Predictive Analytics

Prediction vs. Inference

So far, when we have been looking at data, we've been dealing with trying to figure out how to explain what is going on within the data. Oftentimes with dataset we are exploring, we are looking to be able to properly infer what happened in the data and we want to be able to interpret the findings that we receive.

However, other times you are going to want to apply what you have learned to other situations and datasets. This is where prediction comes in to play. Not only do we want to be able to understand the variables that impact an outcome, we also want to know if these variables will impact the same outcome in several situations.

With predictive models, sometimes the interpretability is less important than the accuracy. It can become a difficult balancing act between interpretability, accuracy, and other metrics such as flexibility.

Today we are going to explore how to create predictive models and how to utilize machine learning techniques to create usable models.

Set up

We are going to be using the `tidymodels` package today to build models. This is a process that requires several packages, such as `glmnet`, `caret`, and `ranger`.

```
library(rsample)
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 0.1.3 --

## v broom        0.7.6      v recipes      0.1.16
## v dials        0.0.9      v tibble       3.1.0
## v dplyr        1.0.5      v tidyr        1.1.3
## v ggplot2      3.3.3      v tune         0.1.4
## v infer        0.5.4      v workflows    0.2.2
## v modeldata    0.1.0      v workflowsets 0.0.2
## v parsnip      0.1.5      v yardstick    0.0.8
## v purrr        0.3.4

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x recipes::step()  masks stats::step()
## * Use tidymodels_prefer() to resolve common conflicts.
```

```r
library(skimr)
#install.packages("ranger")
#install.packages("glmnet")
#install.packages("caret")
library(ranger)
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-1
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following objects are masked from 'package:yardstick':
##
##     precision, recall, sensitivity, specificity

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v readr   1.4.0     v forcats 0.5.1
## v stringr 1.4.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x readr::col_factor() masks scales::col_factor()
## x purrr::discard()    masks scales::discard()
## x Matrix::expand()    masks tidyr::expand()
## x dplyr::filter()     masks stats::filter()
## x stringr::fixed()    masks recipes::fixed()
## x dplyr::lag()        masks stats::lag()
## x caret::lift()       masks purrr::lift()
## x Matrix::pack()      masks tidyr::pack()
## x readr::spec()       masks yardstick::spec()
## x Matrix::unpack()    masks tidyr::unpack()
```

```r
library(dotwhisker)  # for visualizing regression results
#install.packages("vip")
library(vip)
```

```
##
## Attaching package: 'vip'

## The following object is masked from 'package:utils':
##
##     vi
```

We are going to use the datasets we used last week. The Life Expectancy dataset will allow us to try and predict the Life Expectancy of a given country under various circumstances. The Heart Failure dataset will allow us to predict the probability that some will die after Heart Failure and perhaps intervene.

To get started, we need to read the datasets in and do a bit of cleaning.

```r
# Reading in Life Expectancy data and getting rid of variables that may be a bit too predictive (counte
le <-read_csv("Life Expectancy Data.csv") %>% select(-`infant deaths`, -`Adult Mortality`, -`under-five
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Country = col_character(),
##   Status = col_character()
## )
## i Use `spec()` for the full column specifications.
```

```r
names(le) <- str_replace_all(names(le), " ", "_")
names(le) <- str_replace_all(names(le), "-", "_")
names(le) <- str_replace_all(names(le), "/", "_")
logvars <- c("Alcohol", "GDP")

le <- le %>%
  drop_na() %>%
  mutate(Developed = ifelse(Status == "Developed", 1, 0)) %>%
  mutate_if(is.character, as.factor) %>%
  mutate_at(logvars, log) %>%
  select(-Status)
```

```r
hf <- read_csv("heart_failure_clinical_records_dataset.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   age = col_double(),
##   anaemia = col_double(),
##   creatinine_phosphokinase = col_double(),
##   diabetes = col_double(),
##   ejection_fraction = col_double(),
```

```
##   high_blood_pressure = col_double(),
##   platelets = col_double(),
##   serum_creatinine = col_double(),
##   serum_sodium = col_double(),
##   sex = col_double(),
##   smoking = col_double(),
##   time = col_double(),
##   DEATH_EVENT = col_double()
## )
```

```
cat_vars <- c("anaemia", "diabetes", "high_blood_pressure", "sex", "smoking", "DEATH_EVENT")
hf <- hf %>%
  mutate_at(cat_vars, factor, ordered = FALSE)
```

Data Splitting

Training and Testing are two of the key aspects of machine learning. So far, we have created models that allow us to explain what is going on within a dataset. This provides us with a good overview of a specific dataset, but it does hinder the ability for us to generalize the findings to other situations.

One way to create a model that should have more flexibility and generalizability is to split your dataset into a training dataset and a testing dataset. The training dataset is used to create an initial model using a portion of the original dataset, while the rest of the original dataset is held out in order to test the efficacy of the model created with the training dataset.

Random Sampling

One way to split the data into a training and testing dataset is to use random sampling. Random sampling will simply grab a random sample of observations based on the proportion given. There are many ways to do this, but we will use the `rsample` package and the `initial_split` function for our purposes.

```
# Fix the random numbers by setting the seed
# This enables the analysis to be reproducible when random numbers are used
set.seed(8382)

# Use the initial_split function to split your dataset by a given proportion.
# Put 3/4 of the data into the training set
le_split <- initial_split(le, prop = 3/4)

# We then create training and testing datasets using the training and testing functions.
train_data <- training(le_split)
test_data  <- testing(le_split)
```

Stratified Random Sampling

Sometimes you want to have a more informative way of splitting a dataset. One way is to use a stratify random sample, which will randomly sample data within whatever group you provide. This could be your outcome variable to ensure that you have an even(ish) amount in each dataset or it could be for a predictor variable that you want to ensure has equal representation across the datasets (race, sex, diabetes status, etc.)

```r
# Put 3/4 of the data into the training set using the Strata option, ensuring a close equal amount of D
hf_split <- initial_split(hf, strata = DEATH_EVENT, prop = 3/4)
hf_split1 <- initial_split(hf, prop = 3/4)

# Create data frames for the two sets:
hf_train_data <- training(hf_split)
hf_test_data  <- testing(hf_split)

hf_train1_data <- training(hf_split1)
hf_test1_data  <- testing(hf_split1)

hf_train_data %>%
  count(DEATH_EVENT) %>%
  mutate(prop = n/nrow(hf_train_data))
```

```
## # A tibble: 2 x 3
##   DEATH_EVENT     n  prop
##   <fct>       <int> <dbl>
## 1 0             153  0.68
## 2 1              72  0.32
```

```r
hf_test_data %>%
  count(DEATH_EVENT) %>%
  mutate(prop = n/nrow(hf_test_data))
```

```
## # A tibble: 2 x 3
##   DEATH_EVENT     n  prop
##   <fct>       <int> <dbl>
## 1 0              50 0.676
## 2 1              24 0.324
```

```r
hf_train1_data %>%
  count(DEATH_EVENT) %>%
  mutate(prop = n/nrow(hf_train1_data))
```

```
## # A tibble: 2 x 3
##   DEATH_EVENT     n  prop
##   <fct>       <int> <dbl>
## 1 0             152 0.676
## 2 1              73 0.324
```

```r
hf_test1_data %>%
  count(DEATH_EVENT) %>%
  mutate(prop = n/nrow(hf_test1_data))
```

```
## # A tibble: 2 x 3
##   DEATH_EVENT     n  prop
##   <fct>       <int> <dbl>
## 1 0              51 0.689
## 2 1              23 0.311
```

Creating a Recipe and Roles

"The recipes package is an alternative method for creating and preprocessing design matrices that can be used for modeling or visualization."

"The idea of the recipes package is to define a recipe or blueprint that can be used to sequentially define the encodings and preprocessing of the data (i.e. "feature engineering")."

```
# The first step in my recipe is to create a quick formula utilizing all of my predictors.
le_rec <-
  recipe(Life_expectancy ~., data = train_data)
le_rec
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         17
```

We are going to hold out the `Country` and `Year` variables to be able to later identify and interesting observations.

