Anwendungsbaustein Unfalltypenkarten

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 Unfalltypenkarte“. https://github.com/bausteine-der-datenanalyse/a-unfalltypenkarte.

BibTeX-Vorlage

```
@misc{BCD-a-Unfalltypenkarte-2026,
 title={Bausteine Computergestützter Datenanalyse.
 Anwendungsbaustein Unfalltypenkarte},
 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-unfalltypenkarte}}
```

Voraussetzungen

Zum erfolgreichen Bearbeiten dieses Anwendungsbausteins benötigen Sie die Inhalte des Methodenbausteins Grundlagen der Statistik und des Werkzeugbausteins Datenanalyse in R. Außerdem sollten die Grundlagen der Unfallanalyse aus dem Merkblatt zur Örtlichen Unfalluntersuchung in Unfallkommissionen (M Uko) der FGSV bekannt sein.

Lernziele

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

Verwendete Datensätze

Die statistischen Ämter des Bundes und der Länder erheben und veröffentlichen mit der Straßenverkehrsunfallstatistik die Verkehrsunfälle mit Personenschaden oder Sachschaden in Deutschland. Im Unfallatlas werden Unfälle mit Personenschaden geführt.

Daten des Straßennetzes des Bundesinformationssystem Straße, Stichwortsuche: BASt + Bundesfernstraßennetz

1 Aufgaben zu Unfalltypenkarten und Unfallhäufungsstellen

Versuchen Sie bei den nachfolgenden Aufgaben zu Unfalltypenkarten weitestgehend die Vorgaben der Forschungsgesellschaft für Straßen- und Verkehrswesen (FGSV) aus dem "Merkblatt zur Örtlichen Unfalluntersuchung in Unfallkommissionen" (M Uko) umzusetzen.

Übungsaufgabe 1.1 (Unfalltypenkarten)**.** Erstellen Sie nach den Vorgaben des M Uko eine Unfalltypenkarte für - eine ausgewählte Autobahn und - alle Autobahnen in Gesamtdeutschland

mit

- `ggplot` und
- `leaflet`

für Unfälle in einem beliebigem Jahr. Welche Darstellungen halten Sie für geeignet?

Übungsaufgabe 1.2 (Unfallhäufungsstellen)**.** Stellen Sie nach den Vorgaben des M Uko Unfallhäufungsstellen in einer Dreijahreskarte in einer beliebigen Stadt dar.

Erstellen Sie eine interaktive `leaflet`-Karte der Unfälle in Bochum, beispielsweise mit `shiny` oder `crosstalk`. Die Interaktivität soll folgendermaßen gestaltet werden:

Übungsaufgabe 1.3 (optional: Interaktive `leaflet`-Karte mit Filtern)**.**

- wählbare Unfallkategorien,
- wählbare Unfalltypen,
- wählbare Unfallbeteiligte und
- wählbare Jahre

Zusätzlich soll unter der Karte eine Tabelle mit den gewählten Filtern dargestellt werden.

Optional : Ein Download-Button für die gewählten Filtereinstellungen, der eine CSV-Datei erzeugt und herunterlädt.

2 Musterlösung Unfalltypenkarte mit Erläuterungen

Daten der Bundesinformationssystem Straße (BISStra) für Straßennetz: Stichwort "Bundesfernstraßennetz BASt"

Daten für Unfälle vom Unfallatlas: https://unfallatlas.statistikportal.de/

2.1 Übungsaufgabe 1.1: Unfalltypenkarte

Für diese Aufgabe betrachten wir die Unfälle aus einem Jahr. Wir wählen das Kalenderjahr 2024.

2.1.1 Datenaufbereitung

Als erstes wird das Straßennetz der Bundesfernstraßen als Datensatz ins Projekt geladen. Diese Daten bekommen wir von der Bundesanstalt für Straßen- und Verkehrswesen (BASt). 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"-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 Unfalldaten 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.

Als nächstes werden die Daten vom Unfallatlas heruntergeladen und in R geladen. Danach wird der Datensatz mit "st_as_sf" in ein räumliches Punktobjekt umgewandelt. Die Koordinaten stammen aus den Spalten "LINREFX" und "LINREFY". Mit CRS (Coordinate Reference System) wird das Koordinatenreferenzsystem EPSG:25832 definiert. Auch hier entfernen wir potentielle Daten der z- oder m-Dimension.

Den Unfalldaten soll nun aufgrund ihrer Lage in Deutschland der nächstgelegene Autobahnabschnitt und die Entfernung zu diesem hinzugefügt werden. “st_nearest_feature” gibt dabei die Zeilennummer des nächstgelegenen Autobahnabschnitts im Datensatz “d_bfstn” wieder. “st_distance” berechnet die Entfernung vom Unfallpunkt zum nächstgelegenen Autobahnabschnitt, dies kann unter Umständen länger dauern. Außerdem suchen wir mit “d_bfstn$Str_Kennung[abschnitt_id]” aus dem Datensatz “d_bfstn” und der Spalte “Str_Kennung” genau den Eintrag, dessen Zeile durch “abschnitt_id” angegeben wird. Darüber erhalten wir statt einer Nummer eines Autobahnabschnittes den Namen der Autobahn (z.B. A1). Zudem wollen wir nur Unfälle auf Autobahnen berücksichtigen. Dafür filtern wir nach Unfällen, die in einem 20 m Umkreis zur Bestandsachse der Autobahnen verortet sind. Die 20 m wählen wir, um möglichst alle Unfälle auf den Fahrbahnen, auch bei 6- oder 8-streifigen Autobahnen, einzuschließen. Es gibt aber keine Garantie, dass alle Unfälle, die wir nun filtern, tatsächlich auf Autobahnen liegen, zum Beispiel bei Unfällen auf Brücken über Autobahnen. Für mehr Übersichtlichkeit vereinfachen wir unseren Datensatz und lassen uns mit “select” nur bestimmte Spalten anzeigen.

Diese beiden Datensätze speichern wir, damit sie, sofern sie vorhanden sind, nur geladen und nicht jedes Mal neu berechnet werden müssen.

```
if (!file.exists("daten/unfaelle.RData")) {
  # Fernstraßennetz
  curl_download(
    "https://www.bast.de/SharedDocs/Daten-TB/Daten-BISStra.zip?__blob=
       publicationFile&v=5",
    destfile = "daten/Daten-BISStra.zip",
    quiet = FALSE
  )
  unzip("daten/Daten-BISStra.zip", exdir = "daten/geo/")
  file <- list.files(
    "daten/geo",
    pattern = "^BFStr_Netz.*\\.gpkg$",
    full.names = TRUE
  ) |>
    tail(n = 1)
  d_bfstn <- read_sf(file) |>
    filter(Str_Klasse_kurz == "A" & Sk_Achse == "Bestandsachse") |>
    mutate(rownumber = row_number()) |>
    st_zm()

  # Unfälle auf Autobahnen
  d_unfaelle_bab_2024 = read_csv2("daten/Unfallorte2024_LinRef.csv") |>
    st_as_sf(coords = c("LINREFX", "LINREFY"), crs = 25832) |>
    st_zm() |>
      mutate(
        abschnitt_id = st_nearest_feature(geometry, d_bfstn),
        distanz = st_distance(geometry, d_bfstn[abschnitt_id, ],
                              by_element = TRUE),
        Str_Kennung = d_bfstn$Str_Kennung[abschnitt_id]
```

```
      ) |>
    filter(as.double(distanz) <= 20) |>
    select(UTYP1, UKATEGORIE, Str_Kennung, abschnitt_id)

  # Daten sichern
  save(
    d_bfstn,
    d_unfaelle_bab_2024,
    file = "daten/unfaelle.RData"
  )
} else {
  load(file = "daten/unfaelle.RData")
}
```

