Big Data Management (HINF5018.03)

A Lecture for Data Visualization using Open Database Connectivity

- *library(RODBC) & library(jsonlite) & library(ggplot2)*

Min-hyung Kim, M.D., M.S.

Fellow in Healthcare Policy and Research

Division of Health Informatics

Joan & Sanford I. Weill Medical College of Cornell University

April 3, 2018

[CONTENTS]

- library(RODBC)
  - Establishing a connection to a database
  - Query table names, column (variable) names, data types, primary keys
- Refine the research question
  - Unit of observation (!)
  - e.g.) period prevalence (proportion) by gender
- library(ggplot2)
  - Grammar of graphics
  - Syntax of library(ggplot2)
- library(jsonlite)

library(RODBC)

Establishing a connection to a database
Query table names, column (variable) names, data types, primary keys

Why use ODBC connection?

- Your local computer hardware cannot load the entire data you would like to analyze.
- Exporting the data into hard drive in csv format and then re-loading into statistical software may be time-consuming and error-prone process.
- The computing power of the database server will be higher than that of your local computer.
- The database server may employ efficient indexing algorithms that outperforms your statistical software for certain queries.

Establishing a connection to a database

```
# An appropriate ODBC driver should be installed in your
computer first!

library(RODBC)
channel = odbcDriverConnect("Driver= {SQL Server};
server=VITS-ARCHSQLP04; database=OMOP_SYNTHETIC")
```

Query distinct table names

```
sqlQuery(channel, "
select distinct TABLE_NAME
from information_schema.columns
;")

#              TABLE_NAME
# 1              CARE_SITE
# 2     CDM_DRUG_EXPOSURE
# 3                CONCEPT
# 4      CONCEPT_ANCESTOR
# 5         CONCEPT_CLASS
# 6  CONCEPT_RELATIONSHIP
# 7       CONCEPT_SYNONYM
# 8         CONDITION_ERA
# 9  CONDITION_OCCURRENCE
# 10                DEATH
# 11      DEVICE_EXPOSURE
# 12             DRUG_ERA
# 13        DRUG_EXPOSURE
# 14             LOCATION
# 15          MEASUREMENT
# 16          OBSERVATION
# 17   OBSERVATION_PERIOD
# 18               PERSON
# 19 PROCEDURE_OCCURRENCE
# 20             PROVIDER
# 21           testingnote
# 22     VISIT_OCCURRENCE
```

Query table names, column names, data types, is_nullable

```
sqlQuery(channel, "
SELECT TABLE_NAME, COLUMN_NAME, IS_NULLABLE, DATA_TYPE
FROM information_schema.columns
WHERE TABLE_NAME in ('PERSON', 'VISIT_OCCURRENCE', 'CONDITION_OCCURRENCE', 'LOCATION')
;")
#              TABLE_NAME                  COLUMN_NAME IS_NULLABLE DATA_TYPE
# 1  CONDITION_OCCURRENCE     CONDITION_OCCURRENCE_ID         YES   varchar
# 2  CONDITION_OCCURRENCE                   PERSON_ID         YES   varchar
# 3  CONDITION_OCCURRENCE        CONDITION_CONCEPT_ID         YES   varchar
# 4  CONDITION_OCCURRENCE        CONDITION_START_DATE         YES   varchar
# 5  CONDITION_OCCURRENCE          CONDITION_END_DATE         YES   varchar
# 6  CONDITION_OCCURRENCE   CONDITION_TYPE_CONCEPT_ID         YES   varchar
# 7  CONDITION_OCCURRENCE                 STOP_REASON         YES   varchar
# 8  CONDITION_OCCURRENCE                 PROVIDER_ID         YES   varchar
# 9  CONDITION_OCCURRENCE        VISIT_OCCURRENCE_ID         YES   varchar
# 10 CONDITION_OCCURRENCE      CONDITION_SOURCE_VALUE         YES   varchar
# 11 CONDITION_OCCURRENCE CONDITION_SOURCE_CONCEPT_ID         YES   varchar
# 12             LOCATION                 LOCATION_ID         YES   varchar
# 13             LOCATION                   ADDRESS_1         YES   varchar
# 14             LOCATION                   ADDRESS_2         YES   varchar
# 15             LOCATION                        CITY         YES   varchar
# 16             LOCATION                       STATE         YES   varchar
# 17             LOCATION                         ZIP         YES   varchar
# 18             LOCATION                      COUNTY         YES   varchar
# 19             LOCATION       LOCATION_SOURCE_VALUE         YES   varchar
# 20               PERSON                   PERSON_ID         YES   varchar
# 21               PERSON           GENDER_CONCEPT_ID         YES   varchar
# 22               PERSON              YEAR_OF_BIRTH         YES   varchar
# 23               PERSON             MONTH_OF_BIRTH         YES   varchar
# 24               PERSON               DAY_OF_BIRTH         YES   varchar
# 25               PERSON              TIME_OF_BIRTH         YES   varchar
# 26               PERSON            RACE_CONCEPT_ID         YES   varchar
# 27               PERSON        ETHNICITY_CONCEPT_ID         YES   varchar
# 28               PERSON                 LOCATION_ID         YES   varchar
# 29               PERSON                 PROVIDER_ID         YES   varchar
# 30               PERSON               CARE_SITE_ID         YES   varchar
# 31               PERSON         PERSON_SOURCE_VALUE         YES   varchar
# 32               PERSON         GENDER_SOURCE_VALUE         YES   varchar
# 33               PERSON    GENDER_SOURCE_CONCEPT_ID         YES   varchar
# 34               PERSON           RACE_SOURCE_VALUE         YES   varchar
# 35               PERSON      RACE_SOURCE_CONCEPT_ID         YES   varchar
# 36               PERSON       ETHNICITY_SOURCE_VALUE         YES   varchar
# 37               PERSON ETHNICITY_SOURCE_CONCEPT_ID         YES   varchar
# 38     VISIT_OCCURRENCE         VISIT_OCCURRENCE_ID         YES   varchar
# 39     VISIT_OCCURRENCE                   PERSON_ID         YES   varchar
# 40     VISIT_OCCURRENCE              VISIT_CONCEPT_ID         YES   varchar
# 41     VISIT_OCCURRENCE              VISIT_START_DATE         YES   varchar
# 42     VISIT_OCCURRENCE              VISIT_START_TIME         YES   varchar
# 43     VISIT_OCCURRENCE                VISIT_END_DATE         YES   varchar
# 44     VISIT_OCCURRENCE                VISIT_END_TIME         YES   varchar
# 45     VISIT_OCCURRENCE       VISIT_TYPE_CONCEPT_ID         YES   varchar
# 46     VISIT_OCCURRENCE                   PROVIDER_ID         YES   varchar
# 47     VISIT_OCCURRENCE                  CARE_SITE_ID         YES   varchar
# 48     VISIT_OCCURRENCE          VISIT_SOURCE_VALUE         YES   varchar
# 49     VISIT_OCCURRENCE     VISIT_SOURCE_CONCEPT_ID         YES   varchar
```

Query key columns & database constrants (if any)

```
sqlQuery(channel, "
SELECT *
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
;")
# [1] CONSTRAINT_CATALOG CONSTRAINT_SCHEMA  CONSTRAINT_NAME    TABLE_CATALOG
TABLE_SCHEMA       TABLE_NAME          COLUMN_NAME         ORDINAL_POSITION
# <0 rows> (or 0-length row.names)
```

Rule-out non-candidate keys.. (not unique)

```
sqlQuery(channel, "
select count(*) as nrow
        , count(distinct CONDITION_OCCURRENCE_ID) as n_distinct_CONDITION_OCCURRENCE_ID
        , count(distinct CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID) as n_distinct_VISIT_OCCURRENCE_ID
        , count(distinct CONDITION_OCCURRENCE.PERSON_ID) as n_distinct_PERSON_ID
from CONDITION_OCCURRENCE
;")
#     nrow n_distinct_CONDITION_OCCURRENCE_ID n_distinct_VISIT_OCCURRENCE_ID n_distinct_PERSON_ID
# 1 160322                              160322                          44526                 836
```

Rule-out non-candidate keys.. (not unique) text manipulation in R

```
varnames = sqlQuery(channel, "
SELECT COLUMN_NAME
FROM information_schema.columns
WHERE TABLE_NAME = 'VISIT_OCCURRENCE'
;")[[1]]

qryText = paste0(
    "select count(*) as nrow"
    , paste0(
        ", count(distinct "
        , varnames
        , ") as n_distinct_"
        , varnames
        , collapse = ""
    )
    , " from VISIT_OCCURRENCE;"
)
qryText
# [1] "select count(*) as nrow, count(distinct CARE_SITE_ID) as n_distinct_CARE_SITE_ID, count(distinct PERSON_ID) as
n_distinct_PERSON_ID, count(distinct PROVIDER_ID) as n_distinct_PROVIDER_ID, count(distinct VISIT_CONCEPT_ID) as
n_distinct_VISIT_CONCEPT_ID, count(distinct VISIT_END_DATE) as n_distinct_VISIT_END_DATE, count(distinct VISIT_END_TIME) as
n_distinct_VISIT_END_TIME, count(distinct VISIT_OCCURRENCE_ID) as n_distinct_VISIT_OCCURRENCE_ID, count(distinct
VISIT_SOURCE_CONCEPT_ID) as n_distinct_VISIT_SOURCE_CONCEPT_ID, count(distinct VISIT_SOURCE_VALUE) as
n_distinct_VISIT_SOURCE_VALUE, count(distinct VISIT_START_DATE) as n_distinct_VISIT_START_DATE, count(distinct VISIT_START_TIME)
as n_distinct_VISIT_START_TIME, count(distinct VISIT_TYPE_CONCEPT_ID) as n_distinct_VISIT_TYPE_CONCEPT_ID from VISIT_OCCURRENCE;"

sqlQuery(channel, qryText) %>% t
# > sqlQuery(channel, qryText) %>% t
#                                     1
# nrow                            47457
# n_distinct_CARE_SITE_ID         20713
# n_distinct_PERSON_ID              842
# n_distinct_PROVIDER_ID          34140
# n_distinct_VISIT_CONCEPT_ID         3
# n_distinct_VISIT_END_DATE        1096
# n_distinct_VISIT_END_TIME           1
# n_distinct_VISIT_OCCURRENCE_ID  47457
# n_distinct_VISIT_SOURCE_CONCEPT_ID      1
# n_distinct_VISIT_SOURCE_VALUE   47457
# n_distinct_VISIT_START_DATE      1096
# n_distinct_VISIT_START_TIME         1
# n_distinct_VISIT_TYPE_CONCEPT_ID    1
```

Rule-out non-candidate keys.. (not unique) in tidyverse/dplyr

```
VISIT_OCCURRENCE = sqlQuery(channel, "
select *
from VISIT_OCCURRENCE
;")
VISIT_OCCURRENCE %>% summarise_all(n_distinct) %>% t
#                           [,1]
# VISIT_OCCURRENCE_ID     47457
# PERSON_ID                 842
# VISIT_CONCEPT_ID            3
# VISIT_START_DATE         1096
# VISIT_START_TIME            1
# VISIT_END_DATE           1096
# VISIT_END_TIME              1
# VISIT_TYPE_CONCEPT_ID       1
# PROVIDER_ID             34140
# CARE_SITE_ID            20713
# VISIT_SOURCE_VALUE      47457
# VISIT_SOURCE_CONCEPT_ID     1
```

Refine the research question
Unit of observation (!)

Refine the research question
Unit of observation (!)

- E.g.) Which gender has more healthcare visits with depression diagnosis in our dataset?

Refine the research question
Unit of observation (!)

- ~~E.g.) Which gender has more healthcare visits with depression diagnosis in our dataset?~~
  - Among the individuals (people) who had one or more healthcare visit(s) with depression diagnosis in our dataset, which gender is more prevalent?
    - unit of observation = person
    - population for inference = population of people
    - (person-level statistics, individual-level statistics)
  - Among the healthcare visits with depression diagnosis in our dataset, which gender ~~is more prevalent~~ compose the larger proportion of the visits?
    - unit of observation = encounter
    - population for inference = population of encounters?
    - (encounter-level statistics? record-level statistics?)

Prevalence

- Point Prevalence
  - Point prevalence is the proportion of a population that has the condition at a specific point in time.
- Period Prevalence
  - Period prevalence is the proportion of a population that has the condition at some time during a given period (e.g., 12 month prevalence), and includes people who already have the condition at the start of the study period as well as those who acquire it during that period.
- Lifetime Prevalence
  - Lifetime prevalence is the proportion of a population that at some point in their life (up to the time of assessment) have experienced the condition.

International Classification of Diseases (ICD)

- Disease (& related health problem) classification system maintained by the World Health Organization (WHO), the directing and coordinating authority for health within the United Nations (UN).
- The Ninth Revision of the International Statistical Classification of Diseases, Injuries, and Causes of Death (ICD-9): since 1978
  - ~ 17,000 codes
- The Tenth revision of the International Statistical Classification of Diseases and Related Health Problems (ICD-10): since 1992
  - ~ 160,000 codes
  - In the U.S., ICD-10 codes are effective 10/2015 (Prior to that, ICD-9 was still used...)
- ICD-11 in development: initially planned for 2017, but pushed back..

ICD-9 codes for Major Depressive Disorder

- Non-specific code 296 Episodic mood disorders
  - Non-specific code 296.2 Major depressive disorder single episode
    - Specific code 296.20 Major depressive affective disorder, single episode, unspecified
    - Specific code 296.21 Major depressive affective disorder, single episode, mild
    - Specific code 296.22 Major depressive affective disorder, single episode, moderate
    - Specific code 296.23 Major depressive affective disorder, single episode, severe, without mention of psychotic behavior
    - Specific code 296.24 Major depressive affective disorder, single episode, severe, specified as with psychotic behavior
    - Specific code 296.25 Major depressive affective disorder, single episode, in partial or unspecified remission
    - Specific code 296.26 Major depressive affective disorder, single episode, in full remission
  - Non-specific code 296.3 Major depressive disorder recurrent episode
    - Specific code 296.30 Major depressive affective disorder, recurrent episode, unspecified
    - Specific code 296.31 Major depressive affective disorder, recurrent episode, mild
    - Specific code 296.32 Major depressive affective disorder, recurrent episode, moderate
    - Specific code 296.33 Major depressive affective disorder, recurrent episode, severe, without mention of psychotic behavior
    - Specific code 296.34 Major depressive affective disorder, recurrent episode, severe, specified as with psychotic behavior
    - Specific code 296.35 Major depressive affective disorder, recurrent episode, in partial or unspecified remission
    - Specific code 296.36 Major depressive affective disorder, recurrent episode, in full remission

* Data Quality check for ICD codes
OMOP_SYNTHETIC CONDITION_OCCURRENCE
group_by(CONDITION_SOURCE_VALUE)
n_distinct(CONDITION_OCCURRENCE_ID)

```
system.time(print(
sqlQuery(channel, paste("
select CONDITION_CONCEPT_ID, CONDITION_SOURCE_VALUE, count(*) as nrow, count(distinct CONDITION_OCCURRENCE_ID) as
CONDITION_OCCURRENCE_ID_n_distinct, count(distinct PERSON_ID) as PERSON_ID_n_distinct
from CONDITION_OCCURRENCE
where (CONDITION_SOURCE_VALUE like '2962%'
          or CONDITION_SOURCE_VALUE like '2963%'
)
group by CONDITION_CONCEPT_ID, CONDITION_SOURCE_VALUE
order by CONDITION_SOURCE_VALUE
;")) %>% as.tibble
))
#    CONDITION_CONCEPT_ID CONDITION_SOURCE_VALUE  nrow CONDITION_OCCURRENCE_ID_n_distinct PERSON_ID_n_distinct
#  *                <int>                 <int> <int>                              <int>                <int>
#  1               432284                 29620    51                                 51                   35
#  2               436945                 29621    14                                 14                   12
#  3               437837                 29622    27                                 27                   24
#  4               441534                 29623    25                                 25                   18
#  5               438406                 29624    10                                 10                    9
#  6               432284                 29625    13                                 13                   13
#  7               433750                 29626     7                                  7                    7
#  8               432285                 29630    60                                 60                   41
#  9               438998                 29631    27                                 27                   24
# 10               432883                 29632    85                                 85                   48
# 11               432883                 29633   118                                118                   53
# 12               434911                 29634    61                                 61                   31
# 13               440075                 29635    20                                 20                   10
# 14               440075                 29636    12                                 12                   12
#    user  system elapsed
#    0.13    0.00    0.24
```

* Data Quality check for ICD codes
OMOP_SYNTHETIC CONDITION_OCCURRENCE
group_by(PERSON_ID)
n_distinct(CONDITION_SOURCE_VALUE)

```
system.time(print(
sqlQuery(channel, paste("
select PERSON_ID, count(*) as nrow, count(distinct CONDITION_OCCURRENCE_ID) as n_distinct_CONDITION_OCCURRENCE_ID, count(distinct
PERSON_ID) as n_distinct_PERSON_ID, count(distinct CONDITION_SOURCE_VALUE) as n_distinct_CONDITION_SOURCE_VALUE
from CONDITION_OCCURRENCE
where (CONDITION_SOURCE_VALUE like '2962%'
          or CONDITION_SOURCE_VALUE like '2963%'
)
group by PERSON_ID
order by count(distinct CONDITION_SOURCE_VALUE) desc
;")) %>% as.tibble
))
# # A tibble: 184 x 5
#    PERSON_ID  nrow n_distinct_CONDITION_OCCURRENCE_ID n_distinct_PERSON_ID n_distinct_CONDITION_SOURCE_VALUE
#  *     <int> <int>                               <int>                <int>                              <int>
#  1       258    17                                  17                    1                                  9
#  2       144    11                                  11                    1                                  6
#  3       132     8                                   8                    1                                  6
#  4       167    33                                  33                    1                                  6
#  5       768     9                                   9                    1                                  6
#  6      1097     9                                   9                    1                                  5
#  7      1076    14                                  14                    1                                  5
#  8       758     5                                   5                    1                                  5
#  9       762     4                                   4                    1                                  4
# 10       598     6                                   6                    1                                  4
# # ... with 174 more rows
```

