

# R dplyr

James McDermott

University of Galway



OLLSCOIL NA GAILLIMHE  
UNIVERSITY OF GALWAY



# Programming and Tools for Artificial Intelligence

Dr James McDermott

dplyr

dplyr is a package for relational operations on data. That is, it does stuff similar to SQL, which many students will be familiar with (also comparable to Excel and Pandas). In particular:

- `filter` (choose rows, like SQL `where`)
- `arrange` (sort rows)
- `select` (choose columns, like SQL `select`)
- `mutate` (add columns)
- `summarise` (condense multiple values)
- `sample_n`, `sample_frac` (for taking a quick look at a sub-sample, see also `head`)
- `inner_join`, `left_join`, `right_join`, `full_join` (join two tables, like SQL `join`)

# dplyr and the pipe operator

All the dplyr verbs (`select`, etc.) have three things in common (from <https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html>):

- 1 The first argument is a data frame [actually, a tibble].
- 2 The subsequent arguments describe what to do with the data frame.  
You can refer to columns in the data frame directly without using `$`.
- 3 The result is a new data frame

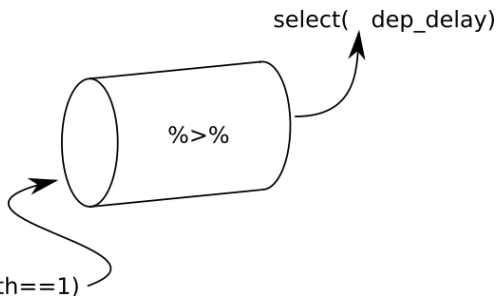
Therefore, it is natural to *chain* operations. That is where the *pipe* operator comes in.

# The pipe %>%



- In Unix, the *pipe* symbol `|` is used to pass data from one command to another, e.g. `ls | grep -v "#"`
- In `magrittr`, the pipe `%>%` passes the output of its left-hand side to become the first argument of its right-hand side.

# The pipe %>%



```
filter(d, month==1) %>% select(dep_delay)
```

# The pipe %>%

In R, special operators are often named with double % symbols. The *pipe* operator %>% is an example.

`t %>% f()` means precisely `f(t)`, i.e. the output of the LHS becomes the first argument of the RHS.

It is useful to avoid complicated nested expressions:

`t %>% f("abc") %>% g("x", "y")` is easier to read than

`g(f(t, "abc"), "x", "y")`

(isn't it?)

## Example: NYC Flights

```
# uncomment if not already installed
```

```
# install.packages("nycflights13")
```

```
library(nycflights13)
```

```
library(tidyverse)
```

```
## -- Attaching packages ----- tidyverse
```

```
## v ggplot2 3.2.1      v purrr  0.3.2
```

```
## v tibble  2.1.3      v dplyr  0.8.3
```

```
## v tidyr   1.0.0      v stringr 1.4.0
```

```
## v readr   1.3.1      v forcats 0.4.0
```

```
## -- Conflicts ----- tidyverse_conflict
```

```
## x dplyr::filter() masks stats::filter()
```

```
## x dplyr::lag()     masks stats::lag()
```



# Take a quick look at data

```
flights
```

```
## # A tibble: 336,776 x 19
```

```
##   year month   day dep_time sched_dep_time dep_delay arr_
```

```
##   <int> <int> <int>   <int>         <int>         <dbl>   <
```

```
## 1  2013     1     1     517           515         2
```

```
## 2  2013     1     1     533           529         4
```

```
## 3  2013     1     1     542           540         2
```

```
## 4  2013     1     1     544           545        -1
```

```
## 5  2013     1     1     554           600        -6
```

```
## 6  2013     1     1     554           558        -4
```

```
## 7  2013     1     1     555           600        -5
```

```
## 8  2013     1     1     557           600        -3
```

```
## 9  2013     1     1     557           600        -3
```

```
## 10 2013     1     1     558           600        -2
```

```
## # ... with 336,766 more rows, and 12 more variables: sched_
```

```
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <
```

