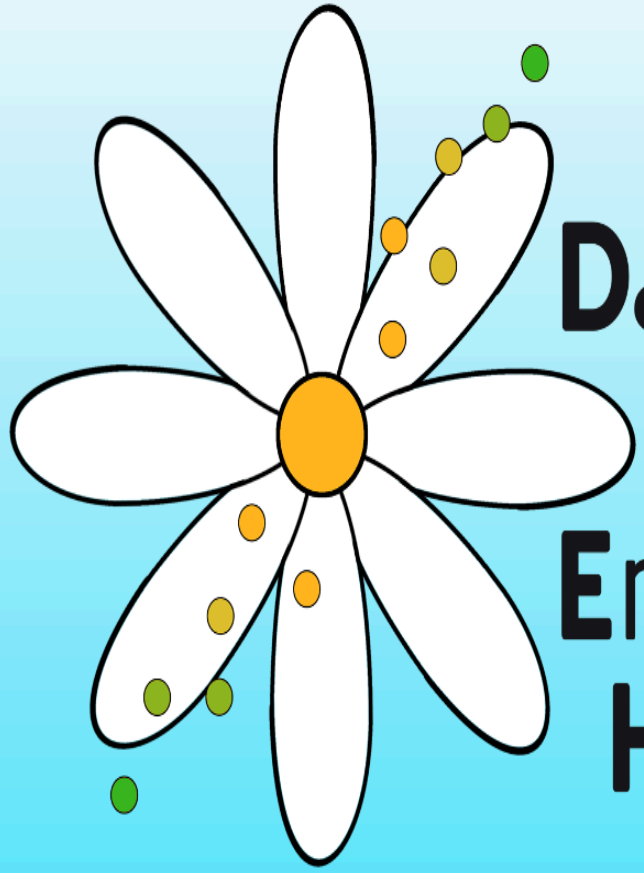
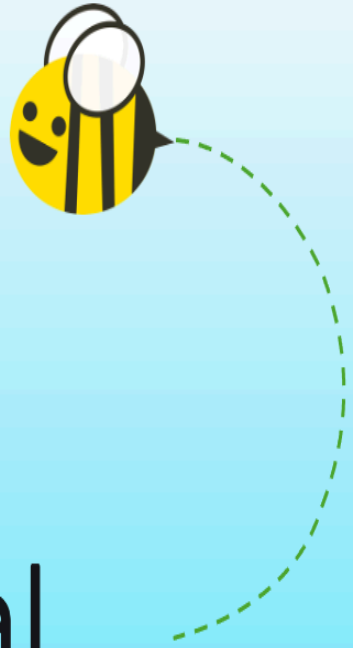


Data Science for Environmental Health



Functions

Writing your own functions

So far we've seen many functions, like `c()`, `class()`, `filter()`, `dim()` ...

Why create your own functions?

- Cut down on repetitive code (easier to fix things!)
- Organize code into manageable chunks
- Avoid running code unintentionally
- Use names that make sense to you

Writing your own functions

The general syntax for a function is:

```
function_name <- function(arg1, arg2, ...) {
 <function body>
}
```

Writing your own functions

Here we will write a function that multiplies some number `x` by 2:

```
div_100 <- function(x) x / 100
```

When you run the line of code above, you make it ready to use (no output yet!). Let's test it!

```
div_100(x = 600)
```

```
[1] 6
```

Writing your own functions: `{ }`

Adding the curly brackets - `{}` - allows you to use functions spanning multiple lines:

```
div_100 <- function(x) {
  x / 100
}
div_100(x = 10)
```

```
[1] 0.1
```

Writing your own functions: **return**

If we want something specific for the function's output, we use `return()`:

```
div_100_plus_4 <- function(x) {
  output_int <- x / 100
  output <- output_int + 4
  return(output)
}
div_100_plus_4(x = 10)
```

```
[1] 4.1
```

Writing your own functions: multiple inputs

Functions can take multiple inputs:

```
div_100_plus_y <- function(x, y) x / 100 + y
div_100_plus_y(x = 10, y = 3)
```

```
[1] 3.1
```

