Data Preprocessing

H. David Shea

10 Jul 2021

Contents

(Note: Updated and modified from the hst953-edx github version.)

Source MIMIC-III demo version data

In the original version (on the hst953-edx github site), they used the MIMIC-III demo version directly loaded. Now, I have in *project_base_dir*/database/mimic3.db the SQLite version of the full MIMIC-III v1.4 database loaded. I'll use that in the processing below - with some pre-coded inclusion criteria to extract just the demo data. The following code chunk attaches the database and loads auxiliary functions for extracting database data (`db_functions.R`) and for doing some MIMIC data interpretation and pre-processing (`mimic3_meta_data.R`) - including the processing to get just the demo data.

```r
base_dir <- here::here("")
db_dir <- fs::path(base_dir, "database")
db_file <- fs::path(db_dir, "mimic3.db")

if(dbCanConnect(RSQLite::SQLite(), db_file)) {
    mimic3 <- dbConnect(RSQLite::SQLite(), db_file)
}

source(fs::path(base_dir, "db_functions.R"))
source(fs::path(base_dir, "mimic3_meta_data.R"))
```

Data integration

Exercise

Aim: This exercise involves combining the separate output datasets exported from separate MIMIC queries into a consistent larger dataset table.

To ensure that the associated observations or rows from the different datasets match up, the right column ID must be used. For example, in MIMIC `SUBJECT_ID` is used to identify each individual patient, so includes their date of birth (DOB), date of death (DOD) and various other clinical detail and laboratory values. Likewise, the hospital admission ID - `HADM_ID` - is used to specifically identify various events and outcomes from an unique hospital admission; and is also in turn associated with the `SUBJECT_ID` of the patient who was involved in that particular hospital admission. Tables pulled from MIMIC can have one or more ID columns. The different tables exported from MIMIC may share some ID columns, which allows us to 'merge' them together, matching up the rows correctly using the unique ID values in their shared ID columns.

To demonstrate this with MIMIC data, some base extraction routines are used to extract some data from the `ADMISSIONS` and `ICUSTAYS` tables. We will use these extracted files to show how to merge datasets in R.

