Assignment 2

Kevin Linares

2025-02-08

Setup

```
pacman::p_load(glmnet, caret, tidymodels, tidyverse)
options(scipen=999)
```

Data

For this exercise we use the Communities and Crime data from the UCI ML repository, which includes information about communities in the US. "The data combines socio-economic data from the 1990 US Census, law enforcement data from the 1990 US LEMAS survey, and crime data from the 1995 FBI UCR"

Source: https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime

First, some data prep.

- Read column names into a character vector and assign it as column names to the data frame.
- Remove columns with missing values.

```
file_path <- "~/repos/UMD_classes_code/ML_social_science_SURV613/assignments/"

varnames <- read_delim(str_c(file_path, "communities.txt"),
                       col_names = FALSE) |>
  pull(2)

crime <- read_csv( str_c(file_path, "communities.data"),
                         na = "?", col_names = varnames) |>
  select_if(~ !any(is.na(.))) |>
```

```
  # remove features
  select(-state, -communityname, -fold)
```

Check whats left.

```
skimr::skim(crime)
```

Table 1: Data summary

Name	crime
Number of rows	1994
Number of columns	100
Column type frequency:	
numeric	100
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
population	0	1	0.06	0.13	0	0.01	0.02	0.05	1	
householdsize	0	1	0.46	0.16	0	0.35	0.44	0.54	1	
racepctblack	0	1	0.18	0.25	0	0.02	0.06	0.23	1	
racePctWhite	0	1	0.75	0.24	0	0.63	0.85	0.94	1	
racePctAsian	0	1	0.15	0.21	0	0.04	0.07	0.17	1	
racePctHisp	0	1	0.14	0.23	0	0.01	0.04	0.16	1	
agePct12t21	0	1	0.42	0.16	0	0.34	0.40	0.47	1	
agePct12t29	0	1	0.49	0.14	0	0.41	0.48	0.54	1	
agePct16t24	0	1	0.34	0.17	0	0.25	0.29	0.36	1	
agePct65up	0	1	0.42	0.18	0	0.30	0.42	0.53	1	
numbUrban	0	1	0.06	0.13	0	0.00	0.03	0.07	1	
pctUrban	0	1	0.70	0.44	0	0.00	1.00	1.00	1	
medIncome	0	1	0.36	0.21	0	0.20	0.32	0.49	1	
pctWWage	0	1	0.56	0.18	0	0.44	0.56	0.69	1	
pctWFarmSelf	0	1	0.29	0.20	0	0.16	0.23	0.37	1	
pctWInvInc	0	1	0.50	0.18	0	0.37	0.48	0.62	1	
pctWSocSec	0	1	0.47	0.17	0	0.35	0.48	0.58	1	
pctWPubAsst	0	1	0.32	0.22	0	0.14	0.26	0.44	1	
pctWRetire	0	1	0.48	0.17	0	0.36	0.47	0.58	1	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
medFamInc	0	1	0.38	0.20	0	0.23	0.33	0.48	1	
perCapInc	0	1	0.35	0.19	0	0.22	0.30	0.43	1	
whitePerCap	0	1	0.37	0.19	0	0.24	0.32	0.44	1	
blackPerCap	0	1	0.29	0.17	0	0.17	0.25	0.38	1	
indianPerCap	0	1	0.20	0.16	0	0.11	0.17	0.25	1	
AsianPerCap	0	1	0.32	0.20	0	0.19	0.28	0.40	1	
HispPerCap	0	1	0.39	0.18	0	0.26	0.34	0.48	1	
NumUnderPov	0	1	0.06	0.13	0	0.01	0.02	0.05	1	
PctPopUnderPov	0	1	0.30	0.23	0	0.11	0.25	0.45	1	
PctLess9thGrade	0	1	0.32	0.21	0	0.16	0.27	0.42	1	
PctNotHSGrad	0	1	0.38	0.20	0	0.23	0.36	0.51	1	
PctBSorMore	0	1	0.36	0.21	0	0.21	0.31	0.46	1	
PctUnemployed	0	1	0.36	0.20	0	0.22	0.32	0.48	1	
PctEmploy	0	1	0.50	0.17	0	0.38	0.51	0.63	1	
PctEmplManu	0	1	0.40	0.20	0	0.25	0.37	0.52	1	
PctEmplProfServ	0	1	0.44	0.18	0	0.32	0.41	0.53	1	
PctOccupManu	0	1	0.39	0.20	0	0.24	0.37	0.51	1	
PctOccupMgmtProf	0	1	0.44	0.19	0	0.31	0.40	0.54	1	
