Criação básica de DataFrames

	A	B	Fixed
1	1	5	1
2	2	6	1
3	3	7	1

```
begin
    DataFrame(A=1:3,B=5:7,Fixed = 1)
end
```

	A	B
1	1	"M"
2	2	"F"
3	3	"F"

```
begin
    data_frame = DataFrame(A = Int[], B = String[])
    push!(data_frame, (1, "M"))
    push!(data_frame, [2, "F"])
    push!(data_frame, Dict(:B => "F", :A => 3))
end
```

random_tab =

	a	b	c
1	0.41124	0.434895	0.675803
2	0.551547	0.110778	0.558085
3	0.0387148	0.175463	0.865529
4	0.00766722	0.55863	0.437181
5	0.799352	0.868233	0.933774
6	0.870858	0.363279	0.819898
7	0.884659	0.880002	0.72473
8	0.476225	0.879316	0.531207
9	0.942984	0.215322	0.0769115
10	0.265046	0.696807	0.154383

```
random_tab = DataFrame(rand(10, 3), [:a, :b, :c])
```

	customer age	first name
1	15	"Rohit"
2	20	"Rahul"
3	25	"Akshat"

```
DataFrame("customer age" => [15, 20, 25],
          "first name" =>["Rohit", "Rahul", "Akshat"])
```

	customer age	first name
1	15	"Rohit"
2	20	"Rahul"
3	25	"Akshat"

```
#d=DataFrames a partir de dicionários
begin
    dict = Dict("customer age" => [15, 20, 25],
                "first name" => ["Rohit", "Rahul", "Akshat"])
    DataFrame(dict)
end
```

	a	b
1	1	3
2	2	4

```
#DataFrames a partir de vetor de tuple
DataFrame((a=[1, 2], b=[3, 4]))
```

	a	b
1	1	0
2	2	0

```
#DataFrame a partir de vetores
DataFrame([(a=1, b=0), (a=2, b=0)])
```

	x1	x2
1	1	0
2	2	0

```
#DataFrame a partir de MATRIZES
DataFrame([1 0; 2 0], :auto)
```

	a	b	c	d
1	1	2	4	5
2	15	58	69	41
3	23	21	26	69

```
begin
    data = [1 2 4 5; 15 58 69 41; 23 21 26 69]
    nomes = ["a", "b", "c", "d"]
    DataFrame(data,nomes)
end
```

Leitura de CSV

	id	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Du
1	0	67	"male"	2	"own"	"NA"	"little"	1169	6
2	1	22	"female"	2	"own"	"little"	"moderate"	5951	48
3	2	49	"male"	1	"own"	"little"	"NA"	2096	12
4	3	45	"male"	2	"free"	"little"	"little"	7882	42
5	4	53	"male"	2	"free"	"little"	"little"	4870	24
6	5	35	"male"	1	"free"	"NA"	"NA"	9055	36
7	6	53	"male"	2	"own"	"quite rich"	"NA"	2835	24
8	7	35	"male"	3	"rent"	"little"	"moderate"	6948	36
9	8	61	"male"	1	"own"	"rich"	"NA"	3059	12
10	9	28	"male"	3	"own"	"little"	"moderate"	5234	30
more									
1000	999	27	"male"	2	"own"	"moderate"	"moderate"	4576	45

```
begin
    german_ref =CSV.read(joinpath(dirname(pathof(DataFrames)),
                                  "..", "docs", "src", "assets",
"german.csv"),
                         DataFrame)

    #obs joinpath(dirname(pathof(DataFrames)),"..", "docs", "src", "assets",
"german.csv")
    #é equivalente a:
#C:\\Users\\gabri\\.julia\\packages\\DataFrames\\vuMM8\\src\\..\\docs\\src\\assets\\
german.csv"

    german = copy(german_ref)

    #obs: também é possivel
    german_ref2 = DataFrame(CSV.File(
"C:\\Users\\gabri\\.julia\\packages\\DataFrames\\vuMM8\\src\\..\\docs\\src\\assets\\
german.csv"))

    #leitura excel
    #df = DataFrame(XLSX.readtable("myfile.xlsx", "mysheet")...)

end
```

Exportar CSV

```
"C:\\Users\\gabri\\OneDrive\\Documents\\teste_csv_julia.csv"
```

```
begin
    caminho_escrita = "C:\\Users\\gabri\\OneDrive\\Documents\\teste_csv_julia.csv"
    CSV.write(caminho_escrita,german)
end
```

Vizualização

	id	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Du
1	0	67	"male"	2	"own"	"NA"	"little"	1169	6
2	1	22	"female"	2	"own"	"little"	"moderate"	5951	48
3	2	49	"male"	1	"own"	"little"	"NA"	2096	12
4	3	45	"male"	2	"free"	"little"	"little"	7882	42
5	4	53	"male"	2	"free"	"little"	"little"	4870	24
6	5	35	"male"	1	"free"	"NA"	"NA"	9055	36
7	6	53	"male"	2	"own"	"quite rich"	"NA"	2835	24
8	7	35	"male"	3	"rent"	"little"	"moderate"	6948	36
9	8	61	"male"	1	"own"	"rich"	"NA"	3059	12
10	9	28	"male"	3	"own"	"little"	"moderate"	5234	30
more									
1000	999	27	"male"	2	"own"	"moderate"	"moderate"	4576	45

```
begin
    first(german,5) #mostra as primeiras 5 linhas do DataFrame
    last(german, 6) #mostra as ultimas 6 linhas do DataFrame

    ### view(DataFrame, array_linhas, array_colunas)
    view(german, 2, 2) #mostra uma unica célula
    view(german, : , :) #mostra as linhas e colunas de algum DataFrame
end
```

