# Data Science for Economics

## Lecture 2: Shrinkage methods

Dawie van Lill

2021/04/05

# Packages

I thought slides would be easier to present than notes.

The code and slides are on the Github page.

Here are the packages that we will be using for this session.

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(dplyr, ggplot2, rsample, caret, glmnet, vip, tidyverse, p
```

Let us give quick rundown of some of the potentially new packages.

**rsample**:

**caret**: Machine learning

**glmnet**:

# Linear regression (one variable)

Quick example of linear regression using machine learning process.

Want to model linear relationship between total above ground living space of a home (`Gr_Liv_Area`) and sale price (`Sale_Price`).

First, we construct our training sample.

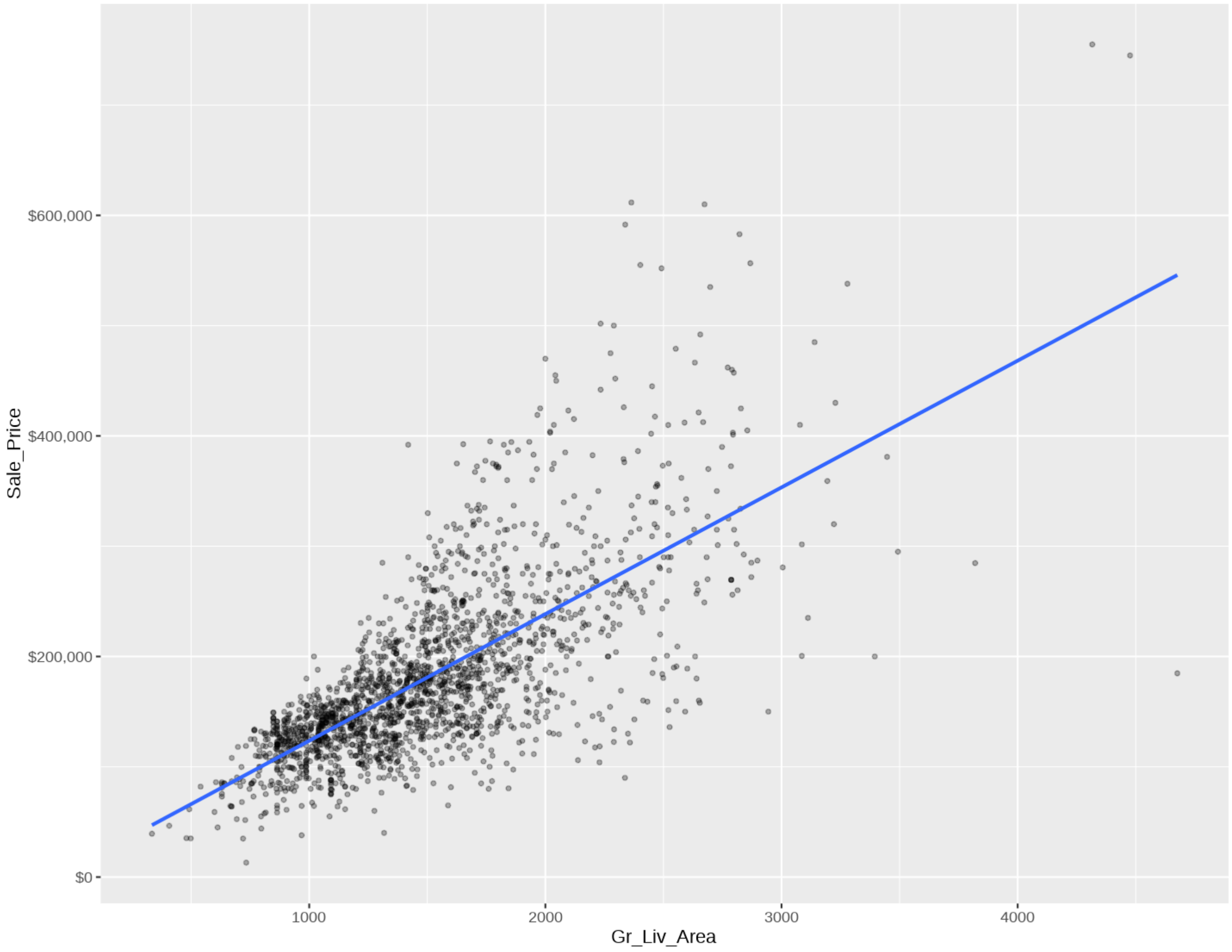
```
set.seed(123)
ames <- AmesHousing::make_ames()
split  <- initial_split(ames, prop = 0.7, strata = "Sale_Price")
ames_train  <- training(split)
ames_test   <- testing(split)
```

Second, we model on training data.

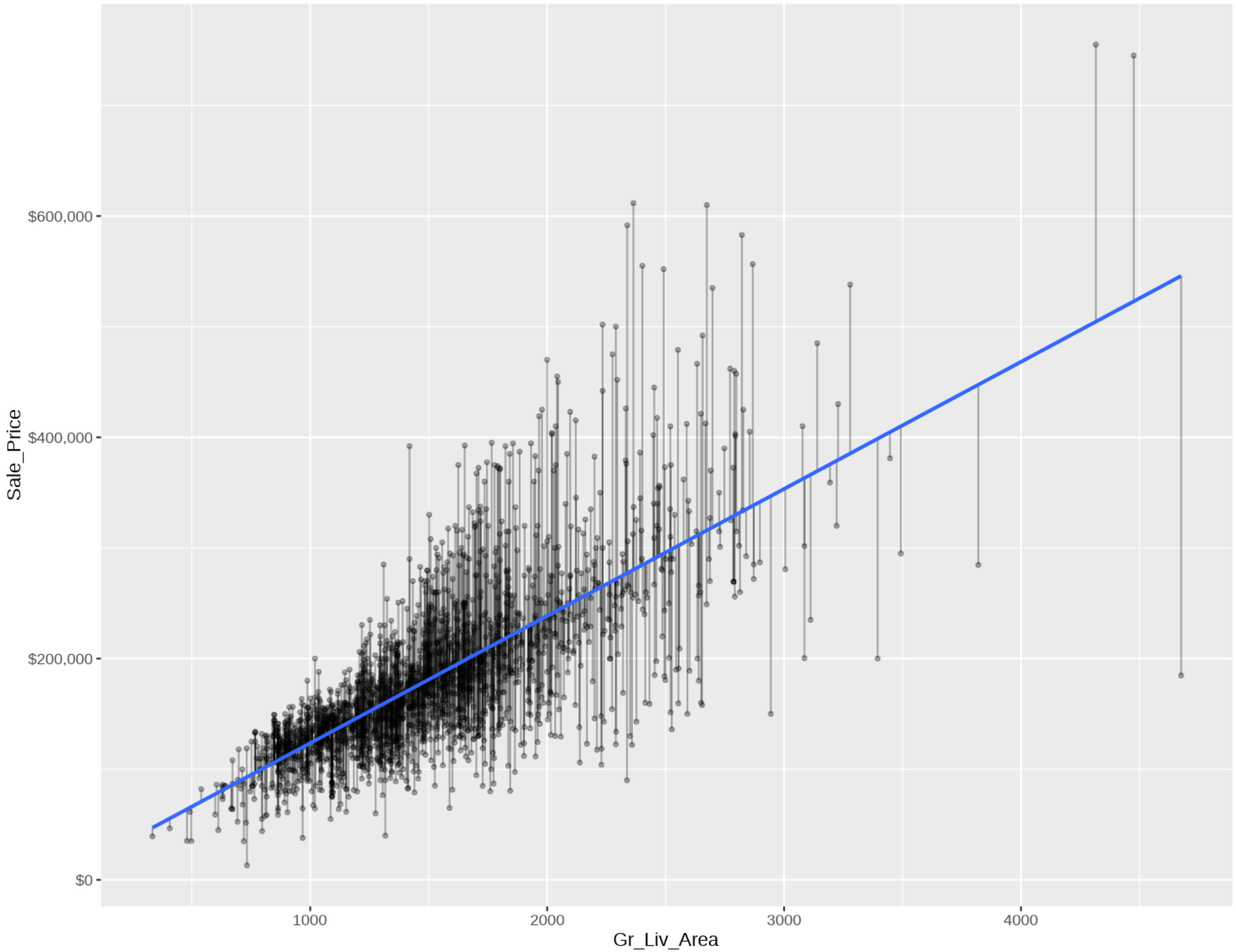
```
model1 <- lm(Sale_Price ~ Gr_Liv_Area, data = ames_train)
```

In the next slide we show the results from this regression.

Fitted regression line



Fitted regression line (with residuals)



# Evaluate results

