Janssen OHDA Assignment
Roy Williams, MPH
As of 2023-07-17

Contents

Problem

The Observational Medical Outcomes Partnership (OMOP) Common Data Model has been designed to support disparate observational databases (including administrative claims and electronic heath records). Details about OMOP common data model are available at https://ohdsi.github.io/CommonDataModel/ . For this exercise you will limit your work to four tables of interest:

- PERSON
- OBSERVATION_PERIOD
- DRUG_EXPOSURE
- CONDITION_OCCURRENCE

The full table descriptions and structure are also available from the documentation link provided.

Write a program (in SAS, SQL, R, or Python) that could be applied to the OMOP common data model to calculate the Charlson index for all people in an exposed population, indexed on the date of first exposure. The program would take as input a list of drug identifiers (e.g. NDCs or RxNorm CUIs) that define the exposure. Any drug exemplar can be used identify the initial cohort. Although this would not be a rigorous approach to identifying an exposure cohort it is sufficient for this exercise. The program should output a table that contains two fields: PERSON_ID, and the Charlson index score.

Solution

In this solution, we've created a Tidy R-based program to calculate the Charlson Comorbidity Index **(CCI)** for individuals exposed to specific drugs in a dataset adhering to the Observational Medical Outcomes Partnership **(OMOP)** Common Data Model. The program takes as input a list of drug identifiers, identifies an exposed population, and computes the CCI for each person in this population.

As a dataset was not provided, this solution starts by connecting to the Eunomia dataset. Eunomia is a standard synthetic dataset in the OMOP Common Data Model that's used for testing and demonstration purposes **(Schuemie & DeFalco, 2020)**. We then load the required tables (PERSON, OBSERVATION_PERIOD, DRUG_EXPOSURE, CONDITION_OCCURRENCE) and auxiliary datasets (the RxNorm CUI list from Athena that is used for drug identification and the SNOMED CT codes list used to identify conditions) into our R environment.

To identify drug exposures, we use RxNorm CUIs obtained from Athena **(OHDSI, 2023)**. RxNorm provides a normalized naming system for generic and branded drugs and can connect drug vocabularies commonly used in pharmacy management and drug interaction software, enabling interoperability between different systems that have adopted this standard.

Conditions contributing to the CCI are identified using SNOMED CT codes from a validated list **(Fortin, Reps, & Ryan, 2022)**. SNOMED CT is an international healthcare terminology that provides a common language enabling a consistent way of capturing, sharing, and aggregating health data across specialties and sites of care. Its use over ICD codes in this context has multiple advantages including better granularity and precision, comprehensive coverage of clinical concepts, and consistent representation of information across different geographical locations.

Once the drug exposures and conditions are identified, the solution calculates the CCI using the weightage defined by **Quan et al., 2005**. The CCI is calculated as the sum of weights associated with each condition a person has, plus a weight based on age. These weights have been validated and widely adopted in the computation of the CCI. This index is then outputted in a table with PERSON_ID and the CCI score.

The solution was demonstrated using Acetaminophen as the drug of interest, with the resulting CCI calculated for the first ten individuals exposed to Acetaminophen. We also compared the CCI scores of individuals exposed to Acetaminophen and Warfarin using the Mann-Whitney U test as a sanity check for our function.

While SNOMED codes offer more granularity, it's important to acknowledge that SNOMED CT codes are not as extensively validated in the context of CCI as ICD codes. Most of the existing literature and validation work on CCI has been done using ICD-9 or ICD-10 codes, making it more challenging to validate a CCI function using SNOMED CT codes. As such, results should be interpreted with appropriate caution. Comparisons with ICD-based approaches were considered, however not undertaken, due to the limited scope of this assignment and potential discrepancies in results obtained from ICD versus SNOMED codes **(Viernes et al., 2020)**.

This solution offers an adaptable and scalable approach to compute the CCI across different drugs and conditions, which can be crucial in clinical decision-making and healthcare research. As future steps, we propose external validation of the results, unit testing of the function, and increased error handling to further enhance the robustness and reliability of this solution.

As a bonus, we also developed a Shiny application as part of this solution. This application allows users to input a drug name, and it will automatically look up the associated drug_concept_ids, calculate the CCI for individuals exposed to the specified drug, and output the results. This application is an additional tool that allows others to easily adapt and use our solution without needing to interact directly with the underlying R code, thereby increasing the accessibility and usability of our work.

