Data Transformation

`dplyr` functions

- `select()` select columns
- `filter()` filter rows
- `mutate()` create new columns
- `arrange()` re-order or arrange rows
- `summarise()` summarize values
- `group_by()` allows for group operations in the "split-apply-combine" concept

---

Select columns

Select columns explicitely

```
  select(mpg, manufacturer, cyl, hwy, cty)
```

```
# A tibble: 234 x 4
   manufacturer   cyl   hwy   cty
   <chr>        <int> <int> <int>
 1 audi             4    29    18
 2 audi             4    29    21
 3 audi             4    31    20
 4 audi             4    30    21
 5 audi             6    26    16
 6 audi             6    26    18
 7 audi             6    27    18
 8 audi             4    26    18
```

Data transformation with dplyr : : **CHEAT SHEET**

dplyr functions work with pipes and expect **tidy data**. In tidy data:


Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …**)**
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(.data, …, wt = NULL, sort = FALSE, name = NULL**)** Count number of rows in each group defined by the variables in … Also **tally()**.
count(mtcars, cyl)

Group Cases

Use **group_by(**.data, …, .add = FALSE, .drop = TRUE**)** to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

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

Use **rowwise(**.data, …**)** to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

```
starwars %>%
  rowwise() %>%
  mutate(film_count = length(films))
```

ungroup(x, …**)** Returns ungrouped copy of table.
ungroup(g_mtcars)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …, .preserve = FALSE**)** Extract rows that meet logical criteria.
filter(mtcars, mpg > 20)

distinct(.data, …, .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(mtcars, gear)

slice(.data, …, .preserve = FALSE**)** Select rows by position.
slice(mtcars, 10:15)

slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE**)** Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
slice_sample(mtcars, n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE**)** and **slice_max()** Select rows with the lowest and highest values.
slice_min(mtcars, mpg, prop = 0.25)

slice_head(.data, …, n, prop**)** and **slice_tail()** Select the first or last rows.
slice_head(mtcars, n = 5)

Logical and boolean operators to use with filter()

==	<	<=	is.na()	%in%	|	xor()
!=	>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …, .by_group = FALSE**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, …, .before = NULL, .after = NULL**)** Add one or more rows to a table.
add_row(cars, speed = 1, dist = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, …**)** Extract column values as a vector, by name or index.
pull(mtcars, wt)

select(.data, …**)** Extract columns as a table.
select(mtcars, mpg, wt)

relocate(.data, …, .before = NULL, .after = NULL**)** Move columns to new position.
relocate(mtcars, mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. select(mtcars, mpg:cyl)

contains(match**)**	**num_range(**prefix, range**)**	**:**, e.g. mpg:cyl
ends_with(match**)**	**all_of(**x**)/any_of(**x, …, vars**)**	**-**, e.g, -gear
starts_with(match**)**	**matches(**match**)**	**everything()**

MANIPULATE MULTIPLE VARIABLES AT ONCE

across(.cols, .funs, …, .names = NULL**)** Summarise or mutate multiple columns in the same way.
summarise(mtcars, across(everything(), mean))

c_across(.cols**)** Compute across columns in row-wise data.
transmute(rowwise(UKgas), total = sum(c_across(1:2)))

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …, .keep = "all", .before = NULL, .after = NULL**)** Compute new column(s). Also **add_column(), add_count(),** and **add_tally().**
mutate(mtcars, gpm = 1 / mpg)

transmute(.data, …**)** Compute new column(s), drop others.
transmute(mtcars, gpm = 1 / mpg)

rename(.data, …**)** Rename columns. Use **rename_with()** to rename with a function.
rename(cars, distance = dist)

```
 9 audi              4    25    16
10 audi              4    28    20
# ... with 224 more rows
```

Negative select

```
  select(mpg, -model, -class)
```

```
# A tibble: 234 x 9
   manufacturer displ  year   cyl trans      drv     cty   hwy fl
   <chr>        <dbl> <int> <int> <chr>      <chr> <int> <int> <chr>
 1 audi           1.8  1999     4 auto(l5)   f        18    29 p
 2 audi           1.8  1999     4 manual(m5) f        21    29 p
 3 audi           2    2008     4 manual(m6) f        20    31 p
 4 audi           2    2008     4 auto(av)   f        21    30 p
 5 audi           2.8  1999     6 auto(l5)   f        16    26 p
 6 audi           2.8  1999     6 manual(m5) f        18    26 p
 7 audi           3.1  2008     6 auto(av)   f        18    27 p
 8 audi           1.8  1999     4 manual(m5) 4        18    26 p
 9 audi           1.8  1999     4 auto(l5)   4        16    25 p
10 audi           2    2008     4 manual(m6) 4        20    28 p
# ... with 224 more rows
```

