Exploring grouped data

DATA MANIPULATION IN JULIA

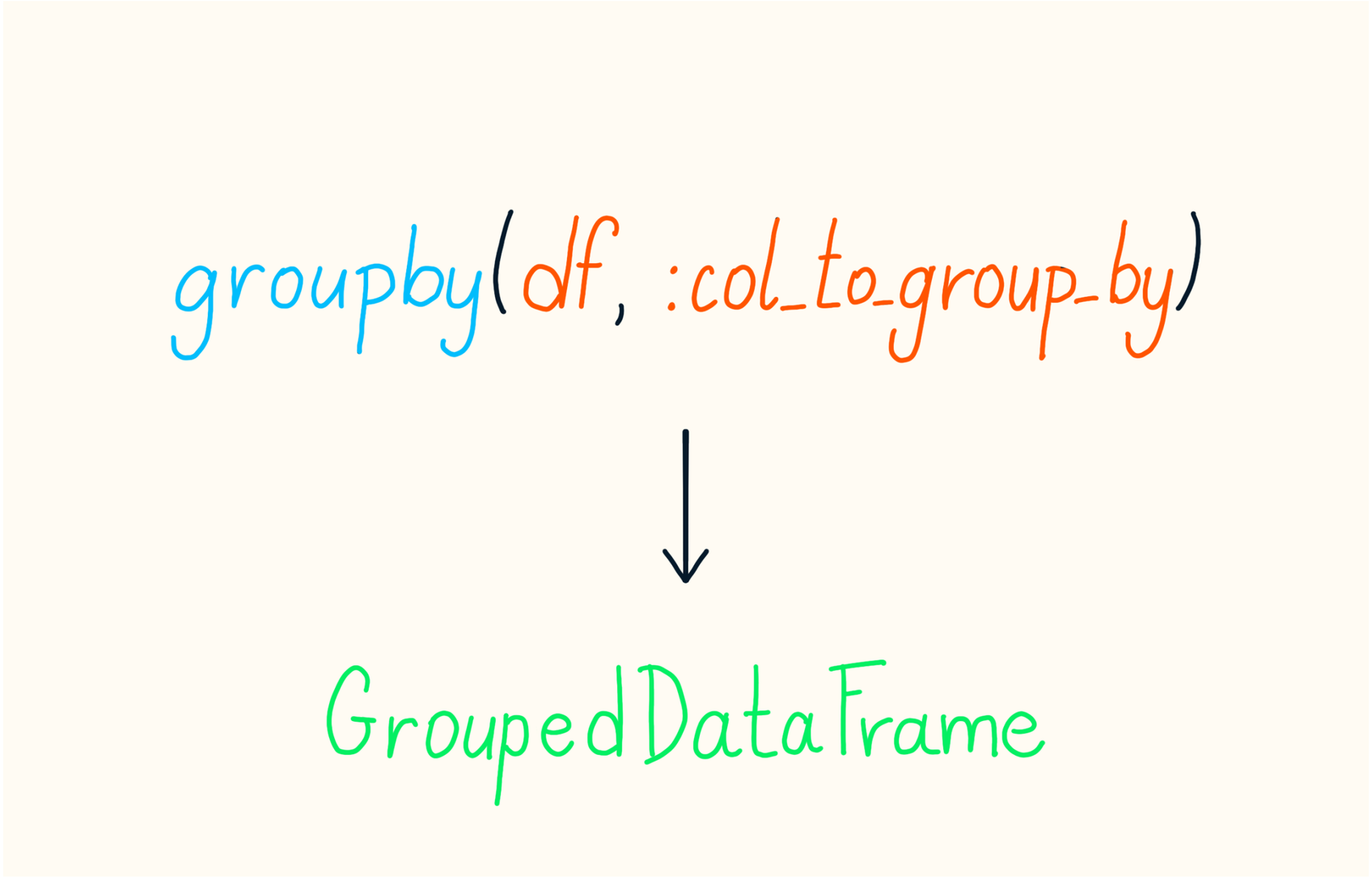
Katerina Zahradova
Instructor

datacamp

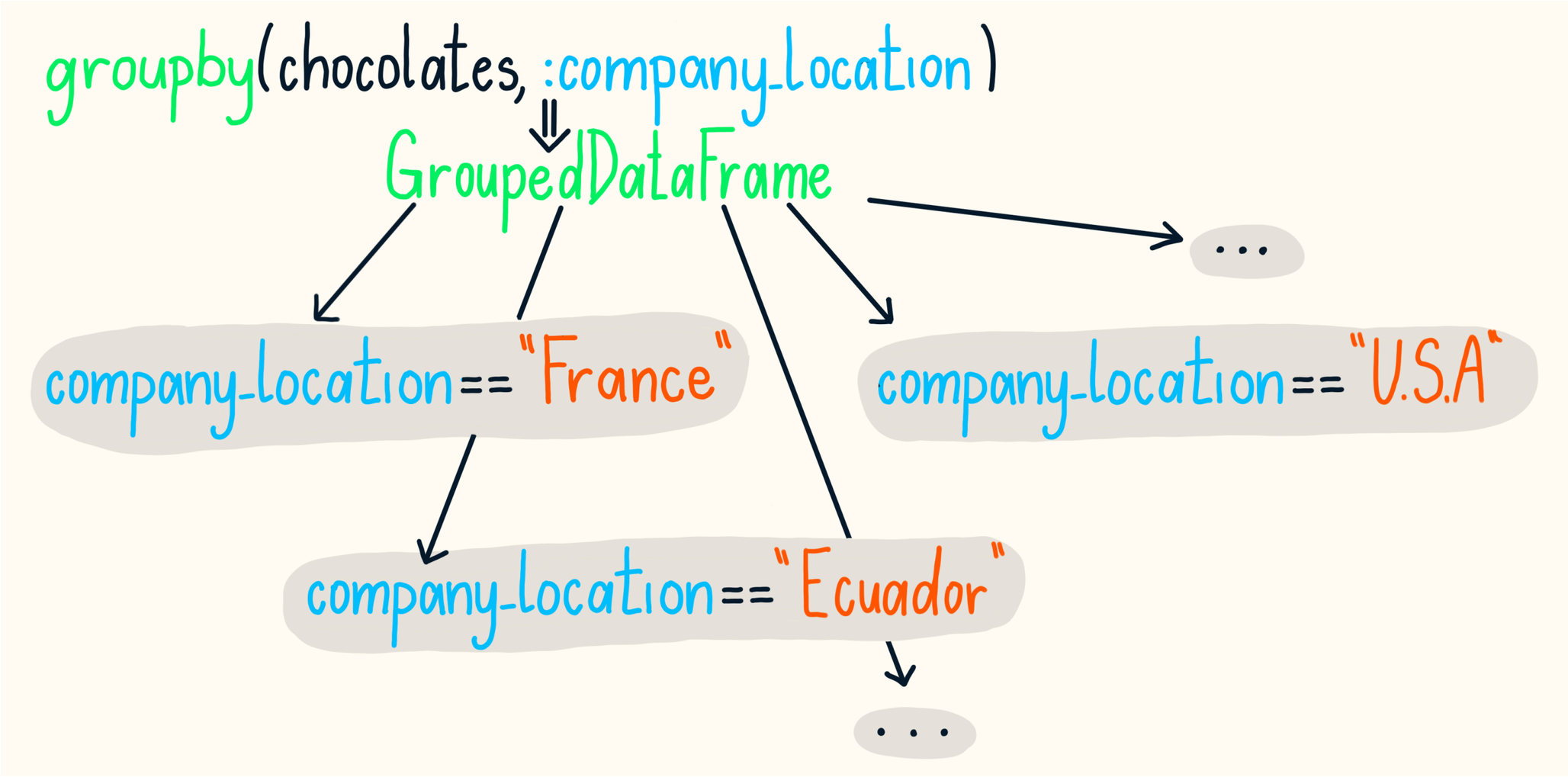
Filtering for groups

```
# Companies in different countries
choc_comp_france = filter(chocolates -> chocolates.company_location == "France", chocolates)

choc_comp_brazil = filter(chocolates -> chocolates.company_location == "Brazil", chocolates)

choc_comp_peru = filter(chocolates -> chocolates.company_location == "Peru", chocolates)

choc_comp_belgium = filter(chocolates -> chocolates.company_location == "Belgium", chocolates)

...
```

groupby() to the rescue!