Writing your own functions: multiple outputs

Functions can return a vector (or other object) with multiple outputs.

```
x_and_y_plus_2 <- function(x, y) {
  output1 <- x + 2
  output2 <- y + 2

  return(c(output1, output2))
}
result <- x_and_y_plus_2(x = 10, y = 3)
result
```

```
[1] 12  5
```

Writing your own functions: defaults

Functions can have “default” arguments. This lets us use the function without using an argument later:

```
div_100_plus_y <- function(x = 10, y = 3) x / 100 + y
div_100_plus_y()
```

```
[1] 3.1
```

```
div_100_plus_y(x = 11, y = 4)
```

```
[1] 4.11
```

Writing another simple function

Let's write a function, `sqdif`, that:

1. takes two numbers `x` and `y` with default values of 2 and 3.
2. takes the difference
3. squares this difference
4. then returns the final value

Writing another simple function

```
sqdif <- function(x = 2, y = 3) (x - y)^2
sqdif()
```

```
[1] 1
```

```
sqdif(x = 10, y = 5)
```

```
[1] 25
```

```
sqdif(10, 5)
```

```
[1] 25
```

```
sqdif(11, 4)
```

```
[1] 49
```

Writing your own functions: characters

Functions can have any kind of input. Here is a function with characters:

```
loud <- function(word) {
  output <- rep(toupper(word), 5)
  return(output)
}
loud(word = "hooray!")
```

```
[1] "HOORAY!" "HOORAY!" "HOORAY!" "HOORAY!" "HOORAY!"
```

Functions for tibbles - curly braces

```
# get means and missing for a specific column
get_summary <- function(dataset, col_name) {
    dataset %>%
    summarise(mean = mean({{col_name}}, na.rm = TRUE),
              na_count = sum(is.na({{col_name}})))
}
```

Examples:

```
get_summary(calenviroscreen, CES4.0Score)
```

```
# A tibble: 1 × 2
   mean na_count
  <dbl>    <int>
1  28.3      103
```

```
get_summary(haa5, perc_pop_exposed_to_exceedances)
```

```
# A tibble: 1 × 2
    mean na_count
   <dbl>    <int>
1 0.0591       11
```

Summary

- Simple functions take the form:
  - `NEW_FUNCTION <- function(x, y){x + y}`
  - Can specify defaults like `function(x = 1, y = 2){x + y}` -`return` will provide a value as output
  - `print` will simply print the value on the screen but not save it

Lab Part 1

Class Website

Lab

Functions on multiple columns

Using your custom functions: `sapply()` - a base R function

Now that you've made a function… You can "apply" functions easily with `sapply()`!

These functions take the form:

```
sapply(<a vector, list, data frame>, some_function)
```

CalEnviroScreen

This dataset was gathered by the California Office of Environmental Health Hazard Assessment. CalEnviroScreen ranks census tracts in California based on potential exposures to pollutants, adverse environmental conditions, socioeconomic factors and the prevalence of certain health conditions. Read more at https://calenviroscreen-oehha.hub.arcgis.com/.

```
head(calenviroscreen)
```

```
# A tibble: 6 × 67
  CensusTract CaliforniaCounty   ZIP Longitude Latitude ApproxLocation
        <dbl> <chr>            <int>     <dbl>    <dbl> <chr>
1  6001400100 "Alameda "       94704     -122.     37.9 Oakland
2  6001400200 "Alameda "       94618     -122.     37.8 Oakland
3  6001400300 "Alameda "       94618     -122.     37.8 Oakland
4  6001400400 "Alameda "       94609     -122.     37.8 Oakland
5  6001400500 "Alameda "       94609     -122.     37.8 Oakland
6  6001400600 "Alameda "       94609     -122.     37.8 Oakland
# ℹ 61 more variables: CES4.0Score <dbl>, CES4.0Percentile <dbl>,
#   CES4.0PercRange <chr>, Ozone <dbl>, OzonePctl <dbl>, PM2.5 <dbl>,
#   PM2.5.Pctl <dbl>, DieselPM <dbl>, DieselPMPctl <dbl>, DrinkingWater <dbl>,
#   DrinkingWaterPctl <dbl>, Lead <dbl>, LeadPctl <dbl>, Pesticides <dbl>,
#   PesticidesPctl <dbl>, ToxRelease <dbl>, ToxReleasePctl <dbl>,
#   Traffic <dbl>, TrafficPctl <dbl>, CleanupSites <dbl>,
#   CleanupSitesPctl <dbl>, GroundwaterThreats <dbl>, …
```

Using your custom functions: `sapply( )`

There are no parentheses on the functions!

