9. Exploring Data with dplyr (2)

CT1100 - J. Duggan

dplyr

- All verbs (functions) work similarly
- The first argument is a data frame/tibble
- The subsequent arguments decide what to do with the data frame
- The result is a data frame (supports chaining of steps)

Function	Purpose
filter()	Pick observations by their values
arrange()	Reorder the rows
select()	Pick variables by their names
mutate()	Create new variables with functions of existing variables
summarise()	Collapse many values down to a single summary

(3) select()

- It is not uncommon to get datasets with hundreds, or even thousands, of variables
- A challenge is to narrow down on the variables of you're interested in
- select() allows you to rapidly zoom in on a useful subset using operations based on the variable names

```
select(mpg,model,year,displ, cty, hwy)
```

```
## # A tibble: 234 x 5
##    model        year displ   cty   hwy
##    <chr>       <int> <dbl> <int> <int>
##  1 a4           1999   1.8    18    29
##  2 a4           1999   1.8    21    29
##  3 a4           2008   2      20    31
##  4 a4           2008   2      21    30
##  5 a4           1999   2.8    16    26
##  6 a4           1999   2.8    18    26
```

Special Function with select

Special functions

As well as using existing functions like `:` and `c`, there are a number of special functions that only work inside `select`

- `starts_with(x, ignore.case = TRUE)`: names starts with `x`
- `ends_with(x, ignore.case = TRUE)`: names ends in `x`
- `contains(x, ignore.case = TRUE)`: selects all variables whose name contains `x`
- `matches(x, ignore.case = TRUE)`: selects all variables whose name matches the regular expression `x`
- `num_range("x", 1:5, width = 2)`: selects all variables (numerically) from x01 to x05.
- `one_of("x", "y", "z")`: selects variables provided in a character vector.
- `everything()`: selects all variables.

(4) mutate()

- It is often useful to add new columns that are functions of existing columns
- mutate() always adds new columns at the end of your data set.

```
sml <- select(mpg,model,displ,cty)
sml <- mutate(sml,Category=ifelse(cty>mean(cty),"AboveAvr","Be
sml
```

```
## # A tibble: 234 x 4
##    model      displ   cty Category
##    <chr>      <dbl> <int> <chr>
##  1 a4           1.8    18 AboveAvr
##  2 a4           1.8    21 AboveAvr
##  3 a4           2      20 AboveAvr
##  4 a4           2      21 AboveAvr
##  5 a4           2.8    16 BelowAvr
##  6 a4           2.8    18 AboveAvr
```

Useful creation functions

- There are many functions for creating new variables that can be used with mutate()
- The key property is that the function must be vectorised:
  - It must take a vector of values as input, and,
  - Return a vector with the same number of values as output

Grouping	Examples
Arithmetic Operators	+, -, *, /, ^
Modular Arithmetic	%/% - Integer division && - Remainder
Logs	log(), log2(), log10()
Offsets	lead() and lag() Find when values change x!=lag(x)
Cumulative and rolling aggregates	cumsum(), cumprod(), cummin(), cummax(), cummean()
Logical comparisons	<, <=, >, >=, !=
Ranking	min_rank()

Summary Two

- dplyr - a grammar of data manipulation
  - **select()**
  - **mutate()**
- Usefully combined with **%>%** operator