Data Analysis using R

Data Wrangling

Sven Werenbeck-Ueding

21.09.2023

Source: Wickham and Grolemund (2016)

Prerequisites

Code

```r
# Load the ENE/ENOE data set
tbl_enoe <- read_csv("data/raw/enoe/enoe.csv", skip = 3, na = c("", "N/A"))

tbl_enoe
```

Output

```
## # A tibble: 165,457 × 12
##          id migrate age   munici…¹ fence year  quarter sex   marit…² empl_…³ educ
##       <dbl> <chr>   <chr> <chr>    <chr> <chr> <chr>   <chr> <chr>   <chr>   <chr>
##  1 189889 No      50    2004     0     2004  Q3      Fema… Single  Unempl… 12
##  2 189889 No      50    2004     0     2004  Q4      Fema… Single  Unempl… 12
##  3 189889 No      50    2004     0     2005  Q1      Fema… Single  Unempl… 12
##  4 189890 No      26    2004     0     2005  Q4      Male  Married Full-t… 10
##  5 189890 No      26    2004     0     2006  Q1      Male  Married Full-t… 10
##  6 189890 No      26    2004     0     2006  Q2      Male  <NA>    Full-t… 10
##  7 189891 No      36    2004     0     2006  Q4      Male  Married Full-t… 6
##  8 189891 No      36    2004     0     2007  Q1      Male  Married Full-t… 6
##  9 189891 No      36    2004     0     2007  Q2      Male  Married Full-t… 6
## 10 189894 No      33    2004     1     2010  Q3      NA    Married Full-t… 9
```

`dplyr`

> `dplyr` is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:
>
> - `mutate()` adds new variables that are functions of existing variables
> - `select()` picks variables based on their names.
> - `filter()` picks cases based on their values.
> - `summarise()` reduces multiple values down to a single summary.
> - `arrange()` changes the ordering of the rows.
>
> These all combine naturally with `group_by()` which allows you to perform any operation "by group".
>
> Wickham, François, Henry, and Müller (2022)

Transforming Variables

`mutate()`

```
?dplyr::mutate
```

`mutate()` adds new variables and preserves or deletes existing ones, depending on the function arguments. New variables overwrite existing variables if they have the same name.

`mutate()`

```
?dplyr::mutate
```

Function call and arguments:

```
mutate(.data,
        ...,
        .keep = c("all", "used",
                  "unused", "none"),
        .before = NULL,
        .after = NULL)
```

`.data`

A `data.frame` or `tibble` that should be transformed

mutate()

```
?dplyr::mutate
```

Function call and arguments:

```
mutate(.data,
       ...,
       .keep = c("all", "used",
                 "unused", "none"),
       .before = NULL,
       .after = NULL)
```

...

- Data masking of name-value pairs
- The name specifies the variable name of the newly created column
- Multiple new variables can be created by separating each name-value pair by commas

`mutate()`

```
?dplyr::mutate
```

Function call and arguments:

```
mutate(.data,
       ...,
       .keep = c("all", "used",
                 "unused", "none"),
       .before = NULL,
       .after = NULL)
```

`.keep`

- "all": All columns are kept in the resulting data frame (the default)
- "used": Only columns used to create new variables are kept
- "unused": Only columns that are not used to create new variables are kept
- "none": Only the newly created variables are kept

mutate()

```
?dplyr::mutate
```

Function call and arguments:

```
mutate(.data,
       ...,
       .keep = c("all", "used",
                 "unused", "none"),
       .before = NULL,
       .after = NULL)
```

`.before`/`.after`

- Control, at which position in the data frame the new columns should be placed
- default option `NULL` will add the columns to the RHS

Data Masking

- `dplyr` functions oftentimes use **tidy evaluation**
- tidy evaluation takes on two forms:
  - **data masking**
  - **tidy selection** (we will come to this later in this lecture)
- data masking allows for referring to variables by the name with which they are residing in an environment variable (see `vignette("programming")`)
  - Also referred to as data-variables
  - Data frame variables can be accessed simply by their name instead of the `$` operator

Data Masking

Makes creating new variables (and other operations) more easy and intuitive:

`base` R

```r
tbl_enoe$female <- ifelse(
  tbl_enoe$sex == "Female",
  "Yes",
  "No"
)
```

`dplyr`

```r
mutate(
  tbl_enoe,
  female = ifelse(
    sex == "Female",
    "Yes",
    "No"
  )
)
```

Mutating a Single Variable

Code

```
# Recode sex to a "Yes"/"No" dummy
tbl_enoe %>%
  mutate(female = ifelse(sex == "Female", "Yes", "No"),
         .before = 2)
```