We can do this using the `update_role` function

```
le_rec <-
  recipe(Life_expectancy ~., data = train_data) %>%
  update_role(Country, Year, new_role = "ID")
le_rec
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##         ID          2
##    outcome          1
##  predictor         15
```

```
summary(le_rec)
```

```
## # A tibble: 18 x 4
##    variable                        type    role      source
##    <chr>                           <chr>   <chr>     <chr>
##  1 Country                         nominal ID        original
##  2 Year                            numeric ID        original
##  3 Alcohol                         numeric predictor original
##  4 percentage_expenditure          numeric predictor original
##  5 Hepatitis_B                     numeric predictor original
##  6 Measles                         numeric predictor original
##  7 BMI                             numeric predictor original
##  8 Polio                           numeric predictor original
```

```
##  9 Total_expenditure               numeric predictor original
## 10 Diphtheria                      numeric predictor original
## 11 HIV_AIDS                        numeric predictor original
## 12 GDP                             numeric predictor original
## 13 thinness__1_19_years            numeric predictor original
## 14 thinness_5_9_years              numeric predictor original
## 15 Income_composition_of_resources numeric predictor original
## 16 Schooling                       numeric predictor original
## 17 Developed                       numeric predictor original
## 18 Life_expectancy                 numeric outcome   original
```

Feature Engineering and Preprocessing

Many machine learning techniques (random forests, LASSO, etc.) require your outcome and predictors to be in a specific format. Some techniques do not play well with missing values or non-normal and will require you to make changes to your variables. Oftentimes you will also have to change a variable to a factor or create dummy variables out of categorical variables. There are many things that need to be done to your data and today we are going to go over just a few. Things like imputation will not be covered and dimension reduction will be covered on Thursday.

Let's first take a look at our Life Expectancy data and how we can create a recipe in order to get it ready for use with different modeling techniques.

```
#skim(train_data)
names(train_data)
```

```
##  [1] "Country"                         "Year"
##  [3] "Life_expectancy"                 "Alcohol"
##  [5] "percentage_expenditure"          "Hepatitis_B"
##  [7] "Measles"                         "BMI"
##  [9] "Polio"                           "Total_expenditure"
## [11] "Diphtheria"                      "HIV_AIDS"
## [13] "GDP"                             "thinness__1_19_years"
## [15] "thinness_5_9_years"              "Income_composition_of_resources"
## [17] "Schooling"                       "Developed"
```

```
summary(le)
```

```
##         Country          Year      Life_expectancy    Alcohol
##  Afghanistan:  16   Min.   :2000   Min.   :44.00   Min.   :-4.6052
##  Albania    :  16   1st Qu.:2004   1st Qu.:65.30   1st Qu.:-0.4155
##  Armenia    :  15   Median :2008   Median :72.50   Median : 1.2726
##  Austria    :  15   Mean   :2008   Mean   :70.03   Mean   : 0.4147
##  Bahrain    :  15   3rd Qu.:2011   3rd Qu.:75.50   3rd Qu.: 1.9988
##  Belarus    :  15   Max.   :2015   Max.   :89.00   Max.   : 2.8831
##  (Other)    :1761
##  percentage_expenditure  Hepatitis_B       Measles           BMI
##  Min.   :    0.00       Min.   : 2.00   Min.   :     0   Min.   : 2.00
##  1st Qu.:   41.91       1st Qu.:77.00   1st Qu.:     0   1st Qu.:21.20
##  Median :  169.20       Median :92.00   Median :    13   Median :44.90
##  Mean   :  764.91       Mean   :80.81   Mean   :  1994   Mean   :39.13
##  3rd Qu.:  591.78       3rd Qu.:96.00   3rd Qu.:   292   3rd Qu.:56.40
##  Max.   :18961.35       Max.   :99.00   Max.   :131441   Max.   :77.10
```

```
##
##      Polio       Total_expenditure   Diphtheria      HIV_AIDS
##  Min.   : 3.00   Min.   : 0.740    Min.   : 2.0   Min.   : 0.100
##  1st Qu.:83.00   1st Qu.: 4.200    1st Qu.:83.0   1st Qu.: 0.100
##  Median :94.00   Median : 5.660    Median :93.0   Median : 0.100
##  Mean   :84.83   Mean   : 5.784    Mean   :85.2   Mean   : 1.778
##  3rd Qu.:97.00   3rd Qu.: 7.350    3rd Qu.:97.0   3rd Qu.: 0.500
##  Max.   :99.00   Max.   :14.390    Max.   :99.0   Max.   :50.600
##
##       GDP          thinness__1_19_years thinness_5_9_years
##  Min.   : 0.5196   Min.   : 0.100       Min.   : 0.100
##  1st Qu.: 6.2663   1st Qu.: 1.700       1st Qu.: 1.800
##  Median : 7.5694   Median : 3.400       Median : 3.400
##  Mean   : 7.4971   Mean   : 4.809       Mean   : 4.844
##  3rd Qu.: 8.6718   3rd Qu.: 6.800       3rd Qu.: 6.800
##  Max.   :11.6883   Max.   :27.200       Max.   :28.200
##
##  Income_composition_of_resources   Schooling       Developed
##  Min.   :0.0000                   Min.   : 0.00   Min.   :0.0000
##  1st Qu.:0.5270                   1st Qu.:10.60   1st Qu.:0.0000
##  Median :0.6910                   Median :12.50   Median :0.0000
##  Mean   :0.6433                   Mean   :12.32   Mean   :0.1468
##  3rd Qu.:0.7760                   3rd Qu.:14.20   3rd Qu.:0.0000
##  Max.   :0.9360                   Max.   :20.70   Max.   :1.0000
##
```

One thing we are going to want to do is to normalize our numeric variables. Later on we are going to use something called LASSO and it prefers to work with normalized data, so here, we will use the `step_normalize` function.

It is also a good idea to get of variables that have zero variance. If you have all females in your dataset or everyone right around the same age, there is no reason to keep those variables in the dataset. The `step_zv` function takes care of those variables.

```
le_rec <-
  recipe(Life_expectancy ~., data = train_data) %>%
  update_role(Country, Year, new_role = "ID")  %>%
  step_normalize(all_numeric_predictors(), -all_outcomes()) %>%
  step_zv(all_predictors())

summary(le_rec)
```

```
## # A tibble: 18 x 4
##    variable                        type    role      source
##    <chr>                           <chr>   <chr>     <chr>
##  1 Country                         nominal ID        original
##  2 Year                            numeric ID        original
##  3 Alcohol                         numeric predictor original
##  4 percentage_expenditure          numeric predictor original
##  5 Hepatitis_B                     numeric predictor original
##  6 Measles                         numeric predictor original
##  7 BMI                             numeric predictor original
##  8 Polio                           numeric predictor original
##  9 Total_expenditure               numeric predictor original
```

```
## 10 Diphtheria                      numeric predictor original
## 11 HIV_AIDS                        numeric predictor original
## 12 GDP                             numeric predictor original
## 13 thinness__1_19_years            numeric predictor original
## 14 thinness_5_9_years              numeric predictor original
## 15 Income_composition_of_resources numeric predictor original
## 16 Schooling                       numeric predictor original
## 17 Developed                       numeric predictor original
## 18 Life_expectancy                 numeric outcome   original
```

We are going to do the same thing with our `hf` dataset, but will be creating dummy variables for any of our categorical variables in the dataset. It doesn't really need it for this dataset, however, I wanted to make sure I showed what it looked like.

```r
#skim(hf)
hf_rec <- recipe(DEATH_EVENT ~., data = hf_train_data) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_zv(all_predictors())

summary(hf_rec)
```

```
## # A tibble: 13 x 4
##    variable                 type    role      source
##    <chr>                    <chr>   <chr>     <chr>
##  1 age                      numeric predictor original
##  2 anaemia                  nominal predictor original
##  3 creatinine_phosphokinase numeric predictor original
##  4 diabetes                 nominal predictor original
##  5 ejection_fraction        numeric predictor original
##  6 high_blood_pressure      nominal predictor original
##  7 platelets                numeric predictor original
##  8 serum_creatinine         numeric predictor original
##  9 serum_sodium             numeric predictor original
## 10 sex                      nominal predictor original
## 11 smoking                  nominal predictor original
## 12 time                     numeric predictor original
## 13 DEATH_EVENT              nominal outcome   original
```

As I mentioned before, you would probably do a lot more work to clean up and get your data ready. It would involve many steps that we looked at before, including extensive data visualization and data exploration using ggplot and dplyr.