Select columns using range

```
  select(mpg, cyl:cty)
```

```
# A tibble: 234 x 4
     cyl trans      drv     cty
   <int> <chr>      <chr> <int>
 1     4 auto(l5)   f        18
 2     4 manual(m5) f        21
 3     4 manual(m6) f        20
 4     4 auto(av)   f        21
 5     6 auto(l5)   f        16
 6     6 manual(m5) f        18
 7     6 auto(av)   f        18
 8     4 manual(m5) 4        18
 9     4 auto(l5)   4        16
```

```
10     4 manual(m6) 4        20
# ... with 224 more rows
```

Select columns using conditions

```
  select(mpg, starts_with("c"))
```

```
# A tibble: 234 x 3
     cyl   cty class
   <int> <int> <chr>
 1     4    18 compact
 2     4    21 compact
 3     4    20 compact
 4     4    21 compact
 5     6    16 compact
 6     6    18 compact
 7     6    18 compact
 8     4    18 compact
 9     4    16 compact
10     4    20 compact
# ... with 224 more rows
```

Combine select methods

```
  select(iris, Species, ends_with("Width"))
```

```
# A tibble: 150 x 3
   Species Sepal.Width Petal.Width
   <fct>         <dbl>       <dbl>
 1 setosa          3.5         0.2
 2 setosa          3           0.2
 3 setosa          3.2         0.2
 4 setosa          3.1         0.2
 5 setosa          3.6         0.2
 6 setosa          3.9         0.4
 7 setosa          3.4         0.3
 8 setosa          3.4         0.2
 9 setosa          2.9         0.2
10 setosa          3.1         0.1
# ... with 140 more rows
```

Exercise 1 - dataset

Take a look at dataset `babynames`

```
# install.packages("babynames")
require(babynames)
babynames
```

```
# A tibble: 1,924,665 x 5
    year sex   name          n   prop
   <dbl> <chr> <chr>     <int>  <dbl>
 1  1880 F     Mary       7065 0.0724
 2  1880 F     Anna       2604 0.0267
 3  1880 F     Emma       2003 0.0205
 4  1880 F     Elizabeth  1939 0.0199
 5  1880 F     Minnie     1746 0.0179
 6  1880 F     Margaret   1578 0.0162
 7  1880 F     Ida        1472 0.0151
 8  1880 F     Alice      1414 0.0145
 9  1880 F     Bertha     1320 0.0135
10  1880 F     Sarah      1288 0.0132
# ... with 1,924,655 more rows
```

Exercise 1

Alter the code to select just the `n` column:

```
select(babynames, _______)
```

Quiz

Which of these is NOT a way to select the `name` and `n` columns together?

```
select(babynames, -c(year, sex, prop))
select(babynames, name:n)
select(babynames, starts_with("n"))
select(babynames, ends_with("n"))
```

Filter rows

Simple condition

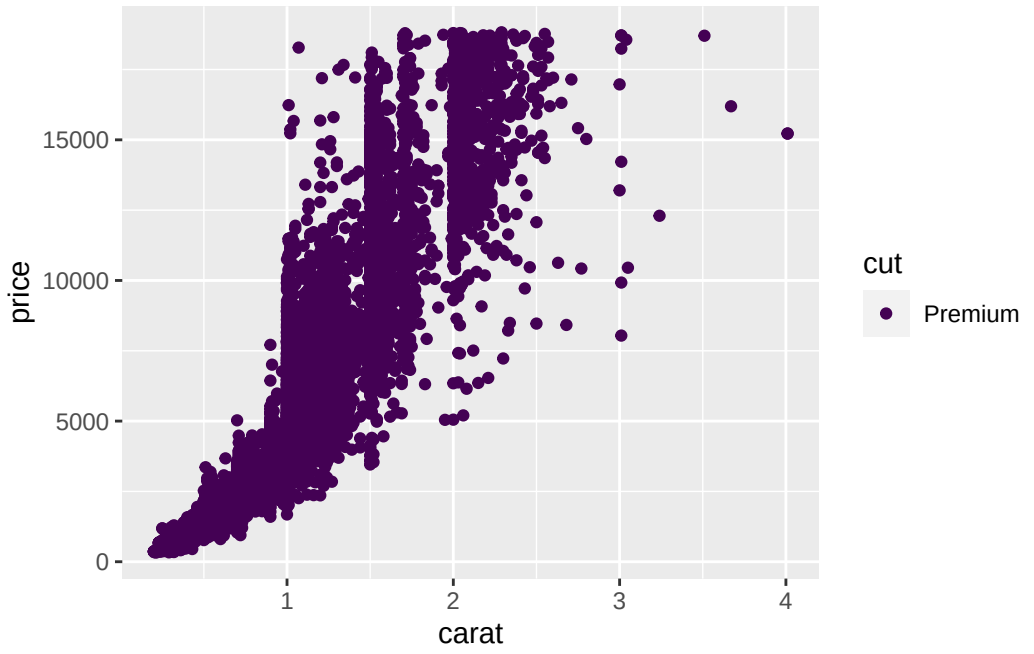
```
filter(mpg, hwy <= 14)
```

```
# A tibble: 7 x 11
  manufacturer model               displ  year   cyl trans      drv     cty   hwy fl    clas
  <chr>        <chr>               <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr
1 chevrolet    k1500 tahoe 4wd       5.3  2008     8 auto(l4)   4        11    14 e     suv
2 dodge        dakota pickup 4wd     4.7  2008     8 auto(l5)   4         9    12 e     picku
3 dodge        durango 4wd           4.7  2008     8 auto(l5)   4         9    12 e     suv
4 dodge        ram 1500 pickup 4wd   4.7  2008     8 auto(l5)   4         9    12 e     picku
5 dodge        ram 1500 pickup 4wd   4.7  2008     8 manual(m6) 4         9    12 e     picku
6 jeep         grand cherokee 4wd    4.7  2008     8 auto(l5)   4         9    12 e     suv
7 jeep         grand cherokee 4wd    6.1  2008     8 auto(l5)   4        11    14 p     suv
```

