Data Import

Guy J. Abel

Basic Data

- Data frames are the most flexible and one of the most used object type in R.
- There are many R functions to load, manipulate and save data frames.
- In RStudio you can use the import data wizard.
  - `File | Import Dataset | From CSV ...`
  - `File | Import Dataset | From Excel ...`
  - `File | Import Dataset | From SPSS ...`
  - `File | Import Dataset | From SAS ...`
  - `File | Import Dataset | From Stata ...`
- Alternatively can use R code.
- Using R code
  - Quicker to run.
  - Easy to share and for others to replicate

Comma Separated Values

- Easiest way to import data is from files in the Comma Separated Values (CSV) format.
- A typical .csv file will look something like this.

```
"COUNTRY","YEAR","SAMPLE","SERIAL","GEOLEV1","GEOLEV2","PERNUM","PERWT","AGE","NATI
591,1960,591196001,1000,591004,591004003,1,20,53,1,1,110
591,1960,591196001,1000,591004,591004003,2,20,54,1,1,120
591,1960,591196001,1000,591004,591004003,3,20,31,1,1,120
591,1960,591196001,1000,591004,591004003,4,20,22,1,2,212
591,1960,591196001,1000,591004,591004003,5,20,20,1,2,212
591,1960,591196001,1000,591004,591004003,6,20,16,1,2,212
591,1960,591196001,1000,591004,591004003,7,20,13,1,2,212
591,1960,591196001,1000,591004,591004003,8,20,5,1,0,0
591,1960,591196001,1000,591004,591004003,9,20,3,1,0,0
591,1960,591196001,1000,591004,591004003,10,20,2,1,0,0
591,1960,591196001,2000,591004,591004003,1,20,42,1,1,110
591,1960,591196001,3000,591004,591004003,1,20,58,1,1,110
591,1960,591196001,3000,591004,591004003,2,20,82,1,1,110
...
```

- CSV files can be viewed in Excel with commas removed
- Can convert a single Excel spreadsheet as a CSV file using the Save As option.

Comma Separated Values

- The two most common ways to read CSV files into R are using:
  - `read.csv()` in the `base` package
  - `read_csv()` in the `readr` package.
- The `readr` package is part of the tidyverse set of packages (more on the tidyverse later)
- When data is read in they are different types of R objects
  - `data.frame` from `read.csv()`
  - `tbl_df` (tibble) from `read_csv()`
- They display differently when printed to the R console. . .

Comma Separated Values

- Demonstrate loading data using R code with IPUMSI data
  - http://international.ipums.org/international
  - A large data base containing an census micro-data from around the world.
  - Can download CSV files (as well as Stata, SPSS and SAS)
  - Free. Registration required.
- The `file.show()` function opens files in their default program

```
> file.show("./data/ipumsi_pan1960.csv")
```

Comma Separated Values

```
df0 <- read.csv(file = "./data/ipumsi_pan1960.csv")
df0
>       COUNTRY YEAR    SAMPLE   SERIAL GEOLEV1   GEOLEV2 PERNUM PERWT AGE NATIVITY
> 1         591 1960 591196001     1000  591004 591004003      1    20  53        1
> 2         591 1960 591196001     1000  591004 591004003      2    20  54        1
> 3         591 1960 591196001     1000  591004 591004003      3    20  31        1
> 4         591 1960 591196001     1000  591004 591004003      4    20  22        1
> 5         591 1960 591196001     1000  591004 591004003      5    20  20        1
> 6         591 1960 591196001     1000  591004 591004003      6    20  16        1
> 7         591 1960 591196001     1000  591004 591004003      7    20  13        1
> 8         591 1960 591196001     1000  591004 591004003      8    20   5        1
> 9         591 1960 591196001     1000  591004 591004003      9    20   3        1
> 10        591 1960 591196001     1000  591004 591004003     10    20   2        1
> 11        591 1960 591196001     2000  591004 591004003      1    20  42        1
> 12        591 1960 591196001     3000  591004 591004003      1    20  58        1
> 13        591 1960 591196001     3000  591004 591004003      2    20  82        1
> 14        591 1960 591196001     3000  591004 591004003      3    20  57        1
> 15        591 1960 591196001     4000  591004 591004003      1    20  62        1
> 16        591 1960 591196001     4000  591004 591004003      2    20  26        1
> 17        591 1960 591196001     5000  591004 591004003      1    20  42        1
> 18        591 1960 591196001     5000  591004 591004003      2    20  35        1
> 19        591 1960 591196001     5000  591004 591004003      3    20  17        1
> 20        591 1960 591196001     5000  591004 591004003      4    20  15        1
> 21        591 1960 591196001     5000  591004 591004003      5    20   8        1
```

