Handling Data - Data Importing

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 `readr` package.
  - `read.csv()` in the `base` package
- The `readr` package is part of the tidyverse set of packages (more on the tidyverse later)
  - `library(tidyverse)` loads `readr` along with other useful packages (e.g. `ggplot2`)
- When data is read in they are different types of R objects
  - `tibble` from `read_csv()`
  - `data.frame` from `read.csv()`
- They display differently when printed to the R console (see slide after next)

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("../slides-data/ipumsi_pan1960.csv")
```

Comma Separated Values

```
df0 <- read.csv(file = "../slides-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
> 22        591 1960 591196001     5000  591004 591004003      6    20   6        1
```

Tibbles

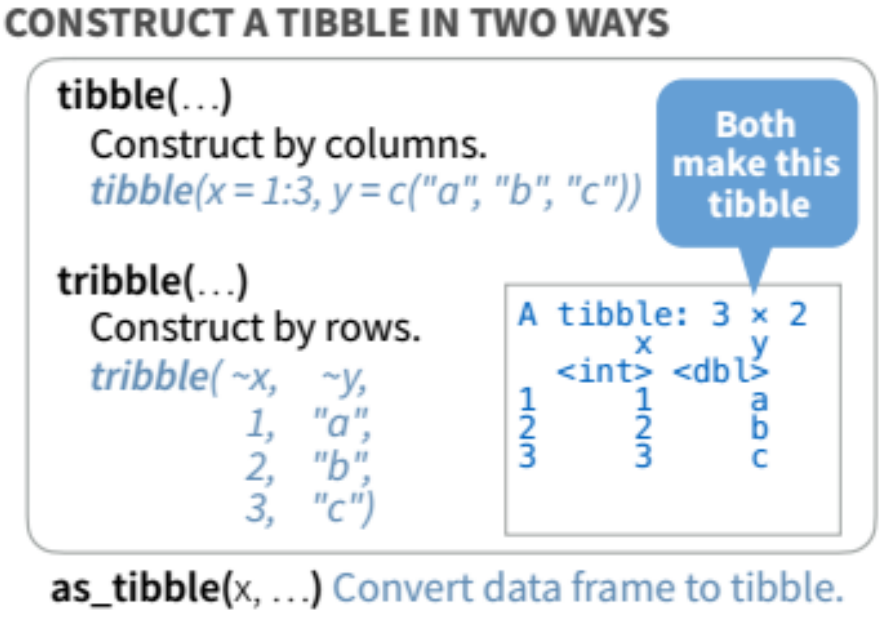
```
> library(tidyverse)
> df1 <- read_csv(file = "../slides-data/ipumsi_pan1960.csv")
> df1
# A tibble: 53,553 x 12
   COUNTRY  YEAR SAMPLE SERIAL GEOLEV1 GEOLEV2 PERNUM PERWT   AGE NATIVITY
     <dbl> <dbl>  <dbl>  <dbl>   <dbl>   <dbl>  <dbl> <dbl> <dbl>    <dbl>
 1     591  1960 5.91e8   1000  591004  5.91e8      1    20    53        1
 2     591  1960 5.91e8   1000  591004  5.91e8      2    20    54        1
 3     591  1960 5.91e8   1000  591004  5.91e8      3    20    31        1
 4     591  1960 5.91e8   1000  591004  5.91e8      4    20    22        1
 5     591  1960 5.91e8   1000  591004  5.91e8      5    20    20        1
 6     591  1960 5.91e8   1000  591004  5.91e8      6    20    16        1
 7     591  1960 5.91e8   1000  591004  5.91e8      7    20    13        1
 8     591  1960 5.91e8   1000  591004  5.91e8      8    20     5        1
 9     591  1960 5.91e8   1000  591004  5.91e8      9    20     3        1
10     591  1960 5.91e8   1000  591004  5.91e8     10    20     2        1
# ... with 53,543 more rows, and 2 more variables: EDATTAIN <dbl>,
#   EDATTAIND <dbl>
```

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 `tibble` is an improved `data.frame` with nice methods for high-level inspection.
  - By default `tibble` 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.

Tibbles

- Can create small tibbles by hand by entering data row or column wise
- Convert a `data.frame` object to a `tibble` using `as_tibble()` function

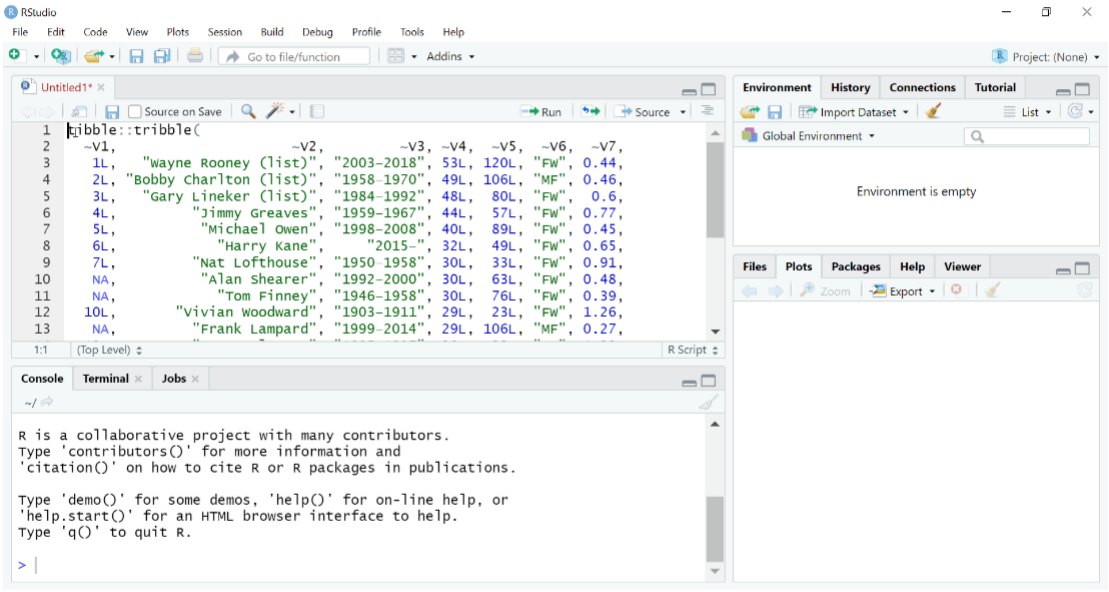


Tibbles