MalePctDivorce	0	1	0.46	0.18	0	0.33	0.47	0.59	1	
MalePctNevMarr	0	1	0.43	0.18	0	0.31	0.40	0.50	1	
FemalePctDiv	0	1	0.49	0.18	0	0.36	0.50	0.62	1	
TotalPctDiv	0	1	0.49	0.18	0	0.36	0.50	0.63	1	
PersPerFam	0	1	0.49	0.15	0	0.40	0.47	0.56	1	
PctFam2Par	0	1	0.61	0.20	0	0.49	0.63	0.76	1	
PctKids2Par	0	1	0.62	0.21	0	0.49	0.64	0.78	1	
PctYoungKids2Par	0	1	0.66	0.22	0	0.53	0.70	0.84	1	
PctTeen2Par	0	1	0.58	0.19	0	0.48	0.61	0.72	1	
PctWorkMomYoungKids	0	1	0.50	0.17	0	0.39	0.51	0.62	1	
PctWorkMom	0	1	0.53	0.18	0	0.42	0.54	0.65	1	
NumIlleg	0	1	0.04	0.11	0	0.00	0.01	0.02	1	
PctIlleg	0	1	0.25	0.23	0	0.09	0.17	0.32	1	
NumImmig	0	1	0.03	0.09	0	0.00	0.01	0.02	1	
PctImmigRecent	0	1	0.32	0.22	0	0.16	0.29	0.43	1	
PctImmigRec5	0	1	0.36	0.21	0	0.20	0.34	0.48	1	
PctImmigRec8	0	1	0.40	0.20	0	0.25	0.39	0.53	1	
PctImmigRec10	0	1	0.43	0.19	0	0.28	0.43	0.56	1	
PctRecentImmig	0	1	0.18	0.24	0	0.03	0.09	0.23	1	
PctRecImmig5	0	1	0.18	0.24	0	0.03	0.08	0.23	1	
PctRecImmig8	0	1	0.18	0.24	0	0.03	0.09	0.23	1	
PctRecImmig10	0	1	0.18	0.23	0	0.03	0.09	0.23	1	
PctSpeakEnglOnly	0	1	0.79	0.23	0	0.73	0.87	0.94	1	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
PctNotSpeakEnglWell	0	1	0.15	0.22	0	0.03	0.06	0.16	1	
PctLargHouseFam	0	1	0.27	0.20	0	0.15	0.20	0.31	1	
PctLargHouseOccup	0	1	0.25	0.19	0	0.14	0.19	0.29	1	
PersPerOccupHous	0	1	0.46	0.17	0	0.34	0.44	0.55	1	
PersPerOwnOccHous	0	1	0.49	0.16	0	0.39	0.48	0.58	1	
PersPerRentOccHous	0	1	0.40	0.19	0	0.27	0.36	0.49	1	
PctPersOwnOccup	0	1	0.56	0.20	0	0.44	0.56	0.70	1	
PctPersDenseHous	0	1	0.19	0.21	0	0.06	0.11	0.22	1	
PctHousLess3BR	0	1	0.50	0.17	0	0.40	0.51	0.60	1	
MedNumBR	0	1	0.31	0.26	0	0.00	0.50	0.50	1	
HousVacant	0	1	0.08	0.15	0	0.01	0.03	0.07	1	
PctHousOccup	0	1	0.72	0.19	0	0.63	0.77	0.86	1	
PctHousOwnOcc	0	1	0.55	0.19	0	0.43	0.54	0.67	1	
PctVacantBoarded	0	1	0.20	0.22	0	0.06	0.13	0.27	1	
PctVacMore6Mos	0	1	0.43	0.19	0	0.29	0.42	0.56	1	
MedYrHousBuilt	0	1	0.49	0.23	0	0.35	0.52	0.67	1	
PctHousNoPhone	0	1	0.26	0.24	0	0.06	0.18	0.42	1	
PctWOFullPlumb	0	1	0.24	0.21	0	0.10	0.19	0.33	1	
OwnOccLowQuart	0	1	0.26	0.22	0	0.09	0.18	0.40	1	
OwnOccMedVal	0	1	0.26	0.23	0	0.09	0.17	0.39	1	
OwnOccHiQuart	0	1	0.27	0.24	0	0.09	0.18	0.38	1	
RentLowQ	0	1	0.35	0.22	0	0.17	0.31	0.49	1	
RentMedian	0	1	0.37	0.21	0	0.20	0.33	0.52	1	
RentHighQ	0	1	0.42	0.25	0	0.22	0.37	0.59	1	
MedRent	0	1	0.38	0.21	0	0.21	0.34	0.53	1	
MedRentPctHousInc	0	1	0.49	0.17	0	0.37	0.48	0.59	1	
MedOwnCostPctInc	0	1	0.45	0.19	0	0.32	0.45	0.58	1	
MedOwnCostPctIncNoMtg	0	1	0.40	0.19	0	0.25	0.37	0.51	1	
NumInShelters	0	1	0.03	0.10	0	0.00	0.00	0.01	1	
NumStreet	0	1	0.02	0.10	0	0.00	0.00	0.00	1	
PctForeignBorn	0	1	0.22	0.23	0	0.06	0.13	0.28	1	
PctBornSameState	0	1	0.61	0.20	0	0.47	0.63	0.78	1	
PctSameHouse85	0	1	0.54	0.18	0	0.42	0.54	0.66	1	
PctSameCity85	0	1	0.63	0.20	0	0.52	0.67	0.77	1	
PctSameState85	0	1	0.65	0.20	0	0.56	0.70	0.79	1	
LandArea	0	1	0.07	0.11	0	0.02	0.04	0.07	1	
PopDens	0	1	0.23	0.20	0	0.10	0.17	0.28	1	
PctUsePubTrans	0	1	0.16	0.23	0	0.02	0.07	0.19	1	
LemasPctOfficDrugUn	0	1	0.09	0.24	0	0.00	0.00	0.00	1	
ViolentCrimesPerPop	0	1	0.24	0.23	0	0.07	0.15	0.33	1	

