Data Science in Julia

Data manipulation

by Yueh-Hua Tu

Outline

- Missings
- DataFrames
- Query

DataFrames

In [1]:

```
using Statistics
using DataFrames, CSV, Query
using RDatasets
```

In [2]:

```
using Pkg
Pkg.status(["DataFrames", "CSV", "Query", "RDatasets"])
```

```
Status `~/.julia/environments/v1.4/Project.toml`
  [336ed68f] CSV v0.6.2
  [a93c6f00] DataFrames v0.21.1
  [1a8c2f83] Query v0.12.2
  [ce6b1742] RDatasets v0.6.8
```

Missing

- missing is used to represent a missing value in Julia, and it is a built-in object.
- missing is the singleton of Missing type.
- Whenever operations including missing are undetermined will return missing.

In [3]:
```julia
true && missing
```

Out[3]:
```
missing
```

In [4]:
```julia
1 + missing
```

Out[4]:
```
missing
```

Short-cut logic

In [5]:
```
false && missing
```
Out[5]:
```
false
```

In [6]:
```
true || missing
```
Out[6]:
```
true
```

In [7]:
```
ismissing(missing)
```
Out[7]:
```
true
```

DataFrame

- DataFrame is a table-like data structure containing a series of vectors.
- Each column corresponds to a variable and has the same length.

In [8]:

```
df = DataFrame(A = 1:4, B = ["M", "F", "F", "M"])
```

Out[8]:

4 rows × 2 columns

	A	B
	Int64	String
1	1	M
2	2	F
3	3	F
4	4	M

In [9]:

```
df = DataFrame() # 先初始化一個空的，再放入資料
df[!, :A] = 1:8
df[!, :B] = ["M", "F", "F", "M", "F", "M", "M", "F"]
df
```

Out[9]:

8 rows × 2 columns

	A	B
	Int64	String
1	1	M
2	2	F
3	3	F
4	4	M
5	5	F
6	6	M
7	7	M
8	8	F

Initialize by rows

In [10]:
```
df = DataFrame(A=Int64[], B=String[])
```

Out[10]:

0 rows × 2 columns

	A	B
	Int64	String

In [11]:
```
push!(df, (1, "M"))
```

Out[11]:

1 rows × 2 columns

	A	B
	Int64	String
1	1	M

In [12]:
```
push!(df, [2, "F"])
```

Out[12]:

2 rows × 2 columns

	A	B
	Int64	String
1	1	M
2	2	F

In [13]:

```julia
push!(df, Dict(:B => "M", :A => 4))
```

Out[13]:

3 rows × 2 columns

	A	B
	Int64	String
1	1	M
2	2	F
3	4	M

Initialize by matrix

In [14]:

```
mat = [1 "M"; 2 "F"; 3 "F"]
DataFrame(mat, [:A, :B])
```

Out[14]:

3 rows × 2 columns

	A	B
	Any	Any
1	1	M
2	2	F
3	3	F

Initialize by dictionary

In [15]:

```
d = Dict("A" => [1, 2, 3], "B" => ["M", "F", "F"])
DataFrame(d)
```

Out[15]:

3 rows × 2 columns

	A	B
	Int64	String
1	1	M
2	2	F
3	3	F

Get column

In [16]:
```
df[!, :A]
```

Out[16]:
```
3-element Array{Int64,1}:
 1
 2
 4
```

In [17]:
```
df[!, 1]
```

Out[17]:
```
3-element Array{Int64,1}:
 1
 2
 4
```

In [18]:
```
df.A
```

Out[18]:
```
3-element Array{Int64,1}:
 1
 2
 4
```

Indexing by integer

In [19]:
```
df[1, 1]
```

Out[19]:
```
1
```

Indexing by column name

In [20]:
```
df[1, :A]
```

Out[20]:
```
1
```

Dimension information

In [21]:
```
size(df)
```
Out[21]:
```
(3, 2)
```

In [22]:
```
nrow(df)
```
Out[22]:
```
3
```

In [23]:
```
ncol(df)
```
Out[23]:
```
2
```

Get column names

In [24]:
```
names(df)
```

Out[24]:
```
2-element Array{String,1}:
 "A"
 "B"
```

In [25]:
```
df[1:3, :]
```

Out[25]:

3 rows × 2 columns

	A	B
	Int64	String
1	1	M
2	2	F
3	4	M

Get slice

In [26]:

```
df[1:3, [:B, :A]]
```

Out[26]:

3 rows × 2 columns

	B	A
	String	Int64
1	M	1
2	F	2
3	M	4

Assign column

In [27]:

```
df[!, :C] = ['a', 'b', 'c']
df
```

Out[27]:

3 rows × 3 columns

	A	B	C
	Int64	**String**	**Char**
1	1	M	'a'
2	2	F	'b'
3	4	M	'c'

In [28]:

```
df[!, :B] = ['α', 'β', 'γ']
df
```

Out[28]:

3 rows × 3 columns

	A	B	C
	Int64	**Char**	**Char**
1	1	'α'	'a'
2	2	'β'	'b'
3	4	'γ'	'c'

Conditioning

In [29]:
```
df[df[!,:A] .% 2 .== 0, :] # 取符合條件的列，跟所有欄位
```

Out[29]:

2 rows × 3 columns

	A	B	C
	Int64	Char	Char
1	2	'β'	'b'
2	4	'γ'	'c'

Calculation

In [30]:

```
df[!, :D] = df[!, :B] .* df[!, :C]
df
```

Out[30]:

3 rows × 4 columns

	A	B	C	D
	Int64	**Char**	**Char**	**String**
1	1	'α'	'a'	αa
2	2	'β'	'b'	βb
3	4	'γ'	'c'	γc

Aggregation

In [31]:
```
mean(df[!,1]) # 對第1欄取平均
```

Out[31]:
```
2.3333333333333335
```

In [32]:
```
median(df[!,:A]) # 對第一個欄位取中位數
```

Out[32]:
```
2.0
```

In [33]:
```
df = DataFrame(A = 1:4, B = randn(4))
mapcols(cumsum, df)
```

Out[33]:

4 rows × 2 columns

	A	B
	Int64	**Float64**
1	1	0.238524
2	3	1.09133
3	6	0.384994
4	10	0.286134

Storage

In [34]:
```
CSV.write("test.csv", df)
```

Out[34]:
```
"test.csv"
```

In [35]:
```
df = CSV.read("test.csv")
```

Out[35]:

4 rows × 2 columns

	A	B
	Int64	Float64
1	1	0.238524
2	2	0.852803
3	3	-0.706332
4	4	-0.0988603

Play with data

In [36]:
```
first(RDatasets.datasets(), 6) # 可以選你要的資料集
```

Out[36]:

6 rows × 5 columns

	Package	Dataset	Title	Rows	Columns
	String	String	String	Int64	Int64
1	COUNT	affairs	affairs	601	18
2	COUNT	azdrg112	azdrg112	1798	4
3	COUNT	azpro	azpro	3589	6
4	COUNT	badhealth	badhealth	1127	3
5	COUNT	fasttrakg	fasttrakg	15	9
6	COUNT	lbw	lbw	189	10

In [37]:
```julia
iris = dataset("datasets", "iris")
first(iris, 6)
```

Out[37]:

6 rows × 5 columns

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
	Float64	**Float64**	**Float64**	**Float64**	**Cat…**
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

In [38]:
```julia
size(iris) # 看一下他有幾列幾行
```

Out[38]:
```
(150, 5)
```

Concatenate DataFrames

In [39]:
```
size(vcat(iris, iris))
```
Out[39]:
```
(300, 5)
```

In [40]:
```
size(hcat(iris, iris, makeunique=true))
```
Out[40]:
```
(150, 10)
```

Check if missing exists in each row

In [41]:
```
completecases(iris)  # true means no missing
```

Out[41]:
```
150-element BitArray{1}:
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 ⋮
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
 1
```

