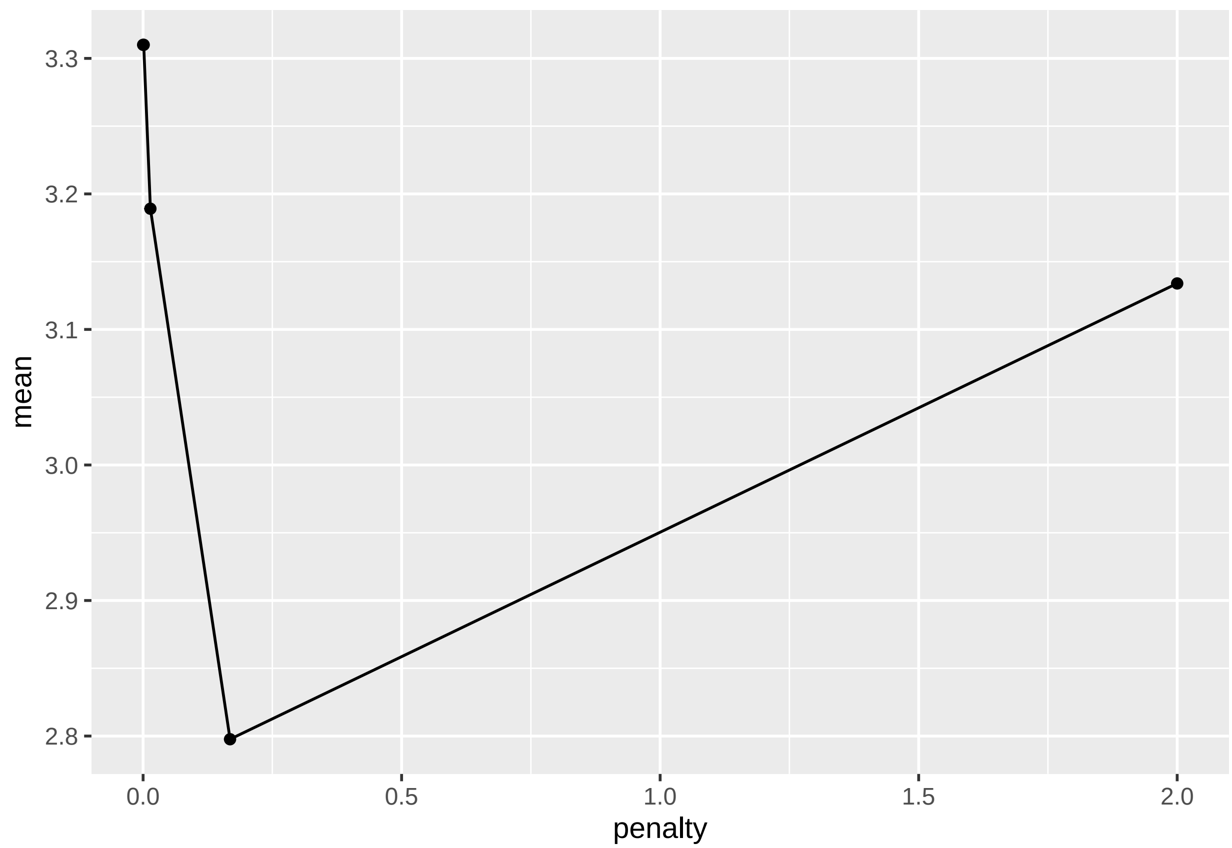
Solução Lista 04 - Aprendizado de Máquinas

18 March, 2025

```r
#Pré Processamento de dados
#library(tidymodels)

df = as_tibble(mtcars)

init.split = initial_split(df, prop = 0.8)
train = training(init.split)
test = testing(init.split)

receita = recipe( mpg ~ ., data = df ) %>%
  step_center(all_predictors()) %>%   # Centra os dados pela média
  step_scale(all_predictors()) # Escalona os preditores com o desvio padrão

receita_prep = prep(receita, training = train)
# ^ Prepara a receita sobre os
# dados de treinamento
train_prep = juice(receita_prep) # < Altera o conjunto de dados de treinamento

map_dbl(train_prep,mean)

map_dbl(train_prep,sd)

test_prep = bake(receita_prep,new_data = test)
# < Altera o conjunto de testes
# de acordo com nossa receita de
# preparação dos dados.

map_dbl(test_prep,mean)
map_dbl(test_prep,sd)

#Escolher os hiper-parametros
library(glmnet)
## Define um modelo de regressão linear regularizada,
## queremos encontrar o melhor parâmetro de penalização
## \lambda (= penalty)
lin.model = linear_reg(penalty = tune(), #Marcamos a penalização para ajuste com o tune
                       mixture = 1) %>% # 0 = Ridge, 1 = Lasso, (0,1) = Elastic-Net
            set_engine('glmnet')

## Malha para buscar o melhor valor para o parâmetro penalty.
## A função penalty() indica que queremos uma malha para este
## parâmetro e indica limiares recomendados para o parâmetro.

lm.grid <- grid_regular(# Define intervalo e escala recomendada para o
```

```r
                        # parâmetro penalty. Neste caso, alterei para
                        # o intervalo de busca para [0.0001,2] (chamei
                        # a função log10 porque os parâmetros são escalonados
                        # em uma escala de log na base 10).
                        penalty(range = log10(c(0.0001, 2))),
                        levels = 5) #Define o número de pontos que desejamos gerar
                                    #Para cada parâmetro testado


## Gerando 10-folds para validação cruzada
vfolds = vfold_cv(df, v = 10)

## Calculando parâmetros
tune.res = tune_grid(
  lin.model,          # Nosso modelo de aprendizado de máquina
  receita,            # Nossa receita de preparo de dados
  resamples = vfolds, # O nosso conjunto de validação cruzada
  grid = lm.grid      # Malha de parâmetros para busca
)


tune.res %>%
  collect_metrics() %>%
  filter(.metric == "rmse") %>%
  ggplot(aes(x = penalty, y = mean)) +
  geom_line() +
  geom_point()
```




```r
#Podemos ver os melhores com o comando:
show_best(tune.res,metric = "rmse")

#Resultados omitidos do pdf
```

