R Language Basics and Data Importing

PSYC 259: Principles of Data Science

John Franchak

R language basics

- Calculation and assignment
- Calling functions
- Importing data in tibbles
- Extending the R language with packages

Follow along from the Github repo

Last updated: 2022-01-11

```r
# R can run simple calculations
1 + 1
```

```
[1] 2
```

```r
# R can run simple calculations
1 + 1
4^2
```

```
[1] 2

[1] 16
```

```r
# R can run simple calculations
1 + 1
4^2

# Calculations aren't useful unless we put the results somewhere
# The assignment operator stores the result into a variable

two <- 1 + 1
```

```
[1] 2

[1] 16
```

```r
# R can run simple calculations
1 + 1
4^2

# Calculations aren't useful unless we put the results somewhere
# The assignment operator stores the result into a variable

two <- 1 + 1
two
```

```
[1] 2

[1] 16

[1] 2
```

```r
# R can run simple calculations
1 + 1
4^2

# Calculations aren't useful unless we put the results somewhere
# The assignment operator stores the result into a variable

two <- 1 + 1
two

# Once assigned, variables can be modified by re-assigning a new value to them

two <- 1 + 3
```

```
[1] 2

[1] 16

[1] 2
```

```r
# R can run simple calculations
1 + 1
4^2

# Calculations aren't useful unless we put the results somewhere
# The assignment operator stores the result into a variable

two <- 1 + 1
two

# Once assigned, variables can be modified by re-assigning a new value to them

two <- 1 + 3
two
```

```
[1] 2

[1] 16

[1] 2

[1] 4
```

```r
# R can run simple calculations
1 + 1
4^2

# Calculations aren't useful unless we put the results somewhere
# The assignment operator stores the result into a variable

two <- 1 + 1
two

# Once assigned, variables can be modified by re-assigning a new value to them

two <- 1 + 3
two

# Variables can be removed from the workspace with rm
# After removing two, calling it again would lead to an error
rm(two)
```

```
[1] 2

[1] 16

[1] 2

[1] 4
```

```r
# R can run simple calculations
1 + 1
4^2

# Calculations aren't useful unless we put the results somewhere
# The assignment operator stores the result into a variable

two <- 1 + 1
two

# Once assigned, variables can be modified by re-assigning a new value to them

two <- 1 + 3
two

# Variables can be removed from the workspace with rm
# After removing two, calling it again would lead to an error
rm(two)

# Variables can be reused in expressions to calculate new variables/outputs
var1 <- 5
```

```
[1] 2

[1] 16

[1] 2

[1] 4
```

```r
# R can run simple calculations
1 + 1
4^2

# Calculations aren't useful unless we put the results somewhere
# The assignment operator stores the result into a variable

two <- 1 + 1
two

# Once assigned, variables can be modified by re-assigning a new value to them

two <- 1 + 3
two

# Variables can be removed from the workspace with rm
# After removing two, calling it again would lead to an error
rm(two)

# Variables can be reused in expressions to calculate new variables/outputs
var1 <- 5
var2 <- 10
```

```
[1] 2

[1] 16

[1] 2

[1] 4
```

```r
# R can run simple calculations
1 + 1
4^2

# Calculations aren't useful unless we put the results somewhere
# The assignment operator stores the result into a variable

two <- 1 + 1
two

# Once assigned, variables can be modified by re-assigning a new value to them

two <- 1 + 3
two

# Variables can be removed from the workspace with rm
# After removing two, calling it again would lead to an error
rm(two)

# Variables can be reused in expressions to calculate new variables/outputs
var1 <- 5
var2 <- 10
var3 <- var1 + var2
```

```
[1] 2

[1] 16

[1] 2

[1] 4
```

```r
# R can run simple calculations
1 + 1
4^2

# Calculations aren't useful unless we put the results somewhere
# The assignment operator stores the result into a variable

two <- 1 + 1
two

# Once assigned, variables can be modified by re-assigning a new value to them

two <- 1 + 3
two

# Variables can be removed from the workspace with rm
# After removing two, calling it again would lead to an error
rm(two)

# Variables can be reused in expressions to calculate new variables/outputs
var1 <- 5
var2 <- 10
var3 <- var1 + var2

# When assigning variables, nothing prints to the console
# Let's use the function print
print(var3)
```

```
[1] 2

[1] 16

[1] 2

[1] 4

[1] 15
```

```r
# Print and rm are functions that we use in R
# We call functions by writing their name followed
# by a list of arguments in parentheses

abs(-1)
```

```
[1] 1
```

```r
# Print and rm are functions that we use in R
# We call functions by writing their name followed
# by a list of arguments in parentheses

abs(-1)
min(var1, var2)
```

```
[1] 1

[1] 5
```

```r
# Print and rm are functions that we use in R
# We call functions by writing their name followed
# by a list of arguments in parentheses

abs(-1)
min(var1, var2)
sum(var1, var2, var3)
```

```
[1] 1

[1] 5

[1] 30
```

```r
# Print and rm are functions that we use in R
# We call functions by writing their name followed
# by a list of arguments in parentheses

abs(-1)
min(var1, var2)
sum(var1, var2, var3)

# c is a function that combines values together
my_vars <- c(var1, var2, var3)
```

```
[1] 1

[1] 5

[1] 30
```

```r
# Print and rm are functions that we use in R
# We call functions by writing their name followed
# by a list of arguments in parentheses

abs(-1)
min(var1, var2)
sum(var1, var2, var3)

# c is a function that combines values together
my_vars <- c(var1, var2, var3)
my_vars
```

```
[1] 1

[1] 5

[1] 30

[1]  5 10 15
```

```r
# Print and rm are functions that we use in R
# We call functions by writing their name followed
# by a list of arguments in parentheses

abs(-1)
min(var1, var2)
sum(var1, var2, var3)

# c is a function that combines values together
my_vars <- c(var1, var2, var3)
my_vars

# RStudio has built-in help for every function
?c
```

```
[1] 1

[1] 5

[1] 30

[1]  5 10 15
```

```r
# Print and rm are functions that we use in R
# We call functions by writing their name followed
# by a list of arguments in parentheses

abs(-1)
min(var1, var2)
sum(var1, var2, var3)

# c is a function that combines values together
my_vars <- c(var1, var2, var3)
my_vars

# RStudio has built-in help for every function
?c

# Functions can also be used to import data
ds <- read.csv('data_raw/vocab16.csv')
```

```
[1] 1

[1] 5

[1] 30

[1]  5 10 15
```

```r
# Print and rm are functions that we use in R
# We call functions by writing their name followed
# by a list of arguments in parentheses

abs(-1)
min(var1, var2)
sum(var1, var2, var3)

# c is a function that combines values together
my_vars <- c(var1, var2, var3)
my_vars

# RStudio has built-in help for every function
?c

# Functions can also be used to import data
ds <- read.csv('data_raw/vocab16.csv')
print(ds)
```

```
[1] 1

[1] 5

[1] 30

[1]  5 10 15

  age   word
1  16  shoes
2  16  berry
3  16     hi
4  16 diaper
5  16  teeth
6  16   uhoh
```

```r
# Print and rm are functions that we use in R
# We call functions by writing their name followed
# by a list of arguments in parentheses

abs(-1)
min(var1, var2)
sum(var1, var2, var3)

# c is a function that combines values together
my_vars <- c(var1, var2, var3)
my_vars

# RStudio has built-in help for every function
?c

# Functions can also be used to import data
ds <- read.csv('data_raw/vocab16.csv')
print(ds)

# read.csv is part of base R, the default fx set
# When we want to use functions to expand R, we
# need to use library fx to load packages

library(readr)  #for read_csv
```

```
[1] 1

[1] 5

[1] 30

[1]  5 10 15

  age   word
1  16  shoes
2  16  berry
3  16     hi
4  16 diaper
5  16  teeth
6  16   uhoh
```

