Sampling Distributions

PSYC 2020-A01 / PSYC 6022-A01 | 2025-09-26 | Lab 6

Jessica Helmer

Outline

- Assignment 5 Review
- Sampling Distributions

Learning objectives:
R: Making data tidy, line graphs

Housekeeping

Lecture the next couple weeks will be a little more theoretical

We'll be doing a bit more general data science, analytics stuff in lab

We'll be back matched up to do some modeling!

Lab 5 Review

Internal Structure of `data.frame`s

Represented as a list of vectors (vectors being the columns in our data)

Makes operating on columns easy!

Lab 5 Review

Summarizing Factor Variables

Back to the does exercise depend on substance use (smoking) question!

```
pulse_data |>
  select(ExerciseFct, SmokesFct) |>
  table()
```

```
            SmokesFct
ExerciseFct Yes No
   High       0 14
   Moderate   6 53
   Low        5 32
```

```
pulse_data |>
  select(ExerciseFct, SmokesFct) |>
  table() |>
  prop.table() |>
  round(3)
```

```
            SmokesFct
ExerciseFct   Yes    No
   High     0.000 0.127
   Moderate 0.055 0.482
   Low      0.045 0.291
```

Assignment 5 Review: Packages and Knitting

Remove or comment out `install.packages()` from your RMarkdown file when you're ready to knit

But make sure you're reading the instructions closely (including which packages to `library()`)

The knitting program doesn't know from where to install packages by default

- But no real reason to keep installation code after installing for our purposes

Assignment 5 Review: `summarize()` review

`summarize(data, .by = grouping_variable, summarized_var_1 = someSummaryFunction(var_1))`

Example: trying to summarize scores across students, by test

```
1 test_data
```

```
  student    test score
1       1    Math 93.65
2       1 Science 92.32
3       1 Reading 71.34
4       2    Math 99.55
5       2 Science 87.39
6       2 Reading 43.64
7       3    Math 99.23
8       3 Science 93.90
9       3 Reading 88.87
```

```
1 test_data |>
2   summarize(.by = test,
3             score_mean = mean(score),
4             score_sd = sd(score))
```

```
     test score_mean  score_sd
1    Math   97.47667  3.317851
2 Science   91.20333  3.395620
3 Reading   67.95000 22.804765
```

`summary()` gives us printed output, `summarize()` gives us a condensed dataframe that we can keep working with

Sampling Distributions

Important topic in statistics! Help us make inferences about the estimates we calculate

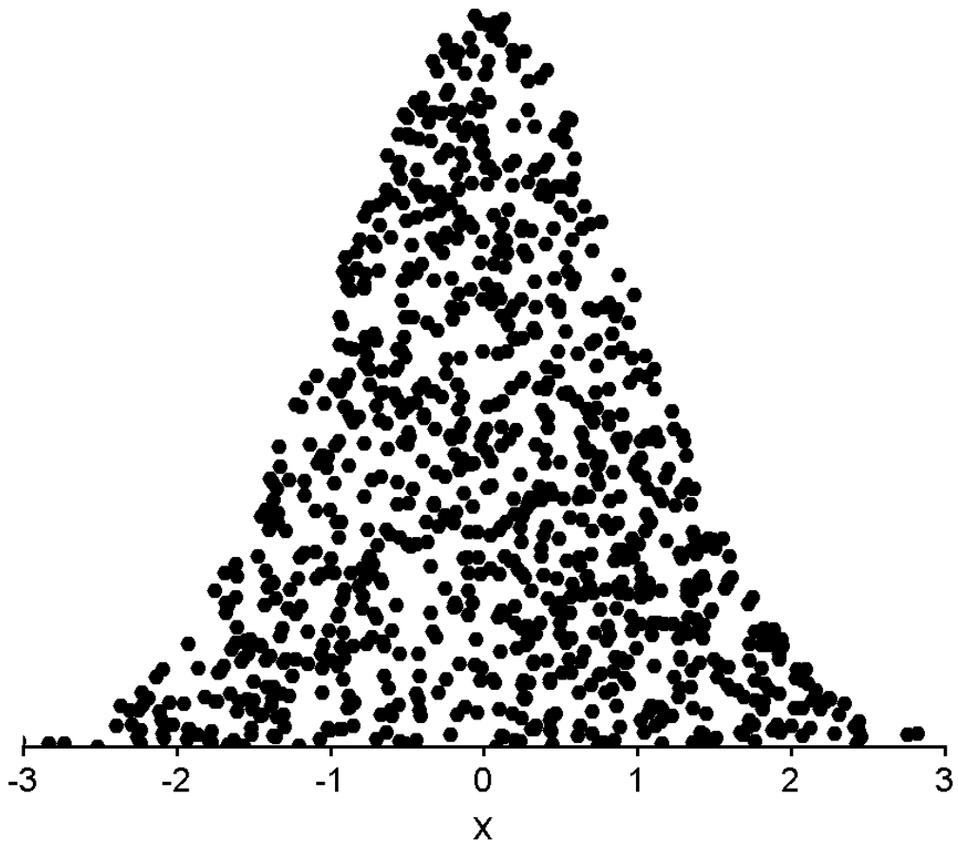
Sample Distribution

Distribution of the data within our sample

Sample Distribution

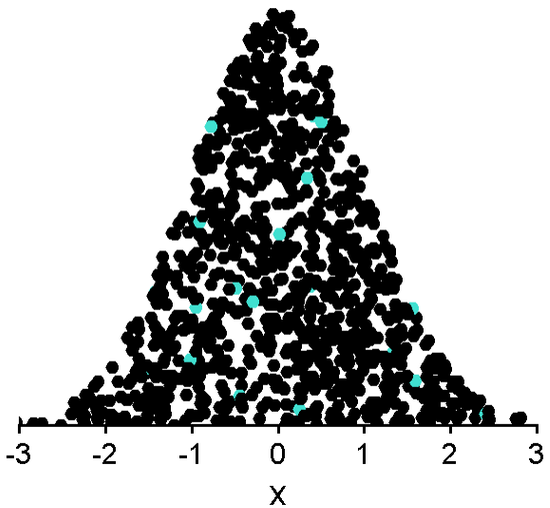
(More theoretical) distribution of our statistic over repeated sampling

