post1-Ming-Chen

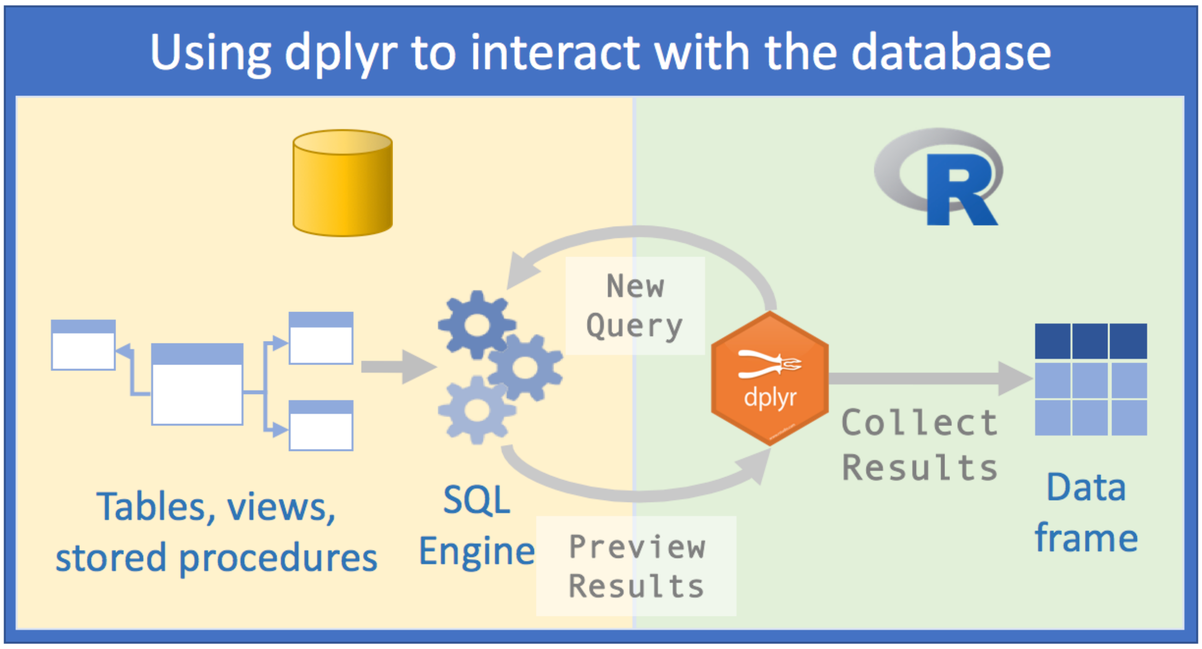
Ming Chen

October 28, 2017

Manipulating data with powerful "dplyr"

Introduction and background

Data frame is a common structure of data set. Unlike R matrice and R vector, data frame don't need to be atomic structure. Because of the variety of data types, manipulating data is more complicated in data table(data frame is a form of data table). We can see the following picture, which shows how dplyr plays an important roll in manipulating data.


Using dplyr to interact with the database
New Query
Tables, views, stored procedures
SQL Engine
dplyr
Collect Results
Preview Results
Data frame

We learnd two ways to manipulate data in stat133 lecture and labs: traditional way and dplyr.

Traditional way:

- bracket notation [ ]: select rows, columns, or element in specific location.
- dollar operator $: get specific columns.
- order( ): reorder rows
- aggregate( ): perform grouped operation
- unique( ): remove duplicate element
- sample( ): randomly select n columns

Dplyr:

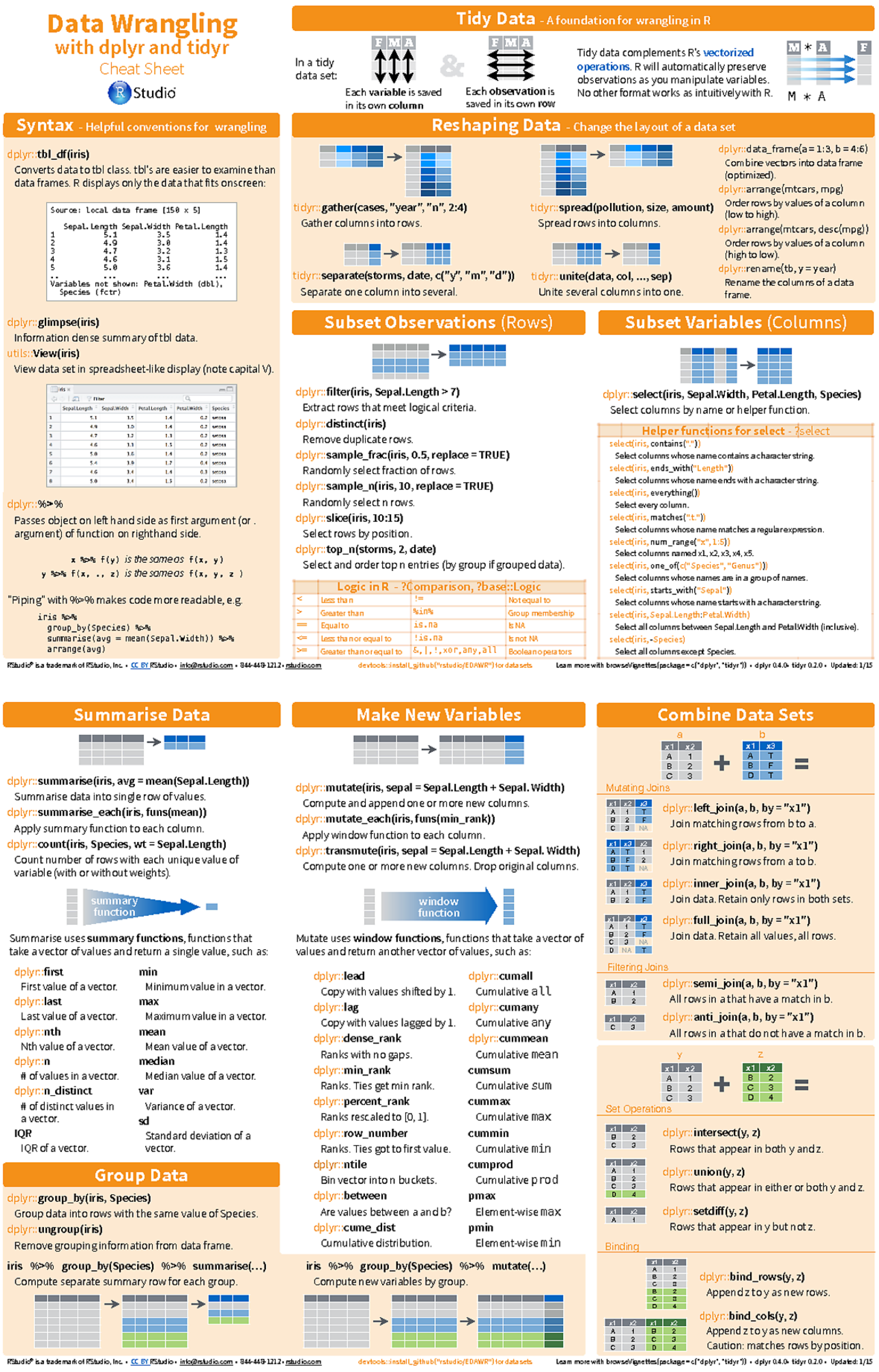
- slice( ): select rows by position
- filter( ): select rows by condition
- select( ): select columns by name
- mutate( ): add new variables
- arrange( ): reorder rows
- summarise( ): summarize values.
- group_by( ): perform grouped operation
- distinct( ):remove duplicate element
- sample_n( ): randomly select n rows
- sample_frac( ): randomly select fracttion of n rows

In this post, I will compare two different ways to accomplish the same tasks and introduce more powerful functions for set operation in dplyr.