# filter

Let's see just flights on Jan 1:

```
flights %>% filter(month == 1, day == 1)
```

```
## # A tibble: 842 x 19
```

```
##   year month   day dep_time sched_dep_time dep_delay arr_
```

```
##   <int> <int> <int>   <int>         <int>         <dbl> <
```

```
## 1  2013     1     1     517           515         2
```

```
## 2  2013     1     1     533           529         4
```

```
## 3  2013     1     1     542           540         2
```

```
## 4  2013     1     1     544           545        -1
```

```
## 5  2013     1     1     554           600        -6
```

```
## 6  2013     1     1     554           558        -4
```

```
## 7  2013     1     1     555           600        -5
```

```
## 8  2013     1     1     557           600        -3
```

```
## 9  2013     1     1     557           600        -3
```

```
## 10 2013     1     1     558           600        -2
```

```
## # ... with 832 more rows, and 12 more variables: sched_arr_
```

# filter

Remember, this is internally translated to:

```
filter(flights, month == 1, day == 1)
```

```
## # A tibble: 842 x 19
```

```
##   year month   day dep_time sched_dep_time dep_delay arr_
```

```
##   <int> <int> <int>   <int>         <int>         <dbl> <
```

```
## 1  2013     1     1     517           515         2
```

```
## 2  2013     1     1     533           529         4
```

```
## 3  2013     1     1     542           540         2
```

```
## 4  2013     1     1     544           545        -1
```

```
## 5  2013     1     1     554           600        -6
```

```
## 6  2013     1     1     554           558        -4
```

```
## 7  2013     1     1     555           600        -5
```

```
## 8  2013     1     1     557           600        -3
```

```
## 9  2013     1     1     557           600        -3
```

```
## 10 2013     1     1     558           600        -2
```

```
## # ... with 832 more rows, and 12 more variables: sched_arr_
```

# filter

Which flight departure was delayed the longest? We can use filter again:

```
flights %>% filter(dep_delay == max(dep_delay, na.rm=TRUE))

## # A tibble: 1 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>   <int>         <int>       <dbl>   <int>
## 1  2013     1     9     641           900       1301     1
## # ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dtm>
```

Notice that `na.rm=TRUE` *inside* `max()` is essential: consider `max(flights$dep_delay)` to see why.

# Syntax notes

- 1 `na.rm` argument
- 2 We can refer to columns with no special syntax (not even quotes)
- 3 Remember `==` for equality (I put spaces), but `=` for passing keyword arguments (I don't put spaces, as in Python).

# filter examples

Boolean AND: use comma as we already saw

```
flights %>% filter(month == 1, day == 1)
```

```
## # A tibble: 842 x 19
```

```
##   year month   day dep_time sched_dep_time dep_delay arr_
```

```
##   <int> <int> <int>   <int>         <int>         <dbl> <
```

```
## 1  2013     1     1     517           515         2
```

```
## 2  2013     1     1     533           529         4
```

```
## 3  2013     1     1     542           540         2
```

```
## 4  2013     1     1     544           545        -1
```

```
## 5  2013     1     1     554           600        -6
```

```
## 6  2013     1     1     554           558        -4
```

```
## 7  2013     1     1     555           600        -5
```

```
## 8  2013     1     1     557           600        -3
```

```
## 9  2013     1     1     557           600        -3
```

```
## 10 2013     1     1     558           600        -2
```

```
## # ... with 832 more rows, and 12 more variables: sched_arr_
```

# filter examples

Boolean OR: use |

```
flights %>% filter(month == 1 | month == 12)
```

```
## # A tibble: 55,139 x 19
```

```
##   year month   day dep_time sched_dep_time dep_delay arr_
```

```
##   <int> <int> <int>   <int>         <int>         <dbl>   <
```

```
## 1  2013     1     1     517           515           2
```

```
## 2  2013     1     1     533           529           4
```

```
## 3  2013     1     1     542           540           2
```

```
## 4  2013     1     1     544           545          -1
```

```
## 5  2013     1     1     554           600          -6
```

```
## 6  2013     1     1     554           558          -4
```

```
## 7  2013     1     1     555           600          -5
```

```
## 8  2013     1     1     557           600          -3
```

```
## 9  2013     1     1     557           600          -3
```

```
## 10 2013     1     1     558           600          -2
```

```
## # ... with 55,129 more rows, and 12 more variables: sched_a
```

