Task 1: Python

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. You can find more information about the unintentional injuries data set here, including a data dictionary and here for more information about the health boards dataset. Do not import the data via the URL, you must use the data files provided for this assignment.

Follow the steps below to ultimately answer the final questions with the data.

- load the necessary packages and modules
- read in the data
- join the two data sets together
- check the data types of the variables of interest and ensure they are they type you want them to be. If not, change them.
- check for the content/information contained within your variables of interest and ensure you are satisfied with the presentation
- check for and deal with any missing data, if deemed appropriate
- check the data formats (wide vs long) and transform if it is not appropriate

Question: **Among people aged 65 and older in 2020/21, did NHS Lothian and NHS Greater Glasgow and Clyde have the same 4 most common cause of accidental injuries leading to emergency hospital admissions? What were they (the 4 most common cause of accidental injury in these regions)?**

In [1]:
```
# imports
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#load the necessary data
injuries = pd.read_csv('../data/scot_unintentional_injuries.csv')
healthboards = pd.read_csv('../data/healthboards.csv')
```

In [3]:
```
# Detailed info about structure
print("Injuries DataFrame Info:")
injuries.info()

print("\nHealthboards DataFrame Info:")
healthboards.info()
```

```
Injuries DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 390177 entries, 0 to 390176
Data columns (total 15 columns):
 #   Column              Non-Null Count   Dtype 
---  ------              --------------   ----- 
 0   id                  390177 non-null  int64 
 1   FinancialYear       390177 non-null  object
 2   HBR                 390177 non-null  object
 3   HBRQF               12150 non-null   object
 4   CA                  390177 non-null  object
 5   CAQF                12150 non-null   object
 6   AgeGroup            390177 non-null  object
 7   AgeGroupQF          44541 non-null   object
 8   Sex                 390177 non-null  object
 9   SexQF               132057 non-null  object
 10  InjuryLocation      390177 non-null  object
 11  InjuryLocationQF    80091 non-null   object
 12  InjuryType          390177 non-null  object
 13  InjuryTypeQF        43353 non-null   object
 14  NumberOfAdmissions  390177 non-null  int64 
dtypes: int64(2), object(13)
memory usage: 44.7+ MB

Healthboards DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              18 non-null     int64  
 1   HB              18 non-null     object 
 2   HBName          18 non-null     object 
 3   HBDateEnacted   18 non-null     int64  
 4   HBDateArchived  4 non-null      float64
 5   Country         18 non-null     object 
 6   CountryName     18 non-null     object 
dtypes: float64(1), int64(2), object(4)
memory usage: 1.1+ KB
```

In [4]:
```
# Merge and cast relevant columns
merged_data = injuries.merge(healthboards, left_on='HBR', right_on='HB',
merged_data['NumberOfAdmissions'] = pd.to_numeric(merged_data['NumberOfAd
merged_data['FinancialYear'] = merged_data['FinancialYear'].astype('categ
merged_data['InjuryType'] = merged_data['InjuryType'].astype('category')
merged_data['AgeGroup'] = merged_data['AgeGroup'].astype('category')
merged_data['HBName'] = merged_data['HBName'].astype('category')
```

In [5]:
```
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 390177 entries, 0 to 390176
Data columns (total 22 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   id_x                390177 non-null  int64
 1   FinancialYear       390177 non-null  category
 2   HBR                 390177 non-null  object
 3   HBRQF               12150 non-null   object
 4   CA                  390177 non-null  object
 5   CAQF                12150 non-null   object
 6   AgeGroup            390177 non-null  category
 7   AgeGroupQF          44541 non-null   object
 8   Sex                 390177 non-null  object
 9   SexQF               132057 non-null  object
 10  InjuryLocation      390177 non-null  object
 11  InjuryLocationQF    80091 non-null   object
 12  InjuryType          390177 non-null  category
 13  InjuryTypeQF        43353 non-null   object
 14  NumberOfAdmissions  390177 non-null  int64
 15  id_y                378027 non-null  float64
 16  HB                  378027 non-null  object
 17  HBName              378027 non-null  category
 18  HBDateEnacted       378027 non-null  float64
 19  HBDateArchived      0 non-null       float64
 20  Country             378027 non-null  object
 21  CountryName         378027 non-null  object
dtypes: category(4), float64(3), int64(2), object(13)
memory usage: 55.1+ MB
```