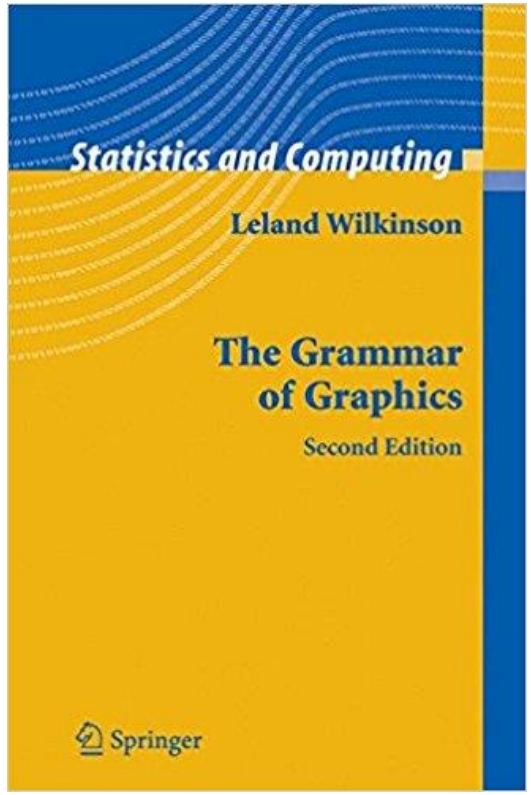
* Data Quality check for ICD codes
OMOP_SYNTHETIC CONDITION_OCCURRENCE
group_by(PERSON_ID)
n_distinct(CONDITION_SOURCE_VALUE)

```
system.time(print(
sqlQuery(channel, paste("
select CONDITION_OCCURRENCE_ID, VISIT_OCCURRENCE_ID, PERSON_ID, CONDITION_SOURCE_VALUE, CONDITION_START_DATE, CONDITION_END_DATE
from CONDITION_OCCURRENCE
where (CONDITION_SOURCE_VALUE like '2962%'
        or CONDITION_SOURCE_VALUE like '2963%'
) and PERSON_ID = 258
order by CONDITION_START_DATE
;")) %>% as.tibble
))
# > system.time(print(
# + sqlQuery(channel, paste("
# + select CONDITION_OCCURRENCE_ID, VISIT_OCCURRENCE_ID, PERSON_ID, CONDITION_SOURCE_VALUE, CONDITION_START_DATE, CONDITION_END_DATE
# + from CONDITION_OCCURRENCE
# + where (CONDITION_SOURCE_VALUE like '2962%'
# +         or CONDITION_SOURCE_VALUE like '2963%'
# + ) and PERSON_ID = 258
# + order by CONDITION_START_DATE
# + ;")) %>% as.tibble
# + ))
# # A tibble: 17 x 6
#    CONDITION_OCCURRENCE_ID VISIT_OCCURRENCE_ID PERSON_ID CONDITION_SOURCE_VALUE CONDITION_START_DATE CONDITION_END_DATE
#  *                   <int>               <int>     <int>                  <int>                <int>              <int>
#  1                  290742              107945       258                  29623             20080126           20080126
#  2                  254192               86392       258                  29624             20080223           20080223
#  3                    8500               86392       258                  29631             20080223           20080223
#  4                  563561               13095       258                  29625             20080601           20080601
#  5                  102598               13095       258                  29632             20080601           20080601
#  6                   21670               78816       258                  29634             20080607           20080607
#  7                  113158               99707       258                  29632             20080817           20080817
#  8                  249392               80973       258                  29624             20081006           20081006
#  9                   31750               18870       258                  29634             20090206           20090206
# 10                   21883               17434       258                  29633             20090405           20090414
# 11                   21456               17434       258                  29630             20090405           20090414
# 12                   25776               17434       258                  29630             20090405           20090414
# 13                   34656               17434       258                  29630             20090405           20090414
# 14                   39216               17434       258                  29630             20090405           20090414
# 15                  169722              123267       258                  29623             20090910           20090910
# 16                  213462              123267       258                  29623             20090910           20090910
# 17                  259780               49281       258                  29620             20100622           20100622
#    user  system elapsed
#    0.16    0.00    0.23
```

library(ggplot2)

The Grammar of Graphics

- Wilkinson L. The grammar of graphics. 2ed. Springer. 2006.



ELEMENT: *point*(*position*(birth*death), *size*(0), *label*(country))
ELEMENT: *contour*(*position*(
smooth.density.kernel.epanechnikov.joint(birth*death)),
color.hue())
GUIDE: *form.line*(*position*((0,0),(30,30)), *label*("*Zero Population Growth*"))
GUIDE: *axis*(*dim*(1), *label*("Birth Rate"))
GUIDE: *axis*(*dim*(2), *label*("Death Rate"))

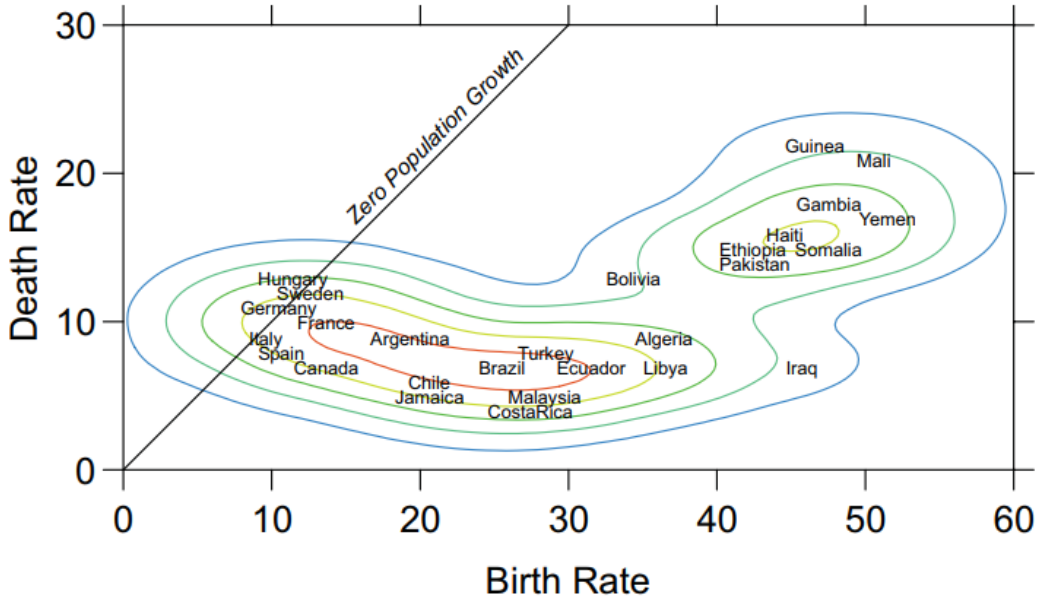


Figure 1.1 *Plot of death rates against birth rates for selected countries*

18.1.1.1 Syntax for Typical Graphs

For review, we now show examples of GPL programs for typical graphs.

scatterplot
ELEMENT: *point*(*position*(d*r))

line chart
ELEMENT: *line*(*position*(d*r))

bar chart
ELEMENT: *interval*(*position*(d*r))

horizontal bar chart
COORD: *rotate*(270)
ELEMENT: *interval*(*position*(d*r))

clustered bar chart
ELEMENT: *interval.dodge*(*position*(d*r), *color*(c))

stacked bar chart
ELEMENT: *interval.stack*(*position*(*summary.proportion*(r)), *color*(c))

stacked bars chart
ELEMENT: *interval.stack*(*position*(*summary.proportion*(d*r)), *color*(c))

pie chart
COORD: *polar.theta*(*dim*(1))
ELEMENT: *interval.stack*(*position*(*summary.proportion*(r)), *color*(c))

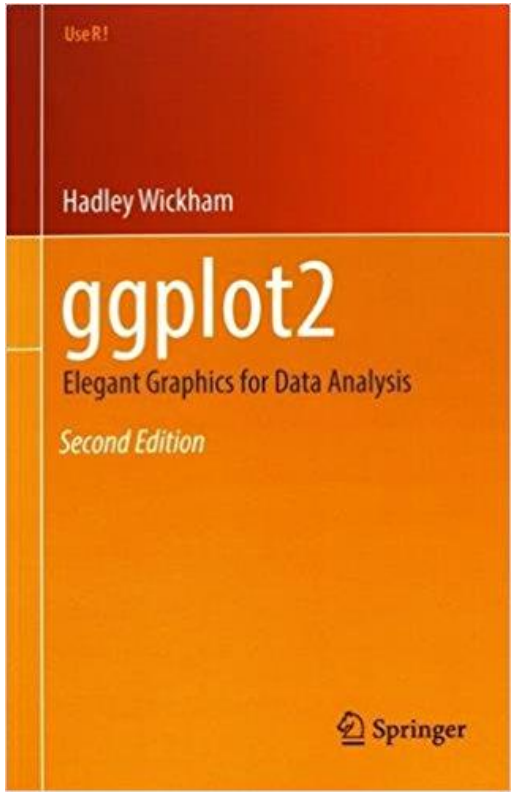
paneled pie charts
COORD: *rect*(*dim*(2), *polar.theta*(*dim*(1)))
ELEMENT: *interval.stack*(*position*(*summary.proportion*(d*r)), *color*(c))

map
ELEMENT: *polygon*(*position*(longitude*latitude))

choropleth map
ELEMENT: *polygon*(*position*(longitude*latitude, *split*(s)),
color.hue(*summary.mean*(y)))

library(ggplot2)

- Wickham H. ggplot2: elegant graphics for data analysis. Springer; 2016.
- https://github.com/hadley/ggplot2-book
- https://github.com/tidyverse/ggplot2



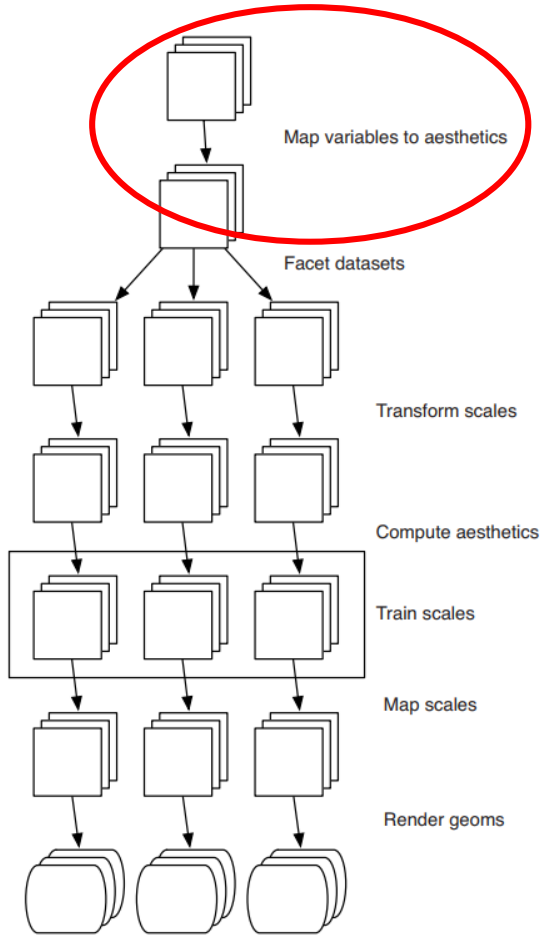


Fig. 3.7: Schematic description of the plot generation process. Each square represents a layer, and this schematic represents a plot with three layers and three panels. All steps work by transforming individual data frames, except for training scales which doesn't affect the data frame and operates across all datasets simultaneously.

ggplot2 cheatsheet

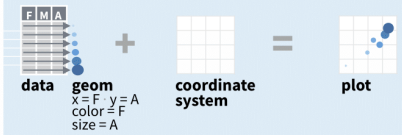
Rstudio Menu - Help - Cheatsheets

Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

required

```
ggplot (data = <DATA>) +
<GEOM_FUNCTION> (mapping = aes(<MAPPINGS>),
 stat = <STAT> , position = <POSITION> ) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

Not required, sensible defaults supplied

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings | data | geom

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1**)**
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group)**)**
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)**)** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)**)** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1)**)**
b + geom_hline(aes(yintercept = lat)**)**
b + geom_vline(aes(xintercept = long)**)**

b + geom_segment(aes(yend=lat+1, xend=long+1)**)**
b + geom_spoke(aes(angle = 1:1155, radius = 1)**)**

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)** x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)**)** x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(fl))
```

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)** x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2**)**
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)**, x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)**, x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)**, x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)**, x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**, x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

```
g <- ggplot(diamonds, aes(cut, color))
```

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
```

j + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

k + geom_map(aes(map_id = state), map = map**)**
+ expand_limits(x = map$long, y = map$lat**)**,
map_id, alpha, color, fill, linetype, size

THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z)**)**, hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z)**)**, x, y, alpha, color, fill, linetype, size, width

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **http://ggplot2.tidyverse.org** • ggplot2 2.1.0 • Updated: 2016-11

ggplot2 cheatsheet

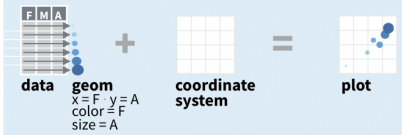
Rstudio Menu - Help - Cheatsheets

Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

required

```
ggplot (data = <DATA>) +
<GEOM_FUNCTION> (mapping = aes(<MAPPINGS>),
 stat = <STAT> , position = <POSITION> ) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

Not required, sensible defaults supplied

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings | data | geom

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1**)**
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group)**)**
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)**)** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)**)** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1)**)**
b + geom_hline(aes(yintercept = lat)**)**
b + geom_vline(aes(xintercept = long)**)**

b + geom_segment(aes(yend=lat+1, xend=long+1)**)**
b + geom_spoke(aes(angle = 1:1155, radius = 1)**)**

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)** x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)**)** x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(fl))
```

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)** x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2**)**
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)**, x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)**, x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)**, x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)**, x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**, x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

```
g <- ggplot(diamonds, aes(cut, color))
```

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
```

j + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

k + geom_map(aes(map_id = state), map = map**)**
+ expand_limits(x = map$long, y = map$lat**)**,
map_id, alpha, color, fill, linetype, size

THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z)**)**, hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z)**)**, x, y, alpha, color, fill, linetype, size, width

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **http://ggplot2.tidyverse.org** • ggplot2 2.1.0 • Updated: 2016-11

dplr cheatsheet

Rstudio Menu - Help - Cheatsheets

Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

&

Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries. Also **summarise_**().
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to created a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

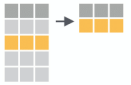
group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

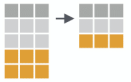
Manipulate Cases

EXTRACT CASES

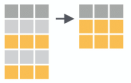
Row functions return a subset of rows as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.



filter(.data, …) Extract rows that meet logical criteria. Also **filter_()**. *filter(iris, Sepal.Length > 7)*

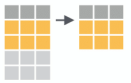


distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values. Also **distinct_()**.
distinct(iris, Species)



sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*



slice(.data, …) Select rows by position. Also **slice_()**. *slice(iris, 10:15)*

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

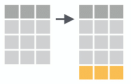
<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …)
Order rows by values of a column (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES



add_row(.data, ..., .before = NULL, .after = NULL)
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Column functions return a set of columns as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

select(.data, …)
Extract columns by name. Also **select_if()**
select(iris, Sepal.Length, Species)

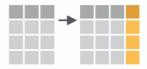
Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)	**num_range**(prefix, range)	**:**, e.g. mpg:cyl
ends_with(match)	**one_of**(…)	**-**, e.g, -Species
matches(match)	**starts_with**(match)	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function



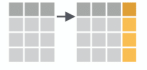
mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs()**.
mutate_all(faithful, funs(log(.), log2(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

mutate_if(.tbl, .predicate, .funs, …)
Apply funs to all columns of one type.
Use with **funs()**.
mutate_if(iris, is.numeric, funs(log(.)))



add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s).
add_column(mtcars, new = 1:32)

rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.5.0 • tibble 1.2.0 • Updated: 2017-01

dplr cheatsheet

Rstudio Menu - Help - Cheatsheets

Vectorized Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function →

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons

MISC

dplyr::**between()** - x >= left & x <= right
dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function →

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - mean absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also has_**rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(...) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join data. Retain only rows with matches.

