Machine Learning Model

Muhammad Aryaan Lakhani

2025-06-24

pacman::p_load(AER, fixest, modelsummary, kableExtra, tidyverse, rsample, # new package for sample spliting glmnet, # new package for lasso broom, # new package for model extraction tidymodels # tools for ML in tidyverse syntax
)

```
library(AER)
```

```
## Loading required package: car
```

```
## Loading required package: carData
```

```
## Loading required package: lmtest
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: sandwich
```

```
## Loading required package: survival
```

```
library(fixest)
library(modelsummary)
library(kableExtra)
library(tidyverse)
```

```
## ── Attaching packages
## ───────────────────────────────────────
## tidyverse 1.3.2 ──
## ✔ ggplot2 3.4.0     ✔ purrr   1.0.1
## ✔ tibble  3.2.1     ✔ dplyr   1.1.0
## ✔ tidyr   1.2.1     ✔ stringr 1.5.0
## ✔ readr   2.1.3     ✔ forcats 0.5.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter()     masks stats::filter()
## ✖ dplyr::group_rows() masks kableExtra::group_rows()
## ✖ dplyr::lag()        masks stats::lag()
## ✖ dplyr::recode()     masks car::recode()
## ✖ purrr::some()       masks car::some()
```

```
library(rsample)
library(glmnet)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-7
```

```
library(broom)
library(tidymodels)
```

```
## ── Attaching packages ────────────────────────────────────── tidymodels 1.0.0 ──
## ✔ dials        1.1.0     ✔ tune         1.0.1
## ✔ infer        1.0.4     ✔ workflows    1.1.3
## ✔ modeldata    1.1.0     ✔ workflowsets 1.0.0
## ✔ parsnip      1.0.4     ✔ yardstick    1.1.0
## ✔ recipes      1.0.5
## ── Conflicts ───────────────────────────────────────── tidymodels_conflicts() ──
## ✖ scales::discard()   masks purrr::discard()
## ✖ Matrix::expand()    masks tidyr::expand()
## ✖ dplyr::filter()     masks stats::filter()
## ✖ recipes::fixed()    masks stringr::fixed()
## ✖ dplyr::group_rows() masks kableExtra::group_rows()
## ✖ dplyr::lag()        masks stats::lag()
## ✖ Matrix::pack()      masks tidyr::pack()
## ✖ dplyr::recode()     masks car::recode()
## ✖ purrr::some()       masks car::some()
## ✖ yardstick::spec()   masks readr::spec()
## ✖ recipes::step()     masks stats::step()
## ✖ Matrix::unpack()    masks tidyr::unpack()
## ✖ recipes::update()   masks Matrix::update(), stats::update()
## • Search for functions across packages at https://www.tidymodels.org/find/
```

```
df <- read_csv("/Users/aryaanlakhani/Desktop//Hitters.csv") %>%

# transforming characters to factors
  mutate(across(where(is.character),~factor(.x)))
```

```
## Rows: 263 Columns: 20
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (3): League, Division, NewLeague
## dbl (17): AtBat, Hits, HmRun, Runs, RBI, Walks, Years, CAtBat, CHits, CHmRun...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Data set splitting

```
# setting random seed for replication
set.seed(1999)

# splitting data into training/test
df_split<- initial_split(df, prop = 3/4)

# Creating data
test  <- testing(df_split)
train <- training(df_split)
```

Models and Errors

```
models <- list(
  Basic    = lm(Salary ~ Years + RBI + HmRun + CRBI + CHmRun, data = train),
  Full     = lm(Salary ~ ., data = train),
  Extended = lm(Salary ~ (. - Division- League - NewLeague)^2 +Division + League + NewLe
ague, data = train)
)


coef(models$Basic) %>% length()
```

```
## [1] 6
```

```
coef(models$Full) %>% length()
```

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

