Loading and writing CSV files

DATA MANIPULATION IN JULIA

Katerina Zahradova
Instructor

datacamp

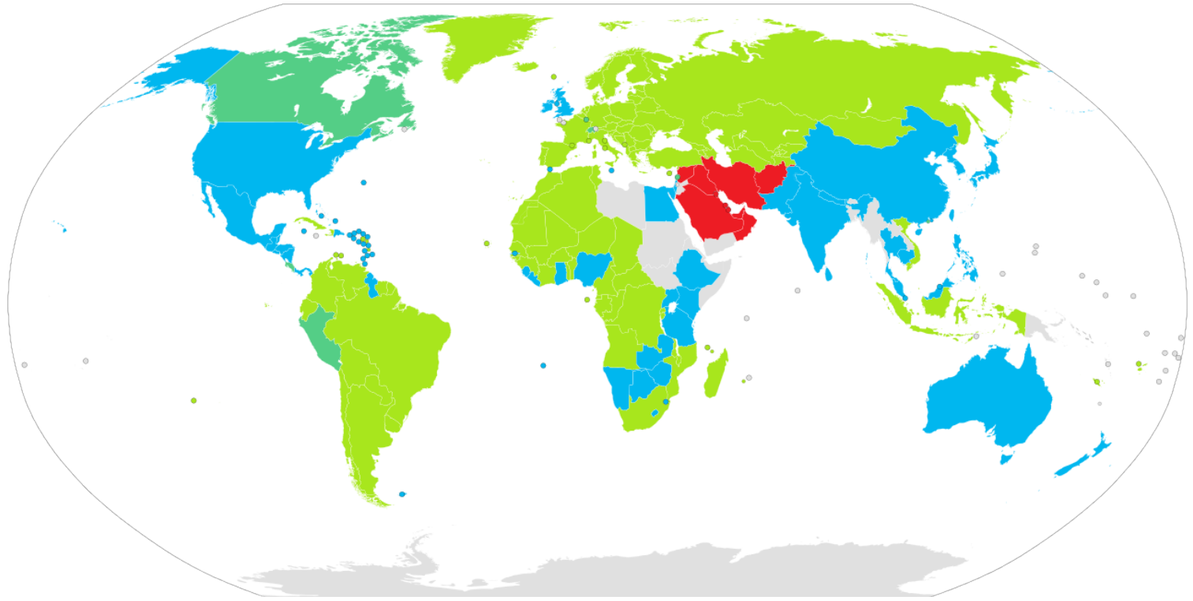
Delimiters

- Delimiter = a character or a string used to separate values
- Examples of a delimiter include, e.g., `","` , `" "` , `"\t"` , ...

```
# Loading file with a space as a delimiter
penguins = DataFrame(CSV.File("penguins.csv",delim=" "))
```

Decimal mark

- Decimal point (blue), e.g., `3.14`
- Decimal comma (green), e.g., `3,14`
- Both (dark green)
- Arabic decimal separator (red)



[1] By NuclearVacuum, Wikipedia

```
# Loading file with comma as decimal mark
penguins = DataFrame(CSV.File(
        "penguins.csv",
        decimal=',', delim=" "))
```

Loading parts of datasets

```
# Loading lines 13 till 27
penguins_part = DataFrame(CSV.File("penguins.csv", skipto=10, limit=3))
```

```
3×7 DataFrame
Row species  island     culmen_length_mm  culmen_depth_mm  ...
    String7  String15   Float64           Float64          ...

──────────────────────────────────────────────────────────────
1   Adelie   Torgersen  38.6              21.2             ...
2   Adelie   Torgersen  34.6              21.1             ...
3   Adelie   Torgersen  36.6              17.8             ...
```

Header

```
# Specifying header as a line
penguins_header = DataFrame(CSV.File("penguins.csv", header = 1))
```

```
333×7 DataFrame
Row species  island     culmen_length_mm  culmen_depth_mm  ...
    String7  String15   Float64           Float64          ...

_____________________________________________________________
1   Adelie   Torgersen  39.1              18.7             ...
2   Adelie   Torgersen  39.5              17.4             ...
3   Adelie   Torgersen  40.3              18.0             ...
...
```

