technische universität dortmund

Fakultät Rehabilitationswissenschaften

Methoden der empirischen Bildungsforschung

Eine praktische Einführung in die Statistik mit R und RStudio

-

Ein Lehrskript für Studierende der Rehabilitationspädagogik,
Rehabilitationswissenschaften sowie
(Sonder-)Pädagogik

Christian Kißler, Florian Krieger & Jörg-Tobias Kuhn

1. Auflage
(Version 1.0)

Dortmund: Eldorado.

2023

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Publikation: 2023, Dortmund, Eldorado

Impressum und Kontakt:

Technische Universität Dortmund

Fakultät Rehabilitationswissenschaften

Methoden der empirischen Bildungsforschung

Hauptverantwortliche Kontaktperson: Christian Kißler

Emil-Figge-Str. 50, 44227 Dortmund

(+49)231 755-6544

Sekretariat-eb.fk13@tu-dortmund.de

christian.kissler@tu-dortmund.de

Vorwort

Das Ziel dieses Manuskripts ist, Studierenden der Rehabilitationspädagogik/ Rehabilitationswissenschaften und der (Sonder-)Pädagogik ein grundlegendes Verständnis über die Anwendung statistischer Verfahren zur Auswertung von Daten zu vermitteln. Den Autoren dieses Lehrskripts ist bewusst, dass viele Studierende pädagogischer Studiengänge ihr Studienfach nicht explizit zur Vertiefung mathematisch-statistischer Inhalte gewählt haben und meist über keine oder geringe Programmierkenntnisse verfügen – erfahrungsgemäß haben viele Studierende nach dem Abitur sogar gehofft, zukünftig der Mathematik aus dem Weg gehen zu können. Allerdings sind statistische Auswertungsverfahren ein wichtiges Element zur wissenschaftlichen Analyse von Daten und somit ist es notwendig, entsprechende Grundlagenkenntnisse im Rahmen des Studiums zu erwerben.

Notwendige Kenntnisse zur Anwendung statistischer Verfahren werden in diesem Lehrskript unter besonderer Berücksichtigung des sehr heterogenen Vorwissens der Studierenden vermittelt. Dies bedeutet: Machen Sie sich keine Sorgen darüber, dass Sie nicht mitkommen werden! In diesem Lehrskript werden alle notwendigen Inhalte ausführlich dargestellt und es wird kein umfangreiches Vorwissen vorausgesetzt. Sie brauchen lediglich etwas Motivation und Interesse am Studienfach sowie Konzentration bei der Lektüre dieses Lehrskripts. Wir empfehlen auch, sich den zu diesem Lehrwerk zugehörigen Datensatz (Kißler, 2023a) sowie das entsprechende R-Skript (Kißler, 2023b) herunterzuladen, um die in diesem Werk dargestellten Schritte in `RStudio`/ `R` parallel zur Lektüre dieses Lehrskripts selbst ausführen zu können. Auch kann es hilfreich sein, bei möglicherweise auftretenden Verständnisproblemen noch einmal im Lehrskript zu vorangegangenen Kapiteln zurückzugehen, um Details nachzulesen und den Inhalt zu festigen. Dann sollten Sie in der Lage sein, die Inhalte dieses Lehrskripts zu verstehen.

1 Einleitung

Die Anwendung statistischer Verfahren zur Auswertung von Daten gehört zu den gängigen methodischen Ansätzen der pädagogischen sowie psychologischen Forschung. In diesem Lehrskript, welches sich inhaltlich auf eine Übungsveranstaltung des Moduls „Empirische Forschungsmethoden“ der Studiengänge Rehabilitationspädagogik und Sonderpädagogik an der Technischen Universität Dortmund bezieht, wird die Auswertung der Daten unter Verwendung der Programmiersprache `R` erfolgen. Da es in einer Übung um das praktische Ausführen bestimmter Handlungen geht, um die eigenen Fertigkeiten weiterzuentwickeln, steht in diesem Lehrskript insbesondere die Vermittlung handlungspraktischer Inhalte, also die Vermittlung der Anwendung statistischer Methoden zur Datenanalyse im Mittelpunkt.

Dieses Werk gliedert sich in vier Teile. Im ersten Teil werden Grundlagen im Umgang mit `R` und `RStudio` (der in diesem Manuskript verwendeten Software bzw. Entwicklungsumgebung zur Datenauswertung; im Folgenden wird aber auch der Einfachheit wegen von „Programm“ gesprochen) vermittelt. Die Installation des Programmes wird beschrieben und es wird dargestellt, wie das Programm `RStudio` aufgebaut ist. Außerdem wird auf die grundlegende Struktur der Programmiersprache `R` eingegangen, welche im „Hintergrund“ von `RStudio` agiert und die Operationen bzw. Berechnungen ausführt. Im zweiten Teil dieses Werkes wird grundlegendes Wissen zum Themenkomplex der deskriptiven Statistik (u.a. Mittelwerte, Streuungsmaße, Erstellung und Verwendung von Abbildungen, einfache Zusammenhangsmaße) vermittelt. Darauf aufbauend folgt im vierten Kapitel die Darstellung der Inferenzstatistik (u.a. *Gauß-Test*, *t-Test*, *ANOVA*). Im letzten Teil des Werkes werden weitere statistische Verfahren dargestellt und so ein Ausblick auf weitere Möglichkeiten im Umgang mit `R` bzw. `RStudio` gegeben (z.B. Grundlagen *bayesianischer Statistik*). Dieser vierte Teil geht über die Inhalte der Übung, zu welcher dieses Manuskript gehört, hinaus und richtet sich eher an Studierende, die im Rahmen einer Bachelorarbeit Fragestellungen unter Verwendung statistischer Methoden nachgehen möchten und dabei ggf. auf andere Verfahren zurückgreifen möchten oder müssen, da sie (besser) geeignet sind, um spezifische Fragestellungen in Abschlussarbeiten zielführend zu bearbeiten.

In diesem Lehrskript wird konsequent mit einem einzigen Datensatz (Kißler, 2023a) gearbeitet, den Sie hier kostenlos herunterladen können: https://eldorado.tu-dortmund.de/handle/2003/42178 oder https://dx.doi.org/10.17877/DE290R-24012

Außerdem werden Sie in diesem Lehrskript *Funktionen* bzw. *Befehle* zur Ausführung von Operationen und Berechnungen in `RStudio` kennenlernen. Diese *Funktionen* bzw. *Befehle* werden in einem sogenannten *RScript* bzw. *R-Skript* verfasst, welches gespeichert werden kann,

um die Berechnungen jederzeit reproduzieren zu können. Die jeweiligen *Befehle* und *Funktionen* werden Ihnen in diesem Lehrskript in Form von Screenshots vorgestellt, wobei diese Screenshots stets nur die für Sie in diesem Moment notwendigen Informationen enthalten, um Verwirrung zu vermeiden und den Cognitive Load bzw. die kognitive Belastung möglichst zu reduzieren. Das *R-Skript* (Kißler, 2023b), welches die zentralen *Befehle* sowie *Funktionen* umfasst, die Sie in diesem Lehrwerk lernen, können Sie hier kostenlos herunterladen, um die Befehle während des Lesens direkt auszuführen: https://eldorado.tu-dortmund.de/handle/2003/42199 oder https://dx.doi.org/10.17877/DE290R-24033

2 Grundlagen im Umgang mit R und RStudio

In diesem Kapitel werden Grundlagen zum Umgang mit `R` und `RStudio` vorgestellt, sodass Sie darauf vorbereitet werden, ab Kapitel 3 mit der Anwendung statistischer Methoden zur Auswertung von Daten zu beginnen. So wird im Folgenden zunächst vorgestellt, worum es sich bei dem Programm `RStudio` und der Programmiersprache `R` handelt (Kapitel 2.1) und es wird beschrieben, wie Sie beides installieren (Kapitel 2.2). Danach wird Ihnen der Aufbau der Benutzeroberfläche von `RStudio` erklärt (Kapitel 2.3). Anschließend (Kapitel 2.4) wird aufgezeigt, wie Funktionen in `RStudio` prinzipiell aufgebaut sind und ausgeführt werden können (*Code*-Struktur). Darauf aufbauend wird erläutert, wie der Funktionsumfang von `RStudio` mit *Packages* erweitert werden kann (Kapitel 2.5). Weil in `RStudio` über eine interne *Help-Funktion* hilfreiche Informationen zur Anwendung von Funktionen abgerufen werden können, wird diese *Help-Funktion* in Kapitel 2.6 eingeführt. Anschließend wird dargestellt, wie Variablen Werte zugewiesen bzw. wie Werte in `RStudio` zur weiteren Verwendung zwischengespeichert werden können (Kapitel 2.7) und welche Datentypen bei der Auswertung von Daten prinzipiell zu unterscheiden sind (Kapitel 2.8). Zur Auswertung von Daten ist es stets notwendig, Datensätze einzulesen – daher wird in Kapitel 2.9 dargestellt, wie solche Datensätze in `RStudio` eingelesen werden können. In Kapitel 2.10 werden dann die ersten einfacheren Operationen in `RStudio` ausgeführt, indem unterschiedliche (Teil-) Datensätze zu einem gemeinsamen Datensatz zusammengefügt werden. Nach diesen einleitenden Kapiteln sind Sie bereit, in Kapitel 3 mit der (deskriptiven) Auswertung von Daten zu beginnen.

2.1 Worum handelt es sich bei R und RStudio?

Bei `R` handelt es sich um eine Programmiersprache, die sich insbesondere zur Datenmanipulation (Import / Export von Daten, Transformation von Variablen etc.), zur statistische Datenanalyse (lineare / nicht-lineare Regressionen, Varianzanalyse etc.) und zur grafischen Darstellung von Daten bzw. der Erstellung von Abbildungen eignet. Diese Datenmanipulation, statistische Datenanalyse und grafische Darstellung von Daten erfolgen in `R` jeweils mithilfe von *Befehlen* in Form von *Funktionen*. Die konkreten *Befehle*, die in `R` ausgeführt werden sollen, werden mithilfe der Programmiersprache ausgedrückt.

Bei `RStudio` handelt es sich um einen *Editor* bzw. um eine *Entwicklungsumgebung*, in welcher diese *Befehle* in Form der Programmiersprache `R` niedergeschrieben werden. Der *Code*, der dabei produziert wird, wird als *Skript* bzw *Script* bezeichnet. `RStudio` bietet den Vorteil, dass längere Skripts in diesem Programm gespeichert und einzelne Arbeitsschritte

dokumentiert werden können. Außerdem können Zwischenergebnisse, die sich bei Berechnungen ergeben, abgespeichert und bei späteren Berechnungen weiterverwendet werden.

Viele statistische Grundfunktionen (z. B. das Berechnen eines Mittelwertes oder einer Standardabweichung) sind in `R` bzw. `RStudio` bereits hinterlegt und können durch spezifische Funktionen schnell ausgeführt werden. Der Funktionsumfang in `R` bzw. `RStudio` kann durch *packages* (vergleichbar mit „Add-Ons"), die aus dem Internet heruntergeladen und dann auf dem Endgerät installiert werden, erweitert werden.

2.2 Installation von R und RStudio

In diesem Unterkapitel wird die Installation von `R` und `RStudio` auf Ihrem Endgerät kurz beschrieben. Den Download der beiden Programme können Sie über die folgenden Links vornehmen:

1) R: https://cran.r-project.org/
2) RStudio: https://www.rstudio.com/products/rstudio/download/#download

Wichtig ist dabei, dass Sie nach dem Download **zuerst die Programmiersprache `R`** und **dann anschließend `RStudio`** installieren. Die Programmiersprache `R` muss in jedem Fall zuerst installiert werden, da sie die Voraussetzung dafür ist, dass `RStudio` korrekt funktioniert.

Wenn Sie dem Link zum Download von `R` gefolgt sind, so müssen Sie die korrekte Version von `R` wählen. Welche Version in Ihrem Fall zu downloaden ist, hängt dabei von Ihrem Betriebssystem ab (Windows, macOS oder Linux). Bitte downloaden Sie nur die Version, die dem Betriebssystem entspricht, das auf Ihrem Endgerät installiert ist. Nach dem Download müssen Sie die heruntergeladene Datei öffnen und den Installationsvorgang durchführen. Wie genau dieser Installationsvorgang abläuft, hängt von dem Betriebssystem ab, das auf Ihrem Endgerät installiert ist. Jede Software, die Sie im Rahmen dieses Skripts benötigen und herunterladen, wird von Drittanbietern bereitgestellt, auf deren Inhalte wir keinen Einfluss haben. Daher kann eine Haftung für die Funktionalität dieser Software bzw. für eventuelle Schäden, die an Ihrem Endgerät durch die Software entstehen, nicht übernommen werden.

Nachdem Sie `R` installiert haben, können Sie `RStudio` installieren. Zuvor müssen Sie die Installationsdatei für das Programm `RStudio` über den oben genannten Link herunterladen. Es gibt auf der Seite von `RStudio` unterschiedliche Versionen zum Download, in der Regel dürfte für Sie die „Free"-Version die korrekte Version sein. Auch bei `RStudio` gibt es für unterschiedliche Betriebssysteme unterschiedliche Download-Versionen. Bitte achten Sie auch

hier darauf, die richtige Version herunterzuladen. Sobald Sie die Installationsdatei heruntergeladen haben, können Sie diese Datei starten und den Installationsvorgang wie bei der Installation von R durchführen.

Sobald Sie beide Programme installiert haben, öffnen Sie bitte das Programm RStudio. R und RStudio können Sie anhand des Logos auseinanderhalten:

- R:  (R Foundation, 2016)
- RStudio:  (RStudio, 2023)

Wählen Sie das Programm mit dem korrekten Symbol aus (bei RStudio kann es der Fall sein, dass das Symbol für das Programm nur das weiße R im hellblauen Kreis ohne das Wort „Studio“ in Schriftsprache ist) und starten Sie das Programm. Sobald Sie dies getan haben, sehen Sie die Benutzeroberfläche des Programms RStudio vor sich. Bereits nach der Installation von R erscheint das Icon von R auf der Benutzeroberfläche Ihres Endgerätes bzw. Desktops. Auf dieses Icon müssen Sie aber innerhalb dieses Kurses niemals klicken. Zum Öffnen von RStudio klicken Sie auf das Icon, das nach der Installation von RStudio auf Ihrem Endgerät verfügbar ist und mit dem Sie RStudio öffnen. Achten Sie also darauf, dass Sie auf das korrekte Icon klicken, um RStudio zu öffnen.

2.3 Aufbau der Benutzeroberfläche von RStudio

In diesem Unterkapitel wird die Benutzeroberfläche des Programms RStudio beschrieben. Sollte die Benutzeroberfläche, die Sie auf Ihrem Bildschirm sehen, anders aussehen, als es im Folgenden dargestellt wird, so prüfen Sie bitte, ob Sie das korrekte Programm geöffnet haben – eventuell haben Sie anstatt von RStudio versehentlich R geöffnet. In jedem Fall sollte das Programm, das Sie sehen, so aussehen, wie es in Abbildung 1 oder 3 dargestellt ist.

In Abbildung 1 fehlt in der Entwicklungsumgebung noch das *Script*-Fenster, in das Sie Ihren *Programmiercode* (bzw. in Kurzform auch *Code* genannt) niederschreiben können. Im Folgenden wird nun ein solches *Script*-Fenster geöffnet. Dazu klicken Sie bitte ganz oben links im Programm auf das weiße Plus im grünen Kreis. Dort wählen Sie *RScript* aus. Diesen Schritt können Sie auch Abbildung 2 entnehmen.

Wenn Sie *RScript* ausgewählt haben, so öffnet sich bei Ihnen nun das *Script*, in das Sie Ihren *Code* eingeben können. Ihre Entwicklungsumgebung sollte nun so aussehen, wie sie in Abbildung 3 dargestellt wird. Dies bedeutet, dass die Benutzeroberfläche nun aus 4 Quadranten bzw. Fenstern bestehen sollte, die jedoch unterschiedlich groß sein können.

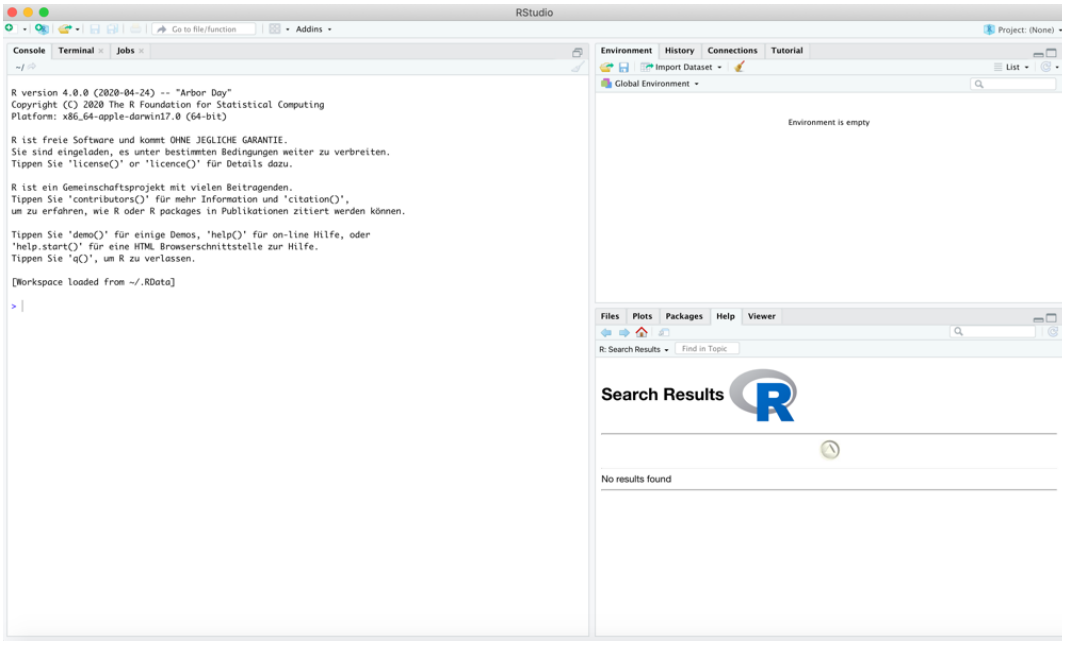

Abbildung 1: Oberfläche von RStudio

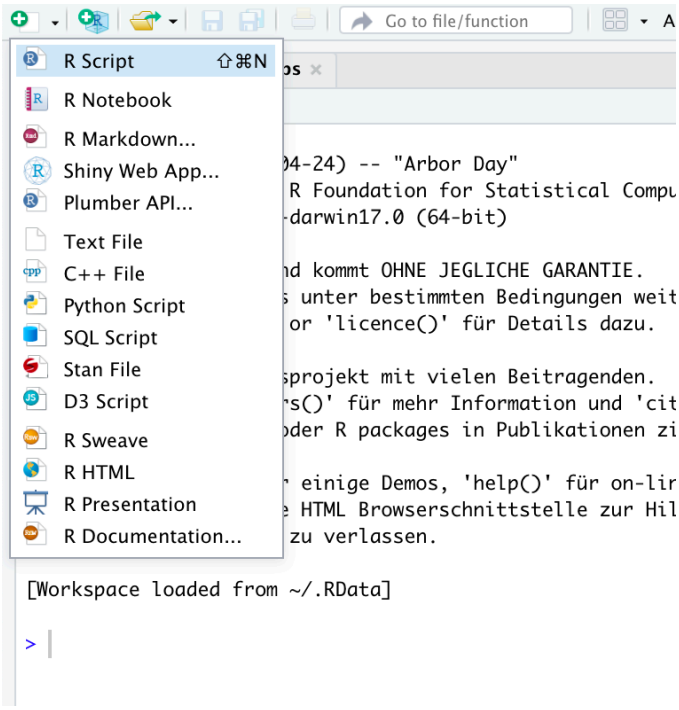

Abbildung 2: RScript/ R-Skript öffnen

Im Folgenden wird nun dargestellt, 1) was die Funktionen der unterschiedlichen Fenster sind, die in Abbildung 3 in `RStudio` geöffnet sind, und 2) wie diese unterschiedlichen Fenster jeweils sowie in ihrer Gesamtheit genannt werden. Die konkreten Bezeichnungen sind von Relevanz, um sicherzustellen, dass es in den folgenden Kapiteln nicht zu Missverständnissen kommt und Sie alle Ausführungen nachvollziehen können.

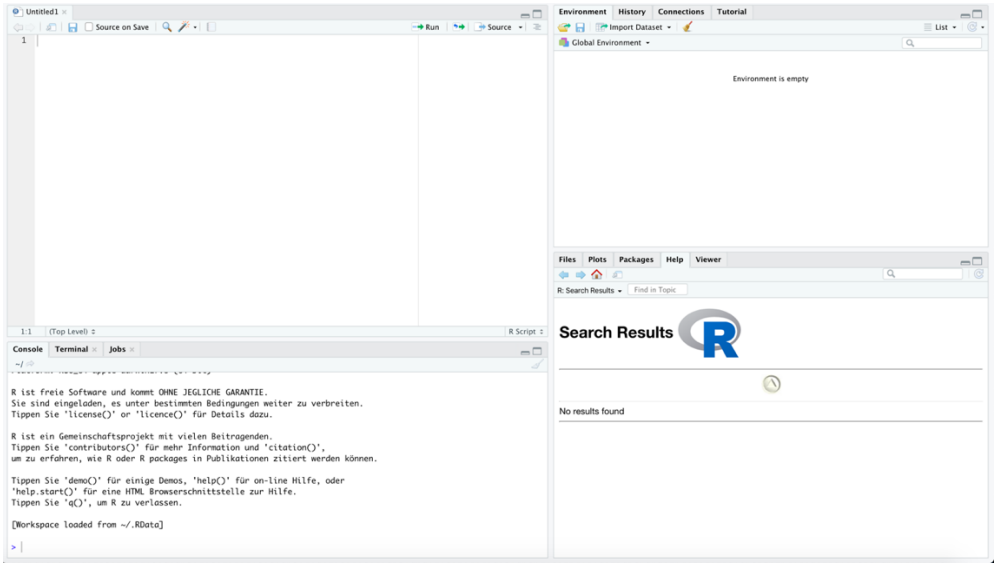

Abbildung 3: Oberfläche von RStudio mit geöffnetem RScript

In Abbildung 4 sehen Sie den sogenannten *Work-Space* rot markiert. Unter dem *Work-Space* ist also die Benutzeroberfläche zu verstehen, welche alle vier Arbeitsfenster in ihrer Gesamtheit umfasst.

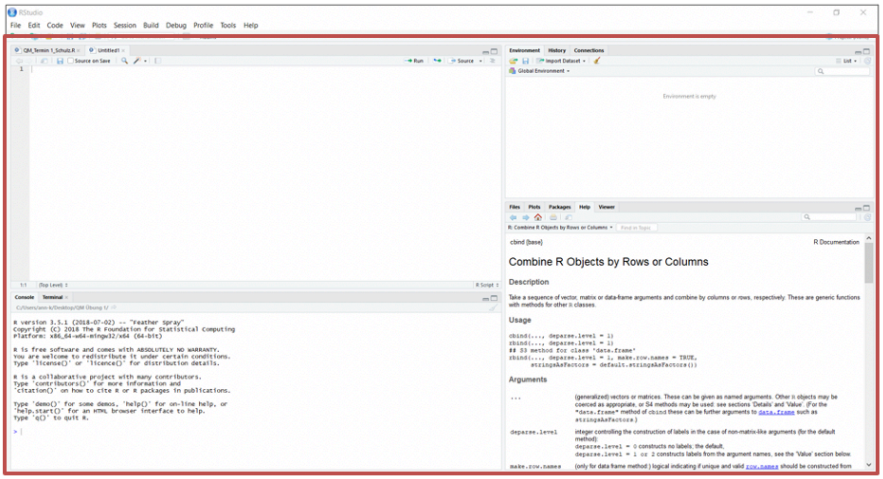

Abbildung 4: In rot markiert: Work Space

In Abbildung 5 ist im *Work-Space* das Fenster links-oben rot hervorgehoben. Es handelt sich dabei um das *RScript-Fenster*, das Sie bereits in einem vorherigen Schritt selbst geöffnet haben. Hier können Sie *Code* zeilenweise hinzufügen, um *Befehle* in `RStudio` auszuführen. Geschriebene *Befehle* können Sie in `RStudio` ausführen, indem Sie auf Ihrer Tastatur die Tasten *Strg+Enter* (bei Windows) oder *Command+Enter* (bei macOS) gleichzeitig drücken. Der Aufbau von *Code* wird Ihnen in Kapitel 2.4 genauer erklärt.

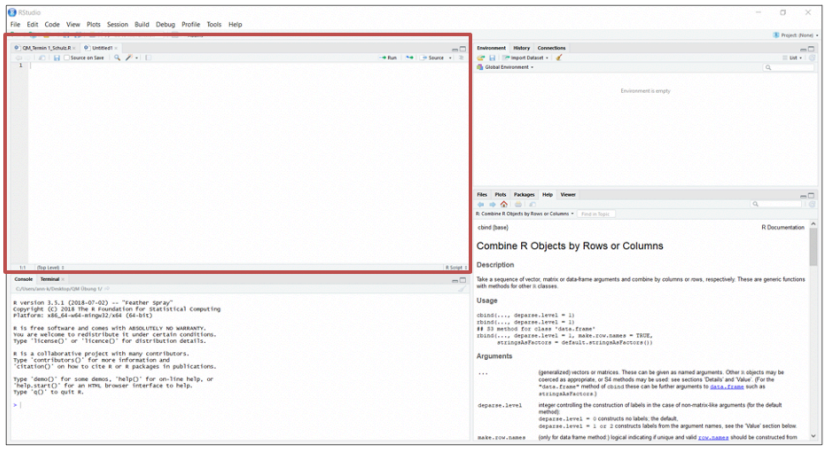

Abbildung 5: In rot markiert: Das RScript/ R-Skript

Wenn Sie *Code* im *RScript*-Fenster ausgeführt haben, indem Sie *Strg+Enter* oder *Command+Enter* gedrückt haben, so wird der ausgeführte *Code* sowie ggf. das Ergebnis des ausgeführten *Codes* in der *R-Konsole* angezeigt (Abbildung 6). Wichtig ist, dass Sie Ihren *Code* **nicht** in die *R-Konsole* schreiben. Potenziell ist es möglich, dort *Code* hineinzuschreiben, z. B. für kurze Einzelbefehle. Es ist allerdings ratsam, dies eher selten zu tun, da ansonsten die Erzeugung eines konsistenten *RScriptes* erschwert wird. Wir empfehlen daher, dass Sie Ihren Code stets im *RScript* verfassen und die Ergebnisse aus der *R-Konsole* ablesen.

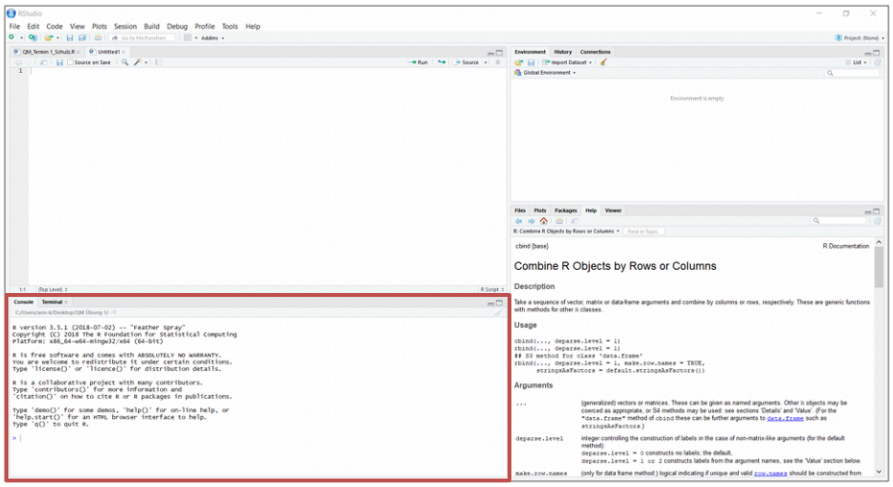

Abbildung 6: In rot markiert: Die R-Konsole

In Abbildung 7 ist das *Environment* (rechts-oben im *Work-Space*) zu sehen. Wenn Sie komplexere Berechnungen durchführen möchten, so kann es sein, dass Sie Zwischenergebnisse speichern müssen, um diese später wieder abrufen und mit ihnen weiterrechnen zu können. Außerdem werden Sie im Folgenden immer wieder mit Datensätzen umgehen müssen, die Sie in `RStudio` einlesen oder erstellen werden. Diese in `RStudio` hinterlegten bzw. gespeicherten

Datensätze und Zwischenergebnisse sind im Environment einsehbar. Sie können sich das Environment also wie das Arbeitsgedächtnis von `RStudio` vorstellen.

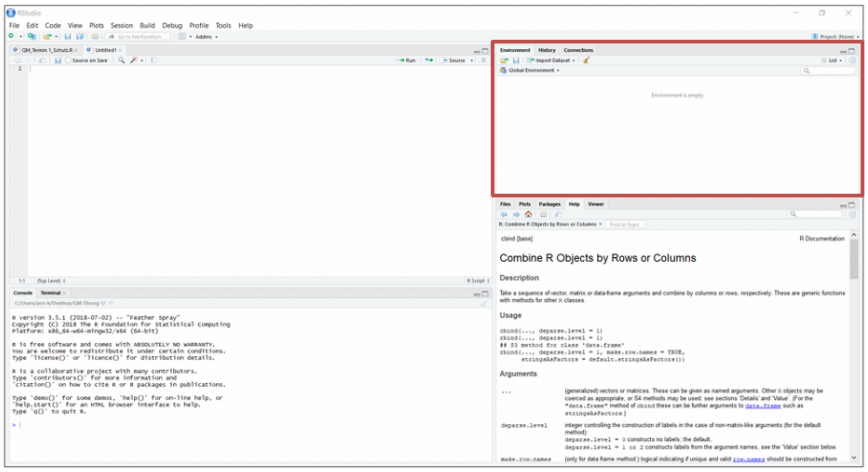

Abbildung 7: In rot markiert - History/ Environment

Unten rechts im *Work-Space* (Abbildung 8) sehen Sie in rot markiert ein Fenster mit mehreren Reitern (u.a. *Plot*, *Help*, *Packages*). Diesen Bereich nennen wir im Folgenden *Information-Space*, weil ihm viele zusätzliche Informationen zu entnehmen sind. So kann der *Information-Space* zu verschiedenen Zwecken genutzt werden und wird Ihnen immer wieder von Nutzen sein. Falls Sie Abbildungen erstellen, werden Ihnen diese in diesem Fenster angezeigt. In `RStudio` gibt es darüber hinaus auch eine *Help-Funktion*, die Ihnen den Aufbau von Befehlen/ Funktionen erklärt. Auch diese Hilfestellungen, die Sie mit der *Help-Funktion* aufrufen können, werden in dem Fenster unten rechts angezeigt. Außerdem können Sie die *Packages* (Add-Ons) einsehen, die Sie bisher installiert haben, und Sie können überprüfen, ob die *Packages* aktiviert sind, sodass die Funktionen aus den *Packages* abgerufen werden können.

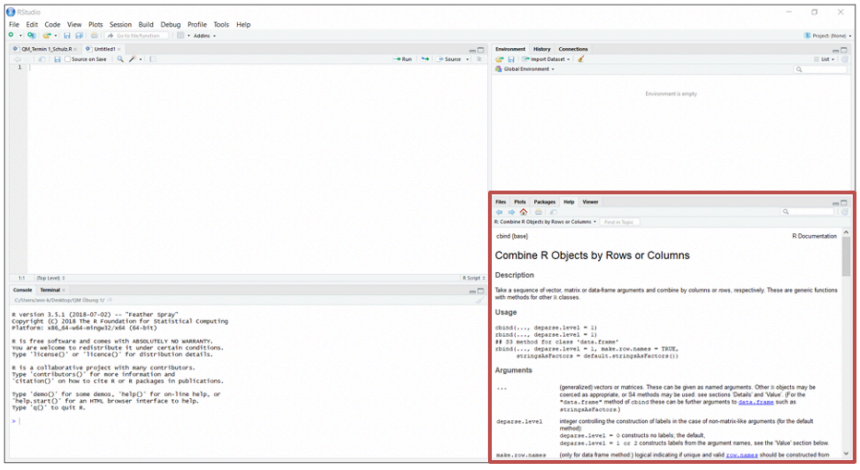

Abbildung 8: In rot markiert - Packages, Plot, Help (Information-Space)

Im folgenden Kapitel erfahren Sie mehr über diese *Packages* und den Aufbau von *Code*, den Sie im *RScript* verfassen können.

2.4 Code, grundlegende mathematische Operationen, Funktionen & Packages

Als *Code* wird das bezeichnet, das 1) im *RScript*-Fenster von Ihnen verfasst wird und 2) als Befehl ausgeführt werden soll. Neben *Code* können Sie im *RScript*-Fenster auch *Kommentare* schreiben. Kommentare können Sie nutzen, um für sich oder andere Personen Anmerkungen (z.B. zur Erläuterung Ihres Codes) im *RScript*-Fenster zu hinterlassen. Jede Kommentarzeile wird mit dem Rautezeichen (#) eingeleitet und beim Ausführen des *Script-Codes* nicht als Befehl ausgeführt bzw. wird von R ignoriert. Befehle können einfache Rechenoperationen sein oder es kann sich bei ihnen um komplexere Funktionen handeln.

Abbildung 9 ist die Durchführung einfacher Rechenoperationen zu entnehmen. Oben links im *RScript* wurde der entsprechende *Code* niedergeschrieben, der dann mit der Tastenkombination *Strg+Enter* ausgeführt wurde. Die ausgeführten Rechnungen und die dazugehörigen Ergebnisse sind unten-links in der *R-Konsole* zu sehen.

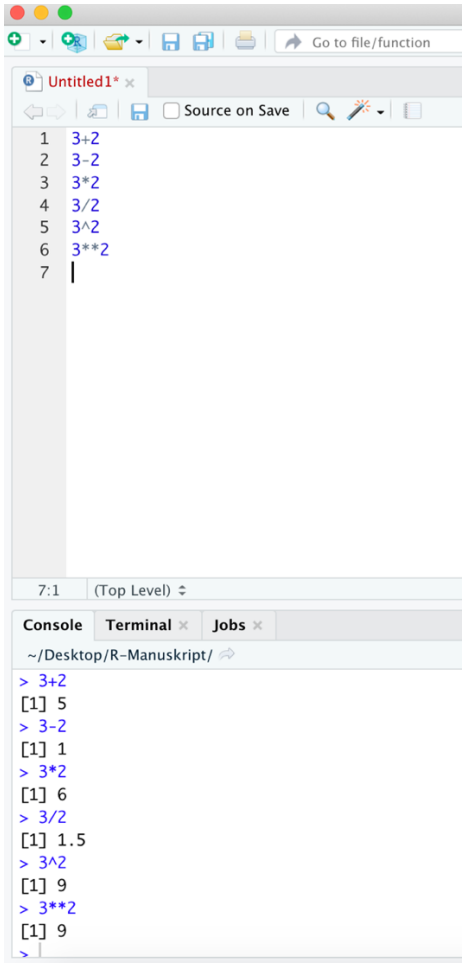

Abbildung 9: Code zur Durchführung einfacher Rechenoperationen

In Tabelle 1 wird auf der rechten Seite der (Beispiel-)*Code* dargestellt, mit dem einfache Rechenoperationen durchgeführt werden können. Bitte beachten Sie in diesem Kontext, welche Zeichen Sie zur Durchführung spezifischer Berechnungen verwenden müssen. Neben den einfachen Rechenoperationen, die in Abbildung 9 bereits dargestellt sind, wird in Tabelle 1 auch eine Funktion eingeführt, mit der in `R` eine spezifische Berechnung (das mathematische Ziehen einer Wurzel) durchgeführt werden kann*:*

```
sqrt()
```

Tabelle 1: Einfache Rechenoperationen in R

Arithmetische Operatoren	Beispielhafte Eingabe in R
Addition (+)	`2+2`
Subtraktion (-)	`2-2`
Multiplikation (*)	`2*2`
Division (/)	`2/2`
Potenzieren (^)	`2^2 oder 2**2`
Wurzelziehen (√)	`sqrt (2)`

Funktionen haben in `R` grundlegend/ prinzipiell den folgenden Aufbau:

```
function(x)
```

Dabei bezeichnet der Ausdruck `function` die Funktion, die ausgeführt werden soll (z.B. gibt es in `R` die Funktion `mean` zur Berechnung eines *Mittelwertes*). Hinter einer Funktion verbirgt sich in der Regel eine Reihe von Operationen, die nacheinander durchgeführt werden (so wird bei der Mittelwertberechnung durch die Funktion `mean` die Summe einer Reihe von Werten gebildet, die dann durch die Anzahl der Werte dividiert wird). Hinter der Bezeichnung der Funktion steht eine Klammer, in welcher angegeben wird, auf was sich die Funktion bezieht (im obigen Beispiel bezieht sich die Funktion `function` auf das *x*). Das, was in der Klammer steht, wird als *Argument* bezeichnet. Häufig kommt es vor, dass mehrere *Argumente* hintereinander in der Klammer stehen, z.B. `function(x, y, z)`. Diese *Argumente* können dabei a) auf *Objekte* (z.B. *Datensätze*), mit denen etwas gemacht werden soll, oder b) auf

Spezifikationen, die bei der Ausführung der Funktion beachtet werden sollen (z.B. die Nicht-Berücksichtigung fehlender Werte) verweisen.

Eine Funktion, die weiter oben bereits eingeführt wurde, lautet `sqrt(x)`, wobei *x* dann beispielsweise eine Zahl oder ein Datensatz sein könnte. In Abbildung 10 ist der Befehl `sqrt(16)` dargestellt. Der Befehlt `sqrt(16)` bedeutet, dass die Funktion `sqrt()` auf die Zahl `16` angewendet werden soll. Bei `sqrt()` handelt es sich um eine Funktion, welche die Wurzel aus der Zahl errechnet, die bei der Funktion `sqrt()` in Klammern als Argument angegeben ist. Somit könnte der Befehl `sqrt(16)` auch folgendermaßen sinngemäß gelesen werden: Ziehe die Wurzel aus der Zahl 16. Das Ergebnis dieser Berechnung (also: 4) wird dann im Fenster unten links (in der *R-Konsole*) ausgegeben. Abbildung 10 ist diese Berechnung sowie der Output und somit die prinzipielle Funktionsweise von Funktionen in `RStudio` zu entnehmen.

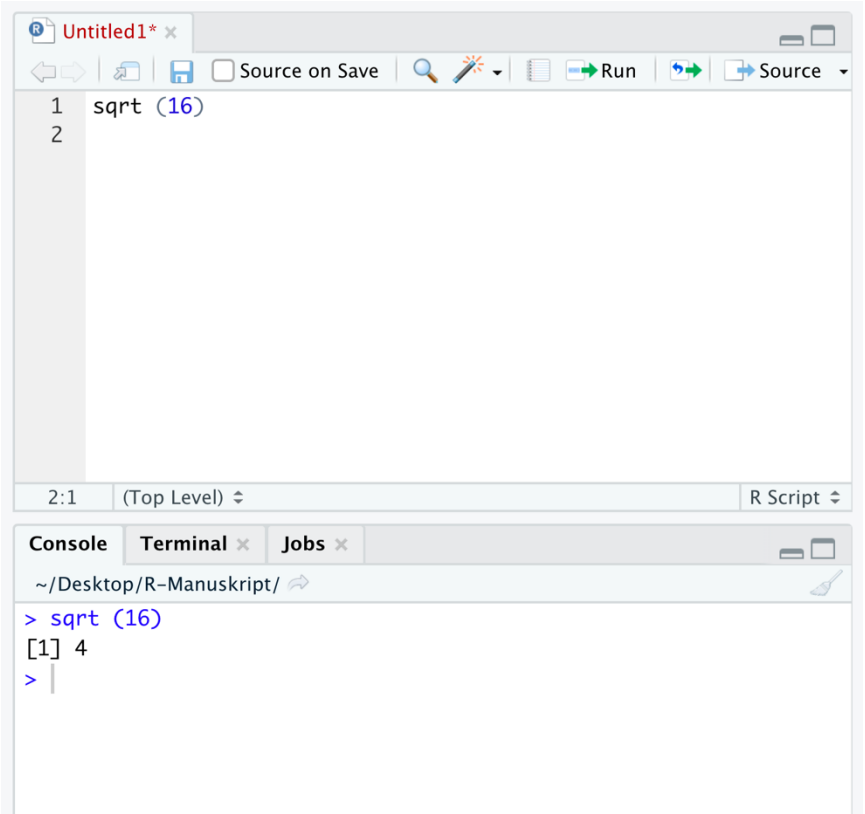

Abbildung 10: Prinzipielle funktionsweise von Funktionen in RStudio

Während die Funktion `sqrt()` mit nur einem Argument operiert, gibt es andere Funktionen, bei welchen mehrere Argumente angegeben werden können. Ein Beispiel für eine Funktion mit mehr als einem Argument ist `round()`. Die Funktion `round()` kann eingesetzt werden, um Zahlen mathematisch (nicht kaufmännisch!) zu runden. Beim kaufmännischen Runden wird

abgerundet, wenn die jeweilige Dezimalstelle hinter (von links nach rechts) der Dezimalstelle, auf die gerundet werden soll, 0, 1, 2, 3 oder 4 beträgt; im Falle von 5, 6, 7, 8 und 9 wird aufgerundet. Wenn also die Zahl 0.145 auf die zweite Dezimalstelle gerundet werden soll, so würde das Ergebnis 0.15 lauten, weil die erste Dezimalstelle nach der zweiten Dezimalstelle eine 5 ist und bei 5 immer aufgerundet wird, wenn das kaufmännische Runden angewendet wird. Beim mathematischen Runden wird prinzipiell so wie beim kaufmännischen Runden gerundet, jedoch wird bei einer 5 anders als beim kaufmännischen Runden verfahren. Denn bei einer 5 wird immer so gerundet, dass die letzte Ziffer nach dem Runden eine „gerade Zahl“ ist (also 0, 2, 4, 6 oder 8 ist). So würde die obige Zahl (0.145) auf 0.14 abgerundet werden, weil 0,15 keine gerade Zahl wäre, 0.14 jedoch die nächste gerade Zahl ist. Prinzipiell ist die Funktion `round()`, die den Regeln des mathematischen Rundens folgt, folgendermaßen aufgebaut:

```
round (x, digits = y)
```

Das Argument `x` gibt dabei an, welche Zahl gerundet werden soll und das *Argument digits = y* gibt an, auf wie viele Stellen nach dem Komma gerundet werden soll. Der Abbildung 11 kann beispielhaft entnommen werden, wie die Funktion `round()` funktioniert. Wenn Sie zwingend abrunden möchten, dann können Sie die Funktion `floor()` benutzen; zum Aufrunden von Zahlen eignet sich die Funktion `ceiling()`.

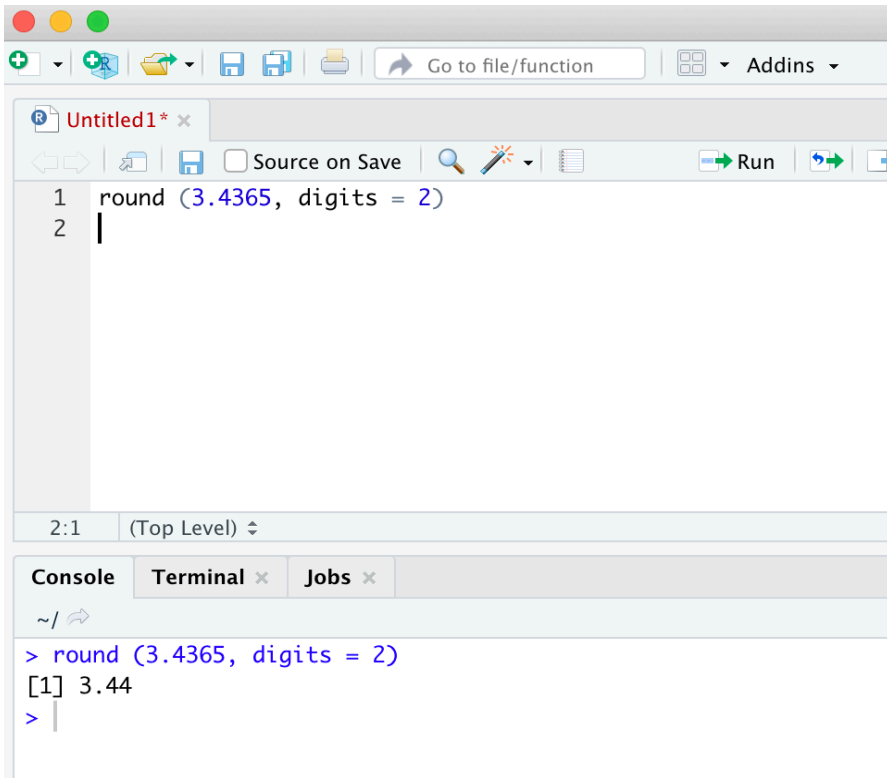

Abbildung 11: Runden von Zahlen

2.5 Packages zur Funktionserweiterung

Im Vorherigen Kapitel wurden Funktionen besprochen, um spezifische Berechnungen in `RStudio` vornehmen zu können. Der Funktionsumfang kann dabei durch das Einbinden (= Download und Installation) von *Packages* darüber hinaus noch deutlich erweitert werden. *Packages* stellen dabei eine Ansammlung weiterer Funktionen dar.

Es gibt einige *Packages*, die sich im praktischen Umgang mit `RStudio` bzw. bei der Auswertung von Daten als sehr nützlich erwiesen haben und die den Funktionsumfang von `RStudio` deutlich erweitern. In Tabelle 2 werden einige dieser *Packages* dargestellt, wobei einige dieser *Packages* bezüglich ihres Funktionsumfanges auch über die Inhalte dieses Manuskripts hinausgehen.

Tabelle 2: Package-Empfehlungen

Name des Packages	*Autor:innen*	*Funktionen*
psych	*Revelle (2022)*	*Multivariate Analysen und Skalenkonstruktion*
car	*Fox, Weisberg et al. (2022)*	*Regressionsanalysen*
lme4 *nlme*	*Bates et al. (2022)* *Pinheiro et al. (2022)*	*Mehrebenenanalysen*
Lavaan *sem*	*Rosseel et al. (2022)* *Fox, Nie et al. (2022)*	*Strukturgleichungsmodelle*
mice	*Van Buuren et al. (2022)*	*Multiple Imputationen*

Es bietet sich an, diese *Packages* herunterzuladen und zu installieren, damit sie bei Bedarf über die Funktion `library()` oder `require()` aktiviert werden können. Ob Sie zur Aktivierung eines *Packages* die Funktion `require()` oder `library()` verwenden, ist dabei nachrangig: Sie führen zum gleichen Ergebnis. Das *Package*, das in der Klammer als *Argument* benannt und somit angesteuert wird, wird aktiviert. In `RStudio` werden Sie immer wieder auf unterschiedliche Funktionen stoßen, welche die gleichen oder sehr ähnliche Operationen ausführen. So würde sowohl die Ausführung des Befehls

```
library(psych)
```

als auch die Ausführung des Befehls

```
require(psych)
```

dazu führen, dass das *Package psych* aktiviert wird, insofern es zuvor heruntergeladen und installiert wurde. Die Installation eines *Packages* wird mit dem Befehl `install.packages()` ausgeführt und würde in dem Fall, dass das *Package psych* von Revelle (2022) installiert werden soll, folgendermaßen lauten:

```
install.packages („psych")
```

Wie zu sehen ist, steht bei der Verwendung der Funktion `install.packages()` das Argument *psych* in Anführungszeichen, während dies bei der Funktion `library()` oder `require()` nicht der Fall ist. Dies liegt daran, dass sich das, was bei einer Funktion in der Klammer stehen muss, je nach Funktion unterscheiden kann. Hier bietet es sich an, den Aufbau von häufig verwendeten Funktionen individuell zu lernen (wie Vokabeln), um das entsprechende Wissen bei Bedarf abrufen zu können. Insbesondere bei seltener verwendeten Funktionen erweist es sich als nützlich, dass `RStudio` Ihnen eine Hilfe anbietet: Sie können nämlich auf die *Help-Funktion* zurückgreifen, um den Aufbau dessen, was bei einer spezifischen Funktion wie in der Klammer stehen muss, nachzuschlagen. Im folgenden Kapitel wird Ihnen die *Help-Funktion* vorgestellt.

2.6 Die Help-Funktion

Geläufige Befehle in `R` können meist schnell (wie Vokabeln) gelernt werden. Dazu gehört auch die Struktur dessen, was in der Klammer eines Befehls wie angegeben werden muss, damit der Befehl korrekt ausgeführt werden kann. Auch erfahrenen `R`-Nutzer:innen passiert es jedoch immer wieder, dass konkrete Details zu Befehlen nicht vollständig bekannt sind oder vergessen werden. Manchmal würde man auch gerne mehr Details zu einem Befehl erfahren (z.B. unter welchen Bedingungen der Befehl überhaupt ausgeführt werden kann und darf, ob es weiterführende Literatur zu diesem Befehl gibt und/ oder ob spezifische Modifikationen bei den Berechnungen mit diesem Befehl vorgenommen werden können).

Um eine detaillierte Beschreibung der Funktion abzurufen, muss ein

```
?
```

direkt vor den jeweiligen Befehl geschrieben werden: So wird die *Help-Funktion* für diesen Befehl ausgeführt. An dieser Stelle wird die Verwendung der Help-Funktion für den bereits

bekannten Befehl `sqrt()` beispielhaft dargestellt. Durch die Befehlskombination

```
?sqrt
```

öffnet sich unten rechts im *Work-Space* nun ein Fenster, das Ihnen viele Informationen zu dem jeweiligen Befehl anzeigt. In Abbildung 12 wird ein Screenshot von dem *Help-Fenster*, das sich dann öffnet, angezeigt. Dabei kann das Layout der angezeigten Informationen in Abhängigkeit der Größe Ihres Help-Fensters variieren und somit von der Darstellung in Abbildung 12 abweichen.

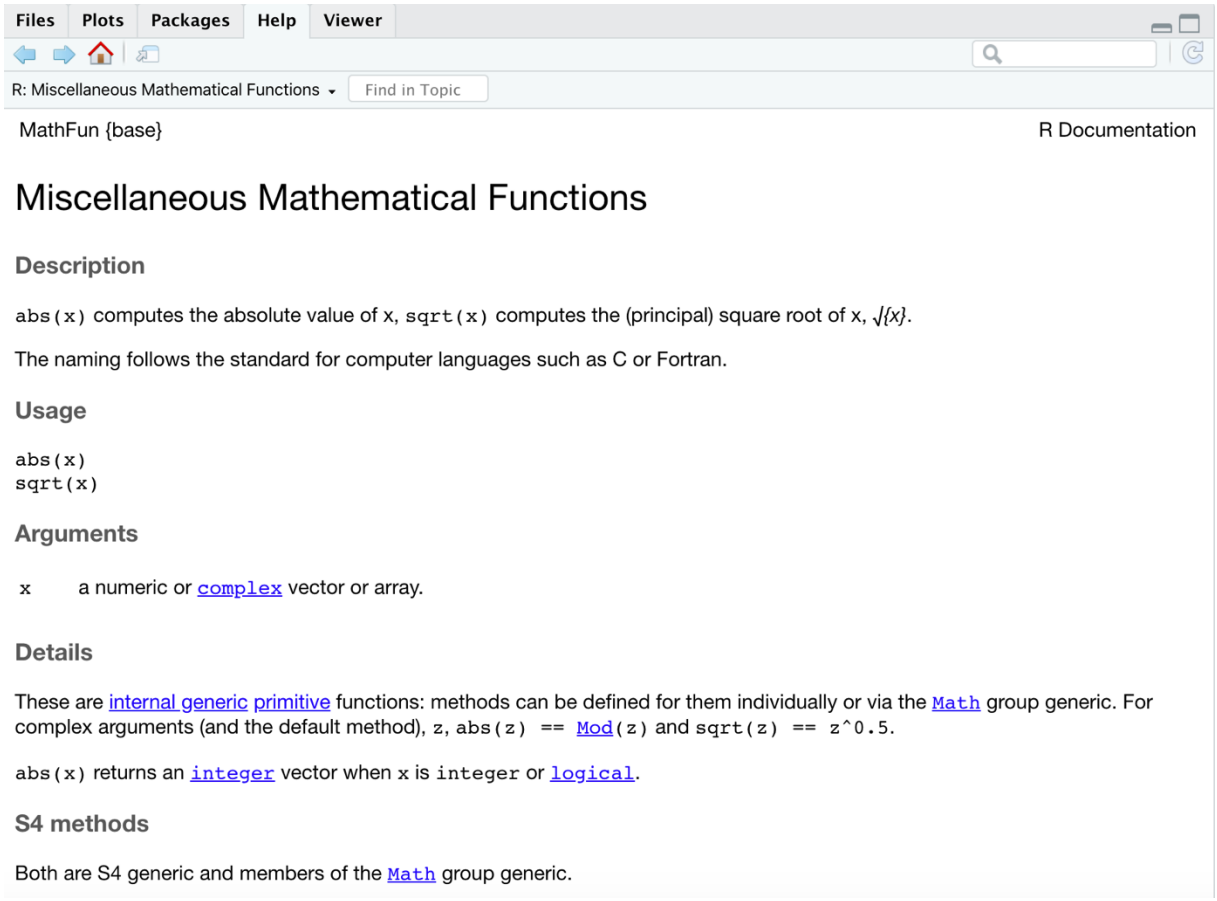

Abbildung 12: Help-Fenster im Information-Space

Alternativ zu dem Befehl

```
?sqrt
```

kann auch die Funktion

```
help(sqrt)
```

genutzt werden, welche zum gleichen Resultat führt (Abbildung 13). Der Befehl `help()` kann also alternativ zur `?`-Funktion verwendet werden.