GroupedDataFrames



groupby() in action

```
# Group by company_location
groupby(chocolates, :company_location)
```

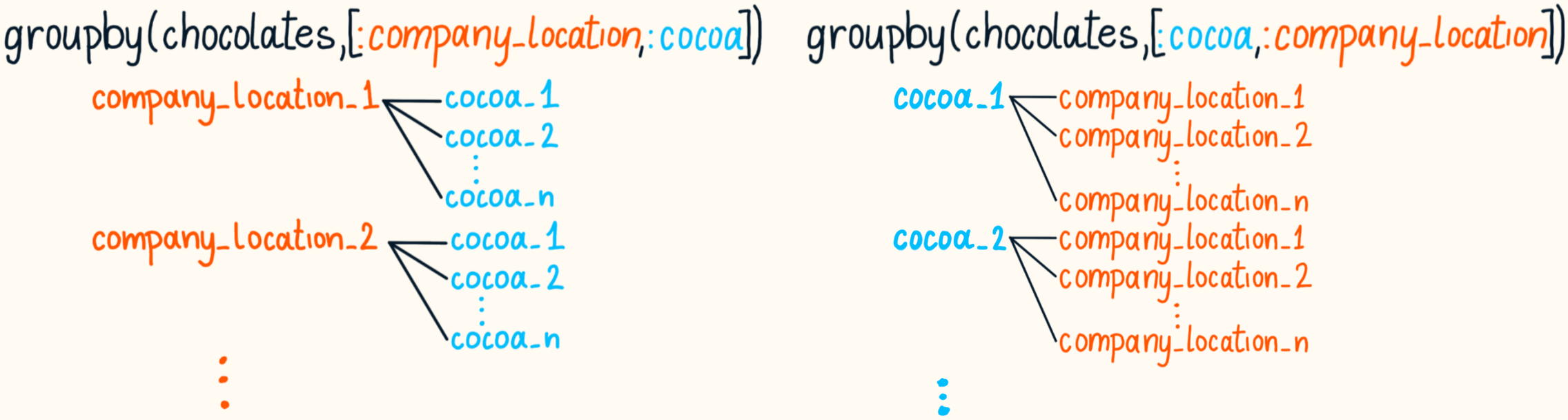
```
GroupedDataFrame with 60 groups based on key: company_location
First Group (156 rows): company_location = "France"
Row  company   bean_origin  ...
     String    String       ...

------------------------------
1    A. Morin  Agua Grande  ...
...
Last Group(4 rows): company_location = "Ireland"
Row  company           bean_origin  ...
     String            String       ...

------------------------------
1    Wilkie's Organic  Amazonas     ...
```

groupby() on multiple columns

```
groupby(chocolates, [:company_location, :cocoa])
```

```
GroupedDataFrame with 373 groups based on keys: company_location, cocoa
First Group (5 rows): company_location = "France", cocoa = 63.0
Row  company   bean_origin  ...
     String    String       ...

_____________________________
1    A. Morin  Agua Grande  ...
...
Last Group (1 row): company_location = "Ireland", cocoa = 89.0
Row  company          bean_origin  ...
     String           String       ...

_____________________________
```

Order matters



Number of records

```
# Group by country
chocolates_by_country = groupby(chocolates, :company_location)
# Number of records per group
combine(chocolates_by_country, nrow => :count)
```

```
60x2 DataFrame
Row  company_location  nrow
     String31          Int64

------------------------------
1    France            156
2    U.S.A.            764
3    Fiji              4
...
```

Sort by number of records

```
# Group by country
chocolates_by_country = groupby(chocolates, :company_location)
# Sort the number of records
sort(combine(chocolates_by_country, nrow => :count), :count, rev = true)
```

```
60x2 DataFrame
Row  company_location  nrow
     String31          Int64

-----------------------------
1    U.S.A             764
2    France            156
3    Canada            125
...
```

unique() on a vector

```
# Unique elements
unique(chocolates.company_location)
```

```
60-element Vector{String31}:
 "France"
 "U.S.A."
 "Fiji"
 "Ecuador"
 "Mexico"
 "Switzerland"
 "Netherlands"
...
```

unique() on a DataFrame

```
# Create a new DF containing only unique rows
unique(chocolates)
```

```
1795×10 DataFrame
Row company  bean_origin  ...
    String   String       ...

_______________________________
1   A. Morin Agua Grande  ...
...
```

unique() with specified columns

```
# Unique company records
unique(chocolates, :company)
```

```
416×10 DataFrame
Row company  bean_origin  ...
    String   String       ...
─────────────────────────────
1   A. Morin Agua Grande  ...
...
```

```
# Unique company AND cocoa contents
unique(chocolates, [:company, :cocoa])
```

```
968×10 DataFrame
Row company  bean_origin  ...
    String   String       ...
─────────────────────────────
1   A. Morin Agua Grande  ...
...
```

Let's practice!