```
> # convert a data.frame to a tibble using as_tibble()
> as_tibble(df0)
# A tibble: 53,553 x 12
   COUNTRY  YEAR SAMPLE SERIAL GEOLEV1 GEOLEV2 PERNUM PERWT   AGE NATIVITY
     <int> <int>  <int>  <int>   <int>   <int>  <int> <int> <int>    <int>
 1     591  1960 5.91e8   1000  591004  5.91e8      1    20    53        1
 2     591  1960 5.91e8   1000  591004  5.91e8      2    20    54        1
 3     591  1960 5.91e8   1000  591004  5.91e8      3    20    31        1
 4     591  1960 5.91e8   1000  591004  5.91e8      4    20    22        1
 5     591  1960 5.91e8   1000  591004  5.91e8      5    20    20        1
 6     591  1960 5.91e8   1000  591004  5.91e8      6    20    16        1
 7     591  1960 5.91e8   1000  591004  5.91e8      7    20    13        1
 8     591  1960 5.91e8   1000  591004  5.91e8      8    20     5        1
 9     591  1960 5.91e8   1000  591004  5.91e8      9    20     3        1
10     591  1960 5.91e8   1000  591004  5.91e8     10    20     2        1
# ... with 53,543 more rows, and 2 more variables: EDATTAIN <int>,
#   EDATTAIND <int>
```

Datapasta

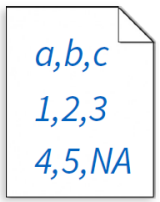
- Copy and paste tables to an R script using the `datapasta` package
- Once installed
  - Highlight table and `Ctrl + C`
  - `Addins | Paste as tribble`



Comma Separated Values

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

Useful arguments



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**)*

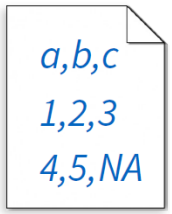
A	B	C
1	2	3
NA	NA	NA

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

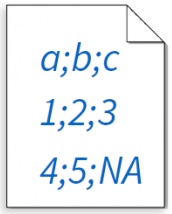


A	B	C
1	2	3
4	5	NA

read_csv()

Reads comma delimited files.

read_csv("file.csv")

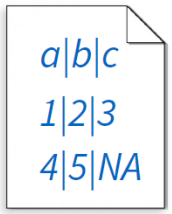


A	B	C
1	2	3
4	5	NA

read_csv2()

Reads Semi-colon delimited files.

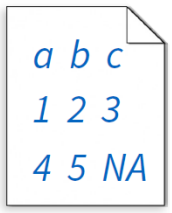
read_csv2("file2.csv")



A	B	C
1	2	3
4	5	NA

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

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 = "../slides-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 (ex31.R)

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

#     b) Load the tidyverse package (that includes the readr package)

##
##
##
# 1.  Open the "2010_Census_Populations_by_Zip_Code.csv" file in the data folder
#     using the file.show() function

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


# 3.  Open the "unhcr_popstats_export_persons_of_concern.csv" file in the data folde

# 4.  Use read_csv to read the data into R and call the results d2 setting appropria
#     a. skip argument
#     b. na argument



# 5.  Open the "tgs00097.tsv" file using the file.show() function

# 6.  Use read_csv to read the data into R and call the results d3 setting appropria
#     na argument
```

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 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
  EDATTAIN EDATTAIND
1        1       110
2        1       120
3        1       120
4        2       212
5        2       212
6        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
```

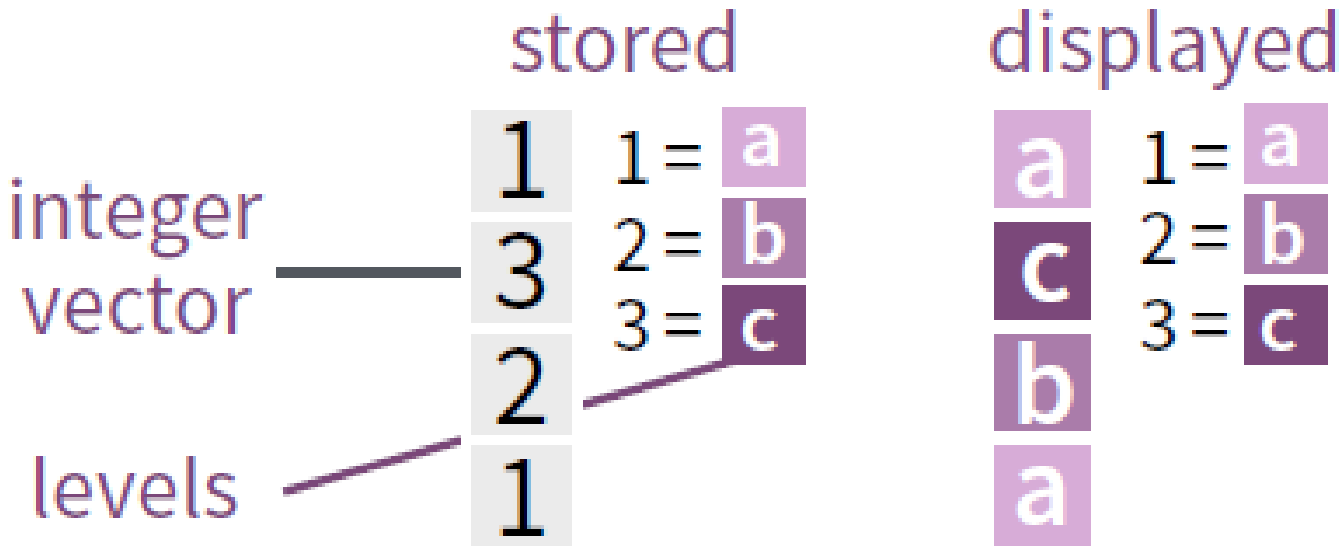
Data Exploration

- 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

- R represents categorical data with factors.
- A factor is an integer vector with a levels attribute.
- The levels attribute stores a set of mappings between integers and categorical values.
- When you view a factor, R displays not the integers, but the values associated with them.



Factors

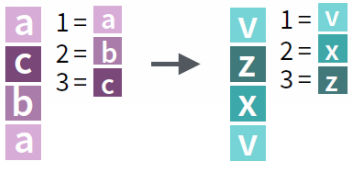
- Data importation functions in the tidyverse, such as `read_csv`, do not create factors by default
  - This is partly why they are faster at importing.
  - Usually only need to define factors when plotting categorical data or fitting a model with a categorical covariate and want a special ordering.
- Older importing functions, such as `read.csv()`, will create factors by default when bringing data into R. (`stringsAsFactors = TRUE`)
- Create factors with the `factor()` function.
  - Levels follows alpha-numeric order by default.

Factors

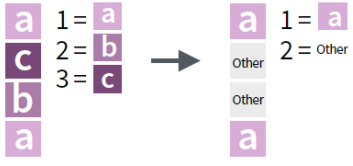
```
> # original numeric data
> head(df2, n = 3)
# A tibble: 3 x 7
  name                          period    mid_period   imr   tfr sex_ratio    pop
  <chr>                         <chr>          <dbl> <dbl> <dbl>     <dbl>  <dbl>
1 China                         1950-1955       1953 129.   6.11      1.07 5.93e5
2 Dem. Peoples Republic of Ko~ 1950-1955       1953 121.   3.46      1.05 9.96e3
3 Japan                         1950-1955       1953  47.7  2.96      1.06 8.69e4
> levels(df2$name)
NULL
>
> # covert to factor
> df2$name <- factor(df2$name)
> levels(df2$name)
[1] "China"                          "Dem. Peoples Republic of Korea"
[3] "Japan"                          "Mongolia"
[5] "Republic of Korea"
> head(df2, n = 3)
# A tibble: 3 x 7
  name                          period    mid_period   imr   tfr sex_ratio    pop
  <fct>                         <chr>          <dbl> <dbl> <dbl>     <dbl>  <dbl>
1 China                         1950-1955       1953 129.   6.11      1.07 5.93e5
2 Dem. Peoples Republic of Ko~ 1950-1955       1953 121.   3.46      1.05 9.96e3
3 Japan                         1950-1955       1953  47.7  2.96      1.06 8.69e4
```

Changing Factors Values

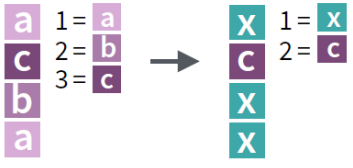
- The `forcats` package has lots of helpful functions help work with factors and categorical data.
  - Loaded as part of the `tidyverse`
- Forcats has functions to change categorical values (i.e. recode)



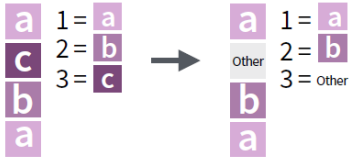
fct_recode(.f, ...) Manually change levels. Also **fct_relabel** which obeys purrr::map syntax to apply a function or expression to each level.
fct_recode(f, v = "a", x = "b", z = "c")
fct_relabel(f, ~ paste0("x", .x))



fct_lump(f, n, prop, w = NULL, other_level = "Other", ties.method = c("min", "average", "first", "last", "random", "max")) Lump together least/most common levels into a single level. Also **fct_lump_min**.
fct_lump(f, n = 1)



fct_collapse(.f, ...) Collapse levels into manually defined groups.
fct_collapse(f, x = c("a", "b"))



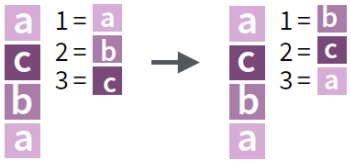
fct_other(f, keep, drop, other_level = "Other") Replace levels with "other."
fct_other(f, keep = c("a", "b"))

Changing Factors Values