Output

```
## # A tibble: 165,457 × 13
##        id female migrate age   munic…¹ fence year  quarter sex   marit…² empl_…³
##     <dbl> <chr>  <chr>   <chr> <chr>   <chr> <chr> <chr>   <chr> <chr>   <chr>
##  1 189889 Yes    No      50    2004    0     2004  Q3      Fema… Single  Unempl…
##  2 189889 Yes    No      50    2004    0     2004  Q4      Fema… Single  Unempl…
##  3 189889 Yes    No      50    2004    0     2005  Q1      Fema… Single  Unempl…
##  4 189890 No     No      26    2004    0     2005  Q4      Male  Married Full-t…
##  5 189890 No     No      26    2004    0     2006  Q1      Male  Married Full-t…
##  6 189890 No     No      26    2004    0     2006  Q2      Male  <NA>    Full-t…
##  7 189891 No     No      36    2004    0     2006  Q4      Male  Married Full-t…
##  8 189891 No     No      36    2004    0     2007  Q1      Male  Married Full-t…
##  9 189891 No     No      36    2004    0     2007  Q2      Male  Married Full-t…
## 10 189894 No     No      33    2004    1     2010  Q3      NA    Married Full-t…
```

Mutating Multiple Variables

Code

```
# Recode marital status to a two-category variable
tbl_enoe %>%
  mutate(female = ifelse(sex == "Female", "Yes", "No"),
         married = ifelse(marital_status == "Married", "Yes", "No"),
         employed = ifelse(marital_status == "Unemployed", "No", "Yes"),
         .before = 2)
```

Output

```
## # A tibble: 165,457 × 15
##        id female married emplo…¹ migrate age   munic…² fence year  quarter sex
##     <dbl> <chr>  <chr>   <chr>   <chr>   <chr> <chr>   <chr> <chr> <chr>   <chr>
##  1 189889 Yes    No      Yes     No      50    2004    0     2004  Q3      Fema…
##  2 189889 Yes    No      Yes     No      50    2004    0     2004  Q4      Fema…
##  3 189889 Yes    No      Yes     No      50    2004    0     2005  Q1      Fema…
##  4 189890 No     Yes     Yes     No      26    2004    0     2005  Q4      Male
##  5 189890 No     Yes     Yes     No      26    2004    0     2006  Q1      Male
##  6 189890 No     <NA>    <NA>    No      26    2004    0     2006  Q2      Male
##  7 189891 No     Yes     Yes     No      36    2004    0     2006  Q4      Male
##  8 189891 No     Yes     Yes     No      36    2004    0     2007  Q1      Male
```

Tidy Selection

Variable selection via a concise set of helper functions:

Selection helper	Description
`everything()`	Selects all columns in the data frame
`starts_with()`/`ends_with()`/`contains()`	Selects all columns starting/ending/containing a string
`all_of()`/`any_of()`	Selects all/any columns given by a character vector
`where()`	Selects columns based on a condition, e. g. `is.numeric`

→

Can be used in conjunction with `dplyr`'s `across()` function to transform multiple variables at the same time!

Task 1: Modify the ENE/ENOE Data

Task Modify the ENE/ENOE data set for use in our analysis:

- Recode `migrate`, `sex`, `marital_status` and `empl_status` to meaningful 0/1 dummy variables
- Recode numerical variables that are currently stored as character columns to numerics
- Create a column containing the logarithm of `income`
- Recode the municipality column to a factor variable
- Create a period variable from the year and quarter columns

Code

```
tbl_enoe <- tbl_enoe %>%
  mutate(female = ifelse(sex == "Female", "Yes", "No"), # Recode to 0/1 dummies
         married = ifelse(marital_status == "Married", "Yes", "No"),
         employed = ifelse(empl_status == "Unemployed", "No", "Yes"),
         across(all_of(c("migrate", "female", "married", "employed")),
                ~ ifelse(. == "Yes", 1, 0)),
         # Convert all of income, educ, age and year to numerics
```

Selecting Variables

select()

```
?dplyr::select
```

Selects (and renames) variables in a data frame, making use of tidy selection.

select()

```
?dplyr::select
```

Function call and arguments:

```
select(.data,
        ...)
```

`.data`

A `data.frame` or `tibble` from which columns should be selected

Selecting Variables by Names

```
tbl_enoe %>%
  select(id, migrate, marital_status, empl_status, educ, income, female)
```

```
## # A tibble: 165,457 × 7
##        id migrate marital_status empl_status  educ income female
##     <dbl>   <dbl> <chr>          <chr>       <dbl>  <dbl>  <dbl>
##  1 189889       0 Single         Unemployed     12      0      1
##  2 189889       0 Single         Unemployed     12      0      1
##  3 189889       0 Single         Unemployed     12      0      1
##  4 189890       0 Married        Full-time      10     NA      0
##  5 189890       0 Married        Full-time      10     NA      0
##  6 189890       0 <NA>           Full-time      10     NA      0
##  7 189891       0 Married        Full-time       6   3440      0
##  8 189891       0 Married        Full-time       6   3440      0
##  9 189891       0 Married        Full-time       6   3440      0
## 10 189894       0 Married        Full-time       9    559      0
## # … with 165,447 more rows
```