1. Base data extractions:

```
adm <- db_get_admissions(mimic3, where = demo_subject_ids)

str(adm)
#> tibble [129 x 18] (S3: tbl_df/tbl/data.frame)
#>  $ SUBJECT_ID         : int [1:129] 10006 10011 10013 10017 10019 10026 10027 10029 10032 10033 ...
#>  $ HADM_ID            : int [1:129] 142345 105331 165520 199207 177759 103770 199395 132349 140372
#>  $ ADMITTIME          : POSIXct[1:129], format: "2164-10-23 21:09:00" "2126-08-14 22:32:00" ...
#>  $ DISCHTIME          : POSIXct[1:129], format: "2164-11-01 17:15:00" "2126-08-28 18:59:00" ...
#>  $ DEATHTIME          : POSIXct[1:129], format: NA "2126-08-28 18:59:00" ...
#>  $ ADMISSION_TYPE     : chr [1:129] "EMERGENCY" "EMERGENCY" "EMERGENCY" "EMERGENCY" ...
#>  $ ADMISSION_LOCATION : chr [1:129] "EMERGENCY ROOM ADMIT" "TRANSFER FROM HOSP/EXTRAM" "TRANSFER FR
#>  $ DISCHARGE_LOCATION : chr [1:129] "HOME HEALTH CARE" "DEAD/EXPIRED" "DEAD/EXPIRED" "SNF" ...
#>  $ INSURANCE          : chr [1:129] "Medicare" "Private" "Medicare" "Medicare" ...
#>  $ LANGUAGE           : chr [1:129] "" "" "" "" ...
#>  $ RELIGION           : chr [1:129] "CATHOLIC" "CATHOLIC" "CATHOLIC" "CATHOLIC" ...
#>  $ MARITAL_STATUS     : chr [1:129] "SEPARATED" "SINGLE" "" "DIVORCED" ...
#>  $ ETHNICITY          : chr [1:129] "BLACK/AFRICAN AMERICAN" "UNKNOWN/NOT SPECIFIED" "UNKNOWN/NOT S
#>  $ EDREGTIME          : POSIXct[1:129], format: "2164-10-23 16:43:00" NA ...
#>  $ EDOUTTIME          : POSIXct[1:129], format: "2164-10-23 23:00:00" NA ...
#>  $ DIAGNOSIS          : chr [1:129] "SEPSIS" "HEPATITIS B" "SEPSIS" "HUMERAL FRACTURE" ...
#>  $ HOSPITAL_EXPIRE_FLAG: int [1:129] 0 1 1 0 1 0 0 0 0 0 ...
#>  $ HAS_CHARTEVENTS_DATA: int [1:129] 1 1 1 1 1 1 1 1 1 1 ...

icu <- db_get_icustays(mimic3, where = demo_subject_ids)

str(icu)
#> tibble [136 x 11] (S3: tbl_df/tbl/data.frame)
#>  $ SUBJECT_ID    : int [1:136] 10006 10011 10013 10017 10019 10026 10027 10029 10032 10033 ...
#>  $ HADM_ID       : int [1:136] 142345 105331 165520 199207 177759 103770 199395 132349 140372 157235
#>  $ ICUSTAY_ID    : int [1:136] 206504 232110 264446 204881 228977 277021 286020 226055 267090 254543
#>  $ DBSOURCE      : chr [1:136] "carevue" "carevue" "carevue" "carevue" ...
#>  $ FIRST_CAREUNIT: chr [1:136] "MICU" "MICU" "MICU" "CCU" ...
#>  $ LAST_CAREUNIT : chr [1:136] "MICU" "MICU" "MICU" "CCU" ...
```

```
#>  $ FIRST_WARDID  : int [1:136] 52 15 15 7 15 33 12 33 52 33 ...
#>  $ LAST_WARDID   : int [1:136] 52 15 15 7 15 33 12 33 52 33 ...
#>  $ INTIME        : POSIXct[1:136], format: "2164-10-23 21:10:15" "2126-08-14 22:34:00" ...
#>  $ OUTTIME       : POSIXct[1:136], format: "2164-10-25 12:21:07" "2126-08-28 18:59:00" ...
#>  $ LOS           : num [1:136] 1.63 13.85 2.65 2.14 1.29 ...
```

These base data extraction routines use R's `DBI` package interface to execute SQL statements and pull back data in nicely formatted tibbles.

2. R code: Demonstrating data integration

Merging `ADMISSIONS` and `ICUSTAYS`: to get the rows to match up correctly, we need to merge on both the `SUBJECT_ID` and `HADM_ID`. This is because each subject/patient could have multiple `HADM_ID` from different hospital `ADMISSIONS` during the EHR course of the MIMIC database.