```
coef(models$Extended) %>% length()
```

```
## [1] 140
```

```
models$Basic$rank
```

```
## [1] 6
```

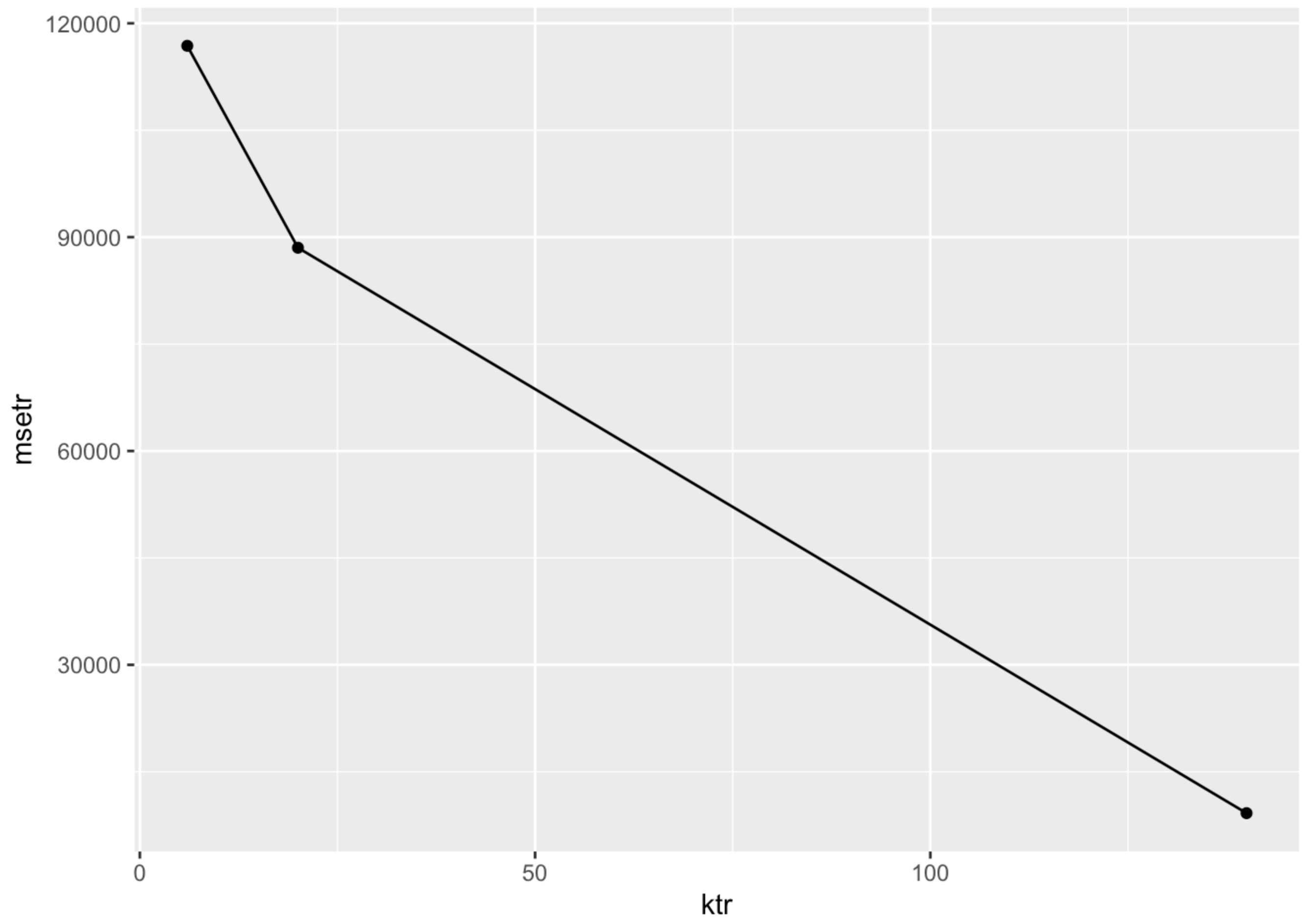
```
models$Full$rank
```

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

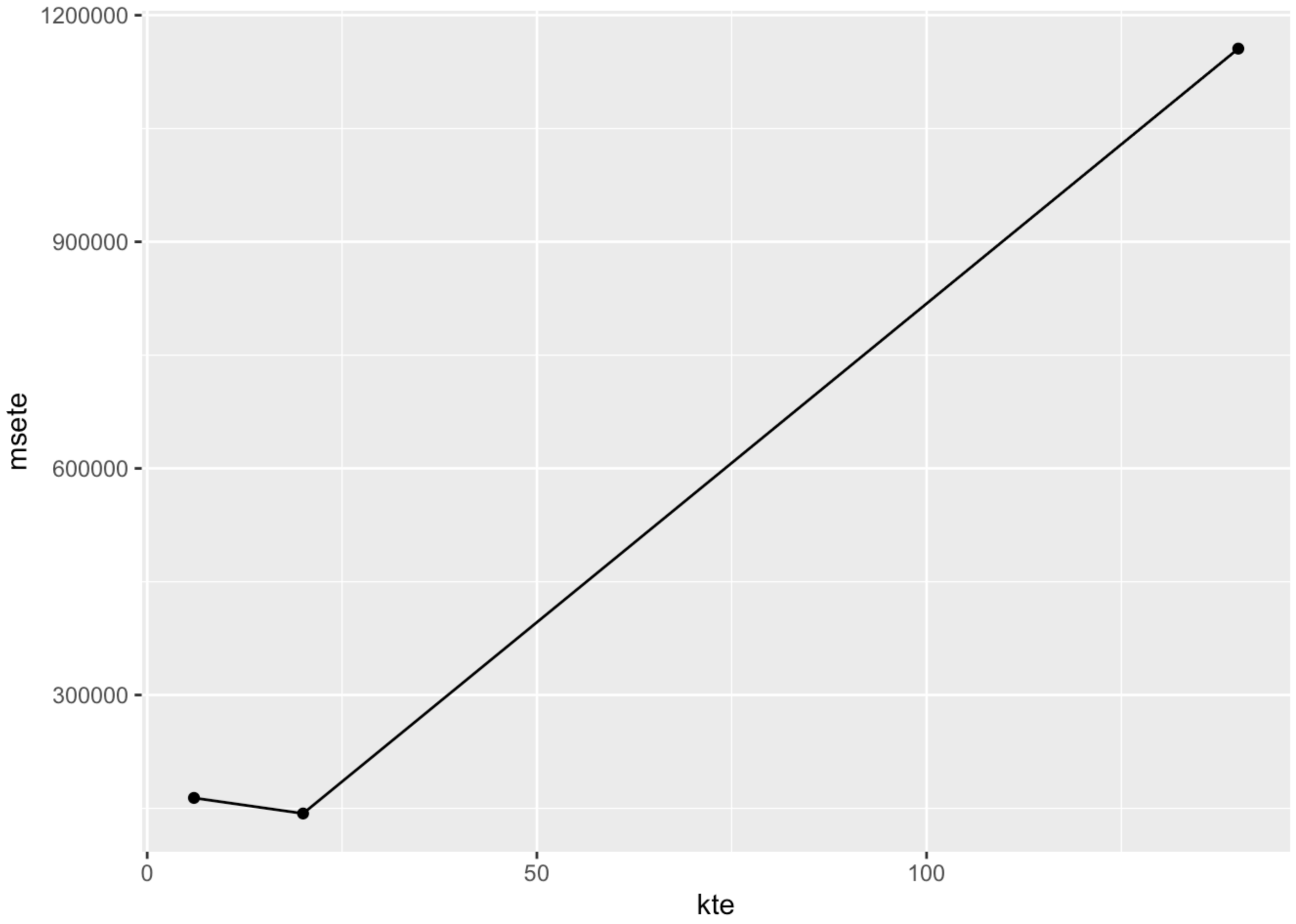
```
models$Extended$rank
```