```
> # current order
> levels(df2$name)
[1] "China"                          "Dem. Peoples Republic of Korea"
[3] "Japan"                          "Mongolia"
[5] "Republic of Korea"
>
> # reorder factors (defined)
> df2$name <- fct_recode(df2$name,
+                        "North Korea" = "Dem. Peoples Republic of Korea",
+                        "South Korea" = "Republic of Korea")
> levels(df2$name)
[1] "China"       "North Korea" "Japan"       "Mongolia"    "South Korea"
> df2
# A tibble: 70 x 7
   name        period    mid_period   imr   tfr sex_ratio     pop
   <fct>       <chr>          <dbl> <dbl> <dbl>     <dbl>   <dbl>
 1 China       1950-1955       1953 129.   6.11      1.07 593366.
 2 North Korea 1950-1955       1953 121.   3.46      1.05   9957.
 3 Japan       1950-1955       1953  47.7  2.96      1.06  86870.
 4 Mongolia    1950-1955       1953 183.   5.6       1.03    823.
 5 South Korea 1950-1955       1953 159.   5.65      1.06  20293.
 6 China       1955-1960       1958 131.   5.48      1.07 640296.
 7 North Korea 1955-1960       1958  82.6  5.12      1.05  10844.
 8 Japan       1955-1960       1958  36.1  2.17      1.06  91878.
 9 Mongolia    1955-1960       1958 164.   6.3       1.03    910.
```

Changing Factors Order

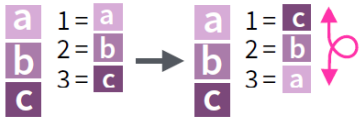
- Often need to rearrange the orders of the factors when plotting or modelling categorical data.



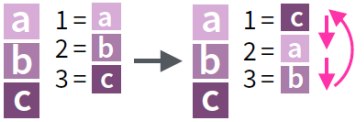
fct_relevel(.f, ..., after = 0L)
Manually reorder factor levels.
fct_relevel(f, c("b", "c", "a"))

fct_infreq(f, ordered = NA)
Reorder levels by the frequenc
in which they appear in the
data (highest frequency first).
f3 <- factor(c("c", "c", "a"))
fct_infreq(f3)

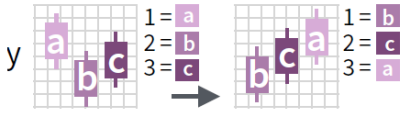
fct_inorder(f, ordered = NA)
Reorder levels by order in
which they appear in the data.
fct_inorder(f2)



fct_rev(f) Reverse level order.
f4 <- factor(c("a","b","c"))
fct_rev(f4)



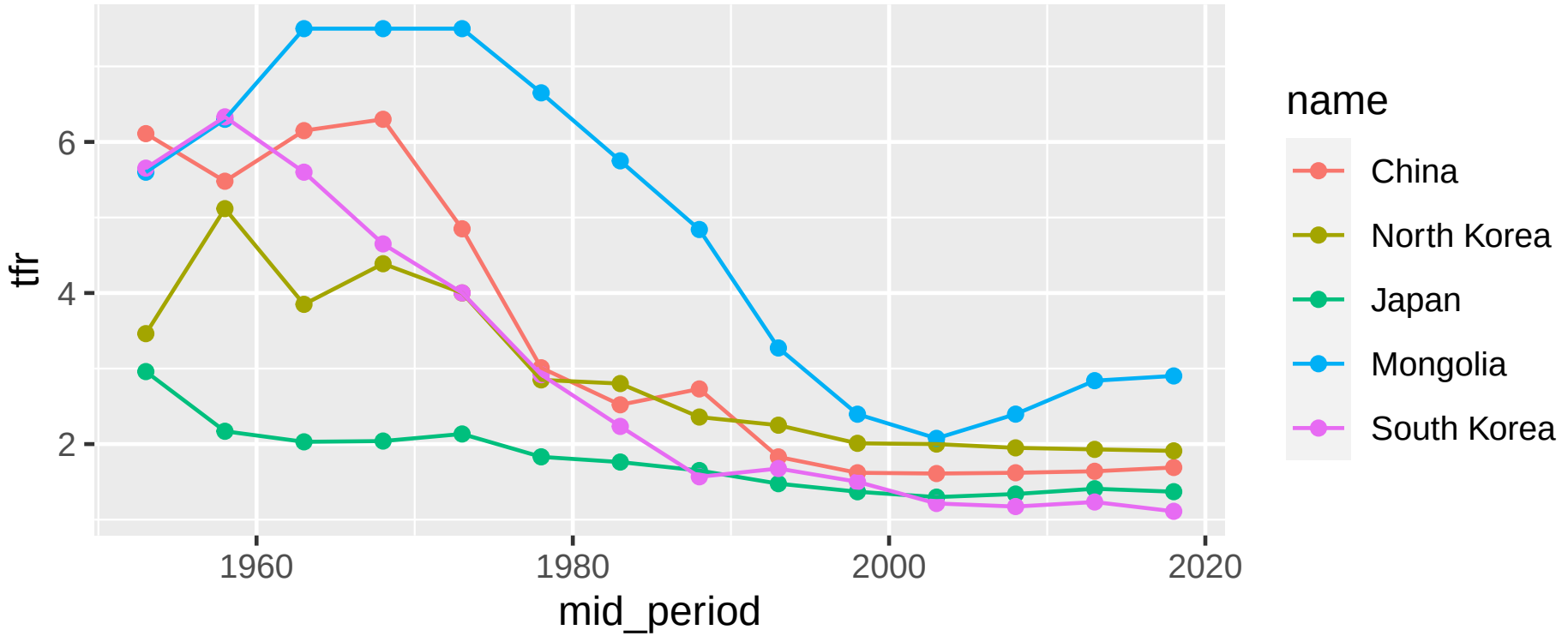
fct_shift(f) Shift levels to left
or right, wrapping around end.
fct_shift(f4)



fct_reorder(.f, .x, .fun=median, ...,
.desc = FALSE) Reorder levels by
their relationship with another
variable.
boxplot(data = iris, Sepal.Width ~
fct_reorder(Species, Sepal.Width))

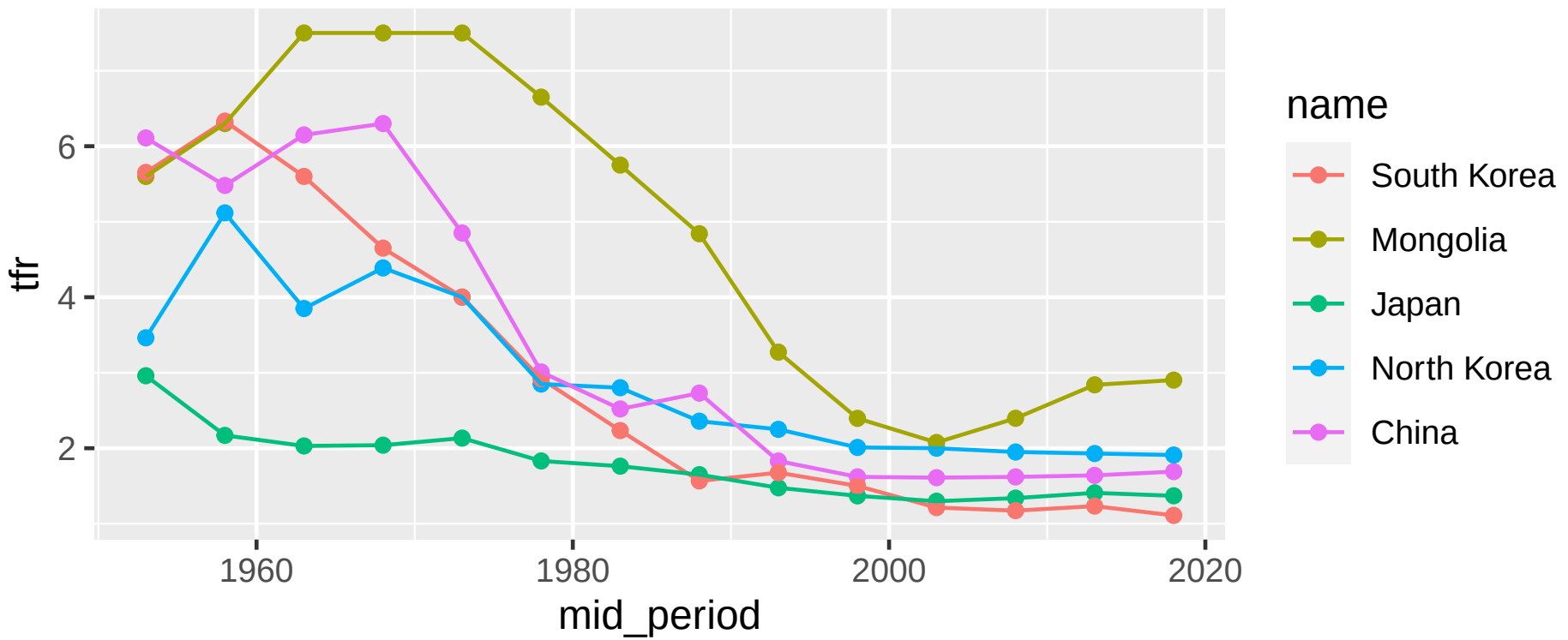
Changing Factors Order

