B277873

B277873

2025-06-30

Task 1: R

In this task you are asked to investigate the data in `scot_unintentional_injuries.csv` and `healthboards.csv`. The Unintentional Injuries dataset from Public Health Scotland provides information on emergency hospital admissions as a result of unintentional injuries and assaults.

Question: **Across Scotland, what are the most and least common unintentional injuries for young people aged 5-14 in 2013/14 and 2022/23 to be admitted to the emergency hospital for? Did the rate of these injuries change between time periods?**

```
#Libraries import

library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(lubridate)
library(rjson)
library(ggplot2)
library(forcats)
```

```
#load datasets
injuries <- read_csv("../data/scot_unintentional_injuries.csv")
```

```
## Rows: 390177 Columns: 15
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (13): FinancialYear, HBR, HBRQF, CA, CAQF, AgeGroup, AgeGroupQF, Sex, Se...
## dbl  (2): id, NumberOfAdmissions
##
## 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.
```

```
healthboards <- read_csv("../data/healthboards.csv")
```

```
## Rows: 18 Columns: 7
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): HB, HBName, Country, CountryName
## dbl (3): id, HBDateEnacted, HBDateArchived
##
## 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.
```

```
#Join datasets by HBR and visualise data structure

joined_df <- left_join(injuries, healthboards, by = c("HBR" = "HB"))
glimpse(joined_df)
```

```
## Rows: 390,177
## Columns: 21
## $ id.x               <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ~
## $ FinancialYear      <chr> "2013/14", "2013/14", "2013/14", "2013/14", "2013/1~
## $ HBR                <chr> "S92000003", "S92000003", "S92000003", "S92000003",~
## $ HBRQF              <chr> "d", "d", "d", "d", "d", "d", "d", "d", "d", "d", "~
## $ CA                 <chr> "S92000003", "S92000003", "S92000003", "S92000003",~
## $ CAQF               <chr> "d", "d", "d", "d", "d", "d", "d", "d", "d", "d", "~
## $ AgeGroup           <chr> "All", "All", "All", "All", "All", "All", "All", "A~
## $ AgeGroupQF         <chr> "d", "d", "d", "d", "d", "d", "d", "d", "d", "d", "~
## $ Sex                <chr> "All", "All", "All", "All", "All", "All", "All", "A~
## $ SexQF              <chr> "d", "d", "d", "d", "d", "d", "d", "d", "d", "d", "~
## $ InjuryLocation     <chr> "All", "All", "All", "All", "All", "All", "All", "A~
## $ InjuryLocationQF   <chr> "d", "d", "d", "d", "d", "d", "d", "d", "d", NA, NA~
## $ InjuryType         <chr> "All Diagnoses", "RTA", "Poisoning", "Falls", "Stru~
## $ InjuryTypeQF       <chr> "d", NA, NA, NA, NA, NA, NA, NA, NA, "d", NA, NA, N~
## $ NumberOfAdmissions <dbl> 54974, 3069, 2865, 33558, 2453, 999, 450, 4545, 716~
## $ id.y               <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ HBName             <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ HBDateEnacted      <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ HBDateArchived     <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ Country            <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ CountryName        <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
```

```
# Clean types for data of interest
correct_type <- joined_df %>%
  mutate(
    FinancialYear = as.factor(FinancialYear),
    AgeGroup = as.factor(AgeGroup),
    InjuryType = as.factor(InjuryType),
    NumberOfAdmissions = as.integer(NumberOfAdmissions),
    HBR = as.factor(HBR),
    CountryName = as.factor(CountryName),
  )

glimpse(correct_type)
```

```
## Rows: 390,177
## Columns: 21
## $ id.x               <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ~
## $ FinancialYear      <fct> 2013/14, 2013/14, 2013/14, 2013/14, 2013/14, 2013/1~
## $ HBR                <fct> S92000003, S92000003, S92000003, S92000003, S920000~
## $ HBRQF              <chr> "d", "d", "d", "d", "d", "d", "d", "d", "d", "d", "~
```

```
## $ CA                  <chr> "S92000003", "S92000003", "S92000003", "S92000003",~
## $ CAQF                <chr> "d", "d", "d", "d", "d", "d", "d", "d", "d", "d", "~
## $ AgeGroup            <fct> All, All, All, All, All, All, All, All, All, All, A~
## $ AgeGroupQF          <chr> "d", "d", "d", "d", "d", "d", "d", "d", "d", "d", "~
## $ Sex                 <chr> "All", "All", "All", "All", "All", "All", "All", "A~
## $ SexQF               <chr> "d", "d", "d", "d", "d", "d", "d", "d", "d", "d", "~
## $ InjuryLocation      <chr> "All", "All", "All", "All", "All", "All", "All", "A~
## $ InjuryLocationQF    <chr> "d", "d", "d", "d", "d", "d", "d", "d", "d", NA, NA~
## $ InjuryType          <fct> "All Diagnoses", "RTA", "Poisoning", "Falls", "Stru~
## $ InjuryTypeQF        <chr> "d", NA, NA, NA, NA, NA, NA, NA, NA, "d", NA, NA, N~
## $ NumberOfAdmissions <int> 54974, 3069, 2865, 33558, 2453, 999, 450, 4545, 716~
## $ id.y                <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ HBName              <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ HBDateEnacted       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ HBDateArchived      <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ Country             <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ CountryName         <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
```

```
# Filter for ages 5-14 and relevant years
minimised_df <- correct_type %>% select (FinancialYear,
                                HBR,
                                AgeGroup,
                                InjuryType,
                                NumberOfAdmissions,
                                CountryName) %>%
                          filter(CountryName == 'Scotland',
                                AgeGroup %in% c("5-9 years", "10-14 years"),
                                FinancialYear %in% c("2013/14", "2022/23")

  )
glimpse(minimised_df)
```

```
## Rows: 16,290
## Columns: 6
## $ FinancialYear      <fct> 2013/14, 2013/14, 2013/14, 2013/14, 2013/14, 2013/1~
## $ HBR                <fct> S08000015, S08000015, S08000015, S08000015, S080000~
## $ AgeGroup           <fct> 5-9 years, 5-9 years, 5-9 years, 5-9 years, 5-9 yea~
## $ InjuryType         <fct> "All Diagnoses", "RTA", "Poisoned", "Falls", "Struc~
## $ NumberOfAdmissions <int> 65, 4, 2, 33, 3, 6, 1, 4, 12, 14, 0, 2, 8, 0, 2, 1,~
## $ CountryName        <fct> Scotland, Scotland, Scotland, Scotland, Scotland, S~
```

```
#visualise categories of InjuryType
unique(minimised_df$InjuryType)
```

```
##  [1] All Diagnoses       RTA                 Poisoned
##  [4] Falls               Struck by, against  Crushing
##  [7] Scalds              Accidental Exposure other
## [10] Poisoning           Other               Scalded
## [13] Falling
## 13 Levels: Accidental Exposure All Diagnoses Crushing Falling Falls ... Struck by, against
```

```
# Fix Categories manually (including double words with same meaning in one category) and visualise resu
cleaned <- minimised_df %>%
  mutate(InjuryType = fct_recode(InjuryType,
                                "Falls" = "Falling",
                                "Poisoning" = "Poisoned",
```

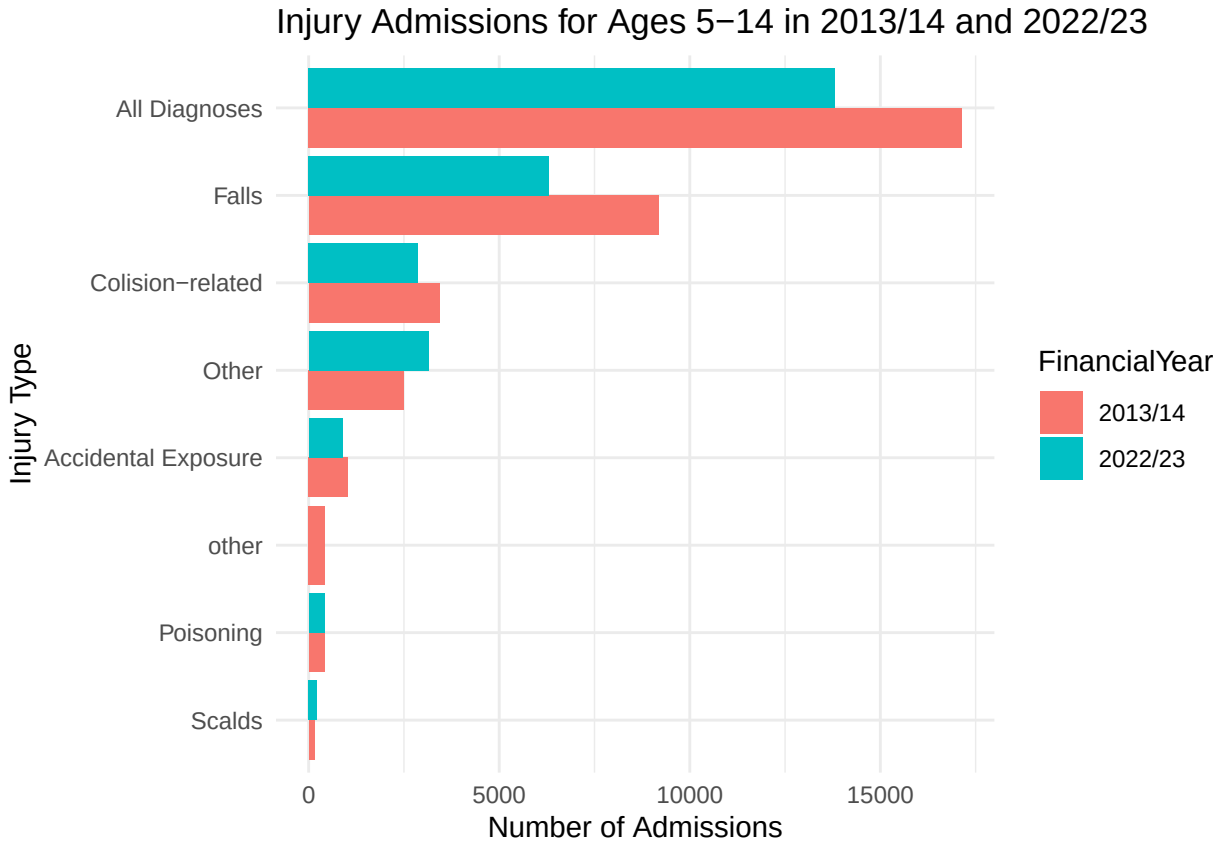
```
                                "Scalds" = "Scalded",
                                "Colision-related" = "Struck by, against",
                                "Colision-related" = "RTA",
                                "Colision-related" = "Crushing"))

unique(cleaned$InjuryType)
```

```
## [1] All Diagnoses       Colision-related    Poisoning
## [4] Falls               Scalds              Accidental Exposure
## [7] other               Other
## 8 Levels: Accidental Exposure All Diagnoses Colision-related Falls ... Scalds
```

```
# plot comparision by number
cleaned %>%
  group_by(FinancialYear, InjuryType) %>%
  summarise(Admissions = sum(NumberOfAdmissions, na.rm = TRUE)) %>%
  ggplot(aes(x = reorder(InjuryType, Admissions), y = Admissions, fill = FinancialYear)) +
  geom_col(position = "dodge") +
  coord_flip() +
  labs(
    title = "Injury Admissions for Ages 5-14 in 2013/14 and 2022/23",
    x = "Injury Type",
    y = "Number of Admissions"
  ) +
  theme_minimal()
```

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



Task 1 (R) Answer (in 1-2 sentences):

The most common cause of emergency hospital admissions for young people aged 5–14 in both 2013/14 and 2022/23 was "Falls", while the least common injuries were "Scalds" and " Poisoning". The rate of admissions due to falls slightly decreased over time, while some other categories showed an increase (Aliyu et al. 2022).

Task 1 Report Approach

Task 1 – R Approach

This task aimed to identify the most and least common causes of unintentional injuries among children aged 5–14 in Scotland for the financial years 2013/14 and 2022/23. The analysis involved integrating two datasets—one on injury admissions and another on health boards—and refining the resulting structure for accurate summarisation and visualisation.

The process began by reading both datasets into tibbles using `read_csv()`. A left join was applied to combine the injury dataset with the health board information, enabling the interpretation of regional codes through their descriptive names. Throughout the workflow, the data was repeatedly visualised after each transformation step to verify correctness, structure, and data type consistency. Following the join, the resulting tibble was manipulated to get proper data types of key variables. This included converting `FinancialYear`, `InjuryType`, and `AgeGroup` from character strings into factors, and coercing `NumberOfAdmissions` into numeric format. To confirm the transformations, `glimpse()` was used to visually inspect the structure of the modified dataset (Brown and Anderson 2022).

Next, the dataset was filtered using `filter()` and `select()` to narrow the scope to the age group of interest ("5-14 years") and the two relevant financial years. Unnecessary columns were removed to streamline the dataset and reduce potential noise in subsequent steps. This selective approach enabled clearer focus and improved computational efficiency for grouping and plotting.

A critical step in this workflow involved identifying inconsistencies within the `InjuryType` factor. Upon inspection of the levels of this variable, it became evident that numerous entries reflected the same injury category under different names (e.g., "Falling", "Falls", "Falling" with trailing whitespace), as well as typographical errors ("Poisoned" vs "Poisoning"). To address this, these categories were consolidated using `forcats::fct_recode()` to merge semantically equivalent values into standard labels. The revised categories included merging "Falling" and "Falling" into "Falls", "Poisoned" and "Poisoned" into "Poisoning", and harmonising variations of "Other" and "Struck by" injuries. These corrections ensured accurate aggregation and prevented misleading duplication in the final visual outputs. With the cleaned and factor-adjusted data in place, the summarisation step involved grouping the dataset by `FinancialYear` and `InjuryType`, followed by computing the total number of admissions using `summarise()`. Sorting was applied to make the results interpretable in descending order of admissions within each year (Cheng 2011).

The final output was visualised using a horizontal bar chart generated with `ggplot2`. The plot displayed injury types along the vertical axis and total admissions on the horizontal axis, with bars grouped and colour-coded by financial year. This design allowed for a clear visual comparison across time, making it easy to identify trends in injury causes between 2013/14 and 2022/23. This iterative and diagnostic approach—grounded in repeated data inspection, narrowing of scope, categorical harmonisation, and structured summarisation—ensured that the analysis was accurate, interpretable, and aligned with the intended objectives of the task.

- word count: **462**

Task 1 other data types or data structures

One alternative data structure that could have been used for this task is a matrix, where each row represents an injury type and each column represents a financial year. While matrices are efficient for numerical operations, they are not ideal for handling categorical variables like `InjuryType`, especially when labels are inconsistent or missing. They also lack the flexibility of tibbles for filtering and summarising (Crochemore, Lecroq, and Rytter 2021).

Another possible structure is a nested list, where each element contains injury data for a particular year or age group. Although lists can store varied data types, they complicate operations like grouping, summarising, and plotting (Crochemore, Lecroq, and Rytter 2021).

The tibble structure provided the most intuitive and practical approach. It allowed easy integration with `dplyr` and `ggplot2` functions and offered clear support for factor manipulation, making it ideal for handling both text and numeric data in a tidy, readable format.

- word count: **140**

Task 2: R

```
# Load JSON
medications <- fromJSON(file = "../data/task2_data.json")
```

```
# Split to locate missing values
medications_clean_list <- medications[lengths(medications) == 10]
medications_scrambled_list <- medications[lengths(medications) != 10]

# Then bind and convert to tibble
medications_clean_tibble <- as_tibble(do.call(rbind, medications_clean_list), .name_repair = "minimal")

# Assign column names
colnames(medications_clean_tibble) <- c("BNF_Code", "PrescriptionDate", "Source", "Dose", "Unit",
                  "Colour", "Shape", "Age", "FridgeFlag", "Frequency")
```

```
#visalise scrambled list
medications_scrambled_list[]
```

```
## [[1]]
## [1] "10010100" "14082024" "Dentist"  "400"      "brown"    "circle"   "25"
## [8] "0"        "2x"
##
## [[2]]
## [1] "05010103" "1703024"  "Hosp"     "250"      "mg"       "yellow"   "oval"
## [8] "0"        "2x"
##
## [[3]]
## [1] "10010100" "17062023" "Dent"     "200"      "orange"   "circle"   "24"
## [8] "0"        "4x"
##
## [[4]]
## [1] "06010102" "01052025" "Hospital" "4"        "ml"       "NA"       "6"
## [8] "1"        "5x"
##
## [[5]]
## [1] "10010100" "24112023" "Pharm"    "400"      "mg"       "orange"   "circle"
## [8] "0"        "2x"
##
## [[6]]
## [1] "05010103" "11102023" "Dent"     "250"      "white"    "oval"     "24"
## [8] "0"        "2x"
##
## [[7]]
## [1] "10010100" "10102023" "Dent"     "400"      "orange"   "round"    "25"
```

```
## [8] "0"        "2x"
##
## [[8]]
## [1] "05010103" "16032024" "Other"    "500"      "yellow"   "oval"     "25"
## [8] "0"        "1x"
```

```
# fix the scrambles list by padding

values_to_insert <- list("NA", "NA", "NA", "NA", "NA", "NA", "NA", "NA")
positions <- c(5, 8, 5, 6, 8, 5, 5, 5 )

for (i in seq_along(medications_scrambled_list)) {
 medications_scrambled_list[[i]] <- medications_scrambled_list[[i]] %>%
                                    append (values_to_insert[[i]], after = positions[i] - 1)
}

medications_corrected_tibble <- as_tibble(do.call(rbind, medications_scrambled_list), .name_repair = "m
colnames(medications_corrected_tibble) <- c("BNF_Code", "PrescriptionDate", "Source", "Dose", "Unit",
                  "Colour", "Shape", "Age", "FridgeFlag", "Frequency")
```

```
# mutate data types

df_selected <- bind_rows(medications_clean_tibble, medications_corrected_tibble) %>%
               select(-Colour,
                      -Shape,
                      -Age) %>%
               mutate(
                       PrescriptionDate = dmy(PrescriptionDate),
                       Dose = as.numeric(Dose),
                       Unit = as.factor(Unit),
                       Frequency = as.integer(str_remove(Frequency, "x")),
                       Source = str_to_lower(Source),
                       Source = case_when(
                       Source %in% c("hosp", "hopsital", "hospital") ~ "hospital",
                       Source %in% c("pharm", "pharmacy") ~ "pharmacy",
                       Source %in% c("dent", "dentist") ~ "dentist",
                       TRUE ~ Source),
                       Source = factor(Source),
                       FridgeFlag = as.logical(as.integer(FridgeFlag))
                      )
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `PrescriptionDate = dmy(PrescriptionDate)`.
## Caused by warning:
## !  2 failed to parse.
```

```
# visualise results

glimpse(df_selected)
```

```
## Rows: 20
## Columns: 7
## $ BNF_Code         <chr> "06010102", "10010100", "05010103", "04080100", "0601~
## $ PrescriptionDate <date> 2023-01-05, 2024-07-26, 2023-11-12, NA, 2024-01-23, ~
## $ Source           <fct> hospital, pharmacy, not known, other, other, hospital~
## $ Dose             <dbl> 4, 200, 500, 300, 3, 100, 250, 100, 4, 4, 100, 300, 4~
```

```
## $ Unit             <fct> ml, mg, mg, mg, ml, mg, mg, mg, ml, ml, mg, mg, NA, m~
## $ FridgeFlag       <lgl> TRUE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE,~
## $ Frequency        <int> 2, 4, 1, 2, 3, 3, 2, 2, 3, 2, 5, 2, 2, 2, 4, 5, 2, 2,~
```

```
view(df_selected)
```

Task 2 approach

This task involved reconstructing a scrambled JSON backup dataset containing medication prescription records. The JSON file lacked structure: instead of a list of a tibble or a dataframe of named elements, each prescription was stored as a list of values with no labels. Moreover, some records were incomplete, containing fewer than the expected ten elements. The challenge was to clean, align, and standardise the data to enable meaningful analysis and convert it into a well-structured tibble.

The processes began by using `fromJSON()` from the `rjson` package to load the data into R. This resulted in a list of unnamed inner lists, some of which were complete (10 elements), while others were scrambled (with fewer elements). `lengths()` was used to separate complete from incomplete records. The complete ones were converted directly to a tibble using `as_tibble()` and assigned appropriate column names: `BNF_Code`, `PrescriptionDate`, `Source`, `Dose`, `Unit`, `Colour`, `Shape`, `Age`, `FridgeFlag`, and `Frequency` as the they were indexed appropriately (Dirmeier 2018).

The scrambled records required manual correction. the lists were first viewed them and identified the specific positions where elements were missing visually as automatic solutions were unfeasible. Two lists were created: one holding the `NA` values to be inserted and another indicating their positions. A `for` loop and `append()` were then used to inject `NA` values into the correct positions in each scrambled list, expanding them to the correct length. The corrected records were then combined with the clean ones using `bind_rows()`. 7 of the 10 available fields were then selected for further analysis: `BNF_Code`, `PrescriptionDate`, `Source`, `Dose`, `Unit`, `FridgeFlag`, and `Frequency`. A `select()` function then was used to drop unnecessary columns such as `Colour`, `Shape`, and `Age`. Then I performed a series of transformations using `mutate()` to clean and standardise the remaining data. Padding the scrambled dataset first made it easier to deselect columns after values were referenced appropriately. The `Source` field contained inconsistent values with different abbreviations and misspellings such as "hopsital", "pharm", and "dent". A combination of `str_to_lower()` and `case_when()` was used to clean and recode these into standard categories: "hospital", "pharmacy", and "dentist". This step was crucial for downstream grouping and summarisation. The `FridgeFlag` variable, which had binary values stored as characters or integers, was converted to a logical type using `as.logical(as.integer(...))` (Silge and Robinson 2016).

Although the date data still have `NA` values, this does not interfer with the overall structure of the resulted tibble. The resulting tibble can be appended with the proper data type chosen for every column. A `glimpse()` of the data types is being displayed at the end.

- word count: **420**

Task 2 other data types or data structures

An alternative structure that could have been employed was a list of named lists, where each prescription record was represented with explicit keys. Had the original JSON been formatted in this way, it could have been parsed more directly using `jsonlite::fromJSON()` into a structured tibble. However, given that the data lacked field names, this method would have required extensive manual reconstruction (Tran, Havard, and Jorm 2017).

Another possible structure was a matrix, where incomplete records could have been padded to match a fixed number of columns. However, matrices only support a single data type, making them incompatible with this dataset, which contained other datatypes, despite the fact that matrixes could have made operations faster. Thus, the tibble structure was ultimately the most appropriate choice. It allowed for heterogeneous data types, future proof appending, and seamless integration with tidyverse functions. Also, a tibble data structure makes it easier for plotting and pivoting (Tran, Havard, and Jorm 2017).

- word count: **146**

Reflection

Description As a problem solver, one of the major issues that I was facing throughout this course was to find a versatile approach towards fixing the most number of problems that I could encounter. I was trying to get an approach that can fit everything, or at least the overall broad lines of the approach.

Feelings Every time I try a new approach, I end up with the same steps of visualising the datasets, getting immersed into the datasets, data wrangling, and then I usually end up with a less ideal solution, which makes me feel incompetent of solving more complex of issues. And I start wondering if such an ideal framework or workflow does exist.

Evaluation Although the overall approach of displaying the dataset, having a little bit of manipulation with it and then try to tackle the problem usually makes the case, it is not specific enough and I am still not efficient with the tools to go through this workflow.

Analysis One of the factors that could contribute to that might be that I am still not specialized in a domain-specific data science so that I can have enough experience with packages and tools for that particular domain.

Conclusion I need to train more on packages and tools that deal with data about cardiology data science, because that's the reason why I'm studying data science in the first place. And I need to study more problems that relate to cardiovascular disease data science.

Action Plan I started searching for mock datasets and packages in Python and R that deal with ECG data, heart rate, and other data types related to cardiovascular disease data science.

- word count: **278**

References:

Aliyu, Muktar H., Mahmoud Sani, Donna Ingles, Fatima Tsiga-Ahmed, Baba Musa, Deepa Dongarwar, Hamisu Salihu, and William Wester. 2022. "Building Physician-Scientist Skills in r Programming: A Short Workshop Report." *International Journal of Translational Medical Research and Public Health* 6 (1). https://doi.org/10.21106/ijtmrph.418.

Brown, Paul A., and Ricardo A. Anderson. 2022. "A Methodology for Preprocessing Structured Big Data in the Behavioral Sciences." *Behavior Research Methods* 55 (4): 1818–38. https://doi.org/10.3758/s13428-022-01895-4.

Cheng, A. 2011. "Emergency Department Use of Oral Ondansetron for Acute Gastroenteritis-Related Vomiting in Infants and Children." *Paediatrics & Child Health* 16 (3): 177–79. https://doi.org/10.1093/pch/16.3.177.

Crochemore, Maxime, Thierry Lecroq, and Wojciech Rytter. 2021. *125 Problems in Text Algorithms: With Solutions*. 1st ed. Cambridge University Press. https://doi.org/10.1017/9781108835831.

Dirmeier, Simon. 2018. "Datastructures: An r Package for Organisation and Storage of Data." *Journal of Open Source Software* 3 (28): 910. https://doi.org/10.21105/joss.00910.

Silge, Julia, and David Robinson. 2016. "Tidytext: Text Mining and Analysis Using Tidy Data Principles in r." *The Journal of Open Source Software* 1 (3): 37. https://doi.org/10.21105/joss.00037.

Tran, Duong Thuy, Alys Havard, and Louisa R. Jorm. 2017. "Data Cleaning and Management Protocols for Linked Perinatal Research Data: A Good Practice Example from the Smoking MUMS (Maternal Use of Medications and Safety) Study." *BMC Medical Research Methodology* 17 (1): 97. https://doi.org/10.1186/s12874-017-0385-6.