2 Beschreibende Statistik

Markus Lippitz
2. Mai 2022

Ziele Sie können Methoden der beschreibenden Statistik *anwenden* und in Julia *implementieren*, um Aussagen über Datensätze zu treffen.

- Mittelwert und Standardabweichung
- Momente
- Korrelation und Kovarianz

Literatur Stahel Kap. 2 & 3, Bevington Kap. 1

Überblick

Durch eine *Messung* erhalten wir uns einen Messwert einer physikalischen Größe. Wir können die Messung bei ansonsten unveränderten Bedingungen wiederholen und erhalten viele Messwerte, die nur im idealisierten Fall identisch sind. Diese Menge von Messwerten wird in der Statistik *Stichprobe* genannt. In diesem Kapitel versuchen wir zunächst, Aussagen über diese Stichprobe zu machen. Eigentlich wollen wir aber natürlich etwas über die Wirklichkeit sagen können. Der Schluss von den Messwerten auf die Natur wird in der Statistik *Schätzung* genannt. Das ist dann Inhalt eines späteren Kapitels.

Temperaturverlauf

Als konkretes Beispiel betrachten wir den Temperaturverlauf am 1. August 2021 in einem unserer (klimatisierten) Labore. In einer idealen Welt wäre die Temperatur genau $21\,°\mathrm{C}$, aber natürlich ändert sie sich im Laufe des Tages. Diese Zeitabhängigkeit ignorieren wir zunächst und betrachten dies als wiederholte Messung bei unveränderten Bedingungen.

Zunächst laden wir den Datensatz. Hilfreich sind dabei die Pakete DataFrames und CSV.

```
1 using DataFrames, CSV
```

Dann laden wir die Datei in die Variable 'datensatz'. Die Spalten sind durch ein Tab ('\t') getrennt und die ersten 4 Zeilen beinhalten einen Datumsstempel, der uns hier nicht interessiert.

```
datensatz =
```

	Time	Tist	Tsoll	Kuelhlluft	Kuehlwasser	Zuluft	PIDstellwert	PI
1	00:00:04	20.9968	21.0	19.469	7.7987	21.5885	0.0	-1
2	00:00:13	20.9947	21.0	19.5322	7.8089	21.5844	-11.2608	-7
3	00:00:23	20.9988	21.0	19.6036	7.8191	21.5844	-13.1579	-1
4	00:00:33	20.9988	21.0	19.6668	7.817	21.5844	-13.1692	-1
5	00:00:43	21.0029	21.0	19.6934	7.8109	21.5844	-14.1261	-1
6	00:00:53	21.009	21.0	19.7076	7.8007	21.5844	-11.8549	-1
7	00:01:03	21.0111	21.0	19.7158	7.7987	21.5844	-13.2978	-1
8	00:01:13	21.0172	21.0	19.7036	7.7803	21.5844	-14.3151	-1
9	00:01:23	21.0192	21.0	19.6668	7.7477	21.5722	-13.4572	-1
10	00:01:33	21.0192	21.0	19.5791	7.7191	21.5783	-12.5823	-1
⋮ more								
8640	23:59:57	21.0437	21.0	17.1194	7.8986	21.4029	-12.632	-8

```
1 datensatz =
  CSV.read(download("https://raw.githubusercontent.com/MarkusLippitz/teca/main/res/
  02-beschreibende-statistik/Temperatur_1_0_05_1_Tag_210801.dat"), DataFrame;
  delim='\t', header=5)
```

Unsere Stichprobe sei die Ist-Temperatur

```
stichprobe = 8640-element SentinelArrays.ChainedVector{Float64, Vector{Float64}}:
             20.9968
             20.9947
             20.9988
             20.9988
             21.0029
             21.009
             21.0111
              ⋮
             21.0661
             21.0641
             21.062
             21.06
             21.0437
             21.0437
```

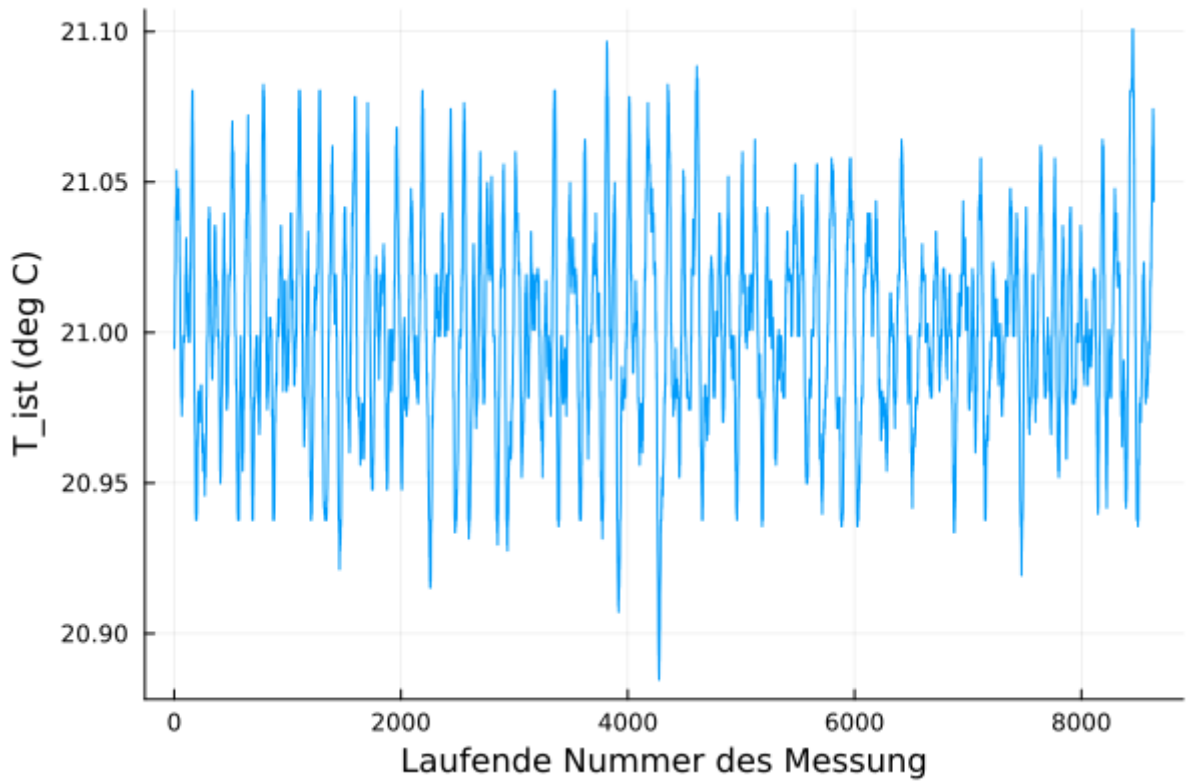
```
1 stichprobe = datensatz.Tist
```

Grafische Darstellung

Typischerweise ist es nicht sehr sinnvoll, alle Werte der Stichprobe in der Reihenfolge ihrer Messung darzustellen. Hier tun wir es trotzdem.