	Sex	Saving accounts
1	"male"	"NA"
2	"female"	"little"
3	"male"	"little"
4	"male"	"little"
5	"male"	"NA"
6	"male"	"quite rich"
7	"male"	"little"
8	"male"	"rich"
9	"male"	"little"
10	"female"	"little"
	more	
999	"male"	"moderate"

```
begin
    #usando betewwn, cols, not
    german[:, Not(:Age)] #retorna todo a DataFrame menos a coluna :Age
    german[:, Between(:Sex, :Housing)] #Retorna as colunas Sex, Job, Housing
    german[:,Cols("Age",Between("Sex","Job"))]#Retorna as colunas Age,Sex,Job,
Housing
    german[Not(5), r"S"] #retorna colunas com a letra S menos a linhas 5
end
```

```
["Sex", "Housing", "Saving accounts", "Checking account", "Purpose"]
```

```
begin
    names(german) #retorna uma lista de nomes das colunas
    names(german, String) #retorna uma lista de nomes das Colunas do tipo String
end
```

```
[:id, :Age, :Sex, :Job, :Housing, Symbol("Saving accounts"), Symbol("Checking account"),
```

```
propertynames(german) #retorna uma lista de simbolos de l=nomes das colunas
```

```
[Int64, Int64, String, Int64, String, String, String, Int64, Int64, String]
```

```
eltype.(eachcol(german)) #retorna uma lista de tipos de cada coluna
```

empty x empty!

	id	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpc

```
begin
    a = empty(german) #cria um novo DataFrame com o formato de german, mas vazio
    #b = empty!(german) #exclui as linhas de german
end
```

Infos Basicas do DataFrame

10

```
begin
    size(german) #(1000,100)
    size(german,1) #1000
    size(german,2) #10

    nrow(german) #1000
    ncol(german) #10
end
```

	variable	mean	min	median	max	nmiss
1	:id	499.5	0	499.5	999	0
2	:Age	35.546	19	33.0	75	0
3	:Sex	nothing	"female"	nothing	"male"	0
4	:Job	1.904	0	2.0	3	0
5	:Housing	nothing	"free"	nothing	"rent"	0
6	Symbol("Saving accounts")	nothing	"NA"	nothing	"rich"	0
7	Symbol("Checking account")	nothing	"NA"	nothing	"rich"	0
8	Symbol("Credit amount")	3271.26	250	2319.5	18424	0
9	:Duration	20.903	4	18.0	72	0
10	:Purpose	nothing	"business"	nothing	"vacation/others"	0

```
describe(german)
```

	variable	mean	min	median	max	nmissing	eltype
1	:id	499.5	0	499.5	999	0	Int64
2	:Age	35.546	19	33.0	75	0	Int64
3	:Sex	nothing	"female"	nothing	"male"	0	String

```
describe(german, cols=1:3) # define quais colunas são analisadas
```

```
#função show(german, allrows=True) -> imprimirá todas as linhas
```

Análises de dados

35.546

```
begin
    mean(skipmissing(german."Age"))
end
```

	id	Age	Sex	Job	Housing	Saving accounts	Che
1	0	4489	"malemale"	4	"ownown"	"NANA"	"litt
2	1	484	"femalefemale"	4	"ownown"	"littlelittle"	"mode
3	4	2401	"malemale"	1	"ownown"	"littlelittle"	"NANA
4	9	2025	"malemale"	4	"freefree"	"littlelittle"	"litt
5	16	2809	"malemale"	4	"freefree"	"littlelittle"	"litt
6	25	1225	"malemale"	1	"freefree"	"NANA"	"NANA
7	36	2809	"malemale"	4	"ownown"	"quite richquite rich"	"NANA
8	49	1225	"malemale"	9	"rentrent"	"littlelittle"	"mode
9	64	3721	"malemale"	1	"ownown"	"richrich"	"NANA
10	81	784	"malemale"	9	"ownown"	"littlelittle"	"mode
	more						
1000	998001	729	"malemale"	4	"ownown"	"moderatemoderate"	"mode

```
mapcols(id -> id .^ 2, german)

#realia uma função a todas coluna. Deve passar a coluna como argumento da função
#não altera a função original german
```

Renomear Colunas

```
md" ### Renomear Colunas"
```

	id	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Tin
1	0	67	"male"	2	"own"	"NA"	"little"	1169	6
2	1	22	"female"	2	"own"	"little"	"moderate"	5951	48
3	2	49	"male"	1	"own"	"little"	"NA"	2096	12
4	3	45	"male"	2	"free"	"little"	"little"	7882	42
5	4	53	"male"	2	"free"	"little"	"little"	4870	24
6	5	35	"male"	1	"free"	"NA"	"NA"	9055	36
7	6	53	"male"	2	"own"	"quite rich"	"NA"	2835	24
8	7	35	"male"	3	"rent"	"little"	"moderate"	6948	36
9	8	61	"male"	1	"own"	"rich"	"NA"	3059	12
10	9	28	"male"	3	"own"	"little"	"moderate"	5234	30
more									
1000	999	27	"male"	2	"own"	"moderate"	"moderate"	4576	45

```
rename!(german,  "Duration" => "Time")
```

