Welcome!

Nameplates please. And technology encouraged today!

All TF materials are available at github.com/nolankav/api-203.

If you want to follow along, download the dataset here:

In R: `df <- read.csv("http://tinyurl.com/api-203-tf-4")`

Parks and Prediction

API 203: TF Session 4

R

Nolan M. Kavanagh
March 29, 2024

Goals for today

1. **Contrast the goals of causal inference and prediction.**
2. **Review a few strategies for predicting modeling, including LASSO.**
3. **Learn how to generate predictive models in R.**
4. **Practice evaluating the accuracy of predictive models.**

We'll treat this session like a workshop with an interactive example.

Overview of our sample data

Dataset of ~17,000 U.S. adults surveyed in 2019

`casein`	**Respondent identifier**	*Cooperative Election Study*
`commonweight`	**Post-stratification survey weight**	*Cooperative Election Study*
`age`	**Respondent age (in years)**	*Cooperative Election Study*
`gender`	**Respondent gender**	*Cooperative Election Study*
`race_eth`	**Respondent race/ethnicity**	*Cooperative Election Study*
`education`	**Respondent educational attainment**	*Cooperative Election Study*
`marital`	**Respondent marital status**	*Cooperative Election Study*
`party_scale`	**Party identification (-3 = strong Rep. to 3 = strong Dem.)**	*Cooperative Election Study*
`public_ins`	**Respondent has public health insurance (1) or not (0)**	*Cooperative Election Study*
`public_option`	**Supports a public option in Medicare (1) or not (0)**	*Cooperative Election Study*

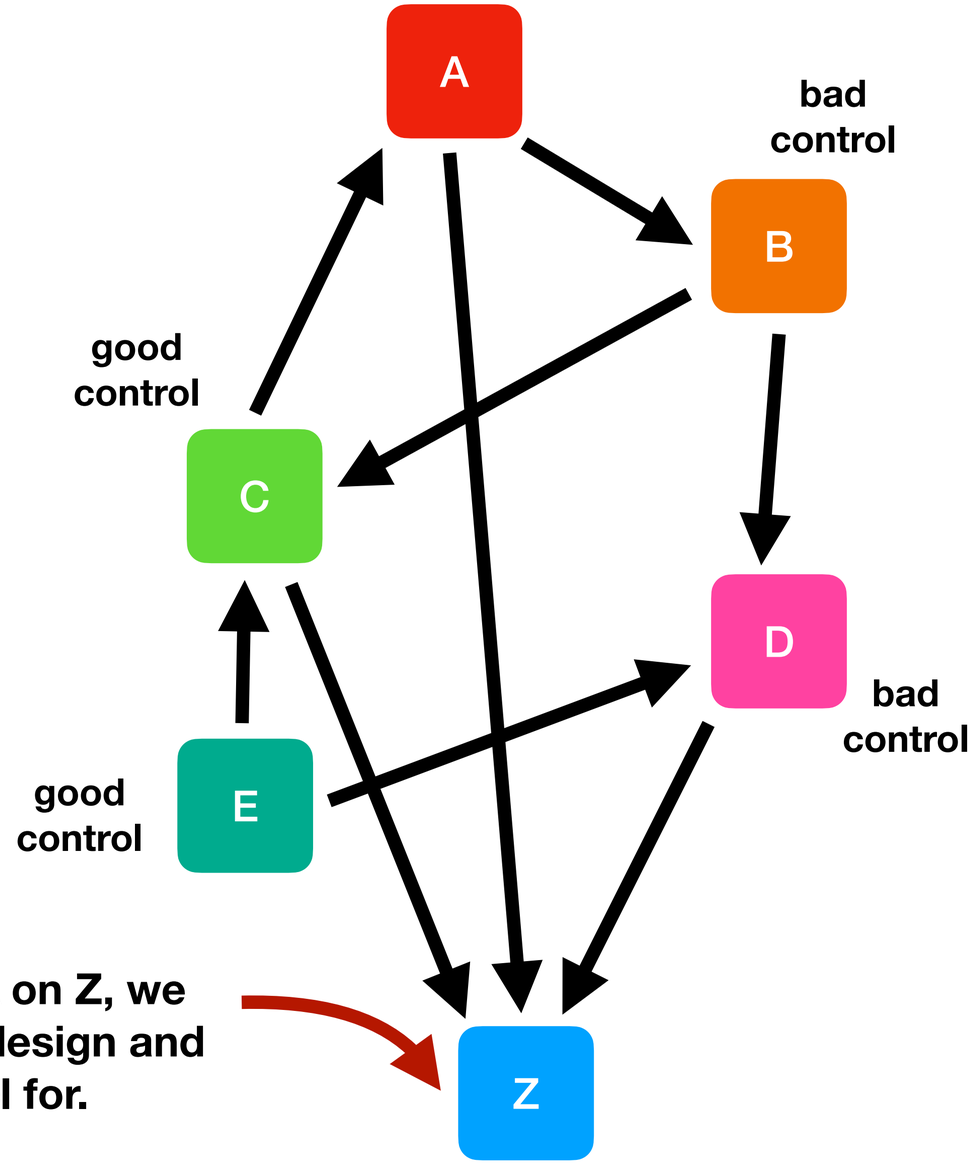
What's our goal?

