# Übungszettel Plots und Grafiken - Exercise sheet plots and graphics

M.Psy.205, Dozent: Dr. Peter Zezula

Johannes Brachem (johannes.brachem@stud.uni-goettingen.de)

## Deutsche Version

### Links

Übungszettel als PDF-Datei zum Drucken

**Übungszettel mit Lösungen**

Lösungszettel als PDF-Datei zum Drucken

Der gesamte Übugszettel als .Rmd-Datei (Zum Downloaden: Rechtsklick > Speichern unter…)

### Hinweise zur Bearbeitung

1. Bitte beantworten Sie die Fragen in einer .Rmd Datei. Sie können Sie über `Datei > Neue Datei > R Markdown...` eine neue R Markdown Datei erstellen. Den Text unter dem *Setup Chunk* (ab Zeile 11) können Sie löschen. Unter diesem Link können Sie auch unsere Vorlage-Datei herunterladen (Rechtsklick > Speichern unter…).
2. Informationen, die Sie für die Bearbeitung benötigen, finden Sie auf der Website der Veranstaltung
3. Zögern Sie nicht, im Internet nach Lösungen zu suchen. Das effektive Suchen nach Lösungen für R-Probleme im Internet ist tatsächlich eine sehr nützliche Fähigkeit, auch Profis arbeiten auf diese Weise. Die beste Anlaufstelle dafür ist der R-Bereich der Programmiererplattform Stackoverflow
4. Auf der Website von R Studio finden Sie sehr hilfreiche Übersichtszettel zu vielen verschiedenen R-bezogenen Themen. Ein guter Anfang ist der Base R Cheat Sheet

### Ressourcen

Da es sich um eine praktische Übung handelt, können wir Ihnen nicht alle nützlichen Befehle einzeln vorstellen. Stattdessen finden Sie hier Verweise auf sinnvolle Ressourcen, in denen Sie für die Bearbeitung unserer Aufgaben nachschlagen können.

| Ressource | Beschreibung |
|---|---|
| Field, Kapitel 4 | Buchkapitel, das Schritt für Schritt in die Arbeit mit `ggplot2` einführt. **Große Empfehlung!** |
| ggplot2 Cheat Sheet | Übersicht über die meisten gängigen ggplot2-Befehle |
| Peters ggplot2-Referenz | Peter pflegt eine große Sammlung von Beispielen für Plots mit dem dazugehörigen Code: Eine Ressource zum Nachschlagen |

## Tipp der Woche

Mit der Tastenkombination `ctrl` + `shift` + `m` (Windows) oder `cmd` + `shift` + `m` (Mac) können Sie per Knopfdruck das Pipe- Symbol %>% einfügen.

## Daten einlesen

1. Setzen Sie ein sinnvolles Arbeitsverzeichnis für den Übungszettel (in der Regel der Ordner, in dem Ihre .Rmd liegt). Fügen Sie eine passende Code-Zeile an den Anfang ihres .Rmd-Dokuments ein.
2. Laden Sie den Datensatz `mpg.csv` herunter und speichern ihn in Ihrem Arbeitsverzeichnis (idealerweise haben Sie noch den Ordner vom letzten Übungszettel - speichern Sie den Datensatz im Unterordner /data).
3. Öffnen Sie `mpg.csv` mit einem einfachen Texteditor und schauen Sie, mit welchem Zeichen die einzelnen Spalten getrennt sind.
4. Laden Sie die Pakete des `tidyverse`. Fügen Sie eine passende Code-Zeile an den Anfang ihres .Rmd-Dokuments.
5. Nutzen Sie den Befehl `read_delim()`, um `mpg.csv` unter dem Namen `mpg_data` einzulesen. `read_delim()` ist eine verallgemeinerte Form von `read_csv()`, in der sie mit dem Argument `delim = "<Trennzeichen>"` manuell angeben können, durch welches Zeichen die Spalten in Ihrem Datensatz getrennt sind.

### Lösung

**Unteraufgabe 1** Hier sollte in den Anführungszeichen Ihr Dateipfad stehen.

```
# Beispiele, bitte ein Verzeichnis mit Schreibrechten angeben
setwd("~/mv")
# oder
setwd("P:/mv")
```

**Unteraufgabe 2** Bitte folgen Sie den Anweisungen im Aufgabentext.

**Unteraufgabe 3** Bitte folgen Sie den Anweisungen im Aufgabentext.

```


library(tidyverse)
```

**Unteraufgabe 4**

**Unteraufgabe 5** In den Anführungszeichen sollte Ihr eigener Dateipfad stehen. Bei mir liegt die Datei `mpg.csv` im Unterordner `data` des Ordners `sheets` in meinem Arbeitsverzeichnis.

```
# mpg_data <- read_delim("data/mpg.csv", delim = "|")
# bzw. direkt aus URL einlesen:
mpg_data <- read_delim("http://md.psych.bio.uni-goettingen.de/mv/data/div/mpg.csv", delim = "|")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   manufacturer = col_character(),
##   model = col_character(),
##   displ = col_double(),
##   year = col_double(),
##   cyl = col_double(),
##   trans = col_character(),
##   drv = col_character(),
##   cty = col_double(),
##   hwy = col_double(),
##   fl = col_character(),
##   class = col_character()
## )
```

## Grundlegendes zu Grafiken

Zum Erstellen von Grafiken nutzen wir das Paket `ggplot2`. Da das Paket zum `tidyverse` gehört, brauchen Sie es nicht extra laden, wenn Sie bereits das `tidyverse` geladen haben (Siehe Aufgabe 1.4).

### Objekt erstellen

Grafiken mit `ggplot2` werden eigentlich immer nach dem gleichen Muster erstellt. Zunächst nutzen Sie den Befehl `ggplot()`, um ein Grafik-Objekt zu erzeugen:

```
example_plot <- ggplot(data, aes(x = variable_1, y = variable_2))
```

Sie sehen: Das erste Argument in der Klammer ist der Datensatz, den Sie verwenden möchten. Anschließend legen Sie in dem Argument `aes()` fest, welche Variable auf der x-Achse und welche auf der y-Achse dargestellt werden soll.

### Grafische Ebene hinzufügen

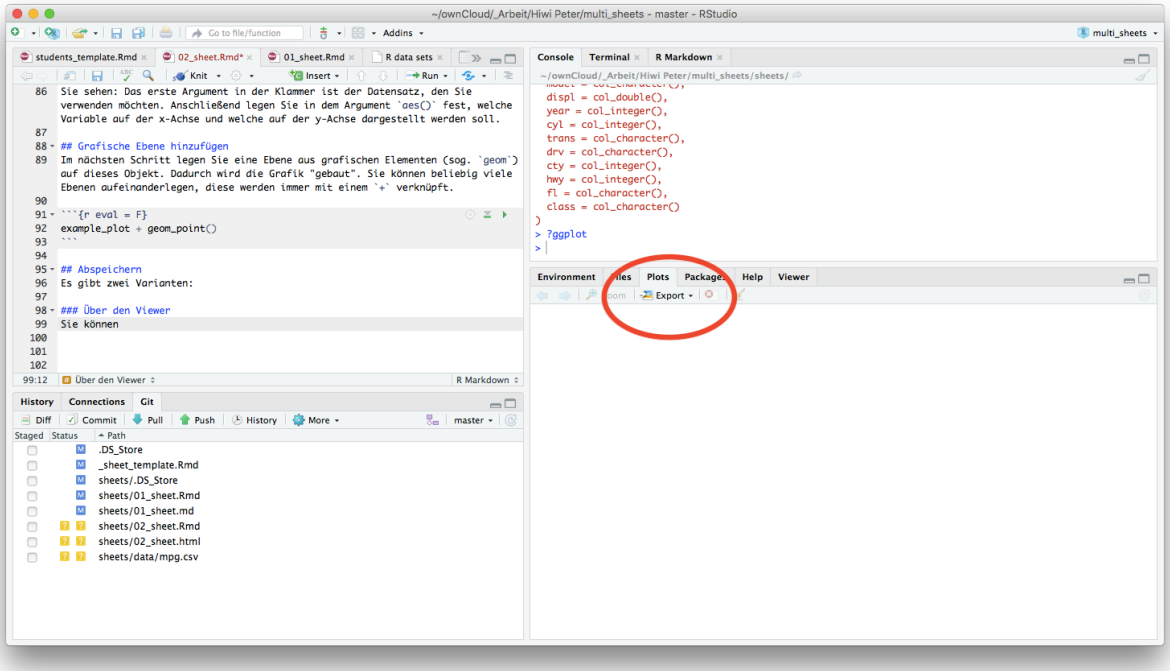
Im nächsten Schritt legen Sie eine Ebene aus grafischen Elementen (sog. `geom`) auf dieses Objekt. Dadurch wird die Grafik "gebaut". Sie können beliebig viele Ebenen aufeinanderlegen, diese werden immer mit einem + verknüpft.

```
example_plot + geom_point()
```

### Abspeichern

Es gibt zwei Varianten:

**Über den Viewer** Sie können über die “Export” Schaltfläche im Viewer arbeiten:




**Über die Syntax** Wenn Sie einen Plot über die Syntax speichern möchten, dann müssen Sie zunächst die Ebenen, die Sie hinzugefügt haben, mit abspeichern:

```
exmaple_plot <- example_plot + geom_point()
```

Dann verwenden Sie `ggsave()`, um den Plot zu speichern. Er wird in Ihrem Arbeitsverzeichnis abgelegt. Sie geben in dem Befehl zunächst den gewünschten Dateinamen in Anführungszeichen an (Endung nicht vergessen!), und geben als nächstes den Plot an, den Sie speichern möchten.

```
ggsave("example_plot.png", example_plot)
```

## Scatterplot

1. Nutzen Sie `?mpg`, um sich eine Beschreibung des Datensatzes anzeigen zu lassen.
2. Erstellen Sie mit `ggplot()` ein Objekt namens `displ_plot`, in dem Sie als Datensatz `mpg_data` spezifizieren. Auf der x-Achse sollten Sie den Hubraum (*engine displacement*) des Modells eingeben. Auf der y-Achse sollte stehen, wie viele Meilen pro Gallone Treibstoff (*city miles per gallon*) gefahren werden können.
3. Nutzen Sie `geom_point()`, um zu diesem Objekt nun eine Ebene aus Punkten hinzuzufügen.
4. Nutzen Sie `geom_smooth()` mit den Argumenten `method = "lm"` und `se = FALSE`, um dem Plot zusätzlich eine Regressionsgerade hinzuzufügen.
5. Probieren Sie aus, was passiert, wenn man bei `geom_smooth()` die beiden Argumente aus der vorherigen Aufgabe weglässt.

**Lösung**

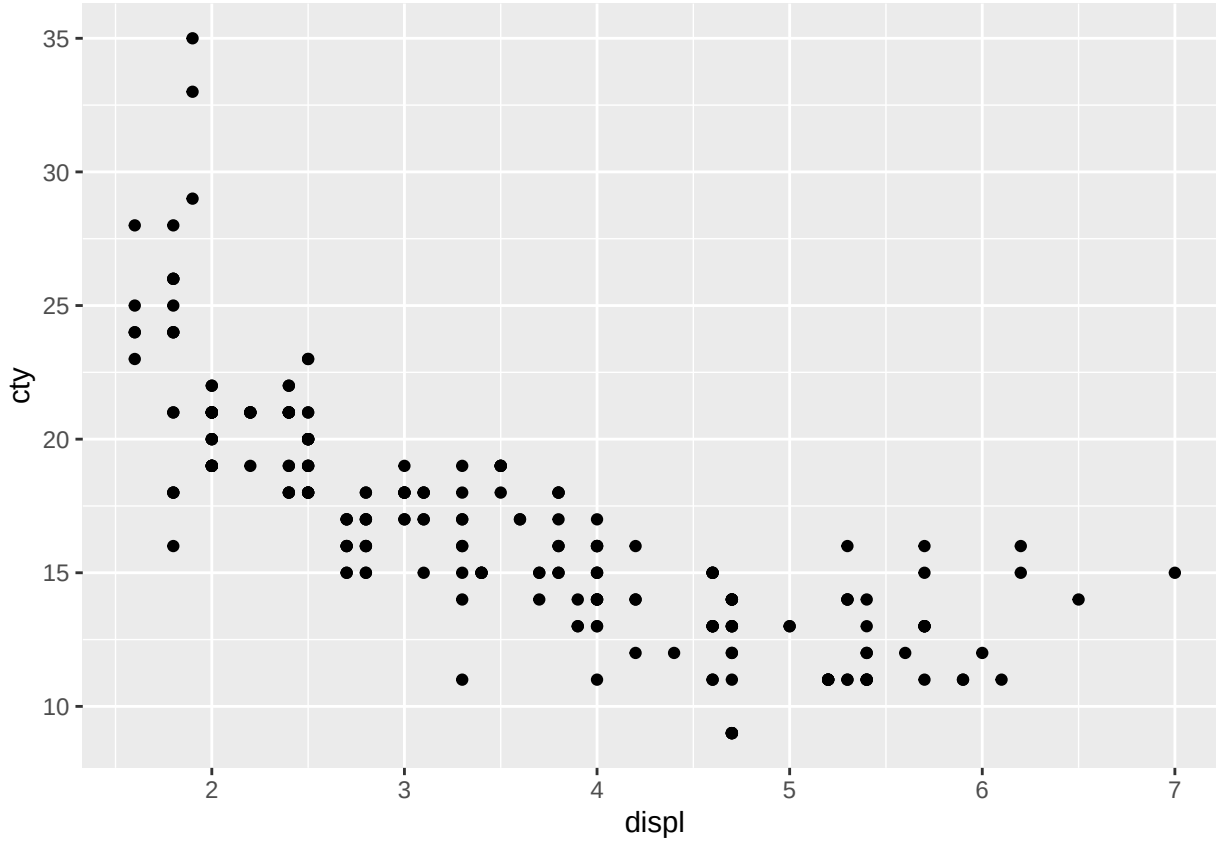
```
?mpg
```

**Unteraufgabe 1**

```
displ_plot <- ggplot(mpg_data, aes(x = displ, y = cty))
```

**Unteraufgabe 2**

```
displ_plot + geom_point()
```

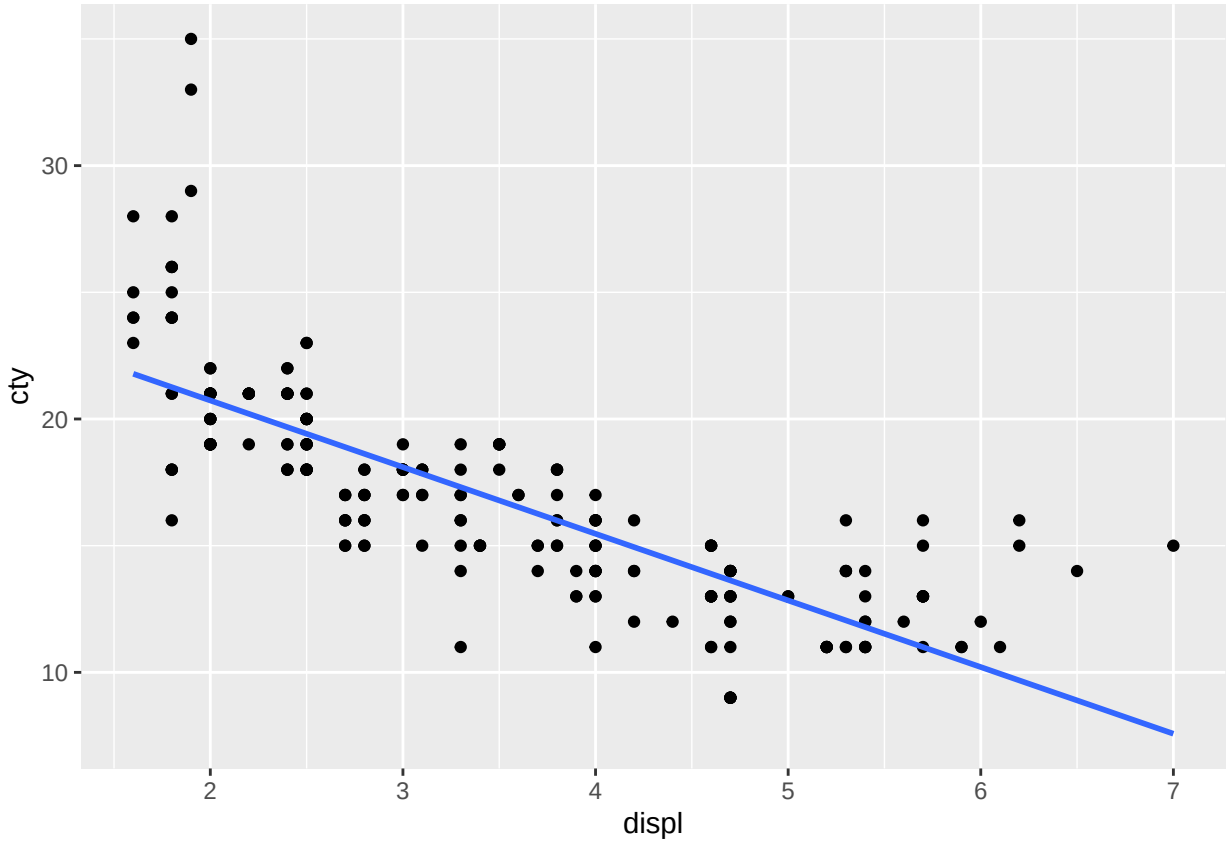



**Unteraufgabe 3**

```
displ_plot + geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

### Unteraufgabe 4

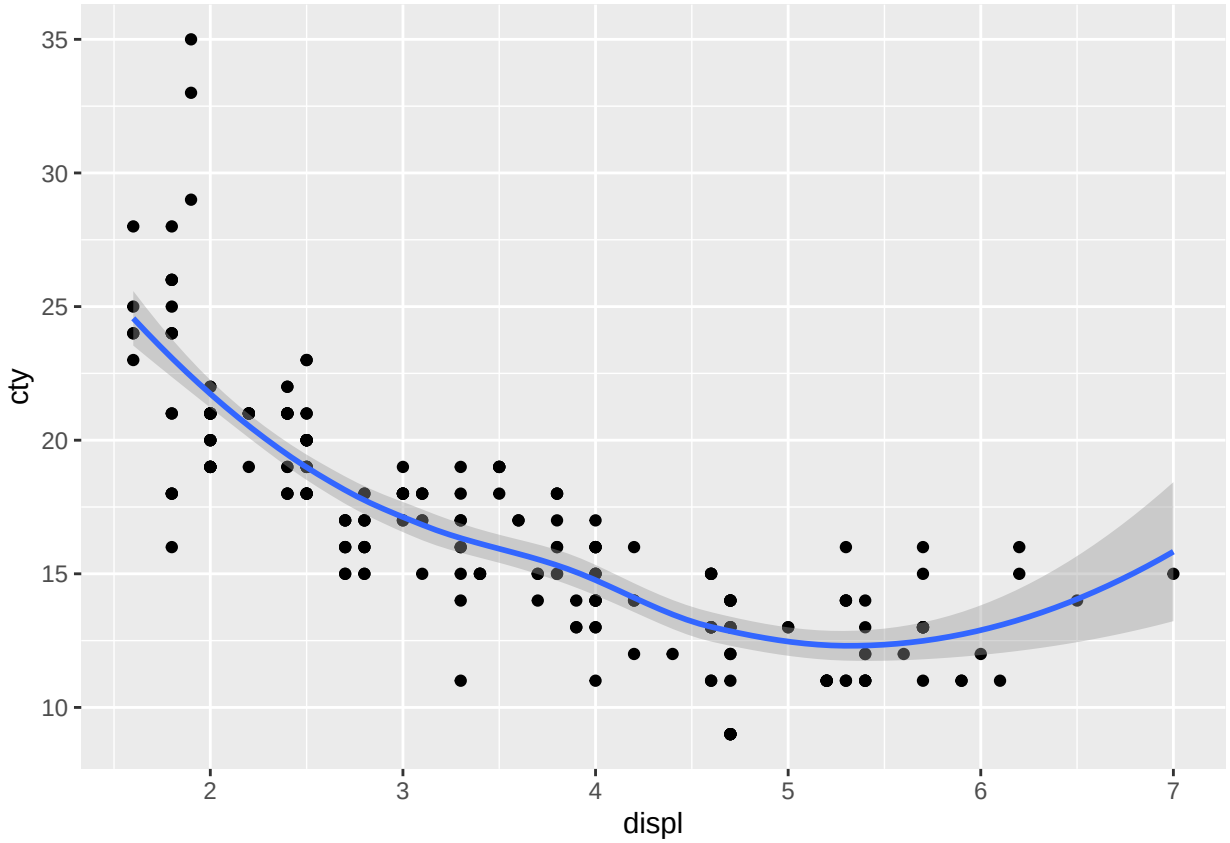
```
## `geom_smooth()` using formula 'y ~ x'
```




