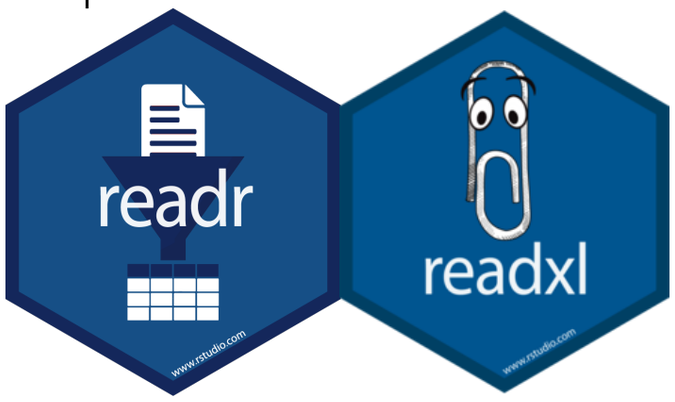
Import Data with

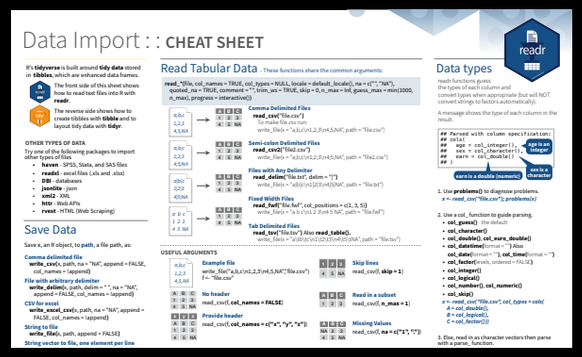


Until now we've focused on learning how to use the main packages in the tidyverse: ggplot2 and dplyr. To do this, we've used data that comes with R. But all of the data that you want to analyze will come from somewhere else. Now we're going to learn about two R packages for importing data into R: readr (for CSV files) and readxl (for Excel files).

Exercise: Importing Data

- Split into groups and ask your partner:
  - *Where is the data that your want to analyze?*
  - *How do you currently access it?*

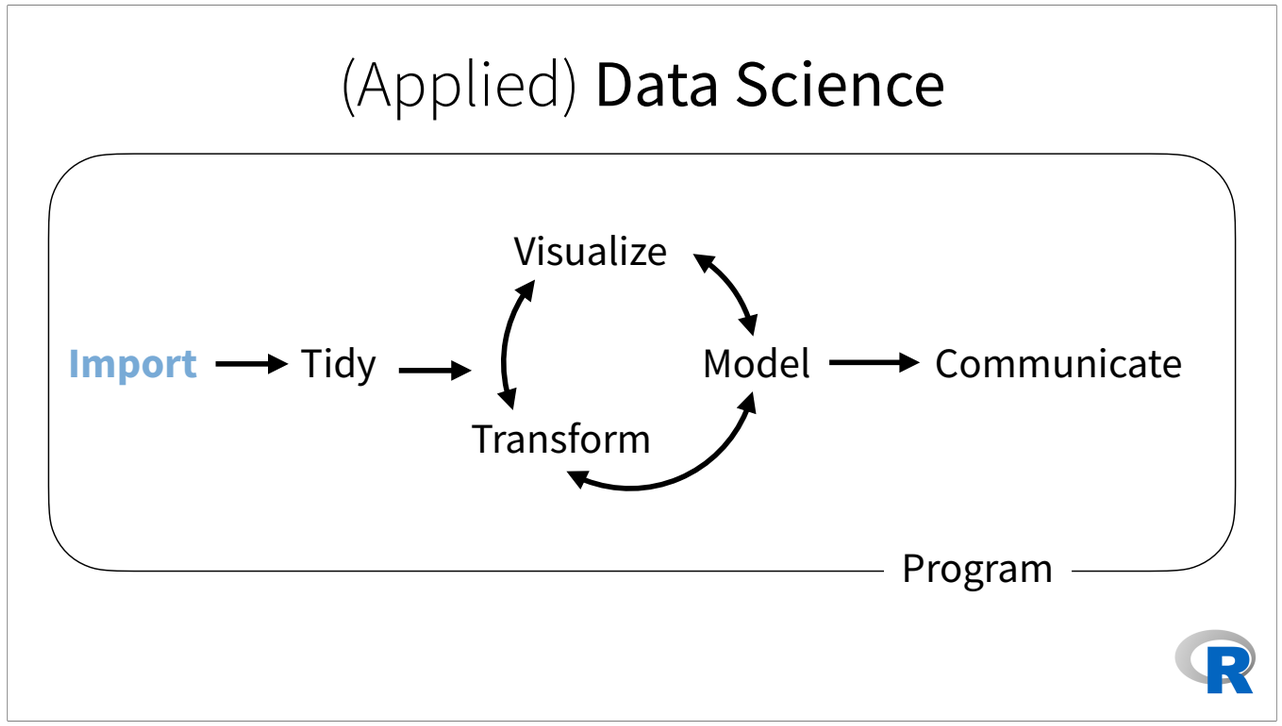
Let's start by reviewing what you already know.



