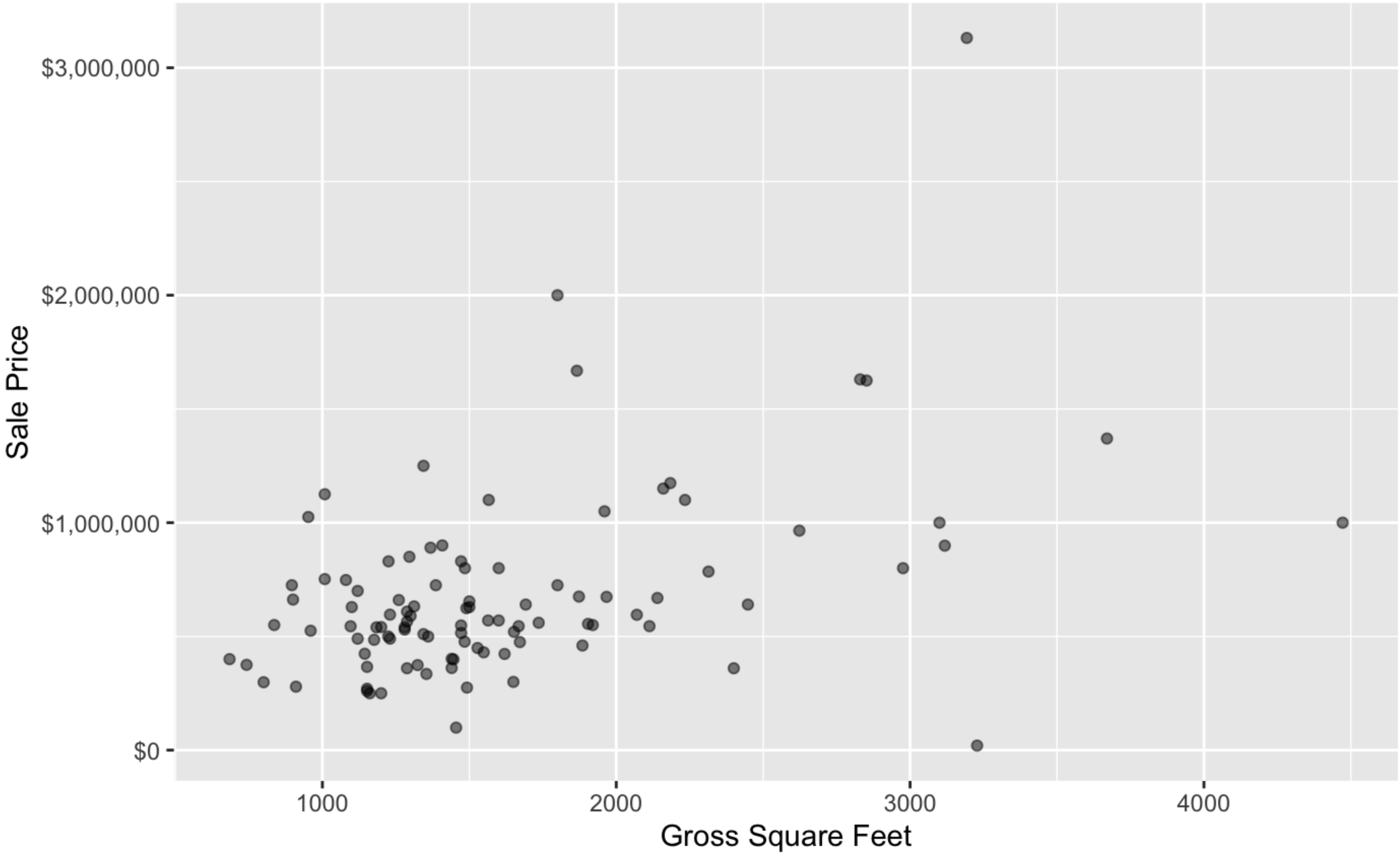
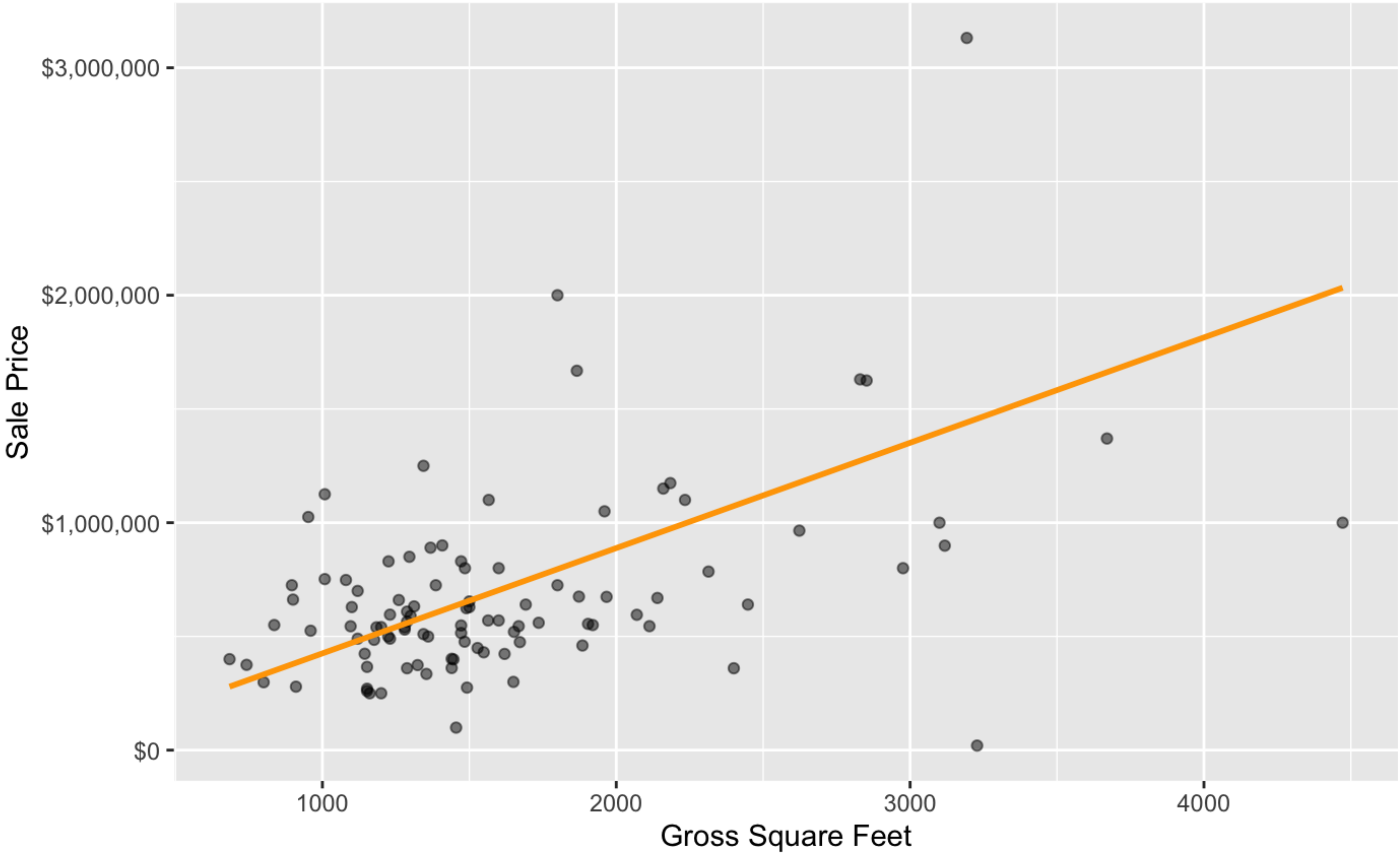
More techniques

