

Bio Data Science

Prof. Dr. Jochen Kruppa

25. August 2022

Inhaltsverzeichnis

Willkommen

Auf den folgenden Seiten wirst du eine Menge über Statistik oder Data Science lernen. Du musst dafür nicht eine meiner Veranstaltungen besuchen. Gerne kannst du hier und dort einmal schauen, ob etwas für dich dabei ist. Das Skript wird fortlaufend von mir ergänzt. Neben dem Skript gibt es auch noch die erklärenden YouTube Videos. Ich freue mich, dass du Lust hast hier etwas zu lernen... oder aber du *musst* – da bald eine Klausur ansteht. Wie auch immer – schau dich einfach mal um. Im Anhang findest du auch einen kleinen Leitfaden für das Schreiben einer Abschlussarbeit. Vielleicht hilft dir die Anleitung ja beim Schreiben.

Gesammelte Klausurfragen Bio Data Science

Du findest die [gesammelten Klausurfragen auf GitHub](#). Die Klausurfragen zu den einzelnen Vorlesungen in einem Modul werden in den entsprechenden Übungen besprochen. Bitte komme in die Übungen.

Lernen...

Du liest hier gerade das Skript für [meine Vorlesungen](#) an der Hochschule Osnabrück an der Fakultät Agrarwissenschaften und Landschaftsarchitektur (AuL). Wie immer Leben kannst du auf verschiedene Arten und Weisen den Stoff, den ich vermitteln will, lernen. Daher gibt es noch zwei andere Möglichkeiten. Zum einen Lernen auf YouTube, mit meinen Lernvideos oder du schaust dir das Material auf GitHub an. Auf GitHub habe ich auch Informationen, die du vielleicht brauchen kannst. Ebenso findest du im Kapitel ?? noch andere Literaturempfehlungen.

... auf YouTube



Wenn du möchtest kannst du auf YouTube unter <https://www.youtube.com/c/JochenKruppa> noch einige Lehrvideos als Ergänzung schauen. In den Videos wiederhole ich Inhalte und du kannst auf Pause drücken um nochmal Programmierschritte nachverfolgen zu können.

... auf GitHub



Alle Materialien von mir findest du immer auf GitHub unter <https://github.com/jkruppa/teaching>. Selbst wenn du nicht mehr in einem meiner Kurse bist, kannst du so auf die Lehrinhalte immer nochmal zugreifen und die aktuellen Versionen haben. Auf GitHub liegt auch immer eine semesteraktuelle Version der [gesammelten Klausurfragen](#) für meine Module.

Kontakt

Wie erreichst du mich? Am einfachsten über die gute, alte E-Mail. Bitte beachte, dass gerade kurz vor den Prüfungen ich mehr E-Mails kriege. Leider kann es dann einen Tick dauern.



Einfach an j.kruppa@hs-osnabrueck.de schreiben. Du findest hier auch eine kurze Formulierungshilfe. Einfach auf den Ausklappfeil klicken.

Bitte gib immer in deiner E-Mail dein Modul - was du belegst - mit an. Pro Semester unterrichte ich immer *drei* sehr ähnlich klingende Module. Daher schau nochmal [hier in der Liste](#), wenn du unsicher bist.

💡 E-Mailvorlage mit beispielhafter Anrede

Hallo Herr Kruppa,
... ich belege gerade Ihr Modul **Modulname** und hätte eine Bitte/Frage/Anregung...
... ich benötige Hilfe bei der Planung/Auswertung meiner Bachelorarbeit...
Mit freundlichen Grüßen
M. Muster

1 Organisation

Den Teil kannst du hier überspringen, wenn es dich nicht so richtig interessiert, was ich alles an Vorlesungen anbiete. Wenn es dir um *statistische* Inhalte geht, dann gehe einfach weiter in das nächste Kapitel.

1.1 Statistische Beratung

Ich biete auch Termine für die statistische Beratung von Abschlussarbeiten sowie Projekten an. Dafür musst du mir einfach nur eine [E-Mail schreiben](#) und dann erhältst du einen Termin innerhalb der nächsten zwei Wochen.

Die Beratung ist grundsätzlich anonym und vertraulich. Wenn du willst kannst du gerne noch dein:e Betreuer:in mitbringen. Das ist aber keine Voraussetzung oder Notwendigkeit.

1.2 R Tutorium

Im Rahmen der statistischen Beratung bieten wir auch ein R Tutorium für alle Mitglieder:innen der Fakultät Agrarwissenschaften und Landschaftsarchitektur (AuL) an. Die aktuellen Termine findest du in Tabelle ??.

Im R Tutorium besprechen wir aktuelle Themen der Teilnehmer:innen. Meist sind dies aktuelle Fragen zu den Bachelorarbeiten. Auch wenn du kein dringendes Problem hats, kannst du gerne kommen und dir die Fragestellungen anhören. Bitte beachte folgende Hinweise zu den Terminen.

Hinweise zu dem R Tutorium

Das R Tutorium findet **nicht** im Prüfungszeitraum der Fakultät Agrarwissenschaften und Landschaftsarchitektur (AuL) statt.

Das R Tutorium findet **nicht** im *Februar* und *März* statt.

Das R Tutorium findet **nicht** im *August* und *September* statt.

Tabelle 1.1: Aktuelle Termine des R Tutoriums im Semester

Termin	Uhrzeit	Raum	Anmerkung
<i>nächste</i>	<i>Termine</i>	<i>ab</i>	<i>Oktober 2022</i>

1.3 Vorlesungen an der Hochschule Osnabrück

Von mir angebotene Vorlesungen werden an der Hochschule Osnabrück an der Fakultät Agrarwissenschaften und Landschaftsarchitektur (AuL) in ILIAS verwaltet. Alle notwendigen Informationen und Materialien sind auf ILIAS unter <https://lms.hs-osnabrueck.de/> zu finden. Wenn du in dem Kurs nicht angemeldet bist, dann kontaktiere mich bitte **per Mail**. Auch die Kommunikation erfolgt von meiner Seite aus über ILIAS.

Wenn du nicht in der Fakultät Agrarwissenschaften und Landschaftsarchitektur (AuL) studierst oder aber in einem Studiengang, der meine Module nicht anbietet, steht es dir natürlich frei, sich in meine Vorlesungen zu setzen. Du findest in Tabelle ?? eine Übersicht der angebotenen Module und auch die inhaltliche Ordnung nach Lernstufe. Bitte informiere dich in deinem Studierendensekretariat über die Modalitäten zur Prüfungsteilnahme.

Eine inhaltliche Übersicht findet sich auf dem [Google Spreadsheet](#). Die Planung ist aktuell (Stand Sommer 2022) noch nicht abgeschlossen. Im Zweifel einfach bei mir einmal **per Mail** anfragen.

Auf ILIAS findest du alle aktuellen Kursinformationen und erhältst auch die Mails, wenn Änderungen im Kursablauf stattfinden.

Abbildung 1.2: Angebotene Statistik Module an der Fakultät Agrarwissenschaften und Landschaftsarchitektur (AuL).
 Die Spaltenüberschriften geben das Lernniveau an.

	Landwirtschaft; Angewandte Pflanzen- biologie – Gartenbau, Pflanzen- technologie	Wirtschafts- ingenieur- wesen Agrar / Le- bensmittel	Bioverfahrenstechnik in Agrar- und Lebensmit- telwirtschaft	Angewandte Nutztier- technik Pflanzenwis- sen- schaften
1	Mathematik und Statistik	Statistik	Angewandte Statistik für Bioverfahrens- technik	
2	Angewandte Statistik und Versuchswesen	Angewandte Statistik und Versuchswesen		
3	Spezielle Statistik und Versuchswesen			Biostatistik

2 Literatur

Was ist gute Literatur? Immer schwer zu beurteilen. Im folgenden liste ich einige Literaturquellen auf. Zum einen basiert eine Menge von dem R Code auf Wickham (2016) zum Anderen möchtest du dich vielleicht nochmal rechts oder links weiter bilden. Du musst aber nicht um die Klausur bestehen zu können. Siehe es eher als ein Angebot.

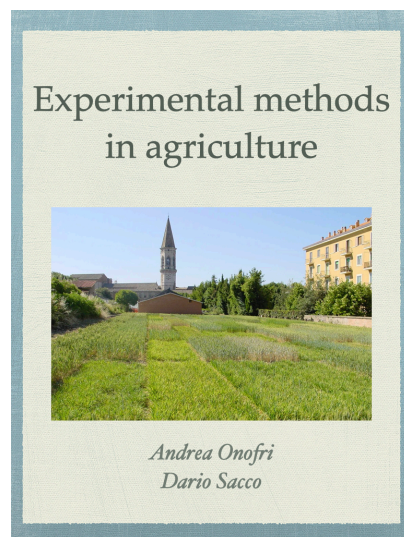
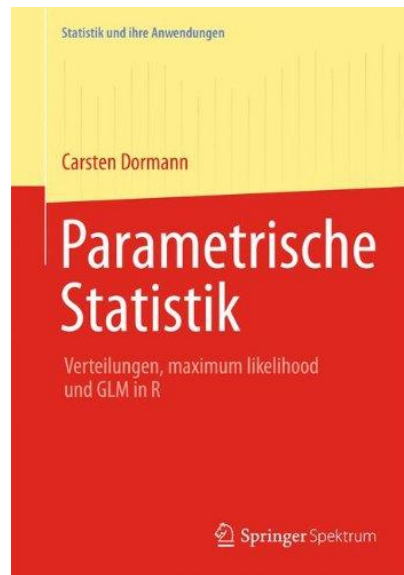
Die Frage nach der Klausur...

Und daher hier nochmal gleich zu Anfang, es ist nicht notwendig mehr als das Skript durchzuarbeiten und bei den Übungen zu sein um die Klausur zu bestehen. Für deine Bachelorarbeit wirst du aber Programmieren in R können müssen.

Neben diesem Modul musst du vermutlich noch andere Module belegen. Deshalb hier eine Auswahl Literatur, die dir helfen mag. Zum einen ist die Literatur anders geschrieben und zum anderen sind dort andere Inhalte.

2.1 Parametrische Statistik

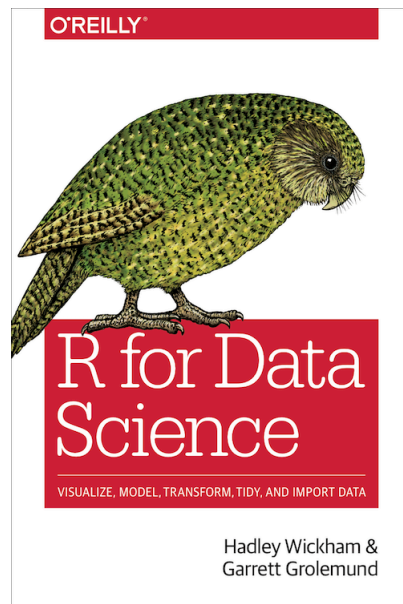
Dormann (2013) liefert ein tolles deutsches Buch für die Vertiefung in die Statistik. Insbesondere wenn du wissenschaftlich Arbeiten willst weit über die Bachelorarbeit hinaus. Dormann baut in seinem Buch eine hervorragende Grundlage auf. Das Buch ist an der Hochschule Osnabrück kostenlos [über den Link](#) zu erhalten.



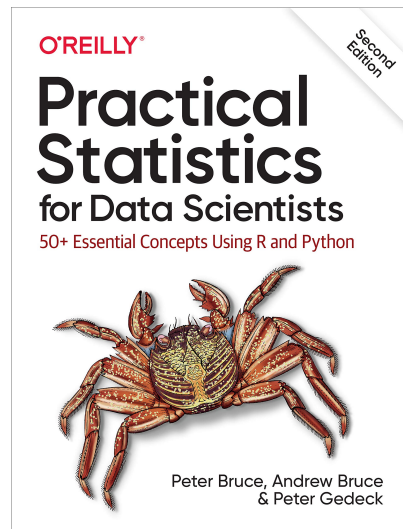
2.2 Experimental methods in agriculture

Onofri und Sacco (2021) haben das Buch [Experimental methods in agriculture](#) geschrieben. Wir werden auf dieses englische Buch ab und zu mal verweisen. Insbesondere der Einleitungstext zur Wissenschaft und dem Design von Experimenten ist immer wieder lesenswert. Spätere Teile des Buches sind etwas mathematischer und nicht für den Einstieg unbedingt geeignet. Aber schau es dir selber an.

2.3 R for Data Science



Wickham (2016) ist die Grundlage für die R Programmierung. Das Material von Wickham findet sich kostenlos online unter <https://r4ds.had.co.nz/> und <https://www.tidyverse.org/>. Wir werden uns hauptsächlich mit R wie es Wickham lehrt beschäftigen. Somit ist Wickham unsere Grundlage für R.



2.4 Practical Statistics for Data Scientists

Bruce (2020) schreibt ein Buch für den Anwender. Ohne Vorkenntnisse ist das Buch vermutlich etwas schwer zu lesen. Dafür bietet das Buch aber *nach* einem Statistikkurs sehr gute Anknüpfungspunkte Richtung maschinelles Lernen und somit der Klassifikation. Das Buch ist auch hier in der [englischen Version](#) und hier in der [deutschen Version](#) zu erhalten. *Beide Links benötigen den Zugang über die Hochschule Osnabrück.*

2.5 Data Science for Agriculture in R

Schmidt liefert auf der Webseite <https://schmidtpaul.github.io/DSFAIR/index.html> eine tolle Sammlung an experimentellen Designs bzw. Versuchsanlagen samt der Auswertung in R. Ohne Vorkenntnisse schwer zu verstehen. Sollte aber nach einem Kurs Statistik dann möglich sein. Gerne hier auch mich fragen, dann können wir gemeinsam das passende Design raussuchen und besprechen.



Odds & Ends

Introducing Probability & Decision with a Visual Emphasis

2.6 Odds & Ends

Am Ende dann noch eine Mathebuch von Weisberg zu finden unter <https://jonathanweisberg.org/vip/>. Eigentlich eher ein Buch über Wahrscheinlichkeiten und wenn ein Buch am Ende stehen muss, dann ist es dieses Buch. Ich finde es sehr spannend zu lesen, aber das ist dann vermutlich *special intrest*.

Referenzen

3 Einführung

In diesem Kapitel nenne ich die wichtigsten Lernziele, die nach dem Lesen des Skriptes von dir erreicht worden sein sollten. Je nach besuchten Kurs kann natürlich nicht alles geschafft worden sein. So sehe diese Übersicht als Einführung für das was später kommt. Wenn du die Beispiele hier verstehst, dann hast du eine gute und solide Grundlage in Statistik und Bio Data Science.

Ein Wort der Warnung...

Wenn du dieses Bild eines niedergeschlagenen *Engels der Statistik* siehst...



... dann bedeutet der niedergeschlagene Engel der Statistik:

- 1) Wir opfern Genauigkeit für Anwendbarkeit. Ja, manchmal ist es eben statistisch nicht richtig was hier steht, aber aus Gründen der Anwendung fahren wir mal über den Engel drüber. *Schade.*
- 2) Wir sind hier Anfänger und Anwender. Später kannst du noch tiefer ins Detail gehen. Hier wollen wir die Grundlagen lernen. Das hat dann einen Preis an *Richtigkeit.*
- 3) Wir wollen fertig werden. Durch geschicktes Manövrieren können wir an einen Punkt kommen, wo kein statistischer Test mehr passt. Das wollen wir nicht. Deshalb zahlen wir hier auch einen Preis. Passt aber.

