Data in R

Today's agenda:

Discuss loading data, data types, etc.,

Pointers on data

Load and examine some data

Electronic Formats

R can handle most common data formats

E.g., .csv, .txt

Proprietary data formats may require additional packages

E.g., Excel .xlsx files, ESRI .shp files

R has its own file types:

E.g., .RDS

Different pros/cons to data formats!

Getting data into R

- Functions used will depend on data types
- Often multiple options for a given file type
- Functions can mess things up when importing them!
- R can pull from different places (online, locally, databases, etc.)

Getting data into R: example functions

File type	Extension(s)	Common R import function(s)	Package(s)
Comma-separated values	`.csv`	`read.csv()`, `read_csv()`	base, readr
Tab-delimited text	`.tsv`, `.txt`	`read.delim()`, `read_tsv()`	base, readr
Excel (old)	`.xls`	`read.xls()`, `read_excel()`	gdata, readxl
Excel (modern)	`.xlsx`	`read.xlsx()`, `read_excel()`	openxlsx, readxl
R object (binary)	`.RData`, `.rda`	`load()`	base
R object (single)	`.RDS`	`readRDS()`	base
SPSS	`.sav`	`read.spss()`, `read_sav()`	foreign, haven
Stata	`.dta`	`read.dta()`, `read_dta()`	foreign, haven
SAS	`.sas7bdat`, `.xpt`	`read_sas()`, `read.xport()`	haven, sas7bdat, foreign
JSON	`.json`	`fromJSON()`	jsonlite
XML	`.xml`	`xmlParse()`, `read_xml()`	XML, xml2
Feather/Arrow	`.feather`, `.arrow`	`read_feather()`, `read_parquet()`	arrow
HDF5	`.h5`	`h5read()`	rhdf5
NetCDF	`.nc`	`nc_open()`	ncdf4
SQLite database	`.sqlite`, `.db`	`dbConnect(RSQLite::SQLite(), ...)`	RSQLite
Shapefile (GIS)	`.shp`	`st_read()`	sf
GeoPackage	`.gpkg`	`st_read()`	sf

Getting data into R: example

1. Go to the course Github site https://github.com/bmaitner/Statistical_ecology_course
2. Navigate to data/Avonet
3. Download the file “AVONET1_BirdLife.csv”

Getting data into R: example

- Since the file “AVONET1_BirdLife.csv” is a csv, we’ll use `read.csv()`

- We can do this using
  - Absolute paths
  - Relative paths
  - A remote copy of the file

Getting data into R: relative vs absolute paths

Absolute vs relative paths

- Absolute: path from the root of your file system
- Relative: path relative to your working directory

Getting data into R: relative vs absolute paths

Absolute vs relative paths

- Absolute: path from the root of your file system
- Relative: path relative to your working directory

Examples: which is which?

“C:/Users/Brian Maitner/Desktop/current_projects/Statistical_ecology_course/data/ Avonet/ AVONET1_BirdLife.csv”

“data/Avonet/AVONET1_BirdLife.csv”

Getting data into R: relative vs absolute paths

Try to read in the file you downloaded (AVONET1_BirdLife.csv)

- Try reading it using both relative and absolute paths
- Use `read.csv()`

Getting data into R: remote data

Many R functions can also read in online data

For example, you can read in the Avonet data directly from Github

```
avonet_v3 <-
read.csv("https://github.com/bmaitner/Statistical_ecology_course/raw/refs/heads/main/data/Avonet/AVONET1_BirdLife.csv")
```

Give it a try!

Data Types in R

In R, all data have a type, e.g.:

- Numeric (2005)
- Logical (TRUE)
- Character (“Optimus Prime”)
- Factor (“Small” vs “Medium” vs “Large”)

Data Types in R

In R, all data have a type, e.g.:

- Numeric (2005)
- Logical (TRUE)
- Character (“Optimus Prime”)
- Factor (“Small” vs “Medium” vs “Large”)

To figure out what type something is, use `class()`

Data Types in R

Try `class()` out on a few things

```
class("something")

class(1)

class(1L)

class(TRUE)
```

Data Types in R

R tries to guess what type things are, but sometimes get it very wrong!

- Numeric vs logical
- Date vs character or number
- Factor vs Numeric

These mistakes can cause big problems in analyses!

It’s important to check your data!

Higher data types

A single value is often not very useful, often we want sets of data:

- Vector
  - the most fundamental structure;
  - an ordered collection of elements of the same type.
- Matrix
  - a 2D vector, all elements must be the same type.
- Array
  - an n-dimensional generalization of a matrix.

Higher data types

- List
  - ordered collection of objects that can be of different types or lengths (a very flexible container).
- Data.frame
  - tabular data; essentially a list of equal-length vectors, often used for datasets.
- Tibble
  - a modern variant of data.frame with cleaner printing and stricter rules.

Higher data types

You can also use `class()` with these higher data types

> Try this and see what class the Avonet data you loaded earlier is

Accessing data within an object

For a vector or a list, you can access elements by their **position**:

```
test_vector <- c("Optimus Prime","Megatron")

test_vector[[1]]

test_vector[1]
```

Accessing data within an object

For a vector or a list, you can access elements by their **name**:

```
test_vector <- c("A"="Optimus Prime","B"="Megatron")

test_vector[[A]]

test_vector[A]
```

Accessing data within an object

For a vector or a list, you can access elements by their **name**:

```
test_vector <- c("A"="Optimus Prime","B"="Megatron")

test_vector[[A]]

test_vector[A]
```

You can find the names with the function `names()`

```
names(test_vector)
```

Accessing data within an object

For matrices, data.frames, and arrays there are more options!

- Rows (name or number)
- Columns (name or number)
- Row and column (name or number)

Accessing data within an object

Let’s use the Avonet data from earlier

To access the species information, we can use:

```
Avonet$Species1
```

```
Avonet[1]
```

```
Avonet[,1]
```

Accessing data within an object

Let’s use the Avonet data from earlier

To access the information for a particular row:

```
Avonet[1,]
```

```
Avonet[“1”,]
```
(note: only works because the row is named “1”)

Checking data with in the Avonet dataset

Use `class()` to check the data type in a column

```
class(avonet$Species1)

class(avonet[,2])
```

Checking data with str()

Can check all the fields using the structure command, `str()`

```
str(avonet)
```

Try it out!

Other quick ways to check objects

`str()` - gives info on object structure

`summary()` - provides summary information that varies by column class

`head()` - shows the first few rows of data

`table()` - used for categorical variables, shows how often combinations occur

`rownames()` - row names

`colnames()` - column names

Try these out!

What if R get the type wrong?

You can convert between many different types using “as” commands

```
as.factor()

as.character()

as.data.frame()

as.matrix()
```

What if R get the type wrong?

You can convert between many different types using “as” commands

```
as.factor()

as.character()

as.data.frame()

as.matrix()
```

Dealing with NAs

- You can remove any record that contains an NA using `na.omit()`

- Some functions have arguments for handling NA values
  - na.rm in `mean()`
  - na.action in `lm()`
  - use in `cor()`

- You can turn the NA values into other values (e.g, 0)
  - Only do this if it makes sense (e.g., if you tally the number of species seen in a day)
  - We'll discuss how to do this later

Next time:

Before class:

- read 2.4 - 2.5

During class:

- Discuss 2.4 - 2.5
- Discuss data we’re interested in
- Work through 2.6