```
summary(model1)
```

```
##
## Call:
## lm(formula = Sale_Price ~ Gr_Liv_Area, data = ames_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -361143  -30668   -2449   22838  331357
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 8732.938   3996.613   2.185    0.029 *
## Gr_Liv_Area  114.876      2.531  45.385   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 56700 on 2051 degrees of freedom
## Multiple R-squared:  0.5011,    Adjusted R-squared:  0.5008
## F-statistic:  2060 on 1 and 2051 DF,  p-value: < 2.2e-16
```

# Multiple linear regression

You can include more than one predictor, such as above ground square footage and year house was built.

```
model2 <- lm(Sale_Price ~ Gr_Liv_Area + Year_Built, data = ames_train)
```

One could add as many predictors as you want.

```
model3 <- lm(Sale_Price ~ ., data = ames_train)
```

Let us now test the results from the models to see which one is the most accurate.

# Linear regression

```r
set.seed(123)
(cv_model1 <- train(form = Sale_Price ~ Gr_Liv_Area,
  data = ames_train,
  method = "lm",
  trControl = trainControl(method = "cv", number = 10)
))
```

```
## Linear Regression
##
## 2053 samples
##    1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1846, 1848, 1848, 1848, 1848, 1848, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   56410.89  0.5069425  39169.09
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

# Multiple linear regression

```r
set.seed(123)
(cv_model2 <- train(form = Sale_Price ~ Gr_Liv_Area + Year_Built,
  data = ames_train,
  method = "lm",
  trControl = trainControl(method = "cv", number = 10)
))
```

```
## Linear Regression
##
## 2053 samples
##    2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1846, 1848, 1848, 1848, 1848, 1848, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   46292.38  0.6703298  32246.86
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

# Out of sample performance

