Tutorial to Visualize Missing Data

Edgar G. Dorsey Trevino

11/4/2020

Introduction

In this tutorial we are going to learn about how to visualize and handle missing data. Missing data is everywhere, and the best way to handle it is to avoid having missing data. However, real world is cruel, and usually we have to bear with missing data. Luckily, R is a very powerful software that has several packages that were design to deal with missing data, and hopefully, at the end of this tutorial you will have solid basis to deal with unwanted missing data.

I will divide this R tutorial in two main section: visualization and imputation. Every section will include an R tutorial and an interpretation of the output. It is expected that for the reader to be familiar with the R package 'tidyverse' and the so-called tidy approach, in which every row represents an observation and every column represents a variable.

Data Visualization

Summarizing Missing Data

At first glance, it may sound counterintuitive the phrasing "missing data visualization" . . . I mean, if the data is missing, how can we look at it? I guess we could manually browse for 'NAs' on the data set but with gigantic datasets this task may be a bit onerous. An easier way to explore missing data is using the R package "naniar"; developed by Nicholas Tierney. He believed that missing data can be think of data from another dimension. Inspired by this analogy, Tierney labeled the package 'naniar' as a reference to the move "Narnia" that takes place in another imaginary dimension.

Before starting, let's remember some concepts and differentiate between vectors and data frames.

- Vector: Is a collection of data represented in R with the function 'c()'

```
x <- c('text', 'text2', 'text3', 'text4')
print(x)
```

```
## [1] "text"  "text2" "text3" "text4"
```

- Data frame: Imagine collecting a lot of vectors and arranging them in columns. THIS IS A DATAFRAME!

```r
dataframe <- data.frame(
  var1 = c('text', 'text2', 'text3', 'text4'), #This is a vector
  var2 = c(1, 2, 3, 4) #This is a vector
)
print(dataframe) #When joined together it becomes a data frame
```

```
##    var1 var2
## 1  text    1
## 2 text2    2
## 3 text3    3
## 4 text4    4
```

Awesome! You are now an expert differentiating vectors and data frames. Let's dive into the topic and code some more!

Let's start by uploading the packages 'naniar' and 'tidyverse' and explore some of its features.

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

x <- c('text', 'text2', NA, 'NOT NA')

any_na(x)
```

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

```r
are_na(x)
```

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

The two function above work only on vectors, not on data frames. The function 'any_na' will return a logical (TRUE or FALSE) to indicate if there is any missing value in the vector as whole. Conversely, the function 'are_na' will return a logical element-wise indicating if the value is missing or not.

The two function above are very useful to explore vectors but most of the times we will be dealing with data frames. Let us explore some function to explore missing data in data frames.

```r
data <- read.table('weight_data.txt', header = T) %>% as_tibble()

dim(data)
```

```
## [1] 100   3
```

```r
names(data)
```

```
## [1] "ID"     "weight" "height"
```

```r
print(data)
```

```
## # A tibble: 100 x 3
##       ID weight height
##    <int>  <dbl>  <dbl>
##  1     1   95.3 NA
##  2     2   73.4 NA
##  3     3   60.2  2.77
##  4     4   63.2 NA
##  5     5   92.3  1.74
##  6     6   73.4  0.222
##  7     7   87.9  0.702
##  8     8  108.   1.92
##  9     9   NA   NA
## 10    10   79.9  1.20
## # ... with 90 more rows
```

```
#Counts the total number of missing values in 'data'
n_miss(data)
```

```
## [1] 30
```

```
n_miss(data$weight)
```

```
## [1] 15
```

```
#Counts the total number of complete values in 'data'
n_complete(data)
```

```
## [1] 270
```

```
n_complete(data$weight)
```

```
## [1] 85
```

```
#Counts the proportion of complete and missing values in 'data'
prop_miss(data)
```

```
## [1] 0.1
```

```
prop_complete(data$weight)
```

```
## [1] 0.85
```

Our data frame consists of 100 patients and 3 variables (ID, weight, and height). The functions above can be used to detect either missing or complete values in the whole data frame or in specific variable. For instance, when using the function 'n_miss()', the output tells us that in the whole data frame there is a total of 30 missing values, of which 15 missing values are within the variable 'weight'.

You now know a lot about your data, kudos! However, imagine having a data set of 30 variables. . . Gosh! Exploring every variable would be very time consuming. Let's see how we can deal with this problem in a more efficient way.

The following functions provide you with a summary of missing data of either all variables or rows. For the following examples we will use the data set already installed in R named 'airquality'.

```r
airquality <- as_tibble(airquality)
print(airquality)
```

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

```r
miss_var_summary(airquality) #column-wise
```

```
## # 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
```

```r
miss_case_summary(airquality) #row-wise
```

```
## # 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
```

Both functions provide a data frame with the first row being either the variable name or the row number. The column 'n_miss' represents the number of values that are missing, and the column 'pct_miss' represents the percentage of the previous column. Therefore, the output of the 'miss_var_summary' function, indicates that the variable 'Ozone' has 37 missing values and that it represents 24.2% of the total number of values

(153) in the variable Ozone (37 / 153 = 0.242). Conversely, the output of the function 'miss_case_summary' tells us that the row number 5 has 2 missing values that represents 33.3% of the total number of values (3) in that column (2 / 3 = 0.333).

We can apply the same logic as before for the following functions that help us summarize missingness of variables or rows grouped by a given variable.

```
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
```

```
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
```

Thus far have learned what are missing values, how to count them, and how to summarize them. Now, let's focus on some of the built-in visualizations that come with 'naniar' to explore our data visually.

Visualizing Summary of Missing Data

Visualizing our missing data can provide rapid insights about our data. The package 'naniar' provides a friendly family of missing data visualizations. Each visualization will correspond to a summary statistic seen before. And although you can always summarize missingness using the functions above, this task becomes repetitive, cumbersome, and your code starts getting dirty very quickly when analyzing multiple variables.

- Lesson Overview
  - How to get a bird's view of the data
  - How to look at missing values of variables and cases (rows)
  - How to visualize missingness across groups in the data

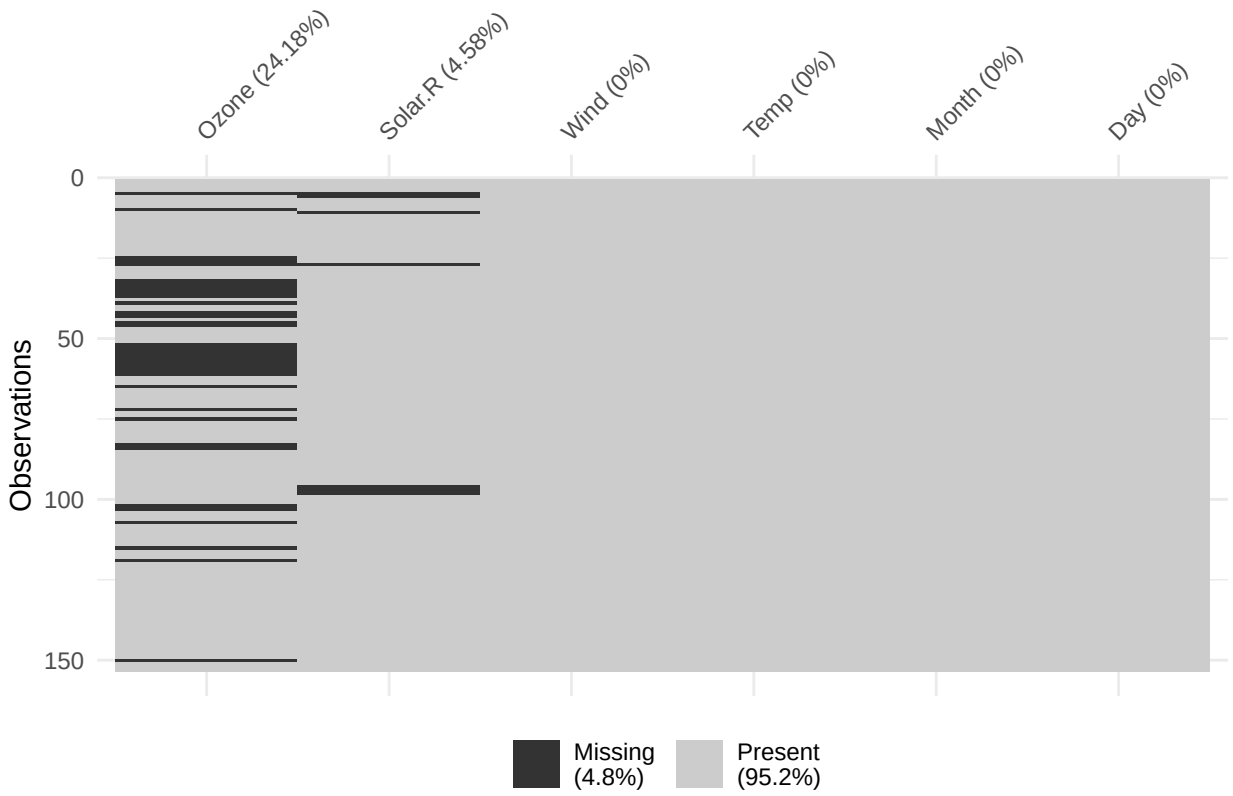
For this section, we will need to upload the package 'visdat'.

