1 Ein- und Ausgabe I/O

zahlreiche Varianten und Funktionen, das Folgende ist eine kleine Auswahl

1.1 Konsole

- OS-abhängig; üblicherweise 3 Kanäle *(streams)*: `stdin, stdout, stderr` (Standardinput, -output, -errorkanal)
- Schreiben nach `stdout`: `print()`,`println()`,`printstyled()`
- Schreiben nach `stderr`: `print(strerr,...)`, `println(stderr,...)`, `printstyled(stderr,...)`
- Lesen von `stdin`: `readline()`

1.1.1 Umwandeln von Strings in andere Typen:

- `chomp()` entfernt newline
- `split()` zerlegt in "Wörter"
- `parse()` wandelt in andere Typen um

1.1.2 Buffer

- `write`-Zugriffe werden gebuffert.
  - `flush(stdout)` leert Buffer

[1]:
```
# aus dem ersten Beispielprogramm

function input(prompt = "Eingabe:")
    println(prompt)
    flush(stdout)
    return chomp(readline())
end
```

[1]:
```
input (generic function with 2 methods)
```

[2]:
```
a = input("Bitte 2 Zahlen eingeben!")
```

```
Bitte 2 Zahlen eingeben!
stdin> 33 67
```

[2]:
```
"33 67"
```

[3]:
```
av = split(a)
```

[3]:
```
2-element Vector{SubString{String}}:
 "33"
 "67"
```

[4]:
```
parse.(Int, av)
```

[4]:
```
2-element Vector{Int64}:
 33
 67
```

[5]:
```
# Ausgaben auf den Fehlerkanal stderr erscheinen im Jupyter in rot:

println(stderr, "Das sollte nicht passieren!")
```

```
Das sollte nicht passieren!
```

1.1.3 Einzelne Tastenanschläge einlesen

- `readline()` u.ä. warten auf den Abschluss der Eingabe durch Drücken der `Enter`-Taste.
- Zum Einlesen einzelner *keystrokes*:
  - https://stackoverflow.com/questions/56888266/how-to-read-keyboard-inputs-at-every-keystroke-in-julia
  - https://stackoverflow.com/questions/60954235/how-can-i-test-whether-stdin-has-input-available-in-juli

1.2 Formatierte Ausgabe mit dem `Printf`-Paket

Die Macros `@sprintf` und `@printf` sind den gleichnamigen C-Funktionen nachempfunden

- Formatstring: Normaler String mit Platzhaltern
- Platzhalter haben die Form

```
%[flags][width][.precision]type
```

(wobei die Angaben in eckigen Klammern alle optional sind) - Typen:

```
%s      string
%i      integer
%o      integer octal (base=8)
%x, %X  integer hexadecimal (base=16) with digits 0-9abcdef  or 0-9ABCDEF, resp.
%f      floatong point number
%e      floating point number, scientific representation
%g      floating point, uses %f or %e depending on value
```

- Flags:

```
Pluszeichen:    rechtsbündig (Standard)
Minuszeichen:   linksbündig
Null:           mit führenden Nullen
```

- Width:

```
Anzahl der minimal verwendeten Zeichen (wenn nötig, werden auch mehr genommen)
```

Zeit für Beispiele:

[6]:
```
using Printf    # Paket laden nicht vergessen!
```

[7]:
```
@printf("|%s|", "Hallo")     # string mit Platzhalter für String
```

```
|Hallo|
```

[8]:
```
@printf("|%10s|", "Hallo")   # Minimallänge, rechtsbündig
```

```
|     Hallo|
```

[9]:
```
@printf("|%-10s|", "Hallo")     # linksbündig
```

```
|Hallo     |
```

[10]:
```
@printf("|%3s|", "Hallo")     # Längenangabe kann überschritten werden
                              # besser eine 'kaputt formatierte' Tabelle als falsche Werte!!
```

```
|Hallo|
```

[11]:
```
j = 123
k = 90019001
l = 3342678

@printf("j= %012i, k= %-12i, l = %12i", j, k, l)   #  0-Flag für führende Nullen
```

```
j= 000000000123, k= 90019001    , l =      3342678
```

`@printf` und `@sprintf` können wie alle Macros wie Funktionen aufgerufen werden:

[12]:
```
@printf("%i %i", 22, j)
```

```
22 123
```

– oder wie Macros, also ohne Funktionsklammern und ohne Komma:

[13]:
```
@printf "%i %i" 22 j
```

```
22 123
```

`@printf` kann als erstes Argument noch einen stream übergeben bekommen.