Wir benutzen hier die Plots-Bibliothek und das interaktive plotly-backend

```
1 using PlotlyBase,Plots
```

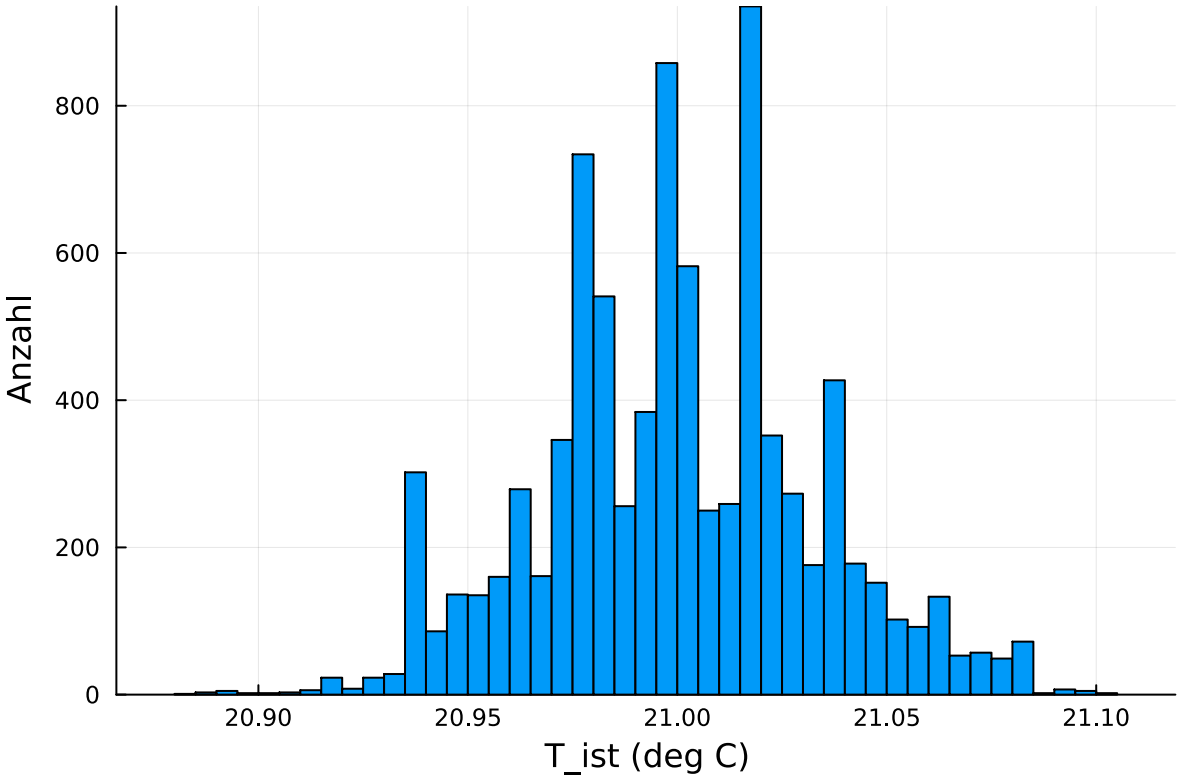



```
1 plot(stichprobe, xlabel="Laufende Nummer des Messung", ylabel="T_ist (deg C)",
  legend=false)
```

Auf den ersten Blick funktioniert die Klimaanlage ganz gut. Es ist keine systematische Variation zu erkennen.

Histogramm

Relevanter ist die Darstellung der Stichprobe als Histogramm. Man zählt, wie oft ein Wert innerhalb eines Intervalls, einer *Klasse*, vorkommt und zeichnet Balken entsprechender Höhe und Breite.




```
Plots.histogram(stichprobe, xlabel="T_ist (deg C)", ylabel="Anzahl",
legend=false)
```

Da scheint eine gewisse Präferenz für Temperaturen im Abstand von ca. 0.02 Grad zu sein, wo immer das herkommt.

Wichtig ist bei Histogrammen, dass das Integral über die x-Achse die Gesamtzahl der Messwerte bzw. eine Wahrscheinlichkeit von 1 ergibt. Das wird insbesondere dann relevant, wenn man unterschiedlich breite Klassen kombinieren möchte, weil beispielsweise am Rand nur wenige Ereignisse sind. Effektiv löst man dabei die Grenze zwischen den Klassen auf. Der neue Balken hat den Mittelwert der alten als Höhe, nicht seine Summe!

Das ist allerdings für die Histogramm-Funktion aus 'Plots' zu kompliziert. Wir benutzten 'StatsBase'.

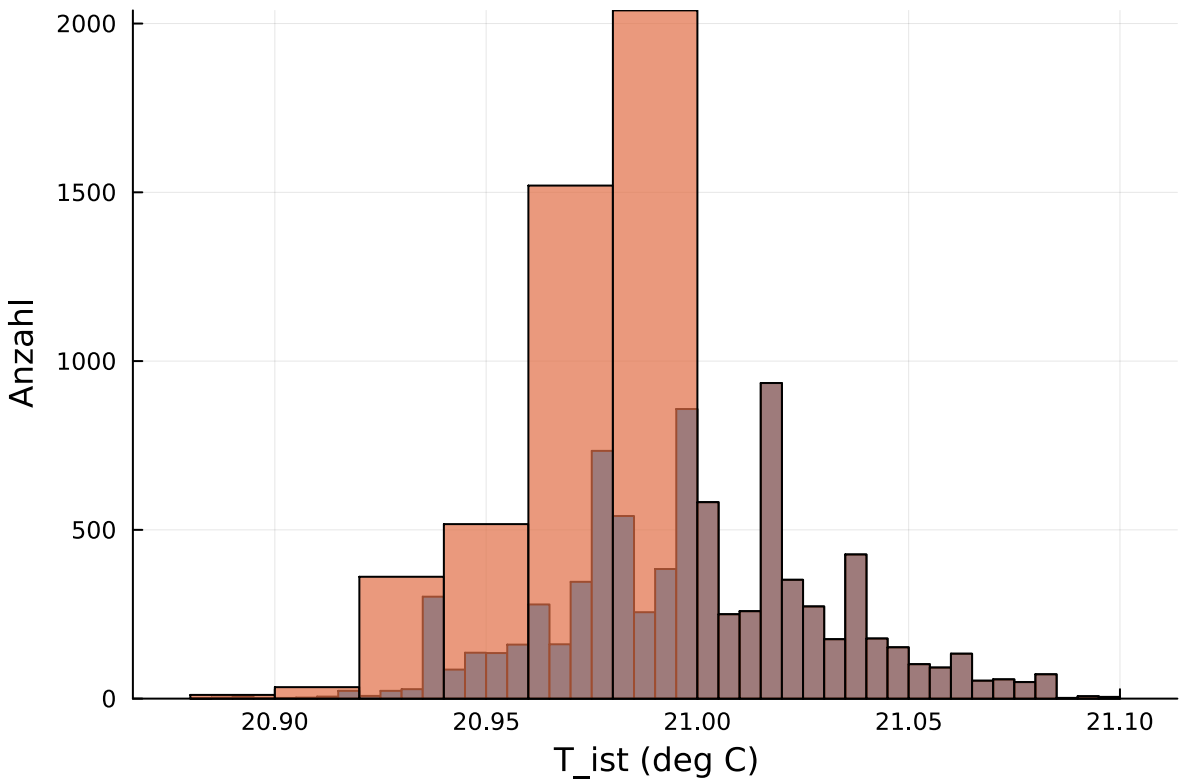
```
using StatsBase
# Man muss im Folgenden das "StatsBase." nicht mit schreiben. Ich tue es hier,
um deutlich zu machen, wo die Funktion her kommt
```

0.02

```
@bind left_bin_width  Slider( 0.005: 0.005 : 0.1; default= 0.02, show_value =
true)
```

○ pdf
○ density
○ probability
◉ none

```
@bind histo_mode Radio(["pdf", "density", "probability", "none"], default="none")
```



