Week 5. Big Data Analytics
data.frame manipulation with dplyr

Hyeonsu B. Kang
hyk149@eng.ucsd.edu

April 2016

1 Join with dplyr

In the last lecture we have seen how to efficiently manipulate a single table of data using dplyr's group_by, filter, and mutate. In today's lecture, we will learn how to cope with many tables that contribute to an analysis. (The following content below is adapted from https://goo.gl/T29rlU)

Install the nycflight13 package using

```
> install.packages("nycflights13")
> library(nycflights13)
```

This package contains information about all flights that departed from NYC (e.g. EWR, JFK and LGA) in 2013. 336,776 flights in total. To help understand what causes delays, it also includes a number of other useful datasets:

- weather: hourly meterological data for each airport
- planes: construction information about each plane
- airports: airport names and locations
- airlines: translation between two letter carrier codes and names

There are 3 verbs to remember in working with two tables at a time

- **Mutating joins**, which add new variables to one table from matching rows in another
- **Filtering joins**, which filter observations from one table based on whether or not they match an observation in the other table
- **Set operations**, which combine the observations in the data sets as if they were set elements

1.1 Mutating joins

Mutating joins allow you to combine variables from multiple tables. First, let us have a look into the flights and airlines data sets.

```
> dim(flights)
[1] 336776     16
> dim(airlines)
[1] 16  2
```

In flights, we have flight information with an abbreviation for carrier, and in airlines we have a mapping between abbreviations and full names. First, let us create a subset of the flights data set with only year, month, day, hour, origin, dest, tailnum and carrier variables.

```
> subdata = flights %>% select(year, month, day, hour, origin, dest, tailnum, carrier)
```

(if you hit an error here, you probably have not loaded the dplyr package).

1.2 Controlling how the tables are matched

Then, joining `subdata` with carrier names in `airlines` with `left_join()` is as follows:

```
> subdata = flights %>% select(year, month, day, hour, origin, dest, tailnum, carrier)
> head(subdata)
Source: local data frame [6 x 8]

   year month   day  hour origin  dest tailnum carrier
  (int) (int) (int) (dbl)  (chr) (chr)   (chr)   (chr)
1  2013     1     1     5    EWR   IAH  N14228      UA
2  2013     1     1     5    LGA   IAH  N24211      UA
3  2013     1     1     5    JFK   MIA  N619AA      AA
4  2013     1     1     5    JFK   BQN  N804JB      B6
5  2013     1     1     5    LGA   ATL  N668DN      DL
6  2013     1     1     5    EWR   ORD  N39463      UA

t
> subdata %>% left_join(airlines)
Joining by: "carrier"
Source: local data frame [336,776 x 9]

    year month   day  hour origin  dest tailnum carrier                     name
   (int) (int) (int) (dbl)  (chr) (chr)   (chr)   (chr)                   (fctr)
1   2013     1     1     5    EWR   IAH  N14228      UA    United Air Lines Inc.
2   2013     1     1     5    LGA   IAH  N24211      UA    United Air Lines Inc.
3   2013     1     1     5    JFK   MIA  N619AA      AA   American Airlines Inc.
4   2013     1     1     5    JFK   BQN  N804JB      B6          JetBlue Airways
5   2013     1     1     5    LGA   ATL  N668DN      DL     Delta Air Lines Inc.
6   2013     1     1     5    EWR   ORD  N39463      UA    United Air Lines Inc.
7   2013     1     1     5    EWR   FLL  N516JB      B6          JetBlue Airways
8   2013     1     1     5    LGA   IAD  N829AS      EV ExpressJet Airlines Inc.
9   2013     1     1     5    JFK   MCO  N593JB      B6          JetBlue Airways
10  2013     1     1     5    LGA   ORD  N3ALAA      AA   American Airlines Inc.
..   ...   ...   ...   ...    ...   ...     ...     ...                      ...
```

Each mutating join takes an argument by that controls which variables are used to match observations in the two tables. There are a few ways to specify it, as illustrated below with various tables from `nycflights13`:

- `NULL`, the default (**natural join**): `dplyr` will use all variables that appear in both tables. For example, the `flights` and `weather` tables match on their common variables: `year`, `month`, `day`, `hour` and `origin`.

```
> subdata %>% left_join(weather)
Joining by: c("year", "month", "day", "hour", "origin")
Source: local data frame [336,776 x 17]

    year month   day  hour origin  dest tailnum carrier  temp  dewp humid wind_dir
   (dbl) (dbl) (int) (dbl)  (chr) (chr)   (chr)   (chr) (dbl) (dbl) (dbl)    (dbl)
1   2013     1     1     5    EWR   IAH  N14228      UA    NA    NA    NA       NA
2   2013     1     1     5    LGA   IAH  N24211      UA    NA    NA    NA       NA
3   2013     1     1     5    JFK   MIA  N619AA      AA    NA    NA    NA       NA
4   2013     1     1     5    JFK   BQN  N804JB      B6    NA    NA    NA       NA
5   2013     1     1     5    LGA   ATL  N668DN      DL    NA    NA    NA       NA
6   2013     1     1     5    EWR   ORD  N39463      UA    NA    NA    NA       NA
7   2013     1     1     5    EWR   FLL  N516JB      B6    NA    NA    NA       NA
8   2013     1     1     5    LGA   IAD  N829AS      EV    NA    NA    NA       NA
9   2013     1     1     5    JFK   MCO  N593JB      B6    NA    NA    NA       NA
10  2013     1     1     5    LGA   ORD  N3ALAA      AA    NA    NA    NA       NA
..   ...   ...   ...   ...    ...   ...     ...     ...   ...   ...   ...      ...
Variables not shown: wind_speed (dbl), wind_gust (dbl), precip (dbl), pressure (dbl),
visib (dbl)
```

- A character vector `x`, with `by = "x"`. Like a natural join, but uses only some of the common variables. For example, `flights` and `planes` have `year` columns, but they have different meanings so we only want to join by `tailnum`.

```
> subdata %>% left_join(planes, by = "tailnum")
Source: local data frame [336,776 x 16]
```

```
   year.x month   day  hour origin  dest tailnum carrier year.y type
    (int) (int) (int) (dbl)  (chr) (chr)   (chr)   (chr)  (int) (chr)
1    2013     1     1     5    EWR   IAH  N14228      UA   1999 Fixed wing multi engine
2    2013     1     1     5    LGA   IAH  N24211      UA   1998 Fixed wing multi engine
3    2013     1     1     5    JFK   MIA  N619AA      AA   1990 Fixed wing multi engine
4    2013     1     1     5    JFK   BQN  N804JB      B6   2012 Fixed wing multi engine
5    2013     1     1     5    LGA   ATL  N668DN      DL   1991 Fixed wing multi engine
6    2013     1     1     5    EWR   ORD  N39463      UA   2012 Fixed wing multi engine
7    2013     1     1     5    EWR   FLL  N516JB      B6   2000 Fixed wing multi engine
8    2013     1     1     5    LGA   IAD  N829AS      EV   1998 Fixed wing multi engine
9    2013     1     1     5    JFK   MCO  N593JB      B6   2004 Fixed wing multi engine
10   2013     1     1     5    LGA   ORD  N3ALAA      AA     NA                       NA
..    ...   ...   ...   ...    ...   ...     ...     ...    ...                      ...
Variables not shown: manufacturer (chr), model (chr), engines (int), seats (int),
speed (int), engine (chr)
```

Note that the `year` columns in the output are disambiguated with suffix `.x` and `.y`.