```
library(visdat)
```

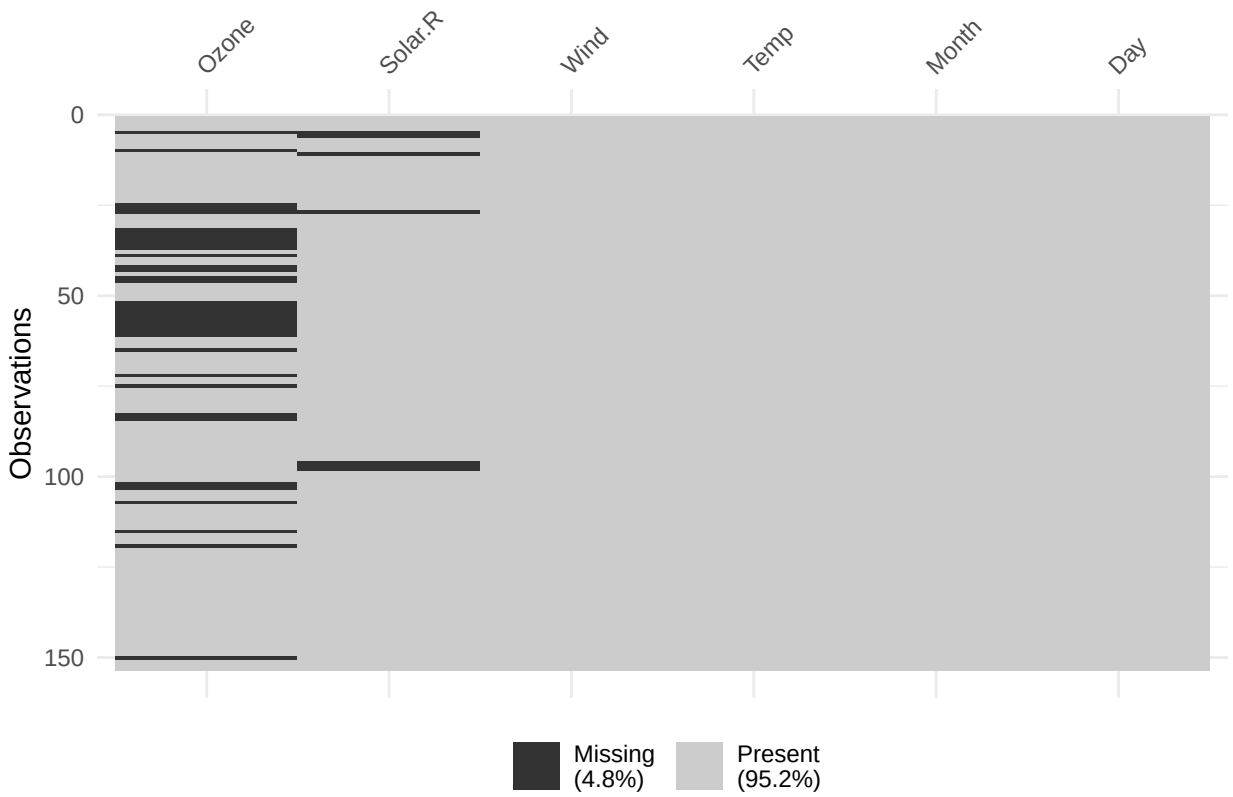
How to get a bird's view of the data

The following function provides a heat map that allows you to quickly look at the missing and complete values of your data. At the top of the plot you can see a summary statistic indicating the percentage of missingness of a particular variable; this can be turned off with the argument 'show_perc_col = F'.