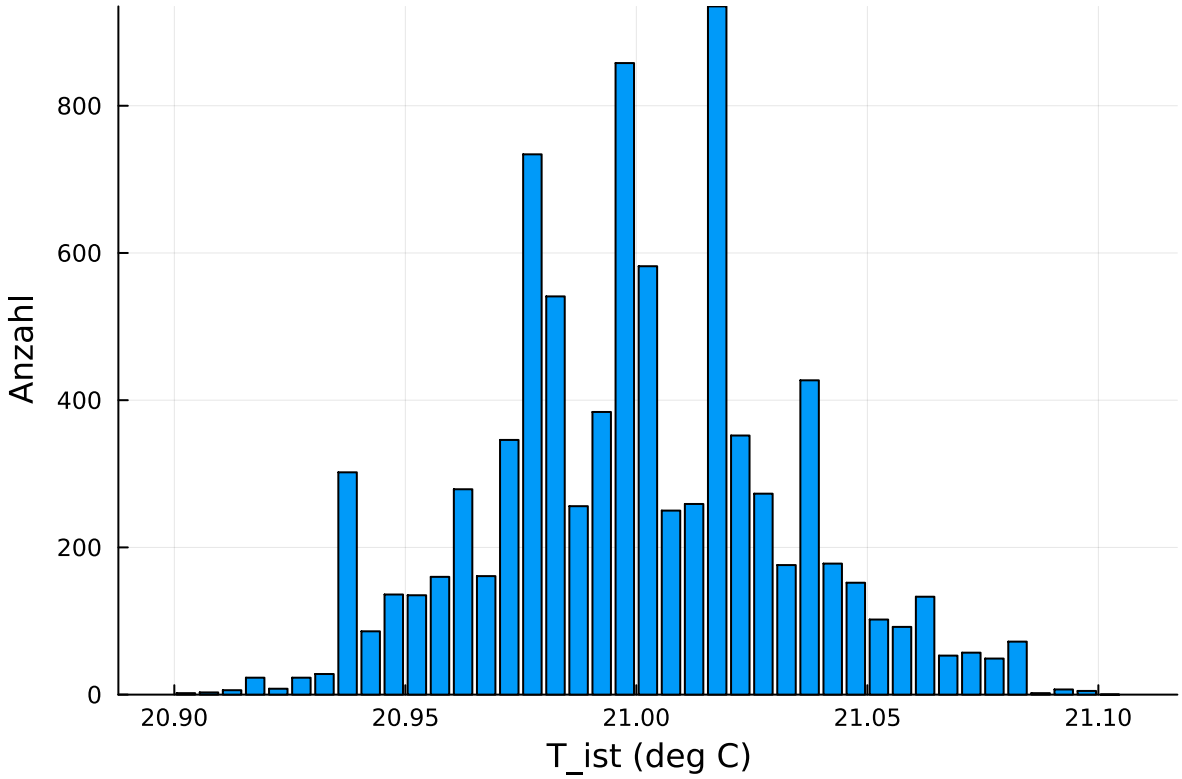

```
let
    # Histogram with variable bin size on left half
    bins_l = range(20.88, 21; step=left_bin_width)
    bins_r = range(21, 21.1; step=0.005)
    h1 = StatsBase.fit(Histogram, stichprobe, [bins_l; bins_r])
    h1 = LinearAlgebra.normalize(h1,mode=Symbol(histo_mode))

    # for comparison: histogram with fixed, small bin size
    h2 = StatsBase.fit(Histogram, stichprobe, range(20.88, 21.1; step=0.005))
    h2 = LinearAlgebra.normalize(h2,mode=Symbol(histo_mode))

    # adjust label according to normalization
    yaxis =  Dict([ (:none, "Anzahl"),
                    (:pdf, "WK-Dichte (1/deg C)"),
                    (:probability, "Wahrscheinlichkeit"),
                    (:density, "Anzahl (1/deg C)")] )

    # overlay both histograms
    plot(h2, legend=false )
    plot!(h1, fillopacity=0.7, xlabel="T_ist (deg C)",
            ylabel=yaxis[Symbol(histo_mode)])
end
```

Histogramme selbst gemacht




```
let
    binsize = 0.005
    edges = range(20.9, 21.1; step=binsize)
    counts = zeros(length(edges))

    for id = 1:length(edges)-1
        edge_low = edges[id]
        edge_high = edges[id+1]
        counts[id] = count(x -> edge_low <= x <edge_high , stichprobe)
    end

    Plots.bar(edges .+ binsize/2, counts,
        xlabel="T_ist (deg C)", ylabel="Anzahl", legend=false)
end
```

Rang und kumulative Verteilungsfunktion

Unsere Stichprobe besteht aus den Werten x_i mit dem laufenden Index $i = 1 \ldots n$. Wir können die x_i der Größe nach sortieren und bezeichnen dann mit $x_{[i]}$ den i-ten Eintrag nach Sortierung. Also ist $x_{[1]}$ die kleinste Zahl und $x_{[n]}$ die größte.

In Julia ist die zuerst gemessene Zahl

```
20.9968
```
```
stichprobe[1]
```

und die letzte Zahl ist

```
21.0437
```
```
stichprobe[end]
```

Die drittletzte ist

```
21.06
```
```
stichprobe[end-2]
```

Wir können die Stichprobe sortieren und direkt auf die sortierte Liste zugreifen, zB die zweitgrößte Zahl ist

```
21.1008
```
```
sort(stichprobe)[end-1]
```

Der **Rang** (engl. rank) einer Zahl gibt an, wie wievielt-kleinste Zahl sie ist. Das schreibt man als $\mathrm{Rang}(x_i)$. Es gibt verschiedene Methoden damit umzugehen, dass ein Wert mehrfach vorkommt.

```
x = [3.4, 6.1, 1.2, 7.8, 3.4];
```

Man ignoriert einfach, dass manche Zahlen doppelt vorkommen

```
▸ [2, 4, 1, 5, 3]
```
```
StatsBase.ordinalrank(x)
```

oder wie bei Olympia

```
▸ [2, 4, 1, 5, 2]
```
```
StatsBase.competerank(x)
```

Oft erhalten alle identischen Zahlen den gleichen Rang, und zwar den Mittelwert der von ihnen belegten Ränge.

```
▸ [2.5, 4.0, 1.0, 5.0, 2.5]
```

```
1 StatsBase.tiedrank(x)
```