Train and test set

Next, we want to split the data into a training (75%) and a test (25%) part. This can be done by random sampling with `sample`. Note that there is a `fold` variable in the data set, but here we want to follow our own train/test procedure.

```
set.seed(3940)

crime_split <- initial_split(crime,
                             prop = .75, breaks = 2)
# train split
crime_train <- training(crime_split)
# test split
crime_test <- testing(crime_split)
```

Now, prepare the training data for running regularized regression models via `glmnet`. Our prediction outcome is `ViolentCrimesPerPop`. As X, use all variables except `state`, `communityname`, and `fold`.

```
X <-  model.matrix(ViolentCrimesPerPop ~ .,crime_train)
y <- crime_train |> pull(ViolentCrimesPerPop)
```

Check whether X looks ok.

```
dim(X)
```

```
[1] 1495  100
```

Lasso

Estimate a sequence of Lasso models using `glmnet`. You can stick with the defaults for choosing a range of lambdas.

```
m1 <-  glmnet(X, y, alpha = 1)
```

Here we want to display lambda and the coefficients of the first and last Lasso model.

```
# lambda value
cat(str_c(
  "Lambda for the first LASSO model is ", round(m1$lambda[1], 6),
  "\nand Lambda for the last LASSO model is ",
  round(tail(m1$lambda, n=1), 6)
) )
```

```
Lambda for the first LASSO model is 0.173677
and Lambda for the last LASSO model is 0.000077
```

```
# coefficients
m1$beta |>
  as.matrix() |>
  as.data.frame() |>
  rownames_to_column() |>
  select(1:2, last_col(0)) |>
  rename(first_LASSO = 2, last_LASSO = 3) |>
  mutate_at(vars(2, 3), round, 6)
```

```
                rowname first_LASSO last_LASSO
1           (Intercept)           0   0.000000
2            population           0   0.000000
3         householdsize           0  -0.046712
4          racepctblack           0   0.140542
5          racePctWhite           0  -0.078411
6          racePctAsian           0   0.003365
7          racePctHisp            0   0.037784
8          agePct12t21            0   0.054710
9          agePct12t29            0  -0.242012
10         agePct16t24            0  -0.079467
11          agePct65up            0   0.073459
12           numbUrban            0  -0.105736
```