2.1.2 Grafische Darstellung mit `ggplot`

Für die Darstellung definieren wir uns die Farben der Unfalltypen gemäß des M Uko. (FGSV 2012, 10) Statt weiß nutzen wir allerdings grau für eine bessere Sichtbarkeit auf weißem Grund.

```
farben_utyp <- c(
"1" = "green",
"2" = "yellow",
"3" = "red",
"4" = "grey",
"5" = "blue",
"6" = "orange",
"7" = "black"
)
```

Gemäß des M Uko soll die Unfallschwere an der Größe des Punktes abzulesen sein. (FGSV 2012, 10) Dies ist in der Umsetzung schwierig zu differenzieren, daher nutzen wir verschiedene Formen (15 = Quadrat, 16 = Kreis, 17 = Dreieck).

```
formen_ukat <- c(
  "1" = 15,
  "2" = 16,
  "3" = 17
)
```

2.1.2.1 Grafische Darstellung von Unfällen auf der Beispielautobahn A40 mit ggplot

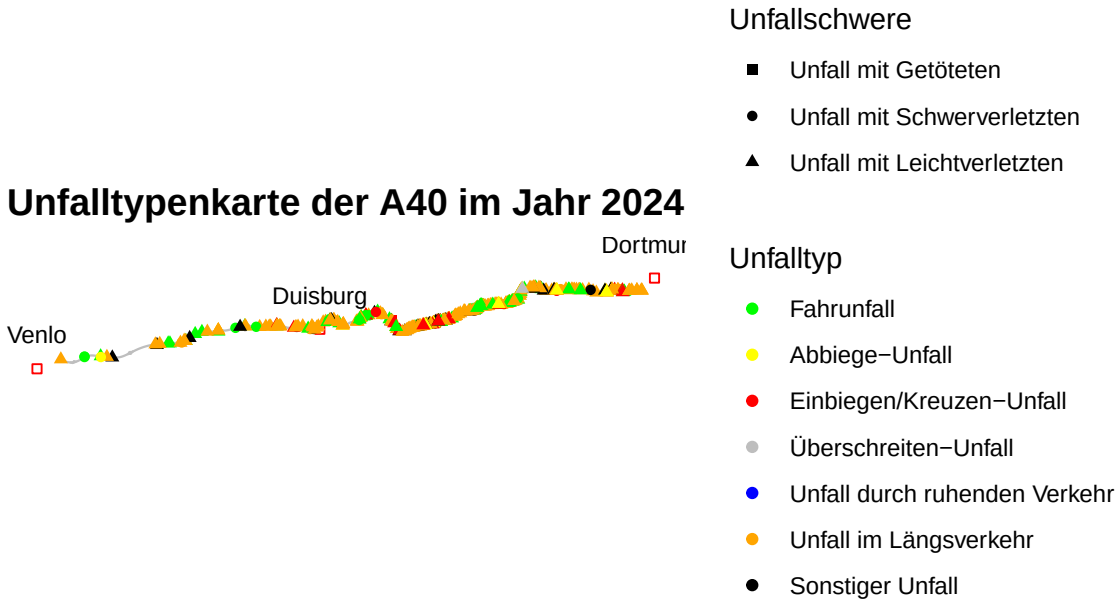
Bei der grafischen Darstellung wollen wir im Hintergrund den Verlauf der A40 sehen, daher filtern wir uns aus dem Datensatz der Bundesautobahnen “d_bfstn” die Autobahn 40. Aus dem Datensatz der Unfälle auf Bundesautobahnen im Jahr 2024 filtern wir alle Unfälle auf der

Beispielautobahn A40. Außerdem wollen wir uns drei Städte entlang der A40 für eine bessere Übersicht darstellen. Dafür suchen wir uns die Koordinaten der Städte aus dem Internet. Diese sind in dem Koordinatenreferenzsystem 4326, daher transformieren ("st_transform") wir sie in das CRS der anderen Daten (25832). In unserem ggplot stellen wir nun den Verlauf der A40 ("data = d_A40"), die Städte ("data = d_staedte") und die Unfälle ("data = d_unfaelle_a40_2024") dar und beschriften die Städte mit "geom_sf_text()". Zudem legen wir mit "scale_colour_manual" den Namen, die Beschriftung und die Farben nach den Unfalltypen fest und mit "scale_shape_manual" den Namen, die Beschriftung und die Form nach Unfallschwere (Unfallkategorie) fest.

```
d_staedte <- st_as_sf(
  tibble::tribble(
    ~ stadt, ~lat, ~lon,
    "Dortmund", 51.5142, 7.4684,
    "Duisburg", 51.4351, 6.7627,
    "Venlo", 51.37, 6.1681
  ),
  coords = c("lon", "lat"),
  crs = 4326
) |>
  st_transform(25832)

ggplot () +
  geom_sf(data = filter(d_bfstn, Str_Kennung == "A40"),
          colour = "grey", size = 0.4) +
  geom_sf(data = d_staedte, colour = "red", shape = 22) +
  geom_sf_text(data = d_staedte, aes(label = stadt),
               nudge_y = 5000, size = 3) +
  geom_sf(
    data = filter(d_unfaelle_bab_2024, Str_Kennung == "A40"),
    mapping = aes(
      group = UTYP1,
      colour = as.character(UTYP1),
      shape = as.character(UKATEGORIE)),
    size = 1.5) +
  scale_colour_manual(name = "Unfalltyp", values = farben_utyp,
                      labels = c(
                        "Fahrunfall","Abbiege-Unfall",
                        "Einbiegen/Kreuzen-Unfall", "Überschreiten-Unfall"
                           ,
                        "Unfall␣durch␣ruhenden␣Verkehr",
                        "Unfall␣im␣Längsverkehr", "Sonstiger␣Unfall")) +
  scale_shape_manual(name = "Unfallschwere", values = formen_ukat,
                     labels = c(
                       "Unfall␣mit␣Getöteten", "Unfall␣mit␣
                          Schwerverletzten",
                       "Unfall␣mit␣Leichtverletzten")) +
  labs(title = "Unfalltypenkarte␣der␣A40␣im␣Jahr␣2024") +
```

```
  theme_minimal() +
  theme(
    panel.grid = element_blank(),
    plot.title = element_text(size = 14, face = "bold", hjust = 0.5),
    axis.title = element_blank(),
    axis.text  = element_blank(),
    axis.ticks = element_blank()
  )
```



Die Unfalltypenkarte der A40 mit ggplot zeigt viele Häufungen von Unfällen, insbesondere zwischen Dortmund und Duisburg. Der Maßstab lässt keine weiteren Aussagen zu.

