Anwendungsbaustein Analyse von Verkehrsmengen

Lukas Arnold Simone Arnold Florian Bagemihl
Matthias Baitsch Marc Fehr Franca Hollmann
Maik Poetzsch Sebastian Seipel

2026-02-27

Inhaltsverzeichnis

Einleitung

Zitiervorschlag

Arnold, Lukas, Simone Arnold, Florian Bagemihl, Matthias Baitsch, Marc Fehr, Franca Hollmann, Maik Poetzsch, und Sebastian Seipel. 2026. „Bausteine Computergestützter Datenanalyse. “Anwendungsbaustein Analyse von Verkehrsmengen“. https://github.com/bausteine-der-datenanalyse/a-verkehrsdatenanalyse.

BibTeX-Vorlage

```
@misc{BCD-a-verkehrsdatenanalyse-2026,
 title={Bausteine Computergestützter Datenanalyse.
 Anwendungsbaustein Analyse von Verkehrsmengen},
 author={Arnold, Lukas and Arnold, Simone and Bagemihl, Florian and
 Baitsch, Matthias and Fehr, Marc and Hollmann, Franca and Poetzsch,
 Maik and Seipel, Sebastian},
 year={2026},
 url={https://github.com/bausteine-der-datenanalyse/a-verkehrsdatenanalyse
    }}
```

Voraussetzungen

Zum erfolgreichen Bearbeiten dieses Anwendungsbausteins benötigen Sie die Inhalte des Methodenbausteins Grundlagen der Statistik und des Werkzeugbausteins Datenanalyse in R.

Lernziele

Ziel dieses Bausteines ist es, mit Hilfe der Programmiersprache R Verkehrsmengen, die als CSV-Dateien öffentlich zur Verfügung stehen, auszuwerten und die Ergebnisse in geeigneter Weise grafisch darzustellen.

Verwendete Datensätze

Verwendet werden die Rohdaten der automatischen Dauerzählstellen auf Bundesfernstraßen, die bei der Bundesanstalt für Straßen- und Verkehrswesen (BASt) online als CSV-Datei bezogen werden können.

1 Aufgaben zur Analyse von Verkehrsmengen

Für die Bewertung der bestehenden und für die Planung der zukünftigen Straßenverkehrsinfrastruktur werden Informationen über das aktuelle sowie Prognosen für das zukünftige Verkehrsaufkommen benötigt. Auf Autobahnen und Bundesstraßen unterhält die Bundesanstalt für Straßen- und Verkehrswesen (BASt) dazu Dauerzählstellen. Die gewonnenen Daten sowie die Beschreibung der Daten werden auch online auf der Seite der BASt zur Verfügung gestellt.

Übungsaufgabe 1.1 (Deutschlandkarte mit Zählstellendaten)**.** Erstellen Sie eine Karte des deutschen Autobahnnetzes und stellen Sie in dieser Karte die Verkehrsbelastungen für ein beliebiges Jahr dar.

Übungsaufgabe 1.2 (Vergleich von Verkehrsmengen auf Bundesautobahnen)**.** Wählen Sie zwei Zählstellen von Autobahnen in einem Ballungsraum und zwei weitere Zählstellen von Autobahnen außerhalb eines Ballungsraumes. Führen Sie folgende Auswertungen durch:

a) Erstellen Sie für eine ausgewählte Zählstelle eine Ganglinie für jeden Wochentag und stellen Sie diese in einem Diagramm dar.

b) Bestimmen Sie die mittlere werktägliche Verkehrsbelastung (mittlere Stundenwerte dienstags bis donnerstags, ohne Feiertage und außerhalb der Schulferien) und stellen Sie die Ganglinien eines mittleren Werktages für alle Zählstelle in einem Diagramm dar. Erstellen Sie mit denselben Filtern (ohne Feiertage und außerhalb der Schulferien) für alle Zählstelle ein Diagramm für Samstage und Sonntage.

c) Bestimmen Sie für eine der vier Zählstellen die Verkehrsstärke der 50. höchst belasteten Stunde des Jahres und geben das Ergebnis in einer Tabelle aus. Wählen Sie zudem eine geeignete Darstellungsform, um die 50. Stunde im Vergleich zu den restlichen Stunden eines Jahres auszugeben.

d) Untersuchen Sie den prozentualen Anteil des Schwerverkehrs in der Zeitspanne 2003 bis 2023 auf deutschen Autobahnen. Welche Entwicklung / Tendenz lässt sich erkennen?

2 Musterlösung mit Erläuterungen

2.1 Aufgabe 1.1 : Deutschlandkarte mit Zählstellendaten

Als ersten Schritt interessieren uns besonders stark befahrene Streckenabschnitte von Autobahnen und wir wollen diese in einer Deutschlandkarte darstellen. Wir gehen hierbei schrittweise vor:

1. Erstellung einer Deutschlandkarte.
2. Hinzufügen des Autobahnnetzes in die Deutschlandkarte.
3. Die Daten der Zählstellen in die Deutschlandkarte plotten.

2.1.1 Deutschlandkarte

Als erstes erstellen wir eine Deutschlandkarte.

Mit dem Befehl `gisco_get_nuts()` aus dem Paket `giscoR` werden NUTS (Nomenclature of territorial units for statistics) -Regionen als `sf` (simple feature) Polygone, Punkte und Linien ausgegeben. Wir interessieren uns für die deutschen Grenzen (`nuts_level = 0`) und wollen diese als Datensatz speichern.

```
d_de <- gisco_get_nuts(country = "Germany", nuts_level = 0, resolution =
    03)
```

Die verwendeten Elemente lassen sich in drei Abschnitte unterteilen:

1. `country =` gibt an um welches Land es sich handelt.
2. `nuts_level =` gibt Ebene der NUTS an. (0 für Staat, 1 für Bundesländer, 2 für Regierungsbezirke)
3. `resolution =` bestimmt die Auflösung.

2.1.2 Autobahnverlauf

Für Deutschland stellt das BASt jedes Quartal einen aktualisierten Datensatz zum Bundesfernstraßennetz zur Verfügung. Diesen laden wir ins Projekt. Das Bundesfernstraßennetz beinhaltet Daten zu Bundesautobahnen und Bundesstraßen. Aufgrund der Größe des Datensatzes kann dieser nicht auf Github zur Verfügung gestellt werden, daher fügen wir eine Funktion ein, die prüft, ob der Datensatz bereits im "daten/geo"-Ordner liegt und falls nein, er direkt von der Webseite der BASt heruntergeladen, entpackt und an der richtigen Stelle gespeichert wird. Da für uns nur die Autobahnen relevant sind, filtern wir diese heraus. Wir wählen als Achse die Bestandsachse, da dies die mittlere Achse der Straße ist und wir keine Unterscheidung der Fahrtrichtung vornehmen müssen. Wir ergänzen eine Spalte ("mutate") mit der Zeilennummer (wird später zur Aufbereitung der Dauerzählstellen benötigt). Falls sich in den geometrischen Daten noch Informationen zur Höhe (z-Dimension) oder Messwerte oder Attribute (m-Dimension) enthalten sind, werden diese nun entfernt, damit wir gleich sauber mit ihnen weiterrechnen können.

```
if (!file.exists("daten/bfstn.RData")) {
  file_gpkg <- list.files("daten/geo", pattern = "^BFStr_Netz.*\\.gpkg$",
                          full.names = TRUE) |>
    tail(n = 1)

  if (length(file_gpkg) == 0) {
    curl_download(
      "https://www.bast.de/SharedDocs/Daten-TB/Daten-BISStra.zip?__blob=
         publicationFile&v=5",
      destfile = "daten/geo/Daten-BISStra.zip",
      quiet = FALSE
    )
    unzip("daten/geo/Daten-BISStra.zip", exdir = "daten/geo/")
    file_gpkg <- list.files("daten/geo",
                            pattern = "^BFStr_Netz.*\\.gpkg$",
                            full.names = TRUE) |>
      tail(n = 1)
  }

  d_bfstn <- read_sf(file_gpkg) |>
    filter(Str_Klasse_kurz == "A" & Sk_Achse == "Bestandsachse") |>
    mutate(rownumber = row_number()) |>
    st_zm()

  save(d_bfstn, file = "daten/bfstn.RData")
} else {
  load("daten/bfstn.RData")
}
```

