Auswertung von Verkehrsdaten

1 Verkehrsdatenanalyse

In diesem Baustein werden Aspekte der Verkehrsdatenanalyse wie Tagesganglinien, Zählstellendatenauswertung in Kartendarstellung und die Bestimmung der 50. Stunde behandelt.

1.1 Voraussetzungen

Zum erfolgreichen Bearbeiten dieses Anwendungsbausteins benötigen sie die Inhalte des Methodenbausteins Grundlagen der Statistik [Referenz] und des Werkzeugbausteins Datenmanagements [Referenz]

1.2 Lernziele

Ziel dieses Bausteins ist es typische Inhalte von verkehrsrelevanten Fragestellungen in grafischer Form mithilfe der Programmiersprache R darzustellen. Dabei lernen sie wo deutschsprachige Verkehrsdaten gefunden werden können, wie diese eingelesen werden und welche verschiedenen Darstellungsformen sich für bestimmte Datentypen eignen.

1.3 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 Autobahnverlaufs in die Deutschlandkarte.
3. Die Daten der Zählstellen in die Deutschlandkarte plotten.

1.3.1 Deutschlandkarte

Wie in [Videoquelle] benötigen wir für die Erstellung einer Deutschlandkarte die Pakete `giscoR` (https://ropengov.github.io/giscoR/), `sf` (https://r-spatial.github.io/sf/) und `ggplot2` (https://ggplot2.tidyverse.org/).

Mithilfe des Pakets `giscoR` werden geografische Informationen abgerufen, die mithilfe von `sf` in `ggplot2` dargestellt werden können.

Mit dem Befehl `gisco_get_nuts()` werden NUTS-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:

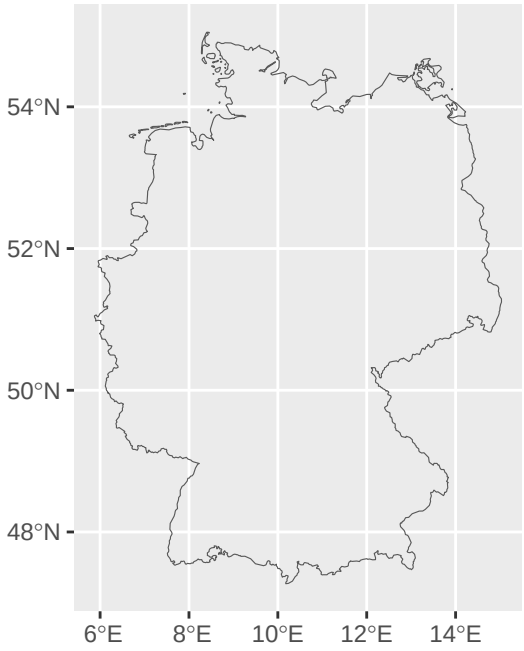
1. `country =` gibt an um welches Land es sich handelt.
2. `nuts_level =` gibt Ebene der NUTS an. (0 für Staat, 1 für Bundesländer, 2 für Regierungsbezirke)
3. `resolution =` bestimmt die Auflösung.

Für zusätzliche Informationen ist es hilfreich in die Dokumentation des Pakets zu schauen.

Weitere Details zu den NUTS(Nomenclature of territorial units for statistics)-Regionen können auf der Seite des Statistischen Amt der Europäischen Union gefunden werden.

Zur ersten Überprüfung ob dieser Schritt entsprechend unserer Erwartung funktioniert hat, können wir mithilfe des Datensatzes bereits mit `ggplot2` plotten. Dafür leiten wir die erstellten *simple features* an `ggplot2` weiter und visualisieren diese mit der Funktion `geom_sf()`. Diese Funktion arbeitet nach dem bekannten Muster aus Kapitel [Referenz]. Als zusätzliche Option geben wir mit `fill = NA` an, dass die Deutschlandkarte nicht farblich eingefärbt werden soll.

```
ggplot() +
  geom_sf(data = d_de, fill = NA, size = 0.5)
```



Wir sehen, dass dies eine funktionierende Deutschlandkarte darstellt. In den weiteren Schritten schauen wir uns an, wie wir weitere Informationen in diese Karte hinzufügen.

1.3.2 Autobahnverlauf

Wie in [Videoquelle] benötigen wir für die Erstellung des Autobahnverlaufs das Paket `osmdata`https://cran.r-project.org/web/packages/osmdata/vignettes/osmdata.html.Dieses Paket verwendet Daten auf Grundlage von *Open Street Map*.

Um nur die Daten aus Deutschland zu verwenden, müssen wir zuerst eine Anfrage (engl. *query*) mit der *Bounding Box* Deutschland erstellen. Die Funktion `opq()` erstellt die Anfrage und die Funktion `getbb()` legt den Koordinatenbereich der *Bounding Box* fest. Wir interessieren uns für `"Deutschland"` und müssen noch spezifieren, dass wir uns für das Land als Merkmal (`featuretype = "country"`) interessieren. Diese Anfrage speichern wir als Objekt zur späteren Verwendung ab.

```
q <- opq(bbox = getbb("Deutschland", featuretype = "country"), timeout = 600)
```

