# The Missing Book

Nicholas Tierney & Allison Horst

2022-04-07

# Table of contents

# Preface

## Welcome

Welcome to The Missing (Data) Book! Through this book you will learn concepts and tools to explore, consider, and deal with missing values in your data.

### What you will learn

After reading and completing the exercises in this book, you will be able to answer the following questions and apply them to your own data:

- What are missing values, and why do we care about them?
- How can I find and explore missing values in data?
- How can I wrangle and tidy missing data?
- How can I investigate why values are missing?
- How can I impute missing values?

### Prerequisites

For this course we assume you have:

- Basic to intermediate experience with R
- Experience creating plots using `ggplot2`
- Experience using `dplyr` to manipulate data
- Basic experience fitting linear models in R

## Narrative story / example

### Why care about missing data?

> The best thing to do with missing data is to not have any
>
> **–Gertrude Mary Cox**

As true as what Statistician Gertrude Mary Cox said, it is not the world we live in. Working with data means working with missing data. To be a great analyst you need to know how to deal with missing values.

Well, why should we care about missing data? Understanding how missing data work is important as they can have unexpected effects on your analysis. For example, fitting a linear model on data with missing values deletes chunks of data. This means your decisions aren't based on the right evidence. Simiarly, we need to take care when we replace missing values, a process called *imputation*. Imputation has to be done very carefully. If we insert the wrong values, we can end up with poor estimates and decisions. Imagine substituting salt for sugar in a cake - the result is disastrous!

## How to read this book

We have broken this book into 7 parts. Most of these parts each have accompanying exercises for you to complete online. These seven sections are:

1. Introduction to Missing Data
2. Missing Data Gotchya's
3. Explore Missing Values
4. Cleaning Missing Data
5. Representing Missing Data
6. Mechanisms of Missingness
7. Single Imputation of Missing Data

The book has been designed to be read in this order, as we build upon material in each section. And while seven sections might sound like a lot, these sections are all quite short!

# Part I

# Introduction to missing data

# 1 Introduction to missing data

## 1.1 What are missing values?

First, we need to define missing values:

> Missing values are values that **should have been recorded but were not**.

Consider these two examples where you are out counting birds in an area:

1. You see a bird, but forget to record the observation and leave the value blank.
2. You do not record any bird sitings, and record a 0 value.

The first of these is an example of a **missing value** - the value was intended to be recorded but was not. The second is a record of the absence of birds.

In other words: if you did not see any birds, you *should* have entered a value indicating no birds were seen. Because there is *no record* where there should have been one, it is a missing value.

How do we note when a value is missing? There are many ways it might be recorded, depending on a variety of factors, such a the standards in a given field, or the way that the data was collected. Here are a few examples:

- blank records (e.g. empty cells in a spreadsheet)
- Consistently recorded indicators for missing values, such as "NA", "N/A", or "-9999", throughout the data
- A combination of *different* values meant to indicate a missing value (for example, if a team of researchers are recording urchin diameters, Researcher A might enter "-9999" for a missing measurement, while Researcher B enters "N/A")

You can imagine how chaotic this might get when we have many different types of missing value all recorded together. Imagine a dataset like this:

| bird | count | researcher |
|---|---|---|
| kookaburra | NA | A |
| kookaburra | 0 | B |
| crow | NA | A |

| bird | count | researcher |
|---|---|---|
| crow | 1 | B |
| pigeon | -999 | A |
| pigeon | -9999 | B |

To simplify things, we will start by exploring cleaned up missing values - those stored as `NA`, which is R's standard way of representing missing values. Transforming the chaos above, this is how it would be represented if the missing values appropriately.

| bird | count | researcher |
|---|---|---|
| kookaburra | NA | A |
| kookaburra | 0 | B |
| crow | NA | A |
| crow | 1 | B |
| pigeon | NA | A |
| pigeon | NA | B |

To help explore and understand missing values, we'll be using the **`naniar`** package, which provides many helpers to make it easier to explore, understand, and visualise missing values.

## 1.2 How does R deal with missing values?

Before we start exploring missingness, we need to understand how R interprets and processes missing values. R stores missing values as `NA`, which stands for **N**ot **A**vailable. R deals with `NA`s in unique, and sometimes unexpected, ways.

### 1.2.1 Missing values in basic R operations

What happens when we mix missing values (`NA`) with our calculations? We need to know how R deals with missing values in operations so we can recognize these cases and deal with them appropriately.

The general rule for `NA`s in R calculations is:

> Calculations with `NA` return `NA`.

Several outcomes for common operations that include `NA` are:

- `NA` + [anything] = `NA`

- NA - [anything] = NA
- NA * [anything] = NA
- NA / [anything] = NA
- NA == [anything*] = NA

For example, suppose we have a `heights` dataset containing the heights of four friends (Sophie, Dan, Fred, and Liz):

```
heights <- tibble::tibble(
  name = c("Sophie", "Dan", "Fred", "Liz"),
  height = c(163, 175, NA, NA)
)

heights
```

```
# A tibble: 4 x 2
  name   height
  <chr>   <dbl>
1 Sophie    163
2 Dan       175
3 Fred       NA
4 Liz        NA
```

The sum of the `height` variable returns `NA`:

```
sum(heights$height)
```

```
[1] NA
```

This is because we cannot know the sum of a number and a missing value. Similarly, if we try to find the mean height, `NA` is returned:

```
mean(heights$height)
```

```
[1] NA
```

When an operation on data containing an `NA` returns an `NA`, it tells us the missing values are *not* being ignored in the calculation, reflecting the default argument `na.rm = FALSE` (read: "Remove NAs? No!") in many functions.

> Always check the default `NA` action (e.g. `na.rm = FALSE`) for functions. As we will see later, the **default in some functions is to remove `NA`** - sometimes without warning.

Can we override the default `NA` action? Sure! For example, we can calculate the mean of the *non-missing* heights in our example dataset by updating the action to `na.rm = TRUE` (read: "Remove NAs? Yes!"). The mean value is then calculated based on the two existing height values, and any `NA` are ignored.

```
mean(heights$height, na.rm = TRUE)
```

```
[1] 169
```

Now that we know a bit about how R stores and handles missing values, we can start exploring them.

## 1.3 Do my data contain missing values?

```
library(naniar)
```

Missing values don't jump out and scream "I'm here!". They're usually hidden, like a needle in a haystack - especially in large datasets. We need tools (or rather, functions) to quickly identify and count missing values.

Let's create an example vector `x`, which contains missing values encoded as `NA`:

```
x <- c(1, NA, 3, NA, NA, 5, 8)
x
```

```
[1]  1 NA  3 NA NA  5  8
```

In this small vector (n = 7), we can quickly see that the $2^{nd}$, $4^{th}$ and $5^{th}$ values in the vector are `NA`. With larger data, however, we would want tools to identify these for us, instead of manually looking for them. Two functions for identifying `NA`s are `are_na()` and `any_na()`. These are from the `naniar` R package.

### 1.3.1 `are_na()`: which values are `NA`?

The `are_na()` function checks each value in a vector or data frame (i.e., for each value it asks "is this value `NA`"?) then returns TRUE (if `NA`) or FALSE (if anything *besides* `NA`).

```r
are_na(x)
```

```
[1] FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE
```

As expected, the three `NA` elements in `x` return `TRUE`.

### 1.3.2 `any_na()`: are there *any* `NA`s?

The `are_na()` function tells us *which* values are `NA`. If we instead want to know if *any* elements in our data are `NA`, we can instead use `any_na()`. The `any_na` function returns TRUE if there are *any* missing values (stored as `NA`s), and FALSE if there are none.

```r
any_na(x)
```

```
[1] TRUE
```

Because `x` contains at least one `NA`, we see that `any_na(x)` returns TRUE, and will return FALSE if there are no `NA` values:

```r
any_na(c(1, 2, 3, 4))
```

```
[1] FALSE
```

The `any_na` and `are_na` functions can give us a "heads up" about whether or not our data contains missing values. To deal with them responsibly, however, we need to dig further into patterns of missingess. The next step is exploring missingness visually.

### 1.3.3 Your Turn: Exercises

You can complete the exercises in an interactive environment using the learnr exercises for this section at (link).

# 2 Missing data gotchya's

```
library(naniar)
```

Missing data are a special part of R, they are baked right into the software, and aren't only made available by certain R packages. However, there are some quirks of missing data that mean they can catch you off guard. Let's call these the "missing data gotchya's". Let's discuss some of these now.

## 2.1 NaN vs NA

In R, there is a special value, `NaN`, which stands for "Not a Number". A `NaN` will come from operations like the square root of -1:

```
sqrt(-1)
```

```
Warning in sqrt(-1): NaNs produced
```

```
[1] NaN
```

Now, R actually interprets `NaN` as a missing value, treating it the same way it treats `NA`. Even if it is technically not a missing value.

```
any_na(NaN)
```

```
[1] TRUE
```

This might come up in a data analysis, if you were to transform some data with the square root and then count the number of missing values, and there is a negative value, you might get caught out.

```
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
v ggplot2 3.3.6     v purrr   0.3.4
v tibble  3.1.7     v dplyr   1.0.9
v tidyr   1.2.0     v stringr 1.4.0
v readr   2.1.2     v forcats 0.5.1
```

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

```r
library(naniar)
vec <- c(-1:4)
sqrt(vec)
```

```
Warning in sqrt(vec): NaNs produced
```

```
[1]      NaN 0.000000 1.000000 1.414214 1.732051 2.000000
```

```r
sqrt(vec) %>% n_miss()
```

```
Warning in sqrt(vec): NaNs produced
```

```
[1] 1
```

## 2.2 NULL vs NA

In R, `NULL` is an empty value. For example, if we create a vector of NULL values, only one appears

```r
c(NULL, NULL, NULL)
```

```
NULL
```

Compare this to a vector of NA values:

```r
c(NA, NA, NA)
```

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

Importantly, NULL values are not missing values, but rather just "empty" values. This is subtly different from missing: An empty bucket isn't **missing** water.

```r
any_na(NULL)
```

```
[1] FALSE
```

Another way to think about this is if you were recording features of animals - animals are all quite different! So you record `horn_length` of a mouse as NULL - because mice do not have horns. It's not that it *should* have been recorded and wasn't - it shouldn't be recorded because it doesn't exist.

## 2.3 Inf vs NA

`Inf` is an Infinite value, and results from equations like `10/0`:

```r
10 / 0
```

```
[1] Inf
```

It is not counted as a missing value

```r
any_na(Inf)
```

```
[1] FALSE
```

# 3 “NA” vs NA

Using the function `is.na()` will return true for `NA`

```
is.na(NA)
```

```
[1] TRUE
```

But for a quoted character, “NA”, is not missing.

```
is.na("NA")
```

```
[1] FALSE
```

## 3.1 Conditional statements and NA

Beware of conditional statements with missing values. For example:

- NA or TRUE is TRUE
- NA or FALSE is NA
- NA + NaN is NA
- NaN + NA is NaN

```
NA | TRUE
```

```
[1] TRUE
```

```
NA | FALSE
```

```
[1] NA
```

```r
NA + NaN
```

```
[1] NA
```

```r
NaN + NA
```

```
[1] NaN
```

## 3.2 The multiple flavours of NA values

`NA` values represent missing values in R. There are actually many different flavours of NA values in R:

- `NA` for logical
- `NA_character_` for characters
- `NA_integer_` for integer values
- `NA_real_` for doubles (values with decimal points)
- `NA_complex_` for complex values (like `1i`)

So what? What does this mean?

```r
is.na(NA)
```

```
[1] TRUE
```

```r
is.na(NA_character_)
```

```
[1] TRUE
```

```r
is.character(NA_character_)
```

```
[1] TRUE
```

```r
is.double(NA_character_)
```

```
[1] FALSE
```

```r
is.integer(NA_integer_)
```

```
[1] TRUE
```

```r
is.logical(NA)
```

```
[1] TRUE
```

Uhhh-huh. So, neat? Right? NA values are this double entity that have different classes? Yup! And they're among the special reserved words in R. That's a fun fact.

OK, so why care about this? Well, in R, when you create a vector, it has to resolve to the same class. Not sure what I mean?

Well, imagine you want to have the values 1:3

```r
c(1,2,3)
```

```
[1] 1 2 3
```

And then you add one that is in quotes, "hello there":

```r
c(1,2,3, "hello there")
```

```
[1] "1"           "2"           "3"           "hello there"
```

They all get converted to "character".

Well, it turns out that `NA` values need to have that feature as well, they aren't this amorphous value that magically takes on the class. Well, they kind of are actually, and that's kind of the point - we don't notice it, and it's one of the great things about R, it has native support for NA values.

So, imagine this tiny vector, then:

```r
vec <- c("a", NA)
vec
```

```
[1] "a" NA
```

```
is.character(vec[1])
```

```
[1] TRUE
```

```
is.na(vec[1])
```

```
[1] FALSE
```

```
is.character(vec[2])
```

```
[1] TRUE
```

```
is.na(vec[2])
```

```
[1] TRUE
```

OK, so, what's the big deal? What's the deal with this long lead up? Stay with me, we're nearly there:

```
vec <- c(1:5)
vec
```

```
[1] 1 2 3 4 5
```

Now, let's say we want to replace values greater than 4 to be the next line in the song by Feist.

If we use the base R, `ifelse`:

```
ifelse(vec > 4, yes = "tell me that you love me more", no = vec)
```

```
[1] "1"                             "2"
[3] "3"                             "4"
[5] "tell me that you love me more"
```

It converts everything to a character. We get what we want here.

Now, if we use `dplyr::if_else`:

```r
dplyr::if_else(vec > 4, true = "tell me that you love me more", false = vec)
```

```
Error in `dplyr::if_else()`:
! `false` must be a character vector, not an integer vector.
```

ooo, an error? This is useful because you might have a case where you do something like this:

```r
dplyr::if_else(vec > 4, true = "5", false = vec)
```

```
Error in `dplyr::if_else()`:
! `false` must be a character vector, not an integer vector.
```

Which wouldn't be protected against in base:

```r
ifelse(vec > 4, yes = "5", no = vec)
```

```
[1] "1" "2" "3" "4" "5"
```

#### So why does that matter for NA values?

Well, because if you try and replace values more than 4 with `NA`, you'll get the same error:

```r
dplyr::if_else(vec > 4, true = NA, false = vec)
```

```
Error in `dplyr::if_else()`:
! `false` must be a logical vector, not an integer vector.
```

But this can be resolved by using the appropriate `NA` type:

```r
dplyr::if_else(vec > 4, true = NA_integer_, false = vec)
```

```
[1]  1  2  3  4 NA
```

And that's why it's important to know about.

It's one of these somewhat annoying things that you can come across in the tidyverse, but it's also kind of great. It's opinionated, and it means that you will almost certainly save yourself a whole world of pain later.

What is kind of fun is that using base R you can get some interesting results playing with the different types of `NA` values, like so:

```
ifelse(vec > 4, yes = NA, no = vec)
```

```
[1]  1  2  3  4 NA
```

```
ifelse(vec > 4, yes = NA_character_, no = vec)
```

```
[1] "1" "2" "3" "4" NA
```

It's also worth knowing that you'll get the same error appearing in `case_when`:

```
dplyr::case_when(
  vec > 4 ~ NA,
  TRUE ~ vec
  )
```

```
Error in names(message) <- `*vtmp*`: 'names' attribute [1] must be the same length as the ve
```

But this can be resolved by using the appropriate `NA` value

```
dplyr::case_when(
  vec > 4 ~ NA_integer_,
  TRUE ~ vec
  )
```

```
[1]  1  2  3  4 NA
```

# Part II

# Explore Missing Values

# 4 Explore missing values

```
library(naniar)
library(dplyr)
```

In previous sections, we learned what missing values are, how R deals with them in basic operations, and several ways (including with `any_na` and `are_na`) to perform a cursory check for missing values in our data. A critical next step in exploring missingness is to *visualize* missing values, which can reveal patterns of missingness across variables (columns) and cases (rows) in our data.

## 4.1 Explore "big picture" missingness

To start, recommend getting a "bird's eye view" of missingness in the data. We can get started with a few big picture questions:

- **Where**, and how frequently, do missing values occur in the data overall?
- **How often**, and across **what variables**, do missing values co-occur?
- Are there notable **patterns** of missingness across groups?

We will approach these questions using using five functions from the `naniar` package:

- `vis_miss` to visualize where `NA` exist in a data frame
- `n_miss` for overall frequency of `NA`
- `prop_miss` for the proportion of values in the data that are `NA`
- `gg_miss_upset` to visualize overall co-occurrence of missingness
- `gg_miss_fct` for a heatmap of missingness across variables, by groups

Note that not all of these functions are for visualisation.

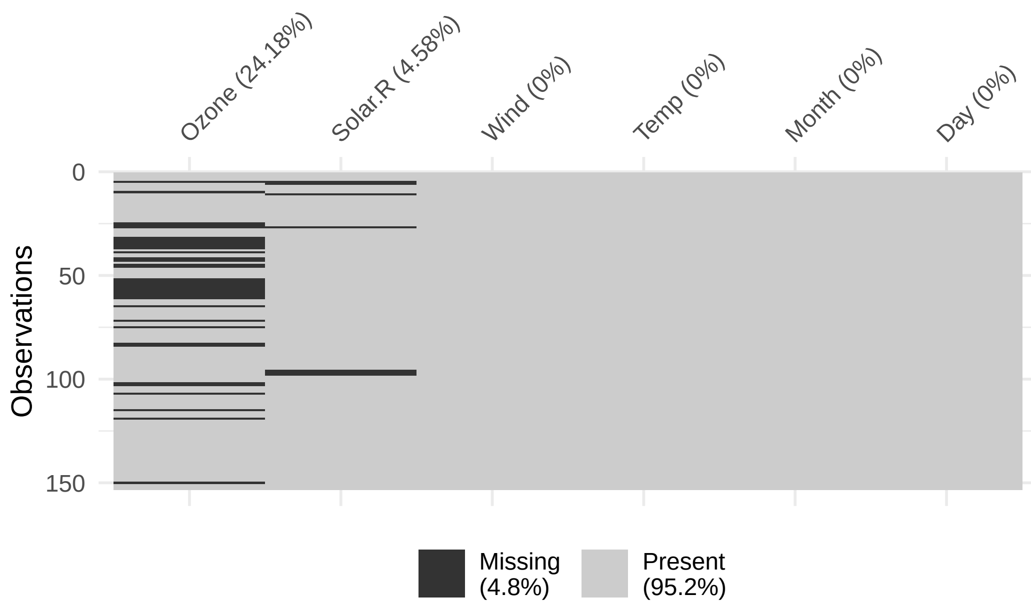
### 4.1.1 `vis_miss` to visualize locations of missing values

When you first get a dataset, it can be difficult to get a visceral sense of **where** missing values are. To get an overview of the prevalence and patterns of missingness in the data, use the `vis_miss` function. This function is in `naniar` and is exported from the `visdat` package.

For example, with the built-in `airquality` dataset:

```r
vis_miss(airquality)
```

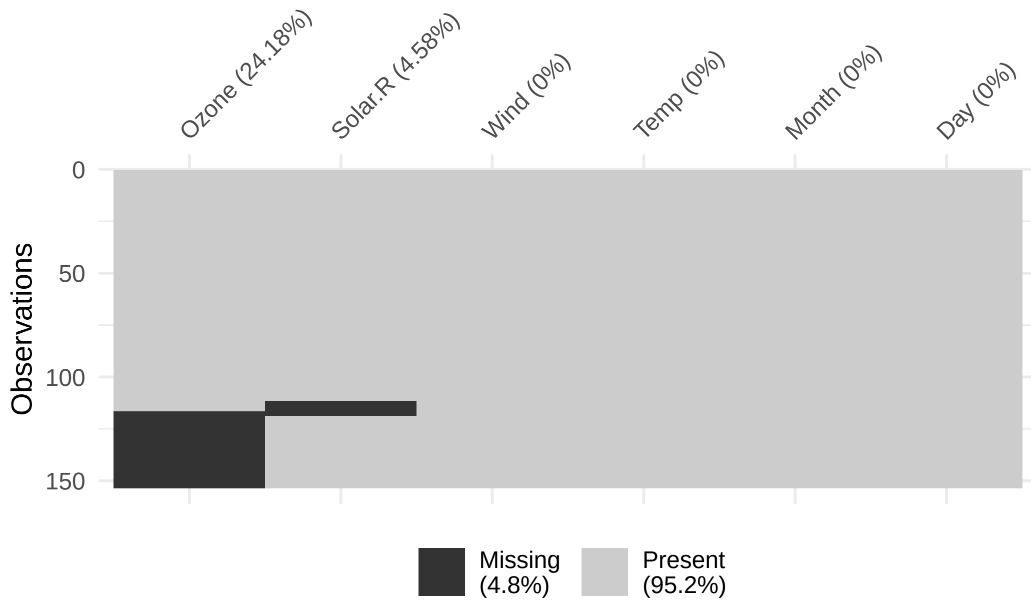
```
Warning: `gather_()` was deprecated in tidyr 1.2.0.
Please use `gather()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```




The `vis_miss` function produces an out-of-the-box binary heatmap of missing values, where each "cell" in the heatmap corresponds to an element in the original data. In other words, we can think of the heatmap as a "spreadsheet" of the original data, where values have been replaced with one of two colors: black cells to indicate missing values, and gray cells to indicate non-missing values.

`vis_miss` also provides useful summary statistics, showing the overall percentage of missingness in the legend (at bottom), and the amount of missings in each variable (alongside column labels). The function also allows for clustering of the missing data by setting `cluster = TRUE`: this orders the rows by missingness to identify common co-occurrences.

```r
vis_miss(airquality, cluster = TRUE)
```




**What can we learn from the `vis_miss` output?**

From the `vis_miss` output, we can consider big picture questions about missingness prevalence and patterns. For example:

- Do missing values appear randomly distributed throughout the data, or are they clustered within several variables?
- Are there notable streaks of missingness within variable(s), and why might those exist?

Considering the visualization above for `vis_miss(airquality)`. We might interpret the output as follows: Overall prevalence of missingness is low (4.8% missing), and there are only missing values in two variables: `Ozone` (24.8% missing) and `Solar.R` (4.58% missing). Missing values do not necessarily co-occur. There are several streaks of missingness (shown as black areas spanning adjacent cases), most notably in the `Ozone` variable from rows 52 - 61.

### 4.1.2 Overall counts and proportions of missing values

If we have a very small dataset, like the vector x shown below, we can locate and count missing values manually, by simply counting in our heads how many `NA` values there are.

```
x <- c(1, NA, 3, NA, NA, 5, 8)
```

But this doesn't scale. What if we had a vector of length 2,892? Or a 52 column × 841,000 row data frame? It would be nigh-impossible to find and count all `NA` values.

Visualisation is great, but sometimes we just need a hard number, so you can say something like, "54% of the data is missing!". Use `n_miss` and `prop_miss` for a quick quantitative summary of overall missingness. Both return a single value for the total count and proportion of missing values in the entire data frame.

The `n_miss` function returns the total count all values in the data that are `NA`:

```
n_miss(x)
```

```
[1] 3
```

The `prop_miss` function returns the proportion of missings, which gives important context: here, we see that 42.86% of values in the data are missing.

```
prop_miss(x)
```

```
[1] 0.4285714
```

The complements of `n_miss` and `prop_miss` are `n_complete` and `prop_complete`, which return the number and proportion of *complete* (non-missing) values, respectively.

```
n_complete(x)
```

```
[1] 4
```

```
prop_complete(x)
```

```
[1] 0.5714286
```

The examples above show how we can use `n_miss`, `prop_miss`, and their complements for a small vector (`x`). We can apply them similarly to the larger `airquality` data used in the `vis_miss` example above.

To find the total number of `NA` in `airquality`:

```
n_miss(airquality)
```

```
[1] 44
```

And for the proportion of `NA` in `airquality`:

```
prop_miss(airquality)
```

```
[1] 0.04793028
```

Which tells us that there are 44 total missing values in `airquality`, or 4.79%. Note that this total proportion of missingness matches the value reported from `vis_miss`.

That is much quicker, easier (and safer, and more reproducible!) than manually searching for and counting missing values, especially in larger data.

**An Aside**

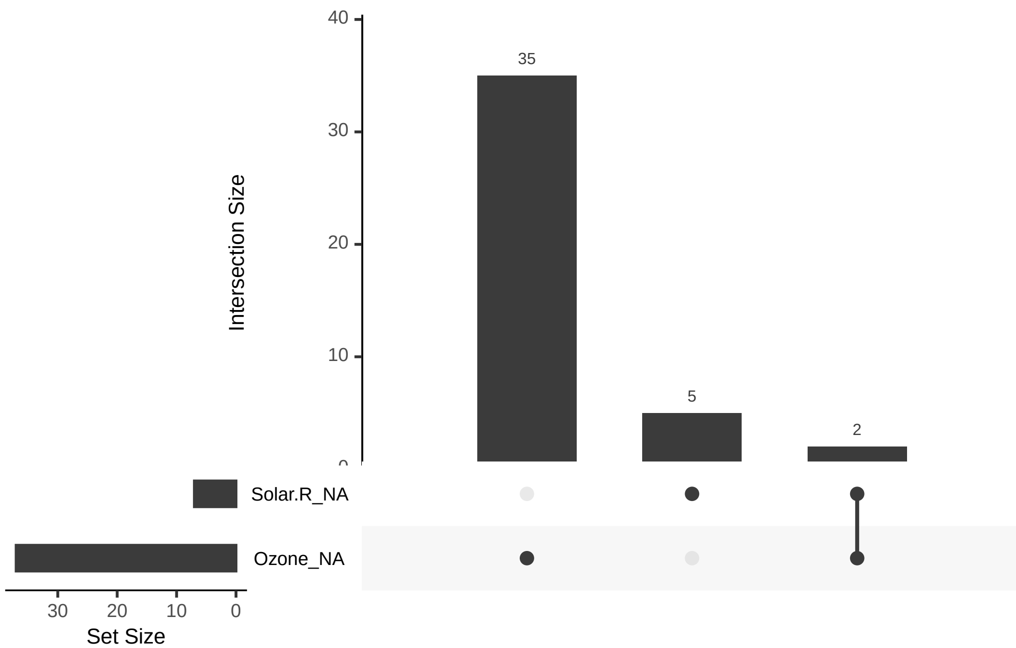
Under the hood, `n_miss(x)` is computed as `sum(is.na(x))`, and `prop_miss(x)` is `mean(is.na(x))`. These are rather brief short hand functions - so you might ask why bother making them? I believe it is because there is actually a fair bit packed into these functions. `sum(is.na(x))` works because the output of `is.na(x)` is logical, and you can add logicals together, as TRUE and FALSE are coerced to 1 and 0, respectively. Similarly, you can take the mean of logicals. However, as a new R user, I found this a bit magical and wasn't able to remember the right way to do it. A nice consequence of more descriptive names, `n_miss` and `prop_miss` is that we can take the complement, so a user can use tab-complete to also find `n_complete` and `prop_complete`. These functions are implemented as `sum(!is.na(x))` and `prop(!is.na(x))`, which again, I think can be a lot to remember, especially when first starting out with R. Descriptively naming these functions is an important part of how `naniar` is designed.

### 4.1.3 Co-occurrence of missing values

To visualise common combinations of missingness - for example, which variables and cases tend to be missing together - we can use `gg_miss_upset` to create an UpSet plot [**?**]. This powerful visualisation shows the frequency of unique combinations of missing value co-occurrence.

We create an UpSet plot like so:

```
gg_miss_upset(airquality)
```




There is a lot going on, here are the main pieces:

- The vertical bars indicate the frequency of unique missingness combinations, indicated by the dots below each bar that correspond to variable names to their left
- The horizontal bars in the lower left indicate the total number of missing values for each variable

For example, let's first consider the vertical bar of height 2 (on the far right), beneath which there are black dots next to both `Solar.R_NA` and `Ozone_NA`. That bar indicates that there are **2 cases** (rows) in the data where exactly the `Solar.R` and `Ozone` variables contain `NA`.

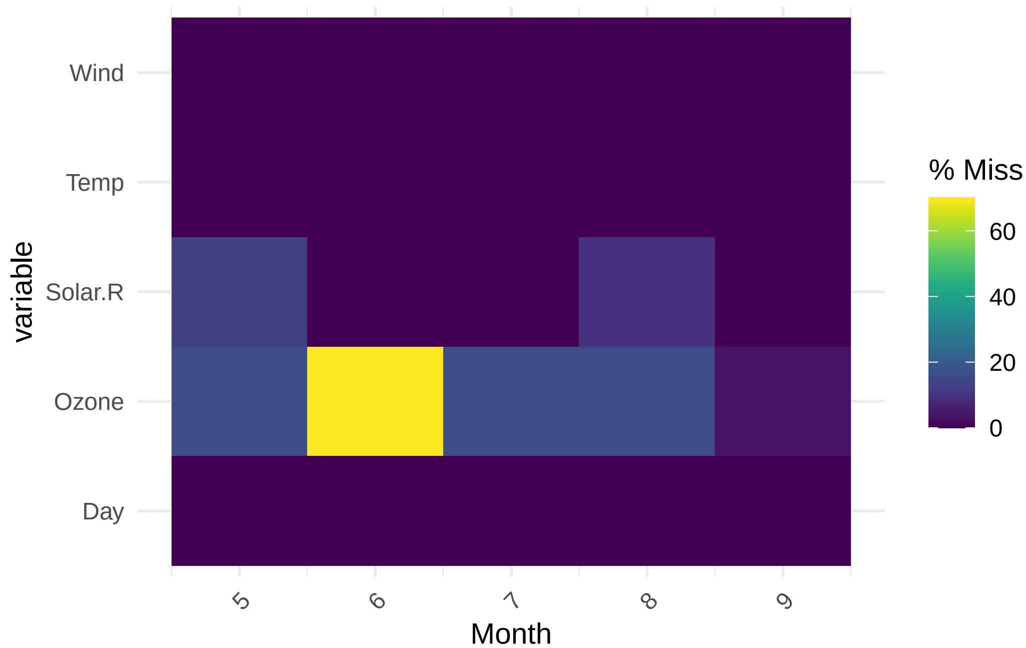
The other two bars indicate that there are exactly 35 cases in which only the `Ozone` variable value is `NA`, and 5 cases in which only the `Solar.R` variable value is `NA`.

To summarize: `gg_miss_upset` creates an UpSet plot to visualize frequency of missing value combinations (co-occurrence) across variables.

### 4.1.4 Visualize missingness by factor level

To explore how missingness varies across factor levels within variables, use `gg_miss_fct`. Below we create a heatmap showing the prevalence of missingness across all variables in `airquality`, separated by each level in the `Month` variable:

```r
gg_miss_fct(x = airquality, fct = Month)
```




The output is a heatmap, with the x-axis showing the levels of the specified factors, and the y-axis showing other variables in the data, and colour showing the frequency of missingness (purple = lower missingness, yellow = higher missingness).

We see that the output of `gg_miss_fct` here, with each level of Month on the x-axis, clearly shows the highest proportion of missingness occurs for the Ozone variable in Month 6 (in agreement with previous summary outputs). Note: `gg_miss_fct` does not support facetting.

These "big picture" analyses or missingness are an essential starting point in exploration because they show us how much of the data is missing overall, and can reveal patterns in missingness (e.g. streaks, co-occurrence, and differences between factor levels). Next, it is important to further investigate how missingness occurs within variables (columns) and cases (rows).

# 5 Missingness by variables (columns) and cases (rows)

```r
library(naniar)
library(dplyr)
```

Once we have a broad overview of missingness in the data, the next step is to explore how missingness exists at finer resolution within variables and cases. You might also refer to variables and cases as "columns" and "rows", but for consistency, we will use "variables" and "cases". Below are the functions we will use in this section to explore missingness in variables and cases:

**Functions to explore missingness by variable:**

- `gg_miss_var`: visualise frequency of missingness by variable
- `miss_var_summary`: table of missingness frequency by variable
- `miss_var_table`: table of missing frequencies by variable

**Functions to explore missingness by case:**

- `gg_miss_case`: visualise frequency of missing values by case
- `miss_case_summary`: table of missingness frequency by case
- `miss_case_table`: table of missing frequencies by case

**An aside**

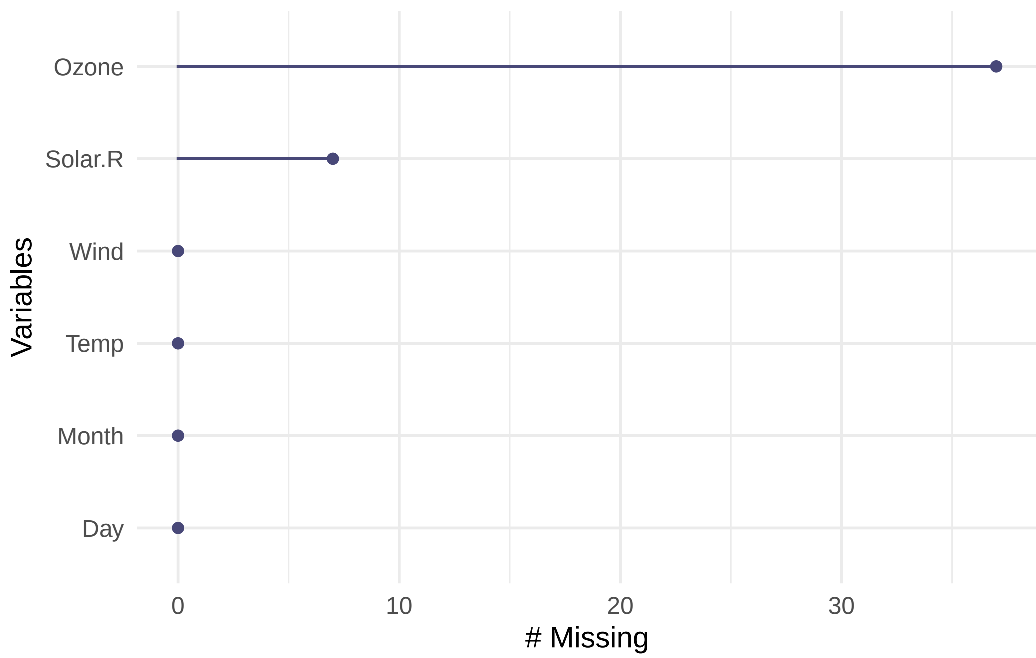
You might notice that there is a lot of similarity in the naming and purpose of each of these functions - this is intentional! They are designed to have names that help cue your understanding or purpose of the next task. There are many other functions that start with `miss_var_` and `miss_case`, and `gg_miss`. These functions have been written like so to facilitate exploration of new functions, and also to reduce the cognitive burden of trying to remember what a function is called. Instead, you can focus on, "I want to explore missings by variable - I'll start by exploring what is in `miss_var`", and if you want to create visualisations, you can use `gg_miss_*` to explore available options.

### 5.0.1 Missingness within variables

The `gg_miss_var` and `miss_var_summary` functions in `naniar` return visual and tabular summaries, respectively, of missingness within each variable. For example, `gg_miss_var` applied to the `airquality` returns a lollipop plot with the frequency of missingness on the x axis, and the variable name on the y-axis:

```
gg_miss_var(airquality)
```

```
Warning: It is deprecated to specify `guide = FALSE` to remove a guide. Please
use `guide = "none"` instead.
```




Note that the visualisation is ordered (high-to-low) by variable highest frequency of missing values.

To instead create a **table** with the number and percentage of missing values for each variable (column), use `miss_var_summary`:

```
miss_var_summary(airquality)
```

```
# A tibble: 6 x 3
  variable n_miss pct_miss
  <chr>     <int>    <dbl>
1 Ozone        37    24.2
2 Solar.R       7     4.58
3 Wind          0     0
4 Temp          0     0
5 Month         0     0
6 Day           0     0
```

We see that `miss_var_summary()` returns a data frame where each row in the output contains the total number (`n_miss`) and percentage (`pct_miss`) of missing values for each variable in the original data. Note that `miss_var_summary` and `gg_miss_summary` give us the same information, presented differently: the values in the `miss_var_summary` table for each variable align with the frequency of missingness indicated on the x-axis from the `gg_miss_var` output.

An example of how to interpret missingness within variables from `miss_var_summary` and `gg_miss_var` is:

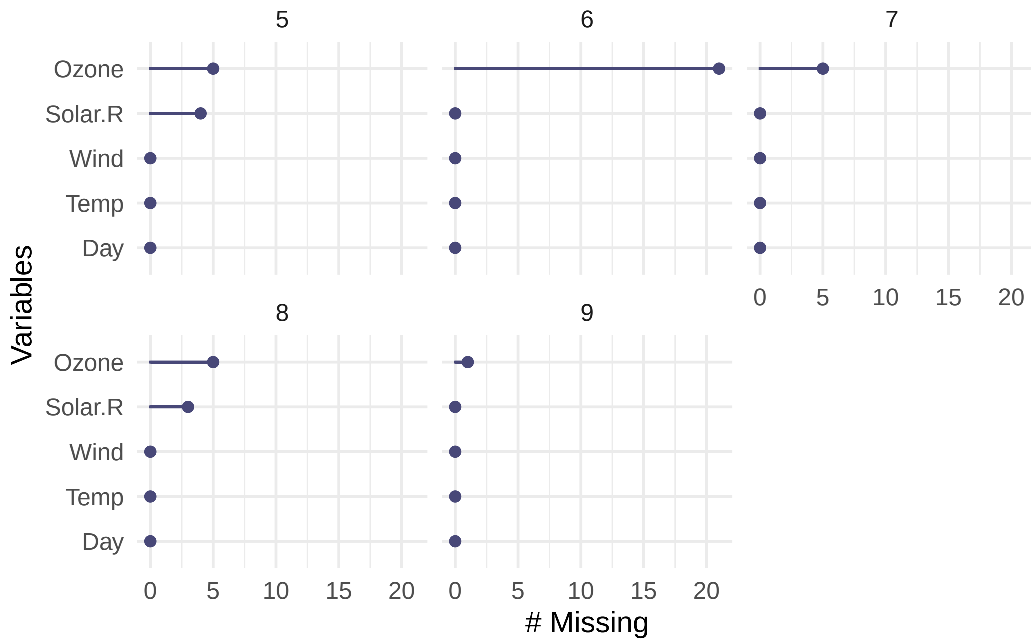
> An overview of missingness in the in the airquality dataset ($n_{obs}$ = 153). The Ozone variable has the highest frequency of missing values ($n_{missing}$ = 37; percent missing = 24.2%), followed by Solar.R ($n_{missing}$ = 7; percent missing = 4.6%). The remaining four variables (Wind, Temp, Month, and Day) contain no missing values.

#### 5.0.1.1 Missingness by variable, within groups

We can explore missingness in detail within each variable by using the `facet` argument, which will split each plot into one facet per level of a variable. The example below shows the number of missing values in each variable (`gg_miss_var`) in the `airquality` data, now broken up by the different levels in `Month`.