```
##
## Call:
## summary.resamples(object = resamples(list(model1 = cv_model1, model2
##  = cv_model2)))
##
## Models: model1, model2
## Number of resamples: 10
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## model1 34457.58 36323.74 38943.81 39169.09 41660.81 45005.17    0
## model2 28094.79 30594.47 31959.30 32246.86 34210.70 37441.82    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## model1 47211.34 52363.41 54948.96 56410.89 60672.31 67679.05    0
## model2 37698.17 42607.11 45407.14 46292.38 49668.59 54692.06    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## model1 0.3598237 0.4550791 0.5289068 0.5069425 0.5619841 0.5965793    0
## model2 0.5714665 0.6392504 0.6800818 0.6703298 0.7067458 0.7348562    0
```

# Regularised regression

Data typically have large number of features.

As number of features grow models tend to **overfit the data**.

Regularisation methods provide a means to constrain / regularise the estimated coefficients.

Easiest way to understand is to see how and why it is applied to OLS.

Objective in OLS is to find *hyperplane* that minimises SSE between observed and predicted response values.

This means identifying hyperplane that minimises grey lines in next slide.


$600,000
$400,000
$200,000
Sale_Price
1000
1500
2000
2500
Gr_Liv_Area

# Regularised regression

OLS adheres to some specific assumptions

- Linear relationship
- More observations (n) than features (p) - which means n > p
- No or little multicollinearity

However, for many datasets, such as those involved in text mining, we have more features than observations.

