# R Workshop 4: Intro housekeeping

Teal Potter

**10/15/2021**

# Table of Contents



By 'housekeeping', I just mean keeping data tidy and easy to use (reformat, visualize, model). In my mind, this concept is natural to introduce with ways for looking at key information about your data in R so you can then do actions to make/keep your data tidy as you work with it.

This tutorial will be focused on looking at the data, whereas reformatting data will be included in the data wrangling tutorial. Note that all the functions used to look at the data are base R functions so no addtional packages need to be loaded.

We'll be working with the Soils dataset yet again. And first, I'll save the pH column from this dataset as a stand-alone vector so I can show how some of these functions work with data frames like Soils as well as this new vector.

## Setup

```
library(tidyverse) # will be used in the NA section
library(car)

pH_vector <- Soils$pH # making a vector to demo on
soils <- Soils # also saving a copy of the Soils dataset
```

# Tricks for examining data without looking at entire data frame

## Basic functions that show info about your data

`dim()` reveals the dimensions of your object, returns `NULL` if not a rectangular dataset

```
dim(Soils)

## [1] 48 14

dim(pH_vector)

## NULL
```

`length()` returns the number of elements in a vector and the number of columns in a data frame or matrix

```
length(Soils)

## [1] 14

length(pH_vector)

## [1] 48
```

`nrow()` and `ncol()` reveal the number of rows and columns respectively

```
nrow(Soils)

## [1] 48

ncol(Soils)

## [1] 14
```

`levels()` shows you how many groups exist in a column that contains factor data type

```
levels(Soils$Depth)

## [1] "0-10"  "10-30" "30-60" "60-90"
```

# Basic functions that show you parts of your data object

`names(Soils)` shows column names, same as `colnames()`

```
names(Soils)

##  [1] "Group"   "Contour" "Depth"   "Gp"      "Block"   "pH"      "N"
##  [8] "Dens"    "P"       "Ca"      "Mg"      "K"       "Na"      "Conduc"
```

`rownames(Soils)` shows rownames if they exist, default is #s as shown

```
rownames(Soils)

##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14" "15"
## [16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29" "30"
## [31] "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41" "42" "43" "44" "45"
## [46] "46" "47" "48"
```

`head()` is handy for showing the top few rows of data as a default. But you can be more specific by adding arguments to change the number of columns and rows you want it to return.

```
head(Soils) # default reveals first few rows

##   Group Contour Depth Gp Block   pH     N Dens   P    Ca   Mg    K   Na Conduc
## 1     1     Top  0-10 T0     1 5.40 0.188 0.92 215 16.35 7.65 0.72 1.14   1.09
## 2     1     Top  0-10 T0     2 5.65 0.165 1.04 208 12.25 5.15 0.71 0.94   1.35
## 3     1     Top  0-10 T0     3 5.14 0.260 0.95 300 13.02 5.68 0.68 0.60   1.41
## 4     1     Top  0-10 T0     4 5.14 0.169 1.10 248 11.92 7.88 1.09 1.01   1.64
## 5     2     Top 10-30 T1     1 5.14 0.164 1.12 174 14.17 8.12 0.70 2.17   1.85
## 6     2     Top 10-30 T1     2 5.10 0.094 1.22 129  8.55 6.92 0.81 2.67   3.18

head(Soils)[1]

##   Group
## 1     1
## 2     1
## 3     1
## 4     1
## 5     2
## 6     2

head(Soils)[1:3]

##   Group Contour Depth
## 1     1     Top  0-10
## 2     1     Top  0-10
## 3     1     Top  0-10
## 4     1     Top  0-10
## 5     2     Top 10-30
## 6     2     Top 10-30
```

```
head(Soils, 2)
```

```
##   Group Contour Depth Gp Block   pH     N Dens   P    Ca   Mg    K   Na Conduc
## 1     1     Top  0-10 T0     1 5.40 0.188 0.92 215 16.35 7.65 0.72 1.14   1.09
## 2     1     Top  0-10 T0     2 5.65 0.165 1.04 208 12.25 5.15 0.71 0.94   1.35
```