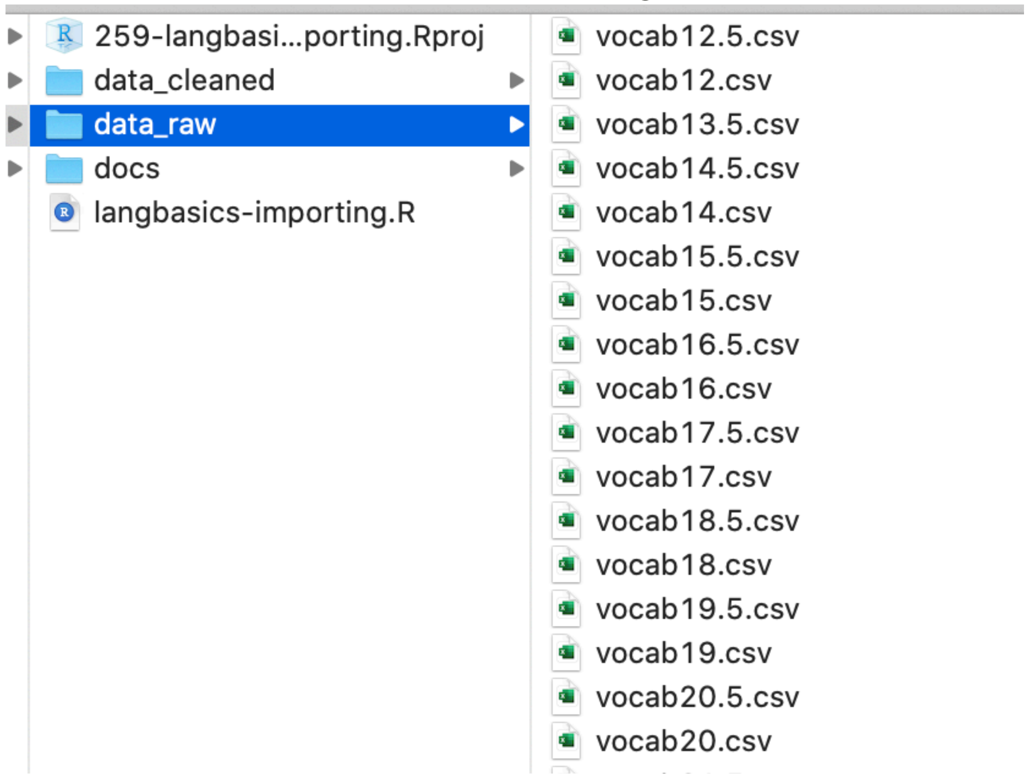
```r
# Print and rm are functions that we use in R
# We call functions by writing their name followed
# by a list of arguments in parentheses

abs(-1)
min(var1, var2)
sum(var1, var2, var3)

# c is a function that combines values together
my_vars <- c(var1, var2, var3)
my_vars

# RStudio has built-in help for every function
?c

# Functions can also be used to import data
ds <- read.csv('data_raw/vocab16.csv')
print(ds)

# read.csv is part of base R, the default fx set
# When we want to use functions to expand R, we
# need to use library fx to load packages

library(readr)  #for read_csv
ds <- read_csv('data_raw/vocab16.csv')
```

```
[1] 1

[1] 5

[1] 30

[1]  5 10 15

  age   word
1  16  shoes
2  16  berry
3  16     hi
4  16 diaper
5  16  teeth
6  16   uhoh
```

