Road casualties in the UK (data.gov.uk)

Task

Tell a data story of British traffic accidents ending with death or serious injury ("road casualties") in the past five years, based on the authentic data from the British government's website `data.gov.uk`.

Introduction

In this exercise, you are going to work with authentic data "in the wild". Fortunately, these data sets are quite neat. Also, the site provides a metadata file at this URL: https://data.dft.gov.uk/road-accidents-safety-data/dft-road-casualty-statistics-road-safety-open-dataset-data-guide-2023.xlsx. This file seems to have been created for traffic data safety data sets produced by the Department of Transport, but it is not clear how well it corresponds to the casualty data sets. It may be outdated or just incomplete. This is very often the case in open data provided by public administration.

This real-world exercise will:

- **make you familiar with the RNotebook format - RMarkdown.** You can comfortably write a long text in it as well as embed code chunks. With RNotebooks you can generate nice reports that you can directly export to html, pdf, or even MS Word. For a quick reference to Markdown formatting, go to RStudio Help > Markdown Quick Reference (which opens in the RStudio Help pane) or to https://raw.githubusercontent.com/rstudio/cheatsheets/main/rmarkdown.pdf.
- show you how to read csv and MS Excel files from the web
- show you how to inspect and make sense of such files

Load the relevant libraries

```
library(readr, warn.conflicts = FALSE, quietly = TRUE) # to read csv files
library(dplyr, warn.conflicts = FALSE, quietly = TRUE) # to use pipe and to manipulate data frames
library(readxl, warn.conflicts = FALSE, quietly = TRUE) # to read MS Excel files

## Warning: package 'readxl' was built under R version 4.4.2

library(ggplot2, warn.conflicts = FALSE, quietly = TRUE) # to draw plots
```

Road Safety Data (Department of Transport)

This is the website that contains the casualty datasets. Have a look at it.

https://www.data.gov.uk/dataset/cb7ae6f0-4be6-4935-9277-47e5ce24a11f/road-safety-data

One could spend hours following all the hyperlinks. Let us decide that we want to get all csv files that explicitly refer to road casualties from the Data Links section (Fig. #SectionDataLinks). Please also hit the *See more* button to get files from earlier years, starting with 2019). We will look for guidance in the file that we will name `guide.xlsx` (see Fig. #guide).

Data links

Link to the data	Format	File added	Data preview
Road Safety Data - Casualties 2023	CSV	30 September 2024	Preview
Road Safety Data - Vehicles 2023	CSV	30 September 2024	Preview
Road Safety Data - Collisions 2023	CSV	30 September 2024	Preview
Road Safety Data - Casualties 2022	CSV	30 September 2024	Preview
Road Safety Data - Vehicles 2022	CSV	30 September 2024	Preview

Show more

Road Casualties csv files

Supporting documents

Link to the document	Format	Date added
Published statistics and supporting documents	HTML	05 October 2015
Understanding historical road safety data	.docx	14 October 2021
Road Safety Open Data Guide - 2023	.xlsx	30 September 2024
Road Safety Statistics - User Feedback Survey	HTML	28 November 2023
Road Safety Data - Severity Adjustement Giudance	.docx	27 September 2024

Metadata file to the Road casualties datasets

Read the files

Use `readr::read_csv` to read the csv files with casualties for the years available (Fig. #guide) . By this, you will directly create data frame objects in your workspace. Note that this will not download the original csv files to your computer (but in this case we believe this is just fine).

```
rc_2019 <- read_csv("https://data.dft.gov.uk/road-accidents-safety-data/dft-road-casualty-statistics-casualty-2019.csv", show_col_types = FALSE)
```

```r
rc_2020 <- read_csv("https://data.dft.gov.uk/road-accidents-safety-data/dft-road-casualty-statistics-casualty-2020.csv", show_col_types = FALSE)

rc_2021 <- read_csv("https://data.dft.gov.uk/road-accidents-safety-data/dft-road-casualty-statistics-casualty-2021.csv", show_col_types = FALSE)

rc_2022 <- read_csv("https://data.dft.gov.uk/road-accidents-safety-data/dft-road-casualty-statistics-casualty-2022.csv", show_col_types = FALSE)

rc_2023 <- read_csv("https://data.dft.gov.uk/road-accidents-safety-data/dft-road-casualty-statistics-casualty-2023.csv", show_col_types = FALSE)
```

All casualties

This is how you combine all years into one single data frame. You could only do it correctly because all these data frames had the same columns. If the data frames had had different sets of columns, you would have obtained a data frame containing all these columns and NA values in columns that were not present in all data frames.

