Datenvisualisierung mit ggplot2

Sebastian Kuhn

# Datenvisualisierung mit ggplot2 - Videocasts

## Inhalt

* [Datenvisualisierung mit ggplot2 - Teil 2: Das erste Scatterplot](#35nkun2)
  + [Und los geht’s!](#1ksv4uv)
  + [ggplot-Befehle sinnvoll aufbauen](#z337ya)

## Datenvisualisierung mit ggplot2 - Teil 2: Das erste Scatterplot

Herzlich Willkommen zum zweiten Teil der Video-Tutorial-Reige zu ggplot2. Nachdem ich im ersten Video einiges über die theoretischen Grundlagen von ggplot2 und über die Grundidee der “Grammar of Graphics” erzählt habe, wird es jetzt Zeit, die Theorie zu verlassen und zur Praxis zu kommen.

### Die Datenbasis

Bevor wir aber unser erstes ggplot gemeinsam erstellen, will ich kurz etwas zur Datenbasis sagen, die ich hier verwende. Sie sehen oben hier im Environment, dass bereits ein Datensatz geladen ist - und zwar handelt es sich dabei um einen Auszug aus dem *Quality of Government*-Datensatz des [Quality of Government Institute](https://www.gu.se/en/quality-government) der Universität Göteborg. Der Fokus dieses Datensatzes liegt - wie es der Name erahnen lässt - auf *Good Governance* und beinhaltet Mikro- und Makrodaten aus unterschiedlichen Quellen, die für über 190 Länder aufbereitet werden. Die Daten sind dabei thematisch - beispielsweise nach den Bereichen Bildung, Gesundheit, Migration und Umwelt - gegliedert und der Datensatz liegt in mehreren Varianten vor - so beinhaltet der [Standard-Datensatz](https://www.gu.se/en/quality-government/qog-data/data-downloads/standard-dataset), den wir hier auch nutzen, beispielsweise über 2.000 Variablen wohingegen der [Basic-Datasatz](https://www.gu.se/en/quality-government/qog-data/data-downloads/basic-dataset) eine kompaktere Version mit nur etwa 300 Variablen darstellt. Darüber hinaus liegen viele Daten auch in einem [Zeitreihendatensatz](https://www.gu.se/en/quality-government/qog-data/data-downloads/standard-dataset) vor. Für dieses Tutorial habe ich einen Auszug aus dem Standard-Datensatz sowie aus dem Zeitreihen-Datensatz erstellt, später werden wir auch einen Auszug aus dem Zeitreihen-Datensatz verwenden, und diese Auszüge enthalten einige Variablen weniger, so dass der Datensatz nicht mehr ganz so groß ist.

Wie bereits erwähnt, habe ich diesen Auszug aus dem Datensatz in der Session bereits geladen, im Skript finden Sie aber hier unter “Import des QOG Standard-Datensatzes” und “Import des QOG Zeitreihen-Datensatzes” Beispiele zum Laden der vollständigen Datensätze über die Original-URL des Quality of Government Institute. Bevor wird die Datensätze importieren, laden wir via library(tidyverse) alle Pakete des tidyverse-Universums - darunter neben ggplot2 auch das Paket readr, welches auch die Funktion read\_csv() beinhaltet, die wir hier verwenden.

***## Wichtige Pakete laden und Datensatz importieren***  
library(tidyverse) *# lädt alle Pakete des "tidyverse" - unter anderem ggplot2 und readr*  
  
*# Import des QOG Standard\_Datensatzes*  
  
qog <- readr::read\_csv("~/Dokumente/R Projekte/ggplot\_FUH\_Videocast/Data/qog\_sample.csv")  
*#qog <- readr::read\_csv("/home/seba/Dokumente/R Projekte/ggplot\_tut/Data/qog\_std\_cs\_jan21.csv")*  
*#qog <- readr::read\_csv("https://www.qogdata.pol.gu.se/data/qog\_std\_cs\_jan21.csv") #Standard-Datensatz*  
  
