Principles of statistical learning

James Scott

ECO 395M: Data Mining and Statistical Learning

Outline

1. Introduction
2. Parametric vs nonparametric models: KNN
3. Measuring accuracy
4. Out-of-sample predictions
5. Train/test splits
6. Bias-variance tradeoff
7. A more careful look at measuring model accuracy

Introduction to predictive modeling

The goal is to predict a target variable (y) with feature variables (x).

- Zillow: predict price (y) using a house's features (x = size, beds, baths, age, ...)
- Citadel: predict next month's S&P (y) using this month's economic indicators (x = unemployment, GDP growth rate, inflation, ...)
- MD Anderson: predict a patient's disease progression (y) using his or her clinical, demographic, and genetic indicators (x)
- Etc.

In data mining/ML/AI, this is called "supervised learning." In intro Prob/Stats, we saw a simple example (OLS with one x feature)

Introduction to predictive modeling

A useful way to frame this problem is to postulate that:

$$y_i = f(x_i) + e_i$$

- y_i is a numerical *outcome* or *target* variable
- $x_i = (x_{i1}, x_{i2}, ...x_{iP})$ is a vector of features
- f is an unknown function
- each pair (x_i, y_i) is called an "example"

Our main purpose is to:

- *learn* $f(x)$ from the observed data
- make predictions on new x points, $\hat{y} = f(x)$.

Note: this is regression; *classification* (when y is a categorical rather than numerical) is a similar problem that we'll discuss later.

Example: predicting electricity demand

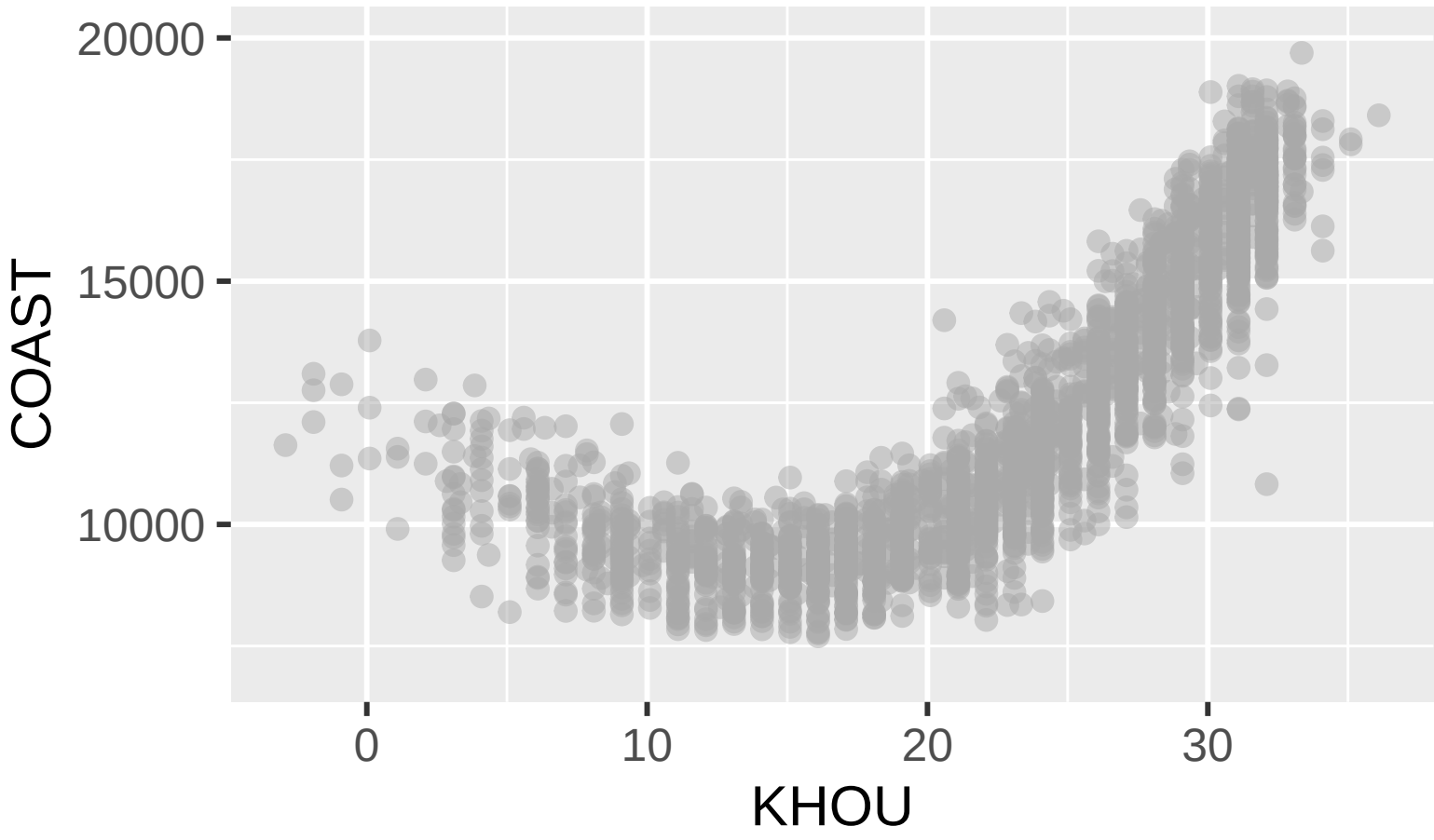
ERCOT operates the electricity grid for 75% of Texas by area.