Notes form & Cheat Sheet

Please take out

There are two documents that I'd like you to take out before we start. The first is the notes form I gave you. The second is a cheat sheet.



As a reminder, we started by visualizing data, which is arguably the best way to understand it. Then we stepped backwards, and learned how to take a large dataframe (babynames), and transform is to we could visualize it (a graph of your name's popularity over time). Now we're going to learn how to import data into R, which you can then transform and visualize. In some sense, we're just increasing the number of datasets you can extract value from.

What makes Importing Data special is that your data can be in multiple formats / locations. And the Tidyverse has a separate package for each data store. This is fundamentally different than how the Tidyverse handles Visualization, where there's just one package (ggplot2). And Transforming Data Frames, where there's also just one package (dplyr).

Data Import in the Tidyverse
A package for each storage type!

package	accesses
readr	csv, tsv, etc.
haven	SPSS, Stata, and SAS files
readxl	excel files (.xls, .xlsx)
jsonlite	json
xml2	xml
httr2	web API's
rvest	web pages (web scraping)
DBI	databases
sparklyr	data loaded into spark

There are a ton of places data can reside. The Tidyverse gives you consistent, high-quality functions for getting data into R, regardless of the source. I want to draw attention to 2 of those places now: CSV files and Excel files. We're going to do exercises for working with both those packages today. But I wanted you to at least be aware of the other packages and sources.

Reading in CSV Files

We're going to go through an exercise reading in CSV files. This file is specially designed to give you experience with two of the most common problems reading in data into R: dealing with NA values and dealing with data types

readr

Simple, consistent functions for working with strings / csv data.

```
# install.packages("tidyverse")
library(readr)
```

When you install the tidyverse package it automatically installs the readr package. The package is loaded every time you load the tidyverse package

Open **Import-Data-Exercises.Rmd**

readr functions

function	reads
read_csv()	**Comma separated values**
read_csv2()	Semi-colon separated values
read_delim()	General delimited files
read_fwf()	Fixed width files
read_log()	Apache log files
read_table()	Space separated
read_tsv()	Tab delimited values

The first dataset that we'll be dealing with is in a CSV file, so we'll be using the function read_csv. But you should be aware that the readr package is very robust. It has functions for dealing with all sorts of formats similar to CSV.

nimbus.csv

```
date,longitude,latitude,ozone
1985-10-01T00:00:00Z,-179.375,-87.5,.
1985-10-01T00:00:00Z,-178.125,-87.5,.
1985-10-01T00:00:00Z,-176.875,-87.5,.
1985-10-01T00:00:00Z,-175.625,-87.5,.
1985-10-01T00:00:00Z,-174.375,-87.5,.
1985-10-01T00:00:00Z,-173.125,-87.5,.
1985-10-01T00:00:00Z,-171.875,-87.5,.
1985-10-01T00:00:00Z,-170.625,-87.5,.
1985-10-01T00:00:00Z,-169.375,-87.5,.
```

The file that we'll be working with is nimbus.csv, and this is what it looks like. The first row is the name of the columns, which are date, longitude, latitude and ozone.

nimbus.csv

```
date,longitude,latitude,ozone
1985-10-01T00:00:00Z,-179.375,-87.5,.
1985-10-01T00:00:00Z,-178.125,-87.5,.
1985-10-01T00:00:00Z,-176.875,-87.5,.
1985-10-01T00:00:00Z,-175.625,-87.5,.
1985-10-01T00:00:00Z,-174.375,-87.5,.
1985-10-01T00:00:00Z,-173.125,-87.5,.
1985-10-01T00:00:00Z,-171.875,-87.5,.
1985-10-01T00:00:00Z,-170.625,-87.5,.
1985-10-01T00:00:00Z,-169.375,-87.5,.
```

As you can see, each of the values are separated with commas.

Nimbus is the name of a series of satellites that made measurements of the atmosphere. Specifically, in the mid-80s they demonstrated that a hole was developing in the Ozone layer over Antarctica.

read_csv()

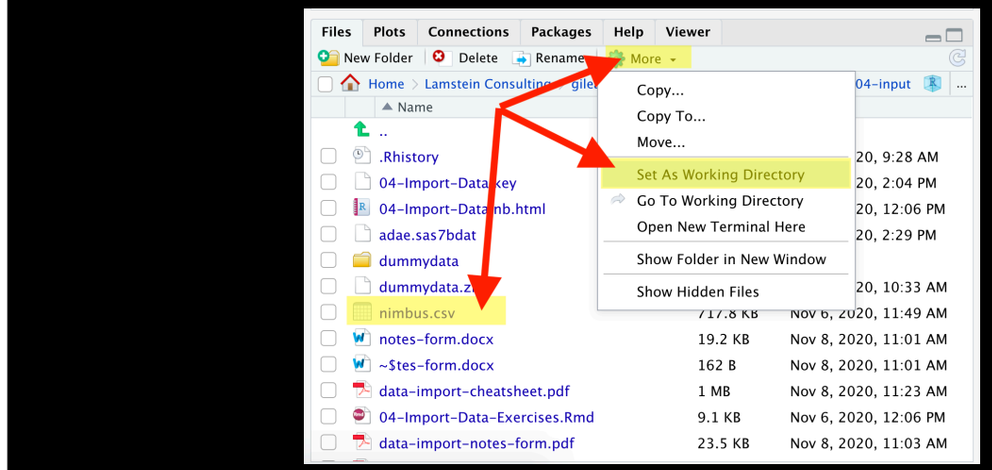
readr functions share a common syntax

```
df <- read_csv("path/to/file.csv", …)
```

object to save output into

path from working directory to file

All readr functions share a common syntax. The first argument is simply the path of the file.