Wenn Sie anstatt eines Fragezeichens (`?`) zwei Fragezeichen (`??`) vor einen Befehl schreiben (z.B. `??sqrt()`), so öffnet sich ein erweitertes *Help-Fenster*, dem Sie zum Beispiel entnehmen können, zu welchem *Package* ein Befehl gehört. Dies kann insbesondere dann nützlich sein, wenn bei der Ausführung der jeweiligen Funktion eine Fehlermeldung im Output-Fenster erscheinen sollte, weil das *Package*, zu welchem die Funktion gehört, nicht installiert oder nicht aktiviert wurde.

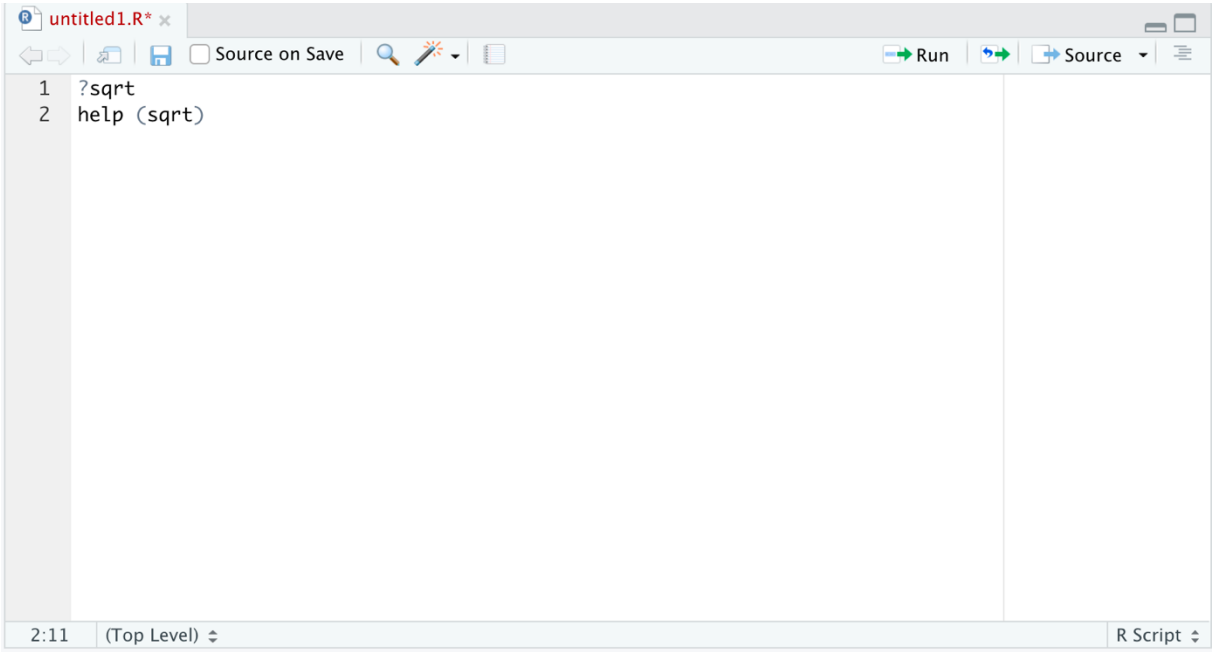

Abbildung 13: Beispielhafte Verwendung der Help-Funktion

2.7 Zuweisungen

In Kapitel 2.3 wurde das *Environment* (das Fenster oben rechts im *Worksprace* von *RStudio*) bereits als ein spezieller Teil vom *Workspace* beschrieben. Das *Environment* ist beispielsweise dann hilfreich, wenn komplexere Berechnungen durchgeführt werden müssen. Denn im *Environment* können Zwischenergebnisse sowie Datensätze gespeichert werden, um diese später wieder abrufen und mit ihnen weiterrechnen zu können. Das *Environment* kann also prinzipiell mit der Funktionsweise eines Arbeitsgedächtnisses verglichen werden. In diesem Unterkapitel werden wir lernen, einfache Daten im *Environment* prinzipiell abzuspeichern bzw. zu hinterlegen.

Um Daten im *Environment* abspeichern zu können, sind *Zuweisungen* notwendig. Zuweisungen führen Sie mithilfe des Zuweisungspfeils (`<-`) aus, welcher aus dem

mathematischen Zeichen für „kleiner als“ (<) und einem Minus (-) besteht. Prinzipiell funktionieren Zuweisungen so, dass sie „etwas“, das rechts vom Zuweisungspfeil steht, „etwas anderem“, das links vom Zuweisungspfeil steht, zuordnen. So können Sie beispielhaft a) dem Objekt *x* den Wert 3 zuordnen

```
x <- 3
```

oder b) das Ergebnis der Rechnung `4+5` als *b* abspeichern:

```
b <- 4+5
```

Anschließend könnten Sie mit *x* und *b* weiterrechnen und das Ergebnis als *c* abspeichern:

```
c <- x*b
```

Abbildung 14 zeigt, wie der zugehörige *Code* im *RScript,* die Zuweisungen im Environment sowie die Rechnungen und die Outputs in *RStudio* aussehen.

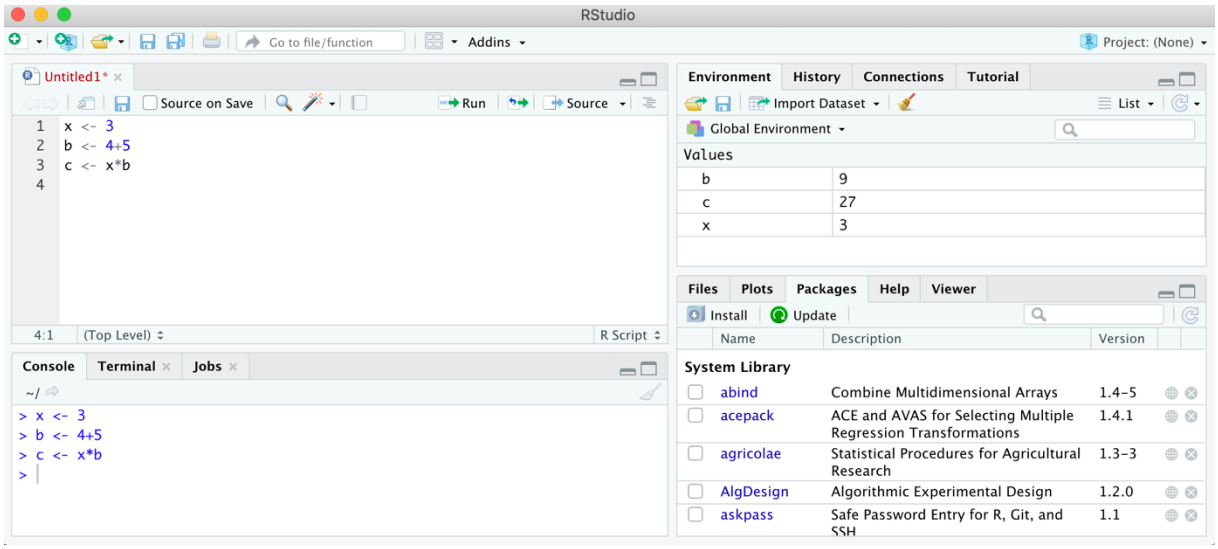

Abbildung 14: Zuweisungen

2.8 Datentypen

Bisher haben wir uns in `RStudio` insbesondere mit Zahlen, mit dem Umgang mit Zahlen und mit Zuweisungen, die für Zahlen stehen, beschäftigt. Neben Zahlen, mit denen „ganz normal“ gerechnet werden kann (= ganze und reelle Zahlen, die auch als *numeric* bezeichnet werden), gibt es aber auch andere Datentypen in `R`. Tabelle 3 stellt gängige Datentypen in `R` dar. *Characters* sind Buchstaben oder Zeichenfolgen wie zum Beispiel Namen. In `RStudio` werden *characters* dadurch erkannt, dass sie nicht wie Zahlen einfach ins Skript geschrieben werden, sondern dass sie zwischen Anführungszeichen stehen. *Characters* zeichnen sich dadurch aus,

dass mit ihnen die meisten mathematischen Operationen nicht sinnvoll möglich sind (so gibt es beispielsweise für die Rechnung Hugo+Klaus kein sinnvolles Ergebnis).

Tabelle 3: Datentypen

Datentyp	Beispiel	Beschreibung
numeric	3.14	Ganze und reelle Zahlen
character	“Otto“, “Hello World“	Buchstaben oder Zeichenfolgen (müssen immer in Hochkommata gesetzt werden)
factor	“12“	Zahlenfolge, die nominal skaliert ist (z. B. ID-Code)
double	.3333337	“floating point value“
integer	-1, 0, 1	Ganze Zahlen
Boolean/logical	TRUE, FALSE	Abfrage von *wahr* oder *falsch*

Faktoren bzw. *factors* zeichnen sich dadurch aus, dass es sich dem Anschein nach um Zahlen handelt, die allerdings nominal skaliert sind und mit denen deshalb ebenso wie mit *characters* nicht sinnvoll gerechnet werden kann. Stellen Sie sich beispielsweise einen Datensatz vor, in dem Sie männliche Personen mit 0, weibliche Personen mit 1 und diverse Personen mit 2 kodieren. Sie können dann nicht einfach eine männliche Person (0) und einen diverse Person (2) addieren und das Ergebnis durch zwei teilen, sodass sie sagen könnten, die Personen wären im Durchschnitt weiblich, weil (0+2)/2 = 1 ist, wobei 1 (wie bereits dargestellt) für weiblich steht. Mit solchen *Faktoren* sind also viele mathematische Operationen nicht möglich, obwohl es sich um Zahlen zu handeln scheint. Daher ist es wichtig, Daten vom Datentyp *numeric* und Daten vom Datentyp *factor* zu unterscheiden (siehe Kapitel 3.1 zu Skalenniveaus).

Für Zuordnungen können aber nicht nur Daten vom Datentyp *numeric* verwendet werden. Abbildung 15 zeigt, dass es auch möglich ist, *characters* für Zuweisungen zu verwenden. Wichtig ist bei der Zuweisung von *characters*, dass das Setzen der Anführungszeichen im *Code* nicht vergessen wird (vgl. Abbildung 15).

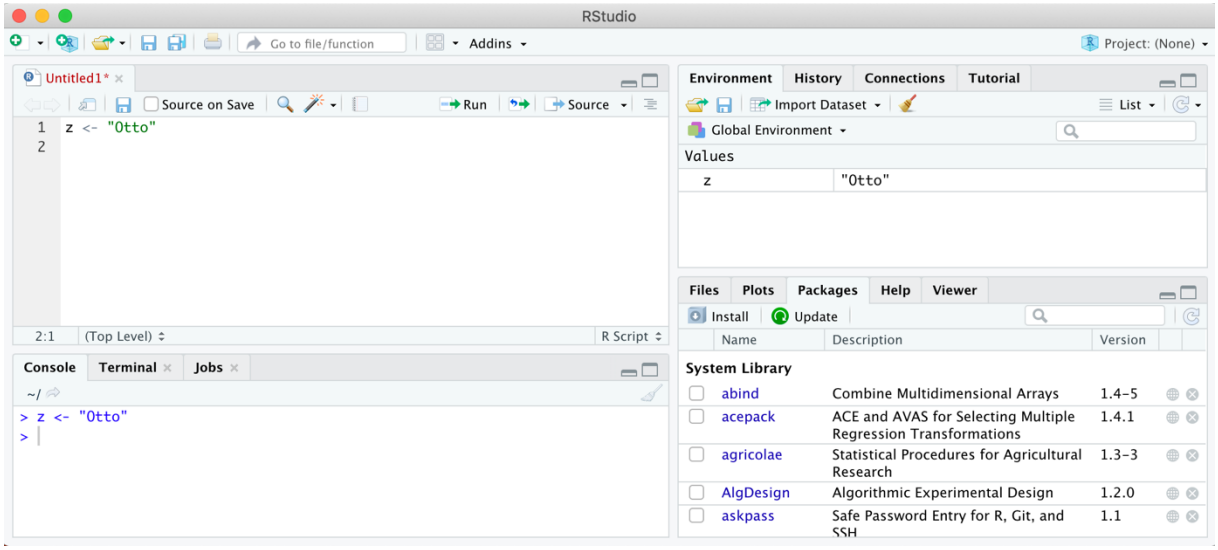

Abbildung 15: Beispielhafte Zuweisung für Characters

Falls unbekannt ist, welchen Datentyp eine Variable aktuell hat, so kann in `RStudio` der hinterlegte Datentyp abgefragt werden, sodass er anschließend in der *Konsole* ausgegeben wird. Die zugehörige Funktion lautet `class()`. Wenn Sie beispielsweise wissen wollen, welchen Datentyp *z* hat, so können Sie dies mit der `class()`−Funktion erfragen:

```
class(z)
```

Anschließend würde in der Konsole *character* ausgegeben werden, was bedeuten würde, dass die Variable *z* den Datentyp *character* hat.

2.9 Datensätze abspeichern, erstellen und einlesen

Bis hierher wurden in diesem Skript nur einzelne Daten im Environment abgespeichert. Abbildung 16 ist zu entnehmen, dass es unter Verwendung der Funktion

```
c()
```

auch möglich ist, mehrere Daten gebündelt im *Environment* zu hinterlegen. Diese (eindimensionale) Bündelung von Komponenten eines Datentyps wird Vektor genannt. Im Environment ist an dem Kürzel *num* des Objektes *a* zu erkennen, dass die Zahlenfolge automatisch als Daten vom Datentyp *numeric* erkannt wurde, während die Namensliste *goppen* als Daten vom Typ *character* erkannt wurde (*chr*). Sowohl *a* als auch *goppen* stellen Vektoren dar, weil sie jeweils geordnete Daten eines Datentyps (entweder *numeric* oder *character*) umfassen und diese Anordnung von Daten jeweils eine einzelne Spalte darstellt und somit eindimensional ist.

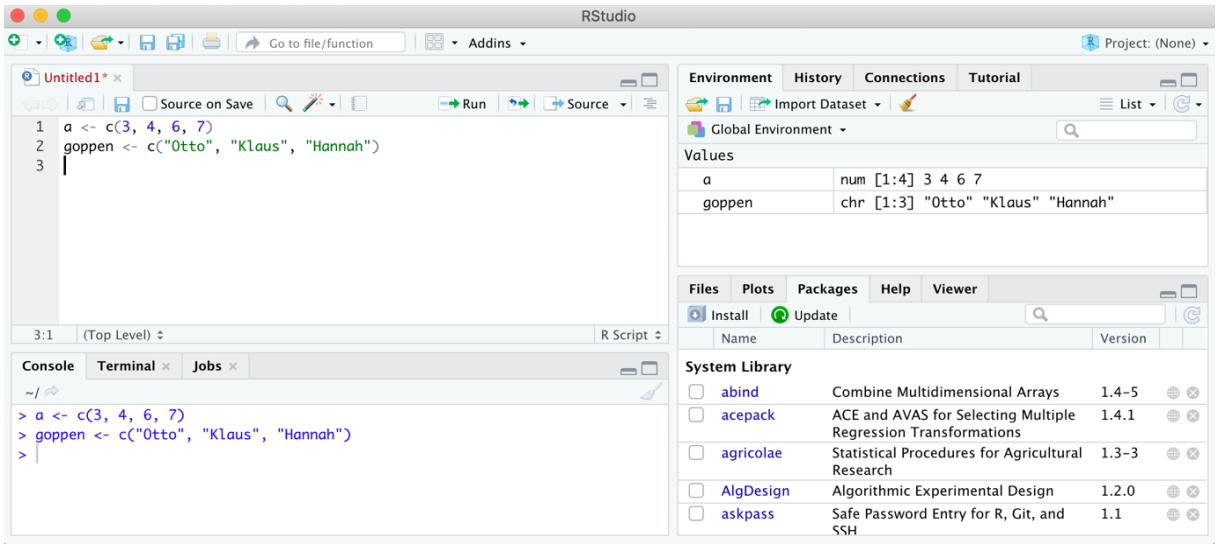

Abbildung 16: Komplexere Zuweisungen in RStudio

Abbildung 16 ist auch zu entnehmen, dass die Variablen bzw. Vektoren, denen etwas zugewiesen wird, beliebig benannt werden können: Sie können dazu einzelne Buchstaben verwenden (zum Beispiel: *a*) oder Buchstabenkombinationen (zum Beispiel: *goppen*). Sie können auch Zahlen mit Buchstaben kombinieren (z.B. *W12d3*). Die Verwendung von Leerzeichen hingegen ist nicht möglich – stattdessen kann jedoch ein Unterstrich (_) verwendet werden. So wäre zum Beispiel *dat_2* eine mögliche bzw. zulässige Bezeichnung.

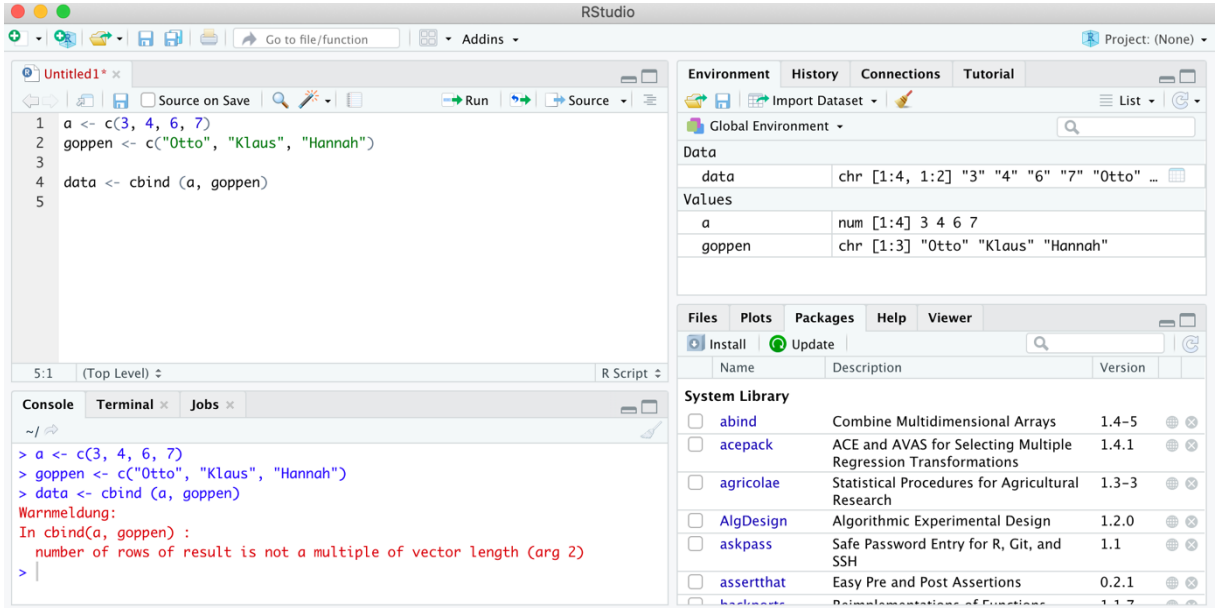

Abbildung 17: Verwendung der Funktion cbind()

Mit der Funktion

```
cbind()
```

können die beiden Vektoren, die zuvor mit der Funktion `c` erzeugt wurden, als zwei Spalten zusammengefügt werden, sodass eine Matrix entsteht, die in diesem Beispiel nun *data* genannt wird (Abbildung 17).

Die Bezeichnung der Funktion `cbind()` ist dabei von „column bind“ (auf Deutsch: Spalten zusammenbinden) abgeleitet. In Abbildung 17 ist zu sehen, dass unten in der *Konsole* nun eine Fehlermeldung angezeigt wird, wenn die `cbind` Funktion in diesem Fall ausgeführt wird. Dies liegt daran, dass der Vektor *a* über 4 Elemente verfügt und der Vektor *goppen* nur 3 Elemente besitzt. Somit kann nicht jedem Element des Vektors *a* auch ein Element des Vektors *goppen* zugeordnet werden. Werden Vektoren miteinander kombiniert, so entsteht eine Matrix: Die eindimensionalen Vektoren werden nebeneinander angeordnet, sodass eine mehrdimensionale Matrix resultiert. Diese Matrix, die in unserem Beispiel nun *data* heißt und erzeugt wurde, ist mit der Funktion

```
View()
```

einsehbar – in diesem Fall lautet der entsprechende Befehl konkret: `View(data)`. Wichtig ist dabei, dass das V in `View()` groß geschrieben werden muss, damit die Funktion korrekt ausgeführt werden kann.

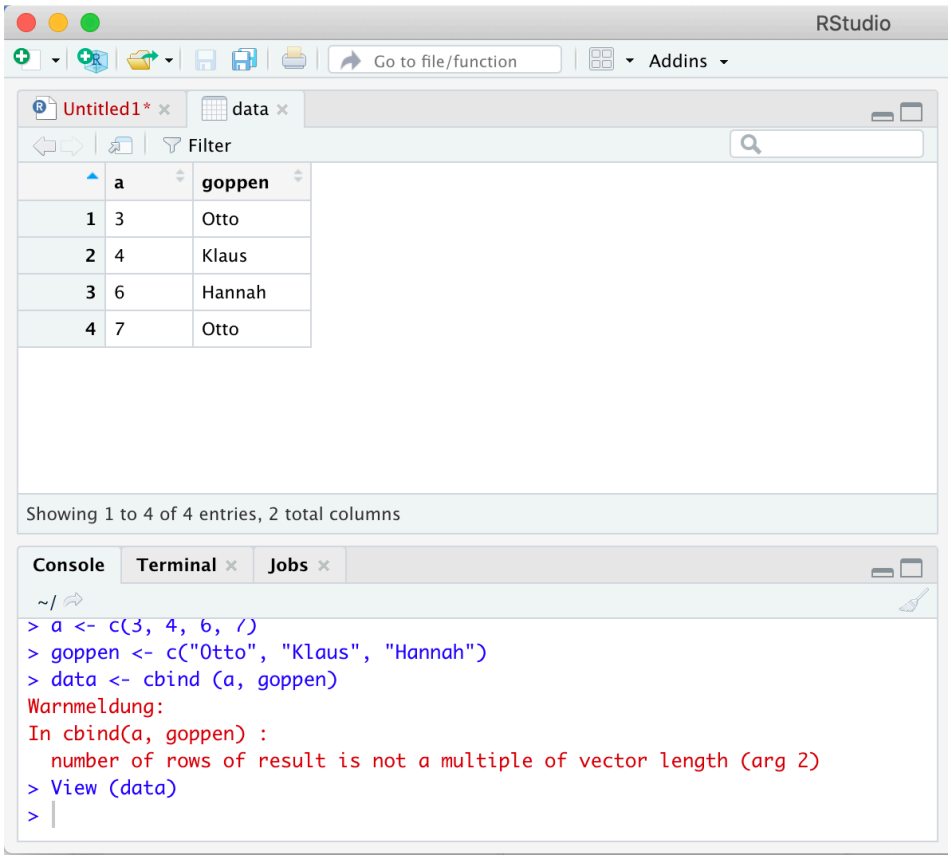

Abbildung 18: View zur Betrachtung von Datensätzen

Abbildung 18 ist zu entnehmen, dass sich, nachdem die Funktion `View(data)` ausgeführt wurde, neben dem *Script*, das in diesem Beispiel „*Untitled1*“ heißt, ein weiterer Reiter geöffnet hat, der „*data*“ heißt. Dort, wo normalerweise das *RScript* zu sehen ist, ist in `RStudio` nun die Matrix zu sehen. Es ist zu erkennen, dass jedem Wert aus *a* ein Datenwert aus *goppen* zugeordnet wurde – entsprechend der Reihenfolge der Objekte in den Vektoren. Auffällig ist jedoch, dass auch der *7* der *character Otto* zugeordnet wurde. Dies liegt daran, dass zwei Vektoren unterschiedlicher Länge zusammengefügt wurden. Bitte achten Sie daher darauf, dass die Vektoren, die Sie zusammenfügen, stets die gleiche Länge aufweisen.

Aber was kann nun getan werden, wenn zwei Vektoren zusammengefügt werden sollen, aber ein Wert (z.B. ein Name) in einem Vektor fehlt bzw. zu wenig vorhanden ist? Um dieses Problem zu beheben, kann an der Stelle, an der die Angabe fehlt, *NA* in den Vektor geschrieben werden. *NA* steht dabei für „*not applicable*“ und verdeutlicht, dass hier ein Wert fehlt. Dies wird in Abbildung 19 dargestellt.

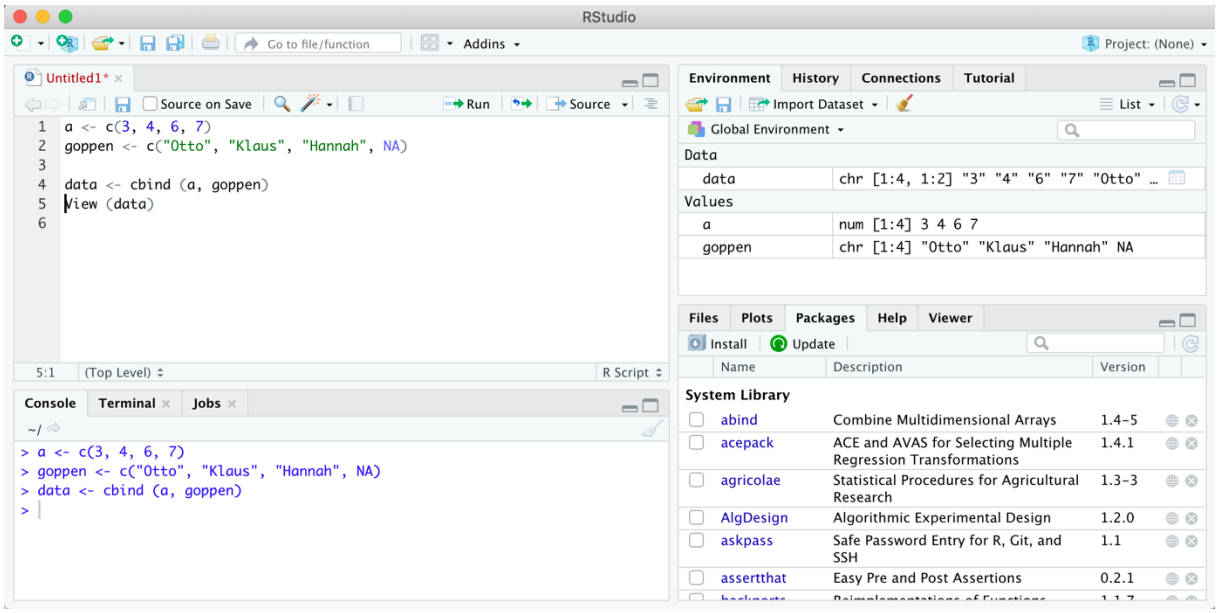

Abbildung 19: Verwendung von NA für fehlende Angaben

Wenn die Matrix *data* nun mit `View()` eingesehen wird (die *Syntax* ist Abbildung 19 zu entnehmen), so öffnet sich dadurch ein Fenster, das in Abbildung 20 dargestellt wird. In diesem Fenster ist zu erkennen, dass an der letzten Stelle, an der eine Angabe fehlt, *NA* steht und *NA* (im Gegensatz zu den anderen Angaben) in grau angezeigt wird. Durch diese andere Einfärbung wird deutlich, dass die jeweilige Angabe fehlt – das Fehlen einer Angabe hat ggf. Einfluss auf Operationen, die in `R` ausgeführt werden können, sowie auf die Ergebnisse, die mit `R` berechnet werden.

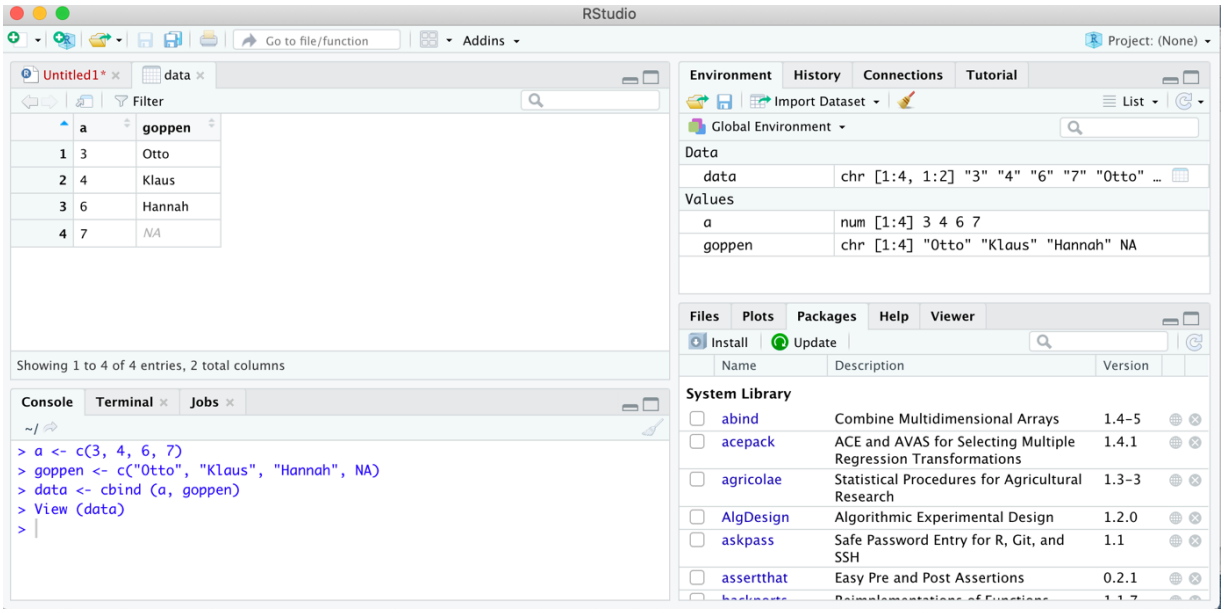

Abbildung 20: Datensätze mit fehlenden Werten (NA)

Mithilfe der zuvor vorgestellten Prinzipien können Sie potenziell umfangreiche Matrizen in `RStudio` eingeben und so Datensätze selbst erzeugen. Häufig sollen aber Datensätze, die bereits in anderen Programmen angelegt wurden (z.B. *Excel* oder *SPSS*) in `RStudio` eingelesen werden. Auch dies ist möglich.

Um einen Datensatz einlesen zu können, muss `RStudio` wissen, wo dieser Datensatz überhaupt liegt. Ein PC verfügt über viele Ordner und `RStudio` muss daher mitgeteilt werden, in welchen Ordner (Fachbegriff: *working directory*) `RStudio` „hineinschauen“ soll, um den Datensatz zu finden. Um zu wissen, auf welchen Ordner `RStudio` aktuell zugreift bzw. „in welchen Ordner `RStudio` aktuell hineinschaut“, wird der Befehl

```
getwd()
```

verwendet (Abbildung 21). Im Skript von Abbildung 21 ist zu sehen, dass der Befehl `getwd()` ausgeführt wurde, sodass im Output-Fenster der Dateipfad *"/Users/christiankissler"* ausgegeben wurde. Dieser Dateipfad entspricht vielleicht jedoch nicht dem Dateipfad, auf den `RStudio` zugreifen soll, da der Datensatz, welcher eingelesen werden soll, an einer anderen Stelle bzw. in einem anderen Ordner auf dem Computer zu finden ist. In diesem Fall ist es notwendig, den Dateipfad, auf den `RStudio` zugreift, zu ändern. Dies ist mit dem Befehl

```
setwd()
```

möglich. In Abbildung 22 ist zu sehen, dass mithilfe des Befehls `setwd()` die *working directory* geändert wurde, sodass die geänderte *working directory* anschließend mit `getwd()`

abgerufen und die Änderung auf Korrektheit überprüft werden konnte (Abbildung 22).

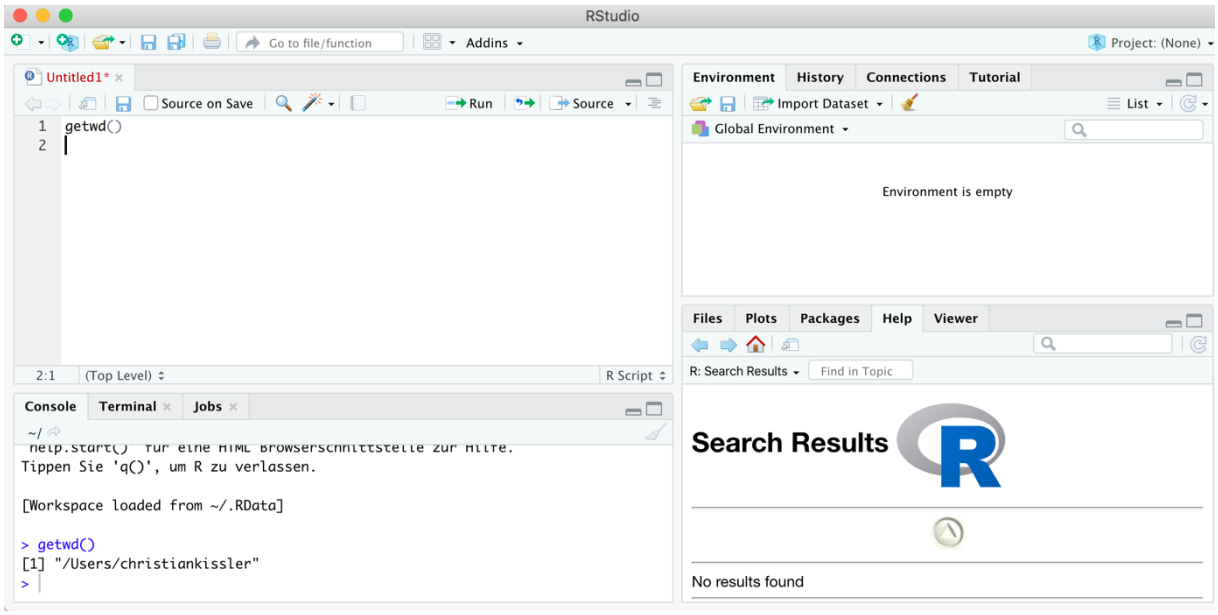

Abbildung 21: getwd()

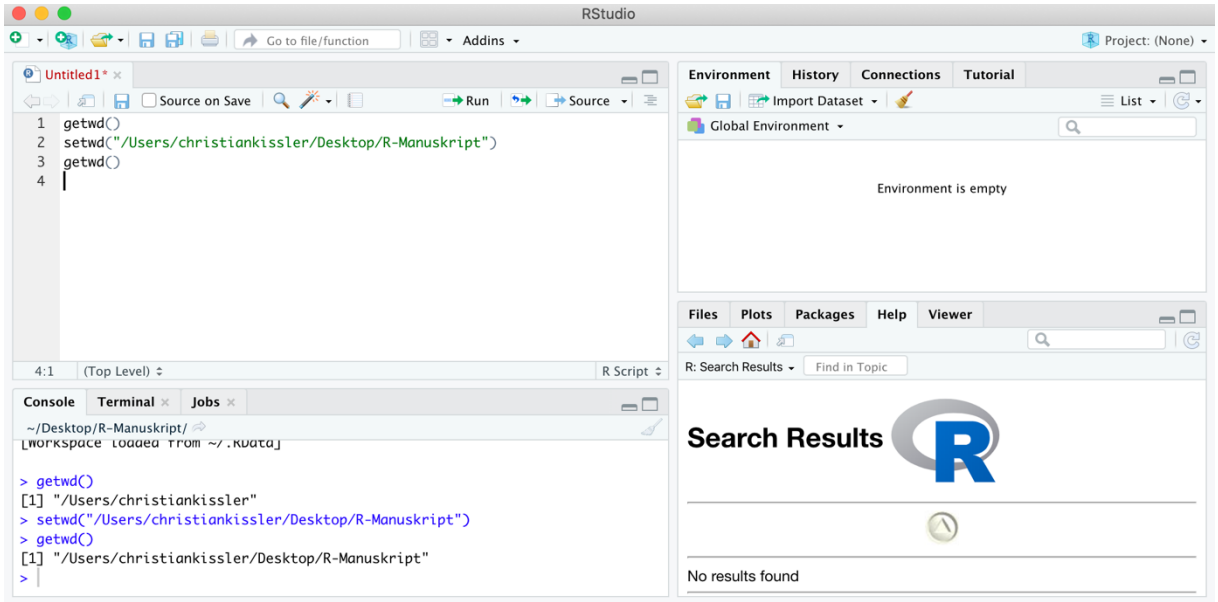

Abbildung 22: getwd() und setwd()

Wie Abbildung 22 zu entnehmen ist, wurde der Dateipfad erfolgreich geändert. Falls Sie nicht wissen, woher Sie den Dateipfad bekommen, den Sie brauchen, so wird Ihnen diesbezüglich in Abhängigkeit Ihres Betriebssystems (Windows oder MacOS) in den Kapiteln 2.9.1 bzw. 2.9.2 weitergeholfen.

Sobald unter Verwendung von `getwd()` und ggf. `setwd()` der korrekte Dateipfad ausgewählt wurde, kann der Datensatz eingelesen werden. In diesem Manuskript wird stets mit dem Datensatz *datensatz.xlsx* gearbeitet, der im Environment als *data* abgelegt werden soll.

Abbildung 23 ist zu entnehmen, wie der Datensatz eingelesen wird. Das genaue Vorgehen wird außerdem im Folgenden beschrieben. Der Datensatz *datensatz.xlsx* kann über den folgenden Link über die Plattform Eldorado heruntergeladen werden (Kißler, 2023a): https://eldorado.tu-dortmund.de/handle/2003/42178 oder http://dx.doi.org/10.17877/DE290R-24012

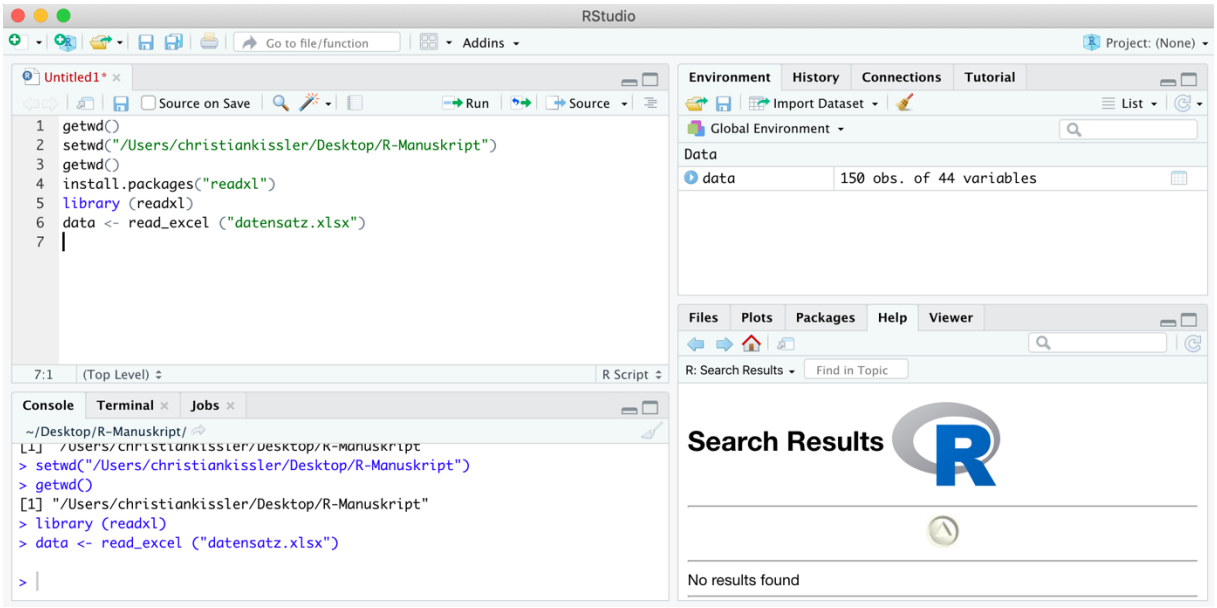

Abbildung 23: Einlesen eines Datensatzes

Nachdem mit `setwd()` der korrekte Dateipfad gesetzt und die Aktion mit `getwd()` überprüft wurde, wurde ein *Package* (*readxl*) installiert, das zum Einlesen des Datensatzes benötigt wird. Anschließend wurde über die Funktion `library()` das heruntergeladene *Package* aktiviert und anschließend die Funktion `read_excel()` aus dem *Package readxl* verwendet, um den Datensatz *datensatz.xlsx* in `RStudio` einzulesen und unter der Bezeichnung *data* im *Environment* zu hinterlegen. Die entsprechende Syntax ist Abbildung 23 zu entnehmen. Im *Environment* können Sie dem Ausdruck „*150 obs. of 44 variables*“ entnehmen, dass der Datensatz *data* 150 Proband:innen umfasst, zu welchen 44 Variablen vorhanden sind.

Zu beachten ist, dass im Environment Objekte bzw. Datensätze überschrieben werden können: Weiter oben im Manuskript hatten wir bereits unter Verwendung der `cbind()`-Funktion eine Matrix mit dem Namen *data* erstellt – sollten Sie diese Matrix noch in Ihrem *Environment* abgespeichert gehabt haben, so wurde diese nun durch den neuen Datensatz *data*, den Sie über die `read_excel()`-Funktion eingelesen haben, ersetzt bzw. überschrieben. Bei der Arbeit mit `RStudio` müssen Sie daher darauf achten, dass Sie eindeutige und einmalige Bezeichnungen für Objekte bzw. Datensätze, die Sie im *Environment* abspeichern, vergeben, da ansonsten eine Überschreibung im *Environment* erfolgt.

Um Dateien unterschiedlicher Dateiformate (z.B. Excel-Dateien, SPSS-Dateien, CSV-Dateien etc.) in `RStudio` einzulesen, sind auch unterschiedliche Befehle/ Funktionen notwendig, welche aus unterschiedlichen *Packages* stammen. Tabelle 4 zeigt solche Befehle zum Einlesen gängiger Dateiformate. Außerdem ist der Tabelle zu entnehmen, welches *Package* installiert sowie aktiviert sein muss, um den jeweiligen Befehl ausführen zu können. Sollten Sie andere Dateiformate einlesen wollen, so ist dies häufig möglich, indem Sie ein entsprechendes *Package* vorab installieren und dann den jeweiligen Befehl nutzen. In einem solchen Fall empfiehlt es sich, eine entsprechende Suchmaschine (z.B. *Google* oder *Bing*) zu verwenden, um einen geeigneten Befehl und ein geeignetes *Package* zu identifizieren. Meistens ist es zielführend, die Recherche auf Englisch durchzuführen, da dadurch in der Regel die Anzahl potenzieller Treffer (deutlich) erhöht werden kann und somit schneller eine entsprechende Lösung gefunden wird.

Tabelle 4: Einlesen von Datensätzen

Dateityp	Eingabe in R	Package
Textdatei (.txt)	`read.table()`	*base* (vorinstalliert)
CSV-Datei (Dezimaltrennzeichen = **Punkt**)	`read.csv()`	*utils* (meistens vorinstalliert)
CSV-Datei (Dezimaltrennzeichen = **Komma**)	`read.csv2()`	*utils* (meistens vorinstalliert)
Excel-Dateien (.xlsx)	`read_excel`	*readxl*
DTA-Dateien	`read.dta()`	*foreign* oder *haven*
SPSS-Dateien	`read.spss()`	*foreign* oder *haven*

Bei der Benutzung der Einlesefunktionen ist zu bedenken, dass die Syntax zum Einlesen eines Datensatzes je nach Befehl/ Funktion variieren kann. Beim Einlesen eines *txt*-Datensatzes mit dem Namen *dataset* müsste beispielsweise die folgende Syntax verwendet werden:

```
read.table („dataset.txt“, header=TRUE)
```

Der Zusatz *header=TRUE* ist dabei zwingend dann zu verwenden, wenn in der ersten Zeile eines Datensatzes im txt-Format abweichend zu den übrigen Zeilen keine Daten, sondern die Variablen-Namen stehen. Damit die Funktion `read.table()` den Aufbau einer solchen Datei dennoch korrekt versteht, ist der genannte Zusatz erforderlich. Sollten keine Variablennamen vorhanden sein, so ist die Verwendung des Zusatzes *header=TRUE* nicht notwendig.

Auf Variablen vom Datensatz *data* können Sie nun ebenfalls die `class()`-Funktion anwenden, um den Datentyp der jeweiligen Variable zu bestimmen. Mit der folgenden Funktion können Sie zum Beispiel den Datentyp der Variable *id* im Datensatz *data* erfragen:

```
class(data$id)
```

Wenn die Funktion `class(data$id)` ausgeführt wird, wird in der Konsole *character* ausgegeben, weil der Datentyp von der Variable *id* im Datensatz *data character* ist. Mit der Funktion `class()` wird der Datensatz *data* angesteuert und anschließend wird in dem Datensatz *data* die Variable *id* angesteuert. Vor dem *$*-Zeichen wird also der Datensatz benannt, auf den Bezug genommen wird, und hinter das *$*-Zeichen wird die Variable geschrieben, die im ausgewählten Datensatz angesteuert werden soll.

2.9.1 Datensätze unter Windows einlesen

Zum Einlesen eines Datensatzes in `RStudio` benötigen Sie den Dateipfad des jeweiligen Datensatzes. Auf Windows-Endgeräten gibt es prinzipiell zwei Wege, um den Dateipfad zum Einlesen eines Datensatzes herauszufinden. In diesem Unterkapitel werden Ihnen beide Wege aufgezeigt. Anzumerken ist an dieser Stelle, dass es in Abhängigkeit der Windows-Version, die auf Ihrem Endgerät installiert ist, möglicherweise (leichte) Abweichungen geben könnte.

Der erste Weg zum Einlesen eines Dateipfads führt über das Fenster „*Eigenschaften*“. Wenn Sie auf den Datensatz, der eingelesen werden soll, mit der rechten Maustaste klicken, öffnet sich ein Fenster. Hier wählen Sie *Eigenschaften* aus (Abbildung 24). Nun öffnet sich ein Fenster, wie es in Abbildung 25 dargestellt ist. Hier ist der Dateipfad zu finden, der nun in `RStudio` zu kopieren ist, um den jeweiligen Datensatz einlesen zu können. In Abbildung 25 ist der Dateipfad zur besseren Kenntlichmachung in blau hervorgehoben. Zunächst müssen Sie den Dateipfad mit der Maustaste auswählen. Unter Verwendung der Tastenkombination *Strg+c* können Sie den Dateipfad dann kopieren und in `RStudio` können Sie im *Skript* den Dateipfad anschließend mit der Tastenkombination *Strg+v* einfügen.

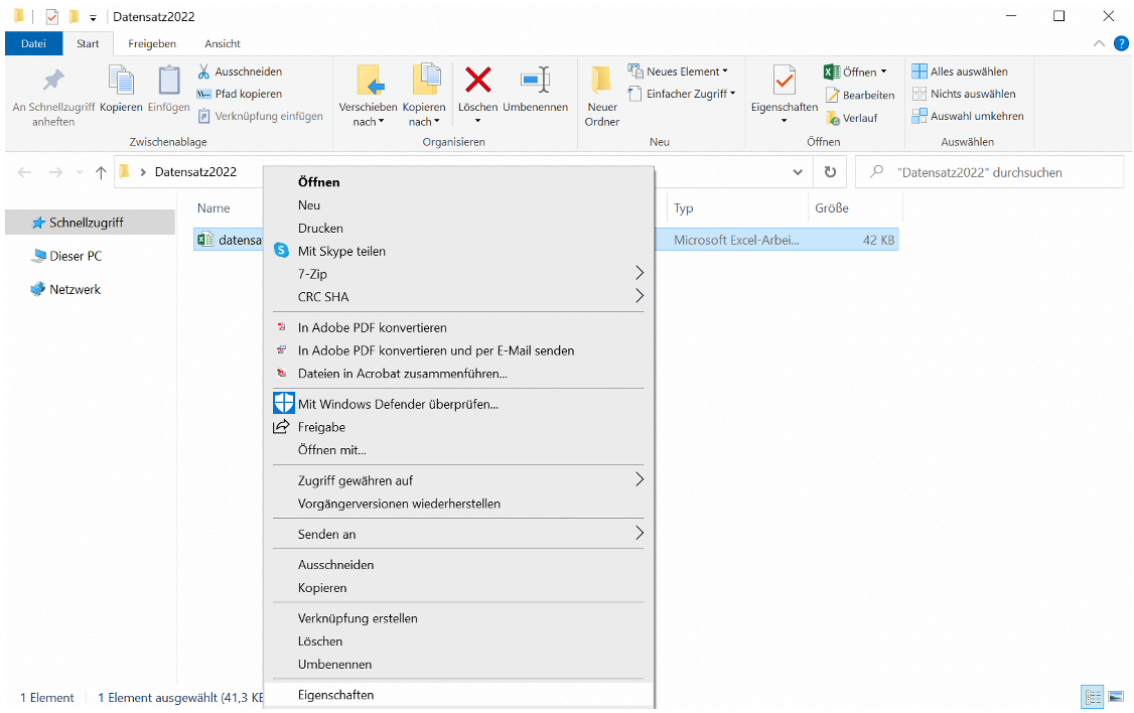

Abbildung 24: Das Eigenschaften-Fenster unter Windows

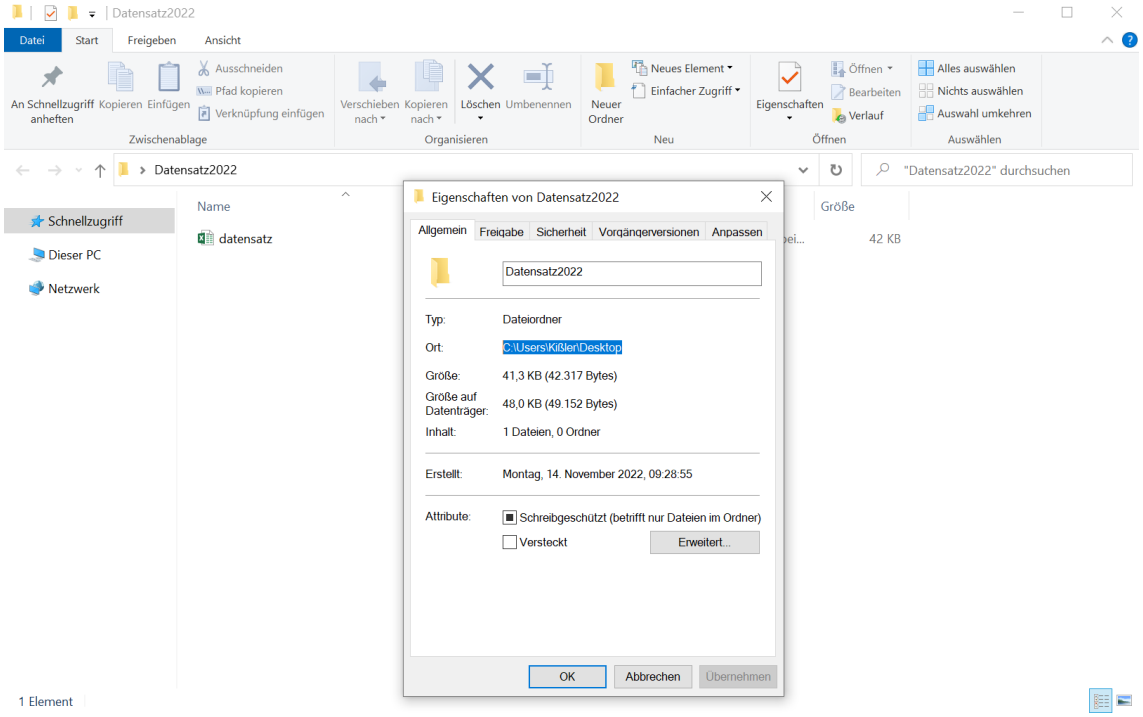

Abbildung 25: Der Dateipfad im Eigenschaftenfenster

Der zweite Weg zum Auffinden eines Dateipfads führt über die Ordnerleiste. Wenn Sie sich in dem Ordner befinden, in dem der Datensatz hinterlegt ist, können Sie den Dateipfad in der Regel oben in der Ordnerleiste finden (in Abbildung 26 wurde der entsprechende Dateipfad zur

Visualisierung rot umrandet). Damit Ihnen der Dateipfad vollständig angezeigt wird, müssen Sie in der Regel einmal mit der linken Maustaste in die Ordnerleiste klicken. Unter Verwendung der Tastenkombination *Strg+c* können Sie den Dateipfad kopieren und in `RStudio` können Sie den Dateipfad mit der Tastenkombination *Strg+v* einfügen.

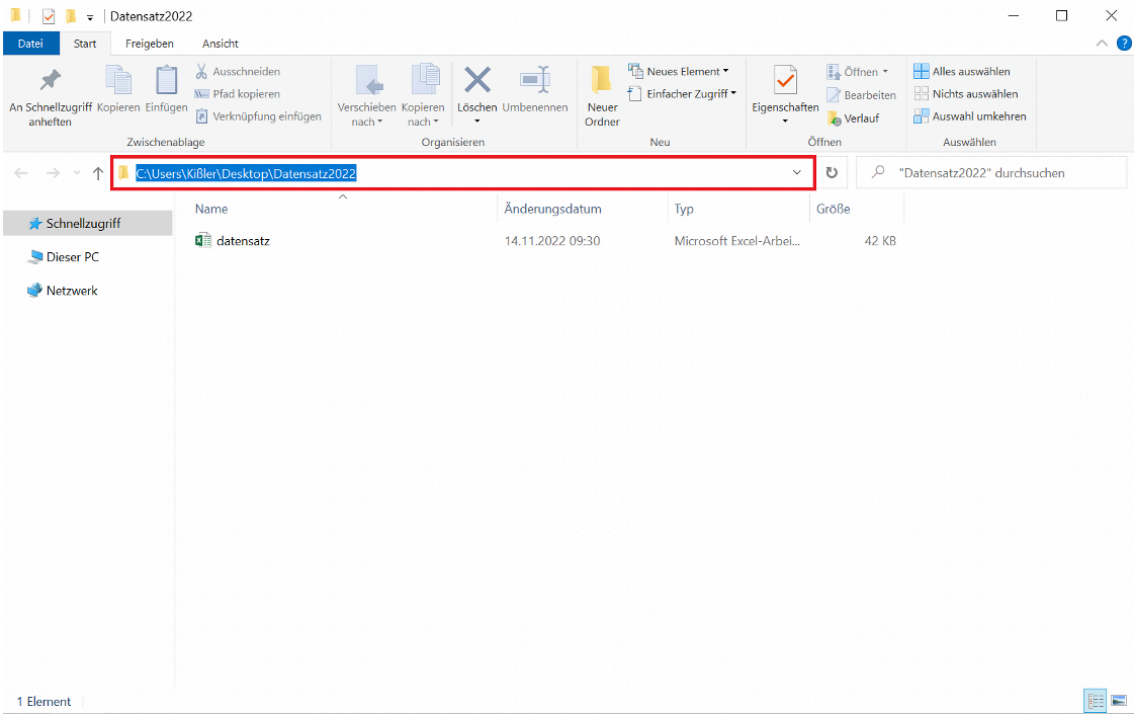

Abbildung 26: Der Dateipfad in der Ordnerleiste

2.9.2 Mac-Besonderheiten: Dateipfad kopieren

Auf einem Endgerät mit dem Betriebssystem MacOS (in der Regel bei Apple-Produkten) kann das Finden des Dateipfades, der zum Einlesen von Datensätzen in `RStudio` benötigt wird, zunächst misslingen. Im Folgenden wird dargestellt, wie Sie den Dateipfad in diesem Fall in aller Regel finden und kopieren können.