Example: predicting electricity demand

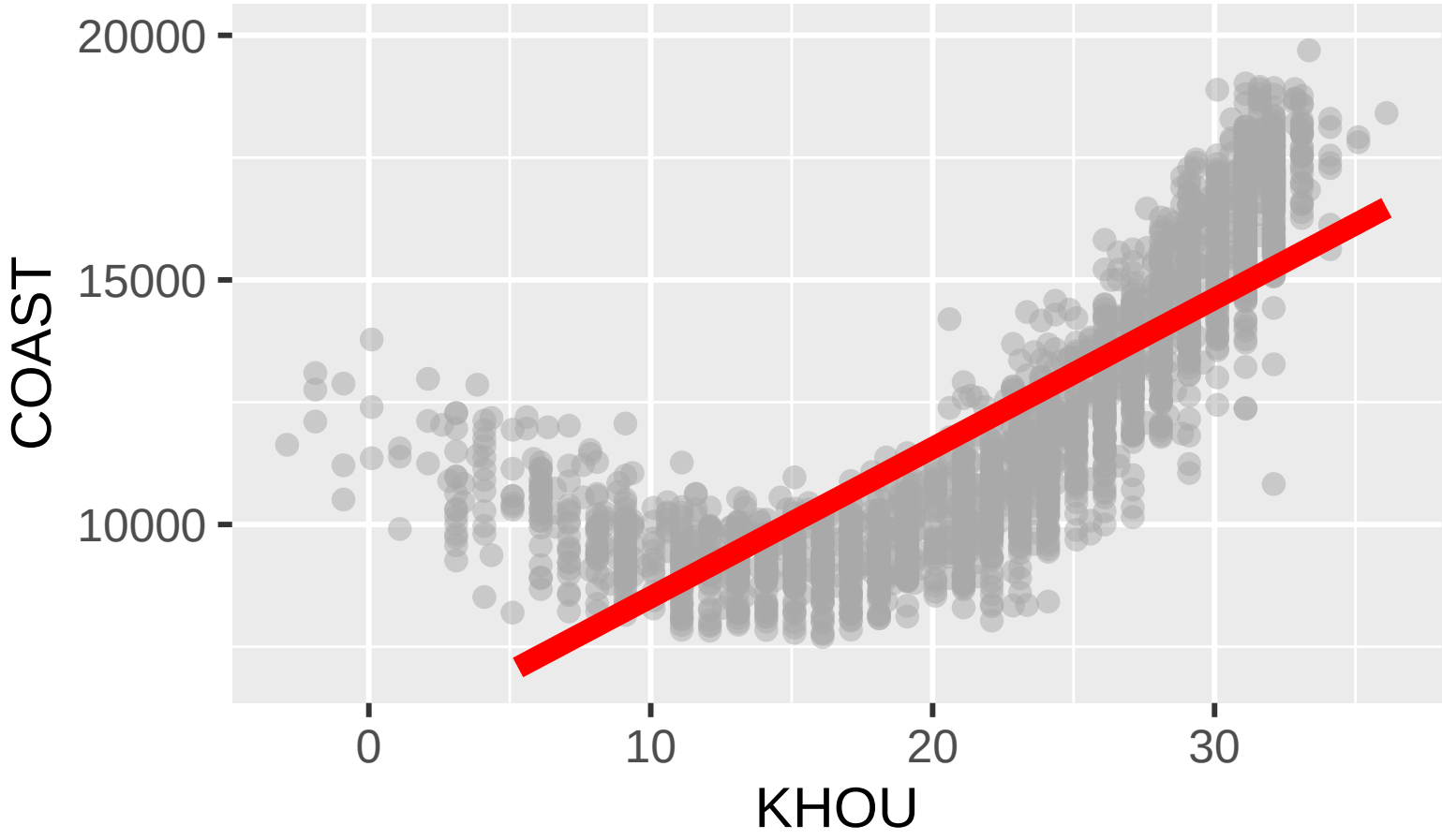
The 8 ERCOT regions are shown at left. We'll focus on a basic prediction task:

- y = demand (megawatts) in the Coast region at 3 PM, every day from 2010-2016.
- x = average daily temperature at Houston's Hobby Airport (degrees Celsius)

Demand versus temperature

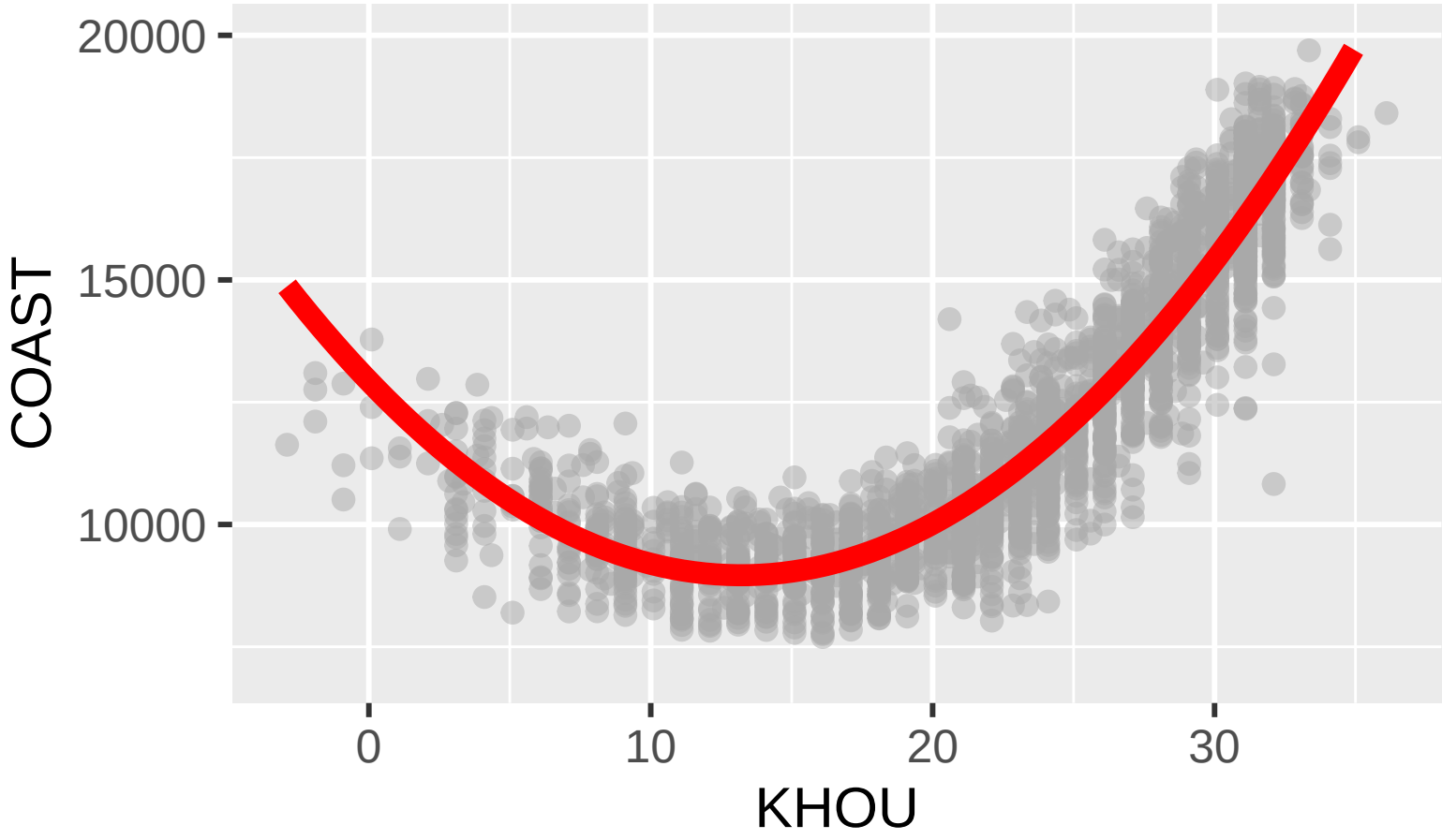


A linear model?