- intersect( ): rows appears in both data frames
- union( ): rows appears in either or both data frames
- setdiff( ): rows appears in one data frame but not in another one.

Motivation

After I learned dplyr, I used it a lot in my following homeworks and labs because I found it is easier to use and the syntax is simple to understand. However, the functions we learned is to manipulate data in the single data frame. If we want to deal with two data sets, we need to know more functions to help with manipulating data. In this post, I will introduce more powerful functions for set operation in dplyr. Here is the cheatsheet about the powerful functions in dplyr.



Data Wrangling
with dplyr and tidyr
Cheat Sheet
RStudio

Tidy Data - A foundation for wrangling in R

In a tidy data set: Each **variable** is saved in its own **column** & Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.

Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**
Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]

   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**
Information dense summary of tbl data.

utils::**View(iris)**
View data set in spreadsheet-like display (note capital V).

dplyr::**%>%**
Passes object on left hand side as first argument (or . argument) of function on righthand side.

`x %>% f(y)` *is the same as* `f(x, y)`
`y %>% f(x, ., z)` *is the same as* `f(x, y, z )`

"Piping" with %>% makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Reshaping Data - Change the layout of a data set

tidyr::**gather(cases, "year", "n", 2:4)**
Gather columns into rows.

tidyr::**spread(pollution, size, amount)**
Spread rows into columns.

tidyr::**separate(storms, date, c("y", "m", "d"))**
Separate one column into several.

tidyr::**unite(data, col, ..., sep)**
Unite several columns into one.

dplyr::data_frame(a = 1:3, b = 4:6)
Combine vectors into data frame (optimized).

dplyr::arrange(mtcars, mpg)
Order rows by values of a column (low to high).

dplyr::arrange(mtcars, desc(mpg))
Order rows by values of a column (high to low).

dplyr::rename(tb, y = year)
Rename the columns of a data frame.

Subset Observations (Rows)

dplyr::**filter(iris, Sepal.Length > 7)**
Extract rows that meet logical criteria.

dplyr::**distinct(iris)**
Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**
Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**
Randomly select n rows.

dplyr::**slice(iris, 10:15)**
Select rows by position.

dplyr::**top_n(storms, 2, date)**
Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic			
<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&,|,!,xor,any,all	Boolean operators

Subset Variables (Columns)

dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**
Select columns by name or helper function.

Helper functions for select - ?select

select(iris, contains("."))
Select columns whose name contains a character string.

select(iris, ends_with("Length"))
Select columns whose name ends with a character string.

select(iris, everything())
Select every column.

select(iris, matches(".t."))
Select columns whose name matches a regular expression.

select(iris, num_range("x", 1:5))
Select columns named x1, x2, x3, x4, x5.

select(iris, one_of(c("Species", "Genus")))
Select columns whose names are in a group of names.

select(iris, starts_with("Sepal"))
Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)
Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, -Species)
Select all columns except Species.

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com   devtools::install_github("rstudio/EDAWR") for data sets   Learn more with browseVignettes(package = c("dplyr", "tidyr")) • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

Summarise Data

dplyr::**summarise(iris, avg = mean(Sepal.Length))**
Summarise data into single row of values.

dplyr::**summarise_each(iris, funs(mean))**
Apply summary function to each column.

dplyr::**count(iris, Species, wt = Sepal.Length)**
Count number of rows with each unique value of variable (with or without weights).

Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

dplyr::**first** First value of a vector.	**min** Minimum value in a vector.
dplyr::**last** Last value of a vector.	**max** Maximum value in a vector.
dplyr::**nth** Nth value of a vector.	**mean** Mean value of a vector.
dplyr::**n** # of values in a vector.	**median** Median value of a vector.
dplyr::**n_distinct** # of distinct values in a vector.	**var** Variance of a vector.
IQR IQR of a vector.	**sd** Standard deviation of a vector.

