Data I/O + Structure
What did I just read in?

- `nrow()` displays the number of rows of a data frame
- `ncol()` displays the number of columns
- `dim()` displays a vector of length 2: # rows, # columns

```
dim(ufo)
[1] 88875 11
	nrow(ufo)
[1] 88875
	ncol(ufo)
[1] 11
```
**All Column Names**

- `colnames()` displays the column names

```r
colnames(ufo)

<table>
<thead>
<tr>
<th>1</th>
<th>datetime</th>
<th>4</th>
<th>country</th>
<th>7</th>
<th>duration (hours/min)</th>
<th>10</th>
<th>latitude</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&quot;datetime&quot;</td>
<td></td>
<td>&quot;country&quot;</td>
<td></td>
<td>&quot;duration (hours/min)&quot;</td>
<td></td>
<td>&quot;latitude&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;city&quot;</td>
<td></td>
<td>&quot;shape&quot;</td>
<td></td>
<td>&quot;comments&quot;</td>
<td></td>
<td>&quot;longitude&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;state&quot;</td>
<td></td>
<td></td>
<td></td>
<td>&quot;duration (seconds)&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;date posted&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

3/21
Data Input

- Sometimes you get weird messages when reading in data.
- The `problems()` function shows you any issues with the data read-in.

```r
problems(ufo)
```

# A tibble: 199 x 5

<table>
<thead>
<tr>
<th>row</th>
<th>col</th>
<th>expected</th>
<th>actual</th>
<th>file</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;int&gt;</td>
<td>&lt;chr&gt;</td>
<td>&lt;chr&gt;</td>
<td>&lt;chr&gt;</td>
<td>&lt;chr&gt;</td>
</tr>
<tr>
<td>1</td>
<td>877</td>
<td>&lt;NA&gt;</td>
<td>11 columns</td>
<td>12 columns</td>
</tr>
<tr>
<td>2</td>
<td>1712</td>
<td>&lt;NA&gt;</td>
<td>11 columns</td>
<td>12 columns</td>
</tr>
<tr>
<td>3</td>
<td>1814</td>
<td>&lt;NA&gt;</td>
<td>11 columns</td>
<td>12 columns</td>
</tr>
<tr>
<td>4</td>
<td>2857</td>
<td>&lt;NA&gt;</td>
<td>11 columns</td>
<td>12 columns</td>
</tr>
<tr>
<td>5</td>
<td>3733</td>
<td>&lt;NA&gt;</td>
<td>11 columns</td>
<td>12 columns</td>
</tr>
<tr>
<td>6</td>
<td>4755</td>
<td>&lt;NA&gt;</td>
<td>11 columns</td>
<td>12 columns</td>
</tr>
<tr>
<td>7</td>
<td>5388</td>
<td>&lt;NA&gt;</td>
<td>11 columns</td>
<td>12 columns</td>
</tr>
<tr>
<td>8</td>
<td>5422</td>
<td>&lt;NA&gt;</td>
<td>11 columns</td>
<td>12 columns</td>
</tr>
<tr>
<td>9</td>
<td>5613</td>
<td>&lt;NA&gt;</td>
<td>11 columns</td>
<td>12 columns</td>
</tr>
<tr>
<td>10</td>
<td>5848</td>
<td>&lt;NA&gt;</td>
<td>11 columns</td>
<td>12 columns</td>
</tr>
</tbody>
</table>

# ... with 189 more rows

```r
dim(problems(ufo))
```

[1] 199 5
Data Input

- The `spec()` functions show you the specification of how the data was read in, `cols_condense` can help with this.

```r
spec(ufo)

cols(
    datetime = col_character(),
    city = col_character(),
    state = col_character(),
    country = col_character(),
    shape = col_character(),
    `duration (seconds)` = col_double(),
    `duration (hours/min)` = col_character(),
    comments = col_character(),
    `date posted` = col_character(),
    latitude = col_character(),
    longitude = col_double()
)

cols_condense(spec(ufo))

cols(
    .default = col_character(),
    `duration (seconds)` = col_double(),
    longitude = col_double()
)
```
Data Input

This specification is passed to `readr` functions:

```r
ufi_char = read_csv("../data/ufo/ufo_data_complete.csv", col_types = cols(
  .default = col_character(),
  longitude = col_double()
))
```

Warning: 196 parsing failures.

<table>
<thead>
<tr>
<th>row</th>
<th>col</th>
<th>expected columns</th>
<th>actual columns</th>
<th>file</th>
</tr>
</thead>
<tbody>
<tr>
<td>877</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>../data/ufo/ufo_data_complete.csv</td>
</tr>
<tr>
<td>1712</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>../data/ufo/ufo_data_complete.csv</td>
</tr>
<tr>
<td>1814</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>../data/ufo/ufo_data_complete.csv</td>
</tr>
<tr>
<td>2857</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>../data/ufo/ufo_data_complete.csv</td>
</tr>
<tr>
<td>3733</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>../data/ufo/ufo_data_complete.csv</td>
</tr>
</tbody>
</table>

See `problems(...)` for more details.

```r
dim(problems(ufo))
```

```
[1] 199 5
```

```r
dim(problems(ufo_char))
```

```
[1] 196 5
```
Data Input: Checking for problems

- The `stop_for_problems()` function will stop if your data had an error when reading in. If this occurs, you can either use `col_types` (from `spec()`) for the problematic columns, or set `guess_max = Inf` (takes much longer):

  `stop_for_problems(ufo)`
The `vis_dat` function can give you an overview

```r
library(visdat) # only so many rows can be visualized are good
dojo_samp <- dojo %>% sample_n(size = 1000)
vis_dat(dojo_samp)
```
Missing data with the `vizdat/naniar` packages

`vis_miss(ufo_samp)`
Missing data with the `vizdat/naniar` packages

```
library(naniar)
gg_miss_var(ufo)
```
Missing data with the `vizdat/naniar` packages

`miss_case_summary` which rows have missing data

```
miss_case_summary(ufo)

# A tibble: 88,875 x 3
  case n_miss pct_miss
  <int>   <int>    <dbl>
1    877     6     54.5
2  1712     6     54.5
3  5613     6     54.5
4  7515     6     54.5
5  7625     6     54.5
6 10156     6     54.5
7 16634     6     54.5
8 18240     6     54.5
9 19813     6     54.5
10 19908     6     54.5
# ... with 88,865 more rows
```
Missing data with the `vizdat/naniar` packages

`miss_var_summary` which variables have missing data

```r
miss_var_summary(ufo)

# A tibble: 11 x 3
  variable              n_miss pct_miss
       <chr>    <int>   <dbl>
1  country       12561  14.1
2    state        7519  8.46
3    shape        3118  3.51
4 duration (hours/min) 3101  3.49
5     city        196  0.221
6  comments       126  0.142
7 duration (seconds)    5 0.00563
  8   datet ime       0  0
  9 date posted       0  0
10    latitude       0  0
11   longitude       0  0
```
After hours of cleaning...
More ways to save: write_rds

If you want to save one object, you can use `readr::write_rds` to save to a compressed rds file:

```r
write_rds(ufo, path = "ufo_dataset.rds", compress = "xz")
```
More ways to save: `read_rds`

To read this back in to R, you need to use `read_rds`, but need to assign it:

```r
ufo3 = read_rds(path = "ufo_dataset.rds")
idemical(ufo, ufo3) # test if they are the same

[1] TRUE
```
More ways to save: `save`

The `save` command can save a set of R objects into an “R data file”, with the extension `.rda` or `.RData`.

```r
x = 5
save(ufo, x, file = "ufo_data.rda")
```
More ways to save: load

The opposite of save is load. The `ls()` command lists the items in the workspace/environment and `rm` removes them.
Data Output

While it's nice to be able to read in a variety of data formats, it's equally important to be able to output data somewhere.

`write_delim()`: Write a data frame to a delimited file “This is about twice as fast as write.csv(), and never writes row names.”

```r
args(readr::write_delim)

function (x, path, delim = " ", na = "NA", append = FALSE, col_names = !append,
quote_escape = "double")
NULL
```
Data Output

\( x \): A data frame to write to disk

\texttt{path}: the file name where you want to R object written. It can be an absolute path, or a filename (which writes the file to your working directory)

\texttt{delim}: what character separates the columns?

- "," = .csv - Note there is also a \texttt{write_csv()} function
- " " = tab delimited
Data Output

For example, we can write back out the ufo dataset with the new column name:

```r
write_csv(ufo[1:100,], path = "ufo_first100.csv")
```
Lab

Link to Lab