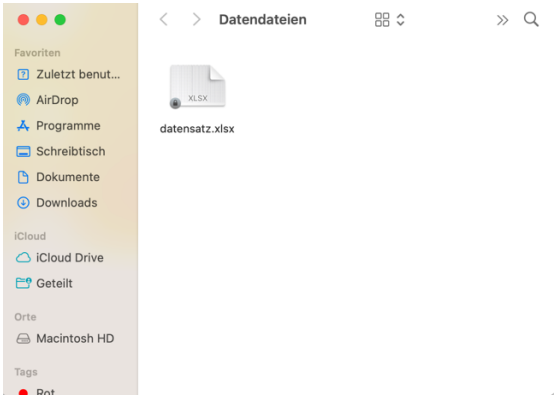

Abbildung 27: Datensatz-Datei im Ordner auf MacOS

Zuerst gehen Sie in den Ordner, in dem sich Ihr Datensatz (*datensatz.xlsx*) befindet. In diesem Fall befindet sich die Datei in einem Ordner, der zuvor erstellt und „*Datendateien*“ genannt wurde (Abbildung 27). Dies können Sie gerne ebenso handhaben.

Klicken Sie nun mit einem Doppelklick (mit zwei Fingern gleichzeitig) auf den Datensatz, den Sie in `RStudio` einlesen möchten. Es öffnet sich das folgende Fenster (Abbildung 28):

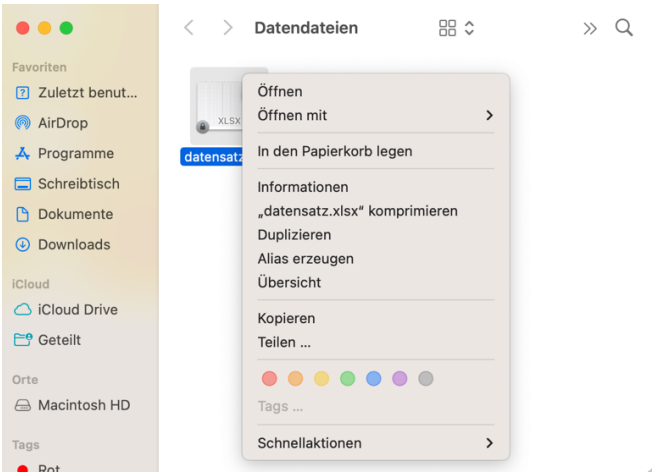

Abbildung 28: Dateipfad finden und kopieren

Sie sehen nun das Feld „*Kopieren*“. Wenn Sie jetzt auf Ihrer Tastatur die Taste „*option*“ drücken und diese Taste gedrückt halten, wird statt „*Kopieren*“ die folgende Option angezeigt: „*datensatz.xlsx als Pfadname kopieren*“. Klicken Sie nun auf „*datensatz.xlsx als Pfadname kopieren*“. Jetzt öffnen Sie wieder `RStudio` und können mit der Tastenkombination „*Command*“ und „*v*“ (gleichzeitig drücken) den Pfadnamen einfügen.

Achtung: Es kann sein, dass hinten am eigefügten Dateipfad der Zusatz „*datensatz.xlsx*“ steht. Diesen Zusatz müssen Sie löschen, sodass nur noch der Dateipfad angegeben ist.

2.10 Datensätze umformen & Daten manipulieren

In Forschungsprozessen kommt es regelmäßig vor, dass Sie Datensätze manipulieren müssen. Mit dem Ausdruck der *Datenmanipulation* ist jedoch weder ein wissenschaftlicher Betrug noch eine sonstige illegale Praxis zur Fälschung von Daten gemeint. Vielmehr sind unter dem Ausdruck der *Datenmanipulation* legale Operationen zu verstehen, mit denen Datensätze für anschließende Berechnungen aufbereitet werden.

So kann es beispielsweise notwendig werden, dass ein Teildatensatz des Gesamtdatensatzes benötigt wird, da Berechnungen nur mit einer Teilgruppe aller

Proband:innen durchgeführt werden sollen. Umgekehrt kann es auch notwendig werden, dass Datensätze, die sich beispielsweise in unterschiedlichen Datenerhebungsphasen ergeben haben, zu einem gemeinsamen Datensatz zusammengefügt werden müssen. Darüber hinaus müssen Variablen regelmäßig umbenannt werden, um den Datensatz beispielsweise übersichtlicher zu gestalten. Solche und weitere Möglichkeiten der Datenmanipulation werden in diesem Unterkapitel dargestellt.

Alle folgenden Datenmanipulationen werden an dem Datensatz ausgeführt, der bereits in Kapitel 2.9 eingelesen wurde. Es kann jedoch passieren, dass beim Einlesen eines Datensatzes in `RStudio` der Datentyp von Variablen nicht korrekt erkannt wurde oder aus anderen Gründen geändert werden muss. In Kapitel 2.8 wurde bereits aufgezeigt, dass mit der Funktion `class()` der Datentyp einer Variable abgefragt werden kann. Mit dem Befehl

```
as.factor(data$id)
```

können Sie den Datentyp der Variable *id* im Datensatz *data* zu *factor* ändern und mit der Funktion

```
as.numeric(data$id)
```

könnten Sie den Datentyp ebendieser Variable zum Datentyp *numeric* ändern. Denken Sie daran, dass Sie eine Syntax in der folgenden Form (mit Zuweisungspfeil) benötigen, insofern Sie die Veränderung des Datentyps der Variable *id* (in diesem Fall von *character* zu *factor*) im Datensatz *data* speichern möchten:

```
data$id <- as.factor(data$id)
```

Auch wenn es prinzipiell/ theoretisch möglich ist, die Variable *id* im Datensatz *data* als *numeric* zu hinterlegen, so sollte die Variable *id* in der Praxis als *character* oder *factor* gehandhabt werden, da es sich bei diesen Nummern nicht um Zahlen, sondern um anonymisierte Namen handelt. Mit der Funktion `is.factor()` können Sie abfragen, ob eine Variable vom Datentyp *factor* ist; die Antwort (*TRUE* oder *FALSE*) wird dann in der Konsole ausgegeben. Insofern abgefragt werden soll, ob die Variable *id* im Datensatz *data* den Datentyp *factor* hat, wäre also der folgende Befehl zu verwenden:

```
is.factor(data$id)
```

Die *Packages tidyverse* und *dplyr* (Wickham et al., 2023) bieten zahlreiche Funktionen, die zur Datenmanipulation geeignet/ notwendig sind. Tabelle 5 bietet eine Übersicht von einigen solcher Funktionen.

Tabelle 5: Funktionen zur Datenmanipulation (u.a. aus den Packages tidyverse und dplyr)

Befehl	Funktion
`filter()`	Auswahl einer bestimmten Teilmenge von Fällen (= Zeilen, i.d.R. Personen)
`select()`	Auswahl einer Teilmenge von Variablen (= Spalten)
`arrange()`	Neuanordnung eines Datensatzes (z.B. Sortieren nach einem Merkmal)
`mutate()`	Transformierung und Neuerzeugung von Variablen auf Basis bestehender Variablen
`names()`	(Um-)Benennung von Variablen
`recode()`	Umkodierung von Variablenausprägungen

Im Folgenden wird gezeigt, wie die Funktion `filter()` funktioniert, indem wir einen Teildatensatzes des Datensatzes *data* erstellen. Dieser Teildatensatz soll nur weibliche Personen (Ausprägung *1* bei der Variable *geschlecht*) umfassen und als Datensatz mit dem Namen *data_f* im Environment gespeichert werden. Abbildung 29 ist die Verwendung der `filter()`-Funktion sowie das Ergebnis der Ausführung des Befehls zu entnehmen: Der Datensatz *data_f*, der nun nur die Daten der weiblichen Personen beinhält, umfasst 77 Personen (77 obs.) und alle 44 Variablen. Der Code, der in Abbildung 29 genutzt wurde, lautet:

```
data_f  <- filter (data, geschlecht == „1“)
```

Dieser Code diente dazu, den Teildatensatz *data_f* zu erstellen und ist folgendermaßen zu verstehen: Links vom Zuweisungspfeil (<-) steht die Bezeichnung, die der Teildatensatz tragen soll (*data_f*). Rechts vom Zuweisungspfeil ist die Funktion `filter()` mit ihren Spezifikationen in der Klammer zu finden: Zunächst muss der Datensatz benannt werden, der durch die `filter()`-Funktion angesteuert werden soll (*data*); es folgt ein Komma und die Bezeichnung der Variable im angesteuerten Datensatz (in diesem Fall heißt die Variable *geschlecht*). Hinter diese Variable werden zwei Gleichheitszeichen geschrieben und es folgt die Ausprägung, welche definiert, wie die `filter()`-Funktion durchgeführt werden soll (in

diesem Fall *1* für weiblich): Nur Personen, welche diese Ausprägung (also: *1*) aufweisen, werden aus dem Ursprungsdatensatz *data* herausgefiltert und in den Datensatz *data_f* übernommen. Wichtig ist in diesem Kontext zu berücksichtigen, dass in diesem Fall die Ausprägung (*1*) in Anführungszeichen zu schreiben ist, da der Datentyp der Variable *geschlecht character* ist.

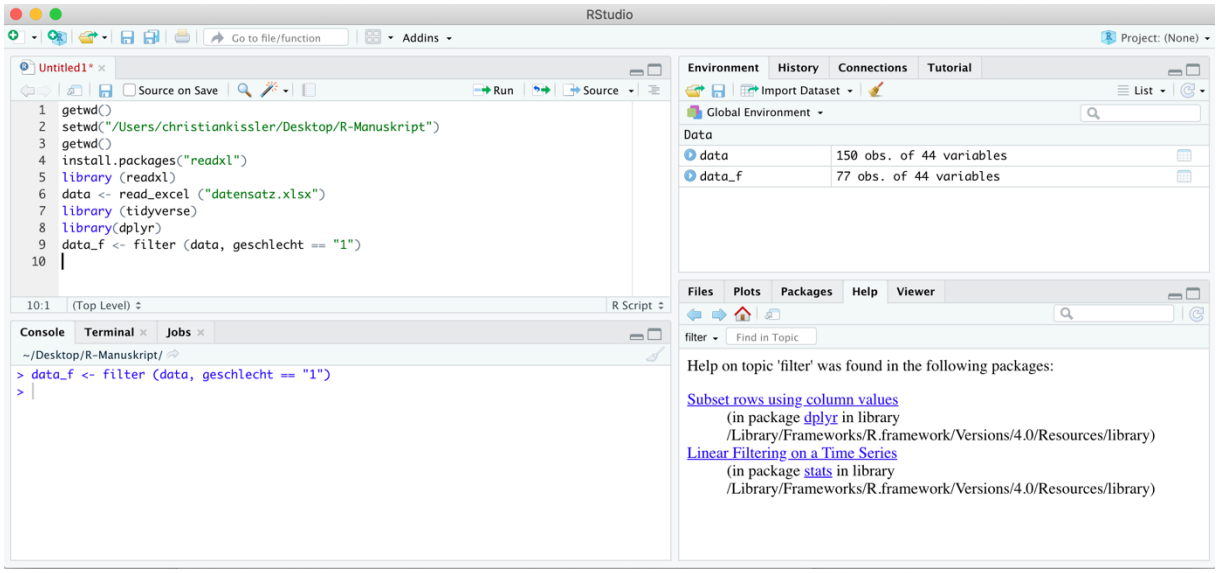

Abbildung 29: Verwendung der filter()-Funktion

Darüber hinaus ist es möglich, mehr als nur eine Bedingung zur Erstellung eines Teildatensatzes zu berücksichtigen (im vorherigen Beispiel wurde nur die Variable *geschlecht* berücksichtigt). So können auch zwei oder mehr Bedingungen genannt werden (z.B.: die Probanden sollen männlich sein **und** ihr *IQ* soll gleichzeitig höher als 115 sein). Die Syntax für die Erstellung dieses Teildatensatzes, der *data_sub* genannt wird, ist in Abbildung 30 zu sehen und lautet:

```
data_sub <- filter (data, geschlecht == „0“, IQ > 115)
```

Prinzipiell ist die Syntax aus Abbildung 30 mit der Syntax aus Abbildung 29 identisch: Zu beachten ist allerdings, dass die einzelnen Variablen zusammen mit ihren jeweiligen Bedingungen mit Kommata abgegrenzt werden müssen, wenn bei der Verwendung der `filter()`-Funktion mehrere Bedingungen zur Erstellung eines Teildatensatzes beachtet werden sollen: So steht zwischen dem Ausdruck *geschlecht == „0“* und *IQ >115* ein Komma.

Wenn nun der Datensatz *data_sub* mit dem Befehl `View(data_sub)` visualisiert wird und die Variable *Lieblingsfarbe* betrachtet wird, so fällt auf, dass die Reihenfolge, in welcher die Probanden im Datensatz vorliegen, in keinem Zusammenhang zur Reihenfolge der Ausprägungen der Lieblingsfarbe stehen (Abbildung 31): Die ersten beiden Personen im

Datensatz haben die Lieblingsfarbe rot, die dritte Person die Lieblingsfarbe blau, die vierte Person die Lieblingsfarbe rot etc. – eine sinnvolle Ordnung ist hier nicht zu erkennen.

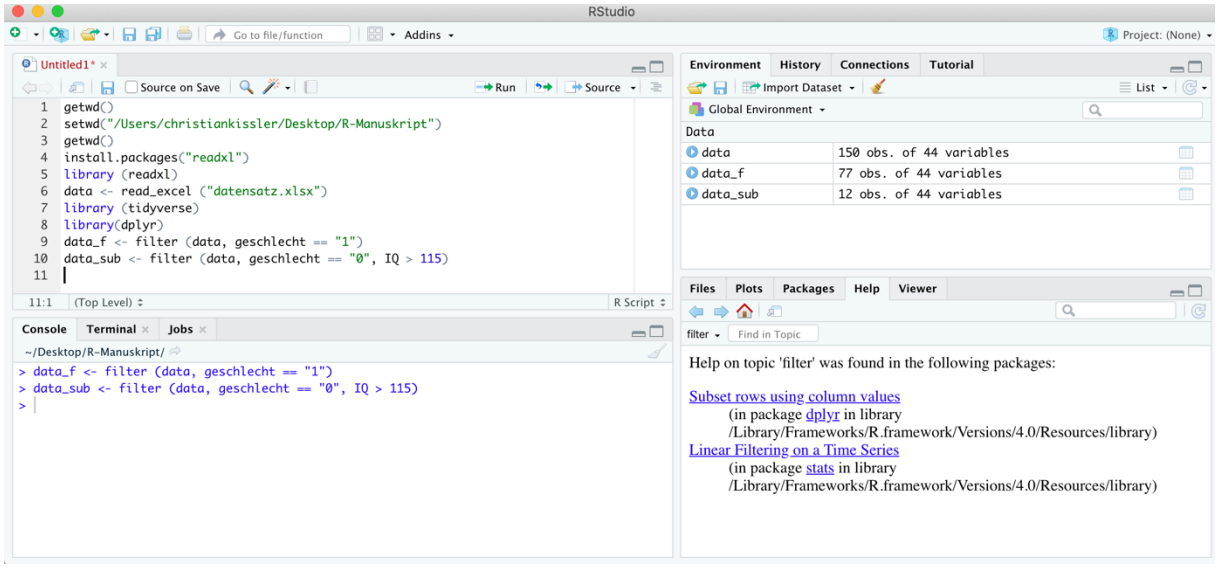

Abbildung 30: Verwendung der filter()-Funktion II

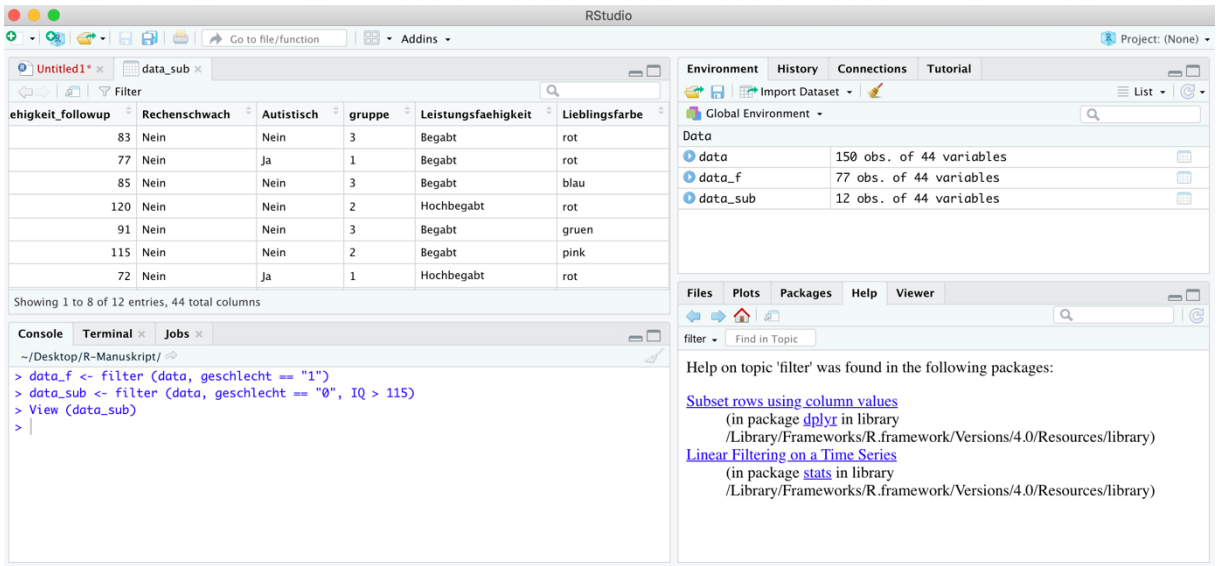

Abbildung 31: Reihenfolge von Daten im Datensatz

Mit der Funktion `arrange()` kann die Reihenfolge der Personen neu geordnet werden. So soll nun der Befehl

```
arrange(data_sub, Lieblingsfarbe)
```

angewendet werden, um die Probanden im Datensatz *data_sub* so neu zu ordnen, dass sie anschließend hinsichtlich ihrer Lieblingsfarbe geordnet sind. In der Klammer steht dabei zunächst der Datensatz, auf den sich die Funktion bezieht (*data_sub*); es folgt ein Komma und

danach die Variable (*Lieblingsfarbe*), nach der die Daten im Datensatz neu geordnet werden sollen. Dieser neu geordnete Datensatz wird unter dem Namen *data_sub_arrange* abgespeichert (Abbildung 32).

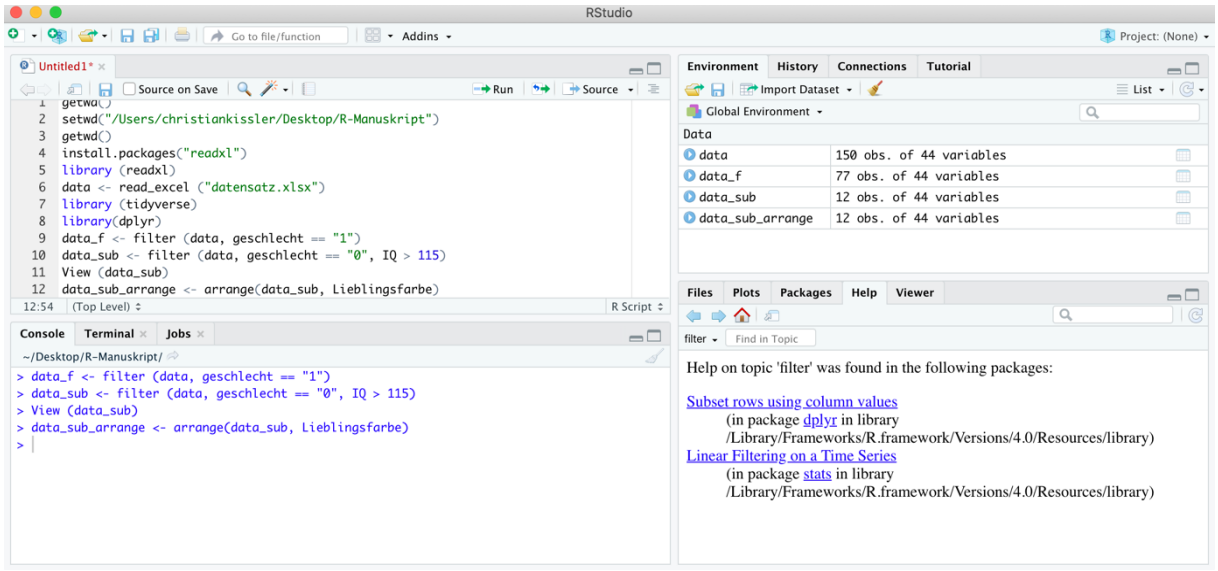

Abbildung 32: Die arrange()-Funktion

Mit dem Befehl `View(data_sub_arrange)` kann nun der neue Datensatz angezeigt werden (analog zu Abbildung 31). Nun werden die Personen im Datensatz entsprechend ihrer Lieblingsfarbe (in alphabetischer Reihenfolge der Lieblingsfarbe) geordnet sein: Ganz oben stehen alle Personen mit der einen Lieblingsfarbe, dann weiter unten alle Personen mit einer anderen Lieblingsfarbe etc., weil die Personen nach Lieblingsfarben sortiert wurden.

Der Datensatz, den wir verwenden, besteht aus Zeilen (die Personen/ Proband:innen) und Spalten (die Variablen). Mit der Funktion `filter()` war es möglich, mit Blick auf definierte Kriterien (z.B. das Geschlecht sollte weiblich sein) spezifische Zeilen (also Personen) auszuwählen und so einen neuen Datensatz zu erzeugen, der nur Personen mit spezifischen Merkmalen umfasst. Dabei wurden jedoch für eine jede Person stets alle Spalten (also alle Variablen) in den neuen Datensatz übernommen. Die Funktion `select()` stellt quasi das Gegenstück zu `filter()` dar: Mit der Funktion `select()` werden spezifische Spalten für alle Zeilen bzw. Personen ausgewählt, sodass auf diese Weise ein neuer Datensatz erzeugt werden kann, der zwar alle Proband:innen, aber nur ausgewählte Variablen umfasst. In Abbildung 33 ist zu sehen, wie mit der *Code*-Zeile

```
data_select <- select (data, id, geschlecht, IQ)
```

aus dem Datensatz *data* ein Teildatensatz mit dem Namen *data_select* erzeugt werden kann,

welcher die Variablen *id*, *geschlecht* und *IQ* umfasst. In Abbildung 33 ist nach dem Ausführen der `select()`-Funktion im Environment zu sehen, dass dieser neue Datensatz mit dem Namen *data_select* zwar alle 150 Proband:innen, aber anstelle der 44 Variablen des Gesamtdatensatzes *data* nur 3 Variablen umfasst.

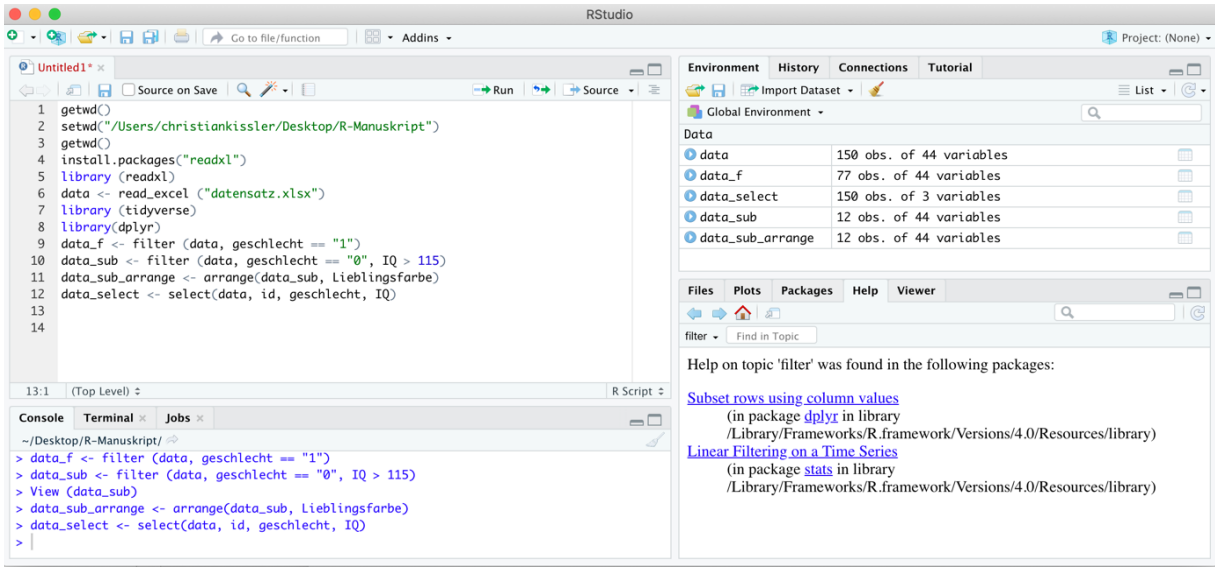

Abbildung 33: select()

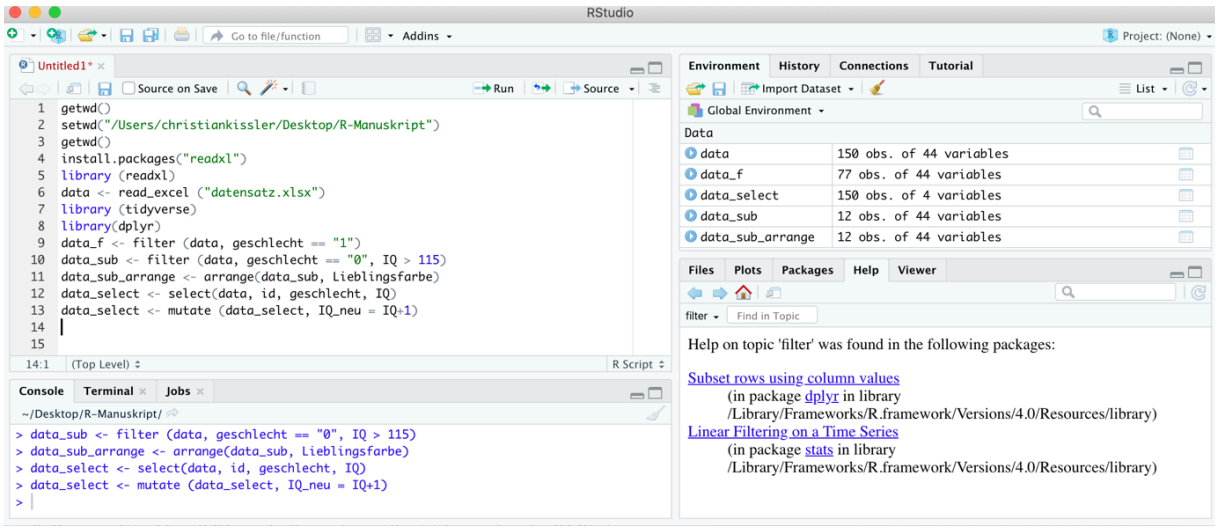

Abbildung 34: mutate() I

Mit der Funktion `mutate()` ist es möglich, Ausprägungen von Variablen zu verändern und neue Variablen auf der Basis von bestehenden Variablen zu erzeugen. Nun soll zur Demonstration der `mutate()`-Funktion im Datensatz *data_select* eine neue Variable mit dem Namen *IQ_neu* angelegt werden. Alle Proband:innen im Datensatz *data_select* sollen Werte für diese neue Variable *IQ_neu* haben und der Wert der Variable *IQ_neu* soll für jede Person

genau um 1 größer sein als der bisherige Wert mit Blick auf die Variable *IQ* (also: *IQ* + 1 = *IQ_neu*). Diese Berechnung ergibt inhaltlich in der Praxis zwar in der Regel nicht viel Sinn, ist zur Veranschaulichung des allgemeinen Prinzips der Datenmanipulation aber zielführend und strukturell leicht nachzuvollziehen, weshalb an dieser Stelle dieses Beispiel gewählt wurde. Dazu wird der folgende *Code* verwendet:

```
data_select <- mutate (data_select, IQ_neu = IQ+1)
```

Dieser *Code* sowie das Resultat der Ausführung der Funktion ist in Abbildung 34 zu sehen.

In Zeile 13 der Syntax von Abbildung 34 steht der *Code*, der benötigt wird, um das auszuführen, was zuvor als Ziel beschrieben wurde. Links vom Zuweisungspfeil steht der Ausdruck *data_select*, weil in diesem Datensatz die neu berechnete Variable hinterlegt werden soll. Dies bedeutet in diesem Kontext, dass im Datensatz *data_select* eine neue Variable *IQ_neu* angelegt werden soll, in welcher das Ergebnis nach Anwendung der `mutate()`-Funktion abgespeichert wird. Rechts vom Zuweisungspfeil steht in Zeile 13 der Befehl, der ausgeführt werden soll: In der Klammer der `mutate()`-Funktion steht zunächst der Datensatz, auf den sich die Funktion bezieht *(data_select*). Nach einem Komma folgt die Vorschrift, nach der eine neue Variable erzeugt werden soll: Die neue Variable soll *IQ_neu* heißen und sie soll entstehen, indem der *IQ*-Wert um eins erhöht wird (*+1*). Im Environment ist zu sehen, dass der Datensatz *data_select* nach der erfolgreichen Ausführung der Zeile nicht mehr nur 3 Variablen, sondern 4 Variablen umfasst (Abbildung 33 im Vergleich zu Abbildung 34). Mit der Funktion `View(data_select)` kann der Datensatz eingesehen und das, was durch die `mutate()`-Funktion bewirkt wurde, überprüft werden.

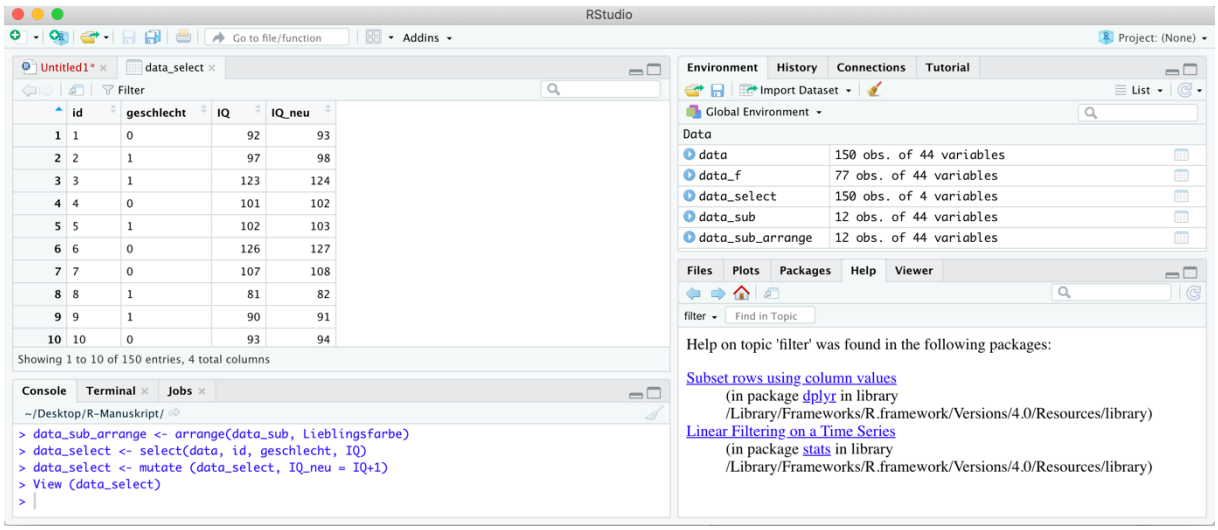

Abbildung 35: mutate() II

In Abbildung 35 ist zu erkennen, dass der Wert der neuen Variable *IQ_neu* bei jeder Person stets um 1 höher ist als der jeweilige Wert der Variable *IQ*. Somit hat die Umformung funktioniert. Prinzipiell ist eine solche Umformung aber auch ohne die `mutate()`-Funktion möglich. In Abbildung 36 ist eine alternative Berechnungsmöglichkeit dargestellt.

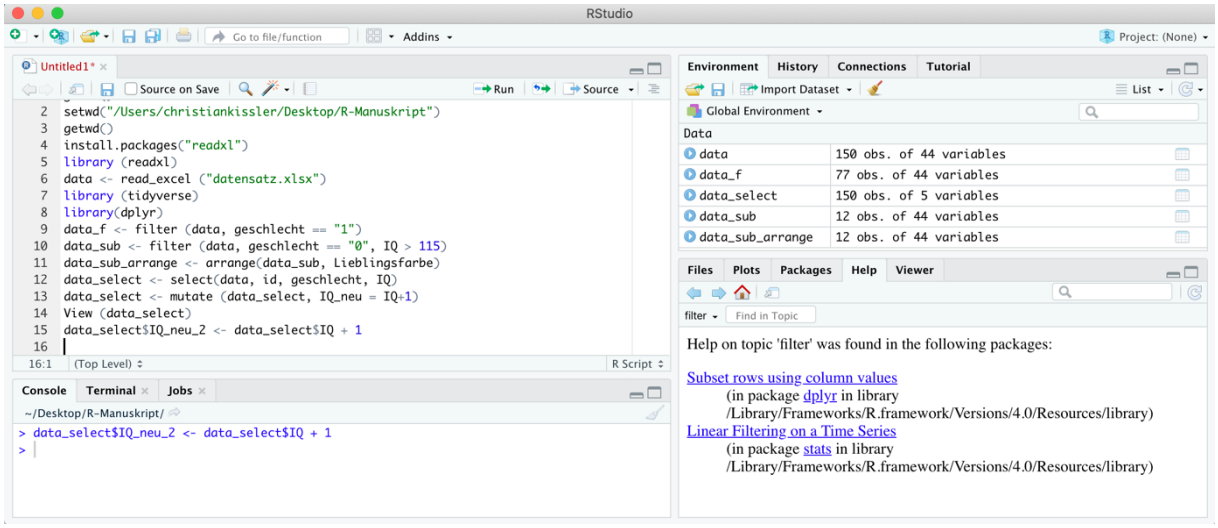

Abbildung 36: Alternative zu mutate()

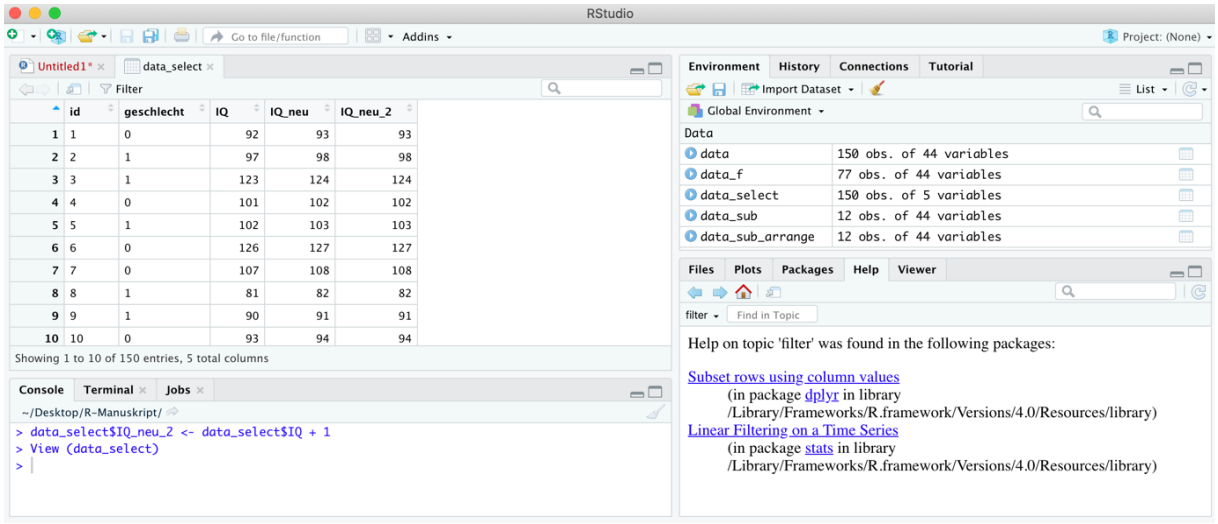

Abbildung 37: Ergebnis des alternativen Befehls zu mutate()

In Zeile 15 von Abbildung 36 ist in dem *Code*

```
data_select$IQ_neu_2 <- data_select$IQ + 1
```

rechts vom Zuweisungspfeil angegeben, wie die Berechnung der neuen Variablenwerte erfolgen soll: Der *IQ*-Wert aus dem Datensatz *data_select* soll für jede Person um 1 erhöht werden. Links vom Zuweisungspfeil steht dann, dass diese Rechenergebnisse unter der Variable *IQ_neu_2* im Datensatz *data_select* abgespeichert werden sollen

(*data_select$IQ_neu2*). Da die Variable *IQ_neu_2* zuvor nicht existierte, wird sie beim Ausführen der Zeile neu angelegt. Im *Environment* ist zu sehen, dass der Datensatz *data_select* nach der Ausführung des Befehls nun 5 Variablen umfasst (Abbildung 36).

Wenn der Datensatz nun über `View(data_select)` eingesehen wird (Abbildung 37), so fällt auf, dass die Werte der Variable *IQ_neu* und die Werte der Variable *IQ_neu_2* vollkommen identisch sind. Es wird also deutlich, dass dieser Befehl, der in Abbildung 36 dargestellt ist, zum gleichen Ergebnis wie die `mutate()`-Funktion führt, die davor dargestellt wurde.

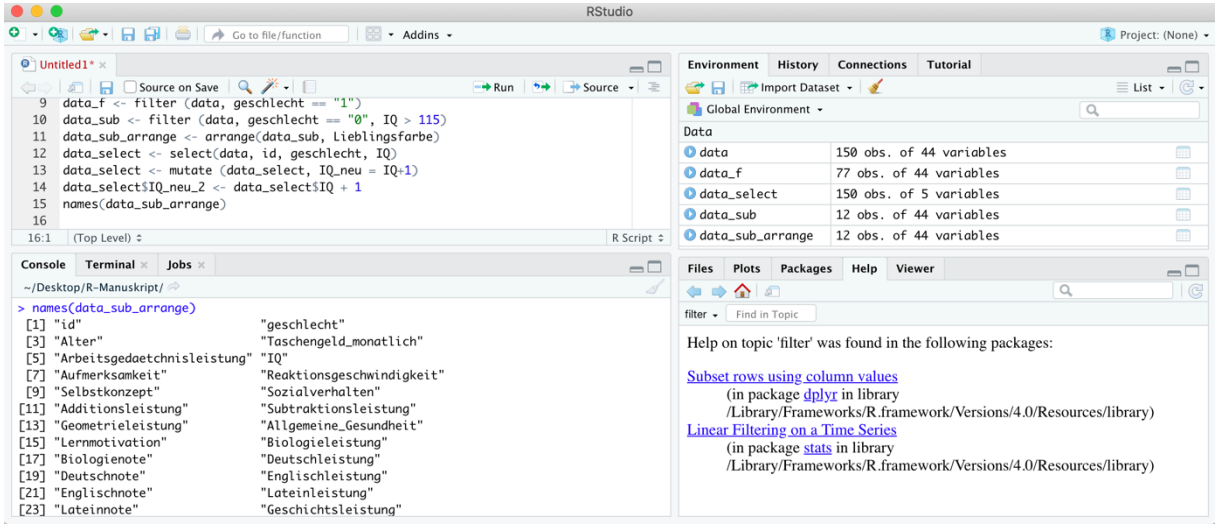

Abbildung 38: names() zum Anzeigen der Namen eines Datensatzes

Nicht selten ist es notwendig bzw. gewünscht, dass Variablen umbenannt werden. Über die Funktion `names()` können sowohl die Namen eines Datensatzes angezeigt werden, als auch die Namen von Variablen geändert werden. In Abbildung 38 ist in der Konsole eine Aufzählung aller Variablen-Namen des Datensatzes *data_sub_arrange* zu sehen, da die Funktion

```
names(data_sub_arrange)
```

ausgeführt wurde. Nun könnte es als störend empfunden werden, dass der Name der Variable *geschlecht* in den Datensätzen stets kleingeschrieben ist. Daher soll im Datensatz *data_select* der Variablenname der Variable *geschlecht* zu *Geschlecht* geändert werden. In Abbildung 39 ist zu sehen, wie eine solche Umbenennung prinzipiell erfolgen kann:

```
names (data_select) [2] <- „Geschlecht“
```

Rechts vom Zuweisungspfeil steht in Anführungszeichen der neue Variablennamen (*Geschlecht*), links vom Zuweisungspfeil steht zunächst die Funktion `names()` und in den

Funktionsklammern steht der Datensatz, auf den Bezug genommen wird. Dahinter steht in eckigen Klammern, an welcher Stelle im Datensatz die Variable steht, die umbenannt werden soll (die Variable *geschlecht* steht an zweiter Stelle im Datensatz *data_select*).

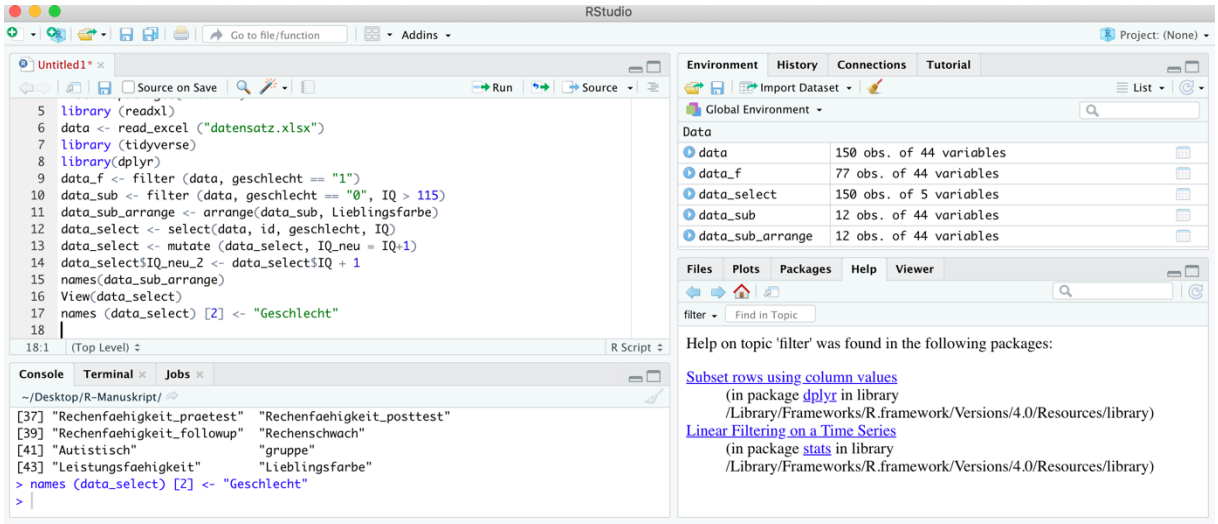

Abbildung 39: Änderung eines Variablennamens mit names()

Sobald der Befehl in Zeile 17 von Abbildung 39 ausgeführt wurde, kann mit der `View()`-Funktion überprüft werden, ob die Änderung erfolgreich war (Abbildung 40): Der Name der Variable *geschlecht* wurde korrekt in *Geschlecht* umbenannt.

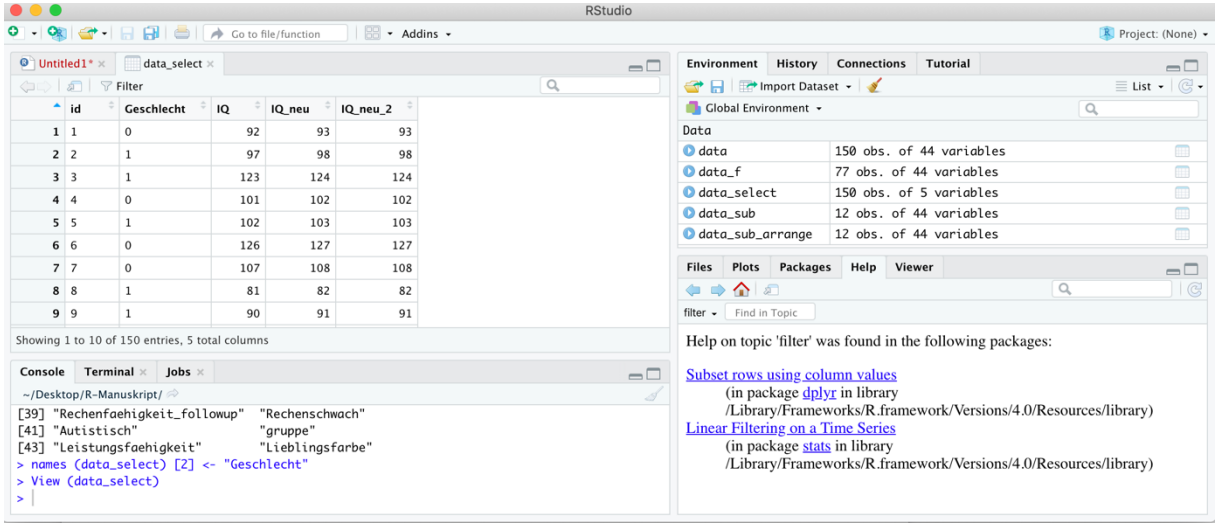

Abbildung 40: Überprüfung der Umbenennung eines Variablennamens

Manchmal kann es darüber hinaus gewünscht oder notwendig sein, dass die Ausprägungen einer Variable umkodiert bzw. umbenannt werden. Bisher hatten wir in unseren Datensätzen männlich mit *0* und weiblich mit *1* kodiert. Diese Bedeutung der Kodierung ist für Außenstehende nicht intuitiv verständlich und es kann außerdem geschehen, dass die

Bedeutung der Kodierung vergessen wird, wenn eine längere Zeit nicht mit dem Datensatz gearbeitet wird. Daher soll im Folgenden im Datensatz *data_select* die Ausprägung *0* für männlich zu *m* und die Ausprägung *1* für weiblich zu *w* umkodiert werden. Der Code, der zu dieser Rekodierung notwendig ist, ist in Abbildung 41 (Zeile 18) dargestellt:

```
data_select$Geschlecht <- recode(data_select$Geschlecht,
      „0“ = „m“, „1“ = „w“)
```

In diesem Code steht rechts vom Zuweisungspfeil die `recode()`-Funktion und in den Klammern steht zunächst die Variable, von welcher die Ausprägungen rekodiert werden sollen. Danach werden (jeweils mit Kommata abgetrennt) die Ausprägungen genannt, die geändert werden sollen: *0* soll zu *m* für männlich und *1* zu *w* für weiblich rekodiert werden.

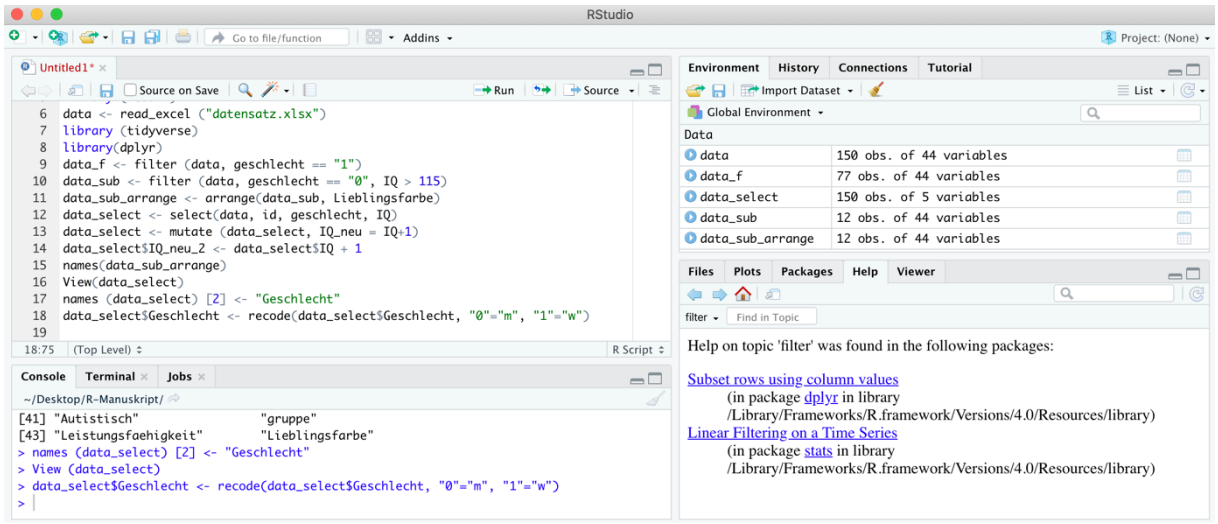

Abbildung 41: recode()

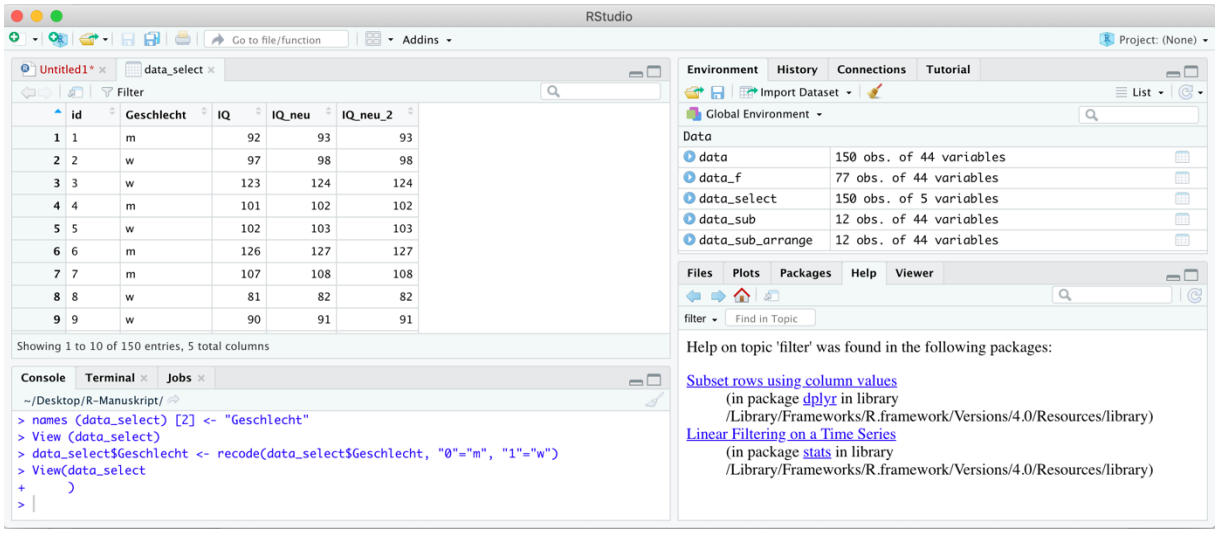

Abbildung 42: Ergebnis der Rekodierung

In Abbildung 42 ist das Ergebnis der Rekodierung dargestellt: Die Variable Geschlecht verfügt im Datensatz *data_select* nun über die Ausprägungen *m* und *w*.

Bisher wurden aus einem großen Datensatz stets kleinere Teildatensätze erzeugt, um anschließend mit diesen zu arbeiten. Es ist aber auch möglich, mehrere Datensätze zusammenzufügen. Abbildung 43 verdeutlicht dabei das Prinzip, dass bzw. wie zwei Datensätze (ein rot und ein blau eingefärbter Datensatz) mit der Funktion `bind_rows()` zusammengefügt wurden.

ID	Var1	Var2	Var3
1	19	3	2
2	13	6	1
3	24	1	2
4	11	6	0

Abbildung 43: Prinzipielle Funktionsweise der Funktion bind_rows()

Auf diese Weise sollen nun die Datensätze *data_f* und *data_sub* zusammengefügt werden. Dieser neue Datensatz soll den Namen *data_merged* tragen.