Lasso and Ridge Regression

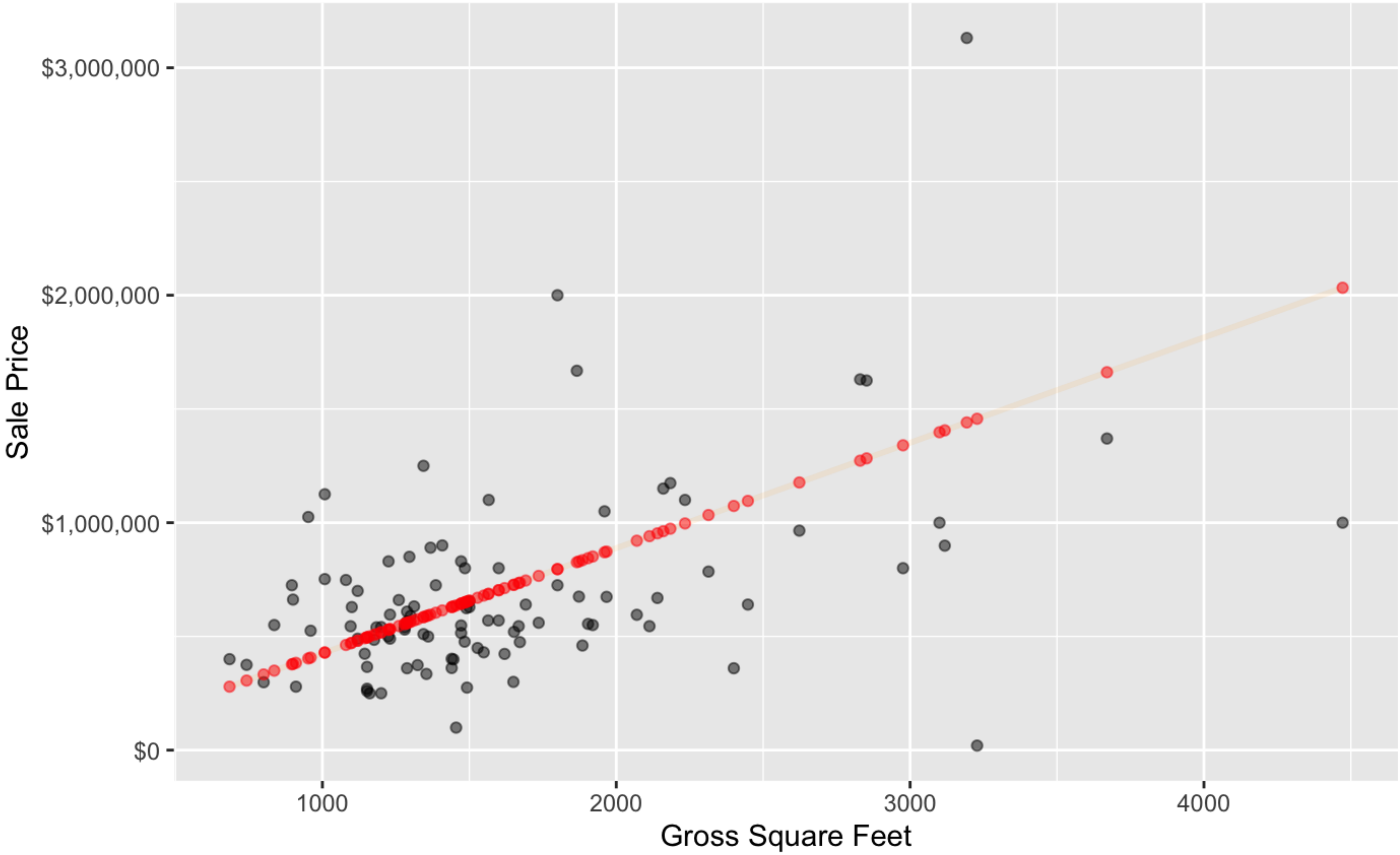
Bagging and Boosting

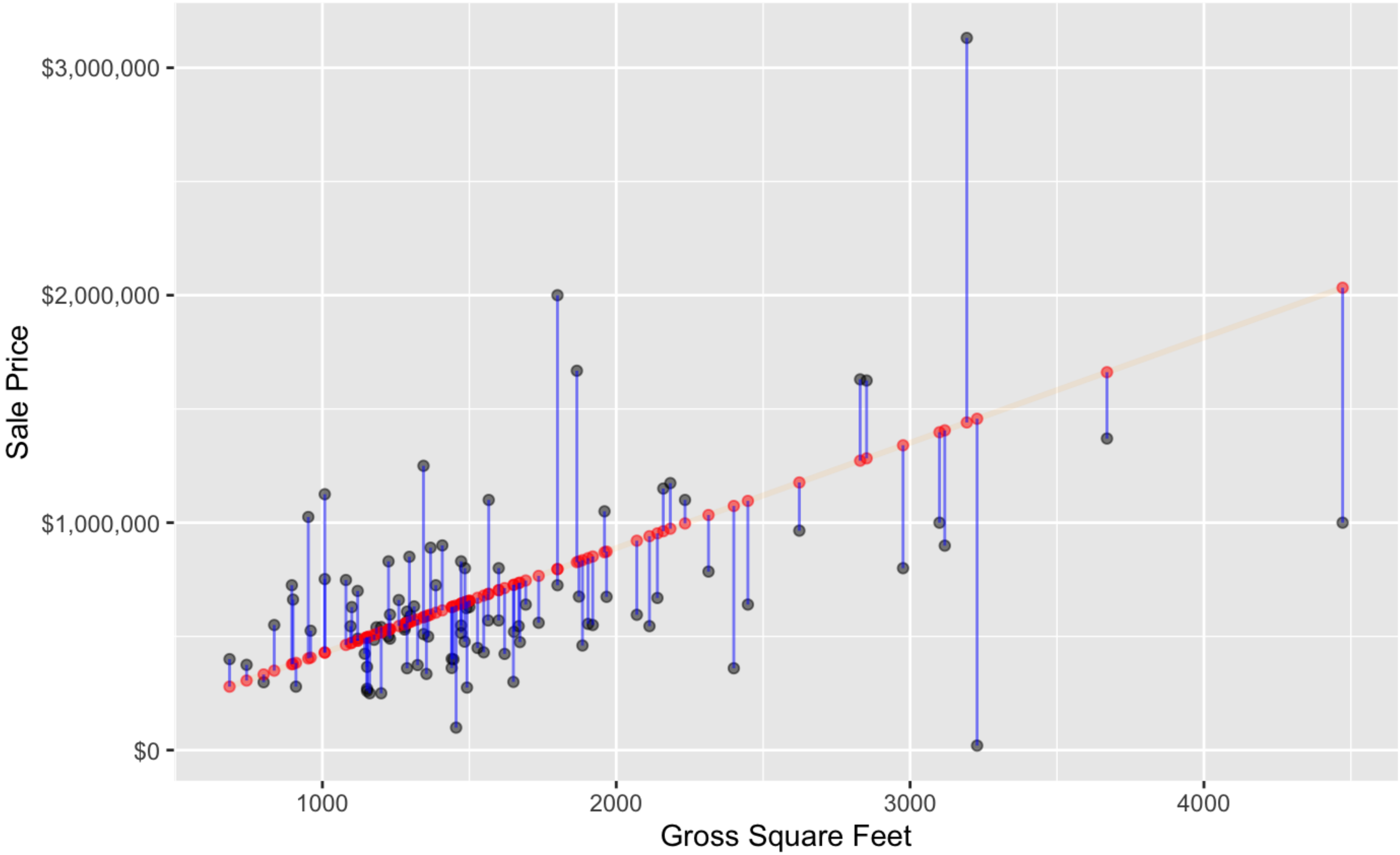


$3,000,000
$2,000,000
$1,000,000
$0
Sale Price
1000
2000
3000
4000
Gross Square Feet

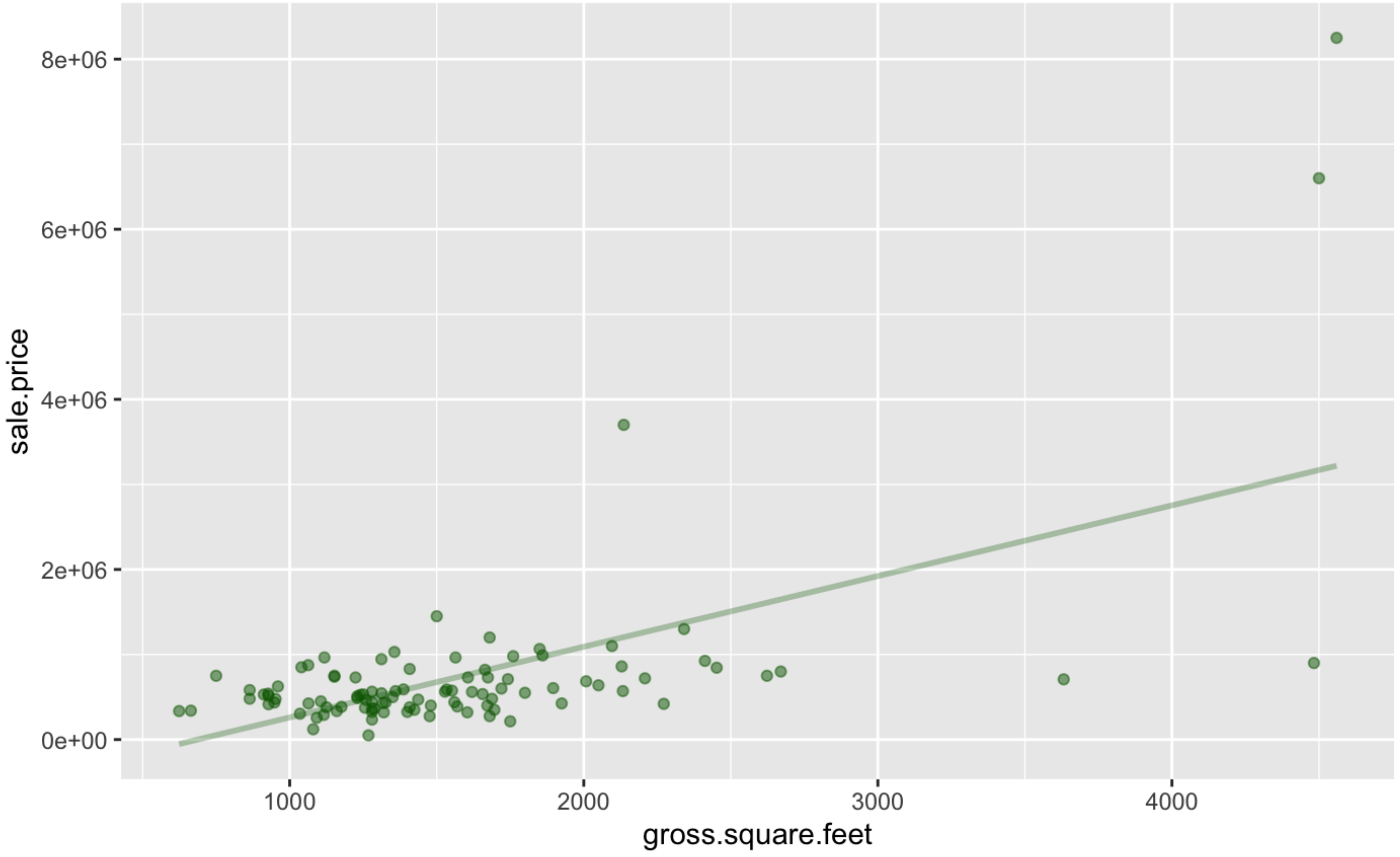


$3,000,000
$2,000,000
$1,000,000
$0
Sale Price
1000
2000
3000
4000
Gross Square Feet

