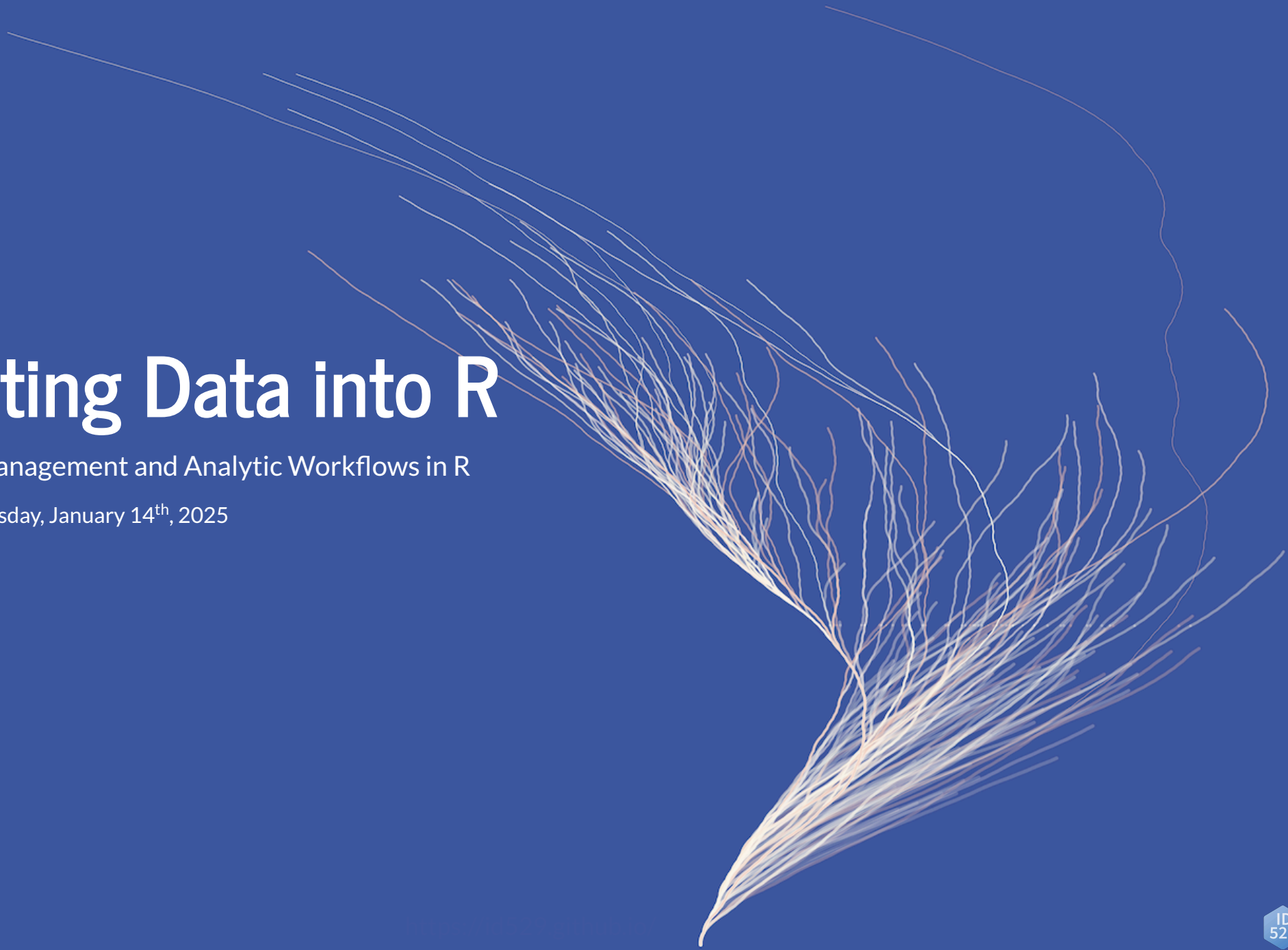
Importing Data into R

ID 529: Data Management and Analytic Workflows in R

Dean Marengi | Tuesday, January 14th, 2025



Motivation

- **So far, we've learned a bit about:**
  - **Git and GitHub** for code version control and management
  - **R and R Studio** as tools for working with and analyzing data

- **Before data cleaning, analysis, and visualization can begin data must be imported into R**
  - First step in a data analysis
  - Reading data into R is often straightforward
  - However, it is important to be aware of idiosyncrasies in data storage methods to ensure:
    - The data import process is efficient
    - The accuracy and integrity of data is maintained

Learning objectives

- **Learn about common data storage methods you may encounter as researcher**
  - Flat data files
  - Statistical software files
  - Excel files
- **Learn different approaches for reading data into R**
  - Base R functions
  - R packages with functions to accommodate more file types
- **Understand the potential challenges you may face when reading data into R**
  - File types
  - Data formatting
  - Data types