# filter examples

%in% operator does the same as above:

```
flights %>% filter(month %in% c(1, 12))
```

```
## # A tibble: 55,139 x 19
```

```
##   year month   day dep_time sched_dep_time dep_delay arr_
```

```
##   <int> <int> <int>   <int>         <int>         <dbl> <
```

```
## 1  2013     1     1     517           515         2
```

```
## 2  2013     1     1     533           529         4
```

```
## 3  2013     1     1     542           540         2
```

```
## 4  2013     1     1     544           545        -1
```

```
## 5  2013     1     1     554           600        -6
```

```
## 6  2013     1     1     554           558        -4
```

```
## 7  2013     1     1     555           600        -5
```

```
## 8  2013     1     1     557           600        -3
```

```
## 9  2013     1     1     557           600        -3
```

```
## 10 2013     1     1     558           600        -2
```

```
## # ... with 55,129 more rows, and 12 more variables: sched_a
```



# arrange

dplyr becomes like a programmatic interface to Excel, e.g. sort-by-column:

```
arrange(flights, dep_delay) # sort-by-column
```

```
## # A tibble: 336,776 x 19
```

```
##   year month   day dep_time sched_dep_time dep_delay arr_
##   <int> <int> <int>   <int>         <int>         <dbl>  <
```

```
## 1  2013    12     7    2040           2123         -43
```

```
## 2  2013     2     3    2022           2055         -33
```

```
## 3  2013    11    10    1408           1440         -32
```

```
## 4  2013     1    11    1900           1930         -30
```

```
## 5  2013     1    29    1703           1730         -27
```

```
## 6  2013     8     9     729            755         -26
```

```
## 7  2013    10    23    1907           1932         -25
```

```
## 8  2013     3    30    2030           2055         -25
```

```
## 9  2013     3     2    1431           1455         -24
```

```
## 10 2013     5     5     934            958         -24
```

```
## # ... with 336,766 more rows, and 12 more variables: sched_
```

# arrange

Descending order:

```
arrange(flights, desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
```

```
##   year month   day dep_time sched_dep_time dep_delay arr_
```

```
##   <int> <int> <int>   <int>         <int>         <dbl>  <
```

```
## 1  2013     1     9     641           900       1301
```

```
## 2  2013     6    15    1432          1935       1137
```

```
## 3  2013     1    10    1121          1635       1126
```

```
## 4  2013     9    20    1139          1845       1014
```

```
## 5  2013     7    22     845          1600       1005
```

```
## 6  2013     4    10    1100          1900        960
```

```
## 7  2013     3    17    2321           810        911
```

```
## 8  2013     6    27     959          1900        899
```

```
## 9  2013     7    22    2257           759        898
```

```
## 10 2013    12     5     756          1700        896
```

```
## # ... with 336,766 more rows, and 12 more variables: sched_
```

## select chooses columns

```
select(flights, day, month, year)
```

```
## # A tibble: 336,776 x 3
```

```
##       day month  year
```

```
##   <int> <int> <int>
```

```
## 1      1      1  2013
```

```
## 2      1      1  2013
```

```
## 3      1      1  2013
```

```
## 4      1      1  2013
```

```
## 5      1      1  2013
```

```
## 6      1      1  2013
```

```
## 7      1      1  2013
```

```
## 8      1      1  2013
```

```
## 9      1      1  2013
```

```
## 10     1      1  2013
```

```
## # ... with 336,766 more rows
```

Notice that `select` also re-orders columns according to the arguments.