2.1.3 Zählstellendaten

Nun fügen wir die Verkehrsdaten als Datensatz hinzu. Wir verwenden hier die Daten von Dauerzählstellen auf Autobahnen für das Jahr 2023, die die Bundesanstalt für Straßenwesen (BASt) zur Verfügung stellt. Durch `locale = locale(encoding = 'iso-8859-1')` bestimmen wir die Kodierung nach der ISO-Norm 8859-1, damit die Datei auch in anderen Systemen geladen werden kann. Wir filtern nach der Straßenklasse "A" für Autobahn und nach allen Dauerzählstellen, die auch tatsächlich Daten enthalten.

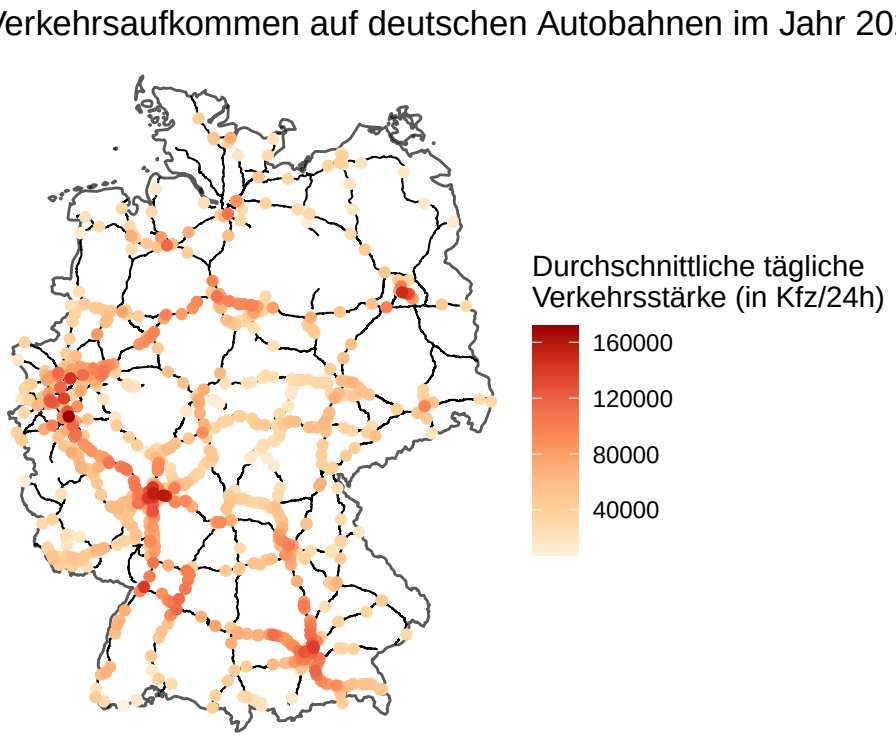
```
d_dzs_2023 <- read_csv2("daten/Jawe2023.csv",
                        locale = locale(encoding = 'iso-8859-1')) |>
  filter(Str_Kl == "A") |>
  drop_na(DTV_Kfz_MobisSo_Q) |>
  arrange(DTV_Kfz_MobisSo_Q)
```

2.1.4 Grafische Darstellung

Nun können wir alle Elemente kombinieren: Wir plotten nun also die Deutschlandkarte, die Karte des Autobahnverlaufs und die Daten der Zählstellen in eine Grafik. Ebenfalls in der Datensatzbeschreibung können wir die Variablen für die Länge (`Koor_WGS84_E`) und Breite (`Koor_WGS84_N`) der Lagekoordinaten in WGS84 für die jeweilige Zählstelle finden. Diese Werte setzen wir als x-Koordinate und y-Koordinate ein.

Für Barrierefreiheit nutzen wir die Farbpalette `scale_color_distiller`. Mit `theme_void` sorgen wir dafür, dass `ggplot`-Thema komplett leer ist, wir also keine Achsen, Achsenbeschriftungen, Hintergründe oder Gitternetzlinien in unserer Grafik sehen.

```
ggplot() +
  geom_sf(data = d_de, fill = NA, size = 0.5) +
  geom_sf(data = d_bfstn,  size = 0.35, show.legend = FALSE) +
  geom_point(data = d_dzs_2023,
             mapping = aes(
              x = Koor_WGS84_E,
              y = Koor_WGS84_N,
              color = DTV_Kfz_MobisSo_Q),
             size = 1.5, na.rm = FALSE) +
  scale_color_distiller(palette = 8 , direction = 1) +
  theme_void() +
  ggtitle("Verkehrsaufkommen␣auf␣deutschen␣Autobahnen␣im␣Jahr␣2023") +
  labs(color = "Durchschnittliche␣tägliche␣\nVerkehrsstärke␣(in␣Kfz/24h)")
```



2.2 Aufgabe 1.2: Vergleich von Verkehrsmengen auf Bundesautobahnen

Als weitere wichtige Größe in der Verkehrsdatenanalyse schauen wir uns Tagesganglinien an. Dazu benötigen wir die Daten einer Zählstelle für jeden Tag. Eine Auflistung der Zählstellen und ihres Datensatzes findet sich ebenfalls auf der Seite der BASt.

Exemplarisch verwenden wir hier die Zählstelle 5116 der A42 in Wanne-Eickel. Diese Daten müssen wir zuerst einlesen:

```
d_zst5116 <- read_csv2("daten/zst5116_2023.csv",
                       locale = locale(encoding = 'iso-8859-1'))
```

2.2.1 Tagesganglinien

Wir wollen nun die Tagesganglinien der ausgewählten Zählstelle für jeden Wochentag erstellen. Wir brauchen also Mittelwerte für jede Stunde jedes Wochentages. Mithilfe der Funktion `group_by()` können wir unseren Datensatz nach Variablen gruppieren. Für unseren Fall sind dies die Variablen `Stunde` für die Erhebungsstunde und `Wotag` für den Wochentag.

Danach leiten wir diese Werte an die `summarise()`-Funktion weiter. Diese gibt für jede Kombination der Gruppierungsvariable eine Reihe aus. Dazu benötigen wir für Kombination ebenfalls die Mittelwerte für beide Fahrtrichtungen (`KFZ_R1` und `KFZ_R2`). Diesen Datensatz speichern wir als neues Element `ZST5116_DTV`. Für die weitere Verwendung bietet es sich an, die Variable `Wotag` mit `factor ()` in einen Faktor umzuwandeln und die Ebenen des Faktors (`levels =`) korrekt nach den Wochentagen zu benennen (`labels =`).

```
d_zst5116_dtv <- d_zst5116 |>
  group_by(Stunde, Wotag) |>
  summarise(avg_KFZ_R1 = mean(KFZ_R1),avg_KFZ_R2 = mean(KFZ_R2))

d_zst5116_dtv$Wochentag <- factor(d_zst5116_dtv$Wotag,
  levels = c(1,2,3,4,5,6,7),
  labels = c("Mo","Di","Mi","Do","Fr","Sa","So"))
```

Nun können wir diese Daten (exemplarisch hier Fahrtrichtung 1 mit `KFZ_R1`) in `ggplot2` verwenden. Wir wollen für eine Tagesganglinie nun für jede Stunde einen Datenpunkt darstellen und diese mit einer Linie verbinden.