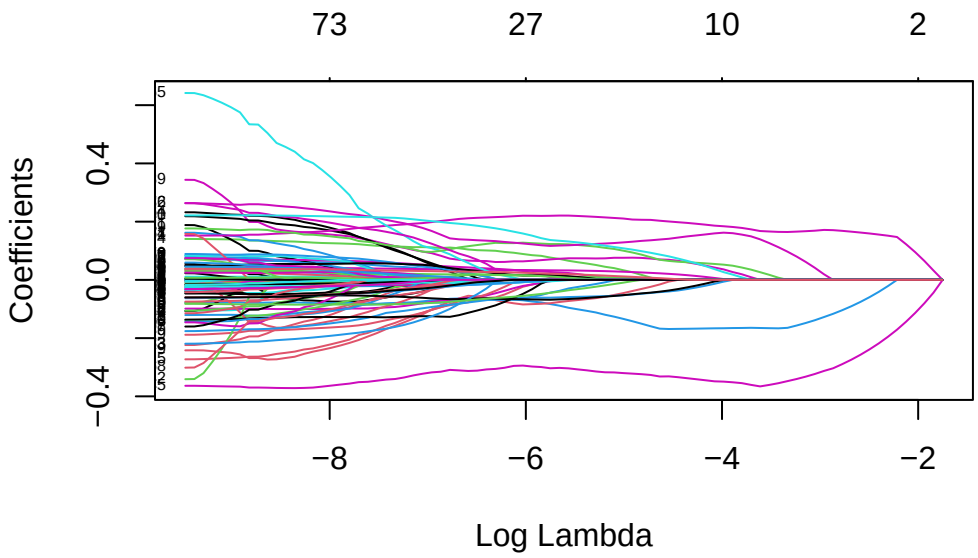
13	pctUrban	0	0.046265
14	medIncome	0	-0.106862
15	pctWWage	0	-0.272741
16	pctWFarmSelf	0	0.036157
17	pctWInvInc	0	-0.120688
18	pctWSocSec	0	0.000000
19	pctWPubAsst	0	-0.045734
20	pctWRetire	0	-0.098743
21	medFamInc	0	0.188307
22	perCapInc	0	0.000000
23	whitePerCap	0	-0.223758
24	blackPerCap	0	-0.008522
25	indianPerCap	0	-0.026912
26	AsianPerCap	0	0.010933
27	HispPerCap	0	0.071571
28	NumUnderPov	0	-0.038794
29	PctPopUnderPov	0	-0.188519
30	PctLess9thGrade	0	-0.139051
31	PctNotHSGrad	0	0.161928
32	PctBSorMore	0	0.084171
33	PctUnemployed	0	0.000000
34	PctEmploy	0	0.232146
35	PctEmplManu	0	-0.074705
36	PctEmplProfServ	0	-0.015853
37	PctOccupManu	0	0.078336
38	PctOccupMgmtProf	0	0.083536
39	MalePctDivorce	0	0.343719
40	MalePctNevMarr	0	0.218602
41	FemalePctDiv	0	0.000000
42	TotalPctDiv	0	-0.341068
43	PersPerFam	0	-0.023251
44	PctFam2Par	0	-0.030492
45	PctKids2Par	0	-0.363683
46	PctYoungKids2Par	0	0.001137
47	PctTeen2Par	0	-0.005665
48	PctWorkMomYoungKids	0	0.045662
49	PctWorkMom	0	-0.175982
50	NumIlleg	0	-0.035411
51	PctIlleg	0	0.152193
52	NumImmig	0	-0.136441
53	PctImmigRecent	0	0.031854
54	PctImmigRec5	0	-0.015620
55	PctImmigRec8	0	-0.001177

56	PctImmigRec10	0	0.000403
57	PctRecentImmig	0	-0.031779
58	PctRecImmig5	0	0.000000
59	PctRecImmig8	0	0.022800
60	PctRecImmig10	0	0.000000
61	PctSpeakEnglOnly	0	-0.019463
62	PctNotSpeakEnglWell	0	-0.105718
63	PctLargHouseFam	0	-0.143844
64	PctLargHouseOccup	0	0.000000
65	PersPerOccupHous	0	0.641817
66	PersPerOwnOccHous	0	-0.145868
67	PersPerRentOccHous	0	-0.160208
68	PctPersOwnOccup	0	-0.301488
69	PctPersDenseHous	0	0.176664
70	PctHousLess3BR	0	0.088980
71	MedNumBR	0	0.002126
72	HousVacant	0	0.263434
73	PctHousOccup	0	-0.046278
74	PctHousOwnOcc	0	0.157837
75	PctVacantBoarded	0	0.029697
76	PctVacMore6Mos	0	-0.062929
77	MedYrHousBuilt	0	-0.024099
78	PctHousNoPhone	0	0.024841
79	PctWOFullPlumb	0	-0.015400
80	OwnOccLowQuart	0	-0.114532
81	OwnOccMedVal	0	0.000000
82	OwnOccHiQuart	0	0.071088
83	RentLowQ	0	-0.219192
84	RentMedian	0	0.000000
85	RentHighQ	0	-0.006322
86	MedRent	0	0.263606
87	MedRentPctHousInc	0	0.048579
88	MedOwnCostPctInc	0	-0.060485
89	MedOwnCostPctIncNoMtg	0	-0.082457
90	NumInShelters	0	0.059486
91	NumStreet	0	0.221465
92	PctForeignBorn	0	0.074764
93	PctBornSameState	0	0.003789
94	PctSameHouse85	0	-0.042362
95	PctSameCity85	0	0.007277
96	PctSameState85	0	0.045544
97	LandArea	0	-0.017618
98	PopDens	0	-0.032132

```
99          PctUsePubTrans            0  -0.060171
100   LemasPctOfficDrugUn             0   0.037696
```