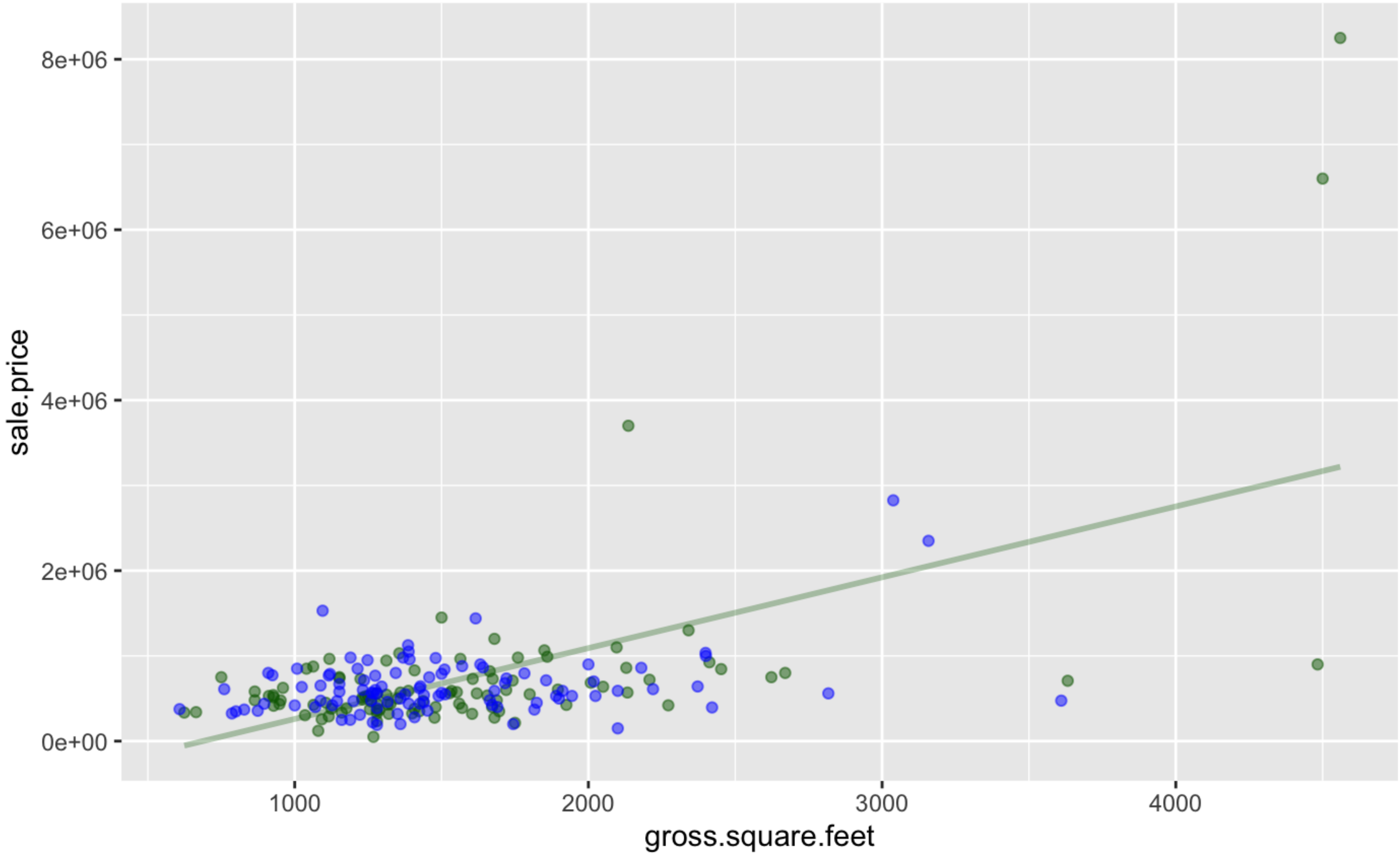

$3,000,000
$2,000,000
$1,000,000
$0
Sale Price
1000
2000
3000
4000
Gross Square Feet



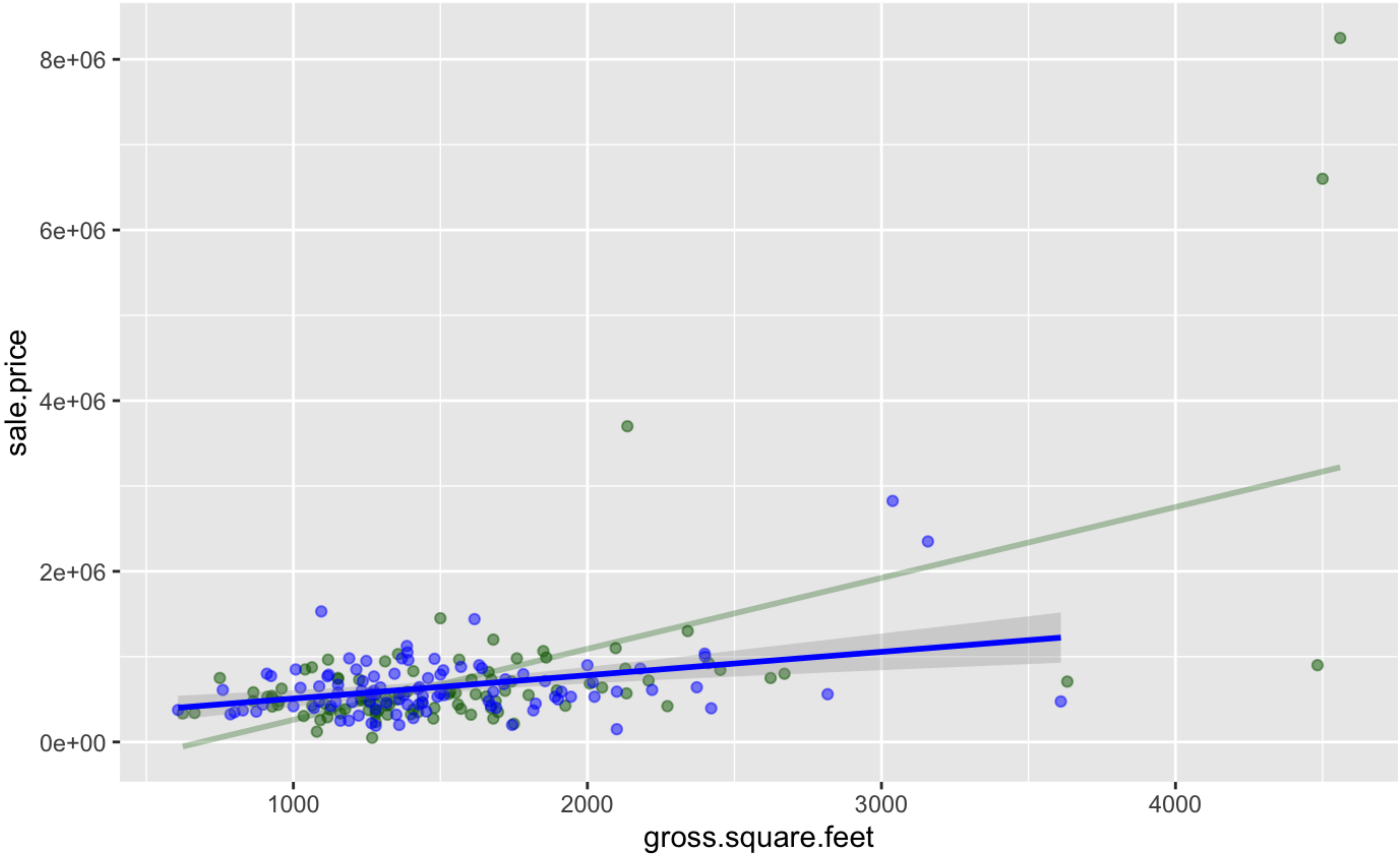
$3,000,000
$2,000,000
$1,000,000
$0
Sale Price
1000
2000
3000
4000
Gross Square Feet