Nun können wir unserer Anfrage Objekte mit der Funktion `add_osm_feature()` hinzufügen. Mit den Spezifikationen `key` für die Kategorie und `value` für die genauen Objekte werden. In unserem Fall interessieren uns nur die Autobahnen: Ein Blick in die Dokumentation von Open Street Map zeigt uns, dass wir `key = "highway"` und `value = "motorway"` benötigen. Dies leiten wir nun an die Funktion `osmdata_sf()` weiter, damit unsere Anfrage im *simple*

feature-Format weiterverarbeitet wird. Wir verwenden nur die `osm_lines` und nicht die anderen Objekte wie `osm_points` und `osm_polygons`. Diese rohen Datensatz speichern wir als `d_bab_raw` .

```
  d_bab_raw <- (add_osm_feature(q, key = "highway", value = "motorway") |> osmdata_sf())$osm
```

Diese Daten müssen nun noch weiter aufbereitet werden. Dabei handelt es sich um den schwierigsten Schritt in diesem Baustein: Ein Blick in den rohen Datensatz zeigt, dass dort viele für unseren Fall uninteressanten Variablen und einige nicht vorhandene Werte vorliegen. Außerdem werden die Autobahnen über die deutschen Grenzen hinaus dargestellt, wir interessieren uns aber momentan nur für die Werte in Deutschland.

Um nur den Autobahnverlauf in Deutschland dazustellen benötigen wir einen Filter mit `filter()`. Als Element des Filters verwenden wir die Funktion `st_contains()` aus dem Paket `sf` und fügen unsere Datensätze `d_de` und `d_bab_raw` ein, sodass nur die Werte aus `d_bab_raw` übernommen werden, die sich in `d_de` befinden. Zusätzlich geben wir mit `sparse = FALSE` an, dass Mit `drop_na(ref)` werden alle Reihen exkludiert, die einen nicht vorhandenen (*NA* englisch für *not available*) Wert für die Variable `ref` besitzen. Die Variable `ref` stellt die Bezeichnung der jeweiligen Autobahn da. Wir wollen also alle Objekte, die nicht zugeordnet werden entfernen.Mit `group_by(ref)` und `summarise()` wird der Datensatz nach der `ref`-Variable gruppiert und alle Reihen mit demselben Wert werden zusammengefasst. Zuletzt verwenden wir `st_simplify()` um den Autobahnverlauf zu vereinfachen und nicht jeden Eckpunkt darzustellen.Mit `dTolerance =` stellen wir den Toleranzparameter für unsere Vereinfachung in Metern ein (in unserem Fall beträgt dies 100 Meter). Diesen Datensatz speichern wir als `d_bab` ab.

```
q <- opq(bbox = getbb("Deutschland", featuretype = "country"), timeout = 600)
d_bab_raw <- (add_osm_feature(q, key = "highway", value = "motorway") |> osmdata_sf())$osm
d_bab <- d_bab_raw |>
  filter(st_contains(d_de, d_bab_raw, sparse = FALSE)[1,]) |>
  drop_na(ref) |>
  group_by(ref) |>
  summarise() |>
  st_simplify(dTolerance = 100)
save(d_bab, d_bab_raw, file = "data/bab.RData")
```

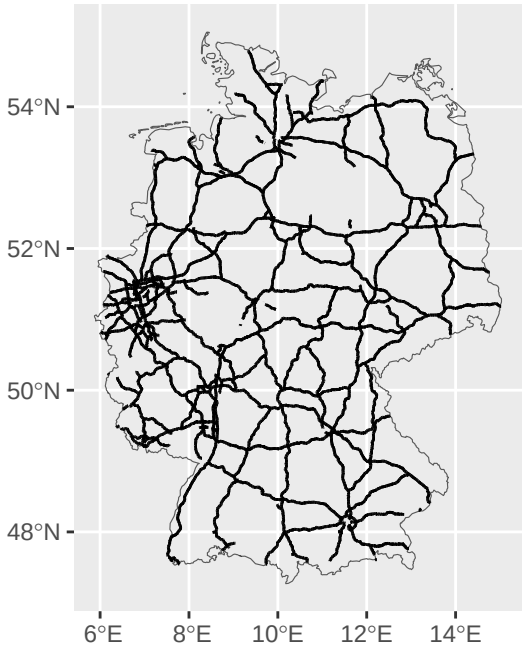
> **Rechenleistung und Zeit einsparen**
>
> Um zu verhindern, dass die durchaus lange Bearbeitungsdauer der 'add_osm_features'-Funktion jedes Mal beim Rendern des Dokuments durchgeführt wird, lohnt es sich die rohen RData-Dateien abzuspeichern und mithilfe einer if-Abfrage zu schauen, ob diese Dateien bereits vorhanden sind und dann zu laden. Beispielhaft könnte dies mit einer

if-Abfrage so aussehen:

```
if (!file.exists("data/d_bab.RData")) {
  q <- opq(bbox = getbb("Deutschland", featuretype = "country"), timeout = 600)
  d_bab_raw <- (add_osm_feature(q, key = "highway", value = "motorway") |> osmdata_sf())$os
  d_bab <- d_bab_raw |>
    filter(st_contains(d_de, d_bab_raw, sparse = FALSE)[1,]) |>
    drop_na(ref) |>
    group_by(ref) |>
    summarise() |>
    st_simplify(dTolerance = 100)
  save(d_bab, d_bab_raw, file = "data/bab.RData")
} else {
  load(file = "data/d_bab.RData")
}
```

Wenn man diese Daten nun mit der bereits erstellten Deutschlandkarte kombiniert, erhält man diese Darstellung:

```
ggplot() +
  geom_sf(data = d_de, fill = NA, size = 0.5) +
  geom_sf(data = d_bab,  size = 0.35, show.legend = FALSE)
```