Population

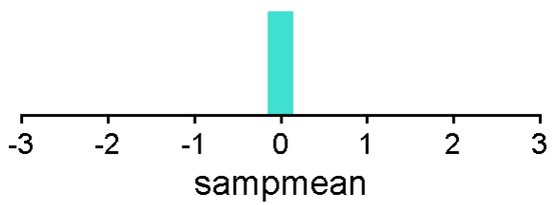


Sample

```
1  sampdist()
```

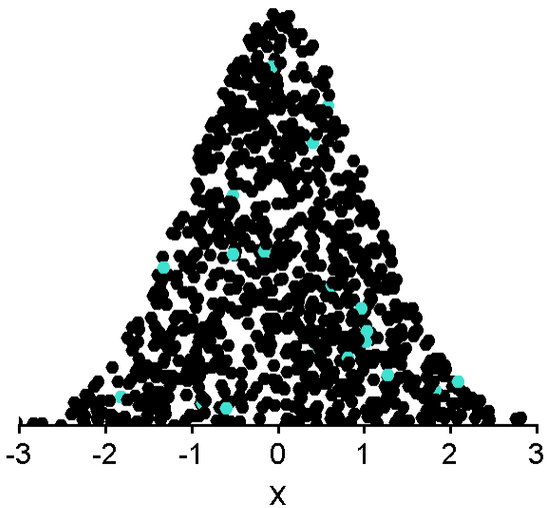



```
1  samplingdist()
```

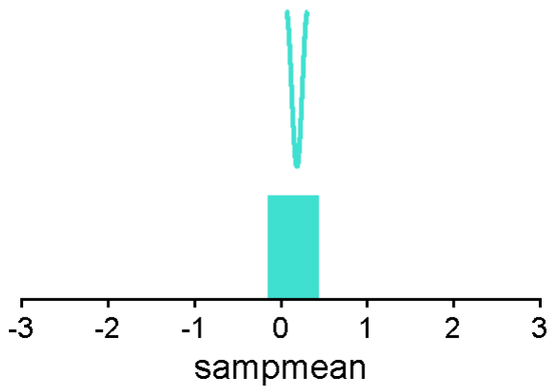


Sample

```
1  sampdist()
```

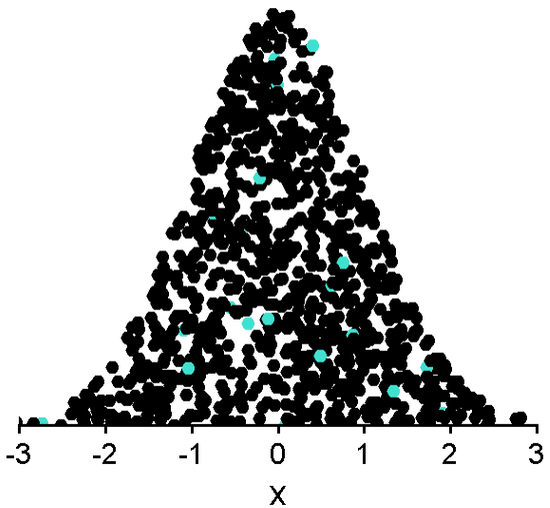



```
1  samplingdist()
```

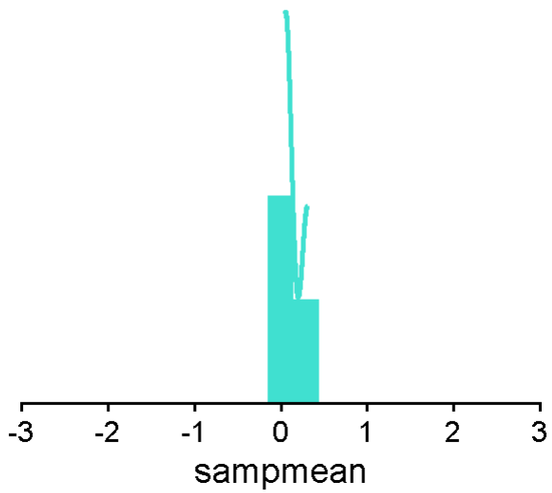


Sample

```
1  sampdist()
```

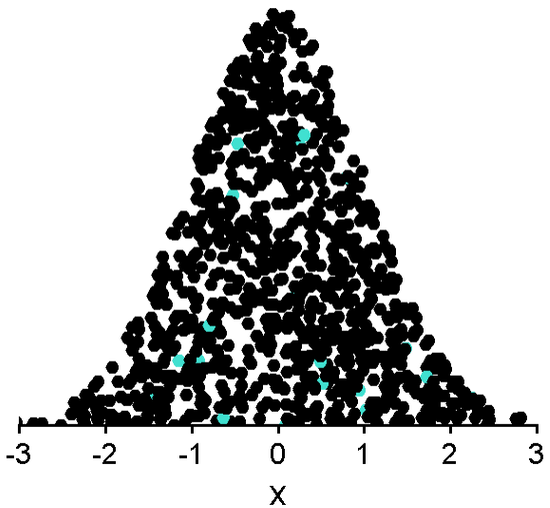



```
1  samplingdist()
```

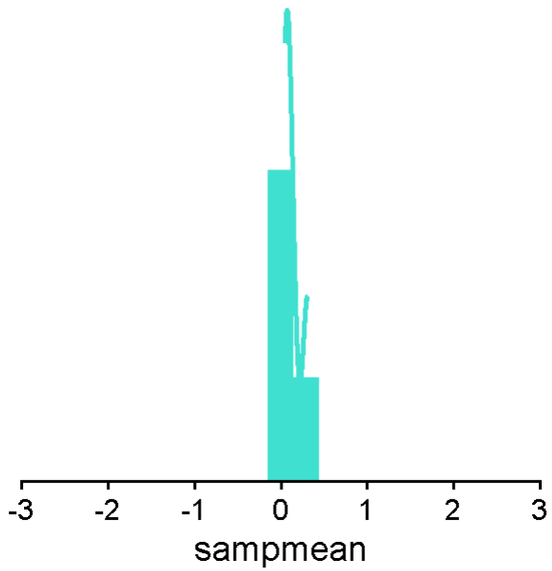


Sample

```
1  sampdist()
```

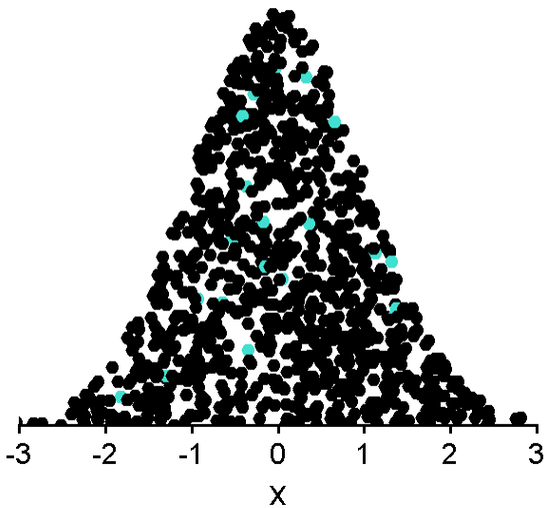