Now, plot the coefficient paths.

```
plot(m1, label=T,  xvar = "lambda")
```



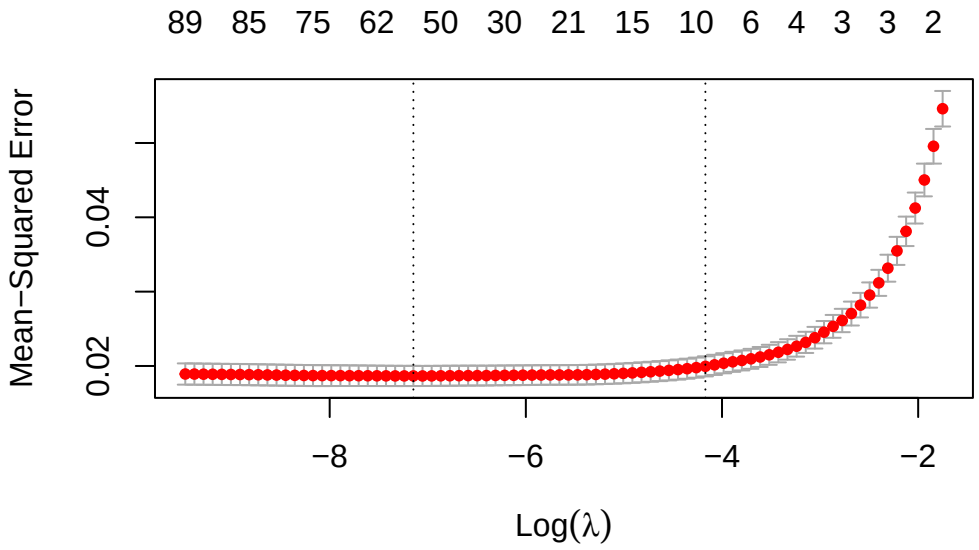
Cross-validation (CV), tuning the Lambda grid to find best models

Next, we need to decide which Lasso model to pick for prediction. Use CV for this.

```
m1_cv <- cv.glmnet(X, y, alpha = 1)
```

And plot the Cross-validation results.

```
plot(m1_cv)
```



In your own words, briefly describe the CV plot. (1) What is plotted here, (2) what can you infer about the relation between the number of variables and prediction accuracy?

- Here we plot the CV curve denoted by the red dotted line along with error bars across the lambda values, also known as the lambda grid. The dotted vertical lines represent the lowest lambda value which gives us the minimum mean CV error, and the second provides us the lambda value that gives the most regularized model resulting in a CV error within one standard error from the minimum lambda value. Consequently, these two proposed lambda values are reasonable tuning parameters for choosing a trained model to estimate testing errors and compare. The numbers above represents non-zero coefficient estimates. We can infer that having more features does not improve training accuracy as we would expect a U-shape. Instead, we suggest that only a proportion of features contribute to the training accuracy here to a certain point as after decreasing features after 10 we observe an increase in the training MSE.

Now, store the lambda value of the model with the smallest CV error as `bestlam1`.

```
bestlam1 <- m1_cv$lambda.min
```

Create `bestlam2` as the lambda according to the 1-standard error rule.

```
bestlam2 <- m1_cv$lambda.1se
```

Prediction in test set

Finally, we investigate the performance of our models in the test set. For this task, construct a X matrix from the test set.

```
Xt <- model.matrix(ViolentCrimesPerPop ~ ., crime_test)
```

Use the `predict` function to generate predicted values for both models (i.e., both lambdas stored earlier).

```
p_lasso_min <- predict(m1, s = bestlam1, newx = Xt)
p_lasso_1se <- predict(m1, s = bestlam2, newx = Xt)
```

Compute the test MSE of our models.

```
mean((p_lasso_min - crime_test$ViolentCrimesPerPop)^2)
```

```
[1] 0.01856262
```

```
mean((p_lasso_1se - crime_test$ViolentCrimesPerPop)^2)
```

```
[1] 0.01970093
```

In addition, use another performance metric and compute the corresponding values for both models.

```
postResample(p_lasso_min, crime_test$ViolentCrimesPerPop)
```

```
      RMSE   Rsquared        MAE
0.13624470 0.64810959 0.09419406
```

```
postResample(p_lasso_1se, crime_test$ViolentCrimesPerPop)
```

```
      RMSE   Rsquared        MAE
0.14035998 0.62727376 0.09756406
```

Which model is better? Does it depend on the performance measure that is used?

