Data Manipulation: R package dplyr

Chih-Hui Wang (Jason)

April 28, 2015; Revised: March 10, 2016

This is an introduction to the R package `dplyr`, written by Hadley Wickham. Personally, I found it incredibly easy and useful to use `dplyr` when I did data manipulation. It's usually the first package I call when I start a data analysis project. The functions in the package work similar to SQL syntax. You almost can find any database function you want in `dplyr`. We will start by introducing the class `tbl`, the basic functions (verbs) to other special functions. We will also talk about how the amazing syntax %>% (like the pipe in Unix) makes your code more elegant.

```
library(dplyr)
library(hflights)
library(rbenchmark)
```

We use `hflight` dataset for demo.

```
data(hflights)
names(hflights)
```

```
 [1] "Year"              "Month"             "DayofMonth"
 [4] "DayOfWeek"         "DepTime"           "ArrTime"
 [7] "UniqueCarrier"     "FlightNum"         "TailNum"
[10] "ActualElapsedTime" "AirTime"           "ArrDelay"
[13] "DepDelay"          "Origin"            "Dest"
[16] "Distance"          "TaxiIn"            "TaxiOut"
[19] "Cancelled"         "CancellationCode"  "Diverted"
```

1. Data Type: tbl (table)

First of all, we introduce the class `tbl`. It will be better if we change the class to `tbl`. When you print the data.frame, R will dump everything to the console, even the data.frame is too big to fit in your console. However, if you convert the class to `tbl` and print it, it will give you a better output format. Note that the function in `dplyr` can be used to data.frame as well. We will compare the difference between `tbl` and data.frame later.

```
# hflights_df is data.frame
hflights_df <- hflights
class(hflights_df)
```

```
[1] "data.frame"
```

```
# Transform data.frame to table
hflights <- tbl_df(hflights)
class(hflights)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

```
# Nicer output format
hflights
```

```
Source: local data frame [227,496 x 21]

    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier
   (int) (int)      (int)     (int)   (int)   (int)          (chr)
1   2011     1          1         6    1400    1500             AA
2   2011     1          2         7    1401    1501             AA
3   2011     1          3         1    1352    1502             AA
4   2011     1          4         2    1403    1513             AA
5   2011     1          5         3    1405    1507             AA
6   2011     1          6         4    1359    1503             AA
7   2011     1          7         5    1359    1509             AA
8   2011     1          8         6    1355    1454             AA
9   2011     1          9         7    1443    1554             AA
10  2011     1         10         1    1443    1553             AA
..   ...   ...        ...       ...     ...     ...            ...
Variables not shown: FlightNum (int), TailNum (chr), ActualElapsedTime
  (int), AirTime (int), ArrDelay (int), DepDelay (int), Origin (chr), Dest
  (chr), Distance (int), TaxiIn (int), TaxiOut (int), Cancelled (int),
  CancellationCode (chr), Diverted (int)
```

2. Verb 1: filter (Row Operation)

Subset is one of the most important things in data manipulation. In base R, we will use `subset` with conditions or just boolean vector with square brackets `[` to retrieve the rows we want. In `dplyr`, we use `filter(data, conditions)` to subset the data.

Let's say that we want to get the flights in March.

```
# In dplyr
filter(hflights, Month == 3)
```

```
Source: local data frame [19,470 x 21]

    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier
   (int) (int)      (int)     (int)   (int)   (int)          (chr)
1   2011     3          1         2    1633    1734             AA
2   2011     3          2         3    1632    1750             AA
3   2011     3          3         4    1635    1734             AA
4   2011     3          4         5    1630    1747             AA
5   2011     3          5         6    1633    1745             AA
6   2011     3          6         7    1630    1730             AA
7   2011     3          7         1    1627    1737             AA
8   2011     3          8         2    1650    1749             AA
9   2011     3          9         3    1635    1744             AA
10  2011     3         10         4    1636    1749             AA
..   ...   ...        ...       ...     ...     ...            ...
Variables not shown: FlightNum (int), TailNum (chr), ActualElapsedTime
  (int), AirTime (int), ArrDelay (int), DepDelay (int), Origin (chr), Dest
  (chr), Distance (int), TaxiIn (int), TaxiOut (int), Cancelled (int),
  CancellationCode (chr), Diverted (int)
```