```
1  samplingdist()
```

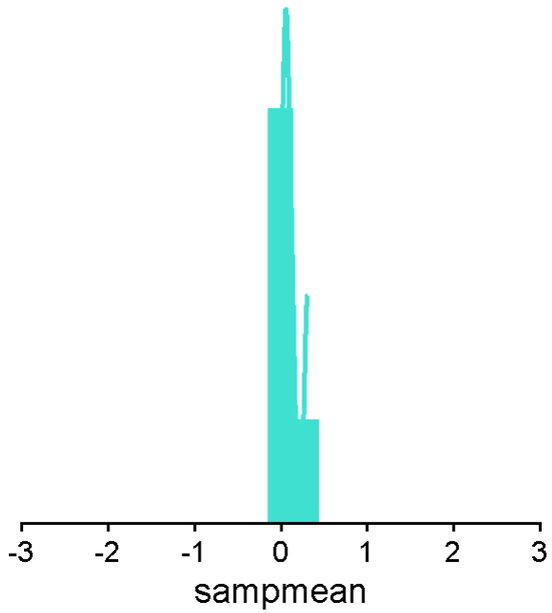


Sample

```
1  sampdist()
```

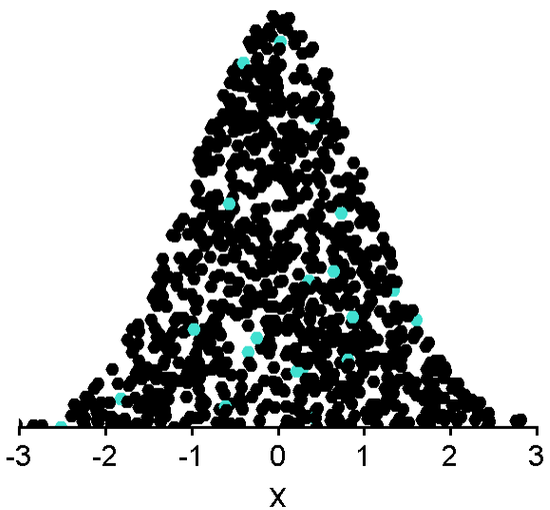



```
1  samplingdist()
```

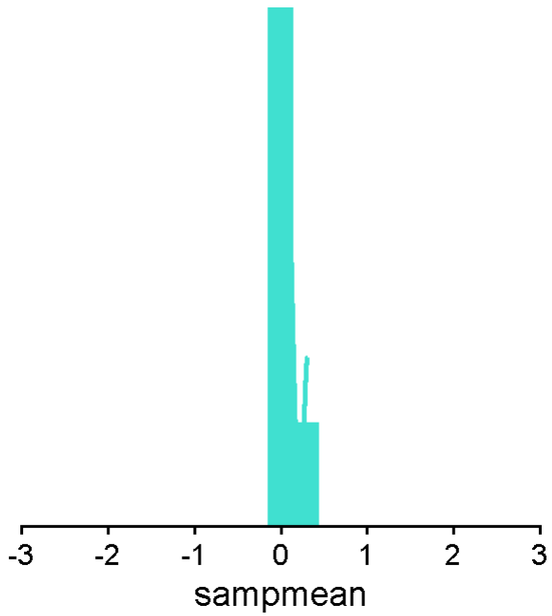


Sample

```
1  sampdist()
```

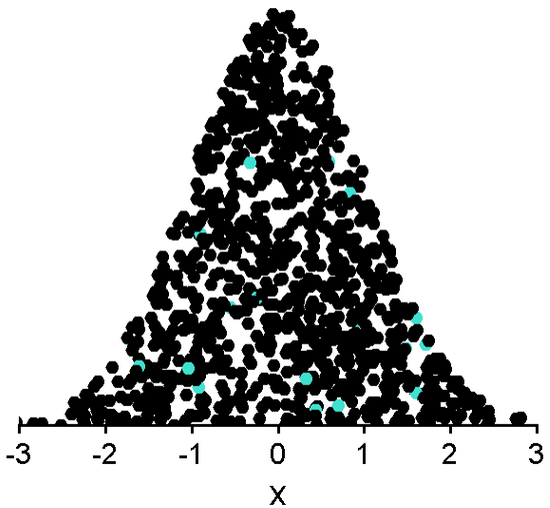



```
1  samplingdist()
```

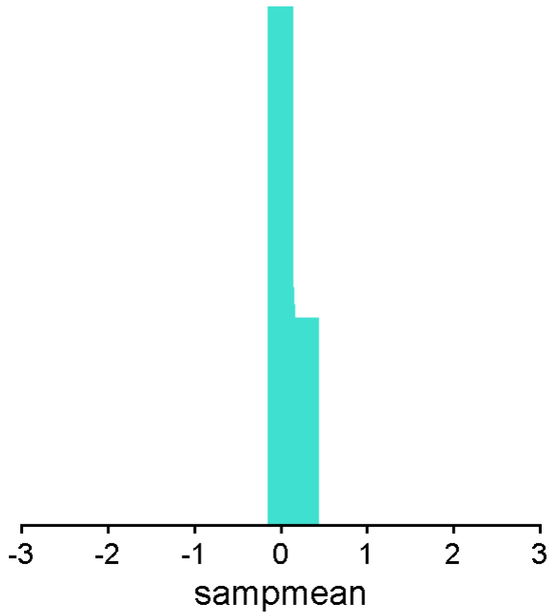


Sample

```
1 sampdist()
```

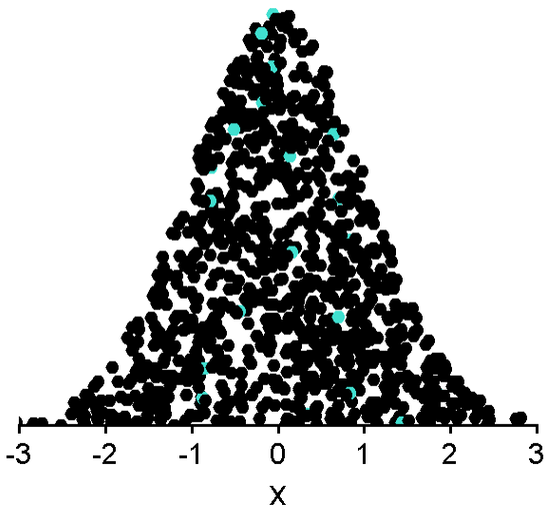



```
1 samplingdist()
```

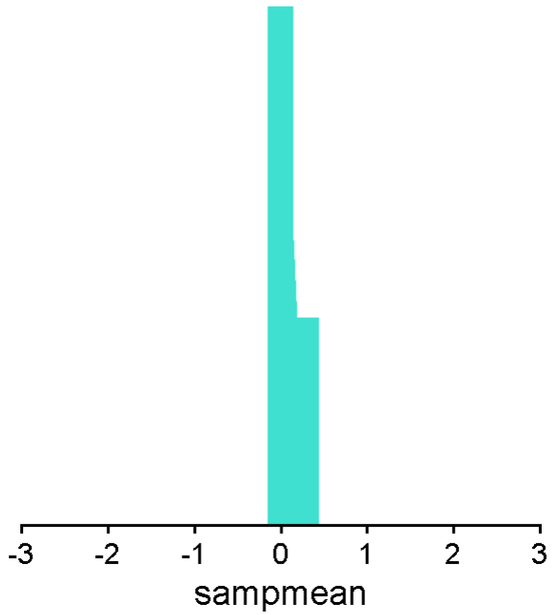


Sample

```
1  sampdist()
```

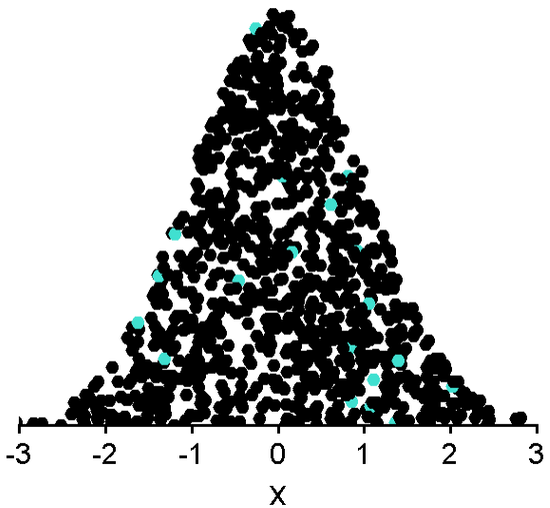