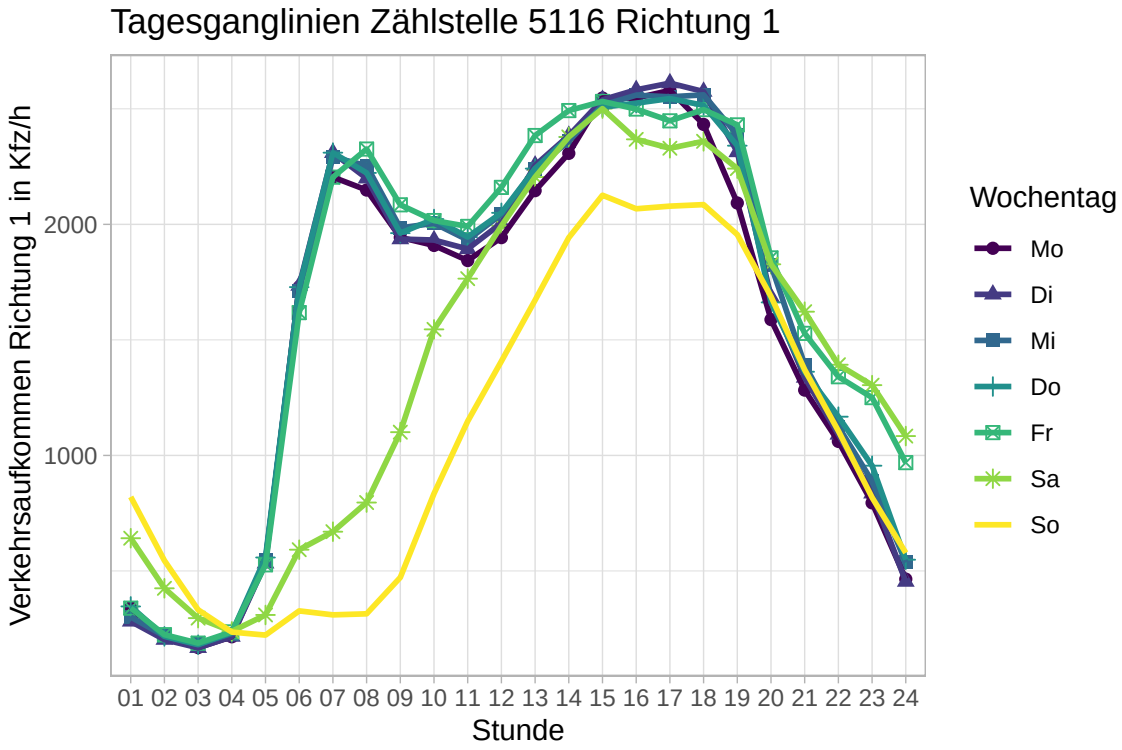
Zuerst erstellen wir mit `ggplot()` unsere Umgebung zum Erstellen von Grafiken. Da wir wie angesprochen zwei Formen (Datenpunkte und Linien) verwenden, bietet es sich an, die Einstellungen wie der verwendete Datensatz und die verwendeten Variablen global in `ggplot()` und nicht bei den einzelnen Elementen festzulegen. Dazu legen wir mit `data = ZST5116_DTV` den neu erstellten Datensatz als Datensatz fest. Mit `aes()` legen wir fest, wie Variablen visuelle Eigenschaften zugeordnet werden. Mit `x = Stunde` und `y = avg_KFZ_R1` bestimmen wir, dass die Stunde auf der X-Achse abgetragen wird und unsere neu erstellten Mittelwerte auf der y-Achse. Wir wollen unsere Daten aber noch nach Wochentagen aufteilen und einfärben und dies geschieht mit `group = Wochentag` und `color = factor(Wochentag)`. All diese Eigenschaften werden an nachfolgende *geoms* "weitervererbt". Mit `geom_point` fügen wir die Datenpunkte hinzu und mit `geom_line` die Linie. Die Zusatzoptionen `size = 2` und `linewidth = 1` verändern die Größe der Elemente und dienen der Ästhetik, sind aber natürlich persönliche Präferenz. Zuguterletzt beschriften wir mit `labs()` unsere Grafik. Dazu ändern wir mit `title =` den Titel, mit `x =` und `y =` die Beschriftung der jeweiligen Achse und mit `color =` die Legende für die Farben.

Als Zusatzoption verwenden wir mit `scale_color_viridis_d` die Viridisfarbpalette, die so gestaltet wurde, dass Personen mit häufigen Formen von Farbenblindheit diese besser wahrnehmen können.

Mit `theme_light()` verwenden wir ein vorgefertigtes Thema, um die Darstellungsform aller nicht Daten-Elemente zu steuern. Auch hier handelt es sich um eine persönliche Präferenz und andere Themen können ebenfalls gewählt werden und gute Grafiken erzeugen.

```
ggplot(data = d_zst5116_dtv,
       aes(
         x = Stunde,
         y = avg_KFZ_R1,
```

```
        group = Wochentag,
        color = factor(Wochentag),
        shape = Wochentag)) +
geom_point(size = 2) +
geom_line(linewidth = 1) +
labs(title = "Tagesganglinien␣Zählstelle␣5116␣Richtung␣1",
     x = "Stunde",
     y = "Verkehrsaufkommen␣Richtung␣1␣in␣Kfz/h",
     color = "Wochentag",
     shape = "Wochentag") +
scale_color_viridis_d() +
theme_light()
```



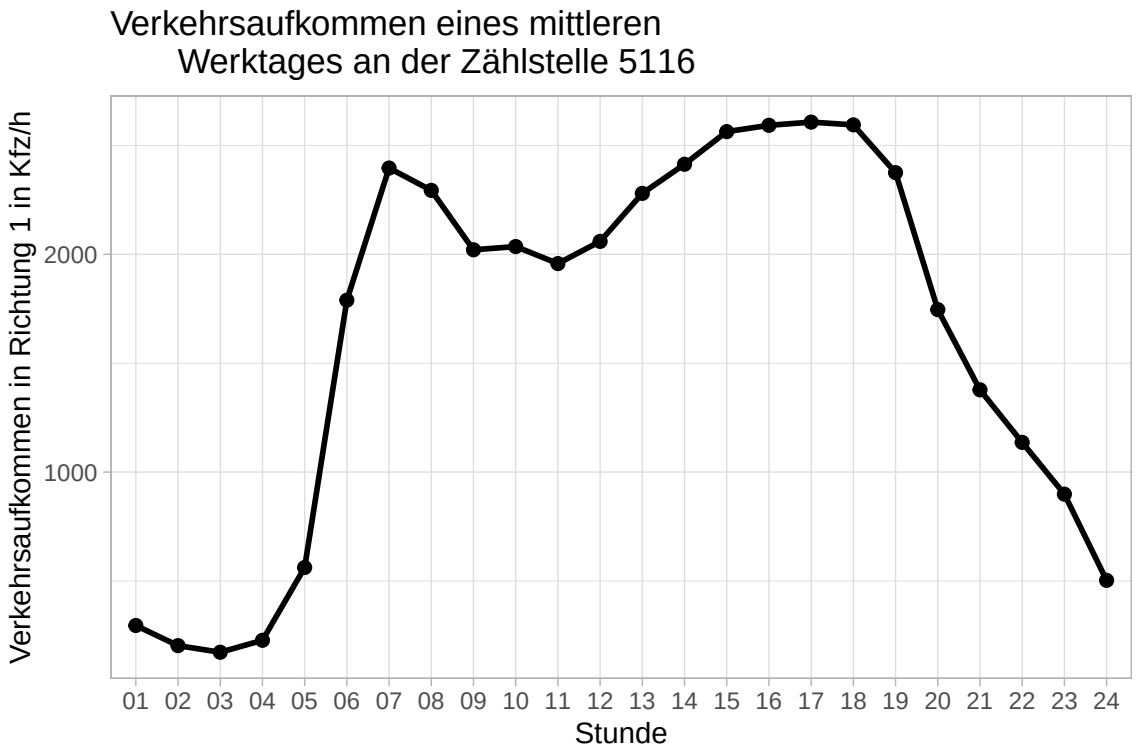
2.2.2 Mittlere werktägliche Verkehrsbelastung an vier Zählstellen

