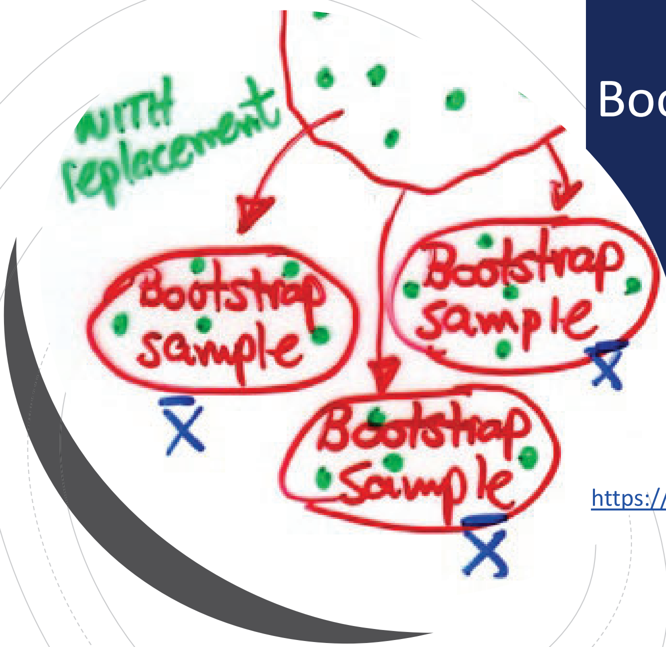
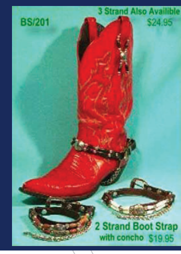


Bootstrapping



Michael Friendly
Psych 6135
https://friendly.github.io/6135/

#psy6135

Bootstrapping

- Classical statistical inference relies on
  - Distributional assumptions, e.g., $\varepsilon \sim N(0, \sigma^2)$
  - Asymptotic results, e.g., in SEM: $F_{ML} \sim \chi 2$ as $n \rightarrow \infty$
- Bootstrapping is a non-parametric approach to inference that substitutes computation for assumptions

Functional bootstraps: help to pull you up from where you are (data), to where you want to be (reasonable conclusions)

bootstrap (v): help oneself, often through improvised means

Decorative bootstraps: we don't need these

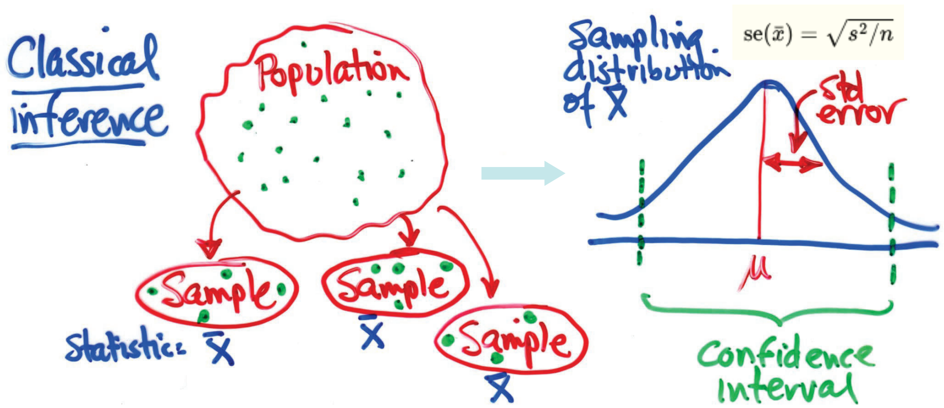
Bootstrapping

- Can provide more accurate inferences when data is badly behaved or *n* is small
  - linear models, SEM, …
- Can be applied when *no* sampling theory is available
  - Tests of equality of ratios: $(y_1/x_1) =_? (y_2/x_2)$
  - fMRI studies: differences among patterns of brain activation
  - Shoeless Joe Jackson: how did he hit in clutch situations?
- Can be applied to complex data-collection plans (stratified/clustered samples)

