Statistik mit R – Wesentliches

Wieso soll ich mir das antun?

Statistik und Programmieren?! Tatsächlich werden die beiden eher polarisierenden Aktivitäten zusammen nicht schwieriger (sondern einfacher) und auch nicht nerdiger (doch, das eigentlich schon…).

Datenauswertung (Data Sciences) – ist eine wesentliche Kompetenz, werden in unserer Zeit doch enorm viele Daten produziert: Messwerte von smarten Sensoren (Internet of Things), Benutzerdaten von Social Media, Gesundheitsdaten von Lebewesen, und auch KI basiert auf riesigen Datenmengen.

Für einfache Datenauswertung wie Mittelwerte berechnen und Diagramme erstellen wird meist Excel verwendet. Das Programm ist intuitiv und für viele Aufgaben auch wirklich sehr gut geeignet. Doch schon bald wird es unübersichtlich, unklar und aufwendig – wer nur schon ein Diagramm individuell anpassen wollte, kann ein Lied davon singen.

R ist eine Programmiersprache, die für Statistikaufgaben entwickelt wurde. Man muss mit ihr programmieren, also Anweisungen schreiben und nicht zusammenklicken. Doch was auf den ersten Moment schwierig klingt, hat viele Vorteile:

- Die Programmanweisungen sind als Text vorhanden und sofort lesbar. Was wie gemacht wurde und ob es einen Fehler enthält, kann man rasch herausfinden.
- Ein Programm kann sehr einfach kopiert und angepasst werden. Die gesamte Auswertung mit einem anderen Datensatz machen? Mit wenigen Änderungen im Code ist das bereits erledigt.
- Ob es um einen Mittelwert oder einen Chi-Quadrat-Test geht – in Büchern, online-Foren oder Tutorials findet man die passenden Codezeilen, die sich rasch kopieren und anpassen lassen.
- Zu Beginn überraschend haben sich KI als sehr gute Programmierer herausgestellt – auch für R. Wie erstellt man ein Histogramm, und zwar mit roten Säulen und blauem Rahmen? Die KI weiss Bescheid und wird auch nicht müde, jeden Befehl genau zu erklären.

Was diese Anleitung bringt

Hoffentlich ziemlich viel… Im Vorwort lernten Sie, dass Excel nicht so geeignet sei. Doch um Daten zu erfassen, ist es schon praktisch und wird v.a. sehr häufig dafür genutzt. Es geht hier also darum, wie man die Daten, welche man bereits übersichtlich in Excel hat, mit R statistisch auswertet. Also das Beste aus zwei Welten herauszuholen.

Abkürzung für ein erstes Kennenlernen: Gehen Sie auf https://margadant.net/no_wp/statistik/r.html um direkt im Webbrowser die R-Programme anzupassen und auszuführen – ohne Installation und ohne RStudio zu lernen. Springen Sie also gleich zu Seite 3.