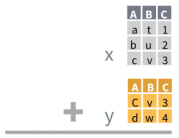
full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES



Use **bind_rows()** to paste tables below each other as they are.

bind_rows(..., .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

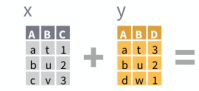
intersect(x, y, ...)
Rows that appear in both x and y.

setdiff(x, y, ...)
Rows that appear in x but not y.

union(x, y, ...)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS



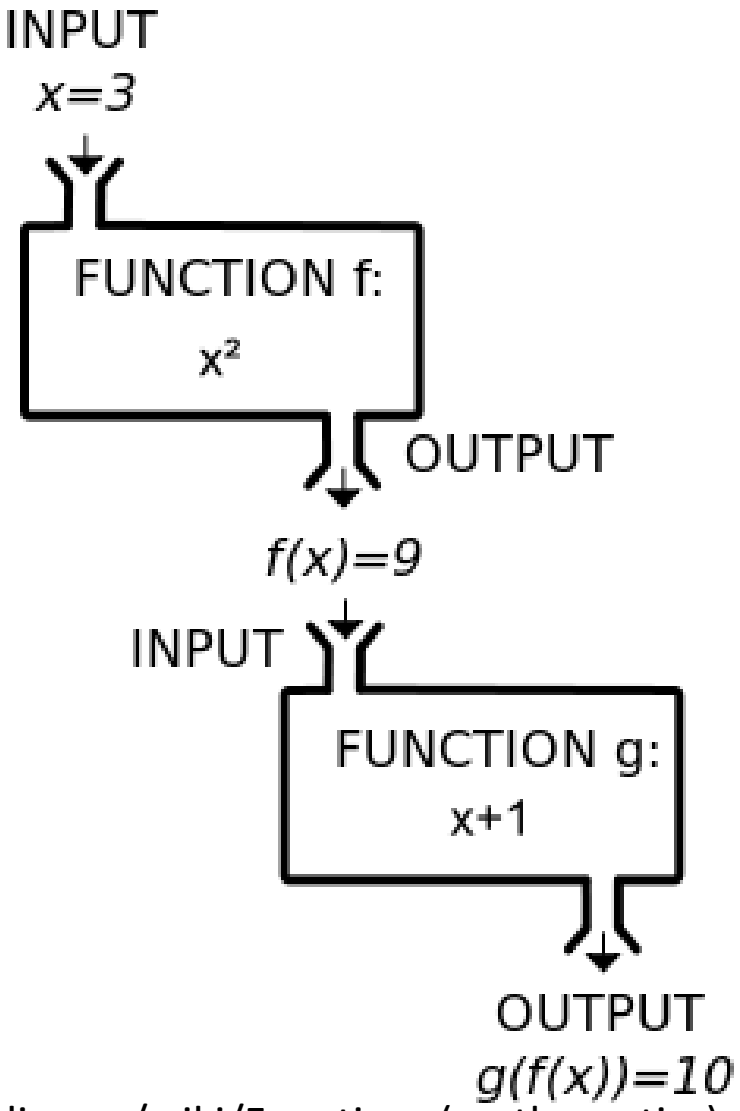
Use a "**Filtering Join**" to filter one table against the rows of another.

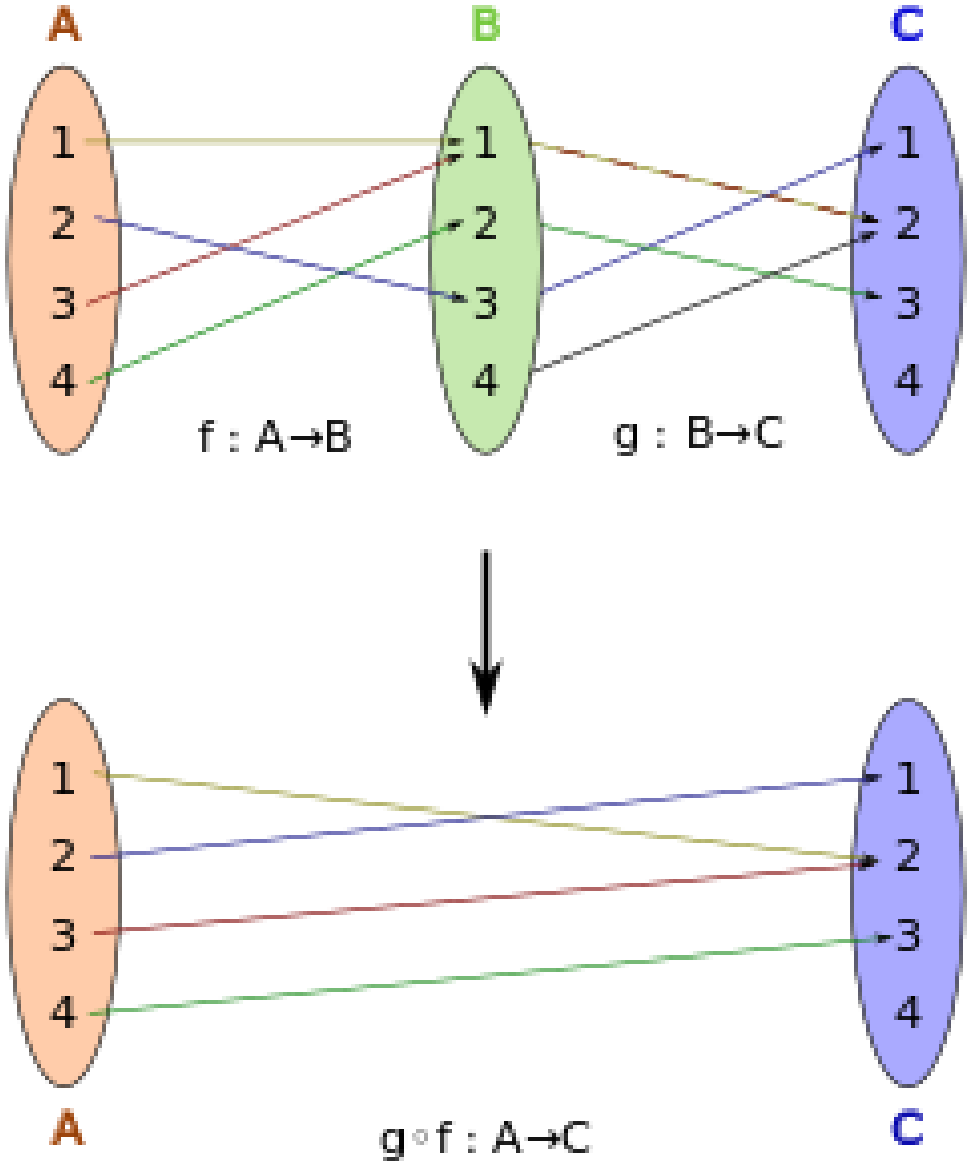
semi_join(x, y, by = NULL, ...)
Return rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...)
Return rows of x that do not have a match in y.
USEFUL TO SEE WHAT WILL NOT BE JOINED.

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.5.0 • tibble 1.2.0 • Updated: 2017-01

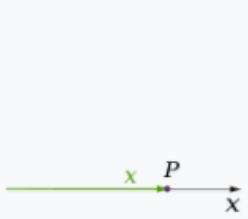
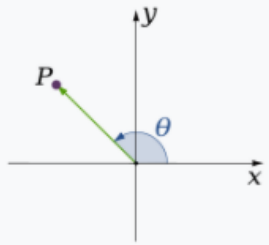
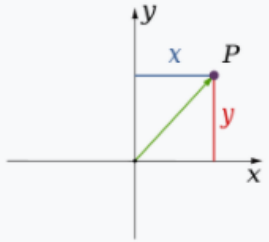
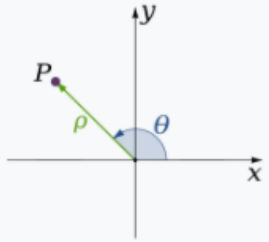
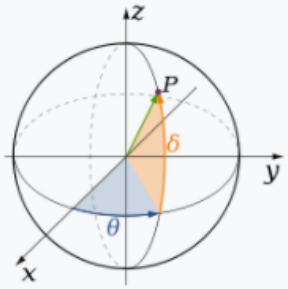
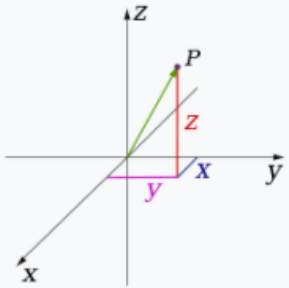
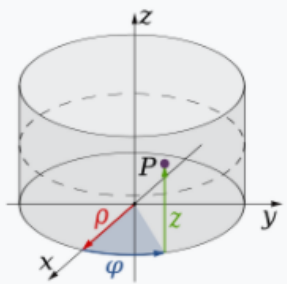
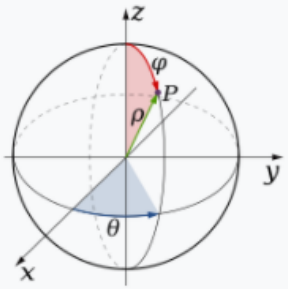
function
f:X→Y (*f* maps *X* into *Y*)





https://en.wikipedia.org/wiki/Function_(mathematics)

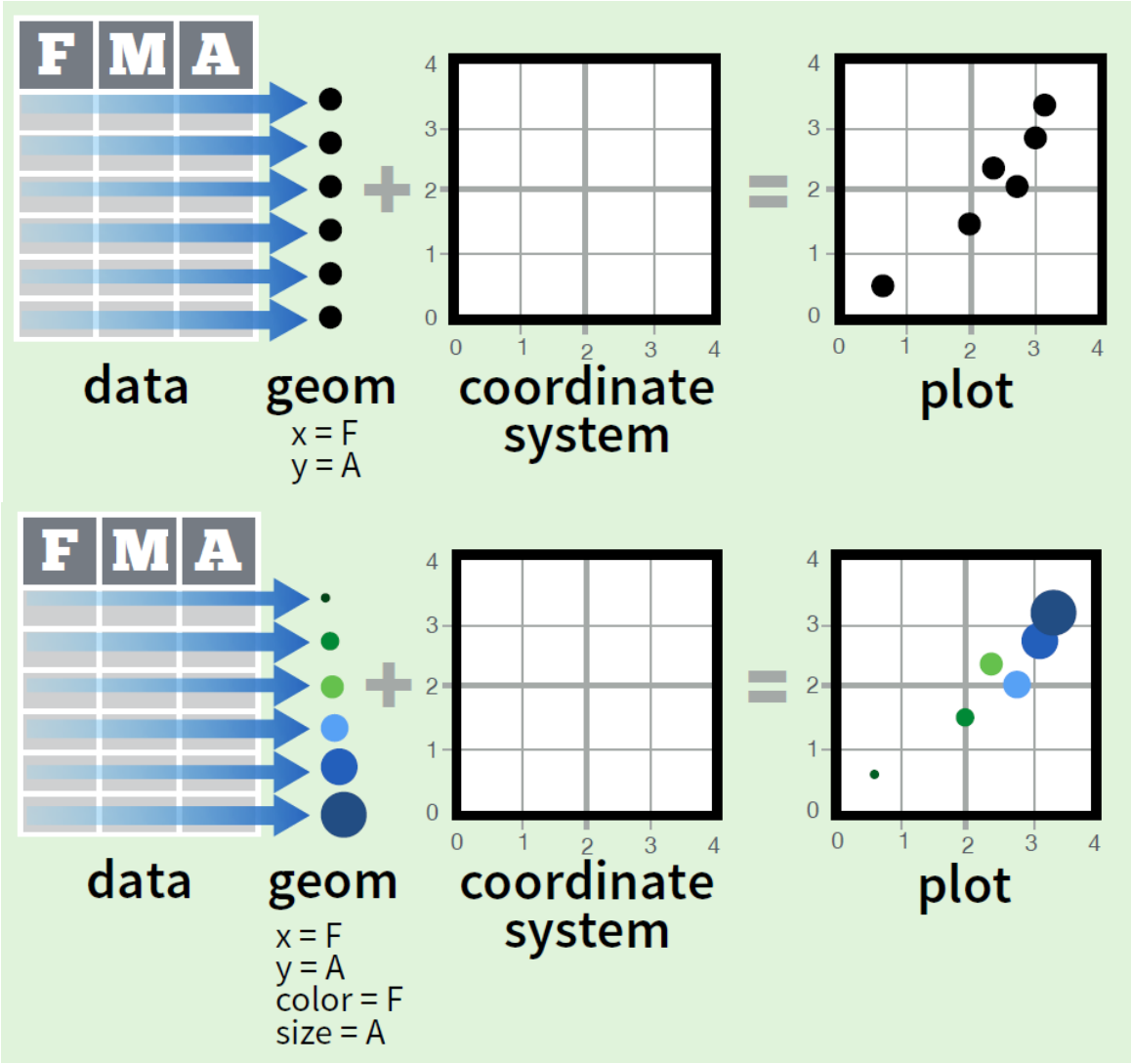
Dimension

Number of dimensions	Example co-ordinate systems
1	 Number line  Angle
2	 Cartesian (two-dimensional)  Polar  Latitude and longitude
3	 Cartesian (three-dimensional)  Cylindrical  Spherical

https://en.wikipedia.org/wiki/Dimension

ggplot2

- **ggplot2** is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data set**, a **coordinate system**, and **geoms**—visual marks that represent data points.
- To display values, **map variables** in the data to **visual properties of the geom (aesthetics)** like size, color, and x and y locations.



ggplot2 syntax

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
    mapping = aes(<MAPPINGS>),
    stat = <STAT>,
    position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

Required

Not required, sensible defaults supplied

Period Prevalence, by Year, by Gender

Independent variables = group_by(VISIT_START_YEAR, GENDER_SOURCE_VALUE)
Dependent variable = n_distinct(PERSON_ID)

How to convert text data type into datetime type in MS SQL Server

- https://docs.microsoft.com/en-us/sql/t-sql/functions/cast-and-convert-transact-sql
- -- Syntax for CONVERT:
- CONVERT (data_type [(length)] , expression [, style])

Standard	Input/Output ([3])
Default for datetime and small datetime	mon dd yyyy hh:miAM (or PM)
U.S.	1 = mm/dd/yy
	101 = mm/dd/yyyy
ANSI	2 = yy.mm.dd
	102 = yyyy.mm.dd
British/French	3 = dd/mm/yy
	103 = dd/mm/yyyy
German	4 = dd.mm.yy
	104 = dd.mm.yyyy
Italian	5 = dd-mm-yy
	105 = dd-mm-yyyy
-	6 = dd mon yy
	106 = dd mon yyyy
-	7 = Mon dd, yy
	107 = Mon dd, yyyy
USA	10 = mm-dd-yy
	110 = mm-dd-yyyy
JAPAN	11 = yy/mm/dd
	111 = yyyy/mm/dd
ISO	12 = yymmdd
	112 = yyyymmdd

Prevalence, by Year, by Gender
Independent variables = group_by(VISIT_START_YEAR, GENDER_SOURCE_VALUE)
Dependent variable = n_distinct(PERSON_ID)

```
#@ total & MDD & pMDD (PERSON_ID_n_distinct, 1-year period prevalence/proportion) ------
tblTotal = sqlQuery(channel, paste("
select year(convert(datetime, CONDITION_START_DATE, 112)) as CONDITION_START_year
           , GENDER_SOURCE_VALUE
           , count(*) as nrow
           , count(distinct CONDITION_OCCURRENCE_ID) as n_distinct_CONDITION_OCCURRENCE_ID
           , count(distinct CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID) as n_distinct_VISIT_OCCURRENCE_ID
           , count(distinct CONDITION_OCCURRENCE.PERSON_ID) as n_distinct_PERSON_ID
from (
           CONDITION_OCCURRENCE
           left join VISIT_OCCURRENCE
           on CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID = VISIT_OCCURRENCE.VISIT_OCCURRENCE_ID
           left join PERSON
           on VISIT_OCCURRENCE.PERSON_ID = PERSON.PERSON_ID
)
group by year(convert(datetime, CONDITION_START_DATE, 112)), GENDER_SOURCE_VALUE
order by year(convert(datetime, CONDITION_START_DATE, 112)), GENDER_SOURCE_VALUE
;"))


tblMDD = sqlQuery(channel, paste("
select year(convert(datetime, CONDITION_START_DATE, 112)) as CONDITION_START_year
           , GENDER_SOURCE_VALUE
           , count(*) as nrow
           , count(distinct CONDITION_OCCURRENCE_ID) as n_distinct_CONDITION_OCCURRENCE_ID
           , count(distinct CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID) as n_distinct_VISIT_OCCURRENCE_ID
           , count(distinct CONDITION_OCCURRENCE.PERSON_ID) as n_distinct_PERSON_ID
from (
           CONDITION_OCCURRENCE
           left join VISIT_OCCURRENCE
           on CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID = VISIT_OCCURRENCE.VISIT_OCCURRENCE_ID
           left join PERSON
           on VISIT_OCCURRENCE.PERSON_ID = PERSON.PERSON_ID
)
where (CONDITION_SOURCE_VALUE like '2962%'
           or CONDITION_SOURCE_VALUE like '2963%'
)
group by year(convert(datetime, CONDITION_START_DATE, 112)), GENDER_SOURCE_VALUE
order by year(convert(datetime, CONDITION_START_DATE, 112)), GENDER_SOURCE_VALUE
;"))
```

Prevalence, by Year, by Gender
Independent variables = group_by(VISIT_START_YEAR, GENDER_SOURCE_VALUE)
Dependent variable = n_distinct(PERSON_ID)