```
displ_plot + geom_point() +
  geom_smooth()
```

### Unteraufgabe 5

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




## Barplot

1. Erstellen Sie ein neues Plot-Objekt namens `drv_plot` mit folgenden Spezifikationen: Der Datensatz sollte wieder `mpg_data` sein, auf der x-Achse sollte die Art des Antriebs (front, rear oder 4wd) dargestellt sein, und auf der y-Achse wieder der Treibstoffverbrauch in *Miles per Gallon*.

2. Fügen Sie nun eine *summary*-Ebene mit dem Befehl `stat_summary()` zu dem Plot hinzu. Nutzen Sie in `stat_summary()` das Argument `fun.y = mean`, um `ggplot` mitzuteilen, dass Sie den Mittelwert der Beobachtungen auf der y-Achse darstellen möchten. Nutzen Sie außerdem in `stat_summary()` das Argument `geom = "bar"`, um sich einen Barplot ausgeben zu lassen.

3. Fügen Sie dem Befehl `aes()` bei der Definition des Objektes `drv_plot` das Argument `fill = drv` hinzu, um den Barplot farblich aufzuhübschen. Führen Sie den Befehl aus der vorherigen Aufgabe danach erneut aus, um den Plot zu aktualisieren.

4. Fügen Sie dem Plot nun eine Beschriftungs-Ebene mit dem Befehl `labs()` hinzu. Vergeben Sie in `labs()` mithilfe der Argumente `x = "text"`, `y = "text"`, `title = "text"`und `fill = "text"` sinnvolle Namen für die x-und y-Achse, geben Sie dem Plot einen Titel und eine Beschriftung für die Farben.

5. Verliehen Sie den Säulen einen schwarzen Rand, indem Sie im Befehl `stat_summary()` das Argument `color = "black"` hinzufügen.

### Lösung

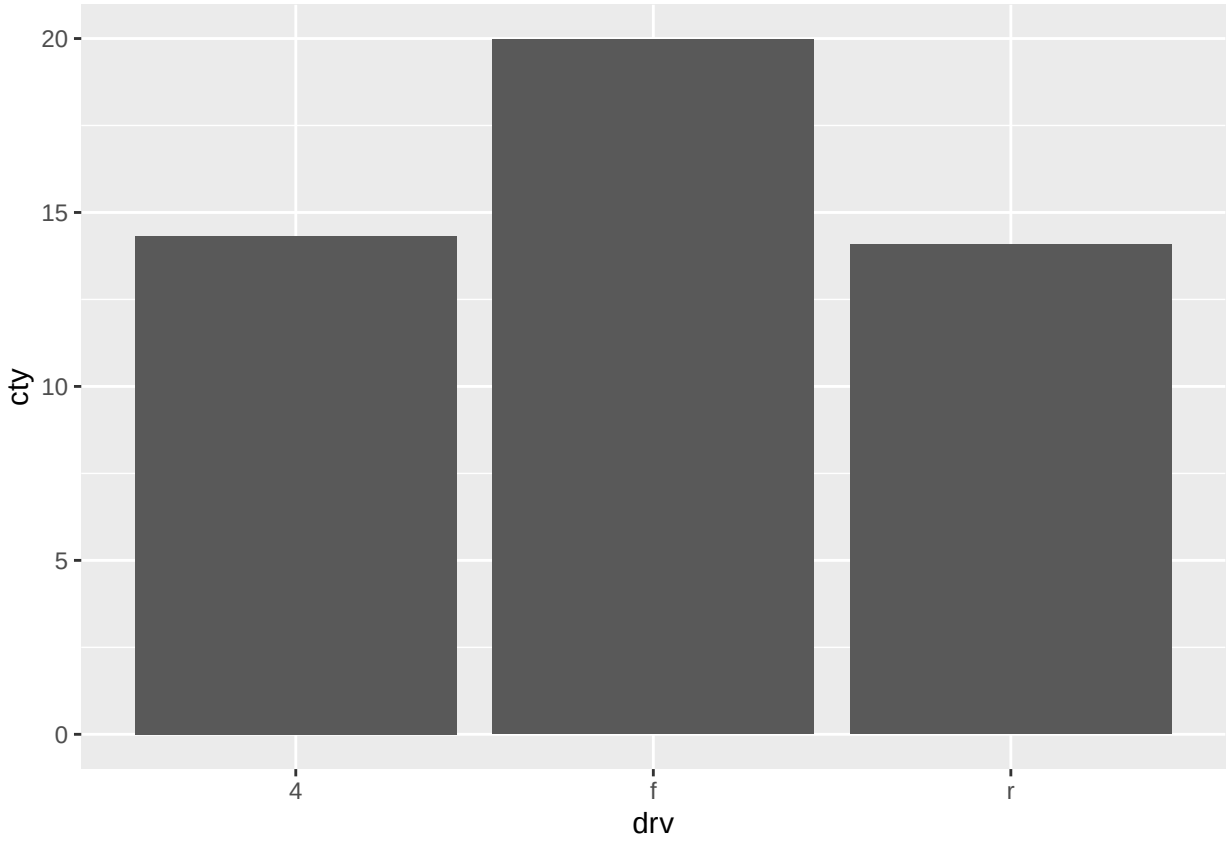
```
drv_plot <- ggplot(mpg_data, aes(x = drv, y = cty))
```

**Unteraufgabe 1**

```
drv_plot + stat_summary(fun.y = mean, geom = "bar")
```

**Unteraufgabe 2**

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```

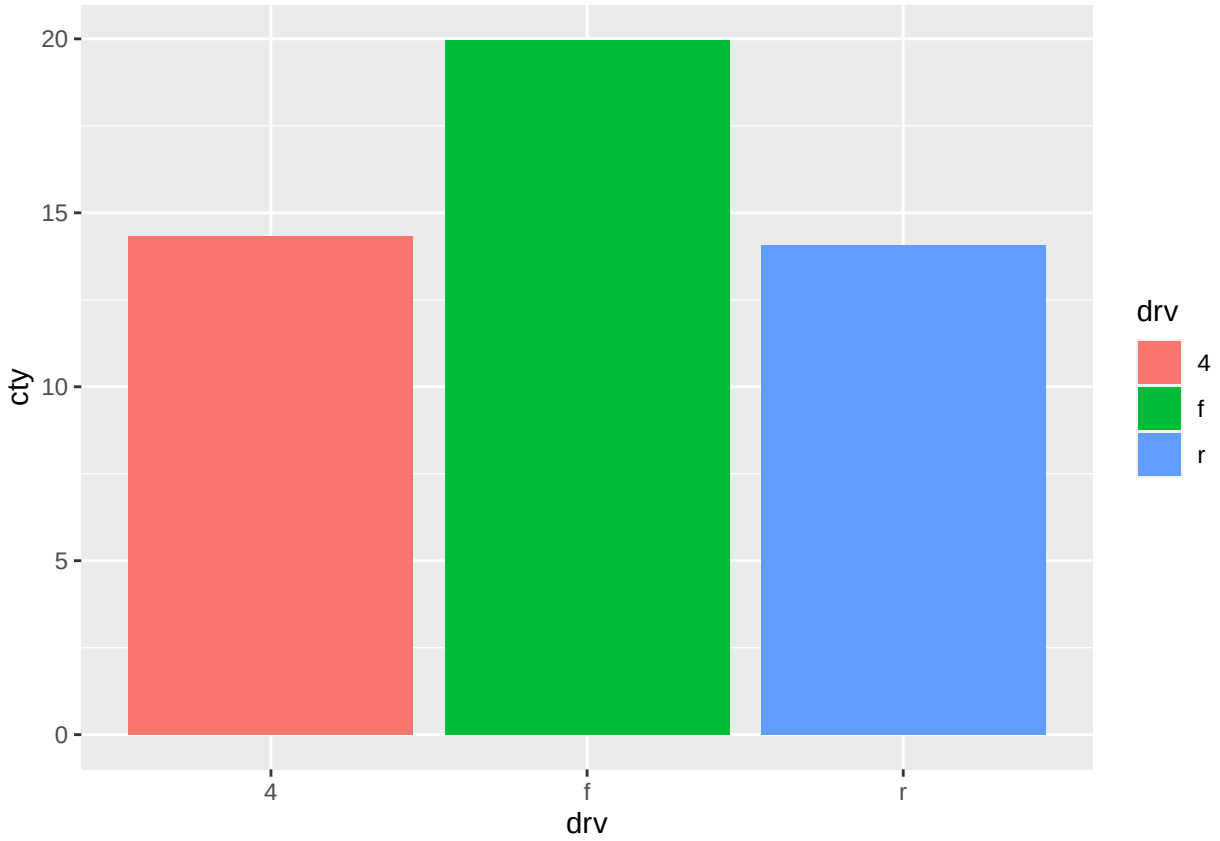



```
drv_plot <- ggplot(mpg_data, aes(x = drv, y = cty, fill = drv))

drv_plot + stat_summary(fun.y = mean, geom = "bar")
```

**Unteraufgabe 3**

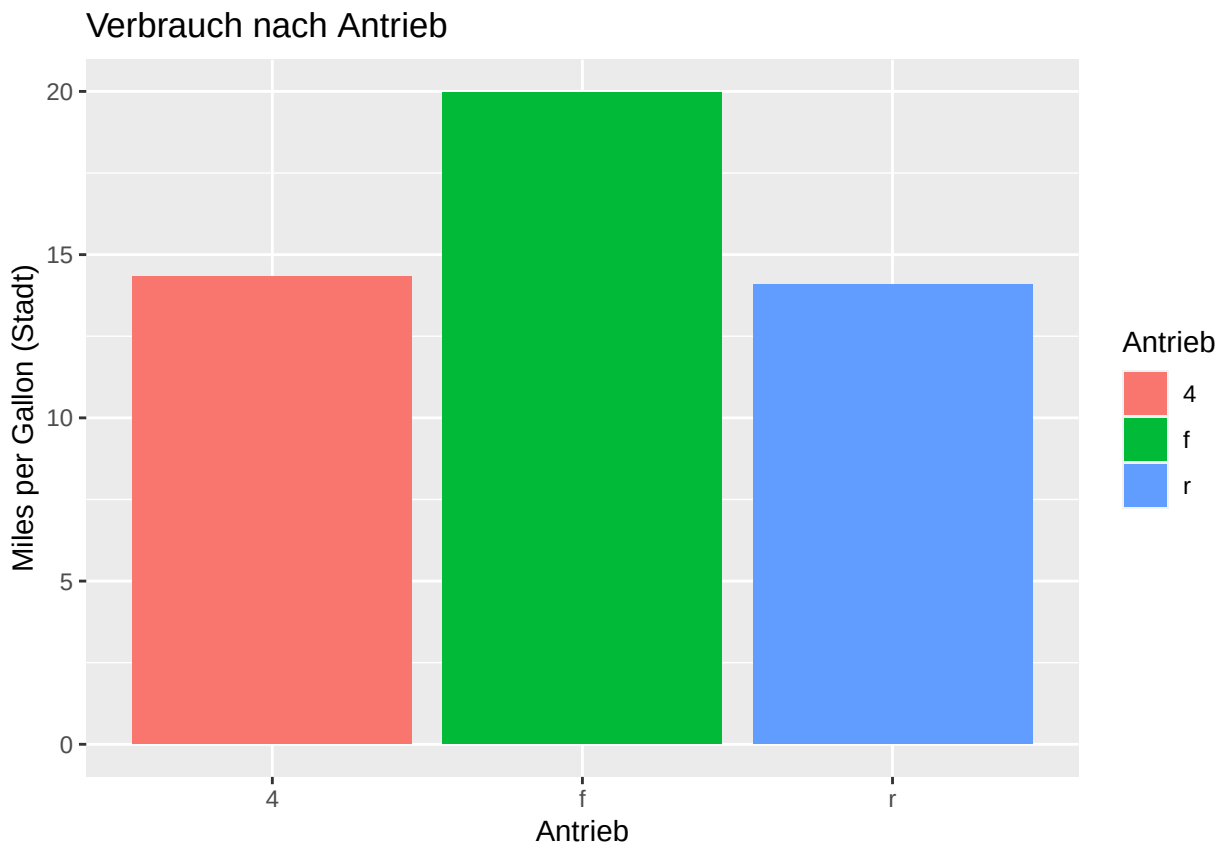
```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```




```
drv_plot + stat_summary(fun.y = mean, geom = "bar") +
  labs(x = "Antrieb",
       y = "Miles per Gallon (Stadt)",
       title = "Verbrauch nach Antrieb",
       fill = "Antrieb")
```

**Unteraufgabe 4**

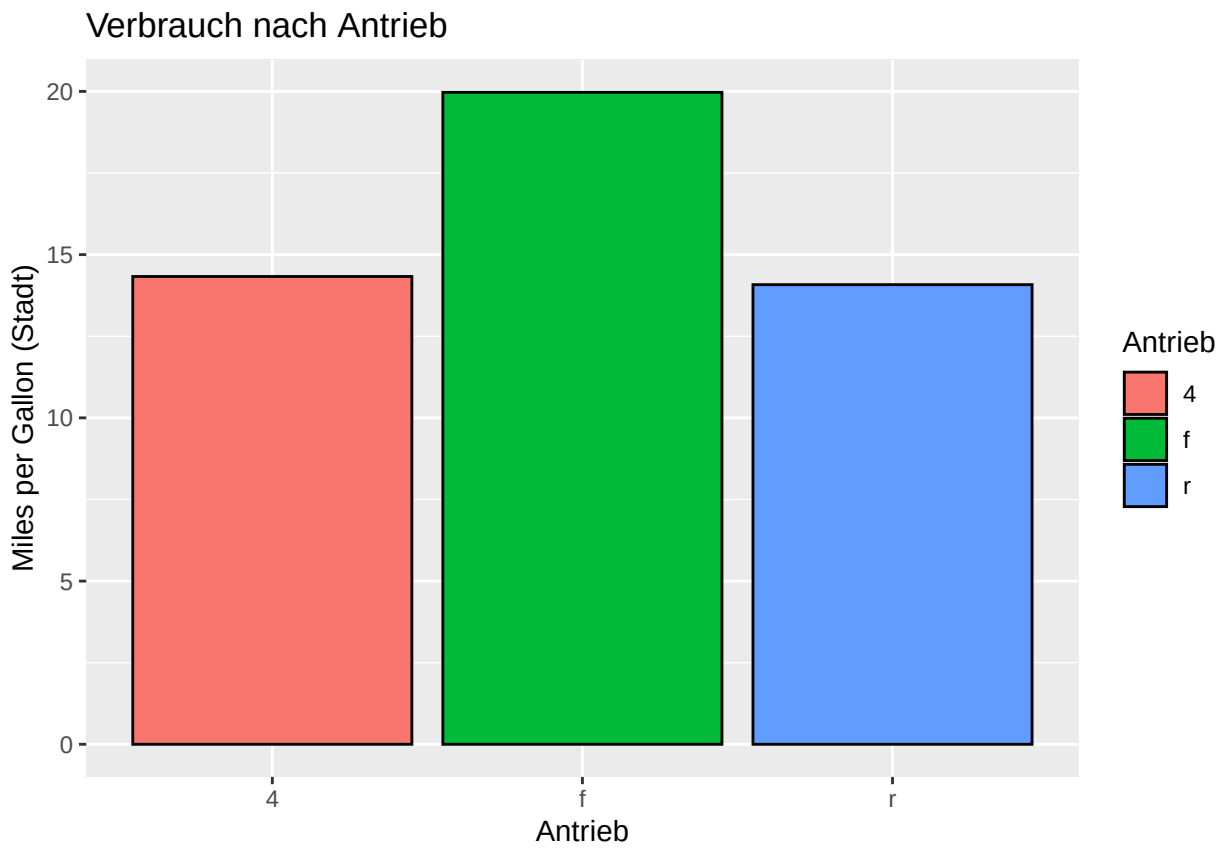
```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```




```
drv_plot + stat_summary(fun.y = mean, geom = "bar", color = "black") +
  labs(x = "Antrieb",
       y = "Miles per Gallon (Stadt)",
       title = "Verbrauch nach Antrieb",
       fill = "Antrieb")
```

**Unteraufgabe 5**

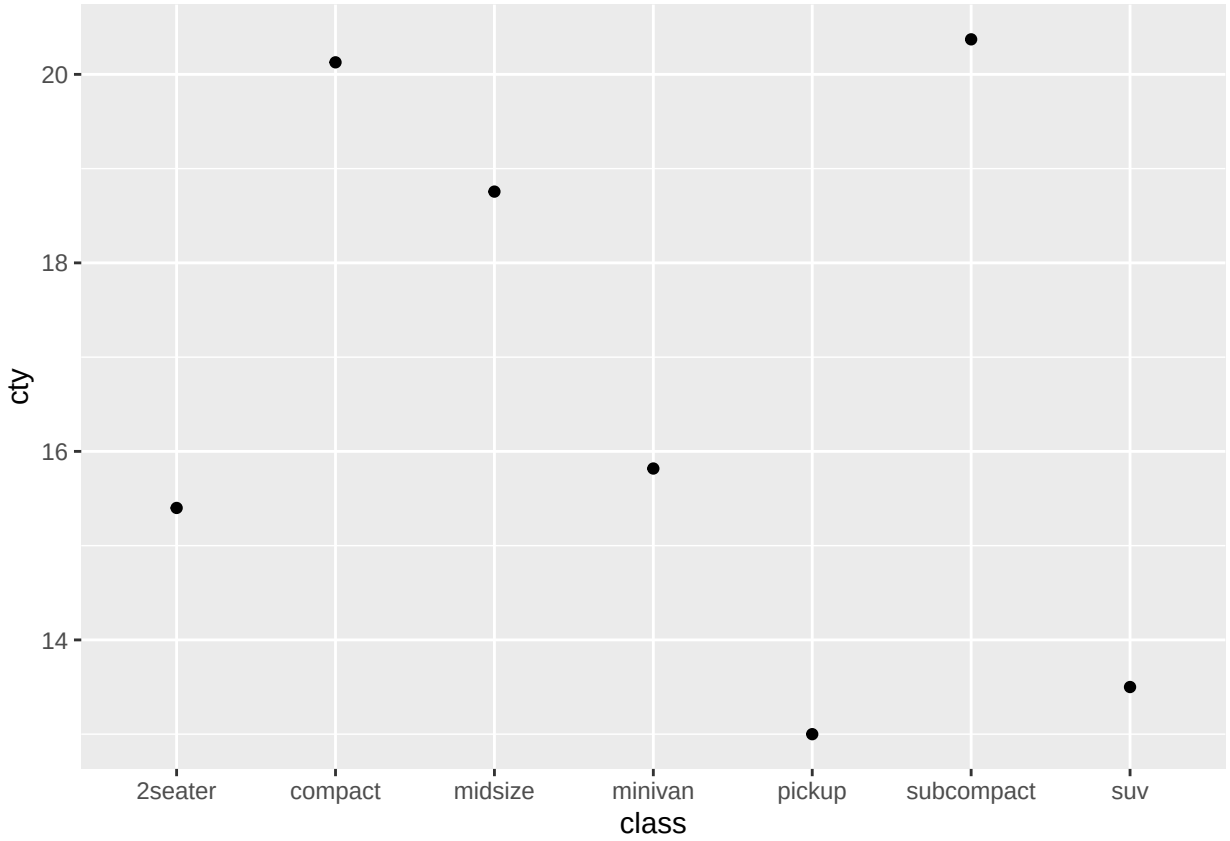
```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```