If we have that p > n there are many potential problems

- Model is less interpretable
- Infinite solutions to OLS problem

Make an assumption that small subset of these features exhibit strongest effect

This is called the **sparsity principle**

# Regularised regression

Objective function of regularised regression model includes penalty parameter $P$.

$$\min(\mathrm{SSE} + \mathrm{P})$$

Penalty parameter constrains the size of coefficients.

Coefficients can only increase is if we have decrease in SSE (i.e. the loss function).

Generalises to other linear models as well (such as logistic regression).

Three common penalty parameters

1. Ridge
2. Lasso [**think cowboys!**]
3. Elastic net (combination of ridge and lasso)I

# Ridge regression

Ridge regression controls coefficients by adding $\lambda \sum_{j=1}^{p} \beta_j^2$ to objective function:

$$\min(\text{SSE} + \lambda \sum_{j=1}^{p} \beta_j^2)$$

Size of the penalty is referred to as the $L^2$ norm, or Euclidean norm.

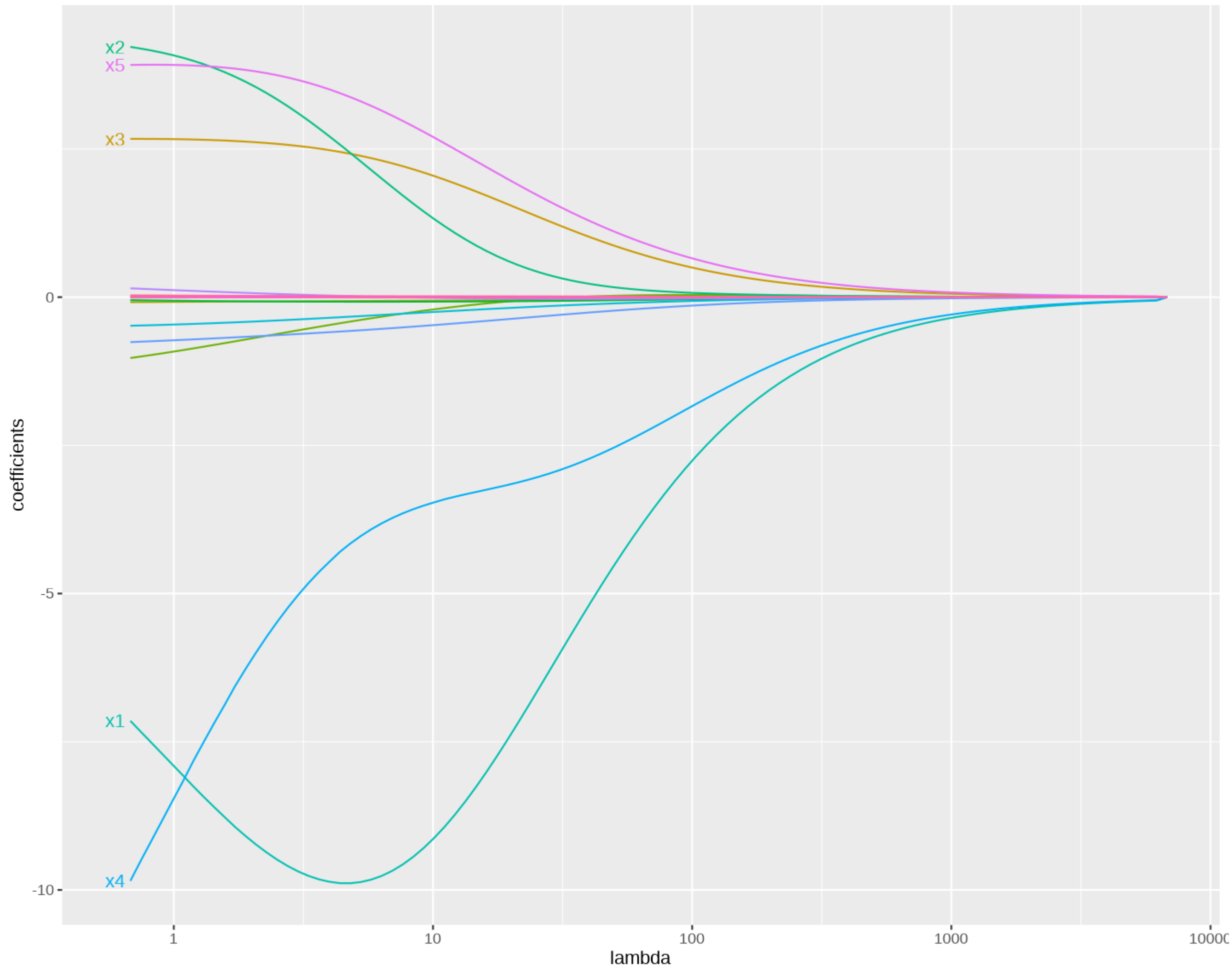
This penalty can take on different values depending on the tuning parameter $\lambda$.

If $\lambda = 0$, then we have OLS regression.

As $\lambda \to \infty$ the penalty becomes large and forces coefficients to zero.

**Important to note**: Does not force all the way to zero, only asymptotically. Does not perform feature selection.

The following slide illustrates dynamics for ridge regression coefficient values as the tuning parameter approaches zero.


x2
x5
x3
x1
x4
0
-5
-10
coefficients
1
10
100