read_csv versus read.csv

- read.csv is base R, read_csv is tidyverse
- read_csv is faster
- read_csv makes fewer assumptions about your data
- read_csv can combine multiple data files into one

The following examples use .csv (comma separated value) files in the data_raw directory



259-langbasi...porting.Rproj
data_cleaned
data_raw
docs
langbasics-importing.R

vocab12.5.csv
vocab12.csv
vocab13.5.csv
vocab14.5.csv
vocab14.csv
vocab15.5.csv
vocab15.csv
vocab16.5.csv
vocab16.csv
vocab17.5.csv
vocab17.csv
vocab18.5.csv
vocab18.csv
vocab19.5.csv
vocab19.csv
vocab20.5.csv
vocab20.csv

```r
# We can use read_csv to read individual files
ds12 <- read_csv('data_raw/vocab12.csv')
```

```r
# We can use read_csv to read individual files
ds12 <- read_csv('data_raw/vocab12.csv')
print(ds12)
```

```
# A tibble: 3 × 2
    age word
  <dbl> <chr>
1    12 book
2    12 ball
3    12 bye bye
```

```r
# We can use read_csv to read individual files
ds12 <- read_csv('data_raw/vocab12.csv')
print(ds12)

ds12.5 <- read_csv('data_raw/vocab12.5.csv')
```

```
# A tibble: 3 × 2
    age word
  <dbl> <chr>
1    12 book
2    12 ball
3    12 bye bye
```

```r
# We can use read_csv to read individual files
ds12 <- read_csv('data_raw/vocab12.csv')
print(ds12)

ds12.5 <- read_csv('data_raw/vocab12.5.csv')
print(ds12.5)
```

```
# A tibble: 3 × 2
    age word
  <dbl> <chr>
1    12 book
2    12 ball
3    12 bye bye

# A tibble: 1 × 2
    age word
  <dbl> <chr>
1  12.5 dad
```

```r
# We can use read_csv to read individual files
ds12 <- read_csv('data_raw/vocab12.csv')
print(ds12)

ds12.5 <- read_csv('data_raw/vocab12.5.csv')
print(ds12.5)

ds13.5 <- read_csv('data_raw/vocab13.5.csv')
```

```
# A tibble: 3 × 2
    age word
  <dbl> <chr>
1    12 book
2    12 ball
3    12 bye bye

# A tibble: 1 × 2
    age word
  <dbl> <chr>
1  12.5 dad
```

```r
# We can use read_csv to read individual files
ds12 <- read_csv('data_raw/vocab12.csv')
print(ds12)

ds12.5 <- read_csv('data_raw/vocab12.5.csv')
print(ds12.5)

ds13.5 <- read_csv('data_raw/vocab13.5.csv')
print(ds13.5)
```

```
# A tibble: 3 × 2
    age word
  <dbl> <chr>
1    12 book
2    12 ball
3    12 bye bye

# A tibble: 1 × 2
    age word
  <dbl> <chr>
1  12.5 dad

# A tibble: 1 × 2
    age word
  <dbl> <chr>
1  13.5 cat
```

```r
# We can use read_csv to read individual files
ds12 <- read_csv('data_raw/vocab12.csv')
print(ds12)

ds12.5 <- read_csv('data_raw/vocab12.5.csv')
print(ds12.5)

ds13.5 <- read_csv('data_raw/vocab13.5.csv')
print(ds13.5)

# bind_rows can append tibbles together
ds_all <- bind_rows(ds12, ds12.5, ds13.5)
```

```
# A tibble: 3 × 2
    age word
  <dbl> <chr>
1    12 book
2    12 ball
3    12 bye bye

# A tibble: 1 × 2
    age word
  <dbl> <chr>
1  12.5 dad

# A tibble: 1 × 2
    age word
  <dbl> <chr>
1  13.5 cat
```

```r
# We can use read_csv to read individual files
ds12 <- read_csv('data_raw/vocab12.csv')
print(ds12)

ds12.5 <- read_csv('data_raw/vocab12.5.csv')
print(ds12.5)

ds13.5 <- read_csv('data_raw/vocab13.5.csv')
print(ds13.5)

# bind_rows can append tibbles together
ds_all <- bind_rows(ds12, ds12.5, ds13.5)
print(ds_all)
```

```
# A tibble: 3 × 2
    age word
  <dbl> <chr>
1    12 book
2    12 ball
3    12 bye bye

# A tibble: 1 × 2
    age word
  <dbl> <chr>
1  12.5 dad

# A tibble: 1 × 2
    age word
  <dbl> <chr>
1  13.5 cat

# A tibble: 5 × 2
    age word
  <dbl> <chr>
1  12   book
2  12   ball
3  12   bye bye
4  12.5 dad
5  13.5 cat
```

