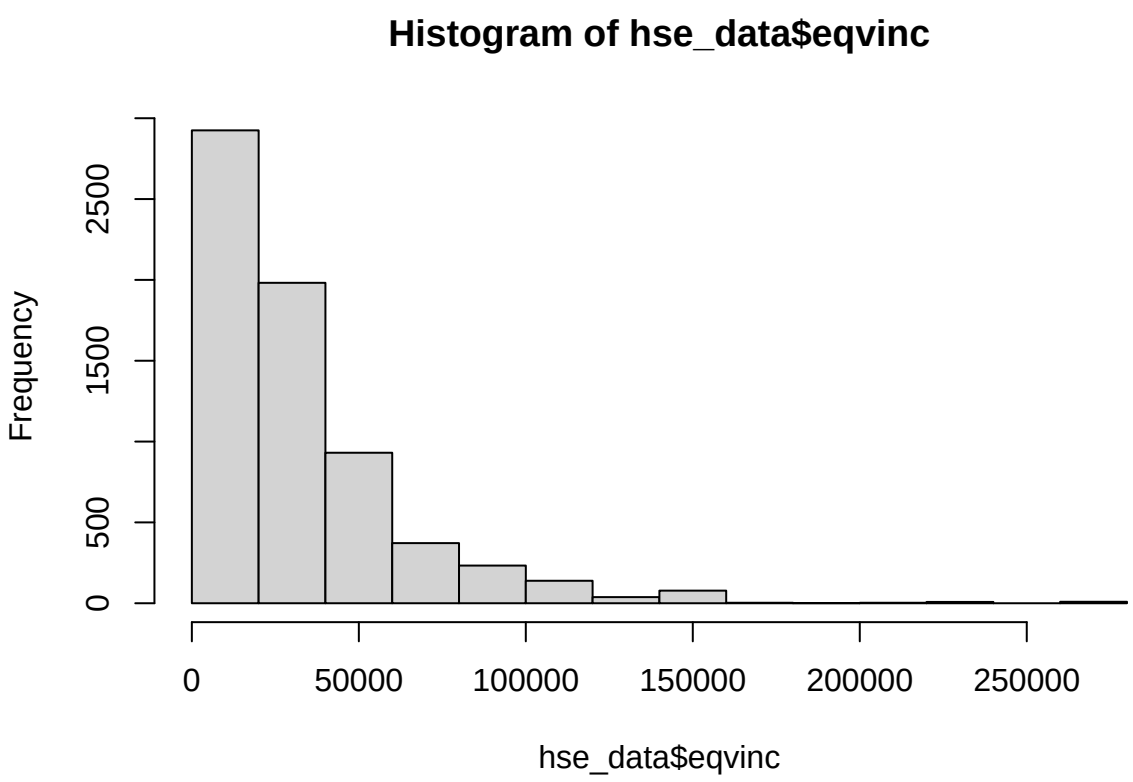
```
hist(hse_data$age)
```



Exercise

Use the `hist()` function to plot a histogram of the `eqvinc` variable.

```
hist(hse_data$eqvinc)
```



Plots for describing data: `ggplot2` Histogram

While `hist()` can be very helpful for quickly checking the distribution of a variable, base `R` plotting functions can be difficult to customise and make visually appealing.

A commonly used alternative to base `R` plotting functions is the `ggplot2` package, which is included in the `tidyverse`. `ggplot2` uses an entire 'grammar of graphics' to make it possible to create an practically infinite range of data-driven visualisations. The topic of plotting using `ggplot2` is too large to cover in detail in this module, but a full book published by its creator is available for free online.