Deshalb konzentrieren wir uns auf einige wichtige Lernziele, die wir jetzt einmal nacheinander durchgehen.

3.1 Lernziel 1: Eine explorative Datenanalyse durchführen

Gleich zu Beginn R Code zu zeigen und eine entsprechende Abbildung ist vielleicht ungewöhnlich, aber wir wollen zu dieser Abbildung ?? hin. In Abbildung ?? siehst du einen Boxplot. Und wie wir aus den Daten `flea_dog_cat.xlsx` einen Boxplot erstellen, das soll uns in den nächsten Kapitel beschäftigen. Dafür müssen wir nämlich eine Menge in dem Codeblock verstehen und dann auch Anwenden können. Und natürlich lernen was eigentlich ein Boxplot ist und was in einem Boxplot eigentlich dargestellt ist.

Data

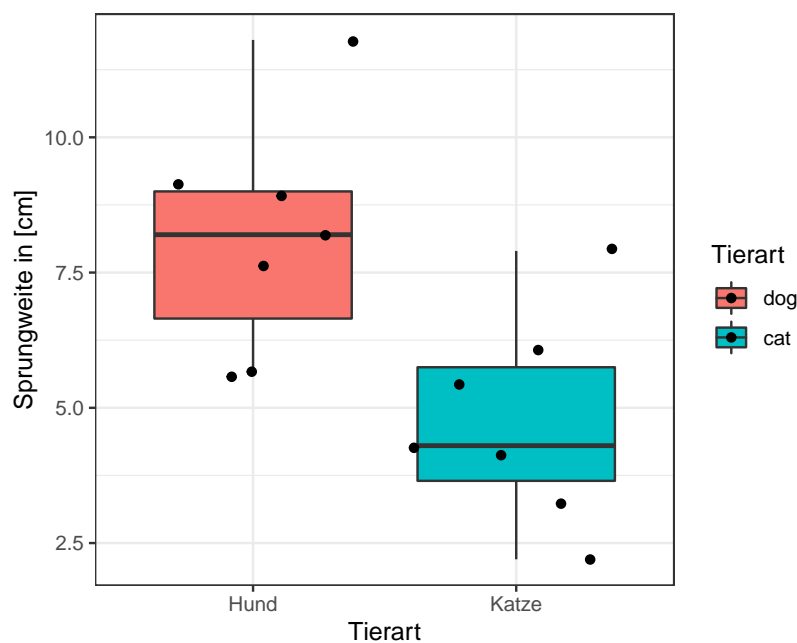


Abbildung 3.1: Boxplot der Sprungweiten [cm] von Hunden und Katzen.

Hier ist der Codeblock der in R die Abbildung ?? erstellt.

```

## Einlesen von Daten aus Excel
data_tbl <- read_excel("data/flea_dog_cat.xlsx")

## Umformen der <chr> Spalte in einen Factor <fct>
data_tbl <- data_tbl %>%
  mutate(animal = as_factor(animal))

## Auswählen der wichtigen Spalten für den Boxplot
data_tbl <- data_tbl %>%
  select(animal, jump_length)

## Generieren des Boxplots in ggplot()
ggplot(data_tbl, aes(x = animal, y = jump_length,
                     fill = animal)) +
  geom_boxplot() +
  geom_jitter() +
  labs(x = "Tierart", y = "Sprungweite in [cm]",
       fill = "Tierart") +
  scale_x_discrete(labels = c("Hund", "Katze")) +
  theme_bw()

```

Wir müssen nun folgende Dinge lernen um den Codeblock zu verstehen:

- Wir müssen das Datenbeispiel verstehen. Was sind das eigentlich für Daten, die wir da abbilden? Was sind überhaupt Daten im Sinne der Statistik bzw. für R.
- Wir müssen den R Code verstehen. Von einzelnen wichtigen Operatoren wie `->` und `%>%` zu dem den Unterschieden von Worten und Objekten.
- Wie kriegen wir Daten aus Excel in R hinein? Wir können die Daten ja nicht einfach in R eintragen sondern haben die Daten ja meist in einer (Excel) Datei wie `flea_dog_cat.xlsx`.
- Was ist eigentlich ein Boxplot und welche statistischen Maßzahlen werden hier eigentlich abgebildet?
- Wie funktioniert eigentlich die Funktion `ggplot()` mit der wir den Boxplot erstellt haben?

All diese Fragen und weitere Fragen, die sich diesen Fragen anschließen, wollen wir uns in den nächsten Kapitel anschauen.

Leider kann ich hier nur *linear* schreiben. Deshalb musst du eventuell mal ein Kapitel wiederholen oder etwas quer lesen. Du kannst dir ja auch nicht immer alles auf einmal merken.

3.2 Lernziel 2: RStudio und R

Um Data Science durchführen zu können musst du etwas Programmieren können. Wir programmieren in R und nutzen die Software um Abbildungen zu erstellen und Analysen zu rechnen.

Wir arbeiten in R und nutzen dafür das RStudio. Führe einfach folgende Schritte aus um erst R zu installieren und dann das RStudio.

1. R installieren unter <https://cran.rstudio.com/>
2. RStudio installieren unter <https://www.rstudio.com/products/rstudio/download/#download>

Bitte die Reihenfolge beachten. Beide Schritte kannst du dir auch nochmals im Video anschauen oder aber du kommst in das R Tutorium was regelmäßig an der Hochschule Osnabrück von mir angeboten wird. Die Termine findest du im Kapitel ??.

 Was ist eigentlich RStudio und woher kriege ich das?

Du findest auf YouTube [Einführung in R - Teil 01 - Installation von RStudio und R](#) als Video. Ich gehe in dem Video einmal alle wichtigen Schritte durch und so kannst du dir Rstudio und R installieren.

3.3 Lernziel 3: Statistische Versuche verstehen

Wie funktioniert ein *statistischer* Versuch? Ich könnte auch wissenschaftliches Experiment schreiben, aber ein wissenschaftliches Experiment ist sehr abstrakt. Wir wollen ja einen Versuch durchführen und danach - ja was eigentlich? Was wollen wir nach dem Versuch haben? Meistens eine neue Erkenntnis. Um

diese Erkenntnis zu validieren oder aber abzusichern nutzen wir Statistik. Dazu musst du noch wissen, dass wir eine spezielle Form der Statistik nutzen: die *frequentistische Statistik*.

Die *frequentistische Statistik* basiert - wie der Name andeutet - auf Wiederholungen in einem Versuch. Daher der Name frequentistisch. Also eine Frequenz von Beobachtungen. Ist ein wenig gewollt, aber daran gewöhnen wir uns schon mal. Konkret, ein Experiment welches wir frequentistisch Auswerten wollen besteht immer aus biologischen Wiederholungen. Wir müssen also ein Experiment planen in dem wir wiederholt ein Outcome an vielen Tieren, Pflanzen oder Menschen messen. Auf das Outcome gehen wir noch später ein. Im Weiteren konzentrieren wir uns hier auf die *parametrische Statistik*. Die parametrische Statistik beschäftigt sich mit Parametern von Verteilungen. Ein schwieriger Satz. Schauen wir uns einmal eine Verteilung an.

3.3.1 Poissonverteilung

Abbildung ?? zeigt eine Poissonverteilung. Eine Poissonverteilung beschreibt Zählraten. Mehr zu der Poissonverteilung findest du im Kapitel ???. Wir zählen bei 39 Hunden wieviele Flöhe jeder Hund jeweils hatte. Danach zeichnen wir uns einen Dotplot der die Verteilung der Anzahl Flöhe auf den Hunden widerspiegelt.

Eine Verteilung hat Parameter. Parameter sind die Eigenschaften einer Verteilung, die notwendig sind um eine Verteilung vollständig zu beschreiben.

Im Falle der Poissonverteilung brauchen wir nur einen Parameter für den höchsten Punkt der Kurve. Wir nennen diesen Punkt *Lambda* (λ). Die Ausbreitung der Kurve ist eine Funktion von λ und steigt mit λ an.

3.3.2 Normalverteilung

Abbildung ?? zeigt eine Normalverteilung. Mehr zu der Poissonverteilung findest du im Kapitel ???. Hier haben wir das Flohgewicht von den Flöhen von 24 Hunden gemessen, die mit Flöhen befallen waren. Wir sehen, dass sich eine Glockenkurve

biologische Wiederholung
Ein Punkt in Response y.
oder Mensch. Eine **technische**
Wiederholung ist die gleiche
Messung an dem gleichen Tier,
Pflanze oder Mensch.

Parameter sind Zahlen, die eine
Verteilungskurve beschreiben.

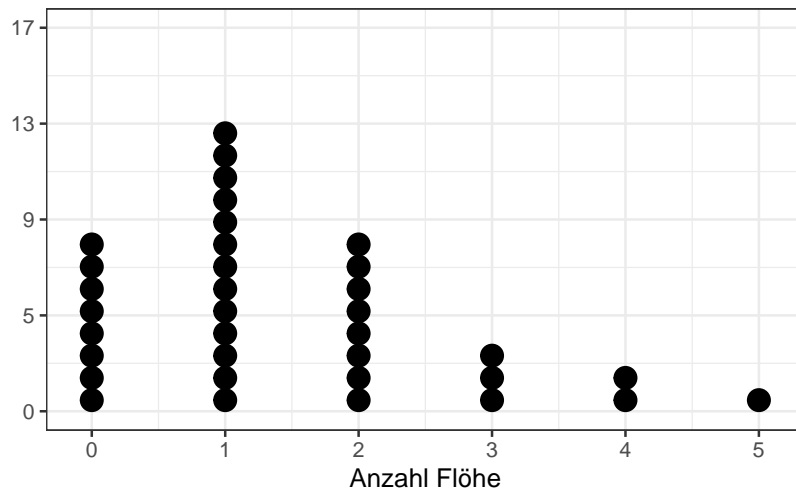


Abbildung 3.2: An 39 Hunden wurde die Anzahl an Flöhen gezählt.

bildet oder zumindestens etwas ähnliches. Wir können annehmen, dass das Gewicht *approximativ* normalverteilt ist.

Im Falle der Normalverteilung brauchen wir einen Parameter für den höchsten Punkt der Kurve, sowie einen Parameter für die Ausbreitung, also wie weit geht die Kurve nach links und nach rechts. Der Mittelwert \bar{y} beschreibt den höchsten Punkt einer Normalverteilung. Die Standardabweichung s_y beschreibt die Ausbreitung einer Normalverteilung.

Wir nutzen das Wort **approximativ** wenn wir sagen wollen, dass ein Outcome näherungsweise normalverteilt ist.

i Wie gehen wir nun vor, wenn wir ein Experiment durchführen wollen?

- 1) Wir müssen auf jeden Fall wiederholt ein Outcome an verschiedenen Tieren, Pflanzen oder Menschen messen.
- 2) Wir überlegen uns aus welcher Verteilungsfamilie unser Outcome stammt, damit wir dann die entsprechende Verfahren zur Analyse nehmen können.

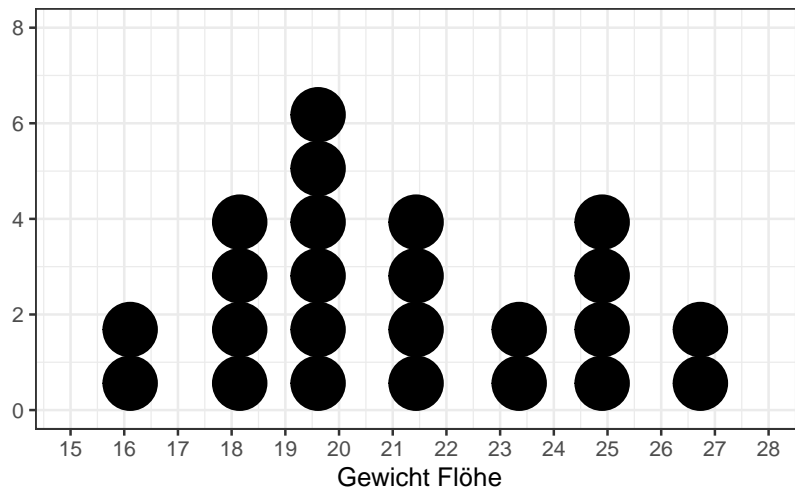


Abbildung 3.3: An 39 Hunden wurde die Anzahl an Flöhen gezählt.

3.4 Lernziel 4: Falsifikationsprinzip

💡 Grundlagen der Wissenschaft und Falsifikationsprinzip

Du findest auf YouTube [Grundlagen der Wissenschaft und Falsifikationsprinzip](#) als Video Reihe.

Wenn wir ein Experiment durchführen dann erheben wir einmalig Daten D_1 . Wir könnten das Experiment wiederholen und erneut Daten D_2 erheben. Wir können das Experiment j -mal wiederholen und haben dann Daten von D_1, \dots, D_j . Dennoch werden wir nie *alle* Daten erheben können, die mit einem Experiment verbunden sind.

Nehmen wir das Beispiel, dass wir die Sprungweite von Hunde- und Katzenflöhen vergleichen wollen. Wir können nicht *alle* Hunde- und Katzenflöhe messen. Wir können nur eine Stichprobe an Daten D_1 erheben. Über diese Daten D_1 können wir dann später durch statistische Algorithmen eine Aussage treffen. Wichtig ist hier sich zu merken, dass wir eine Grundgesamtheit haben aus der wir eine Stichprobe ziehen. Wir müssen darauf achten, dass die Stichprobe *repräsentativ* ist und damit *strukturgleich* zur Grundgesamtheit ist. Die Strukturgleichheit

Strukturgleichheit erreichen wir durch **Randomisierung**.

erreichen wir durch Randomisierung. Wir veranschaulichen diesen Zusammenhang in Abbildung ?? . Ein Rückschluß von der Stichprobe ist nur möglich, wenn die Stichprobe die Grundgesamtheit repräsentiert. Auch eine Randomisierung mag dieses Ziel nicht immer erreichen. Im Beispiel der Hundeflöhe könnte wir eine Art an Flöhen übersehen und diese Flohart nicht mit in die Stichprobe aufnehmen. Ein Rückschluß auf diese Flohart wäre dann mit unserem Experiment nicht möglich.

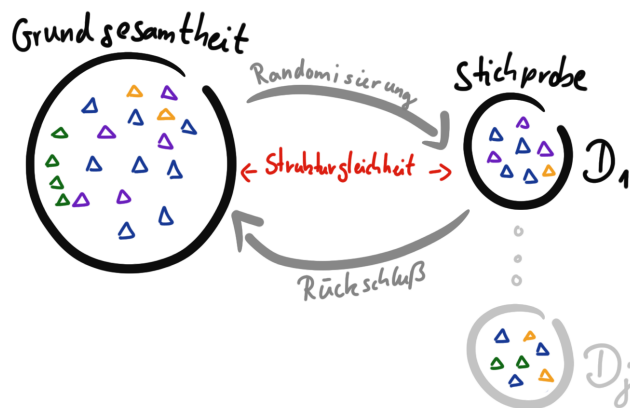


Abbildung 3.4: Abbildung über die Grundgesamtheit und die Stichprobe(n) D_1 bis D_j . Durch Randomisierung wird Sturkturgleichheit erreicht, die dann einen Rückschluß von der Stichprobe auf die Grundgesamtheit erlaubt. Jede Stichprobe ist anders und nicht jede Randomisierung ist erfolgreich was die Strukturgleichheit betrifft.