More general ideas: Resampling

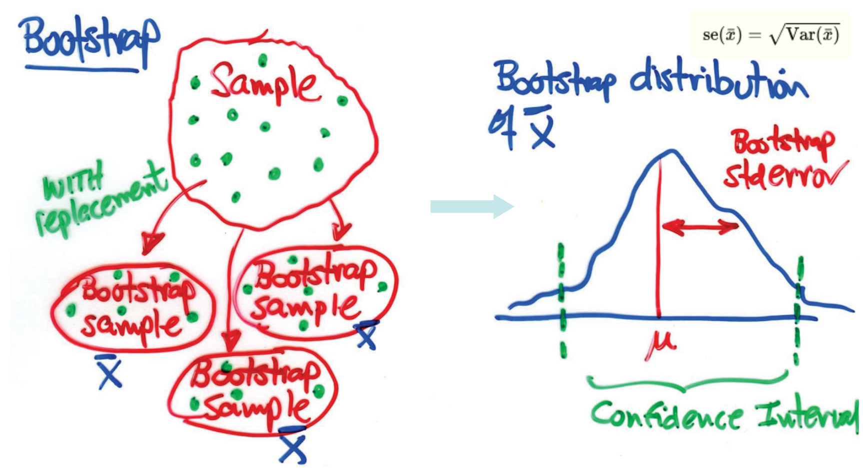
- The bootstrap is an example of the general idea of ***resampling*** from an original data set for statistical inference
- Other examples:
  - Jackknife: leave-one-out analysis
  - Cross-validation: choosing optimal model fitting parameters
  - Permutation tests: totally non-parametric
- Uses:
  - Std errors, CIs with small samples
  - Subset selection in linear models
  - Dealing with missing data
  - Complex algorithms: ML neural networks

Classical statistical inference



Here, we rely on statistical theory (CLT) & assumptions (independence, normality, constant variance) to take us to the sampling distribution of the statistic of interest.

Bootstrap



Key idea:

Population is to the sample	AS	Sample is to the bootstrap sample

Bootstrap resampling demo

```
# devtools::install_github("wilkelab/ungeviz")
library(ungeviz)
bs <- bootstrapper(3)        # create 3 draws
(draws <- bs(data.frame(letter = LETTERS[1:4])))
```

The bootstrapper function creates a function to create bootstrap samples
-- here 3 draws of 4 letters

```
# A tibble: 12 x 6
# Groups:   .draw [3]
   .draw   .id .original_id letter .copies  .row
   <int> <int>        <int> <chr>    <dbl> <int>
 1     1     1            1 A            1     1
 2     1     2            4 D            2     2
 3     1     3            4 D            2     3
 4     1     4            3 C            1     4
 5     2     1            4 D            1     5
 6     2     2            1 A            2     6
 7     2     3            1 A            2     7
 8     2     4            2 B            1     8
 9     3     1            1 A            1     9
10     3     2            2 B            1    10
11     3     3            3 C            2    11
12     3     4            3 C            2    12
```

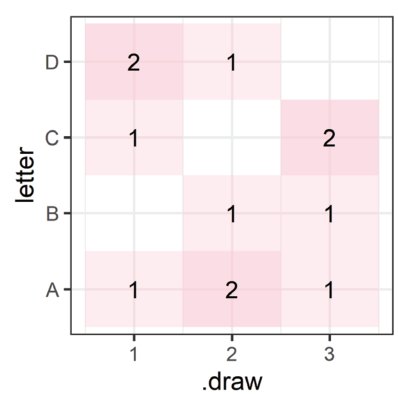
letter is the data value.