Import Libraries

First, we begin by importing the necessary libraries.

```
# Collection of Data Science Packages
library(tidyverse)
# Package for Handling Dates and Times
library(lubridate)
# Package for Creating PDF Tables
library(kableExtra)
# Package for Data Cleaning
library(janitor)
# Package for Standardizing Clinical Data Vocabulary
library(Eunomia)
# Package for Connecting to Databases
library(DatabaseConnector)
# Package for Coalescing Duplicate Record IDs
library(coviData)
# Package for Tidying Statistical Test Outputs Into Tables
library(broom)
# Package for Report Aesthetics
library(RColorBrewer)
```

Connect to Eunomia

As a datase for this exercise was not provided, we begin by connecting to *Eunomia*. *Eunomia* is a standard dataset in the OMOP (Observational Medical Outcomes Partnership) Common Data Model (CDM) for testing and demonstration purposes.

```
# Initialize Eunomia Connection Details
connectionDetails <- getEunomiaConnectionDetails()
```

```r
# Establish Eunomia Connection
connection <- connect(connectionDetails)
```

Load Crosswalks

First, we define a helper function, *load_data*, to read CSV files and standardize column names by removing spaces, special characters, and resolving other formatting inconsistencies. If the file loading or cleaning process encounters an error, it displays an error message.

This function is then used to load two datasets: rxnorm_list, which contain RxNorm identifiers for drug exposures retrieved from Athena **(OHDSI, 2023)** and snomed_list, which contains a complete and validated list of SNOMED CT codes for conditions used to calculate the Charlson Comorbidity Index **(CCI)** as published by **(Fortin, Reps, & Ryan, 2022)**.

```r
# Define Helper Function to Load Data
load_data <- function(file_name) {
    # Try to Load the Data and Clean the Names
    data <- tryCatch({
        read_csv(file_name) %>%
            clean_names()
    }, error = function(error_condition) {
        # Output a Message if an Error Occurred
        message("Failed to Load Data: ", error_condition)
        return(NULL)
    })

    return(data)
}

# Load RxNorm CUI List for Drug Exposure
rxnorm_list <- load_data("data/athena_rx_norm_concepts.csv")

# Load SNOMED CT Codes List for Conditions
snomed_list <- load_data("data/concept_codes.csv")
```

Load CDM Tables

Subsequently, the function *load_table* is defined. This function loads specific tables from the CDM and makes their names lowercase. It constructs a SQL query to retrieve all the records from the specified table. If there is an error during the execution of the query, it will catch the error, log a message with details about the error, and return a NULL value. This function is then used to load the PERSON, OBSERVATION_PERIOD, DRUG_EXPOSURE, and CONDITION_OCCURRENCE tables from the CDM.

```
# Define Helper Function to Load CDM Tables And Make Names Lowercase
load_table <- function(connection, table_name) {
  # Construct SQL Query
  sql_query <- paste0("SELECT * FROM ", table_name, ";")

  # Execute the Query and Clean the Table Names
  table <- tryCatch({
    querySql(connection, sql_query) %>% clean_names()
  }, error = function(error_condition) {
    # Output a Message if an Error Occurred
    message("Failed to Load Table: ", error_condition)
    return(NULL)
  })

  return(table)
}

# Obtain Person Table Used for Calculating Age Within CCI
person <- load_table(connection, "PERSON") %>%
  select(person_id, birth_datetime)

# Obtain Observation Period Table
observation_period <- load_table(connection, "OBSERVATION_PERIOD")

# Obtain Drug Exposure Table
drug_exposure <- load_table(connection, "DRUG_EXPOSURE")

# Obtain Condition Occurrence Table
condition_occurrence <- load_table(connection, "CONDITION_OCCURRENCE")
```

Get Drug Concept IDs

Next, we define a function that identifies all drug concepts based on a provided drug name. The function joins the drug exposure table with the RxNorm list and filters rows where the concept name contains the specified drug name. The function returns a list of all unique drug concept IDs.

```
# Define Function That Takes String Input To Output Corresponding Drug Concept IDs
get_drug_concept_ids <- function(drug_name) {
  # Join Drug Exposure Table with RxNorm List
  drug_exposure_with_names <- drug_exposure %>%
    inner_join(rxnorm_list, by = c("drug_concept_id" = "concept_id"))

  # Filter Rows Where Concept Name Contains the Specified Drug Name
  drug_concepts <- drug_exposure_with_names %>%
```

```
    filter(stringr::str_detect(tolower(concept_name), tolower(drug_name)))

  # Get Unique Drug Concept IDs
  drug_concept_codes <- drug_concepts %>%
    select(drug_concept_id) %>%
    distinct_all() %>%
    pull()  # Convert the 'drug_concept_id' Column into a List

  return(drug_concept_codes)  # Return the List of Drug Concept IDs
}
```

Defining Conditions and Their Weights

Next, the conditions and their corresponding weights as per the Charlson Comorbidity Index are defined per **(Quan et al., 2005)** This list sets the weights for various health conditions, which will later be used in calculating the Charlson Comorbidity Index.

```
# Define Conditions And Their Corresponding Weights Via Quan 2005
conditions_weights <- list(
  aids_hiv = 6,
  cerebrovascular_disease = 1,
  chronic_pulmonary_disease = 1,
  congestive_heart_failure = 1,
  dementia = 1,
  diabetes_with_chronic_complications = 2,
  diabetes_without_chronic_complications = 1,
  hemiplegia_or_paraplegia = 2,
  malignancy_except_skin_neoplasms = 2,
  metastatic_solid_tumor = 6,
  mild_liver_disease = 1,
  moderate_or_severe_liver_disease = 3,
  myocardial_infarction = 1,
  peripheral_vascular_disease = 1,
  peptic_ulcer_disease = 1,
  renal_disease = 2,
  rheumatic_disease = 1
)
```