## select chooses columns

Select all columns from year to day (behind the scenes, the columns are *numbers*):

```
select(flights, year:day)
```

```
## # A tibble: 336,776 x 3
```

```
##   year month   day
```

```
##   <int> <int> <int>
```

```
## 1  2013     1     1
```

```
## 2  2013     1     1
```

```
## 3  2013     1     1
```

```
## 4  2013     1     1
```

```
## 5  2013     1     1
```

```
## 6  2013     1     1
```

```
## 7  2013     1     1
```

```
## 8  2013     1     1
```

```
## 9  2013     1     1
```

```
## 10 2013     1     1
```

# Using select to deselect

```
select(flights, -(month:minute))
```

```
## # A tibble: 336,776 x 2
##   year time_hour
##   <int> <dtm>
## 1  2013 2013-01-01 05:00:00
## 2  2013 2013-01-01 05:00:00
## 3  2013 2013-01-01 05:00:00
## 4  2013 2013-01-01 05:00:00
## 5  2013 2013-01-01 06:00:00
## 6  2013 2013-01-01 05:00:00
## 7  2013 2013-01-01 06:00:00
## 8  2013 2013-01-01 06:00:00
## 9  2013 2013-01-01 06:00:00
## 10 2013 2013-01-01 06:00:00
## # ... with 336,766 more rows
(Note we can't mix positive and negative)
```

# More ways to select columns

```
select(flights, starts_with("d"))
```

```
## # A tibble: 336,776 x 5
```

```
##       day dep_time dep_delay dest  distance
```

```
##    <int>    <int>    <dbl> <chr>    <dbl>
```

```
##  1      1      517         2 IAH      1400
```

```
##  2      1      533         4 IAH      1416
```

```
##  3      1      542         2 MIA      1089
```

```
##  4      1      544        -1 BQN      1576
```

```
##  5      1      554        -6 ATL       762
```

```
##  6      1      554        -4 ORD       719
```

```
##  7      1      555        -5 FLL      1065
```

```
##  8      1      557        -3 IAD       229
```

```
##  9      1      557        -3 MCO       944
```

```
## 10     1      558        -2 ORD       733
```

```
## # ... with 336,766 more rows
```

# mutate

```
select(flights, year:day, ends_with("delay"),
       distance, air_time) %>%
  mutate(gain=dep_delay - arr_delay) %>%
  mutate(speed=distance / air_time * 60)
```

```
## # A tibble: 336,776 x 9
```

##	year	month	day	dep_delay	arr_delay	distance	air_time	
##	<int>	<int>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	
##	1	2013	1	1	2	11	1400	227
##	2	2013	1	1	4	20	1416	227
##	3	2013	1	1	2	33	1089	160
##	4	2013	1	1	-1	-18	1576	183
##	5	2013	1	1	-6	-25	762	116
##	6	2013	1	1	-4	12	719	150
##	7	2013	1	1	-5	19	1065	158
##	8	2013	1	1	-3	-14	229	53
##	9	2013	1	1	-3	-8	944	140

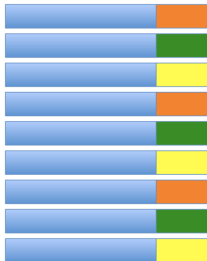
Remember none of these operations change the tibble itself, just return a new one. So we may decide to save the result in a new variable.

```
sml <- select(flights, year:day, ends_with("delay"),
              distance, air_time) %>%
  mutate(gain=dep_delay - arr_delay) %>%
  mutate(speed=distance / air_time * 60)
```



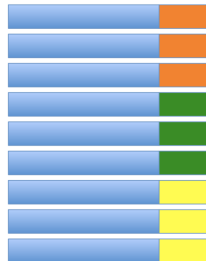
# group\_by

Original data frame



group\_by()

Grouped data frame



summarise()



(R4DS)

## group\_by and summarise go well together

```
group_by(flights, carrier) %>%  
  summarise(dep_delay=mean(dep_delay, na.rm=TRUE))
```

```
## # A tibble: 16 x 2  
##   carrier dep_delay  
##   <chr>      <dbl>  
## 1 9E         16.7  
## 2 AA          8.59  
## 3 AS          5.80  
## 4 B6         13.0  
## 5 DL          9.26  
## 6 EV         20.0  
## 7 F9         20.2  
## 8 FL         18.7  
## 9 HA          4.90  
## 10 MQ        10.6  
## 11 OO        12.6
```