```r
gg_miss_var(airquality, facet = Month)
```

```
Warning: It is deprecated to specify `guide = FALSE` to remove a guide. Please
use `guide = "none"` instead.
```




In the graph above, we can see there is now a separate panel for each month appearing in the data (5, 6, 7, 8, and 9), to allow for a comparison of missingness by variable across and within each month.

The same information can be reported in tabular form by `miss_var_summary` in combination with `group_by` to designate which variable to group by. For example, we parse missingness by Month in the airquality dataset:

```
airquality %>%
  group_by(Month) %>%
  miss_var_summary()
```

```
# A tibble: 25 x 4
# Groups:   Month [5]
   Month variable n_miss pct_miss
   <int> <chr>     <int>    <dbl>
 1     5 Ozone         5     16.1
 2     5 Solar.R       4     12.9
 3     5 Wind          0      0
 4     5 Temp          0      0
 5     5 Day           0      0
 6     6 Ozone        21     70
```

```
 7      6 Solar.R       0       0
 8      6 Wind          0       0
 9      6 Temp          0       0
10      6 Day           0       0
# ... with 15 more rows
```

Here, we see the Ozone variable contains 5 missing values for Month 5, and 21 missing values for Month 6.

#### 5.0.1.2 `miss_var_table`

It can be useful to explore *how often* (i.e., for how many variables or cases) different frequencies of missingness occur. That's kind of a brainful, so here are some example questions we might ask:

> "How many variables contain zero missing values?"
>
> "How many variables contain one missing values?"
>
> "How many variables contain two missing values?"

The `miss_var_table()` function tells us how many variables contain different frequencies of missingness. For example, we can use `miss_var_table` with our airquality data to calculate and return the *number of variables* with different frequencies of missing values:

```r
miss_var_table(airquality)
```

```
# A tibble: 3 x 3
  n_miss_in_var n_vars pct_vars
          <int>  <int>    <dbl>
1             0      4     66.7
2             7      1     16.7
3            37      1     16.7
```

The table returned above tells us the following:

- Row 1 in output table: **four variables** (n_vars = 4, or 66.7% of all variables) contain **zero missing values** (n_miss_in_var = 0)
- Row 2 in output table: **one variable** (n_vars = 1, or 16.7% of all variables) contains **7 missing values** (n_miss_in_var = 7)
- Row 3 in output table: **one variable** (n_vars = 1, or 16.7% of all variables) contains **37 missing values** (n_miss_in_var = 37)

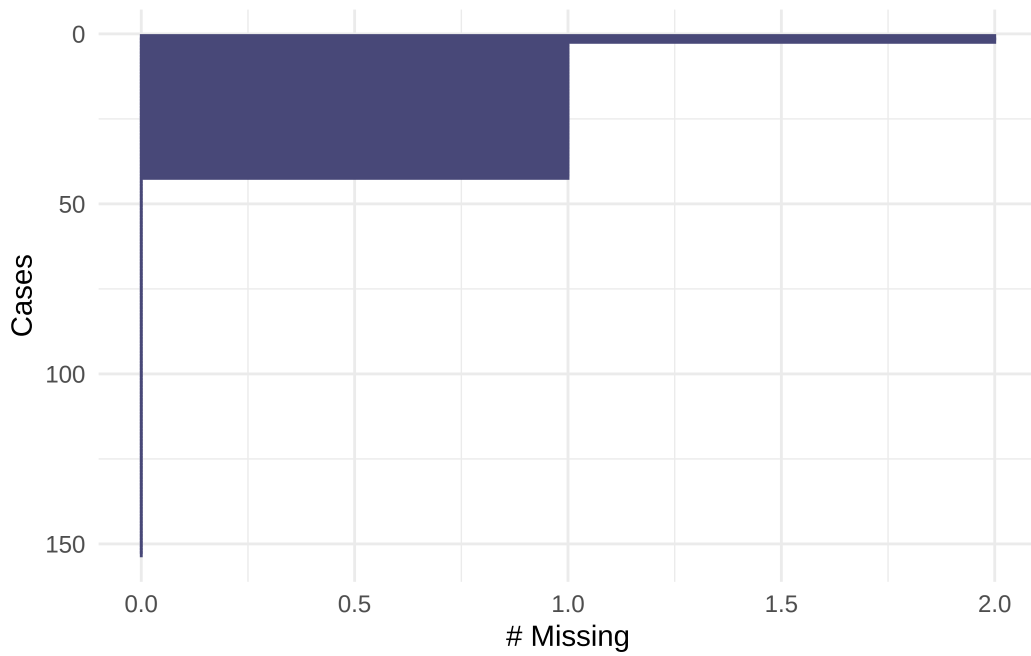
Writing this as a brief summary, we might write:

> Four variables (~ 66.7% of columns) contain no missing values; one variable contains 7 missing values, and one variable contains 37 missing values.

### 5.0.2 Missingness by case (rows)

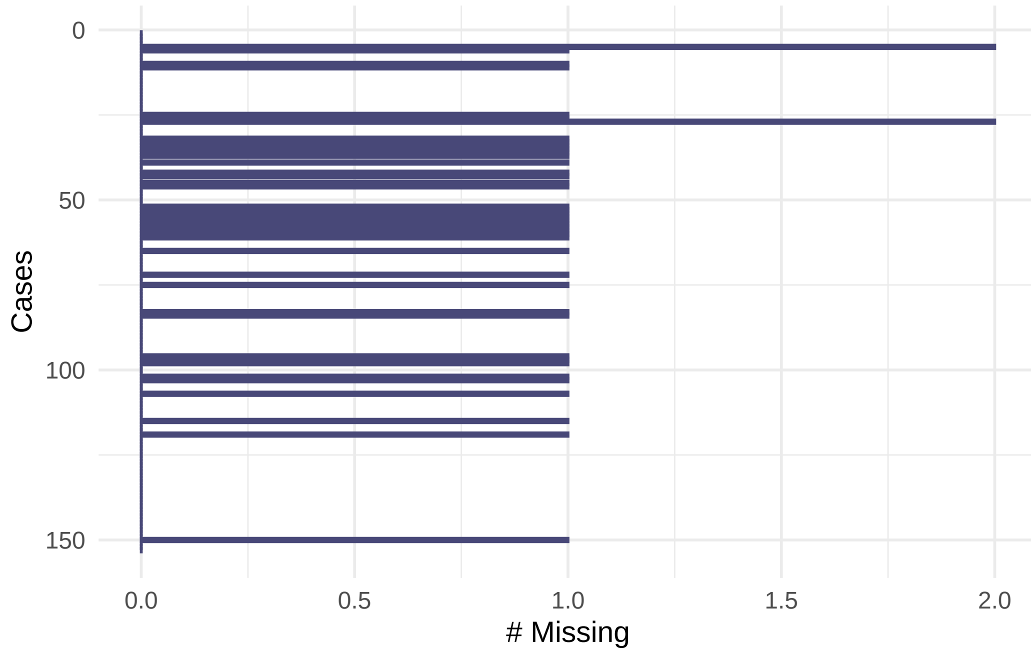
The `gg_miss_case()` and `miss_case_summary()` functions return visual and tabular summaries of missingness by case (row) within the data. For example, to visualize missingness by case:

```
gg_miss_case(airquality)
```




Note that the visualisation is similarly ordered (high-to-low) by case(s) with the highest frequency of missing values. The ordering in `gg_miss_case` can be turned off with option, `order_cases = FALSE`, which will keep the order of the data as presented to the function.

```
gg_miss_case(airquality, order_cases = FALSE)
```




For a tabular summary of missingness by case, use `miss_case_summary`. The `miss_case_summary` function returns a summary data frame with the frequency and percentage of missing values for *each case* (row) in the original data, arranged by decreasing missingness.

```
miss_case_summary(airquality)
```

```
# A tibble: 153 x 3
    case n_miss pct_miss
   <int>  <int>    <dbl>
 1     5      2     33.3
 2    27      2     33.3
 3     6      1     16.7
 4    10      1     16.7
 5    11      1     16.7
 6    25      1     16.7
 7    26      1     16.7
 8    32      1     16.7
 9    33      1     16.7
10    34      1     16.7
# ... with 143 more rows
```

In the example output above, the *case* column contains the original row number (case) in the data, and the frequency and percent missing is returned for each case. Here, we can interpret the first two rows of this summary as follows:

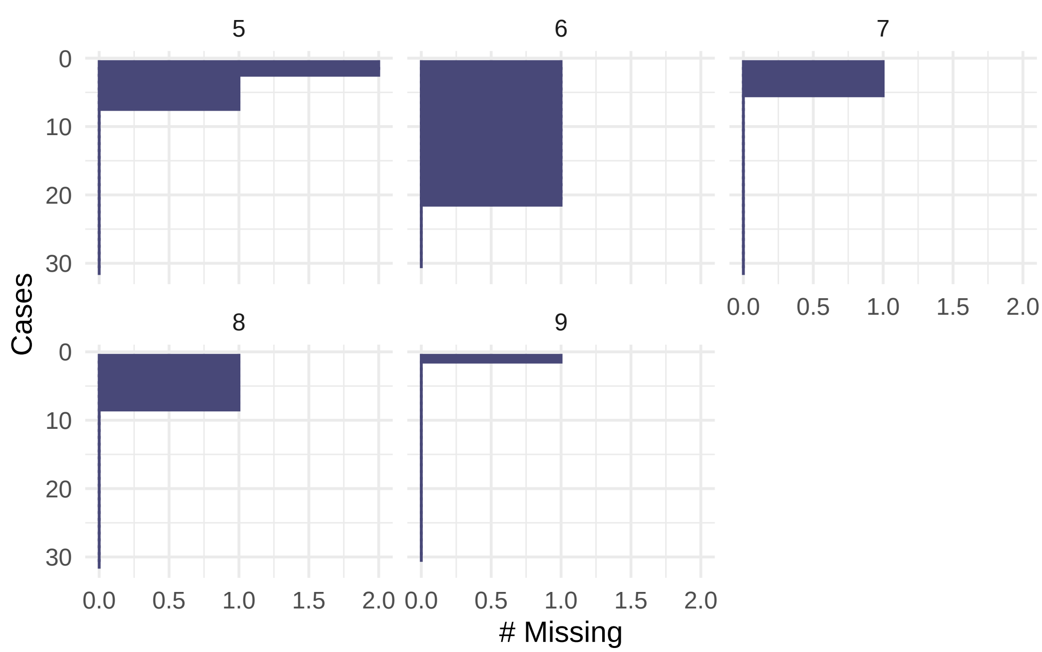
> In the airquality dataset, cases 5 and 27 - the 5$^{th}$ and 27$^{th}$ rows in the original dataset - each contain 2 missing values (i.e. 2 of 6 values, or 33.3%, in each row are missing).

#### 5.0.2.1 Missingness by case, within groups

Missingness by case can be further explored within groups by faceting (with `gg_miss_case`) or in combination with `group_by`.

To visualize missingness by case within groups, add a faceting variable to the argument `facet`. The example below shows the number of missing values in each case (`gg_miss_case`) in the `airquality` data, faceted by `Month`.

```
gg_miss_case(airquality, facet = Month)
```




To return the above data (missingness by case, separated by each level in `Month`) in tabular form, use `miss_case_summary` in combination with `group_by`:

```r
airquality %>%
  group_by(Month) %>%
  miss_case_summary()
```

```
# A tibble: 153 x 4
# Groups:   Month [5]
   Month  case n_miss pct_miss
   <int> <int>  <int>    <dbl>
 1     5     5      2       40
 2     5    27      2       40
 3     5     6      1       20
 4     5    10      1       20
 5     5    11      1       20
 6     5    25      1       20
 7     5    26      1       20
 8     5     1      0        0
 9     5     2      0        0
10     5     3      0        0
# ... with 143 more rows
```

#### 5.0.2.2 `miss_case_table`

We may want to know:

> "How many cases are complete (no missing values)?"
>
> "How many cases have one missing value?"
>
> "How many cases have two missing values?"

The `miss_case_table` function in `naniar` tells us how many cases contain different frequencies of missingness. The example below returns the number of cases (rows) in the `airquality` data that contain different numbers of missing values:

```r
miss_case_table(airquality)
```

```
# A tibble: 3 x 3
  n_miss_in_case n_cases pct_cases
           <int>   <int>     <dbl>
1              0     111     72.5
2              1      40     26.1
3              2       2      1.31
```

We could summarize the output above as follows:

> The majority of cases (72.5%) are complete, 26.1% of cases contain one missing value, and ~1.3% of cases contain two missing values; no cases contain more than two missing values.

Using the `naniar` functions in this section, we got a more detailed view of missingness within variables and cases. In the next section, we investigate patterns of missingness *streaks* and *spans*.

# 6 Missingness in spans and streaks

```
library(naniar)
library(tidyverse)
```

In previous sections, we learned how to visualize and tabulate missingness for our data overall, and at finer resolution within variables and cases. Another pattern of missingness we can explore are the **streaks** and **spans** of missingness, which are defined as follows:

- **Streaks** are sequential missing or non-missing values. For example, the vector `c(4, 8, NA, NA, NA, 5)` has a streak of two non-missing values (4 and 8), followed by a streak of three missing values, followed by a "streak" of one non-missing value (5). You can see this below in figure @ref(fig:plot-span-streak) on the column "weekday", where there is a streak of missingness at the start, and at the end of the column. We see that there is some overall pattern here, but we do not have information on the details of the streak, specifically, how many observations before the missingness starts, between missingness, and so on.

- **Spans** are repeated periods within the data that we want to explore missingness within, and between. For example, if we have air quality data recorded at 1-hour intervals, we may want to explore the prevalence of missingness within each 1-day span. In that case, each *span* would consist of 24 sequential observations. We can see missingness over repeating spans in the "temp" column in figure @ref(fig:plot-span-streak). Notably, we can get some information from this that missingness appears to repeat and be a similar size, but we do not have further details on the size of these patches of missingness.

```
add_n_na <- function(x, n_na){
  x[sample(x = vctrs::vec_size(x), size = n_na)] <- NA
  x
}

splice_n_na <- function(x, position, n_na){
  x[position:(position+n_na)] <- NA
  x
}
```

```r
dat_span <- expand_grid(
  weekday = 1:7,
  hour = 1:24
) %>%
  mutate(
    temp = floor(runif(n = 168, min = 11, max = 29)),
    temp = splice_n_na(temp, position = 12, n_na = 5),
    temp = splice_n_na(temp, position = 36, n_na = 5),
    temp = splice_n_na(temp, position = 60, n_na = 5),
    temp = splice_n_na(temp, position = 84, n_na = 5),
    temp = splice_n_na(temp, position = 108, n_na = 5),
    temp = splice_n_na(temp, position = 132, n_na = 5),
    weekday = splice_n_na(weekday, position = 26, n_na = 40),
    weekday = splice_n_na(weekday, position = 98, n_na = 15),
  )
```

```r
vis_miss(dat_span)
```

```
Warning: `gather_()` was deprecated in tidyr 1.2.0.
Please use `gather()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

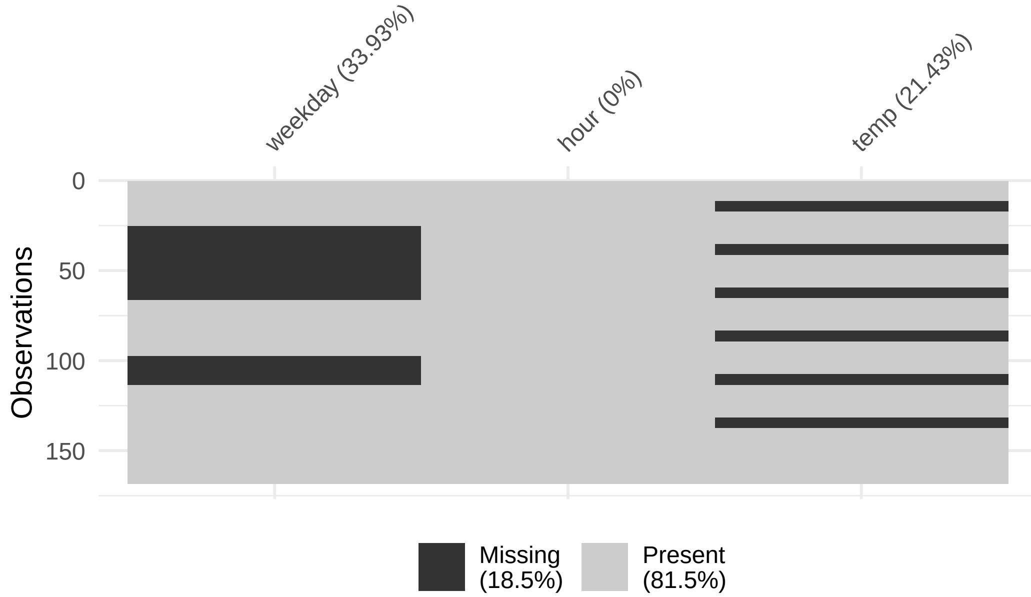



Figure 6.1: Made up data of weekday, hour, and temperature

Below are the functions we will use in this section to explore missingness in **spans** and **streaks**

- `gg_miss_span()`: visualise the proportion of missing values by **span**
- `miss_var_span()`: table containing counts and proportions of missingness by **span**
- `miss_var_run()`: table containing lengths of **streaks** for missing and non-missing values in the data

### 6.0.1 Missingness in spans

The `gg_miss_span()` and `miss_var_span()` functions in `naniar` provide visual and tabular summaries of missing values in user-specified *spans*, or equally-sized periods, within the data.

Let's show the data we visualised using `vis_miss()` in figure @ref(fig:plot-span-streak):

```
knitr::kable(head(dat_span, 26))
```

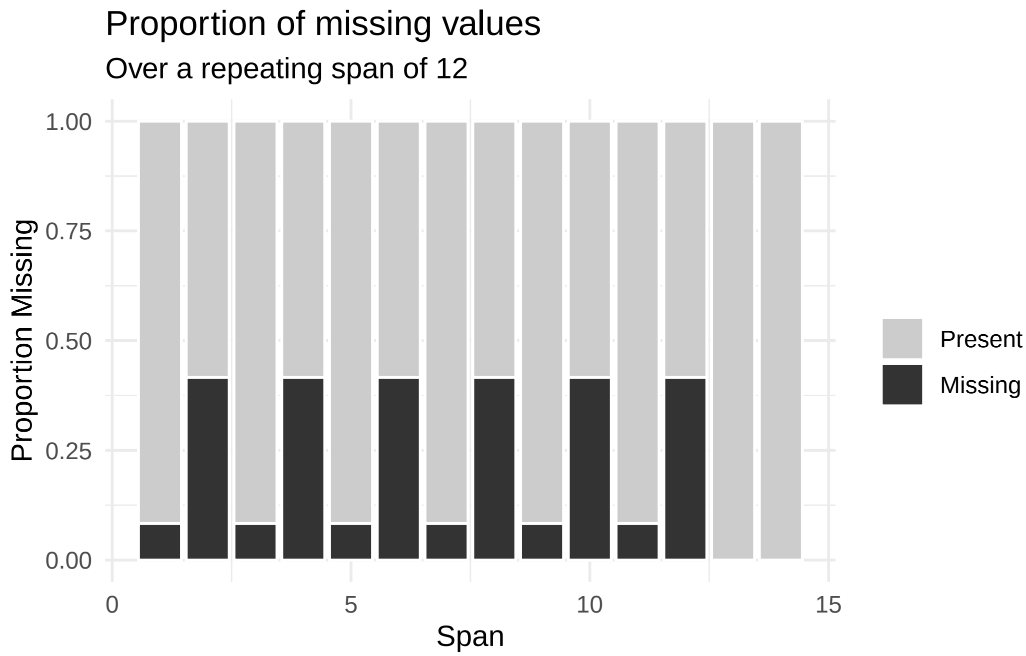
| weekday | hour | temp |
|---:|---:|---:|
| 1 | 1 | 22 |
| 1 | 2 | 24 |

| weekday | hour | temp |
|---|---|---|
| 1 | 3 | 19 |
| 1 | 4 | 24 |
| 1 | 5 | 15 |
| 1 | 6 | 19 |
| 1 | 7 | 22 |
| 1 | 8 | 12 |
| 1 | 9 | 24 |
| 1 | 10 | 13 |
| 1 | 11 | 26 |
| 1 | 12 | NA |
| 1 | 13 | NA |
| 1 | 14 | NA |
| 1 | 15 | NA |
| 1 | 16 | NA |
| 1 | 17 | NA |
| 1 | 18 | 11 |
| 1 | 19 | 11 |
| 1 | 20 | 17 |
| 1 | 21 | 14 |
| 1 | 22 | 19 |
| 1 | 23 | 21 |
| 1 | 24 | 15 |
| 2 | 1 | 13 |
| NA | 2 | 12 |

This is a fake dataset that contains information of weekday (1 through to 7), the hour of the day, and the temperature recorded that day.

We noticed before the regular "stripey" patterns of missingness in Figure @ref(fig:plot-span-streak) in the `temp` variable. Although we have information on the amount of missingness in this variable from `vis_miss`, we do not have further information on how often it occurs. Let's learn more about this by using the `gg_miss_span()` function:

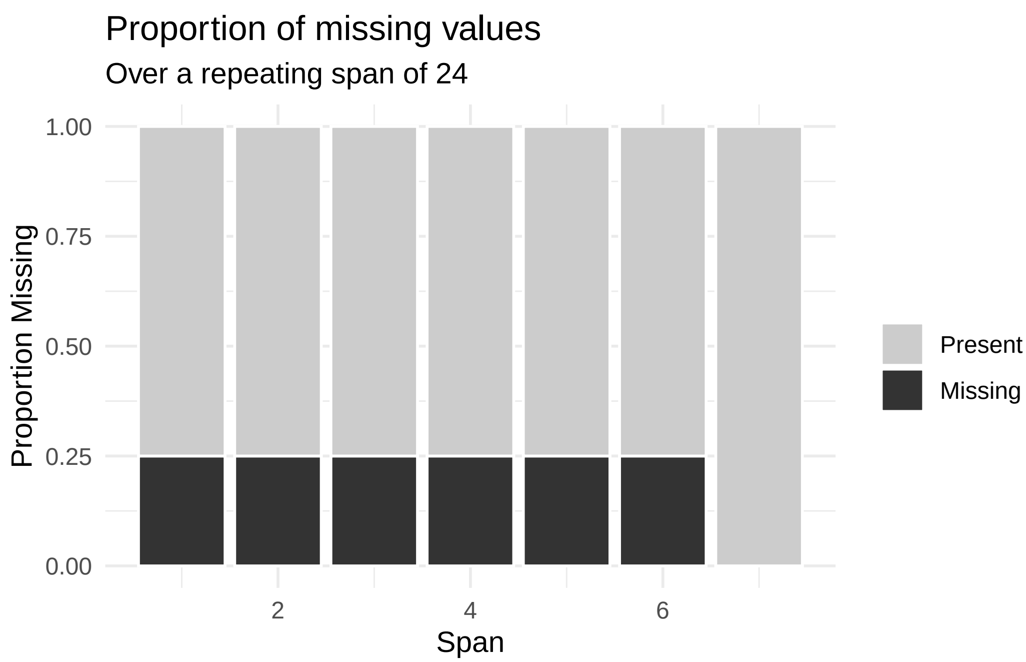
```
gg_miss_span(data = dat_span,
             var = temp,
             span_every = 12)
```




What is figure @ref(fig:gg-miss-span-dat)) showing us? We have calculated the missingness in the `temp` variable, where the missingness is calculated over some repeating span. The `span_every` argument of 12 means missing frequency and proportion will be evaluated every 12 row. So, for observations 1 to 12, then 13-24, then 25-36, and so on. Each of the spans is indicated on the x-axis; and on the y-axis, we see the *proportion of values within each span*.

How do we interpret figure @ref(fig:gg-miss-span-dat)? In this case, since the data occurs every hour, with a span of 12, we are looking at the proportion of missingness at every 12 hour interval. Notice we get a strong repeating pattern of missingness, but it seems a bit lopsided, like the proportion of missingness is bleeding over from hours 12-15, perhaps? What happens if we explore every 24 hours?

```
gg_miss_span(data = dat_span,
             var = temp,
             span_every = 24)
```




Ah! Notice that we're getting some missingness every 12 hours.

### 6.0.2 Example: Pedestrian data

Let's consider the `pedestrian` dataset in `naniar`, which contains "hourly counts of pedestrians from 4 sensors around Melbourne". See the dataset documentation (`?pedestrian`) for further details and citation.

```
glimpse(pedestrian)
```

```
Rows: 37,700
Columns: 9
$ hourly_counts <int> 883, 597, 294, 183, 118, 68, 47, 52, 120, 333, 761, 1352~
$ date_time     <dttm> 2016-01-01 00:00:00, 2016-01-01 01:00:00, 2016-01-01 02~
$ year          <int> 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 20~
$ month         <ord> January, January, January, January, January, January, Ja~
$ month_day     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
$ week_day      <ord> Friday, Friday, Friday, Friday, Friday, Friday, Friday, ~
$ hour          <int> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16~
$ sensor_id     <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,~
$ sensor_name   <chr> "Bourke Street Mall (South)", "Bourke Street Mall (South~
```

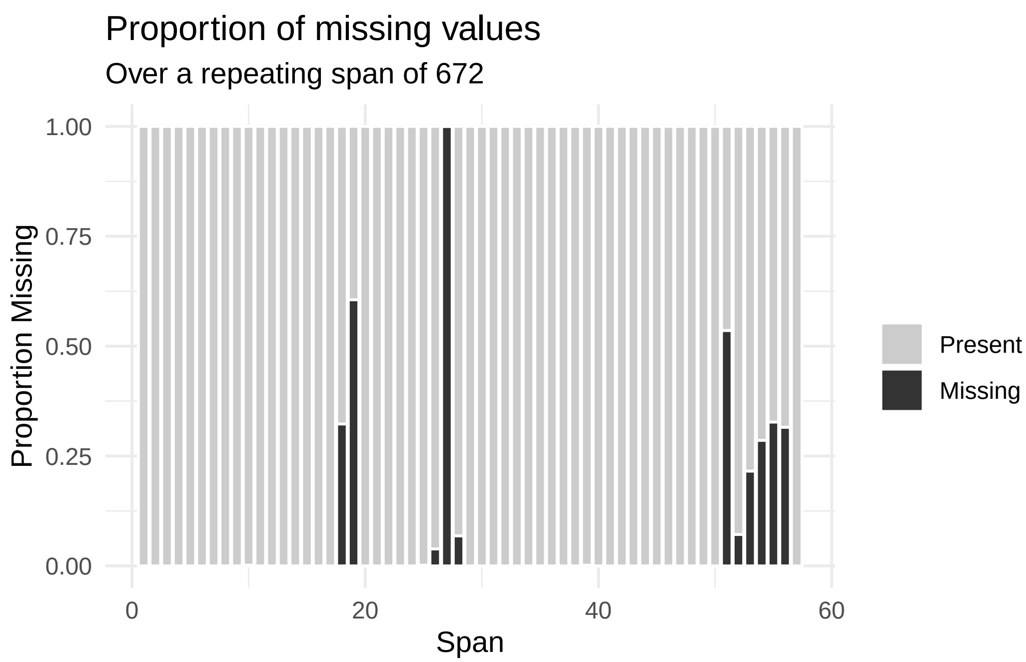
#### An aside on choosing interval size

> Importantly, observations in the `pedestrian` data are recorded at equal hourly intervals, making equal spans of interest. If data are recorded at non-equal intervals, or intermittently, investigating missingness by span may not be meaningful or informative. This is because when we explore something by a fixed interval, we want the data to have meaning at that fixed interval. If we explored our data that occurs at hourly intervals in 10 hour intervals, it might be hard to understand why 10 hours is chosen, as it might not make sense as it goes on from hour 0-10, 11-20, 21-30, and so on. Whereas if instead 12 hour or 24 hour intervals were chosen then those naturally break down into the first and second half of a day. So, all this is to say that it is important to think carefully on interval size when investigating equally-sized spans data.

Since the `pedestrian` observations *are* recording at equal intervals (and therefore spans are meaningful), it may be useful to explore the prevalence of missing values within repeated, equally-sized spans.

In the example below, missingness in the `hourly_counts` variable from `pedestrian` is calculated over repeating spans; the `span_every` argument indicates that missingness should be evaluated for each *span* of 672 observations. Why 672? Well there aer 168 hours every 7 days, and 672 hours every 4 weeks - so this shows us the amount of missing data every 4 weeks.

```
gg_miss_span(pedestrian, hourly_counts, span_every = 672)
```




How do we interpret the output above from `gg_miss_span`? We see that with a selected span size of 672, there are a total of 56 spans included, since there are 37,700 rows (672 * 56 = 37,632). Each of the spans is indicated on the x-axis; on the y-axis, the *proportion of values within each span* is indicated.

We can get the tabular format of the data put into `gg_miss_span` with `miss_var_span`:

```
miss_var_span(pedestrian,
              var = hourly_counts,
              span_every = 672)
```

```
# A tibble: 57 x 6
   span_counter n_miss n_complete prop_miss prop_complete n_in_span
          <int>  <int>      <int>     <dbl>         <dbl>     <int>
 1            1      0        672     0                 1       672
 2            2      0        672     0                 1       672
 3            3      0        672     0                 1       672
 4            4      0        672     0                 1       672
 5            5      0        672     0                 1       672
 6            6      0        672     0                 1       672
 7            7      0        672     0                 1       672
 8            8      0        672     0                 1       672
```

```
 9              9       0          672    0                 1            672
10             10       1          671    0.00149           0.999        672
# ... with 47 more rows
```

Notice that the outputs for the two examples above reveal the same information, either in visual or tabular form. Let's interpret some values to see how they align:

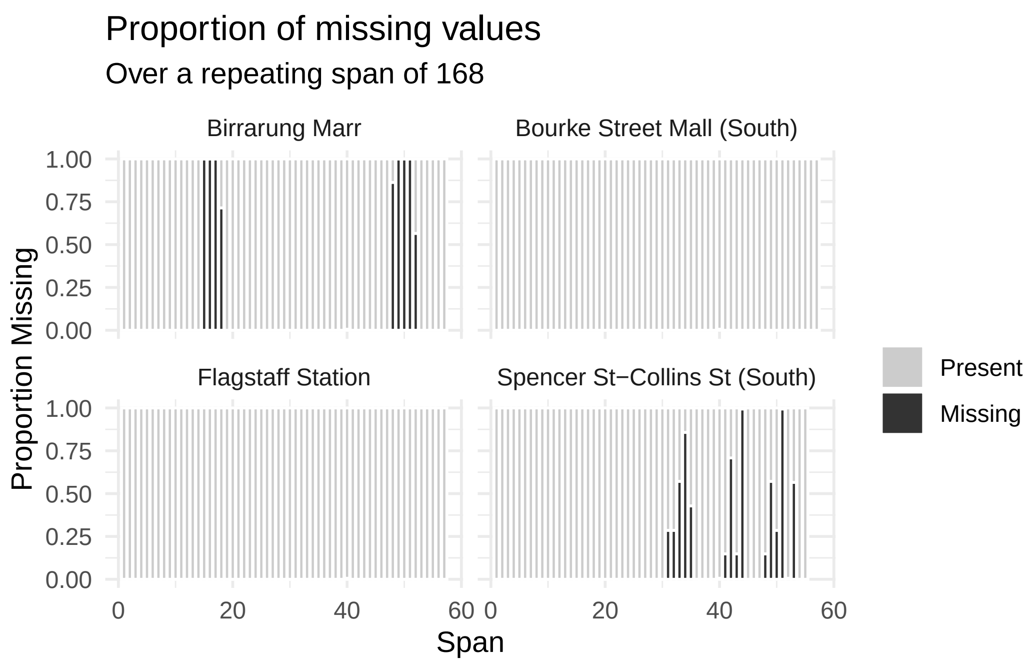
- The first column in the graph from `gg_miss_span` above does not appear to contain any missing values; that is confirmed in the first *row* from `miss_var_span` above showing that in `span_counter` 1 there are 0 missing values
- The tenth column in the `gg_miss_span` graph (`span_counter` 10) has some proportion of missing values within the span; from the `miss_var_span` output we can see that there is 1 missing values in that span (0.1% missing)

#### 6.0.2.1 Missingness within spans, by group

You can further break down missingness within spans *by group*, by faceting with `gg_miss_span` or grouping data prior to using `miss_var_span`.

For example, the above investigation of missingness for `hourly_counts` in `pedestrian`, using a span size of 168 cases (1 week), can be faceted by `sensor_name` as follows:

```
gg_miss_span(data = pedestrian,
             var = hourly_counts,
             span_every = 168,
             facet = sensor_name)
```




We can produce the analogous tabular version of that result by grouping data (`group_by(month)`) before `miss_var_summary` as follows:

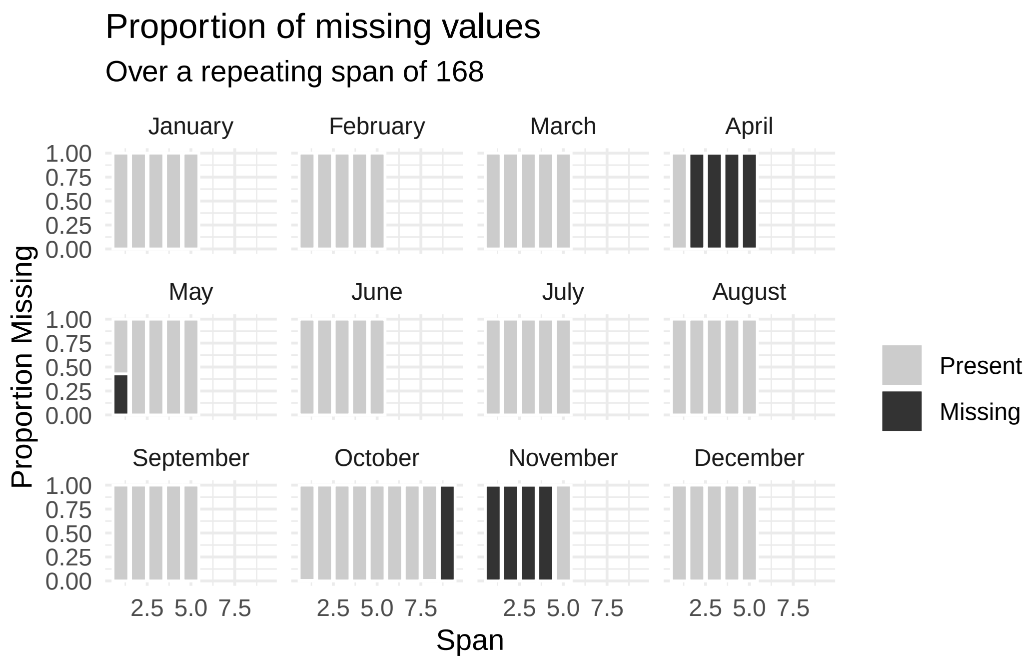
```
pedestrian %>%
  group_by(sensor_name) %>%
  miss_var_span(var = hourly_counts,
                span_every = 168)
```

```
# A tibble: 226 x 7
# Groups:   sensor_name [4]
   sensor_name  span_counter n_miss n_complete prop_miss prop_complete n_in_span
   <chr>               <int>  <int>      <int>     <dbl>         <dbl>     <int>
 1 Bourke Stre~            1      0        168         0             1       168
 2 Bourke Stre~            2      0        168         0             1       168
 3 Bourke Stre~            3      0        168         0             1       168
 4 Bourke Stre~            4      0        168         0             1       168
 5 Bourke Stre~            5      0        168         0             1       168
 6 Bourke Stre~            6      0        168         0             1       168
 7 Bourke Stre~            7      0        168         0             1       168
 8 Bourke Stre~            8      0        168         0             1       168
 9 Bourke Stre~            9      0        168         0             1       168
10 Bourke Stre~           10      0        168         0             1       168
```

# ... with 216 more rows

How do we interpret these outputs grouped by `sensor_name`? Well, this is very interesting - it looks like there is only missingness in two of the sensors, Birrarung Marr and Spencer St - Collins St (South). Within those two, it looks like some of the sensors were down a few weeks. Let's filter down to "Birrarun Marr" and explore that further, facetting by month and showing the weekly amounts of missinginess:

```
pedestrian %>%
  filter(sensor_name == "Birrarung Marr") %>%
  gg_miss_span(var = hourly_counts,
               span_every = 168,
               facet = month)
```




It looks like there was an outage from the second week in April until the first week of May, then into October and November.

**Aside: What happens to span remainders**

> What happens if you have a span that doens't fit into the number of rows of a dataset? For example, if you have spans of 50, and there are 168 rows? The final span, which would be have rows 151-168, and the proportion of missingness will be calculated as that set of data.

### 6.0.3 Streaks of missingness

Another way to explore patterns in missingness is by lengths of streaks for non-missing and missing values. For any vector (or variable in a data frame), the `miss_var_run` function in `naniar` returns the length of runs for complete and missing values. This can be particularly useful for finding repeating patterns of missingness.

For example, to explore streaks of missingness in the `hourly_counts` variable from the `pedestrians` data we can use:

```
miss_var_run(pedestrian, hourly_counts)
```

```
# A tibble: 35 x 2
   run_length is_na
        <int> <chr>
 1       6628 complete
 2          1 missing
 3       5250 complete
 4        624 missing
 5       3652 complete
 6          1 missing
 7       1290 complete
 8        744 missing
 9       7420 complete
10          1 missing
# ... with 25 more rows
```

What can we learn from the output above? There is a long initial streak (n = 6,628) of complete values for hourly_counts, the a single missing value, followed by another long streak of complete values (n = 5,250) before a more substantial streak of missingness (n = 624), and so on.

We can use `miss_var_run` with `group_by` to explore runs of missing data within months:

```
pedestrian %>%
  group_by(month) %>%
  miss_var_run(var = hourly_counts)
```

```
# A tibble: 51 x 3
# Groups:   month [12]
   month    run_length is_na
   <ord>         <int> <chr>
```

```
 1 January        2976 complete
 2 February       2784 complete
 3 March          2976 complete
 4 April           888 complete
 5 April           552 missing
 6 April          1440 complete
 7 May             744 complete
 8 May              72 missing
 9 May            2160 complete
10 June           2880 complete
# ... with 41 more rows
```

Or within sensors:

```
pedestrian %>%
  group_by(sensor_name) %>%
  miss_var_run(var = hourly_counts)
```

```
# A tibble: 38 x 3
# Groups:   sensor_name [4]
   sensor_name                run_length is_na
   <chr>                           <int> <chr>
 1 Bourke Street Mall (South)       6628 complete
 2 Bourke Street Mall (South)          1 missing
 3 Bourke Street Mall (South)       2898 complete
 4 Birrarung Marr                   2352 complete
 5 Birrarung Marr                    624 missing
 6 Birrarung Marr                   3652 complete
 7 Birrarung Marr                      1 missing
 8 Birrarung Marr                   1290 complete
 9 Birrarung Marr                    744 missing
10 Birrarung Marr                    792 complete
# ... with 28 more rows
```

or within each month for each sensor name:

```
pedestrian %>%
  group_by(month,
           sensor_name) %>%
  miss_var_run(var = hourly_counts)
```

```
# A tibble: 82 x 4
# Groups:   month, sensor_name [48]
   month     sensor_name                run_length is_na
   <ord>     <chr>                           <int> <chr>
 1 January   Bourke Street Mall (South)        744 complete
 2 February  Bourke Street Mall (South)        696 complete
 3 March     Bourke Street Mall (South)        744 complete
 4 April     Bourke Street Mall (South)        720 complete
 5 May       Bourke Street Mall (South)        744 complete
 6 June      Bourke Street Mall (South)        720 complete
 7 July      Bourke Street Mall (South)        744 complete
 8 August    Bourke Street Mall (South)        744 complete
 9 September Bourke Street Mall (South)        720 complete
10 October   Bourke Street Mall (South)         52 complete
# ... with 72 more rows
```

We can imagine questions that might arise when considering streaks of missingness: Were there changes in sampling protocols? Did the person, equipment, or study site change? Did funding get cut? Any of these might help to understand *why* values are missing, an important question when working with incomplete data and useful when deciding how to deal with missing values in analyses.

# Part III

# Cleaning missing data

# 7 Cleaning missing data

```
library(naniar)
library(dplyr)
```

## 7.1 Find and replace missing values

In previous sections, we learned how to count, summarise and visualise missing values stored as `NA`. Often, however, raw data contain missing values that have been recorded as something *other* than `NA`. These include things like characters (e.g. "missing", "N/A", or "no data") or impossible values (e.g. "-9999" for a dolphin length variable).

Always take care when working with data, *especially* if you did not collect or record it yourself:

> Never assume that all missing values are stored as `NA`

The problem is most functions assessing missingness *only recognize `NA`*, so they will not recognize other missing value inputs, such as "NA", or "missing". That means the first thing we often need to do is **search for** missing values stored as something other than `NA` in our data, then **replace those non-`NA` missing values with `NA`** so our assessments of missingness, and subsequent analyses, are accurate.

**In this section, we introduce tools and strategies to:**

- Search for missing values stored as something other than `NA`
- Replace them with `NA`

We introduce the following functions to help us:

- `miss_scan_count()`: search for missing values stored as something other than `NA` (e.g. "N/A", "-999", ".", etc.)
- `replace_with_na()`: replace non-`NA` values with `NA`

We will use a dataset called `chaos`, shown below, which contains gnarly values like plain whitespace, " ", full-stops (or periods)".", "N/A", and "missing" - all of which, in this case, *should* be stored as `NA`.

```
chaos <- tibble::tibble(
  score =   c(3L, -99L, 4L, -99L, 7L, 10L, 12L, 16L, 9L),
  grade = c("N/A", "E", "missing", "na", "n/a", " ", ".", NA, "N/a"),
  place = c(-99, 97, 95, 92, -98, "missing", 88, ".", 86)
)

knitr::kable(chaos)
```

| score | grade | place |
|---:|---|---|
| 3 | N/A | -99 |
| -99 | E | 97 |
| 4 | missing | 95 |
| -99 | na | 92 |
| 7 | n/a | -98 |
| 10 | | missing |
| 12 | . | 88 |
| 16 | NA | . |
| 9 | N/a | 86 |

**An Aside: Talk to the people who collect or curate the data**

> If you have access to the people who collect or curate the data, talk to them! It is amazing how much they can tell you about the data that you might not have ever known. You will get the most out of the conversation if you've had a look at the data first, and noted any abnormalities. Asking questions like "what did you do with missing data? How are missing values encoded? How did you collect the data? Did you summarise the data before giving it to me? Is this the most raw form of the data? Are good questions to help you get started. Also, remember to be friendly to these people. I know from experience that it can be very frustrating to have data that is poor or low quality, where missing values are deleted, or the data is summarised to the point of no variation. However it is important to keep in mind that these people who collect or curate the data are often trying to help you by saving you time summarising. Be kind, and be curious. And ask for the data in the rawest form.

### 7.1.1 Search for missing values

Before we can start replacing unexpected missing values with `NA`, we should get a sense of how big this missing data problem is by searching for those strange missing values. The `miss_scan_count` function in `naniar` allows you to search for likely records of missing values stored as something other than NA. For example, if we want to check for missing values that are input as "N/A", we can use:

```
chaos %>%
  miss_scan_count(search = list("N/A"))
```

```
# A tibble: 3 x 2
  Variable     n
  <chr>    <int>
1 score        0
2 grade        1
3 place        0
```

This returns a dataframe with two columns: "Variable" - the variables in the `chaos` data frame, and "n", the number of times that string appears in each variable. Here, we see that "N/A" appears once in the `grade` variable, and never in the `score` or `place` variables.