Ensure Condition Columns Exist

In the next block, the helper function *ensure_columns_exist* is defined. This function checks whether each health condition exists as a column in the dataframe and if not, it creates the column and sets all values to 0. This function will be used within the calculate_charlson_index function to ensure all necessary columns exist in the dataframe.

```
# Define Helper Function to Ensure Column Exists and Transform Data into 1 or 0
ensure_columns_exist <- function(df, conditions_weights) {
  # Ensure That Each Column Exists in the Dataframe,
    # If Not, Create Column and Set All Values to 0
  for (col in names(conditions_weights)) {
```

```
    if (!col %in% names(df)) {
      # If Column Does Not Exist in the Dataframe
      df <- df %>%
        mutate(!!col := 0)  # Create Column and Set All Values to 0
    } else {
      # If Column Exists in the Dataframe
      # Apply Function to Each Column in the List That Existed Originally
      df <- df %>%
        # Replace NA Values with 0 and Non-NA Values with 1
        mutate_at(vars(col), ~ ifelse(is.na(.), 0, 1))
    }
  }
  return(df)  # Return the Modified Dataframe
}
```

Calculate Charlson Index Function

Finally, the main function *calculate_charlson_index* is defined in its own block. This function performs the main task of calculating the Charlson Comorbidity Index. It takes a list of drug identifiers as input, identifies the relevant exposures, calculates the age at the time of drug exposure, checks whether each health condition exists as a column in the dataframe, calculates the condition weights, and finally calculates the Charlson Comorbidity Index.

```
# Define Main Function to Calculate Charlson Comorbidity Index
calculate_charlson_index <- function(drug_ids) {
  # Use the Drug Identifiers to Select the Relevant Exposures
  drug_exposure_with_names <- drug_exposure %>%
    inner_join(rxnorm_list,
               by = c("drug_concept_id" = "concept_id")) %>%
    # Filter Rows Based on Drug IDs
    filter(drug_concept_id %in% drug_ids)

  # Obtain First Instance of Drug Exposure to Index
  first_exposure <- drug_exposure_with_names %>%
    group_by(person_id) %>%
    filter(drug_exposure_start_date == min(drug_exposure_start_date)) %>%
    slice(1) %>%
    # Ungroup Dataframe After Group-wise Operations
    ungroup()

  # Join First Exposure DF and Conditions DF
  exposure_and_conditions <- first_exposure %>%
    inner_join(condition_occurrence, by = "person_id") %>%
    inner_join(snomed_list, by = c("condition_concept_id" = "concept_id")) %>%
    pivot_wider(
      names_from = "comorbid_condition",
      values_from = "comorbid_condition",
      values_fill = NA
    ) %>%
    # Names to Lowercase with Underscores Instead of Spaces
    clean_names() %>%
    coalesce_dupes(person_id) %>%
```

```
    inner_join(person, by = "person_id") %>%
    mutate(
      age_at_drug_exposure = round(as.numeric(
        difftime(drug_exposure_start_datetime, birth_datetime, units = "secs")
      ) / seconds_in_year),
      # Calculate Age at Drug Exposure
      age_weight = case_when(
        age_at_drug_exposure < 50 ~ 0,
        age_at_drug_exposure >= 50 &
          age_at_drug_exposure < 60 ~ 1,
        age_at_drug_exposure >= 60 &
          age_at_drug_exposure < 70 ~ 2,
        age_at_drug_exposure >= 70 &
          age_at_drug_exposure < 80 ~ 3,
        age_at_drug_exposure >= 80 ~ 4,
        TRUE ~ NA_real_
      )  # Calculate Age Weight Based on Age at Drug Exposure
    )

  exposure_and_conditions <-
    # Ensure All Required Columns Exist
    ensure_columns_exist(exposure_and_conditions, conditions_weights)

  # Assigning Condition Weights as per Charlson Comorbidity Index
  df_condition_weights <- exposure_and_conditions %>%
    # Initialize 'condition_weight' Column with Values Set to 0
    # As Preparatory Step for Adding Condition-Specific Weights
    mutate(condition_weight = 0)
  for (col in names(conditions_weights)) {
    df_condition_weights <- df_condition_weights %>%
        # Add Weighted Condition Values to 'condition_weight' Column
        mutate(condition_weight = condition_weight + conditions_weights[[col]]
                     * (!!rlang::sym(col)))
  }

  # Calculating Charlson Comorbidity Index Score
  df_charlson_index <- df_condition_weights %>%
    # Calculate Charlson Index as Sum of Age Weight and Condition Weight
    mutate(charlson_index = age_weight + condition_weight)

  # The Final Output Table
  output_table <- df_charlson_index %>%
    select(person_id, charlson_index)  # Select Only Required Columns

  return(output_table)  # Return the Final Output Table
}
```