Data Storage Methods

Flat files

- **Class of file types that store data in a simple, text-based format**
  - Each row represents a data record
  - Each column represents a variable
  - **Parsed using a delimiter** (comma, tab, etc.)
  - **Highly compatible** with most software tools
    - No formatting or other features
    - Small file size relative to other formats
- **Flat file types include:**
  - `csv`: comma-separated values
  - `txt`: plain text with delimiter
    - values separated by character (e.g., `;` `|`)
    - fixed-width files (`fwf`)
    - tab-separated values (`tsv`)

```
id,survdate,age,edu,smoke,cigs,sbp,dbp,chol,glucose
32955,2019-11-04,54,2,1,15,230,110,262,97
36997,2019-10-31,49,3,1,9,110,76,281,78
34659,2019-07-14,57,1,0,0,146,76,258,87
36762,2019-04-18,58,2,1,60,150,97,250,65
32197,2019-02-20,56,1,1,3,145,100,285,86
33534,2019-07-07,45,3,1,23,115,76,263,78
36763,2019-01-04,37,1,1,20,112,73.5,166,93
33026,2018-09-18,48,1,1,25,102,66.5,304,66
35364,2019-06-13,62,1,0,0,130,87,157,47
34177,2019-03-19,54,1,0,0,136.5,83,254,95
32650,2017-03-20,45,3,0,0,121,78.5,222,
32631,2019-01-17,47,2,1,3,120,80,198,76
34380,2017-03-15,42,1,1,10,111,67.5,194,47
```

```
id;survdate;age;edu;smoke;cigs;sbp;dbp;chol;glucose
32955;2019-11-04;54;2;1;15;230;110;262;97
36997;2019-10-31;49;3;1;9;110;76;281;78
34659;2019-07-14;57;1;0;0;146;76;258;87
36762;2019-04-18;58;2;1;60;150;97;250;65
32197;2019-02-20;56;1;1;3;145;100;285;86
33534;2019-07-07;45;3;1;23;115;76;263;78
36763;2019-01-04;37;1;1;20;112;73.5;166;93
33026;2018-09-18;48;1;1;25;102;66.5;304;66
35364;2019-06-13;62;1;0;0;130;87;157;47
34177;2019-03-19;54;1;0;0;136.5;83;254;95
32650;2017-03-20;45;3;0;0;121;78.5;222;NA
32631;2019-01-17;47;2;1;3;120;80;198;76
34380;2017-03-15;42;1;1;10;111;67.5;194;47
```

```
id      survdate age     edu     smoke   cigs    sbp     dbp     chol    glucose
32955   2019-11-04      54      2       1       15      230     110     262     97
36997   2019-10-31      49      3       1       9       110     76      281     78
34659   2019-07-14      57      1       0       0       146     76      258     87
36762   2019-04-18      58      2       1       60      150     97      250     65
32197   2019-02-20      56      1       1       3       145     100     285     86
33534   2019-07-07      45      3       1       23      115     76      263     78
36763   2019-01-04      37      1       1       20      112     73.5    166     93
33026   2018-09-18      48      1       1       25      102     66.5    304     66
35364   2019-06-13      62      1       0       0       130     87      157     47
34177   2019-03-19      54      1       0       0       136.5   83      254     95
```

```
3230559 20 0244.0124.0264120
3484062 11  242.0141.0206 94
3568665 10 0235.0100.0240297
3573556 1120200.0120.0217 72
3351661 10 0200.0125.0265256
3469763 30 0196.0102.0161 88
3382758 10 0195.0 90.0294127
3346766 30 0193.0132.0280 73
3563665 2120192.5110.0212 71
3609445 2120189.0 87.0234 90
3535361 10 0189.0121.0217100
3546966 10 0188.0 92.0280101
3266046 20 0188.0123.0184
3538265 10 0186.0109.0202 75
```

Excel files

- **Excel can store data in file formats other than a csv**
  - `xlsx` and the older `xls` format
- Supports Excel formatting, formulas, or other features
- Distinct from flat data files, which store data in a simple text format

- **Drawbacks include:**
  - **Less portable** between statistical software packages and data systems
  - **Require specific import functions** to accommodate unique data storage features
  - **Compatibility issues from updates** to proprietary file formats over time

Statistical software files

- **Most statistical software packages have proprietary file formats (binary files)**
  - **R**: `rds` and `rda`
  - **Stata**: `dta`
  - **SAS**: `sas7bdat`
  - **SPSS**: `sav`
- Optimized storage and use in the software for which they were developed
- Leverage internal software capabilities not available in other formats

- **Drawbacks include:**
  - **Less portable** between statistical software packages and data systems
  - **Require specific import functions** to accommodate unique data storage features
    - Data types, layout, or other attributes, such as variable labels

Relational databases

- **Databases are efficient, scalable, and secure modes of data storage**
- Several R packages enable users to connect to databases to read data. For example:
  - `RODBC`
  - `RMySQL`
  - `RSQLite`
  - `RPostgreSQL`
- Users must supply database credentials and other information to connect to databases
  - Hostname, port number, and other parameters
- Database tables can be queried from R once a connection is established

How can we import data into R?