`tail()` shows last few rows and works the same way as head

```
tail(Soils)
```

```
##    Group    Contour Depth Gp Block   pH     N Dens   P   Ca    Mg    K    Na
## 43    11 Depression 30-60 D3     3 4.35 0.032 1.55  82 5.99  9.73 0.22  7.02
## 44    11 Depression 30-60 D3     4 4.64 0.065 1.46 152 4.43 10.54 0.22  7.61
## 45    12 Depression 60-90 D6     1 3.82 0.038 1.40 105 4.65  9.85 0.18 10.15
## 46    12 Depression 60-90 D6     2 4.24 0.035 1.47 100 4.56  8.95 0.33 10.51
## 47    12 Depression 60-90 D6     3 4.22 0.030 1.56  97 5.29  8.37 0.14  8.27
## 48    12 Depression 60-90 D6     4 4.41 0.058 1.58 130 4.58  9.46 0.14  9.28
##    Conduc
## 43   8.60
## 44   9.09
## 45  12.26
## 46  11.29
## 47   9.51
## 48  12.69
```

`summary()` provides a different output for each variable, depending on its class. For numeric data such as pH, `summary()` displays the minimum, 1st quartile, median, mean, 3rd quartile, and maximum. These values help us understand how the data are distributed without making a graph.

```
summary(Soils)
```

```
##      Group          Contour     Depth          Gp     Block        pH
##  1      : 4   Depression:16   0-10 :12   D0     : 4   1:12   Min.   :3.740
##  2      : 4   Slope     :16   10-30:12   D1     : 4   2:12   1st Qu.:4.058
##  3      : 4   Top       :16   30-60:12   D3     : 4   3:12   Median :4.545
##  4      : 4                  60-90:12   D6     : 4   4:12   Mean   :4.669
##  5      : 4                             S0     : 4          3rd Qu.:5.140
##  6      : 4                             S1     : 4          Max.   :6.670
##  (Other):24                             (Other):24
##        N                Dens             P                Ca
##  Min.   :0.03000   Min.   :0.780   Min.   : 79.0   Min.   : 3.820
##  1st Qu.:0.05075   1st Qu.:1.127   1st Qu.:108.8   1st Qu.: 5.040
##  Median :0.08450   Median :1.400   Median :131.0   Median : 7.305
##  Mean   :0.10194   Mean   :1.316   Mean   :166.2   Mean   : 8.029
##  3rd Qu.:0.12925   3rd Qu.:1.502   3rd Qu.:214.2   3rd Qu.: 9.735
##  Max.   :0.29800   Max.   :1.600   Max.   :445.0   Max.   :16.350
##
##        Mg               K                Na              Conduc
##  Min.   : 5.150   Min.   :0.1400   Min.   : 0.600   Min.   : 0.670
##  1st Qu.: 7.537   1st Qu.:0.2750   1st Qu.: 2.545   1st Qu.: 2.790
```

```
##  Median : 8.515   Median :0.4250   Median : 5.520   Median : 6.635
##  Mean   : 8.465   Mean   :0.4662   Mean   : 5.600   Mean   : 6.589
##  3rd Qu.: 9.648   3rd Qu.:0.6425   3rd Qu.: 8.355   3rd Qu.: 9.852
##  Max.   :10.960   Max.   :1.0900   Max.   :11.040   Max.   :13.320
##
```