Solução Exercício 1

```r
#Exercicio 1
#Carregar e pré-processar os dados:
#library(tidyverse)

# Carregar os dados
file_url = "https://drive.google.com/uc?export=download&id=1jiWcGsl_tbqK5FOryUTq48kcDTKWTTuk"
df = file_url %>%
  read.csv %>%
  as_tibble %>%
  select(Age, Overall, Potential, Wage, Special,
         Acceleration, Aggression, Agility, Balance, Ball.control, Composure, Crossing, Curve, Dribbling
  mutate(Wage = as.integer(str_extract(Wage, "[0-9]+"))) %>%
  mutate_if(is.character, as.integer) %>%
```

```r
  na.omit()

#Aplicar o método Lasso com validação cruzada
# Definir o modelo Lasso
lasso_model = linear_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet")

# Definir a validação cruzada
folds = vfold_cv(df, v = 10)

# Criar um grid de valores de lambda para testar
lambda_grid = grid_regular(penalty(), levels = 50)

# Ajustar o modelo com validação cruzada
lasso_tune = tune_grid(
  lasso_model,
  Wage ~ .,
  resamples = folds,
  grid = lambda_grid,
  metrics = metric_set(rmse)
)

# Selecionar o melhor lambda com base no menor RMSE
best_lambda = select_best(lasso_tune, metric = "rmse")
best_lambda
```

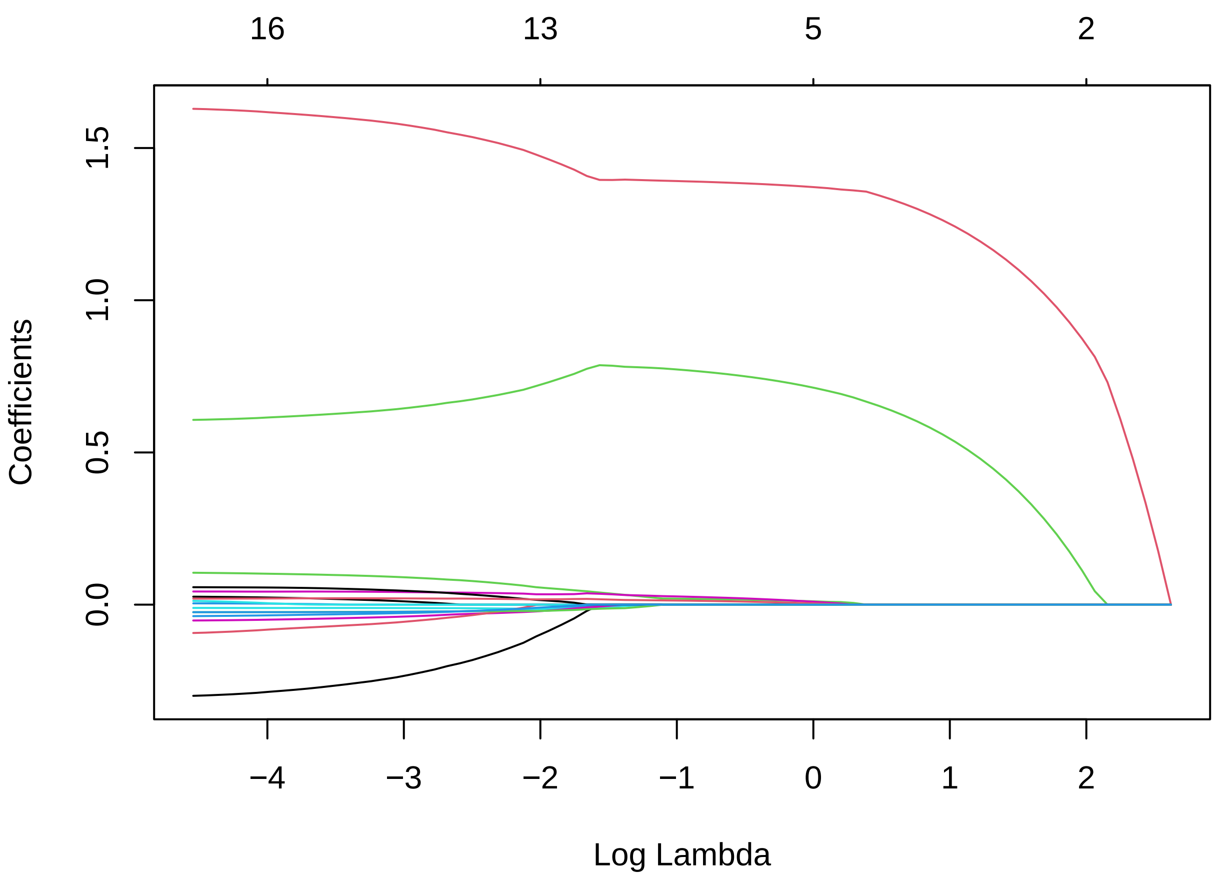
```
## # A tibble: 1 x 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 Preprocessor1_Model01
```

```r
# Ajustar o modelo final com o melhor lambda
final_lasso = finalize_model(lasso_model, best_lambda) %>%
  fit(Wage ~ ., data = df)

# Extrair os coeficientes
betas = final_lasso %>%
  pluck("fit") %>%
  coef(s = best_lambda$penalty)

# Exibir os coeficientes
betas
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)  -1.315680e+02
## Age          -2.994345e-01
## Overall       1.628711e+00
## Potential     6.069710e-01
## Special       .
## Acceleration -3.762196e-02
## Aggression   -1.076259e-02
## Agility      -5.203326e-02
```

```
## Balance       2.620403e-02
## Ball.control -9.305066e-02
## Composure     1.048813e-01
## Crossing      .
## Curve         4.220616e-03
## Dribbling     1.227868e-02
## Finishing     4.326695e-02
## Positioning   5.740235e-02
## Vision        1.984738e-02
## Stamina      -2.471196e-02
## Strength     -2.470653e-02
```

```
# Ajustar as margens do gráfico
par(mar = c(4, 4, 2, 1))  # Reduzir as margens

# Plotar o gráfico
plot(final_lasso %>% pluck("fit"), xvar = "lambda")
abline(v = log(best_lambda$penalty), col = "red", lty = 2)
```