But imagine having to read in every file in the list with separate read_csv commands…

```r
#function for listing files a directory
list.files('data_raw', full.names = TRUE)
```

```
 [1] "data_raw/vocab12.5.csv" "data_raw/vocab12.csv"  
 [3] "data_raw/vocab13.5.csv" "data_raw/vocab14.5.csv"
 [5] "data_raw/vocab14.csv"   "data_raw/vocab15.5.csv"
 [7] "data_raw/vocab15.csv"   "data_raw/vocab16.5.csv"
 [9] "data_raw/vocab16.csv"   "data_raw/vocab17.5.csv"
[11] "data_raw/vocab17.csv"   "data_raw/vocab18.5.csv"
[13] "data_raw/vocab18.csv"   "data_raw/vocab19.5.csv"
[15] "data_raw/vocab19.csv"   "data_raw/vocab20.5.csv"
```

```r
# read_csv can read a list of files!

# Make a variable containing the list of data files
full_file_names <- list.files('data_raw', full.names = TRUE)
```

```r
# read_csv can read a list of files!

# Make a variable containing the list of data files
full_file_names <- list.files('data_raw', full.names = TRUE)

# Pass the list to read_csv to read all of them into a single tibble
ds_all <- read_csv(full_file_names)
```

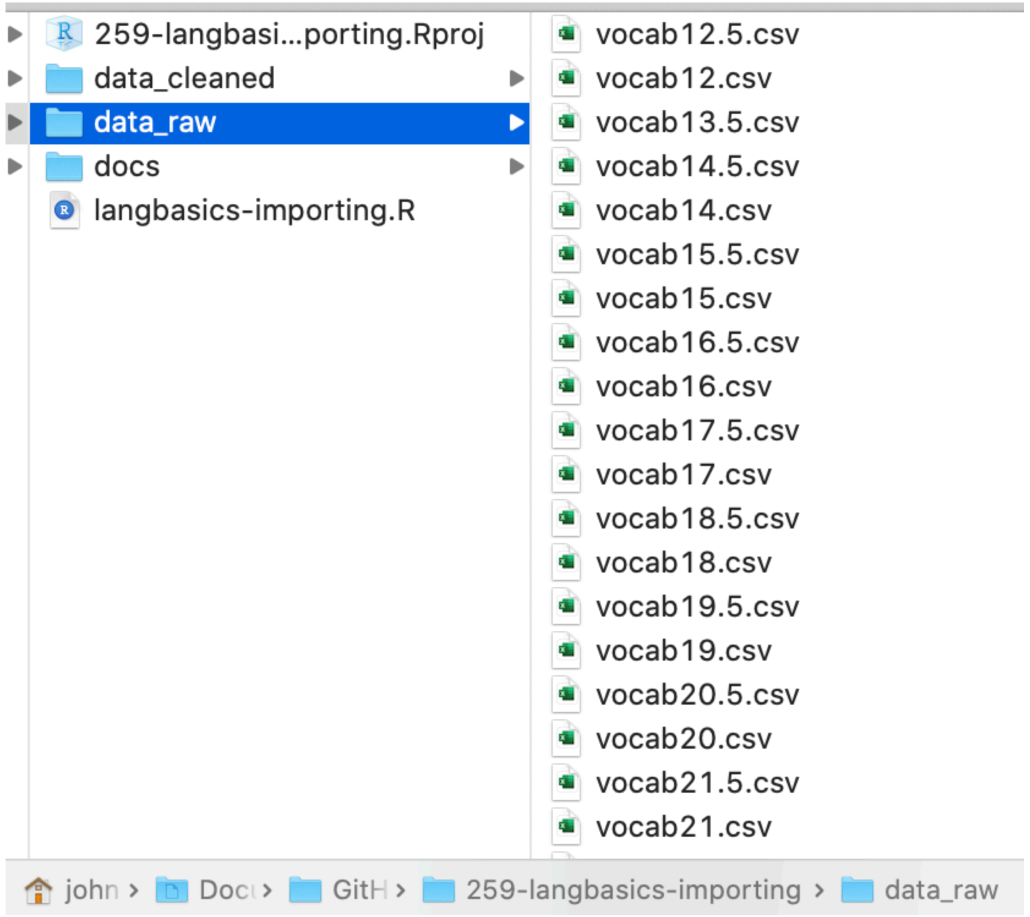
```r
# read_csv can read a list of files!

# Make a variable containing the list of data files
full_file_names <- list.files('data_raw', full.names = TRUE)

# Pass the list to read_csv to read all of them into a single tibble
ds_all <- read_csv(full_file_names)
print(ds_all)
```

```
# A tibble: 440 × 2
     age word
   <dbl> <chr>
 1  12.5 dad
 2  12   book
 3  12   ball
 4  12   bye bye
 5  13.5 cat
 6  14.5 socks/shoes
 7  14.5 dog (animal)
 8  14   baby
 9  15.5 cheese
10  15.5 turkey
# … with 430 more rows
```

Why did we need full.names?