```
## [1] 140
```

```
# Calculating training errors for all models.
train.mse.Basic <- mean((train$Salary - predict(models$Basic))^2)
train.mse.Full <-  mean((train$Salary - predict(models$Full))^2)
train.mse.Extended <- mean((train$Salary - predict(models$Extended))^2)

# Calculating test errors for all models.
test.mse.Basic <- mean((test$Salary - predict(models$Basic, newdata = test))^2)
test.mse.Full <-  mean((test$Salary - predict(models$Full, newdata = test))^2)
test.mse.Extended <- mean((test$Salary - predict(models$Extended, newdata = test))^2)

mse.train <- tibble(
  msetr = c(train.mse.Basic, train.mse.Full, train.mse.Extended),
  ktr   = c(models$Basic$rank, models$Full$rank, models$Extended$rank)
)

mse.test <- tibble(
  msete = c(test.mse.Basic, test.mse.Full, test.mse.Extended),
  kte   = c(models$Basic$rank, models$Full$rank, models$Extended$rank)
)

ggplot(mse.train, aes(y = msetr , x = ktr)) +
  geom_line()  +
  geom_point()
```



```
ggplot(mse.test, aes(y = msete , x = kte)) +
  geom_line()  +
  geom_point()
```

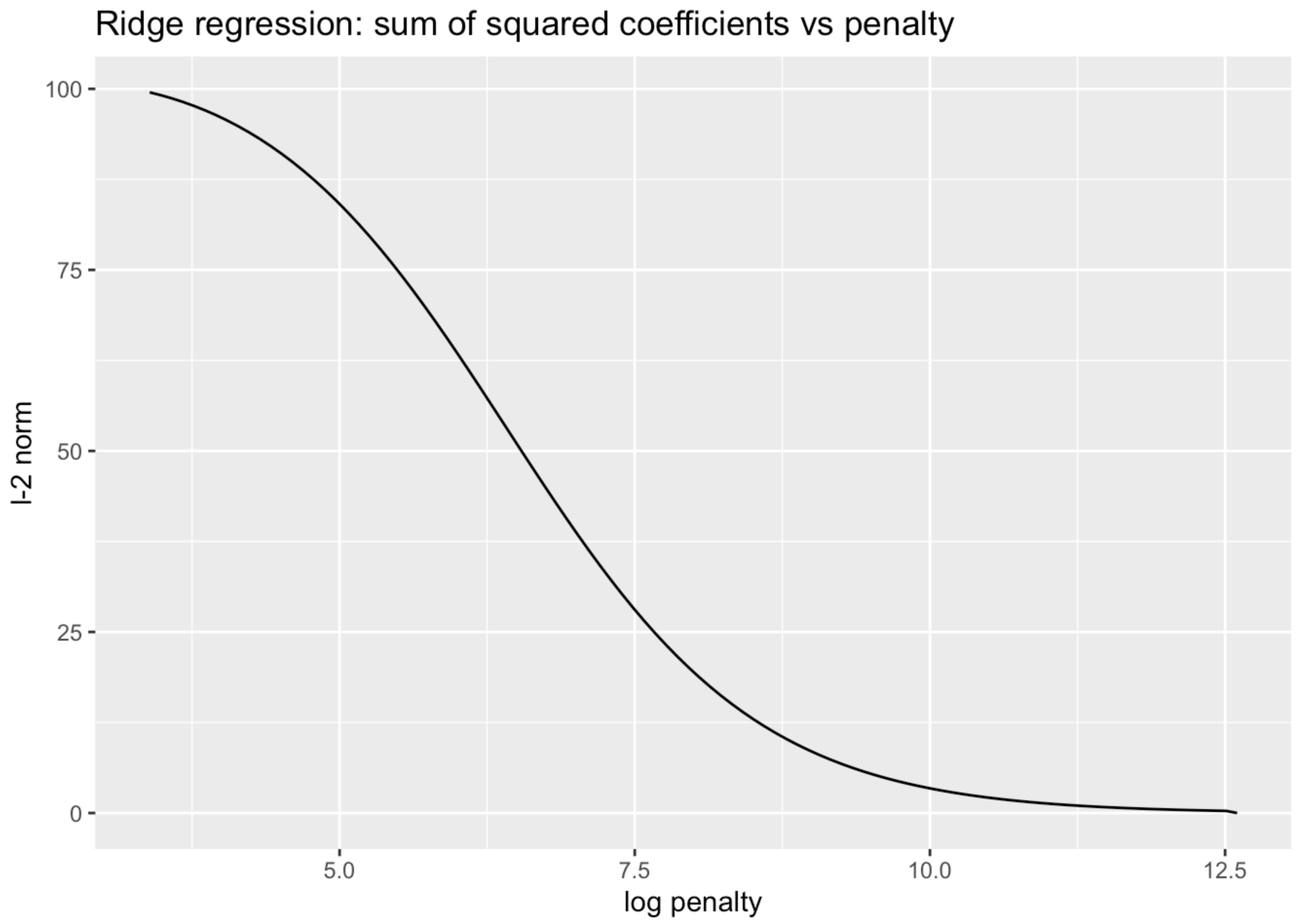


```
# Looking at the training dataset, the Basic Model with 6 parameters has the highest MS
E, indicating the least accurate fit. The Full Model shows some improvement, with a lowe
r MSE than the Basic Model, but it still doesn’t outperform the Extended Model. The Exte
nded Model, with 140 parameters, achieves the lowest MSE, suggesting it fits the trainin
g data best and performs the strongest within this dataset.