You can also pipe into your function.

```
sapply(calenviroscreen, class) # also: calenviroscreen %>% sapply(class)
```

```
      CensusTract  CaliforniaCounty               ZIP
        "numeric"       "character"         "integer"
        Longitude          Latitude      ApproxLocation
        "numeric"       "numeric"         "character"
      CES4.0Score CES4.0Percentile   CES4.0PercRange
        "numeric"       "numeric"         "character"
            Ozone         OzonePctl             PM2.5
        "numeric"       "numeric"         "numeric"
         PM2.5.Pctl          DieselPM      DieselPMPctl
        "numeric"       "numeric"         "numeric"
      DrinkingWater DrinkingWaterPctl              Lead
        "numeric"       "numeric"         "numeric"
          LeadPctl        Pesticides    PesticidesPctl
        "numeric"       "numeric"         "numeric"
       ToxRelease   ToxReleasePctl           Traffic
        "numeric"       "numeric"         "numeric"
       TrafficPctl     CleanupSites CleanupSitesPctl
        "numeric"       "numeric"         "numeric"
GroundwaterThreats GroundwaterThreatsPctl         HazWaste
        "numeric"       "numeric"         "numeric"
      HazWastePctl     ImpWaterBodies ImpWaterBodiesPctl
        "numeric"       "integer"         "numeric"
        SolidWaste   SolidWastePctl    PollutionBurden
```

Using your custom functions: **sapply( )**

Use the `div_100` function we created earlier to convert 0-100 percentiles to proportions.

```
calenviroscreen %>%
  select(ends_with("Pctl")) %>%
  sapply(div_100) %>%
  head()
```

```
     OzonePctl PM2.5.Pctl DieselPMPctl DrinkingWaterPctl LeadPctl
[1,]    0.0312     0.3627       0.3476             0.0421   0.0774
[2,]    0.0312     0.4197       0.9271             0.0421   0.6820
[3,]    0.0312     0.4390       0.8977             0.0421   0.6418
[4,]    0.0312     0.4281       0.7910             0.0421   0.6708
[5,]    0.0312     0.4281       0.6758             0.0421   0.6795
[6,]    0.0312     0.4281       0.8376             0.0421   0.6970
     PesticidesPctl ToxReleasePctl TrafficPctl CleanupSitesPctl
[1,]              0         0.5603     0.5594           0.5817
[2,]              0         0.5543     0.3749           0.0000
[3,]              0         0.5504     0.4248           0.1183
[4,]              0         0.5590     0.3800           0.0000
[5,]              0         0.5648     0.4868           0.3387
[6,]              0         0.5565     0.6706           0.2262
     GroundwaterThreatsPctl HazWastePctl ImpWaterBodiesPctl SolidWastePctl
[1,]                 0.5242       0.9252              0.2388         0.3572
[2,]                 0.8793       0.2851              0.0000         0.0000
[3,]                 0.8529       0.7407              0.0000         0.0000
[4,]                 0.9256       0.5189              0.0000         0.0000
[5,]                 0.8434       0.5640              0.0000         0.0000
[6,]                 0.7906       0.5827              0.0000         0.0000
```

