Week 4: Data Frames

POP88162 Introduction to Quantitative Research Methods

Tom Paskhalis

Department of Political Science, Trinity College Dublin

So Far

- *Vector* is the core *data structure* in R.
- Vectors can be one of the main data types (*character*, *numeric*, *logical*).
- Probabilities for critical values can be calculated using R.

Topics for Today

- Lists
- Data frames
- Working with data frames

Review: Data Structures

Structure	Description	Dimensionality	Data Type
`vector`	Atomic vector (scalar)	1d	homogenous
`matrix`	Matrix	2d	homogenous
`array`	One-, two or n-dimensional array	1d/2d/nd	homogenous
`list`	List	1d	heterogeneous
`data.frame`	Rectangular data	2d	heterogeneous

Lists

- As opposed to vectors, **lists** can contain elements of any type.
- List can also have nested lists within it.
- Lists are constructed using `list()` function in R.
- Data frames are just lists with some special characteristics!

```r
# We can combine different data types in a list and, optionally, name elements (e.g. B below)
l <- list(2:4, list("a"), B = c(TRUE, FALSE, FALSE))
l
```

```
[[1]]
[1] 2 3 4

[[2]]
[[2]][[1]]
[1] "a"


$B
[1]  TRUE FALSE FALSE
```

List Subsetting

- As with vectors you can use `[]` to subset lists
- This will return a list of length one
- Components of the list can be individually extracted using `[[` and `$` operators

```
list[index]
list[[index]]
list$name
```

Data Frames

- Data frame is the workhorse of data analysis in 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)
)
df
```

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

Data Frames: Example

```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
# If we want to explicitly extract number of rows
nrow(df)
```

```
[1] 4
```

```r
# Which is equivalent to
dim(df)[1]
```

```
[1] 4
```

```r
# If we want to explicitly extract number of columns
ncol(df)
```

```
[1] 3
```

```r
# In contrast to matrix length() of data frame displays the length of underlying list
# Which is just a number of columns
length(df)
```

```
[1] 3
```

Creating Data Frame: Example

```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
```

```r
str(df)
```

```
'data.frame':   5 obs. of  3 variables:
 $ x: int  1 2 3 4 5
 $ y: chr  "a" "b" "c" "d" ...
 $ z: logi  TRUE FALSE TRUE FALSE TRUE
```

Subsetting Data Frame

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

```
data_frame[column_indices]
data_frame[column_name(s)]
data_frame$column_name
```

  - If you subset with two vectors, it behaves as a matrix.

```
data_frame[row_indices, column_indices]
data_frame[row_indices, column_name(s)]
```

Subsetting Columns: Example

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

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

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

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

Subsetting Rows: Example

```r
# Subsetting on an existing binary variable coded as logical
df[df$z == TRUE,]
```

```
  x y    z
1 1 a TRUE
3 3 c TRUE
5 5 e TRUE
```

```r
# Subsetting on a dynamically created logical vector
df[df$y == "b",]
```

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

```r
# Note that the internal expression evaluates to logical vector
df$y == "b"
```

```
[1] FALSE  TRUE FALSE FALSE FALSE
```

Manipulating Columns

```r
# New columns can also be created/modified by assignment
# (if the right-hand side object has correct length)
df["r"] <- rnorm(5, mean = 3, sd = 2)
df
```

```
  x y     z        r
1 1 a  TRUE 2.847346
2 2 b FALSE 2.215921
3 3 c  TRUE 3.872687
4 4 d FALSE 5.023907
5 5 e  TRUE 2.920144
```

```r
# Individual columns can also be selected with $ operator
df$r_standardised <- (df$r - mean(df$r)) / sd(df$r)
df
```

```
  x y     z        r r_standardised
1 1 a  TRUE 2.847346     -0.4828276
2 2 b FALSE 2.215921     -1.0595152
3 3 c  TRUE 3.872687      0.4536294
4 4 d FALSE 5.023907      1.5050536
5 5 e  TRUE 2.920144     -0.4163403
```

Column Names

```r
# colnames() or names() attribute for data frames contains column names
colnames(df)
```

```
[1] "x"              "y"              "z"              "r"
[5] "r_standardised"
```