With causal inference, the name of the game was bias: what causes it and how to get rid of it.

Why? Our goal was to estimate an unbiased effect.

With prediction, we don't care about bias. In fact, we invite bias. We'll use whatever we can.

Why? Our goal is accurate prediction.



To get the causal estimate of A on Z, we have to be careful about study design and which variables we control for.

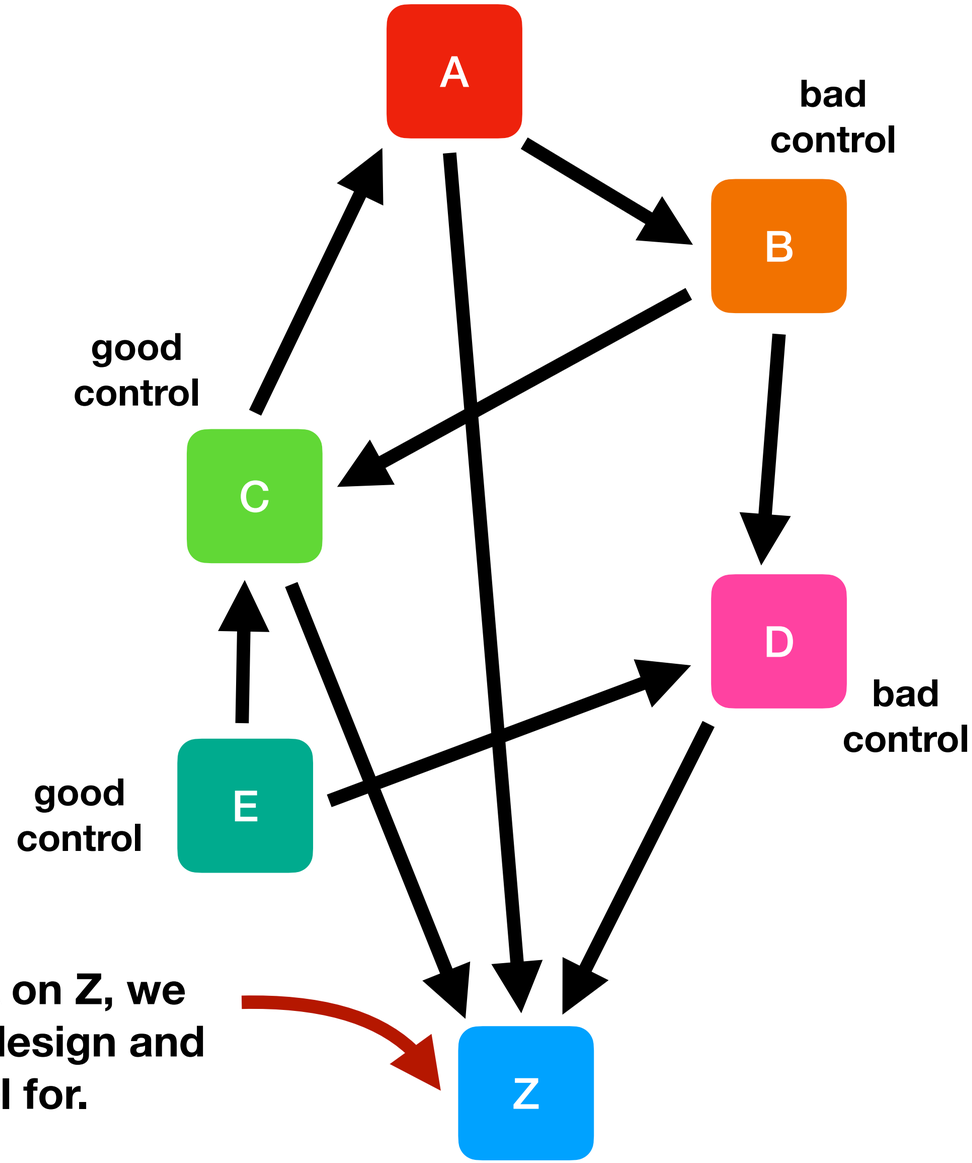
But if we just want to predict Z, every relevant variable is fair game.

What's our goal?

But with all possible variables at our disposal, we do have to worry about overfitting our models.