The `miss_scan_count` function accepts multiple arguments in the search, so you can look for all the strangely recorded missing values you like! Here we see that when searching for capital "N/A" and "N/a", there are two hits for the variable, `grade` (and still 0 for both `score` and `place`).

```
chaos %>%
  miss_scan_count(search = list("N/A",
                                "N/a"))
```

```
# A tibble: 3 x 2
  Variable     n
  <chr>    <int>
1 score        0
2 grade        2
3 place        0
```

The `naniar` package also contains two helpful datasets to explore missingness, `common_na_numbers`, and `common_na_strings`:

```
common_na_numbers
```

```
[1]    -9   -99  -999 -9999  9999    66    77    88
```

```
common_na_strings
```

```
 [1] "NA"     "N A"    "N/A"    "#N/A"   "NA "    " NA"    "N /A"   "N / A"
 [9] " N / A" "N / A " "na"     "n a"    "n/a"    "na "    " na"    "n /a"
[17] "n / a"  " a / a" "n / a " "NULL"   "null"   ""       "\\?"    "\\*"
[25] "\\."
```

These can be put inside of `miss_scan_count` and we can see we've got even more matches!

```
chaos %>%
  miss_scan_count(search = common_na_numbers)
```

```
# A tibble: 3 x 2
  Variable     n
  <chr>    <int>
1 score        2
2 grade        0
3 place        3
```

```
chaos %>%
  miss_scan_count(search = common_na_strings)
```

```
# A tibble: 3 x 2
  Variable     n
  <chr>    <int>
1 score        9
2 grade        8
3 place        9
```

You can also look for both:

```
chaos %>%
  miss_scan_count(search = c(common_na_numbers,
                             common_na_strings))
```

```
# A tibble: 3 x 2
  Variable     n
  <chr>    <int>
```

```
1 score        9
2 grade        8
3 place        9
```

Note that you do still need to carefully explore the data and metadata to get an idea of how missing values were recorded so that you don’t miss an obscure missing record. Also consider that some o these values might have other meanings - finding a match of the numbers in `common_na_numbers` might not mean they all match missing values, since, for example, you could conceivably have values -99.

**An Aside on **

> Note that in `common_na_strings`, there are some \\ for values such as `.` and `*` and `?`. This is because under the hood, `miss_scan_count` uses a thing called “regular expressions” to search for characters in the data. Briefly, regular expressions allow you to find and extract parts of text from collections of text. For example, the regular expression “`*.csv$`” means “find words that contain anything up until”.csv”, and “.csv” is also the last thing in the word. So these values, `*`, `.`, and `?` all have special meaning in regular expressions. We use \\ to “escape” the regular expression. It’s our way of saying, “No really, just look for”*“, or”.”, or “?”. Regular expressions are a really powerful tool, but can take some (sometimes a lot) of time to get your head around. Two places that provide a nice way to test out regular expressions is https://regex101.com/ and https://regexr.com/.

### 7.1.2 Replacing missing values with `NA`

Once you’ve explored and searched for missing values stored as something other than `NA`, you can replace them with `NA` using the `replace_with_na()` function. For example, in the `chaos` dataset we can replace “N/A” and “N/a” entries that appear in the `grade` variable as follows:

```
chaos %>%
  replace_with_na(replace = list(grade = c("N/A", "N/a")))
```

```
# A tibble: 9 x 3
  score grade     place
  <int> <chr>     <chr>
1     3  <NA>     -99
2   -99 "E"       97
3     4 "missing" 95
4   -99 "na"      92
5     7 "n/a"     -98
6    10 " "       missing
```

```
7     12 "."       88
8     16  <NA>     .
9      9  <NA>     86
```

The above code can be read as follows:

> Start with the `chaos` data, then within the variable `grade` replace any existing values of "N/A" and "N/a" with `NA`.

We can see this has replaced some of the missing values, but note that it only replaces the *exact* specified strings ("N/A" and "N/a") - even slight variations ("na" and "n/a") still exist.

We can even use `common_na_strings` in `replace_with_na` - but be warned! This should only be done if you really, truly, 100% for sure know that all the values in `common_na_strings` should be missing values in your data. Do not apply this without careful thought! You have been warned!

```
chaos %>%
  replace_with_na(replace = list(grade = common_na_strings))
```

```
# A tibble: 9 x 3
  score grade     place
  <int> <chr>     <chr>
1     3  <NA>     -99
2   -99 "E"       97
3     4 "missing" 95
4   -99  <NA>     92
5     7  <NA>     -98
6    10 " "       missing
7    12 "."       88
8    16  <NA>     .
9     9 "N/a"     86
```

UP TO HERE

### 7.1.3 Useful variants of `replace_with_na`

The `replace_with_na` function can be repetitive if you need to use it across many variables, for many different values. Or, for more complex cases where you might only want to replace values less than -1, or only treat character columns. To account for these situations, `naniar` borrows from `dplyr`'s *scoped variants* and extends `replace_with_na` to create three useful functions:

- `replace_with_na_all()`: operates on all variables.
- `replace_with_na_at()`: operates on a subset of selected variables
- `replace_with_na_if()`: operates on a subset of variables that fulfil a condition (e.g. only on numeric variables)

**Example:** `replace_with_na_all`

The scoped variants of `replace_with_na` follow a specific syntax. You provide a condition argument, and pass it a special function that starts with the squiggly line, tilde, `~`, and when referring to a variable, you use `.x`. For example, if we want to replace all cases of -99 in a dataset, we use `replace_with_na_all`, and write:

```
chaos %>%
  replace_with_na_all(condition = ~.x == -99)
```

```
# A tibble: 9 x 3
  score grade     place
  <int> <chr>     <chr>
1     3 "N/A"     <NA>
2    NA "E"       97
3     4 "missing" 95
4    NA "na"      92
5     7 "n/a"     -98
6    10 " "       missing
7    12 "."       88
8    16  <NA>     .
9     9 "N/a"     86
```

We can read the above code as:

> start with `chaos`, THEN `replace_with_na_all` where any variable (~.x) is equal to -99.

Extending this a bit further, we can replace values "N/A", "missing", or "na" with `NA` across all variables in `chaos` with the following:

```
chaos %>%
  replace_with_na_all(condition = ~.x %in% c("N/A", "missing", "na"))
```

```
# A tibble: 9 x 3
  score grade place
  <int> <chr> <chr>
```

```
1      3  <NA> -99
2    -99 "E"   97
3      4  <NA> 95
4    -99  <NA> 92
5      7 "n/a" -98
6     10 " "   <NA>
7     12 "."   88
8     16  <NA> .
9      9 "N/a" 86
```

We can read the code above as:

> Start with `chaos` data, THEN `replace_with_na_all` across all variables where the existing value is "N/A", "missing", or "na"

**Example:** `replace_with_na_at`

To select specific columns to apply `replace_with_na` to slected variables by name, use the scoped variant `replace_with_na_at`. For example, to only replace values with `NA` in the `place` column of `chaos`, we can use:

```
chaos %>%
  replace_with_na_at(
    .vars = "place",
    condition = ~.x %in% c("missing", "na", ".")
    )
```

```
# A tibble: 9 x 3
  score grade     place
  <int> <chr>     <chr>
1     3 "N/A"     -99
2   -99 "E"       97
3     4 "missing" 95
4   -99 "na"      92
5     7 "n/a"     -98
6    10 " "       <NA>
7    12 "."       88
8    16  <NA>     <NA>
9     9 "N/a"     86
```

We can see that those recorded missings have been replaced with `NA` *only in the `place` variable.*

**Example:** `replace_with_na_if`

The `replace_with_na_if` function allows us to replace values with `NA` in columns that satisfy a condition (e.g. if I only want to replace with `NA` in a *character* column).

For example, to replace values with `NA` in character columns in chaos, we can use the following code:

```
chaos %>%
  replace_with_na_if(
    .predicate = is.character,
    condition = ~.x %in% c("N/A", "N/a", "na", "n/a", ".", "", "missing")
    )
```

```
# A tibble: 9 x 3
  score grade place
  <int> <chr> <chr>
1     3  <NA> -99
2   -99 "E"   97
3     4  <NA> 95
4   -99  <NA> 92
5     7  <NA> -98
6    10 " "   <NA>
7    12  <NA> 88
8    16  <NA> <NA>
9     9  <NA> 86
```

Note that all of those varied records of missingness in the two character columns (`grade` and `place`) have been replaced with `NA`.

It is worthwhile to think about which records were *not* replaced with `NA` in the example above. Perhaps these were incorrectly recorded, or indicate a missing value?

Overall, the scoped variants of `replace_with_na` provide more control over which values in the data are replaced by `NA`.

### 7.1.4 Alternatives to `replace_with_na`

The `replace_with_na` function, and scoped variants, provide a high degree of control over what you replace, and over which variables. However, they can sometimes be a bit slow for larger datasets. If you do not need that level of control, and would like to have a bit more speed, several options exist to replace values with `NA`.

#### 7.1.4.1 `dplyr::na_if`

If you need to replace a single non-`NA` entry (e.g. "N/A") throughout the dataset, you can use the `dplyr::na_if` function. Note that it only works to replace a *single value*, like "N/A", and cannot handle vectors of multiple values (e.g. it breaks with `c("N/A", "na", ".")`).

The following code replaces all "missing" occurrences in `chaos` with `NA`:

```
chaos %>%
  na_if("missing")
```

```
# A tibble: 9 x 3
  score grade place
  <int> <chr> <chr>
1     3 "N/A" -99
2   -99 "E"   97
3     4  <NA> 95
4   -99 "na"  92
5     7 "n/a" -98
6    10 " "   <NA>
7    12 "."   88
8    16  <NA> .
9     9 "N/a" 86
```

You can also use `na_if` with `across`, but not to same flexibility as `replace_with_na`:

```
# in across
chaos %>%
  mutate(
    across(
      .cols = everything(),
      .fns = ~na_if(., -99)
        )
  )

chaos %>%
  mutate(
    across(
      .cols = "place",
      # note that you cannot specify multiple values to replace in `place`
      .fns = ~na_if(., c("missing")
        )
```

```
  )

chaos %>%
  mutate(
    across(
      .cols = where(is.character),
      # note again that you cannot specify multiple values to replace with NA
      .fns = ~na_if(., "N/A")
        )
  )
```

#### 7.1.4.2 Argument `na =` in `readr`

Similarly, you can replace a specified non-`NA` throughout the dataset *when reading the dataset into R* using the optional `na =` argument in `readr`.

For example, if we were reading in a theoretical .csv file in our current folder called 'hiking' that contains missing values recorded as "no data" throughout, we could read it in and *replace all "no data" with `NA`* as follows:

```
df <- read_csv("hiking.csv", na = "no data")
```

One workflow here might be to use the tools in `naniar`, `miss_scan_count(search = list("N/A"))` and perhaps `replace_with_na` to understand and check your missing value replacements, then put all of the values that are missing you have found and confirmed into the `na` argument of `read_csv`.

**An aside: `dplyr` and `across`**

> When `naniar` was written, `dplyr`'s *scoped variants* were a very new feature, but since writing, this feature has become superceded by the new `across` feature. We are still working through some bugs in `naniar` to try and make `across` work with `replace_with_na`. Although it is possible to use `na_if` to some extent with `across`, because it only accepts single values, it does not really work in the same was as `replace_with_na`. The idea with `replace_with_na` and `across` would be for it to look something like the following instead:

```
replace_with_na_all(
  data = chaos,
  condition = ~.x == -99
  )
```

```r
# in across
chaos %>%
  mutate(
    across(
      .cols = everything(),
      .fns = replace_with_na,
      condition = ~.x == -99
        )
  )

replace_with_na_at(
  chaos
  .vars = "place",
  condition = ~.x %in% c("missing", "na", ".")
  )

chaos %>%
  mutate(
    across(
      .cols = "place",
      .fns = replace_with_na,
      condition = ~.x %in% c("missing", "na", ".")
        )
  )

replace_with_na_if(
  data = chaos,
  .predicate = is.character,
  condition = ~.x %in% c("N/A", "N/a", "na", "n/a", ".", "", "missing")
  )

chaos %>%
  mutate(
    across(
      .cols = where(is.character),
      .fns = replace_with_na,
      condition = ~.x %in% c("N/A", "N/a", "na", "n/a", ".", "", "missing")
        )
  )
```

# 8 Missing, missing data: explicit and implicit missings

```
library(naniar)
library(tidyr)
```

So far, we have learned how to apply tools and strategies to explore, search for and replace missing values. We know how to search for and replace those recorded missing values masquerading as real values, including sneaky strings like "N/A", "missing", and "no record". But what if an entire row is omitted? Then there is *no record of those missing values in our data, but they are still missing*. So far, we have only explored missing values *that exist in our data*. Sounds strange, perhaps? Well these values that exist in our data as a recorded missing value, are called *explicit missing values*. They are in fact, *missing* missing values, more often called *implicit missings*.

More briefly: missing values in a dataset can either be **explicit**, meaning they are *missing but recorded*, or **implicit**, meaning that their presence is only implied based on other information (e.g. existing factor levels) in the data.

### 8.0.1 Explore implicit missings

Imagine we have tetris scores for three friends: Robin, Sam, and Blair. Their scores are recorded in the morning, afternoon, and evening, as shown below:

```
set.seed(2020-07-08)
tetris <- data.frame(
  name = c(rep("robin", 3),
           rep("sam", 2),
           rep("blair",3)),
  time = c("morning", "afternoon", "evening",
           "morning", "afternoon",
           "morning", "afternoon", "evening"),
  value = c(floor(runif(3, 0L, 1000L)),
            floor(runif(2, 20L, 1000L)),
```

```
              floor(runif(3, 850L, 1000L)))
)
```

```
knitr::kable(tetris)
```

| name | time | value |
|---|---|---|
| robin | morning | 832 |
| robin | afternoon | 86 |
| robin | evening | 897 |
| sam | morning | 93 |
| sam | afternoon | 688 |
| blair | morning | 952 |
| blair | afternoon | 954 |
| blair | evening | 955 |

Do you notice something different about one of the friends' records? Sam's score is recorded for morning and afternoon, **but their evening score is missing entirely**. Sam's evening score *is not recorded as missing - the evening record is not even there*! This becomes clearer if we spread out the data, so that we have one column for afternoon, evening, and morning.

```
tetris %>%
  pivot_wider(id_cols = name,
              names_from = time,
              values_from = value) %>%
  knitr::kable()
```

| name | morning | afternoon | evening |
|---|---|---|---|
| robin | 832 | 86 | 897 |
| sam | 93 | 688 | NA |
| blair | 952 | 954 | 955 |

Notice how there is now an `NA` indicated for Sam's evening score? The missing value we see here did not show up before - in long format, it was actually a *missing* missing value!

In this example, Sam's evening score is an *implicit missing value*.

### 8.0.2 Making implicit missings explicit

It can sometimes be useful to make implicit missing values explicit (even in long format), which we can do using the `complete` function from `tidyr`. With the `tetris` data, that looks like this:

```
tetris %>%
  tidyr::complete(name, time)
```

```
# A tibble: 9 x 3
  name  time      value
  <chr> <chr>     <dbl>
1 blair afternoon   954
2 blair evening     955
3 blair morning     952
4 robin afternoon    86
5 robin evening     897
6 robin morning     832
7 sam   afternoon   688
8 sam   evening      NA
9 sam   morning      93
```

We see that now an observation has been created for Sam's evening score, with value recored as `NA`.

What is the `complete` function actually doing? Based on the specified variables `name` and `time`, the function has identified expected combinations of those two variables across all groups (i.e., because Blair and Robin have an evening score, we expect that Sam should too) - and a new observation is created to make Sam's *implicit* missing evening score an *explicit* one that appears in the data.

Whereas the implicit missing for Sam's evening score would not be detected using the tools to count, summarize and visualize `NA` values we have learned so far, when converted to an *explicit* missing using `complete`, it would be detected because it has been populated with `NA`.

### 8.0.3 Handling explicitly missing values

Sometimes missing data is entered to help make a dataset more readable. For example, imagine if we had the following structure for our tetris data:

```r
tetris_empty <- tibble::tibble(
  name = c("robin", NA, NA,
           "sam", NA, NA,
           "blair",NA, NA),
  time = c("morning", "afternoon", "evening",
           "morning", "afternoon", "evening",
           "morning", "afternoon", "evening"),
  value = c(floor(runif(3, 0L, 1000L)),
            floor(runif(3, 20L, 1000L)),
            floor(runif(3, 850L, 1000L)))
)
```

```r
knitr::kable(tetris_empty)
```

| name | time | value |
|---|---|---:|
| robin | morning | 340 |
| NA | afternoon | 376 |
| NA | evening | 527 |
| sam | morning | 594 |
| NA | afternoon | 399 |
| NA | evening | 26 |
| blair | morning | 939 |
| NA | afternoon | 974 |
| NA | evening | 974 |

Sometimes this kind of format is used to make something more pleasant to read in a spreadsheet. Now, we happen to know something about the data structure here - that there are three records per person, at morning, afternoon, and evening. What we want to do is **fill** these missing values by populating each `NA` with the player's name that comes before it. The `fill` function from `tidyr` does just that: each `NA` in a variable is populated with the most recent *non-NA* value before (i.e., above) it.

```r
tetris_empty %>%
  tidyr::fill(name) %>%
  knitr::kable()
```

| name | time | value |
|---|---|---:|
| robin | morning | 340 |
| robin | afternoon | 376 |

| name | time | value |
|---|---|---|
| robin | evening | 527 |
| sam | morning | 594 |
| sam | afternoon | 399 |
| sam | evening | 26 |
| blair | morning | 939 |
| blair | afternoon | 974 |
| blair | evening | 974 |

This method of filling in missing values is referred to as "last observation carried forward" and is sometimes abbreviated as "locf".

> Beware: this requires that your data are carefully organized *before* using `fill`! The `fill` function does NOT predict what the entry *should be* based on other variable values or factor levels; it simply populates each missing value with the most recent non-missing value for that variable. Be very careful with this method to populate missings, and understand that it is only useful in unique cases and not a generally suggested option to replace missing values.

# Part IV

# Representing Missing Data

# 9 Representing Missing Data

We've covered how to create summaries and visualize missing values. But how do we link these summaries of missingness back to values in the data? This chapter explores two special data structures to facilitate working with missing data:

1. The Shadow Matrix
2. Nabular data

## 9.1 Motivation

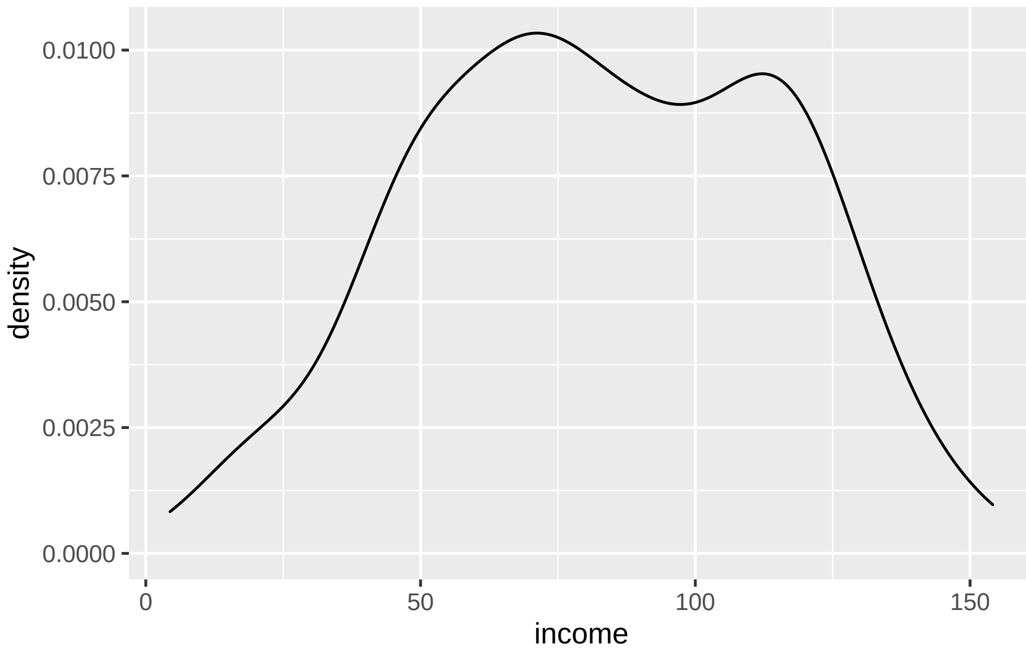
Let's imagine that we have some census data that contains two columns: income, and education.

```
Rows: 200 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): education
dbl (1): income

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.
```

| income | education |
|---|---|
| 73.13497 | NA |
| 66.78344 | high_school |
| 47.18483 | NA |
| 31.19808 | high_school |
| 64.41645 | NA |
| 51.80495 | NA |

There are some missing values in education. If we look at the distribution of income, we see that it looks like most of the values are around 70-80 thousand dollars a year.

```r
ggplot(census,
       aes(x = income)) +
  geom_density()
```




But if we create a new variable that tells us if education is missing, `education_NA`, using `if_else`. This will contain the value "NA" when `education` is missing, and "!NA" when `education` is not missing (! meaning NOT).

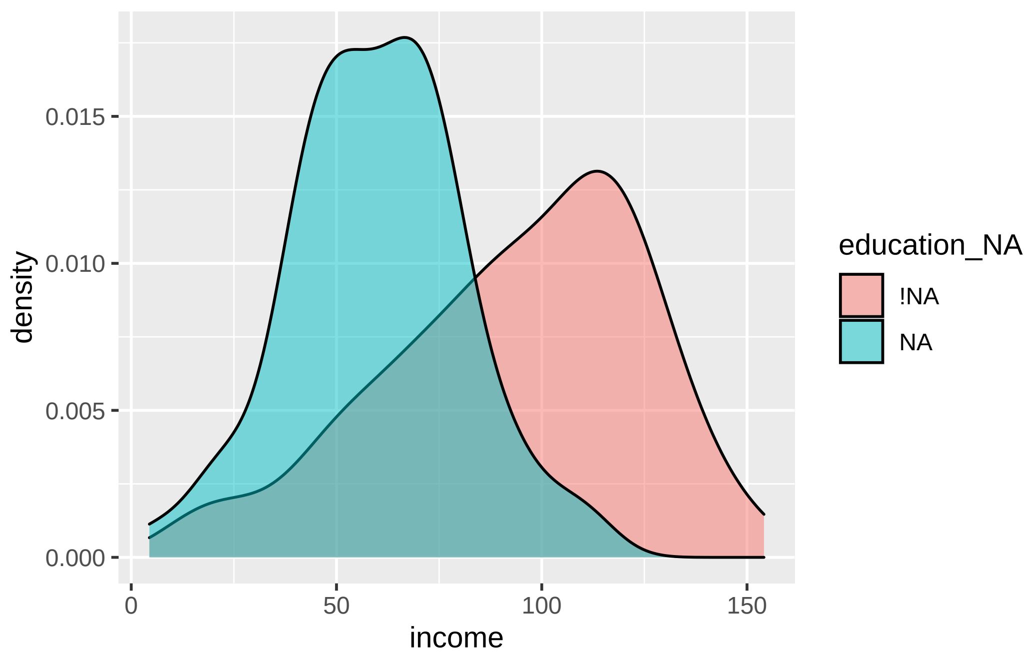
```r
census_na <- census %>%
  mutate(education_NA = if_else(condition = is.na(education),
                                true = "NA",
                                false = "!NA"))

census_na
```

```
# A tibble: 200 x 3
   income education   education_NA
    <dbl> <chr>       <chr>
 1   73.1 <NA>        NA
 2   66.8 high_school !NA
 3   47.2 <NA>        NA
```

```
 4   31.2 high_school !NA
 5   64.4 <NA>        NA
 6   51.8 <NA>        NA
 7   52.6 <NA>        NA
 8   17.5 high_school !NA
 9   61.2 <NA>        NA
10   21.2 high_school !NA
# ... with 190 more rows
```

Then this new variable `education_NA` allows us to explore how income changes depending on whether or not education is missing.

```
ggplot(census_na,
       aes(x = income,
           fill = education_NA)) +
  geom_density(alpha = 0.5)
```




We can see that indeed, your value of income does change whether your education value is missing or not.

Plots like this are really useful to explore missingness in a more principled way. `naniar` provides special data structures that facilitate this in a powerful way. This chapter introduces these special data structures, the **shadow matrix**, and **nabular data**, and demonstrates how their use in analysis.

## 9.2 The shadow matrix

We previously showed how the new variable, `education_NA` can be used to explore missing data. This variable can be thought of as the "shadow" of `education`:

```
census_na %>%
  select(education,
         education_NA) %>%
  slice(1:10) %>%
  knitr::kable()
```

| education | education_NA |
|---|---|
| NA | NA |
| high_school | !NA |
| NA | NA |
| high_school | !NA |
| NA | NA |
| NA | NA |
| NA | NA |
| high_school | !NA |
| NA | NA |
| high_school | !NA |

Creating these shadow variables is handy! But doing it for each variable, each time you want to explore missingness adds a lot of extra work. We can instead shift our focus to look at what if we turned all of the variables into shadow versions of themselves. We call this a "Shadow matrix". You can convert your data to a shadow matrix using `as_shadow()`.

```
as_shadow(census)
```

```
# A tibble: 200 x 2
   income_NA education_NA
   <fct>     <fct>
 1 !NA       NA
 2 !NA       !NA
 3 !NA       NA
```

```
 4 !NA        !NA
 5 !NA        NA
 6 !NA        NA
 7 !NA        NA
 8 !NA        !NA
 9 !NA        NA
10 !NA        !NA
# ... with 190 more rows
```

While you can get something similar by using `is.na()`

```r
is.na(census) %>% head()
```

```
     income education
[1,]  FALSE      TRUE
[2,]  FALSE     FALSE
[3,]  FALSE      TRUE
[4,]  FALSE     FALSE
[5,]  FALSE      TRUE
[6,]  FALSE      TRUE
```

This is has some shortcomings - the first being that it is now actually a matrix, not a dataframe:

```r
is.na(census) %>% head() %>% class()
```

```
[1] "matrix" "array"
```

and the second being that it is not entirely clear what TRUE means! Does it mean TRUE missing or TRUE, present? The shadow matrix from `as_shadow` returns a dataframe, and contains two features that make it easier to use in a data analysis:

1. Coordinated names: Variables in the shadow matrix gain the same name as in the data, with the suffix "_NA". This makes the variables missingness clear to refer to. It also indicates that we shift our thinking from "what is this variable's values" to "what is the missingness of this variable".

2. Clear values. The values are either !NA - "not missing", or NA - "missing". This is clearer than 1s and 0s for missing/not missing

The shadow matrix is most useful when combined with the data, which we call `nabular` data, which we now discuss.

## 9.3 Creating nabular data

To get the most out of the shadow matrix, it needs to be attached, column-wise, to the data. Putting the data in this form is referred to as **nabular** data - so called because it is a portmanteau or "NA", and "Tabular". You can create this data with **nabular()**:

```
nabular(census)
```

```
# A tibble: 200 x 4
   income education   income_NA education_NA
    <dbl> <chr>       <fct>     <fct>
 1   73.1 <NA>        !NA       NA
 2   66.8 high_school !NA       !NA
 3   47.2 <NA>        !NA       NA
 4   31.2 high_school !NA       !NA
 5   64.4 <NA>        !NA       NA
 6   51.8 <NA>        !NA       NA
 7   52.6 <NA>        !NA       NA
 8   17.5 high_school !NA       !NA
 9   61.2 <NA>        !NA       NA
10   21.2 high_school !NA       !NA
# ... with 190 more rows
```

So here we have the income values and education, and then their shadow representations - `income_NA`, and `education_NA`.

**An aside: data storage and nabular data**

It's worth mentioning that using nabular data does increase the size of your data:

```
lobstr::obj_size(census)
```

```
6.38 kB
```

```
lobstr::obj_size(nabular(census))
```

```
7.70 kB
```

```
lobstr::obj_size(riskfactors)
```

```
49.23 kB
```

```
lobstr::obj_size(nabular(riskfactors))
```

```
99.99 kB
```

> if size is an issue for you, one option could be to down sample your data. The philosophy behind exploring your data with naniar is to get a handle on the general issues of missing data first. Although speed is important, we want to make sure that these techniques work well before making them super fast. In the future we will hopefully explore some techniques for making the size of `nabular` data smaller.
>
> One way to reduce `nabular` data size is to only add `shadow` columns for values that are missing, using the `only_miss` argument in `nabular`:

```
lobstr::obj_size(riskfactors)
```

```
49.23 kB
```

```
lobstr::obj_size(nabular(riskfactors))
```

```
99.99 kB
```

```
nabular(riskfactors, only_miss = TRUE)
```

```
# A tibble: 245 x 58
   state sex      age weight_lbs height_inch   bmi marital     pregnant children
   <fct> <fct>  <int>      <int>       <int> <dbl> <fct>       <fct>       <int>
 1 26    Female    49        190          64  32.7 Married     <NA>            0
 2 40    Female    48        170          68  25.9 Divorced    <NA>            0
 3 72    Female    55        163          64  28.0 Married     <NA>            0
 4 42    Male      42        230          74  29.6 Married     <NA>            1
 5 32    Female    66        135          62  24.7 Widowed     <NA>            0
 6 19    Male      66        165          70  23.7 Married     <NA>            0
 7 45    Male      37        150          68  22.9 Married     <NA>            3
 8 56    Female    62        170          70  24.4 NeverMarri~ <NA>            0
 9 18    Male      38        146          70  21.0 Married     <NA>            2
10 8     Female    42        260          73  34.4 Separated   No              3
```

```
# ... with 235 more rows, and 49 more variables: education <fct>,
#   employment <fct>, income <fct>, veteran <fct>, hispanic <fct>,
#   health_general <fct>, health_physical <int>, health_mental <int>,
#   health_poor <int>, health_cover <fct>, provide_care <fct>,
#   activity_limited <fct>, drink_any <fct>, drink_days <int>,
#   drink_average <int>, smoke_100 <fct>, smoke_days <fct>, smoke_stop <fct>,
#   smoke_last <fct>, diet_fruit <int>, diet_salad <int>, ...
```

```
lobstr::obj_size(nabular(riskfactors, only_miss = TRUE))
```

```
85.24 kB
```

## 9.4 Data summaries with nabular data

Now that you can create nabular data, let's use it to do something useful, like calculate summary statistics based on the missingness of something else. We take the airquality data, then use `nabular()` to turn the data into nabular data.

```
nabular(airquality)
```

```
# A tibble: 153 x 12
   Ozone Solar.R  Wind  Temp Month   Day Ozone_NA Solar.R_NA Wind_NA Temp_NA
   <int>   <int> <dbl> <int> <int> <int> <fct>    <fct>      <fct>   <fct>
 1    41     190   7.4    67     5     1 !NA      !NA        !NA     !NA
 2    36     118   8      72     5     2 !NA      !NA        !NA     !NA
 3    12     149  12.6    74     5     3 !NA      !NA        !NA     !NA
 4    18     313  11.5    62     5     4 !NA      !NA        !NA     !NA
 5    NA      NA  14.3    56     5     5 NA       NA         !NA     !NA
 6    28      NA  14.9    66     5     6 !NA      NA         !NA     !NA
 7    23     299   8.6    65     5     7 !NA      !NA        !NA     !NA
 8    19      99  13.8    59     5     8 !NA      !NA        !NA     !NA
 9     8      19  20.1    61     5     9 !NA      !NA        !NA     !NA
10    NA     194   8.6    69     5    10 NA       !NA        !NA     !NA
# ... with 143 more rows, and 2 more variables: Month_NA <fct>, Day_NA <fct>
```

Note that we have the airquality variables, `Ozone`, `Solar.R`, etc., and the shadow matrix variables, `Ozone_NA`, `Solar.R_NA` and so on.

We can perform some summaries on the data using `group_by` and `summarise()` to calculate the mean of Wind speed, according to the missingness of Ozone:

```r
airquality %>%
  nabular() %>%
  group_by(Ozone_NA) %>%
  summarise(mean = mean(Wind))
```

```
# A tibble: 2 x 2
  Ozone_NA  mean
  <fct>    <dbl>
1 !NA       9.86
2 NA       10.3
```

We see that the mean values of Wind are relatively similar, but slightly higher when Ozone is missing, than when Ozone is not missing.

# 10 Exploring conditional missings with ggplot

```
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.1 --

v ggplot2 3.3.6     v purrr   0.3.4
v tibble  3.1.7     v dplyr   1.0.9
v tidyr   1.2.0     v stringr 1.4.0
v readr   2.1.2     v forcats 0.5.1

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

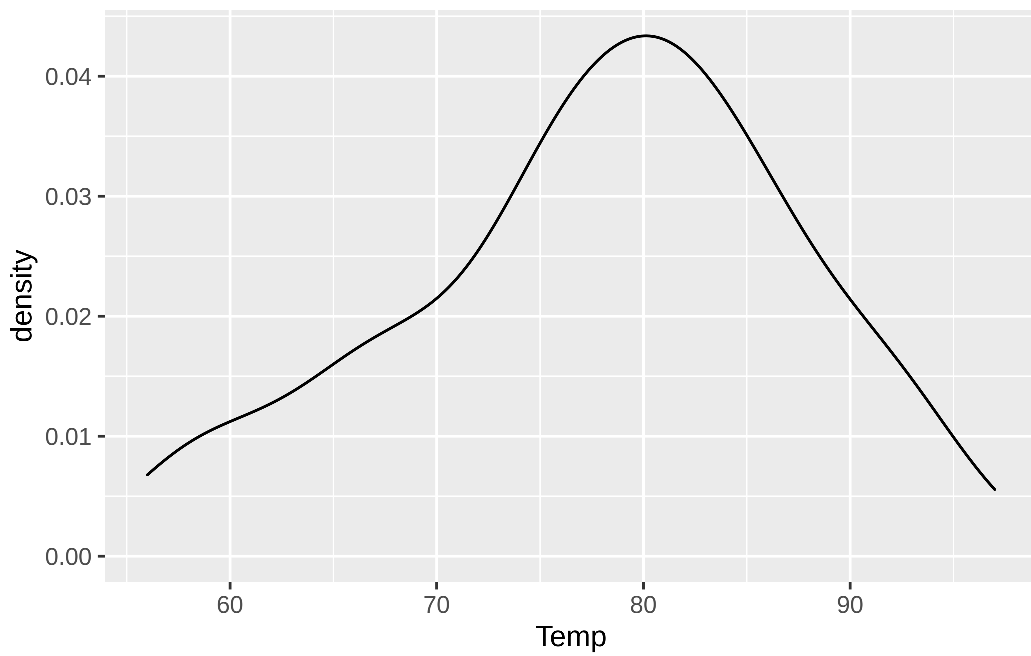
```
library(naniar)
```

Now that we've explored some ways to summarise data using nabular data, we are going to explore how you can use `nabular` data to explore how variables vary as other variables go missing. We'll demonstrate this using ggplot, showing how to visualise densities, boxplots, and some ways of creating multiple plots, for each type of missingness.

## 10.1 Visualizing missings using densities

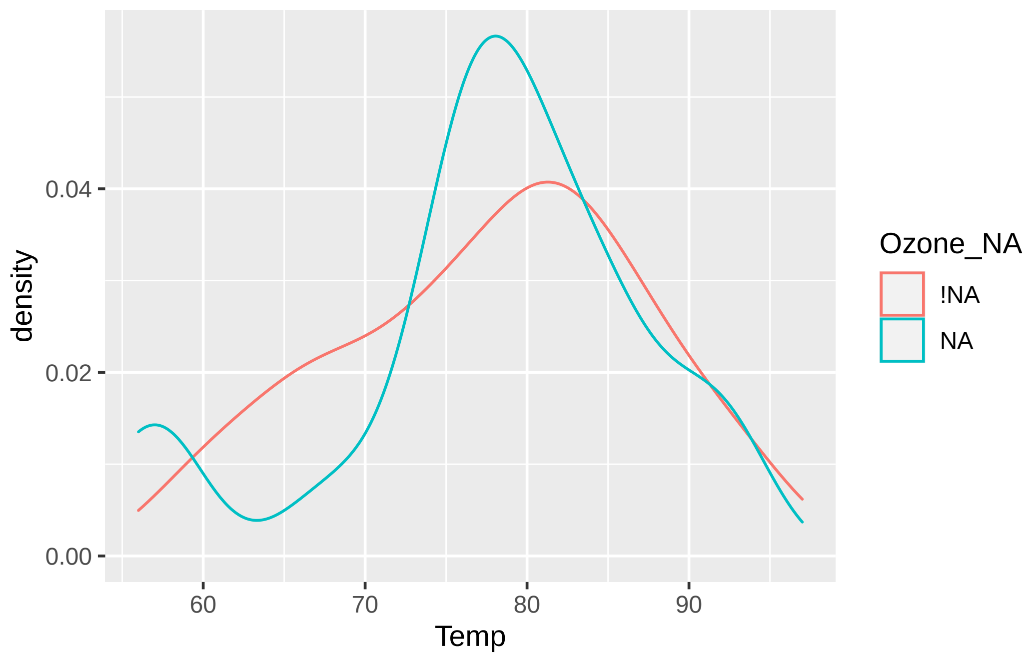
To begin, we can look at the distribution of temperature using ggplot, placing `Temp` on the X axis, and then using `geom_density()` to visualise temperature as a density, or a distribution.

```
ggplot(airquality,
       aes(x = Temp)) +
  geom_density()
```




To explore how temperature changes when ozone is missing, we create the `nabular` data with `nabular()`, and then add in our aesthetics, `colour = Ozone_NA`.

```
airquality %>%
  nabular() %>%
  ggplot(aes(x = Temp,
             color = Ozone_NA)) +
  geom_density()
```

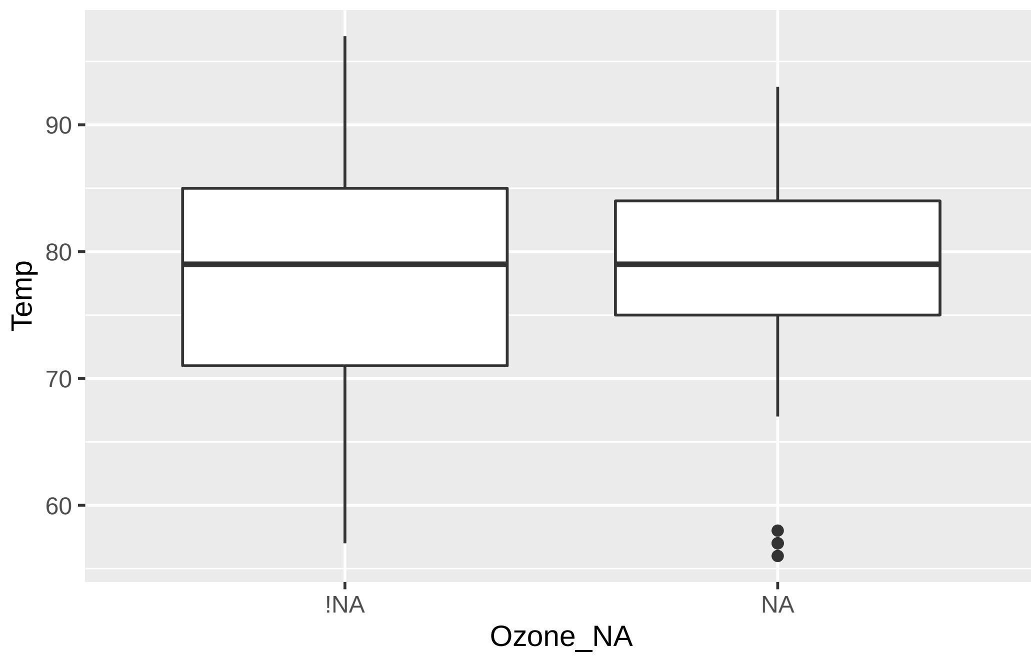



This now splits the density into two densities, one for temperature when ozone is present, and one for temperature when ozone is absent. This shows us that the values of temperature don't change much when ozone is present or absent.

## 10.2 Visualizing missings using boxplots

Similarly, you can use boxplots to explore missing data, by putting the missingness that you would like to explore by on the x axis (`Ozone_NA`), and temperature on the y axis, then using `geom_boxplot()`.