In [6]:
```
# Filter (scoped to the question)
subset = merged_data[
    (merged_data['AgeGroup'].isin(['65-74 years', '75plus years'])) &
    (merged_data['FinancialYear'] == '2020/21') &
    (merged_data['HBName'].isin(['NHS Lothian', 'NHS Greater Glasgow and
]
subset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2682 entries, 291420 to 310256
Data columns (total 22 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id_x                2682 non-null   int64
 1   FinancialYear       2682 non-null   category
 2   HBR                 2682 non-null   object
 3   HBRQF               0 non-null      object
 4   CA                  2682 non-null   object
 5   CAQF                0 non-null      object
 6   AgeGroup            2682 non-null   category
 7   AgeGroupQF          0 non-null      object
 8   Sex                 2682 non-null   object
 9   SexQF               900 non-null    object
 10  InjuryLocation      2682 non-null   object
 11  InjuryLocationQF    540 non-null    object
 12  InjuryType          2682 non-null   category
 13  InjuryTypeQF        298 non-null    object
 14  NumberOfAdmissions  2682 non-null   int64
 15  id_y                2682 non-null   float64
 16  HB                  2682 non-null   object
 17  HBName              2682 non-null   category
 18  HBDateEnacted       2682 non-null   float64
 19  HBDateArchived      0 non-null      float64
 20  Country             2682 non-null   object
 21  CountryName         2682 non-null   object
dtypes: category(4), float64(3), int64(2), object(13)
memory usage: 410.6+ KB
```

In [7]:
```
# make sure values exist
assert 'NHS Lothian' in merged_data['HBName'].values
assert 'NHS Greater Glasgow and Clyde' in merged_data['HBName'].values
```

In [8]:
```
#examine categories
subset['InjuryType'].cat.categories
```

Out[8]:
```
Index(['Accidental Exposure', 'All Diagnoses', 'Crushing', 'Falling',
       'Falling ', 'Falls', 'Other', 'Poisoned', 'Poisoned ', 'Poisonin
g',
       'RTA', 'Scalded', 'Scalds', 'Struck by, against', 'other', 'other
'],
      dtype='object')
```

In [9]:
```
#fix categories then check

subset['InjuryType'] = (
    subset['InjuryType']
    .str.strip()  # remove leading/trailing spaces
    .str.lower()
    .str.title()
    .replace({
        'Falling': 'Falls',
        'Falling ': 'Falls',
        'Poisoned': 'Poisoning',
        'Poisoned ': 'Poisoning',
        'Scalded': 'Scalds',
        'Rta': 'Collision-Related',
        'Struck By, Against': 'Collision-Related',
         'Crushing': 'Collision-Related',
```

```
        'Other ': 'Other',
        'other': 'Other',
        'other ': 'Other'
    })
)
subset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2682 entries, 291420 to 310256
Data columns (total 22 columns):
 #   Column              Non-Null Count  Dtype   
---  ------              --------------  -----   
 0   id_x                2682 non-null   int64   
 1   FinancialYear       2682 non-null   category
 2   HBR                 2682 non-null   object  
 3   HBRQF               0 non-null      object  
 4   CA                  2682 non-null   object  
 5   CAQF                0 non-null      object  
 6   AgeGroup            2682 non-null   category
 7   AgeGroupQF          0 non-null      object  
 8   Sex                 2682 non-null   object  
 9   SexQF               900 non-null    object  
 10  InjuryLocation      2682 non-null   object  
 11  InjuryLocationQF    540 non-null    object  
 12  InjuryType          2682 non-null   object  
 13  InjuryTypeQF        298 non-null    object  
 14  NumberOfAdmissions  2682 non-null   int64   
 15  id_y                2682 non-null   float64 
 16  HB                  2682 non-null   object  
 17  HBName              2682 non-null   category
 18  HBDateEnacted       2682 non-null   float64 
 19  HBDateArchived      0 non-null      float64 
 20  Country             2682 non-null   object  
 21  CountryName         2682 non-null   object  
dtypes: category(3), float64(3), int64(2), object(14)
memory usage: 428.3+ KB
```

```
/tmp/ipykernel_792/2761504185.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  subset['InjuryType'] = (
```