1000
10000
lambda

# Lasso

Similar to ridge penalty. Swap out $L^2$ norm for $L^1$ norm: $\lambda \sum_{j=1}^{p} |\beta_j|$

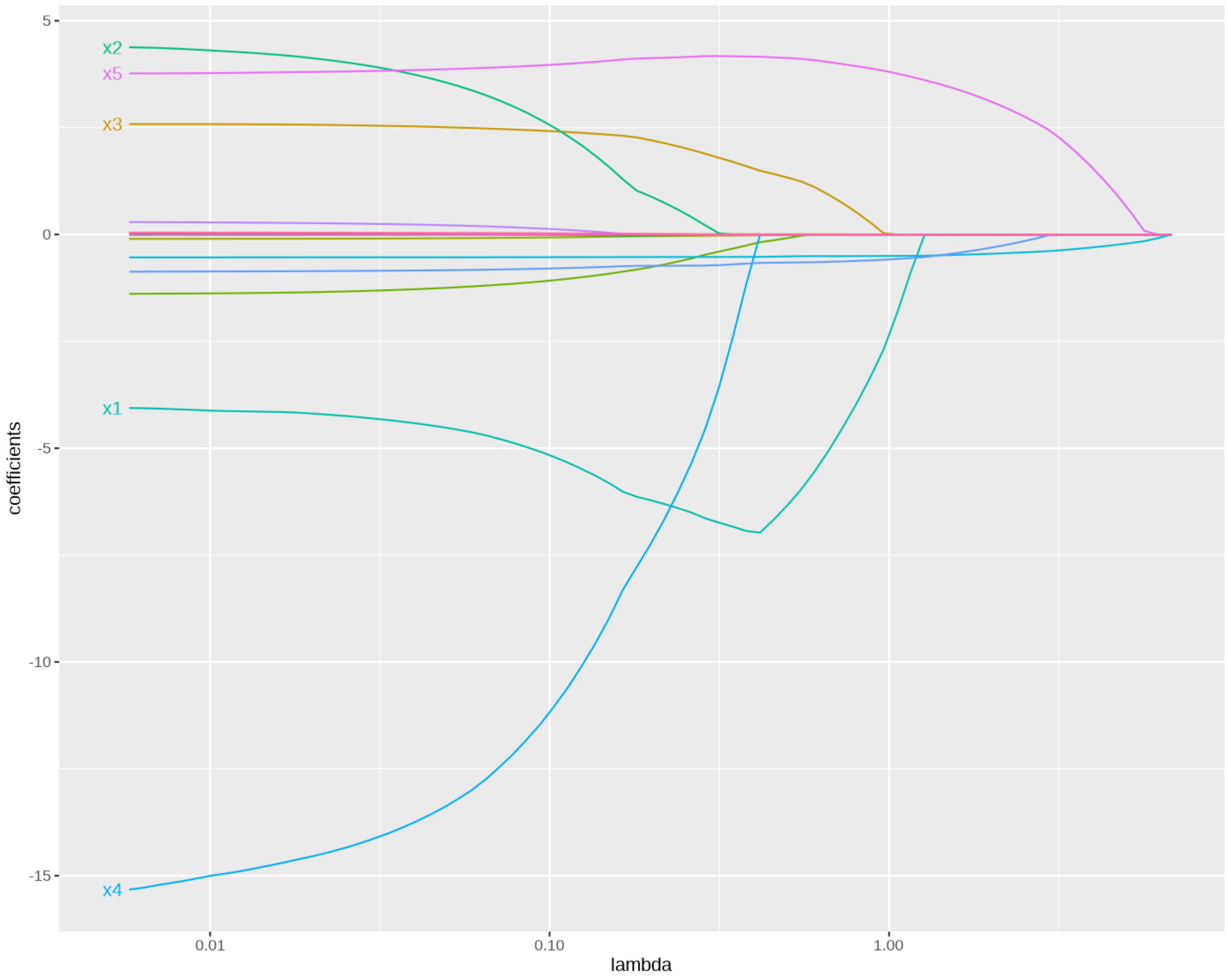
$$\min(\mathrm{SSE} + \lambda \sum_{j=1}^{p} |\beta_j|)$$

Ridge penalty pushes variables to approximately zero.

Lasso actually pushes coefficients all the way to zero.

Automated **feature selection**!

Look at the figure on the following slide, can you see the difference?



x2
x5
x3
x1
x4
5
0
-5
-10
-15
coefficients
0.01
0.10
1.00
lambda

# Elastic nets

Generalisation of ridge and lasso penalties:

$$\min(\text{SSE} + \lambda_1 \sum_{j=1}^{p} \beta_j^2 + \lambda_2 \sum_{j=1}^{p} |\beta_j|)$$

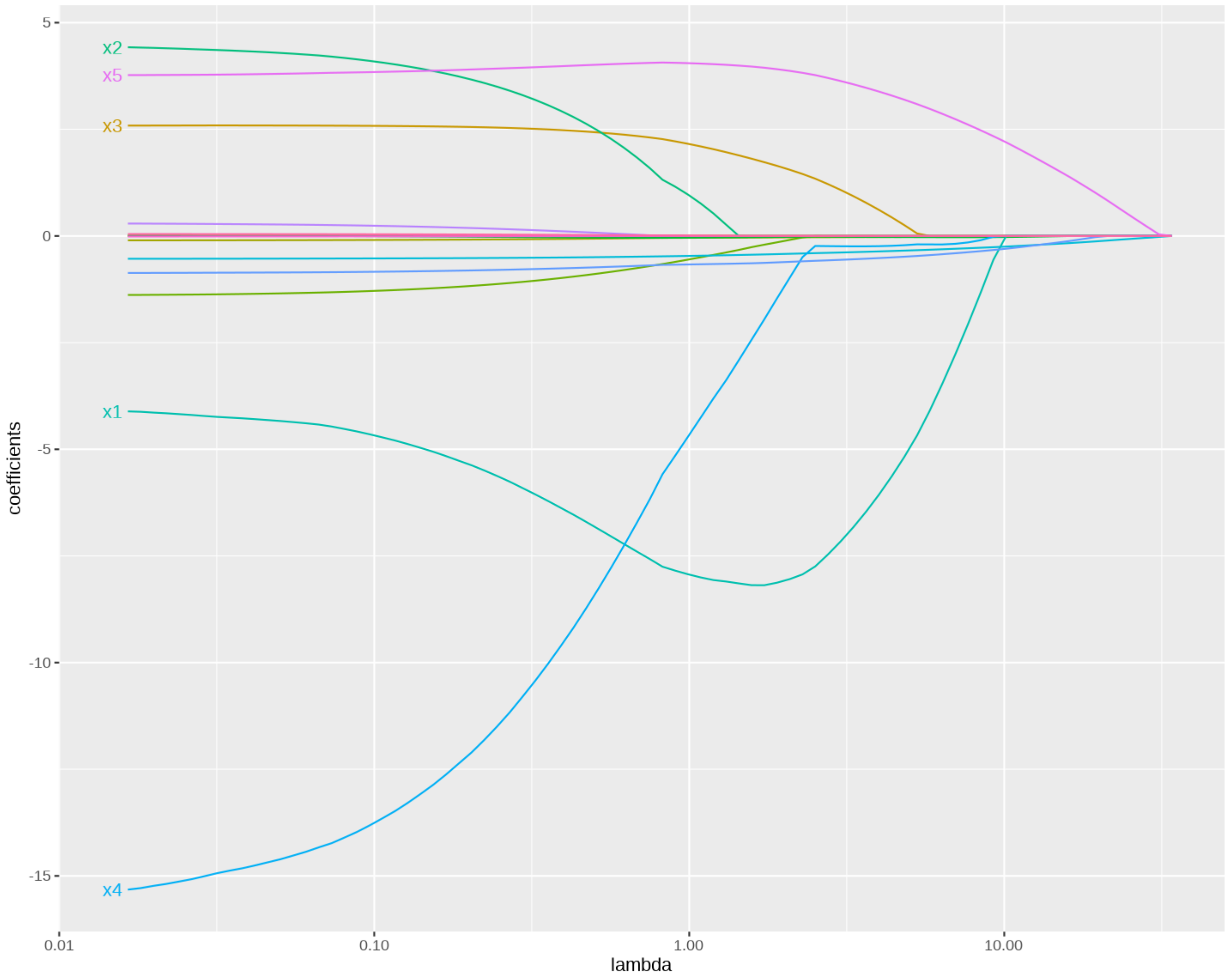
Lasso performs feature selection, **but** when we have two strongly correlated features one may be pushed to zero and the other remains.

Ridge regression penalty more effective in handling correlated features.

Elastic net combines bits of both these methods.

If you are interested in these topics I suggest you read more on them in **ISLR**.

Penalty models are easy to implement and results are surprisingly good.



x2
x5
x3
x1
x4
5
0
-5
-10
-15
coefficients
0.01
0.10
1.00
10.00
lambda

# glmnet

We will be using `glmnet` to conduct our regularised regression analysis.

This package is extremely efficient and fast. It utilises Fortran code.

For those who like Python, one can use `Scikit-Learn` modules.

There are also other R packages available (e.g. `h20`, `elasticnet` and `penalized`).

We have to do some basic transformations to use this package.