Für die mittlere werktägliche Verkehrsbelastung interessieren uns die Wochentage Dienstag, Mittwoch und Donnerstag an normalen Werktage außerhalb der Schulferien. Ein Blick in die Datensatzbeschreibung der BASt zeigt uns, dass diese Unterteilung im Datensatz bereits vorliegt. Die Variable `Fahrtzw` besitzt die Ausprägungen w für Werktage, u für Urlaubswerktage und s für Sonn- und Feiertage. Somit müssen wir in unseren Filter lediglich die Abfrage `Fahrtzw == "w"` einfügen.

```
d_zst5116_w <- d_zst5116 |>
  filter(Fahrtzw == "w", Wotag %in% c(2:4)) |>
  group_by(Stunde) |>
  summarise(avg_KFZ_R1 = mean(KFZ_R1),avg_KFZ_R2 = mean(KFZ_R2)) |>
  mutate(
    anteil_R1 = avg_KFZ_R1/sum(avg_KFZ_R1)*100
  )
```

Diese Daten können wir nun analog zum ersten Teil dieser Aufgabe plotten:

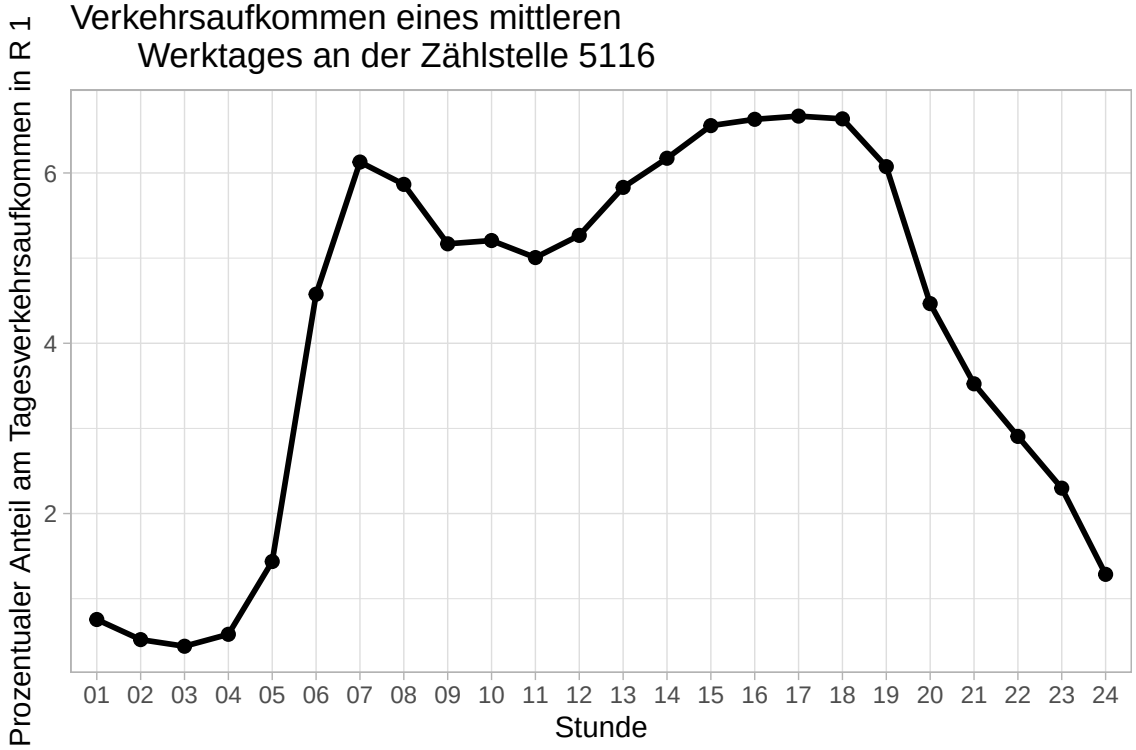
```
ggplot(data = d_zst5116_w, aes(x = Stunde, y = avg_KFZ_R1, group = 1)) +
  geom_point(size = 2) +
  geom_line(linewidth = 1) +
  labs(title = "Verkehrsaufkommen␣eines␣mittleren
␣␣␣␣␣␣␣Werktages␣an␣der␣Zählstelle␣5116",
       x = "Stunde",
       y = "Verkehrsaufkommen␣in␣Richtung␣1␣in␣Kfz/h") +
  theme_light()
```



Um ein besseres Verständnis zu bekommen, wie groß der Anteil der jeweiligen Stunde am Verkehrsaufkommen ist, können wir dies ebenfalls plotten:

```
ggplot(data = d_zst5116_w, aes(x = Stunde, y = anteil_R1, group = 1)) +
  geom_point(size = 2) +
```

```
  geom_line(linewidth = 1) +
  labs(title = "Verkehrsaufkommen␣eines␣mittleren
␣␣␣␣␣␣␣Werktages␣an␣der␣Zählstelle␣5116",
       x = "Stunde",
       y = "Prozentualer␣Anteil␣am␣Tagesverkehrsaufkommen␣in␣R␣1") +
  theme_light()
```



Oft möchte man nicht nur die Daten einer Zählstelle auswerten, sondern mehrere Zählstellen miteinander vergleichen oder Entwicklungen über Jahre auswerten. Um mehrere Zählstellen miteinander zu vergleichen gehen wir zuerst exakt so vor wie in für die Zählstelle 5116. Wir importieren hier nun die Datensätze der Zählstellen 5113, 5125 und 5128. Die Zählstellen 5116 (A42, Wanne-Eickel) und 5033 (A3, Leverkusen) in einem Ballungsraum, die Zählstellen 5117 (A31 bei Coesfeld) und 5136 (A44 bei Lichtenau) außerhalb von Ballungsräumen.

```
d_zst5033 <- read_csv2("daten/zst5033_2023.csv",
                       locale = locale(encoding = 'iso-8859-1'))
d_zst5117 <- read_csv2("daten/zst5117_2023.csv",
                       locale = locale(encoding = 'iso-8859-1'))
d_zst5136 <- read_csv2("daten/zst5136_2023.csv",
                       locale = locale(encoding = 'iso-8859-1'))
```

> **Mit Funktionen mehrere Datensätze einlesen**
>
> Bei einer großen Anzahl an Datensätze mit gleichen Einlesungsoperationen bieten sich Funktionen an, um effizienter zu arbeiten und mögliche Fehler bei *copy-and-paste* zu verhindern: siehe dazu auch das Kapitel "Mehrere Datensätze" aus den Werkzeugbausteinen R

Um diese verschiedenen Datensätze in einem Datensatz zu kombinieren, verwenden wir die `bind_rows()`-Funktion des Pakets `dplyr` aus dem `tidyverse`. Dazu geben wir dem Befehl eine Liste der zu kombinierenden Datensätze mit `list()` und fügen mit `.id =` eine Identifizierungsspalte hinzu. Standardmäßig werden diese Variable der Identifizierungsspalte einfach durchnummeriert. Es bietet sich aber an die jeweilige Nummer oder den Namen der Zählstelle zu verwenden. In unserem Fall entscheiden wir uns für den Namen. Wir wandeln die Variable `zaehlstelle` in einen Faktor mit den Namen als Ebenen um.