Solução Exercício 2

```r
# Exercicio 2
library(tidyverse)
library(tidymodels)

# Carregar os dados (mesmo pré-processamento do Exercício 1)
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
  mutate(Wage = as.integer(str_extract(Wage, "[0-9]+"))) %>%
  mutate_if(is.character, as.integer) %>%
  na.omit()

# Definir o modelo Ridge (mixture = 0)
ridge_model <- linear_reg(penalty = tune(), mixture = 0) %>%
  set_engine("glmnet")

# Definir a validação cruzada
folds <- vfold_cv(df, v = 10)

# Criar um grid de valores de lambda para testar
lambda_grid <- grid_regular(penalty(), levels = 50)

# Ajustar o modelo com validação cruzada
ridge_tune <- tune_grid(
  ridge_model,
  Wage ~ .,
  resamples = folds,
  grid = lambda_grid,
  metrics = metric_set(rmse)
)

# Selecionar o melhor lambda com base no menor RMSE
best_lambda_ridge <- select_best(ridge_tune, metric = "rmse")
best_lambda_ridge
```

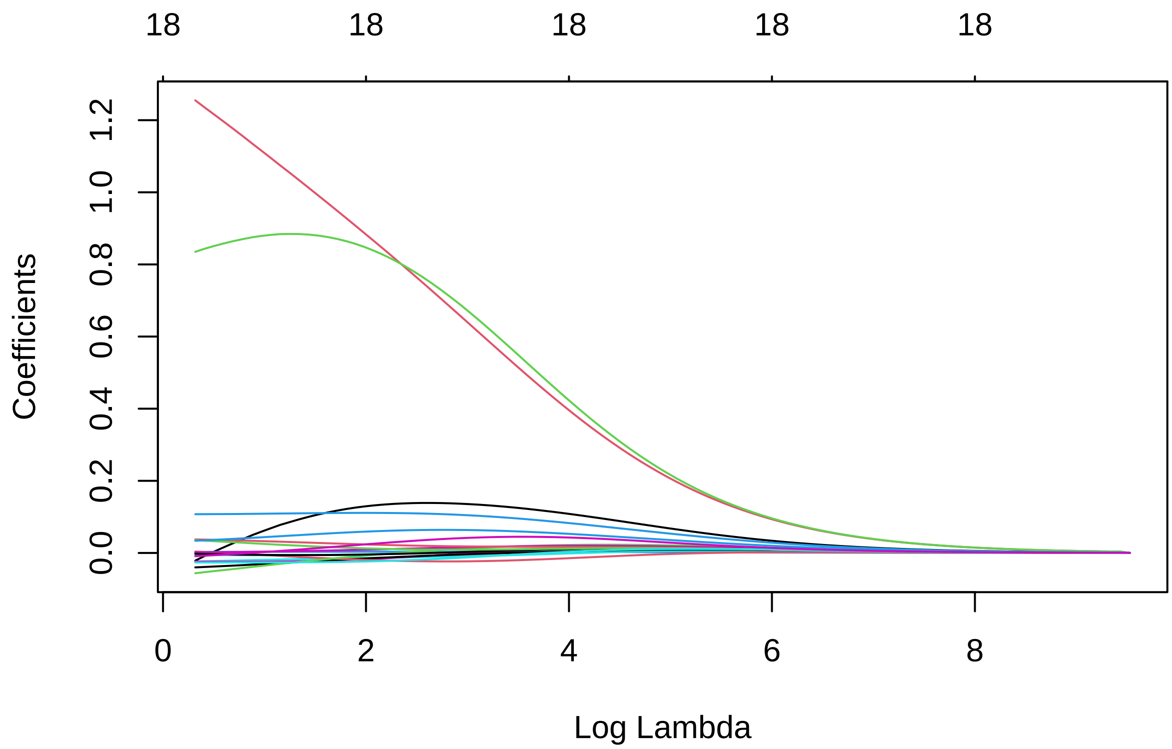
```
## # A tibble: 1 x 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 Preprocessor1_Model01
```

```r
# Ajustar o modelo final com o melhor lambda
final_ridge <- finalize_model(ridge_model, best_lambda_ridge) %>%
  fit(Wage ~ ., data = df)

# Extrair os coeficientes
betas_ridge <- final_ridge %>%
  pluck("fit") %>%
```

```r
  coef(s = best_lambda_ridge$penalty)

# Exibir os coeficientes
betas_ridge
```

```
## 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
# Gráfico do decaimento dos coeficientes
plot(final_ridge %>% pluck("fit"), xvar = "lambda")
abline(v = log(best_lambda_ridge$penalty), col = "red", lty = 2)
```