```
# Create training  feature matrices
# we use model.matrix(...)[, -1] to discard the intercept
X <- model.matrix(Sale_Price ~ ., ames_train)[, -1]

# transform y with log transformation
Y <- log(ames_train$Sale_Price)
```

# glmnet

Let us become more familiar with the different components of this package.

`alpha` tells **glmnet** which method to implement.

- `alpha` = 0: ridge penalty
- `alpha` = 1: lasso penalty
- 0 < `alpha` < 1: elastic net model

**glmnet** does two things that one needs to be aware of.

1. Standardises features
2. Fits ridge models across wide range of $\lambda$ values (automatically)

# Parameter tuning

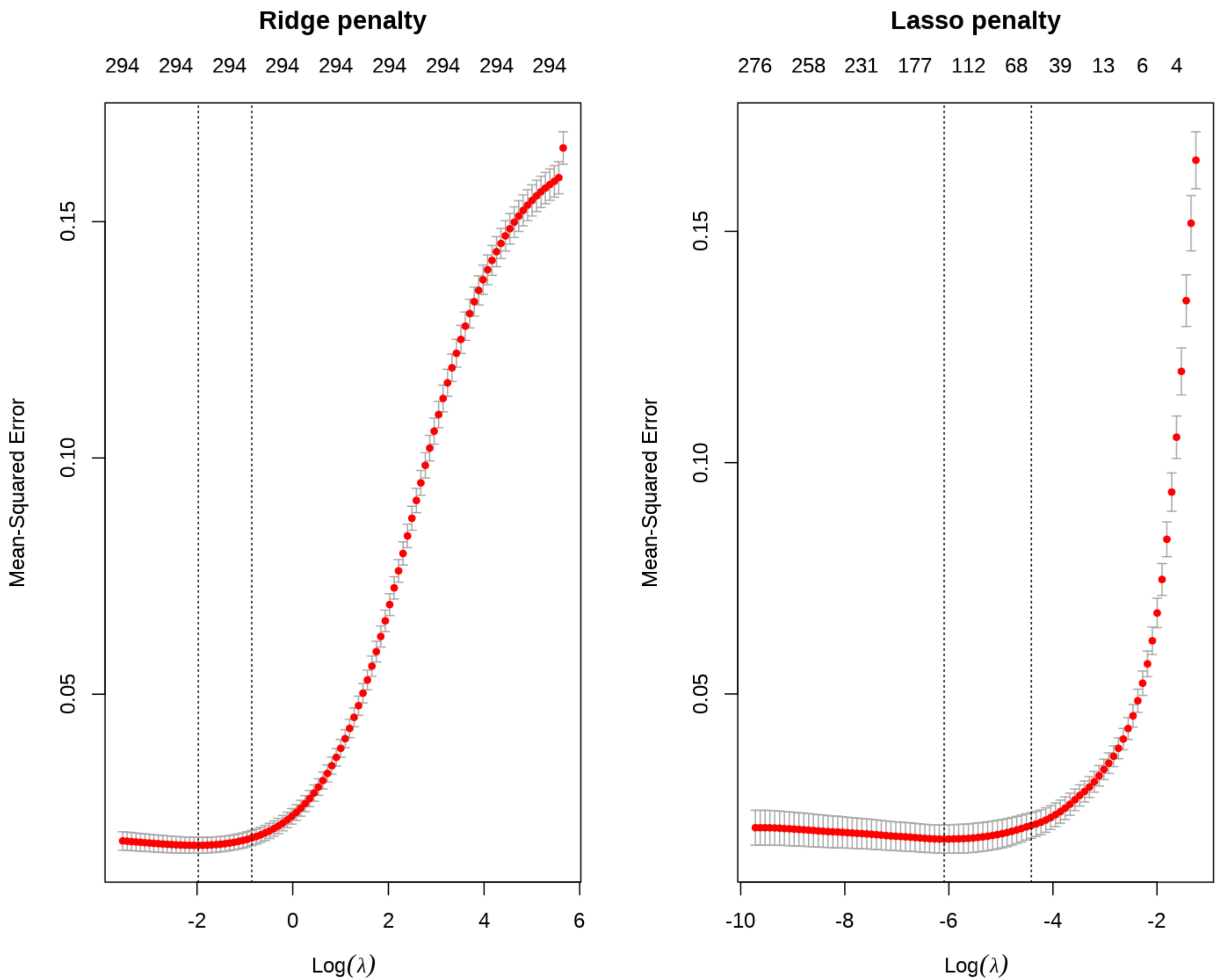
$\lambda$ is the tuning parameter and it helps prevent overfitting training data.

To identify optimal $\lambda$ value we can use $k$-fold cross validation.

