Read and Write Using R

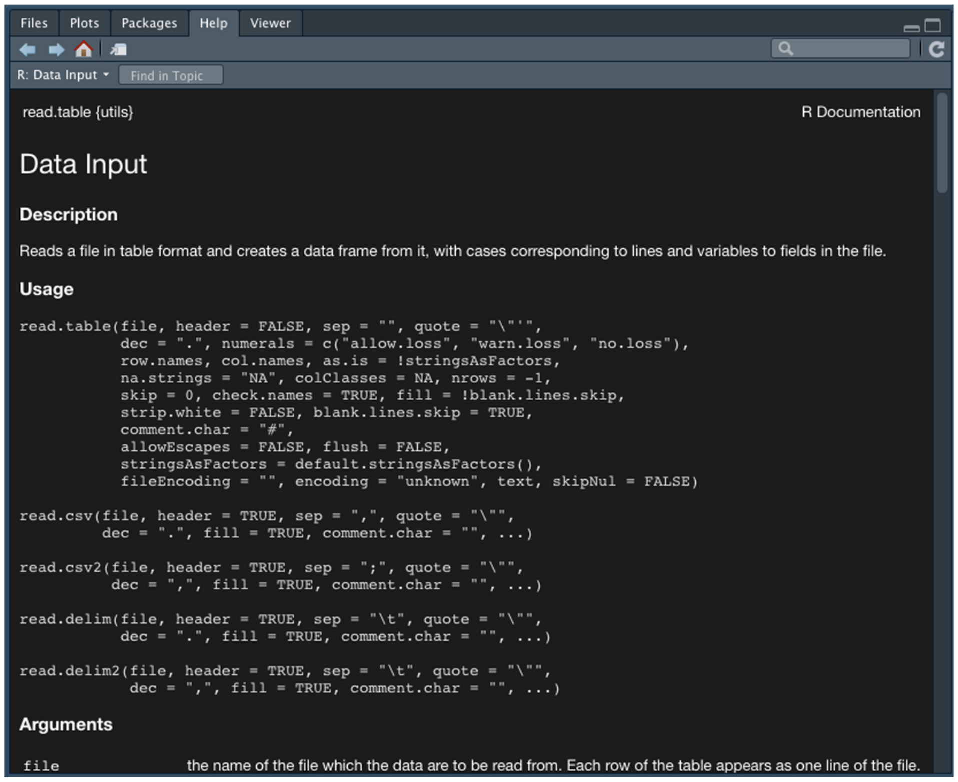
Now that Jeremy understands how to structure a function in R, he's ready to start loading up some data.

Data analysis and visualization typically begin with reading in an external data source into our programming environment.

There are built-in R functions to import the most common data formats, such as comma-separated values (CSV) and JavaScript Object Notation (JSON), as well as plenty of documentation and support online to import more advanced data structures.

To read in a CSV file, we use R's `read.csv()` function. `read.csv()` has a few required arguments to work properly. To identify the required arguments, type the following code into the R console to look at the `read.csv()` documentation in the Help pane, listed under the subhead "Usage" in the image below:

```
> ?read.csv()
```



read.table {utils} R Documentation

Data Input

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Usage

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)

read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)

read.csv2(file, header = TRUE, sep = ";", quote = "\"",
          dec = ",", fill = TRUE, comment.char = "", ...)

read.delim(file, header = TRUE, sep = "\t", quote = "\"",
           dec = ".", fill = TRUE, comment.char = "", ...)