Comparação entre os métodos de Regressão

A escolha entre Ridge e Lasso depende da natureza do problema: o Ridge reduz a magnitude dos coeficientes sem zerá-los, sendo ideal quando todas as variáveis são consideradas relevantes. Já o Lasso realiza seleção de variáveis, zerando coeficientes de preditores irrelevantes, sendo mais adequado quando há suspeita de que algumas variáveis não contribuem para o modelo. Portanto, use Ridge para problemas onde todas as variáveis são importantes e Lasso quando deseja selecionar um subconjunto de preditores relevantes.

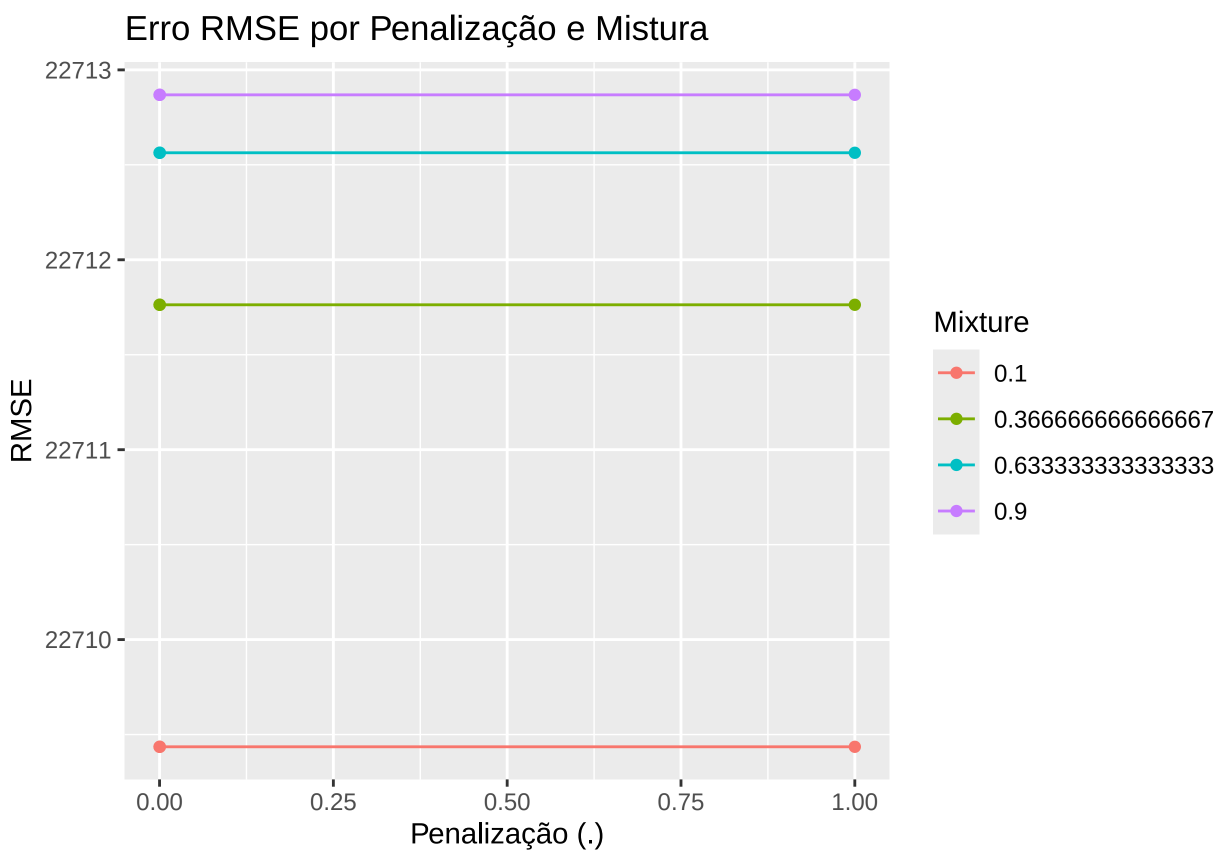
Resolução Exercício 3

```
# Importação das bibliotecas
# library(tidymodels)  # Conjunto de pacotes para ML
# library(tidyverse)   # Manipulação de dados
# library(car)         # Banco de dados Salaries

# Importação e exploração dos dados
df <- as_tibble(Salaries)
glimpse(df)
```

```
## Rows: 397
```

```
## Columns: 6
## $ rank          <fct> Prof, Prof, AsstProf, Prof, Prof, AssocProf, P~
## $ discipline    <fct> B, B, B, B, B, B, B, B, B, B, B, B, B, B, B, B~
## $ yrs.since.phd <int> 19, 20, 4, 45, 40, 6, 30, 45, 21, 18, 12, 7, 1~
## $ yrs.service   <int> 18, 16, 3, 39, 41, 6, 23, 45, 20, 18, 8, 2, 1,~
## $ sex           <fct> Male, Male, Male, Male, Male, Male, Male, Male~
## $ salary        <int> 139750, 173200, 79750, 115000, 141500, 97000, ~
```

```r
# Criando a receita de pré-processamento
receita <- recipe(salary ~ ., data = df) %>%
  step_dummy(all_nominal(), one_hot = TRUE) %>%  # Variáveis categóricas em dummies
  step_normalize(all_predictors())               # Normaliza preditores numéricos

# Criando o modelo Elastic-Net
modelo <- linear_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")

# Definição da malha de hiperparâmetros
grid_parametros <- grid_regular(
  penalty(),
  mixture(range = c(0.1, 0.9)),
  levels = 4
)

# Criando os conjuntos de treinamento e validação
set.seed(123)
vfolds <- vfold_cv(df, v = 10)

# Ajustando os hiperparâmetros com tune_grid()
resultados <- tune_grid(
  modelo,
  receita,
  resamples = vfolds,
  grid = grid_parametros
)

# Analisando os resultados
ggplot(resultados %>% collect_metrics() %>% filter(.metric == "rmse"),
       aes(x = penalty, y = mean, color = as.factor(mixture))) +
  geom_line() +
  geom_point() +
  labs(title = "Erro RMSE por Penalização e Mistura",
       x = "Penalização ( )",
       y = "RMSE",
       color = "Mixture")
```




```r
# Escolhendo os melhores hiperparâmetros
melhores_parametros <- show_best(resultados, metric = "rmse")
print(melhores_parametros)
```

```
## # 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   22709.    10   1040. Prepro~
## 2 0.000000215    0.1   rmse    standard   22709.    10   1040. Prepro~
## 3 0.000464       0.1   rmse    standard   22709.    10   1040. Prepro~
## 4 1              0.1   rmse    standard   22709.    10   1040. Prepro~
## 5 0.0000000001   0.367 rmse    standard   22712.    10   1040. Prepro~
```

```r
# Ajustando o modelo final
melhor_penalty <- melhores_parametros$penalty[1]
melhor_mixture <- melhores_parametros$mixture[1]

modelo_final <- linear_reg(penalty = melhor_penalty, mixture = melhor_mixture) %>%
  set_engine("glmnet") %>%
  fit(salary ~ ., data = df)

# Analisando os coeficientes
coeficientes <- modelo_final %>%
  pluck("fit") %>%
  coef(s = melhor_penalty)
```