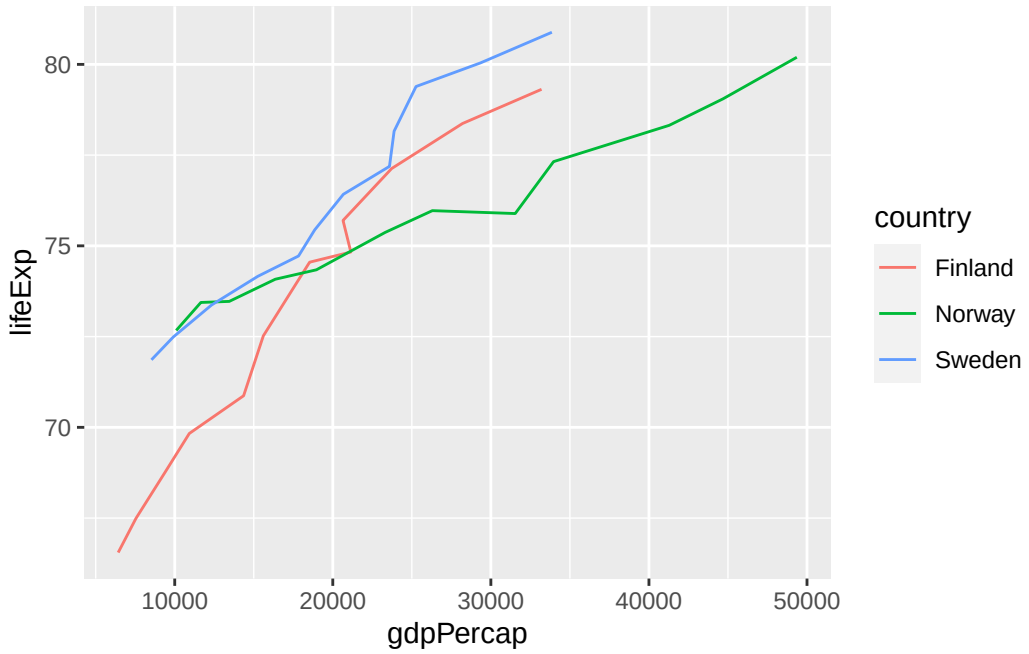
Boolean operators

- math: `==`, `<`, `<=`, `!=`
- logical: and `&`, or `|`
- set: `%in%`
- valid values: `is.na()`
- negation: `!is.na()`

```
data <- filter(diamonds, cut == "Premium")
ggplot(data, aes(carat, price, color = cut)) +
  geom_point()
```



```
data <- filter(gapminder, country %in% c("Finland", "Sweden", "Norway"))
ggplot(data, aes(gdpPercap, lifeExp, color = country)) +
  geom_path()
```



