summary_stats_anycode_over_year

Enci Cai

2025-04-01

Overview of 'Any Code Over Year" function

Similar to the main summary stats function, this additional function is also powerful in facilitating data aggregation, summarization, and visualization for large datasets. In addition, this function also captures 'Time', which is essential in seeing and comparing trends. This vignette demonstrates its application on the RA (Rheumatoid Arthritis) dataset, which contains information on 6,131 patients. The function is particularly useful for quality control (QC), exploratory data analysis, and generating insights through visualization.

The vignette showcases the following key features:

- Aggregating raw data into a structured format.
- Extract user-selected codes of interest.
- Mapping variables to their corresponding ontology descriptions using a dictionary.
- Visualizing the results through customizable line charts.

The function supports flexible workflows, making it adaptable for datasets spanning medications, laboratory tests, procedures, diagnoses and CUI.

Loading package

```
library(SummaryStats)
library(RSQLite)
```

Connect to the dataset and dictionary

Here we show the process of using RA sqlite file of monthly count that is available on O2 server for the test. In order to assign correct name to each Parent_Code in the dataset, we also load a mapping dictionary that includes corresponding Common Ontology Description and Group Description to the Code. For testing purpose locally, we are using simulated dataset instead.

```
if (FALSE) {
 db_path <- "/n/data1/hsph/biostat/celehs/lab/projects/RAPROD/extdata/ra_monthly.sqlite"
 con <- dbConnect(SQLite(), dbname = db_path)

# Load the dictionary_mapping file
```

```r
  file_path <- "/n/data1/hsph/biostat/celehs/lab/datasets/dictionary_mapping_v3.4.tsv"
  dictionary_mapping <- read_tsv(file_path, show_col_types = FALSE)
}
```

Example dataset & dictionary Creation

To illustrate the functionality of the any_codes_over_year function, we simulate an example dataset that mimics the structure of real-world data. This dataset contains patient observations, coded medical data, and frequency counts, similar to those found in electronic health records (EHRs). By using this simulated dataset, we demonstrate the data aggregation, selection, mapping, and visualization steps without requiring access to the full RA data via O2 server.

```r
# Step 1: Simulating an example dataset & dictionary for mapping

dictionary_mapping <- data.frame(
  Group_Code = c("RXNORM:123", "RXNORM:456"),
  Common_Ontology_Code = c("RXNORM:001", "RXNORM:002"),
  Common_Ontology_Description = c("Acetaminophen", "Ibuprofen"),
  Group_Description = c("Pain Relievers", "NSAIDs")
)

df_ehr <- data.frame(
  Year = sample(1995:2020, 1000, replace = TRUE),
  Patient = sample(1:500, 1000, replace = TRUE),
  Parent_Code = sample(c("RXNORM:123", "RXNORM:456", "RXNORM:789"), 1000, replace = TRUE),
  Count = sample(1:10, 1000, replace = TRUE)
)

# Step 2: Creating an intermediary SQLite database
test_db_path <- tempfile()  # Temporary SQLite file
test_db <- dbConnect(SQLite(), test_db_path)
dbWriteTable(test_db, 'df_monthly', df_ehr, overwrite = TRUE)
```

Workflow Overview

Step 1: Data Aggregation

The first step involves aggregating the raw data into a structured format. The input dataset includes the following columns:

- `Patient` # Unique patient identifier
- `Month` # Time period of the observation
- `Parent_Code` # Code representing the observation (e.g., medication, lab test)
- `Count` # Frequency of the observation

The function `generate_intermediary_sqlite`, when specifying `time_column`, aggregates this data into a new SQLite database containing:

- `Patient`
- `Parent_Code`
- `Year` # Extracted from the first four characters of the time_column
- `Count` # Total count of this observation for this specific Patient

Again, we show how the function works on O2 and implement simulated dataset for local testing purpose. Note `time_column = "Month"` indicates that the time information can be captured from column `Month` in the RA dataset.

Important: The column specified by time_column (e.g., "Month" or "Year") must begin with a valid 4-digit year in every entry (e.g., "2013", "2020-11", "2018-07-22"). The function extracts the first 4 characters only — it is the user's responsibility to ensure this format.

```
# Running RA dataset on O2 server
if (FALSE) {
generate_intermediary_sqlite(con_test, output_sqlite_path = "./test_time.sqlite",
                             time_column = "Month")
}
```

Note `time_column = "Year"` indicates that the time information can be captured from column `Year` in our simulated dataset as we generated the df_ehr.

```
# Run SQL aggregation to generate intermediary file
intermediary_test_db_path <- tempfile()
generate_intermediary_sqlite(test_db, output_sqlite_path = intermediary_test_db_path,
                             time_column = "Year")

# Disconnect from test database
dbDisconnect(test_db)

# Load intermediary file
interm_test_db <- dbConnect(SQLite(), intermediary_test_db_path)
df_summ <- dbGetQuery(interm_test_db, 'SELECT * FROM processed_data;')
dbDisconnect(interm_test_db)
```

This step reduces data redundancy and prepares it for analysis.

Step 2: Data Selection, Mapping and Extraction

The second step depends on the analytical goal:

- `codes_of_interest`: User can define any codes of interest as input

During this step, the `Parent_Code` is mapped to its corresponding `Group_Description` or `Common_Ontology_Description` using a dictionary, ensuring standardized descriptions.

The function `extract_patient_counts_over_years` will use the intermediary data file generated from Step 1, based on number of codes(n) as inputs, prepare n+1 tables that represent patient counts for each code and a additional table (data$combined) that puts all the codes together.

Here in the simulated dataset, we are going to show the most common scenario of usage. As shown in the output, `data_test$RXNORM:123` shows the patient counts of `RXNORM:123` across available years and `data_test$combined` shows the patients counts of all codes listed in `codes_of_interest` across available years.

```
data_test <- extract_patient_counts_over_years(
  sqlite_file = intermediary_test_db_path,
  codes_of_interest = c("RXNORM:123","RXNORM:456","RXNORM:789"),
  dictionary_mapping = dictionary_mapping
)

head(data_test$`RXNORM:123`,3)
```

```
##   Year Parent_Code Patient_Count                 Name
## 1 1995  RXNORM:123             9 Pain Relievers (123)
## 4 1996  RXNORM:123            11 Pain Relievers (123)
## 7 1997  RXNORM:123             8 Pain Relievers (123)
```

```
head(data_test$combined,6)
```

```
##   Year Parent_Code Patient_Count                 Name
## 1 1995  RXNORM:123             9 Pain Relievers (123)
## 2 1995  RXNORM:456            13         NSAIDs (456)
## 3 1995  RXNORM:789            10            789 (789)
## 4 1996  RXNORM:123            11 Pain Relievers (123)
## 5 1996  RXNORM:456            18         NSAIDs (456)
## 6 1996  RXNORM:789             7            789 (789)
```

Here shows what we've been doing with the actual RA dataset on O2 server.

- `sqlite_file = ./RA_intermediary_time.sqlite` indicates the intermediary sqlite file we were using under the working directory on O2 server.
- `codes_of_interest = c("PheCode:714.1", "RXNORM:5487", "RXNORM:6851")` specifies the codes of our interests that we want the function to extract their corresponding patient counts across all available years.
- `dictionary_mapping = dictionary_mapping` indicates the input of dictionary we used to map codes of our interests. Here the name of our dictionary dataframe is 'dictionary_mapping' as well.

```
if (FALSE) {
  output_data <- extract_patient_counts_over_years(
    sqlite_file = "./RA_intermediary_time.sqlite",
    codes_of_interest = c("PheCode:714.1", "RXNORM:5487", "RXNORM:6851"),
    dictionary_mapping = dictionary_mapping
  )
}
```

Step 3:Data Visualization

The final step involves creating line charts to visualize the data. The `plot_patient_counts_over_time` function generates:

Line charts of Patient Counts: Showcases the number of patients associated with each code across selected years. * X-axis: Year * Y-axis: Patient Count

Below are examples showing the outputs from real EHR data from RA. We saved the end product from Step 2 on O2 server and made it usable for step 3 locally.

```
data("summary_stats_data_year")
```

Example

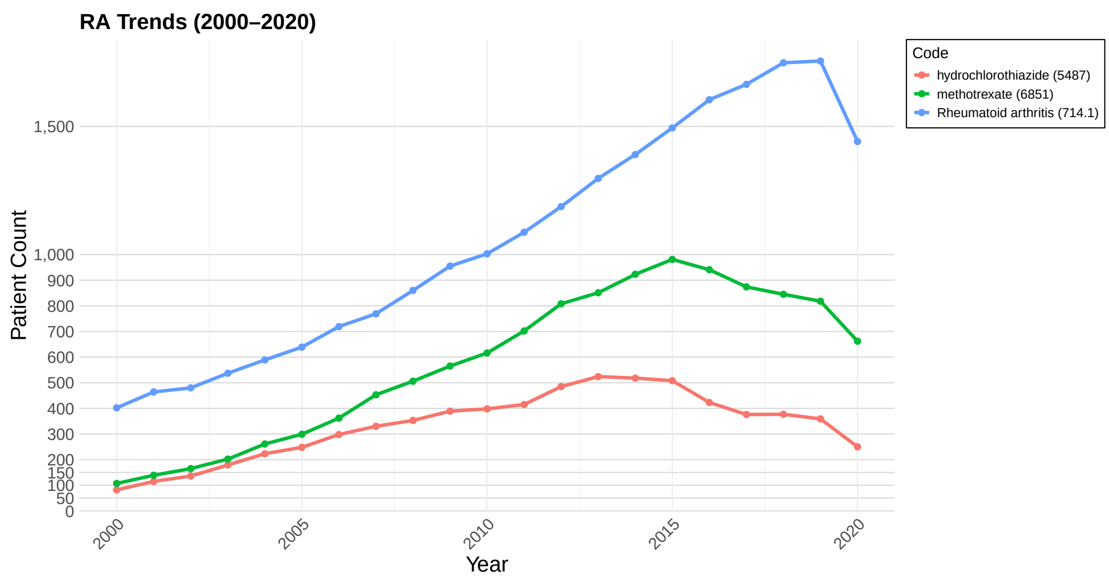
- `data = summary_stats_data_year$combined` indicates the data table that is ready to view. Selecting `combined` will show lines for all codes of interests in one chart while `any specific code` will only show one line.
- `year_range = c(2000, 2020)` indicates that we want to only see the trends within the time interval between year 2000 to year 2020.
- `auto_breaks = FALSE` is default. If set to `TRUE`, the function will automatically generate non-uniform breaks on the y-axis based on the range of patient counts. This is especially useful when codes have a wide range of prevalence — for example, some codes occur in only a few patients while others occur in thousands. Using `auto_breaks = TRUE` improves readability by balancing spacing between low and high patient counts.
- `log_scale = FALSE` is default. If set to `TRUE`, the y-axis is displayed on a log10 scale. This is helpful when the range of patient counts is large and you want to visualize smaller trends more clearly.
- `save_plots = FALSE` is default, indicating that we do not want to save the output plots to the `output_path`. If the plots need to be saved, simply add parameter `save_plots = TRUE`.
- `output_path = tempdir()` is default. If certain directory is specified, for example the current working directory, we can use `"output_path = ./output"`.