# A bigger example: who

who is a dataset from the World Health Organisation which needs a lot of cleaning:

```
who %>%  
  # gather many columns to 1  
  gather(key, value, new_sp_m014:newrel_f65,  
         na.rm = TRUE) %>%  
  # fix inconsistent spelling  
  mutate(key = stringr::str_replace(key, "newrel", "new_rel"))  
  # split eg "new_sp_m014" -> "new", "sp", "m014"  
  separate(key, c("new", "var", "sexage")) %>%  
  # remove redundant/unneeded columns  
  select(-new, -iso2, -iso3) %>%  
  # split eg "m014" -> "m", "014"  
  separate(sexage, c("sex", "age"), sep = 1)
```

```
## # A tibble: 76,046 x 6
```

# A bigger example: `who`

See <https://r4ds.had.co.nz/tidy-data.html> for detailed explanation.

**Exercise:** look at the output of each step of this transformation, starting at `who` itself, to understand the need for the next.

# Some more handy functions (some from dplyr)

- Offset a vector of values: `lead` and `lag`
- Cumulative calculations: `cumsum`, `cummax`, etc.
- Where does each value come in a sort? `min_rank`
- Counts: `n`, `n_distinct`

# Functional programming in R

- dplyr and the pipe are already examples of functional programming! E.g. all these operations don't change their input, just return a new version.
- Our friend map also exists in R. It makes a list.
- map\_dbl and friends may be more useful since they return vectors. Compare map and map\_dbl in the following.

```
d = 1:5
```

```
# R uses the opposite argument ordering, compared to Python  
map(d, sqrt)
```

```
d = 1:5
```

```
map_dbl(d, sqrt)
```

# Functional programming in R

```
d %>% map_dbl(sqrt) # equivalent, using pipe
```

# Functional programming in R

`map` and friends come from the `purrr` package, well-documented here:  
<https://r4ds.had.co.nz/iteration.html#the-map-functions>

- The Joy of Functional Programming ACM Tech Talk webcast with Hadley Wickham can be viewed here:  
<https://learning.acm.org/techtalks/functionalprogramming>  
(Prerequisites: basic R, tibbles, distinction between lists, vectors, dataframes)



# Summary

- tidy data: columns are variables, rows are observations
- tibbles
- pipe %>%
- verbs including `select`, `filter`, `mutate`, `arrange`, `rename`, `gather`, `spread`

Let's look at a cheatsheet for `dplyr`:

<https://rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>

# Exercises

- Exercise 1: Our sort times data is available in tidy format as `sort_times_tidy.csv`. Use `group_by` and `summarise` to get the mean and the standard deviation for each `n`, and then for each `run_number`.
- A dataset of characters in *Star Wars* is available as `dplyr::starwars`.  
Exercise 2: Find all the human females. Exercise 3: Find the characters who are human *or* Wookiee. Exercise 4: Find the shortest character. Hint: recall we might need `na.rm`. Exercise 5: Add a new column called BMI giving the body mass index, where the formula is  $BMI = m/h^2$  for mass  $m$  in kg and height  $h$  in metres. [https://en.wikipedia.org/wiki/Body\\_mass\\_index](https://en.wikipedia.org/wiki/Body_mass_index). Exercise 6: Which character has the highest BMI?

# Solutions

# Exercise 1

```
d <- read_csv("data/sort_times_tidy.csv")
```

```
## Parsed with column specification:
```

```
## cols(
```

```
##   n = col_double(),
```

```
##   run_number = col_double(),
```

```
##   run_time = col_double()
```

```
## )
```

```
d %>% group_by(n) %>% summarise(mean_run_time=mean(run_time),
```

```
## # A tibble: 10 x 3
```

```
##           n mean_run_time sd_run_time
```

```
##      <dbl>      <dbl>      <dbl>
```

```
## 1 1000000      0.105      0.00654
```

```
## 2 2000000      0.209      0.0269
```

```
## 3 3000000      0.334      0.0387
```

```
## 4 4000000      0.473      0.0442
```

```
## 5 5000000      0.574      0.0308
```

# Exercise 1

Notice that the mean and stddev for  $n = 7$  million are anomalously high. One way this could occur is if our computer had a spike in CPU usage during the experiment, e.g. due to a browser loading a video.