Example

```
ggplot(data = hse_data)  # Initialise ggplot2
```

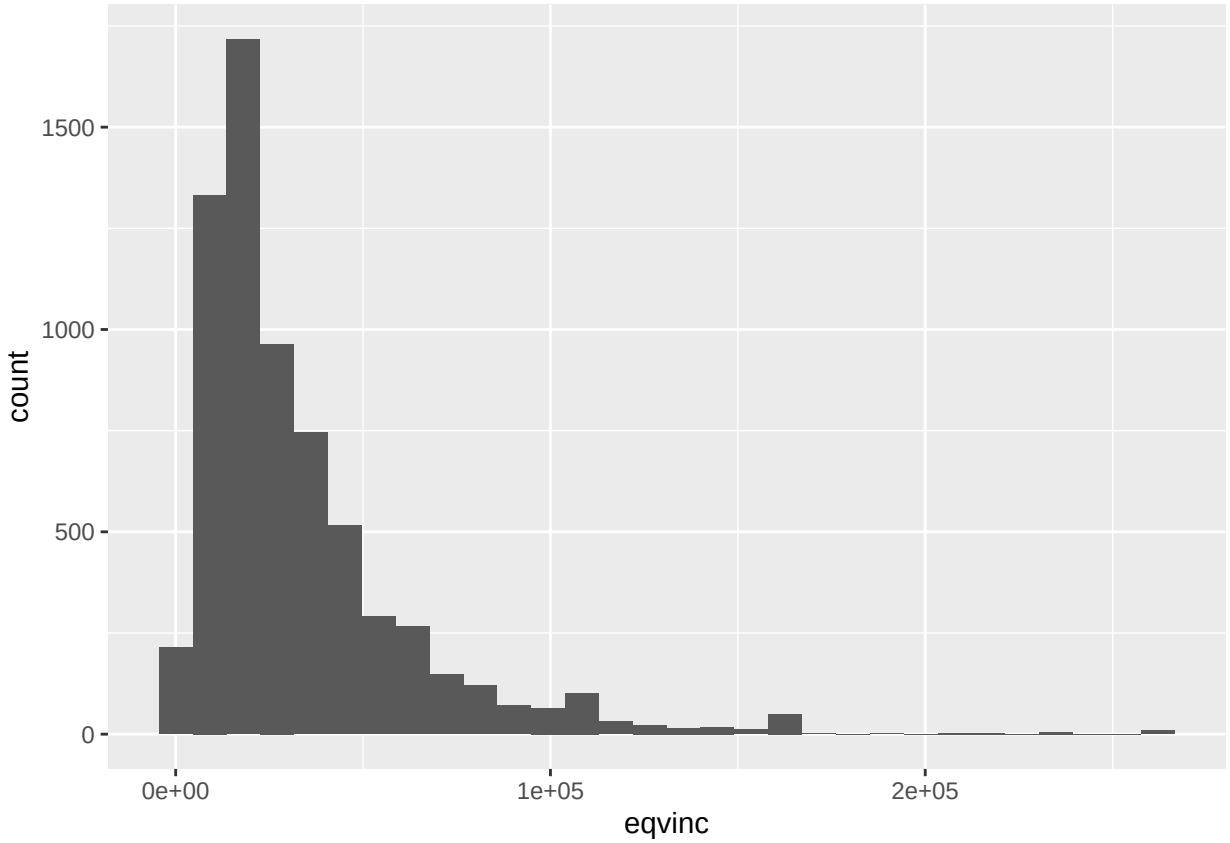
There is no point running this code yet as we have not entered any arguments for data, but there is already quite a lot going on. Note that we initialise `ggplot2` first using the `ggplot()` function. This is kind of like putting your keys in the ignition of the car and starting the engine: we haven't decided where we're going get or put our car into gear, but we are telling R to get ready for our instructions. Any arguments in the `ggplot()` function are included throughout all following arguments as 'collective geoms'. For example, in `ggplot()` I have included the argument `data = hse_data`, which will mean that we will use this dataset in all of the geoms that follow.

Now let's add our arguments for `geom_histogram()`. Rather than making assumptions about what kind of data visualisation is appropriate, `ggplot2` requires you to specify which `geom_` you wish to use. This is because it allows you to layer multiple geoms on top of one another. The `geom_` function for a histogram is `geom_histogram()`, so we would start our data visualisation code with the following:

```
ggplot(data = hse_data) + # Initialise ggplot2
  geom_histogram(aes(x = eqvinc))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 1781 rows containing non-finite values (`stat_bin()`).
```



One thing to note is that instead of using the pipe we have used before (`%>%`), we chain together layers of the `ggplot` function using a plus sign (`+`). Using a pipe instead of a plus is a common error when using `ggplot` (one that I make all the time!), but it generally has a pretty instructive error message.

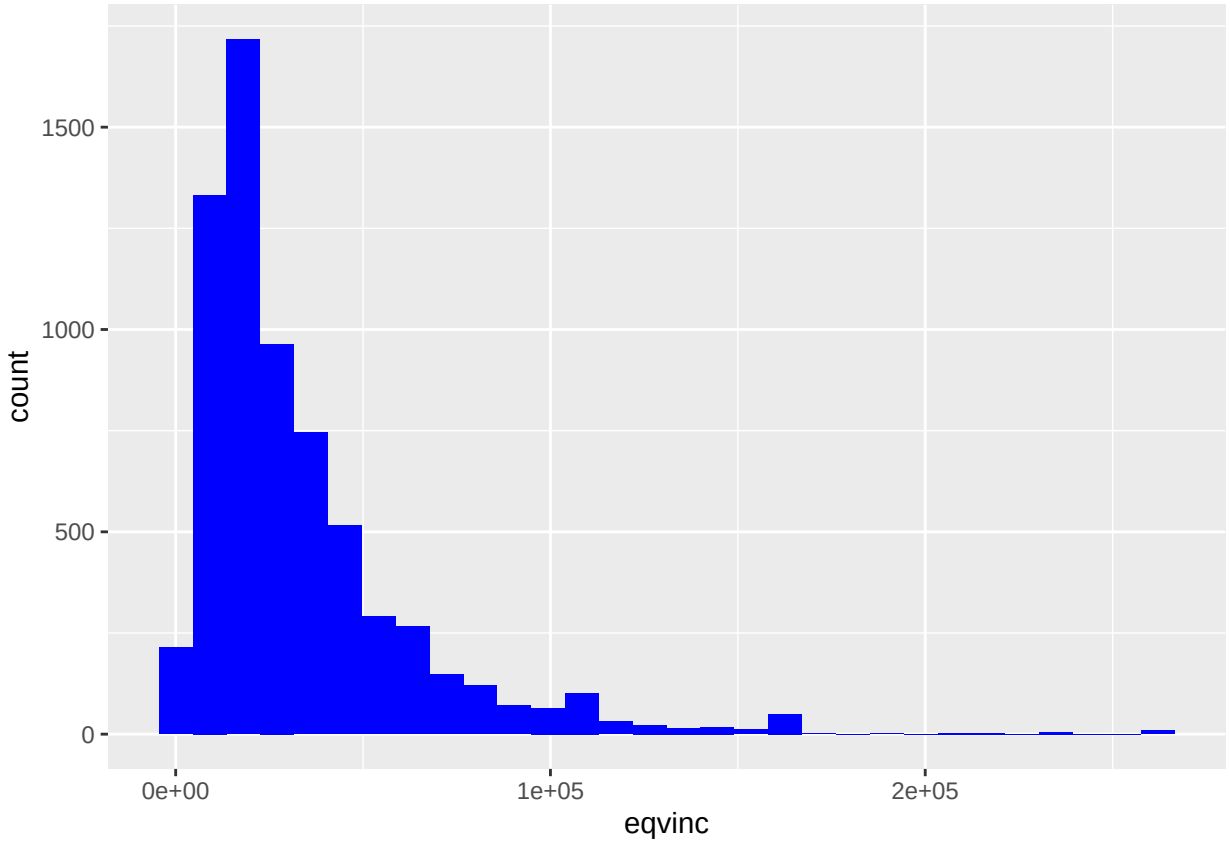
Notice that the variable we are visualising is wrapped within an `aes()` function. `aes()` is used to delineate between fixed (things that are constant and do not differ) and variable (things that differ) parts of the appearance of the plot. Because we are interested in visualising something that varies — a variable, in this case income — we need to include this within the `aes()` function.