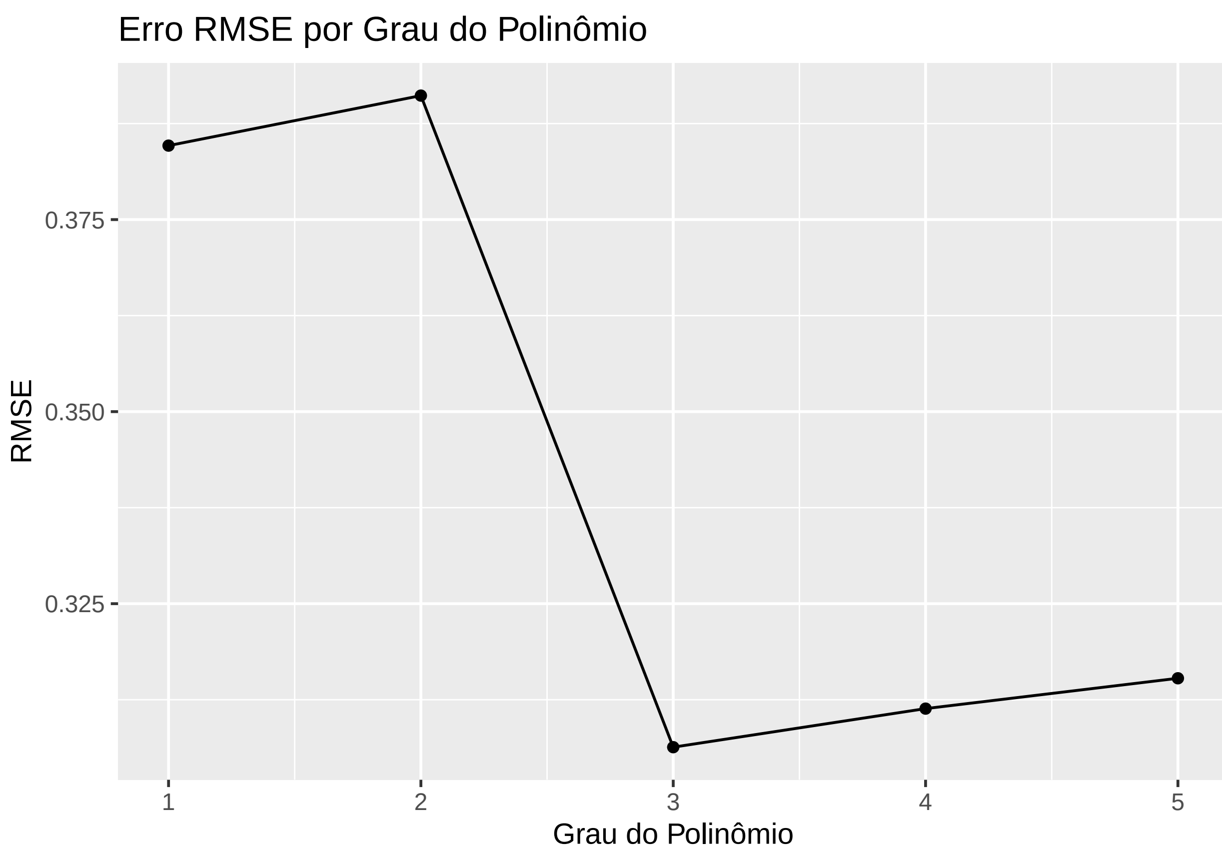
```r
print(coeficientes)
```

```
## 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
```