Selecting Variables by Index

```r
tbl_enoe %>%
  select(1, 2, 8:12)
```

```
## # A tibble: 165,457 × 7
##        id migrate sex    marital_status empl_status  educ income
##     <dbl>   <dbl> <chr>  <chr>          <chr>       <dbl>  <dbl>
##  1 189889       0 Female Single         Unemployed     12      0
##  2 189889       0 Female Single         Unemployed     12      0
##  3 189889       0 Female Single         Unemployed     12      0
##  4 189890       0 Male   Married        Full-time      10     NA
##  5 189890       0 Male   Married        Full-time      10     NA
##  6 189890       0 Male   <NA>           Full-time      10     NA
##  7 189891       0 Male   Married        Full-time       6   3440
##  8 189891       0 Male   Married        Full-time       6   3440
##  9 189891       0 Male   Married        Full-time       6   3440
## 10 189894       0 NA     Married        Full-time       9    559
## # … with 165,447 more rows
```

Selecting Variables using Tidy Selection

```
tbl_enoe %>%
  select(all_of(c("female", "employed")), ends_with("income"), where(is.factor))
```

```
## # A tibble: 165,457 × 5
##    female employed income ln_income municipality
##     <dbl>    <dbl>  <dbl>     <dbl> <fct>
##  1      1        0      0   -Inf    2004
##  2      1        0      0   -Inf    2004
##  3      1        0      0   -Inf    2004
##  4      0        1     NA     NA    2004
##  5      0        1     NA     NA    2004
##  6      0        1     NA     NA    2004
##  7      0        1   3440      8.14 2004
##  8      0        1   3440      8.14 2004
##  9      0        1   3440      8.14 2004
## 10      0        1    559      6.33 2004
## # … with 165,447 more rows
```

Selecting and Renaming Variables

```
tbl_enoe %>%
  select(ID = id, empl = employed)
```

```
## # A tibble: 165,457 × 2
##        ID  empl
##     <dbl> <dbl>
##  1 189889     0
##  2 189889     0
##  3 189889     0
##  4 189890     1
##  5 189890     1
##  6 189890     1
##  7 189891     1
##  8 189891     1
##  9 189891     1
## 10 189894     1
## # … with 165,447 more rows
```

Filtering

filter()

```
?dplyr::filter
```

`filter()` subsets data frames and keeps all rows that satisfy one or more specified conditions. Applied on the rows, the condition(s) must produce the logical `TRUE` for the row to be kept in the data frame. If a condition evaluates to `NA`, the corresponding row is dropped.

filter()

```
?dplyr::filter
```

Function call and arguments:

```
filter(.data,
       ...)
```

.data

A data.frame or tibble that should be filtered

`filter()`

```
?dplyr::filter
```

Function call and arguments:

```
filter(.data,
       ...,)
```

`...`

- A data masking expression that returns a logical value and that is defined in terms of the variables in the `.data` argument
- Multiple expressions can be combined by the "and" (`&`) and/or "or" (`|`) operator

Filter by a Single Variable

```r
tbl_enoe %>%
  filter(married == 1)
```

```
## # A tibble: 94,973 × 17
##        id migrate   age munici…¹ fence  year quarter sex   marit…² empl_…³  educ
##     <dbl>   <dbl> <dbl> <fct>    <chr> <dbl>   <dbl> <chr> <chr>   <chr>   <dbl>
##  1 189890       0    26 2004     0      2005       3 Male  Married Full-t…    10
##  2 189890       0    26 2004     0      2006       0 Male  Married Full-t…    10
##  3 189891       0    36 2004     0      2006       3 Male  Married Full-t…     6
##  4 189891       0    36 2004     0      2007       0 Male  Married Full-t…     6
##  5 189891       0    36 2004     0      2007       1 Male  Married Full-t…     6
##  6 189894       0    33 2004     1      2010       2 NA    Married Full-t…     9
##  7 189894       0    33 2004     1      2011       0 Fema… Married Full-t…     9
##  8 189894       0    33 2004     1      2011       1 NA    Married Full-t…     9
##  9 189895       0    32 2004     1      2012       0 Male  Married Full-t…     9
## 10 189895       0    32 2004     1      2012       1 Male  Married Full-t…     9
## # … with 94,963 more rows, 6 more variables: income <dbl>, female <dbl>,
## #   married <dbl>, employed <dbl>, ln_income <dbl>, period <dbl>, and
## #   abbreviated variable names ¹municipality, ²marital_status, ³empl_status
```

Filter by Multiple Variables

```
tbl_enoe %>%
  filter(married == 1, income > 0 & income <= 50000)
```

```
## # A tibble: 51,546 × 17
##        id migrate   age munici…¹ fence  year quarter sex   marit…² empl_…³  educ
##     <dbl>   <dbl> <dbl> <fct>    <chr> <dbl>   <dbl> <chr> <chr>   <chr>   <dbl>
##  1 189891       0    36 2004     0      2006       3 Male  Married Full-t…     6
##  2 189891       0    36 2004     0      2007       0 Male  Married Full-t…     6
##  3 189891       0    36 2004     0      2007       1 Male  Married Full-t…     6
##  4 189894       0    33 2004     1      2010       2 NA    Married Full-t…     9
##  5 189894       0    33 2004     1      2011       0 Fema… Married Full-t…     9
##  6 189894       0    33 2004     1      2011       1 NA    Married Full-t…     9
##  7 189895       0    32 2004     1      2012       0 Male  Married Full-t…     9
##  8 189895       0    32 2004     1      2012       1 Male  Married Full-t…     9
##  9 189895       0    32 2004     1      2012       2 Male  Married Full-t…     9
## 10 189902       0    30 2004     1      2010       2 Male  Married Full-t…     8
## # … with 51,536 more rows, 6 more variables: income <dbl>, female <dbl>,
## #   married <dbl>, employed <dbl>, ln_income <dbl>, period <dbl>, and
## #   abbreviated variable names ¹municipality, ²marital_status, ³empl_status
```