# Examining the models on the test dataset, we observe that the Basic Model performs bet
ter here than it did on the training set, as indicated by its lower MSE. In contrast, th
e Extended Model, which had the lowest MSE during training, shows the highest MSE on the
test data—suggesting it may have overfit the training set. The best-performing model on
the test data is the Full Model with 20 parameters, achieving the lowest MSE among all t
he models.
```

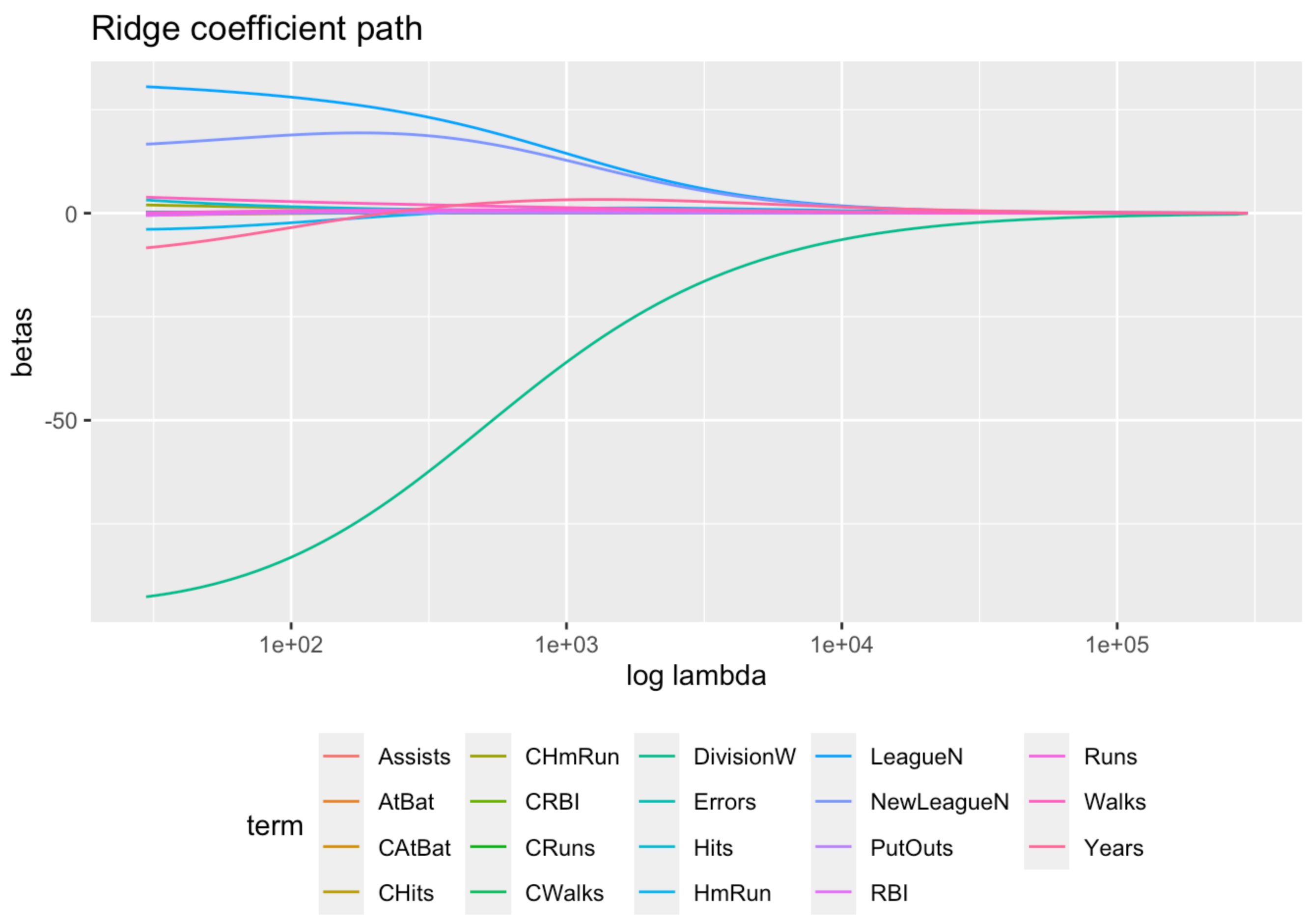
Demonstration