Regression vs. Classification

In machine learning, many of the problems can be broken down into two categories: Regression or Classification. Regression problems are generally looking to predict a continuous outcome, while a classification problem tends to try and predict a specific discrete outcome. Confusingly enough, a logistic regression problem can be seen as a form of a classification problem.

Linear Regression Model

To get us use to using the tidymodel syntax, we are going to fit a normal linear regression model using our recipe from earlier.

The first step is to say which type of model you want to run and pull up the model engine. This is similar to using the `carat` package and may require you to install a package for specific model types.

```
linear_mod <-
  linear_reg() %>%
  set_engine("lm")
```

We then add our engine to our recipe within a workflow. This just helps to tie together our data processing and our modeling into one step.

```
le_wflow <-
  workflow() %>%
  add_model(linear_mod) %>%
  add_recipe(le_rec)

le_wflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_normalize()
## * step_zv()
##
## -- Model -----------------------------------------------------------------------
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

We then run the `fit` function on our training data to fit a model to our training set. This model will look just like when we did it last week.

```
le_fit <- le_wflow %>% fit(train_data)
le_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: linear_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_normalize()
## * step_zv()
```

```
##
## -- Model -----------------------------------------------------------------------
##
## Call:
## stats::lm(formula = ..y ~ ., data = data)
##
## Coefficients:
##                  (Intercept)                         Alcohol
##                     69.89086                         0.26362
##         percentage_expenditure                    Hepatitis_B
##                      0.44474                        -0.02425
##                      Measles                             BMI
##                      0.03171                         0.83285
##                        Polio               Total_expenditure
##                      0.29673                         0.19964
##                   Diphtheria                        HIV_AIDS
##                      0.48362                        -3.73759
##                          GDP            thinness__1_19_years
##                      0.84222                         0.06348
##            thinness_5_9_years  Income_composition_of_resources
##                     -0.40645                         1.80525
##                    Schooling                       Developed
##                      2.34675                         0.30839
```

```
le_fit %>%
  pull_workflow_fit() %>%
  tidy()
```

```
## # A tibble: 16 x 5
##    term                            estimate std.error statistic   p.value
##    <chr>                              <dbl>     <dbl>     <dbl>     <dbl>
##  1 (Intercept)                      69.9        0.109   639.    0.
##  2 Alcohol                           0.264      0.126     2.09  3.72e-  2
##  3 percentage_expenditure            0.445      0.146     3.04  2.43e-  3
##  4 Hepatitis_B                      -0.0243     0.141    -0.172 8.64e-  1
##  5 Measles                           0.0317     0.113     0.279 7.80e-  1
##  6 BMI                               0.833      0.142     5.87  5.39e-  9
##  7 Polio                             0.297      0.144     2.07  3.90e-  2
##  8 Total_expenditure                 0.200      0.116     1.72  8.64e-  2
##  9 Diphtheria                        0.484      0.160     3.03  2.50e-  3
## 10 HIV_AIDS                         -3.74       0.117   -32.0   1.21e-168
## 11 GDP                               0.842      0.158     5.32  1.21e-  7
## 12 thinness__1_19_years              0.0635     0.272     0.233 8.16e-  1
## 13 thinness_5_9_years               -0.406      0.271    -1.50  1.34e-  1
## 14 Income_composition_of_resources   1.81       0.175    10.3   4.42e- 24
## 15 Schooling                         2.35       0.194    12.1   5.02e- 32
## 16 Developed                         0.308      0.137     2.26  2.43e-  2
```

The final portion is to compare our predicted outcomes to our actual outcomes within the training and testing dataset. We used the data_grid functions before to do this, but here I will show another way to do this that will be extendable to other machine learning models.

```r
predict(le_fit, train_data)
```

```
## # A tibble: 1,390 x 1
##    .pred
##    <dbl>
##  1  63.1
##  2  63.6
##  3  63.6
##  4  62.8
##  5  62.2
##  6  61.2
##  7  59.3
##  8  57.4
##  9  58.1
## 10  58.9
## # ... with 1,380 more rows
```

```r
# Adding the actual train data results
le_pred <-
  predict(le_fit, train_data) %>%
  bind_cols(train_data %>% select(Life_expectancy))

le_pred
```

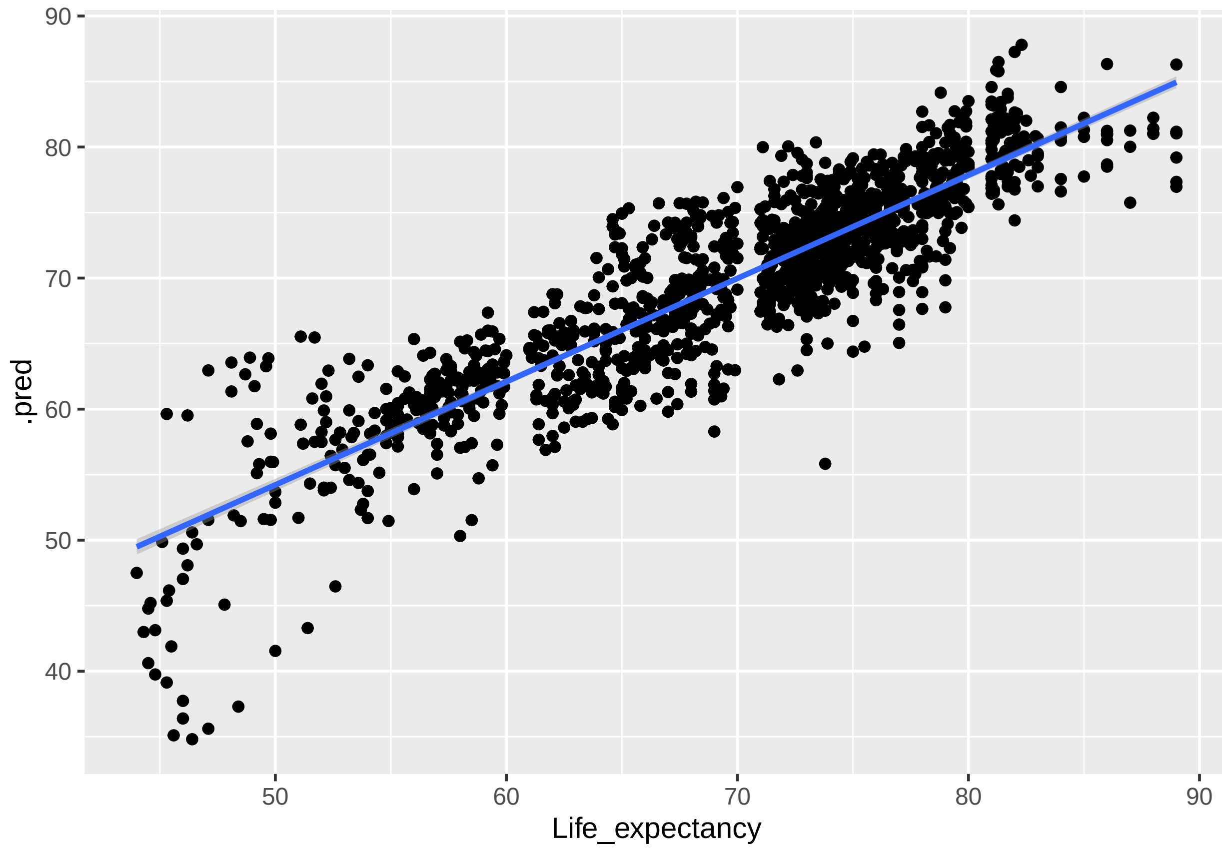
```
## # A tibble: 1,390 x 2
##    .pred Life_expectancy
##    <dbl>           <dbl>
##  1  63.1            65
##  2  63.6            59.9
##  3  63.6            59.9
##  4  62.8            58.8
##  5  62.2            58.6
##  6  61.2            57.5
##  7  59.3            57.3
##  8  57.4            57
##  9  58.1            56.7
## 10  58.9            56.2
## # ... with 1,380 more rows
```