```
airquality %>%
  nabular() %>%
  ggplot(aes(x = Ozone_NA,
             y = Temp)) +
  geom_boxplot()
```



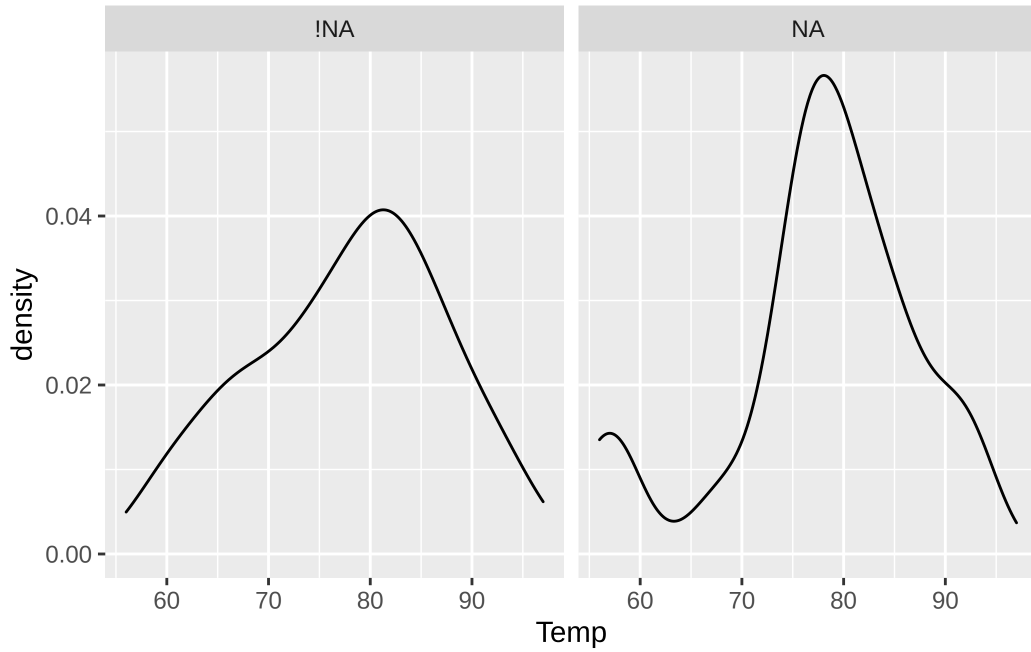

What can we learn from this? The values of temperature are similar when ozone is missing versus not missing. However, there is generally less variation for temperature when ozone is missing, but there are also some temperature outliers.

## 10.3 Visualizing missings using facets

We can visualise two densities for temperature according to the missingness of ozone. This is similar to the previous density visualisation, except the densities are not overlaid, and are faceted - they are in separate plots.

A similar visualisation to the previous visualisation of densities can be made using facets. Here, we use `nabular` data to create a density plot, using `facet_wrap(~Ozone_NA)`.

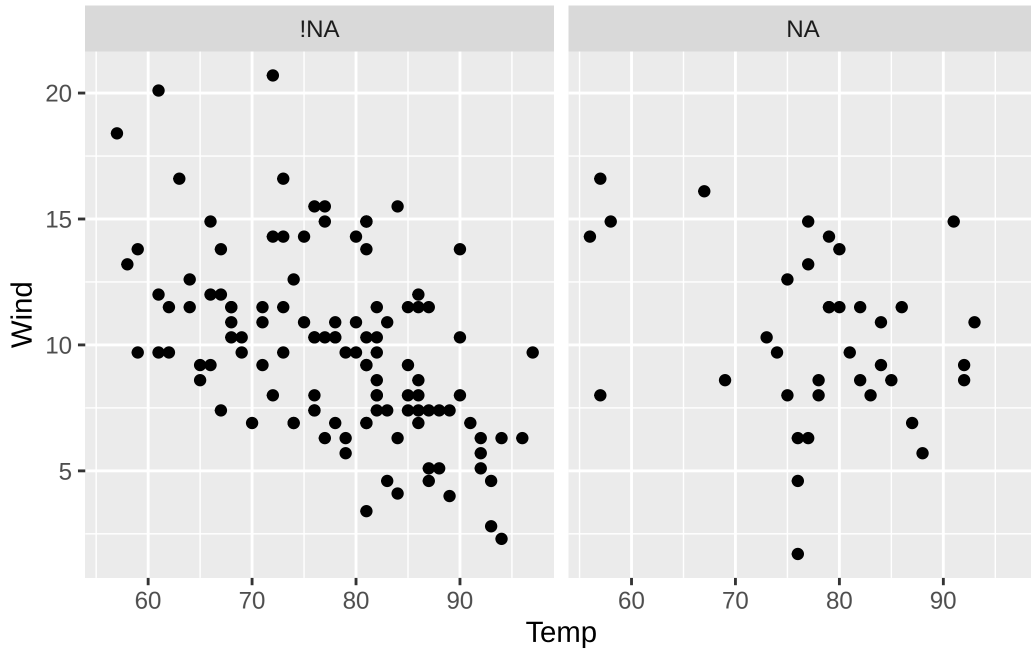
```
airquality %>%
  nabular() %>%
  ggplot(aes(x = Temp)) +
  geom_density() +
  facet_wrap(~Ozone_NA)
```




Splitting by facet can be useful if you want to compare different types of visualisations.

You can look at two scatterplots, facetting by the missingness of Ozone using `Ozone_NA`, for the values temperature and wind.

```
airquality %>%
  nabular() %>%
  ggplot(aes(x = Temp,
             y = Wind)) +
  geom_point() +
  facet_wrap(~Ozone_NA)
```

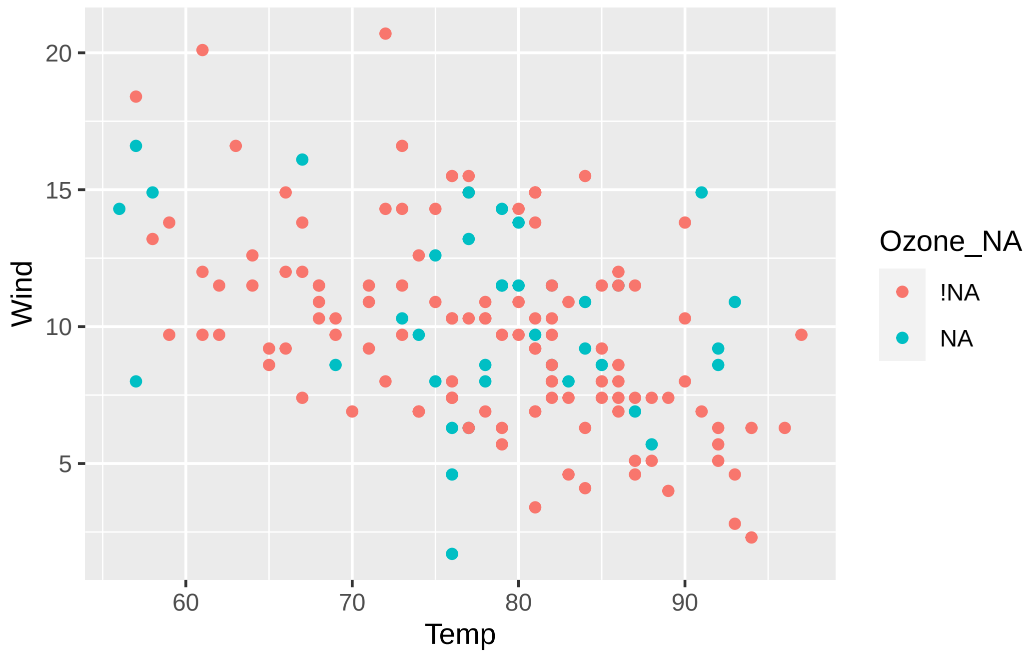



Note there are fewer wind and temperature scores when ozone is missing, and that these tend to occur for temperatures over 70 and wind speeds over 5. Overall, the values of wind and temperature when ozone is missing seem similar to when ozone is present.

## 10.4 Visualizing missings using colour

Equivalently to the previous facetted plot, you can visualise the points according to whether they are missing.

```
airquality %>%
  nabular() %>%
  ggplot(aes(x = Temp,
             y = Wind,
             color = Ozone_NA)) +
  geom_point()
```

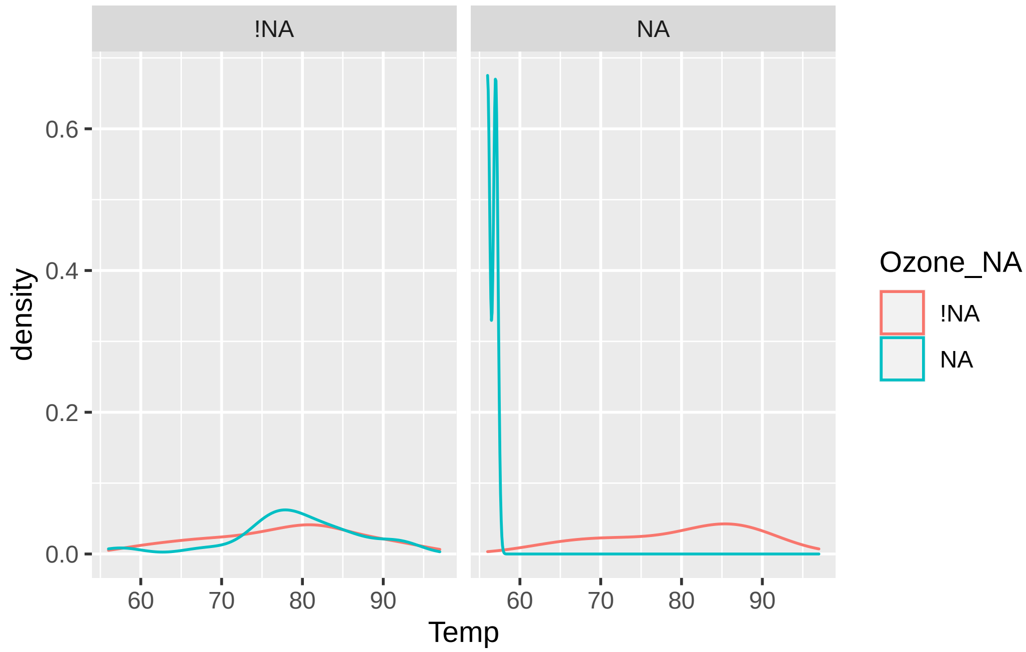



This overlays the points rather than creating separate plots. This can sometimes help make comparisons easier, although this is not always the case. In the example above I cannot see any clear pattern in these points.

## 10.5 Adding layers of missingness

A useful advantage to using `facet` to split by missings is that this allows you to look at another condition of missingness. For example, create two plots by the missingness of solar radiation, and then colour the densities by missingness of ozone.

```
airquality %>%
  nabular() %>%
  ggplot(aes(x = Temp,
             color = Ozone_NA)) +
  geom_density()  +
  facet_wrap(~Solar.R_NA)
```




This shows us that there isn't much difference in temperature when solar radiation isn't missing, but when solar radiation is missing, the temperatures are quite low!

Now that we've covered some methods for visually exploring missing data using `nabular` data and `ggplot2`, it's time to practice using this on some other data.

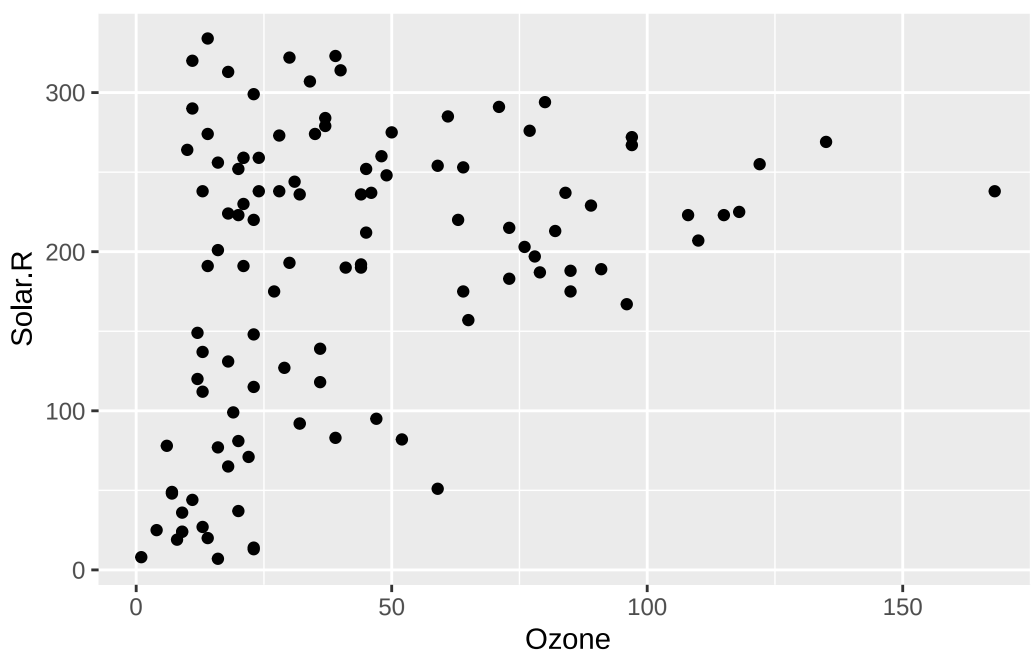
# 11 Visualizing missingness across two variables

We have previously discussed the use of **nabular** data, a way to represent missing data alongside the data itself. This data structure underpins how **naniar** performs data visualisation and summaries. This chapter discusses how to use the **nabular** data structure with data visualisation to further explore **why** data could be missing, looking across two variables.

If you want to explore two variables in a dataset, a scatterplot is a natural graphic to show. Let's explore ozone and solar radiation like so:

```
ggplot(airquality,
       aes(x = Ozone,
           y = Solar.R)) +
  geom_point()
```

```
Warning: Removed 42 rows containing missing values (geom_point).
```



However, note the warning message:

```
Warning message:
Removed 42 rows containing
missing values (geom_point).
```

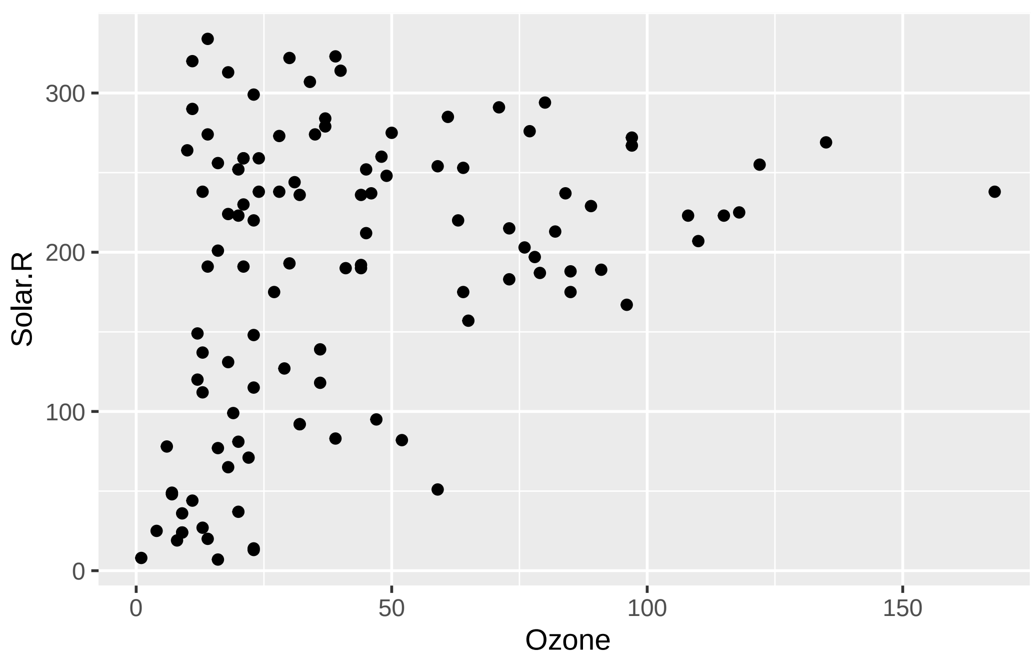
What? What does this mean? Why would ggplot do this? Well, it turns out that it's really nice that `ggplot2` provides this warning, since removing missing values is often done in modelling and other graphics without you being made aware of it.

So, how do you visualise those missing values? How does visualising *missingness* make sense? This is the focus of this chapter.

### 11.0.1 The problem of visualizing missing data in two dimensions

```
ggplot(airquality,
       aes(x = Ozone,
           y = Solar.R)) +
  geom_point()
```

```
Warning: Removed 42 rows containing missing values (geom_point).
```



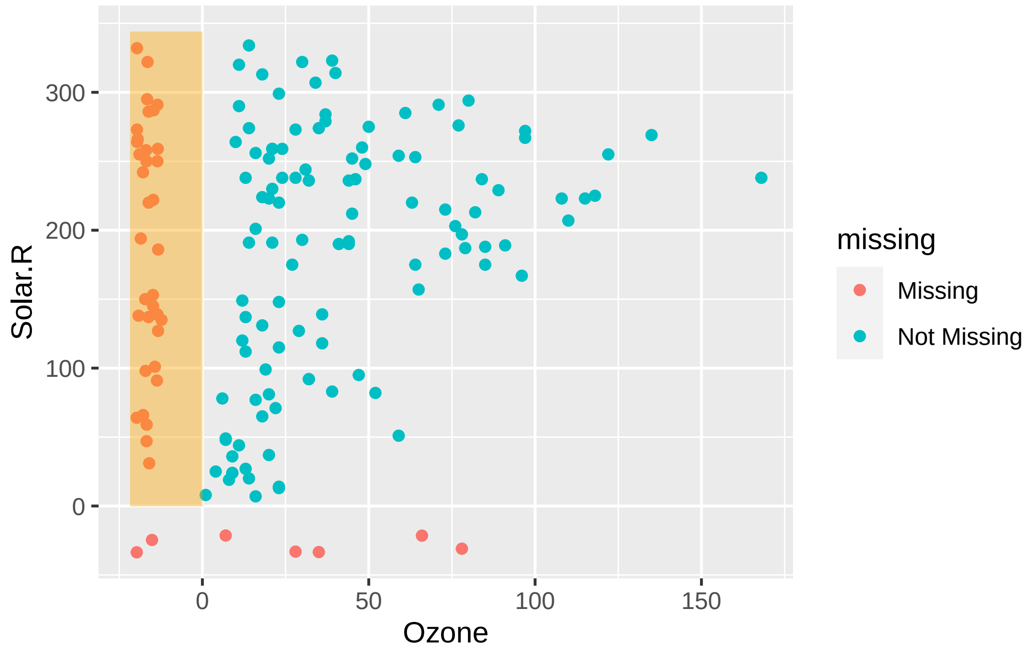
The problem with visualising a scatterplot when the data has missing values is that it removes any observations - entire rows - that have missing values. `ggplot2` is actually very nice here and gives a warning that missing values are being dropped. The same cannot be said of other all functions in R!

### 11.0.2 Introduction to `geom_miss_point()`

```
gg_miss_point <- ggplot(airquality,
       aes(x = Ozone,
           y = Solar.R)) +
  geom_miss_point()
```

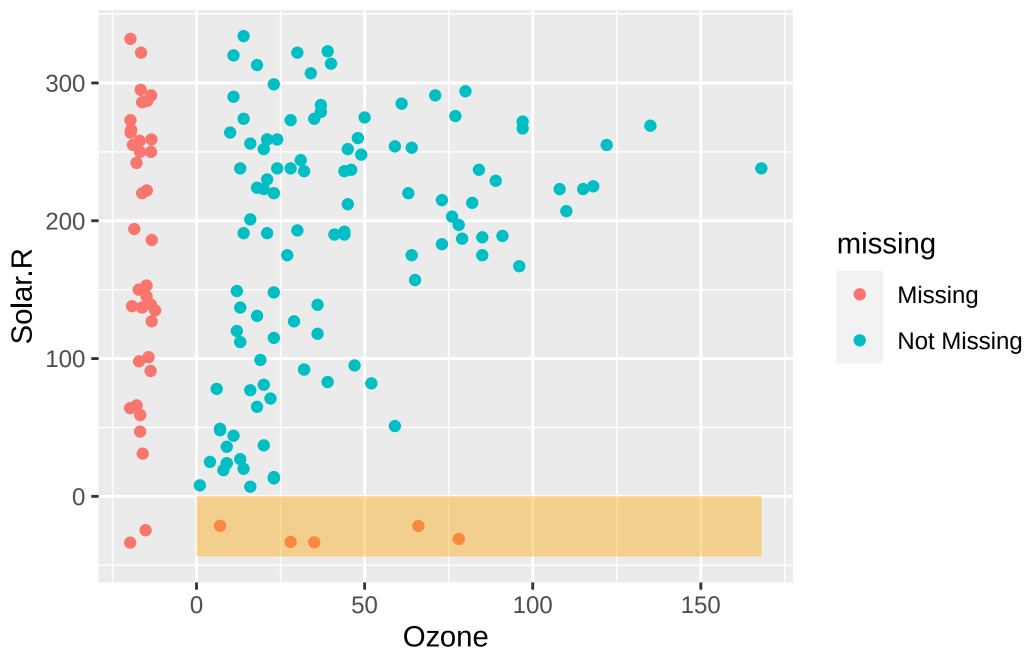
To explore the missings in a scatter plot, we can use `geom_miss_point()`. `geom_miss_point()` visualises the missing values by placing them in the margins.

```
airquality_rect <- airquality %>%
  as_tibble() %>%
  impute_below_at(.vars = c("Ozone", "Solar.R")) %>%
  summarise(xmin = min(Ozone) + min(Ozone)*0.1,
            xmax = 0,
            ymin = 0,
            ymax = max(Solar.R) + 10)

gg_miss_point +
  geom_rect(data = airquality_rect,
            inherit.aes=FALSE,
            aes(xmin=xmin, xmax=xmax,ymin=ymin,ymax=ymax),
            alpha = 0.4,
            fill = "orange")
```



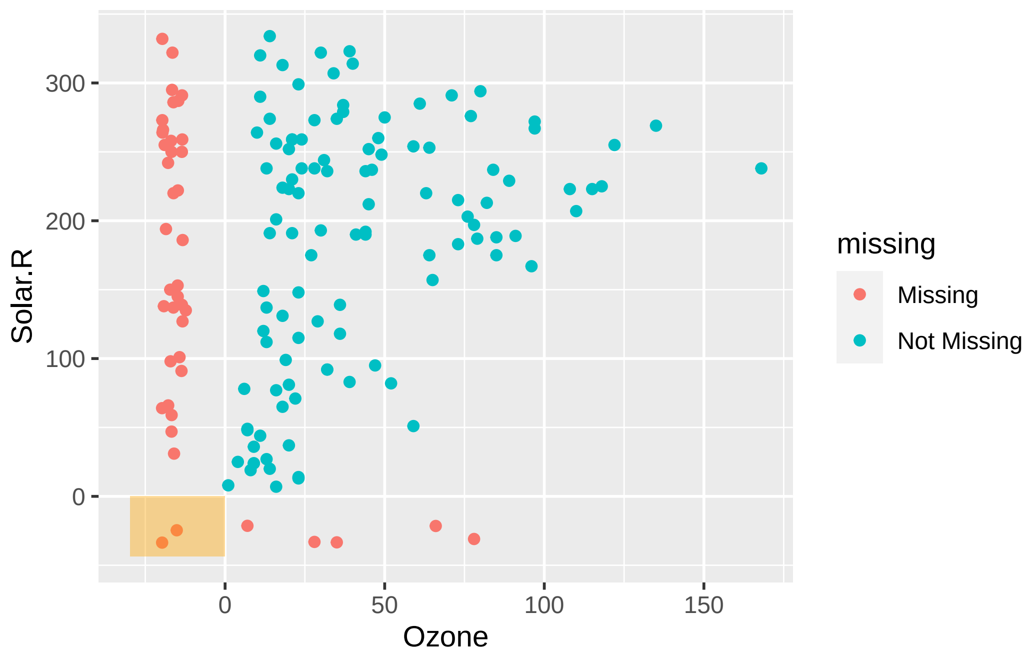

On the left in the highlighted orange section red we can see the values of solar radiation when ozone is missing. This shows us that the values of solar radiation are reasonably uniform.

```
airquality_rect <- airquality %>%
  as_tibble() %>%
  impute_below_at(.vars = c("Ozone", "Solar.R")) %>%
  summarise(xmin = 0,
            xmax = max(Ozone),
            ymin = min(Solar.R) - 10,
            ymax = 0)

gg_miss_point +
  geom_rect(data = airquality_rect,
            inherit.aes=FALSE,
            aes(xmin=xmin, xmax=xmax,ymin=ymin,ymax=ymax),
            alpha = 0.4,
            fill = "orange")
```




The values of ozone when Solar.R is missing are shown in red on the bottom, this shows us that the missing values tend to occur at lower values of ozone.

```
airquality_rect <- airquality %>%
  as_tibble() %>%
  impute_below_at(.vars = c("Ozone", "Solar.R")) %>%
  summarise(xmin = min(Ozone) - 10,
            xmax = 0,
            ymin = min(Solar.R) - 10,
            ymax = 0)

gg_miss_point +
  geom_rect(data = airquality_rect,
            inherit.aes=FALSE,
            aes(xmin=xmin, xmax=xmax,ymin=ymin,ymax=ymax),
            alpha = 0.4,
            fill = "orange")
```




In the bottom left we show cases where there are missings in both ozone and solar radiation. To explain how and why this visualisation works, we are going to take a brief moment to unpack the data transformation that occurs here.

#### 11.0.2.1 Aside: How geom_miss_point() works

`geom_miss_point` performs a transformation on the data and actually imputes (fills in, replaces) the values that are missing. Under the hood, the data is represented like so, for the ozone data:

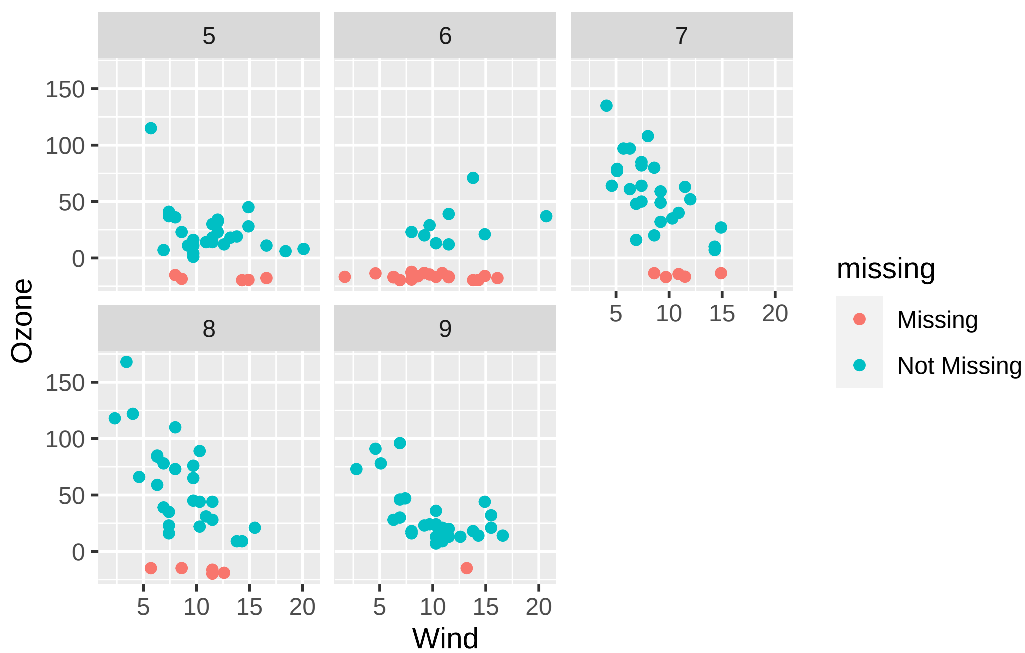
| Ozone | Ozone_shift | Ozone_NA |
|---:|---:|:---|
| 41 | 41.00000 | !NA |
| 36 | 36.00000 | !NA |
| 12 | 12.00000 | !NA |
| 18 | 18.00000 | !NA |
| NA | -19.72321 | NA |
| 28 | 28.00000 | !NA |

Notice that we have our `nabular` data here - with `Ozone` and `Ozone_NA`. We also have a new column, `Ozone_shift`. This contains the imputed data. This data is imputed **10% below the minimum value of ozone**. To keep track of which values were imputed, we can use the Ozone_NA column! We'll come back to this idea of tracking missing values in the next chapter.

### 11.0.3 Exploring missingness using facets

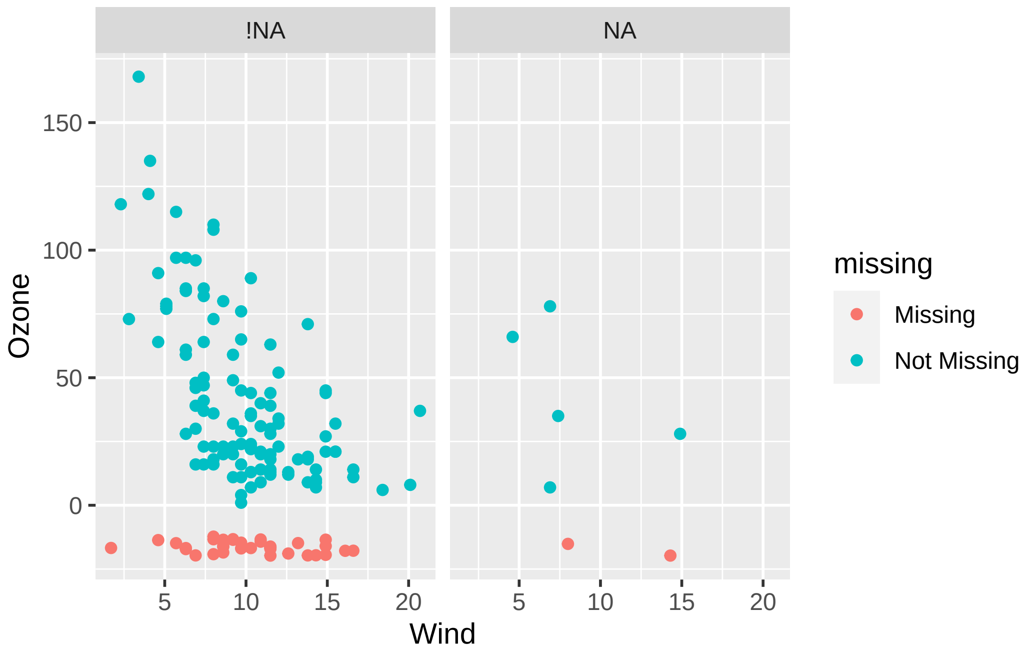
Because geom_miss_point() is a defined ggplot2 geometry, it behaves like any other ggplot. This means, you can use ggplot features like facets, to further explore your missing data. For example, you can facet by Month, to explore how the missingness changes over month:

```
ggplot(airquality,
       aes(x = Wind,
           y = Ozone)) +
  geom_miss_point() +
  facet_wrap(~Month)
```




You can even use nabular data from the previous lesson, and explore the missingness by another variable being missing. For example, you can explore how the missingness changes when solar radiation is missing.

```
airquality %>%
  nabular() %>%
  ggplot(aes(x = Wind,
             y = Ozone)) +
    geom_miss_point() +
    facet_wrap(~Solar.R_NA)
```



# Part V

# Mechanisms of Missingness

# 12 Mechanisms of missingness

```r
library(naniar)
library(readr)
library(dplyr)
library(mice)
library(here)
library(tidyverse)
```

Once we have explored and cleaned up our messy missing data so that they are consistently stored as `NA` throughout, we need to dig further into missingness to responsibly decide on next steps. However, before further analysis, we need to ask and answer questions, such as:

- How should we deal with missing values (e.g. should we delete cases, or impute values)?
- How might that decision impact our analyses and outcomes?

To answer these questions, we need to understand and explore **mechanisms of missingness**.

Mechanisms of missingness answer the question "*Why* are values missing?". For example, it could be that as your income increases, you might be less likely to report how much you paid in tax on a survey. So increased income leads to increasing missingness. Answering this question with certainty is **really hard**, and sometimes, impossible. We need to investigate missing data dependence, however, to inform decisions about dealing with missing values.

In this chapter, we introduce three mechanisms of missingness:

- MCAR - Missing Completely at Random
- MAR - Missing At Random
- MNAR - Missing Not At Random

Then, we explore and compare how those mechanisms of missingness might appear in missing data exploration using `naniar` functions for data visualization introduced in previous sections.

## 12.1 Missing completely at random (MCAR)

Missing completely at random, or MCAR, is missingness that has no association with any data you have observed, or not observed. In other words, the cause of the missingness can be considered *truly random*, and unrelated to observed or unobserved variables meaningful to the data and your analyses.

For example, imagine you are a tornado researcher. You are determined to deploy small devices into a tornado that, when suspended in the tornado, will record windspeeds and dynamics (yes - this is the plot of the classic film *Twister* starring Bill Paxton). One day while driving to try to launch your devices, your car runs out of gas, and you are unable to obtain windspeed readings. Those unrecorded windspeeds show up as `NA` in the dataset for that tornado. In this case, the *cause of the missingness* (running out of gas) is *unrelated to tornado windspeeds* - it can be considered a truly "random" cause of missingness, or missing completely at random (MCAR).

An important distinction: MCAR **does not mean there is "no reason" for missingness**. In this example, windspeed is missing for this tornado because you ran out of gas. It is still MCAR because the cause of missingness is unrelated to tornado windspeed in a meaningful way.

**Critical thinking:** Imagining that you are the tornado researcher in the example above, what other hypothetical causes may result in tornado windspeeds being missing completely at random (MCAR)?

### 12.1.1 How might MCAR appear in data?

A hypothetical example of how we might **want** MCAR to appear for the `max_windspeed` variable is shown below:

| date | severity | ave_temp_f | daily_precip_mm | max_windspeed_mph | notes |
|---|---|---|---|---|---|
| 7/22/20 | ef2 | 84 | 94 | 124 | NA |
| 8/9/20 | ef3 | 79 | 52 | 130 | NA |
| 6/15/20 | ef1 | 73 | 71 | 109 | NA |
| 9/18/20 | ef1 | 86 | 43 | 94 | NA |
| 10/5/19 | ef0 | 71 | 18 | 75 | NA |
| 10/15/19 | ef1 | 90 | 57 | NA | Ran out of gas; could not deploy |
| 9/8/19 | ef0 | 82 | 22 | 81 | NA |
| 8/17/18 | ef4 | 80 | 102 | 196 | NA |
| 8/26/18 | ef1 | 73 | 53 | NA | Team unavailable; could not deploy |
| 9/2/17 | ef2 | 78 | 39 | 114 | NA |

| date | severity | ave_temp_f | daily_precip_mm | max_windspeed_mph | notes |
|---|---|---|---|---|---|
| 9/15/16 | ef5 | 85 | 164 | 208 | NA |

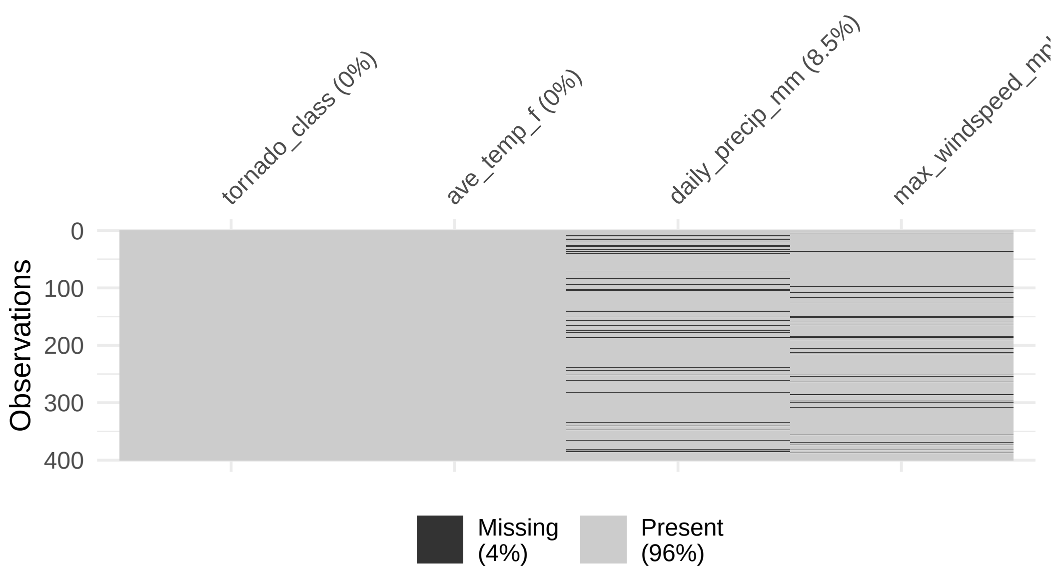
In the dataset above, we see two missing values (`NA`) in the `max_windspeed` column. For each, the comments in the `notes` column describe reasons for missingness that are unrelated to tornado windspeeds, and can thus be considered MCAR.

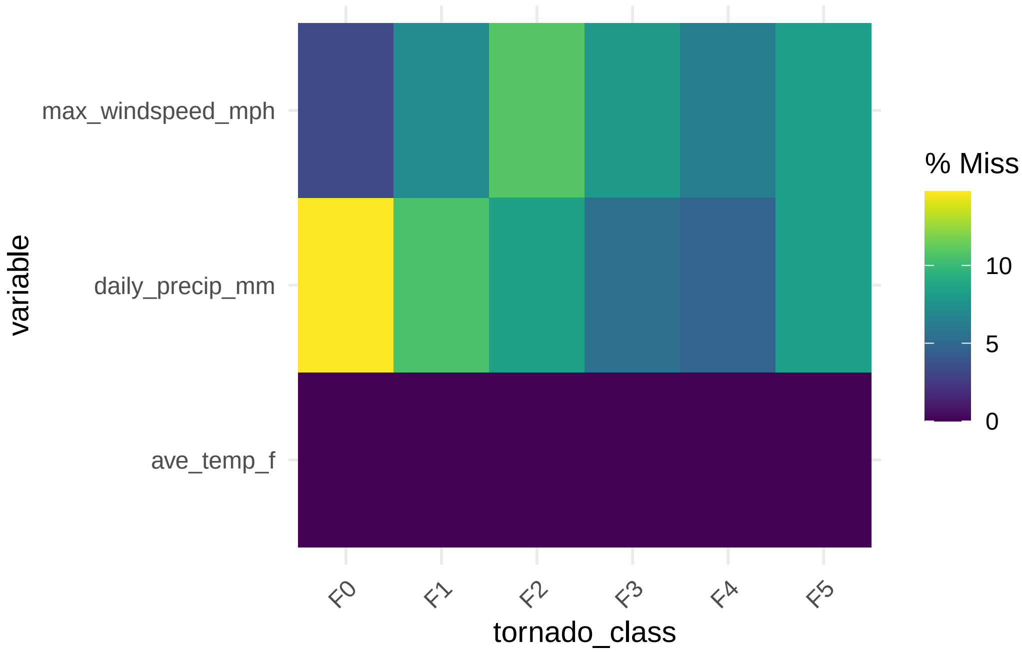
**Aside on note keeping**

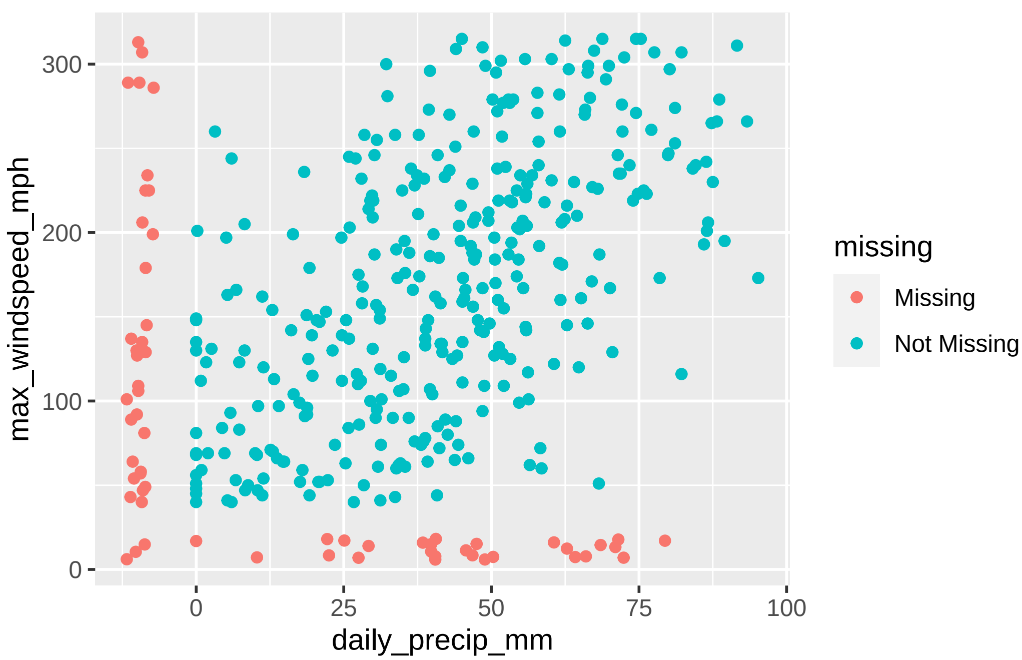
> The only reason we would definitively know missingness in windspeed is MCAR is due to the `notes` variable included. Keep this in mind when collecting your own data: taking contemporaneous notes about data collection, obstacles, etc. can be very useful when trying to determine *why* values are missing.

More often, we do not have notes explaining each missing value. If that is the case, how might we expect MCAR to appear in a larger dataset? Here, we again use a **theoretical dataset**, `twister`, for tornadoes to obviate how missing mechanisms *might* appear.