Other variables identify all aspects of the bootstrap

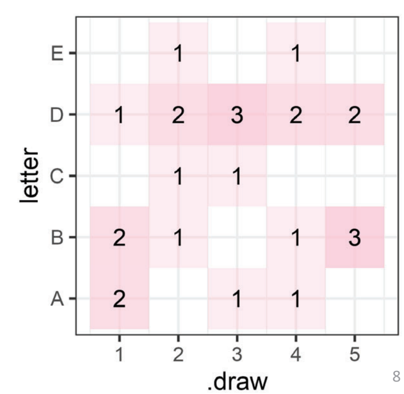
Bootstrap resampling demo

```
ggplot(draws, aes(x=.draw, y=letter)) +
    geom_tile(fill="pink", alpha=0.3) +
    geom_text(aes(label=.copies), size=6)
```

Each tile shows the number of times that letter was picked in a given .draw



The same for 5 draws of LETTERS[1:5]

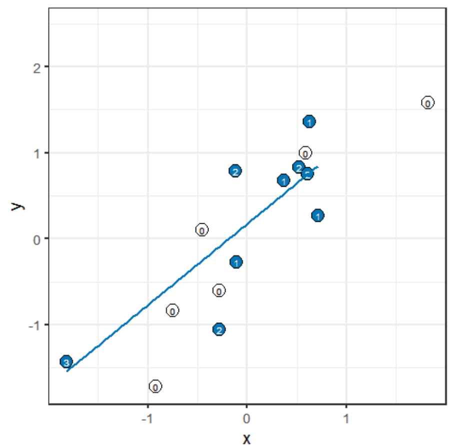


Regression illustration

```
# randomly generate dataset
set.seed(12345)
x <- rnorm(15)
df <- data.frame(x,
                 y = x + 0.5*rnorm(15))
# bootstrapper object
bsr <- bootstrapper(10)
```

Animated plot, by .draw:

```
ggplot(df, aes(x,y) +
  geom_point(data=bsr, aes(group= .row) +
  geom_text(data = bsr, aes(label = .copies) +
  geom_smooth(data = bsr, aes(group = .draw),
              method = "lm“) +
  # animation
  transition_states(.draw, 1, 2) +
  enter_fade() + exit_fade()
```

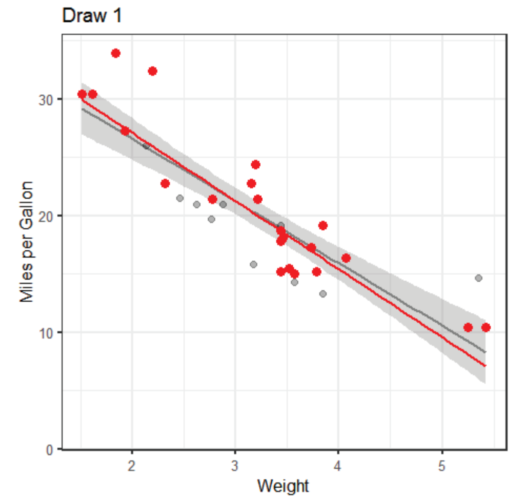


Bootstrapped confidence bands

The same method can be used to illustrate the uncertainty around the regression line, as reflected in the confidence band

However, the std conf. band is calculated using classical normal theory

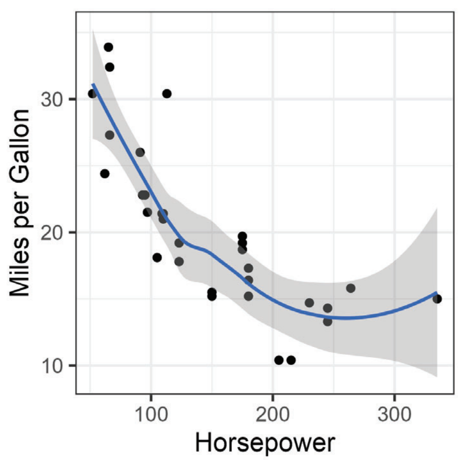
The bootstrapped fits trace out an empirical confidence band.



