Regularized Regression

⛰️ 🤠 🕸️

Applied Machine Learning in R
Pittsburgh Summer Methodology Series

Lecture 3-A July 21, 2021

Overview

Lecture Topics

Linear Regression Review

- Ordinary least squares
- Minimizing sum-of-squared-errors
- Limitations of OLS regression

Introduction to Regularization

- Why regularize?
- Bias-variance tradeoff
- Coefficient paths
- Feature selection
- Tuning hyperparameters

Ridge

- L_2 penalty
- Parameter shrinkage towards zero

Lasso

- L_1 penalty
- Some parameters actually go to zero

Elastic Net

- Combining penalty terms
- λ and α tuning parameters

Linear Regression Review

Linear Regression

Linear regression and closely related models (ridge, lasso, elastic net, etc.) can all be written in the form:

$$y_i = b_0 + b_1 x_{i1} + b_2 x_{i2} + \ldots + b_P x_{iP} + e_i$$

where

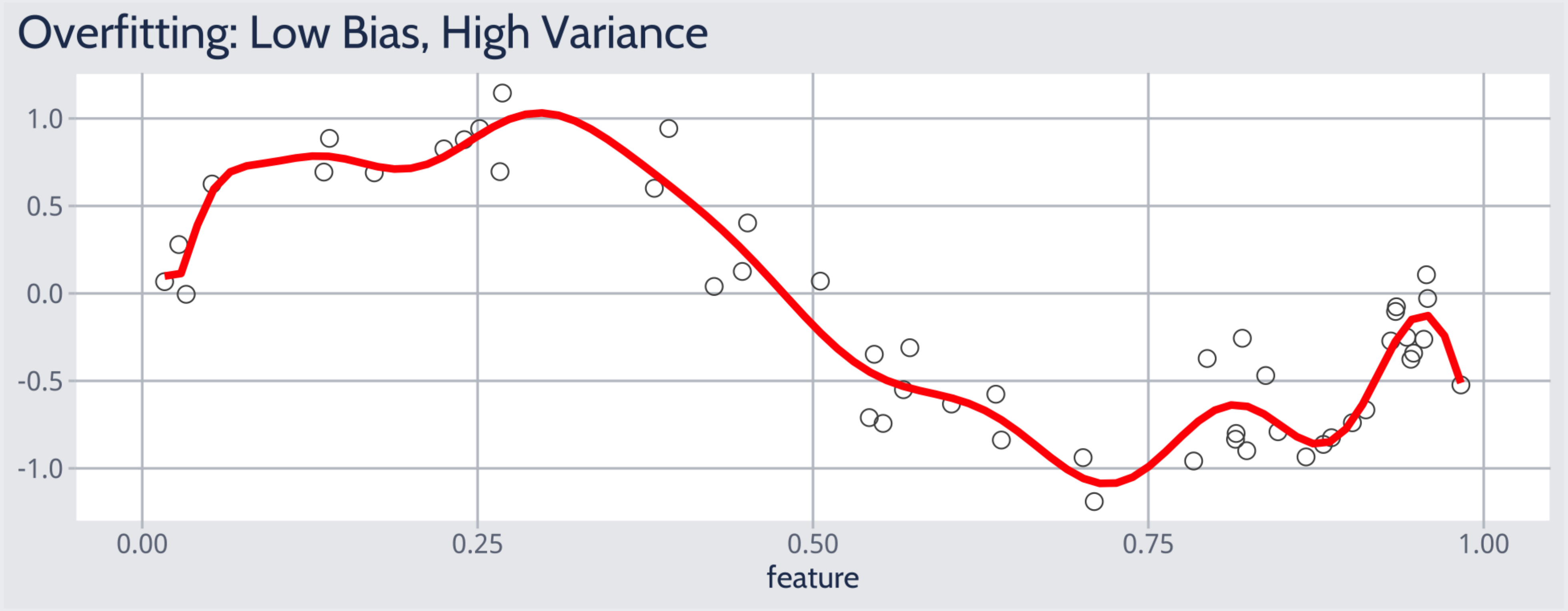
- y_i: value of the response for the ith observation
- b_0: estimated intercept
- b_j: estimated coefficient for the jth predictor
- x_{ij}: value of the jth predictor for the ith observation
- e_i: random error unexplained by the model

Such models are linear in the parameters because each parameter (e.g., b_1, b_2, etc.) appears only with a power of 1 and is not multipled or divided by any other parameter.

Nonlinear *variables* (e.g., x_1^2) can still be included in linear regression as long as the *parameters* remain linear.

Ordinary Least Squares Regression

In ordinary least squares regression, the parameters (e.g., b_1, b_2, etc.) are estimated to minimize model bias at the expense of increasing model variance.



Ordinary Least Squares Regression

Specifically, parameters in OLS regression are chosen to minimize the sum-of-squared errors (SSE):

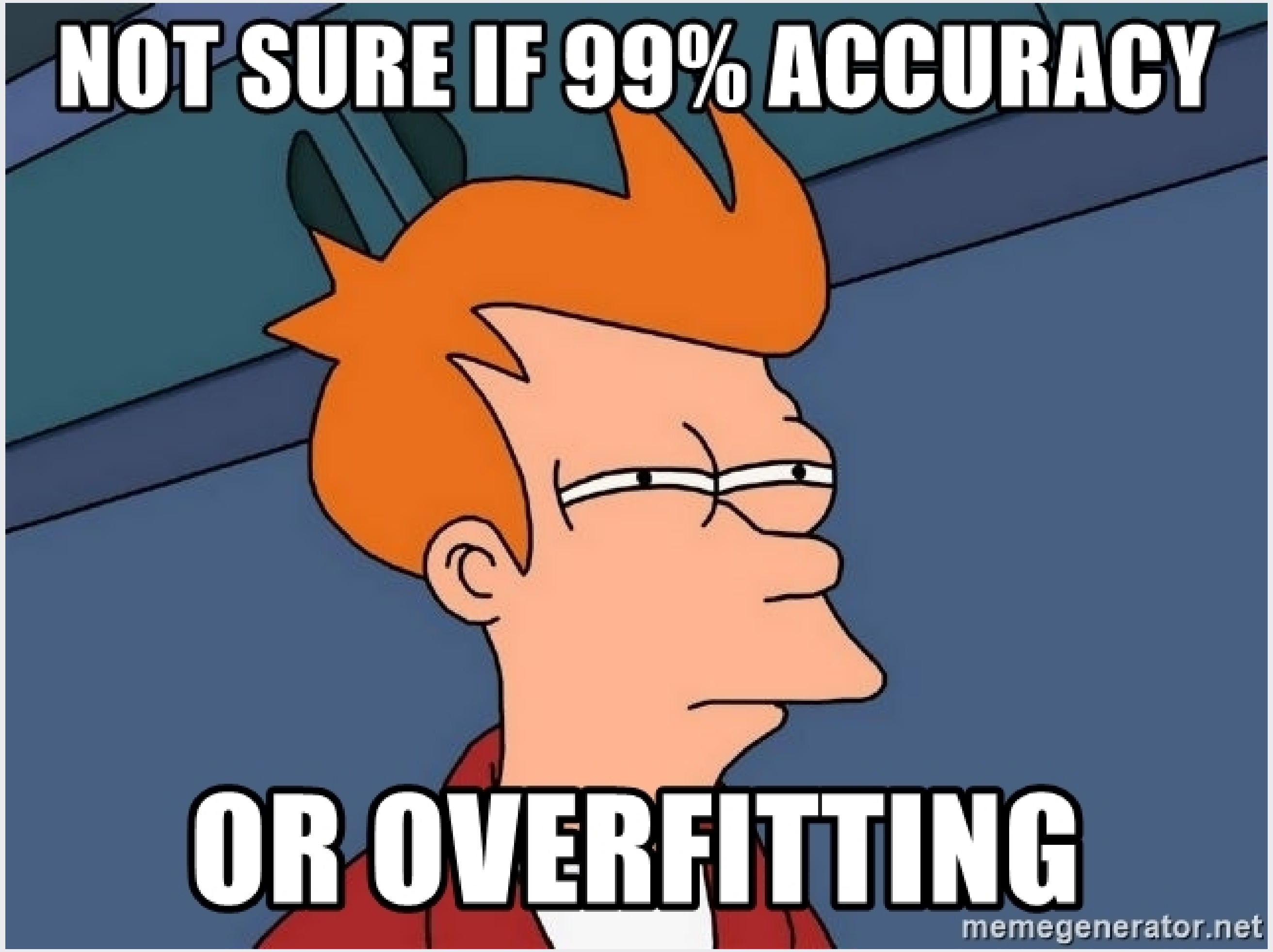
$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where