*# Import des QOG Zeitreihen-Datensatz*  
  
qog\_ts <- readr::read\_csv("~/Dokumente/R Projekte/ggplot\_FUH\_Videocast/Data/qog\_ts\_sample.csv")  
*# qog\_ts <- readr::read\_csv("/home/seba/Dokumente/R Projekte/ggplot\_tut/Data/qog\_std\_ts\_jan21.csv")*  
*#qog\_ts <- readr::read\_csv("https://www.qogdata.pol.gu.se/data/qog\_std\_ts\_jan21.csv") #Zeitreihen-Datensatz*

### Und los geht’s!

#### Empfehlung

So, und damit geht’s los. Bevor wir anfangen möchte ich Ihnen empfehlen, alle Übungen die wir hier machen auch tatsächlich manuell durchzuführen, indem Sie den Code wirklich selbst eintippen (statt ihn zu kopieren und einzufügen). Jetzt werden Sie sich vielleicht fragen: “Code selbst eintippen?” Das klingt in Zeiten des “Copy und Paste” vielleicht etwas antiquiert, ist aber bei weitem der effektivste Weg, um sich an die Logik von ggplot zu gewöhnen und ein Gefühl für die Syntax von R zu bekommen. Nur wenn Sie selbst Fehler machen - und ich bin mir sicher: das werden Sie unweigerlich tun - werden Sie auch schnell in der Lage sein, die eigenen Fehler zu diagnostizieren und daraus zu lernen.

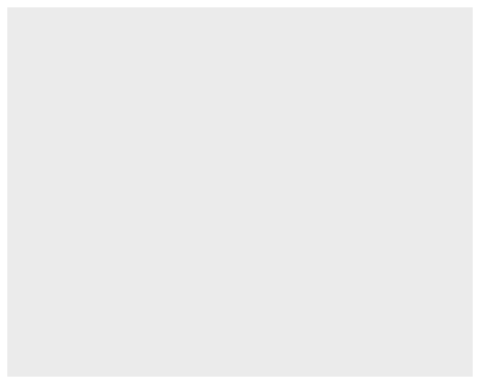
#### Das erste Streudiagramm

Wenn wir jetzt zum ersten Streudiagramm kommen, dann nehmen wir einfach mal an, wir interessieren uns für den Zusammenhang zwischen der Höhe staatlicher Gesundheitsausgaben und der Lebenserwartung in allen Ländern, für die wir eben Daten haben. Beide Variablen finden sich im Quality of Government-Datensatz - und zwar unter der Bezeichnung wdi\_dgovhexp - für die öffentlichen Gesundheitsausgaben - und unter der Bezeichnung wdi\_lifexp - für die Lebenserwartung. Das Präfix “wdi” deutet übrigens auf die ursprüngliche Datenquelle hin - beide Variablen stammen aus den “World Development Indicators”-Datensatz der Welt Bank.

***## Variablen:***  
*# wdi\_dgovhexp: Öffentliche Ausgaben für Gesundheit aus inländischen Quellen als Anteil der Wirtschaft, gemessen am BIP.*  
*# wdi\_lifexp: Die Lebenserwartung bei der Geburt gibt die Anzahl der Jahre an, die ein Neugeborenes leben würde, wenn die zum Zeitpunkt seiner Geburt vorherrschenden Sterblichkeitsmuster zum Zeitpunkt seiner Geburt sein ganzes Leben lang gleich bleiben würden.*

So, jetzt wollen wir aber wirklich loslegen mit ggplot und dazu teilen wir ggplot in der Funktion ggplot() erstmal mit, in welchem Datensatz sich die Variablen befinden, die uns interessieren. Im ersten Video hatte ich ja bereits gesagt, dass die Funktion ggplot() grundsätzlich als erstes Argument Daten erwartet, in diesem Fall also die Daten “qog”. Wenn wir diesen Befehl ausführen passiert erstmal…

***## Daten übergeben***  
ggplot(data = qog)



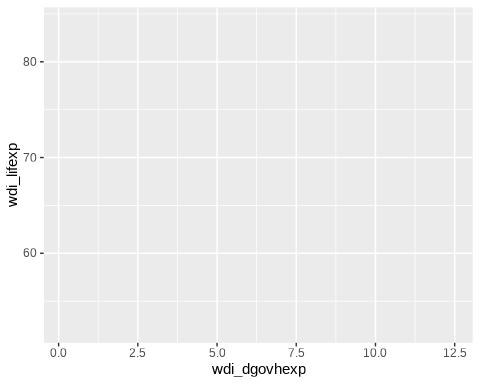
*# ggplot(qog) # Führt zum selben Ergebnis*

…nichts. Naja, das ist nicht ganz richtig - wie Sie gesehen haben hat sich hier im Plot-Fenster das Fenster “eingegraut”, d.h. Sie haben eine große graue Fläche erhalten - vielmehr ist aber nicht passiert. Warum?