Define Functions for Display, Summarization, Visualization

Now that we have devised our main function to calculate CCI, let's define some helper functions that will assist with data display, visualization, and summarization. This will allow us to display our results in a comprehensive manner without significantly duplicating code. Normally, helper functions would be provided

in an external file, but for the purposes of the this exercise, they are left here to easily identify.

```
# Function To Create Kable With All Numeric Values Rounded To 2 Decimal Places.
tab <- function (.data, ...) {
    # Apply Rounding Function To All Numeric Columns
    .data[] <- lapply(.data, function(x)
        # If The Value Is Numeric And Less Than 0.01, Replace It With "<0.01"
        if (is.numeric(x))
            ifelse(x < 0.01, "<0.01",
                        # Else, Format The Numeric Value To Have 2 Decimal Places
                        formatC(
                           x, format = "f", digits = 2
                        ))
        else
            x)

    # Create Kable With Specified Formatting
    kable_input <- knitr::kable(
        .data,
        format.args = list(big.mark = ",", scientific = FALSE),
        booktabs = TRUE,
        longtable = TRUE,
        align = "l",
        ...
    ) %>% column_spec(1, width = "10cm")

    # Style Kable
    kable_output <- kableExtra::kable_styling(
        kable_input,
        full_width = FALSE,
        position = "center",
        font_size = 10,
        latex_options = c("striped", "repeat_header", "HOLD_position", "scale_down")
    )
    return(kable_output)
}

# Define Function To Format Title
title <- function(data) {
    result <- data %>%
        # Replace All Underscores In Column Names With Spaces
        rename_all(~ gsub("_", " ", .)) %>%
        # Convert All Column Names To Title Case
        rename_all(str_to_title)

    return(result)
}

# Define Function For Frequency Table
freq<- function(data, colName) {
  result <- data %>%
    # Group Data By Specified Column Name
    group_by({{ colName }}) %>%
    # Count The Number Of Each Unique Value In The Specified Column
```

```
    tally() %>%
    # Remove Rows With NA Values
    drop_na() %>%
    # Calculate The Percentage That Each Unique Value Represents Of The Total
    mutate(percent = str_c("(", round(n / sum(n), 2) * 100, "%)")) %>%
    # Sort The Resulting Data Frame In Descending Order By Count
    arrange(desc(n))

  return(result)
}

# Define Function For Bar Plot
bar <- function(data, x, y, title = "title", xlab = "xlab", ylab = "ylab") {
  ggplot(data, aes(
    # Reorder The X Values In Descending Order By The Y Values
    x = reorder({{ x }}, -{{ y }}),
    # Set The Y Values
    y = {{ y }},
    # Set The Fill Color To Be Based On The X Values
    fill = {{ x }}
  )) +
    # Add A Bar Layer To The Plot With The Specified Aesthetics
    geom_bar(stat = "identity", color = "black", size = 0.25) +
    # Add A Text Layer To The Plot With The Specified Aesthetics
    geom_text(aes(label = n), vjust = -0.3, size = 3.5) +
    # Set The Theme Of The Plot To Be Minimal
    theme_minimal() +
    # Remove The Legend And The Major Grid Lines From The Plot
    theme(legend.position = "none", panel.grid.major = element_blank()) +
    # Center The Plot Title And Add A Bottom Margin To It
    theme(plot.title = element_text(hjust = 0.5, margin = margin(b = 15))) +
    # Set The Fill Colors To Be A Set Of 3 Colors
    scale_fill_brewer(palette = "Set3") +
    # Add A Right Margin To The Y Axis Title And Bold It
    theme(axis.title.y = element_text
                (margin = margin(t = 0, r = 20, b = 0, l = 0), face = "bold")) +
    # Wrap The X Axis Labels To A Maximum Width Of 10 Characters
    scale_x_discrete(labels = function(x) str_wrap(x, width = 10)) +
    # Bold The Plot Title
    theme(plot.title = element_text(face = "bold")) +
    # Bold The Y Axis Title
    theme(axis.title.y = element_text(face = "bold")) +
    # Bold The X Axis Title And Add A Top Margin To It
    theme(axis.title.x = element_text(face = "bold", margin = margin(t = 15))) +
    # Bold The Axis Text
    theme(axis.text = element_text(face = "bold")) +
    # Set The Plot Title, X Axis Label, And Y Axis Label
    ggtitle(title) + xlab(xlab) + ylab(ylab)
}

# Define Descriptive Statistics Summary Function
summarize_statistics <- function(df, column) {
    summary_table <- df %>%
```

```
        # Calculate The Number Of Non-NA Values In The Specified Column
        summarize(
            N = sum(!is.na(!!rlang::sym(column))),
            # Calculate The Mean Of The Specified Column
            mean = round(mean(!!rlang::sym(column), na.rm = TRUE), 2),
            # Calculate The Median Of The Specified Column
            median = round(median(!!rlang::sym(column), na.rm = TRUE), 2),
            # Calculate The Standard Deviation Of The Specified Column
            sd = round(sd(!!rlang::sym(column), na.rm = TRUE), 2)
        )
    return(summary_table)
}
```