Ansonsten besteht die Argumentliste aus

- Formatstring mit Platzhaltern
- Variablen in der Reihenfolge der Platzhalter, in Anzahl und Typ zu den Platzhaltern passend

[14]:
```
@printf(stderr, "Erstes Resultat: %i %s\nZweites Resultat %i",
                                  j, "(geschätzt)"       ,k)
```

```
Erstes Resultat: 123 (geschätzt)
Zweites Resultat 90019001
```

`@sprintf` druckt nichts, sondern liefert den ausgefüllten formatierten String zurück:

[15]:
```
str = @sprintf("x = %10.6f", π );
```

[16]:
```
str
```

[16]:
```
"x =   3.141593"
```

Formatierung der Gleitkommazahlen: Bedeutung des *Precision*-Wertes:

- `%f` und `%e`-Format: max. Anzahl der Nachkommastellen
- `%g`-Format: max. Anzahl von ausgegebenen Ziffern (Vor- + Nachkommastellen)

[17]:
```
x = 123456.7890123456

@printf("%20.4f   %20.4e", x, x)     # 4 Nachkommastellen
```

```
         123456.7890             1.2346e+05
```

[18]:
```
@printf("%20.7f %20.7e", x, x)     # 7 Nachkommastellen
```

```
      123456.7890123        1.2345679e+05
```

[19]:
```
@printf("%20.7g %20.4g", x, x)    # insgesamt 7 bzw. 4 Stellen
```

```
            123456.8            1.235e+05
```

1.3 Dateioperationen

Dateien werden

- geöffnet ==> Dabei ensteht ein neues *stream*-Objekt (zusätzlich zu `stdin, stdout, stderr`)
- dann kann dieser *stream* gelesen und geschrieben werden
- geschlossen ==> *stream*-Objekt wird von Datei getrennt

`stream = open(path, mode)`

- path: Dateiname/pfad
- mode:

```
"r"    read, öffnet am Dateianfang
"w"    write, öffnet am Dateianfang (Datei wird neu angelegt oder überschrieben)
"a"    append, öffnet zum Weiterschreiben am Dateiende
```

[20]:
```
f = open("datei.txt", "w")
```

[20]:
```
IOStream(<file datei.txt>)
```

[21]:
```
@printf(f, "%20i\n", k)
```

[22]:
```
println(f, " zweite Zeile")
```

[23]:
```
close(f)
```

[24]:
```
;cat datei.txt
```

```
           90019001
 zweite Zeile
```

[25]:
```
f = open("datei.txt", "r")
```

[25]:
```
IOStream(<file datei.txt>)
```

[26]:
```
n = 0
for i in readlines(f)    # Lese zeilenweise
    n += 1
    println(n, i)       # Drucke mit Zeilennummer
end
```

```
1           90019001
2 zweite Zeile
```

1.4 Pakete für Dateiformate

Pakete für die Ein- und Ausgabe in den verschiedensten Dateiformaten

- PrettyTables.jl Ausgabe von formatierten Tabellen
- DelimitedFiles.jl Ein- und Ausgabe von Matrizen u.ä.
- CSV.jl Ein- und Ausgabe von Dateien mit “comma-delimited values” u.ä.
- XLSX.jl Ein- und Ausgabe von Excel-Dateien

und viele andere mehr…

1.4.1 Delimited Files

[27]:
```
using DelimitedFiles
```

[28]:
```
A = rand(200,3)
```

[28]:
```
200×3 Matrix{Float64}:
 0.340458   0.825769   0.10208
 0.0268893  0.949268   0.908559
 0.649896   0.395813   0.353179
 0.14032    0.291844   0.830224
 0.63363    0.276789   0.831877
 0.524021   0.246762   0.419489
 0.585347   0.880953   0.608495
 0.777373   0.387001   0.0595369
 0.347182   0.991371   0.72133
 0.418528   0.478036   0.955822
 0.602542   0.84481    0.755724
 0.19726    0.48635    0.223986
 0.488213   0.32438    0.640762
 ⋮
 0.0109543  0.464666   0.0338036
 0.424558   0.739999   0.69583
 0.958684   0.432141   0.809327
 0.562518   0.273888   0.232715
 0.779498   0.0941955  0.447725
 0.65658    0.236674   0.0702908
 0.240148   0.216886   0.31698
```

```
 0.878407   0.453928   0.715879
 0.387353   0.864377   0.0140109
 0.818247   0.30614    0.724261
 0.699524   0.204064   0.308865
 0.07843    0.104628   0.715755
```