Set Your Working Directory

All of the exercises we do today will require your working directory be set to the directory that contains the file nimbus.csv.

To do this, please use the file pane in RStudio to find this directory. Then click the "more" (gear) button. Then click the button that says "Set as Working Directory".

Your Turn 1

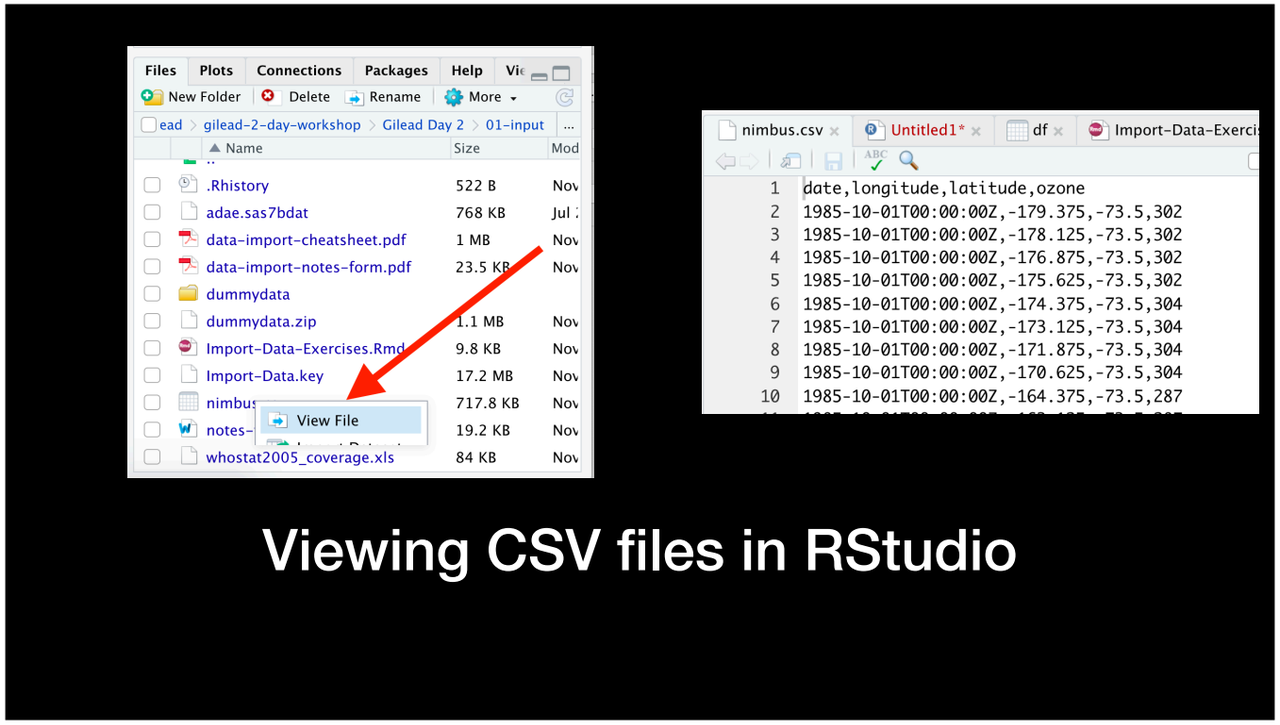
Find **nimbus.csv** (in your working directory). Then read it into an object. Then view the results.

Your Turn 1

Find **nimbus.csv** (in your working directory). Then read it into an object. Then view the results.

```
nimbus <- read_csv("nimbus.csv")
```

```
nimbus
```



Viewing CSV files in RStudio

Some of you might not know this. But if you click on the file nimbus.csv in the File pane, then a button opens up that lets you click "View File". If you do that, then you can load the entire CSV file in the text viewer.

Parsing

Quiz

What class is ozone?

```
nimbus %>% pluck("ozone") %>% class()
```

Just take a guess: what class is the column "ozone"?
"pluck" just lets you get the column as a vector.

```
nimbus %>% pluck("ozone") %>% class()
```

```
[1] "character"
```