```
vis_miss(airquality)
```

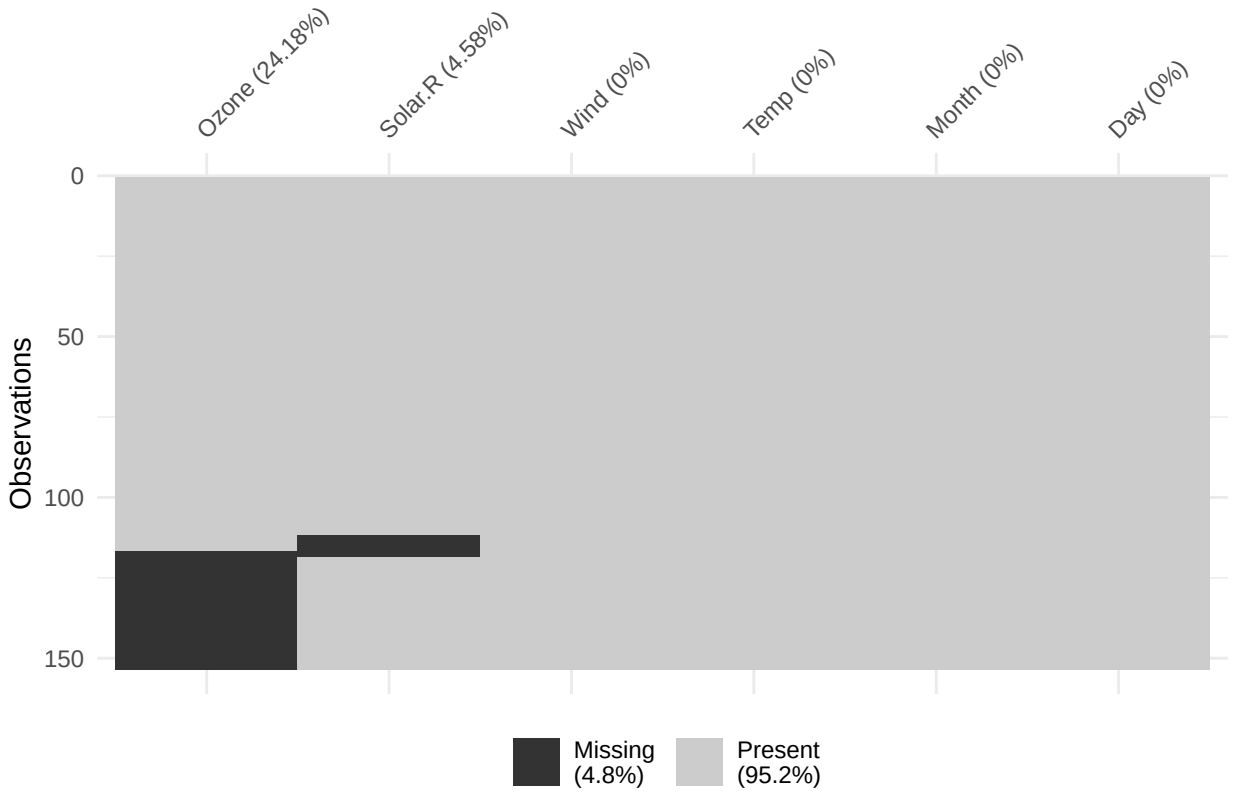


```
vis_miss(airquality, show_perc_col = F)
```



You can also cluster the data so you can identify patterns of missingness occurring in different variables.

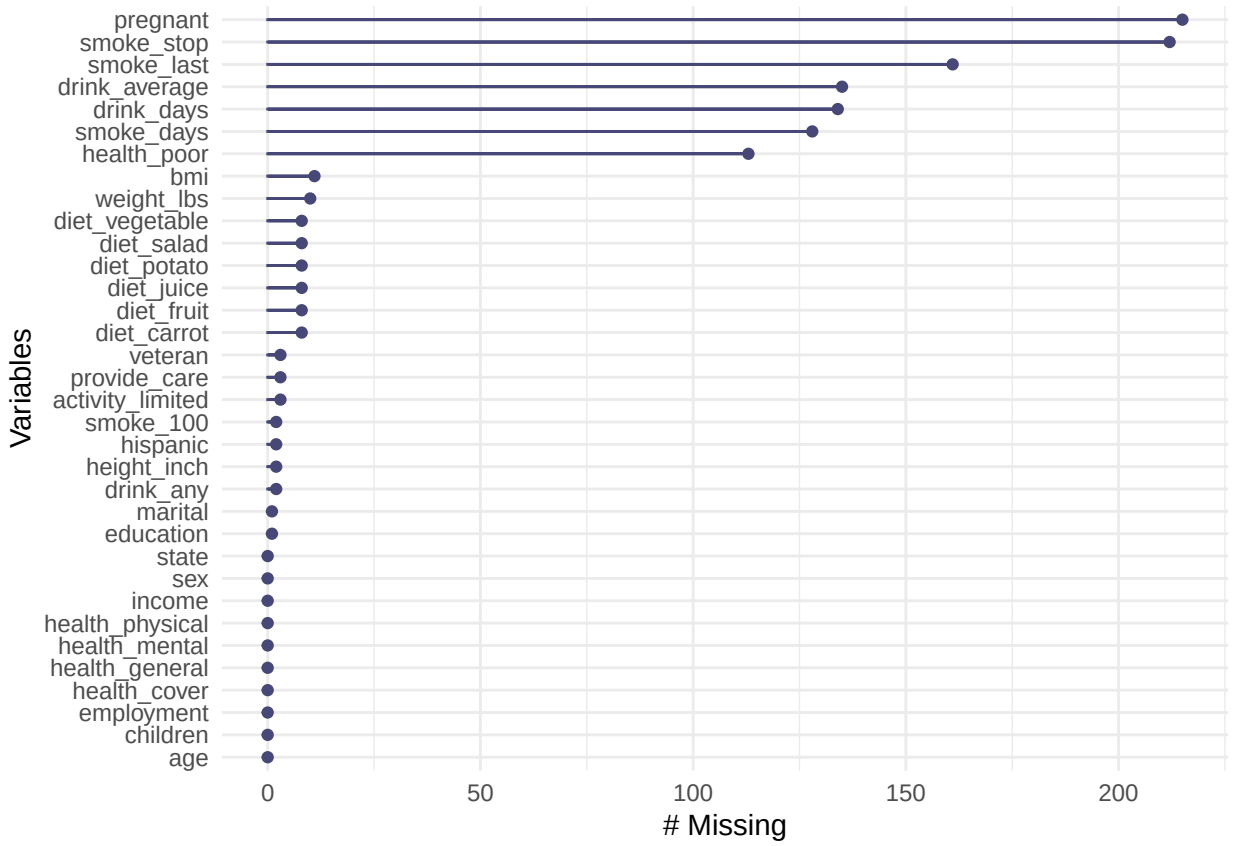
```
vis_miss(airquality, cluster = T)
```



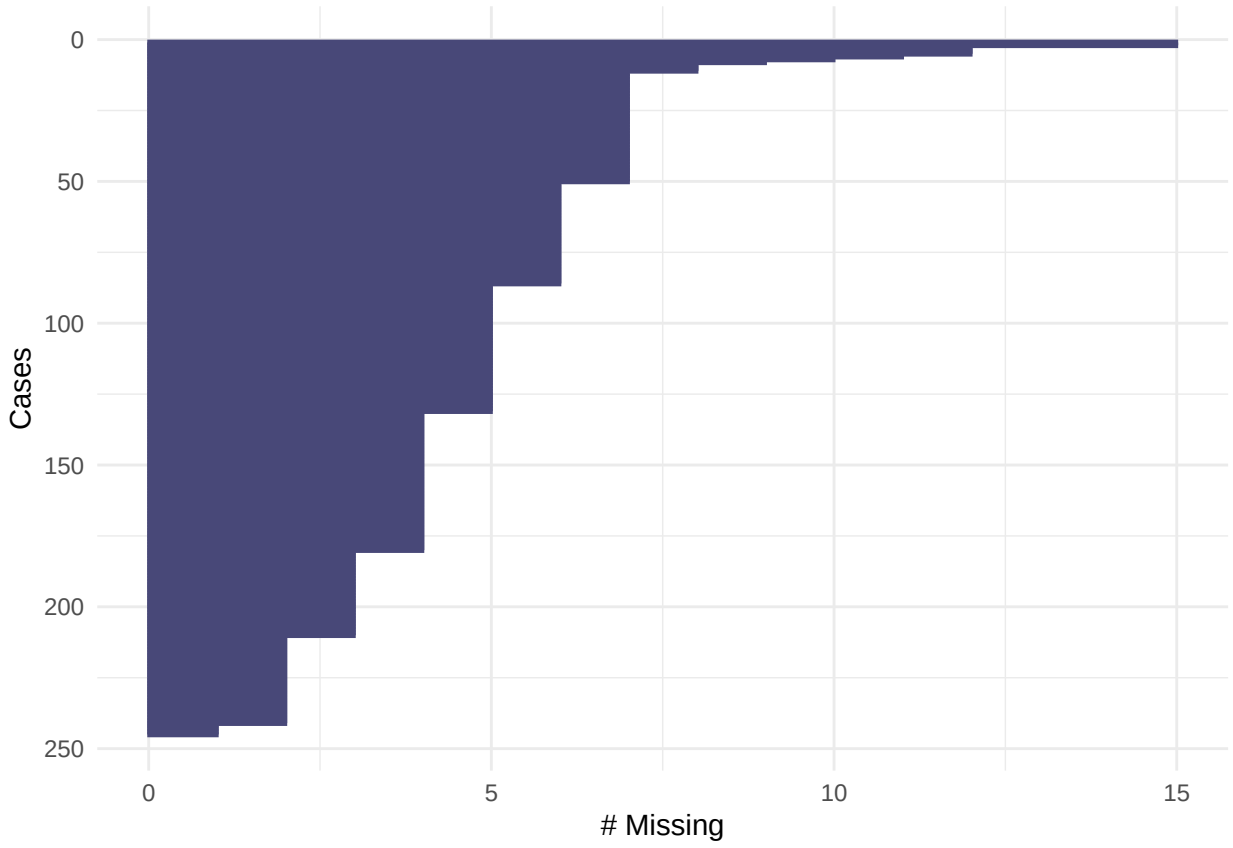
How to look at missing values of variables and cases (rows)

The following functions provide a visualization of missingness of variables and cases. You will note that y-axis is ordered based on the amount of missingness

```
gg_miss_var(riskfactors)
```




```
gg_miss_case(riskfactors) #You can turn off the ordered of the y-axis
```

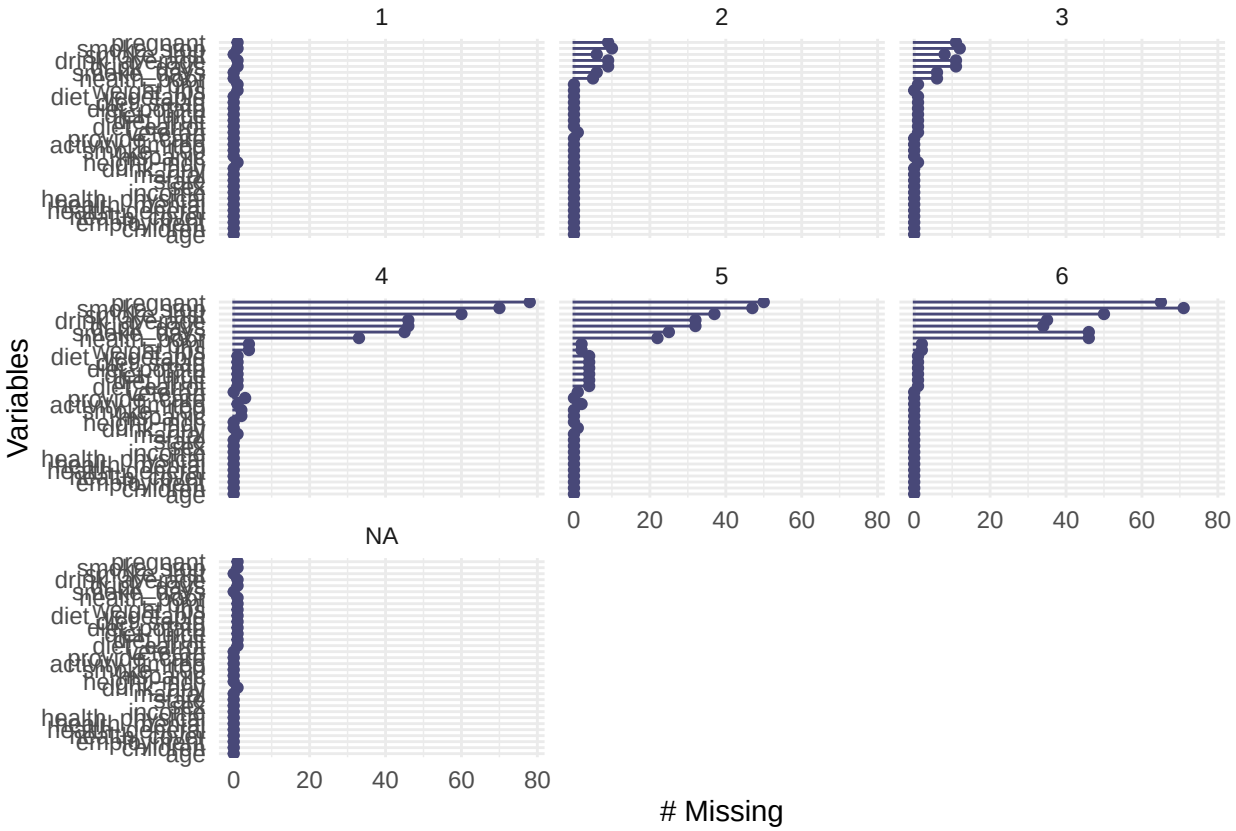