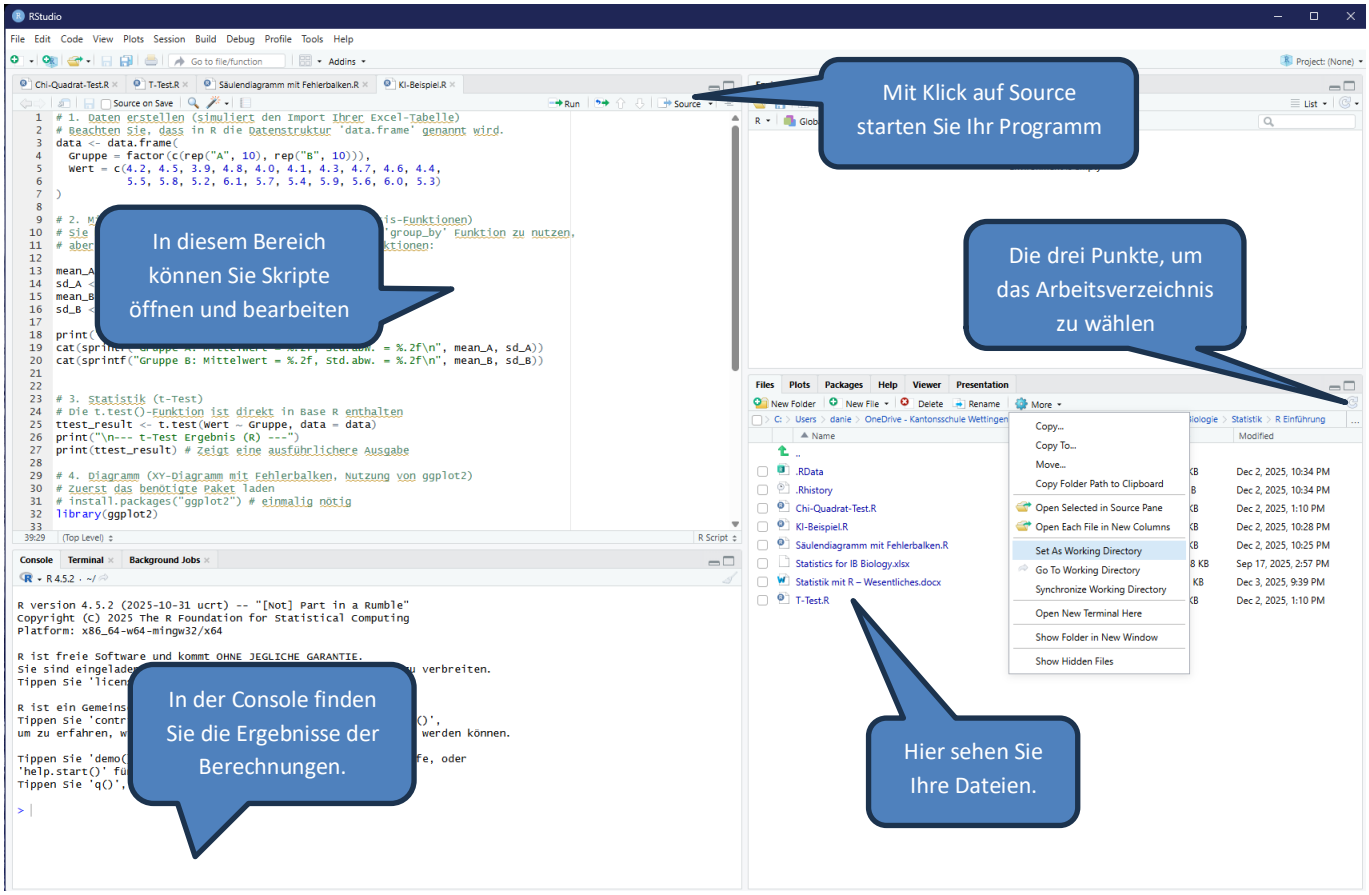
Installieren

Um mit R zu programmieren, braucht man die Programmiersprache R und RStudio, eine benutzerfreundliche Oberfläche, welche vieles erleichtert. Beides ist gratis für Windows wie MacOS erhältlich:

R: https://cran.r-project.org/
RStudio: https://posit.co/download/rstudio-desktop/#download
Eine genaue Anleitung (falls gewünscht): https://www.datacamp.com/de/tutorial/r-studio-tutorial
Installieren Sie R und danach RStudio. Starten Sie nun RStudio.

Einrichten und sich zurechtfinden

RStudio ist ein vier Bereiche gegliedert. Der linke obere Bereich sehen Sie erst, wenn Sie ein Programm geöffnet haben.



Um bequem auf Ihre Dateien zugreifen zu können, gehen Sie als ersten in Ihr Arbeitsverzeichnis, indem Sie auf die **drei Punkte** klicken. Klicken Sie auf **More** und dann **Set as working directory**.