Tibbles

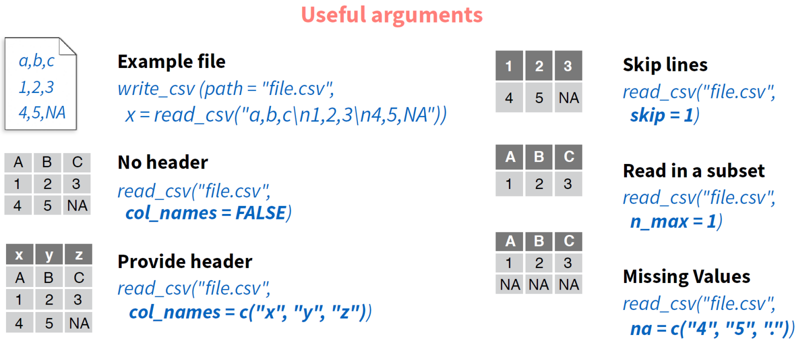
```
> library(readr)
> df1 <- read_csv(file = "./data/ipumsi_pan1960.csv")
> df1
# A tibble: 53,553 x 12
   COUNTRY  YEAR    SAMPLE SERIAL GEOLEV1   GEOLEV2 PERNUM PERWT   AGE
     <int> <int>     <int>  <int>   <int>     <int>  <int> <int> <int>
 1     591  1960 591196001   1000  591004 591004003      1    20    53
 2     591  1960 591196001   1000  591004 591004003      2    20    54
 3     591  1960 591196001   1000  591004 591004003      3    20    31
 4     591  1960 591196001   1000  591004 591004003      4    20    22
 5     591  1960 591196001   1000  591004 591004003      5    20    20
 6     591  1960 591196001   1000  591004 591004003      6    20    16
 7     591  1960 591196001   1000  591004 591004003      7    20    13
 8     591  1960 591196001   1000  591004 591004003      8    20     5
 9     591  1960 591196001   1000  591004 591004003      9    20     3
10     591  1960 591196001   1000  591004 591004003     10    20     2
# ... with 53,543 more rows, and 3 more variables: NATIVITY <int>,
#   EDATTAIN <int>, EDATTAIND <int>
```

Tibbles

- When you print `data.frames` you get everything.
  - If you are dealing with non-small data sets this is annoying
  - If you are dealing with very large data sets the printing can take a long time.
- A `tbl_df` is an improved `data.frame` with nice methods for high-level inspection.
  - By default `tbl_df` will print only the first 10 rows for large data sets
  - Will subdue extra columns that won't fit into your console
  - Provides the column type in a three letter abbreviations under the column names
  - Dimension information at the top.

Comma Separated Values

- The `read_csv()` function can be much faster and avoids converting to factors (more on factors later).



Useful arguments

```
a,b,c
1,2,3
4,5,NA
```

Example file

```
write_csv (path = "file.csv",
 x = read_csv("a,b,c\n1,2,3\n4,5,NA"))
```

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"))
```

1	2	3
4	5	NA

Skip lines

```
read_csv("file.csv",
 skip = 1)
```

A	B	C
1	2	3

Read in a subset

```
read_csv("file.csv",
 n_max = 1)
```

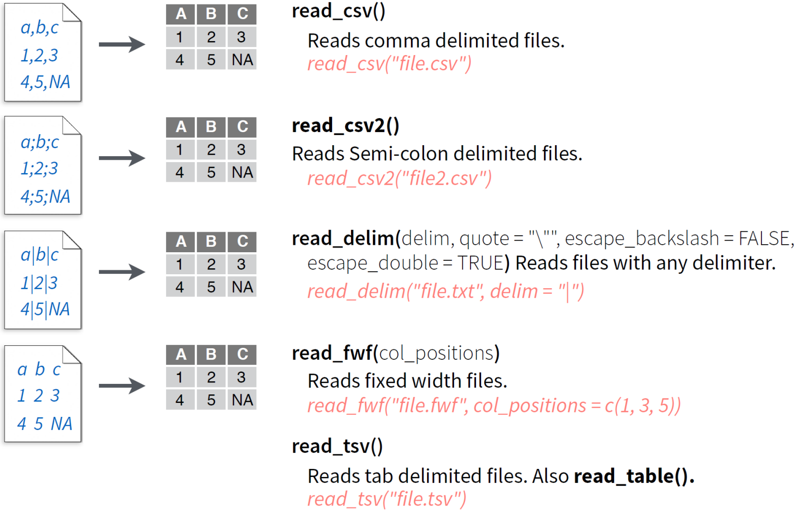
A	B	C
1	2	3
NA	NA	NA

Missing Values