## Linienplot

1. Bauen Sie den folgenden Plot nach. Hinweis: Die Punkte stellen die Mittelwerte dar.

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```



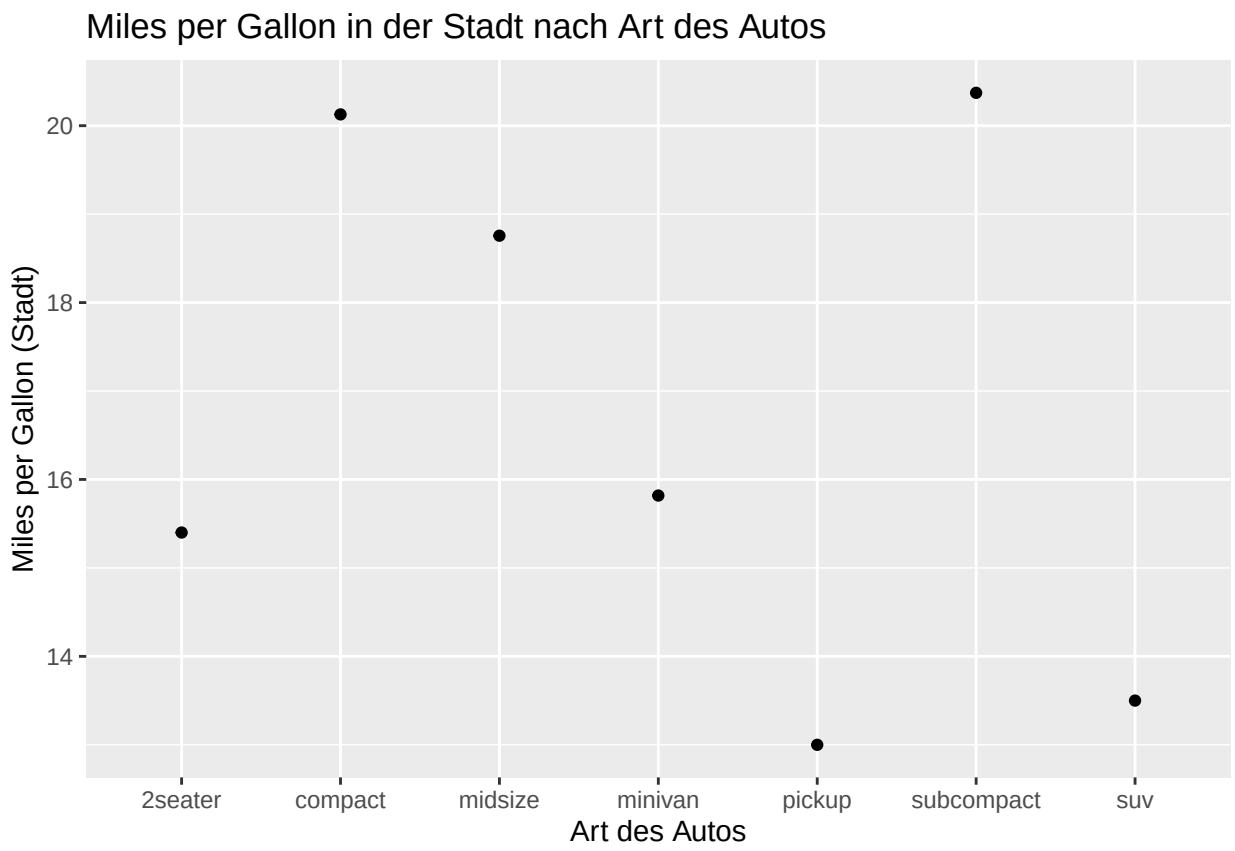

2. Nutzen Sie den Befehl `stat_summary()`, um dem Plot eine Ebene mit einer Linie hinzuzufügen, die die Punkte verbindet. Tipp: Nutzen Sie das Argument `aes(group = 1)` in `stat_summary()`, um `ggplot` mitzuteilen, dass alle Punkte in einer Gruppe gruppiert werden.
3. Nutzen Sie den Befehl `stat_summary()` mit den Argumenten `fun.data = mean_cl_normal` und `geom = "errorbar"`, um dem Plot eine Ebene mit Fehlerbalken hinzuzufügen, die die 95%-Konfidenzintervalle der Mittelwerte angeben.
4. Nutzen Sie das Argument `width = 0.2` in `stat_summary()`, um die Breite der Konfidenzintervalle einzustellen.
5. Fügen Sie dem Plot sinnvolle Achsenbeschriftungen und einen Titel hinzu.

**Lösung**

```
year_plot <- ggplot(mpg_data, aes(x = class, y = cty))
year_plot + stat_summary(fun.y = mean, geom = "point") +
  labs(x = "Art des Autos",
       y = "Miles per Gallon (Stadt)",
       title = "Miles per Gallon in der Stadt nach Art des Autos")
```

**Unteraufgabe 1**

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```

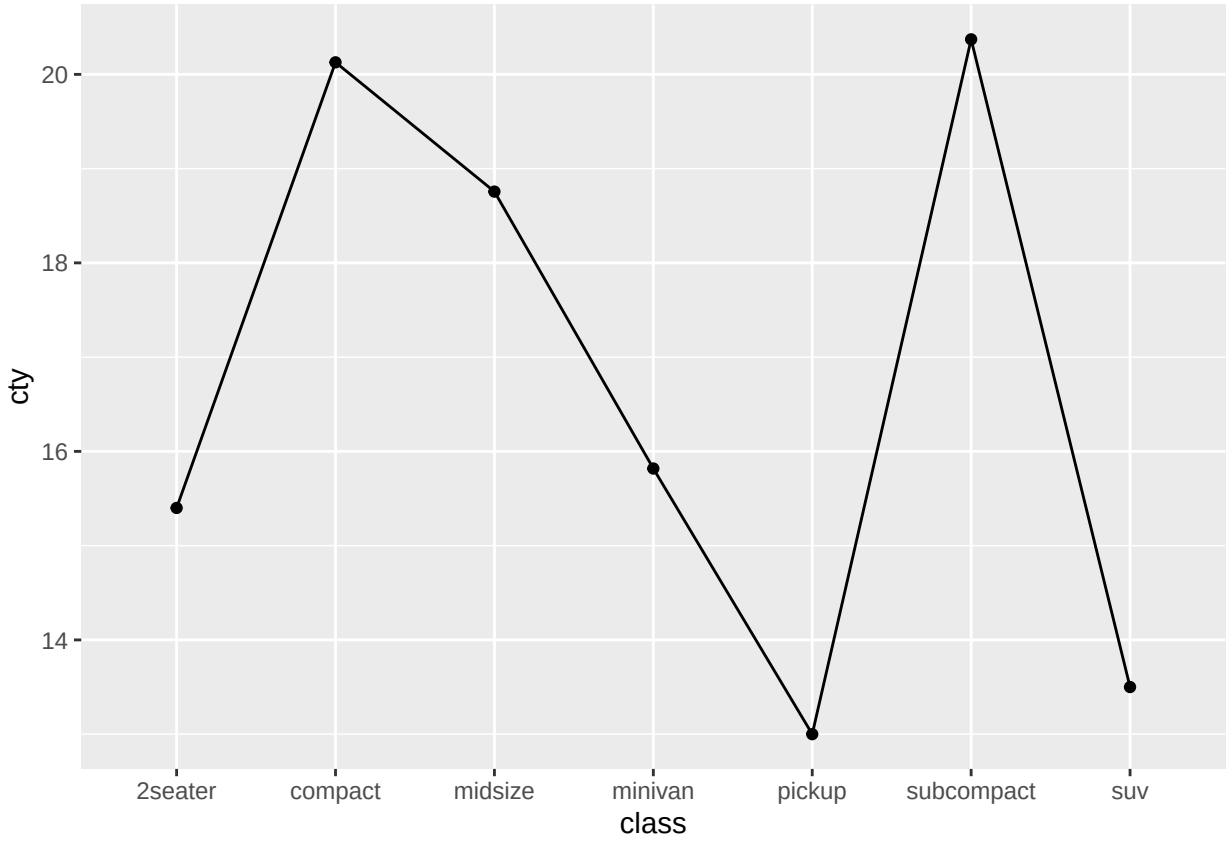



```
year_plot <- ggplot(mpg_data, aes(x = class, y = cty))
year_plot + stat_summary(fun.y = mean, geom = "point") +
  stat_summary(fun.y = mean, geom = "line", aes(group = 1))
```

**Unteraufgabe 2**

```
## Warning: `fun.y` is deprecated. Use `fun` instead.

## Warning: `fun.y` is deprecated. Use `fun` instead.
```

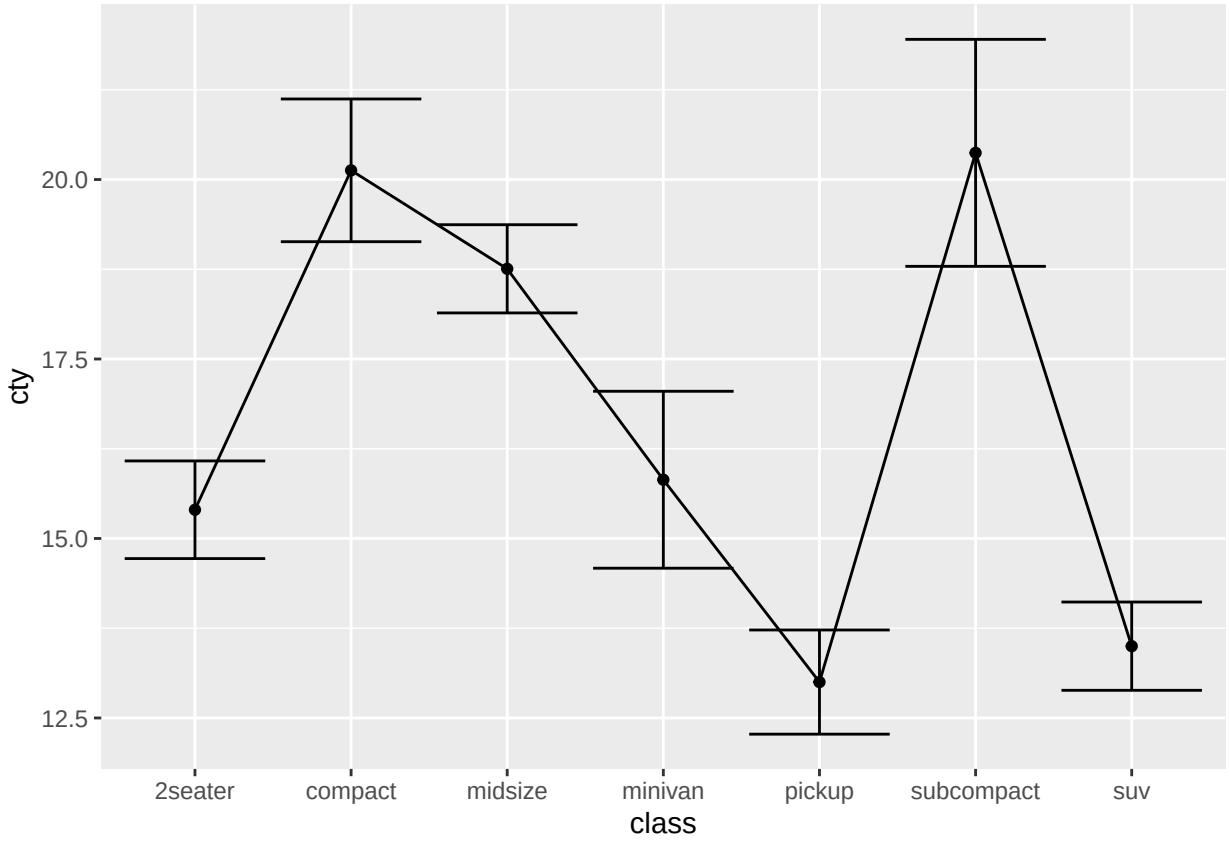



```
year_plot <- ggplot(mpg_data, aes(x = class, y = cty))
year_plot + stat_summary(fun.y = mean, geom = "point") +
  stat_summary(fun.y = mean, geom = "line", aes(group = 1)) +
  stat_summary(fun.data = mean_cl_normal, geom = "errorbar")
```

**Unteraufgabe 3**

```
## Warning: `fun.y` is deprecated. Use `fun` instead.

## Warning: `fun.y` is deprecated. Use `fun` instead.
```

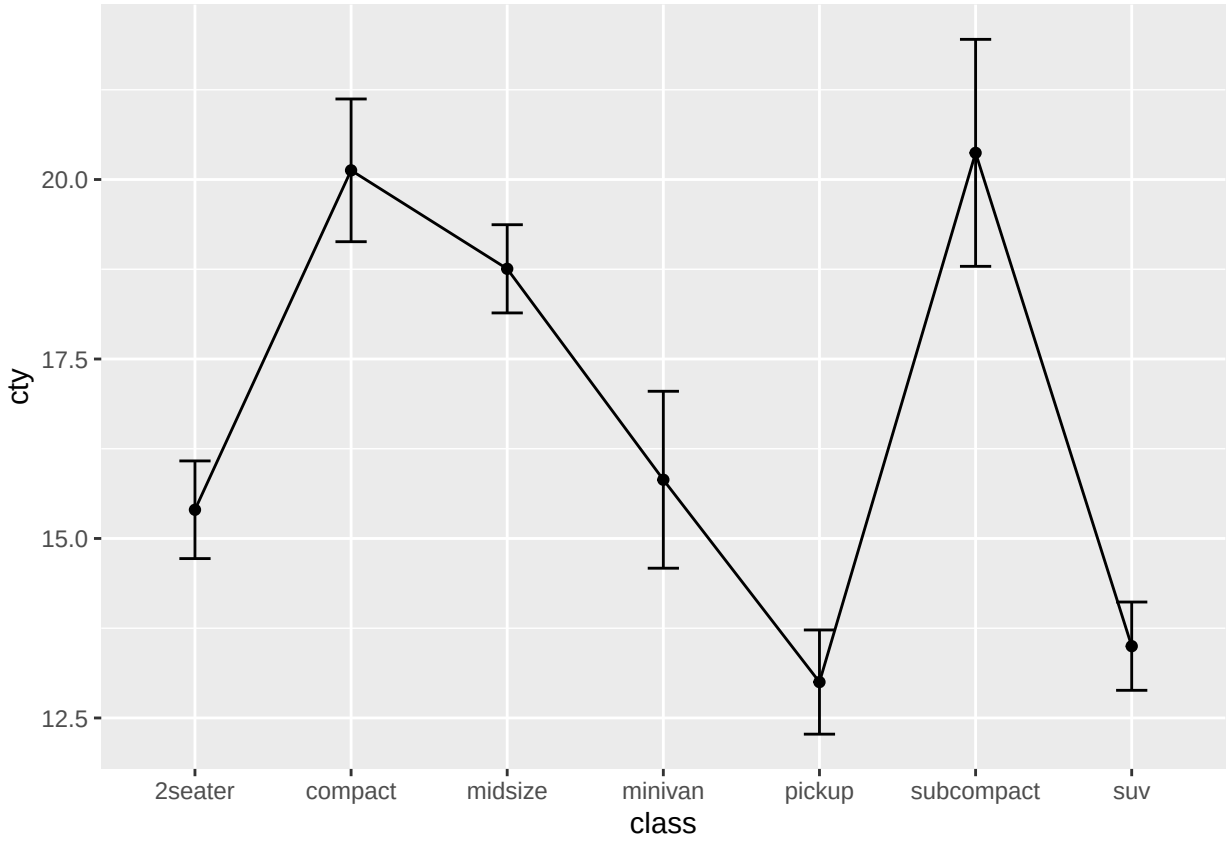



```
year_plot <- ggplot(mpg_data, aes(x = class, y = cty))
year_plot + stat_summary(fun.y = mean, geom = "point") +
  stat_summary(fun.y = mean, geom = "line", aes(group = 1)) +
  stat_summary(fun.data = mean_cl_normal, geom = "errorbar", width = 0.2)
```

**Unteraufgabe 4**

```
## Warning: `fun.y` is deprecated. Use `fun` instead.

## Warning: `fun.y` is deprecated. Use `fun` instead.
```

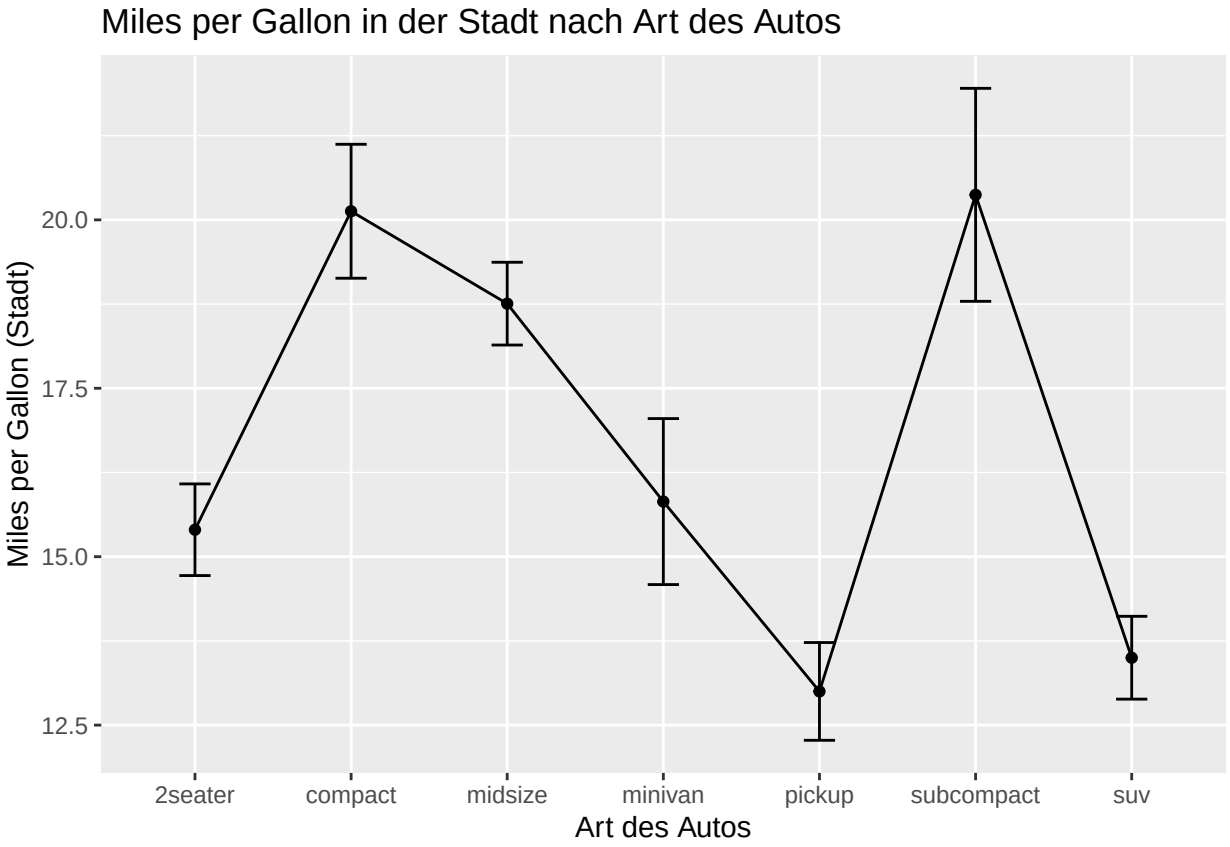



```
year_plot <- ggplot(mpg_data, aes(x = class, y = cty))
year_plot + stat_summary(fun.y = mean, geom = "point") +
  stat_summary(fun.y = mean, geom = "line", aes(group = 1)) +
  stat_summary(fun.data = mean_cl_normal, geom = "errorbar", width = 0.2) +
  labs(x = "Art des Autos",
       y = "Miles per Gallon (Stadt)",
       title = "Miles per Gallon in der Stadt nach Art des Autos")
```

**Unteraufgabe 5**

```
## Warning: `fun.y` is deprecated. Use `fun` instead.

## Warning: `fun.y` is deprecated. Use `fun` instead.
```




## Histogramm

1. Nutzen Sie `geom_histogram()`, um sich ein Histogramm der *Miles per Gallon* (Stadt) anzeigen zu lassen.
2. Auf Seite 2 des ggplot2-Cheatsheets (unten, rechts mittig) finden Sie eine Übersicht über verschiedene *Themes*, mit denen Sie Ihre Plots verschönern können. Geben Sie Ihrem Histogramm ein *Theme*, das Ihnen gefällt.