In [10]:
```
subset['InjuryType'] = subset['InjuryType'].astype('category')
subset['InjuryType'].cat.categories
```

```
/tmp/ipykernel_792/299904502.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  subset['InjuryType'] = subset['InjuryType'].astype('category')
```

Out[10]:
```
Index(['Accidental Exposure', 'All Diagnoses', 'Collision-Related', 'Fal
ls',
       'Other', 'Poisoning', 'Scalds'],
      dtype='object')
```

In [11]:
```
# Group by board and injury type
grouped = subset.groupby(['HBName', 'InjuryType'], observed=True)['Number
grouped
```

Out[11]:

	HBName	InjuryType	NumberOfAdmissions
0	NHS Greater Glasgow and Clyde	Accidental Exposure	936
1	NHS Greater Glasgow and Clyde	All Diagnoses	20800
2	NHS Greater Glasgow and Clyde	Collision-Related	472
3	NHS Greater Glasgow and Clyde	Falls	18032
4	NHS Greater Glasgow and Clyde	Other	900
5	NHS Greater Glasgow and Clyde	Poisoning	480
6	NHS Greater Glasgow and Clyde	Scalds	66
7	NHS Lothian	Accidental Exposure	380
8	NHS Lothian	All Diagnoses	16776
9	NHS Lothian	Collision-Related	488
10	NHS Lothian	Falls	14896
11	NHS Lothian	Other	744
12	NHS Lothian	Poisoning	300
13	NHS Lothian	Scalds	36

In [12]:
```
# Get top per board
top5 = (
    grouped.sort_values(['HBName', 'NumberOfAdmissions'], ascending=[True
    .groupby('HBName', observed=True)
    .head(5)
)

top5
```

Out[12]:

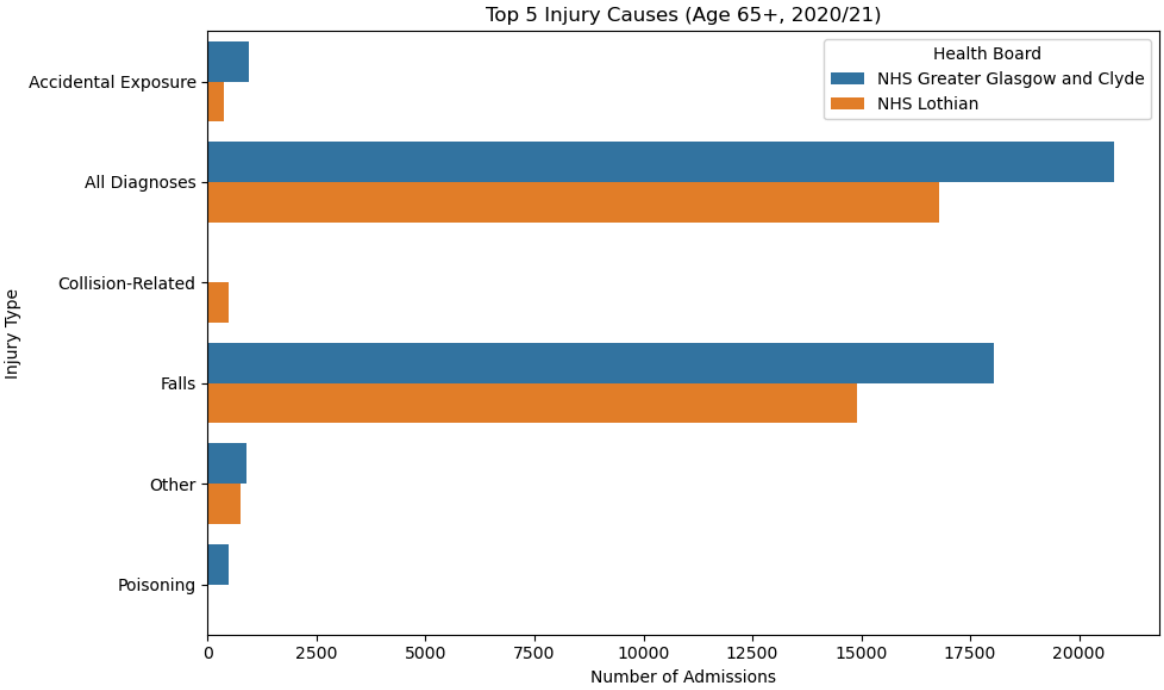
	HBName	InjuryType	NumberOfAdmissions
1	NHS Greater Glasgow and Clyde	All Diagnoses	20800
3	NHS Greater Glasgow and Clyde	Falls	18032
0	NHS Greater Glasgow and Clyde	Accidental Exposure	936
4	NHS Greater Glasgow and Clyde	Other	900
5	NHS Greater Glasgow and Clyde	Poisoning	480
8	NHS Lothian	All Diagnoses	16776
10	NHS Lothian	Falls	14896
11	NHS Lothian	Other	744
9	NHS Lothian	Collision-Related	488
7	NHS Lothian	Accidental Exposure	380