```
read_csv("file.csv",
 na = c("4", "5", "."))
```

Delimited Files

- The `read_csv()` function is a special case of `read_delim()`
- Different people and/or countries use different formats to separate values



a,b,c
1,2,3
4,5,NA

A	B	C
1	2	3
4	5	NA

read_csv()

Reads comma delimited files.
read_csv("file.csv")

a;b;c
1;2;3
4;5;NA

A	B	C
1	2	3
4	5	NA

read_csv2()

Reads Semi-colon delimited files.
read_csv2("file2.csv")

a|b|c
1|2|3
4|5|NA

A	B	C
1	2	3
4	5	NA

read_delim(delim, quote = "\"", escape_backslash = FALSE, escape_double = TRUE**) Reads files with any delimiter.**
read_delim("file.txt", delim = "|")

a b c
1 2 3
4 5 NA

A	B	C
1	2	3
4	5	NA

read_fwf(col_positions**)**

Reads fixed width files.
read_fwf("file.fwf", col_positions = c(1, 3, 5))

read_tsv()

Reads tab delimited files. Also **read_table().**
read_tsv("file.tsv")

Saving Data

- The two most common ways to write (save) CSV files into R are using:
  - `write.csv()` in the `base` package
  - `write_csv()` in the `readr` package.
- `write.csv()` will add a first column with the row name (usually a number)
- `write_csv()` saves just the data frame by default

```
> write_csv(x = df0, path = "./data/mynewfile.csv")
```

- `write_excel_csv()` works well with Chinese characters (can get lost with `write_csv()`)
- CSV files are the most common external data format to used with R.
- Users tend to save data as CSV, even if imported from a different format as they are small and simple.

Exercise 1

- Open ex31.R and complete the following exercises. Once you have filled in, save the exercise file as "ex31.R".

```
# 0.  Clear your workspace and set your working directory to your data folder in the
rm(list = ls())
setwd(dir = "C:/Users/Guy/Dropbox/APPI2017/exercise/data/")
##
##
##
# 1.  Load the readr package

# 2.  Open the "2010_Census_Populations_by_Zip_Code.csv" file using the file.show()

# 3.  Use read_csv to read the data into R and call the results d1


# 4.  Open the "unhcr_popstats_export_persons_of_concern.csv" file using the file.sh

# 5.  a) Q: How many lines should we skip when reading the data?
#          A: 3
#     b) Q: Are their any missing values (if so how are they represented)?
#          A: *
# 6.  Use read_csv to read the data into R and call the results d2
```

Data Exploration

There are many R functions to explore your data.

Function	Description
`head()`	First rows of the data frame
`tail()`	Last rows of the data frame
`str()`	Structure of data frame
`summary()`	Summary of each column of the data frame
`dim()`	Dimensions of the data frame
`nrow()`	Number of rows in the data frame
`ncol()`	Number of columns in the data frame
`rownames()`	Row names in the data frame
`colnames()`	Column names in the data frame
`dimnames()`	Row and column names in the data frame
`View()`	Invoke a Data Viewer

Data Exploration

```
> head(df0)
  COUNTRY YEAR    SAMPLE SERIAL GEOLEV1   GEOLEV2 PERNUM PERWT AGE
1     591 1960 591196001   1000  591004 591004003      1    20  53
2     591 1960 591196001   1000  591004 591004003      2    20  54
3     591 1960 591196001   1000  591004 591004003      3    20  31
4     591 1960 591196001   1000  591004 591004003      4    20  22
5     591 1960 591196001   1000  591004 591004003      5    20  20
6     591 1960 591196001   1000  591004 591004003      6    20  16
  NATIVITY EDATTAIN EDATTAIND
1        1        1       110
2        1        1       120
3        1        1       120
4        1        2       212
5        1        2       212
6        1        2       212
```

Data Exploration

```
> str(df0)
'data.frame':   53553 obs. of  12 variables:
 $ COUNTRY  : int  591 591 591 591 591 591 591 591 591 591 ...
 $ YEAR     : int  1960 1960 1960 1960 1960 1960 1960 1960 1960 1960 ...
 $ SAMPLE   : int  591196001 591196001 591196001 591196001 591196001 591196001 5911
 $ SERIAL   : int  1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 ...
 $ GEOLEV1  : int  591004 591004 591004 591004 591004 591004 591004 591004 591004 5
 $ GEOLEV2  : int  591004003 591004003 591004003 591004003 591004003 591004003 5910
 $ PERNUM   : int  1 2 3 4 5 6 7 8 9 10 ...
 $ PERWT    : int  20 20 20 20 20 20 20 20 20 20 ...
 $ AGE      : int  53 54 31 22 20 16 13 5 3 2 ...
 $ NATIVITY : int  1 1 1 1 1 1 1 1 1 1 ...
 $ EDATTAIN : int  1 1 1 2 2 2 2 0 0 0 ...
 $ EDATTAIND: int  110 120 120 212 212 212 212 0 0 0 ...
```

Data Exploration

