

Subsetting Data in R

Data Wrangling in R

Overview

We showed different ways to read data into R using:

```
readr::read_csv()  
readr::read_delim()  
readxl::read_excel()
```

In this module, we will show you how select rows and columns of datasets.

Setup

We will be using the **dplyr** package in the tidyverse.

Here are several resources on how to use `dplyr`:

- <https://dplyr.tidyverse.org/>
- <https://r4ds.had.co.nz/>
- <https://cran.rstudio.com/web/packages/dplyr/vignettes/dplyr.html>
- <https://stat545.com/dplyr-intro.html>

The `dplyr` package also interfaces well with tibbles.

Dataset

We will be using the `diamonds` dataset in the `ggplot2` package as an example (so make sure you initiate the `ggplot2` package if you are following along on your own).

```
head(diamonds)
```

```
# A tibble: 6 x 10
  carat cut      color clarity depth table price      x      y      z
  <dbl> <ord>    <ord> <ord>    <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23 Ideal      E      SI2     61.5    55   326   3.95   3.98   2.43
2  0.21 Premium    E      SI1     59.8    61   326   3.89   3.84   2.31
3  0.23 Good      E      VS1     56.9    65   327   4.05   4.07   2.31
4  0.29 Premium    I      VS2     62.4    58   334   4.2    4.23   2.63
5  0.31 Good      J      SI2     63.3    58   335   4.34   4.35   2.75
6  0.24 Very Good J      VVS2     62.8    57   336   3.94   3.96   2.48
```

Selecting a single column of a `data.frame`:

To grab just the values from a single column, you would use the `pull` function. The output will be a vector (and not a `tibble`).

Since this is a long vector we will just show the first 6 values using the `head` function around the output of the `pull` function.

```
head(pull(diamonds, carat))
```

```
[1] 0.23 0.21 0.23 0.29 0.31 0.24
```

Using the `pipe` (comes with `dplyr`):

That was a lot of typing and nested functions, which can be confusing. Recently, the pipe `%>%` makes things such as this much more readable. It reads left side “pipes” into right side. RStudio CMD/Ctrl + Shift + M shortcut.

Using the `pipe` (comes with `dplyr`):

Pipe `diamonds` into `select`, then pipe that into `pull`, and then show the head:

```
diamonds %>% pull(carat) %>% head()
```

```
[1] 0.23 0.21 0.23 0.29 0.31 0.24
```

Selecting a single column of a `data.frame`:

The `pull` function is equivalent to using the `$` method (in base R).

We can grab the `carat` column using the `$` operator.

```
head(pull(diamonds, carat))
```

```
[1] 0.23 0.21 0.23 0.29 0.31 0.24
```

```
head(diamonds$carat)
```

```
[1] 0.23 0.21 0.23 0.29 0.31 0.24
```

Note this does *not* return a `tibble` (or `data.frame`) but rather a vector.

Selecting a single column of a `data.frame`:

The `select` function extracts one or more columns from a `tibble` or `data.frame` and returns a `tibble` (not a vector).

```
select(diamonds, carat)
```

```
# A tibble: 53,940 x 1
  carat
  <dbl>
1  0.23
2  0.21
3  0.23
4  0.29
5  0.31
6  0.24
7  0.24
8  0.26
9  0.22
10 0.23
# ... with 53,930 more rows
```

Selecting multiple columns of a `data.frame`:

The `select` command from `dplyr` is very flexible. You just need to list all columns you want to extract separated by commas. You can use this as a way to just keep the columns you want for example.

```
select(diamonds, carat, depth)
```

```
# A tibble: 53,940 x 2
  carat depth
  <dbl> <dbl>
1  0.23  61.5
2  0.21  59.8
3  0.23  56.9
4  0.29  62.4
5  0.31  63.3
6  0.24  62.8
7  0.24  62.3
8  0.26  61.9
9  0.22  65.1
10 0.23  59.4
# ... with 53,930 more rows
```

See the Select “helpers”

Run the command:

```
??tidyselect::select_helpers
```

Here are a few:

```
last_col()  
ends_with()  
starts_with()  
contains() # like searching
```

Tidymodels helpers

For example, we can take all columns that start with a "c":

```
diamonds %>% select(starts_with("c"))
```

```
# A tibble: 53,940 x 4
  carat cut      color clarity
  <dbl> <ord>    <ord> <ord>
1  0.23 Ideal      E      SI2
2  0.21 Premium    E      SI1
3  0.23 Good       E      VS1
4  0.29 Premium    I      VS2
5  0.31 Good       J      SI2
6  0.24 Very Good J      VVS2
7  0.24 Very Good I      VVS1
8  0.26 Very Good H      SI1
9  0.22 Fair       E      VS2
10 0.23 Very Good H      VS1
# ... with 53,930 more rows
```

Tidysselect helpers

Or we can take all columns that end with an "e":

```
diamonds %>% select(ends_with("e"))
```

```
# A tibble: 53,940 x 2
  table price
  <dbl> <int>
1     55   326
2     61   326
3     65   327
4     58   334
5     58   335
6     57   336
7     57   336
8     55   337
9     61   337
10    61   338
# ... with 53,930 more rows
```

Tidymodel helpers

We are going to cover “fancier” ways of matching column names (and strings more generally) in the data cleaning lecture.

Subset rows of a `data.frame`:

The command in `dplyr` for subsetting rows is `filter`. Try `?filter`.

The easiest way to filter is by testing whether numeric observations are greater than or less than some cutoff:

```
filter(diamonds, depth > 60)
```

```
# A tibble: 48,315 x 10
```

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|----|-------|-----------|-------|---------|-------|-------|-------|-------|-------|-------|
| | <dbl> | <ord> | <ord> | <ord> | <dbl> | <dbl> | <int> | <dbl> | <dbl> | <dbl> |
| 1 | 0.23 | Ideal | E | SI2 | 61.5 | 55 | 326 | 3.95 | 3.98 | 2.43 |
| 2 | 0.29 | Premium | I | VS2 | 62.4 | 58 | 334 | 4.2 | 4.23 | 2.63 |
| 3 | 0.31 | Good | J | SI2 | 63.3 | 58 | 335 | 4.34 | 4.35 | 2.75 |
| 4 | 0.24 | Very Good | J | VVS2 | 62.8 | 57 | 336 | 3.94 | 3.96 | 2.48 |
| 5 | 0.24 | Very Good | I | VVS1 | 62.3 | 57 | 336 | 3.95 | 3.98 | 2.47 |
| 6 | 0.26 | Very Good | H | SI1 | 61.9 | 55 | 337 | 4.07 | 4.11 | 2.53 |
| 7 | 0.22 | Fair | E | VS2 | 65.1 | 61 | 337 | 3.87 | 3.78 | 2.49 |
| 8 | 0.3 | Good | J | SI1 | 64 | 55 | 339 | 4.25 | 4.28 | 2.73 |
| 9 | 0.23 | Ideal | J | VS1 | 62.8 | 56 | 340 | 3.93 | 3.9 | 2.46 |
| 10 | 0.22 | Premium | F | SI1 | 60.4 | 61 | 342 | 3.88 | 3.84 | 2.33 |

```
# ... with 48,305 more rows
```

Note, no `$` or subsetting is necessary. R “knows” `depth` refers to a column of `diamonds`.

Subset rows of a `data.frame`:

You can also using piping here:

```
diamonds %>% filter(depth > 60)
```

```
# A tibble: 48,315 x 10
```

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|----|-------|-----------|-------|---------|-------|-------|-------|-------|-------|-------|
| | <dbl> | <ord> | <ord> | <ord> | <dbl> | <dbl> | <int> | <dbl> | <dbl> | <dbl> |
| 1 | 0.23 | Ideal | E | SI2 | 61.5 | 55 | 326 | 3.95 | 3.98 | 2.43 |
| 2 | 0.29 | Premium | I | VS2 | 62.4 | 58 | 334 | 4.2 | 4.23 | 2.63 |
| 3 | 0.31 | Good | J | SI2 | 63.3 | 58 | 335 | 4.34 | 4.35 | 2.75 |
| 4 | 0.24 | Very Good | J | VVS2 | 62.8 | 57 | 336 | 3.94 | 3.96 | 2.48 |
| 5 | 0.24 | Very Good | I | VVS1 | 62.3 | 57 | 336 | 3.95 | 3.98 | 2.47 |
| 6 | 0.26 | Very Good | H | SI1 | 61.9 | 55 | 337 | 4.07 | 4.11 | 2.53 |
| 7 | 0.22 | Fair | E | VS2 | 65.1 | 61 | 337 | 3.87 | 3.78 | 2.49 |
| 8 | 0.3 | Good | J | SI1 | 64 | 55 | 339 | 4.25 | 4.28 | 2.73 |
| 9 | 0.23 | Ideal | J | VS1 | 62.8 | 56 | 340 | 3.93 | 3.9 | 2.46 |
| 10 | 0.22 | Premium | F | SI1 | 60.4 | 61 | 342 | 3.88 | 3.84 | 2.33 |

```
# ... with 48,305 more rows
```


Subset rows of a `data.frame`:

You can combine filtering on multiple columns by separating the filter arguments with commas:

```
diamonds %>% filter(depth > 60, table > 60, price > 2775)
```

```
# A tibble: 1,704 x 10
```

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|----|-------|-----------|-------|---------|-------|-------|-------|-------|-------|-------|
| | <dbl> | <ord> | <ord> | <ord> | <dbl> | <dbl> | <int> | <dbl> | <dbl> | <dbl> |
| 1 | 0.72 | Premium | F | SI1 | 61.8 | 61 | 2777 | 5.82 | 5.71 | 3.56 |
| 2 | 0.72 | Very Good | H | VS1 | 60.6 | 63 | 2782 | 5.83 | 5.76 | 3.51 |
| 3 | 0.81 | Good | G | SI2 | 61 | 61 | 2789 | 5.94 | 5.99 | 3.64 |
| 4 | 0.71 | Premium | F | VS1 | 60.1 | 62 | 2790 | 5.77 | 5.74 | 3.46 |
| 5 | 0.71 | Premium | G | VS1 | 62.4 | 61 | 2803 | 5.7 | 5.65 | 3.54 |
| 6 | 0.74 | Fair | F | VS2 | 61.1 | 68 | 2805 | 5.82 | 5.75 | 3.53 |
| 7 | 0.7 | Good | F | VS1 | 62.8 | 61 | 2810 | 5.57 | 5.61 | 3.51 |
| 8 | 0.7 | Very Good | F | VS2 | 60.9 | 61 | 2812 | 5.66 | 5.71 | 3.46 |
| 9 | 0.71 | Good | E | SI1 | 62.8 | 64 | 2817 | 5.6 | 5.54 | 3.5 |
| 10 | 0.7 | Premium | E | VS2 | 62.4 | 61 | 2818 | 5.66 | 5.63 | 3.52 |

```
# ... with 1,694 more rows
```

Subset rows of a `data.frame`:

You can also filter character strings by a single value or category:

```
diamonds %>% filter(color == "I",  
                    clarity == "SI2", cut == "Premium")
```

```
# A tibble: 312 x 10  
  carat cut      color clarity depth table price      x      y      z  
  <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>  
1  0.42 Premium I      SI2     61.5    59   552  4.78  4.84  2.96  
2    1    Premium I      SI2     58.2    60  2795  6.61  6.55  3.83  
3  0.9    Premium I      SI2     62.2    59  2826  6.11  6.07  3.79  
4  1.05 Premium I      SI2     58.3    57  2911  6.72  6.67  3.9  
5  0.91 Premium I      SI2     62      59  2913  6.18  6.23  3.85  
6  0.9    Premium I      SI2     62.5    58  2948  6.15  6.1    3.83  
7  0.9    Premium I      SI2     60.6    60  2948  6.28  6.23  3.79  
8  1.06 Premium I      SI2     61.5    57  2968  6.57  6.49  4.02  
9  0.91 Premium I      SI2     60.2    59  2981  6.29  6.24  3.77  
10 0.9    Premium I      SI2     60.6    60  3001  6.23  6.28  3.79  
# ... with 302 more rows
```

Subset rows of a `data.frame`:

Sometimes you want to be able to filter on matching several values or categories. The `%in%` operator is useful here:

```
diamonds %>% filter(clarity %in% c("SI1", "SI2"))
```

```
# A tibble: 22,259 x 10
```

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|----|-------|-----------|-------|---------|-------|-------|-------|-------|-------|-------|
| | <dbl> | <ord> | <ord> | <ord> | <dbl> | <dbl> | <int> | <dbl> | <dbl> | <dbl> |
| 1 | 0.23 | Ideal | E | SI2 | 61.5 | 55 | 326 | 3.95 | 3.98 | 2.43 |
| 2 | 0.21 | Premium | E | SI1 | 59.8 | 61 | 326 | 3.89 | 3.84 | 2.31 |
| 3 | 0.31 | Good | J | SI2 | 63.3 | 58 | 335 | 4.34 | 4.35 | 2.75 |
| 4 | 0.26 | Very Good | H | SI1 | 61.9 | 55 | 337 | 4.07 | 4.11 | 2.53 |
| 5 | 0.3 | Good | J | SI1 | 64 | 55 | 339 | 4.25 | 4.28 | 2.73 |
| 6 | 0.22 | Premium | F | SI1 | 60.4 | 61 | 342 | 3.88 | 3.84 | 2.33 |
| 7 | 0.31 | Ideal | J | SI2 | 62.2 | 54 | 344 | 4.35 | 4.37 | 2.71 |
| 8 | 0.2 | Premium | E | SI2 | 60.2 | 62 | 345 | 3.79 | 3.75 | 2.27 |
| 9 | 0.3 | Ideal | I | SI2 | 62 | 54 | 348 | 4.31 | 4.34 | 2.68 |
| 10 | 0.3 | Good | J | SI1 | 63.4 | 54 | 351 | 4.23 | 4.29 | 2.7 |

```
# ... with 22,249 more rows
```

Subset rows of a `data.frame`:

You can mix and match filtering on numeric and categorical/character columns in the same `filter()` command:

```
diamonds %>% filter(clarity %in% c("SI1", "SI2"),  
                    cut == "Premium", price > 3000)
```

```
# A tibble: 3,976 x 10
```

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|----|-------|---------|-------|---------|-------|-------|-------|-------|-------|-------|
| | <dbl> | <ord> | <ord> | <ord> | <dbl> | <dbl> | <int> | <dbl> | <dbl> | <dbl> |
| 1 | 0.9 | Premium | I | SI2 | 60.6 | 60 | 3001 | 6.23 | 6.28 | 3.79 |
| 2 | 0.81 | Premium | F | SI1 | 61.9 | 58 | 3004 | 5.99 | 5.96 | 3.7 |
| 3 | 0.92 | Premium | D | SI2 | 60.2 | 61 | 3004 | 6.32 | 6.27 | 3.79 |
| 4 | 0.9 | Premium | D | SI1 | 62.2 | 60 | 3013 | 6.08 | 6.05 | 3.77 |
| 5 | 0.96 | Premium | E | SI2 | 62.8 | 60 | 3016 | 6.3 | 6.24 | 3.94 |
| 6 | 0.93 | Premium | G | SI2 | 61.4 | 56 | 3019 | 6.27 | 6.23 | 3.84 |
| 7 | 0.78 | Premium | D | SI1 | 60.4 | 57 | 3019 | 6.02 | 5.97 | 3.62 |
| 8 | 0.75 | Premium | E | SI1 | 61.7 | 60 | 3024 | 5.84 | 5.8 | 3.59 |
| 9 | 0.75 | Premium | D | SI1 | 59.2 | 58 | 3024 | 5.96 | 5.93 | 3.52 |
| 10 | 1.02 | Premium | G | SI2 | 61.7 | 58 | 3027 | 6.46 | 6.41 | 3.97 |

```
# ... with 3,966 more rows
```

Subset rows of a `data.frame`:

Other useful logical tests:

`&` : AND

`|` : OR

`<=` : less than or equals

`>=` : greater than or equals

`!=` : not equals

Subset rows of a `data.frame`:

The AND operator (&) is the what is being performed “behind the scenes” when chaining together filter statements with commas:

```
diamonds %>% filter(depth > 60 & table > 60 & price > 2775)
```

```
# A tibble: 1,704 x 10
```

| | carat <dbl> | cut <ord> | color <ord> | clarity <ord> | depth <dbl> | table <dbl> | price <int> | x <dbl> | y <dbl> | z <dbl> |
|----|----------------|--------------|----------------|------------------|----------------|----------------|----------------|------------|------------|------------|
| 1 | 0.72 | Premium | F | SI1 | 61.8 | 61 | 2777 | 5.82 | 5.71 | 3.56 |
| 2 | 0.72 | Very Good | H | VS1 | 60.6 | 63 | 2782 | 5.83 | 5.76 | 3.51 |
| 3 | 0.81 | Good | G | SI2 | 61 | 61 | 2789 | 5.94 | 5.99 | 3.64 |
| 4 | 0.71 | Premium | F | VS1 | 60.1 | 62 | 2790 | 5.77 | 5.74 | 3.46 |
| 5 | 0.71 | Premium | G | VS1 | 62.4 | 61 | 2803 | 5.7 | 5.65 | 3.54 |
| 6 | 0.74 | Fair | F | VS2 | 61.1 | 68 | 2805 | 5.82 | 5.75 | 3.53 |
| 7 | 0.7 | Good | F | VS1 | 62.8 | 61 | 2810 | 5.57 | 5.61 | 3.51 |
| 8 | 0.7 | Very Good | F | VS2 | 60.9 | 61 | 2812 | 5.66 | 5.71 | 3.46 |
| 9 | 0.71 | Good | E | SI1 | 62.8 | 64 | 2817 | 5.6 | 5.54 | 3.5 |
| 10 | 0.7 | Premium | E | VS2 | 62.4 | 61 | 2818 | 5.66 | 5.63 | 3.52 |

```
# ... with 1,694 more rows
```

You can use either syntax.