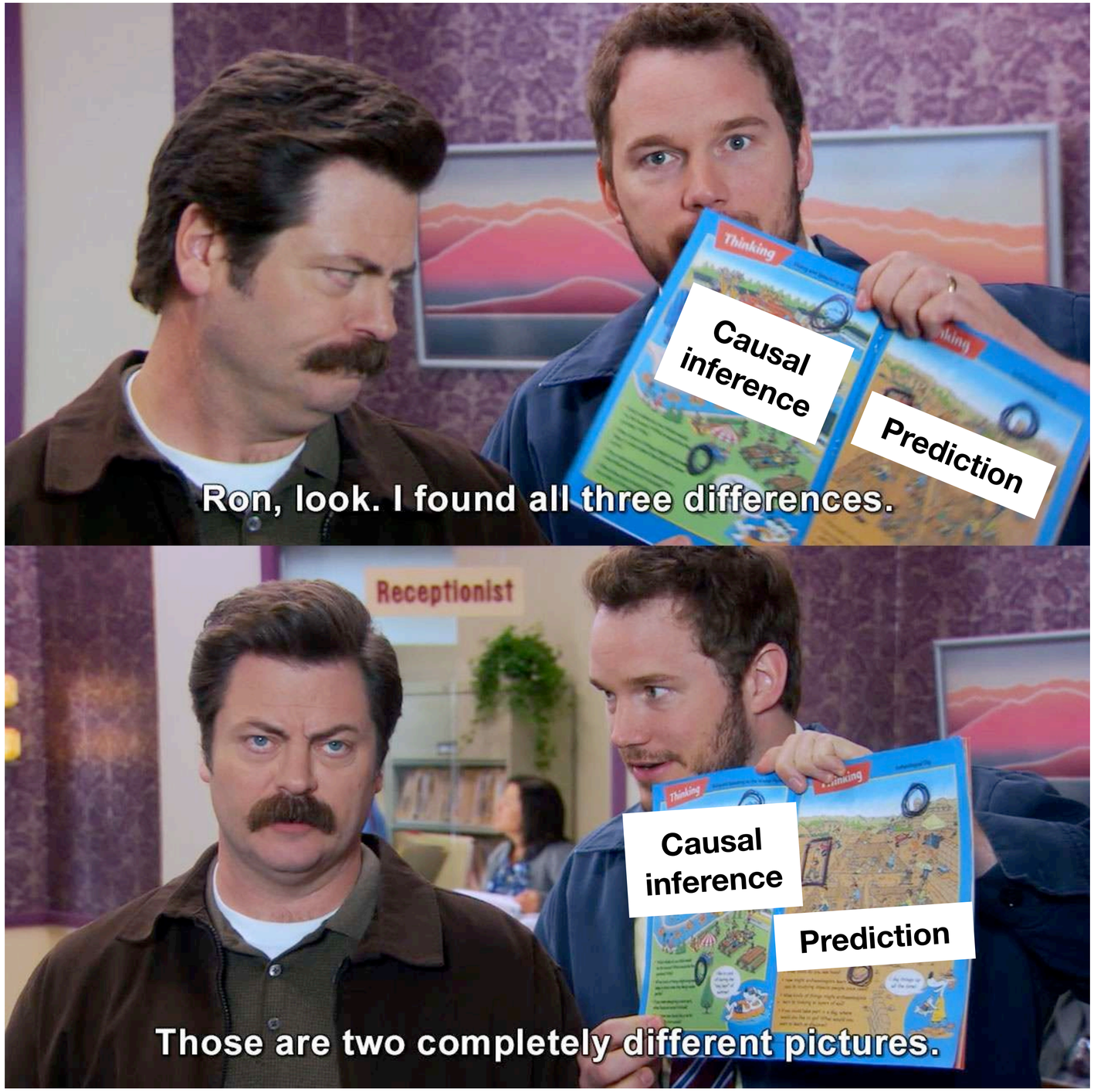
Overfitted models look good in-sample, but they tend to perform poorly out-of-sample.

Above all, we want useful predictions!



To get the causal estimate of A on Z, we have to be careful about study design and which variables we control for.

But if we just want to predict Z, every relevant variable is fair game.


Causal inference
Prediction
Ron, look. I found all three differences.
Receptionist
Causal inference
Prediction
Those are two completely different pictures.