- **Base R read functions**
  - Installed as part of the core R distribution
  - Fast, simple solution for data imports
  - Limited range of file types supported
- **Packages for reading data**
  - User-friendly functions with good documentation
  - Functions that accommodate many data formats
  - Robust error handling
- **The read functions you use will ultimately depend on:**
  - The unique needs of your project
  - The characteristics of the underlying data

Base R read functions

Base R read funtions

- Reading tabular data from flat and rds files
- Imported data stored as a data frame
- Number of function arguments to customize data import
- **Core functions:**
  - `read.table()`: multiple file types
  - `read.csv()`: comma-separated values (csv)
  - `read.delim()`: delimited text files
  - `read.fwf()`: fixed-width text files
  - `readRDS()`: R data stream (rds) files
  - ^*Notice the* `.` *used in function names*
- Above functions are wrappers around `read.table` (except `readRDS`)
- Run `?read.table()` in R Studio console to learn more

```
read.table(
  file,
  header = FALSE,
  sep = "",
  quote = "\"'",
  dec = ".",
  numerals = c("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,
  stringsAsFactors = FALSE,
  fileEncoding = ""
  )
```

Example: `read.csv()`

```r
# Read in the data
data <- read.csv("data/cvdstudy.csv")

# Print
data
```

```
        id   survdate age edu smoke cigs   sbp   dbp chol glucose
1    32955 2019-11-04  54   2     1   15 230.0 110.0  262      97
2    36997 2019-10-31  49   3     1    9 110.0  76.0  281      78
3    34659 2019-07-14  57   1     0    0 146.0  76.0  258      87
4    36762 2019-04-18  58   2     1   60 150.0  97.0  250      65
5    32197 2019-02-20  56   1     1    3 145.0 100.0  285      86
6    33534 2019-07-07  45   3     1   23 115.0  76.0  263      78
7    36763 2019-01-04  37   1     1   20 112.0  73.5  166      93
8    33026 2018-09-18  48   1     1   25 102.0  66.5  304      66
9    35364 2019-06-13  62   1     0    0 130.0  87.0  157      47
10   34177 2019-03-19  54   1     0    0 136.5  83.0  254      95
11   32650 2017-03-20  45   3     0    0 121.0  78.5  222      NA
12   32631 2019-01-17  47   2     1    3 120.0  80.0  198      76
13   34380 2017-03-15  42   1     1   10 111.0  67.5  194      47
14   36118 2018-10-21  46   1     0    0 145.0  90.0  295      79
15   35064 2019-07-04  39   1     1   30 139.0  90.0  300     107
16   35134 2018-11-27  39   3     0    0 125.0  87.0  213      75
17   34949 2018-02-14  54   1     0    0 123.0  75.0  250      71
18   35208 2018-05-18  45   2     1   20 117.5  76.0  311      67
19   33548 2019-06-05  46   2     1   30 143.5  92.0  235     115
20   36513 2019-02-28  45   1     0    0 127.5  83.5  238      78
21   33580 2018-05-09  58   1     0    0  85.5  51.0  260     206
22   36734 2018-12-13  55   2     1    9 157.0  82.5  248      83
23   36715 2019-10-21  54   2     1    3 127.5  83.0  231     115
24   32719 2018-04-14  41   3     1   20 166.0  71.0  240      57
25   34485 2019-07-21  46   1     1   40 154.0  91.0  210      82
```

R Packages for Importing Data

readr

- Reading tabular data from csv, flat, and rds files
- Part of the **core** `tidyverse` package ecosystem
- Imported data stored as **tibble data frame**
- Informative error and warning messages

- **Core functions:**
  - `read_table()`: multiple file types
  - `read_csv()`: comma-separated values (csv)
  - `read_tsv()`: tab-separated values (tsv)
  - `read_delim()`: delimited text files
  - `read_fwf()`: fixed-width text files
  - `read_rds()`: R data stream (rds) files
  - ^*Notice the _ used in function names*

```r
read_csv(
  file,
  col_names = TRUE,
  col_types = NULL,
  col_select = NULL,
  id = NULL,
  locale = default_locale(),
  na = c("", "NA"),
  quoted_na = TRUE,
  quote = "\"",
  comment = "",
  trim_ws = TRUE,
  skip = 0,
  n_max = Inf,
  guess_max = min(1000, n_max),
  name_repair = "unique",
  num_threads = readr_threads(),
  progress = show_progress(),
  show_col_types = should_show_types(),
  skip_empty_rows = TRUE,
  lazy = should_read_lazy()
)
```