Acetaminophen Results

With our main function and helper functions ready, it's time to calculate the Charlson Comorbidity Index (CCI) for the medication Acetaminophen. We'll first fetch the drug concept IDs corresponding to Acetaminophen and then calculate the CCI for these IDs. Next, we'll summarize the CCI data and visualize it to understand the distribution of comorbidity in patients exposed to Acetaminophen.

```
# Get Drug Concept IDs for 'Acetaminophen'
acetaminophen_drug_ids <- get_drug_concept_ids("acetaminophen")

# Calculate Charlson Index for the Obtained Drug Concept IDs
charlson_index_acetaminophen <- calculate_charlson_index(acetaminophen_drug_ids)

# Get The First 10 Rows Of The CCI For Those Exposed To Acetaminophen And Format Them
charlson_index_acetaminophen %>%
    head(10) %>%
    title() %>%
    tab(caption = "CCI for First 10 Individuals with Acetaminophen Exposure")
```

Table 1: CCI for First 10 Individuals with Acetaminophen Exposure

Person Id	Charlson Index
1.00	1.00
2.00	2.00
3.00	1.00
9.00	1.00
11.00	1.00
12.00	1.00
23.00	1.00
32.00	2.00
35.00	1.00
38.00	1.00

```
# Calculate Summary Statistics for Acetaminophen Exposure
summary_table <-
```

```
    summarize_statistics(charlson_index_acetaminophen, "charlson_index") %>%
    title() %>%
    tab(caption = "CCI Summary Statistics for Acetaminophen Exposure")
summary_table
```

Table 2: CCI Summary Statistics for Acetaminophen Exposure

N	Mean	Median	Sd
1227.00	1.19	1.00	0.45

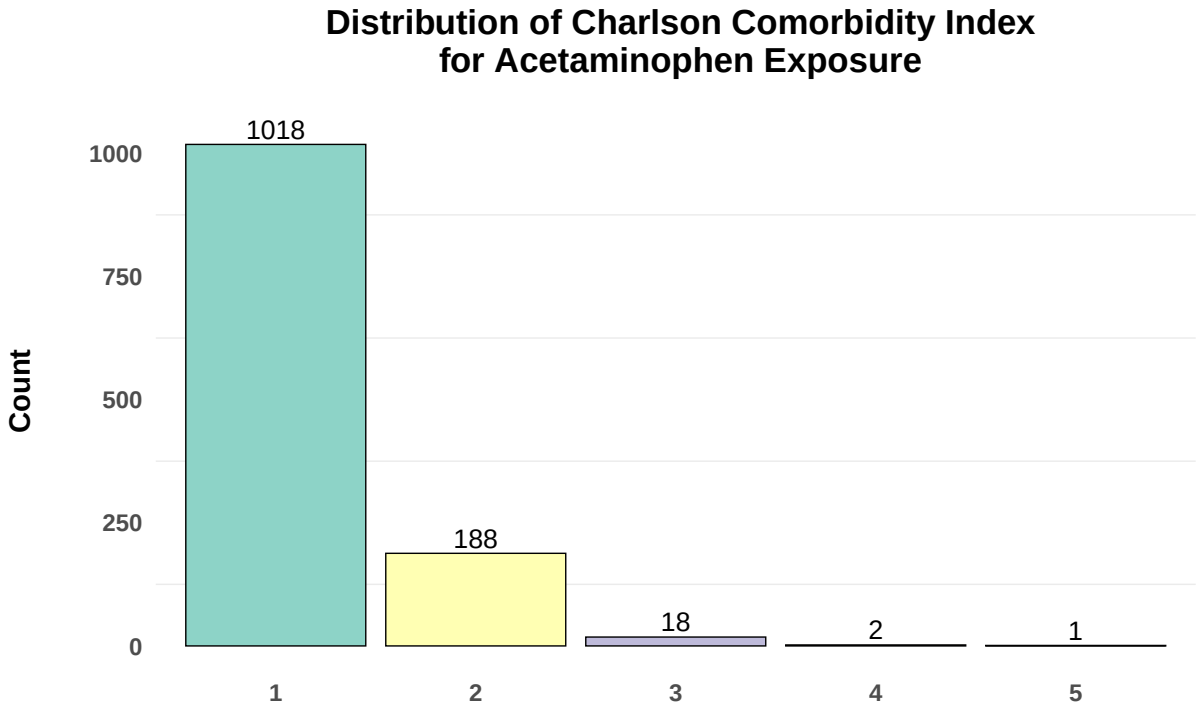
```
# Calculate The Frequency Distribution Of CCI For Those Exposed To Acetaminophen
acetaminophen_charlson_frequency <- charlson_index_acetaminophen %>%
    freq(charlson_index) %>%
    arrange(charlson_index)

# Display The Frequency Distribution In A Formatted Table
acetaminophen_charlson_frequency_table <-
    acetaminophen_charlson_frequency %>%
    adorn_totals() %>%
    title() %>%
    tab(caption = "Distribution of CCI \n for Acetaminophen Exposure")
acetaminophen_charlson_frequency_table
```