1.3.3 Zählstellendaten

Zuguterletzt fügen wir die tatsächlichen Verkehrsdaten zu unserer Darstellung hinzu. Wir verwenden hier die Daten, die Bundesanstalt für Straßenwesen (BAST) online zur Verfügung stellt. Die heruntergeladene CSV-Datei importieren wir mithilfe der `read_csv2()`-Funktion des Pakets `readr` aus der Paketsammlung `tidyverse`.

```
d_Jawe <- read_csv2("data/Jawe2022.csv", locale = locale(encoding = 'iso-8859-1'))
```

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. Danach müssen wir den Datensatz aufbereiten (siehe [Referenz]) und für unseren Zweck relevante Daten filtern (siehe [Referenz]). Dazu benötigen wir die ebenfalls online verfügbare Datensatzbeschreibung der BAST.

Dort sehen wir, dass die Variable `Str_Kl` die Straßenklasse beschreibt. Diese Variable hat die Ausprägungen "A" für Autobahnen und "B" für Bundesstraßen. In unserem Fall interessieren uns nur die Daten für Autobahnen, also die Ausprägung "A". Dies können wir mit `filter(Str_Kl == "A")` erreichen.

Da dieser Datensatz ebenfalls viele nicht vorhandene,sogenannte *NA*(englisch *not available*)-Werte beinhaltet, können wir diese ebenfalls herausfiltern. Dies geschieht zum Beispiel mit `filter(DTV_Kfz_MobisSo_Q != "NA" )`. Diese beiden Filter können mit dem Operator "&" verknüpft werden. Eine elegantere Lösung ist die Verwendung der `drop_na()`-Funktion aus dem Paket `tidyr` des `tidyverse`.

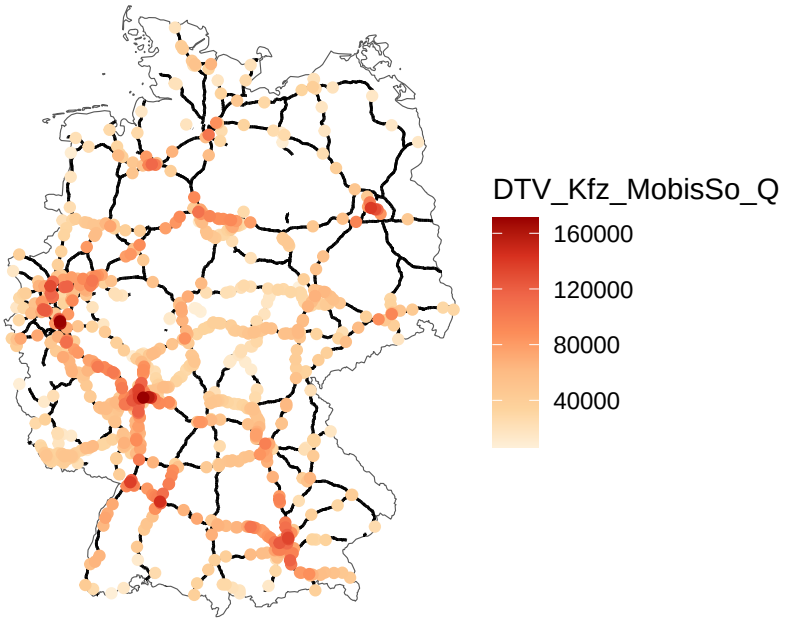
Eine weitere Maßnahme für unseren speziellen Fall ist die Sortierung des Datensatzes: `ggplot2` geht den Datensatz von oben nach unten durch um Datenpunkte zu plotten. Wir wollen aber, dass besonders befahrene Abschnitte vor den anderen Punkte liegen und somit gut sichtbar auf der Karte sind. Dazu benutzen wir die `arrange()`-Funktion und sortieren den Datensatz aufsteigend nach der Variable `DTV_Kfz_MobisSo_Q`.

```
d_JaweNA <- d_Jawe |>
  filter(Str_Kl == "A") |>
  drop_na(DTV_Kfz_MobisSo_Q) |>
  arrange(DTV_Kfz_MobisSo_Q)
```

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.

Um die Darstellung zu verschönern, verwenden wir einige zusätzliche Optionen von `ggplot2`. `scale_color_distiller` ist Teil der Farbpalette "brewer", entwickelt von Cynthia Brewer. Mit `palette = 8` benutzen wir in unserem Fall die 8. Palette und `direction = 1` invertiert die Skala, damit hohe Werte dunkler dargestellt werden und niedrige Werte heller. 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_bab,  size = 0.35, show.legend = FALSE) +
  geom_point(data = d_JaweNA, mapping = aes(x = Koor_WGS84_E, y = Koor_WGS84_N, color = DTV_
  scale_color_distiller(palette = 8 , direction = 1) +
  theme_void()
```



1.4 Tagesganglinien

Als weitere wichtige Größe in der Verkehrsdatenanalyse schauen wir uns die 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 bei der A42 in Wanne-Eickel. Diese Daten müssen wir zuerst einlesen:

```
ZST5116 <- read_csv2("data/zst5116_2022.csv", locale = locale(encoding = 'iso-8859-1'))
```

Auch hier hilft es, sich die Datensatzbeschreibung der BAST anzuschauen.

1.4.1 Tagesganglinien für einzelne Wochentage

Wir wollen nun die Tagesganglinien der ausgewählten Zählstelle für jeden Wochentag erstellen. Wir brauchen also Mittelwerte für jede Stunde jedes Wochentages. Dazu benötigen wir das Paket `dplyr` aus der Paketsammlung `tidyverse`.

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 =`).