So, nun kann es losgehen! Öffnen Sie ein R-Programm im Arbeitsverzeichnis oder erstellen Sie via File ein neues leeres Programm.

Programm können Sie starten, indem Sie auf **Source** klicken. Die Ergebnisse erscheinen unten links in der **Console**.

Die Console wird auch benötigt, um Zusatzpakete zu installieren. Schreiben Sie die folgenden Zeilen, wobei Sie nach jeder Enter klicken.

```
install.packages("readxl")
```

```
install.packages("ggplot2")
```

So, damit bekam R die Fähigkeit, Excel-Dateien zu lesen und komplexere Grafiken zu zeichnen. Diese Pakete mussten wir nur einmal pro Computer installieren.

Die Beispielprogramme

Das Excel-Dokument **Statistics for IB Biology.xlsx** enthält die Daten und auch die Excel-Berechnungen zum Vergleich. Beachten Sie, dass das Excel-Dokument geschlossen sein muss, wenn Sie mit einem R-Programm darauf zugreifen.

Empfohlener Lernprozess

Lesen Sie das Programm hier im Skript durch und versuchen Sie es zu verstehen. Öffnen Sie es in RStudio und starten Sie es. Schauen Sie unten bei «Nützliche Befehle» nach, wie diese funktionieren. Ändern Sie es, um es auf Ihre Daten anzupassen. Machen Sie sich im Internet schlau, um mehr zu lernen. Fragen Sie die KI, um Befehle erklärt zu bekommen oder neue Funktionen zu erfahren.

Mittelwerte und SD

```
# Mittelwerte und Standardabweichung mit Daten aus "Statistics for IB Biology.xlsx"

# Daten einlesen
library(readxl)
library(ggplot2)
daten_roh <- read_excel(
  path = "Statistics for IB Biology.xlsx",      # Name Ihrer Datei
  sheet = "mean and SD",                        # Name des Arbeitsblatts
  range = "B19:C23",                            # Der definierte Zellbereich
  col_names = c("ohne", "mit")                  # Namen für die zwei Spalten
)
werte_ohneDunger <- daten_roh$ohne
werte_mitDunger <- daten_roh$mit

# Deskriptive Statistik
mittelwert_ohneDunger <- mean(werte_ohneDunger)
mittelwert_mitDunger <- mean(werte_mitDunger)
sd_ohneDunger <- sd(werte_ohneDunger)
sd_mitDunger <- sd(werte_mitDunger)

cat(sprintf("Mittelwert und SD mit Dünger:  %.2f +-%.2f\n", mittelwert_mitDunger, sd_mitDunger))
cat(sprintf("Mittelwert und SD ohne Dünger: %.2f +-%.2f\n", mittelwert_ohneDunger, sd_ohneDunger))
```

Passen Sie diese 4 Werte an, um andere Daten aus anderen Excel-Dokumenten zu verwenden

Hier passiert die eigentliche Berechnung. Variablennamen wie sd_mitDunger dürfen Sie auf für Sie Logischeres anpassen

Sieht kompliziert aus (und ist es auch zu Beginn) Es dient aber nur die Ausgabe. Lesen Sie am Ende des Skripts nach, wie der sprintf() Befehl funktioniert

In der Console sehen Sie folgendes:

```
Mittelwert und SD mit Dünger:  18.82 +-1.45
Mittelwert und SD ohne Dünger: 11.90 +-1.95
```