- n: number of observations
- y_1: ith observed response
- $\hat{y}_i$: ith predicted response

However, any dataset is influenced by both the underlying data-generating mechanisms *and* sampling error, which by definition is not shared with other samples drawn from the same population.

Ordinary Least Squares Regression



Ordinary Least Squares Regression

While OLS regression is readily interpretable and easy to compute, there are several noteworthy limitations:

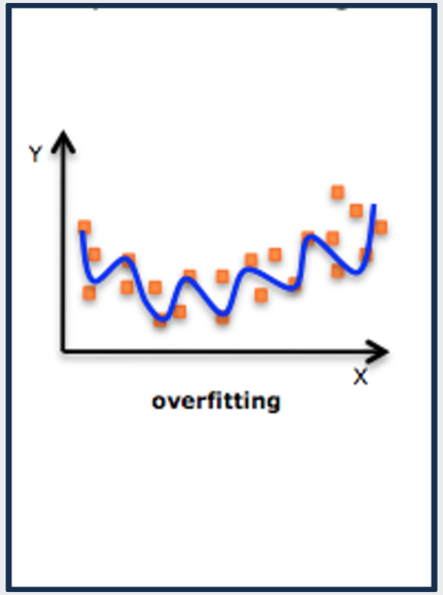
- Risk of overfitting
- Inflated parameter estimates
- Poor predictive accuracy in new datasets
- Sensitive to outliers (adjusting parameter estimates to better accommodate outlier observations with large residuals in order to minimize SSE)
- Cannot handle datasets with high multicollinearity
- Cannot handle datasets with more predictors than observations
- May not adequately capture nonlinear relationships[1]

Regularization addresses many (but not all) of these problems.

[1] This remains a problem with regularized regression models, but see methods for adding nonlinearity to linear models in lecture 2-A (Feature Eningeering).

Regularization

Why regularize?



Reduce Overfitting

OLS regression models overfit the data and inflate parameter estimates.

Regularized regression adds an additional penalty term to the error function, which constrains parameter estimates and penalizes model complexity.

Compared to OLS regression models, regularized regression models have a higher bias but lower variance.

In essense, we make our model less sensitive to the data in the training set in order to achieve higher accuracy in the test set.

Why regularize?

Feature Selection

We are often interested in finding a subset of 'good' predictors.

However, there are are many problems[1] with the traditional approach of stepwise regression models.

One benefit of regularization is that it shrinks parameter estimates towards zero. In some cases, such as lasso regression, some parameters are actually set to zero.

Thus, regularization models can simultaneously reduce overfitting and perform feature selection.

[1] see Harrell (2015) for details and explanation: https://link.springer.com/book/10.1007/978-3-319-19425-7

Comprehension check

Question 1

Determine if each of the following models are linear or nonlinear.

a) $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$

b) $y = \beta_0 + \beta_1^2 x_1 + \beta_2 x_2 + \epsilon$

c) $y = \beta_0 + \beta_1 x_1^2 + \beta_2 \sin(x_1 x_2) + \epsilon$

d) $y = \beta_0 + e^{\beta_1 x_1} + \beta_2 x_2 + \epsilon$

Question 2

Which is not a benefit of regularized compared to nonregularized models?

a) Feature selection

b) Improves out-of-sample prediction

c) Overcomes measurement errors

d) Limits overfitting

Ridge Regression

Ridge Regression

Recall the SSE error function for OLS regression:

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Ridge regression adds a penalty term to this function known as the L_2 penalty:

$$SSE_{L2} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{P} \beta_j^2$$

where

- n: number of observations
- y_1: ith observed response
- $\hat{y}_i$: ith predicted response
- P: number of parameters
- β_j: the jth parameter

Ridge Regression

Whereas OLS regression minimizes the difference between the observed and predicted data, ridge regression has an additional goal of minimizing λ * the squared value of all parameter estimates.

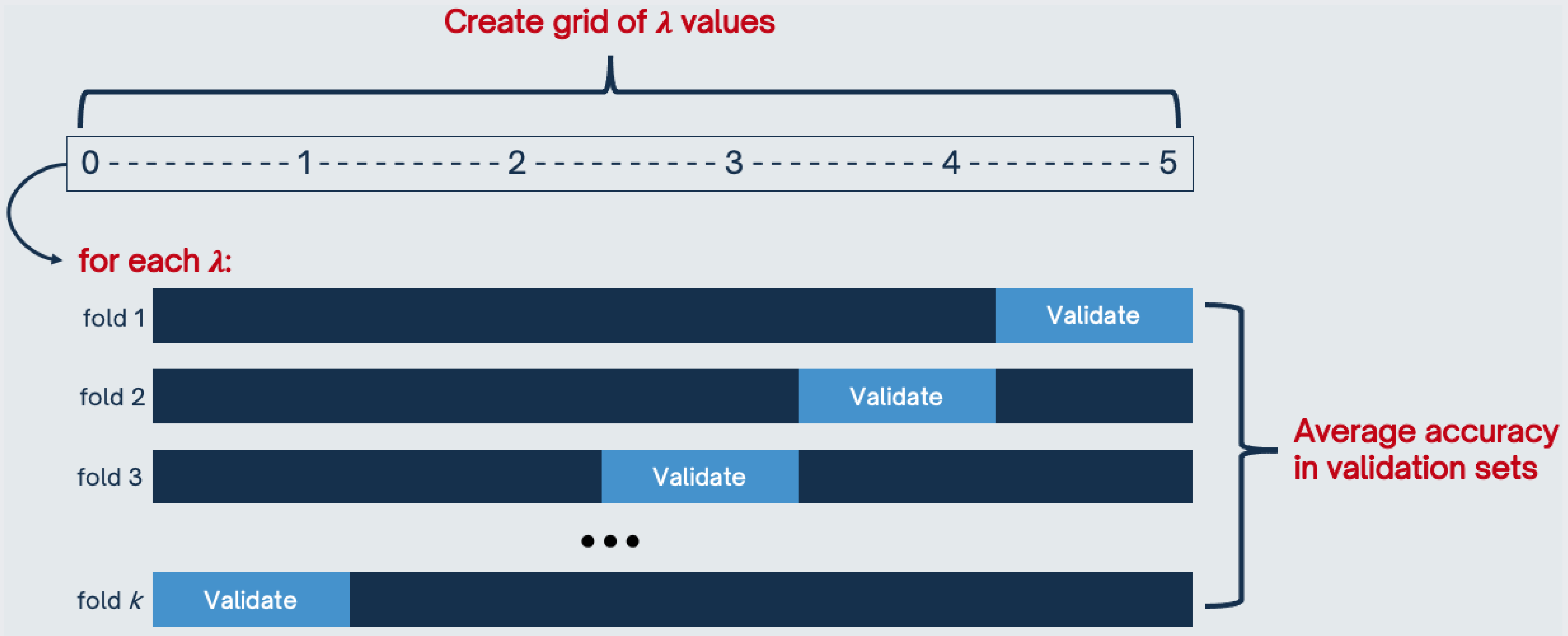
Thus, parameter estimates in ridge regression are only able to become large if there is a proportional reduction in SSE_{L2}.