- We compare two models, the model with the minimum CV error and the model with the CV error within one standard error fromo the minimum error value. We determine that the smallest CV error LASSO model had a slightly lower (R)MSE than the 1SE model at the cost of having 43 more features making it the more complicated model. Additionally, we used a second performance measures R^2 and also observe that the smallest CV error model had a slightly higher r-squared value of .65, compared to the second model, .63. These models are similar on performance measures, yet we are inclined to pick the model with the lowest test error (minimum CV error) based on these two performance measures which corroborate each other. However, given that these model performance measures are almost similar, we would not be wrong to pick the 1SE model on the basis that it is more parsimonious, less features.

Tidy Remix: Using the tidy-workflow for LASSO

We begin our workflow by loading the tidymodels suite of packges and create a recipe of our model.

- We can scale and center features, but for the sake of matching the example above we will comment out these arguments.

```
# define the model with recipes
crime_rec <- recipe(ViolentCrimesPerPop ~ .,
                     # specify trainig data from before
                    data = crime_train) |>
  # assign all features as numeric
  step_zv(all_numeric(), -all_outcomes()) #|>
  # center & scale
  #step_normalize(all_numeric(), -all_outcomes())

crime_rec # prints information about the data
```

Next we define the type of model we want to fit to our data, in this case LASSO and set the engine to glmnet which we used earlier. By assigning mixture =1, we specify a LASSO model where mixture=0 would be a ridge model.

```
# build spec
lasso_spec <-  linear_reg(penalty = tune(), mixture = 1) |>
  set_mode("regression") |>
  set_engine("glmnet")

lasso_spec
```

```
Linear Regression Model Specification (regression)

Main Arguments:
  penalty = tune()
  mixture = 1
```

```
Computational engine: glmnet
```

We now create our modeling workflow. This will allow us to store modeling information, fit, and use to make predictions on the test set.

```
# convinience function set up workflow
## can add a model to it, used for setting up modular analyis
wf <- workflow() |>
  add_recipe(crime_rec)

wf # prints information to be used in modeling
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: None

-- Preprocessor ----------------------------------------------------------------
1 Recipe Step

* step_zv()
```

Once we have our workflow, we can now begin training the model on data.

```
# we can train the model with our workflow
lasso_fit <- wf |>
  add_model(lasso_spec) |>
  fit(data=crime_train)
```

As before, we want to print the first last model's lambda values which we can use the parsnip::extract_fit_engine() to get these values out of the fit object, and than pass it through the yardstick::tidy() to allow us to work with these values.

```
lasso_fit |>
  extract_fit_engine() |>
  tidy() |>
  select(lambda) |>
  filter(row_number() %in% c(1, n())) |>
  mutate_all( round, 6)
```

```
# A tibble: 2 x 1
    lambda
     <dbl>
1 0.174
2 0.000077
```