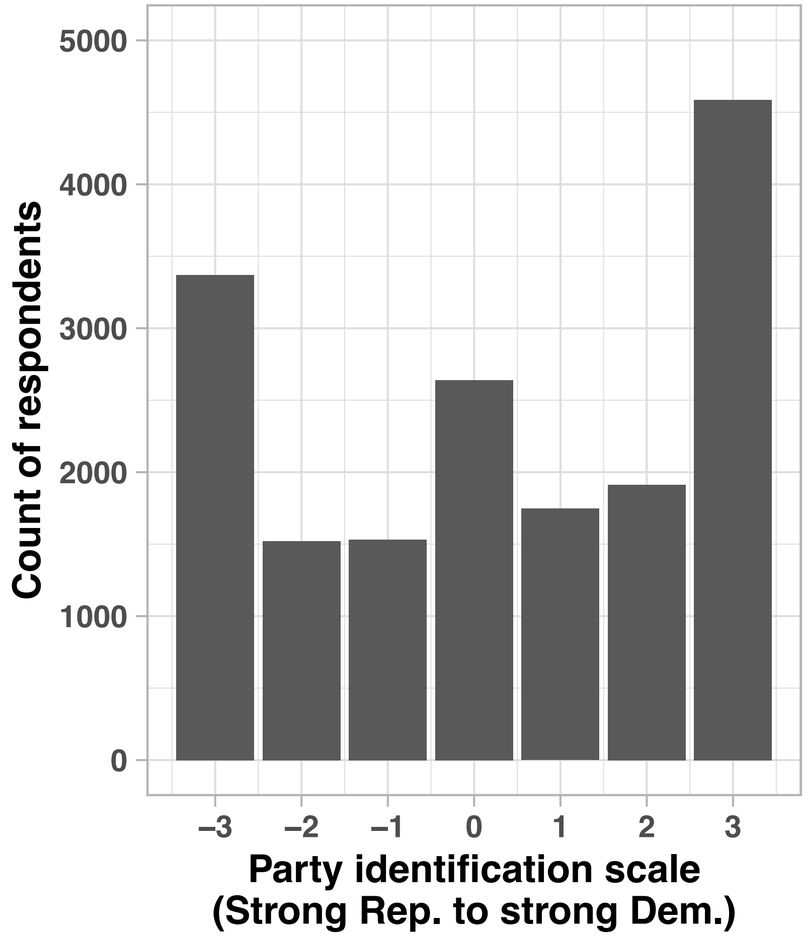
OK, so what are we predicting?

Let's try to predict political party.

Parties want to know whom they can contact for fundraising, get out the vote, etc.

This is "supervised" machine learning since we can validate our predictions on the "truth".

```
# Generate bar plot of parties
plot_1 <- ggplot(data=df, aes(x=party_scale)) +
  geom_bar(stat="count") +
  xlab("Party identification scale\n(Strong Rep. to strong Dem.)") +
  ylab("Count of respondents") +
  theme_light() +
  theme(text = element_text(size = 10, face = "bold")) +
  scale_x_continuous(breaks = seq(-3, 3, 1)) +
  scale_y_continuous(limits = c(0,5000),
                     breaks = seq(0,5000,1000))
```



First, let's split our data into two.

We want to designate "training" and "test" sets. One is for developing our model, and the other is for validating its predictions.

Here, we will do a 70/30 split. Generally, we want more data in the training set.

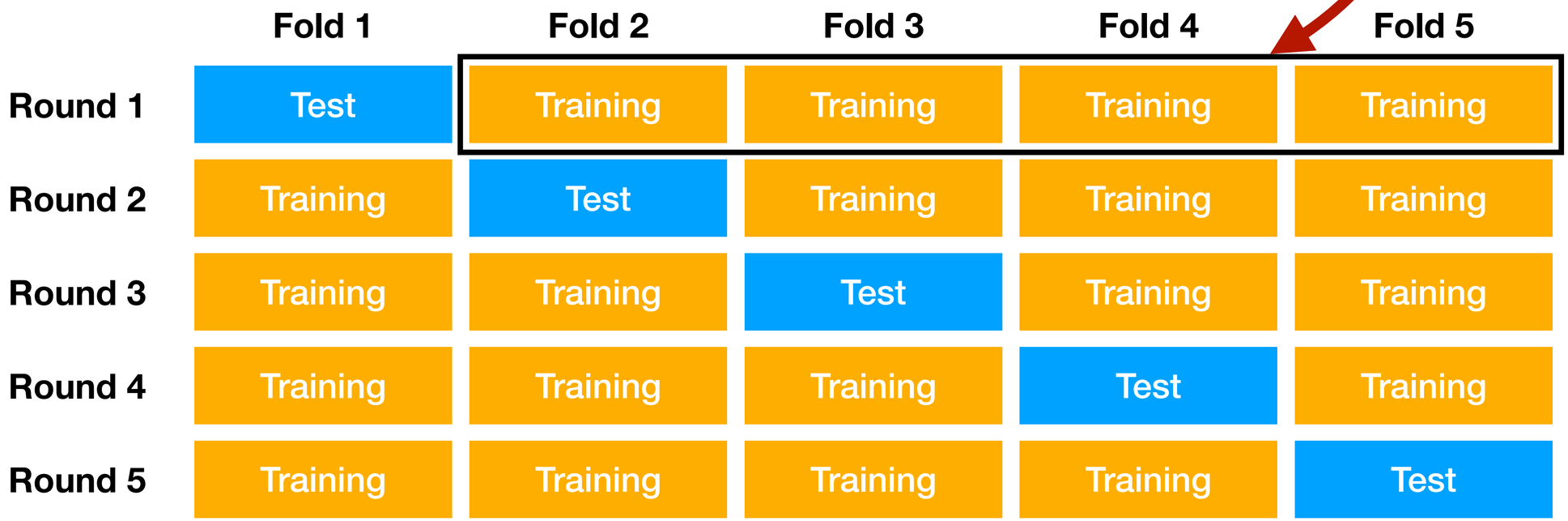
```
# Designate training and test sets
set.seed(1234)
split    <- initial_split(df, 0.7)
df_train <- training(split)
df_test  <- testing(split)
```

An advanced approach is K-fold cross-validation.