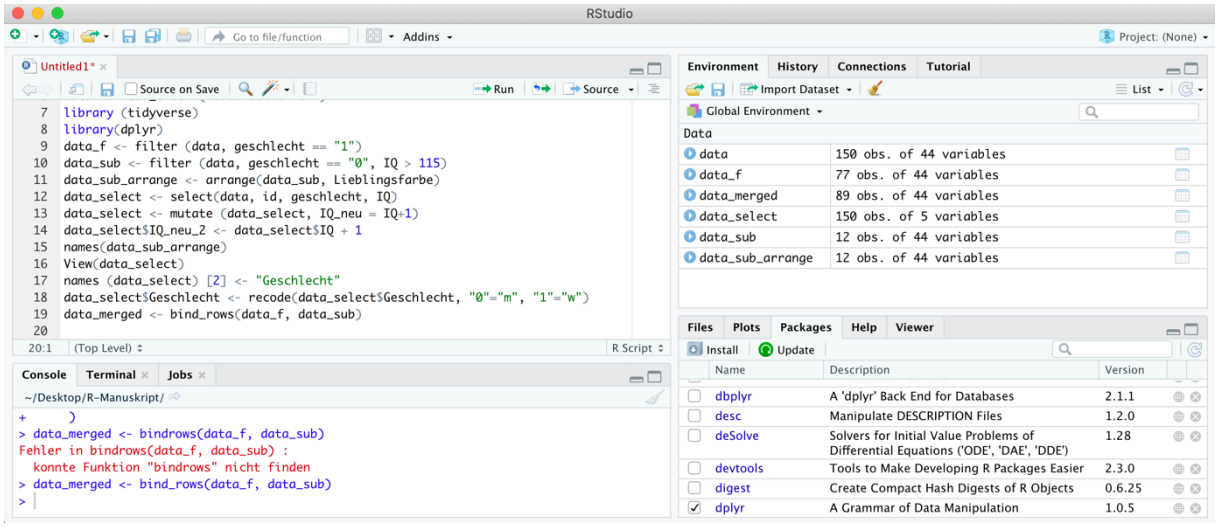

Abbildung 44: Die Funktion bind_rows()

Abbildung 44 ist die zugehörige Syntax sowie das Ergebnis der ausgeführten Syntax zu entnehmen:

```
data_merged <- bind_rows(data_f, data_sub)
```

Links vom Zuweisungspfeil steht die Bezeichnung des neuen Datensatzes, der erstellt werden soll (*data_merged*). Rechts vom Zuweisungspfeil steht die Funktion `bind_rows()` und in den Klammern werden – getrennt durch Kommata – die Datensätze benannt, die zusammengefügt werden sollen. Im Environment von Abbildung 44 ist zu sehen, dass der Datensatz *data_merged* 89 Personen umfasst – also genau die Summe der Personenanzahl aus den Datensätzen *data_f* (77 Personen) und *data_sub* (12 Personen). Also ist das Zusammenfügen der Datensätze geglückt.

Abbildung 45 verdeutlich das Prinzip, wie zwei Datensätze (ein rot und ein blau eingefärbter Datensatz) mit der Funktion `bind_cols()` zusammengefügt werden können.

ID	Var1	Var2	Var3
1	19	3	2
2	13	6	1
3	24	1	2
4	11	6	0

Abbildung 45: Prinzipielle Funktionsweise der Funktion bind_cols()

Im Folgenden sollen nun auf diese Weise die Datensätze *data* und *data_select* mithilfe des folgenden Codes zusammengeführt werden:

```
data_cols <- bind_cols (data, data_select)
```

Dieser neue Datensatz soll den Namen *data_cols* tragen. Abbildung 46 ist die zugehörige Syntax sowie das Ergebnis der ausgeführten Syntax zu entnehmen. Links vom Zuweisungspfeil steht die Bezeichnung des neuen Datensatzes, der erstellt werden soll (*data_cols*). Rechts vom Zuweisungspfeil steht die Funktion `bind_cols()` und in den Klammern werden – getrennt durch Kommata – die Datensätze benannt, die zusammengefügt werden sollen. Im *Environment* von Abbildung 46 ist zu sehen, dass der Datensatz *data_cols* 150 Personen umfasst; auch die Datensätze *data* und *data_select* umfassen stets 150 Personen. Somit wurden die Datensätze nicht untereinander, sondern nebeneinander zusammengefügt (wie in Abbildung 45 dargestellt). Der neue Datensatz *data_cols* verfügt deshalb über 49 Variablen – also genau die Summe aus der Variablenanzahl aus den Datensätzen *data* (44) und *data_select* (5). Dies

bedeutet, dass das Zusammenfügen der Datensätze geglückt ist. Das Resultat kann nun auch über die `View()`-Funktion eingesehen werden.

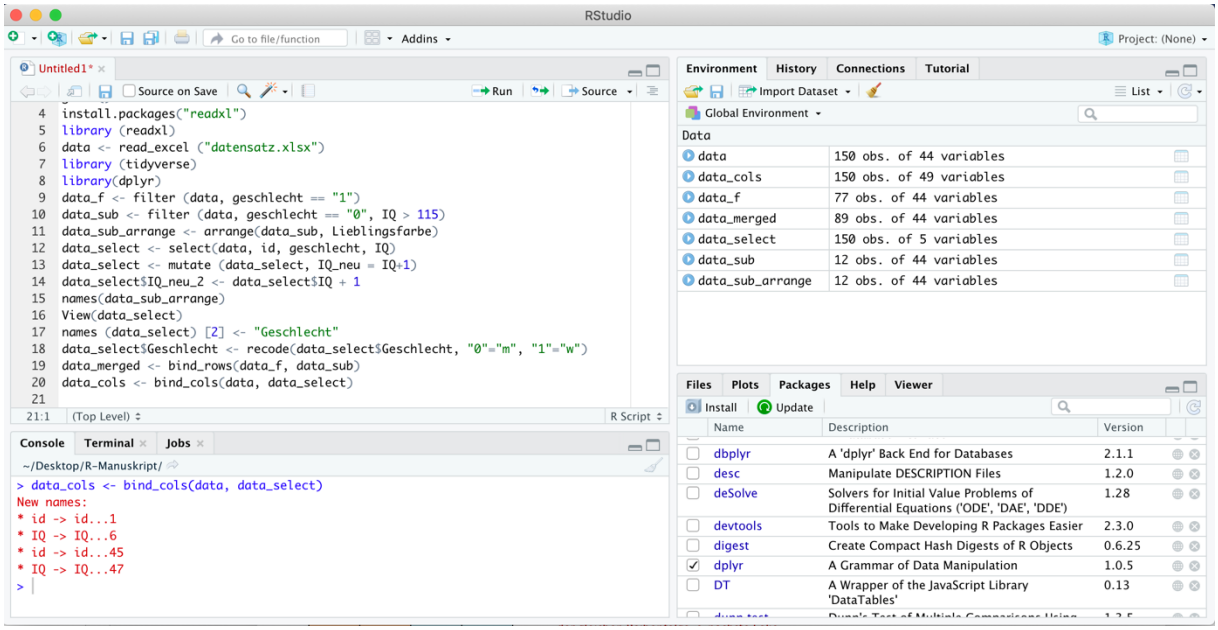

Abbildung 46: Die Funktion bind_cols()

In der Konsole von Abbildung 46 fallen jedoch in rot markiert spezifische Anmerkungen auf, die ausgegeben werden, wenn der Befehl in Zeile 20 der Abbildung 46 ausgeführt wird. Dies ist damit zu erklären, dass in dem Datensatz *data* und in dem Datensatz *data_select* jeweils einige Variablen vorkommen, welche den gleichen Namen tragen. Da Variablennamen eindeutig und einmalig pro Datensatz vergeben werden müssen, hat `R` hier Umbenennungen dieser Variablen vorgenommen. In diesem Fall handelt es sich um die Variablen *IQ* und *id*, welche in beiden Datensätzen (*data* und *data_select*) vorkommen und in diesem Fall sogar nicht nur Variablen mit gleichen Namen, sondern sogar Variablen mit exakt den gleichen Werten sind.

Die Variable *id* dient dazu, jeder Person im Datensatz eine eindeutige Nummer zuzuschreiben – die Nummern funktionieren also prinzipiell wie Namen, obwohl es sich um Zahlen handelt. Sowohl im Datensatz *data* als auch im Datensatz *data_select* gibt es Personen mit der gleichen *id*-Nummer, weil es sich in beiden Datensätzen um Daten der gleichen Personen handelt. Somit erscheint es sinnvoll, beim Zusammenfügen von Datensätzen die *id*-Nummer zu berücksichtigen, sodass jeder Person die neuen Daten zugeschrieben werden, die per id-Nummer zu ihr gehören. Häufig ist es in der wissenschaftlichen Praxis so, dass sich die Daten in unterschiedlichen Datensätzen auf die gleichen Personen beziehen, wobei ggf. die Reihenfolge der Personen in den Datensätzen nicht identisch ist. Allerdings kann jede Person

dann über die jeweilige *id*-Nummer eindeutig identifiziert werden. Mit `R` ist es möglich, die Daten dann anhand der jeweiligen *id*-Nummern zusammenzufügen. Das Prinzip wird in Abbildung 47 verdeutlicht.

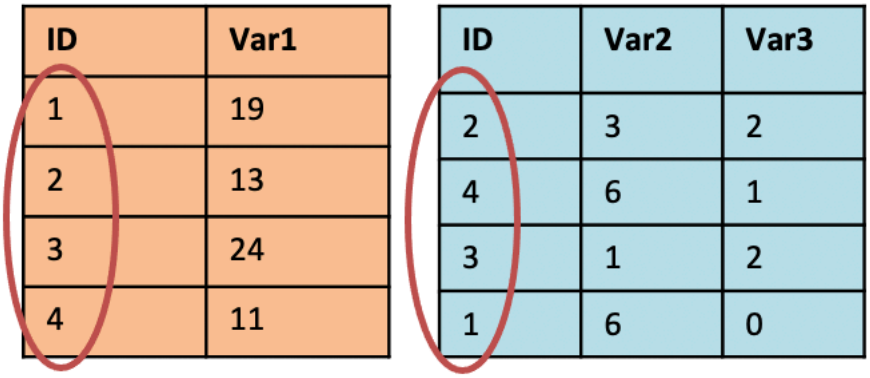

Abbildung 47: Zusammenfügen von Datensätzen via ID-Nummer – merge()-Funktion

Um diese Art des Zusammenfügens zu realisieren, kann die Syntax verwendet werden, die in Abbildung 48 dargestellt ist:

```
data_byID <- merge (data, data_select, by = „id")
```

Zu diesem Zweck wird die `merge()`-Funktion verwendet, die prinzipiell wie die `bind_cols()`-Funktion funktioniert, allerdings wird nach der Benennung der Datensätze (abgetrennt mit einem Komma) noch ausgedrückt, wie die Datensätze zusammengefügt werden sollen: Dazu wird zunächst der Ausdruck *by* geschrieben, es folgt ein Gleichheitszeichen und dann der Variablenname der Variable, die verwendet werden soll, um die Datensätze zusammenzufügen (in diesem Fall: *id*).

Im *Environment* von Abbildung 48 fällt auf, dass der mithilfe der Syntax neu erzeugte Datensatz *data_byID* 48 Variablen umfasst, während der Datensatz *data_cols* 49 Variablen beinhält, obwohl die gleichen Datensätze (in beiden Fällen *data* und *data_select*) verwendet wurden, um die neuen Datensätze zu erzeugen. Der Grund für den Unterschied ist, dass bei dem Datensatz *data_cols* die beiden Datensätze, die als Grundlage zur Erstellung des Datensatzes gedient haben, einfach hintereinander zusammengefügt wurden (44 Variablen + 5 Variablen), während bei *data_byID* die Variable *id* die Funktion erfüllt hat, die Datensätze gezielt zusammenzuführen. Dabei wurde berücksichtigt, dass die Variable *id* in beiden Datensätzen vorkommt und daher im resultierenden Datensatz nur einmal vorkommen muss. Deshalb besitzt der Datensatz *data_cols* die *id*-Variable (mit unterschiedlichen Namen) zweimal, während der

Datensatz *data_byID* die *id*-Variable nur einmal aufweist. So erklärt sich die unterschiedliche Variablenanzahl in den Datensätzen.

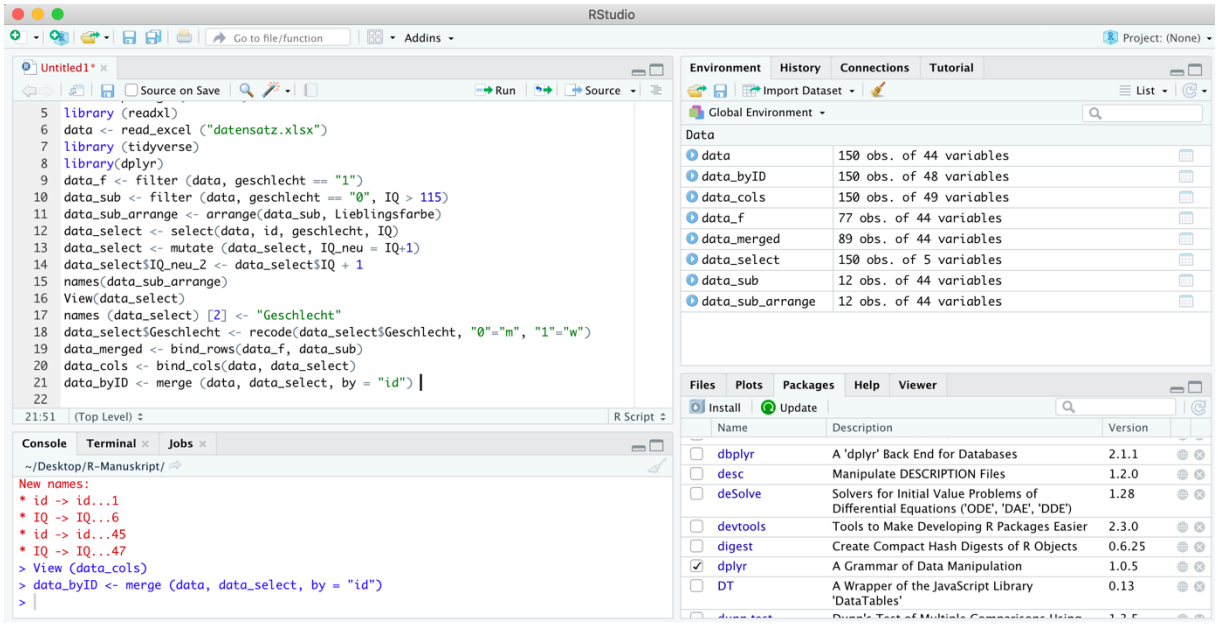

Abbildung 48: Zusammenfügen von Datensätzen mithilfe der id-Nummer

3 Deskriptive Statistik

Im Folgenden wird dargestellt, wie Sie die deskriptive Datenanalyse in `RStudio` vornehmen können. Zur statistischen Datenauswertung ist die Verwendung spezifischer Funktionen und Befehle notwendig, die Sie prinzipiell wie Vokabeln lernen können. Alle Funktionen und Befehle, die im Folgenden verwendet werden, werden vorgestellt und erklärt. Zentrale Aspekte werden mit Screenshots dargestellt, um Ihnen das Verstehen nach Möglichkeit zu erleichtern. Viele nützliche Informationen zur Verwendung von `RStudio` können Sie auch in Form eines sogenannten „Cheat Sheets" einsehen: https://www.rstudio.com/resources/cheatsheets/. Zur Vertiefung der Konzepte und Theorien, auf denen die in diesem Kapitel dargestellten Informationen zur praktischen Anwendung statistischer Verfahren in `R` bzw. `RStudio` basieren, eignen sich insbesondere die Werke von Holling und Gediga (2011), Döring und Bortz (2016) sowie Bortz und Schuster (2010). Falls Interesse besteht, sich über die im Folgenden dargestellten Inhalte hinaus mit der Statistik in `R` zu beschäftigen, so ist insbesondere das Werk von Navarro (2016) zu empfehlen.

In diesem Kapitel sowie in den folgenden Unterkapiteln werden wir den Datensatz, den Sie bereits in Kapitel 2.9 eingelesen haben, verwenden. Bitte beachten Sie bei der Datenauswertung, dass in diesem Datensatz unter der Variable *Geschlecht* bzw. *geschlecht* männlich als *0* und weiblich als *1* definiert wird.

3.1 Skalenniveaus

In `R` müssen mindestens drei Skalenniveaus unterschieden werden, da von dem jeweiligen Skalenniveau der vorliegenden Daten abhängt, welche statistischen Methoden angewendet werden können. In diesem Kontext unterscheiden wir 1) das *nominale Skalenniveau*, 2) das *ordinale Skalenniveau* und 3) das *metrische bzw. intervallskalierte Skalenniveau*. Dabei ist der Informationsgehalt des ordinalen Skalenniveaus höher als der Informationsgehalt des nominalen Skalenniveaus und der Informationsgehalt des metrischen Skalenniveaus ist höher als der Informationsgehalt des ordinalen Skalenniveaus.

Als *nominal skaliert* werden Daten bezeichnet, deren mögliche Ausprägungen zwar unterschieden werden können, aber nicht in eine sinnvolle Reihenfolge gebracht werden können. Wenn beispielsweise die Namen von Proband:innen notiert werden, so kann die Variable *Name* unterschiedliche Ausprägungen haben, z.B.: *Otto*, *Ali*, *Hannah*, *Hugo*, *Lisa*, *Hatice*, *Otto*. Diese Namen können unterschieden werden und es kann gezählt werden, wie oft ein Name vorkommt (die Ausprägung *Otto* kommt in der vorherigen Aufzählung z.B. zweimal vor); aber Sie können die Namen nicht in eine eindeutige bzw. sinnvolle Rangreihenfolge

bringen und die Reihenfolge der Ausprägungen ist somit austauschbar bzw. beliebig. Somit ist es möglich, unterschiedliche Merkmalsausprägungen abzubilden, aber nicht die Intensität der Merkmalsausprägungen dabei auszudrücken. Ein besonderer Fall der Nominalskala ist die dichotome bzw. binäre Ausprägung von Variablen: Wenn nur zwei Ausprägungen vorliegen (z.B. *männlich*/ *weiblich* oder *Ja*/ *Nein*), so wird von einer dichotomen bzw. binären Variable gesprochen. Auch diese dichotomen/ binären Variablen befinden sich auf dem nominalen Skalenniveau. Bei nominal skalierten Variablen können Sie das *arithmetische Mittel* oder den *Median* nicht berechnen bzw. in den allermeisten Fällen nicht sinnvoll interpretieren. Der *Modus* (= welche Ausprägung kommt am häufigsten vor) kann jedoch bestimmt werden. Im obigen Beispiel wäre der Modus *Otto*, weil die Ausprägung *Otto* zweimal vorkommt, während alle anderen Ausprägungen nur einmal vorkommen. Für alle übrigen Skalenniveaus kann der Modus prinzipiell auch bestimmt werden.

Der Informationsgehalt des *ordinalen Skalenniveaus* ist höher als der Informationsgehalt des nominalen Skalenniveaus, aber niedriger als der Informationsgehalt des metrischen Skalenniveaus. Bei Variablen, die ordinal skaliert sind, können die Ausprägungen in eine sinnvolle Rangreihenfolge gebracht werden, allerdings sind keine Aussagen über die Abstände möglich. Ein beliebtes Beispiel in diesem Kontext sind die Platzierungen als Ergebnis bei einem Autorennen: Der erste Platz liegt vor dem zweiten Platz und der zweite Platz liegt vor der dem dritten Platz; allerdings kann anhand der bloßen Platzierung keine Aussage darüber getroffen werden, wie weit die Abstände zwischen den Plätzen sind/ waren. So kann es z.B. sein, dass das Auto auf dem ersten Platz 3 Minuten vor dem Auto auf dem zweiten Platz ins Ziel gekommen ist, aber zwischen dem zweiten und dritten Platz nur wenige Sekunden lagen. Daher kann der Abstand zwischen den Ausprägungen/ Rängen bei einem ordinalen Skalenniveau nicht sinnvoll interpretiert werden. Auch Schulnoten werden meist als ordinal skaliert interpretiert (auch wenn diese in der praktischen Datenauswertung einiger Studien wiederum wie intervallskalierte Variablen behandelt werden, was jedoch oft nicht sinnvoll bzw. zielführend ist). Bei ordinal skalierten Variablen können Sie das arithmetische Mittel nicht sinnvoll interpretieren, aber dafür zusätzlich zum Modus den Median sinnvoll bestimmen.

Der Informationsgehalt von Variablen, die sich auf dem *metrischen Skalenniveau* befinden, ist höher als der Informationsgehalt von Variablen auf dem ordinalen oder nominalen Skalenniveau. Alternativ kann das metrische Skalenniveau auch als mindestens intervallskaliert beschrieben werden. So umfasst das metrische Skalenniveau in den meisten Definitionen nämlich sowohl intervallskalierte Daten (z.B. IQ-Werte, bei denen es aber keinen natürlichen Nullpunkt gibt) als auch verhältnisskalierte Daten (z.B. die je individuelle Körpergröße mit

einem natürlichen Nullpunkt). Das Gewicht oder die Länge eines Gegenstandes, die Geschwindigkeit eines Fahrzeuges sowie das Alter von Personen oder das individuelle Einkommen sind typische Variablen, deren Ausprägungen in der Regel als metrisch zu interpretieren sind. Auch bei metrischen Variablen ist die Bildung einer Rangreihenfolge möglich, allerdings können bei metrisch skalierten Variablen darüber hinaus die Abstände zwischen den Ausprägungen sinnvoll interpretiert werden. Daher können bei metrisch skalierten Variablen auch Differenzen und Summen gebildet werden. Außerdem können zahlreiche Berechnungen mit metrisch skalierten Variablen durchgeführt werden: Bei metrisch skalierten Variablen können Sie den Modus, den Median und das arithmetische Mittel bestimmen.

3.2 Häufigkeiten

Bei jedem Skalenniveau können Sie die Häufigkeiten der Ausprägungen einer Variable bestimmen. Dazu müssen Sie zählen, wie häufig eine Ausprägung vorkommt. Diese Arbeit kann Ihnen `R` bzw. `RStudio` abnehmen. Zum Abzählen von Häufigkeiten eignet sich der Befehl `table()`.

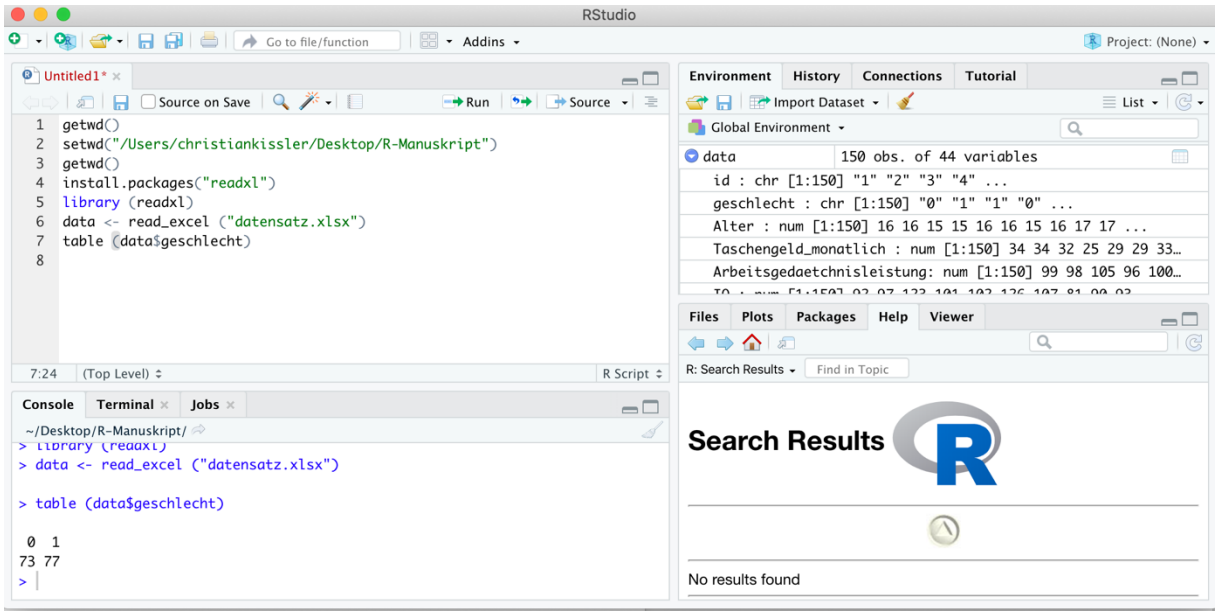

Abbildung 49: Abzählen mit table()

Wenn Sie wissen wollen, wie viele Personen in Ihrem Datensatz männlich bzw. weiblich sind, so können Sie dies mit dem Befehl

```
table(data$geschlecht)
```

herausfinden (siehe Abbildung 49). Dieser Befehl ist folgendermaßen zu verstehen: Mit der Funktion `table()` wird der Datensatz *data* angesteuert und anschließend wird in dem Datensatz *data* die Variable *geschlecht* angesteuert. Vor dem *$*-Zeichen benennen Sie also den Datensatz, auf den Sie sich beziehen, und hinter das *$*-Zeichen schreiben Sie die Variable, die Sie im ausgewählten Datensatz ansteuern. Anschließend wird in der Konsole das Ergebnis ausgegeben (Abbildung 49). Es ist zu erkennen, dass im Datensatz *data* 73 Personen das Merkmal *0* (also männlich) und 77 Personen das Merkmal *1* (also weiblich) aufweisen.

Potenziell können Sie aber auch komplexere Abzählaufgaben mit der Funktion `table()` lösen. Wenn Sie sich zum Beispiel für zwei Variablen (*geschlecht* und *Lieblingsfarbe*) interessieren und visuell überprüfen wollen, ob es einen Zusammenhang zwischen *Geschlecht* und *Lieblingsfarbe* gibt, so können Sie zur Lösung dieser Aufgabe ebenfalls *`table()`* benutzen:

```
table(data$Lieblingsfarbe, data$geschlecht, exclude = NULL)
```

Sie steuern also nacheinander beide Variablen an und trennen diese mit einem Komma. Der Zusatz *exclude = NULL* würde in dem Fall, dass fehlende Werte vorliegen würden, dafür sorgen, dass auch diese in der Tabelle angezeigt werden. Abbildung 50 ist das Ergebnis zu entnehmen, das herauskommt, wenn der oben genannte Befehl ausgeführt wurde.

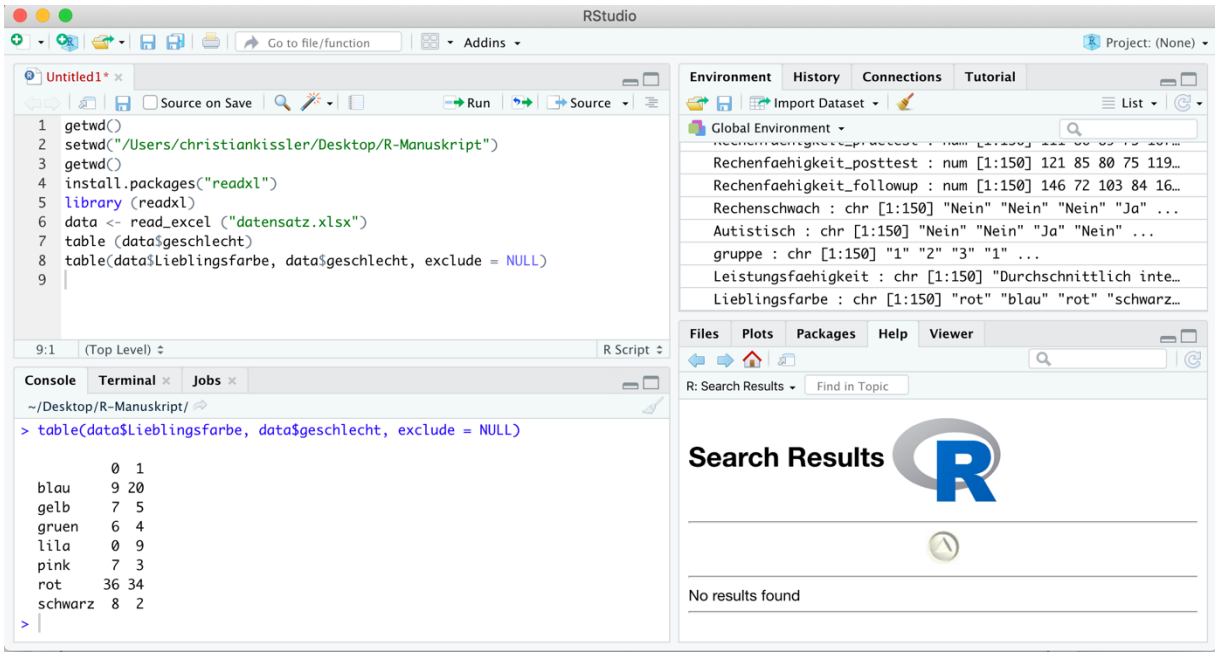

Abbildung 50: Kreuztabellen

In der Konsole von Abbildung 50 ist zu erkennen, dass 9 *männliche* Personen die Farbe *Blau* bevorzugen, während 20 *weibliche* Personen die Farbe *Blau* als ihre Lieblingsfarbe benannt

haben. Die Farbe *Blau* ist also die Lieblingsfarbe von 29 Personen (20+9). Diese Operation, um die Gesamtheit der Personen, welche eine spezifische Farbe bevorzugen, auszurechnen, wurde nun händisch durchgeführt. Solche *Randhäufigkeiten* können aber auch mit dem Befehl `addmargins()` und in diesem Fall konkret mit dem Code

```
addmargins (table(data$Lieblingsfarbe, data$geschlecht, exclude
      = NULL))
```

durch `R` bestimmt und ausgeben werden (Abbildung 51).

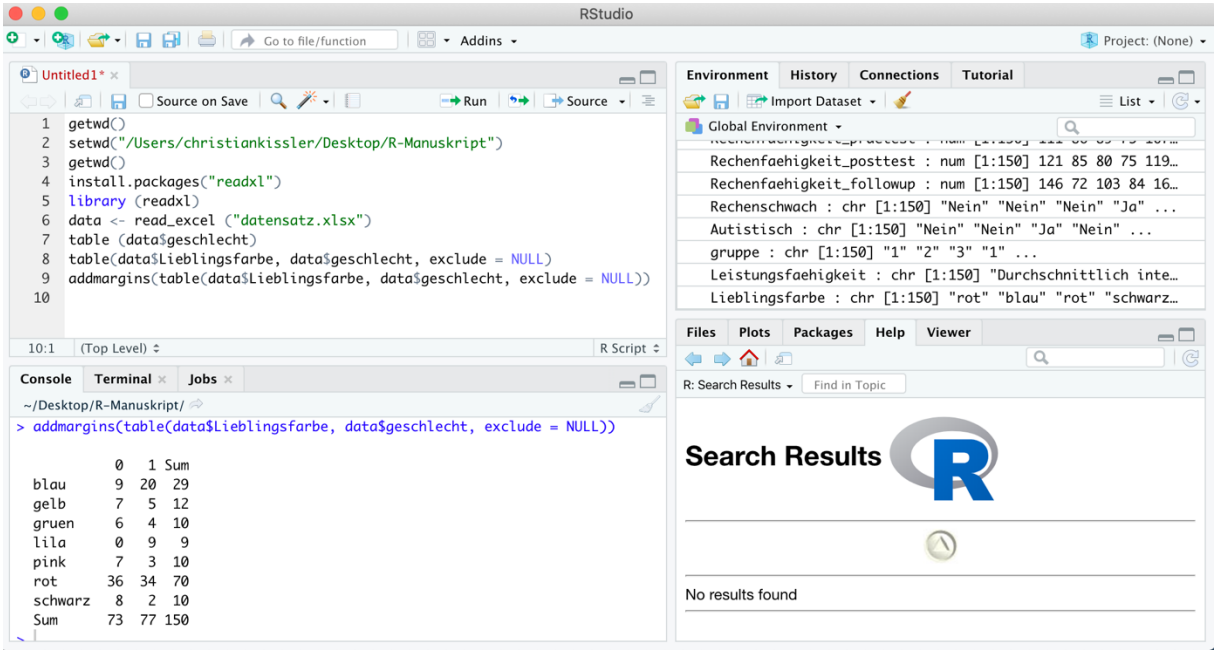

Abbildung 51: Randhäufigkeiten

Der Konsole von Abbildung 51 ist zu entnehmen, dass die Tabelle aus Abbildung 50 um die Randhäufigkeiten (*Sum*) ergänzt wurde. Ganz unten rechts in der Tabelle von Abbildung 51 ist die Gesamtanzahl der Proband:innen zu erkennen (150). Im Skript von Abbildung 51 ist zu sehen, dass zur Erstellung der Randhäufigkeiten die Funktion `addmargins()` verwendet wurde, wobei in den Klammern von `addmargings()` der bereits aus Abbildung 50 bekannte `table()`-Befehl steht. Dies bedeutet, dass auf die vorherige Funktion, welche bereits in Abbildung 50 dargestellt wurde, zusätzlich noch die Funktion `addmargins()` angewendet wurde.

Bisher wurden Häufigkeitstabellen für *absolute Häufigkeiten* ausgegeben. Es ist aber auch möglich, die *relativen Häufigkeiten* darzustellen. Dazu ist ähnlich wie bei `addmargins()` aus Abbildung 51 nun die Funktion `prop.table()` auf die `table()`-Funktion anzuwenden:

```
prop.table (table(data$Lieblingsfarbe, data$geschlecht, exclude
        = NULL))
```

Das Resultat dieses Befehls ist in Abbildung 52 dargestellt: Nun werden anstelle von absoluten Häufigkeiten die relativen Häufigkeiten ausgegeben, die sich in der Summe zu 1 aufaddieren.

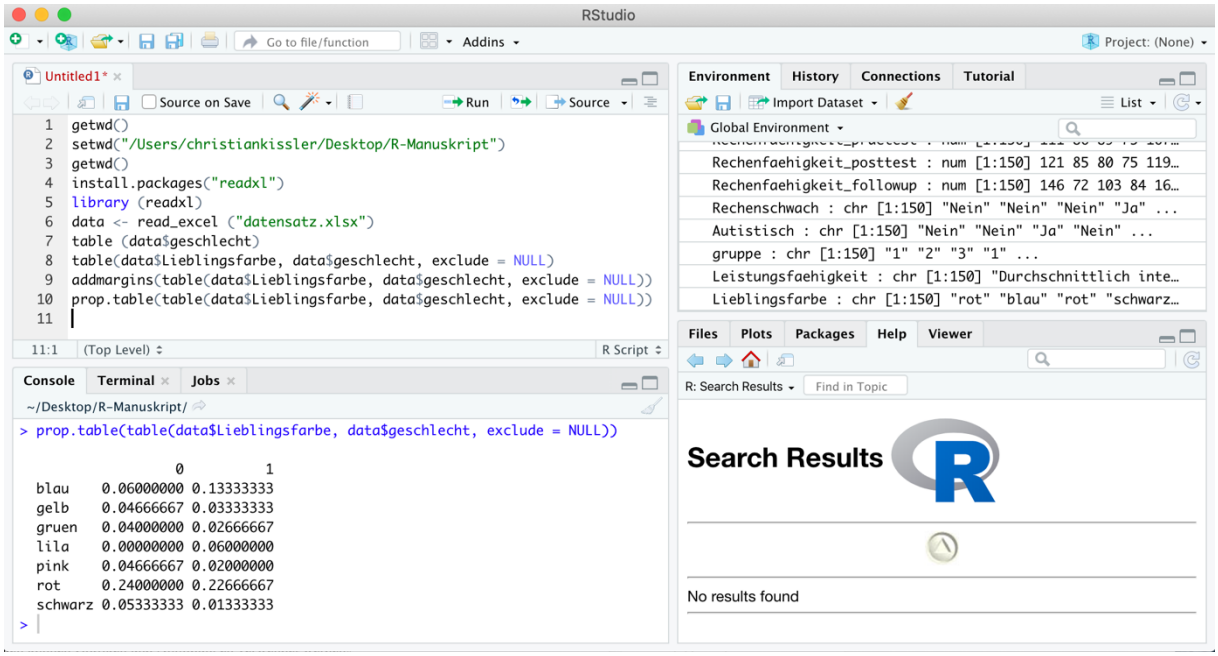

Abbildung 52: Relative Häufigkeiten mit prop.table()

Nach Belieben kann auch die `addmargins()`-Funktion mit der `prop.table()`-Funktion kombiniert werden, damit zusätzlich zu den Angaben in Abbildung 52 die relativen Randhäufigkeiten ausgeben werden (Abbildung 53). Der entsprechende *Code* dazu würde für diesen Fall folgendermaßen lauten:

```
addmargins (prop.table (table(data$Lieblingsfarbe,
      data$geschlecht, exclude = NULL)))
```

Das Resultat dieses Befehls ist in Abbildung 53 dargestellt. Auf diese Weise können Sie sich mithilfe von Tabellen einen ersten Eindruck über die Verteilung der Daten verschaffen. Die visuelle Inspektion von Daten stellt einen zentralen Schritt in der Datenanalyse dar. Diese grundlegenden Funktionen, die in diesem Kapitel vorgestellt wurden, sind auch notwendig, um einige Grafiken bzw. Abbildungen zu erzeugen, mit denen Sie die Daten noch besser visualisieren können. Das Verknüpfen von Funktionen, das in diesem Kapitel z.B. anhand der Verknüpfung von `addmargins()` und `prop.table()` demonstriert wurde, stellt in `R` bzw. `RStudio` außerdem ein zentrales Prinzip dar.

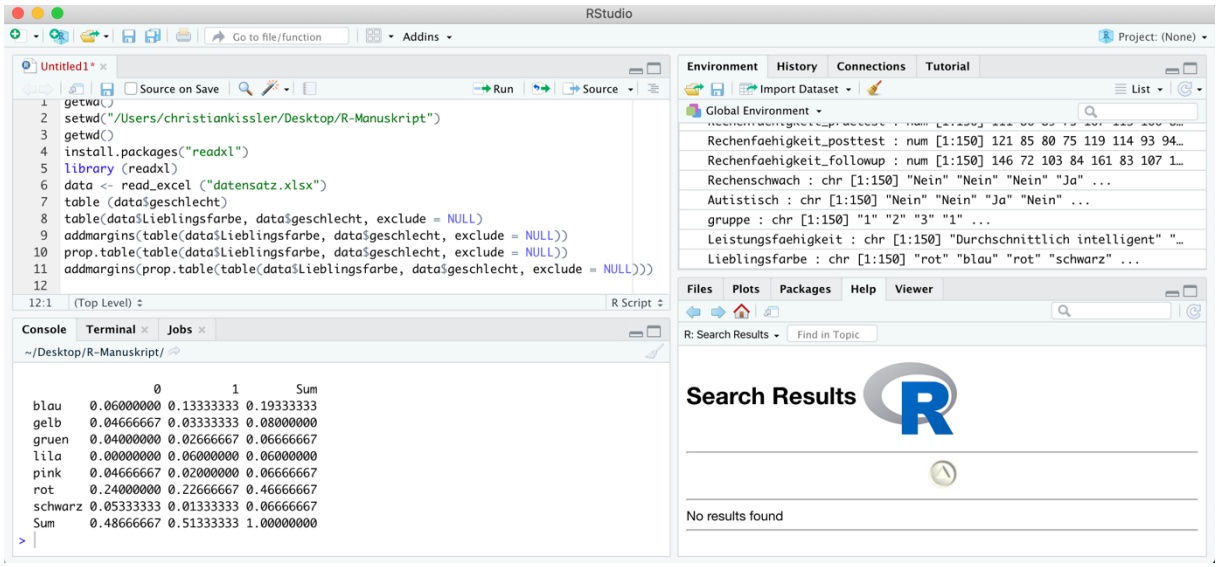

Abbildung 53: Relative Häufigkeiten und Randhäufigkeiten kombiniert

3.3 Quantile

Ein Quantil ist ein spezifisches Lagemaß. Mit einem Quantil kann eine Verteilung in zwei Teile geteilt werden: einen Teil, der maximal den gewünschten Wert annimmt, und einen Teil, der darüber liegt. Wenn beispielsweise bei der Variable *IQ* im Datensatz *data* das 25%-Quantil bestimmt werden soll, so ist das 25%-Quantil der Wert, für den gilt, dass 25% der Werte im Datensatz *data* kleiner bzw. maximal gleich zu diesem IQ-Wert sind. Die übrigen 75% der Werte dieser Variable im Datensatz *data* würden dann also über diesem Wert liegen.

Ein solches Quantil kann mit der `quantile()`-Funktion berechnet werden. Die Funktion lautet für das bereits genannte Beispiel:

```
quantile (data$IQ, na.rm=TRUE)
```

Der Zusatz *na.rm=TRUE* wird dabei verwendet, damit fehlende Daten bei der Berechnung des Quantils nicht berücksichtigt werden; *rm* steht dabei für „*remove*“ und *na* steht für „*not applicable*“. Das Ergebnis des beschriebenen Befehls ist in Abbildung 54 dargestellt. Der Ausgabe in der Konsole ist zu entnehmen, dass das 25%-Quantil bei 90 liegt: 25% der Proband:innen haben also einen IQ von maximal 90, die übrigen 75% der Stichprobe liegen über diesem Wert. Das 75% Quantil liegt bei 109.25: 75% der Proband:innen haben also einen *IQ* von maximal 109.25, die übrigen 25% der Stichprobe liegen über diesem Wert.

In Abbildung 54 ist also dargestellt, wie Quantile im Abstand von 25%-Intervallen erzeugt werden können. Dazu wurde der oben bereits dargestellte Code verwendet. Unter Verwendung der Standardeinstellungen der Funktion `quantile()` werden also 25%-Quantile erzeugt.

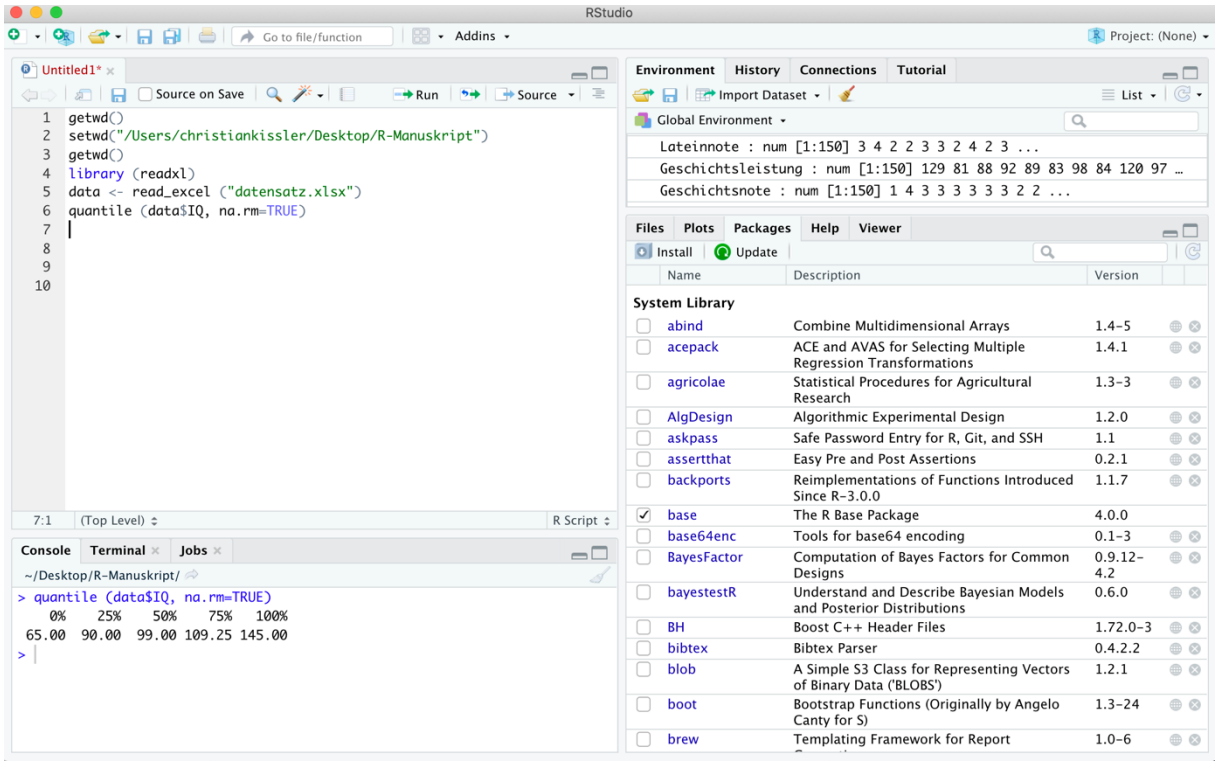

Abbildung 54: Bestimmung von Quantilen

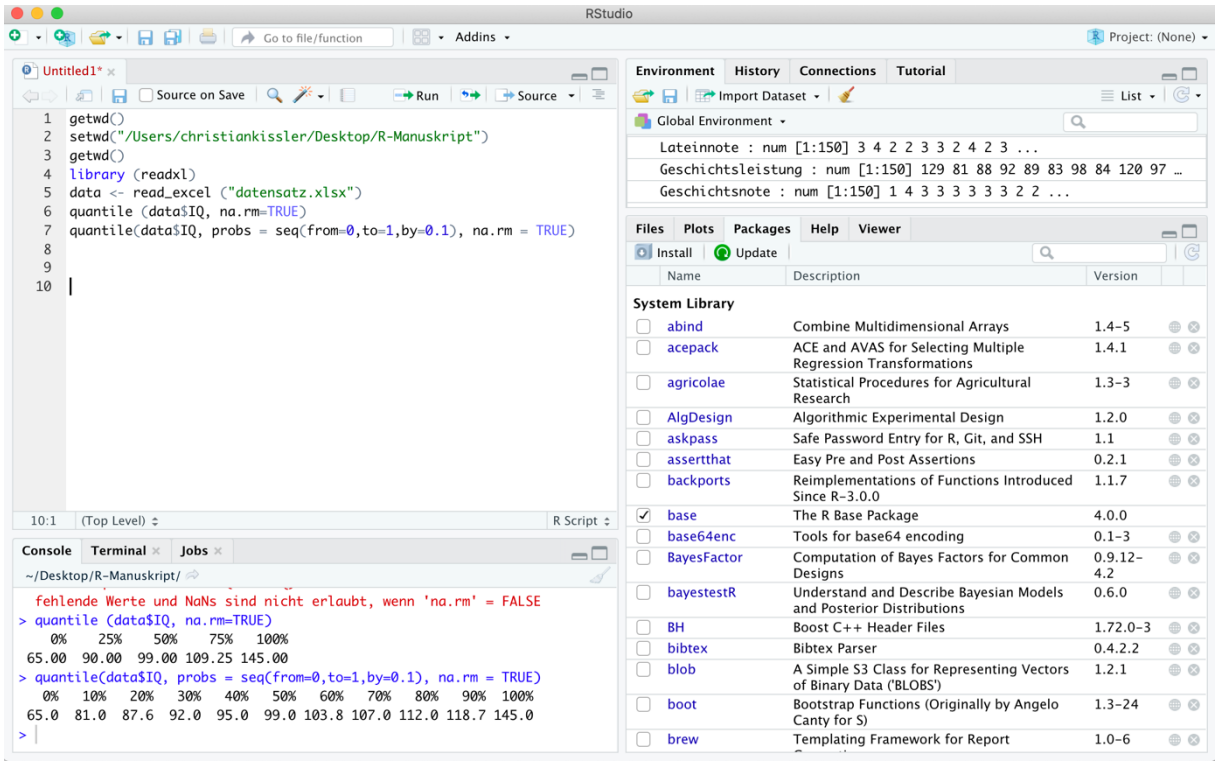

Abbildung 55: Bestimmung spezifischer Quantile

Es kann aber eventuell auch von Interesse sein, andere Quantile bzw. Quantile in anderen Abständen zu erzeugen. Dies ist mit dem folgenden Befehl z.B. für Quantile im Abstand von 10% möglich:

```
quantile(data$IQ, probs = seq(from=0,to=1,by=0.1), na.rm = TRUE)
```

Das Ergebnis dieser Berechnung wird in Abbildung 55 dargestellt. Die Abstände der Quantile können also innerhalb der `quantile()`-Funktion mit dem Argument *probs = seq(from=0,to=1,by=0.1)* variiert werden, wobei mit *by=0.1* festgelegt wird, dass die Abstände z.B. 10% betragen sollen. Diese Abstände können prinzipiell nach Belieben angepasst werden. Mit „*from=0, to=1*“ wird angegeben, dass für den gesamten Datensatz, also von 0% (=0) bis 100% (=1) die 10%-Quantile ausgegeben werden sollen.

3.4 Maße der zentralen Tendenz und Streuungsmaße

Häufig erscheinen Rohdaten zunächst sehr unübersichtlich und meistens können diesen ad hoc nicht viele Informationen entnommen werden. Daher müssen diese Rohdaten aufbereitet und visualisiert werden. Um sich die erhobenen Daten besser vorstellen zu können, sind Grafiken und Abbildungen sinnvoll. Aber auch spezifische Kennwerte haben sich als geeignet erwiesen: Die Maße der zentralen Tendenz und die Streuungsmaße.

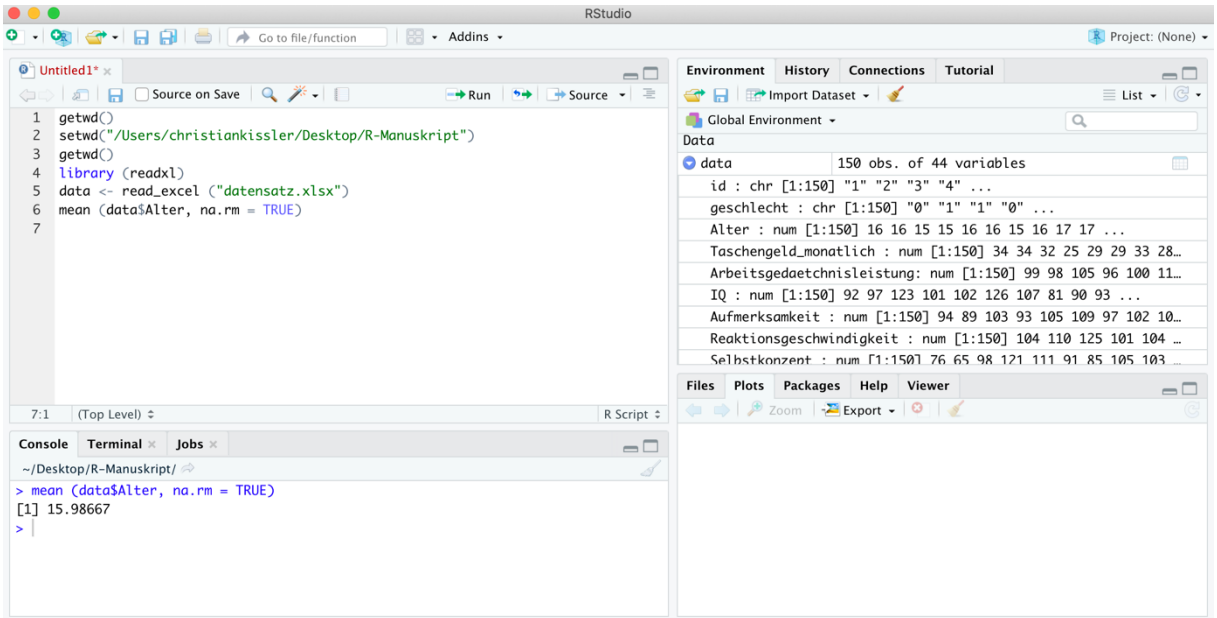

Abbildung 56: Verwendung der mean()-Funktion

Unter den Maßen der zentralen Tendenz sind der Modus, der Median und das arithmetische Mittel zu verstehen. Die Funktion `mean()` wird genutzt, um den Mittelwert einer mindestens intervallskalierten Variable zu berechnen. Dies bedeutet, dass die `mean()`-Funktion nicht auf nominal oder ordinal skalierte Variablen angewendet werden kann. Konkret kann mit dem Befehl

```
mean(data$Alter, na.rm=TRUE)
```

der Mittelwert der Variable *Alter* aus dem Datensatz *data* berechnet werden. Das Ergebnis der Berechnung ist in der Abbildung 56 zu sehen. Der Zusatz *na.rm=TRUE* wird dabei verwendet, damit fehlende Daten bei der Berechnung des Mittelwertes nicht berücksichtigt werden; *rm* steht dabei für „*remove*“ und na steht für „*not applicable*“. Sollten fehlende Werte vorhanden sein und der Zusatz *na.rm=TRUE* nicht verwendet werden, so wird in der Konsole eine Fehlermeldung angezeigt oder es wird kein Ergebnis ausgegeben.

Ein anderes Maß der zentralen Tendenz ist der *Median*. Der Median kann bei Variablen berechnet werden, die mindestens ordinalskaliert sind – dies bedeutet, dass der Median auch in den Fällen einer intervallskalierten Variable berechnet werden kann. Bei nominalskalierten Variablen hingegen kann der Median nicht bestimmt werden.

Der Median ist ein Lagemaß, bei dem alle Ausprägungen einer Variable in zwei Gruppen geteilt werden: 1) Mindestens 50% der Beobachtungen einer Variable sind kleiner oder gleich dem Median und 2) mindestens 50% der Beobachtungen einer Variable sind größer oder gleich dem Median. Der Median wird in `R` mit der `median()`-Funktion berechnet, welche Prinzipiell exakt wie die `mean()`-Funktion aufgebaut ist. Mit dem Befehl

```
median(data$Alter, na.rm=TRUE)
```

kann der Median der Variable *Alter* aus dem Datensatz *data* bestimmt werden. In Abbildung 57 ist zu sehen, dass der Median in diesem Fall 16 beträgt.