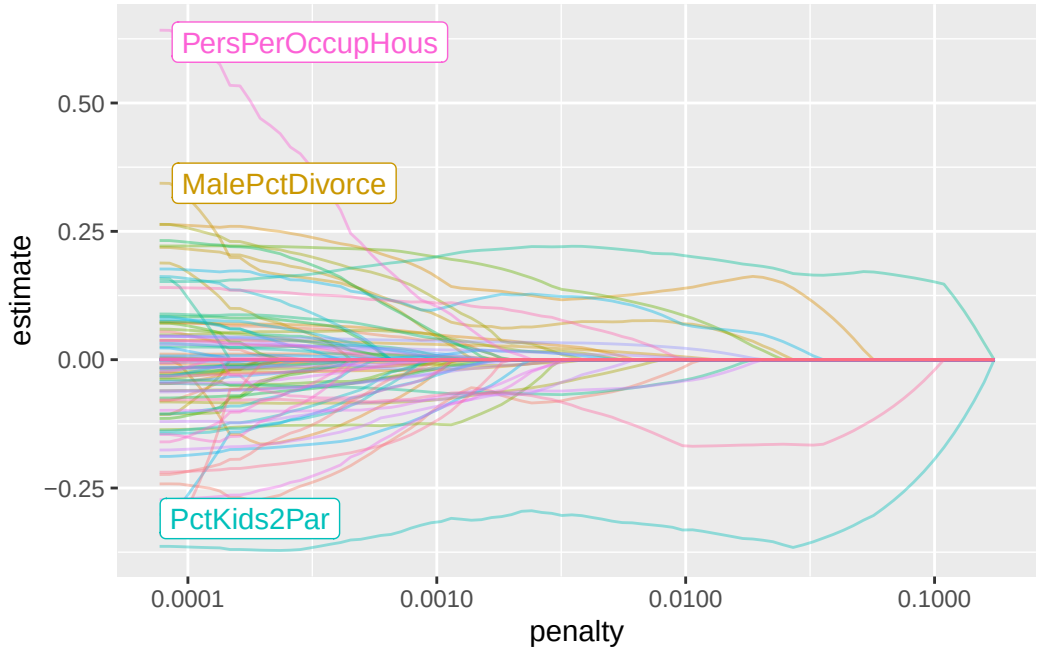
We can also subset coefficient beta values for any LASSO model using the parsnip::extract_fit_engine(). Here we show the first and last LASSO beta coefficients to match the previous process above.

```
lasso_fit |>
    extract_fit_engine() |>
    tidy() |>
  select(term, step, estimate) |>
  pivot_wider(names_from = step, values_from = estimate) |>
  select(1:2, last_col(0)) |>
  # convert NA to 0s
  mutate_at(vars(2, 3), replace_na, 0) |>
  # drop intercept
  filter(term != "(Intercept)") |>
  rename(first_LASSO = 2, last_LASSO = 3) |>
  mutate_at(vars(2, 3), round, 6)
```

```
# A tibble: 92 x 3
   term          first_LASSO last_LASSO
   <chr>               <dbl>      <dbl>
 1 population              0     0
 2 householdsize           0    -0.0467
 3 racepctblack            0     0.141
 4 racePctWhite            0    -0.0784
 5 racePctAsian            0     0.00336
 6 racePctHisp             0     0.0378
 7 agePct12t21             0     0.0547
 8 agePct12t29             0    -0.242
 9 agePct16t24             0    -0.0795
10 agePct65up              0     0.0735
# i 82 more rows
```

We plot the coefficient paths as before, the only difference is that workflowsets::autoplot() transforms lambda from the natural log for us.

```
lasso_fit |>
  extract_fit_parsnip() |>
  autoplot()
```



Tuning the penalty term to find optimal lambda using cross validation.

We start by taking a 10 fold of our training set using the rsample::vfold_cv().

```
crime_cv <- rsample::vfold_cv(crime_train, v=10)
```

We want to set the spacing for lambda using dials::grid_regular(). The levels argument is the number of parameters to use in the grid.

```
lamda_grid <- grid_regular(penalty(), levels=100)
```

Now we can tune the lambda grid with our cross validation approach while doing this in parallel.

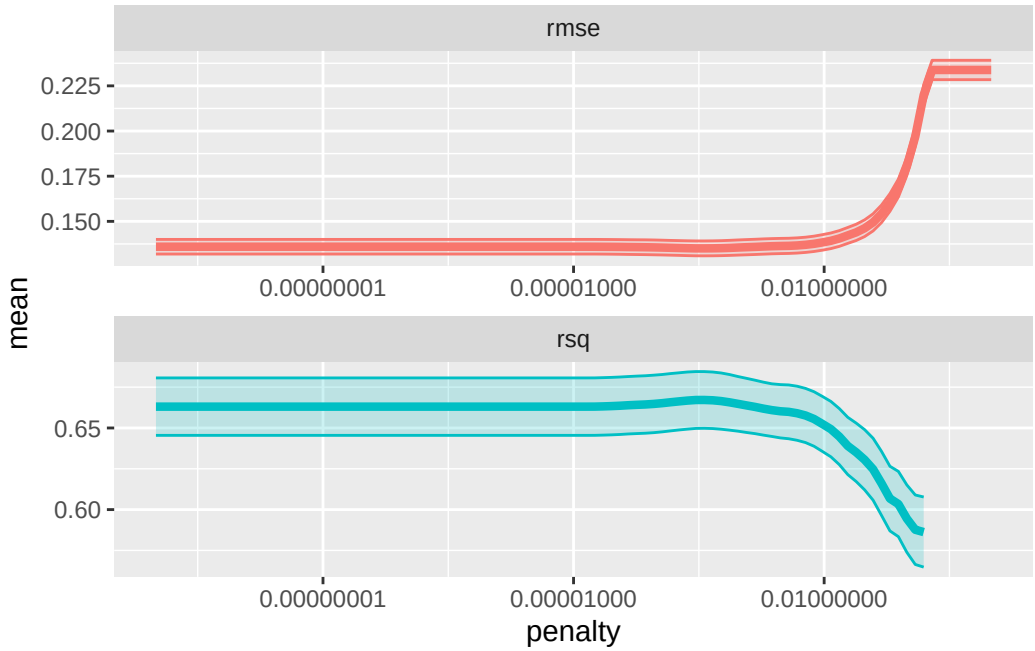
```
doParallel::registerDoParallel()

# tune the lambda grid
lasso_grid <- tune_grid(
  wf |> add_model(lasso_spec),
  resamples=crime_cv,
  grid=lamda_grid
)
```