```
> # used order of levels in plot
> levels(df2$name)
[1] "China"       "North Korea" "Japan"       "Mongolia"    "South Korea"
>
> ggplot(data = df2,
+        mapping = aes(x = mid_period, y = tfr, colour = name)) +
+   geom_point() +
+   geom_line()
```



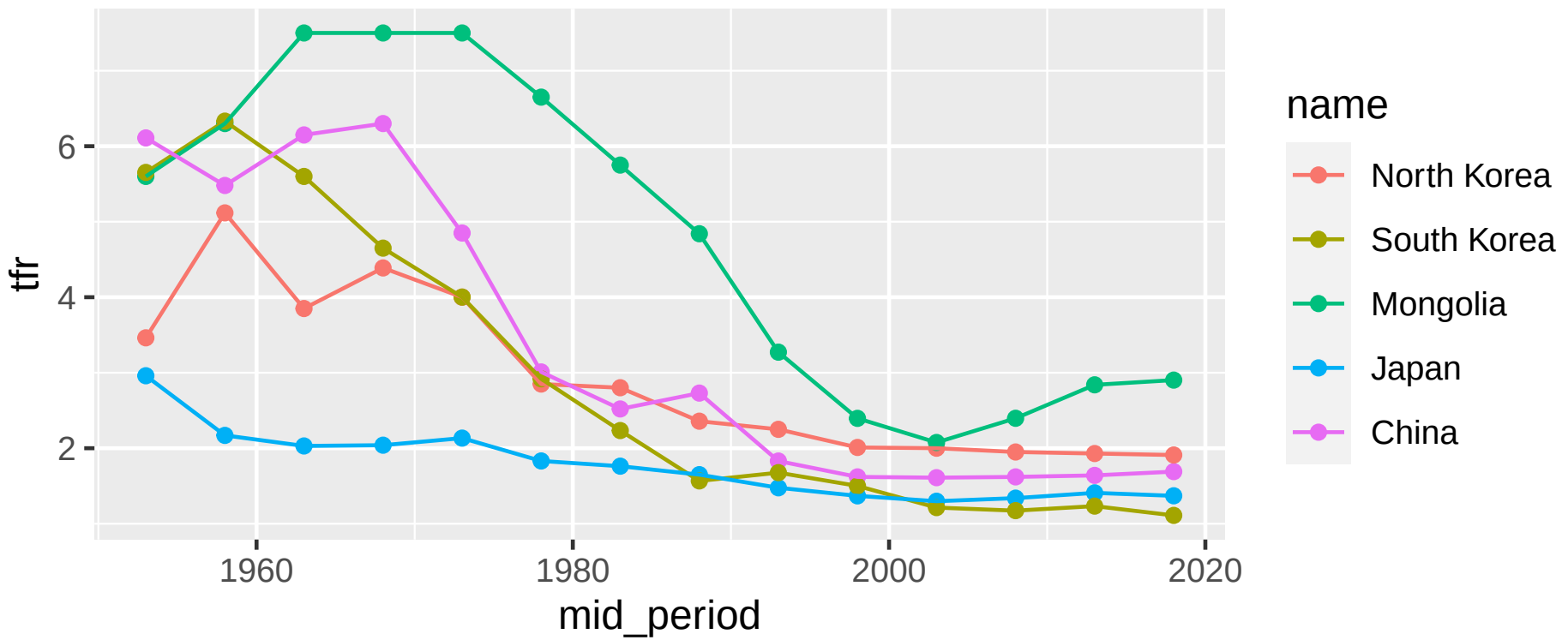
Changing Factors Order