Filter using Tidy Selection

`if_any()` Filter if **any** variable satisfies the condition:

```
tbl_enoe %>%
  filter(if_any(all_of(c("income", "employed")), ~ !is.na(.)))
```

```
## # A tibble: 165,416 × 17
##         id migrate   age munici…¹ fence  year quarter sex   marit…² empl_…³  educ
##      <dbl>   <dbl> <dbl> <fct>    <chr> <dbl>   <dbl> <chr> <chr>   <chr>   <dbl>
##  1 189889        0    50 2004     0      2004       2 Fema… Single  Unempl…    12
##  2 189889        0    50 2004     0      2004       3 Fema… Single  Unempl…    12
##  3 189889        0    50 2004     0      2005       0 Fema… Single  Unempl…    12
##  4 189890        0    26 2004     0      2005       3 Male  Married Full-t…    10
##  5 189890        0    26 2004     0      2006       0 Male  Married Full-t…    10
##  6 189890        0    26 2004     0      2006       1 Male  <NA>    Full-t…    10
##  7 189891        0    36 2004     0      2006       3 Male  Married Full-t…     6
##  8 189891        0    36 2004     0      2007       0 Male  Married Full-t…     6
##  9 189891        0    36 2004     0      2007       1 Male  Married Full-t…     6
## 10 189894        0    33 2004     1      2010       2 NA    Married Full-t…     9
```

`if_all()` Filter if **all** variables satisfy the condition:

Task 2: Filter the ENE/ENOE Data and Keep a Subset of Columns

Task

1. Select all columns that are not of type "character" except `year` and `quarter`.

2. Remove all rows from the resulting data frame that contain any missing values in all columns except of `income` and `ln_income`.

Code

```
tbl_enoe <- tbl_enoe %>%
  select(!where(is.character), -year, -quarter) %>%
  filter(if_all(!ends_with("income"), ~ !is.na(.)))

tbl_enoe
```

Output

```
## # A tibble: 157,248 × 11
##        id migrate   age municipality  educ income female married emplo…¹ ln_in…²
##     <dbl>   <dbl> <dbl> <fct>        <dbl>  <dbl>  <dbl>   <dbl>   <dbl>   <dbl>
```

Arrange Observations

arrange()

```
?dplyr::arrange
```

Orders the rows in a data frame by the values of the columns provided in the function arguments.

→

Useful if observations should be sorted, e. g. in a descriptive summary table!

Arranging Observations by Income

```
tbl_enoe %>%
  arrange(income)
```

```
## # A tibble: 157,248 × 11
##        id migrate   age municipality  educ income female married emplo…¹ ln_in…²
##     <dbl>   <dbl> <dbl> <fct>        <dbl>  <dbl>  <dbl>   <dbl>   <dbl>   <dbl>
##  1 189889       0    50 2004            12      0      1       0       0    -Inf
##  2 189889       0    50 2004            12      0      1       0       0    -Inf
##  3 189889       0    50 2004            12      0      1       0       0    -Inf
##  4 189898       0    28 2004             9      0      1       1       0    -Inf
##  5 189898       0    28 2004             9      0      1       1       0    -Inf
##  6 189898       0    28 2004             9      0      1       1       0    -Inf
##  7 189899       0    27 2004             6      0      1       1       0    -Inf
##  8 189908       0    15 2004             9      0      0       0       0    -Inf
##  9 189909       0    16 2004             9      0      1       0       0    -Inf
## 10 189909       0    16 2004             9      0      1       0       0    -Inf
## # … with 157,238 more rows, 1 more variable: period <dbl>, and abbreviated
## #   variable names ¹employed, ²ln_income
```

Arranging Observations by Sex and Income in Descending Order

```
tbl_enoe %>%
  arrange(female, desc(income))
```

```
## # A tibble: 157,248 × 11
##         id migrate   age municipal…¹  educ income female married emplo…² ln_in…³
##      <dbl>   <dbl> <dbl> <fct>       <dbl>  <dbl>  <dbl>   <dbl>   <dbl>   <dbl>
##  1 5890042       0    60 28022           3 7.50e5      0       1       1    13.5
##  2 5892262       0    41 28027          12 1.72e5      0       1       1    12.1
##  3 5912451       0    55 28033          19 1.67e5      0       1       1    12.0
##  4 5886803       0    33 28022          12 1.29e5      0       1       1    11.8
##  5 5886803       0    33 28022          12 1.29e5      0       1       1    11.8
##  6 5924420       0    42 28040          17 1.29e5      0       1       1    11.8
##  7  376898       0    53 2002           17 1.12e5      0       1       1    11.6
##  8  218665       0    36 2004           12 1.03e5      0       1       1    11.5
##  9  390155       0    53 2002           16 1.00e5      0       1       1    11.5
## 10  390155       0    53 2002           16 1.00e5      0       1       1    11.5
## # … with 157,238 more rows, 1 more variable: period <dbl>, and abbreviated
## #   variable names ¹municipality, ²employed, ³ln_income
```