Solução Exercício 4

```r
# Importação das bibliotecas
# library(splines)
# library(tidymodels)  # Conjunto de pacotes para ML
# library(tidyverse)   # Manipulação de dados

# Gerando o banco de dados artificial
df <- tibble(
  x = runif(100, -1, 1),
  y = 2 * x^3 + x + 10 + rnorm(100, 0, 0.3)
)

# Criando a receita para regressão polinomial
receita <- recipe(y ~ ., data = df) %>%
  step_poly(x, degree = tune())  # Variação do grau do polinômio

# Definição do modelo de regressão linear
modelo <- linear_reg() %>%
  set_engine("lm")

# Criando a malha de hiperparâmetros para o grau do polinômio
grid_parametros <- grid_regular(
  degree(range = c(1, 5)),  # Testa graus de 1 a 5
  levels = 5
)

# Criando os conjuntos de treinamento e validação
set.seed(123)
vfolds <- vfold_cv(df, v = 10)

# Ajustando os hiperparâmetros com tune_grid()
resultados <- tune_grid(
  modelo,
  receita,
  resamples = vfolds,
```

```r
  grid = grid_parametros
)

# Analisando os resultados
ggplot(resultados %>% collect_metrics() %>% filter(.metric == "rmse"),
       aes(x = degree, y = mean)) +
  geom_line() +
  geom_point() +
  labs(title = "Erro RMSE por Grau do Polinômio",
       x = "Grau do Polinômio",
       y = "RMSE")
```




```r
# Escolhendo o melhor grau do polinômio
melhor_grau <- show_best(resultados, metric = "rmse")$degree[1]

# Ajustando o modelo final
modelo_final <- linear_reg() %>%
  set_engine("lm") %>%
  fit(y ~ poly(x, melhor_grau), data = df)

# Exibir coeficientes do modelo final
print(coef(modelo_final$fit))
```

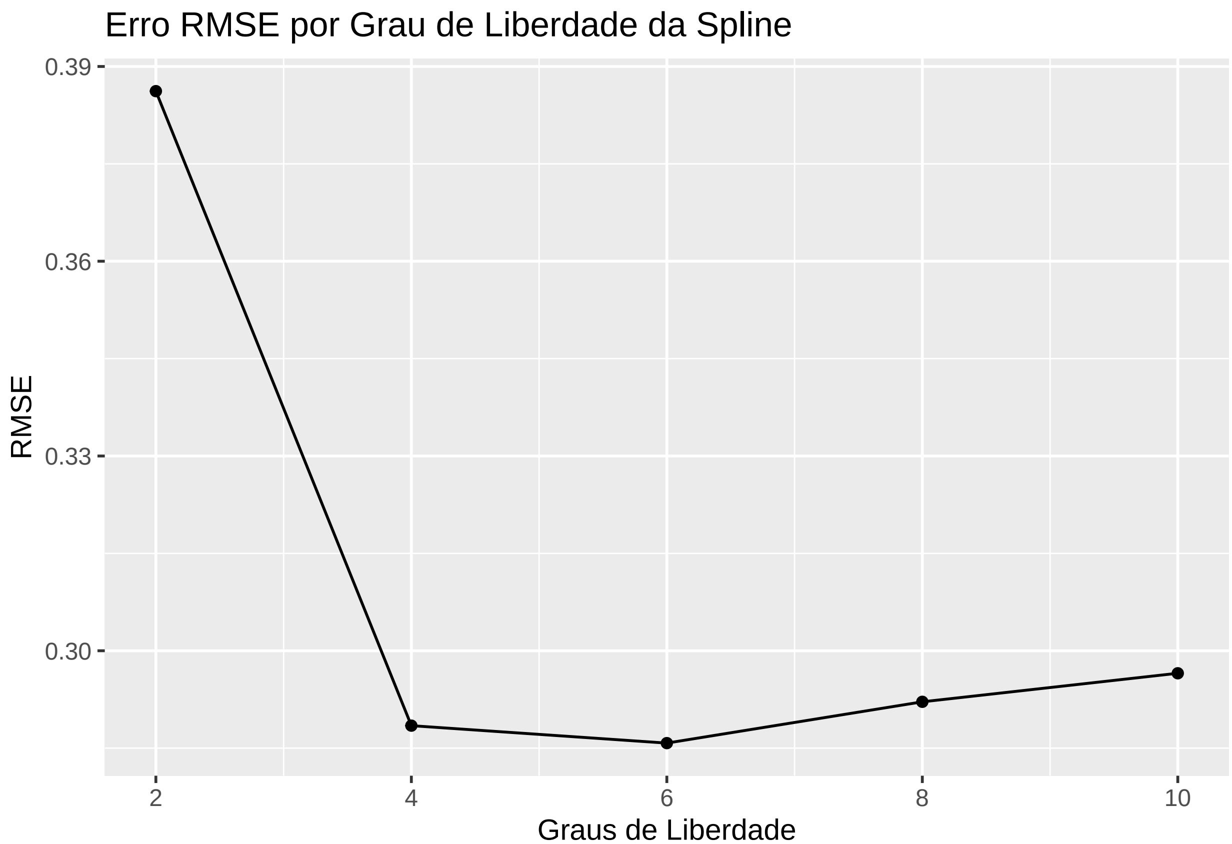
```
##          (Intercept) poly(x, melhor_grau)1 poly(x, melhor_grau)2
```

```
##             10.003012             12.416752              0.309988
## poly(x, melhor_grau)3
##              2.564541
```