### Lösung

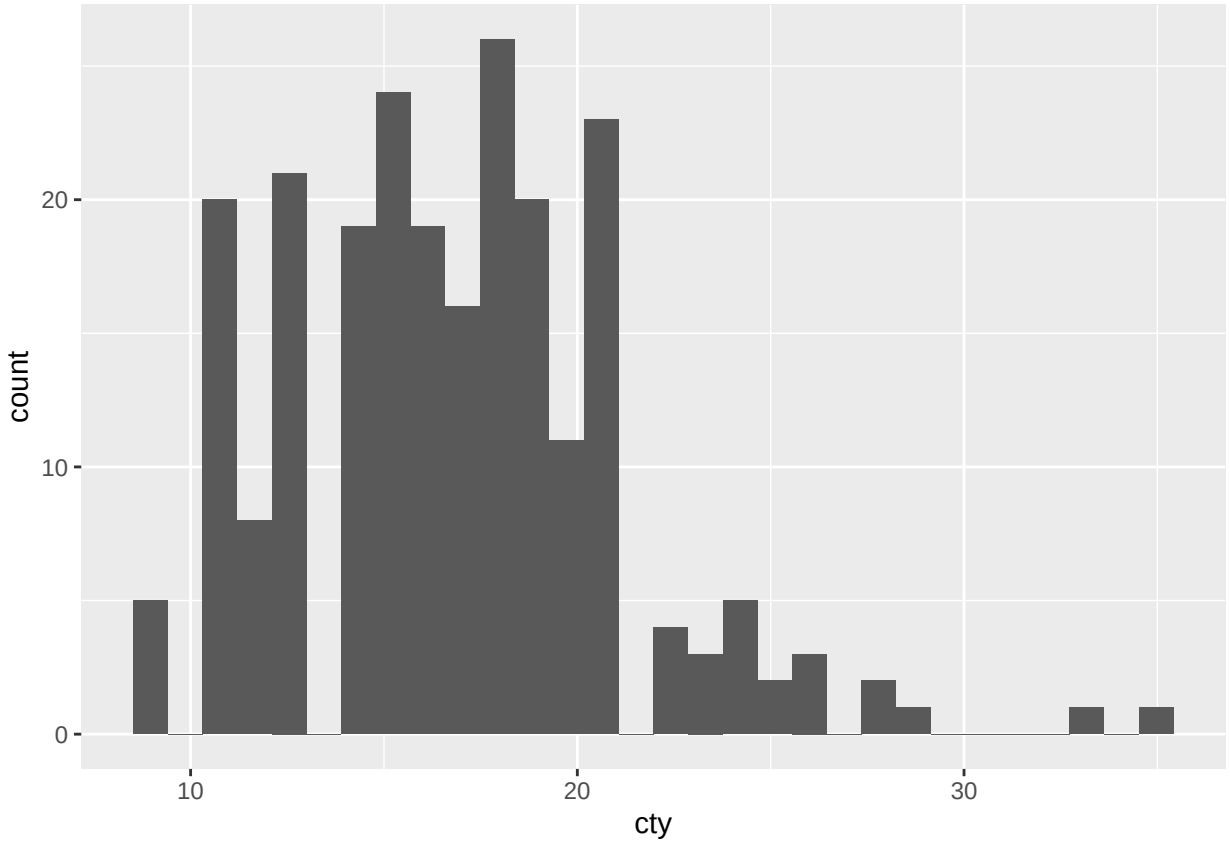
**Unteraufgabe 1** Objekt erstellen. Bei einem Histogramm brauchen wir immer nur eine Variable.

```
cty_hist <- ggplot(mpg_data, aes(cty))
```

Histogramm hinzufügen

```
cty_hist + geom_histogram()
```

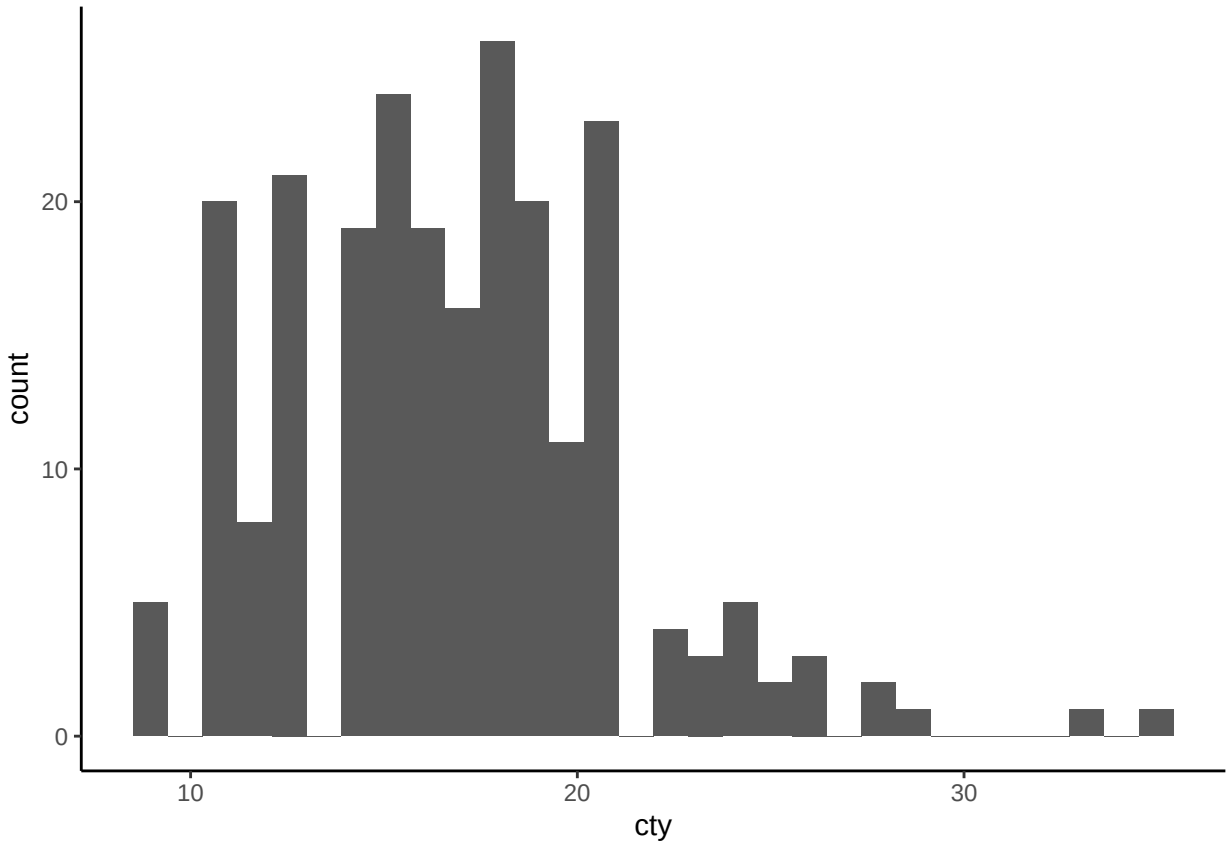
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
cty_hist + geom_histogram() + theme_classic()
```

**Unteraufgabe 2**

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

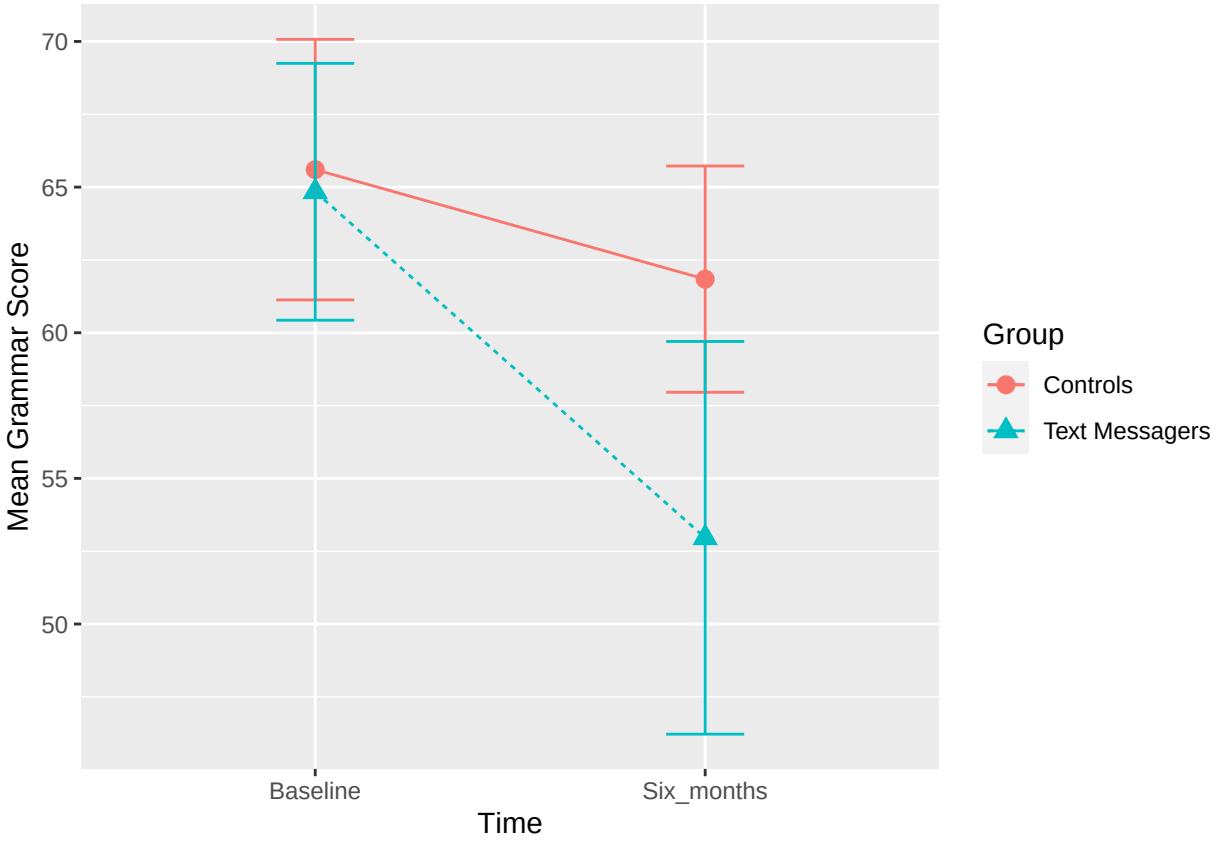



## Plot nachbauen

Bauen Sie mit dem Datensatz `text_messages.dat` den unten angezeigten Plot nach. Sie müssen ihn dafür zunächst einlesen und dann vom *wide* ins *long* Format bringen. Der Datensatz enthält die Testergebnisse in einem Grammatik-Test zu einem Baseline-Zeitpunkt und einem 6-Monats-Follow up. Es gab zwei Gruppen, die in der Variable `Groups` definiert sind.

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```




Tipps:

1. Der Datensatz enthält Spalten, die mit *Tabstopps* getrennt sind. Diese werden in R durch `"\t"` kodiert.
2. Sie müssen den Datensatz nicht nur einlesen, sondern auch vom *wide*- ins *long* Format bringen. Denken Sie dazu an den Befehl `gather()`.
3. Denken Sie daran, bei den Ebenen passende Gruppierungen anzugeben. Sie können Variablen zum Gruppieren benutzen.
4. Werfen Sie einen Blick auf das ggplot2-Cheatsheet zu den Befehlen `geom_point()` und `geom_line()`. Deren Argumente können Sie auch in `stat_summary()` verwenden, wenn Sie dort `geom = "point"` oder `geom = "line"` spezifiziert haben.

**Lösung**

Zunächst lesen wir den Datensatz ein.

```
text_data <- read_delim("data/text_messages.dat", delim = "\t")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Group = col_character(),
##   Baseline = col_double(),
##   Six_months = col_double()
## )
```

Anschließend bringen wir die Daten ins Long-Format. Im `gather()`Befehl nutze ich hier `2:3`, um der Funktion mitzuteilen, dass die Spalten 2 bis 3 umgeformt werden sollen. An der Stelle könne man auch die Namen verwenden, indem man `Baseline, Six_months` schreibt (heißt für R: `Baseline` **und** `Six_months`). Die Namen können auch mit dem Doppelpunkt verwendet werden: `Baseline:Six_months` heißt für R: `Baseline` **bis** `Six_months`.

```
text_data <- text_data %>%
  gather(key = "time", value = "grammar_score", 2:3)
```

Hier wird das Plot-Objekt erstellt. Beachten Sie, dass hier bereits die Farbe anhand der Gruppen definiert wird.

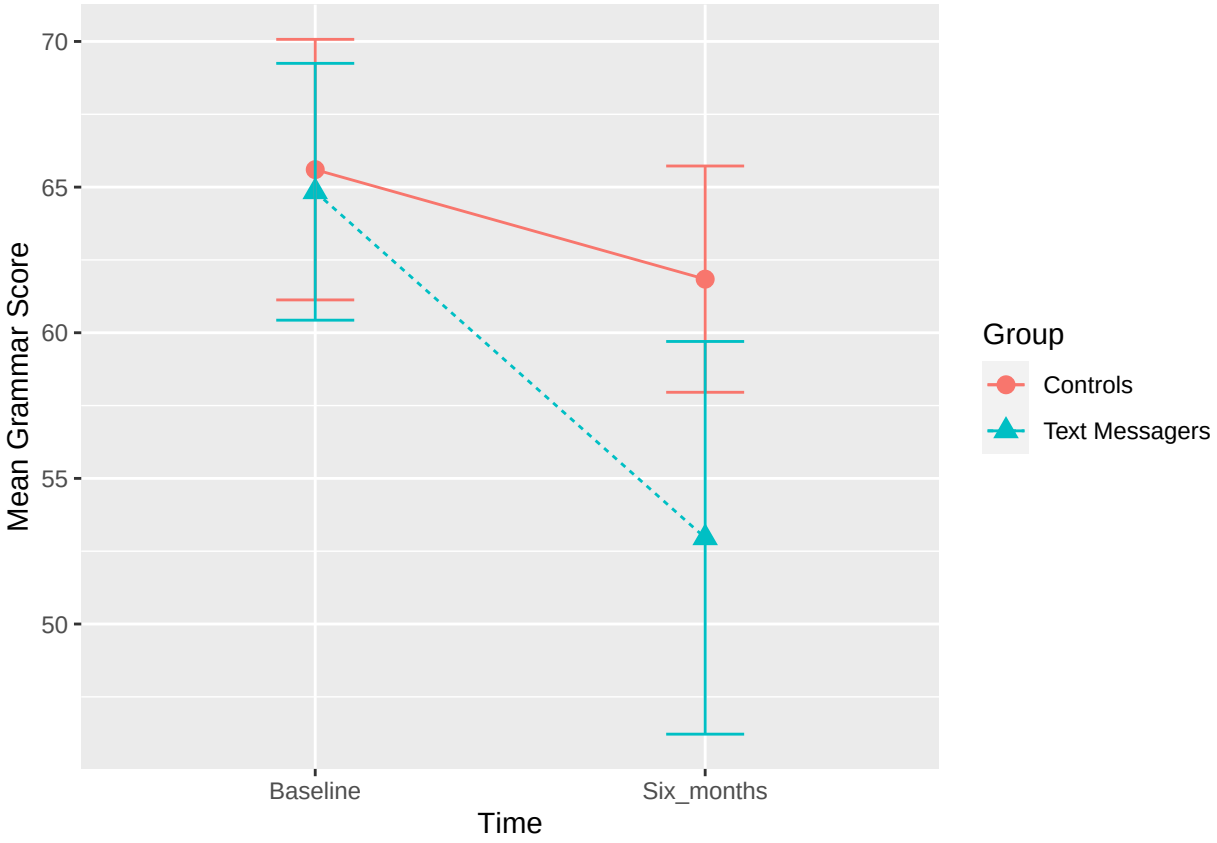
```
text_plot = ggplot(text_data, aes(time, grammar_score, colour = Group))
```

Hier werden dem Plot die Ebenen hinzugefügt.

```
text_plot +
  stat_summary(
    fun.y = mean,           # Mittelwerte anzeigen
    geom = "point",         # Als Punkte anzeigen
    aes(shape = Group),     # Punktform nach Gruppe bestimmen
    size = 3                # Punktgröße ändern
    ) +
  stat_summary(
    fun.y = mean,           # Mittelwerte anzeigen
    geom = "line",          # Als Linien anzeigen
    aes(group = Group,      # Gruppierung der Linien nach Gruppe
        linetype = Group)   # Linienart nach Gruppe
  ) +
  stat_summary(
    fun.data = mean_cl_normal, # Konfidenzintervalle berechnen
    geom = "errorbar",         # Als Fehlerbalken anzeigen
    width = 0.2                # Breite der Fehlerbalken auf 20%
    ) +
  labs(x = "Time",
       y = "Mean Grammar Score",
       colour = "Group")
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.