```
ZST5116_DTV <- ZST5116 |>
  group_by(Stunde, Wotag) |>
  summarise(avg_KFZ_R1 = mean(KFZ_R1),avg_KFZ_R2 = mean(KFZ_R2))

ZST5116_DTV$Wochentag <- factor(ZST5116_DTV$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, wie es bei dem Namen Tagesganglinie vermuten lässt, 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* weiter"vererbt". 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.

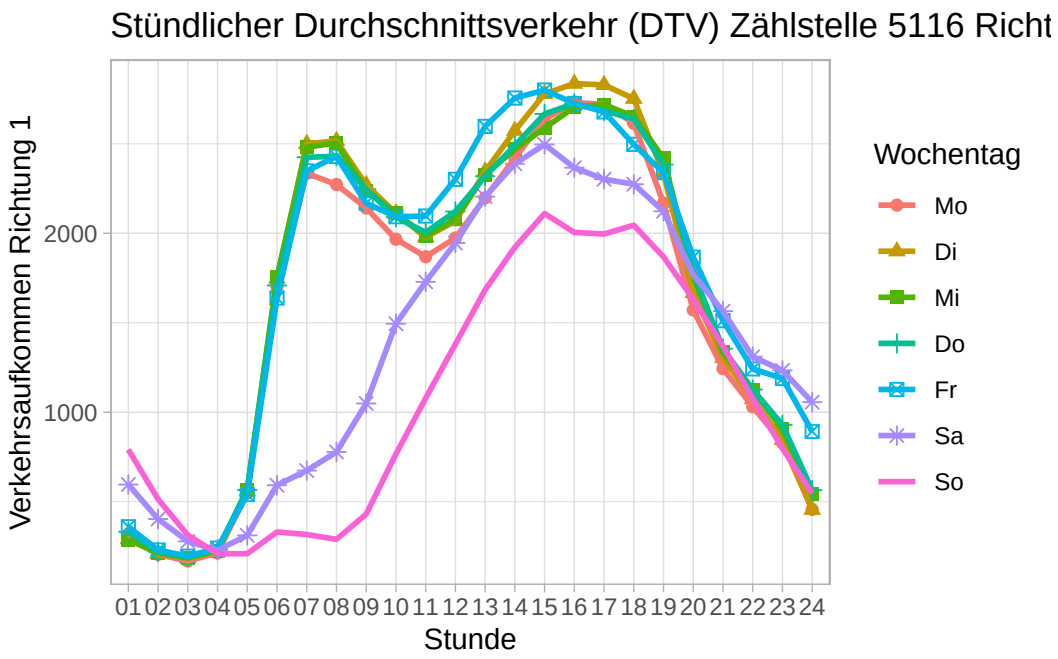
Als Zusatzoption verwenden wir mit `scale_color_viridis_d` die Viridisfarbpalette, die so gestaltet wurde, dass Personen mit häufigen Formen von Farbenblindheit diese problemlos wahrnehmen können.

Mit `theme_light()` verwenden wir ein vorgefertigtes Thema (https://ggplot2.tidyverse.org/reference/ggtheme.html), um die Darstellungsform aller nicht Daten-Elemente zu steuern. Auch hier handelt es sich natürlich um eine persönliche Präferenz und andere Themen können ebenfalls gewählt werden und gute Grafiken erzeugen.

```
# Erstellen des Streudiagramms für den stündlichen DTV Zählstelle 5116 Richtung 1
ggplot(data = ZST5116_DTV, aes(x = Stunde, y = avg_KFZ_R1 ,group = Wochentag ,color = factor
  geom_point(size = 2) +
  geom_line(linewidth = 1) +
  labs(title = "Stündlicher Durchschnittsverkehr (DTV) Zählstelle 5116 Richtung 1",
       x = "Stunde",
       y = "Verkehrsaufkommen Richtung 1",
       color = "Wochentag",
      shape = "Wochentag") +
  theme_light()
```

```
Warning: The shape palette can deal with a maximum of 6 discrete values because more
than 6 becomes difficult to discriminate
i you have requested 7 values. Consider specifying shapes manually if you need
  that many have them.
```

```
Warning: Removed 24 rows containing missing values or values outside the scale range
(`geom_point()`).
```



1.5 50. Stunde

Als weiteres relevante Maß schauen wir uns das Verkehrsaufkommen in der 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.

```
ZST5116_50 <- 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 Freitag (Wotag 5), den 11. Februar (Datum 220211) in der 14. Stunde handelt und in diesem zeitraum 3173 KFZ erfasst wurden.

```
kable(ZST5116_50)
```

Datum	Wotag	Stunde	KFZ_R1
220211	5	14	3173

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 `1:nrow()`).