```r
colnames(df)[4] <- "rand"
colnames(df)[5] <- "rand_standardised"
df
```

```
  x y     z     rand rand_standardised
1 1 a  TRUE 2.847346        -0.4828276
2 2 b FALSE 2.215921        -1.0595152
3 3 c  TRUE 3.872687         0.4536294
4 4 d FALSE 5.023907         1.5050536
5 5 e  TRUE 2.920144        -0.4163403
```

Merging Data Frames

- Oftentimes, analysis involves more than one dataset.
- This requires merging (joining) data frames together.
- R function `merge()` can be used for this purpose.
- Assuming the two columns share the same columns name:

```
merge(x, y, by = "column_name")
```

- If the column names differ, the `by.x` and `by.y` arguments can be used:

```
merge(x, y, by.x = "column_name_x", by.y = "column_name_y")
```

- Note that in either case the datasets must share some **unique identifier**.

Merging Data Frames: Example 1

Data Frame 1

```r
candidates1 <- data.frame(
  id = 1:4,
  name = c(
    "Sean M", "Orla G", "Rian P", "Fiona F"
  ),
  party = c(
    "Left", "Right", "Left", "Right"
  )
)
candidates1
```

```
  id    name party
1  1  Sean M  Left
2  2  Orla G Right
3  3  Rian P  Left
4  4 Fiona F Right
```

Data Frame 2

```r
candidates2 <- data.frame(
  id = 1:4,
  yob = c(
    1971, 1985, 1977, 1962
  ),
  experience = c(
    TRUE, FALSE, FALSE, TRUE
  )
)
candidates2
```

```
  id  yob experience
1  1 1971       TRUE
2  2 1985      FALSE
3  3 1977      FALSE
4  4 1962       TRUE
```

Merging Data Frames: Example 1

```r
candidates_merged <- merge(candidates1, candidates2, by = "id")
```

```r
candidates_merged
```

```
  id    name party  yob experience
1  1  Sean M  Left 1971       TRUE
2  2  Orla G Right 1985      FALSE
3  3  Rian P  Left 1977      FALSE
4  4 Fiona F Right 1962       TRUE
```

The order of arguments in `merge()` when the column name is the same is not important (but affects the order of columns in the merged data frame).

```r
candidates_merged <- merge(candidates2, candidates1, by = "id")
```

```r
candidates_merged
```

```
  id  yob experience    name party
1  1 1971       TRUE  Sean M  Left
2  2 1985      FALSE  Orla G Right
3  3 1977      FALSE  Rian P  Left
4  4 1962       TRUE Fiona F Right
```

Merging Data Frames: Example 2

Data Frame 1

```r
candidates1 <- data.frame(
  candidate_id = 1:4,
  name = c(
    "Sean M", "Orla G", "Rian P", "Fiona F"
  ),
  party = c(
    "Left", "Right", "Left", "Right"
  )
)
candidates1
```

```
  candidate_id    name party
1            1  Sean M  Left
2            2  Orla G Right
3            3  Rian P  Left
4            4 Fiona F Right
```

Data Frame 2

```r
candidates2 <- data.frame(
  id = 1:4,
  yob = c(
    1971, 1985, 1977, 1962
  ),
  experience = c(
    TRUE, FALSE, FALSE, TRUE
  )
)
candidates2
```

```
  id  yob experience
1  1 1971       TRUE
2  2 1985      FALSE
3  3 1977      FALSE
4  4 1962       TRUE
```

Merging Data Frames: Example 2

```r
candidates_merged <- merge(
  candidates1,
  candidates2,
  by.x = "candidate_id",
  by.y = "id"
)
```

```r
candidates_merged
```

```
  candidate_id    name party  yob experience
1            1  Sean M  Left 1971       TRUE
2            2  Orla G Right 1985      FALSE
3            3  Rian P  Left 1977      FALSE
4            4 Fiona F Right 1962       TRUE
```

But it is important to keep track which dataset is `x` and which is `y`:

```r
candidates_merged <- merge(
  candidates2,
  candidates1,
  by.x = "id",
  by.y = "candidate_id"
)
```

```r
candidates_merged
```

```
  id  yob experience    name party
1  1 1971       TRUE  Sean M  Left
2  2 1985      FALSE  Orla G Right
3  3 1977      FALSE  Rian P  Left
4  4 1962       TRUE Fiona F Right
```

Next

- Tutorial:
  - Data frames & Plotting
- Next week:
  - Factor variables