```
> # reverse order
> df2$name <- fct_rev(df2$name)
> levels(df2$name)
[1] "South Korea" "Mongolia"    "Japan"       "North Korea" "China"
>
> ggplot(data = df2,
+        mapping = aes(x = mid_period, y = tfr, colour = name)) +
+   geom_point() +
+   geom_line()
```



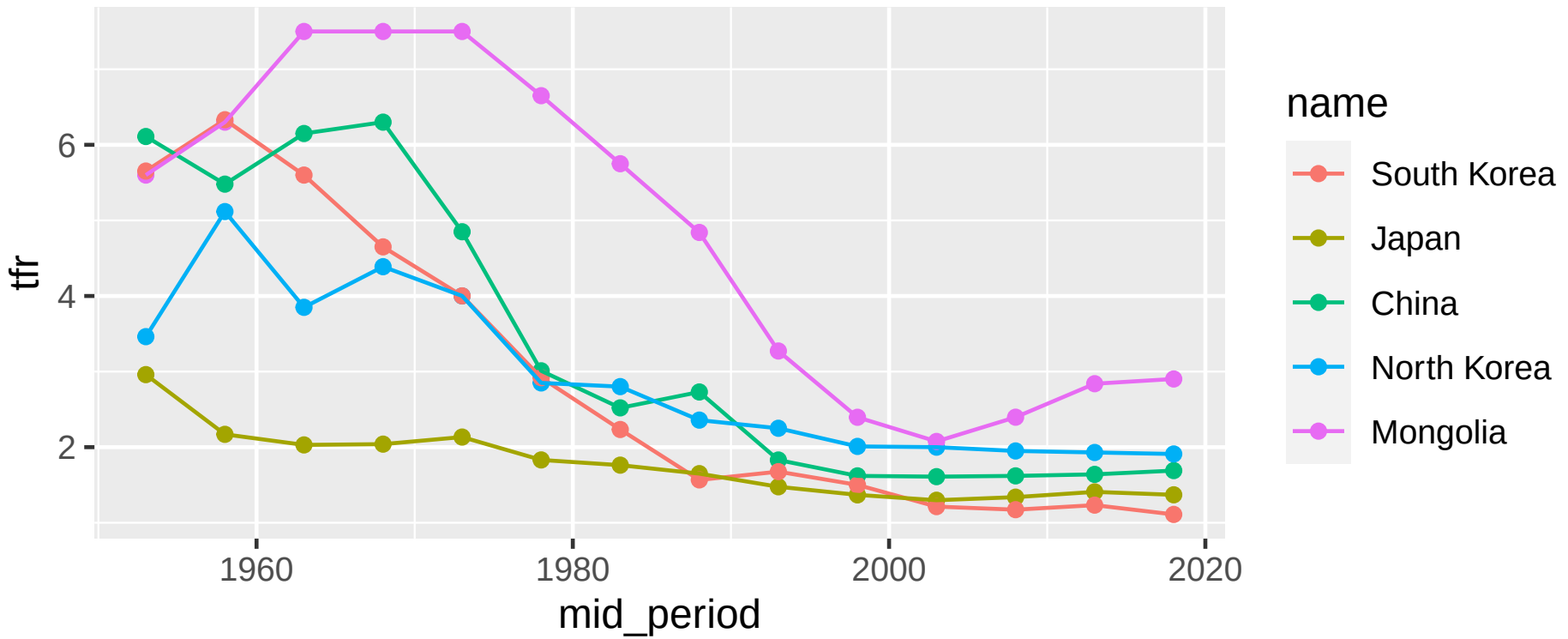
Changing Factors Order

```
> # manual reorder
> df2$name <- fct_relevel(df2$name, "North Korea")
> levels(df2$name)
[1] "North Korea" "South Korea" "Mongolia"    "Japan"       "China"
>
> ggplot(data = df2,
+        mapping = aes(x = mid_period, y = tfr, colour = name)) +
+   geom_point() +
+   geom_line()
```



Changing Factors Order