8e+06
6e+06
4e+06
2e+06
0e+00
sale.price
1000
2000
3000
4000
gross.square.feet



8e+06
6e+06
4e+06
2e+06
0e+00
sale.price
1000
2000
3000
4000
gross.square.feet



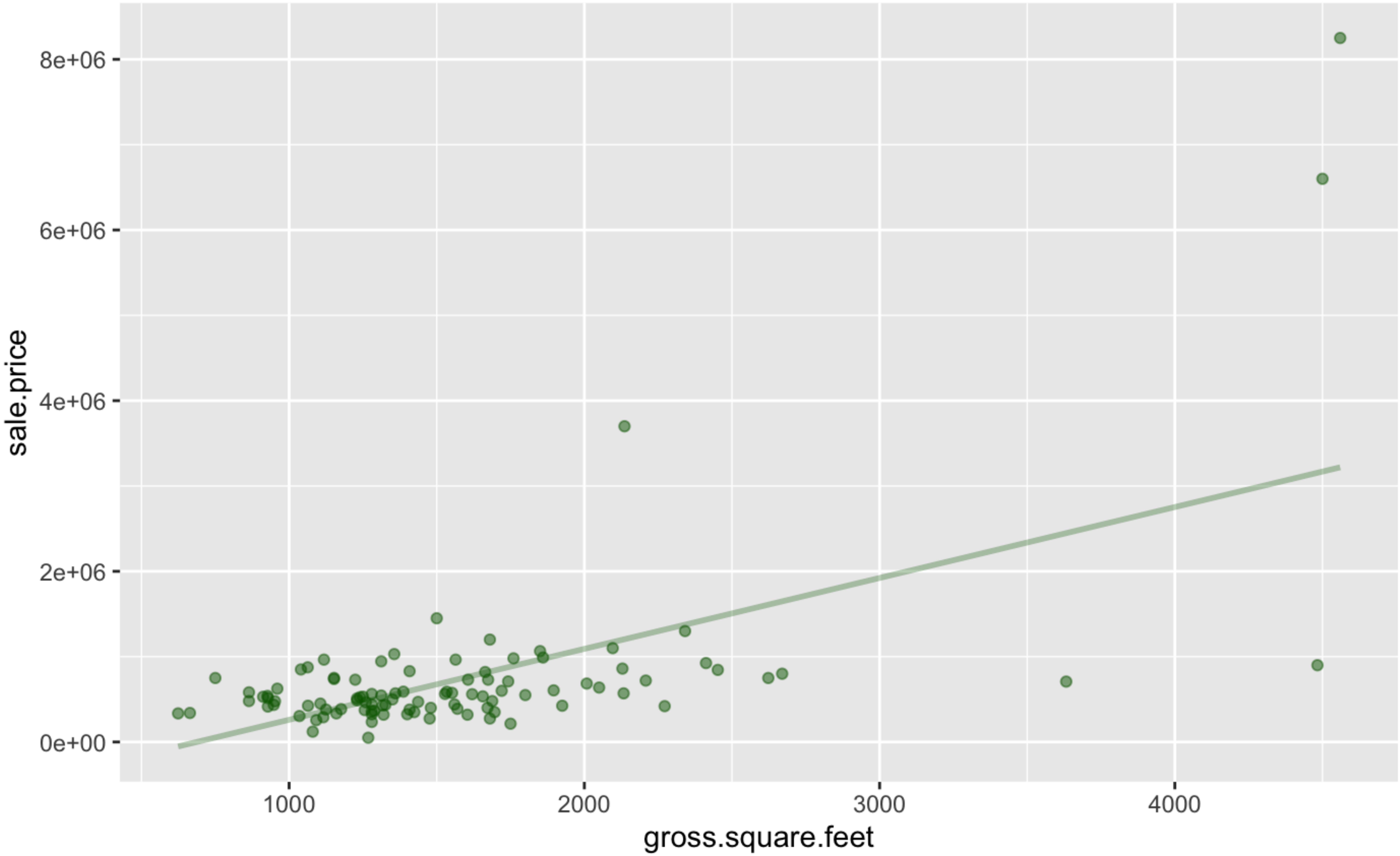
sale.price
8e+06
6e+06
4e+06
2e+06
0e+00
1000
2000
3000
4000
gross.square.feet

We want to move the green line closer to the blue line but only with access to the green dots (training data)

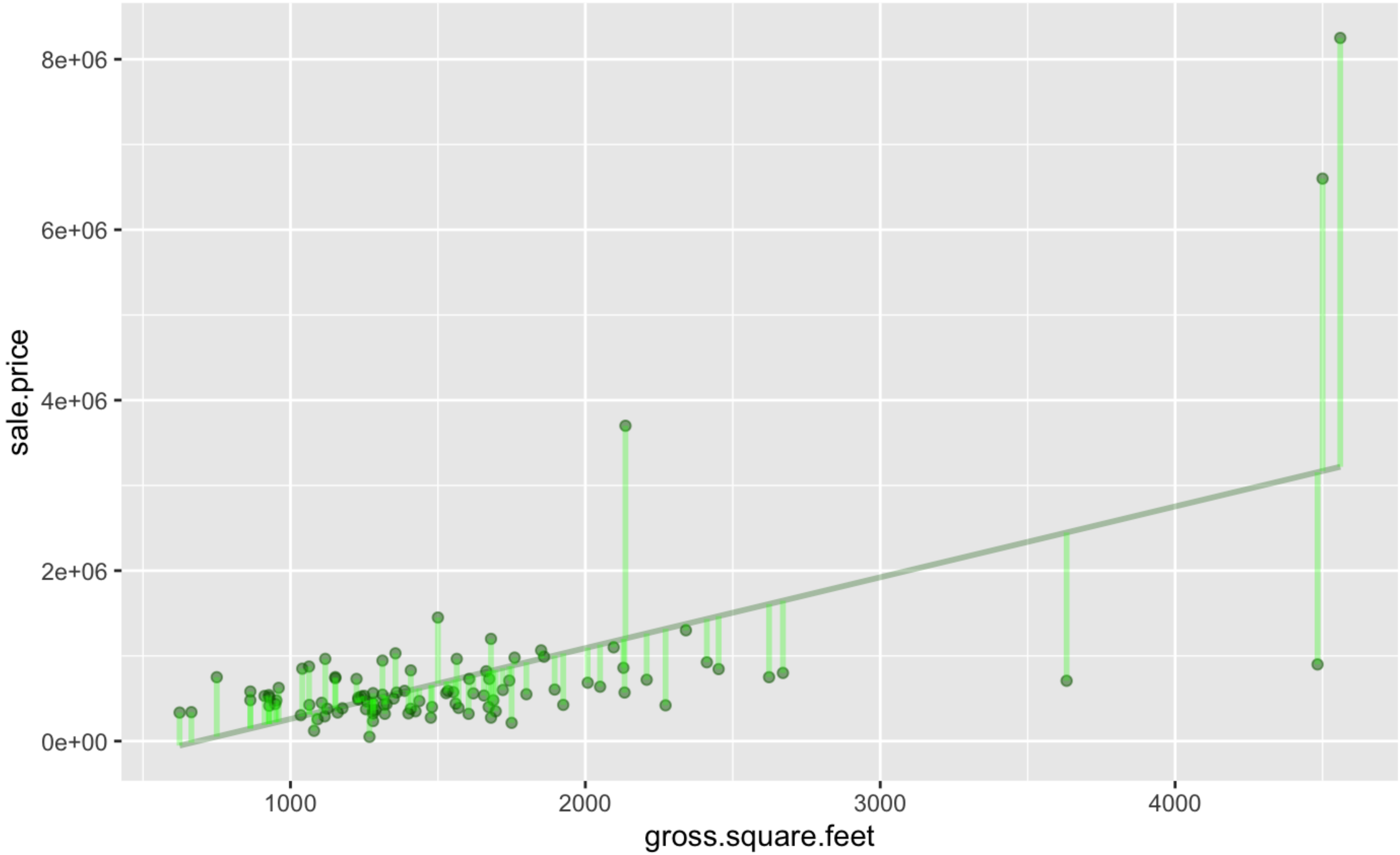
That is, we want to reduce the overfitting to the training data