In [13]:
```
# Drop unused categories to remove ghost labels from categorical columns
top5['HBName'] = top5['HBName'].cat.remove_unused_categories()
top5['InjuryType'] = top5['InjuryType'].cat.remove_unused_categories()
top5
```

Out[13]:

	HBName	InjuryType	NumberOfAdmissions
1	NHS Greater Glasgow and Clyde	All Diagnoses	20800
3	NHS Greater Glasgow and Clyde	Falls	18032
0	NHS Greater Glasgow and Clyde	Accidental Exposure	936
4	NHS Greater Glasgow and Clyde	Other	900
5	NHS Greater Glasgow and Clyde	Poisoning	480
8	NHS Lothian	All Diagnoses	16776
10	NHS Lothian	Falls	14896
11	NHS Lothian	Other	744
9	NHS Lothian	Collision-Related	488
7	NHS Lothian	Accidental Exposure	380

In [14]:
```
# Plot only the top injury causes per board
plt.figure(figsize=(10, 6))
sns.barplot(
    data=top5,
    x='NumberOfAdmissions',
    y='InjuryType',
    hue='HBName'
)
plt.title('Top 5 Injury Causes (Age 65+, 2020/21)')
plt.xlabel('Number of Admissions')
plt.ylabel('Injury Type')
plt.legend(title='Health Board')
plt.tight_layout()
plt.show()
```



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

The four most common causes of emergency hospital admissions in 2020/21 among people aged 65 and over were similar in NHS Lothian and NHS Greater Glasgow and Clyde, but not identical. Both health boards shared three of the top four causes: Falls, Accidental Exposure, and Other categories, with variations in ranking and one differing category: Collision-Related for NHS Lothian and Poisoning for NHS Greater Glasgow and Clyde.

Task 1 – Python Approach

In this task, the aim is to investigate the top four most common causes of unintentional injury admissions in 2020/21 among individuals aged 65 and older within two specific Scottish health boards: NHS Lothian and NHS Greater Glasgow and Clyde. The analysis was performed in Python, using the pandas library to manage data and seaborn for visualization (Garnett et al., 2022)

The process was started by loading two datasets: one containing unintentional injury admission records and another with health board codes. The datasets were merged using a left join on the health board reference columns (`HBR` and `HB`) to obtain full health board names in the main injury dataset. After ensuring the merge was successful, data types of all relevant columns were checked and several of them were converted to categorical data types, including `AgeGroup` , `FinancialYear` , `HBName` , and `InjuryType` , to improve filtering and grouping operations.

To isolate the target population, `merged_data` were filtered for records from the 2020/21 financial year, individuals in the `65–74 years` and `75plus years` age groups, and admissions from NHS Lothian and NHS Greater Glasgow and Clyde. This gives a relevant subset of the data to work with.

One of the challenges encountered was inconsistency in the `InjuryType` column. Some values had trailing spaces, inconsistent casing, or were semantically similar but worded differently (e.g., "Falls", "Falling", "Poisoned", "Poisoning"). To address this, the tibble was cleaned manually by stripping whitespace, converting to title case, and applying a series of replacements using `.replace()` to unify categories. This ensured that similar values were grouped together accurately in subsequent aggregation steps (Garnett et al., 2022).

With the cleaned dataset, the data was grouped by `HBName` and `InjuryType`, summing the `NumberOfAdmissions` for each combination. This allowed to rank injury types by total admissions for each health board. Results were sorted and the top four injury types per board were extracted by slicing. The final comparison was presented in tabular format a grouped bar plot was created using Seaborn to visualise differences and overlaps between the two regions.