```
# Apply CV ridge regression to Ames data
ridge <- cv.glmnet(
  x = X,
  y = Y,
  alpha = 0)

# Apply CV lasso regression to Ames data
lasso <- cv.glmnet(
  x = X,
  y = Y,
  alpha = 1)
```

Now let's plot the result...


Ridge penalty
294 294 294 294 294 294 294 294 294
Mean-Squared Error
0.05
0.10
0.15
-2 0 2 4 6
Log(λ)
Lasso penalty
276 258 231 177 112 68 39 13 6 4
Mean-Squared Error
0.05
0.10
0.15
-10 -8 -6 -4 -2
Log(λ)

```r
# Ridge model
min(ridge$cvm)        # minimum MSE
```

```
## [1] 0.01803344
```

```r
ridge$lambda.min      # lambda for this min MSE
```

```
## [1] 0.1389758
```

```r
# Lasso model
min(lasso$cvm)        # minimum MSE
```

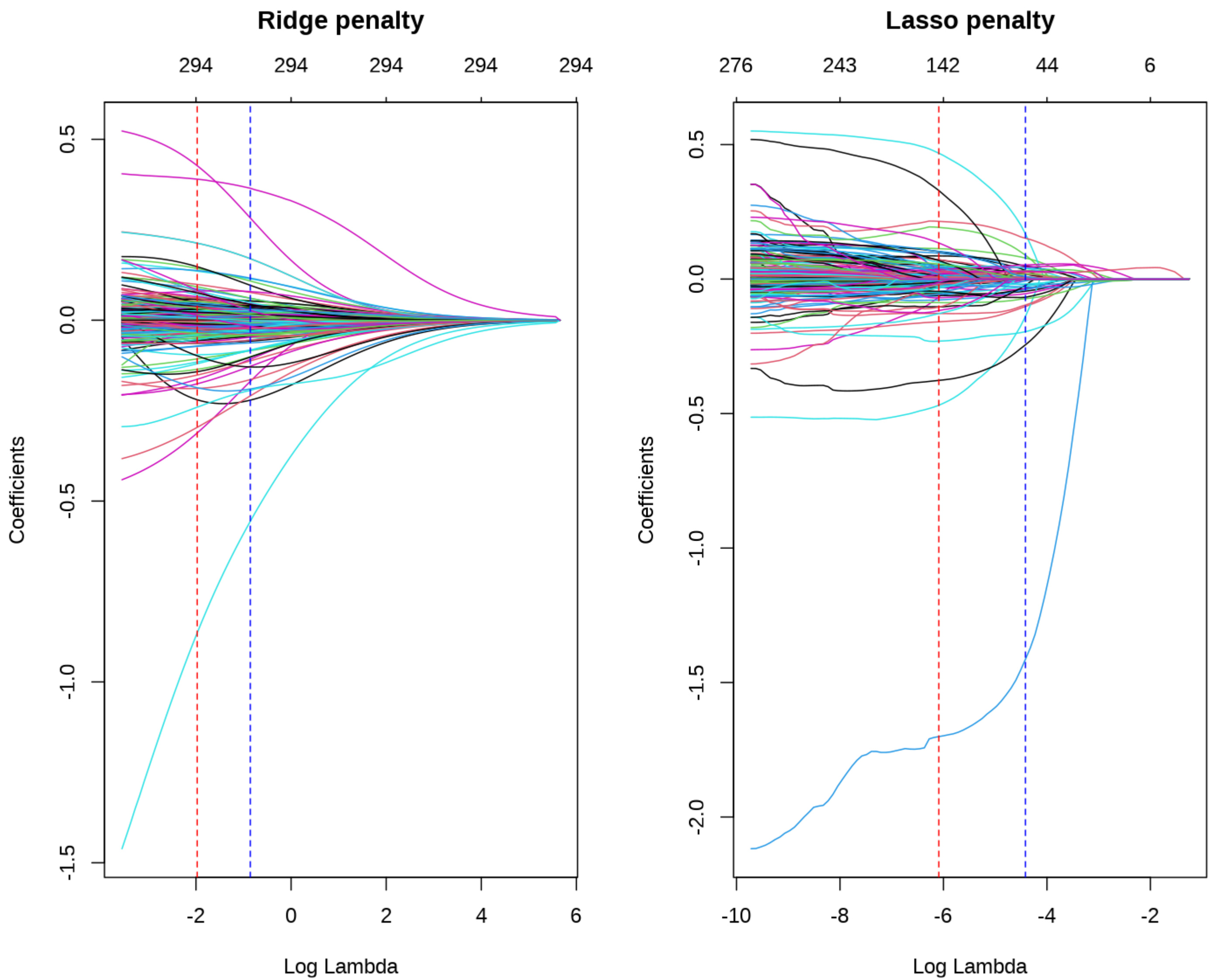
```
## [1] 0.01862178
```

```r
lasso$lambda.min      # lambda for this min MSE
```

```
## [1] 0.002264959
```

```r
lasso$nzero[lasso$lambda == lasso$lambda.min] # No. of coef | Min MSE
```

```
## s52
## 142
```


Ridge penalty
294
294
294
294
294
0.5
0.0
-0.5
-1.0
-1.5
Coefficients
-2
0
2
4
6
Log Lambda
Lasso penalty
276
243
142
44
6
0.5
0.0
-0.5
-1.0
-1.5
-2.0
Coefficients
-10
-8
-6
-4
-2
Log Lambda