(Note: in this updated version, I have switched to using the `tidyverse` package relational wrangling techniques instead of the older base `merge` function.)

```
icu_adm <- adm %>%
  left_join(icu, by = c("SUBJECT_ID", "HADM_ID"))

str(icu_adm)
#> tibble [136 x 27] (S3: tbl_df/tbl/data.frame)
#>  $ SUBJECT_ID          : int [1:136] 10006 10011 10013 10017 10019 10026 10027 10029 10032 10033 ...
#>  $ HADM_ID             : int [1:136] 142345 105331 165520 199207 177759 103770 199395 132349 140372
#>  $ ADMITTIME           : POSIXct[1:136], format: "2164-10-23 21:09:00" "2126-08-14 22:32:00" ...
#>  $ DISCHTIME           : POSIXct[1:136], format: "2164-11-01 17:15:00" "2126-08-28 18:59:00" ...
#>  $ DEATHTIME           : POSIXct[1:136], format: NA "2126-08-28 18:59:00" ...
#>  $ ADMISSION_TYPE      : chr [1:136] "EMERGENCY" "EMERGENCY" "EMERGENCY" "EMERGENCY" ...
#>  $ ADMISSION_LOCATION  : chr [1:136] "EMERGENCY ROOM ADMIT" "TRANSFER FROM HOSP/EXTRAM" "TRANSFER FR
#>  $ DISCHARGE_LOCATION  : chr [1:136] "HOME HEALTH CARE" "DEAD/EXPIRED" "DEAD/EXPIRED" "SNF" ...
#>  $ INSURANCE           : chr [1:136] "Medicare" "Private" "Medicare" "Medicare" ...
#>  $ LANGUAGE            : chr [1:136] "" "" "" "" ...
#>  $ RELIGION            : chr [1:136] "CATHOLIC" "CATHOLIC" "CATHOLIC" "CATHOLIC" ...
#>  $ MARITAL_STATUS      : chr [1:136] "SEPARATED" "SINGLE" "" "DIVORCED" ...
#>  $ ETHNICITY           : chr [1:136] "BLACK/AFRICAN AMERICAN" "UNKNOWN/NOT SPECIFIED" "UNKNOWN/NOT S
#>  $ EDREGTIME           : POSIXct[1:136], format: "2164-10-23 16:43:00" NA ...
#>  $ EDOUTTIME           : POSIXct[1:136], format: "2164-10-23 23:00:00" NA ...
#>  $ DIAGNOSIS           : chr [1:136] "SEPSIS" "HEPATITIS B" "SEPSIS" "HUMERAL FRACTURE" ...
#>  $ HOSPITAL_EXPIRE_FLAG: int [1:136] 0 1 1 0 1 0 0 0 0 0 ...
#>  $ HAS_CHARTEVENTS_DATA: int [1:136] 1 1 1 1 1 1 1 1 1 1 ...
#>  $ ICUSTAY_ID          : int [1:136] 206504 232110 264446 204881 228977 277021 286020 226055 267090
#>  $ DBSOURCE            : chr [1:136] "carevue" "carevue" "carevue" "carevue" ...
#>  $ FIRST_CAREUNIT      : chr [1:136] "MICU" "MICU" "MICU" "CCU" ...
#>  $ LAST_CAREUNIT       : chr [1:136] "MICU" "MICU" "MICU" "CCU" ...
#>  $ FIRST_WARDID        : int [1:136] 52 15 15 7 15 33 12 33 52 33 ...
#>  $ LAST_WARDID         : int [1:136] 52 15 15 7 15 33 12 33 52 33 ...
#>  $ INTIME              : POSIXct[1:136], format: "2164-10-23 21:10:15" "2126-08-14 22:34:00" ...
#>  $ OUTTIME             : POSIXct[1:136], format: "2164-10-25 12:21:07" "2126-08-28 18:59:00" ...
#>  $ LOS                 : num [1:136] 1.63 13.85 2.65 2.14 1.29 ...
```

Note: in the same way each subject/patient could have multiple `HADM_ID` from different hospital admissions during the EHR course of the MIMIC database, each `HADM_ID` can have multiple `ICUSTAY_ID`.

Data transformation

Exercise

Aim: To transform the presentation of data values in some ways so that the new format is more suitable for the subsequent statistical analysis. The main processes involved are normalization, aggregation and generalization.

1. Base data extractions:

This extraction relies on a (very complicated) view that was provided with the original course material on the github site for the course. As such, it uses a lower level database routine for extraction.