```
1  samplingdist()
```

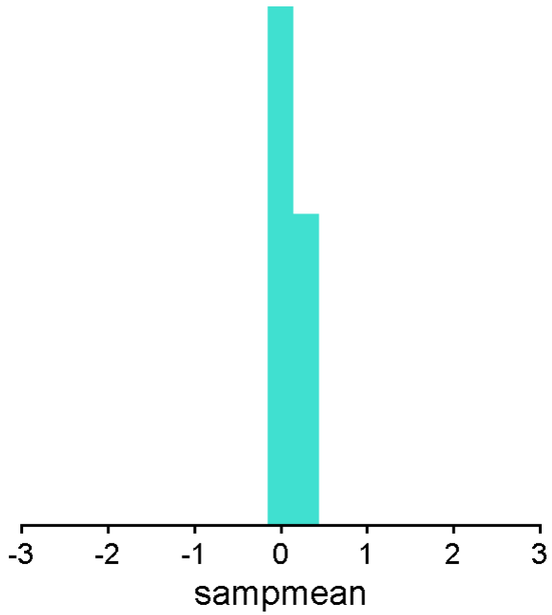


Sample

```
1  sampdist()
```

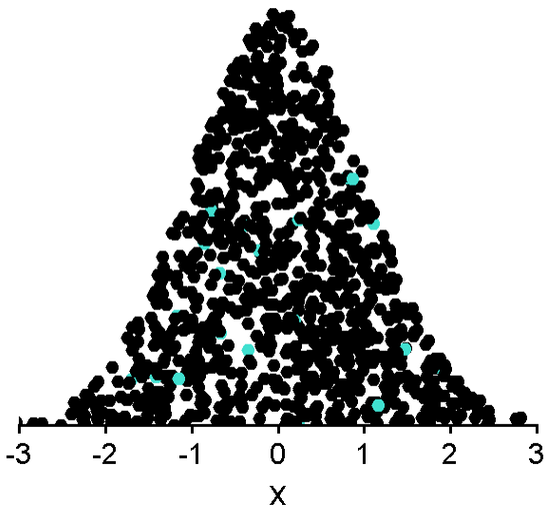



```
1  samplingdist()
```

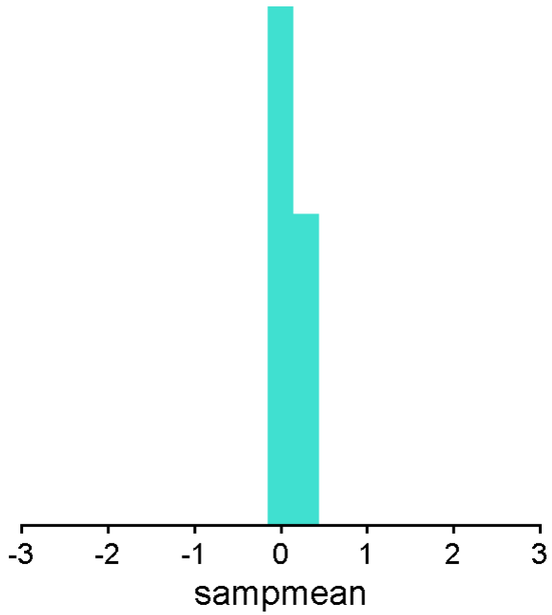


Sample

```
1  sampdist()
```




```
1  samplingdist()
```



Tidy Data

In a lot of cases, your data will not come in tidy

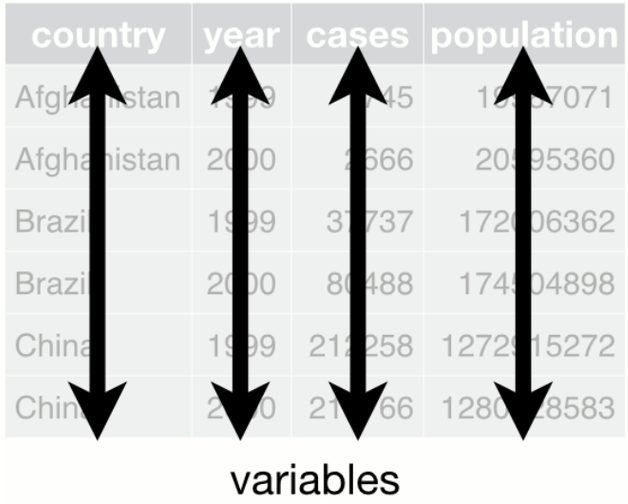
Need to know how to work with this!

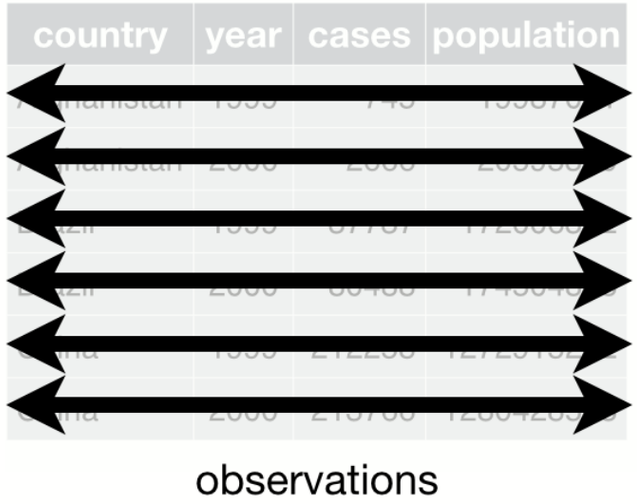
Let Us Review…

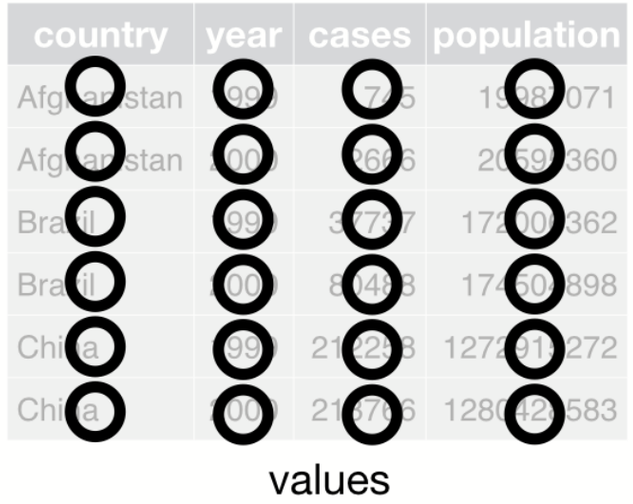
What are the tidy principles?