This answer might surprise you. After all, the column appears to completely be numeric. Let's take a look at what happened.

```
nimbus %>% pluck("ozone") %>% unique()
```

```
 [1] "302" "304" "287" "274" "264" "242" "211" "195" "197" "196" "198" "193" "187"
[14] "190" "199" "194" "213" "218" "221" "229" "209" "186" "188" "191" "189" "184"
[27] "181" "."   "215" "312" "319" "320" "311" "300" "290" "267" "226" "210" "200"
[40] "203" "201" "192" "204" "206" "208" "205" "223" "232" "238" "243" "220" "202"
[53] "185" "219" "222" "216" "324" "336" "333" "323" "308" "295" "244" "212" "237"
[66] "248" "239" "241" "250" "249" "252" "234" "318" "313" "326" "335" "337" "316"
[79] "266" "207" "227" "251" "253" "257" "261" "214" "228" "273" "285" "288" "291"
[92] "270" "254" "317" "325" "332" "340" "344" "338" "297" "247" "217" "225" "231"
[105] "235" "236" "262" "260" "265" "272" "278" "280" "279" "255" "245" "224" "181"
[118] "240" "269" "296" "307" "315" "321" "306" "299" "298" "283" "327" "322" "328"
[131] "331" "310" "275" "233" "258" "276" "281" "289" "330" "346" "305" "334" "359"
[144] "347" "314" "301" "256" "263" "277" "284" "282" "271" "246" "183" "182" "230"
[157] "349" "351" "350" "342" "329" "355" "371" "309" "303" "292" "259" "268" "341"
[170] "343" "348" "345" "354" "361" "372" "382" "376" "356" "293" "286" "353" "355"
[183] "358" "360" "363" "370" "384" "380" "294" "339" "362" "352" "368" "373" "377"
```

If we look at the unique values in the vector, we can see that there's a . in there. And since all elements of a vector need to be one type, and the period can't be converted to a number, the numbers all became character vectors.

. = NA

```
nimbus
```

date <S3: POSIXct>	longitude <dbl>	latitude <dbl>	ozone <chr>
1985-10-01	-179.375	-87.5	.
1985-10-01	-178.125	-87.5	.
1985-10-01	-176.875	-87.5	.
1985-10-01	-175.625	-87.5	.
1985-10-01	-174.375	-87.5	.
1985-10-01	-173.125	-87.5	.
1985-10-01	-171.875	-87.5	.
1985-10-01	-170.625	-87.5	.
1985-10-01	-169.375	-87.5	

The reason why the period is present at all is because the . is used to represent NA values in this dataset. We need a method to convert these values to NAs in R.

read_csv()

readr functions share a common syntax

```
nimbus <- read_csv("nimbus.csv", na = ".")
```

object to save output into

path from working directory to file

Value(s) to convert to NA

All readr functions have a parameter called "na" that lets you specify values you want the function to convert to NA.

Your Turn

Reread in **nimbus.csv**. But this time convert the "."'s to NA's. How many NA's are in the ozone column?

Now re-read in Nimbus. But make sure that the periods get converted to NAs. Use "pluck" and "class" to verify that the class of the ozone column is numeric. Then count the number of NAs.

Your Turn

Reread in **nimbus.csv**. But this time convert the "."'s to NA's. How many NA's are in the ozone column?

```
nimbus <- read_csv("nimbus.csv", na = ".")
nimbus %>%
  filter(is.na(ozone)) %>%
  summarize(n = n())
##     n
## 1  155
```

Quiz

What "type" of column is ozone?

As a programming language, R is a bit unusual in that it has both a notion of a class and a type. Just tell your partner what you think the "type" of the ozone column is.

```
nimbus <- read_csv("nimbus.csv", na = ".")
```

```
date                longitude latitude ozone
<dttm>                  <dbl>    <dbl> <dbl>
1985-10-01 00:00:00     -179.    -73.5   302
1985-10-01 00:00:00     -178.    -73.5   302
1985-10-01 00:00:00     -177.    -73.5   3
1985-10-01 00:00:00     -176.    -73.5   3
1985-10-01 00:00:00     -174.    -73.5   3
1985-10-01 00:00:00     -173.    -73.5   3
1985-10-01 00:00:00     -172.    -73.5   304
1985-10-01 00:00:00     -171.    -73.5   304
1985-10-01 00:00:00     -164.    -73.5   287
1985-10-01 00:00:00     -163.    -73.5   287
… with 18,953 more rows
```

<dbl> stands for "double"

There are two main types of numeric data in R: integers and doubles. Doubles are the type that has decimal points.

suppose

```
nimbus <- read_csv("nimbus.csv", na = ".")
```

date <S3: POSIXct>	longitude <dbl>	latitude <dbl>	ozone <chr>
1985-10-01	-179.375	-87.5	*NA*
1985-10-01	-178.125	-87.5	*NA*
1985-10-01	-176.875	-87.5	*NA*
1985-10-01	-175.625	-87.5	*NA*
1985-10-01	-174.375	-87.5	*NA*
1985-10-01	-173.125	-87.5	*NA*
1985-10-01	-171.875	-87.5	*NA*
1985-10-01	-170.625	-87.5	*NA*
1985-10-01	-169.375	-87.5	*NA*
1985-10-01	-168.125	-87.5	*NA*

<chr> stands for character string (not a number)

read_csv makes a guess about the type of your data. But sometimes it guesses wrong. Here's an example of it guessing wrong - assuming that ozone (which is numeric) is actually a character. We haven't spoken about data types a lot in this course, but I think we can all imagine something like this happening when we import data. A more common example might be R assuming that the date column is actually character.