Select complete cases

In [42]:

```
first(iris[completecases(iris), :], 10)  # equivalent to complete_cases!(iris)
```

Out[42]:

10 rows × 5 columns

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
	Float64	Float64	Float64	Float64	Cat...
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa

Get unique combinations

In [43]:

```
first(unique(iris), 10)
```

Out[43]:

10 rows × 5 columns

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
	Float64	Float64	Float64	Float64	Cat...
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa

In [44]:

```
iris[!,:PetalArea] = iris[!,:PetalLength] .* iris[!,:PetalWidth]
first(iris)
```

Out[44]:

DataFrameRow (6 columns)

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species	PetalArea
	Float64	Float64	Float64	Float64	Cat...	Float64
1	5.1	3.5	1.4	0.2	setosa	0.28

Sorting

In [45]:
```
first(sort!(iris, :SepalLength), 10)
```

Out[45]:

10 rows × 6 columns

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species	PetalArea
	Float64	Float64	Float64	Float64	Cat...	Float64
1	4.3	3.0	1.1	0.1	setosa	0.11
2	4.4	2.9	1.4	0.2	setosa	0.28
3	4.4	3.0	1.3	0.2	setosa	0.26
4	4.4	3.2	1.3	0.2	setosa	0.26
5	4.5	2.3	1.3	0.3	setosa	0.39
6	4.6	3.1	1.5	0.2	setosa	0.3
7	4.6	3.4	1.4	0.3	setosa	0.42
8	4.6	3.6	1.0	0.2	setosa	0.2
9	4.6	3.2	1.4	0.2	setosa	0.28
10	4.7	3.2	1.3	0.2	setosa	0.26

Sorting multiple columns and specify the order

In [46]:
```
first(sort!(iris, [:Species, :SepalLength, :SepalWidth], rev=[true, false, false]), 10)
```

Out[46]:

10 rows × 6 columns

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species	PetalArea
	Float64	Float64	Float64	Float64	Cat...	Float64
1	4.9	2.5	4.5	1.7	virginica	7.65
2	5.6	2.8	4.9	2.0	virginica	9.8
3	5.7	2.5	5.0	2.0	virginica	10.0
4	5.8	2.7	5.1	1.9	virginica	9.69
5	5.8	2.7	5.1	1.9	virginica	9.69
6	5.8	2.8	5.1	2.4	virginica	12.24
7	5.9	3.0	5.1	1.8	virginica	9.18
8	6.0	2.2	5.0	1.5	virginica	7.5
9	6.0	3.0	4.8	1.8	virginica	8.64
10	6.1	2.6	5.6	1.4	virginica	7.84

Join

In [47]:
```
employees = DataFrame(ID = [1, 2, 3], Name = ["John Doe", "Jane Doe", "Andy Doe"])
```

Out[47]:

3 rows × 2 columns

	ID	Name
	Int64	String
1	1	John Doe
2	2	Jane Doe
3	3	Andy Doe

In [48]:
```
jobs = DataFrame(ID = [1, 2, 4], Job = ["Lawyer", "Doctor", "Chief"])
```

Out[48]:

3 rows × 2 columns

	ID	Job
	Int64	String
1	1	Lawyer
2	2	Doctor
3	4	Chief

Inner join

In [49]:

```
full = innerjoin(employees, jobs, on=:ID)
```

Out[49]:

2 rows × 3 columns

	ID	Name	Job
	Int64	String	String
1	1	John Doe	Lawyer
2	2	Jane Doe	Doctor

Left join

In [50]:

```
left_join = leftjoin(employees, jobs, on=:ID)
```

Out[50]:

3 rows × 3 columns

	ID	Name	Job
	Int64	String	String?
1	1	John Doe	Lawyer
2	2	Jane Doe	Doctor
3	3	Andy Doe	*missing*

Right join

In [51]:
```
right_join = rightjoin(employees, jobs, on=:ID)
```

Out[51]:

3 rows × 3 columns

	ID	Name	Job
	Int64	String?	String
1	1	John Doe	Lawyer
2	2	Jane Doe	Doctor
3	4	*missing*	Chief

Outer join

In [52]:

```
outer_join = outerjoin(employees, jobs, on=:ID)
```

Out[52]:

4 rows × 3 columns

	ID	Name	Job
	Int64	**String?**	**String?**
1	1	John Doe	Lawyer
2	2	Jane Doe	Doctor
3	3	Andy Doe	*missing*
4	4	*missing*	Chief

Other join

- Semi join: similar to inner join， but only outputs left table
- Anti join
- Cross join: Cartesian product

In [53]:

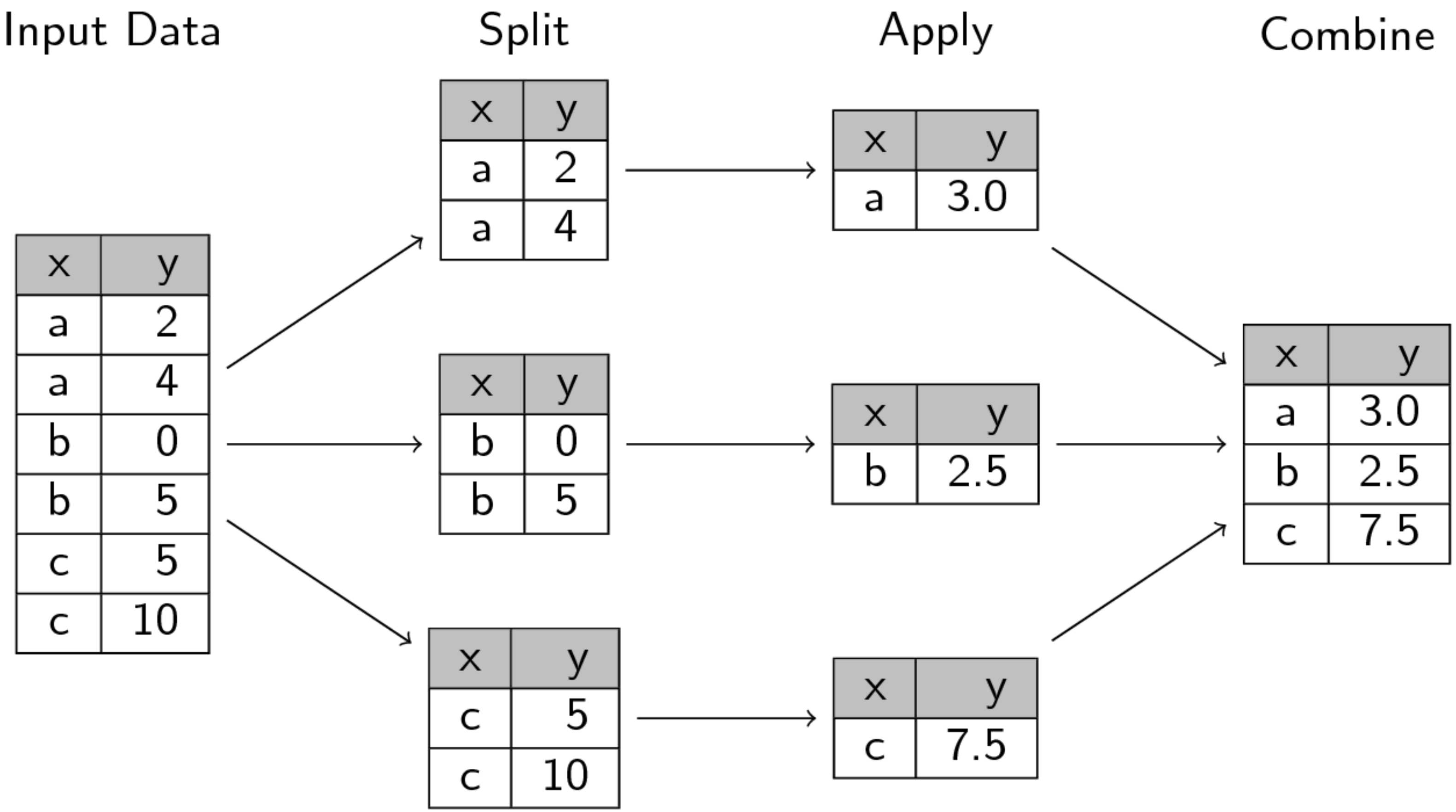
```
cross_join = crossjoin(employees, jobs, makeunique=true) # 不需要key
```