Tabelle ?? zeigt nochmal die Zusammenfassung von der Grundgesamtheit un der Stichprobe im Vergleich. Wichtig ist zu merken, dass wir mit unserem kleinen Experiment Daten D generieren mit denen wir einen Rückschluß und somit eine Verallgemeinerung erreichen wollen.

Tabelle 3.1: Vergleich von Grundgesamtheit und Stichprobe.

Grundgesamtheit	Stichprobe
... n ist riesig bis unfassbar.	... n von D ist klein.
... der Mittelwert wird mit μ_y beschrieben.	... der Mittelwert wird mit \bar{y} beschrieben.

Grundgesamtheit	Stichprobe
... die Varianz wird mit σ^2 beschrieben.	... die Varianz wird mit s^2 beschrieben.
... die Standardabweichung wird mit σ beschrieben.	... die Standardabweichung wird mit s beschrieben.

Teil I

Datenbeispiele

Wir brauchen am Anfang erstmal ein simples Beispiel. Konkrete Zahlen mit denen wir arbeiten können und Grundlagen aufbauen können. Was liegt da näher als sich einmal am Kopf zu kratzen und zu fragen, was juckt den da? Genau! Flöhe. Wir schauen uns einmal Flöhe auf Hunden und Katzen an. Daran können wir viel über Zahlen und Buchstaben in der Statistik und dann im Programmieren lernen.

i Zahlen, Buchstaben und Wörter

Mir ist bewusst, dass du die Unterschiede kennst. Nur leider ist eine Zahl nicht nur eine Zahl und ein Wort nicht immer ein Wort. Das hat mit der eingeschränkten Kommunikationsfähigkeit von Computerprogrammen zu tun. R braucht da deine Mithilfe und dein *neues* Verständnis von Buchstaben und Zahlen. Eben wie ein Computer denkt.

Von Flöhen und Hunden

In unserem ersten Beispiel in Kapitel ?? geht es darum einmal ein Gefühl für Daten zu kriegen. Also was sind diese Zahlen und Buchstaben eigentlich? Wie sind Daten aufgebaut und wie musst du Daten bauen, so dass wir auch mit den Daten arbeiten können? Wir schauen uns dafür einmal Flöhe auf Hunden an und fragen uns welche Typen von Zahlen können wir erheben?

Von Flöhen, Hunden und Katzen

In unserem zweiten Beispiel in Kapitel ?? erweitern wir unserer erstes Beispiel aus Kapitel ?? um die Katzen. Das heist, dass eigentlich alles gleich bleibt. Wir schauen uns *zusätzlich* noch als zweite Gruppe die Katzen an. Nun können wir die Frage stellen, unterscheiden sich Flöhe auf Hunden und Katzen gegeben von gemessenen Eigenschaften?

Von Flöhen auf Tieren

In unserem dritten Beispiel in Kapitel ?? erweitern wir das Beispiel um den Fuchs mit einem weiteren Tier. Dadurch haben wir nicht mehr einen Faktor mit zwei Leveln vorliegen sondern einen mit drei Leveln. Die Fragestellung erweitert sich jetzt auf einen *multiplen* Gruppenvergleich. Wir vergleichen nicht mehr nur noch zwei Gruppen miteinander sondern drei.

Von Flöhen auf Tieren in Habitaten

In unserem vierten Beispiel in Kapitel ?? schauen wir uns zusätzlich zu dem dritten Beispiel in Kapitel ?? noch verschiedene Habitate (eng. *site*) an. Wir haben nämlich die Hunde-, Katzen-, und Fuchsflöhe nicht nur an einem Ort sondern an verschiedenen Orten gesammelt und gemessen. Wir haben einen zweiten Faktor vorliegen.

4 Von Flöhen und Hunden

In unserem ersten Beispiel wollen wir uns verschiedene Daten D von Hunden und Hundeflöhen anschauen. Unter anderem sind dies die Sprungweite, die Anzahl an Flöhen, die Boniturnoten auf einer Hundemesse sowie der Infektionsstatus. Hier nochmal detailliert, was wir uns im folgenden im Kapitel einmal anschauen wollen.

- **Sprungweite** in [cm] von verschiedenen Flöhen

$$Y_{jump} = \{5.7, 8.9, 11.8, 8.2, 5.6, 9.1, 7.6\}.$$

- **Anzahl an Flöhen** auf verschiedenen Hunden

$$Y_{count} = \{18, 22, 17, 12, 23, 18, 21\}.$$

- **Boniturnoten** [1 = schlechteste bis 9 = beste Note] von verschiedenen Hunden

$$Y_{grade} = \{8, 8, 6, 8, 7, 7, 9\}.$$

- **Infektionsstatus** [0 = gesund, 1 = infiziert] mit Flöhen von verschiedenen Hunden

$$Y_{infected} = \{0, 1, 1, 0, 1, 0, 0\}.$$

Je nachdem was wir messen, nimmt Y andere Zahlenräume an. Wir sagen, Y folgt einer Verteilung. Die Sprungweite ist normalverteilt, die Anzahl an Flöhen folgt einer Poisson Verteilung, die Boniturnoten sind multinominal/ordinal bzw. kategorial verteilt. Der Infektionsstatus ist binomial verteilt. Wir werden uns später die Verteilungen anschauen und visualisieren. Das können wir hier aber noch nicht. Wichtig ist, dass du schon

mal gehört hast, dass Y unterschiedlich *verteilt* ist, je nachdem welche Dinge wir messen.

Tabelle ?? zeigt dir die Darstellung der Daten von oben in einer einzigen Tabelle. Bitte achte, dass genau eine Zeile für eine Beobachtung, in diesem Fall einem Hund, vorgesehen ist.

Tabelle 4.1: Sprunglängen [cm] für Hundeflöhe. Die Tabelle ist im Long-Format dargestellt.

animal	jump_length	flea_count	grade	infected
dog	5.7	18	8	0
dog	8.9	22	8	1
dog	11.8	17	6	1
dog	8.2	12	8	0
dog	5.6	23	7	1
dog	9.1	18	7	0
dog	7.6	21	9	0

💡 Datei für von Flöhen und Hunden

Du findest die Datei `flea_dog.xlsx` auf GitHub jkrup-pa.github.io/data/ als Excel oder auch als CSV.

5 Von Flöhen, Hunden und Katzen

Wir wollen jetzt das Beispiel von den Hunden und Flöhen um eine Spezies erweitern. Wir nehmen noch die Katzen mit dazu und fragen uns, wie sieht es mit der Sprungfähigkeit von Katzen und Hundeflöhen aus? Konzentrieren wir uns hier einmal auf die Sprungweite. Wir können wie in dem Beispiel ?? die Sprungweiten [cm] wieder aufschreiben:

$$Y_{jump} = \{3.2, 2.2, 5.4, 4.1, 4.3, 7.9, 6.1\}.$$

Wenn wir jetzt die Sprungweiten der Hundeflöhe mit den Katzenflöhen vergleichen wollen haben wir ein Problem. Beide Zahlenvektoren heißen gleich, nämlich Y_{jump} . Wir könnten jeweils in die Indizes noch *dog* und *cat* schreiben als $Y_{jump, dog}$ und $Y_{jump, cat}$ und erhalten folgende Vektoren.

$$Y_{jump, dog} = \{5.7, 8.9, 11.8, 8.2, 5.6, 9.1, 7.6\}$$

$$Y_{jump, cat} = \{3.2, 2.2, 5.4, 4.1, 4.3, 7.9, 6.1\}$$

Dadurch werden die Indizes immer länger und unübersichtlicher. Auch das Y einfach Y_{dog} oder Y_{cat} zu nennen ist keine Lösung - wir wollen uns vielleicht später nicht nur die Sprungweite vergleichen, sondern vielleicht auch die Anzahl an Flöhen oder den Infektionsstatus. Dann ständen wir wieder vor dem Problem die Y für die verschiedenen Outcomes zu unterscheiden. Daher erstellen wir uns die Tabelle ?. Wir haben jetzt eine *Datentabelle*.

Tabelle 5.1: Sprunglängen [cm] für Hunde- und Katzenflöhe.
Die Tabelle ist im Wide-Format dargestellt.

dog	cat
5.7	3.2
8.9	2.2
11.8	5.4
8.2	4.1
5.6	4.3
9.1	7.9
7.6	6.1

Intuitiv ist die Tabelle ?? übersichtlich und beinhaltet die Informationen die wir wollten. Dennoch haben wir das Problem, das wir in dieser Tabelle ?? nicht noch weitere Outcomes angeben können. Wir können die Anzahl an Flöhen auf den Hunde und Katzen nicht darstellen. Als Lösung ändern wir die Tabelle ?? in das Long-Format. Dargestellt in Tabelle ?. Jede Beobachtung belegt nun eine Zeile. Dies ist sehr wichtig im Kopf zu behalten, wenn du eigene Daten in z.B. Excel eingibts.

Tabelle 5.2: Tabelle der Sprunglängen [cm], Anzahl an Flöhen, Boniturnote sowie der Infektionsstatus von Hunde- und Katzenflöhe. Die Tabelle ist im Long-Format dargestellt.

animal	jump_length	flea_count	grade	infected
dog	5.7	18	8	0
dog	8.9	22	8	1
dog	11.8	17	6	1
dog	8.2	12	8	0
dog	5.6	23	7	1
dog	9.1	18	7	0
dog	7.6	21	9	0
cat	3.2	12	7	1
cat	2.2	13	5	0
cat	5.4	11	7	0
cat	4.1	12	6	0
cat	4.3	16	6	1
cat	7.9	9	6	0

animal	jump_length	flea_count	grade	infected
cat	6.1	7	5	0

💡 Datei für von Flöhen, Hunden und Katzen

Du findest die Datei `flea_dog_cat.xlsx` auf GitHub jkruppa.github.io/data/ als Excel oder auch als CSV.

6 Von Flöhen auf Tieren

Wir wollen jetzt das Beispiel von den Hunde- und Katzenflöhen um eine *weitere* Spezies erweitern. Wir nehmen noch die Füchse mit dazu und fragen uns, wie sieht es mit der Sprungfähigkeit von Hunde-, Katzen- und Fuchsflöhen aus?

Tabelle 6.1: Sprunglängen [cm] für Hunde-, Katzen- und Fuchsflöhe.

animal	jump_length	flea_count	grade	infected
dog	5.7	18	8	0
dog	8.9	22	8	1
dog	11.8	17	6	1
dog	8.2	12	8	0
dog	5.6	23	7	1
dog	9.1	18	7	0
dog	7.6	21	9	0
cat	3.2	12	7	1
cat	2.2	13	5	0
cat	5.4	11	7	0
cat	4.1	12	6	0
cat	4.3	16	6	1
cat	7.9	9	6	0
cat	6.1	7	5	0
fox	7.7	21	5	1
fox	8.1	25	4	1
fox	9.1	31	4	1
fox	9.7	12	5	1
fox	10.6	28	4	0
fox	8.6	18	4	1
fox	10.3	19	3	0

💡 Datei für von Flöhen auf Tieren

Du findest die Datei `flea_dog_cat_fox.xlsx` auf GitHub jkruppa.github.io/data/ als Excel oder auch als CSV.

7 Von Flöhen auf Tieren in Habitaten

Wir schauen uns in diesem Beispiel wiederum drei Tierarten an: Hunde, Katzen und Füchse. Auf diesen Tierarten messen wir die Sprunglänge von jeweils zehn Tieren. Im Vergleich zu dem vorherigen Beispiel erweitern wir die Daten um eine Spalte **site** in der wir vier verschiedene Messorte protokollieren. Es ergibt sich folgende Tabelle ?? und die dazugehörige Abbildung ??.

Tabelle 7.1: Sprunglängen [cm] für Hunde-, Katzen- und Fuchsflohe in verschiedenen Habitaten.

animal	site	rep	jump_length
cat	city	1	12.04
cat	city	2	11.98
cat	city	3	16.10
cat	city	4	13.42
cat	city	5	12.37
cat	city	6	16.36
cat	city	7	14.91
cat	city	8	11.17
cat	city	9	12.38
cat	city	10	15.06
cat	smalltown	1	15.24
cat	smalltown	2	13.36
cat	smalltown	3	15.08
cat	smalltown	4	12.83
cat	smalltown	5	14.68
cat	smalltown	6	10.73
cat	smalltown	7	13.35
cat	smalltown	8	14.54
cat	smalltown	9	12.99
cat	smalltown	10	14.51

animal	site	rep	jump_length
cat	village	1	17.59
cat	village	2	11.24
cat	village	3	12.44
cat	village	4	13.63
cat	village	5	14.92
cat	village	6	17.43
cat	village	7	18.30
cat	village	8	16.35
cat	village	9	16.34
cat	village	10	14.23
cat	field	1	13.70
cat	field	2	15.13
cat	field	3	17.99
cat	field	4	14.60
cat	field	5	16.16
cat	field	6	14.26
cat	field	7	15.39
cat	field	8	16.85
cat	field	9	19.02
cat	field	10	18.76
dog	city	1	19.35
dog	city	2	17.10
dog	city	3	19.85
dog	city	4	15.33
dog	city	5	15.15
dog	city	6	19.57
dog	city	7	15.44
dog	city	8	16.09
dog	city	9	15.91
dog	city	10	13.01
dog	smalltown	1	17.72
dog	smalltown	2	17.11
dog	smalltown	3	17.57
dog	smalltown	4	17.12
dog	smalltown	5	16.02
dog	smalltown	6	22.61
dog	smalltown	7	16.49
dog	smalltown	8	18.64
dog	smalltown	9	17.21

animal	site	rep	jump_length
dog	smalltown	10	19.90
dog	village	1	16.60
dog	village	2	15.28
dog	village	3	16.91
dog	village	4	15.08
dog	village	5	18.56
dog	village	6	16.34
dog	village	7	17.61
dog	village	8	14.80
dog	village	9	17.52
dog	village	10	16.93
dog	field	1	15.78
dog	field	2	17.02
dog	field	3	15.41
dog	field	4	15.61
dog	field	5	19.87
dog	field	6	19.24
dog	field	7	17.65
dog	field	8	18.83
dog	field	9	17.60
dog	field	10	14.67
fox	city	1	19.50
fox	city	2	18.49
fox	city	3	19.78
fox	city	4	19.45
fox	city	5	21.56
fox	city	6	21.37
fox	city	7	18.64
fox	city	8	20.08
fox	city	9	21.62
fox	city	10	20.68
fox	smalltown	1	19.81
fox	smalltown	2	17.78
fox	smalltown	3	19.65
fox	smalltown	4	16.38
fox	smalltown	5	17.46
fox	smalltown	6	17.02
fox	smalltown	7	19.38
fox	smalltown	8	15.89

animal	site	rep	jump_length
fox	smalltown	9	17.15
fox	smalltown	10	17.43
fox	village	1	15.32
fox	village	2	17.59
fox	village	3	15.70
fox	village	4	18.58
fox	village	5	16.85
fox	village	6	18.25
fox	village	7	18.75
fox	village	8	16.96
fox	village	9	13.38
fox	village	10	18.38
fox	field	1	16.85
fox	field	2	13.55
fox	field	3	13.89
fox	field	4	15.67
fox	field	5	16.38
fox	field	6	14.59
fox	field	7	14.03
fox	field	8	13.63
fox	field	9	14.09
fox	field	10	15.52

Die Datentabelle ist in dieser Form schon fast nicht mehr überschaubar. Daher hilft hier die explorative Datenanalyse weiter. Wir schauen uns daher die Daten einmal als einen Boxplot in Abbildung ?? an. Wir sehen hier, dass wir drei Tierarten an vier Orten die Sprungweite in [cm] gemessen haben.