We want to REDUCE variance

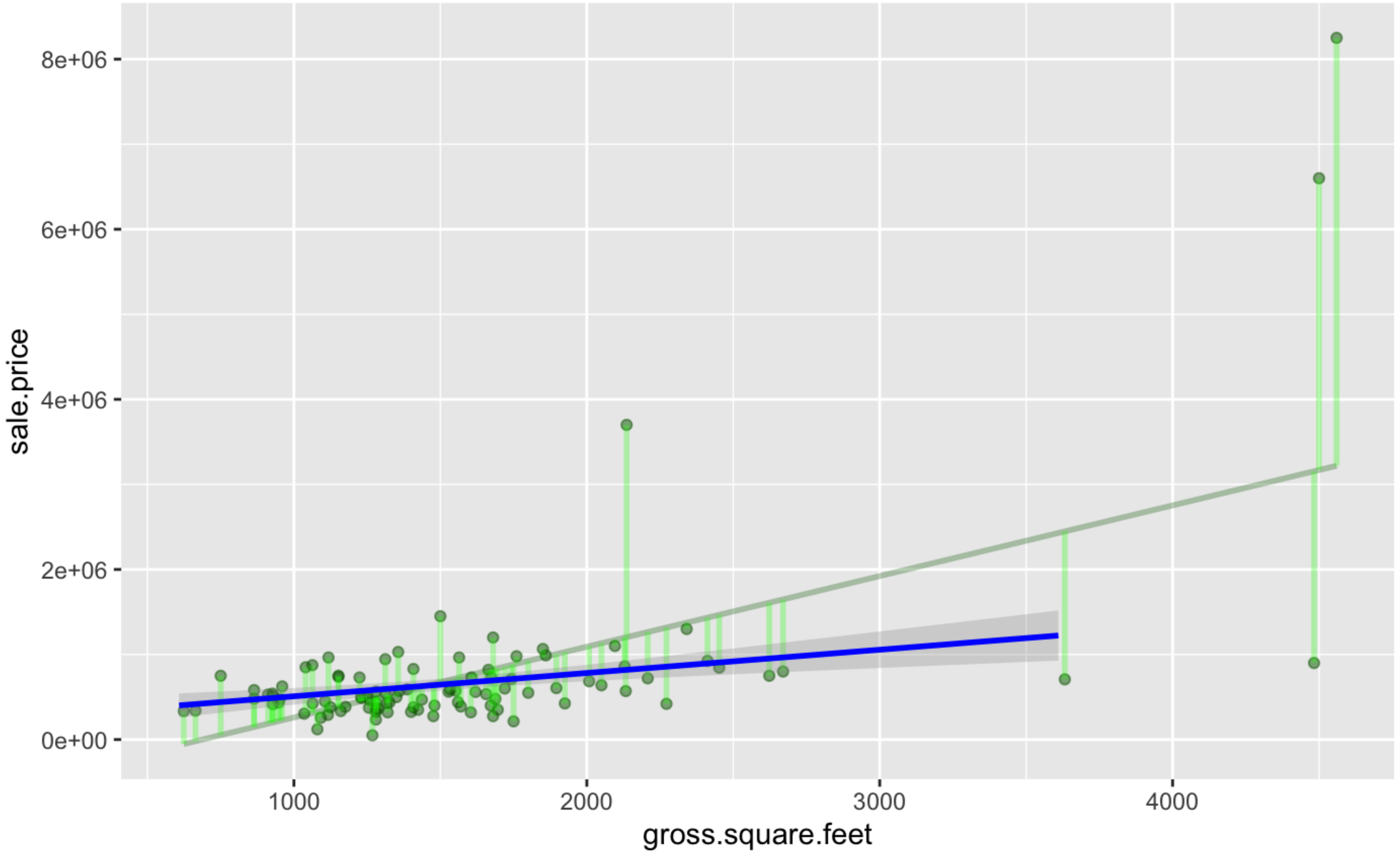
We want to INCREASE bias



$$sale\,.\,price = gross\,.\,square\,.\,feet * m + c$$



$$sale.price = gross.square.feet * m + c$$



$$sale\,.\,price = gross\,.\,square\,.\,feet * m + c$$

$$sale\,.price = gross\,.square\,.feet * m + c$$

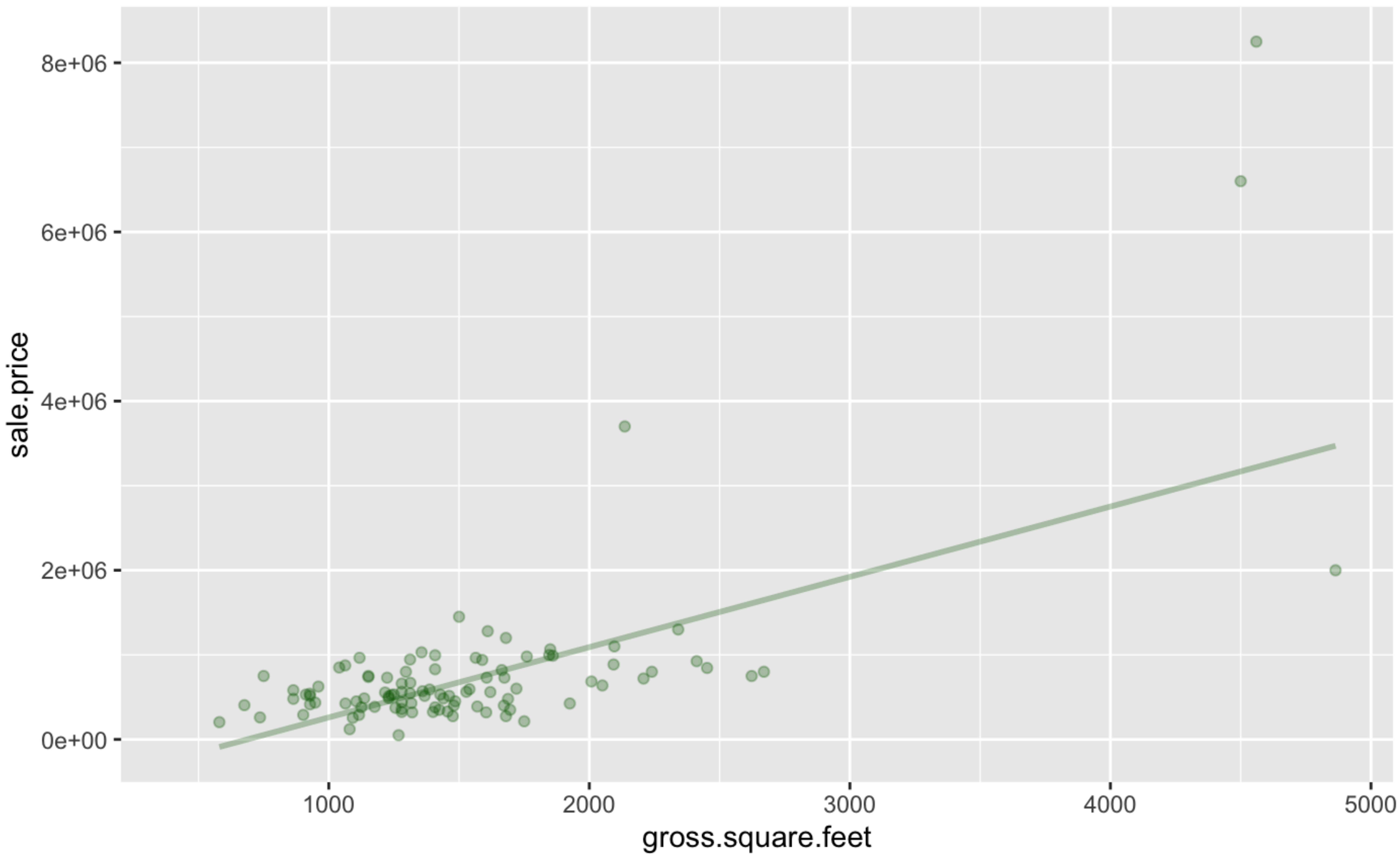
- Linear Regression
  - Minimize “errors”
  - Minimize RMSE - equivalent to minimize “sum of the squared residuals”

- **Ridge Regression**
  - Minimize “sum of the squared residuals + **penalty**”
  - Minimize “sum of the squared residuals + $\lambda * m^2$”

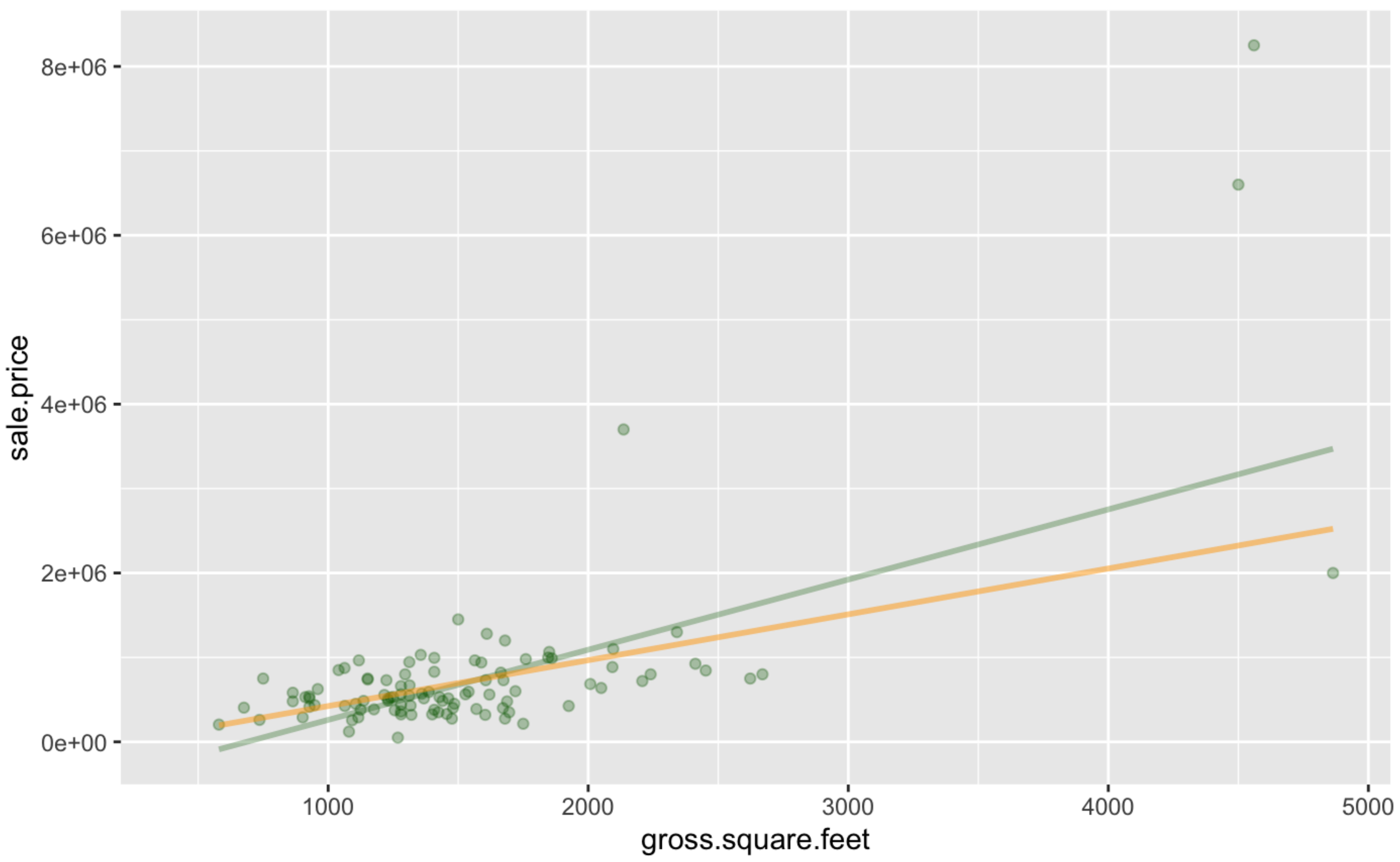
- **Ridge Regression**
  - Minimize “sum of the squared residuals + **penalty**”
  - Minimize “sum of the squared residuals + $\lambda * m^2$”
- Penalty gets smaller as m^2 gets smaller, i.e. *penalty is lower when slope shrinks towards zero* - that is what we want!

Why?