Three rules to a tidy dataset

1. Each variable is a column; each column is a variable.
2. Each observation is a row; each row is an observation.
3. Each value is a cell; each cell is a single value.


country year cases population
variables


country year cases population
observations


country year cases population
values

Long vs. Wide Data

Long: each observation is a row (tidy)

- Observation does not necessarily mean “person”
- Can be within-person measurements over time

Wide: columns might represent different “values” of a variables

Long vs. Wide Data

```
# A tibble: 9 × 3
  person   book   score
  <chr>    <chr>  <int>
1 Harry    book 1     1
2 Harry    book 2     3
3 Harry    book 3     3
4 Ron      book 1     3
5 Ron      book 2     3
6 Ron      book 3     1
7 Hermione book 1     2
8 Hermione book 2     3
9 Hermione book 3     1
```

```
# A tibble: 3 × 4
  person   `book 1` `book 2` `book 3`
  <chr>       <int>    <int>    <int>
1 Harry           1        3        3
2 Ron             3        3        1
3 Hermione        2        3        1
```

Wide data common in spreadsheets, data from survey platforms

Long data typically easier to work with

Key Functions

Wide to Long

pivot_longer()

```
longdat <- dat %>%
  pivot_longer(cols = c("book 1", "book 2", "book
               names_to = "book", values_to = "sc
longdat
```

```
# A tibble: 9 × 3
  person   book   score
  <chr>    <chr>  <int>
1 Harry    book 1     1
2 Harry    book 2     3
3 Harry    book 3     3
4 Ron      book 1     3
5 Ron      book 2     3
6 Ron      book 3     1
7 Hermione book 1     2
8 Hermione book 2     3
9 Hermione book 3     1
```

Long to Wide

pivot_wider()

```
widedat <- longdat %>%
  pivot_wider(names_from = book, values_from = sc
widedat
```

```
# A tibble: 3 × 4
  person   `book 1` `book 2` `book 3`
  <chr>       <int>    <int>    <int>
1 Harry           1        3        3
2 Ron             3        3        1
3 Hermione        2        3        1
```

Key Functions

Wide to Long

`pivot_longer()`

- `cols` = set of columns to pivot
- `names_to` = name of new column with the column names from before
- `values_from` = name of new column with the values from before

Long to Wide

`pivot_wider()`

- `names_from` = column with values to be column names
- `values_from` = column with values to be in those columns

Some Examples: Example 1

```
1 table1
```

```
# A tibble: 6 × 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

Looks good!

Some Examples: Example 2

```
1 table2
```

```
# A tibble: 12 × 4
   country      year type            count
   <chr>       <dbl> <chr>           <dbl>
 1 Afghanistan  1999 cases             745
 2 Afghanistan  1999 population   19987071
 3 Afghanistan  2000 cases            2666
 4 Afghanistan  2000 population   20595360
 5 Brazil       1999 cases           37737
 6 Brazil       1999 population  172006362
 7 Brazil       2000 cases           80488
 8 Brazil       2000 population  174504898
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
```

To RStudio!!

Some Examples: Example 2

Example code

```
1 table2 |>
2   pivot_wider(names_from = type, values_from = count)
```

```
# A tibble: 6 × 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

Some Examples: Example 3

```
1 table3
```

```
# A tibble: 6 × 3
  country      year rate
  <chr>       <dbl> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

It's unfortunately pretty common to get data that has multiple values in a cell!

New function!

`separate_wider_delim()`

- `delim` = character to split on

To RStudio!!

Some Examples: Example 3

Example code

```r
table3 |>
  separate_wider_delim(cols = rate, delim = "/",
                       names = c("cases", "population")) |>
  mutate(cases = as.numeric(cases),
         population = as.numeric(population),
         rate = cases / population)
```

```
# A tibble: 6 × 5
  country      year  cases population      rate
  <chr>       <dbl>  <dbl>      <dbl>     <dbl>
1 Afghanistan  1999    745   19987071 0.0000373
2 Afghanistan  2000   2666   20595360 0.000129
3 Brazil       1999  37737  172006362 0.000219
4 Brazil       2000  80488  174504898 0.000461
5 China        1999 212258 1272915272 0.000167
6 China        2000 213766 1280428583 0.000167
```

`as.numeric()` attempts to convert to numeric data type

If you create columns in a `mutate()` call, you can refer to those columns

in the same call after you create them

Some Examples: Example 4

```
1 table4a
2 table4b
```

```
# A tibble: 3 × 3
  country     `1999` `2000`
  <chr>        <dbl>  <dbl>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

```
# A tibble: 3 × 3
  country         `1999`     `2000`
  <chr>            <dbl>      <dbl>
1 Afghanistan   19987071   20595360
2 Brazil       172006362  174504898
3 China       1272915272 1280428583
```

Sometimes we get data in multiple pieces

New function!

`full_join()`

- `x` = left dataframe to join
- `y` = right dataframe to join
- `by` = character, matching column(s) in both dataframes
- 'join by' columns need to have exact same name in both

To RStudio!!

Some Examples: Example 4

Example code

```
table4a_long <- table4a %>%
  pivot_longer(c("1999", "2000"),
               names_to = "year", values_to = "ca

table4b_long <- table4b %>%
  pivot_longer(c("1999", "2000"),
               names_to = "year", values_to = "ca

full_join(table4a_long, table4b_long, by = c("cou
```

```
# A tibble: 6 × 4
  country     year  cases.x    cases.y
  <chr>       <chr>   <dbl>      <dbl>
1 Afghanistan 1999      745   19987071
2 Afghanistan 2000     2666   20595360
3 Brazil      1999    37737  172006362
4 Brazil      2000    80488  174504898
5 China       1999   212258 1272915272
6 China       2000   213766 1280428583
```

```
full_join(
  table4a %>%
    pivot_longer(c("1999", "2000"),
                 names_to = "year", values_to = '
  table4b %>%
    pivot_longer(c("1999", "2000"),
                 names_to = "year", values_to = '
  by = c("country", "year")
)
```

```
# A tibble: 6 × 4
  country     year  cases.x    cases.y
  <chr>       <chr>   <dbl>      <dbl>
1 Afghanistan 1999      745   19987071
2 Afghanistan 2000     2666   20595360
3 Brazil      1999    37737  172006362
4 Brazil      2000    80488  174504898
5 China       1999   212258 1272915272
6 China       2000   213766 1280428583
```

Some Examples: Example 5

```
1 table5
```

```
# A tibble: 6 × 4
  country     century year  rate
  <chr>       <chr>   <chr> <chr>
1 Afghanistan 19      99    745/19987071
2 Afghanistan 20      00    2666/20595360
3 Brazil      19      99    37737/172006362
4 Brazil      20      00    80488/174504898
5 China       19      99    212258/1272915272
6 China       20      00    213766/1280428583
```

`paste()` and `paste0()` combine characters into one string

- `paste()` adds a space between inputs
- `paste0()` does not

To RStudio!!

Some Examples: Example 5

Example code

```
table5 |>
  separate_wider_delim(cols = rate, delim = "/",
                       names = c("cases", "population")) |>
  mutate(cases = as.numeric(cases),
         population = as.numeric(population),
         rate = cases / population,
         year = paste0(century, year) |> as.numeric(),
         .keep = "unused")
```

```
# A tibble: 6 × 5
  country      year  cases population      rate
  <chr>       <dbl>  <dbl>      <dbl>     <dbl>
1 Afghanistan  1999    745   19987071 0.0000373
2 Afghanistan  2000   2666   20595360 0.000129
3 Brazil       1999  37737  172006362 0.000219
4 Brazil       2000  80488  174504898 0.000461
5 China        1999 212258 1272915272 0.000167
6 China        2000 213766 1280428583 0.000167
```

Selecting Functions

Sometimes, we want to select many columns (to pivot, to remove, etc.)