Naja, im Prinzip kennt ggplot ja jetzt nur unsere Daten, weiß aber noch nicht, welche Variablen wir auf welche Weise darstellen wollen. Bevor wir weiter machen nochmal kurz der Hinweis, dass die Angabe von “data=” in diesem Fall optional ist - ich habe ja gesagt: die ´Funktion ggplot() geht grundsätzlich davon aus, dass als erstes Argument ein Datensatz übergeben wird und dementsprechend würde auch der Befehl ohne die Angabe “data=” zum selben Ergebnis führen.

Wir müssen ggplot also als nächstes mitteilen, welche Variablen in den Daten durch welche visuelle Elemente im Diagramm dargestellt werden sollen. Diese Zuordnung erfolgt - wie auch im ersten Video schon erläutert - über die “Aesthetics-” bzw. aes()-Funktion, die sich in aller Regel sofort an die Angabe des Datensatzes anschließt - und das sieht dann in etwa so aus:

*# Das "Mapping-Argument" - die Zuordnung von Variablen zu Aesthetics*  
ggplot(qog, aes(x = wdi\_dgovhexp, y = wdi\_lifexp))



Jetzt haben wir der ggplot()-Funktion also zwei Argumente gegeben: Erstens sagen wir ggplot welchen Datensatz es verwendet werden soll und wo die Variablen zu finden sind, die im Zusammenhang des Plots verwendet werden sollen. Zwar können in einer ggplot-Darstellung durchaus auch zwei oder mehr Datensätze verwendet werden, in aller Regel finden sich aber alle Variablen in einem Datensatz.

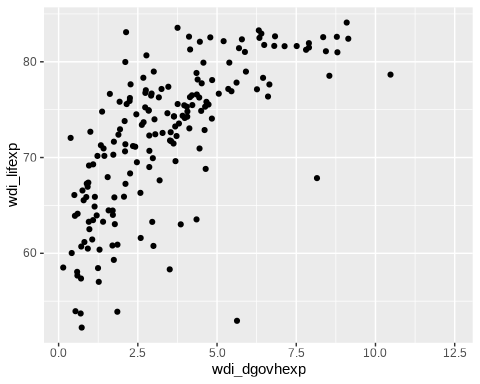
Als nächstes folgt das sogenannte “Mapping-Argument”, das innerhalb der aes()-Funktion angegeben wird. Die Argumente, die wir der Funktion übergeben, sind im Prinzip nichts anderes als Definitionen, die ggplot später verwenden wird. Im vorliegenden Fall könnte man den Codeschnipsel also folgendermaßen übersetzen:

“Nimm bitte den Datensatz”qog" zur Hand und stelle die Variable “wdi\_dgovhexp” auf der x-Achse und die Variable “wdi\_lifexp” auf der y-Achse dar". Wie wir später noch sehen werden, könnten wir Variablen hier auch zu anderen ästhetischen Elementen zuordnen - beispielsweise zu Farben, Formen oder Linientypen - erstmal bleiben wir hier aber bei der Zuordnung zu den Achsen.

Jetzt führen wir den Befehl einmal aus - und wie Sie sehen ist durch die Ausführung des Befehls etwas mehr passiert als vorhin. Die Ausgabe ist nun nicht mehr vollständig weiß oder grau, sondern R hat bereits die x- und y-Achse erstellt und beschriftet und eine grundlegende Design-Entscheidung bezüglich des Hintergrunds und einiger Hilfslinien getroffen - und das alles ohne dass Sie eine konkrete Aufforderung im ggplot-Code geschrieben hätten. Das liegt daran, dass für diese Ebenen bereits Voreinstellungen definiert wurden, die nun standardmäßig abgerufen werden.