The best way to understand this is probably if we add something to the 'fixed' part of the `geom_histogram()` function for contrast — let's change the colour that each bar of the histogram is filled in with to be blue.

```
ggplot(data = hse_data) + # Initialise ggplot2
  geom_histogram(aes(x = eqvinc), fill = "blue")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 1781 rows containing non-finite values (`stat_bin()`).
```



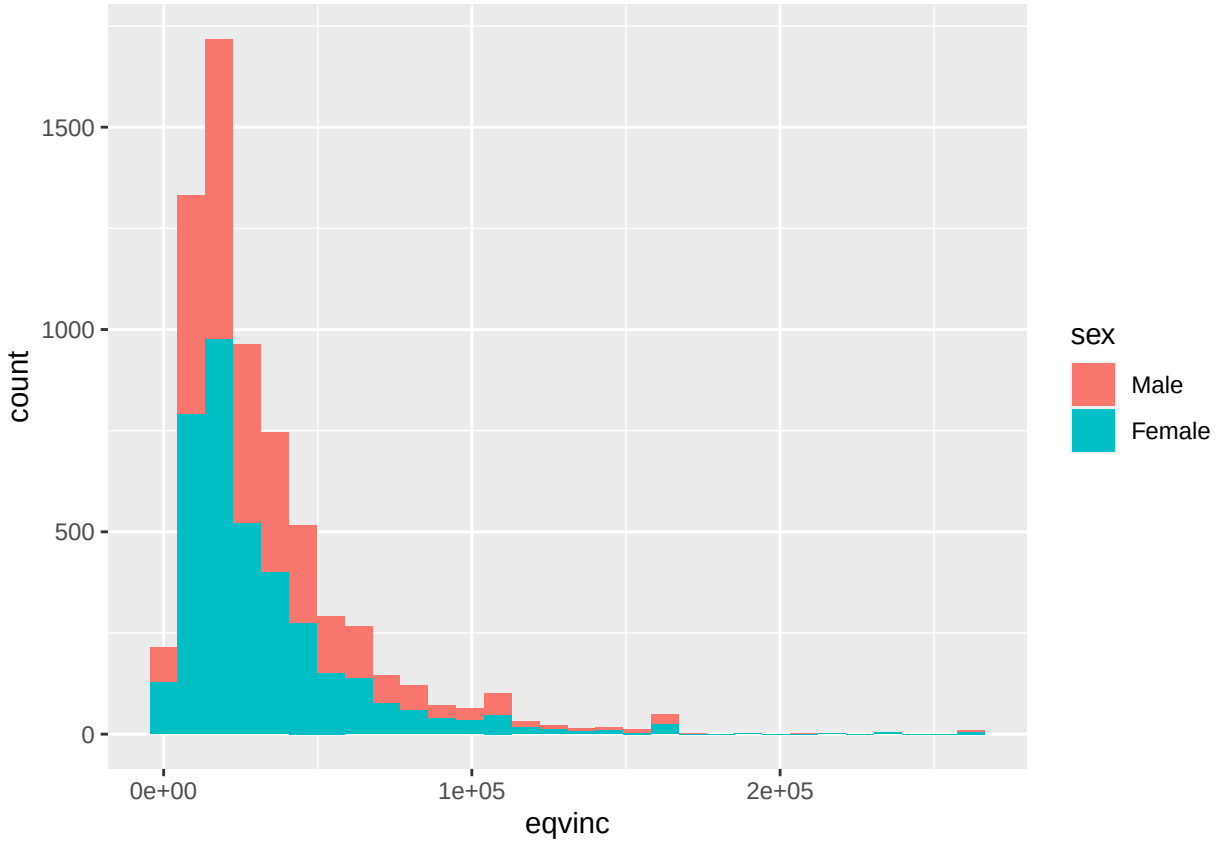
Because the `fill` argument is outside of the `aes()` function's brackets it means we are telling ggplot we want everything to follow the same rule here — not for it to be variable based on something that changes in our data. In this case, we're telling it to make all of the bins in the histogram blue.