Notice the documentation for `pivot_longer()`

`<tidy-select>` functions

- `starts_with()` and `ends_with()`

Selecting Functions

Simple example, but…

```r
dat <- data.frame(id = 1:5,
                  Q1 = rbinom(5, 1, .1),
                  Q2 = rbinom(5, 1, .2),
                  Q3 = rbinom(5, 1, .3),
                  Q4 = rbinom(5, 1, .4),
                  Q5 = rbinom(5, 1, .5))
dat
```

```
  id Q1 Q2 Q3 Q4 Q5
1  1  1  0  0  0  0
2  2  0  0  1  0  0
3  3  0  1  1  0  0
4  4  0  0  0  1  0
5  5  0  1  0  1  1
```

```r
dat |>
  pivot_longer(cols = starts_with("Q"),
               names_to = "question", values_
```

```
# A tibble: 25 × 3
      id question correct
   <int> <chr>      <int>
 1     1 Q1             1
 2     1 Q2             0
 3     1 Q3             0
 4     1 Q4             0
 5     1 Q5             0
 6     2 Q1             0
 7     2 Q2             0
 8     2 Q3             1
 9     2 Q4             0
10     2 Q5             0
# ℹ 15 more rows
```

Visualize: ggplot2 line graphs

Let's see some chicks grow!

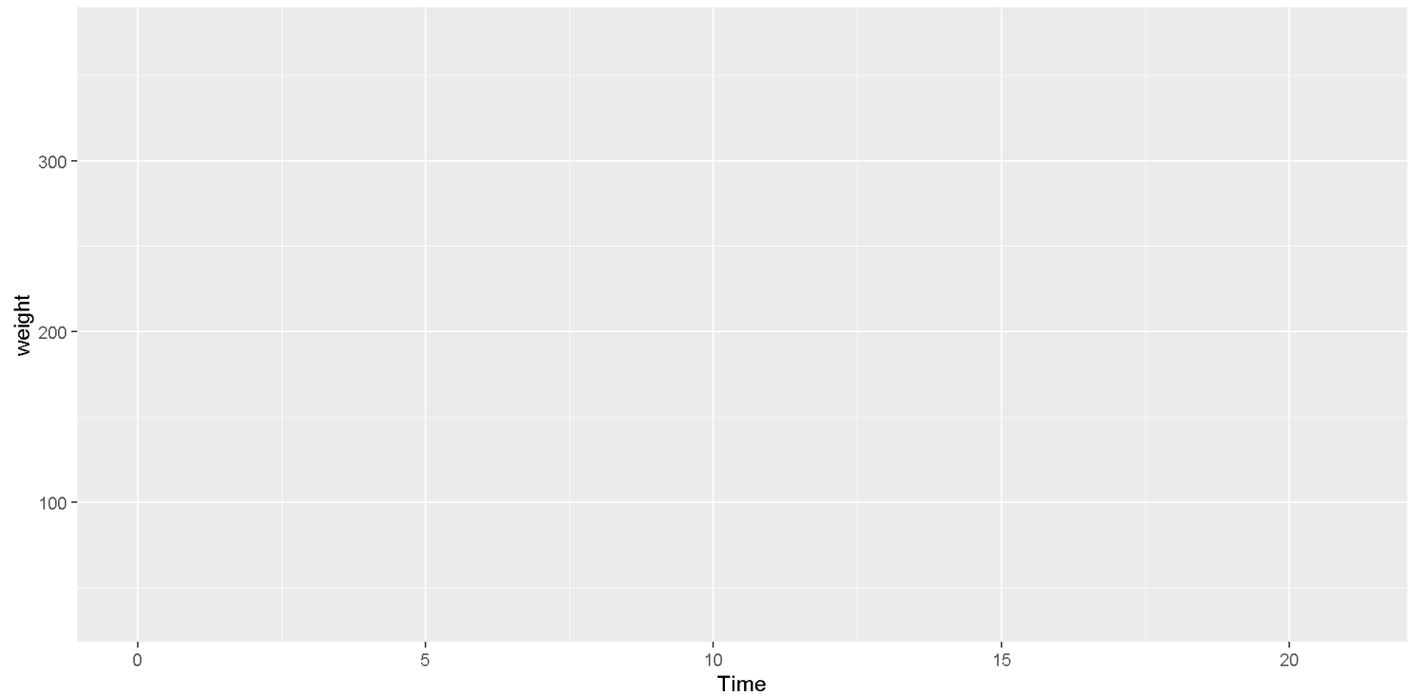
```
1  ChickWeight |> head(20)
```

```
   weight Time Chick Diet
1      42    0     1    1
2      51    2     1    1
3      59    4     1    1
4      64    6     1    1
5      76    8     1    1
6      93   10     1    1
7     106   12     1    1
8     125   14     1    1
9     149   16     1    1
10    171   18     1    1
11    199   20     1    1
12    205   21     1    1
13     40    0     2    1
```