```r
glance(lm(.pred ~ Life_expectancy, data = le_pred))
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.788         0.788  3.60     5155.       0     1 -3751. 7509. 7524.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```r
le_pred %>%
ggplot(aes(Life_expectancy, .pred)) + geom_point() + geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
predict(le_fit, test_data)
```

```
## # A tibble: 463 x 1
##    .pred
##    <dbl>
##  1  63.6
##  2  62.1
##  3  61.9
##  4  60.7
##  5  75.6
##  6  71.4
##  7  70.5
##  8  73.5
##  9  73.7
## 10  69.0
## # ... with 453 more rows
```

```
# Adding the actual test data results
le_pred <-
  predict(le_fit, test_data) %>%
  bind_cols(test_data %>% select(Life_expectancy))

le_pred
```

```
## # A tibble: 463 x 2
```

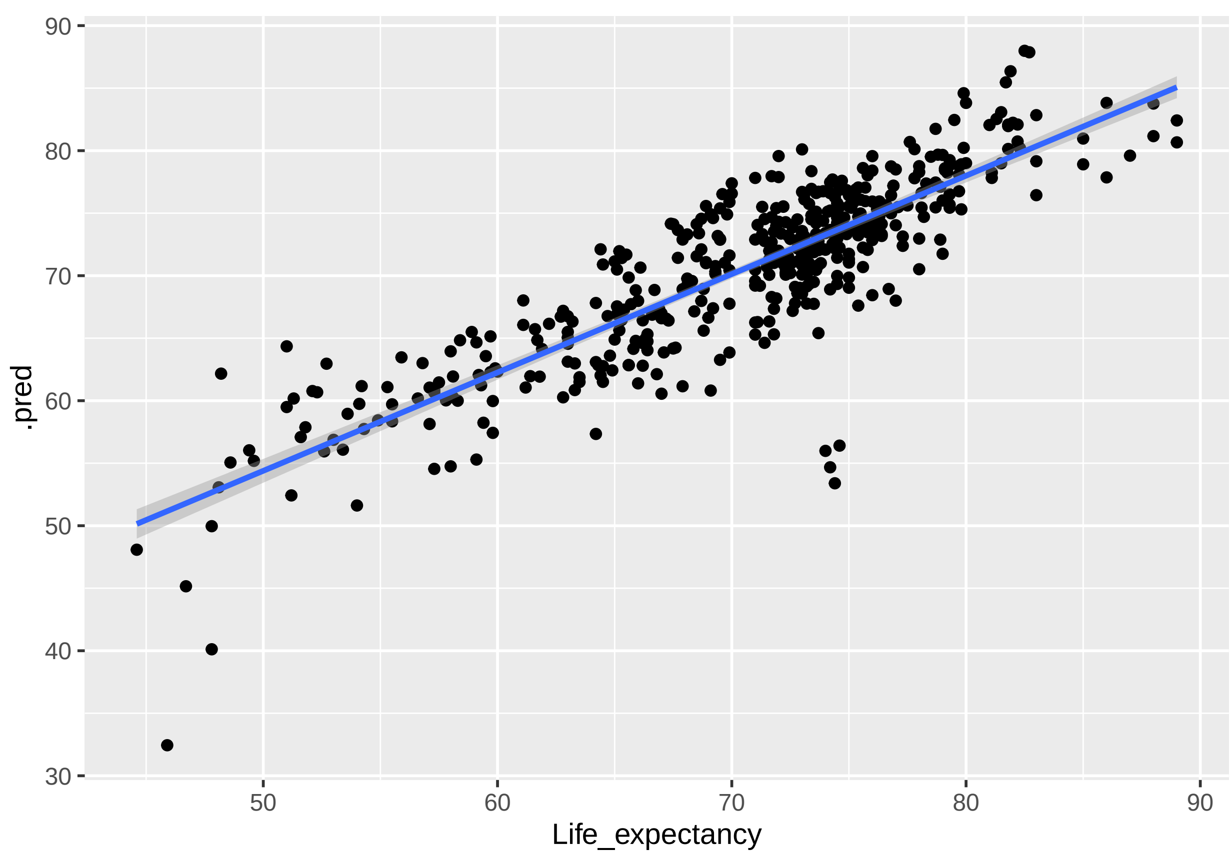
```
##     .pred Life_expectancy
##     <dbl>           <dbl>
##  1  63.6            59.5
##  2  62.1            59.2
##  3  61.9            58.1
##  4  60.7            57.3
##  5  75.6            77.5
##  6  71.4            73
##  7  70.5            73.6
##  8  73.5            75.4
##  9  73.7            74.7
## 10  69.0            72.9
## # ... with 453 more rows
```