## Warning: `fun.y` is deprecated. Use `fun` instead.
```




**Als Barplot**

Versuchen Sie nun die gleiche Information, inklusive Fehlerbalken, als Barplot darzustellen.

Tipps:

1. Bei Barplots gibt `fill` die Farbe der Säule (die Füllung) an, und `color` die Farbe des Rahmens.
2. Das Argument `position = "dodge"` kann helfen, wenn die Säulen sich überlappen.
3. Das Argument `position = position_dodge(width = .90)` kann helfen, schmale Ebenen an breiteren Ebenen auszurichten.

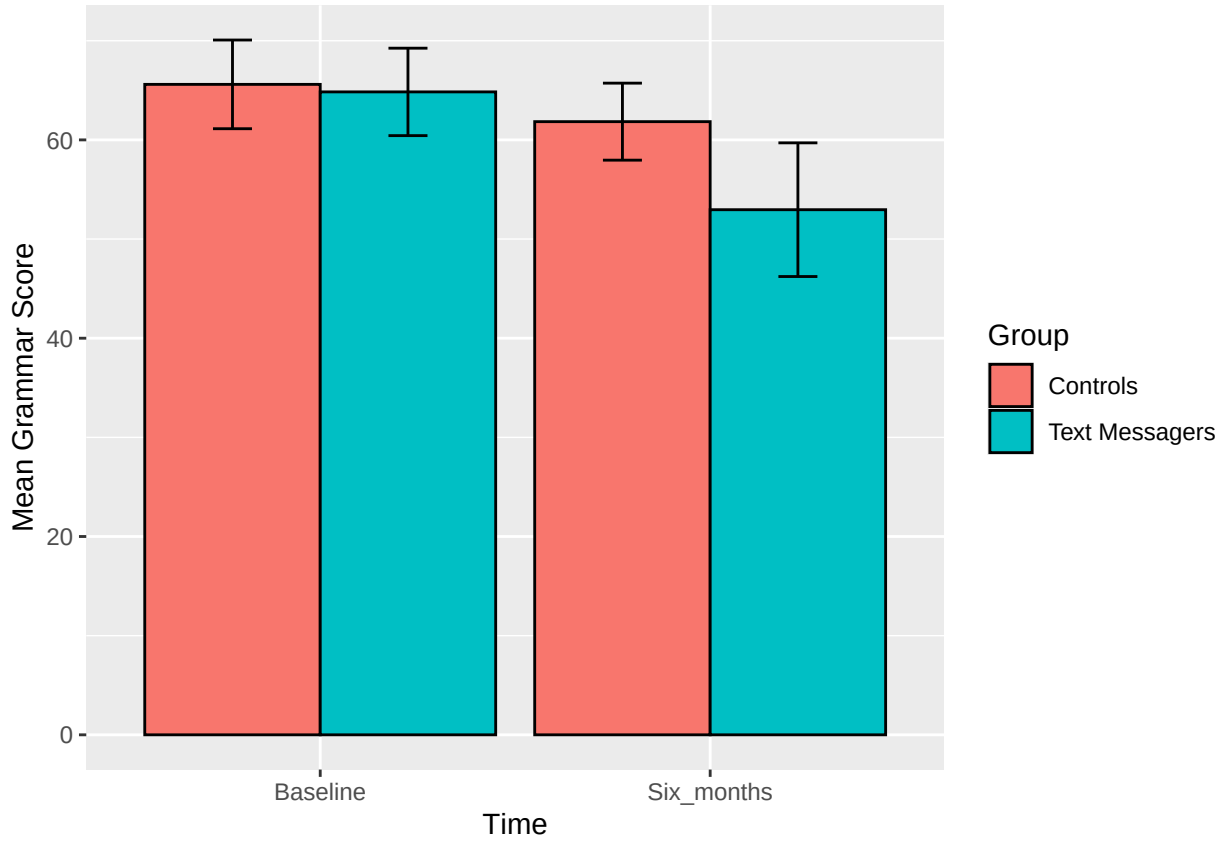
**Lösung**

```
text_plot = ggplot(text_data, aes(time, grammar_score, fill = Group))
```

```
text_plot +
  stat_summary(
    fun.y = mean,           # Mittelwerte anzeigen
    geom = "bar",           # Als Säulen anzeigen
    position = "dodge",     # Säulen nebeneinander platzieren
    color = "black"         # Schwarzer Rahmen
    ) +
```

```
  stat_summary(
    fun.data = mean_cl_normal,    # Konfidenzintervalle berechnen
    geom = "errorbar",            # Als Fehlerbalken anzeigen
    width = 0.2,                  # Breite der Fehlerbalken auf 20%
    position = position_dodge(width = .90) # Fehlerbalken passend zu Säulen
    ) +
  labs(x = "Time",
       y = "Mean Grammar Score",
       colour = "Group")
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```




## Rendern

Rendern, bzw. knitten Sie nun das Dokument über die Tastenkombination `strg` + `shift` + `k` (Windows) oder `cmd` + `shift` + `k`. Wenn das funktioniert: Top gemacht! Wenn nicht: Schauen Sie sich die Fehlermeldung an, und betrachten Sie insbesondere die Zeilen Ihrer Syntax, die in der Fehlermeldung auftauchen. Suchen Sie nach dem Fehler und probieren Sie es erneut!

## Literatur

*Anmerkung*: Diese Übungszettel basieren zum Teil auf Aufgaben aus dem Lehrbuch *Dicovering Statistics Using R* (Field, Miles & Field, 2012). Sie wurden für den Zweck dieser Übung modifiziert, und der verwen-

dete R-Code wurde aktualisiert.

Field, A., Miles, J., & Field, Z. (2012). *Discovering Statistics Using R*. London: SAGE Publications Ltd.

# English Version

Exercise sheet in PDF format for printing sheet_plots.pdf

**Exercise sheet with solutions included**

Exercises with solutions included PDF

Soruce code of the omplete exercise sheet Rmd (to download: right click > save as …)

## Some hints

1. Please give your answers in a .Rmd file. You may generate one from scratch using the file menu: 'File > new file > R Markdown …' Delete the text below *Setup Chunk* (starting from line 11). Alternatively you may use this sample Rmd by donloading it.
2. You may need the informations from the start page of this course.
3. Don't hesitate to google for solutions. Effective web searches to find solutions for R-problems is a very useful ability, professionals to that too … A really good starting point might be the R area of the programmers platform Stackoverflow
4. You can find very useful cheat sheets for various R-related topics. A good starting point is the Base R Cheat Sheet.

## Ressources

This is a hands on course. We cannot present you all the useful commands in detail. Instead we give you links to useful ressources, where you might find hints to help you with the exercises.

| Ressource | Description |
|---|---|
| Field, Chapter 4 | Book chapter with a step for step introduction to `ggplot2`. **Recommendation!** |
| ggplot2 Cheat Sheet | Overview of most common ggplot2 commands |
| Peters ggplot2-Referenz | Peter offers a big collection of plots with source code to generate them. A resource to find running examples. |

## Tip of the week

We can insert the pipe symbol %>% using the shortcut `ctrl` + `shift` + `m` (Windows) oder `cmd` + `shift` + `m` (Mac).

## Read data

1. Define an appropriate working directory for this exercise sheet. This should usually be the folder, where your Rmd-file is located. Insert a command to do that in your Rmd-document.
2. Load the data `mpg.csv` and store it in your working directory. You might still have the folder you used for your last sheet - then store the data in a data subdirectory.
3. Open the file `mpg.csv` in a text editor and check, which character separates the content in the lines to form the future columns.
4. Assure, that the package `tidyverse` are loaded. Insert a code line for that in the beginning of your Rmd-file.
5. Use the command `read_delim()` and define a data object named `mpg_data` using the data in file `mpg.csv`. `read_delim()` is a generalized version of `read_csv()`, with which you define manually the separator used in your data file by setting the argument `delim = "<separator>"`. C. f. Tabulator is specified by "".

### Solutions

**Subtask 1** You should put your quoted data path

```
setwd("P:/R/mv") # replace this with the path you use
```

**Subtask 2 and 3** Please follow the recommendation of the exercise text.

```
library(tidyverse)
```

**Subtask 4**

**Subtask 5** Your quoted data path should be used. We assume here, that working directory is set to `P:/R/mv`. In this example we find the file `mpg.csv` in the subfolder `data` of our working directory.

```
mpg_data <- read_delim("data/mpg.csv", delim = "|")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   manufacturer = col_character(),
##   model = col_character(),
##   displ = col_double(),
##   year = col_double(),
##   cyl = col_double(),
##   trans = col_character(),
##   drv = col_character(),
##   cty = col_double(),
##   hwy = col_double(),
##   fl = col_character(),
##   class = col_character()
## )
```

## Plot basics

We use package `ggplot2` for creating plots. This package is part of `tidyverse` and we don't have to load it separately, when we already have loaded `tidyverse` (see task 1.4).

### Create plot objekt

Generating plots works always the same. We create a plot-object using the command `ggplot()`

```
example_plot <- ggplot(data, aes(x = variable_1, y = variable_2))
```

As we can see, the first argument is the data-object to use. With `aes()` we define, which variable assigned to x-axis and y-axis respectively.

### Add a layer

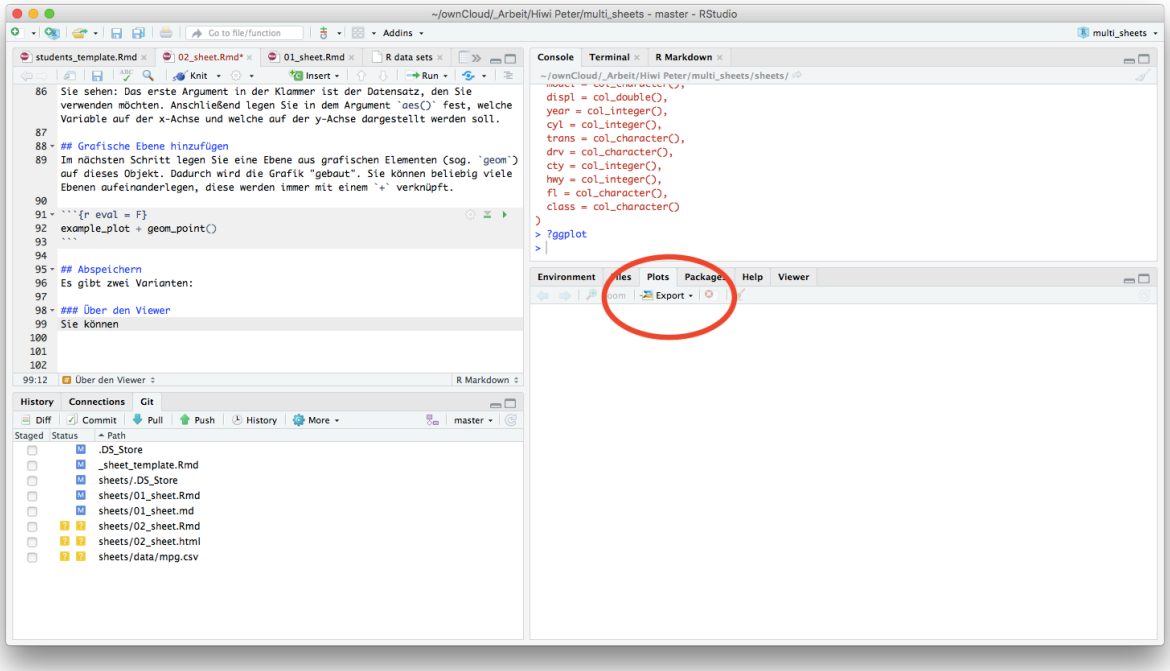
Next step is to add a layer of graphical elements, a so called `geom` to the object. This is, how plots are constructed. We may add as much layers, as we like. They are always connected using +.

```
example_plot + geom_point()
```

### Saving

There are two variants:

**Via menu** We can click on "Export" in the "Plots" tab in the lower right part of RStudio:



**Via syntax** To save a plot object with all its layers we have to first store the complete plot-object:

```
exmaple_plot <- example_plot + geom_point()
```

Then we use `ggsave()` to save the plot-object. It is stored in your working directory. Add a name for the plot-file in quotation marks (including extension) as first parameter. Next parameter is the plot-object to store.

```
ggsave("example_plot.png", example_plot)
```

## Scatterplot

1. Use the command `?mpg` to get a description of the data we use.
2. Create a ggplot-object named `displ_plot`, which uses `mpg_data` as dataset. X-axis should show *engine displacement* of the model. Y-axis should visualize the *city miles per gallon*.
3. Use `geom_point()` to add a layer to the above object that shows points at the coordinates of the two variables.
4. Use `geom_smooth()` with the arguments `method = "lm"` and `se = FALSE` to add an additional layer with a regression line.
5. Try what happens if you don't add the above two arguments you set in your `geom_smooth()` command.

### Solutions

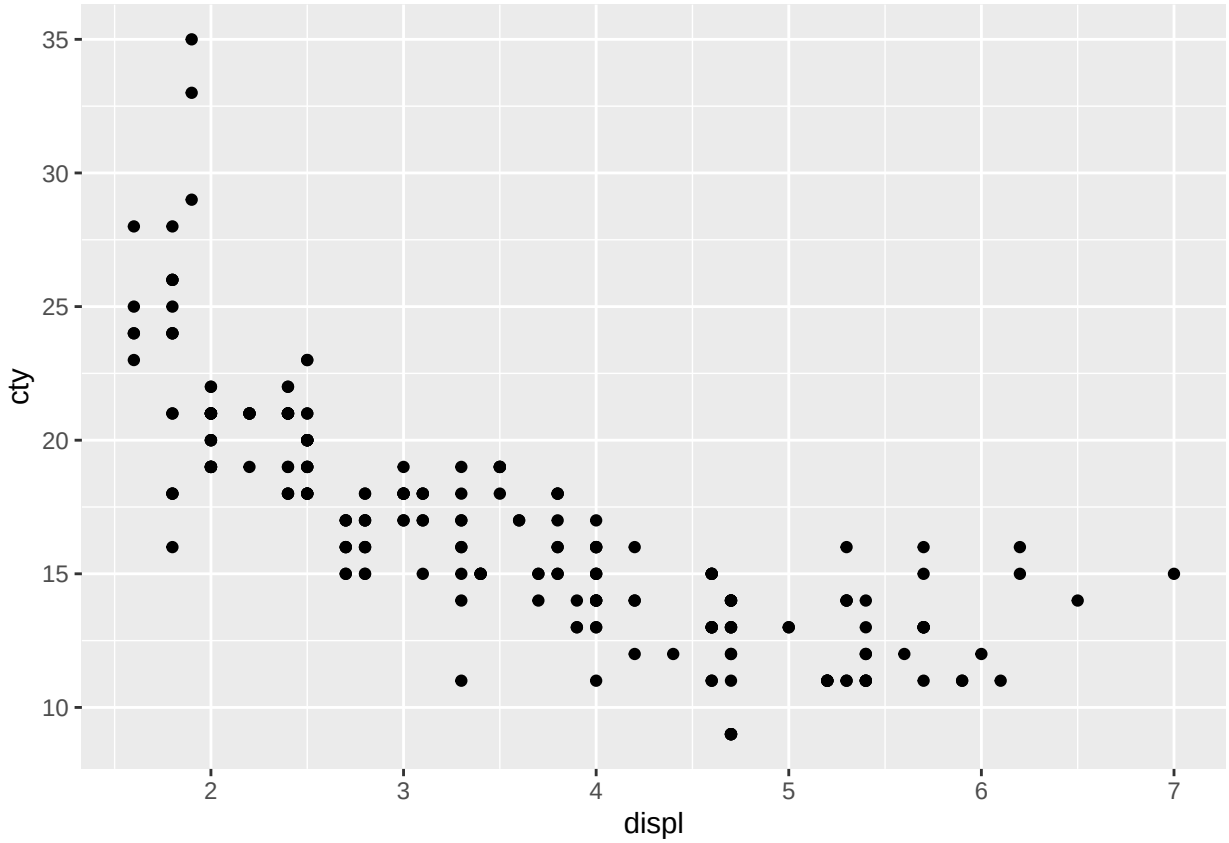
```
?mpg
```

### Subtask 1

```
displ_plot <- ggplot(mpg_data, aes(x = displ, y = cty))
```

### Subtask 2

```
displ_plot + geom_point()
```

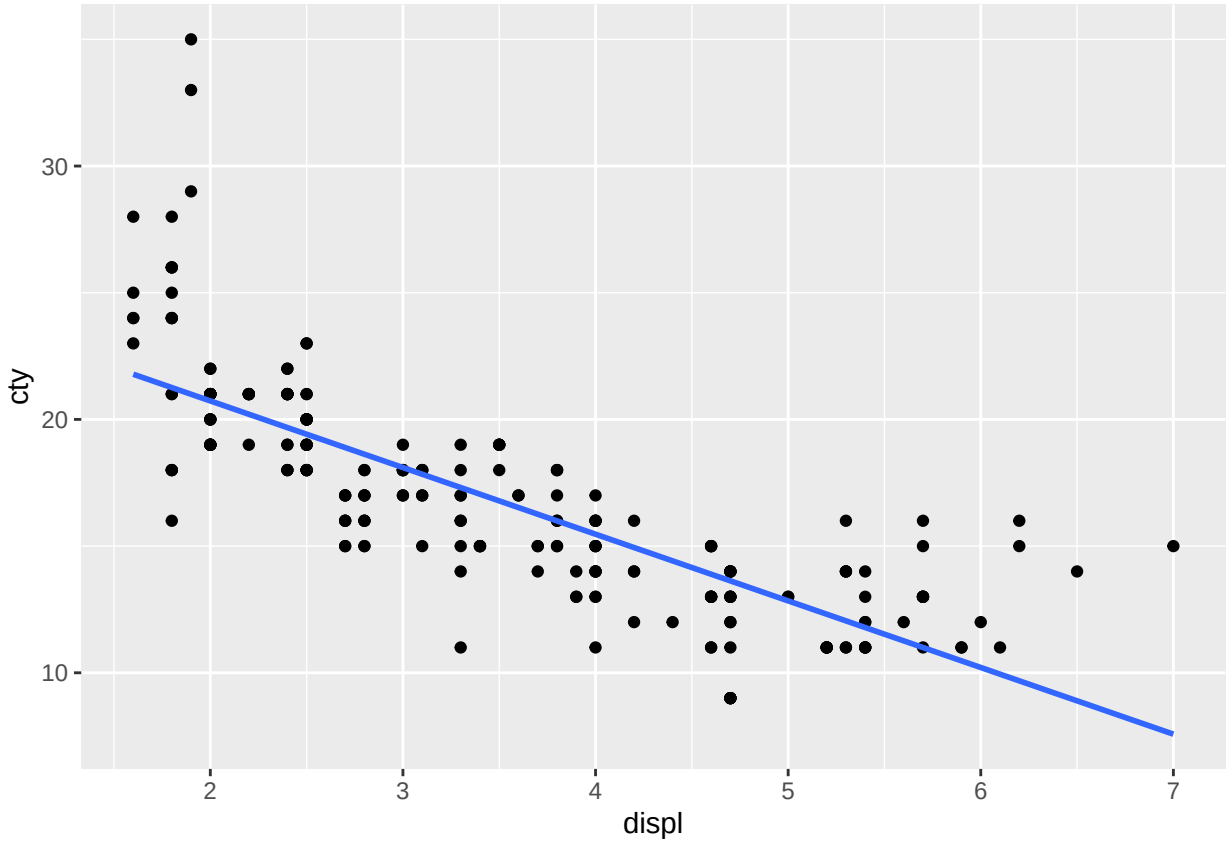



**Subtask 3**

```
displ_plot + geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

**Subtask 4**

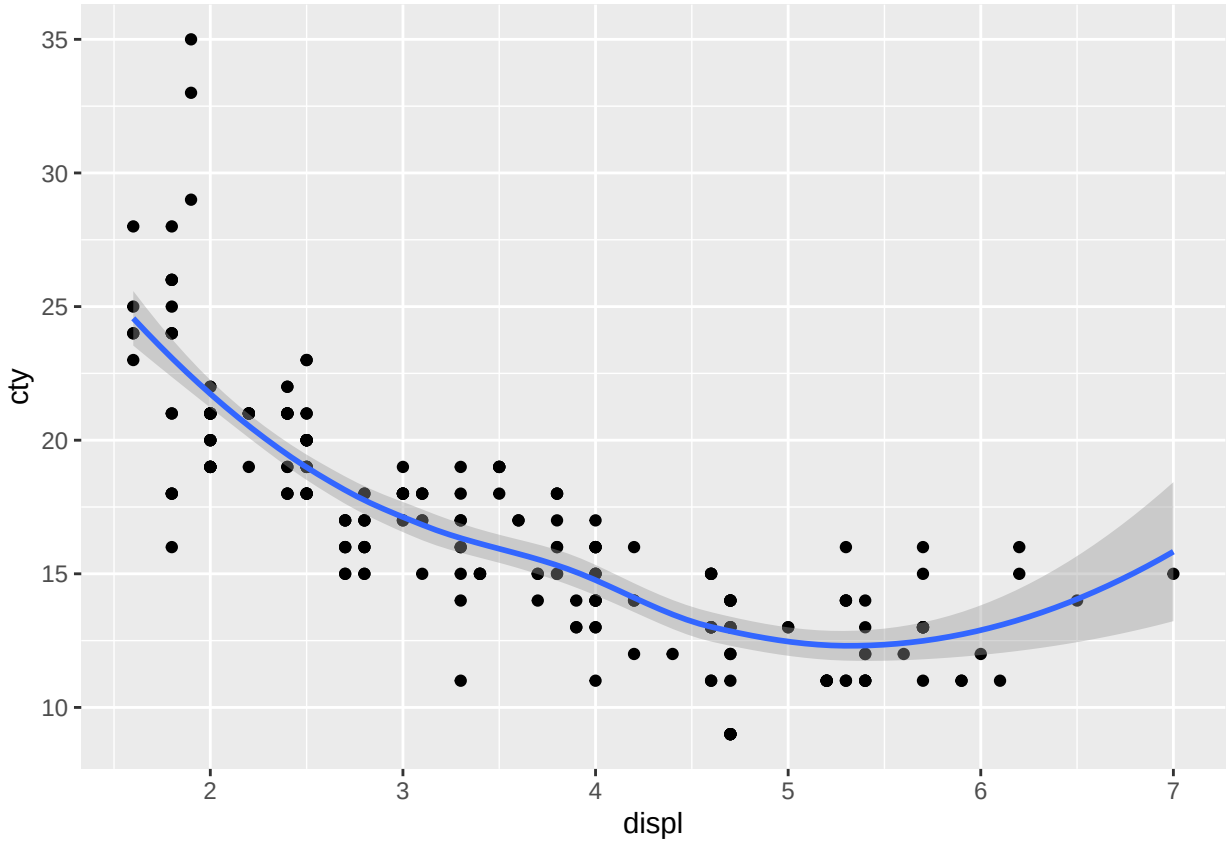
```
## `geom_smooth()` using formula 'y ~ x'
```




