Working with Data: Day 1&2

May 3-4, 2018

Outline

- review *dplyr* operations
- more on grouped summaries
- grouped mutation
- window functions
- operations on two tables
- data manipulation in base R
- tidy data

Review: *dplyr* verbs

- `select`: view only some variables (columns)
- `filter`: choose observations by their values
- `arrange`: order observations (rows)
- `mutate`: create new variables
- `summarise`: calculate a summary of many variable values

Review: *dplyr*

Each verb works similarly:

- input data frame in the first argument
- other arguments can refer to variables as if they were local objects
- output is another data frame
- use `%>%` to "pipe" the left-hand output as the right-hand's input

Grouped summaries: Review and counting

Summaries are functions that given a set of values as an input produce as result a single scalar. (E.g., mean, median, etc.) In *dplyr*, the `summarise` verb takes as additional arguments the summary functions whose result will be returned as a column in the result.

```
library(dplyr)
summarise(mtcars, mpg=mean(mpg), wt=median(wt))
```

```
##        mpg    wt
## 1 20.09062 3.325
```

```
summarise(mtcars, mpg=mean(mpg), wt=median(wt), rat=mpg/wt)
```

```
##        mpg    wt      rat
## 1 20.09062 3.325 6.042293
```

Note that naming the argument also names the column in the result, and that a summary can refer to any summaries that preceded it in the argument order.

When combined with a grouping verb `group_by`, summaries are calculated per-group:

```
mtcars %>%
  group_by(cyl, am) %>%
  summarise(mpg = mean(mpg), wt=median(wt))
```

```
## # A tibble: 6 x 4
## # Groups:   cyl [?]
##     cyl    am   mpg    wt
##   <dbl> <dbl> <dbl> <dbl>
## 1    4.    0.  22.9  3.15
## 2    4.    1.  28.1  2.04
## 3    6.    0.  19.1  3.44
## 4    6.    1.  20.6  2.77
## 5    8.    0.  15.0  3.81
## 6    8.    1.  15.4  3.37
```

Note that the grouping variable are included in the result.

In addition to the standard statistical and summaries familiar from base R, *dplyr* provides a few unique ones:

- `n()`: the number of observations
- `n_distinct(x)`: the number of unique values of variable `x`
- `first(x)`, `last(x)`, and `nth(x, n)`: the first (or last, or `n`-th) value of variable `x`

For instance, to count the number of cars of each cylinder type:

```
mtcars %>%
  group_by(cyl) %>%
  summarise(n = n())
```

```
## # A tibble: 3 x 2
##     cyl     n
##   <dbl> <int>
## 1    4.    11
## 2    6.     7
## 3    8.    14
```

Counting is so useful that there are two shortcuts:

- `tally`: equivalent to `summarise(n = n())`
- `count`: `group_by` + `tally`

```
mtcars %>%
  group_by(cyl) %>%
  tally
```

```
## # A tibble: 3 x 2
##     cyl     n
##   <dbl> <int>
## 1    4.    11
## 2    6.     7
## 3    8.    14
```

```
mtcars %>%
  count(cyl)
```

```
## # A tibble: 3 x 2
##     cyl     n
##   <dbl> <int>
## 1    4.    11
```

```
## 2    6.     7
## 3    8.    14
```