- `list.files('data_raw')` gives `vocab12.csv`, `vocab12.5.csv`...which aren't in the working directory
- `list.files('data_raw', full.names = TRUE)` gives the relative path including the directory:

`/data_raw/vocab12.csv`, `/data_raw/vocab12.5.csv`


259-langbasi...porting.Rproj
data_cleaned
data_raw
docs
langbasics-importing.R

vocab12.5.csv
vocab12.csv
vocab13.5.csv
vocab14.5.csv
vocab14.csv
vocab15.5.csv
vocab15.csv
vocab16.5.csv
vocab16.csv
vocab17.5.csv
vocab17.csv
vocab18.5.csv
vocab18.csv
vocab19.5.csv
vocab19.csv
vocab20.5.csv
vocab20.csv
vocab21.5.csv
vocab21.csv

john › Docı › GitH › 259-langbasics-importing › data_raw

What's a tibble?

- Tibbles are the tidyverse equivalent of base R data frames.
- A tibble/data frame is a rectangular data spreadsheet, with columns of variables and rows of observations
- Tibbles/data frames will always appear in the "Data" section of the RStudio environment
- If you click on the blue "play" button on a tibble in your RStudio environment, you can view a tibble
- Unlike in Excel, you cannot edit the values. This is a feature, not a bug!
- `ds$variable` lets you access one variable of a data set

```r
# Find the minimum and maximum ages
min(ds_all$age)
```

```
[1] 12
```

```r
# Find the minimum and maximum ages
min(ds_all$age)
max(ds_all$age)
```

```
[1] 12
```

```
[1] 24
```

```r
# Find the minimum and maximum ages
min(ds_all$age)
max(ds_all$age)

# Create a new column in a dataset
ds_all$ppt_name <- "Jonah"
```

```
[1] 12

[1] 24
```

```r
# Find the minimum and maximum ages
min(ds_all$age)
max(ds_all$age)

# Create a new column in a dataset
ds_all$ppt_name <- "Jonah"

# Create a calculated column
ds_all$age_round <- round(ds_all$age)
```

```
[1] 12

[1] 24
```

```r
# Find the minimum and maximum ages
min(ds_all$age)
max(ds_all$age)

# Create a new column in a dataset
ds_all$ppt_name <- "Jonah"

# Create a calculated column
ds_all$age_round <- round(ds_all$age)

# See the results
print(ds_all)
```

```
[1] 12

[1] 24

# A tibble: 440 × 4
     age word         ppt_name age_round
   <dbl> <chr>        <chr>        <dbl>
 1  12.5 dad          Jonah           12
 2  12   book         Jonah           12
 3  12   ball         Jonah           12
 4  12   bye bye      Jonah           12
 5  13.5 cat          Jonah           14
 6  14.5 socks/shoes  Jonah           14
 7  14.5 dog (animal) Jonah           14
 8  14   baby         Jonah           14
 9  15.5 cheese       Jonah           16
10  15.5 turkey       Jonah           16
# … with 430 more rows
```

```r
# Find the minimum and maximum ages
min(ds_all$age)
max(ds_all$age)

# Create a new column in a dataset
ds_all$ppt_name <- "Jonah"

# Create a calculated column
ds_all$age_round <- round(ds_all$age)

# See the results
print(ds_all)

# Let's write the combined data to disk
write_csv(ds_all, file = "data_cleaned/vocab_combined.csv")
```

```
[1] 12

[1] 24

# A tibble: 440 × 4
     age word          ppt_name age_round
   <dbl> <chr>         <chr>        <dbl>
 1  12.5 dad           Jonah           12
 2  12   book          Jonah           12
 3  12   ball          Jonah           12
 4  12   bye bye       Jonah           12
 5  13.5 cat           Jonah           14
 6  14.5 socks/shoes   Jonah           14
 7  14.5 dog (animal)  Jonah           14
 8  14   baby          Jonah           14
 9  15.5 cheese        Jonah           16
10  15.5 turkey        Jonah           16
# … with 430 more rows
```