```
### Train data

# Extracting Salary as a vector
Y_train <- train$Salary

# Extracting X as matrix
X_train <-  model.matrix(Salary ~ ., data = train)[, -1]

### Testing data

# Extracting Salary as a vector
Y_test <- test$Salary

# Extracting X as matrix
X_test <- model.matrix(Salary ~  ., data = test)[, -1]



# Ridge
ridge <- glmnet(x = X_train,
                y = Y_train,
                alpha=0)

ridge.data <- tidy(ridge)
```

```
# this code chunk is complete, setting eval = T
ridge.data %>%
  filter(term != "(Intercept)") %>%
  group_by(step) %>%
  summarize(sum.coef = sqrt(sum(estimate^2)) ,
            lambda = mean(lambda)) %>%
  ggplot(aes(y = sum.coef, x = log(lambda))) +
  geom_line() +
  labs(
    y = "l-2 norm",
    x = "log penalty",
    title = "Ridge regression: sum of squared coefficients vs penalty"
  )
```



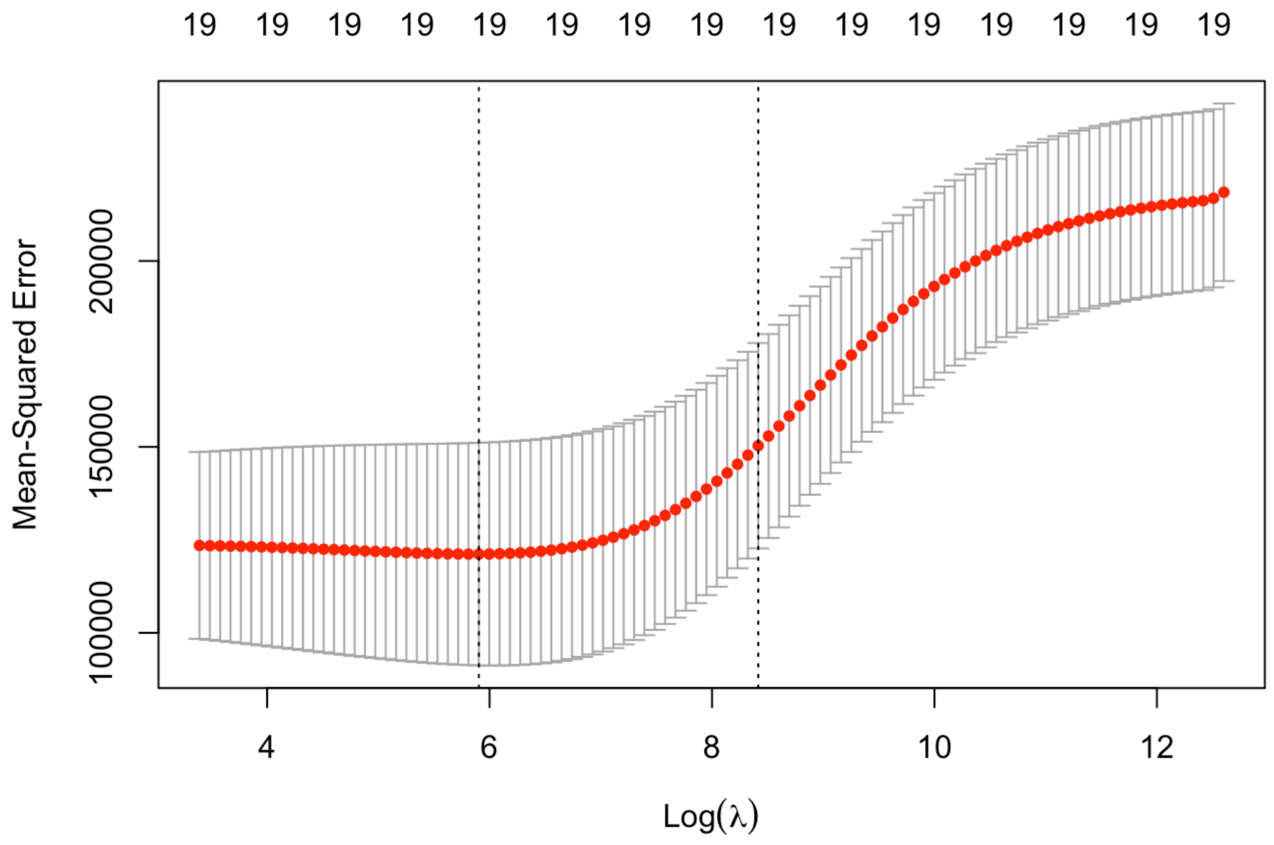
```
ridge.data %>%
  filter(term != "(Intercept)") %>%
ggplot(aes(lambda, estimate, color = term)) +
  geom_line() +
  scale_x_log10() +
  labs(
    title = "Ridge coefficient path",
    y = "betas",
    x = "log lambda"
  ) +
  theme(legend.position = 'bottom')
```



```
cv.ridge <- cv.glmnet(
  y = Y_train,
  x = X_train,
  nfolds = 5,
  alpha = 0)

# output of optimal penalty
cv.ridge
```

```
##
## Call:  cv.glmnet(x = X_train, y = Y_train, nfolds = 5, alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure    SE Nonzero
## min    366    73  121172 29937      19
## 1se   4517    46  150316 27604      19
```

```
# plot of mse vs penalty
plot(cv.ridge)
```



```
# this code chunk is complete, set eval = T
cv.ridge$lambda.min
```

```
## [1] 366.3678
```

```
## [1] 30.7
```

```
# Note the syntax here.
ridge.mse <- mean((Y_test - predict(cv.ridge, s = cv.ridge$lambda.min, newx = X_test))^
2)
ridge.mse
```

```
## [1] 116412.2
## [1] 129888

# When comparing the mean squared error (MSE) of Ridge regression to that of the other l
inear models on the test data, Ridge regression yields a lower MSE—indicating better gen
eralization—i.e., MSE<sub>Ridge</sub> < MSE<sub>test</sub> for the other models. Howeve
r, on the training data, the Extended Model achieves the lowest MSE, outperforming Ridge
in that context. It’s important to note, though, that these models may be optimizing dif
ferent objectives or using different assumptions, so direct comparisons based solely on
MSE may not fully capture their relative performance.
```

Coding

```
Y_train <- train$Salary


X_train <-  model.matrix(Salary ~ (. - Division- League - NewLeague)^2 +Division + Leagu
e + NewLeague, data = train)[, -1]