```
##### The following codes will give your the same result #####
# subset(hflights, Month == 3)
# hflights[hflights$Month == 3, ]
```

If you have multiple condition, you can use commas to separate several conditions. For example, `filter(hflights, Month == 3, DayofMonth == 16)` will pull out the filght on March 16. Also, you can use & and | operation in the arguments like `filter(hflights, Month == 3 & DayofMonth == 16)`.

We use the `benchmark` function in package `rbenchmark` to compare four methods. We replicate the procedure 100 times and calculate the time. You can also use the `system.time` to compare method while its result varies. `benchmark` gives us a more robust comparison.
From the results, you can find that `filter` with class `tbl` is the fastest. As we mentioned earlier, the class `tbl` will be better than data.frame for large dataset.

```
# compare four methods
benchmark(
  method1=filter(hflights, Month == 3),
  method1_df=filter(hflights_df, Month == 3),
  method2=subset(hflights, Month == 3),
  method3=hflights[hflights$Month == 3, ],
  replications=100,
  columns=c("test", "elapsed", "relative", "user.self", "sys.self")
)
```

```
        test elapsed relative user.self sys.self
1    method1    0.37    1.000      0.36     0.02
2 method1_df    0.45    1.216      0.45     0.00
3    method2    2.01    5.432      1.90     0.11
4    method3    1.50    4.054      1.45     0.04
```

3. Verb 2: select (Column Operation)

We can use `select` function to select the column. We just put the column name in the arguments.

```
# In dplyr
select(hflights, Month, DayofMonth, FlightNum)
```

```
Source: local data frame [227,496 x 3]

   Month DayofMonth FlightNum
   (int)      (int)     (int)
1      1          1       428
2      1          2       428
3      1          3       428
4      1          4       428
5      1          5       428
6      1          6       428
7      1          7       428
8      1          8       428
9      1          9       428
10     1         10       428
..   ...        ...       ...
```

If you have a lot of column names shared some characters, without typing the explicit name you can use `contains`, `starts_with` and `ends_with` to pull out those columns. (Note that we do not show the result here. You can try the code.)

```
# contains
select(hflights, contains("delay"))

# starts_with
select(hflights, starts_with("M"))

# use ends_with
select(hflights, ends_with("th"))
```

A nicer way to write codes in dplyr

Sometimes we will combine the `filter` and `select` function together. You may want to find out the flight number and Origin of those American Airlines (AA) flights. You will write code like the following:

```
filter(select(hflights, FlightNum, Origin, UniqueCarrier), UniqueCarrier == "AA")
```

When you see the above codes, it may take a while to get what the codes is doing. However, in `dplyr`, it gives you a more elegant way to organize their code. It works like the pipe in Unix. We can rewrite the code as following:

```
# Clearer way to write code
hflights %>%
    select(FlightNum, Origin, UniqueCarrier) %>%
    filter(UniqueCarrier == "AA")
```

```
Source: local data frame [3,244 x 3]

   FlightNum Origin UniqueCarrier
       (int)  (chr)         (chr)
1        428    IAH            AA
2        428    IAH            AA
3        428    IAH            AA
4        428    IAH            AA
5        428    IAH            AA
6        428    IAH            AA
7        428    IAH            AA
8        428    IAH            AA
9        428    IAH            AA
10       428    IAH            AA
..       ...    ...           ...
```

You can interpret the %>% syntax as "then". The above code will be like that we first get the hflights data, select FlightNum, Origin, UniqueCarrier, and then filter the outcome to find those UniqueCarrier is AA. It throws the output in front of it to the argument of next function. In the following demo, I will use %>%.

4. Verb 3 arrange

Suppose we want to find out which airlines had the longest flights delayed in the data. We have to sort our data by the delay time. To do it in `dplyr`, we can use `arrange` function.

```
# To sort decreasingly, use desc
hflights %>%
  select(DepDelay, UniqueCarrier) %>%
  arrange(desc(DepDelay))
```

```
Source: local data frame [227,496 x 2]

   DepDelay UniqueCarrier
      (int)         (chr)
1       981            CO
2       970            AA
3       931            MQ
4       869            UA
5       814            MQ
6       803            MQ
7       780            CO
8       758            CO
9       730            DL
10      691            MQ
..      ...           ...
```

5. Verb 4 mutate

Sometimes, you may want to create a new variable. For instance, if we want to create the date variable in flight data, use `mutate` function.