read_csv()

readr functions share a common syntax

```
nimbus <- read_csv("nimbus.csv", na = ".",
  col_types = list(ozone = col_double()))
```

Manually specify column types.

list

column name

Column type function

If this happens, you want to use the col_types parameter. This allows you to force R to treat a column as if its a certain type.

type function	data type
col_character()	character
col_date()	Date
col_datetime()	POSIXct (date-time)
col_double()	double (numeric)
col_factor()	factor
col_guess()	let readr guess (default)
col_integer()	integer
col_logical()	logical
col_number()	numbers mixed with non-number characters
col_numeric()	double or integer
col_skip()	do not read
col_time()	time

Here is the list of all the functions that cover all the types that read_csv can read. Again, you only need to use these functions when readr is incorrectly guessing the column types.

type function	data type
col_character()	character
col_date()	Date
col_datetime()	POSIXct (date-time)
col_double()	**double (numeric)**
col_factor()	factor
col_guess()	let readr guess (default)
col_integer()	integer
col_logical()	logical
col_number()	numbers mixed with non-number characters
col_numeric()	double or integer
col_skip()	do not read
col_time()	time

Your Turn

Read in **nimbus.csv**. accounting for NA's and setting the col_type of ozone to a double. Then make this plot. What do you see?

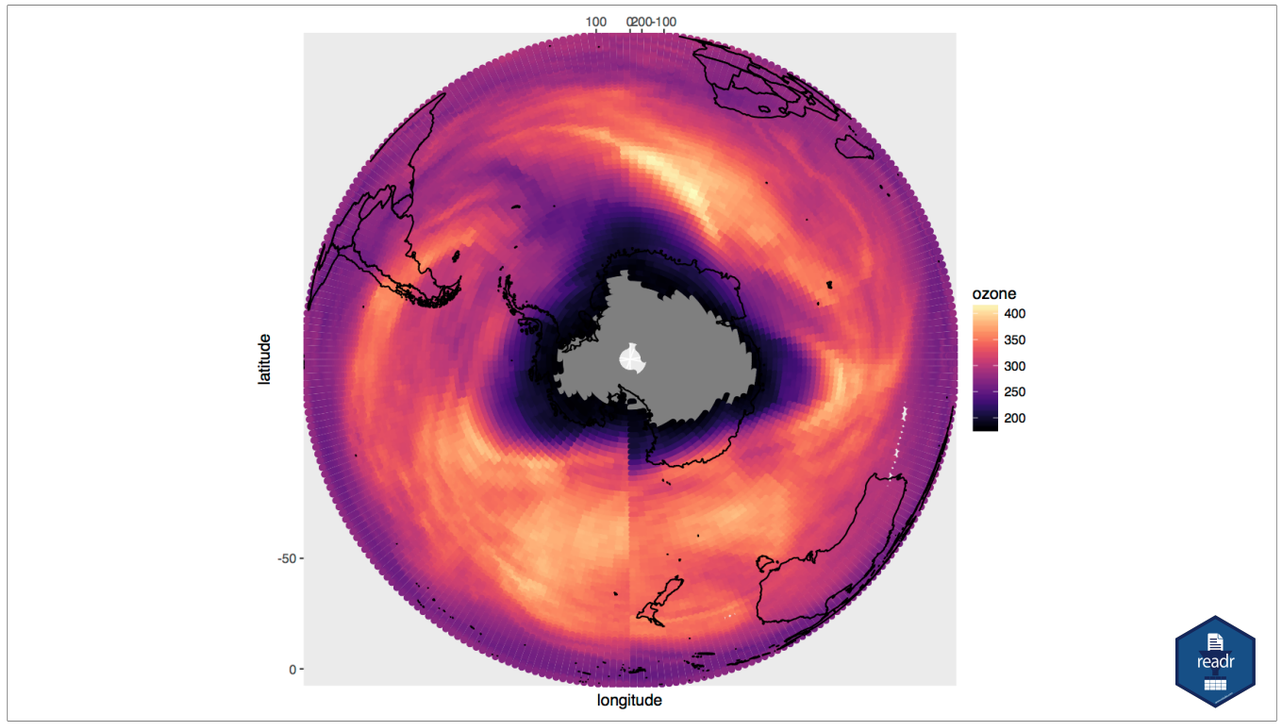
```
library(viridis)
world <- map_data(map = "world")
nimbus %>%
  ggplot() +
    geom_point(aes(longitude, latitude, color = ozone)) +
    geom_path(aes(long, lat, group = group), data = world) +
    coord_map("ortho", orientation=c(-90, 0, 0)) +
    scale_color_viridis(option = "A")
```

ggplot2 can do a lot of things that we didn't cover earlier. Among them is creating maps. The code here has really already been written. But try to spend a few minutes looking at both the code and result, and interpreting them both. Experiment by removing various layers and recreating the plot

```
nimbus <- read_csv("nimbus.csv", na = ".",
  col_types = list(ozone = col_double()))

library(viridis)
world <- map_data(map = "world")
nimbus %>%
  ggplot() +
    geom_point(aes(longitude, latitude, color = ozone)) +
    geom_path(aes(long, lat, group = group), data = world) +
    coord_map("ortho", orientation=c(-90, 0, 0)) +
    scale_color_viridis(option = "A")
```

Walk through the code. Specifically each layer of the ggplot2.