2.1.2.2 Grafische Darstellung aller Unfälle auf Bundesautobahnen mit ggplot

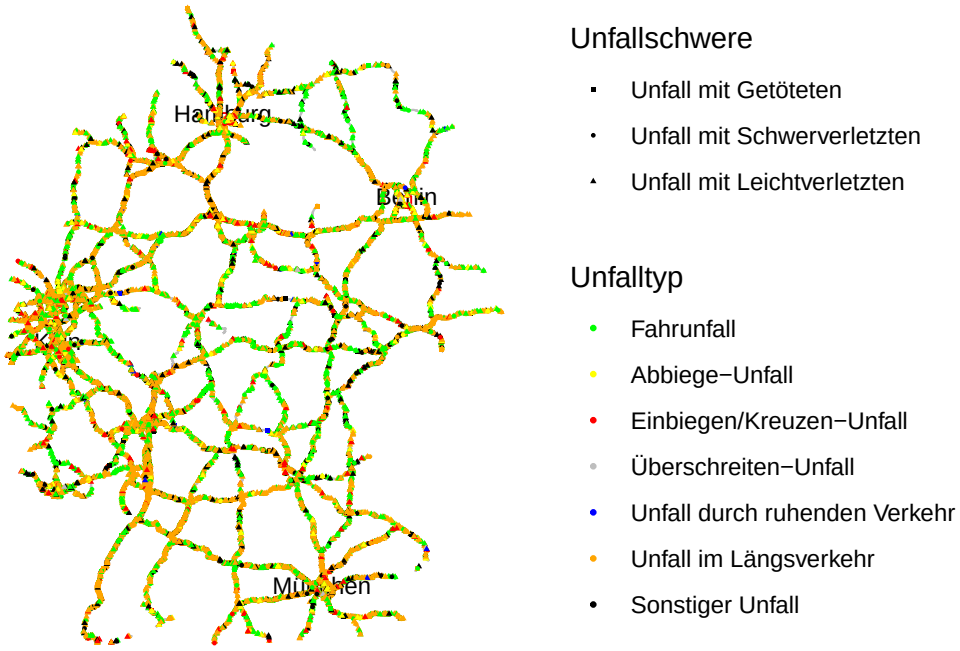
Der Code für die grafische Darstellung aller Unfälle auf Bundesautobahnen ähnelt dem obigen Code. Statt der drei Städte entlang der A40 lassen wir uns nun die vier größten Städte Deutschlands einblenden.

```
d_staedte2 <- st_as_sf(
  tibble::tribble(
    ~ stadt, ~lat, ~lon,
    "Berlin", 52.5244, 13.4105,
```

```
    "Hamburg", 53.5511, 9.9937,
    "München", 48.1372, 11.5761,
    "Köln", 50.9352, 6.9531
  ),
  coords = c("lon", "lat"),
  crs = 4326
) |>
  st_transform(25832)

ggplot () +
  geom_sf(data = d_bfstn, colour = "grey", size = 0.4) +
  geom_sf(data = d_staedte2, colour = "red", shape = 22) +
  geom_sf_text(data = d_staedte2, aes(label = stadt),
               nudge_y = 5000, size = 3) +
  geom_sf(
    data = d_unfaelle_bab_2024,
    mapping = aes(
      group = UTYP1,
      colour = as.character(UTYP1),
      shape = as.character(UKATEGORIE)),
    size = 0.5) +
  scale_colour_manual(name = "Unfalltyp", values = farben_utyp,
                      labels = c(
                        "Fahrunfall","Abbiege-Unfall",
                        "Einbiegen/Kreuzen-Unfall", "Überschreiten-Unfall"
                           ,
                        "Unfall␣durch␣ruhenden␣Verkehr",
                        "Unfall␣im␣Längsverkehr", "Sonstiger␣Unfall")) +
  scale_shape_manual(name = "Unfallschwere", values = formen_ukat,
                     labels = c(
                       "Unfall␣mit␣Getöteten", "Unfall␣mit␣
                          Schwerverletzten",
                       "Unfall␣mit␣Leichtverletzten")) +
  labs(title = "Unfalltypenkarte␣aller␣deutschen␣Autobahnen␣im␣Jahr␣2024")
      +
  theme_minimal() +
  theme(
    panel.grid = element_blank(),
    plot.title = element_text(size = 10, face = "bold", hjust = 0.5),
    axis.title = element_blank(),
    axis.text  = element_blank(),
    axis.ticks = element_blank()
  )
```

nfalltypenkarte aller deutschen Autobahnen im Jahr 2024



Die Unfalltypenkarte aller Bundesautobahnen hat recht wenig Mehrwert, da wir aufgrund der Menge der Unfälle und der fehlenden Möglichkeit heranzuzoomen nur eine grobe Übersicht bekommen, aber keine konkreten Aussagen treffen können. Daher nutzen wir nun leaflet zur Darstellung.

2.1.3 Grafische Darstellung mit `leaflet`

Als erstes formen wir unsere vorherige Farbpalette der Unfalltypen in ein Format, mit dem Leaflet arbeiten kann, um.

```
farben_utyp_leaf <- colorFactor(unname(farben_utyp), domain = 1:7)
```

2.1.3.1 Grafische Darstellung von Unfällen auf der Beispielautobahn A40 mit leaflet

Nun erstellen wir unsere Leaflet-Karte. “addTiles” fügt Hintergrundkarte von OpenStreetMap ein, mit “addProviderTiles” können wir die Kartenoptik noch anpassen (verschiedene Provider hier). Leaflet nutzt ein anderes CRS, daher transformieren wir unsere Unfalldaten. Mit “addCircleMarkers” werden Kreise als Markierungen hinzugefügt. Der Radius der Kreise soll anhand der Unfallkategorie bestimmt werden, ein Unfall der Kategorie 1 hat dabei beispielsweise einen Radius von 12 Pixel. Das orientiert sich am M Uko, ist aber eine Design-Entscheidung, wie

es ansprechend aussieht. Kreise, die mit “addCircleMarkers” erzeugt werden, verändern ihre Größe dynamisch mit dem Zoom. Für Kreise, die immer gleich groß sein sollen, verwendet man “addCircles” und gibt den Radius an. Für die Umrandung und Füllung der Marker nutzen wir das Farbschema des M Uko. Außerdem fügen wir ein Pop-Up hinzu, dass uns die Unfallkategorie und den Unfalltypen anzeigt. Hinzu kommt auch eine Legende unten rechts mit der Erklärung der Unfalltypen mit “addLegend”. Für die Legende der Radien brauchen wir “addLegendSize”. Eine gute Übersicht zu leaflet-Karten und ihren Einstellungsmöglichkeiten findet man hier.

```
leaflet(data = d_unfaelle_bab_2024 |> filter(Str_Kennung == "A40")
        |> st_transform(crs = 4326)) |>
  addTiles() |>
  addCircleMarkers(
    radius = ~c(12, 8, 5)[UKATEGORIE],
    color = ~farben_utyp_leaf(UTYP1), opacity = 1,
    fillColor = ~farben_utyp_leaf(UTYP1), fillOpacity = 0.9,
    popup = ~paste("Unfallkategorie:", UKATEGORIE, "<br>Unfalltyp:", UTYP1
       )
  ) |>
  addLegend(
    position = "bottomright",
    colors = c("green", "yellow",  "red", "grey", "blue", "orange", "black
       "),
    labels = c("Fahrunfall","Abbiege-Unfall", "Einbiegen/Kreuzen-Unfall",
               "Überschreiten-Unfall","Unfall␣durch␣ruhenden␣Verkehr",
               "Unfall␣im␣Längsverkehr", "Sonstiger␣Unfall"),
    title = "Unfalltyp",
    opacity = 1
  ) |>
  addLegendSize(
    values = c(12, 8, 5),
    breaks = c("Unfall␣mit␣Getöteten" = 12,
               "Unfall␣mit␣Schwerverletzten" = 8,
               "Unfall␣mit␣Leichtverletzten" = 5),
    color = "black",
    fillColor = "black",
    fillOpacity = 0.9,
    shape = "circle",
    position = "bottomleft",
    title = "Unfallschwere"
  )
```