The view uses data from the `DIAGNOSES_ICD`, `DRGCODES` and `ADMISSIONS` tables to implement a version of the Elixhauser Comorbidity Index. The Elixhauser Comorbidity Index is a method of categorizing comorbidities of patients based on the International Classification of Diseases (ICD) diagnosis codes.

```
Elixhauser <- db_select_data(mimic3, "SELECT * FROM ELIXHAUSER_AHRQ")

str(Elixhauser)
#> tibble [129 x 32] (S3: tbl_df/tbl/data.frame)
#>  $ SUBJECT_ID              : int [1:129] 10056 42430 41914 42135 44228 10026 40612 10117 10011 41983
#>  $ HADM_ID                 : int [1:129] 100375 100969 101361 102203 103379 103770 104697 105150 105
#>  $ CONGESTIVE_HEART_FAILURE: int [1:129] 1 0 0 0 0 0 0 0 0 0 ...
#>  $ CARDIAC_ARRHYTHMIAS     : int [1:129] 0 1 0 0 0 0 0 0 0 0 ...
#>  $ VALVULAR_DISEASE        : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ PULMONARY_CIRCULATION   : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ PERIPHERAL_VASCULAR     : int [1:129] 0 1 0 0 0 0 0 0 0 0 ...
#>  $ HYPERTENSION            : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ PARALYSIS               : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ OTHER_NEUROLOGICAL      : int [1:129] 0 0 0 0 0 0 1 0 0 0 ...
#>  $ CHRONIC_PULMONARY       : int [1:129] 0 1 0 0 0 0 1 0 0 0 ...
#>  $ DIABETES_UNCOMPLICATED  : int [1:129] 0 1 1 0 0 0 0 0 0 0 ...
#>  $ DIABETES_COMPLICATED    : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ HYPOTHYROIDISM          : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ RENAL_FAILURE           : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ LIVER_DISEASE           : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ PEPTIC_ULCER            : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ AIDS                    : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ LYMPHOMA                : int [1:129] 0 0 0 0 0 0 0 1 0 0 ...
#>  $ METASTATIC_CANCER       : int [1:129] 0 0 0 0 1 0 0 0 0 0 ...
#>  $ SOLID_TUMOR             : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ RHEUMATOID_ARTHRITIS    : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ COAGULOPATHY            : int [1:129] 0 0 0 0 0 0 0 0 1 0 ...
#>  $ OBESITY                 : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ WEIGHT_LOSS             : int [1:129] 0 0 0 1 0 0 0 0 0 0 ...
#>  $ FLUID_ELECTROLYTE       : int [1:129] 1 0 0 1 0 1 0 1 0 1 ...
#>  $ BLOOD_LOSS_ANEMIA       : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ DEFICIENCY_ANEMIAS      : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ ALCOHOL_ABUSE           : int [1:129] 0 0 1 1 0 0 0 0 0 0 ...
#>  $ DRUG_ABUSE              : int [1:129] 0 0 0 0 0 0 0 0 1 0 ...
#>  $ PSYCHOSES               : int [1:129] 0 0 0 0 0 0 1 0 0 0 ...
#>  $ DEPRESSION              : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
```

Note the total number of rows (obs) and columns (variables) in `Elixhauser` tibble results.

2. R code: Demonstrating data transformation

Aggregation and Normalization steps

Here we `mutate` the `Elixhauser` tibble to add an `OVERALL_SCORE` column which is the sum of all of the co-morbidity values, and a normalized value - `OVERALL_NML` - of each observation's `OVERALL_SCORE` divided by the maximum `OVERALL_SCORE` for the sample.