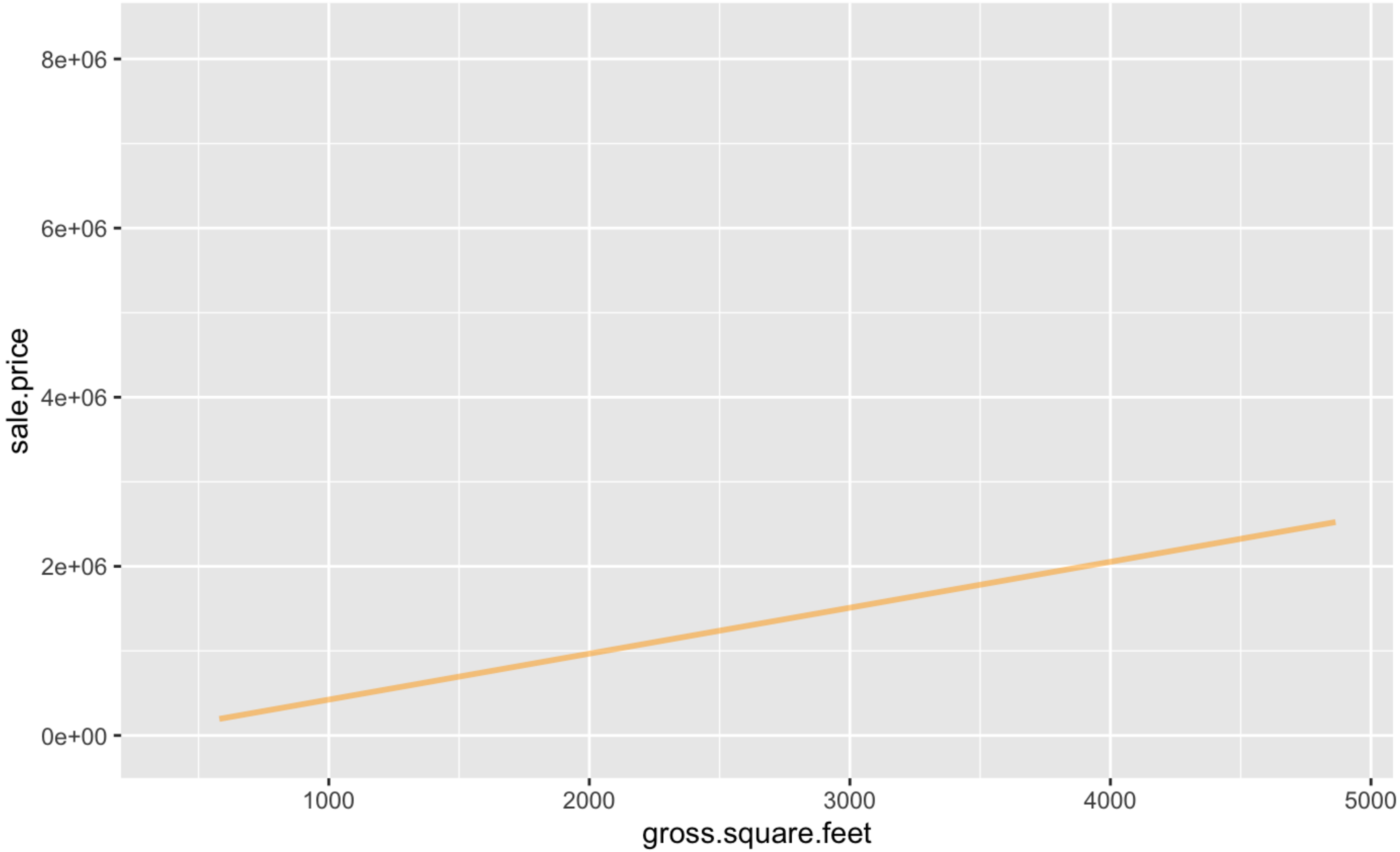
- Why do we want to shrink slope towards zero
- When slope is large - *small change in* ***x*** *creates large changes in* ***y***
  - High variance - sign of overfit
- Shrinking slope implies
  - *small change in* ***x*** *creates small(er) changes in* ***y***
  - Lower variance - cost of added bias

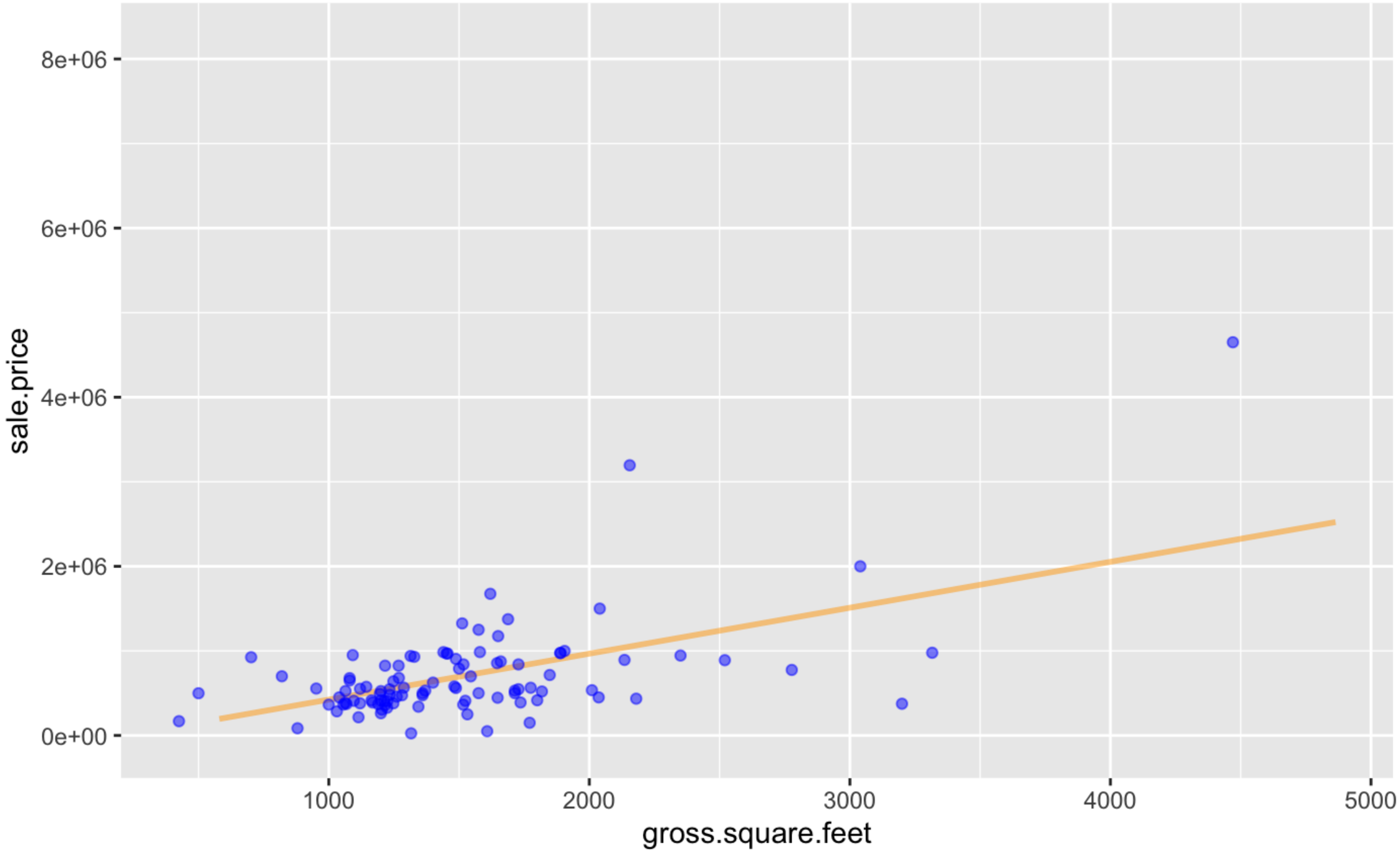


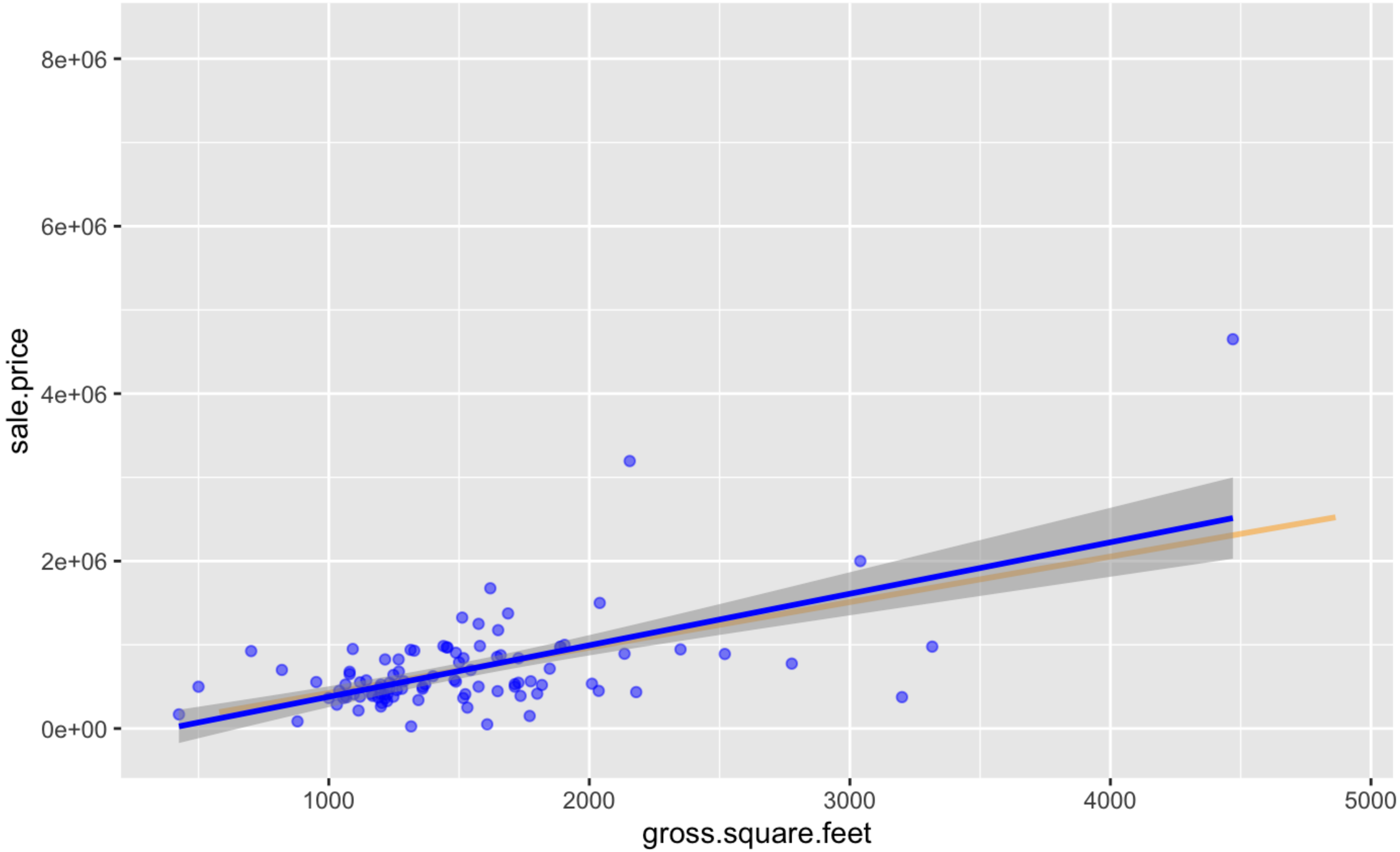
8e+06
6e+06
4e+06
2e+06
0e+00
sale.price
1000
2000
3000
4000
5000
gross.square.feet