Acessando coluna

```
PooledArrays.PooledVector{String, UInt32, Vector{UInt32}}: ["male", "female", "male", "
```

```
begin
    #acessando a coluna Sex do DataFrame
    german.Sex
    german."Sex"
    german[!,"Sex"]
    col_name = "Sex"
    german[!,col_name]
end
```

```
PooledArrays.PooledVector{String, UInt32, Vector{UInt32}}: ["male", "female", "male", "
```

```
begin
    #acessando a coluna Sex do DataFrame
    german[:,col_name]
    #dessa forma é criada uma cópia da coluna que não irá afetar a coluna original
end
```

```
false
```

```
begin
    german.Sex === german[!, :Sex] #True
    german.Sex === german[:, :Sex] #False
end
```

Acessando Index

data_frame[selected_rows, selected_columns]

data_frame[array , array] -> Retorna um DataFrame

data_frame[array , Symbol/Strin] -> Retorna um Array

pode-se usar data_frame[condicao,coluna]

É sempre necessário passar linhas e colunas. Operado4: [:] indica todos os elementos da linha ou coluna

select(DataFrame, colunas)

tem a vantagem de poder ter vários métodos de seleção de colunas

ex. select(DF, 1, :Sex, r'a')

	id	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Tin
1	0	67	"male"	2	"own"	"NA"	"little"	1169	6
2	1	22	"female"	2	"own"	"little"	"moderate"	5951	48
3	2	49	"male"	1	"own"	"little"	"NA"	2096	12
4	3	45	"male"	2	"free"	"little"	"little"	7882	42
5	4	53	"male"	2	"free"	"little"	"little"	4870	24
6	5	35	"male"	1	"free"	"NA"	"NA"	9055	36
7	6	53	"male"	2	"own"	"quite rich"	"NA"	2835	24
8	7	35	"male"	3	"rent"	"little"	"moderate"	6948	36
9	8	61	"male"	1	"own"	"rich"	"NA"	3059	12
10	9	28	"male"	3	"own"	"little"	"moderate"	5234	30
more									
1000	999	27	"male"	2	"own"	"moderate"	"moderate"	4576	45

```
begin
    german[1:5,1:2] #indicies de linha e coluna
    german[1:5, [:Sex , :Age]] #indices de linha e nomes por Symbol das colunas
    german[1:5,:] #apenas as 5 primeiras linhas, mas todas as colunas
    german[[1,6,15] , :]
    german[! , :] # Retorna todas as linhas e colunas do próprio DataFrame (!)
end
```

```
PooledArrays.PooledVector{String, UInt32, Vector{UInt32}}: ["male", "female", "male", "
```

```
begin
    german[:, [:Sex]] #retorna um DataFrame
    german[:, :Sex] #retorna um Array
end
```

	Job	Housing	Saving accounts	Checking account	Credit amount	Time	Purpose
1	2	"own"	"NA"	"little"	1169	6	"radio/TV"
2	2	"own"	"little"	"moderate"	5951	48	"radio/TV"
3	1	"own"	"little"	"NA"	2096	12	"education"
4	2	"free"	"little"	"little"	7882	42	"furniture/equipment'
5	2	"free"	"little"	"little"	4870	24	"car"
6	1	"free"	"NA"	"NA"	9055	36	"education"
7	2	"own"	"quite rich"	"NA"	2835	24	"furniture/equipment'
8	3	"rent"	"little"	"moderate"	6948	36	"car"
9	1	"own"	"rich"	"NA"	3059	12	"radio/TV"
10	3	"own"	"little"	"moderate"	5234	30	"car"
more							
1000	2	"own"	"moderate"	"moderate"	4576	45	"car"

```
begin
    select(german, r"S")
    select(german, Not(["Sex","id","Age"]))
end
```

Acesso condicional

data_frame[condicao,coluna]

	id	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Time
1	0	67	"male"	2	"own"	"NA"	"little"	1169	6
2	2	49	"male"	1	"own"	"little"	"NA"	2096	12
3	3	45	"male"	2	"free"	"little"	"little"	7882	42
4	4	53	"male"	2	"free"	"little"	"little"	4870	24
5	5	35	"male"	1	"free"	"NA"	"NA"	9055	36
6	6	53	"male"	2	"own"	"quite rich"	"NA"	2835	24
7	7	35	"male"	3	"rent"	"little"	"moderate"	6948	36
8	8	61	"male"	1	"own"	"rich"	"NA"	3059	12
9	13	60	"male"	1	"own"	"little"	"little"	1199	24
10	16	53	"male"	2	"own"	"NA"	"NA"	2424	24
	more								
461	997	38	"male"	2	"own"	"little"	"NA"	804	12

```
begin
    ### ACESSO CONDICIONAL ###

    #Retorna apenas as linhas com Age> que 30
    german[german."Age" .> 30,:] #necessário o .> para realizar comparação célula a celula

    german[(german."Age" .> 30) .& (german."Sex" .== "male"),:]

end
```