Alternatively, we could include the `fill` argument *within* the `aes` function to make it variable based on some other variable *in* the dataset. For example, if we wanted to fill the histogram bars based on the `sex` of the respondents in each bin (bar), we could move the fill argument *inside* the aes() function and make it equal the `sex` variable:

```
ggplot(data = hse_data) + # Initialise ggplot2
  geom_histogram(aes(x = eqvinc, fill = sex))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 1781 rows containing non-finite values (`stat_bin()`).
```



Incorrect placement of 'static/fixed' and 'dynamic/variable' aesthetic mappings is a frequent source of errors in ggplot, so it's worth remembering that basic rule.

By adding this additional variable we can incorporate more information into our plot — in this case, we see how much of each bar is made up by women and how much is made up by men (though - keep in mind - this dataset may not necessarily be the best for exploring this question, especially if household income is used rather than individual income).

There are a wide range of aesthetic arguments that we can map to fixed or variable data. Some commonly used ones include:

- `size` = the size of points or lines
- `fill` = the filled in colour of a shape
- `colour` = usually, the border colour of a shape, or the fill when `fill` is not specified
- `alpha` = the transparency of the shapes in the geom, 1 = no transparency, 0 = complete transparency (not visible)
- `pch` = the shape of points (e.g. circles, triangles, boxes)

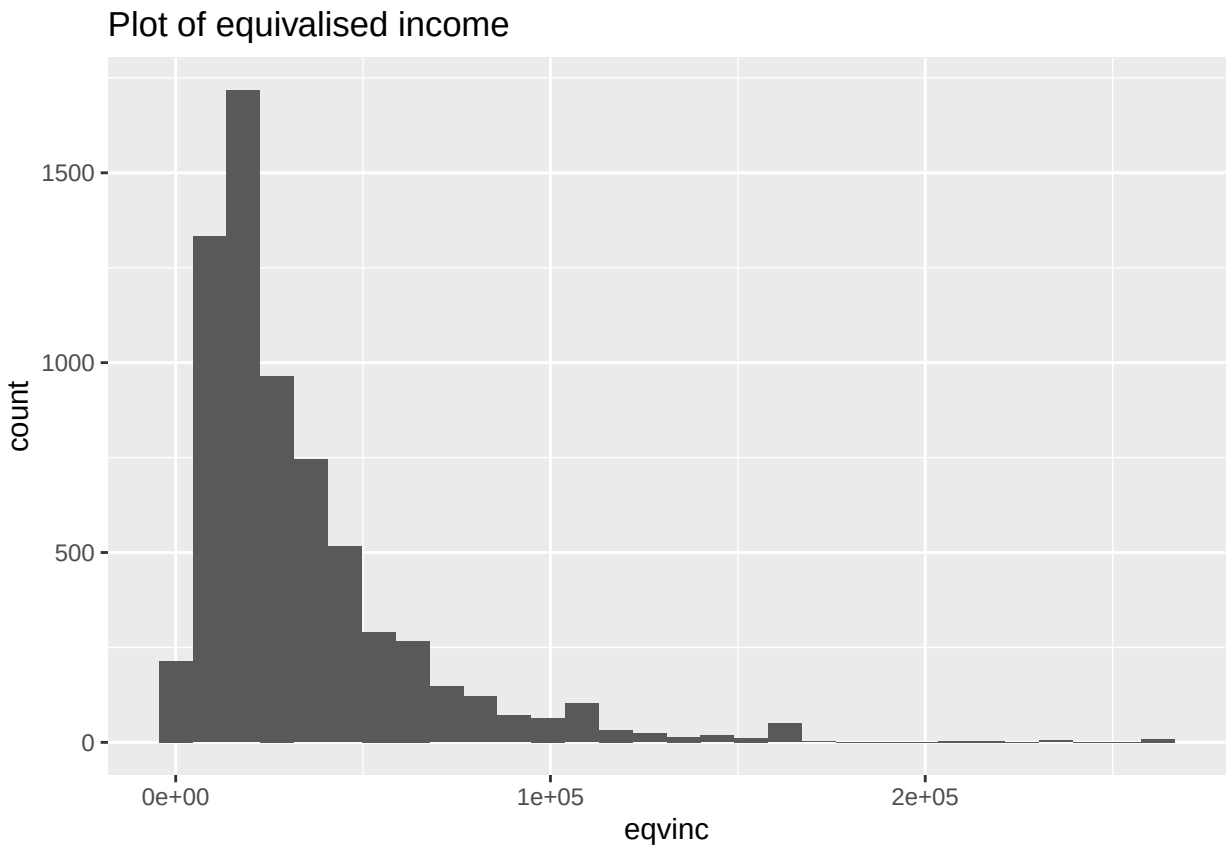
It can be a little overwheming compared to the simplicity of `hist()`, but ggplot is a skill that can be developed over time and will save you a lot of time in the future compared to trying to manipulate base R's plotting functions to do exactly what you want.