Säulendiagramm mit Fehlerbalken

```
# Mittelwerte und Standardabweichung mit Daten aus "Statistics for IB Biology.xlsx"

# Daten einlesen
library(readxl)
library(ggplot2)
daten_roh <- read_excel(
  path = "Statistics for IB Biology.xlsx",      # Name Ihrer Datei
  sheet = "mean and SD",                        # Name des Arbeitsblatts
  range = "B19:C23",                            # Der definierte Zellbereich
  col_names = c("ohne", "mit")                  # Namen für die zwei Spalten
)
werte_ohneDunger <- daten_roh$ohne
werte_mitDunger <- daten_roh$mit

# Deskriptive Statistik
mittelwert_ohneDunger <- mean(werte_ohneDunger)
mittelwert_mitDunger <- mean(werte_mitDunger)
sd_ohneDunger <- sd(werte_ohneDunger)
sd_mitDunger <- sd(werte_mitDunger)

# Erstellen eines Data Frames, den ggplot2 einfach verarbeiten kann
summary_data <- data.frame(
  Gruppe = c("Ohne Dünger", "Mit Dünger"),
  Mittelwert = c(mittelwert_ohneDunger, mittelwert_mitDunger),
  SD = c(sd_ohneDunger, sd_mitDunger)
)

# Balkendiagramm mit Fehlerbalken
mein_duenger_plot <- ggplot(summary_data, aes(x = Gruppe, y = Mittelwert)) +

  # Balken zeichnen:
  geom_bar(stat = "identity", fill = "green", color = "black") +

  # Fehlerbalken hinzufügen:
  geom_errorbar(
    aes(ymin = Mittelwert - SD, ymax = Mittelwert + SD),
    width = 0.2,      # Breite der Endkappen
    linewidth = 1     # Dicke der Linie
  ) +

  # Beschriftungen und Design
  labs(
    title = "Einfluss von Dünger auf den Messwert",
    x = "Versuchsgruppe",
    y = "Mittelwert der Messungen"
  ) +
  theme_classic()

print(mein_duenger_plot)
```

Das ist der genau gleiche Code wie im Beispiel «Mittelwerte und SD»

Einiges klingt etwas umständlich, aber das muss uns nicht weiter kümmern. Wir brauchen nicht alles zu verstehen.

Alle hier rot geschriebenen Texte dienen nur der Beschriftung und dem Design des Diagramms – ändern Sie sie nach Belieben!

Wechseln Sie hier zwischen Ihren Dateien und den Plots

Chi-Quadrat-Test

```
# Chi-Quadrat-Test mit Daten aus "Statistics for IB Biology.xlsx"

# Daten einlesen
library(readxl)
daten_roh <- read_excel(
  path = "Statistics for IB Biology.xlsx",      # Name Ihrer Datei
  sheet = "Chi-squared test",                   # Name des Arbeitsblatts
  range = "B11:C16",                            # Der definierte Zellbereich
  col_names = c("Gemessen", "Erwartet")         # Name für die zwei Spalten
)

# Die eingelesenen Daten (Counts/Häufigkeiten)
erwartete_werte <- daten_roh$Erwartet
gemessene_werte <- daten_roh$Gemessen


# Der Chi-Quadrat-Test vergleicht die beobachteten Häufigkeiten (gemessene_werte)
# mit den erwarteten Wahrscheinlichkeiten (Proportionen).
# Wir berechnen die Proportionen, indem wir die erwarteten Werte durch deren Summe teilen:
erwartete_proportionen <- erwartete_werte / sum(erwartete_werte)

# Chi-Quadrat-Test
# Ist der p-Wert kleiner als 0.05, dann gibt es einen signifikanten Unterschied der erwarteten zu den
gemessenen Werten

chi_sq_test <- chisq.test(
  x = gemessene_werte,
  p = erwartete_proportionen
)

print(chi_sq_test)
```

Immer der sehr ähnliche Code um Daten aus Excel zu lesen…

Das Programm ist eigentlich sehr kurz. Das Meiste sind Kommentare.

Das Kernstück: Die eigentliche Berechnung des Chi-Quadrat-Tests

In der Console sehen Sie folgendes:

```
        Chi-squared test for given probabilities

data:  gemessene_werte
X-squared = 27.5, df = 5, p-value = 4.558e-05
```

Die Print-Anweisung liefert viele Informationen. Der p-Wert ist kleiner als 0.05 und somit gibt es einen Unterschied.