100
0
200
-100
latitude
-50
0
longitude
ozone
400
350
300
250
200
readr

Writing

readr functions

function	writes
write_csv()	Comma separated values
write_excel_csv()	CSV intended for opening in Excel
write_delim()	General delimited files
write_file()	Single string, written as is
write_lines()	Vector of strings, one element per line
write_tsv()	Tab delimited values

readr also has functions for writing csv and other similar files.

write_csv()

Saves data set as a csv on your computer.

```
write_csv(nimbus, file = "nimbus2.csv")
```

Table to save

file path to save at

write_csv requires 2 arguments: the first is the data frame. The second is the file path.

Reading in Excel Files

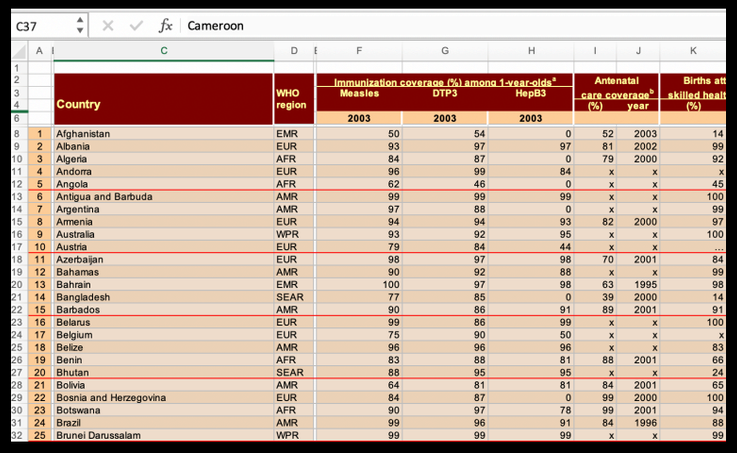
Now let's turn our attention to reading in Excel files, which present their own special problems

readxl

Reading in Excel Files

```
# install.packages("tidyverse")
library(readxl)
```

When you install the tidyverse package it automatically installs the readxl package. However, you still need to load it with the library command.



C37 | Cameroon

	A	C	D	F	G	H	I	J	K
1									
2				Immunization coverage (%) among 1-year-olds[a]			Antenatal		Births att
3		Country	WHO region	Measles	DTP3	HepB3	care coverage[b]		skilled healt
4							(%)	year	(%)
6				2003	2003	2003			
8	1	Afghanistan	EMR	50	54	0	52	2003	14
9	2	Albania	EUR	93	97	97	81	2002	99
10	3	Algeria	AFR	84	87	0	79	2000	92
11	4	Andorra	EUR	96	99	84	x	x	x
12	5	Angola	AFR	62	46	0	x	x	45
13	6	Antigua and Barbuda	AMR	99	99	99	x	x	100
14	7	Argentina	AMR	97	88	0	x	x	99
15	8	Armenia	EUR	94	94	93	82	2000	97
16	9	Australia	WPR	93	92	95	x	x	100
17	10	Austria	EUR	79	84	44	x	x	...
18	11	Azerbaijan	EUR	98	97	98	70	2001	84
19	12	Bahamas	AMR	90	92	88	x	x	99
20	13	Bahrain	EMR	100	97	98	63	1995	98
21	14	Bangladesh	SEAR	77	85	0	39	2000	14
22	15	Barbados	AMR	90	86	91	89	2001	91
23	16	Belarus	EUR	99	86	99	x	x	100
24	17	Belgium	EUR	75	90	50	x	x	x
25	18	Belize	AMR	96	96	96	x	x	83
26	19	Benin	AFR	83	88	81	88	2001	66
27	20	Bhutan	SEAR	88	95	95	x	x	24
28	21	Bolivia	AMR	64	81	81	84	2001	65
29	22	Bosnia and Herzegovina	EUR	84	87	0	99	2000	100
30	23	Botswana	AFR	90	97	78	99	2001	94
31	24	Brazil	AMR	99	96	91	84	1996	88
32	25	Brunei Darussalam	WPR	99	99	99	x	x	99

who.xls

Country-Level Immunization Stats from the World Health Organization

The last company I taught this to worked in Pharma, so I used immunization data for them. This workbook comes from the WHO, contains immunization data on each country and was published in 2005.