Group Data

dplyr::**group_by(iris, Species)**
Group data into rows with the same value of Species.

dplyr::**ungroup(iris)**
Remove grouping information from data frame.

iris %>% group_by(Species) %>% summarise(…)
Compute separate summary row for each group.

Make New Variables

dplyr::**mutate(iris, sepal = Sepal.Length + Sepal. Width)**
Compute and append one or more new columns.

dplyr::**mutate_each(iris, funs(min_rank))**
Apply window function to each column.

dplyr::**transmute(iris, sepal = Sepal.Length + Sepal. Width)**
Compute one or more new columns. Drop original columns.

Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

dplyr::**lead** Copy with values shifted by 1.	dplyr::**cumall** Cumulative all
dplyr::**lag** Copy with values lagged by 1.	dplyr::**cumany** Cumulative any
dplyr::**dense_rank** Ranks with no gaps.	dplyr::**cummean** Cumulative mean
dplyr::**min_rank** Ranks. Ties get min rank.	**cumsum** Cumulative sum
dplyr::**percent_rank** Ranks rescaled to [0, 1].	**cummax** Cumulative max
dplyr::**row_number** Ranks. Ties got to first value.	**cummin** Cumulative min
dplyr::**ntile** Bin vector into n buckets.	**cumprod** Cumulative prod
dplyr::**between** Are values between a and b?	**pmax** Element-wise max
dplyr::**cume_dist** Cumulative distribution.	**pmin** Element-wise min

iris %>% group_by(Species) %>% mutate(…)
Compute new variables by group.

Combine Data Sets

a

x1	x2
A	1
B	2
C	3

b

x1	x3
A	T
B	F
D	T

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::**left_join(a, b, by = "x1")**
Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::**right_join(a, b, by = "x1")**
Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::**inner_join(a, b, by = "x1")**
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::**full_join(a, b, by = "x1")**
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

dplyr::**semi_join(a, b, by = "x1")**
All rows in a that have a match in b.

x1	x2
C	3

dplyr::**anti_join(a, b, by = "x1")**
All rows in a that do not have a match in b.

y

x1	x2
A	1
B	2
C	3

z

x1	x2
B	2
C	3
D	4

Set Operations

x1	x2
B	2
C	3

dplyr::**intersect(y, z)**
Rows that appear in both y and z.

x1	x2
A	1
B	2
C	3
D	4

dplyr::**union(y, z)**
Rows that appear in either or both y and z.

x1	x2
A	1

dplyr::**setdiff(y, z)**
Rows that appear in y but not z.

Binding

x1	x2
A	1
B	2
C	3
B	2
C	3
D	4

dplyr::**bind_rows(y, z)**
Append z to y as new rows.

x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

dplyr::**bind_cols(y, z)**
Append z to y as new columns.
Caution: matches rows by position.

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com   devtools::install_github("rstudio/EDAWR") for data sets   Learn more with browseVignettes(package = c("dplyr", "tidyr")) • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

Get data set

I will use the data of birth to compare the functions in dplyr and base functions and introduce more dplyr functions. In this data set, it shows the number of births in each day from 1969 to 1978.Download material from

:https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/mosaicData/Births.csv

```
data <- read.csv('data/Births.csv')
dim(data)
```

```
## [1] 7305    9
```

```
head(data, 5)
```

```
##   X       date births  wday year month day day_of_year day_of_week
## 1 1 1969-01-01   8486   Wed 1969     1   1           1           3
## 2 2 1969-01-02   9002 Thurs 1969     1   2           2           4
## 3 3 1969-01-03   9542   Fri 1969     1   3           3           5
## 4 4 1969-01-04   8960   Sat 1969     1   4           4           6
## 5 5 1969-01-05   8390   Sun 1969     1   5           5           7
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
## 
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
## 
##     filter, lag
```

```
## The following objects are masked from 'package:base':
## 
##     intersect, setdiff, setequal, union
```