Manipulação de dados

df1 =

	Age	Sex	Job
1	67	"male"	2
2	22	"female"	2
3	49	"male"	1
4	45	"male"	2
5	53	"male"	2
6	35	"male"	1

```
df1 = german[1:6, 2:4]
```

	Age	Sex	Job
1	80	"male"	2
2	85	"female"	2
3	98	"male"	1
4	95	"male"	2
5	78	"male"	2
6	89	"male"	1

```
begin
    val = [80, 85, 98, 95, 78, 89]
    df1.Age = val #alterar todos os valores da coluna
    view(df1, :,:)
end
```

	Age	Sex	Job
1	80	"male"	10
2	85	"female"	10
3	98	"male"	10
4	95	"male"	2
5	78	"male"	2
6	89	"male"	1

```
begin
    df1[1:3, :Job] = [10, 10, 10] #muda apenas os valores acessados
    view(df1,:,:)
end
```

	Age	Sex	Job
1	80	"male"	10
2	85	"female"	10
3	78	"male"	4
4	95	"transgender"	2
5	78	"female"	2
6	89	"male"	1

```
begin
    df1[!, :Sex] = ["male", "female", "female", "transgender", "female", "male"]
    df1[3, 1:3] = [78, "male", 4] #muda apenas os valores acessados
    view(df1,:,:)
end
```

DataFrameRow (3 columns)

	Age	Sex	Job
	Int64	String	Int64
2	100	male	2

```
begin
    df2 = df1[2, :] #não cria uma cópia de df1. Toda alteração em df2, altera df1
    df2.Age = 100
    df2[2:3] = ["male", 2]
    df2
end
```

	Age	Sex	Job	Customers	City
1	80	"male"	4	"Rohit"	"Kanpur"
2	100	"male"	4	"Akshat"	"Lucknow"
3	78	"male"	4	"Rahul"	"Bhuvneshwar"
4	95	"transgender"	4	"Aayush"	"Jaipur"
5	78	"female"	4	"Prateek"	"Ranchi"
6	89	"male"	4	"Anam"	"Dehradoon"

```
begin
    df1[!, :Customers] = ["Rohit", "Akshat", "Rahul", "Aayush", "Prateek", "Anam"]
    df1[:, :City] = ["Kanpur", "Lucknow", "Bhuvneshwar", "Jaipur", "Ranchi",
"Dehradoon"]
    df1[:, 3] .= 4 #precisa do .= para colocar o valor em todo o array
    view(df1,:,:)
end
```

	Age	Sex	Job	Customers	City
1	"Economics"	"male"	4	"Rohit"	"Kanpur"
2	"Economics"	"male"	4	"Akshat"	"Lucknow"
3	"Economics"	"male"	4	"Rahul"	"Bhuvneshwar"
4	"Economics"	"transgender"	4	"Aayush"	"Jaipur"
5	"Economics"	"female"	4	"Prateek"	"Ranchi"
6	"Economics"	"male"	4	"Anam"	"Dehradoon"

```
begin
    #df1[:, :Age] .= "Economics"
    #ERROR: não pode converter a cópia de df1 em Economics



    df1[!, :Age] .= "Economics" #não dá erro
    view(df1,:,:)
end
```

Inserção de colunas

insertcols!(DataFrame, index_da_coluna, Symbol_Name => valor)

	Country	Age	Sex	Job	Customers	City
1	"India"	"Economics"	"male"	4	"Rohit"	"Kanpur"
2	"India"	"Economics"	"male"	4	"Akshat"	"Lucknow"
3	"India"	"Economics"	"male"	4	"Rahul"	"Bhuvneshwar"
4	"India"	"Economics"	"transgender"	4	"Aayush"	"Jaipur"
5	"India"	"Economics"	"female"	4	"Prateek"	"Ranchi"
6	"India"	"Economics"	"male"	4	"Anam"	"Dehradoon"

```
insertcols!(df1, 1, :Country => "India")
```

Transformações

tranformacao(DataFrameSource, coluna => transformations => coluna_alvo_nome)

---

tranformações:

- combine -> cria um novo DataFrame populado com a transformação
- select -> cria um novo DataFrame com o mesmo número de linhas do Source, e populado com a transformação
- select! -> altera a Source
- transform -> cria um novo DataFrame com o mesmo número de Linhas e Colunas do Source, e popupado com a transformação
- transform! -> altera o Source

---

transformations:

- mean
- unique
- uppercase
- sqrt
- exp
- sin

	mean_age
1	35.546

```
combine(german, :Age => mean => :mean_age)
```

	mean_age
1	35.546
2	35.546
3	35.546
4	35.546
5	35.546
6	35.546
7	35.546
8	35.546
9	35.546
10	35.546
	more
1000	35.546

```
select(german, :Age => mean => :mean_age)
```

	mean_age	housing
1	35.546	"own"
2	35.546	"free"
3	35.546	"rent"

```
combine(german, :Age => mean => :mean_age, :Housing => unique => :housing)
```