Die interaktive leaflet-Karte ermöglicht uns im Gegensatz zu ggplot, dass wir an die Unfälle heranzoomen können und mit der Möglichkeit der Pop-ups auch dynamische Unfall-Beschriftungen. Allerdings sind diese Vorteile in einem pdf-Dokument nicht gegeben. Die Methode, die wir verwenden, orientiert sich folglich auch immer an unseren Vorgaben.

2.1.3.2 Grafische Darstellung aller Unfälle auf Bundesautobahnen mit leaflet

Die gleichen Schritte wiederholen wir nun für die Unfälle auf Bundesautobahnen.

```
cat("*Interaktive Karte nur in der HTML-Version verfügbar.*")
```

```
*Interaktive Karte nur in der HTML-Version verfügbar.*
```

```
leaflet(data = st_transform(d_unfaelle_bab_2024, 4326)) |>
  addTiles() |>
  addCircleMarkers(
    radius = ~c(12, 8, 5)[UKATEGORIE],
    color = ~farben_utyp_leaf(UTYP1), opacity = 1,
    fillColor = ~farben_utyp_leaf(UTYP1), fillOpacity = 0.9,
    popup = ~paste("Unfallkategorie:", UKATEGORIE, "<br>Unfalltyp:",
                   UTYP1, "<br>Autobahn:", Str_Kennung)
  ) |>
  addLegend(
    position = "bottomright",
    colors = c("green", "yellow",  "red", "grey", "blue", "orange", "black
        "),
    labels = c("Fahrunfall","Abbiege-Unfall", "Einbiegen/Kreuzen-Unfall",
               "Überschreiten-Unfall","Unfall durch ruhenden Verkehr",
               "Unfall im Längsverkehr", "Sonstiger Unfall"),
    title = "Unfalltyp",
    opacity = 1
  ) |>
  addLegendSize(
    values = c(12, 8, 5),
    breaks = c("Unfall mit Getöteten" = 12,
               "Unfall mit Schwerverletzten" = 8,
               "Unfall mit Leichtverletzten" = 5),
    color = "black",
    fillColor = "black",
    fillOpacity = 0.9,
    shape = "circle",
    position = "bottomleft",
    title = "Unfallschwere"
  )
```

2.2 Übungsaufgabe 1.2 : Unfallhäufungsstellen

Für Unfallhäufungsstellen betrachten wir Unfalldaten aus drei (aufeinanderfolgenden) Jahren am Beispiel der Stadt Bochum, vergleichbar zu einer Dreijahreskarte (3-JK). Laut M Uko wird dann von einer Unfallhäufungsstelle gesprochen, wenn auf Stadtstraßen in der 3-JK mindestens fünf Unfälle mit Personenschaden ohne besondere Berücksichtigung von Unfalltypen

an einem Knotenpunkt auftreten. Die räumliche Ausdehnung ist hierbei 50 Meter um den Achsenschnittpunkt. Auf der freien Strecke ist eine Unfallhäufungsstelle ebenfalls ab fünf Unfällen mit Personenschaden in maximal 50 Metern Ausdehnung ab Knoteneinfluss. (FGSV 2012, 15)

2.2.1 Daten einlesen und aufbereiten

Als erstes laden wir die Unfalldaten aus den drei aufeinanderfolgenden Jahren und verbinden sie mit "bind_rows". Nun bereiten wir sie ähnlich wie in Aufgabe 1.1 auf. Zusätzlich filtern wir nach dem Gemeindeschlüssel der Stadt Bochum (05 = NRW, 9 = Regierungsbezirk Arnsberg, …). Außerdem interessieren uns dieses Mal nur Stadtstraßen, dazu ermitteln wir wie bei der obigen Aufgabe die nächstgelegene Autobahn und die Entfernung zu dieser, nur dass wir nun alle Unfälle, die mindestens 20 Meter von einer Autobahn entfernt sind, behalten. Auch hier behalten wir für die Übersichtlichkeit nur ausgewählte Spalten ("select"). Damit wir diesen Schritt, der länger dauern kann, nicht jedes Mal durchführen müssen, speichern wir den Datensatz und rufen ihn beim nächsten Mal wieder auf.

```
if (!file.exists("daten/unfaelle_alle.RData")) {

  d_unfaelle_bochum_3j <- bind_rows(
    read_csv2("daten/Unfallorte2024_LinRef.csv") |> mutate(jahr = 2024),
    read_csv2("daten/Unfallorte2023_LinRef.csv") |> mutate(jahr = 2023),
    read_csv2("daten/Unfallorte2022_LinRef.csv") |> mutate(jahr = 2022)
    ) |>
    filter(
      ULAND == "05",
      UREGBEZ == "9",
      UKREIS == "11",
      UGEMEINDE == "000"
    ) |>
    st_as_sf(coords = c("LINREFX", "LINREFY"), crs = 25832) |>
    st_zm()  |>
    mutate(
      abschnitt_id_bab = st_nearest_feature(geometry, d_bfstn),
      distanz_bab = st_distance(geometry, d_bfstn[abschnitt_id_bab, ],
                                by_element = TRUE),
      name_bab = d_bfstn$Str_Kennung[abschnitt_id_bab]
    ) |>
    filter(as.double(distanz_bab) > 20) |>
    select(OID_, UJAHR, UMONAT, USTUNDE, UKATEGORIE, UTYP1,
           IstPKW, IstRad, IstFuss, IstKrad,
           ULICHTVERH, IstStrassenzustand)

  save(
    d_unfaelle_bochum_3j,
    file = "daten/unfaelle_alle.RData"
```

```
  )
} else {
  load(file = "daten/unfaelle_alle.RData")
}
```

Damit wir später unsere OpenStreetMap(OSM)-Daten anhand des Stadtgebiets der Stadt Bochum zuschneiden können und da die Stadtgrenze als open data zur Verfügung steht, laden wir sie in unser Projekt. Dieser Schritt ist optional, sorgt aber dafür, dass unser Endergebnis ansprechender ist, da OSM die Daten aus einem Rechteck abfragt.

```
d_bochum_stadtgebiet <- st_read("daten/geo/Stadtgrenze.shp", quiet = TRUE)
    |>
  st_transform(25832)
```

2.2.2 Knoten-Kanten-Modell

Als nächstes möchten wir ein simples Knoten-Kanten-Modell der Stadtstraßen erstellen, bei dem jeder Knotenpunkt (jede Kreuzung, jeder Kreisverkehr) ein Punkt und jede Verbindung zwischen Punkten eine Linie ist. Das ist mit OSM recht schwierig, da die Daten sehr detailliert und kleinteilig sind. Der hier gezeigte Ansatz erhebt daher keinen Anspruch auf Perfektion oder Vollständigkeit, sondern soll ein Mittelmaß zwischen ansprechender Anschaulichkeit und nicht zu hoher Komplexität zeigen.