8e+06
6e+06
4e+06
2e+06
0e+00
sale.price
1000
2000
3000
4000
5000
gross.square.feet


8e+06
6e+06
4e+06
2e+06
0e+00
sale.price
1000
2000
3000
4000
5000
gross.square.feet


8e+06
6e+06
4e+06
2e+06
0e+00
sale.price
1000
2000
3000
4000
5000
gross.square.feet



8e+06
6e+06
4e+06
2e+06
0e+00
sale.price
1000
2000
3000
4000
5000
gross.square.feet

What about the λ ?

- λ (lambda) can be any value from 0 -> infinity
- Larger λ means larger penalty -> slope closer to 0
- Smaller λ means lower penalty -> slope closer to normal linear regression
- If λ = 0, Ridge regression = linear regression !!!!!!!

How to decide λ?
Hyper parameter tuning!

```r
set.seed(345)
ridge_spec <- linear_reg(penalty = tune(), mixture = 0) %>%
    set_engine("glmnet")

```

```
> ridge_spec
Linear Regression Model Specification (regression)

Main Arguments:
  penalty = tune()
  mixture = 0

Computational engine: glmnet
```

```r
set.seed(345)
ridge_spec <- linear_reg(penalty = tune(), mixture = 0) %>%
    set_engine("glmnet")

# Create cross validation folds
cv_folds <- vfold_cv(train, v = 5)
```

```
> cv_folds
#  5-fold cross-validation
# A tibble: 5 × 2
  splits           id
  <list>           <chr>
1 <split [100/26]> Fold1
2 <split [101/25]> Fold2
3 <split [101/25]> Fold3
4 <split [101/25]> Fold4
5 <split [101/25]> Fold5
```

```r
set.seed(345)
ridge_spec <- linear_reg(penalty = tune(), mixture = 0) %>%
    set_engine("glmnet")

# Create cross validation folds
cv_folds <- vfold_cv(train, v = 5)

# Set up grid of lambda values to try
lambda_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 100)
```

```
> lambda_grid%>%
+     filter(row_number() %% 5 == 1)
# A tibble: 20 × 1
        penalty
          <dbl>
 1      0.00001
 2      0.0000320
 3      0.000102
 4      0.000327
 5      0.00105
 6      0.00335
 7      0.0107
 8      0.0343
 9      0.110
10      0.351
11      1.12
12      3.59
13     11.5
14     36.8
15    118.
16    376.
17   1205.
18   3854.
19  12328.
20  39442.
```

```r
penalty(range = c(-10, 0), trans = transform_log10())
```

Arguments

`range`	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

```r
set.seed(345)
ridge_spec <- linear_reg(penalty = tune(), mixture = 0) %>%
    set_engine("glmnet")

# Create cross validation folds
cv_folds <- vfold_cv(train, v = 5)

# Set up grid of lambda values to try
lambda_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 100)

train <- train %>% mutate(dummy = 1)
ridge_recipe <- recipe(sale.price ~ gross.square.feet + dummy, data = train)
```

```r
set.seed(345)
ridge_spec <- linear_reg(penalty = tune(), mixture = 0) %>%
    set_engine("glmnet")

# Create cross validation folds
cv_folds <- vfold_cv(train, v = 5)

# Set up grid of lambda values to try
lambda_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 100)

train <- train %>% mutate(dummy = 1)
ridge_recipe <- recipe(sale.price ~ gross.square.feet + dummy, data = train)

ridge_wf <- workflow() %>%
  add_model(ridge_spec) %>%
  add_recipe(ridge_recipe)
```

```
> ridge_wf
══ Workflow ════════════════════════════════════
Preprocessor: Recipe
Model: linear_reg()

── Preprocessor ────────────────────────────────
0 Recipe Steps

── Model ───────────────────────────────────────
Linear Regression Model Specification (regression)

Main Arguments:
  penalty = tune()
  mixture = 0

Computational engine: glmnet
```

```r
set.seed(345)
ridge_spec <- linear_reg(penalty = tune(), mixture = 0) %>%
    set_engine("glmnet")

# Create cross validation folds
cv_folds <- vfold_cv(train, v = 5)

# Set up grid of lambda values to try
lambda_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 100)

train <- train %>% mutate(dummy = 1)
ridge_recipe <- recipe(sale.price ~ gross.square.feet + dummy, data = train)

ridge_wf <- workflow() %>%
  add_model(ridge_spec) %>%
  add_recipe(ridge_recipe)

# Tune the model
tune_results <- tune_grid(
    ridge_wf,
    resamples = cv_folds,
    grid = lambda_grid
)

# Find best lambda
best_lambda <- select_best(tune_results, metric = "rmse")
```

```
> best_lambda
# A tibble: 1 × 2
  penalty .config
    <dbl> <chr>
1  100000 Preprocessor1_Model100
```

```r
set.seed(345)
ridge_spec <- linear_reg(penalty = tune(), mixture = 0) %>%
    set_engine("glmnet")

# Create cross validation folds
cv_folds <- vfold_cv(train, v = 5)

# Set up grid of lambda values to try
lambda_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 100)

train <- train %>% mutate(dummy = 1)
ridge_recipe <- recipe(sale.price ~ gross.square.feet + dummy, data = train)

ridge_wf <- workflow() %>%
  add_model(ridge_spec) %>%
  add_recipe(ridge_recipe)

# Tune the model
tune_results <- tune_grid(
    ridge_wf,
    resamples = cv_folds,
    grid = lambda_grid
)

# Find best lambda
best_lambda <- select_best(tune_results, metric = "rmse")

# Finalize workflow with best lambda
final_ridge_workflow <- ridge_wf %>%
    finalize_workflow(best_lambda)

# Fit final model
ridge_fit <- final_ridge_workflow %>%
    fit(data = train)

# Look at results
ridge_fit %>%
    tidy()
```

```r
set.seed(345)
ridge_spec <- linear_reg(penalty = tune(), mixture = 0) %>%
    set_engine("glmnet")

# Create cross validation folds
cv_folds <- vfold_cv(train, v = 5)

# Set up grid of lambda values to try
lambda_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 100)

train <- train %>% mutate(dummy = 1)
ridge_recipe <- recipe(sale.price ~ gross.square.feet + dummy, data = train)

ridge_wf <- workflow() %>%
  add_model(ridge_spec) %>%
  add_recipe(ridge_recipe)

# Tune the model
tune_results <- tune_grid(
    ridge_wf,
    resamples = cv_folds,
    grid = lambda_grid
)

# Find best lambda
best_lambda <- select_best(tune_results, metric = "rmse")

# Finalize workflow with best lambda
final_ridge_workflow <- ridge_wf %>%
    finalize_workflow(best_lambda)
```

```
> final_ridge_workflow
══ Workflow ════════════════════════════════════════════
Preprocessor: Recipe
Model: linear_reg()

── Preprocessor ────────────────────────────────────────
0 Recipe Steps

── Model ───────────────────────────────────────────────
Linear Regression Model Specification (regression)

Main Arguments:
  penalty = 1e+05
  mixture = 0

Computational engine: glmnet
```

```r
set.seed(345)
ridge_spec <- linear_reg(penalty = tune(), mixture = 0) %>%
    set_engine("glmnet")

# Create cross validation folds
cv_folds <- vfold_cv(train, v = 5)

# Set up grid of lambda values to try
lambda_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 100)

train <- train %>% mutate(dummy = 1)
ridge_recipe <- recipe(sale.price ~ gross.square.feet + dummy, data = train)

ridge_wf <- workflow() %>%
  add_model(ridge_spec) %>%
  add_recipe(ridge_recipe)

# Tune the model
tune_results <- tune_grid(
    ridge_wf,
    resamples = cv_folds,
    grid = lambda_grid
)

# Find best lambda
best_lambda <- select_best(tune_results, metric = "rmse")

# Finalize workflow with best lambda
final_ridge_workflow <- ridge_wf %>%
    finalize_workflow(best_lambda)

# Fit final model
ridge_fit <- final_ridge_workflow %>%
    fit(data = train)

# Look at results
ridge_fit %>%
    tidy()
```

````r
rmse_ridge <- rmse(lm_ridge_test_predictions, truth = sale.price, estimate = ridge_pred)
lmse_ridge <- rmse(lm_ridge_test_predictions, truth = sale.price, estimate = lm_pred)

print(paste("Ridge RMSE:", round(rmse_ridge$.estimate)))
print(paste("LM RMSE:", round(lmse_ridge$.estimate)))
```
````