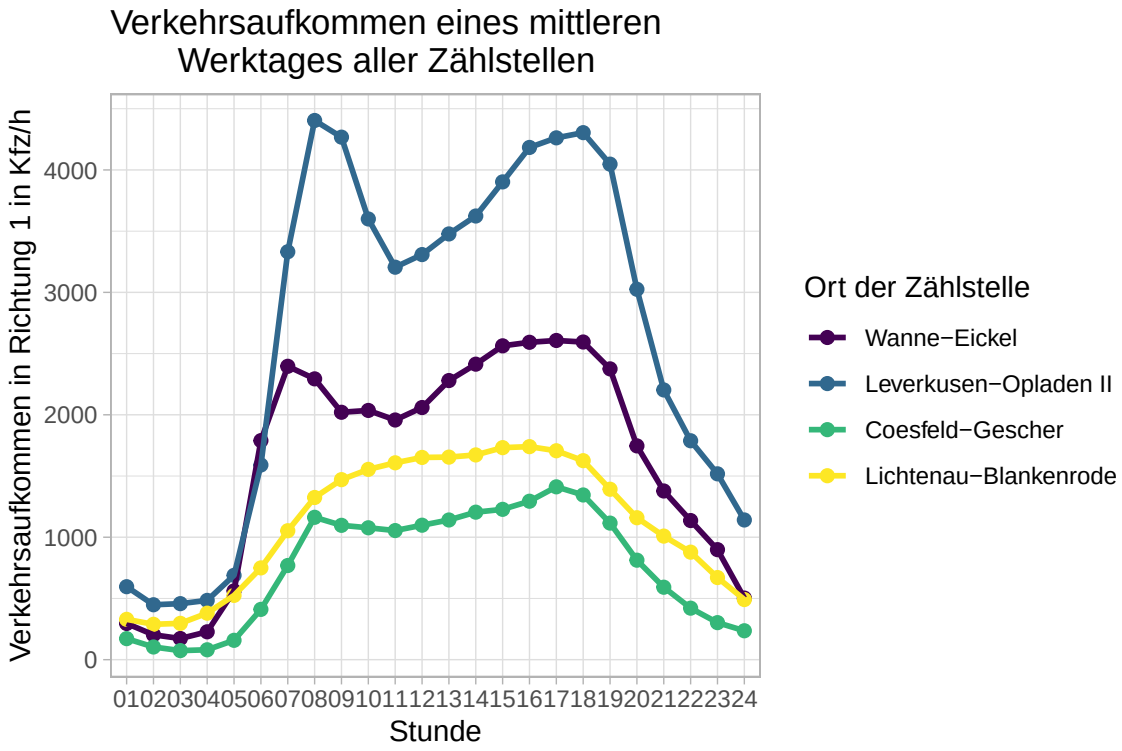
```
d_zst_alle <- bind_rows(
    list(d_zst5116, d_zst5033, d_zst5117, d_zst5136),
    .id = "zaehlstelle"
  ) |>
  mutate(
    zaehlstelle = factor(zaehlstelle,
      levels = c("1", "2", "3", "4"),
      labels = c("Wanne-Eickel", "Leverkusen-Opladen␣II",
                 "Coesfeld-Gescher", "Lichtenau-Blankenrode"))
  )
```

```
d_zst_alle_w <- d_zst_alle |>
  filter(Fahrtzw == "w", Wotag %in% c(2:4)) |>
  group_by(Stunde, zaehlstelle) |>
  summarise(avg_KFZ_R1 = mean(KFZ_R1),avg_KFZ_R2 = mean(KFZ_R2)) |>
  mutate(
    anteil_R1 = avg_KFZ_R1/sum(avg_KFZ_R1)*100
  )
```

Wenn wir diese gefilterten Daten darstellen, erhalten wir folgenden Plot:

```
ggplot(data = d_zst_alle_w,
       aes(
         x = Stunde,
         y = avg_KFZ_R1,
         group = zaehlstelle,
         color = zaehlstelle)) +
  geom_point(size = 2) +
  geom_line(linewidth = 1) +
  labs(title = "Verkehrsaufkommen␣eines␣mittleren
␣␣␣␣␣␣␣Werktages␣aller␣Zählstellen",
       x = "Stunde",
       y = "Verkehrsaufkommen␣in␣Richtung␣1␣in␣Kfz/h",
```

```
      color = "Ort␣der␣Zählstelle") +
  scale_color_viridis_d() +
  theme_light()
```



Analog zu den Werktagen filtern wir unsere Daten nun für Samstage. Für die Sonntage ist ein Blick in die Datensatzbeschreiben (siehe oben) hilfreich. In den Daten werden Sonn- und Feiertage gemeinsam aufgeführt, sodass wir unseren Filter dort nicht anwenden können.