```
> summary(df0)
    COUNTRY         YEAR          SAMPLE              SERIAL
 Min.   :591   Min.   :1960   Min.   :591196001   Min.   :    1000
 1st Qu.:591   1st Qu.:1960   1st Qu.:591196001   1st Qu.: 2883000
 Median :591   Median :1960   Median :591196001   Median : 6135000
 Mean   :591   Mean   :1960   Mean   :591196001   Mean   : 5974039
 3rd Qu.:591   3rd Qu.:1960   3rd Qu.:591196001   3rd Qu.: 9063000
 Max.   :591   Max.   :1960   Max.   :591196001   Max.   :11869000
    GEOLEV1          GEOLEV2             PERNUM          PERWT
 Min.   :591002   Min.   :5.91e+08   Min.   : 1.000   Min.   :20
 1st Qu.:591004   1st Qu.:5.91e+08   1st Qu.: 2.000   1st Qu.:20
 Median :591004   Median :5.91e+08   Median : 3.000   Median :20
 Mean   :591006   Mean   :5.91e+08   Mean   : 3.715   Mean   :20
 3rd Qu.:591008   3rd Qu.:5.91e+08   3rd Qu.: 5.000   3rd Qu.:20
 Max.   :591008   Max.   :5.91e+08   Max.   :28.000   Max.   :20
      AGE          NATIVITY       EDATTAIN       EDATTAIND
 Min.   : 0.0   Min.   :1.00   Min.   :0.000   Min.   :  0.0
 1st Qu.: 7.0   1st Qu.:1.00   1st Qu.:0.000   1st Qu.:  0.0
 Median :18.0   Median :1.00   Median :1.000   Median :120.0
 Mean   :22.7   Mean   :1.08   Mean   :1.016   Mean   :112.9
 3rd Qu.:35.0   3rd Qu.:1.00   3rd Qu.:1.000   3rd Qu.:120.0
 Max.   :98.0   Max.   :9.00   Max.   :9.000   Max.   :999.0
```

Data Exploration

- In RStudio can also use the `View` function or click on the data frame in the Environment tab to (initially) view the first 1000 rows.
- Can filter and sort columns within the data view.

```
> View(df0)
```

Factors

- When dealing with categorical data in R we often use `factors`
  - Special vectors that are character strings with an additional level attribute.
  - The level attribute provides further information on the order of the categorical data
- Older importing functions, such as `read.csv()`, will create factors by default when bringing data into R. (`stringsAsFactors = TRUE`)
  - Data importation functions in the tidyverse, such as `read_csv`, do not create factors by default
  - This is partly why they are faster at importing.
- Creating a factors can be done easily with the `factor()` function.
  - Levels follows alpha-numeric order by default.
- Rearranging the orders of the factors with the can be difficult. The `forcats` package has lots of helpful functions help work with factors and categorical data.
  - `fct_recode()` change the factor names
  - `fct_relevel()` reorder factors by hand
  - `fct_reorder()` reorder factors by sorting on a variable
  - `fct_inorder()` order factors by their appearance

Factors

```
> # original numeric data
> table(df1$NATIVITY)

    1     2     9
49399  4132    22
> # covert to factor
> df1$NATIVITY <- factor(df1$NATIVITY)
> table(df1$NATIVITY)

    1     2     9
49399  4132    22
> # rename the levels with character strings
> library(forcats)
> df1$NATIVITY <- fct_recode(df1$NATIVITY, "native" = "1", "foreign" = "2", "missin
> table(df1$NATIVITY)

 native foreign missing
  49399    4132      22
> # reorder factors
> df1$NATIVITY <- fct_relevel(df1$NATIVITY, "foreign", "missing", "native")
> table(df1$NATIVITY)

foreign missing  native
   4132      22   49399
```

Data Modes

- R can easily change the mode and object types of columns in data frames.

Function	Description
`as.numeric()`	creates a numeric vector
`as.character()`	creates a character vector
`as.integer()`	creates an integer vector
`as.factor()`	creates a factor vector
`as.data.frame()`	will turn vectors, matrices and lists into data frames

Exercise 2 (ex32.R)

```
# 0.  a) Check your working directory is in the course folder

#     b) Load the data and packages by sourcing the solution file for ex31.R
#####("./exercise-solutions/ex31.R")
##
##
##
# 1. What are the dimensions of d1 data frame object

# 2. Show a summary of each column of d1

# 3. Show the first three rows of d1
#    (Hint: Use head() function. See ?head to set the number of rows)

# 4. Show the column names of d1

# 5. Covert the `Year` column in d2 from integers to a character string

# 6. Convert the `Origin` column in d2 from a character string to a factor

# 7. Show the levels of Origin column

# 8. Convert the name column in d5 to a factor

# 9. Show the levels of name column
```

Excel Data

- There is no functions in base R to read excel files.
- The `readxl` package has a `read_excel()` function.
- In `read_excel()` you can specify the `sheet` (either a name or number).
- To demonstrate we will use the `SAPE18DT14.xls` spreadsheet from the UK ONS on population estimates by age group in England and Wales output areas.

