Statistical Modeling

Abhijit Dasgupta

November 7, 2019

Goals today

- Computational inference and permutation tests
- Using data models
    - Types and purpose
- Common statistical models
    - Linear models
    - Logistic models
- The "formula interface" in R to express models
- Learning to extract and use the results of models
    - The `broom` package
    - Graphical representations
- What predictors/covariates are "important"
- Model building and selection
- Good practices

Revisting the homework

- Know where on your computer the data is stored!!!
  - `fnames <- dir('~/Dropbox/BIOF339_Fall2019/data/GSE123519_RAW/', pattern='txt')`
  - ~ is the home directory for your computer on Mac and Linux machines
  - On my Windows machine the same command would be `fnames <- dir('C:/Users/dasgupab/Dropbox/BIOF339_Fall2019/data/GSE123519_RAW/', pattern='txt')`
- Make sure your data are stored in the right type, since this has knock-on effects
  - If you're working on a data.frame, use `dplyr` verbs like `select`,`mutate`,`filter`, etc.
  - If you're working on a list, use `map`. You'd have to embed the `dplyr` verbs within `map` in the form of the 2nd argument. See Lecture 7

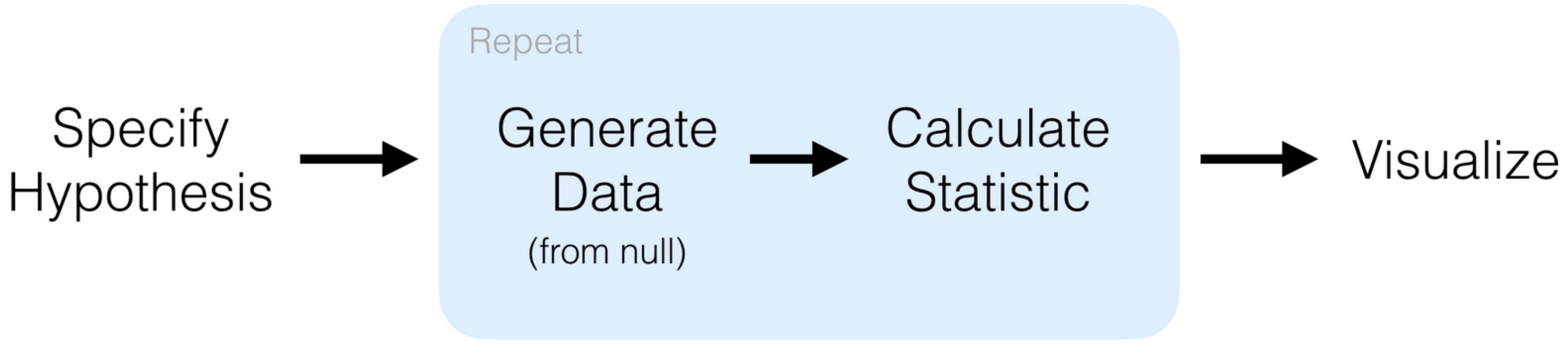
Revisiting the homework

- Check spelling and matching brackets and quotation marks.
- When working on a list of compatibly formatted data.frames, work on one of them first to figure out the data manipulation recipe, and then use `map` to replicate using the same recipe on all the data.frames.
    - By **recipe** I mean a pipeline of functions, typically, or a custom function that does all the manipulation within it for a single data.frame

A bit of computational inference

Computational inference

- Using the computer and simulation to simulate null distributions or sampling distributions of statistics
    - *sampling distribution* is the distribution of values we'd see for a statistics if we repeated the experiment over and over
    - *null distribution* is the distribution of values we'd expect to see if the null hypothesis we're using happens to be true.



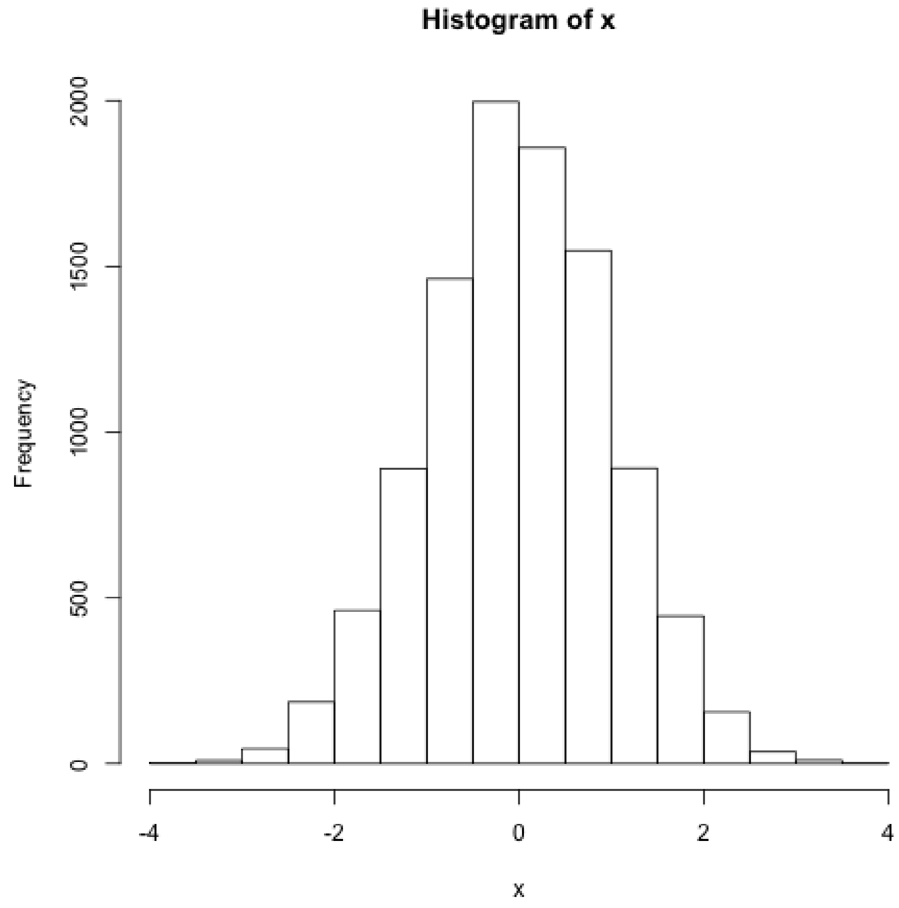
Simulation

R has several standard distributions programmed in, from which random numbers can be drawn and distributions visualized