Using your custom functions "on the fly" to iterate

Also called "anonymous function".

```
calenviroscreen %>%
  select(ends_with("Pctl")) %>%
  sapply(function(x) x / 100) %>%
  head()
```

```
     OzonePctl PM2.5.Pctl DieselPMPctl DrinkingWaterPctl LeadPctl
[1,]    0.0312     0.3627       0.3476             0.0421   0.0774
[2,]    0.0312     0.4197       0.9271             0.0421   0.6820
[3,]    0.0312     0.4390       0.8977             0.0421   0.6418
[4,]    0.0312     0.4281       0.7910             0.0421   0.6708
[5,]    0.0312     0.4281       0.6758             0.0421   0.6795
[6,]    0.0312     0.4281       0.8376             0.0421   0.6970
     PesticidesPctl ToxReleasePctl TrafficPctl CleanupSitesPctl
[1,]              0         0.5603      0.5594           0.5817
[2,]              0         0.5543      0.3749           0.0000
[3,]              0         0.5504      0.4248           0.1183
[4,]              0         0.5590      0.3800           0.0000
[5,]              0         0.5648      0.4868           0.3387
[6,]              0         0.5565      0.6706           0.2262
     GroundwaterThreatsPctl HazWastePctl ImpWaterBodiesPctl SolidWastePctl
[1,]                 0.5242       0.9252              0.2388         0.3572
[2,]                 0.8793       0.2851              0.0000         0.0000
[3,]                 0.8529       0.7407              0.0000         0.0000
[4,]                 0.9256       0.5189              0.0000         0.0000
[5,]                 0.8434       0.5640              0.0000         0.0000
[6,]                 0.7906       0.5827              0.0000         0.0000
     PollutionBurdenPctl AsthmaPctl LowBirthWeightPctl                  /
[1,]              0.2662     0.0444             0.2306
```

Anonymous functions: alternative syntax

```
calenviroscreen %>%
  select(ends_with("Pctl")) %>%
  sapply(\(x) x / 100) %>%
  head()
```

```
     OzonePctl PM2.5.Pctl DieselPMPctl DrinkingWaterPctl LeadPctl
[1,]    0.0312     0.3627       0.3476             0.0421   0.0774
[2,]    0.0312     0.4197       0.9271             0.0421   0.6820
[3,]    0.0312     0.4390       0.8977             0.0421   0.6418
[4,]    0.0312     0.4281       0.7910             0.0421   0.6708
[5,]    0.0312     0.4281       0.6758             0.0421   0.6795
[6,]    0.0312     0.4281       0.8376             0.0421   0.6970
     PesticidesPctl ToxReleasePctl TrafficPctl CleanupSitesPctl
[1,]              0         0.5603     0.5594           0.5817
[2,]              0         0.5543     0.3749           0.0000
[3,]              0         0.5504     0.4248           0.1183
[4,]              0         0.5590     0.3800           0.0000
[5,]              0         0.5648     0.4868           0.3387
[6,]              0         0.5565     0.6706           0.2262
     GroundwaterThreatsPctl HazWastePctl ImpWaterBodiesPctl SolidWastePctl
[1,]                 0.5242       0.9252              0.2388         0.3572
[2,]                 0.8793       0.2851              0.0000         0.0000
[3,]                 0.8529       0.7407              0.0000         0.0000
[4,]                 0.9256       0.5189              0.0000         0.0000
[5,]                 0.8434       0.5640              0.0000         0.0000
[6,]                 0.7906       0.5827              0.0000         0.0000
     PollutionBurdenPctl AsthmaPctl LowBirthWeightPctl
[1,]              0.2662     0.0444             0.2306
[2,]              0.2418     0.0980             0.2792
[3,]              0.3337     0.2657             0.2162
```

across

Using functions in `mutate()` and `summarize()`

Already know how to use functions to modify columns using `mutate()` or calculate summary statistics using `summarize()`.

- Pesticides - pounds of selected active pesticide / square mile
- Poverty - percent of population living below two times the federal poverty level
- LowBirthWeight - Percent low birth weight

```
calenviroscreen %>%
  summarize(max_pest = max(Pesticides, na.rm = T),
            max_pov = max(Poverty, na.rm = T),
            low_bw = max(LowBirthWeight, na.rm = T))
```

```
# A tibble: 1 × 3
  max_pest max_pov low_bw
     <dbl>   <dbl>  <dbl>
1   80811.    96.7   13.7
```

Applying functions with **across** from **dplyr**

`across()` makes it easy to apply the same transformation to multiple columns. Usually used with `summarize()` or `mutate()`.

```
summarize(across(<columns>,function))
```

or

```
mutate(across(<columns>,function))
```

- List columns first : `.cols =`
- List function next: `.fns =`
- If there are arguments to a function (e.g., `na.rm = TRUE`), use an anonymous function.