DATA MANIPULATION IN JULIA

Grouped summary statistics

DATA MANIPULATION IN JULIA

Katerina Zahradova
Instructor

datacamp

What we know now

```
# Calculate mean minimum wage
combine(wages, :effective_min_wage_2020_dollars => mean)
```

```
1x1 DataFrame
Row | effective_min_wage_2020_dollars_mean
    | Float64
____|_____________________________________
   1| 8.37093
```

What we know know

```
# Filter and calculate mean
first(combine(filter(r -> r.region =="W", wages), :effective_min_wage_2020_dollars => mean))
first(combine(filter(r -> r.region =="S", wages), :effective_min_wage_2020_dollars => mean))
first(combine(filter(r -> r.region =="NE", wages), :effective_min_wage_2020_dollars => mean))
```

```
DataFrameRow (1 columns)
Row | effective_min_wage_2020_dollars_mean
    | Float64
____|_____________________
   1| 8.75413
...
```

Using combine() and groupby()

```
# Group by region
wages_by_region = groupby(wages, :region)
# Compute average per group
combine(wages_by_region, :effective_min_wage_2020_dollars => mean)
```

```
4x2 DataFrame
Row region    effective_min_wage_2020_dollars_mean
    String    Float64

--------------------------
1    S        8.15458
2    W        8.59119
3    NE       8.75413
4    MW       8.1514
```

Combining combine() and groupby()

```
# Combine them together
combine(groupby(wages, :region), :effective_min_wage_2020_dollars => mean)
```

```
4x2 DataFrame
Row region    effective_min_wage_2020_dollars_mean
    String    Float64

--------------------------
1    S        8.15458
2    W        8.59119
3    NE       8.75413
4    MW       8.1514
```

Using combine() and groupby()

```
# Rename the column
combine(groupby(wages, :region),
        :effective_min_wage_2020_dollars => mean => :average_min_wage_2020_dollars)
```

```
4x2 DataFrame
Row region    average_min_wage_2020_dollars
    String    Float64

--------------------------
1    S        8.15458
2    W        8.59119
3    NE       8.75413
4    MW       8.1514
```

Multiple functions on one column

```
# Use multiple functions on a single column
combine(groupby(wages, :region),
                    :effective_min_wage_2020_dollars .=> [mean, median, maximum])
```

```
4x4 DataFrame
Row  region  effective_min_wage_2020_dollars_mean  ...
     String  Float64                               ...
──────────────────────────────────────────────────────────────────────────
1    S       8.15458                               ...
2    W       8.59119                               ...
3    NE      8.75413                               ...
4    MW      8.1514                                ...
```

Multiple functions on one column

```
# Use multiple functions on a single column
combine(groupby(wages, :region), :effective_min_wage_2020_dollars .=> [mean, median] .=> [:average, :median])
```

```
4x2 DataFrame
Row region  average  median
    String  Float64  Float64
____________________________
1    S      8.15458  8.0
2    W      8.59119  8.34
...
```

```
# DON'T forget the dot!
combine(groupby(wages, :region), :effective_min_wage_2020_dollars => [mean, median])
```

```
ArgumentError: Unrecognized column selector ...
```

Multiple columns with one function

```
combine(groupby(wages, :region), [:state_min_wage, :federal_min_wage] .=> mean)
```

```
4x2 DataFrame
Row region  state_min_wage_mean  federal_min_wage_mean
    String  Float64              Float64
_____________________________________________________
1   S       2.73128              4.35566
2    W       4.26638              4.35566
...
```

```
# DON'T forget the dot
combine(groupby(wages, :region), [:state_min_wage, :federal_min_wage] => mean)
```

```
MethodError: objects of type ...
```