```
file.show("./data/SAPE18DT14.xls")
```

- Similar options to `read_csv()` (`na =`, `col_names =`, `skip =`)
- The `guess_max =` argument can very useful for when dealing with long data frames
    - By default `guess_max = 1000`, i.e. a guess of the data type will be based on the first 1000 rows.
    - Sometimes the guess might be wrong and throw a big red warning message - set `guess_max` to a higher value - at the cost of speed

Excel Data

```
> library(readxl)
> df2 <- read_excel(path = "./data/SAPE18DT14.xls", sheet = 2, skip = 3)
> df2
# A tibble: 7,549 x 23
   `Area Codes`  `Area Names`              X__1 `All ages` `0-4` `5-9`
          <chr>         <chr>             <chr>      <dbl> <dbl> <dbl>
 1    E06000047 County Durham              <NA>     519695 28446 28859
 2    E02004297          <NA> County Durham 001       7912   455   421
 3    E02004290          <NA> County Durham 002       5851   251   313
 4    E02004298          <NA> County Durham 003       9858   488   524
 5    E02004299          <NA> County Durham 004       8588   555   506
 6    E02004291          <NA> County Durham 005       6957   427   376
 7    E02004300          <NA> County Durham 006       7840   496   506
 8    E02004292          <NA> County Durham 007       7845   350   444
 9    E02004301          <NA> County Durham 008       9205   653   666
10    E02004302          <NA> County Durham 009       7766   508   469
# ... with 7,539 more rows, and 17 more variables: `10-14` <dbl>,
#   `15-19` <dbl>, `20-24` <dbl>, `25-29` <dbl>, `30-34` <dbl>,
#   `35-39` <dbl>, `40-44` <dbl>, `45-49` <dbl>, `50-54` <dbl>,
#   `55-59` <dbl>, `60-64` <dbl>, `65-69` <dbl>, `70-74` <dbl>,
#   `75-79` <dbl>, `80-84` <dbl>, `85-89` <dbl>, `90+` <dbl>
```

Excel Data

- Saving R output as Excel file is not too easy. . . use CSV if you can.
- There are a number of older packages e.g. `xlsx` and `XLConnect` that are especially tricky.
  - Not to easy to install as require correct Java version.
  - Do have alternative options to read data from Excel.
- The `openxlsx` package does not require Java.
  - Must first define a workbook and sheet. . .

Excel Data

```
> # install.packages("openxlsx")
> library(openxlsx)
> # create a empty workbook to fill
> wb0 <- createWorkbook(creator = "Guy")
> # create a empty sheet in the workbook
> addWorksheet(wb = wb0, sheetName = "small area population")
> # add your data
> writeData(wb = wb0, sheet = 1, x = df2)
> # add a filter and freeze the top row
> addFilter(wb = wb0, sheet = 1, rows = 1, cols = names(df2))
> freezePane(wb = wb0, sheet = 1, firstRow = TRUE)
>
> ## Save workbook to working directory
> saveWorkbook(wb0, file = "./data/xlexample.xlsx", overwrite = TRUE)
> file.show("./data/xlexample.xlsx")
```

Exercise 3 (ex33.R)

```
# 0.  Clear your workspace and set your working directory to your data folder in the


##
##
##
# 1. load the readxl package

# 2. Use read_excel to read data on Male population totals in SAPE18DT14.xls into R


# 3. Use read_excel to read data on ESTIMATES in WPP2015_FERT_F04_TOTAL_FERTILITY.xl


# 4. Use read_excel to read data on MEDIUM VARIANT in WPP2015_POP_F07_1_POPULATION_B
```

SPSS Data

- There are number of packages to import data from SPSS, Stata and SAS files.
- The two most popular are the `haven` and `foriegn` packages
- Each extends R `read()` and `write()` functions for different file types.
- The `haven` package is part of the tidyverse. It is bit newer, bit faster, outputs tibbles!

SPSS Data

```
> library(haven)
> df3 <- read_sav(file = "./data/ipumsi_pan1960.sav")
> df3
# A tibble: 53,553 x 12
     COUNTRY      YEAR    SAMPLE SERIAL GEOLEV1   GEOLEV2 PERNUM PERWT
   <dbl+lbl> <dbl+lbl> <dbl+lbl>  <dbl>   <dbl>     <dbl>  <dbl> <dbl>
 1       591      1960 591196001   1000  591004 591004003      1    20
 2       591      1960 591196001   1000  591004 591004003      2    20
 3       591      1960 591196001   1000  591004 591004003      3    20
 4       591      1960 591196001   1000  591004 591004003      4    20
 5       591      1960 591196001   1000  591004 591004003      5    20
 6       591      1960 591196001   1000  591004 591004003      6    20
 7       591      1960 591196001   1000  591004 591004003      7    20
 8       591      1960 591196001   1000  591004 591004003      8    20
 9       591      1960 591196001   1000  591004 591004003      9    20
10       591      1960 591196001   1000  591004 591004003     10    20
# ... with 53,543 more rows, and 4 more variables: AGE <dbl+lbl>,
#   NATIVITY <dbl+lbl>, EDATTAIN <dbl+lbl>, EDATTAIND <dbl+lbl>
```

SPSS Data

- The labels for each data code do not display from `read_sav`
- However they are known to R.