[29]:
```
f = open("data2.txt", "w")
```

[29]:
```
IOStream(<file data2.txt>)
```

[30]:
```
writedlm(f, A)
```

[31]:
```
close(f)
```

[32]:
```
;head data2.txt
```

```
0.3404584106811863      0.8257693260643905      0.10208011760128599
0.026889328033128268    0.9492675653854479      0.9085591450702581
0.6498963717719178      0.3958125045190971      0.35317912384984795
0.14032004382389018     0.2918439613799869      0.8302239965868584
0.633630005283942       0.2767888491501992      0.8318766514610799
0.5240208117170729      0.24676197726593674     0.41948911446627135
0.585346515700062       0.8809527810080108      0.6084948567669999
0.7773727819682476      0.38700114682941644     0.05953685007070941
0.34718210294604357     0.9913709382217284      0.7213300130637681
0.41852821927063044     0.47803588615962034     0.9558221787263325
```

[33]:
```
B = readdlm("data2.txt")
```

[33]:
```
200×3 Matrix{Float64}:
 0.340458   0.825769   0.10208
 0.0268893  0.949268   0.908559
 0.649896   0.395813   0.353179
 0.14032    0.291844   0.830224
 0.63363    0.276789   0.831877
 0.524021   0.246762   0.419489
 0.585347   0.880953   0.608495
 0.777373   0.387001   0.0595369
 0.347182   0.991371   0.72133
 0.418528   0.478036   0.955822
 0.602542   0.84481    0.755724
 0.19726    0.48635    0.223986
 0.488213   0.32438    0.640762
 ⋮
 0.0109543  0.464666   0.0338036
 0.424558   0.739999   0.69583
 0.958684   0.432141   0.809327
 0.562518   0.273888   0.232715
 0.779498   0.0941955  0.447725
 0.65658    0.236674   0.0702908
 0.240148   0.216886   0.31698
 0.878407   0.453928   0.715879
 0.387353   0.864377   0.0140109
 0.818247   0.30614    0.724261
 0.699524   0.204064   0.308865
 0.07843    0.104628   0.715755
```

[34]:
```
# man kann open() auch als 1.Argument eine function(iostream) übergeben, die auf den stream
# angewendet wird, wonach der stream automatisch geschlosssen wird.
#
# Mit der do-Notation sieht obiger code so aus:
```

```
open("data2.txt", "w") do io
    writedlm(io, A)
end
```

1.4.2 CSV und DataFrames

- DataFrames.jl ist ein Paket zum bequemen Umgang mit tabellarischen Daten

[35]:
```
using CSV, DataFrames, Downloads
```

[36]:
```
# Wetterdaten von Westerland, s. https://dev.meteostat.net/bulk/hourly.html


url = "https://bulk.meteostat.net/v2/hourly/10018.csv.gz"
```