Über die explorative Datenanalyse erfährst du mehr im Kapitel ??

💡 Datei für von Flöhen auf Tieren in Habitaten

Du findest die Datei `flea_dog_cat_fox_site.xlsx` auf GitHub jkruppa.github.io/data/ als Excel oder auch als CSV.

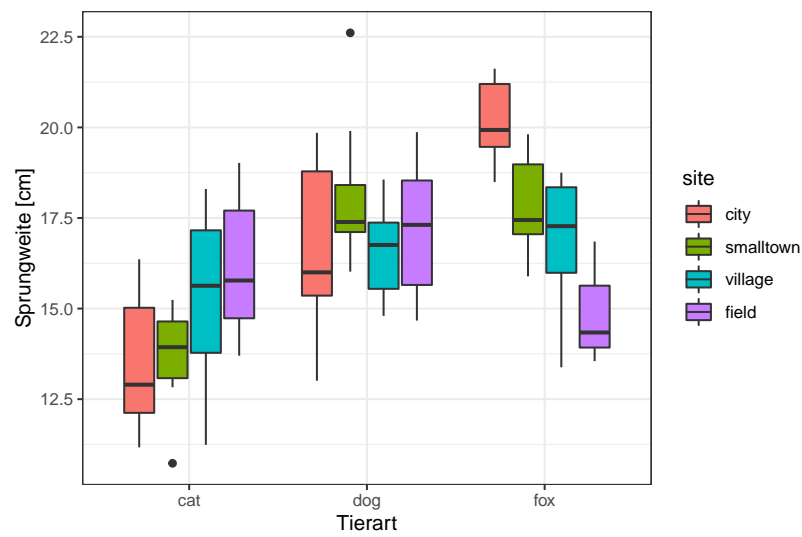


Abbildung 7.1: Boxplot der Sprungweiten [cm] für Hunde-, Katzen- und Fuchsflöhe in verschiedenen Habitaten.

8 Von vielen Flöhen auf Hunden und Katzen

Wir schauen uns in diesem Beispiel wiederum nur zwei Tierarten an: Hunde und Katzen. Auf diesen Tierarten messen wir wieder die Sprunglänge in [cm] von jeweils 400 Tieren. Im Vergleich zu dem vorherigen Beispiel erweitern wir die Daten um eine Spalte `jump_weight` in [mg] sowie `sex` [male, female]. Es ergibt sich folgende Tabelle ?? mit den ersten zehn Beobachtungen und die dazugehörige Abbildung ??.

Tabelle 8.1: Sprunglängen [cm], Gewichte [mg], Geschlecht [sex] für Hunde- und Katzenflöhe.

animal	sex	weight	jump_length
cat	male	6.02	16.93
cat	male	5.99	16.22
cat	male	8.05	18.96
cat	male	6.71	19.83
cat	male	6.19	17.37
cat	male	8.18	14.45
cat	male	7.46	15.46
cat	male	5.58	15.81
cat	male	6.19	19.14
cat	male	7.53	15.72

Die Datentabelle ist in dieser Form schon fast nicht mehr überschaubar. Daher hilft hier die explorative Datenanalyse weiter. Wir schauen uns daher die Daten einmal als einen Scatterplot in Abbildung ?? an. Wir sehen hier, dass wir das mit dem Gewicht [mg] der Flöhe auch die Sprungweite in [cm] steigt.

Über die explorative Datenanalyse erfährst du mehr im Kapitel ??

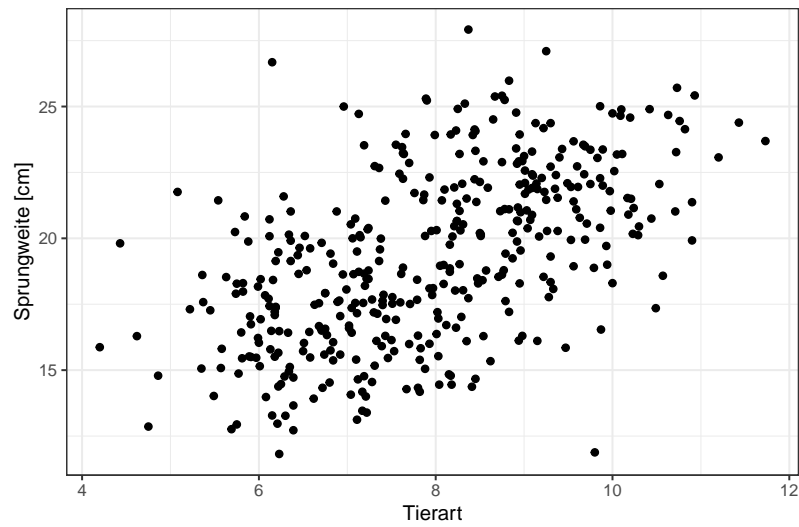


Abbildung 8.1: Scatterplot der Sprunglängen [cm] und Gewichte [mg] für Hunde- und Katzenflöhe.

💡 Datei für von vielen Flöhen auf Hunden und Katzen

Du findest die Datei `flea_dog_cat_length_weight.xlsx` auf GitHub jkruppa.github.io/data/ als Excel oder auch als CSV.

Teil II

Programmieren in R

Was solltest du nun zuerst Lesen? Es ist sehr schwierig die Programmierung exakt so zu schreiben, dass das Programmieren *linear* verständlich ist. Du brauchst im Prinzip das Wissen aus Kapitel ?? *Operatoren, Funktionen und Pakete* um die Grundlagen zu verstehen. Auf der anderen Seite fehlt dir vielleicht noch das Verständnis von Buchstaben und Zahlen in R. Diesen Zusammenhang zwischen Buchstaben und Zahlen erkläre ich als erstes im folgenden Kapitel ?? *Von Buchstaben und Zahlen*.

Im vorherigen Kapitel zu den Beispielen haben wir die Datentabelle Tabelle ?? mit den Hunde- und Katzenflöhen erschaffen. Bevor wir uns weiter mit statistischen Kennzahlen beschäftigen, wollen wir uns einmal die Realisierung der Tabelle Tabelle ?? mit den Hunde- und Katzenflöhen in R anschauen. Dabei wollen wir auch Eigenschaften von Zahlen und Buchstaben lernen, die notwendig sind um mit einem Programm wie R kommunizieren zu können. Wir wollen später R nutzen um die explorative Datenanalyse anzuwenden. Über die explorative Datenanalyse lernen wir in späteren Kapiteln mehr.

Einführung in R per Video

Du findest auf YouTube [Grundlagen in R](#) als Video Reihe. Ich werde zwar alles nochmal hier als Text aufschreiben, aber manchmal ist das Sehen und Hören dann einfacher.

9 Von Buchstaben und Zahlen

Im Kapitel ?? haben wir uns folgende Daten in Tabelle ?? angeschaut. Bevor wir uns weiter mit statistischen Kennzahlen beschäftigen, wollen wir uns einmal die Realisierung der Tabelle Tabelle ?? in R anschauen. Das heist, wie ist eine Tabelle in R aufgebaut und was sehen wir da eigentlich?

Tabelle 9.1: Sprunglängen [cm] für Hunde- und Katzenflöhe.

animal	jump_length	flea_count	grade	infected
dog	5.7	18	8	FALSE
dog	8.9	22	8	TRUE
dog	11.8	17	6	TRUE
dog	8.2	12	8	FALSE
dog	5.6	23	7	TRUE
dog	9.1	18	7	FALSE
dog	7.6	21	9	FALSE
cat	3.2	12	7	TRUE
cat	2.2	13	5	FALSE
cat	5.4	11	7	FALSE
cat	4.1	12	6	FALSE
cat	4.3	16	6	TRUE
cat	7.9	9	6	FALSE
cat	6.1	7	5	FALSE

Dabei wollen wir auch Eigenschaften von Zahlen und Buchstaben lernen, die notwendig sind um mit einem Programm wie R kommunizieren zu können. Nun haben wir Tabelle Tabelle ?? mit Daten zu verschiedenen Outcomes, wie Sprungweite [cm], Anzahl an Flöhen auf Hunden und Katzen, die Boniturnoten oder aber den Infektionsstatus. Die Tabelle Tabelle ?? ist zwar nicht groß aber auch nicht wirklich klein. Wir wollen uns nun damit beschäftigen, die Zahlen sinnvoll in R darzustellen. Wir

wollen mit der Darstellung einer Datentabelle in R beginnen, einem `tibble()`.

9.1 Daten in R sind `tibble()`

Im folgenden sehen wir die Datentabelle Tabelle ?? in R als `tibble` dargestellt. Was ist nun ein `tibble`? Ein `tibble` ist zu aller erst ein Speicher für Daten in R. Das heist wir haben Spalten und Zeilen. Jede Spalte repräsentiert eine Messung oder Variable und die Zeilen jeweils eine Beobachtung.

```
# A tibble: 14 x 5
  animal jump_length flea_count grade infected
  <chr>      <dbl>      <int> <dbl> <lgl>
1 dog         5.7         18     8 FALSE
2 dog         8.9         22     8  TRUE
3 dog        11.8         17     6  TRUE
4 dog         8.2         12     8 FALSE
5 dog         5.6         23     7  TRUE
6 dog         9.1         18     7 FALSE
7 dog         7.6         21     9 FALSE
8 cat         3.2         12     7  TRUE
9 cat         2.2         13     5 FALSE
10 cat         5.4         11     7 FALSE
11 cat         4.1         12     6 FALSE
12 cat         4.3         16     6  TRUE
13 cat         7.9          9     6 FALSE
14 cat         6.1          7     5 FALSE
```

Als erstes erfahren wir, dass wir einen `A tibble: 14 x 5` vorliegen haben. Das heist, wir haben 14 Zeile und 5 Spalten. In einem `tibble` wird immer in der ersten Zeile angezeigt wieviele Beobachtungen wir in dem Datensatz haben. Wenn das `tibble` zu groß wird, werden wir nicht mehr das ganze `tibble` sehen sondern nur noch einen Ausschnitt. Im Weiteren hat jede Spalte noch eine Eigenschaft unter dem Spaltennamen:

- `<chr>` bedeutet **character**. Wir haben also hier Worte vorliegen.

- `<dbl>` bedeutet **double**. Ein **double** ist eine Zahl mit Kommastellen.
- `<int>` bedeutet **integer**. Ein **integer** ist eine ganze Zahl ohne Kommastellen.
- `<lgl>` bedeutet **logical** oder **boolean**. Hier gibt es nur die Ausprägung *wahr* oder *falsch*. Somit **TRUE** oder **FALSE**. Statt den Worten **TRUE** oder **FALSE** kann hier auch 0 oder 1 stehen.
- `<str>` bedeutet **string** der aus verschiedenen **character** besteht kann, getrennt durch Leerzeichen.

💡 Zahlen, Buchstaben, Skalenniveau - Was ist das eigentlich?

Du findest auf YouTube [Einführung in R - Teil 06 - Zahlen, Buchstaben, Skalenniveau - Was ist das eigentlich?](#) als Video. Hier erkläre ich den Zusammenhang nochmal in einem Video.

9.2 Faktoren als Wörter zu Zahlen

Ein Computer und somit auch eine Programmiersprache wie R kann keine Buchstaben *verrechnen*. Ein Programm kann nur mit Zahlen rechnen. Wir haben aber in der Datentabelle Tabelle ?? in der Spalte **animal** Buchstaben stehen. Da wir hier einen Kompromiss eingehen müssen führen wir Faktoren ein. Ein Faktor kombiniert Buchstaben mit Zahlen. Wir als Anwender sehen die Buchstaben, die Wörter bilden. Intern steht aber jedes Wort für eine Zahl, so dass R mit den Zahlen rechnen kann. Klingt ein wenig kryptisch, aber wir schauen uns einen **factor** einmal an.

```
data_tbl$animal[1:8]
```

```
[1] "dog" "dog" "dog" "dog" "dog" "dog" "dog" "cat"
```

Was haben wir gemacht? Als erstes haben wir die Spalte **animal** aus dem Datensatz **data_tbl** mit dem Dollarzeichen **\$** *herausgezogen*. Mit dem **\$** Zeichen können wir uns eine einzelne Spalte

Ein **Faktor** ist eine Variable mit mehreren **Faktorstufen** oder **Leveln**. Für uns sieht der Faktor wie ein Wort aus, hinter jedem Wort steht aber eine Zahl mit der gerechnet werden kann.

aus dem Datensatz `data_tbl` rausziehen. Du kannst dir das `$` wie einen Kleiderbügel und das `data_tbl` als einen Schrank für Kleiderbügel vorstellen. An dem Kleiderbügel hängen dann die einzelnen Zahlen und Worte. Wir nehmen aber nicht den ganzen Vektor sondern nur die Zahlen 1 bis 8, dargestellt durch `[1:8]`. Die Gänsefüße `"` um `dog` zeigen uns, dass wir hier Wörter oder `character` vorliegen haben. Schauen wir auf das Ergebnis, so erhalten wir sieben Mal `dog` und einmal `cat`. Insgesamt die ersten acht Einträge der Datentabelle. Wir wollen diesen Vektor uns nun einmal als Faktor anschauen. Wir nutzen die Funktion `as_factor()`.

```
as.factor(data_tbl$animal[1:8])
```

```
[1] dog dog dog dog dog dog dog cat
Levels: cat dog
```

Im direkten Vergleich verschwinden die Gänsefüße `"` um `dog` und zeigen uns, dass wir hier keine `character` mehr vorliegen haben. Darüber hinaus sehen wir auch, dass die der Faktor jetzt `Levels` hat. Exakt zwei Stück. Jeweils einen für `dog` und einen für `cat`. Wir werden später Faktoren benötigen, wenn wir zum Beispiel eine einfaktorielle ANOVA rechnen. Hier siehst du schon den Begriff *Faktor* wieder.

9.3 Von Wörtern und Objekten

Das mag etwas verwirrend sein, denn es gibt in R Wörter `string <str>` oder `character <chr>`. Wörter sind was anderes als Objekte. Streng genommen sind beides Wörter, aber in Objekten werden Dinge gespeichert wohin gegen das Wort einfach ein Wort ist. Deshalb kennzeichnen wir Wörter auch mit Gänsefüßchen als `win "wort"` und zeigen damit, dass es sich hier um einen String handelt.

Wir tippen `"animal"` in R und erhalten `"animal"` als Wort zurück. Das sehen wir auch an dem Ausdruck mit den Gänsefüßchen.

Wir brauchen später zum Modellieren einen Datensatz, der *meist* aus einer **Outcome**-Spalte, einer **Faktor**-Spalte mit der *Behandlung* und einer **Faktor**-Spalte mit dem *Block* oder *Cluster* besteht.

Outcome ~ Behandlung + Block

```
"animal"
```

```
[1] "animal"
```

Wir tippen `animal` ohne die Anführungszeichen in R und erhalten den Inhalt von `animal` ausgegeben. Dafür müssen wir aber das Objekt `animal` erst einmal über den Zuweisungspfeil `<-` erschaffen.

```
animal <- c("dog", "cat", "fox")  
animal
```

```
[1] "dog" "cat" "fox"
```

Sollte es das Objekt `animal` nicht geben, also nicht über den Zuweisungspfeil `<-` erschaffen worden, dann wird eine Fehlermeldung von R ausgegeben:

```
Fehler in eval(expr, envir, enclos) : Objekt 'animal'  
nicht gefunden
```

9.4 Zusammenfassung

Tabelle ?? zeigt eine Übersicht wie einzelne Variablennamen und deren zugehörigen Beispielen sowie den Namen in R, der Informatik allgemein, als Skalenniveau und welcher Verteilungsfamilie die Variable angehören würde. Leider ist es so, dass wieder gleiche Dinge unterschiedliche benannt werden. Aber an dieses doppelte Benennen können wir uns in der Statistik schonmal gewöhnen.