```
# Create a new variable, date
hflights <- hflights %>%
  mutate(date=paste(Year, Month, DayofMonth, sep="-"))

hflights %>% select(date)
```

```
Source: local data frame [227,496 x 1]

        date
       (chr)
1   2011-1-1
2   2011-1-2
3   2011-1-3
4   2011-1-4
5   2011-1-5
6   2011-1-6
7   2011-1-7
```

```
8   2011-1-8
9   2011-1-9
10 2011-1-10
..       ...
```

6. Verb 5 summarise + Verb 6 group_by

Often, we will want to do something based on several groups such as what is the average delayed time for each airline. In base R, we use `tapply(hflights$DepDelay, hflights$UniqueCarrier, mean, na.rm=TRUE)` or `aggregate(DepDelay ~ UniqueCarrier, hflights, mean, na.rm=TRUE)`.

```
# Use group_by + summarise
hflights %>%
  group_by(UniqueCarrier) %>%
  summarise(average_delay=mean(DepDelay, na.rm=T))
```

```
Source: local data frame [15 x 2]

   UniqueCarrier average_delay
           (chr)         (dbl)
1             AA      6.390144
2             AS      3.712329
3             B6     13.320532
4             CO      9.261313
5             DL      9.370627
6             EV     12.482193
7             F9      5.093637
8             FL      4.716376
9             MQ     11.071745
10            OO      8.885482
11            UA     12.918707
12            US      1.622926
13            WN     13.488241
14            XE      7.713728
15            YV      1.538462
```

Sometimes, we may want to apply the function to several columns or apply different functions to column.

```
#Apply function to multiple columns
hflights %>%
  group_by(UniqueCarrier) %>%
  summarise_each(funs(mean(., na.rm=T)), DepDelay, ArrDelay)
```

```
Source: local data frame [15 x 3]

   UniqueCarrier  DepDelay   ArrDelay
           (chr)     (dbl)      (dbl)
1             AA  6.390144  0.8917558
2             AS  3.712329  3.1923077
3             B6 13.320532  9.8588410
4             CO  9.261313  6.0986983
5             DL  9.370627  6.0841374
6             EV 12.482193  7.2569543
7             F9  5.093637  7.6682692
8             FL  4.716376  1.8536239
9             MQ 11.071745  7.1529751
10            OO  8.885482  8.6934922
11            UA 12.918707 10.4628628
12            US  1.622926 -0.6307692
13            WN 13.488241  7.5871430
14            XE  7.713728  8.1865242
15            YV  1.538462  4.0128205
```

```
#Apply multiple functions to one column
hflights %>%
  group_by(UniqueCarrier) %>%
  summarise_each(funs(mean(., na.rm=T), min(., na.rm=T), max(., na.rm=T)), DepDelay)
```

```
Source: local data frame [15 x 4]

   UniqueCarrier      mean   min   max
           (chr)     (dbl) (int) (int)
1             AA  6.390144   -15   970
2             AS  3.712329   -15   172
3             B6 13.320532   -14   310
4             CO  9.261313   -18   981
5             DL  9.370627   -17   730
6             EV 12.482193   -18   479
7             F9  5.093637   -15   275
8             FL  4.716376   -14   507
9             MQ 11.071745   -23   931
10            OO  8.885482   -33   360
11            UA 12.918707   -11   869
12            US  1.622926   -17   425
13            WN 13.488241   -10   548
14            XE  7.713728   -19   628
15            YV  1.538462   -11    54
```

7. Others

In this section, we are going to introduce some useful functions that can facilitate our analysis. The first one is `n()`. It can help us count the number in the group. For example, we want to calculate the number of flights for each airline.

```
#In dplyr
hflights %>%
  group_by(UniqueCarrier) %>%
  summarise(flight_count=n())
```

```
Source: local data frame [15 x 2]

   UniqueCarrier flight_count
           (chr)        (int)
1             AA         3244
2             AS          365
3             B6          695
4             CO        70032
5             DL         2641
6             EV         2204
7             F9          838
8             FL         2139
9             MQ         4648
10            OO        16061
11            UA         2072
12            US         4082
13            WN        45343
14            XE        73053
15            YV           79
```

Suppose we also want to sort the airlines by the number of flights. We can add `arrange(fligh_count)` after `summarise`. Alternative is to use `count` function with argument `sort=TRUE` or, another useful function can count and sort at the same time:

```
#Another easier way to write it
hflights %>%
  group_by(UniqueCarrier) %>%
  tally(sort=TRUE)
```

```
Source: local data frame [15 x 2]

   UniqueCarrier     n
```

```
            (chr) (int)
1              XE 73053
2              CO 70032
3              WN 45343
4              OO 16061
5              MQ  4648
6              US  4082
7              AA  3244
8              DL  2641
9              EV  2204
10             FL  2139
11             UA  2072
12             F9   838
13             B6   695
14             AS   365
15             YV    79
```

`n_distinct` funtion provides the number of unique subjects in a column.