The Python implementation gave precise control over data transformation and allowed for applying string operations and conditional logic fluently. The pandas library handled filtering, grouping, and reshaping very efficiently. Compared to R, Python requires more explicit management of data types and cleaning steps, but provides more flexibility in handling text-based inconsistencies.

Although the general approach requires checking on the data structures frequently, as Python's attitude towards restoring everything to objects interferes with some data manipulation processes.

Task 1 – Alternative Data Types or Structures

An alternative data structure could have been a dictionary of dictionaries, allowing for multiple layers of description while maintaining the table-like structure of the datasets. While this would allow fast lookups and reduce redundancy, it might complicate filtering, sorting, and grouping by multiple conditions, especially across categorical dimensions like year and age group.

Using NumPy structured arrays could also be another option. NumPy structured arrays offer efficient memory usage and fast computations. However, they lack the high-level functionality of pandas dataframes for handling mixed data types, categorical filtering and a user friendly presentation. For instance, grouping by multiple string-based columns or dealing with inconsistent categorical labels would be more complex using NumPy arrays (Rossi, 2025).

The tabular nature of this task, combined with the need for filtering and summarising, made the pandas DataFrame the most appropriate and practical structure. Its built-in operations aligned well with the demands of the analysis.

Task 2: Python

There has been an IT failure in the prescribing databases in a local area. The only remaining data around medications is the backup file below, but it has been scrambled.

Explore the following data object `medications`. We know from notes that the data should include information on pill shape, color, where it was prescribed, when it was prescribed, the BNF (British National Formulary) code, dose frequency, dose, unit of measure of dose, maximum safe temperature for storage in Celsius, and if it needs to be refrigerated.

Select **7 of the 10** elements of data and reconstruct the dataset about medications. Present `medications` in a clearer and better structured format, ensuring that the data structure and data types are appropriate, given the (limited) information provided.

The code to load the data object `medications` has been provided below.

In [15]:
```
import json

# this function just loads the data from files, there is no need to under
def load_json_file_named(file_name):
    try:
        loaded_data = []
        file_location = f"../data/{file_name}"
        with open(file_location, 'r') as file: # or f"data/{file_name}" d
            loaded_data =  json.load(file)
    except OSError as e:
        print(f"Error. Does the file exist in this folder? {file_location
    return loaded_data

medications = load_json_file_named('task2_data.json')
medications
```

Out[15]:
```
[['06010102',
  '05012023',
  'Hospital',
  '4',
  'ml',
  'cloudy',
  'NA',
  '6',
  '1',
  '2x'],
 ['10010100',
  '26072024',
  'Pharmacy',
  '200',
  'mg',
  'orange',
  'round',
  '25',
  '0',
  '4x'],
 ['05010103',
  '12112023',
  'Not Known',
  '500',
  'mg',
  'pink',
  'oval',
  '25',
  '0',
  '1x'],
 ['04080100',
  '31042022',
  'Other',
  '300',
  'mg',
  'yellow',
  'capsule',
  '24',
  '0',
  '2x'],
 ['06010102', '23012024', 'Other', '3', 'ml', 'cloudy', 'NA', '5', '1',
'3x'],
 ['10010100',
  '14082024',
  'Dentist',
  '400',
  'brown',
  'circle',
  '25',
  '0',
  '2x'],
 ['04080100',
  '19022023',
  'Hospital',
  '100',
  'mg',
  'white',
  'capsule',
  '24',
  '0',
```

```
  '3x'],
 ['05010103', '1703024', 'Hosp', '250', 'mg', 'yellow', 'oval', '0', '2
x'],
 ['10010100', '17062023', 'Dent', '200', 'orange', 'circle', '24', '0',
'4x'],
 ['05010103',
  '01082024',
  'Not Known',
  '250',
  'mg',
  'yellow',
  'oval',
  '25',
  '0',
  '2x'],
 ['04080100',
  '05072022',
  'Pharm',
  '100',
  'mg',
  'white',
  'capsule',
  '24',
  '0',
  '2x'],
 ['06010102', '04092023', 'Other', '4', 'ml', 'cloudy', 'NA', '5', '1',
'3x'],
 ['06010102', '01052025', 'Hospital', '4', 'ml', 'NA', '6', '1', '5x'],
 ['06010102',
  '01072023',
  'Hospital',
  '4',
  'ml',
  'cloudy',
  'NA',
  '6',
  '1',
  '2x'],
 ['10010100', '24112023', 'Pharm', '400', 'mg', 'orange', 'circle', '0',
'2x'],
 ['05010103', '11102023', 'Dent', '250', 'white', 'oval', '24', '0', '2
x'],
 ['04080100',
  '09042024',
  'Hopsital',
  '100',
  'mg',
  'white',
  'capsule',
  '24',
  '0',
  '5x'],
 ['10010100', '10102023', 'Dent', '400', 'orange', 'round', '25', '0',
'2x'],
 ['05010103', '16032024', 'Other', '500', 'yellow', 'oval', '25', '0',
'1x'],
 ['04080100',
  '07092023',
  'Pharm',
  '300',
```

```
  'mg',
  'white',
  'capsule',
  '24',
  '0',
  '2x']]
```