The λ parameter is a hyperparameter controlling the degree of regularization that ranges from [0, inf].

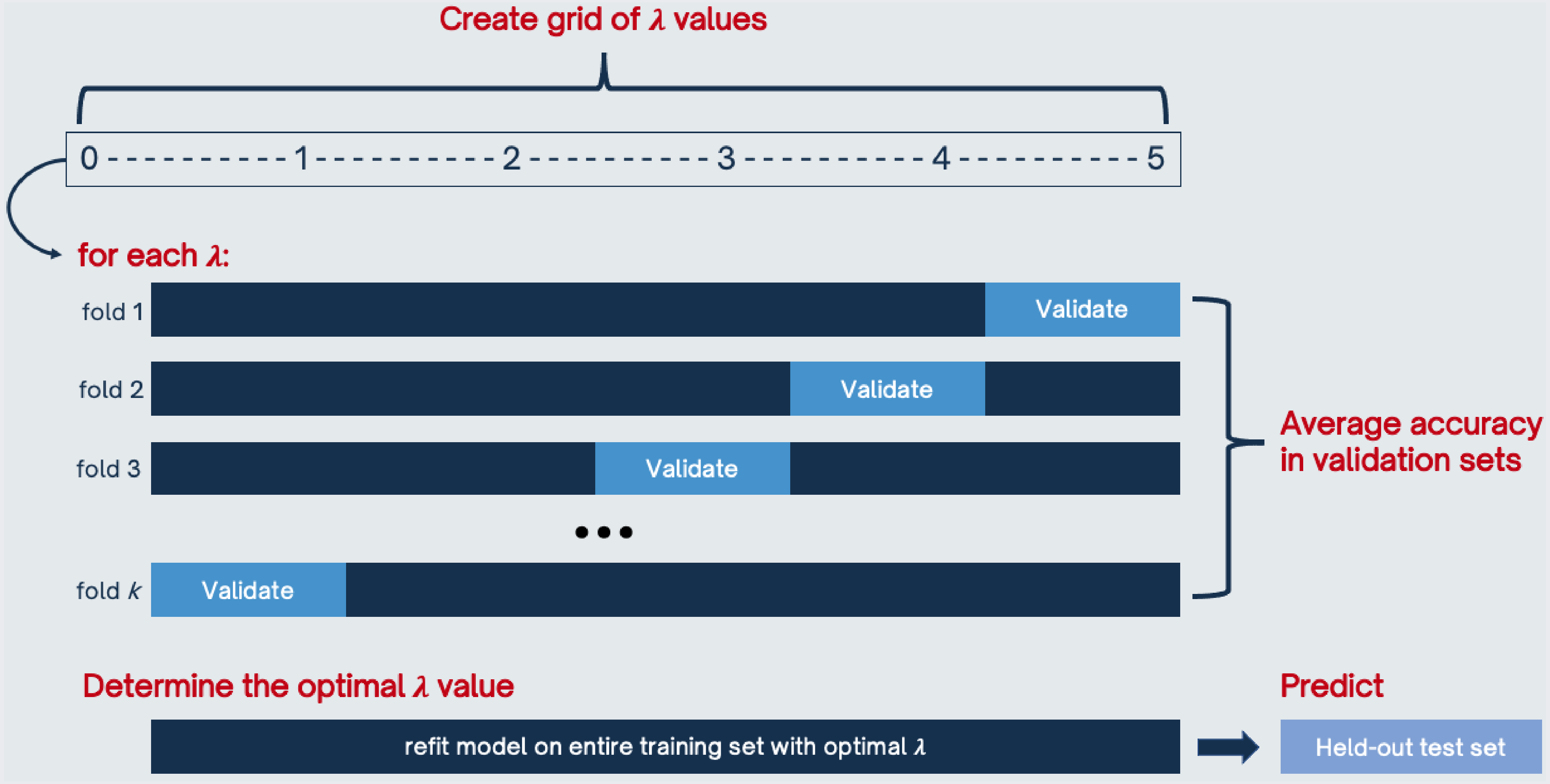
At higher values of λ, parameters will be shrunk closer to zero.

We can find the 'best' value of λ through cross-validation hyperparameter tuning.

Hyperparameter Tuning



Hyperparameter Tuning



Dataset

Today, we'll work with a simulated dataset examining predictors of eating disorder severity[1]. Simulated variables include theoretically *related* and theoretically *unrelated* to eating disorders severity:

- Emotion regulation
- Depression
- Impulsivity
- Self-criticism
- Anxiety
- Race (categorical)
- Age
- Family psychiatric history
- Prior psychiatric treatment
- Length of time being in treatment
- Perfectionism

- Temperature that day
- Average rainfall over past month
- Number of siblings
- Number of cellphones owned
- Time spent reading the news
- Time spent watching TV
- Number of pets owned

[1] Note: Slides will offer a regression example and live coding will offer a classification example in a different dataset.

Ridge Regression in R

Step 0: Split into a training (including cross-validation) and held-out test set.

```r
# use caret for simple data split
library(caret)
set.seed(2021)
trainIndex <- createDataPartition(EDsim$ED_severity, p = 0.8, list = FALSE, times = 1)
EDsim_train <- EDsim[trainIndex, ]
EDsim_test <- EDsim[-trainIndex, ]
```

```r
# check data splitting
dim(EDsim_train)
```

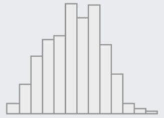
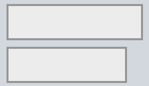
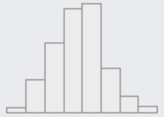
```
## [1] 400  20
```

```r
dim(EDsim_test)
```

```
## [1] 100  20
```

