Parameter Tuning

Exercise 00 - Preprocessing Recipe

Before starting the exercises, here is an introduction to some tools from the tidymodels library that are very useful when performing the following tasks:

1. Data preprocessing for training and testing - through recipes.
2. Hyperparameter tuning through a cross-validation process.

1. Preparing Recipes for Your Data

First, we use "recipes" - from the recipes package - to prepare the data. In these recipes, we declare a set of steps to process the dataset before training and testing. For instance, we can use a recipe to center and scale the data, as shown in the example below. When declaring the recipe, the data remains unchanged. We need to call the 'prep' function to perform the recipe calculation and the 'juice' function to apply the recipe to the training set.

2. Hyperparameter Tuning

In order to tune hyperparameters through a cross-validation process, we need to apply the following steps:

1. For each hyperparameter θ of interest, do the following:
   (a) For each fold in our cross-validation set, do the following:
      i. Prepare the recipe for the training set of the current fold.
      ii. Train the model with the data prepared by the recipe.
      iii. Apply the resulting model to the testing set of the current fold, appropriately "baked" with the recipe.
      iv. Calculate the measure of interest, such as the RMSE.
   (b) Collect measures of interest, like RMSE, and calculate the average values across different folds.
2. Choose the parameter θ with the optimal value for the measure.

```
library(tidymodels)

rcp <- function(df, target, reg_type) {

  # Split the dataset into training and testing sets
  init.split <- initial_split(df, prop = 0.8)
  train <- training(init.split)
  test <- testing(init.split)

  # Scale the data and set up the recipe
  rcp <- recipe(as.formula(paste0(target, " ~ .")), data = df) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())
```

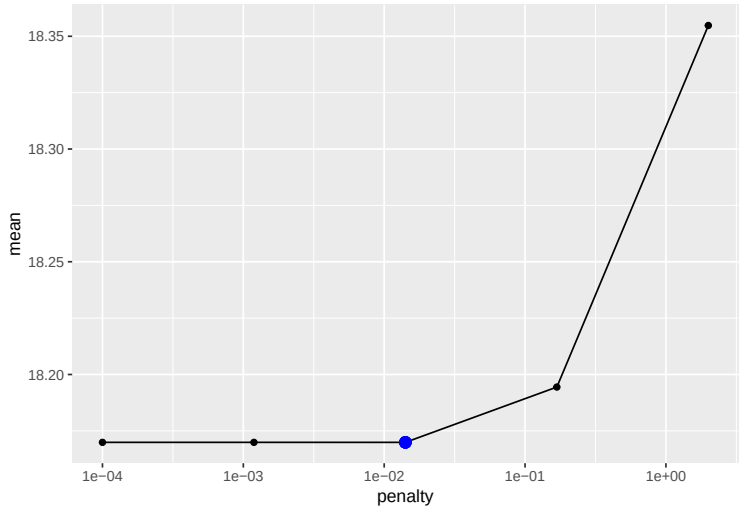
```
  # Prepare the recipe on training data
  rcp_prep <- prep(rcp, training = train)
  train_prep <- juice(rcp_prep)

  # Run the recipe on testing data
  test_prep <- bake(rcp_prep, new_data = test)

  # Create regularized Linear Regression models
  # with penalty parameters to be tuned
  if (reg_type == "Ridge") {
    lin.model <- linear_reg(penalty = tune(), mixture = 0)
  } else if (reg_type == "Lasso") {
    lin.model <- linear_reg(penalty = tune(), mixture = 1)
  }

  lin.model <- set_engine(lin.model, "glmnet")

  # Use grid search to find the optimal parameter for penalty
  lm.grid <- grid_regular(
  penalty(range = log10(c(0.0001,2))),
  levels = 5)

  # Generate 10-fold cross validation
  vfolds <- vfold_cv(df, v = 10)

  # Calculate parameters
  tune.res <- tune_grid(
  lin.model,
  rcp,
  resamples = vfolds,
  grid = lm.grid
  )

  # Create a plot to show the optimal parameter
  plot <- tune.res %>%
  collect_metrics() %>%
  filter(.metric == "rmse") %>%
  ggplot(aes(x = penalty, y = mean)) +
  geom_line() +
  geom_point()

  print(plot)

  # Return the optimal parameter for penalty (lowest rmse)
  lambda <- show_best(tune.res, metric = "rmse") %>%
  arrange(mean) %>%
  slice(1:1) %>% pull(penalty)

  return(lambda)
}
```