```
Warning: `gather_()` was deprecated in tidyr 1.2.0.
Please use `gather()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```




% Miss
10
5
0
max_windspeed_mph
daily_precip_mm
ave_temp_f
variable
F0
F1
F2
F3
F4
F5
tornado_class


300
200
100
0
max_windspeed_mph
0
25
50
75
100
daily_precip_mm
missing
Missing
Not Missing

#### 12.1.1.1 MCAR Implications

We can deal with missing values in a number of ways, but here we focus on two broad approaches: delete observations (rows) containing missing values for variables included in analyses, or impute (manufacture) data to "fill in" the missing values with reasonable values.

If missing values are MCAR, deleting observations containing missing values will not bias results, but reduces sample size (sometimes substantially). We call deleting entire rows, *listwise deletion*, **if you decide to use listwise deletion, make sure to check how many observations are included in your analysis.** Ideally do not delete unless there is less than 5% data loss. But really, you should be imputing your data always.

## 12.2 Missing at random (MAR)

Missing at random (MAR) occurs when missingness depends on data you **have** observed, but not on unobserved data.

Returning to our Twister tornado example: Imagine that you are again driving to release your wind speed devices into a tornado. Due to heavy rainfall, however (for which you *do* have data), several river crossings are flooded and you are unable to safely approach the tornado. Therefore, missingness in wind speed is due to *another recorded variable in the data* (rainfall, recorded as `daily_precip_mm`).

In this case, wind speed is *Missing at Random* because it is dependent on another recorded variable.

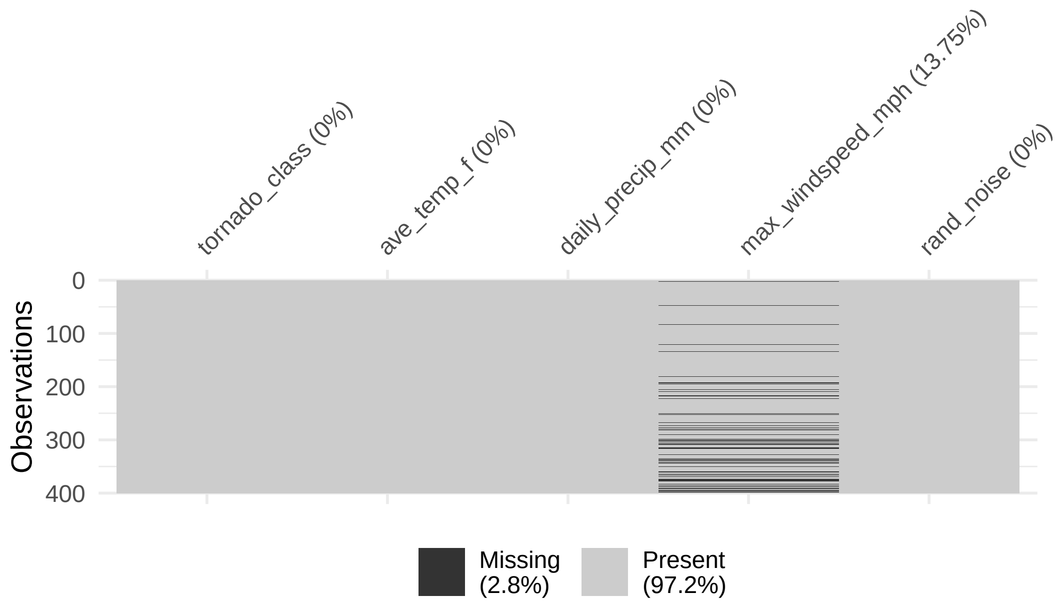
**An Aside: Naming missingness**

> If you are thinking "Missing At Random (MAR) seems like a bad name - it is not random at all! The missingness is impacted by another recorded variable!" you are in abundant company. A number of people have called for this mechanism to be renamed as "Missing *conditionally* at random" instead, but so far the change has not gained widespread traction. At another meeting, "Missing For … Reasons" was also proposed, but was not specific enough. (Joking).

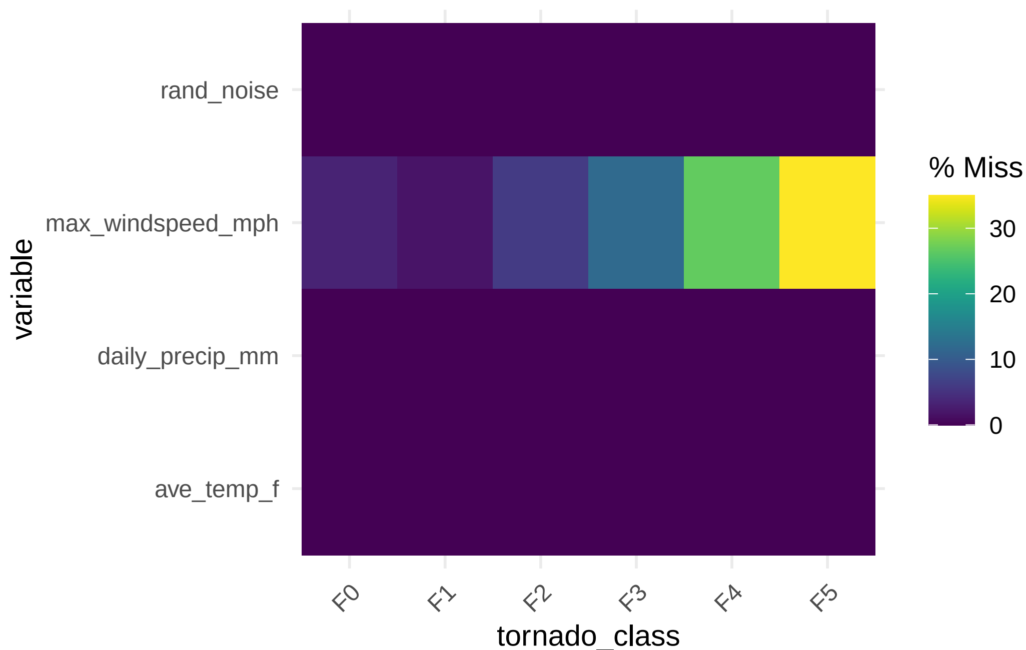
**Critical thinking:** Imagine you are the tornado researcher in the example above, what other hypothetical causes may result in tornado wind speeds being missing at random?

Let's explore a different (also theoretical) dataset with twister recordings that might indicate values that are missing at random.

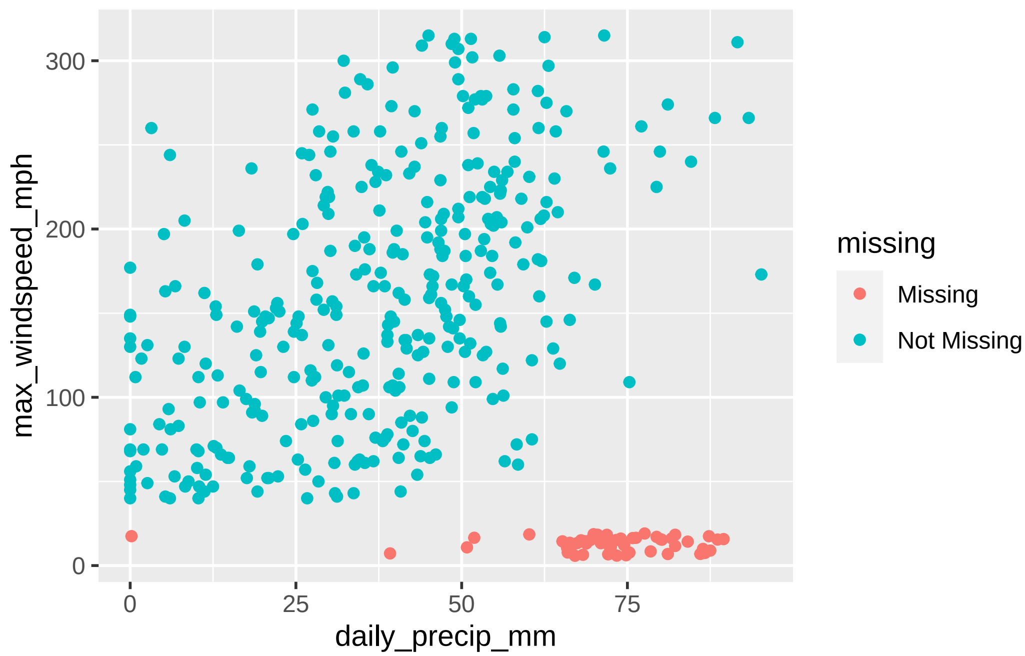
```
vis_miss(df_mar)
```




```
gg_miss_fct(df_mar, fct = tornado_class)
```



```r
ggplot(data = df_mar) +
  geom_miss_point(aes(x = daily_precip_mm, y = max_windspeed_mph))
```




### 12.2.1 MAR: Implications

MAR data means you should be carefully imputing your data. Deleting observations with missing values is not appropriate, as you will likely bias your results.

[TODO: Add more details and worked examples of the implications in this section]

## 12.3 Missing not at random (MNAR)

### 12.3.1 MNAR explanation

If missingness within a variable is related to unobserved data (including values of the missing variable itself), the missingness is missing **not** at random (MNAR).

Let's again envision that we are Bill Paxton, driving out to a tornado to release our devices that record wind speed. In this scenario, the tornado wind speeds are so high that upon approaching the tornado our truck is tipped over, thwarting our efforts to release the devices.

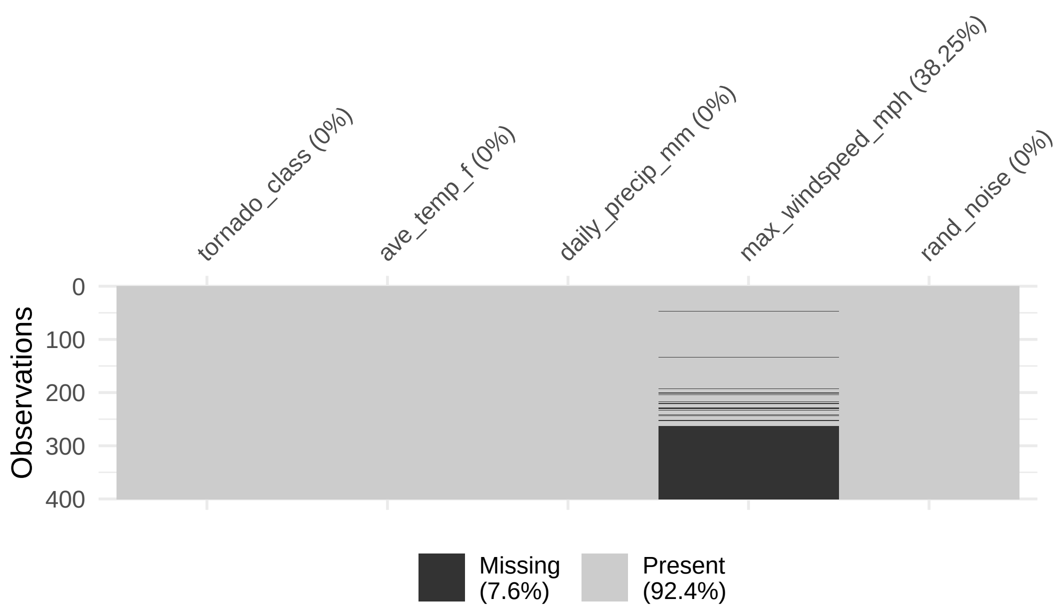
Therefore, we are missing wind speed data for the tornado *because the wind speeds were so high.*

Because the *missingness in wind speed* depends on the unrecorded high values of *wind speed*, the values are missing **not** at random.

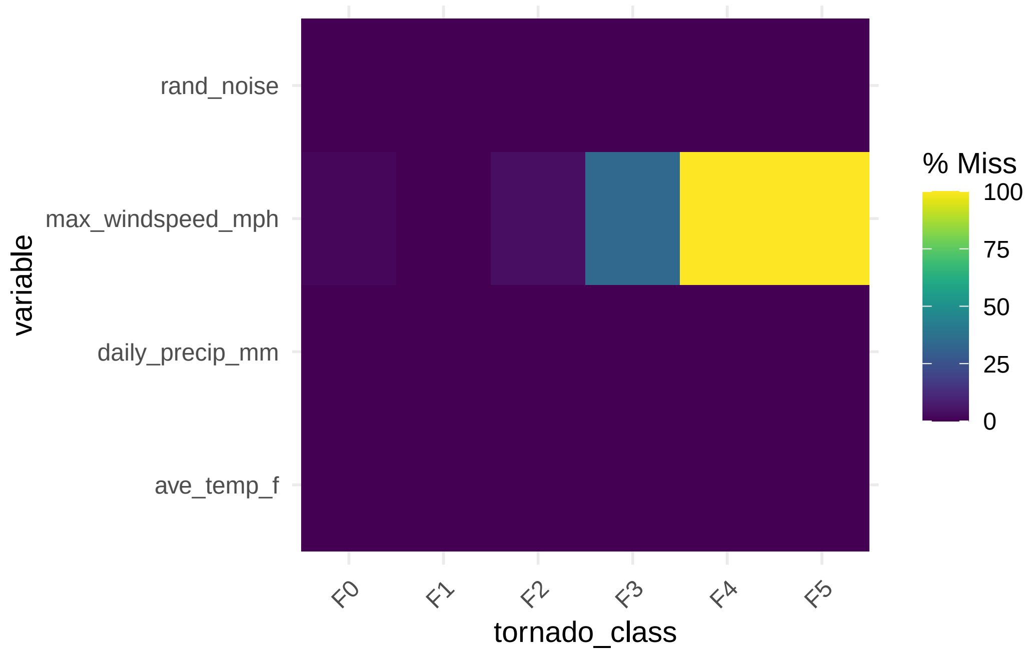
**Critical thinking:** Brainstorm other examples of how values could be MNAR, either from your own work or hypothetically.

```
df_mnar <- twister %>%
  mutate(
    # add some auxiliary random noise to add a sprinkle of missingness
    rand_noise = runif(n = dplyr::n()),
    max_windspeed_mph = case_when(
    max_windspeed_mph >= 200 ~ NA_real_,
    rand_noise > 0.99 ~ NA_real_,
    TRUE ~ max_windspeed_mph
  ))
```

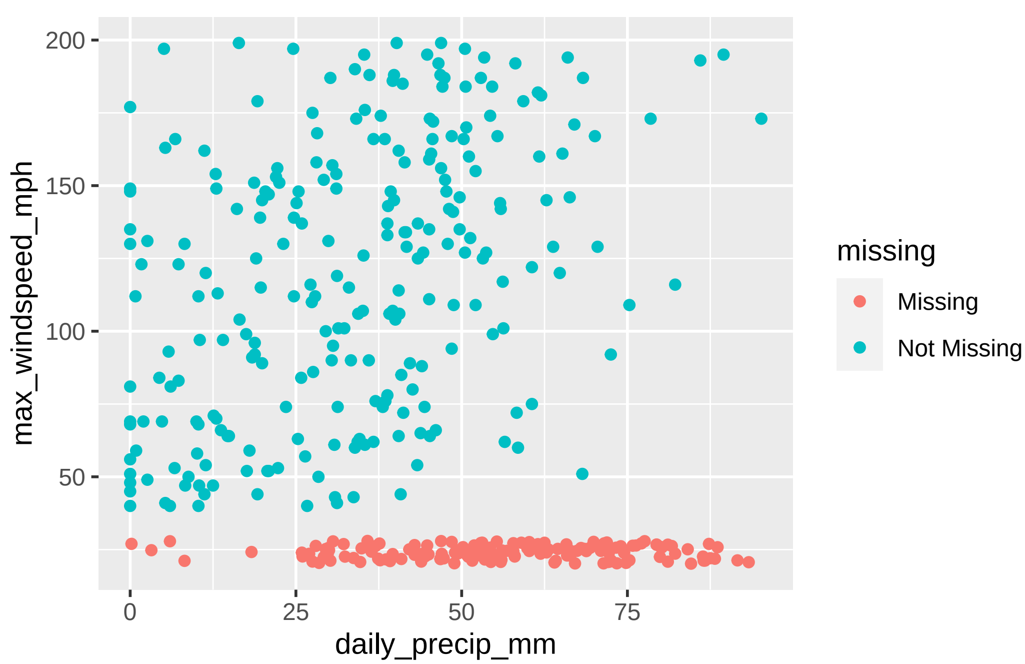
```
vis_miss(df_mnar)
```




```
gg_miss_fct(df_mnar, fct = tornado_class)
```




```
ggplot(data = df_mnar) +
  geom_miss_point(aes(x = daily_precip_mm, y = max_windspeed_mph))
```



[TODO: unpack the difficulty in recognising this type of missingness]

It is hard to understand and identify this missingness precisely, as we can see in the example above, we've set values to be missing once windspeed is over 200Mph, we no longer have those values being recorded! Instead, it appears as though something happens to missing data once daily precipitation goes over 25mm.

#### 12.3.1.1 MNAR: Implications

It is important to recognise MNAR as it introduces bias into the estimation of associations and parameters of interest.

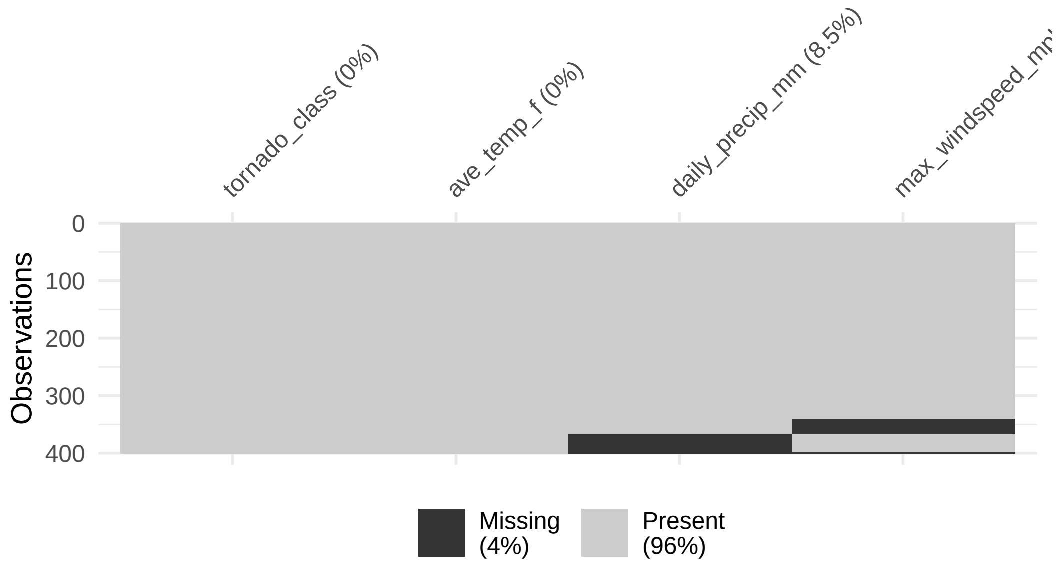
## 12.4 Some more examples of MCAR, MAR, and MNAR

[TODO: combine these examples with the above examples for twister data]

### 12.4.1 Example: MCAR

Now we are going to cover some visualisations to show what certain missingness structures might look like.

```r
vis_miss(df_mcar, cluster = TRUE)
```

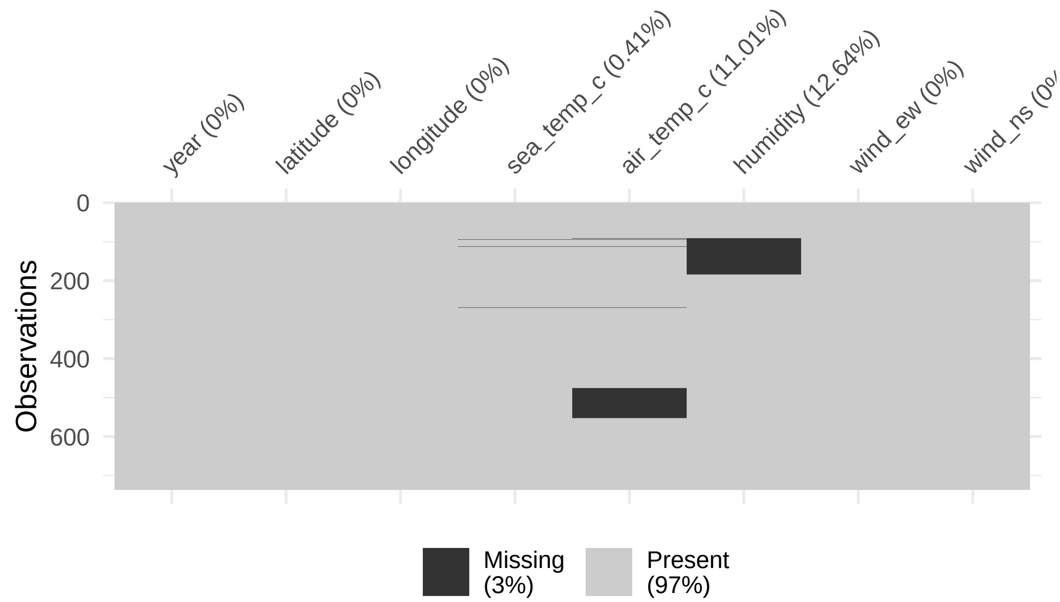



Looking at our data mt_cars, we have applied some clustering to the missingness - and we see that there is still a lot of noise in the missingness. We can also try arranging by a few different variables, but the important thing to take away here is that "random" or "noisy" looking pattern generally suggests there isn't much variation going on in our data. We could say that it is MCAR.

### 12.4.2 Example: MAR

We can do something similar for another dataset, `oceanbuoys`.

```
oceanbuoys %>%
  arrange(year) %>%
  vis_miss()
```



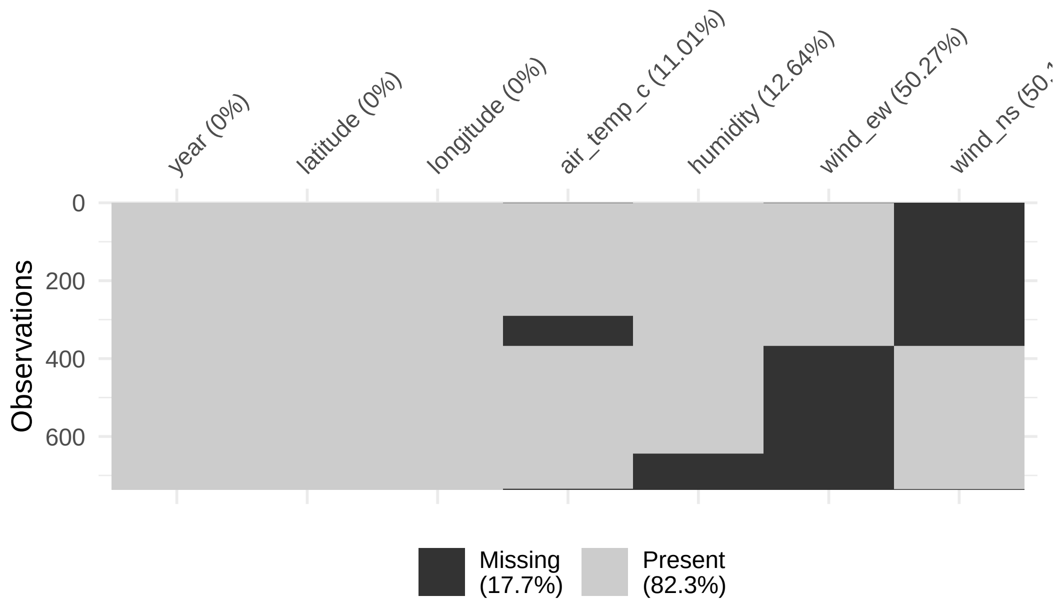

Arranging by variable `year` we see that there is some definite clustering of missingness - this is a common symptom of data MAR.

### 12.4.3 Example: MNAR

```
ocean <- oceanbuoys %>%
  mutate(wind_ew = if_else(sea_temp_c < 26.55,
                           true = NA_real_,
                           false = wind_ew),
         wind_ns = if_else(sea_temp_c > 26.55,
                           true = NA_real_,
                           false = wind_ns)) %>%
  select(-sea_temp_c)
```

Finally, here is some data MNAR.

```
vis_miss(ocean, cluster = TRUE)
```




Here, we have our ocean data, but I have made wind variables be missing according to a variable I have removed from the dataset - something now unobserved. In this case, we can see some very clear structure, but this is not always the case.

It is important to remember it can be very difficult to ascertain whether missingness MCAR, MAR or MNAR. These visualisations are one way to explore missingness, but they are not definitive - we will cover some more useful methods later on in the course.

# Part VI

# Single Imputation of Missing Data

# 13 Single Imputation of missing data

In this section, we are going to focus on two areas:

1. **Using imputations to understand data structure**
2. **Visualising and exploring imputed values**.

The goal is to develop skills in imputing data and tracking missing values, and visualising imputed values against data.

Some of these techniques might look familiar. This is one of the benefits to using `naniar`; the methods applied for exploring missing values are similar to exploring imputations.

## 13.1 Performing and tracking imputation

```r
library(naniar)
library(tidyverse)
```

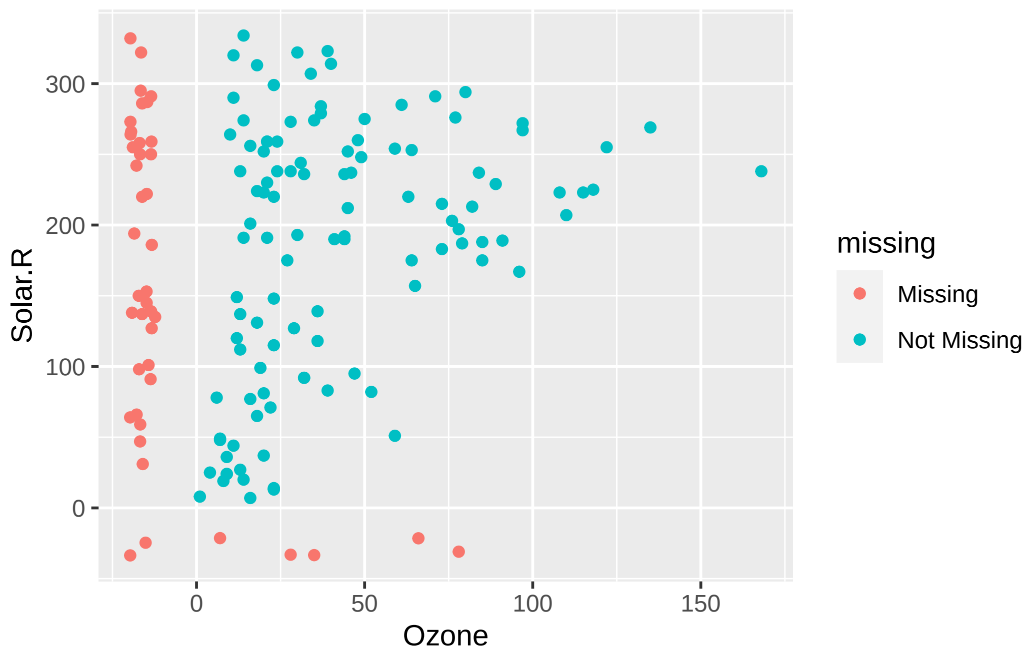
```
-- Attaching packages --------------------------------------- tidyverse 1.3.1 --

v ggplot2 3.3.6     v purrr   0.3.4
v tibble  3.1.7     v dplyr   1.0.9
v tidyr   1.2.0     v stringr 1.4.0
v readr   2.1.2     v forcats 0.5.1

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

One of the goals in exploring missing data is to understand any underlying biases and make the data suitable for analysis. Once we understand our data and the relationships amongst the variables and the missingness, it is a good idea to perform imputation, so that you can conduct analysis with a full dataset.

## 13.2 Using imputations to understand data structure

Previous chapters used **`geom_miss_point()`** to explore missing values. This "shifted" the missing values below the range of the data so we could see them.

```
ggplot(airquality,
       aes(x = Ozone,
           y = Solar.R)) +
  geom_miss_point()
```




This **shifting** was actually "imputing" the data! Remember, "Impute" means to fill in a missing value. We are going to recreate these visualisations using **`impute_below()`** from **`naniar`**. This imputes values below the range of the data. For example, for this vector of numbers 5:10 with one missing value:

```
vec <- c(5,6,7,NA,9,10)
impute_below(vec)
```

```
[1]  5.00000  6.00000  7.00000  4.40271  9.00000 10.00000
```

it imputes the value 4.4 into the missing value, since this is lower than the lowest value of the data at hand, namely 5.000.

### 13.2.1 impute_below()

We can use `impute_below()` in combination with `mutate()` to impute specific values.

For example:

```
airquality %>%
  mutate(Ozone = impute_below(Ozone))
```

```
       Ozone Solar.R Wind Temp Month Day
1   41.00000     190  7.4   67     5   1
2   36.00000     118  8.0   72     5   2
3   12.00000     149 12.6   74     5   3
4   18.00000     313 11.5   62     5   4
5  -19.72321      NA 14.3   56     5   5
6   28.00000      NA 14.9   66     5   6
7   23.00000     299  8.6   65     5   7
8   19.00000      99 13.8   59     5   8
9    8.00000      19 20.1   61     5   9
10 -18.51277     194  8.6   69     5  10
11   7.00000      NA  6.9   74     5  11
12  16.00000     256  9.7   69     5  12
13  11.00000     290  9.2   66     5  13
14  14.00000     274 10.9   68     5  14
15  18.00000      65 13.2   58     5  15
16  14.00000     334 11.5   64     5  16
17  34.00000     307 12.0   66     5  17
18   6.00000      78 18.4   57     5  18
19  30.00000     322 11.5   68     5  19
20  11.00000      44  9.7   62     5  20
21   1.00000       8  9.7   59     5  21
22  11.00000     320 16.6   73     5  22
23   4.00000      25  9.7   61     5  23
24  32.00000      92 12.0   61     5  24
25 -17.81863      66 16.6   57     5  25
26 -19.43853     266 14.9   58     5  26
27 -15.14310      NA  8.0   57     5  27
28  23.00000      13 12.0   67     5  28
29  45.00000     252 14.9   81     5  29
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 30 | 115.00000 | 223 | 5.7 | 79 | 5 | 30 |
| 31 | 37.00000 | 279 | 7.4 | 76 | 5 | 31 |
| 32 | -16.17315 | 286 | 8.6 | 78 | 6 | 1 |
| 33 | -14.65883 | 287 | 9.7 | 74 | 6 | 2 |
| 34 | -17.85609 | 242 | 16.1 | 67 | 6 | 3 |
| 35 | -13.29299 | 186 | 9.2 | 84 | 6 | 4 |
| 36 | -16.16323 | 220 | 8.6 | 85 | 6 | 5 |
| 37 | -19.60935 | 264 | 14.3 | 79 | 6 | 6 |
| 38 | 29.00000 | 127 | 9.7 | 82 | 6 | 7 |
| 39 | -19.65780 | 273 | 6.9 | 87 | 6 | 8 |
| 40 | 71.00000 | 291 | 13.8 | 90 | 6 | 9 |
| 41 | 39.00000 | 323 | 11.5 | 87 | 6 | 10 |
| 42 | -13.40961 | 259 | 10.9 | 93 | 6 | 11 |
| 43 | -13.53728 | 250 | 9.2 | 92 | 6 | 12 |
| 44 | 23.00000 | 148 | 8.0 | 82 | 6 | 13 |
| 45 | -19.65993 | 332 | 13.8 | 80 | 6 | 14 |
| 46 | -16.48342 | 322 | 11.5 | 79 | 6 | 15 |
| 47 | 21.00000 | 191 | 14.9 | 77 | 6 | 16 |
| 48 | 37.00000 | 284 | 20.7 | 72 | 6 | 17 |
| 49 | 20.00000 | 37 | 9.2 | 65 | 6 | 18 |
| 50 | 12.00000 | 120 | 11.5 | 73 | 6 | 19 |
| 51 | 13.00000 | 137 | 10.3 | 76 | 6 | 20 |
| 52 | -17.17718 | 150 | 6.3 | 77 | 6 | 21 |
| 53 | -16.74073 | 59 | 1.7 | 76 | 6 | 22 |
| 54 | -13.65786 | 91 | 4.6 | 76 | 6 | 23 |
| 55 | -16.78786 | 250 | 6.3 | 76 | 6 | 24 |
| 56 | -12.30098 | 135 | 8.0 | 75 | 6 | 25 |
| 57 | -13.33171 | 127 | 8.0 | 78 | 6 | 26 |
| 58 | -16.77414 | 47 | 10.3 | 73 | 6 | 27 |
| 59 | -17.08225 | 98 | 11.5 | 80 | 6 | 28 |
| 60 | -15.98818 | 31 | 14.9 | 77 | 6 | 29 |
| 61 | -19.17558 | 138 | 8.0 | 83 | 6 | 30 |
| 62 | 135.00000 | 269 | 4.1 | 84 | 7 | 1 |
| 63 | 49.00000 | 248 | 9.2 | 85 | 7 | 2 |
| 64 | 32.00000 | 236 | 9.2 | 81 | 7 | 3 |
| 65 | -14.27138 | 101 | 10.9 | 84 | 7 | 4 |
| 66 | 64.00000 | 175 | 4.6 | 83 | 7 | 5 |
| 67 | 40.00000 | 314 | 10.9 | 83 | 7 | 6 |
| 68 | 77.00000 | 276 | 5.1 | 88 | 7 | 7 |
| 69 | 97.00000 | 267 | 6.3 | 92 | 7 | 8 |
| 70 | 97.00000 | 272 | 5.7 | 92 | 7 | 9 |
| 71 | 85.00000 | 175 | 7.4 | 89 | 7 | 10 |
| 72 | -13.51764 | 139 | 8.6 | 82 | 7 | 11 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 73 | 10.00000 | 264 | 14.3 | 73 | 7 | 12 |
| 74 | 27.00000 | 175 | 14.9 | 81 | 7 | 13 |
| 75 | -13.48998 | 291 | 14.9 | 91 | 7 | 14 |
| 76 | 7.00000 | 48 | 14.3 | 80 | 7 | 15 |
| 77 | 48.00000 | 260 | 6.9 | 81 | 7 | 16 |
| 78 | 35.00000 | 274 | 10.3 | 82 | 7 | 17 |
| 79 | 61.00000 | 285 | 6.3 | 84 | 7 | 18 |
| 80 | 79.00000 | 187 | 5.1 | 87 | 7 | 19 |
| 81 | 63.00000 | 220 | 11.5 | 85 | 7 | 20 |
| 82 | 16.00000 | 7 | 6.9 | 74 | 7 | 21 |
| 83 | -16.92150 | 258 | 9.7 | 81 | 7 | 22 |
| 84 | -16.60335 | 295 | 11.5 | 82 | 7 | 23 |
| 85 | 80.00000 | 294 | 8.6 | 86 | 7 | 24 |
| 86 | 108.00000 | 223 | 8.0 | 85 | 7 | 25 |
| 87 | 20.00000 | 81 | 8.6 | 82 | 7 | 26 |
| 88 | 52.00000 | 82 | 12.0 | 86 | 7 | 27 |
| 89 | 82.00000 | 213 | 7.4 | 88 | 7 | 28 |
| 90 | 50.00000 | 275 | 7.4 | 86 | 7 | 29 |
| 91 | 64.00000 | 253 | 7.4 | 83 | 7 | 30 |
| 92 | 59.00000 | 254 | 9.2 | 81 | 7 | 31 |
| 93 | 39.00000 | 83 | 6.9 | 81 | 8 | 1 |
| 94 | 9.00000 | 24 | 13.8 | 81 | 8 | 2 |
| 95 | 16.00000 | 77 | 7.4 | 82 | 8 | 3 |
| 96 | 78.00000 | NA | 6.9 | 86 | 8 | 4 |
| 97 | 35.00000 | NA | 7.4 | 85 | 8 | 5 |
| 98 | 66.00000 | NA | 4.6 | 87 | 8 | 6 |
| 99 | 122.00000 | 255 | 4.0 | 89 | 8 | 7 |
| 100 | 89.00000 | 229 | 10.3 | 90 | 8 | 8 |
| 101 | 110.00000 | 207 | 8.0 | 90 | 8 | 9 |
| 102 | -14.78907 | 222 | 8.6 | 92 | 8 | 10 |
| 103 | -16.19151 | 137 | 11.5 | 86 | 8 | 11 |
| 104 | 44.00000 | 192 | 11.5 | 86 | 8 | 12 |
| 105 | 28.00000 | 273 | 11.5 | 82 | 8 | 13 |
| 106 | 65.00000 | 157 | 9.7 | 80 | 8 | 14 |
| 107 | -19.73591 | 64 | 11.5 | 79 | 8 | 15 |
| 108 | 22.00000 | 71 | 10.3 | 77 | 8 | 16 |
| 109 | 59.00000 | 51 | 6.3 | 79 | 8 | 17 |
| 110 | 23.00000 | 115 | 7.4 | 76 | 8 | 18 |
| 111 | 31.00000 | 244 | 10.9 | 78 | 8 | 19 |
| 112 | 44.00000 | 190 | 10.3 | 78 | 8 | 20 |
| 113 | 21.00000 | 259 | 15.5 | 77 | 8 | 21 |
| 114 | 9.00000 | 36 | 14.3 | 72 | 8 | 22 |
| 115 | -18.92235 | 255 | 12.6 | 75 | 8 | 23 |

```
116  45.00000     212  9.7   79      8  24
117 168.00000     238  3.4   81      8  25
118  73.00000     215  8.0   86      8  26
119 -14.86296     153  5.7   88      8  27
120  76.00000     203  9.7   97      8  28
121 118.00000     225  2.3   94      8  29
122  84.00000     237  6.3   96      8  30
123  85.00000     188  6.3   94      8  31
124  96.00000     167  6.9   91      9   1
125  78.00000     197  5.1   92      9   2
126  73.00000     183  2.8   93      9   3
127  91.00000     189  4.6   93      9   4
128  47.00000      95  7.4   87      9   5
129  32.00000      92 15.5   84      9   6
130  20.00000     252 10.9   80      9   7
131  23.00000     220 10.3   78      9   8
132  21.00000     230 10.9   75      9   9
133  24.00000     259  9.7   73      9  10
134  44.00000     236 14.9   81      9  11
135  21.00000     259 15.5   76      9  12
136  28.00000     238  6.3   77      9  13
137   9.00000      24 10.9   71      9  14
138  13.00000     112 11.5   71      9  15
139  46.00000     237  6.9   78      9  16
140  18.00000     224 13.8   67      9  17
141  13.00000      27 10.3   76      9  18
142  24.00000     238 10.3   68      9  19
143  16.00000     201  8.0   82      9  20
144  13.00000     238 12.6   64      9  21
145  23.00000      14  9.2   71      9  22
146  36.00000     139 10.3   81      9  23
147   7.00000      49 10.3   69      9  24
148  14.00000      20 16.6   63      9  25
149  30.00000     193  6.9   70      9  26
150 -14.83089     145 13.2   77      9  27
151  14.00000     191 14.3   75      9  28
152  18.00000     131  8.0   76      9  29
153  20.00000     223 11.5   68      9  30
```

However, sometimes you want to do this across many variables. Using the same approach for all variables in the dataset could be at best repetitive, and at worst lead to unintended mistakes. We can work around this by using `across`.