Lastly, we can change other parts of the plot, like through adding a title with the `ggtitle()` function.

```
ggplot(data = hse_data) + # Initialise ggplot2
  geom_histogram(aes(x = eqvinc)) +
  ggtitle("Plot of equivalised income")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 1781 rows containing non-finite values (`stat_bin()`).
```



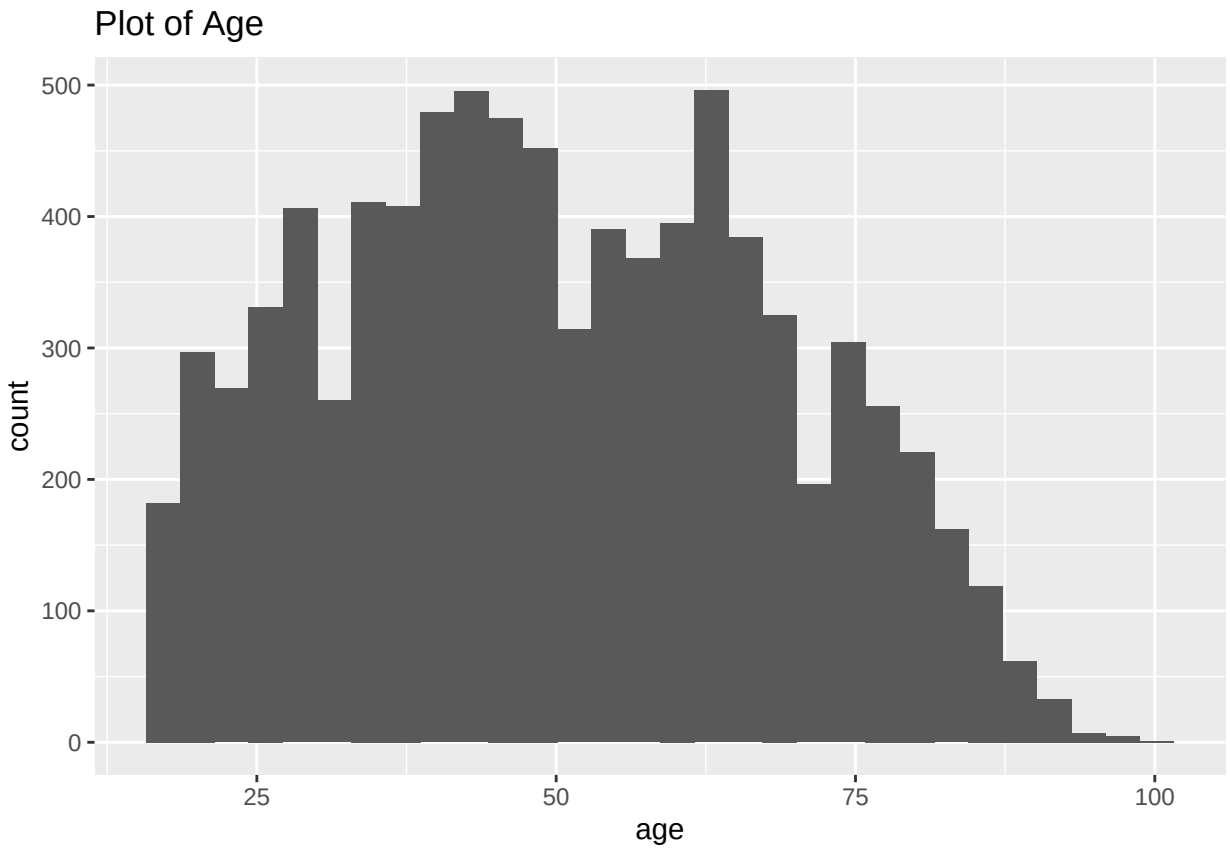
I would recommend you spend some time with either the ggplot2 book (https://ggplot2-book.org) or with the R for Data Science book (https://r4ds.had.co.nz) if you want to learn more about ggplot.

The best way to learn `ggplot2` is through practice - experimenting with many different kinds of visualisation using different forms of data — for example by taking part in TidyTuesday. However, once you have mastered it you can create unique and interesting visualisations very quickly.

Try creating a histogram of the `age` or `porfv` variables before moving onto the next section.

```
ggplot(data = hse_data) + # Initialise ggplot2
  geom_histogram(aes(x = age)) +
  ggtitle("Plot of Age")
```