### Test data


Y_test <- test$Salary


X_test <- model.matrix(Salary ~ (. - Division- League - NewLeague)^2 +Division + League
+ NewLeague, data = test)[, -1]
```

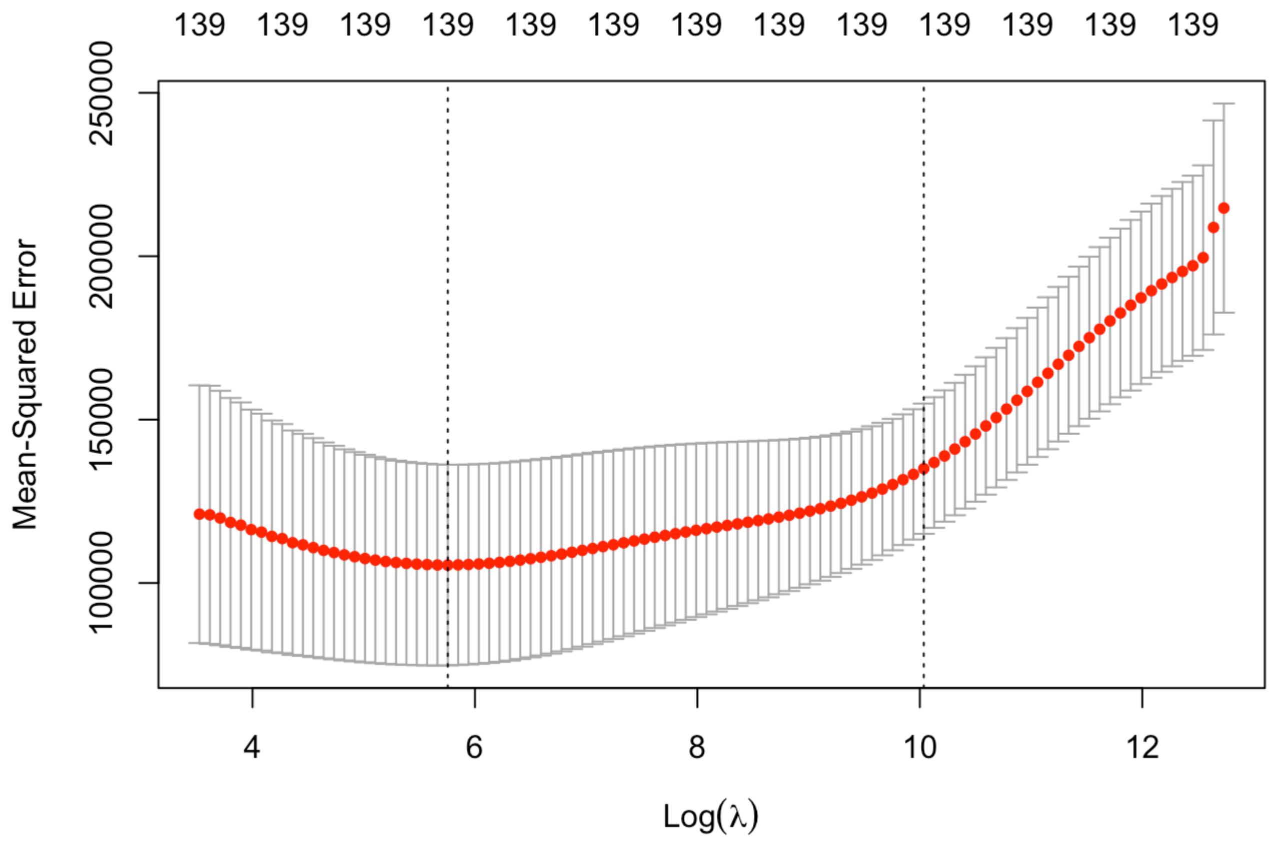
```
cv.ridge <- cv.glmnet(
   y = Y_train,
  x = X_train,
  nfolds = 5,
  alpha = 0)

ridge.mse <- mean((Y_test - predict(cv.ridge, s = cv.ridge$lambda.min, newx = X_test))^
2) %>% mean
ridge.mse
```

```
## [1] 86228.38
```

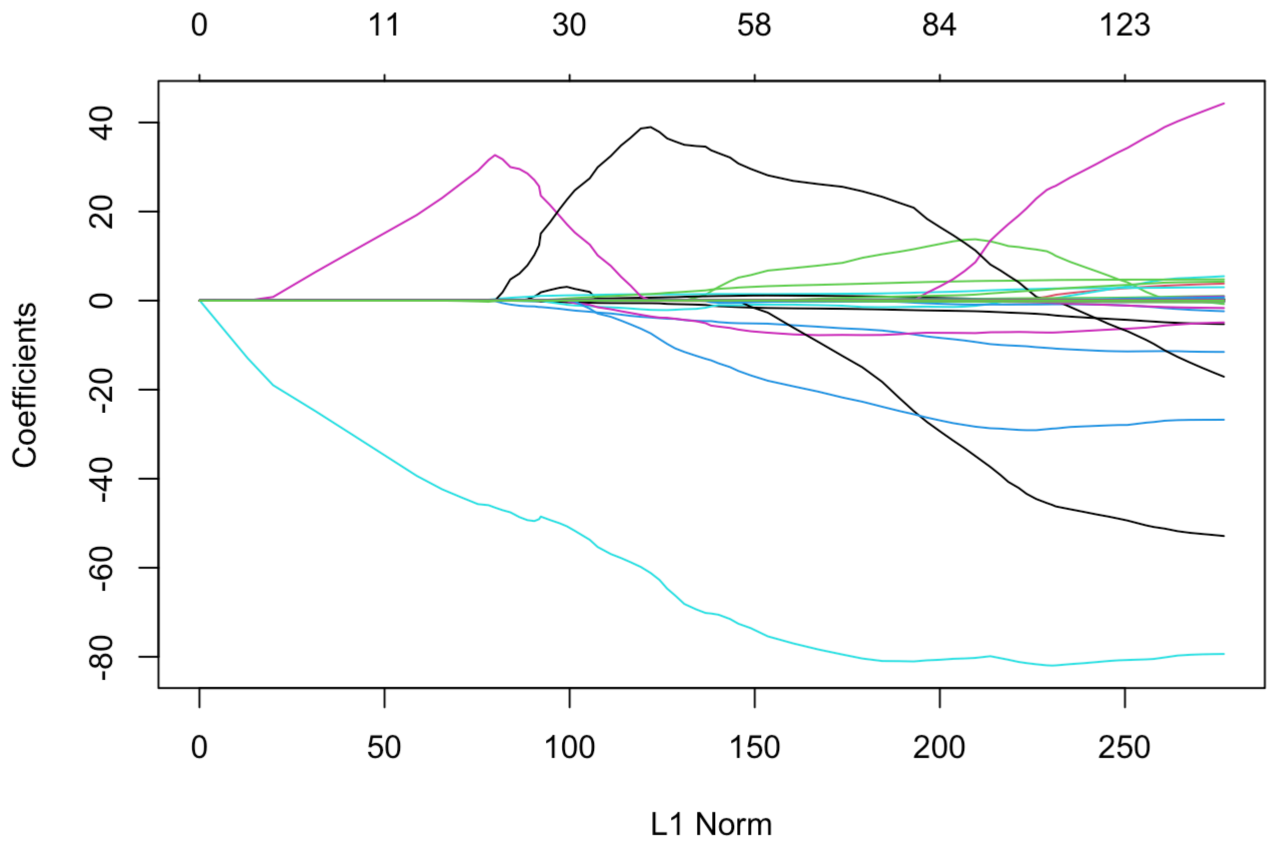
```
cv.ridge$lambda.min
```