```
glance(lm(.pred ~ Life_expectancy, data = le_pred))
```

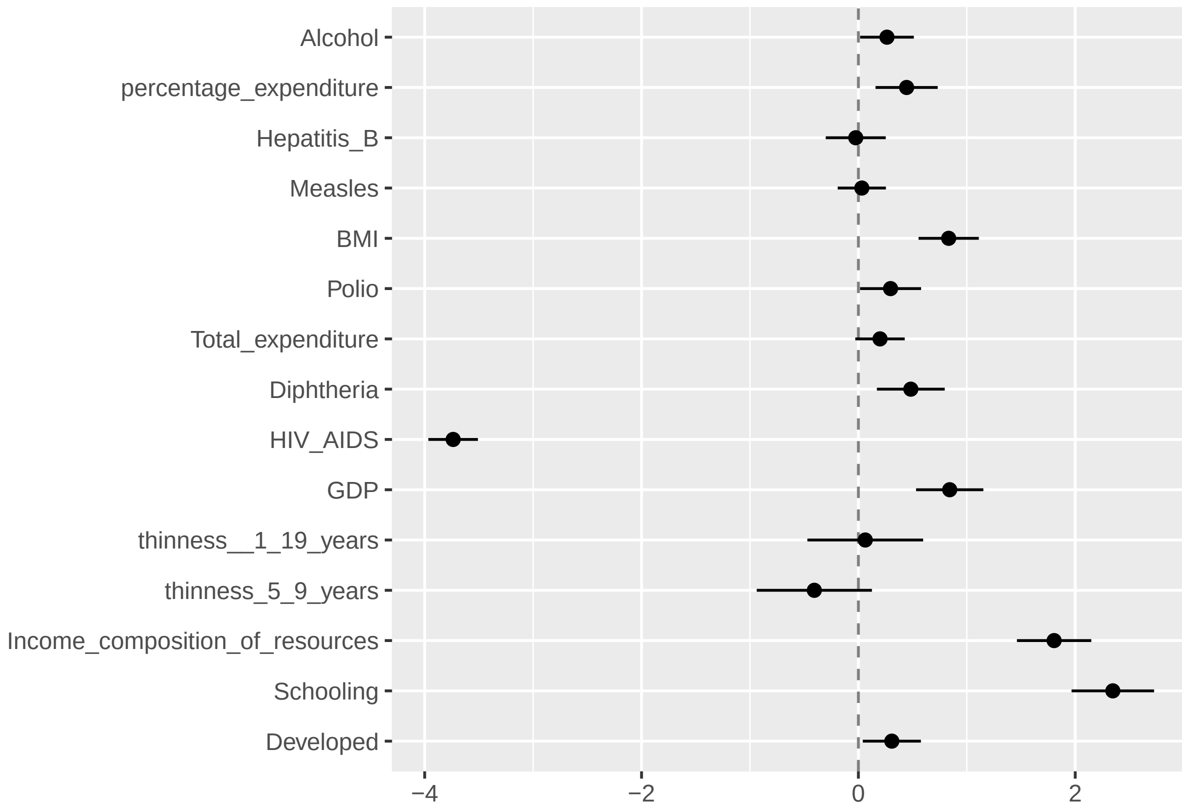
```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic   p.value    df  logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>     <dbl> <dbl>   <dbl> <dbl> <dbl>
## 1     0.734         0.733  3.80     1271. 1.28e-134     1 -1274. 2555. 2567.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
le_pred %>%
ggplot(aes(Life_expectancy, .pred)) + geom_point() + geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
tidy(le_fit) %>%
  dwplot(dot_args = list(size = 2, color = "black"),
         whisker_args = list(color = "black"),
         vline = geom_vline(xintercept = 0, colour = "grey50", linetype = 2))
```



LASSO and RIDGE REGRESSION

Lasso and Ridge Regression are extensions to linear regression. They are considered regularized models and they tend to penalize coefficients that aren't important to a model and to penalize models that have too many predictors. They are used to help combat overfitting a model, which can cause a true lack of flexibility and generalizability. Models are only taking in a certain amount of data and could be trained on a dataset that has more noise than you would like. Having regularization allows a model to be able to handle some noise without overfitting the model.

"Overfit can happen in linear models as well when dealing with multiple features. If not filtered and explored up front, some features can be more destructive than helpful, repeat information that already expressed by other features and add high noise to the dataset."

The difference between ridge regression and the Lasso is that the Lasso can have a coefficient drop all the way to 0, while Ridge Regression does not. These both work to minimize the impact of coefficients as much as possible, while still retaining some impact of the predictors.

The first thing we need to do is set how much regularization we want, which is handled using the penalty and mixture arguments in the `linear_reg` function. Penalty shows the total amount of regularization you want and mixture goes from 0 to 1 with 0 being a ridge regression model and 1 being a lasso model. We then set the engine, which will come from the `glmnet` package.

```
?linear_reg
```

```
## starting httpd help server ... done
```

```
lasso_spec <- linear_reg(penalty = 0.1, mixture = 1) %>%
  set_engine("glmnet")
```

From there, we work the same magic we did before by adding the model to a workflow along with a recipe.

```
wf <- workflow() %>%
  add_recipe(le_rec)

lasso_fit <- wf %>%
  add_model(lasso_spec) %>%
  fit(data = train_data)

lasso_fit %>%
  pull_workflow_fit() %>%
  tidy()
```

```
## # A tibble: 16 x 3
##    term                            estimate penalty
##    <chr>                              <dbl>   <dbl>
##  1 (Intercept)                       69.9       0.1
##  2 Alcohol                            0.205     0.1
##  3 percentage_expenditure             0.408     0.1
##  4 Hepatitis_B                        0         0.1
##  5 Measles                            0         0.1
##  6 BMI                                0.789     0.1
##  7 Polio                              0.248     0.1
##  8 Total_expenditure                  0.120     0.1
##  9 Diphtheria                         0.431     0.1
## 10 HIV_AIDS                          -3.65      0.1
## 11 GDP                                0.813     0.1
## 12 thinness__1_19_years               0         0.1
## 13 thinness_5_9_years                -0.334     0.1
## 14 Income_composition_of_resources    1.79      0.1
## 15 Schooling                          2.41      0.1
## 16 Developed                          0.282     0.1
```

```
# Notice the number of 0 coefficients.
lasso_pred <-
  predict(lasso_fit, test_data) %>%
  bind_cols(test_data %>% select(Life_expectancy))

lasso_pred
```

```
## # A tibble: 463 x 2
```

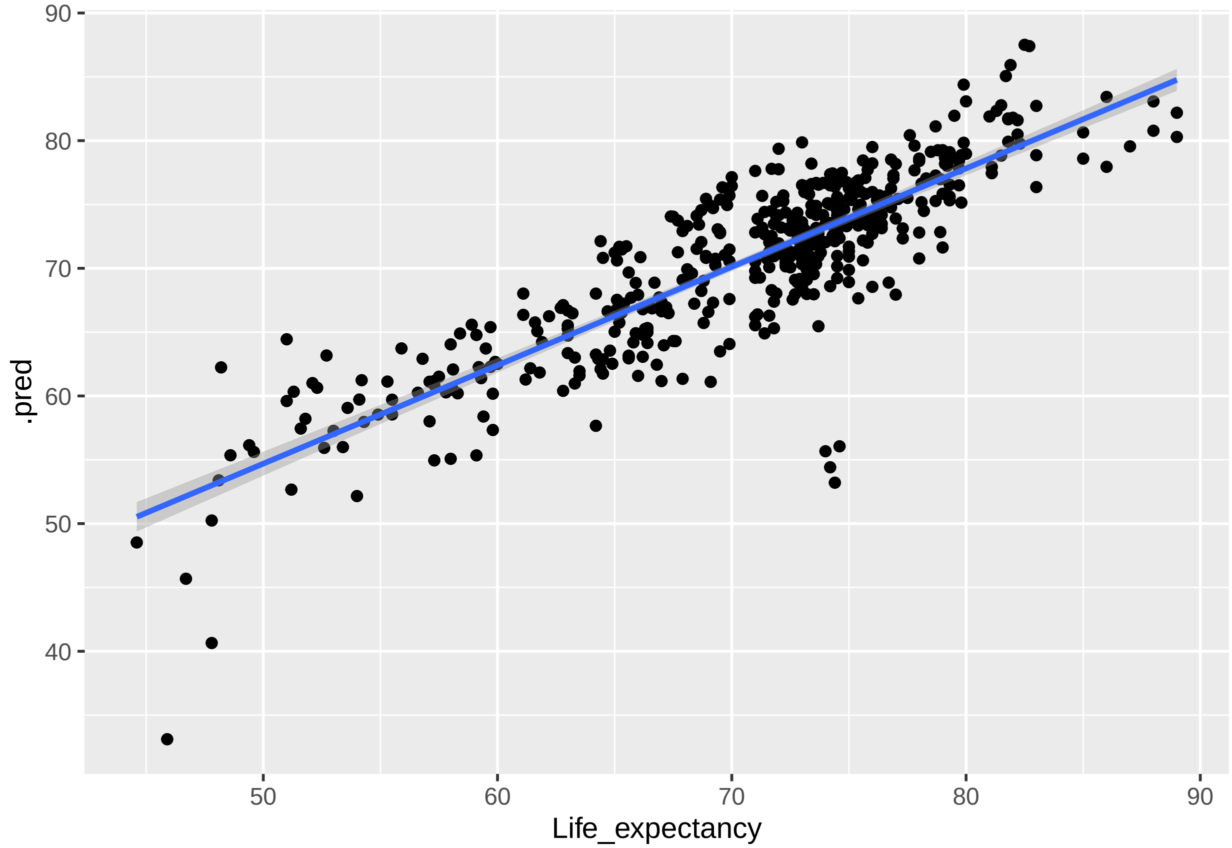
```
##    .pred Life_expectancy
##    <dbl>           <dbl>
##  1  63.7            59.5
##  2  62.3            59.2
##  3  62.1            58.1
##  4  60.9            57.3
##  5  75.5            77.5
##  6  71.2            73
##  7  70.3            73.6
##  8  73.5            75.4
##  9  73.7            74.7
## 10  69.2            72.9
## # ... with 453 more rows
```

```
glance(lm(.pred ~ Life_expectancy, data = lasso_pred))
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic   p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>     <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.732         0.731  3.75     1258. 7.31e-134     1 -1267. 2541. 2553.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
# Rather similar to the linear regression model.
lasso_pred %>%
ggplot(aes(Life_expectancy, .pred)) + geom_point() + geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



RESAMPLING WITH BOOTSTRAP

We chose our regularization penalty at random, but we could try out several different penalty parameters many times. To do this, we will use a bootstrap method. Bootstrapping allows you to take a small sample of data over and over again to test out several different parameters.