This approach splits the data into K folds.

Then, we use each fold as the test set for one round of modeling and average across all rounds.

In each round, we pool all the training folds into one big training set.




Causal inference
Ruining springtime for MPPs
Predictive modeling

Let's try a very basic model.

```
# OLS regression
model_1 <- lm(party_scale ~ gender, data=df_train)
summary(model_1)

Call:
lm(formula = party_scale ~ gender, data = df_train)

Residuals:
    Min      1Q  Median      3Q     Max
-3.4210 -2.0718 -0.0718  2.5790  2.9282

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.07180    0.03052   2.353   0.0186 *
genderFemale  0.34924    0.04092   8.535   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.238 on 12112 degrees of freedom
Multiple R-squared:  0.005979,      Adjusted R-squared:  0.005897
F-statistic: 72.85 on 1 and 12112 DF,  p-value: < 2.2e-16
```

Let's try a very basic model.

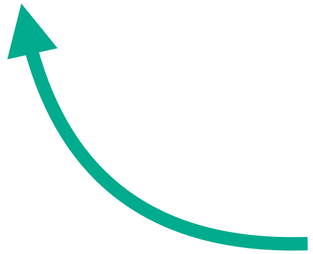
```
# OLS regression
model_1 <- lm(party_scale ~ gender, data=df_train)
summary(model_1)

Call:
lm(formula = party_scale ~ gender, data = df_train)

Residuals:
    Min      1Q  Median      3Q     Max
-3.4210 -2.0718 -0.0718  2.5790  2.9282

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.07180    0.03052   2.353   0.0186 *
genderFemale  0.34924    0.04092   8.535   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.238 on 12112 degrees of freedom
Multiple R-squared:  0.005979,      Adjusted R-squared:  0.005897
F-statistic: 72.85 on 1 and 12112 DF,  p-value: < 2.2e-16
```

One way of evaluating predictive models is R-squared. It's less ideal than (R)MSE, which is measured in units of our outcome, but R-squared can be useful at a glance.

Here, the R-squared is low, so maybe we should temper our expectations…

Let's evaluate our predictions!

```
# In-sample prediction & MSE
df_train$predict <- predict(model_1, df_train)
df_train %>% summarise(mse = mean((party_scale - predict)^2))

       mse
1 5.005678

# Out-of-sample prediction & MSE
df_test$predict <- predict(model_1, df_test)
df_test %>% summarise(mse = mean((party_scale - predict)^2))

       mse
1 5.016616
```



To compute the MSE, we take the error between the true and predicted values, square them, and take the mean.

Hence, mean squared error.

Let's evaluate our predictions!

```
# In-sample prediction & MSE
df_train$predict <- predict(model_1, df_train)
df_train %>% summarise(mse = mean((party_scale - predict)^2))

       mse
1 5.005678

# Out-of-sample prediction & MSE
df_test$predict <- predict(model_1, df_test)
df_test %>% summarise(mse = mean((party_scale - predict)^2))

       mse
1 5.016616
```

To get the RMSE, we just take the square root:
sqr(5.01) = 2.24
sqr(5.02) = 2.24

Remember that our scale is –3 to 3, so on average, we are off by a fair amount!

Also, notice that the in-sample predictions are a tiny bit better than out-of-sample predictions.

Let's try a more complicated model.

```
# OLS regression
model_2 <- lm(party_scale ~ age + gender + race_eth + education, data=df_train)
summary(model_2)

Call:
lm(formula = party_scale ~ age + gender + race_eth + education,
    data = df_train)

Residuals:
    Min      1Q  Median      3Q     Max
-5.8095 -1.9455  0.1316  1.7723  3.9824

Coefficients:
                                      Estimate Std. Error t value Pr(>|t|)
(Intercept)                           1.479182   0.133753  11.059  < 2e-16 ***
age                                  -0.014695   0.001157 -12.696  < 2e-16 ***
genderFemale                          0.232517   0.038988   5.964 2.53e-09 ***
race_ethBlack                         1.406425   0.138982  10.119  < 2e-16 ***
race_ethHispanic/Latino               0.146127   0.137354   1.064   0.2874
race_ethNative American              -0.429021   0.216037  -1.986   0.0471 *
race_ethOther                        -0.245887   0.164220  -1.497   0.1343
race_ethWhite                        -0.519662   0.127776  -4.067 4.79e-05 ***
educationHigh school graduate        -0.722277   0.051289 -14.082  < 2e-16 ***
educationLess than high school       -0.539031   0.093625  -5.757 8.75e-09 ***
educationSome college or 2-year degree -0.518299   0.047272 -10.964  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.119 on 12103 degrees of freedom
Multiple R-squared:  0.1092, Adjusted R-squared:  0.1085
F-statistic: 148.3 on 10 and 12103 DF,  p-value: < 2.2e-16
```