```
# Erstellen des Streudiagramms für den stündlichen DTV Zählstelle 5116 Richtung 1
ZST5116_top <- ZST5116 |>
  arrange(desc(KFZ_R1))

ZST5116_top$Nummer = 1:nrow(ZST5116_top)
```

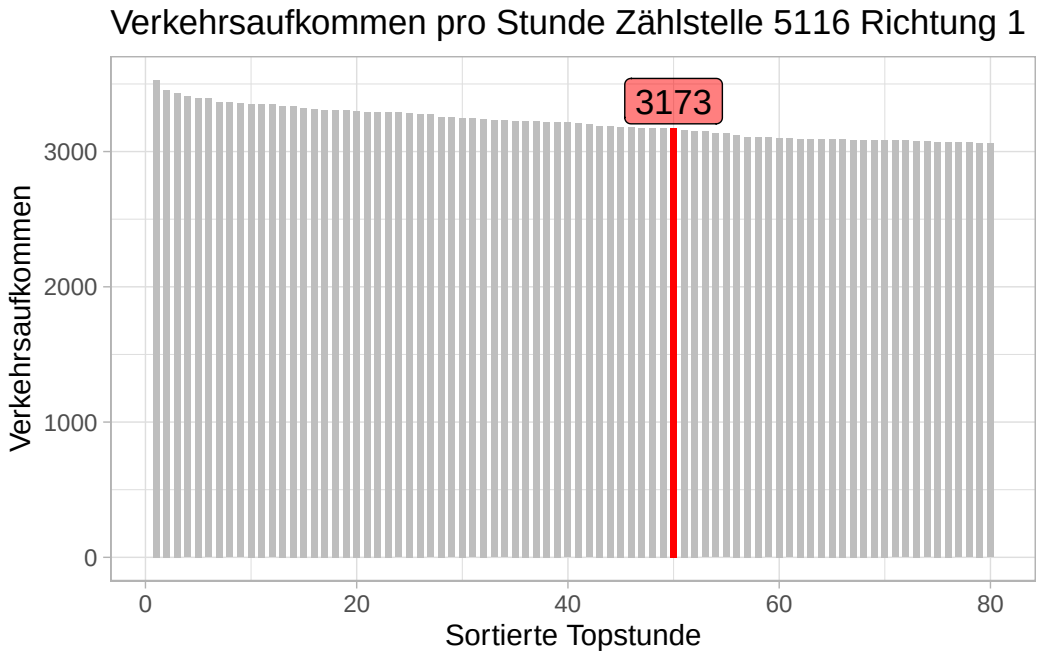
Nun können wir unsere Säulendiagramme plotten: Um nicht alle (in unserem Fall 8760) Beobachtungen zu plotten, wird der zu plottende Datensatz mithilfe von dem Zusatz `[1:80,]` bei der Festlegung des Datensatzes auf die ersten 80 Beobachtungen reduziert. Es bietet sich an , die 50. Stunde farblich hervorzuheben 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. Durch das Einfügen von `values = c("black", "red")` bestimmen wir, dass für den Fall dass der Operator stimmt (also die 50. Stunde), die Farbe rot verwendet wird, andernfalls die Farbe schwarz. Der Zusatz `guide = "none"` sorgt dafür, dass diese Logikabfrage nicht in der Legende aufgeführt wird.

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(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 Positierung wird analog `nudge_x`verwendet.
- `size` bestimmt die Größe des Labels.
- `alpha` stellt die Transparenz ein.

```r
ggplot(data = ZST5116_top[1:80, ], aes(x = Nummer,y = KFZ_R1, fill = Nummer == 50, width = .
  geom_bar(stat = "identity") +
  labs(title = "Verkehrsaufkommen pro Stunde Zählstelle 5116 Richtung 1",
       x = "Sortierte Topstunde",
       y = "Verkehrsaufkommen",
       legend = "Wochentag") +
  scale_fill_manual(values = c("grey", "red"), guide = "none") +
  geom_label(
    data = filter(ZST5116_top,Nummer == 50),
    mapping = aes(label = KFZ_R1),
    nudge_y = 200, size = 4.5, alpha = 0.5
  ) +
  theme_light()
```



1.6 Weitere Anwendungsbeispiele

In diesem Abschnitt befinden sich weitere anwendungsbezogene Beispiele und Vorschläge zu weiteren Untersuchungen der Zählstellendaten.

1.6.1 Mittlere Werktage

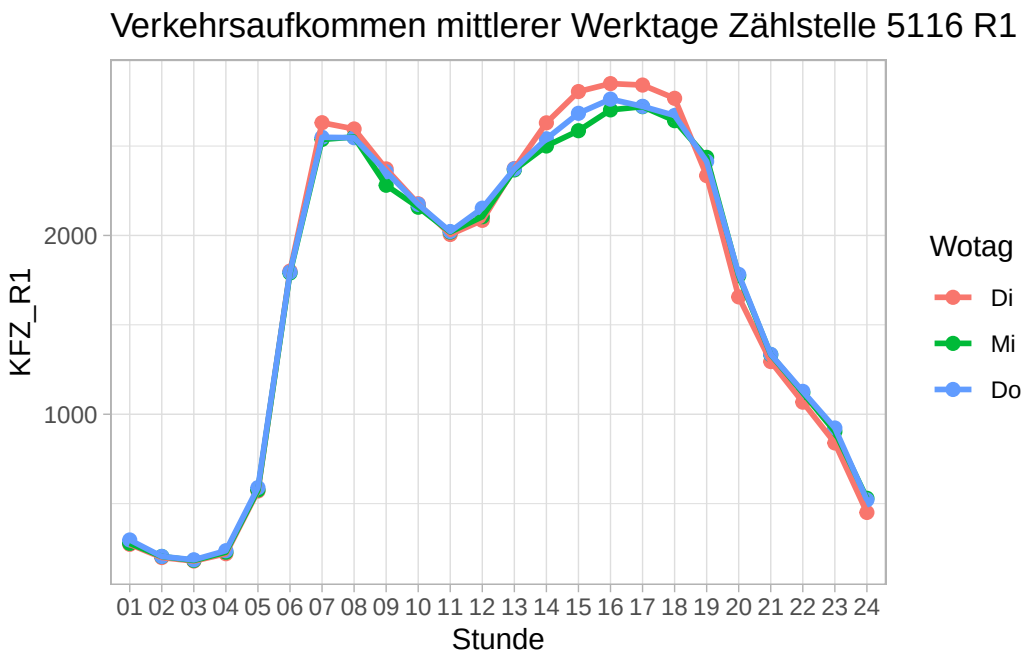
Häufig können in der Analyse sogenannte mittlere Werktage verwendet werden, um der Varianz vom Beginn und Ende der Woche zu entgehen. In diesem Fall interessieren uns also die Wochentage Dienstag, Mittwoch und Donnerstag an normalen Werktage außerhalb der Schulferien. Ein Blick in die Datensatzbeschreibung der BAST zeigt uns, dass diese Unterteilung im Datensatz bereits vorliegt. Die Variable `Fahrtzw` besitzt die Ausprägungen w für Werktags, u für Urlaubswerktage und s für Sonn- und Feiertage. Somit müssen wir in unseren Filter lediglich die Abfrage `Fahrtzw == "w"` einfügen.