Exercise 01

We will use the FIFA database from the previous lesson. Run Lasso Regression and choose the best value for λ using cross validation, which is the best option for penalizing the L1 norm. After selecting the optimal parameter, fit the model to all the data and observe which features are selected. Create a plot showing how coefficients are shrunk as the penalty increases.

```
library(tidyverse)

# Load the data
file_url <- "https://drive.google.com/uc?export=download&id=1jiWcGsl_tbqK5F0ryUTq48kcDTKWTTuk"
df <- file_url %>%
  read.csv %>%
  as_tibble %>%
  select(Age, Overall, Potential, Wage, Special,
  Acceleration, Aggression, Agility, Balance, Ball.control,
  Composure, Crossing, Curve, Dribbling, Finishing,Positioning,Vision,Stamina,
  Strength) %>%
  mutate( Wage = as.integer(str_extract(Wage,"[0-9]+")) ) %>%
  mutate_if(is.character,as.integer) %>%
  na.omit()


# Define a recipe to center and scale the data
rec <- recipe( Wage ~ ., data = df ) %>%
step_center(all_predictors()) %>%
step_scale(all_predictors())

# Use grid search for parameter tuning
lm.grid <- grid_regular( penalty(range = log10(c(0.0001,2))),
                         levels = 5
)

# Generate 10-fold cross-validation sets
vfolds <- vfold_cv(df, v = 10)

# Define the model using Lasso Regression
lin.model <- linear_reg( penalty = tune(),
                         mixture = 1
) %>%
set_engine("glmnet")

# Tune the parameters
tune.res <- tune_grid( lin.model,
                       rec,
                       resamples = vfolds,
                       grid = lm.grid
)

# Plot a graph with the results
tune.res %>%
collect_metrics() %>%
filter(.metric == "rmse") %>%
ggplot(aes(x = penalty, y = mean)) +
```

```
scale_x_continuous(trans = 'log10') +
geom_line() +
geom_point() +
geom_point(aes( x = penalty[3], mean[3]),
           color = "blue",
           size = 3
)
```




```
# Show the best parameters
show_best(tune.res,metric = "rmse")
```

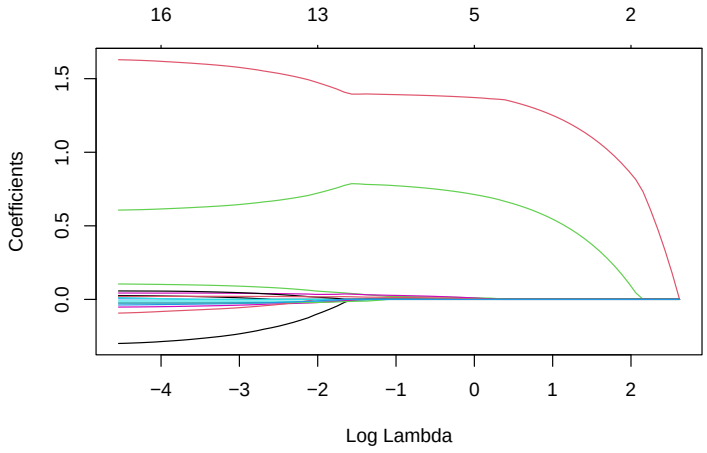
```
## # A tibble: 5 x 7
##   penalty .metric .estimator  mean     n std_err .config
##     <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.0141  rmse    standard    18.2    10   0.599 Preprocessor1_Model3
## 2 0.0001  rmse    standard    18.2    10   0.599 Preprocessor1_Model1
## 3 0.00119 rmse    standard    18.2    10   0.599 Preprocessor1_Model2
## 4 0.168   rmse    standard    18.2    10   0.606 Preprocessor1_Model4
## 5 2       rmse    standard    18.4    10   0.642 Preprocessor1_Model5
```

```
penalty <- (show_best(tune.res,metric = "rmse") %>% pull(penalty))[1]

# Calculate the coefficients
lin.fit <- linear_reg( penalty = penalty,
mixture = 1) %>%
set_engine("glmnet") %>%
fit( Wage ~ . , data = df )

# Extract the coefficients
betas <- lin.fit %>%
pluck("fit") %>%
coef(s = penalty)
print(betas)
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
```

```
##                         s1
## (Intercept)  -1.316755e+02
## Age          -2.943795e-01
## Overall       1.624419e+00
## Potential     6.099492e-01
## Special       .
## Acceleration -3.654544e-02
## Aggression   -1.084314e-02
## Agility      -5.094264e-02
## Balance       2.506747e-02
## Ball.control -8.848495e-02
## Composure     1.034600e-01
## Crossing      .
## Curve         3.939348e-03
## Dribbling     9.319129e-03
## Finishing     4.302562e-02
## Positioning   5.694561e-02
## Vision        2.000341e-02
## Stamina      -2.465036e-02
## Strength     -2.461760e-02
```

```r
# Plot a graph of the shrinkage
plot( lin.fit %>% pluck("fit"), xvar = "lambda")
```