In [16]:
```
# Rows with 10 fields (clean)
medications_clean = [row for row in medications if len(row) == 10]

# Rows with fewer fields (scrambled)
medications_scrambled = [row for row in medications if len(row) != 10]
medications_scrambled
```

Out[16]:
```
[['10010100',
  '14082024',
  'Dentist',
  '400',
  'brown',
  'circle',
  '25',
  '0',
  '2x'],
 ['05010103', '1703024', 'Hosp', '250', 'mg', 'yellow', 'oval', '0', '2
x'],
 ['10010100', '17062023', 'Dent', '200', 'orange', 'circle', '24', '0',
'4x'],
 ['06010102', '01052025', 'Hospital', '4', 'ml', 'NA', '6', '1', '5x'],
 ['10010100', '24112023', 'Pharm', '400', 'mg', 'orange', 'circle', '0',
'2x'],
 ['05010103', '11102023', 'Dent', '250', 'white', 'oval', '24', '0', '2
x'],
 ['10010100', '10102023', 'Dent', '400', 'orange', 'round', '25', '0',
'2x'],
 ['05010103', '16032024', 'Other', '500', 'yellow', 'oval', '25', '0',
'1x']]
```

In [17]:
```
# Known missing positions for each bad row after visualisation
insertion_positions = [5, 8, 5, 6, 8, 5, 5, 5]  # 1-based indexes
values_to_insert = [None] * len(insertion_positions)

# Pad rows by inserting None at specified positions
corrected_rows = []
for row, pos in zip(medications_scrambled, insertion_positions):
    padded = row[:pos-1] + [None] + row[pos-1:]
    corrected_rows.append(padded)
```

In [18]:
```
# Column names
col_names = ["BNF_Code", "PrescriptionDate", "Source", "Dose", "Unit",
             "Colour", "Shape", "Age", "FridgeFlag", "Frequency"]

# Combine all rows
all_rows = medications_clean + corrected_rows

# Create DataFrame
df = pd.DataFrame(all_rows, columns=col_names)
```

In [19]:
```
# Clean and transform data
df_cleaned = (
```

```
    df.copy()
    .assign(
        PrescriptionDate=pd.to_datetime(df['PrescriptionDate'], format='%
        Dose=pd.to_numeric(df['Dose'], errors='coerce'),
        Frequency=pd.to_numeric(df['Frequency'].str.replace('x', '', rege
        Source=df['Source'].str.lower().str.strip().replace({
            'hosp': 'hospital',
            'hopsital': 'hospital',
            'hopitalal': 'hospital',
            'pharm': 'pharmacy',
            'dent': 'dentist',
            'dentistist': 'dentist'
        }),
        FridgeFlag=df['FridgeFlag'].astype(str).str.strip().replace({
            '1': True, '0': False, 'TRUE': True, 'FALSE': False
        })
    )
)
df_cleaned
```

```
/tmp/ipykernel_792/3014371174.py:16: FutureWarning: Downcasting behavior i
n `replace` is deprecated and will be removed in a future version. To reta
in the old behavior, explicitly call `result.infer_objects(copy=False)`. T
o opt-in to the future behavior, set `pd.set_option('future.no_silent_down
casting', True)`
  FridgeFlag=df['FridgeFlag'].astype(str).str.strip().replace({
```

Out[19]:

	BNF_Code	PrescriptionDate	Source	Dose	Unit	Colour	Shape	Age	Fri
0	06010102	2023-01-05	hospital	4	ml	cloudy	NA	6	
1	10010100	2024-07-26	pharmacy	200	mg	orange	round	25	
2	05010103	2023-11-12	not known	500	mg	pink	oval	25	
3	04080100	NaT	other	300	mg	yellow	capsule	24	
4	06010102	2024-01-23	other	3	ml	cloudy	NA	5	
5	04080100	2023-02-19	hospital	100	mg	white	capsule	24	
6	05010103	2024-08-01	not known	250	mg	yellow	oval	25	
7	04080100	2022-07-05	pharmacy	100	mg	white	capsule	24	
8	06010102	2023-09-04	other	4	ml	cloudy	NA	5	
9	06010102	2023-07-01	hospital	4	ml	cloudy	NA	6	
10	04080100	2024-04-09	hospital	100	mg	white	capsule	24	
11	04080100	2023-09-07	pharmacy	300	mg	white	capsule	24	
12	10010100	2024-08-14	dentist	400	None	brown	circle	25	
13	05010103	NaT	hospital	250	mg	yellow	oval	None	
14	10010100	2023-06-17	dentist	200	None	orange	circle	24	
15	06010102	2025-05-01	hospital	4	ml	None	NA	6	
16	10010100	2023-11-24	pharmacy	400	mg	orange	circle	None	
17	05010103	2023-10-11	dentist	250	None	white	oval	24	
18	10010100	2023-10-10	dentist	400	None	orange	round	25	
19	05010103	2024-03-16	other	500	None	yellow	oval	25	

In [20]:
```
# Drop rows missing key fields
df_final = df_cleaned.drop(columns=['Age', 'Shape', 'Colour'])

# Sort by date
df_final = df_final.sort_values(by='PrescriptionDate')
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 20 entries, 7 to 13
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype         
---  ------            --------------  -----         
 0   BNF_Code          20 non-null     object        
 1   PrescriptionDate  18 non-null     datetime64[ns]
 2   Source            20 non-null     object        
 3   Dose              20 non-null     int64         
 4   Unit              15 non-null     object        
 5   FridgeFlag        20 non-null     bool          
 6   Frequency         20 non-null     int64         
dtypes: bool(1), datetime64[ns](1), int64(2), object(3)
memory usage: 1.1+ KB
```

In [21]:
```
df_final[['Source', 'Unit']] = df_final[['Source', 'Unit']].astype({'Sour
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 20 entries, 7 to 13
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype         
---  ------            --------------  -----         
 0   BNF_Code          20 non-null     object        
 1   PrescriptionDate  18 non-null     datetime64[ns]
 2   Source            20 non-null     category      
 3   Dose              20 non-null     int64         
 4   Unit              15 non-null     category      
 5   FridgeFlag        20 non-null     bool          
 6   Frequency         20 non-null     int64         
dtypes: bool(1), category(2), datetime64[ns](1), int64(2), object(1)
memory usage: 1.2+ KB
```

Task 2 – Python Approach

Task 2 involved processing a semi-structured JSON file containing prescription records, where each record was an unnamed list of lists. The records had inconsistent lengths, some complete with all ten expected fields, others missing one or more values at different indexes, causing misalignment when converted directly to a dataframe. The goal was to transform this data into a clean and usable pandas DataFrame with the correct column mapping and aligned fields (Fröhlich et al., 2025).

To begin, the file was loaded using a custom function and the `json` module. The structure of the list was then examined: each element was a prescription entry represented as a list rather than a dictionary. to facilitate visuale examination, the dataset was split into two datasets with exactly ten values (`medications_clean`) and those with fewer (`medications_scrambled`). Following a similar strategy used in the R report, scrambled rows were then corrected manually by identifying the missing positions and inserting `None` values to preserve alignment. These corrected rows were then combined with the clean ones to produce a uniform table.

Column names were then assigned manually, ensuring that each row aligned correctly with fields such as `BNF_Code` , `PrescriptionDate` , `Source` , `Dose` , `Unit` , and others. The resulting DataFrame had full structure but required cleaning and transformation to make the data analysis ready (Fröhlich et al., 2025).