Multiple columns with multiple functions

```
# Functions as 1-row matrix
combine(groupby(wages, :region), [:state_min, :federal_min] .=> [mean median])
```

```
Row region  state_min_mean  federal_min_mean  state_min_median  federal_min_median
_____________________________________________________________________________
1   S       2.73128         4.35566           2.0                4.25
...
```

```
# Functions as a vector
combine(groupby(wages, :region), [:state_min, :federal_min] .=> [mean, median])
```

```
Row region  state_min_mean  federal_min_median
_____________________________________________
1   S       2.73128         4.25
...
```

Possible functions

Functions that can be used:

- Usual statistics functions as `sum()` , `mean()` , `minimum()` , ...
- User predefined functions (broadcasted)
- Anonymous functions, wrapped in `ByRow()`
- Special DataFrames function: `nrow` , `proprow` , ...

Cheat sheet

```
# Grouped DataFrame gdf
# 1 column + 1 function
combine(gdf, :c => f => :new_c)

# 1 column + 2+ functions
combine(gdf, :c .=> [f1, f2, ...] .=> [:new_c_f1, :new_c_f2, ...])

# 2+ columns + 1 function
combine(gdf, [:c1, :c2, ...] .=> f .=> [:new_c1_f, :new_c2_f, ...])

# 2+ columns + 2+ functions - all combinations
combine(gdf, [:c1, :c2, ...] .=> [f1 f2 ...] .=> [:c1_f1, :c2_f1, ..., :c1_f2, ...])

# 2+ columns + 2+ functions - pairwise
combine(gdf, [:c1, :c2, ...] .=> [f1, f2, ...] .=> [:new_c1_f1, :new_c2_f2, ...])
```

Let's practice!

DATA MANIPULATION IN JULIA

Pivoting data

DATA MANIPULATION IN JULIA

Katerina Zahradova
Instructor

datacamp

Pivot tables

- Way of summarizing data
- Reorganization of columns and rows to make data more readable

Pivot table using unstack()

```
unstack(df, cols_as_rows,
        col_as_columns,
        col_for_values,
        combine = f_to_use_on_values)
```

unstack() to pivot

```
# Using unstack to pivot
# unstack(DataFrame, :col_as_rows, :col_as_cols, :values, combine = f_to_aggregate)
unstack(penguins, :species, :sex, :body_mass_g, combine = median)
```

```
3×3 DataFrame
Row  species    MALE     FEMALE
     String15   Float64  Float64

_____________________________________
1    Adelie     4000.0   3400.0
2    Chinstrap  3950.0   3550.0
3    Gentoo     5500.0   4700.0
```

Pivoting on multiple columns as rows

```
# Pivot on more columns as rows
unstack(penguins, [:species, :island], :sex, :body_mass_g, combine = median)
```

```
5×4 DataFrame
Row  species    island     MALE       FEMALE
     String15   String15   Float64?   Float64?
──────────────────────────────────────────────
1    Adelie     Torgersen  4000.0     3400.0
2    Adelie     Biscoe     4000.0     3375.0
3    Adelie     Dream      3987.5     3400.0
4    Chinstrap  Dream      3950.0     3550.0
5    Gentoo     Biscoe     5500.0     4700.0
```

Pivoting on multiple columns elsewhere

```
# Using multiple columns as columns
unstack(penguins, :sex, [:species, :island], :body_mass_g, combine = sum)
```

```
MethodError: no method matching
unstack(::DataFrame, ::Symbol, ::Vector{Symbol}, ::Symbol; combine = sum)
```

Missing values

```
# Missing values for certain combinations
unstack(penguins_missing, :species, :sex, :body_mass_g, combine = median)
```

```
3×3 DataFrame
Row  species    MALE     FEMALE
     String15   Float64  Float64?
─────────────────────────────────
1    Adelie     4000.0   3400.0
2    Chinstrap  3950.0   3550.0
3    Gentoo     5500.0   missing
```

Replacing missing values

```
# Missing values for certain combinations
unstack(penguins_missing, :species, :sex, :body_mass_g, combine = median, fill = -1)
```

```
3×3 DataFrame
Row  species    MALE      FEMALE
     String15   Float64   Float64

-------------------------------------
1    Adelie     4000.0     3400.0
2    Chinstrap  3950.0     3550.0
3    Gentoo     5500.0     -1
```

Pivot tables are DataFrames

```
# Saving the pivot table
pivot_penguins = unstack(penguins_missing, :species, :sex, :body_mass_g, combine = median)

# Select only female penguins
select(pivot_penguins, :species, :FEMALE)
```

```
3×2 DataFrame
Row  species    FEMALE
     String15   Float64?
------------------------------
1    Adelie      3400.0
...
```

Let's practice!