```
> # reorder factors (by minimum tfr)
> df2$name <- fct_reorder(df2$name, .x = df2$tfr, .fun = min)
> levels(df2$name)
[1] "South Korea" "Japan"       "China"       "North Korea" "Mongolia"
>
> ggplot(data = df2,
+        mapping = aes(x = mid_period, y = tfr, colour = name)) +
+   geom_point() +
+   geom_line()
```



Changing Factors Order

```
> # remember apperance of categories in data is the same,
> # only the order of levels (the categroies) has changed
> head(df2)
# A tibble: 6 x 7
  name        period    mid_period   imr   tfr sex_ratio     pop
  <fct>       <chr>          <dbl> <dbl> <dbl>     <dbl>   <dbl>
1 China       1950-1955       1953 129.   6.11      1.07 593366.
2 North Korea 1950-1955       1953 121.   3.46      1.05   9957.
3 Japan       1950-1955       1953  47.7  2.96      1.06  86870.
4 Mongolia    1950-1955       1953 183.   5.6       1.03    823.
5 South Korea 1950-1955       1953 159.   5.65      1.06  20293.
6 China       1955-1960       1958 131.   5.48      1.07 640296.
```

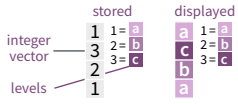
RStudio Data Import Cheatsheet

Factors with forcats : : CHEAT SHEET

The **forcats** package provides tools for working with factors, which are R's data structure for categorical data.

Factors

R represents categorical data with factors. A **factor** is an integer vector with a **levels** attribute that stores a set of mappings between integers and categorical values. When you view a factor, R displays not the integers, but the values associated with them.



Create a factor with factor()

factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA) Convert a vector to a factor. Also **as_factor**.
f <- factor(c("a", "c", "b", "a"), levels = c("a", "b", "c"))

Return its levels with levels()

levels(x) Return/set the levels of a factor. *levels(f); levels(f) <- c("x","y","z")*

Use unclass() to see its structure

Inspect Factors

fct_count(f, sort = FALSE) Count the number of values with each level. *fct_count(f)*

fct_unique(f) Return the unique values, removing duplicates. *fct_unique(f)*

Combine Factors

fct_c(…) Combine factors with different levels.
f1 <- factor(c("a", "c"))
f2 <- factor(c("b", "a"))
fct_c(f1, f2)

fct_unify(fs, levels = lvls_union(fs)) Standardize levels across a list of factors.
fct_unify(list(f2, f1))

Change the order of levels

fct_relevel(.f, ..., after = 0L) Manually reorder factor levels.
fct_relevel(f, c("b", "c", "a"))

fct_infreq(f, ordered = NA) Reorder levels by the frequency in which they appear in the data (highest frequency first).
f3 <- factor(c("c", "c", "a"))
fct_infreq(f3)

fct_inorder(f, ordered = NA) Reorder levels by order in which they appear in the data.
fct_inorder(f2)

fct_rev(f) Reverse level order.
f4 <- factor(c("a","b","c"))
fct_rev(f4)

fct_shift(f) Shift levels to left or right, wrapping around end.
fct_shift(f4)

fct_shuffle(f, n = 1L) Randomly permute order of factor levels.
fct_shuffle(f4)

fct_reorder(.f, .x, .fun=median, ..., .desc = FALSE) Reorder levels by their relationship with another variable.
boxplot(data = iris, Sepal.Width ~ fct_reorder(Species, Sepal.Width))

fct_reorder2(.f, .x, .y, .fun = last2, ..., .desc = TRUE) Reorder levels by their final values when plotted with two other variables.
ggplot(data = iris, aes(Sepal.Width, Sepal.Length, color = fct_reorder2(Species, Sepal.Width, Sepal.Length))) + geom_smooth()

Change the value of levels

fct_recode(.f, ...) Manually change levels. Also **fct_relabel** which obeys purrr::map syntax to apply a function or expression to each level.
fct_recode(f, v = "a", x = "b", z = "c")
fct_relabel(f, ~ paste0("x", .x))

fct_anon(f, prefix = "")) Anonymize levels with random integers. *fct_anon(f)*

fct_collapse(.f, ...) Collapse levels into manually defined groups.
fct_collapse(f, x = c("a", "b"))

fct_lump(f, n, prop, w = NULL, other_level = "Other", ties.method = c("min", "average", "first", "last", "random", "max")) Lump together least/most common levels into a single level. Also **fct_lump_min**.
fct_lump(f, n = 1)

fct_other(f, keep, drop, other_level = "Other") Replace levels with "other."
fct_other(f, keep = c("a", "b"))

Add or drop levels

fct_drop(f, only) Drop unused levels.
f5 <- factor(c("a","b"),c("a","b","x"))
f6 <- fct_drop(f5)

fct_expand(f, …) Add levels to a factor. *fct_expand(f6, "x")*

fct_explicit_na(f, na_level="(Missing)") Assigns a level to NAs to ensure they appear in plots, etc.
fct_explicit_na(factor(c("a", "b", NA)))

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_tibble()`	creates a tibble

Data Modes

```
> # Turn NATIVITY to a character vector
> df1$NATIVITY <- as.character(df1$NATIVITY)
> head(df1, n = 3)
# A tibble: 3 x 12
  COUNTRY  YEAR SAMPLE SERIAL GEOLEV1 GEOLEV2 PERNUM PERWT   AGE NATIVITY
    <dbl> <dbl>  <dbl>  <dbl>   <dbl>   <dbl>  <dbl> <dbl> <dbl> <chr>
1     591  1960 5.91e8   1000  591004  5.91e8      1    20    53 1
2     591  1960 5.91e8   1000  591004  5.91e8      2    20    54 1
3     591  1960 5.91e8   1000  591004  5.91e8      3    20    31 1
# ... with 2 more variables: EDATTAIN <dbl>, EDATTAIND <dbl>
>
> as_tibble(swiss)
# A tibble: 47 x 6
   Fertility Agriculture Examination Education Catholic Infant.Mortality
       <dbl>       <dbl>       <int>     <int>    <dbl>            <dbl>
 1      80.2        17             15        12     9.96             22.2
 2      83.1        45.1            6         9    84.8              22.2
 3      92.5        39.7            5         5    93.4              20.2
 4      85.8        36.5           12         7    33.8              20.3
 5      76.9        43.5           17        15     5.16             20.6
 6      76.1        35.3            9         7    90.6              26.6
 7      83.8        70.2           16         7    92.8              23.6
 8      92.4        67.8           14         8    97.2              24.9
 9      82.4        53.3           12         7    97.7              21
10      82.9        45.2           16        13    91.4              24.4
```

Exercise 2 (ex32.R)