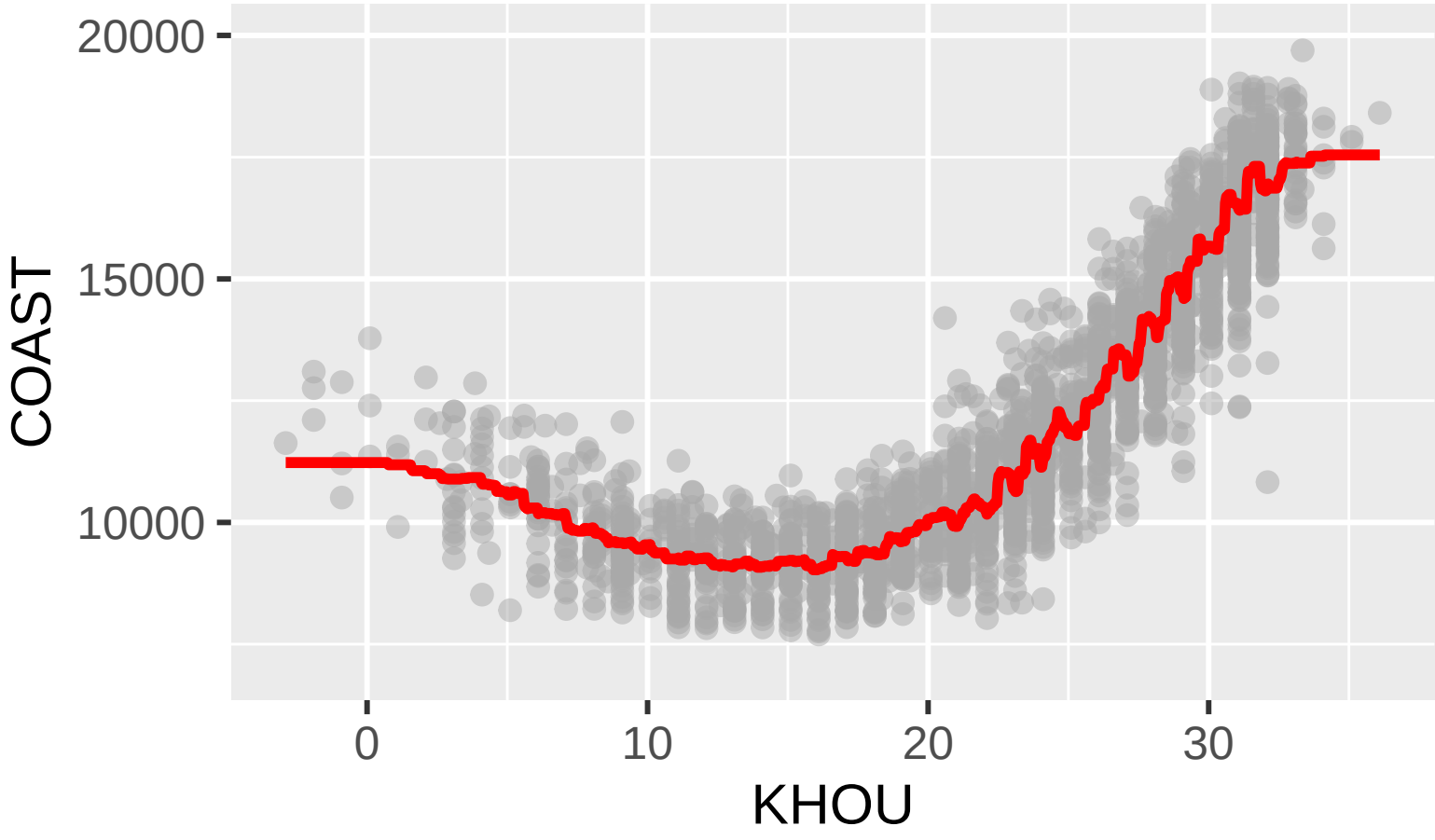
$$f(x) = \beta_0 + \beta_1 x$$

A quadratic model?