Header over multiple lines

```
# Multiline header
penguins_header = DataFrame(CSV.File("penguins.csv", header = [1, 2]))
```

```
332×7 DataFrame
Row species_Adelie  island_Torgersen  culmen_length_mm_39.1  ...
    String7         String15          Float64                ...
──────────────────────────────────────────────────────────────────
1   Adelie          Torgersen         39.5                   ...
2   Adelie          Torgersen         40.3                   ...
3   Adelie          Torgersen         36.7                   ...
...
```

Replacing the header

```
# Replacing header
penguins_header = DataFrame(CSV.File("penguins.csv",
                        header = [:species, :area, :culmen_l_mm, :culmen_d_mm,
                            :flipper_l_mm, :weight_g, :sex]))
```

```
333×7 DataFrame
Row species  area       culmen_l_mm  culmen_d_mm      ...
    String7  String15   Float64      Float64          ...

-----------------------------------------------------------
1   Adelie   Torgersen  39.1         18.7             ...
2   Adelie   Torgersen  39.5         17.4             ...
3   Adelie   Torgersen  40.3         18.0             ...
...
```

Writing CSV files

```
# Save DataFrame
CSV.write("temp/transformed_penguins.csv", delim = " ", decimal = ',')
```

Cheat sheet

- `delim=` : a `Char` or `String` separating values in columns; e.g., `species,island,...`
- `decimal=` : a `Char` indicating how decimal places are separated in floats; e.g., `.` in `3.14`
- `skipto=` : an `Int` specifying the row number in the file where you want to start loading; beware - header is included!
- `limit=` : an `Int` specifying the number of rows you want to load
- `header=` : an `Int` for row number of a header, a `Vector{Int}` for multiple lines, a `Vector{String}` or `Vector{Symbol}` to rewrite header
- `CSV.File(path)` loads a file in `path`
- `CSV.write(path, df)` writes `df` as a CSV in `path`

Documentation for **`CSV.File()`** and **`CSV.write`**

Let's practice!