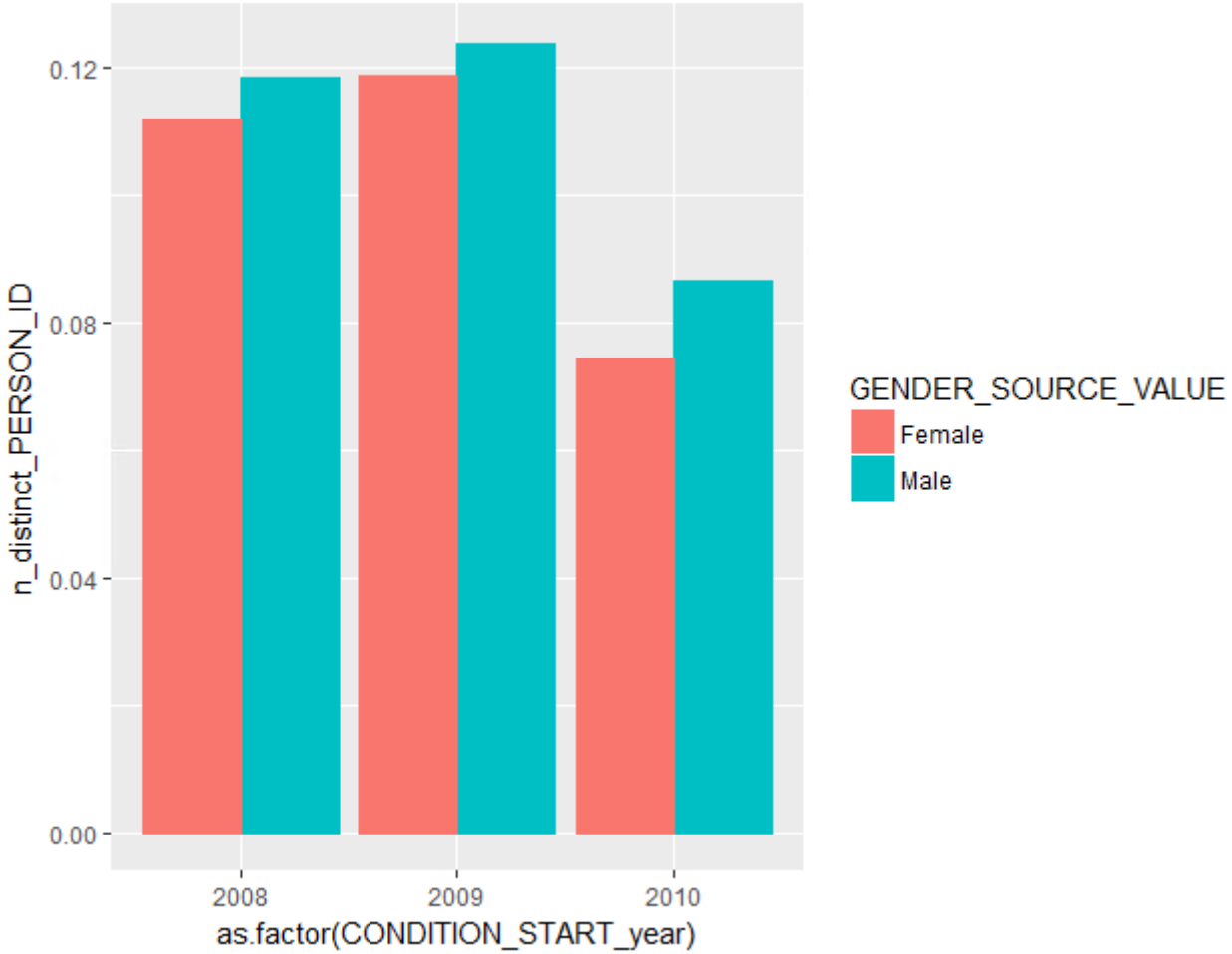
```
tblTotal
tblMDD = full_join(select(tblTotal, -nrow, -n_distinct_CONDITION_OCCURRENCE_ID, -n_distinct_VISIT_OCCURRENCE_ID, -n_distinct_PERSON_ID), tblMDD)
tblMDD
tblMDDp = tblMDD; tblMDDp[,c("nrow", "n_distinct_CONDITION_OCCURRENCE_ID", "n_distinct_VISIT_OCCURRENCE_ID", "n_distinct_PERSON_ID")] = tblMDD[,c("nrow",
"n_distinct_CONDITION_OCCURRENCE_ID", "n_distinct_VISIT_OCCURRENCE_ID", "n_distinct_PERSON_ID")]/tblTotal[,c("nrow", "n_distinct_CONDITION_OCCURRENCE_ID",
"n_distinct_VISIT_OCCURRENCE_ID", "n_distinct_PERSON_ID")]
tblMDDp
# > tblTotal
#   CONDITION_START_year GENDER_SOURCE_VALUE  nrow n_distinct_CONDITION_OCCURRENCE_ID n_distinct_VISIT_OCCURRENCE_ID n_distinct_PERSON_ID
# 1                 2008              Female 30327                            30327                            8439                 375
# 2                 2008                Male 27379                            27379                            7667                 346
# 3                 2009              Female 31871                            31871                            8917                 404
# 4                 2009                Male 31302                            31302                            8703                 387
# 5                 2010              Female 20231                            20231                            5509                 390
# 6                 2010                Male 19212                            19212                            5291                 369
# > tblMDD
#   CONDITION_START_year GENDER_SOURCE_VALUE nrow n_distinct_CONDITION_OCCURRENCE_ID n_distinct_VISIT_OCCURRENCE_ID n_distinct_PERSON_ID
# 1                 2008              Female   90                              90                              65                  42
# 2                 2008                Male  100                             100                              56                  41
# 3                 2009              Female  103                             103                              68                  48
# 4                 2009                Male  132                             132                              75                  48
# 5                 2010              Female   56                              56                              34                  29
# 6                 2010                Male   49                              49                              35                  32
# > tblMDDp = tblMDD; tblMDDp[,c("nrow", "n_distinct_CONDITION_OCCURRENCE_ID", "n_distinct_VISIT_OCCURRENCE_ID", "n_distinct_PERSON_ID")] = tblMDD[,c("nrow",
"n_distinct_CONDITION_OCCURRENCE_ID", "n_distinct_VISIT_OCCURRENCE_ID", "n_distinct_PERSON_ID")]/tblTotal[,c("nrow", "n_distinct_CONDITION_OCCURRENCE_ID",
"n_distinct_VISIT_OCCURRENCE_ID", "n_distinct_PERSON_ID")]
# > tblMDDp
#   CONDITION_START_year GENDER_SOURCE_VALUE        nrow n_distinct_CONDITION_OCCURRENCE_ID n_distinct_VISIT_OCCURRENCE_ID n_distinct_PERSON_ID
# 1                 2008              Female 0.002967653                       0.002967653                      0.007702334          0.11200000
# 2                 2008                Male 0.003652434                       0.003652434                      0.007304030          0.11849711
# 3                 2009              Female 0.003231778                       0.003231778                      0.007625883          0.11881188
# 4                 2009                Male 0.004216983                       0.004216983                      0.008617718          0.12403101
# 5                 2010              Female 0.002768029                       0.002768029                      0.006171719          0.07435897
# 6                 2010                Male 0.002550489                       0.002550489                      0.006615007          0.08672087
```

Time series, Prevalence
Independent variables = group_by(VISIT_START_YEAR, VISIT_START_MONTH, GENDER_SOURCE_VALUE)
Dependent variable = n_distinct(PERSON_ID)

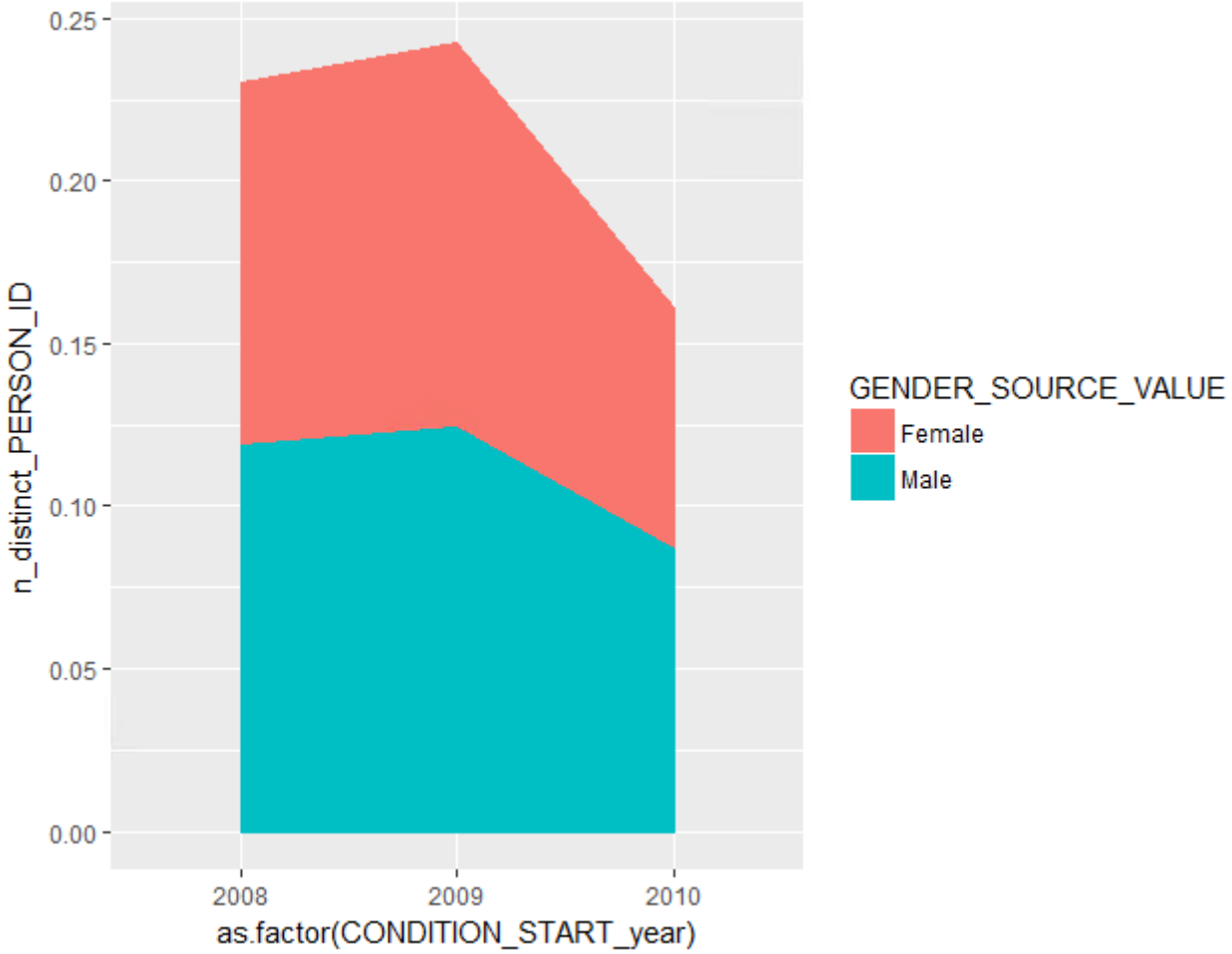
```
g = tblMDDp %>% ggplot(aes(x = as.factor(CONDITION_START_year), y = n_distinct_PERSON_ID, group = GENDER_SOURCE_VALUE, color = GENDER_SOURCE_VALUE, fill = GENDER_SOURCE_VALUE))

g+geom_area()
g+geom_area(position = "stack")
g+geom_area(position = "fill")
g+geom_col()
g+geom_col(position = "stack")
g+geom_col(position = "dodge")
g+geom_col(position = "fill")
```

ggplot(aes(x = as.factor(CONDITION_START_month), y = n_distinct_PERSON_ID, group = GENDER_SOURCE_VALUE, color = GENDER_SOURCE_VALUE, fill = GENDER_SOURCE_VALUE))+facet_grid(CONDITION_START_year~.)

g+geom_col(position = "dodge")



g+geom_area(position = "stack")



Period Prevalence, by Month, by Gender

Independent variables = group_by(VISIT_START_YEAR, VISIT_START_MONTH, GENDER_SOURCE_VALUE)
Dependent variable = n_distinct(PERSON_ID)

Prevalence, by Month, by Gender

Independent variables = group_by(VISIT_START_YEAR, VISIT_START_MONTH, GENDER_SOURCE_VALUE)

Dependent variable = n_distinct(PERSON_ID)

```
#@ total & MDD & pMDD (n_distinct_PERSON_ID, 1-month period prevalence/proportion) ------
system.time((
tblTotal = sqlQuery(channel, paste("
select year(convert(datetime, CONDITION_START_DATE, 112)) as CONDITION_START_year
            , month(convert(datetime, CONDITION_START_DATE, 112)) as CONDITION_START_month
            , GENDER_SOURCE_VALUE
            , count(*) as nrow
            , count(distinct CONDITION_OCCURRENCE_ID) as n_distinct_CONDITION_OCCURRENCE_ID
            , count(distinct CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID) as n_distinct_VISIT_OCCURRENCE_ID
            , count(distinct CONDITION_OCCURRENCE.PERSON_ID) as n_distinct_PERSON_ID
from (
            CONDITION_OCCURRENCE
            left join VISIT_OCCURRENCE
            on CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID = VISIT_OCCURRENCE.VISIT_OCCURRENCE_ID
            left join PERSON
            on VISIT_OCCURRENCE.PERSON_ID = PERSON.PERSON_ID
)
group by year(convert(datetime, CONDITION_START_DATE, 112)), month(convert(datetime, CONDITION_START_DATE, 112)), GENDER_SOURCE_VALUE
order by year(convert(datetime, CONDITION_START_DATE, 112)), month(convert(datetime, CONDITION_START_DATE, 112)), GENDER_SOURCE_VALUE
;"))
))

system.time((
tblMDD = sqlQuery(channel, paste("
select year(convert(datetime, CONDITION_START_DATE, 112)) as CONDITION_START_year
            , month(convert(datetime, CONDITION_START_DATE, 112)) as CONDITION_START_month
            , GENDER_SOURCE_VALUE
            , count(*) as nrow
            , count(distinct CONDITION_OCCURRENCE_ID) as n_distinct_CONDITION_OCCURRENCE_ID
            , count(distinct CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID) as n_distinct_VISIT_OCCURRENCE_ID
            , count(distinct CONDITION_OCCURRENCE.PERSON_ID) as n_distinct_PERSON_ID
from (
            CONDITION_OCCURRENCE
            left join VISIT_OCCURRENCE
            on CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID = VISIT_OCCURRENCE.VISIT_OCCURRENCE_ID
            left join PERSON
            on VISIT_OCCURRENCE.PERSON_ID = PERSON.PERSON_ID
)
where (CONDITION_SOURCE_VALUE like '2962%'
            or CONDITION_SOURCE_VALUE like '2963%'
)
group by year(convert(datetime, CONDITION_START_DATE, 112)), month(convert(datetime, CONDITION_START_DATE, 112)), GENDER_SOURCE_VALUE
order by year(convert(datetime, CONDITION_START_DATE, 112)), month(convert(datetime, CONDITION_START_DATE, 112)), GENDER_SOURCE_VALUE
;"))
))
```

Prevalence, by Month, by Gender

Independent variables = group_by(VISIT_START_YEAR, VISIT_START_MONTH, GENDER_SOURCE_VALUE)

Dependent variable = n_distinct(PERSON_ID)