From the `bootstraps` documentation: "A bootstrap sample is a sample that is the same size as the original data set that is made using replacement. This results in analysis samples that have multiple replicates of some of the original rows of the data. The assessment set is defined as the rows of the original data that were not included in the bootstrap sample. This is often referred to as the"out-of-bag" (OOB) sample."

```
#?bootstraps
lasso_boot <- bootstraps(train_data)
```

The `tune`, `grid_regular` and `penalty` functions allows you to try several different penalty levels across samples.

```
tune_spec <- linear_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet")

lambda_grid <- grid_regular(penalty(), levels = 50)

#We can reuse the workflow from before.
lasso_grid <- tune_grid(
  wf %>% add_model(tune_spec),
```

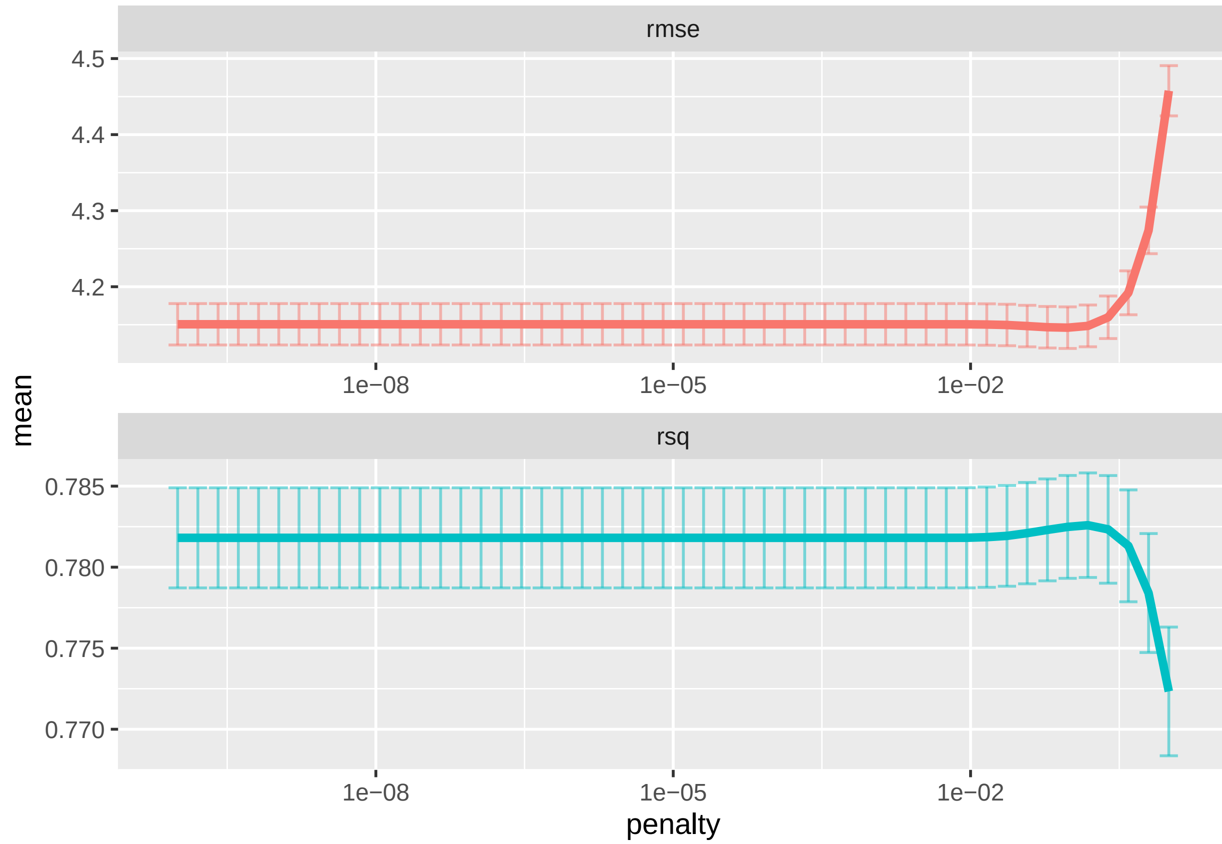
```
  resamples = lasso_boot,
  grid = lambda_grid
)

lasso_grid %>%
  collect_metrics()
```

```
## # A tibble: 100 x 7
##     penalty .metric .estimator  mean     n std_err .config
##       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1 1.00e-10 rmse    standard   4.15     25 0.0272  Preprocessor1_Model01
##  2 1.00e-10 rsq     standard   0.782    25 0.00309 Preprocessor1_Model01
##  3 1.60e-10 rmse    standard   4.15     25 0.0272  Preprocessor1_Model02
##  4 1.60e-10 rsq     standard   0.782    25 0.00309 Preprocessor1_Model02
##  5 2.56e-10 rmse    standard   4.15     25 0.0272  Preprocessor1_Model03
##  6 2.56e-10 rsq     standard   0.782    25 0.00309 Preprocessor1_Model03
##  7 4.09e-10 rmse    standard   4.15     25 0.0272  Preprocessor1_Model04
##  8 4.09e-10 rsq     standard   0.782    25 0.00309 Preprocessor1_Model04
##  9 6.55e-10 rmse    standard   4.15     25 0.0272  Preprocessor1_Model05
## 10 6.55e-10 rsq     standard   0.782    25 0.00309 Preprocessor1_Model05
## # ... with 90 more rows
```

Because the `Income_composition_of_resources` has such a large hold over the model, the models do not change very much between penalty levels. The following steps allow you to analyze the best penalty parameter possible.

```
lasso_grid %>%
  collect_metrics() %>%
  ggplot(aes(penalty, mean, color = .metric)) +
  geom_errorbar(aes(
    ymin = mean - std_err,
    ymax = mean + std_err
  ),
  alpha = 0.5
  ) +
  geom_line(size = 1.5) +
  facet_wrap(~.metric, scales = "free", nrow = 2) +
  scale_x_log10() +
  theme(legend.position = "none")
```