> If you were working on your own, you should have better double-checked that you could combine the data frames. The easiest way would be checking this for column names of `r_2019` paired with each other year's data frame and delve deeper where you spot a problem.

```r
all.equal(colnames(rc_2019), colnames(rc_2020))

## [1] TRUE
```

> Nevertheless, you can rely on the annual data sets to be safe to be combined.

```r
all_rc <- dplyr::bind_rows(rc_2019, rc_2020, rc_2021, rc_2022, rc_2023)
```

If you add the data frames as named elements like below, you can generate an additional column that will contain the names. If you do not name the elements and just add a `.id` column, it will contain integers: `1` for the first data frame, `2` for the second, and so on.

```r
all_rc <- dplyr::bind_rows(rc_2019 = rc_2019, rc_2020 = rc_2020, rc_2021 = rc_2021, rc_2022 = rc_2022, rc_2023 = rc_2023, .id = "ID")
```

Inspect the big file

File structure

The most common ways to inspect a data frame are `str()` , `summary()`, and `glimpse()` .

```r
slice_sample(all_rc, n = 10) %>% str()

## spc_tbl_ [10 × 22] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ ID                                 : chr [1:10] "rc_2022" "rc_2022"
"rc_2023" "rc_2022" ...
##  $ accident_index                     : chr [1:10] "2022522300128"
```

```
"2022010358596" "2023052300534" "2022141239098" ...
##  $ accident_year                     : num [1:10] 2022 2022 2023 2022 2019
...
##  $ accident_reference                : chr [1:10] "522300128" "010358596"
"052300534" "141239098" ...
##  $ vehicle_reference                 : num [1:10] 2 1 1 2 1 1 1 1 1 2
##  $ casualty_reference                : num [1:10] 1 2 1 1 1 1 2 1 1 1
##  $ casualty_class                    : num [1:10] 1 3 1 2 1 1 2 3 1 1
##  $ sex_of_casualty                   : num [1:10] 1 1 2 2 1 1 2 1 1 2
##  $ age_of_casualty                   : num [1:10] 38 73 62 35 25 25 47 31
14 60
##  $ age_band_of_casualty              : num [1:10] 7 10 9 6 5 5 8 6 3 9
##  $ casualty_severity                 : num [1:10] 3 3 3 3 3 3 3 3 3 3
##  $ pedestrian_location               : num [1:10] 0 1 0 0 0 0 0 10 0 0
##  $ pedestrian_movement               : num [1:10] 0 3 0 0 0 0 0 9 0 0
##  $ car_passenger                     : num [1:10] 0 0 0 1 0 0 1 0 0 0
##  $ bus_or_coach_passenger            : num [1:10] 0 0 0 0 0 0 0 0 0 0
##  $ pedestrian_road_maintenance_worker: num [1:10] 0 0 0 0 0 0 0 0 0 0
##  $ casualty_type                     : num [1:10] 3 0 9 9 9 9 9 0 1 1
##  $ casualty_home_area_type           : num [1:10] 1 1 1 -1 1 -1 1 1 1 1
##  $ casualty_imd_decile               : num [1:10] 2 3 4 -1 1 -1 3 1 2 1
##  $ lsoa_of_casualty                  : chr [1:10] "E01014685" "E01002550"
"E01007000" "-1" ...
##  $ enhanced_casualty_severity        : num [1:10] -1 -1 -1 3 -1 3 -1 3 -1
3
##  $ casualty_distance_banding         : num [1:10] 1 1 1 -1 1 -1 5 1 1 2
##  - attr(*, "spec")=
##   .. cols(
##   ..   accident_index = col_character(),
##   ..   accident_year = col_double(),
##   ..   accident_reference = col_character(),
##   ..   vehicle_reference = col_double(),
##   ..   casualty_reference = col_double(),
##   ..   casualty_class = col_double(),
##   ..   sex_of_casualty = col_double(),
##   ..   age_of_casualty = col_double(),
##   ..   age_band_of_casualty = col_double(),
##   ..   casualty_severity = col_double(),
##   ..   pedestrian_location = col_double(),
##   ..   pedestrian_movement = col_double(),
##   ..   car_passenger = col_double(),
##   ..   bus_or_coach_passenger = col_double(),
##   ..   pedestrian_road_maintenance_worker = col_double(),
##   ..   casualty_type = col_double(),
##   ..   casualty_home_area_type = col_double(),
##   ..   casualty_imd_decile = col_double(),
##   ..   lsoa_of_casualty = col_character(),
##   ..   enhanced_casualty_severity = col_double(),
##   ..   casualty_distance_banding = col_double()
```

```
##   .. )
##  - attr(*, "problems")=<externalptr>
```

Whichever way you inspect the data frame, it turns out that most columns are numeric, although their names suggests categorical variables (e.g. `pedestrian_location`). You need to decode these digits. Therefore you manually inspect the source website to find a such a metadata file (see Fig. #guide).

Make labels comprehensible

They call the metadata file a *Road Safety Open Data Guide* and it is obviously an Excel () spreadsheet (more precisely *.xlsx*): https://data.dft.gov.uk/road-accidents-safety-data/dft-road-casualty-statistics-road-safety-open-dataset-data-guide-2023.xlsx . To inspect it, you need the `read_xlsx()` function from the `readxl` library (you have already loaded it with the other libraries).

Now, this function cannot read files from urls but requires a local file. Therefore you first must use the base R function `download.file()`.

Here comes a caveat for Windows users: you have to use this additional parameter: `mode = "wb"` . This is because Excel files behave like zipped files and to download these you have to explicitly tell Windows to store them as so-called *binary* files (with this parameter). Otherwise, the file will download corrupted.

```
download.file(url = "https://data.dft.gov.uk/road-accidents-safety-data/dft-road-casualty-statistics-road-safety-open-dataset-data-guide-2023.xlsx",
                destfile = "guide.xlsx",
                mode="wb",
                cacheOK = FALSE)
```

This ought to work. If you still experience problems with reading the file with the next code snippet, you will have to download and save the file manually through the Windows Explorer.

The `read_xlsx()` function can only read one worksheet at a time. It will read the first worksheet by default, but there is a parameter with which you could override the choice (`sheet`).

```
guide <- read_excel(path = "guide.xlsx", sheet = 1) # this function reads only local files, no urls
 glimpse(guide)

## Rows: 1,784
## Columns: 5
## $ table         <chr> "accident", "accident", "accident", "accident", "acciden…
## $ `field name`  <chr> "collision_index", "collision_year", "collision_referenc…
## $ `code/format` <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, "1", "3", "4", "…
```

```
## $ label          <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
"Metropolitan Po…
## $ note           <chr> "unique value for each accident. The accident_index
comb…
```

It seems that the most interesting columns are `field name` and `label`. The former will, hopefully, at least partially overlap with the column names of the annual casualty data. Only those that occur in the casualty data are relevant, so let us get rid of those that are irrelevant.

To find out, you have to compare the column names of `all_rc` with the de-duplicated `field name` column.

```
deduplicated <- guide %>%
  dplyr::distinct(`field name`) %>%
  pull() # outputs a character vector

colnames(all_rc) # also a character vector

##  [1] "ID"                                "accident_index"
##  [3] "accident_year"                     "accident_reference"
##  [5] "vehicle_reference"                 "casualty_reference"
##  [7] "casualty_class"                    "sex_of_casualty"
##  [9] "age_of_casualty"                   "age_band_of_casualty"
## [11] "casualty_severity"                 "pedestrian_location"
## [13] "pedestrian_movement"               "car_passenger"
## [15] "bus_or_coach_passenger"
"pedestrian_road_maintenance_worker"
## [17] "casualty_type"                     "casualty_home_area_type"
## [19] "casualty_imd_decile"               "lsoa_of_casualty"
## [21] "enhanced_casualty_severity"        "casualty_distance_banding"
```

You need to find all elements that occur both in `colnames(all_rc)` and in `field name` column in `guide`. If you made sure you have both as character vectors, you can use the `intersect()` function from base R. If you have loaded the `dplyr` library (you must have had to make this code work all the way down to here), you are going to get error messages. This is because `dplyr` also has a function with that name, and R is going to think you mean that one. Whenever you get error messages saying that a function is masked by another function, tell R explicitly the name of the library where your function belongs. Here it says `base::` for base R.

```
overlap <- base::intersect(colnames(all_rc), deduplicated)
overlap

##  [1] "accident_index"                    "accident_year"
##  [3] "accident_reference"                "vehicle_reference"
##  [5] "casualty_reference"                "casualty_class"
##  [7] "sex_of_casualty"                   "age_of_casualty"
##  [9] "age_band_of_casualty"              "casualty_severity"
## [11] "pedestrian_location"               "pedestrian_movement"
```

```
## [13] "car_passenger"                       "bus_or_coach_passenger"
## [15] "pedestrian_road_maintenance_worker" "casualty_type"
## [17] "casualty_home_area_type"             "casualty_imd_decile"
## [19] "lsoa_of_casualty"                    "enhanced_casualty_severity"
## [21] "casualty_distance_banding"
```

Get rid of all guide rows that capture labels that are not in the casualties data by filtering out all rows that contain these values in the `field_name` columns.

```
guide2 <- guide %>% dplyr::filter(`field name` %in% overlap )
```

That was a considerable reduction!

```
nrow(guide)

## [1] 1784

nrow(guide2)

## [1] 147
```

The `guide` lists to each `field name` a `code/format` and a `label` value. The `code/format` column displays the values that occur in the `all_rc` data and the `label` column displays their verbal explanations,which are short enough to make good labels in the `all_rc` data and replace the numeric codes.

We will pick these three variables in `all_rc` and decode their labels using `dplyr::mutate`:

- `sex_of_casualty`
- `age_of_casualty`
- `casualty_class`

You will need to look at their values and manually encode the mutation across the corresponding column, with the correct conditions. To break down this task, first make small data frames from `guide2` containing just these `format/codes` and their `labels`.

```
sex_of_casualty_df <- guide2 %>% filter(`field name` == "sex_of_casualty")
%>%
  select(`code/format`, label)
sex_of_casualty_df # add this "df" at the end so you can always tell it apart
from the like-namedcolumn

## # A tibble: 4 × 2
##   `code/format` label
##   <chr>         <chr>
## 1 1             Male
## 2 2             Female
## 3 9             unknown (self reported)
## 4 -1            Data missing or out of range
```

Create a new variable for experimentation so you don't have to croll up to re-run the code to re-create `all_rc` if something goes wrong.

```
all_rc2 <- all_rc %>%
  mutate(across(sex_of_casualty,
                ~ case_when(.x == "1" ~ "Male",
                            .x == "2" ~ "Female",
                            .x == "9" ~ "unknown (self reported)",
                            .x == "-1" ~ "Data missing or out of range",
                            .default = as.character(.x)
                            )
                ))
```

The added parameter `.default` was set to `.x`. This means that, if other values appear in the data, they will remain unchanged, but if they are numbers, they will become strings. We have said that R coerces the data classes, but in this context you could receive an error message on data class mismatch.

Check the result by displaying the de-duplicated values of `all_rc`'s `sex_of_casualty` column.

```
all_rc2 %>% dplyr::distinct(sex_of_casualty)

## # A tibble: 4 × 1
##   sex_of_casualty
##   <chr>
## 1 Male
## 2 Female
## 3 Data missing or out of range
## 4 unknown (self reported)
```

So this has worked fine, and the data is consistent with the guide, since no unexpected values have occurred.

The sex values are just a few, but imagine the chore to manually encode all values if they were many. To avoid retyping values by hand, you could look them up in the small data frame you created and use a loop like so:

```
all_rc2 <- all_rc # start with all_rc2 identical with all_rc
for (i in 1:nrow(sex_of_casualty_df)) {
  all_rc2 <- all_rc2 %>%
    mutate(across(sex_of_casualty,
                  ~ case_when(.x == sex_of_casualty_df$`code/format`[i] ~
sex_of_casualty_df$label[i],
                              .default = as.character(.x))))
}
```

Breaking down what is happening in the loop:

The number of iterations is the number of rows in the small data frame (`nrow(sex_of_casualty_df)`).

You mutate the all_rc2's sex_of_casualty column with the following formula: "if the value of the sex_of_casualty column in all_rc2 equals the value in the ith row of sex_of_casualty_df's column code/format, replace that value with the value in the ith row of sex_of_casualty_df's label column. Very important: you have to set the .default parameter to keep other values unchanged. Otherwise each step would only replace that one ith value and overwrite all others with NA.

Also, when you glimpse back at the structure of all_rc, it its sex_of_casualty column is numeric, while you are replacing numbers with strings. In a loop, this would throw errors even more likely than in the previous case. Therefore you have to convert the numeric value to character in .default. To live through what exactly would happen, you can modify the code and run it without the as_character trick. You are going to see class mismatch messages over and over when you code, so remember to make sure that you replace values by values of the same class - not only when running a loop but in many other contexts.

Here check the result again:

```
all_rc2 %>% dplyr::distinct(sex_of_casualty)

## # A tibble: 4 × 1
##   sex_of_casualty
##   <chr>
## 1 Male
## 2 Female
## 3 Data missing or out of range
## 4 unknown (self reported)
```

This has worked well again! You can run this code on the original all_rc.

```
for (i in 1:nrow(sex_of_casualty_df)) {
  all_rc <- all_rc %>%
    mutate(across(sex_of_casualty,
                  ~ case_when(.x == sex_of_casualty_df$`code/format`[i] ~
sex_of_casualty_df$label[i],
                              .default = as.character(.x))))
}
```

Now, for your training, you can try and replace values in the other columns age_of_casualty and casualty_class, either by retyping their values by hand or by creating small look-up data frame and glide over them with a loop. Feel free to experiment with other columns!

First insights from the data

Now that you have decoded the categories of all categorical variables you are interested in, you can start asking questions. Here come a few.

Count of casualties per accident

It seems that each row of the `all_rc` represents a person who got severely injured or killed in an accident. There is no column that would mark each such casualty person with a unique ID. There is a column called `casualty_reference`, but according to the `str()` call, the values repeat. A quick check here:

```
all_rc$casualty_reference %>% summary()

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    1.00    1.00    1.36    1.00  999.00

table(all_rc$casualty_reference)

##
##      1      2      3      4      5      6      7      8      9     10
11
## 514864 102232  30651  10818   3917   1396    604    266    144    104
78
##     12     13     14     15     16     17     18     19     20     21
22
##     57     37     31     22     17     15     11     11      7      6
7
##     23     24     25     26     27     28     29     30     31     32
33
##      4      4      6      3      3      3      3      3      3      3
4
##     34     35     36     37     38     39     40     41     42     43
44
##      3      3      3      3      3      3      4      4      2      2
2
##     45     46     47     48     49     50     51     52     53     54
55
##      2      2      2      2      2      2      2      2      1      1
1
##     56     57     58     59     60     61     62     63     64     65
66
##      1      1      1      1      1      1      1      1      1      1
1
##     67     68     69     70    101    111    256    902    991    992
999
##      1      1      1      1      1      1      1      1      1      2
1
```

Perhaps these reference indices can become unique IDs when combined with the unique IDs of accidents, which are in columns `accident index` and `accident reference`. The former seems to contain the latter and add encoded accident date, like here:

```
all_rc %>%
  select(c(`accident_index`, `accident_reference`, `casualty_reference`)) %>%
  slice_max(casualty_reference,n = 10, with_ties = FALSE)
```

```
## # A tibble: 10 × 3
##    accident_index accident_reference casualty_reference
##    <chr>          <chr>                           <dbl>
##  1 2022141207146  141207146                         999
##  2 2020501008741  501008741                         992
##  3 2023201458669  201458669                         992
##  4 2019410902098  410902098                         991
##  5 2020500986881  500986881                         902
##  6 2019470882287  470882287                         256
##  7 2019470879747  470879747                         111
##  8 2022052201813  052201813                         101
##  9 2023520300610  520300610                          70
## 10 2023520300610  520300610                          69
```

So, the idea is, that each accident listed in this table has at least one casualty referred to as 1. The highest value is 999, so you should get 999 rows of the same accident, shouldn't you? But that does not work, since an accident with a casualty referred to as 999 ought to produce rows with casualty 998, 997, 996 etc., all the way to 1. A possible explanation is that all accident participants get this index, and the casualties tables filter just the severely and deadly injured and rename the filtered `participants` (or so) column to `casualty_reference`. Anyway, let's look at the distribution of casualties in accidents to make a sanity check of this guess.

How to do that: group the data by `accident_index` and display counts.

```
all_rc %>% group_by(accident_index) %>%
  count(name = "casualties") %>%
  ungroup() %>%
  slice_max(casualties, with_ties = FALSE, n = 10)
```

```
## # A tibble: 10 × 2
##    accident_index casualties
##    <chr>               <int>
##  1 2023520300610          70
##  2 2019500885809          52
##  3 2020440349165          41
##  4 2019220855375          25
##  5 202163CF00721          22
##  6 2019350900122          20
##  7 2019410889448          19
##  8 2019440129002          19
##  9 2020990939366          19
## 10 2023991309739          19
```

Mind to ungroup before slicing!!! Otherwise you will get the entire data frame, because it is going to look for the top ten values for each accident index, but will obviously find just one for each because each accident occurs only once now that we have counted the rows where it occurred in the original data containing individual observations.

So this table says that the worst accident had 70 casualties.

Look at the accident with the highest index of casualty. First find out the accident index of that accident.

```
all_rc %>% slice_max(order_by = casualty_reference, n = 1, with_ties = TRUE)
```

```
## # A tibble: 1 × 22
##   ID      accident_index accident_year accident_reference
vehicle_reference
##   <chr>   <chr>                  <dbl> <chr>
<dbl>
## 1 rc_2022 2022141207146           2022 141207146
1
## # ℹ 17 more variables: casualty_reference <dbl>, casualty_class <dbl>,
## #   sex_of_casualty <chr>, age_of_casualty <dbl>, age_band_of_casualty
<dbl>,
## #   casualty_severity <dbl>, pedestrian_location <dbl>,
## #   pedestrian_movement <dbl>, car_passenger <dbl>,
## #   bus_or_coach_passenger <dbl>, pedestrian_road_maintenance_worker
<dbl>,
## #   casualty_type <dbl>, casualty_home_area_type <dbl>,
## #   casualty_imd_decile <dbl>, lsoa_of_casualty <chr>, …
```

```
all_rc %>% filter(accident_index == "2022141207146")
```

```
## # A tibble: 1 × 22
##   ID      accident_index accident_year accident_reference
vehicle_reference
##   <chr>   <chr>                  <dbl> <chr>
<dbl>
## 1 rc_2022 2022141207146           2022 141207146
1
## # ℹ 17 more variables: casualty_reference <dbl>, casualty_class <dbl>,
## #   sex_of_casualty <chr>, age_of_casualty <dbl>, age_band_of_casualty
<dbl>,
## #   casualty_severity <dbl>, pedestrian_location <dbl>,
## #   pedestrian_movement <dbl>, car_passenger <dbl>,
## #   bus_or_coach_passenger <dbl>, pedestrian_road_maintenance_worker
<dbl>,
## #   casualty_type <dbl>, casualty_home_area_type <dbl>,
## #   casualty_imd_decile <dbl>, lsoa_of_casualty <chr>, …
```

So, accident with this index seems to have just one casualty. How about accidents with highest casualty references?

```
all_rc %>% slice_max(order_by = casualty_reference, n = 5, with_ties = TRUE)
```

```
## # A tibble: 5 × 22
##   ID      accident_index accident_year accident_reference
vehicle_reference
##   <chr>   <chr>                  <dbl> <chr>
<dbl>
```

```
## 1 rc_2022 2022141207146          2022 141207146
1
## 2 rc_2020 2020501008741          2020 501008741
1
## 3 rc_2023 2023201458669          2023 201458669
2
## 4 rc_2019 2019410902098          2019 410902098
1
## 5 rc_2020 2020500986881          2020 500986881
2
## # ℹ 17 more variables: casualty_reference <dbl>, casualty_class <dbl>,
## #   sex_of_casualty <chr>, age_of_casualty <dbl>, age_band_of_casualty
<dbl>,
## #   casualty_severity <dbl>, pedestrian_location <dbl>,
## #   pedestrian_movement <dbl>, car_passenger <dbl>,
## #   bus_or_coach_passenger <dbl>, pedestrian_road_maintenance_worker
<dbl>,
## #   casualty_type <dbl>, casualty_home_area_type <dbl>,
## #   casualty_imd_decile <dbl>, lsoa_of_casualty <chr>, …

all_rc %>%
  filter(accident_index %in% c("2022141207146", "2020501008741",
                              "2023201458669", "2019410902098",
                              "2020500986881")) %>%
  select(c("accident_index", "casualty_reference")) %>%
arrange(accident_index)

## # A tibble: 10 × 2
##    accident_index casualty_reference
##    <chr>                       <dbl>
##  1 2019410902098                 991
##  2 2019410902098                   2
##  3 2020500986881                   1
##  4 2020500986881                 902
##  5 2020501008741                   1
##  6 2020501008741                   3
##  7 2020501008741                 992
##  8 2022141207146                 999
##  9 2023201458669                   1
## 10 2023201458669                 992
```

This shows that, for instance, there were two casualties in Accident 2019410902098 (the first one), referenced as 991 and 2. Other two casualties (referenced as 1 and 902) occurred in Accident 2020500986881. Three casualties occurred in Accident 2020501008741 under references 1, 3, and 992. And so on.

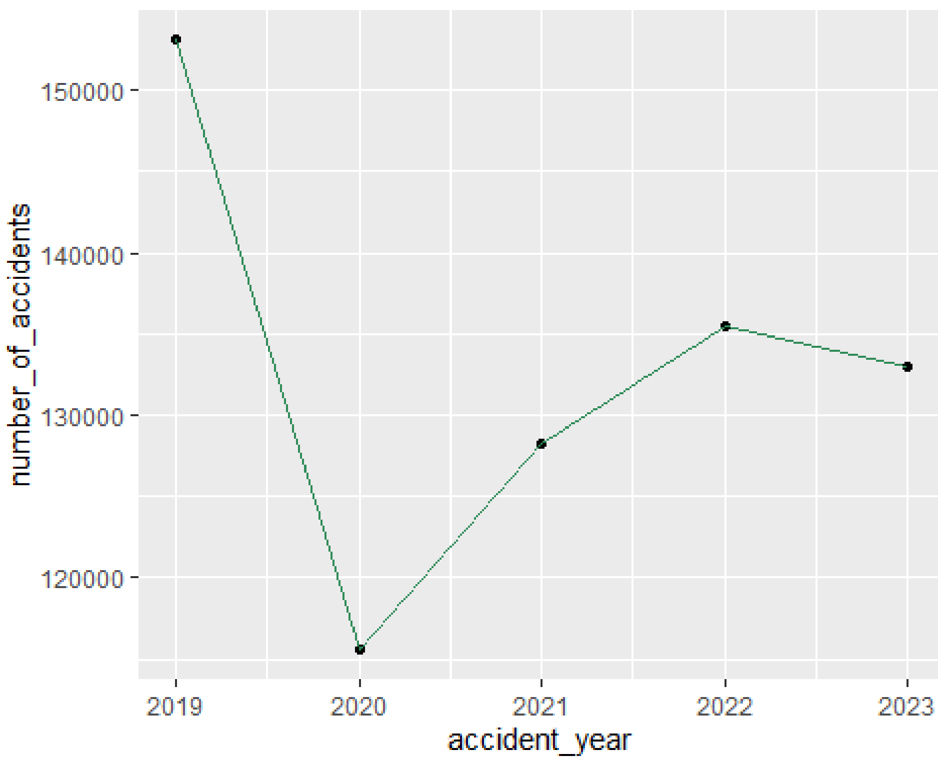
It is a plausible explanation that the casualty index is identical to what probably is an accident participant index in other data sets describing these accidents.

This data do not indicate how many persons were involved in the individual accidents, but it shows the number of casualties in each accident resulting in at least one casualty.

Visualizations of casualties counts

Plot the numbers of accidents broken by years in a scatterplot or a line plot.

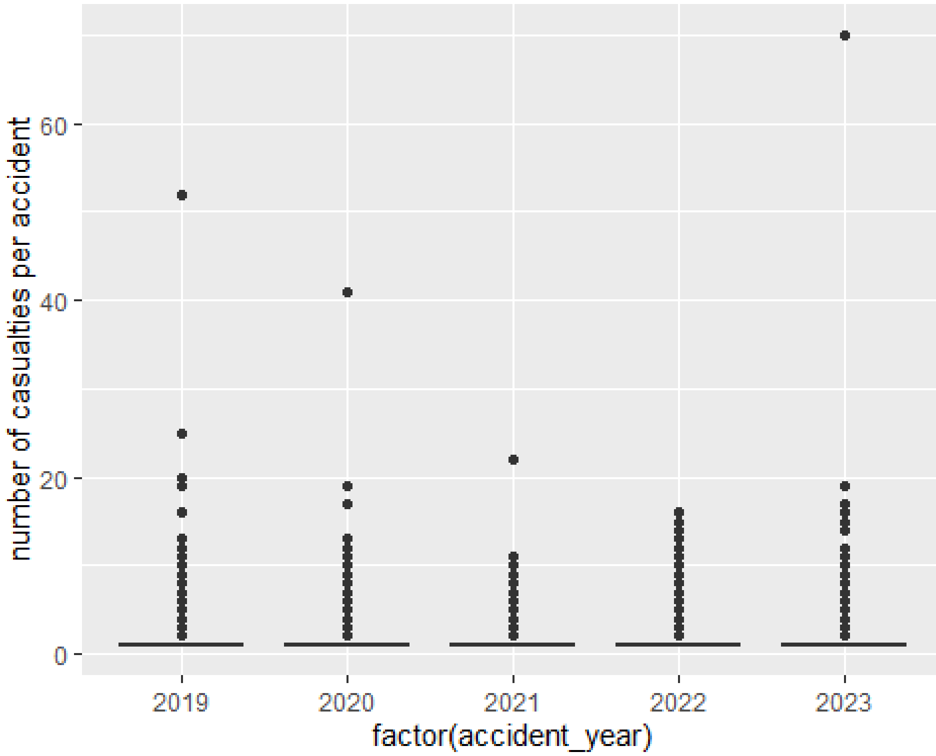
```
all_rc %>%
  group_by(accident_year) %>%
  count(name = "number_of_accidents") %>%
  ungroup() %>%
  ggplot(mapping = aes(x = accident_year, y = number_of_accidents)) +
  geom_point() +
  geom_line(color = "seagreen")
```



Draw a plot that would show for each year the distribution of numbers of casualties by years. For instance, you can draw a boxplot for each year (x-axis) with numbers of casualties per accident (y-axis). You will again have to aggregate the table accordingly before drawing the plot. Mind to ungroup the data frame to keep the code clean.

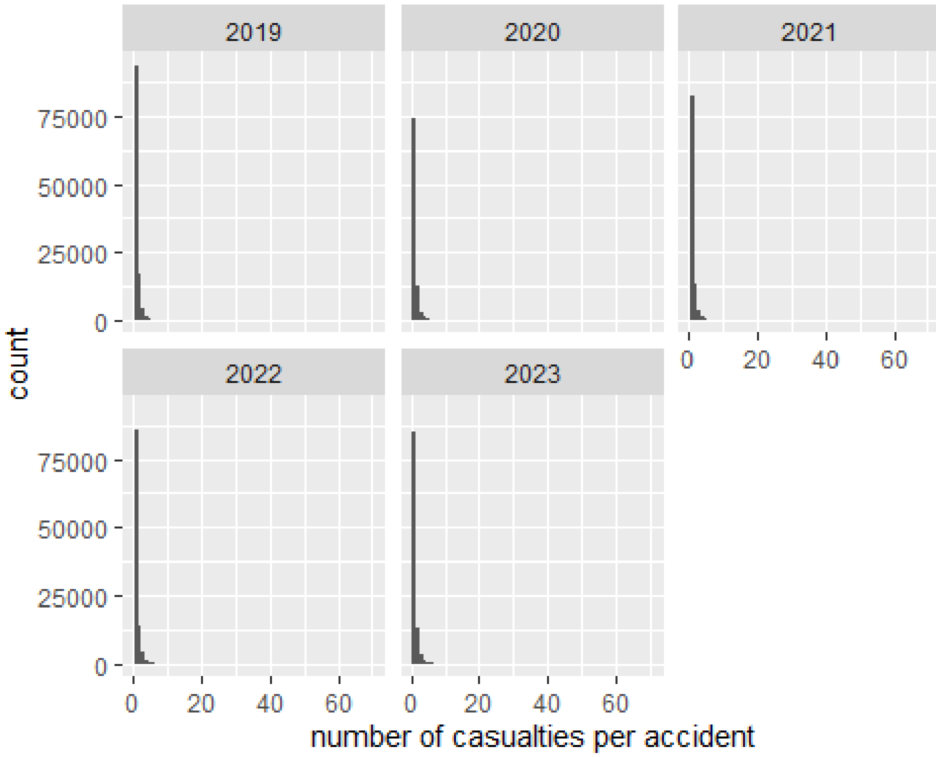
```
all_rc %>%
  group_by(accident_year, accident_index) %>%
  count(name = "number of casualties per accident") %>%
 ggplot(mapping = aes(x = factor(accident_year),
                      y = `number of casualties per accident`,
                      #group = accident_year
```

```
                           )) +
  geom_boxplot()
```



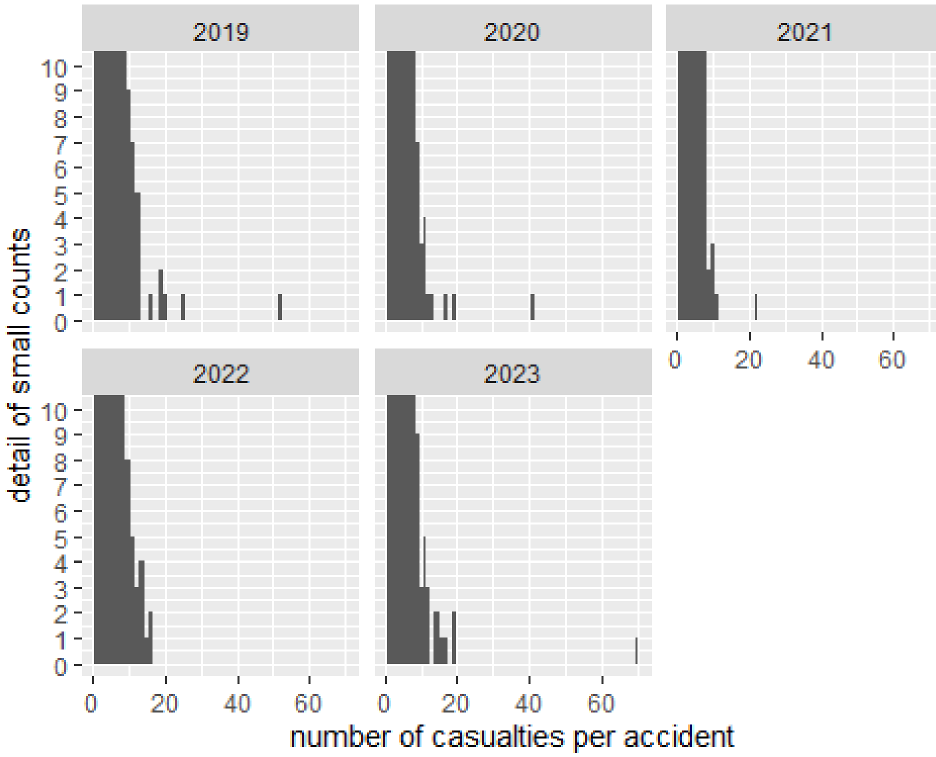
Another way would be to make facets with histograms

```
all_rc %>%
  group_by(accident_index, accident_year) %>%
  count(name = "number of casualties per accident") %>%
  ggplot(mapping = aes(x = `number of casualties per accident`)) +
  geom_histogram(binwidth = 1) +
  facet_wrap(~ accident_year)
```



Zoom in at the small numbers

```
all_rc %>%
  group_by(accident_index, accident_year) %>%
  count(name = "number of casualties per accident") %>%
  ggplot(mapping = aes(x = `number of casualties per accident`)) +
  geom_histogram(binwidth = 1) +
  facet_wrap(~ accident_year) +
  coord_cartesian(ylim = c(0,10)) +
  scale_y_continuous(name = "detail of small counts",
                     breaks = seq(0,10, 1)
                     )
```



2019
2020
2021
2022
2023
detail of small counts
number of casualties per accident