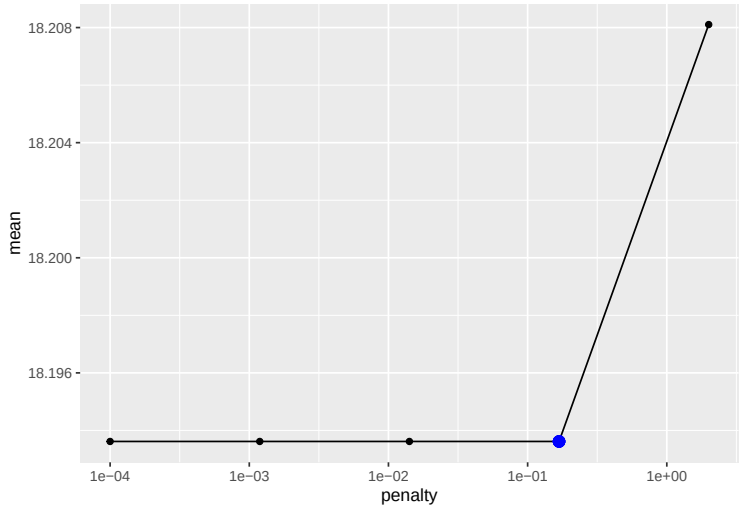


Exercise 02

Repeat the previous exercise using Ridge Regression. This can be done by changing the 'mixture' parameter to 0 in the 'linear_reg' function. Notice how the j parameters are shrunk as the penalty increases and compared the results to those from the previous exercise. Justify the number of coefficients set to zero using Lasso and Ridge Regression.

```r
# Define the model using Ridge Regression
lin.model <- linear_reg( penalty = tune(),
                         mixture = 0
) %>%
```

```r
set_engine("glmnet")

# Tune the parameters
tune.res <- tune_grid( lin.model,
                       rec,
                       resamples = vfolds,
                       grid = lm.grid
)

# Plot a graph with the results
tune.res %>%
collect_metrics() %>%
filter(.metric == "rmse") %>%
ggplot(aes(x = penalty, y = mean)) +
scale_x_continuous(trans = 'log10') +
geom_line() +
geom_point() +
geom_point(aes( x = penalty[4], mean[4]),
           color = "blue",
           size = 3
)
```



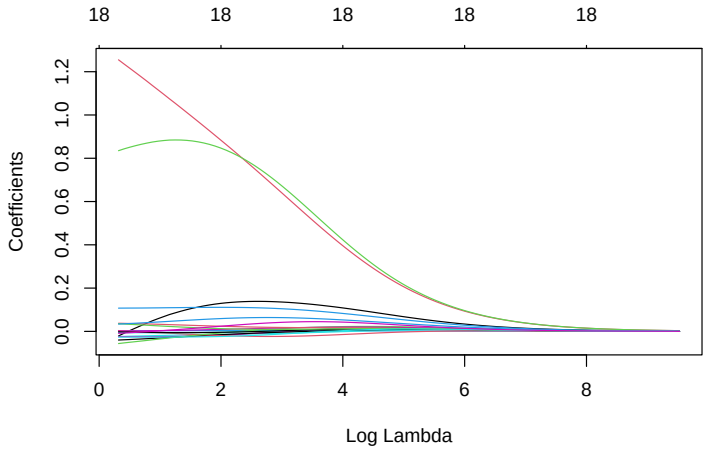

```r
# Show the best parameters
show_best(tune.res,metric = "rmse")
```

```
## # A tibble: 5 x 7
##   penalty .metric .estimator  mean     n std_err .config
##     <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.0001  rmse    standard    18.2    10   0.607 Preprocessor1_Model1
## 2 0.00119 rmse    standard    18.2    10   0.607 Preprocessor1_Model2
## 3 0.0141  rmse    standard    18.2    10   0.607 Preprocessor1_Model3
## 4 0.168   rmse    standard    18.2    10   0.607 Preprocessor1_Model4
## 5 2       rmse    standard    18.2    10   0.610 Preprocessor1_Model5
```

```r
penalty <- (show_best(tune.res,metric = "rmse") %>% pull(penalty))[1]

# Calculate the coefficients
lin.fit <- linear_reg( penalty = penalty,
mixture = 0) %>%
set_engine("glmnet") %>%
fit( Wage ~ . , data = df )

# Extract the coefficients
betas <- lin.fit %>%
pluck("fit") %>%
coef(s = penalty)
print(betas)
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                          s1
## (Intercept)  -1.355250e+02
## Age          -2.130769e-02
## Overall       1.254960e+00
## Potential     8.350092e-01
## Special       2.903180e-03
## Acceleration -2.391981e-02
## Aggression   -2.392827e-02
## Agility      -4.004113e-02
## Balance       3.731210e-03
## Ball.control -5.625603e-02
## Composure     1.074157e-01
## Crossing     -5.714549e-03
## Curve         1.458683e-03
## Dribbling    -2.185288e-03
## Finishing     3.753086e-02
## Positioning   3.548807e-02
## Vision        3.380685e-02
## Stamina      -2.649387e-02
## Strength     -8.890575e-03
```

```r
# Plot a graph of the shrinkage
plot( lin.fit %>% pluck("fit"), xvar = "lambda")
```