Das wichtigste aber - nämlich die Darstellung der beiden Variablen - fehlt noch. Das liegt schlicht und ergreifend daran, dass das Mapping alleine noch nichts darüber sagt, wie die Darstellung genau erfolgen soll - ob also Punkte, Balken, Linien oder andere geometrische Objekte gezeichnet werden sollen. Wir müssen dem Plot daher noch mindestens eine weitere Ebene hinzufügen, die über die Art des Diagramms entscheidet. Konkret bedeutet das, dass wir uns für eine der knapp 50 vordefinierten geometrischen Funktion entscheiden müssen, die in ggplot enthalten sind. In diesem Fall verwenden wir geom\_point() - ohne Angabe weiterer Argumente in der Klammer. Durch die bereits vorab definierte Zuordnung der jeweiligen Variablen zu den x- und y-Achsen ist jetzt schon klar, welche x- und y-Werte in einem Streudiagramm darstellt werden sollen. Hinzugefügt wird die geom\_-Funktion über ein den “+”-Operator.

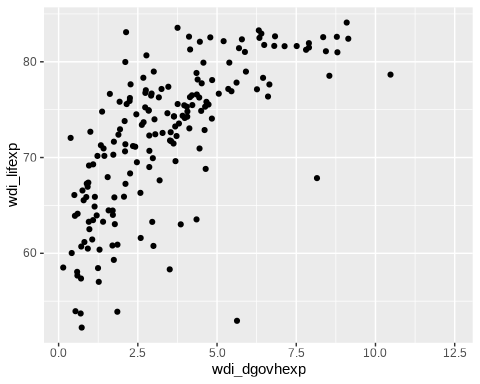
*# Das erste Streudiagramm*  
ggplot(qog, aes(x = wdi\_dgovhexp, y = wdi\_lifexp)) + *# Neue Ebenen werden mit dem "+"-Operator hinzugefügt*  
 geom\_point()



Jetzt führen wir auch diesen Befehl einmal aus - und siehe da: Schon ist das erste Streudiagramm fertig! Zugegeben, es sieht noch nicht wirklich berauschend aus und würde so vermutlich nicht in einer Studienarbeit oder in einem Journal publiziert werden, aber der erste Schritt ist schon mal getan. Bevor wir uns jetzt aber weiteren Details von ggplot2 widmen, möchte ich Sie an der Stelle noch auf einige praktische Punkte hinweisen.

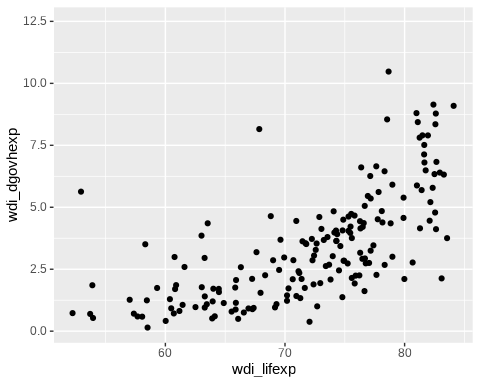
Wenn Sie in Lehrbüchern oder im Internet ggplot2-Code sehen, dann werden Sie merken, dass es sehr unterschiedliche Vorlieben bezüglich des Code-Stils gibt und dass insbesondere erfahrende R-Nutzer manchmal dazu tendieren, sehr kurzen und effizienten Code zu schreiben, der für Anfänger häufig nicht gut lesbar ist. Ein Beispiel: Mit dem folgenden Code, den ich hier markiere und einmal ausführe, kommen Sie zu dem selben Ergebnis - und das, obwohl er deutlich kürzer ist als der Code weiter oben.

*# Das erste Streudiagramm - kürzere Variante*  
ggplot(qog, aes(wdi\_dgovhexp, wdi\_lifexp)) + geom\_point()



Wie wir bereits oben bei der optionalen Angabe des “data=”-Arguments gesehen haben, sind einige Angaben optional, da die verwendeten Funktionen einige Voreinstellungen haben. Wenn man beispielsweise weiß, dass die Aesthetics-Funktion ohne nähere Angabe immer davon ausgeht, dass als erstes Argument eine x-Variable und als zweites Argument eine y-Variable übergeben wird, dann kann man sich die Angabe von “x=” und “y=” - wie wir sie oben gemacht hatten - auch schlicht und ergreifend sparen. Hier ist allerdings aus meiner Sicht Vorsicht geboten, denn wenn die Reihenfolge der Variablen vertauscht wird, ändert sich natürlich auch die Zuordnung der Variablen zur x- bzw. y-Achse. Das habe ich in dem Befehl hier einmal so gemacht - da ist einfach die Reihenfolge vertausch:

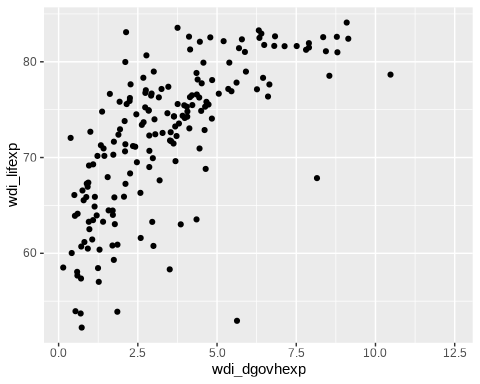
*# Das erste Streudiagramm - kürzere Variante mit anderer Reihenfolge*  
ggplot(qog, aes(wdi\_lifexp, wdi\_dgovhexp)) + geom\_point()



Und Sie sehen, dann werden die auch die Zuordnungen der Variablen zur x- und zur y-Achse verändert.

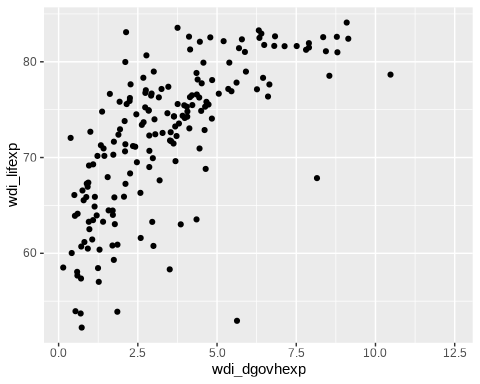
Für Anfänger bietet sich daher die explizite Angabe - auch zur besseren Lesbarkeit - durchaus an. Die explizite Angabe von “x” und “y” “schlägt” dann im Übrigen auch die Reihenfolge der Variablen - also wenn Sie mit “y=” zuerst die y-Variable angeben und dann die x-Variable (mit “x=”), dann werden Sie sehen, dass wir wieder das ursprüngliche Diagramm haben und dass die y-Variable - also in dem Fall die Lebenserwartung - auf die y-Achse geplottet wurde, obwohl es die erste Variable in der aes()-Funktion ist. Und die Gesundheitsausgaben wurden auf die x-Achse geplottet, obwohl es die zweite Variable war.

*# Das erste Streudiagramm - y vor x*  
ggplot(qog, aes(y = wdi\_lifexp, x = wdi\_dgovhexp)) + geom\_point()



Kein guter Stil ist die Definition neuer Ebenen in der gleichen Zeile - wie in dem Beispiel mit + geom\_point() in der selben Zeile wie der Rest des Codes. Das funktioniert zwar, aber wenn Sie neue ggplot-Ebenen auch einfach gleich in neue Zeilen schreiben, dann erhöht das erheblich die Übersicht und Lesbarkeit des Codes. Deswegen würde ich Ihnen empfehlen, sich das gleich von Anfang an anzugewöhnen.

*# Das erste Streudiagramm - y vor x*  
ggplot(qog, aes(y = wdi\_lifexp, x = wdi\_dgovhexp)) +   
 geom\_point() *# Zur besseren Lesbarkeit und Übersicht bitte in einer neuen Zeile*

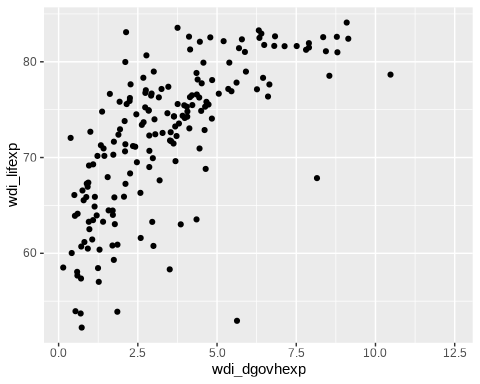


## ggplot-Befehle sinnvoll aufbauen

### ggplots als Objekte speichern

Was Sie ebenfalls in nahezu allen Lehrbüchern und Blogbeiträgen zu ggplot2 häufig sehen werden ist, dass ggplot-Nutzer ihre Grafiken meistens gar nicht direkt erstellen sondern schrittweise vorgehen und Zwischenschritte in Objekte speichern - das kommt nämlich auch der Logik der “Grammar of Graphics” im Prinzip entgegen. In unserem Fall sieht das beispielsweise so aus:

*# Daten und Mapping in "p" zwischenspeichern*  
p <- ggplot(qog, aes(y = wdi\_lifexp, x = wdi\_dgovhexp))  
*# Darstellung als Scatterplot*  
p + geom\_point()

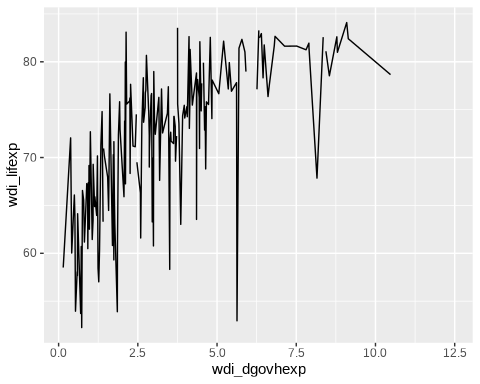


Was hier passiert, ist, dass die Informationen zum Datensatz sowie die Informationen zum Mapping der Variablen auf der x- und y-Achse in einem neuen Objekt namens “p” zwischengespeichert werden und erst im zweiten Schritt wird dann der Darstellungstyp festgelegt oder werden andere Festlegungen getroffen.

Wenn Sie diese Codeline einmal ausführen, dann passiert im Plotfenster erstmal gar nichts - was passiert, ist dass Sie im Environment ein neues Objekt namens “p” enthalten haben. Eine Darstellung im Plotfenster wird durch die Zuordnung der ggplot-Funktion zum Objekt “p” nicht ausgelöst. Wenn Sie eine Darstellung haben möchten, dann würden Sie in dem Fall einfach das Objekt “p” abrufen - und das würde in dem Fall wieder dazu führen, dass Sie (wie wir vorher in dem Beispiel schon gesehen hatten) ein mehr oder weniger leeren Plot ausgegeben bekommen in dem nur die Achsen und die Bezeichnungen angelegt sind, aber noch keine Punkte, denn: Das haben Sie ja hier in der Darstellung noch gar nicht festgelegt. Im zweiten Schritt würden Sie dann z.B. indem Sie zu dem Objekt “p” eine geom\_-Funktion hinzufügen dann (in diesem Fall ein Scatter-) Plot wirklich darstellen. Durch die Angabe “p + geom\_point()” wird Ihnen dann das gewohnte Streudiagramm ausgegeben, das wir ja schon ein paar mal gesehen haben.

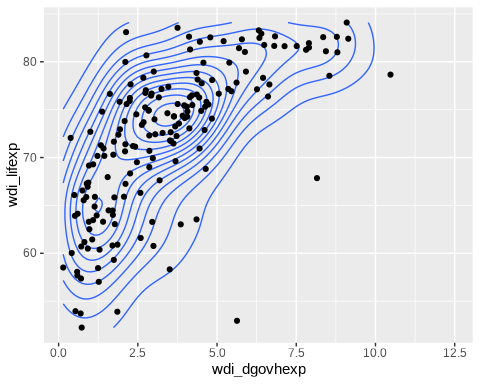
Zwei Punkte sind mir an dieser Stelle wichtig: Erstens die Erkenntnis, dass auch von ggplot erstellte Plots oder Funktionen in R als Objekte gespeichert werden können. Zweitens natürlich - und das ist der wichtige Punkt - die Frage nach dem Sinn der Übung. Meistens dient diese Arbeitsweise dazu, herumexperimentieren zu können, ohne immer wieder Code wiederholen zu müssen. Sie könnten beispielsweise auf diese Art schnell ausprobieren, ob sich nicht vielleicht ein Liniendiagramm besser zur Darstellung der Daten eignen könnte als ein Streudiagramm. In diesem Fall fügen Sie dem Objekt “p” eben nicht die Funktion geom\_point() hinzu, sondern die Funktion geom\_line() - die Ihnen ein Liniendiagramm erstellt.