Table 3: Distribution of CCI for Acetaminophen Exposure

Charlson Index	N	Percent
1	1018.00	(83%)
2	188.00	(15%)
3	18.00	(1%)
4	2.00	(0%)
5	1.00	(0%)
Total	1227.00	-

```
# Plot The Frequency Distribution For Acetaminophen Exposure
acetaminophen_charlson_frequency_graph <-
    acetaminophen_charlson_frequency %>%
    mutate(charlson_index = as.character(charlson_index)) %>%
    bar(
        charlson_index ,
        n,
        "\n  Distribution of Charlson Comorbidity Index \n for Acetaminophen Exposure",
        "\n Charlson Comorbidity Index (Quan 2005)",
        "\n Count"
    )
acetaminophen_charlson_frequency_graph
```



From this, we can see the majority of individuals with acetaminophen exposure have a CCI of 1 **(1011, 82%)**. The mean for this group is **1.18**. The median is **1.00**. There was a total N = **1227** in Eunomia.

Warfarin Results

Now, as a sanity check, let's repeat the same process for Warfarin. Warfarin is a blood thinner and it is reasonable to suspect that those exposed to this medication would have a higher CCI. We'll fetch the drug concept IDs for 'Warfarin', calculate the CCI for these IDs, summarize the data, and visualize it. This will give us insights into the comorbidity distribution in patients exposed to Warfarin. While this is not external validation of our function, it can help validate our result.

```
# Get Drug Concept IDs For Warfarin
warfarin_drug_ids <- get_drug_concept_ids("warfarin")

# Calculate CCI For Warfarin
charlson_index_warfarin <- calculate_charlson_index(warfarin_drug_ids)

# Calculate Summary Statistics for Warfarin
summary_table_warfarin <- summarize_statistics(charlson_index_warfarin, "charlson_index")

# Display The Summary Statistics In A Formatted Table
summary_table_warfarin %>%
  title() %>%
  tab(caption = "CCI Summary Statistics for Warfarin Exposure")
```

Table 4: CCI Summary Statistics for Warfarin Exposure

N	Mean	Median	Sd
87.00	3.61	4.00	1.39

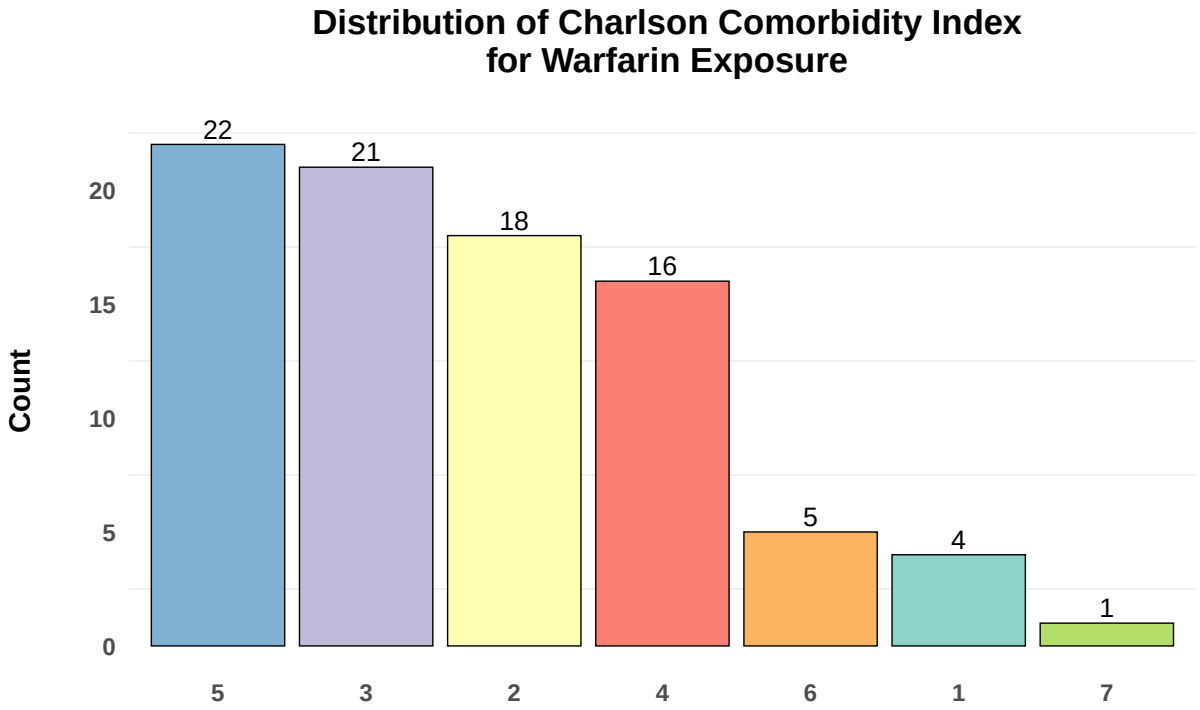
```
# Calculate The Frequency Distribution Of The CCI For Those Exposed To Warfarin
warfarin_charlson_frequency <- charlson_index_warfarin %>%
  freq(charlson_index) %>%
  arrange(charlson_index)

# Display The Frequency Distribution In A Formatted Table
warfarin_charlson_frequency_table <- warfarin_charlson_frequency %>%
  adorn_totals() %>%
  title() %>%
  tab(caption = "Distribution of Charlson Comorbidity Index \n for Warfarin Exposure")
warfarin_charlson_frequency_table
```