Simulation

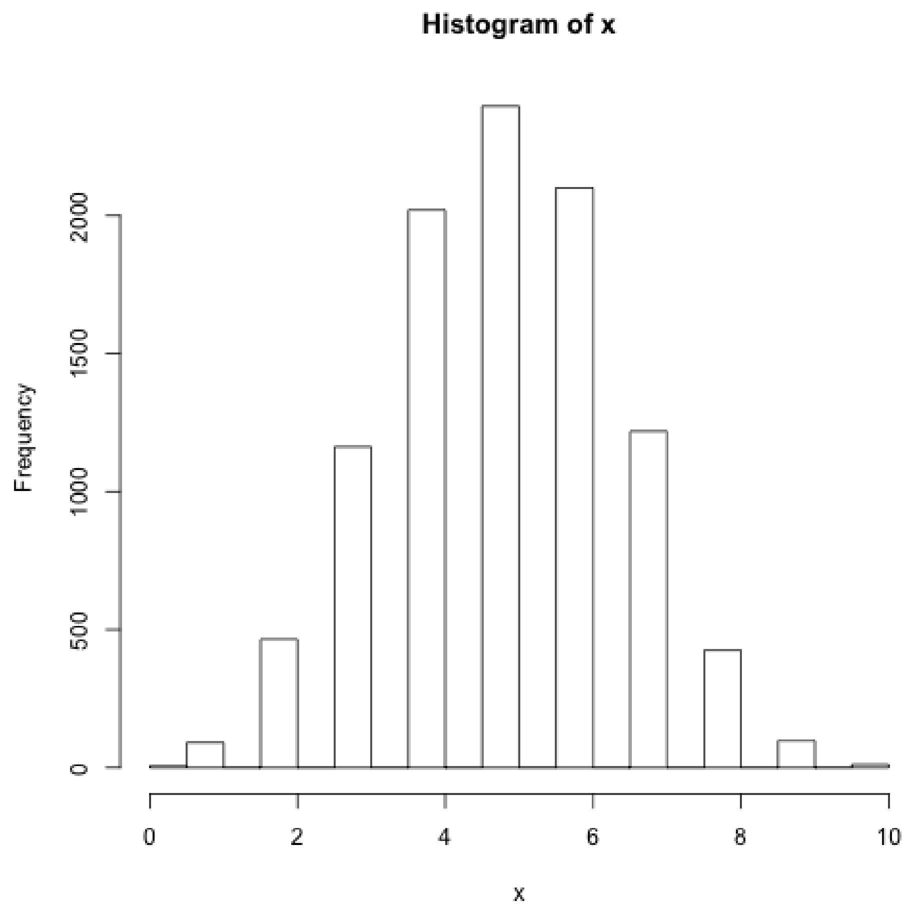
The Gaussian or normal distribution

```
x <- rnorm(10000) # 10,000 random numbers from standa
hist(x) # This is the base R way to create a histogra
```



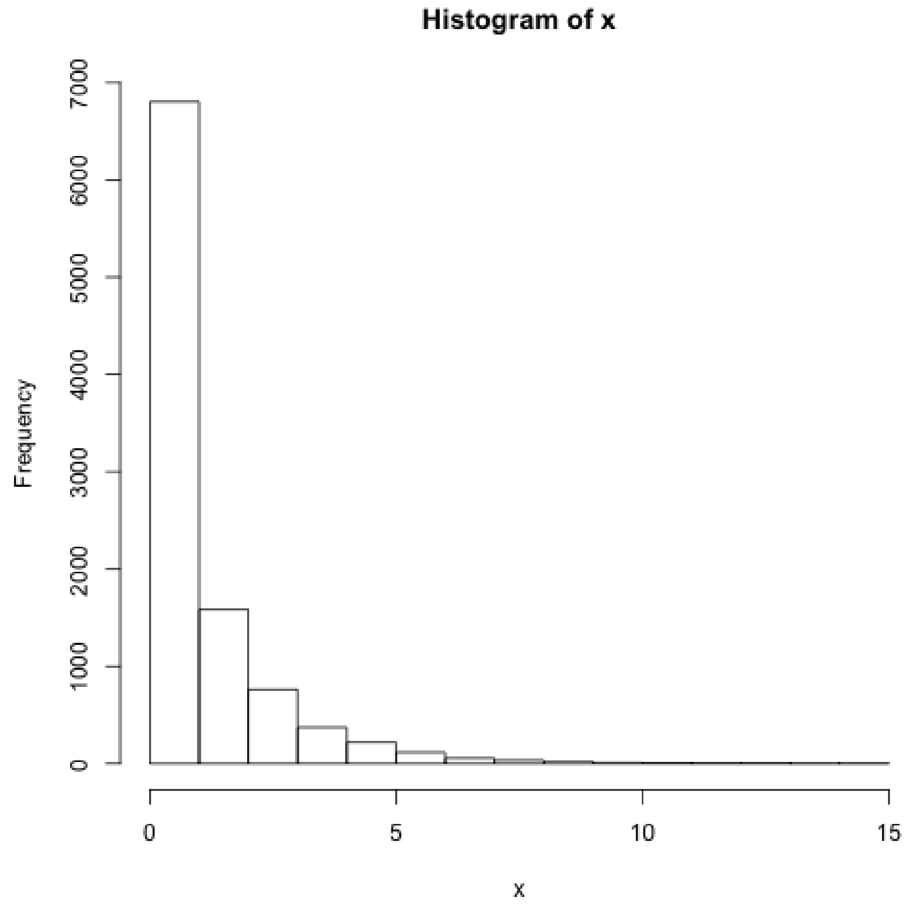
Simulation

```r
# Toss a fair coin 10 times, count number of heads
# Repeat 10,000 times
x <- rbinom(10000, size = 10, prob = 0.5)
hist(x)
```



Simulation

```
# 10,000 random dumbers from a chi-square distributio
# with 1 d.f.

x <- rchisq(10000, 1)
hist(x)
```