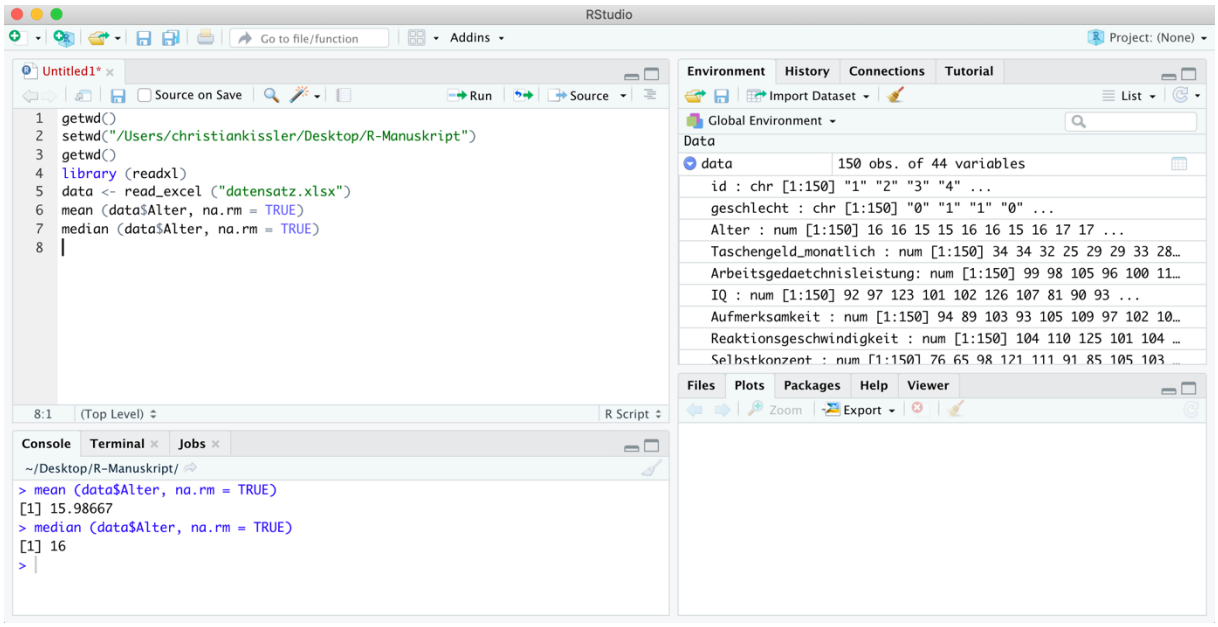

Abbildung 57: Die *median()-Funktion*

Der *Modus* ist ein weiteres Lagemaß, welches allerdings bei allen Variablen unabhängig von der Skalierung bestimmt werden kann – dies bedeutet, dass der Modus auch bei nominalskalierten Variablen bestimmbar ist. Der Modus ist die Ausprägung einer Variable, die am häufigsten vorkommt, und kann beispielsweise grafisch (z.B. über ein Balkendiagramm, welches im folgenden Kapitel dargestellt wird) oder alternativ über die `table()`-Funktion bestimmt werden. Der Konsole von Abbildung 58 kann entnommen werden, dass im Datensatz *data* für die Variable *Alter* der Modus 16 beträgt, weil die Ausprägung 16 94-mal vorhanden ist und somit die häufigste Ausprägung der Variable *Alter* ist. Dies wurde mit der bereits zuvor eingeführten `table()`-Funktion bestimmt.

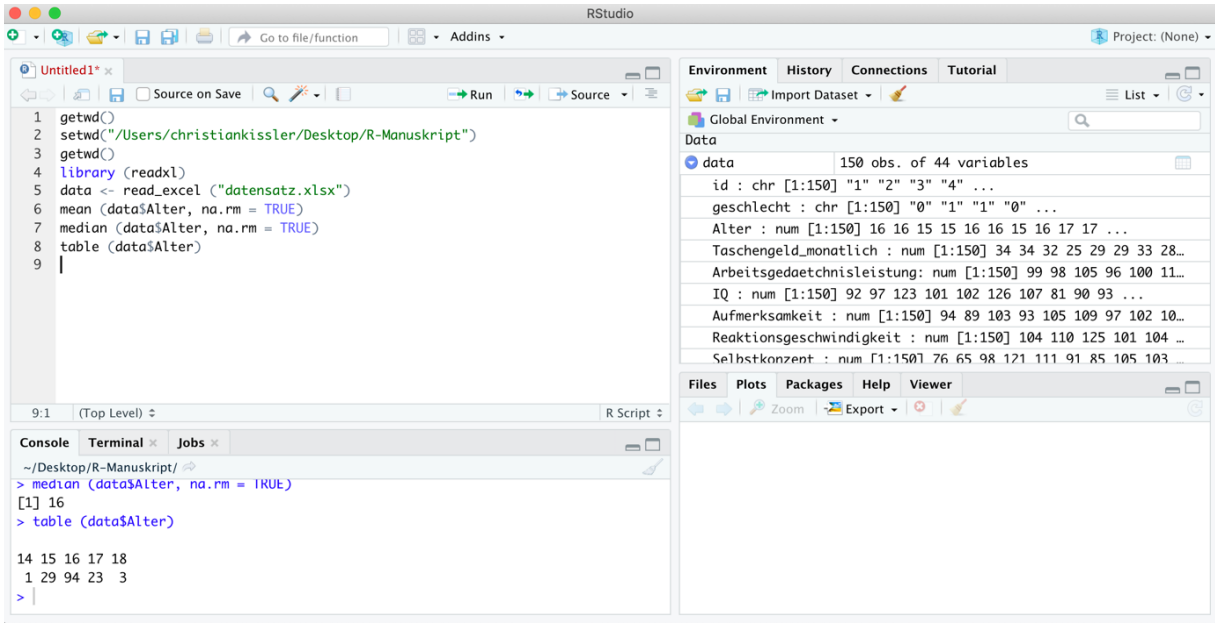

Abbildung 58: Bestimmung vom Modus mit der table()-Funktion

Unter die Streuungsmaße fallen insbesondere die *Spannweite*, die *Standardabweichung* und die *Varianz*. Die Spannweite einer Variable gibt an, von wo bis wo die erhobenen Ausprägungen einer Variable reichen (wichtig: Die Spannweite muss nicht alle möglichen Ausprägungen umfassen, die hätten erhoben werden können. So kann es beispielsweise sein, dass auf einer 9-stufigen Skala, die von 0 bis 8 reicht, nur die Werte 1 bis 5 erhoben wurden. Dann reicht die Spannweite von 1 bis 5, obwohl die Erhebungsskala in beide Richtungen potenziell noch andere Werte hätte annehmen können). Zur Bestimmung der Spannweite kann die Funktion

```
range()
```

genutzt werden, die in ihrem prinzipiellen Aufbau der `mean()`-Funktion sehr ähnlich ist. In Abbildung 59 ist zu erkennen, dass im Datensatz *data* die Spannweite für die Variable *Alter*

von 14 bis 18 reicht. Dies hätte theoretisch auch mit der `table()`-Funktion bestimmt werden können, indem beim jeweiligen Output der geringste und höchste Wert manuell identifiziert werden (Abbildung 59).

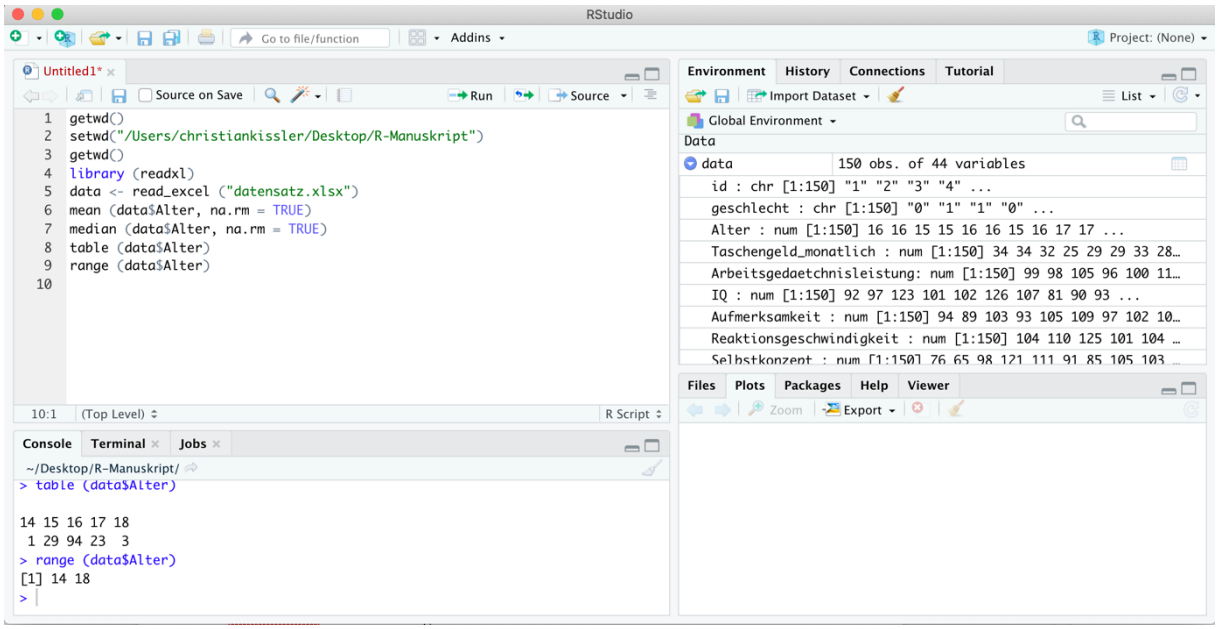

Abbildung 59: Spannweite mit der range()-Funktion bestimmen

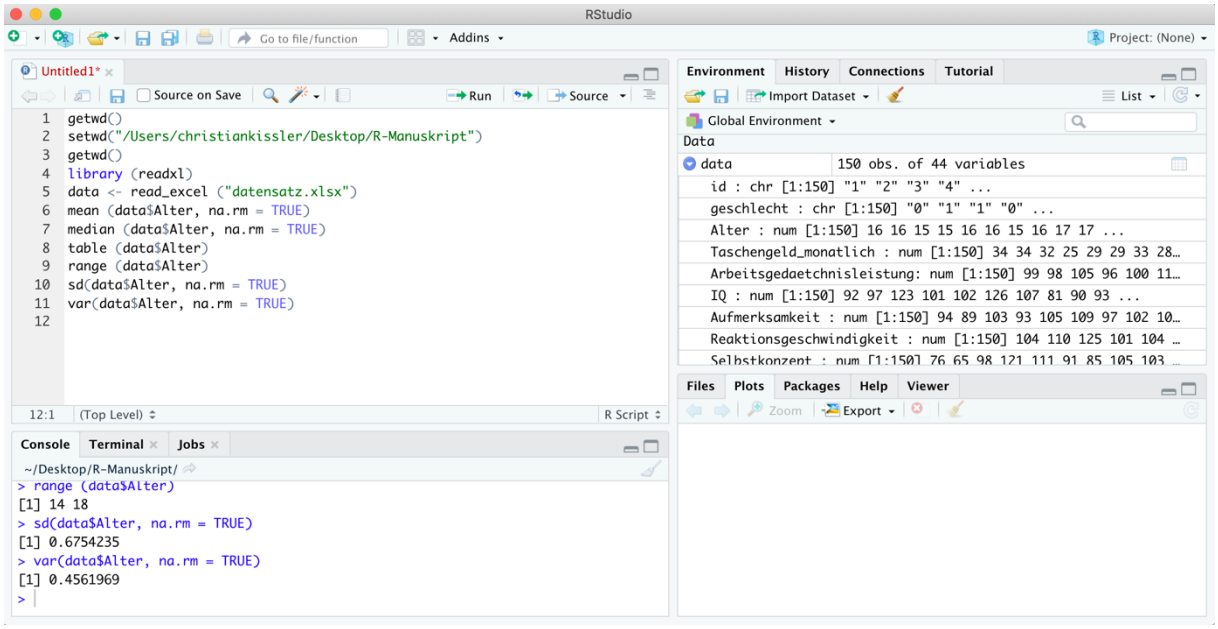

Abbildung 60: Bestimmung der Varianz und Standardabweichung

Ein anderes gängiges Streuungsmaß ist die Standardabweichung. Diese kann mit der Funktion

```
sd()
```

berechnet werden, die ebenfalls prinzipiell wie die `mean()`-Funktion aufgebaut ist. In

Abbildung 60 ist zu sehen, dass die Standardabweichung für die Variable *Alter* im Datensatz *data* bei etwa 0.68 liegt. Die Varianz kann mit der Funktion

```
var()
```

bestimmt werden und liegt für die Variable *Alter* im Datensatz *data* bei etwa 0.46 (Abbildung 60). Es ist festzuhalten, dass die Wurzel aus der Varianz die Standardabweichung ergibt bzw. umgekehrt das Quadrat der Standardabweichung die Varianz ergibt.

Bisher wurden für Variablen Streuungsmaße und Maße der zentralen Tendenz immer für alle Proband:innen eines Datensatzes gemeinsam berechnet. Manchmal interessieren wir uns aber beispielsweise für das mittlere Alter von Jungs einerseits und das mittlere Alter von Mädchen andererseits: Das mittlere Alter soll also nicht für alle Proband:innen zusammen berechnet werden, sondern die Berechnung soll jeweils für Teilgruppen getrennt erfolgen. Eine solche getrennte Berechnung ist mit der `tapply()`-Funktion möglich. Im Folgenden sollen das mittlere Alter von Jungs und das mittlere Alter von Mädchen aus dem Datensatz *data* jeweils getrennt berechnet werden. Dies ist mit der `tapply()`-Funktion folgendermaßen möglich:

```
tapply(data$Alter, data$geschlecht, mean, na.rm=TRUE)
```

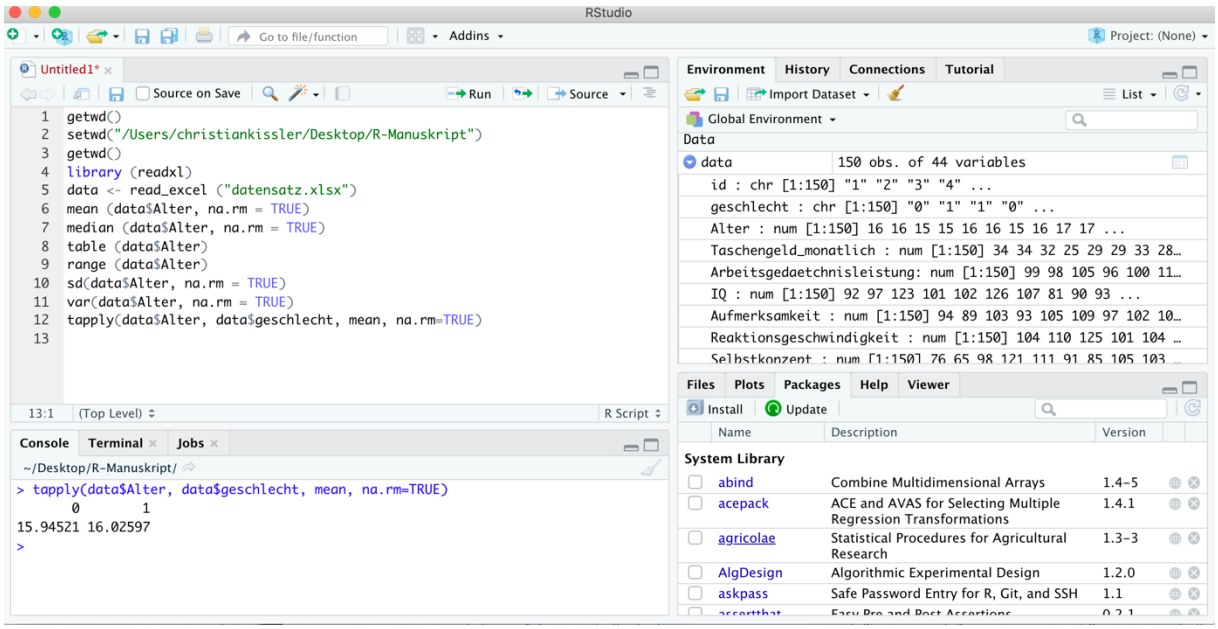

Abbildung 61: Verwendung der tapply()-Funktion

Innerhalb der Klammern der `tapply()`-Funktion wird zunächst benannt, für welche Variable aus welchem Datensatz etwas berechnet werden soll (*data$Alter*). Anschließend wird benannt, in welche Gruppen der Datensatz bei der Berechnung untergliedert werden soll: in diesem Fall

nach dem *Geschlecht* (*data$geschlecht*), wobei im Datensatz *data* die Ausprägung *0* für Jungs und die Ausprägung *1* für Mädchen steht. Danach wird angegeben, welche Berechnung durchgeführt werden soll: Wenn ein Mittelwert berechnet werden soll, so ist *mean* anzugeben; wenn jedoch beispielsweise die Standardabweichung berechnet werden soll, so ist *sd* anzugeben – für den Median wäre *median* und für die Varianz wäre *var* anzugeben. Zuletzt muss (wie bei der `mean()`-Funktion) angegeben werden, dass fehlende Daten bei den Berechnungen ausgelassen werden sollen (*na.rm=TRUE*). In Abbildung 61 ist die Durchführung der soeben beschriebenen Berechnung für das arithmetische Mittel (*mean*) dargestellt: Das mittlere Alter der Jungs liegt bei 15.95 und das mittlere Alter der Mädchen liegt bei 16.03.

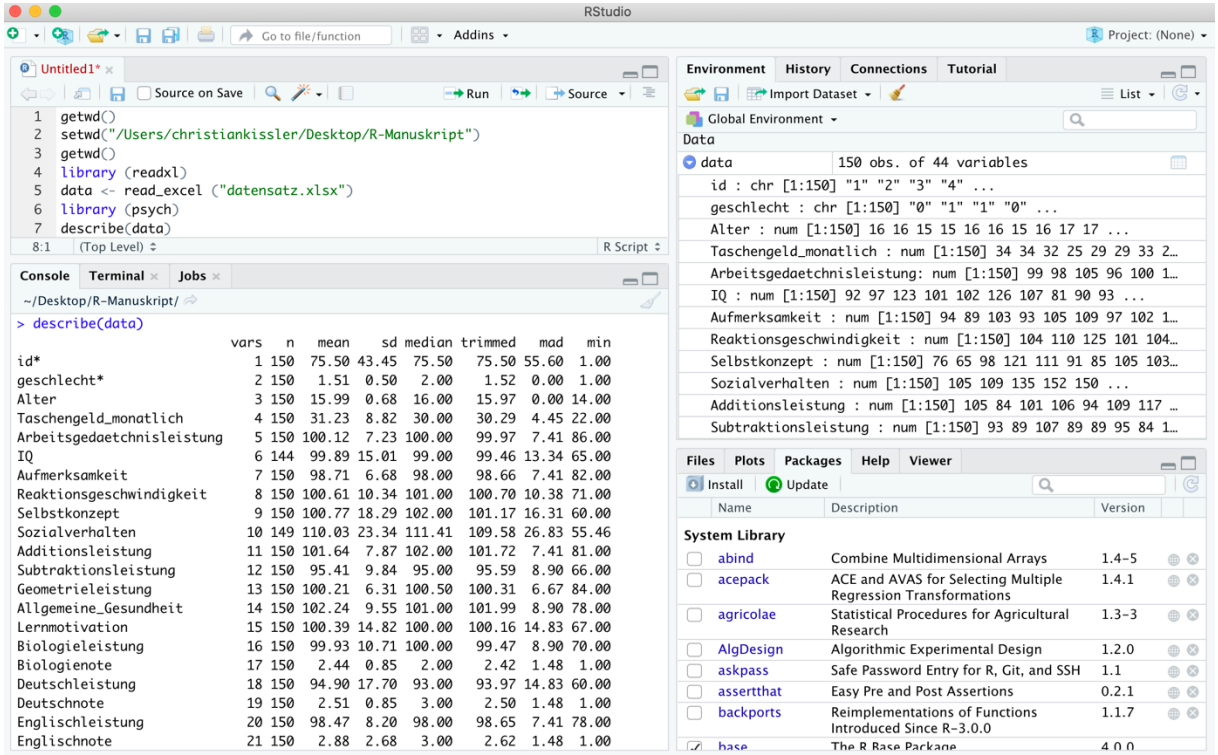

Abbildung 62: Verwendung der describe()-Funktion

Bisher wurde dargestellt, wie einzelne Werte der deskriptiven Statistik berechnet werden können. Es ist jedoch auch möglich, sich mehrere Werte zur Beschreibung einer Stichprobe gebündelt ausgeben zu lassen. Dazu ist das *Package psych* von Revelle (2022) zu installieren sowie zu aktivieren (z.B. über die `library()`- oder *`require()`*-Funktion) und anschließend die *`describe()`*-Funktion zu nutzen. Mit dem Befehl

```
describe(data)
```

können zu allen Variablen im Datensatz *data* mehrere geläufige Kennwerte ausgegeben werden (Abbildung 62).

In der Konsole von Abbildung 62 ist dargestellt, was herauskommt, wenn die `describe()`-Funktion auf den Datensatz *data* angewendet wird. Horizontal können Sie in der obersten Zeile der Ausgabe sehen, welche Kennzahlen mit der `describe()`-Funktion ausgegeben werden (z.B. *mean*, *sd*, *median*). Ganz links ist vertikal untereinander eine Auflistung aller Variablen des Datensatzes dargestellt und hinter einer jeden Variable ist eine Auflistung der berechneten Kennwerte für ebendiese Variable vorhanden. Im Folgenden wird anhand der Variable *IQ* (6. Zeile in der Konsolenausgabe von Abbildung 62) dargestellt, was diese Kennwerte bedeuten.

Zunächst ist zu sehen, dass die Werte-Ausgabe mit einem Wert beginnt, der mit *vars* überschrieben ist: *vars* ist eine durchlaufende Nummerierung und jeder Variable wird eine Nummer zugeordnet. Die Variable *IQ* hat die Variablennummer 6 zugeordnet bekommen. Somit ist der *vars*-Wert für die Interpretation der Daten nicht von weiterer Relevanz.

Unter *n* ist angegeben, wie viele Proband:innen gültige Werte (= nicht fehlende Werte) aufweisen: bei der Variable *IQ* liegen für 144 Personen entsprechende Daten vor, die bei den folgenden Berechnungen berücksichtigt werden können. Bei den meisten anderen Variablen sind es 150 Personen, für die entsprechende Werte vorliegen. Die Variable *IQ* weist also fehlende Werte auf.

Von den folgenden Kennwerten sind insbesondere die Werte für *mean* (arithmetisches Mittel), *sd* (Standardabweichung) und *min* (Minimalwert = der geringste Wert, der gemessen wurde) von Relevanz. Darüber hinaus gibt die `describe()`-Funktion noch weitere Werte aus, die aus Platzgründen allerdings nach einem Zeilenumbruch an anderer Stelle in der Konsole angezeigt werden. Wenn in der Konsole weiter nach unten gescrollt wird, so ist eine Fortsetzung der ausgegebenen Werte über *min* (den Minimalwert) hinaus zu finden (Abbildung 63).

Im Rahmen dieser Fortsetzung der berechneten Kennwerte ist zu sehen, dass unter *max* der Maximalwert (der höchste gemessene Wert) ausgegeben wird. Der Wert, der unter *range* ausgegeben wird, ist jedoch nicht so zu interpretieren, dass hier angegeben wird, von wo bis wo die erhobenen Werte der jeweiligen Variable theoretisch reichen können oder im Datensatz tatsächlich reichen. Es handelt sich hier vielmehr um einen Wert, der aussagt, wie breit das Spektrum vom tatsächlich erhobenen Minimalwert (bei der Variable *IQ* = 65) bis zum tatsächlich erhobenen Maximalwert (bei der Variable *IQ* = 145) ist: 145-65 = 80.

Außerdem werden noch die Schiefe bzw. Skewness (*skew*), die Kurtosis (*kurtosis*) und der Standardfehler (se*)* ausgegeben: Werte, die in wissenschaftlichen Ausarbeitungen häufig

zur Beschreibung einer Stichprobe genutzt werden. Denn mit ihnen kann die Verteilung der Werte genauer beschrieben werden.

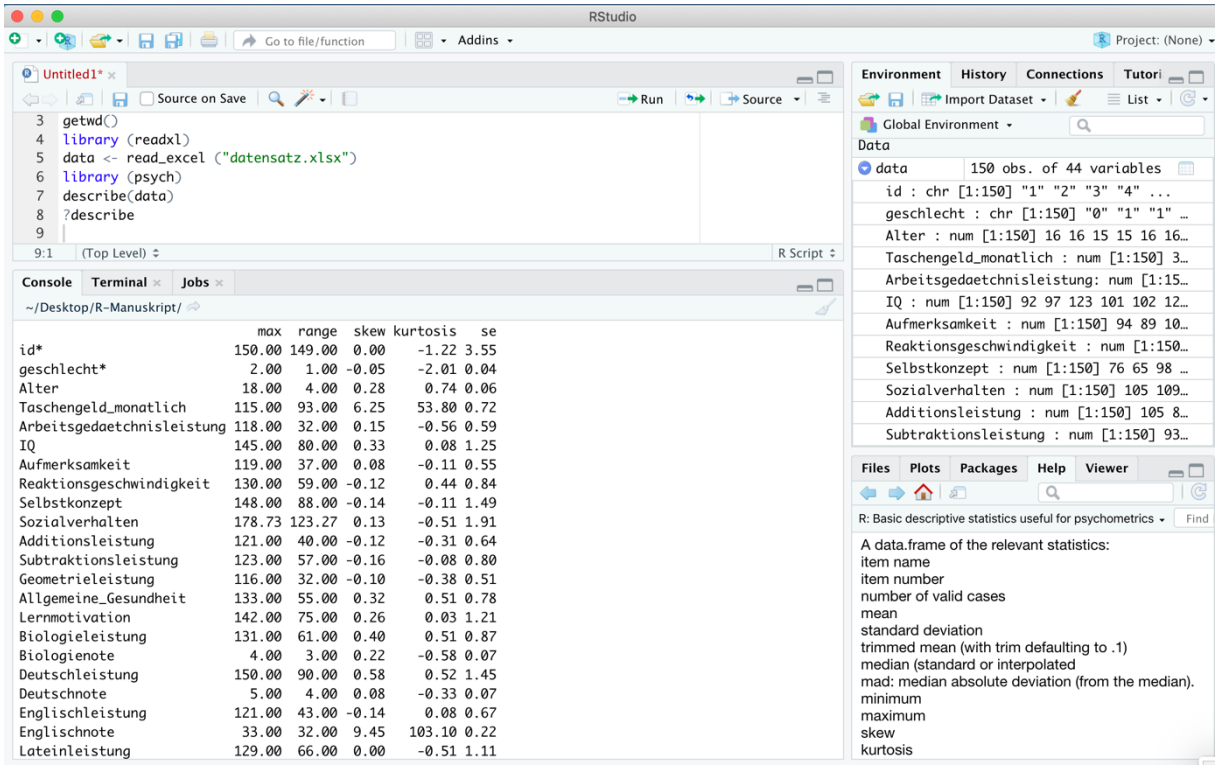

Abbildung 63: Die describe()-Funktion: Fortsetzung der Ausgabe

Alternativ können Skewness bzw. Schiefe und Kurtosis auch mit anderen Befehlen ausgegeben werden. Zur Bestimmung der Skewness wird die Funktion

```
skewness()
```

und zur Bestimmung der Kurtosis die Funktion

```
kurtosis()
```

aus dem *Package moments* von Komsta und Novomestky (2015) verwendet, wobei diese beiden Funktionen prinzipiell ebenso wie die `mean()`-Funktion aufgebaut sind (Bitte denken Sie ggf. daran, den Zusatz *na.rm=TRUE* zu verwenden).

Kritisch ist bei der Verwendung der `describe()`-Funktion anzumerken, dass stets reflektiert werden sollte, ob die berechneten Werte wirklich sinnvoll sind. Denn die `describe()`-Funktion berechnet für alle Variablen die oben beschriebenen Kennwerte unabhängig vom Skalenniveau. So wird beispielsweise auch zu der Variable *id* ein arithmetischer Mittelwert angegeben (Abbildung 62), obwohl diese Variable nominalskaliert ist. Daher sollte stets darüber nachgedacht werden, ob die berechneten Werte sinnvoll zu

interpretieren sind oder nicht. Allerdings ist die `describe()`-Funktion dazu geeignet, einen schnellen sowie groben Überblick über den Datensatz und so ein Gefühl für die vorliegenden Daten zu gewinnen.

Mit der Funktion `describeBy()` ist es außerdem möglich (ähnlich wie bei der `tapply()`-Funktion), den Gesamtdatensatz nach Gruppen aufzuteilen und sich getrennt nach Gruppen die oben beschriebenen Werte berechnen zu lassen. So würde der Befehl

```
describeBy(data, data$geschlecht)
```

dazu führen, dass die oben beschriebenen Werte 1) für Jungs und 2) für Mädchen separiert berechnet werden. Im Ausgabefenster würde dann für jede Gruppe eine entsprechende Tabelle mit den zahlreichen Kennwerten ausgegeben werden. So kann schnell ein Gefühl für die Daten entwickelt werden, wenn z.B. spezifische Gruppenvergleiche interessieren.

3.5 Grafiken/ Abbildungen erstellen

Zur Veranschaulichung von Daten bzw. zur Darstellung der Häufigkeiten von Ausprägungen spezifischer Variablen können Grafiken und Abbildungen verwendet werden. Die `table()`-Funktion, die zur prinzipiellen Bestimmung der Häufigkeit von Ausprägungen einer Variable geeignet ist, wurde in vorherigen Kapiteln bereits dargestellt. Viele Funktionen zur Erstellung von Abbildungen/ Grafiken bauen auf ebendieser `table()`-Funktion auf.

Das Kreisdiagramm stellt eine gängige Grafik dar, um Variablen mit wenigen Ausprägungen übersichtlich zu visualisieren. Ein Kreisdiagramm kann mit der `pie()`-Funktion erstellt werden, wobei die Darstellung des Kreisdiagramms auf der `table()`-Funktion aufbaut. Dass Funktionen miteinander verknüpft werden, stellt ein gängiges Vorgehen bzw. Prinzip in `RStudio` dar. Im Folgenden wird eine Grafik erzeugt, die visualisiert, wie viele Personen im Datensatz *data* autistisch (*ja*) bzw. nicht autistisch (*nein*) sind. Dazu wird der folgende Befehl verwendet:

```
pie(table(data$Autistisch), main = „Ist das Kind autistisch?“)
```

Der Befehl beginnt mit der `pie()`-Funktion, wobei in der Klammer zunächst die `table()`-Funktion folgt, welche sich wiederum auf die Variable *Autistisch* im Datensatz *data* bezieht. Nach der `table()`-Funktion kann der Grafik eine Überschrift gegeben werden: Mit dem Ausdruck *main = „Ist das Kind autistisch?“* wurde die Überschrift der Grafik erzeugt (Abbildung 64).

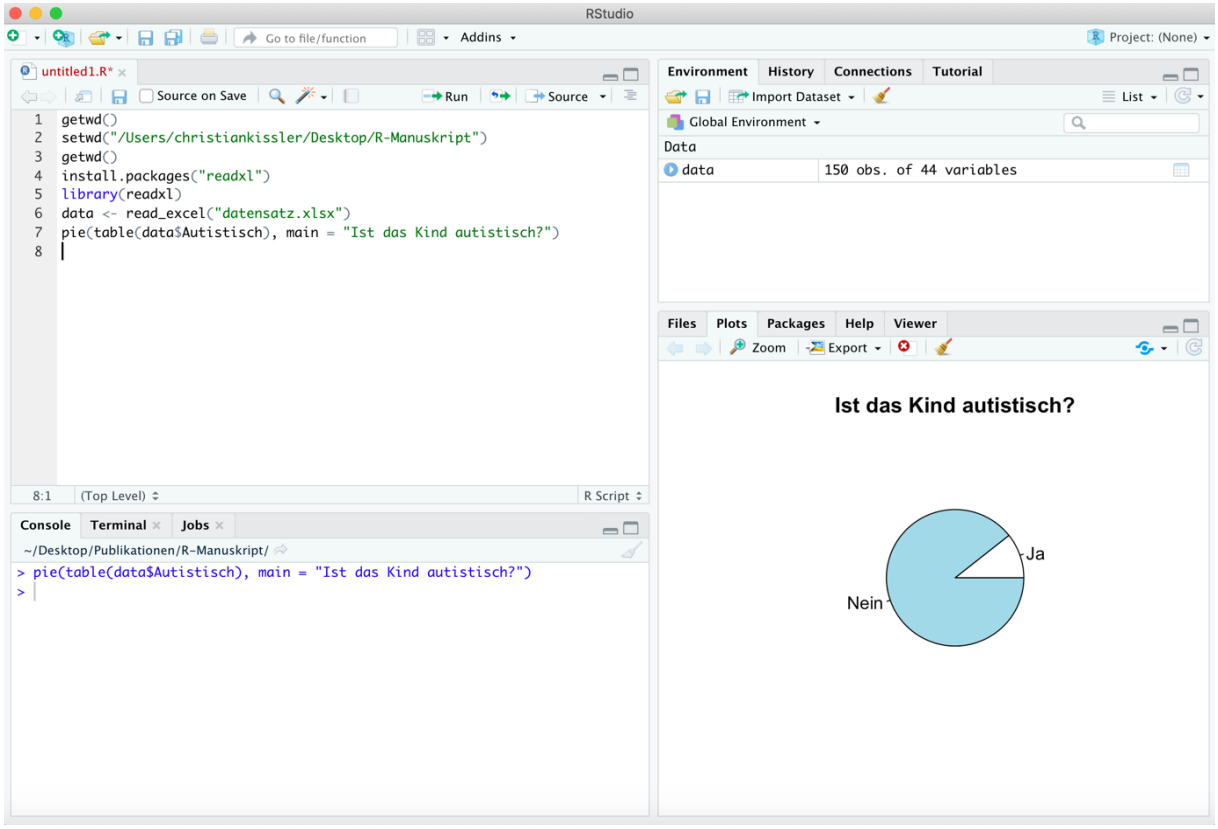

Abbildung 64: Verwendung der pie()-Funktion zur Erzeugung eines Kreisdiagramms

Eine andere Möglichkeit zur Darstellung von Daten ist das Balkendiagramm. Das Balkendiagramm kann mit der Funktion `barplot()` erzeugt werden und die Verwendung dieses Befehls funktioniert strukturell ähnlich zur `pie()`-Funktion. In Abbildung 65 werden die Daten, die in Abbildung 64 als Kreisdiagramm dargestellt wurden, als Balkendiagramm dargestellt. Dieses Balkendiagramm wurde mit dem folgenden Befehl erzeugt:

```
barplot (table(data$Autistisch),
      xlab="Ergebnis der Diagnostik",
      ylab="Anzahl der Personen",
      main = "Ist das Kind autistisch?",
      ylim=c(0, 200))
```

Viele Teile der verwendeten `barplot()`-Funktion sind dabei bereits aus der *`pie()`*-Funktion bekannt, während andere im Folgenden erklärt werden müssen. Wie es bei der *`pie()`*-Funktion der Fall ist, beginnt die *`barplot()`*-Funktion auch mit der *`table()`*-Funktion, welche sich auf die Variable *Autistisch* im Datensatz *data* bezieht. Mit *xlab=* und *ylab=* können die x-Achse (*xlab*) bzw. die y-Achse (*ylab*) beschriftet werden. Mit *main* wird die Gesamtüberschrift der Abbildung festgelegt und mit *ylim* kann festgelegt werden, von wo bis wo die y-Achse reichen soll (in diesem Fall von 0 bis 200).

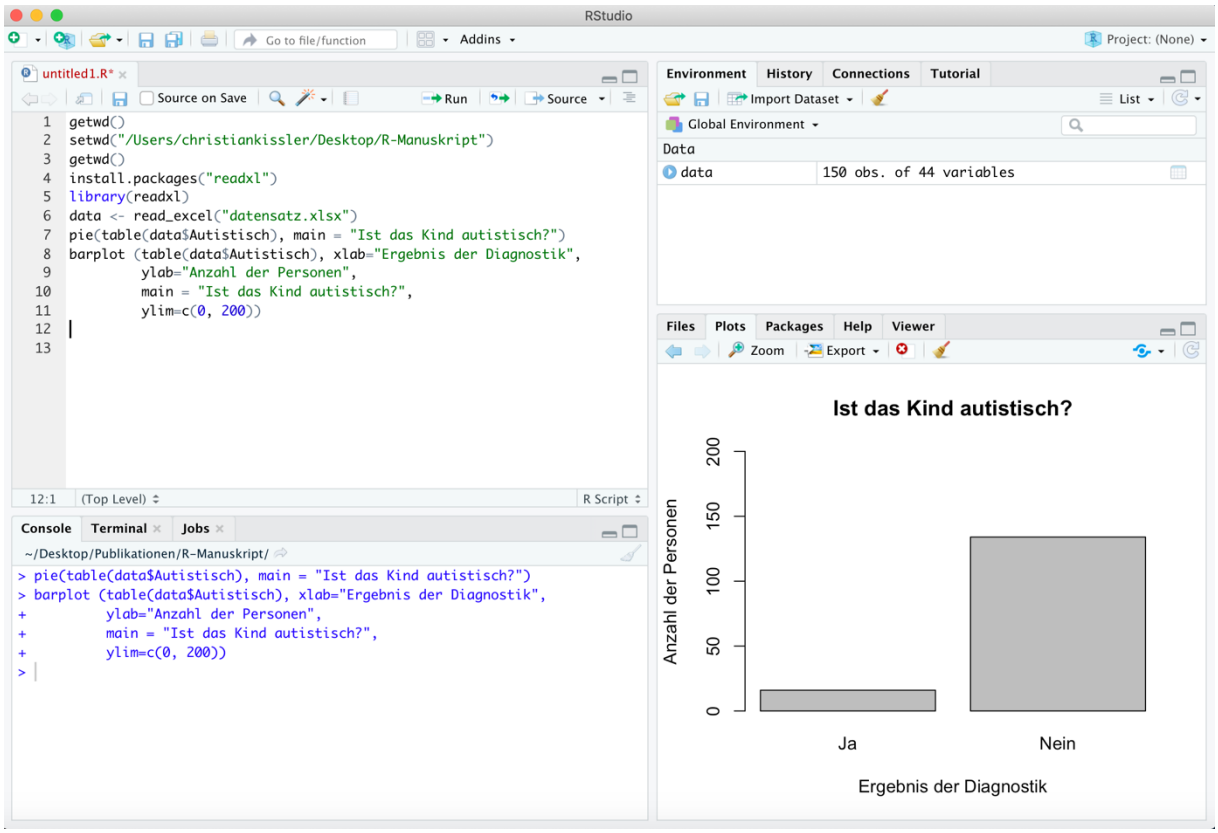

Abbildung 65: Erstellung eines Balkendiagramms mit der barplot()-Funktion

Mit der Funktion `hist()` kann ein Histogramm erzeugt werden. Im Folgenden wird ein Histogramm für die gemessene *Mathematikleistung* der Proband:innen im Datensatz *data* erzeugt. Dies ist über den folgenden Befehl möglich:

```
hist(data$Matheleistung, xlab="Leistung in Mathematik",
     ylab="Anzahl der Personen", main = "Verteilung der
     Mathematikleistung", ylim=c(0, 60), labels=TRUE)
```

Wichtig ist an dieser Stelle festzuhalten, dass die *hist()*-Funktion nicht auf einer `table()`-Funktion aufbaut - im Gegensatz zur `pie()`- und `barplot()`-Funktion. Stattdessen beginnt die `hist()`-Funktion direkt mit einem Bezug auf den Datensatz und die Variable, die angesteuert werden soll, um das Histogramm zu erzeugen. Die Beschriftung der x-Achse mit *xlab* bzw. der y-Achse mit *ylab* und die Erstellung einer Gesamtüberschrift mit *main* funktioniert ebenso wie beim Balkendiagramm. Auch die Skalierung der y-Achse mit *ylim* erfolgt so, wie es beim Balkendiagramm der Fall ist. Darüber hinaus kann mit *labels=TRUE* eingestellt werden, dass über den Flächen des Histogramms angezeigt wird, wie viele Personen einem spezifischen Teilbereich zuzuordnen sind (Abbildung 66). Dies erleichtert in vielen Fällen die Interpretation des Histogramms.

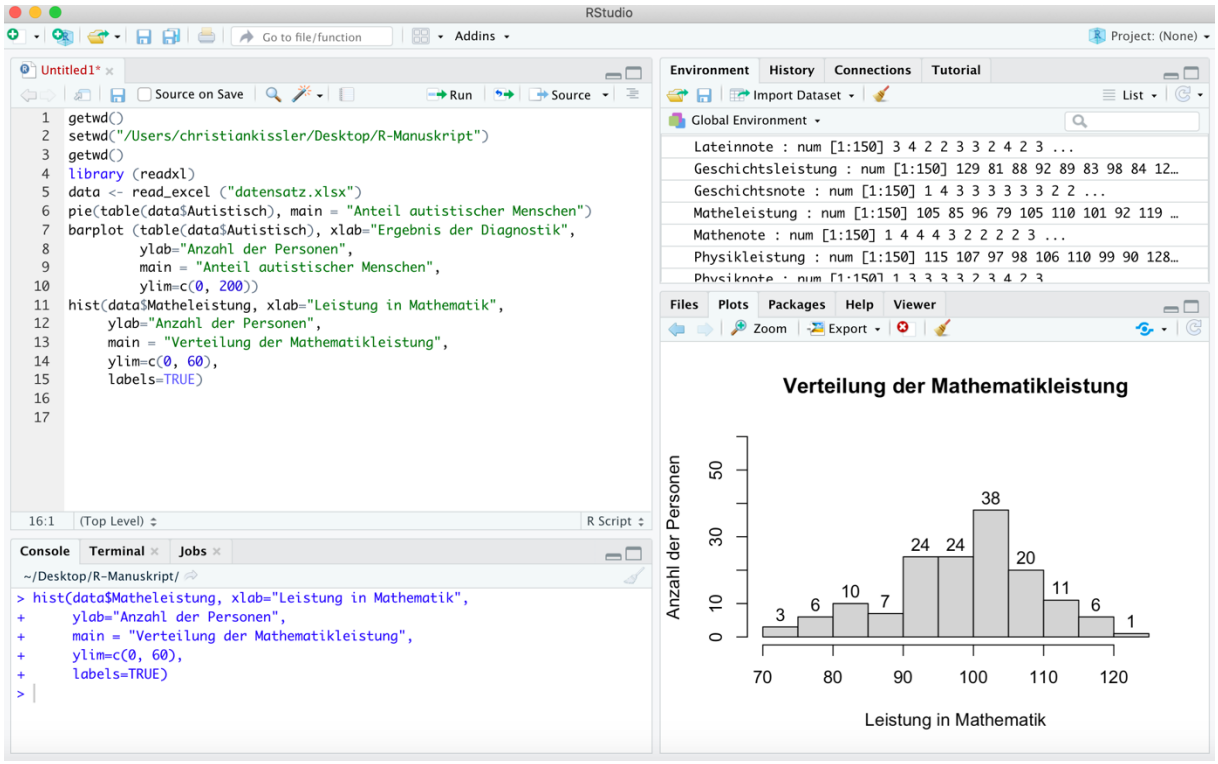

Abbildung 66: Erstellung eines Histogramms mit der hist()-Funktion

Eine andere gängige Möglichkeit zur Darstellung der Verteilung von Daten ist der Boxplot, da dieser Darstellungsweise viele relevante Informationen entnommen werden können. Abbildung 67 ist ein solcher Boxplot zur Variable *Mathematikleistung* im Datensatz *data* zu entnehmen.

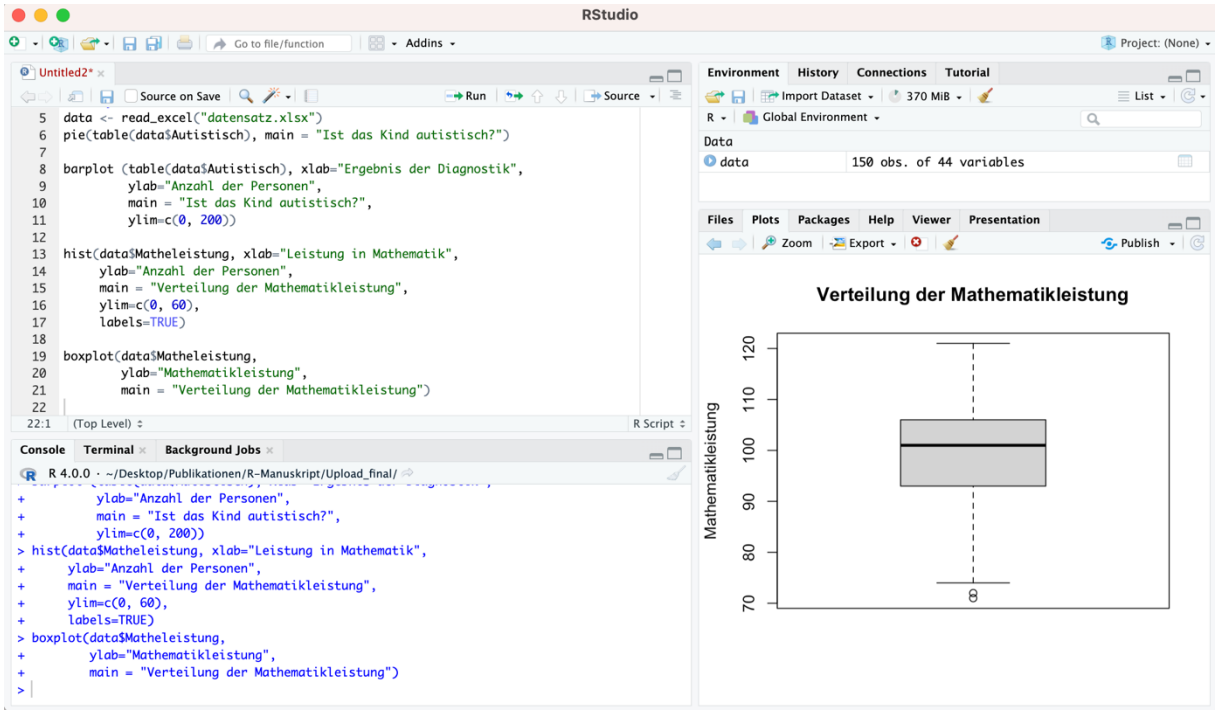

Abbildung 67: Erzeugung eines Boxplots

Dieser Boxplot wurde mit dem folgenden Befehl erzeugt:

```
boxplot(data$Matheleistung,
     ylab="Mathematikleistung",
     main = "Verteilung der Mathematikleistung")
```

Abbildung 68 ist eine Beschriftung zum Boxplot aus Abbildung 67 zu entnehmen, um den Boxplot interpretieren zu können. Es ist erkennbar, dass der schwarze Strich in der Mitte den Median markiert und auch andere zentrale Werte wie spezifische Quantile, Maxima und Minima sowie Ausreißer/ Extremwerte abgelesen werden können.

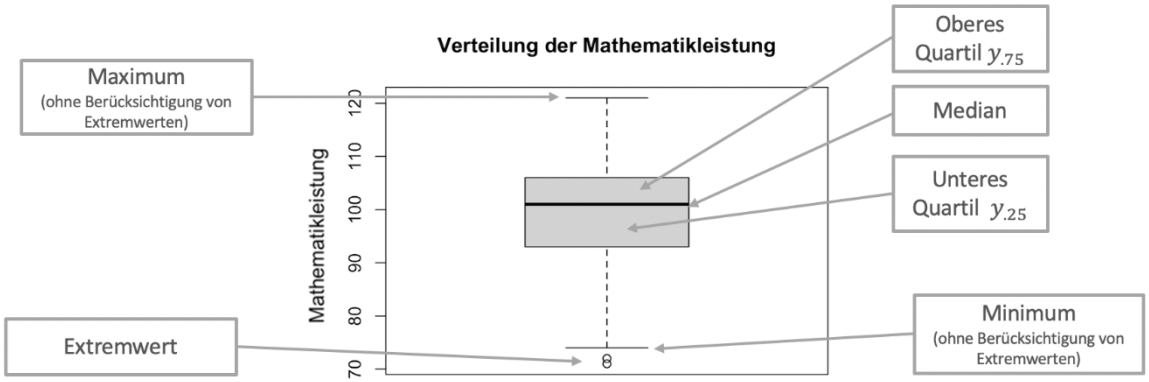

Abbildung 68: Interpretation eines Boxplots

3.6 Standardisierungen

Häufig werden Ausprägungen von Variablen auf sehr unterschiedlichen Skalen erhoben, wobei den einzelnen Ausprägungen verschiedene Bedeutungen zugschrieben werden können: So kann es zum Beispiel sein, dass Proband:innen einerseits ihr Wohlbefinden auf einer Skala von 1 (sehr niedrig) bis 9 (sehr hoch) einschätzen sollen, andererseits aber ihre Sportlichkeit auf einer Skala von 0 (sehr niedrig) bis 100 (sehr hoch) selbst bewerten sollen. Die erhobenen Rohwerte sind dann schwer zu interpretieren und es ist nicht leicht, sich eine Vorstellung von den Daten zu machen sowie diese miteinander in Beziehung zu setzen: während der Wert 9 bei Wohlbefinden nämlich ein sehr hohes Wohlbefinden bedeuten würde, würde ein Wert von 9 bei der Variable Sportlichkeit für eine tendenziell sehr niedrige Sportlichkeit stehen. Dieses Problem bezüglich der Interpretation von Variablenausprägungen kann durch *Standardisierungen* (teilweise) behoben werden. Die Standardisierung von Daten stellt nämlich eine Möglichkeit dar, den erhobenen Daten mehr Informationsgehalt zu geben und mehr

Vergleichbarkeit zu gewährleisten. Besonders gängig sind dabei drei Arten der Standardisierung: 1) Die z-Standardisierung, 2) die Standardisierung in Form von T-Werten und 3) die Standardisierung auf der IQ-Skala. Hierzu ist bereits jetzt festzuhalten, dass 2) und 3) auf z-Standardisierungen aufbauen, was im Folgenden aber auch noch dargestellt wird.

Bei der z-Standardisierung werden die Daten so aufbereitet, dass anschließend der Wert 0 der Mittelwert des Datensatzes ist und der Wert -1 exakt eine Standardabweichung unter dem Mittelwert liegt, während der Wert 1 eine Standardabweichung über dem Mittelwert liegt. Wenn im obigen Beispiel bezüglich der Bewertung der Sportlichkeit (Skala von 0 bis 100) eines Individuums bei einem individuellen Wert von 62 nicht klar gewesen wäre, ob dieser Wert als hoch oder niedrig im Verhältnis zu anderen Personen zu bewerten wäre, so würde der z-Wert von -0.25, der sich in diesem imaginären Beispiel nach einer z-Standardisierung des Datensatzes aus der 62 ergeben hätte, aussagen, dass der Proband sich im Vergleich zu den anderen Proband:innen als unterdurchschnittlich sportlich einschätzt, da der Wert negativ ist: Genauer gesagt würde sein individueller Wert 0.25 Standardabweichungen unter dem Mittelwert liegen.

Die z-Standardisierung kann in `RStudio` mithilfe eines einfachen Befehls vorgenommen werden: `scale()`. Eine Variable, die auf diese Weise z-standardisiert wurde, sollte nach der z-Standardisierung als neue Variable im Datensatz abgespeichert werden, damit diese neu berechnete Variable für weitere Berechnungen bereitsteht bzw. verwendet werden kann. Im Folgenden wird das prinzipielle Verfahren zur z-Standardisierung anhand der Variable *Sozialverhalten* aus dem Datensatz *data* aufgezeigt. Mit dem folgenden *Code* kann die z-Standardisierung vorgenommen und gleichzeitig die neu erzeugte Variable unter dem Namen *z_Sozialverhalten* im Datensatz *data* abgespeichert werden:

```
data$z_Sozialverhalten <- scale(data$Sozialverhalten)
```

Mit den Befehlen `mean(data$z_Sozialverhalten, na.rm=TRUE)` und `sd(data$z_Sozialverhalten, na.rm=TRUE)` kann nun überprüft werden, ob der Mittelwert der neuen Variable tatsächlich bei 0 liegt und ob die Standardabweichung 1 beträgt. Insofern dies der Fall ist, ist die z-Standardisierung gelungen. Denn wie bereits oben beschrieben wurde, zeichnen sich z-standardisierte Variablen dadurch aus, dass ihr Mittelwert bei 0 liegt und die Standardabweichung 1 beträgt. In der Konsole von Abbildung 69 sind der berechnete Mittelwert und die berechnete Standardabweichung der Variable *z_Sozialverhalten* zu sehen. Bei dem Ergebnis der Mittelwertberechnung ist zu erkennen, dass der Mittelwert nicht exakt 0 beträgt. Stattdessen steht dort eine andere Zahl, die korrekt interpretiert werden muss.

1.000056e-16 bedeutet dabei nichts anderes als 1.000056 * 10^{-16} bzw. ausgeschrieben 0.0000000000000001000056: Es handelt sich also um eine extremst kleine Zahl, die quasi mit 0 gleichzusetzen ist. Darüber hinaus beträgt die Standardabweichung der neu erzeugten Variable *z_Sozialverhalten* exakt 1. Die z-Standardisierung wurde also korrekt durchgeführt.