Creating group-wise variables

What happens when we use `mutate` on grouped data? The new variable(s) are calculated per-group. For instance, to calculate the Z-score of fuel-efficiency within each cylinder group:

```
mtcars %>%
  group_by(cyl) %>%
  mutate(z = (mpg - mean(mpg)) / sd(mpg))
```

```
## # A tibble: 32 x 12
## # Groups:   cyl [3]
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  21.0    6.  160.  110.  3.90  2.62  16.5    0.    1.    4.    4.
##  2  21.0    6.  160.  110.  3.90  2.88  17.0    0.    1.    4.    4.
##  3  22.8    4.  108.   93.  3.85  2.32  18.6    1.    1.    4.    1.
##  4  21.4    6.  258.  110.  3.08  3.22  19.4    1.    0.    3.    1.
##  5  18.7    8.  360.  175.  3.15  3.44  17.0    0.    0.    3.    2.
##  6  18.1    6.  225.  105.  2.76  3.46  20.2    1.    0.    3.    1.
##  7  14.3    8.  360.  245.  3.21  3.57  15.8    0.    0.    3.    4.
##  8  24.4    4.  147.   62.  3.69  3.19  20.0    1.    0.    4.    2.
##  9  22.8    4.  141.   95.  3.92  3.15  22.9    1.    0.    4.    2.
## 10  19.2    6.  168.  123.  3.92  3.44  18.3    1.    0.    4.    4.
## # ... with 22 more rows, and 1 more variable: z <dbl>
```

Window functions

Window functions work on entire vectors, like aggregates, but return another vector of the same length instead of a scalar. For example, think of producing a ranking of cars by weight: you need the weights of all cars, and the result is a vector from 1 to the number of cars. There are many different window functions available in R and *dplyr*, but they fall into one of the following categories:

- ranking and ordering (`row_number`/`min_rank`/`dense_rank`,`ntile`)
- cumulative aggregates (e.g., cumulative sum `cumsum`)
- rolling aggregates calculate an aggregate over a fixed-width window (e.g., mean revenue over the last 12 months – *zoo*, *RcppRoll*)
- offsets: functions that depend on the preceding or following values in the input vector (`lag` and `lead`)

Ranking functions return the "place" of the variable value if all the values were sorted from smallest to largest. For example, given `x`:

```
x <- c(22, 10, 11, 5, 7)
```

There are five unique values. The very first one, 22, is the largest one, so it gets the largest rank (i.e., 5). The next one, 10, is the third smallest, so it gets rank 3. The smallest value, 5, gets rank 1 and because it's in fourth position in `x`, the fourth position of the ranking vector will be 1:

```
min_rank(x)
```

```
## [1] 5 3 4 1 2
```

The ranking functions work so that small values get low ranks. (I.e., the smallest value gets rank 1.) Think fastest runner getting the first place. If you want the largest value to get rank 1 (e.g., the longest jumper

gets the first place), you need to put the variable inside `desc()`:

```
min_rank(desc(x))
```

```
## [1] 1 3 2 5 4
```

The three ranking functions, `row_number`, `min_rank`, and `dense_rank` differ only in what they do when there are ties. For example:

```
x <- c(5, 7, 11, 10, 5)
row_number(x)
```

```
## [1] 1 3 5 4 2
```

```
min_rank(x)
```

```
## [1] 1 3 5 4 1
```

```
dense_rank(x)
```

```
## [1] 1 2 4 3 1
```

Note thank there are two elements of `x` with the lowest value `5`. With `min_rank`, they are both ranked 1, and the next-ranked element (7) gets ranked 3. (There are no elements ranked 2.) With `dense_rank`, 5s would still get ranked 1, but 7 would get the next available rank, that is 2. With `row_number`, there are no duplicate ranks: ties get resolved on a first-come first-serve basis (even when ranking in descending order. Try `row_number(desc(x))`.)

`ntile(x, n)` also uses ordering by value of a variable `x`, but what does with it is chunks the data into `n` equal-sized portions. What this means is that if `n` is 4, you get your data divided into quartiles. (`n=100` would get you percentiles, etc.)