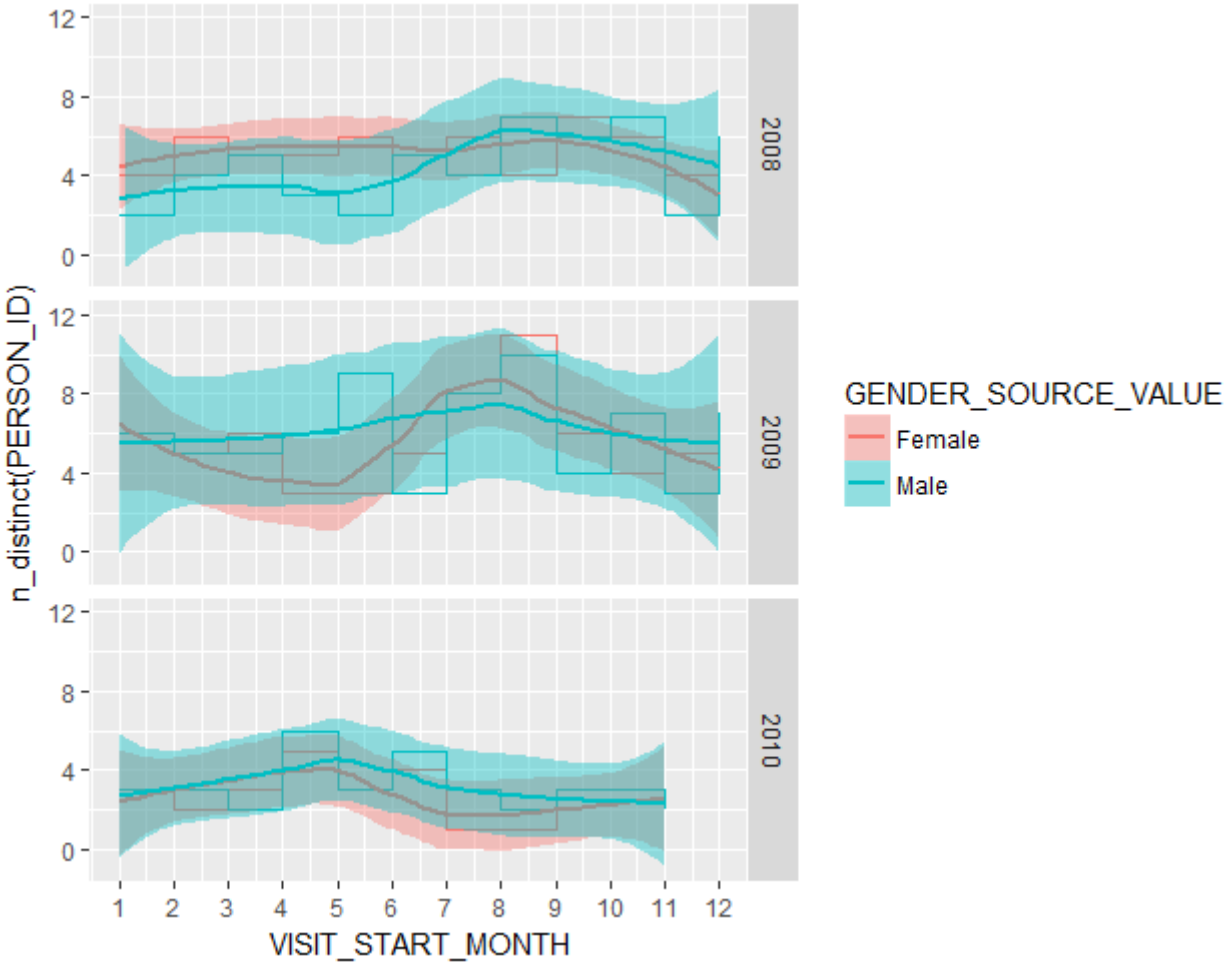
```
tblTotal
tblMDD = full_join(select(tblTotal, -nrow, -n_distinct_CONDITION_OCCURRENCE_ID, -n_distinct_VISIT_OCCURRENCE_ID, -n_distinct_PERSON_ID), tblMDD)
tblMDD
tblMDDp = tblMDD; tblMDDp[,c("nrow", "n_distinct_CONDITION_OCCURRENCE_ID", "n_distinct_VISIT_OCCURRENCE_ID", "n_distinct_PERSON_ID")] = tblMDD[,c("nrow",
"n_distinct_CONDITION_OCCURRENCE_ID", "n_distinct_VISIT_OCCURRENCE_ID", "n_distinct_PERSON_ID")]/tblTotal[,c("nrow", "n_distinct_CONDITION_OCCURRENCE_ID",
"n_distinct_VISIT_OCCURRENCE_ID", "n_distinct_PERSON_ID")]
tblMDDp
# > tblMDDp
#    CONDITION_START_year CONDITION_START_month GENDER_SOURCE_VALUE         nrow n_distinct_CONDITION_OCCURRENCE_ID n_distinct_VISIT_OCCURRENCE_ID n_distinct_PERSON_ID
# 1                  2008                     1               Female 0.0029962547                      0.0029962547                    0.011299435          0.025157233
# 2                  2008                     1                 Male 0.0037771483                      0.0037771483                    0.006944444          0.017241379
# 3                  2008                     2               Female 0.0073839662                      0.0073839662                    0.013157895          0.028846154
# 4                  2008                     2                 Male 0.0092535472                      0.0092535472                    0.011160714          0.023255814
# 5                  2008                     3               Female 0.0019654088                      0.0019654088                    0.007342144          0.019455253
# 6                  2008                     3                 Male 0.0031180401                      0.0031180401                    0.008319468          0.022935780
# 7                  2008                     4               Female 0.0018698579                      0.0018698579                    0.006693440          0.018382353
# 8                  2008                     4                 Male 0.0013297872                      0.0013297872                    0.004680187          0.012875536
# 9                  2008                     5               Female 0.0032549729                      0.0032549729                    0.009138381          0.021739130
# 10                 2008                     5                 Male 0.0012864494                      0.0012864494                    0.003044140          0.008264463
# 11                 2008                     6               Female 0.0045801527                      0.0045801527                    0.009370817          0.018939394
# 12                 2008                     6                 Male 0.0020576132                      0.0020576132                    0.007052186          0.019531250
# 13                 2008                     7               Female 0.0024973243                      0.0024973243                    0.007853403          0.021352313
# 14                 2008                     7                 Male 0.0019290123                      0.0019290123                    0.005494505          0.015094340
# 15                 2008                     8               Female 0.0014326648                      0.0014326648                    0.005044136          0.014134276
# 16                 2008                     8                 Male 0.0079064971                      0.0079064971                    0.008641975          0.025830258
# 17                 2008                     9               Female 0.0033259424                      0.0033259424                    0.009210526          0.023809524
# 18                 2008                     9                 Male 0.0024916944                      0.0024916944                    0.008915305          0.023255814
# 19                 2008                    10               Female 0.0034495975                      0.0034495975                    0.008010681          0.021428571
# 20                 2008                    10                 Male 0.0069049553                      0.0069049553                    0.012931034          0.026515152
# 21                 2008                    11               Female 0.0033234860                      0.0033234860                    0.005361930          0.014440433
# 22                 2008                    11                 Male 0.0023464998                      0.0023464998                    0.002801120          0.008032129
# 23                 2008                    12               Female 0.0010434783                      0.0010434783                    0.003750000          0.010563380
# 24                 2008                    12                 Male 0.0023923445                      0.0023923445                    0.008547009          0.023904382
# 25                 2009                     1               Female 0.0029784066                      0.0029784066                    0.007792208          0.020000000
# 26                 2009                     1                 Male 0.0028756290                      0.0028756290                    0.007884363          0.021897810
```

Time series, Prevalence
Independent variables = group_by(VISIT_START_YEAR, VISIT_START_MONTH, GENDER_SOURCE_VALUE)
Dependent variable = n_distinct(PERSON_ID)

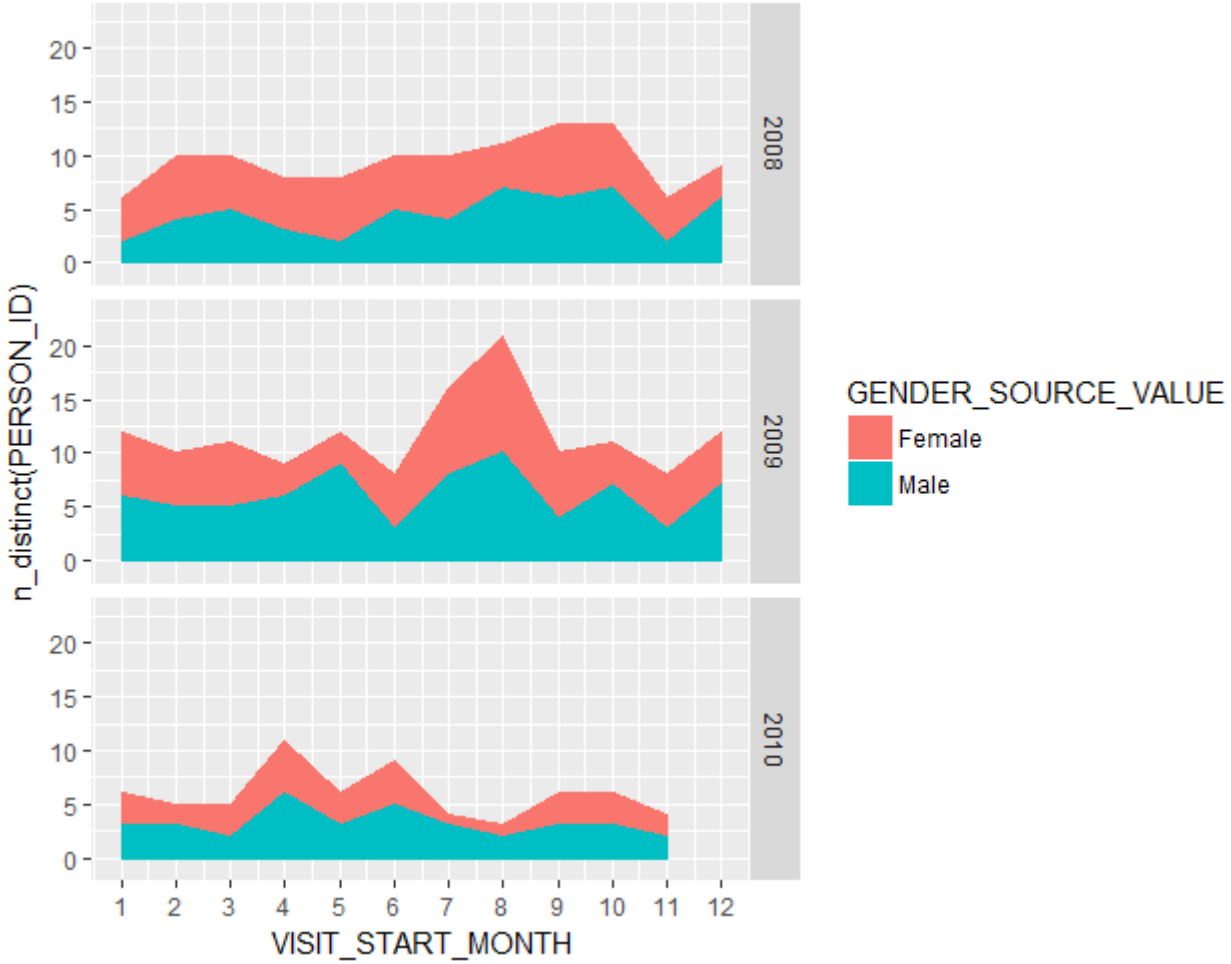
```
g = tblMDDp %>%
    ggplot(aes(
        x = as.factor(CONDITION_START_month)
        , y = n_distinct_PERSON_ID
        , group = GENDER_SOURCE_VALUE
        , color = GENDER_SOURCE_VALUE
        , fill = GENDER_SOURCE_VALUE)) +
    facet_grid(CONDITION_START_year~.)

g+geom_step()
g+geom_step()+geom_smooth(method = "loess")
g+geom_step()+geom_smooth(method = "lm")
g+geom_line()
g+geom_line()+geom_smooth(method = "loess")
g+geom_line()+geom_smooth(method = "lm")
g+geom_area()
```

ggplot(aes(x = as.factor(CONDITION_START_month), y = n_distinct_PERSON_ID, group = GENDER_SOURCE_VALUE, color = GENDER_SOURCE_VALUE, fill = GENDER_SOURCE_VALUE))+facet_grid(CONDITION_START_year~.)

g+geom_line()+geom_smooth(method = "loess")



g+geom_area()



Period Prevalence, by Year, by State

Independent variables = group_by(VISIT_START_YEAR, STATE)
Dependent variable = n_distinct(PERSON_ID)

Prevalence, by Year, by State
Independent variables = group_by(VISIT_START_YEAR, STATE)
Dependent variable = n_distinct(PERSON_ID)

```
#@ total & MDD & pMDD (n_distinct_PERSON_ID, 1-year period prevalence/proportion, by state) ------
system.time((
tblTotal = sqlQuery(channel, paste("
select year(convert(datetime, CONDITION_START_DATE, 112)) as CONDITION_START_year
            , STATE
            , count(*) as nrow
            , count(distinct CONDITION_OCCURRENCE_ID) as n_distinct_CONDITION_OCCURRENCE_ID
            , count(distinct CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID) as n_distinct_VISIT_OCCURRENCE_ID
            , count(distinct CONDITION_OCCURRENCE.PERSON_ID) as n_distinct_PERSON_ID
from (
            CONDITION_OCCURRENCE
            left join VISIT_OCCURRENCE
            on CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID = VISIT_OCCURRENCE.VISIT_OCCURRENCE_ID
            left join PERSON
            on VISIT_OCCURRENCE.PERSON_ID = PERSON.PERSON_ID
            left join LOCATION
            on PERSON.LOCATION_ID = LOCATION.LOCATION_ID
)
group by year(convert(datetime, CONDITION_START_DATE, 112)), STATE
order by year(convert(datetime, CONDITION_START_DATE, 112)), STATE
;")) %>% as.tibble
))

system.time((
tblMDD = sqlQuery(channel, paste("
select year(convert(datetime, CONDITION_START_DATE, 112)) as CONDITION_START_year
            , STATE
            , count(*) as nrow
            , count(distinct CONDITION_OCCURRENCE_ID) as n_distinct_CONDITION_OCCURRENCE_ID
            , count(distinct CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID) as n_distinct_VISIT_OCCURRENCE_ID
            , count(distinct CONDITION_OCCURRENCE.PERSON_ID) as n_distinct_PERSON_ID
from (
            CONDITION_OCCURRENCE
            left join VISIT_OCCURRENCE
            on CONDITION_OCCURRENCE.VISIT_OCCURRENCE_ID = VISIT_OCCURRENCE.VISIT_OCCURRENCE_ID
            left join PERSON
            on VISIT_OCCURRENCE.PERSON_ID = PERSON.PERSON_ID
            left join LOCATION
            on PERSON.LOCATION_ID = LOCATION.LOCATION_ID
)
where (CONDITION_SOURCE_VALUE like '2962%'
            or CONDITION_SOURCE_VALUE like '2963%'
)
group by year(convert(datetime, CONDITION_START_DATE, 112)), STATE
order by year(convert(datetime, CONDITION_START_DATE, 112)), STATE
;")) %>% as.tibble
))
```

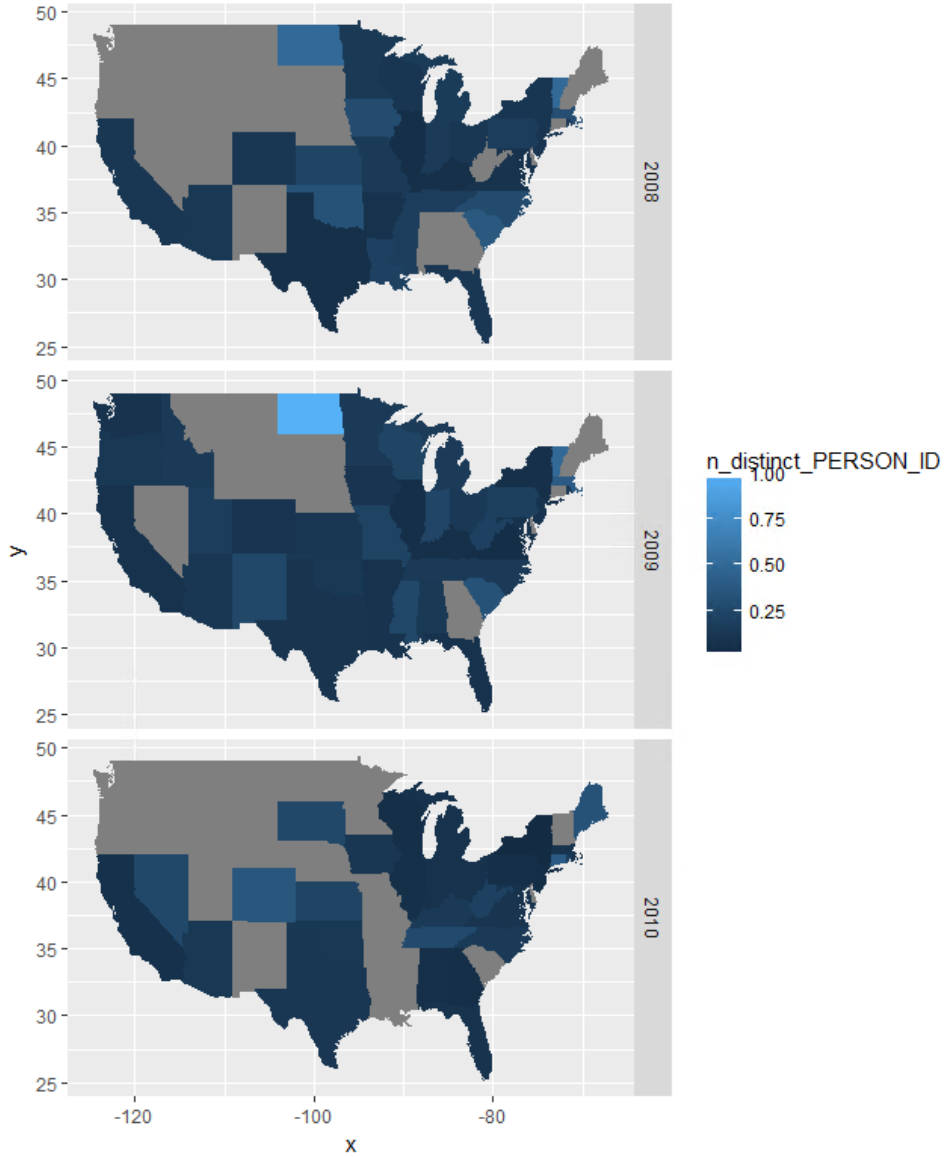
Prevalence, by Year, by State
Independent variables = group_by(VISIT_START_YEAR, STATE)
Dependent variable = n_distinct(PERSON_ID)

```
tblTotal
tblMDD = full_join(select(tblTotal, -nrow, -n_distinct_CONDITION_OCCURRENCE_ID, -n_distinct_VISIT_OCCURRENCE_ID, -n_distinct_PERSON_ID), tblMDD)
tblMDD
tblMDDp = tblMDD; tblMDDp[,c("nrow", "n_distinct_CONDITION_OCCURRENCE_ID", "n_distinct_VISIT_OCCURRENCE_ID", "n_distinct_PERSON_ID")] = tblMDD[,c("nrow",
"n_distinct_CONDITION_OCCURRENCE_ID", "n_distinct_VISIT_OCCURRENCE_ID", "n_distinct_PERSON_ID")]/tblTotal[,c("nrow", "n_distinct_CONDITION_OCCURRENCE_ID",
"n_distinct_VISIT_OCCURRENCE_ID", "n_distinct_PERSON_ID")]
tblMDDp


tblMDDp %>% full_join(., tibble(state.abb, state.name, state.name.tolower = tolower(state.name)), by = c("STATE" = "state.abb"))
# > tblMDDp %>% full_join(., tibble(state.abb, state.name, state.name.tolower = tolower(state.name)), by = c("STATE" = "state.abb"))
# # A tibble: 156 x 8
#    CONDITION_START_year STATE     nrow n_distinct_CONDITION_OCCURRENCE_ID n_distinct_VISIT_OCCURRENCE_ID n_distinct_PERSON_ID state.name
state.name.tolower
#                   <int> <chr>    <dbl>                              <dbl>                           <dbl>                <dbl> <chr>
<chr>
#  1                 2008 ""     NA                                  NA                              NA                      NA      NA
NA
#  2                 2008 AK     NA                                  NA                              NA                      NA      Alaska
alaska
#  3                 2008 AL     NA                                  NA                              NA                      NA      Alabama
alabama
#  4                 2008 AR      0.00110                             0.00110                         0.00412                 0.0833 Arkansas
arkansas
#  5                 2008 AZ      0.00696                             0.00696                         0.00495                 0.125  Arizona
arizona
#  6                 2008 CA      0.00764                             0.00764                         0.0124                  0.111  California
california
#  7                 2008 CO      0.00177                             0.00177                         0.00629                 0.111  Colorado
colorado
#  8                 2008 CT     NA                                  NA                              NA                      NA      Connecticut
connecticut
#  9                 2008 DC     NA                                  NA                              NA                      NA      NA
NA
# 10                 2008 DE     NA                                  NA                              NA                      NA      Delaware
delaware
# # ... with 146 more rows
```