Grouping and Summarizing Observations

`group_by()`

```
?dplyr::group_by
```

- Groups data based on the specified columns and returns a grouped `tibble`
- Very useful in combination with `summary()` to create grouped summary statistics

`summarize()`

```
?dplyr::summarize
```

- Creates a data frame with one (or more) rows for each combination of grouping variables
- Without grouping variables, the output has one row summarizing all observations
- Output contains one column for each specified summary statistic
- Tidy selection can be used to select columns to summarize

Useful Summary Statistics

Function	Description
`mean()`	Returns the mean of a numerical vector.
`median()`	Returns the median of a numerical vector. See above for handling missing values.
`sd()`	Returns the standard deviation of a numerical vector.
`min()`/`max()`	Returns the minimum/maximum value of a numerical vector.
`n()`	Counts the numbers of observations.

⚠

Be sure to specify `na.rm = TRUE` if the vector has missing values, otherwise `NA` will be returned!

Count Observations in Each Municipality

```
tbl_enoe %>%
  group_by(municipality) %>%
  summarise(n = n())
```

```
## # A tibble: 24 × 2
##    municipality     n
##    <fct>        <int>
##  1 2004         63908
##  2 NA            1135
##  3 2002         35176
##  4 2003          4731
##  5 5025          2623
##  6 5002          2893
##  7 5014           649
##  8 8037         10485
##  9 8005            25
## 10 26055         3021
## # … with 14 more rows
```

Check for Missings in Income for Each Pair of Municipality and Period

```
tbl_enoe %>%
  group_by(municipality, period) %>%
  summarise(na_income = sum(is.na(income)))
```

```
## `summarise()` has grouped output by 'municipality'. You can override using the
## `.groups` argument.

## # A tibble: 782 × 3
## # Groups:   municipality [24]
##    municipality period na_income
##    <fct>         <dbl>     <int>
##  1 2004           0.5         51
##  2 2004           0.75        41
##  3 2004           1           44
##  4 2004           1.25        38
##  5 2004           1.5         38
##  6 2004           1.75        33
##  7 2004           2           33
```

Create Summary Statistics for Dummy Variables

```
tbl_enoe %>%
  summarise(across(c("female", "married", "employed"), ~ mean(.)))
```

```
## # A tibble: 1 × 3
##   female married employed
##    <dbl>   <dbl>    <dbl>
## 1  0.502   0.576    0.623
```

Create Summary Statistics for Continous Variables

```r
tbl_enoe %>%
  summarise(
    across(
      c("income", "age", "educ"),
      .fns = list(mean = ~ mean(., na.rm = T), sd = ~ sd(., na.rm = T))
    )
  )
```

```
## # A tibble: 1 × 6
##   income_mean income_sd age_mean age_sd educ_mean educ_sd
##         <dbl>     <dbl>    <dbl>  <dbl>     <dbl>   <dbl>
## 1       3449.     5430.     35.1   13.6      9.19    4.10
```

Tidy Data

What makes data tidy?

- Data can be represented in many ways:
  - Variables values may be spread over several columns, e. g. one colum for each year
  - Many variables may be stored in one column, e. g. `income` and `age` are stored in the same column `value` and another column specifies which variable the value in `value` corresponds to
  - Observations may be spreach across columns
- Tidy data is an organizational framework that ensure that data is stored in the correct format, i. e. it follows these rules:
  1. Variables stored in separate columns.
  2. Rows uniquely identify observations.
  3. All values are stored in their own cell.

Example for Messy Data

```r
tbl_enoe %>%
  summarise(
    across(
      c("income", "age", "educ"),
      .fns = list(
        mean = ~ mean(., na.rm = T),
        sd = ~ sd(., na.rm = T)
      )
    )
  )
```

```
## # A tibble: 1 × 6
##   income_mean income_sd age_mean age_sd educ_mean educ_sd
##         <dbl>     <dbl>    <dbl>  <dbl>     <dbl>   <dbl>
## 1       3449.     5430.     35.1   13.6      9.19    4.10
```

Example for Messy Data

```
## # A tibble: 1 × 6
##   income_mean income_sd age_mean age_sd educ_mean educ_sd
##         <dbl>     <dbl>    <dbl>  <dbl>     <dbl>   <dbl>
## 1       3449.     5430.     35.1   13.6      9.19    4.10
```

- The observation here corresponds to the column in our data set that is summarized
- The variables are the summary statistics ("mean" and "sd")

⚠

Data is stored in a too wide format!

tidyr

> The goal of `tidyr` is to help you create tidy data. Tidy data is data where:
>
> 1. Every column is variable.
> 2. Every row is an observation.
> 3. Every cell is a single value.
>
> Tidy data describes a standard way of storing data that is used wherever possible throughout the tidyverse. If you ensure that your data is tidy, you’ll spend less time fighting with the tools and more time working on your analysis.
>
> Wickham and Girlich (2022)

pivot_longer

```
?tidyr::pivot_longer
```

`pivot_longer()` "lengthens" data, increasing the number of rows and decreasing the number of columns. The inverse transformation is `pivot_wider()`.