Useful readr capabilities

- read_csv(), read_tsv(), read_delim() are tailored to different inputs (also write_csv(), write_tsv(), write_delim() for saving data)
- Important read_*() arguments are:
  - `col_names = TRUE` (reads column names from first line by default)
  - `col_names = FALSE` (treats the first line as data)
  - `col_names = c("col1name", "col2name")` (to specify the names)
  - `col_types = NULL` (by default, guesses the data type)
  - `col_types = "ccDin"` (specify types character/date/integer/number)
  - `skip = 10` (skip the first 10 lines)

```r
fname <- "data_cleaned/vocab_combined.csv"
colname <- c("AGE", "WORD", "NAME", "MONTH")
coltypes <- "cccc"
ds <- read_csv(file = fname)
print(ds)
```

```
# A tibble: 440 × 4
     age word         ppt_name age_round
   <dbl> <chr>        <chr>        <dbl>
 1  12.5 dad          Jonah           12
 2  12   book         Jonah           12
 3  12   ball         Jonah           12
 4  12   bye bye      Jonah           12
 5  13.5 cat          Jonah           14
 6  14.5 socks/shoes  Jonah           14
 7  14.5 dog (animal) Jonah           14
 8  14   baby         Jonah           14
 9  15.5 cheese       Jonah           16
10  15.5 turkey       Jonah           16
# … with 430 more rows
```

```r
fname <- "data_cleaned/vocab_combined.csv"
colname <- c("AGE", "WORD", "NAME", "MONTH")
coltypes <- "cccc"
ds <- read_csv(file = fname, col_names = FALSE)
print(ds)
```

```
# A tibble: 441 × 4
   X1    X2           X3       X4
   <chr> <chr>        <chr>    <chr>
 1 age   word         ppt_name age_round
 2 12.5  dad          Jonah    12
 3 12    book         Jonah    12
 4 12    ball         Jonah    12
 5 12    bye bye      Jonah    12
 6 13.5  cat          Jonah    14
 7 14.5  socks/shoes  Jonah    14
 8 14.5  dog (animal) Jonah    14
 9 14    baby         Jonah    14
10 15.5  cheese       Jonah    16
# … with 431 more rows
```

```r
fname <- "data_cleaned/vocab_combined.csv"
colname <- c("AGE", "WORD", "NAME", "MONTH")
coltypes <- "cccc"
ds <- read_csv(file = fname, col_names = colname)
print(ds)
```

```
# A tibble: 441 × 4
   AGE   WORD         NAME     MONTH
   <chr> <chr>        <chr>    <chr>
 1 age   word         ppt_name age_round
 2 12.5  dad          Jonah    12
 3 12    book         Jonah    12
 4 12    ball         Jonah    12
 5 12    bye bye      Jonah    12
 6 13.5  cat          Jonah    14
 7 14.5  socks/shoes  Jonah    14
 8 14.5  dog (animal) Jonah    14
 9 14    baby         Jonah    14
10 15.5  cheese       Jonah    16
# … with 431 more rows
```

```r
fname <- "data_cleaned/vocab_combined.csv"
colname <- c("AGE", "WORD", "NAME", "MONTH")
coltypes <- "cccc"
ds <- read_csv(file = fname, col_names = colname, skip = 1)
print(ds)
```

```
# A tibble: 440 × 4
     AGE WORD         NAME  MONTH
   <dbl> <chr>        <chr> <dbl>
 1  12.5 dad          Jonah    12
 2  12   book         Jonah    12
 3  12   ball         Jonah    12
 4  12   bye bye      Jonah    12
 5  13.5 cat          Jonah    14
 6  14.5 socks/shoes  Jonah    14
 7  14.5 dog (animal) Jonah    14
 8  14   baby         Jonah    14
 9  15.5 cheese       Jonah    16
10  15.5 turkey       Jonah    16
# … with 430 more rows
```

```r
fname <- "data_cleaned/vocab_combined.csv"
colname <- c("AGE", "WORD", "NAME", "MONTH")
coltypes <- "cccc"
ds <- read_csv(file = fname, col_names = colname, skip = 1, col_types = coltypes)
print(ds)
```

```
# A tibble: 440 × 4
   AGE   WORD         NAME  MONTH
   <chr> <chr>        <chr> <chr>
 1 12.5  dad          Jonah 12
 2 12    book         Jonah 12
 3 12    ball         Jonah 12
 4 12    bye bye      Jonah 12
 5 13.5  cat          Jonah 14
 6 14.5  socks/shoes  Jonah 14
 7 14.5  dog (animal) Jonah 14
 8 14    baby         Jonah 14
 9 15.5  cheese       Jonah 16
10 15.5  turkey       Jonah 16
# … with 430 more rows
```

More options

- tidyverse data import "cheatsheets"
- Read the documentation: `?read_csv`, `?write_csv`
- specialized import packages
    - `haven` for SPSS/Stata/SAS
    - `readxl` for .xlsx
    - `googlesheets4` for Google sheets

Data import with the tidyverse : : CHEAT SHEET

Read Tabular Data with readr

read_*(file, col_names = TRUE, col_types = NULL, col_select = NULL, id = NULL, locale, n_max = Inf, skip = 0, na = c("", "NA"), guess_max = min(1000, n_max), show_col_types = TRUE) See **?read_delim**

```
A|B|C
1|2|3
4|5|NA
```
→

A	B	C
1	2	3
4	5	NA

read_delim("file.txt", delim = "|") Read files with any delimiter. If no delimiter is specified, it will automatically guess.
To make file.txt, run: write_file("A|B|C\n1|2|3\n4|5|NA", file = "file.txt")

```
A,B,C
1,2,3
4,5,NA
```
→