```
> print_labels(df3$NATIVITY)

Labels:
 value                label
     0 NIU (not universe)
     1          Native-born
     2         Foreign-born
     9      Unknown/missing
```

SPSS Data

- Can change column to a characters using `as_factor()` function in the haven package

```
> # single column
> table(df3$NATIVITY)

    1     2     9
49399  4132    22
> df3$NATIVITY <- as_factor(df3$NATIVITY, "labels")
> table(df3$NATIVITY)

NIU (not universe)          Native-born        Foreign-born
                 0                49399                4132
   Unknown/missing
                22
> # all columns
> head(df3)
# A tibble: 6 x 12
      COUNTRY        YEAR     SAMPLE SERIAL GEOLEV1   GEOLEV2 PERNUM PERWT
  <dbl+lbl> <dbl+lbl> <dbl+lbl>  <dbl>   <dbl>     <dbl>  <dbl> <dbl>
1         591        1960  591196001   1000  591004 591004003      1    20
2         591        1960  591196001   1000  591004 591004003      2    20
3         591        1960  591196001   1000  591004 591004003      3    20
4         591        1960  591196001   1000  591004 591004003      4    20
```

SPSS Data

- The foreign package has limited support for newer SPSS formats (since 2000).
- For SPSS files will create a list by default (to.data.frame = FALSE)

```
> # install.packages("foreign")
> library(foreign)
> df4 <- read.spss(file = "./data/ipumsi_pan1960.sav", use.value.labels = FALSE, to
re-encoding from UTF-8
> head(df4, n = 2)
  COUNTRY YEAR    SAMPLE SERIAL GEOLEV1   GEOLEV2 PERNUM PERWT AGE
1     591 1960 591196001   1000  591004 591004003      1    20  53
2     591 1960 591196001   1000  591004 591004003      2    20  54
  NATIVITY EDATTAIN EDATTAIND
1        1        1       110
2        1        1       120
> df4 <- read.spss(file = "./data/ipumsi_pan1960.sav", to.data.frame = TRUE)
re-encoding from UTF-8
> head(df4, n = 2)
  COUNTRY YEAR    SAMPLE SERIAL GEOLEV1   GEOLEV2 PERNUM PERWT AGE
1  Panama 1960 Panama 1960   1000  591004 591004003      1    20  53
2  Panama 1960 Panama 1960   1000  591004 591004003      2    20  54
     NATIVITY                     EDATTAIN              EDATTAIND
1 Native-born Less than primary completed           No schooling
2 Native-born Less than primary completed Some primary completed
```

Stata Data

- The haven package works well for Stata files of all ages

```
> # haven
> df5 <- read_dta("./data/ipumsi_pan1960.dta")
> # gives a tibble
> head(df5, n = 3)
# A tibble: 3 x 12
    country      year    sample serial geolev1   geolev2 pernum perwt
  <dbl+lbl> <dbl+lbl> <dbl+lbl>  <dbl>   <dbl>     <dbl>  <dbl> <dbl>
1       591      1960 591196001   1000  591004 591004003      1    20
2       591      1960 591196001   1000  591004 591004003      2    20
3       591      1960 591196001   1000  591004 591004003      3    20
# ... with 4 more variables: age <dbl+lbl>, nativity <dbl+lbl>,
#   edattain <dbl+lbl>, edattaind <dbl+lbl>
> df5 <- as_factor(df5)
> head(df5, n = 3)
# A tibble: 3 x 12
  country   year      sample serial geolev1   geolev2 pernum perwt    age
   <fctr> <fctr>      <fctr>  <dbl>   <dbl>     <dbl>  <dbl> <dbl> <fctr>
1  panama   1960 panama 1960   1000  591004 591004003      1    20     53
2  panama   1960 panama 1960   1000  591004 591004003      2    20     54
3  panama   1960 panama 1960   1000  591004 591004003      3    20     31
# ... with 3 more variables: nativity <fctr>, edattain <fctr>,
#   edattaind <fctr>
```

Stata Data

- For Stata $>$ v12 `read.dta` in the foreign package will not work.