Let's try a more complicated model.

```
# OLS regression
model_2 <- lm(party_scale ~ age + gender + race_eth + education, data=df_train)
summary(model_2)

Call:
lm(formula = party_scale ~ age + gender + race_eth + education,
    data = df_train)

Residuals:
    Min      1Q  Median      3Q     Max
-5.8095 -1.9455  0.1316  1.7723  3.9824

Coefficients:
                                      Estimate Std. Error t value Pr(>|t|)
(Intercept)                           1.479182   0.133753  11.059  < 2e-16 ***
age                                  -0.014695   0.001157 -12.696  < 2e-16 ***
genderFemale                          0.232517   0.038988   5.964 2.53e-09 ***
race_ethBlack                         1.406425   0.138982  10.119  < 2e-16 ***
race_ethHispanic/Latino               0.146127   0.137354   1.064   0.2874
race_ethNative American              -0.429021   0.216037  -1.986   0.0471 *
race_ethOther                        -0.245887   0.164220  -1.497   0.1343
race_ethWhite                        -0.519662   0.127776  -4.067 4.79e-05 ***
educationHigh school graduate        -0.722277   0.051289 -14.082  < 2e-16 ***
educationLess than high school       -0.539031   0.093625  -5.757 8.75e-09 ***
educationSome college or 2-year degree -0.518299   0.047272 -10.964  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.119 on 12103 degrees of freedom
Multiple R-squared:  0.1092, Adjusted R-squared:  0.1085
F-statistic: 148.3 on 10 and 12103 DF,  p-value: < 2.2e-16
```

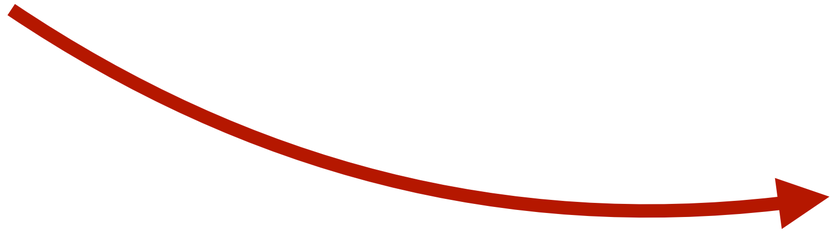
The R-squared went up: 0.01 vs. 0.11. That's a good sign!

Let’s evaluate our predictions!

```
# In-sample prediction & MSE
df_train$predict_2 <- predict(model_2, df_train)
df_train %>% summarise(mse = mean((party_scale - predict_2)^2))

       mse
1 4.485947

# Out-of-sample prediction & MSE
df_test$predict_2 <- predict(model_2, df_test)
df_test %>% summarise(mse = mean((party_scale - predict_2)^2))

       mse
1 4.553173
```

Let's evaluate our predictions!

```
# In-sample prediction & MSE
df_train$predict_2 <- predict(model_2, df_train)
df_train %>% summarise(mse = mean((party_scale - predict_2)^2))

       mse
1 4.485947

# Out-of-sample prediction & MSE
df_test$predict_2 <- predict(model_2, df_test)
df_test %>% summarise(mse = mean((party_scale - predict_2)^2))

       mse
1 4.553173
```



To get the RMSE, we take the square root:
sqr(4.49) = 2.12
sqr(4.55) = 2.13

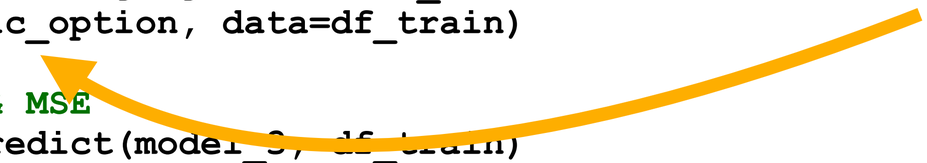
This is better than our last model!


Making predictions with random models.
“I have no idea what I’m doing,”
Making predictions with random models.
“But I know I’m doing it well”

Let's bring out the big guns.

We'll use every variable and interact them!

```
# OLS regression
model_3 <- lm(party_scale ~ age*gender*race_eth*education*
marital*public_ins*public_option, data=df_train)

# In-sample prediction & MSE
df_train$predict_3 <- predict(model_3, df_train)
df_train %>% summarise(mse = mean((party_scale - predict_3)^2))

       mse
1 3.585402

# Out-of-sample prediction & MSE
df_test$predict_3 <- predict(model_3, df_test)
df_test %>% summarise(mse = mean((party_scale - predict_3)^2))

       mse
1 252.4782
```



…including some variables, like political opinions, that are causally downstream of party affiliation!

But that's OK: Our goal is predictive accuracy, not causal identification.