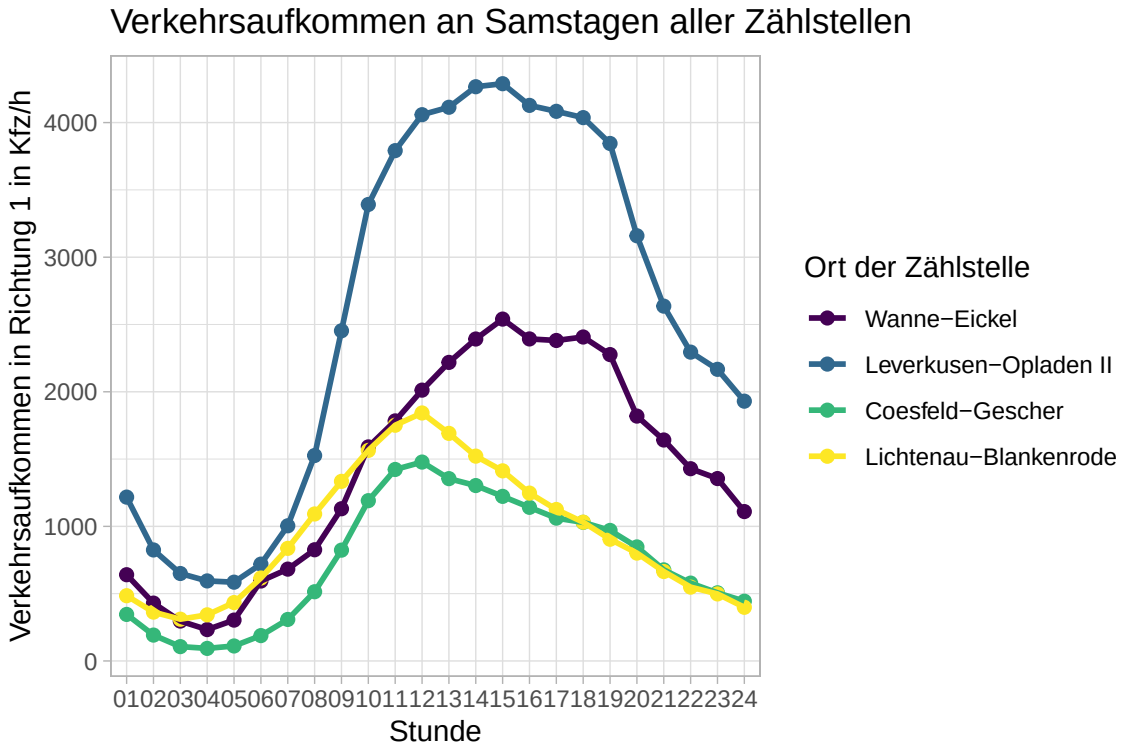
```
#Samstag
d_zst_alle_sa <- d_zst_alle |>
  filter(Fahrtzw == "w", Wotag == "6") |>
  group_by(Stunde, zaehlstelle) |>
  summarise(avg_KFZ_R1 = mean(KFZ_R1),avg_KFZ_R2 = mean(KFZ_R2)) |>
  mutate(
    anteil_R1 = avg_KFZ_R1/sum(avg_KFZ_R1)*100
  )

#Sonntag
d_zst_alle_so <- d_zst_alle |>
  filter(Wotag == "7") |>
  group_by(Stunde, zaehlstelle) |>
  summarise(avg_KFZ_R1 = mean(KFZ_R1),avg_KFZ_R2 = mean(KFZ_R2)) |>
  mutate(
```

```
    anteil_R1 = avg_KFZ_R1/sum(avg_KFZ_R1)*100
  )
```

Plot für Samstage:

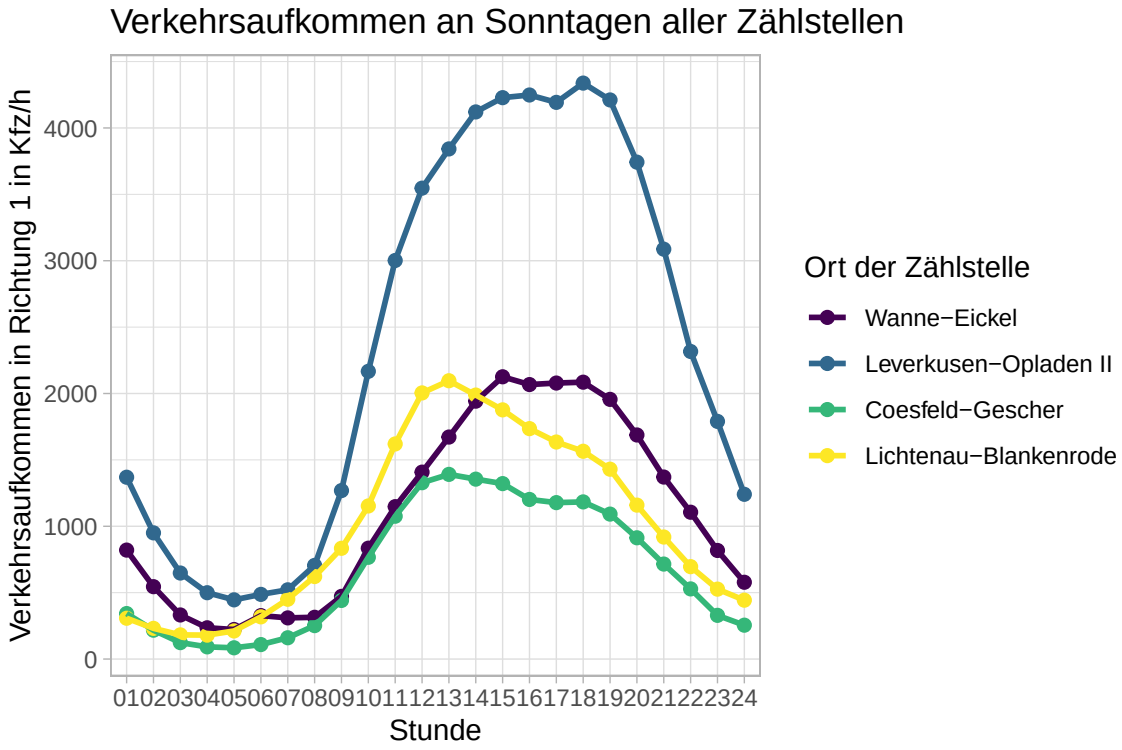
```
ggplot(data = d_zst_alle_sa,
       aes(
         x = Stunde,
         y = avg_KFZ_R1,
         group = zaehlstelle,
         color = zaehlstelle)) +
  geom_point(size = 2) +
  geom_line(linewidth = 1) +
  labs(title = "Verkehrsaufkommen an Samstagen aller Zählstellen",
       x = "Stunde",
       y = "Verkehrsaufkommen in Richtung 1 in Kfz/h",
       color = "Ort der Zählstelle") +
  scale_color_viridis_d() +
  theme_light()
```



Plot für Sonntage:

```
ggplot(data = d_zst_alle_so,
       aes(
```

```
        x = Stunde,
        y = avg_KFZ_R1,
        group = zaehlstelle,
        color = zaehlstelle)) +
geom_point(size = 2) +
geom_line(linewidth = 1) +
labs(title = "Verkehrsaufkommen an Sonntagen aller Zählstellen",
     x = "Stunde",
     y = "Verkehrsaufkommen in Richtung 1 in Kfz/h",
     color = "Ort der Zählstelle") +
scale_color_viridis_d() +
theme_light()
```



2.2.3 50. Stunde

Als einen weiteren wichtigen Kennwert schauen wir uns die Verkehrsstärke der 50. höchst belasteten Stunde des Jahres, die sogenannte "50. Stunde" an. Für eine rein deskriptive Beschreibung der 50. Stunden können wir den Datensatz absteigend sortieren und dann den 50. Wert auswählen. Dazu benutzen wir erneut die `arrange()`-Funktion, dieses Mal aber mit dem Zusatz `desc()`(kurz für englisch *descending* = absteigend). Um eine Reihe anhand der Position auszuwählen, verwenden wir den `slice()`-Befehl. Dies leiten wir nun an die `select()`-Funktion

weiter, die es uns ermöglicht anhand des Namens auszuwählen, welche Variablen wir für unsere Darstellung behalten möchten. In diesem Fall wählen wir das Datum (`Datum`), den Wochentag (`Wotag`), die Stunde des Tages (`Stunde`) und den tatsächlichen Verkehrswert (`KFZ_R1`) aus.

```
d_zst5116_50 <- d_zst5116 |>
    arrange(desc(KFZ_R1)) |>
    slice(50) |>
  select(Datum,Wotag,Stunde,KFZ_R1)
```

Mit der Funktion `kable()` aus dem Paket `kableExtra` können wir dies nun tabellarisch darstellen. Wir sehen, dass es sich um Samstag (Wotag 6), den 22. April (Datum 230422) in der 16. Stunde handelt und in diesem Zeitraum 2935 KFZ erfasst wurden.

```
kable(d_zst5116_50)
```

Datum	Wotag	Stunde	KFZ_R1
230422	6	16	2935

Dies zeigt uns aber nur den einzelnen Wert der 50. Stunde, ohne dass wir ihn in Relation zu den anderen Werten setzen können. Ein wichtiger Bestandteil der 50. Stunde ist die Einschätzung des Verkehrsaufkommens und wie hoch der Anteil der 50. Stunde im Vergleich zur Topstunde ist. Dafür bietet sich eine Darstellung als Säulendiagramm an.