```
displ_plot + geom_point() +
  geom_smooth()
```

**Subtask 5**

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




## Barplot

1. Crate a new plot-object named `drv_plot` with the following specifications: The dataset should be `mpg_data` again. X-axis should be type of drive *drv* (front, rear or 4wd). Y-axis should be consumption *Miles per Gallon*.
2. Add a *summary*-layer using `stat_summary()`. In `stat_summary()` use argument `fun.y = mean` to tell `ggplot` that you want to show the mean of the observed data on the y-axis. Use also the argument `geom = "bar"` to get a barplot.
3. Add argument `fill = drv` while defining object `drv_plot` to colorize the barplot. Rerun the command of the subtask above to update the plot.
4. Add a label-layer to the plot using the command `labs()`. In `labs()` set arguments `x = "text"` and `y = "text"` and `fill = "text"` to give meaningful informations. Add also a title and an information of the meaning of the colors used.
5. Add a black margin to the bars using the argument `color = "black"` in the command `stat_summary()`.

#### Solution

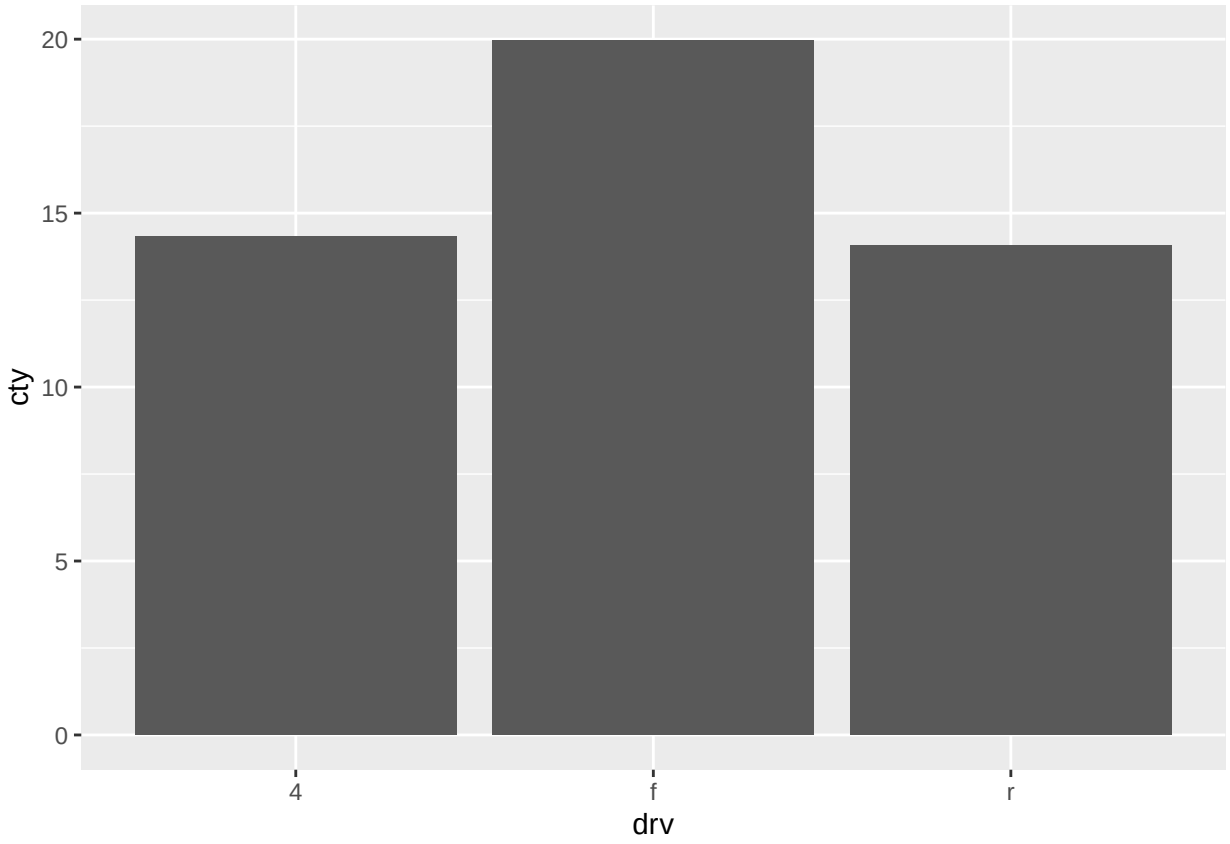
```
drv_plot <- ggplot(mpg_data, aes(x = drv, y = cty))
```

#### Subtask 1

```
drv_plot + stat_summary(fun.y = mean, geom = "bar")
```

### Subtask 2

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```

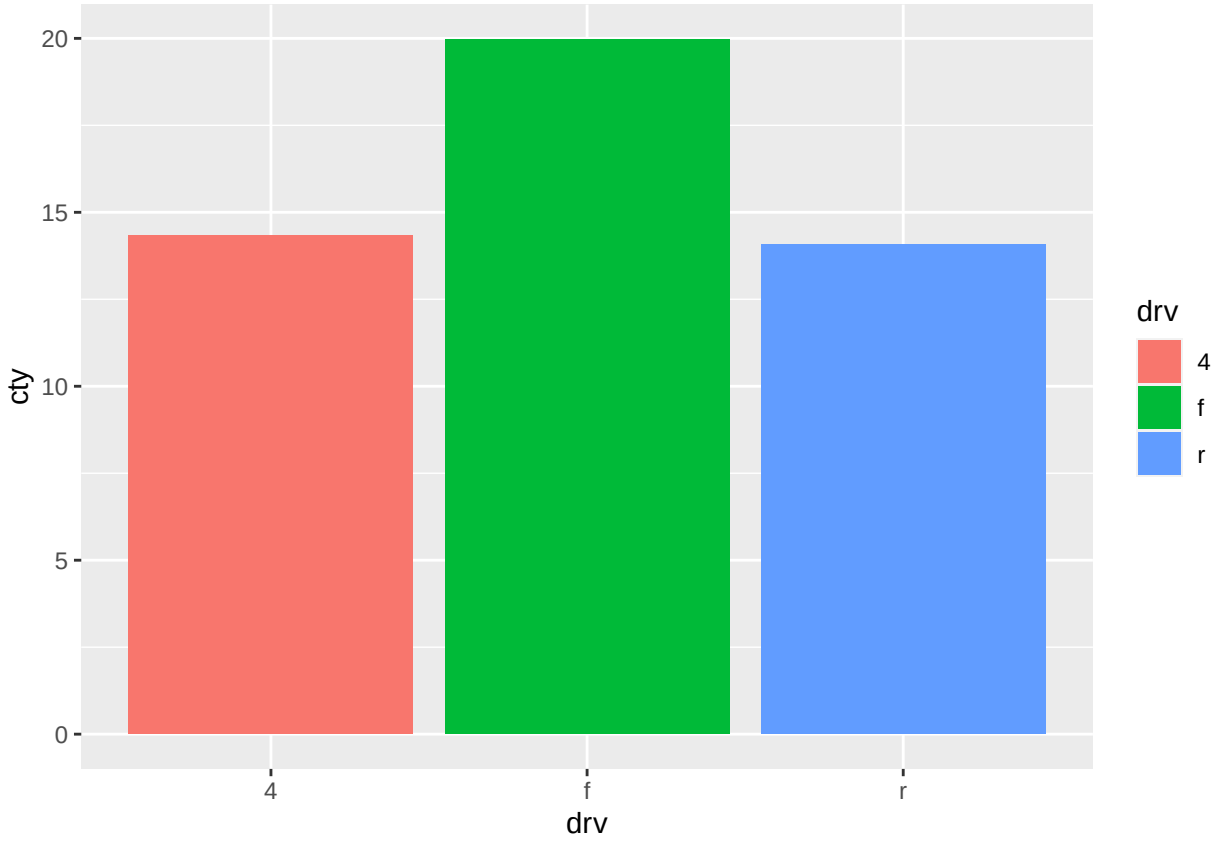



```
drv_plot <- ggplot(mpg_data, aes(x = drv, y = cty, fill = drv))

drv_plot + stat_summary(fun.y = mean, geom = "bar")
```

### Subtask 3

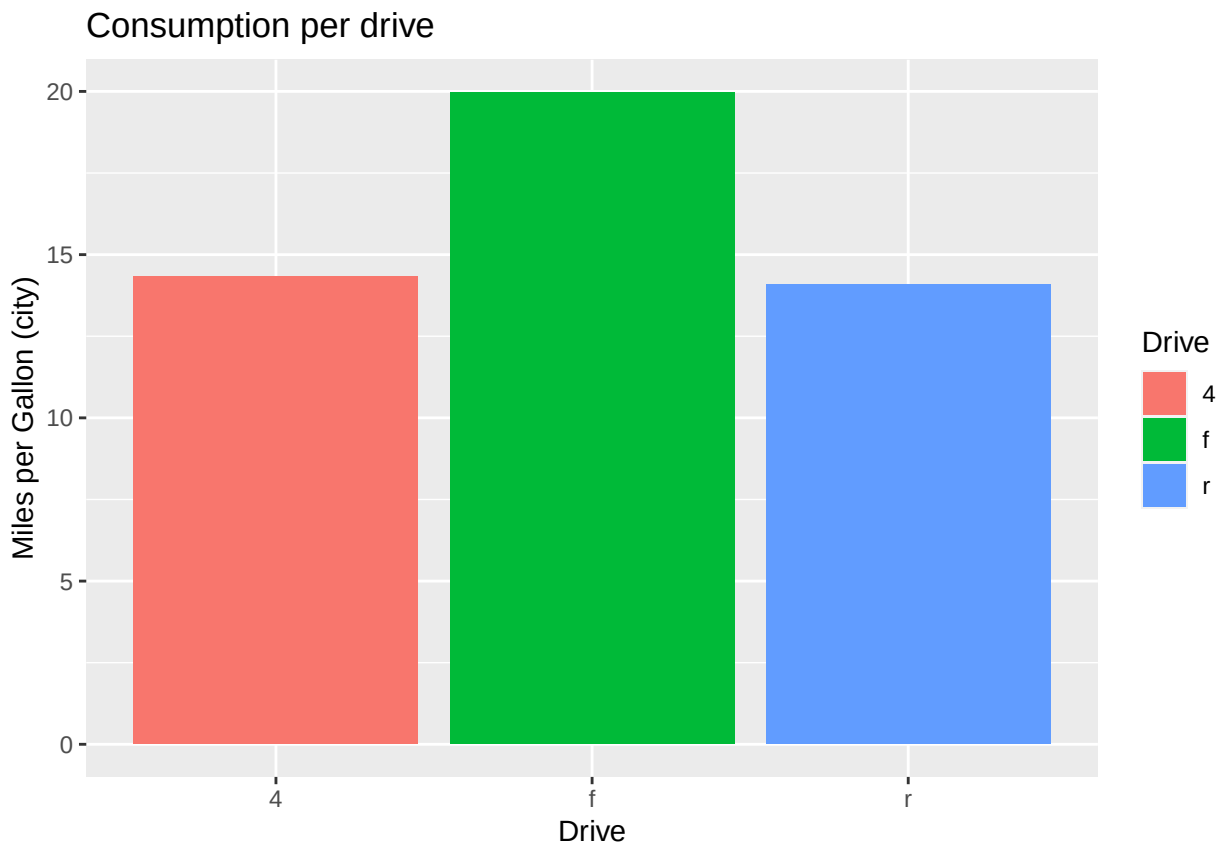
```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```




```
drv_plot + stat_summary(fun.y = mean, geom = "bar") +
  labs(x = "Drive",
       y = "Miles per Gallon (city)",
       title = "Consumption per drive",
       fill = "Drive")
```

**Subtask 4**

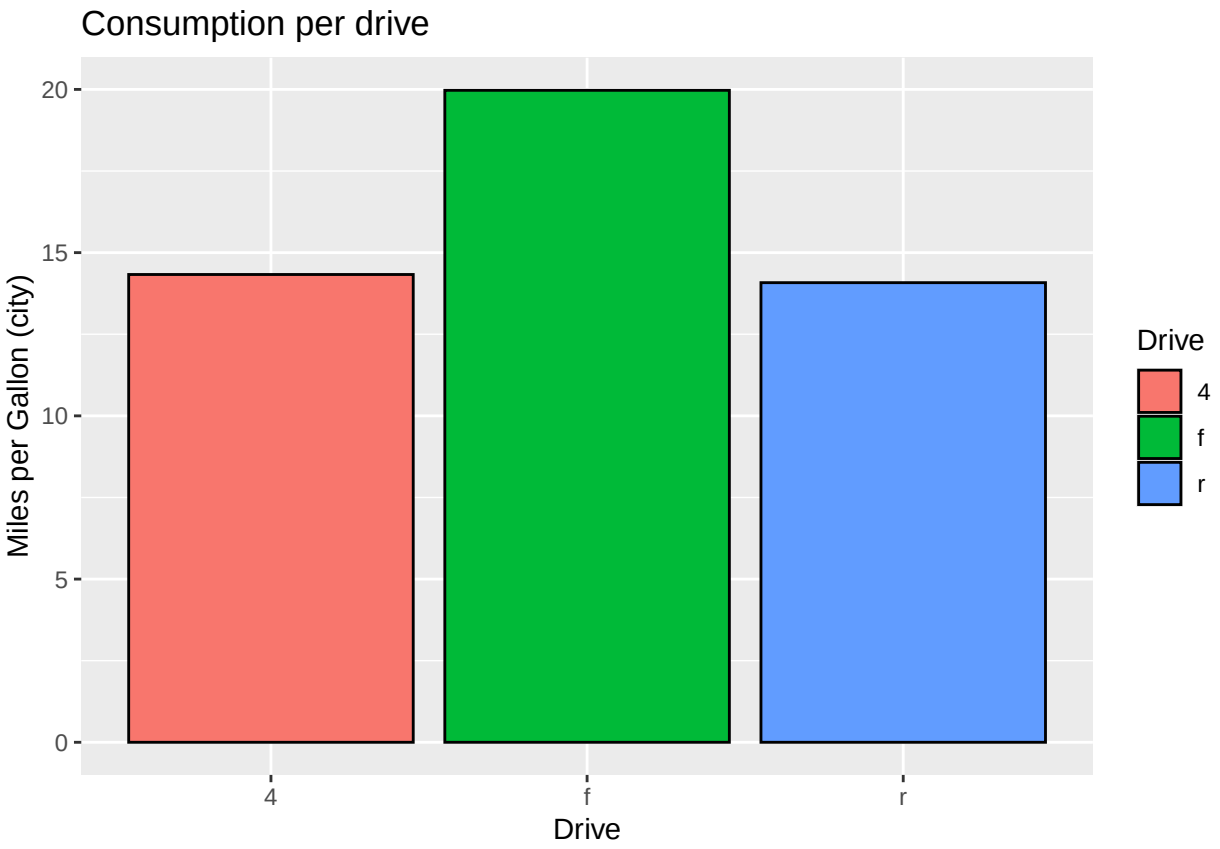
```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```




```
drv_plot + stat_summary(fun.y = mean, geom = "bar", color = "black") +
  labs(x = "Drive",
       y = "Miles per Gallon (city)",
       title = "Consumption per drive",
       fill = "Drive")
```

**Subtask 5**

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```

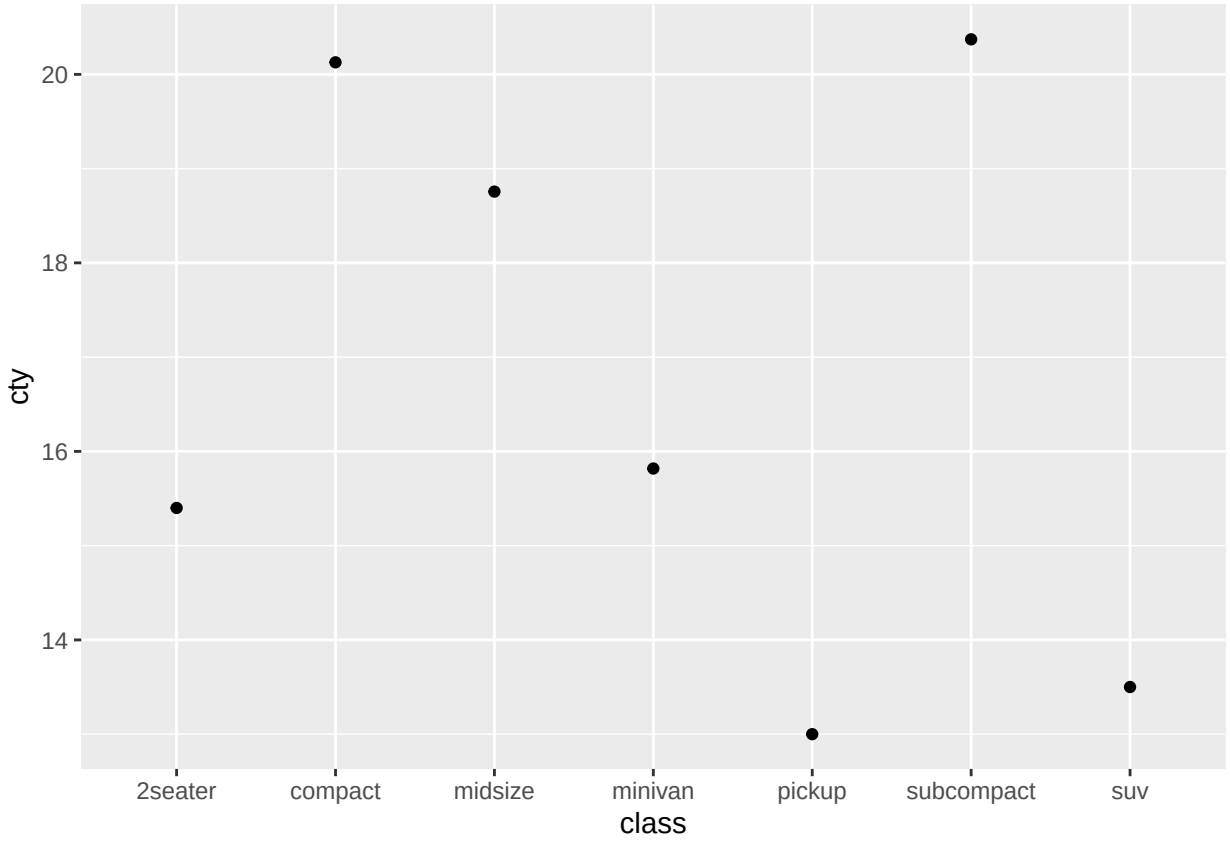



## Lineplot

1. Create the following plot.

Hint: points show mean values.

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```



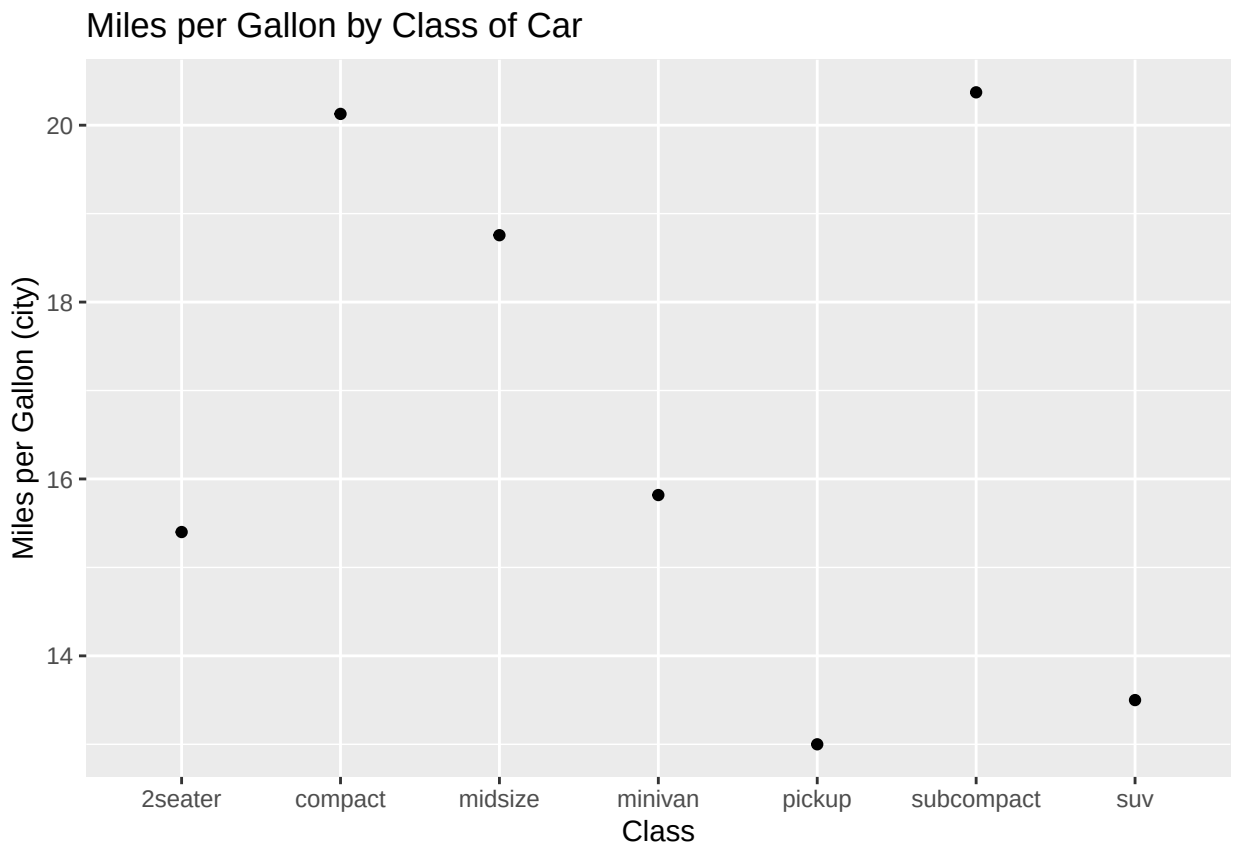

2. Use the command `stat_summary()` to add a layer that connects the points with a line. Tip: use argument `aes(group = 1)` in `stat_summary()` to tell `ggplot` that all points should be grouped in one single group.
3. Use the command `stat_summary()` with the arguments `fun.data = mean_cl_normal` and `geom = "errorbar"` to add a layer with errorbars, that show the 95% confidence interval around the means.
4. Use the argument `width = 0.2` in `stat_summary()` to configure the width of the whiskers of the confidence interval indicator.
5. Add meaningful labels for the axes and an appropriate title.

**Solutions**