Function call and arguments:

```
pivot_longer(data,
             cols,
             names_to = "name",
             names_sep = NULL,
             values_to = "value")
```

`cols`

Tidy selection of columns to restructure into long format

pivot_longer

```
?tidyr::pivot_longer
```

`pivot_longer()` "lengthens" data, increasing the number of rows and decreasing the number of columns. The inverse transformation is `pivot_wider()`.

Function call and arguments:

```
pivot_longer(data,
             cols,
             names_to = "name",
             names_sep = NULL,
             values_to = "value")
```

`names_to`

- Character vector specifying the new column(s) that are created when pivoting from wide to long format
- If more than one column are created, `names_sep` (or `names_pattern`) has to be specified as well

pivot_longer

```
?tidyr::pivot_longer
```

`pivot_longer()` "lengthens" data, increasing the number of rows and decreasing the number of columns. The inverse transformation is `pivot_wider()`.

Function call and arguments:

```
pivot_longer(data,
             cols,
             names_to = "name",
             names_sep = NULL,
             values_to = "value")
```

`names_sep`

Either a numeric vector that specifies the position to separate the name on or a single string that specifies a regular expression to separate the name

pivot_longer

```
?tidyr::pivot_longer
```

`pivot_longer()` "lengthens" data, increasing the number of rows and decreasing the number of columns. The inverse transformation is `pivot_wider()`.

Function call and arguments:

```
pivot_longer(data,
             cols,
             names_to = "name",
             names_sep = NULL,
             values_to = "value")
```

`values_to`

A character vector of length 1 that specifies the column name in which to store the value

Pivoting Summary Statistics from Wide to Long Format

Code

```
tbl_enoe %>%
  summarise(across(c("income", "age", "educ"),
                   .fns = list(mean = ~ mean(., na.rm = T), sd = ~ sd(., na.rm = T)))) %>%
  pivot_longer(everything(),
               names_to = c("variable", "statistic"),
               names_sep = "_",
               values_to = "value")
```

Output

```
## # A tibble: 6 × 3
##   variable statistic   value
##   <chr>    <chr>       <dbl>
## 1 income   mean      3449.
## 2 income   sd        5430.
## 3 age      mean        35.1
## 4 age      sd          13.6
## 5 educ     mean         9.19
## 6 educ     sd           4.10
```

pivot_wider

```
?tidyr::pivot_wider
```

`pivot_wider()` "widens" data, increasing the number of columns and decreasing the number of rows. The inverse transformation is `pivot_longer()`.

Function call and arguments:

```
pivot_wider(data,
             names_from = name,
             values_from = value)
```

`names_from`

Tidy selection of columns to get the name of the output column

pivot_wider

```
?tidyr::pivot_wider
```

pivot_wider() "widens" data, increasing the number of columns and decreasing the number of rows. The inverse transformation is pivot_longer().

Function call and arguments:

```
pivot_wider(data,
            names_from = name,
            values_from = value)
```

values_from

Tidy selection of columns to get the cell value from

Pivoting from Long to Wide Format

Code

```
tbl_enoe %>%
  summarise(across(c("income", "age", "educ"),
                   .fns = list(mean = ~ mean(., na.rm = T), sd = ~ sd(., na.rm = T)))) %>%
  pivot_longer(everything(),
               names_to = c("variable", "statistic"),
               names_sep = "_",
               values_to = "value") %>%
  pivot_wider(names_from = statistic, values_from = value)
```

Output

```
## # A tibble: 3 × 3
##   variable    mean      sd
##   <chr>      <dbl>   <dbl>
## 1 income   3449.   5430.
## 2 age        35.1    13.6
## 3 educ        9.19    4.10
```

Merging Data

Appending Data Sets

Row-wise

```
?dplyr::bind_rows
```

- Efficient implementation of `base` R's `cbind()` function that takes several data frames or a list of data frames as argument and returns a single row-binded data frame.

- Column names are used for matching the columns of the data frames and if there are columns missing in a data frame, these are filled with `NA`.

- Using the `.id` argument, the names of data frames lists are added in a new column in the resulting data frame.

Column-wise

```
?dplyr::bind_cols
```

- Efficient implementation of `base` R's `r`bind()` function that takes several data frames or a list of data frames as argument and returns a single column-binded data frame

Binding Columns of Summary Statistics

Code

```
# Variables to summarize
sum_stat_vars <- c("income", "age", "educ")

# Create data frame with means for each variable in sum_stat_vars
sum_stat_mean <- tbl_enoe %>%
  summarize(across(all_of(sum_stat_vars), ~ mean(., na.rm = T))) %>%
  pivot_longer(everything(),
               names_to = "variable",
               values_to = "mean")

# Create data frame with standard deviations for each variable in sum_stat_vars
sum_stat_sd <- tbl_enoe %>%
  summarize(across(all_of(sum_stat_vars), ~ sd(., na.rm = T))) %>%
  pivot_longer(everything(),
               names_to = "variable",
               values_to = "sd")

# Bind all columns of sum_stat_mean and the "sd" column of sum_stat_sd
bind_cols(sum_stat_mean, sum_stat_sd[, "sd"])
```

Output

Task 3: Import and Prepare the Fence Construction Data Sets

Task Take a look at the fence construction data sets residing in the directory `data/raw/fence_construction/csv/`. For each year, we have quarterly information on whether border fence construction started in a municipality or not.

Import all data sets at once, making use of the `purrr` packages. Then, bring the data into the correct format (incl. mutating columns as in Task 1) and append individual data sets to one data frame.

Hint: To get a list of all files residing in a directory, you can use the `list.files()` function.

Code

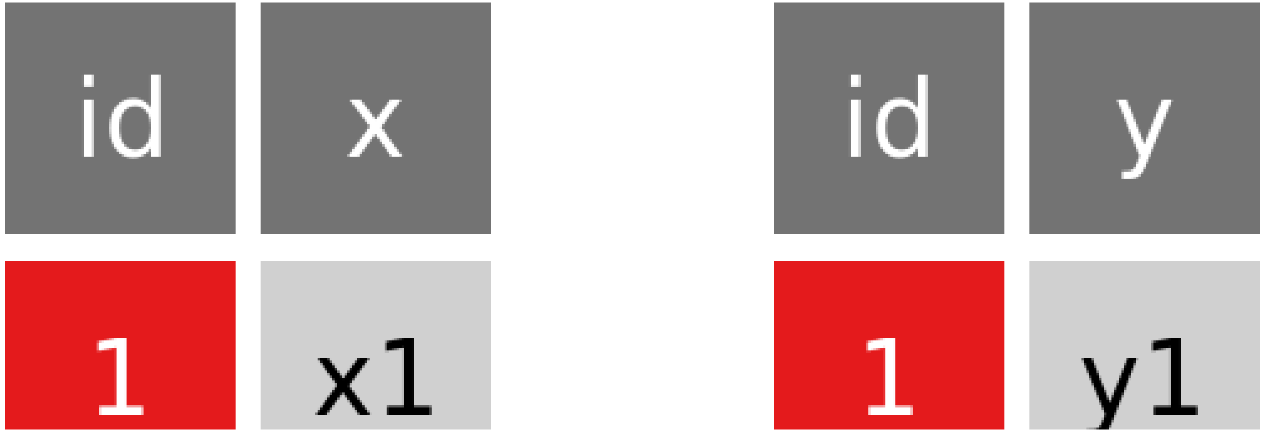
```r
library(purrr)

# List all files in the directory
dir <- "data/raw/fence_construction/csv/"
dir_files <- list.files(dir)

tbl_fence <- paste0(dir, dir_files) %>% # Paste directory path and file names
  set_names(str_remove(dir_files, "\\.csv")) %>% # Set names to file names; remove suffix
  map(read_csv) %>% # Apply read_csv() over all file paths (returning a list of data frames)
```

Joining Data Frames

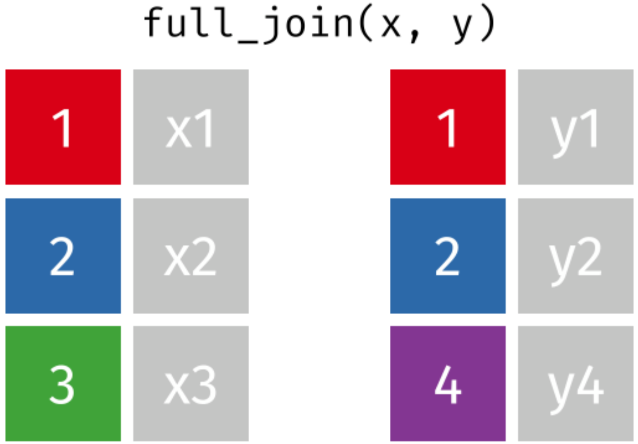
- Combining a pair of data frames is achieved by joining them
- Observations in both data frames are matched by keys
- The data frames on the right have a unique identifier to match observations on (`id`)
- `dplyr` offers several ways to join both data frames by `id` to create a single data frame with both, `x` and `y`, columns



id	x
1	x1

id	y
1	y1

Full Join



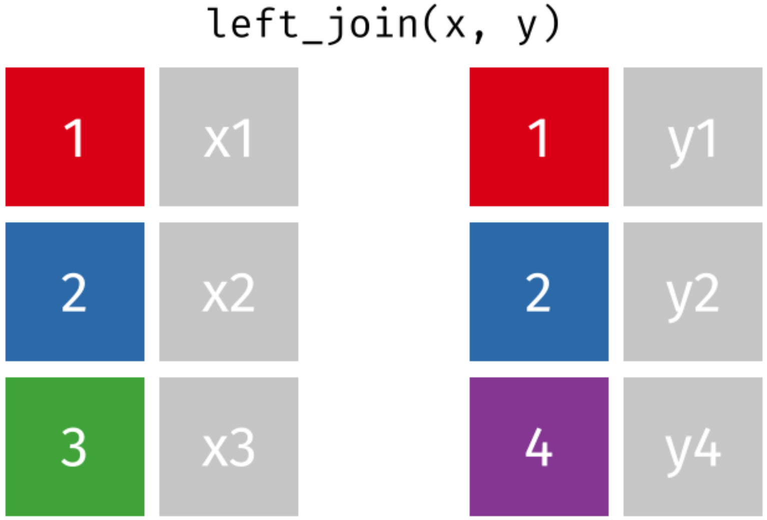
```
full_join(x, y)
```

1	x1
2	x2
3	x3

1	y1
2	y2
4	y4

Aden-Buie (2018)