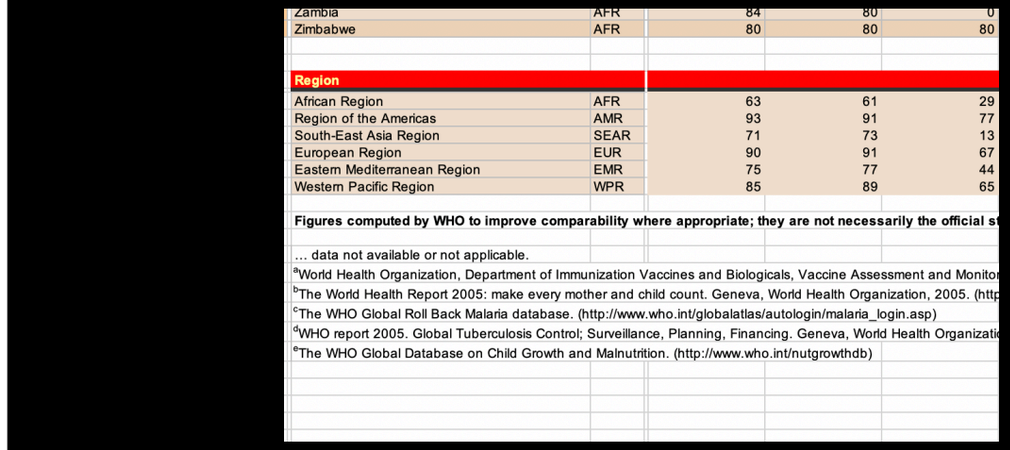
Quiz

Open up **who.xls** in Excel.

What problems might you encounter reading this data into R?

(Hint: Is this data "tidy"?)

3 Minute Exercise.
Open up this file in Excel. What problems might you encounter reading this data into R? This exercise is important, because it reinforces how real-world data needs to be modified before it can be used by the Tidyverse.



Zambia	AFR	84	80	0
Zimbabwe	AFR	80	80	80
Region				
African Region	AFR	63	61	29
Region of the Americas	AMR	93	91	77
South-East Asia Region	SEAR	71	73	13
European Region	EUR	90	91	67
Eastern Mediterranean Region	EMR	75	77	44
Western Pacific Region	WPR	85	89	65
Figures computed by WHO to improve comparability where appropriate; they are not necessarily the official st				
... data not available or not applicable.				
[a]World Health Organization, Department of Immunization Vaccines and Biologicals, Vaccine Assessment and Monitor				
[b]The World Health Report 2005: make every mother and child count. Geneva, World Health Organization, 2005. (http				
[c]The WHO Global Roll Back Malaria database. (http://www.who.int/globalatlas/autologin/malaria_login.asp)				
[d]WHO report 2005. Global Tuberculosis Control; Surveillance, Planning, Financing. Geneva, World Health Organizatio				
[e]The WHO Global Database on Child Growth and Malnutrition. (http://www.who.int/nutgrowthdb)				

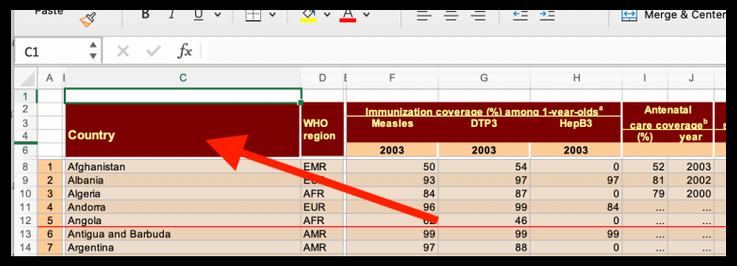
Sheet != Table

Perhaps most seriously, the sheet itself contains multiple tables and footnotes. By definition, Excel sheets do not have to be "tidy". (This does not make them bad - footnotes are useful!)

2.7	50
x	50
x	45
x	23
x	28
x	50

2 NA values

The bottom table actually has two values for NA: x and ""



Multi-row columns

It also breaks the concept of "tidy data" by having column names that span multiple rows. There's also a blank row at the top.

read_excel()

Read in an Excel File

```
df <- read_excel("who.xls", na = c("x", ""))
```

object to save output into

path from working directory to file

Value(s) to convert to NA

Luckily the function read_excel can handle all of this for us. What I want you to be aware of right now is that, just like the readr package's read_csv function, read_excel has an "na" parameter that lets you specify the NA values you want. We'll definitely need that. It also lets you specify a vector of values, like you see here. Although "" is always considered NA, so this is not strictly necessary here.

read_excel()

Read in an Excel File

```
df <- read_excel("who.xls", na = c("x", ""), range="A1:B2")
```

Range of Cells to read

read_excel also has a "range" parameter that lets you read in just a range of cells. The cells need to be separated by a :

Exercise: Reading in the Small Table

Read in the small table in **who.xls.** Make it look like the image on the right

1. Set the **range** parameter set to read in just the second, smaller table.
2. Set the **NA** parameter appropriately.
3. Set **another option** ... so that the first row is not treated as a column name (see ?read_excel)

	...1	...2	...3	...4
1	African Region	AFR	*NA*	
2	Region of the Americas	AMR	*NA*	
3	South-East Asia Region	SEAR	*NA*	
4	European Region	EUR	*NA*	
5	Eastern Mediterranean Region	EMR	*NA*	
6	Western Pacific Region	WPR	*NA*	

Note that the column names are auto-generated - the key thing is that the first row of the table is not treated as row names

Exercise: Reading in the Small Table

```
read_excel("who.xls",
           na        = c("x", ""),
           range     = "C204:U209",
           col_names = FALSE)
```