p + geom\_line()



Das macht offenkundig keinen Sinn. Vielleicht wollen Sie aber mal ein Density-Plot - also ein Dichte-Diagramm - erstellen, das die Verteilung nummerischer Variable in einer bestimmten Weise abbildet - das würden Sie zum Beispiel mit diesem Code hier machen:

p + geom\_density\_2d() +   
 geom\_point()



Und auch da kann man sich jetzt darüber streiten, ob das sinnvoll war, es geht mir an der Stelle aber noch gar nicht darum zu verstehen, was hier passiert ist - wichtig ist nur: Durch diesen Aufbau des Codes (und das Zwischenspeichern von Ergebnissen in Objekten) sind Sie in der Lage, schnell und ohne viel weitere Schreibarbeit herumzuexperimentieren - zumindest solange Sie die selbe Datengrundlage und die selben Variablen verwenden. Mein Tipp: Gewöhnen Sie sich das gleich an - ich werde es in den kommenden Videos auch so machen und ich glaube es ist sinnvoll, sich das von Anfang an direkt so anzugewöhnen.

Übrigens müssen Sie das Objekt dabei natürlich nicht zwingend “p” nennen - das “p” hat sich allerdings - vermutlich als Abkürzung für “plot” - weitgehend durchgesetzt und wird Ihnen in vielen Lehrbüchern so begegnen.

Ich gebe zu, das alles mag für Anfänger etwas kompliziert erscheinen und wir haben jetzt relativ viel Zeit darauf verwendet, ein recht simples Plot zu erstellen - wichtig ist es aber gerade am Anfang das Grundkonzept von ggplot zu verstehen. Und die gute Nachricht ist nun: An diesem Konzept ändert sich auch bei komplexeren Darstellungen nichts grundsätzliches. Komplexere Darstellungen bestehen eben meistens nur aus mehr Ebenen. In den Darstellungen, die wir uns im Laufe der weiteren Videos anschauen werden, werden wir beispielsweise andere geom()-Funktionen verwenden, das Koordinatensystem, Skalen, Farben und Beschriftungen ändern - konzeptionell werden wir jedoch immer das Gleiche tun. Deswegen würde ich Ihnen gerne zum Ende des ersten Praxisvideos die zentralen konzeptuellen Schritte bei der Erstellung eines ggplots nochmal wiederholen:

Die zentralen Schritte sind - Erstens - das Sie der ggplot()-Funktion mitteilen, welche Daten Sie verwenden wollen, dass Sie dann - Zweitens - ggplot() mitteilen, welche Variablen Sie dargestellt haben wollen - und der Einfachheit halber diesen Schritt auch schon in einem Objekt zwischenspeichern. In einem dritten Schritt teilen Sie ggplot mit, welchen Darstellungstyp Sie gerne sehen wollen und in einem vierten Schritt können Sie dann weitere Zusatzfunktionen verwenden, um Skalen, Beschriftungen, Markierungen, Titel und vieles mehr hinzufügen.

*# Wir beginnen mit einem möglichst "aufgeräumten" Datensatz und gehen dann wie folgt vor:*  
*# 1. Teilen Sie der Funktion ggplot() mit, welche Daten Sie verwenden wollen.*  
*# 2. Sagen Sie ggplot(), welche Variablen Sie dargestellt haben wollen. Der Einfachheit halber speichern Sie die Ergebnisse der ersten beiden Schritte in ein Objekt namens "p".*  
*# 3. Teilen Sie ggplot mit, welchen Darstellungstyp Sie gerne sehen wollen. Verwenden Sie dazu die vorgesehenen geom\_()-Funktionen, indem Sie diese einzeln und nacheinander zum Objekt "p" hinzufügen.*  
*# 4. Verwenden Sie Zusatzfunktionen, um Skalen, Beschriftungen, Markierungen, Titel und vieles mehr hinzufügen.*

All’ das werden wir uns in den kommenden Videos anschauen - für heute würde ich es gerne dabei bewenden lassen. Bis dahin: viel Spaß beim Ausprobieren und bis zum nächsten Video!