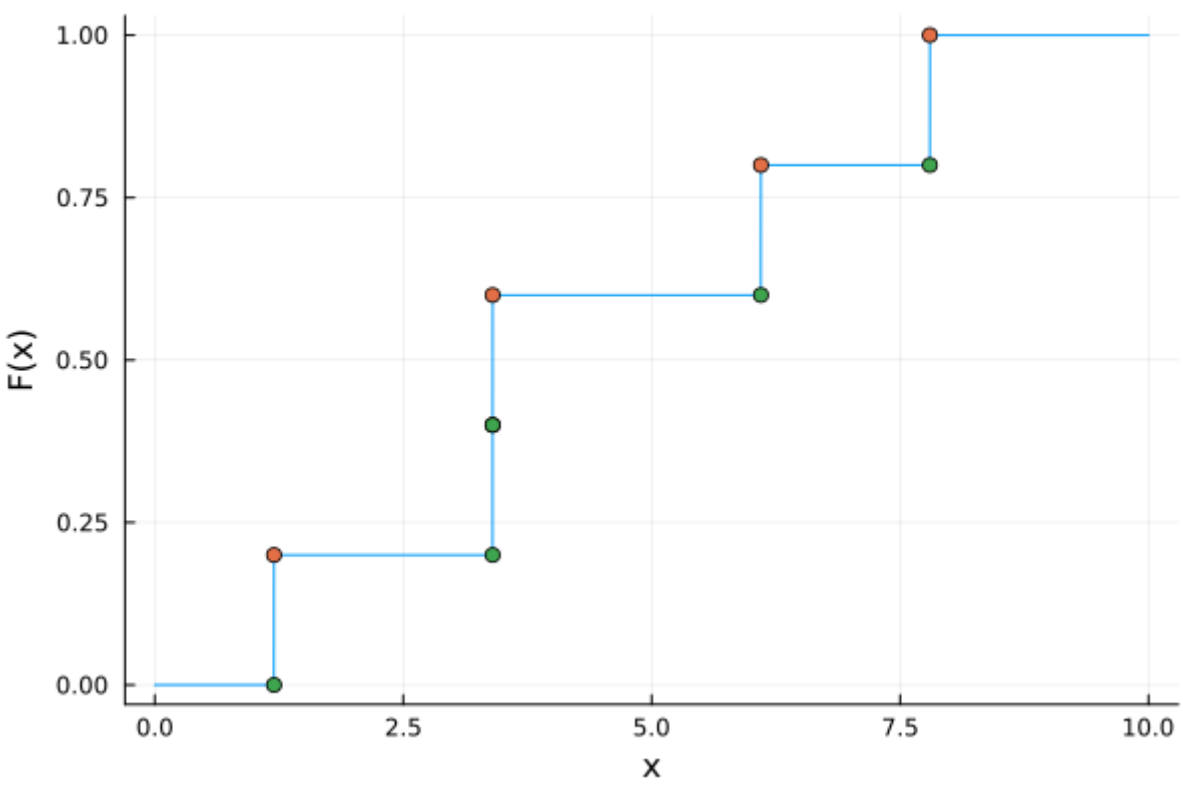
Die **empirische kumulative Verteilungsfunktion** (engl. empirical cumulative distribution function, ecdf) ordnet jedem Wert x den Anteil der Wert x_i zu, die kleiner oder gleich x sind.

$$F(x) = \frac{1}{n} \text{Anzahl } (i | x_i \leq x)$$

Der Graph dieser Funktion $F(x)$ zeigt Stufen mit Ecken an den Koordinaten $(x_{[k]}, (k-1)/n)$ und $(x_{[k]}, k/n)$.

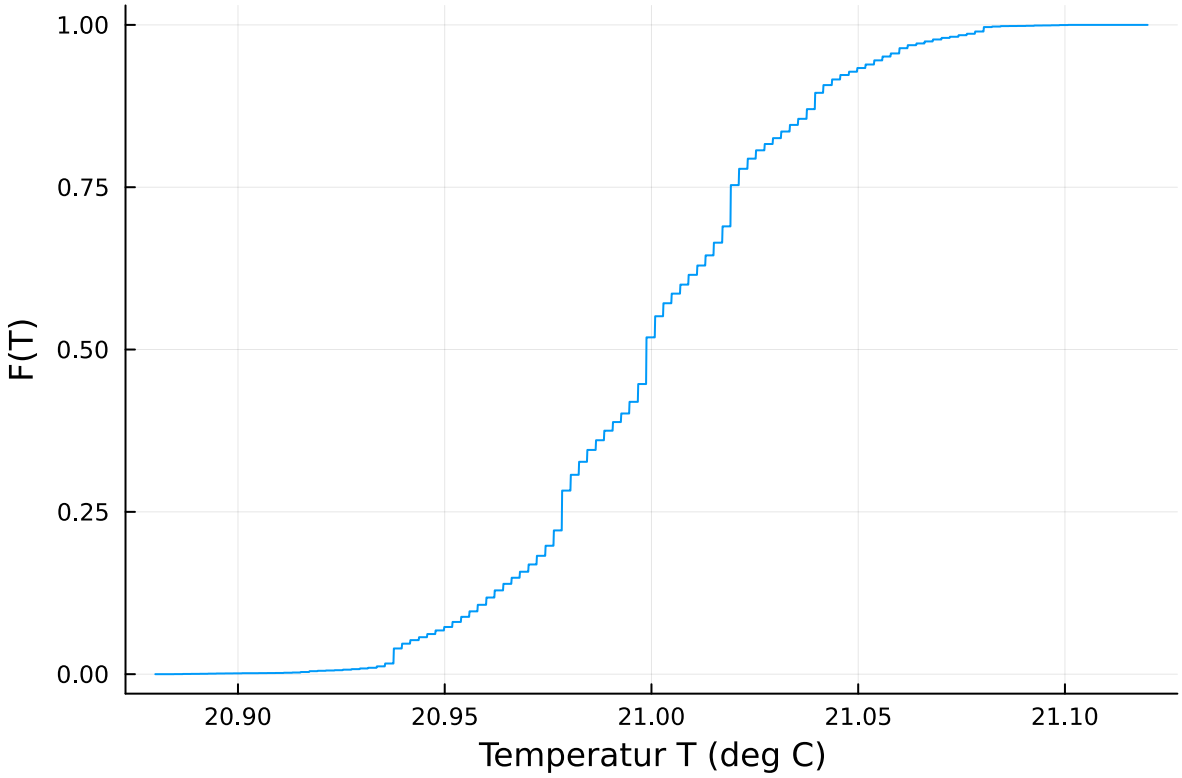
In Julia erzeugt 'ecdf' eine Funktion, wie wir aufrufen können.

```
1 ecdf_x = StatsBase.ecdf(x);
```




```
1 let
2     u = range(0, 10; step=0.001)
3     plot(u, ecdf_x(u), xlabel="x", ylabel="F(x)", legend=false)
4     # zum Vergleich 'von Hand'
5     scatter!(sort(x),  (1:length(x)) / length(x))
6     scatter!(sort(x), ((1:length(x)) .- 1) / length(x))
7 end
```

Für die Labortemperatur ist dies




```
let
    u = range(20.88, 21.12; step=0.0001)
    plot(u, StatsBase.ecdf(stichprobe)(u), xlabel="Temperatur T (deg C)",
    ylabel="F(T)", legend=false)
end
```

Kennzahlen

Mit einigen Kennzahlen können wir die Stichprobe beschreiben. Die bekannteste ist sicherlich der **Mittelwert** (engl. mean), Durchschnitt oder Schwerpunkt

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

In Julia ist das

```
21.00023890046296
```

```
StatsBase.mean(stichprobe)
```

oder

```
21.00023890046296
```
```
1 sum(stichprobe) / length(stichprobe)
```

Manchmal ist der **Median** oder Zentralwert aussagekräftiger. Das ist der mittlere Wert einer geordneten Stichprobe, also

$$\mathrm{med}_i \langle x_i \rangle = x_{[(n+1)/2]} \quad \text{bzw.} \quad = \frac{1}{2}\left(x_{[n/2]} + x_{[(n/2)+1]}\right)$$

Der Median ist weniger anfällig gegen Ausreißer. Selbst wenn man den größten Wert der Stichprobe mit 100 multipliziert, ändert sich der Median nicht, der Mittelwert aber schon.

In Julia

```
20.9988
```
```
1 StatsBase.median(stichprobe)
```

oder 'von Hand'. Die Division liefert einen Fließkomma-Zahl, die erst in ein Integrer gewandelt werden muss, bevor sie als Index verwendet werden kann.

```
20.9988
```
```
1 sort(stichprobe)[Int32(length(stichprobe)/2)]
```

Analog zum Median kann man die sortierte Stichprobe an anderen Stellen als der Mitte auslesen. Das untere bzw. obere **Quartil** liest den Wert bei $1/4$ bzw. $3/4$. **Perzentile** lesen bei p Prozent aus. Allgemein nennt man diese Maße Quantile.