Solução Exercício 5

```r
# Importação das bibliotecas
# library(splines)
# library(tidymodels)  # Conjunto de pacotes para ML
# library(tidyverse)   # Manipulação de dados

# Gerando o banco de dados artificial
df <- tibble(
  x = runif(100, -1, 1),
  y = 2 * x^3 + x + 10 + rnorm(100, 0, 0.3)
)

# Criando a receita para regressão com splines
receita <- recipe(y ~ ., data = df) %>%
  step_ns(x, deg_free = tune())  # Variação dos graus de liberdade

# Definição do modelo de regressão linear
modelo <- linear_reg() %>%
  set_engine("lm")

# Criando a malha de hiperparâmetros para os graus de liberdade das splines
grid_parametros <- grid_regular(
  deg_free(range = c(2, 10)),  # Testa graus de liberdade de 2 a 10
  levels = 5
)

# Criando os conjuntos de treinamento e validação
set.seed(123)
vfolds <- vfold_cv(df, v = 10)

# Ajustando os hiperparâmetros com tune_grid()
resultados <- tune_grid(
  modelo,
  receita,
  resamples = vfolds,
  grid = grid_parametros
)

# Analisando os resultados
ggplot(resultados %>% collect_metrics() %>% filter(.metric == "rmse"),
       aes(x = deg_free, y = mean)) +
  geom_line() +
  geom_point() +
  labs(title = "Erro RMSE por Grau de Liberdade da Spline",
```

```
       x = "Graus de Liberdade",
       y = "RMSE")
```




```
# Escolhendo o melhor grau de liberdade
melhor_grau <- show_best(resultados, metric = "rmse")$deg_free[1]

# Ajustando o modelo final
modelo_final <- linear_reg() %>%
  set_engine("lm") %>%
  fit(y ~ ns(x, df = melhor_grau), data = df)

# Exibir coeficientes do modelo final
print(coef(modelo_final$fit))
```

```
##              (Intercept) ns(x, df = melhor_grau)1
##                 7.217625                 2.402402
## ns(x, df = melhor_grau)2 ns(x, df = melhor_grau)3
##                 2.696511                 3.222665
## ns(x, df = melhor_grau)4 ns(x, df = melhor_grau)5
##                 3.183612                 6.256321
## ns(x, df = melhor_grau)6
##                 4.663246
```

Solução Exercício 6

Objetivo

demonstrar que a função $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - \xi)_+^3$ é uma **spline cúbica de regressão** com um nó em ξ.

Dado que, **Uma spline cúbica é uma função polinomial por partes, na qual cada segmento é representado por um polinômio cúbico. Essa função é contínua em todos os seus pontos, incluindo os nós (pontos de junção entre os segmentos), e também possui derivadas primeira e segunda contínuas nesses nós, garantindo suavidade e consistência ao longo de toda a curva.**

A função é definida como:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - \xi)_+^3,$$

em que $(x - \xi)_+^3 = (x - \xi)^3$ se $x > \xi$ e 0 caso contrário.