Let's bring out the big guns.

We'll use every variable and interact them!

```
# OLS regression
model_3 <- lm(party_scale ~ age*gender*race_eth*education*
marital*public_ins*public_option, data=df_train)

# In-sample prediction & MSE
df_train$predict_3 <- predict(model_3, df_train)
df_train %>% summarise(mse = mean((party_scale - predict_3)^2))

       mse
1 3.585402

# Out-of-sample prediction & MSE
df_test$predict_3 <- predict(model_3, df_test)
df_test %>% summarise(mse = mean((party_scale - predict_3)^2))

       mse
1 252.4782
```

Our in-sample predictions got better…
sqr(3.59) = 1.89

…but out-of-sample, they're terrible!
sqr(252.48) = 15.89

This is what happens when we "overfit" our model.
We have only 12,114 people in the training set yet fit 1,920 coefficients!

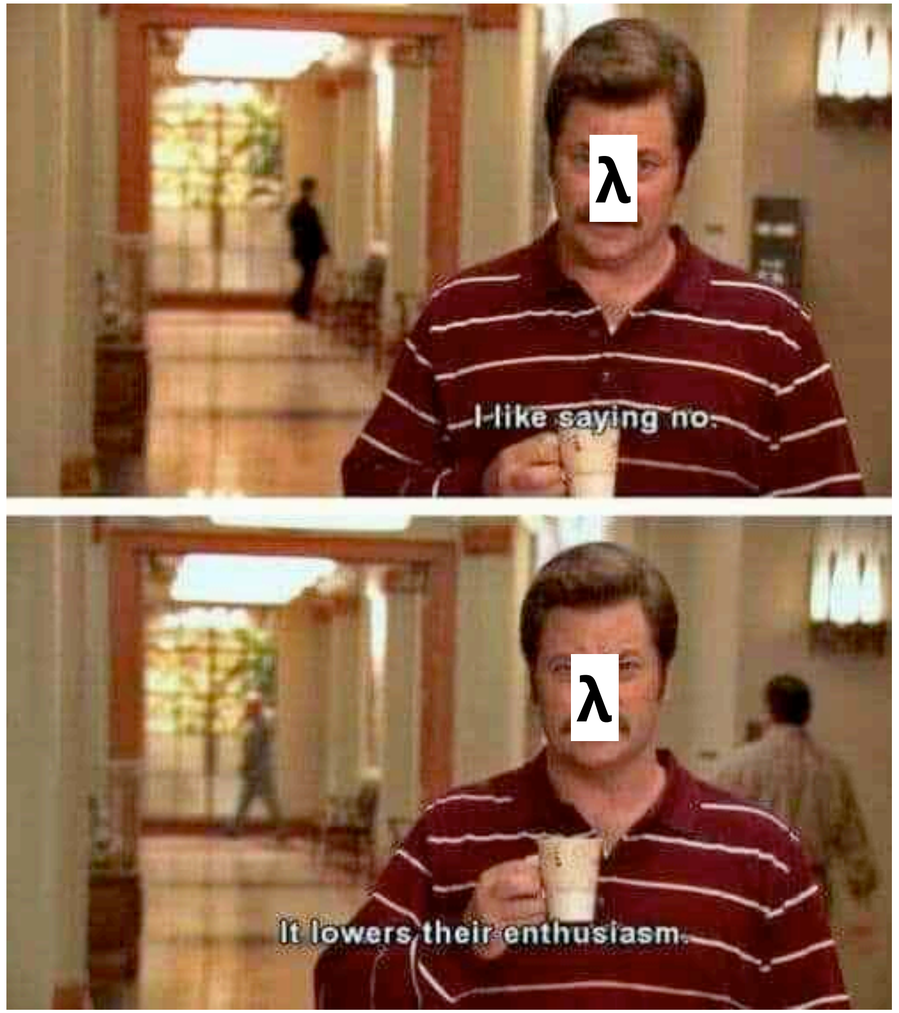
We need some guardrails.

Enter: LASSO regression.

This is normal OLS.

$$\hat{\beta}_{\lambda}^{LASSO} = \arg\min_{\beta} \left[\underbrace{\sum_{i=1}^{n} \left(y_i - (\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}) \right)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=1}^{p} |\beta_j|}_{\text{Regularizer}} \right]$$

Where $\lambda \geq 0$ is called a *tuning parameter*.



This penalizes us for more and larger betas.

This way, we reach a point at which adding more betas doesn't improve fit.


There's only one thing I hate more than OLS, LASSO .
Which is OLS that's lying about being something new and fancy

Alright, here we go.

First, we will load the glmnet library.

```
# Load library
library(glmnet)

# Set the variables that are "fair game"
Y <- df_train$party_scale
X <- data.matrix(df_train[, c("age", "gender", "race_eth", "education",
                              "marital", "public_ins", "public_option")])

# LASSO regression
lasso <- cv.glmnet(x=X, y=Y)
plot(lasso); log(lasso$lambda.min)

[1] -6.125561
```

Second, we will select the Y and X variables that we want to use.