Das 2%-Perzentil unseres Datensatzes ist

```
20.9377
```
```
1 StatsBase.percentile(stichprobe, 2)
```

Die Quartilen sind

```
▸ [20.8846, 20.9784, 20.9988, 21.0192, 21.1008]
```
```
1 StatsBase.nquantile(stichprobe, 4)
```

wobei die beiden 'äußeren' Werte das Minimum und das Maximum des Datensatzes angeben

```
(20.8846, 21.1008)
```

```
minimum(stichprobe), maximum(stichprobe)
```

Die **Varianz** ist ein Maß für die Streuung der Datenpunkte

$$\mathrm{var} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right]$$

Die **Standardabweichung** ist die Wurzel der Varianz

$$\sigma = \sqrt{\mathrm{var}} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Man beachte den Term $n-1$ in dieser Definition. Es gibt eine sehr ähnliche Definition mit n an Stelle von $n-1$, nur dass dann auch $\bar{x}$ durch μ ersetzt ist. Doch dazu mehr in dem Kapitel über Schätzer.

In Julia 'von Hand' ist die Standardabweichung unsere Stichprobe

```
0.032742359226086
```

```
sqrt(sum( (stichprobe .- mean(stichprobe)).^2) / (length(stichprobe)-1))
```

Es geht auch mit der 'eingebauten' Funktion 'std'. Durch 'corrected' kann man zwischen der Variante $n-1$ (true) und n (false) umschalten.

```
(0.816497, 0.816497, 1.0)
```

```
std([1,2,3], corrected=false), sqrt(2/3) , std([1,2,3], corrected=true)
```

oder aus 'StatsBase' Mittelwert und Standardabweichung in einem Aufwasch

```
(2.0, 1.0)
```

```
StatsBase.mean_and_std( [1,2,3], corrected=true)
```

Man kann die Definition der Varianz ausbauen zu allgemeinen **statistischen Momenten**. Das dritte Moment ist die **Schiefe** (engl. skewness)

$$\frac{1}{n}\sum\left(\frac{x_i-\bar{x}}{\sigma}\right)^3$$

und das vierte die **Wölbung** (engl. kurtosis)

$$\frac{1}{n}\sum\left(\frac{x_i-\bar{x}}{\sigma}\right)^4$$

Durch die Division durch die Standardabweichung sind diese Momente einheitenfrei.

In Julia

```
▸ (0.0413242, -0.0776418)
1 StatsBase.skewness(stichprobe), StatsBase.kurtosis(stichprobe)
```

Kennzahlen klassierter Daten

Schon beim Histogramm hatten wir Daten in Klassen eingeteilt. Alle Messwerte, die in ein gewisses Temperatur-Intervall gefallen sind, zählten zum gleichen Balken des Histogramms. Manchmal möchte man solche klassierte Daten weiter verarbeiten, manchmal liegen Daten auch direkt nur klassiert vor. Ein Beispiel ist die Notenverteilung einer Klausur. Die Noten können dabei nur wenige verschiedene Werte annehmen.

Auch aus klassierten Daten kann man die oben besprochenen Kennzahlen berechnen. Sei c_l die Grenze der Klassen, und somit $z_l = (c_{l-1} + c_l)/2$ die Mitten der Klassen. Beim Übergang zu klassierten Daten ersetzt man in den Gleichen oben jedes x_i durch 'seine' Klassenmitte $z_{l(i)}$, summiert also weiterhin über i. Dabei sind dann natürlich viele $z_{l(i)}$ identisch, so dass man diese Teilsummen ausklammern kann und durch die Anzahl h_l der Werte in der Klasse l ersetzen kann.

Ausgeschrieben ergibt sich für den **Mittelwert**

$$\bar{x} \approx \frac{1}{n} \sum_l h_l z_l$$

und die **Varianz**

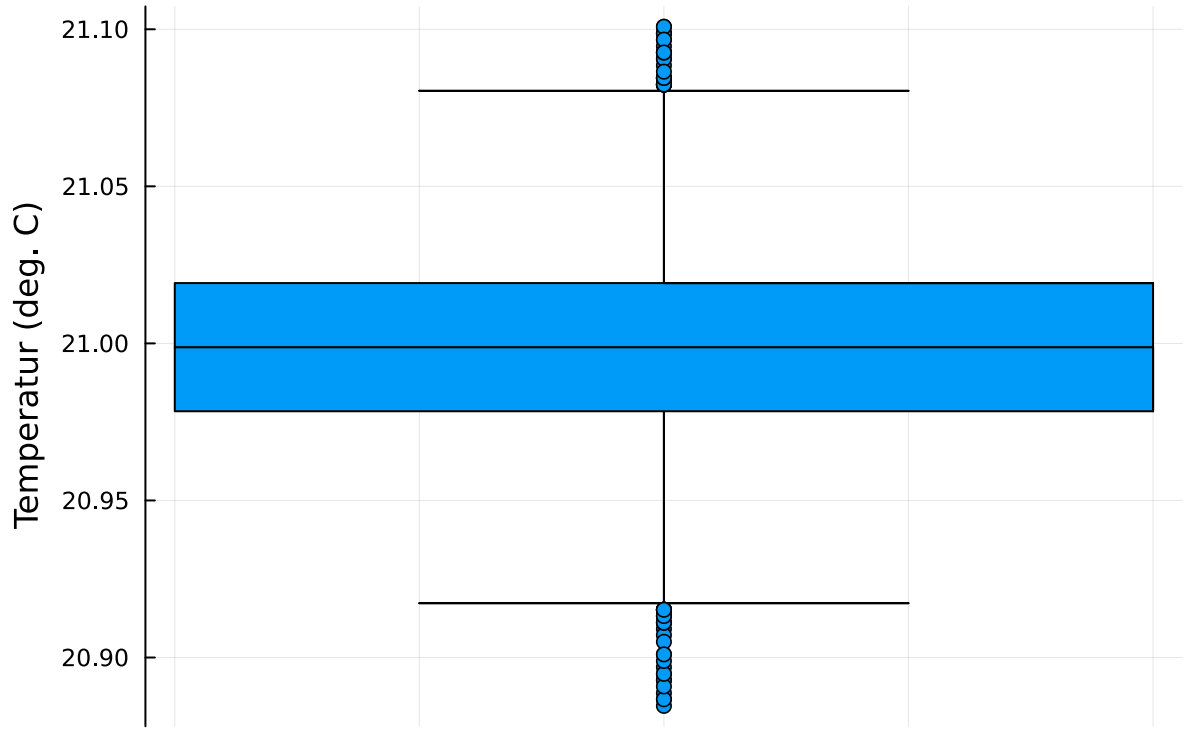
$$\mathbf{var} \approx \frac{1}{n-1} \sum_l h_l (z_l - \bar{x})^2 = \frac{1}{n-1} \left(\sum_l h_l z_l^2 - n \bar{x}^2 \right)$$

weil $\sum_l h_l = n$.