If we want to impute all variables, we can use `across` like so:

```
airquality %>%
  mutate(across(everything(),impute_below))
```

```
        Ozone   Solar.R Wind Temp Month Day
1    41.00000 190.00000  7.4   67     5   1
2    36.00000 118.00000  8.0   72     5   2
3    12.00000 149.00000 12.6   74     5   3
4    18.00000 313.00000 11.5   62     5   4
5   -19.72321 -33.57778 14.3   56     5   5
6    28.00000 -33.07810 14.9   66     5   6
7    23.00000 299.00000  8.6   65     5   7
8    19.00000  99.00000 13.8   59     5   8
9     8.00000  19.00000 20.1   61     5   9
10  -18.51277 194.00000  8.6   69     5  10
11    7.00000 -21.37719  6.9   74     5  11
12   16.00000 256.00000  9.7   69     5  12
13   11.00000 290.00000  9.2   66     5  13
14   14.00000 274.00000 10.9   68     5  14
15   18.00000  65.00000 13.2   58     5  15
16   14.00000 334.00000 11.5   64     5  16
17   34.00000 307.00000 12.0   66     5  17
18    6.00000  78.00000 18.4   57     5  18
19   30.00000 322.00000 11.5   68     5  19
20   11.00000  44.00000  9.7   62     5  20
21    1.00000   8.00000  9.7   59     5  21
22   11.00000 320.00000 16.6   73     5  22
23    4.00000  25.00000  9.7   61     5  23
24   32.00000  92.00000 12.0   61     5  24
25  -17.81863  66.00000 16.6   57     5  25
26  -19.43853 266.00000 14.9   58     5  26
27  -15.14310 -24.60954  8.0   57     5  27
28   23.00000  13.00000 12.0   67     5  28
29   45.00000 252.00000 14.9   81     5  29
30  115.00000 223.00000  5.7   79     5  30
31   37.00000 279.00000  7.4   76     5  31
32  -16.17315 286.00000  8.6   78     6   1
33  -14.65883 287.00000  9.7   74     6   2
34  -17.85609 242.00000 16.1   67     6   3
35  -13.29299 186.00000  9.2   84     6   4
36  -16.16323 220.00000  8.6   85     6   5
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 37 | -19.60935 | 264.00000 | 14.3 | 79 | 6 | 6 |
| 38 | 29.00000 | 127.00000 | 9.7 | 82 | 6 | 7 |
| 39 | -19.65780 | 273.00000 | 6.9 | 87 | 6 | 8 |
| 40 | 71.00000 | 291.00000 | 13.8 | 90 | 6 | 9 |
| 41 | 39.00000 | 323.00000 | 11.5 | 87 | 6 | 10 |
| 42 | -13.40961 | 259.00000 | 10.9 | 93 | 6 | 11 |
| 43 | -13.53728 | 250.00000 | 9.2 | 92 | 6 | 12 |
| 44 | 23.00000 | 148.00000 | 8.0 | 82 | 6 | 13 |
| 45 | -19.65993 | 332.00000 | 13.8 | 80 | 6 | 14 |
| 46 | -16.48342 | 322.00000 | 11.5 | 79 | 6 | 15 |
| 47 | 21.00000 | 191.00000 | 14.9 | 77 | 6 | 16 |
| 48 | 37.00000 | 284.00000 | 20.7 | 72 | 6 | 17 |
| 49 | 20.00000 | 37.00000 | 9.2 | 65 | 6 | 18 |
| 50 | 12.00000 | 120.00000 | 11.5 | 73 | 6 | 19 |
| 51 | 13.00000 | 137.00000 | 10.3 | 76 | 6 | 20 |
| 52 | -17.17718 | 150.00000 | 6.3 | 77 | 6 | 21 |
| 53 | -16.74073 | 59.00000 | 1.7 | 76 | 6 | 22 |
| 54 | -13.65786 | 91.00000 | 4.6 | 76 | 6 | 23 |
| 55 | -16.78786 | 250.00000 | 6.3 | 76 | 6 | 24 |
| 56 | -12.30098 | 135.00000 | 8.0 | 75 | 6 | 25 |
| 57 | -13.33171 | 127.00000 | 8.0 | 78 | 6 | 26 |
| 58 | -16.77414 | 47.00000 | 10.3 | 73 | 6 | 27 |
| 59 | -17.08225 | 98.00000 | 11.5 | 80 | 6 | 28 |
| 60 | -15.98818 | 31.00000 | 14.9 | 77 | 6 | 29 |
| 61 | -19.17558 | 138.00000 | 8.0 | 83 | 6 | 30 |
| 62 | 135.00000 | 269.00000 | 4.1 | 84 | 7 | 1 |
| 63 | 49.00000 | 248.00000 | 9.2 | 85 | 7 | 2 |
| 64 | 32.00000 | 236.00000 | 9.2 | 81 | 7 | 3 |
| 65 | -14.27138 | 101.00000 | 10.9 | 84 | 7 | 4 |
| 66 | 64.00000 | 175.00000 | 4.6 | 83 | 7 | 5 |
| 67 | 40.00000 | 314.00000 | 10.9 | 83 | 7 | 6 |
| 68 | 77.00000 | 276.00000 | 5.1 | 88 | 7 | 7 |
| 69 | 97.00000 | 267.00000 | 6.3 | 92 | 7 | 8 |
| 70 | 97.00000 | 272.00000 | 5.7 | 92 | 7 | 9 |
| 71 | 85.00000 | 175.00000 | 7.4 | 89 | 7 | 10 |
| 72 | -13.51764 | 139.00000 | 8.6 | 82 | 7 | 11 |
| 73 | 10.00000 | 264.00000 | 14.3 | 73 | 7 | 12 |
| 74 | 27.00000 | 175.00000 | 14.9 | 81 | 7 | 13 |
| 75 | -13.48998 | 291.00000 | 14.9 | 91 | 7 | 14 |
| 76 | 7.00000 | 48.00000 | 14.3 | 80 | 7 | 15 |
| 77 | 48.00000 | 260.00000 | 6.9 | 81 | 7 | 16 |
| 78 | 35.00000 | 274.00000 | 10.3 | 82 | 7 | 17 |
| 79 | 61.00000 | 285.00000 | 6.3 | 84 | 7 | 18 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 80 | 79.00000 | 187.00000 | 5.1 | 87 | 7 | 19 |
| 81 | 63.00000 | 220.00000 | 11.5 | 85 | 7 | 20 |
| 82 | 16.00000 | 7.00000 | 6.9 | 74 | 7 | 21 |
| 83 | -16.92150 | 258.00000 | 9.7 | 81 | 7 | 22 |
| 84 | -16.60335 | 295.00000 | 11.5 | 82 | 7 | 23 |
| 85 | 80.00000 | 294.00000 | 8.6 | 86 | 7 | 24 |
| 86 | 108.00000 | 223.00000 | 8.0 | 85 | 7 | 25 |
| 87 | 20.00000 | 81.00000 | 8.6 | 82 | 7 | 26 |
| 88 | 52.00000 | 82.00000 | 12.0 | 86 | 7 | 27 |
| 89 | 82.00000 | 213.00000 | 7.4 | 88 | 7 | 28 |
| 90 | 50.00000 | 275.00000 | 7.4 | 86 | 7 | 29 |
| 91 | 64.00000 | 253.00000 | 7.4 | 83 | 7 | 30 |
| 92 | 59.00000 | 254.00000 | 9.2 | 81 | 7 | 31 |
| 93 | 39.00000 | 83.00000 | 6.9 | 81 | 8 | 1 |
| 94 | 9.00000 | 24.00000 | 13.8 | 81 | 8 | 2 |
| 95 | 16.00000 | 77.00000 | 7.4 | 82 | 8 | 3 |
| 96 | 78.00000 | -30.94374 | 6.9 | 86 | 8 | 4 |
| 97 | 35.00000 | -33.38707 | 7.4 | 85 | 8 | 5 |
| 98 | 66.00000 | -21.48980 | 4.6 | 87 | 8 | 6 |
| 99 | 122.00000 | 255.00000 | 4.0 | 89 | 8 | 7 |
| 100 | 89.00000 | 229.00000 | 10.3 | 90 | 8 | 8 |
| 101 | 110.00000 | 207.00000 | 8.0 | 90 | 8 | 9 |
| 102 | -14.78907 | 222.00000 | 8.6 | 92 | 8 | 10 |
| 103 | -16.19151 | 137.00000 | 11.5 | 86 | 8 | 11 |
| 104 | 44.00000 | 192.00000 | 11.5 | 86 | 8 | 12 |
| 105 | 28.00000 | 273.00000 | 11.5 | 82 | 8 | 13 |
| 106 | 65.00000 | 157.00000 | 9.7 | 80 | 8 | 14 |
| 107 | -19.73591 | 64.00000 | 11.5 | 79 | 8 | 15 |
| 108 | 22.00000 | 71.00000 | 10.3 | 77 | 8 | 16 |
| 109 | 59.00000 | 51.00000 | 6.3 | 79 | 8 | 17 |
| 110 | 23.00000 | 115.00000 | 7.4 | 76 | 8 | 18 |
| 111 | 31.00000 | 244.00000 | 10.9 | 78 | 8 | 19 |
| 112 | 44.00000 | 190.00000 | 10.3 | 78 | 8 | 20 |
| 113 | 21.00000 | 259.00000 | 15.5 | 77 | 8 | 21 |
| 114 | 9.00000 | 36.00000 | 14.3 | 72 | 8 | 22 |
| 115 | -18.92235 | 255.00000 | 12.6 | 75 | 8 | 23 |
| 116 | 45.00000 | 212.00000 | 9.7 | 79 | 8 | 24 |
| 117 | 168.00000 | 238.00000 | 3.4 | 81 | 8 | 25 |
| 118 | 73.00000 | 215.00000 | 8.0 | 86 | 8 | 26 |
| 119 | -14.86296 | 153.00000 | 5.7 | 88 | 8 | 27 |
| 120 | 76.00000 | 203.00000 | 9.7 | 97 | 8 | 28 |
| 121 | 118.00000 | 225.00000 | 2.3 | 94 | 8 | 29 |
| 122 | 84.00000 | 237.00000 | 6.3 | 96 | 8 | 30 |

```
123  85.00000 188.00000  6.3   94      8  31
124  96.00000 167.00000  6.9   91      9   1
125  78.00000 197.00000  5.1   92      9   2
126  73.00000 183.00000  2.8   93      9   3
127  91.00000 189.00000  4.6   93      9   4
128  47.00000  95.00000  7.4   87      9   5
129  32.00000  92.00000 15.5   84      9   6
130  20.00000 252.00000 10.9   80      9   7
131  23.00000 220.00000 10.3   78      9   8
132  21.00000 230.00000 10.9   75      9   9
133  24.00000 259.00000  9.7   73      9  10
134  44.00000 236.00000 14.9   81      9  11
135  21.00000 259.00000 15.5   76      9  12
136  28.00000 238.00000  6.3   77      9  13
137   9.00000  24.00000 10.9   71      9  14
138  13.00000 112.00000 11.5   71      9  15
139  46.00000 237.00000  6.9   78      9  16
140  18.00000 224.00000 13.8   67      9  17
141  13.00000  27.00000 10.3   76      9  18
142  24.00000 238.00000 10.3   68      9  19
143  16.00000 201.00000  8.0   82      9  20
144  13.00000 238.00000 12.6   64      9  21
145  23.00000  14.00000  9.2   71      9  22
146  36.00000 139.00000 10.3   81      9  23
147   7.00000  49.00000 10.3   69      9  24
148  14.00000  20.00000 16.6   63      9  25
149  30.00000 193.00000  6.9   70      9  26
150 -14.83089 145.00000 13.2   77      9  27
151  14.00000 191.00000 14.3   75      9  28
152  18.00000 131.00000  8.0   76      9  29
153  20.00000 223.00000 11.5   68      9  30
```

Here we use the `everything()` helper function from dplyr, to select all variables. We can use any type of selection, from `dplyrs` tidy select.

We can impute only those variables that satisfy a condition, like is this column numeric with `is.numeric()` using `where()` like so:

```
airquality %>%
  mutate(across(where(is.numeric),impute_below))
```

```
        Ozone   Solar.R Wind Temp Month Day
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 41.00000 | 190.00000 | 7.4 | 67 | 5 | 1 |
| 2 | 36.00000 | 118.00000 | 8.0 | 72 | 5 | 2 |
| 3 | 12.00000 | 149.00000 | 12.6 | 74 | 5 | 3 |
| 4 | 18.00000 | 313.00000 | 11.5 | 62 | 5 | 4 |
| 5 | -19.72321 | -33.57778 | 14.3 | 56 | 5 | 5 |
| 6 | 28.00000 | -33.07810 | 14.9 | 66 | 5 | 6 |
| 7 | 23.00000 | 299.00000 | 8.6 | 65 | 5 | 7 |
| 8 | 19.00000 | 99.00000 | 13.8 | 59 | 5 | 8 |
| 9 | 8.00000 | 19.00000 | 20.1 | 61 | 5 | 9 |
| 10 | -18.51277 | 194.00000 | 8.6 | 69 | 5 | 10 |
| 11 | 7.00000 | -21.37719 | 6.9 | 74 | 5 | 11 |
| 12 | 16.00000 | 256.00000 | 9.7 | 69 | 5 | 12 |
| 13 | 11.00000 | 290.00000 | 9.2 | 66 | 5 | 13 |
| 14 | 14.00000 | 274.00000 | 10.9 | 68 | 5 | 14 |
| 15 | 18.00000 | 65.00000 | 13.2 | 58 | 5 | 15 |
| 16 | 14.00000 | 334.00000 | 11.5 | 64 | 5 | 16 |
| 17 | 34.00000 | 307.00000 | 12.0 | 66 | 5 | 17 |
| 18 | 6.00000 | 78.00000 | 18.4 | 57 | 5 | 18 |
| 19 | 30.00000 | 322.00000 | 11.5 | 68 | 5 | 19 |
| 20 | 11.00000 | 44.00000 | 9.7 | 62 | 5 | 20 |
| 21 | 1.00000 | 8.00000 | 9.7 | 59 | 5 | 21 |
| 22 | 11.00000 | 320.00000 | 16.6 | 73 | 5 | 22 |
| 23 | 4.00000 | 25.00000 | 9.7 | 61 | 5 | 23 |
| 24 | 32.00000 | 92.00000 | 12.0 | 61 | 5 | 24 |
| 25 | -17.81863 | 66.00000 | 16.6 | 57 | 5 | 25 |
| 26 | -19.43853 | 266.00000 | 14.9 | 58 | 5 | 26 |
| 27 | -15.14310 | -24.60954 | 8.0 | 57 | 5 | 27 |
| 28 | 23.00000 | 13.00000 | 12.0 | 67 | 5 | 28 |
| 29 | 45.00000 | 252.00000 | 14.9 | 81 | 5 | 29 |
| 30 | 115.00000 | 223.00000 | 5.7 | 79 | 5 | 30 |
| 31 | 37.00000 | 279.00000 | 7.4 | 76 | 5 | 31 |
| 32 | -16.17315 | 286.00000 | 8.6 | 78 | 6 | 1 |
| 33 | -14.65883 | 287.00000 | 9.7 | 74 | 6 | 2 |
| 34 | -17.85609 | 242.00000 | 16.1 | 67 | 6 | 3 |
| 35 | -13.29299 | 186.00000 | 9.2 | 84 | 6 | 4 |
| 36 | -16.16323 | 220.00000 | 8.6 | 85 | 6 | 5 |
| 37 | -19.60935 | 264.00000 | 14.3 | 79 | 6 | 6 |
| 38 | 29.00000 | 127.00000 | 9.7 | 82 | 6 | 7 |
| 39 | -19.65780 | 273.00000 | 6.9 | 87 | 6 | 8 |
| 40 | 71.00000 | 291.00000 | 13.8 | 90 | 6 | 9 |
| 41 | 39.00000 | 323.00000 | 11.5 | 87 | 6 | 10 |
| 42 | -13.40961 | 259.00000 | 10.9 | 93 | 6 | 11 |
| 43 | -13.53728 | 250.00000 | 9.2 | 92 | 6 | 12 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 44 | 23.00000 | 148.00000 | 8.0 | 82 | 6 | 13 |
| 45 | -19.65993 | 332.00000 | 13.8 | 80 | 6 | 14 |
| 46 | -16.48342 | 322.00000 | 11.5 | 79 | 6 | 15 |
| 47 | 21.00000 | 191.00000 | 14.9 | 77 | 6 | 16 |
| 48 | 37.00000 | 284.00000 | 20.7 | 72 | 6 | 17 |
| 49 | 20.00000 | 37.00000 | 9.2 | 65 | 6 | 18 |
| 50 | 12.00000 | 120.00000 | 11.5 | 73 | 6 | 19 |
| 51 | 13.00000 | 137.00000 | 10.3 | 76 | 6 | 20 |
| 52 | -17.17718 | 150.00000 | 6.3 | 77 | 6 | 21 |
| 53 | -16.74073 | 59.00000 | 1.7 | 76 | 6 | 22 |
| 54 | -13.65786 | 91.00000 | 4.6 | 76 | 6 | 23 |
| 55 | -16.78786 | 250.00000 | 6.3 | 76 | 6 | 24 |
| 56 | -12.30098 | 135.00000 | 8.0 | 75 | 6 | 25 |
| 57 | -13.33171 | 127.00000 | 8.0 | 78 | 6 | 26 |
| 58 | -16.77414 | 47.00000 | 10.3 | 73 | 6 | 27 |
| 59 | -17.08225 | 98.00000 | 11.5 | 80 | 6 | 28 |
| 60 | -15.98818 | 31.00000 | 14.9 | 77 | 6 | 29 |
| 61 | -19.17558 | 138.00000 | 8.0 | 83 | 6 | 30 |
| 62 | 135.00000 | 269.00000 | 4.1 | 84 | 7 | 1 |
| 63 | 49.00000 | 248.00000 | 9.2 | 85 | 7 | 2 |
| 64 | 32.00000 | 236.00000 | 9.2 | 81 | 7 | 3 |
| 65 | -14.27138 | 101.00000 | 10.9 | 84 | 7 | 4 |
| 66 | 64.00000 | 175.00000 | 4.6 | 83 | 7 | 5 |
| 67 | 40.00000 | 314.00000 | 10.9 | 83 | 7 | 6 |
| 68 | 77.00000 | 276.00000 | 5.1 | 88 | 7 | 7 |
| 69 | 97.00000 | 267.00000 | 6.3 | 92 | 7 | 8 |
| 70 | 97.00000 | 272.00000 | 5.7 | 92 | 7 | 9 |
| 71 | 85.00000 | 175.00000 | 7.4 | 89 | 7 | 10 |
| 72 | -13.51764 | 139.00000 | 8.6 | 82 | 7 | 11 |
| 73 | 10.00000 | 264.00000 | 14.3 | 73 | 7 | 12 |
| 74 | 27.00000 | 175.00000 | 14.9 | 81 | 7 | 13 |
| 75 | -13.48998 | 291.00000 | 14.9 | 91 | 7 | 14 |
| 76 | 7.00000 | 48.00000 | 14.3 | 80 | 7 | 15 |
| 77 | 48.00000 | 260.00000 | 6.9 | 81 | 7 | 16 |
| 78 | 35.00000 | 274.00000 | 10.3 | 82 | 7 | 17 |
| 79 | 61.00000 | 285.00000 | 6.3 | 84 | 7 | 18 |
| 80 | 79.00000 | 187.00000 | 5.1 | 87 | 7 | 19 |
| 81 | 63.00000 | 220.00000 | 11.5 | 85 | 7 | 20 |
| 82 | 16.00000 | 7.00000 | 6.9 | 74 | 7 | 21 |
| 83 | -16.92150 | 258.00000 | 9.7 | 81 | 7 | 22 |
| 84 | -16.60335 | 295.00000 | 11.5 | 82 | 7 | 23 |
| 85 | 80.00000 | 294.00000 | 8.6 | 86 | 7 | 24 |
| 86 | 108.00000 | 223.00000 | 8.0 | 85 | 7 | 25 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 87 | 20.00000 | 81.00000 | 8.6 | 82 | 7 | 26 |
| 88 | 52.00000 | 82.00000 | 12.0 | 86 | 7 | 27 |
| 89 | 82.00000 | 213.00000 | 7.4 | 88 | 7 | 28 |
| 90 | 50.00000 | 275.00000 | 7.4 | 86 | 7 | 29 |
| 91 | 64.00000 | 253.00000 | 7.4 | 83 | 7 | 30 |
| 92 | 59.00000 | 254.00000 | 9.2 | 81 | 7 | 31 |
| 93 | 39.00000 | 83.00000 | 6.9 | 81 | 8 | 1 |
| 94 | 9.00000 | 24.00000 | 13.8 | 81 | 8 | 2 |
| 95 | 16.00000 | 77.00000 | 7.4 | 82 | 8 | 3 |
| 96 | 78.00000 | -30.94374 | 6.9 | 86 | 8 | 4 |
| 97 | 35.00000 | -33.38707 | 7.4 | 85 | 8 | 5 |
| 98 | 66.00000 | -21.48980 | 4.6 | 87 | 8 | 6 |
| 99 | 122.00000 | 255.00000 | 4.0 | 89 | 8 | 7 |
| 100 | 89.00000 | 229.00000 | 10.3 | 90 | 8 | 8 |
| 101 | 110.00000 | 207.00000 | 8.0 | 90 | 8 | 9 |
| 102 | -14.78907 | 222.00000 | 8.6 | 92 | 8 | 10 |
| 103 | -16.19151 | 137.00000 | 11.5 | 86 | 8 | 11 |
| 104 | 44.00000 | 192.00000 | 11.5 | 86 | 8 | 12 |
| 105 | 28.00000 | 273.00000 | 11.5 | 82 | 8 | 13 |
| 106 | 65.00000 | 157.00000 | 9.7 | 80 | 8 | 14 |
| 107 | -19.73591 | 64.00000 | 11.5 | 79 | 8 | 15 |
| 108 | 22.00000 | 71.00000 | 10.3 | 77 | 8 | 16 |
| 109 | 59.00000 | 51.00000 | 6.3 | 79 | 8 | 17 |
| 110 | 23.00000 | 115.00000 | 7.4 | 76 | 8 | 18 |
| 111 | 31.00000 | 244.00000 | 10.9 | 78 | 8 | 19 |
| 112 | 44.00000 | 190.00000 | 10.3 | 78 | 8 | 20 |
| 113 | 21.00000 | 259.00000 | 15.5 | 77 | 8 | 21 |
| 114 | 9.00000 | 36.00000 | 14.3 | 72 | 8 | 22 |
| 115 | -18.92235 | 255.00000 | 12.6 | 75 | 8 | 23 |
| 116 | 45.00000 | 212.00000 | 9.7 | 79 | 8 | 24 |
| 117 | 168.00000 | 238.00000 | 3.4 | 81 | 8 | 25 |
| 118 | 73.00000 | 215.00000 | 8.0 | 86 | 8 | 26 |
| 119 | -14.86296 | 153.00000 | 5.7 | 88 | 8 | 27 |
| 120 | 76.00000 | 203.00000 | 9.7 | 97 | 8 | 28 |
| 121 | 118.00000 | 225.00000 | 2.3 | 94 | 8 | 29 |
| 122 | 84.00000 | 237.00000 | 6.3 | 96 | 8 | 30 |
| 123 | 85.00000 | 188.00000 | 6.3 | 94 | 8 | 31 |
| 124 | 96.00000 | 167.00000 | 6.9 | 91 | 9 | 1 |
| 125 | 78.00000 | 197.00000 | 5.1 | 92 | 9 | 2 |
| 126 | 73.00000 | 183.00000 | 2.8 | 93 | 9 | 3 |
| 127 | 91.00000 | 189.00000 | 4.6 | 93 | 9 | 4 |
| 128 | 47.00000 | 95.00000 | 7.4 | 87 | 9 | 5 |
| 129 | 32.00000 | 92.00000 | 15.5 | 84 | 9 | 6 |

```
130  20.00000 252.00000 10.9   80      9   7
131  23.00000 220.00000 10.3   78      9   8
132  21.00000 230.00000 10.9   75      9   9
133  24.00000 259.00000  9.7   73      9  10
134  44.00000 236.00000 14.9   81      9  11
135  21.00000 259.00000 15.5   76      9  12
136  28.00000 238.00000  6.3   77      9  13
137   9.00000  24.00000 10.9   71      9  14
138  13.00000 112.00000 11.5   71      9  15
139  46.00000 237.00000  6.9   78      9  16
140  18.00000 224.00000 13.8   67      9  17
141  13.00000  27.00000 10.3   76      9  18
142  24.00000 238.00000 10.3   68      9  19
143  16.00000 201.00000  8.0   82      9  20
144  13.00000 238.00000 12.6   64      9  21
145  23.00000  14.00000  9.2   71      9  22
146  36.00000 139.00000 10.3   81      9  23
147   7.00000  49.00000 10.3   69      9  24
148  14.00000  20.00000 16.6   63      9  25
149  30.00000 193.00000  6.9   70      9  26
150 -14.83089 145.00000 13.2   77      9  27
151  14.00000 191.00000 14.3   75      9  28
152  18.00000 131.00000  8.0   76      9  29
153  20.00000 223.00000 11.5   68      9  30
```

This reads as:

> Use airquality then across variables where they are numeric, impute below

We can choose specific variables like so:

```
airquality %>%
  mutate(across(c(Ozone, Solar.R),impute_below))
```

```
        Ozone   Solar.R Wind Temp Month Day
1    41.00000 190.00000  7.4   67     5   1
2    36.00000 118.00000  8.0   72     5   2
3    12.00000 149.00000 12.6   74     5   3
4    18.00000 313.00000 11.5   62     5   4
5   -19.72321 -33.57778 14.3   56     5   5
6    28.00000 -33.07810 14.9   66     5   6
7    23.00000 299.00000  8.6   65     5   7
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 8 | 19.00000 | 99.00000 | 13.8 | 59 | 5 | 8 |
| 9 | 8.00000 | 19.00000 | 20.1 | 61 | 5 | 9 |
| 10 | -18.51277 | 194.00000 | 8.6 | 69 | 5 | 10 |
| 11 | 7.00000 | -21.37719 | 6.9 | 74 | 5 | 11 |
| 12 | 16.00000 | 256.00000 | 9.7 | 69 | 5 | 12 |
| 13 | 11.00000 | 290.00000 | 9.2 | 66 | 5 | 13 |
| 14 | 14.00000 | 274.00000 | 10.9 | 68 | 5 | 14 |
| 15 | 18.00000 | 65.00000 | 13.2 | 58 | 5 | 15 |
| 16 | 14.00000 | 334.00000 | 11.5 | 64 | 5 | 16 |
| 17 | 34.00000 | 307.00000 | 12.0 | 66 | 5 | 17 |
| 18 | 6.00000 | 78.00000 | 18.4 | 57 | 5 | 18 |
| 19 | 30.00000 | 322.00000 | 11.5 | 68 | 5 | 19 |
| 20 | 11.00000 | 44.00000 | 9.7 | 62 | 5 | 20 |
| 21 | 1.00000 | 8.00000 | 9.7 | 59 | 5 | 21 |
| 22 | 11.00000 | 320.00000 | 16.6 | 73 | 5 | 22 |
| 23 | 4.00000 | 25.00000 | 9.7 | 61 | 5 | 23 |
| 24 | 32.00000 | 92.00000 | 12.0 | 61 | 5 | 24 |
| 25 | -17.81863 | 66.00000 | 16.6 | 57 | 5 | 25 |
| 26 | -19.43853 | 266.00000 | 14.9 | 58 | 5 | 26 |
| 27 | -15.14310 | -24.60954 | 8.0 | 57 | 5 | 27 |
| 28 | 23.00000 | 13.00000 | 12.0 | 67 | 5 | 28 |
| 29 | 45.00000 | 252.00000 | 14.9 | 81 | 5 | 29 |
| 30 | 115.00000 | 223.00000 | 5.7 | 79 | 5 | 30 |
| 31 | 37.00000 | 279.00000 | 7.4 | 76 | 5 | 31 |
| 32 | -16.17315 | 286.00000 | 8.6 | 78 | 6 | 1 |
| 33 | -14.65883 | 287.00000 | 9.7 | 74 | 6 | 2 |
| 34 | -17.85609 | 242.00000 | 16.1 | 67 | 6 | 3 |
| 35 | -13.29299 | 186.00000 | 9.2 | 84 | 6 | 4 |
| 36 | -16.16323 | 220.00000 | 8.6 | 85 | 6 | 5 |
| 37 | -19.60935 | 264.00000 | 14.3 | 79 | 6 | 6 |
| 38 | 29.00000 | 127.00000 | 9.7 | 82 | 6 | 7 |
| 39 | -19.65780 | 273.00000 | 6.9 | 87 | 6 | 8 |
| 40 | 71.00000 | 291.00000 | 13.8 | 90 | 6 | 9 |
| 41 | 39.00000 | 323.00000 | 11.5 | 87 | 6 | 10 |
| 42 | -13.40961 | 259.00000 | 10.9 | 93 | 6 | 11 |
| 43 | -13.53728 | 250.00000 | 9.2 | 92 | 6 | 12 |
| 44 | 23.00000 | 148.00000 | 8.0 | 82 | 6 | 13 |
| 45 | -19.65993 | 332.00000 | 13.8 | 80 | 6 | 14 |
| 46 | -16.48342 | 322.00000 | 11.5 | 79 | 6 | 15 |
| 47 | 21.00000 | 191.00000 | 14.9 | 77 | 6 | 16 |
| 48 | 37.00000 | 284.00000 | 20.7 | 72 | 6 | 17 |
| 49 | 20.00000 | 37.00000 | 9.2 | 65 | 6 | 18 |
| 50 | 12.00000 | 120.00000 | 11.5 | 73 | 6 | 19 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 51 | 13.00000 | 137.00000 | 10.3 | 76 | 6 | 20 |
| 52 | -17.17718 | 150.00000 | 6.3 | 77 | 6 | 21 |
| 53 | -16.74073 | 59.00000 | 1.7 | 76 | 6 | 22 |
| 54 | -13.65786 | 91.00000 | 4.6 | 76 | 6 | 23 |
| 55 | -16.78786 | 250.00000 | 6.3 | 76 | 6 | 24 |
| 56 | -12.30098 | 135.00000 | 8.0 | 75 | 6 | 25 |
| 57 | -13.33171 | 127.00000 | 8.0 | 78 | 6 | 26 |
| 58 | -16.77414 | 47.00000 | 10.3 | 73 | 6 | 27 |
| 59 | -17.08225 | 98.00000 | 11.5 | 80 | 6 | 28 |
| 60 | -15.98818 | 31.00000 | 14.9 | 77 | 6 | 29 |
| 61 | -19.17558 | 138.00000 | 8.0 | 83 | 6 | 30 |
| 62 | 135.00000 | 269.00000 | 4.1 | 84 | 7 | 1 |
| 63 | 49.00000 | 248.00000 | 9.2 | 85 | 7 | 2 |
| 64 | 32.00000 | 236.00000 | 9.2 | 81 | 7 | 3 |
| 65 | -14.27138 | 101.00000 | 10.9 | 84 | 7 | 4 |
| 66 | 64.00000 | 175.00000 | 4.6 | 83 | 7 | 5 |
| 67 | 40.00000 | 314.00000 | 10.9 | 83 | 7 | 6 |
| 68 | 77.00000 | 276.00000 | 5.1 | 88 | 7 | 7 |
| 69 | 97.00000 | 267.00000 | 6.3 | 92 | 7 | 8 |
| 70 | 97.00000 | 272.00000 | 5.7 | 92 | 7 | 9 |
| 71 | 85.00000 | 175.00000 | 7.4 | 89 | 7 | 10 |
| 72 | -13.51764 | 139.00000 | 8.6 | 82 | 7 | 11 |
| 73 | 10.00000 | 264.00000 | 14.3 | 73 | 7 | 12 |
| 74 | 27.00000 | 175.00000 | 14.9 | 81 | 7 | 13 |
| 75 | -13.48998 | 291.00000 | 14.9 | 91 | 7 | 14 |
| 76 | 7.00000 | 48.00000 | 14.3 | 80 | 7 | 15 |
| 77 | 48.00000 | 260.00000 | 6.9 | 81 | 7 | 16 |
| 78 | 35.00000 | 274.00000 | 10.3 | 82 | 7 | 17 |
| 79 | 61.00000 | 285.00000 | 6.3 | 84 | 7 | 18 |
| 80 | 79.00000 | 187.00000 | 5.1 | 87 | 7 | 19 |
| 81 | 63.00000 | 220.00000 | 11.5 | 85 | 7 | 20 |
| 82 | 16.00000 | 7.00000 | 6.9 | 74 | 7 | 21 |
| 83 | -16.92150 | 258.00000 | 9.7 | 81 | 7 | 22 |
| 84 | -16.60335 | 295.00000 | 11.5 | 82 | 7 | 23 |
| 85 | 80.00000 | 294.00000 | 8.6 | 86 | 7 | 24 |
| 86 | 108.00000 | 223.00000 | 8.0 | 85 | 7 | 25 |
| 87 | 20.00000 | 81.00000 | 8.6 | 82 | 7 | 26 |
| 88 | 52.00000 | 82.00000 | 12.0 | 86 | 7 | 27 |
| 89 | 82.00000 | 213.00000 | 7.4 | 88 | 7 | 28 |
| 90 | 50.00000 | 275.00000 | 7.4 | 86 | 7 | 29 |
| 91 | 64.00000 | 253.00000 | 7.4 | 83 | 7 | 30 |
| 92 | 59.00000 | 254.00000 | 9.2 | 81 | 7 | 31 |
| 93 | 39.00000 | 83.00000 | 6.9 | 81 | 8 | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 94 | 9.00000 | 24.00000 | 13.8 | 81 | 8 | 2 |
| 95 | 16.00000 | 77.00000 | 7.4 | 82 | 8 | 3 |
| 96 | 78.00000 | -30.94374 | 6.9 | 86 | 8 | 4 |
| 97 | 35.00000 | -33.38707 | 7.4 | 85 | 8 | 5 |
| 98 | 66.00000 | -21.48980 | 4.6 | 87 | 8 | 6 |
| 99 | 122.00000 | 255.00000 | 4.0 | 89 | 8 | 7 |
| 100 | 89.00000 | 229.00000 | 10.3 | 90 | 8 | 8 |
| 101 | 110.00000 | 207.00000 | 8.0 | 90 | 8 | 9 |
| 102 | -14.78907 | 222.00000 | 8.6 | 92 | 8 | 10 |
| 103 | -16.19151 | 137.00000 | 11.5 | 86 | 8 | 11 |
| 104 | 44.00000 | 192.00000 | 11.5 | 86 | 8 | 12 |
| 105 | 28.00000 | 273.00000 | 11.5 | 82 | 8 | 13 |
| 106 | 65.00000 | 157.00000 | 9.7 | 80 | 8 | 14 |
| 107 | -19.73591 | 64.00000 | 11.5 | 79 | 8 | 15 |
| 108 | 22.00000 | 71.00000 | 10.3 | 77 | 8 | 16 |
| 109 | 59.00000 | 51.00000 | 6.3 | 79 | 8 | 17 |
| 110 | 23.00000 | 115.00000 | 7.4 | 76 | 8 | 18 |
| 111 | 31.00000 | 244.00000 | 10.9 | 78 | 8 | 19 |
| 112 | 44.00000 | 190.00000 | 10.3 | 78 | 8 | 20 |
| 113 | 21.00000 | 259.00000 | 15.5 | 77 | 8 | 21 |
| 114 | 9.00000 | 36.00000 | 14.3 | 72 | 8 | 22 |
| 115 | -18.92235 | 255.00000 | 12.6 | 75 | 8 | 23 |
| 116 | 45.00000 | 212.00000 | 9.7 | 79 | 8 | 24 |
| 117 | 168.00000 | 238.00000 | 3.4 | 81 | 8 | 25 |
| 118 | 73.00000 | 215.00000 | 8.0 | 86 | 8 | 26 |
| 119 | -14.86296 | 153.00000 | 5.7 | 88 | 8 | 27 |
| 120 | 76.00000 | 203.00000 | 9.7 | 97 | 8 | 28 |
| 121 | 118.00000 | 225.00000 | 2.3 | 94 | 8 | 29 |
| 122 | 84.00000 | 237.00000 | 6.3 | 96 | 8 | 30 |
| 123 | 85.00000 | 188.00000 | 6.3 | 94 | 8 | 31 |
| 124 | 96.00000 | 167.00000 | 6.9 | 91 | 9 | 1 |
| 125 | 78.00000 | 197.00000 | 5.1 | 92 | 9 | 2 |
| 126 | 73.00000 | 183.00000 | 2.8 | 93 | 9 | 3 |
| 127 | 91.00000 | 189.00000 | 4.6 | 93 | 9 | 4 |
| 128 | 47.00000 | 95.00000 | 7.4 | 87 | 9 | 5 |
| 129 | 32.00000 | 92.00000 | 15.5 | 84 | 9 | 6 |
| 130 | 20.00000 | 252.00000 | 10.9 | 80 | 9 | 7 |
| 131 | 23.00000 | 220.00000 | 10.3 | 78 | 9 | 8 |
| 132 | 21.00000 | 230.00000 | 10.9 | 75 | 9 | 9 |
| 133 | 24.00000 | 259.00000 | 9.7 | 73 | 9 | 10 |
| 134 | 44.00000 | 236.00000 | 14.9 | 81 | 9 | 11 |
| 135 | 21.00000 | 259.00000 | 15.5 | 76 | 9 | 12 |
| 136 | 28.00000 | 238.00000 | 6.3 | 77 | 9 | 13 |

```
137   9.00000  24.00000 10.9   71     9  14
138  13.00000 112.00000 11.5   71     9  15
139  46.00000 237.00000  6.9   78     9  16
140  18.00000 224.00000 13.8   67     9  17
141  13.00000  27.00000 10.3   76     9  18
142  24.00000 238.00000 10.3   68     9  19
143  16.00000 201.00000  8.0   82     9  20
144  13.00000 238.00000 12.6   64     9  21
145  23.00000  14.00000  9.2   71     9  22
146  36.00000 139.00000 10.3   81     9  23
147   7.00000  49.00000 10.3   69     9  24
148  14.00000  20.00000 16.6   63     9  25
149  30.00000 193.00000  6.9   70     9  26
150 -14.83089 145.00000 13.2   77     9  27
151  14.00000 191.00000 14.3   75     9  28
152  18.00000 131.00000  8.0   76     9  29
153  20.00000 223.00000 11.5   68     9  30
```

We can take advantage of selection helpers from `dplyrs` tidy select:

```
airquality %>%
  mutate(across(c(Ozone, Solar.R, starts_with("T")),impute_below))
```