Example: `read_csv()`

```r
# Load readr package (not necessary if you've loaded the tidyverse)
library(readr)

# Read in data, which is stored in a csv file
data <- read_csv("data/cvdstudy.csv", na = "")

# Look at the data
data
```

```
# A tibble: 2,401 × 10
      id survdate      age   edu smoke  cigs   sbp   dbp  chol glucose
   <dbl> <date>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
 1 32955 2019-11-04    54     2     1    15  230   110     262      97
 2 36997 2019-10-31    49     3     1     9  110    76     281      78
 3 34659 2019-07-14    57     1     0     0  146    76     258      87
 4 36762 2019-04-18    58     2     1    60  150    97     250      65
 5 32197 2019-02-20    56     1     1     3  145   100     285      86
 6 33534 2019-07-07    45     3     1    23  115    76     263      78
 7 36763 2019-01-04    37     1     1    20  112    73.5   166      93
 8 33026 2018-09-18    48     1     1    25  102    66.5   304      66
 9 35364 2019-06-13    62     1     0     0  130    87     157      47
10 34177 2019-03-19    54     1     0     0  136.   83     254      95
# ℹ 2,391 more rows
```

- Returns a tibble, which displays useful information about the dataset
- Note that these functions guess data types for each column, and are not always correct
- **Let's try explicitly defining data types for `id`, `smoke`, and `edu` during the import!**

Example: `read_csv()`

```r
# Load readr package (not necessary if you've loaded the tidyverse)
library(readr)

# Read in data and specify data types for specific columns
data <- read_csv("data/cvdstudy.csv", na = "",
                 col_types = cols(
                   id = col_character(),
                   smoke = col_factor(levels = c("0", "1")),
                   edu = col_factor(levels = c("1", "2", "3", "4"))
                   )
                 )

# Look at the data
data
```

```
# A tibble: 2,401 × 10
   id    survdate     age edu   smoke  cigs   sbp   dbp  chol glucose
   <chr> <date>     <dbl> <fct> <fct> <dbl> <dbl> <dbl> <dbl>   <dbl>
 1 32955 2019-11-04    54 2     1        15  230   110     262      97
 2 36997 2019-10-31    49 3     1         9  110    76     281      78
 3 34659 2019-07-14    57 1     0         0  146    76     258      87
 4 36762 2019-04-18    58 2     1        60  150    97     250      65
 5 32197 2019-02-20    56 1     1         3  145   100     285      86
 6 33534 2019-07-07    45 3     1        23  115    76     263      78
 7 36763 2019-01-04    37 1     1        20  112    73.5   166      93
 8 33026 2018-09-18    48 1     1        25  102    66.5   304      66
 9 35364 2019-06-13    62 1     0         0  130    87     157      47
10 34177 2019-03-19    54 1     0         0  136.   83     254      95
# ℹ 2,391 more rows
```

- We did it! Many read functions include arguments to define data types or address other issues
- **Always look at your data, and inspect it for potential data import problems**

Example: `read_delim()`

```r
# Read in data and specify data types for specific columns and the delimiter used in the file
data <- read_delim("data/cvdstudy.txt", delim = "\t", na = c("N/A", "NA"),
                   col_types = cols(
                     id = col_character(),
                     smoke = col_factor(levels = c("0", "1")),
                     edu = col_factor(levels = c("1", "2", "3", "4"))
                   )
                  )

# Look at the data
data
```

```
# A tibble: 2,401 × 10
   id    survdate     age edu   smoke  cigs   sbp   dbp  chol glucose
   <chr> <date>     <dbl> <fct> <fct> <dbl> <dbl> <dbl> <dbl>   <dbl>
 1 32955 2019-11-04    54 2     1        15  230   110     262      97
 2 36997 2019-10-31    49 3     1         9  110    76     281      78
 3 34659 2019-07-14    57 1     0         0  146    76     258      87
 4 36762 2019-04-18    58 2     1        60  150    97     250      65
 5 32197 2019-02-20    56 1     1         3  145   100     285      86
 6 33534 2019-07-07    45 3     1        23  115    76     263      78
 7 36763 2019-01-04    37 1     1        20  112    73.5   166      93
 8 33026 2018-09-18    48 1     1        25  102    66.5   304      66
 9 35364 2019-06-13    62 1     0         0  130    87     157      47
10 34177 2019-03-19    54 1     0         0  136.   83     254      95
# ℹ 2,391 more rows
```