DATA MANIPULATION IN JULIA

Improving readability with Chain.jl

DATA MANIPULATION IN JULIA

Katerina Zahradova
Instructor

datacamp

Problems with complicated code

```
# Not readable
combine(groupby(
    select(wages, :year, :eff_2020)
        :year),
        :eff_2020 => mean)
```

```
# Easy to forget a bracket
combine(groupby(wages, :year, ... )
```

```
syntax: incomplete:
premature end of input ...
```

```
# Many intermediate variables
w_tmp = select(wages, :state, :year, :eff_2020)
year_groups = groupby(w_tmp, :year)
combine(year_groups, :eff_2020 => mean)
```

```
# Easy to overwrite something important
select!(wages, :year, :eff_2020)
groupby(wages, :state)
```

```
ArgumentError:
column name "state" not found ...
```

What is piping?

- Coding approach
- Combines multiple subsequent function calls
- Keeps code easy to read
- No need for saving of intermediate results
- No nesting

Using Chain.jl

Computing average minimum wage by year

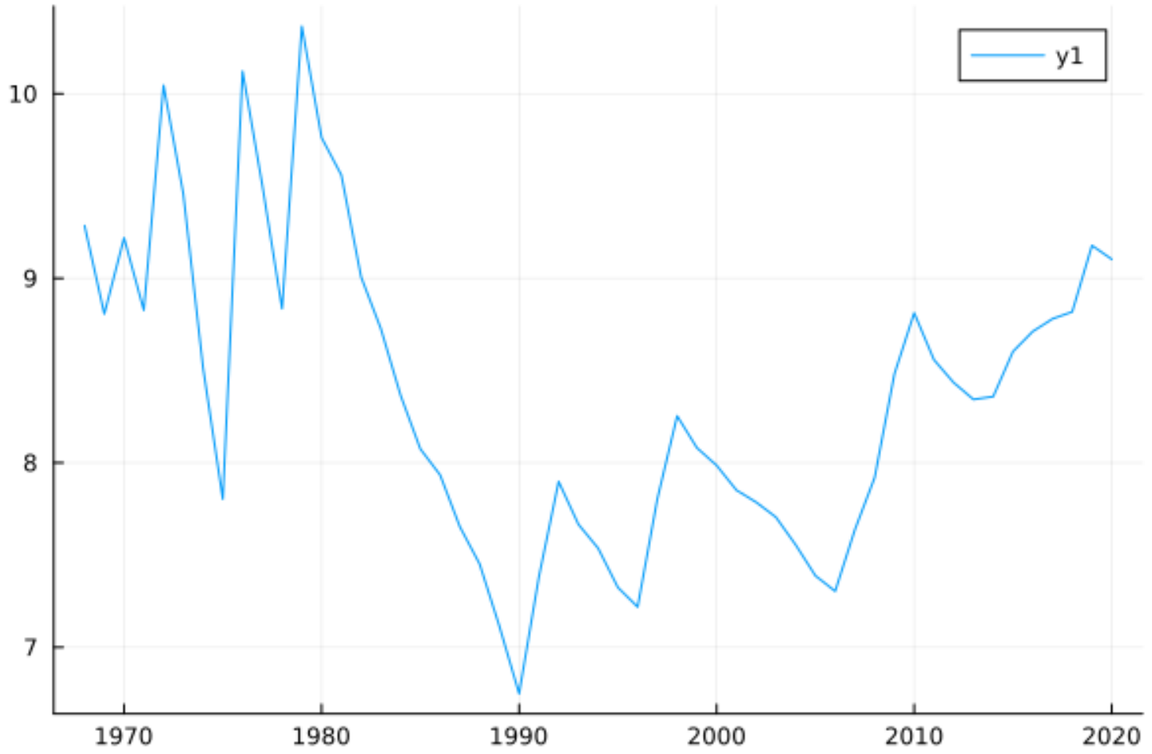
- selecting right columns
- group the DataFrame by `:year`
- computing the average per year

```
@chain wages begin
    select(:state, :year, :eff_2020)
    groupby(:year)
    combine(:eff_2020 => mean)
end
```

```
53×2 DataFrame
Row     year    eff_2020_mean
        Int64   Float64

--------------------------
1       1968    9.28529
...
```