Boxplot

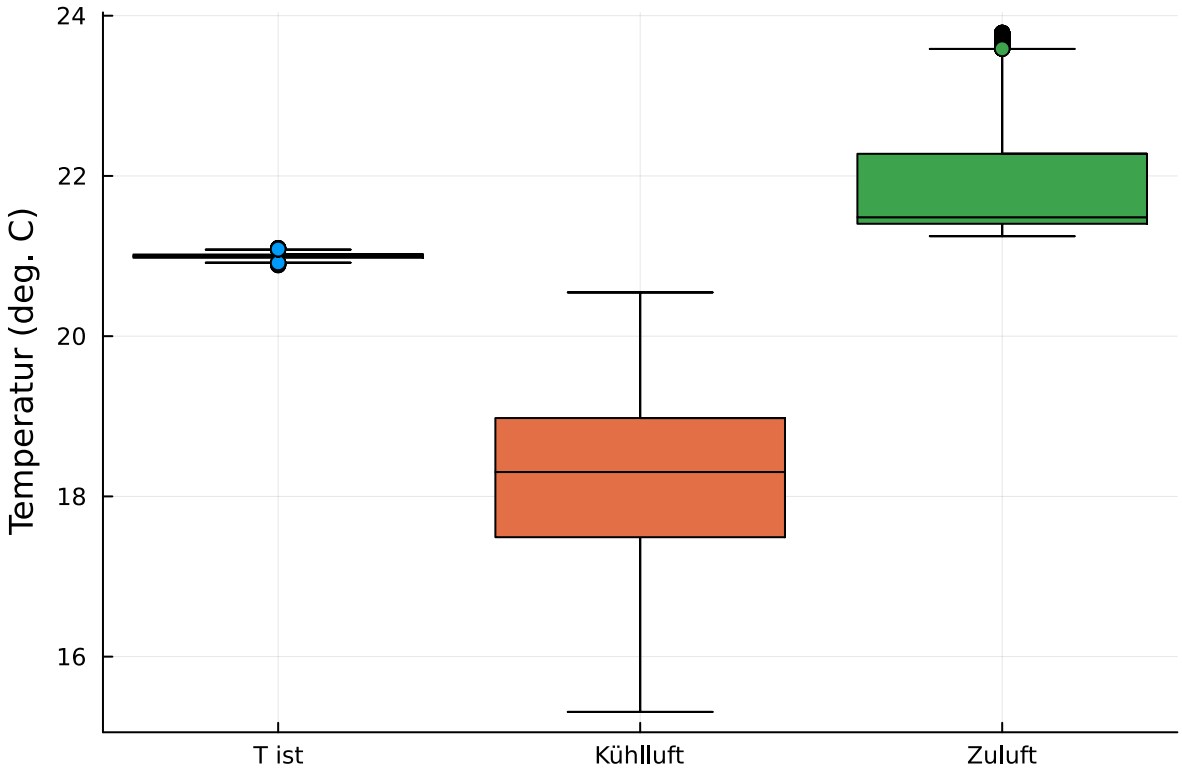
Wenn man verschiedene Stichproben auf einen Blick vergleichen will, dann ist der 'boxplot' hilfreich. Er zeichnet eine Kiste vom oberen zum unteren Quartil mit einem Strich beim Median. Die 'Fehlerbalken' haben typischerweise die Länge von 1.5 mal der Kistenhöhe, werden aber zum nächstgelegenen Wert nach 'innen' gerundet. Alle weiter außen liegenden Werte werden eingezeichnet.

```
1 using StatsPlots
```




```
1  StatsPlots.boxplot(datensatz.Tist, ylabel = "Temperatur (deg. C)",
legend=false, xaxis=false)
```

So kann man gut mehrere Verteilungen vergleichen.




```
StatsPlots.boxplot([ datensatz.Tist, datensatz.Kuelhlluft,  datensatz.Zuluft],
ylabel = "Temperatur (deg. C)",  xticks = (1:3, ["T ist", "Kühlluft",
"Zuluft"]), legend=false)
```

Mehrdimensionale Daten

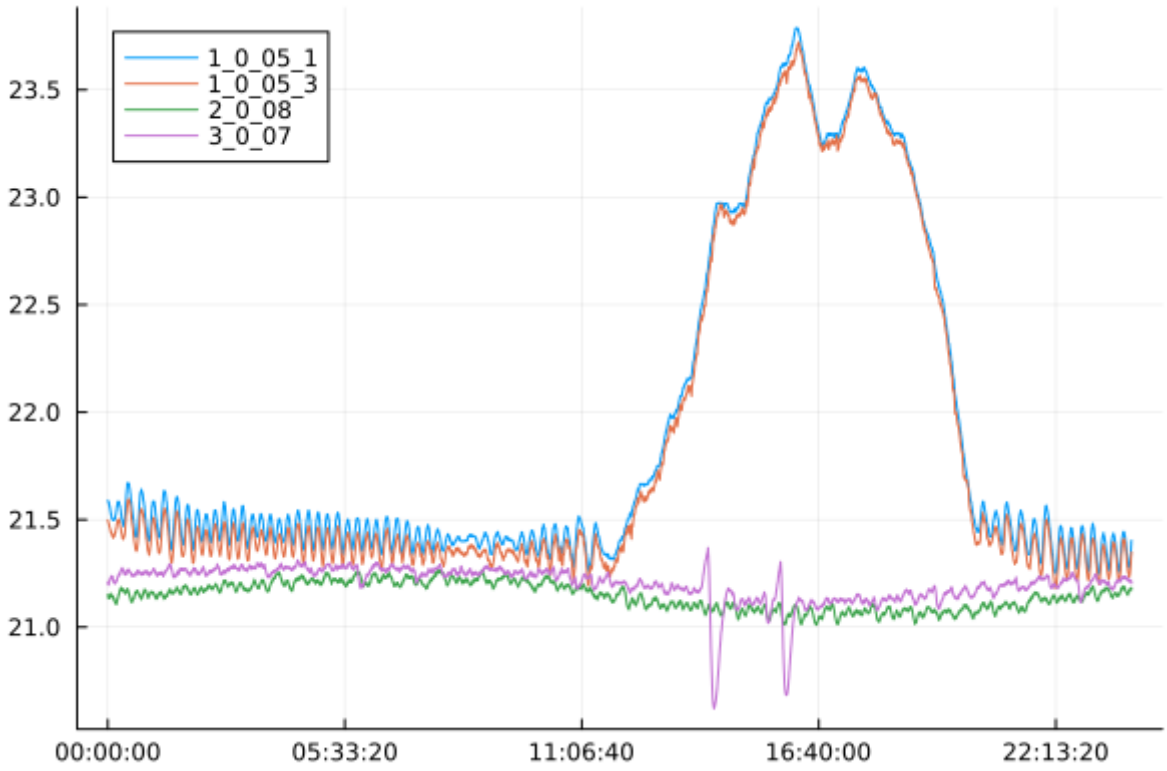
Oft werden mit einer Beobachtung, einer Messung mehrere Größen gemessen. Damit erhält man mehrdimensionale oder multivariate Daten.

Im Prinzip war schon unser Temperatur-Beispiel von oben mehrdimensional, da wir zu jedem Zeitpunkt nicht nur die Raumtemperatur, sondern auch Temperaturen an anderen Stellen vorliegen hatten. Die hatten wir bisher ignoriert. Nun nehmen wir sie hinzu, und auch noch die gleichzeitige Messung der gleichen Größen in anderen Räumen.