Multiple conditions

```
filter(gapminder, continent == "Europe", year >= 2000)
```

```
# A tibble: 60 x 6
   country                continent  year lifeExp      pop gdpPercap
   <fct>                  <fct>     <int>   <dbl>    <int>     <dbl>
 1 Albania                Europe     2002    75.7  3508512     4604.
 2 Albania                Europe     2007    76.4  3600523     5937.
 3 Austria                Europe     2002    79.0  8148312    32418.
 4 Austria                Europe     2007    79.8  8199783    36126.
 5 Belgium                Europe     2002    78.3 10311970    30486.
 6 Belgium                Europe     2007    79.4 10392226    33693.
 7 Bosnia and Herzegovina Europe     2002    74.1  4165416     6019.
 8 Bosnia and Herzegovina Europe     2007    74.9  4552198     7446.
 9 Bulgaria               Europe     2002    72.1  7661799     7697.
10 Bulgaria               Europe     2007    73.0  7322858    10681.
# ... with 50 more rows
```

Exercise 2

Use `filter`, `babynames`, and the logical operators to find:

- All of the names where prop is greater than or equal to 0.08
- All of the children named "Sea"

```
filter(babynames, ____________)
```

Exercise 3

Use Boolean operators to return only the rows that contain:

- *Boys* named Sue
- Names that were used by exactly 5 or 6 children in 1880
- Names that are one of Acura, Lexus, or Yugo

Sort values

Ascending sort

```
arrange(mpg, model)
```

```
# A tibble: 234 x 11
   manufacturer model       displ  year   cyl trans      drv     cty   hwy fl    class
   <chr>        <chr>       <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
 1 toyota       4runner 4wd   2.7  1999     4 manual(m5) 4        15    20 r     suv
 2 toyota       4runner 4wd   2.7  1999     4 auto(l4)   4        16    20 r     suv
 3 toyota       4runner 4wd   3.4  1999     6 auto(l4)   4        15    19 r     suv
 4 toyota       4runner 4wd   3.4  1999     6 manual(m5) 4        15    17 r     suv
 5 toyota       4runner 4wd   4    2008     6 auto(l5)   4        16    20 r     suv
 6 toyota       4runner 4wd   4.7  2008     8 auto(l5)   4        14    17 r     suv
 7 audi         a4            1.8  1999     4 auto(l5)   f        18    29 p     compact
 8 audi         a4            1.8  1999     4 manual(m5) f        21    29 p     compact
 9 audi         a4            2    2008     4 manual(m6) f        20    31 p     compact
10 audi         a4            2    2008     4 auto(av)   f        21    30 p     compact
# ... with 224 more rows
```

Descending sort

```
arrange(mpg, desc(model))
```

```
# A tibble: 234 x 11
   manufacturer model             displ  year   cyl trans      drv     cty   hwy fl    class
   <chr>        <chr>             <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
 1 toyota       toyota tacoma 4wd   2.7  1999     4 manual(m5) 4        15    20 r     picku
 2 toyota       toyota tacoma 4wd   2.7  1999     4 auto(l4)   4        16    20 r     picku
 3 toyota       toyota tacoma 4wd   2.7  2008     4 manual(m5) 4        17    22 r     picku
 4 toyota       toyota tacoma 4wd   3.4  1999     6 manual(m5) 4        15    17 r     picku
 5 toyota       toyota tacoma 4wd   3.4  1999     6 auto(l4)   4        15    19 r     picku
 6 toyota       toyota tacoma 4wd   4    2008     6 manual(m6) 4        15    18 r     picku
 7 toyota       toyota tacoma 4wd   4    2008     6 auto(l5)   4        16    20 r     picku
 8 hyundai      tiburon             2    1999     4 auto(l4)   f        19    26 r     subco
 9 hyundai      tiburon             2    1999     4 manual(m5) f        19    29 r     subco
10 hyundai      tiburon             2    2008     4 manual(m5) f        20    28 r     subco
# ... with 224 more rows
```

Descending sort - numeric

```
arrange(mpg, -hwy)
```

```
# A tibble: 234 x 11
   manufacturer model      displ  year   cyl trans      drv     cty   hwy fl    class
   <chr>        <chr>      <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
 1 volkswagen   jetta        1.9  1999     4 manual(m5) f        33    44 d     compact
 2 volkswagen   new beetle   1.9  1999     4 manual(m5) f        35    44 d     subcompact
 3 volkswagen   new beetle   1.9  1999     4 auto(l4)   f        29    41 d     subcompact
 4 toyota       corolla      1.8  2008     4 manual(m5) f        28    37 r     compact
 5 honda        civic        1.8  2008     4 auto(l5)   f        25    36 r     subcompact
 6 honda        civic        1.8  2008     4 auto(l5)   f        24    36 c     subcompact
 7 toyota       corolla      1.8  1999     4 manual(m5) f        26    35 r     compact
 8 toyota       corolla      1.8  2008     4 auto(l4)   f        26    35 r     compact
 9 honda        civic        1.8  2008     4 manual(m5) f        26    34 r     subcompact
10 honda        civic        1.6  1999     4 manual(m5) f        28    33 r     subcompact
# ... with 224 more rows
```