```
year_plot <- ggplot(mpg_data, aes(x = class, y = cty))
year_plot + stat_summary(fun.y = mean, geom = "point") +
  labs(x = "Class",
       y = "Miles per Gallon (city)",
       title = "Miles per Gallon by Class of Car")
```

**Subtask 1**

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```

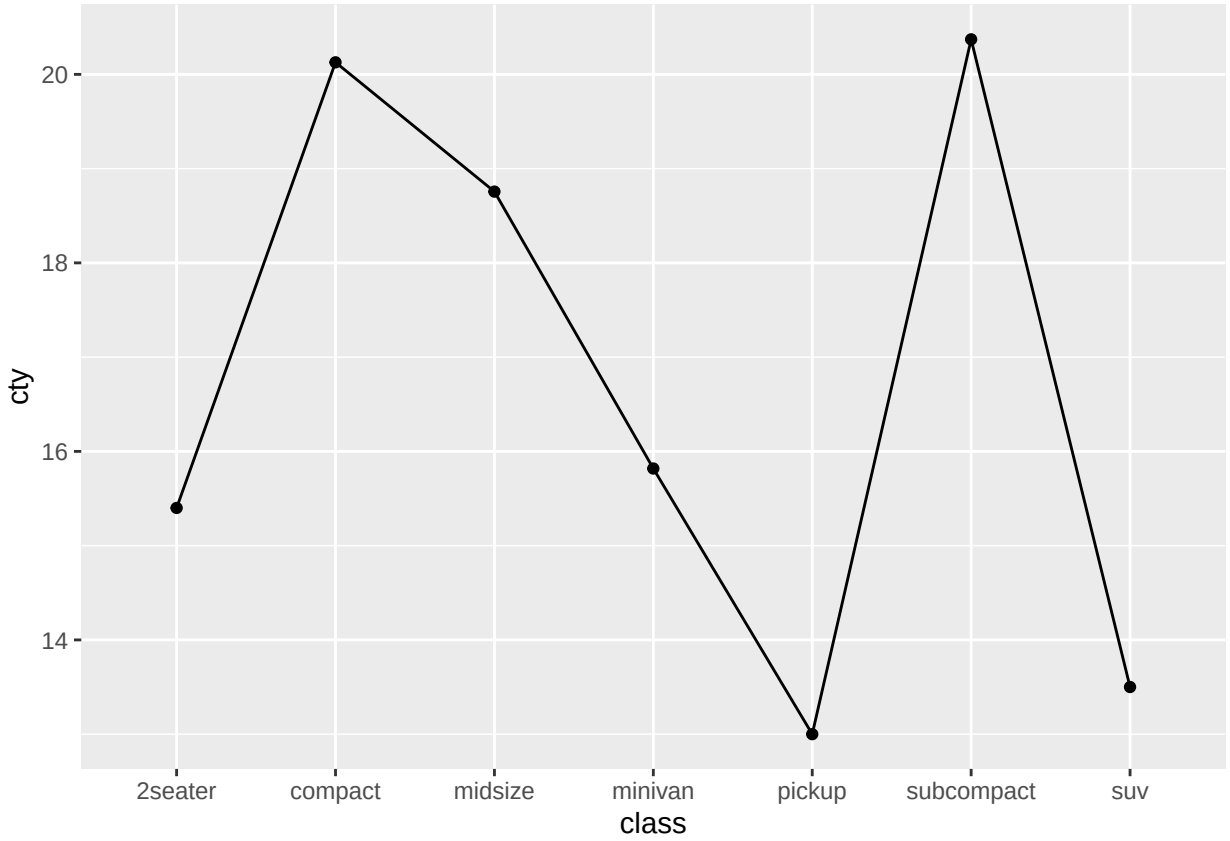



```
year_plot <- ggplot(mpg_data, aes(x = class, y = cty))
year_plot + stat_summary(fun.y = mean, geom = "point") +
  stat_summary(fun.y = mean, geom = "line", aes(group = 1))
```

**Subtask 2**

```
## Warning: `fun.y` is deprecated. Use `fun` instead.

## Warning: `fun.y` is deprecated. Use `fun` instead.
```

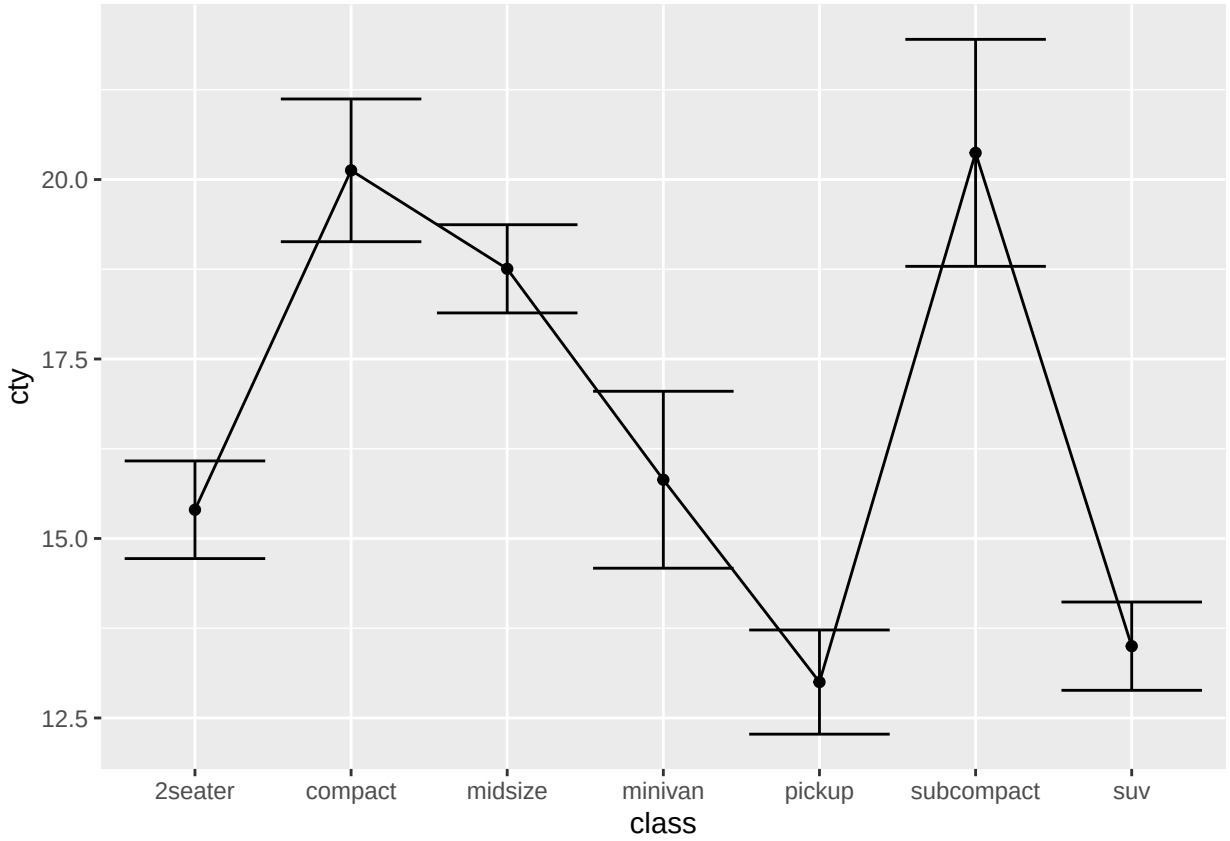



```
year_plot <- ggplot(mpg_data, aes(x = class, y = cty))
year_plot + stat_summary(fun.y = mean, geom = "point") +
  stat_summary(fun.y = mean, geom = "line", aes(group = 1)) +
  stat_summary(fun.data = mean_cl_normal, geom = "errorbar")
```

**Subtask 3**

```
## Warning: `fun.y` is deprecated. Use `fun` instead.

## Warning: `fun.y` is deprecated. Use `fun` instead.
```

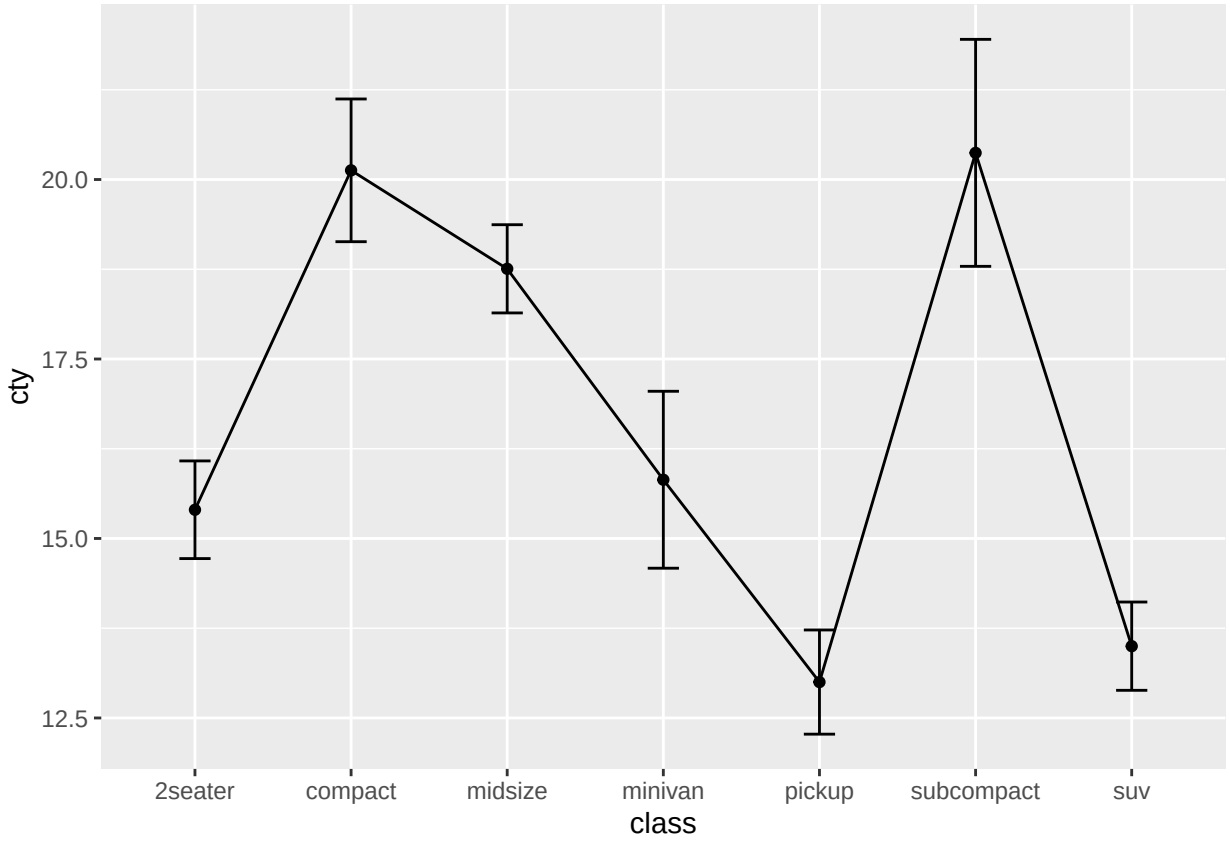



```
year_plot <- ggplot(mpg_data, aes(x = class, y = cty))
year_plot + stat_summary(fun.y = mean, geom = "point") +
  stat_summary(fun.y = mean, geom = "line", aes(group = 1)) +
  stat_summary(fun.data = mean_cl_normal, geom = "errorbar", width = 0.2)
```

**Subtask 4**

```
## Warning: `fun.y` is deprecated. Use `fun` instead.

## Warning: `fun.y` is deprecated. Use `fun` instead.
```

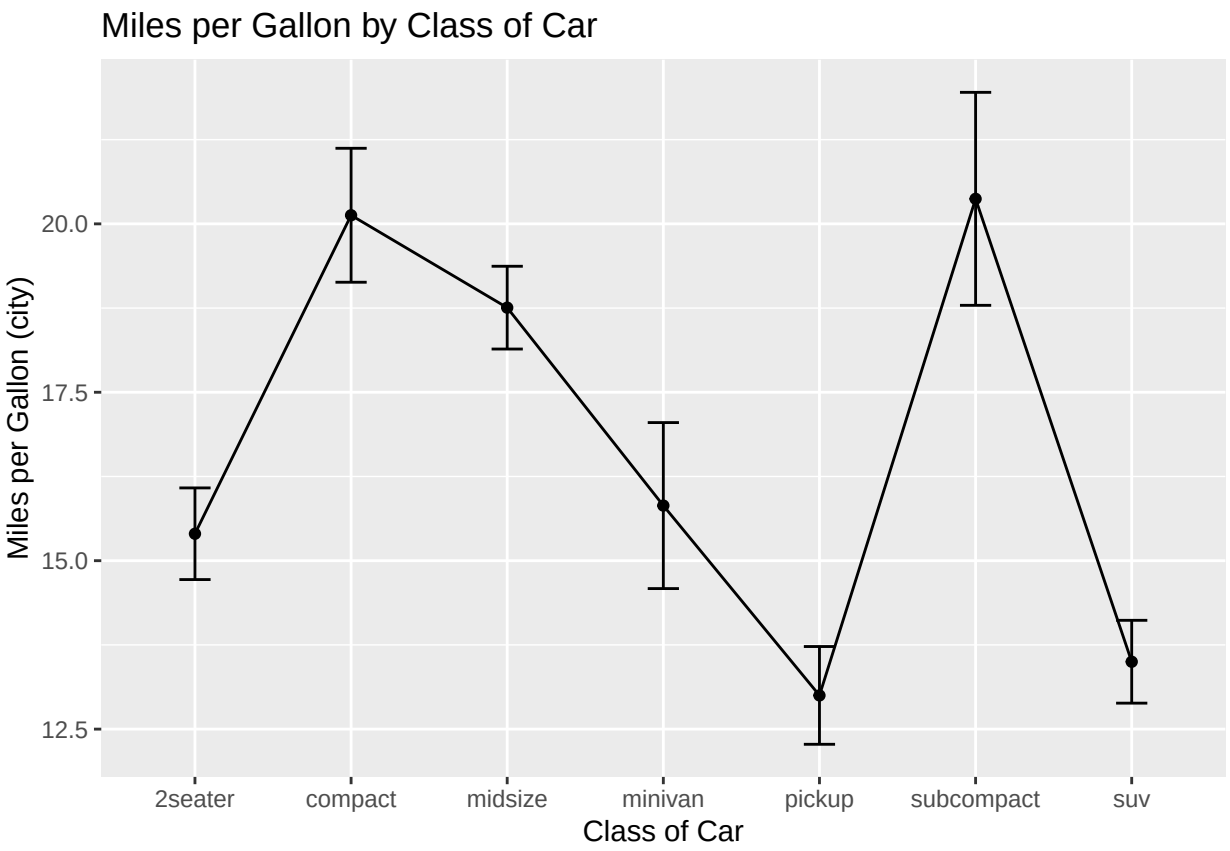



```
year_plot <- ggplot(mpg_data, aes(x = class, y = cty))
year_plot + stat_summary(fun.y = mean, geom = "point") +
  stat_summary(fun.y = mean, geom = "line", aes(group = 1)) +
  stat_summary(fun.data = mean_cl_normal, geom = "errorbar", width = 0.2) +
  labs(x = "Class of Car",
       y = "Miles per Gallon (city)",
       title = "Miles per Gallon by Class of Car")
```

**Subtask 5**

```
## Warning: `fun.y` is deprecated. Use `fun` instead.