read.delim2(file, header = TRUE, sep = "\t", quote = "\"",
            dec = ",", fill = TRUE, comment.char = "", ...)
```

Arguments

`file` the name of the file which the data are to be read from. Each row of the table appears as one line of the file.

As we can see from the documentation, `read.csv()` is one of many `read` functions that all serve the same purpose: to read in tabular, character-delimited files and create a data frame object within our R environment.

Depending on what delimiter (or value-separating character) is used, we can use `read.csv()` for comma-delimited files, `read.delim()` for tab-delimited files, or `read.table()` if we need to manually tell the function what delimiter is used.

Although optional arguments are used to parse more complicated datasets, for our purposes, we'll only concentrate on the following arguments:

- **file**
- **header**
- **sep**
- **check.names**
- **stringsAsFactors**

Referring to the R documentation for the `read.csv()` function, match the following arguments to their corresponding descriptions:

1. `header` ✓ tells the function if a header is present in the CSV file. By default, this is TRUE.
2. `stringsAsFactors` ✓ tells the function that if the column is a string data type, to cast it as a factor. We'll discuss factors later in the module, but most of the time we'll want to manually create our factors. Therefore, we need to set this flag to FALSE.
3. `check.names` ✓ will tell the function to check for spaces, punctuation, and other characters in the header and, if present, change them to periods ("."). By default, this is TRUE.
4. `file` ✓ is the local file path of the file we wish to read into our environment. Technically, it is the only required argument to run `read.csv()` .
5. `sep` ✓ tells the function what the file uses as a delimiter. By default, this is a comma (",").

Now that we understand how to use the function, let's practice reading in a demo CSV file containing a hypothetical coworker's vehicle information.

To practice reading in a CSV file, first download our sample CSV file:

Download demo.csv (https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_15/demo.csv)

After `demo.csv` has downloaded, place the data file into your active working directory. Next, we'll use `read.csv()` in our source RScript pane to read in the demo file into our R environment. Type the following code:

```
demo_table <- read.csv(file='demo.csv',check.names=F,stringsAsFactors = F)
```

NOTE

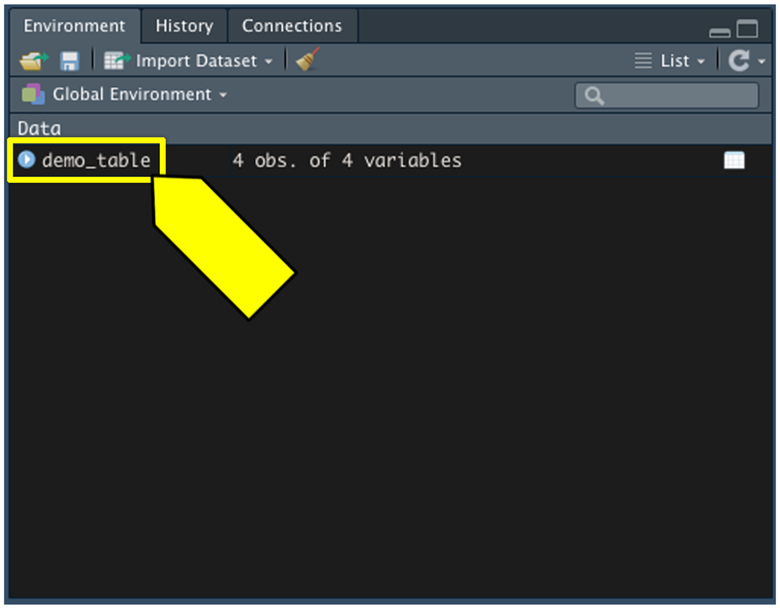
It isn't necessary to put a source file into our active working directory. If we ever want to read in a file from elsewhere on our computer, we would provide the full file path to our file argument.

By writing our read statement in our RScript, we can always quickly create and recreate the data frame by sending our assignment function in the RScript to our R console.

There are two ways to send RScript lines to our R console. We can either use the "Run" button in the top-right source pane or use the following shortcut:

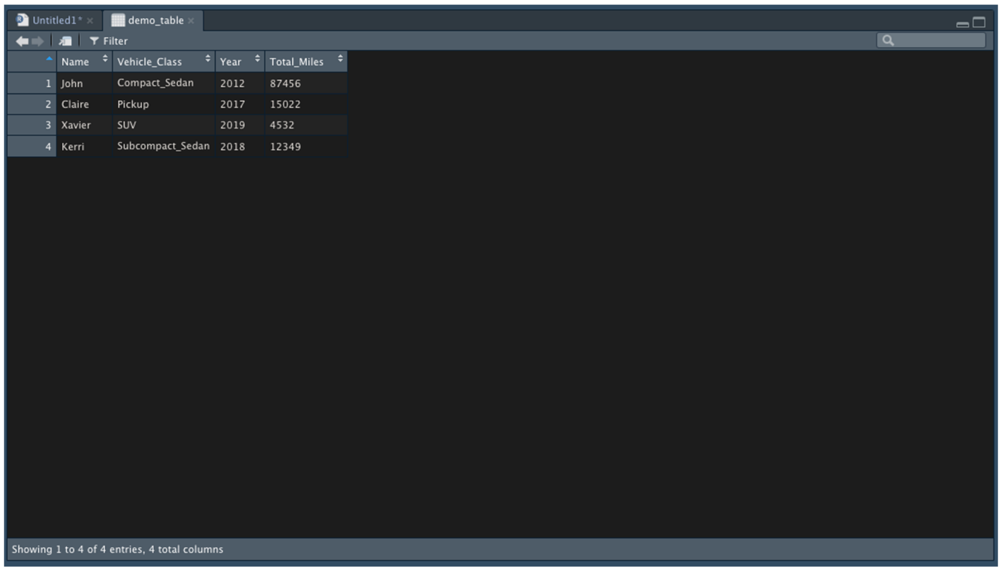
- Command + Enter (Mac)
- CTRL + Enter (Windows)

If we send our `read.csv()` function to the R console, we should see our `demo_table` created in our R environment without any errors, as shown in the following image:



demo_table 4 obs. of 4 variables

Additionally, if we click on the `demo_table` in our environment pane, it will show us our data frame in a view-only tab in the source pane. Refer to the following image:



	Name	Vehicle_Class	Year	Total_Miles
1	John	Compact_Sedan	2012	87456
2	Claire	Pickup	2017	15022
3	Xavier	SUV	2019	4532
4	Kerri	Subcompact_Sedan	2018	12349

Showing 1 to 4 of 4 entries, 4 total columns

This view-only data frame tab can be very helpful when we are trying to transform columns and rows of data in our RScripts, or when trying to select data to use in a visualization or statistical model.

But what if we want to bring in a dataset from an application programming interface (API) query?

Retake

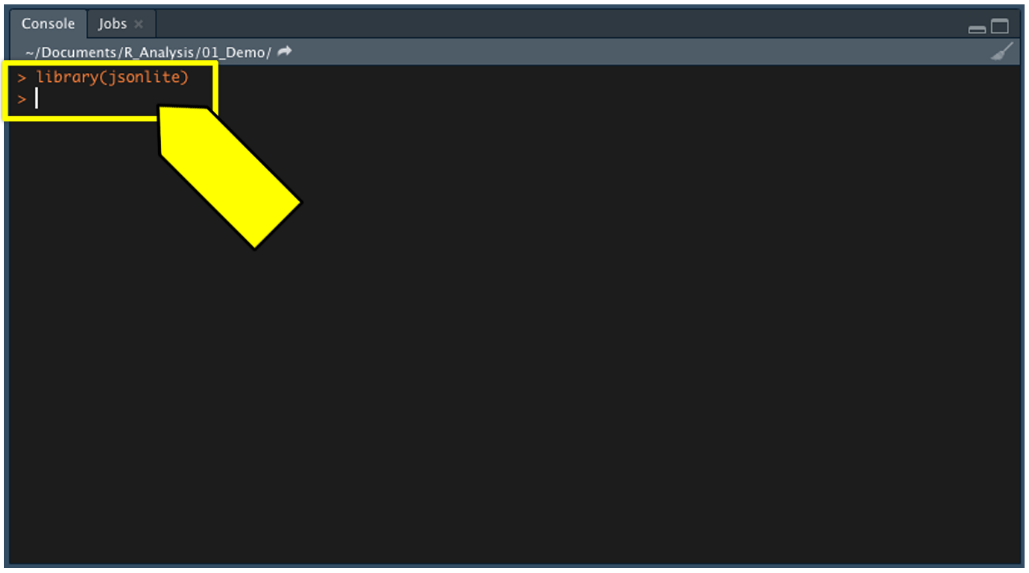
The JSON format is one of the most common data formats returned from a URL request. Although native JSON data can be easier to work with in Python, many data scientists still prefer to use R for their data analysis. To accommodate this need, R developers created the `jsonlite` library to read in JSON data structures and convert them to an R data frame. Because the `jsonlite` library was not built into R, we must import it into our R environment.

To import a library into R, we'll use the `library(package)` function. Just like in Python, it's good practice to import any required libraries at the top of our RScript.

Let's try loading in our installed `jsonlite` package using the `library(jsonlite)` function. Be sure to write the statement in your RScript and then send the statement to your R console (Command + Enter for Mac or CTRL + Enter for Windows).

SHOW PRO TIP

If we imported our library correctly, we should see the R console return characters without any errors:



```
> library(jsonlite)
> 
```

NOTE

Often you'll see conflicts or warnings when loading up packages in R, and usually these can be ignored. Only worry if your commands throw an error, which will print out in red text.

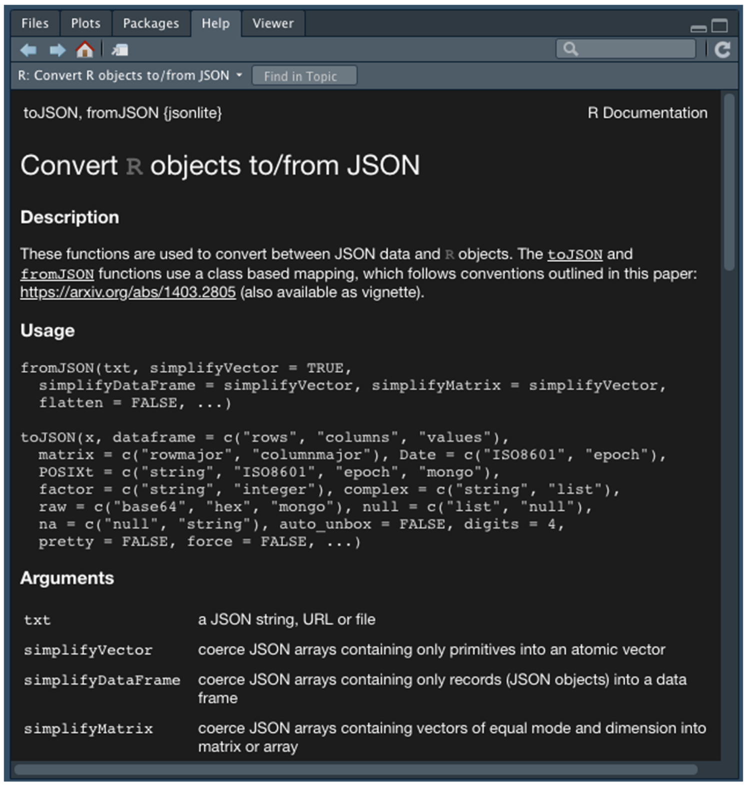
If at any time your R console throws errors when trying to import a library, you can always try to reinstall the package using the `install.packages()` function.

At this point, you may wish to save your R script by going to File, then Save. That way, you'll be able to open it later as you left it.

Now that we have successfully imported our `jsonlite` package, we can use the `fromJSON()` function to read in a JSON file into R.

First, type the following code into the R console to look at the `fromJSON()` documentation in the Help pane:

```
> ?fromJSON()
```



toJSON, fromJSON {jsonlite} R Documentation

Convert R objects to/from JSON

Description

These functions are used to convert between JSON data and R objects. The toJSON and fromJSON functions use a class based mapping, which follows conventions outlined in this paper: https://arxiv.org/abs/1403.2805 (also available as vignette).

Usage

```
fromJSON(txt, simplifyVector = TRUE,
  simplifyDataFrame = simplifyVector, simplifyMatrix = simplifyVector,
  flatten = FALSE, ...)