`str()` is actually a very general function that you can use on most objects in R. Any time you want to understand the structure of something (a dataset, function, etc.), str() is a good place to start. `attributes()` also works to see what types of statistical info is saved in a statistial model's output.

```
str(Soils)
```

```
## 'data.frame':    48 obs. of  14 variables:
##  $ Group  : Factor w/ 12 levels "1","2","3","4",..: 1 1 1 1 2 2 2 2 3 3 ...
##  $ Contour: Factor w/ 3 levels "Depression","Slope",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ Depth  : Factor w/ 4 levels "0-10","10-30",..: 1 1 1 1 2 2 2 2 3 3 ...
##  $ Gp     : Factor w/ 12 levels "D0","D1","D3",..: 9 9 9 9 10 10 10 10 11 11 ...
##  $ Block  : Factor w/ 4 levels "1","2","3","4": 1 2 3 4 1 2 3 4 1 2 ...
##  $ pH     : num  5.4 5.65 5.14 5.14 5.14 5.1 4.7 4.46 4.37 4.39 ...
##  $ N      : num  0.188 0.165 0.26 0.169 0.164 0.094 0.1 0.112 0.112 0.058 ...
##  $ Dens   : num  0.92 1.04 0.95 1.1 1.12 1.22 1.52 1.47 1.07 1.54 ...
##  $ P      : int  215 208 300 248 174 129 117 170 121 115 ...
##  $ Ca     : num  16.4 12.2 13 11.9 14.2 ...
##  $ Mg     : num  7.65 5.15 5.68 7.88 8.12 ...
##  $ K      : num  0.72 0.71 0.68 1.09 0.7 0.81 0.39 0.7 0.74 0.77 ...
##  $ Na     : num  1.14 0.94 0.6 1.01 2.17 2.67 3.32 3.76 5.74 5.85 ...
##  $ Conduc : num  1.09 1.35 1.41 1.64 1.85 3.18 4.16 5.14 5.73 6.45 ...
```

```
attributes(Soils)
```

```
## $names
##  [1] "Group"   "Contour" "Depth"   "Gp"      "Block"   "pH"      "N"
##  [8] "Dens"    "P"       "Ca"      "Mg"      "K"       "Na"      "Conduc"
##
## $row.names
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14" "15"
## [16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29" "30"
## [31] "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41" "42" "43" "44" "45"
## [46] "46" "47" "48"
##
## $class
## [1] "data.frame"
```

```
mod <- lm(Soils$pH ~ Soils$N) # example linear regression model
attributes(mod)
```

```
## $names
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
```

```
##  [9] "xlevels"       "call"          "terms"         "model"
##
## $class
## [1] "lm"
```

`unique()` shows unique groups or values depending on the class of the vector provided

```
unique(Soils$Depth) # unique levels of factor
```

```
## [1] 0-10  10-30 30-60 60-90
## Levels: 0-10 10-30 30-60 60-90
```

```
unique(Soils$pH) # unique values
```

```
##  [1] 5.40 5.65 5.14 5.10 4.70 4.46 4.37 4.39 4.17 3.89 3.88 4.07 3.74 5.11 5.46
## [16] 5.61 5.85 4.57 4.78 6.67 3.96 4.00 4.12 4.99 3.80 3.93 4.02 5.24 5.20 5.30
## [31] 5.67 4.91 4.79 5.36 3.94 4.52 4.35 4.64 3.82 4.24 4.22 4.41
```

You can also compare 2 datasets. Here are two examples using `identical()`. You may also want to investigate the set operators including `union()`, `intersect()`, `setdiff()`, and `set_equal()` for comparing and manipulating vectors based on similarity.

```
identical(soils, Soils)
```

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

## Use basic indexing instead of functions to look at parts of data objects

```
names(Soils)[1] # shows first column's name
```

```
## [1] "Group"
```

```
Soils[1:3, 1:5]  #show specified rows and columns, format is data[rows, columns]
```

```
##   Group Contour Depth Gp Block
## 1     1     Top  0-10 T0     1
## 2     1     Top  0-10 T0     2
## 3     1     Top  0-10 T0     3
```

# Dealing with NAs

NA stands for 'not available' which is different from NaN which means 'Not a Number'. R recognizes NA is neither a string nor a numeric value. It is an indicator of missingness and therefore behaves differently than other types

of data when you attempt to observe and manipulate data based on missingness. Thus NA deserves its own section in this tutorial.

NA is never the same as zero. For example, imagine that you are counting the number of thrips on leaves of plants in a greenhouse experiment. In your count column, a zero would mean you counted no thrips on a leaf, while NA would mean that that plant may have died early in the experiment so there are no thrips data available. In the case of soil pH in the example below, zero isn't a reasonable measurment to get but you may have lost some soil samples reserved for pH measurement which would be recorded as NA for pH.