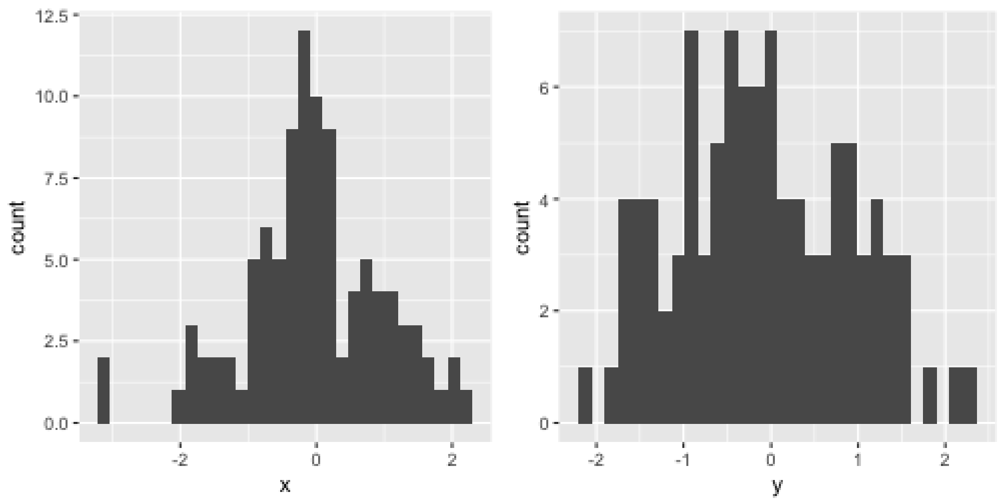


Simulations

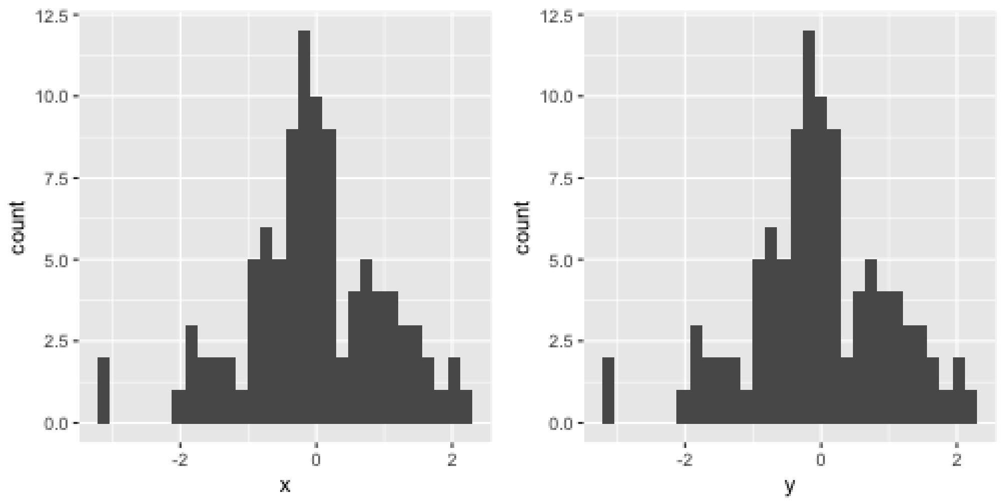
Simulations on a computer aren't exactly random, but *pseudo-random*. They form a *complex, deterministic* series of numbers which have some properties.

You can set the starting point of the series. It's called the **seed**.

```
set.seed(28954)

dd <- data.frame(x = rnorm(100))

dd$y <- rnorm(100)

plt1 <- ggplot(dd, aes(x))+geom_histogram()
plt2 <- ggplot(dd, aes( y)) + geom_histogram()

cowplot::plot_grid(plt1, plt2, nrow=1)
```



```
set.seed(28954)

dd <- data.frame(x = rnorm(100))
set.seed(28954)
dd$y <- rnorm(100)

plt1 <- ggplot(dd, aes(x))+geom_histogram()
plt2 <- ggplot(dd, aes( y)) + geom_histogram()

cowplot::plot_grid(plt1, plt2, nrow=1)
```



Always set a seed to ensure replicability of simulation experiments

Permutation tests

We will use the R package `infer` for this section, as well as the `pbc` data. We'll first do a permutation test to see if bilirubin is significantly different by treatment group.

A classical thing to do would be a `t.test` or a `wilcox.test`.

```r
library(survival)

t.test(bili~trt, data=pbc)
```

```

    Welch Two Sample t-test

data:  bili by trt
t = -1.5074, df = 270.39, p-value = 0.1329
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.7878314  0.2372643
sample estimates:
mean in group 1 mean in group 2 
       2.873418        3.648701 
```

Permutation tests

In a permution test, we assume the null hypothesis of no difference between the groups, which means

- if we shuffled the group memberships (re-assigned individuals to different treatments) nothing should change.