Prevalence, by Year, by State
Independent variables = group_by(VISIT_START_YEAR, STATE)
Dependent variable = n_distinct(PERSON_ID)

```
library(ggplot2)
library(maps)
library(maps)
map = map_data("state")
g = tblMDDp %>% full_join(., tibble(state.abb, state.name, state.name.tolower =
tolower(state.name)), by = c("STATE" = "state.abb")) %>%
    ggplot(aes(fill = n_distinct_PERSON_ID)) +
    facet_grid(CONDITION_START_year~.)
g+geom_map(aes(map_id = state.name.tolower), map = map)+expand_limits(x=map$long, y=map$lat)

detach("package:maps", unload=TRUE)
```

```
g = tblMDDp %>% full_join(., tibble(state.abb, state.name, state.name.tolower = tolower(state.name)), by = c("STATE" = "state.abb")) %>%
    ggplot(aes(fill = n_distinct_PERSON_ID)) +
    facet_grid(CONDITION_START_year~.)
g+geom_map(aes(map_id = state.name.tolower), map = map)+expand_limits(x=map$long, y=map$lat)
```



library(jsonlite)

Read from JSON file

```
library(jsonlite)
CMS_SynPUF_CDMv5_YZ.json = fromJSON("CMS_SynPUF_CDMv5_YZ.json")
```

JSON tree structure -> list in R

```
CMS_SynPUF_CDMv5_YZ.json %>% length
CMS_SynPUF_CDMv5_YZ.json[1000] %>% str
# > CMS_SynPUF_CDMv5_YZ.json %>% length
# [1] 1000
# > CMS_SynPUF_CDMv5_YZ.json[1000] %>% str
# List of 1
#  $ 999:List of 5
#   ..$ DOB : int 1940
#   ..$ appt:List of 7
#   .. ..$ 20090501:List of 5
#   .. .. ..$ actualdate: chr "20090501"
#   .. .. ..$ diag      : chr [1:2] "462" "46421"
#   .. .. ..$ drug      : list()
#   .. .. ..$ proc      : chr "71020"
#   .. .. ..$ type      : chr "887623388569362"
#   .. ..$ 20090503:List of 5
#   .. .. ..$ actualdate: chr "20090503"
#   .. .. ..$ diag      : chr [1:3] "1731" "1731" "2329"
#   .. .. ..$ drug      : list()
#   .. .. ..$ proc      : chr [1:2] "73130" "88305"
#   .. .. ..$ type      : chr "887683385170808"
#   .. ..$ 20090607:List of 5
#   .. .. ..$ actualdate: chr "20090607"
#   .. .. ..$ diag      : chr [1:4] "1101" "1101" "1171" "7098"
#   .. .. ..$ drug      : list()
#   .. .. ..$ proc      : chr [1:2] "99212" "99212"
#   .. .. ..$ type      : chr "887733386054944"
#   .. ..$ 20090724:List of 5
#   .. .. ..$ actualdate: chr "20090724"
#   .. .. ..$ diag      : chr "4270"
#   .. .. ..$ drug      : list()
#   .. .. ..$ proc      : chr "36415"
#   .. .. ..$ type      : chr "542122281130114"
#   .. ..$ 20100207:List of 5
#   .. .. ..$ actualdate: chr "20100207"
#   .. .. ..$ diag      : chr [1:2] "3540" "3569"
#   .. .. ..$ drug      : list()
#   .. .. ..$ proc      : chr "99222"
#   .. .. ..$ type      : chr "887713388554440"
#   .. ..$ 20100410:List of 5
#   .. .. ..$ actualdate: chr "20100410"
#   .. .. ..$ diag      : chr [1:4] "4011" "73300" "73301" "73301"
#   .. .. ..$ drug      : list()
#   .. .. ..$ proc      : chr "77080"
#   .. .. ..$ type      : chr "887473385353685"
#   .. ..$ 20100905:List of 5
#   .. .. ..$ actualdate: chr "20100905"
#   .. .. ..$ diag      : chr [1:2] "7237" "72402"
#   .. .. ..$ drug      : list()
#   .. .. ..$ proc      : chr "62311"
#   .. .. ..$ type      : chr "887163385548265"
#   ..$ lab :List of 1
#   .. ..$ 20100207:List of 1
#   .. .. ..$ Audiogram abnormal: chr "255"
#   ..$ race: chr "White"
#   ..$ sex : chr "Male"
```

list -> map into a data_frame

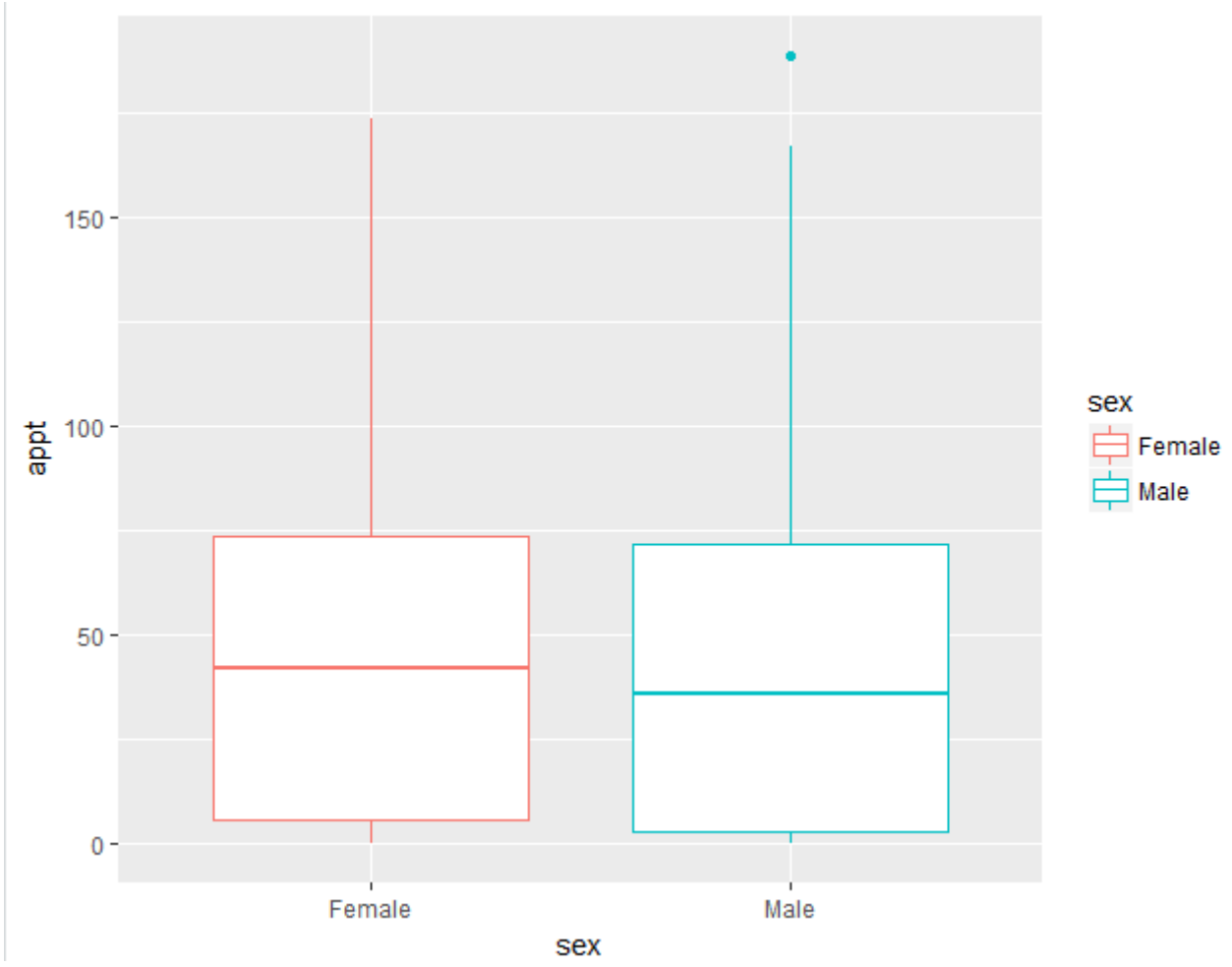
```
#@ CMS_SynPUF_CDMv5_YZ.json.map_df = CMS_SynPUF_CDMv5_YZ.json %>% purrr::map_df(function(ls)
{ ----
CMS_SynPUF_CDMv5_YZ.json.map_df = CMS_SynPUF_CDMv5_YZ.json %>% purrr::map_df(function(ls) {
    ls %>% purrr::map(function(x) {
        if (is.list(x))
            x = length(x)
        if (length(x) > 1)
            x = length(x)
        x
    })
})
CMS_SynPUF_CDMv5_YZ.json.map_df
# # A tibble: 1,000 x 5
#      DOB  appt   lab race   sex
#    <int> <int> <int> <chr>  <chr>
#  1  1932    65     3 white  Male
#  2  1957     8     1 Others Female
#  3  1942     2     1 white  Female
#  4  1939    15     1 white  Female
#  5  1928     0     0 white  Female
#  6  1935     2     0 white  Female
#  7  1943     2     0 white  Female
#  8  1924     1     0 white  Male
#  9  1939    22     0 white  Female
# 10  1935   108     6 white  Female
# # ... with 990 more rows
```

list -> map into a data_frame -> ggplot

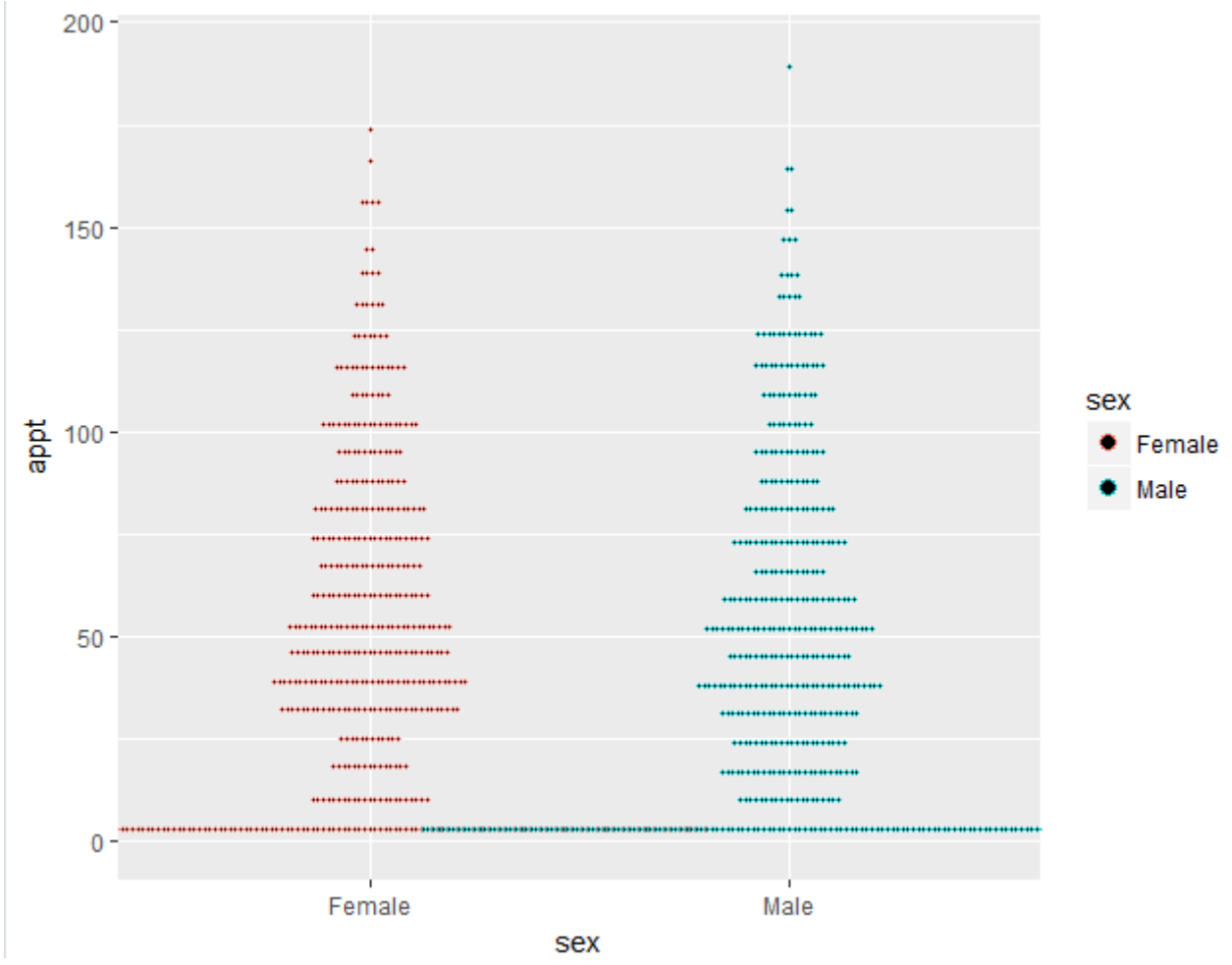
```
g = CMS_SynPUF_CDMv5_YZ.json.map_df %>%
    ggplot(aes(x = sex, y = appt, color = sex))

g+geom_boxplot()
g+geom_dotplot(binaxis = "y", stackdir = "center", dotsize = 1/5)
g+geom_point()
g+geom_count()
g+geom_jitter()
g+geom_violin()
```

ggplot(aes(x = sex, y = appt, color = sex))

g+geom_boxplot()

