Methods Camp 2025: Day 1

Christina Pao, Sofia Avila, Florencia Torche

August 30, 2025

Outline

- [x] Getting Started in R
- [x] Managing Workflow
- [x] R Settings
- [x] Objects in R
- [x] Tidyverse and Data Manipulation
- [ ] **Recoding Variables**
- [ ] Matrices

Any lingering questions before we move on?

Recoding Variables

A few examples of things that social scientists do when recoding variables:

- change a column type (e.g. from character to numeric)
- add labels to factor variables (ie. 1 = male, 2 = female)
- create indicator, or binary, variables
- create categorical variables based on conditions

If you've used another statistical programming language to clean data, these tasks should be familiar to you! We'll work through these together today.

Vectors to data frames

- Every data frame (or tibble in R) can be thought of as a bunch of **vectors** of different **types** stuck together as columns.
- Indeed, one way to **create** a dataframe is to use `bind_cols()` to attach same-length vectors together as columns in a tibble

```r
agevec <- c(18, 21, 23, 20)
gendervec <- c("male", "female", "other", "female")
df = bind_cols(age = agevec, gender = gendervec)
df
```

```
# A tibble: 4 × 2
    age gender
  <dbl> <chr>
1    18 male
2    21 female
3    23 other
4    20 female
```

Vectors to data frames

- We know that elements in a vector need to be of the same type, otherwise, **type coercion** happens

```
# What happened here?
c(18, "20", FALSE)
```

```
[1] "18"    "20"    "FALSE"
```

```
# How about here?
c(1, 2, 3, TRUE, FALSE)
```

```
[1] 1 2 3 1 0
```

- Similarly, every element in a given data frame column must be of the same type.

Vectors to data frames

When working with vectors, we can convert from one type to another using functions like:

- `as.numeric()`
- `as.character()`
- `as.factor()`

```r
agevec
```

```
[1] 18 21 23 20
```

```r
as.character(agevec)
```

```
[1] "18" "21" "23" "20"
```

```r
gendervec
```

```
[1] "male"   "female" "other"  "female"
```

```r
as.numeric(gendervec) # doesnt always make sense to do so, though!
```

```
[1] NA NA NA NA
```

We can do the same for columns in a data frame!

Recoding variables: changing data types

- To change the type of a column in our data, use `mutate()` with `as.numeric()` or `as.character()` commands.
- One common use for this is when two datasets do not merge despite a column that looks similar. This can be because one column is a character while the other is numeric.

```
df2 <- df %>%
  mutate(age = as.character(age))
```

```
left_join(df, df2,
          by = "age") # will not merge because age is numeric in df and character in df2
```

```
df %>%
  mutate(age = as.character(age)) %>%
  left_join(.,df2, by = "age") # will merge because age is character in both
```

Recoding variables: beware of leading zeros!

- It is not always safe to assume that a character column that only has numbers can be safely converted to numeric.
- If you've worked with data that has geographic indicators, you may be familiar with **fips codes**, which uniquely identify geographic areas in the US.
- The length and format of fips codes is very specific: e.g. a state fips code is always two digits: e.g. 06 for California, not just 6. County fips are obtained by concatenating the state fips code with a three-digit county code, e.g. 06001 for Alameda County, California.

```
ces = read_csv('data/ces.csv')
head(ces,1) %>% select(1:9)
```

```
# A tibble: 1 × 9
   year state county     state_fips county_fips zipcode urban_rural gender4   age
  <dbl> <chr> <chr>      <chr>      <chr>         <dbl> <chr>       <chr>   <dbl>
1  2024 CA    Alameda … 06         001           94541 city        man        67
```

Recoding variables: beware of leading zeros!

In the code below, I make the mistake of converting fips codes to numeric!

```
ces_fips = ces %>%
  select(starts_with('state'),starts_with('county')) %>%
  mutate(state_fips = as.numeric(state_fips),
         county_fips = as.numeric(county_fips)) %>%
  mutate(geofips = paste0(state_fips, county_fips)) %>% # wrong fips!
  head(1)

ces_fips
```

```
# A tibble: 1 × 5
  state state_fips county         county_fips geofips
  <chr>      <dbl> <chr>                <dbl> <chr>
1 CA             6 Alameda County           1 61
```

Recoding variables: beware of leading zeros!