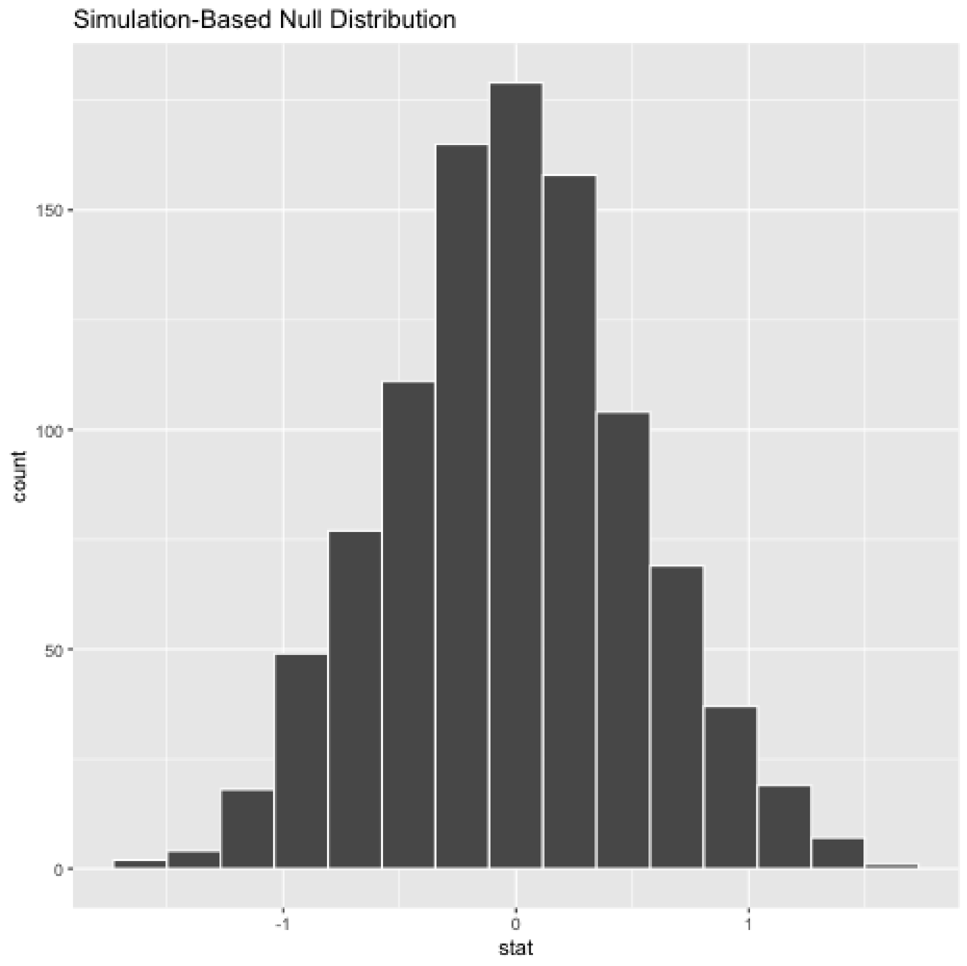
If we compute the test statistic over different permutations, we'll get the distribution of test statistic values we'd see if the null hypothesis was true.

Permutation tests

```r
library(infer)

set.seed(10193)
sims <- pbc %>%
  mutate(trt = as.factor(trt)) %>%
  specify(bili ~ trt) %>%
  hypothesize(null = 'independence') %>%
  generate(reps = 1000, type = 'permute') %>%
  calculate(stat = 'diff in means', order = c('1','2'

visualize(sims)
```

```r
(obs_stat <- pbc %>% mutate(trt=as.factor(trt)) %>%
  specify(bili ~ trt) %>%
  calculate(stat='diff in means',order = c('1','2')))
```

```
# A tibble: 1 x 1
    stat
   <dbl>
1 -0.775
```

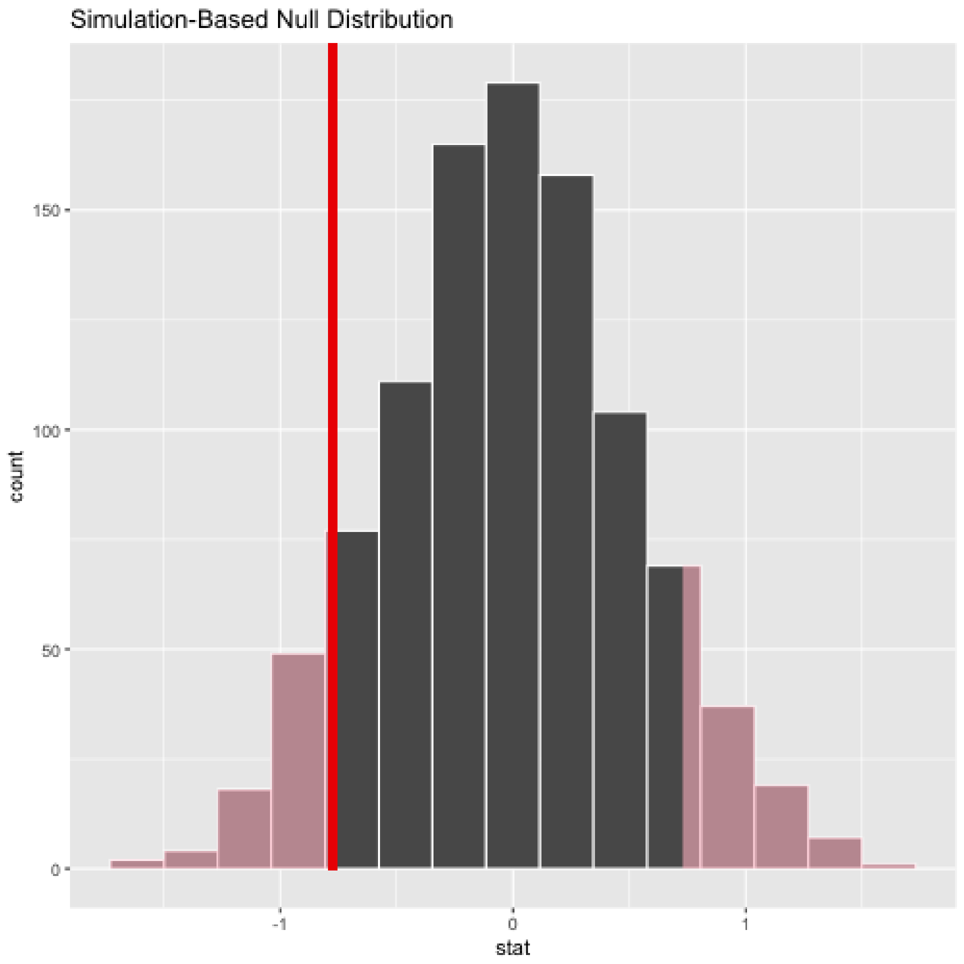


Permutation tests

```
library(infer)

set.seed(10193)
sims <- pbc %>%
  mutate(trt = as.factor(trt)) %>%
  specify(bili ~ trt) %>% # trt must be a factor
  hypothesize(null = 'independence') %>%
  generate(reps = 1000, type = 'permute') %>%
  calculate(stat = 'diff in means', order = c('1','2'

visualize(sims) + shade_p_value(obs_stat, direction =
```

```
sims %>% get_pvalue(obs_stat, direction ='both')
```

```
# A tibble: 1 x 1
  p_value
    <dbl>
1   0.158
```



Bootstrapping for confidence intervals

Suppose we want to get a confidence interval for the mean bilirubin level overall.