Subset rows of a `data.frame`:

The OR operator (`|`) is more permissive than the AND operator:

```
diamonds %>% filter(depth > 60 | table > 60 | price > 2775)
```

```
# A tibble: 52,198 x 10
```

| | carat | cut | color | clarity | depth | table | price | x | y | z |
|----|-------|-----------|-------|---------|-------|-------|-------|-------|-------|-------|
| | <dbl> | <ord> | <ord> | <ord> | <dbl> | <dbl> | <int> | <dbl> | <dbl> | <dbl> |
| 1 | 0.23 | Ideal | E | SI2 | 61.5 | 55 | 326 | 3.95 | 3.98 | 2.43 |
| 2 | 0.21 | Premium | E | SI1 | 59.8 | 61 | 326 | 3.89 | 3.84 | 2.31 |
| 3 | 0.23 | Good | E | VS1 | 56.9 | 65 | 327 | 4.05 | 4.07 | 2.31 |
| 4 | 0.29 | Premium | I | VS2 | 62.4 | 58 | 334 | 4.2 | 4.23 | 2.63 |
| 5 | 0.31 | Good | J | SI2 | 63.3 | 58 | 335 | 4.34 | 4.35 | 2.75 |
| 6 | 0.24 | Very Good | J | VVS2 | 62.8 | 57 | 336 | 3.94 | 3.96 | 2.48 |
| 7 | 0.24 | Very Good | I | VVS1 | 62.3 | 57 | 336 | 3.95 | 3.98 | 2.47 |
| 8 | 0.26 | Very Good | H | SI1 | 61.9 | 55 | 337 | 4.07 | 4.11 | 2.53 |
| 9 | 0.22 | Fair | E | VS2 | 65.1 | 61 | 337 | 3.87 | 3.78 | 2.49 |
| 10 | 0.23 | Very Good | H | VS1 | 59.4 | 61 | 338 | 4 | 4.05 | 2.39 |

```
# ... with 52,188 more rows
```

Subset rows of a `data.frame`:

The OR operator (`|`) can be a substitute for `%in%` (although it might take more typing):

```
diamonds %>% filter(clarity == "SI1" | clarity == "SI2") %>% head(2)
```

```
# A tibble: 2 x 10
  carat cut      color clarity depth table price      x      y      z
<dbl> <ord>    <ord> <ord>    <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23 Ideal    E      SI2      61.5    55    326  3.95  3.98  2.43
2  0.21 Premium E      SI1      59.8    61    326  3.89  3.84  2.31
```

```
diamonds %>% filter(clarity %in% c("SI1", "SI2")) %>% head(2)
```

```
# A tibble: 2 x 10
  carat cut      color clarity depth table price      x      y      z
<dbl> <ord>    <ord> <ord>    <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23 Ideal    E      SI2      61.5    55    326  3.95  3.98  2.43
2  0.21 Premium E      SI1      59.8    61    326  3.89  3.84  2.31
```


Combining **filter** and **select**:

You can combine `filter` and `select` to subset the rows and columns, respectively, of a `data.frame`:

```
diamonds %>%  
  filter(clarity == "SI2") %>%  
  select(starts_with("c"))
```

A tibble: 9,194 x 4

| | carat <dbl> | cut <ord> | color <ord> | clarity <ord> |
|----|----------------|--------------|----------------|------------------|
| 1 | 0.23 | Ideal | E | SI2 |
| 2 | 0.31 | Good | J | SI2 |
| 3 | 0.31 | Ideal | J | SI2 |
| 4 | 0.2 | Premium | E | SI2 |
| 5 | 0.3 | Ideal | I | SI2 |
| 6 | 0.3 | Good | I | SI2 |
| 7 | 0.33 | Ideal | I | SI2 |
| 8 | 0.33 | Ideal | I | SI2 |
| 9 | 0.32 | Good | H | SI2 |
| 10 | 0.32 | Very Good | H | SI2 |

... with 9,184 more rows

Combining **filter** and **select**:

The order of these functions matters though, since you can remove columns that you might want to filter on.

```
diamonds %>%  
  select(starts_with("c")) %>%  
  filter(table > 60))
```

This will result in an error because the table column is now gone after the `select()` function!

Lab

[Link to Lab](#)