Ridge Regression in R

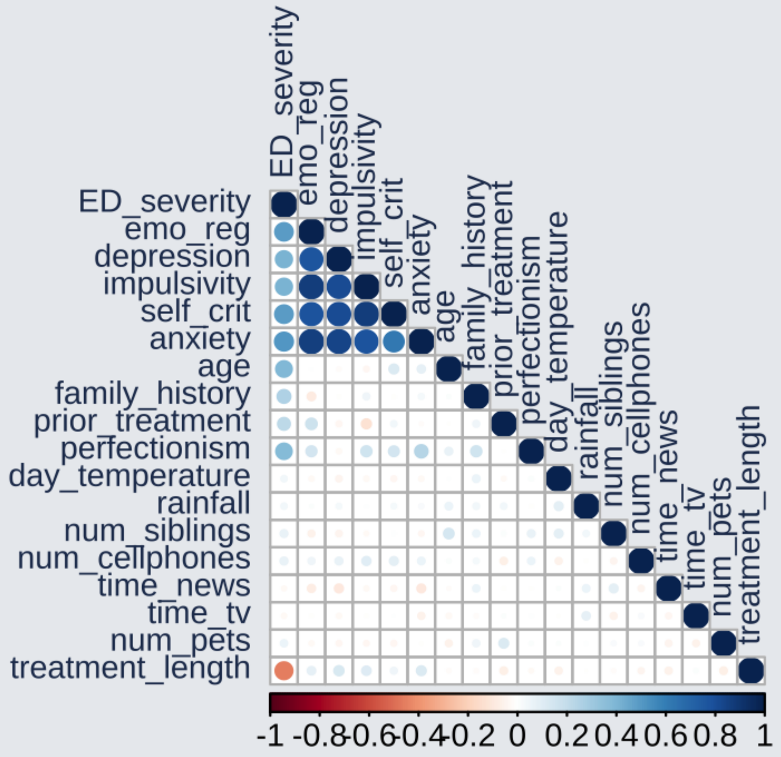
Step 1: Exploratory Data Analysis

No	Variable	Stats / Values	Freqs (% of Valid)	Graph	Missing
1	ED_severity [numeric]	Mean (sd) : -4.2 (2.4) min < med < max: -9.8 < -4.1 < 2.3 IQR (CV) : 3.4 (-0.6)	400 distinct values		0 (0.0%)
2	ED_diagnosis [factor]	1. no_ED 2. ED	212 (53.0%) 188 (47.0%)		0 (0.0%)
3	emo_reg [numeric]	Mean (sd) : 4.8 (1.3) min < med < max: 1.4 < 4.8 < 8.6 IQR (CV) : 1.7 (0.3)	400 distinct values		0 (0.0%)
4	depression	Mean (sd) : 2.7 (1.3) min < med < max: -1.4 < 2.7 <	400 distinct		0 (0.0%)

Generated by summarytools 0.9.9 (R version 4.1.0)
2021-07-14

Ridge Regression in R

Step 1: Exploratory Data Analysis



Ridge Regression in R

Step 2: Feature Engineering[1].

Normalizing (standardizing) features before regularization is important to ensure that the shrinkage parameter λ affects all features equally.

```
EDsim_recipe <-
  EDsim_train %>%
  recipe(ED_severity ~ .) %>%
  step_rm(ED_diagnosis) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_dummy(all_nominal_predictors())
```

[1] Note than when using `recipes`, dummy variables should be created manually with `step_dummy`. If using `caret` by itself without `recipes`, dummy variables are created automatically.

Ridge Regression in R

Step 3: Set model training and tuning methods in `caret::traincontrol()`

```
EDsim_control <- trainControl(method = 'repeatedcv',
                              number = 10,
                              repeats = 3,
                              classProbs = FALSE,
                              summaryFunction = defaultSummary)
```

This sets our up training procedure with the following specifications:

- 10-fold cross-validation, repeated 3 times
- Other commonly used options include `method = 'cv'` for a single (non-repeated) cross-validation and `method = 'boot'` for bootstrapping
- For classification, set `classProbs = TRUE` and `summaryFunction = twoClassSummary` or `summaryFunction = prSummary`

Ridge Regression in R

Note that the cross-validation process involves choosing random indices for data splitting. To ensure reproducibility, you can set seeds to use for resampling in the model training process:

```r
set.seed(2021)
seeds <- vector(mode = "list", length = 30) # length = n data splits
for(i in 1:30) seeds[[i]] <- sample.int(1000, 21)
seeds[[31]] <- sample.int(1000, 1) # for the last model
```

This `seeds` object can be included in the `trainControl` function to obtain the same resampling seeds each time the code is run for fully reproducible results.

```r
EDsim_control <- trainControl(method = 'repeatedcv',
                              number = 10,
                              repeats = 3,
                              classProbs = FALSE,
                              summaryFunction = defaultSummary,
                              seeds = seeds)
```

Ridge Regression in R

Step 4: Set a tuning grid (optional)

There are several ways to set a tuning grid in `caret`.

The first option is to do nothing, as `caret` will automatically create a tuning grid if one is not specified. By default, `caret::train()` creates a tuning grid of size 3^P, where P is the number of tuning parameters.

The second option is to specify the length of the tuning grid in the `caret::train()` function:

```r
train(Y ~ X, method = 'glmnet', tuneLength = 10)
```

The third option is to specify the grid yourself to call in `caret::train()`:

```r
ridgegrid <- expand.grid(alpha = 0, lambda = seq(0, 1, 0.01))
```

This option is useful for training ridge regression models within the `glmnet` method in `caret`, which allows for nice data visualizations. However, `caret` will automatically create a grid of both α and λ values for `method = 'glmnet'`, but we can avoid that by setting the tuning grid ourselves.

Ridge Regression in R

Step 5: Train your model using `caret::train()`

```
set.seed(2021)
ridgefit <- train(EDsim_recipe, data = EDsim_train,
                  method = 'glmnet',
                  tuneGrid = ridgegrid,
                  trControl = EDsim_control)
```

```
ridgefit
```

```
## glmnet
##
## 400 samples
##  19 predictor
##
## Recipe steps: rm, normalize, dummy
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 360, 360, 360, 360, 360, 360, ...
```

Ridge Regression in R

Examine results of internal cross-validation:

```
ridgefit$results
```

```
##      alpha lambda      RMSE  Rsquared       MAE     RMSESD
## 1        0   0.00 0.9858604 0.8273531 0.8502405 0.06838189
## 2        0   0.01 0.9858604 0.8273531 0.8502405 0.06838189
## 3        0   0.02 0.9858604 0.8273531 0.8502405 0.06838189
## 4        0   0.03 0.9858604 0.8273531 0.8502405 0.06838189
## 5        0   0.04 0.9858604 0.8273531 0.8502405 0.06838189
## 6        0   0.05 0.9858604 0.8273531 0.8502405 0.06838189
## 7        0   0.06 0.9858604 0.8273531 0.8502405 0.06838189
```