The bootstrap samples the original data **with replacement** to get a dataset of the same size.

Since sampling is with replacement, some observations are repeated, some are omitted.

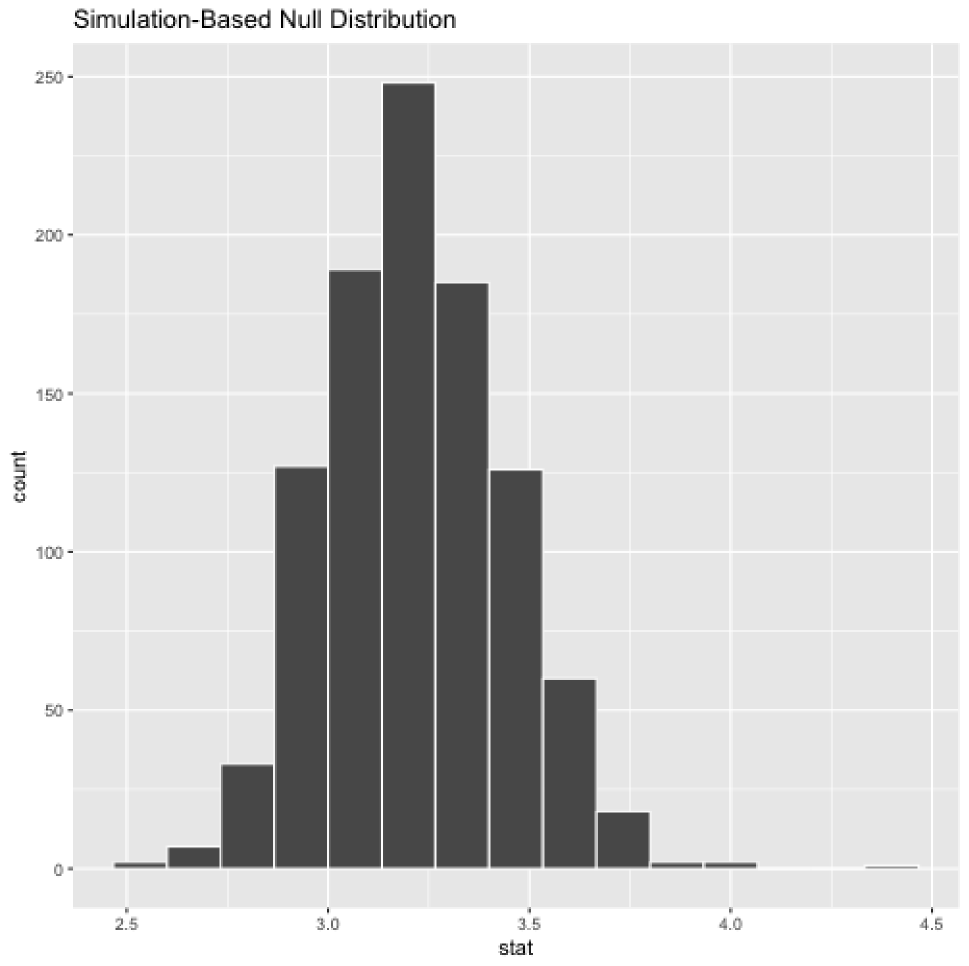
Strong theory from the 80s and 90s says that if we repeatedly take bootstrap samples of our data and compute the sample means, their distribution will be very close to the true sampling distribution of the sample mean.

Bootstrapping for confidence intervals

```
x <- pbc %>%
  specify(response = bili) %>%
  generate(reps = 1000, type = 'bootstrap') %>%
  calculate('mean')
```

```
(ci <- x %>% get_confidence_interval())
```

```
# A tibble: 1 x 2
  `2.5%` `97.5%`
   <dbl>   <dbl>
1   2.84    3.66
```



Resources

1. Several infer examples
2. The R Companion chapter on permuation tests
3. This site gives alternative methods for doing permutation tests in R
4. The coin package provides a richer set of permutation tests, but `infer` covers what you need most often.

Statistical models

> *All models are wrong, but some are useful*

G.E.P. Box

Models

Models are our way of understanding nature, usually using some sort of mathematical expression

Famous mathematical models include Newton's second law of motion, the laws of thermodynamics, the ideal gas law

All probability distributions, like Gaussian, Binomial, Poisson, Gamma, are models

Mendel's laws are models **that result in** particular mathematical models for inheritance and population prevalence

Models

We use models all the time to describe our understanding of different processes

- Cause-and-effect relationships
- Supply-demand curves
- Financial planning
- Optimizing travel plans (perhaps including traffic like *Google Maps*)
- Understanding the effects of change
    - Climate change
    - Rule changes via Congress or companies
    - Effect of a drug on disease outcomes
    - Effect of education and behavioral patterns on future earnings

Data-driven models

> Can we use data collected on various aspects of a particular context to understand the relationships between the different aspects?

- How does increased smoking affect your risk of getting lung cancer? (causality/association)
    - Does genetics matter?
    - Does the kind of smoking matter?
    - Does gender matter?

Data-driven models

> Can we use data collected on various aspects of a particular context to understand the relationships between the different aspects?

- What is your lifetime risk of breast cancer? (prediction)
    - What if you have a sister with breast cancer?
    - What if you had early menarche?
    - What if you are of Ashkenazi Jewish heritage?

The Gail Model from NCI

Association models

These are more traditional, highly interpretative models that look at **how** different predictors affect outcome.