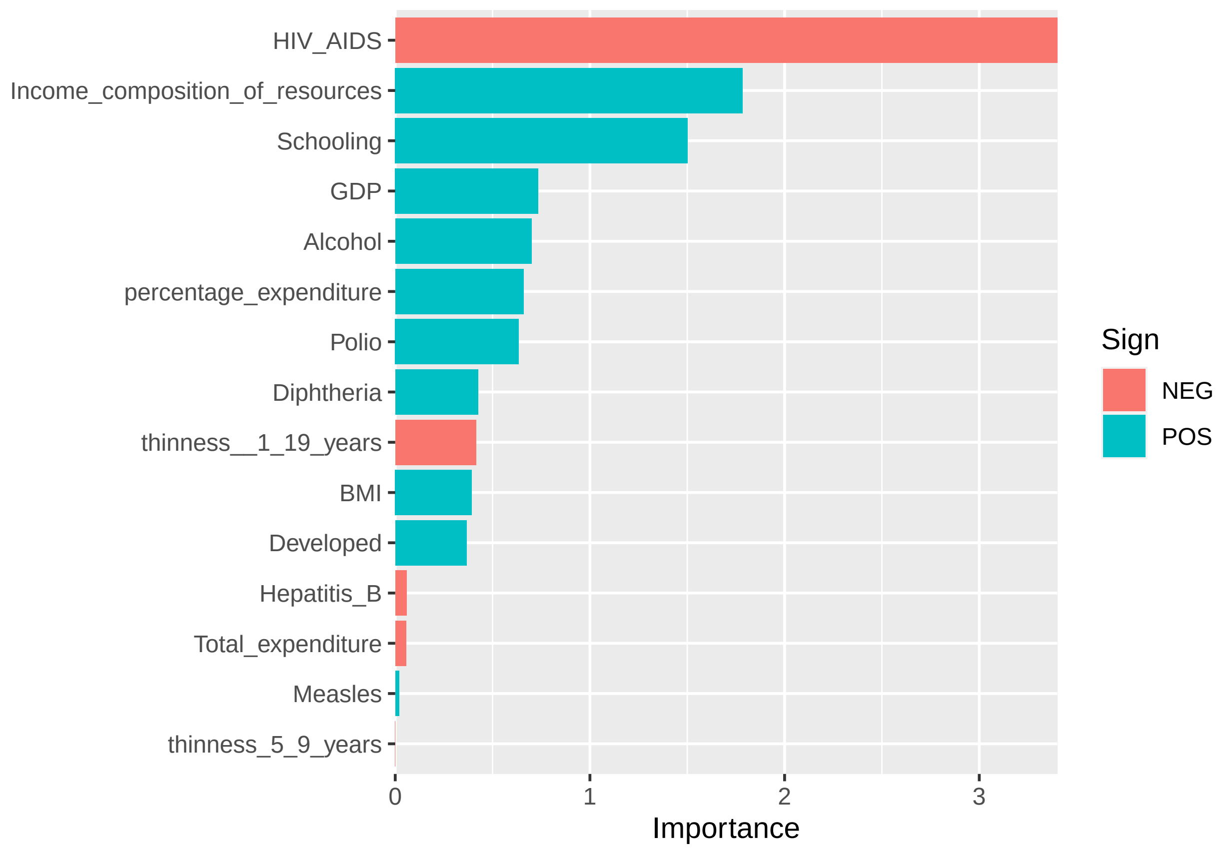



```
lowest_rmse <- lasso_grid %>%
  select_best("rmse")
# Preprocessor1_Model44 -> .0596 penalty
```

You then use that lowest value to fit that model to a test set to check and see how well your model fits.

```
final_lasso <- finalize_workflow(
  wf %>% add_model(tune_spec),
  lowest_rmse
)

final_lasso %>%
  fit(test_data) %>%
  pull_workflow_fit() %>%
  vi(lambda = lowest_rmse$penalty) %>%
  mutate(
    Importance = abs(Importance),
    Variable = fct_reorder(Variable, Importance)
  ) %>%
  ggplot(aes(x = Importance, y = Variable, fill = Sign)) +
  geom_col() +
  scale_x_continuous(expand = c(0, 0)) +
  labs(y = NULL)
```




```
last_fit(
  final_lasso,
  le_split
) %>%
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       4.16  Preprocessor1_Model1
## 2 rsq     standard       0.732 Preprocessor1_Model1
```

```
#Again, this stays pretty close to what we had before.
```

CLASSIFICATION

We are now going to try a classification problem, checking to see if we can predict Death from Heart failure.

The first step is to fit a logistic regression model, just as we did with the linear regression model. Here we use the `logistic_reg` function and the `glm` engine.

```
lr_mod <-
  logistic_reg() %>%
  set_engine("glm")
```

```r
hf_wflow <-
  workflow() %>%
  add_model(lr_mod) %>%
  add_recipe(hf_rec)

hf_wflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: logistic_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_zv()
##
## -- Model -----------------------------------------------------------------------
## Logistic Regression Model Specification (classification)
##
## Computational engine: glm
```

```r
hf_fit <- hf_wflow %>% fit(hf_train_data)
hf_fit %>%
  pull_workflow_fit() %>%
  tidy()
```

```
## # A tibble: 13 x 5
##    term                        estimate  std.error statistic       p.value
##    <chr>                          <dbl>      <dbl>     <dbl>         <dbl>
##  1 (Intercept)               6.30       6.13          1.03   0.304
##  2 age                       0.0455     0.0181        2.52   0.0117
##  3 creatinine_phosphokinase  0.0000784  0.000192      0.408  0.684
##  4 ejection_fraction        -0.0788     0.0192       -4.11   0.0000404
##  5 platelets                -0.00000189 0.00000213   -0.884  0.377
##  6 serum_creatinine          0.668      0.195         3.43   0.000611
##  7 serum_sodium             -0.0353     0.0438       -0.805  0.421
##  8 time                     -0.0203     0.00337      -6.01   0.00000000185
##  9 anaemia_X1               -0.0315     0.415        -0.0759 0.939
## 10 diabetes_X1               0.0279     0.402         0.0694 0.945
## 11 high_blood_pressure_X1   -0.371      0.428        -0.868  0.386
## 12 sex_X1                   -0.523      0.461        -1.13   0.257
## 13 smoking_X1                0.0739     0.470         0.157  0.875
```

```r
#Notice we get 1s and 0s instead of continuous measures
predict(hf_fit, hf_train_data)
```

```
## # A tibble: 225 x 1
##    .pred_class
##    <fct>
##  1 1
```

```
##   2 1
##   3 1
##   4 1
##   5 1
##   6 1
##   7 0
##   8 1
##   9 1
## 10 0
## # ... with 215 more rows
```

```
comb <- predict(hf_fit, hf_train_data) %>%
 bind_cols(hf_train_data %>% select(DEATH_EVENT))
xtabs(~DEATH_EVENT + `.pred_class`, data = comb)
```

```
##              .pred_class
## DEATH_EVENT   0   1
##             0 138  15
##             1  22  50
```

```
# proportion incorrect = (18+12)/(141+12+18+54)
# 13.3 % incorrect
(18+12)/(141+12+18+54)
```

```
## [1] 0.1333333
```

```
predict(hf_fit, hf_test_data)
```

```
## # A tibble: 74 x 1
##    .pred_class
##    <fct>
##   1 1
##   2 0
##   3 1
##   4 1
##   5 1
##   6 1
##   7 1
##   8 1
##   9 0
## 10 1
## # ... with 64 more rows
```

```
comb <- predict(hf_fit, hf_test_data) %>%
 bind_cols(hf_test_data %>% select(DEATH_EVENT))
xtabs(~DEATH_EVENT + `.pred_class`, data = comb)
```

```
##              .pred_class
## DEATH_EVENT  0  1
##             0 49  1
##             1 10 14
```

```r
# proportion incorrect = (11+8)/(42+8+11+13)
# 25.7 % incorrect, Not great
1 - (11+8)/(42+8+11+13)
```

```
## [1] 0.7432432
```

