Dealing with Missing Values

Dr Ebin Deni Raj

12/11/2022

Contents

Introduction

This document will help you to deal with missing data in a dataset. Missing data is part of any real world data analysis. We have seen some missing data while dealing with the cleaning Data using tidyr.Missing values can crop up in unexpected places, making analyses challenging to understand. In this document,you will revise the use of tidyverse tools and you will be introduced to a new package called naniar R package to visualize missing values. You'll tidy missing values so they can be used in analysis and explore missing values to find bias in the data. Lastly, you'll reveal other underlying patterns of missingness. You will also learn how to "fill in the blanks" of missing values with imputation models, and how to visualize, assess, and make decisions based on these imputed datasets.

Prerequisite: Basics of R + Tidyverse packages+ Statistical modelling in R Note: Please complete the exercises in the tidyverse packages and statistical modelling using R documents.

Why do we care about missing data?

The best thing to do with missing data is not to have any. But unfortunately, Working with real world data means working with missing data. This document will give you an idea to work with missing values. Understanding how missing data works 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 are not based on the right evidence. Replacing missing values are known as imputations. This should be very carefully and will depend on the type of data you are dealing with. Inserting mean value, in the place of missing value can lead to poor estimate and decision at times.

We will learn the following in this document:

- What missing values are
- How to find missing data
- How to wrangle and tidy missing data
- Explore why is data missing

- Impute missing values

What is meant by missing value?
Missing values are values that should have been recorded but were not. NA= **Not Available** How to check missing values?

Missing values are usually like checking needle in a haystack. But there are packages which can help in making this task easier. For this tutorial install *naniar* package.

```
install.packages("naniar")

install.packages("visdat")
```

Some examples on missing values:

When working with missing data, there are a couple of commands that you should be familiar with - firstly, you should be able to identify if there are any missing values, and where these are.

Using the any_na() and are_na() tools, identify which values are missing.

```
library(naniar)
x <- c(1, NA, 3, NA, NA, 5)
any_na(x)
```

```
## [1] TRUE
```

```
are_na(x)
```

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

```
n_miss(x)
```

```
## [1] 3
```

```
prop_miss(x)
```

```
## [1] 0.5
```

```
any_na(NaN)
```

```
## [1] TRUE
```

```
any_na(NULL)
```

```
## [1] FALSE
```

```
any_na(Inf)
```

```
## [1] FALSE
```

```
#What happens if we mix missing values with our calculations and comparisons
NA | TRUE
```

```
## [1] TRUE
```

```
NA | FALSE
```

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

```
NA + NaN
```

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

```
NaN + NA
```

```
## [1] NaN
```

```r
# Create x, a vector, with values NA, NaN, Inf, ".", and "missing"
x <- c(NA, NaN, Inf, ".", "missing")

# Use any_na() and are_na() on to explore the missings
any_na(x)
```

```
## [1] TRUE
```

```r
are_na(x)
```

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

How many Missing values are there?

One of the first things that you will want to check with a new dataset is if there are any missing missing values, and how many there are.

You could use *are_na()* to and count up the missing values, but the most efficient way to count missings is to use the *n_miss()* function. This will tell you the total number of missing values in the data.

You can then find the percent of missing values in the data with the pct_miss function. This will tell you the percentage of missing values in the data.

You can also find the complement to these - how many complete values there are - using *n_complete* and *pct_complete.*

Now that you understand the behavior of missing values in R, and how to count them, let's scale up to more detailed summaries of missingness. We need to summarise missing data to identify variables, cases or patterns of missingness, as these can bias our data analysis. There are two main summaries: basic and dataframe summaries. Basic summaries return a single number, like the number of missing or complete values using 'n' miss or 'n complete'. However you will need more detailed missingness summaries to help you on your journey through data analysis.

The nainiar package contains a family of functions all starting with *miss_*, which each of them provide different summaries of missingness, and return a dataframe. This allows us to see features that can be difficult to articulate or time consuming to calculate. For example *miss_var_summary* and *miss_case_summary* return the number and percentage of missings in each variable or case. These summaries work well with *dplyr* s *group_by*, so you can fluidly explore missingness by each groups.

We can use *miss_var_summary* to summarise the number of missings in each variable. This returns a dataframe where each row is a variable. It also includes the summaries of the number and percentage of missings for each variable in the dataset, and is sorted by the number of missings. The *miss_case_summary* returns a summary dataframe, where each case represents a dataset row number. Tabulation of missingness counts the number of times there are 0,1,2,3 and so on missings. They are very useful compact summaries that reveal interesting structure. *miss_var_table* returns a dataframe with number of missings in a variable, and the number and percentage of variables affected. *miss_case_table* returns the same information but for cases.

We can also look at the missingness over a span or run for a given variable using *miss_var_span*and *miss_var_run.* These can be really useful for data with many regular measurements , like time series data. *miss_var_span* calculates the number of missings in a variable for a repeating span. This is useful in time series data to look for weekly 7 day patterns of missingness. *miss_var_span* returns a dataframe with columns "span_counter" which identifies the span- the first , second and so on, and also includes the number and proportion of missing and complete values. *miss_var_run* returns the "runs" or"streaks" of missingness. This is useful to try and find unusual patterns of missingnesss. It returns the length of run of "complete" and "missing" data. This is particularly useful in finding repeating patterns of missingness.

Summarizing missingness

Sometimes you are interested in missingness for groups of data. Each missingness summary function can be calculated by group , using *by_group* from *dplyr*. Here we will be using *miss_var_summary()* and *miss_case_summary()*, and also explore how they can be applied for groups in a dataframe, using the *group_by* function from *dplyr*. We will do the following tasks:

Calculate summaries of missingness in the airquality dataset for variables using the *miss_var_summary()* function.

Calculate summaries of missingness in the airquality dataset for the cases using the *miss_case_summary()* function.

Using the airquality dataset, use *group_by()* to create summaries for each variable and case, by each Month.

```
# Summarise missingness in each variable of the `airquality` dataset
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
```

```
# Summarise missingness in each case of the `airquality` dataset
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
## # i Use `print(n = ...)` to see more rows
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Return the summary of missingness in each variable, grouped 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
## # i Use `print(n = ...)` to see more rows
```

```
# Return the summary of missingness in each case, grouped by Month, in the `airquality` dataset
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
## # i Use `print(n = ...)` to see more rows
```

Now you have learned to summarise missingness for variables and cases for each groups!

Tabulating Missingness

The summaries of missingness we just calculated give us the number and percentage of missing observations for the cases and variables.

Another way to summarise missingness is by tabulating the number of times that there are 0, 1, 2, 3, missings in a variable, or in a case. For the *airquality* dataset: Tabulate missingness for each variable using *miss_var_table()*. Tabulate missingness for every case using *miss_case_table()*. Combine previous tabulations with dplyr's *group_by()* function to create tabulations for each variable and case, by each Month.

```
# Tabulate missingness in each variable and case of the `airquality` dataset
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
```

```
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
```

```
# Tabulate the missingness in each variable, grouped by Month, in the `airquality` dataset
airquality %>% group_by(Month) %>% miss_var_table()
```

```
## # A tibble: 12 x 4
## # Groups:   Month [5]
##    Month n_miss_in_var n_vars pct_vars
##    <int>         <int>  <int>    <dbl>
##  1     5             0      3       60
##  2     5             4      1       20
##  3     5             5      1       20
##  4     6             0      4       80
##  5     6            21      1       20
##  6     7             0      4       80
##  7     7             5      1       20
##  8     8             0      3       60
##  9     8             3      1       20
## 10     8             5      1       20
## 11     9             0      4       80
## 12     9             1      1       20
```

```
# Tabulate of missingness in each case, grouped by Month, in the `airquality` dataset
airquality %>% group_by(Month) %>% miss_case_table()
```

```
## # A tibble: 11 x 4
## # Groups:   Month [5]
##    Month n_miss_in_case n_cases pct_cases
##    <int>          <int>   <int>     <dbl>
##  1     5              0      24     77.4
##  2     5              1       5     16.1
##  3     5              2       2      6.45
##  4     6              0       9     30
##  5     6              1      21     70
##  6     7              0      26     83.9
##  7     7              1       5     16.1
##  8     8              0      23     74.2
##  9     8              1       8     25.8
## 10     9              0      29     96.7
## 11     9              1       1      3.33
```

Now lets do this for *pedestrian* dataset

Other summaries of missingness

Some summaries of missingness are particularly useful for different types of data. For example, *miss_var_span()* and *miss_var_run()*.

miss_var_span() calculates the number of missing values in a specified variable for a repeating span. This is really useful in time series data, to look for weekly (7 day) patterns of missingness

miss_var_run() calculates the number of "runs" or "streaks" of missingness. This is useful to find unusual patterns of missingness, for example, you might find a repeating pattern of 5 complete and 5 missings.

Both *miss_var_span()* and *miss_var_run()* work with the group_by operator from dplyr.

Using the *pedestrian* dataset from naniar:

Calculate summaries of missingness for the variables in datasets using *miss_var_span()*, for a *span* of 4000. Calculate summaries of missingness for the cases in datasets using *miss_var_run()*. Combine with dplyr's group_by operator for month.

```
# Calculate the summaries for each run of missingness for the variable, hourly_counts
miss_var_run(pedestrian, var = 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
## # i Use `print(n = ...)` to see more rows
```

```
# Calculate the summaries for each span of missingness, for a span of 4000, for the variable hourly_cou
miss_var_span(pedestrian, var = hourly_counts, span_every = 4000)
```

```
## # A tibble: 10 x 6
##    span_counter n_miss n_complete prop_miss prop_complete n_in_span
##           <int>  <int>      <int>     <dbl>         <dbl>     <int>
##  1            1      0       4000   0                 1          4000
##  2            2      1       3999   0.00025           1.00       4000
##  3            3    121       3879   0.0302            0.970      4000
##  4            4    503       3497   0.126             0.874      4000
##  5            5    745       3255   0.186             0.814      4000
##  6            6      0       4000   0                 1          4000
##  7            7      1       3999   0.00025           1.00       4000
##  8            8      0       4000   0                 1          4000
##  9            9    745       3255   0.186             0.814      4000
## 10           10    432       1268   0.254             0.746      1700
```

```
# For each `month` variable, calculate the run of missingness for hourly_counts
pedestrian %>% group_by(month) %>% miss_var_run(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
## # i Use `print(n = ...)` to see more rows
```

```
# For each `month` variable, calculate the span of missingness of a span of 2000, for the variable hour
pedestrian %>% group_by(month) %>% miss_var_span(var = hourly_counts, span_every =2000)
```

```
## # A tibble: 25 x 7
## # Groups:   month [12]
##    month    span_counter n_miss n_complete prop_miss prop_complete n_in_span
##    <ord>           <int>  <int>      <int>     <dbl>         <dbl>     <int>
##  1 January             1      0       2000     0             1          2000
##  2 January             2      0        976     0             1           976
##  3 February            1      0       2000     0             1          2000
##  4 February            2      0        784     0             1           784
##  5 March               1      0       2000     0             1          2000
##  6 March               2      0        976     0             1           976
##  7 April               1    552       1448     0.276         0.724      2000
##  8 April               2      0        880     0             1           880
##  9 May                 1     72       1928     0.036         0.964      2000
## 10 May                 2      0        976     0             1           976
## # ... with 15 more rows
## # i Use `print(n = ...)` to see more rows
```

You have now learnt how to summarise missingness over repeating spans, and find runs of missingness!

Missing data visualizations - Introduction

Lets take a look at some builtin visualizations that comes with *naniar*. Data summaries are very useful, but sometimes an idea or thought can be quickly captured with a visualisation. Each visulaisation is a nice compact shorthand for the data summaries. When you first get a dataset it can be difficult to get a visual sense of where the missings are. To get an overview of the amount of missingness we will use the *vis_miss* function in *visdat* package. *vis_miss* produces a "heatmap" of the missingness - like as if the plot correspond to the dataset as a giant spread sheet, with values coloured black for missing and grey for present. *vis_miss* also provides missingness summary statistics, showing the overall percentage of missingness in the legend, and the amount of missingness in each variable. *vis_miss* also allows clustering of missing data by setting *cluster=TRUE* : this orders the rows by missingnesss to identify common co-occurences. To quickly show the missingness in variables and cases, we visualise them using *gg_miss_var* and *gg_miss_case*. These are visual analogues of the *miss_var_summary* and *miss_case_summary* functions.

To visualize the common combinations of missingness - which variables and cases go missing together , we use *gg_miss_upset*. This powerful visualization shows the number of combinations of missing values that co-occur. To explore how missigness in each variable changes across a factor, use *gg_miss_fct*. This displays a heatmap visualisation showing the factors on the x-axis, and other variables on the y-axis , and the amount of missingness coloured from dark purple to yellow. *gg_miss_fct* does not support facetting.

gg_miss_span is the visual analogue of *miss_var_span*. This calculates the number of missings in a given span , say the number of missings for every 3000 rows. It displays the amount of missing values in each span in a filled barplot. *gg_miss_span* supports facetting.

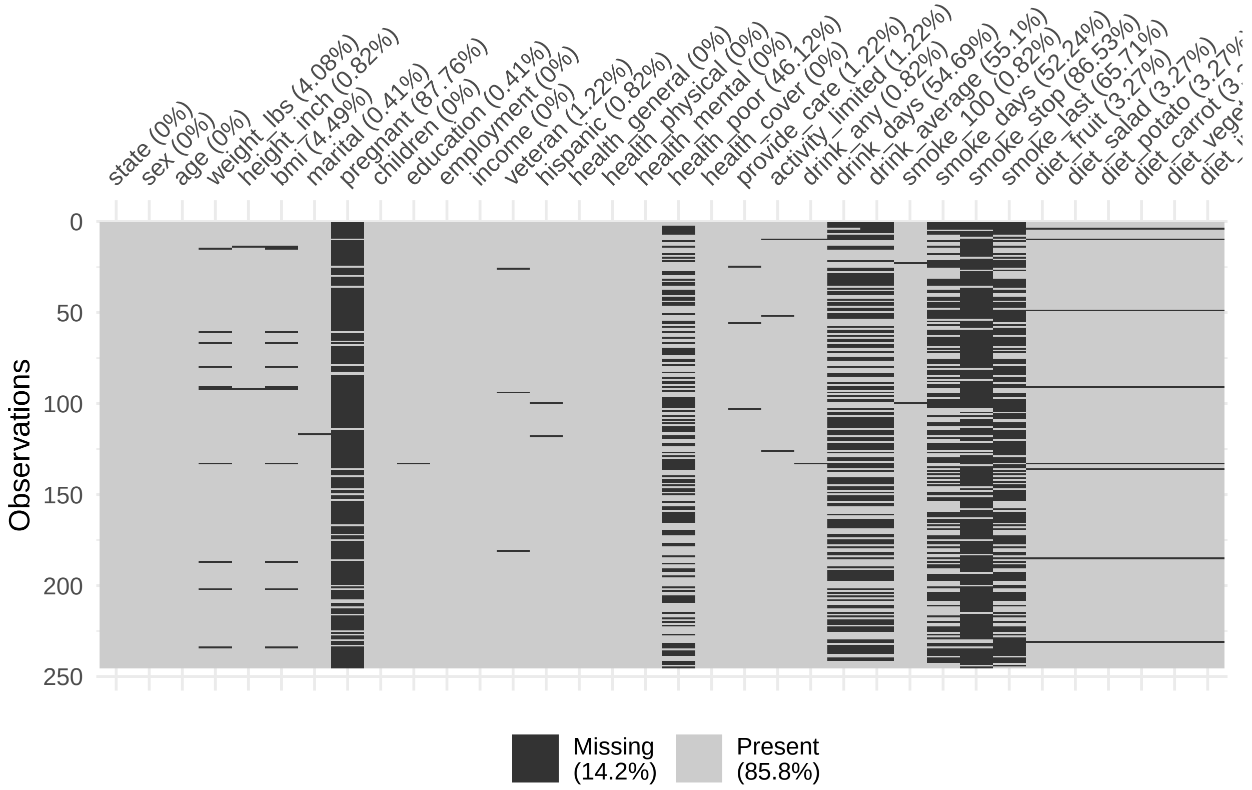
The function *vis_miss()* creates an overview visualization of the missingness in the data. It also has options to cluster rows based on missingness, using cluster = TRUE; as well as options for sorting the columns, from most missing to least missing (sort_miss = TRUE)

Using the *riskfactors* dataset from *naniar*:

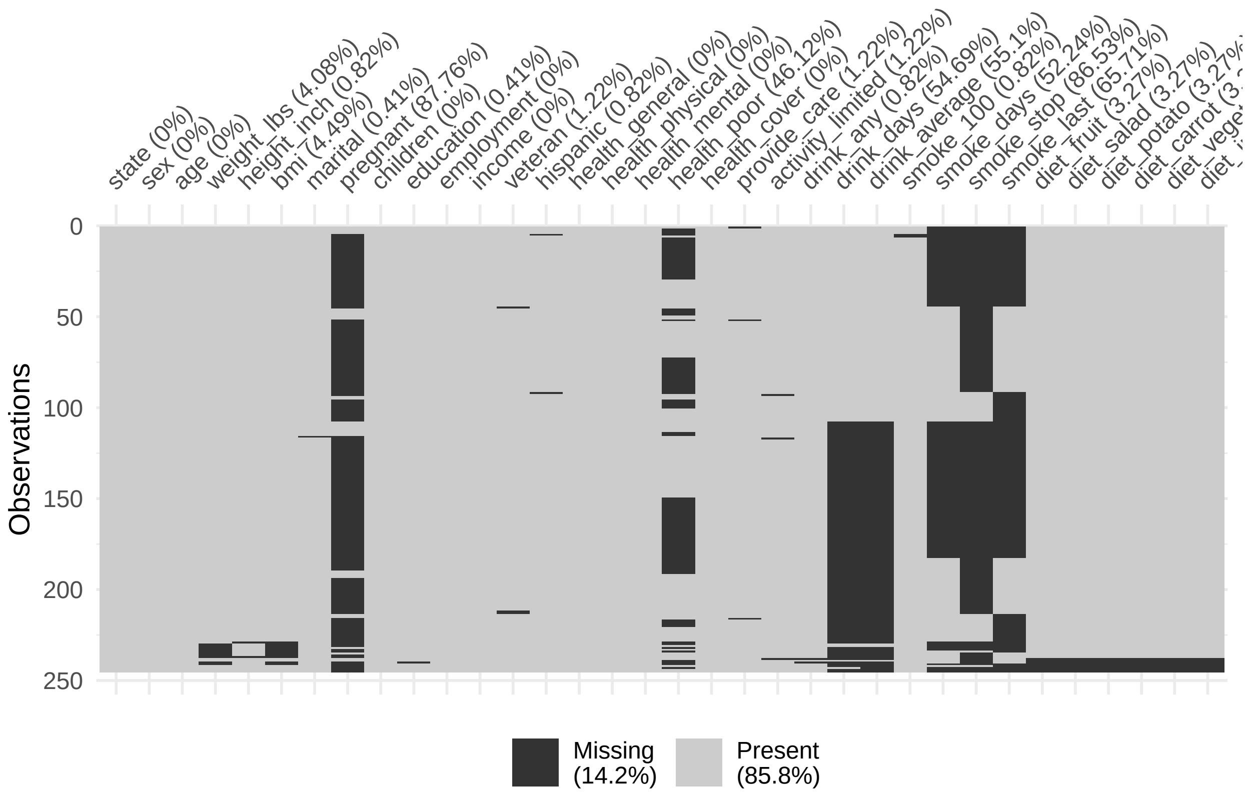
Use *vis_miss()* to visualise the missingness in the data.
Use *vis_miss()* with cluster = TRUE to explore some clusters of missingness.
Use *vis_miss()* and sort the missings with sort_miss to arrange the columns by missingness.

```
# Visualize all of the missingness in the `riskfactors`  dataset
vis_miss(riskfactors)
```

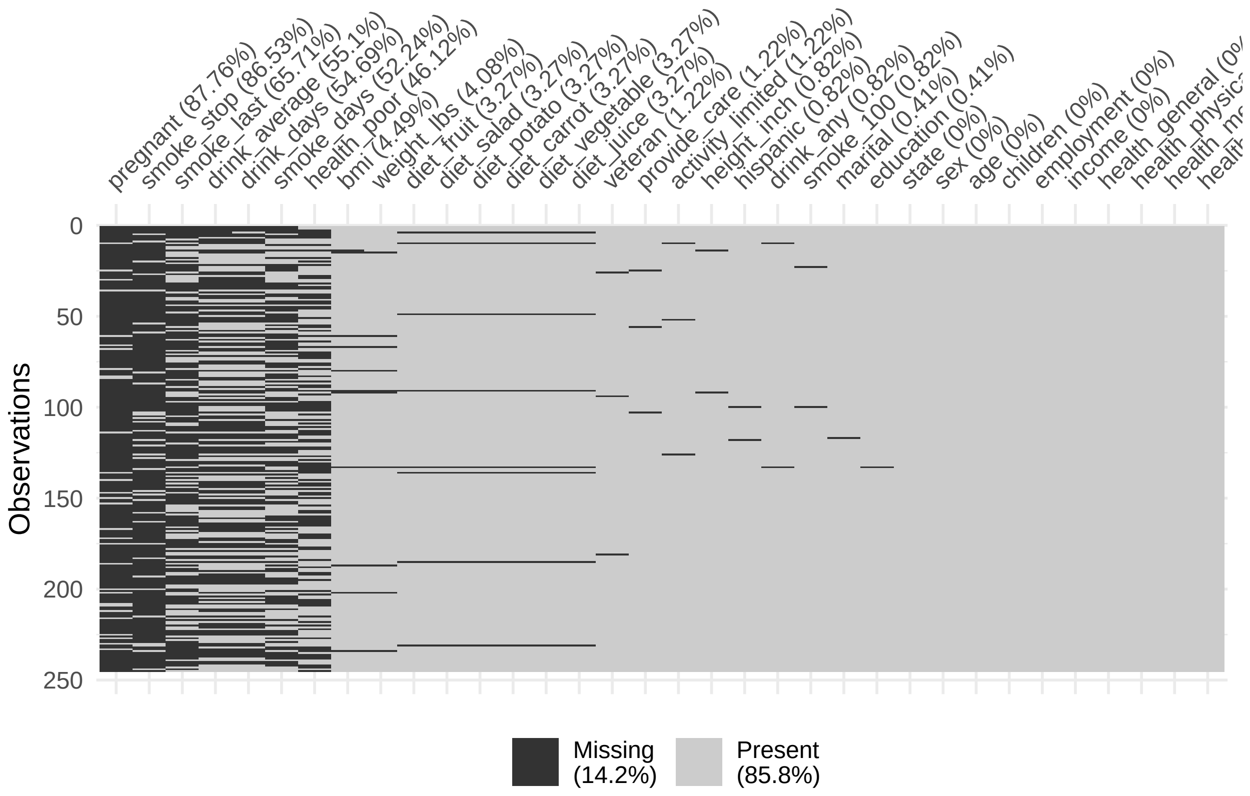
```
## Warning: `gather_()` was deprecated in tidyr 1.2.0.
## Please use `gather()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```



```
# Visualize and cluster all of the missingness in the `riskfactors` dataset
vis_miss(riskfactors, cluster = TRUE)
```