Ridge Regression in R

Step 6: Predict on held-out test set

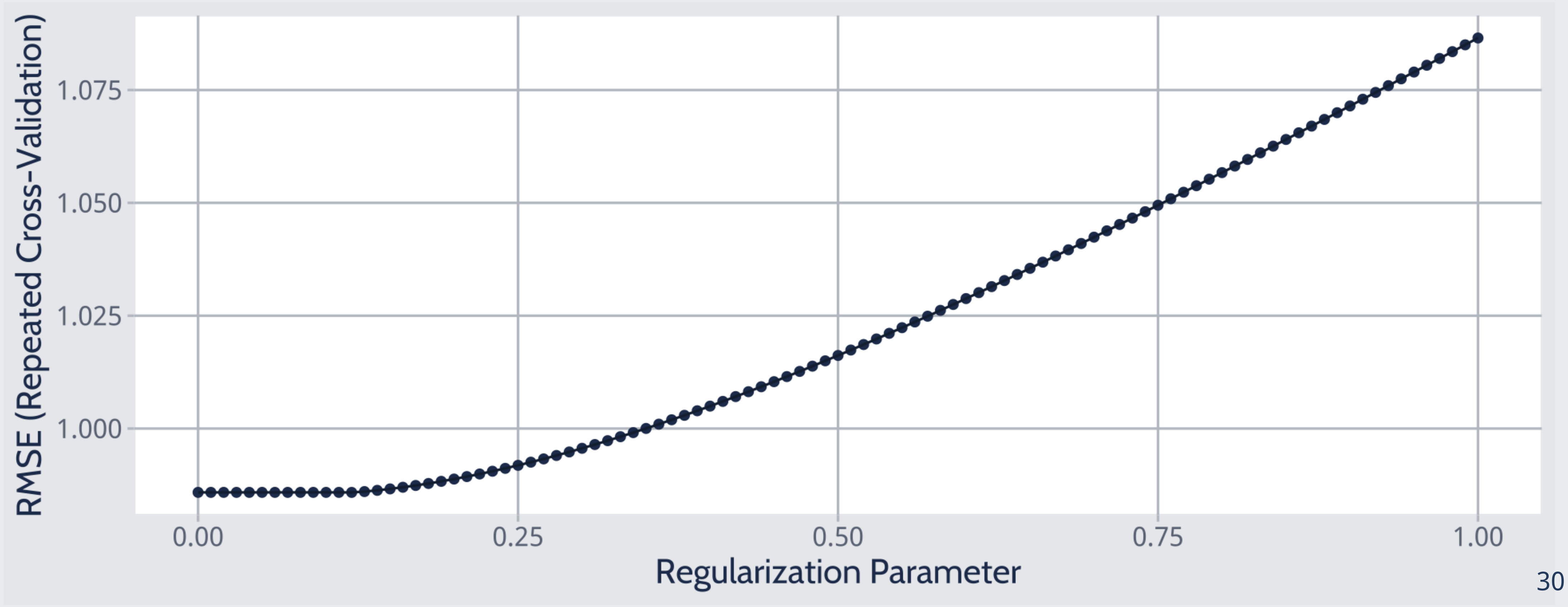
```r
EDsim_test_baked <-
  EDsim_recipe %>%
  prep(training = EDsim_train) %>%
  bake(new_data = EDsim_test)

ridgepred_test <- predict(ridgefit, newdata = EDsim_test)

postResample(pred = ridgepred_test, obs = EDsim_test_baked$ED_severity)
```

```
##      RMSE  Rsquared       MAE
## 0.9425573 0.8630514 0.8213249
```

Ridge Regression in R

Examine accuracy as a function of λ.

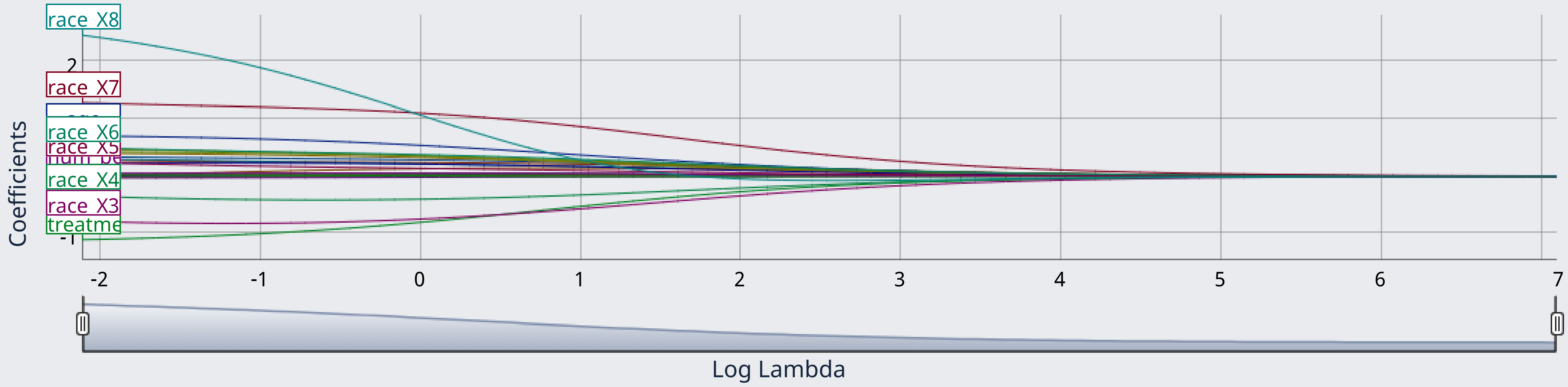
```
ggplot(ridgefit)
```



Ridge Regression in R

Examine coefficients as a function of λ.

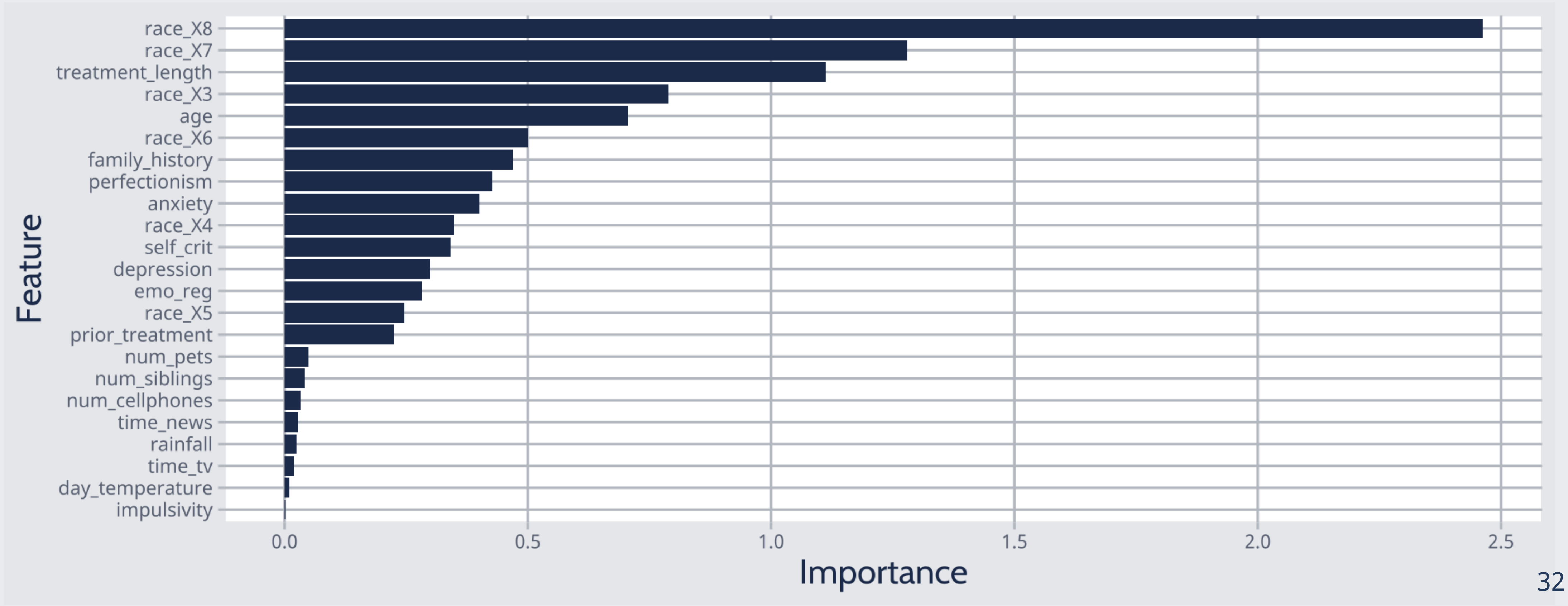
```
library(coefplot)
coefpath(ridgefit$finalModel)
```



Ridge Regression in R

Examine variable importance.

```r
ggplot(varImp(ridgefit, scale = FALSE))
```



Lasso Regression

Lasso Regression

Lasso stands for the Least Absolute Shrinkage and Selection Operator.