Non-linear relations: smoothing

We know how to use loess to estimate a non-parametric smoothed curve
There is also theory that allows calculation of a (approx.) confidence envelope

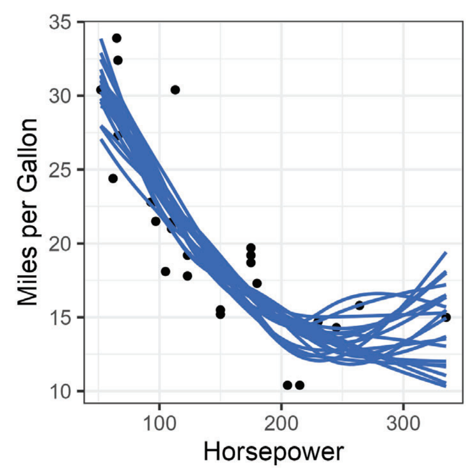
```
ggplot(mtcars, aes(hp, mpg)) +
  geom_point(size = 2) +
  geom_smooth(method = "loess") +
  labs(x = "Horsepower",
       y = "Miles per Gallon")
```



Resampling: smooth draws

Instead, resampling methods generate outcome draws from a smooth fit using mgcv::gam(). The collection of draws provide an empirical confidence envelope

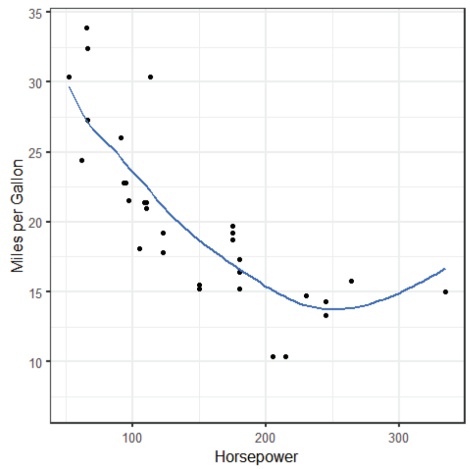
```
plt <-
ggplot(mtcars, aes(hp, mpg)) +
  geom_point(size = 2) +
  stat_smooth_draws(times = 20,
         aes(group = stat(.draw) ))

plt
```



Resampling: smooth draws

Animation shows how the collection of sampled smooths develop over time
The animation transitions over draws (.draw)
shadow_trail() keeps the previous curves

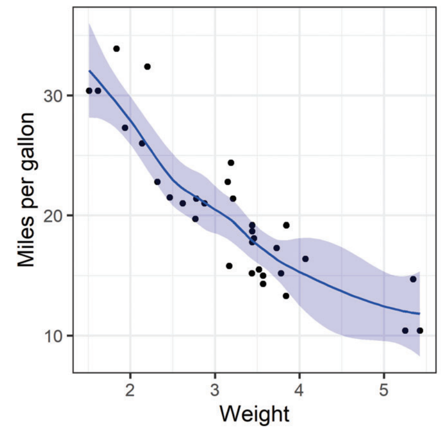
```
plt +
  transition_states(stat(.draw)) +
  enter_fade() +
  exit_fade(alpha=0.8) +
  shadow_trail()
```



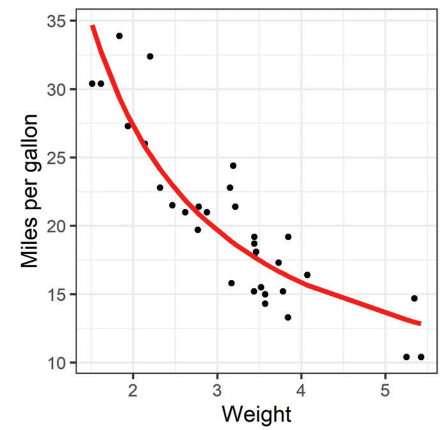
Bootstrapping models

Rather than fitting a nonparametric smoothed curve, we might want to fit a parametric but nonlinear model, perhaps for substantive interpretation