Als erstes laden wir uns die Straßendaten von OSM herunter. 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.

```
if (!file.exists("daten/strassen_bo.RData")) {
  d_strassen <- opq(bbox = "Bochum, Germany") |>
    add_osm_feature(
      key = "highway",
      value = c(
        "trunk", "trunk_link",
        "primary", "primary_link",
        "secondary", "secondary_link",
        "tertiary", "tertiary_link",
        "residential", "unclassified"
      )
    ) |>
    osmdata_sf()
  save(d_strassen, file = "daten/strassen_bo.RData")
} else {
  load(file = "daten/strassen_bo.RData")
}
```

Als nächstes nutzen wir die Spalte “osm_lines” aus dem gerade erzeugten Datensatz für unsere Kanten.

```
d_kanten <- d_strassen$osm_lines |>
  select(name, highway, geometry) |>
  st_transform(25832)
```

Da für unsere Unfallhäufungsstellen die Fahrtrichtung keine (große) Rolle spielt, bilden wir aus den Kanten ein Netzwerk und glätten mit “convert(to_spatial_smooth) ein erstes Mal.

```
d_netz <- as_sfnetwork(d_kanten, directed = FALSE) |>
  convert(to_spatial_smooth)
```

OSM-Daten sind sehr detailliert: Ein einzelner Knotenpunkt (z. B. eine große Kreuzung oder ein Kreisverkehr) besteht in der OSM-Darstellung oftmals aus mehreren Netzwerk-Knoten, nämlich überall dort, wo sich zwei Fahrspuren oder Abbiegestreifen treffen. Für unsere Unfallanalyse wollen wir solche Knoten-Cluster jeweils zu einem einzigen Knoten zusammenfassen. Dafür bestimmen wir zunächst Nachbarschaften. Knoten des Netzwerks werden als sf-Objekt extrahiert und mit “st_is_within_distance” wird für jeden Knoten ermittelt, welche anderen Knoten innerhalb eines Schwellenwertes von 20 m liegen. Das Ergebnis ist eine Liste, die die räumliche Nähe beschreibt. Anschließend wird diese Liste mit “graph_from_adj_list” aus dem Paket “igraph” in einen ungerichteten Graphen überführt. Dessen zusammenhängende Komponenten (“components”) liefern eine Gruppen-ID: Alle Knoten, die innerhalb von 20 Metern zueinander liegen, erhalten dieselbe Gruppe. So wird z. B. ein Kreisverkehr mit zehn OSM-Knoten, die jeweils paarweise nahe beieinander liegen, zu einer einzigen Gruppe zusammengefasst. Nun wird mit “convert(to_spatial_contracted, group, simplify = TRUE)” jede Knotengruppe zu einem einzelnen Knoten verschmolzen und dem Datensatz zu unserem Netzwerk hinzugefügt. Die Kanten, die vorher zwischen Knoten derselben Gruppe verliefen, entfallen. Kanten zu externen Knoten werden auf den neuen, zusammengefassten Knoten umgehängt. Der Parameter simplify = TRUE entfernt dabei entstehende Mehrfachkanten und Schleifen. Die Variablen “knoten_sf”, “ “nachbarknoten” und “gruppen_id” sind Einmal-Variablen, die mit local() gekapselt werden können, sodass sie danach nicht im Environment verbleiben.

```
if (!file.exists("daten/strassennetz_bo.RData")) {
  d_netz <- local({
    knoten_sf <- d_netz |> activate("nodes") |> st_as_sf()
    nachbarn <- st_is_within_distance(knoten_sf, dist = set_units(20, "m")
        )
    gruppen <- components(graph_from_adj_list(nachbarn, mode = "all"))$
        membership
    d_netz |>
      activate("nodes") |>
      mutate(group = gruppen) |>
      convert(to_spatial_contracted, group, simplify = TRUE)
  })
  save(d_netz, file = "daten/strassennetz_bo.RData")
} else {
```

```
  load(file = "daten/strassennetz_bo.RData")
}
```

Durch die Knotenkontraktion können Mehrfachkanten (mehrere Kanten zwischen demselben Knotenpaar) und Schleifen (Kanten von einem Knoten zu sich selbst) entstehen. Diese werden nun bereinigt: Die Kanten werden zunächst nach Länge sortiert ("arrange(edge_length())"), sodass beim anschließenden Filtern jeweils die kürzeste Verbindung erhalten bleibt. edge_is_multiple() und edge_is_loop() identifizieren die überflüssigen Kanten, die dann entfernt werden.

```
d_netz <- d_netz |>
  activate("edges") |>
  arrange(edge_length()) |>
  filter(!edge_is_multiple(), !edge_is_loop())
```

Für die weiteren Berechnungen extrahieren wir uns aus unserem Netzwerk die Knoten und Kanten.

```
d_knoten <- d_netz |>
  activate("nodes") |>
  st_as_sf() |>
  st_filter(d_bochum_stadtgebiet)

d_kanten <- d_netz |>
  activate("edges") |>
  st_as_sf() |>
  st_filter(d_bochum_stadtgebiet)
```

2.2.3 Ermittlung von Unfallhäufungsstellen

Sowohl für die Unfallhäufungsstellen an Knotenpunkten als auch an freien Strecken brauchen wir je Unfall den nächstgelegenen Knotenpunkt und die Entfernung zu diesem. Daher fügen wir diese mit "st_nearest_feature" und "st_distance" dem Datensatz der Unfalldaten hinzu.

```
d_unfaelle_bochum_3j <- d_unfaelle_bochum_3j |>
  mutate(
    knoten_id = st_nearest_feature(geometry, d_knoten),
    distanz_knoten = as.numeric(st_distance(geometry,
                                            d_knoten[knoten_id, ], by_
                                               element = TRUE))
  )
```

2.2.3.1 Unfallhäufungsstellen an Knotenpunkten

Um Unfälle im Umkreis von 50 Metern um Knotenpunkte zu ermitteln, erhält jeder Knotenpunkt zuerst eine fortlaufende ID ("row_number"). Dann werden die Unfalldaten mit "inner_join"

mit der Knotengeometrie verknüpft und mit “count” alle Unfälle innerhalb von 50 m um einen Knotenpunkt nach ihrer knoten_id gezählt. Danach filtern wir nach Knotenpunkten mit mindestens fünf Unfällen.

```
d_knoten_uhs <- d_knoten |>
  mutate(knoten_id = row_number()) |>
  inner_join(
    d_unfaelle_bochum_3j |>
      st_drop_geometry() |>
      filter(distanz_knoten <= 50) |>
      count(knoten_id, name = "anzahl_unfaelle") |>
      filter(anzahl_unfaelle >= 5),
    by = "knoten_id"
  )
```