Similar to ridge regression, lasso regression adds an additional penalty term (in this case, the $L1$ penalty) to the OLS error function:

$$SSE_{L1} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{P}|\beta_j|$$

where

- n: number of observations
- y_1: ith observed response
- $\hat{y}_i$: ith predicted response
- P: number of parameters
- β_j: the jth parameter

Lasso Regression

Here, we aim to minimize the **absolute value** of all parameters (rather than the square of parameters as with ridge).

While this may seem like a small change, it has the effect of fully penalizing some parameter values down to zero, whereas ridge only shrinks values **towards** zero.

If λ is set high enough, all parameters will be shrunk to zero.

Ridge and lasso also differ in their handling of **multicollinearity.**

Whereas ridge tends to shrink coefficients of correlated predictors towards each other, lasso tends to **pick one and ignore the rest**.

Lasso Regression in R

Set tuning grid and train model

```
lassogrid <- expand.grid(alpha = 1, lambda = seq(0, 1, 0.01))
set.seed(2021)
lassofit <- train(EDsim_recipe, data = EDsim_train,
                  method = 'glmnet',
                  tuneGrid = lassogrid,
                  trControl = EDsim_control)
```

```
lassofit
```

```
## glmnet
##
## 400 samples
##  19 predictor
##
## Recipe steps: rm, normalize, dummy
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 360, 360, 360, 360, 360, 360, ...
```

Lasso Regression in R

Examine results of internal cross-validation:

```
lassofit$results
```

```
##      alpha lambda      RMSE  Rsquared       MAE     RMSESD
## 1        1   0.00 0.9878097 0.8266817 0.8442848 0.07342380
## 2        1   0.01 0.9852440 0.8276105 0.8461214 0.07214010
## 3        1   0.02 0.9830170 0.8285856 0.8466662 0.06822920
## 4        1   0.03 0.9812399 0.8297381 0.8466669 0.06628500
## 5        1   0.04 0.9807786 0.8306284 0.8472703 0.06472962
## 6        1   0.05 0.9821676 0.8310571 0.8489385 0.06366622
## 7        1   0.06 0.9851791 0.8310911 0.8518176 0.06317558
```

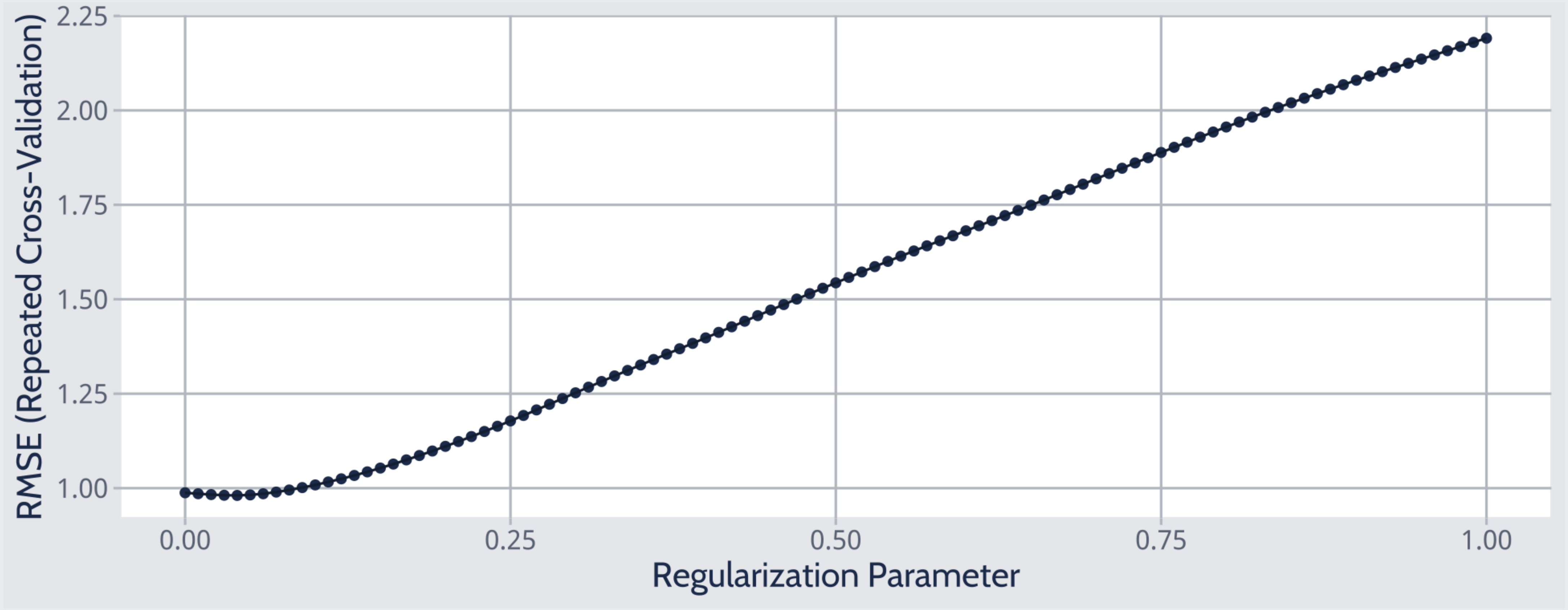
Predict on held-out test set

```
lassopred_test <- predict(lassofit, EDsim_test)
postResample(pred = lassopred_test, obs = EDsim_test_baked$ED_severity)
```

```
##      RMSE  Rsquared       MAE
## 0.9390276 0.8654581 0.8224741
```

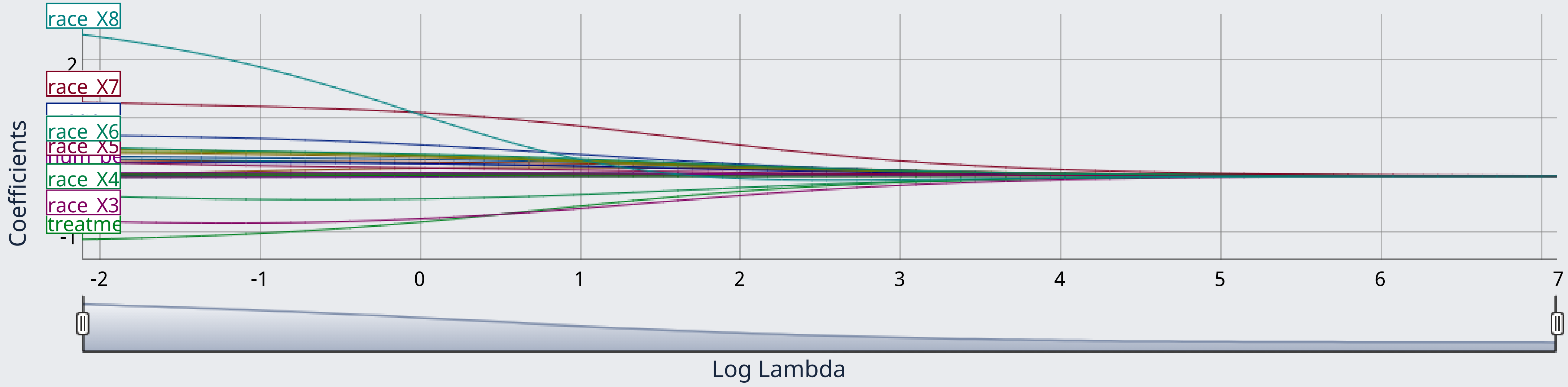
Lasso Regression in R

```
ggplot(lassofit)
```