Wir laden vier Dateien in eine gemeinsame Variable

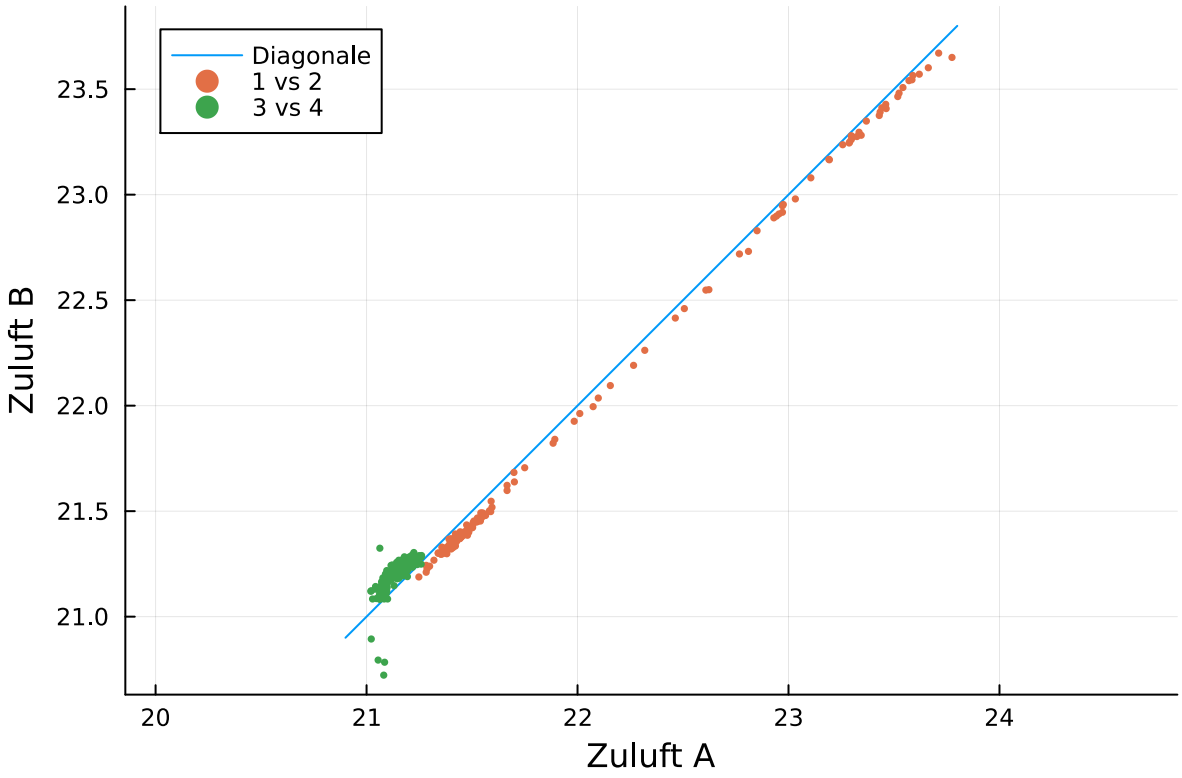
```
begin
    räume = ["1_0_05_1","1_0_05_3","2_0_08","3_0_07"]
    files = ["https://raw.githubusercontent.com/MarkusLippitz/teca/main/res/02-beschreibende-statistik/Temperatur_$(r)_Tag_210801.dat" for r in räume]
    datensätze = CSV.read.(download.(files), DataFrame; delim='\t', header=5)
    # man beachte den Punkt in "read.(download" und den in "download.(files" !
    # -> Wende alles auf alle Einträge in "räume" an
end;
```

Zunächst stellen wir wieder die Daten als Funktion der Zeit dar. Die erste Ziffer der Raumnummer bezeichnet das Gebäudeteil im BGI. In Bauteil 1 scheint die Zuluft im Sommer merklich warm zu werden, in den Bauteilen 2 und 3 nicht.



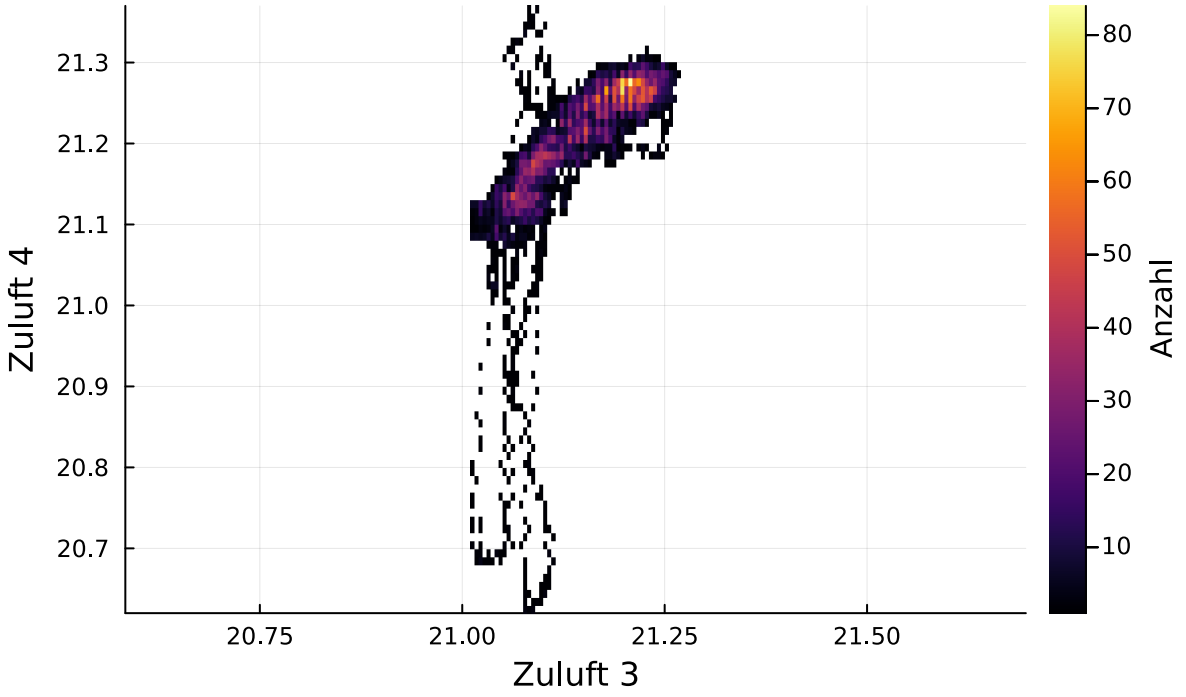

```
begin
    plot()  # generate empty plot
    for i in 1:length(räume) # plot 4 traces on top of each other
       plot!(datensätze[i].Time, datensätze[i].Zuluft, label=räume[i])
    end
    plot!() # show last plot
end
```

In einem **Streudiagramm** stellt man eine Größe gegenüber der anderen dar, um so Zusammenhänge zu erkennen. Wenn die Punkte sehr dicht liegen reduziert das die Aussagekraft. Hier ist einfach nur jeder 50. Datenpunkt gezeichnet.




```
let
    dec = 50
    plot([20.9, 23.8], [20.9, 23.8], label="Diagonale")
    Plots.scatter!(datensätze[1].Zuluft[1:dec:end],
    datensätze[2].Zuluft[1:dec:end],
        label="1 vs 2",  markersize=2, markerstrokewidth=0)
    Plots.scatter!(datensätze[3].Zuluft[1:dec:end],
    datensätze[4].Zuluft[1:dec:end],
        label="3 vs 4",markersize=2, markerstrokewidth=0,
        xlabel="Zuluft A", ylabel="Zuluft B",aspect_ratio=:equal,)
end
```

Alternativ kann man ein zweidimensionales Histogramm zeichnen, und die Anzahl pro bin farbkodieren.




```
Plots.histogram2d(datensätze[3].Zuluft, datensätze[4].Zuluft,
bins=100,aspect_ratio=:equal, xlabel="Zuluft 3", ylabel="Zuluft 4",
colorbar_title="Anzahl")
```

Produktmomenten-Korrelation

Eine Kennzahl für den Zusammenhang zwischen zwei Messgrößen ist die *Korrelation*, von der es verschiedene Varianten gibt.