(Note: in this updated version, the aggregation and normalization are done in a single `mutate` rather than a series of base R steps.)

```
Elixhauser <- Elixhauser %>%
  mutate(
    OVERALL_SCORE = rowSums(select(., CONGESTIVE_HEART_FAILURE:DEPRESSION)),
    OVERALL_NML = OVERALL_SCORE/max(OVERALL_SCORE)
  )

str(Elixhauser)
#> tibble [129 x 34] (S3: tbl_df/tbl/data.frame)
#>  $ SUBJECT_ID              : int [1:129] 10056 42430 41914 42135 44228 10026 40612 10117 10011 41983
#>  $ HADM_ID                 : int [1:129] 100375 100969 101361 102203 103379 103770 104697 105150 105
#>  $ CONGESTIVE_HEART_FAILURE: int [1:129] 1 0 0 0 0 0 0 0 0 0 ...
#>  $ CARDIAC_ARRHYTHMIAS     : int [1:129] 0 1 0 0 0 0 0 0 0 0 ...
#>  $ VALVULAR_DISEASE        : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ PULMONARY_CIRCULATION   : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ PERIPHERAL_VASCULAR     : int [1:129] 0 1 0 0 0 0 0 0 0 0 ...
#>  $ HYPERTENSION            : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ PARALYSIS               : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ OTHER_NEUROLOGICAL      : int [1:129] 0 0 0 0 0 0 1 0 0 0 ...
#>  $ CHRONIC_PULMONARY       : int [1:129] 0 1 0 0 0 0 1 0 0 0 ...
#>  $ DIABETES_UNCOMPLICATED  : int [1:129] 0 1 1 0 0 0 0 0 0 0 ...
#>  $ DIABETES_COMPLICATED    : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ HYPOTHYROIDISM          : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ RENAL_FAILURE           : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ LIVER_DISEASE           : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ PEPTIC_ULCER            : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ AIDS                    : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ LYMPHOMA                : int [1:129] 0 0 0 0 0 0 0 1 0 0 ...
#>  $ METASTATIC_CANCER       : int [1:129] 0 0 0 0 1 0 0 0 0 0 ...
#>  $ SOLID_TUMOR             : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ RHEUMATOID_ARTHRITIS    : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ COAGULOPATHY            : int [1:129] 0 0 0 0 0 0 0 0 1 0 ...
#>  $ OBESITY                 : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ WEIGHT_LOSS             : int [1:129] 0 0 0 1 0 0 0 0 0 0 ...
#>  $ FLUID_ELECTROLYTE       : int [1:129] 1 0 0 1 0 1 0 1 0 1 ...
#>  $ BLOOD_LOSS_ANEMIA       : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ DEFICIENCY_ANEMIAS      : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
#>  $ ALCOHOL_ABUSE           : int [1:129] 0 0 1 1 0 0 0 0 0 0 ...
#>  $ DRUG_ABUSE              : int [1:129] 0 0 0 0 0 0 0 0 1 0 ...
#>  $ PSYCHOSES               : int [1:129] 0 0 0 0 0 0 1 0 0 0 ...
#>  $ DEPRESSION              : int [1:129] 0 0 0 0 0 0 0 0 0 0 ...
```

```
#>  $ OVERALL_SCORE            : num [1:129] 2 4 2 3 1 1 3 2 2 1 ...
#>  $ OVERALL_NML              : num [1:129] 0.222 0.444 0.222 0.333 0.111 ...
```

Generalization Step

Aim: Consider only the group of patients sicker than the average Elixhauser score. Here, we will create a new tibble `Elixhauser_sicker_sample` of those subjects from the whole sample who have an `OVERALL_NML` value >= 0.5 - the sickest half of the population based on number of morbidity indications. For this tibble, we will keep the `SUBJECT_ID`, `HADM_ID`, and `OVERAL_NML` values only. We will save that to the CSV file: `sicker_sample.csv`

```
Elixhauser_sicker_sample <- Elixhauser %>%
  filter(OVERALL_NML >= 0.5) %>%
  select(SUBJECT_ID, HADM_ID, OVERALL_NML)

str(Elixhauser_sicker_sample)
#> tibble [17 x 3] (S3: tbl_df/tbl/data.frame)
#>  $ SUBJECT_ID : int [1:17] 43735 41795 40655 10045 43798 41795 10006 42075 41976 43881 ...
#>  $ HADM_ID    : int [1:17] 112662 118192 126002 126949 130870 138132 142345 151323 153826 167021 ...
#>  $ OVERALL_NML: num [1:17] 0.556 1 0.556 0.556 0.556 ...

write.csv(Elixhauser_sicker_sample,
          fs::path(base_dir, "exercises/data_preprocessing/Elixhauser_sicker_sample.csv"), row.names = F
```

Data reduction

Exercise

Aim: To reduce or reshape the input data by means of a more effective representation of the dataset without compromising the integrity of the original data. One element of data reduction is eliminating redundant records while preserving needed data, which we will demonstrate in Part 1. The other element involves reshaping the dataset into a "tidy" format, which we will demonstrate in Part 2.