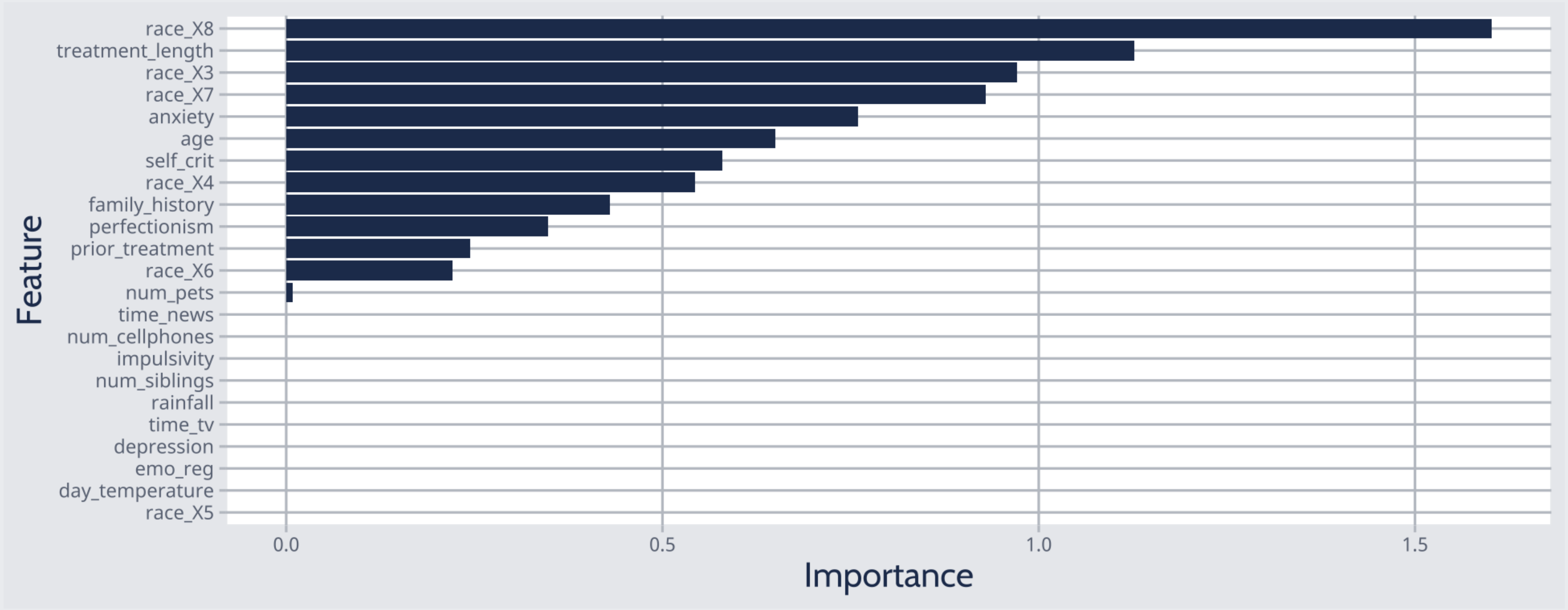
Lasso Regression in R

```
coefpath(lassofit$finalModel)
```



Lasso Regression in R

```
ggplot(varImp(lassofit, scale = FALSE))
```



Elastic Net Regression

Elastic Net Regression

Elastic net regression is a generalization of the lasso model, which combines both $L1$ (lasso) and $L2$ (ridge) penalty terms together in the error function:

$$SSE_{EN} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{P} \beta_j^2 + \lambda_2 \sum_{j=1}^{P} |\beta_j|$$

where

- n: number of observations
- y_1: ith observed response
- $\hat{y}_i$: ith predicted response
- P: number of parameters
- β_j: the jth parameter

Elastic Net

The elastic net model allows for effective ridge-like regularization with lasso-like feature selection.

In `caret`, we now tune two hyperparameters: λ and α.

As with ridge and lasso, λ controls the degree of regularization.

The new α hyperparameter is called the elastic net mixing parameter, and ranges from [0, 1].

At $\alpha = 0$ ridge is performed and at $\alpha = 1$ lasso is performed.

Elastic net is particularly good at handling correlated predictors.

Elastic Net in R

Set tuning grid and train model

```
enfit <- train(EDsim_recipe, data = EDsim_train,
               method = 'glmnet',
               tuneLength = 20,
               trControl = EDsim_control)
```

```
enfit
```

```
## glmnet
##
## 400 samples
##  19 predictor
##
## Recipe steps: rm, normalize, dummy
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 360, 360, 360, 360, 360, 360, ...
```

Elastic Net in R

Examine results of internal cross-validation:

```
enfit$results
```

```
##          alpha       lambda      RMSE  Rsquared       MAE
## 1   0.1000000 0.0003766177 0.9882897 0.8301414 0.8455116
## 2   0.1000000 0.0005839516 0.9882897 0.8301414 0.8455116
## 3   0.1000000 0.0009054262 0.9882897 0.8301414 0.8455116
## 4   0.1000000 0.0014038776 0.9882897 0.8301414 0.8455116
## 5   0.1000000 0.0021767344 0.9882886 0.8301424 0.8455110
## 6   0.1000000 0.0033750611 0.9880892 0.8302132 0.8454500
## 7   0.1000000 0.0052330856 0.9876279 0.8303726 0.8453460
```

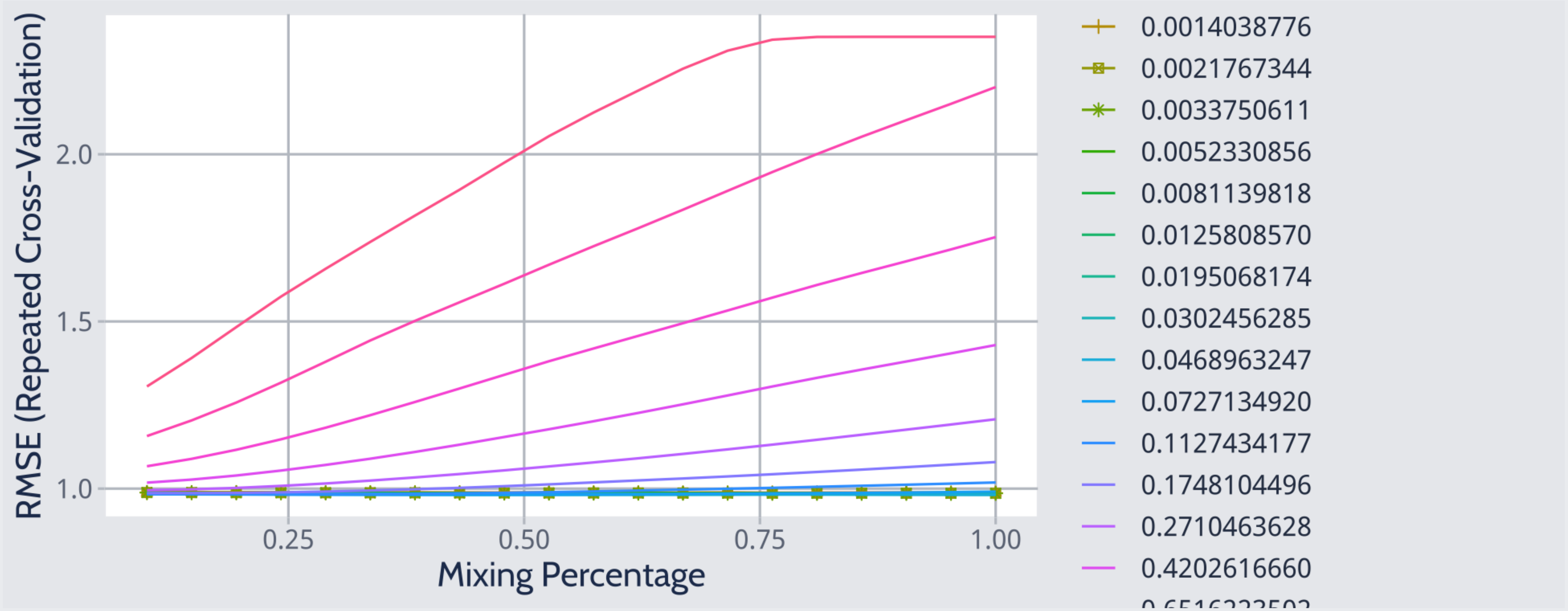
Predict on held-out test set

```
enpred_test <- predict(enfit, EDsim_test)
postResample(pred = enpred_test, obs = EDsim_test_baked$ED_severity)
```

```
##      RMSE  Rsquared       MAE
## 0.9367022 0.8652197 0.8187489
```