Visualize: `ggplot2` line graphs

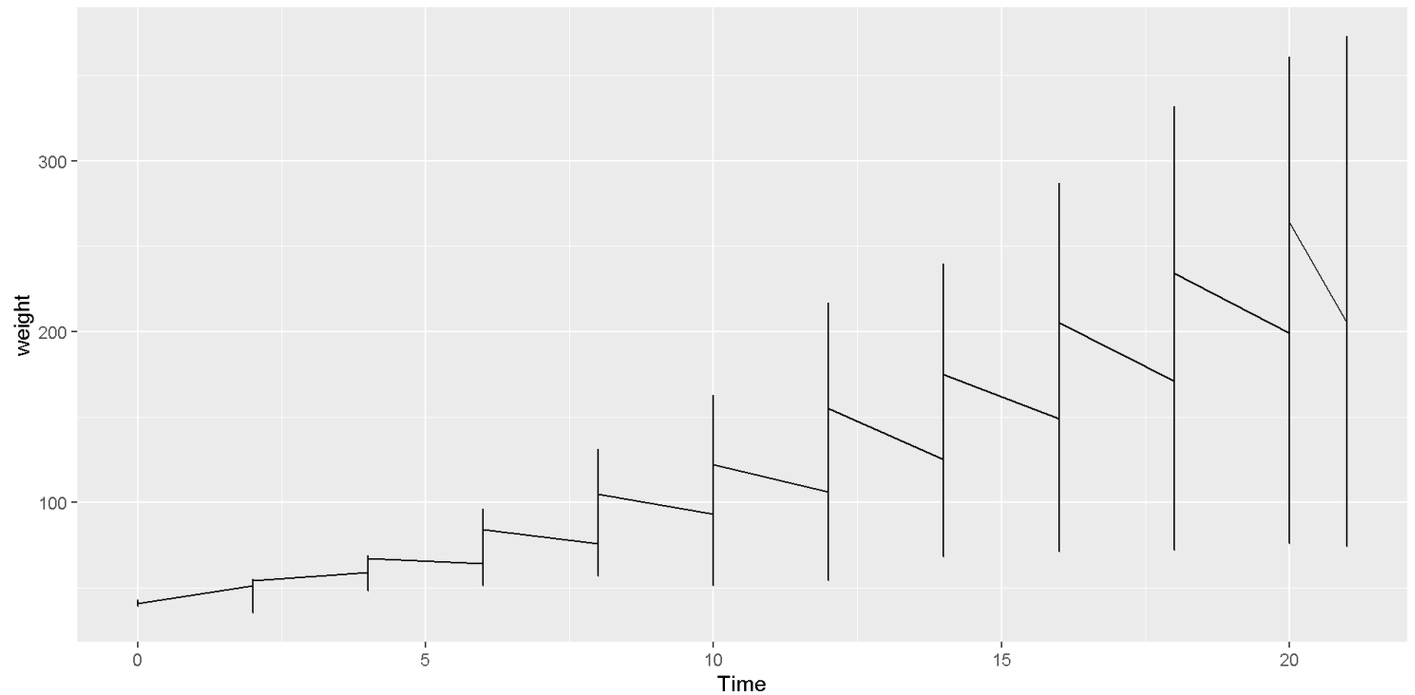
Plot | Code

Start like we did before, with just a grid



Visualize: `ggplot2` line graphs

Plot | Code



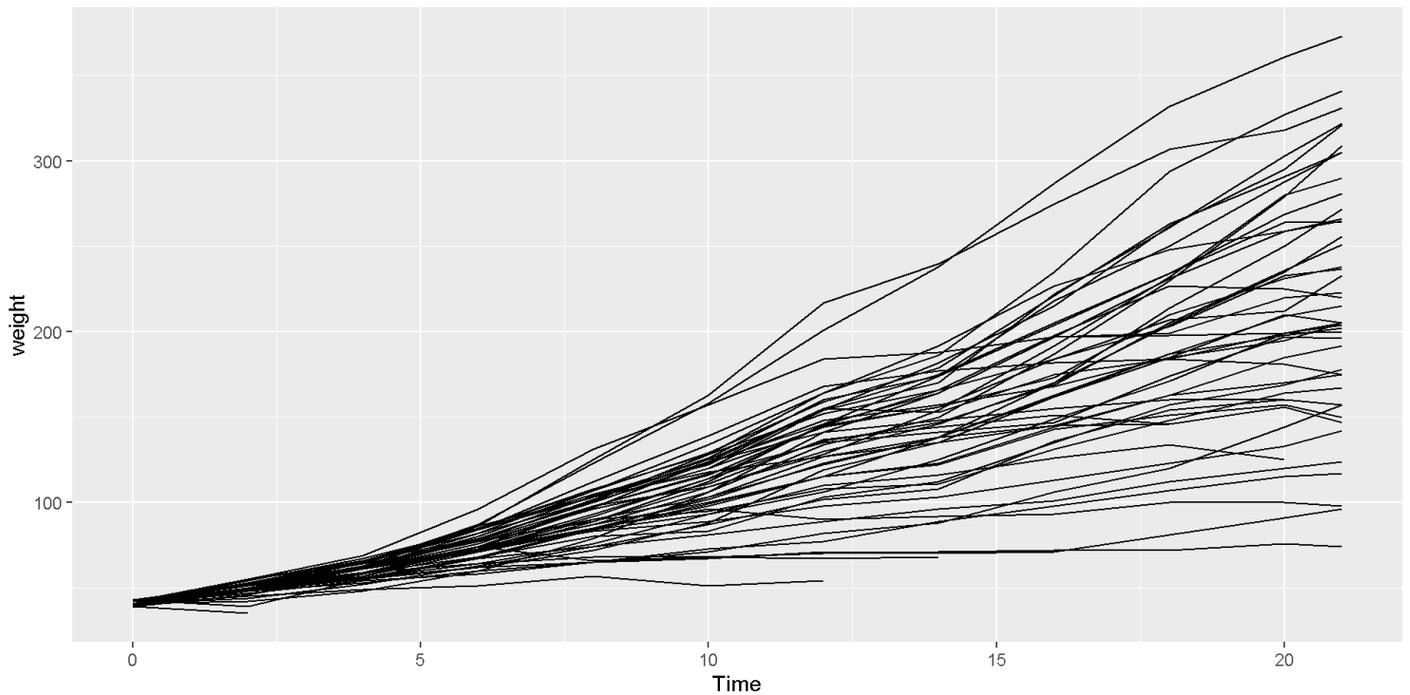
Add a `geom_line()`

Looks wacky!

If we want different lines for different groups of observations (here, different chicks), need to specify the `group` argument within the `aes()` function

Visualize: `ggplot2` line graphs

Plot | Code



What can we see from this?

Any critiques / limitations?

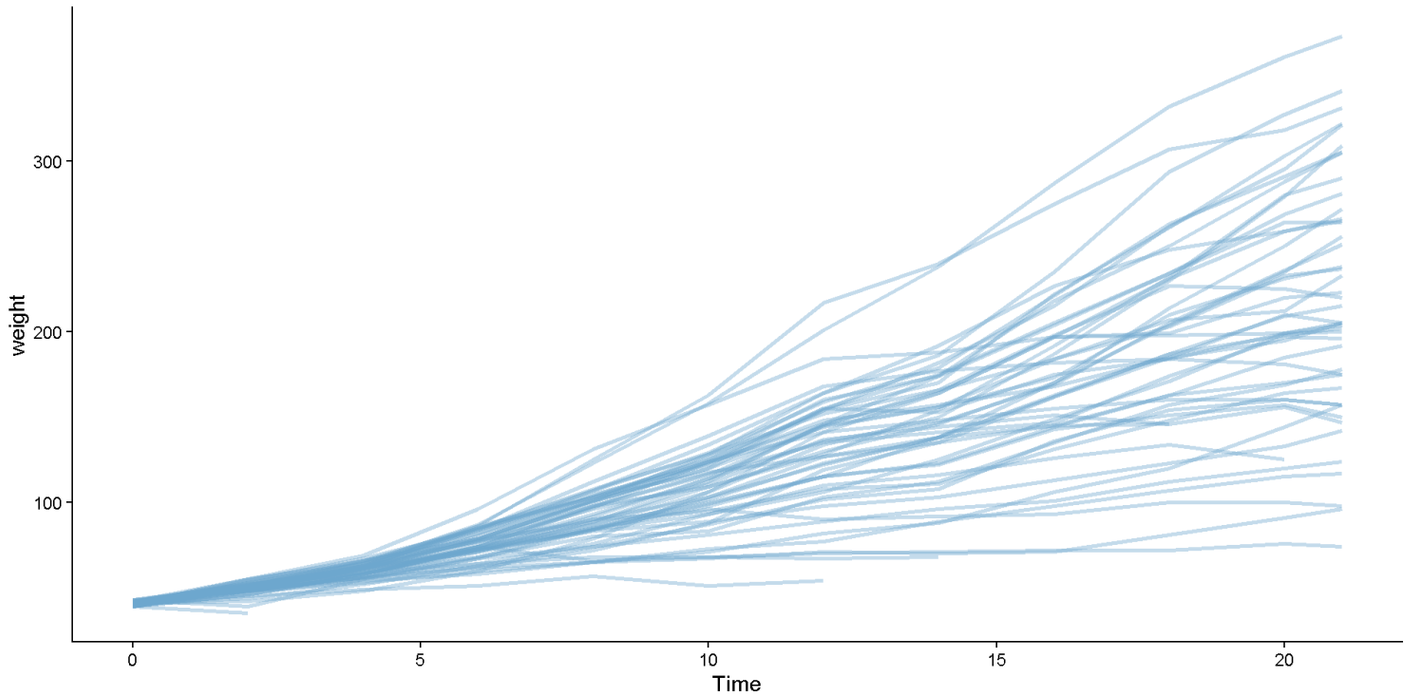
Hard to see individual lines in beginnings