To fix it, I need to convert it back to character and add leading zeros using `str_pad()` from the `stringr` package.

```r
ces_fips %>%
  mutate(state_fips = str_pad(as.character(state_fips), width = 2, pad = "0"),
         county_fips = str_pad(as.character(county_fips), width = 3, pad = "0")) %>%
  mutate(geofips = paste0(state_fips, county_fips)) %>% # correct fips!
  head(1)
```

```
# A tibble: 1 × 5
  state state_fips county         county_fips geofips
  <chr> <chr>      <chr>          <chr>       <chr>
1 CA    06         Alameda County 001         06001
```

Recoding variables: factor variables

Often, when we work with categorical survey data, we can use **factor variables**.

Example: convert the variable `us_citizen` to a factor variable and save it in a new object `ces2`.

```
ces2 <- ces %>%
  mutate(us_citizen = factor(us_citizen,
                             levels = c(1,0),
                             labels = c("us_citizen", "not_us_citizen")))

# Check new variable against old variable
str(ces$us_citizen)
```

```
num [1:934] 1 1 1 1 1 1 1 1 1 1 ...
```

```
str(ces2$us_citizen)
```

```
Factor w/ 2 levels "us_citizen","not_us_citizen": 1 1 1 1 1 1 1 1 1 1 ...
```

Creating binary variables

To make a binary variable, you can use the `ifelse()` function.

The syntax is:

> ifelse(condition, value if logical condition is true, value if logical condition is false)

The syntax for creating a new variable uses `mutate()`:

> mutate(new_variable = ifelse(condition based on an existing variable, value of new variable if true, value of new variable if false))

Detour, logical operators

The main logical operators:

1. equals: `==`
2. not: `!`
    - e.g. `!=` or `!is.na(x)`
3. comparison: `<`, `<=`, `>`, `>=`
4. and: `&`
    - e.g. `x > 5 & x < 10`
5. or: `|`
    - e.g. `x < 5 | x > 10`

Creating binary variables

Example: create a new variable called `prime_work_age` that takes a value of 1 if the respondent is in their prime working age–between 25 and 54, and 0 otherwise.

```
ces2 = ces2 %>%
  mutate(prime_work_age = ifelse(age >= 25 & age <= 54, 1, 0),
         .after = age)# put this new column *before* the gender4 column - notice the period

head(ces2) %>% select(year,state,gender4,age,prime_work_age)
```

```
# A tibble: 6 × 5
   year state gender4   age prime_work_age
  <dbl> <chr> <chr>   <dbl>          <dbl>
1  2024 CA    man        67              0
2  2023 CA    woman      48              1
3  2024 CA    man        42              1
4  2023 CA    woman      39              1
5  2023 CA    man        61              0
6  2024 CA    woman      57              0
```

Categorical variables based on many conditions

Example: want to modify the variable `vote_pres_2020` to replace numeric values with their corresponding labels. Below is a mini codebook for the variable:

- 1 = Biden
- 2 = Trump
- 3 = Other
- 4 = Did not vote

Categorical variables based on many conditions

`ifelse()` only allows for two conditions. So we could nest `ifelse()` statements:

```r
ces2 %>%
  mutate(vote_pres_2020 = ifelse(vote_pres_2020 == 1,"biden",
                              ifelse(vote_pres_2020 == 2,'trump',
                              ifelse(vote_pres_2020 == 3,'other',
                              ifelse(vote_pres_2020 == 4,'did_not_vote', NA))))) %>%
  select(1:5,vote_pres_2020) %>%
  head(3)
```

```
# A tibble: 3 × 6
   year state county         state_fips county_fips vote_pres_2020
  <dbl> <chr> <chr>          <chr>      <chr>       <chr>
1  2024 CA    Alameda County 06         001         biden
2  2023 CA    Alameda County 06         001         biden
3  2024 CA    Alameda County 06         001         biden
```

…but you can see how this might get complicated with many conditions!