```r
# model with lowest RMSE
cv_glmnet$bestTune
```

```
##   alpha     lambda
## 7   0.1 0.02007035
```

```r
# results for model with lowest RMSE
cv_glmnet$results %>%
  filter(alpha == cv_glmnet$bestTune$alpha, lambda == cv_glmnet$bestTune
```

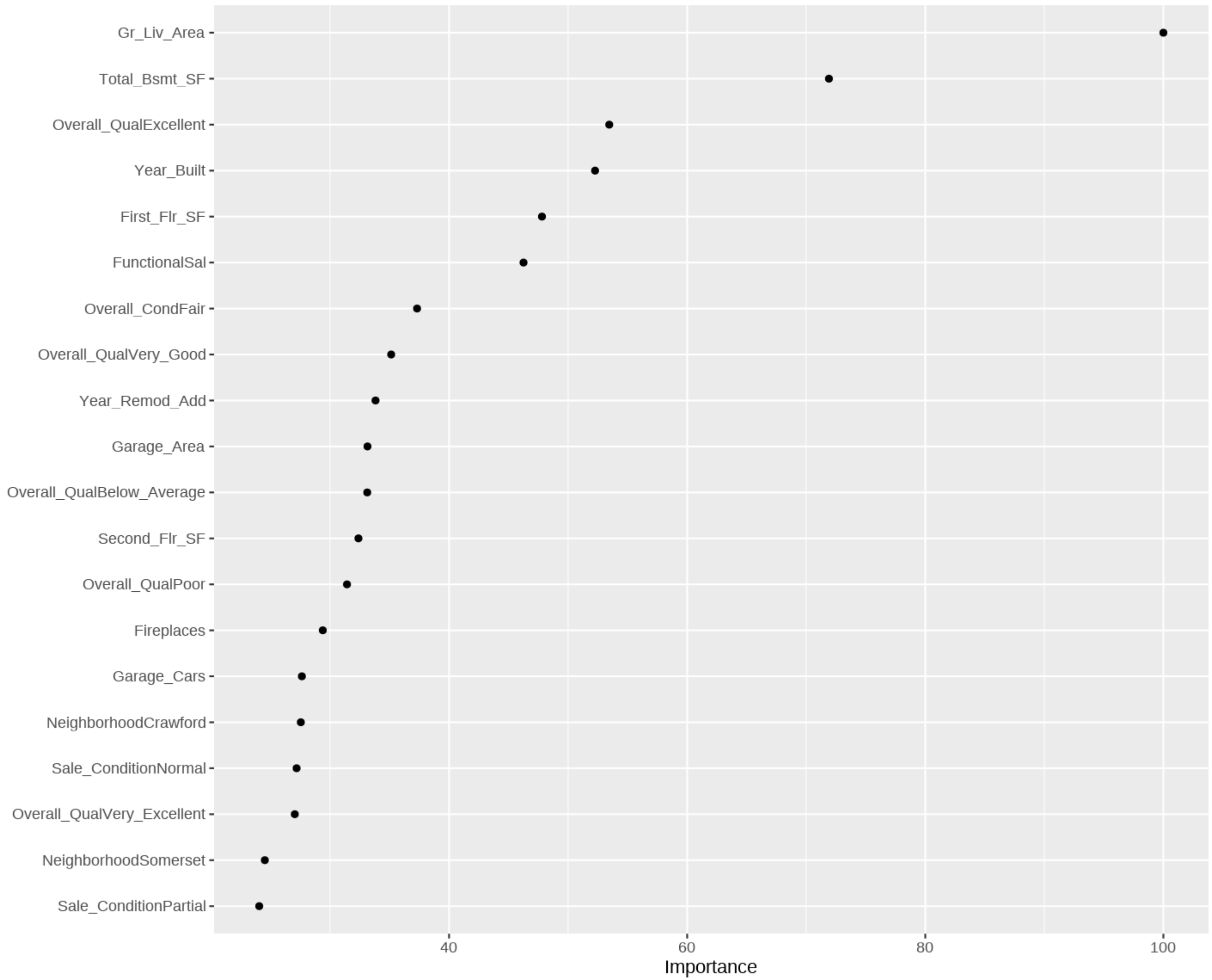
```
##   alpha     lambda      RMSE  Rsquared        MAE     RMSESD RsquaredSD
## 1   0.1 0.02007035 0.1277585 0.9001487 0.08102427 0.02235901  0.0346677
##         MAESD
## 1 0.005667366
```

```r
# predict sales price on training data
pred <- predict(cv_glmnet, X)

# compute RMSE of transformed predicted
RMSE(exp(pred), exp(Y))
```

```
## [1] 19905.05
```

```r
# RMSE of multiple linear regression was 26098.00
```


Gr_Liv_Area
Total_Bsmt_SF
Overall_QualExcellent
Year_Built
First_Flr_SF
FunctionalSal
Overall_CondFair
Overall_QualVery_Good
Year_Remod_Add
Garage_Area
Overall_QualBelow_Average
Second_Flr_SF
Overall_QualPoor
Fireplaces
Garage_Cars
NeighborhoodCrawford
Sale_ConditionNormal
Overall_QualVery_Excellent
NeighborhoodSomerset
Sale_ConditionPartial
40
60
80
100
Importance