```
## [1] 316.0422
```

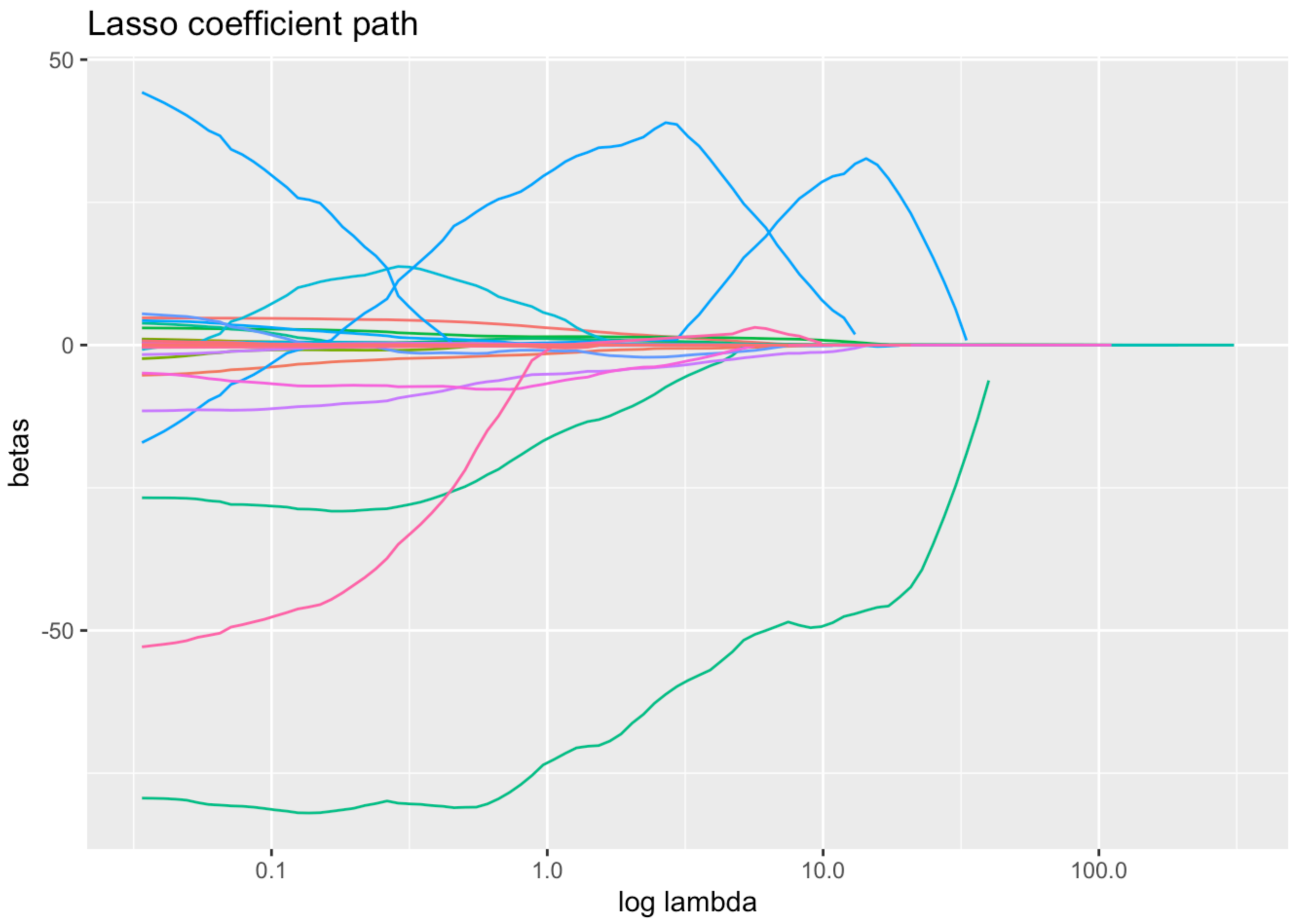
```
plot(cv.ridge)
```



```
# lasso
lasso <- glmnet(y = Y_train,
  x = X_train,
  nfolds = 5,
  alpha = 1)