- Linear regression
- Logistic regression
- Cox proportional hazards regression
- Decision trees

Since these models have a particular known structure determined by the modeler, they can be used on relatively small datasets

You can easily understand which predictors have more "weight" in influencing the outcome

You can literally write down how a prediction would be made

Predictive models

These are more recent models that primarily look to provide good predictions of an outcome, and the way the predictions are made is left opaque (often called a *black box*)

- Deep Learning (or Neural Networks)
- Random Forests
- Support Vector Machines
- Gradient Boosting Machines

These models require data to both determine the structure of the model as well as make the predictions, so they require lots of data to *train* on

The relative "weight" of predictors in influencing the **predictions** can be obtained

The effect of individual predictors is not easily interpretable, though this is changing

They require a different **philosophic perspective** than traditional association models

Datasets

We will use the `pbc` data from the `survival` package, and the in-built `mtcars` dataset.

```r
library(survival)
str(pbc)
```

```
'data.frame':	418 obs. of  20 variables:
 $ id      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ time    : int  400 4500 1012 1925 1504 2503 1832 2466 2400 51 ...
 $ status  : int  2 0 2 2 1 2 0 2 2 2 ...
 $ trt     : int  1 1 1 1 2 2 2 2 1 2 ...
 $ age     : num  58.8 56.4 70.1 54.7 38.1 ...
 $ sex     : Factor w/ 2 levels "m","f": 2 2 1 2 2 2 2 2 2 2 ...
 $ ascites : int  1 0 0 0 0 0 0 0 0 1 ...
 $ hepato  : int  1 1 0 1 1 1 1 0 0 0 ...
 $ spiders : int  1 1 0 1 1 0 0 0 1 1 ...
 $ edema   : num  1 0 0.5 0.5 0 0 0 0 0 1 ...
 $ bili    : num  14.5 1.1 1.4 1.8 3.4 0.8 1 0.3 3.2 12.6 ...
 $ chol    : int  261 302 176 244 279 248 322 280 562 200 ...
 $ albumin : num  2.6 4.14 3.48 2.54 3.53 3.98 4.09 4 3.08 2.74 ...
 $ copper  : int  156 54 210 64 143 50 52 52 79 140 ...
 $ alk.phos: num  1718 7395 516 6122 671 ...
 $ ast     : num  137.9 113.5 96.1 60.6 113.2 ...
 $ trig    : int  172 88 55 92 72 63 213 189 88 143 ...
 $ platelet: int  190 221 151 183 136 NA 204 373 251 302 ...
 $ protime : num  12.2 10.6 12 10.3 10.9 11 9.7 11 11 11.5 ...
 $ stage   : int  4 3 4 4 3 3 3 3 2 4 ...
```

The formula interface

Representing model relationships

In R, there is a particularly convenient way to express models, where you have

- one dependent variable
- one or more independent variables, with possible transformations and interactions

```
y ~ x1 + x2 + x1:x2 + I(x3^2) + x4*x5
```

`y` depends on ...

- `x1` and `x2` linearly
- the interaction of `x1` and `x2` (represented as `x1:x2`)
- the square of `x3` (the `I()` notation ensures that the `^` symbol is interpreted correctly)
- `x4`, `x5` and their interaction (same as `x4 + x5 + x4:x5`)

Representing model relationships

```
y ~ x1 + x2 + x1:x2 + I(x3^2) + x4*x5
```

This interpretation holds for the vast majority of statistical models in R

- For decision trees and random forests and neural networks, don't add interactions or transformations, since the model will try to figure those out on their own

Our first model

```
myLinearModel <- lm(chol ~ bili, data = pbc)
```

Note that everything in R is an **object**, so you can store a model in a variable name.

This statement runs the model and stored the fitted model in `myLinearModel`

R does not interpret the model, evaluate the adequacy or appropriateness of the model, or comment on whether looking at the relationship between cholesterol and bilirubin makes any kind of sense.

> *It just fits the model it is given*

Our first model

```
myLinearModel
```

```

Call:
lm(formula = chol ~ bili, data = pbc)

Coefficients:
(Intercept)         bili
     303.20        20.24
```

> Not very informative, is it?

Our first model

```
summary(myLinearModel)
```

```

Call:
lm(formula = chol ~ bili, data = pbc)

Residuals:
    Min      1Q  Median      3Q     Max 
-565.39  -89.90  -35.36   44.92 1285.33 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  303.204     15.601  19.435  < 2e-16 ***
bili          20.240      2.785   7.267 3.63e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 213.2 on 282 degrees of freedom
  (134 observations deleted due to missingness)
Multiple R-squared:  0.1577,	Adjusted R-squared:  0.1547 
F-statistic:  52.8 on 1 and 282 DF,  p-value: 3.628e-12
```

> A little better

Our first model

```
broom::tidy(myLinearModel)
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)    303.      15.6      19.4  5.65e-54
2 bili            20.2      2.79      7.27 3.63e-12
```

```
broom::glance(myLinearModel)
```

```
# A tibble: 1 x 11
  r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
      <dbl>         <dbl> <dbl>     <dbl>    <dbl> <int>  <dbl> <dbl> <dbl>
1     0.158         0.155  213.      52.8 3.63e-12     2 -1925. 3856. 3867.
# … with 2 more variables: deviance <dbl>, df.residual <int>
```

Our first model

We do need some sense as to how well this model fit the data

```r
# install.packages('ggfortify')
library(ggfortify)
autoplot(myLinearModel)
```

Our first model

Let's see if we have some strangeness going on

```
ggplot(pbc, aes(x = bili))+geom_density()
```