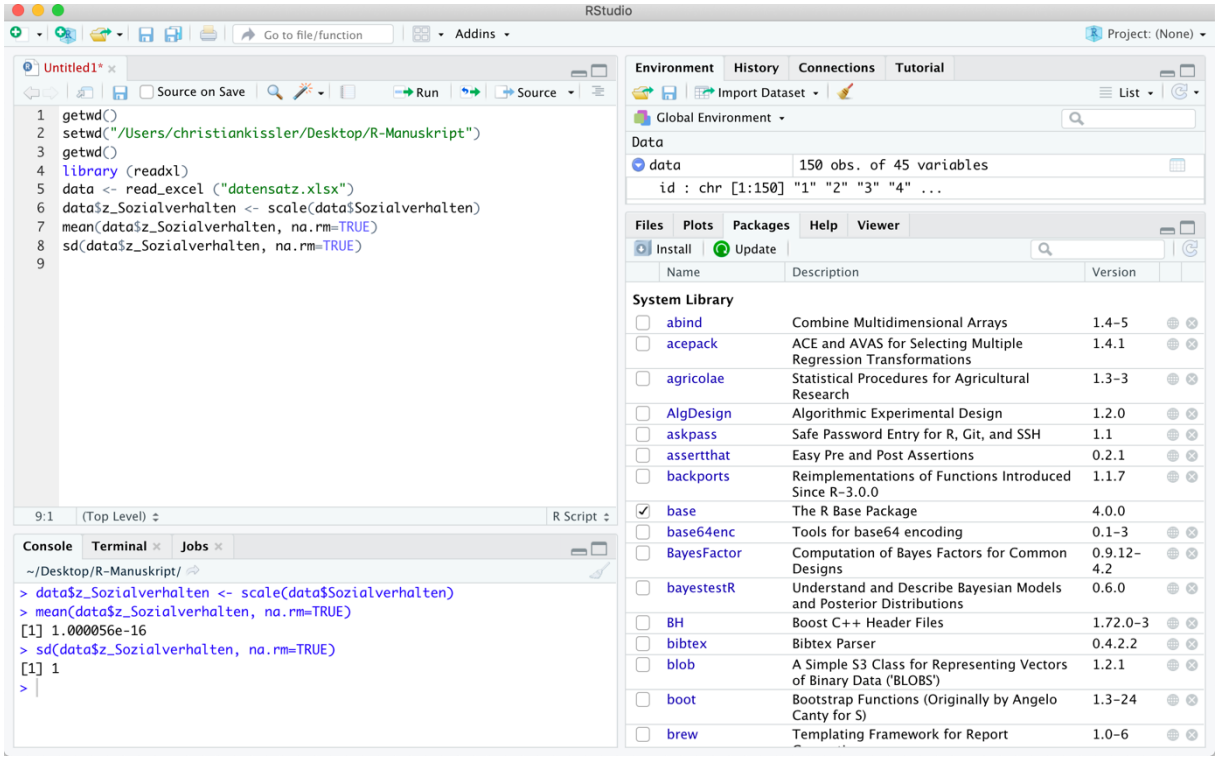

Abbildung 69: z-Standardisierung von Variablen

Wie bereits beschrieben wurde, gibt es auch andere Arten der Standardisierung: Die IQ-Standardisierung und die T-Standardisierung. IQ-standardisierte Variablen zeichnen sich dadurch aus, dass ihr Mittelwert bei 100 liegt und die Standardabweichung 15 beträgt. T-standardisierte Variablen hingegen weisen einen Mittelwert von 50 und eine Standardabweichung von 10 auf. Auf der Grundlage von z-standardisierten Variablen lassen sich relativ mühelos mit den folgenden beiden *Code*-Zeilen T- bzw. IQ-standardisierte Variablen erzeugen:

```
data$IQ_Sozialverhalten <- 100 + 15 * data$z_Sozialverhalten
data$T_Sozialverhalten <- 50 + 10 * data$z_Sozialverhalten
```

Die Variable *IQ_Sozialverhalten* wurde mit dem obigen Befehl auf der IQ-Skala standardisiert und die Variable *T_Sozialverhalten* ist T-standardisiert. Auch hier kann mittels der Anwendung der `mean()`- und `sd()`-Funktionen auf die neu berechneten Variablen überprüft werden, ob die jeweiligen Standardisierungen gelungen sind.

3.7 Zusammenhänge und Zusammenhangsmaße

In diesem Kapitel wird dargestellt, wie Variablen untereinander auf Zusammenhänge untersucht werden können. Grafisch kann der Zusammenhang zwischen zwei Variablen mit einem Scatterplot veranschaulicht sowie dargestellt werden. Jedoch sind diese Scatterplots mit dem bloßen Auge nicht immer einfach zu interpretieren. Im Folgenden soll im Datensatz *data* der Zusammenhang zwischen *Matheleistung* und *IQ* untersucht werden (zunächst grafisch mit einem Scatterplot, dann rechnerisch durch die Berechnung eines Korrelationskoeffizienten). Ein solcher Scatterplot kann mit dem folgenden *Code* erzeugt werden:

```
plot(data$IQ, data$Matheleistung)
```

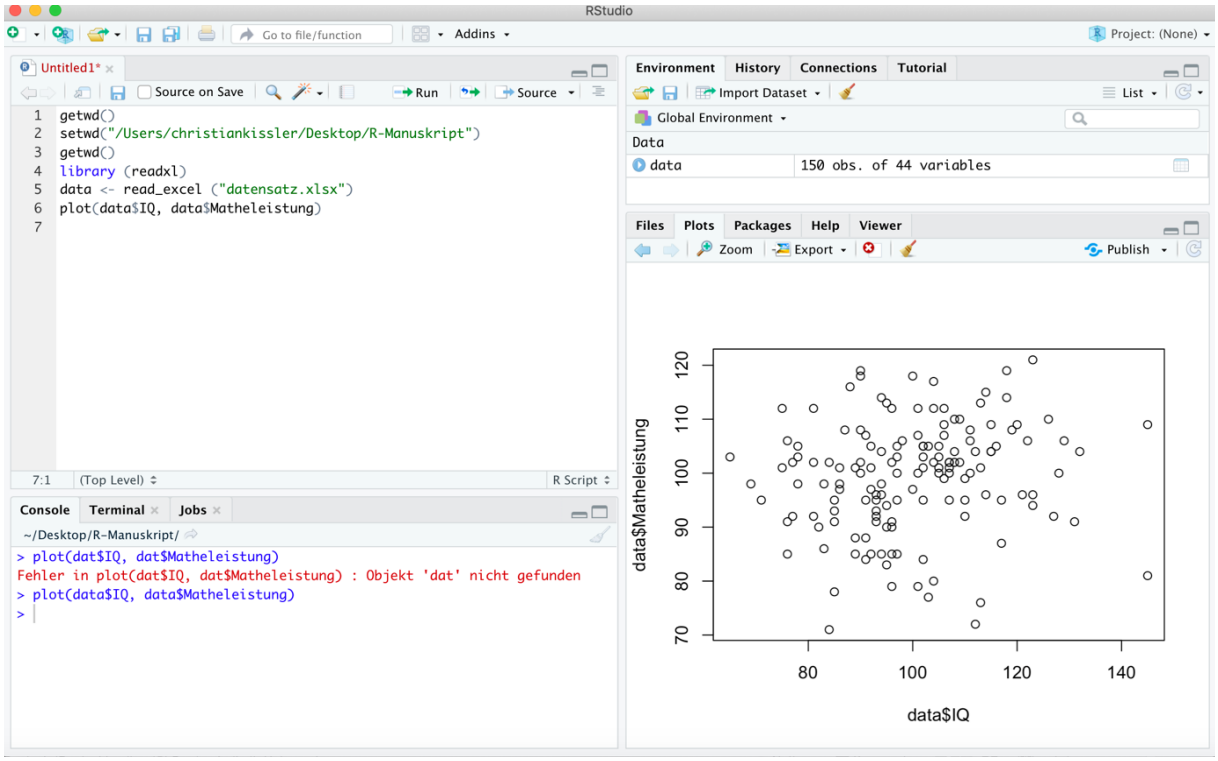

Abbildung 70: Erzeugung eines Scatterplots

Die `plot()`-Funktion ist also so aufgebaut, dass in ihr die beiden Variablen, die grafisch dargestellt werden sollen, hintereinander in die Klammer geschrieben werden müssen. Der entsprechende Output ist der Abbildung 70 zu entnehmen.

In Abbildung 70 ist zu erkennen, dass die Variable, die in den Klammern der `plot()`-Funktion zuerst benannt wurde (*IQ*) auf der X-Achse und die danach benannte Variable (*Matheleistung*) auf der y-Achse abgetragen wurde. Jeder Punkt im Koordinatensystem stellt eine individuelle IQ-Matheleistung-Zuordnung dar. Mit bloßem Auge lässt sich nur schwer erkennen, ob es hier einen (bedeutenden bzw. überzufälligen) Zusammenhang zwischen den beiden Variablen gibt, auch wenn die Daten einen schwachen „je-mehr-desto-mehr"-

Zusammenhang zwischen den beiden Variablen zumindest grob andeuten. Es wird deutlich, dass die Existenz eines Zusammenhangs nicht immer zweifelsfrei über eine bloße visuelle Inspektion der Daten überprüft werden kann.

Ein bekanntes bzw. gängiges Maß, um den (linearen) Zusammenhang zwischen zwei Variablen zu quantifizieren, ist der Korrelationskoeffizient. Dieser Korrelationskoeffizient kann für die Variablen *Matheleistung* und *IQ* im Datensatz *data* mit dem folgenden Code berechnet werden:

```
cor(data$Matheleistung, data$IQ, use="pairwise.complete.obs",
     method = "pearson")
```

Die `cor()`-Funktion dient also prinzipiell zur Berechnung des Korrelationskoeffizienten. Innerhalb der Klammern werden zunächst die beiden Variablen, für die der Korrelationskoeffizient berechnet werden soll, durch ein Komma getrennt benannt. Anschließend wird hinter einem weiteren Komma angegeben, wie mit fehlenden Werten umgegangen werden soll: Mit dem Befehl *use = "pairwise.complete.obs"* werden fehlende Werte paarweise aus der Berechnung ausgeschlossen. Dieses Argument ist jedoch optional: Wenn nämlich keine fehlenden Werte vorliegen, müsste es nicht angegeben werden. Das Argument *method* entscheidet, wie der Korrelationskoeffizient konkret berechnet werden soll (hier gibt es unterschiedliche Ansätze). Prinzipiell kann eine Pearson-Korrelation, eine Spearman-Korrelation oder eine Kendall-Korrelation berechnet werden (siehe Abbildung 71). Während die Pearson-Korrelation für die Berechnung eines Zusammenhangs von zwei intervallskalierten Variablen geeignet ist, insofern eine bivariate Normalverteilung der Merkmale als Voraussetzung erfüllt ist (Döring & Bortz, 2016), können Spearman-Korrelationen insbesondere für ordinale Daten verwendet werden oder zum Einsatz kommen, wenn die Voraussetzungen für eine Pearson-Korrelation nicht erfüllt sind (Holling & Gediga, 2011).

Konventionsgemäß wird bei einem Korrelationskoeffizienten von mindestens .10 von einer kleinen bzw. schwachen Korrelation, bei einem Korrelationskoeffizienten von mindestens .30 von einer mittleren Korrelation und bei einem Korrelationskoeffizienten von mindestens .50 von einer großen bzw. starken Korrelation gesprochen (Cohen, 1988; Holling & Gediga, 2011). Unabhängig von der gewählten Methode zur Berechnung des Korrelationskoeffizienten würde also der Zusammenhang von *Matheleistung* und *IQ* im Datensatz *data* als schwacher Zusammenhang klassifiziert werden, weil die Korrelationskoeffizienten stets über .10, aber unter .30 liegen (Abbildung 71).

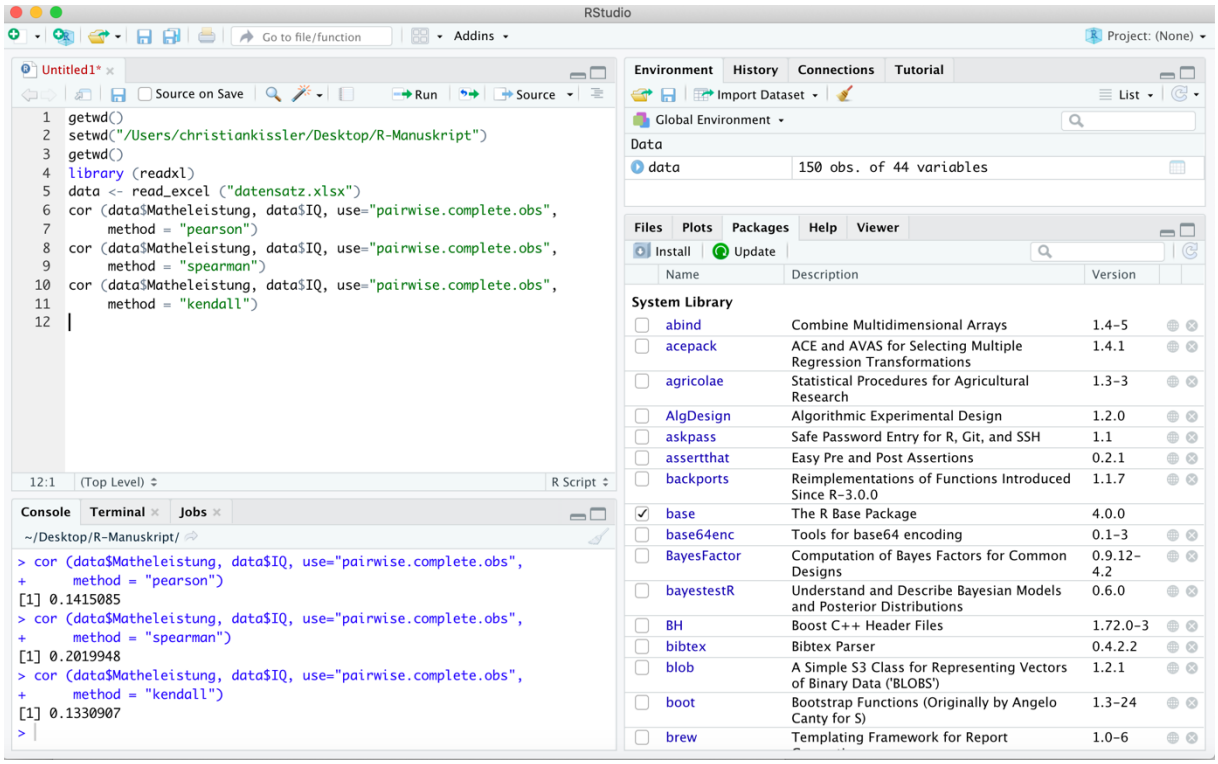

Abbildung 71: Korrelationskoeffizienten berechnen

In Kapitel 4 wird noch einmal vertiefend auf das Konzept der Korrelation eingegangen. Dort wird dann der weiterführenden Frage nachgegangen, ob ein solcher Zusammenhang bedeutungsvoll bzw. überzufällig (fachsprachlich: signifikant) ist.

4 Inferenzstatistik

Im Folgenden wird dargestellt, wie Sie die inferenzstatistische Datenanalyse in `RStudio` vornehmen können. Zur statistischen Datenauswertung ist die Verwendung spezifischer Befehle notwendig, die Sie wie Vokabeln lernen können. Alle Befehle, die im Folgenden verwendet werden, werden vorgestellt und erklärt. Viele nützliche Informationen zum Umgang mit `RStudio` sowie zur Datenauswertung können Sie auch auf der Internetseite von `RStudio` in Form eines „Cheat Sheets" einsehen: https://www.rstudio.com/resources/cheatsheets/. In diesem Kapitel wird erneut der Datensatz, der bereits in Kapitel 2.9 eingelesen wurde, verwendet. Zur Vertiefung der Konzepte und Theorien, auf denen die in diesem Kapitel dargestellten Informationen zur praktischen Anwendung statistischer Verfahren in `R` bzw. `RStudio` basieren, eignen sich insbesondere die Werke von Holling und Gediga (2015), Döring und Bortz (2016) sowie Bortz und Schuster (2010). Falls Interesse besteht, sich über die im Folgenden dargestellten Inhalte hinaus mit der Statistik in `R` zu beschäftigen, so ist insbesondere das Werk von Navarro (2016) zu empfehlen.

4.1 Zufallsvariablen und Verteilungen

Um die folgenden Kapitel inhaltlich verstehen zu können, müssen zunächst einige theoretische Grundlagen dargestellt und zentrale Begriffe eingeführt werden. Von einer Zufallsvariable wird dann gesprochen, wenn ein spezifischer Wert vom Zufall abhängig ist. Wenn es sich bei der Zufallsvariable um eine Zahl handelt, so ist von einer Zufallszahl die Sprache. Ein Beispiel für eine solche Zufallsvariable bzw. Zufallszahl ist die Augenzahl beim zweimaligen Würfeln mit einem entsprechenden sechsseitigen (fairen) Würfel. Alle möglichen Ergebnisse, die dabei resultieren können, bilden den Ergebnisraum (als Ω bezeichnet). Im Falle des zweimaligen Würfelns würde der Ergebnisraum folgendermaßen aussehen:

$$\Omega = \{(1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (2,1), \ldots, (6,5), (6,6)\}$$

Im Ergebnisraum stellt beispielsweise (1,2) ein Paar dar, wobei die erste Zahl das Ergebnis des einen Würfelns (in diesem Beispiel: 1) und die zweite Zahl das Ergebnis des zweiten Würfelns (in diesem Beispiel: 2) darstellt. Die Summe der gewürfelten Augen – bei (1,2) also 3 – ist eine Zufallsvariable *Y*: Der Wertebereich der Zufallsvariable *Y* reicht dabei in diesem Fall von 2 bis 12. Denn die kleinste Augensumme, die gewürfelt werden kann, beträgt 2 (gewürfelt wurden dann eine 1 und eine weitere 1) und die größte Augensumme, die gewürfelt werden kann, beträgt 12 (gewürfelt wurden dann eine 6 und eine weitere 6).

Dabei sind den unterschiedlichen möglichen Ausprägungen der Zufallsvariable *Y* unterschiedliche Wahrscheinlichkeiten zuzuordnen, da es für das Auftreten der unterschiedlichen Ergebnisse/ Augensummen beim Würfeln stets unterschiedlich viele Kombinationsmöglichkeiten gibt. Dies wird in Tabelle 6 dargestellt.

In Tabelle 6 ist zu sehen, dass die gewürfelten Augensummen von 2 bis 12 reichen können. Dabei sind jeder Augensumme die möglichen Ergebnisse aus dem Ergebnisraum zugeordnet, die zu dieser Augensumme führen können. Während es für die Augensummen 2 und 12 nur jeweils eine Kombinationsmöglichkeit gibt, gibt es für die Augensumme 7 sogar 6 mögliche Kombinationen. Insgesamt ergeben sich 36 mögliche Ergebnisse beim Würfeln mit zwei Würfeln bzw. beim zweimaligen Würfeln. Dies bedeutet, dass die Wahrscheinlichkeit für die Augensumme 2 bei 1/36 (also 1 von 36) liegt und die Wahrscheinlichkeit für eine Augensumme von 7 bei 6/36 (6 von 36). Die Wahrscheinlichkeit wird nämlich berechnet, indem man die möglichen Kombinationen für ein spezifisches Ereignis (für das Eintreten der Augensumme 7: 6 Möglichkeiten) durch die Anzahl aller Möglichkeiten, zu denen es insgesamt kommen kann, (beim zweimaligen Würfeln: 36) dividiert.

Tabelle 6: Zweimaliges Würfeln - Ergebnisse und Wahrscheinlichkeiten

Gewürfelte Augensumme	Mögliche Paare aus dem Ergebnisraum	Wahrscheinlichkeit
2	(1,1)	1/36
3	(1,2), (2,1)	2/36
4	(1,3), (2,2), (3,1)	3/36
5	(1,4), (2,3), (3,2), (4,1)	4/36
6	(1,5), (2,4), (3,3), (4,2), (1,5)	5/36
7	(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)	6/36
8	(2,6), (3,5), (4,4), (5,3), (6,2)	5/36
9	(3,6), (4,5), (5,4), (6,3)	4/36
10	(4,6), (5,5), (6,4)	3/36
11	(5,6), (6,5)	2/36
12	(6,6)	1/36

Grundlagen der deskriptiven Statistik wurden bereits in Kapitel 3 dargestellt. Dort wurden bei einer Stichprobe insbesondere Mittelwerte und die Variabilität (beschrieben durch die Varianz bzw. Standardabweichung) als Maße verwendet, um die Stichprobe und somit die Lage sowie

Verteilung der Daten zu beschreiben. Bei einer Zufallsvariable *Y* wird zur Beschreibung der Lage jedoch anstelle des Mittelwerts der *Erwartungswert* oder der „*Populationsmittelwert"* verwendet ((durch *E(Y)* oder μ symbolisiert); zur Beschreibung der Variabilität werden bei einer Zufallsvariable Y die *Varianz* (durch *Var(Y)* oder σ^2 symbolisiert) bzw. die *Standardabweichung* (durch σ symbolisiert) verwendet. Der Erwartungswert einer Zufallsvariable *Y* kann dabei als der Mittelwert der Versuchsausgänge interpretiert werden, der sich bei (hypothetisch) unendlich vielen unabhängigen Wiederholungen eines durch *Y* beschriebenen Zufallsexperiments ergibt. Dieser Effekt wird durch das *Gesetz der großen Zahlen* beschrieben.

Das Gesetz der großen Zahlen beschreibt, dass, wenn wiederholt mit zwei Würfeln gewürfelt wird und die resultierenden Augensummen immer systematisch notiert werden, sich die relativen Häufigkeiten (in diesem Fall folgendermaßen berechnet: „Wie häufig wurde eine spezifische Augensumme gewürfelt" geteilt durch „wie häufig wurde insgesamt gewürfelt") bezüglich ihres tatsächlichen Auftretens spezifischer Augensummen den erwarteten Wahrscheinlichkeiten für das Auftreten ebendieser Augensummen sukzessive annähern. Das bedeutet für das zweimalige Würfeln konkret, dass die Augensumme 7 am häufigsten auftreten sollte und in etwa 16,67% der Fälle (=6/36) auftreten sollte. Die Augensumme 12 sollte stattdessen nur in ca. 2,78% der Fälle (1/36) auftreten. Sollte nach sehr häufigem Würfeln die relative Häufigkeit für die Augensumme 12 beispielsweise bei 15% (und somit deutlich höher als 2,78%) liegen, so würde die Frage im Raum stehen, ob die Würfel vielleicht gezinkt (also: manipuliert) sind, sodass sie häufiger die 6 zeigen, als es bei einem fairen Würfel eigentlich der Fall sein sollte. Sollte ein Ereignis also viel häufiger (oder seltener) eintreten, als es zu erwarten wäre, so müsste überlegt werden, ob die Grundannahme (jeder Würfel zeigt jede Fläche mit gleicher Wahrscheinlichkeit) überhaupt korrekt ist. Dies ist eine wichtige Erkenntnis, denn auf genau dieser Überlegung basiert die Prüfung von wissenschaftlichen Hypothesen.

Entscheidungen über die Haltbarkeit von Hypothesen orientieren sich nämlich prinzipiell an einem Vergleich von dem, 1) was bei Gültigkeit einer Hypothese beobachtet werden sollte und 2) was tatsächlich beobachtet wird. Dabei wird überprüft, ob das, was bei Gültigkeit einer Hypothese beobachtet werden sollte, und das, was tatsächlich beobachtet wird, miteinander vereinbar ist. Die Entscheidungsregel lautet dabei im Prinzip folgendermaßen: Fällt der für eine Stichprobe berechnete Wert einer Statistik in einen Bereich von Werten, der bei Gültigkeit einer Hypothese unwahrscheinlich ist, wird mit einer bestimmten Fehlerwahrscheinlichkeit eine Entscheidung gegen die Hypothese getroffen (Hypothese wird verworfen).

Dies mag an dieser Stelle vielleicht etwas kryptisch klingen, wird aber im Folgenden genauer erklärt werden. Bei der Prüfung von Hypothesen wird mit sogenannten Signifikanzniveaus gearbeitet, die der Entscheidung dienen, wann die beobachteten Daten nicht oder nicht ausreichend gut zur Nullhypothese (auch H0 genannt) passen. Die Nullhypothese unterstellt dabei regelmäßig, dass es keinen Effekt bzw. keinen Unterschied (z.B. zwischen zwei Gruppen) gibt. Die Alternativhypothese (auch H1 oder Ha genannt) unterstellt hingegen das Gegenteil – zum Beispiel, dass es einen Effekt bzw. Unterschied gibt. In der klassischen (frequentistischen) Statistik wird üblicherweise die Nullhypothese auf ihre Plausibilität überprüft. Dies bedeutet: Wenn sich herausstellt, dass die erhobenen Daten nicht zur Nullhypothese passen, so wird die Nullhypothese verworfen und stattdessen die Alternativhypothese angenommen. Denn die Alternativhypothese ist die einzige Hypothese, die übrig bleibt, wenn die Nullhypothese nicht zutreffen kann und aufgrund der Datenlage als falsch anerkannt werden muss. Sollten die vorliegenden Daten jedoch nicht zur Ablehnung der Nullhypothese genügen, so wird die Nullhypothese beibehalten. Geprüft wird also, ob die Beobachtungen bzw. die tatsächlich vorliegenden Daten gegen die Nullhypothese sprechen, bzw. ob die beobachteten Daten unter der Annahme, dass die Nullhypothese zutrifft, unwahrscheinlich sind. Die Logik lautet also: Wenn die Nullhypothese falsch ist bzw. abzulehnen ist, dann gilt die Alternativhypothese bzw. dann muss die Alternativhypothese richtig sein.

Zur fundierten Überprüfung der Nullhypothese wird eine Statistik verwendet, die als Teststatistik oder Prüfgröße bezeichnet wird. Die Teststatistik setzt üblicherweise einen auf beobachteten Daten basierenden Wert (z. B. beobachteter Mittelwert) mit einem unter der Nullhypothese erwarteten Wert (z. B. erwarteter Mittelwert) in Relation, z. B. in Form einer Differenz. Diese Differenz zwischen Beobachtung und Erwartung wird bei der Teststatistik dann üblicherweise in Relation zur erwarteten Streuung des beobachteten Wertes gesetzt, um zu entscheiden, ob die Abweichung (beobachtet vs. erwartet) besonders groß oder eher klein ist. Ergibt sich aus den Beobachtungen ein Wert der Teststatistik, der bei Gültigkeit der Nullhypothese sehr unwahrscheinlich ist, so wird die Nullhypothese zugunsten der Alternativhypothese abgelehnt bzw. verworfen; andernfalls wird die Nullhypothese (vorläufig) beibehalten. Vor allem bei computergestützten Analysen wird im Rahmen sogenannter frequentistischer Analysen häufig der *p*-Wert für eine Testentscheidung verwendet und nicht überprüft, ob die Teststatistik selbst den kritischen Wert (= der Wert, der definiert, ab wann die Nullhypothese abzulehnen ist) überschreitet. Es besteht jedoch ein Zusammenhang zwischen Teststatistik, Größe der Stichprobe und *p*-Wert.

Der *p*-Wert gibt die Wahrscheinlichkeit an, den aus den Daten berechneten Wert der Teststatistik oder einen noch extremeren Wert der Teststatistik unter Gültigkeit der Nullhypothese zu beobachten. Ist der *p*-Wert kleiner als das Signifikanzniveau α (in der Regel ist $\alpha = 0.05$, selten aber auch $\alpha = 0.01$) wird die Nullhypothese verworfen und ansonsten (also: der *p*-Wert ist größer als 0.05) beibehalten. Das Verwerfen der Nullhypothese bedeutet also, dass die Gültigkeit der Nullhypothese mit Blick auf die erhobenen Daten sehr unwahrscheinlich ist, womit die Alternativhypothese als einzige Alternative zur Nullhypothese gültig sein muss. Das Beibehalten der Nullhypothese darf dabei jedoch nicht als ein Beweis der Nullhypothese interpretiert werden. Aus einer Beibehaltung der Nullhypothese ergibt sich nämlich nicht automatisch Evidenz für die Nullhypothese selbst.

Bei den meisten Testverfahren, die im Folgenden dargestellt werden, wird die Nullhypothese getestet und dann mit Blick auf den *p*-Wert verworfen oder beibehalten. Wenn die Nullhypothese verworfen wird (= die Daten sind unter Gültigkeit der Nullhypothese so unwahrscheinlich, dass sie nicht gültig sein kann), so wird die Alternativhypothese angenommen und es gibt Evidenz für die Alternativhypothese: Denn die Alternativhypothese ist komplementär zur Nullhypothese formuliert, sodass nur die Alternativhypothese als Hypothese übrigbleibt, wenn die Nullhypothese nicht zutreffen kann. Was dies konkret bedeutet, wird anhand mehrerer Beispiele in den folgenden Kapiteln dargestellt werden. Wichtig ist an dieser Stelle jedoch noch zu betonen, dass Alternativhypothesen und Nullhypothesen stets komplementär formuliert sein müssen und gerichtet sowie ungerichtet formuliert sein können. Was dies mit Blick auf empirische Daten konkret bedeutet, wird in Kapitel 4.2 anhand eines Beispiels dargestellt.

4.2 Gauß-Test

Der Gauß-Test stellt ein Testverfahren dar, bei dem anhand eines Stichproben-Mittelwerts Hypothesen über den Erwartungswert der Grundgesamtheit, aus der die Stichprobe stammt, überprüft werden. So wird im Folgenden untersucht, inwiefern Hypothesen über den allgemeinen Gesundheitszustand (Variable: *Allgemeine_Gesundheit* im Datensatz *data*) der Grundgesamtheit, aus der die Stichprobe stammt, mithilfe der vorliegenden Daten verifiziert werden können. Dabei sprechen hohe Werte bei der Variable *Allgemeine_Gesundheit* für eine hohe allgemeine Gesundheit und niedrige Werte für eine niedrige allgemeine Gesundheit.

Um den allgemeinen Gesundheitszustand zu erheben, wurde mit jeder Person aus der Stichprobe ein Test durchgeführt. Aus dem Manual des verwendeten Tests wissen wir, dass der Test in der Gesamtbevölkerung normiert ist: 100 stellt in der Gesamtbevölkerung den

Mittelwert dar und die Standardabweichung in der Gesamtbevölkerung liegt bei 15. Somit ist der Test auf der IQ-Skala normiert.

Im Folgenden soll überprüft werden, ob die vorliegende Stichprobe aus einer Grundgesamtheit stammt, welche einen anderen Erwartungswert als 100 aufweist (also gesünder oder weniger gesund als die Gesamtbevölkerung ist). Die Alternativhypothese, die im Folgenden getestet wird, lautet also: Die Stichprobe stammt aus einer Grundgesamtheit, welche sich mit Blick auf ihre allgemeine Gesundheit von der Gesamtbevölkerung unterscheidet. Es handelt sich dabei um eine ungerichtete Alternativhypothese, da sie keine Richtung vorgibt: *Es wird davon ausgegangen, dass die Stichprobe aus einer Grundgesamtheit stammt, deren Erwartungswert* ***höher oder niedriger*** *als 100 ist*. Die zugehörige Nullhypothese ist komplementär zur Alternativhypothese formuliert und lautet: *Es gibt mit Blick auf die Variable allgemeine Gesundheit keinen Unterschied zwischen der Gesamtbevölkerung und der Grundgesamtheit, aus welcher die Stichprobe stammt*. Alternativhypothesen und Nullhypothesen können auch mithilfe einer Kurzschreibweise ausgedrückt werden. Die folgende Kurzschreibweise kann verwendet werden, um den zuvor geschilderten Sachverhalt auszudrücken:

- Nullhypothese: H_0: $\mu = 100$
- Alternativhypothese: H_1: $\mu \neq 100$

Um die Nullhypothese nun zu testen (es wird – wie bereits beschrieben wurde – immer die Nullhypothese getestet), muss zunächst das *Package BSDA* von Arnholt & Evans (2021) installiert und aktiviert werden. Anschließend kann mit der folgenden Syntax die Hypothesenprüfung durchgeführt werden:

```
z.test (data$Allgemeine_Gesundheit, alternative = „two.sided“,
     mu = 100, sigma.x = 15, conf.level=0.95)
```

`z.test()` ist dabei die Funktion, die benötigt wird, um einen Gauß-Test durchzuführen. In der Klammer steht dann zunächst, auf welche Variable aus welchem Datensatz sich der Test bezieht (*data$Allgemeine_Gesundheit*). Anschließend ist nach einem Komma aufgeführt, dass es sich um eine ungerichtete Hypothesenprüfung handelt (eine so genannte zweiseitige Testung): *alternative = „two.sided“*. Danach werden der bekannte Mittelwert (*mu =100*) und die bekannte Standardabweichung (*sigma.x= 15*) aufgeführt – Werte, die wir in diesem Fall aus dem oben genannten Manual übernommen haben. Schlussendlich wird das Signifikanzniveau genannt, das bei der Testung verwendet werden soll, wobei „*conf.level=0.95*“ einem

Signifikanzniveau von α = 0.05 entspricht. Das Ergebnis dieses Gauß-Tests sowie die zugehörige Syntax sind in Abbildung 72 zu sehen.

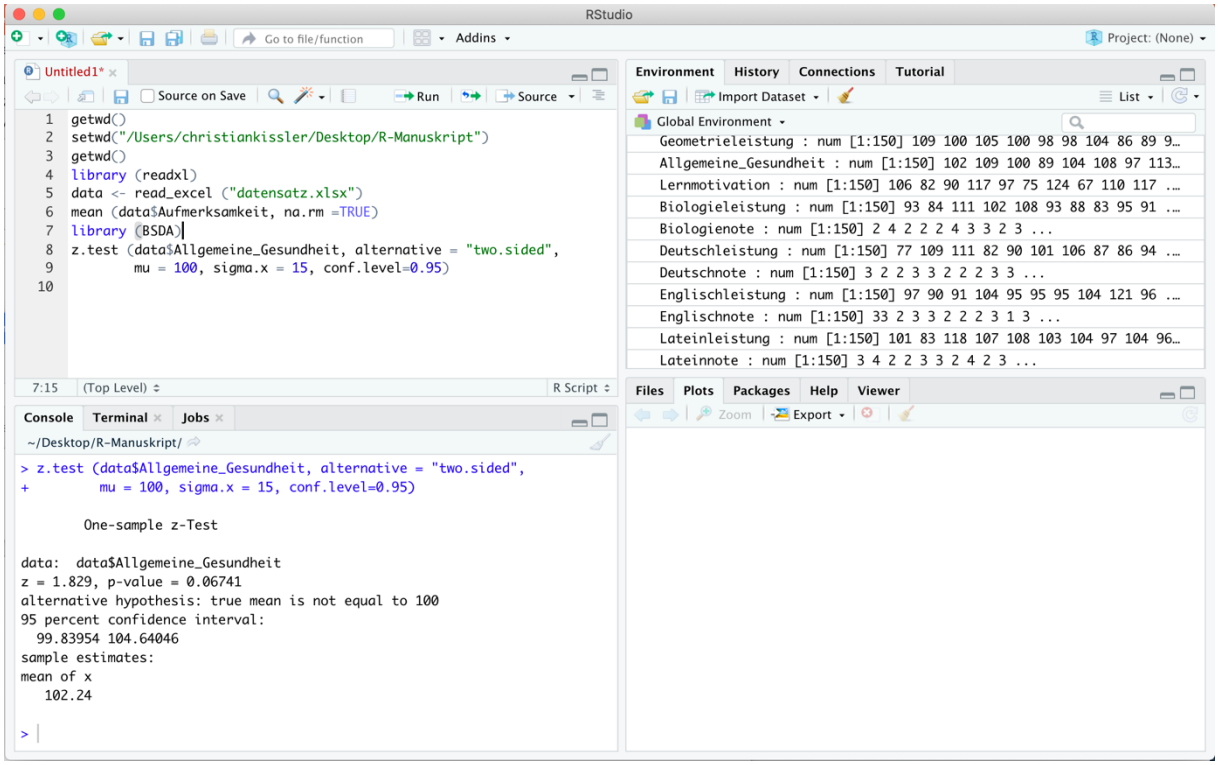

Abbildung 72: Gauß-Test - zweiseitig

Der Konsole in Abbildung 72 kann der Wert der Teststatistik (z = 1.829) sowie der p-Wert (0.06741) entnommen werden. Unter diesen Angaben ist in englischer Sprache die Alternativhypothese ausformuliert. Ganz unten steht der Mittelwert der Stichprobe für die Variable *Allgemeine_Gesundheit* (102.24) im Datensatz *data*. Für die Interpretation, ob die Nullhypothese verworfen werden muss, ist der p-Wert der statistische Kennwert, der genauer betrachtet werden muss. Wenn dieser p-Wert 0.05 unterschreitet, so ist die Nullhypothese zu verwerfen. Da der p-Wert in diesem Fall jedoch größer als 0.05 ist (der p-Wert liegt bei 0.06741), ist die Nullhypothese nicht zu verwerfen und muss beibehalten werden. Wir sehen also, dass der Mittelwert für die Variable *Allgemeine_Gesundheit* in der vorliegenden Stichprobe größer als 100 ist (der Wert liegt in der Stichprobe bei 102.24), allerdings können wir nicht mit ausreichender Gewissheit sagen, ob es sich in dieser Stichprobe um eine zufällige Abweichung von 100 handelt, oder ob diese Abweichung tatsächlich auf eine Besonderheit in der zugehörigen Grundgesamtheit zurückzuführen ist (denn der p-Wert ist größer als 0.05). Daher ist die Nullhypothese nicht zu verwerfen und wird beibehalten.

Bisher wurde zweiseitig getestet, da wir eine ungerichtete Alternativhypothese formuliert hatten. Dies bedeutet, dass die Alternativhypothese so formuliert wurde, dass

angenommen wurde, dass der Erwartungswert der Grundgesamtheit, aus der die Stichprobe stammt, größer oder kleiner als 100 ist. Es ist jedoch auch möglich, eine gerichtete Testung durchzuführen: Dann könnte bspw. angenommen werden (Alternativhypothese), dass der Erwartungswert der Grundgesamtheit, aus der die Stichprobe stammt, größer als 100 ist.

So hätten wir in diesem imaginären Beispiel vielleicht (aufgrund einer zuvor formulierten und gut begründeten Vermutung) annehmen können, dass unsere Stichprobe aus einer Grundgesamtheit stammt, welche einen höheren Erwartungswert als 100 aufweist (also tendenziell gesünder als der Durchschnitt der Gesamtbevölkerung ist). In diesem Fall hätten wir also die Richtung des erwarteten Unterschieds (größer als 100) klar vorgegeben, sodass eine gerichtete Alternativhypothese hätte formuliert werden können: *Die Stichprobe stammt aus einer Grundgesamtheit, welche mit Blick auf ihre allgemeine Gesundheit höhere Werte als die Gesamtbevölkerung aufweist.* Auch diese gerichtete Alternativhypothese kann getestet werden, was im Folgenden geschehen wird. Bei dieser gerichteten Alternativhypothese wird also davon ausgegangen, dass die Stichprobe aus einer Grundgesamtheit stammt, deren Erwartungswert **höher** als 100 ist. Die zugehörige Nullhypothese wäre dann komplementär zur Alternativhypothese zu formulieren und würde folgendermaßen lauten: *Der Erwartungswert der Grundgesamtheit, aus welcher die Stichprobe stammt, ist kleiner als 100* ***oder*** *es gibt keinen Unterschied zu 100*. Kurz könnte das Folgende geschrieben werden, um diesen Sachverhalt auszudrücken:

- Nullhypothese: H_0: $\mu < 100$ oder $\mu = 100$
- Alternativhypothese: H_1: $\mu > 100$

Um diese Nullhypothese nun zu testen (es wird – wie bereits beschrieben wurde – immer die Nullhypothese getestet), ist die folgende Syntax für die Hypothesenprüfung zu verwenden:

```
z.test (data$Allgemeine_Gesundheit, alternative = "greater", mu
     = 100, sigma.x = 15, conf.level=0.95)
```

Der Unterschied zur vorherigen zweiseitigen Testung ist, dass nun *alternative = „greater“* statt *alternative = „two.sided“* verwendet wird. Denn mit Blick auf die Alternativhypothese ist jetzt von Interesse, ob der Erwartungswert der Grundgesamtheit größer als 100 ist.

Das Ergebnis dieser Testung ist in Abbildung 73 zu sehen. Der Konsole von Abbildung 73 ist zu entnehmen, dass der *p*-Wert beim einseitigen Testen (*greater*) bei 0.0337 und somit unter 0.05 liegt. In diesem Fall ist also die Nullhypothese zu verwerfen und die Alternativhypothese kann angenommen werden. Es liegt also Evidenz für die Alternativhypothese vor.

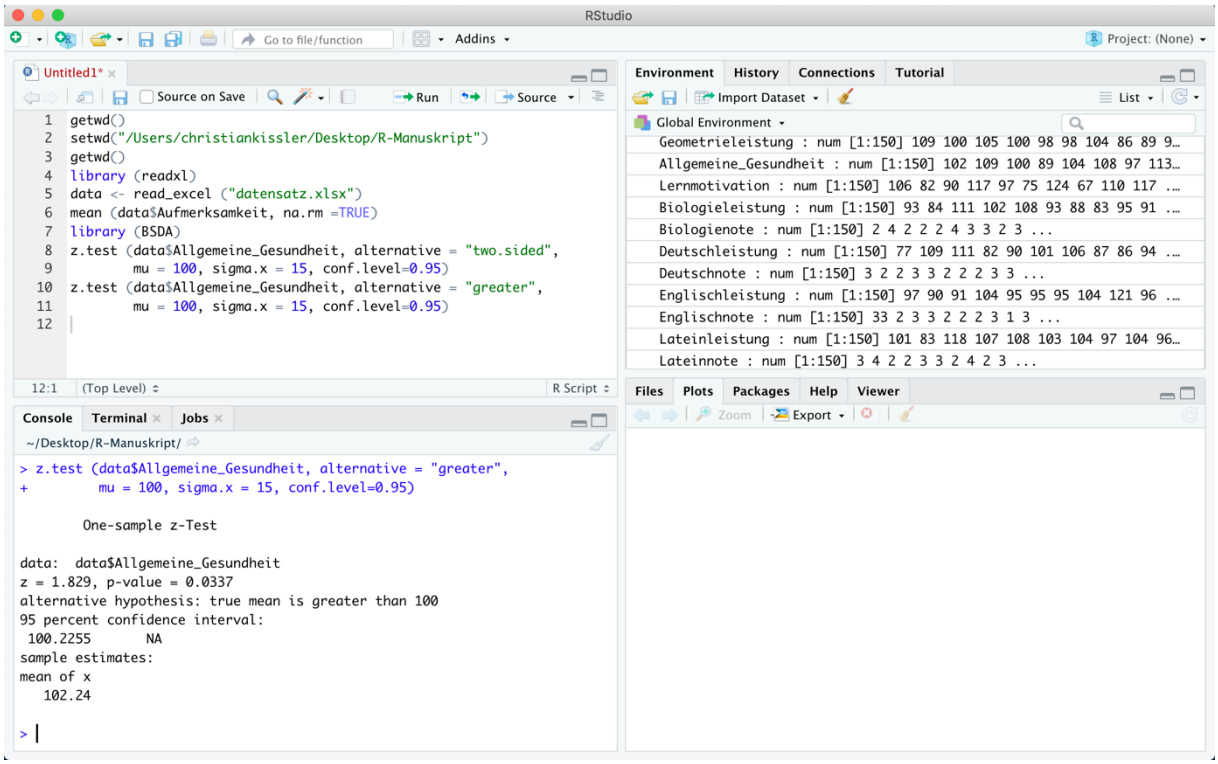

Abbildung 73: Gauß-Test - einseitig

Es konnte gezeigt werden, dass mit Blick auf dieselben Daten die Entscheidung, ob die Nullhypothese verworfen wird oder nicht, insbesondere auch von der Art der Testung abhängt (einseitig vs. zweiseitig). Dabei ist das zweiseitige Testen mit Blick auf das Verwerfen der Nullhypothese als konservativer zu bewerten. Eine einseitige Testung sollte deshalb nur dann durchgeführt werden, wenn es gute Gründe für die einseitig formulierte Alternativhypothese gibt.

Neben *greater* kann zum einseitigen Testen auch *less* genutzt werden. So könnte mit der Verwendung von *less* der folgende Sachverhalt überprüft werden:

- Nullhypothese: H_0: $\mu > 100$ oder $\mu = 100$
- Alternativhypothese: H_1: $\mu < 100$

Der Gauß-Test hat vorausgesetzt, dass ein Mittelwert, mit dem ein Vergleich durchgeführt werden soll, sowie die Standardabweichung auf Populationsebene bekannt sind. Dies ist jedoch nicht immer der Fall. Manchmal ist die Standardabweichung auf Populationsebene nicht bekannt oder man möchte sogar zwei Stichproben miteinander vergleichen – in diesen Fällen kommt der Gauß-Test an seine Grenzen. Mit *t*-Tests können diese Problemstellungen jedoch angegangen und entsprechende Forschungsfragen bearbeitet werden. Daher werden im folgenden Kapitel ebendiese *t*-Tests thematisiert und mögliche Anwendungsfälle anhand von Beispielen dargestellt.

4.3 t-Tests und Levene-Test

Manchmal interessiert die Frage, ob sich bei einer Variable die Werte einer Gruppe hinsichtlich ihrer Lage von einem bekannten Mittelwert einer anderen Grundgesamtheit unterscheiden, wobei (im Gegensatz zum Gauß-Test) die Standardabweichung unbekannt ist. Auch hier wird dann die Differenz zwischen den Werten der Stichprobe und dem bekannten Wert (z.B. dem bekannten Mittelwert einer Grundgesamtheit) auf Signifikanz geprüft. Der Einstichproben-*t*-Test kann dann eingesetzt werden, wenn der Mittelwert dieser Grundgesamtheit bekannt ist, aber die Standardabweichung unbekannt ist (was in der Regel der Fall ist). Der Zweistichproben-*t*-Test dient dazu, die Mittelwerte von zwei Gruppen zu vergleichen.

Im Folgenden wird untersucht, inwiefern Hypothesen über die Grundgesamtheit, aus der die Stichprobe stammt, hinsichtlich der Fähigkeit Zahlen zu addieren (Variable: *Additionsleistung* im Datensatz *data*) verifiziert werden können. Dabei sprechen hohe Werte bei der Variable *Additionsleistung* für eine hohe Fähigkeit in diesem Teilbereich und niedrige Werte für eine niedrige Fähigkeit in diesem Teilbereich. Um die individuelle Additionsleistung zu erheben, wurde mit jeder Person aus der Stichprobe ein entsprechender Leistungstest durchgeführt. Aus dem Manual des verwendeten Tests ist in diesem imaginären Beispiel bekannt, dass der Test normiert ist: Der Wert 100 stellt in der Gesamtbevölkerung den Mittelwert dar, die Standardabweichung hingegen ist unbekannt.

4.3.1 Der Einstichproben-t-Test

Im Folgenden soll überprüft werden, ob die vorliegende Stichprobe aus einer Grundgesamtheit stammt, welche einen anderen Erwartungswert als 100 aufweist (also besser oder schlechter addieren kann). Dies ist die Alternativhypothese, die im Folgenden getestet wird. Es handelt sich dabei um eine ungerichtete Alternativhypothese, da sie keine Richtung vorgibt: Es wird davon ausgegangen, dass die Stichprobe aus einer Grundgesamtheit stammt, deren Erwartungswert hinsichtlich der Variable *Additionsleistung* **höher oder niedriger** als 100 ist. Die zugehörige Nullhypothese ist komplementär zur Alternativhypothese formuliert und lautet: Es gibt keinen Unterschied. Kurz kann das Folgende geschrieben werden, um diesen Sachverhalt auszudrücken

- Nullhypothese: H_0: $\mu = 100$
- Alternativhypothese: H_1: $\mu \neq 100$

Die Syntax, die notwendig ist, um die Überprüfung der Nullhypothese durchzuführen, lautet:

```
t.test (data$Additionsleistung, alternative = "two.sided", mu =
     100, conf.level=0.95)
```

Die Funktion `t.test()` ist dabei notwendig, um den einfachen bzw. Einstichproben-*t*-Test durchzuführen. Die Angaben in der Klammer sind synonym zu den Angaben in der Klammer beim Gauß-Test in Kapitel 4.2 zu verstehen. Die soeben vorgestellte Syntax sowie das Ergebnis dieses *t*-Tests sind auch Abbildung 74 zu entnehmen.

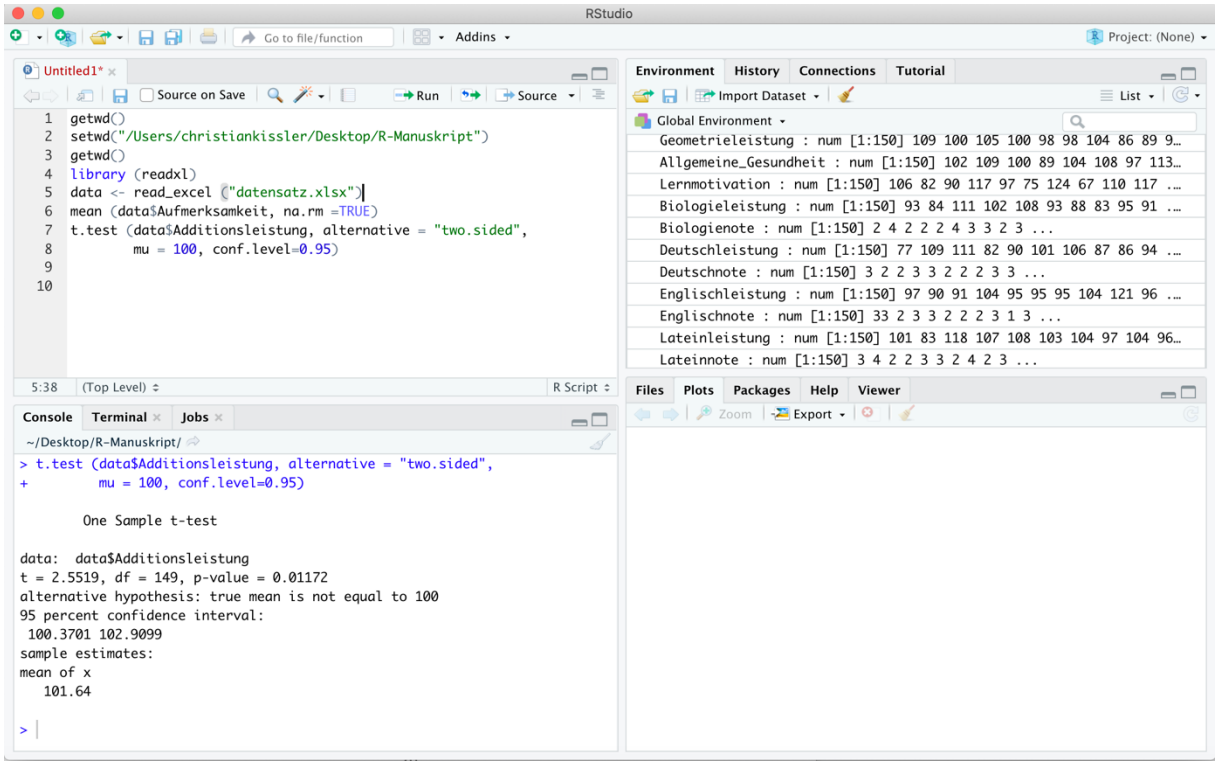

Abbildung 74: Einstichproben-t-Test

Die Ausgabe in der Konsole von Abbildung 74 ist ebenfalls analog zu den Ausführungen zum Gauß-Test zu interpretieren. Es wird deutlich, dass der *p*-Wert 0.01172 beträgt und somit kleiner als 0.05 ist: Die Nullhypothese ist also zu verwerfen und die Alternativhypothese anzunehmen. Analog zum Gauß-Test kann auch einseitig (unter Verwendung von *greater* oder *less* anstatt von *two.sided*) getestet werden.

4.3.2 Der Zweistichproben-t-Test und der Levene-Test

Neben dem Einstichproben-*t*-Test existiert auch der Zweistichproben-*t*-Test, welcher eines der geläufigsten Testverfahren in pädagogischen und psychologischen Studien darstellt. In diesen Fachrichtungen interessiert nämlich sehr häufig die Frage, ob sich zwei Gruppen (Populationen) hinsichtlich der Lage eines Parameters bzw. ihres Mittelwertes bezüglich einer Variable unterscheiden. So richten sich die Hypothesen regelmäßig auf die Unterschiedlichkeit der Erwartungswerte von zwei Gruppen. Dazu werden dann die Differenzen der

Stichprobenmittelwerte auf Signifikanz geprüft. Der Zweistichproben-*t*-Test existiert für abhängige und unabhängige Stichproben (es gibt also zwei Versionen, die penibel unterschieden werden müssen). Außerdem muss stets geprüft werden, ob Varianzhomogenität (= Varianzen der Stichproben sind gleich) oder Varianzheterogenität (= Varianzen der Stichproben sind unterschiedlich) vorliegt.

Im Folgenden soll ein Zweistichproben-*t*-Test (im Folgenden nur noch *t*-Test genannt) durchgeführt werden: Es soll überprüft werden, ob sich *Kinder mit Autismus* und *Kinder ohne Autismus* bezüglich ihrer *Aufmerksamkeitsleistung* unterscheiden (im Datensatz *data* heißt die unabhängige Variable *Autistisch* und hat die Ausprägungen *Ja* und *Nein*, sie ist also dichotom). Somit wird die gesamte Stichprobe (150 Proband:innen) in 2 Gruppen geteilt: 1) Kinder, die autistisch sind, und 2) Kinder, die nicht autistisch sind. Da zwei Gruppen verglichen werden und die Kinder stets entweder der einen oder der anderen Gruppe (aber nie beiden Gruppen gleichzeitig) angehören, handelt es sich um unabhängige Stichproben, sodass ein *t-Test für unabhängige Stichproben* zu verwenden ist. Die beiden Gruppen werden bezüglich ihrer Aufmerksamkeitsleistung (abhängige Variable: *Aufmerksamkeit*) verglichen. Bevor der *t*-Test jedoch durchgeführt werden kann, muss überprüft werden, ob die Varianzen in den beiden Gruppen gleich (= Varianzhomogenität liegt vor) oder unterschiedlich (= Varianzhomogenität liegt nicht vor) sind. Wenn Varianzhomogenität nicht vorliegt, so wird auch von Varianzheterogenität gesprochen.

Zur Überprüfung der Varianzhomogenität bzw. Varianzheterogenität ist vor der Durchführung des eigentlichen *t*-Tests ein sogenannter *Levene-Test* durchzuführen. Dazu ist das *Package lawstat* von Gastwirth et al. (2022) zu installieren sowie zu aktivieren, um dann den Befehl `levene.test()` durchführen zu können. Die Syntax, die notwendig ist, um für das oben beschriebene Beispiel die Varianzhomogenität mithilfe eines Levene-Tests zu überprüfen, lautet:

```
levene.test (data$Aufmerksamkeit, data$Autistisch)
```

Es wird also die Funktion `levene.test()` verwendet und in die Klammern wird zunächst die abhängige Variable geschrieben, die später mit dem *t*-Test überprüft werden soll (*Aufmerksamkeit*). Auf die abhängige Variable folgt hinter einem Komma die Gruppenvariable (*Autistisch*: *ja/nein*).

