Filtering & Subsetting Data

Author: Cole Brookson **Date:** 20 July 2022

To filter or subset data is to take a dataframe or some other object, and select only part of that object based on some criteria. Most commonly we want to make a new dataframe from an existing dataframe, by filtering the rows of the dataframe based on one or more column.

As with all things data, we provide an option to perform the task using **core R** functions, as well as in the **Tidyverse**.

Filter Based on Single Value

We will use an example dataset here:

```
# remotes package is needed to get the most recent version of the package
install.packages("remotes")
# install package
remotes::install_github("lter/lterdatasampler")

library(lterdatasampler)
library(tidyverse)
```

If we want to filter a dataset based on a single value in a column, we can make use of Boolean operators.

```
# first call the dataset from the lterdatasampler package
crabs <- lterdatasampler::pie_crab
```

Let's look at the columns

```
# this will show us the first six rows
head(crabs)
```

```
## # A tibble: 6 x 9
##   date       latitude site   size air_temp air_temp_sd water_temp water_temp_sd
##   <date>        <dbl> <chr> <dbl>    <dbl>       <dbl>      <dbl>         <dbl>
## 1 2016-07-24       30 GTM    12.4     21.8        6.39       24.5          6.12
## 2 2016-07-24       30 GTM    14.2     21.8        6.39       24.5          6.12
## 3 2016-07-24       30 GTM    14.5     21.8        6.39       24.5          6.12
## 4 2016-07-24       30 GTM    12.9     21.8        6.39       24.5          6.12
## 5 2016-07-24       30 GTM    12.4     21.8        6.39       24.5          6.12
## 6 2016-07-24       30 GTM    13.0     21.8        6.39       24.5          6.12
## # ... with 1 more variable: name <chr>
```

We see there is a column called `site`. We can get the unique values of that column:

```
unique(crabs$site)
```

```
##  [1] "GTM" "SI"  "NIB" "ZI"  "RC"  "VCR" "DB"  "JC"  "CT"  "NB"  "CC"  "BC"
## [13] "PIE"
```

So we might want to filter our data such that we have a new dataframe, where `site` is **only** equal to "VCR".

Core R

Filtering in core R makes use of the square bracket indexing discussed elsewhere. Recall that the index goes *[row,column]*. In the *row* position, we use the `which()` function, which simply returns where some indices are TRUE. Since we want to keep all the columns in our dataset, we leave the *column* position empty, indicating that all columns should be retained. In our example that looks like this:

```
vcr_crabs <- crabs[which(crabs$site == "VCR"),]
```

We can check it worked by calling `unique()` on the new dataframe as above:

```
unique(vcr_crabs$site)
```

```
## [1] "VCR"
```

And we can see it worked as expected.

Tidyverse

Filtering in the Tidyverse uses mostly the `dplyr` package, which is EXTREMELY useful for most data management exercises. Here we use the aptly named `filter()` function to perform the operation.

```
vc_crabs <- crabs %>%
  dplyr::filter(site == "VCR")
```

The `%>%` operator to pass the original data object `crabs` to our function, `filter()`, and the Boolean operator `==` is the only argument needed for this function.

There are many permutations of this single value option with Boolean operators. For example, using either approach, we could filter asking R to select rows where `site == "VCR" | site == "BC"` which would select rows where the site was either "VCR" or "BC".

In the case where we had a separate vector dictating the focal sites we were interested in, we could use that vector in combination with the `%in%` operator to deal with this:

Core R

```
focal_sites <- c("ZI", "RC", "VCR", "DB", "JC")

focal_crabs <- crabs[which(crabs$site %in% focal_sites),]
```