Über den Zuweisungspfeil `<-` kannst du im Kapitel ?? mehr erfahren.

Variablennamen meint hier immer den **Namen der Spalte** im Datensatz bzw. **tibble**

Abbildung 9.2: Zusammenfassung und Übersicht von Variablennamen und deren Benennung in R, in der Informatik allgemein, als Skalenniveau und die dazugehörige Verteilungsfamilie.

Variable	Beispiel	R	Informatik	Skalenniveau	Verteilungsfamilie
weight	12.3, 12.4, 5.4, 21.3, 13.4	numeric	double	continuous	Gaussian
count	5, 0, 12, 23, 1, 4, 21	integer	integer	continuous	Poisson
dosis	low, mid, high	ordered		categorical / discrete / ordinal	Ordinal
field	mainz, berlin, kiel	factor		categorical / discrete	Multinomial
cancer	0, 1	factor		dichotomous / binary / nominal	Binomial
treatment	“placebo”, “aspirin”	character	character	dichotomous / binary / nominal	Binomial
birth	2001-12-02, 2005-05-23	date			

10 Operatoren, Funktionen und Pakete

Es ist immer schwierig, wann die Grundlagen von R einmal gelehrt werden sollte. Wenn du nichts von Programmierung bis jetzt gehört hast, dann mag es keinen Sinn ergeben mit Operatoren, wie dem Zuweisungspfeil `<-` und der Pipe `%>%` zu beginnen. Wir brauchen aber für die Programmierung folgende zentrale Konzepte.

- Wir müssen zusätzliche Pakete in R installieren und laden können (siehe Kapitel ??).
- Wir müssen verstehen wie wir uns einen Vektor mit `c()` bauen (siehe Kapitel ??).
- Wir müssen wissen was eine Funktion in R ist (siehe Kapitel ??).
- Wir müssen den Operator Zuweisungspfeil `<-` verstehen und anwenden können (siehe Kapitel ??).
- Wir müssen den Operator Pipe `%>%` verstehen und anwenden können (siehe Kapitel ??).
- Wir müssen den Operator `$` verstehen, da manche Funktionen in R nicht mit Datensätzen sondern nur mit Vektoren arbeiten können (siehe Kapitel ??).
- Wir müssen verstehen wie wir ein Modell in R mit der Tilde `~` definieren (siehe Kapitel ??).
- Wir müssen wissen und verstehen wie wir mit `?` englische Hilfeseiten öffnen können (siehe Kapitel ??).

Nicht alle Konzepte brauchst du *unmittelbar* aber ich nutze diese Konzepte wiederholt in allen Kapiteln, so dass du hier immer wieder mal schauen kannst, was die Grundlagen sind.

10.1 Pakete und library()

In der *Vanilla*-Variante hat R sehr wenige Funktionen. Ohne zusätzliche Pakete ist R mehr ein sehr potenter Taschenrechner. Leider mit der Funktionalität aus den 90'zigern, was die Programmierungsumgebung und die Funktionen angeht. Das wollen wir aber nicht. Wir wollen auf den aktuellen Stand der Technik und auch Sprache programmieren. Daher nutzen wir zusätzliche R Pakete.

Als Vanilla beschreibt man in der Informatikerwelt ein Programm, was keine zusätzlichen Pakete geladen hat. Also die reinst Form ohne zusätzlichen Geschmack.

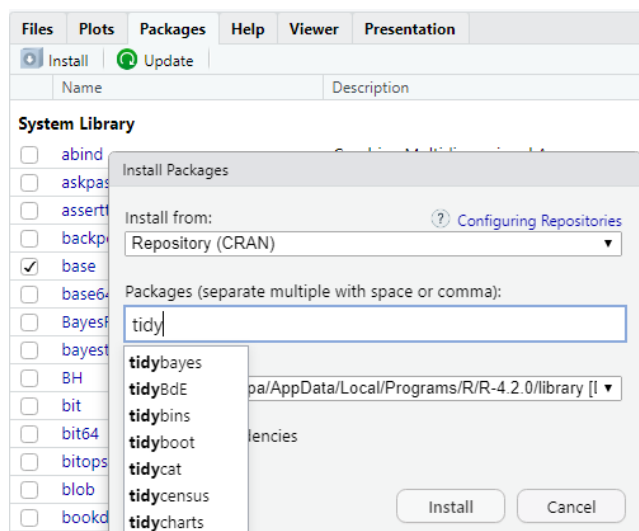


Abbildung 10.1: Auf den Reiter *Packages* klicken und dann *Install*. In der deutschen version vom RStudio mögen die Begriffe leicht anders sein.

In Abbildung ?? wird gezeigt wie du ein zusätzliches Paket installieren kannst. Hierbei ist nochmal wichtig den semantischen Unterschied zu wissen. Es gibt das Paket **tidyverse** was wir viel nutzen. Wir isnatllieren *einmalig* Pakete der Funktion `install.packages()` oder eben wie in Abbildung ?? gezeigt. Wir nutzen die Funktion `library()` um ein Paket in R zu laden. Ja, es müsste anders heissen, tut es aber nicht.

```
## Das Paket tidyverse installieren - einmalig  
install.packages(tidyverse)
```

```
## Das Paket tidyverse laden - jedes Mal  
library(tidyverse)
```

Nun muss man sich immer merken, ob das Paket schon installiert ist oder man schreibt relativ viele `library()` untereinander. Das passiert schnell, wenn du viele Pakete laden willst. Dafür erlaubt dir das Paket **pacman** eine Vereinfachung. Die Funktion `p_load()` installiert Pakete, wenn die Pakete nicht installiert sind. Sollten die Pakete installiert sein, so werden die Pakete geladen. Du musst nur einmal `install.packages(pacman)` ausführen um das Paket **pacman** zu installieren.

```
pacman::p_load(tidyverse, magrittr, readxl)
```

Schlussendlich gibt es noch die Möglichkeit sich alles nochmal bei YouTube anzuschauen.

💡 Unterschied von Packages und Libraries in R

Du findest auf YouTube [Einführung in R - Teil 03 - Unterschied Packages und Libraries in R](#) als Video. Hier erkläre ich nochmal den Ablauf zwischen Installieren eines Paketes und dem Laden eines Paketes.

10.2 Einen Vektor bauen `c()`

Wir können mit der Funktion `c()` Zahlen und Wörter zu einem Vektor kombinieren.

```
c("dog", "dog", "cat", "cat", "fox", "fox")
```

```
[1] "dog" "dog" "cat" "cat" "fox" "fox"
```

Hier werden die Wörter “dog”, “cat” und “fox” miteinander in einen Vektor kombiniert. Wir erinnern uns an das `$` Zeichen, was uns erlaubt eine Variable als Vektor aus einem `tibble()` herauszuziehen.

10.3 Funktionen

Wir haben schon einige Funktion nebenbei in R kennengelernt. Zum einen `as.factor()` um einen Faktor zu erstellen oder aus dem Kapitel ??, wo wir die Funktion `install.packages()` nutzen um ein Paket zu installieren oder aber die Funktion `library()` um ein Paket in R zu laden.

Funktionen sehen aus wie Wörter. Haben aber keine Gänsefüßchen und beinhalten auch keine Daten oder Vektoren. Funktionen können mit Daten und Vektoren rechnen und geben das Berechnete dann wieder. Nehmen wir als Beispiel die Funktion `mean()`, die den Mittelwert von einer Reihe Zahlen berechnet.

```
y <- c(1.2, 3.4, 2.1, 6, 4.3)
mean(y)
```

```
[1] 3.4
```

Wir sehen, dass wir mit der Funktion `c()` die Zahlen 1.2, 3.4, 2.1, 6, 4.3 zusammenkleben. Danach speichern wir die Zahlen in den Objekt `y` als einen Vektor ab. Wir müssen nicht erst erschaffen, das Erschaffen und Speichern passiert in R in einem Schritt. Wir stecken nun den Vektor `y` in die Funktion `mean()` und erhalten den Mittelwert von 3.4 der Zahlen wiedergegeben.

Eigentlich müssen in der Programmierung Objekte erst **deklariert** werden und somit erschaffen. Erst dann können Objekte **initialisiert** und somit befüllt bzw. etwas zugewiesen werden.

10.4 Zuweisungspfeil <-

Mit dem Zuweisungspfeil speichern wir *Dinge* in Objekte in R. Das heist wir speichern damit intern in R Datensätze und viele andere Sachen, die wir dan später wieder verwenden wollen. Schauen wir uns das einmal im Beispiel an. Schrieben wir nur den Vektor `c()` mit Hunden und Katzen darin, so erscheint eine Ausgabe in R.

```
c("dog", "dog", "cat", "cat", "fox", "fox")
```

```
[1] "dog" "dog" "cat" "cat" "fox" "fox"
```

Schreiben wir den gleichen Vektor und nutzen den Zuweisungspfeil, dann wird der Vektor in dem Objekt `animal` gespeichert.

```
animal <- c("dog", "dog", "cat", "cat", "fox", "fox")
```

Wie kommen wir jetzt an die Sachen, die in `animal` drin sind? Wir können einfach `animal` in R schreiben und dann wird uns der Inhalt von `animal` ausgegeben.

```
animal
```

```
[1] "dog" "dog" "cat" "cat" "fox" "fox"
```

Wir nutzen den Zuweisungspfeil `<-` ist zentral für die Nutzung von R. Wir brauchen den Zuweisungspfeil `<-` um Objekte in R zu erschaffen und Ergebnisse intern abzuspeichern. Zusammen mit Funktionen nutzen wir nur noch die Pipe `%>%` öfter.

Der Zuweisungspfeil `<-` ist zentral für die Nutzung von R.

10.5 Pipe `%>%`

Pipes in R

Du findest auf YouTube [Einführung in R - Teil 11 - Pipes in R](#) als Video. Hier erkläre ich den Zusammenhang nochmal in einem Video.

Im Weiteren nutzen wir den Pipe Operator dargestellt als `%>%`. Du kannst dir den Pipe Operator als eine Art Röhre vorstellen in dem die Daten verändert werden und dann an die nächste Funktion weitergeleitet werden. Im folgenden siehst du viele Funktionen, die aneinander über Objekte miteinander verbunden werden. Im Kapitel ?? erfährst du mehr über die Funktionen `select()` und `filter()`.

```
data_tbl <- read_excel("data/flea_dog_cat.xlsx")
animal_1_tbl <- select(data_tbl, animal, jump_length)
animal_2_tbl <- filter(animal_1_tbl, jump_length >= 4)
sort(animal_2_tbl$jump_length)
```

```
[1] 4.1 4.3 5.4 5.6 5.7 6.1 7.6 7.9 8.2 8.9 9.1 11.8
```

```
data_tbl %>%
  select(animal, jump_length) %>%
  filter(jump_length >= 4) %>%
  pull(jump_length) %>%
  sort
```

```
[1] 4.1 4.3 5.4 5.6 5.7 6.1 7.6 7.9 8.2 8.9 9.1 11.8
```

Im unteren Beispiel siehst du die Nutzung des Pipe Operators `%>%`. Das Ergebnis ist das gleiche, aber der Code ist einfacher zu lesen. Wir nehmen den Datensatz `data_tbl` leiten den Datensatz in den Funktion `select()` und wählen die Spalten `animal` sowie `jump_length`. Dann filtern wir noch nach `jump_length` größer als 4 cm. Dann ziehen wir uns mit der Funktion `pull()` die Spalte `jump_length` aus dem Datensatz. Den Vektor leiten wir dann weiter in die Funktion `sort()` und erhalten die sortierten Sprunglängen zurück.

10.6 Spalte extrahieren \$

Wir nutzen eigentlich die Funktion `pull()` um eine Spalte bzw. Vektor aus einem Datensatz zu extrahieren.

```
data_tbl %>%
  pull(animal)
```

```
[1] "dog" "dog" "dog" "dog" "dog" "dog" "dog" "cat" "cat" "cat" "cat" "cat"
[13] "cat" "cat"
```

Manche Funktionen in R, besonders die älteren Funktionen, benötigen keinen Datensatz sondern meist zwei bis drei Vektoren. Das heißt, wir können nicht einfach einen Datensatz in eine Funktion über `data = data_tbl` stecken sondern müssen der Funktion Vektoren übergeben. Dafür nutzen wir den `$` Operator.

```
data_tbl$animal
```

```
[1] "dog" "dog" "dog" "dog" "dog" "dog" "dog" "cat" "cat" "cat" "cat" "cat"
[13] "cat" "cat"
```

```
data_tbl$jump_length
```

```
[1] 5.7 8.9 11.8 8.2 5.6 9.1 7.6 3.2 2.2 5.4 4.1 4.3 7.9 6.1
```

Wir werden versuchen diese Schreibweise zu vermeiden, aber manchmal ist es sehr nützlich die Möglichkeit zu haben auf diese Weise eine Spalte zu extrahieren.

10.7 Modelle definieren mit formula

Wir müssen später Modelle in R definieren um zum Beispiel den t Test oder aber eine lineare Regression rechnen zu können. Wir nutzen dazu in R die `formula` Syntax. Das heißt links von der Tilde `~` steht das y , also der Spaltenname aus dem Datensatz `data` = den wir nutzen, der das Outcome repräsentiert. Rechts von der Tilde `~` stehen alle x_1, \dots, x_p , also alle Spalten aus dem Datensatz `data` = den wir nutzen, der die Einflussfaktoren repräsentiert.

In unserem Beispiel mit den Hunde- und Katzenflöhen aus Kapitel ?? wäre das y die Spalte `jump_length` und das x der Faktor `animal`. Wir erstellen mit der Funktion `formula()` das Modell in R. Wir brauchen später die Funktion `formula` nur implizit, aber hier ist es gut, das du einmal siehst, wie so eine Formula in R aussieht.

```
formula(jump_length ~ animal)
```

```
jump_length ~ animal
```

Wenn die Formel sehr lang wird bzw. wir die Namen der Spalten aus anderen Funktionen haben, können wir auch die Funktion `reformulate()` nutzen. Wir brauchen die Funktion aber eher im Bereich des maschinellen Lernens. Hier ist die Funktion `reformulate()` aufgeführt, da es inhaltlich passt.

```
reformulate(termlabels = c("animal", "sex", "site"),  
            response = "jump_length",  
            intercept = TRUE)
```

```
jump_length ~ animal + sex + site
```

10.8 Hilfe mit ?

Das Fragezeichen ? vor einem Funktionsnamen erlaubt die Hilfeseite zu öffnen. Die Hilfsseiten findest du auch in einem der Reiter im RStudio.

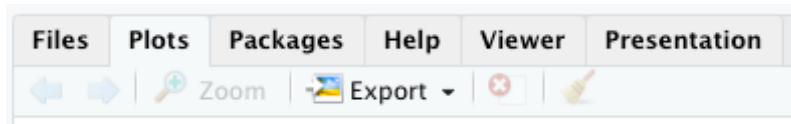


Abbildung 10.2: Neben den Paketen in R findet sich auch der Reiter Help, wo du Hilfe für die einzelnen Funktionen findest..