In Lasso regression, as we increase the values of lambda, most coefficients are set to zero. This is because the L1 penalty of the Lasso regression tends to zero out coefficients, thus performing feature selection.

In contrast, in Ridge regression, coefficients tend to approach zero for high values of lambda, but they are never set to zero due to the L2 penalty.

Therefore, Lasso regression is capable of performing feature selection by explicitly setting coefficients to zero, and it tends to yield better results when there are few significant features. On the other hand, Ridge regression often performs better in models with a large number of features associated with the target variable.

Exercise 03

In this exercise, we will fit an Elastic-Net regression model to the 'Salaries' dataset from the 'car' library. This dataset contains salaries of college professors. There are several categorical variables - or nominal qualitative variables -, represented as factors. We can add dummy variables to the recipe using the 'step_dummy' function.

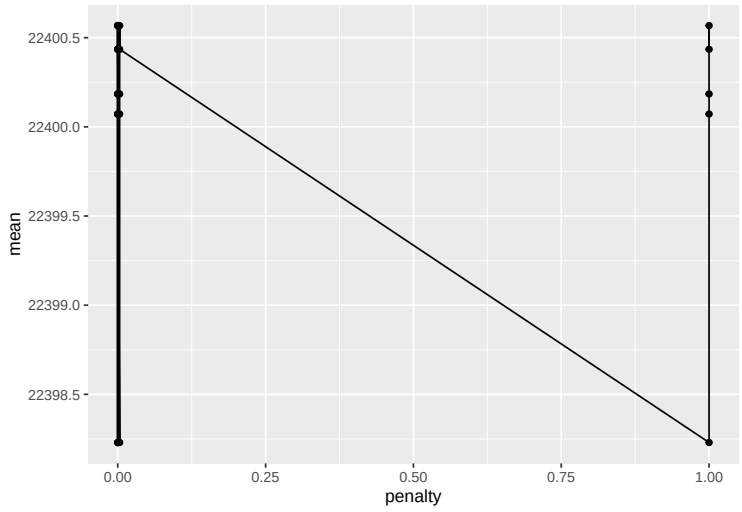
To run Elastic-Net Regression, choose a value for the 'mixture' parameter between (0, 1). Use the 'tune()' function to adjust both the 'penalty' and 'mixture' parameters. Use grid search for cross-validation.

```
library(car)
df <- as_tibble(Salaries)

# Creating a Preprocessing Recipe for Elastic-Net Regression:

# Split the dataset intro training and testing sets
init.split <- initial_split(df, prop = 0.8)
train <- training(init.split)
test <- testing(init.split)

# Add dummy variables to the recipe
en_rcp <- recipe(salary ~ ., data = df) %>%
step_dummy(all_nominal()) %>%
step_center(all_predictors()) %>%
step_scale(all_predictors())

# set up the grid for parameter tuning
```