Example: select the lightest quartile of cars and sort them by gas mileage.

```
add_rownames(mtcars) %>%
  mutate(wt_cat = ntile(wt, 4)) %>%
  filter(wt_cat == 1) %>%
  arrange(desc(mpg))
```

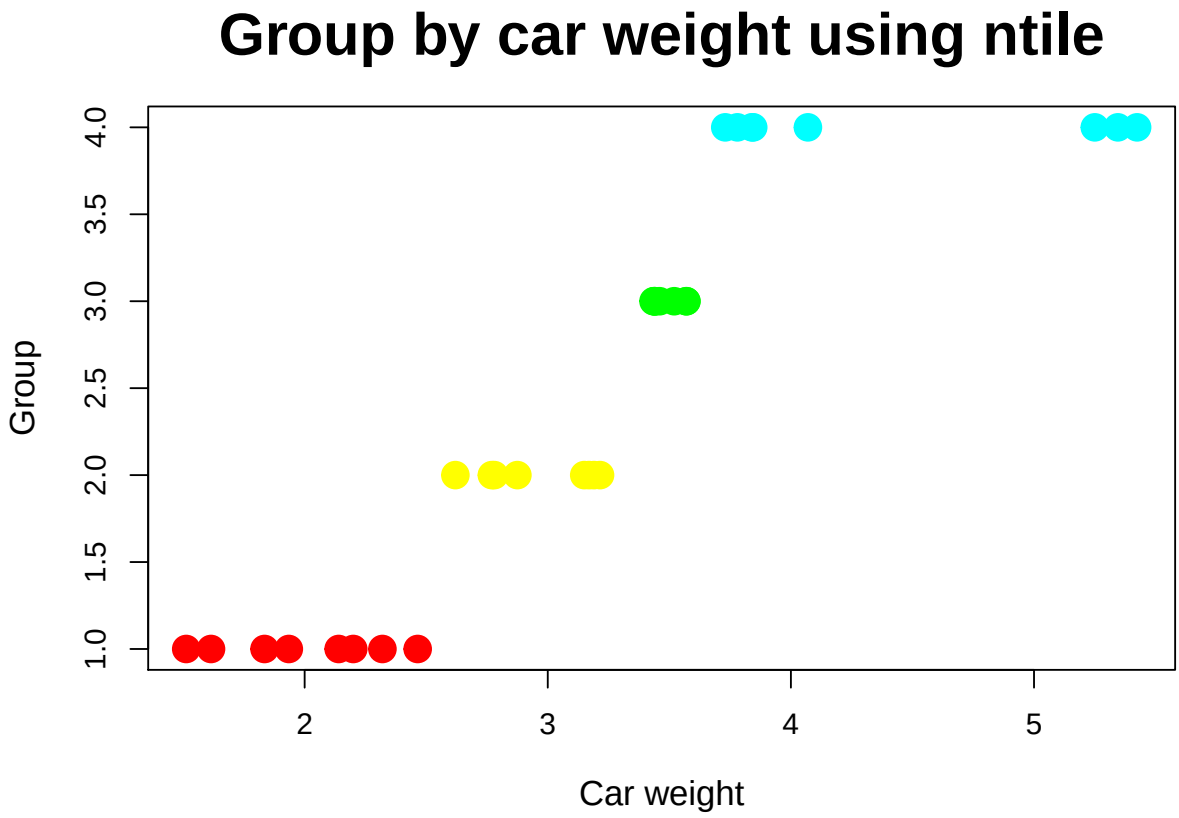
```
## Warning: Deprecated, use tibble::rownames_to_column() instead.
```

```
## # A tibble: 8 x 13
##   rowname        mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear
##   <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Toyota Coro~  33.9    4.  71.1   65.  4.22  1.84  19.9    1.    1.    4.
## 2 Fiat 128      32.4    4.  78.7   66.  4.08  2.20  19.5    1.    1.    4.
## 3 Honda Civic   30.4    4.  75.7   52.  4.93  1.62  18.5    1.    1.    4.
## 4 Lotus Europa  30.4    4.  95.1  113.  3.77  1.51  16.9    1.    1.    5.
## 5 Fiat X1-9     27.3    4.  79.0   66.  4.08  1.94  18.9    1.    1.    4.
## 6 Porsche 914~  26.0    4. 120.    91.  4.43  2.14  16.7    0.    1.    5.
## 7 Datsun 710    22.8    4. 108.    93.  3.85  2.32  18.6    1.    1.    4.
## 8 Toyota Coro~  21.5    4. 120.    97.  3.70  2.46  20.0    1.    0.    3.
## # ... with 2 more variables: carb <dbl>, wt_cat <int>
```