Out[53]:

9 rows × 4 columns

	ID	Name	ID_1	Job
	Int64	**String**	**Int64**	**String**
1	1	John Doe	1	Lawyer
2	1	John Doe	2	Doctor
3	1	John Doe	4	Chief
4	2	Jane Doe	1	Lawyer
5	2	Jane Doe	2	Doctor
6	2	Jane Doe	4	Chief
7	3	Andy Doe	1	Lawyer
8	3	Andy Doe	2	Doctor
9	3	Andy Doe	4	Chief

Split-Apply-Combine Strategy



[https://cities.github.io/datascience2018/11-split-apply-combine.html (https://cities.github.io/datascience2018/11-split-apply-combine.html)](https://cities.github.io/datascience2018/11-split-apply-combine.html)

Split by :**Species**, calculate **size()**

arguments

1. DataFrame
2. The column which you like to split
3. The function or expression to apply to each group of data

In [54]:

```
by(iris, :Species, size)
```

```
┌ Warning: `by(d::AbstractDataFrame, cols::Any, f::Base.Callable; sort::Bool = false, skipmissing::Bool = false)` is deprecated, use `combine(f, groupby(d, cols, sort = sort, skipmissing = skipmissing))` instead.
│   caller = top-level scope at In[54]:1
└ @ Core In[54]:1
```

Out[54]:

3 rows × 2 columns

	Species	x1
	Cat…	Tuple…
1	setosa	(50, 6)
2	versicolor	(50, 6)
3	virginica	(50, 6)

More complex situation

In [55]:

```
by(iris, :Species) do df
   DataFrame(μ = mean(df[!,:PetalLength]), σ = var(df[!,:PetalLength]))
end
```

```
┌ Warning: `by(f::Base.Callable, d::AbstractDataFrame, cols::Any; sort::Bool = false, skipmissing::Bool = false)` is deprecated, use `combine(f, groupby(d, cols, sort = sort, skipmissing = skipmissing))` instead.
│   caller = top-level scope at In[55]:1
└ @ Core In[55]:1
```

Out[55]:

3 rows × 3 columns

	Species	μ	σ
	Cat...	Float64	Float64
1	setosa	1.462	0.0301592
2	versicolor	4.26	0.220816
3	virginica	5.552	0.304588

Group by **:Species**, calculate sum and average

arguments

1. DataFrame
2. The column which you like to split
3. The function or expression to apply to each group of data

In [56]:
```
aggregate(iris, :Species, [sum, mean])
```

```
┌ Warning: `aggregate(d, cols, fs, sort=false, skipmissing=false)` is deprecated. Instead use combine(groupby(d, cols, sort=false, skipmissing=false), [names(d, Not(cols)) .=> f for f in fs]...)` if functions in `fs` have unique names.
│   caller = top-level scope at In[56]:1
└ @ Core In[56]:1
```

Out[56]:

3 rows × 11 columns (omitted printing of 5 columns)

	Species	SepalLength_sum	SepalWidth_sum	PetalLength_sum	PetalWidth_sum	PetalArea_sum
	Cat…	Float64	Float64	Float64	Float64	Float64
1	setosa	250.3	171.4	73.1	12.3	18.28
2	versicolor	296.8	138.5	213.0	66.3	286.02
3	virginica	329.4	148.7	277.6	101.3	564.81