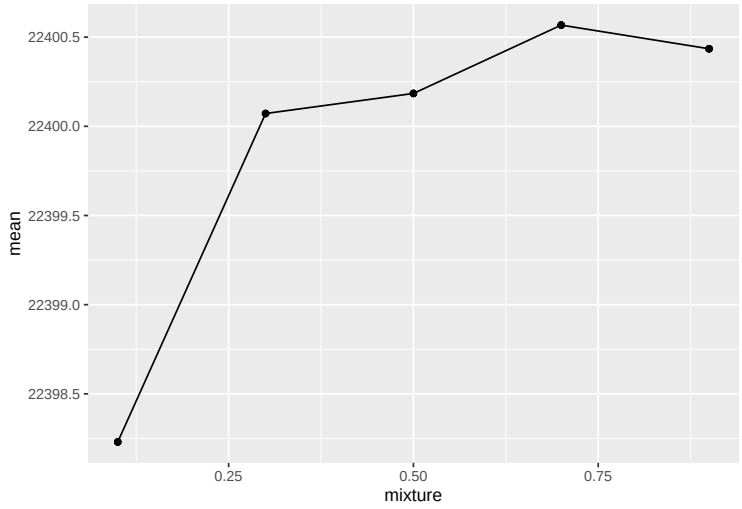
```
grid <- grid_regular( penalty(),
                      mixture(range = c(0.1,0.9)),
                      levels = 5
)


# Create an Elastic-Net Regression model
# with penalty and mixture parameters to be tuned
lin.model <- linear_reg(penalty = tune(),
                        mixture = tune())
lin.model <- set_engine(lin.model, "glmnet")

# Use grid search to find the optimal parameters for penalty and mixture
lm.grid <- grid_regular( penalty(),
                         mixture(range = c(0.1,0.9)),
                         levels = 5)

# Generate 10-fold cross validation
vfolds <- vfold_cv(df, v = 10)

# Calculate parameters
tune.res <- tune_grid(
  lin.model,
  en_rcp,
  resamples = vfolds,
  grid = lm.grid
)

# Show best parameters
show_best(tune.res, metric = "rmse")
```

```
## # A tibble: 5 x 8
##        penalty mixture .metric .estimator   mean     n std_err .config
##          <dbl>   <dbl> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1 0.0000000001     0.1 rmse    standard   22398.    10   1090. Prepro~
## 2 0.0000000316     0.1 rmse    standard   22398.    10   1090. Prepro~
## 3 0.00001          0.1 rmse    standard   22398.    10   1090. Prepro~
## 4 0.00316          0.1 rmse    standard   22398.    10   1090. Prepro~
## 5 1                0.1 rmse    standard   22398.    10   1090. Prepro~
```

```
# Create a plot to show the optimal parameter for penalty
plot <- tune.res %>%
collect_metrics() %>%
filter(.metric == "rmse") %>%
ggplot(aes(x = penalty, y = mean)) +
geom_line() +
geom_point()

print(plot)
```




```
# Create a plot to show the optimal parameter for mixture
plot <- tune.res %>%
collect_metrics() %>%
filter(.metric == "rmse") %>%
ggplot(aes(x = mixture, y = mean)) +
geom_line() +
geom_point()

print(plot)
```

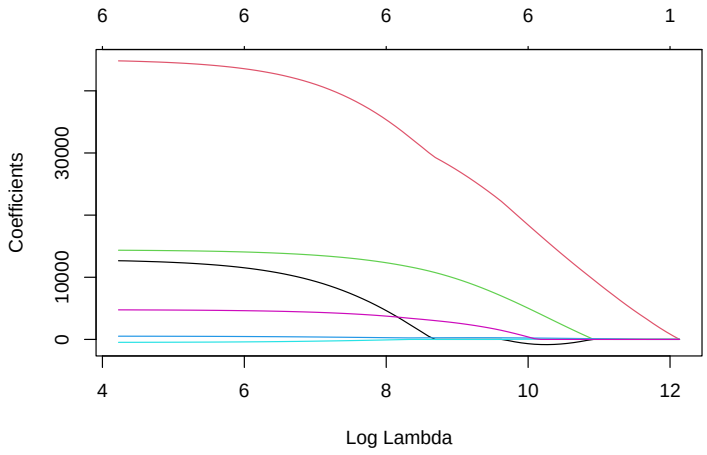



```
# Return the optimal parameter for penalty (lowest rmse)
penalty <- (show_best(tune.res,metric = "rmse") %>% pull(penalty))[1]

# Return the optimal parameter for mixture (lowest rmse)
mixt <- (show_best(tune.res,metric = "rmse") %>% pull(mixture))[1]

# Calculate the coefficients using optimal parameters
lin.fit <- linear_reg( penalty = penalty,
```

```
mixture = mixt) %>%
set_engine("glmnet") %>%
fit( salary ~ . , data = df )

# Extract the coefficients
betas <- lin.fit %>%
pluck("fit") %>%
coef(s = penalty)
print(betas)
```