Table 5: Distribution of Charlson Comorbidity Index for Warfarin Exposure

Charlson Index	N	Percent
1	4.00	(5%)
2	18.00	(21%)
3	21.00	(24%)
4	16.00	(18%)
5	22.00	(25%)
6	5.00	(6%)
7	1.00	(1%)
Total	87.00	-

```
# Plot The Frequency Distribution
warfarin_charlson_frequency_graph <-
    warfarin_charlson_frequency %>%
    mutate(charlson_index = as.character(charlson_index)) %>%
    bar(
        charlson_index ,
        n,
        "\n  Distribution of Charlson Comorbidity Index \n for Warfarin Exposure",
        "\n Charlson Comorbidity Index (Quan 2005)",
        "\n Count"
    )
warfarin_charlson_frequency_graph
```



The majority of individuals with Warfarin exposure had a CCI of 5 **(22, 25%)**. Their mean CCI was **3.60** with a median of approximately **4.00**. This is more than the median of **1.00** for those with Acetaminophen exposure. In total, there was N = **87** individuals.

Mann-Whitney U Test

Finally, we want to perform some type of inference to determine if there is a difference between these two populations. First, we identify individuals who were exposed to both Acetaminophen and Warfarin by finding the common person_ids in the two dataframes charlson_index_warfarin and charlson_index_acetaminophen. We then exclude these individuals from both dataframes as we want to strictly compare the Charlson Comorbidity Index (CCI) between two separate groups: Those exposed to Acetaminophen and those exposed to Warfarin. If we did not exclude, this would violate the assumption of independence required for the test. In practice, this type of exclusion is not ideal. There may be something particularly interesting about individuals with both exposures. However, for the purpose of this exercise it is sufficient.

After excluding individuals with exposure to both medications, we perform a Mann-Whitney U test (also known as Wilcoxon rank-sum test) to compare the distributions of CCI for the two groups. This non-parametric test is used as it does not assume a normal distribution of the CCI scores. The wilcox.test function is used for this purpose.

```
# Get Unique person_ids Who Were Exposed to Both Acetaminophen and Warfarin
common_person_ids <-
    intersect(charlson_index_warfarin$person_id,
                        charlson_index_acetaminophen$person_id)

# Print Common Individual Length
common_person_ids %>%
```

```
    length() %>%
    tidy() %>%
    tab(caption = "Common Person ID Length",
            col.names = ("Count of Individuals with Both Exposures"))
```

Table 6: Common Person ID Length

Count of Individuals with Both Exposures
86.00

```
# Let's Inspect These IDs and Print Them
print(common_person_ids)
```

```
##  [1]    2    3  263  299  432  454  469  507  621  642  703  742  757  759  863
## [16]  869  935 1219 1223 1227 1318 1359 1476 1506 1538 1626 1634 1661 1762 1848
## [31] 1946 2025 2108 2151 2152 2287 2333 2334 2392 2414 2497 2758 2771 2782 2827
## [46] 2917 2942 3028 3134 3177 3178 3236 3248 3295 3342 3561 3593 3622 3639 3694
## [61] 3714 3791 3823 3859 3917 3968 3973 4017 4090 4365 4376 4433 4521 4596 4633
## [76] 4675 4712 4942 4986 5035 5068 5073 5157 5222 5235 5301
```

```
# Identify Person_id of Patients with Both Exposures
both_exposures <-
    intersect(charlson_index_warfarin$person_id,
                        charlson_index_acetaminophen$person_id)

# Exclude These Patients From Each Dataframe
charlson_index_warfarin <-
    subset(charlson_index_warfarin,!(person_id %in% both_exposures))
charlson_index_acetaminophen <-
    subset(charlson_index_acetaminophen,
                !(person_id %in% both_exposures))

# Perform Mann-Whitney U Test To Examine Statistical Significance
# Between CCI For Those Exposed To Acetaminophen vs Warfarin
test_result <- wilcox.test(
    charlson_index_acetaminophen$charlson_index,
    charlson_index_warfarin$charlson_index
) %>%
    # Make Test Result Kable Formatable
    tidy() %>%
    # Select Columns For Output To Fit In Margins
    select(statistic, p.value) %>%
    tab(caption = "Comparison of CCI for Acetaminophen and Warfarin")

# Print Test Results
test_result
```

Table 7: Comparison of CCI for Acetaminophen and Warfarin

statistic	p.value
99.50	0.03

After **86** individuals with both Acetaminophen and Warfarin exposure are eliminated from the analysis, based on the results of a Mann-Whitney U Test, there is a statistically significant difference in CCI for those with Acetaminophen vs those with Warfarin exposure, $\mathbf{p = 0.03 < 0.05}$. The results for individuals with Warfarin exposure are significantly different than those with Acetaminophen exposure once indiviudals with both exposures are removed.