[36]:
```
"https://bulk.meteostat.net/v2/hourly/10018.csv.gz"
```

[37]:
```
http_response = Downloads.download(url)
```

[37]:
```
"/tmp/jl_R2uC9y"
```

[38]:
```
file = CSV.File(http_response, header=false)
```

[38]:
```
161343-element CSV.File:
 CSV.Row: (Column1 = Dates.Date("1989-03-18"), Column2 = 7, Column3 = 6.0,
Column4 = missing, Column5 = missing, Column6 = missing, Column7 = missing,
Column8 = 270, Column9 = 11.2, Column10 = missing, Column11 = missing, Column12
= missing, Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("1989-03-18"), Column2 = 8, Column3 = 6.0,
Column4 = -1.9, Column5 = 57, Column6 = missing, Column7 = missing, Column8 =
250, Column9 = 11.2, Column10 = missing, Column11 = missing, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("1989-03-18"), Column2 = 9, Column3 = 7.0,
Column4 = -3.0, Column5 = 49, Column6 = missing, Column7 = missing, Column8 =
250, Column9 = 14.8, Column10 = missing, Column11 = missing, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("1989-03-18"), Column2 = 10, Column3 = 7.0,
Column4 = -1.9, Column5 = 53, Column6 = missing, Column7 = missing, Column8 =
250, Column9 = 18.4, Column10 = missing, Column11 = missing, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("1989-03-18"), Column2 = 11, Column3 = 8.0,
Column4 = -1.0, Column5 = 53, Column6 = missing, Column7 = missing, Column8 =
220, Column9 = 24.1, Column10 = missing, Column11 = missing, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("1989-03-18"), Column2 = 12, Column3 = 8.0,
Column4 = 0.0, Column5 = 57, Column6 = missing, Column7 = missing, Column8 =
240, Column9 = 27.7, Column10 = missing, Column11 = missing, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("1989-03-18"), Column2 = 13, Column3 = 8.0,
Column4 = -1.0, Column5 = 53, Column6 = missing, Column7 = missing, Column8 =
240, Column9 = 25.9, Column10 = missing, Column11 = missing, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("1989-03-18"), Column2 = 14, Column3 = 8.0,
Column4 = 0.0, Column5 = 57, Column6 = missing, Column7 = missing, Column8 =
230, Column9 = 29.5, Column10 = missing, Column11 = missing, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("1989-03-18"), Column2 = 15, Column3 = 8.0,
Column4 = 0.0, Column5 = 57, Column6 = missing, Column7 = missing, Column8 =
230, Column9 = 31.7, Column10 = missing, Column11 = missing, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("1989-03-23"), Column2 = 9, Column3 = 7.0,
Column4 = -0.9, Column5 = 57, Column6 = missing, Column7 = missing, Column8 =
```

```
280, Column9 = 37.1, Column10 = missing, Column11 = missing, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("1989-03-23"), Column2 = 10, Column3 = 7.0,
Column4 = -0.9, Column5 = 57, Column6 = missing, Column7 = missing, Column8 =
270, Column9 = 64.8, Column10 = missing, Column11 = missing, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("1989-03-23"), Column2 = 11, Column3 = 7.0,
Column4 = -0.9, Column5 = 57, Column6 = missing, Column7 = missing, Column8 =
280, Column9 = 33.5, Column10 = missing, Column11 = missing, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("1989-03-27"), Column2 = 7, Column3 = 6.0,
Column4 = 4.0, Column5 = 87, Column6 = missing, Column7 = missing, Column8 =
160, Column9 = 11.2, Column10 = missing, Column11 = missing, Column12 = missing,
Column13 = missing)
 ⋮
 CSV.Row: (Column1 = Dates.Date("2022-06-20"), Column2 = 0, Column3 = 15.0,
Column4 = 11.2, Column5 = 78, Column6 = missing, Column7 = missing, Column8 =
300, Column9 = 30.2, Column10 = missing, Column11 = 1012.5, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("2022-06-20"), Column2 = 6, Column3 = 13.1,
Column4 = 9.4, Column5 = 78, Column6 = missing, Column7 = missing, Column8 =
303, Column9 = 25.2, Column10 = missing, Column11 = 1012.2, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("2022-06-20"), Column2 = 12, Column3 = 14.3,
Column4 = 10.1, Column5 = 76, Column6 = missing, Column7 = missing, Column8 =
294, Column9 = 28.1, Column10 = missing, Column11 = 1011.5, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("2022-06-20"), Column2 = 18, Column3 = 18.0,
Column4 = 13.9, Column5 = 77, Column6 = missing, Column7 = missing, Column8 =
300, Column9 = 26.3, Column10 = missing, Column11 = 1010.4, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("2022-06-21"), Column2 = 0, Column3 = 14.9,
Column4 = 10.9, Column5 = 77, Column6 = missing, Column7 = missing, Column8 =
297, Column9 = 21.6, Column10 = missing, Column11 = 1012.1, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("2022-06-21"), Column2 = 6, Column3 = 13.4,
Column4 = 9.8, Column5 = 79, Column6 = missing, Column7 = missing, Column8 =
276, Column9 = 20.9, Column10 = missing, Column11 = 1010.5, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("2022-06-21"), Column2 = 12, Column3 = 14.5,
Column4 = 10.9, Column5 = 79, Column6 = missing, Column7 = missing, Column8 =
277, Column9 = 22.0, Column10 = missing, Column11 = 1010.9, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("2022-06-21"), Column2 = 18, Column3 = 18.1,
Column4 = 14.6, Column5 = 80, Column6 = missing, Column7 = missing, Column8 =
286, Column9 = 21.6, Column10 = missing, Column11 = 1009.9, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("2022-06-22"), Column2 = 0, Column3 = 15.2,
Column4 = 11.6, Column5 = 79, Column6 = missing, Column7 = missing, Column8 =
277, Column9 = 19.4, Column10 = missing, Column11 = 1009.5, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("2022-06-22"), Column2 = 6, Column3 = 13.2,
Column4 = 9.6, Column5 = 79, Column6 = missing, Column7 = missing, Column8 =
274, Column9 = 18.0, Column10 = missing, Column11 = 1008.0, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("2022-06-22"), Column2 = 12, Column3 = 15.0,
Column4 = 11.2, Column5 = 78, Column6 = missing, Column7 = missing, Column8 =
266, Column9 = 20.2, Column10 = missing, Column11 = 1008.0, Column12 = missing,
Column13 = missing)
 CSV.Row: (Column1 = Dates.Date("2022-06-22"), Column2 = 18, Column3 = 18.8,
Column4 = 15.1, Column5 = 79, Column6 = missing, Column7 = missing, Column8 =
```

```
262, Column9 = 15.8, Column10 = missing, Column11 = 1006.0, Column12 = missing,
Column13 = missing)
```