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

How about this model?



We can't write down an equation for this $f(x)$. But we can define it by its behavior! (If $x = 15$, what is $f(x)$? What about if $x = 30$?)

How do we estimate f?

Our *training data* consists of pairs

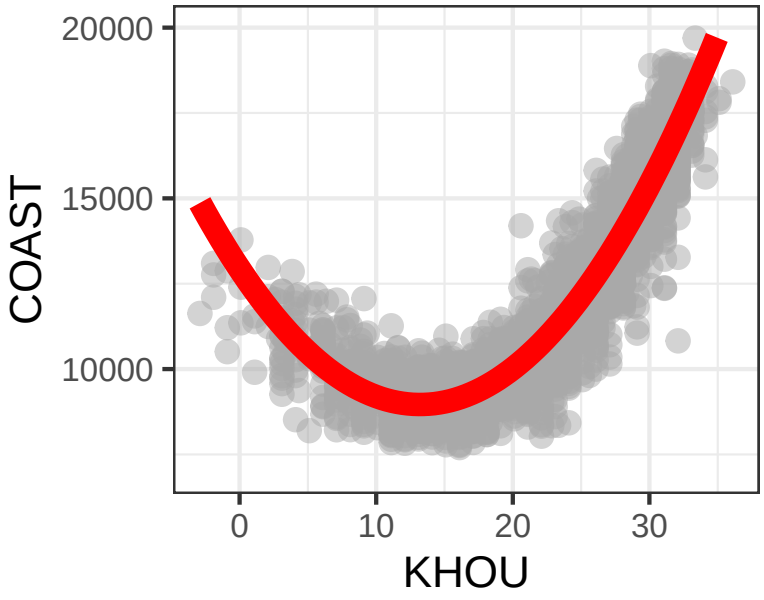
$$D_{\text{tr}} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

We then use some statistical method to estimate $f(x)$. Here "statistical" just means "we apply some recipe to the data."

There are two general families of strategy.

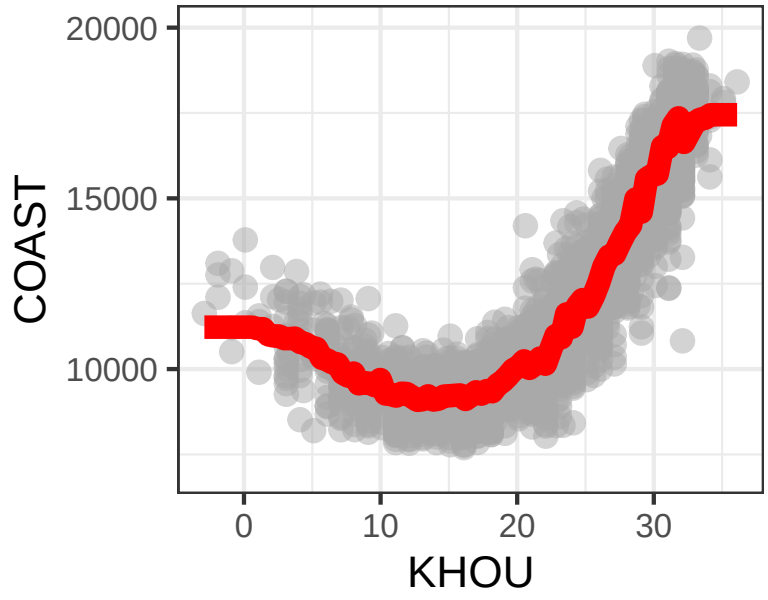
- Parametric models: assume a particular, restricted functional form (e.g. linear, quadratic, logs, exp)
- nonparametric models: flexible forms not easily described by simple math functions. (Requires you to give up the grade-school idea that a function is an equation you can write down.)