```
ZST5116$Wotag <- factor(ZST5116$Wotag,
 levels = c(1,2,3,4,5,6,7),
 labels = c("Mo","Di","Mi","Do","Fr","Sa","So"))


data_werktag <- ZST5116 |>
  filter(Fahrtzw == "w", Wotag %in% c("Di", "Mi", "Do"))

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

```
ggplot(data = data_werktag_mean, aes(x = Stunde, y = avg_KFZ_R1, group = Wotag, color = Wotag
  geom_point(size = 2) +
  geom_line(linewidth = 1) +
  labs(title = "Verkehrsaufkommen mittlerer Werktage Zählstelle 5116 R1",
       x = "Stunde",
       y = "KFZ_R1",
       color = "Wotag") +
  theme_light()
```



1.6.2 Mehrere Zählstellen vergleichen

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 Section 1.4. Wir importieren hier nun die Datensätze der Zählstellen 5113,5116,5125 und 5128.

```
ZST5113 <- read_csv2("E:/Arbeit/bcd-anwendung/data/zst5113_2022.csv", locale = locale(encodi
ZST5116 <- read_csv2("E:/Arbeit/bcd-anwendung/data/zst5116_2022.csv", locale = locale(encodi
ZST5125 <- read_csv2("E:/Arbeit/bcd-anwendung/data/zst5125_2022.csv", locale = locale(encodi
ZST5128 <- read_csv2("E:/Arbeit/bcd-anwendung/data/zst5128_2022.csv", locale = locale(encodi
```

> **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 [Referenz zu anderem Kapitel] https://r4ds.hadley.nz/functions.html

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. Analog zu Section 1.4 wandeln wir die Variable `Zählstelle` in einen Faktor mit den Namen als Ebenen um. Zusätzlich benennen wir auch die Wochentage wie bereits getan erneut um.

```
all_data <- bind_rows(list(ZST5113,ZST5116,ZST5125,ZST5128), .id = "Zählstelle")

all_data$Zählstelle <- factor(all_data$Zählstelle, levels = c("1", "2", "3", "4"), labels = c

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

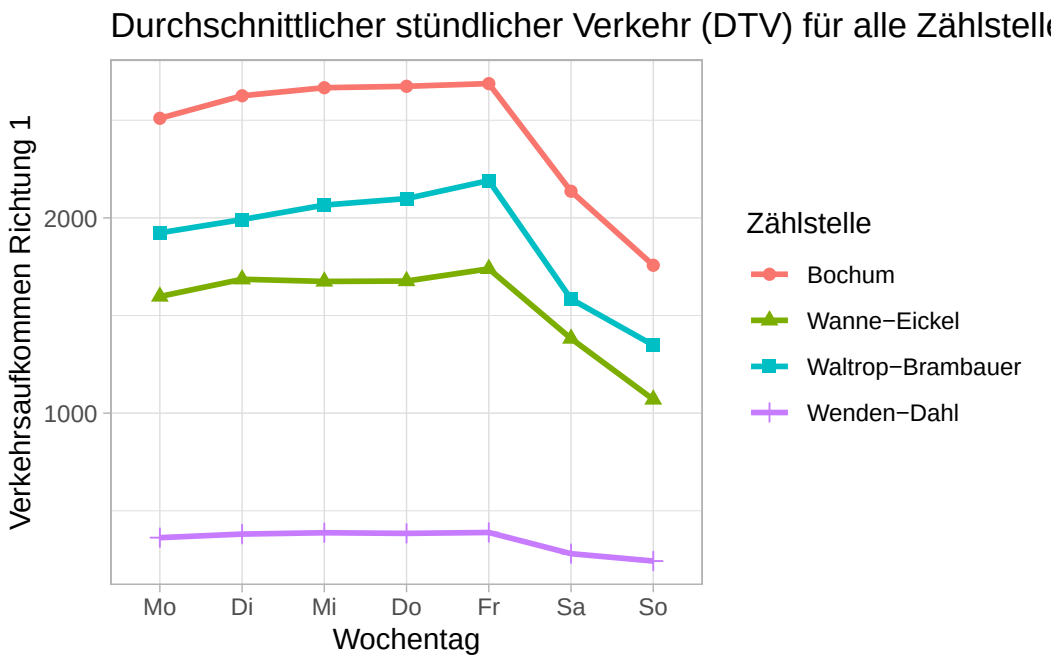
1.6.2.1 Über alle Tage

Wir wollen nun die Daten der Zählstellen für alle Tage gemittelt vergleichen. Dazu verwenden wir unseren kombinierten Datensatz und gruppieren diesen mit `group_by` nach der Identifikationsvariable der Zählstelle (`Zählstelle`) und dem Wochentag (`Wotag`).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.