plot(lasso)
```



```
# for the plot
lasso %>%
  tidy() %>%
  filter(term != "(Intercept)") %>%
ggplot(aes(lambda, estimate, color = term)) +
  geom_line() +
  scale_x_log10() +
  labs(
    title = "Lasso coefficient path",
    y = "betas",
    x = "log lambda"
  ) +
  theme(legend.position = "none")
```

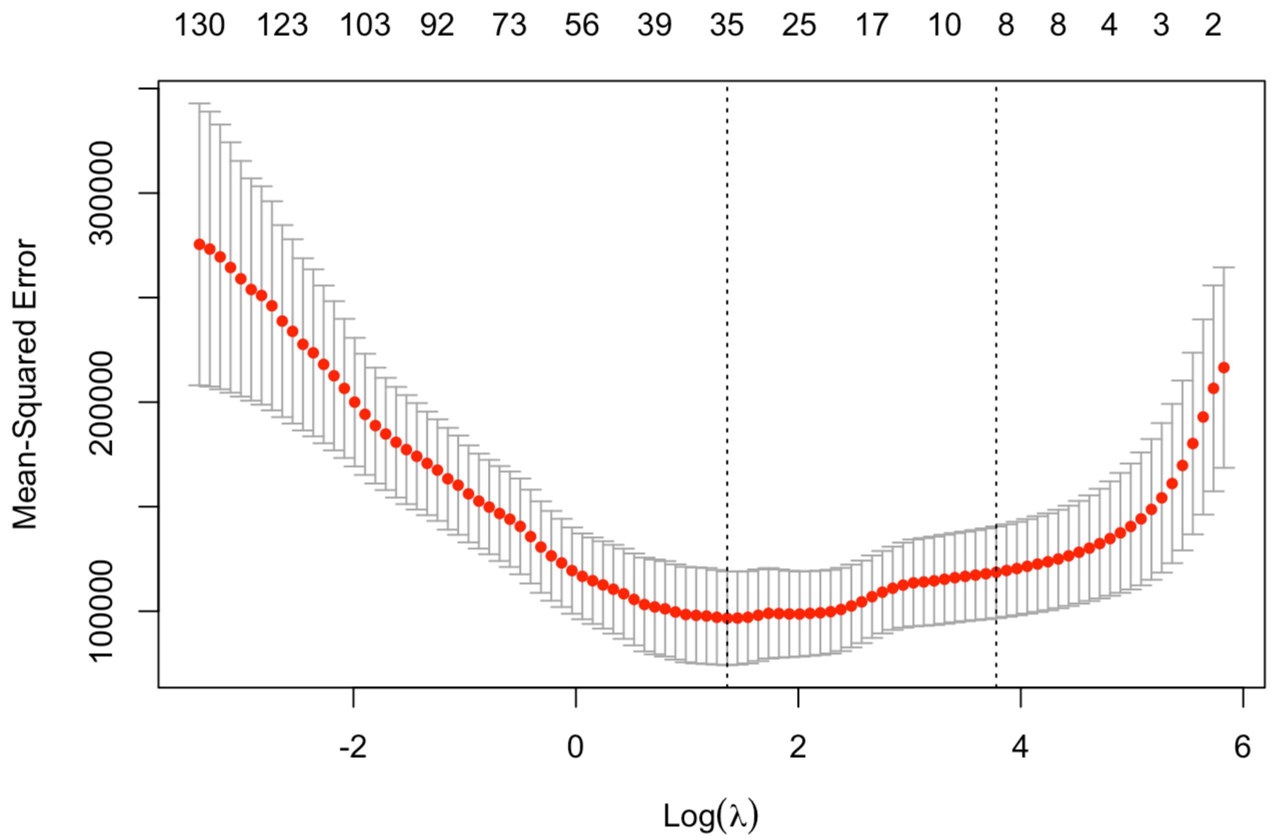


```
#The coefficient path for Lasso regression appears more "spiky" compared to that of Ridg
e regression. This is due to the nature of the Lasso penalty, which is more aggressive i
n driving some coefficients to exactly zero. As the regularization parameter (lambda) in
creases, many coefficients in Lasso are completely eliminated, leaving only a few with n
on-zero values—hence the jagged, abrupt changes in the path. On the other hand, Ridge re
gression applies a softer penalty that continuously shrinks all coefficients toward zero
without eliminating them entirely, resulting in a much smoother and more gradual coeffic
ient path.
```

```
cv.lasso <- cv.glmnet(
y = Y_train,
  x = X_train,
  nfolds = 5,
  alpha = 1)

cv.lasso$lambda.min
```

```
## [1] 3.896315
```

```
plot(cv.lasso)
```



```
lasso.mse <- mean((Y_test - predict(cv.lasso, s = cv.lasso$lambda.min, newx = X_test))^
2) %>% mean
lasso.mse
```

```
## [1] 76997.48
```

```
ols <- glmnet(
  y = Y_train,
  x = X_train,
  lmabda = 0
  )


ols.mse   <- (Y_test - predict(ols, s = 0, newx = X_test))^2 %>% mean

results <- tibble(
  model = c("ols", "lasso", "ridge"),
  mse = c(ols.mse, lasso.mse, ridge.mse)
)

results %>%
  kbl() %>%
  kable_classic_2(full_width = F)
```

model	mse
ols	147384.35
lasso	76997.48
ridge	86228.38

```
# print the results
print(results)
```

```
## # A tibble: 3 × 2
##   model     mse
##   <chr>   <dbl>
## 1 ols   147384.
## 2 lasso  76997.
## 3 ridge  86228.
```

```
# Among the models evaluated, the Ridge regression model achieved the lowest mean square
d error (MSE), indicating the best performance. The Lasso model followed closely behind,
while the Ordinary Least Squares (OLS) model had the highest MSE, making it the least ef
fective in this case. Therefore, Ridge regression outperforms the others in this scenari
o.

# The final lasso model includes only a subset of the original variables. The coefficien
ts for some of the variables are exactly zero, indicating that they have been entirely e
liminated from the model. The variables that are still present in the model are consider
ed important by the lasso regression algorithm. In the output provided, the intercept is
the first coefficient and is equal to 340. The second coefficient corresponds to the var
iable "AtBat" and has a value of -0.595. The third coefficient is for the variable "Hit
s," but its value is "." indicating that it has been set to zero. The same is true for t
he "HmRun", "Runs", "RBI", "CAtBat", "CRBI", "CWalks", "DivisionW", "Errors", "Assists",
and "NewLeagueN" variables. On the other hand, the following variables have non-zero coe
fficients, and thus, they are included in the final model: "Years", "CHits", "CHmRun",
"CRuns", "CWalks", "LeagueA", "LeagueN", "PutOuts", "Hits:HmRun", "Hits:Walks", "Hits:CH
its", "Hits:CRuns", "Hits:CRBI", "Hits:LeagueN", "Hits:DivisionW", "HmRun:LeagueN", "HmR
un:Assists", "HmRun:NewLeagueN", "Runs:CHmRun", "RBI:CRuns", "Walks:CRBI". The lasso reg
ression algorithm suggests that these variables are the most important for predicting th
e response variable in this model. However, we should be careful not to interpret these
coefficients causally.
```