[39]:
```
# https://dev.meteostat.net/bulk/hourly.html#endpoints

# Spalte 1  Datum
#        2  Uhrzeit (Stunde)
#        3  Temp
#        5  Luftfeuchtigkeit
#        6  Niederschlag
#        8 Windrichtung
#        9 Windstärke


df =  DataFrame(file)
```

[39]:

	Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	
	Date	Int64	Float64?	Float64?	Int64?	Float64?	Missing	Int64?	Float64?	
1	1989-03-18	7	6.0	*missing*	*missing*	*missing*	*missing*	270	11.2	…
2	1989-03-18	8	6.0	-1.9	57	*missing*	*missing*	250	11.2	…
3	1989-03-18	9	7.0	-3.0	49	*missing*	*missing*	250	14.8	…
4	1989-03-18	10	7.0	-1.9	53	*missing*	*missing*	250	18.4	…
5	1989-03-18	11	8.0	-1.0	53	*missing*	*missing*	220	24.1	…
6	1989-03-18	12	8.0	0.0	57	*missing*	*missing*	240	27.7	…
7	1989-03-18	13	8.0	-1.0	53	*missing*	*missing*	240	25.9	…
8	1989-03-18	14	8.0	0.0	57	*missing*	*missing*	230	29.5	…
9	1989-03-18	15	8.0	0.0	57	*missing*	*missing*	230	31.7	…
10	1989-03-23	9	7.0	-0.9	57	*missing*	*missing*	280	37.1	…
11	1989-03-23	10	7.0	-0.9	57	*missing*	*missing*	270	64.8	…
12	1989-03-23	11	7.0	-0.9	57	*missing*	*missing*	280	33.5	…
13	1989-03-27	7	6.0	4.0	87	*missing*	*missing*	160	11.2	…
14	1989-03-27	8	8.0	4.9	81	*missing*	*missing*	160	14.8	…
15	1989-03-27	9	*missing*	*missing*	*missing*	*missing*	*missing*	160	14.8	…
16	1989-03-27	10	*missing*	*missing*	*missing*	*missing*	*missing*	150	14.8	…
17	1989-03-27	11	*missing*	*missing*	*missing*	*missing*	*missing*	150	22.3	…
18	1989-03-27	12	*missing*	*missing*	*missing*	*missing*	*missing*	140	27.7	…
19	1989-03-27	14	*missing*	*missing*	*missing*	*missing*	*missing*	140	29.5	…
20	1989-03-31	8	10.0	6.0	76	*missing*	*missing*	270	18.4	…
21	1989-03-31	9	10.0	6.0	76	*missing*	*missing*	270	14.8	…
22	1989-03-31	10	10.0	6.0	76	*missing*	*missing*	280	18.4	…
23	1989-03-31	11	11.0	5.9	71	*missing*	*missing*	280	18.4	…
24	1989-03-31	12	11.0	5.9	71	*missing*	*missing*	280	18.4	…
25	1989-03-31	13	11.0	5.9	71	*missing*	*missing*	290	18.4	…
26	1989-04-01	7	7.0	1.1	66	*missing*	*missing*	80	37.1	…
27	1989-04-01	8	8.0	2.0	66	*missing*	*missing*	70	27.7	…
28	1989-04-01	9	8.0	0.9	61	*missing*	*missing*	90	33.5	…
29	1989-04-01	10	9.0	0.9	57	*missing*	*missing*	80	38.9	…
30	1989-04-01	11	9.0	-2.0	46	*missing*	*missing*	70	33.5	…
…	…	…	…	…	…	…	…	…	…	