```
hflights %>%
  group_by(Dest) %>%
  summarise(flight_count=n(), plane_count=n_distinct(TailNum))
```

```
Source: local data frame [116 x 3]

    Dest flight_count plane_count
   (chr)        (int)       (int)
1    ABQ         2812         716
2    AEX          724         215
3    AGS            1           1
4    AMA         1297         158
5    ANC          125          38
6    ASE          125          60
7    ATL         7886         983
8    AUS         5022        1015
9    AVL          350         142
10   BFL          504          70
..   ...          ...         ...
```

To check the data types of each column, use `glimpse`.

```
#Like the str function in base R
glimpse(hflights)
```

```
Observations: 227,496
Variables: 22
$ Year              (int) 2011, 2011, 2011, 2011, 2011, 2011, 2011, 20...
$ Month             (int) 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
$ DayofMonth        (int) 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1...
$ DayOfWeek         (int) 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6,...
$ DepTime           (int) 1400, 1401, 1352, 1403, 1405, 1359, 1359, 13...
$ ArrTime           (int) 1500, 1501, 1502, 1513, 1507, 1503, 1509, 14...
$ UniqueCarrier     (chr) "AA", "AA", "AA", "AA", "AA", "AA", "AA", "A...
$ FlightNum         (int) 428, 428, 428, 428, 428, 428, 428, 428, 428,...
$ TailNum           (chr) "N576AA", "N557AA", "N541AA", "N403AA", "N49...
$ ActualElapsedTime (int) 60, 60, 70, 70, 62, 64, 70, 59, 71, 70, 70, ...
$ AirTime           (int) 40, 45, 48, 39, 44, 45, 43, 40, 41, 45, 42, ...
$ ArrDelay          (int) -10, -9, -8, 3, -3, -7, -1, -16, 44, 43, 29,...
$ DepDelay          (int) 0, 1, -8, 3, 5, -1, -1, -5, 43, 43, 29, 19, ...
$ Origin            (chr) "IAH", "IAH", "IAH", "IAH", "IAH", "IAH", "I...
$ Dest              (chr) "DFW", "DFW", "DFW", "DFW", "DFW", "DFW", "D...
$ Distance          (int) 224, 224, 224, 224, 224, 224, 224, 224, 224,...
$ TaxiIn            (int) 7, 6, 5, 9, 9, 6, 12, 7, 8, 6, 8, 4, 6, 5, 6...
$ TaxiOut           (int) 13, 9, 17, 22, 9, 13, 15, 12, 22, 19, 20, 11...
$ Cancelled         (int) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ CancellationCode  (chr) "", "", "", "", "", "", "", "", "", "", "", ...
$ Diverted          (int) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ date              (chr) "2011-1-1", "2011-1-2", "2011-1-3", "2011-1-...
```

We can also do sampling easily either by number or fraction.

```
# Sample by number
hflights %>% sample_n(10)

# Or by fraction
hflights %>% sample_frac(0.25, replace=T)
```

8. Connecting with database

- dplyr can connect to a database as if the data was loaded into a data frame
- Instruction for create a database

```
# connect to an SQLite database
my_db <- src_sqlite("my_db.sqlite3")

# connect to the "hflights" table in that database
```

```r
flights_tbl <- tbl(my_db, "hflights")

# identical query using the database
flights_tbl %>%
    select(UniqueCarrier, DepDelay) %>%
    arrange(desc(DepDelay))

# ask dplyr for the SQL commands
flights_tbl %>%
    select(UniqueCarrier, DepDelay) %>%
    arrange(desc(DepDelay)) %>%
    explain()
```

References

- Introduction to dplyr on CRAN
- Hands-on dplyr tutorial for faster data manipulation in R
- A more comprehensive and advanced tutorial