- Notice that the syntax is identical to read_csv, with the exception of the `delim` argument
  - `\t` is passed to the `delim` argument to indicate a tab delimiter
- We can also indicate `NA` and `N/A` as character strings to interpret as missing values
  - The source data includes the text “NA” or “N/A” in place of missing values

readxl

- Reading tabular data from Excel files
- Part of the `tidyverse` package ecosystem
  - Requires separate package installation
  - Imported data stored as **tibble data frame**
  - Informative error and warning messages
- **Core functions:**
  - `read_excel()`: xls or xlsx (uses file extension)
  - `read_xlsx()`: xlsx only
  - `read_xls()`: xls only

```r
read_xlsx(
  path,
  sheet = NULL,
  range = NULL,
  col_names = TRUE,
  col_types = NULL,
  na = "",
  trim_ws = TRUE,
  skip = 0,
  n_max = Inf,
  guess_max = min(1000, n_max),
  progress = readxl_progress(),
  .name_repair = "unique"
)
```

Example: `read_xlsx()`

```
# Load readxl package
library(readxl)

# Read in the xlsx file, which we know is located on sheet 2 inside of the file
data <- read_xlsx("data/cvdstudy.xlsx", sheet = 2, na = "")

# Look at the data
data
```

```
# A tibble: 2,411 × 10
   `Summary statistics` ...2     ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10
   <chr>                <chr>    <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 <NA>                 Age      SBP   DBP   Chol  Gluc… <NA>  <NA>  <NA>  <NA>
 2 Mean                 49.9433… 132.… 82.8… 236.… 49.9… <NA>  <NA>  <NA>  <NA>
 3 Std. Dev             8.65920… 21.8… 11.8… 45.4… 8.65… <NA>  <NA>  <NA>  <NA>
 4 Min                  33       85.5  48    124   40    <NA>  <NA>  <NA>  <NA>
 5 Max                  70       244   141   696   370   <NA>  <NA>  <NA>  <NA>
 6 <NA>                 <NA>     <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 <NA>                 <NA>     <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>                 <NA>     <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>                 <NA>     <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 id                   survdate age   edu   smoke cigs  sbp   dbp   chol  gluc…
# ℹ 2,401 more rows
```

- Oh no! A pesky collaborator included summary statistics at the top of the excel file
- **Not to worry! Let's take advantage of the `skip` argument to read in the correct data**

Example: `read_xlsx()`

```r
# Create a vector of column types (limited to skip, guess, logical, numeric, date, text or list)
col_types <- c("text", "date", "numeric", "guess", "guess", "numeric",
               "numeric", "numeric", "numeric", "numeric")

# Read in the xlsx file, which we know is located on sheet 2 inside of the file
data <- read_xlsx(
  "data/cvdstudy.xlsx",
  sheet = 2,
  skip = 10,
  col_names = TRUE,
  col_types = col_types
)

# Look at the data
data
```

```
# A tibble: 2,401 × 10
   id    survdate   age   edu smoke  cigs   sbp   dbp  chol glucose
   <chr> <dttm>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
 1 32955 NA          54     2     1    15  230   110     262      97
 2 36997 NA          49     3     1     9  110    76     281      78
 3 34659 NA          57     1     0     0  146    76     258      87
 4 36762 NA          58     2     1    60  150    97     250      65
 5 32197 NA          56     1     1     3  145   100     285      86
 6 33534 NA          45     3     1    23  115    76     263      78
 7 36763 NA          37     1     1    20  112    73.5   166      93
 8 33026 NA          48     1     1    25  102    66.5   304      66
 9 35364 NA          62     1     0     0  130    87     157      47
10 34177 NA          54     1     0     0  136.   83     254      95
# ℹ 2,391 more rows
```

Example: `read_xlsx()`

```r
# Read in the xlsx file, which we know is located on sheet 2 inside of the file
data <- read_xlsx("data/cvdstudy.xlsx", sheet = 2, skip = 10, col_names = TRUE)

# Base R approach to updating column data types (we'll learn more elegant solutions, soon!)
data$survdate <- as.Date(data$survdate)
data$edu <- factor(data$edu, levels = c("1", "2", "3", "4"))
data$smoke <- factor(data$smoke, levels = c("0", "1"))

# Look at the data
data
```

```
# A tibble: 2,401 × 10
   id     survdate      age edu   smoke  cigs   sbp   dbp  chol glucose
   <chr>  <date>      <dbl> <fct> <fct> <dbl> <dbl> <dbl> <dbl>   <dbl>
 1 32955  2019-11-04     54 2     1        15  230   110     262      97
 2 36997  2019-10-31     49 3     1         9  110    76     281      78
 3 34659  2019-07-14     57 1     0         0  146    76     258      87
 4 36762  2019-04-18     58 2     1        60  150    97     250      65
 5 32197  2019-02-20     56 1     1         3  145   100     285      86
 6 33534  2019-07-07     45 3     1        23  115    76     263      78
 7 36763  2019-01-04     37 1     1        20  112    73.5   166      93
 8 33026  2018-09-18     48 1     1        25  102    66.5   304      66
 9 35364  2019-06-13     62 1     0         0  130    87     157      47
10 34177  2019-03-19     54 1     0         0  136.   83     254      95
# ℹ 2,391 more rows
```