Zweistichproben-T-Test

```
# Zweistichproben-T-Test mit Daten aus "Statistics for IB Biology.xlsx"

# Daten einlesen
library(readxl)
daten_roh <- read_excel(
  path = "Statistics for IB Biology.xlsx",      # Name Ihrer Datei
  sheet = "t-test",                             # Name des Arbeitsblatts
  range = "B10:C15",                            # Der definierte Zellbereich
  col_names = c("Katzen", "Hunde")              # Namen für die zwei Spalten
)

# Die eingelesenen Daten (Counts/Häufigkeiten)
werte_katzen <- daten_roh$Katzen
werte_hunde <- daten_roh$Hunde

# Führt den t-Test durch. R verwendet standardmäßig den Welch-Test, der ungleiche Varianzen annimmt
(was meistens die sicherste Wahl ist).
# Ist der p-Wert kleiner als 0.05, sind die Mittelwerte der Katzen- und Hunde-Gruppe signifikant
unterschiedlich.

t_test_ergebnis <- t.test(werte_katzen, werte_hunde)

print(t_test_ergebnis)
```

Das ist inzwischen hoffentlich wohlbekannt…

Der eigentliche Test ist noch einfacher als der Chi-Quadrat-Test durchzuführen.

In der Console sehen Sie folgendes:

```
        Welch Two Sample t-test

data:  werte_katzen and werte_hunde
t = -1.3081, df = 9.7958, p-value = 0.2207
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -7.221636  1.888302
sample estimates:
mean of x mean of y
 7.666667 10.333333
```

Der p-Wert ist grösser als 0.05, die Daten zeigen keinen Unterschied zwischen Hunden und Katzen

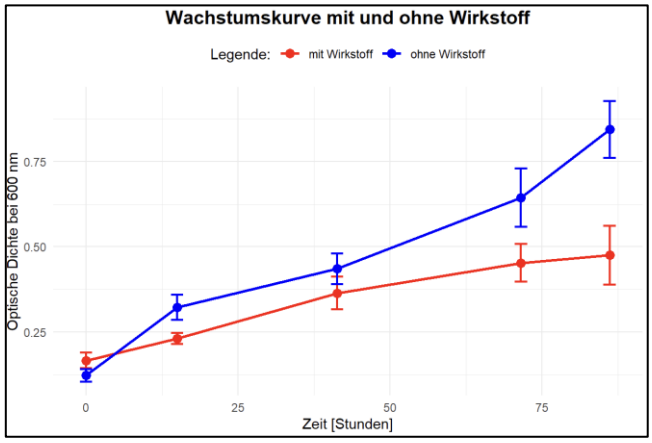
Bakterienversuch – XY-Streudiagramm mit Fehlerbalken

```
# Benötigte Bibliotheken laden
library(readxl)
library(dplyr)
library(ggplot2)

# Daten MIT Wirkstoff einlesen
mit_wirkstoff <- read_excel(
  path = "Bakterienversuch.xlsx",
  range = "A1:F6",
  col_names = TRUE
)

# Daten OHNE Wirkstoff einlesen
ohne_wirkstoff <- read_excel(
  path = "Bakterienversuch.xlsx",
  range = "A8:F13",
  col_names = TRUE
)

# Startzeit festlegen
startzeit <- min(mit_wirkstoff[[1]])

# Daten berechnen - MIT Wirkstoff
mit_stats <- mit_wirkstoff %>%
  mutate(Zeitdifferenz = as.numeric(difftime(.[[1]], startzeit, units = "hours"))) %>%
  rowwise() %>%
  mutate(
    Mittelwert = mean(c_across(2:6), na.rm = TRUE),
    SD = sd(c_across(2:6), na.rm = TRUE)
  ) %>%
  ungroup() %>%
  select(Zeitdifferenz, Mittelwert, SD) %>%
  mutate(Gruppe = "mit Wirkstoff")

# Daten berechnen - OHNE Wirkstoff
ohne_stats <- ohne_wirkstoff %>%
  mutate(Zeitdifferenz = as.numeric(difftime(.[[1]], startzeit, units = "hours"))) %>%
  rowwise() %>%
  mutate(
    Mittelwert = mean(c_across(2:6), na.rm = TRUE),
    SD = sd(c_across(2:6), na.rm = TRUE)
  ) %>%
  ungroup() %>%
  select(Zeitdifferenz, Mittelwert, SD) %>%
  mutate(Gruppe = "ohne Wirkstoff")

# Beide Datensätze kombinieren
alle_daten <- bind_rows(mit_stats, ohne_stats)

# XY-Diagramm erstellen
bakterien_diagramm <- ggplot(alle_daten,
  aes(x = Zeitdifferenz, y = Mittelwert, color = Gruppe)) +
  geom_line(linewidth = 1) +
  geom_point(size = 3) +
  geom_errorbar(aes(ymin = Mittelwert - SD, ymax = Mittelwert + SD),
                width = 2, linewidth = 0.8) +
  scale_color_manual(values = c("mit Wirkstoff" = "red", "ohne Wirkstoff" = "blue")) +
  labs(
    title = "Wachstumskurve mit und ohne Wirkstoff",
    x = "Zeit (Stunden)",
    y = "Optische Dichte bei 600 nm",
    color = "Legende:"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
    legend.position = "top"
  )
print(bakterien_diagramm)
```