```r
# 0.  a) Check your working directory is in the course folder. Load the .Rproj file
getwd()
#     b) Load the tidyverse package (that includes the readr package)
library(tidyverse)
#     c) Run the code in ex32_prelim.R to import the data for this exercise
source("./exercise/ex32_prelim.R")
##
##
##
# 1. Show 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. Convert the `name` column in d5 from a character string to a factor

# 6. Check the order of the new levels

# 7. Reverse the order of the levels for the name column in d5

# 8. Check the order of the new levels
```

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)
New names:
* `` -> ...3
> df2
# A tibble: 7,549 x 23
   `Area Codes` `Area Names` ...3  `All ages` `0-4` `5-9` `10-14` `15-19`
   <chr>        <chr>        <chr>      <dbl> <dbl> <dbl>   <dbl>   <dbl>
 1 E06000047    County Durh~ <NA>      519695 28446 28859   25936   30787
 2 E02004297    <NA>         Coun~       7912   455   421     328     387
 3 E02004290    <NA>         Coun~       5851   251   313     321     300
 4 E02004298    <NA>         Coun~       9858   488   524     516     509
 5 E02004299    <NA>         Coun~       8588   555   506     441     427
 6 E02004291    <NA>         Coun~       6957   427   376     336     379
 7 E02004300    <NA>         Coun~       7840   496   506     350     392
 8 E02004292    <NA>         Coun~       7845   350   444     360     385
 9 E02004301    <NA>         Coun~       9205   653   666     550     578
10 E02004302    <NA>         Coun~       7766   508   469     312     419
# ... with 7,539 more rows, and 15 more variables: `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 = colnames(df2))
> freezePane(wb = wb0, sheet = 1, firstRow = TRUE)
>
> ## save workbook to working directory
> saveWorkbook(wb0, file = "../slides-data/example.xlsx", overwrite = TRUE)
> file.show("../slides-data/example.xlsx")
```

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!
- When reading SPSS and STATA files categorical data comes in a labelled format
  - Usually need to convert to factor for modelling or plotting

SPSS Data

```
> library(haven)
> df3 <- read_sav(file = "../slides-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 [Pan~ 1960 [196~ 5.91e8 [Pan~   1000  591004  5.91e8      1    20
 2 591 [Pan~ 1960 [196~ 5.91e8 [Pan~   1000  591004  5.91e8      2    20
 3 591 [Pan~ 1960 [196~ 5.91e8 [Pan~   1000  591004  5.91e8      3    20
 4 591 [Pan~ 1960 [196~ 5.91e8 [Pan~   1000  591004  5.91e8      4    20
 5 591 [Pan~ 1960 [196~ 5.91e8 [Pan~   1000  591004  5.91e8      5    20
 6 591 [Pan~ 1960 [196~ 5.91e8 [Pan~   1000  591004  5.91e8      6    20
 7 591 [Pan~ 1960 [196~ 5.91e8 [Pan~   1000  591004  5.91e8      7    20
 8 591 [Pan~ 1960 [196~ 5.91e8 [Pan~   1000  591004  5.91e8      8    20
 9 591 [Pan~ 1960 [196~ 5.91e8 [Pan~   1000  591004  5.91e8      9    20
10 591 [Pan~ 1960 [196~ 5.91e8 [Pan~   1000  591004  5.91e8     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

- Closer look at `EDATTAIN` labelled variable:

```
> df3[,-(1:6)]
# A tibble: 53,553 x 6
   PERNUM PERWT        AGE     NATIVITY            EDATTAIN            EDATTAIND
    <dbl> <dbl>  <dbl+lbl>    <dbl+lbl>          <dbl+lbl>          <dbl+lbl>
 1      1    20 53 [53]     1 [Native-b~ 1 [Less than prima~ 110 [No schooling]
 2      2    20 54 [54]     1 [Native-b~ 1 [Less than prima~ 120 [Some primary c~
 3      3    20 31 [31]     1 [Native-b~ 1 [Less than prima~ 120 [Some primary c~
 4      4    20 22 [22]     1 [Native-b~ 2 [Primary complet~ 212 [Primary (6 yrs~
 5      5    20 20 [20]     1 [Native-b~ 2 [Primary complet~ 212 [Primary (6 yrs~
 6      6    20 16 [16]     1 [Native-b~ 2 [Primary complet~ 212 [Primary (6 yrs~
 7      7    20 13 [13]     1 [Native-b~ 2 [Primary complet~ 212 [Primary (6 yrs~
 8      8    20  5 [5]      1 [Native-b~ 0 [NIU (not in uni~   0 [NIU (not in un~
 9      9    20  3 [3]      1 [Native-b~ 0 [NIU (not in uni~   0 [NIU (not in un~
10     10    20  2 [2 yea~ 1 [Native-b~ 0 [NIU (not in uni~   0 [NIU (not in un~
# ... with 53,543 more rows
```

Labelled Data

- The labels for EDATTAIN are attached to each value.
- The column itself is a set of values though
  - If use in a model or plot will take the numeric codes not the labels
  - The labels are more useful though when plotting or modelling

```
> print_labels(df3$EDATTAIN)

Labels:
 value                          label
     0        NIU (not in universe)
     1 Less than primary completed
     2            Primary completed
     3          Secondary completed
     4          University completed
     9                       Unknown
>
> table(df3$EDATTAIN)

    0     1     2     3     4     9
14605 26260 10939  1311   319   119
```

Labelled Data

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

```
> df3$EDATTAIN <- as_factor(df3$EDATTAIN, "labels")
> table(df3$EDATTAIN)

      NIU (not in universe) Less than primary completed
                      14605                       26260
          Primary completed         Secondary completed
                      10939                        1311
       University completed                     Unknown
                        319                         119
>
> head(df3)[,-(1:6)]
# A tibble: 6 x 6
  PERNUM PERWT       AGE     NATIVITY EDATTAIN                    EDATTAIND
   <dbl> <dbl> <dbl+lbl>    <dbl+lbl> <fct>                        <dbl+lbl>
1      1    20   53 [53] 1 [Native-b~ Less than primary ~ 110 [No schooling]
2      2    20   54 [54] 1 [Native-b~ Less than primary ~ 120 [Some primary com~
3      3    20   31 [31] 1 [Native-b~ Less than primary ~ 120 [Some primary com~
4      4    20   22 [22] 1 [Native-b~ Primary completed   212 [Primary (6 yrs) ~
5      5    20   20 [20] 1 [Native-b~ Primary completed   212 [Primary (6 yrs) ~
6      6    20   16 [16] 1 [Native-b~ Primary completed   212 [Primary (6 yrs) ~
```

Labelled Data

- Can bulk convert labelled variables to factors