loess: nonparametric



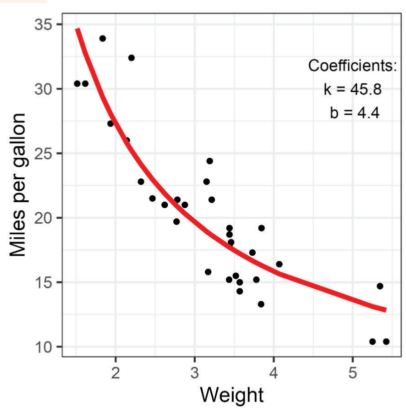
An inverse relation: $y = \frac{k}{x} + b$



Nonlinear model: nls()

```
nlsfit <- nls(mpg ~ k / wt + b, mtcars,
               start = list(k = 1, b = 0))
summary(nlsfit)
```

```
Formula: mpg ~ k/wt + b

Parameters:
  Estimate Std. Error t value Pr(>|t|)
k   45.829      4.249  10.786 7.64e-12 ***
b    4.386      1.536   2.855  0.00774 **
```

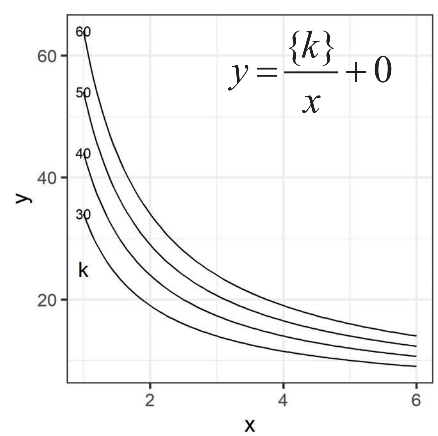


This uses stats::nls() to fit nonlinear models
There is also a {nlstools} package (that does bootstrapping)

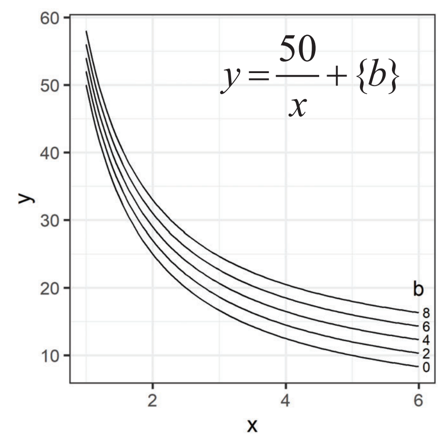
Inverse model

What are the parameters in this model?

k: starting value



b: asymptote



rsample package

```
set.seed(27)
boots <- bootstraps(mtcars, times = 500)

boots
# Bootstrap sampling
# A tibble: 500 x 2
   splits          id
   <list>          <chr>
 1 <split [32/10]> Bootstrap001
 2 <split [32/12]> Bootstrap002
 3 <split [32/10]> Bootstrap003
 4 <split [32/10]> Bootstrap004
 5 <split [32/11]> Bootstrap005
 6 <split [32/14]> Bootstrap006
 7 <split [32/11]> Bootstrap007
 8 <split [32/8]>  Bootstrap008
 9 <split [32/11]> Bootstrap009
10 <split [32/13]> Bootstrap010
# ... with 490 more rows
```

Generate 'times' bootstrapped samples

{rsample} provides a more general approach, allowing cross-validation

For bootstrapping, each split[n/m] contains:
[n] sample with replacement
[/m] items not selected in that sample

Running the bootstrap

```
fit_nls<- function(split) {
  nls(mpg ~ k / wt + b,
      data= analysis(split),
      start = list(k = 1, b = 0))
}
```

Create a helper function to fit an nls() model on each bootstrap sample. rsample::analysis() extracts that sample.

```
boot_models <-
  boots %>%
  mutate(model = map(splits, fit_nls),
         coef_info = map(model, tidy))
```