Tasks

Arrange function vs. order function

```
#use arrange to reorder rows by births
birth1 <- arrange(data, births)
head(birth1, 5)
```

```
##      X       date births wday year month day day_of_year day_of_week
## 1 2686 1976-05-09   6675  Sun 1976     5   9         130           7
## 2 1580 1973-04-29   6757  Sun 1973     4  29         120           7
## 3 2672 1976-04-25   6817  Sun 1976     4  25         116           7
## 4 2185 1974-12-25   6820  Wed 1974    12  25         360           3
## 5 1587 1973-05-06   6892  Sun 1973     5   6         127           7
```

```
dim(birth1)
```

```
## [1] 7305    9
```

```
# use order function to reorder rows by births
birth2 <- data[order(data$births), ]
head(birth2, 5)
```

```
##         X       date births wday year month day day_of_year day_of_week
## 2686 2686 1976-05-09   6675  Sun 1976     5   9         130           7
## 1580 1580 1973-04-29   6757  Sun 1973     4  29         120           7
## 2672 2672 1976-04-25   6817  Sun 1976     4  25         116           7
## 2185 2185 1974-12-25   6820  Wed 1974    12  25         360           3
## 1587 1587 1973-05-06   6892  Sun 1973     5   6         127           7
```

```
dim(birth2)
```

```
## [1] 7305    9
```

Let’s check the the index of the rows. We can know that arrange function gives a new data frame but order function manipulate the original data frame.

Filter function vs. bracket notation

```
#use filter function to present the births on Fridays in May, 1969.
filter(data, month == 5, year == 1969, wday == 'Fri')
```

```
##     X       date births wday year month day day_of_year day_of_week
## 1 122 1969-05-02   9736  Fri 1969     5   2         123           5
## 2 129 1969-05-09   9362  Fri 1969     5   9         130           5
## 3 136 1969-05-16   9824  Fri 1969     5  16         137           5
## 4 143 1969-05-23   9584  Fri 1969     5  23         144           5
## 5 150 1969-05-30   9154  Fri 1969     5  30         151           5
```

```
#use bracket notation to present the birth on Fridays in  May, 1969.
data[data$month== 5 & data$year == 1969 & data$wday ==  "Fri",]
```

```
##       X       date births wday year month day day_of_year day_of_week
## 122 122 1969-05-02   9736  Fri 1969     5   2         123           5
## 129 129 1969-05-09   9362  Fri 1969     5   9         130           5
## 136 136 1969-05-16   9824  Fri 1969     5  16         137           5
## 143 143 1969-05-23   9584  Fri 1969     5  23         144           5
## 150 150 1969-05-30   9154  Fri 1969     5  30         151           5
```

When you use bracket notation, you should be careful on subsetting. Filter function is easier to understand and use.

Distinct vs. unique

```
#use distinct function to find out different weekdays in data set
distinct(data, wday)
```

```
##    wday
## 1   Wed
## 2 Thurs
## 3   Fri
## 4   Sat
## 5   Sun
## 6   Mon
## 7  Tues
```

```
#use unique function to find out different weekdays in data set
unique(data$wday)
```

```
## [1] Wed   Thurs Fri   Sat   Sun   Mon   Tues
## Levels: Fri Mon Sat Sun Thurs Tues Wed
```

They both give us distinct or unique element in specific column. Unique function works with list.We can also use distinct(data_frame)to remove duplicate rows.

