

## Accessing Data

# Datasets Available in Base R

Some datasets are easily available in R and can be accessed with the `data` function.

```
data(rivers)
```

```
rivers[1:5]
```

```
[1] 735 320 325 392 524
```

We can get a list of all available datasets by entering `data()` into the console.

Note: Base R refers to R with no packages loaded.

## Using the data Function

When we first `data(rivers)`, R creates one or more objects that have value `<Promise>`.

Once we have actually used the objects, the contents become available in the environment.

Try it out to see!

## data Examples

- ▶ islands
- ▶ volcano
- ▶ WorldPhones

What types of objects are contained in this data?

# The data Function Can Create Multiple Objects

How many and what types of objects are contained in the euro data?

# The Data Frame

```
data(stackloss)
```

```
str(stackloss)
```

```
'data.frame':  21 obs. of  4 variables:
 $ Air.Flow   : num  80 80 75 62 62 62 62 62 58 58 ...
 $ Water.Temp: num  27 27 25 24 22 23 24 24 23 18 ...
 $ Acid.Conc.: num  89 88 90 87 87 87 93 93 87 80 ...
 $ stack.loss: num  42 37 37 28 18 18 19 20 15 14 ...
```

```
str(stack.x)
```

```
num [1:21, 1:3] 80 80 75 62 62 62 62 62 58 58 ...
- attr(*, "dimnames")=List of 2
 ..$ : NULL
 ..$ : chr [1:3] "Air.Flow" "Water.Temp" "Acid.Conc."
```

```
str(stack.loss)
```

```
num [1:21] 42 37 37 28 18 18 19 20 15 14 ...
```

# Data Frames are Like Matrices

For the data frame `stackloss`:

- ▶ Find number of rows
- ▶ Find number of columns
- ▶ Find column means
- ▶ Find column standard deviations
- ▶ Extract the first 3 columns
- ▶ Extract the last row

# Data Frames are Not Like Matrices

We cannot do linear algebra on data frames without transforming them.

```
rep(1/nrow(stackloss), nrow(stackloss))%*%stackloss
```

Data frames can contain columns of different modes.

Columns of data frames can be accessed using \$.

```
stackloss$stack.loss
```

Using \$ creates a vector - we can confirm.

```
all.equal(stackloss$stack.loss, stack.loss)
```



## Can we turn a data frame back into a matrix?

Sometimes!

```
rep(1/nrow(stackloss), nrow(stackloss))%*%  
  as.matrix(stackloss)
```

```
      Air.Flow Water.Temp Acid.Conc. stack.loss  
[1,] 60.42857  21.09524   86.28571   17.52381
```

This will only work when the data frame does not contain characters or factors (which are a special way of storing characters that we're about to learn about).

## How do we make a data frame?

We can construct a data frame by combining matrices with  $n$  rows and vectors with  $n$  elements.

```
sl <- data.frame(stack.x, stack.loss)
```

We can confirm they are the same!

```
[1] TRUE
```

## Data Frames with Non-Numeric Data

Again, data frames can contain columns of different modes.

```
data(chickwts)
```

```
head(chickwts)
```

	weight	feed
1	179	horsebean
2	160	horsebean
3	136	horsebean
4	227	horsebean
5	217	horsebean
6	168	horsebean

You can also look at a data frame as if it is a spreadsheet in R by using the View function. Try it!

```
View(chickwts)
```

# Factors

What appears to be a character vector is actually stored as a “factor.”

```
str(chickwts)
```

```
'data.frame':   71 obs. of  2 variables:  
 $ weight: num  179 160 136 227 217 168 108 124 143 140 ...  
 $ feed : Factor w/ 6 levels "casein","horsebean",...: 2 2
```

We will talk about this more in a bit.

# Datasets Available in Packages

Different packages contain additional datasets, often for use as a demonstration of certain functions.

There are also some R packages that exist for the sole purpose of helping you load in specific datasets from online sources, e.g. `yahoofinancer` for downloading Yahoo Finance data.

# Loading in Datasets

In real life, we probably want to load in data that's not already available in base R or some R package.

The most common format is a `.csv` file, where `.csv` stands for “comma separated value.” These are often directly available or indirectly available, e.g. as an option for saving an spreadsheet in Excel.

The downside of `.csv` files is that they can be a bit big - we'll talk about dealing with big files soon.

## Downloading a .csv file

Let's download files from a recent Evolution paper that have been made available:

<https://datadryad.org/dataset/doi:10.5061/dryad.rs610>.

The paper is available here, if you're curious:

<https://academic.oup.com/evolut/article/69/10/2662/6851963>.