2.2.3.2 Unfallhäufungsstellen an freien Strecken

Für die Unfälle an freien Strecken betrachten wir die Unfälle, deren Entfernung zu Knotenpunkten größer als 50 Metern ist. Die Schritte für die Unfallhäufungsstellen an freien Strecken nutzen Funktionen, bei deren leere Datensätze zu Fehlern oder unerwarteten Ergebnissen führen, daher beginnen wir unseren Code damit, den Datensatz gleich NULL zu setzen, wenn die nachfolgenden Funktionen keine Unfallhäufungsstellen liefern. Die Unfallhäufungsstellen an freien Strecken beginnen wie bei der Bereinigung der Knotenpunkte mit einer Nachbarschaftszählung. Für jeden Strecken-Unfall wird ein 50-m-Puffer erzeugt (“st_buffer”) und gezählt, wie viele andere Unfälle in diesem Umkreis liegen (“n_nearby”). Nur Unfälle mit mindestens 4 Nachbarn (also 5 Unfälle insgesamt) werden als Kandidaten weiterverarbeitet. Dieser Vorfilter reduziert die Datenmenge erheblich und stellt sicher, dass isolierte Einzelunfälle gar nicht erst ins Clustering eingehen. Darauf folgt das Clustering mit dbscan. Die verbleibenden Kandidaten werden mit dem DBSCAN-Algorithmus räumlich geclustert. dbscan eignet sich hier besonders gut, weil es keine vorab festgelegte Clusteranzahl benötigt und beliebig geformte Häufungen erkennen kann – etwa entlang einer kurvigen Straße. Die Parameter entsprechen der Logik des Vorfilters: Mindestens 5 Unfälle (“minPts”) innerhalb von 50 m (“eps”) bilden eine Unfallhäufungsstelle. Zuletzt wird für jedes gefundene Cluster die Anzahl der zugehörigen Unfälle berechnet und der Schwerpunkt (“st_centroid”) aller Unfallpunkte als repräsentativer Standort der Unfallhäufungsstelle ermittelt. Punkte, die dbscan als Rauschen klassifiziert (“cluster == 0”), werden verworfen.

```
d_strecke_uhs <- NULL

d_unfaelle_strecke <- d_unfaelle_bochum_3j |>
  filter(distanz_knoten > 50)

if (nrow(d_unfaelle_strecke) > 0) {
  d_unfaelle_strecke$n_nearby <- lengths(
    st_intersects(st_buffer(d_unfaelle_strecke, 50), d_unfaelle_strecke)
```

```
  )

  d_uhl_kandidaten <- d_unfaelle_strecke |>
    filter(n_nearby >= 5)

  if (nrow(d_uhl_kandidaten) > 0) {
    d_uhl_kandidaten$cluster <- dbscan(st_coordinates(d_uhl_kandidaten),
                                       eps = 50, minPts = 5)$cluster

    d_strecke_uhs <- d_uhl_kandidaten |>
      filter(cluster > 0) |>
      group_by(cluster) |>
      summarise(
        anzahl_unfaelle = n(),
        geometry = st_centroid(st_combine(geometry)),
        .groups = "drop"
      )
  }
}
```

2.2.4 Grafische Darstellung der Unfallhäufungsstellen auf Stadtstraßen in Bochum

Wie bei der Aufgabe 1.1 transformieren wir die Datensätze, die wir darstellen wollen, in das von leaflet verwendete CRS. Für die Darstellung nutzen wir die Kanten des Straßennetzes als Hintergrund ("addPolylines"). Die Unfallhäufungsstellen an Knotenpunkten stellen wir mit lila farbenen Kreisen mit dem festen Radius von 50 Metern dar, Unfallhäufungsstellen in pink. Durch die festen Radien sind die UHS beim Herauszoomen schlecht zu identifizieren. Daher ergänzen wir noch Marker ("addAwesomeMarkers"). Eine Auswahl an verschiedenen Markern kann beispielsweise hier gefunden werden. Den Unfällen fügen wir einen kleinen räumlichen Versatz ("st_jitter") hinzu, damit Unfälle, die an der exakt gleichen Stelle sind oder so im System eingetragen wurden, dargestellt werden und sich nicht überlagern. Außerdem stellen wir die Unfälle mit den gleichen Farben wie in Aufgabe 1.1 dar. Für diese Karte wählen wir für alle Unfallkategorien die gleiche Größe der Kreise. Zudem ergänzen wir beim Pop-Up noch Informationen zum Zeitpunkt des Unfalls. Außerdem ergänzen wir noch zwei Legenden und eine Layer-Kontrolle zum Ein- und Ausblenden. Auch für die grafische Darstellung sichern wir uns dagegen ab, dass bei der Berechnung von UHS an freien Strecken NULL ergibt, was beim Renderung zu Laufzeitfehlern führen kann.

```
karte <- leaflet() |>
  addTiles() |>
  addPolylines(
    data = st_transform(d_kanten, 4326),
    color = "black", weight = 1, opacity = 1,
    group = "Straßennetz"
  ) |>
```

```
  addCircles(
    data = st_transform(d_knoten_uhs, 4326),
    radius = 50,
    fillOpacity = 0.4,
    color = "purple", weight = 2,
    popup = ~paste("UHS␣an␣Knotenpunkt<br>Anzahl␣Unfälle:", anzahl_
       unfaelle),
    group = "UHS␣an␣Knotenpunkten"
  ) |>
  addAwesomeMarkers(
    data = st_transform(d_knoten_uhs, 4326),
    icon = awesomeIcons(icon = "alert-circled", library = "ion",
                        markerColor = "purple", iconColor = "black"),
    group = "UHS␣an␣Knotenpunkten"
  ) |>
  addCircleMarkers(
    data = st_transform(st_jitter(d_unfaelle_bochum_3j, amount = 3), 4326)
       ,
    radius = 3,
    color = ~farben_utyp_leaf(UTYP1), opacity = 0.7,
    fillColor = ~farben_utyp_leaf(UTYP1), fillOpacity = 0.7,
    popup = ~paste0(
      "Unfall␣im␣",
      month(as.numeric(UMONAT), label = TRUE, abbr = FALSE), "␣", UJAHR,
      "␣um␣", USTUNDE, "␣Uhr<br>",
      "Typ:␣", UTYP1, "␣|␣Kategorie:␣", UKATEGORIE
    ),
    group = "Unfälle"
  )

if (!is.null(d_strecke_uhs) && nrow(d_strecke_uhs) > 0) {
  karte <- karte |>
    addCircles(
      data = st_transform(d_strecke_uhs, 4326),
      radius = 50,
      fillOpacity = 0.4,
      color = "pink", weight = 2,
      popup = ~paste("UHS␣an␣freier␣Strecke<br>Anzahl␣Unfälle:",
                     anzahl_unfaelle),
      group = "UHS␣an␣freier␣Strecke"
    ) |>
    addAwesomeMarkers(
      data = st_transform(d_strecke_uhs, 4326),
      icon = awesomeIcons(icon = "alert-circled", library = "ion",
                          markerColor = "pink", iconColor = "black"),
      group = "UHS␣an␣freier␣Strecke"
    )
}
```

```
karte |>
  addLegend(
    position = "bottomleft",
    colors = c("purple", "pink"),
    labels = c("UHS␣an␣Knotenpunkten␣ (␣5␣Unfälle)",
               "UHS␣an␣freier␣Strecke␣ (␣5␣Unfälle)"),
    title = "Unfallhäufungsstellen",
    opacity = 1
  ) |>
  addLegend(
    position = "bottomright",
    colors = c("green", "yellow", "red", "grey", "blue", "orange", "black"
       ),
    labels = c("Fahrunfall", "Abbiege-Unfall", "Einbiegen/Kreuzen-Unfall",
               "Überschreiten-Unfall", "Unfall␣durch␣ruhenden␣Verkehr",
               "Unfall␣im␣Längsverkehr", "Sonstiger␣Unfall"),
    title = "Unfalltyp",
    opacity = 1
  ) |>
  addLayersControl(
    overlayGroups = c("Straßennetz", "UHS␣an␣Knotenpunkten",
                      "UHS␣an␣freier␣Strecke", "Unfälle"),
    options = layersControlOptions(collapsed = FALSE)
  )
```