	...1	...2	...3	...4
1	African Region	AFR	*NA*	
2	Region of the Americas	AMR	*NA*	
3	South-East Asia Region	SEAR	*NA*	
4	European Region	EUR	*NA*	
5	Eastern Mediterranean Region	EMR	*NA*	
6	Western Pacific Region	WPR	*NA*	

excel_sheets()

List sheets in an Excel file

```
excel_sheets("who.xls")
[1] "2.Health service coverage"
```

Excel files often contain multiple sheets. The file **excel_sheets** can list how those sheets need to be referenced in code. And you can then provide that to the read_excel file

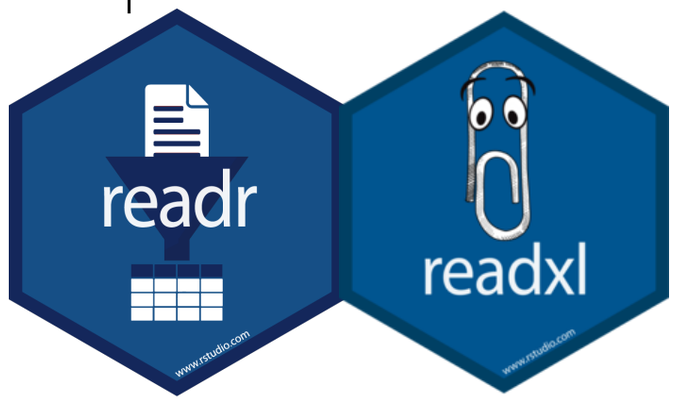
Data Import in the Tidyverse
A package for each storage type!

package	accesses
readr	csv, tsv, etc.
haven	SPSS, Stata, and SAS files
readxl	excel files (.xls, .xlsx)
jsonlite	json
xml2	xml
httr	web API's
rvest	web pages (web scraping)
DBI	databases
sparklyr	data loaded into spark

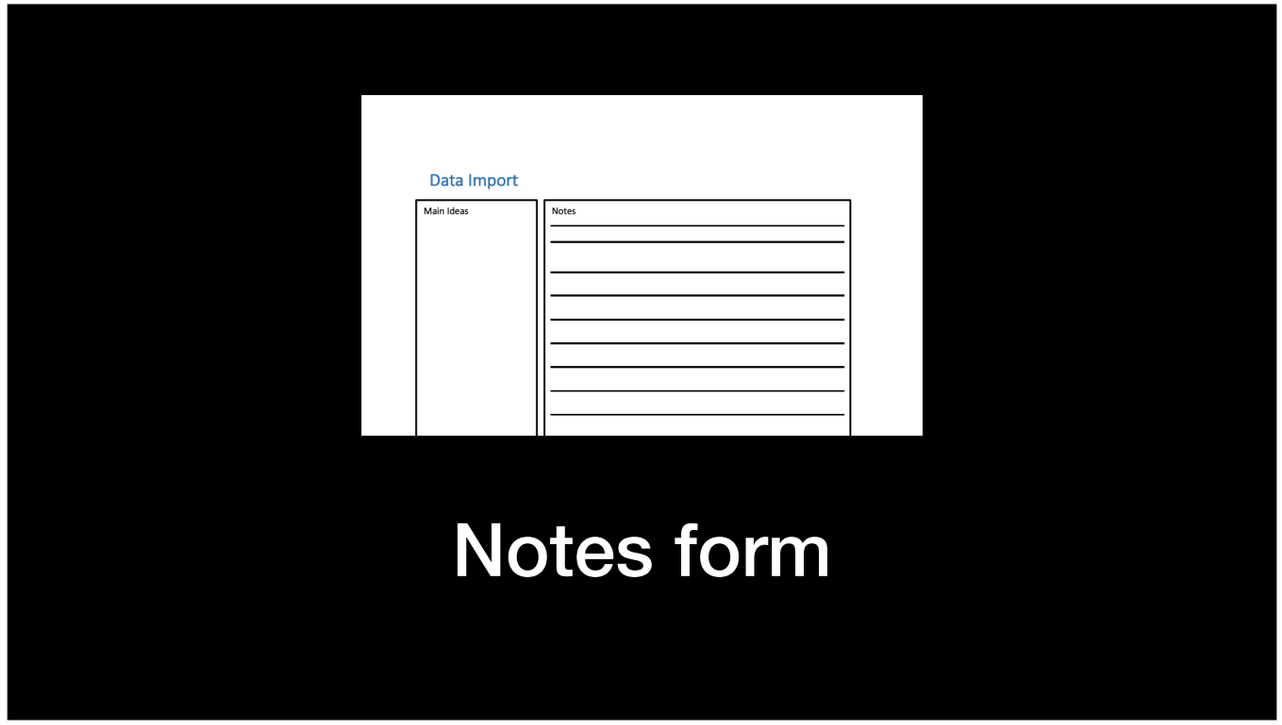
Again, the Tidyverse provides a suite of packages for reading in data. One package for each data source. If you ever need to get data from somewhere, this is a good place to look.

The key issues when dealing with reading in data are NA values and parsing data types. Certain speciality data formats, like Excel, will have their own special issues (such as reading in a subset of cells).

Import Data with



Any questions?



Notes form

I'd now like to give you a minute to write down any thoughts or questions you have about what you just learned.