Visualization of the functionality of `ntile(x, n)`:

```
plot(mtcars$wt,ntile(mtcars$wt, 4),xlab='Car weight',ylab='Group',main='Group by car weight using ntile
colorwt=c()
colorwt[ntile(mtcars$wt, 4)==1]='red'
colorwt[ntile(mtcars$wt, 4)==2]='yellow'
colorwt[ntile(mtcars$wt, 4)==3]='green'
```

```
colorwt[ntile(mtcars$wt, 4)==4]='cyan'
points(mtcars$wt,ntile(mtcars$wt, 4),col=colorwt,cex=2,pch=19)
```



Example: calculate the average engine size and gas mileage for each quartile of weights:

```
mtcars %>%
  mutate(wt_quart = ntile(wt, 4)) %>%
  group_by(wt_quart) %>%
  summarise(mpg=mean(mpg), wt=mean(wt))
```

```
## # A tibble: 4 x 3
##   wt_quart   mpg    wt
##      <int> <dbl> <dbl>
## 1        1  28.1  2.00
## 2        2  20.9  2.97
## 3        3  16.7  3.48
## 4        4  14.6  4.41
```

Working with multiple tables

In practice, data relevant to your analysis often resides in multiple tables (e.g., a patient's personal info vs. daily measurements), and from time to time you will want to combine them in various ways. Dplyr provides "two-table verbs" that do just that.

All of these verbs work similarly:

- input data frames in the two argument
- output is another data frame
- unlike single-table verbs, additional arguments are used less often

Set operations

These are the easiest verbs to understand because they work just like their mathematical (and base R) equivalents:

- `intersect`: return only observations in both tables
- `union`: return unique observations in both tables
- `setdiff`: return observations only in the left table

Example:

```
df1 <- data.frame(x = 1:2, y = c(1L, 1L))
df2 <- data.frame(x = 1:2, y = 1:2)
intersect(df1, df2)
```

```
##   x y
## 1 1 1
```

```
union(df1, df2)
```

```
##   x y
## 1 1 1
## 2 2 1
## 3 2 2
```

Filtering joins

Filtering joins keep or drop observations from one table if they match an observation in another table. The matches are done on all the variables in common, or as specified by the `by` argument.

- `semi_join`: keeps all observations that have a match
- `anti_join`: drops all observations that have a match

These are most useful when checking your data to find table mismatches/missing entries. For example: to find all flights in `flights` that don't have a matching plane in the `planes` table:

```
#install.packages("nycflights13")
library(nycflights13)
flights %>%
  anti_join(planes, by = 'tailnum') %>%
  count(tailnum, sort=TRUE)
```

```
## # A tibble: 722 x 2
##    tailnum     n
##    <chr>   <int>
##  1 <NA>     2512
##  2 N725MQ    575
##  3 N722MQ    513
##  4 N723MQ    507
##  5 N713MQ    483
##  6 N735MQ    396
```

```
##  7 NOEGMQ    371
##  8 N534MQ    364
##  9 N542MQ    363
## 10 N531MQ    349
## # ... with 712 more rows
```

Why do I have to say `by = 'tailnum'`? Because both tables also have a column 'year', but it means different things (flight date vs. the plane's year manufactured) and only 'tailnum' is a valid "key" to match on. If I don't specify the key(s) to match on, *dplyr* will use all variables that appear in both tables, so in this case I have to narrow this down to just `tailnum`.