toJSON(x, dataframe = c("rows", "columns", "values"),
  matrix = c("rowmajor", "columnmajor"), Date = c("ISO8601", "epoch"),
  POSIXt = c("string", "ISO8601", "epoch", "mongo"),
  factor = c("string", "integer"), complex = c("string", "list"),
  raw = c("base64", "hex", "mongo"), null = c("list", "null"),
  na = c("null", "string"), auto_unbox = FALSE, digits = 4,
  pretty = FALSE, force = FALSE, ...)
```

Arguments

`txt`	a JSON string, URL or file
`simplifyVector`	coerce JSON arrays containing only primitives into an atomic vector
`simplifyDataFrame`	coerce JSON arrays containing only records (JSON objects) into a data frame
`simplifyMatrix`	coerce JSON arrays containing vectors of equal mode and dimension into matrix or array

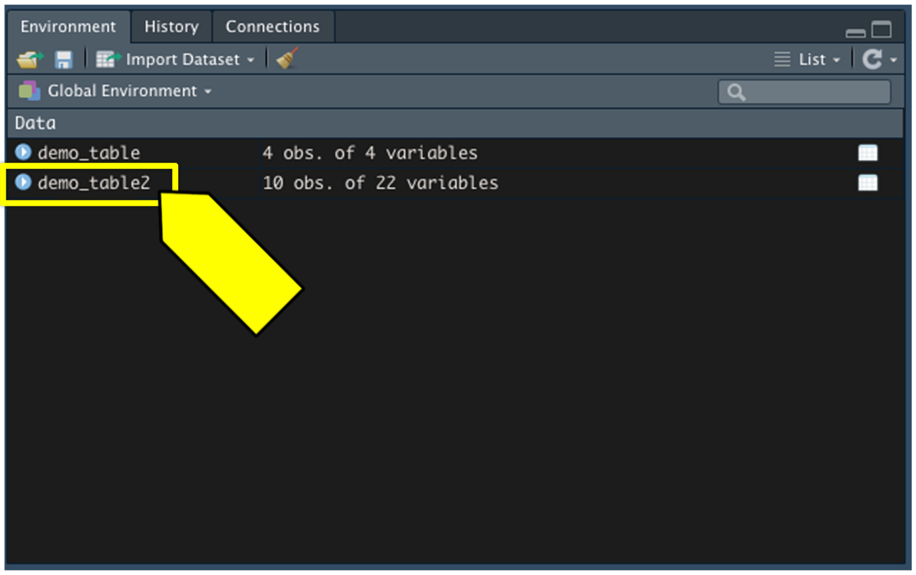
As we can see, we only need to provide the `txt` argument to properly read in a JSON file into R because the other parameters have default values indicated with equations (e.g. `simplifyVector=True`). `txt` is the file path of the JSON file on our machine. Alternatively, we can provide the `fromJSON()` function a JSON URL directly. Now let's practice reading in our first JSON file.

First, download the sample JSON file containing used car data using the link below.

Download demo.json (https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_15/demo.json)

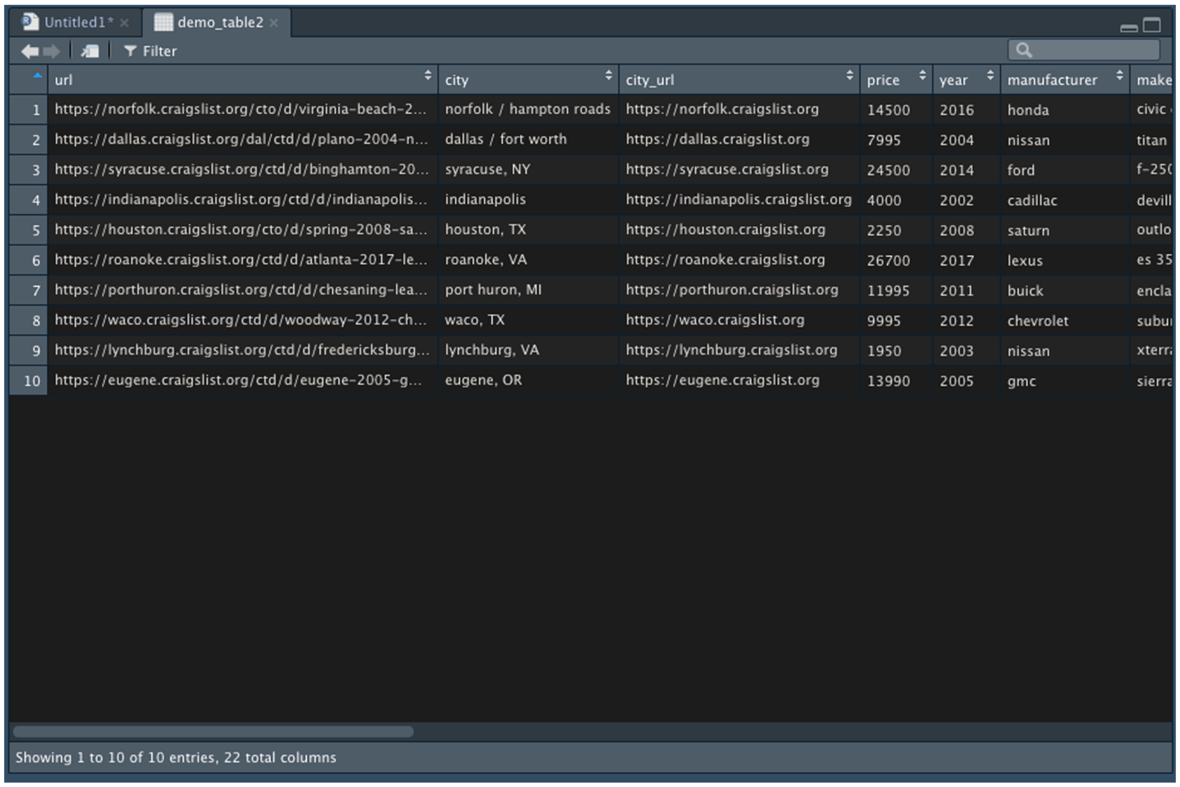
Place the downloaded data file in your active working directory. Next, use `fromJSON()` in our source RScript pane to read in the used car data into our R environment, as follows:

```
demo_table2 <- fromJSON(txt='demo.json')
```



```
demo_table     4 obs. of 4 variables
demo_table2    10 obs. of 22 variables
```

Once again, if we click the `demo_table2` in our environment pane, it will show us our data frame in a view-only tab in the source pane:



	url	city	city_url	price	year	manufacturer	make
1	https://norfolk.craigslist.org/cto/d/virginia-beach-2...	norfolk / hampton roads	https://norfolk.craigslist.org	14500	2016	honda	civic
2	https://dallas.craigslist.org/dal/ctd/d/plano-2004-n...	dallas / fort worth	https://dallas.craigslist.org	7995	2004	nissan	titan
3	https://syracuse.craigslist.org/ctd/d/binghamton-20...	syracuse, NY	https://syracuse.craigslist.org	24500	2014	ford	f-25
4	https://indianapolis.craigslist.org/ctd/d/indianapolis...	indianapolis	https://indianapolis.craigslist.org	4000	2002	cadillac	devill
5	https://houston.craigslist.org/cto/d/spring-2008-sa...	houston, TX	https://houston.craigslist.org	2250	2008	saturn	outlo
6	https://roanoke.craigslist.org/ctd/d/atlanta-2017-le...	roanoke, VA	https://roanoke.craigslist.org	26700	2017	lexus	es 35
7	https://porthuron.craigslist.org/ctd/d/chesaning-lea...	port huron, MI	https://porthuron.craigslist.org	11995	2011	buick	encla
8	https://waco.craigslist.org/ctd/d/woodway-2012-ch...	waco, TX	https://waco.craigslist.org	9995	2012	chevrolet	subu
9	https://lynchburg.craigslist.org/ctd/d/fredericksburg...	lynchburg, VA	https://lynchburg.craigslist.org	1950	2003	nissan	xterr
10	https://eugene.craigslist.org/ctd/d/eugene-2005-g...	eugene, OR	https://eugene.craigslist.org	13990	2005	gmc	sierr

Showing 1 to 10 of 10 entries, 22 total columns

Now that we know how to create data structures and data frames in R, let's learn how to slice and sample our datasets.