Categorical variables based on many conditions

Instead, use `case_when()` if there are 2 or more conditions for creating a variable:

```
ces2 %>%
  mutate(vote_pres_2020 = case_when(
    vote_pres_2020 == 1 ~ "biden",
    vote_pres_2020 == 2 ~ "trump",
    vote_pres_2020 == 3 ~ "other",
    vote_pres_2020 == 4 ~ "did_not_vote",
    .default = NA # this is the default value if none of the above conditions are met
  )) %>%
  select(1:5,vote_pres_2020) %>%
  head(3)
```

```
# A tibble: 3 × 6
   year state county         state_fips county_fips vote_pres_2020
  <dbl> <chr> <chr>          <chr>      <chr>       <chr>
1  2024 CA    Alameda County 06         001         biden
2  2023 CA    Alameda County 06         001         biden
3  2024 CA    Alameda County 06         001         biden
```

Review

1. By hand: write out code to create a variable `pacific` set to 1 if the respondent lives in California (CA), Oregon (OR), or Washington (WA), and 0 otherwise.
2. By hand: write out code to create a variable `region` set to “Pacific” if the respondent lives in California (CA), Oregon (OR), or Washington (WA), “Middle Atlantic” if they live in New Jersey (NJ), New York (NY), or Pennsylvania (PA), and “Other” otherwise.
    - Try it using `ifelse()` and then using `case_when()`.
3. What would the following code return?

```
10 %in% 1:9
```

```
(1 + 1 == 4) | (2 + 2 == 4)
```

```
as.numeric(!is.na('a'))
```

```
a <- TRUE
b <- FALSE
c <- FALSE

(a | b) & c
a | (b & c)
a | b & c
```

Review

```r
10 %in% 1:9
```

```
[1] FALSE
```

```r
(1 + 1 == 4) | (2 + 2 == 4)
```

```
[1] TRUE
```

```r
as.numeric(!is.na('a'))
```

```
[1] 1
```

```r
a <- TRUE
b <- FALSE
c <- FALSE

(a | b) & c
```

```
[1] FALSE
```

```r
a | (b & c)
```

```
[1] TRUE
```

```r
a | b & c
```

```
[1] TRUE
```

Questions?

Outline

- [x] Getting Started in R
- [x] Managing Workflow
- [x] R Settings
- [x] Objects in R
- [x] Tidyverse and Data Manipulation
- [x] Recoding Variables
- [ ] Matrices

Working with matrices

- A **matrix** is a rectangular array of numbers.
- It has **rows** and **columns**.
- All elements must be of the **same type** (usually numeric).
- Elements are denoted as a_{ij}, where i is the row number and j is the column number.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Matrix Creation in R

```r
# Create a 2x3 matrix
A <- matrix(1:6, nrow = 2, ncol = 3)
A
```

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

```r
A <- matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE)
A
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Matrix Addition

- Only works for matrices of the **same size**.
- Add corresponding elements.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

```r
A <- matrix(c(1, 3, 2, 4), nrow = 2)
B <- matrix(c(5, 7, 6, 8), nrow = 2)
A + B
```

```
     [,1] [,2]
[1,]    6    8
[2,]   10   12
```

Matrix Subtraction

- Like addition, subtract corresponding elements.

$$\begin{bmatrix} 5 & 4 \\ 3 & 2 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix}$$

```r
A <- matrix(c(5, 3, 4, 2), nrow = 2)
B <- matrix(1, nrow = 2, ncol = 2)
A - B
```

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

Transposing a Matrix

- Usually denoted as $A^\top$.
- Flips rows and columns: $A^\top_{ij} = A_{ji}$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^\top = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