	Sex	Sex_UpperCase
1	"male"	"MALE"
2	"female"	"FEMALE"
3	"male"	"MALE"
4	"male"	"MALE"
5	"male"	"MALE"
6	"male"	"MALE"
7	"male"	"MALE"
8	"male"	"MALE"
9	"male"	"MALE"
10	"male"	"MALE"
	more	
1000	"male"	"MALE"

```
begin
    #necessita da função lambda quando acessa os valores da coluna 1 a 1
    select(german, :Sex => (x -> uppercase.(x)) => :Sex)

    #ByRow realiza a transformação célula a celula
    select(german, :Sex ,:Sex => ByRow(uppercase) => :Sex_UpperCase)
end
```

	x1	x2
1	"male"	67
2	"female"	22
3	"male"	49
4	"male"	45
5	"male"	53
6	"male"	35
7	"male"	53
8	"male"	35
9	"male"	61
10	"male"	28
	more	
1000	"male"	27

```
 select(german, :Sex => :x1, :Age => :x2) #apenas crria um dataset igual com nomes diferentes
```

	Age	Job	res
1	67	2	69
2	22	2	24
3	49	1	50
4	45	2	47
5	53	2	55
6	35	1	36
7	53	2	55
8	35	3	38
9	61	1	62
10	28	3	31
	more		
1000	27	2	29

```
select(german, :Age, :Job, [:Age, :Job] => (+) => :res) #soma de coluna
```

	id	Age	Sex	Job	Housing
1	0	67	"male"	2	"own"
2	1	67	"female"	2	"own"
3	2	67	"male"	1	"own"
4	3	67	"male"	2	"free"
5	4	67	"male"	2	"free"
6	5	67	"male"	1	"free"
7	6	67	"male"	2	"own"
8	7	67	"male"	3	"rent"

```
begin
    df = german_ref[1:8, 1:5]
    transform(df, :Age => maximum => "Age")
end
```

	id	Age	Sex	Job	Housing
1	0	"male"	67	2	"own"
2	1	"female"	22	2	"own"
3	2	"male"	49	1	"own"
4	3	"male"	45	2	"free"
5	4	"male"	53	2	"free"
6	5	"male"	35	1	"free"
7	6	"male"	53	2	"own"
8	7	"male"	35	3	"rent"

```
transform(df, :Age => :Sex, :Sex => :Age)
```

	a	b	c
1	0.41124	0.434895	0.675803
2	0.551547	0.110778	0.558085
3	0.0387148	0.175463	0.865529
4	0.00766722	0.55863	0.437181
5	0.799352	0.868233	0.933774
6	0.870858	0.363279	0.819898
7	0.884659	0.880002	0.72473
8	0.476225	0.879316	0.531207
9	0.942984	0.215322	0.0769115
10	0.265046	0.696807	0.154383

```
random_tab
```

	a	b	c	prediction
1	0.41124	0.434895	0.675803	:c
2	0.551547	0.110778	0.558085	:c
3	0.0387148	0.175463	0.865529	:c
4	0.00766722	0.55863	0.437181	:b
5	0.799352	0.868233	0.933774	:c
6	0.870858	0.363279	0.819898	:a
7	0.884659	0.880002	0.72473	:a
8	0.476225	0.879316	0.531207	:b
9	0.942984	0.215322	0.0769115	:a
10	0.265046	0.696807	0.154383	:b

```
transform(random_tab, AsTable(:) => ByRow(argmax) => :prediction)
```

Limpeza de dados

dropmissing(DataFrame, colunas,

	i	x	y
1	1	missing	missing
2	2	4	missing
3	3	missing	"c"
4	4	2	"d"
5	5	1	"e"

```
begin
    data_missing = DataFrame(i = 1:5,
                        x = [missing, 4, missing, 2, 1],
                        y = [missing, missing, "c", "d", "e"])
end
```

	i	x	y
1	2	4	missing
2	4	2	"d"
3	5	1	"e"

```
dropmissing!(data_missing, ["x"], disallowmissing=true ) #remove linhas com valores missing
```

Juntar DataFrames

innerjoin(DF1, DF2, on = coluna_de_juncao)

- **innerjoin** : Retona apenas os valores que tem correspondência entre os 2 dataframes
- **leftjoin** : Retorna todo o Dataframe da esquerda e apenas as correspondências da direita
- **rightjoin** : Retorna todo o Dataframe da direita e apenas as correspondências da esquerda
- **outerjoin** : Retorna todos os valores dos 2 dataframes, mas preenche o que não corresponder com Missing
- **semijoin** : Igual innerjoy, mas apenas com os valores do dataframe da esquerda
- **antijoin** : Retorna apenas os valores do dataframe da esquerda que não tem correspondência com algum dataframe da direita

---

Caso a correspondencia seja em colunas com nomes diferentes em cada dataframe, usa-se:

innerjoin(a, b, on = coluna_a => coluna_b)

innerjoin(a, b, on = :ID => :IDNew)

```
jobs =
```

	ID	Job
1	20	"Lawyer"
2	60	"Doctor"

```
jobs = DataFrame(ID = [20, 60], Job = ["Lawyer", "Doctor"])
```

people =

	ID	Name
1	20	"John Doe"
2	40	"Jane Doe"

```
people = DataFrame(ID = [20, 40], Name = ["John Doe", "Jane Doe"])
```