If the key I want to match on appear under different names in each table, I can use a named vector notation, e.g., `by = c(x = y)` to match variable `x` in the first table to variable `y` in the second.

```
df1 %>% semi_join(df2, by = c('x' = 'y'))
```

```
##   x y
## 1 1 1
## 2 2 1
```

Mutating joins

Mutating joins combine variables from two tables. There are four verbs in this family, distinguished by what they do when there isn't a match.

- `inner_join`: only include observations in both tables

  ```
  df1 <- data.frame(x = 1:2, y = 2:1)
  df2 <- data.frame(x = c(1, 3), a = 10, b = 'foo')
  df1 %>% inner_join(df2)
  ```

  ```
  ## Joining, by = "x"
  ```

  ```
  ##   x y  a   b
  ## 1 1 2 10 foo
  ```

- `left_join`: include all observations from the left table, filling in NAs for variables where the observation doesn't appear on the right

  ```
  df1 %>% left_join(df2)
  ```

  ```
  ## Joining, by = "x"
  ```

  ```
  ##   x y  a    b
  ## 1 1 2 10  foo
  ## 2 2 1 NA <NA>
  ```

- `right_join`: like `left_join`, but including all observations from the right tables, etc.

  ```
  df1 %>% right_join(df2)
  ```

  ```
  ## Joining, by = "x"
  ```

  ```
  ##   x  y  a   b
  ## 1 1  2 10 foo
  ## 2 3 NA 10 foo
  ```

- `full_join`: include all observations from both tables

  ```
  df1 %>% full_join(df2)
  ```

```
## Joining, by = "x"
```

```
##   x  y  a    b
## 1 1  2 10  foo
## 2 2  1 NA <NA>
## 3 3 NA 10  foo
```

Data manipulation in base R

All the operations we've seen with *dplyr* can also be done with the base R. The reason so many users prefer *dplyr* is because it is more readable and consistent than base R. Compare:

- `filter` + `select` can be done with indexing or `subset`:

```
mtcars %>% filter(cyl == 4) %>% select(disp, mpg)
```

```
##     disp  mpg
## 1  108.0 22.8
## 2  146.7 24.4
## 3  140.8 22.8
## 4   78.7 32.4
## 5   75.7 30.4
## 6   71.1 33.9
## 7  120.1 21.5
## 8   79.0 27.3
## 9  120.3 26.0
## 10  95.1 30.4
## 11 121.0 21.4
```

```
mtcars[mtcars$cyl == 4, c('disp', 'mpg')]
```

```
##                 disp  mpg
## Datsun 710     108.0 22.8
## Merc 240D      146.7 24.4
## Merc 230       140.8 22.8
## Fiat 128        78.7 32.4
## Honda Civic     75.7 30.4
## Toyota Corolla  71.1 33.9
## Toyota Corona  120.1 21.5
## Fiat X1-9       79.0 27.3
## Porsche 914-2  120.3 26.0
## Lotus Europa    95.1 30.4
## Volvo 142E     121.0 21.4
```

```
subset(mtcars, cyl == 4, select = c(disp, mpg))
```

```
##                 disp  mpg
## Datsun 710     108.0 22.8
## Merc 240D      146.7 24.4
## Merc 230       140.8 22.8
## Fiat 128        78.7 32.4
## Honda Civic     75.7 30.4
## Toyota Corolla  71.1 33.9
## Toyota Corona  120.1 21.5
## Fiat X1-9       79.0 27.3
## Porsche 914-2  120.3 26.0
```

```
## Lotus Europa    95.1 30.4
## Volvo 142E     121.0 21.4
```

- `arrange` can be done with `order` in the row index:

```
mtcars %>% arrange(cyl, desc(mpg))
```

```
##     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1  33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## 2  32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## 3  30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## 4  30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## 5  27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## 6  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## 7  24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## 8  22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 9  22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## 10 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## 11 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## 12 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## 13 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## 14 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## 15 19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## 16 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## 17 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## 18 17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## 19 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## 20 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## 21 17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## 22 16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## 23 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## 24 15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## 25 15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## 26 15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## 27 15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## 28 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## 29 14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## 30 13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## 31 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## 32 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
```

```
mtcars[order(mtcars$cyl, mtcars$mpg, decreasing=c(F, T), method='radix'), ]
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Lotus Europa      30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Fiat X1-9         27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2     26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Toyota Corona     21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
```

```
## 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
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
```

