

Subsetting Data in R

Data Wrangling in R

Subsetting part 2

Data

Let's continue to work with the Diamond dataset from the `ggplot2` package of the tidyverse.

We will often use the `glimpse()` function of the `dplyr` package of the tidyverse to look at a rotated view of the data.

```
library(tidyverse)
head(diamonds)
```

```
# A tibble: 6 × 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
```

Let's learn more about this data

We can use `?diamonds` to get more informatin in the `Help` pane.

We might decide to rename some columns,

- `x` to be `length`
- `y` to be `width`
- `z` to be `depth`
- but first changing `depth` to be `depth_percentage`

Renaming Columns of a data frame or tibble

To rename columns in `dplyr`, you can use the `rename` function.

Notice the new name is listed **first**!

```
# general format! not code!  
{data you are creating or changing} <- {data you are using} %>%  
  rename({New Name} = {Old name})
```

```
diamonds_2 <- diamonds %>%  
  rename(depth_percentage = depth)  
head(diamonds_2, n = 3)
```

```
# A tibble: 3 × 10  
  carat cut      color clarity depth_percentage 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
```

More Renaming

```
diamonds_2<- diamonds %>%  
  rename(depth_percentage = depth,  
         length = x,  
         width = y,  
         depth = z)  
glimpse(diamonds_2)
```

Rows: 53,940

Columns: 10

\$ carat	<dbl>	0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22,
\$ cut	<ord>	Ideal, Premium, Good, Premium, Good, Very Good, Very
\$ color	<ord>	E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J,
\$ clarity	<ord>	SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, S
\$ depth_percentage	<dbl>	61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1,
\$ table	<dbl>	55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 5
\$ price	<int>	326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339
\$ length	<dbl>	3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87,
\$ width	<dbl>	3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78,
\$ depth	<dbl>	2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49,

Take Care with Column Names

When you can, avoid spaces, special punctuation, or numbers in column names, as these require special treatment to refer to them.

See https://jhudatascience.org/intro_to_r/quotes_vs_backticks.html for more guidance.

```
diamonds %>% rename(depth percentage = depth) # this will cause an error
```

```
diamonds %>% rename(depth_percentage = depth) # this will work
```

```
diamonds %>% rename(`depth percentage` = depth) # not recommended
```

Unusual Column Names

It's best to avoid unusual column names where possible, as things get tricky later.

We just showed the use of ``` backticks. You may see people use quotes as well.



Other atypical column names are those with:

- spaces
- number without characters
- number starting the name
- other punctuation marks (besides “_” or “.” and not at the beginning)

A solution!

Rename tricky column names so that you don't have to deal with them later!



Example

```
glimpse(diamonds_bad_names)
```

```
Rows: 53,940
Columns: 10
$ carat      <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26,
$ cut        <ord> Ideal, Premium, Good, Premium, Good, Very Good,
$ color      <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E,
$ clarity    <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2,
$ depth      <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9,
$ table      <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56,
$ `Price(in US dollars)` <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 33
$ `Length (in mm)`      <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07,
$ `Width in mm`         <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11,
$ `Depth percentage`    <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53,
```

```
diamonds_bad_names %>%
  rename(price = `Price(in US dollars)`)
```

```
# A tibble: 53,940 × 10
```

	carat	cut	color	clarity	depth	table	price	Length (in ... ¹	Width... ²	Depth
	<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	4.20	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

Renaming all columns of a data frame: dplyr

To rename all columns you use the `rename_with()`. In this case we will use `toupper()` to make all letters upper case. Could also use `tolower()` function.

```
diamonds_upper <- diamonds %>% rename_with(toupper)
head(diamonds_upper, 2)
```

```
# A tibble: 2 × 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_upper %>% rename_with(tolower) %>% head(n = 2)
```

```
# A tibble: 2 × 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
```

Janitor package

```
#install.packages("janitor")
library(janitor)
clean_names(diamonds_bad_names) %>% glimpse()
```

Rows: 53,940

Columns: 10

\$ carat	<dbl>	0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.
\$ cut	<ord>	Ideal, Premium, Good, Premium, Good, Very Good, Ve
\$ color	<ord>	E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I,
\$ clarity	<ord>	SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1
\$ depth	<dbl>	61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65
\$ table	<dbl>	55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61
\$ price_in_us_dollars	<int>	326, 326, 327, 334, 335, 336, 336, 337, 337, 338,
\$ length_in_mm	<dbl>	3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.
\$ width_in_mm	<dbl>	3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.
\$ depth_percentage	<dbl>	2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.

Subset based on a class

The **where()** function can help select columns of a specific class

`is.character()` and `is.numeric()` are often the most helpful

```
head(diamonds, 2)
```

```
# A tibble: 2 × 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 %>% select(where(is.numeric)) %>% head(n = 2)
```

```
# A tibble: 2 × 7
  carat depth table price      x      y      z
  <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23   61.5    55   326   3.95   3.98   2.43
2  0.21   59.8    61   326   3.89   3.84   2.31
```

The **where()** function can help select columns of a specific class

`is.character()` and `is.numeric()` are often the most helpful

```
head(diamonds, 2)
```

```
# A tibble: 2 × 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 %>% select(where(is.ordered)) %>% head(n = 2)
```

```
# A tibble: 2 × 3
  cut      color clarity
  <ord>   <ord> <ord>
1 Ideal    E     SI2
2 Premium E     SI1
```

distinct() function

To filter for distinct values from a variable, multiple variables, or an entire tibble you can use the `distinct()` function from the `dplyr` package. Similar to `count`, but without the number of times the value shows up.

```
distinct(diamonds, cut)
```

```
# A tibble: 5 × 1
  cut
<ord>
1 Ideal
2 Premium
3 Good
4 Very Good
5 Fair
```


Adding/Removing Columns

Adding columns to a data frame: dplyr (tidyverse way)

The `mutate` function in `dplyr` allows you to add or modify columns of a data frame.

General format - Not the code!

```
{data object to update} <- {data to use} %>%  
  mutate({new variable name} = {new variable source})
```

1 US dollar = 1.32 Canadian dollars

```
diamonds %>%  
  mutate(price_canadian = price * 1.32) %>% glimpse()
```

Rows: 53,940

Columns: 11

```
$ carat      <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, 0...  
$ cut        <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very Go...  
$ color      <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J, J...  
$ clarity    <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI1...  
$ depth      <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, 5...  
$ table      <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54,...  
$ price      <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339, ...  
$ x          <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87, 4...  
$ y          <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78, 4...  
$ z          <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49, 2...  
$ price_canadian <dbl> 430.32, 430.32, 431.64, 440.88, 442.20, 443.52, 443.52,...
```

Use mutate to modify existing columns

The `mutate` function in `dplyr` allows you to add or modify columns of a data frame.

```
# General format - Not the code!  
{data object to update} <- {data to use} %>%  
  mutate({variable name to change} = {variable modification})
```

```
mutate(diamonds, price = price * 1.32) %>% glimpse()
```

Rows: 53,940

Columns: 10

```
$ carat   <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, 0.23, 0...  
$ cut     <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very Good, Ver...  
$ color   <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J, J, J, I,...  
$ clarity <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI1, VS1, ...  
$ depth   <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, 59.4, 64...  
$ table   <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54, 62, 58...  
$ price   <dbl> 430.32, 430.32, 431.64, 440.88, 442.20, 443.52, 443.52, 444.84...  
$ x       <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87, 4.00, 4...  
$ y       <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78, 4.05, 4...  
$ z       <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49, 2.39, 2...
```

remember to save your data

If you want to actually make the change you need to reassign the data object.

```
diamonds <- diamonds %>% mutate(price = price * 1.32) %>% glimpse()
```

Removing columns of a data frame: dplyr

The `select` function can remove a column with minus (-)

```
select(diamonds, - price) %>% glimpse()
```

Rows: 53,940

Columns: 9

```
$ carat    <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, 0.23, 0...  
$ cut      <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very Good, Ver...  
$ color    <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J, J, J, I,...  
$ clarity  <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI1, VS1, ...  
$ depth    <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, 59.4, 64...  
$ table    <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54, 62, 58...  
$ x        <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87, 4.00, 4...  
$ y        <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78, 4.05, 4...  
$ z        <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49, 2.39, 2...
```

Or, you can simply select the columns you want to keep, ignoring the ones you want to remove.

Removing columns in a data frame: dplyr

You can use `c()` to list the columns to remove.

Remove `newcol` and `drat`:

```
select(diamonds, -c("x", "y", "z")) %>% glimpse()
```

Rows: 53,940

Columns: 7

```
$ carat    <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, 0.23, 0...  
$ cut      <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very Good, Ver...  
$ color    <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J, J, J, I,...  
$ clarity  <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI1, VS1, ...  
$ depth    <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, 59.4, 64...  
$ table    <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54, 62, 58...  
$ price    <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339, 340, 34...
```

Ordering columns

Ordering the columns of a data frame: dplyr

The `select` function can reorder columns.

```
head(diamonds, n = 2)
```

```
# A tibble: 2 × 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 %>% select(price, depth, carat, cut, color) %>% head(n = 2)
```

```
# A tibble: 2 × 5
  price depth carat cut      color
  <int> <dbl> <dbl> <ord>   <ord>
1   326  61.5  0.23 Ideal    E
2   326  59.8  0.21 Premium E
```


Ordering the columns of a data frame: dplyr

The `select` function can reorder columns. Put `price` first, then select the rest of columns:

```
head(diamonds, n = 2)
```

```
# A tibble: 2 × 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 %>% select(price, everything()) %>% head(n = 2)
```

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

Ordering the columns of a data frame: dplyr

Put `price` at the end ("remove, everything, then add back in"):

```
head(diamonds, n = 2)
```

```
# A tibble: 2 × 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 %>% select(-price, everything(), price) %>% head(n = 2)
```

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

Ordering the columns of a data frame: dplyr

In addition to `select` we can also use the `relocate()` function of `dplyr` to rearrange the columns for more complicated moves.

For example, let say we just wanted `price` to be before `carat`.

```
head(diamonds, n = 2)
```

```
# A tibble: 2 × 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 %>% relocate( price, .before = cut) %>% head(n = 2)
```

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

Ordering the column names of a data frame: alphabetically

Using the base R `order()` function.

```
order(colnames(diamonds))
```

```
[1] 1 4 3 2 5 7 6 8 9 10
```

```
diamonds %>% select(order(colnames(diamonds)))
```

```
# A tibble: 53,940 × 10
```

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

```
# ... with 53,930 more rows
```

Ordering rows

Ordering the rows of a data frame: dplyr

The `arrange` function can reorder rows By default, `arrange` orders in increasing order:

```
diamonds %>% arrange(cut)
```

```
# A tibble: 53,940 × 10
  carat cut    color clarity depth table price      x      y      z
  <dbl> <ord> <ord> <ord>    <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.22 Fair   E      VS2     65.1    61   337   3.87   3.78   2.49
2  0.86 Fair   E      SI2     55.1    69  2757   6.45   6.33   3.52
3  0.96 Fair   F      SI2     66.3    62  2759   6.27   5.95   4.07
4  0.7  Fair   F      VS2     64.5    57  2762   5.57   5.53   3.58
5  0.7  Fair   F      VS2     65.3    55  2762   5.63   5.58   3.66
6  0.91 Fair   H      SI2     64.4    57  2763   6.11   6.09   3.93
7  0.91 Fair   H      SI2     65.7    60  2763   6.03   5.99   3.95
8  0.98 Fair   H      SI2     67.9    60  2777   6.05   5.97   4.08
9  0.84 Fair   G      SI1     55.1    67  2782   6.39   6.2    3.47
10 1.01 Fair   E      I1      64.5    58  2788   6.29   6.21   4.03
# ... with 53,930 more rows
```

Ordering the rows of a data frame: dplyr

Use the `desc` to arrange the rows in descending order:

```
diamonds %>% arrange(depth)
```

```
# A tibble: 53,940 × 10
  carat cut    color clarity depth table price      x      y      z
  <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1     1 Fair    G      SI1     43     59  3634  6.32  6.27  3.97
2   1.09 Ideal  J      VS2     43     54  4778  6.53  6.55  4.12
3     1 Fair    G      VS2     44     53  4032  6.31  6.24  4.12
4   1.43 Fair    I      VS1    50.8     60  6727  7.73  7.25  3.93
5   0.3 Fair    E      VVS2    51     67   945  4.67  4.62  2.37
6   0.7 Fair    D      SI1    52.2     65  1895  6.04  5.99  3.14
7   0.37 Fair    F      IF     52.3     61  1166  4.96  4.91  2.58
8   0.56 Fair    H      VS2    52.7     70  1293  5.71  5.57  2.97
9   1.02 Fair    I      SI1     53     63  2856  6.84  6.77  3.66
10  0.96 Fair    E      SI2    53.1     63  2815  6.73  6.65  3.55
# ... with 53,930 more rows
```

Ordering the rows of a data frame: dplyr

Use the `desc` to arrange the rows in descending order:

```
diamonds %>% arrange(desc(price))
```

```
# A tibble: 53,940 × 10
```

	carat	cut	color	clarity	depth	table	price	x	y	z
	<dbl>	<ord>	<ord>	<ord>	<dbl>	<dbl>	<int>	<dbl>	<dbl>	<dbl>
1	2.29	Premium	I	VS2	60.8	60	18823	8.5	8.47	5.16
2	2	Very Good	G	SI1	63.5	56	18818	7.9	7.97	5.04
3	1.51	Ideal	G	IF	61.7	55	18806	7.37	7.41	4.56
4	2.07	Ideal	G	SI2	62.5	55	18804	8.2	8.13	5.11
5	2	Very Good	H	SI1	62.8	57	18803	7.95	8	5.01
6	2.29	Premium	I	SI1	61.8	59	18797	8.52	8.45	5.24
7	2.04	Premium	H	SI1	58.1	60	18795	8.37	8.28	4.84
8	2	Premium	I	VS1	60.8	59	18795	8.13	8.02	4.91
9	1.71	Premium	F	VS2	62.3	59	18791	7.57	7.53	4.7
10	2.15	Ideal	G	SI2	62.6	54	18791	8.29	8.35	5.21

```
# ... with 53,930 more rows
```


Ordering the rows of a data frame: dplyr

You can combine increasing and decreasing orderings:

```
arrange(diamonds, desc(carat), table)
```

```
# A tibble: 53,940 × 10
```

	carat	cut	color	clarity	depth	table	price	x	y	z
	<dbl>	<ord>	<ord>	<ord>	<dbl>	<dbl>	<int>	<dbl>	<dbl>	<dbl>
1	5.01	Fair	J	I1	65.5	59	18018	10.7	10.5	6.98
2	4.5	Fair	J	I1	65.8	58	18531	10.2	10.2	6.72
3	4.13	Fair	H	I1	64.8	61	17329	10	9.85	6.43
4	4.01	Premium	I	I1	61	61	15223	10.1	10.1	6.17
5	4.01	Premium	J	I1	62.5	62	15223	10.0	9.94	6.24
6	4	Very Good	I	I1	63.3	58	15984	10.0	9.94	6.31
7	3.67	Premium	I	I1	62.4	56	16193	9.86	9.81	6.13
8	3.65	Fair	H	I1	67.1	53	11668	9.53	9.48	6.38
9	3.51	Premium	J	VS2	62.5	59	18701	9.66	9.63	6.03
10	3.5	Ideal	H	I1	62.8	57	12587	9.65	9.59	6.03

```
# ... with 53,930 more rows
```

Summary

- `rename` can change a name - new name = old name
- `clean_names` of the `janitor` package can change many names
- `select()` and `relocate()` can be used to reorder columns
- `arrange()` can be used to reorder rows
- can arrange in descending order with `desc()`
- can remove rows with `filter()`
- can remove a column in a few ways:
 - using `select()` with negative sign in front of column name(s)
 - just not selecting it
- `mutate()` can be used to modify an existing variable or make a new variable

Lab

[Link to Lab](#)