```
        Ozone   Solar.R Wind Temp Month Day
1    41.00000 190.00000  7.4   67     5   1
2    36.00000 118.00000  8.0   72     5   2
3    12.00000 149.00000 12.6   74     5   3
4    18.00000 313.00000 11.5   62     5   4
5   -19.72321 -33.57778 14.3   56     5   5
6    28.00000 -33.07810 14.9   66     5   6
7    23.00000 299.00000  8.6   65     5   7
8    19.00000  99.00000 13.8   59     5   8
9     8.00000  19.00000 20.1   61     5   9
10  -18.51277 194.00000  8.6   69     5  10
11    7.00000 -21.37719  6.9   74     5  11
12   16.00000 256.00000  9.7   69     5  12
13   11.00000 290.00000  9.2   66     5  13
14   14.00000 274.00000 10.9   68     5  14
15   18.00000  65.00000 13.2   58     5  15
16   14.00000 334.00000 11.5   64     5  16
17   34.00000 307.00000 12.0   66     5  17
18    6.00000  78.00000 18.4   57     5  18
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 19 | 30.00000 | 322.00000 | 11.5 | 68 | 5 | 19 |
| 20 | 11.00000 | 44.00000 | 9.7 | 62 | 5 | 20 |
| 21 | 1.00000 | 8.00000 | 9.7 | 59 | 5 | 21 |
| 22 | 11.00000 | 320.00000 | 16.6 | 73 | 5 | 22 |
| 23 | 4.00000 | 25.00000 | 9.7 | 61 | 5 | 23 |
| 24 | 32.00000 | 92.00000 | 12.0 | 61 | 5 | 24 |
| 25 | -17.81863 | 66.00000 | 16.6 | 57 | 5 | 25 |
| 26 | -19.43853 | 266.00000 | 14.9 | 58 | 5 | 26 |
| 27 | -15.14310 | -24.60954 | 8.0 | 57 | 5 | 27 |
| 28 | 23.00000 | 13.00000 | 12.0 | 67 | 5 | 28 |
| 29 | 45.00000 | 252.00000 | 14.9 | 81 | 5 | 29 |
| 30 | 115.00000 | 223.00000 | 5.7 | 79 | 5 | 30 |
| 31 | 37.00000 | 279.00000 | 7.4 | 76 | 5 | 31 |
| 32 | -16.17315 | 286.00000 | 8.6 | 78 | 6 | 1 |
| 33 | -14.65883 | 287.00000 | 9.7 | 74 | 6 | 2 |
| 34 | -17.85609 | 242.00000 | 16.1 | 67 | 6 | 3 |
| 35 | -13.29299 | 186.00000 | 9.2 | 84 | 6 | 4 |
| 36 | -16.16323 | 220.00000 | 8.6 | 85 | 6 | 5 |
| 37 | -19.60935 | 264.00000 | 14.3 | 79 | 6 | 6 |
| 38 | 29.00000 | 127.00000 | 9.7 | 82 | 6 | 7 |
| 39 | -19.65780 | 273.00000 | 6.9 | 87 | 6 | 8 |
| 40 | 71.00000 | 291.00000 | 13.8 | 90 | 6 | 9 |
| 41 | 39.00000 | 323.00000 | 11.5 | 87 | 6 | 10 |
| 42 | -13.40961 | 259.00000 | 10.9 | 93 | 6 | 11 |
| 43 | -13.53728 | 250.00000 | 9.2 | 92 | 6 | 12 |
| 44 | 23.00000 | 148.00000 | 8.0 | 82 | 6 | 13 |
| 45 | -19.65993 | 332.00000 | 13.8 | 80 | 6 | 14 |
| 46 | -16.48342 | 322.00000 | 11.5 | 79 | 6 | 15 |
| 47 | 21.00000 | 191.00000 | 14.9 | 77 | 6 | 16 |
| 48 | 37.00000 | 284.00000 | 20.7 | 72 | 6 | 17 |
| 49 | 20.00000 | 37.00000 | 9.2 | 65 | 6 | 18 |
| 50 | 12.00000 | 120.00000 | 11.5 | 73 | 6 | 19 |
| 51 | 13.00000 | 137.00000 | 10.3 | 76 | 6 | 20 |
| 52 | -17.17718 | 150.00000 | 6.3 | 77 | 6 | 21 |
| 53 | -16.74073 | 59.00000 | 1.7 | 76 | 6 | 22 |
| 54 | -13.65786 | 91.00000 | 4.6 | 76 | 6 | 23 |
| 55 | -16.78786 | 250.00000 | 6.3 | 76 | 6 | 24 |
| 56 | -12.30098 | 135.00000 | 8.0 | 75 | 6 | 25 |
| 57 | -13.33171 | 127.00000 | 8.0 | 78 | 6 | 26 |
| 58 | -16.77414 | 47.00000 | 10.3 | 73 | 6 | 27 |
| 59 | -17.08225 | 98.00000 | 11.5 | 80 | 6 | 28 |
| 60 | -15.98818 | 31.00000 | 14.9 | 77 | 6 | 29 |
| 61 | -19.17558 | 138.00000 | 8.0 | 83 | 6 | 30 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 62 | 135.00000 | 269.00000 | 4.1 | 84 | 7 | 1 |
| 63 | 49.00000 | 248.00000 | 9.2 | 85 | 7 | 2 |
| 64 | 32.00000 | 236.00000 | 9.2 | 81 | 7 | 3 |
| 65 | -14.27138 | 101.00000 | 10.9 | 84 | 7 | 4 |
| 66 | 64.00000 | 175.00000 | 4.6 | 83 | 7 | 5 |
| 67 | 40.00000 | 314.00000 | 10.9 | 83 | 7 | 6 |
| 68 | 77.00000 | 276.00000 | 5.1 | 88 | 7 | 7 |
| 69 | 97.00000 | 267.00000 | 6.3 | 92 | 7 | 8 |
| 70 | 97.00000 | 272.00000 | 5.7 | 92 | 7 | 9 |
| 71 | 85.00000 | 175.00000 | 7.4 | 89 | 7 | 10 |
| 72 | -13.51764 | 139.00000 | 8.6 | 82 | 7 | 11 |
| 73 | 10.00000 | 264.00000 | 14.3 | 73 | 7 | 12 |
| 74 | 27.00000 | 175.00000 | 14.9 | 81 | 7 | 13 |
| 75 | -13.48998 | 291.00000 | 14.9 | 91 | 7 | 14 |
| 76 | 7.00000 | 48.00000 | 14.3 | 80 | 7 | 15 |
| 77 | 48.00000 | 260.00000 | 6.9 | 81 | 7 | 16 |
| 78 | 35.00000 | 274.00000 | 10.3 | 82 | 7 | 17 |
| 79 | 61.00000 | 285.00000 | 6.3 | 84 | 7 | 18 |
| 80 | 79.00000 | 187.00000 | 5.1 | 87 | 7 | 19 |
| 81 | 63.00000 | 220.00000 | 11.5 | 85 | 7 | 20 |
| 82 | 16.00000 | 7.00000 | 6.9 | 74 | 7 | 21 |
| 83 | -16.92150 | 258.00000 | 9.7 | 81 | 7 | 22 |
| 84 | -16.60335 | 295.00000 | 11.5 | 82 | 7 | 23 |
| 85 | 80.00000 | 294.00000 | 8.6 | 86 | 7 | 24 |
| 86 | 108.00000 | 223.00000 | 8.0 | 85 | 7 | 25 |
| 87 | 20.00000 | 81.00000 | 8.6 | 82 | 7 | 26 |
| 88 | 52.00000 | 82.00000 | 12.0 | 86 | 7 | 27 |
| 89 | 82.00000 | 213.00000 | 7.4 | 88 | 7 | 28 |
| 90 | 50.00000 | 275.00000 | 7.4 | 86 | 7 | 29 |
| 91 | 64.00000 | 253.00000 | 7.4 | 83 | 7 | 30 |
| 92 | 59.00000 | 254.00000 | 9.2 | 81 | 7 | 31 |
| 93 | 39.00000 | 83.00000 | 6.9 | 81 | 8 | 1 |
| 94 | 9.00000 | 24.00000 | 13.8 | 81 | 8 | 2 |
| 95 | 16.00000 | 77.00000 | 7.4 | 82 | 8 | 3 |
| 96 | 78.00000 | -30.94374 | 6.9 | 86 | 8 | 4 |
| 97 | 35.00000 | -33.38707 | 7.4 | 85 | 8 | 5 |
| 98 | 66.00000 | -21.48980 | 4.6 | 87 | 8 | 6 |
| 99 | 122.00000 | 255.00000 | 4.0 | 89 | 8 | 7 |
| 100 | 89.00000 | 229.00000 | 10.3 | 90 | 8 | 8 |
| 101 | 110.00000 | 207.00000 | 8.0 | 90 | 8 | 9 |
| 102 | -14.78907 | 222.00000 | 8.6 | 92 | 8 | 10 |
| 103 | -16.19151 | 137.00000 | 11.5 | 86 | 8 | 11 |
| 104 | 44.00000 | 192.00000 | 11.5 | 86 | 8 | 12 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 105 | 28.00000 | 273.00000 | 11.5 | 82 | 8 | 13 |
| 106 | 65.00000 | 157.00000 | 9.7 | 80 | 8 | 14 |
| 107 | -19.73591 | 64.00000 | 11.5 | 79 | 8 | 15 |
| 108 | 22.00000 | 71.00000 | 10.3 | 77 | 8 | 16 |
| 109 | 59.00000 | 51.00000 | 6.3 | 79 | 8 | 17 |
| 110 | 23.00000 | 115.00000 | 7.4 | 76 | 8 | 18 |
| 111 | 31.00000 | 244.00000 | 10.9 | 78 | 8 | 19 |
| 112 | 44.00000 | 190.00000 | 10.3 | 78 | 8 | 20 |
| 113 | 21.00000 | 259.00000 | 15.5 | 77 | 8 | 21 |
| 114 | 9.00000 | 36.00000 | 14.3 | 72 | 8 | 22 |
| 115 | -18.92235 | 255.00000 | 12.6 | 75 | 8 | 23 |
| 116 | 45.00000 | 212.00000 | 9.7 | 79 | 8 | 24 |
| 117 | 168.00000 | 238.00000 | 3.4 | 81 | 8 | 25 |
| 118 | 73.00000 | 215.00000 | 8.0 | 86 | 8 | 26 |
| 119 | -14.86296 | 153.00000 | 5.7 | 88 | 8 | 27 |
| 120 | 76.00000 | 203.00000 | 9.7 | 97 | 8 | 28 |
| 121 | 118.00000 | 225.00000 | 2.3 | 94 | 8 | 29 |
| 122 | 84.00000 | 237.00000 | 6.3 | 96 | 8 | 30 |
| 123 | 85.00000 | 188.00000 | 6.3 | 94 | 8 | 31 |
| 124 | 96.00000 | 167.00000 | 6.9 | 91 | 9 | 1 |
| 125 | 78.00000 | 197.00000 | 5.1 | 92 | 9 | 2 |
| 126 | 73.00000 | 183.00000 | 2.8 | 93 | 9 | 3 |
| 127 | 91.00000 | 189.00000 | 4.6 | 93 | 9 | 4 |
| 128 | 47.00000 | 95.00000 | 7.4 | 87 | 9 | 5 |
| 129 | 32.00000 | 92.00000 | 15.5 | 84 | 9 | 6 |
| 130 | 20.00000 | 252.00000 | 10.9 | 80 | 9 | 7 |
| 131 | 23.00000 | 220.00000 | 10.3 | 78 | 9 | 8 |
| 132 | 21.00000 | 230.00000 | 10.9 | 75 | 9 | 9 |
| 133 | 24.00000 | 259.00000 | 9.7 | 73 | 9 | 10 |
| 134 | 44.00000 | 236.00000 | 14.9 | 81 | 9 | 11 |
| 135 | 21.00000 | 259.00000 | 15.5 | 76 | 9 | 12 |
| 136 | 28.00000 | 238.00000 | 6.3 | 77 | 9 | 13 |
| 137 | 9.00000 | 24.00000 | 10.9 | 71 | 9 | 14 |
| 138 | 13.00000 | 112.00000 | 11.5 | 71 | 9 | 15 |
| 139 | 46.00000 | 237.00000 | 6.9 | 78 | 9 | 16 |
| 140 | 18.00000 | 224.00000 | 13.8 | 67 | 9 | 17 |
| 141 | 13.00000 | 27.00000 | 10.3 | 76 | 9 | 18 |
| 142 | 24.00000 | 238.00000 | 10.3 | 68 | 9 | 19 |
| 143 | 16.00000 | 201.00000 | 8.0 | 82 | 9 | 20 |
| 144 | 13.00000 | 238.00000 | 12.6 | 64 | 9 | 21 |
| 145 | 23.00000 | 14.00000 | 9.2 | 71 | 9 | 22 |
| 146 | 36.00000 | 139.00000 | 10.3 | 81 | 9 | 23 |
| 147 | 7.00000 | 49.00000 | 10.3 | 69 | 9 | 24 |

```
148  14.00000  20.00000 16.6   63     9  25
149  30.00000 193.00000  6.9   70     9  26
150 -14.83089 145.00000 13.2   77     9  27
151  14.00000 191.00000 14.3   75     9  28
152  18.00000 131.00000  8.0   76     9  29
153  20.00000 223.00000 11.5   68     9  30
```

## 13.3 Tracking missing values

We need to track the missing values, once we impute them. Otherwise we don't know what was imputed and what was not. We can see that in this example, once we impute the data, we have no way to recognise which one it is.

```r
df <- tibble(var1 = c(5, 6, 7, NA, 9, 10))
df
```

```
# A tibble: 6 x 1
   var1
  <dbl>
1     5
2     6
3     7
4    NA
5     9
6    10
```

```r
df %>%
  mutate(across(everything(),impute_below))
```

```
# A tibble: 6 x 1
   var1
  <dbl>
1  5
2  6
3  7
4  4.40
5  9
6 10
```

We can identify missings by using `nabular` to turn the data into `nabular` form.

```
nabular(df)
```

```
# A tibble: 6 x 2
   var1 var1_NA
  <dbl> <fct>  
1     5 !NA    
2     6 !NA    
3     7 !NA    
4    NA NA     
5     9 !NA    
6    10 !NA    
```

Now when we impute the data, we can see that the shadow variable, `var1_NA` reveals the imputed value, 4.40.

```
df %>%
  nabular() %>%
  mutate(across(everything(),impute_below))
```

```
# A tibble: 6 x 2
   var1 var1_NA
  <dbl> <fct>  
1  5    !NA    
2  6    !NA    
3  7    !NA    
4  4.40 NA     
5  9    !NA    
6 10    !NA    
```

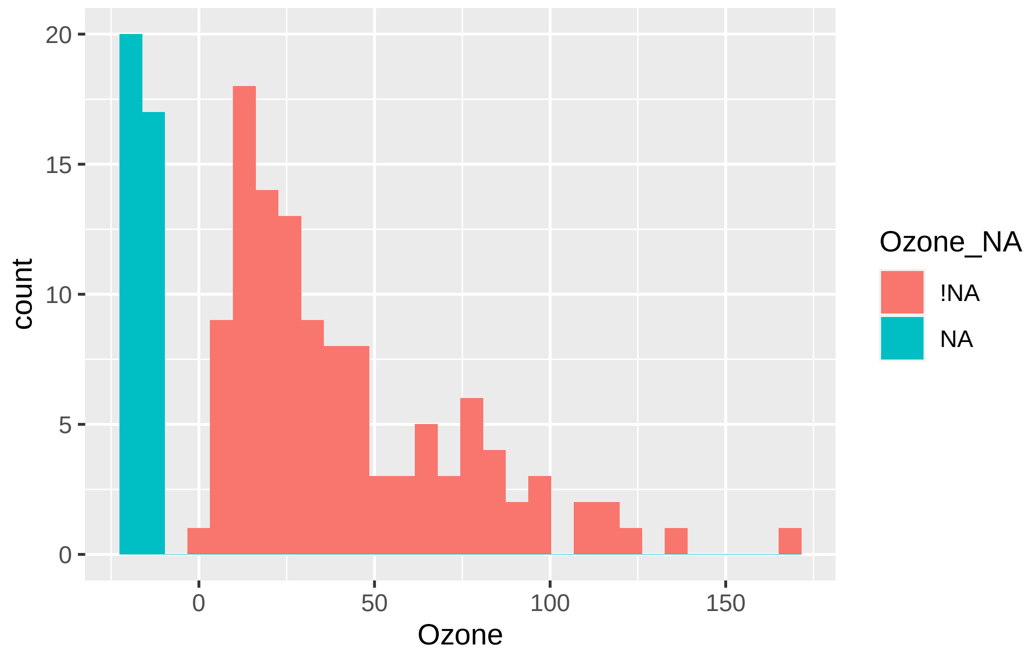
## 13.4 Visualise imputed values against data values using histograms

Using this imputed data, we can explore the number of missings in a single variable, along with it's distribution, using a histogram and colouring the missings using `fill = Ozone_NA`.

```
aq_imp <- airquality %>%
   nabular() %>%
   mutate(across(everything(),impute_below))

ggplot(aq_imp,
```

```
        aes(x = Ozone,
            fill = Ozone_NA)) +
    geom_histogram()
```

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




Here we see that there are a few missing values - two bars around 20, so just under 40 missing values.

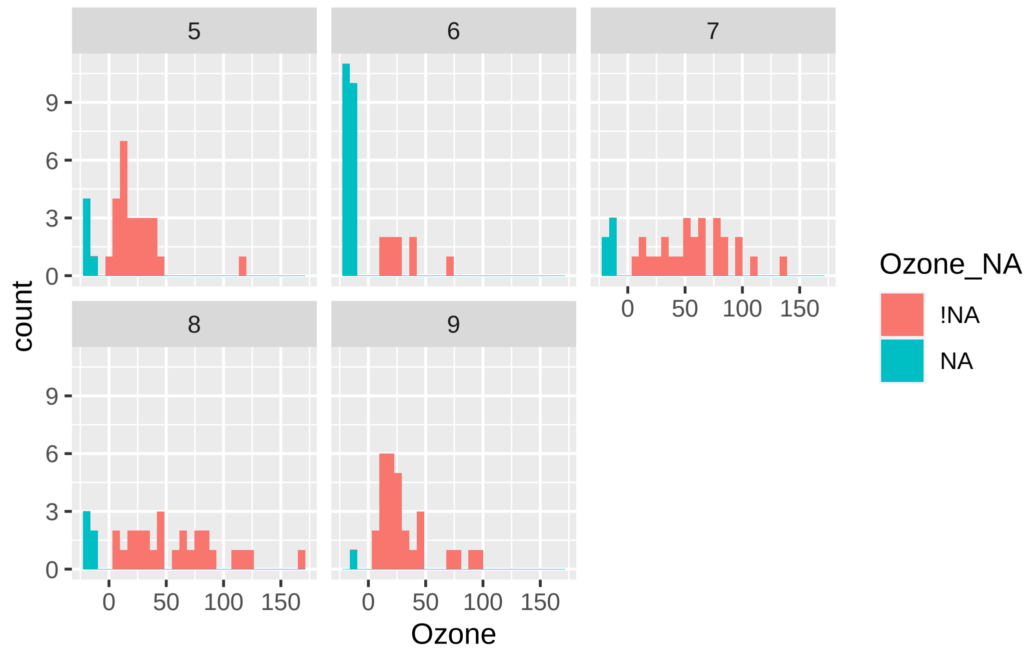
## 13.5 Visualise imputed values against data values using facets

We can take this same plot and visualise it across facets. For example, plot it by month, which shows us that most missing values occur in month 6 - which didn't have many high values of ozone.

```
ggplot(aq_imp,
       aes(x = Ozone,
           fill = Ozone_NA)) +
  geom_histogram() +
```

```
  facet_wrap(~Month)
```

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

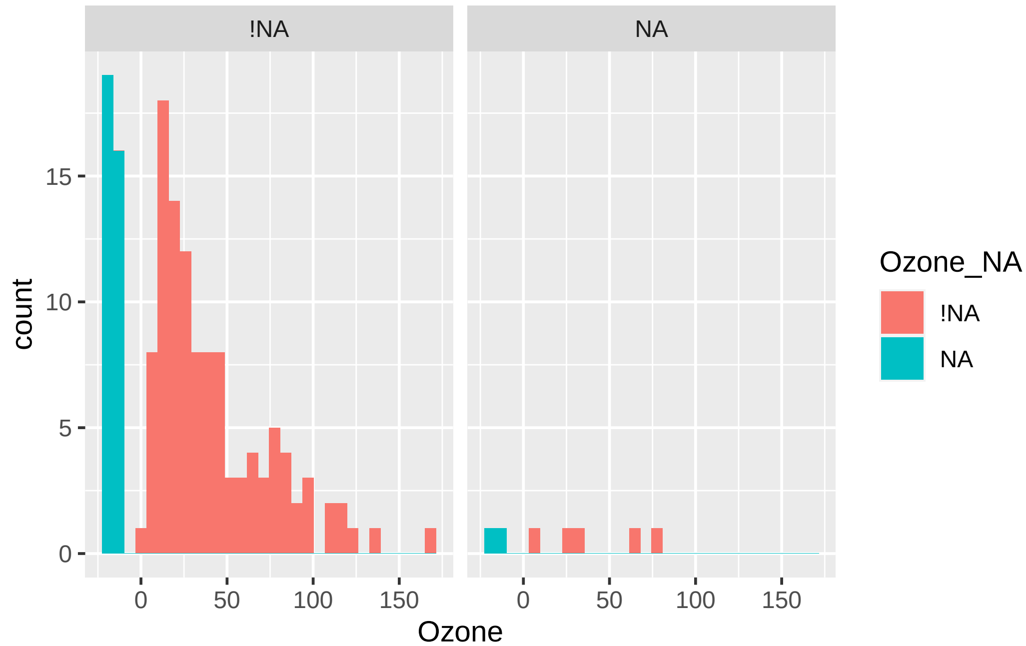



## 13.6 Visualize imputed values using facets

We can split the plot according to the missingness of solar radiation by referring to it as `Solar.R_NA`

```
ggplot(aq_imp,
       aes(x = Ozone,
           fill = Ozone_NA)) +
  geom_histogram() +
  facet_wrap(~Solar.R_NA)
```

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




This shows us that there aren't many missing values in ozone when solar radiation is missing.

## 13.7 Visualize imputed values against data values using scatterplots

Previously we could identify imputed values by referring to the shadow variable - e.g., `Ozone_NA`. However, if you want to colour by two variables, you just need to know **if any of them were imputed**. We can add a column with labels to identify whether there is a missing value in a column. The function `add_label_missings` does this for us, adding a column, `any_missing`.

```
aq_imp <- airquality %>%
  nabular() %>%
  add_label_missings() %>%
  mutate(across(everything(),impute_below))

aq_imp
```

```
# A tibble: 153 x 13
   Ozone Solar.R  Wind  Temp Month   Day Ozone_NA Solar.R_NA Wind_NA Temp_NA
   <dbl>   <dbl> <dbl> <int> <int> <int> <fct>    <fct>      <fct>   <fct>
 1  41       190     7.4    67     5     1 !NA      !NA        !NA     !NA
```

```
 2  36      118      8       72      5      2 !NA       !NA         !NA      !NA
 3  12      149     12.6     74      5      3 !NA       !NA         !NA      !NA
 4  18      313     11.5     62      5      4 !NA       !NA         !NA      !NA
 5 -19.7    -33.6   14.3     56      5      5 NA        NA          !NA      !NA
 6  28      -33.1   14.9     66      5      6 !NA       NA          !NA      !NA
 7  23      299      8.6     65      5      7 !NA       !NA         !NA      !NA
 8  19       99     13.8     59      5      8 !NA       !NA         !NA      !NA
 9   8       19     20.1     61      5      9 !NA       !NA         !NA      !NA
10 -18.5    194      8.6     69      5     10 NA        !NA         !NA      !NA
# ... with 143 more rows, and 3 more variables: Month_NA <fct>, Day_NA <fct>,
#   any_missing <chr>
```

We can now recreate the same figure as `geom_miss_point()`!

```
ggplot(aq_imp,
       aes(x = Ozone,
           y = Solar.R,
           colour = any_missing)) +
  geom_point()
```

# 14 Assessing imputation

```
library(naniar)
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.1 --

v ggplot2 3.3.6     v purrr   0.3.4
v tibble  3.1.7     v dplyr   1.0.9
v tidyr   1.2.0     v stringr 1.4.0
v readr   2.1.2     v forcats 0.5.1

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

In this chapter we discuss whether imputation is appropriate, and the features of good and bad imputations. You will learn how to evaluate imputed values by using visualisations to assess their summary features: the mean/median, scale, and spread.

## 14.1 What makes a good imputation

Imputing missing values needs to be done with care - you want to avoid imputing unlikely values like mid winter temperatures into the middle of summer, giving pigs a wing span measurement, or heavy rainfall into a known drought.

## 14.2 When to impute

Not everything can be solved with imputation. If you don't already have the information, sometimes it is not appropriate to impute variables like, age, race, sex and gender without first confirming known facts about your population. Sometimes this means talking to the person who collected or curated the data to understand the population studied. You might learn some variables remain fixed over time, and so can be perfectly imputed. Other times, we might not know, so leaving the values as missing might be the most appropriate action. Imputation isn't always the answer.

Another consideration for imputation is when there is simply too much missing data. For example, if you have a variable with 50% of the values missing, imputation might not be appropriate unless you have a very strong modelling case. For example, you know all the ages were not recorded for every person, but were the same for each person, so you can impute perfectly. When there are missing values in the outcome variable (the "Y", the Dependent Variable, DV, it has many names!), it is generally not a good idea to impute data for these values. The reason is that you are effectively using the data twice.

[TODO: add a small simulation on this]

[TODO: add caveats around Bayesian/likelihood simulation approaches]

## 14.3 Understanding the good by understanding the bad

To understand good imputation, it is useful to understand bad imputations. One particularly bad imputation is mean imputation, which takes the mean of complete values as the imputed value.

For example, in a dataframe with 5 values and one missing, we calculate the mean from complete observations using `na.rm = TRUE`, and use this to impute the missing values. The steps are shown here:

```
df <- tibble(x = c(1, 4, 9, 16, NA, 36))
df
```

```
# A tibble: 6 x 1
      x
  <dbl>
1     1
2     4
3     9
4    16
5    NA
6    36
```

```
mean(df$x, na.rm = TRUE)
```

```
[1] 13.2
```

```r
df[is.na(df)] <- mean(df$x, na.rm = TRUE)
df
```

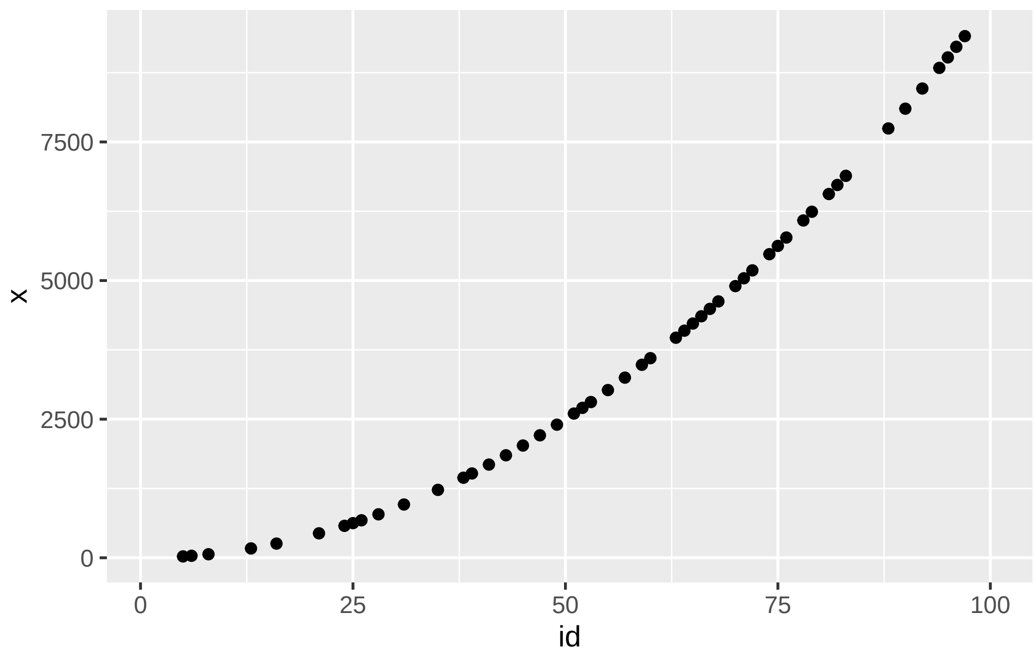
```
# A tibble: 6 x 1
      x
  <dbl>
1   1
2   4
3   9
4  16
5  13.2
6  36
```

### 14.3.1 Demonstrating mean imputation

This is generally a terribly idea. For example, imagine we had data like this, with missing values:

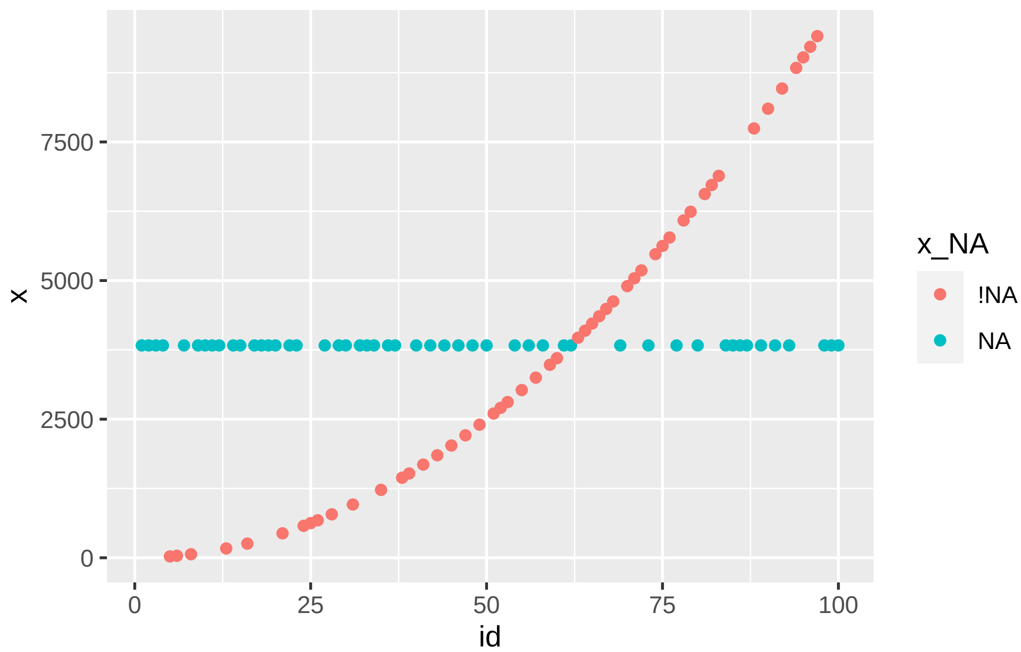
```r
ggplot(df,
       aes(x = id,
           y = x)) +
  geom_point()
```

```
Warning: Removed 50 rows containing missing values (geom_point).
```




Imputing the mean value in this graph, we get this:

```
df %>%
  nabular() %>%
  mutate(across(everything(), impute_mean)) %>%
  ggplot(aes(x = id,
             y = x,
             colour = x_NA)) +
  geom_point()
```




The mean does not respect the underlying process of the data. Visualisation is a very key tool here to explore and demonstrate this pattern.

### 14.3.2 Explore bad imputations: The mean

To examine these bad imputations, we use the `impute_mean` function from the `naniar` package. Similar to `impute_below` used in the previous chapter, we can use `across` and friends with `impute_mean`. So it can work on a vector, on variables based on some condition like are they numeric, for specified variables, or for all variables.

### 14.3.3 Tracking missing values

To visualise imputations we use the same process as for `impute_below`:

```
aq_impute_mean <- airquality %>%
  nabular(only_miss = TRUE) %>%
  mutate(across(everything(), impute_mean)) %>%
  add_label_shadow()

aq_impute_mean
```

```
# A tibble: 153 x 9
   Ozone Solar.R  Wind  Temp Month   Day Ozone_NA Solar.R_NA any_missing
   <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl> <fct>    <fct>      <chr>
 1  41       190   7.4    67     5     1 !NA      !NA        Not Missing
 2  36       118   8      72     5     2 !NA      !NA        Not Missing
 3  12       149  12.6    74     5     3 !NA      !NA        Not Missing
 4  18       313  11.5    62     5     4 !NA      !NA        Not Missing
 5  42.1     186. 14.3    56     5     5 NA       NA         Missing
 6  28       186. 14.9    66     5     6 !NA      NA         Missing
 7  23       299   8.6    65     5     7 !NA      !NA        Not Missing
 8  19        99  13.8    59     5     8 !NA      !NA        Not Missing
 9   8        19  20.1    61     5     9 !NA      !NA        Not Missing
10  42.1     194   8.6    69     5    10 NA       !NA        Missing
# ... with 143 more rows
```

We first create nabular data to track missing values. Then, we do our imputations. Then, we add a label to identify cases with missing observations using `add_label_shadow()`. One thing to have up your sleeve it `only_miss` option, which binds only columns with missing values. This makes the data bit smaller and easier to handle.

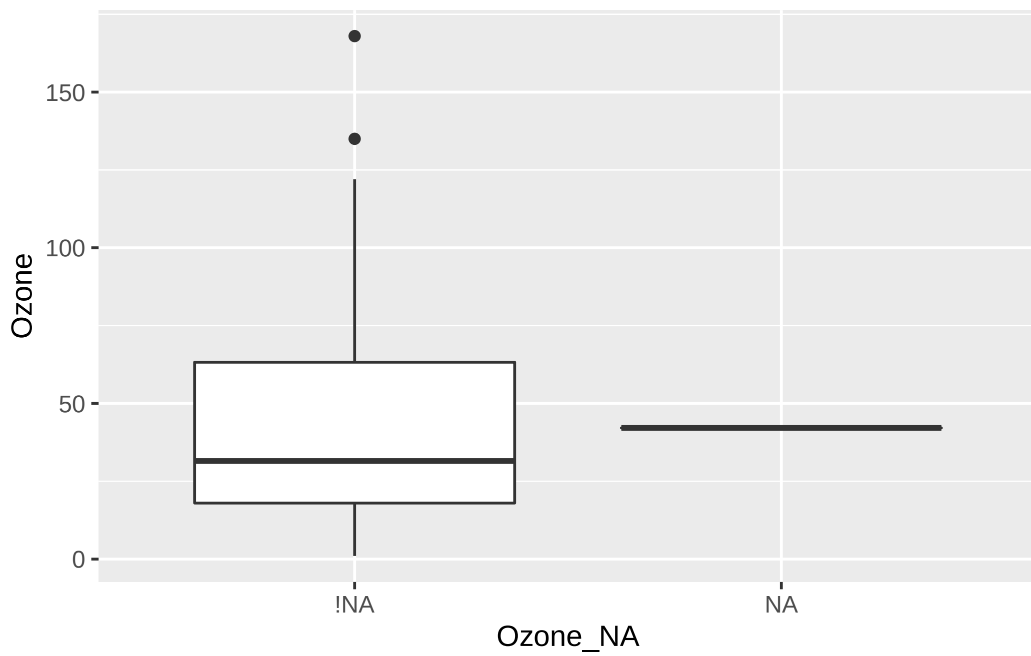
Now that we know a way to impute our data, let's explore it. We can explore the imputed values in the same way we did for the previous lesson. But this time our intention is different, and we want to consider evaluating imputations by looking for changes in the **mean**, the **spread**, and the **scale**.

[TODO **A small figure/plot that clearly shows what we mean by mean, spread, scale]**

### 14.3.4 Using a boxplot to explore how the mean changes

We can evaluate changes in the mean or median using a boxplot. We put the missingness of ozone, `ozone_NA`, on the x axis, and the values of ozone on the y axis, and use `geom_boxplot`.

```
ggplot(aq_impute_mean,
       aes(x = Ozone_NA,
           y = Ozone)) +
  geom_boxplot()
```




From this visualisation, we learn the median value is similar in each group, but the median is lower for the not missing group. The take away message is the mean isn't changing. This is good, but there is more than one feature to explore!

## 14.4 Using a scatterplot to Explore how spread changes with imputation

The **spread** of imputations can be explored using a scatter plot. We plot our airquality imputed with the mean, with Ozone and solar radiation on the x and y axis, and colouring according to missingness, `any_missing`.

```
ggplot(aq_impute_mean,
       aes(x = Ozone,
           y = Solar.R,
           colour = any_missing)) +
  geom_point()
```


We learn there is no variation in the spread of the points! Although we do notice the imputed values are within a reasonable range of the data.

### 14.4.1 How to explore imputations for many variables

```
aq_imp <- airquality %>%
  nabular() %>%
  mutate(across(everything(), impute_mean))
```

To make it easier to explore many variables, we use the `shadow_long` function to return `nabular` data in long format. This is similar to `pivot_longer`, but with our nabular data.

```
aq_imp_long <- shadow_long(aq_imp,
                           Ozone,
                           Solar.R)

aq_imp_long
```

```
# A tibble: 306 x 4
   variable value variable_NA value_NA
```

```
   <chr>    <dbl> <chr>       <chr>
 1 Ozone     41   Ozone_NA    !NA
 2 Ozone     36   Ozone_NA    !NA
 3 Ozone     12   Ozone_NA    !NA
 4 Ozone     18   Ozone_NA    !NA
 5 Ozone     42.1 Ozone_NA    NA
 6 Ozone     28   Ozone_NA    !NA
 7 Ozone     23   Ozone_NA    !NA
 8 Ozone     19   Ozone_NA    !NA
 9 Ozone      8   Ozone_NA    !NA
10 Ozone     42.1 Ozone_NA    NA
# ... with 296 more rows
```

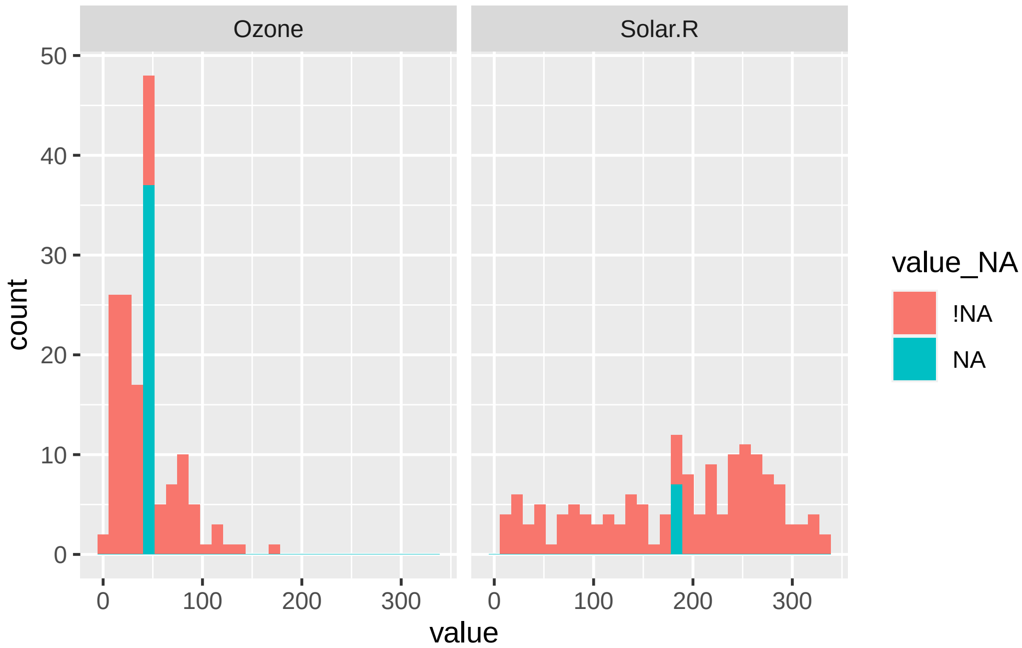
Here, we enter in our data, followed by the variables that we want to focus on - in this case, `Ozone` and `Solar.R`. This returns to us data with the columns `variable`, `value`, and the shadow columns, `variable_NA` and `value_NA`.

### 14.4.2 Exploring imputations for many variables

We can then use this in a ggplot, placing `value` in the x axis, and filling by the missingness of the value, `value_NA`, and then using `geom_histogram`, facetting by `variable`.

```
ggplot(aq_imp_long,
       aes(x = value,
           fill = value_NA)) +
  geom_histogram() +
  facet_wrap(~variable)
```

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




The format from `shadow_long` makes it simpler to explore missing values for many variables.

**[TODO perhaps another example of this with other data]**

# 15 Imputing with different models

There are many imputation packages in R. We are going to focus on using the `simputation` package by Mark van der Loo. `simputation` provides a simple interface to many imputation models. We will impute values using a linear model, using the function `impute_lm()`.

Building a good imputation model is super important, but it is a complex topic - there is as much to building a good imputation model as there is for building a good statistical model. We focus on how to build up different imputation models and assess and compare them.

```r
library(naniar)
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.1 --

v ggplot2 3.3.6     v purrr   0.3.4
v tibble  3.1.7     v dplyr   1.0.9
v tidyr   1.2.0     v stringr 1.4.0
v readr   2.1.2     v forcats 0.5.1

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

```r
library(simputation)
```

```
Attaching package: 'simputation'

The following object is masked from 'package:naniar':

    impute_median
```

When you develop imputation models, it is a good idea to try out a few different models, to see how the imputed values change according to your assumptions. In this chapter, we are going to impute data using linear regression.

**An Aside: Running many models used to make me feel like I was cheating**

> I (Nick), studied psychology in my undergraduate degree. The training for statistics was very focussed on developing a theory first, then designing a study design and appropriate statistial model. You'd then collect data, and turn the cranks on the statistical (usually ANOVA) machinery and out would come your answer - is the result significant: yes, or no? When I started my PhD in statistics, the idea of exploring many different fits to your data felt strange, and like I was cheating! Fitting many models in psych would have felt like we were chasing significance, and that I'd be labelled a "p-value hacker". In these cases, however, we weren't conducting controlled experiments to test a theory, rather we were exploring data **that wasn't collected with a research question in mind**. In our case with missing data, fitting many models is about trying to identify the most sensible values that could have existed, so that we can draw sensible inferences on the data.

## 15.1 How imputation using a linear model works

We previously explored using mean imputation. This is generally a bad imputation method to use, as it artificially increases the mean and reduces variance, so you aren't capturing the natural variation in the data. Similar to how the mean was imputed, we can use a linear model to impute data. This can take into account some features of the data, to better predict missing values.

To impute values using a linear model, we use `impute_lm` from `simputation`. Let's create some fake data, `df`:

```
df <- tibble(
  y = c(2.67, 3.87, NA, 5.21, NA),
  x1 = c(2.43, 3.55, 2.9, 2.72, 4.29),
  x2 = c(3.27, 1.45, 1.49, 1.84, 1.15)
  )

df
```

```
# A tibble: 5 x 3
      y    x1    x2
  <dbl> <dbl> <dbl>
1  2.67  2.43  3.27
```

```
2  3.87  3.55  1.45
3 NA     2.9   1.49
4  5.21  2.72  1.84
5 NA     4.29  1.15
```

Now to use `impute_lm()`, we specify the variable we would like to impute on as the `y` or dependent variable, just as you would with a linear model. On the right hand side of the formula are the variables we would like to use to inform the imputations on the right hand side. This returns a data frame with imputed values in `y`:

```
df %>%
  impute_lm(y ~ x1 + x2)
```

```
# A tibble: 5 x 3
      y    x1    x2
* <dbl> <dbl> <dbl>
1  2.67  2.43  3.27
2  3.87  3.55  1.45
3  5.54  2.9   1.49
4  5.21  2.72  1.84
5  2.56  4.29  1.15
```

Of course, we need to use **nabular** to make sure we can track what values were imputed in `y_NA`:

```
df %>%
  nabular() %>%
  impute_lm(y ~ x1 + x2)
```

```
# A tibble: 5 x 6
      y    x1    x2 y_NA  x1_NA x2_NA
* <dbl> <dbl> <dbl> <fct> <fct> <fct>
1  2.67  2.43  3.27 !NA   !NA   !NA
2  3.87  3.55  1.45 !NA   !NA   !NA
3  5.54  2.9   1.49 NA    !NA   !NA
4  5.21  2.72  1.84 !NA   !NA   !NA
5  2.56  4.29  1.15 NA    !NA   !NA
```

## 15.2 Using `impute_lm`

Using airquality data, we can impute the values in `Solar.R` using `Wind`, `Temp`, and `Month`, and chain another imputation step in to impute `Ozone` with the same variables. This gives us imputations like the following:

```
aq_imp_lm <- airquality %>%
  nabular() %>%
  add_label_shadow() %>%
  as.data.frame() %>%
  impute_lm(Solar.R ~ Wind + Temp + Month) %>%
  impute_lm(Ozone ~ Wind + Temp + Month) %>%
  as_tibble()

aq_imp_lm
```

```
# A tibble: 153 x 13
   Ozone Solar.R  Wind  Temp Month   Day Ozone_NA Solar.R_NA Wind_NA Temp_NA
   <dbl>   <dbl> <dbl> <int> <int> <int> <fct>    <fct>      <fct>   <fct>  
 1 41        190    7.4    67     5     1 !NA      !NA        !NA     !NA    
 2 36        118    8      72     5     2 !NA      !NA        !NA     !NA    
 3 12        149   12.6    74     5     3 !NA      !NA        !NA     !NA    
 4 18        313   11.5    62     5     4 !NA      !NA        !NA     !NA    
 5 -9.04     138.  14.3    56     5     5 NA       NA         !NA     !NA    
 6 28        178.  14.9    66     5     6 !NA      NA         !NA     !NA    
 7 23        299    8.6    65     5     7 !NA      !NA        !NA     !NA    
 8 19         99   13.8    59     5     8 !NA      !NA        !NA     !NA    
 9  8         19   20.1    61     5     9 !NA      !NA        !NA     !NA    
10 35.2      194    8.6    69     5    10 NA       !NA        !NA     !NA    
# ... with 143 more rows, and 3 more variables: Month_NA <fct>, Day_NA <fct>,
#   any_missing <chr>
```

Note: the `as.data.frame()` is necessary for the time being due to a workaround with `simputation`. Nick is hoping to arrive on a better solution to this soon.