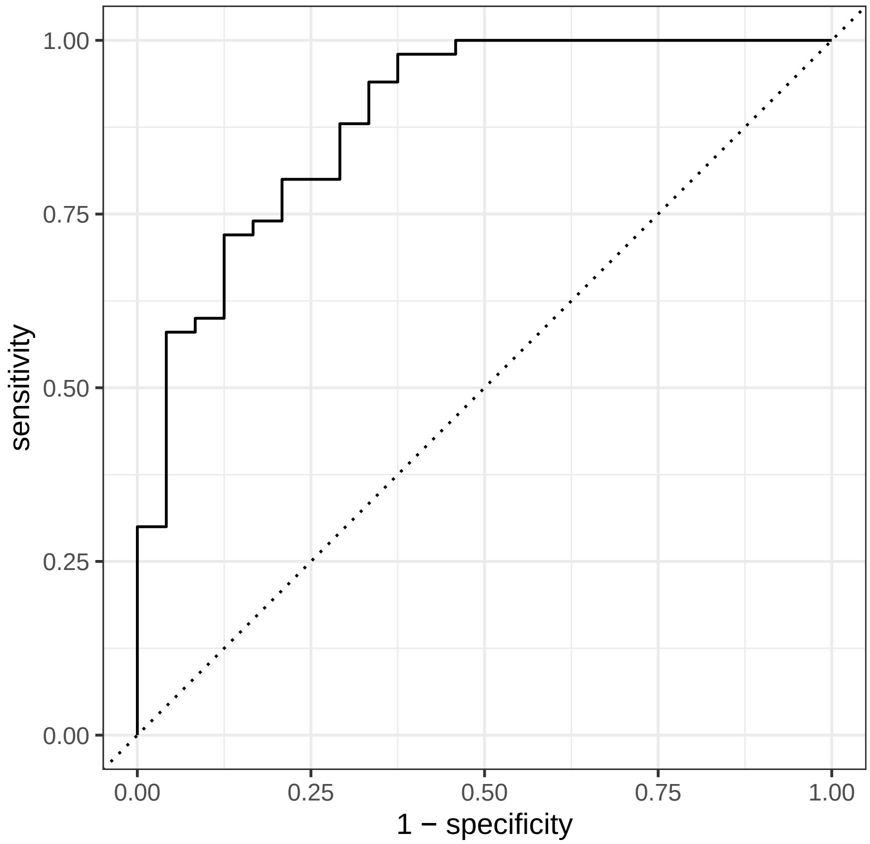
```r
# You could also look at the probability given for each prediction
hf_pred <-
  predict(hf_fit, hf_test_data, type = "prob") %>%
  bind_cols(hf_test_data %>% select(DEATH_EVENT))

hf_pred %>% print(n = Inf)
```

```
## # A tibble: 74 x 3
##    .pred_0 .pred_1 DEATH_EVENT
##      <dbl>   <dbl> <fct>
##  1  0.0694 0.931   1
##  2  0.744  0.256   1
##  3  0.0462 0.954   1
##  4  0.209  0.791   1
##  5  0.109  0.891   1
##  6  0.121  0.879   1
##  7  0.117  0.883   1
##  8  0.0984 0.902   1
##  9  0.724  0.276   1
## 10  0.239  0.761   1
## 11  0.248  0.752   1
## 12  0.607  0.393   1
## 13  0.628  0.372   0
## 14  0.397  0.603   1
## 15  0.163  0.837   1
## 16  0.518  0.482   1
## 17  0.564  0.436   0
## 18  0.179  0.821   1
## 19  0.541  0.459   0
## 20  0.823  0.177   0
## 21  0.899  0.101   0
## 22  0.696  0.304   0
## 23  0.759  0.241   0
## 24  0.586  0.414   0
## 25  0.452  0.548   0
## 26  0.728  0.272   0
## 27  0.810  0.190   0
## 28  0.778  0.222   1
## 29  0.649  0.351   0
## 30  0.544  0.456   0
## 31  0.673  0.327   0
## 32  0.880  0.120   0
## 33  0.229  0.771   1
## 34  0.900  0.100   0
## 35  0.829  0.171   0
## 36  0.457  0.543   1
## 37  0.547  0.453   1
```

```
## 38  0.734  0.266   0
## 39  0.733  0.267   0
## 40  0.832  0.168   0
## 41  0.850  0.150   1
## 42  0.926  0.0736  0
## 43  0.938  0.0625  0
## 44  0.717  0.283   1
## 45  0.968  0.0317  1
## 46  0.985  0.0154  0
## 47  0.794  0.206   0
## 48  0.600  0.400   0
## 49  0.976  0.0239  0
## 50  0.841  0.159   0
## 51  0.945  0.0551  0
## 52  0.906  0.0936  0
## 53  0.861  0.139   0
## 54  0.938  0.0623  0
## 55  0.982  0.0175  0
## 56  0.924  0.0764  0
## 57  0.970  0.0301  0
## 58  0.995  0.00533 0
## 59  0.878  0.122   1
## 60  0.988  0.0118  0
## 61  0.988  0.0117  0
## 62  0.977  0.0226  0
## 63  0.981  0.0190  0
## 64  0.947  0.0526  0
## 65  0.960  0.0399  0
## 66  0.961  0.0393  0
## 67  0.886  0.114   0
## 68  0.994  0.00571 0
## 69  0.989  0.0107  0
## 70  0.967  0.0335  0
## 71  0.975  0.0247  0
## 72  0.993  0.00700 0
## 73  0.991  0.00871 0
## 74  0.992  0.00777 0
```

```
hf_pred %>%
  roc_curve(truth = DEATH_EVENT, .pred_0) %>%
  autoplot()
```




```
hf_pred %>%
  roc_auc(truth = DEATH_EVENT, .pred_0)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.888
```

```
hf_pred %>%
  roc_auc(truth = DEATH_EVENT, .pred_1)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.112
```

Random Forest Models

Random Forest Models are a very popular technique in both classification and regression. Essentially, random forest models are a series of decision trees that are bagged together to create a large amount of trees that are not correlated with each other.

```r
rf_mod <-
  rand_forest(mtry = 3, trees = 1000) %>%
  set_engine("ranger") %>%
  set_mode("classification")

rf_wflow <-
  workflow() %>%
  add_model(rf_mod) %>%
  add_recipe(hf_rec)

rf_wflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_zv()
##
## -- Model -----------------------------------------------------------------------
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = 3
##   trees = 1000
##
## Computational engine: ranger
```

```r
rf_fit <- rf_wflow %>% fit(hf_train_data)

comb <- predict(rf_fit, hf_test_data) %>%
 bind_cols(hf_test_data %>% select(DEATH_EVENT))
xtabs(~DEATH_EVENT + `.pred_class`, data = comb)
```

```
##            .pred_class
## DEATH_EVENT  0  1
##           0 46  4
##           1  7 17
```

```r
rf_training_pred <-
  predict(rf_fit, hf_train_data) %>%
  bind_cols(predict(rf_fit, hf_train_data, type = "prob")) %>%
  # Add the true outcome data back in
  bind_cols(hf_train_data %>%
              select(DEATH_EVENT))

rf_training_pred %>%  # training set predictions
  roc_auc(truth = DEATH_EVENT, `.pred_1` )
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary       0.00309
```

```
rf_training_pred %>%  # training set predictions
  accuracy(truth = DEATH_EVENT, `.pred_class`)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.956
```

```
predict(rf_fit, hf_test_data) %>%
bind_cols(predict(rf_fit, hf_test_data)) %>%
# Add the true outcome data back in
bind_cols(hf_test_data %>%
          select(DEATH_EVENT)) %>% group_by(DEATH_EVENT) %>%
  count(`.pred_class...2`)
```

```
## New names:
## * .pred_class -> .pred_class...1
## * .pred_class -> .pred_class...2

## # A tibble: 4 x 3
## # Groups:   DEATH_EVENT [2]
##   DEATH_EVENT .pred_class...2     n
##   <fct>       <fct>           <int>
## 1 0           0                  46
## 2 0           1                   4
## 3 1           0                   7
## 4 1           1                  17
```

```
rf_testing_pred <-
  predict(rf_fit, hf_test_data) %>%
  bind_cols(predict(rf_fit, hf_test_data, type = "prob")) %>%
  bind_cols(hf_test_data %>% select(DEATH_EVENT))

rf_testing_pred %>%   # test set predictions
  roc_auc(truth = DEATH_EVENT, `.pred_0`)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.944
```

```
xtabs(~DEATH_EVENT + .pred_class, data = rf_testing_pred)
```

```
##            .pred_class
## DEATH_EVENT  0  1
##           0 46  4
##           1  7 17
```

```r
1-(4+9)/(4+9+46+15)
```

```
## [1] 0.8243243
```

```r
rf_testing_pred %>% # test set predictions
  accuracy(truth = DEATH_EVENT, `.pred_class`)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.851
```

Additional Resources

- Tidy Models Tutorial. This tutorial provided the framework for this lesson.