Das Ergebnis dieses Tests ist Abbildung 75 zu entnehmen. Im Output von Abbildung 75 ist in diesem Fall insbesondere der *p*-Wert von Relevanz. Ein *p*-Wert von über 0.10 bedeutet beim Levene-Test nämlich, dass von Varianzhomogenität auszugehen ist und ein *p*-Wert unter

0.10 bedeutet, dass von Varianzheterogenität ausgehen ist. Da der *p*-Wert bei 0.4322 und somit deutlich über 0.10 liegt, ist im Folgenden von Varianzhomogenität auszugehen.

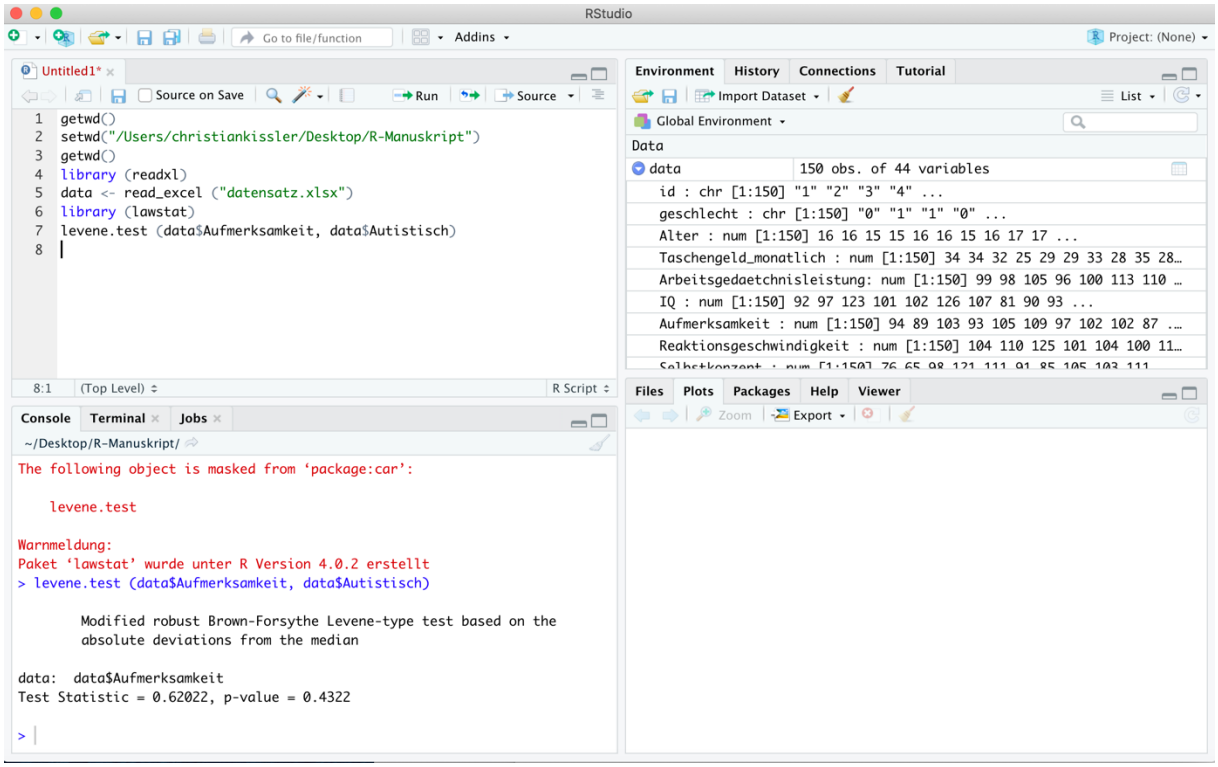

Abbildung 75: Levene-Test

Nun, da bekannt ist, dass ein *t*-Test für unabhängige Stichproben unter Voraussatzung der Varianzhomogenität zu verwenden ist, kann ebendieser *t*-Test zur Überprüfung der Nullhypothese angewendet werden. Dabei wird im Folgenden von der Alternativhypothese ausgegangen, dass sich die Gruppen (Kinder mit Autismus vs. Kinder ohne Autismus) hinsichtlich der Variable *Aufmerksamkeit* in irgendeine Richtung unterscheiden. Die entsprechenden Hypothesen lauten also:

- Nullhypothese: H_0: Aufmerksamkeit von Kindern mit Autismus = Aufmerksamkeit von Kindern ohne Autismus
- Alternativhypothese: H_1: Aufmerksamkeit von Kindern mit Autismus ≠ Aufmerksamkeit von Kindern ohne Autismus

Die Syntax, die benötigt wird, um diese Testung durchzuführen, lautet:

```
t.test(Aufmerksamkeit ~ Autistisch, data = data, paired = FALSE,
       var.equal = TRUE,  alternative = "two.sided", conf.level =
```

```
0.95)
```

Wie beim Einstichproben-*t*-Test wird also der Befehl `t.test()` verwendet, wobei sich die Angaben in den Klammern leicht unterscheiden. Im Folgenden wird nur auf die Unterschiede eingegangen. Der auffälligste Unterschied mag dieser Ausdruck sein: *Aufmerksamkeit ~ Autistisch, data = data.* Das Zeichen, das wie eine Welle (~) aussieht, wird dabei Tilde genannt. Vor der Tilde steht die abhängige Variable, hinter der Tilde steht die Gruppierungsvariable (auch unabhängige Variable genannt) und hinter diesem Ausdruck steht (mit einem Komma abgetrennt), wie der Datensatz heißt, der verwendet werden soll (in diesem Fall: *data*). Würde der Datensatz nicht *data*, sondern *DATENSATZ* heißen, so würde dort *data = DATENSATZ* stehen. Der Ausdruck *paired = FALSE* gibt an, dass es sich um einen *t*-Test für unabhängige Stichproben handelt und der Ausdruck *var.equal = TRUE* gibt an, dass von Varianzhomogenität ausgegangen wird. Die Syntax sowie das Ergebnis dieses Tests sind in Abbildung 76 dargestellt.

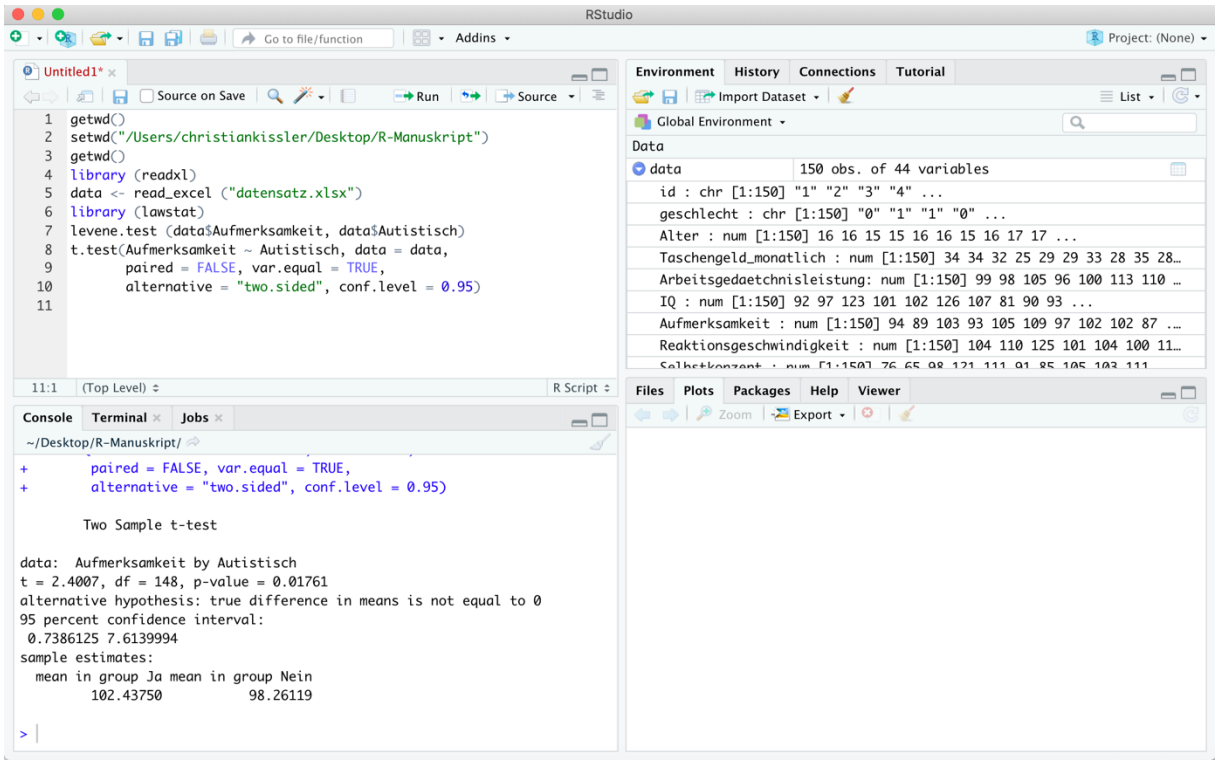

Abbildung 76: t-Test für unabhängige Stichproben

In der Konsole von Abbildung 76 sind die entsprechenden Kennwerte zu erkennen, die bei einem *t*-Test zu interpretieren sind: Der Testwert liegt bei 2.4007 und der *p*-Wert liegt bei 0.01761. Da der *p*-Wert unter 0.05 liegt, kann die Nullhypothese verworfen und die Alternativhypothese angenommen werden. Kinder mit Autismus scheinen sich also bezüglich

ihrer Aufmerksamkeitsleistung von Kindern ohne Autismus zu unterscheiden. Anhand des *p*-Wertes kann aber noch nicht abgelesen werden, ob Kinder mit Autismus im Vergleich zu Kindern ohne Autismus über eine höhere oder niedrigere Aufmerksamkeitsleistung verfügen. Dies kann aber weiter unten in der Ausgabe abgelesen werden. Dort steht, dass Kinder mit Autismus (*Ja*) einen Mittelwert von 102.43750 aufweisen und Kinder ohne Autismus (*Nein*) einen Mittelwert von 98.26119 haben. Dies würde bedeutet, dass Kinder mit Autismus höhere Werte in der Aufmerksamkeitsleistung aufweisen als Kinder ohne Autismus.

Neben *t-Tests für unabhängige Stichproben* gibt es (wie bereits angedeutet wurde) auch *t-Tests für abhängige Stichproben* (synonym: verbundene Stichproben). Diese *t*-Tests sind in der Praxis insbesondere dann von Bedeutung, wenn die gleichen Proband:innen zweimal getestet wurden und diese Testergebnisse dann miteinander verglichen werden (z.B. im Rahmen eines *Prätest- und Posttest-Designs* bzw. eines „vorher und nachher Vergleichs"). Auf diese Weise kann z.B. die Wirksamkeit eines spezifischen Trainings überprüft werden: Die Leistungsfähigkeit einer Person vor dem jeweiligen Training wird dann mit der Leistungsfähigkeit ebendieser Person nach dem jeweiligen Training verglichen.

Im Datensatz *data* gibt es die Variablen *Lesefaehigkeit_praetest* und *Lesefaehigkeit_poststest* – diese beiden Variablen werden im Folgenden von Bedeutung sein. Die inhaltliche Rahmung zu diesen beiden Variablen könnte dabei folgendermaßen aussehen: Alle 150 Kinder in der Stichprobe *data* haben eine Leseförderung erhalten. Vor dieser Leseförderung wurde ein Lesetest durchgeführt (*Lesefaehigkeit_praetest*) und nach dieser Leseförderung wurde ebendieser Lesetest wieder durchgeführt (*Lesefaehigkeit_posttest*). Dies bedeutet, dass die Gruppe der Kinder (anders als bei unabhängigen Stichproben) nicht in Subgruppen unterteilt wurde, um die Subgruppen miteinander zu vergleichen. Stattdessen liegt für jedes Kind in diesem Beispiel stets ein Wert für beide Tests bzw. beide Testzeitpunkte vor. Mithilfe dieser Werte für beide Tests soll nun überprüft werden, ob sich die Lesefähigkeit durch die Leseförderung gebessert hat. Dabei wird also quasi für jedes Kind das Testergebnis vor der Leseförderung mit dem Testergebnis nach der Leseförderung (von demselben Kind) verglichen: *Lesefaehigkeit_praetest* von Kind 1 vs. *Lesefaehigkeit_posttest* von Kind 1*; Lesefaehigkeit_praetest* von Kind 2 vs. *Lesefaehigkeit_posttest* von Kind 2*; Lesefaehigkeit_praetest* von Kind 3 vs. *Lesefaehigkeit_posttest* von Kind 3 etc.

Für diesen Vergleich ist nun mittels eines *t*-Tests für verbundene Stichproben einseitig zu testen: Denn eine Verschlechterung der Leseleistung durch das Lesetraining kann aus logischen Gründen weitgehend ausgeschlossen werden und es ist von einer Verbesserung der

Lesefähigkeit auszugehen. Die entsprechenden Hypothesen, die für die anstehende Testung von Relevanz sein werden, lauten also:

- Nullhypothese: H_0: *Lesefaehigkeit_praetest* > *Lesefaehigkeit_posttest*
  oder
  Lesefaehigkeit_praetest = *Lesefaehigkeit_posttest*
- Alternativhypothese: H_1: *Lesefaehigkeit_praetest* < *Lesefaehigkeit_posttest*

Die Syntax, um dies zu überprüfen, lautet:

```
t.test(data$Lesefaehigkeit_praetest,
      data$Lesefaehigkeit_posttest, paired = TRUE, alternative =
      "less", conf.level = 0.95)
```

In der Klammer nach dem Befehl `t.test()` stehen die beiden Variablen, für die der Mittelwertvergleich durchgeführt werden soll. Sie werden mit einem Komma voneinander getrennt. Das Argument *alternative* ist auf *less* zu setzen, weil der Mittelwert der Variable, die in der Klammer vom Befehl `t.test()` zuerst genannt wird, kleiner sein soll als der Mittelwert der Variable, die in der Klammer danach genannt wird (*Lesefaehigkeit_praetest* < *Lesefaehigkeit_posttest*). *Paired* ist in diesem Fall auf *TRUE* zu setzen, weil es sich um abhängige Stichproben handelt. Der Output dieser Berechnung ist Abbildung 77 zu entnehmen.

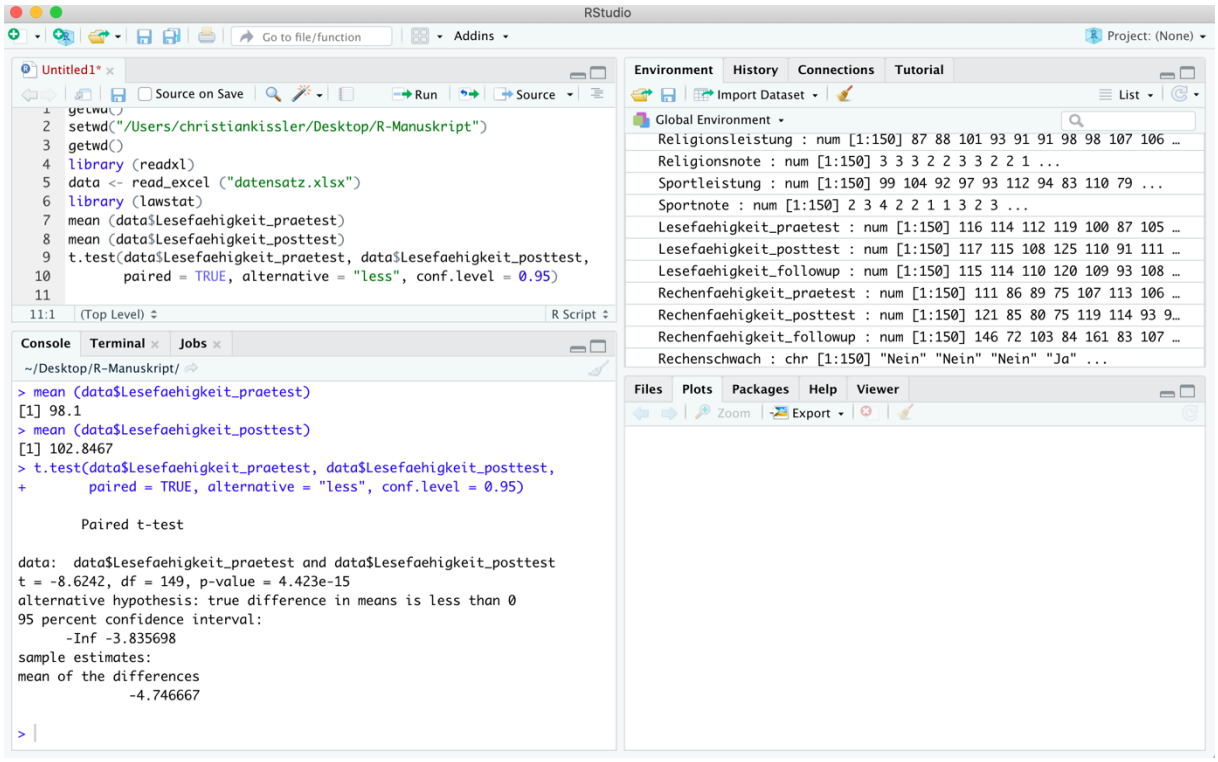

Abbildung 77: t-Test für abhängige Stichproben

In Abbildung 77 ist zu erkennen, dass der *p*-Wert deutlich kleiner als 0.05 ist und somit die Nullhypothese verworfen sowie die Alternativhypothese angenommen werden kann. Darüber hinaus ist Abbildung 77 zu entnehmen, dass neben dem *p*-Wert auch die Mittelwerte für die Variablen *Lesefaehigkeit_praetest* (M = 98.1) und *Lesefaehigkeit_posttest* (M = 102.8467) berechnet wurden, sodass die signifikante Verbesserung der Lesefähigkeit nun auch in Zahlen beschrieben werden kann. Allerdings fällt es anhand dieser Werte schwer, valide bzw. fundiert abzuschätzen, ob es sich um eine starke oder eher schwache Verbesserung der Lesefähigkeit handelt. Um die *Effektstärke* quantifizieren zu können, gibt es ebenfalls Kennwerte, die berechnet werden können. Auf diese Kennzahlen für Effektstärken wird im folgenden Kapitel eingegangen.

4.4 Effektstärken

Die Effektstärke ist eine Kennzahl, um beispielsweise quantifizieren zu können, wie effektiv ein Training war bzw. wie stark sich Werte verbessert bzw. verschlechtert haben. Ein gängiges Maß zur Quantifizierung der Effektstärke ist Cohens *d.* Zur Berechnung von Cohens *d* ist das *Package lsr* von Navarro (2022) zu installieren sowie zu aktivieren. Anschließend kann mit dem Befehl `cohensD()` die Effektstärke berechnet werden. Am Ende von Kapitel 4.3 wurde die Leseleistung von Kindern vor (*Lesefaehigkeit_praetest*) und nach (*Lesefaehigkeit_posttest*) einem Lesetraining auf Veränderung überprüft und es konnte festgestellt werden, dass sich die Leseleistung signifikant verbessert hat. Allerdings war es nicht möglich, diese Verbesserung genauer zu quantifizieren. Dies wird jedoch in diesem Kapitel erfolgen.

Cohens *d* ist ein Maß für einen (vermuteten) Effekt. Es ist üblich, Effekte in standardisierter Form (z.B. in Form von Cohens *d*) anzugeben, um Veränderungen bzw. Unterschiede miteinander vergleichen und sie so besser interpretieren zu können. Die Effektstärke Cohens *d* gibt dabei an, um wie viele Standardabweichungen sich die Erwartungswerte von zwei Populationen unterscheiden. Auf der *IQ*-Skala würde ein Unterschied von 15 *IQ*-Punkten also einen Cohens *d*-Wert von genau 1 und ein Unterschied von 30 *IQ*-Punkten einen Cohens *d*-Wert von genau 2 bedeuten. Zur Bewertung von Cohens *d* gibt es die Konvention, dass Werte von 1) über .20 als kleiner Effekt, 2) über .50 als mittlerer Effekt und 3) über .80 als großer Effekt bezeichnet werden (Döring & Bortz, 2016). Anzumerken ist an dieser Stelle, dass teilweise auch die Verwendung anderer Grenzen für die Klassifizierung von Effektstärken empfohlen wird (Lovakov & Agadullina, 2019).

Abbildung 78 zeigt in der Konsole den Output sowie im Skript-Fenster die *Syntax* zur Berechnung von Cohens *d*, um die Effektstärke für das oben genannte Beispiel zu berechnen.

Es ist zu erkennen, dass Cohens *d* bei ca. 0.38 liegt und es sich somit um einen kleinen Effekt handelt.

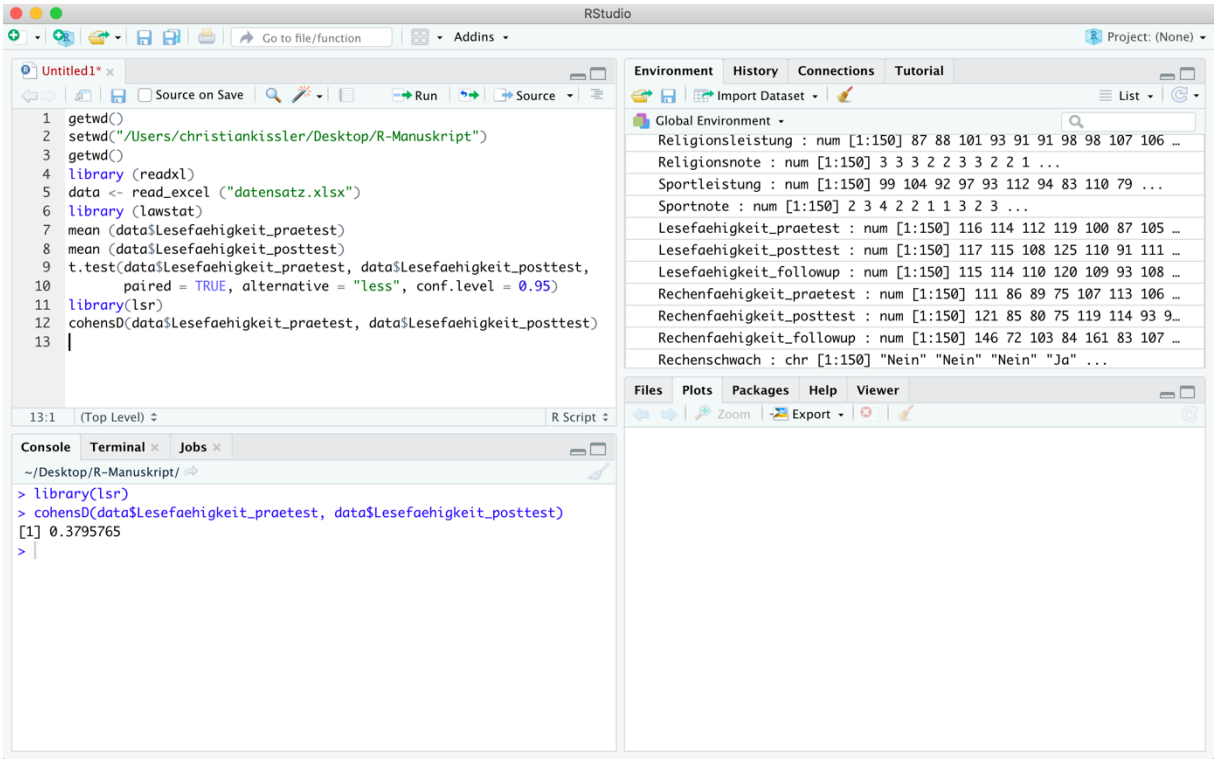

Abbildung 78: Berechnung von Cohens D

4.5 Wilcoxon-Rangsummentest

In Kapitel 4.3 wurde bereits der *t*-Test beschrieben. Dieser besitzt jedoch einige Voraussetzungen, die erfüllt sein müssen, damit er angewendet werden kann. So muss die abhängige Variable mindestens intervallskaliert sein und die Normalverteilungsannahme des interessierenden Merkmals auf Populationsebene sollte erfüllt sein (insbesondere bei kleinen Stichproben). Der *Wilcoxon-Rangsummen-Test*, der auch *Wilcoxon-Mann-Whitney-Test* oder *Mann-Whitney-U-Test* oder *U-Test* genannt wird, ist eine Alternative zum *t*-Test für unabhängige Stichproben mit ordinalen Daten und ohne Normalverteilungsannahme (z.B. wenn viele Ausreißer vorliegen). Insbesondere wenn viele Ausreißer vorliegen, kann der *U*-Test im Vergleich zum *t*-Test zu anderen (und häufig aussagekräftigeren bzw. valideren) Ergebnissen kommen.

Zur Veranschaulichung des Wilcoxon-Rangsummen-Tests soll (ähnlich wie in Kapitel 4.3) zunächst die Alternativhypothese überprüft werden, dass sich die Gruppen (Kinder mit Autismus vs. Kinder ohne Autismus) bezüglich der Variable Aufmerksamkeit in irgendeine Richtung unterscheiden. Die entsprechenden Hypothesen lauten also:

- Nullhypothese: H_0: Aufmerksamkeit von Kindern mit Autismus = Aufmerksamkeit von Kindern ohne Autismus
- Alternativhypothese: H_1: Aufmerksamkeit von Kindern mit Autismus ≠ Aufmerksamkeit von Kindern ohne Autismus

Die Syntax, die benötigt wird, um diese Testung durchzuführen, lautet:

```
wilcox.test(Aufmerksamkeit ~ Autistisch, data = data, paired =
    FALSE, alternative = "two.sided", conf.level = 0.95)
```

Die Funktion, die benötigt wird, ist also `wilcox.test()` und der Aufbau innerhalb der Klammer ist quasi identisch zum Aufbau der Klammer beim *t*-Test. Der Output dieser Berechnung ist Abbildung 79 zu entnehmen.

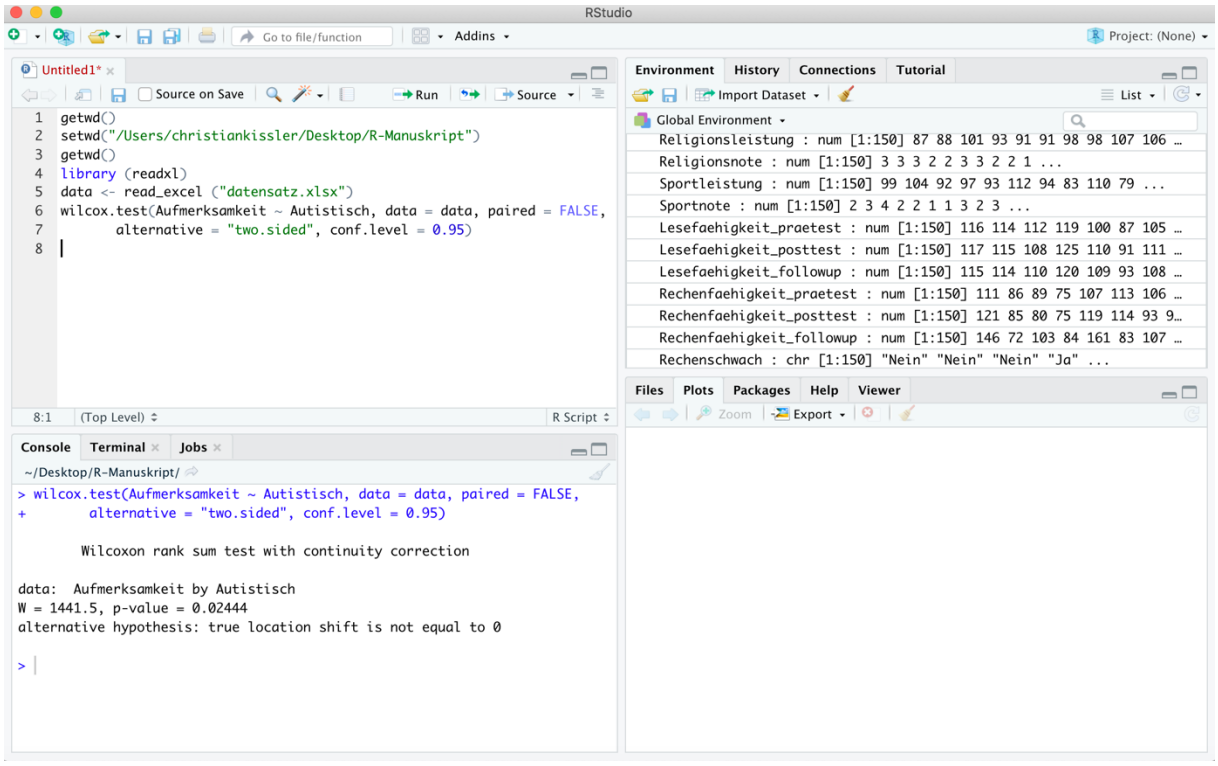

Abbildung 79: Wilcoxon-Rangsummentest

Es ist zu erkennen, dass der *p*-Wert mit 0.02444 kleiner als 0.05 ist: Daher ist die Nullhypothese zu verwerfen und die Alternativhypothese kann angenommen werden. Beim Vergleich des Outputs von Abbildung 76 mit dem Output von Abbildung 79 wird deutlich, dass sich der *p*-Wert beim Wilcoxon-Rangsummentest zwar vom *p*-Wert beim *t*-Test unterscheidet, allerdings

in diesem Fall dennoch die Interpretation der Kennzahlen zur gleichen Schlussfolgerung über die Verteilung der Daten führt (= die Nullhypothese wird verworfen und die Alternativhypothese wird angenommen). Allerdings müssen der Wilcoxon-Rangsummentest und der *t*-Test nicht immer zu Ergebnissen führen, welche die gleiche Interpretation der Daten nahelegen. Daher können Wilcoxon-Rangsummentests auch verwendet werden, um die Robustheit der Ergebnisse von *t*-Tests zu überprüfen.

Darüber hinaus können mit dem Wilcoxon-Rangsummentest Gruppenvergleiche für abhängige Variablen durchgeführt werden, für die der *t*-Test ungeeignet ist. So sind beispielsweise Noten (z. B: die Variablen *Religionsnote* oder *Physiknote* im Datensatz *data*) ordinal skaliert, sodass *t*-Tests in diesen Fällen nicht sinnvoll angewendet werden können. Stattdessen kann jedoch der Wicoxon-Rangsummen-Test verwendet werden.

4.6 ANOVAs

Im Rahmen von *t*-Tests wurden bisher stets zwei Gruppen miteinander verglichen (also unabhängige Variablen mit dichotomer Ausprägung). Zur Testung von mehr als zwei Gruppen gleichzeitig ist der *t*-Test ungeeignet. Die *ANOVA* (**AN**alysis **O**f **VA**riance; häufig auch als Varianzanalyse bezeichnet) stellt eine Verallgemeinerung des *t*-Tests dar: Das Ziel der ANOVA ist die Bestimmung des Einflusses von einer unabhängigen Variable oder mehreren unabhängigen Variablen auf eine abhängige Variable. Die abhängige Variable muss intervallskaliert bzw. stetig sein, die unabhängige Variable ist in der Regel nominalskaliert. Dabei werden die unabhängigen Variablen in der ANOVA als Faktoren und ihre Ausprägungen als Stufen bezeichnet.

Die einfachste Form der ANOVA ist die einfaktorielle ANOVA, bei der es nur einen Faktor gibt. Die Stufen des Faktors definieren dabei die Gruppen. Bei dem Faktor *Geschlecht* mit den Ausprägungen *männlich/ weiblich/ divers* hätte der Faktor *Geschlecht* also drei Stufen. Geprüft werden soll bei der einfaktoriellen ANOVA, ob auf Populationsebene Mittelwertsunterschiede zwischen den Gruppen bestehen. Im Zwei-Gruppen-Fall wird für Mittelwertvergleiche üblicherweise der *t*-Test verwendet, wobei eine ANOVA in diesem Zwei-Gruppen-Fall bei homogenen Varianzen zu denselben Ergebnissen führen würde. Um zu prüfen, ob der Faktor einen Einfluss auf die Erwartungswerte der abhängigen Variable hat, wird die Nullhypothese, dass keine Unterschiede bestehen, gegen die Alternativhypothese, dass Unterschiede bestehen, getestet.

Voraussetzung für die Durchführung der einfaktoriellen ANOVA ist, dass die Normalverteilungsannahme erfüllt ist – bei großen Stichproben können Verstöße gegen die

Normalverteilungsannehme jedoch in aller Regel vernachlässigt werden. Die Prüfung der Normalverteilungsannahme kann mithilfe von Box-Plots und Histogrammen für jede Gruppe sowie mit Shapiro-Wilks-Tests (Test auf Normalverteilung) realisiert werden.

Außerdem sollte die Homoskedastizitätsannahme überprüft werden: Als pragmatische Faustregel wird regelmäßig angenommen, dass die maximale Standardabweichung je Gruppe höchstens doppelt bis dreimal so groß wie die minimale Standardabweichung, die eine Gruppe bezüglich der abhängigen Variable aufweist, sein sollte; alternativ oder ergänzend kann auch der Levene-Test durchgeführt werden.

Im Folgenden soll eine ANOVA in Bezug auf die abhängige Variable *Aufmerksamkeit* durchgeführt werden. Der Datensatz *data* weist eine Gruppierungsvariable (*gruppe*) auf, welche die Gesamtstichprobe in 3 Teilgruppen aufteilt (wobei nicht genauer bekannt ist, auf welcher Grundlage diese Gruppeneinteilung erfolgt). Diese 3 Gruppen sollen nun in Bezug auf ihre Aufmerksamkeit miteinander verglichen werden. Dies kann mit der folgenden Syntax umgesetzt werden:

```
anova_01 <- aov(Aufmerksamkeit ~ gruppe, data = data)
summary (anova_01)
```

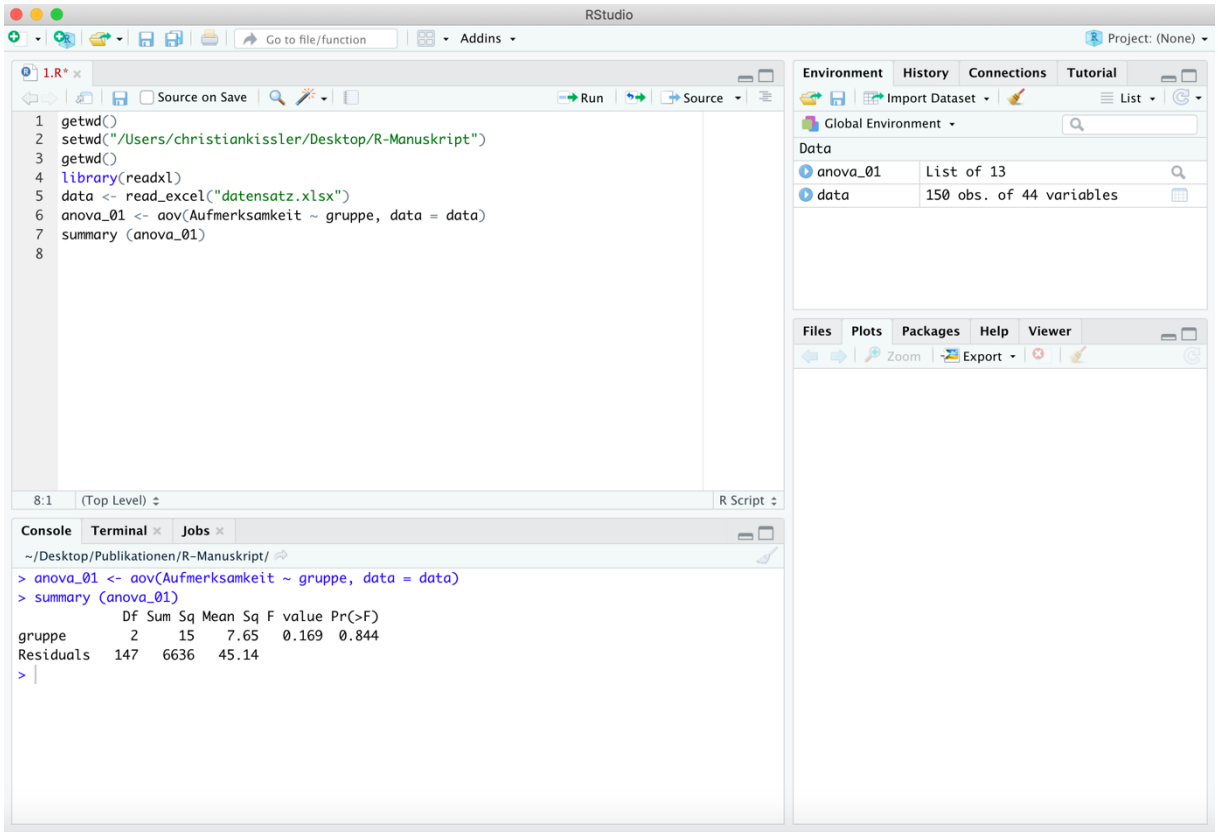

Abbildung 80: Einfaktorielle ANOVA

Der Befahl `aov()` dient dabei der Berechnung der ANOVA. In der Klammer wird zunächst die abhängige Variable (*Aufmerksamkeit*) und dann der Faktor (*gruppe*) aufgeführt, wobei zwischen die beiden Variablen die Tilde geschrieben wird. Hinter diesem Ausdruck wird der Datensatz (*data*) benannt, in welchem sich die zuvor genannten Variablen befinden. Das Ergebnis wird als *anova_01* abgespeichert und anschließend in einem nächsten Schritt mit dem Befehl `summary()` abgerufen. Der Output sowie die Syntax selbst sind Abbildung 80 zu entnehmen.

Dem Output, der in der Konsole von Abbildung 80 dargestellt ist, ist zu entnehmen, dass der Testwert (*F*) bei 0.169 liegt und der *p*-Wert 0.844 beträgt. Somit besteht zwischen den drei Gruppen in Bezug auf die Aufmerksamkeitsleistung kein signifikanter Unterschied und die Nullhypothese muss beibehalten werden: Denn der *p*-Wert liegt deutlich über 0.05.

Anders fällt das Ergebnis aus, wenn eine ANOVA zwar für die unabhängige Variable *Gruppe*, jedoch für die abhängige Variable *Lesefaehigkeit_praetest* durchgeführt wird. Die Berechnung dieser ANOVA führt zu einem signifikanten Ergebnis, da der *p*-Wert unter 0.05 liegt. In diesem Fall ist die Nullhypothese zu verwerfen und die Alternativhypothese anzunehmen: Die Lesefähigkeit unterscheidet sich zwischen den Gruppen.

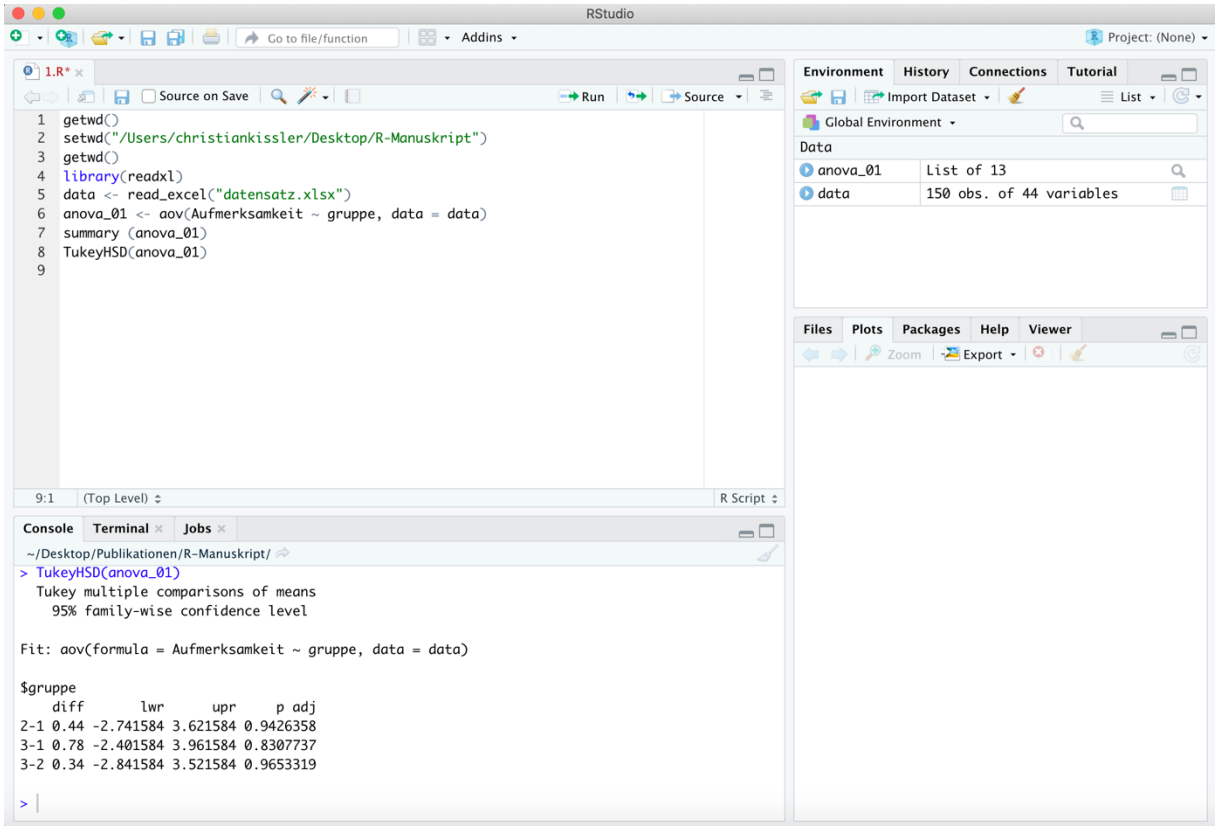

Abbildung 81: Post-Hoc-Tests nach Tukey

Die ANOVA selbst gibt jedoch keine Auskunft darüber, zwischen welchen Gruppen der Unterschied vorliegt oder ob sich sogar alle drei Gruppen signifikant voneinander unterscheiden. Eine solche Überprüfung ist mit *Post-Hoc-Tests nach Tukey* möglich:

```
TukeyHSD (anova_01)
```

Die Syntax sowie der Output sind der Abbildung 81 zu entnehmen. Die Funktion `TukeyHSD()` wurde also auf das zuvor erstellte und im *Environment* gespeicherte Objekt *anova_01* angewendet. In dem Objekt *anova_01* sind die Ergebnisse der ANOVA für die abhängige Variable *Aufmerksamkeit* abgespeichert. Es wird deutlich, dass der Vergleich von Gruppe 2 und 1 zu einem *p*-Wert über 0.05 führt (0.94). Auch der *p*-Wert für den Vergleich von Gruppe 3 mit Gruppe 1 liegt über 0.05 (0.83). Der *p*-Wert für den Vergleich von Gruppe 3 und 2 übersteigt ebenfalls deutlich das Signifikanzniveau von 0.05 (0.97). Daher kann in diesem Beispiel bezüglich der Aufmerksamkeitsleistung kein Unterschied zwischen den einzelnen Gruppen festgestellt werden.

Wird ein solcher Post-Hoc-Test nach Tukey für die ANOVA mit der unabhängigen Variable *gruppe* und der abhängigen Variable *Lesefaehigkeit_praetest* durchgeführt, so kommt es hingegen zu den folgenden Resultaten: Es wird deutlich, dass der Vergleich von Gruppe 2 und 1 zu einem *p*-Wert über 0.05 führt (0.056) und auch der *p*-Wert für den Vergleich zwischen Gruppe 3 und 1 über 0.05 liegt (0.502). Der *p*-Wert für den Vergleich von Gruppe 3 mit Gruppe 2 liegt aber deutlich unter 0.05 (0.002), sodass hier von einem signifikanten Unterschied zwischen Gruppe 2 und 3 in Bezug auf die Variable *Lesefaehigkeit_praetest* auszugehen ist.

4.7 Chi-Quadrat und Kramers V

Auch nominalskalierte Variablen können auf Zusammenhänge überprüft werden. So könnte beispielsweise mit Blick auf die Daten des vorliegenden Datensatzes *data* die Frage gestellt werden, ob eine Autismusdiagnose (*Autismus: Ja/ Nein*) mit dem Geschlecht (*männlich/ weiblich*) in einem Zusammenhang steht.

Die Grundidee zur Berechnung dieses Zusammenhangs ist der Vergleich einer beobachteten Kontingenztabelle mit einer fiktiven Kontingenztabelle, in der keine Beziehung zwischen den Variablen besteht bzw. in der die betrachteten Variablen also voneinander unabhängig sind (Indifferenztabelle). Je größer die Abweichung der beiden Tabellen voneinander ist, desto stärker ist der Zusammenhang. Vertiefende theoretische Ausführungen zu diesem Thema können insbesondere der Literatur von Holling & Gediga (2015) entnommen werden. Zur Beschreibung der Größe des Zusammenhangs wird bei nominalskalierten

Variablen regelmäßig die χ^2-Statistik verwendet. Der χ^2-Unabhängigkeitstest berechnet dabei den Zusammenhang zwischen zwei kategorialen (nominalskalierten) Variablen. Die entsprechenden Hypothesen lauten dabei:

- Nullhypothese (H_0): *X* und *Y* sind stochastisch unabhängig.
- Alternativhypothese (H_1): *X* und *Y* sind nicht stochastisch unabhängig.

Für das oben genannte Beispiel lautet die Syntax zur Berechnung des χ^2-Unabhängigkeitstests:

```
chisq.test(data$geschlecht, data$Autistisch)
```

Der Output ist Abbildung 82 zu entnehmen. Es ist zu erkennen, dass der *p*-Wert (0.7058) deutlich größer als 0.05 ist und somit muss die Nullhypothese beibehalten werden.

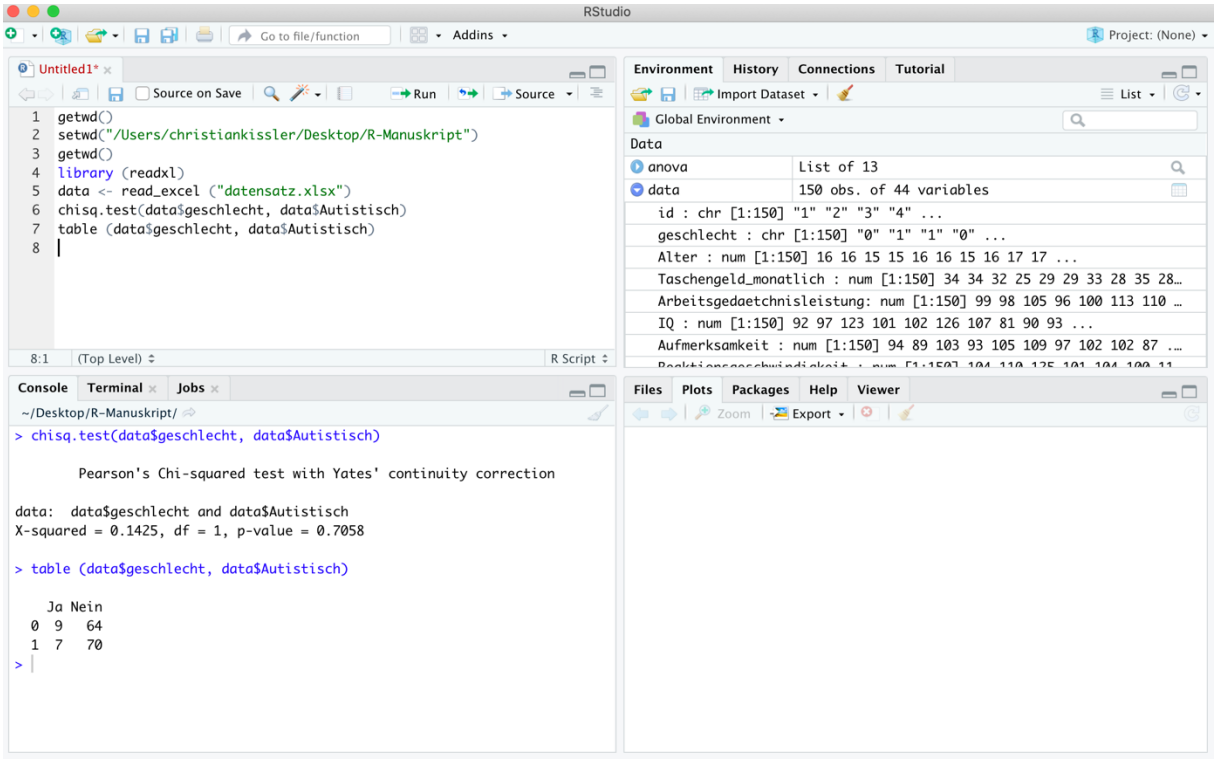

Abbildung 82: χ^2-Unabhängigkeitstest

Abbildung 82 ist neben dem Output des χ^2-Unabhängigkeitstests auch eine Tabelle zu entnehmen, in welcher das Geschlecht (*0* = männlich; *1* = weiblich) und das Vorliegen einer *Autismusdiagnose* (autistisch: *Ja/Nein*) dargestellt wird. Mit Blick auf die Tabelle wird deutlich, dass sich in dieser Stichprobe das Vorliegen bzw. Nicht-Vorliegen von Autismusdiagnosen relativ gleichmäßig über beide Geschlechter zu verteilen scheint. So kann mit der `table()`-Funktion das Ergebnis des χ^2-Unabhängigkeitstests (Ergebnis: Es liegt kein Zusammenhang zwischen *Autismusdiagnose* und *Geschlecht* vor) visualisiert werden.

Eine Effektstärke, um den Zusammenhang zwischen zwei nominalskalierten Variablen zu quantifizieren, ist Cramers *V*. Cramers *V* stellt dabei ein normiertes und auf der χ^2-Statistik basierendes Zusammenhangsmaß für nominalskalierte Variablen dar. Cramers *V* kann Werte im Intervall von 0 bis 1 annehmen: Im Falle einer vollständigen Unabhängigkeit nimmt Cramers *V* immer den Wert 0 an und bei einer perfekten Abhängigkeit nimmt Cramers *V* den Wert 1 an (Holling & Gediga, 2011). Die Interpretation von Cramers *V* hängt dabei insbesondere davon ab, wie viele Zeilen und Spalten die Tabelle hätte, wenn man sie zeichnen würde (im obigen Beispiel hatte die Tabelle 2 Zeilen und Spalten). Bei einem Minimum von 2 Zeilen und Spalten spricht man bei einem Wert von größer als 1) 0.10 von einem kleinen Effekt, 2) 0.30 von einem mittleren Effekt, 3) 0.50 von einem großen Effekt; bei einem Minimum von 3 Zeilen und Spalten spricht man bei einem Wert von größer als 1) 0.07 von einem kleinen Effekt, 2) 0.21 von einem mittleren Effekt, 3) 0.35 von einem großen Effekt; bei einem Minimum von 4 Zeilen und Spalten spricht man bei einem Wert von größer als 1) 0.06 von einem kleinen Effekt, 2) 0.17 von einem mittleren Effekt, 3) 0.29 von einem großen Effekt (Cohen, 1988, p. 222).

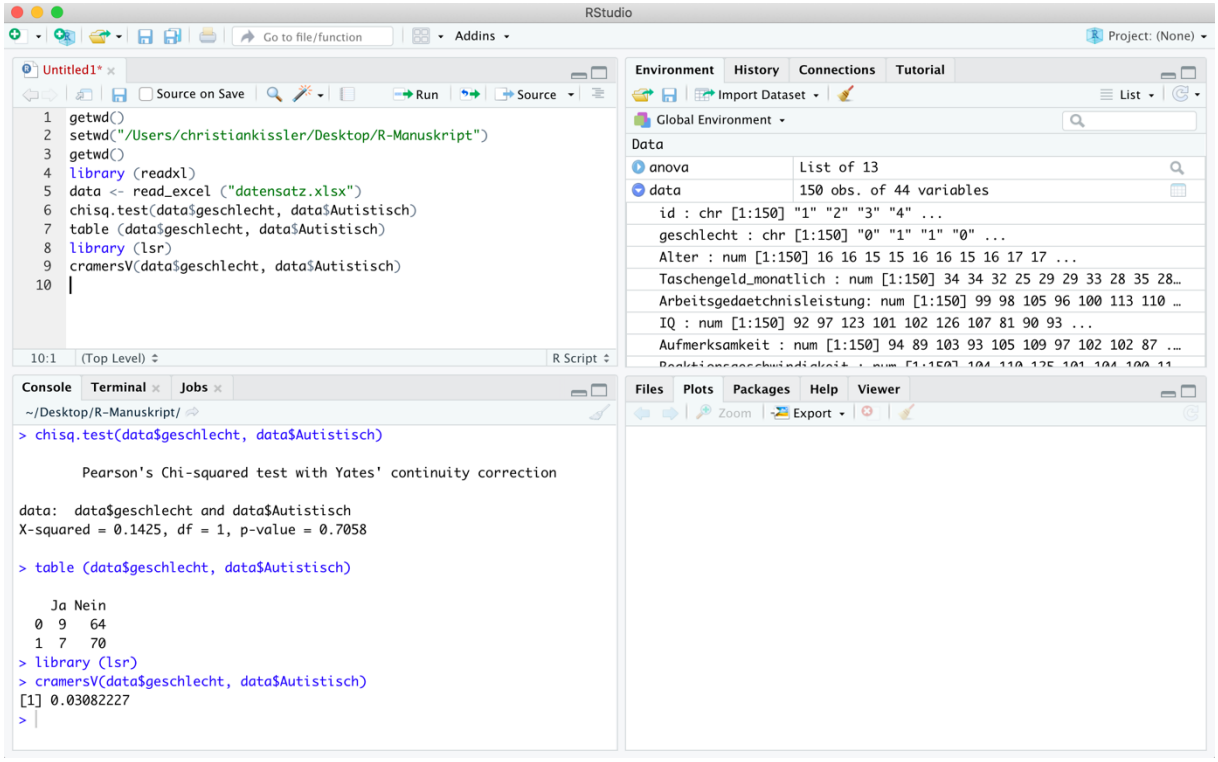

Abbildung 83: Cramers V

Um Cramers *V* in `RStudio` zu berechnen, ist zunächst ggf. das *Package lsr* von Navarro (2022) zu installieren und zu aktivieren. Der *Code*

```
cramersV(data$geschlecht, data$Autistisch)
```

dient der Berechnung von Cramers *V* und ist zusammen mit dem Output Abbildung 83 zu entnehmen. In Abbildung 83 ist zu sehen, dass Cramers *V* gerundet 0.031 beträgt und somit deutlich unter dem Grenzwert für einen kleinen Effekt liegt. Dies ist an dieser Stelle nicht verwunderlich, da bereits zuvor festgestellt werden konnte, dass kein bedeutender Zusammenhang besteht.