```
data_wochentag <- all_data |>
  group_by(Zählstelle, Wotag) |>
  summarise(avg_KFZ_R1 = mean(KFZ_R1),avg_KFZ_R2 = mean(KFZ_R2))
```

Mit diesen Daten können wir nun bereits mit `ggplot2` eine Grafik erstellen. Dazu verwenden wir grundsätzlich die verwendeten Funktionen aus Section 1.4. Dieses Mal gruppieren und färben wir den Plot aber nach der Variable `Zählstelle` anstelle von `Wotag`.

```
ggplot(data = data_wochentag, aes(x = Wotag, y = avg_KFZ_R1, group = Zählstelle, color = Zähl
  geom_point(size = 2) +
  geom_line(linewidth = 1) +
  labs(title = "Durchschnittlicher stündlicher Verkehr (DTV) für alle Zählstellen",
        x = "Wochentag",
        y = "Verkehrsaufkommen Richtung 1",
        color = "Zählstelle",
        shape = "Zählstelle") +
  theme_light()
```



1.6.2.2 Einen Wochentag im Detail

Nun wollen wir uns einen Wochentag im Detail zwischen den verschiedenen Zählstellen vergleichen. Erneut filtern wir mit `filter()` unsere Daten so, dass in unserem Fall nur noch Dienstage übrig bleiben (`Wotag == "Di"`) und speichern diesen Datensatz. Zur weiteren Verarbeitung gruppieren wir diesen Datensatz nach Zählstellen und Stunde (`group_by(Zählstelle, Stunde)`) und erstellen mit `summarise()` für jede Stunde jeder Zählstelle eine neue Reihe und erzeugen den Mittelwert mit `mean()`.

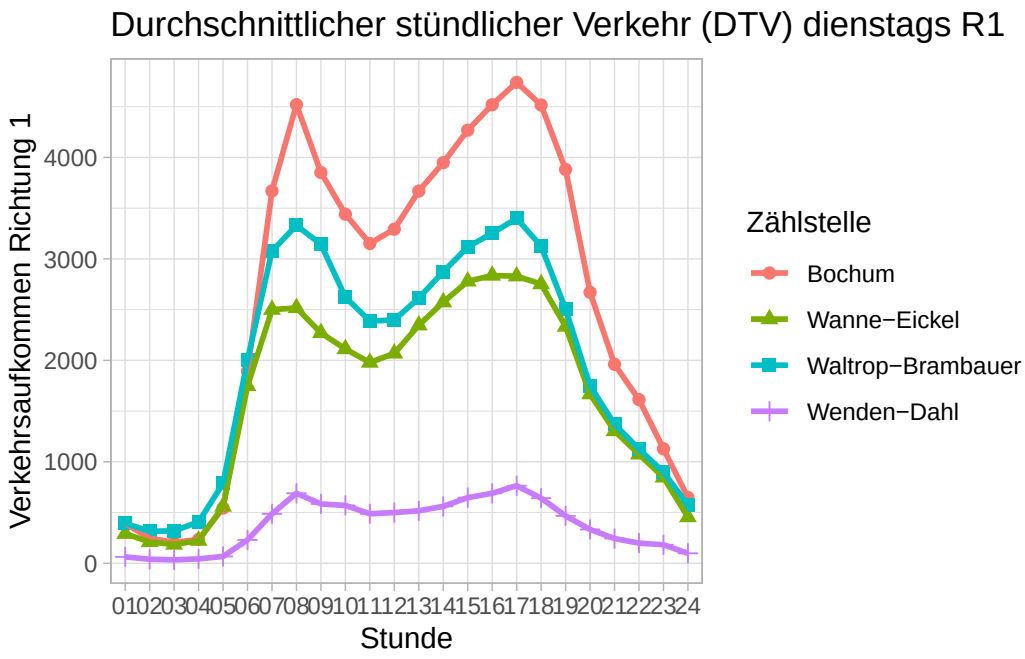
TO-DO: Angabe der prozentualen stündlichen Anteile (+ Tabelle)

```
data_dienstag <- all_data |>
  filter(Wotag == "Di")


data_dienstag_mean <- data_dienstag |>
  group_by(Zählstelle, Stunde) |>
  summarise(avg_KFZ_R1 = mean(KFZ_R1))
```

Der erstellte Plot ist nahezu identisch mit dem aus Section 1.6.2.1, nur werden auf der x-Achse jetzt die einzelnen Stunden des Dienstags aufgeführt.