11 Daten einlesen

Die Daten aus unserem Experiment müssen rein in R. Das heißt, wir haben meist unsere Daten in einer Exceldatei vorliegen und wollen diese Daten nun in R einlesen.

Gängige Fehler beim Einlesen von Dateien in R sind folgende Probleme. Wir wollen diese Probleme nacheinander einmal durchgehen. Aber keine Sorge, das Einlesen von Daten in R ist immer am Anfang etwas frickelig. Du kannst gerne in das R Tutorium (siehe Kapitel ?? für Raum und Zeiten) kommen, dann können wir dir da beim Einlesen der Daten helfen.

- das Format der Daten ist nicht richtig (Kapitel ??)
- der Pfad zur Datei ist falsch (Kapitel ??)
- in der Datei sind komische Zeichen, wie Umlaute und Co. (Kapitel ??)
- in der Datei sind Leerzeichen in den Spaltennamen (Kapitel ??)

Im Anhang ?? findest du Beispiele für die Auswertung von Daten. Du kannst dir dort das Format anschauen und dann entsprechend deine Daten formatieren. Du findest auch alle Dateien auf GitHub unter jkruppa.github.io/data/ als Excel oder auch als CSV. Schau dir die Beispiele einmal an.

11.1 Genutzte R Pakete für das Kapitel

Wir wollen folgende R Pakete in diesem Kapitel nutzen.

```
pacman::p_load(tidyverse, magrittr, janitor)
```

Am Ende des Kapitels findest du nochmal den gesamten R Code in einem Rutsch zum selber durchführen oder aber kopieren.

11.2 Dateiformat

Wir unterscheiden bei Datenformaten zwischen dem Wide Format und dem Long Format. Meistens gibst du die Daten intuitiv

Das Buch *Cookbook for R* stellt auch Beispiele für die Funktion `gather()` zu Verfügung für die Umwandlung von Wide zu Long Format: [Converting data between wide and long format](#)

im Wide Format in Excel ein. Das ist in Excel auch übersichtlicher. R und später die Funktion `ggplot()` zur Visualisierung der Daten kann aber nur mit dem Long Format arbeiten. Wir können aber mit der Funktion `gather()` das Wide Format in das Long Format umwandeln.

11.2.1 Wide Format

In Tabelle ?? sehen wir eine typische Datentabelle in einem Wide Format. Die Spalten ergeben jeweils die Tierart wieder und die Einträge in den Spalten sind die Sprungweiten in [cm].

Tabelle 11.1: Eine Datentabelle mit Sprungweiten in [cm] von Hunde- und Katzenflöhen im Wide Format.

	dog	cat
	5.2	10.1
	4.9	9.4
	12.1	11.8
	8.2	6.7
	5.6	8.2
	9.1	9.1
	7.4	7.1

Wir können diese Datentabelle auch in R erstellen und uns als `tibble()` wiedergeben lassen.

```
jump_wide_tbl <- tibble(dog = c(5.2, 4.9, 12.1, 8.2, 5.6, 9.1, 7.4),
                        cat = c(10.1, 9.4, 11.8, 6.7, 8.2, 9.1, 7.1))
jump_wide_tbl
```

```
# A tibble: 7 x 2
  dog    cat
<dbl> <dbl>
1  5.2  10.1
2  4.9   9.4
3 12.1  11.8
4  8.2   6.7
```

5	5.6	8.2
6	9.1	9.1
7	7.4	7.1

Wir können aber mit einem Wide-Format nicht mit `ggplot()` die Daten aus der Tabelle ?? visualisieren. Deshalb müssen wir entweder das Wide Format in das Long Format umwandeln oder die Daten gleich in Excel im Long Format erstellen.

Wenn du schon Daten hast, dann macht es eventuell mehr Sinn eine **neue** Exceldatei anzulegen in der du dann die Daten in das Long Format kopierst.

11.2.2 Long Format

Wenn du Daten erstellst ist es wichtig, dass du die Daten in Excel im Long-Format erstellst. Dabei muss eine Beobachtung eine Zeile sein. Du siehst in Abbildung ?? ein Beispiel für eine Tabelle in Excel, die dem Long Format folgt.

	A	B	C	D	E
1	animal	jump_length	flea_count	grade	infected
2	dog	5,2	18	8	0
3	dog	4,9	22	7	1
4	dog	12,1	17	5	1
5	dog	8,2	12	6	0
6	dog	5,6	23	7	1
7	dog	9,1	18	7	0
8	dog	7,4	21	9	0
9	cat	3,2	12	9	1
10	cat	1,2	13	5	0
11	cat	6,6	11	7	0
12	cat	4,1	12	8	0
13	cat	4,3	16	6	1
14	cat	7,8	9	6	0
15	cat	6,2	7	8	0

Abbildung 11.1: Beispiel für eine Excel datentabelle in Long Format.

Im Folgenden sehen wir einmal wie die Funktion `gather()` das `tibble()` in Wide Format in ein `tibble()` in Long Format umwandelt. Wir müssen dafür noch die Spalte benennen mit

der Option `key =` in die die Namen der Spalten aus dem Wide Format geschrieben werden sowie den Spaltennamen für die eigentlichen Messwerte mit der Option `value =`.

```
jump_tbl <- tibble(dog = c(5.2, 4.9, 12.1, 8.2, 5.6, 9.1, 7.4),
                  cat = c(10.1, 9.4, 11.8, 6.7, 8.2, 9.1, 7.1)) %>%
  gather(key = "animal", value = "jump_length")
jump_tbl
```

```
# A tibble: 14 x 2
  animal jump_length
  <chr>      <dbl>
1 dog         5.2
2 dog         4.9
3 dog        12.1
4 dog         8.2
5 dog         5.6
6 dog         9.1
7 dog         7.4
8 cat        10.1
9 cat         9.4
10 cat        11.8
11 cat         6.7
12 cat         8.2
13 cat         9.1
14 cat         7.1
```

Wir sehen, dass ein Long Format viel mehr Platz benötigt. Das ist aber in R kein Problem. Wir sehen die Daten kaum sondern nutzen Funktionen wie `ggplot()` um die Daten zu visualisieren. Wichtig ist, dass du die Daten in Excel sauber abgelegt hast.

Im Folgenden schauen wir uns noch komplexere Daten in Tabelle ?? an. Das Datenbeispiel ist im Wide Format mit einem Behandlungsfaktor `treatment` einem Clusterfaktor `block` sowie mehreren Messwiederholungen zu unterschiedlichen Zeitpunkten `t_1` bis `t_6` angelegt.

Tabelle 11.2: Komplexeres Datenbeispiel im Wide Format mit einem Behandlungsfaktor **treatment** einem Clusterfaktor **block** sowie mehreren Messwiederholungen zu unterschiedlichen Zeitpunkten **t_1** bis **t_6**.

treatment	block	t_1	t_2	t_3	t_4	t_5	t_6
A	1	12.84	12.21	21.87	17.90	18.25	14.57
A	2	14.21	14.09	15.56	15.32	17.96	16.58
A	3	16.10	16.70	14.73	17.32	18.42	19.34
A	4	14.39	14.32	15.25	16.42	19.69	19.16
B	1	15.24	13.92	19.98	18.58	14.96	14.27
B	2	14.80	11.76	15.55	18.94	19.72	21.04
B	3	19.04	17.39	16.29	17.38	13.26	13.89
B	4	10.80	15.78	16.26	14.50	19.11	20.05
C	1	12.97	18.18	15.02	16.39	18.40	18.09
C	2	16.79	11.68	16.24	13.97	19.80	18.89
C	3	18.90	8.25	15.92	16.11	19.73	18.26
C	4	13.54	16.05	15.27	17.23	18.81	21.96
D	1	13.56	17.18	16.36	16.07	19.52	14.17
D	2	15.02	11.74	18.66	15.30	17.54	15.82
D	3	13.78	17.14	20.66	15.63	15.97	17.64
D	4	16.74	17.08	19.24	17.53	17.55	20.05

Im Folgenden Codeblock sehen wir, wie die Funktion `gather()` die Daten in Tabelle Tabelle ?? in ein Long Format umwandelt. Die Funktion fasst die Messwiederholungen der Spalten **t_1** bis **t_6** zusammen, in dem die Werte alle in der Spalte **drymatter** untereinander geklebt werden. Die Spalten **treatment** und **block** werden dann sechs Mal wiederholt untereinander geklebt.

```
data_tbl %>%
  gather(key = "time_point", value = "drymatter", t_1:t_6) %>%
  arrange(treatment, block)
```

```
# A tibble: 96 x 4
  treatment block time_point drymatter
<fct>      <int> <chr>      <dbl>
```

```

1 A          1 t_1          12.8
2 A          1 t_2          12.2
3 A          1 t_3          21.9
4 A          1 t_4          17.9
5 A          1 t_5          18.2
6 A          1 t_6          14.6
7 A          2 t_1          14.2
8 A          2 t_2          14.1
9 A          2 t_3          15.6
10 A         2 t_4          15.3
# ... with 86 more rows

```

11.3 Importieren mit RStudio

Wir können das RStudio nutzen um Daten mit Point-and-Klick rein zuladen und dann den Code wieder in den Editor kopieren. Im Prinzip ist dieser Weg der einfachste um einmal zu sehen, wie ein pfad funktioniert und der Code lautet. Später benötigt man diese ‘Krücke’ nicht mehr. Wir nutzen dann direkt den Pfad zu der Datei. Abbildung ?? zeigt einen Ausschnitt, wo wir im RStudio die *Import Dataset* Funktionalität finden.

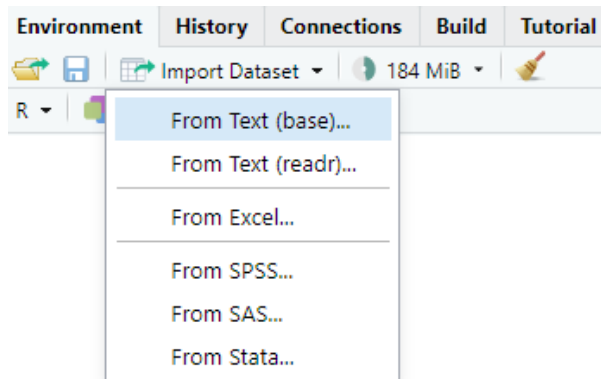


Abbildung 11.2: Auf den Reiter *Environment* klicken und dann *Import Dataset*. In der deutschen version vom RStudio mögen die Begriffe leicht anders sein.

💡 Importieren mit RStudio als Video

Du findest auf YouTube [Einführung in R - Teil 21.0 - Daten importieren mit RStudio - Point and Klick](#) als Video. Point and Klick ist als Video einfacher nachzuvollziehen als Screenshots in einem Fließtext.

11.4 Importieren per Pfad

In Abbildung ?? können wir sehen wie wir den Pfad zu unserer Excel Datei `flea_dog_cat.xlsx` finden. Natürlich kannst du den Pfad auch anders herausfinden bzw. aus dem Explorer oder Finder kopieren.

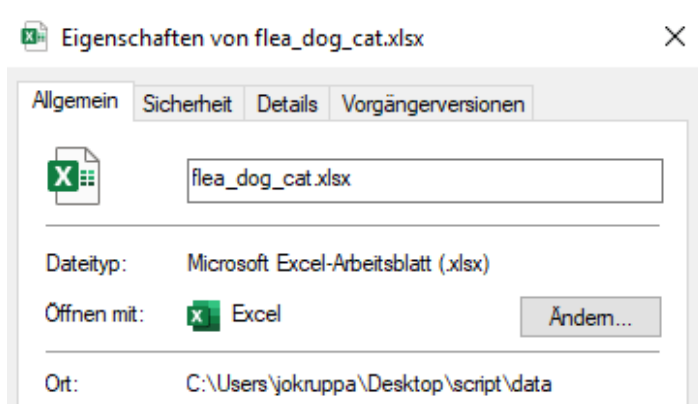


Abbildung 11.3: Durch den Rechts-Klick auf die Eigenschaften einer Datei kann man sich den Pfad zur Datei anzeigen lassen. **Achtung!** Unter Windows muss der Slash \ noch in den Backslash / gedreht werden.

Nachdem wir den Pfad gefunden haben, können wir den Pfad in die Funktion `read_excel()` kopieren und die Datei in das Objekt `data_tbl` einlesen. Ja, es wird nichts in der R Console ausgegeben, da sich die Daten jetzt in dem Object `data_tbl` befinden.

```
## Ganzer Pfad zur Datei flea_dog_cat.xlsx  
data_tbl <- read_excel("data/flea_dog_cat.xlsx")
```

! Unterschied zwischen \ in Windows und / in R

Achte einmal auf den Slash im Pfad in R und einem im Pfad in Windows. Einmal ist es der Slash \ im Dateipfad und einmal der Backslash /. Das ist sehr ärgerlich, aber dieses Problem geht zurück in die 80'ziger. Bill hat entschieden für sein Windows / zu nutzen und Steve (und Unix) eben /. Und mit dieser Entscheidung müssen wir jetzt leben...

11.5 Auf ein englisches Wort in Dateien

Ein großes Problem in Dateien sind Umlaute (ä,ö,ü) oder aber andere (Sonder)zeichen (ß, ?, oder #). Als dies sollte vermieden werden. Eine gute Datei für R beinhaltet nur *ganze* Wörter, Zahlen oder aber leere Felder. Ein leeres Feld ist ein fehlender Wert. Abbildung ?? zeigt eine gute Exceldateitabelle. Wir schreiben `jump_length` mit Unterstrich um den Namen besser zu lesen zu können. Sonst ist auch alles in Englisch geschrieben. Wir vermeiden durch die englische Schreibweise *aus versehen* einen Umlaut oder anderweitig problematische Zeichen zu verwenden. Später können wir alles noch für Abbildungen anpassen.

11.6 Spaltennamen in der (Excel)-Datei

Die Funktion `clean_names()` aus dem R Paket `janitor` erlaubt es die Spaltennamen einer eingelesenen Datei in eine für R gute Form zu bringen.

- Keine Leerzeichen in den Spaltennamen.
- Alle Spaltennamen sind klein geschrieben.

	A	B	C	D	E
1	animal	jump_length	flea_count	grade	infected
2	dog	5,2	18	8	0
3	dog	4,9	22	7	1
4	dog	12,1	17	5	1
5	dog	8,2	12	6	0
6	dog	5,6	23	7	1
7	dog	9,1	18	7	0
8	dog	7,4	21	9	0
9	cat	3,2	12	9	1
10	cat	1,2	13	5	0
11	cat	6,6	11	7	0
12	cat	4,1	12	8	0
13	cat	4,3	16	6	1
14	cat	7,8	9	6	0
15	cat	6,2	7	8	0

Abbildung 11.4: Beispiel für eine gute (Excel)Datentabelle.
Keine Umlaute sind vorhanden und die Spaltennamen haben keine Leerzeichen oder Sonderzeichen.


```
data_tbl %>%  
  clean_names()
```

```
# A tibble: 14 x 5  
  animal jump_length flea_count grade infected  
  <chr>      <dbl>      <dbl> <dbl>    <dbl>  
1 dog         5.7         18     8        0  
2 dog         8.9         22     8        1  
3 dog        11.8         17     6        1  
4 dog         8.2         12     8        0  
5 dog         5.6         23     7        1  
6 dog         9.1         18     7        0  
7 dog         7.6         21     9        0  
8 cat         3.2         12     7        1  
9 cat         2.2         13     5        0  
10 cat        5.4         11     7        0  
11 cat         4.1         12     6        0  
12 cat         4.3         16     6        1  
13 cat         7.9          9     6        0  
14 cat         6.1          7     5        0
```

12 Daten bearbeiten

Wir haben in dem vorherigen Kapitel Daten eingelesen. Jetzt wollen wir die Daten aufräumen (eng. *tidy*). Es ist notwendig, dass wir die Daten so aufarbeiten, dass R damit umgehen kann. Insbesondere das Erstellen von Faktoren ist wichtig, wenn die Spalte ein Faktor ist. R muss wissen was für Eigenschaften eine Spalte hat. Sonst funktionieren spätere Anwendungen in R nicht richtig oder geben einen Fehler wieder.