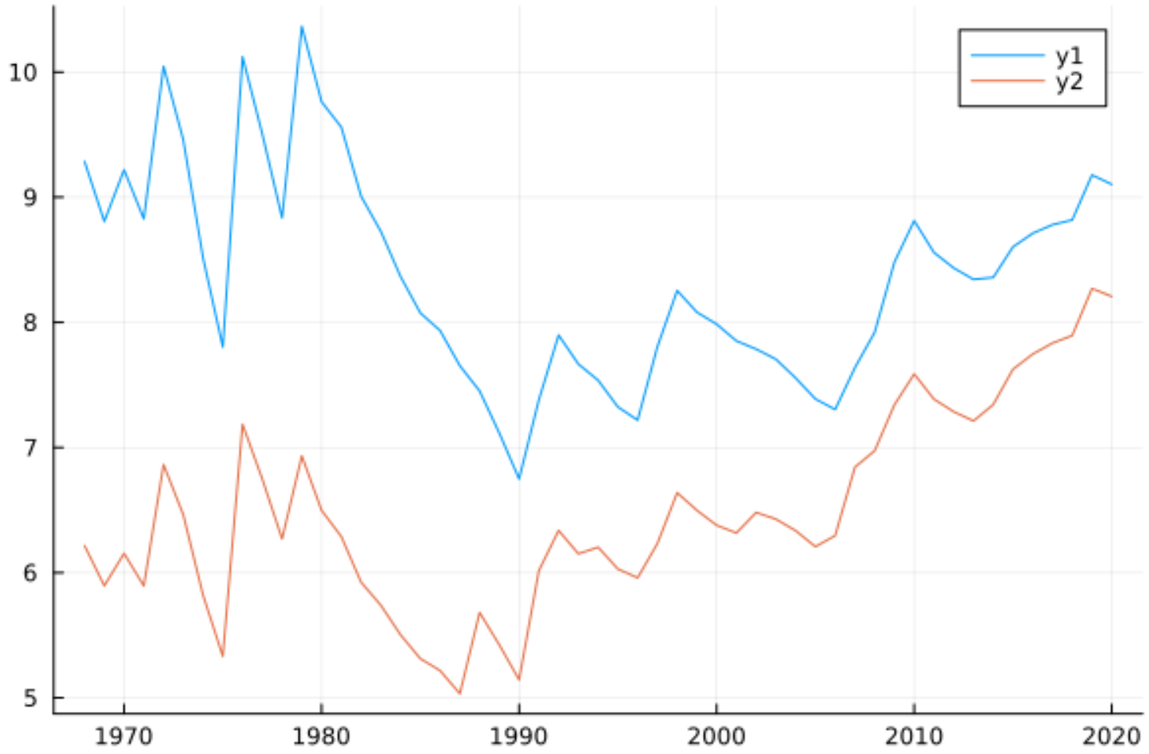
Piping with _

```
@chain wages begin
    select(:state, :year, :eff_2020)
    groupby(:year)
    combine(:eff_2020 => mean)
    # Use _ to pipe in multiple places
    plot(_.year, _.eff_2020_mean)
end
```



Skipping piping with @aside

```
@chain wages begin
    select(:state, :year,
        :eff_2020, :state_2020)
    groupby(:year)
    combine([:eff_2020,
        :state_2020] .=> mean)
    # Use @aside to skip piping
    @aside plot(_.year,_.eff_2020_mean)
    plot!(_.year,_.state_2020_mean)
end
```



Saving the result

```
# Save the output of the chain macro as a variable
wages_mean_by_year = @chain wages begin
    select(:state, :year, :eff_2020, :state_2020)
    groupby(:year)
    combine([:eff_2020, :state_2020] .=> mean)
end
# Print the first line
println(first(wages_mean_by_year))
```

```
DataFrameRow
Row | year     eff_2020_mean   state_2020_mean
    | Int64    Float64         Float64
____|_____________________________________
   1| 1968     9.28529         6.21549
```

Let's practice!

DATA MANIPULATION IN JULIA