```
ggplot(data = data_dienstag_mean, aes(x = Stunde, y = avg_KFZ_R1, group = Zählstelle, color =
  geom_point(size = 2) +
  geom_line(linewidth = 1) +
  labs(title = "Durchschnittlicher stündlicher Verkehr (DTV) dienstags R1",
       x = "Stunde",
       y = "Verkehrsaufkommen Richtung 1",
       color = "Zählstelle",
       shape = "Zählstelle") +
  theme_light()
```



1.6.2.3 Schwer und Leichtverkehr

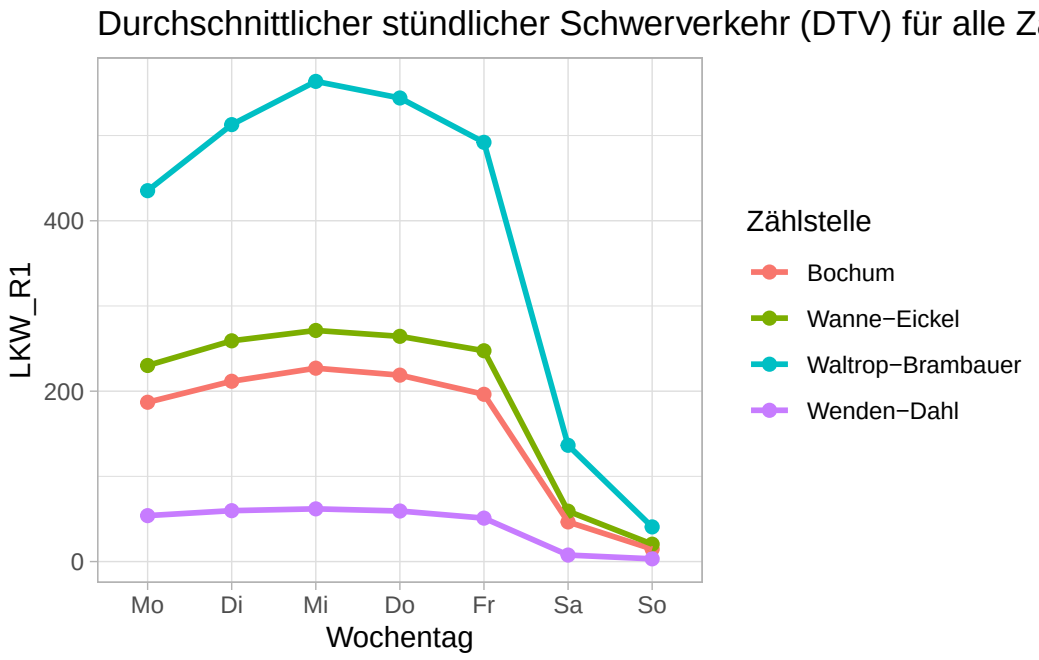
TO-DO: Erläuterungen

```
data_schwer <- all_data |>
  group_by(Zählstelle, Wotag) |>
  summarise(avg_LKW_R1 = mean(Lkw_R1),avg_LKW_R2 = mean(Lkw_R2))
```

```
`summarise()` has grouped output by 'Zählstelle'. You can override using the
`.groups` argument.
```

```r
# Erstellung des Streudiagrammes Richtung 1.
ggplot(data = data_schwer, aes(x = Wotag, y = avg_LKW_R1, group = Zählstelle, color = Zählste
  geom_point(size = 2) +
  geom_line(size = 1) +
  labs(title = "Durchschnittlicher stündlicher Schwerverkehr (DTV) für alle Zählstellen R1",
       x = "Wochentag",
       y = "LKW_R1",
       color = "Zählstelle") +
  theme_light()
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



1.6.3 Schwerverkehr-Anteile über die Jahre

TO-DO: Erläuterungen

```r
# Notiz für Matthias: Das hier ist eine frühe, nicht sonderlich gute Lösung, die ich aber noc

list_of_files <- list.files(path = "data_jw",
                            recursive = TRUE,
```

```
                              pattern = "\\.csv$",
                              full.names = TRUE)

df <- readr::read_csv2(list_of_files, id = "Jahr", locale = locale(encoding = 'iso-8859-1'))
```

```
i Using "','" as decimal and "'.'" as grouping mark. Use `read_delim()` for more control.

New names:
* `` -> `...255`

Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)

Rows: 24056 Columns: 256
-- Column specification --------------------------------------------------------
Delimiter: ";"
chr  (37): DZ_Nr, DZ_Name, Land_Code, Str_Kl, Str_Zus, Erf_Art, Fernziel_Ri1...
dbl (134): TK_Nr, Land_Nr, Str_Nr, Anz_Fs_Q, Betriebs_km, vT_MobisSo, vT_W, ...
num  (81): DTV_Kfz_MobisSo_Q, DTV_Kfz_MobisSo_Ri1, DTV_Kfz_MobisSo_Ri2, DTV_...
lgl   (3): TGMax2_SoFei_Ri1, TGMax2_SoFei_Ri2, ...255

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
df$Jahr <- str_remove_all(df$Jahr, "[data_jw/Jawe]")
df$Jahr <- str_remove_all(df$Jahr, "[.csv]")
df$Jahr <- as.numeric(df$Jahr)

df_NA <- df |>
  filter(DTV_Kfz_MobisSo_Q != 'NA' & DTV_SV_MobisSo_Q != 'NA')

df_schwer <- df_NA |>
  group_by(Jahr) |>
  summarise(SV = sum(DTV_SV_MobisSo_Q),Alle = sum(DTV_Kfz_MobisSo_Q),Anteil = (SV/Alle)*100)
```

```
ggplot(data = df_schwer, aes(x = Jahr, y = Anteil)) +
  geom_bar(stat='identity', fill = "blue") +
  labs(x = "Jahr", y = "Anteil am gesamten Verkehrsaufkommen") +
```

```
 ggtitle("Anteil des Schwerverkehrs von 2003 bis 2018 in 1000") +
 scale_x_continuous(breaks=seq(2003,2018,1)) +
theme_light()
```