Es gibt zwei Möglichkeiten wie du mit deinen Daten umgehst:

- 1) Du änderst all deine Daten in Excel. Das mag bei einem kleinen Datensatz gut funktionieren. Dann musst du dich nicht mit dem *Programmieren* beschäftigen.
- 2) Du willst lernen die Daten auch in R zu verändern. Dann hilft dir dieses Kapitel. Auch in den folgenden Kapiteln werde ich immer wieder Funktionen wie `select()`, `filter()` und `mutate()` nutzen. Dann kannst du hier nochmal schauen, was die Funktionen machen.

Im Folgenden wollen wir den Datensatz `data_tbl` in R bearbeiten. Das heißt wir wollen Spalten auswählen mit `select()` oder Zeilen auswählen mit `filter()`. Schlussendlich wollen wir auch die Eigenschaften von Spalten mit der Funktion `mutate` ändern. Wir laden also den Datensatz `flea_dog_cat.xlsx` einmal in R.

```
data_tbl <- read_excel("data/flea_dog_cat_fox.xlsx")
```

Es ergibt sich folgende Tabelle ??, die wir schon aus vorherigen Kapiteln kennen.

Tabelle 12.1: Tabelle der Sprunglängen [cm], Anzahl an Flöhen, Boniturnote sowie der Infektionsstatus von Hunden, Katzen und Füchsen.

animal	jump_length	flea_count	grade	infected
dog	5.7	18	8	0
dog	8.9	22	8	1
dog	11.8	17	6	1
dog	8.2	12	8	0
dog	5.6	23	7	1
dog	9.1	18	7	0
dog	7.6	21	9	0
cat	3.2	12	7	1
cat	2.2	13	5	0
cat	5.4	11	7	0
cat	4.1	12	6	0
cat	4.3	16	6	1
cat	7.9	9	6	0
cat	6.1	7	5	0
fox	7.7	21	5	1
fox	8.1	25	4	1
fox	9.1	31	4	1
fox	9.7	12	5	1
fox	10.6	28	4	0
fox	8.6	18	4	1
fox	10.3	19	3	0

12.1 Genutzte R Pakete für das Kapitel

Wir wollen folgende R Pakete in diesem Kapitel nutzen.

```
pacman::p_load(tidyverse, readxl, magrittr, janitor)
```

Am Ende des Kapitels findest du nochmal den gesamten R Code in einem Rutsch zum selber durchführen oder aber kopieren.

12.2 Spalten wählen mit `select()`

💡 YouTube - Spalten auswählen mit `select()`

Du findest auf YouTube [Einführung in R - Teil 12 - Spalten auswählen mit `select\(\)`](#) als Video zum nochmal anschauen.

Der Datensatz, den wir im Experiment erschaffen, ist meist riesig. Jetzt könnten wir natürlich eine Exceltabelle mit unterschiedlichen Sheets bzw. Reitern erstellen oder aber die *Spalten* die wir brauchen in R selektieren. Wir nutzen die Funktion `select()` um Spalten zu wählen. Im folgenden Codeblock wählen wir die Spalten `animal`, `jump_length` und `flea_count`.

Wir nutzen die Funktion `select()` um Spalten zu wählen.

```
data_tbl %>%  
  select(animal, jump_length, flea_count)
```

```
# A tibble: 21 x 3  
  animal jump_length flea_count  
  <chr>      <dbl>      <dbl>  
1 dog         5.7         18  
2 dog         8.9         22  
3 dog        11.8         17  
4 dog         8.2         12  
5 dog         5.6         23  
6 dog         9.1         18  
7 dog         7.6         21  
8 cat         3.2         12  
9 cat         2.2         13  
10 cat        5.4         11  
# ... with 11 more rows
```


Wir können die Spalten beim selektieren auch umbenennen und in eine andere Reihenfolge bringen.

```
data_tbl %>%  
  select(Sprungweite = jump_length, flea_count, animal)
```

```
# A tibble: 21 x 3
  Sprungweite flea_count animal
    <dbl>      <dbl> <chr>
1      5.7        18 dog
2      8.9        22 dog
3     11.8        17 dog
4      8.2        12 dog
5      5.6        23 dog
6      9.1        18 dog
7      7.6        21 dog
8      3.2        12 cat
9      2.2        13 cat
10     5.4        11 cat
# ... with 11 more rows
```

Du findest auf der englischen [Hilfeseite für select\(\)](#) noch weitere Beispiele für die Nutzung.

12.3 Zeilen wählen mit `filter()`

 YouTube - Zeilen auswählen mit `filter()`

Du findest auf YouTube [Einführung in R - Teil 13 - Zeilen auswählen mit filter\(\)](#) als Video zum nochmal anschauen.

Während wir die Auswahl an Spalten gut und gerne auch in Excel durchführen können, so ist dies bei der Auswahl der Zeilen nicht so einfach. Wir können in R hier auf die Funktion `filter()` zurückgreifen. Wir nutzen die Funktion `filter()` um Zeilen nach Kriterien zu wählen.

Im folgenden Codeblock wählen wir die Zeilen aus in denen die Worte `dog` und `fox` stehen. Wir nutzen dazu den Operator `%in%` um auszudrücken, dass wir alle Einträge in der Spalte `animal` wollen die in dem Vektor `c("dog", "fox")` beschrieben sind.

```
data_tbl %>%
  filter(animal %in% c("dog", "fox"))
```

Wir nutzen die Funktion `filter()` um Zeilen nach Kriterien zu wählen.

```
# A tibble: 14 x 5
  animal jump_length flea_count grade infected
  <chr>      <dbl>      <dbl> <dbl>    <dbl>
1 dog         5.7         18      8         0
2 dog         8.9         22      8         1
3 dog        11.8         17      6         1
4 dog         8.2         12      8         0
5 dog         5.6         23      7         1
6 dog         9.1         18      7         0
7 dog         7.6         21      9         0
8 fox         7.7         21      5         1
9 fox         8.1         25      4         1
10 fox         9.1         31      4         1
11 fox         9.7         12      5         1
12 fox        10.6         28      4         0
13 fox         8.6         18      4         1
14 fox        10.3         19      3         0
```

Es stehen dir Folgende logische Operatoren zu Verfügung wie in Tabelle ?? gezeigt. Am Anfang ist es immer etwas schwer sich in den logischen Operatoren zurechtzufinden. Daher kann ich dir nur den Tipp geben einmal die Operatoren selber auszuprobieren und zu schauen, was du da so rausfilterst.

Tabelle 12.2: Logische Operatoren und R und deren Beschreibung

Logischer Operator	Beschreibung
<	kleiner als (eng. <i>less than</i>)
<=	kleiner als oder gleich (eng. <i>less than or equal to</i>)
>	größer als (eng. <i>greater than</i>)
>=	größer als oder gleich (eng. <i>greater than or equal to</i>)
==	exakt gleich (eng. <i>exactly equal to</i>)
!=	nicht gleich (eng. <i>not equal to</i>)
!x	nicht (eng. <i>not x</i>)
x y	oder (eng. <i>x or y</i>)
x & y	und (eng. <i>x and y</i>)

Hier ein paar Beispiele. Probiere gerne auch mal Operatoren selber aus. Im folgenden Codeblock wollen wir nur die Zeilen haben, die eine Anzahl an Flöhen größer von 15 haben.

```
data_tbl %>%  
  filter(flea_count > 15)
```

```
# A tibble: 13 x 5  
  animal jump_length flea_count grade infected  
  <chr>      <dbl>      <dbl> <dbl>    <dbl>  
1 dog         5.7         18     8        0  
2 dog         8.9         22     8        1  
3 dog        11.8         17     6        1  
4 dog         5.6         23     7        1  
5 dog         9.1         18     7        0  
6 dog         7.6         21     9        0  
7 cat         4.3         16     6        1  
8 fox         7.7         21     5        1  
9 fox         8.1         25     4        1  
10 fox        9.1         31     4        1  
11 fox        10.6         28     4        0  
12 fox         8.6         18     4        1  
13 fox        10.3         19     3        0
```

Wir wollen nur die infizierten Tiere haben.

```
data_tbl %>%  
  filter(infected == TRUE)
```

```
# A tibble: 10 x 5  
  animal jump_length flea_count grade infected  
  <chr>      <dbl>      <dbl> <dbl>    <dbl>  
1 dog         8.9         22     8        1  
2 dog        11.8         17     6        1  
3 dog         5.6         23     7        1  
4 cat         3.2         12     7        1  
5 cat         4.3         16     6        1  
6 fox         7.7         21     5        1  
7 fox         8.1         25     4        1
```

8 fox	9.1	31	4	1
9 fox	9.7	12	5	1
10 fox	8.6	18	4	1

Wir wollen nur die infizierten Tiere haben UND die Tiere mit einer Flohanzahl größer als 20.

```
data_tbl %>%
  filter(
    infected == TRUE & flea_count > 20
  )
```

```
# A tibble: 5 x 5
  animal jump_length flea_count grade infected
  <chr>      <dbl>      <dbl> <dbl>    <dbl>
1 dog         8.9         22      8         1
2 dog         5.6         23      7         1
3 fox         7.7         21      5         1
4 fox         8.1         25      4         1
5 fox         9.1         31      4         1
```

Du findest auf der englischen [Hilfeseite für filter\(\)](#) noch weitere Beispiele für die Nutzung.

12.4 Spalten ändern mit mutate()

 YouTube - Eigenschaften von Variablen ändern mit mutate()

Du findest auf YouTube [Einführung in R - Teil 14 - Eigenschaften von Variablen ändern mit mutate\(\)](#) als Video zum nochmal anschauen.

Nachdem wir die Spalten mit `select()` und eventuell die Zeilen mit `filter()` gewählt haben, müssen wir jetzt noch die Eigenschaften der Spalten ändern. Das Ändern müssen wir nicht immer tun, aber häufig müssen wir noch einen Faktor erschaffen. Wir nutzen noch die Funktion `pull()` um uns die Spalte

Wir nutzen die Funktion `mutate()` um die Eigenschaften von Spalten daher Variablen zu ändern.

Die Reihenfolge der Funktionen ist wichtig um unliebsame Effekte zu vermeiden.

- 1) Erst wählen wir die Spalten mit `select()`
- 2) Dann filtern wir die Zeilen mit `filter()`
- 3) Abschließend ändern wir die Eigenschaften der Spalten mit `mutate()`

`animal` aus dem Datensatz zu ziehen. Nur so sehen wir die vollen Eigenschaften des Faktors. Später nutzen wir `pull` seltener und nur um zu kontrollieren, was wir gemacht haben.

Im folgenden Codeblock verwandeln wir die Variable `animal` in einen Faktor durch die Funktion `as_factor`. Wir sehen, dass die Level des Faktors so sortiert sind, wie das Auftreten in der Spalte `animal`.

```
data_tbl %>%  
  mutate(animal = as_factor(animal)) %>%  
  pull(animal)
```

```
[1] dog dog dog dog dog dog dog cat cat cat cat cat cat cat cat fox fox fox fox fox  
[20] fox fox  
Levels: dog cat fox
```

Wollen wir die Sortierung der Level ändern, können wir die Funktion `factor()` nutzen. Wir ändern die Sortierung des Faktors zu `fox`, `dog` und `cat`.

```
data_tbl %>%  
  mutate(animal = factor(animal, levels = c("fox", "dog", "cat"))) %>%  
  pull(animal)
```

```
[1] dog dog dog dog dog dog dog cat cat cat cat cat cat cat cat fox fox fox fox fox  
[20] fox fox  
Levels: fox dog cat
```

Wir können auch die Namen (eng. *labels*) der Level ändern. Hier musst du nur aufpassen wie du die alten Labels überschreibst. Wenn ich *gleichzeitig* die Level und die Labels ändere komme ich häufig durcheinander. Da muss du eventuell nochmal schauen, ob auch alles so geklappt hat wie du wolltest.

```
data_tbl %>%  
  mutate(animal = factor(animal, labels = c("Hund", "Katze", "Fuchs"))) %>%  
  pull(animal)
```

```
[1] Katze Katze Katze Katze Katze Katze Katze Hund Hund Hund Hund Hund
[13] Hund Hund Fuchs Fuchs Fuchs Fuchs Fuchs Fuchs Fuchs
Levels: Hund Katze Fuchs
```

Du findest auf der englischen [Hilfeseite für mutate\(\)](#) noch weitere Beispiele für die Nutzung. Insbesondere die Nutzung von `mutate()` über mehrere Spalten gleichzeitig erlaubt sehr effizientes Programmieren. Aber das ist für den Anfang etwas viel.

i Die Funktionen `select()`, `filter()` und `mutate()` in R

Bitte schaue dir auch die Hilfeseiten der Funktionen an. In diesem Skript kann ich nicht alle Funktionalitäten der Funktionen zeigen. Oder du kommst in das R Tutorium welches ich anbiete und fragst dort nach den Möglichkeiten Daten in R zu verändern.

12.5 Gruppieren mit `group_by()`

Sobald wir einen Faktor erschaffen haben, können wir die Daten in R auch nach dem Faktor *gruppieren*. Das heißt wir nutzen die Funktion `group_by()` um R mitzuteilen, dass nun folgende Funktionen *getrennt* für die einzelnen Gruppen erfolgen sollen. Im folgenden Codeblock siehst du die Anwendung.

```
data_tbl %>%
  mutate(animal = as_factor(animal)) %>%
  group_by(animal)
```

```
# A tibble: 21 x 5
# Groups:   animal [3]
  animal jump_length flea_count grade infected
  <fct>      <dbl>      <dbl> <dbl>    <dbl>
1 dog         5.7         18     8         0
2 dog         8.9         22     8         1
3 dog        11.8         17     6         1
4 dog         8.2         12     8         0
5 dog         5.6         23     7         1
```

```

6 dog          9.1          18      7      0
7 dog          7.6          21      9      0
8 cat          3.2          12      7      1
9 cat          2.2          13      5      0
10 cat         5.4          11      7      0
# ... with 11 more rows

```

Auf den ersten Blick ändert sich nicht viel. Es entsteht aber die Zeile `# Groups: animal [3]`. Wir wissen nun, dass wir nach der Variable `animal` mit drei Gruppen die Datentabelle gruppiert haben. Die Anwendung siehst du in Kapitel ?? bei der Berechnung von deskriptiven Maßzahlen.