	ID	Name	Job
1	20	"John Doe"	"Lawyer"

```
innerjoin(people,jobs,on ="ID")
```

	ID	Name	Job
1	20	"John Doe"	"Lawyer"
2	40	"Jane Doe"	missing

```
leftjoin(people, jobs, on= :ID)
```

	ID	Name
1	20	"John Doe"

```
semijoin(people, jobs, on=:ID)
```

	ID	Name
1	40	"Jane Doe"

```
antijoin(people, jobs, on=:ID)
```

Junção por várias colunas:

A:

5 rows × 3 columns

	City String	Job String	Category Int64
1	Amsterdam	Lawyer	1
2	London	Lawyer	2
3	London	Lawyer	3
4	New York	Doctor	4
5	New York	Doctor	5

B

5 rows × 3 columns

	Location String	Work String	Name String
1	Amsterdam	Lawyer	a
2	London	Lawyer	b
3	London	Lawyer	c
4	New York	Doctor	d
5	New York	Doctor	e

```
md"""
Junção por várias colunas:

**A**:
$(A)

**B**
$B

"""
```

	City	Job	Category	Name
1	"Amsterdam"	"Lawyer"	1	"a"
2	"London"	"Lawyer"	2	"b"
3	"London"	"Lawyer"	3	"b"
4	"London"	"Lawyer"	2	"c"
5	"London"	"Lawyer"	3	"c"
6	"New York"	"Doctor"	4	"d"
7	"New York"	"Doctor"	5	"d"
8	"New York"	"Doctor"	4	"e"
9	"New York"	"Doctor"	5	"e"

```
begin
    A = DataFrame(City = ["Amsterdam", "London", "London", "New York", "New York"],
                  Job = ["Lawyer", "Lawyer", "Lawyer", "Doctor", "Doctor"],
                  Category = [1, 2, 3, 4, 5])
    B = DataFrame(Location = ["Amsterdam", "London", "London", "New York", "New
York"],
                  Work = ["Lawyer", "Lawyer", "Lawyer", "Doctor", "Doctor"],
                  Name = ["a", "b", "c", "d", "e"])
    innerjoin(A,B, on= [:City=>:Location, :Job=>:Work])
end
```

Separação - Transformação - Combinação

Agrupamentos

iris =

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	5.1	3.5	1.4	0.2	"Iris-setosa"
2	4.9	3.0	1.4	0.2	"Iris-setosa"
3	4.7	3.2	1.3	0.2	"Iris-setosa"
4	4.6	3.1	1.5	0.2	"Iris-setosa"
5	5.0	3.6	1.4	0.2	"Iris-setosa"
6	5.4	3.9	1.7	0.4	"Iris-setosa"
7	4.6	3.4	1.4	0.3	"Iris-setosa"
8	5.0	3.4	1.5	0.2	"Iris-setosa"
9	4.4	2.9	1.4	0.2	"Iris-setosa"
10	4.9	3.1	1.5	0.1	"Iris-setosa"
	more				
150	5.9	3.0	5.1	1.8	"Iris-virginica"

```
iris = CSV.read((joinpath(dirname(pathof(DataFrames)),
                          "..", "docs", "src", "assets", "iris.csv")),
                DataFrame)
```

GROUPBY

Separa o dataframe em subgrupos dependendo do agrupamento que fizer.Cada grupo terá um dos valores da coluna especificada.

groupby(DataFrame, coluna)

`gdf =`

GroupedDataFrame with 3 groups based on key: Species

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

	SepalLength Float64	SepalWidth Float64	PetalLength Float64	PetalWidth Float64	Species String
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5.0	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
13	4.8	3.0	1.4	0.1	Iris-setosa
14	4.3	3.0	1.1	0.1	Iris-setosa
15	5.8	4.0	1.2	0.2	Iris-setosa
16	5.7	4.4	1.5	0.4	Iris-setosa
17	5.4	3.9	1.3	0.4	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa
⋮	⋮	⋮	⋮	⋮	⋮

⋮

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

	SepalLength Float64	SepalWidth Float64	PetalLength Float64	PetalWidth Float64	Species String
1	6.3	3.3	6.0	2.5	Iris-virginica
2	5.8	2.7	5.1	1.9	Iris-virginica
3	7.1	3.0	5.9	2.1	Iris-virginica
4	6.3	2.9	5.6	1.8	Iris-virginica
5	6.5	3.0	5.8	2.2	Iris-virginica
6	7.6	3.0	6.6	2.1	Iris-virginica
7	4.9	2.5	4.5	1.7	Iris-virginica
8	7.3	2.9	6.3	1.8	Iris-virginica
9	6.7	2.5	5.8	1.8	Iris-virginica