To clean the data, `pandas` string and numeric functions were used. The `Source` column was standardised by stripping white space, converting text to lowercase, and applying replacements to correct typos and abbreviations (e.g., "hopsital" → "hospital", "dent" → "dentist") and to facilitate casting into category data type. `Dose` and `Frequency` were also converted to numeric values and removed any "x" suffix from frequency strings to match the suitable data type for representation. The `FridgeFlag` column was converted to Boolean format. `pd.to_datetime` was used for casting `PrescribtionDate` to date with a custom format to ensure the dates were interpreted correctly. Finally, the `df_final` was manipulated to assign

the right data types for columns to finalise a proper representation for data. This final check was done to facilitate direct future data wrangling.

Although this approach of splitting datasets, visually locating missing data, and manually fixing then remerging it is not the most efficient solution, it offers control over issues that might arise during the data wrangling, ensures complete results, and can be applied in different scenarios when datasets to be corrected ar as small. However, compared to R, where I relied on `tibble` and loops to manually rebuild the table, Python's list comprehensions and pandas chaining offered a more compact, programmatic approach. This made the process more scalable while still allowing for human-guided correction where necessary (Guo et al., 2023).

Task 2 – Alternative Data Types or Structures

One alternative data structure considered for this task was storing each prescription as a dictionary rather than a list. If the JSON data had used key–value pairs, it would have been easier to convert directly into a pandas DataFrame without worrying about misalignment. Each field would map to a column, and missing values could be handled much more easily during conversion. Another option was to work with a list of namedtuples from the `collections` module. This would offer the benefits of clear field names and fixed positions while preserving memory efficiency. However, it would require converting the raw list into tuples with default values for missing fields, which still involves manual alignment.

Given the irregular nature of the JSON file, the most practical and transparent structure was a list of padded lists, converted into a DataFrame after validating and aligning the rows. This allowed full control over column positions and reliable downstream cleaning using pandas.

Reflective Account (Gibbs' Reflective Cycle)

Description
Throughout the course, I have gained multiple skills, mostly in programming, but also in time management, academic writing for data science, and realistic applications of data science for health and care. However, one particular skill I am lacking is efficient navigation across the plethora of open-source learning and research resources.

Feelings
I feel overwhelmed by the number of packages and different approaches to solving problems. Also, it seems that domain-specific data science skill sets that are necessary to develop are highly versatile, and there are multiple ways to achieve the same results, which adds another burden of anxiety when learning a new concept while postponing others to customise my portfolio to my aspirations.

Evaluation
A strength of mine is that I am dedicated to learning data science, and I am getting my hands dirty with material outside of the course's material, which widens my scope

for solutions. However, this can also be an issue that I need to tackle because I am losing my way when considering multiple learning resources when I could have gone straight to the spot.

Analysis

The reason why I am searching for multiple sources of knowledge is that I am looking for a way to describe what I learn in a language close to what I am used to. Also, because the course sometimes jumps between concepts, I find myself needing to define a lot of the concepts.

Conclusion

I need to focus more on one source of material to craft a foundation of knowledge that I might build on for the future and report back for the directors of the course to custom my learning experience for most optimal results.

Action Plan

for the second year of the course, I will keep documentation of skills I need to develop and areas for further study. I have constructed a format for such documentation using Notion.

References

Fröhlich, N., Meier, A., Pardal, N., & Virtema, J. (2025). A logic-based framework for database repairs (arXiv:2306.15516). arXiv. https://doi.org/10.48550/arXiv.2306.15516 Garnett, M. F., Weeks, J. D., & Spencer, M. R. (2022). Unintentional Fall Deaths Among Adults Aged 65 and Over: United States, 2020. NCHS Data Brief, 449, 1–8. Guo, M., Wang, Y., Yang, Q., Li, R., Zhao, Y., Li, C., Zhu, M., Cui, Y., Jiang, X., Sheng, S., Li, Q., & Gao, R. (2023). Normal Workflow and Key Strategies for Data Cleaning Toward Real-World Data: Viewpoint. Interactive Journal of Medical Research, 12(1), e44310. https://doi.org/10.2196/44310 Rossi, F. (2025). Data model and software prototype for antenna geometrical information. https://thesis.unipd.it/handle/20.500.12608/19376?1/FracescaRossi_FinalThesis.pdf