Sort by multiple values

```
arrange(mpg, cyl, -displ)
```

```
# A tibble: 234 x 11
   manufacturer model              displ  year   cyl trans      drv     cty   hwy fl    class
   <chr>        <chr>              <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
 1 toyota       4runner 4wd          2.7  1999     4 manual(m5) 4        15    20 r     suv
 2 toyota       4runner 4wd          2.7  1999     4 auto(l4)   4        16    20 r     suv
 3 toyota       toyota tacoma 4wd    2.7  1999     4 manual(m5) 4        15    20 r     picku
 4 toyota       toyota tacoma 4wd    2.7  1999     4 auto(l4)   4        16    20 r     picku
 5 toyota       toyota tacoma 4wd    2.7  2008     4 manual(m5) 4        17    22 r     picku
 6 nissan       altima               2.5  2008     4 auto(av)   f        23    31 r     midsi
 7 nissan       altima               2.5  2008     4 manual(m6) f        23    32 r     midsi
 8 subaru       forester awd         2.5  1999     4 manual(m5) 4        18    25 r     suv
 9 subaru       forester awd         2.5  1999     4 auto(l4)   4        18    24 r     suv
10 subaru       forester awd         2.5  2008     4 manual(m5) 4        20    27 r     suv
# ... with 224 more rows
```

Exercise 4

- What is the smallest value of `n`?
- What is the largest value of `prop`?

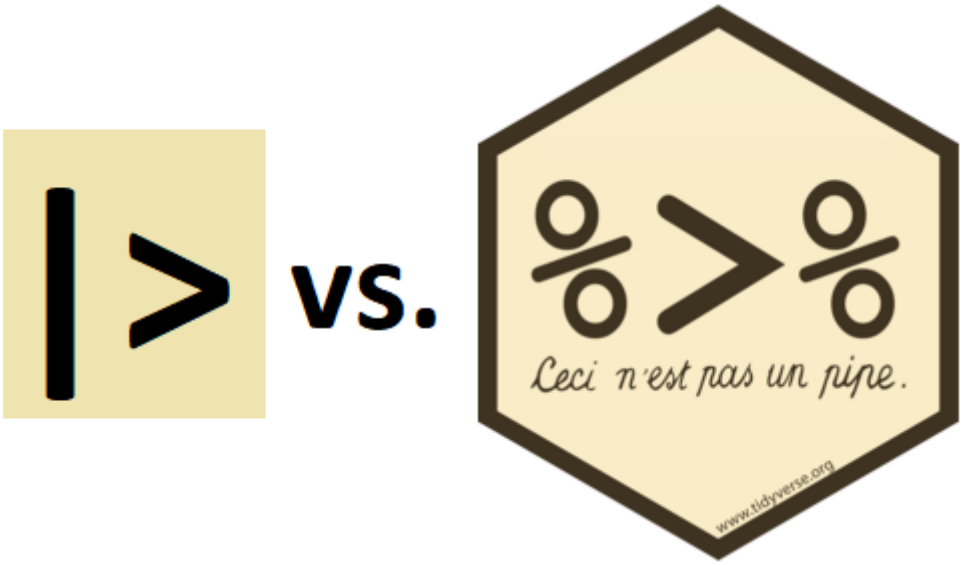
Pipes

Instead of

```
f(x)
```

write

```
x |> f()
```



Instead of

```
f(x, y)
```

write

```
x |> f(y)
```

Instead of

```
f(g(x))
```

write

```
x |> g() |> f()
```

Some hints

- Shortcut: `ctrl` + `shift` + `m`

- Read “then”
- Base `|>` since R 4.1 - use this
- Magrittr `%>%` is slower but more versatile

Pipe with data

```
mpg |> select(manufacturer, cyl, hwy, cty)
```

```
# A tibble: 234 x 4
   manufacturer   cyl   hwy   cty
   <chr>        <int> <int> <int>
 1 audi             4    29    18
 2 audi             4    29    21
 3 audi             4    31    20
 4 audi             4    30    21
 5 audi             6    26    16
 6 audi             6    26    18
 7 audi             6    27    18
 8 audi             4    26    18
 9 audi             4    25    16
10 audi             4    28    20
# ... with 224 more rows
```

Wrap lines

```
mpg |>
  select(manufacturer, cyl, hwy, cty)
```

```
# A tibble: 234 x 4
   manufacturer   cyl   hwy   cty
   <chr>        <int> <int> <int>
 1 audi             4    29    18
 2 audi             4    29    21
 3 audi             4    31    20
 4 audi             4    30    21
 5 audi             6    26    16
 6 audi             6    26    18
 7 audi             6    27    18
 8 audi             4    26    18
```

```
 9 audi              4    25    16
10 audi              4    28    20
# ... with 224 more rows
```