Part 1: Eliminating Redundant Records

To demonstrate this with an example, we will look at multiple records of glucose laboratory values for each patient. This query selects all the non-null measurements of glucose values for all the patients in the MIMIC database.

(Note: I was not sure if - in the original course example - they used the demo database or the full database for this analyses, so I went with the full database. That will be my default from now on unless it is otherwise specified in the older Rmarkdown code.)

1. Base data extractions:

```
where <- "WHERE (itemid = 50931 OR itemid = 50809)
          AND   valuenum IS NOT null
          AND   hadm_id IS NOT null"
all_glucose <- db_get_labevents(mimic3, where = where)
```

```
str(all_glucose)
#> tibble [788,730 x 8] (S3: tbl_df/tbl/data.frame)
#>  $ SUBJECT_ID: int [1:788730] 3 3 3 3 3 3 6 6 6 6 ...
#>  $ HADM_ID   : int [1:788730] 145834 145834 145834 145834 145834 145834 107064 107064 107064 107064
#>  $ ITEMID    : int [1:788730] 50809 50809 50809 50809 50809 50809 50809 50809 50809 50809 ...
#>  $ CHARTTIME : POSIXct[1:788730], format: "2101-10-20 20:04:00" "2101-10-20 21:51:00" ...
#>  $ VALUE     : chr [1:788730] "265" "267" "299" "294" ...
#>  $ VALUENUM  : num [1:788730] 265 267 299 294 140 128 106 146 151 145 ...
#>  $ VALUEUOM  : chr [1:788730] "mg/dL" "mg/dL" "mg/dL" "mg/dL" ...
#>  $ FLAG      : chr [1:788730] "abnormal" "abnormal" "abnormal" "abnormal" ...
```

2. R code: Demonstrating data reduction

There are a variety of methods that can be chosen to aggregate records. In this case we will look at averaging multiple glucose records into a single average glucose for each patient. Other options which may be chosen include using the first recorded value, a minimum or maximum value, etc. For a basic example, the following code demonstrates data reduction by averaging all of the multiple records of glucose into a single record per patient hospital admission. The code uses the `tidyverse` package functions `group_by` and `aggregate` to calculate average glucose values for each distinct `HADM_ID`:

```
avg_glucose <- all_glucose %>%
    arrange(HADM_ID, SUBJECT_ID) %>%
    group_by(HADM_ID, SUBJECT_ID) %>%
    summarize(
      VALUENUM = mean(VALUENUM, na.rm = TRUE)
    )

avg_glucose
#> # A tibble: 50,653 x 3
#> # Groups:   HADM_ID [50,653]
#>    HADM_ID SUBJECT_ID VALUENUM
#>      <int>      <int>    <dbl>
#>  1  100001      58526    165.
#>  2  100003      54610     96.8
#>  3  100006       9895    105
#>  4  100007      23018    118.
#>  5  100009        533    152.
#>  6  100010      55853    112.
#>  7  100011      87977    132.
#>  8  100012      60039    111.
#>  9  100016      68591     81.8
#> 10  100017      16229     77
#> # ... with 50,643 more rows
```

Part 2: Reshaping Dataset

Ideally, we want a “tidy” dataset reorganized in such a way so it follows these 3 rules:

- 1. Each variable forms a column
- 2. Each observation forms a row
- 3. Each value has its own cell

Datasets exported from MIMIC usually are fairly "tidy" already. Therefore, we will construct our own data frame here for ease of demonstration for rule 3. We will also demonstrate how to use some common data tidying packages.