Group by

In [57]:

```
for subdf in groupby(iris, :Species)
    println(size(subdf, 1))
end
```

```
50
50
50
```

Group by and combine

In [58]:
```
gdf = groupby(iris, :Species)
```

Out[58]:

GroupedDataFrame with 3 groups based on key: Species

First Group (50 rows): Species = "setosa"

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species	PetalArea
	Float64	Float64	Float64	Float64	Cat...	Float64
1	4.3	3.0	1.1	0.1	setosa	0.11
2	4.4	2.9	1.4	0.2	setosa	0.28
3	4.4	3.0	1.3	0.2	setosa	0.26
4	4.4	3.2	1.3	0.2	setosa	0.26
5	4.5	2.3	1.3	0.3	setosa	0.39
6	4.6	3.1	1.5	0.2	setosa	0.3
7	4.6	3.2	1.4	0.2	setosa	0.28
8	4.6	3.4	1.4	0.3	setosa	0.42
9	4.6	3.6	1.0	0.2	setosa	0.2
10	4.7	3.2	1.3	0.2	setosa	0.26
11	4.7	3.2	1.6	0.2	setosa	0.32
12	4.8	3.0	1.4	0.1	setosa	0.14
13	4.8	3.0	1.4	0.3	setosa	0.42
14	4.8	3.1	1.6	0.2	setosa	0.32
15	4.8	3.4	1.6	0.2	setosa	0.32
16	4.8	3.4	1.9	0.2	setosa	0.38
17	4.9	3.0	1.4	0.2	setosa	0.28
18	4.9	3.1	1.5	0.1	setosa	0.15
19	4.9	3.1	1.5	0.2	setosa	0.3
20	4.9	3.6	1.4	0.1	setosa	0.14
21	5.0	3.0	1.6	0.2	setosa	0.32
22	5.0	3.2	1.2	0.2	setosa	0.24
23	5.0	3.3	1.4	0.2	setosa	0.28
24	5.0	3.4	1.5	0.2	setosa	0.3

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species	PetalArea
	Float64	**Float64**	**Float64**	**Float64**	**Cat…**	**Float64**
25	5.0	3.4	1.6	0.4	setosa	0.64
26	5.0	3.5	1.3	0.3	setosa	0.39
27	5.0	3.5	1.6	0.6	setosa	0.96
28	5.0	3.6	1.4	0.2	setosa	0.28
29	5.1	3.3	1.7	0.5	setosa	0.85
30	5.1	3.4	1.5	0.2	setosa	0.3
⋮	⋮	⋮	⋮	⋮	⋮	⋮

⋮

Last Group (50 rows): Species = "virginica"

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species	PetalArea
	Float64	**Float64**	**Float64**	**Float64**	**Cat…**	**Float64**
1	4.9	2.5	4.5	1.7	virginica	7.65
2	5.6	2.8	4.9	2.0	virginica	9.8
3	5.7	2.5	5.0	2.0	virginica	10.0
4	5.8	2.7	5.1	1.9	virginica	9.69
5	5.8	2.7	5.1	1.9	virginica	9.69
6	5.8	2.8	5.1	2.4	virginica	12.24
7	5.9	3.0	5.1	1.8	virginica	9.18
8	6.0	2.2	5.0	1.5	virginica	7.5
9	6.0	3.0	4.8	1.8	virginica	8.64
10	6.1	2.6	5.6	1.4	virginica	7.84
11	6.1	3.0	4.9	1.8	virginica	8.82
12	6.2	2.8	4.8	1.8	virginica	8.64
13	6.2	3.4	5.4	2.3	virginica	12.42
14	6.3	2.5	5.0	1.9	virginica	9.5
15	6.3	2.7	4.9	1.8	virginica	8.82
16	6.3	2.8	5.1	1.5	virginica	7.65
17	6.3	2.9	5.6	1.8	virginica	10.08
18	6.3	3.3	6.0	2.5	virginica	15.0

In [59]:
```
combine(gdf, :PetalLength => mean)
```

Out[59]:

3 rows × 2 columns

	Species	PetalLength_mean
	Cat…	Float64
1	setosa	1.462
2	versicolor	4.26
3	virginica	5.552

Reshape

Original data format is wide format

In [60]:

```
iris[!,:id] = 1:size(iris, 1)
first(iris, 10)
```

Out[60]:

10 rows × 7 columns

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species	PetalArea	id
	Float64	Float64	Float64	Float64	Cat...	Float64	Int64
1	4.9	2.5	4.5	1.7	virginica	7.65	1
2	5.6	2.8	4.9	2.0	virginica	9.8	2
3	5.7	2.5	5.0	2.0	virginica	10.0	3
4	5.8	2.7	5.1	1.9	virginica	9.69	4
5	5.8	2.7	5.1	1.9	virginica	9.69	5
6	5.8	2.8	5.1	2.4	virginica	12.24	6
7	5.9	3.0	5.1	1.8	virginica	9.18	7
8	6.0	2.2	5.0	1.5	virginica	7.5	8
9	6.0	3.0	4.8	1.8	virginica	8.64	9
10	6.1	2.6	5.6	1.4	virginica	7.84	10

Stack specified column into data and transform into long format

In [61]:

```
d = stack(iris, [:SepalLength, :SepalWidth, :PetalLength, :PetalWidth])
first(d, 10)
```

Out[61]:

10 rows × 5 columns

	Species	PetalArea	id	variable	value
	Cat...	Float64	Int64	Cat...	Float64
1	virginica	7.65	1	SepalLength	4.9
2	virginica	9.8	2	SepalLength	5.6
3	virginica	10.0	3	SepalLength	5.7
4	virginica	9.69	4	SepalLength	5.8
5	virginica	9.69	5	SepalLength	5.8
6	virginica	12.24	6	SepalLength	5.8
7	virginica	9.18	7	SepalLength	5.9
8	virginica	7.5	8	SepalLength	6.0
9	virginica	8.64	9	SepalLength	6.0
10	virginica	7.84	10	SepalLength	6.1

Stack other columns

In [62]:
```
d = stack(iris, [:SepalLength, :SepalWidth], :Species)
first(d, 10)
```

Out[62]:

10 rows × 3 columns

	Species	variable	value
	Cat...	Cat...	Float64
1	virginica	SepalLength	4.9
2	virginica	SepalLength	5.6
3	virginica	SepalLength	5.7
4	virginica	SepalLength	5.8
5	virginica	SepalLength	5.8
6	virginica	SepalLength	5.8
7	virginica	SepalLength	5.9
8	virginica	SepalLength	6.0
9	virginica	SepalLength	6.0
10	virginica	SepalLength	6.1

Unstack

In [63]:

```
d = stack(iris, [:SepalLength, :SepalWidth, :PetalLength, :PetalWidth])
first(unstack(d, :id, :variable, :value), 10)
```

Out[63]:

10 rows × 5 columns

	id	SepalLength	SepalWidth	PetalLength	PetalWidth
	Int64	Float64?	Float64?	Float64?	Float64?
1	1	4.9	2.5	4.5	1.7
2	2	5.6	2.8	4.9	2.0
3	3	5.7	2.5	5.0	2.0
4	4	5.8	2.7	5.1	1.9
5	5	5.8	2.7	5.1	1.9
6	6	5.8	2.8	5.1	2.4
7	7	5.9	3.0	5.1	1.8
8	8	6.0	2.2	5.0	1.5
9	9	6.0	3.0	4.8	1.8
10	10	6.1	2.6	5.6	1.4

Not specify identifier

In [64]:

```
first(unstack(d, :variable, :value), 10)
```

Out[64]:

10 rows × 7 columns

	Species	PetalArea	id	SepalLength	SepalWidth	PetalLength	PetalWidth
	Cat...	Float64	Int64	Float64?	Float64?	Float64?	Float64?
1	setosa	0.11	101	4.3	3.0	1.1	0.1
2	setosa	0.14	112	4.8	3.0	1.4	0.1
3	setosa	0.14	120	4.9	3.6	1.4	0.1
4	setosa	0.15	118	4.9	3.1	1.5	0.1
5	setosa	0.15	139	5.2	4.1	1.5	0.1
6	setosa	0.2	109	4.6	3.6	1.0	0.2
7	setosa	0.24	122	5.0	3.2	1.2	0.2
8	setosa	0.24	150	5.8	4.0	1.2	0.2
9	setosa	0.26	103	4.4	3.0	1.3	0.2
10	setosa	0.26	104	4.4	3.2	1.3	0.2

Application

Calculate the average of every feature grouped by :Species

In [65]:

```
d = stack(iris)
x = by(d, [:variable, :Species], df -> DataFrame(vsum = mean(df[!,:value])))
unstack(x, :Species, :vsum)
```

```
┌ Warning: `by(d::AbstractDataFrame, cols::Any, f::Base.Callable; sort::Bool = false, skipmissing::Bool = false)` is deprecated, use `combine(f, groupby(d, cols, sort = sort, skipmissing = skipmissing))` instead.
│   caller = top-level scope at In[65]:2
└ @ Core In[65]:2
```

Out[65]:

5 rows × 4 columns

	variable	setosa	versicolor	virginica
	Cat…	Float64?	Float64?	Float64?
1	SepalLength	5.006	5.936	6.588
2	SepalWidth	3.428	2.77	2.974
3	PetalLength	1.462	4.26	5.552
4	PetalWidth	0.246	1.326	2.026
5	PetalArea	0.3656	5.7204	11.2962

Query

In [66]:
```
df = DataFrame(name=["John", "Sally", "Kirk"], age=[23., 42., 59.], children=[3,5,2])
```

Out[66]:

3 rows × 3 columns

	name	age	children
	String	Float64	Int64
1	John	23.0	3
2	Sally	42.0	5
3	Kirk	59.0	2

select-from-where

In [67]:

```
x = @from i in df begin
    @where i.age>50
    @select {i.name, i.children}
    @collect DataFrame
end
```

Out[67]:

1 rows × 2 columns

	name	children
	String	Int64
1	Kirk	2

Sorting

In [68]:

```
df = DataFrame(a=[2,1,1,2,1,3],b=[2,2,1,1,3,2])

x = @from i in df begin
    @orderby descending(i.a), i.b
    @select i
    @collect DataFrame
end
```

Out[68]:

6 rows × 2 columns

	a	b
	Int64	Int64
1	3	2
2	2	1
3	2	2
4	1	1
5	1	2
6	1	3

Filtering

In [69]:

```
df = DataFrame(name=["John", "Sally", "Kirk"], age=[23., 42., 59.], children=[3,5,2])

x = @from i in df begin
    @where i.age > 30. && i.children > 2
    @select i
    @collect DataFrame
end
```

Out[69]:

1 rows × 3 columns

	name	age	children
	String	Float64	Int64
1	Sally	42.0	5

Projecting

In [70]:

```
data = [1,2,3]

x = @from i in data begin
   @select i^2
   @collect
end
```

Out[70]:

```
3-element Array{Int64,1}:
 1
 4
 9
```

Flattening

In [71]:

```
source = Dict(:a=>[1,2,3], :b=>[4,5])

q = @from i in source begin
  @from j in i.second
  @select {Key=i.first,Value=j}
  @collect DataFrame
end
```

Out[71]:

5 rows × 2 columns

	Key	Value
	Symbol	Int64
1	a	1
2	a	2
3	a	3
4	b	4
5	b	5

Grouping

In [72]:

```julia
df = DataFrame(name=["John", "Sally", "Kirk"], age=[23., 42., 59.], children=[3,2,2])

x = @from i in df begin
    @group i.name by i.children
    @collect
end
```

Out[72]:

```
2-element Array{Grouping{Int64,String},1}:
 ["John"]
 ["Sally", "Kirk"]
```

Q & A