Applying functions with **across** from **dplyr**

Combining with `summarize()`

```
calenviroscreen %>%
  summarize(across(
    c(Pesticides, Poverty, LowBirthWeight),
    mean # no parentheses
  ))
```

```
# A tibble: 1 × 3
  Pesticides Poverty LowBirthWeight
       <dbl>   <dbl>           <dbl>
1       268.      NA              NA
```

Applying functions with **across** from **dplyr**

Add anonymous function to include additional arguments (e.g., `na.rm = T`).

```
calenviroscreen %>%
  summarize(across(
    c(Pesticides, Poverty, LowBirthWeight),
    function(x) mean(x, na.rm = T)
  ))
```

```
# A tibble: 1 × 3
  Pesticides Poverty LowBirthWeight
       <dbl>   <dbl>           <dbl>
1       268.    31.3            5.00
```

Applying functions with **across** from **dplyr**

Can use with other tidyverse functions like `group_by`!

```
calenviroscreen %>%
  group_by(CaliforniaCounty) %>%
  summarize(across(
    c(Pesticides, Poverty, LowBirthWeight),
    function(x) mean(x, na.rm = T)
  ))
```

```
# A tibble: 58 × 4
   CaliforniaCounty Pesticides Poverty LowBirthWeight
   <chr>                 <dbl>   <dbl>          <dbl>
 1 "Alameda "            0.948    22.1           5.26
 2 "Alpine "             0        38.9         NaN
 3 "Amador "             2.01     25.2           4.45
 4 "Butte "            736.       39.9           4.64
 5 "Calaveras "          1.20     28.2           3.55
 6 "Colusa "          1186.       36.5           4.11
 7 "Contra Costa"       10.5      20.6           4.71
 8 "Del Norte"          47.4      48.4         NaN
 9 "El Dorado"           5.50     20.9           4.28
10 "Fresno "           586.       45.8           5.96
# ℹ 48 more rows
```

Applying functions with **across** from **dplyr**

Using different `tidyselect()` options (e.g., `starts_with()`, `ends_with()`, `contains()`)

```
calenviroscreen %>%
  group_by(CaliforniaCounty) %>%
  summarize(across(contains("PM"), mean))
```

```
# A tibble: 58 × 5
   CaliforniaCounty PM2.5 PM2.5.Pctl DieselPM DieselPMPctl
   <chr>            <dbl>      <dbl>    <dbl>        <dbl>
 1 "Alameda "        8.87      31.9    0.350         66.4
 2 "Alpine "         3.05       0.07   0.003          1.02
 3 "Amador "         8.01      18.9    0.0111         4.18
 4 "Butte "          8.22      23.0    0.106         33.1
 5 "Calaveras "      8.12      22.6    0.0079         3.08
 6 "Colusa "         7.54      13.3    0.0292        10.6
 7 "Contra Costa"    8.76      31.0    0.210         48.6
 8 "Del Norte"       5.71       2.93   0.0301        11.1
 9 "El Dorado"       6.78       8.91   0.0380        13.6
10 "Fresno "        13.2       91.4    0.181         44.8
# ℹ 48 more rows
```