Wir haben nun mit unserem OSM-Modell und den darauf basierenden Berechnungen 60 UHS an Knotenpunkten und 3 UHS an freier Strecke identifiziert und grafisch dargestellt. Wenn wir die Daten mit den polizeilichen Berichten vergleichen, stellen wir fest, dass wir deutlich mehr identifiziert haben als die Polizei. Dies liegt an unterschiedlichen Filtern, z. B. der Unfallschwere, die sich aber je nach Untersuchungsschwerpunkt im R Code anpassen lassen. Wir haben ein Werkzeug entwickelt, das potenzielle Unfallhäufungen identifiziert, eine abschließende Verifizierung jedoch nicht ersetzt.

3 Code zur interaktiven Karte der Verkehrsunfälle in Bochum

3.1 Interaktive Karte mit `shiny`

Für die interaktive Karte mit shiny öffnen Sie bitte die Datei `unfaelle_shiny.qmd` und klicken Sie oben links `Run Document`. Es dauert einen Moment, dann öffnet sich ein weiteres RStudio-Fenster mit der Karte, den Filtern und der Tabelle mit Download-Button.

3.2 Interaktive Karte mit `crosstalk`

Für die Einbindung der interaktiven Karte im html-Dokument bietet es sich an statt `shiny` `crosstalk` zu nutzen.

````
```{r}
#| include: false
library(leaflet)
library(tidyverse)
library(sf)
library(lubridate)
library(crosstalk)
library(DT)
library(htmltools)
library(knitr)
library(curl)
```
````

Als erstes laden wir uns die Daten vom Bundesfernstraßennetz der BASt herunter bzw aus dem “daten”-Ordner, um gleich die Unfälle filtern zu können. Danach lesen als nächstes alle Unfalldaten ein, die in unserem “daten”-Ordner zur Verfügung stehen. Manche sind im .csv-Format, andere als .txt abgespeichert. Wir stellen sicher, dass alle Spaltennamen und -inhalte der verschiedenen Jahre übereinstimmen. Danach verbinden wir die Datensätze, filtern nach dem Gemeindeschlüssel der Stadt Bochum und filtern nach Unfällen, die mindestens 20 Meter von Autobahnen entfernt sind, da uns nur Unfälle auf Stadtstraßen interessieren. Diese beiden Datensätze speichern wir, damit sie, sofern sie vorhanden sind, nur geladen und nicht jedes Mal neu berechnet werden müssen.

```{r}
if (!file.exists("daten/unfaelle_interaktiv.RData")) {
  # Fernstraßennetz
  curl_download(
    "https://www.bast.de/SharedDocs/Daten-TB/Daten-BISStra.zip?__blob=
       publicationFile&v=5",
    destfile = "daten/Daten-BISStra.zip",
    quiet = FALSE
  )
  unzip("daten/Daten-BISStra.zip", exdir = "daten/geo/")
  file <- list.files(
    "daten/geo",
    pattern = "^BFStr_Netz.*\\.gpkg$",
    full.names = TRUE
  ) |>
    tail(n = 1)
  d_bfstn <- read_sf(file) |>
    filter(Str_Klasse_kurz == "A" & Sk_Achse == "Bestandsachse") |>
    mutate(rownumber = row_number()) |>
    st_zm()

  # Unfalldaten
  unfaelle_einlesen <- function(datei) {
    daten <- read_delim(
      datei,
      delim = ";",
      locale = locale(decimal_mark = ",", grouping_mark = "."),
      col_types = cols(
        ULAND     = col_character(),
        UKREIS    = col_character(),
        UGEMEINDE = col_character(),
        UMONAT    = col_character(),
        USTUNDE   = col_character(),
        .default  = col_guess()
      ),
      show_col_types = FALSE
    )

    daten <- daten |>
      mutate(
        ULAND = str_pad(ULAND, width = 2, pad = "0"),
        UKREIS = str_pad(UKREIS, width = 2, pad = "0"),
        UGEMEINDE = str_pad(UGEMEINDE, width = 3, pad = "0")
      )

    if ("USTRZUSTAND" %in% names(daten) && !("STRZUSTAND" %in% names(daten
       ))) {
      daten <- daten |> rename(STRZUSTAND = USTRZUSTAND)
```

```
  }
  if ("IstStrassenzustand" %in% names(daten) && !("STRZUSTAND" %in%
  names(daten))) {
    daten <- daten |> rename(STRZUSTAND = IstStrassenzustand)
  }
  if ("OID_" %in% names(daten) && !("OBJECTID" %in% names(daten))) {
    daten <- daten |> rename(OBJECTID = OID_)
  }

  daten |>
    mutate(
      UWOCHENTAG = wday(as.numeric(UWOCHENTAG), label = TRUE),
      UMONAT = month(as.numeric(UMONAT), label = TRUE, abbr = FALSE)
    )
}

d_unfalldaten <- list.files(path = "daten", pattern = "\\.(csv|txt)$",
full.names = TRUE) |>
  map(unfaelle_einlesen) |>
  list_rbind() |>
  filter(
    ULAND == "05",
    UREGBEZ == "9",
    UKREIS == "11",
    UGEMEINDE == "000"
  ) |>
  st_as_sf(coords = c("LINREFX", "LINREFY"), crs = 25832) |>
  st_zm()  |>
  mutate(
    abschnitt_id_bab = st_nearest_feature(geometry, d_bfstn),
    distanz_bab = st_distance(geometry, d_bfstn[abschnitt_id_bab, ],
    by_element = TRUE),
    name_bab = d_bfstn$Str_Kennung[abschnitt_id_bab]
  ) |>
  filter(as.double(distanz_bab) > 20) |>
  mutate(
    UKATEGORIE_neu = factor(UKATEGORIE,
      levels = c("1", "2", "3"),
      labels = c("Unfall mit Getöteten",
                 "Unfall mit Schwerverletzten",
                 "Unfall mit Leichtverletzten")),
    UTYP1_neu = factor(as.integer(UTYP1),
      levels = 1:7,
      labels = c(
          "Fahrunfall",
          "Abbiege-Unfall",
          "Einbiegen/Kreuzen-Unfall",
          "Überschreiten-Unfall",
```

```
            "Unfall durch ruhenden Verkehr",
            "Unfall im Längsverkehr",
            "Sonstiger Unfall"
        )),
      IstPKW_neu = ifelse(IstPKW == "1", "beteiligt", "nicht beteiligt"),
      IstRad_neu = ifelse(IstRad == "1", "beteiligt", "nicht beteiligt"),
      IstFuss_neu = ifelse(IstFuss == "1", "beteiligt", "nicht beteiligt")
    )

  # Daten sichern
  save(
    d_bfstn,
    d_unfalldaten,
    file = "daten/unfaelle_interaktiv.RData"
  )
} else {
  load(file = "daten/unfaelle_interaktiv.RData")
}
```