- A named character, i.e. `by = c("x" = "a")`. This will match variable `x` in the first table to variable `a` in the second. For example, each flight has an origin and destination `airport`, so we need to specify which one we want to join to:

```
> subdata = flights %>% select(year, month, day, hour, origin, dest, tailnum, carrier)
> subdata %>% left_join(airports, c("dest" = "faa"))
Source: local data frame [336,776 x 14]

    year month   day  hour origin  dest tailnum carrier  name      lat
   (int) (int) (int) (dbl)  (chr) (chr)   (chr)   (chr) (chr)    (dbl)
1   2013     1     1     5    EWR   IAH  N14228      UA   ...  29.98443
2   2013     1     1     5    LGA   IAH  N24211      UA   ...  29.98443
3   2013     1     1     5    JFK   MIA  N619AA      AA   ...  25.79325
4   2013     1     1     5    JFK   BQN  N804JB      B6   ...        NA
5   2013     1     1     5    LGA   ATL  N668DN      DL   ...  33.63672
6   2013     1     1     5    EWR   ORD  N39463      UA   ...  41.97860
7   2013     1     1     5    EWR   FLL  N516JB      B6   ...  26.07258
8   2013     1     1     5    LGA   IAD  N829AS      EV   ...  38.94453
9   2013     1     1     5    JFK   MCO  N593JB      B6   ...  28.42939
10  2013     1     1     5    LGA   ORD  N3ALAA      AA   ...  41.97860
..   ...   ...   ...   ...    ...   ...     ...     ...   ...       ...
Variables not shown: lon (dbl), alt (int), tz (dbl), dst (chr)
```

1.3 Types of join

So far, we have used the `left_join()` operation only. In fact, there are four types of mutating join, which differ in their behavior when a match is not found. We'll illustrate each with a simple example below.

```
> (df1 = data_frame(x = c(1,2), y = 2:1))
Source: local data frame [2 x 2]

      x     y
  (dbl) (int)
1     1     2
2     2     1

> (df2 = data_frame(x = c(1,3), a = 10, b = "a"))
Source: local data frame [2 x 3]

      x     a     b
  (dbl) (dbl) (chr)
1     1    10     a
2     3    10     a
```

(1) `inner_join(x, y)` only includes observations that match in both `x` and `y`

```
> df1 %>% inner_join(df2)
Joining by: "x"
Source: local data frame [1 x 4]

x     y     a     b
(dbl) (int) (dbl) (chr)
1     1     2    10      a
```

(2) `left_join(x, y)` includes all observations in `x`, regardless of whether they match or not. This is the most commonly used join because it ensures that you do not lose observations from your primary table.

```
> df1 %>% left_join(df2)
Joining by: "x"
Source: local data frame [2 x 4]

x     y     a     b
(dbl) (int) (dbl) (chr)
1     1     2    10      a
2     2     1    NA     NA
```

(3) `right_join(x, y)` includes all observations in `y`. It is equivalent to `left_join(y, x)`, but the columns will be ordered differently.

```
> df1 %>% right_join(df2)
Joining by: "x"
Source: local data frame [2 x 4]

x     y     a     b
(dbl) (int) (dbl) (chr)
1     1     2    10      a
2     3    NA    10      a

> df2 %>% left_join(df1)
Joining by: "x"
Source: local data frame [2 x 4]

x     a     b     y
(dbl) (dbl) (chr) (int)
1     1    10      a     2
2     3    10      a    NA
```

(4) `full_join()` includes all observations from `x` and `y`.

```
> df1 %>% full_join(df2)
Joining by: "x"
Source: local data frame [3 x 4]

      x     y     a     b
  (dbl) (int) (dbl) (chr)
1     1     2    10      a
2     2     1    NA     NA
3     3    NA    10      a
```

The left, right and full joins are collectively know as **outer joins**. When a row does not match in an outer join, the new variables are filled in with missing values.