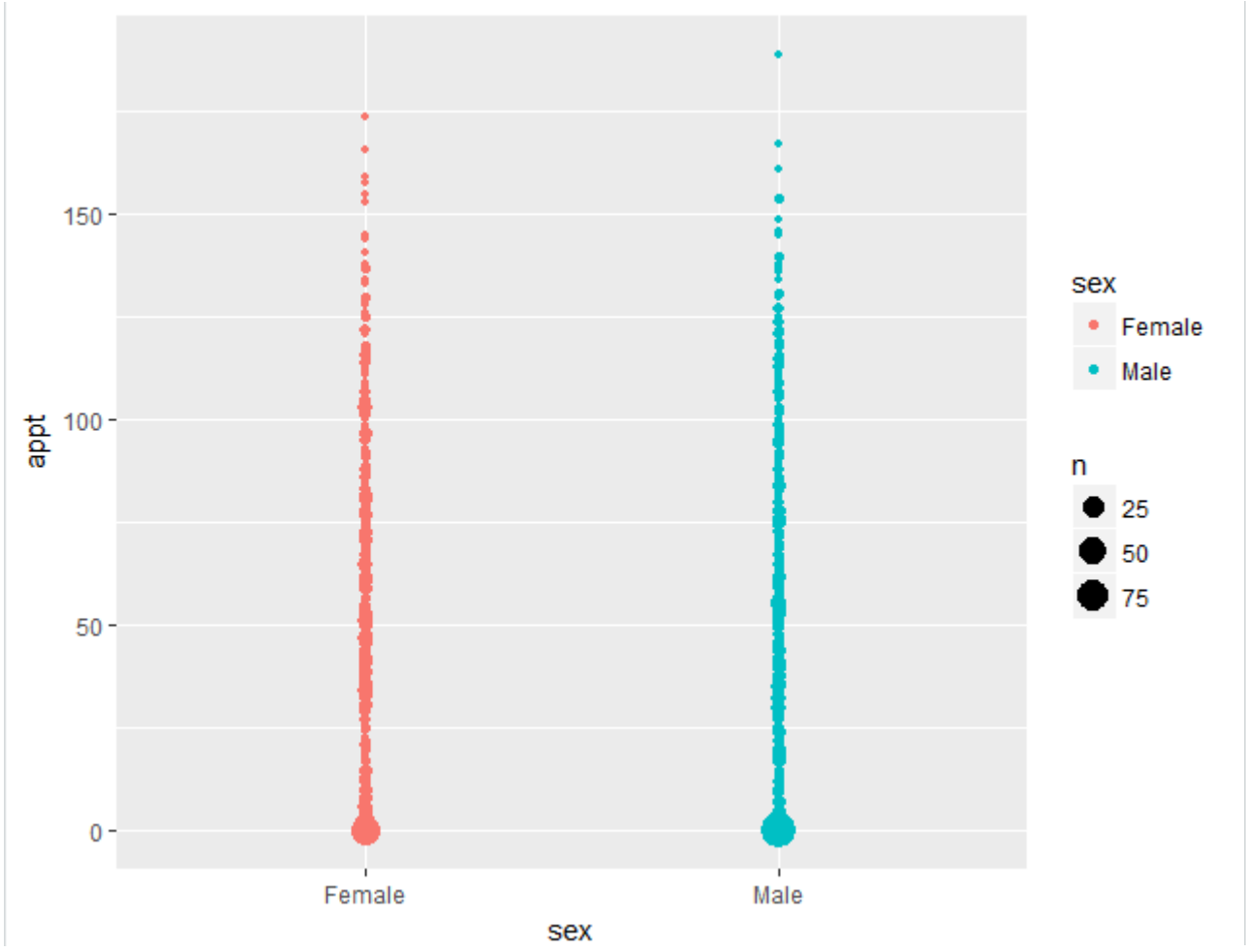


g+geom_dotplot(binaxis = "y", stackdir = "center")

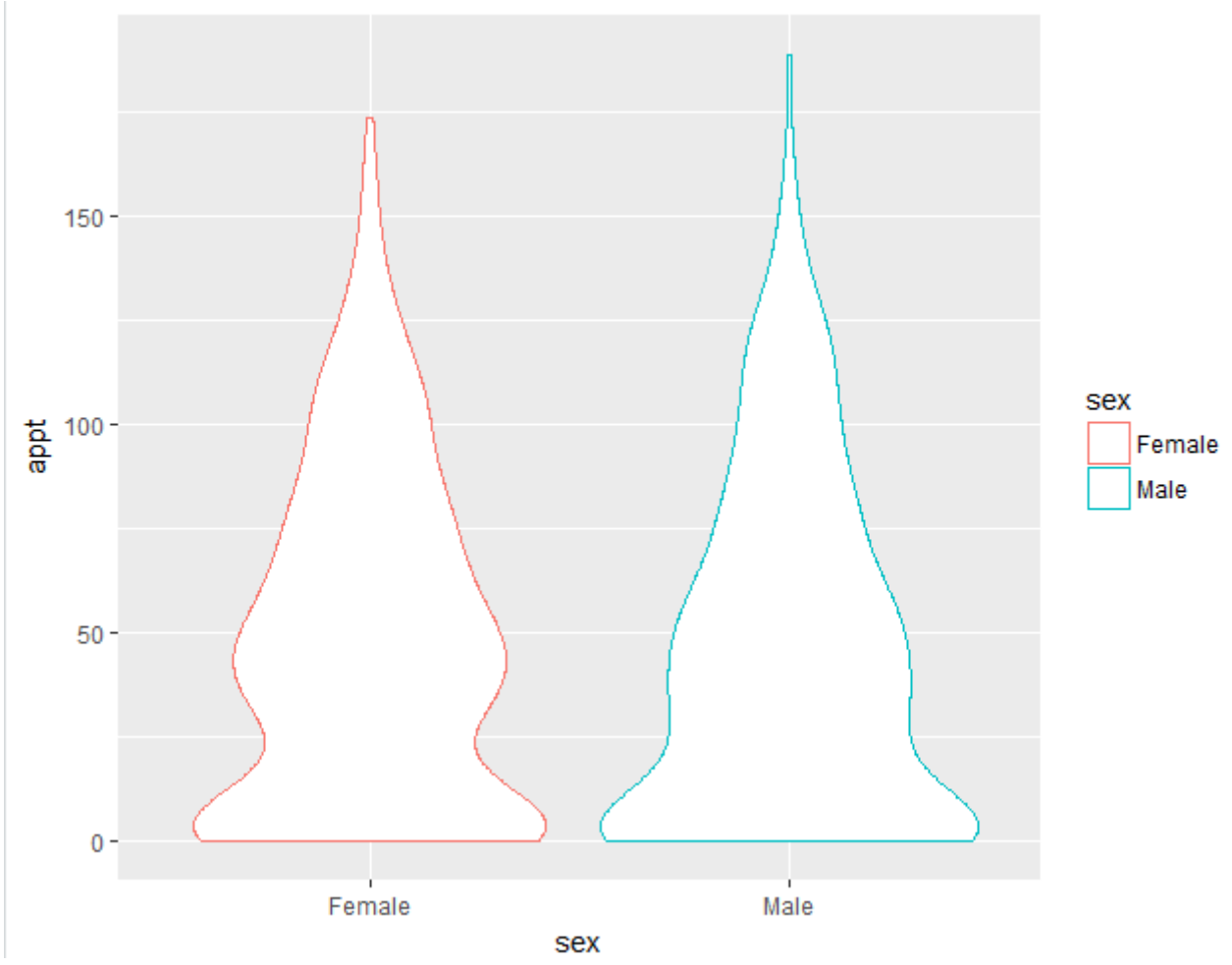


ggplot(aes(x = sex, y = appt, color = sex))

g+geom_count()

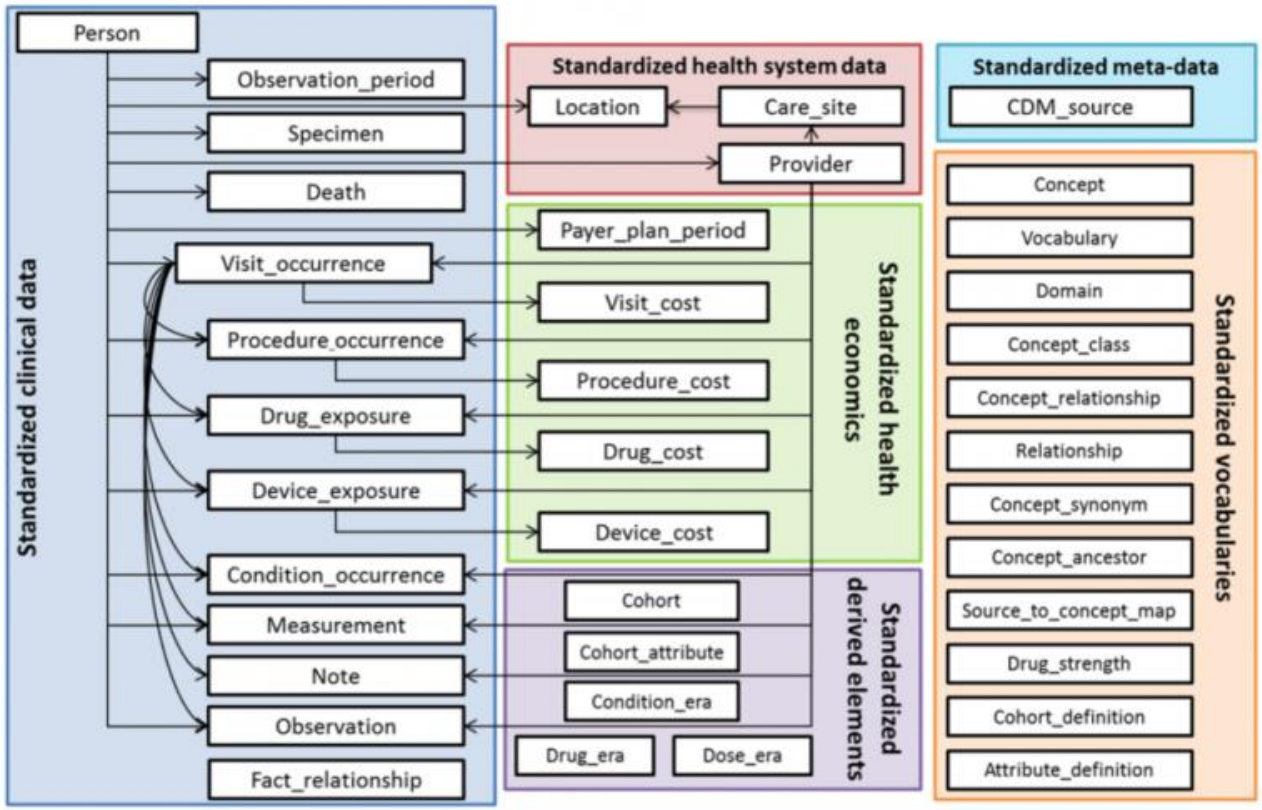


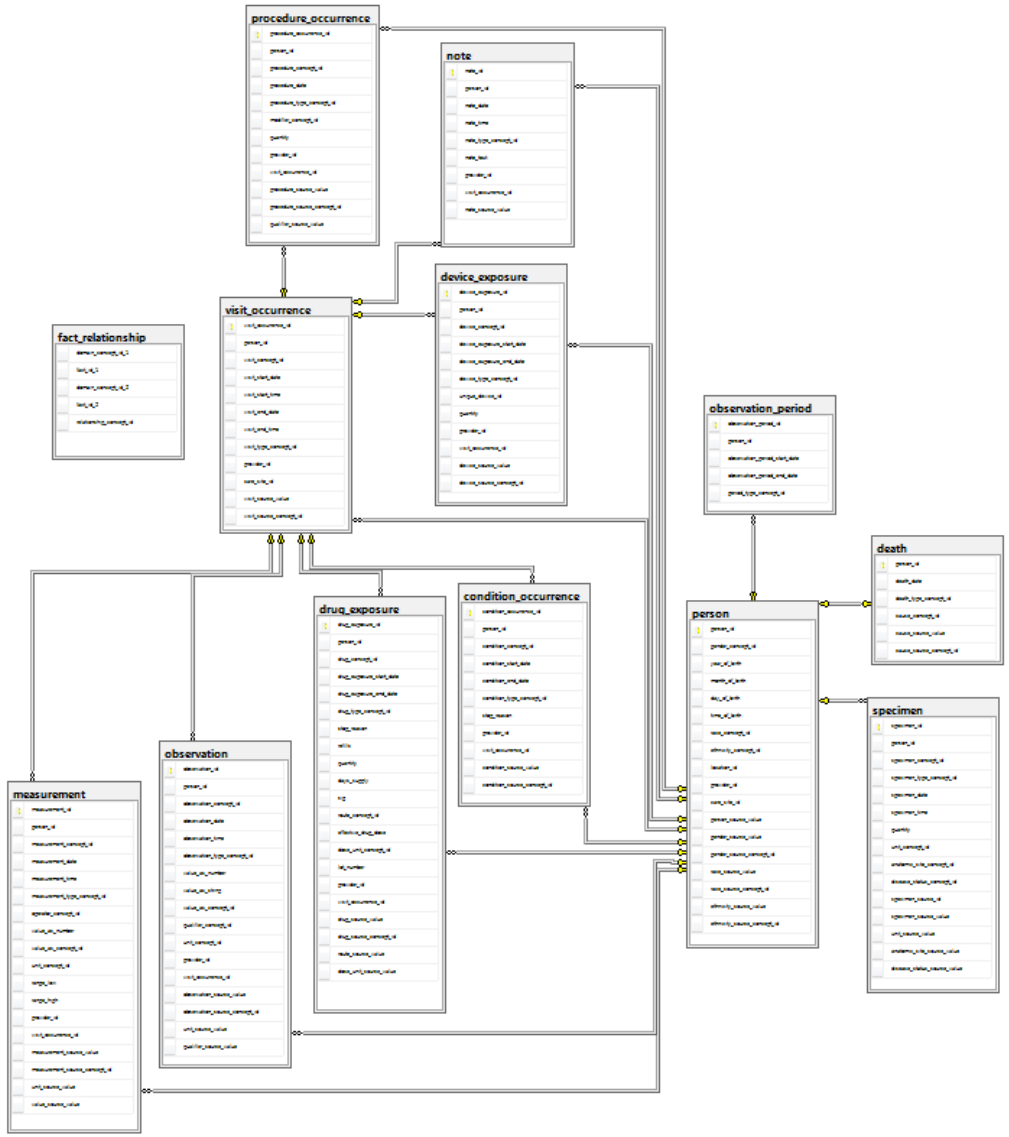
g+geom_violin()



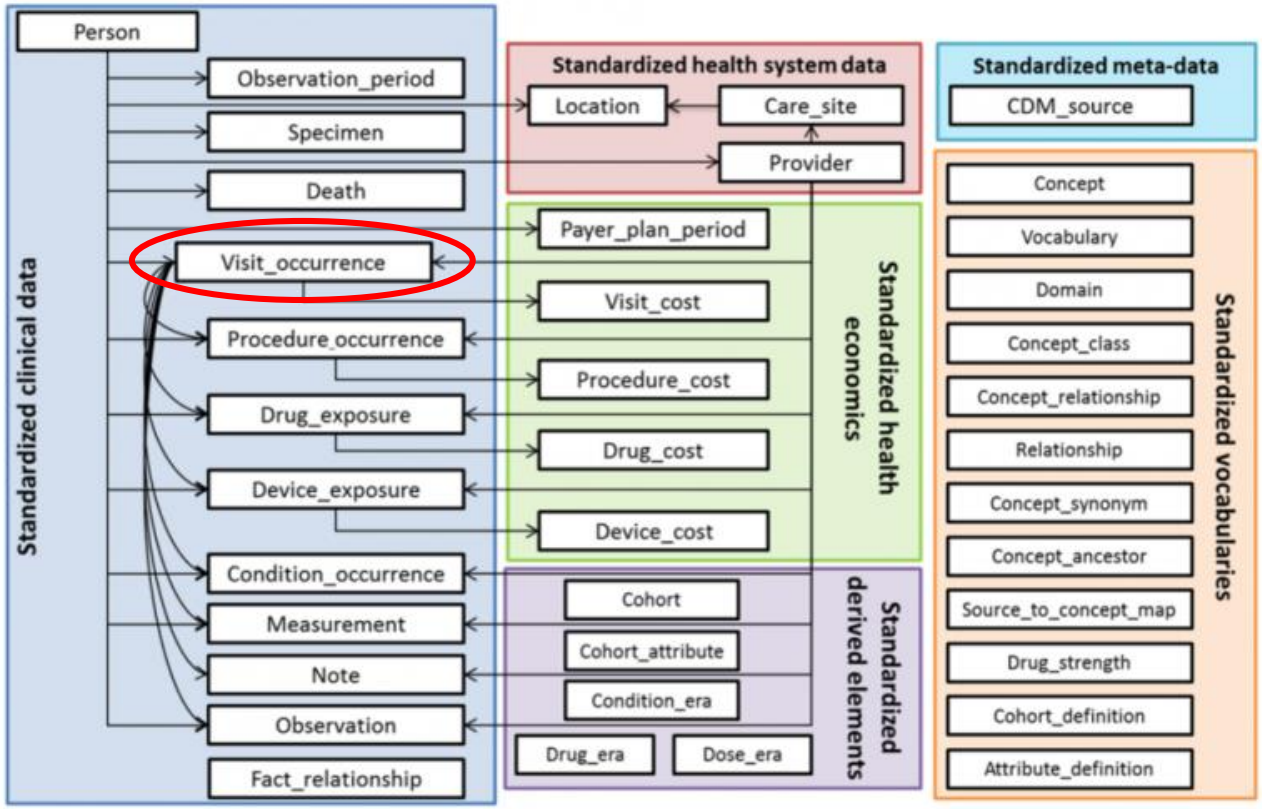
Common Data Model

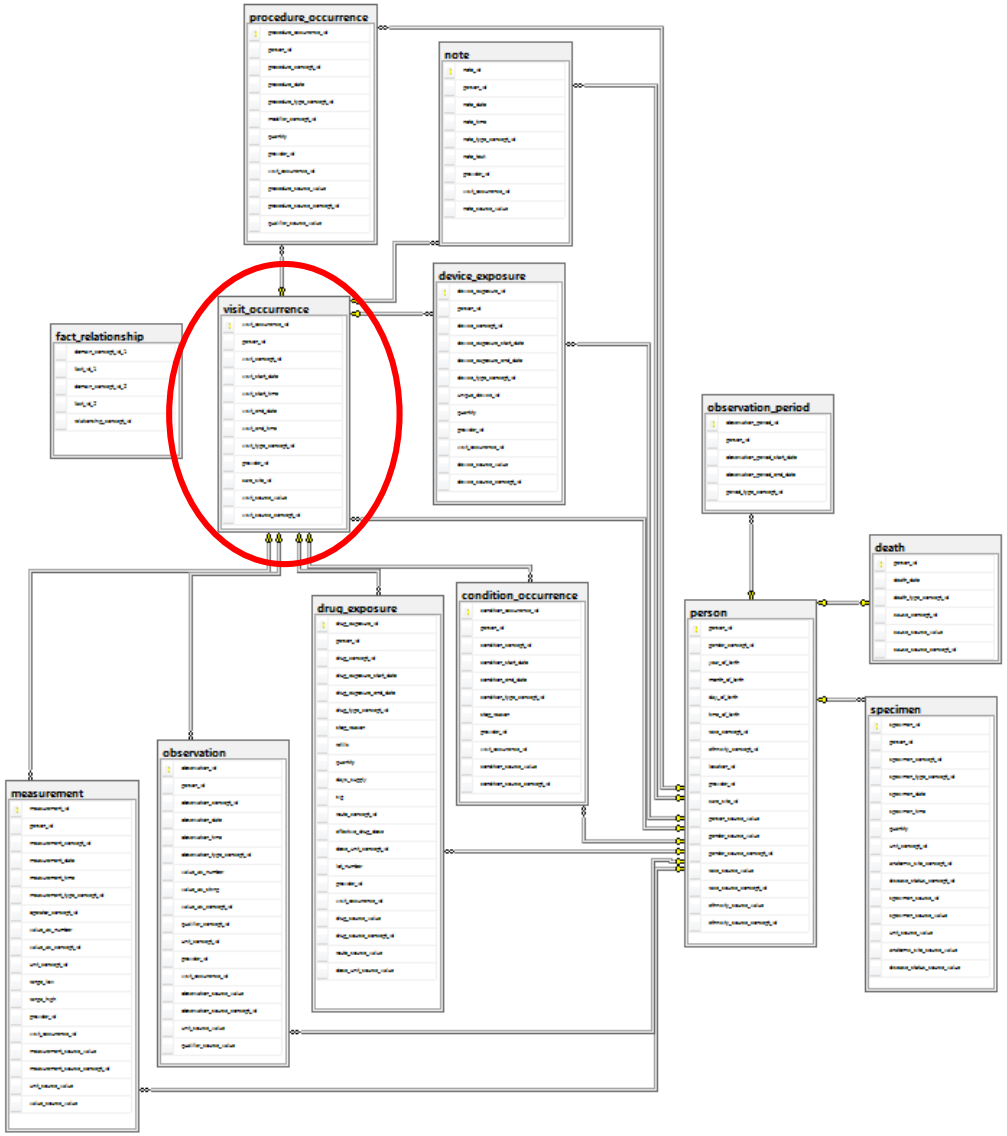
https://github.com/OHDSI/CommonDataModel/wiki