```
#with the argument order_cases = F
```

How to visualize missingness across groups in the data

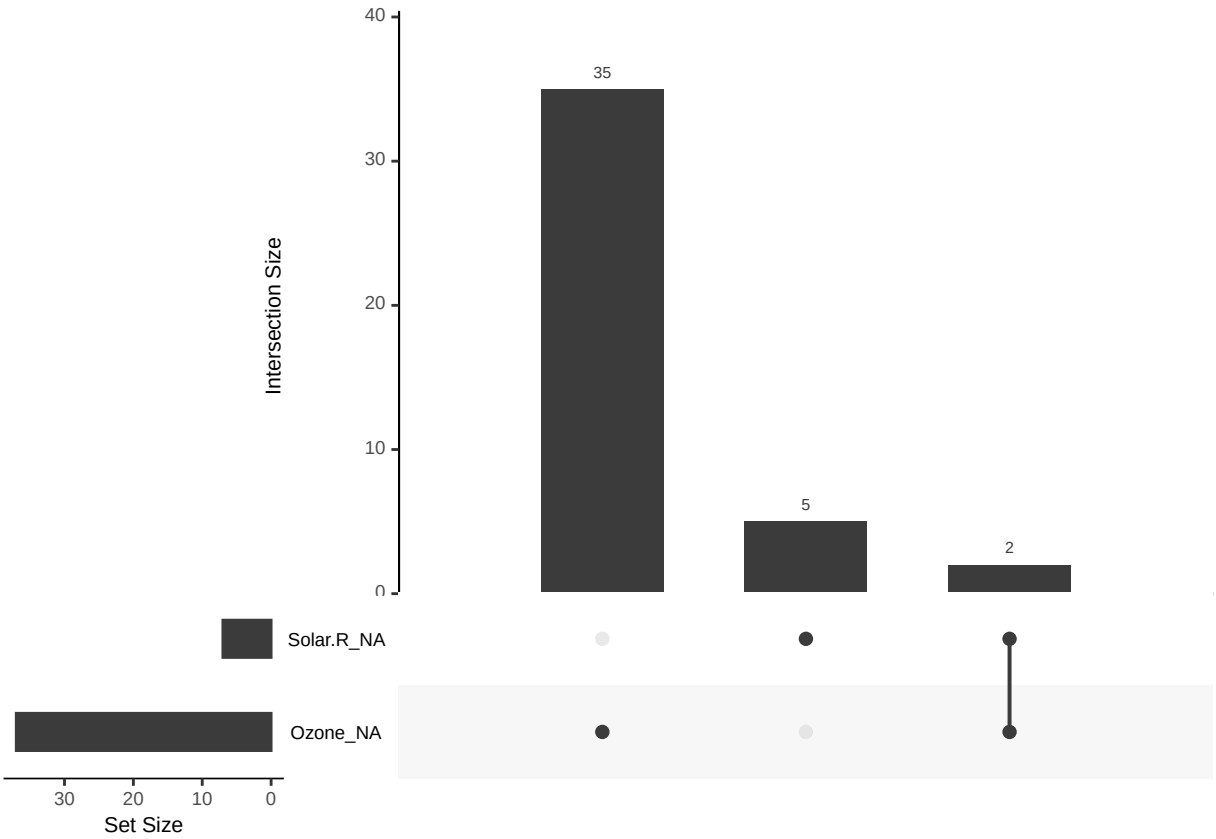
Sometimes, for categorical variables, missingness can occur more in a given level than the rest of the levels, suggesting a pattern. To visualize the missingness within each level of a categorical variable you use the argument 'facet' of the gg_miss_var function and specify the variable you want to facet.

```
gg_miss_var(riskfactors, facet = education)
```



Moreover, missing values may occur in patterns (i.e., missing value causes another missing value.) To detect this, otherwise hidden pattern, we can use the so-called upset plot.

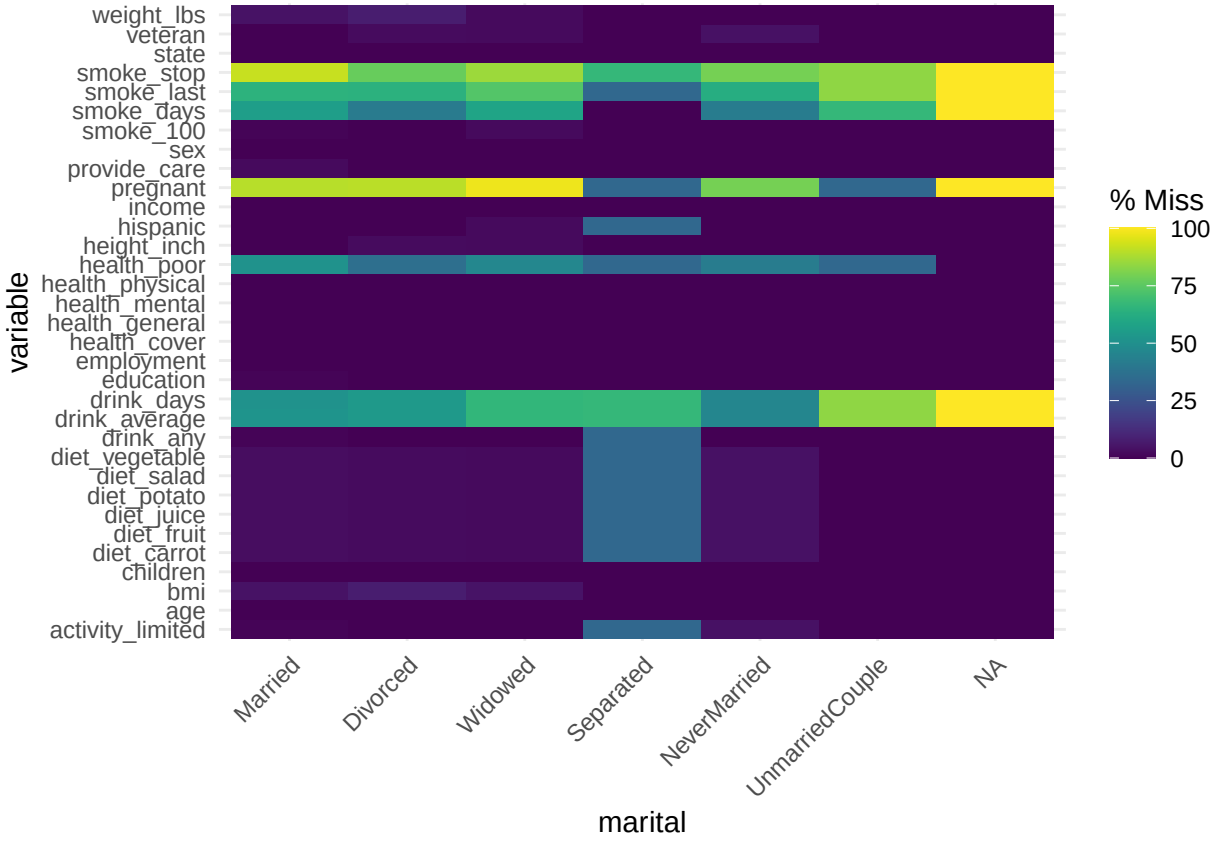
```
gg_miss_upset(airquality)
```



The message conveyed by this plot is that missing values in 'Ozone' are 35, in 'Solar' are 5, and in both variables only 2 missing values were detected.

Alternatively, you can use a heat map where the x-axis is the categories of the variable and the y-axis is the rest of the variables.