We'd like this to be a bit more "Gaussian" for better behavior

Our first model

Let's see if we have some strangeness going on

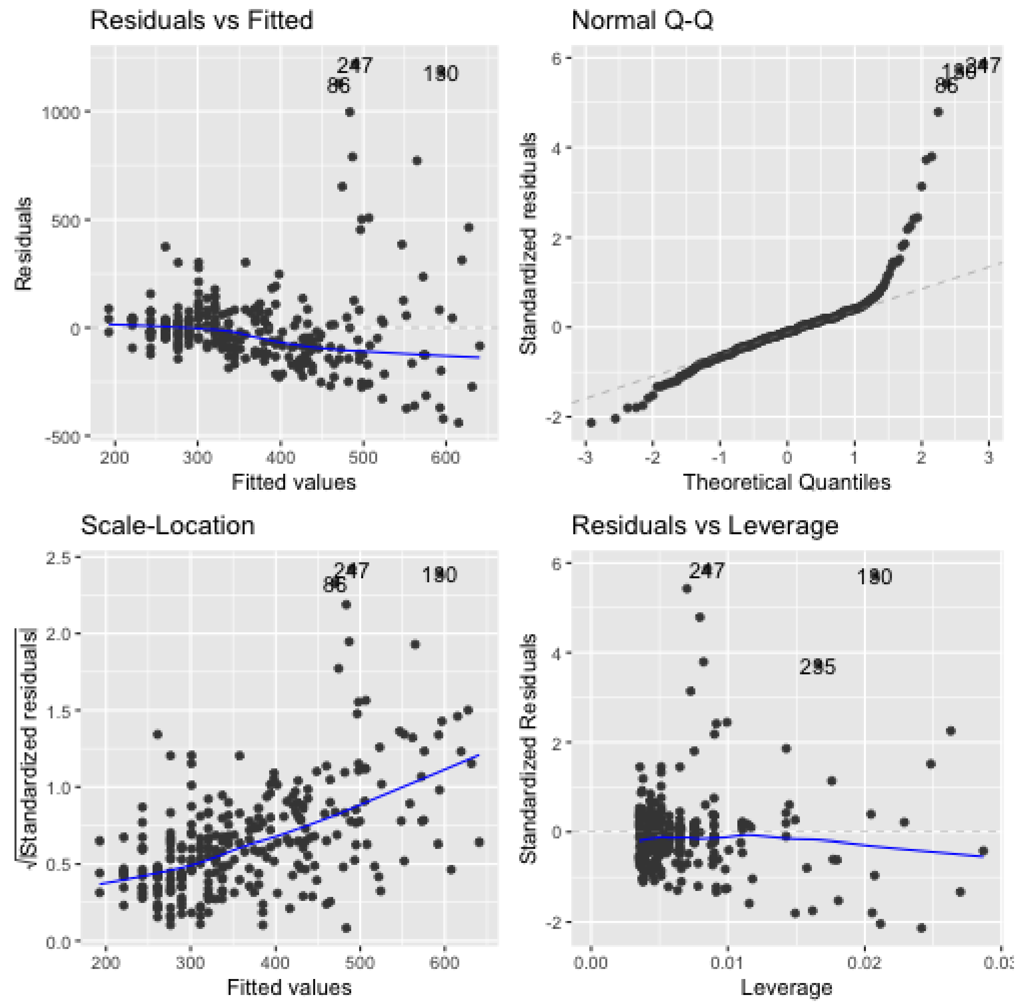
```
ggplot(pbc, aes(x = log(bili)))+geom_density()
```

Our first model

```
myLinearModel2 <- lm(chol~log(bili), data = pbc)
summary(myLinearModel2)
```

```

Call:
lm(formula = chol ~ log(bili), data = pbc)

Residuals:
    Min      1Q  Median      3Q     Max 
-440.07  -94.35  -21.07   42.67 1221.86 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   311.48      14.28  21.816  < 2e-16 ***
log(bili)      98.80      12.07   8.186 9.42e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 208.9 on 282 degrees of freedom
  (134 observations deleted due to missingness)
Multiple R-squared:  0.192,	Adjusted R-squared:  0.1891 
F-statistic: 67.01 on 1 and 282 DF,  p-value: 9.416e-15
```

Our first model



Just the residual plot, please

```
autoplot(myLinearModel2, which=1)
```

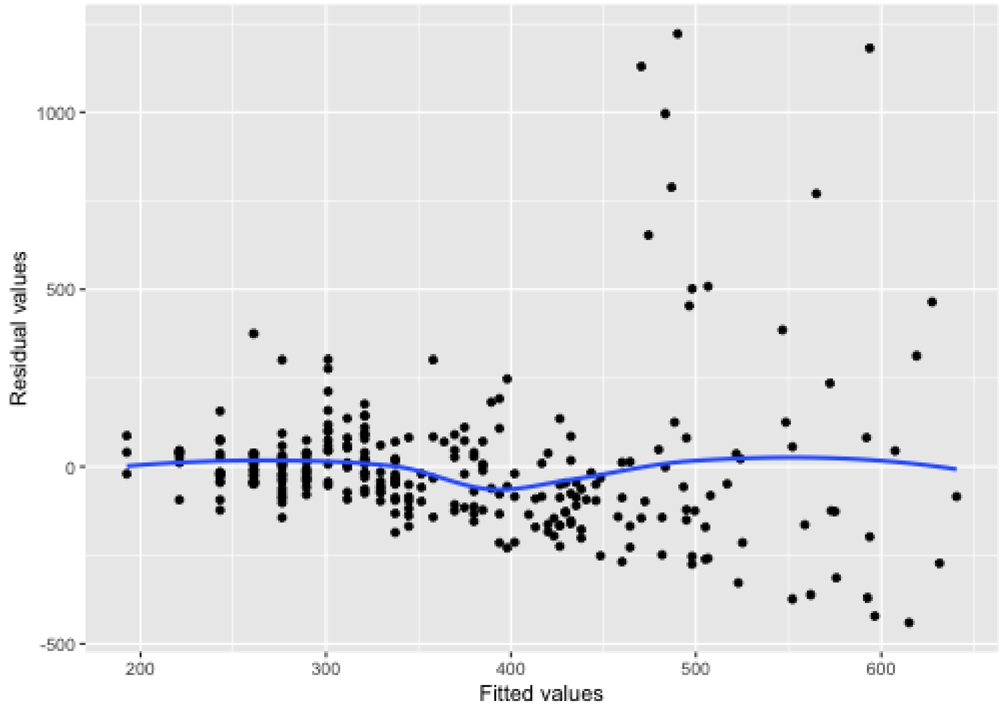
Just the residual plot, please

```
d <- broom::augment(myLinearModel2)
d
```

```
# A tibble: 284 x 10
   .rownames  chol log.bili. .fitted .se.fit .resid    .hat .sigma .cooksd
   <chr>     <int>     <dbl>   <dbl>   <dbl>  <dbl>   <dbl>  <dbl>   <dbl>
 1 1           261    2.67      576.    28.1 -315.  0.0181    208. 2.13e-2
 2 2           302    0.0953    321.    13.7  -18.9 0.00433   209. 1.79e-5
 3 3           176    0.336     345.    12.8 -169.  0.00373   209. 1.23e-3
 4 4           244    0.588     370.    12.4 -126.  0.00352   209. 6.41e-4
 5 5           279    1.22      432.    14.6 -153.  0.00487   209. 1.33e-3
 6 6           248   -0.223     289.    15.8  -41.4 0.00571   209. 1.14e-4
 7 7           322    0         311.    14.3   10.5 0.00467   209. 5.98e-6
 8 8           280   -1.20      193.    24.9   87.5 0.0142    209. 1.28e-3
 9 9           562    1.16      426.    14.2  136.  0.00463   209. 9.84e-4
10 10          200    2.53      562.    26.6 -362.  0.0162    208. 2.51e-2
# … with 274 more rows, and 1 more variable: .std.resid <dbl>
```