Much better. We've come so far!

haven

- Reading data formats used by other statistical packages
- Part of the `tidyverse` package ecosystem
  - Requires separate package installation
  - Imported data stored as **tibble data frame**
  - Informative error and warning messages
  - **Preserves label attributes** from other software
- **Core functions:**
  - `read_sas()`: SAS files (.sas7bdat, sas7bcat)
  - `read_sav()`: SPSS files (.sav)
  - `read_dta()`: Stata files (.dta)

```r
read_dta(
  file,
  encoding = NULL,
  col_select = NULL,
  skip = 0,
  n_max = Inf,
  .name_repair = "unique"
)
```

Example: `read_dta()`

```r
# Load the haven package
library(haven)

# Read in the stata (.dta) file
data <- read_dta("data/cvdstudy.dta")

# Use the base R approach to update data types
data$survdate <- as.Date(data$survdate)
data$edu <- factor(data$edu, levels = c("1", "2", "3", "4"))
data$smoke <- factor(data$smoke, levels = c("0", "1"))

# Look at the data
data
```

```
# A tibble: 2,401 × 10
   id    survdate     age edu   smoke  cigs   sbp   dbp  chol glucose
   <chr> <date>     <dbl> <fct> <fct> <dbl> <dbl> <dbl> <dbl>   <dbl>
 1 32955 2019-11-04    54 2     1        15   230   110     262      97
 2 36997 2019-10-31    49 3     1         9   110    76     281      78
 3 34659 2019-07-14    57 1     0         0   146    76     258      87
 4 36762 2019-04-18    58 2     1        60   150    97     250      65
 5 32197 2019-02-20    56 1     1         3   145   100     285      86
 6 33534 2019-07-07    45 3     1        23   115    76     263      78
 7 36763 2019-01-04    37 1     1        20   112    73.5   166      93
 8 33026 2018-09-18    48 1     1        25   102    66.5   304      66
 9 35364 2019-06-13    62 1     0         0   130    87     157      47
10 34177 2019-03-19    54 1     0         0   136.   83     254      95
# ℹ 2,391 more rows
```

- Note that Stata and other statistical software packages can include variable labels
- **Let's see if the data we just imported has these label attributes**

Example: `read_dta()`

```
1  # Look at the structure of the data object to which we assigned the imported dataset
2  str(data)
```

```
tibble [2,401 × 10] (S3: tbl_df/tbl/data.frame)
 $ id      : chr [1:2401] "32955" "36997" "34659" "36762" ...
  ..- attr(*, "label")= chr "Participant ID"
  ..- attr(*, "format.stata")= chr "%-9s"
 $ survdate: Date[1:2401], format: "2019-11-04" "2019-10-31" ...
 $ age     : num [1:2401] 54 49 57 58 56 45 37 48 62 54 ...
  ..- attr(*, "label")= chr "Age at the time of survey completion"
  ..- attr(*, "format.stata")= chr "%12.0g"
 $ edu     : Factor w/ 4 levels "1","2","3","4": 2 3 1 2 1 3 1 1 1 1 ...
 $ smoke   : Factor w/ 2 levels "0","1": 2 2 1 2 2 2 2 2 1 1 ...
 $ cigs    : num [1:2401] 15 9 0 60 3 23 20 25 0 0 ...
  ..- attr(*, "label")= chr "Number of cigarettes per day"
  ..- attr(*, "format.stata")= chr "%12.0g"
 $ sbp     : num [1:2401] 230 110 146 150 145 ...
  ..- attr(*, "label")= chr "Systolic blood pressure (mmHg)"
  ..- attr(*, "format.stata")= chr "%12.0g"
 $ dbp     : num [1:2401] 110 76 76 97 100 76 73.5 66.5 87 83 ...
  ..- attr(*, "label")= chr "Diastolic blood pressure (mmHg)"
  ..- attr(*, "format.stata")= chr "%12.0g"
 $ chol    : num [1:2401] 262 281 258 250 285 263 166 304 157 254 ...
  ..- attr(*, "label")= chr "Total cholesterol (mg/dL)"
  ..- attr(*, "format.stata")= chr "%12.0g"
 $ glucose : num [1:2401] 97 78 87 65 86 78 93 66 47 95 ...
  ..- attr(*, "label")= chr "Fasting glucose (mg/dL)"
  ..- attr(*, "format.stata")= chr "%12.0g"
```

id Participant ID	survdate Date participant completed the survey	age Age at the time of survey completion
32955	2019-11-04	54
36997	2019-10-31	49
34659	2019-07-14	57
36762	2019-04-18	58
32197	2019-02-20	56
33534	2019-07-07	45
36763	2019-01-04	37
33026	2018-09-18	48

