Session 11
Data Wrangling 1

R for Social Data Science

Jeffrey Ziegler, PhD

Assistant Professor in Political Science & Data Science
Trinity College Dublin

Fall 2022

Road map for today

Last time:

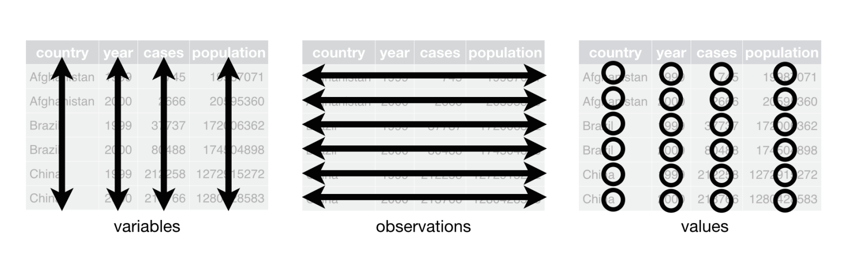
- Debugging
- Handling conditions
- Testing
- Defensive programming

This time:

- Data frames in base R
- Alternatives to data frames

Data organization

- 'Tidy' data is a specific subset of rectangular data, where:
  - Each variable is in a column
  - Each observation is in a row
  - Each value is in a cell



Source: R for Data Science

Data frames

- Data frame is one of the object types available in base R
- Despite their matrix-like appearance, data frames are lists of equal-sized vectors
- Data frames can be created with ‘data.frame()‘ function with named vectors as input

```r
df <- data.frame(
    x = 1:4,
    y = c("a", "b", "c", "d"),
    z = c(TRUE, FALSE, FALSE, TRUE)
)
```

```
x y     z
1 1 a  TRUE
2 2 b FALSE
3 3 c FALSE
4 4 d  TRUE
```

Example Data frame

```r
# str() function applied to data frame is useful in determining variable types
str(df)
```

```
'data.frame': 4 obs. of  3 variables:
$ x: int  1 2 3 4
$ y: chr  "a" "b" "c" "d"
$ z: logi  TRUE FALSE FALSE TRUE
```

```r
# dim() function behaves similar to matrix, showing N rows and N columns, respectively
dim(df)
```

```
[1] 4 3
```

```r
# In contrast to matrix length() of data frame displays the length of underlying list
length(df)
```

```
[1] 4 3
```

Creating data frame

```r
l <- list(x = 1:5, y = letters[1:5], z = rep(c(TRUE, FALSE), length.out = 5))
l
```

```
$x
[1] 1 2 3 4 5

$y
[1] "a" "b" "c" "d" "e"

$z
[1]  TRUE FALSE  TRUE FALSE  TRUE
```

```r
df <- data.frame(l)
df
```

```
x y     z
1 1 a  TRUE
2 2 b FALSE
3 3 c  TRUE
4 4 d FALSE
5 5 e  TRUE
```

Subsetting data frame

- In subsetting data frames, techniques of subsetting matrices and lists are combined
- If you subset with a single vector, it behaves as a list
- If you subset with two vectors, it behaves as a matrix

Ex: Subsetting data frame

```
# Like a list
df[c("x","z")]
```

```
x     z
1 1  TRUE
2 2 FALSE
3 3  TRUE
4 4 FALSE
5 5  TRUE
```

```
# Like a matrix
df[,c("x","z")]
```

```
x     z
1 1  TRUE
2 2 FALSE
3 3  TRUE
4 4 FALSE
5 5  TRUE
```

```
df[df$y=="b",]
```

```
x y     z
2 2 b FALSE
```

Building data frame

- ‘rbind()‘ (row bind) - appends a row to data frame
- ‘cbind()‘ (column bind) - appends a column to data frame
- Both require compatible sizes (number of rows/columns)

Building data frame: Adding columns

```r
set.seed(12345)
rand <- rnorm(5)
rand
```

```
[1]  0.5855288  0.7094660 -0.1093033 -0.4534972  0.6058875
```

```r
df <- cbind(df, rand)
df
```

```
x y     z       rand
1 1 a  TRUE  0.5855288
2 2 b FALSE  0.7094660
3 3 c  TRUE -0.1093033
4 4 d FALSE -0.4534972
5 5 e  TRUE  0.6058875
```

Building data frame: Adding rows

```r
# Note, row has to be list since contains different data types
r <- list(6, letters[6], FALSE, rnorm(1))
r
```

```
[[1]]
[1] 6

[[2]]
[1] "f"

[[3]]
[1] FALSE

[[4]]
[1] -1.817956
```

```r
df <- rbind(df, r)
df
```

```
x y     z       rand
1 1 a  TRUE  0.5855288
2 2 b FALSE  0.7094660
3 3 c  TRUE -0.1093033
4 4 d FALSE -0.4534972
5 5 e  TRUE  0.6058875
6 6 f FALSE -1.8179560
```