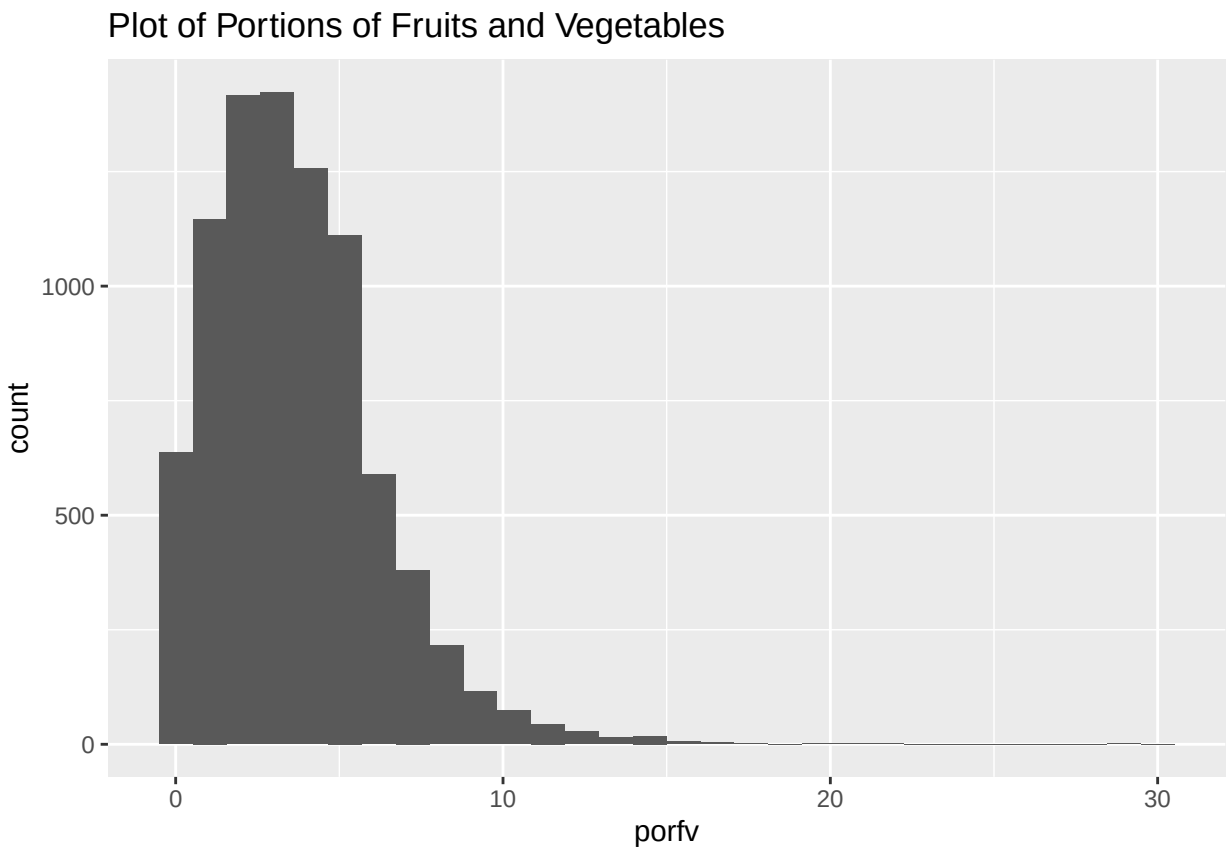
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(data = hse_data) + # Initialise ggplot2
  geom_histogram(aes(x = porfv)) +
  ggtitle("Plot of Portions of Fruits and Vegetables")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 10 rows containing non-finite values (`stat_bin()`).
```



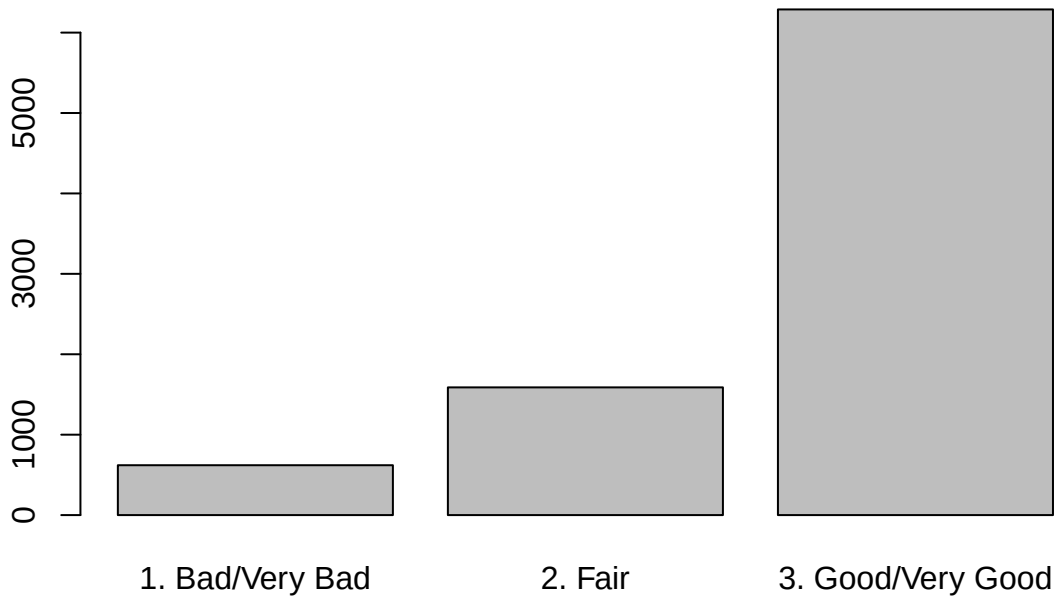
Plots for describing data: Base R Bar Chart

By default, base R's `plot()` function uses the context of the class of the variable you give it to create a suitable data visualisation (interestingly, the default for numeric variables is a scatterplot, not a histogram, hence why we had to call `hist()` before).

Example

Using `plot()` to create a bar chart for the self-reported `health` of participants.

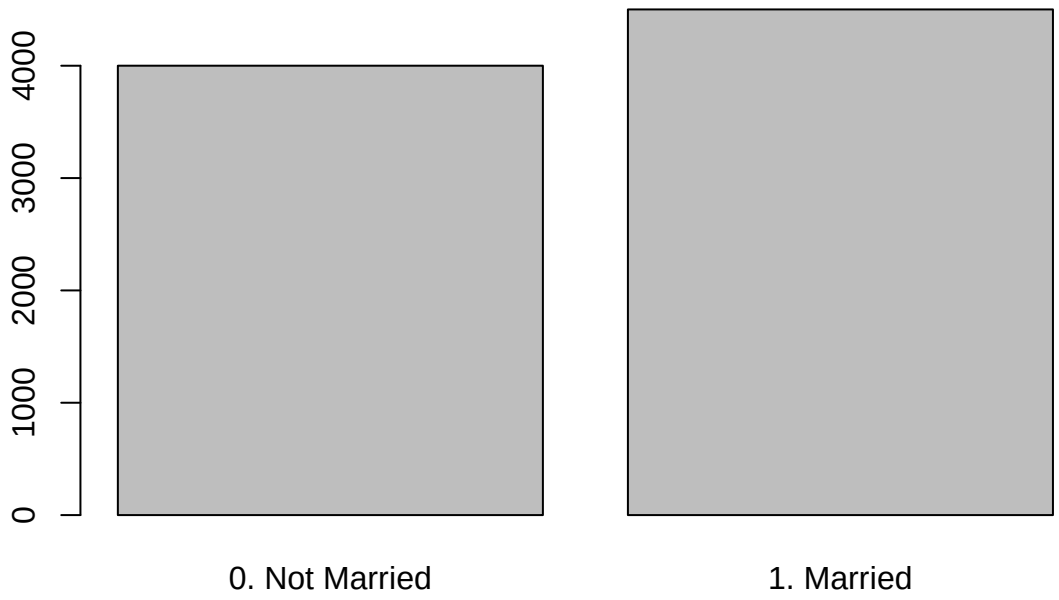
```
plot(hse_data$health)
```



Exercise

Use the `plot()` function to create a bar chart plot for the `marital` status of respondents.