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
	Float64	Float64	Float64	Float64	String
10	7.2	3.6	6.1	2.5	Iris-virginica
11	6.5	3.2	5.1	2.0	Iris-virginica
12	6.4	2.7	5.3	1.9	Iris-virginica
13	6.8	3.0	5.5	2.1	Iris-virginica
14	5.7	2.5	5.0	2.0	Iris-virginica
15	5.8	2.8	5.1	2.4	Iris-virginica
16	6.4	3.2	5.3	2.3	Iris-virginica
17	6.5	3.0	5.5	1.8	Iris-virginica
18	7.7	3.8	6.7	2.2	Iris-virginica
⋮	⋮	⋮	⋮	⋮	⋮

```
gdf = groupby(iris, :Species)
```

GroupedDataFrame with 2 groups based on key: Species

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

	SepalLength Float64	SepalWidth Float64	PetalLength Float64	PetalWidth Float64	Species String
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5.0	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
13	4.8	3.0	1.4	0.1	Iris-setosa
14	4.3	3.0	1.1	0.1	Iris-setosa
15	5.8	4.0	1.2	0.2	Iris-setosa
16	5.7	4.4	1.5	0.4	Iris-setosa
17	5.4	3.9	1.3	0.4	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa
⋮	⋮	⋮	⋮	⋮	⋮

⋮

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

	SepalLength Float64	SepalWidth Float64	PetalLength Float64	PetalWidth Float64	Species String
1	6.3	3.3	6.0	2.5	Iris-virginica
2	5.8	2.7	5.1	1.9	Iris-virginica
3	7.1	3.0	5.9	2.1	Iris-virginica
4	6.3	2.9	5.6	1.8	Iris-virginica
5	6.5	3.0	5.8	2.2	Iris-virginica
6	7.6	3.0	6.6	2.1	Iris-virginica
7	4.9	2.5	4.5	1.7	Iris-virginica
8	7.3	2.9	6.3	1.8	Iris-virginica
9	6.7	2.5	5.8	1.8	Iris-virginica

	SepalLength Float64	SepalWidth Float64	PetalLength Float64	PetalWidth Float64	Species String
10	7.2	3.6	6.1	2.5	Iris-virginica
11	6.5	3.2	5.1	2.0	Iris-virginica
12	6.4	2.7	5.3	1.9	Iris-virginica
13	6.8	3.0	5.5	2.1	Iris-virginica
14	5.7	2.5	5.0	2.0	Iris-virginica
15	5.8	2.8	5.1	2.4	Iris-virginica
16	6.4	3.2	5.3	2.3	Iris-virginica
17	6.5	3.0	5.5	1.8	Iris-virginica
18	7.7	3.8	6.7	2.2	Iris-virginica
⋮	⋮	⋮	⋮	⋮	⋮

```
gdf[ [("Iris-setosa",) , ("Iris-virginica",) ]] #pega apenas 2 dos 3 grupos
```

Tranformações em grupos

Combine

	Species	PetalLength_mean
1	"Iris-setosa"	1.464
2	"Iris-versicolor"	4.26
3	"Iris-virginica"	5.552

```
combine(gdf, :PetalLength=>mean)
```

	Species	nrow
1	"Iris-setosa"	50
2	"Iris-versicolor"	50
3	"Iris-virginica"	50

```
combine(gdf, nrow) #retorna o número de linhas por grupo
```

	Species	NumRows	Mean
1	"Iris-setosa"	50	1.464
2	"Iris-versicolor"	50	4.26
3	"Iris-virginica"	50	5.552

```
combine(gdf, nrow=>:NumRows , :PetalLength =>mean => :Mean)
```

	Species	a	b
1	"Iris-setosa"	0.292449	73.2
2	"Iris-versicolor"	0.717655	213.0
3	"Iris-virginica"	0.842744	277.6

```
combine(gdf, [:PetalLength, :SepalLength] => ((p, s) -> (a=mean(p)/mean(s), b=sum(p))) => AsTable)
#poderia ser passado para um vetor de colunas também
#combine(gdf, [:PetalLength, :SepalLength] => ((p, s) -> (a=mean(p)/mean(s), b=sum(p))) => [:ColA, :ColB])
```

	Species	PetalLength_SepalLength_function
1	"Iris-setosa"	0.492245
2	"Iris-versicolor"	0.910378
3	"Iris-virginica"	0.867923

```
#valor AsTable, passa um dataframe como parâmetro
combine(gdf,
                AsTable([:PetalLength, :SepalLength]) =>
                x -> std(x.PetalLength) / std(x.SepalLength))
```

	Species	min	max
1	"Iris-setosa"	1.0	1.9
2	"Iris-versicolor"	3.0	5.1
3	"Iris-virginica"	4.5	6.9

```
combine(gdf, :PetalLength => (x -> [extrema(x)]) => [:min, :max])
#retorna os minimos e máximos na forma tuple para cada grupo
```

	Species	PetalWidth_unique
1	"Iris-setosa"	0.2
2	"Iris-setosa"	0.4
3	"Iris-setosa"	0.3
4	"Iris-setosa"	0.1
5	"Iris-setosa"	0.5
6	"Iris-setosa"	0.6
7	"Iris-versicolor"	1.4
8	"Iris-versicolor"	1.5
9	"Iris-versicolor"	1.3
10	"Iris-versicolor"	1.6
	more	
27	"Iris-virginica"	1.4

```
combine(gdf, :PetalWidth=>unique)
```