A	B	C
1	2	3
4	5	NA

read_csv("file.csv") Read a comma delimited file with period decimal marks.
write_file("A,B,C\n1,2,3\n4,5,NA", file = "file.csv")

```
A;B;C
1,5;2;3
4,5;5;NA
```
→

A	B	C
1.5	2	3
4.5	5	NA

read_csv2("file2.csv") Read semicolon delimited files with comma decimal marks.
write_file("A;B;C\n1,5;2;3\n4,5;5;NA", file = "file2.csv")

```
A B C
1 2 3
4 5 NA
```
→

A	B	C
1	2	3
4	5	NA

read_tsv("file.tsv") Read a tab delimited file. Also **read_table()**.
read_fwf("file.tsv", fwf_widths(c(2, 2, NA))) Read a fixed width file.
write_file("A\tB\tC\n1\t2\t3\n4\t5\tNA\n", file = "file.tsv")

USEFUL READ ARGUMENTS

A	B	C
1	2	3
4	5	NA

No header
read_csv("file.csv", col_names = FALSE)

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header
read_csv("file.csv",
col_names = c("x", "y", "z"))

Read multiple files into a single table
read_csv(c("f1.csv", "f2.csv", "f3.csv"),
id = "origin_file")

1	2	3
4	5	NA

Skip lines
read_csv("file.csv", skip = 1)

A	B	C
1	2	3

Read a subset of lines
read_csv("file.csv", n_max = 1)

A	B	C
NA	2	3
4	5	NA

Read values as missing
read_csv("file.csv", na = c("1"))

```
A;B;C
1,5;2;3,0
```

Specify decimal marks
read_delim("file2.csv", locale =
locale(decimal_mark = ","))

Save Data with readr

write_*(x, file, na = "NA", append, col_names, quote, escape, eol, num_threads, progress)

A	B	C
1	2	3
4	5	NA

→
```
A,B,C
1,2,3
4,5,NA
```

write_delim(x, file, delim = " ") Write files with any delimiter.

write_csv(x, file) Write a comma delimited file.

write_csv2(x, file) Write a semicolon delimited file.

write_tsv(x, file) Write a tab delimited file.

One of the first steps of a project is to import outside data into R. Data is often stored in tabular formats, like csv files or spreadsheets.

The front page of this sheet shows how to import and save text files into R using **readr**.

The back page shows how to import spreadsheet data from Excel files using **readxl** or Google Sheets using **googlesheets4**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files:

- **haven** - SPSS, Stata, and SAS files
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)
- **readr::read_lines()** - text data

Column Specification with readr

Column specifications define what data type each column of a file will be imported as. By default readr will generate a column spec when a file is read and output a summary.

spec(x) Extract the full column specification for the given imported data frame.

```
spec(x)
# cols(
#   age = col_integer(),
#   sex = col_character(),
#   earn = col_double()
# )
```

age is an integer

sex is a character

earn is a double (numeric)

USEFUL COLUMN ARGUMENTS

Hide col spec message
read_*(file, show_col_types = FALSE)

Select columns to import
Use names, position, or selection helpers.
read_*(file, col_select = c(age, earn))

Guess column types
To guess a column type, read_ *() looks at the first 1000 rows of data. Increase with **guess_max**.
read_*(file, guess_max = Inf)

COLUMN TYPES

Each column type has a function and corresponding string abbreviation.

- **col_logical()** - "l"
- **col_integer()** - "i"
- **col_double()** - "d"
- **col_number()** - "n"
- **col_character()** - "c"
- **col_factor**(levels, ordered = FALSE) - "f"
- **col_datetime**(format = "") - "T"
- **col_date**(format = "") - "D"
- **col_time**(format = "") - "t"
- **col_skip()** - "-", "_"
- **col_guess()** - "?"

DEFINE COLUMN SPECIFICATION

Set a default type
```
read_csv(
  file,
  col_type = list(.default = col_double())
)
```

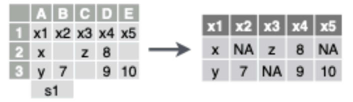
Use column type or string abbreviation
```
read_csv(
  file,
  col_type = list(x = col_double(), y = "l", z = "_")
)
```

Use a single string of abbreviations
```
# col types: skip, guess, integer, logical, character
read_csv(
  file,
  col_type = "_?ilc"
)
```

Import Spreadsheets

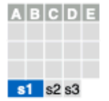
with readxl

READ EXCEL FILES

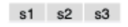


read_excel(path, sheet = NULL, range = NULL)
Read a .xls or .xlsx file based on the file extension. See front page for more read arguments. Also **read_xls()** and **read_xlsx()**.
read_excel("excel_file.xlsx")

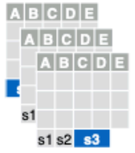
READ SHEETS



read_excel(path, **sheet = NULL)** Specify which sheet to read by position or name.
read_excel(path, sheet = 1)
read_excel(path, sheet = "s1")



excel_sheets(path) Get a vector of sheet names.
excel_sheets("excel_file.xlsx")



To **read multiple sheets:**
1. Get a vector of sheet names from the file path.
2. Set the vector names to be the sheet names.
3. Use purrr::map_dfr() to read multiple files into one data frame.

```
path <- "your_file_path.xlsx"
path %>% excel_sheets() %>%
  set_names() %>%
  map_dfr(read_excel, path = path)
```

READXL COLUMN SPECIFICATION

Column specifications define what data type each column of a file will be imported as.