Applying functions with **across** from **dplyr**

Combining with `mutate()` - the `replace_na` function

Here we will use the `yearly_co2_emissions` data from `dasehr`

`replace_na({data frame}, {list of values})` or `replace_na({vector}, {single value})`

```
yearly_co2_emissions %>%
  select(country, starts_with("194")) %>%
  mutate(across(
    c(`1943`, `1944`, `1945`),
    function(x) replace_na(x, replace = 0)
  ))
```

```
# A tibble: 192 × 11
   country        `1940` `1941` `1942` `1943` `1944` `1945` `1946` `1947` `1948`
   <chr>           <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
 1 Afghanistan        NA     NA     NA      0      0      0     NA     NA     NA
 2 Albania           693    627    744    462    154    121    484    928    704
 3 Algeria           238    312    499    469    499    616    763    744    803
 4 Andorra            NA     NA     NA      0      0      0     NA     NA     NA
 5 Angola             NA     NA     NA      0      0      0     NA     NA     NA
 6 Antigua and B…     NA     NA     NA      0      0      0     NA     NA     NA
 7 Argentina       15900  14000  13500  14100  14000  13700  13700  14500  17400
 8 Armenia           848    745    513    655    613    649    730    878    935
 9 Australia       29100  34600  36500  35000  34200  32700  35500  38000  38500
10 Austria          7350   7980   8560   9620   9400   4570  12800  17600  24500
# ℹ 182 more rows
# ℹ 1 more variable: `1949` <dbl>
```

`purrr` package

Similar to across, `purrr` is a package that allows you to apply a function to multiple columns in a data frame or multiple data objects in a list.

While we won't get into `purrr` too much in this class, its a handy package for you to know about should you get into a situation where you have an irregular list you need to handle!

Multiple Data Frames

Multiple data frames

Lists help us work with multiple data frames

```
AQ_list <- list(AQ1 = airquality, AQ2 = airquality, AQ3 = airquality)
str(AQ_list)
```

```
List of 3
 $ AQ1:'data.frame':    153 obs. of  6 variables:
  ..$ Ozone  : int [1:153] 41 36 12 18 NA 28 23 19 8 NA ...
  ..$ Solar.R: int [1:153] 190 118 149 313 NA NA 299 99 19 194 ...
  ..$ Wind   : num [1:153] 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
  ..$ Temp   : int [1:153] 67 72 74 62 56 66 65 59 61 69 ...
  ..$ Month  : int [1:153] 5 5 5 5 5 5 5 5 5 5 ...
  ..$ Day    : int [1:153] 1 2 3 4 5 6 7 8 9 10 ...
 $ AQ2:'data.frame':    153 obs. of  6 variables:
  ..$ Ozone  : int [1:153] 41 36 12 18 NA 28 23 19 8 NA ...
  ..$ Solar.R: int [1:153] 190 118 149 313 NA NA 299 99 19 194 ...
  ..$ Wind   : num [1:153] 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
  ..$ Temp   : int [1:153] 67 72 74 62 56 66 65 59 61 69 ...
  ..$ Month  : int [1:153] 5 5 5 5 5 5 5 5 5 5 ...
  ..$ Day    : int [1:153] 1 2 3 4 5 6 7 8 9 10 ...
 $ AQ3:'data.frame':    153 obs. of  6 variables:
  ..$ Ozone  : int [1:153] 41 36 12 18 NA 28 23 19 8 NA ...
  ..$ Solar.R: int [1:153] 190 118 149 313 NA NA 299 99 19 194 ...
  ..$ Wind   : num [1:153] 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
  ..$ Temp   : int [1:153] 67 72 74 62 56 66 65 59 61 69 ...
  ..$ Month  : int [1:153] 5 5 5 5 5 5 5 5 5 5 ...
  ..$ Day    : int [1:153] 1 2 3 4 5 6 7 8 9 10 ...
```

Multiple data frames: **sapply**

```
AQ_list %>% sapply(class)
```

```
         AQ1          AQ2          AQ3
"data.frame" "data.frame" "data.frame"
```

```
AQ_list %>% sapply(nrow)
```

```
AQ1 AQ2 AQ3
153 153 153
```

```
AQ_list %>% sapply(colMeans, na.rm = TRUE)
```

```
              AQ1        AQ2        AQ3
Ozone    42.129310  42.129310  42.129310
Solar.R 185.931507 185.931507 185.931507
Wind      9.957516   9.957516   9.957516
Temp     77.882353  77.882353  77.882353
Month     6.993464   6.993464   6.993464
Day      15.803922  15.803922  15.803922
```

Summary

- Apply your functions with `sapply(<a vector or list>, some_function)`
- Use `across()` to apply functions across multiple columns of data
- Need to use `across` within `summarize()` or `mutate()`
- Can use `sapply` or `purrr` to work with multiple data frames within lists simultaneously

Lab Part 2

Class Website

Lab

Image by Gerd Altmann from Pixabay