[40]:
```
describe(df)
```

[40]:

	variable	mean	min	median	max	nmissing	eltype
	Symbol	Union…	Any	Any	Any	Int64	Type
1	Column1		1989-03-18	2010-09-21	2022-06-22	0	Date
2	Column2	11.7822	0	12.0	23	0	Int64
3	Column3	10.5158	-14.0	11.0	34.0	1250	Union{Missing, Float64}
4	Column4	6.91075	-18.1	7.0	25.0	1603	Union{Missing, Float64}
5	Column5	79.9007	18	82.0	100	1603	Union{Missing, Int64}
6	Column6	0.0932762	0.0	0.0	5.0	125723	Union{Missing, Float64}
7	Column7					161343	Missing
8	Column8	208.597	0	230.0	360	2216	Union{Missing, Int64}
9	Column9	22.9206	0.0	22.3	211.3	304	Union{Missing, Float64}
10	Column10	37.1644	5.5	35.2	105.6	127105	Union{Missing, Float64}
11	Column11	1014.08	970.0	1015.0	1049.0	116873	Union{Missing, Float64}
12	Column12					161343	Missing
13	Column13	4.58144	0	4.0	25	124875	Union{Missing, Int64}

[41]:
```
using StatsPlots
```

[42]:
```
using Dates
```

[43]:
```
# neue Spalte mit Sp.1 und 2  (date & time) kombiniert

df[!, :datetime] = DateTime.(df.Column1) .+ Hour.(df.Column2);
```

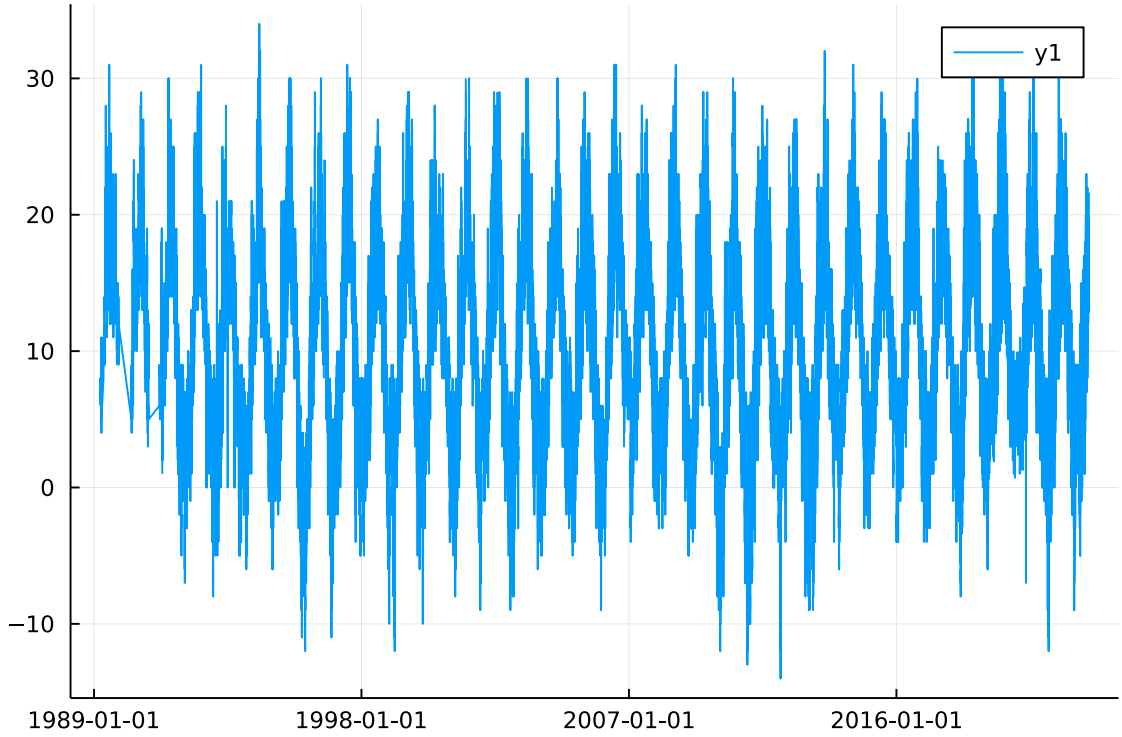
[44]:
```
describe(df)
```

[44]:

	variable	mean	min	median	max	
	Symbol	Union…	Any	Any	Any	
1	Column1		1989-03-18	2010-09-21	2022-06-22	…
2	Column2	11.7822	0	12.0	23	…
3	Column3	10.5158	-14.0	11.0	34.0	…
4	Column4	6.91075	-18.1	7.0	25.0	…
5	Column5	79.9007	18	82.0	100	…
6	Column6	0.0932762	0.0	0.0	5.0	…
7	Column7					…
8	Column8	208.597	0	230.0	360	…
9	Column9	22.9206	0.0	22.3	211.3	…
10	Column10	37.1644	5.5	35.2	105.6	…
11	Column11	1014.08	970.0	1015.0	1049.0	…
12	Column12					…
13	Column13	4.58144	0	4.0	25	…
14	datetime		1989-03-18T07:00:00	2010-09-21T21:00:00	2022-06-22T18:00:00	…

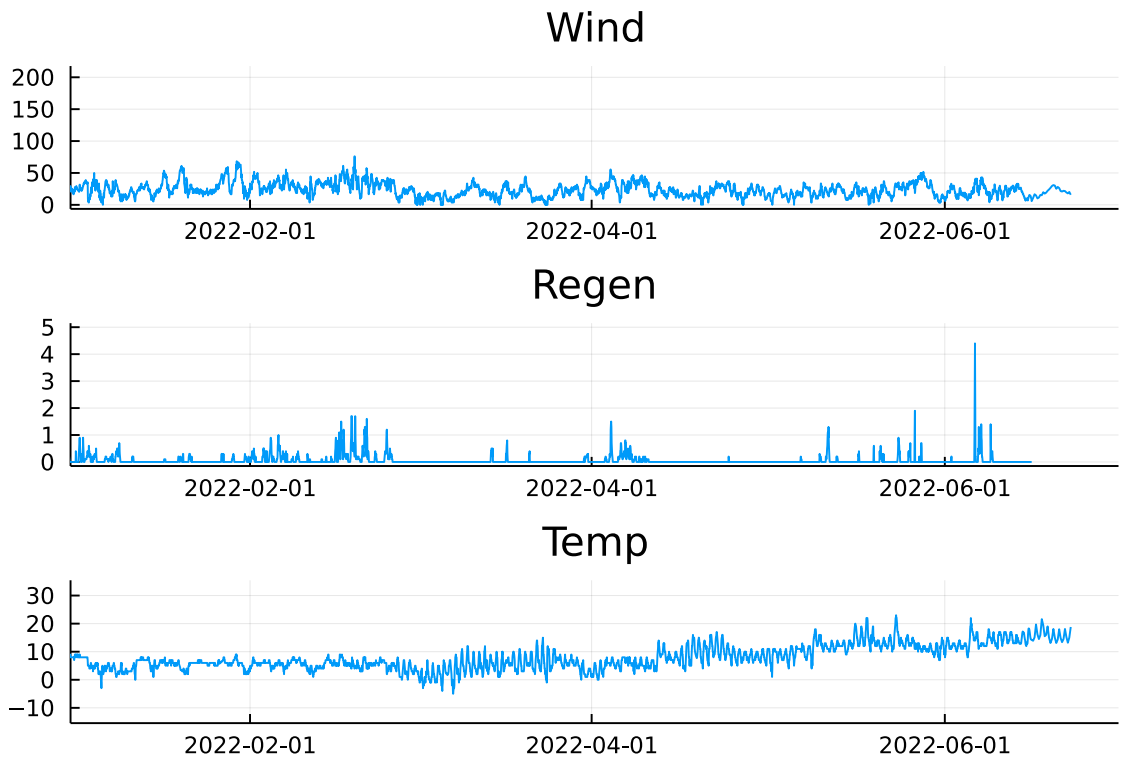
[45]:
```
@df df plot(:datetime, :Column3)
```

[45]:



[46]:
```
@df df plot(:datetime, [:Column9, :Column6, :Column3],
            xlims = (DateTime(2022,1,1), DateTime(2022,7,1)),
            layout=(3,1), title=["Wind" "Regen" "Temp"], legend=:none)
```

[46]:



[ ]: