Summary_stats_function

Enci Cai

2025-02-06

Overview of the Summary Stats Function

The summary_stats function is a powerful tool designed to facilitate data aggregation, summarization, and visualization for large datasets. 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.
- Selecting top-performing codes or user-defined variables of interest.
- Mapping variables to their corresponding ontology descriptions using a dictionary.
- Visualizing the results through customizable histograms.

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

Importance of the Summary Stats Function

- Streamlined Data Aggregation

The function performs efficient SQL-based aggregation to reduce large datasets into manageable, summarized forms. This enables researchers to focus on high-value data points without being overwhelmed by the raw data.

- Flexible Data Selection

With its ability to handle top N counts, additional variables, or user-defined items, the function provides tailored outputs suited to diverse analytical needs.

- Ontology Mapping and Quality Control

Using dictionary mapping, the function ensures that data is consistently labeled with standardized ontology descriptions, improving interpretability and minimizing errors.

- Data Visualization for Insight Generation

Customizable histograms allow researchers to explore trends, identify outliers, and generate publication-ready visualizations.

Loading package

```r
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.

```r
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

 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 summary_stats 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(
  Month = sample(1:12, 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` aggregates this data into a new SQLite database containing:

- `Patient`
- `Parent_Code`
- `Total_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.

```
# Running RA dataset on O2 server
if (FALSE) {
generate_intermediary_sqlite(con, output_sqlite_path ="./RA_intermediary.sqlite")
}
```

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

# 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:

- `Top N Codes`: Select the top 20 (default) codes by count.
- `Top N + Additional Variables`: Include user-defined variables of interest alongside the top N.
- `Specific list of Variables`: Focus on a predefined list of variables with categories.

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

The function `extract_data_for_visualization` will use the intermediary data file generated from Step 1, based on the request arguments, prepare two tables that represent the total and patient counts for the requested variables.

Here in the simulated dataset, we are going to show the most common scenario of usage.

```
data_null <- extract_data_for_visualization(
  sqlite_file = intermediary_test_db_path,
  prefix = "RXNORM:",
  top_n = 20
)

head(data_null$Patient_Counts, 3)
```

```
##    Parent_Code Total_Count Patient_Count                  Name
##         <char>       <int>         <int>                <char>
## 1:  RXNORM:789        2035           259             789 (789)
## 2:  RXNORM:123        1763           242 Pain Relievers (123)
## 3:  RXNORM:456        1741           239         NSAIDs (456)
```

Step 3:Data Visualization

The final step involves creating histograms to visualize the data. The `plot_visualized_data` function generates:

- Total Counts Plot: Highlights the most frequent observations.
- Patient Counts Plot: Showcases the number of patients associated with each code.

For the variables listed on X-axis, they appear as `Description (Code without prefix)`. Description that is longer than 30 letters will be truncated to `[...]`.

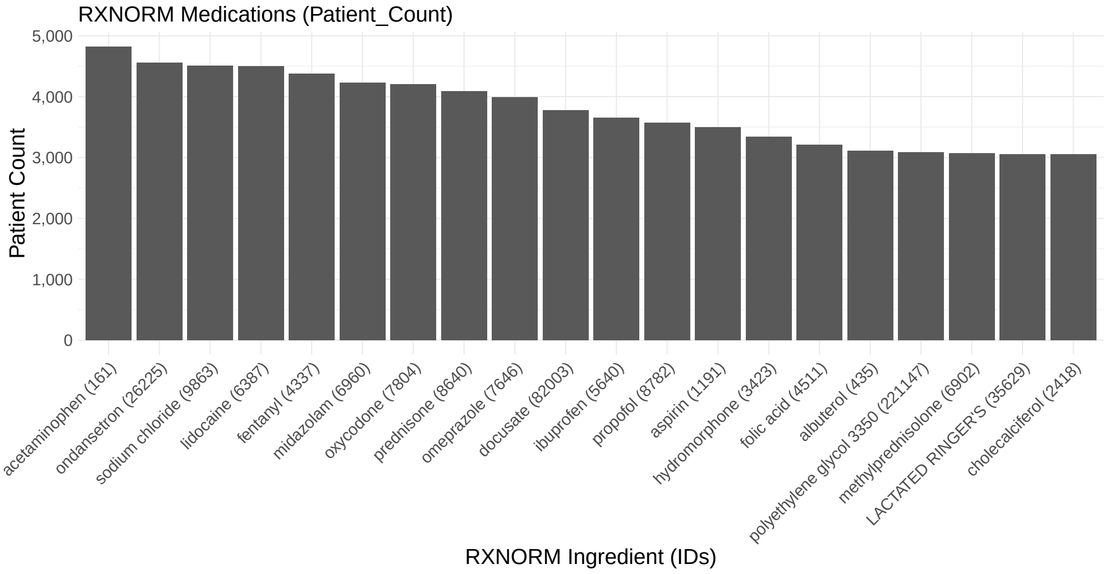
Below are examples showing the outputs from real EHR data from RA. We saved the end product from Step 2 on O2 and made it usable for step 3 locally.

```
data("summary_stats_data")
```

Example 1: This is the most commonly used scenario

- `prefix = "RXNORM:"` indicates the prefix of the Parent_Code of our choice in the dataset is RXNORM:.
- `Top_n = 20` indicates that we want to see the top 20 most frequent observations.
- `dict_prefix = "RXNORM:"` is not shown as the prefix in dataset and dictionary is the same. If necessary, this argument indicates the prefix of the Parent_Code of our choice in the dictionary.
- `description_label = "RXNORM Medications"` indicates the content to be placed in the title.
- `data = data_rxnorm` indicates the data table that is ready to view
- `count_column = "Patient_Counts"` indicates that we only want to see the Patient Counts without Total Counts, vice versa. Without this argument will automatically return both plots.
- `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"`.

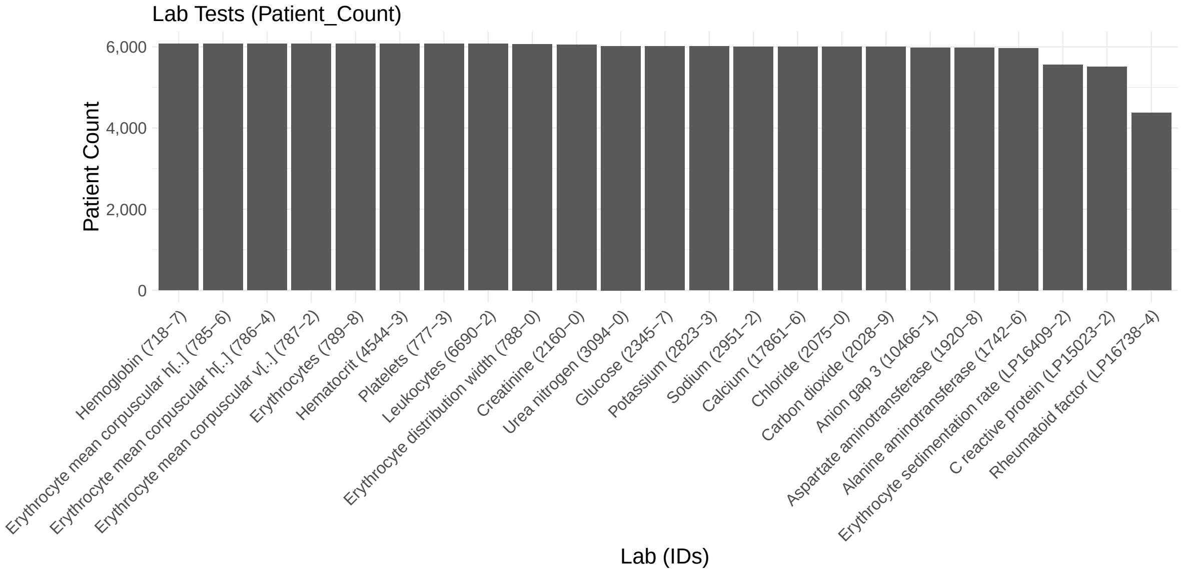
```r
if (FALSE) {
  data_rxnorm <- extract_data_for_visualization(
    sqlite_file = "./RA_intermediary.sqlite",
    prefix = "RXNORM:",
    top_n = 20
  )
}

# This will generate tables including the top 20 most frequent total counts
# and patient counts across all `Parent_Code` with `RXNORM:` prefix."

# Step 3
plot_visualized_data(
  data = data_rxnorm,
  count_column = "Patient_Count",
  prefix = "RXNORM:",
  description_label = "RXNORM Medications"
)
```



Example 2: This example incoporates the usage of additional variables

- `additional_vars <-` indicates that we have three additional variables of interests: `Erythrocyte sedimentation rate`, `C reactive protein` and `Rheumatoid factor`. The function will search these names with description column in the dictionary to get corresponding code for data extraction in the intermediary data file.
- `prefix = "LOINC:"` indicates the prefix of the Parent_Code of our choice in the dataset is LOINC:.
- `additional_vars = additional_vars` calls for the argument that we want to include additional variables in additional to our default top 20 most frequent observations.
- `top_n` is not specified here, thus the function will automatically return with the default value of top 20 in additional to variables of interests.
- `dict_prefix = "LOINC:"` is not shown as the prefix in dataset and dictionary is the same. If necessary, this argument indicates the prefix of the Parent_Code of our choice in the dictionary.
- `description_label = "Lab Tests"` indicates the content to be placed in the title.
- `data = data_add` with `count_column = "Patient_Counts"` returns the histogram of Patient counts.

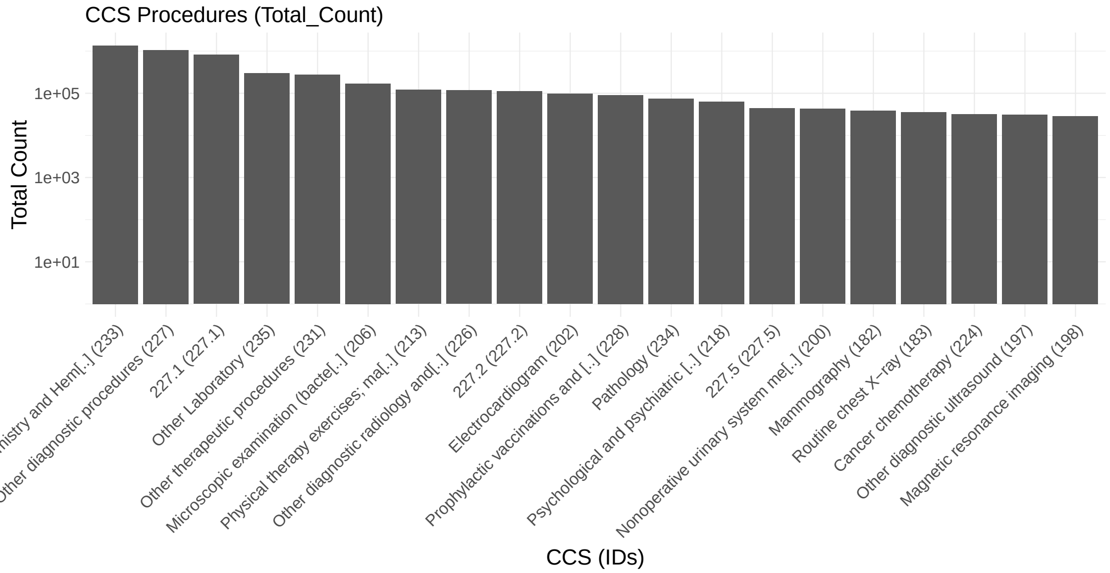
```r
if (FALSE) {
additional_vars <- c("Erythrocyte sedimentation rate", "C reactive protein",
                     "Rheumatoid factor")
# Step 2
data_add <- extract_data_for_visualization(
  sqlite_file = "./RA_intermediary.sqlite",
  prefix = "LOINC:",
  additional_vars = additional_vars
 )
}

# Step 3
plot_visualized_data(
  data = data_add,
  count_column = "Patient_Count",
  prefix = "LOINC:",
  description_label = "Lab Tests"
)
```


Lab Tests (Patient_Count)
Patient Count
6,000
4,000
2,000
0
Hemoglobin (718–7)
Erythrocyte mean corpuscular h[..] (785–6)
Erythrocyte mean corpuscular h[..] (786–4)
Erythrocyte mean corpuscular v[..] (787–2)
Erythrocytes (789–8)
Hematocrit (4544–3)
Platelets (777–3)
Leukocytes (6690–2)
Erythrocyte distribution width (788–0)
Creatinine (2160–0)
Urea nitrogen (3094–0)
Glucose (2345–7)
Potassium (2823–3)
Sodium (2951–2)
Calcium (17861–6)
Chloride (2075–0)
Carbon dioxide (2028–9)
Anion gap 3 (10466–1)
Aspartate aminotransferase (1920–8)
Alanine aminotransferase (1742–6)
Erythrocyte sedimentation rate (LP16409–2)
C reactive protein (LP15023–2)
Rheumatoid factor (LP16738–4)
Lab (IDs)

Example 3: This example explained the case where prefix in dataset & dictionary is different, and incoporates the usage of logarithmic scaling.