- Yes! Our data do have variable labels!
- **Labels can be very helpful for providing further detail about variables in the dataset**

Case Studies

Sample R code and datasets

You can download the R script and datasets used for the two following examples via the following link:

https://github.com/dmarengi/reading_in_data

Either clone the repository or download it directly from github

Case study 1

Your non-technical colleague has reached out to you for assistance with preparing an old dataset for an upcoming project. Your colleague typically receives curated datasets and has not previously worked with data stored in a fixed width file format.

As an expert R programmer, you offer to help. Upon receiving the dataset, you read the data into R to inspect its contents and observe the following:

```r
# Load the readr package
library(readr)

read_fwf("data/newstudy.txt")
```

```
# A tibble: 534 × 2
         X1 X2
      <dbl> <chr>
 1 3230559 20 0244.0124.0264120
 2 3484062 11  242.0141.0206 94
 3 3568665 10 0235.0100.0240297
 4 3573556 1120200.0120.0217 72
 5 3351661 10 0200.0125.0265256
 6 3469763 30 0196.0102.0161 88
 7 3382758 10 0195.0 90.0294127
 8 3346766 30 0193.0132.0280 73
 9 3563665 2120192.5110.0212 71
10 3609445 2120189.0 87.0234 90
# ℹ 524 more rows
```

What additional information do you need from your colleague before you can provide support?

Case study 1 (cont.)

```r
# Vector of column widths
col_widths <- c(5, 2, 2, 1, 2, 5, 5, 3, 3)

# Vector of column names
col_names <- c("id", "age", "edu", "smoke", "cigs", "sbp", "dbp", "chol", "glucose")

# Read in the data, specifying column names and widths
data <- read_fwf("data/newstudy.txt", fwf_widths(widths = col_widths, col_names = col_names))
```

```
# A tibble: 534 × 9
      id   age   edu smoke  cigs   sbp   dbp  chol glucose
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
 1 32305    59     2     0     0   244   124   264     120
 2 34840    62     1     1    NA   242   141   206      94
 3 35686    65     1     0     0   235   100   240     297
 4 35735    56     1     1    20   200   120   217      72
 5 33516    61     1     0     0   200   125   265     256
 6 34697    63     3     0     0   196   102   161      88
 7 33827    58     1     0     0   195    90   294     127
 8 33467    66     3     0     0   193   132   280      73
 9 35636    65     2     1    20   192.  110   212      71
10 36094    45     2     1    20   189    87   234      90
# ℹ 524 more rows
```

Variable Name	Width	Description	Values/attributes
	5	5-digit participant ID number	min: 32006 max: 36998 *character*
	2	Age at the time of study enrollment	min: 33 max: 68 *integer*
	2	Highest level of education attainment	1 = less than high school 2 = high school 3 = some college 4 = college or higher *categorical*
oke	1	Smoking status	0 = no 1 = yes *dichotomous*
cigs	2	Number of cigarettes smoked per day	min: 0 max: 60 *integer*
sbp	5	Baseline systolic blood pressure (mmHg)	min: 90 max: 244 *numeric*
dbp	5	Baseline diastolic blood pressure (mmHg)	min: 57 max: 141 *numeric*
chol	3	Baseline total cholesterol, fasting (mg/dL)	min: 124 max: 432 *numeric*
glucose	3	Baseline glucose, fasting (mg/dL)	min: 40 max: 370