I will first add some NAs randomly into a copy of the dataset to work with because the Soils dataset is a complete dataset with no NAs. I'm also adding in 9999 to show how to change a value to NA later.

```
NA.list = sample(1:nrow(soils), round(nrow(soils)*.1))
soils[NA.list,]$pH <- NA
nine.list = sample(1:nrow(soils), round(nrow(soils)*.02))
soils[nine.list,]$pH <- 9999

soils$pH # now the pH column contains several NAs in the soil dataset

##  [1]    5.40    5.65    5.14    5.14    5.14    5.10    4.70    4.46    4.37
## [10]      NA    4.17    3.89    3.88    4.07      NA      NA    5.11    5.46
## [19]    5.61    5.85    4.57    5.11      NA    6.67    3.96    4.00    4.12
## [28]    4.99    3.80    3.96    3.93    4.02    5.24    5.20      NA    5.67
## [37]    4.46    4.91    4.79    5.36    3.94    4.52    4.35    4.64    3.82
## [46]    4.24    4.22 9999.00
```

# Finding NAs in a data object

# Finding NAs using logicals

`is.na()` returns a logical vector where TRUE indicates positions in the data object that contain NA.

`complete.cases(soils)` also returns a summary of cases where there are no missing values. For a data frame, it shows which rows contain an NA.

```
head(is.na(soils))  # option 1, only showing top few rows which happen to not
contain NAs

##   Group Contour Depth    Gp Block    pH     N  Dens     P    Ca    Mg     K
## 1 FALSE   FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 2 FALSE   FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 3 FALSE   FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 4 FALSE   FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 5 FALSE   FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 6 FALSE   FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##      Na Conduc
## 1 FALSE  FALSE
## 2 FALSE  FALSE
## 3 FALSE  FALSE
```

```
## 4 FALSE  FALSE
## 5 FALSE  FALSE
## 6 FALSE  FALSE

complete.cases(soils) # option 2

##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
## [13]  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
## [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
## [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

# How many NAs do you have?

And here is a quick example of how you can see how many NAs you have in a dataset or column

```
sum(is.na(soils$pH), na.rm = FALSE)  # change to na.rm = TRUE if you want to count TRUE values that exclude NAs if your action was not related to quantifying NAs directly.

## [1] 5
```

# Which rows contain NAs?

You can also look at the rows that contain missing values to help you decide what to do. When I see the ! operator I read "is not", so these options show you rows that are not complete cases.

```
soils[!complete.cases(soils),] # option 1 with indexing on whole dataset

##    Group    Contour Depth Gp Block pH     N Dens   P   Ca    Mg    K    Na
## 10     3        Top 30-60 T3     2 NA 0.058 1.54 115 4.73  6.91 0.77  5.85
## 15     4        Top 60-90 T6     3 NA 0.055 1.53  91 4.98  8.00 0.23  8.78
## 16     4        Top 60-90 T6     4 NA 0.053 1.40  79 5.86 10.14 0.41 11.04
## 23     6      Slope 10-30 S1     3 NA 0.122 1.30 214 8.22  7.75 0.32  3.07
## 35     9 Depression  0-10 D0     3 NA 0.136 1.00 259 9.96  8.08 0.45  1.97
##    Conduc
## 10   6.45
## 15  11.26
## 16  12.15
## 23   3.67
## 35   2.27

filter(soils, is.na(pH))   # option 2 with tidyverse/tidyr::drop_na on pH column

##    Group    Contour Depth Gp Block pH     N Dens   P   Ca    Mg    K    Na
## 10     3        Top 30-60 T3     2 NA 0.058 1.54 115 4.73  6.91 0.77  5.85
## 15     4        Top 60-90 T6     3 NA 0.055 1.53  91 4.98  8.00 0.23  8.78
## 16     4        Top 60-90 T6     4 NA 0.053 1.40  79 5.86 10.14 0.41 11.04
## 23     6      Slope 10-30 S1     3 NA 0.122 1.30 214 8.22  7.75 0.32  3.07
## 35     9 Depression  0-10 D0     3 NA 0.136 1.00 259 9.96  8.08 0.45  1.97
```

```
##    Conduc
## 10   6.45
## 15  11.26
## 16  12.15
## 23   3.67
## 35   2.27
```

# Removing NAs

## Removing rows that contain NAs from data frame

A good practice is to do your analyses and graphing in R using a complete dataset (sometimes called master dataset) that includes rows for which some columns may contain NAs. When you use R as your analysis tool you can subset your dataset in your code and then do your analysis instead of saving and keeping track of a separate version of your master file that has excludes rows with NAs or some other variation.

Here are some simple ways to remove NAs.

Remove all rows in data object that contain one or more NAs.

Note that the tidyverse::dplyr option here does not use `filter(soils, pH == NA)` becuase anything set equal to NA results in NA.

```
na.omit(soils)  # easy option 1 base R function
```

```
##    Group    Contour Depth Gp Block      pH     N Dens   P    Ca    Mg    K
## 1      1        Top  0-10 T0     1    5.40 0.188 0.92 215 16.35  7.65 0.72
## 2      1        Top  0-10 T0     2    5.65 0.165 1.04 208 12.25  5.15 0.71
## 3      1        Top  0-10 T0     3    5.14 0.260 0.95 300 13.02  5.68 0.68
## 4      1        Top  0-10 T0     4    5.14 0.169 1.10 248 11.92  7.88 1.09
## 5      2        Top 10-30 T1     1    5.14 0.164 1.12 174 14.17  8.12 0.70
## 6      2        Top 10-30 T1     2    5.10 0.094 1.22 129  8.55  6.92 0.81
## 7      2        Top 10-30 T1     3    4.70 0.100 1.52 117  8.74  8.16 0.39
## 8      2        Top 10-30 T1     4    4.46 0.112 1.47 170  9.49  9.16 0.70
## 9      3        Top 30-60 T3     1    4.37 0.112 1.07 121  8.85 10.35 0.74
## 11     3        Top 30-60 T3     3    4.17 0.078 1.26 112  6.29  7.95 0.26
## 12     3        Top 30-60 T3     4    3.89 0.070 1.42 117  6.61  9.76 0.41
## 13     4        Top 60-90 T6     1    3.88 0.077 1.25 127  6.41 10.96 0.56
## 14     4        Top 60-90 T6     2    4.07 0.046 1.54  91  3.82  6.61 0.50
## 17     5      Slope  0-10 S0     1    5.11 0.247 0.94 261 13.25  7.55 0.61
## 18     5      Slope  0-10 S0     2    5.46 0.298 0.96 300 12.30  7.50 0.68
## 19     5      Slope  0-10 S0     3    5.61 0.145 1.10 242  9.66  6.76 0.63
## 20     5      Slope  0-10 S0     4    5.85 0.186 1.20 229 13.78  7.12 0.62
## 21     6      Slope 10-30 S1     1    4.57 0.102 1.37 156  8.58  9.92 0.63
## 22     6      Slope 10-30 S1     2    5.11 0.097 1.30 139  8.58  8.69 0.42
## 24     6      Slope 10-30 S1     4    6.67 0.083 1.42 132 12.68  9.56 0.55
## 25     7      Slope 30-60 S3     1    3.96 0.059 1.53  98  4.80 10.00 0.36
## 26     7      Slope 30-60 S3     2    4.00 0.050 1.50 115  5.06  8.91 0.28
## 27     7      Slope 30-60 S3     3    4.12 0.086 1.55 148  6.16  7.58 0.16
```

```
## 28     7      Slope 30-60 S3     4    4.99 0.048 1.46  97  7.49  9.38 0.40
## 29     8      Slope 60-90 S6     1    3.80 0.049 1.48 108  3.82  8.80 0.24
## 30     8      Slope 60-90 S6     2    3.96 0.036 1.28 103  4.78  7.29 0.24
## 31     8      Slope 60-90 S6     3    3.93 0.048 1.42 109  4.93  7.47 0.14
## 32     8      Slope 60-90 S6     4    4.02 0.039 1.51 100  5.66  8.84 0.37
## 33     9 Depression  0-10 D0     1    5.24 0.194 1.00 445 12.27  6.27 0.72
## 34     9 Depression  0-10 D0     2    5.20 0.256 0.78 380 11.39  7.55 0.78
## 36     9 Depression  0-10 D0     4    5.67 0.127 1.13 248  9.12  7.04 0.55
## 37    10 Depression 10-30 D1     1    4.46 0.087 1.24 276  7.24  9.40 0.43
## 38    10 Depression 10-30 D1     2    4.91 0.092 1.47 158  7.37 10.57 0.59
## 39    10 Depression 10-30 D1     3    4.79 0.047 1.46 121  6.99  9.91 0.30
## 40    10 Depression 10-30 D1     4    5.36 0.095 1.26 195  8.59  8.66 0.48
## 41    11 Depression 30-60 D3     1    3.94 0.054 1.60 148  4.85  9.62 0.18
## 42    11 Depression 30-60 D3     2    4.52 0.051 1.53 115  6.34  9.78 0.34
## 43    11 Depression 30-60 D3     3    4.35 0.032 1.55  82  5.99  9.73 0.22
## 44    11 Depression 30-60 D3     4    4.64 0.065 1.46 152  4.43 10.54 0.22
## 45    12 Depression 60-90 D6     1    3.82 0.038 1.40 105  4.65  9.85 0.18
## 46    12 Depression 60-90 D6     2    4.24 0.035 1.47 100  4.56  8.95 0.33
## 47    12 Depression 60-90 D6     3    4.22 0.030 1.56  97  5.29  8.37 0.14
## 48    12 Depression 60-90 D6     4 9999.00 0.058 1.58 130  4.58  9.46 0.14
```

```
check <- filter(soils, !is.na(pH))  # option 2  tidyverse/dplyr::filter()
```

```
check$pH # let's look at the pH column instead of the whole dataset
```

```
##  [1]    5.40    5.65    5.14    5.14    5.14    5.10    4.70    4.46    4.37
## [10]    4.17    3.89    3.88    4.07    5.11    5.46    5.61    5.85    4.57
## [19]    5.11    6.67    3.96    4.00    4.12    4.99    3.80    3.96    3.93
## [28]    4.02    5.24    5.20    5.67    4.46    4.91    4.79    5.36    3.94
## [37]    4.52    4.35    4.64    3.82    4.24    4.22 9999.00
```

# Removing NAs within functions without affecting data frame

For some functions you can add an argument to specify how to handle NAs. In the case of `mean()` you can and the argument `na.rm` which stands for NA remove. This allows you to do the calculation without alterting the dataset.

```
mean(soils$pH) # can't compute with NAs present
```

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

```
mean(soils$pH, na.rm = TRUE)  # the mean is unreasonably high b/c I added 9999 to
the dataset. This is an example why you need to use NA instead of a chosen value
like 9999 to represent NA!
```

```
## [1] 237.1309
```

```
median(soils$pH, na.rm = TRUE)

## [1] 4.64
```

# Fixing NAs in R

What if you just learned about NAs and need to replace NAs with values in a column? Or perhaps you inhereted a dataset where multiple values were used instead of NA? Here are some R code solutions to save you the time investment and possibility of errors of manually updating your Excel files.

## Replace NAs with data

```
soils$pH[is.na(soils$pH)] <- 0  # using indexing

soils$pH  # NA are replaced with zero.

##  [1]    5.40    5.65    5.14    5.14    5.14    5.10    4.70    4.46    4.37
## [10]    0.00    4.17    3.89    3.88    4.07    0.00    0.00    5.11    5.46
## [19]    5.61    5.85    4.57    5.11    0.00    6.67    3.96    4.00    4.12
## [28]    4.99    3.80    3.96    3.93    4.02    5.24    5.20    0.00    5.67
## [37]    4.46    4.91    4.79    5.36    3.94    4.52    4.35    4.64    3.82
## [46]    4.24    4.22 9999.00
```

## Replace data with NAs

```
soils$pH[soils$pH==9999.00] <- NA  # using indexing

soils$pH # that 9999 value is now NA

##  [1] 5.40 5.65 5.14 5.14 5.14 5.10 4.70 4.46 4.37 0.00 4.17 3.89 3.88 4.07 0.00
## [16] 0.00 5.11 5.46 5.61 5.85 4.57 5.11 0.00 6.67 3.96 4.00 4.12 4.99 3.80 3.96
## [31] 3.93 4.02 5.24 5.20 0.00 5.67 4.46 4.91 4.79 5.36 3.94 4.52 4.35 4.64 3.82
## [46] 4.24 4.22   NA
```