DATA MANIPULATION IN JULIA

Joining data

DATA MANIPULATION IN JULIA

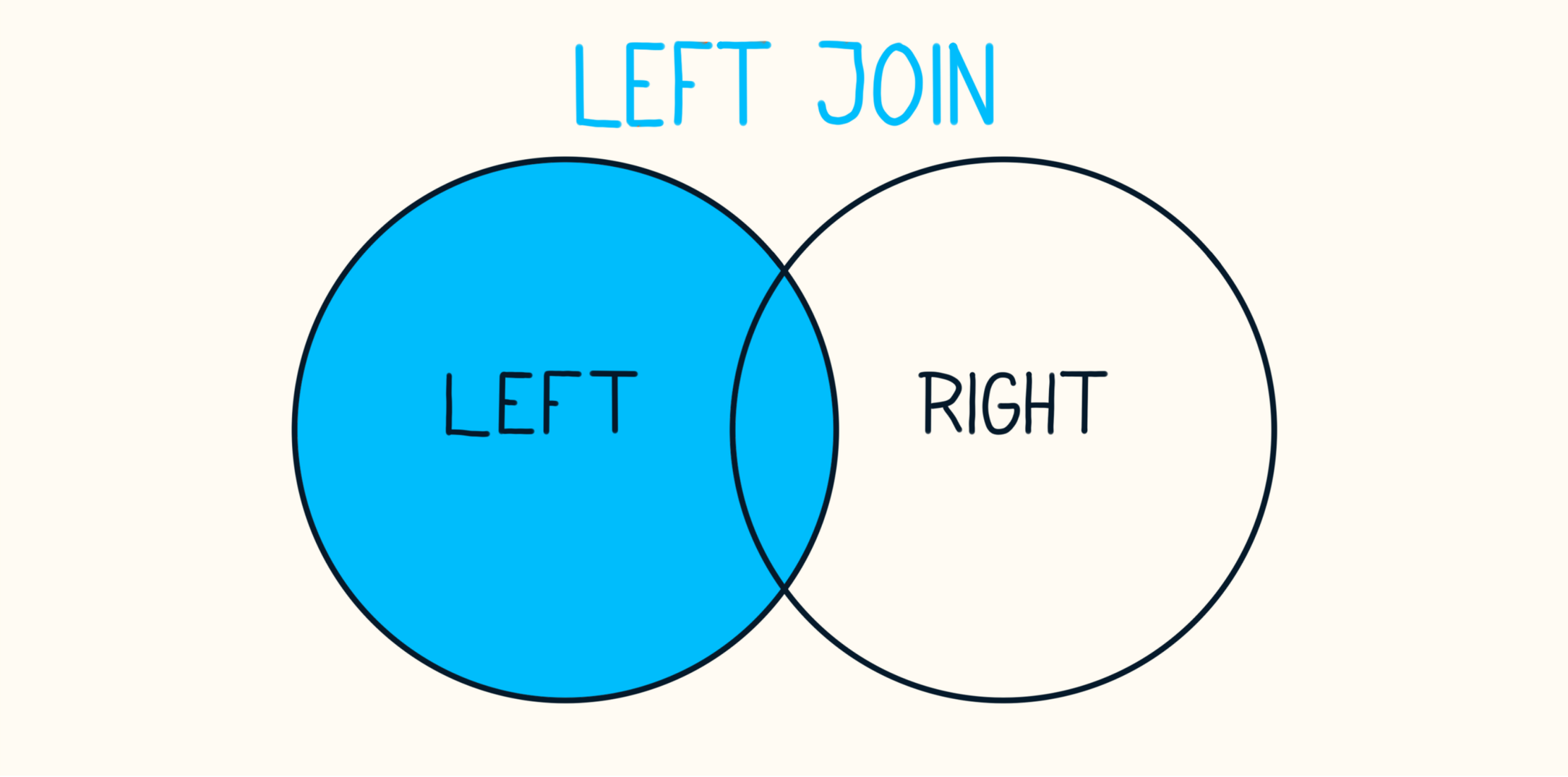
Katerina Zahradova
Instructor

datacamp

Why we join

- More information not already included in the dataset
- Datasets from different sources
- ...

Left join



Left join in practice

More info about joins here

```
# Left join on chocolates and chocolate_companies, using company column
leftjoin(chocolates, chocolate_companies, on = :company)
```

[1] https://dataframes.juliadata.org/stable/man/joins/

Joining on columns with different names

```
# Left join on chocolates and chocolate_companies, using company and company_name
leftjoin(chocolates, chocolate_companies, on = :company => :company_name)
```

Joining on multiple columns

```
# Left join on chocolates and chocolate_companies
# using company and company_location columns
leftjoin(chocolates, chocolate_companies,
                 on = [:company => :company, :company_location => :company_location])
```

Cheat sheet

`leftjoin(left, right, on = :col)`:

- returns all rows and columns from `left`, along with those rows of `right` that have a matching value in `col` with `left`

`leftjoin(left, right, on = :col_left => :col_right)`:

- left join when the columns don't have the same name

`leftjoin(left, right, on = [:c1_l => :c1_r, ...])`:

- left join on multiple columns

Let's practice!

DATA MANIPULATION IN JULIA

Handling missing values

DATA MANIPULATION IN JULIA

Katerina Zahradova
Instructor

datacamp

Finding missing values

```
# Check to see if there are missing values
describe(penguins, :nmissing)
```

```
7×2 DataFrame
Row variable           nmissing
    Symbol             Int64
-------------------------------
1   species            0
2   island             5
3   culmen_length_mm   0
4   culmen_depth_mm    0
5   flipper_length_mm  0
6   body_mass_g        23
7   sex                0
```

ismissing()

```
# Find rows with missing values
penguins[ismissing.(penguins.island),:]
```

```
5x7 DataFrame
Row species   island    culmen_length_mm  culmen_depth_mm  ...
    String15  String15  Float64           Float64          ...

_____________________________________________________________
1   Adelie    missing   39.5              17.4             ...
2   Adelie    missing   40.3              18.0             ...
3   Chinstrip missing   46.7              18.3             ...
4   Gentoo    missing   49.3              13.6             ...
5   Gentoo    missing   43.9              17.8             ...
```

ismissing()

```
# Find rows with missing values
penguins[ismissing.(penguins.island), :species, :sex]
```

```
5x3 DataFrame
Row species   island    sex
    String15  String15  String7

_______________________________
1   Adelie    missing   MALE
2   Adelie    missing   FEMALE
3   Chinstrip missing   MALE
4   Gentoo    missing   MALE
5   Gentoo    missing   FEMALE
```

dropmissing()

```
# Drop all missing values
dropmissing!(penguins)

describe(penguins)
```

```
7×2 DataFrame
Row variable           nmissing
    Symbol             Int64
1   species            0
2   island             0
3   culmen_length_mm   0
4   culmen_depth_mm    0
5   flipper_length_mm  0
6   body_mass_g        0
7   sex                0
```

```
# Drop missing values in island column
dropmissing!(penguins, :island)

describe(penguins)
```

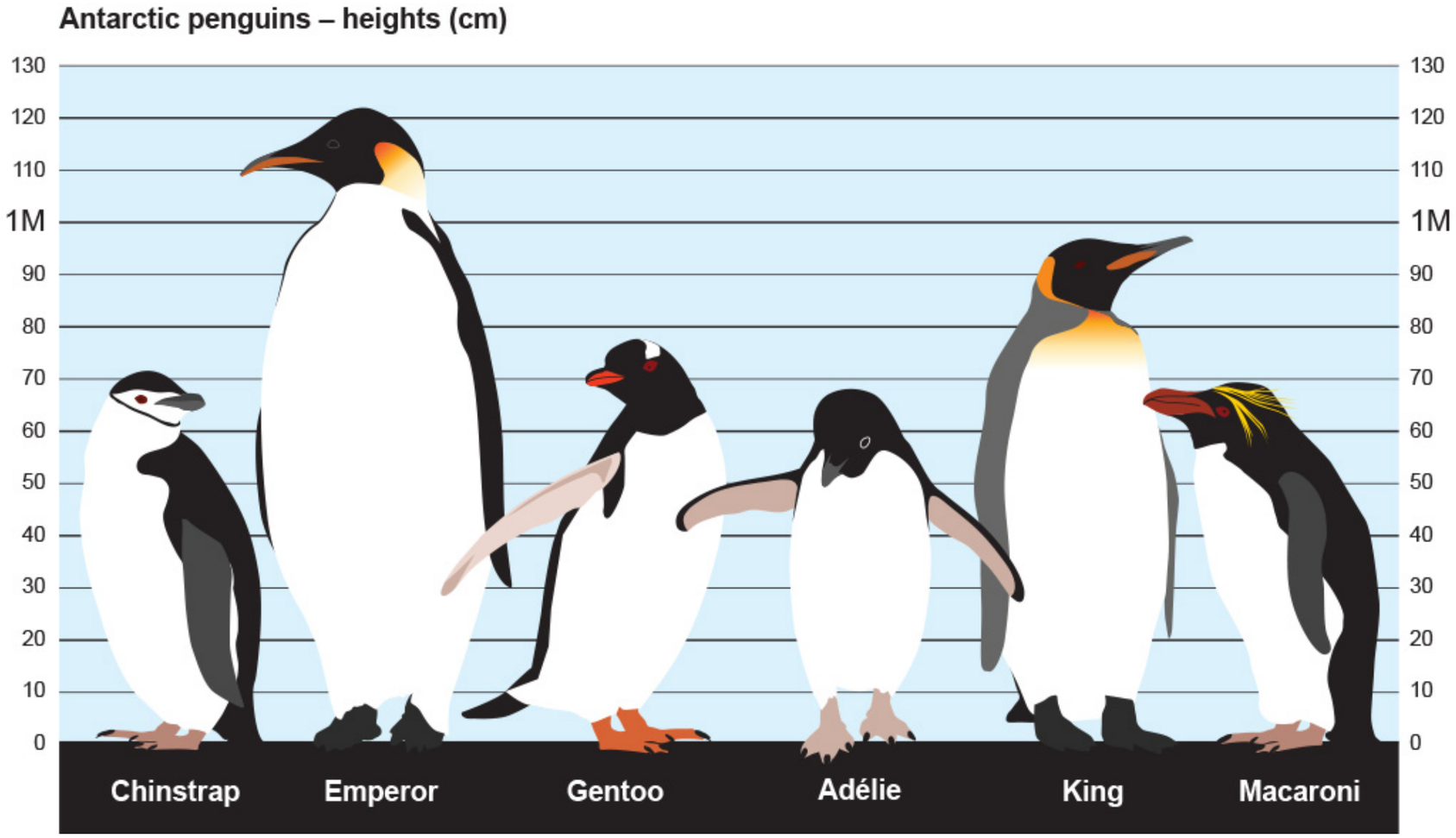
```
7×2 DataFrame
Row variable           nmissing
    Symbol             Int64
1   species            0
2   island             0
3   culmen_length_mm   0
4   culmen_depth_mm    0
5   flipper_length_mm  0
6   body_mass_g        23
7   sex                0
```

replace()

```
# Replace missing by a value
replace!(penguins.body_mass_g, missing => 0)
```

```
# Replace missing by mean
replace!(penguins.body_mass_g, missing => mean(skipmissing(penguins.body_mass_g)))
```