Wir erstellen einen neuen Datensatz, der das Verkehrsaufkommen in absteigender Reihenfolge darstellt (dieselbe Vorgehensweise wie für unsere deskriptive Beschreibung). Im nächsten Schritt fügen wir eine Variable hinzu, die für uns die Position im Datensatz durchnummeriert (simple Lösung mit `row_number()`).

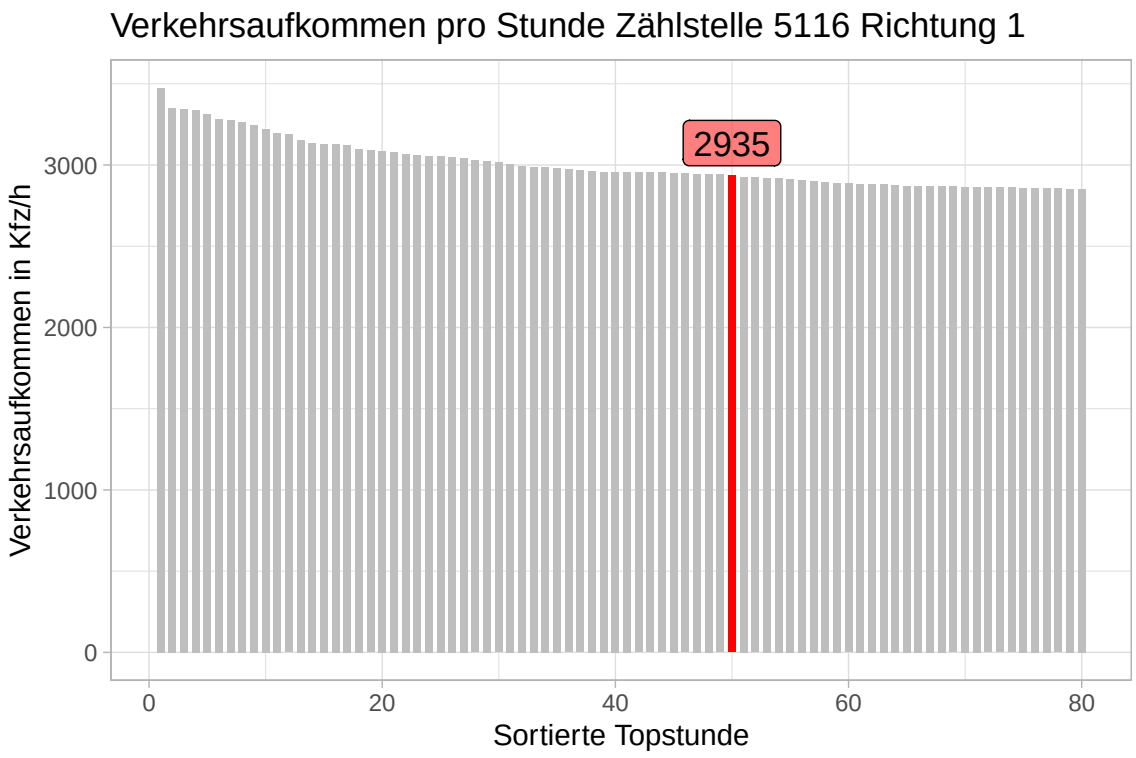
```
d_zst5116_top <- d_zst5116 |>
  arrange(desc(KFZ_R1)) |>
  mutate(
    nummer = row_number()
  )
```

Nun können wir unsere Säulendiagramme plotten: Um nicht alle Beobachtungen zu plotten, wird der zu plottende Datensatz mithilfe von dem Zusatz `[1:80,]` bei der Festlegung des Datensatzes auf die ersten 80 Beobachtungen reduziert. Es bietet sich an, die 50. Stunde farblich hervorzuheben und mit einem Label zu versehen. Dazu verwenden wir einen logischen Operator mit `fill = Nummer == 50` und können dann mit `scale_fill_manual()` das Aussehen festlegen.

Das Label wird mithilfe der Funktion `geom_label`, die analog zu bereits bekannten *geoms* aus `ggplot2` funktioniert, erstellt. Wir filtern den Datensatz mit `data = filter(d_zst5116_top, Nummer == 50)`, sodass nur die 50. Stunde ein Label erhält und wählen als Inhalt des Labels den KFZ-Wert mit `mapping = aes(label = KFZ_R1)`. Die restlichen Optionen dienen der Ästhetik:

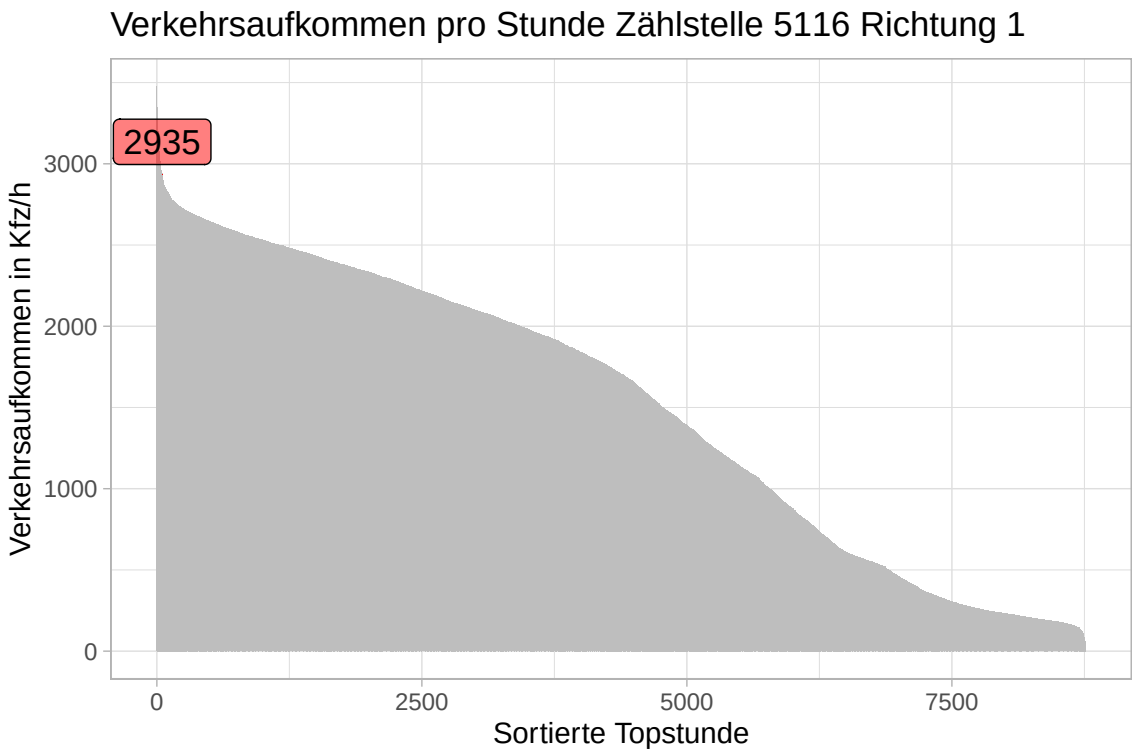
- `nudge_y` verschiebt das Label um eine festgestelegte vertikale Distanz zur besseren Lesbarkeit. Für horizontale Positionierung wird analog `nudge_x`verwendet.
- `size` bestimmt die Größe des Labels.
- `alpha` stellt die Transparenz ein.

```
ggplot(data = d_zst5116_top[1:80, ],
       aes(
         x = nummer,
         y = KFZ_R1,
         fill = nummer == 50,
         width = .65)) +
  geom_bar(stat = "identity") +
  labs(title = "Verkehrsaufkommen␣pro␣Stunde␣Zählstelle␣5116␣Richtung␣1",
       x = "Sortierte␣Topstunde",
       y = "Verkehrsaufkommen␣in␣Kfz/h") +
  scale_fill_manual(values = c("grey", "red"), guide = "none") +
  geom_label(
    data = filter(d_zst5116_top, nummer == 50),
    mapping = aes(label = KFZ_R1),
    nudge_y = 200, size = 4.5, alpha = 0.5
  ) +
  theme_light()
```