1.4 Observations

While mutating joins are primarily used to add new variables, they can also generate new observations. If a match is not unique, a join will add all possible combinations (the Cartesian product) of the matching observations:

```
> (df1 = data_frame(x = c(1,1,2), y = 1:3))
Source: local data frame [3 x 2]

      x     y
  (dbl) (int)
1     1     1
2     1     2
3     2     3

> (df2 = data_frame(x = c(1,1,2), z = c("a", "b", "a")))
Source: local data frame [3 x 2]

      x     z
  (dbl) (chr)
1     1     a
2     1     b
3     2     a

> df1 %>% left_join(df2)
Joining by: "x"
Source: local data frame [5 x 3]

      x     y     z
  (dbl) (int) (chr)
1     1     1     a
2     1     1     b
3     1     2     a
4     1     2     b
5     2     3     a
```

1.5 Filtering joins

Filtering joins match observations in the same way as mutating joins, but affect the observations, not the variables. There are two types in filtering joins:

- semi_join(x, y) **keeps** all observations in x that have a match in y
- anti_join(x, y) **drops** all observations in x that have a match in y

These are most useful for diagnosing join mismatches. For example, there are many flights in the nycflights13 dataset that do not have a matching tail number in the planes table:

```
> flights %>%
+     anti_join(planes, by = "tailnum") %>%
+     count(tailnum, sort = TRUE)
Source: local data frame [722 x 2]

   tailnum     n
     (chr) (int)
1           2512
2    N725MQ   575
3    N722MQ   513
4    N723MQ   507
5    N713MQ   483
6    N735MQ   396
7    N0EGMQ   371
8    N534MQ   364
9    N542MQ   363
10   N531MQ   349
..      ...   ...
```

```
> flights %>%
+     semi_join(planes, by = "tailnum") %>%
+     count(tailnum, sort = TRUE)
Source: local data frame [3,322 x 2]

   tailnum     n
     (chr) (int)
1   N711MQ   486
2   N258JB   427
3   N298JB   407
4   N353JB   404
5   N351JB   402
6   N328AA   393
7   N228JB   388
8   N338AA   388
9   N327AA   387
10  N335AA   385
..     ...   ...
```

1.6 Set operations

The final type of two-table verb is set operations. These expect the x and y inputs to have the same variables, and treat the observations like sets.

- `intersect(x, y)` returns only observations in both x and y
- `union(x, y)` returns unique observations in x and y
- `setdiff(x, y)` returns observations in x, but not in y

Provided this simple data:

```
> (df1 = data_frame(x = 1:2, y = c(1L, 1L)))
Source: local data frame [2 x 2]

      x     y
  (int) (int)
1     1     1
2     2     1


> (df2 = data_frame(x = 1:2, y = 1:2))
Source: local data frame [2 x 2]

      x     y
  (int) (int)
1     1     1
2     2     2
```

the four possible operations are

```
> intersect(df1, df2)
Source: local data frame [1 x 2]

      x     y
  (int) (int)
1     1     1
> union(df1, df2)
Source: local data frame [3 x 2]

      x     y
  (int) (int)
1     1     1
2     2     1
3     2     2
> setdiff(df1, df2)
Source: local data frame [1 x 2]
```

```
      x     y
  (int) (int)
1     2     1
> setdiff(df2, df1)
Source: local data frame [1 x 2]

      x     y
  (int) (int)
1     2     2
```

NOTE: there are straightforward SQL equivalent operations:

R	SQL
inner_join()	SELECT * FROM x JOIN y ON x.a = y.a
left_join()	SELECT * FROM x LEFT JOIN y ON x.a = y.a
right_join()	SELECT * FROM x RIGHT JOIN y ON x.a = y.a
full_join()	SELECT * FROM x FULL JOIN y ON x.a = y.a
semi_join()	SELECT * FROM x WHERE EXISTS (SELECT 1 FROM y WHERE x.a = y.a)
anti_join()	SELECT * FROM x WHERE NOT EXISTS (SELECT 1 FROM y WHERE x.a = y.a)
intersect(x, y)	SELECT * FROM x INTERSECT SELECT * FROM y
union(x, y)	SELECT * FROM x UNION SELECT * FROM y
setdiff(x, y)	SELECT * FROM x EXCEPT SELECT * FROM y

2 Practice

Can you join the `flights` dataset with the `weather` dataset and run regressions between `dep_delay` in `flights` and variables in `weather` that you think are critical for the departure delay (e.g. `temp`, `wind_dir`, `humid`, `wind_gust`, etc.) f the flight?