4.8 Korrelationen und Regressionen

In Kapitel 3.7 wurde bereits der Korrelationskoeffizient vorgestellt, um zu quantifizieren, wie stark der Zusammenhang zwischen zwei Variablen ist. An dieser Stelle ist zu ergänzen, dass auch Korrelationen signifikant bzw. nicht-signifikant/ bedeutend bzw. unbedeutend sein können. In Kapitel 3.7 wurde der Korrelationskoeffizient zwischen dem *IQ* und der Mathematikleistung bzw. der *Matheleistung* berechnet und als schwacher Zusammenhang klassifiziert. Nun soll mithilfe eines *p*-Wertes bestimmt werden, ob dieser Zusammenhang bedeutungsvoll bzw. signifikant ist. Dazu ist das *Package Hmisc* von Harrell Jr. und Dupont (2022) zu nutzen. In Kapitel 3.7 wurde der Korrelationskoeffizient zwischen *Matheleistung* und *IQ* mit dem folgenden Befehl bestimmt:

```
cor(data$Matheleistung, data$IQ, use="pairwise.complete.obs",
    method = "pearson")
```

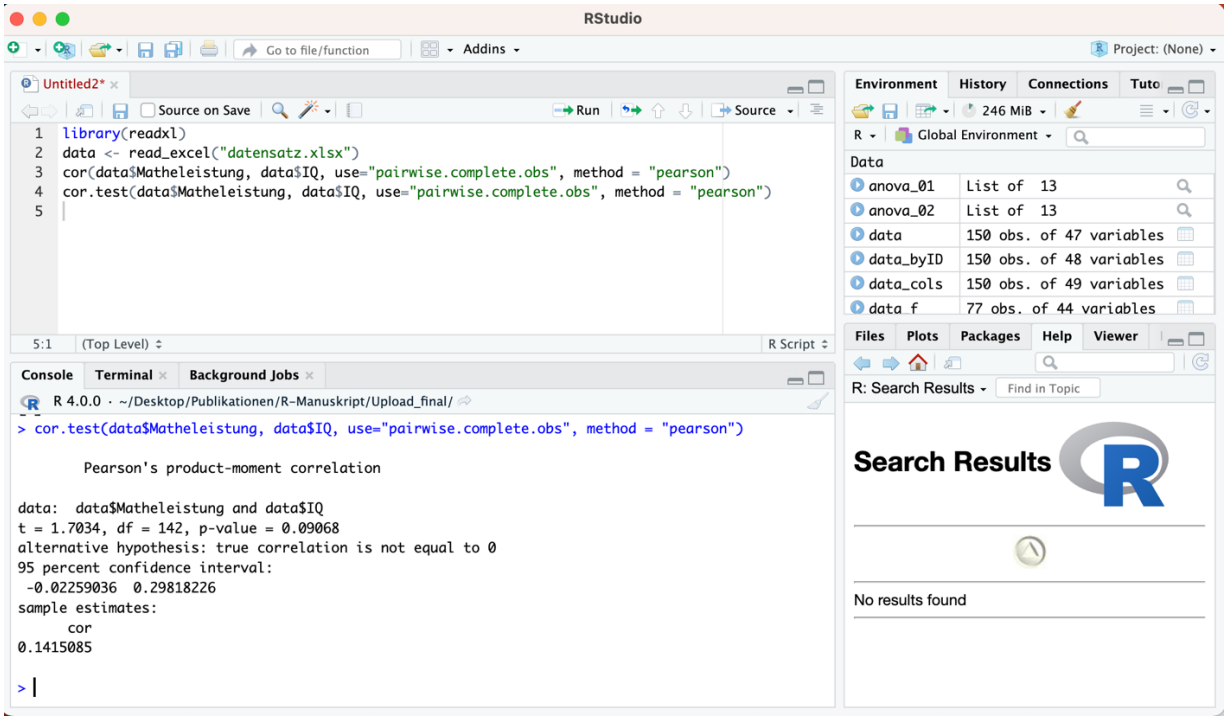

Abbildung 84: Testung auf Signifikanz einer Korrelation

Wenn dieser Befehl ausgeführt wurde, so wurde in der Konsole ausgegeben, dass der Korrelationskoeffizient bei 0.1415085 lag. Im Folgenden soll dieser Zusammenhang bzw. diese Korrelation weiter untersucht werden: So kann nämlich auch überprüft werden, ob dieser Zusammenhang signifikant (also überzufällig) ist. Dies ist mit dem folgenden Befehl möglich:

```
cor.test(data$Matheleistung, data$IQ,
         use="pairwise.complete.obs", method = "pearson")
```

Abbildung 84 ist zu entnehmen, dass es sich bei der Korrelation von *Matheleistung* und *IQ* nicht um einen signifikanten Zusammenhang handelt, da der *p*-Wert bei 0.09068 und somit deutlich über dem Signifikanzniveau von 0.05 liegt. Weiter unten in der Konsole ist zu erkennen, dass die Funktion `cor.test()` auch den Korrelationskoeffizienten (0.1415085) ausgibt.

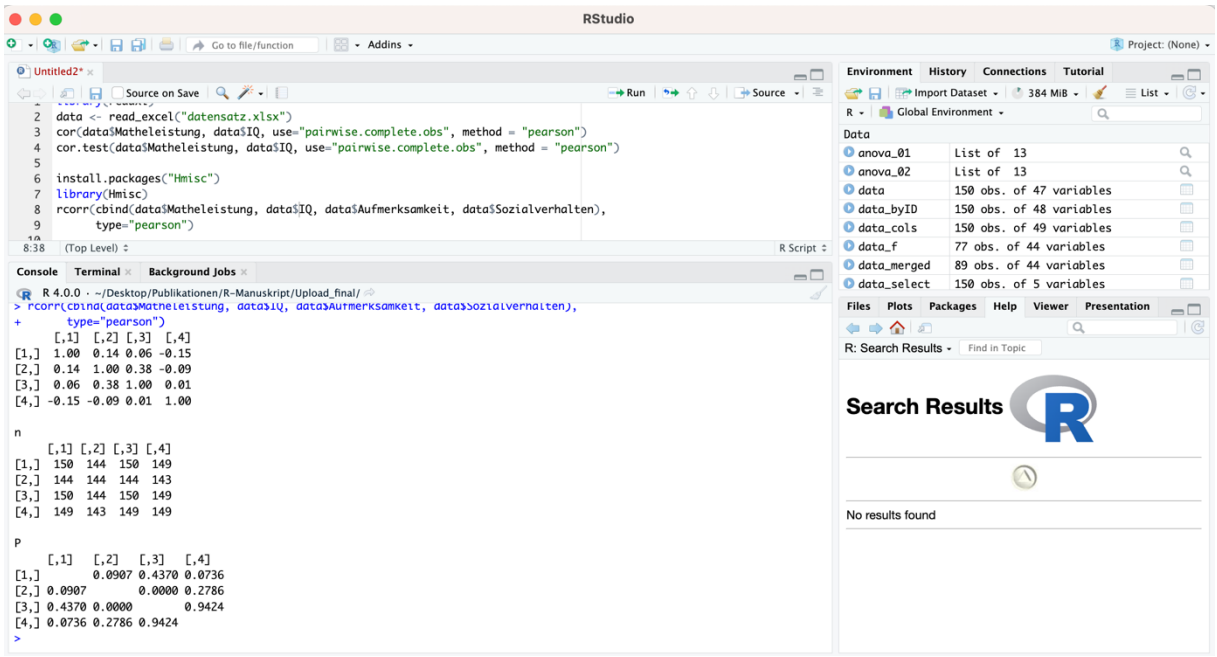

Abbildung 85: Verwendung der rcorr()-Funktion

Manchmal ist es von Interesse, die Korrelation von mehr als nur zwei Variablen zu untersuchen. In diesem Fall müssen die entsprechenden Kennwerte nicht zwingend für jede Korrelation einzeln berechnet werden. Es ist nämlich möglich, diese Berechnungen gebündelt mit der Funktion `rcorr()` aus dem *Package Hmisc* von Harrell Jr. und Dupont (2022) vorzunehmen. Mit `rcorr()` könnten dann beispielsweise Korrelationen der Variablen *Matheleistung*, *IQ*, *Aufmerksamkeit* und *Sozialverhalten* aus dem Datensatz *data* folgendermaßen berechnet sowie auf Signifikanz überprüft werden:

```
rcorr(cbind(data$Matheleistung, data$IQ, data$Aufmerksamkeit,
      data$Sozialverhalten), type="pearson")
```

In den Klammern der `rcorr()`-Funktion werden mithilfe der `cbind()`-Funktion zunächst die Variablen benannt, für welche die Berechnung durchgeführt werden soll. Anschließend wird angegeben, welche Art von Korrelation (in diesem Fall: *Pearson-Korrelation*) berechnet werden soll. Wenn diese Funktion ausgeführt wird, so wird in der Konsole ein umfangreicher Output ausgegeben (Abbildung 85).

Der Output von Abbildung 85 besteht aus 3 Tabellen: Die erste Tabelle hat keine Überschrift, die zweite Tabelle ist mit *n* überschrieben und die dritte Tabelle ist mit *p* überschrieben. In der ersten Tabelle werden die Korrelationskoeffizienten dargestellt; der zweiten Tabelle (*n*) ist zu entnehmen, von wie vielen Personen die Datensätze zur Berechnung der jeweiligen Korrelation berücksichtigt wurden; in der dritten Tabelle (*p*) sind die berechneten *p*-Werte zu erkennen, um zu entscheiden, ob ein signifikanter Zusammenhang vorliegt (oder nicht).

Auffällig ist, dass die Tabellen in Zeilen und Spalten nur mit Zahlen überschrieben sind. Um dies zu verstehen, wird die Syntax benötigt, denn hinter den Zahlen verbergen sich zugehörige Variablen. In dieser Syntax – wie sie weiter oben angegeben wurde - wurden die Variablen angegeben, welche zur Berechnung der Korrelation berücksichtigt werden sollen. Die zuerst genannte Variable entspricht in den Tabellen der 1 (*Matheleistung*), die an zweiter Stelle genannte Variable der 2 (*IQ*), die an dritter Stelle genannte Variable der 3 (*Aufmerksamkeit*) und die an vierter Stelle genannte Variable der 4 (*Sozialverhalten*). Wenn nun in der obersten Tabelle in Abbildung 85 in die zweite Zeile und die erste Spalte geschaut wird, so steht dort: 0.14. Dies ist dann entsprechend der Korrelationskoeffizient für die Korrelation von *IQ* (2) und *Matheleistung* (1). Der Wert in der vierten Spalte und zweiten Zeile in der zweiten Tabelle in Abbildung 85 beträgt 143. Dies bedeutet, dass bei der Korrelation von *Sozialverhalten* (4) und *IQ* (2) die Daten von 143 Personen berücksichtigt werden konnten. Analog dazu sind die dritte Tabelle sowie die übrigen Werte in den Tabellen zu interpretieren.

Ein anderes Verfahren, um die Beziehung zwischen Variablen zu beschreiben, ist die lineare Regression. Bei der (einfachen) linearen Regression wird die Beziehung zwischen einer unabhängigen Variable (UV) und einer abhängigen Variable (AV) durch eine Regressionsgerade (lineare Funktion) beschrieben. Die Regression kann zur Vorhersage von Werten einer Variable durch eine andere Variable verwendet werden. Die AV wird dann auch als Kriterium bzw. Antwortvariable und die UV als Prädiktor oder erklärende Variable bezeichnet. Wenn *Y* die AV und *X* die UV ist, so spricht man von einer *Regression von Y auf*

X. Nun wird mit der folgenden Syntax im Datensatz *data* eine Regression von *Matheleistung* (AV bzw. Y) auf *Intelligenz* (UV bzw. X) berechnet:

```
LM_1 <- lm (Matheleistung ~ IQ, data = data)
summary (LM_1)
```

Diese Abfolge von Befehlen besteht aus zwei Zeilen. In der ersten Zeile wird rechts vom Zuweisungspfeil die lineare Regression mit der Funktion `lm()` bestimmt. Das Ergebnis der Berechnung, die in der Form ausgeführt wird, wie sie rechts vom Zuweisungspfeil beschrieben wird, wird im Environment unter *LM_1*, so wie es links vom Zuweisungspfeil angegeben ist, abgespeichert. Innerhalb der Klammern der `lm()`-Funktion steht zunächst (links von der Tilde) die abhängige Variable (*Matheleistung*) und danach wird (rechts von der Tilde) die unabhängige Variable (*IQ*) benannt. Es folgt ein Komma und hinter diesem Komma wird der Datensatz benannt, der in diesem Beispiel *data* heißt (würde der verwendete Datensatz beispielsweise DATENSATZ heißen, so würde dort anstelle von *data* = *data* das Folgende stehen: *data* = *DATENSATZ*). Abbildung 86 ist der Output zu entnehmen, welcher in der Konsole erscheint, wenn anschließend die Zeile `summary(LM_1)` ausgeführt wird.

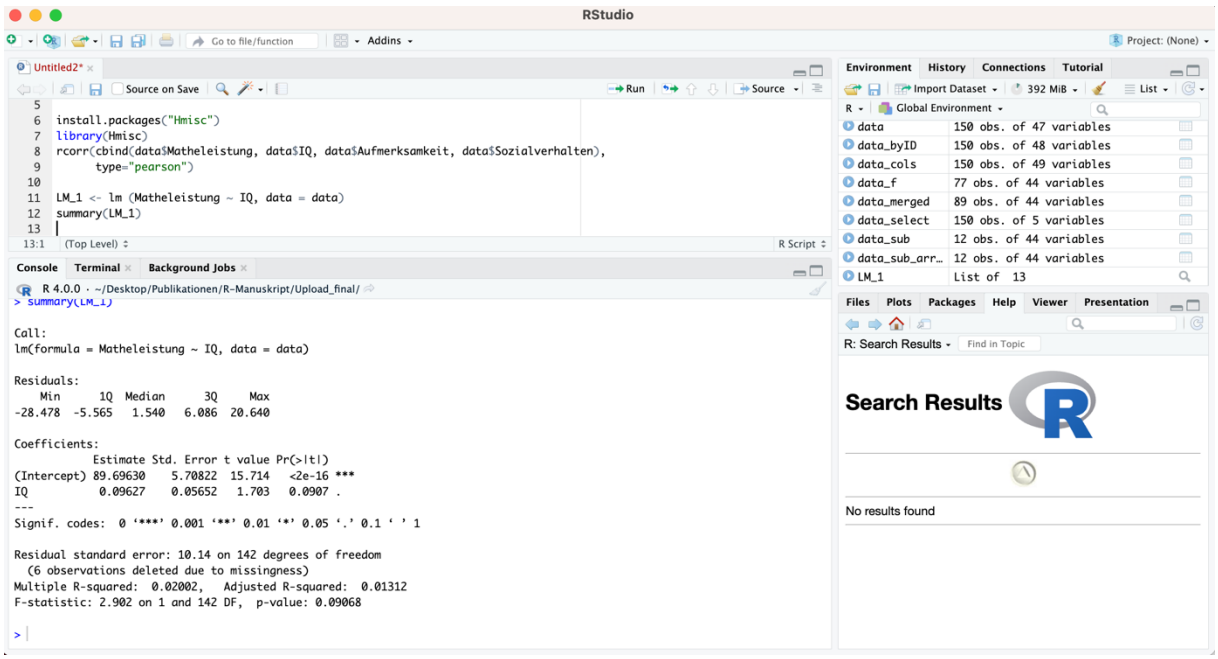

Abbildung 86: Output für Lineare Regressionen

Der Output in Abbildung 86 ist folgendermaßen zu interpretieren: Ganz oben unter „Call“ wird das Lineare Modell, für welches die Berechnung durchgeführt wurde, noch einmal benannt. Unter „*Coefficients*“ sind die geschätzten Koeffizienten angegeben. Von den Angaben unter „*Coefficients*“ ist insbesondere die Angabe, die hinter *IQ* unter dem Punkt „*Estimate*“ steht, von

Relevanz, weil dieser Wert die Steigung der Geraden im Linearen Modell widerspiegelt (in diesem Fall liegt der Wert bei 0.09627). Dieser Wert bedeutet in diesem Kontext, dass entsprechend des Linearen Modells die Steigerung von einem *IQ*-Punkt eine Steigerung der *Matheleistung* von 0.09627 Punkten bedeutet. Über diesem Wert (in der Zeile („*Intercepts*“)) wird die Schnittstelle der Geraden mit der Y-Achse genannt (also der Wert, der in diesem Fall entsprechend des linearen Modells für die Variable *Matheleistung* angenommen werden würde, wenn der *IQ*-Wert bei 0 liegen würde). Aus den Werten, die in Abbildung 86 dargestellt werden, ergibt sich somit, dass die Schnittstelle mit der Y-Ache im Koordinatensystem der Punkt (X = 0 / Y = 89.69630) wäre. Hinter „*Signif. Codes*“ ist erläuternd aufgeführt, was die Sternchen über dieser Zeile bedeuten. Es wird deutlich, dass für die Variable *IQ* das Signifikanzniveau verfehlt wird, da der errechnete *p*-Wert bei 0.0907 und somit über 0.05 liegt. Hinter „*Residual standard error*“ ist der *Standardschätzfehler* (auch bekannt als *SEE*) angegeben und unter dieser Angabe wird hinter „*Multiple R-Squared*“ der Determinationskoeffizient R^2 angegeben (in diesem Fall: 0.02002).

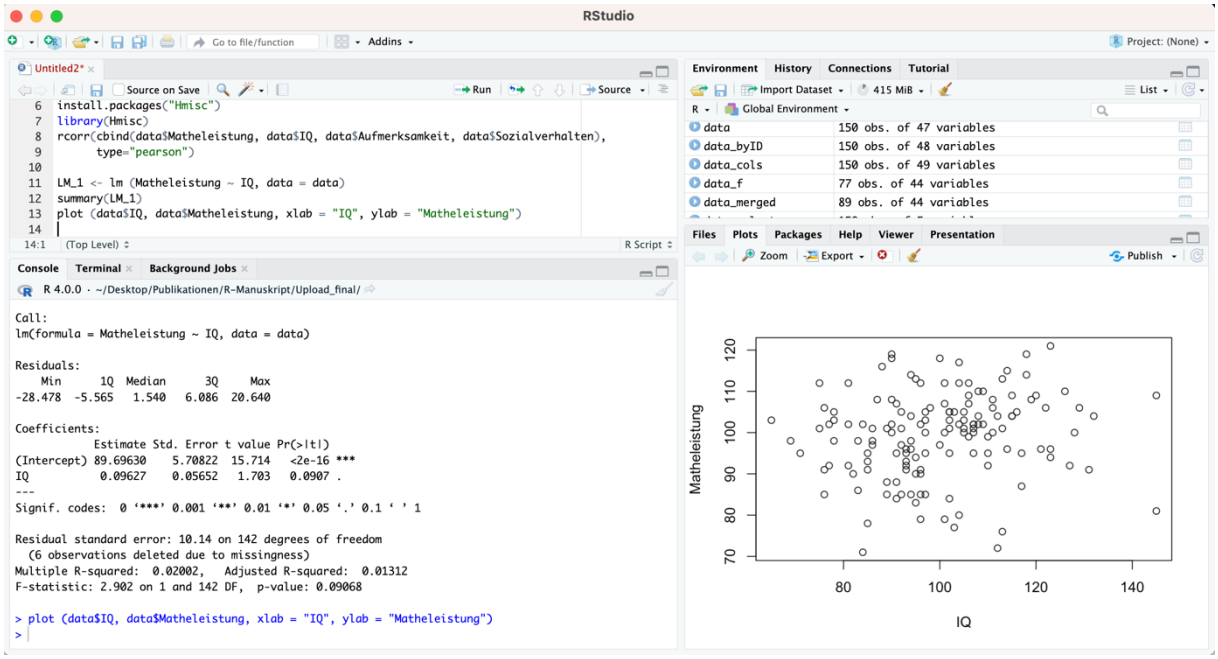

Abbildung 87: Scatterplot

Sie können sich das berechnete lineare Modell auch von `RStudio` in den Scatterplot, welcher bereits in Kapitel 3.7 besprochen wurde, einzeichnen lassen. Dazu müssen Sie zunächst den Scatterplot mit dem folgenden Befehl erzeugen:

```
plot (data$IQ, data$Matheleistung, xlab = "IQ", ylab =
     "Matheleistung")
```

Der entsprechende Scatterplot, der mit diesem Befehl erzeugt wird, ist Abbildung 87 in dem Fenster unten rechts zu entnehmen. Das lineare Modell, das zuvor berechnet wurde, kann nun mithilfe des folgenden Befehls in den Scatterplot eingezeichnet werden, um visuell zu überprüfen, wie stark die Werte um den Scatterplot streuen und um visuell bewerten zu können, wie gut das lineare Modell den Zusammenhang der Daten beschreibt:

```
abline (LM_1, col = "red")
```

Der Befehl `abline()` wird also zum Einzeichnen des Linearen Modells verwendet. Mit *col = „red“* kann die Farbe der eingezeichneten Geraden festgelegt bzw. variiert werden. Abbildung 88 ist das Resultat zu entnehmen, das sich bei der Ausführung des oben genannten Befehls ergibt.

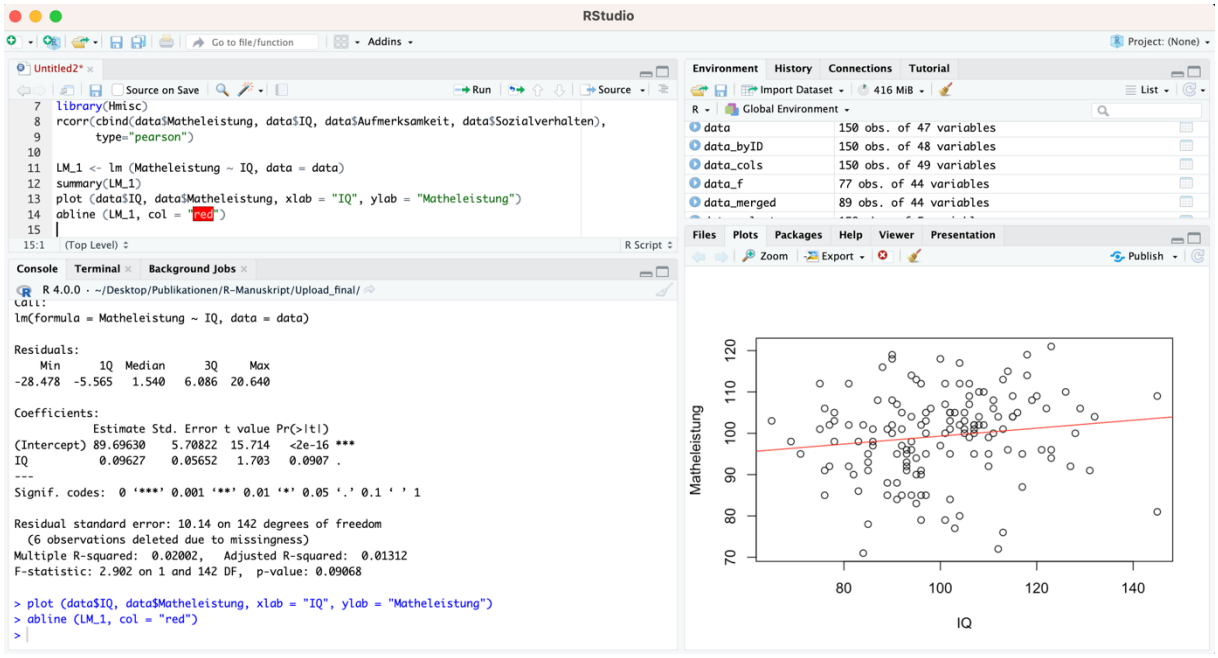

Abbildung 88: Lineares Modell im Scatterplot

5 Vertiefung und Ausblick

Im Folgenden wird ein vertiefender Ausblick auf weitere statistische Ansätze zur Datenanalyse gegeben, wobei diese Ansätze über das Grundniveau, das es mit dem Skript zu vermitteln gilt, hinausgehen. Jedoch kann es mit Blick auf die Bearbeitung spezifischer Fragestellungen (z.B. im Rahmen von Bachelorarbeiten oder anderen Abschlussprüfungen) notwendig sein, sich statistischer Methoden zu bedienen, die über diese Grundlagen, die in diesem Manuskript zuvor dargestellt wurden, hinausgehen. Daher werden im Folgenden einige weitere statistische Ansätze dargestellt und es wird auf weiterführende Literatur verwiesen.

5.1 Grundlagen bayesianischer Statistik

In Kapitel 4 wurde bereits dargestellt, wie die klassische frequentistische Testung von Hypothesen funktioniert. Klassischerweise wird die Nullhypothese getestet und es wird überprüft, wie wahrscheinlich die vorliegenden (oder krassere) Werte unter der Annahme sind, dass die Nullhypothese stimmt. Wenn die Daten unter der Annahme der Nullhypothese zu unwahrscheinlich sind (dies bedeutet in der Regel, dass der *p*-Wert kleiner als das Signifikanzniveau von 0.05 ist), so wird die Nullhypothese verworfen und die Alternativhypothese angenommen. Kritisiert wird bei diesem Ansatz regelmäßig, dass die Alternativhypothese selbst nie getestet wird und dass aus der Annahme, dass die Nullhypothese nicht zutrifft, quasi automatisch geschlussfolgert wird, dass die Alternativhypothese zutreffen muss. Ein weiterer Kritikpunkt ist, dass bei hinreichend großen Stichproben quasi jeder Unterschied/ Zusammenhang signifikant wird, da der *p*-Wert auch von der Stichprobengröße abhängig ist. Außerdem ist es mit den klassischen frequentistischen Methoden nicht möglich die Nullhypothese selbst zu beweisen: Diese wird zwar beibehalten, wenn der *p*-Wert größer als das Signifikanzniveau bzw. 0.05 ist, allerdings bedeutet dies nicht automatisch, dass die Nullhypothese somit bestätigt bzw. bewiesen wäre. Es kann vielmehr auch so sein, dass der Datenumfang bzw. die Stichprobengröße nicht ausreichend groß ist, um die Nullhypothese zu verwerfen – es liegt also eventuell ein Effekt vor, allerdings ist dieser nur mit einer größeren Stichprobengröße nachweisbar, da dieser Effekt zwar vorhanden, aber doch recht klein ist.

Die bayesianische Statistik bietet eine Alternative zur klassischen frequentistischen Statistik, um zu überprüfen, ob eine Hypothese wahrscheinlicher als eine andere Hypothese ist. Mit bayesianischen Ansätzen werden einige (aber nicht alle) Probleme der frequentistischen Statistik gelöst, allerdings weist auch die bayesianische Statistik Schwächen auf: z.B. sind die Ergebnisse, die man erhält, von sogenannten *Prior-Verteilungen* abhängig, die bewusst gewählt werden müssen, und somit kann das Ergebnis der Berechnung potenziell beeinflusst werden.

Allerdings ist das Setzen einer Priorverteilung bzw. eines Priors ebenso willkürlich wie das Setzen des Signifikanzniveaus auf 0.05. Ein großer Vorteil der bayesianischen Statistik ist, dass mit ihr auch Evidenz für die Nullhypothese nachgewiesen werden kann. Außerdem kann die bayesianische Statistik als Ergänzung zur frequentistischen Statistik genutzt werden: Wenn die Ergebnisse der Berechnungen mit beiden Ansätzen prinzipiell in die gleiche Richtung weisen, so ist dies ein Nachweis für die Robustheit der Ergebnisse – daher werden in neueren empirischen Studien teilweise frequentistische und bayesianische Ansätze miteinander kombiniert, um wissenschaftliche Erkenntnisse auf Robustheit zu überprüfen (z.B. Kißer et al., 2020; Kißler et al., 2021).

Bei der Datenauswertung mit bayesianischen statistischen Ansätzen ergeben sich ähnlich wie bei der frequentistischen Statistik Kennzahlen, die zu interpretieren sind. Bei der frequentistischen Statistik ist insbesondere der *p*-Wert von Interesse: Bei einem *p*-Wert, der kleiner als das Signifikanzniveau bzw. 0.05 ist, wird die Nullhypothese verworfen und die Alternativhypothese wird angenommen, während bei einem *p*-Wert, der 0.05 beträgt oder größer als 0.05 ist, die Nullhypothese beibehalten wird. Die Beibehaltung der Nullhypothese bedeutet aber nicht, dass diese Nullhypothese nun bewiesen wurde. Bei der bayesianischen Datenanalyse ergeben sich keine *p*-Werte, sondern Bayes-Faktoren (*BF*s). Der Bayes-Faktor ist dabei typischerweise als der Quotient aus 1) der Evidenz für die Alternativhypothese (im Zähler des Quotienten) und 2) der Evidenz für die Nullhypothese (im Nenner des Quotienten) zu verstehen. Somit gibt der *BF* das Verhältnis an, inwiefern unter Berücksichtigung der vorliegenden Daten Evidenz für die Alternativhypothese bzw. die Nullhypothese vorliegt. Ein *BF* von über 1 spricht also dafür, dass die Evidenz für die Alternativhypothese überwiegt und ein *BF* von unter 1 spricht daher dafür, dass die Evidenz für die Nullhypothese überwiegt. Von einer schwachen Evidenz für die Alternativhypothese wird bei einem BF von mindestens 3 gesprochen, von einer mittleren oder moderaten Evidenz für die Alternativhypothese wird bei einem *BF* von mindestens 10 gesprochen und bei einem BF von mindestens 30 wird von einer starken Evidenz für die Alternativhypothese gesprochen (Kißler et al., 2021; Wagenmakers et al., 2018). Andersherum gilt: Von einer schwachen Evidenz für die Nullhypothese wird bei einem *BF* von kleiner als 0.33 gesprochen, von einer mittleren oder moderaten Evidenz für die Nullhypothese wird bei einem *BF* von niedriger als 0.10 gesprochen und bei einem *BF* von kleiner als 0.033 wird von einer starken Evidenz für die Nullhypothese gesprochen (Kißler et al., 2021; Wagenmakers et al., 2018).

Wichtig ist dabei zu beachten, dass diese Art der Interpretation der Bayes-Faktoren nur möglich ist, wenn die Bayes-Faktoren (wie bereits beschrieben) als Quotient aus 1) der Evidenz

für die Alternativhypothese (im Zähler des Quotienten) und 2) der Evidenz für die Nullhypothese (im Nenner des Quotienten) angegeben werden. Sollte der Quotient andersherum berechnet worden sein (also der Quotient aus der Evidenz für die Nullhypothese im Zähler des Quotienten und der Evidenz für die Alternativhypothese im Nenner des Quotienten gebildet worden sein), so wären die obigen Kennwerte exakt andersherum zu interpretieren: Ein *BF* von über 1 spräche dann also dafür, dass die Evidenz für die Nullhypothese überwiegen würde und ein *BF* von unter 1 würde dann dafür sprechen, dass die Evidenz für Alternativhypothese überwiegen würde. Jedoch ist es üblich, dass der *BF* als Quotient aus 1) der Evidenz für die Alternativhypothese (im Zähler des Quotienten) und 2) der Evidenz für die Nullhypothese (im Nenner des Quotienten) angegeben wird.

Sollte der *BF* logarithmisch oder mithilfe einer anderen mathematischen Manipulation verändert worden sein, so sind die obigen Grenzen zur Interpretation des *BF* nicht ohne Weiteres anwendbar, sondern ebenfalls entsprechend anzupassen. Insbesondere die Verwendung logarithmisch manipulierter *BF* ist nicht unüblich und einige Wissenschaftler:innen bevorzugen die Verwendung solcher logarithmischen *BF* aus unterschiedlichen Gründen (z.B. Anobile et al., 2022). Im Folgenden werden alle Berechnungen allerdings so ausgeführt, dass die Interpretation der *BF* mit Blick auf die oben dargestellten Kennzahlen möglich ist.

Außerdem ist anzumerken, dass im Rahmen der bayesianischen Statistik nur die Evidenz für die (zwei) betrachteten Hypothesen in Relation gesetzt werden. Dies bedeutet, dass die eine Hypothese zwar besser sein kann als die andere Hypothese, aber dennoch potenziell beide Hypothesen falsch sein können und die Wahrheit einer dritten, nicht getesteten Hypothese entspricht.

Für vertiefende Informationen zum Thema bayesianische Statistik ist insbesondere das Buch von Kruschke (2015) zu empfehlen. Eine fundierte Diskussion über die Nützlichkeit von BFs zur Bewertung der Gültigkeit von Hypothesen sowie eine kritische Diskussion zu Problemen bei der Interpretation von BFs stellen Lavine & Schervish (1999) dar. Grundlegende Informationen zur prinzipiellen Denkweise bayesianischer Ansätze sind aber auch in den Methodenteilen der Publikationen von Kißler et al. (2020) sowie Kißler et al. (2021) zu finden – diese Publikationen können auch genutzt werden, um einen (ersten) Einblick darüber zu bekommen, wie bayesianische Ansätze in der Praxis zur Lösung statistischer Probleme im Rahmen der Erziehungswissenschaft/ Psychologie/ Rehabilitationswissenschaften angewendet werden können und wie die Ergebnisse bayesianischer Berechnungen im Rahmen von

Abschlussarbeiten berichtet werden können. Zur weiteren Vertiefung kann z.B. die Veröffentlichung von Aczel et al. (2020) dienen.

5.2 Bayesianische *t*-Tests

Bayesianische *t*-Tests stellen eine Alternative zu den weiter vorne in diesem Werk dargestellten frequentistischen *t*-Tests dar. Sie können verwendet werden, wenn z.B. überprüft werden soll, ob bei unterschiedlichen Gruppen bezüglich einer Variable (z. B. bezüglich der Aufmerksamkeitsleistung) Gleichheit vorliegt. Außerdem kann sich die Verwendung von bayesianischen Testverfahren anbieten, um die Ergebnisse frequentistischer Testverfahren auf Robustheit zu überprüfen. Wenn bayesianische und frequentistische Testverfahren zu Testwerten führen, welche die gleiche Interpretation der Daten nahelegen, so können die Ergebnisse als robust betrachtet werden (Kißler et al., 2021).

Damit bayesianische *t*-Tests durchgeführt werden können, ist zuvor das *Package BayesFactor* von Morey et al. (2022) zu installieren sowie zu aktivieren (vgl. Abbildung 89). Ebenso wie bei den *t*-Tests, die bereits in Kapitel 4.3 besprochen wurden, gibt es auch für bayesianische *t*-Tests zwei Varianten, welche analog zu klassischen *t*-Tests anzuwenden sind: 1) bayesianische *t*-Tests für verbundene bzw. abhängige Stichproben und 2) bayesianische *t*-Tests für unverbundene bzw. unabhängige Stichproben. In diesem Kapitel werden für die Variablen, für die in Kapitel 4.3 klassische *t*-Tests durchgeführt wurden, nun bayesianische *t*-Tests durchgeführt.

Analog zu Kapitel 4.3 wird zunächst ein bayesianischer *t*-Test für unabhängige Stichproben für die abhängige Variable *Aufmerksamkeit* (eine metrisch skalierte Variable) und die unabhängige Variable *Autistisch* (Ausprägungen: *Ja/ Nein*) im Datensatz *data* durchgeführt. Dazu wird die folgende Abfolge von Befehlen verwendet:

```
sub1 <- as.data.frame(select(data, Aufmerksamkeit, Autistisch))
sub1 <- na.omit(sub1)
ttestBF(formula = Aufmerksamkeit ~ Autistisch,
     data = sub1, paired = FALSE, rscale="medium")
```

Der bayesianische *t*-Test selbst wird mit der Funktion `ttestBF()` aus dem *Package BayesFactor* von Morey et al. (2022) berechnet. Die Funktion `ttestBF()` kann jedoch nur korrekt ausgeführt werden, wenn die Variablen, die berücksichtigt werden sollen, keine fehlenden Werte aufweisen. Daher sollte zunächst immer ein Teildatensatz aus dem Gesamtdatensatz erzeugt bzw. gefiltert werden, welcher nur die Variablen, die beim *t*-Test

berücksichtigt werden sollen, umfasst, damit im nächsten Schritt die fehlenden Werte aus diesem Teildatensatz entfernt werden können. Diese Schritte werden mit den zwei Zeilen umgesetzt, die im oben dargestellten Code über der Zeile mit der `ttestBF()`-Funktion stehen. In der ersten Zeile des oben genannten *Codes* wird also zunächst ein Teildatensatz *sub1* erzeugt, indem mit der `select()`-Funktion aus dem Gesamtdatensatz *data* die Variablen *Aufmerksamkeit* und *Autistisch* herausgezogen werden. Die `as.data.frame()`-Funktion umschließt dabei die `select()`-Funktion, um sicherzustellen, dass das Ergebnis der Operation (= der Teildatensatz *sub1*) von `RStudio` in jedem Fall als Datensatz erkannt wird. Anschließend werden in der nächsten Zeile mit der `na.omit()`-Funktion die fehlenden Daten aus dem Datensatz entfernt, damit im nächsten Schritt die `ttestBF()`-Funktion auf den Datensatz *sub1* angewendet werden kann. Der Aufbau der `ttestBF()`-Funktion folgt prinzipiell dem Aufbau, der bereits in vorherigen Kapiteln vorgestellt wurde. Neu am Aufbau der Klammer ist lediglich der Teil *rsclae="medium":* mit *rsclae=* kann der sogenannte Prior angepasst werden, welcher das Ergebnis der Berechnung beeinflusst.

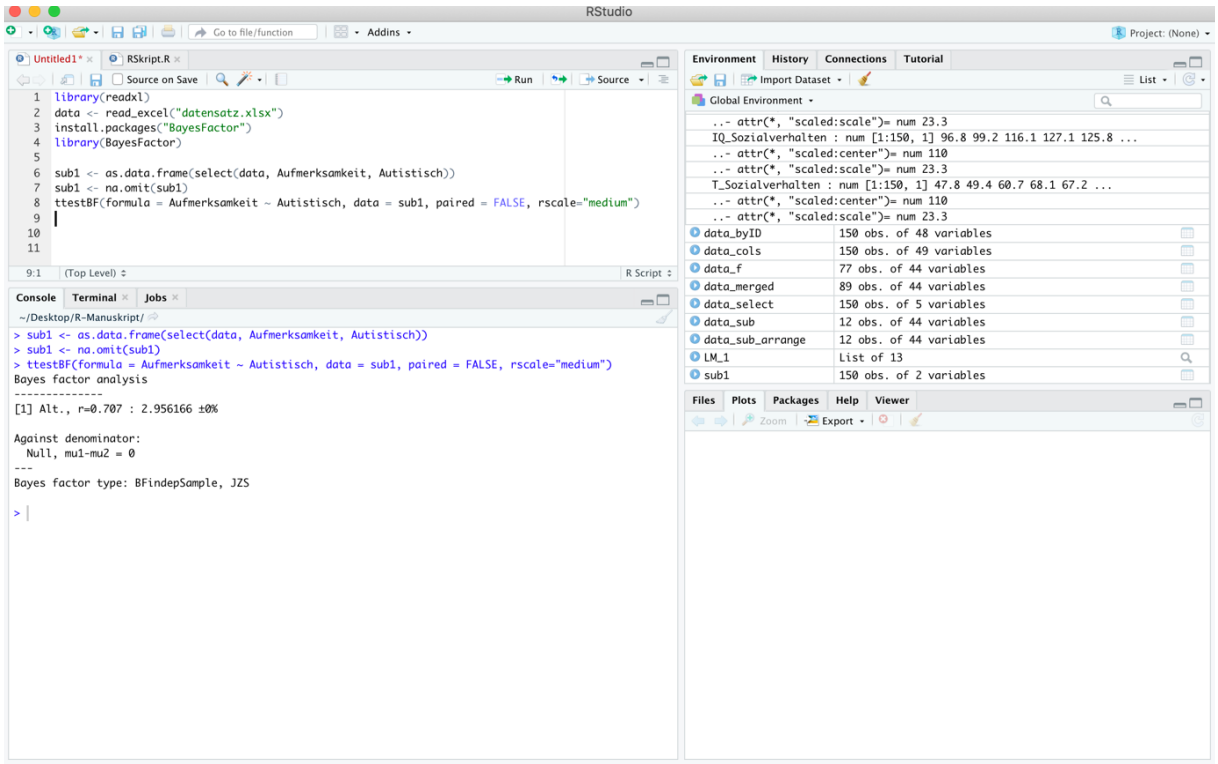

Abbildung 89: Die ttestBF()-Funktion für unabhängige Stichproben

Standardgemäß wird der medium-Prior verwendet, wobei im Rahmen von Sensitivitätsanalysen (Depaoli & van de Schoot, 2017) zusätzlich auch andere Prior verwendet werden sollten, um den Einfluss des Priors auf die Ergebnisse und somit die Robustheit der Ergebnisse bzw.

Interpretationen zu evaluieren. Die obige Syntax sowie das Ergebnis der oben genannten Befehle sind in Abbildung 89 zu erkennen.

Im Output von Abbildung 89 ist zunächst angegeben, welcher Prior bei den Berechnungen verwendet wurde (r= 0.707 steht für den medium-Prior) und dahinter steht der *BF*, der analog zu den Ausführungen in Kapitel 5.1 zu interpretieren ist. In diesem Beispiel liegt der *BF* also bei 2.956166 bzw. gerundet bei 2.96. Dieser BF von 2.96 liegt knapp unter 3, sodass entsprechend der weiter oben genannten Konvention nicht von einer schwachen Evidenz für die Alternativhypothese gesprochen werden kann, da der notwendige Grenzwert nicht überschritten wird. Allerdings liegt der BF über 1, sodass die Evidenz tendenziell in Richtung Alternativhypothese weist: Daher könnte von einer *anekdotischen Evidenz* in die Richtung der Alternativhypothese gesprochen werden.

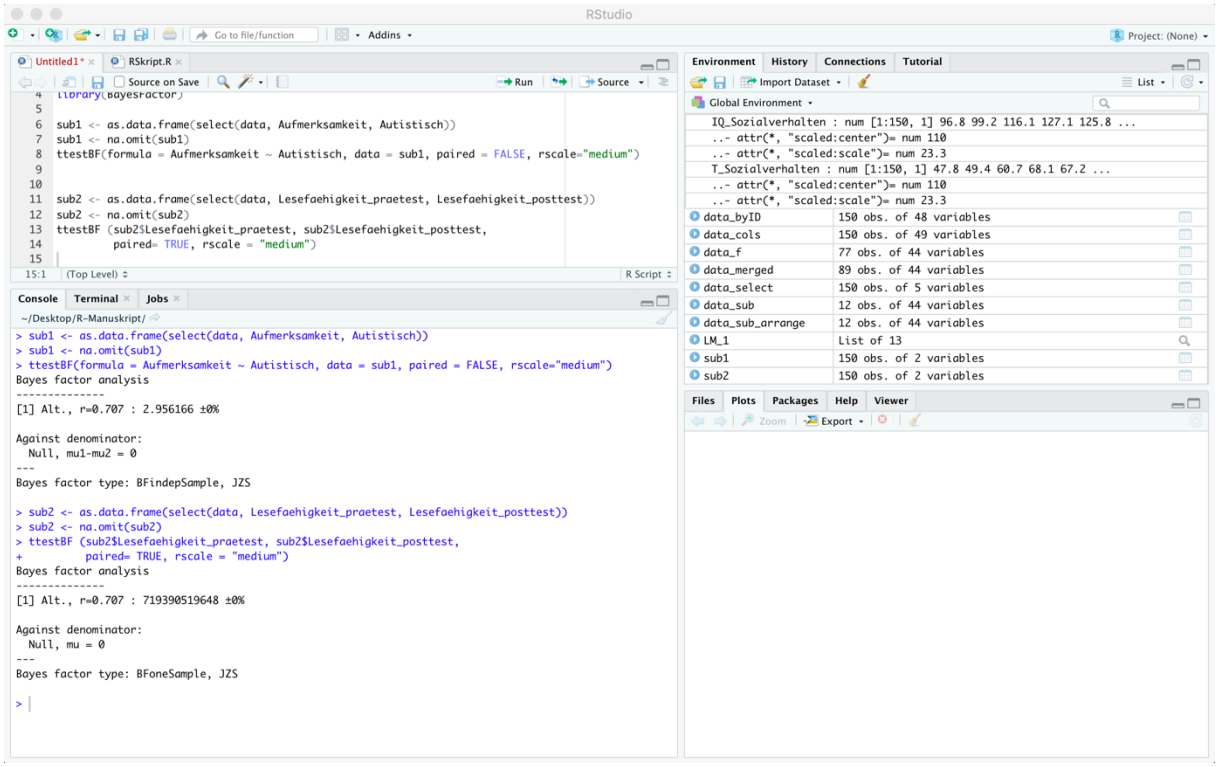

Abbildung 90: Die ttestBF()-Funktion für abhängige Stichproben

Analog zu Kapitel 4.3 wird nun ein bayesianischer *t*-Test für abhängige Stichproben für die Variablen *Lesefaehigkeit_preatest* und *Lesefaehigkeit_posttest* im Datensatz *data* durchgeführt. Dazu wird die folgende Abfolge von Befehlen verwendet:

```
sub2 <- as.data.frame(select(data, Lesefaehigkeit_praetest,
     Lesefaehigkeit_posttest))
sub2 <- na.omit(sub2)
```

```
ttestBF(sub2$Lesefaehigkeit_praetest,
     sub2$Lesefaehigkeit_posttest,
     paired= TRUE, rscale = "medium")
```

Diese Syntax für den bayesianischen *t*-Test für abhängige Stichproben ist in vielerlei Hinsicht identisch zur oben dargestellten Syntax für bayesianische *t*-Tests für unabhängige Stichproben. Abbildung 90 ist der Output für den bayesianischen *t*-Test für abhängige Stichproben für die Variablen *Lesefaehigkeit_preatest* und *Lesefaehigkeit_posttest* im Datensatz *data* zu entnehmen.

Dem Output von Abbildung 90 ist zu entnehmen, dass der *BF* bei 719390519648 bzw. $7.19*10^{11}$ liegt. Dieser BayesFactor ist analog zu den Ausführungen in Kapitel 5.1 zu interpretieren. Daher ist von einer sehr starken Evidenz für die Alternativhypothese zu sprechen.

6 Literaturverzeichnis

Aczel, B., Hoekstra, R., Gelman, A., Wagenmakers, E.-J., Klugkist, I. G., Rouder, J. N., …, Ravenzwaaij (2020). *Discussion points for Bayesian inference*. Nature Human Behaviour, 4, 561-563.

Anobile, G., Bartolli, M., Masi, G., Tacchi A., & Tinelli, F. (2022). Math difficulties in ADHD do not originate from the visual number sense. *Frontiers in Human Neuroscience*, 16:949391. doi: 10.3389/fnhum.2022.949391

Arnholt, A. T. & Evans, B. (2021). *Package 'BSDA'*. https://cran.r-project.org/web/packages/BSDA/BSDA.pdf [Abgerufen am 28.11.2022]

Bates, D., Maechler, M., Bolker, B., Walker, S., Christensen, R. H. B., …, Krivitsky, P. N. (2022). *Package 'lme4'*. https://cran.r-project.org/web/packages/lme4/lme4.pdf [Abgerufen am 28.11.2022]

Bortz, J. & Schuster, C. (2010). *Statistik für Human- und Sozialwissenschaftler* (7. Aufl.). Heidelberg: Springer.

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences (2nd ed.).* Hillsdale, NJ: Earlbaum.

Depaoli, S. & van de Schoot, R. (2017). Improving transparency and replication in Bayesian statistics: The WAMBS-Checklist. *Psychological Methods*, 22, 240 – 261. https://doi.org/10.1037/met0000065

Döring, N. & Bortz, J. (2016). *Forschungsmethoden und Evaluation in den Sozial- und Humanwissenschaften.* 5. Auflage. Berlin: Springer Verlag.

Fox, J., Nie, Z., Byrnes, J., Culbertson, M., DebRoy, S., …, R-Core. (2022). *Package 'sem'*. https://cran.r-project.org/web/packages/sem/sem.pdf [Abgerufen am 28.11.2022]

Fox, J., Weisberg, S., Price, B., Adler, D., Bates, C., …, Gregor, G. (2022). *Package 'car'*. https://cran.r-project.org/web/packages/car/car.pdf [Abgerufen am 28.11.2022]

Gastwirth, J. L., Gel, Y. R., Hui, W. L. W., Lyubchich, V., Miao, W., & Noguchi, K (2022). *Package 'lawstat'*. https://cran.r-project.org/web/packages/lawstat/lawstat.pdf [Abgerufen am 28.11.2022]

Harrell Jr., F. E. & Dupont, C. (2022). *Package 'Hmisc'*. https://cran.r-project.org/web/packages/Hmisc/Hmisc.pdf [Abgerufen am 28.11.2022]

Holling, H. & Gediga, G. (2011). *Statistik – Deskriptive Verfahren*. Bachelorstudium Psychologie. Göttingen: Hogrefe Verlag.

Holling, H. & Gediga, G. (2015). *Statistik – Testverfahren*. Göttingen: Hogrefe Verlag.

Kißler, C. (2023a). *Ein simulierter Datensatz zur Erprobung und Veranschaulichung quantitativer Methoden der Datenanalyse*. Eldorado. https://dx.doi.org/10.17877/DE290R-24012

Kißler, C. (2023b). *R-Skript zum Lehrwerk "Eine praktische Einführung in die Statistik mit R und RStudio" von Christian Kißler, Florian Krieger und Jörg-Tobias Kuhn*. Eldorado. https://dx.doi.org/10.17877/DE290R-24033

Kißler, C., Schwenk, C., & Kuhn, J.-T. (2020). Zur Additivität kognitiver Defizitprofile bei komorbiden Lernstörungen. *Lernen Und Lernstörungen*, 1-13. https://doi.org/10.1024/2235-0977/a000310

Kißler, C., Schwenk, C., & Kuhn, J.-T. (2021). Two Dyscalculia Subtypes With Similar, Low Comorbidity Profiles: A Mixture Model Analysis. *Frontiers in Psychology*, 12, 589506. https://doi.org/10.3389/fpsyg.2021.589506

Komsta, L. & Novomestky, F. (2015). *Package 'moments*'. https://cran.r-project.org/web/packages/moments/moments.pdf

Kruschke, J. K. (2015). *Doing Bayesian Data Analysis. A Tutorial with R, JAGS, and Stan*. 2. Auflage. Amsterdam: Academic Press.

Lavine, M. & Schervish, M. J. (1999). Bayes Factors: What they are and what they are not. *The American Statistician*, 53, 119-122. Doi: https://doi.org/10.1080/00031305.1999.10474443

Lovakov, A. & Agadullina, E. R. (2019). Empirically derived guidelines for effect size interpretation in social psychology. *European Journal of Social Psychology*, 51, 485-504.

Morey, R. D., Rouder, J. N., Jamil, T., Urbanek, S., Forner, K., & Ly, A. (2022). *Package 'BayesFactor'*. https://cran.r-project.org/web/packages/BayesFactor/BayesFactor.pdf [Abgerufen am 28.11.2022]

Navarro, D. (2016). *Learning statistics with R: A tutorial for psychology students and other beginners (Version 0.6)*. https://learningstatisticswithr.com/lsr-0.6.pdf [Abgerufen am 28.11.2022]

Navarro, D. (2022). *Package 'lsr'*. https://cran.r-project.org/web/packages/lsr/lsr.pdf [Abgerufen am 28.11.2022]

Pinheiro, J., Bates, D., DebRoy, S., Sarkar, D., EISPACK authors, …, R Core Team. (2022). *Package 'nlme*'. https://cran.r-project.org/web/packages/nlme/nlme.pdf [Abgerufen am 28.11.2022]

Revelle, W. (2022). *Package 'psych'*. https://cran.r-project.org/web/packages/psych/psych.pdf [Abgerufen am 28.11.2022]

R Foundation (2016). *R Logo*. https://www.r-project.org/logo/ [Abgerufen am 23.11.2022]

Rosseel, Y., Jorgensen, T. D., Rockwood, N., Oberski, D., Byrnes, J., …, Du, H. (2022). *Package 'lavaan'*. https://cran.r-project.org/web/packages/lavaan/lavaan.pdf [Abgerufen am 28.11.2022]

RStudio (2023). *RStudio Logo Usage Guidelines.* https://www.rstudio.com/about/logos/ [Abgerufen am 23.11.2022]

Van Buuren, S., Groothuis-Oudshoorn, K., Vink, G. Schouten, R., Robitzsch, A., ..., Oberman, H. (2022). *Package 'mice'*. https://cran.r-project.org/web/packages/mice/mice.pdf [Abgerufen am 28.11.2022]

Wickham, H., François, R., Henry, L., Müller, K., & Vaughan, D. (2023). *dplyr: A Grammar of Data Manipulation*. R package version 1.1.4. https://dplyr.tidyverse.org/authors.html & https://dplyr.tidyverse.org [Abgerufen am 04.12.2023]

Wagenmakers, E.-J., Love, J., Marsman, M., Jamil, T., Ly, A., Verhagen, J., …, Morey, R. D. (2018). Bayesian inference for psychology. Part II: Example applications with JASP. *Psychonomic Bulletin & Review*, 25, 58 – 76. https://doi.org/10.3758/s13423-017-1323-7