```
plot(hse_data$marital)
```

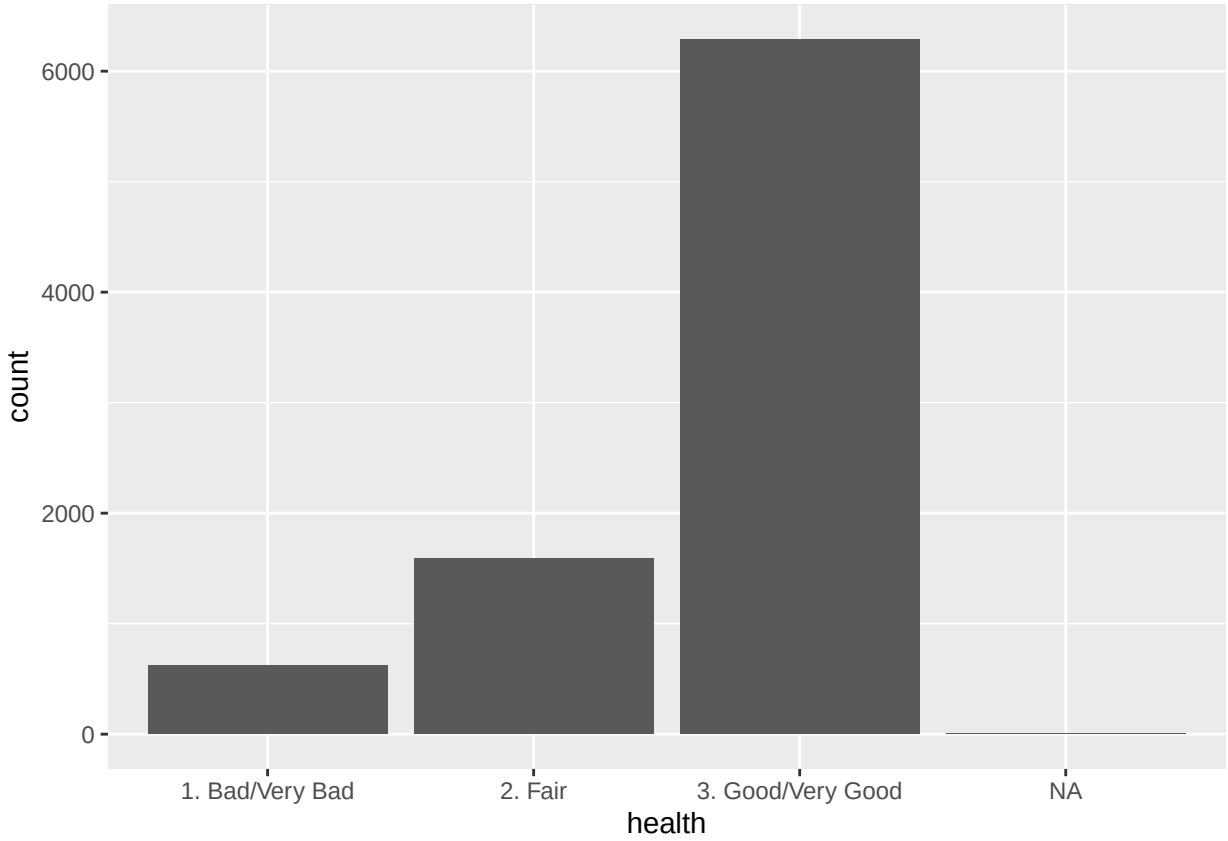


Plots for describing data: `ggplot2` Bar Chart

Finally, we can learn a second geom from the `ggplot2` package for plotting the same bar charts. Intuitively, the geom for plotting bar charts in `ggplot2` is called `geom_bar`.

Example

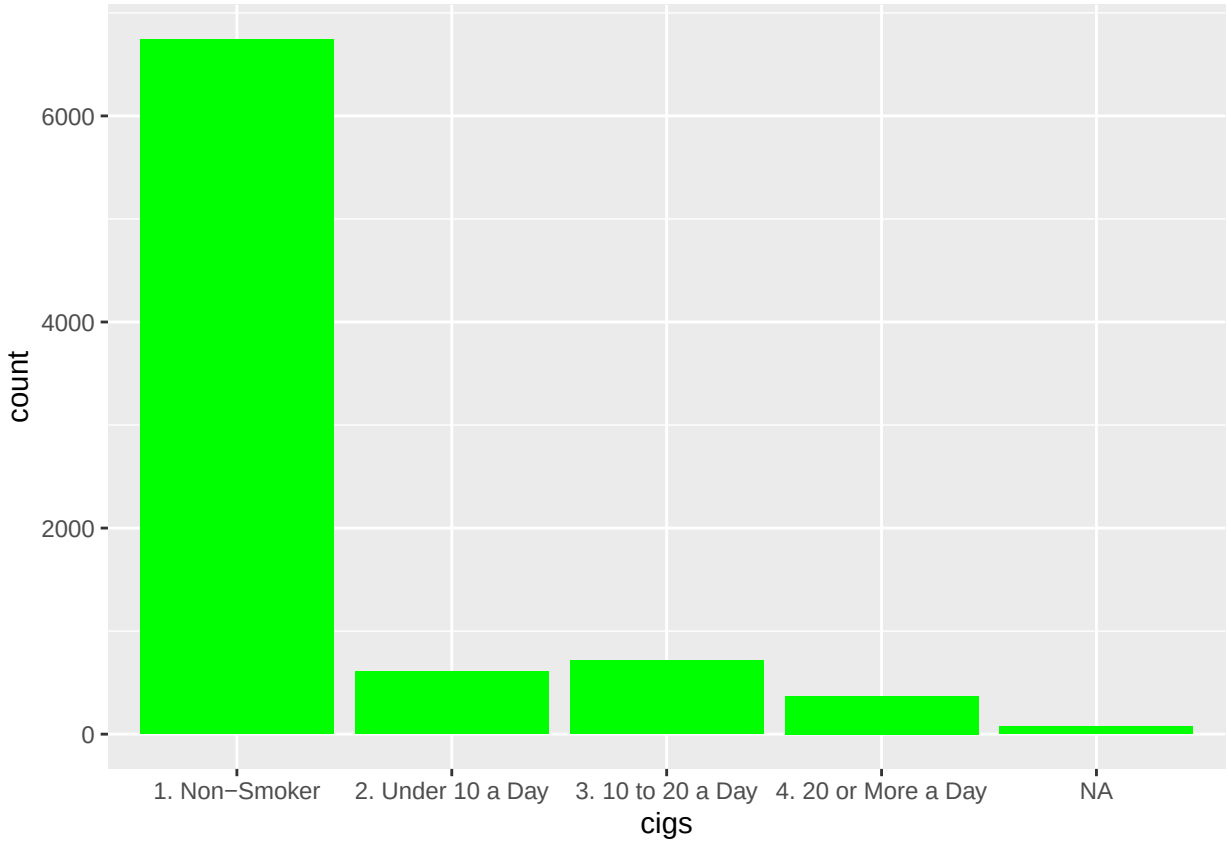
```
ggplot(data = hse_data) +
  geom_bar(aes(x = health))
```



Exercise

Try creating a bar chart for the variable `cigs` and making the `fill` colour for the bars "green".

```
ggplot(data = hse_data) +
  geom_bar(aes(x = cigs), fill = "green")
```



End

Well done for making it through this tutorial! Hopefully you are starting to feel a little more familiar with Rs syntax and have had some experience dealing with error messages.

Being able to describe and visualise variables in our data is an essential skill in quantitative research methods. It is important for identifying outliers or unexpected responses; for checking what the distribution looks like; and for comparing our sample to what we know about the wider population and samples from other studies.

It is common for the first table in a quantitative social research paper to be a table of univariate descriptive statistics for all variables used in an analysis. What we find in our descriptive and summary statistics can help us avoid breaking the assumptions of various tests or models we may wish to run (for example, we would know not to trust the 68, 95, 99.7 rule for standard deviation if we knew our variable was very heavily skewed). Therefore it is an essential part of being an effective and responsible quantitative social science researcher.

We have covered a lot today, as we also did last week! It may feel like a trial by fire but you will be surprised at how far you have come in R by December. At the moment we are covering a breadth of quantitative research skills with a steep learning curve, but after a few intensive weeks we'll be moving more towards covering specific modelling methods in depth, so keep up the good work!