Use purrr::map() to apply this function to all the bootstrap samples at once.
Similarly, create a column of tidy coefficients

```
boot_coefs <-
  boot_models %>%
  unnest(coef_info)
```

Extract the coefficients for all models

Bootstrapped coefficients

The result is a data frame of coefficient statistics for each bootstrap sample

```
> boot_coefs
# A tibble: 1,000 x 8
   splits          id           model  term  estimate std.error statistic  p.value
   <list>          <chr>        <list> <chr>    <dbl>     <dbl>     <dbl>    <dbl>
 1 <split [32/10]> Bootstrap001 <nls>  k        47.1       3.49    13.5   2.74e-14
 2 <split [32/10]> Bootstrap001 <nls>  b         3.60      1.23     2.92  6.62e- 3
 3 <split [32/12]> Bootstrap002 <nls>  k        50.0       5.64     8.87  6.95e-10
 4 <split [32/12]> Bootstrap002 <nls>  b         3.29      2.09     1.57  1.26e- 1
 5 <split [32/10]> Bootstrap003 <nls>  k        42.0       4.38     9.59  1.20e-10
 6 <split [32/10]> Bootstrap003 <nls>  b         5.89      1.51     3.89  5.20e- 4
 7 <split [32/10]> Bootstrap004 <nls>  k        56.7       5.01    11.3   2.36e-12
 8 <split [32/10]> Bootstrap004 <nls>  b         1.49      1.75     0.852 4.01e- 1
 9 <split [32/11]> Bootstrap005 <nls>  k        48.6       3.22    15.1   1.48e-15
10 <split [32/11]> Bootstrap005 <nls>  b         3.01      1.22     2.46  1.98e- 2
# ... with 990 more rows
```

From this we can find confidence intervals (& test hypotheses)

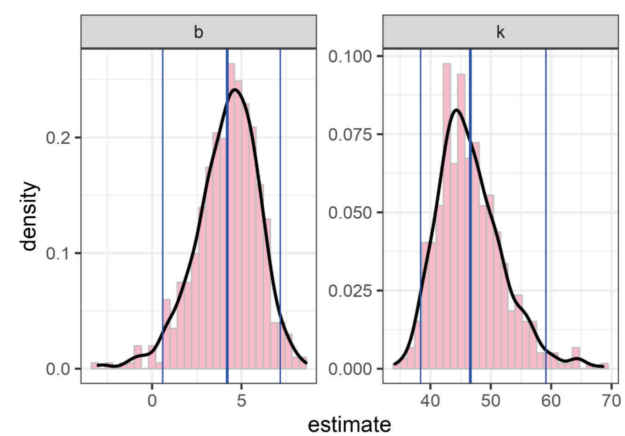
```
> int_pctl(boot_models, coef_info)
# A tibble: 2 x 6
  term  .lower .estimate .upper .alpha .method
  <chr>  <dbl>     <dbl>  <dbl>  <dbl> <chr>
1 b      0.312      4.20   7.04   0.05 percentile
2 k     38.0       46.4   59.0    0.05 percentile
```

Percentile intervals use the (.025, .975) quantiles, but require >1000 samples

Bootstrapped distributions

```
ggplot(boot_coefs, aes(estimate)) +
  geom_histogram(aes(y = ..density..),
            bins = 30, fill="pink", color="gray") +
  geom_density(size = 1.2) +
  facet_wrap( ~ term, scales = "free") + …
```

Plots of bootstrapped coefficients show their shape
-- not quite normal as assumed by std theory



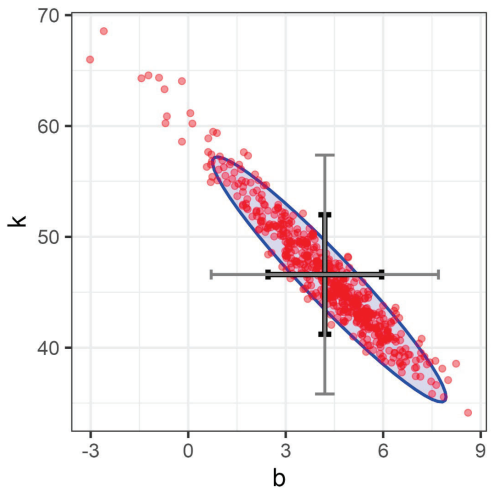
Scatterplot of coefficients

Finally, a fancy scatterplot of the joint distribution of the (b, k) estimates

How did I do this?