Für dieses Programm brauchen wir eine zusätzliche Library: Mit **install.packes(«dplyr»)** installieren (siehe oben)

Der grösste Teils des Programms ist Datenaufbereitung. Hier werden die Kalenderdaten aus Excel in Anzahl Tage umgerechnet – viel lesbarer und für naturwissenschaftliche Grafiken üblich

Den %>% Operator können wir mit «und dann» übersetzen. Das ist eine Spezialität von R und erlaubt das einfachere Lesen von Code anstelle verschachtelter Klammern.



Auch in diesem Programm können wir alle rot geschriebenen Texte problemlos anpassen, um das Design zu individualisieren.

Nützliche Befehle

Alles nach dem #-Zeichen wird nicht als Code, sondern als Text interpretiert.

```
# Dies ist ein Kommentar

# 2+3 wird nicht berechnet, da es als Text gelesen wird
```

Zwischenergebnisse bleiben intern. Eine Ausgabe in der Console erscheint nur mit print() oder cat()

```
cat("Hallo")
```

In der Console lesen wir «Hallo»

Die Ausgabe mit `sprintf()` ist etwas kryptisch, erlaubt aber die clevere Kombination von Zahlen und Text. %.f sind Platzhalter für Zahlen, welche hinten am Text aufgelistet werden

```
sprintf("Hallo ihr %.f", 2)
```

In der Console lesen wir «Hallo ihr 2»

Neben dem f (float=Zahl mit Nachkommastellen) gibt es noch weitere Optionen, die aber seltener gebraucht werden. Nützlich hingegen ist, dass wir davor die Anzahl Nachkommastellen angeben können:

```
sprintf("Ein Drittel = %.5f", 1/3)
```

In der Console lesen wir «Ein Drittel = 0.33333»

Eine Zuweisung zu einer Variablen funktioniert mit dem Pfeil:

```
a <- 15
```

Wir können uns das so merken, dass wir a ändern möchten (deswegen schreiben wir es zuerst) und das, was reinkommt wird mit dem Pfeil angegeben. Wir können den Inhalt der Variablen verrechnen oder mit cat() ausgeben:

```
cat(a)
```

In der Console lesen wir «15»

Verwendete Werkzeuge und Quellen

Google Gemini: Erste Versionen von Code-Beispielen
Anthropic Claude: Code für Website auf margadant.net