Tidyverse

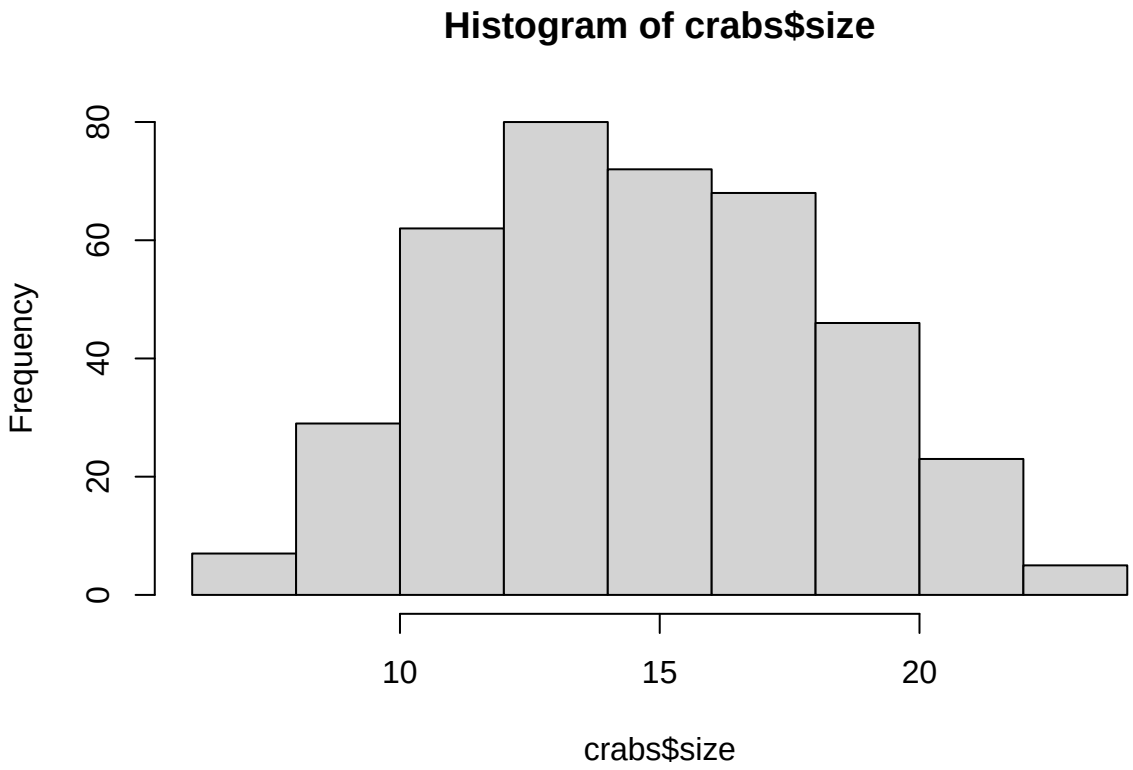
```
focal_sites <- c("ZI", "RC", "VCR", "DB", "JC")

focal_crabs <- crabs %>%
  dplyr::filter(site %in% focal_sites)
```

Filter Based on Multiple Values

This filtering can be extended to use multiple columns relatively simply. There is a variable in this dataset called size.

```
hist(crabs$size)
```



We may want to filter crabs only in our focal sites, but also those above some size (e.g. 15).

Core R

```
site_size <- crabs[which(crabs$site %in% focal_sites &
                           crabs$size > 15), ]
```

Tidyverse

```
site_size <- crabs %>%
  dplyr::filter(site %in% focal_sites & size > 15)
```

Here we just use another logical operator, `&` to link our two conditions together. We can double check:

```
hist(site_size$size)
```


```
unique(site_size$site)
```

```
## [1] "ZI"  "RC"  "VCR" "DB"  "JC"
```

So we can see it worked well.

Filtering Out NA's

For this example, we will use a different dataset.

```
ice <- lterdatasampler::ntl_icecover
head(ice)
```

```
## # A tibble: 6 x 5
##   lakeid       ice_on     ice_off    ice_duration  year
##   <fct>        <date>     <date>            <dbl> <dbl>
## 1 Lake Mendota NA         1853-04-05           NA  1852
## 2 Lake Mendota 1853-12-27 NA                   NA  1853
## 3 Lake Mendota 1855-12-18 1856-04-14          118  1855
## 4 Lake Mendota 1856-12-06 1857-05-06          151  1856
## 5 Lake Mendota 1857-11-25 1858-03-26          121  1857
## 6 Lake Mendota 1858-12-08 1859-03-14           96  1858
```

We can see already that there will be some values of `ice_duration` that will have values of *NA*. To filter those out of the dataset, we can use the `is.na()` function:

Core R

```
trimmed_ice <- ice[which(!is.na(ice$ice_duration)),]
```

Again, we want to keep all the columns present.

Tidyverse

```
trimmed_ice <- ice %>%
  dplyr::filter(!is.na(ice_duration))
```

Here we use the combination of the `!` and `is.na()` to first find the positions where the value is equal to *NA*, and then to remove them with the `!`.

Any conceivable set of criteria can be strung together (carefully!) to filter a dataset with as many conditions as necessary. It is always a good idea however when you are just beginning to use these functions, to work piece by piece, adding conditions one at a time and testing each one to make sure it's doing what you want. Then you can string multiple conditions together without worrying that an error is not being caught.