- `prefix = "CCS-PCS:"` indicates the prefix of the Parent_Code of our choice in the dataset is CCS-PCS:.
- `dict_prefix = "CCS:"` is specified to indicating the prefix of the Parent_Code of our choice in the dictionary is CCS:. Note that the prefix in the dataset and dictionary is different, therefore we need to address the correct prefix for each.
- `log_scale = TRUE` enables logarithmic scaling for the total count plot. By default, `log_scale = FALSE`

```r
# Step 2
if (FALSE) {
data_ccs <- extract_data_for_visualization(
  sqlite_file = "./RA_intermediary.sqlite",
  prefix = "CCS-PCS:",
  top_n = 20,
  dict_prefix = "CCS:"
 )
}

# Step 3
plot_visualized_data(
  data = data_ccs,
  count_column = "Total_Count",
  prefix = "CCS-PCS:",
  description_label = "CCS Procedures",
  log_scale = TRUE
)
```

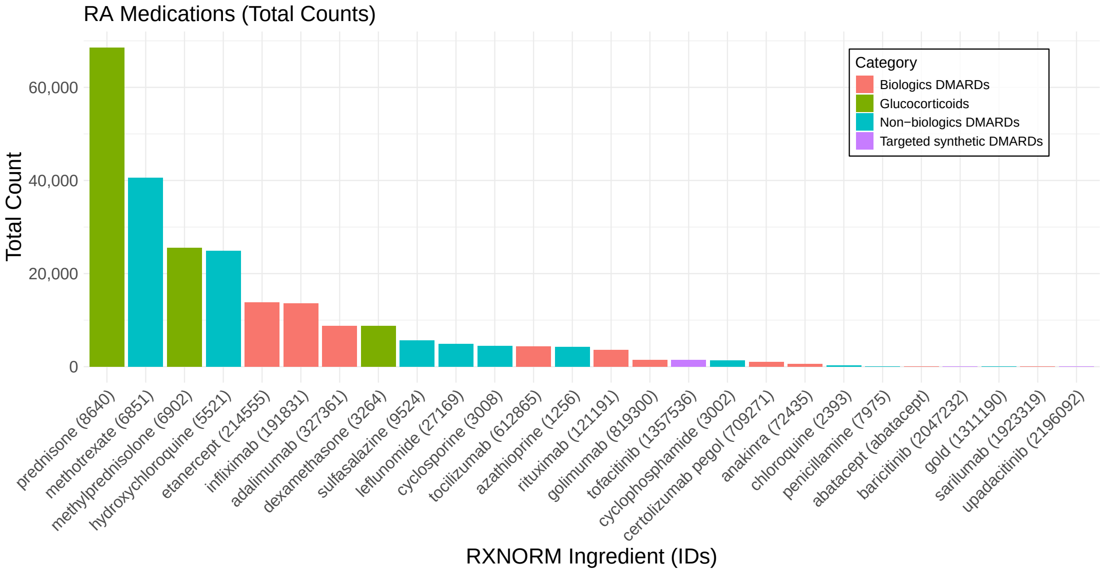


Example 4: This example introduces the scenarios where we want to discover a list of items with categories.