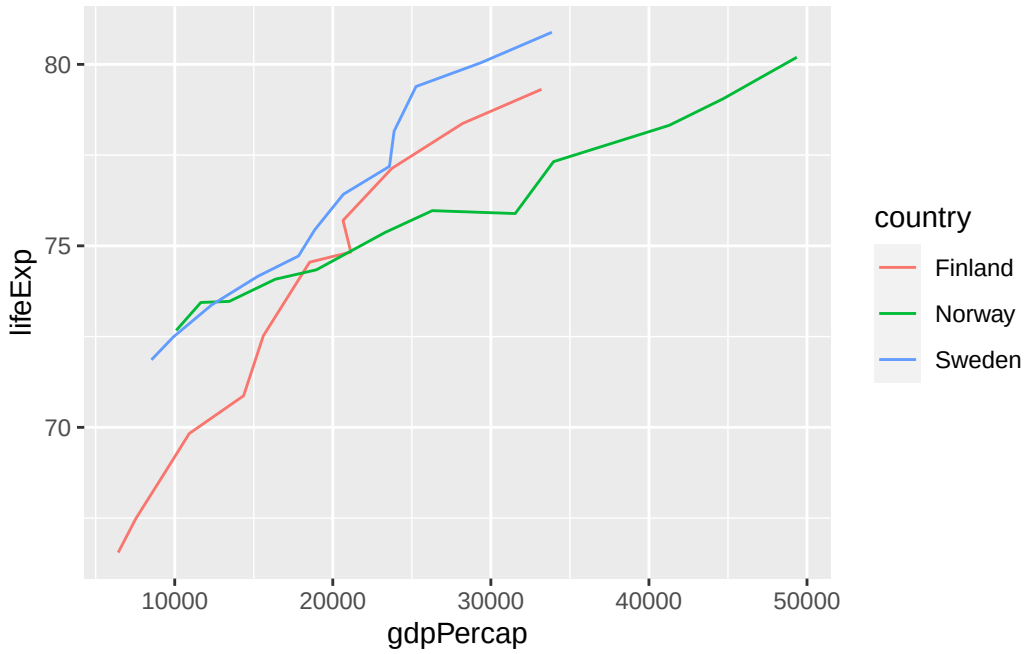
Chain operations

```
mpg |>
  select(manufacturer, cyl, hwy, cty) |>
  filter(cyl == 4)
```

```
# A tibble: 81 x 4
   manufacturer   cyl   hwy   cty
   <chr>        <int> <int> <int>
 1 audi             4    29    18
 2 audi             4    29    21
 3 audi             4    31    20
 4 audi             4    30    21
 5 audi             4    26    18
 6 audi             4    25    16
 7 audi             4    28    20
 8 audi             4    27    19
 9 chevrolet        4    27    19
10 chevrolet        4    30    22
# ... with 71 more rows
```

Together with ggplot

```
gapminder |>
  filter(country %in% c("Finland", "Sweden", "Norway")) |>
  ggplot(aes(gdpPercap, lifeExp, color = country)) +
  geom_path()
```



Exercise 5

Use `|>` to write a sequence of functions that:

1. Filters `babynames` to just the girls that were born in 2017, *then…*
2. Selects the `name` and `n` columns, *then…*
3. Arranges the results so that the most popular names are near the top.

Exercise 6

1. Trim `babynames` to just the rows that contain your `name` and your `sex`

2. Trim the result to just the columns that will appear in your graph (not strictly necessary, but useful practice)
3. Plot the results as a line graph with `year` on the x axis and `prop` on the y axis

Summarize data

Single statistic

```
mpg |>
  summarise(mean_hwy = mean(hwy))
```

```
# A tibble: 1 x 1
  mean_hwy
     <dbl>
1     23.4
```

Multiple values

```
mpg |>
  summarise(mean_hwy = mean(hwy), mean_cty = mean(cty))
```

```
# A tibble: 1 x 2
  mean_hwy mean_cty
     <dbl>    <dbl>
1     23.4     16.9
```

Using two columns or none

```
mpg |>
  summarise(cor = cor(cyl, hwy), count = n())
```

```
# A tibble: 1 x 2
     cor count
   <dbl> <int>
1 -0.762   234
```

Chaining with other functions

```
mpg |>
  filter(manufacturer == "audi", year == 1999) |>
  summarise(max_cyl = max(cyl), random_model = sample(model, 1))
```

```
# A tibble: 1 x 2
  max_cyl random_model
    <int> <chr>
1       6 a4
```

Exercise 7

Complete the code below to extract the rows where `name == "Khaleesi"`. Then use `summarise()` and `sum()` and `min()` to find:

1. The total number of children named Khaleesi
2. The first year Khaleesi appeared in the data

```
babynames ___
  filter(_________________________) ___
  ___________(total = ________, first = _______)
```

Calculate by groups

Seemingly does nothing

```
iris |>
  group_by(Species)
```

```
# A tibble: 150 x 5
# Groups:   Species [3]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
```

```
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
# ... with 140 more rows
```

Chained with `summarise`

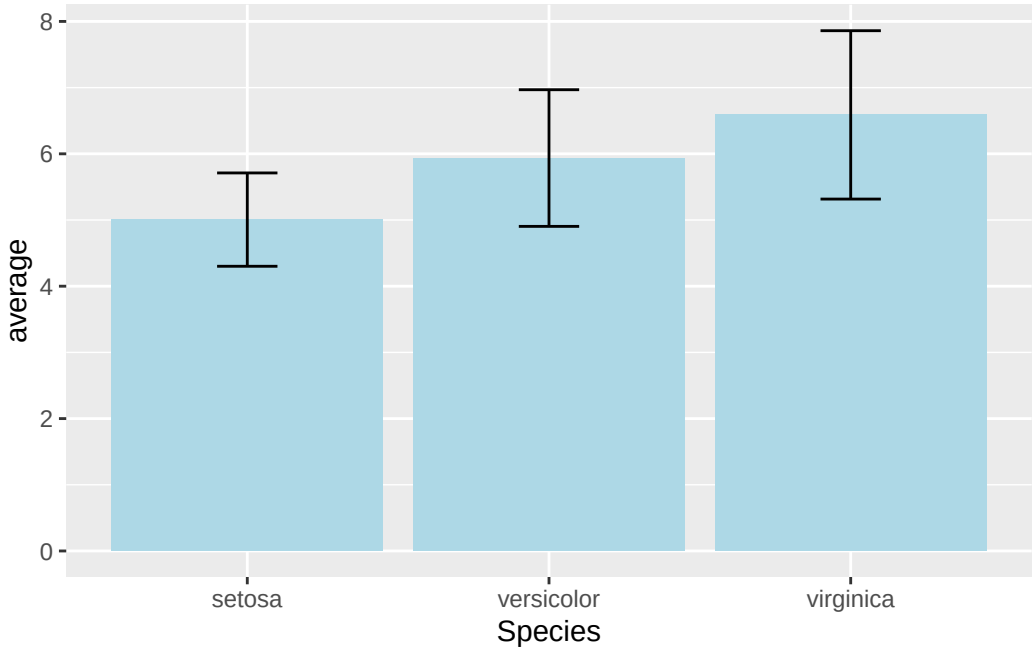
```
iris |>
  group_by(Species) |>
  summarise(mean_sepal_length = mean(Sepal.Length))
```

```
# A tibble: 3 x 2
  Species    mean_sepal_length
  <fct>                  <dbl>
1 setosa                  5.01
2 versicolor              5.94
3 virginica               6.59
```

```
data <- iris |>
  group_by(Species) |>
  summarise(average = mean(Sepal.Length),
            stdev = sd(Sepal.Length))
data
```

```
# A tibble: 3 x 3
  Species    average stdev
  <fct>        <dbl> <dbl>
1 setosa        5.01 0.352
2 versicolor    5.94 0.516
3 virginica     6.59 0.636
```

```r
ggplot(data, aes(Species, average)) +
  geom_col(fill = "lightBlue") +
  geom_errorbar(aes(ymin = average - 2 * stdev,
                    ymax = average + 2 * stdev),
                width = 0.2)
```



Exercise 8

Use `group_by()`, `summarise()`, and `arrange()` to display the ten most popular names. Compute popularity as the *total* number of children of a single gender given a name.

(Hint: Be sure to remove each `_` *before running the code)*

```r
babynames |>
  _______(name, sex) |>
  _______(total = _____(n)) |>
  _______(desc(_____))
```

Exercise 9

Use `group_by()` to calculate and then plot the total number of children born each year over time.