```
> # foreign... does not work
> df6 <- read.dta("./data/ipumsi_pan1960.dta")
> `Error in read.dta("./data/ipumsi_pan1960.dta") :
+   unable to open file: 'No such file or directory'`
```

Stata Data

- The readstata13 package is a good alternative.
  - By default will use labels (generate.factors = TRUE)

```
> # install.packages("readstata13")
> library(readstata13)
> df6 <- read.dta13("./data/ipumsi_pan1960.dta", generate.factors = FALSE)
> head(df6, n = 3)
  country year      sample serial geolev1   geolev2 pernum perwt age
1  panama 1960 panama 1960   1000  591004 591004003      1    20  53
2  panama 1960 panama 1960   1000  591004 591004003      2    20  54
3  panama 1960 panama 1960   1000  591004 591004003      3    20  31
     nativity                    edattain              edattaind
1 native-born less than primary completed           no schooling
2 native-born less than primary completed some primary completed
3 native-born less than primary completed some primary completed
> df6 <- read.dta13("./data/ipumsi_pan1960.dta")
> head(df6, n = 3)
  country year      sample serial geolev1   geolev2 pernum perwt age
1  panama 1960 panama 1960   1000  591004 591004003      1    20  53
2  panama 1960 panama 1960   1000  591004 591004003      2    20  54
3  panama 1960 panama 1960   1000  591004 591004003      3    20  31
     nativity                    edattain              edattaind
1 native-born less than primary completed           no schooling
2 native-born less than primary completed some primary completed
```

SAS data

- SAS files saved as .sas7bdat can work with haven `read_sas`
- More difficult in `foreign` package `read.ssd` (not shown).

```
> # haven
> df7 <- read_sas("./data/ipumsi_pan1960.sas7bdat")
> # gives a tibble
> head(df7)
# A tibble: 6 x 12
  COUNTRY  YEAR    SAMPLE SERIAL GEOLEV1   GEOLEV2 PERNUM PERWT   AGE
    <dbl> <dbl>     <dbl>  <dbl>   <dbl>     <dbl>  <dbl> <dbl> <dbl>
1     591  1960 591196001   1000  591004 591004003      1    20    53
2     591  1960 591196001   1000  591004 591004003      2    20    54
3     591  1960 591196001   1000  591004 591004003      3    20    31
4     591  1960 591196001   1000  591004 591004003      4    20    22
5     591  1960 591196001   1000  591004 591004003      5    20    20
6     591  1960 591196001   1000  591004 591004003      6    20    16
# ... with 3 more variables: NATIVITY <dbl>, EDATTAIN <dbl>,
#   EDATTAIND <dbl>
```

Importing Data Summary

- Recommend `tidyverse` versions first. If not working try alternatives.

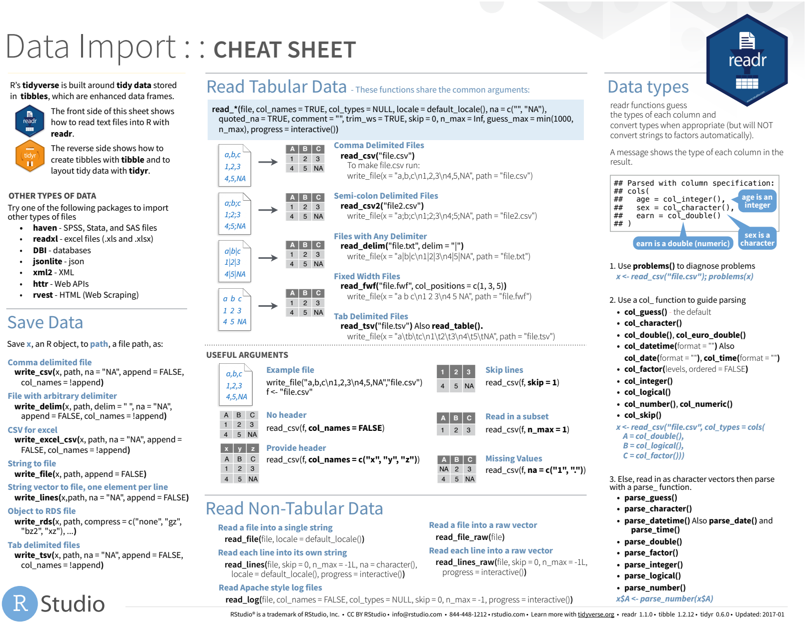
Function	Package	tidyverse	Description
`read.csv()`	base	No	CSV files
`read_csv()`	readr	Yes	CSV files
`read_excel()`	readxl	Yes	Excel files
`read.spss()`	foreign	No	SPSS files
`read_sav()`	haven	Yes	SPSS files
`read.dta()`	foreign	No	Stata files
`read.dta13()`	readstata13	No	Stata files
`read_dta()`	haven	Yes	Stata files
`read.ssd()`	foreign	No	SAS files
`read_sas()`	haven	Yes	SAS files

Exporting Data

- CSV are simple and usually preferred.

Function	Package	tidyverse	Description
`write.csv()`	base	No	CSV files
`write_csv()`	readr	Yes	CSV files
`saveWorkbook()`	openxlsx	No	Excel files.
`write.foreign()`	foreign	No	SPSS, Stata, SAS files
`write_sav()`	haven	Yes	SPSS files
`save.dta13()`	readstata13	No	Stata files
`write_dta()`	haven	Yes	Stata files
`write_sas()`	haven	Yes	SAS files

RStudio Data Import Cheatsheet



Data Import : : CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

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

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save **x**, an R object, to **path**, a file path, as:

Comma delimited file
write_csv(x, path, na = "NA", append = FALSE, col_names = !append)

File with arbitrary delimiter
write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append)

CSV for excel
write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append)

String to file
write_file(x, path, append = FALSE)

String vector to file, one element per line
write_lines(x,path, na = "NA", append = FALSE)

Object to RDS file
write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...)

Tab delimited files
write_tsv(x, path, na = "NA", append = FALSE, col_names = !append)

Read Tabular Data - These functions share the common arguments:

```
read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"),
  quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,
  n_max), progress = interactive())
```

a,b,c
1,2,3
4,5,NA
→

A	B	C
1	2	3
4	5	NA

Comma Delimited Files
read_csv("file.csv")
To make file.csv run:
write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")

a;b;c
1;2;3
4;5;NA
→

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files
read_csv2("file2.csv")
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")

a|b|c
1|2|3
4|5|NA
→

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter
read_delim("file.txt", delim = "|")
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

a b c
1 2 3
4 5 NA
→

A	B	C
1	2	3
4	5	NA

Fixed Width Files
read_fwf("file.fwf", col_positions = c(1, 3, 5))
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

Tab Delimited Files
read_tsv("file.tsv") Also **read_table()**.
write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")

USEFUL ARGUMENTS

a,b,c
1,2,3
4,5,NA

Example file
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")
f <- "file.csv"

1	2	3
4	5	NA

Skip lines
read_csv(f, **skip = 1**)

A	B	C
1	2	3
4	5	NA

No header
read_csv(f, **col_names = FALSE**)

A	B	C
1	2	3

Read in a subset
read_csv(f, **n_max = 1**)

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

Provide header
read_csv(f, **col_names = c("x", "y", "z")**)

A	B	C
NA	2	3
4	5	NA

Missing Values
read_csv(f, **na = c("1", ".")**)

Read Non-Tabular Data

Read a file into a single string
read_file(file, locale = default_locale())

Read each line into its own string
read_lines(file, skip = 0, n_max = -1L, na = character(), locale = default_locale(), progress = interactive())

Read Apache style log files
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())

Read a file into a raw vector
read_file_raw(file)

Read each line into a raw vector
read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive())

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer

earn is a double (numeric)

sex is a character

1. Use **problems()** to diagnose problems
x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing
- **col_guess()** - the default
- **col_character()**
- **col_double()**, **col_euro_double()**
- **col_datetime**(format = "") Also **col_date**(format = ""), **col_time**(format = "")
- **col_factor**(levels, ordered = FALSE)
- **col_integer()**
- **col_logical()**
- **col_number()**, **col_numeric()**
- **col_skip()**

*x <- read_csv("file.csv", col_types = cols(
  A = col_double(),
  B = col_logical(),
  C = col_factor()))*

3. Else, read in as character vectors then parse with a parse_ function.
- **parse_guess()**
- **parse_character()**
- **parse_datetime()** Also **parse_date()** and **parse_time()**
- **parse_double()**
- **parse_factor()**
- **parse_integer()**
- **parse_logical()**
- **parse_number()**

x$A <- parse_number(x$A)

RStudio

Optional Assignment 3 (assign3.R)

```
##
## Assignment 3
##

# 1. Use an internet searh to find some demographic data by region from one country

# 2. Find a link to download the data in any of the formats covered in the class
#    (delimited, Excel, Stata, SPSS, Stata or SAS).
#    Make sure the file is not too large (less than 1MB)
#
#    If there is no file to download or it is too large go back to step 1 with new s

# 3. Save your file in the data folder you have been working on in class.

# 4. Complete the following information below on where you found your data
#    e.g. Data Agency       : United Nations Population Division
#         Data Title        : Total fertility (TFR)
#         Original File Name: WPP2015_FERT_F04_TOTAL_FERTILITY.xls
#         File Type         : Excel
#         URL               : https://esa.un.org/unpd/wpp/Download/Standard/Fertilit
#
# Data Agency       :
# Data Title        :
# Original File Name:
# File Type         :
```