Customizing the Y-axis You can choose between two different y-axis display strategies based on the spread of patient counts:

- Option 1: Non-uniform axis breaks (preserving actual values) Use `auto_breaks = TRUE` to apply customized y-axis intervals. This is especially useful when codes vary widely in frequency (e.g., 50 vs. 5,000 patients), and you want to keep the scale linear without cluttering the axis.

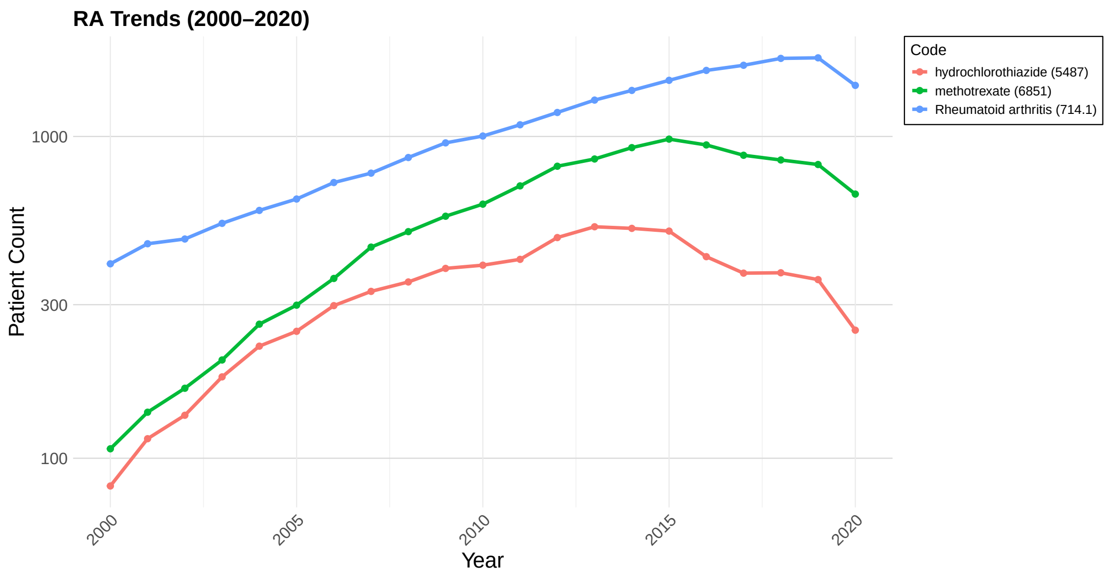
```
plot_patient_counts_over_time(
  data = summary_stats_data_year$combined,
  title = "RA Trends (2000-2020)",
  year_range = c(2000, 2020),
```

```
  auto_breaks = TRUE,
  log_scale = FALSE,
  save_plot = FALSE,
  output_path = tempdir()
)
```



- Option 2: Logarithmic scale Use `log_scale = TRUE` when you're comparing codes with vastly different patient counts, and you want to emphasize relative patterns (e.g., exponential growth) rather than raw numbers. This makes rare codes more visible in the plot.

```
plot_patient_counts_over_time(
  data = summary_stats_data_year$combined,
  title = "RA Trends (2000-2020)",
  year_range = c(2000, 2020),
  auto_breaks = FALSE,
  log_scale = TRUE,
  save_plot = FALSE,
  output_path = tempdir()
)
```


RA Trends (2000–2020)
Code
hydrochlorothiazide (5487)
methotrexate (6851)
Rheumatoid arthritis (714.1)
Patient Count
1000
300
100
2000
2005
2010
2015
2020
Year