Um einen Vergleich mit dem gesamten Verkehrsaufkommen zu erstellen, folgt nun dieses Balkendiagramm mit den Daten aller Stunden:

```
ggplot(data = d_zst5116_top,
       aes(
         x = nummer,
         y = KFZ_R1,
         fill = nummer == 50,
         width = .65)) +
  geom_bar(stat = "identity") +
  labs(title = "Verkehrsaufkommen pro Stunde Zählstelle 5116 Richtung 1",
       x = "Sortierte Topstunde",
       y = "Verkehrsaufkommen in Kfz/h") +
  scale_fill_manual(values = c("grey", "red"), guide = "none") +
  geom_label(
    data = filter(d_zst5116_top, nummer == 50),
    mapping = aes(label = KFZ_R1),
    nudge_y = 200, size = 4.5, alpha = 0.5
  ) +
  theme_light()
```



2.2.4 Schwerverkehr-Anteile über 20 Jahre

Nun schauen wir uns an, wie wir den Verlauf der Schwerverkehr-Anteile über mehrere Jahre hinweg darstellen können.

Um mehrere Dateien gleichzeitig einzulesen, bietet es sich an, eine `function` zu verwenden, damit dieser Schritt in einem Durchlauf erledigt wird und man mögliche Fehler beim Kopieren und Einfügen vermeidet.

Leider besitzt der Datensatz der Zählstellen keine Variable wie `Jahr` anhand derer wir die Identifikation herstellen können. Allerdings besitzen die Dateinamen eine Jahreszugehörigkeit. Daher verwenden wir die Namen der Zählstellendateien als Identifikation und benennen diese Variable mit `id = "Jahr"` als Jahr. Nun müssen wir die anderen Elemente des Strings aus den Dateinamen mit `str_remove_all` entfernen. Am Schluss wandeln wir den String des Dateinamens, der nun nur noch die Jahreszahl enthält mit `as_numeric` in eine Zahl um.

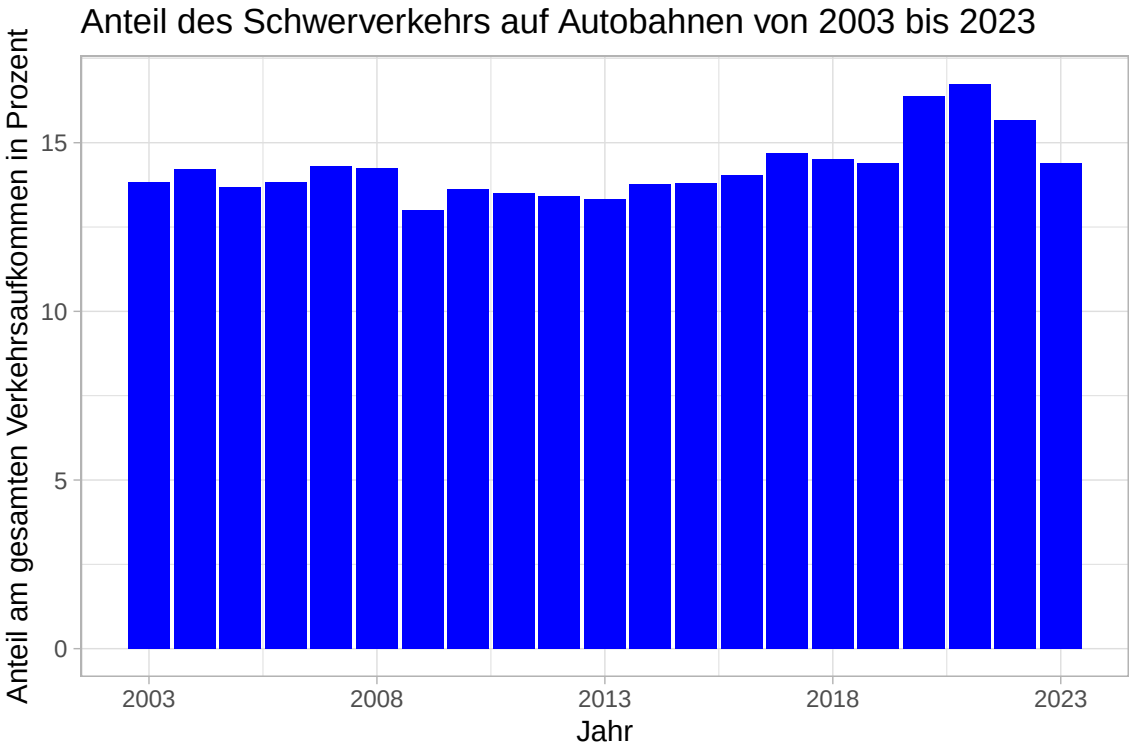
```
dateien <- list.files(path = "daten",
                      pattern = "Jawe.*\\.csv$",
                      recursive = TRUE,
                      full.names = TRUE)

daten <- read_csv2(dateien, id = "Jahr",
                   locale = locale(encoding = "iso-8859-1")) |>
  filter(Str_Kl == "A") |>
  mutate(jahr = as.numeric(str_extract(Jahr, "\\d+"))) |>
  filter(DTV_Kfz_MobisSo_Q != 'NA' & DTV_SV_MobisSo_Q != 'NA')

daten_sv <- daten |>
  group_by(jahr) |>
  summarise(
    sv      = sum(DTV_SV_MobisSo_Q),
    gesamt  = sum(DTV_Kfz_MobisSo_Q),
    anteil  = (sv / gesamt) * 100
  )
```

Diesen nun angepassten Datensatz können wir nun ähnlich wie in #sec-50stunde verwenden, um mit `ggplot` ein Balkendiagramm zu erstellen. Damit für jedes Jahr ein Balken erzeugt wirkt, setzen wir manuell die Skala auf der X-Achse mit `scale_x_continuous`.

```
ggplot(data = daten_sv, aes(x = jahr, y = anteil)) +
  geom_bar(stat='identity', fill = "blue") +
  labs(title ="Anteil␣des␣Schwerverkehrs␣auf␣Autobahnen␣von␣2003␣bis␣2023"
      ,
       x = "Jahr",
       y = "Anteil␣am␣gesamten␣Verkehrsaufkommen␣in␣Prozent") +
  scale_x_continuous(breaks=seq(2003, 2023, 5)) +
 theme_light()
```


Anteil des Schwerverkehrs auf Autobahnen von 2003 bis 2023
Anteil am gesamten Verkehrsaufkommen in Prozent
0
5
10
15
2003
2008
2013
2018
2023
Jahr