Processing:
1. spread coefs -> wide to plot k ~ b
2. find means , se of b & k

Plotting:
1. ellipse: stat_ellipse()
2. geom_point() – after ellipse!
3. geom_errorbar(): se * (1, 2)




```
# 1. pivot wider
boot_coefs_wide <- boot_coefs %>%
  select(id, term, estimate) %>%
  tidyr::pivot_wider(
          names_from = term,
          values_from=estimate)
```

```
> boot_coefs_wide
# A tibble: 500 x 3
   id                k     b
   <chr>         <dbl> <dbl>
 1 Bootstrap001   47.1  3.60
 2 Bootstrap002   50.0  3.29
 3 Bootstrap003   42.0  5.89
 4 Bootstrap004   56.7  1.49
 5 Bootstrap005   48.6  3.01
 6 Bootstrap006   42.7  4.46
 7 Bootstrap007   49.1  3.56
 8 Bootstrap008   49.6  3.19
 9 Bootstrap009   51.8  2.66
10 Bootstrap010   54.0  1.94
# ... with 490 more rows
```

```
# 2. find means , se of b & k
mean_se <- boot_coefs_wide %>%
  summarise(
    sk = sd(k),  sb = sd(b),
     k = mean(k), b = mean(b))
```

```
> mean_se, digits=4
     sk    sb     k     b
1 5.511 1.737 46.37 4.204
```

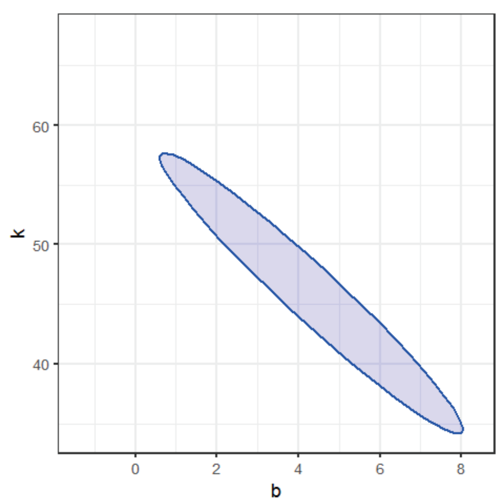


```
boot_coefs_wide %>%
  ggplot(aes(b, k)) +
    stat_ellipse(level = .95,
             geom = "polygon", alpha = 0.15,
             fill = "blue", color = "blue") +    ❶
    geom_point(color = "red", size=2, alpha=0.4) +    ❷
```

Error bars

```
geom_errorbar(data = mean_se,    ❸
      aes(ymin = k - sk,
      ymax = k + sk, x = b), size=2) +

geom_errorbarh(data = mean_se,
       aes(xmin = b - sb,
             xmax = b + sb, y = k), size=2) +
```

Redraw error bars at m ± 2 sd, but thinner ❹



Visualize the fitted curves

```
boot_aug <-
  boot_models %>%
  sample_n(200) %>%
  mutate(augmented =
           map(model, augment)) %>%
  unnest(augmented)
```

Use augment() to visualize the uncertainty in the fitted curve

Use sample_n() to plot only 200

```
ggplot(boot_aug, aes(wt, mpg)) +
  geom_line(aes(y = .fitted, group = id),
     alpha = 0.1) +
  geom_line(data=mtcars,
     aes(x = wt, y = predict(nlsfit)), color="red") +
  geom_point() +
  labs(x = "Weight", y = "Miles per gallon")
```