Finally in this step, we want to visualize training error performance as before. The only difference is that instead of plotting log(lambda) we can plot RMSE and the bottom plot shows the amount of regularization affecting the performance metric.

- The tune::collect_metrics() will give us a table with performance training measures for 100 model candidates.
- We can see the impact of different values of regularization on the MSE below.

```
# visualize model training
lasso_grid |>
  tune::collect_metrics() |>
  ggplot(aes(penalty, mean, color=.metric)) +
  geom_ribbon(aes(ymin = mean - std_err, ymax = mean + std_err,
                  fill=.metric), alpha=.2) +
  geom_line(size=1.5) +
  facet_wrap(~.metric, scales="free", nrow=2) +
  scale_x_log10() +
  theme(legend.position = "none")
```


rmse
0.225
0.200
0.175
0.150
0.00000001
0.00001000
0.01000000
rsq
0.65
0.60
0.00000001
0.00001000
0.01000000
mean
penalty

Test error, following optimal tuning parameter

We are now ready to calculate MSE as before, by first finding the minimal lambda value and 1SE from minimal.

```
tidy_bestlam1 <- select_best(lasso_grid, metric="rmse")
```

We can now finalize our workflow.

```
lasso_final <- finalize_workflow(wf |> add_model(lasso_spec), tidy_bestlam1)
lasso_final_fit <- fit(lasso_final, data = crime_train)
```

We finalize the first model and show below that we get a near identical MSE as before, and RMSE. Additionally we print r-square and observe a near identical value as before.

```
augment(lasso_final_fit, new_data = crime_test) |>
  rmse(truth = ViolentCrimesPerPop, estimate = .pred) |>
  mutate(MSE = .estimate^2)
```

```
# A tibble: 1 x 4
  .metric .estimator .estimate    MSE
  <chr>   <chr>          <dbl>  <dbl>
1 rmse    standard       0.136 0.0186
```

```
augment(lasso_final_fit, new_data = crime_test) |>
  rsq(truth = ViolentCrimesPerPop, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rsq     standard       0.649
```

We can also take the same process to show how the second model, 1SE performs.

- Again we see a similar MSE value as before along with a similar RMSE, but not exact. Same for the r-square.

```
tidy_bestlam2 <- select_by_one_std_err(lasso_grid, metric = "rmse", desc(penalty))


lasso_final <- finalize_workflow(wf |> add_model(lasso_spec), tidy_bestlam2)
lasso_final_fit <- fit(lasso_final, data = crime_train)

augment(lasso_final_fit, new_data = crime_test) |>
  rmse(truth = ViolentCrimesPerPop, estimate = .pred) |>
  mutate(MSE = .estimate^2)
```

```
# A tibble: 1 x 4
  .metric .estimator .estimate    MSE
  <chr>   <chr>          <dbl>  <dbl>
1 rmse    standard       0.139 0.0192
```

```
augment(lasso_final_fit, new_data = crime_test) |>
  rsq(truth = ViolentCrimesPerPop, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rsq     standard       0.635
```

Bonus round, we can also examine feature importance using the vip package.

- Here we use the 1SE model, most parsimonious, and see the top features the model elected for us.
  - We only print below important features to the model.

```
library(vip)


# variance selection, feature importance
lasso_final_fit |>
  pull_workflow_fit() |>
  vip::vi(lambda = tidy_bestlam2$penalty) |>
  mutate(Imporatnce = abs(Importance),
         Variable = fct_reorder(Variable, Importance)) |>
  filter(Importance >=.0001) |>
  ggplot(aes(x=Importance, y=Variable, fill=Sign)) +
  geom_col(show.legend = FALSE) +
  scale_x_continuous(expand = c(0, 0)) +
  labs(y = NULL)
```