# Exercise 1

```
d %>% group_by(run_number) %>%  
  summarise(mean_run_time=mean(run_time),  
            sd_run_time=sd(run_time))
```

```
## # A tibble: 5 x 3
```

```
##   run_number mean_run_time sd_run_time  
##      <dbl>         <dbl>         <dbl>  
## 1         0         0.754         0.512  
## 2         1         0.644         0.368  
## 3         2         0.604         0.353  
## 4         3         0.648         0.369  
## 5         4         0.678         0.416
```

No major anomalies this time.

## Exercise 2

```
sw <- dplyr::starwars
sw %>% filter(species == "Human", gender == "female") # human

## # A tibble: 9 x 13
##   name height mass hair_color skin_color eye_color birth_
##   <chr>   <int> <dbl> <chr>         <chr>         <chr>         <chr>
## 1 Leia~   150    49 brown         light         brown
## 2 Beru~   165    75 brown         light         blue
## 3 Mon ~   150    NA auburn        fair          blue
## 4 Shmi~   163    NA black        fair          brown
## 5 Cordé   157    NA brown        light         brown
## 6 Dormé   165    NA brown        light         brown
## 7 Joca~   167    NA white        fair          blue
## 8 Rey     NA     NA brown        light         hazel
## 9 Padm~   165    45 brown        light         brown
## # ... with 5 more variables: homeworld <chr>, species <chr>
## #   vehicles <list>, starships <list>
```

## Exercise 3

```
sw %>% filter(species == "Human" | species == "Wookiee") # human and wookiee
## # A tibble: 37 x 13
##   name      height  mass hair_color skin_color eye_color birth_year
##   <chr>    <int> <dbl> <chr>      <chr>      <chr>      <dbl>
## 1 Luke~      172    77 blond      fair        blue        19
## 2 Dart~      202   136 none       white       yellow      15
## 3 Leia~      150    49 brown      light       brown       19
## 4 Owen~      178   120 brown, gr~ light       blue        57
## 5 Beru~      165    75 brown      light       blue        54
## 6 Bigg~      183    84 black      light       brown       27
## 7 Obi-~      182    77 auburn, w~ fair        blue-gray   57
## 8 Anak~      188    84 blond      fair        blue        41
## 9 Wilh~      180    NA auburn, g~ fair        blue        58
## 10 Chew~     228   112 brown      unknown     blue        19
## # ... with 27 more rows, and 5 more variables: homeworld <chr>,
## #   species <chr>, films <list>, vehicles <list>, starships <list>
```



## Exercise 4

```
sw %>% filter(height == max(height, na.rm=TRUE))

## # A tibble: 1 x 13
##   name height mass hair_color skin_color eye_color birth_
##   <chr>  <int> <dbl> <chr>      <chr>      <chr>      <chr>
## 1 Yara~   264    NA none        white       yellow
## # ... with 5 more variables: homeworld <chr>, species <chr>
## #   vehicles <list>, starships <list>
```

## Exercise 5

```
# NB convert height from cm to metres before squaring  
BMI <- function(h, m) {m / (h / 100)^2}  
sw <- sw %>% mutate(bmi=BMI(height, mass))
```

## Exercise 6

```
sw %>% filter(bmi == max(bmi, na.rm=TRUE))

## # A tibble: 1 x 14
##   name height mass hair_color skin_color eye_color birth_
##   <chr>  <int> <dbl> <chr>      <chr>      <chr>      <chr>
## 1 Jabb~    175  1358 <NA>      green-tan~ orange
## # ... with 6 more variables: homeworld <chr>, species <chr>
## #   vehicles <list>, starships <list>, bmi <dbl>
```