Replacing with grouped summary statistics



[1] Image courtesy www.bas.ac.uk/about/antarctica/wildlife/penguins/

Replacing using groupby()

```
# Iterate over groups and replace by rounded mean for each group
for group in groupby(penguins, :species)
    group[ismissing.(group.body_mass_g), :body_mass_g] .= round(mean(skipmissing(group.body_mass_g)))
end

# Check missing values
describe(penguins, :nmissing)
```

```
7×2 DataFrame
Row variable           nmissing
    Symbol             Int64
1   species            0
...
6   body_mass_g        0
7   sex                0
```

Replacing using multiple columns

```
# Iterate over more groups and replace by rounded mean for each group
for group in groupby(penguins, [:species, :sex])
    group[ismissing.(group.body_mass_g), :body_mass_g] .= round(mean(skipmissing(group.body_mass_g)))
end
```

Insufficient data

```
# What happens if there are no records in the group
for group in groupby(penguins, [:species, :sex, :flipper_length_mm, :culmen_length_mm])
    group[ismissing.(group.body_mass_g), :body_mass_g] .= round(mean(skipmissing(group.body_mass_g)))
end
```

```
ArgumentError: median of an empty array is undefined, Any[]
```

Cheat sheet - find and drop missing values

- `ismissing(var)` : returns `true` if `var = missing` , `false` otherwise
- `ismissing.(df.col)` : returns a vector of `true` / `false` values
- `df[ismissing.(df.col),:]` : returns those rows of `df` where the value in `col` is `missing`
- `dropmissing(df)` : drops all rows that contain `missing`
- `dropmissing!(df, :col)` : drops all rows that contain `missing` in `col` ; rewrites `df`

Cheat sheet - replace missing values

- `replace!(df.col, missing => mean(skipmissing(df.col)))` : replaces `missing` values in `col` with the mean of `col` (calculated by skipping those missing values)
- To replace `missing` in individual groups

```
for group in groupby(df, :col)
  group[ismissing.(group.col),:col] = value  # or by mean of the group
end
```

Let's practice!

DATA MANIPULATION IN JULIA

Efficient workflow

DATA MANIPULATION IN JULIA

Katerina Zahradova
Instructor

datacamp

Tips for names

- Short, meaningful names
  - `wages` rather than `df`
  - `wages` rather than `us_min_wages_data_between_1968_and_2020_with_inflation_adjusted_column`
- Follow naming conventions/patterns
  - mixing `state_wage_2020` and `effective.2020.dollars` can be hard to remember
  - same with capitals, avoid `state`, `Year`, and `REGION` in the same DataFrame

Too many variables

- Don't create too many new variables
  - clutters memory
  - chaos: what is the difference between `wages_no_missing`, `wages_missing_state_only`, `wages_original_no_missing`, `wages_state_mean_no_missing`, etc.
- Overwrite! Use `select!()`, `transform!()`, etc.
- Use `chain` macros to reduce the need for new versions of the same data

Variables instead of hard coding

- Variables over hard coding values

```
# Rather
replace_missing = 0

replace!(df.col1, missing => replace_missing)
replace!(df.col2, missing => replace_missing)

# Than
replace!(df.col1, missing => 0)
replace!(df.col2, missing => 0)
```

Make a function of it

- Write a function rather than write code over and over and over again!
  - functions prevents typos
  - once set up, they are quicker to use

```
# Function to plot multiple lineplots with labels
function make_line_plot(xs, ys,labels; xlabel="", ylabel="", title="")
    p = plot(title = title, xlabel = xlabel, ylabel = ylabel)
    for (x, y, label) in zip(xs, ys, labels)
        plot!(x, y, label=label)
    end
    p
end
```

Comment and document

- Comments for what we are doing

```
# Standardize names
rename!(df, :ColumnOne => :col_1)

# Lines with missing company
df[ismissing.(df.company),:]

# Pivoting on year and state
unstack(wages, :year, :state, :eff_min_wage)
```