```r
A <- matrix(c(1, 3, 2, 4), nrow = 2)
t(A)
```

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

Vector Dot Product

The **dot product** of two vectors is the sum of the products of their corresponding elements.

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

$$\mathbf{a} \cdot \mathbf{b} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$

Matrix Multiplication (Dot Product)

- Matrix multiplication is just multiple dot products.
- Resulting matrix is the dot product of a row from the first matrix and a column from the second matrix.
  - Element at position (i, j) in the resulting matrix is the dot product of row i from the first matrix and column j from the second matrix.
- Number of columns in first = number of rows in second.

Matrix Multiplication Example

Let:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad (2 \times 3), \quad B = \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} \quad (3 \times 2)$$

Valid multiplication:

$$(2 \times 3) \cdot (3 \times 2) = (2 \times 2)$$

Step-by-Step Multiplication

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad (2 \times 3), \qquad B = \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} \quad (3 \times 2)$$

Compute element-by-element using dot products:

$$AB = \begin{bmatrix} 1 \cdot 7 + 2 \cdot 9 + 3 \cdot 11 & 1 \cdot 8 + 2 \cdot 10 + 3 \cdot 12 \\ 4 \cdot 7 + 5 \cdot 9 + 6 \cdot 11 & 4 \cdot 8 + 5 \cdot 10 + 6 \cdot 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

In R

```
A <- matrix(1:6, nrow = 2, byrow = TRUE)    # 2 x 3
B <- matrix(7:12, nrow = 3, byrow = TRUE)   # 3 x 2

A %*% B
```

```
     [,1] [,2]
[1,]   58   64
[2,]  139  154
```

Matrix Operations Practice

Let the following matrices be:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 3 \\ 1 & 4 \\ 10 & 5 \end{bmatrix}, \quad C = \begin{bmatrix} 2 & -1 \\ 0 & 3 \end{bmatrix}$$

1. Addition: Compute $A + B$
2. Scalar multiplication: Compute $4 \cdot C$
3. Multiplication: Compute $A \cdot C$
4. Transpose: Compute $B^\top$
5. Check your answers in R. Define A and B using `matrix(byrow=F)` and C using `matrix(byrow=T)`.

Matrix Operations Practice

```r
a = matrix(c(1,3,5,2,4,6), nrow = 3, byrow = F)
b = matrix(c(2,1,10,3,4,5), nrow = 3, byrow = F)
c = matrix(c(2,-1,0,3), nrow = 2, byrow = T)
a + b
```

```
     [,1] [,2]
[1,]    3    5
[2,]    4    8
[3,]   15   11
```

```r
4 * c
```

```
     [,1] [,2]
[1,]    8   -4
[2,]    0   12
```

```r
a %*% c
```

```
     [,1] [,2]
[1,]    2    5
[2,]    6    9
[3,]   10   13
```

```r
t(b)
```

```
     [,1] [,2] [,3]
[1,]    2    1   10
[2,]    3    4    5
```

Regression Preview

In SOC 500, you will learn about regression, which is a way to model the relationship between a dependent variable and one or more independent variables. You might have already seen something like:

$$y = X\beta + \varepsilon$$

where y is the dependent variable, X is the matrix of independent variables, β is the vector of coefficients, and ε is the error term.

Regression Preview

In ordinary least squares (OLS) regression, we estimate β using the matrix formula:

$$\hat{\beta} = (X^\top X)^{-1} X^\top y$$

We've now seen almost everything needed to solve a regression:

- matrix multiplication
- transposing
- we're missing one piece: **the inverse**. This is what the `-1` exponent denotes in the formula. We won't go into detail for now, but know that the inverse is kind of like the reciprocal of a number. It asks, "What do I need to multiply this matrix by to get the identity matrix?"

Regression Preview

We can use the tools we've learned to compute $\hat{\beta}$ in R:

```
X <- cbind(1, c(2, 3, 4))  # Add intercept
y <- c(3, 5, 7)
solve(t(X) %*% X) %*% t(X) %*% y
```

```
     [,1]
[1,]   -1
[2,]    2
```

This gives the same result as:

```
lm(y ~ c(2, 3, 4))$coefficients
```

```
(Intercept)  c(2, 3, 4)
         -1           2
```

Summary

- Use `matrix()` to create matrices in R.
- Use `+`, `-`, `t()`, and `%*%` for operations.
- Dimensions must align!
- Matrices are powerful for numerical computation.

Questions?