Mutate

Add new column

```
mpg |>
  select(manufacturer, model, year, hwy) |>
  mutate(hwy_cons = 235.215 / hwy)
```

```
# A tibble: 234 x 5
   manufacturer model       year   hwy hwy_cons
   <chr>        <chr>      <int> <int>    <dbl>
 1 audi         a4          1999    29     8.11
 2 audi         a4          1999    29     8.11
 3 audi         a4          2008    31     7.59
 4 audi         a4          2008    30     7.84
 5 audi         a4          1999    26     9.05
 6 audi         a4          1999    26     9.05
 7 audi         a4          2008    27     8.71
 8 audi         a4 quattro  1999    26     9.05
 9 audi         a4 quattro  1999    25     9.41
10 audi         a4 quattro  2008    28     8.40
# ... with 224 more rows
```

Calculate delayed variables

```
gapminder |>
  filter(country == "India") |>
  mutate(prev_pop = lag(pop))
```

```
# A tibble: 12 x 7
   country continent  year lifeExp        pop gdpPercap   prev_pop
   <fct>   <fct>     <int>   <dbl>      <int>     <dbl>      <int>
 1 India   Asia       1952    37.4  372000000      547.         NA
 2 India   Asia       1957    40.2  409000000      590.  372000000
 3 India   Asia       1962    43.6  454000000      658.  409000000
 4 India   Asia       1967    47.2  506000000      701.  454000000
 5 India   Asia       1972    50.7  567000000      724.  506000000
 6 India   Asia       1977    54.2  634000000      813.  567000000
 7 India   Asia       1982    56.6  708000000      856.  634000000
```

```
 8 India   Asia       1987    58.6  788000000      977.  708000000
 9 India   Asia       1992    60.2  872000000     1164.  788000000
10 India   Asia       1997    61.8  959000000     1459.  872000000
11 India   Asia       2002    62.9 1034172547     1747.  959000000
12 India   Asia       2007    64.7 1110396331     2452. 1034172547
```

Calculate cummulative sums

```
babynames |>
  filter(sex == "M", name == "Mike") |>
  mutate(cumcount = cumsum(n))
```

```
# A tibble: 138 x 6
    year sex   name      n     prop cumcount
   <dbl> <chr> <chr> <int>    <dbl>    <int>
 1  1880 M     Mike     95 0.000802       95
 2  1881 M     Mike     44 0.000406      139
 3  1882 M     Mike     89 0.000729      228
 4  1883 M     Mike     73 0.000649      301
 5  1884 M     Mike     84 0.000684      385
 6  1885 M     Mike     84 0.000724      469
 7  1886 M     Mike     84 0.000706      553
 8  1887 M     Mike     73 0.000668      626
 9  1888 M     Mike     81 0.000624      707
10  1889 M     Mike     72 0.000605      779
# ... with 128 more rows
```

Calculate statistics

```
babynames |>
  filter(sex == "M", name == "Mike") |>
  select(-sex, -name) |>
  mutate(mean_count = mean(n), diff = n - mean_count)
```

```
# A tibble: 138 x 5
    year     n     prop mean_count   diff
   <dbl> <int>    <dbl>      <dbl>  <dbl>
 1  1880    95 0.000802      1481. -1386.
 2  1881    44 0.000406      1481. -1437.
```

```
 3  1882    89 0.000729      1481. -1392.
 4  1883    73 0.000649      1481. -1408.
 5  1884    84 0.000684      1481. -1397.
 6  1885    84 0.000724      1481. -1397.
 7  1886    84 0.000706      1481. -1397.
 8  1887    73 0.000668      1481. -1408.
 9  1888    81 0.000624      1481. -1400.
10  1889    72 0.000605      1481. -1409.
# ... with 128 more rows
```

Exercise 10

Use `mutate()` and `min_rank()`to rank each row in `babynames` from *largest* n to lowest n.

```
# A tibble: 1,924,665 x 4
    year name          n  rank
   <dbl> <chr>     <int> <int>
 1  1880 Mary       7065  8890
 2  1880 Anna       2604 24724
 3  1880 Emma       2003 31080
 4  1880 Elizabeth  1939 31911
 5  1880 Minnie     1746 34676
 6  1880 Margaret   1578 37497
 7  1880 Ida        1472 39464
 8  1880 Alice      1414 40697
 9  1880 Bertha     1320 42842
10  1880 Sarah      1288 43636
# ... with 1,924,655 more rows
```

Exercise 11

Group `babynames` by **year** and then re-rank the data. Filter the results to just rows where `rank == 1`.

```
# A tibble: 138 x 4
# Groups:   year [138]
    year name      n  rank
   <dbl> <chr> <int> <int>
 1  1880 John   9655     1
 2  1881 John   8769     1
 3  1882 John   9557     1
```

```
 4  1883 John   8894     1
 5  1884 John   9388     1
 6  1885 Mary   9128     1
 7  1886 Mary   9889     1
 8  1887 Mary   9888     1
 9  1888 Mary  11754     1
10  1889 Mary  11648     1
# ... with 128 more rows
```

Take aways

- Extract variables with `select()`
- Extract cases with `filter()`
- Arrange cases, with `arrange()`
- Make tables of summaries with `summarise()`
- Make new variables, with `mutate()`
- Do groupwise operations with `group_by()`
- Connect operations with `%>%`