```
[1] "Ridge RMSE: 490396"
[1] "LM RMSE: 507865"
```
```

Notes

- With multiple predictors the error is $\lambda(m_1^2 + m_2^2 + \ldots + m_n^2)$

- Works on Logistic regression as well

- Mixture = 0 -> Ridge regression
- Mixture = 1 -> Lasso regression

Lasso Regression

- Lasso works similarly but uses a different penalty

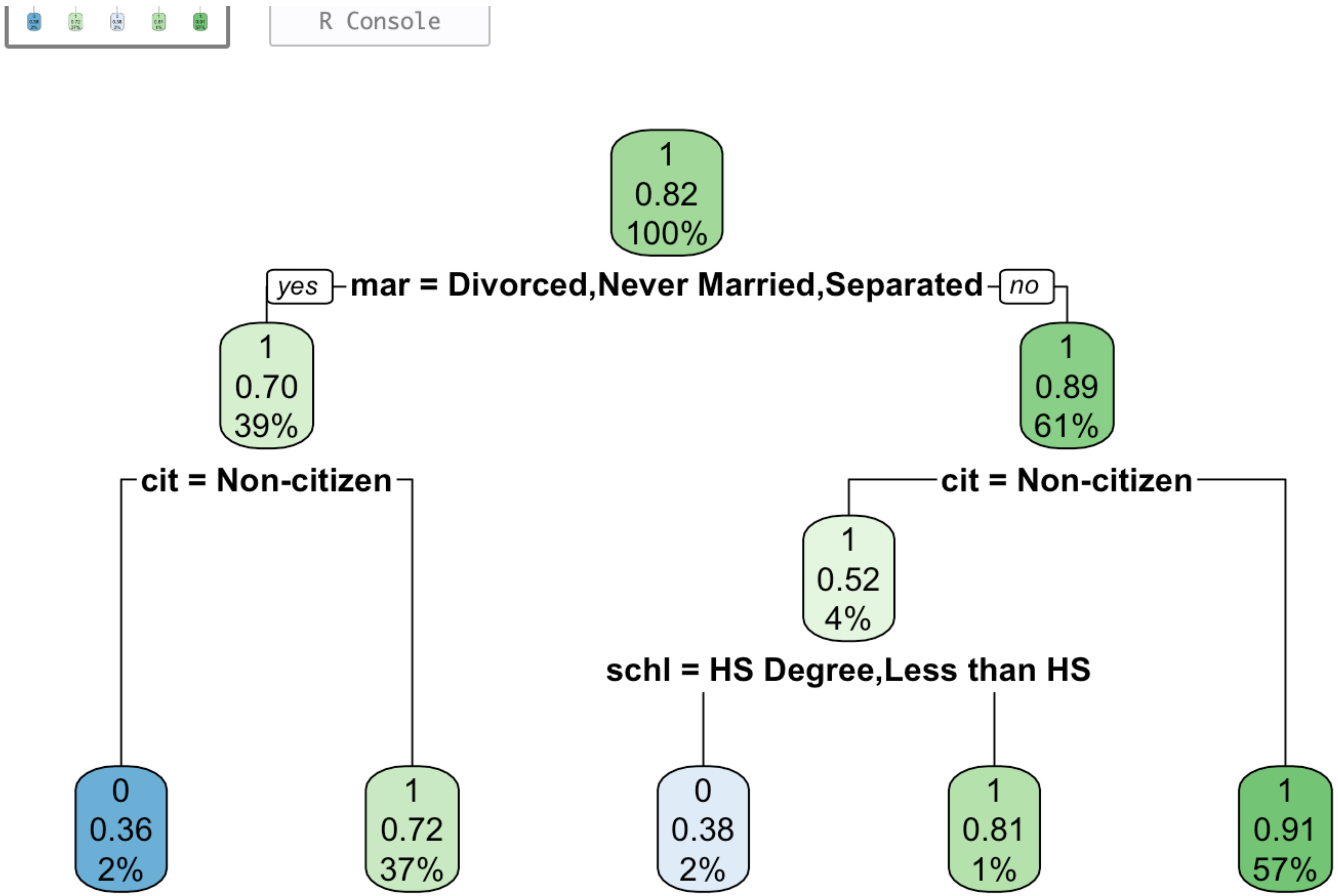
- $\lambda * (|m_1| + |m_2| + \ldots + |m_n|)$

Lasso vs Ridge

- Ridge -
  - “L2 Norm” - Euclidean distance
  - Moves closer to 0
  - Use when all variables are relevant but need shrinkage to avoid overfitting.

- Lasso
  - “L1 Norm” - Manhattan distance
  - Can set to exactly 0 - useful for variable selection
  - Use when you suspect some features are irrelevant for feature selection

Decision Tree



Bagging (Bootstrap AGGregating)

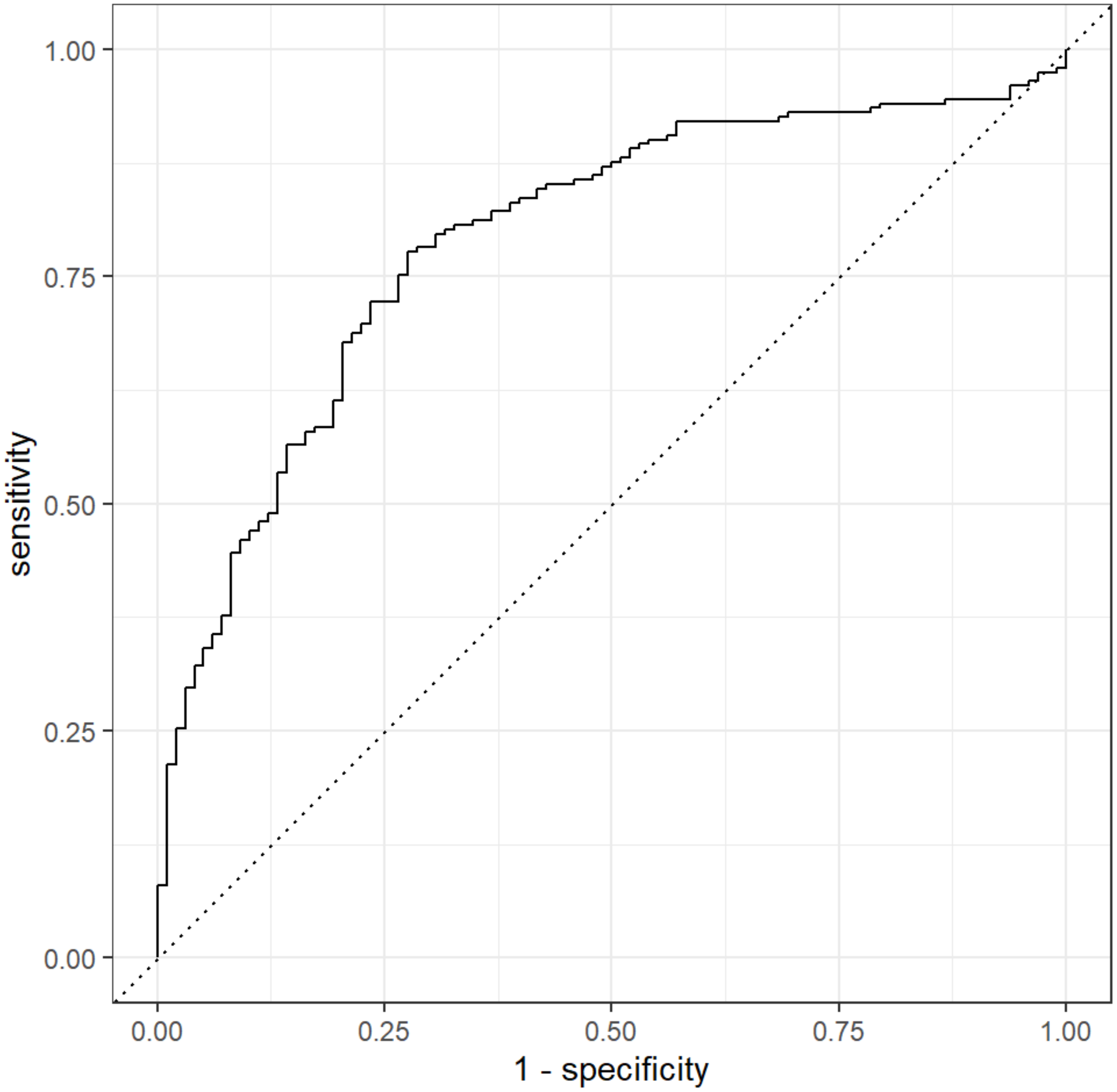
- Uses all available features when building each decision tree. It doesn't perform feature randomization at each split.
- Since all features are considered at each split, the trees in a bagged ensemble tend to be more correlated with each other.

- Difference from random_forest —
  - Randomly select a subset of features at each node split in each tree

- Similarity —
  - Both use random subsets of data for each tree that is trained

Boosting

- Boosting is also an ensemble technique that builds trees sequentially.
  - We make a simple tree, see where it is weakest, and make that part better
  - Process:
    - Train first tree on original data
    - Give higher weight to misclassified samples
    - Train next tree focusing on harder examples
    - Repeat, creating a chain of complementary trees


sensitivity
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1 - specificity