Conclusion

In this solution, we've created an R-based program to calculate the Charlson Comorbidity Index (CCI) based on the **(Quan 2005)** weights for individuals exposed to specific drugs in a dataset adhering to the Observational Medical Outcomes Partnership (OMOP) Common Data Model. The program takes as input a list of drug identifiers, identifies an exposed population, and computes the CCI for each person in this population.

The solution was demonstrated using the synthetic dataset Eunomia **(Schuemie & DeFalco, 2020)** and the drugs Acetaminophen and Warfarin as the drugs of interest. As expected, the distribution of CCI scores in the population exposed to Warfarin, a medication usually prescribed for conditions such as deep vein thrombosis or atrial fibrillation, was significantly different from the population exposed to Acetaminophen, an over-the-counter pain reliever and fever reducer. The majority of individuals exposed to Warfarin had a CCI of **5**, with a mean CCI of **3.60** and a median of **4.00**. In contrast, the majority of individuals exposed to Acetaminophen had a CCI of **1**, with a mean CCI of **1.18** and a median of **1.00**. The Mann-Whitney U test confirmed a statistically significant difference between the two groups **($\mathbf{p < 0.05}$)**.

However, several limitations need to be noted. SNOMED CT codes are not as extensively validated in the context of CCI as ICD codes. Most of the existing literature and validation work on CCI has been done using ICD-9 or ICD-10 codes, making it more challenging to validate a CCI function using SNOMED CT codes. As such, results should be interpreted with appropriate caution. Furthermore, our results were not externally validated due to the scope of this assignment and potential discrepancies in ICD versus SNOMED codes **(Viernes et al., 2020)**.

Future steps include external validation of the results, unit testing of the function, and increased error handling to further enhance the robustness and reliability of this solution. As a bonus, we also developed a Shiny application as part of this solution. This application allows users to input a drug name, and it will automatically look up the associated drug_concept_ids, calculate the CCI for individuals exposed to the specified drug, and output the results. This application is an additional tool that allows others to easily adapt and use our solution without needing to interact directly with the underlying R code, thereby increasing the accessibility and usability of our work.

The solution offers an adaptable and scalable approach to compute the CCI across different drugs and conditions, which can be crucial in clinical decision-making and healthcare research. By leveraging standard vocabularies like RxNorm and SNOMED CT, it allows for the consistent and interoperable representation of drug exposures and conditions across different databases and geographical locations. Furthermore, the use of OMOP CDM, coupled with the synthetic dataset Eunomia, ensures that the solution can be widely applied to different observational databases, including administrative claims and electronic health records, thereby increasing its potential impact and utility.

References

- Fortin SP, Reps J, Ryan P. Adaptation and validation of a coding algorithm for the Charlson Comorbidity Index in administrative claims data using the SNOMED CT standardized vocabulary. BMC Med Inform Decis Mak. 2022 Oct 7;22(1):261. doi: 10.1186/s12911-022-02006-1. Erratum in: BMC Med Inform Decis Mak. 2023 Jun 15;23(1):109. PMID: 36207711; PMCID: PMC9541054.
- Observational Health Data Sciences and Informatics (OHDSI). (2023). Athena - OHDSI Vocabularies Repository [Database].
- Quan H, Sundararajan V, Halfon P, Fong A, Burnand B, Luthi JC, Saunders LD, Beck CA, Feasby TE, Ghali WA. Coding algorithms for defining comorbidities in ICD-9-CM and ICD-10 administrative data. Med Care. 2005 Nov;43(11):1130-9. doi: 10.1097/01.mlr.0000182534.19832.83. PMID: 16224307.
- Schuemie, M., & DeFalco, F. (2020). Eunomia: Synthetic patient-level data for the Observational Health Data Sciences and Informatics (OHDSI) community. OHDSI.
- Viernes, B., Lynch, K. E., Robison, B., Gatsby, E., DuVall, S. L., & Matheny, M. E. (2020). SNOMED CT Disease Hierarchies and the Charlson Comorbidity Index (CCI): An analysis of OHDSI methods for determining CCI. In Proceedings of the OHDSI Symposium. Retrieved from https://www.ohdsi.org/wp-content/uploads/2020/10/Ben-Viernes-Benjamin-Viernes_CCIBySNOMED_2020Symposium.pdf