Left Join



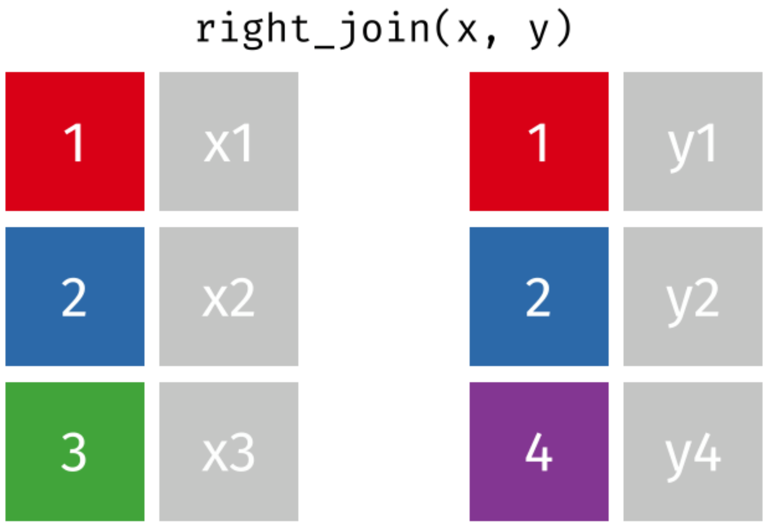
`left_join(x, y)`

1	x1
2	x2
3	x3

1	y1
2	y2
4	y4

Aden-Buie (2018)

Right Join



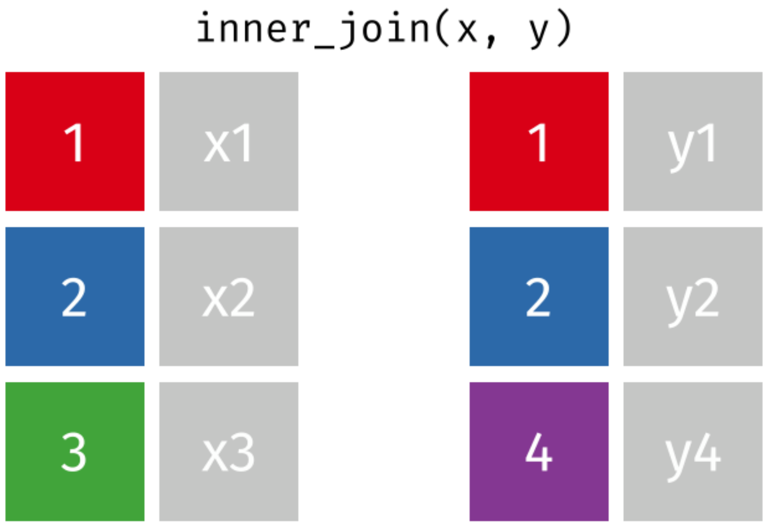
```
right_join(x, y)
```

1	x1
2	x2
3	x3

1	y1
2	y2
4	y4

Aden-Buie (2018)

Inner Join



```
inner_join(x, y)
```

1	x1
2	x2
3	x3

1	y1
2	y2
4	y4

Aden-Buie (2018)

Task 4: Join ENE/ENOE Data and Fence Construction Data

Task In order for us to be able to analyze the effect of border fence construction on the migration of Mexicans to the US, we need to combine information on individuals and information on when a fence was constructed. Fortunately, we have data for both.

Join the ENE/ENOE data frame with the fence construction data frame using appropriate keys to join observations on. Export the resulting data frame to the directory `data/processed/` as csv.

Code

```r
# Ensure consistent data type of key variables
tbl_fence <- tbl_fence %>%
  mutate(municipality = forcats::as_factor(municipality))

# Inner join omits observations for which no information on fence construction is given.
# For the analysis, these observations are not needed.
tbl_out <- inner_join(tbl_enoe, tbl_fence, by = c("municipality", "period"))

write_csv(tbl_out, "data/processed/fence_migration.csv")

tbl_out
```

References

Aden-Buie, G. (2018). *tidyexplain. Tidy Animated Verbs*. URL: https://www.garrickadenbuie.com/project/tidyexplain/ (visited on Jan. 01, 2023).

Wickham, H., R. François, L. Henry, et al. (2022). *dplyr: A Grammar of Data Manipulation*. https://dplyr.tidyverse.org, https://github.com/tidyverse/dplyr.

Wickham, H. and M. Girlich (2022). *tidyr: Tidy Messy Data*. https://tidyr.tidyverse.org, https://github.com/tidyverse/tidyr.

Wickham, H. and G. Grolemund (2016). *R for data science. import, tidy, transform, visualize, and model data*. O'Reilly. URL: https://r4ds.had.co.nz/.