```
gg_miss_fct(riskfactors, fct = marital)
```



Testing Missing Data Dependence

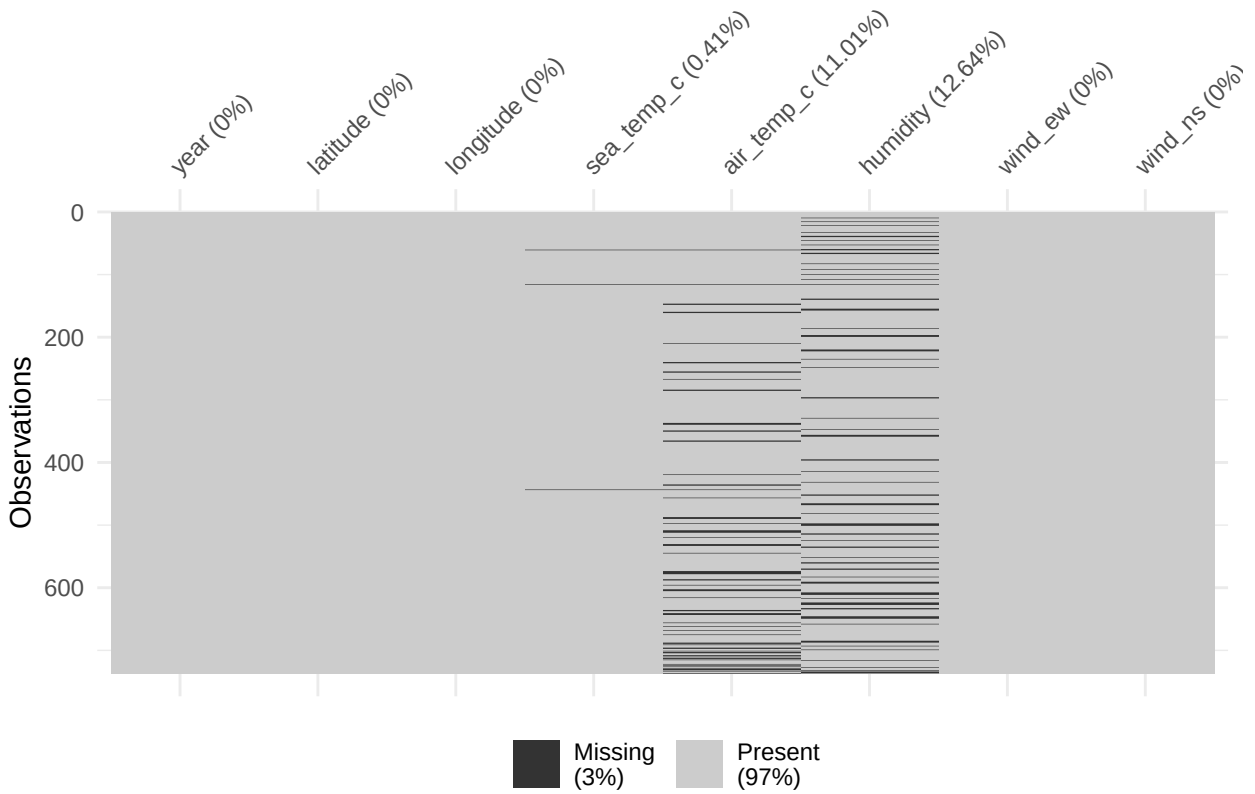
Once we have identified our missing values, the next step is to decide what to do with these missing values. Available options include either delete them or impute the data. Each approach has its pros and cons, but before we decide, we need to evaluate missing data dependence.

Missing data dependence is defined by the relationship that missing data has with other missing data: each type of missing data dependence has its implications.

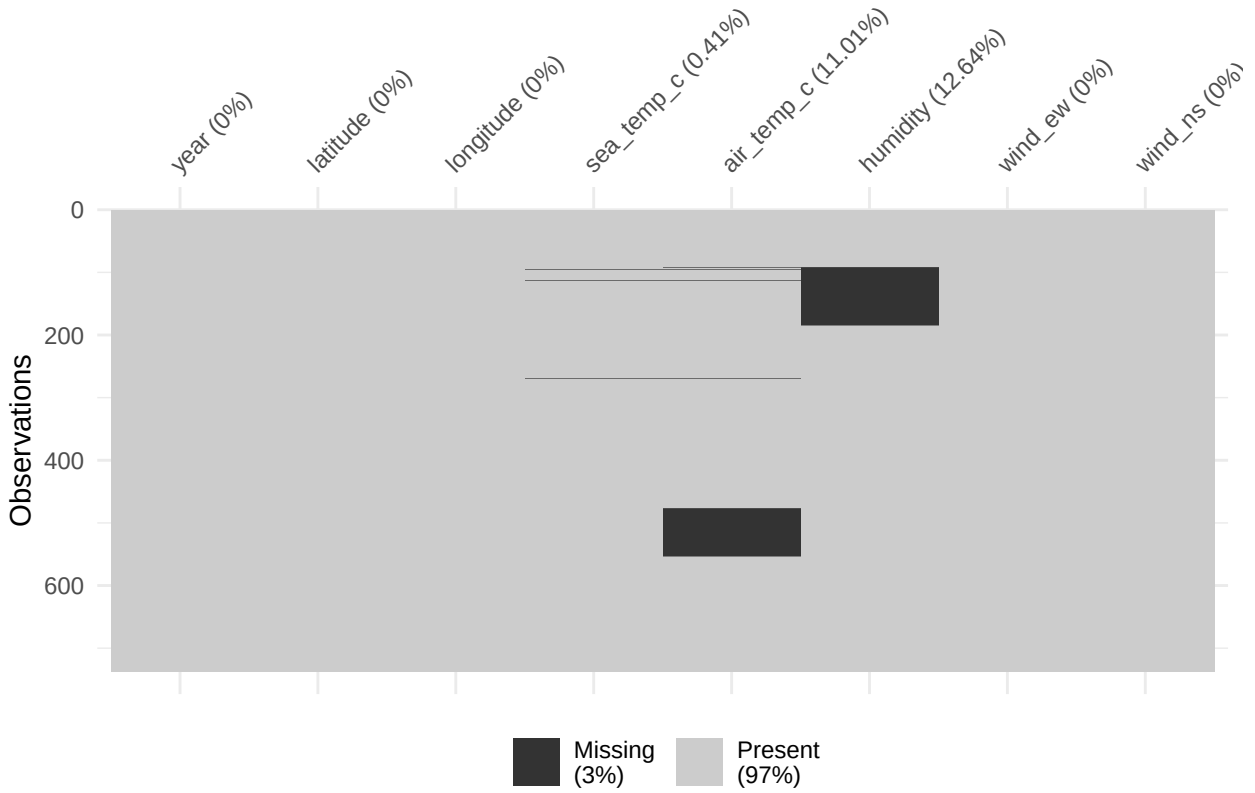
- Types of missing data dependence:
  - Missing Completely at Random (MCAR): Missingness has no association with any data you have observed, or not observed.
    - Imputation is advised
  - Missing at Random (MAR): missingness depends on data observed, but not on data unobserved
    - Imputation is advised but a very conservative approach is recommended
  - Missing Not at Random (MNAR): Missingness of the response is related to an unobserved value relevant to the assessment of interest.
    - Imputation or deletion is not advised as it would introduce bias

It is noteworthy to mention that identifying what type of missing data dependence is present in our data is a complicated task. Visualizing our data is a good place to start but it is not definitive, and thus, a peruse analysis is advised.

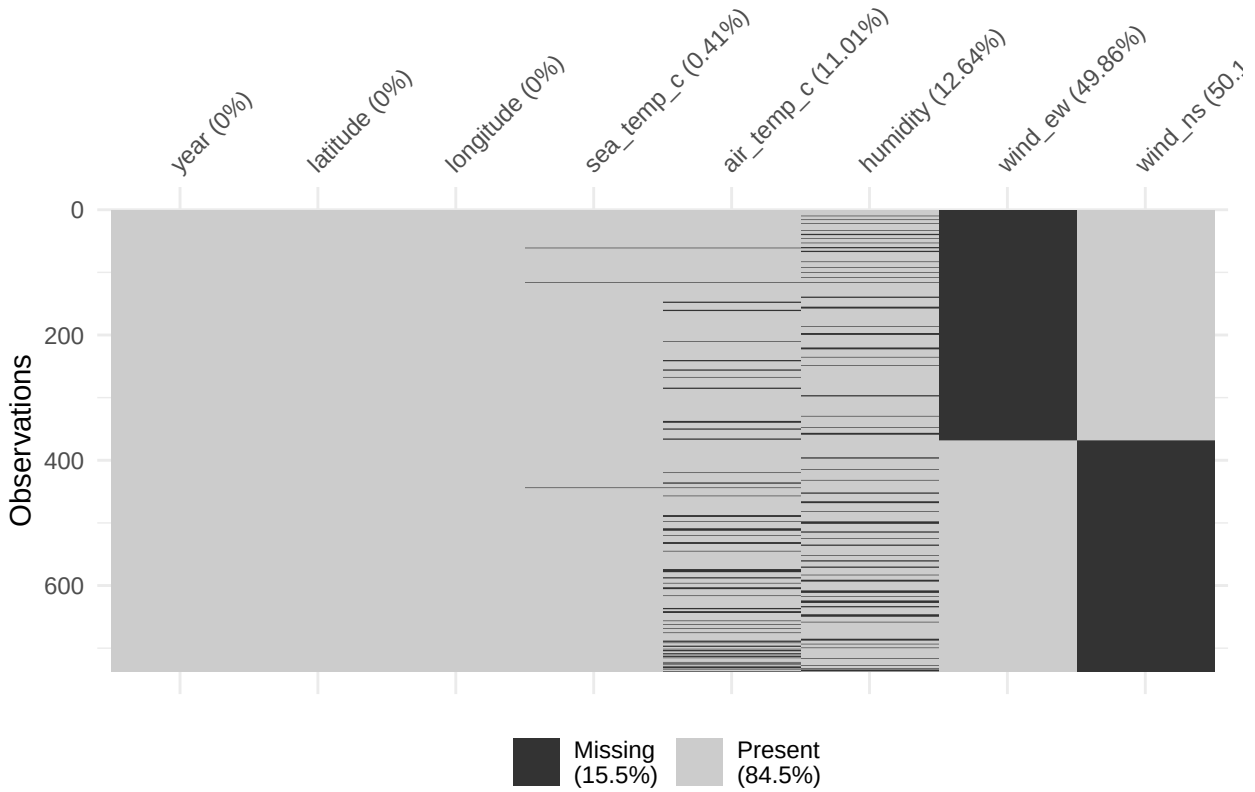
```
#MCAR
oceanbuoys %>%
  arrange(wind_ew) %>%
  vis_miss()
```




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




```
#MNAR
oceanbuoys %>%
  arrange(wind_ew) %>%
  mutate(id = row_number(),
         id = case_when(id < 368 ~ 0, T ~ 1),
         wind_ew = na_if(id, 0),
         wind_ns = na_if(id, 1)) %>%
  select(-id) %>%
  vis_miss()
```



Amazing!!! You are now an expert in visualizing your missing data in a summary format and determine its dependency. It is time to take things to the next rung in the ladder.

Missing Data Workflows

Often times, we are interested in visualizing a specific variable stratified based on the missingness of another variable. To do this, we first must specify which variables have missing values. A possible approach is to mutate on the interested variable to determine whether or not a cell contains a missing value. However, this approach is cumbersome and time consuming. An easier approach is to use a shadow matrix.

It is called a shadow matrix means to replicate the data set but filling out the values based on whether or not it contains missing values. We can do this by using the function 'as_shadow'.

```
print(airquality)
```

```
## # A tibble: 153 x 6
##    Ozone Solar.R  Wind  Temp Month   Day
##    <int>   <int> <dbl> <int> <int> <int>
##  1    41     190   7.4    67     5     1
##  2    36     118   8      72     5     2
##  3    12     149  12.6    74     5     3
##  4    18     313  11.5    62     5     4
##  5    NA      NA  14.3    56     5     5
##  6    28      NA  14.9    66     5     6
##  7    23     299   8.6    65     5     7
##  8    19      99  13.8    59     5     8
##  9     8      19  20.1    61     5     9
```

```
## 10    NA     194   8.6    69     5    10
## # ... with 143 more rows
```

```
as_shadow(airquality)
```

```
## # A tibble: 153 x 6
##    Ozone_NA Solar.R_NA Wind_NA Temp_NA Month_NA Day_NA
##    <fct>    <fct>      <fct>   <fct>   <fct>    <fct>
##  1 !NA      !NA        !NA     !NA     !NA      !NA
##  2 !NA      !NA        !NA     !NA     !NA      !NA
##  3 !NA      !NA        !NA     !NA     !NA      !NA
##  4 !NA      !NA        !NA     !NA     !NA      !NA
##  5 NA       NA         !NA     !NA     !NA      !NA
##  6 !NA      NA         !NA     !NA     !NA      !NA
##  7 !NA      !NA        !NA     !NA     !NA      !NA
##  8 !NA      !NA        !NA     !NA     !NA      !NA
##  9 !NA      !NA        !NA     !NA     !NA      !NA
## 10 NA       !NA        !NA     !NA     !NA      !NA
## # ... with 143 more rows
```

Comparing the data sets, we can see that when a missing value is absent, the shadow matrix fill that value with !NA, whereas for missing values it fills it with NA. But, WHERE IS OUR DATA?!

We need a function that binds our shadow matrix with our original data set, and thus, create the so-called nabular data set. Let's create it!

```
bind_shadow(airquality) #Notice the new names of the variables, they all end with '_NA'
```

```
## # 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>
```

```
bind_shadow(airquality, only_miss = T) #Only attach variables with missing variables
```

```
## # A tibble: 153 x 8
##    Ozone Solar.R  Wind  Temp Month   Day Ozone_NA Solar.R_NA
##    <int>   <int> <dbl> <int> <int> <int> <fct>    <fct>
##  1    41     190   7.4    67     5     1 !NA      !NA
##  2    36     118   8      72     5     2 !NA      !NA
##  3    12     149  12.6    74     5     3 !NA      !NA
##  4    18     313  11.5    62     5     4 !NA      !NA
```

```
##  5    NA      NA  14.3    56     5     5 NA       NA
##  6    28      NA  14.9    66     5     6 !NA      NA
##  7    23     299   8.6    65     5     7 !NA      !NA
##  8    19      99  13.8    59     5     8 !NA      !NA
##  9     8      19  20.1    61     5     9 !NA      !NA
## 10    NA     194   8.6    69     5    10 NA       !NA
## # ... with 143 more rows
```

With this data set, we can start doing very fancy explorations!

Now that you created a nabular data set, let's calculate summary statistics based on the missingness of another variable.

```
oceanbuoys %>%
  bind_shadow() %>%
  janitor::clean_names() %>%
  group_by(humidity_na) %>%
  summarize(
    wind_ew_mean = mean(wind_ew),
    wind_we_sd = sd(wind_ew),
    n_obs = n() #This function counts the number of observations
  )
```

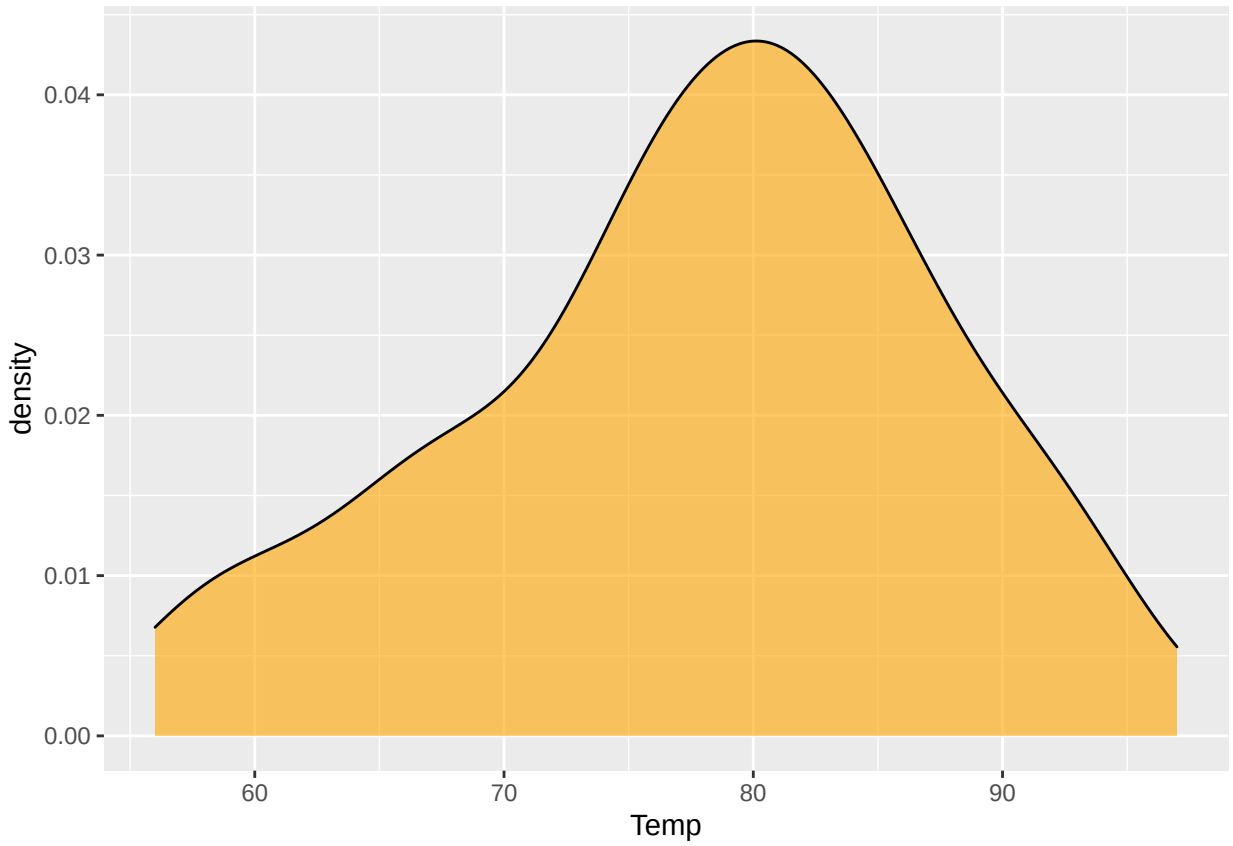
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 4
##   humidity_na wind_ew_mean wind_we_sd n_obs
##   <fct>              <dbl>      <dbl> <int>
## 1 !NA                -3.78       1.90   643
## 2 NA                 -3.30       2.31    93
```

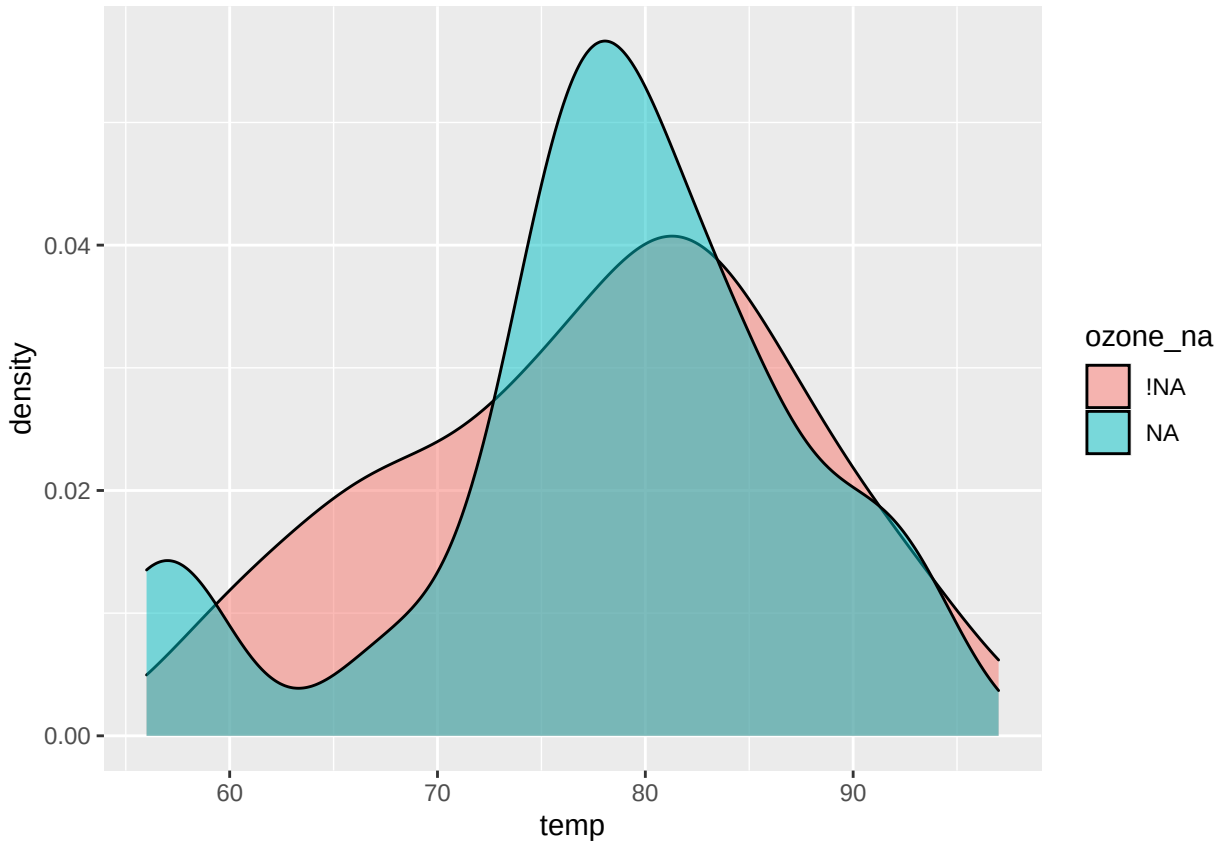
Now let's explore how variables vary as other variables are missing.

For instance, if we look at this density plot, we cannot differentiate the distribution of 'Temp' based on the missingness of 'Ozone'. However, with a nabular data we can!

```
ggplot(airquality, aes(Temp)) +
  geom_density(color = 'black', fill = 'orange', alpha = 0.6)
```

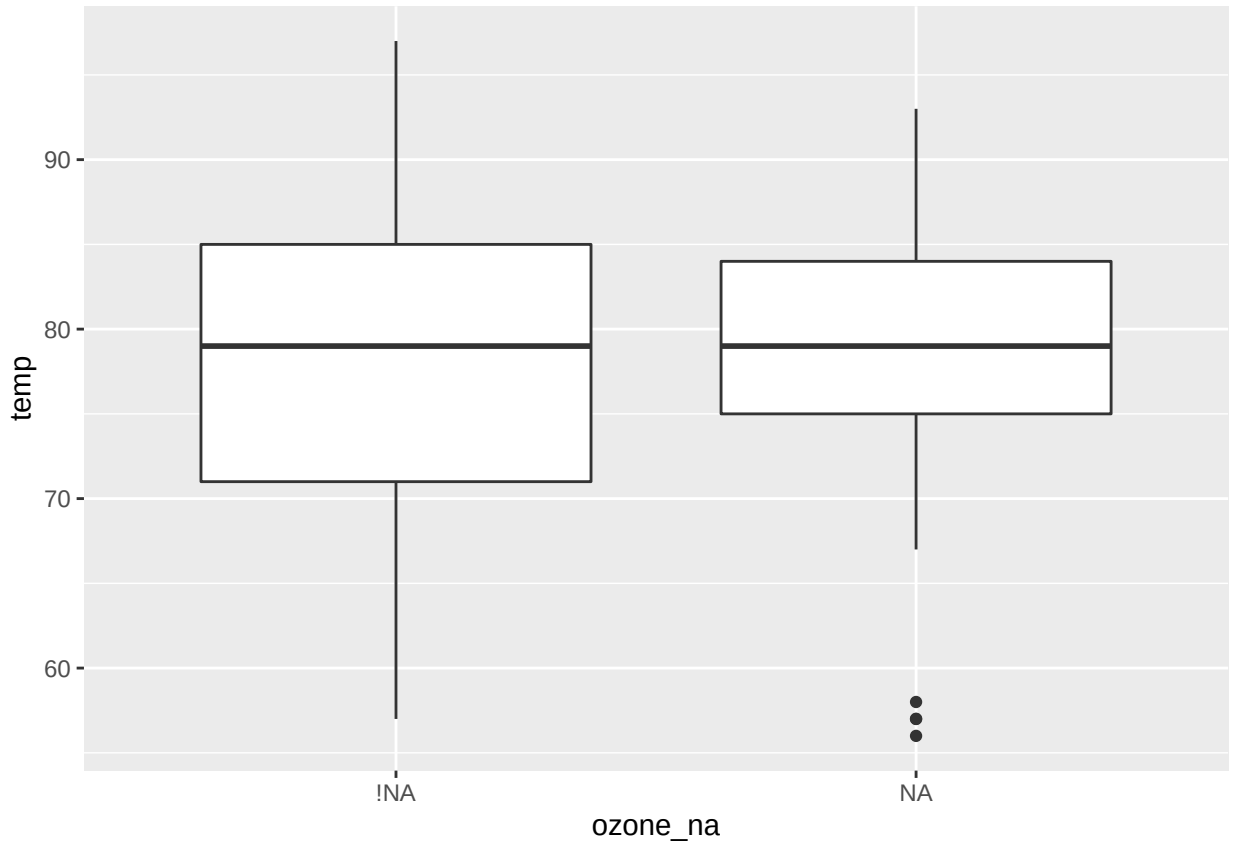



```
airquality %>%
  bind_shadow() %>%
  janitor::clean_names() %>%
  ggplot(aes(temp, fill = ozone_na)) +
  geom_density(color = 'black', alpha = 0.5)
```

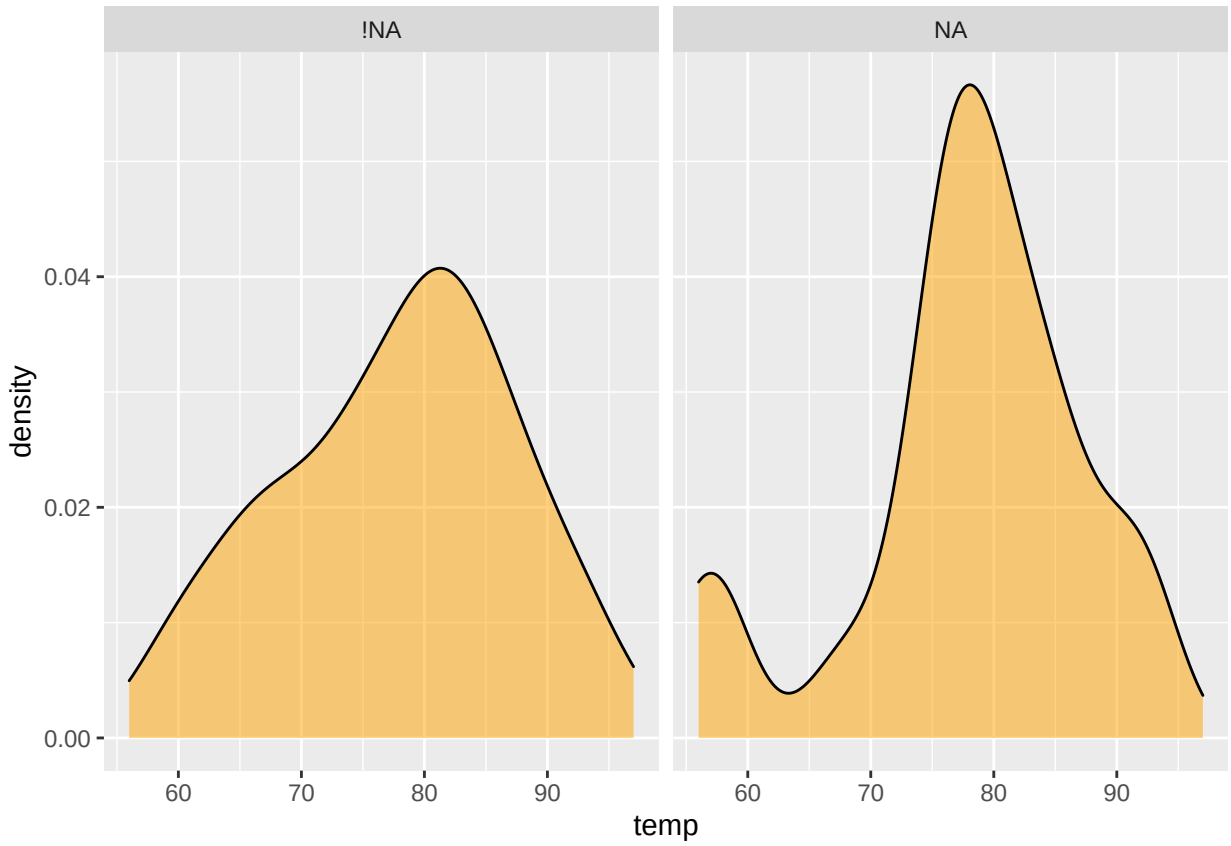


We can determine based on the plot that the values of 'Temp' do not change much if 'Ozone' is missing or not. You can extend this knowledge for other types of plots as well.

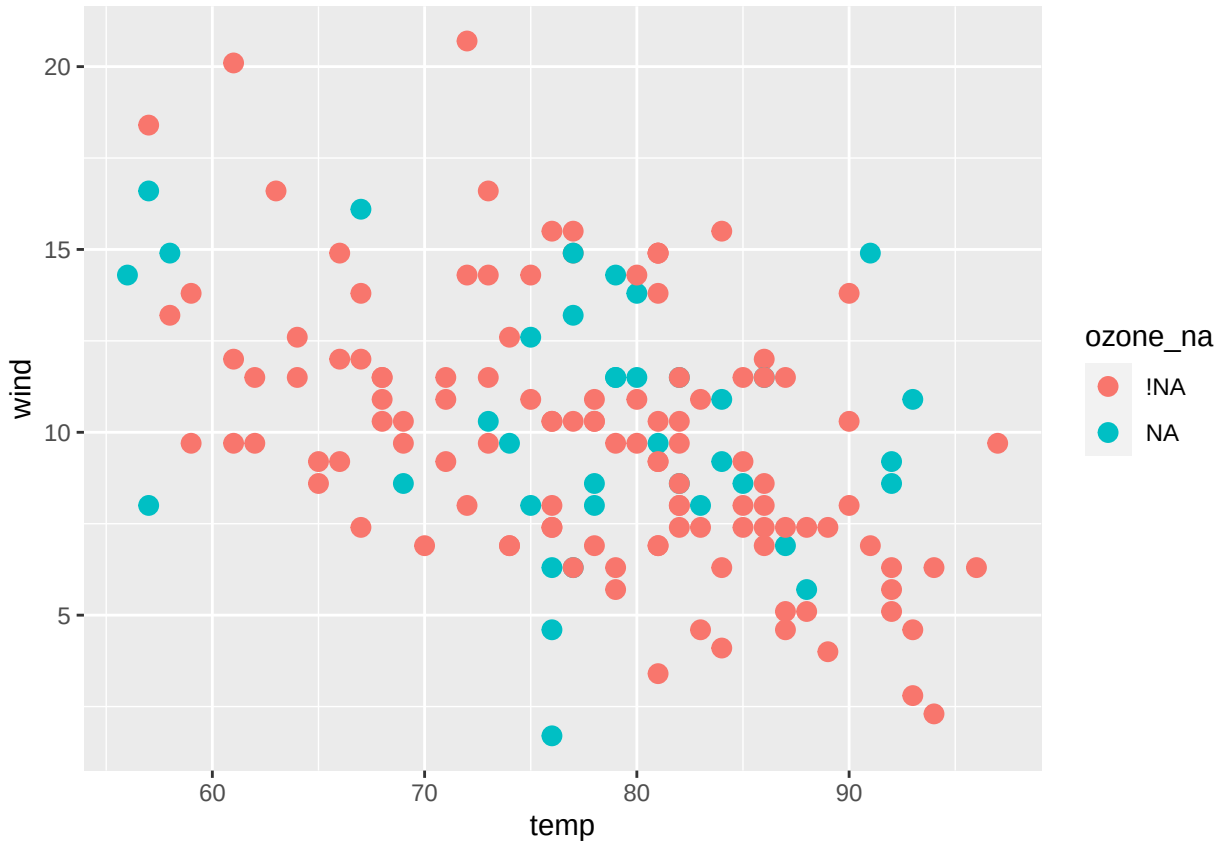
```
airquality %>%
  bind_shadow() %>%
  janitor::clean_names() %>%
  ggplot(aes(ozone_na, temp)) +
  geom_boxplot()
```



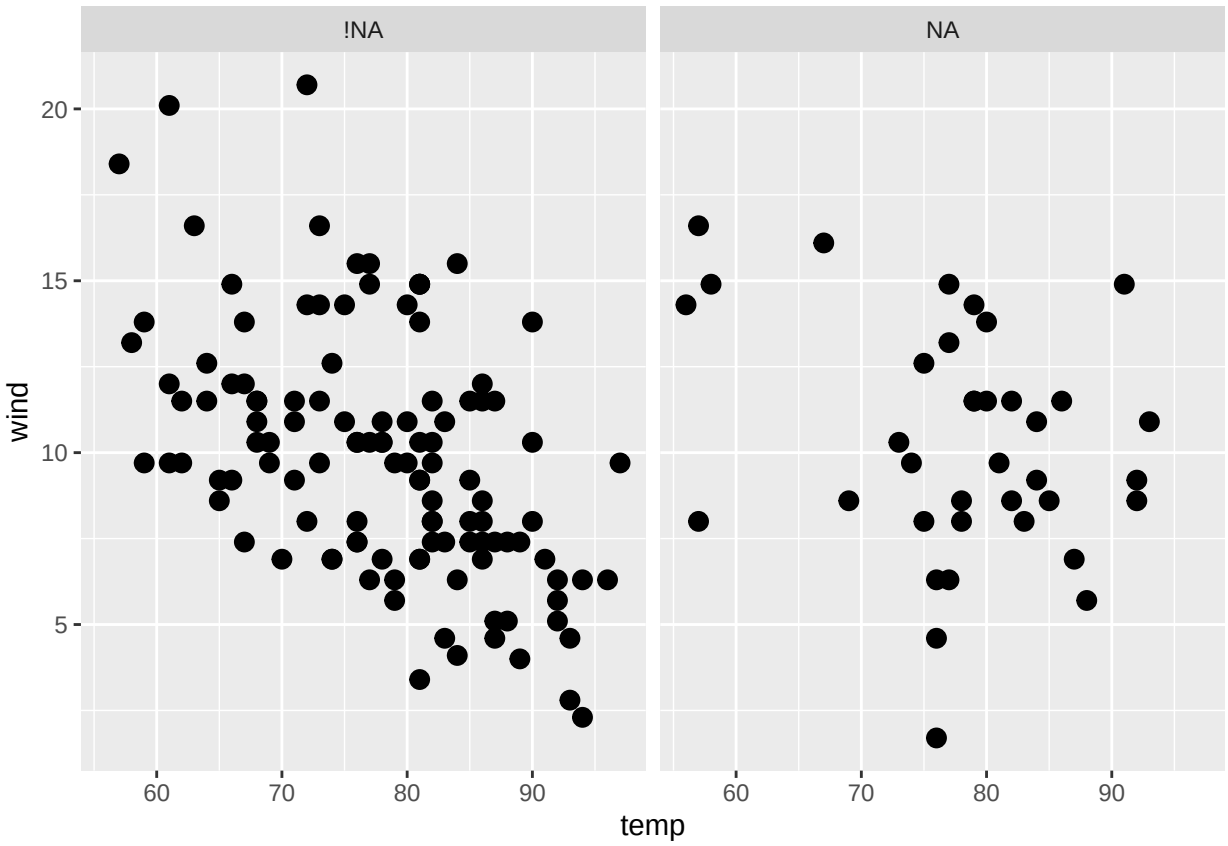

```
airquality %>%
  bind_shadow() %>%
  janitor::clean_names() %>%
  ggplot(aes(temp)) +
  geom_density(fill = 'orange', alpha = 0.5) +
  facet_wrap(~ozone_na)
```




```
airquality %>%
  bind_shadow() %>%
  janitor::clean_names() %>%
  ggplot(aes(temp, wind, color = ozone_na)) +
  geom_point(size = 3)
```



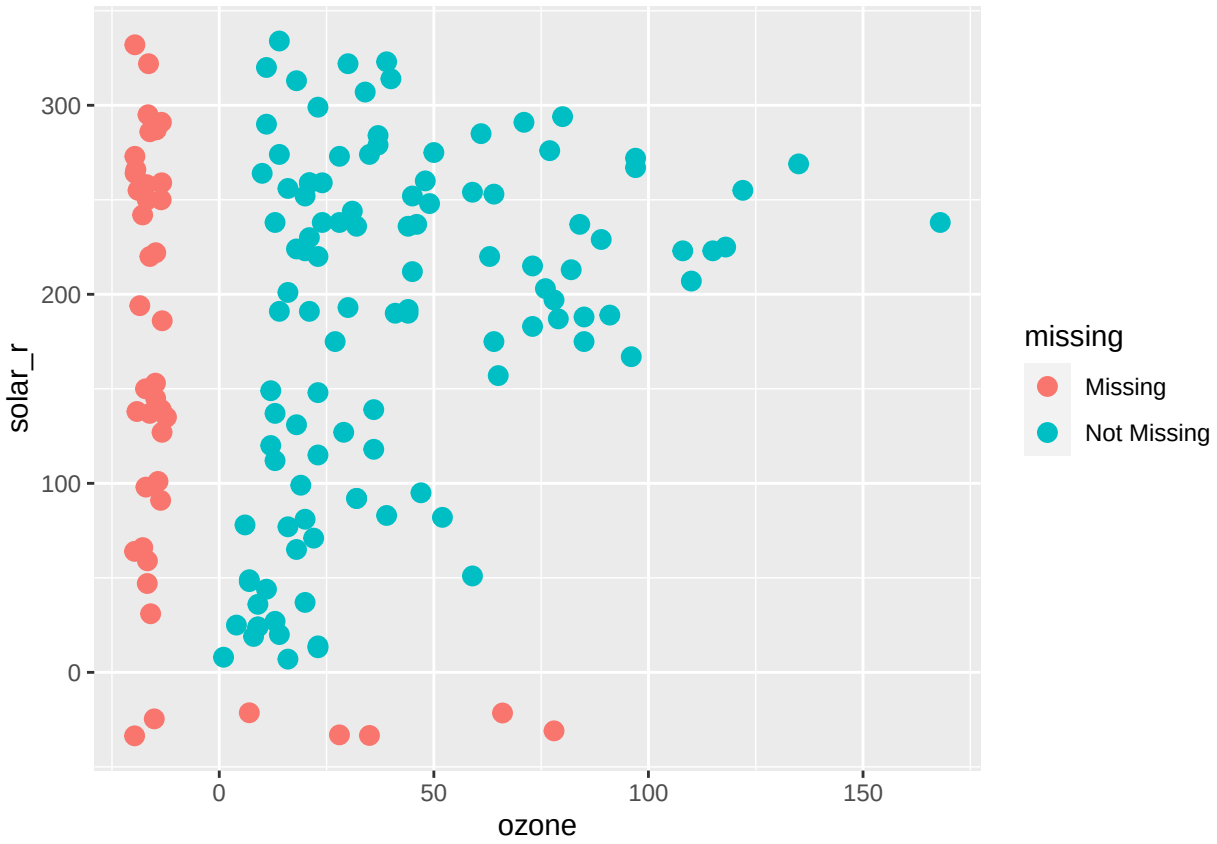

```
airquality %>%
  bind_shadow() %>%
  janitor::clean_names() %>%
  ggplot(aes(temp, wind)) +
  geom_point(size = 3) +
  facet_wrap(~ozone_na)
```



Visualizing Data in Two Dimensions

Missing data is often ignored in scatter plots. The ggplot ecosystem is very nice and prints a warning in the console when missing data has been removed. But, what if we want to see what is the distribution of missing data in a scatter plot. Well, we can do this without creating a nabular data manually. The function geom_miss_point layer do that for us automatically. And the best part is that it is still a ggplot object so you can manipulate it in various ways!

```
airquality %>%
  janitor::clean_names() %>%
  ggplot(aes(ozone, solar_r)) +
  geom_miss_point(size = 3)
```



FANTASTIC JOB!!!!

YOU ARE NOW AN EXPERT IN VISUALIZING MISSING DATA IN VARIOUS WAYS. YOU HAVE LEARNED A LOT. FEEL GREAT WITH YOURSELF BECAUSE YOU ARE AMAZING!

NEXT STEP IS TO IMPUTE AND VISUALIZE THE IMPUTATIONS TO ASSESS ITS ACCURACY.

MUCHAS GRACIAS!!!