### 15.2.1 Tracking missing values

An important part of imputing data is using the `nabular()` and `add_label_shadow()` functions. Without them, we can't identify which values were missing! So, let's recap what they do.

- `nabular()` adds the variables with `_NA` to the data

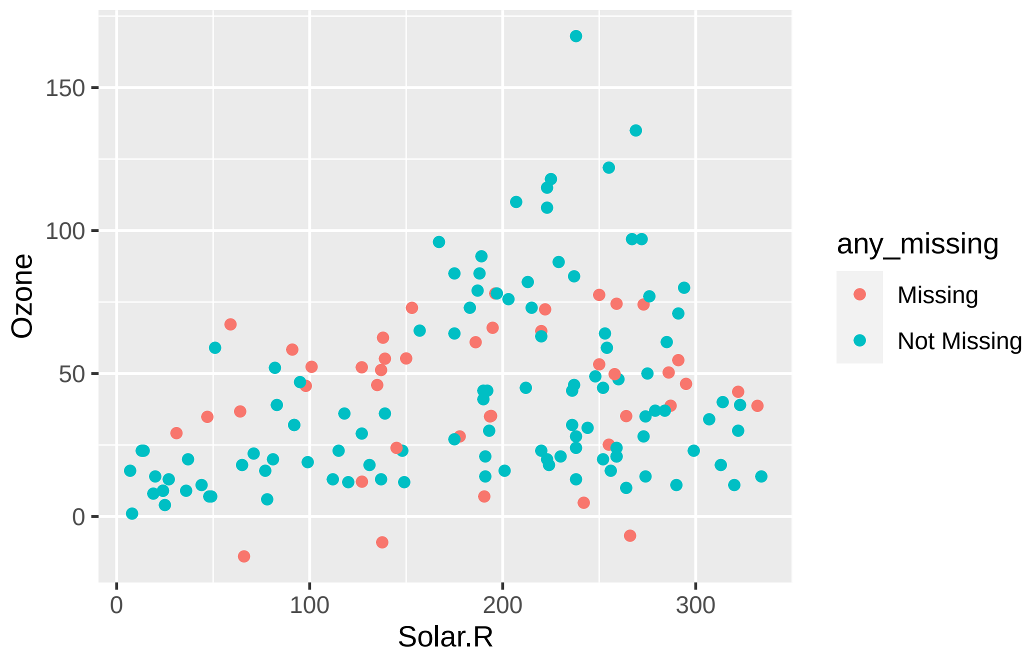
[**TODO: image of this**]

- `add_label_shadow()` adds a variable, `any_missing`, with values "Missing" or "Not Missing".We can use ggplot to show the imputed values, by setting `colour = any_missing` in a ggplot.

```
aq_imp_lm <- airquality %>%
  nabular() %>%
  add_label_missings() %>%
  as.data.frame() %>%
  impute_lm(Solar.R ~ Wind + Temp + Month) %>%
  impute_lm(Ozone ~ Wind + Temp + Month) %>%
  as_tibble()

aq_imp_lm
```

```
# A tibble: 153 x 13
   Ozone Solar.R  Wind  Temp Month   Day Ozone_NA Solar.R_NA Wind_NA Temp_NA
   <dbl>   <dbl> <dbl> <int> <int> <int> <fct>    <fct>      <fct>   <fct>
 1 41        190   7.4    67     5     1 !NA      !NA        !NA     !NA
 2 36        118   8      72     5     2 !NA      !NA        !NA     !NA
 3 12        149  12.6    74     5     3 !NA      !NA        !NA     !NA
 4 18        313  11.5    62     5     4 !NA      !NA        !NA     !NA
 5 -9.04     138.  14.3    56     5     5 NA       NA         !NA     !NA
 6 28        178.  14.9    66     5     6 !NA      NA         !NA     !NA
 7 23        299   8.6    65     5     7 !NA      !NA        !NA     !NA
 8 19         99  13.8    59     5     8 !NA      !NA        !NA     !NA
 9  8         19  20.1    61     5     9 !NA      !NA        !NA     !NA
10 35.2      194   8.6    69     5    10 NA       !NA        !NA     !NA
# ... with 143 more rows, and 3 more variables: Month_NA <fct>, Day_NA <fct>,
#   any_missing <chr>
```

```
ggplot(aq_imp_lm,
       aes(x = Solar.R,
           y = Ozone,
           colour = any_missing)) +
  geom_point()
```




Without the `any_missing` variable, we could only identify the missings of one variable: `Solar.R_NA` or `Ozone_NA`.

## 15.3 Evaluating imputations: Evaluating and comparing imputations

```
aq_imp_small <- airquality %>%
  nabular() %>%
  as.data.frame() %>%
  impute_lm(Ozone ~ Wind + Temp) %>%
  impute_lm(Solar.R ~ Wind + Temp) %>%
  add_label_shadow()

aq_imp_large <- airquality %>%
  nabular() %>%
  as.data.frame() %>%
  impute_lm(Ozone ~ Wind + Temp + Month + Day) %>%
  impute_lm(Solar.R ~ Wind + Temp + Month + Day)  %>%
  add_label_shadow()
```

When you build up an imputation model, it is good practice to compare it to an alternative method. Let's compare two linear regression imputation models. The first with two variables: Wind, and Temperature, and the second with four: Wind, Temperature, Month, and Day.

To facilitate comparing models, we put them into the same dataframe, by binding their rows together using `bind_rows` from `dplyr` package.

```
bound_models <- bind_rows(
  small = aq_imp_small,
  large = aq_imp_large,
  .id = "imp_model"
  ) %>%
  as_tibble()
```

To help us identify which data came from which imputation process, we use the `.id` argument to add a new column that identifies them. By writing `small = aq_imp_small` and `large = aq_imp_large`, we can then use `.id = "imp_model"`. This creates a dataset of all the imputations with an extra column, `imp_model`.

```
head(bound_models)
```

```
# A tibble: 6 x 14
  imp_model Ozone Solar.R  Wind  Temp Month   Day Ozone_NA Solar.R_NA Wind_NA
  <chr>     <dbl>   <dbl> <dbl> <int> <int> <int> <fct>    <fct>      <fct>
1 small      41       190   7.4    67     5     1 !NA      !NA        !NA
2 small      36       118   8      72     5     2 !NA      !NA        !NA
3 small      12       149  12.6    74     5     3 !NA      !NA        !NA
4 small      18       313  11.5    62     5     4 !NA      !NA        !NA
5 small     -11.7     127.  14.3    56     5     5 NA       NA         !NA
6 small      28       160.  14.9    66     5     6 !NA      NA         !NA
# ... with 4 more variables: Temp_NA <fct>, Month_NA <fct>, Day_NA <fct>,
#   any_missing <chr>
```

```
tail(bound_models)
```

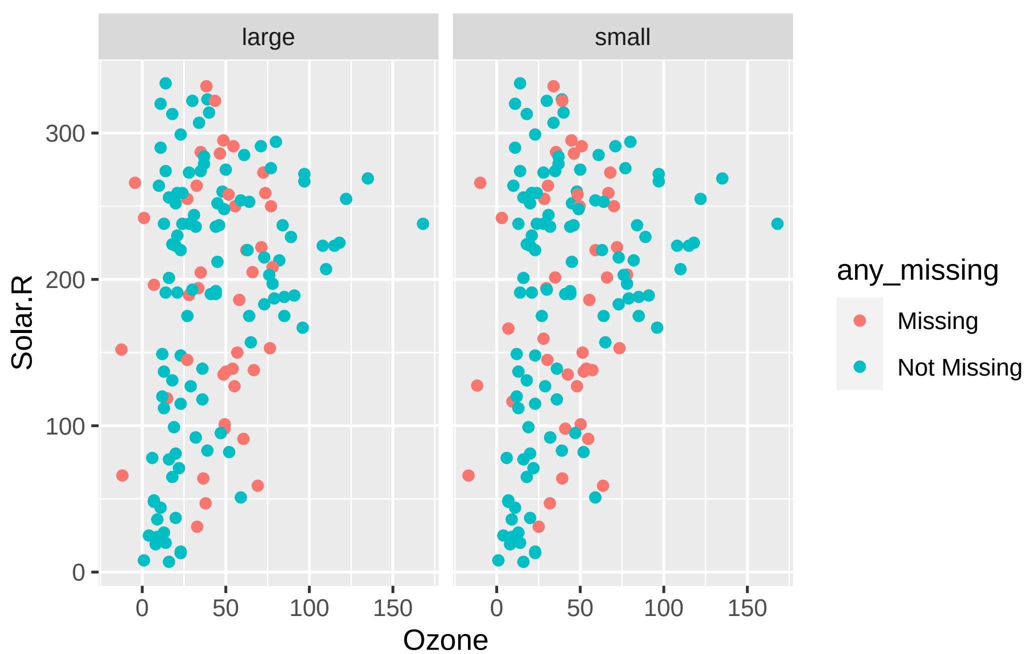
```
# A tibble: 6 x 14
  imp_model Ozone Solar.R  Wind  Temp Month   Day Ozone_NA Solar.R_NA Wind_NA
  <chr>     <dbl>   <dbl> <dbl> <int> <int> <int> <fct>    <fct>      <fct>
1 large      14        20  16.6    63     9    25 !NA      !NA        !NA
2 large      30       193   6.9    70     9    26 !NA      !NA        !NA
3 large      26.9     145  13.2    77     9    27 NA       !NA        !NA
```

```
4 large      14        191  14.3    75     9     28 !NA       !NA         !NA
5 large      18        131   8      76     9     29 !NA       !NA         !NA
6 large      20        223  11.5    68     9     30 !NA       !NA         !NA
# ... with 4 more variables: Temp_NA <fct>, Month_NA <fct>, Day_NA <fct>,
#   any_missing <chr>
```

## 15.4 Evaluating imputations: exploring many imputations

We can look at the values of `Ozone` and `Solar.R` on a scatterplot, colouring by any missings, and facetting by imputation model used: `imp_model`.

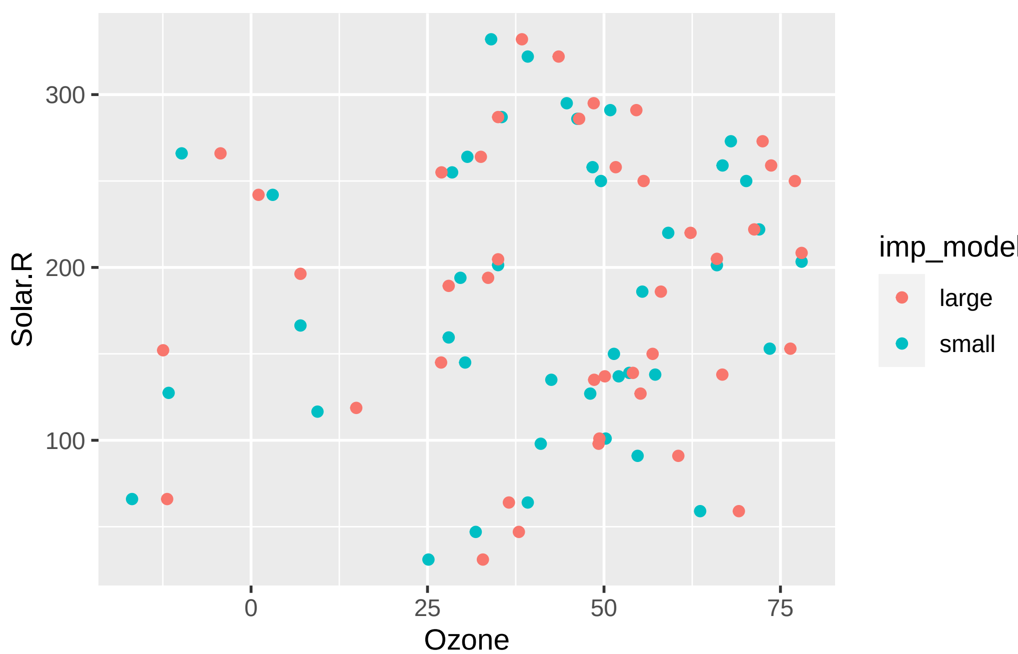
```
ggplot(bound_models,
       aes(x = Ozone,
           y = Solar.R,
           colour = any_missing)) +
  geom_point() +
  facet_wrap(~imp_model)
```




We learn there isn't much difference between either of the linear model methods for imputing data - they both seem to be within the range of the data, but both models did not impute three values.

We can explore only the imputed values against each other by filtering to just “missing” (imputed) values, then plotting them, colouring by the different imputation model.

```
bound_models %>%
  filter(any_missing == "Missing") %>%
  ggplot(aes(x = Ozone,
             y = Solar.R,
             colour = imp_model)) +
  geom_point()
```




We learn that there appears to be some difference in the imputed values, with perhaps the large imputation model imputing higher ozone levels than the small imputation model. We’ll go into a bit more detail on comparing between variable4s in the next section.

## 15.5 Explore imputations in multiple variables and models

To explore the imputations across these different models and variables, we gather the selected four variables, Ozone, Solar Radiation, `any_missing`, and `imp_model`, and then we pivot the `Ozone` and `Solar.R` variables into longer form with `pivot_longer()`.

```r
bound_models_gather <- bound_models %>%
  select(Ozone,
         Solar.R,
         any_missing,
         imp_model) %>%
  pivot_longer(cols = c(Ozone, Solar.R),
               names_to = "variable",
               values_to = "value")

bound_models_gather
```

```
# A tibble: 612 x 4
   any_missing imp_model variable value
   <chr>       <chr>     <chr>    <dbl>
 1 Not Missing small     Ozone     41
 2 Not Missing small     Solar.R  190
 3 Not Missing small     Ozone     36
 4 Not Missing small     Solar.R  118
 5 Not Missing small     Ozone     12
 6 Not Missing small     Solar.R  149
 7 Not Missing small     Ozone     18
 8 Not Missing small     Solar.R  313
 9 Missing     small     Ozone    -11.7
10 Missing     small     Solar.R  127.
# ... with 602 more rows
```

This gives us the columns, `any_missing`, `imp_model`, `variable`, and `value`:

```r
head(bound_models_gather)
```

```
# A tibble: 6 x 4
  any_missing imp_model variable value
  <chr>       <chr>     <chr>    <dbl>
1 Not Missing small     Ozone       41
2 Not Missing small     Solar.R    190
3 Not Missing small     Ozone       36
4 Not Missing small     Solar.R    118
5 Not Missing small     Ozone       12
6 Not Missing small     Solar.R    149
```

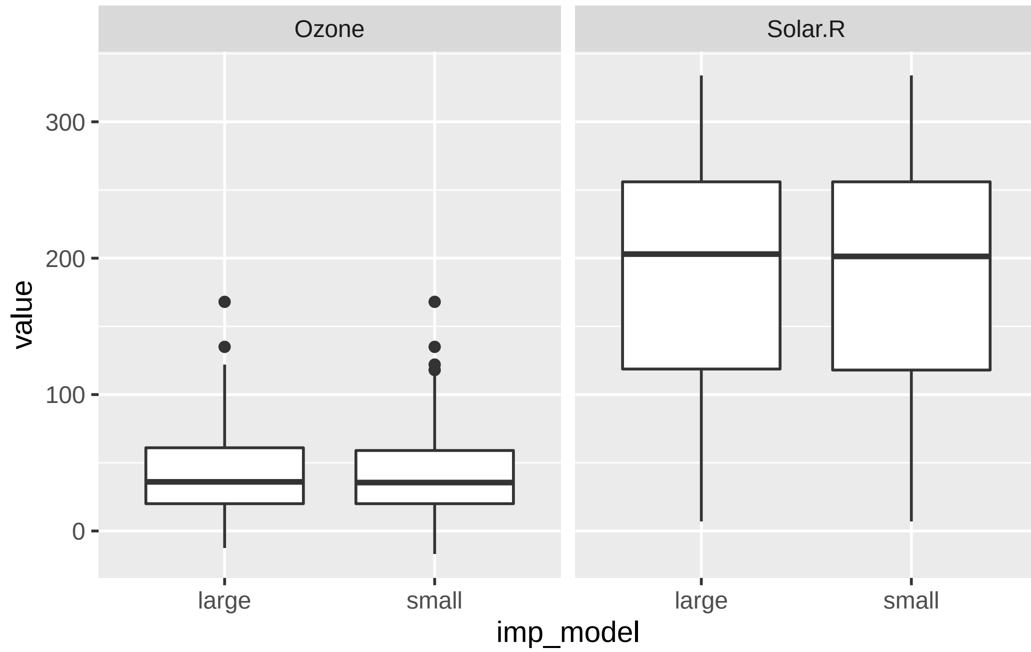
```
tail(bound_models_gather)
```

```
# A tibble: 6 x 4
  any_missing imp_model variable value
  <chr>       <chr>     <chr>    <dbl>
1 Not Missing large     Ozone       14
2 Not Missing large     Solar.R    191
3 Not Missing large     Ozone       18
4 Not Missing large     Solar.R    131
5 Not Missing large     Ozone       20
6 Not Missing large     Solar.R    223
```

### 15.5.1 Explore imputations in multiple variables and models

We can then plot the data as a boxplot, putting the imputation model on the x axis, value on the y axis, and facetting the different values for each variable.

```
ggplot(bound_models_gather,
       aes(x = imp_model,
           y = value)) +
  geom_boxplot() +
  facet_wrap(~variable)
```

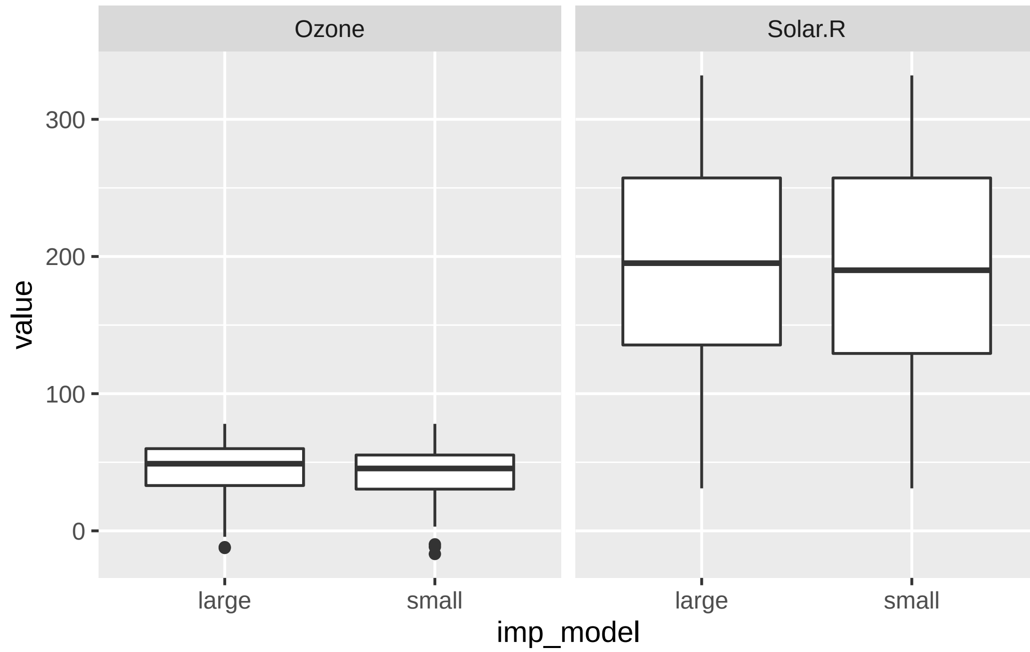



We learn that both models have similar mean and spread, and scale - there isn't much difference between the models.

## 15.6 Explore imputations in multiple variables and models

We can also only look at the imputed values only, by filtering `any_missing` to look at "Missing", and do the same plot.

```
bound_models_gather %>%
  filter(any_missing == "Missing") %>%
  ggplot(aes(x = imp_model,
             y = value)) +
  geom_boxplot() +
  facet_wrap(~variable)
```




We see that the imputed values for the larger model tend to have a slightly higher median than those imputed with the smaller model. But this is a very small difference.

# 16 Assessing inference from imputation

```
library(naniar)
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.1 --

v ggplot2 3.3.6     v purrr   0.3.4
v tibble  3.1.7     v dplyr   1.0.9
v tidyr   1.2.0     v stringr 1.4.0
v readr   2.1.2     v forcats 0.5.1

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

```
library(simputation)
```

```
Attaching package: 'simputation'

The following object is masked from 'package:naniar':

    impute_median
```

In this chapter we discuss methods for assessing model inference across differently imputed datasets. Let's step back, and think about why we are imputing data in the first place. Our goal in performing imputations is to perform an analysis in a way that the missing values do not unfairly bias subsequent inference, or predictions that we make.

## 16.1 Exploring parameters of one model

[TODO: update from airquality dataset, it is getting a bit tired]

Let's fit a model to the airquality dataset using a linear model, predicting temperature, using ozone, solar radiation, wind, month and day.

```
lm(Temp ~ Ozone + Solar.R + Wind + Month + Day, data = airquality)
```

```
Call:
lm(formula = Temp ~ Ozone + Solar.R + Wind + Month + Day, data = airquality)

Coefficients:
(Intercept)        Ozone      Solar.R         Wind        Month          Day
   57.25183      0.16528      0.01082     -0.17433      2.04246     -0.08919
```

We are going to fit this model using two methods:

1. Complete case analysis, where we remove all rows that contain a missing value
2. Imputing data using the linear model imputation from the last lesson.

Using the complete cases provides a nice baseline for comparison, as this removes all missing values, so it is sort of like comparing your model to "doing nothing". Except that it is worse than doing nothing - since you are removing data! You might be able to imagine a few different outcomes of this process:

- The outputs are basically the same, in which case, using the data with imputed values is better from a statistics standpoint, so you may as well use them.
- The imputed data does much better than complete cases, in which case, use the imputed data.
- The imputed data does **worse** than complete cases - which which case, you might want to check your imputed model for errors, or perhaps there are some bias in your data.

## 16.2 Combining the datasets together

There are three steps to comparing our data.

First, we perform the complete case analysis, using `na.omit()`, and converting the data into `nabular` form.

```r
#1.  Complete cases
aq_cc <-   airquality %>%
  na.omit() %>%
  nabular() %>%
  add_label_shadow()
```

Second, we impute the data according to a linear model

```r
#2. Imputation using the imputed data from the last lesson
aq_imp_lm <- airquality %>%
  nabular() %>%
  add_label_shadow() %>%
  as.data.frame() %>%
  impute_lm(Ozone ~ Temp + Wind + Month + Day) %>%
  impute_lm(Solar.R ~ Temp + Wind + Month + Day)  %>%
  as_tibble()
```

Finally, we combine the different datasets together with `bind_rows()`. Note the extra column, `imp_model`, which helps us identify data from the model used.

```r
# 3. Bind the models together
bound_models <- bind_rows(cc = aq_cc,
                          imp_lm = aq_imp_lm,
                          .id = "imp_model")
```

This prepares us for fitting our new models, so we can summarise and compare differences in the data.

The bound models have a column `imp_model`, then the columns from airquality, and our shadow variables and `any_missing`.

```r
head(bound_models)
```

```
# A tibble: 6 x 14
  imp_model Ozone Solar.R  Wind  Temp Month   Day Ozone_NA Solar.R_NA Wind_NA
  <chr>     <dbl>   <dbl> <dbl> <int> <int> <int> <fct>    <fct>      <fct>
1 cc           41     190   7.4    67     5     1 !NA      !NA        !NA
2 cc           36     118   8      72     5     2 !NA      !NA        !NA
3 cc           12     149  12.6    74     5     3 !NA      !NA        !NA
4 cc           18     313  11.5    62     5     4 !NA      !NA        !NA
5 cc           23     299   8.6    65     5     7 !NA      !NA        !NA
6 cc           19      99  13.8    59     5     8 !NA      !NA        !NA
```

```
# ... with 4 more variables: Temp_NA <fct>, Month_NA <fct>, Day_NA <fct>,
#   any_missing <chr>
```

```
tail(bound_models)
```

```
# A tibble: 6 x 14
  imp_model Ozone Solar.R  Wind  Temp Month   Day Ozone_NA Solar.R_NA Wind_NA
  <chr>     <dbl>   <dbl> <dbl> <int> <int> <int> <fct>    <fct>      <fct>
1 imp_lm     14        20  16.6    63     9    25 !NA      !NA        !NA
2 imp_lm     30       193   6.9    70     9    26 !NA      !NA        !NA
3 imp_lm     26.9     145  13.2    77     9    27 NA       !NA        !NA
4 imp_lm     14       191  14.3    75     9    28 !NA      !NA        !NA
5 imp_lm     18       131   8      76     9    29 !NA      !NA        !NA
6 imp_lm     20       223  11.5    68     9    30 !NA      !NA        !NA
# ... with 4 more variables: Temp_NA <fct>, Month_NA <fct>, Day_NA <fct>,
#   any_missing <chr>
```

## 16.3 Exploring the models

Now that we've got our data in the right format, we fit a linear model to each of the datasets. We use the "many models" approach, covered in detail in the R for data science book by Hadley Wickham and Garrett Grolemund.

This involves some functions that we haven't seen before. Let's unpack what's happening below. First we group by the imputation model, then nest the data. This collapses, or nests, the data down into a neat format where each row is one of our datasets.

```
bound_models %>%
  group_by(imp_model) %>%
  nest()
```

```
# A tibble: 2 x 2
# Groups:   imp_model [2]
  imp_model data
  <chr>     <list>
1 cc        <tibble [111 x 13]>
2 imp_lm    <tibble [153 x 13]>
```

This allows us to create linear models on each row of the data, using `mutate`, and a special function, `map`. This tells the function we are applying to look at the data and then fit the linear model to each of the datasets in the `data` column.

```
bound_models %>%
  group_by(imp_model) %>%
  nest() %>%
  mutate(mod = map(data,
                   ~lm(Temp ~ Ozone + Solar.R + Wind + Day + Month,
                       data = .)))
```

```
# A tibble: 2 x 3
# Groups:   imp_model [2]
  imp_model data               mod
  <chr>     <list>             <list>
1 cc        <tibble [111 x 13]> <lm>
2 imp_lm    <tibble [153 x 13]> <lm>
```

Then we then fit the model and create separate columns for residuals, predictions, and coefficients, using the `tidy` function from `broom`, to provide nicely formatted coefficients from our linear model.

```
model_summary <- bound_models %>%
  group_by(imp_model) %>%
  nest() %>%
  mutate(mod = map(data,
                   ~lm(Temp ~ Ozone + Solar.R + Wind + Day + Month,
                       data = .)),
         res = map(mod, residuals),
         pred = map(mod, predict),
         tidy = map(mod, broom::tidy))

model_summary
```

```
# A tibble: 2 x 6
# Groups:   imp_model [2]
  imp_model data               mod    res         pred        tidy
  <chr>     <list>             <list> <list>      <list>      <list>
1 cc        <tibble [111 x 13]> <lm>   <dbl [111]> <dbl [111]> <tibble [6 x 5]>
2 imp_lm    <tibble [153 x 13]> <lm>   <dbl [153]> <dbl [153]> <tibble [6 x 5]>
```

Our data, `model_summary`, has the columns `imp_model`, and `data`, and columns with our fitted linear model (mod), residuals (res), predictions (pred), and tidy coefficients (tidy).

`model_summary` forms the building block for the next steps in our analysis, where we are going to look at the coefficients, the residuals, and the predictions.

This is just one way to fit these kinds of models - there are many other ways, and it might not work for all types of models, but this many models approach can be convenient!

## 16.4 Exploring coefficients of multiple models

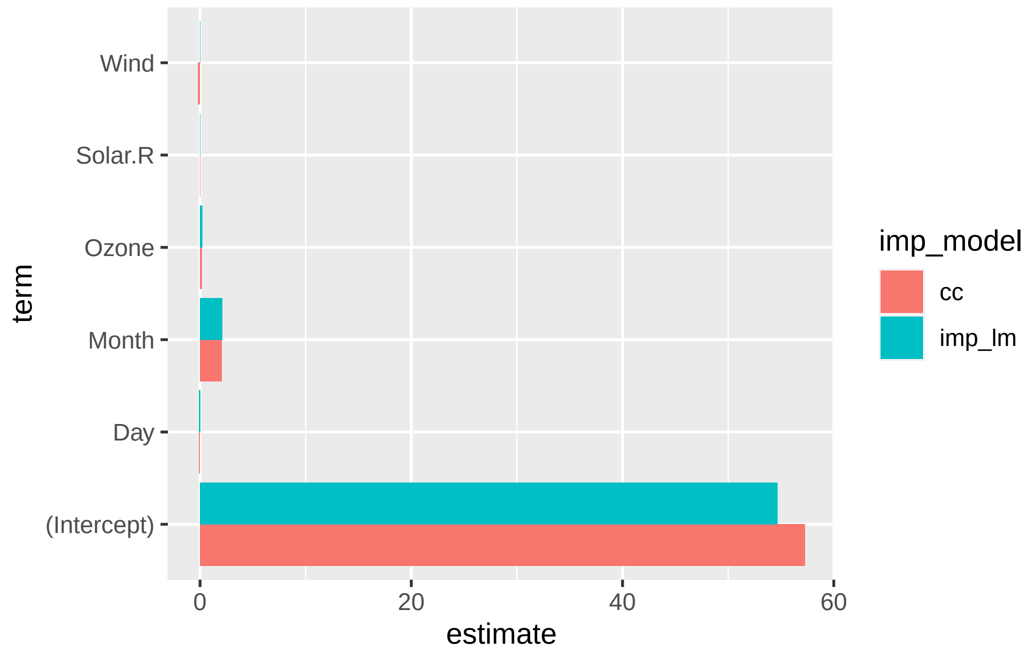
We explore coefficients by selecting the imputation model and the tidy column and unnesting:

```
model_summary_coefs <- model_summary %>%
  select(imp_model,
         tidy) %>%
  unnest(cols = c(tidy))

model_summary_coefs
```

```
# A tibble: 12 x 6
# Groups:   imp_model [2]
   imp_model term        estimate std.error statistic  p.value
   <chr>     <chr>          <dbl>     <dbl>     <dbl>    <dbl>
 1 cc        (Intercept) 57.3       4.50      12.7    5.52e-23
 2 cc        Ozone        0.165     0.0239     6.92   3.66e-10
 3 cc        Solar.R      0.0108    0.00699    1.55   1.24e- 1
 4 cc        Wind        -0.174     0.212     -0.821  4.13e- 1
 5 cc        Day         -0.0892    0.0677    -1.32   1.91e- 1
 6 cc        Month        2.04      0.409      4.99   2.42e- 6
 7 imp_lm    (Intercept) 54.7       3.59      15.2    5.21e-32
 8 imp_lm    Ozone        0.196     0.0205     9.53   4.52e-17
 9 imp_lm    Solar.R      0.0102    0.00577    1.76   7.97e- 2
10 imp_lm    Wind        -0.00642   0.172     -0.0374 9.70e- 1
11 imp_lm    Day         -0.112     0.0538    -2.08   3.92e- 2
12 imp_lm    Month        2.11      0.340      6.21   5.09e- 9
```

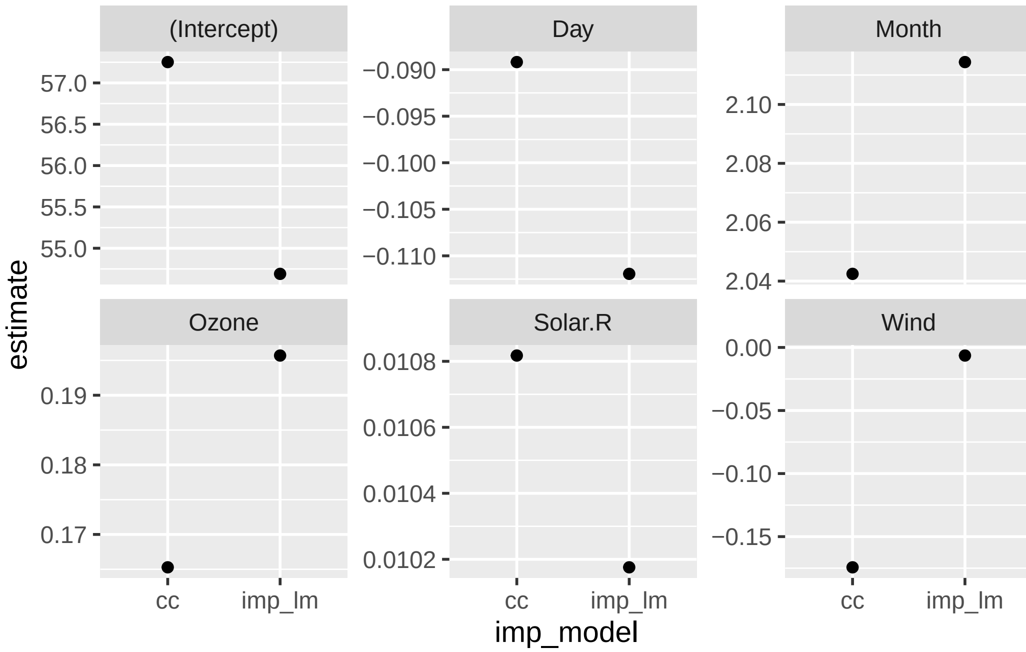
We now see the estimates of the impact of Ozone on temperature. To best understand this we can plot these estimates for each model like so:

```
ggplot(model_summary_coefs,
       aes(x = estimate,
           y = term,
           fill = imp_model)) +
  geom_col(position = "dodge")
```




Plotting these, we see that the estimates are pretty much the same for both, with the intercept being slightly lower for the imputed model, and higher for the complete cases. However, we can probably get a slightly more nuanced view of this by looking at these variables on their own scale:

```
ggplot(model_summary_coefs,
       aes(x = imp_model,
           y = estimate)) +
  geom_point() +
  facet_wrap(~term, scales = "free_y")
```



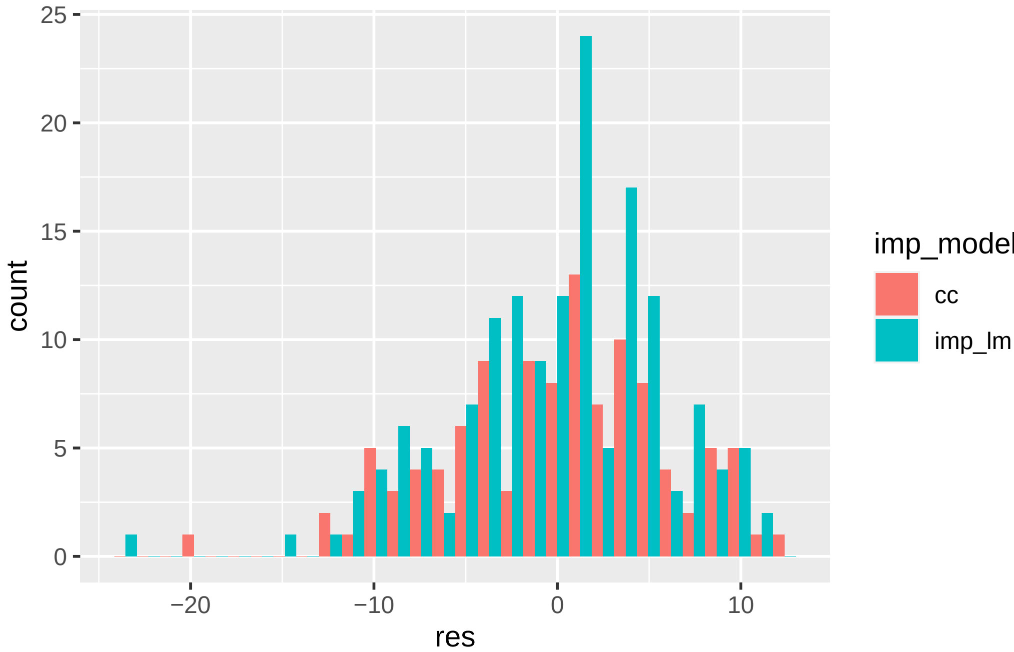

These look like big differences for all of these - but this is intentional, as we have let the y axis be freely varying. These values in this case look to be not very different in a meaningful way for this data, but it is an important step to take for any dataset.

## 16.5 Exploring residuals of multiple models

Let's explore the residuals by selecting the imp model and residuals, and then unnesting the data. We can then create a histogram, using `position = "dodge"` to put residuals for each model next to each other.

```
model_summary %>%
  select(imp_model,
         res) %>%
  unnest(cols = c(res)) %>%
  ggplot(aes(x = res,
             fill = imp_model)) +
  geom_histogram(position = "dodge")
```

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



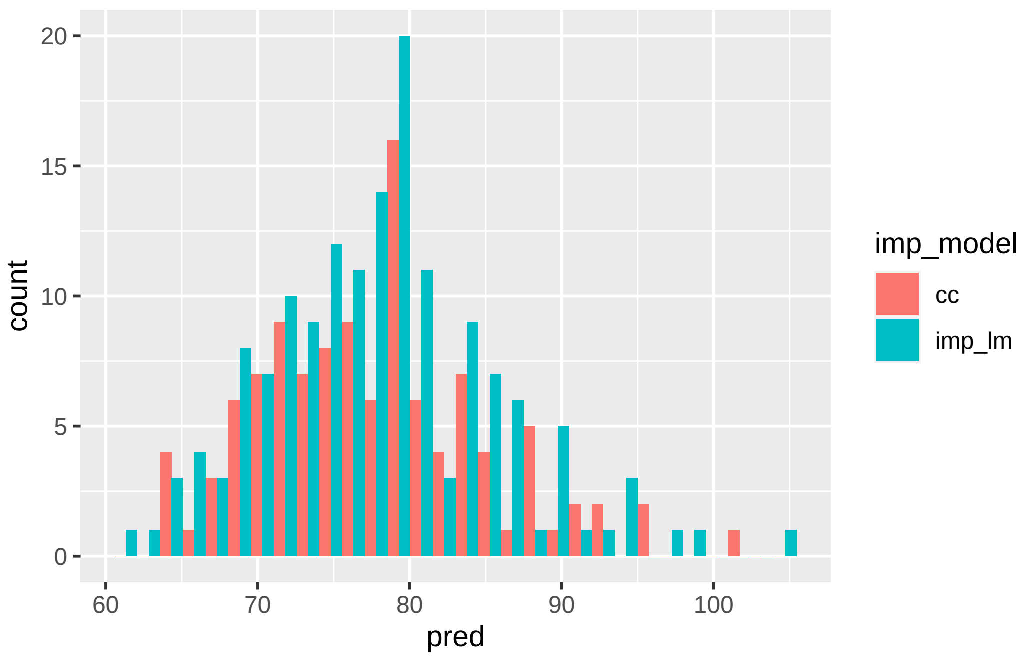

We see that, surprisingly! there isn't much difference between the two, and the residuals of the imputed model seem to be more centered around zero.

## 16.6 Exploring predictions of multiple models

Finally, we can explore the predictions in the data, using a similar pattern.

```
model_summary %>%
  select(imp_model,
         pred) %>%
  unnest(cols = c(pred)) %>%
  ggplot(aes(x = pred,
             fill = imp_model)) +
  geom_histogram(position = "dodge")
```

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




Similar to what we saw with the residuals, the predictions are quite similar to complete case, but with some more extreme values.

# Part VII

# Conclusion

# 17 End

```r
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.1 --

v ggplot2 3.3.6     v purrr   0.3.4
v tibble  3.1.7     v dplyr   1.0.9
v tidyr   1.2.0     v stringr 1.4.0
v readr   2.1.2     v forcats 0.5.1

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

```r
library(naniar)
```

### 17.0.1 This is only the beginning!

Now as they say, this is only the beginning. This course covered an often overlooked area of statistics - missing data, and inside the world of missing data, we also covered yet another area that is often overlooked: How to handle, explore, and visualise missing values.

To continue your journey, and learn more about missing data, you should check out the `naniar` package, which contains many useful functions to explore and evaluate your missing data, as well as numerous vignettes.

The `visdat` package provides more than just heatmaps of missing data, and is well worth looking into to learn more about pre exploratory visualisation:

From here, to continue your journey, you might want to explore other workflows for imputing your missing data.

There are many ways to decide how to impute data. We didn't have time for it in the course, but multiple imputation is another great area of research - to learn more about multiple imputation, I highly recommend Stefan van Buuren's package, `mice`, and his book, Flexible Imputation of Missing Data.

naniar.njtierney.com visdat.njtierney.com mice R package flexible imputation of missing data

# References

# A glossary

- impute, imputation
- nabular
- MCAR
- MAR
- MNAR

# Bibliography