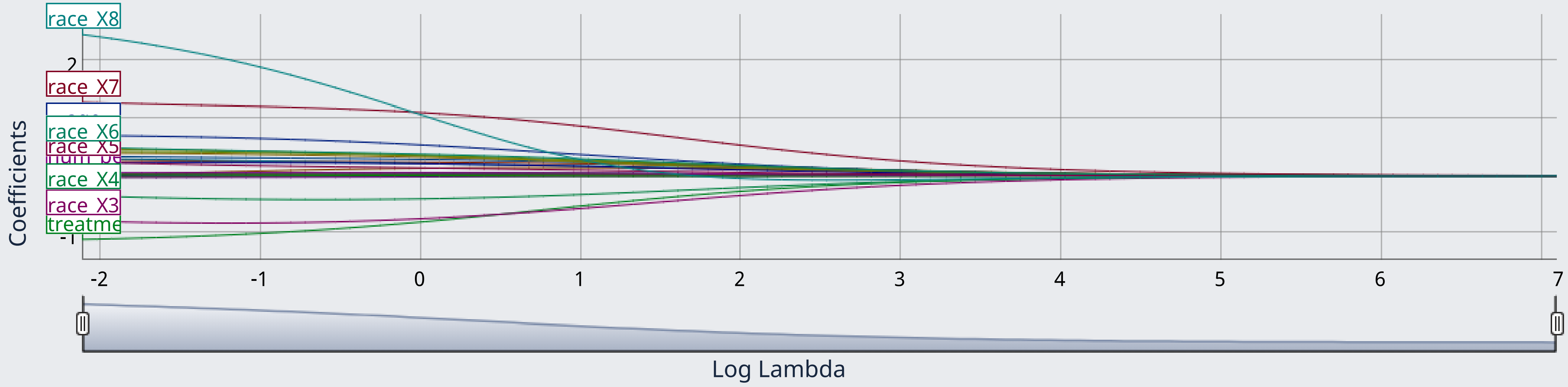
Elastic Net in R

```r
ggplot(enfit)
```



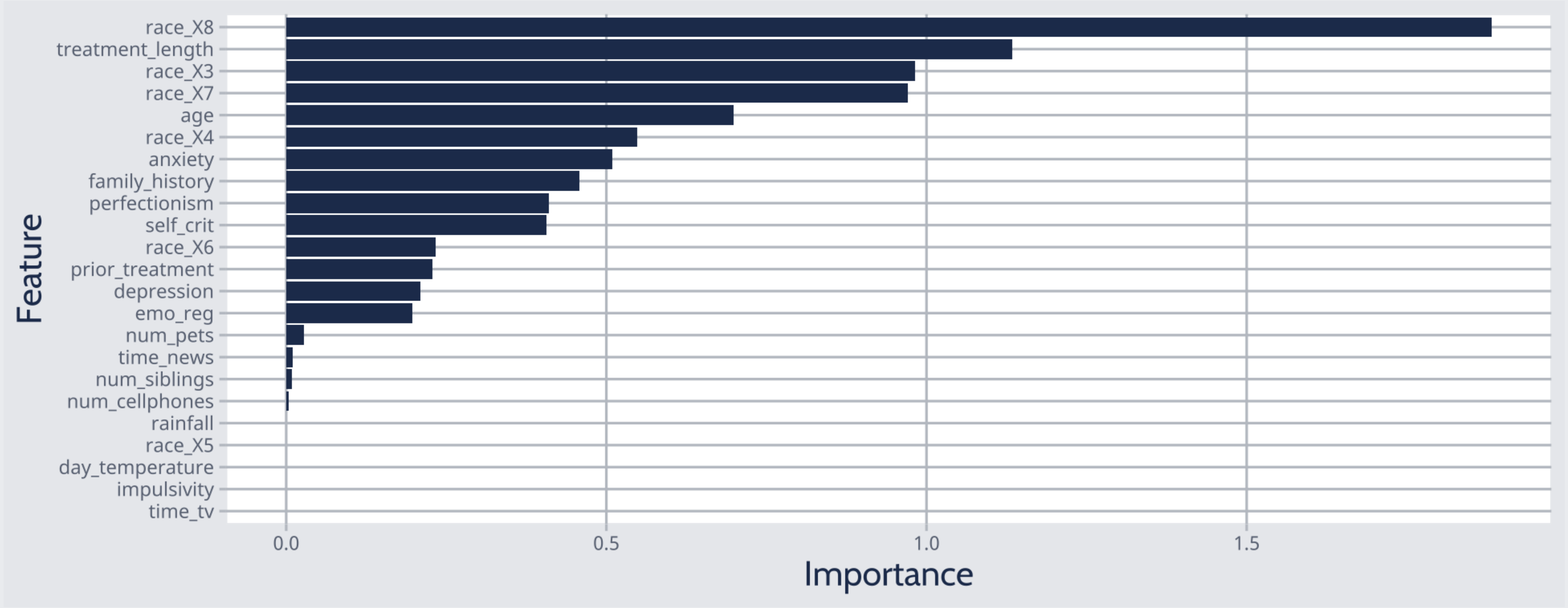
Elastic Net in R

```
coefpath(enfit$finalModel)
```



Elastic Net in R

```r
ggplot(varImp(enfit, scale = FALSE))
```



Comprehension check

Question 1

Which model uses an L_2 squared penalty?

a) Ridge

b) Lasso

c) Elastic Net

d) None of the above

Question 2

What do the λ and α hyperparameters refer to?

a) λ = shrinkage, α = validation

b) λ = feature selection, α = mixing

c) λ = shrinkage, α = mixing

d) λ = feature selection, α = shrinkage

Live Coding

Live Coding Activity

Live Coding: I will walk through examples of ridge, lasso, and elastic net for classification in RStudio.

- You can follow along in your own RStudio (easiest with 1 large or 2+ monitors)
- Or you can download `Day_3A_Activity.Rmd` and follow along
- In either case, please download the `heart.csv` and `hcp_memory.csv` files.

Small Group Activity: Afterwards, we will split you into small breakout room groups to practice a full machine learning workflow using a new dataset.

If you have any questions, please post them in the chat or workshop Slack channel.

We will also float between different breakout rooms to answer questions.

All files are on the workshop OSF page: https://osf.io/3qhc8/.