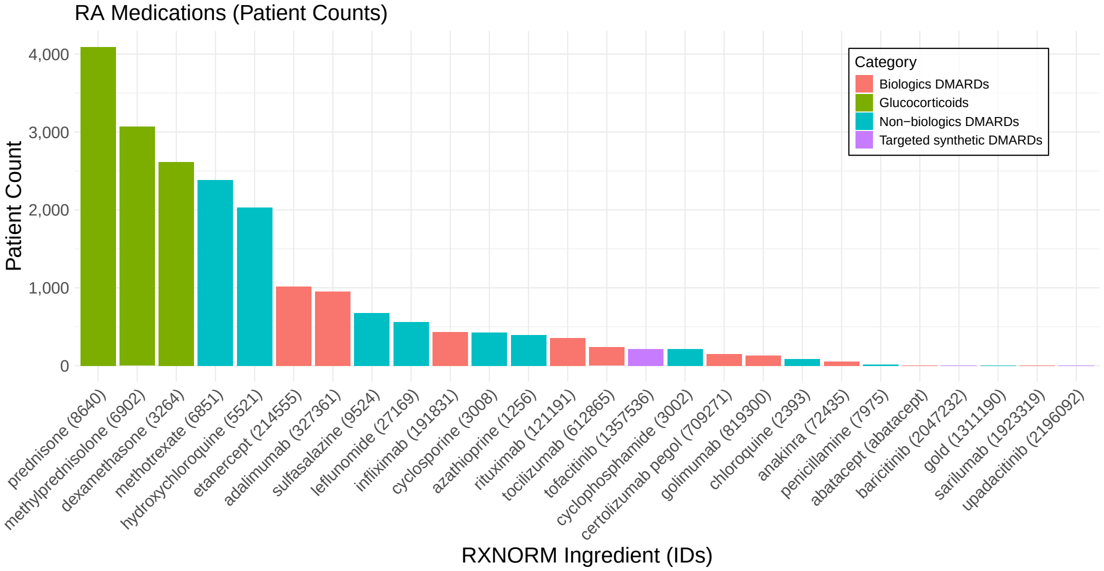
- `wanted_items` shows a data frame that we input for the function to first search by descrption to get code then go back to the intermediary file for data extraction. And in this case, most frequent count `top_n` will not be included.
- `wanted_items_df` specifies a user-defined list of items to focus on, categorized by type and category.
- `prefix = "RXNORM:"` indicates the prefix of the Parent_Code of our choice in the dataset is RXNORM:.
- `dict_prefix = "RXNORM:"` is not shown as the prefix in dataset and dictionary is the same. If necessary, this argument indicates the prefix of the Parent_Code of our choice in the dictionary.
- `data = data_add` without `count_column = "Total_Count"` or `count_column = "Patient_Count"` returns both plots.

```r
wanted_items <- data.frame(
  Type = rep("MED", 26),
  Category = c(rep("Non-biologics DMARDs", 10),
               rep("Biologics DMARDs", 10),
               rep("Targeted synthetic DMARDs", 3),
               rep("Glucocorticoids", 3)),
  Name = c('Azathioprine', 'Cyclophosphamide', 'Hydroxychloroquine',
           'Leflunomide', 'Methotrexate', 'Sulfasalazine', 'Gold',
           'Penicillamine', 'Chloroquine', 'Cyclosporine',
           'Adalimumab', 'Certolizumab pegol', 'Etanercept',
           'Golimumab', 'Infliximab', 'Abatacept', 'Anakinra',
           'Rituximab', 'Sarilumab', 'Tocilizumab',
           'Baricitinib', 'Tofacitinib', 'Upadacitinib',
           'Methylprednisolone', 'Prednisone', 'Dexamethasone')
)

# Step 2
if (FALSE) {
data_want <- extract_data_for_visualization(
  sqlite_file = "./RA_intermediary.sqlite",
  prefix = "RXNORM:",
  wanted_items_df = wanted_items
 )
}

# Step 3
plot_visualized_data(
  data = data_want,
  prefix = "RXNORM:",
  description_label = "RA Medications"
)
```

```
## $Total_Count_Plot
```



```
##
## $Patient_Count_Plot
```


RA Medications (Patient Counts)
Patient Count
Category
Biologics DMARDs
Glucocorticoids
Non−biologics DMARDs
Targeted synthetic DMARDs
prednisone (8640)
methylprednisolone (6902)
dexamethasone (3264)
methotrexate (6851)
hydroxychloroquine (5521)
etanercept (214555)
adalimumab (327361)
sulfasalazine (9524)
leflunomide (27169)
infliximab (191831)
cyclosporine (3008)
azathioprine (1256)
rituximab (121191)
tocilizumab (612865)
tofacitinib (1357536)
cyclophosphamide (3002)
certolizumab pegol (709271)
golimumab (819300)
chloroquine (2393)
anakinra (72435)
penicillamine (7975)
abatacept (abatacept)
baricitinib (2047232)
gold (1311190)
sarilumab (1923319)
upadacitinib (2196092)
RXNORM Ingredient (IDs)