- `mutate` can be done with `transform`, but you can't refer to new variables:

```
mtcars %>% mutate(displ_l = disp / 61.0237)
```

```
##     mpg cyl  disp  hp drat    wt  qsec vs am gear carb  displ_l
## 1  21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4 2.621932
## 2  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4 2.621932
## 3  22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1 1.769804
## 4  21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1 4.227866
## 5  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2 5.899347
## 6  18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1 3.687092
## 7  14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4 5.899347
## 8  24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2 2.403984
## 9  22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2 2.307300
## 10 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4 2.746474
## 11 17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4 2.746474
## 12 16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3 4.519556
## 13 17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3 4.519556
## 14 15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3 4.519556
## 15 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4 7.734700
## 16 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4 7.538055
## 17 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4 7.210313
## 18 32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1 1.289663
## 19 30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2 1.240502
## 20 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1 1.165121
## 21 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1 1.968088
## 22 15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2 5.211090
## 23 15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2 4.981671
## 24 13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4 5.735477
## 25 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2 6.554830
## 26 27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1 1.294579
## 27 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2 1.971365
## 28 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2 1.558411
## 29 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4 5.751864
```

```
## 30 19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6 2.376126
## 31 15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8 4.932510
## 32 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2 1.982836
```

```
transform(mtcars, displ_l = disp / 61.0237)
```

```
##                     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
## Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280           19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4  8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128           32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic        30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla     33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona      21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger   15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin        15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28         13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird   19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9          27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2      26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa       30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L     15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino       19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora      15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E         21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
##                    displ_l
## Mazda RX4          2.621932
## Mazda RX4 Wag      2.621932
## Datsun 710         1.769804
## Hornet 4 Drive     4.227866
## Hornet Sportabout  5.899347
## Valiant            3.687092
## Duster 360         5.899347
## Merc 240D          2.403984
## Merc 230           2.307300
## Merc 280           2.746474
## Merc 280C          2.746474
## Merc 450SE         4.519556
## Merc 450SL         4.519556
## Merc 450SLC        4.519556
## Cadillac Fleetwood 7.734700
```

```
## Lincoln Continental 7.538055
## Chrysler Imperial   7.210313
## Fiat 128            1.289663
## Honda Civic         1.240502
## Toyota Corolla      1.165121
## Toyota Corona       1.968088
## Dodge Challenger    5.211090
## AMC Javelin         4.981671
## Camaro Z28          5.735477
## Pontiac Firebird    6.554830
## Fiat X1-9           1.294579
## Porsche 914-2       1.971365
## Lotus Europa        1.558411
## Ford Pantera L      5.751864
## Ferrari Dino        2.376126
## Maserati Bora       4.932510
## Volvo 142E          1.982836
```

- `group_by` and `summarise` can be done with `by`:

```
mtcars %>% group_by(cyl) %>% summarise(mpg = mean(mpg))
```

```
## # A tibble: 3 x 2
##     cyl   mpg
##   <dbl> <dbl>
## 1    4.  26.7
## 2    6.  19.7
## 3    8.  15.1
```

```
by(mtcars, mtcars$cyl, function(x) mean(x$mpg))
```

```
## mtcars$cyl: 4
## [1] 26.66364
## ------------------------------------------------------------
## mtcars$cyl: 6
## [1] 19.74286
## ------------------------------------------------------------
## mtcars$cyl: 8
## [1] 15.1
```

However, you don't get pipes, and can quickly find yourself having to keep intermediate results in additional objects, deal with indexing, and write your own functions (as you can see in the example above using `by`). It all becomes a lot more "programming" than "analysis".