# Filepaths

In order to read in data, you need to figure out where the data is.

In R, you can identify the current file path using `getwd()`.

```
getwd()
```

You can see what files are there using `list.files()`.

```
list.files()
```

You can also set a new working directory using the `setwd` function and providing a file path.



## Filepath Help

If you're having a hard time finding your file path, you can load in a .csv file by going to the "File" menu, navigating down to "Import Dataset" and clicking "From Text (base)." Then find your .csv file.

Once you do this, a line of code that starts with `read.csv` will get sent to your console. Save it! It will include the path to your file.

# Reading in a .csv file

The `read.csv` function reads a .csv file into R, creating a data frame.

```
data <- read.csv("~/Downloads/seawater.archive.data.csv")
```

```
str(data)
```

```
'data.frame':  63 obs. of  9 variables:
 $ Plate           : int  1 1 1 1 1 1 2 2 2 2 ...
 $ Well.Name       : chr  "B2" "C2" "D2" "E2" ...
 $ Line           : chr  "31" "22" "52" "43" ...
 $ Concentration   : chr  "seawater" "seawater" "seawater" "seawate
 $ Replicate       : int  1 1 1 1 1 1 2 2 2 2 ...
 $ History         : chr  "salt" "salt" "dark" "dark" ...
 $ initial.cell.density: num  41748 54755 15594 224336 15070 ...
 $ final.cell.density : num  1119 10315 874 3077 455 ...
 $ rate.increase   : num  -1.688 0.262 -0.95 -2.358 -1.57 ...
```

# Characters versus Factors

```
data <- read.csv("~/Downloads/seawater.archive.data.csv",  
                 stringsAsFactors = TRUE)
```

```
str(data)
```

```
'data.frame':  63 obs. of  9 variables:  
 $ Plate           : int  1 1 1 1 1 1 2 2 2 2 ...  
 $ Well.Name       : Factor w/ 6 levels "B2","C2","D2",...: 1 2 3 4  
 $ Line           : Factor w/ 33 levels "20","21","22",...: 13 3 26  
 $ Concentration   : Factor w/ 1 level "seawater": 1 1 1 1 1 1 1 1  
 $ Replicate       : int  1 1 1 1 1 1 2 2 2 2 ...  
 $ History         : Factor w/ 4 levels "dark","marine",...: 3 3 1 1  
 $ initial.cell.density: num  41748 54755 15594 224336 15070 ...  
 $ final.cell.density : num  1119 10315 874 3077 455 ...  
 $ rate.increase    : num  -1.688 0.262 -0.95 -2.358 -1.57 ...
```

## What the heck is a factor??

Factors are a mode that we haven't talked about yet. They can be thought of as fancy vectors.

Factors are a way of storing elements as positive integers with each integer value associated with a character label. The character labels are called “levels.”

Generally, factors are annoying. However a nice thing about them is that they can clearly convey the total set of values that a variable could take on, even if certain values are not observed in the data. They are also sometimes convenient for plotting, summarizing, and analyzing data.

## An Example of a Factor

The variable History is treated as a level when we specify `stringsAsFactors = TRUE`.

```
levels(data$History)
```

```
[1] "dark"    "marine"  "salt"    "wild"
```

```
unclass(data$History)
```

```
[1] 3 3 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 3 3 3 3 1 1 1 1 1  
[39] 1 1 3 3 3 3 3 1 1 1 1 1 1 3 3 3 3 3 1 2 1 4 3 3 2  
attr(,"levels")
```

```
[1] "dark"    "marine"  "salt"    "wild"
```

```
as.numeric(data$History)
```

```
[1] 3 3 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 3 3 3 3 1 1 1 1 1  
[39] 1 1 3 3 3 3 3 1 1 1 1 1 1 3 3 3 3 3 1 2 1 4 3 3 2
```

## Converting a Factor to its Values

```
levels(data$History)[as.numeric(data$History)]
```

[1]	"salt"	"salt"	"dark"	"dark"	"dark"	"dark"
[9]	"dark"	"dark"	"salt"	"salt"	"dark"	"dark"
[17]	"dark"	"dark"	"salt"	"salt"	"salt"	"salt"
[25]	"dark"	"dark"	"dark"	"dark"	"salt"	"salt"
[33]	"dark"	"dark"	"dark"	"dark"	"dark"	"dark"
[41]	"salt"	"salt"	"salt"	"salt"	"salt"	"dark"
[49]	"dark"	"dark"	"dark"	"salt"	"salt"	"salt"
[57]	"dark"	"marine"	"dark"	"wild"	"salt"	"salt"