A) Encontrar $f_1(x)$ para $x \leq \xi$

Para $x \leq \xi$, a função $(x - \xi)_+^3 = 0$. Portanto, a função $f(x)$ se reduz a:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3.$$

Assim, o polinômio $f_1(x)$ é:

$$f_1(x) = a_1 + b_1 x + c_1 x^2 + d_1 x^3,$$

onde:

$$a_1 = \beta_0, \quad b_1 = \beta_1, \quad c_1 = \beta_2, \quad d_1 = \beta_3.$$

B) Encontrar $f_2(x)$ para $x > \xi$

Para $x > \xi$, a função $(x - \xi)_+^3 = (x - \xi)^3$. Portanto, a função $f(x)$ se torna:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - \xi)^3.$$

Expandindo $(x - \xi)^3$:

$$(x - \xi)^3 = x^3 - 3\xi x^2 + 3\xi^2 x - \xi^3.$$

Substituindo na expressão de $f(x)$:

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x^3 - 3\xi x^2 + 3\xi^2 x - \xi^3).$$

Agrupando os termos:

$$f(x) = (\beta_0 - \beta_4 \xi^3) + (\beta_1 + 3\beta_4 \xi^2) x + (\beta_2 - 3\beta_4 \xi) x^2 + (\beta_3 + \beta_4) x^3.$$

Assim, o polinômio $f_2(x)$ é:

$$f_2(x) = a_2 + b_2 x + c_2 x^2 + d_2 x^3,$$

em que:

$$a_2 = \beta_0 - \beta_4 \xi^3, \quad b_2 = \beta_1 + 3\beta_4 \xi^2, \quad c_2 = \beta_2 - 3\beta_4 \xi, \quad d_2 = \beta_3 + \beta_4.$$

C) Continuidade de $f(x)$ em ξ

Para mostrar que $f(x)$ é contínua em ξ, verificamos que:

$$f_1(\xi) = f_2(\xi).$$

Calculando $f_1(\xi)$:

$$f_1(\xi) = \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3.$$

Calculando $f_2(\xi)$:

$$f_2(\xi) = (\beta_0 - \beta_4 \xi^3) + (\beta_1 + 3\beta_4 \xi^2)\xi + (\beta_2 - 3\beta_4 \xi)\xi^2 + (\beta_3 + \beta_4)\xi^3.$$

Simplificando $f_2(\xi)$:

$$f_2(\xi) = \beta_0 - \beta_4 \xi^3 + \beta_1 \xi + 3\beta_4 \xi^3 + \beta_2 \xi^2 - 3\beta_4 \xi^3 + \beta_3 \xi^3 + \beta_4 \xi^3.$$

Os termos $-\beta_4 \xi^3$, $3\beta_4 \xi^3$, $-3\beta_4 \xi^3$ e $\beta_4 \xi^3$ se cancelam, resultando em:

$$f_2(\xi) = \beta_0 + \beta_1 \xi + \beta_2 \xi^2 + \beta_3 \xi^3.$$

Portanto, $f_1(\xi) = f_2(\xi)$, e $f(x)$ é contínua em ξ.

D) Continuidade da Primeira Derivada em ξ

A primeira derivada de $f_1(x)$ é:

$$f_1'(x) = \beta_1 + 2\beta_2 x + 3\beta_3 x^2.$$

A primeira derivada de $f_2(x)$ é:

$$f_2'(x) = (\beta_1 + 3\beta_4 \xi^2) + 2(\beta_2 - 3\beta_4 \xi)x + 3(\beta_3 + \beta_4)x^2.$$

Avaliando em $x = \xi$:

$$f_1'(\xi) = \beta_1 + 2\beta_2 \xi + 3\beta_3 \xi^2,$$

$$f_2'(\xi) = (\beta_1 + 3\beta_4 \xi^2) + 2(\beta_2 - 3\beta_4 \xi)\xi + 3(\beta_3 + \beta_4)\xi^2.$$

Simplificando $f_2'(\xi)$:

$$f_2'(\xi) = \beta_1 + 3\beta_4\xi^2 + 2\beta_2\xi - 6\beta_4\xi^2 + 3\beta_3\xi^2 + 3\beta_4\xi^2.$$

Os termos $3\beta_4\xi^2$, $-6\beta_4\xi^2$ e $3\beta_4\xi^2$ se cancelam, resultando em:

$$f_2'(\xi) = \beta_1 + 2\beta_2\xi + 3\beta_3\xi^2.$$

Portanto, $f_1'(\xi) = f_2'(\xi)$, e a primeira derivada é contínua em ξ.

E) Continuidade da Segunda Derivada em ξ

A segunda derivada de $f_1(x)$ é:

$$f_1''(x) = 2\beta_2 + 6\beta_3 x.$$

A segunda derivada de $f_2(x)$ é:

$$f_2''(x) = 2(\beta_2 - 3\beta_4\xi) + 6(\beta_3 + \beta_4)x.$$

Avaliando em $x = \xi$:

$$f_1''(\xi) = 2\beta_2 + 6\beta_3\xi,$$

$$f_2''(\xi) = 2(\beta_2 - 3\beta_4\xi) + 6(\beta_3 + \beta_4)\xi.$$

Simplificando $f_2''(\xi)$:

$$f_2''(\xi) = 2\beta_2 - 6\beta_4\xi + 6\beta_3\xi + 6\beta_4\xi.$$

Os termos $-6\beta_4\xi$ e $6\beta_4\xi$ se cancelam, resultando em:

$$f_2''(\xi) = 2\beta_2 + 6\beta_3\xi.$$

Portanto, $f_1''(\xi) = f_2''(\xi)$, e a segunda derivada é contínua em ξ.