## Warning: `fun.y` is deprecated. Use `fun` instead.
```




## Histogram

1. use `geom_histogram()` to get a histogram of *Miles per Gallon* (city).
2. On page 2 of the ggplot2-Cheatsheets (below, mid right) you find an overview of various *Themes* that are useful to make your plots more beautiful. Apply a *Theme* you like to your histogram.

### Solution

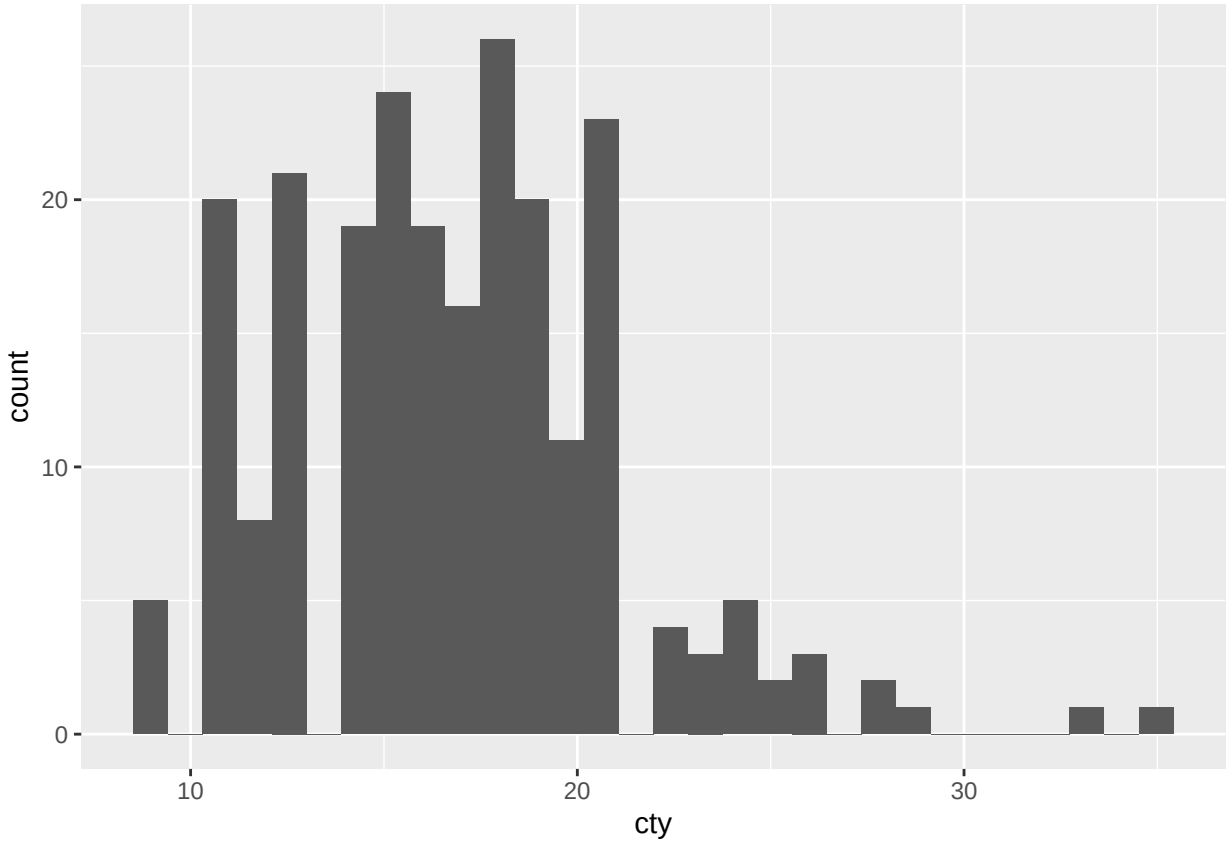
**Subtask 1** Create object. With histograms we use one variable only.

```
cty_hist <- ggplot(mpg_data, aes(cty))
```

We add a histogram.

```
cty_hist + geom_histogram()
```

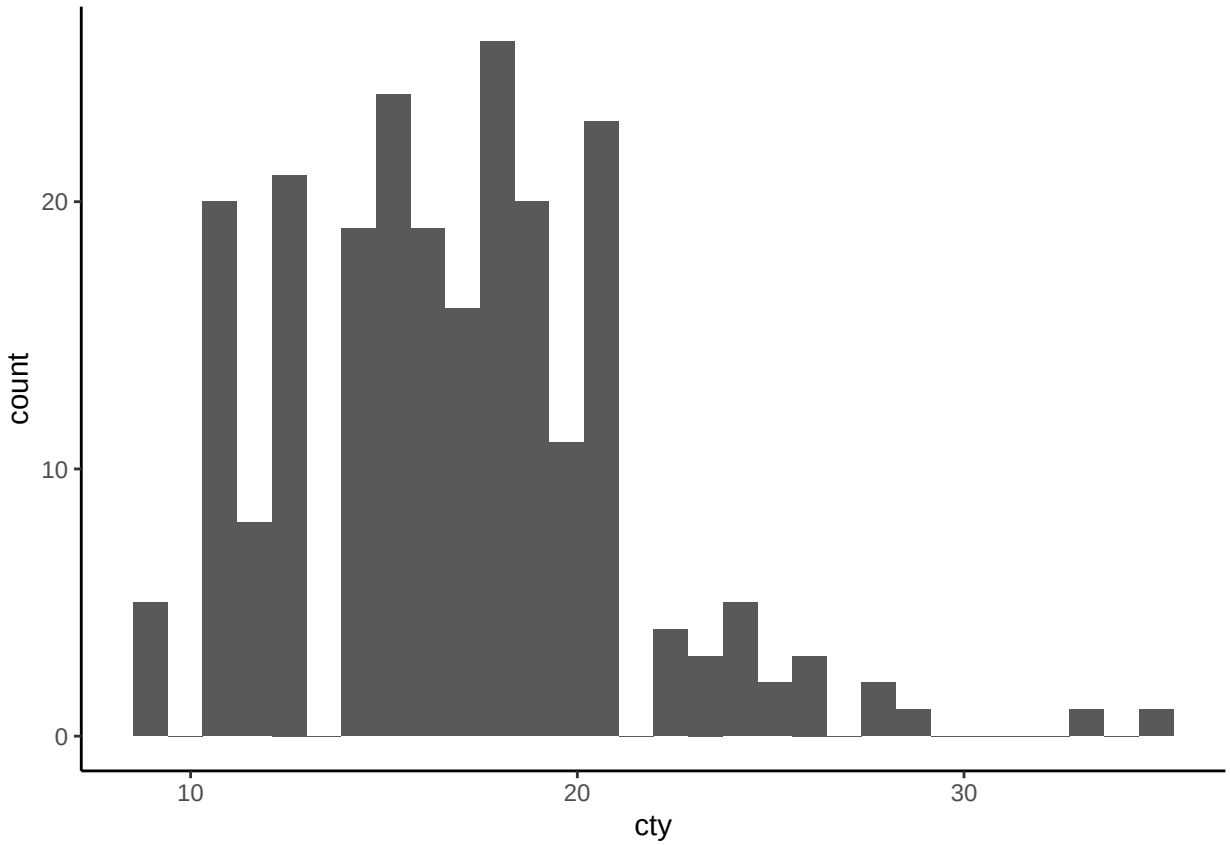
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
cty_hist + geom_histogram() + theme_classic()
```

**Subtask 2**

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

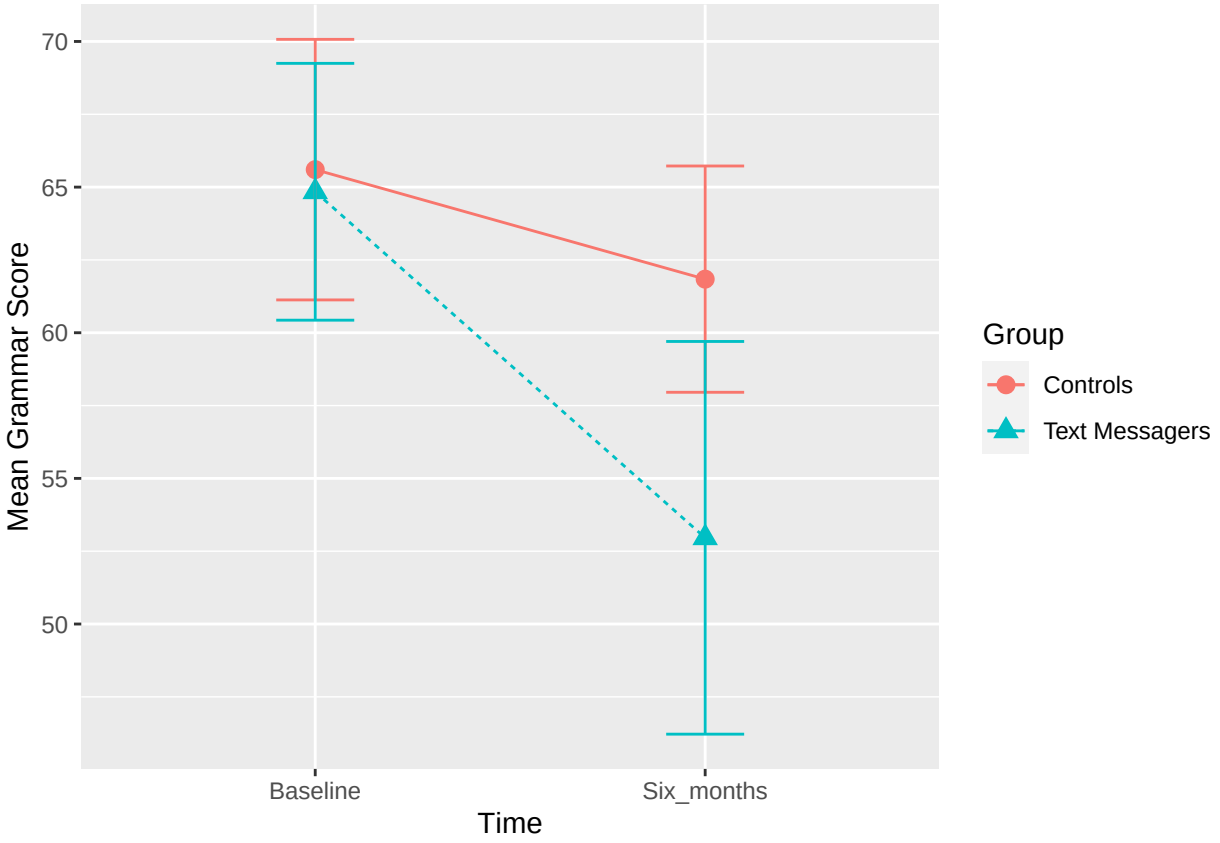



## Reconstruct plot

Use dataset `text_messages.dat` to reconstruct the plot below. You have to first read in the data and then convert them from long to wide format. The data are test results of a grammer test from a baseline and a 6 month follow up. There are two groups that are defined in variable `Groups`.

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```




Tips:

1. The delimiters used in the data file are *tabs*. In R you code tabulators using `"\t"`.
2. You don't have to simply read in the data, you also have to convert it from *wide* to *long* format. Remember the command `gather()` in this context.
3. Remember to set adequate grouping to the layers. You may use variables to group.
4. See ggplot2-Cheatsheet with regard to the commands `geom_point()` and `geom_line()`. Their arguments can also be used in `stat_summary()` if you specified `geom = "point"` or `geom = "line"` there.

**Solutions**

We read the data.

```
text_data <- read_delim("data/text_messages.dat", delim = "\t")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Group = col_character(),
##   Baseline = col_double(),
##   Six_months = col_double()
## )
```

```
# alternatively
text_data <- read_tsv("data/text_messages.dat")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Group = col_character(),
##   Baseline = col_double(),
##   Six_months = col_double()
## )
```

We now convert the data to the long format. The parameter `2:3` is used to tell the command `gather()` that column 2 to 3 should be converted. We could alternatively use the column names, putting `Baseline, Six_months` (This means for R: `Baseline` **and** `Six_months`). The `:` operator can also be used with names: `Baseline:Six_months` would be interpreted by R as `Baseline` **to** `Six_months`.

```
text_data <- text_data %>%
  gather(key = "time", value = "grammar_score", 2:3)
```

We create the plot object. We define here already that colors should be used to code groups.

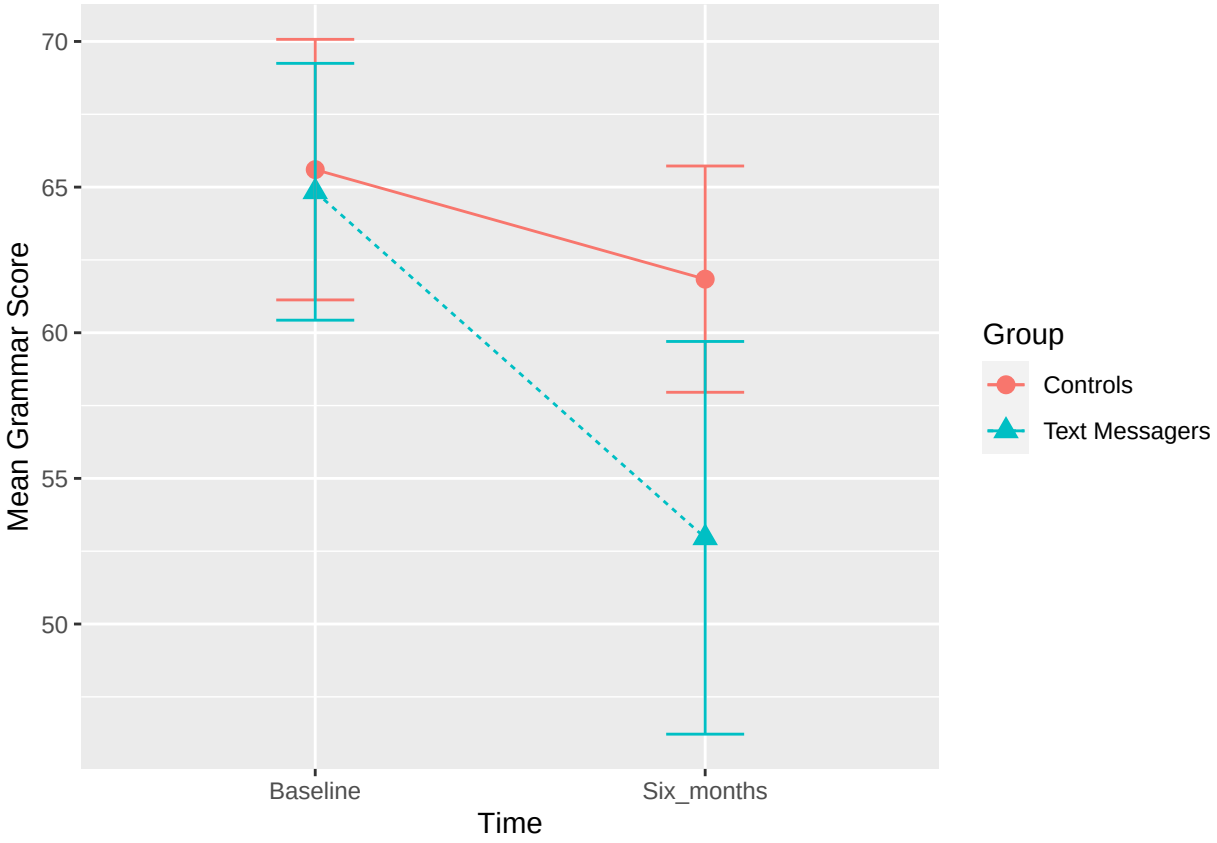
```
text_plot = ggplot(text_data, aes(time, grammar_score, colour = Group))
```

Here we add layers to the plot.

```
text_plot +
  stat_summary(
    fun.y = mean,           # show means
    geom = "point",         # as points
    aes(shape = Group),     # points code groups
    size = 3                # change point size
    ) +
  stat_summary(
    fun.y = mean,           # show means
    geom = "line",          # connect with lines
    aes(group = Group,      # group lines by group
        linetype = Group)   # code groups by line type
  ) +
  stat_summary(
    fun.data = mean_cl_normal, # calculate confidence intervals
    geom = "errorbar",         # show as error bars
    width = 0.2                # limit whiskers width to 20%
    ) +
  labs(x = "Time",
       y = "Mean Grammar Score",
       colour = "Group")
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.

## Warning: `fun.y` is deprecated. Use `fun` instead.
```




## As Barplot

Try to visualize the same information including errorbars as barplot.

Tips:

1. Command `barplot()` understands parameter `fill` that specifies the color of the column (filling) and `color` refers to the frame color.
2. The argument `position = "dodge"` may help, if the columns overlap.
3. The argument `position = position_dodge(width = .90)` may help to adapt smaller layers to larger layers.

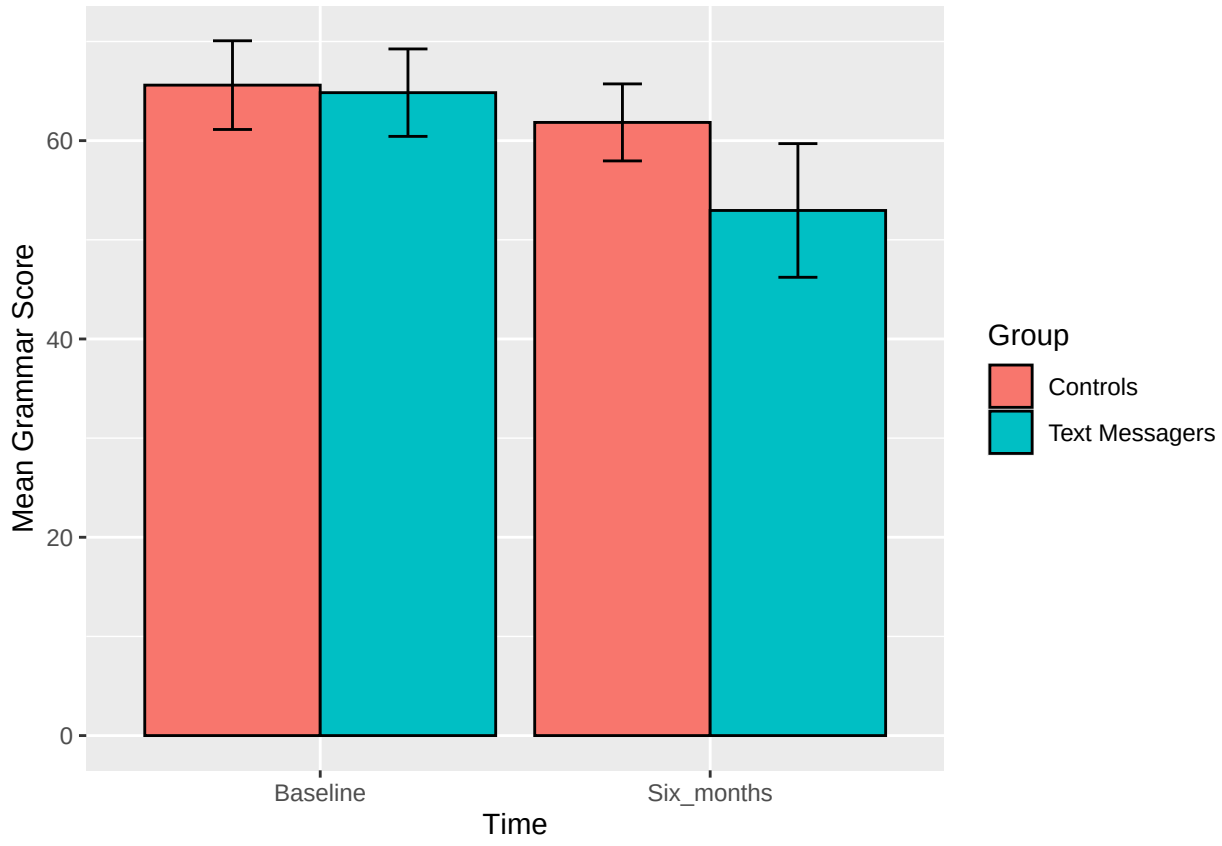
### Solution

```
text_plot = ggplot(text_data, aes(time, grammar_score, fill = Group))
```

```
text_plot +
  stat_summary(
    fun.y = mean,           # show means
    geom = "bar",           # as bars
    position = "dodge",     # put bars side by side
```

```
    color = "black"          # black frame
    ) +
  stat_summary(
    fun.data = mean_cl_normal,    # calculate confidence intervals
    geom = "errorbar",            # show as error bars
    width = 0.2,                  # limit whisker width to 20%
    position = position_dodge(width = .90) # adapt errorbar position to bar position
    ) +
  labs(x = "Time",
       y = "Mean Grammar Score",
       colour = "Group")
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```




## Rendering

Render or knit your Rmd file using the shortcut `strg + shift + k` (Windows) or `cmd + shift + k`. If that works: Well done! If not, look at the error message, in special the lines in the syntax, where the error occured. Correct the error and start over again.

## Literature

*Annotation*: This exercise sheet is based in part on the exercises from the textbook *Dicovering Statistics Using R* (Field, Miles & Field, 2012). We modified it for the purpose of this exercise and actualized the R-Code.

Field, A., Miles, J., & Field, Z. (2012). *Discovering Statistics Using R*. London: SAGE Publications Ltd.

Version: 16 April, 2021 22:39