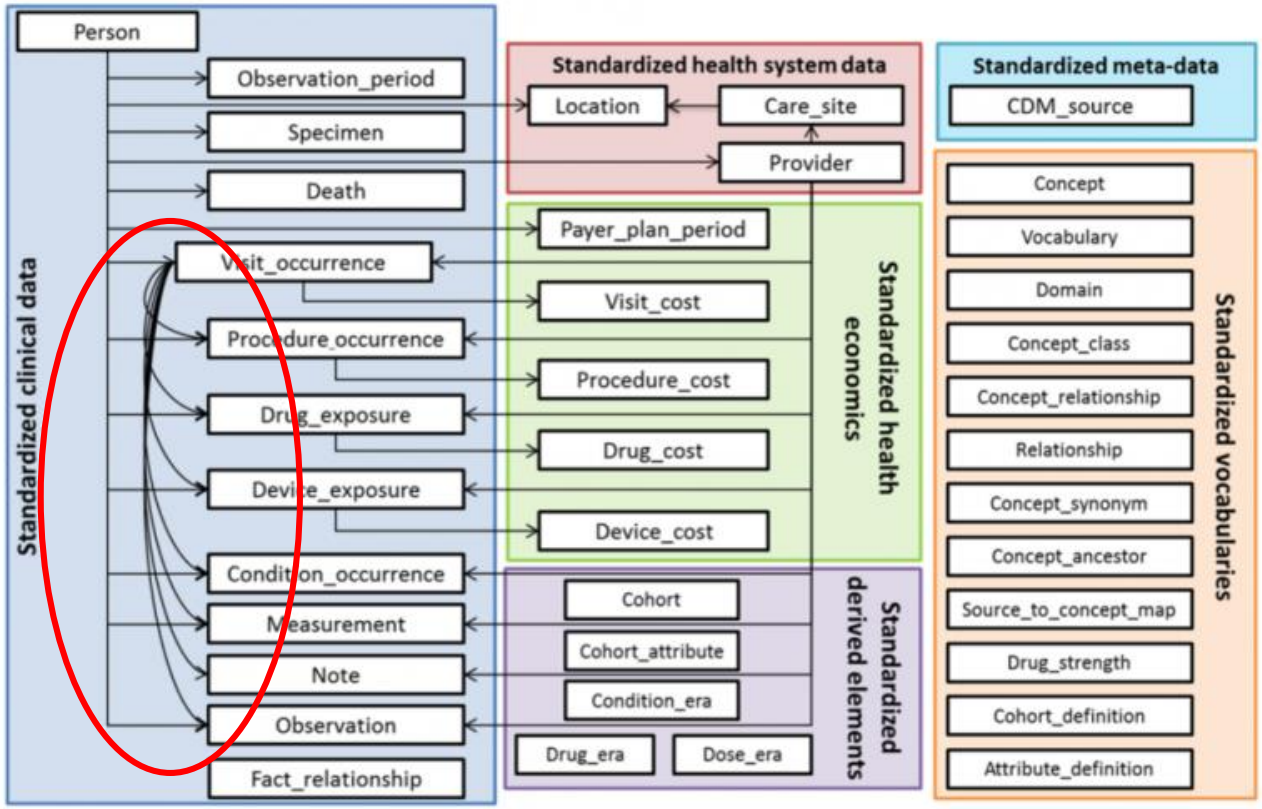


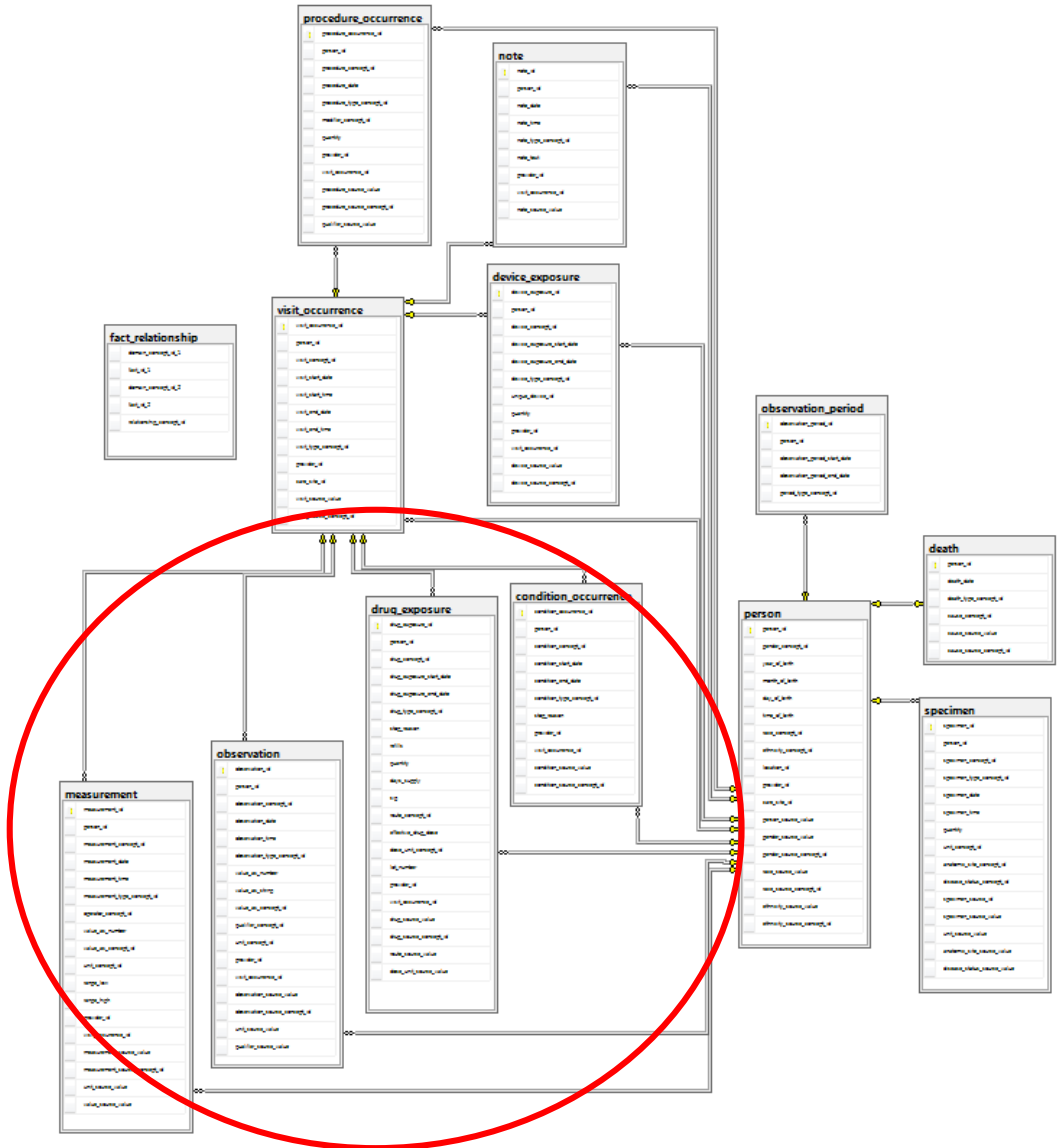
event table?


Person
Observation_period
Specimen
Death
Visit_occurrence
Procedure_occurrence
Drug_exposure
Device_exposure
Condition_occurrence
Measurement
Note
Observation
Fact_relationship
Standardized clinical data
Standardized health system data
Location
Care_site
Provider
Payer_plan_period
Visit_cost
Procedure_cost
Drug_cost
Device_cost
Standardized health economics
Cohort
Cohort_attribute
Condition_era
Drug_era
Dose_era
Standardized derived elements
Standardized meta-data
CDM_source
Concept
Vocabulary
Domain
Concept_class
Concept_relationship
Relationship
Concept_synonym
Concept_ancestor
Source_to_concept_map
Drug_strength
Cohort_definition
Attribute_definition
Standardized vocabularies

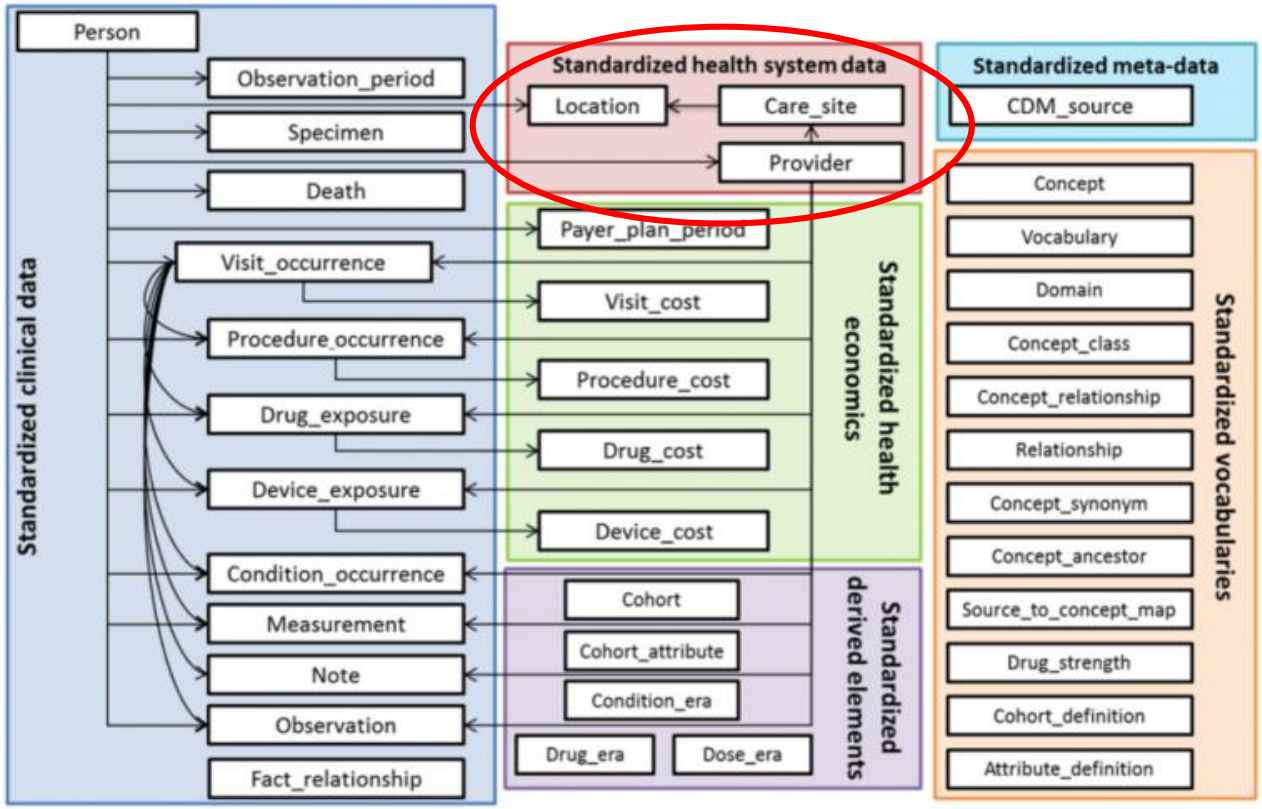

procedure_occurrence
note
visit_occurrence
device_exposure
fact_relationship
observation_period
death
drug_exposure
condition_occurrence
person
specimen
observation
measurement

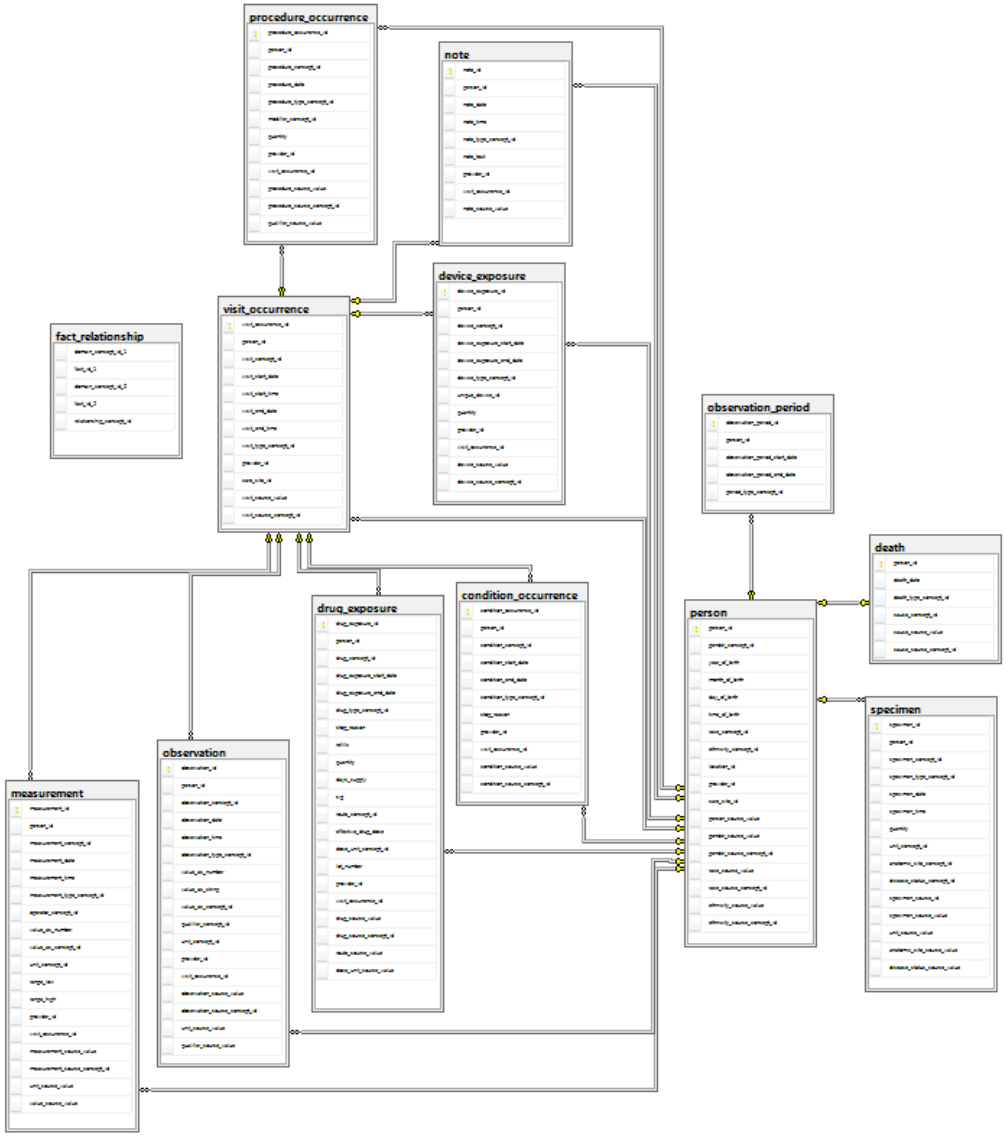
redundant foreign keys?





relationship cardinality?





many-to-many relationship?