12.6 Mehr Informationen durch `glimpse()` und `str()`

Am Ende noch zwei Funktionen zur Kontrolle, was wir hier eigentlich gerade tun. Mit der Funktion `glimpse()` können wir uns einen Einblick in die Daten geben lassen. Wir sehen dann nochmal kurz und knapp wieviel Zeilen und Spalten wir haben und welche Inhalte in den Spalten stehen. Die gleichen Informationen erhalten wir auch durch die Funktion `str()`. Die Funktion `str()` geht aber noch einen Schritt weiter und nennt uns auch Informationen zu dem Objekt. Daher wir wissen jetzt, dass es sich beim dem Objekt `data_tbl` um ein `tibble()` handelt.

```
glimpse(data_tbl)
```

```

Rows: 21
Columns: 5
$ animal      <chr> "dog", "dog", "dog", "dog", "dog", "dog", "dog", "cat", "c~
$ jump_length <dbl> 5.7, 8.9, 11.8, 8.2, 5.6, 9.1, 7.6, 3.2, 2.2, 5.4, 4.1, 4.~
$ flea_count  <dbl> 18, 22, 17, 12, 23, 18, 21, 12, 13, 11, 12, 16, 9, 7, 21, ~
$ grade       <dbl> 8, 8, 6, 8, 7, 7, 9, 7, 5, 7, 6, 6, 6, 5, 5, 4, 4, 5, 4, 4~
$ infected    <dbl> 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1~

```

```
str(data_tbl)
```

```
tibble [21 x 5] (S3: tbl_df/tbl/data.frame)
 $ animal      : chr [1:21] "dog" "dog" "dog" "dog" ...
 $ jump_length: num [1:21] 5.7 8.9 11.8 8.2 5.6 9.1 7.6 3.2 2.2 5.4 ...
 $ flea_count  : num [1:21] 18 22 17 12 23 18 21 12 13 11 ...
 $ grade       : num [1:21] 8 8 6 8 7 7 9 7 5 7 ...
 $ infected    : num [1:21] 0 1 1 0 1 0 0 1 0 0 ...
```

Teil III

Explorative Datenanalyse

Wir haben die Daten jetzt in R Eingelesen und im Zweifel noch angepasst. Nun wollen wir uns die Daten einmal angucken. Nicht in dem Sinne, dass wir auf die *Datentabelle* schauen. Sondern wir wollen die Daten visualisieren. Wir erstellen Abbildungen von den Daten und versuchen so mehr über die Daten zu erfahren. Sehen wir Zusammenhänge zwischen verschiedenen Variablen bzw. Spalten? Wir führen eine explorative Datenanalyse durch. Über die explorative Datenanalyse wollen wir uns in diesem Kapitel einmal Gedanken machen.

Wir kürzen die explorative Datenanalyse häufig als **EDA** ab.

Einführung in R per Video

Du findest auf YouTube [Grundlagen in R](#) als Video Reihe. Du musst die Grundlagen in R verstanden haben, damit du dem R Code folgen kannst.

13 Deskriptive Statistik

Wir nutzen die deskriptive Statistik um Zahlen zusammenzufassen. Das heißt wir haben einen Datensatz vorliegen wie den Datensatz `flea_dog_cat.xlsx`. Wir wollen jetzt den großen Datensatz in wenige Zahlen wiedergeben. Warum wenige Zahlen? Wenn wir das Ergebnis *präsentieren* wollen, dann müssen es wenige Zahlen sein, die den Datensatz gut zusammenfassen. Daher ist es wichtig zu wissen, dass wir *dutzende bis hunderte Zahlen* durch meist eine oder zwei Zahlen beschreiben wollen. Wir brauchen die statistischen Maßzahlen aus diesem Kapitel später um teilweise noch extrem größere Datensätze darstellen zu können. Ebenso werden wir die Maßzahlen aus diesem Kapitel dafür verwenden statistische Tests und Modelle zu rechnen.

Nehmen wir nun als Beispiel die Sprungweiten in [cm] von Hundeflöhen. Wir messen sieben Sprungweiten von sieben Hundeflöhen und messen dabei folgende Werte in [cm]: 5.7, 8.9, 11.8, 8.2, 5.6, 9.1 und 7.6. Wir schreiben nun y als einen Vektor in der Form

$$y = \{5.7, 8.9, 11.8, 8.2, 5.6, 9.1, 7.6\}.$$

In R würde der Vektor wie etwas anders aussehen.

```
y <- c(5.7, 8.9, 11.8, 8.2, 5.6, 9.1, 7.6)
```

Wir wollen nun die Zahlen in y beschreiben und durch wenige andere Zahlen zusammenfassen. Einige der statistischen Maßzahlen sind dir vermutlich schon bekannt, andere eher neu.

Eigentlich *schätzen* wir die **Parameter einer Verteilung**. Aber das kommt nochmal später genauer, wenn wir wissen was Verteilungen sind.

i Parametrik versus Nicht-Parametrik

Wenn wir einen Zahlenvektor wie durch $y = \{5.7, 8.9, 11.8, 8.2, 5.6, 9.1, 7.6\}$ beschrieben zusammenfassen wollen, haben wir zwei Möglichkeiten.

- 1) Die **parametrische Variante** indem wir *mit den Zahlen* rechnen und deskriptive Maßzahlen wie Mittelwert, Varianz und Standardabweichung berechnen. Diese Maßzahlen kommen aber in den Zahlen nicht vor.
- 2) Die **nicht-parametrische Variante** indem wir die Zahlen in Ränge umwandeln, also sortieren, und *mit den Rängen* der Zahlen rechnen. Die deskriptiven Maßzahlen wären dann Median, Quantile und Quartile.

13.1 Mittelwert

Der Mittelwert einer Zahlenreihe beschreibt den Schwerpunkt der Zahlen. Der Mittelwert wird auch als Lageparameter benannt. Wir schreiben den Mittelwert mit einem Strich über den Vektor, der die Zahlen enthält. Im folgenden ist die Formel für den Mittelwert der Sprungweite in [cm] der Hunde gezeigt. Der Mittelwert ist in dem Sinne eine künstliche Zahl, da der Mittelwert häufig nicht in den beobachteten Zahlen vorkommt.

Wir werden immer mal wieder **Formeln vereinfachen**. Zum Beispiel nur \sum schreiben anstatt \sum_i^n , wenn wir einen Vektor aufsummieren und uns die Indizes sparen...

$$\bar{y} = \sum_{i=1}^n \frac{x_i}{n} = \frac{5.7 + 8.9 + 11.8 + 8.2 + 5.6 + 9.1 + 7.6}{7} = 8.13$$

Im Durchschnitt oder im Mittel springen Hundeflöhe 8.13 cm weit.



Der **Mittelwert** und der **Median** sind zwei Lageparameter einer Verteilung. Beide beschreiben die Stelle, wo die Verteilungskurve am dichtesten ist.

In der Abbildung ?? wollen wir die Formel nochmal visualisieren. Vielleicht fällt dir dann der Zusammenhang von dem Index i und der gesamten Fallzahl n leichter.

$$\bar{y} = \sum_{i=1}^n \frac{y_i}{n}$$

y_i	5,7	8,9	11,8	8,2	5,6	9,1	7,6
i	1	2	3	4	5	6	7

n=7

Abbildung 13.1: Zusammenhang zwischen y sowie dem Index i in der Formel für den Mittelwert.

In R können wir den Mittelwert einfach mit der Funktion `mean()` berechnen. Wir wollen dann den Mittelwert noch auf die zweite Kommastelle runden. Das machen wir dann mit der Funktion `round()`.

```
y %>% mean %>% round(2)
```

```
[1] 8.13
```

Wir erhalten das gleiche Ergebnis wie oben in unserer händischen Rechnung. Die Hundeflöhe springen im Mittel 8.13 cm weit.

Der Mittelwert ist eine bedeutende Maßzahl der Normalverteilung. Daher merken wir uns hier schon mal, dass wir den Mittelwert brauchen werden. Auch wenn wir darüber nachdenken ob sich zwei Gruppen unterscheiden, so nutzen wir hierzu den Mittelwert. Unterscheiden sich die *mittleren* Sprungweiten in [cm] von Hunde- und Katzenflöhen?

13.2 Spannweite

Die *Spannweite* erlaubt uns zu überprüfen was die kleinste Zahl und die größte Zahl ist. Also uns das Minimum und das Maximum einer Zahlenreihe anzuschauen. Auf den ersten Blick mag das nicht so sinnig sein, aber wenn wir uns hunderte von Beobachtungen anschauen, wollen wir wissen, ob wir nicht einen Fehler bei Eintragen der Daten gemacht haben. Wir wissen eigentlich, dass z.B keine negativen Zuwachsraten auftreten können.

$$y_{range} = y_{max} - y_{min} = 12.1 - 4.9 = 7.2$$

Die Hundeflöhe springen in einer Spannweite von 7.2 cm. Das kommt einem *normal* vor. Die Spannweite ist nicht übertrieben groß. Der minimale Wert ist 4.9 und der maximale Wert ist 12.1 und somit sind beide Zahlen in Ordnung. Keine der beiden Zahlen ist übertrieben groß oder gar negativ.

In R können wir die Spannweite mit `range()` wie folgt berechnen. Wir erhalten den minimalen und maximalen Wert.

```
range(y)
```

```
[1]  5.6 11.8
```

Wir merken uns, dass die Spannweite eine Maßzahl für die Validität der Daten ist. Hat das Experiment geklappt oder kamen da nur komische Zahlen bei raus, die wir so in der Realität nicht erwarten würden. Zum Beispiel negative Sprungweiten, weil wir einmal auf das Minuszeichen gekommen sind.

13.3 Varianz

Bis jetzt können wir mit dem Mittelwert \bar{y} die Lage oder den Mittelpunkt unserer Zahlenreihe beschreiben. Uns fehlt damit aber die Information über die Streuung der Zahlen. Sind die Zahlen alle eher gleich oder sehr verschieden? Liegen die Zahlen

Die Spannweite dient dazu in einem Datensatz zu überprüfen ob die **Spalte**, oder auch **Variable** genannt, den richtigen Zahlenraum aufweist. Das machen wir durch die Funktion `range()`.

daher alle bei dem Mittelwert oder sind die Zahlen weit um den Mittelwert gestreut.

Die Streuung der Zahlen um den Mittelwert beschreibt die Varianz oder auch s^2 . Wir berechnen die Varianz indem wir von jeder Zahl den Mittelwert aller Zahlen abziehen und dann das Ergebnis quadrieren. Das machen wir für alle Zahlen und addieren dann die Summe auf. Wir erhalten die *Quadratsumme* von y .

$$s^2 = \sum_{i=1}^n \frac{(y_i - \bar{y})^2}{n-1} = \frac{(5.7 - 8.13)^2 + \dots + (7.6 - 8.13)^2}{7-1} = 4.6$$

Die Varianz beschreibt also die Streuung der Zahlen *im Quadrat* um den Mittelwert. Das heißt in unserem Beispiel, dass die Sprungweite eine Varianz von 4.6 cm² hat. Wir können Quadratzentimeter schlecht interpretieren. Deshalb führen wir gleich die Wurzel der Varianz ein: die Standardabweichung.

In R lässt sich die Varianz einfach durch die Funktion `var()` berechnen.

```
y %>% var %>% round(2)
```

```
[1] 4.6
```

Wir benötigen die Varianz häufig nur als Zwischenschritt um die Standardabweichung zu berechnen. Das Konzept der Abweichungsquadrate benötigen wir aber in der Varianzanalyse (ANOVA) und für die Beschreibung einer Normalverteilung.

13.4 Standardabweichung

Die *Standardabweichung* ist die Wurzel der Varianz. Wo die Varianz die Abweichung der Sprungweite in [cm²] beschreibt, beschreibt die Standardabweichung die Streuung der Sprungweite in [cm].

Abweichungsquadrate sind ein wichtiges Konzept in der Statistik. Wenn wir wissen wollen, wie groß eine Abweichung von einer Zahl zu einer anderen ist, dann nutzen wir immer das Quadrat der Abweichung und bilden die Quadratsumme..

$$s = \sqrt{s^2} = \sqrt{4.6} = 2.14$$

Wir können also schreiben, dass die Flöhe im Mittel $8.13 \pm 2.14\text{cm}$ weit springen. Somit haben wir die Lage und die Streuung der Zahlenreihe y der Sprungweite in [cm] mit zwei Zahlen beschrieben.

In R können wir die Standardabweichung einfach mit der Funktion `sd()` berechnen.

```
y %>% sd %>% round(2)
```

```
[1] 2.14
```

Wir schreiben immer den Mittelwert plusminus die Standardabweichung. Also immer $\bar{y} \pm s$.

13.5 Mittelwert und Varianz - eine Herleitung

Was ist der Mittelwert und die Varianz genau? Schauen wir uns das einmal in Abbildung ?? an. Die graue Linie oder Gerade beschreibt den Mittelwert der fünf Beobachtungen. Die fünf Beobachtungen sind als blaue Punkt dargestellt. Auf der x-Achse ist nur der Index des Punktes. Das heißt y_1 ist der erste Punkte, das der Index i gleich 1 ist.

Wenn wir die Summe der Abweichungen von y_1 bis y_5 zu dem Mittelwert bilden, so wird diese Summe 0 sein. Der Mittelwert liegt genau in der Mitte der Punkte. In unserem Beispiel ist der Mittelwert $\bar{y} = 5.8$. Wir können jetzt die Abstände wie in der folgenden Tabelle berechnen.

Wir nennen die Abstände $y_i - \bar{y}$ nach dem griechischen Buchstaben Epsilon ϵ . Das ϵ soll an das e von *Error* erinnern. So meint dann *Error* eben auch Abweichung. Ja, es gibt hier viele Worte für das gleiche Konzept.

Wir berechnen einen Mittelwert von den Epsilons mit $\bar{\epsilon} = 0$. Ein Mittelwert nahe Null bzw. von Null wundert uns nicht. Wir haben die Gerade ja so gebaut, das nach oben und unten

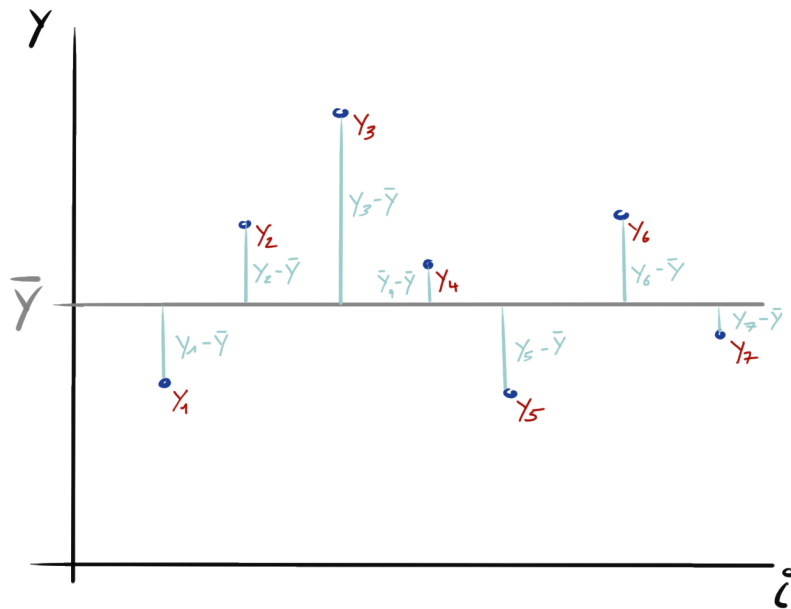


Abbildung 13.2: Die graue Linie beschreibt den Mittelwert der genau so durch die blauen Punkte geht, dass die Abstände der Punkte oberhalb und unterhalb zu Null aufaddieren. Die Linie liegt in der Mitte der Punkte. Die quadrierten Abstände sind die Varianz der blauen Punkte. Auf der x-Achse ist der Index des Punktes eingetragen.