R code: To mirror our own MIMIC dataframe, we construct a dataset with a column of `subject_id` and a column with a list of `diagnoses` for the admission.

(Note: this versions of the code is from the original with only slight modifications.)

```
diag <- data.frame(subject_id = 1:6, diagnosis = c("PNA, CHF", "DKA", "DKA, UTI", "AF, CHF", "AF", "CHF

diag
#>   subject_id diagnosis
#> 1          1  PNA, CHF
#> 2          2       DKA
#> 3          3  DKA, UTI
#> 4          4   AF, CHF
#> 5          5        AF
#> 6          6       CHF
```

Note that the dataset above is not "tidy". There are multiple categorical variables in column "diagnosis" - breaks "tidy" data rule 1. There are multiple values in column "diagnosis" - breaks "tidy" data rule 3. There are many ways to "tidy" and reshape this dataset. We will show one way to do this by making use of R packages "splitstackshape" and "tidyr" to make reshaping the dataset easier.

R package example 1 - `splitstackshape` package functions: The function, `cSplit` from the `splitstackshape` package, can split the multiple categorical values in each cell of column `diagnosis` into different columns, `diagnosis_1` and `diagnosis_2`. If the argument, direction, for `cSplit` is not specified, then the function splits the original dataset "wide".

```
diag2 <- cSplit(diag, "diagnosis", ",")

diag2
#>    subject_id diagnosis_1 diagnosis_2
#> 1:          1         PNA         CHF
#> 2:          2         DKA        <NA>
#> 3:          3         DKA         UTI
#> 4:          4          AF         CHF
#> 5:          5          AF        <NA>
#> 6:          6         CHF        <NA>
```

One could possibly keep it as this if one is interested in primary and secondary diagnoses (though it is not strictly "tidy" yet). Alternatively, if the direction argument is specified as "long", then `cSplit` splits the function "long" like so:

```
diag3 <- cSplit(diag, "diagnosis", ",", direction = "long")

diag3
#>    subject_id diagnosis
#> 1:          1       PNA
#> 2:          1       CHF
#> 3:          2       DKA
#> 4:          3       DKA
#> 5:          3       UTI
```

```
#> 6:          4        AF
#> 7:          4       CHF
#> 8:          5        AF
#> 9:          6       CHF
```

Note diag3 is still not "tidy" as there are still multiple categorical variables under column diagnosis-but we no longer have multiple values per cell.

R package example 2 - `tidyr` package functions: To further "tidy" the dataset, package `tidyr` is pretty useful.

The aim is to split each categorical variable under column `diagnosis` into their own columns with 1 = having the diagnosis and 0 = not having the diagnosis. To do this we first construct a third column, `yes`, that hold all the 1 values initially (because the function we are going to use requires a value column that corresponds to the multiple categories column we want to 'spread' out).

```
diag3$yes <- rep(1, nrow(diag3))

diag3
#>    subject_id diagnosis yes
#> 1:          1       PNA   1
#> 2:          1       CHF   1
#> 3:          2       DKA   1
#> 4:          3       DKA   1
#> 5:          3       UTI   1
#> 6:          4        AF   1
#> 7:          4       CHF   1
#> 8:          5        AF   1
#> 9:          6       CHF   1
```

Then we can use the `spread` function to split each categorical variables into their own columns. The argument, fill = 0, replaces the missing values.

```
diag4 <- spread(diag3, diagnosis, yes, fill = 0)

diag4
#>    subject_id AF CHF DKA PNA UTI
#> 1:          1  0   1   0   1   0
#> 2:          2  0   0   1   0   0
#> 3:          3  0   0   1   0   1
#> 4:          4  1   1   0   0   0
#> 5:          5  1   0   0   0   0
#> 6:          6  0   1   0   0   0
```

One can see that this dataset is now "tidy", as it follows all three "tidy" data rules.