Use the **col_types** argument of **read_excel()** to set the column specification.

Guess column types
To guess a column type, read_ excel() looks at the first 1000 rows of data. Increase with the **guess_max** argument.
read_excel(path, **guess_max** = Inf)

Set all columns to same type, e.g. character
read_excel(path, col_types = "text")

Set each column individually
```
read_excel(
  path,
  col_types = c("text", "guess", "guess","numeric")
)
```

COLUMN TYPES

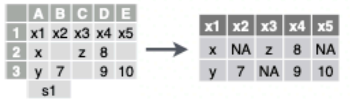
logical	numeric	text	date	list
TRUE	2	hello	1947-01-08	hello
FALSE	3.45	world	1956-10-21	1

- skip
- guess
- logical
- numeric
- text
- date
- list

Use **list** for columns that include multiple data types. See **tidyr** and **purrr** for list-column data.

with googlesheets4

READ SHEETS



read_sheet(ss, sheet = NULL, range = NULL)
Read a sheet from a URL, a Sheet ID, or a dribble from the googledrive package. See front page for more read arguments. Same as **range_read()**.

SHEETS METADATA

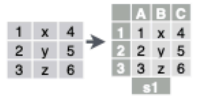
URLs are in the form:
https://docs.google.com/spreadsheets/d/
SPREADSHEET_ID/edit#gid=**SHEET_ID**

gs4_get(ss) Get spreadsheet meta data.

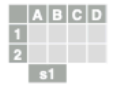
gs4_find(...) Get data on all spreadsheet files.

sheet_properties(ss) Get a tibble of properties for each worksheet. Also **sheet_names()**.

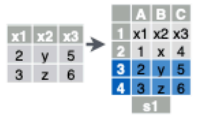
WRITE SHEETS



write_sheet(data, ss = NULL, sheet = NULL) Write a data frame into a new or existing Sheet.



gs4_create(name, ..., sheets = NULL) Create a new Sheet with a vector of names, a data frame, or a (named) list of data frames.



sheet_append(ss, data, sheet = 1) Add rows to the end of a worksheet.

GOOGLESHEETS4 COLUMN SPECIFICATION

Column specifications define what data type each column of a file will be imported as.

Use the **col_types** argument of **read_sheet()/ range_read()** to set the column specification.

Guess column types
To guess a column type read_sheet()/ range_read() looks at the first 1000 rows of data. Increase with **guess_max**.
read_sheet(path, **guess_max** = Inf)

Set all columns to same type, e.g. character
read_sheet(path, col_types = "c")

Set each column individually
```
# col types: skip, guess, integer, logical, character
read_sheets(ss, col_types = "_?ilc")
```

COLUMN TYPES

l	n	c	D	L
TRUE	2	hello	1947-01-08	hello
FALSE	3.45	world	1956-10-21	1

- skip - "_" or "-"
- guess - "?"
- logical - "l"
- integer - "i"
- double - "d"
- numeric - "n"
- date - "D"
- datetime - "T"
- character - "c"
- list-column - "L"
- cell - "C" Returns list of raw cell data.

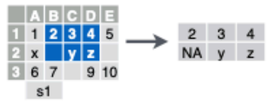
Use list for columns that include multiple data types. See **tidyr** and **purrr** for list-column data.

OTHER USEFUL EXCEL PACKAGES

For functions to write data to Excel files, see:
- **openxlsx**
- **writexl**

For working with non-tabular Excel data, see:
- **tidyxl**

CELL SPECIFICATION FOR READXL AND GOOGLESHEETS4



Use the **range** argument of **readxl::read_excel()** or **googlesheets4::read_sheet()** to read a subset of cells from a sheet.
read_excel(path, range = "Sheet1!B1:D2")
read_sheet(ss, range = "B1:D2")

Also use the range argument with cell specification functions **cell_limits()**, **cell_rows()**, **cell_cols()**, and **anchored()**.

FILE LEVEL OPERATIONS

googlesheets4 also offers ways to modify other aspects of Sheets (e.g. freeze rows, set column width, manage (work)sheets). Go to **googlesheets4.tidyverse.org** to read more.

For whole-file operations (e.g. renaming, sharing, placing within a folder), see the tidyverse package **googledrive** at **googledrive.tidyverse.org**.

Data import homework assignment

- Use the GitHub repo link below to clone the project (no need to fork) to your own user account and then to work locally on your own computer.
- Be sure you are using R version >= 4.0 and readr version >= 2.0
- The homework will make sure you learned what we covered, and will also ask you to try out new things to extend your knowledge
- Push your answers to a public Github repo, and turn in the homework by entering the link to the repo on Canvas

Homework Github repo