Third, we run the model.

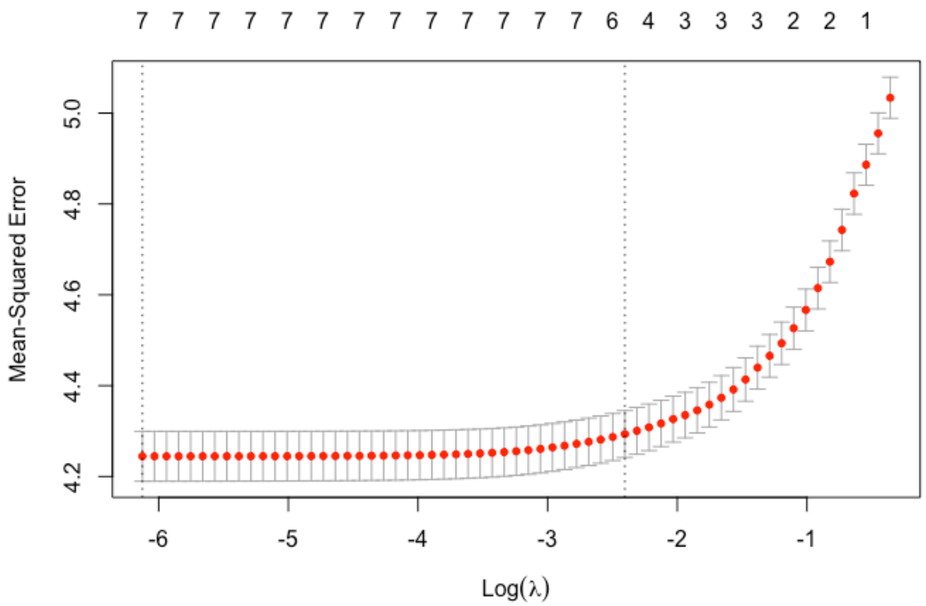
Lastly, we will evaluate the lambda that was selected.

Alright, here we go.

```
# Load library
library(glmnet)

# Set the variables that are "fair game"
Y <- df_train$party_scale
X <- data.matrix(df_train[, c("age", "gender", "race_eth", "education",
                              "marital", "public_ins", "public_option")])

# LASSO regression
lasso <- cv.glmnet(x=X, y=Y)
plot(lasso); log(lasso$lambda.min)

[1] -6.125561
```

Here's all the log(lambdas) plotted.

We want the minimum lambda, as this value minimizes the MSE.



Show me the predictions already!


```
# In-sample prediction & MSE
df_train$predict_4 <- predict(lasso, newx=X, s="lambda.min")[, 1]
df_train %>% summarise(mse = mean((party_scale - predict_4)^2))

       mse
1 4.237769

# Get X variables of test set
X_test <- data.matrix(df_test[, c("age", "gender", "race_eth", "education",
                                  "marital", "public_ins", "public_option")])

# Out-of-sample prediction & MSE
df_test$predict_4 <- predict(lasso, newx=X_test, s="lambda.min")[, 1]
df_test %>% summarise(mse = mean((party_scale - predict_4)^2))

       mse
1 4.185614
```

Show me the predictions already!

```
# In-sample prediction & MSE
df_train$predict_4 <- predict(lasso, newx=X, s="lambda.min")[, 1]
df_train %>% summarise(mse = mean((party_scale - predict_4)^2))

       mse
1 4.237769
```

In-sample RMSE: sqr(4.24) = 2.06

```
# Get X variables of test set
X_test <- data.matrix(df_test[, c("age", "gender", "race_eth", "education",
                                  "marital", "public_ins", "public_option")])

# Out-of-sample prediction & MSE
df_test$predict_4 <- predict(lasso, newx=X_test, s="lambda.min")[, 1]
df_test %>% summarise(mse = mean((party_scale - predict_4)^2))

       mse
1 4.185614
```

Out-of-sample RMSE: sqr(4.24) = 2.05

So far, this is our best model! That said, performance will vary by context and available data.


LASSO
OLS loves it when you can show OLS you're better than it
are at something it loves

Let’s compare our models.

	In-sample RMSE	Out-of-sample RMSE
OLS (gender)	**2.24**	**2.24**
OLS (age + gender + race_eth + education)	**2.12**	**2.13**
OLS (age*gender*race_eth*education* marital*public_ins*public_option)	**1.89**	**15.89**
LASSO (age + gender + race_eth + education + marital + public_ins + public_option)	**2.06**	**2.05**

Of the models we tried, the LASSO one performs best.

Is there a better way…?

Predictions, Part 1.

Remember to keep your eye on the prize: It's all about predictive accuracy.

Be careful not to overfit the model.

LASSO regression can help reduce overfitting by penalizing extra terms.

But there are even better ways…