ADDING/MODIFYING COLUMNS IN BASE R

```r
# New columns can be created/modified by assignment (if RHS object has correct length)
df["r"] <- rnorm(6)
df
```

```
 x y     z       rand          r
1 1 a  TRUE  0.5855288  0.6300986
2 2 b FALSE  0.7094660 -0.2761841
3 3 c  TRUE -0.1093033 -0.2841597
4 4 d FALSE -0.4534972 -0.9193220
5 5 e  TRUE  0.6058875 -0.1162478
6 6 f FALSE -1.8179560  1.8173120
```

```r
# Individual columns can also be selected with $ operator
df$r <- df$r + 5
df
```

```
x y     z       rand        r
1 1 a  TRUE  0.5855288 5.630099
2 2 b FALSE  0.7094660 4.723816
3 3 c  TRUE -0.1093033 4.715840
4 4 d FALSE -0.4534972 4.080678
5 5 e  TRUE  0.6058875 4.883752
6 6 f FALSE -1.8179560 6.817312
```

Renaming columns in base R

```r
# names() attribute for data frames/tibbles contains column names
names(df)
```

```
[1] "x"    "y"    "z"    "rand" "r"
```

```r
# Individual columns can also be selected with $ operator
df$r <- df$r + 5
df
```

```
x y     z   rand_new        r
1 1 a  TRUE  0.5855288 5.630099
2 2 b FALSE  0.7094660 4.723816
3 3 c  TRUE -0.1093033 4.715840
4 4 d FALSE -0.4534972 4.080678
5 5 e  TRUE  0.6058875 4.883752
6 6 f FALSE -1.8179560 6.817312
```

Issues with data frame

- While very versatile (and available out-of-the-box) data frames have their drawbacks:
  - Individual cells (observations) cannot themselves be lists
  - Somewhat limited (and inconsistent) data manipulation functions
  - Memory inefficient (*copy-on-modify* semantics)
  - No parallelisation

What's helpful for this? We'll talk about alternatives next time!

Data formats in R

- ‘.csv‘ (Comma-separated value) files for storing tabular data
  - Standard file format for storing data that is highly interoperable across systems
  - Human-readable and can be opened in a simple text processor
- ‘.rds‘ (R data serialization) files allow to store single R object
  - Can store arbitrary R objects (e.g. fitted model), similar to Python ‘pickle‘
  - Offers data compression
  - Only works within R

Data formats in R

- ‘.rda‘ (R data) files for saving and loading multiple R objects
  - Offers data compression
  - Compares unfavourably to rds and, generally, should be avoided
- ‘.feather‘/‘.parquet‘ - big data formats associated with Apache Hadoop ecosystem
  - Cutting-edge performance (compression and read/write access)
  - Not human-readable
  - Relatively new, could be an overkill for some tasks

Functions for data I/O

- ‘.csv‘ (Comma-separated value)
  - ‘read.csv()‘/‘write.csv()‘ - base R functions
  - ‘readr::read_csv()‘/‘readr::write_csv()‘ - functions from ‘readr‘ package in ‘tidyverse‘
- ‘.rds‘ (R data serialization)
  - ‘readRDS()‘/‘writeRDS()‘ - base R functions
  - ‘readr::read_rds()‘/‘readr::write_rds()‘ - functions from ‘readr‘ (no default compression)

Functions for data I/O

- ‘.rda‘ (R data)
  - ‘save()‘/‘load()‘ - base R functions
- ‘.feather‘/‘.parquet‘
  - ‘arrow::read_feather()‘/‘arrow::write_feather()‘ - functions from
  - ‘arrow::read_parquet()‘/‘arrow::write_parquet()‘ - ‘arrow‘ package in Apache Arrow

Reading data in R example

```r
# assuming your local GitHub is up-to-date
inc_local <- read.csv("../datasets/incumbents_subset.csv")
inc_url <- read.csv("https://raw.githubusercontent.com/jeffreyziegler/R_social_DS/main/datasets/incumbents_subset.csv")
head(inc_local); head(inc_url)
```

```
X  x year congress chalspend incspend   difflog  presvote voteshare inparty incparty seniority
1 53 53 1978       95  11.67655 12.24663 0.5700871 0.5267782 0.6023614       1        1
2 54 54 1978       95  11.62039 12.49136 0.8709687 0.5659233 0.5836368       0        0
3 55 55 1978       95  12.30557 12.73226 0.4266895 0.4646196 0.5922578       1        1
4 56 56 1978       95  10.54843 12.50500 1.9565633 0.5012287 0.6992224       1        1
5 57 57 1978       95  12.10366 12.76171 0.6580556 0.4774266 0.6419783       1        1
6 58 58 1980       96  12.48744 12.83441 0.3469714 0.5901939 0.6257710       0        0
```

Tutorial - Creating & manipulating data frames

- Load 'kaggle_survey_2021_responses.csv' dataframe from GitHub repository to global environment
- Subset dataframe to include 'time to complete' and first 5 Qs
- Subset to women who have earned less than a doctoral degree

Tutorial - Dummy variables

- When analysing categorical data (particularly using it as independent variables in regression) it is common to construct dummy variables

- Where categorical variables are represented by 1's and 0's depending on whether it is true or not for a given observation

- For example, gender of respondents can be represented by 1's that indicate whether a given respondent is female (baseline/reference category) and 0's if they are not

1. Make a dummy variable for your two criteria from above (women and less than doctoral degrees)

2. Subset original dataset based on new dummy variables

OVERVIEW

This time:

- Data frames in base R
- Data input and output

Next time:

- Alternatives to data frames
- ‘tidyverse‘ packages
- Working with tabular data
- Summary statistics