```
> as_factor(df3, levels = "labels")[-(1:6)]
# A tibble: 53,553 x 6
   PERNUM PERWT AGE     NATIVITY    EDATTAIN                 EDATTAIND
    <dbl> <dbl> <fct>   <fct>       <fct>                    <fct>
 1      1    20 53      Native-born Less than primary com~ No schooling
 2      2    20 54      Native-born Less than primary com~ Some primary complet~
 3      3    20 31      Native-born Less than primary com~ Some primary complet~
 4      4    20 22      Native-born Primary completed      Primary (6 yrs) comp~
 5      5    20 20      Native-born Primary completed      Primary (6 yrs) comp~
 6      6    20 16      Native-born Primary completed      Primary (6 yrs) comp~
 7      7    20 13      Native-born Primary completed      Primary (6 yrs) comp~
 8      8    20 5       Native-born NIU (not in universe)  NIU (not in universe)
 9      9    20 3       Native-born NIU (not in universe)  NIU (not in universe)
10     10    20 2 years Native-born NIU (not in universe)  NIU (not in universe)
# ... with 53,543 more rows
```

Labelled Data

- Not all labelled data is necessarily categorical, for example `AGE` in our file
- Can change labelled column to numeric values using `zap_labels()` function in the `haven` package

```
> df3$AGE[1:10]
<labelled<double>[10]>: Age
 [1] 53 54 31 22 20 16 13  5  3  2

Labels:
 value                 label
     0     Less than 1 year
     1                1 year
     2               2 years
     3                     3
     4                     4
     5                     5
     6                     6
     7                     7
     8                     8
     9                     9
    10                    10
    11                    11
    12                    12
    13                    13
```

Labelled Data

```
> # age to numeric
> df3$AGE <- zap_labels(x = df3$AGE)
>
> # rest of labelled data to factors
> df3 <- as_factor(df3, levels = "labels")
>
> df3[, -(1:6)]
# A tibble: 53,553 x 6
   PERNUM PERWT   AGE NATIVITY    EDATTAIN                    EDATTAIND
    <dbl> <dbl> <dbl> <fct>       <fct>                       <fct>
 1      1    20    53 Native-born Less than primary comp~     No schooling
 2      2    20    54 Native-born Less than primary comp~     Some primary completed
 3      3    20    31 Native-born Less than primary comp~     Some primary completed
 4      4    20    22 Native-born Primary completed           Primary (6 yrs) compl~
 5      5    20    20 Native-born Primary completed           Primary (6 yrs) compl~
 6      6    20    16 Native-born Primary completed           Primary (6 yrs) compl~
 7      7    20    13 Native-born Primary completed           Primary (6 yrs) compl~
 8      8    20     5 Native-born NIU (not in universe)       NIU (not in universe)
 9      9    20     3 Native-born NIU (not in universe)       NIU (not in universe)
10     10    20     2 Native-born NIU (not in universe)       NIU (not in universe)
# ... with 53,543 more rows
```

Stata Data

- The `read_dta()` function in the `haven` package for Stata files.

```
> # haven
> df4 <- read_dta("../slides-data/ipumsi_pan1960.dta")
> # gives a tibble
> head(df4, n = 3)
# A tibble: 3 x 12
    country        year        sample serial geolev1 geolev2 pernum perwt      age
  <dbl+lbl>  <dbl+lbl>     <dbl+lbl>  <dbl>   <dbl>   <dbl>  <dbl> <dbl> <dbl+l>
1 591 [pan~ 1960 [196~ 5.91e8 [pan~   1000  591004  5.91e8      1    20 53 [53]
2 591 [pan~ 1960 [196~ 5.91e8 [pan~   1000  591004  5.91e8      2    20 54 [54]
3 591 [pan~ 1960 [196~ 5.91e8 [pan~   1000  591004  5.91e8      3    20 31 [31]
# ... with 3 more variables: nativity <dbl+lbl>, edattain <dbl+lbl>,
#   edattaind <dbl+lbl>
>
> df4 <- as_factor(df4)
> head(df4, n = 3)
# A tibble: 3 x 12
  country year  sample serial geolev1 geolev2 pernum perwt age   nativity
  <fct>   <fct> <fct>   <dbl>   <dbl>   <dbl>  <dbl> <dbl> <fct> <fct>
1 panama  1960  panam~   1000  591004  5.91e8      1    20 53    native~
2 panama  1960  panam~   1000  591004  5.91e8      2    20 54    native~
3 panama  1960  panam~   1000  591004  5.91e8      3    20 31    native~
# ... with 2 more variables: edattain <fct>, edattaind <fct>
```

SAS data

- SAS files saved as .sas7bdat can work with haven `read_sas`

```
> # haven
> df6 <- read_sas("../slides-data/ipumsi_pan1960.sas7bdat")
> # gives a tibble
> head(df6)
# A tibble: 6 x 12
  COUNTRY  YEAR SAMPLE SERIAL GEOLEV1 GEOLEV2 PERNUM PERWT   AGE NATIVITY
    <dbl> <dbl>  <dbl>  <dbl>   <dbl>   <dbl>  <dbl> <dbl> <dbl>    <dbl>
1     591  1960 5.91e8   1000  591004  5.91e8      1    20    53        1
2     591  1960 5.91e8   1000  591004  5.91e8      2    20    54        1
3     591  1960 5.91e8   1000  591004  5.91e8      3    20    31        1
4     591  1960 5.91e8   1000  591004  5.91e8      4    20    22        1
5     591  1960 5.91e8   1000  591004  5.91e8      5    20    20        1
6     591  1960 5.91e8   1000  591004  5.91e8      6    20    16        1
# ... with 2 more variables: 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_sav()`	haven	Yes	SPSS files
`read_dta()`	haven	Yes	Stata 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_sav()`	haven	Yes	SPSS 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()**)**

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")

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

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

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

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

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

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

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

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

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
sex is a character
earn is a double (numeric)

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)

Exercise 3 (ex33.R)

```
# 0. a) Check your working directory is in the course folder. Load the .Rproj file i

#    b) Load the tidyverse, readxl, haven and openxlsx packages
library(tidyverse)



##
##
##
# 1. Use read_excel to read data on Male population totals in SAPE18DT14.xls file in
#    into R and call the results d6



# 2. Use read_excel to read data on ESTIMATES in WPP2019_FERT_F04_TOTAL_FERTILITY.xl



# 3. Read in the data from ipumsi_pan.dta

# 4. Print the labels in d8 for the age variable

# 5. Remove the labels from the age column in d8
```