Just the residual plot, please

```
ggplot(d, aes(x = .fitted, y = .resid))+geom_point()+ geom_smooth(se=F)+
  labs(x = 'Fitted values', y = 'Residual values')
```



Predictions

```
head(predict(myLinearModel2, newdata = pbc))
```

```
       1        2        3        4        5        6 
575.6925 320.9006 344.7277 369.5578 432.3941 289.4371 
```

The `newdata` has to have the same format and components as the original data the model was trained on

Categorical predictors

```r
myLM3 <- lm(chol ~ log(bili) + sex, data = pbc)
broom::tidy(myLM3)
```

```
# A tibble: 3 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)    283.       36.6     7.71  2.14e-13
2 log(bili)       99.6      12.1     8.22  7.37e-15
3 sexf            32.5      37.8     0.858 3.92e- 1
```

R has a somewhat unfortunate notation for categorical varialbes here, as `{variable name}{level}`

Logistic regression

The logistic transformation

For an outcome which is binary (0/1), what is really modeled is the **probability** that the outcome is 1, usually denoted by *p*.

However, we know $0 \leq p \leq 1$, so what if the model gives a prediction outside this range!!

The logistic transform takes *p* to

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

and we model *logit(p)*, which has a range from $-\infty$ to ∞

Logistic regression

Logistic regression is a special case of a **generalized linear model**, so the function we use to run a logistic regression is `glm`

```
myLR <- glm(spiders ~ albumin + bili + chol, data = pbc, family = binomial)
myLR
```

```

Call:  glm(formula = spiders ~ albumin + bili + chol, family = binomial,
    data = pbc)

Coefficients:
(Intercept)      albumin         bili         chol
  2.3326484   -0.9954927    0.0995915   -0.0003176

Degrees of Freedom: 283 Total (i.e. Null);  280 Residual
  (134 observations deleted due to missingness)
Null Deviance:      341.4
Residual Deviance: 315.2     AIC: 323.2
```

- We have to add the `family = binomial` as an argument, since this is a special kind of GLM
- All these models only use complete data; they kick out rows with missing data

Logistic regression

```
broom::tidy(myLR)
```

```
# A tibble: 4 x 5
  term        estimate std.error statistic p.value
  <chr>          <dbl>     <dbl>     <dbl>   <dbl>
1 (Intercept)  2.33     1.30         1.80  0.0717
2 albumin     -0.995    0.362       -2.75  0.00595
3 bili         0.0996   0.0344       2.89  0.00381
4 chol        -0.000318 0.000615    -0.517 0.605
```

```
broom::glance(myLR)
```

```
# A tibble: 1 x 7
  null.deviance df.null logLik   AIC   BIC deviance df.residual
          <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int>
1          341.     283  -158.  323.  338.     315.         280
```

Predictions from logistic regression

```
head(predict(myLR))
```

```
          1           2           3           4           5           6 
 1.10554163 -1.77506554 -1.04814132 -0.09414055 -0.93144911 -1.62851203 
```

These are on the "wrong" scale. We would expect probabilities

```
head(predict(myLR, type='response'))
```

```
        1         2         3         4         5         6 
0.7512970 0.1449135 0.2595822 0.4764822 0.2826308 0.1640343 
```

or you can use `plogis(predict(myLR))` for the inverse logistic transform

Model selection

How to get the "best" model

Generally getting to the best model involves

- looking at a lot of graphs
- Fitting lots of models
- Comparing the model fits to see what seems good

Sometimes if you have two models that fit about the same, you take the smaller, less complex model (Occam's Razor)

Generally it is not recommended that you use automated model selection methods. It screws up your error rates and may not be the right end result for your objectives

> Model building and selection is an art

Clues to follow

You can look at the relative weights (size of coefficient and its p-value) of different predictors

- These weights will change once you change the model, so be aware of that

You can trim the number of variables based on collinearities

- If several variables are essentially measuring the same thing, use one of them

You can look at residuals for clues about transformations

You can look at graphs, as well as science, for clues about interactions (synergies and antagonisms)

Automated model selection

```
# install.packages('leaps')
library(leaps)
mtcars1 <- mtcars %>% mutate_at(vars(cyl, vs:carb), as.factor)
all_subsets <- regsubsets(mpg~., data = mtcars1)
all_subsets
```

```
Subset selection object
Call: coursedown::slide2pdf(knitr_in("slides/lectures/09-Modeling.Rmd"))
16 Variables  (and intercept)
      Forced in Forced out
cyl6      FALSE      FALSE
cyl8      FALSE      FALSE
disp      FALSE      FALSE
hp        FALSE      FALSE
drat      FALSE      FALSE
wt        FALSE      FALSE
qsec      FALSE      FALSE
vs1       FALSE      FALSE
am1       FALSE      FALSE
gear4     FALSE      FALSE
gear5     FALSE      FALSE
carb2     FALSE      FALSE
carb3     FALSE      FALSE
carb4     FALSE      FALSE
carb6     FALSE      FALSE
carb8     FALSE      FALSE
1 subsets of each size up to 8
Selection Algorithm: exhaustive
```

Automated model selection

Which has the best R^2?

```
ind <- which.max(summary(all_subsets)$adjr2)
summary(all_subsets)$which[ind,]
```

```
(Intercept)        cyl6        cyl8        disp          hp        drat 
       TRUE        TRUE       FALSE       FALSE        TRUE       FALSE 
         wt        qsec         vs1         am1       gear4       gear5 
       TRUE       FALSE        TRUE        TRUE       FALSE       FALSE 
      carb2       carb3       carb4       carb6       carb8 
      FALSE       FALSE       FALSE       FALSE       FALSE 
```