Can modify appearance of our geoms

- `alpha` = number [0, 1] representing opacity

Visualize: `ggplot2` line graphs

Plot | Code

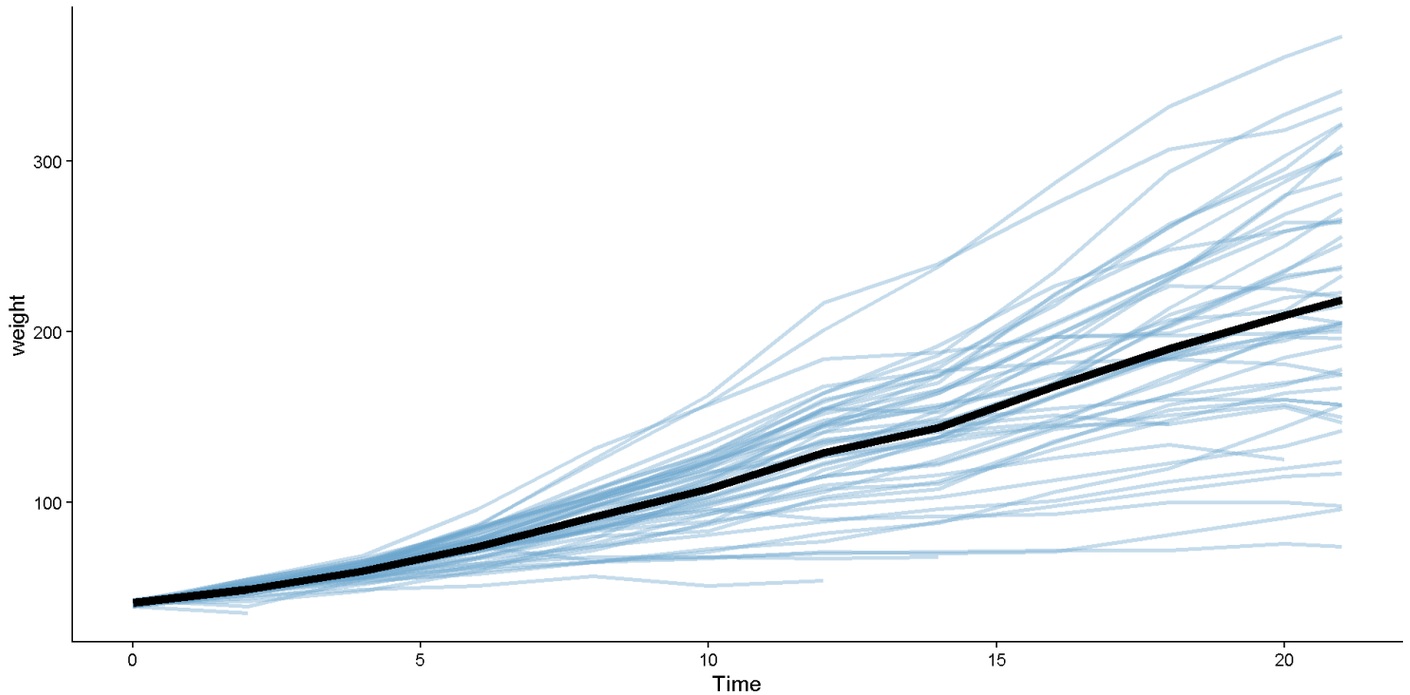


Changed theme to better see color

Line representing the whole trend?

Visualize: `ggplot2` line graphs

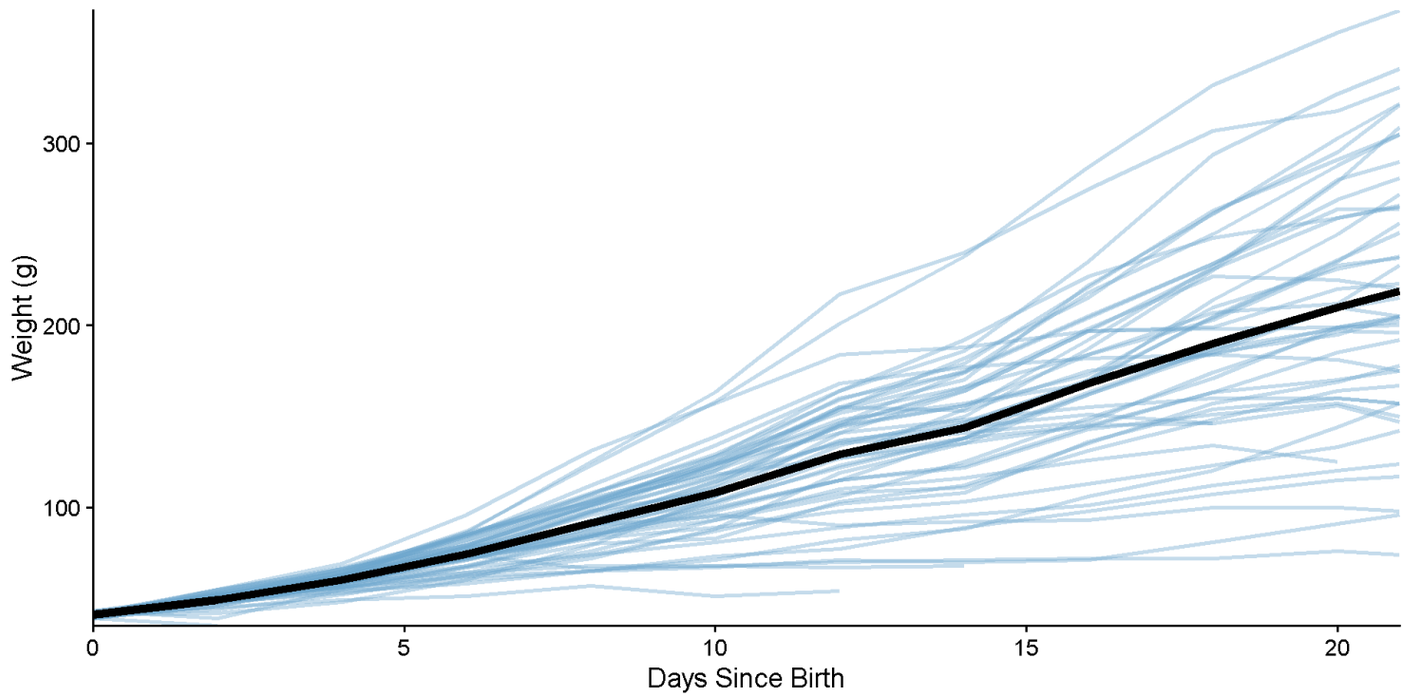
Plot | Code



Great! Now let's do some finishing touches

Visualize: `ggplot2` line graphs

Plot | Code



Great! Now let's do some finishing touches

Assignment 6