Zunächst normieren wir die einzelnen Messwerte so, dass deren Mittelwert 0 und deren Standardabweichung 1 beträgt. Wir definieren also

$$\tilde{x}_i = \frac{x_i - \bar{x}}{\sigma_x}$$

und analog für die zweite Größe y_i. Wenn nun $\tilde{x}_i$ und $\tilde{y}_i$ beide gleichzeitig positiv, oder beide gleichzeitig negativ sind, dann entspricht dies einem gewissen positiven Zusammenhang zwischen x_i und y_i. Ein einfaches Maß für diesen Zusammenhang ist das Produkt $\tilde{x}_i\tilde{y}_i$, also

$$r_{xy} = \frac{1}{n-1}\sum_i \tilde{x}_i\tilde{y}_i$$

Dies ist die (einfache) **Korrelation**, oder auch *Produktmomenten-Korrelation nach Pearson*. Wir können das noch etwas umformen

$$r_{xy} = \frac{\mathrm{var}_{xy}}{\sigma_x\sigma_y} \quad \text{mit} \quad \mathrm{var}_{xy} = \frac{1}{n-1}\sum_i (x_i - \bar{x})(y_i - \bar{y})$$

Die Kovarianz var_{xy} ist sehr analog zur Varianz, nur das das Quadrat einer Größe durch das Produkt von zwei Größen ersetzt wird.

Einfache Spezialfälle sind $x_i = y_i$, was $r_{xy} = 1$ ergibt. Falls ein linearer Zusammenhang besteht in der Art $y_i = a + bx_i$, dann ist ebenfalls $r_{xy} = 1$, falls $b > 0$, und $r_{xy} = -1$, falls $b < 0$. Die Produktmomenten-Korrelation misst also die Stärke des linearen Zusammenhangs.

In Julia ist geht das via StatsBase

```
0.7044937630442083
```
```
1 StatsBase.cor(datensätze[3].Zuluft, datensätze[4].Zuluft)
```

```
0.9997865915236434
```
```
1 StatsBase.cor(datensätze[1].Zuluft, datensätze[2].Zuluft)
```

oder von Hand

```
0.7044937630442066
```

```
let
    x = datensätze[3].Zuluft
    y = datensätze[4].Zuluft
    (m_x, σ_x) =  StatsBase.mean_and_std( x, corrected=true)
    (m_y, σ_y) =  StatsBase.mean_and_std( y, corrected=true)
    xt = (x .- m_x) ./ σ_x
    yt = (y .- m_y) ./ σ_y
    cor = sum( xt .* yt) / (length(xt)-1)
end
```

Achtung Diese Produktmomenten-Korrelation misst nur die Stärke eines linearen Zusammenhangs. Es können noch andere Zusammenhänge höherer Ordnung vorhanden sein, die bei gleichem r_{xy} sehr verschiedene Streudiagramme bewirken! Die Korrelation ist nicht robust gegen Ausreißer.

Rangkorrelation

Ähnlich zum Unterschied zwischen Mittelwert und Median, kann man auch eine Korrelation nicht zwischen den Werten an sich, sondern zwischen den Rängen berechnen. Hier spielen Ausreißer dann quais keine Rolle mehr.

Analog definieren wir die Spearmansche Rangkorrelation

$$r_{xy}^{(\mathrm{Sp})} = \frac{\mathbf{var}_{\mathbf{Rang(x)\ Rang(y)}}}{\sigma_{\mathbf{Rang(x)}}\sigma_{\mathbf{Rang(y)}}}$$

Der Mittelwert aller Ränge ist

$$\overline{\text{Rang}}(x) = \frac{1 + 2 + \cdots + n}{n} = \frac{n+1}{2}$$

Die Standardabweichung ist **XXX TODO** und die Kovarianz

$$\text{var}_{\text{Rang(x) Rang(y)}} = \frac{1}{n-1} \sum_i \left(\text{Rang}(x_i) - \frac{n-1}{2} \right) \left(\text{Rang}(y_i) - \frac{n-1}{2} \right)$$

Zusammen ergibt das

$$r_{xy}^{(\text{Sp})} = 1 - \frac{6}{n(n^2-1)} \sum_i (\text{Rang}(x_i) - \text{Rang}(y_i))^2$$

In Julia ist das

```
0.8547704554642104
```

```
1 StatsBase.corspearman(datensätze[3].Zuluft, datensätze[4].Zuluft)
```

```
0.993682586992326
```

```
1 StatsBase.corspearman(datensätze[1].Zuluft, datensätze[2].Zuluft)
```

Achtung

- Eine Korrelation ist keine Kausalität. Nur weil etwas korreliert ist, heißt das noch nicht, dass es einen interessanten Grund dafür gibt. Die Größe der Storchenpopulation in Deutschland ist sicherlich mit der Geburtenrate der letzten 50 Jahre korreliert.
- Korrelationen können auch entstehen, in dem man beide Größen x und y mit einer gemeinsamen Größe z normiert. Dann ist ggf. x/z mit y/z korreliert, obwohl x nicht mit y korreliert ist.

Lineare Regression

Auch eine lineare Regression sucht nach einem linearen Zusammenhang zwischen x_i und y_i. Der Unterschied liegt in der Interpretation der x_i. Bei der Korrelation waren die x_i und die y_i völlig gleichberechtigt. Es ist ja sogar $r_{xy} = r_{yx}$. Bei einer linearen regression nimmt man die x_i als 'unabhängige Variable' oder 'Ausgangsgröße' an. Diese Werte im Experiment vorgegeben, sind fest und fehlerfrei bekannt. Die y_i sind die 'Zielgröße', der 'Messwert'. Die Aufgabe der linearen Regression ist es, aus bekanntem x ein noch zu messendes y vorherzusagen. B

Bei der linearen Regression suchen wir die Parameter (a, b) einer Funktion

$$f(x) = a + bx$$

so dass die Abweichung zwischen vorhergesagtem Wert $f(x_i)$ und gemessenen Wert y_i möglichst klein wird. Dazu betrachten wir die *Residuen*

$$\Delta_i(a, b) = y_i - (a + bx_i)$$

Ein oft verwendetes Maß ist die Summe der Quadrate der Abweichungen (*Methode der kleinsten Quadrate*)

$$Q(a, b) = \sum_i [\Delta_i(a, b)]^2$$

Wir wollen $Q(a, b)$ minimieren, suchen also Nullstellen der Ableitungen nach a und b. Man findet (siehe z.B. Stahel)

$$b = r_{xy}\frac{\sigma_y}{\sigma_x} \quad \text{und} \quad a = \bar{y} - b\bar{x}$$

Man erkennt den Zusammenhang zwischen Regression und Korrelation durch das Auftauchen von r_{xy}.

In Julia geht das beispielsweise über das Anpassen eines Polynoms 1. Grades

```
1 using Polynomials
```

0.42074011374460674 + 0.9827982878439211·x

```
1 Polynomials.fit(datensätze[3].Zuluft, datensätze[4].Zuluft, 1)
```

oder 'von Hand'

(0.42074, 0.982798)

```
let
    x = datensätze[3].Zuluft
    y = datensätze[4].Zuluft
    (m_x, σ_x) =  StatsBase.mean_and_std( x, corrected=true)
    (m_y, σ_y) =  StatsBase.mean_and_std( y, corrected=true)
    r_xy = StatsBase.cor(x,y)

    b = r_xy * σ_y  / σ_x
    a = m_y - b * m_x
    (a,b)
end
```

Oder ganz allgemein über einen 'least square fit' (kleinste Quadrate [der Residuen])

```
using LsqFit
```

[0.42074, 0.982798]

```
let
    x = datensätze[3].Zuluft
    y = datensätze[4].Zuluft
    model(x, p) = p[1]  .+ p[2] .* x  # Zielfunktion
    p0 = [0.5, 0.5]  # Start-Parameter
    fit = LsqFit.curve_fit(model, x, y, p0)
    fit.param
end
```

```
using PlutoUI,  LinearAlgebra
```

Inhalt


```
1 TableOfContents(title="Inhalt")
```