```r
# visualise and sort the columns by missingness in the `riskfactors` dataset
vis_miss(riskfactors, sort = TRUE)
```



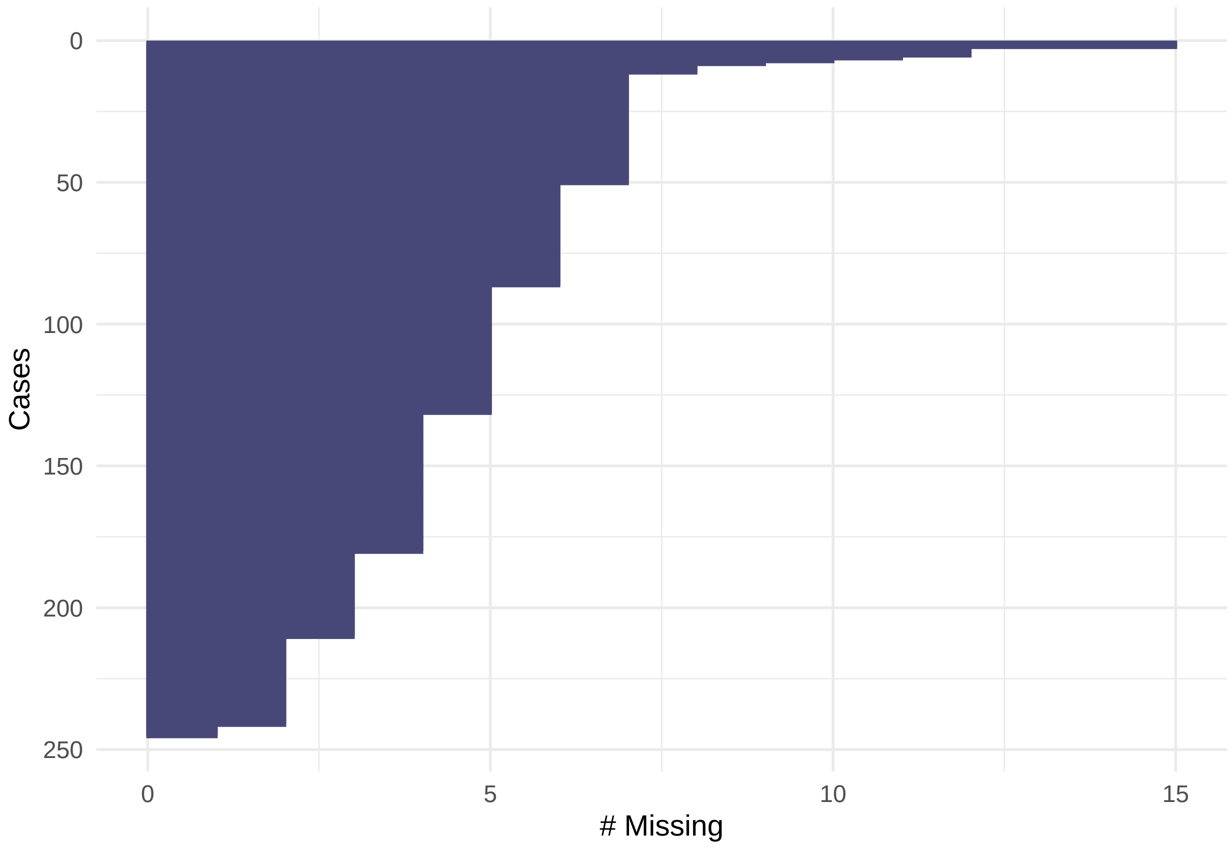
To get a clear picture of the missingness across variables and cases, use *gg_miss_var()* and *gg_miss_case()*.

These are the visual counterpart to *miss_var_summary()* and *miss_case_summary()*.

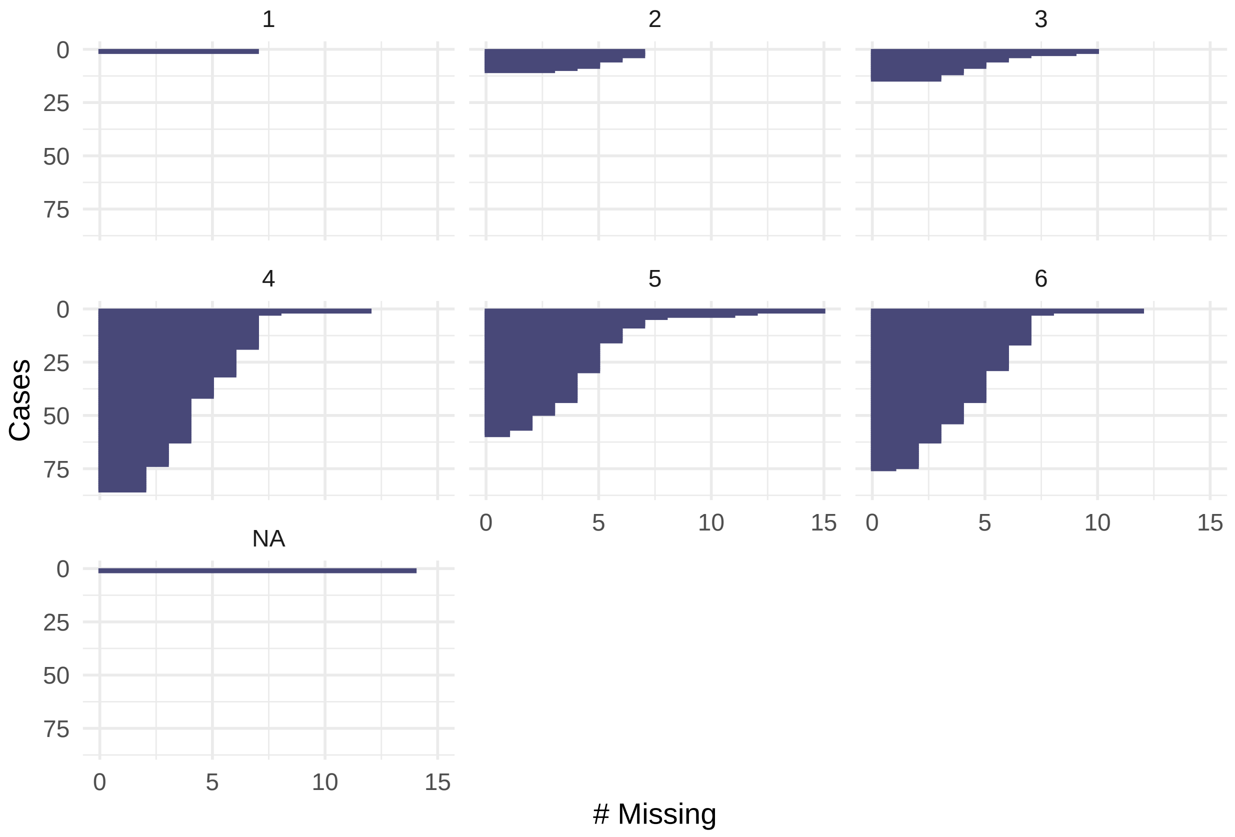
These can be split up into multiple plots with one for each category by choosing a variable to facet by. Using the *riskfactors* dataset:

Visualize the number of missings in cases using *gg_miss_case()*. Explore the number of missings in cases using *gg_miss_case()* and facet by the variable education. Visualize the number of missings in variables using *gg_miss_var()*. Explore the number of missings in variables using *gg_miss_var()* and facet by the variable education.

```
# Visualize the number of missings in cases using `gg_miss_case()`
gg_miss_case(riskfactors)
```

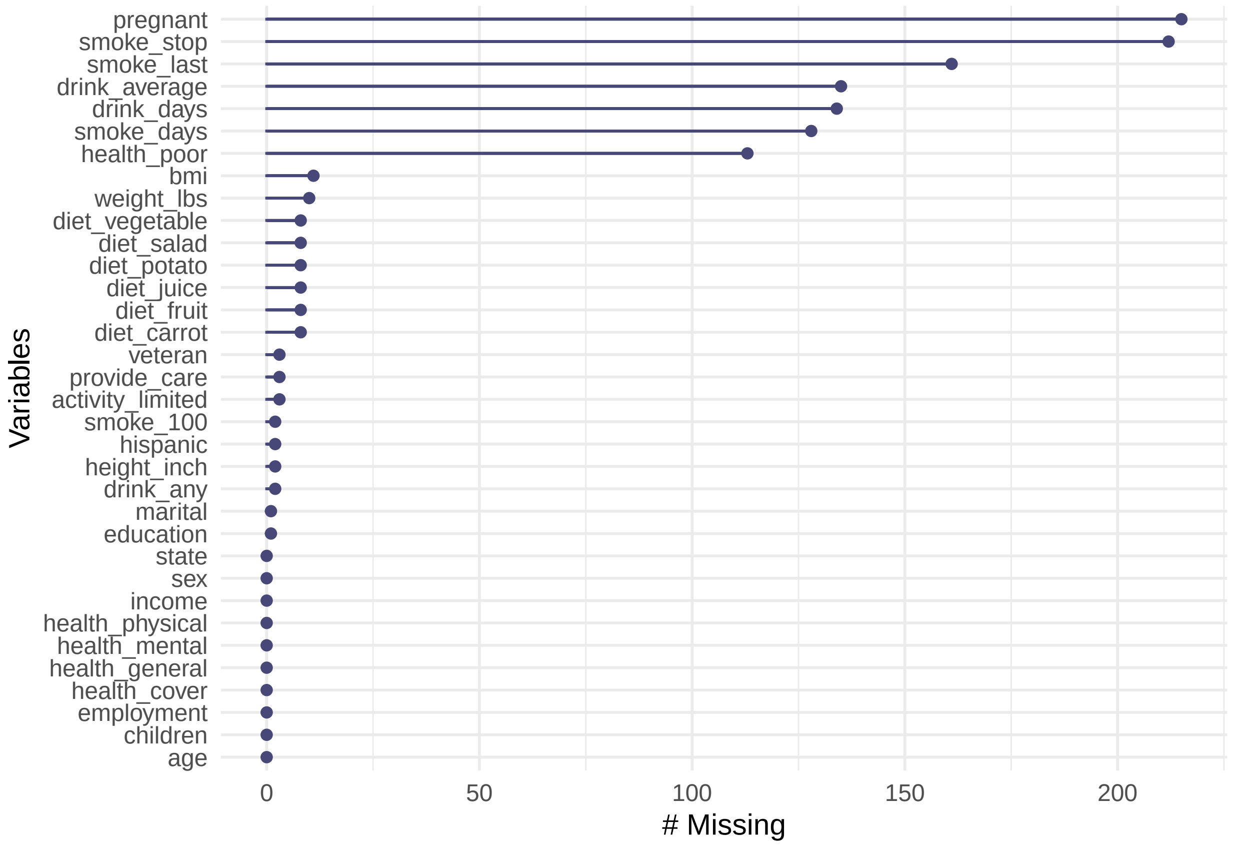


```
# Explore the number of missings in cases using `gg_miss_case()` and facet by the variable `education`
gg_miss_case(riskfactors, facet = education)
```



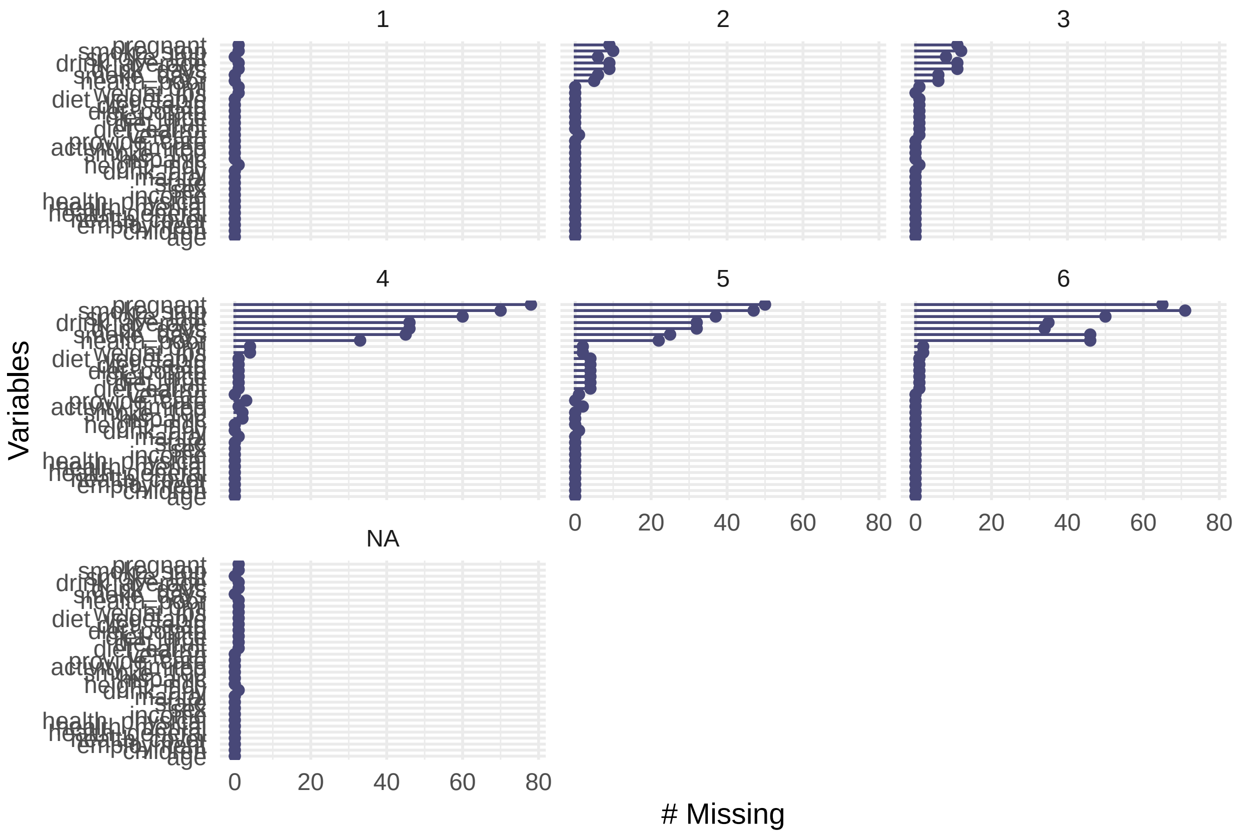
```r
# Visualize the number of missings in variables using `gg_miss_var()`
gg_miss_var(riskfactors)
```

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



```r
# Explore the number of missings in variables using `gg_miss_var()` and facet by the variable `educatio
gg_miss_var(riskfactors, facet = education)
```

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



You have learnt to visualise missingness in a dataset for variables and cases, and explore how other variables interact with missingness.

Visualising missingness patterns

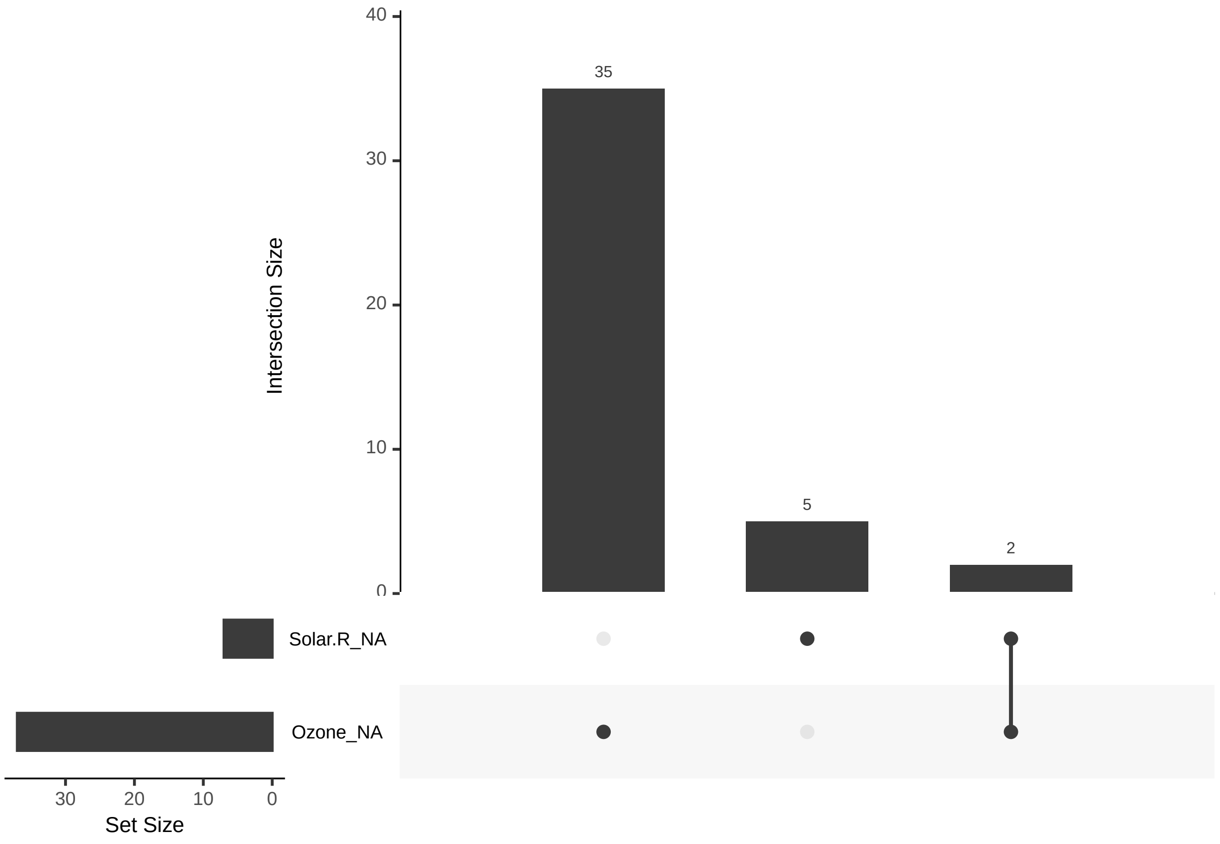
Let's practice a few different ways to vizualise patterns of missingness using:

gg_miss_upset() to give an overall pattern of missingness. *gg_miss_fct()* for a dataset that has a factor of interest: marriage. and *gg_miss_span()* to explore the missingness in a time series dataset.

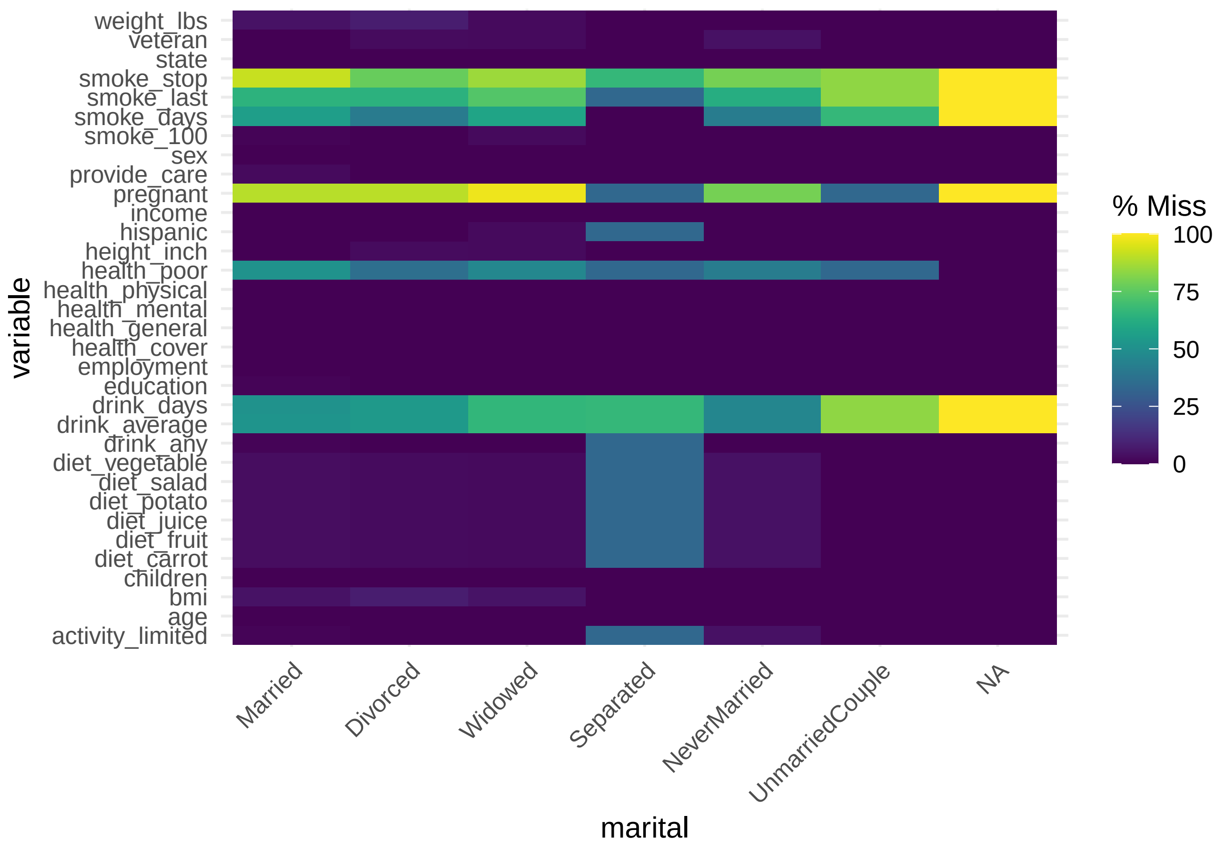
Lets try the following:

Explore missingness pattern of the *airquality* dataset with *gg_miss_upset()*.
Explore how the missingness changes in the *riskfactors* dataset across the marital variable using *gg_miss_fct()*
Explore how the missingness changes in the *pedestrian* dataset across the *hourly_counts* variable over a *span* of 3000 (you can also try different spans from 2000-5000).
Explore the impact of *month* on *hourly_counts* by including it in the *facet* argument, with a *span* of 1000.

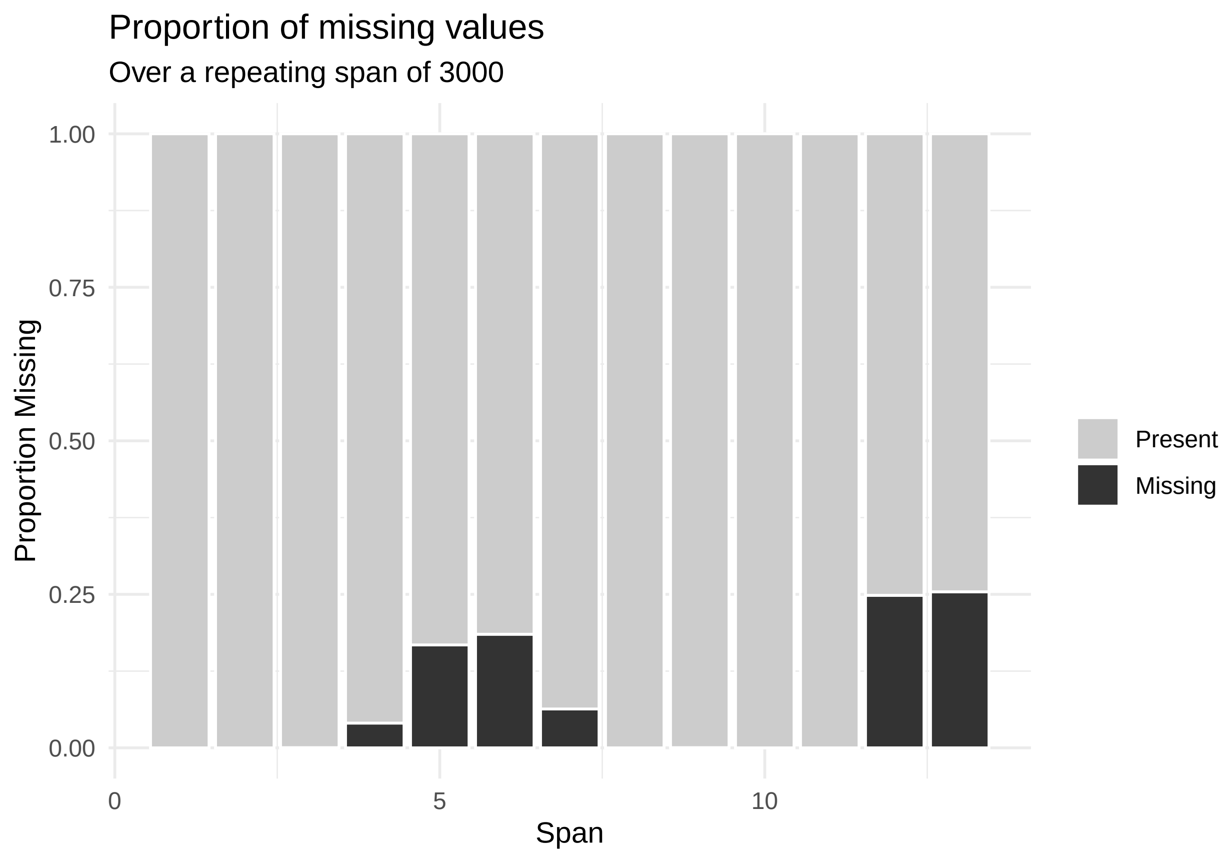
```
# Using the airquality dataset, explore the missingness pattern using gg_miss_upset()
gg_miss_upset(airquality)
```



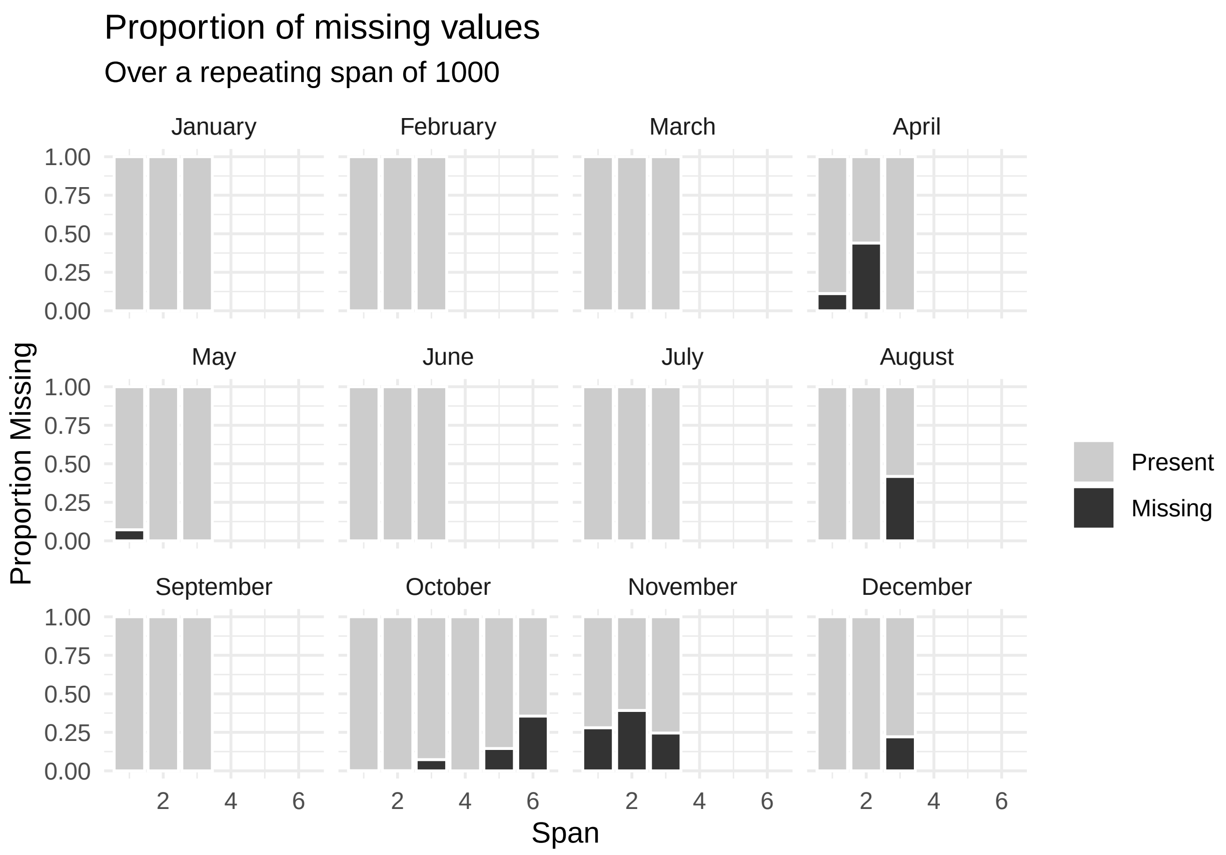
```r
# With the riskfactors dataset, explore how the missingness changes across the marital variable using g
gg_miss_fct(x = riskfactors, fct = marital)
```



```
# Using the pedestrian dataset, explore how the missingness of hourly_counts changes over a span of 300
gg_miss_span(pedestrian, var = hourly_counts, span_every = 3000)
```



```
# Using the pedestrian dataset, explore the impact of month by facetting by month
# and explore how missingness changes for a span of 1000
gg_miss_span(pedestrian, var = hourly_counts , span_every = 1000, facet = month)
```



We have leart to visualise different missingness patterns for different kinds of data.

Wrangling and tidying up missing values

We have seen multiple methods of visualizing missing values !!! But that time, you have assumed that the missing values will be always coded as "NA". But this is often not the case with real world datasets. This section will throw some light on the assumptions in missing data and how to look for missing data. If you have some data where missing values are not labelled as "NA' but as"missing",''Not Available" ,"N/A". In this section, we will see how to identify missing values and replace them with "NA" values, once found.

Using miss_scan_count

You have a dataset with missing values coded as "N/A", "missing", and "na". But before we go ahead and start replacing these with NA, we should get an idea of how big the problem is.

Use miss_scan_count to count the possible missings in the dataset, *pacman*, a dataset of *pacman scores*, containing three columns:

year: the year that person made that score. *initial*: the initials of the person. *score*: the scores of that person.

For the pacman dataset, use *miss_scan_count*() to search for strange missing values:

"N/A", "missing", "na", and " " (a single space). To search for strange missing values all at once.

```
pacman<-read.csv("pacman.csv")
# Explore the strange missing values "N/A"
miss_scan_count(data = pacman, search = list("N/A"))
```

```
## # A tibble: 7 x 2
##   Variable     n
##   <chr>    <int>
## 1 X            0
## 2 year         0
## 3 month        0
## 4 day          0
## 5 initial      0
## 6 score      100
## 7 country      0
```

```
# Explore the strange missing values "missing"
miss_scan_count(data = pacman, search = list("missing"))
```

```
## # A tibble: 7 x 2
##   Variable     n
##   <chr>    <int>
## 1 X            0
## 2 year        93
## 3 month       93
## 4 day         93
## 5 initial      0
## 6 score        0
## 7 country      0
```

```
# Explore the strange missing values "na"
miss_scan_count(data = pacman, search = list("na"))
```

```
## # A tibble: 7 x 2
##   Variable     n
##   <chr>    <int>
## 1 X            0
## 2 year       100
## 3 month      100
## 4 day        100
## 5 initial      0
## 6 score        0
## 7 country      0
```

```
# Explore the strange missing values " " (a single space)
miss_scan_count(data = pacman, search = list(" "))
```

```
## # A tibble: 7 x 2
##   Variable     n
##   <chr>    <int>
## 1 X            0
## 2 year         0
## 3 month        0
## 4 day          0
## 5 initial      0
## 6 score        0
## 7 country      0
```

```
# Explore all of the strange missing values, "N/A", "missing", "na", " "
miss_scan_count(data = pacman, search = list("N/A", "missing","na"," "))
```

```
## # A tibble: 7 x 2
##   Variable     n
##   <chr>    <int>
## 1 X            0
## 2 year       193
## 3 month      193
## 4 day        193
## 5 initial      0
## 6 score      100
## 7 country      0
```

You’ve learned how to search for and count strange missing values

###Using replace_with_na

```
# Print the top of the pacman data using `head()`
pacman<-read.csv("pacman.csv")
head(pacman)
```

```
##   X year month day initial   score country
## 1 1 2007    10  27     LEX 2065812      CA
## 2 2 1995     8  23     PNY 1163465      JP
## 3 3 1980     2   8     MBJ  175380
## 4 4 1982     5   9     QRC 2025632      ES
## 5 5   na    na  na     YPZ  925357      NZ
## 6 6 2013    11  15     RVJ  319733      AU
```

```
# Replace the strange missing values "N/A", "na", and "missing" with `NA` for the variables, year, and
pacman_clean <- replace_with_na(pacman, replace = list(year = c("N/A", "na", "missing"),
                                score = c("N/A", "na", "missing")))

# Test if `pacman_clean` still has these values in it?
miss_scan_count(pacman_clean, search = list("N/A", "na", "missing"))
```

```
## # A tibble: 7 x 2
##   Variable     n
##   <chr>    <int>
## 1 X            0
## 2 year         0
## 3 month      193
## 4 day        193
## 5 initial      0
## 6 score        0
## 7 country      0
```

You’ve now learned how to search for, replace, and confirm strange missing values in a dataset.

### Using replace_with_na scoped variants

To reduce code repetition when replacing values with NA, use the “scoped variants” of replace_with_na():

- replace_with_na_at()
- replace_with_na_if()
- replace_with_na_all()
  The syntax of replacement looks like this: $\sim .x ==$ *“N/A”* This replaces all cases that are equal to “N/A”.

~.x %in% c("N/A", "missing", "na", " ") Replaces all cases that have "N/A", "missing", "na", or " ".

Lets try the following:

For the dataset pacman replace the same special missing values, "N/A", "missing", "na", and " ":

year, *month*, and *day*, using *replace_with_na_at*().
Only character variables using *replace_with_na_if*().
All variables using *replace_with_na_all*().

```
# Use `replace_with_na_at()` to replace with NA
rep_NA<-replace_with_na_at(pacman,
                    .vars = c("year", "month", "day"),
                    ~.x %in% c("N/A", "missing", "na", " "))

# Use `replace_with_na_if()` to replace with NA the character values using `is.character`
rep_NA_if<-replace_with_na_if(pacman,
                    .predicate = is.character,
                    ~.x %in% c("N/A", "missing", "na", " "))

# Use `replace_with_na_all()` to replace with NA
rep_NA_all<-replace_with_na_all(pacman, condition = ~.x %in% c("N/A", "missing", "na", " "))
```

Now you have learned to use the powerful scoped variants of *replace_with_na*() to replace values with NA.

Fix implicit missings using complete()

So far we have discussed about how to handle explore and search for and replace missing values. These values are explicitly missing. It is little more difficult to deal with implicit missing values. i.e values that are implied to be missing, but might not be explicitly listed. They are infact **missing**, missing values. So we will be dealing with those values which are not even in the data. To make implicit missing values explicit, we can use the *complete* function from *tidyr* package.

We are going to explore a new dataset, frogger.

This dataset contains 4 scores per player recorded at different times: *morning*, *afternoon*, *evening*, and *late_night.*

Every player should have played 4 games, one at each of these times, but it looks like not every player completed all of these games.

Use the *complete*() function to make these implicit missing values explicit

```
# Print the frogger data to have a look at it
frogger<-read.csv("frogger.csv")
frogger
```

```
##    S.No   name       time  value  X
## 1     1  jesse    morning   6678 NA
## 2     2  jesse  afternoon 800060 NA
## 3     3  jesse    evening 475528 NA
## 4     4  jesse late_night 143533 NA
## 5     5   andy    morning 425115 NA
## 6     6   andy  afternoon 587468 NA
## 7     7   andy late_night 111000 NA
## 8     8    nic  afternoon 588532 NA
## 9     9    nic late_night 915533 NA
## 10   10    dan    morning 388148 NA
## 11   11     dan     evening 180912 NA
## 12   12    alex     morning 552670 NA
## 13   13    alex   afternoon  98355 NA
## 14   14    alex     evening 266055 NA
## 15   15    alex  late_night 121056 NA
```

```
library(tidyr)
# Use `complete()` on the `time` and `name` variables to make implicit missing values explicit
frogger_tidy <- frogger %>% complete(name,time)
```

Fix explicit missings using fill()**

One type of missing value that can be obvious to deal with is where the first entry of a group is given, but subsequent entries are marked NA.

These missing values often result from empty values in spreadsheets to avoid entering multiple names multiple times; as well as for “human readability”.

This type of problem can be solved by using the fill() function from the tidyr package.

```
# Print the frogger data to have a look at it
frogger
```

```
##    S.No    name        time  value  X
## 1     1   jesse     morning   6678 NA
## 2     2   jesse   afternoon 800060 NA
## 3     3   jesse     evening 475528 NA
## 4     4   jesse  late_night 143533 NA
## 5     5    andy     morning 425115 NA
## 6     6    andy   afternoon 587468 NA
## 7     7    andy  late_night 111000 NA
## 8     8     nic   afternoon 588532 NA
## 9     9     nic  late_night 915533 NA
## 10   10     dan     morning 388148 NA
## 11   11     dan     evening 180912 NA
## 12   12    alex     morning 552670 NA
## 13   13    alex   afternoon  98355 NA
## 14   14    alex     evening 266055 NA
## 15   15    alex  late_night 121056 NA
```

```
# Use `fill()` to fill down the name variable in the frogger dataset
frogger %>% fill(name)
```

```
##    S.No    name        time  value  X
## 1     1   jesse     morning   6678 NA
## 2     2   jesse   afternoon 800060 NA
## 3     3   jesse     evening 475528 NA
## 4     4   jesse  late_night 143533 NA
## 5     5    andy     morning 425115 NA
## 6     6    andy   afternoon 587468 NA
## 7     7    andy  late_night 111000 NA
## 8     8     nic   afternoon 588532 NA
## 9     9     nic  late_night 915533 NA
## 10   10     dan     morning 388148 NA
## 11   11     dan     evening 180912 NA
## 12   12    alex     morning 552670 NA
## 13   13    alex   afternoon  98355 NA
## 14   14    alex     evening 266055 NA
## 15   15    alex  late_night 121056 NA
```

The fill function makes it easy to fill down observations.

Using complete() and fill() together**

Now let's put it together!

Use complete() and fill() together to fix explicit and implicitly missing values in the frogger dataset. Use fill() and complete() to correctly fill and complete the values in the data so we went up with a dataset that has 5 names, all completed, each with 4 times filled in.

```
# Print the frogger data to have a look at it
frogger
```

```
##    S.No    name        time  value  X
## 1     1   jesse     morning   6678 NA
## 2     2   jesse   afternoon 800060 NA
## 3     3   jesse     evening 475528 NA
## 4     4   jesse  late_night 143533 NA
## 5     5    andy     morning 425115 NA
## 6     6    andy   afternoon 587468 NA
## 7     7    andy  late_night 111000 NA
## 8     8     nic   afternoon 588532 NA
## 9     9     nic  late_night 915533 NA
## 10   10     dan     morning 388148 NA
## 11   11     dan     evening 180912 NA
## 12   12    alex     morning 552670 NA
## 13   13    alex   afternoon  98355 NA
## 14   14    alex     evening 266055 NA
## 15   15    alex  late_night 121056 NA
```

```
# Correctly fill() and complete() missing values so that our dataset becomes sensible
frogger %>%
  fill(name) %>%
  complete(name, time)
```

```
## # A tibble: 35 x 5
##    name      time           S.No  value X
##    <chr>     <chr>         <dbl>  <int> <lgl>
##  1 "    nic" "    evening"    NA     NA NA
##  2 "    nic" "    morning"    NA     NA NA
##  3 "    nic" "   morning"     NA     NA NA
##  4 "    nic" "  afternoon"     8 588532 NA
##  5 "    nic" " late_night"    NA     NA NA
##  6 "   andy" "    evening"    NA     NA NA
##  7 "   andy" "    morning"    NA     NA NA
##  8 "   andy" "   morning"      5 425115 NA
##  9 "   andy" "  afternoon"     6 587468 NA
## 10 "   andy" " late_night"     7 111000 NA
## # ... with 25 more rows
## # i Use `print(n = ...)` to see more rows
```

Now we have correctly used the fill and complete functions in the right order to create a sensible dataset.

Exploring missingness dependence

After finding out the missing values , we need to decide whether to delete them or impute? Deleteing and imputing values can have some serious implications on future decisions that we make from our data. To help us frame these decisions we need to discuss some concepts in missing data theory- missing data dependence.

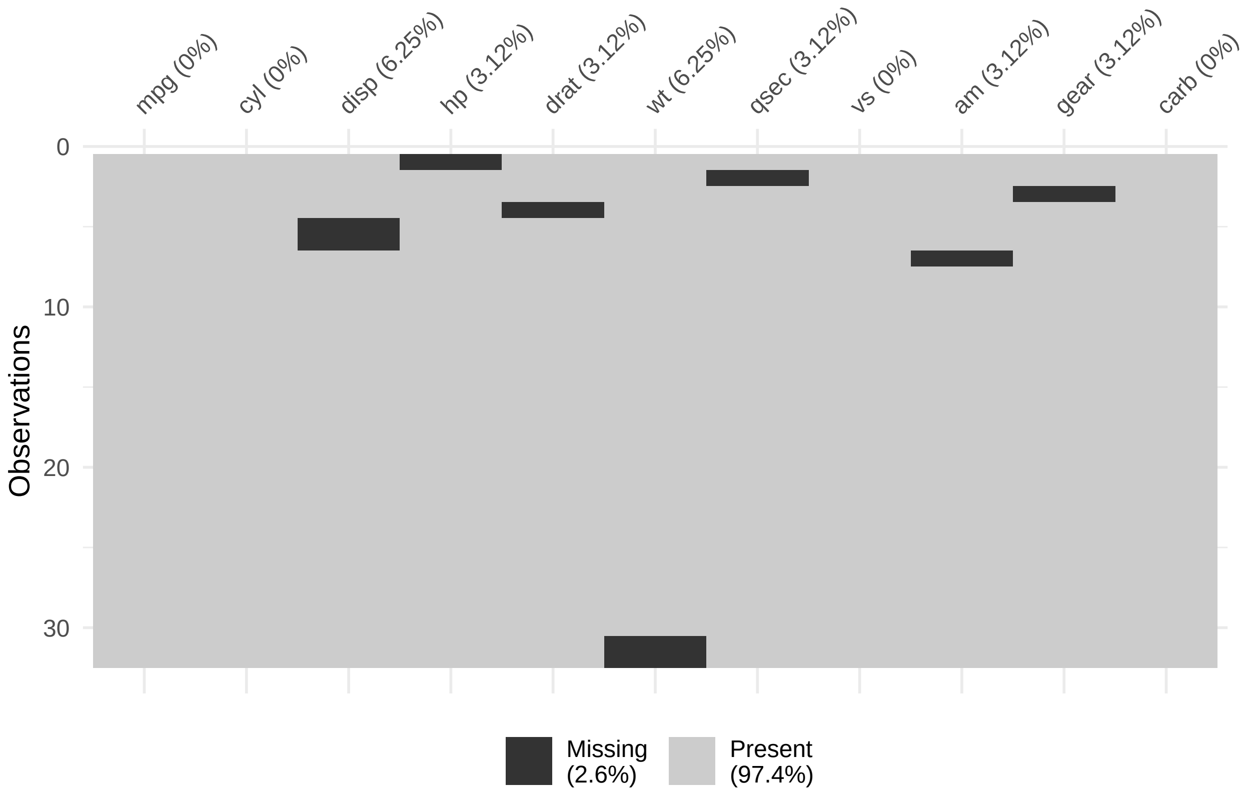
There can be three possible missingness:

- **MCAR** Missing Completely at Random

Missingness has no association with any data you have observed, or not observed. The implications are - Imputation is advisable - Deleting observations may reduce sample size, limiting inference, but will not bias[ideally do not delete unless there is less than 5% data loss]
- You should be imputing data

An example is as shown below:

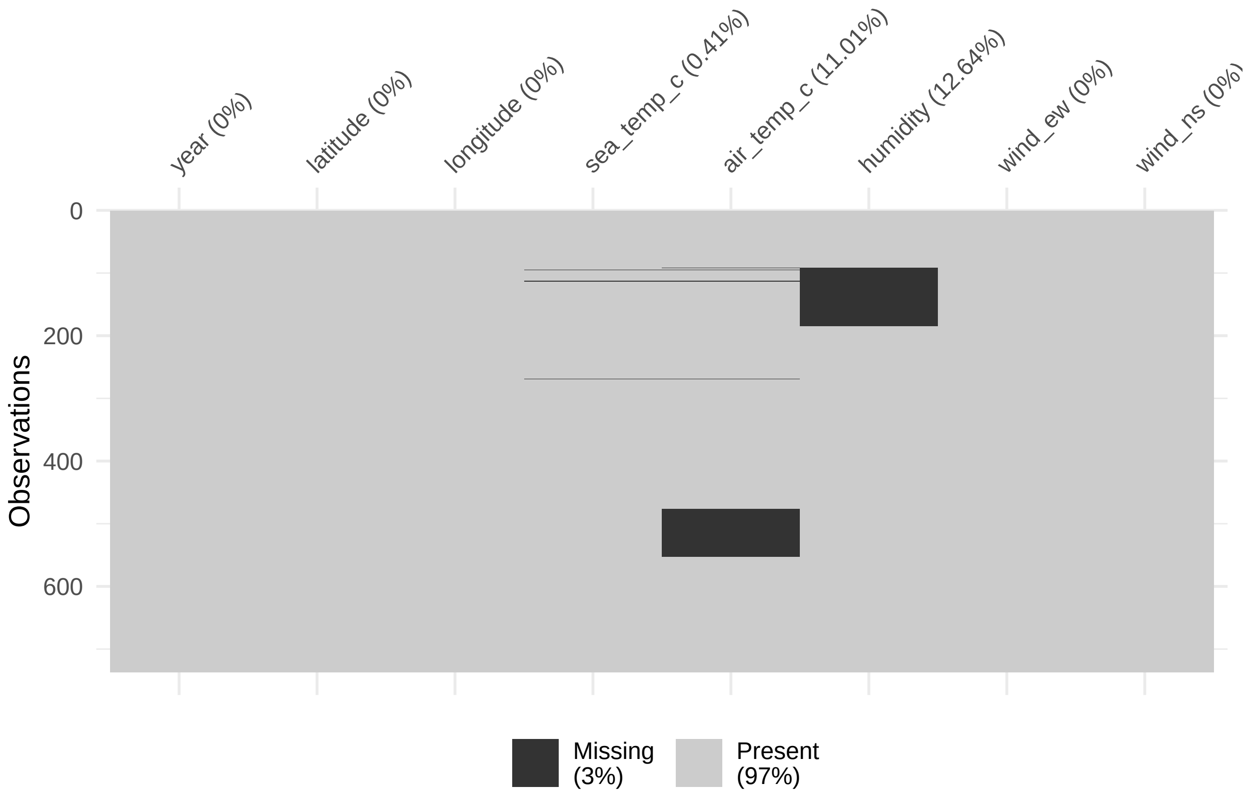
```
library(visdat)
mt_cars<-read.csv("mt_cars.csv")
vis_miss(mt_cars, cluster = TRUE)
```



- **MAR** Missing At Random

Missingness depends on data observed, but not data observed
Implications: - Impute
- Deleting observations not ideal, may lead to bias Example:

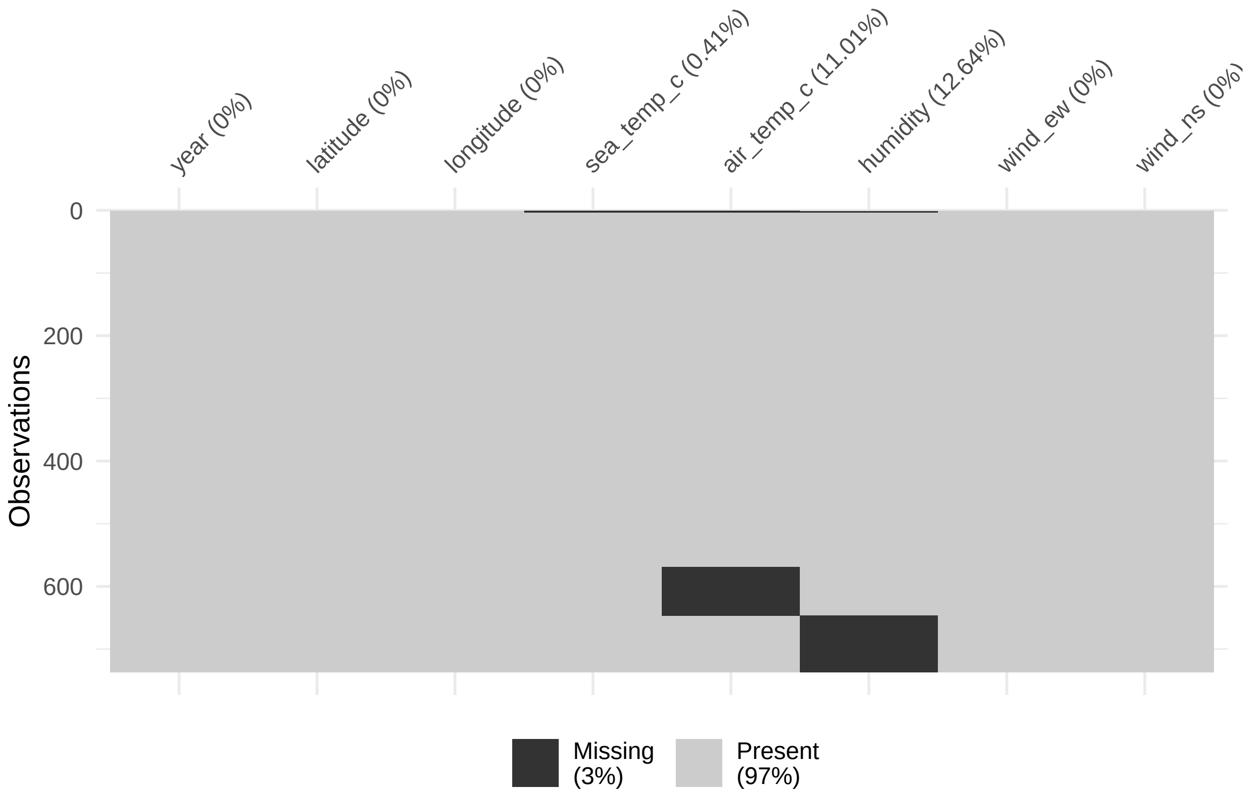
```
oceanbuoys %>% arrange(year) %>% vis_miss()
```



- **MNAR** Missing Not At Random
  Missingness of the response is related to an unobserved value relevant to the assessment of interest
  Implications:
  - Data will be biased from deletion and imputation
  - Inference can be limited, proceed with caution.

Example:

```
ocean<-read.csv("ocean.csv")
vis_miss(oceanbuoys, cluster = TRUE)
```



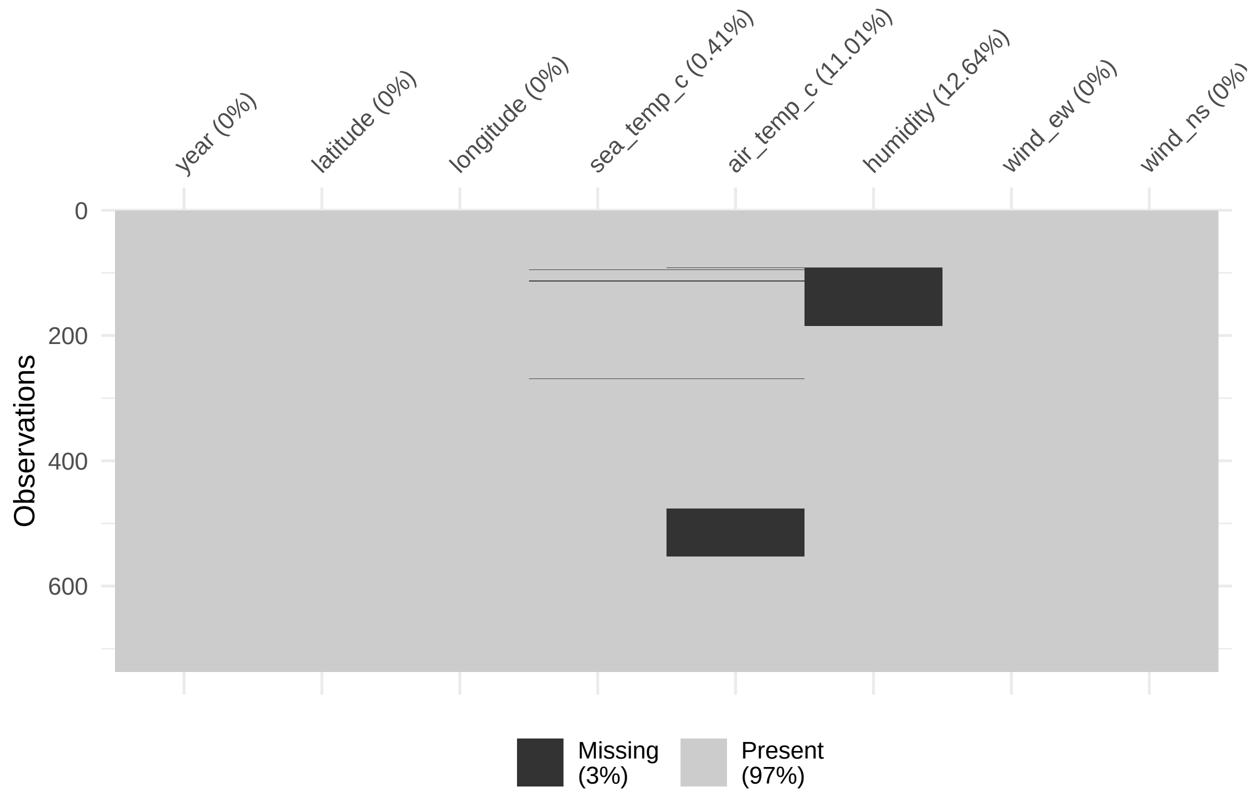
To learn about the structure of the missingness in data, you can explore how sorting changes how missingness is presented.

For the oceanbuoys dataset, explore the missingness with *vis_miss()*, and then arrange by a few different variables

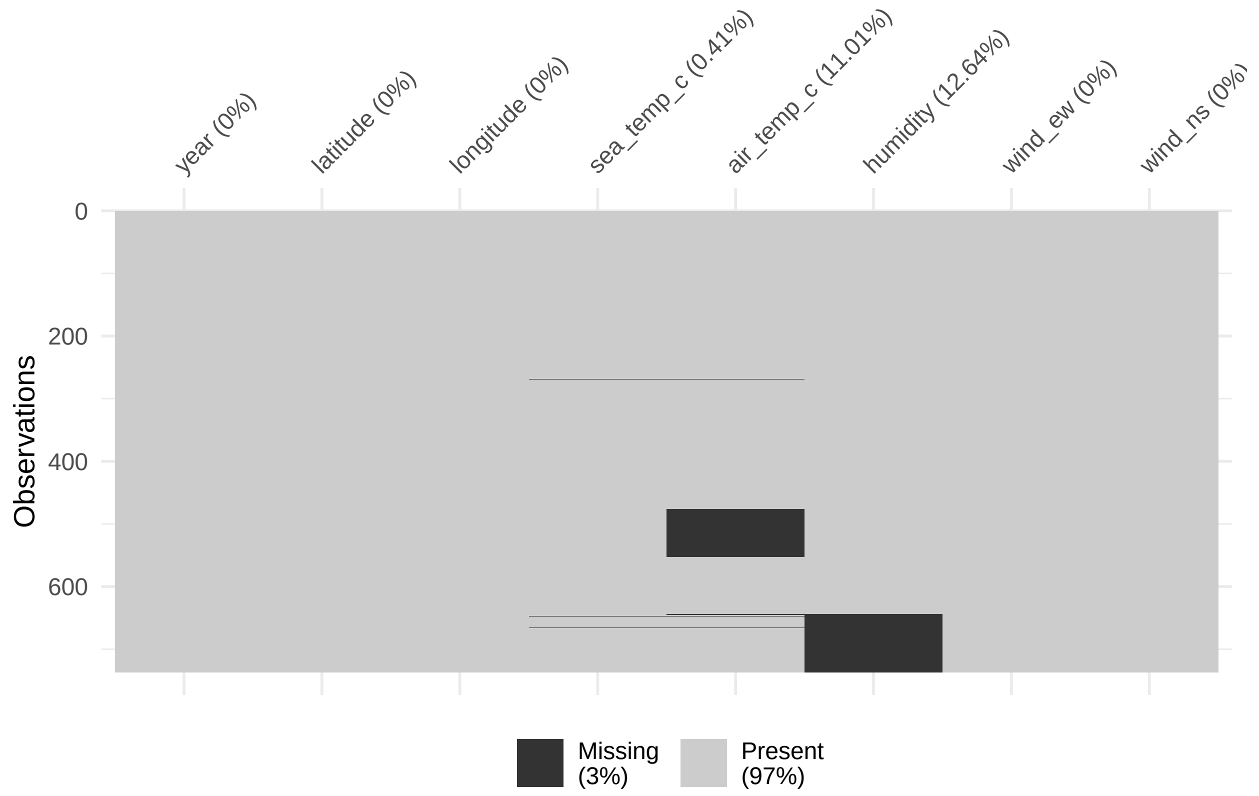
This is not a definitive process, but it will get you started to ask the right questions of your data. We explore more powerful techniques in the next section. Using the dataset, *oceanbuoys*:

Arrange by the variables, *year*, *latitude*, and *wind_ew* using *vis_miss()*.

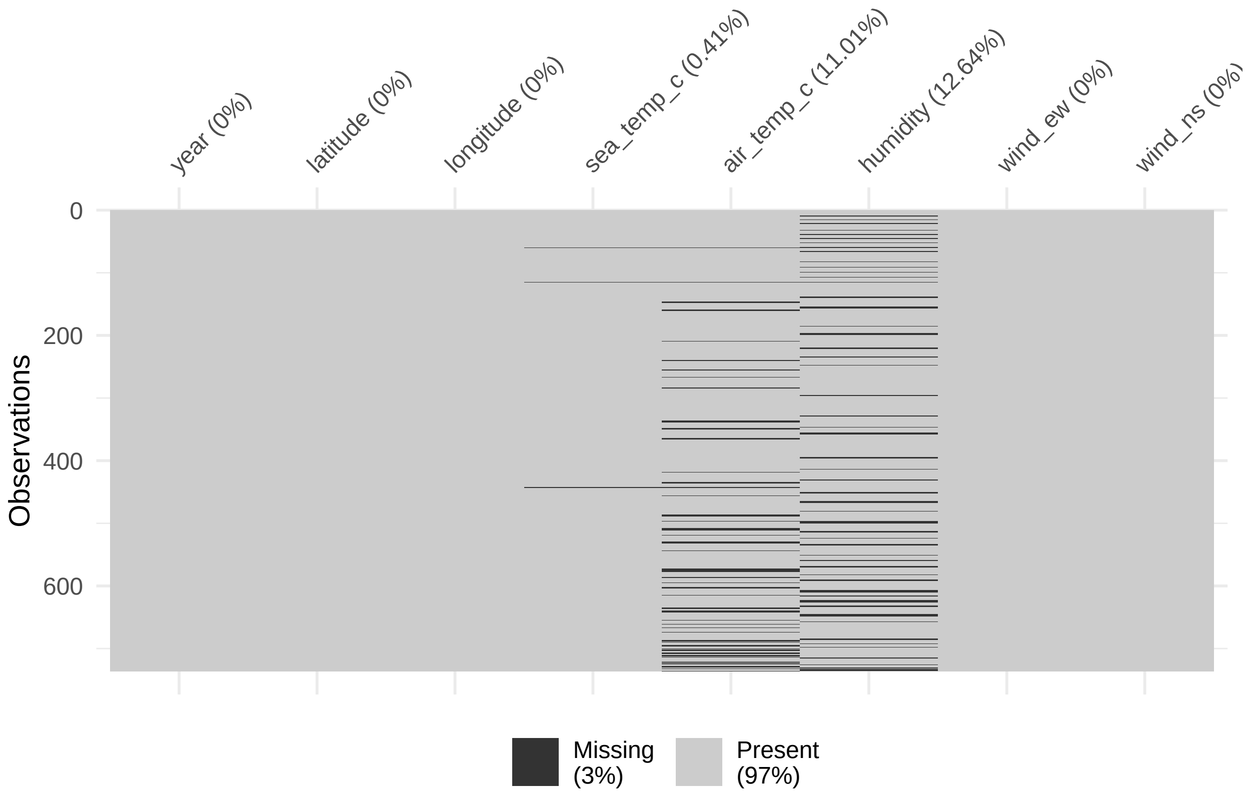
```r
# Arrange by year
oceanbuoys %>% arrange(year) %>% vis_miss()
```



```r
# Arrange by latitude
oceanbuoys %>% arrange(latitude) %>% vis_miss()
```



```
# Arrange by wind_ew (wind east west)
oceanbuoys %>% arrange(wind_ew) %>% vis_miss()
```



Testing missing relationships

We have seen how to create summaries and visualize missing values. But we havent discussed about how to link these summaries of missingness back to values in the data. In this section, we will discuss about a special data structure to facilitate working with missing data: the shadow matrix and nabular data Shadow matrix has the following features: -Coordinated names- Variables in the shadow matrix gain the same name as in the data with the suffix"_NA" -Clear values- The values are either *NA* or *!NA* 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 reffered to as *nabular data*

Creating shadow matrix data

Missing data can be tricky to think about, as they don't usually proclaim themselves for you, and instead hide amongst the weeds of the data.

One way to help expose missing values is to change the way we think about the data - by thinking about every single data value being missing or not missing.

The as_shadow() function in R transforms a dataframe into a shadow matrix, a special data format where the values are either missing (NA), or Not Missing (!NA).

The column names of a shadow matrix are the same as the data, but have a suffix added _NA.

To keep track of and compare data values to their missingness state, use the bind_shadow() function. Having data in this format, with the shadow matrix column bound to the regular data is called nabular data.

Using the oceanbuoys dataset:

Create shadow matrix data with as_shadow() Create nabular data by binding the shadow to the data with bind_shadow() Bind only the variables with missing values by using bind_shadow(only_miss = TRUE)

```
# Create shadow matrix data with `as_shadow()`
as_shadow(oceanbuoys)
```

```
## # A tibble: 736 x 8
##    year_NA latitude_NA longitude_NA sea_temp_c~1 air_t~2 humid~3 wind_~4 wind_~5
##    <fct>   <fct>       <fct>        <fct>        <fct>   <fct>   <fct>   <fct>
##  1 !NA     !NA         !NA          !NA          !NA     !NA     !NA     !NA
##  2 !NA     !NA         !NA          !NA          !NA     !NA     !NA     !NA
##  3 !NA     !NA         !NA          !NA          !NA     !NA     !NA     !NA
##  4 !NA     !NA         !NA          !NA          !NA     !NA     !NA     !NA
##  5 !NA     !NA         !NA          !NA          !NA     !NA     !NA     !NA
##  6 !NA     !NA         !NA          !NA          !NA     !NA     !NA     !NA
##  7 !NA     !NA         !NA          !NA          !NA     !NA     !NA     !NA
##  8 !NA     !NA         !NA          !NA          !NA     !NA     !NA     !NA
##  9 !NA     !NA         !NA          !NA          !NA     !NA     !NA     !NA
## 10 !NA     !NA         !NA          !NA          !NA     !NA     !NA     !NA
## # ... with 726 more rows, and abbreviated variable names 1: sea_temp_c_NA,
## #   2: air_temp_c_NA, 3: humidity_NA, 4: wind_ew_NA, 5: wind_ns_NA
## # i Use `print(n = ...)` to see more rows
```

```
# Create nabular data by binding the shadow to the data with `bind_shadow()`
bind_shadow(oceanbuoys)
```

```
## # A tibble: 736 x 16
##     year latit~1 longi~2 sea_t~3 air_t~4 humid~5 wind_ew wind_ns year_NA latit~6
##    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl> <fct>   <fct>
##  1  1997       0    -110    27.6    27.1    79.6   -6.40    5.40 !NA     !NA
##  2  1997       0    -110    27.5    27.0    75.8   -5.30    5.30 !NA     !NA
##  3  1997       0    -110    27.6    27      76.5   -5.10    4.5  !NA     !NA
##  4  1997       0    -110    27.6    26.9    76.2   -4.90    2.5  !NA     !NA
##  5  1997       0    -110    27.6    26.8    76.4   -3.5     4.10 !NA     !NA
##  6  1997       0    -110    27.8    26.9    76.7   -4.40    1.60 !NA     !NA
##  7  1997       0    -110    28.0    27.0    76.5   -2       3.5  !NA     !NA
##  8  1997       0    -110    28.0    27.1    78.3   -3.70    4.5  !NA     !NA
##  9  1997       0    -110    28.0    27.2    78.6   -4.20    5    !NA     !NA
## 10  1997       0    -110    28.0    27.2    76.9   -3.60    3.5  !NA     !NA
## # ... with 726 more rows, 6 more variables: longitude_NA <fct>,
## #   sea_temp_c_NA <fct>, air_temp_c_NA <fct>, humidity_NA <fct>,
## #   wind_ew_NA <fct>, wind_ns_NA <fct>, and abbreviated variable names
## #   1: latitude, 2: longitude, 3: sea_temp_c, 4: air_temp_c, 5: humidity,
## #   6: latitude_NA
## # i Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

```
# Bind only the variables with missing values by using bind_shadow(only_miss = TRUE)
bind_shadow(oceanbuoys, only_miss = TRUE)
```

```
## # A tibble: 736 x 11
##     year latit~1 longi~2 sea_t~3 air_t~4 humid~5 wind_ew wind_ns sea_t~6 air_t~7
##    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl> <fct>   <fct>
##  1  1997       0    -110    27.6    27.1    79.6   -6.40    5.40 !NA     !NA
##  2  1997       0    -110    27.5    27.0    75.8   -5.30    5.30 !NA     !NA
##  3  1997       0    -110    27.6    27      76.5   -5.10    4.5  !NA     !NA
##  4  1997       0    -110    27.6    26.9    76.2   -4.90    2.5  !NA     !NA
##  5  1997       0    -110    27.6    26.8    76.4   -3.5     4.10 !NA     !NA
##  6  1997       0    -110    27.8    26.9    76.7   -4.40    1.60 !NA     !NA
##  7  1997       0    -110    28.0    27.0    76.5   -2       3.5  !NA     !NA
##  8  1997       0    -110    28.0    27.1    78.3   -3.70    4.5  !NA     !NA
##  9  1997       0    -110    28.0    27.2    78.6   -4.20    5    !NA     !NA
## 10  1997       0    -110    28.0    27.2    76.9   -3.60    3.5  !NA     !NA
## # ... with 726 more rows, 1 more variable: humidity_NA <fct>, and abbreviated
## #   variable names 1: latitude, 2: longitude, 3: sea_temp_c, 4: air_temp_c,
## #   5: humidity, 6: sea_temp_c_NA, 7: air_temp_c_NA
## # i Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

We can now create shadow matrix and nabular data. This is a useful first step in more advanced summaries of missing data.

Performing grouped summaries of missingness

Now that you can create nabular data, let's use it to explore the data. Let's calculate summary statistics based on the missingness of another variable.

To do this we are going to use the following steps:

First, *bind_shadow()* turns the data into nabular data.

Next, perform some summaries on the data using *group_by()* and *summarise()* to calculate the mean and standard deviation, using the *mean()* and *sd()* functions

For the oceanbuoys dataset:

bind_shadow(), then *group_by()* for the missingness of *humidity (humidity_NA)* and calculate the means and standard deviations for wind east west *(wind_ew)* using *summarise()* from *dplyr.*

Repeat this, but calculating summaries for wind north south (wind_ns).

```
# `bind_shadow()` and `group_by()` humidity missingness (`humidity_NA`)
oceanbuoys %>%
  bind_shadow() %>%
  group_by(humidity_NA) %>%
  summarise(wind_ew_mean = mean(wind_ew), # calculate mean of wind_ew
            wind_ew_sd = sd(wind_ew)) # calculate standard deviation of wind_ew
```

```
## # A tibble: 2 x 3
##   humidity_NA wind_ew_mean wind_ew_sd
##   <fct>              <dbl>      <dbl>
## 1 !NA                -3.78       1.90
## 2 NA                 -3.30       2.31
```

```
# Repeat this, but calculating summaries for wind north south (`wind_ns`).
oceanbuoys %>%
  bind_shadow() %>%
  group_by(humidity_NA) %>%
  summarise(wind_ns_mean = mean(wind_ns),
            wind_ns_sd = sd(wind_ns))
```

```
## # A tibble: 2 x 3
##   humidity_NA wind_ns_mean wind_ns_sd
##   <fct>              <dbl>      <dbl>
## 1 !NA                 2.78       2.06
## 2 NA                    1.66       2.23
```

Now we can summarise and explore values at each level of missingness.

Further exploring more combinations of missingness

It can be useful to get a bit of extra information about the number of cases in each missing condition.

In this exercise, we are going to add information about the number of observed cases using *n()* inside the *summarise()* function.

We will then add an additional level of grouping by looking at the combination of humidity being missing (humidity_NA) and air temperature being missing (air_temp_c_NA).

We will do the following:

Using group_by() and summarise() on wind_ew:

Summarize by the missingness of air_temp_c_NA.
Summarize by missingness of air_temp_c_NA and humidity_NA.

```
# Summarise wind_ew by the missingness of `air_temp_c_NA`
oceanbuoys %>%
  bind_shadow() %>%
  group_by(air_temp_c_NA) %>%
  summarise(wind_ew_mean = mean(wind_ew),
            wind_ew_sd = sd(wind_ew),
            n_obs = n())
```

```
## # A tibble: 2 x 4
##   air_temp_c_NA wind_ew_mean wind_ew_sd n_obs
##   <fct>                <dbl>      <dbl> <int>
## 1 !NA                  -3.91       1.85   655
## 2 NA                   -2.17       2.14    81
```

```
# Summarise wind_ew by missingness of `air_temp_c_NA` and `humidity_NA`
oceanbuoys %>%
  bind_shadow() %>%
  group_by(air_temp_c_NA, humidity_NA) %>%
  summarise(wind_ew_mean = mean(wind_ew),
            wind_ew_sd = sd(wind_ew),
            n_obs = n())
```

```
## `summarise()` has grouped output by 'air_temp_c_NA'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 4 x 5
## # Groups:   air_temp_c_NA [2]
##   air_temp_c_NA humidity_NA wind_ew_mean wind_ew_sd n_obs
##   <fct>         <fct>              <dbl>      <dbl> <int>
## 1 !NA           !NA                -4.01       1.74   565
## 2 !NA           NA                 -3.24       2.31    90
## 3 NA            !NA                -2.06       2.08    78
## 4 NA            NA                 -4.97       1.74     3
```

By counting the number of observations in each of the groups, we can get some useful context for our summary statistics.

Nabular data and filling by missingness

We will check how variables vary when other variables go missing. We can explore missingness using ggplot visualizations.

Summary statistics are useful to calculate, but as they say, a picture tells you a thousand words.

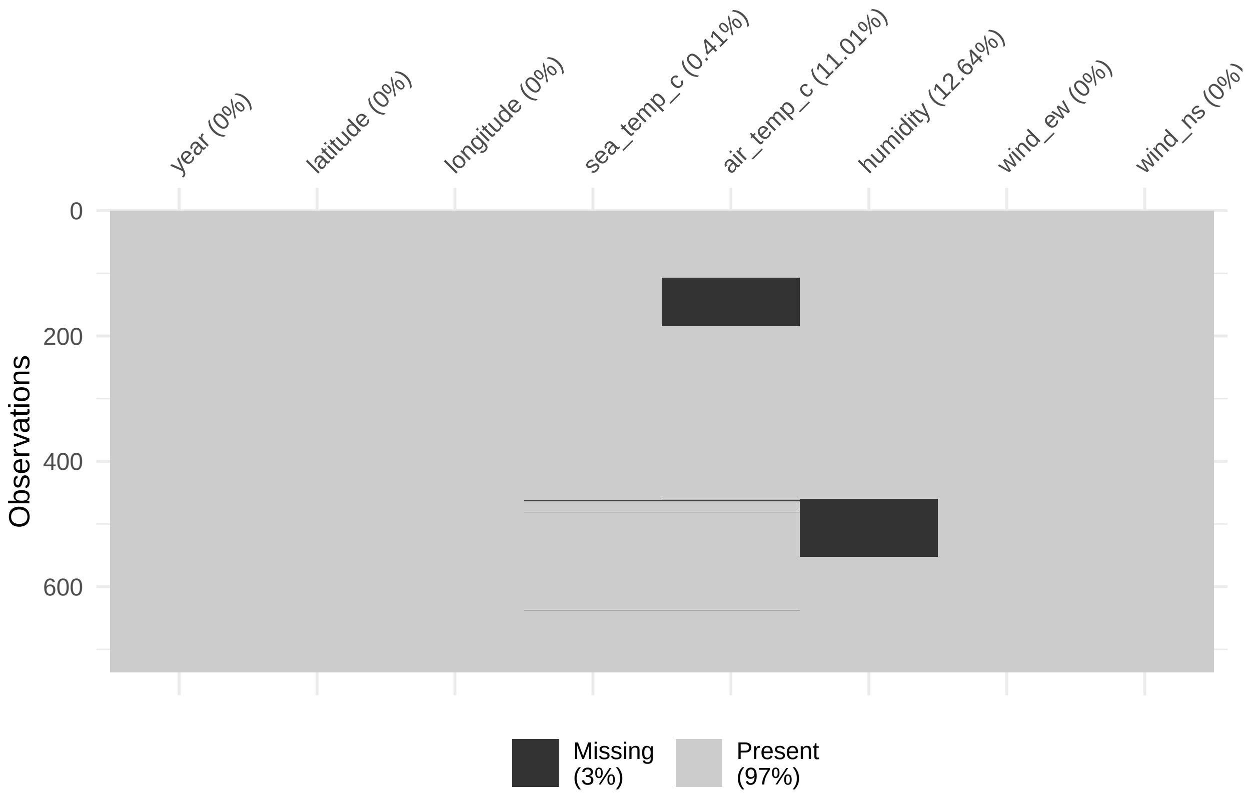
we are going to explore how you can use nabular data to explore the variation in a variable by the missingness of another.

We are going to use the *oceanbuoys* dataset from *naniar*

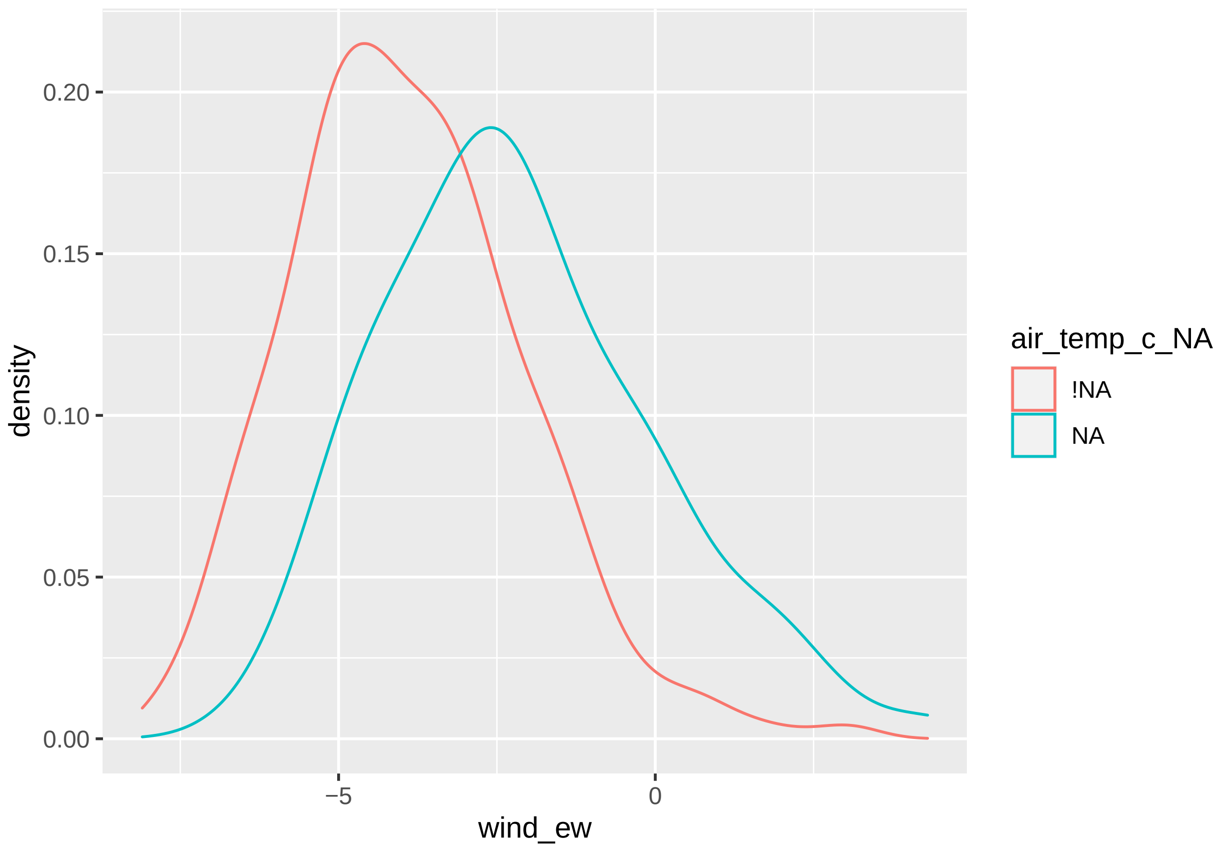
The goal is to:

First explore the missingness structure of oceanbuoys using vis_miss().
Explore the distribution of wind east west (wind_ew) for the missingness of air temperature using geom_density().
Explore the distribution of sea temperature for the missingness of humidity using geom_density()

```
# First explore the missingness structure of `oceanbuoys` using `vis_miss()`
vis_miss(oceanbuoys)
```

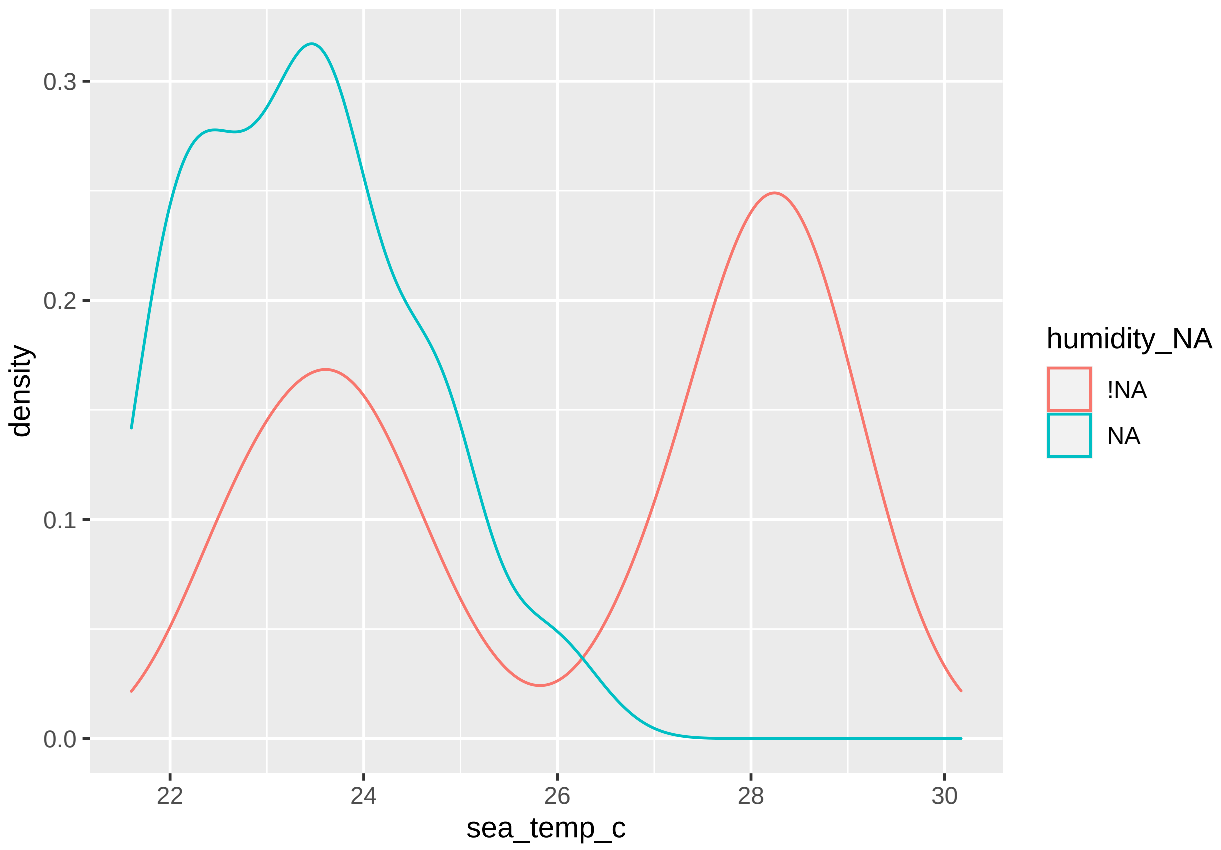


```
library(ggplot2)
# Explore the distribution of `wind_ew` for the missingness of `air_temp_c_NA` using  `geom_density()`
bind_shadow(oceanbuoys) %>%
  ggplot(aes(x = wind_ew,
             color = air_temp_c_NA)) +
  geom_density()
```



```
# Explore the distribution of sea temperature for the missingness of humidity (humidity_NA) using  `geo
  bind_shadow(oceanbuoys) %>%
  ggplot(aes(x = sea_temp_c,
             color = humidity_NA)) +
  geom_density()
```

```
## Warning: Removed 3 rows containing non-finite values (stat_density).
```



You can now use nabular data to visualise and explore missing data using density plots.

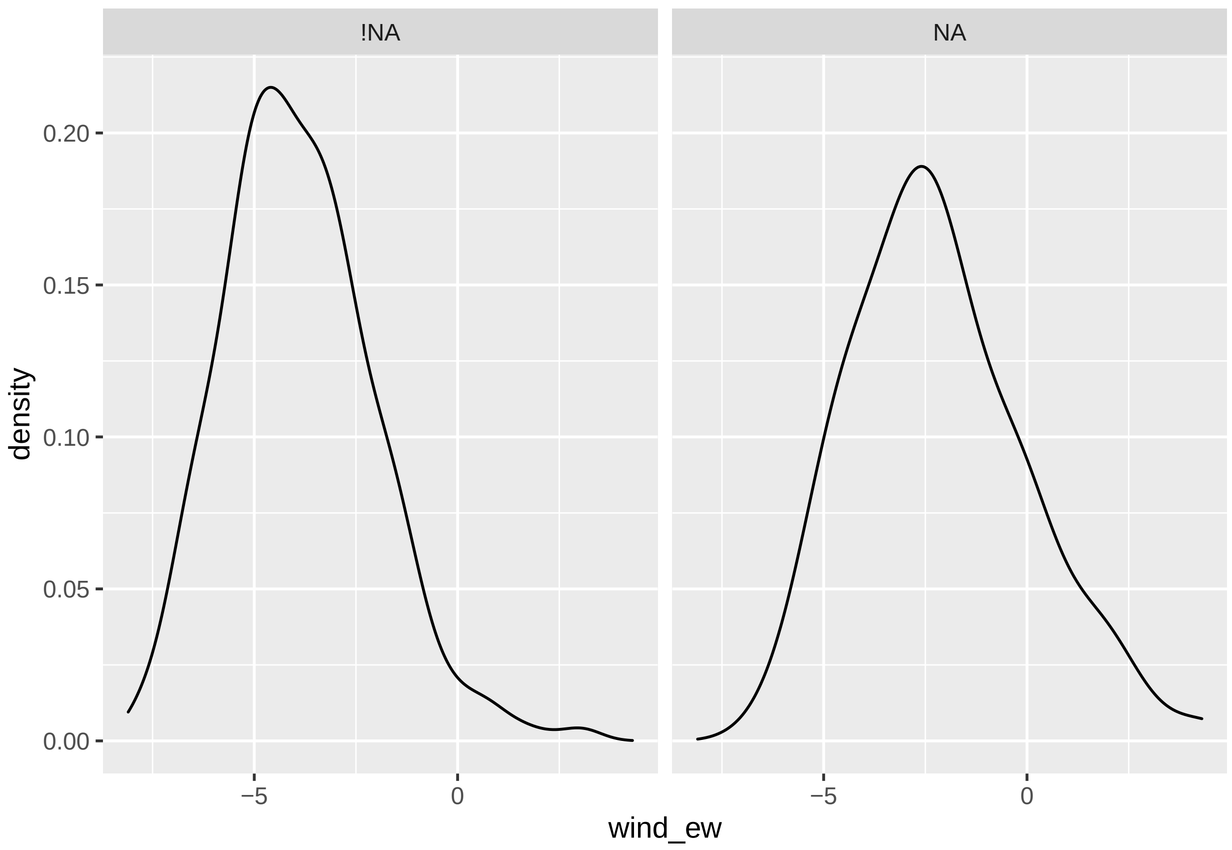
Nabular data and summarising by missingness
In this exercise, we are going to explore how to use nabular data to explore the variation in a variable by the missingness of another.

We are going to use the oceanbuoys dataset from naniar, and then create multiple plots of the data using facets.

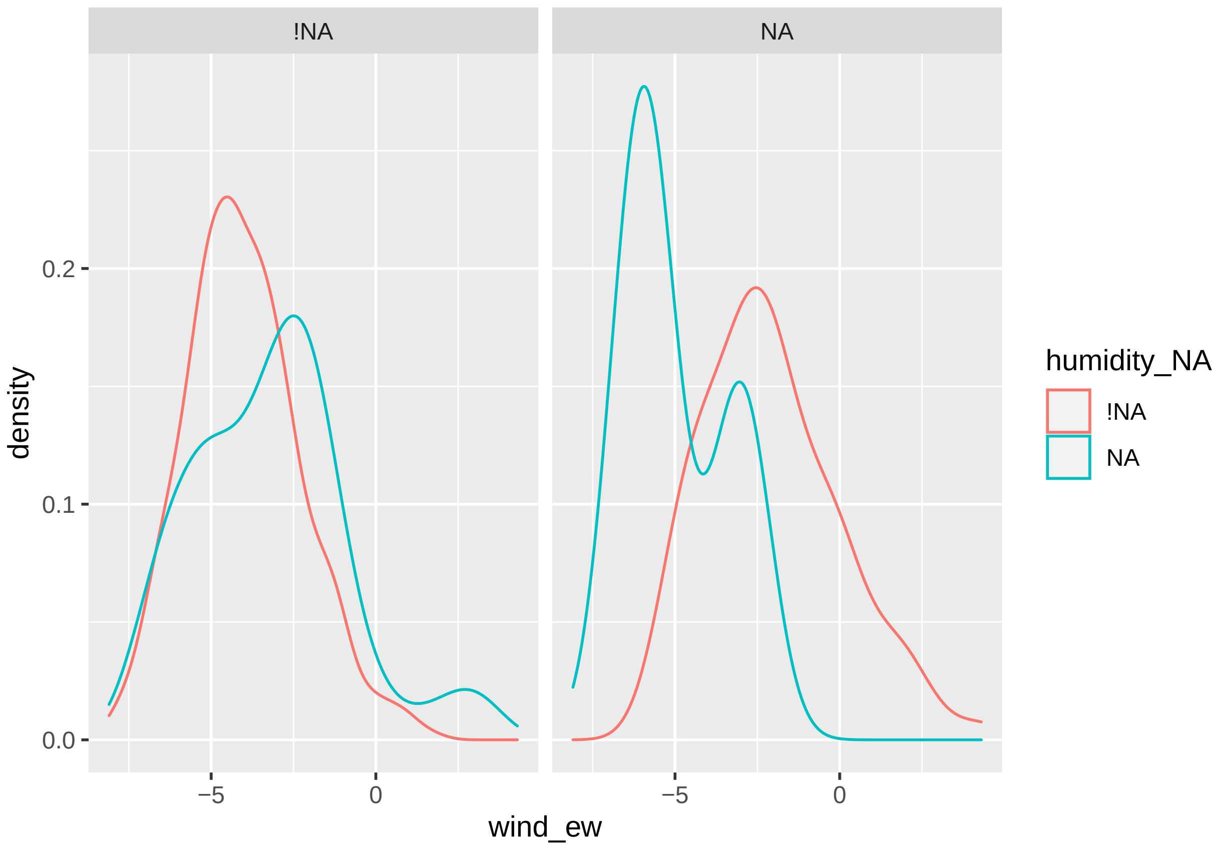
This allows you to explore different layers of missingness. Lets try the following:

Explore the distribution of wind east west (wind_ew) for the missingness of air temperature using geom_density() and faceting by the missingness of air temperature (air_temp_c_NA).
Build upon this visualization by filling by the missingness of humidity (humidity_NA).

```
# Explore the distribution of wind east west (`wind_ew`) for the missingness of air temperature using
oceanbuoys %>%
  bind_shadow() %>%
  ggplot(aes(x = wind_ew)) +
  geom_density() +
  facet_wrap(~air_temp_c_NA)
```



```
# Build upon this visualisation by filling by the missingness of humidity (`humidity_NA`).
oceanbuoys %>%
  bind_shadow() %>%
  ggplot(aes(x = wind_ew,
             color = humidity_NA)) +
  geom_density() +
  facet_wrap(~air_temp_c_NA)
```

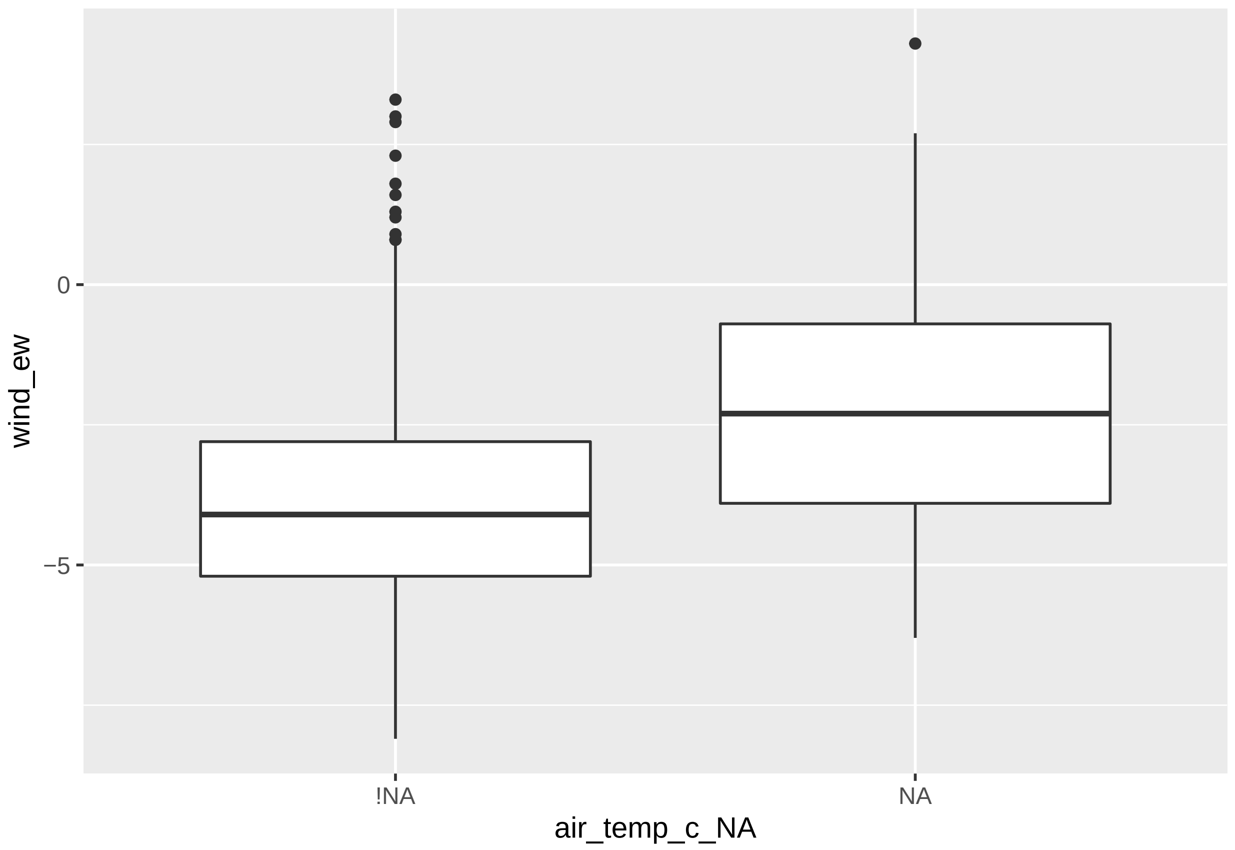


### Explore variation by missingness: boxplots Previous sections use nabular data along with density plots to explore the variation in a variable by the missingness of another.

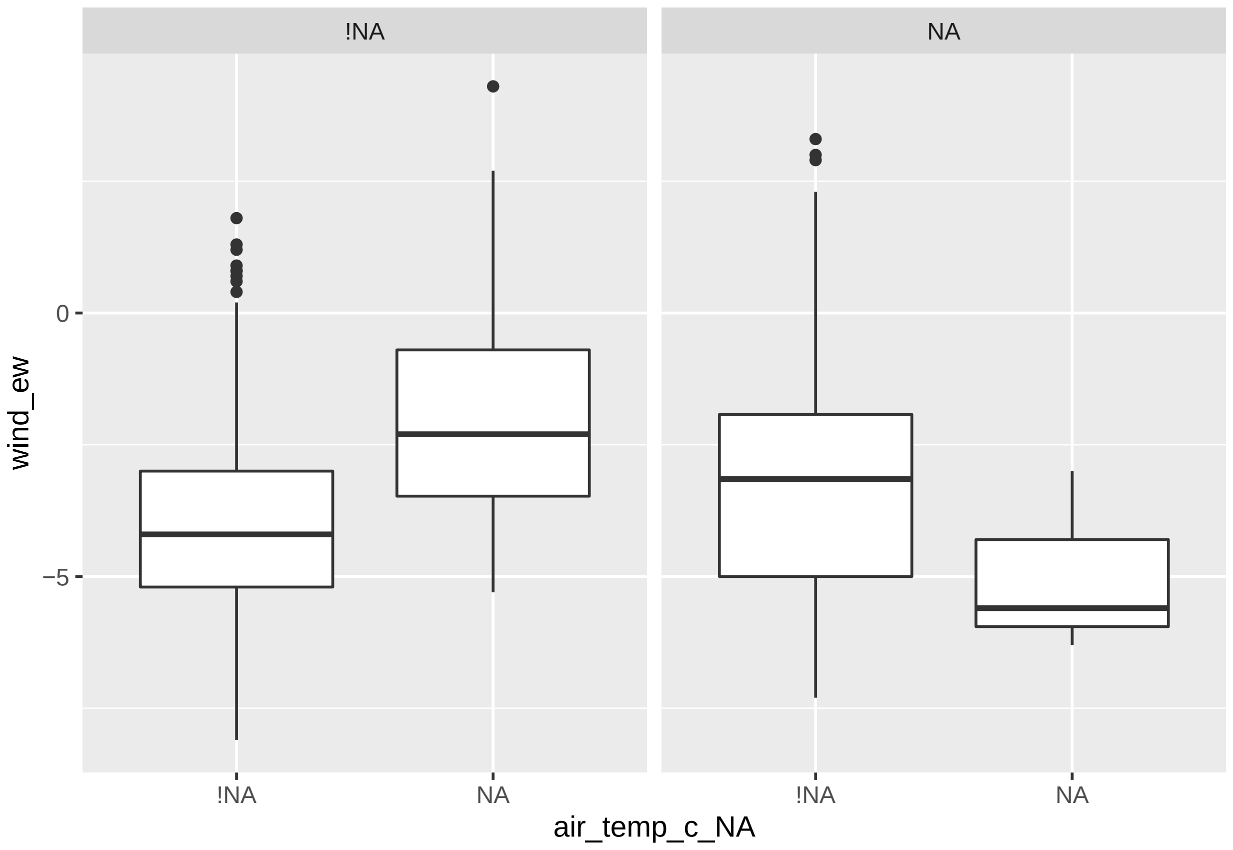
We are going to use the oceanbuoys dataset from naniar, using boxplots instead of facets or others to explore different layers of missingness. We will do the following:

Explore the distribution of wind east west (wind_ew) for the missingness of air temperature using geom_boxplot()
Build upon this visualisation by facetting by the missingness of humidity (humidity_NA).

```
# Explore the distribution of wind east west (`wind_ew`) for the missingness of air temperature using
oceanbuoys %>%
  bind_shadow() %>%
  ggplot(aes(x = air_temp_c_NA,
             y = wind_ew)) +
  geom_boxplot()
```



```
# Build upon this visualisation by facetting by the missingness of humidity (`humidity_NA`).
oceanbuoys %>%
  bind_shadow() %>%
  ggplot(aes(x = air_temp_c_NA,
             y = wind_ew)) +
  geom_boxplot() +
  facet_wrap(~humidity_NA)
```



You can now use nabular data to visualise and explore missing data with boxplots and facet wraps.

Visualizing missingness across two variables

Missing values are typically ignored in a scatterplot. Here we will discuss about how to visualize missings in a scatter plotand how and why that works. The problem with visualising a scatterplot when the data has missing values is that it removes any observations- entire rows- that have missing values. *ggplot* is very nice here, and gives a warning that missing values are being dropped. To explore the missings in a scatterplot , we can use *geom_miss_point()* , this visualises the missing values by placing them in the margins.

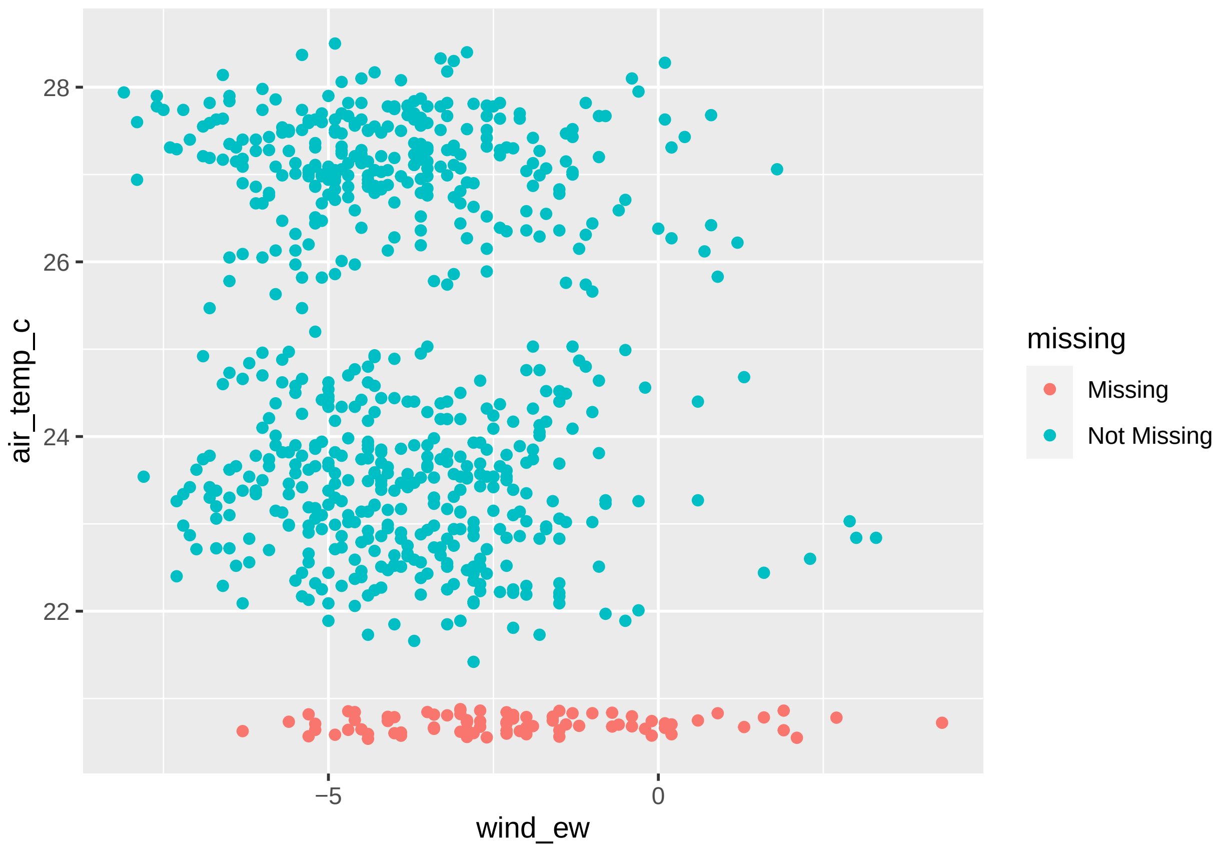
The problem of visualizing missing data in two dimensions

Exploring missing data with scatterplots Missing values in a scatterplot in ggplot2 are removed by default, with a warning.

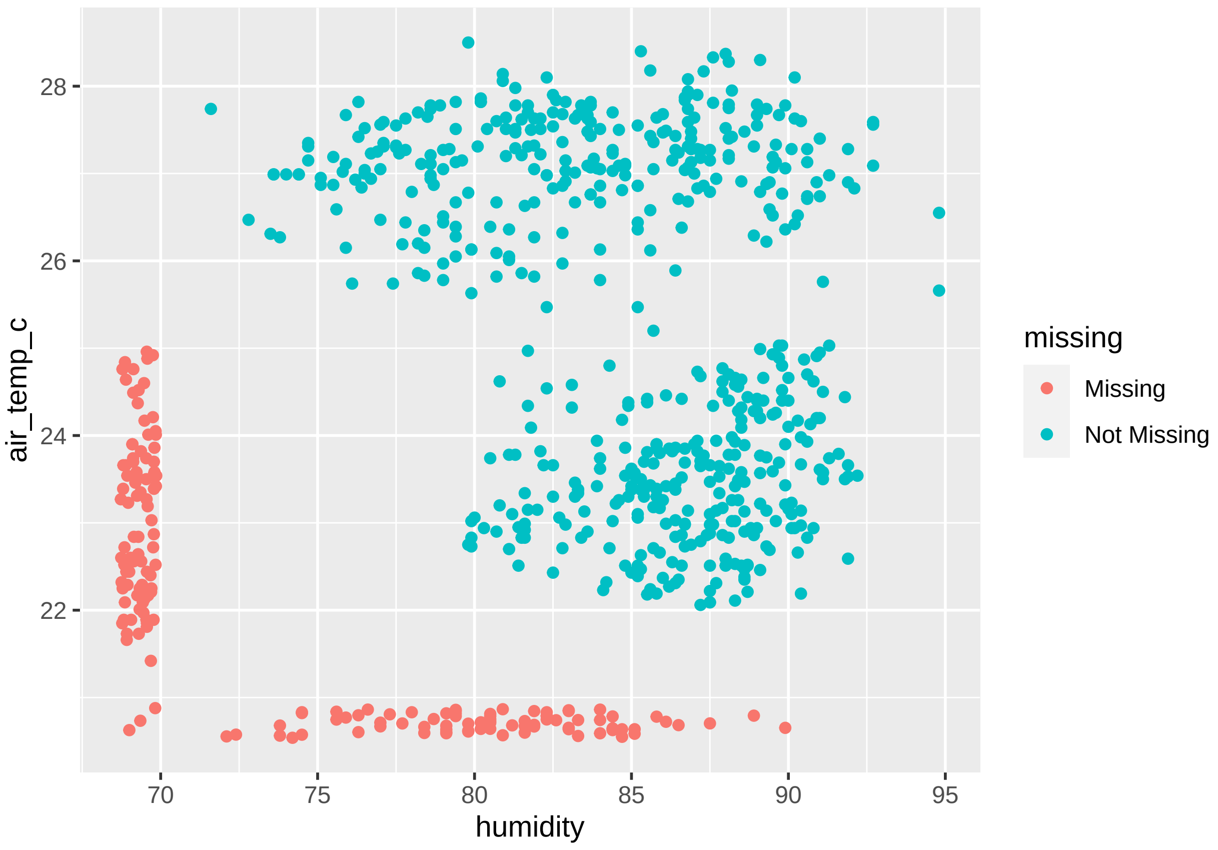
We can display missing values in a scatterplot, using geom_miss_point() - a special ggplot2 geom that shifts the missing values into the plot, displaying them 10% below the minimum of the variable.

Explore the missingness in wind east west (wind_ew) and air temperature, and display the missingness using geom_miss_point(). Explore the missingness in humidity and air temperature, and display the missingness using geom_miss_point().

```
# Explore the missingness in wind and air temperature, and display the missingness using `geom_miss_poi
ggplot(oceanbuoys,
       aes(x = wind_ew,
           y = air_temp_c)) +
  geom_miss_point()
```



```r
# Explore the missingness in humidity and air temperature, and display the missingness using `geom_miss
ggplot(oceanbuoys,
       aes(x = humidity,
           y = air_temp_c)) +
  geom_miss_point()
```



Now you can use geom_miss_point() to explore missing values in a scatterplot

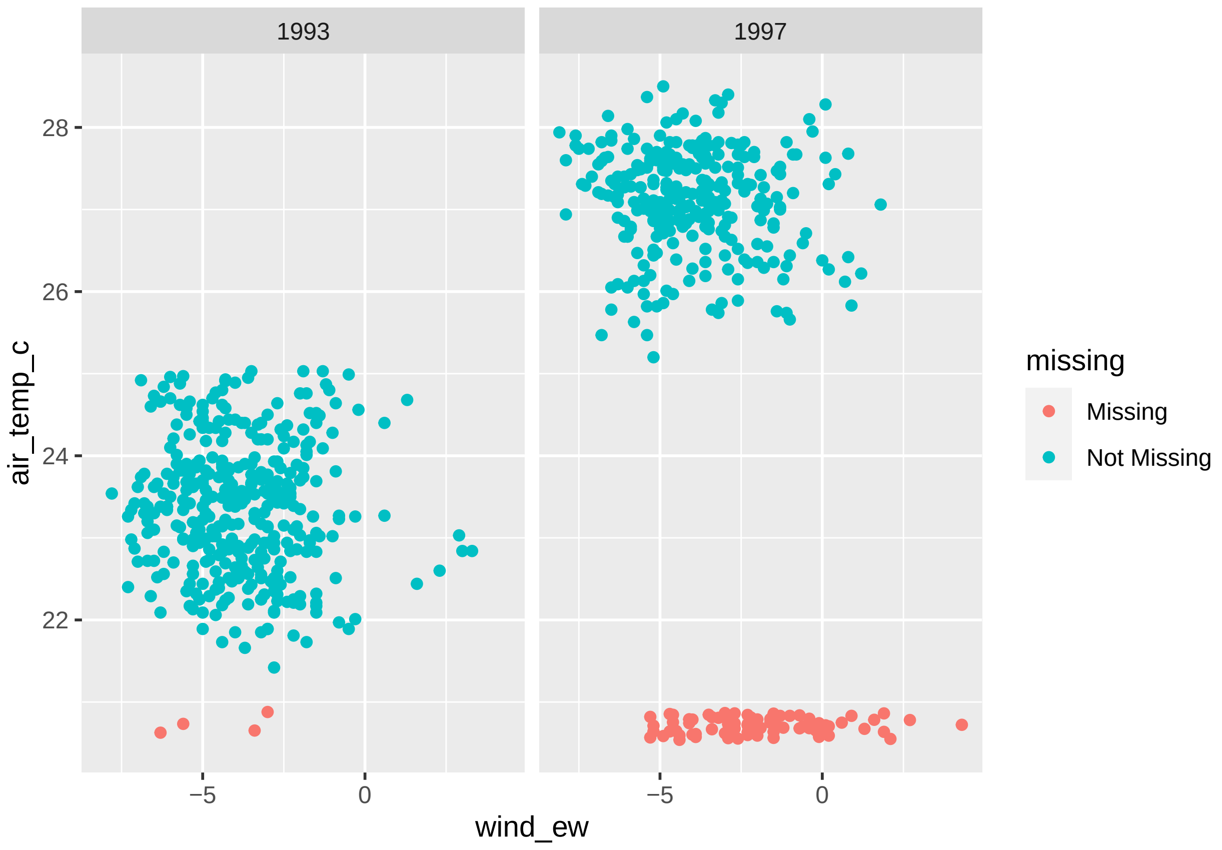
Using facets to explore missingness Because geom_miss_point() is a ggplot geom, you can use it with ggplot2 features like facetting.

This means we can rapidly explore the missingness and stay within the familiar bounds of ggplot2.

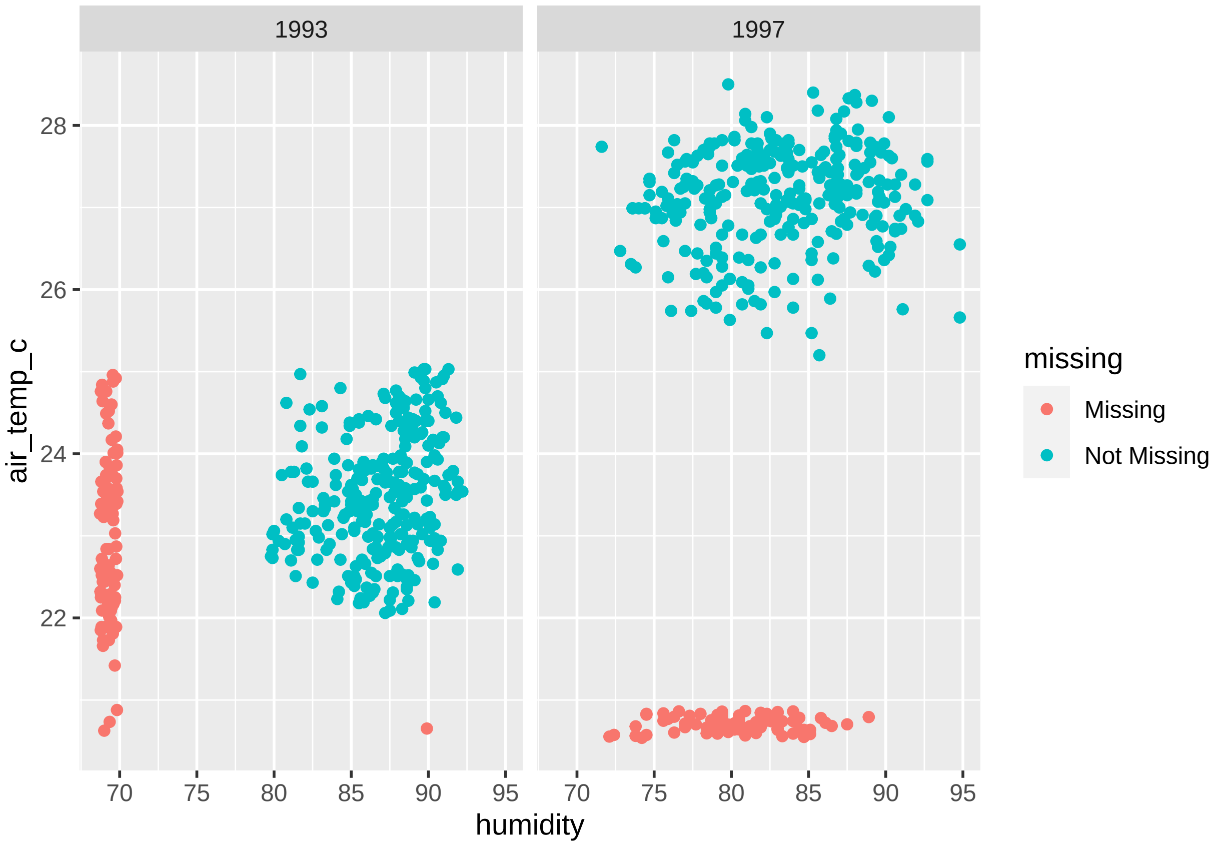
We will try the following:

Explore the missingness in wind and air temperature, and display the missingness using geom_miss_point(). Facet by year to explore this further.
Explore the missingness in humidity and air temperature, and display the missingness using geom_miss_point(). Facet by year to explore this further.

```
# Explore the missingness in wind and air temperature, and display the missingness using `geom_miss_poi
ggplot(oceanbuoys,
       aes(x = wind_ew,
           y = air_temp_c)) +
  geom_miss_point() +
  facet_wrap(~year)
```



```
# Explore the missingness in humidity and air temperature, and display the missingness using `geom_miss
ggplot(oceanbuoys,
       aes(x=humidity,
           y=air_temp_c)) +
  geom_miss_point() +
  facet_wrap(~year)
```



Faceting to explore missingness (multiple plots)

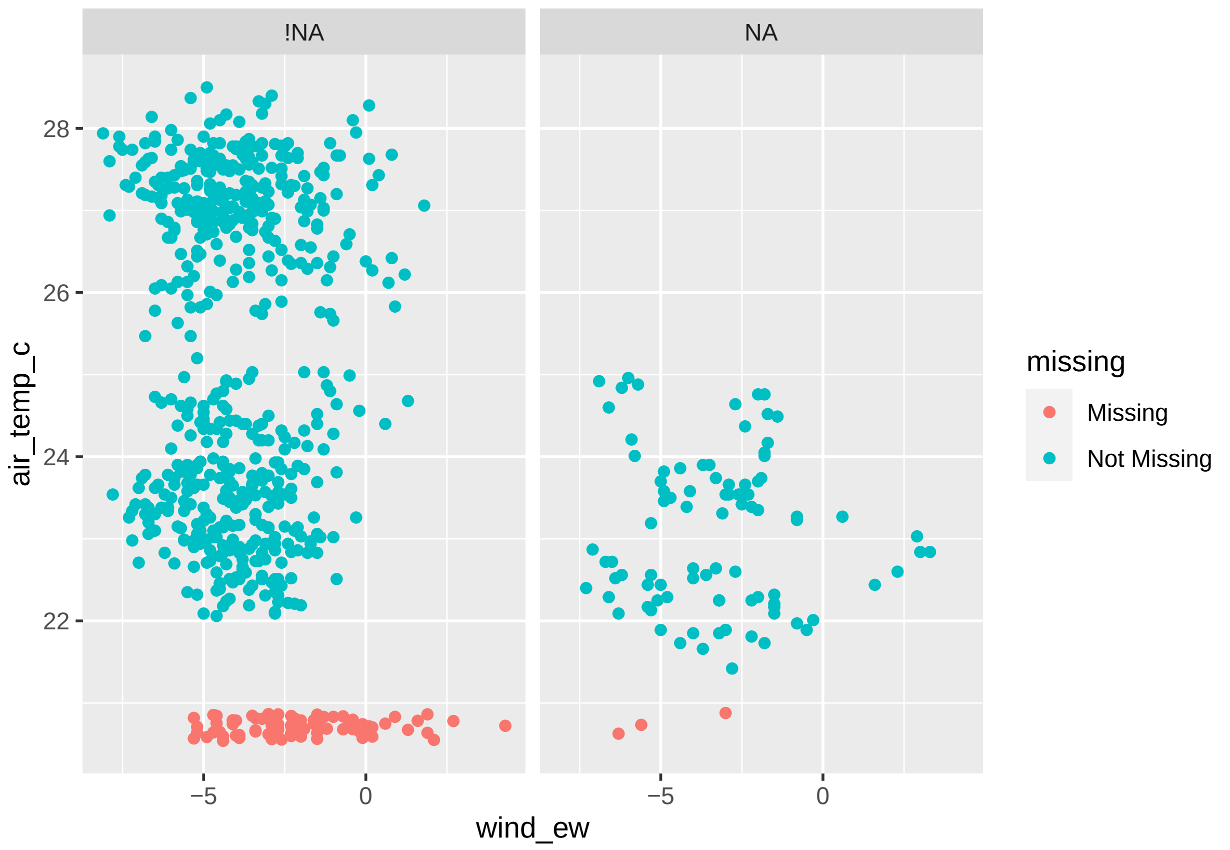
Another useful technique with geommisspoint() is to explore the missingness by creating multiple plots.

Just as we have done in the previous exercises, we can use the nabular data to help us create additional facetted plots.

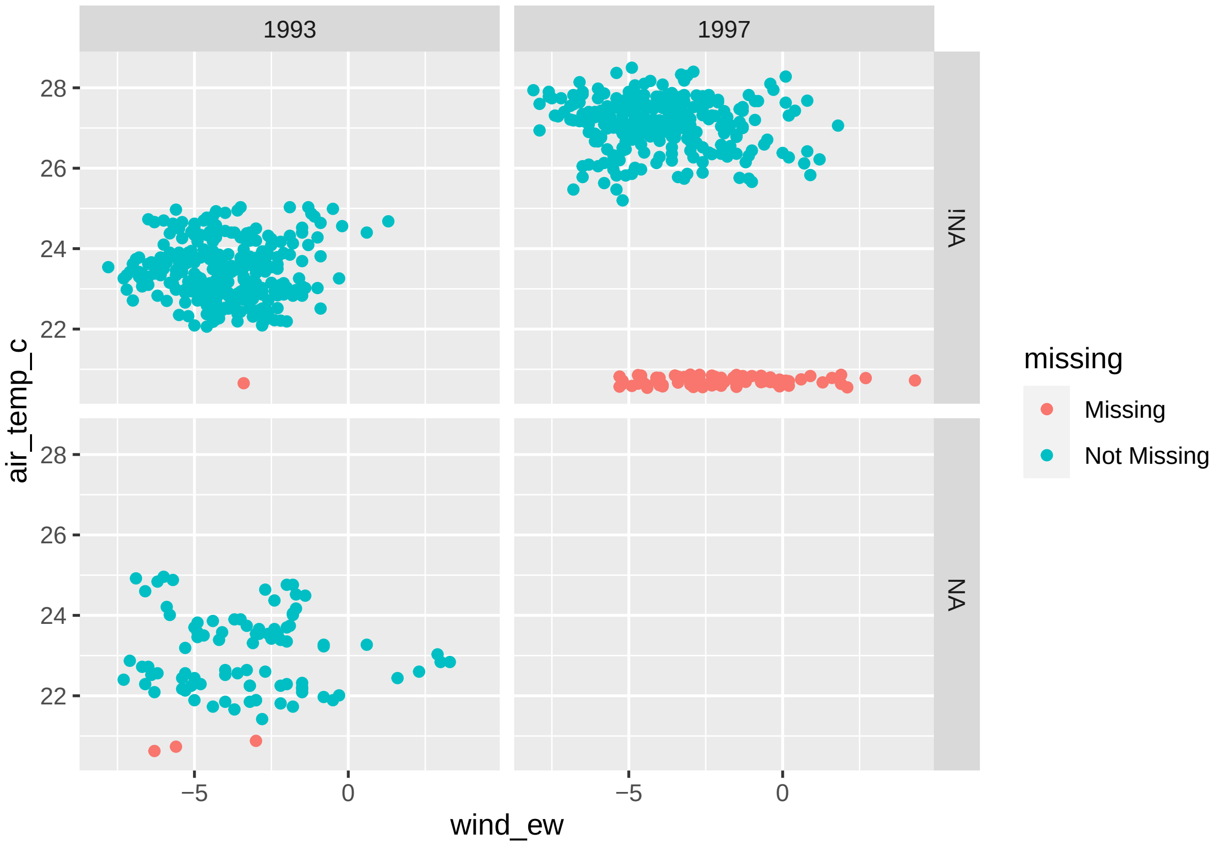
We can even create multiple facetted plots according to values in the data, such as year, and features of the data, such as missingness.

Use geom_miss_point() and facet_wrap() to explore how the missingness in wind_ew and air_temp_c is different for missingness of humidity. Use geom_miss_point() and facet_grid() to explore how the missingness in wind_ew and air_temp_c is different for missingness of humidity and by year.

```
# Use geom_miss_point() and facet_wrap to explore how the missingness in wind_ew and air_temp_c is diff
bind_shadow(oceanbuoys) %>%
  ggplot(aes(x = wind_ew,
             y = air_temp_c)) +
  geom_miss_point() +
  facet_wrap(~humidity_NA )
```

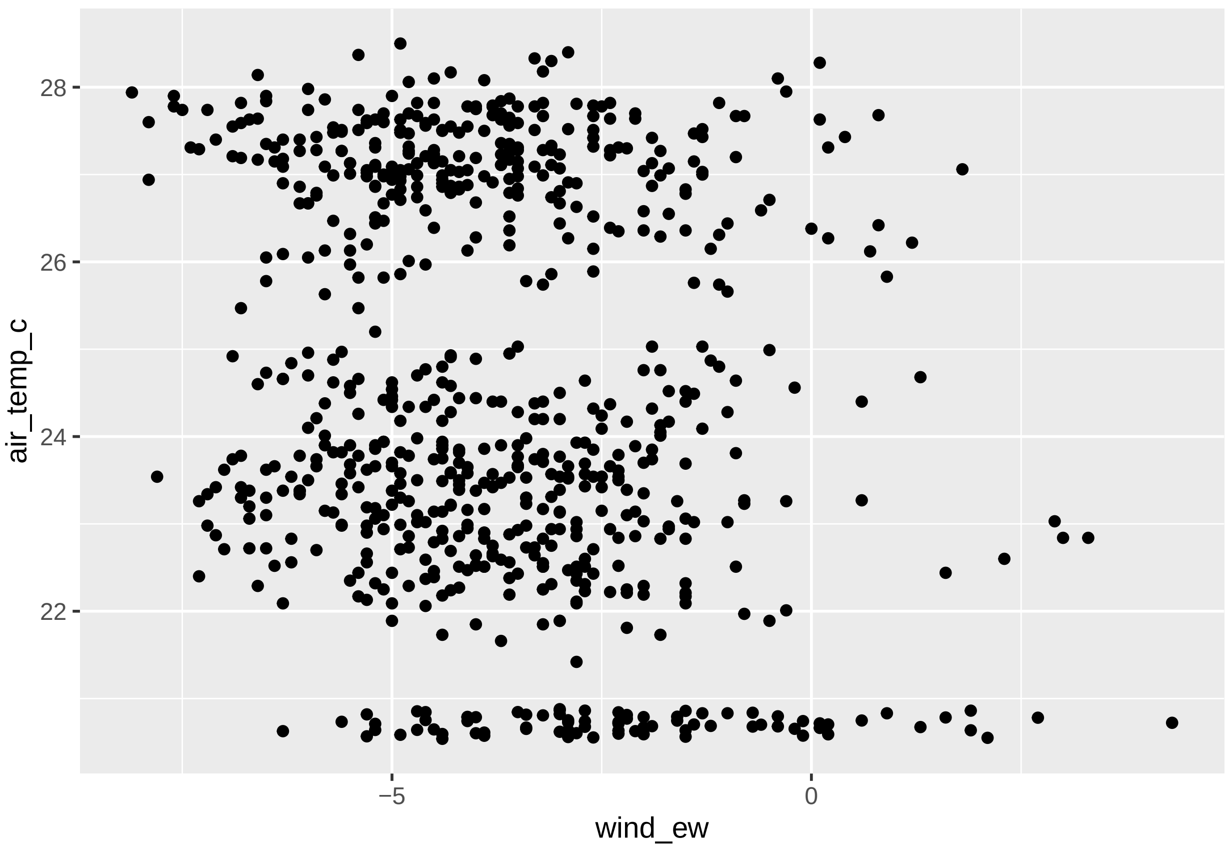


```
# Use geom_miss_point() and facet_grid to explore how the missingness in wind_ew and air_temp_c is diff
bind_shadow(oceanbuoys) %>%
  ggplot(aes(x = wind_ew,
             y = air_temp_c)) +
  geom_miss_point() +
  facet_grid(humidity_NA~year)
```



Performing and Tracking Imputation

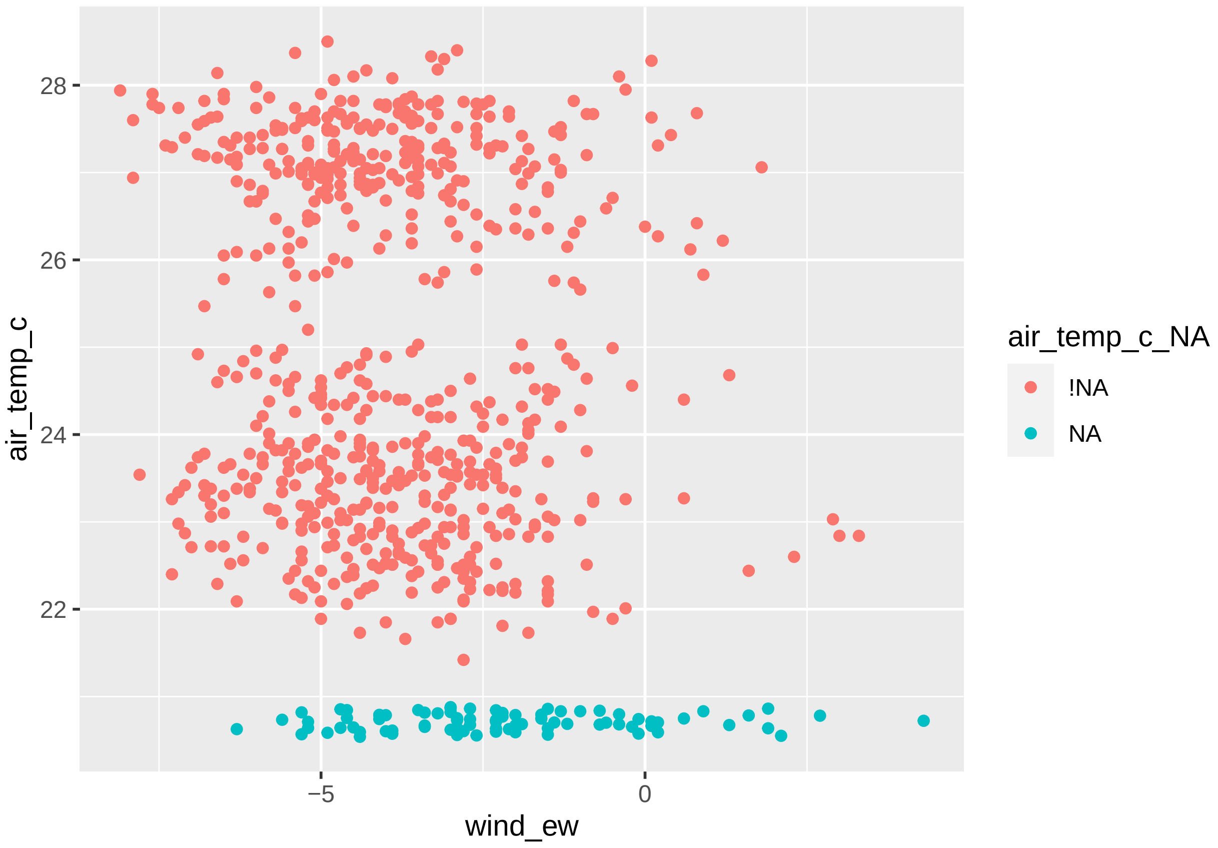
```r
# Impute the oceanbuoys data below the range using `impute_below`.
ocean_imp <- impute_below_all(oceanbuoys)

# Visualise the new missing values
ggplot(ocean_imp,
       aes(x = wind_ew, y = air_temp_c)) +
  geom_point()
```



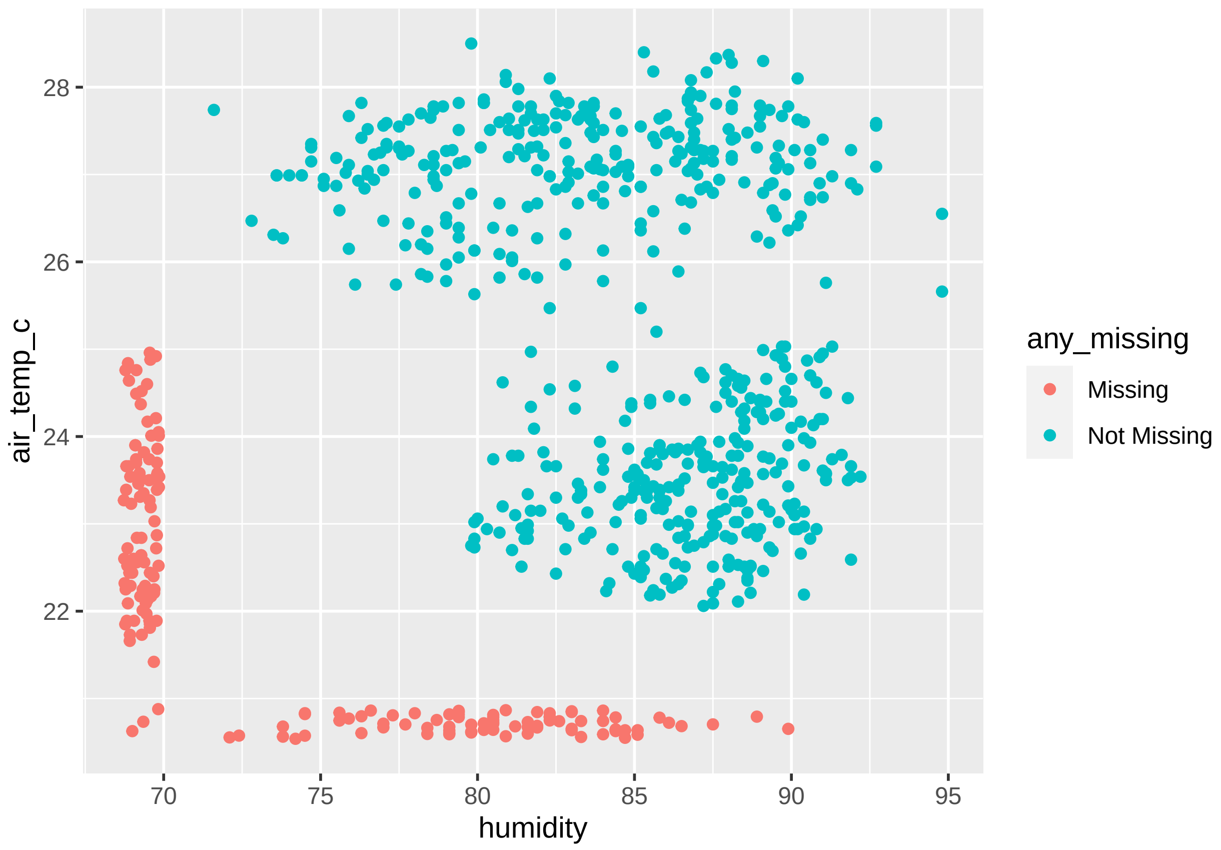
```r
# Impute and track data with `bind_shadow`, `impute_below_all`, and `add_label_shadow`
ocean_imp_track <- bind_shadow(oceanbuoys) %>%
  impute_below_all() %>%
  add_label_shadow()

# Look at the imputed values
ocean_imp_track
```

```
## # A tibble: 736 x 17
##     year latit~1 longi~2 sea_t~3 air_t~4 humid~5 wind_ew wind_ns year_NA latit~6
##    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl> <fct>   <fct>
##  1  1997       0    -110    27.6    27.1    79.6   -6.40    5.40 !NA     !NA
##  2  1997       0    -110    27.5    27.0    75.8   -5.30    5.30 !NA     !NA
##  3  1997       0    -110    27.6    27      76.5   -5.10    4.5  !NA     !NA
##  4  1997       0    -110    27.6    26.9    76.2   -4.90    2.5  !NA     !NA
##  5  1997       0    -110    27.6    26.8    76.4   -3.5     4.10 !NA     !NA
##  6  1997       0    -110    27.8    26.9    76.7   -4.40    1.60 !NA     !NA
##  7  1997       0    -110    28.0    27.0    76.5   -2       3.5  !NA     !NA
##  8  1997       0    -110    28.0    27.1    78.3   -3.70    4.5  !NA     !NA
##  9  1997       0    -110    28.0    27.2    78.6   -4.20    5    !NA     !NA
## 10  1997       0    -110    28.0    27.2    76.9   -3.60    3.5  !NA     !NA
## # ... with 726 more rows, 7 more variables: longitude_NA <fct>,
## #   sea_temp_c_NA <fct>, air_temp_c_NA <fct>, humidity_NA <fct>,
## #   wind_ew_NA <fct>, wind_ns_NA <fct>, any_missing <chr>, and abbreviated
## #   variable names 1: latitude, 2: longitude, 3: sea_temp_c, 4: air_temp_c,
## #   5: humidity, 6: latitude_NA
## # i Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

```
# Impute and track the missing values
ocean_imp_track <- bind_shadow(oceanbuoys) %>%
  impute_below_all() %>%
  add_label_shadow()

# Visualise the missingness in wind and air temperature, coloring missing air temp values with air_temp
ggplot(ocean_imp_track,
       aes(x = wind_ew, y = air_temp_c, color = air_temp_c_NA)) +
  geom_point()
```

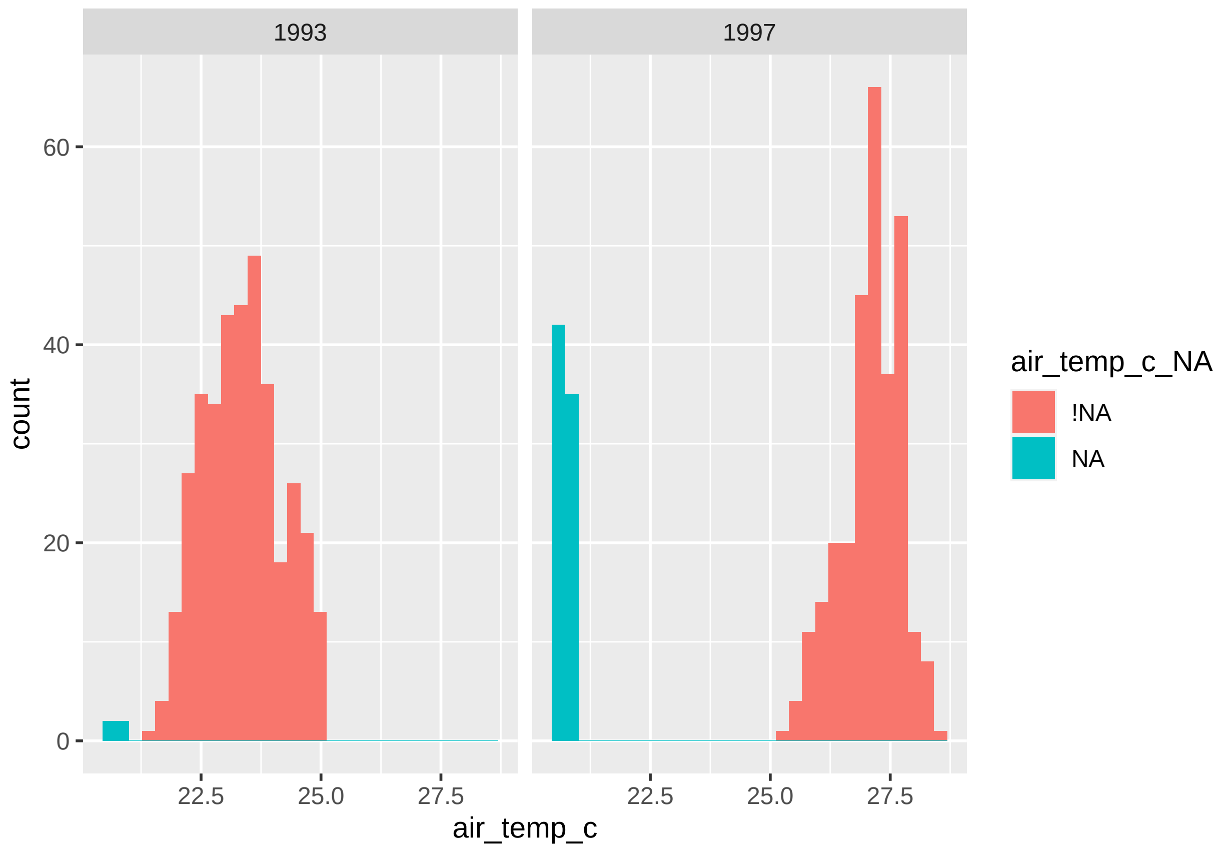


```
# Visualise humidity and air temp, coloring any missing cases using the variable any_missing
ggplot(ocean_imp_track,
       aes(x = humidity, y = air_temp_c, color = any_missing)) +
  geom_point()
```



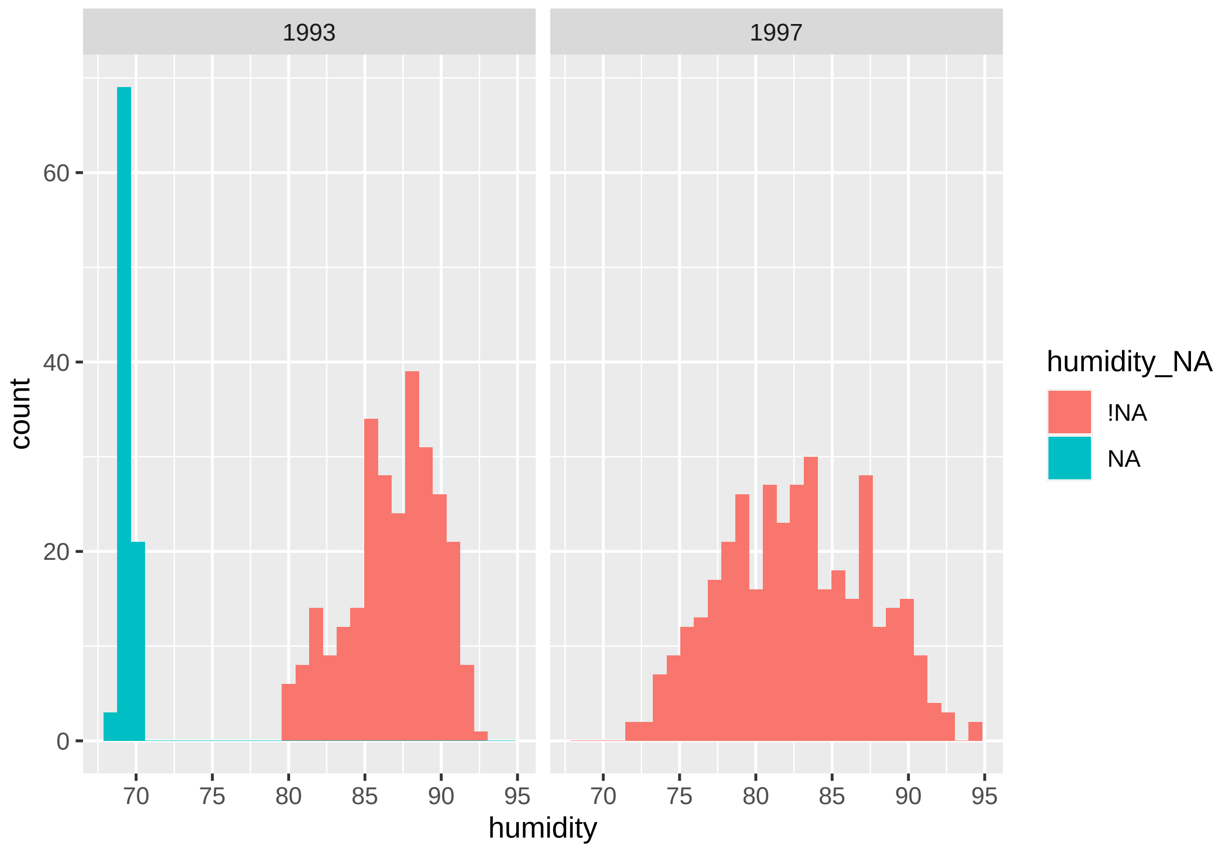
```r
# Explore the values of air_temp_c, visualising the amount of missings with `air_temp_c_NA`.
p <- ggplot(ocean_imp_track, aes(x = air_temp_c, fill = air_temp_c_NA)) +  geom_histogram()

# Expore the missings in humidity using humidity_NA
p2 <- ggplot(ocean_imp_track,  aes(x = humidity, fill = humidity_NA)) + geom_histogram()

# Explore the missings in air_temp_c according to year, using `facet_wrap(~year)`.
p + facet_wrap(~year)
```

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

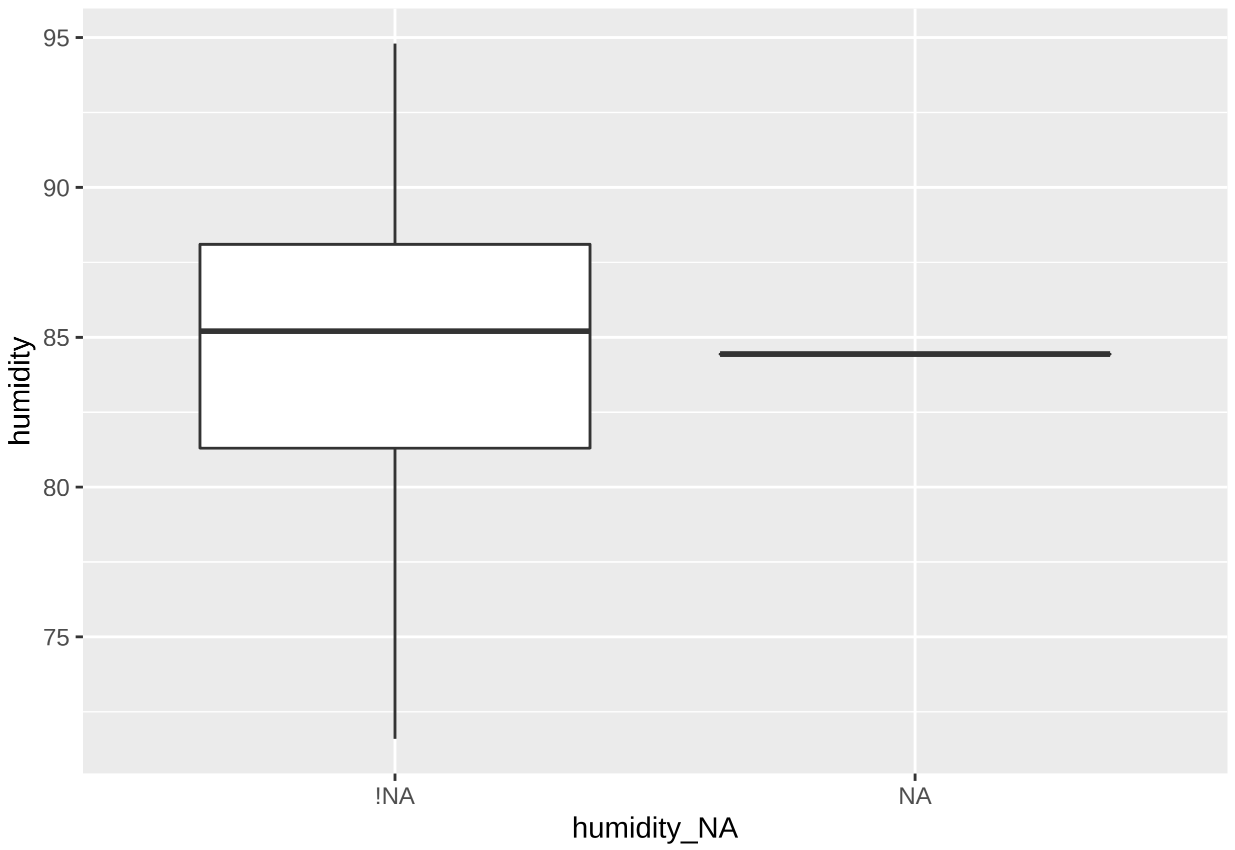


```r
# Explore the missings in humidity according to year, using `facet_wrap(~year)`.
p2 + facet_wrap(~year)
```

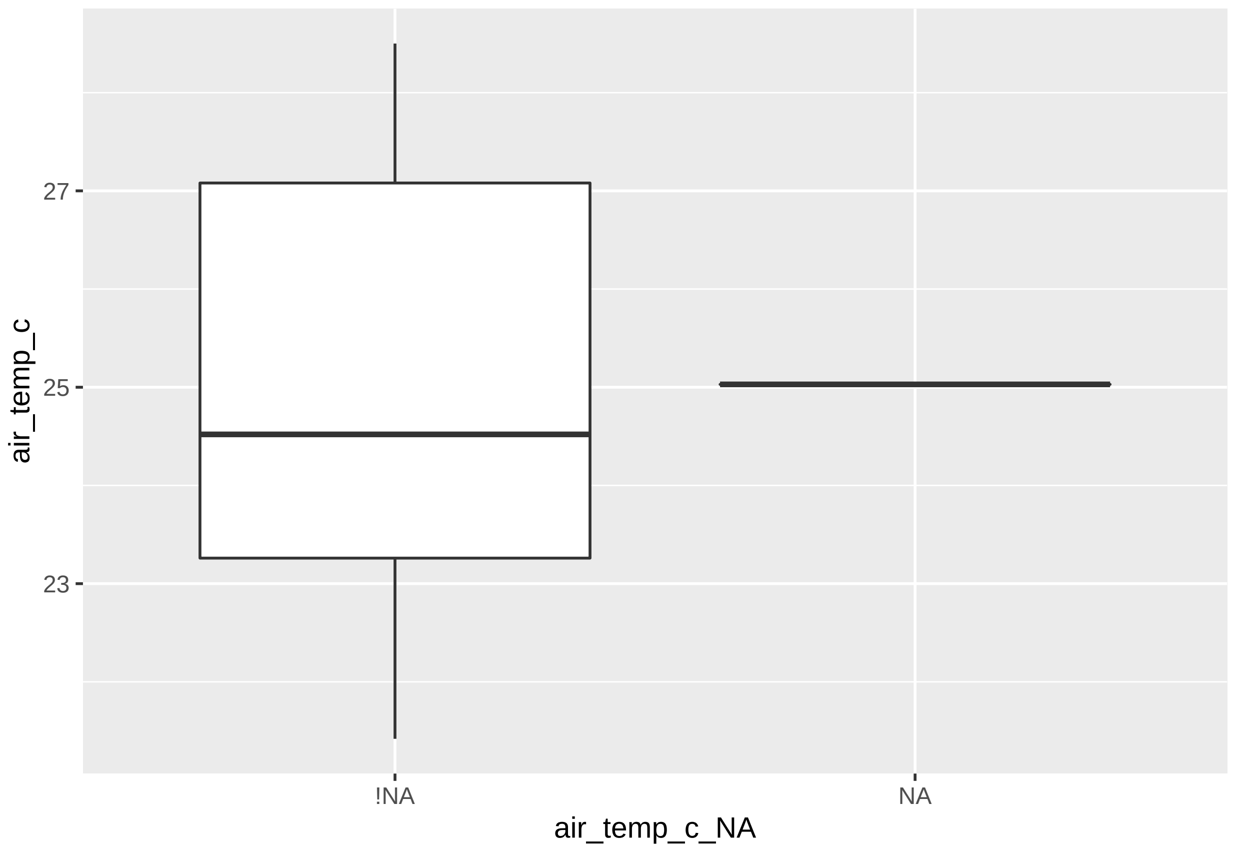
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



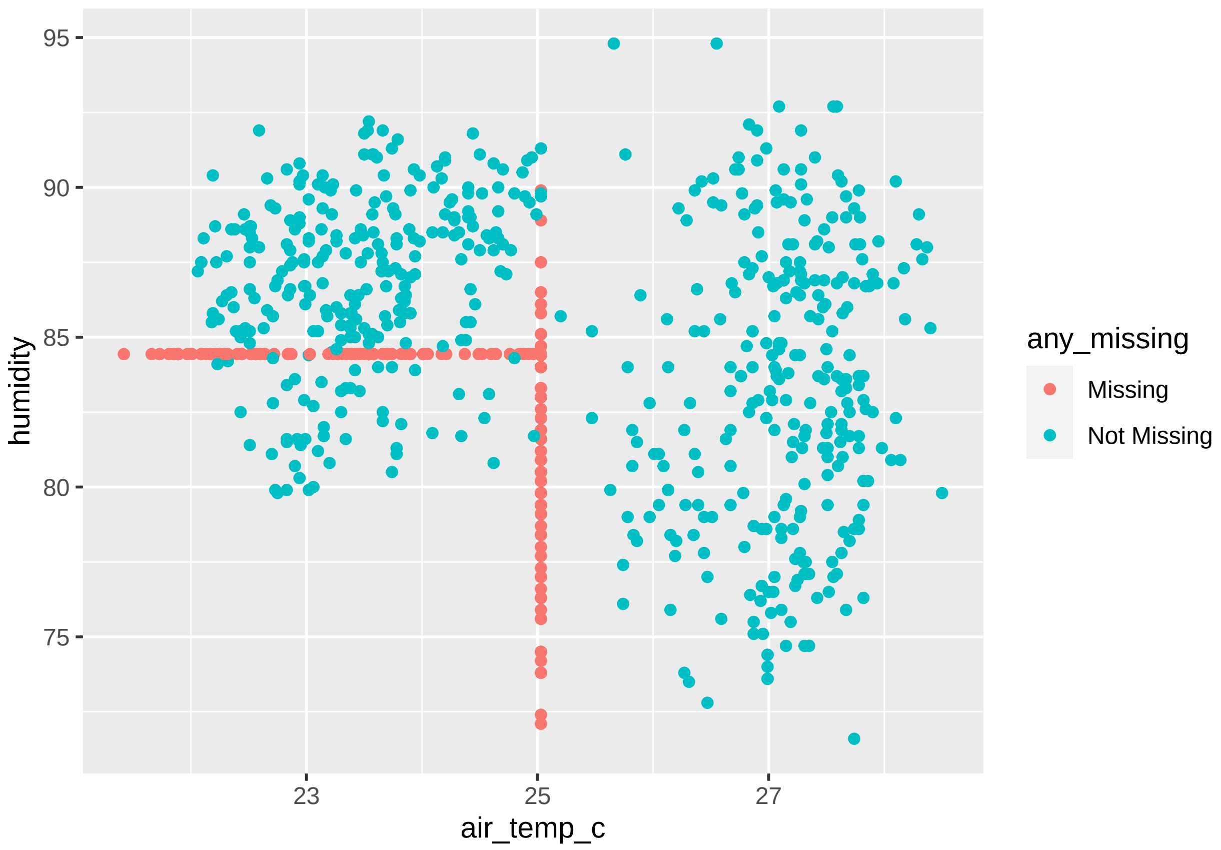
```r
# Impute the mean value and track the imputations
ocean_imp_mean <- bind_shadow(oceanbuoys) %>%
  impute_mean_all() %>%
  add_label_shadow()

# Explore the mean values in humidity in the imputed dataset
ggplot(ocean_imp_mean,
       aes(x = humidity_NA , y = humidity)) +
  geom_boxplot()
```



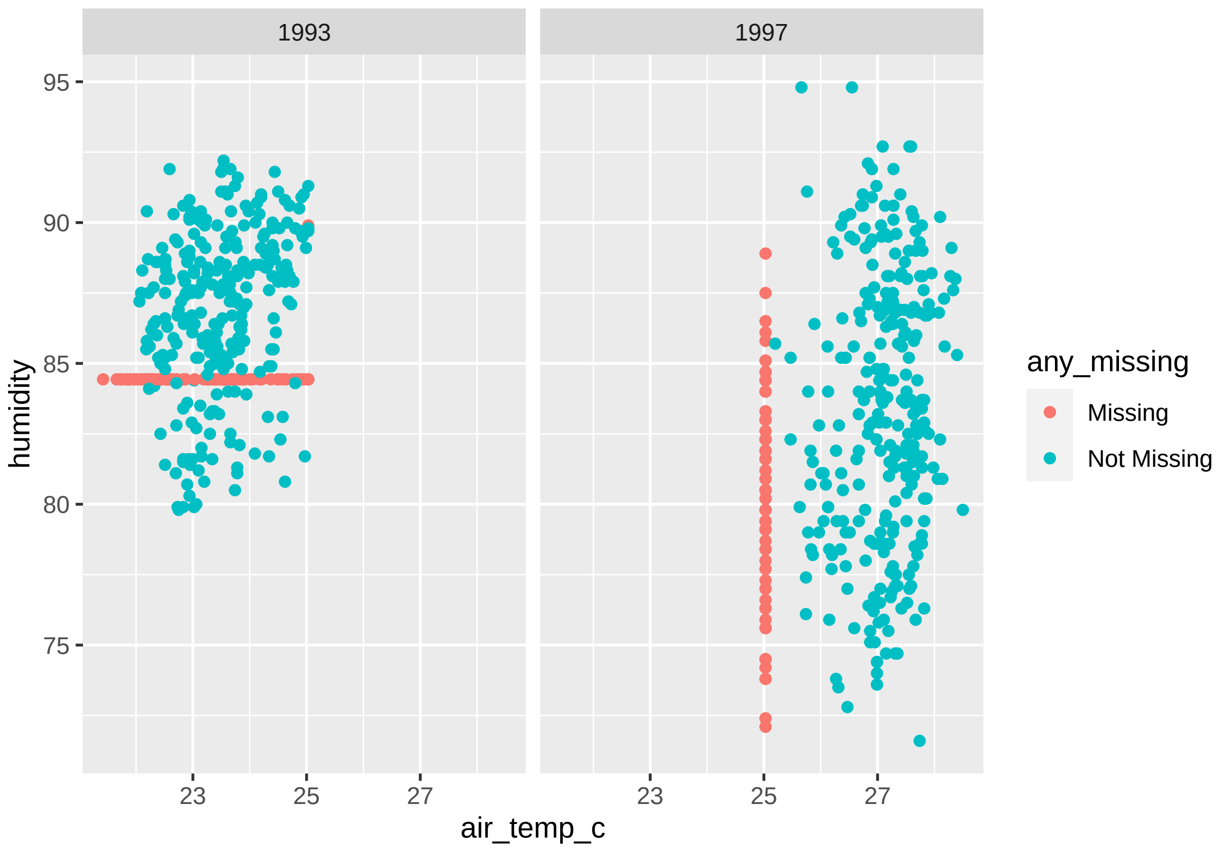
```
# Explore the values in air temperature in the imputed dataset
ggplot(ocean_imp_mean,
       aes(x = air_temp_c_NA, y = air_temp_c)) +
  geom_boxplot()
```



```
# Explore imputations in air temperature and humidity, coloring by the variable, any_missing
ggplot(ocean_imp_mean,
       aes(x = air_temp_c, y = humidity, color = any_missing)) +
  geom_point()
```



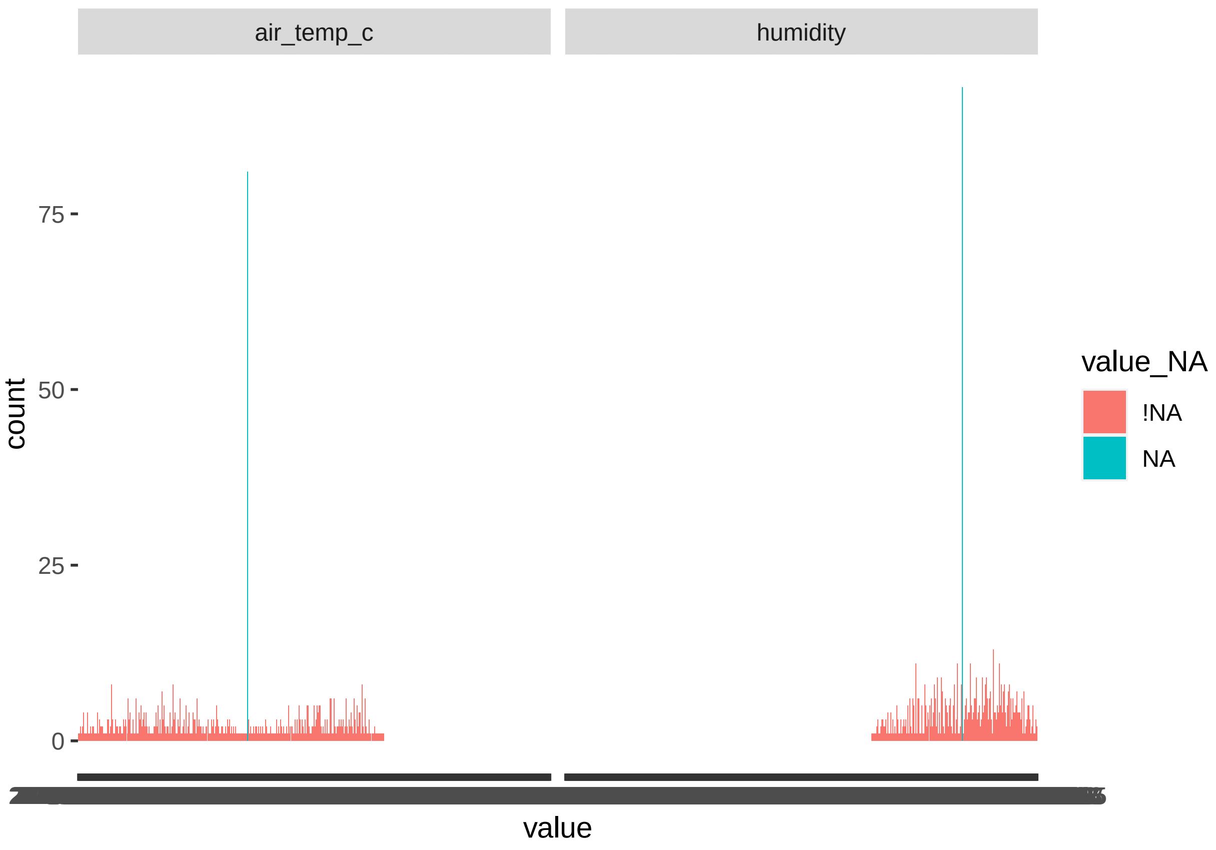
```
# Explore imputations in air temperature and humidity, coloring by the variable, any_missing, and facet
ggplot(ocean_imp_mean,
       aes(x = air_temp_c, y = humidity, color = any_missing)) +
  geom_point() +
  facet_wrap(~year)
```



```r
#Gather the imputed data
ocean_imp_mean_gather <- shadow_long(ocean_imp_mean,
                                     humidity,
                                     air_temp_c)
#Inspect the data
print(ocean_imp_mean_gather)
```

```
## # A tibble: 1,472 x 4
##    variable   value       variable_NA   value_NA
##    <chr>      <chr>       <chr>         <chr>
##  1 air_temp_c 27.14999962 air_temp_c_NA !NA
##  2 air_temp_c 27.02000046 air_temp_c_NA !NA
##  3 air_temp_c 27          air_temp_c_NA !NA
##  4 air_temp_c 26.93000031 air_temp_c_NA !NA
##  5 air_temp_c 26.84000015 air_temp_c_NA !NA
##  6 air_temp_c 26.94000053 air_temp_c_NA !NA
##  7 air_temp_c 27.04000092 air_temp_c_NA !NA
##  8 air_temp_c 27.11000061 air_temp_c_NA !NA
##  9 air_temp_c 27.20999908 air_temp_c_NA !NA
## 10 air_temp_c 27.25        air_temp_c_NA !NA
## # ... with 1,462 more rows
## # i Use `print(n = ...)` to see more rows
```

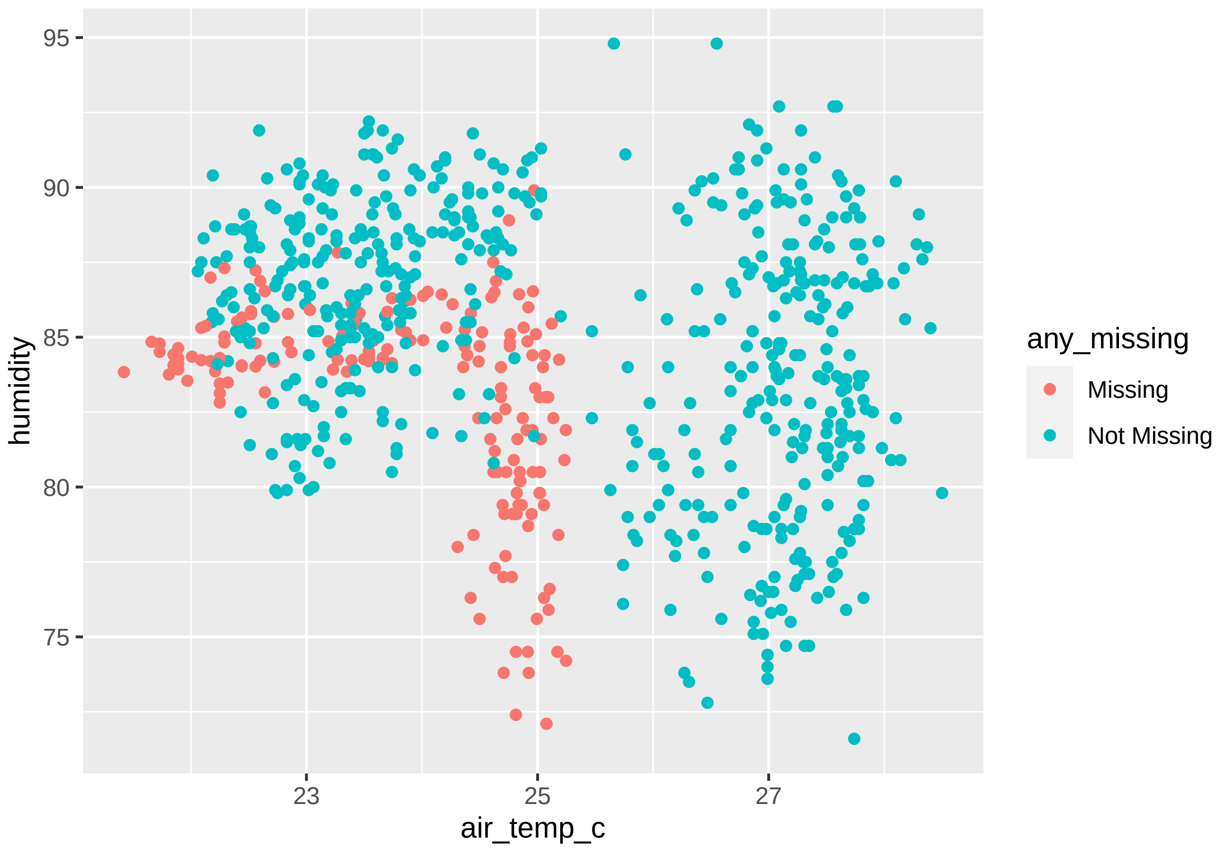
```r
#Explore the imputations in a histogram
ggplot(ocean_imp_mean_gather,
       aes(x = value, fill = value_NA)) +
  geom_bar()  +
  facet_wrap(~variable)
```



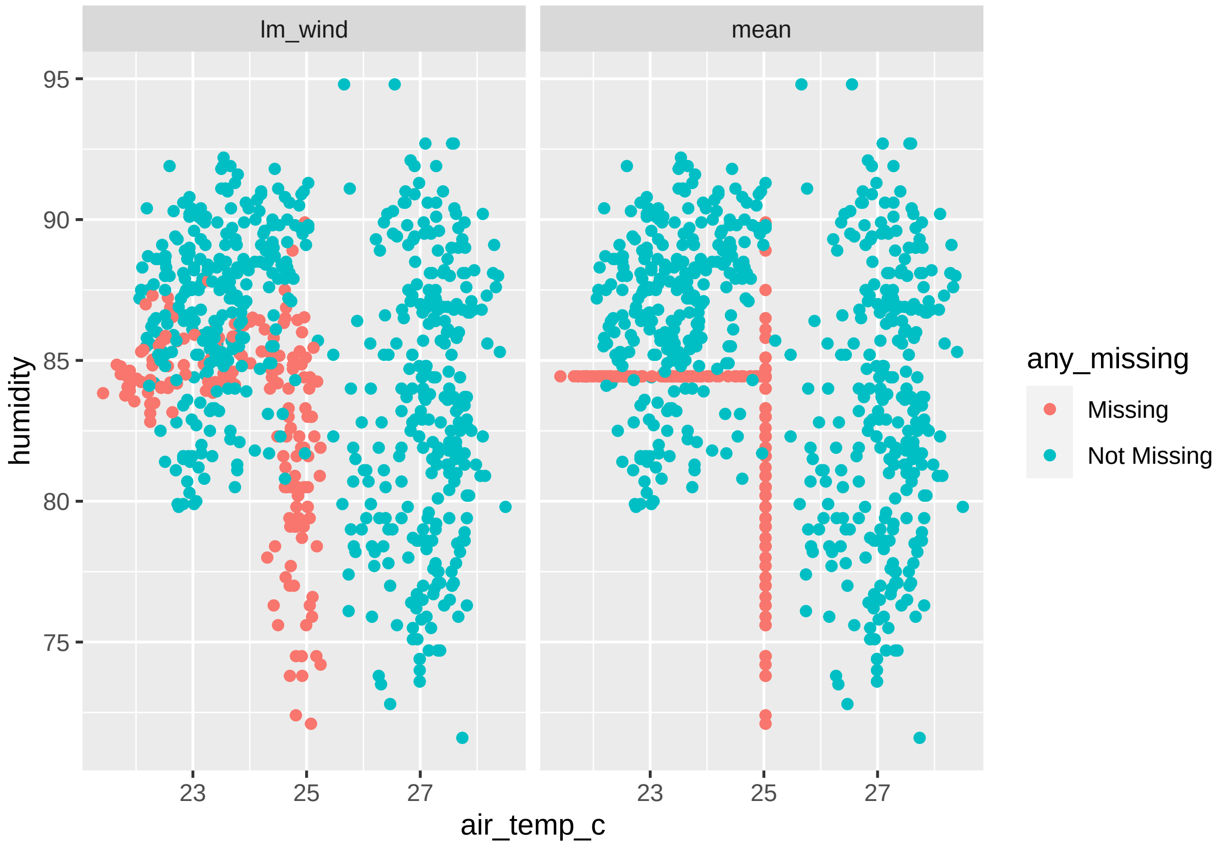
```
# Impute humidity and air temperature using wind_ew and wind_ns, and track missing values
library(simputation)
```

```
##
## Attaching package: 'simputation'

## The following object is masked from 'package:naniar':
##
##     impute_median
```

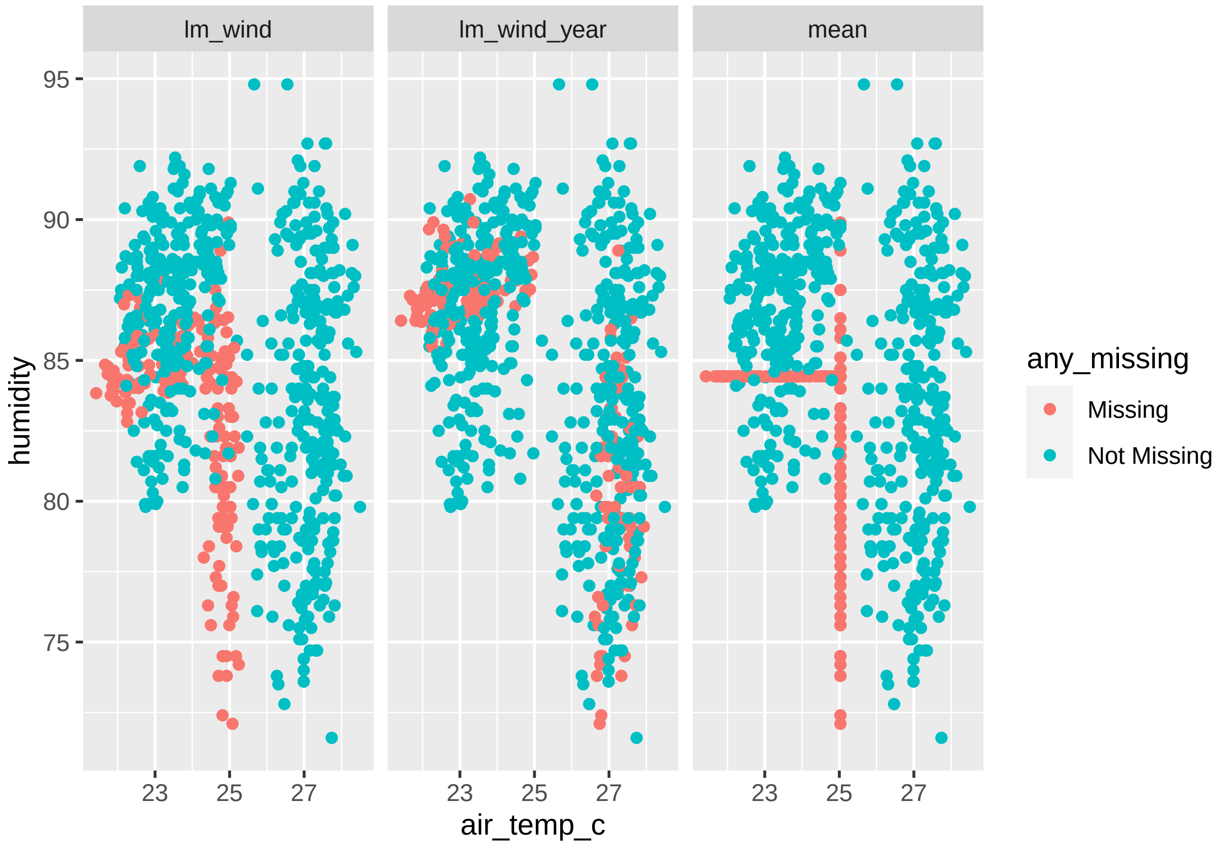
```
ocean_imp_lm_wind <- oceanbuoys %>%
    bind_shadow() %>%
    impute_lm(air_temp_c ~ wind_ew + wind_ns) %>%
    impute_lm(humidity ~ wind_ew + wind_ns) %>%
    add_label_shadow()

# Plot the imputed values for air_temp_c and humidity, colored by missingness
ggplot(ocean_imp_lm_wind,
       aes(x = air_temp_c, y = humidity, color = any_missing)) +
  geom_point()
```



```r
# Bind the models together
bound_models <- bind_rows(mean = ocean_imp_mean,
                          lm_wind = ocean_imp_lm_wind,
                          .id = "imp_model")

# Inspect the values of air_temp and humidity as a scatterplot
ggplot(bound_models,
       aes(x = air_temp_c,
           y = humidity,
           color = any_missing)) +
  geom_point() +
  facet_wrap(~imp_model)
```



```r
# Build a model adding year to the outcome
ocean_imp_lm_wind_year <- bind_shadow(oceanbuoys) %>%
  impute_lm(air_temp_c ~ wind_ew + wind_ns + year) %>%
  impute_lm(humidity ~ wind_ew + wind_ns + year) %>%
  add_label_shadow()

# Bind the mean, lm_wind, and lm_wind_year models together
bound_models <- bind_rows(mean = ocean_imp_mean,
                          lm_wind = ocean_imp_lm_wind,
                          lm_wind_year = ocean_imp_lm_wind_year,
                          .id = "imp_model")

# Explore air_temp and humidity, coloring by any missings, and faceting by imputation model
ggplot(bound_models, aes(x = air_temp_c, y = humidity, color = any_missing)) +
  geom_point() + facet_wrap(~imp_model)
```



```
# Create an imputed dataset using a linear models
ocean_imp_lm_all <- bind_shadow(oceanbuoys) %>%
  add_label_shadow() %>%
  impute_lm(sea_temp_c ~ wind_ew + wind_ns + year + latitude + longitude) %>%
  impute_lm(air_temp_c ~ wind_ew + wind_ns + year + latitude + longitude) %>%
  impute_lm(humidity ~ wind_ew + wind_ns + year + latitude + longitude)

# Bind the datasets
bound_models <- bind_rows(cc = ocean_cc,
                          imp_lm_wind = ocean_imp_lm_wind,
                          imp_lm_all = ocean_imp_lm_all,
                          .id = "imp_model")

# Look at the models
bound_models
```

```
# Create the model summary for each dataset
model_summary <- bound_models %>%
  group_by(imp_model) %>%
  nest() %>%
  mutate(mod = map(data, ~lm(sea_temp_c ~ air_temp_c + humidity + year, data = .)),
         res = map(mod, residuals),
         pred = map(mod, predict),
         tidy = map(mod, tidy))

# Explore the coefficients in the model
model_summary %>%
    select(imp_model,tidy) %>%
    unnest()
best_model <- "imp_lm_all"
```