sample_n/sample_frac vs sample

```
#use sample_n function to randomly choose two rows from the data set.
sample_n(data, size = 2)
```

```
##         X       date births wday year month day day_of_year day_of_week
## 3526 3526 1978-08-27   8471  Sun 1978     8  27         240           7
## 4754 4754 1982-01-06  10182  Wed 1982     1   6           6           3
```

```
#use sample_frac function to randomly choose two rows from the data set.
# At first, we should figure out how many rows in total.
dim(data)
```

```
## [1] 7305    9
```

```
# There are 7305 rows in total. If we want to choose 2 rows, the fraction should be (2/7305)
sample_frac(data, size = 2/7305)
```

```
##         X       date births  wday year month day day_of_year day_of_week
## 1803 1803 1973-12-08   7682   Sat 1973    12   8         343           6
## 1570 1570 1973-04-19   8672 Thurs 1973     4  19         110           4
```

```
#use sample function to randomly choose two columns from the data set.
head(sample(data, size = 2), 5)
```

```
##         date day_of_year
## 1 1969-01-01           1
## 2 1969-01-02           2
## 3 1969-01-03           3
## 4 1969-01-04           4
## 5 1969-01-05           5
```

In data frame, we can easily to randomly choose two rows or any fraction of rows by using sample_n and sample_frac in dplyr. The base function sample( ) randomly choose columns instead.

Set operation

At first, we use filter to get two new data sets about births' data between year 1969 and 1972 and between year 1970 and 1973

```
#births' data between 1969 and 1972
bir_7071 <- filter(data, year < 1972 & year > 1969)
#births' data between 1970 and 1973
bir_7172 <- filter(data, year < 1973 & year > 1970 )
```

Intersect function

```
#show births' data in both bir_7071 and bir_7172
inter_71 <- intersect(bir_7071, bir_7172)
distinct(inter_71, year)
```

```
##   year
## 1 1971
```

The birth's data in both bir_7071 and bir_7172 should be the data of year 1971. By using the intersect function, we can get the intersection of both data frames.

Union function

```
# show births' data between 1969 and 1973
uni_707172 <- union(bir_7071, bir_7172)
distinct(uni_707172, year)
```

```
##   year
## 1 1972
## 2 1971
## 3 1970
```

The birth's data in either or both bir_7071 and bir_7172 should be the data between 1969 and 1973. By using the union function, we can get the union of both data frames.

Setdiff function

```
#rows appears in one bir_7071 but not in bir_7172.
diff <- setdiff(bir_7071, bir_7172)
distinct(diff, year)
```

```
##   year
## 1 1970
```

```
#rows appears in one bir_7172 but not in bir_7071.
diff <- setdiff(bir_7172, bir_7071)
distinct(diff, year)
```

```
##   year
## 1 1972
```

The birth's data in bir_7071 but not in bir_7172 should be the data in year 1970. The birth's data in bir_7172 but not in bir_7071 should be the data in year 1972.By using the setdiff function, we can get the rows appears in one data frame but not in another one.

Conclusion

Dplyr has powerful and intelligible functions to manipulate data. Dplyr can replace most of your common R idioms. Both base functions and dplyr funtions can reach the same results. However, the functions in dplyr is easier to remember and understand. It also provides more functions to find intersection, union and complement, which are useful when we compare two data sets.

References:

data base using R (https://rviews.rstudio.com/2017/05/17/databases-using-r/)

setops(https://rdrr.io/cran/dplyr/man/setops.html)

set operations (https://stackoverflow.com/questions/38921580/set-operations-with-dplyrrowwise) dplyr tutorial (http://genomicsclass.github.io/book/pages/dplyr_tutorial.html)

How dplyr replaced my most common R idioms (http://www.onthelambda.com/2014/02/10/how-dplyr-replaced-my-most-common-r-idioms/)

Performance Comparison of a dplyr Function and a Corresponding Base R Function (http://rstudio-pubs-static.s3.amazonaws.com/45821_38ecfbdf02494a738867f37ecc55f2b2.html)

dplyr and base R filter speed comparison (https://rpubs.com/alsboston/45922)

data set material (https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/mosaicData/Births.csv)