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                          s1
## (Intercept)   66240.0459
## rankAssocProf 12659.2776
## rankProf      44820.5580
## disciplineB   14350.4455
## yrs.since.phd   518.8614
## yrs.service    -470.0371
## sexMale        4755.2620
```

```
# Plot a graph of the shrinkage
plot( lin.fit %>% pluck("fit"), xvar = "lambda")
```



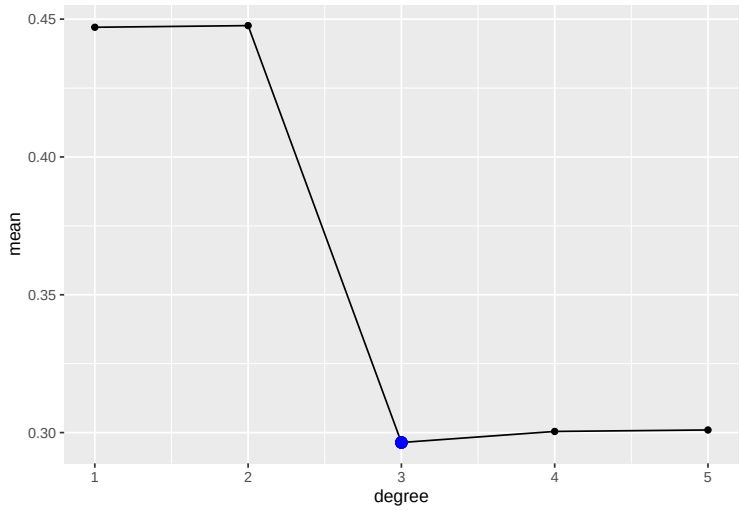
Note that the error is very high, even after tuning the hyperparameters. This means that the linear regression may not be not suitable for this dataset. Perhaps it is necessary to use a logarithmic scale for the salaries before performing linear regression or methods that allow for more flexibility in the model.

Exercise 04

We will generate an artificial database for this exercise, which follows a polynomial of degree 3. If we did not know the underlying degree, we want to determine the best degree for our polynomial regression. To do this, we will try degrees ranging from 1 to 5 by specifying the degree parameter in the 'set_poly' function with 'tune()'. For simple regression without regularization, use 'set_engine("lm")' in the model definition.

After fitting the model, store the result in the 'tune.res' variable and create a plot to visualize the results.

```r
library(tidymodels)
df <- tibble( x = runif(100, -1, 1),
  y = 2 * x ** 3 + x + 10 + rnorm(100, 0, 0.3))

# Creating a Preprocessing Recipe for Polynomial Regression:

# Split the dataset into training and testing sets
init.split <- initial_split(df, prop = 0.8)
train <- training(init.split)
test <- testing(init.split)

# Recipe to select the optimal degree for the polynomial
rec_poly <- recipe(y ~ . ,df) %>%
step_poly(x, degree = tune())

# Create a Polynomial Regression model
# with the degree to be tuned
lin.model <- linear_reg() %>%
  set_engine("lm")

# Create a grid with different degrees to be tested
lm.grid <- expand.grid(degree = seq(1, 5))

# Generate 10-fold cross validation
vfolds <- vfold_cv(df, v = 10)

# Calculate parameters
tune.res <- tune_grid(
lin.model,
rec_poly,
resamples = vfolds,
grid = lm.grid
)

# Create a plot to show the optimal degree
plot <- tune.res %>%
collect_metrics() %>%
filter(.metric == "rmse") %>%
ggplot(aes(x = degree, y = mean)) +
geom_line() +
geom_point() +
geom_point(aes( x = degree[3], mean[3]),
           color = "blue",
           size = 3
)

print(plot)
```




```
# Return the optimal degree (lowest rmse)
best_degree <- show_best(tune.res, metric = "rmse") %>%
arrange(mean) %>%
slice(1:1) %>% pull(degree)

print(paste("The optimal degree is", best_degree))
```

```
## [1] "The optimal degree is 3"
```