dplyr Package

By Connor Humphray, Marnie Guy and Yuuki Hosokawa

An Introduction to Dplyr

Dpylr is a fast and powerful R programming package used to manipulate datasets. It makes working with data easier by constraining the options and providing simple functions to use.

It was released open-source January 7th, 2014 primarily authored by Hadley Wickham. The package includes **five** datasets to perform operations on: band_instruments, band_instruments2, band_members, starwars, and storms. The key object is a representation of a tabular data structure and it currently supports: data frames, data tables, Redshift, MySQL, Bigquery, and MonetDB

```
# If installing for the first time, use the install.packages function below
# install.packages("dplyr")
library(dplyr) # Loads the dplyr packages
```

Data Frames

The term *data frame* is a key data structure in statistics and R. The basic structure contains **one** observation per row and each column represents a variable of that observation. To better manage, interpret and visualize data sets we can use the **dplyr** package. This package makes tedious operations using base R easier to complete at higher efficiency and readability.

Columns

Rows

Selecting columns & filtering rows

There are many useful functions within dplyr. Below are some of the most commonly used and their usage. These functions are the five primary verbs that can be used in conjunction with the dozens of other functions within **dplyr** to “help you solve the most common data manipulation challenges”.

select() - Subset a dataframe by its columns.

filter() - Extract rows from a dataframe

mutate() - Creates new columns based on existing ones

arrange() - Sorts/reorders the rows in the data frame by the value of given columns

summarise() - Collapses given existing values to a single-row summary.

```r
data(mtcars)
# Loads example data of motor trend car road tests from 1974 Motor Trend US magazine
```

```r
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```r
#select(.data, ...) takes a data frame argument and variable names (...) to select a range of variables
```

```r
select(mtcars, hp, mpg, wt)
```

```
##                      hp  mpg    wt
## Mazda RX4           110 21.0 2.620
## Mazda RX4 Wag       110 21.0 2.875
## Datsun 710           93 22.8 2.320
## Hornet 4 Drive      110 21.4 3.215
## Hornet Sportabout   175 18.7 3.440
## Valiant             105 18.1 3.460
## Duster 360          245 14.3 3.570
## Merc 240D            62 24.4 3.190
## Merc 230             95 22.8 3.150
## Merc 280            123 19.2 3.440
## Merc 280C           123 17.8 3.440
```

```
filter(mtcars, hp > 100, wt < 3)
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
# If we want to subset specific rows, we use filter(.data, ...) and it returns the variables that satisfies the c
ondition. In this case we select cars with horsepower more than 100 and the weight variable is less than 3
```

```
summarise(mtcars, avg_hp = mean(hp), min_hp = min(hp),
          max_hp = max(hp), med_hp = median(hp), IQG_hp = IQR(hp))
```

```
##     avg_hp min_hp max_hp med_hp IQG_hp
## 1 146.6875     52    335    123   83.5
```

```
# summarize(.data, ...) is useful for generating summary statistics that can be given with any variables

# If we want the average, minimum, maximum, median and interquartile range summary statistics for one variable (h
orsepower) we use the above.
```

Pipe operator

The pipe operator **%>%** is used to take the output of one function and use it in the next. This new syntax is useful because it's easier to read, write and tells the story of your analysis. Using this operator allows us to chain multiple dplyr commands together.

```
# Instead of this
mtcars_gears <- group_by(mtcars, gear)
# You can use pipe %>% and do
mtcars %>% group_by(gear)
```

NA values

These are examples of functions that are used in order to replace or manipulate values that are **Not Available** (NA). This is helpful as it allows positions in a table to be left without a data point without it changing the columns and row numbers.

na_if() - Converts the value to NA

is.na() - Function that returns true or false for each value in a data set(whether it contains NA not). It can be used to find missing values

replace_na() - Used to replace NAs with specific values

na.omit() - Deletes the na values in your dataset

drop_na() - Drops any rows that contain na(missing values)

Instead of NA values which are missing values in the dataset, the value returned is the symbol NaN which represents Not a Number.

Advantages

- **Speed** - Pre-dplyr, RPostgreSQL was much slower than our current package dylpr as it has the most important parts of its code written in Rcpp. This is a package that integrates R with C++ to accelerate computations. Although there is evidence to suggest that data.table package may be quicker.

- **Syntax** - The code used in dplyr is relatively simple and easy to follow, as well as the inclusion of the pipe function which streamlines the code.

- **External databases** - Instead of using a package like RMySQL in order to access a database dplyr has this access built in. Another example of good integration with databases is the collect() function that is delayed, by dplyr, until the last minute in order to reduce the number of requests that are sent to the database, reducing the stress and time it takes to interact.

Disadvantages

- GGplot2 can be used to in the sameway as the %>% function and i predates it. The advantage of ggplot2 over dylpr is that ggplot2 is able to be manipulated more as it doesn't look for the pipe name within code that can be non-standard evaluation. Therefore, it's much harder to hack around the dplyr when using your own code.

- There is also evidence to suggest that data.table package may be faster than dplyr when the number of groups increase. It also has a high overlap in functions.

Integration

To use Tidyr (used for making code neater) with dplyr you need to take advantage of some of the reshaping functions.

Pivot_wider()/_longer() - These can be used in order to create new columns allowing for the structure of the table to be manipulated from tidyr to dplyr and vice versa

Ggplot can also be used easily to visualise the data used in dplyr.

Grouping data

The **group_by()** function is a very useful part of **dplyr** as it allows for very specific and targeted computation. The constraints passed as arguments of the function can access various properties of the group to be further analyzed or visualized (ex. summarize()). There are other functions that allow you to manipulate it more such as:

cur_data() - Returns the current group, removing grouping variables

ungroup() - Removes all grouping variables

group_keys() - Shows the underlying group data, one row for each group and a column for each grouping variable

cur_group_id() - The current group gets a unique numeric identifier
cur_group_rows() - Row indices for the current group are returned

Joining Tables

bind_cols() returns tables pasted next to each other whilst the bind_cols(...) variation combines the two tables into one

bind_rows() returns tables joined underneath each other or in other words by rows

intersect(x,y,...) joins rows and removes duplicates

set_diff(x,y,...) subsets the first parameter (requires same columns)

union(x,y,...) combines vectors and operates row-wise

A	B	C
1	4	5
2	3	5
5	4	3

Table 1

A	B	D
5	4	5
2	2	5
5	4	1

Table 2

A	B	C	D
1	4	5	5
2	3	5	5
5	4	3	1

Join functions attach columns to tables in different ways for example: left_join(x,y, by = “A”)

There is also a right_join(x,y, by= “A”) which appends x to y, inner_join() which only adds matching rows and a full_join() which joins all the data

By = c allows you to specify the columns to be joined at

semi _join(x,y, by= “A”,...) gives you x which match y   anti_join(x,y, by= “A”,...) gives you x with no match with y

Have a great rest of the day!
Any questions?