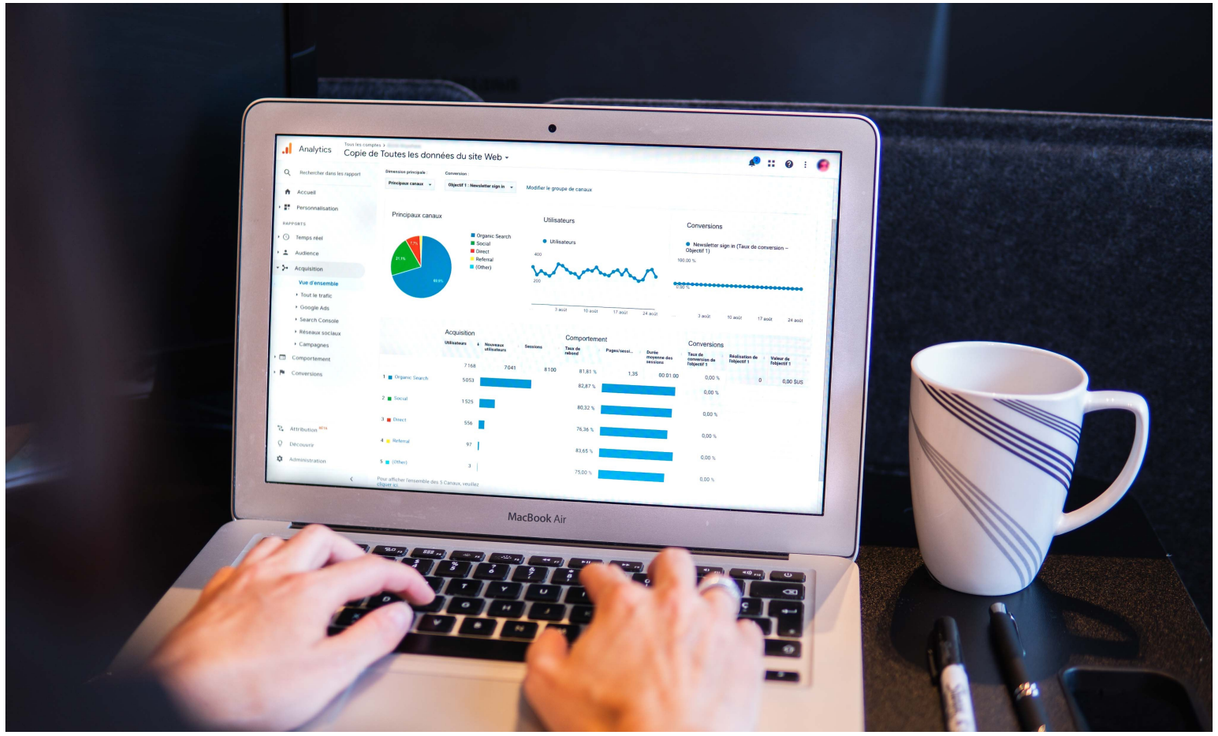
- Document why we are doing things

```
# Replace missing wages by minimum
# As the worst case
min = minimum(skipmissing(df.wages))
replace!(df.wages, missing => min)

# Joining with countries
# To study how countries influence quality
leftjoin(company, countries, on=:location)
```

Get to know the data

- Take the time to understand the data
  - Easier to extract information later
  - Make plots, print the results, ...



[1] Photo by Myriam Jessier on Unsplash

Ask for help!

- Don't reinvent the wheel, use resources available
  - Google
  - Stack Overflow
  - DataCamp Cheat Sheets
  - ...

Have fun!

- Have fun, don't give up, and enjoy!

Flight delays in US airports

airlines.csv
info about airlines
e.g., airline codes

airports.csv
info about airports
e.g., location

flights.csv
info about flights
e.g., delays

Let's practice!

DATA MANIPULATION IN JULIA

Wrap-up

DATA MANIPULATION IN JULIA

Katerina Zahradova
Instructor

datacamp

What you've learned

- Load and save CSV files
- Left join
- `describe()` to check the DataFrame
- `select()` to select, drop, move, and rename columns
- How to apply functions to columns and how to create new columns
- `groupby()` and grouped summary statistics
- Pivot tables with `ustack()`
- `chain` macro so simplify workflow
- Visualizations

... and more!

Next steps

Theoretical courses:

- Communicating Data Insights
- GitHub Concepts

Or learn SQL, Python, and more!

Practice your skills:

- Kaggle
- DataCamp competitions
- Do personal projects

Congratulations!

DATA MANIPULATION IN JULIA