Select e Transform

```
md" #### Select e Transform"
```

	Species	SepalLength_SepalWidth_cor
1	"Iris-setosa"	0.74678
2	"Iris-setosa"	0.74678
3	"Iris-setosa"	0.74678
4	"Iris-setosa"	0.74678
5	"Iris-setosa"	0.74678
6	"Iris-setosa"	0.74678
7	"Iris-setosa"	0.74678
8	"Iris-setosa"	0.74678
9	"Iris-setosa"	0.74678
10	"Iris-setosa"	0.74678
	more	
150	"Iris-virginica"	0.457228

```
select(gdf, 1:2 => cor)
```

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species	Species_funct
1	5.1	3.5	1.4	0.2	"Iris-setosa"	"setosa"
2	4.9	3.0	1.4	0.2	"Iris-setosa"	"setosa"
3	4.7	3.2	1.3	0.2	"Iris-setosa"	"setosa"
4	4.6	3.1	1.5	0.2	"Iris-setosa"	"setosa"
5	5.0	3.6	1.4	0.2	"Iris-setosa"	"setosa"
6	5.4	3.9	1.7	0.4	"Iris-setosa"	"setosa"
7	4.6	3.4	1.4	0.3	"Iris-setosa"	"setosa"
8	5.0	3.4	1.5	0.2	"Iris-setosa"	"setosa"
9	4.4	2.9	1.4	0.2	"Iris-setosa"	"setosa"
10	4.9	3.1	1.5	0.1	"Iris-setosa"	"setosa"
	more					
150	5.9	3.0	5.1	1.8	"Iris-virginica"	"virginica"

```
transform(gdf, :Species => x -> chop.(x, head=5, tail=0))
#função chop(string, head=n, tail = m)
#remove o primeiros n caracteres da string, e os ultimos m caracteres da string
```

Pode-se selecionar grupos desejados a partir de tuples com o valor de separação dos grupos

Stack

Tranforma uma tabela curta em tablea longa a partir das colunas selecionada.

Cada linha terá o valor da coluna

```
md" ### Stack

Tranforma uma tabela curta em tablea longa a partir das colunas selecionada.

Cada linha terá o valor da coluna
"
```

```
stacked = stack(iris, 1:4) #faz um empilhamento das coluna 1:4 (todas menos :value);
```

medias_stacked =

	Species	variable	media
1	"Iris-setosa"	"SepalLength"	5.006
2	"Iris-setosa"	"SepalWidth"	3.418
3	"Iris-setosa"	"PetalLength"	1.464
4	"Iris-setosa"	"PetalWidth"	0.244
5	"Iris-versicolor"	"SepalLength"	5.936
6	"Iris-versicolor"	"SepalWidth"	2.77
7	"Iris-versicolor"	"PetalLength"	4.26
8	"Iris-versicolor"	"PetalWidth"	1.326
9	"Iris-virginica"	"SepalLength"	6.588
10	"Iris-virginica"	"SepalWidth"	2.974
11	"Iris-virginica"	"PetalLength"	5.552
12	"Iris-virginica"	"PetalWidth"	2.026

```
medias_stacked = combine(groupby(stacked, [:Species,:variable]), :value=>mean=>:media)
```

	variable	Iris-setosa	Iris-versicolor	Iris-virginica
1	"SepalLength"	5.006	5.936	6.588
2	"SepalWidth"	3.418	2.77	2.974
3	"PetalLength"	1.464	4.26	5.552
4	"PetalWidth"	0.244	1.326	2.026

```
unstack(medias_stacked,:Species,:media)
```

Ordenamento

```
md" ### Ordenamento"
```

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	4.3	3.0	1.1	0.1	"Iris-setosa"
2	4.4	2.9	1.4	0.2	"Iris-setosa"
3	4.4	3.0	1.3	0.2	"Iris-setosa"
4	4.4	3.2	1.3	0.2	"Iris-setosa"
5	4.5	2.3	1.3	0.3	"Iris-setosa"
6	4.6	3.1	1.5	0.2	"Iris-setosa"
7	4.6	3.2	1.4	0.2	"Iris-setosa"
8	4.6	3.4	1.4	0.3	"Iris-setosa"
9	4.6	3.6	1.0	0.2	"Iris-setosa"
10	4.7	3.2	1.3	0.2	"Iris-setosa"
	more				
150	7.9	3.8	6.4	2.0	"Iris-virginica"

```
sort(iris)
```

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	7.9	3.8	6.4	2.0	"Iris-virginica"
2	7.7	3.8	6.7	2.2	"Iris-virginica"
3	7.7	3.0	6.1	2.3	"Iris-virginica"
4	7.7	2.8	6.7	2.0	"Iris-virginica"
5	7.7	2.6	6.9	2.3	"Iris-virginica"
6	7.6	3.0	6.6	2.1	"Iris-virginica"
7	7.4	2.8	6.1	1.9	"Iris-virginica"
8	7.3	2.9	6.3	1.8	"Iris-virginica"
9	7.2	3.6	6.1	2.5	"Iris-virginica"
10	7.2	3.2	6.0	1.8	"Iris-virginica"
	more				
150	4.3	3.0	1.1	0.1	"Iris-setosa"

```
sort(iris, rev=true) #ordena ao contrário
```