Nun treffen wir Vorbereitungen für die grafische Darstellungen. Zunächst fügen wir den Unfalldaten einen räumlichen Versatz von 3 Metern hinzu, damit Unfälle sich auf der Karte nicht überlagern und transformieren in das CRS, mit dem leaflet arbeitet. Außerdem definieren wir die Farben nach dem M Uko (FGSV 2012).

```
```{r}
d_karte <- d_unfalldaten |>
  st_jitter(amount = 3) |>
  st_transform(4326)

coords <- st_coordinates(d_karte)

d_karte <- d_karte |>
  mutate(
    ID = row_number(),
    lng = coords[, 1],
    lat = coords[, 2]
  )

farben_utyp <- colorFactor(
  palette = c("green", "yellow", "red", "grey", "blue", "orange", "black")
    ,
  levels  = c(
    "Fahrunfall",
    "Abbiege-Unfall",
    "Einbiegen/Kreuzen-Unfall",
    "Überschreiten-Unfall",
    "Unfall durch ruhenden Verkehr",
```
```

```
    "Unfall im Längsverkehr",
    "Sonstiger Unfall"
  )
)
```

Für die Filter mit crosstalk brauchen wir die Daten als “shared data”, daher transformieren wir die Unfalldaten, einmal für die Karte und einmal für die Tabelle.

```{r}
sd <- SharedData$new(d_karte, key = ~ID, group = "unfaelle")

sd_tabelle <- SharedData$new(
  d_karte |>
    st_drop_geometry() |>
    select(ID, Jahr = UJAHR, Monat = UMONAT, Wochentag = UWOCHENTAG,
           Kategorie = UKATEGORIE_neu, Unfalltyp = UTYP1_neu,
           PKW = IstPKW_neu, Rad = IstRad_neu, `Zu Fuß` = IstFuss_neu),
  key = ~ID,
  group = "unfaelle"
)
```

Nun kommen wir zur grafischen Darstellung. “bscols” ermöglicht uns das Anordnen von HTML Elementen in Bootstrap columns. Damit erzeugen wir zuerst drei Spalten für die Filter der Unfallkategorie, des Unfalltyps und der Unfallbeteiligung. Bei diesen Filtern nutzen wir “filter_checkbox”, weil wir die Filter nur ein- oder ausschalten wollen. Für den Jahresfilter nutzen wir “filter_slider”, mit “sep =” ” ” und “round = TRUE” sorgen wir für eine Darstellung der Jahreszahlen ohne Dezimaltrennzeichen oder -lücke. Hilfen für Darstellungen mit crosstalk finden sich hier oder hier.

Unterhalb der Filter lassen wir uns unsere leaflet-Karte darstellen.

Darunter folgt eine Tabelle mit den gefilterten Unfällen. Wir aktivieren “Scroller” und “Buttons”, ermöglichen das horizontale und vertikale Scrollen und ergänzen einen Button zum Download der gefilterten Unfalldaten als .csv-Datei.

Wir stellen ein, dass dieser COde-Chunk nur ausgeführt wird, wenn das Output-Format html ist und bei einem pdf ein Platzhalter-Text erscheint.

```{r}
#| eval: !expr knitr::is_latex_output()
#| when-format: pdf

cat("*Interaktive Karte nur in der HTML-Version verfügbar.*")
```

```{r}
#| eval: !expr knitr::is_html_output()
#| when-format: html

bscols(
  widths = c(4, 4, 4),
  filter_checkbox("kat",  "Unfallkategorie", sd, ~UKATEGORIE_neu),
  filter_checkbox("typ",  "Unfalltyp", sd, ~UTYP1_neu),
  tags$h5("Beteiligung"),
      filter_checkbox("pkw",  "PKW", sd, ~IstPKW_neu),
      filter_checkbox("rad",  "Rad", sd, ~IstRad_neu),
      filter_checkbox("fuss", "Zu Fuß Gehende", sd, ~IstFuss_neu)
)

bscols(
  widths = c(1, 10, 1),
  "",
  filter_slider("jahr","Jahre", sd, ~UJAHR, sep = "", round = TRUE),
  ""
)

leaflet(sd) |>
  addTiles() |>
  addCircleMarkers(
    lng = ~lng,
    lat = ~lat,
    radius      = 3,
    color       = ~farben_utyp(UTYP1_neu),
    opacity     = 0.7,
    fillColor   = ~farben_utyp(UTYP1_neu),
    fillOpacity = 0.7,
    popup = ~paste0(
      "Unfall im ",
      month(as.numeric(UMONAT), label = TRUE, abbr = FALSE), " ", UJAHR,
      " um ", USTUNDE, " Uhr<br>",
      UTYP1_neu, " | ", UKATEGORIE_neu
    )
  ) |>
  addLegend(
    position = "bottomright",
    colors   = c("green","yellow","red","grey","blue","orange","black"),
    labels   = levels(d_karte$UTYP1_neu),
    title    = "Unfalltyp",
    opacity  = 0.7
    )

datatable(
  sd_tabelle,
```

```
  extensions = c("Scroller", "Buttons"),
  style = "bootstrap",
  class = "compact stripe hover",
  width = "100%",
  rownames = FALSE,
  colnames = c("ID","Jahr","Monat","Wochentag","Kategorie",
               "Unfalltyp","PKW","Rad","Zu Fuß"),
  options = list(
    deferRender = TRUE,
    scrollY = 300,
    scrollX = 300,
    scroller = TRUE,
    columnDefs = list(list(visible = FALSE, targets = 0)),
    dom = "frtBiS",
    buttons = list(
      list(extend = "csv", text = "Download CSV"))
  )
)
```

4 Interaktive Karte der Verkehrsunfälle in Bochum


```
*Interaktive Karte nur in der HTML-Version verfügbar.*
```

FGSV. 2012. *Merkblatt zur Örtlichen Unfalluntersuchung in Unfallkommissionen (M Uko)*. Köln: Forschungsgesellschaft für Straßen- und Verkehrswesen (FGSV).