Case study 2

Your colleague asks for another favor. As part of their planned analysis, the research team hopes to incorporate exercise testing data obtained from a subset of study participants who had clinical assessments. Your colleague tells you that the data are stored in a formatted excel file that's exported directly from the exercise testing software, and their team has had trouble getting data into R to analyze. You offer to help, and your colleague provides you with a sample dataset to inspect. You observe the following:

```r
# Load the readxl package
library(readxl)

# Read in the xlsx file
data <- read_xlsx("data/exercise.xlsx")

# Look at the data
data
```

```
# A tibble: 695 × 11
   Participant `123456` ...3  ...4  ...5  ...6     ...7  ...8  ...9  ...10 ...11
   <chr>       <chr>    <chr> <chr> <chr> <chr>    <chr> <chr> <chr> <chr> <chr>
 1 Exercise    42776    <NA>  <NA>  <NA>  <NA>     <NA>  <NA>  <NA>  <NA>  <NA>
 2 <NA>        <NA>     <NA>  <NA>  <NA>  <NA>     <NA>  <NA>  <NA>  <NA>  <NA>
 3 Time        Work     Speed Grade HR    VO2      RR    PETC… SpO2  sysBP diaBP
 4 (min)       (Watts)  (MPH) (%)   (BPM) (mL/kg/… (br/… (mmH… (%)   (mmH… (mmH…
 5 00:19       0        0     0.2   89    3.1      11    36    97    <NA>  <NA>
 6 00:28       0        0     0.2   91    9.4      6     38    97    <NA>  <NA>
 7 00:30       0        0     0.2   98    3.1      11    36    98    <NA>  <NA>
 8 00:41       0        0     0.2   104   6.4      8     38    98    <NA>  <NA>
 9 00:44       0        0     0.2   106   6.4      9     39    98    <NA>  <NA>
10 00:50       0        0     0.2   113   7        10    39    98    <NA>  <NA>
# ℹ 685 more rows
```

What problems do you see, and how should we fix them?

Case study 2 (cont.)

- The data structure issue can be resolved fairly easily
  - Use the `skip` argument to skip the non-rectangular data at the top of the file
  - After rows are skipped, drop the first row of data containing units using `[-1,]`

```
1  data <- read_xlsx("data/exercise.xlsx", skip = 3)[-1,]
2
3  # Look at the data
4  data
```

```
# A tibble: 691 × 11
   Time  Work  Speed Grade HR    VO2   RR    PETCO2 SpO2  sysBP diaBP
   <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>  <chr> <chr> <chr>
 1 00:19 0     0     0.2   89    3.1   11    36     97    <NA>  <NA>
 2 00:28 0     0     0.2   91    9.4   6     38     97    <NA>  <NA>
 3 00:30 0     0     0.2   98    3.1   11    36     98    <NA>  <NA>
 4 00:41 0     0     0.2   104   6.4   8     38     98    <NA>  <NA>
 5 00:44 0     0     0.2   106   6.4   9     39     98    <NA>  <NA>
 6 00:50 0     0     0.2   113   7     10    39     98    <NA>  <NA>
 7 00:53 0     0     0.2   110   7.6   9     39     98    120   70
 8 00:57 0     0     0.2   109   8.6   9     39     98    120   70
 9 01:02 0     0     0.2   105   8.6   11    39     97    120   70
10 01:10 0     0     0.2   104   8.6   11    39     97    120   70
# ℹ 681 more rows
```

Case study 2 (cont.)

```r
# Coerce all but the time column to a numeric data type
data[2:11] <- sapply(data[2:11], as.numeric)

# Standardize the column names (all lower case)
colnames(data) <- tolower(colnames(data))

# Look at the data
data
```

```
# A tibble: 691 × 11
   time    work speed grade    hr   vo2    rr petco2  spo2 sysbp diabp
   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
 1 00:19     0     0   0.2    89   3.1    11     36    97    NA    NA
 2 00:28     0     0   0.2    91   9.4     6     38    97    NA    NA
 3 00:30     0     0   0.2    98   3.1    11     36    98    NA    NA
 4 00:41     0     0   0.2   104   6.4     8     38    98    NA    NA
 5 00:44     0     0   0.2   106   6.4     9     39    98    NA    NA
 6 00:50     0     0   0.2   113   7      10     39    98    NA    NA
 7 00:53     0     0   0.2   110   7.6     9     39    98   120    70
 8 00:57     0     0   0.2   109   8.6     9     39    98   120    70
 9 01:02     0     0   0.2   105   8.6    11     39    97   120    70
10 01:10     0     0   0.2   104   8.6    11     39    97   120    70
# ℹ 681 more rows
```

Key takeaways

- Reading data into R is typically the first step in a data analysis workflow
- In addition to base R functions, many R packages include functions with more robust functionality
  - Different functions have implications for how data are handled when read into R
    - For example, whether the data are stored as a standard data frame or tibble data frame
      - Useful packages include: `readr`, `readxl`, and `haven` (but there are many more!) e.g.,
        - `library(jsonlite)`: JSON files (common format for web-based data)
        - `library(xml2)`: XML files
- Idiosyncrasies in data storage methods warrant specific considerations. For example:
  - Reading xls and xlsx excel files require specific functions, which can accommodate file-specific formatting issues
    - Skipping rows before reading data
    - Selecting a specific range of cells or sheets to read data
- **Always be vigilant about identifying potential issues during the data import process**

Resources

- R for Data Science, 2nd edition by Hadley Wickham et. al
  - https://r4ds.hadley.nz/data-import
  - https://r4ds.hadley.nz/import
- The Epidemiologist R Handbook
  - https://www.epirhandbook.com/en/new_pages/importing.html