A quick comparison



Parametric:

- polynomial model
- $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$



Nonparametric:

- k-nearest neighbors
- $f(x) =$ average y value of the 50 points closest to x

Estimating a parametric model: three steps

1. Choose a functional form of the model, e.g.

$$f(x) = \beta_0 + \beta_1 x$$

2. Choose a *loss function* that measures the difference between the model predictions $f(x)$ and the actual outcomes y. E.g. least squares:

$$\begin{aligned} L(\beta_0, \beta_1) &= \sum_{i=1}^{N} (y_i - f(x_i))^2 \\ &= \sum_{i=1}^{N} (y_i - \{\beta_0 + \beta_1 x_i\})^2 \end{aligned}$$

3. Find the parameters that minimize the loss function.

Estimating k-nearest neighbors

Suppose we have our training data in the form of (x_i, y_i) pairs. Now we want to predict y at some new point $x^\star$.

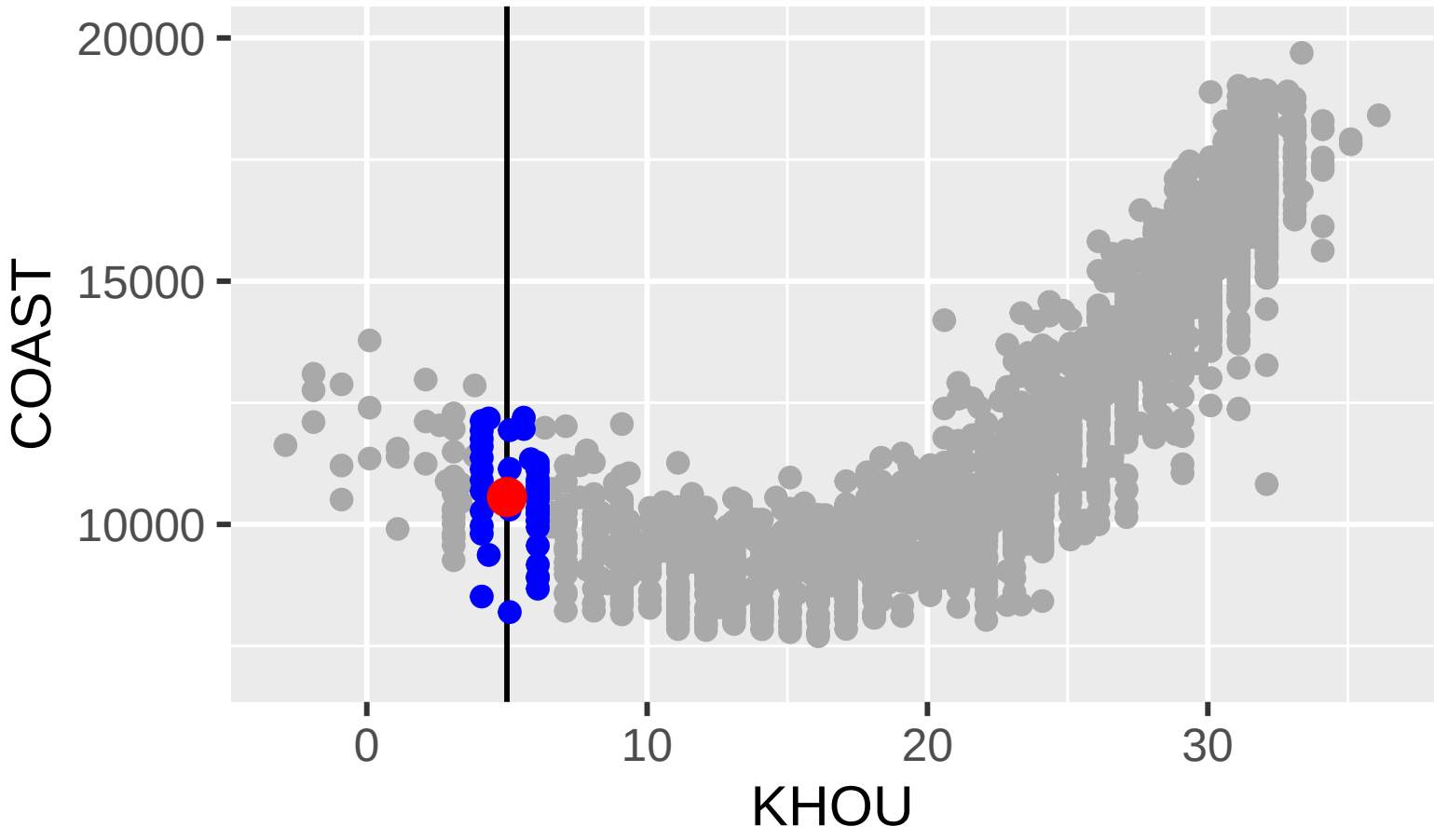
1. Pick the K points in the training data whose x_i values are closest to $x^\star$. Call this neighborhood $\mathcal{N}_K(x^\star)$.
2. Average the y_i values for those points and use this average to estimate $f(x^\star)$:

$$\hat{f}(x^\star) = \frac{1}{K} \sum_{i:x_i \in \mathcal{N}_K(x^\star)} y_i$$

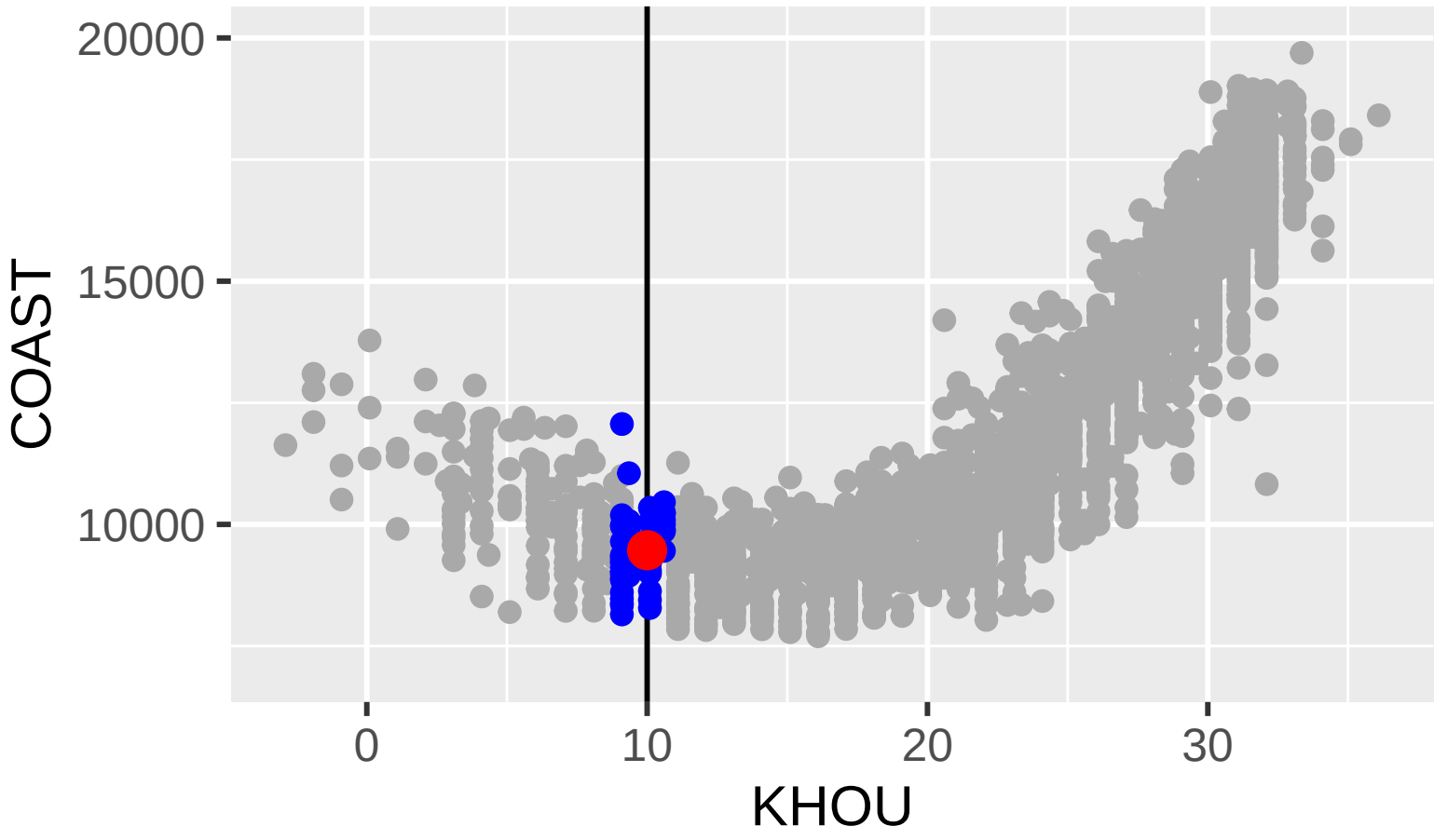
There are **no explicit parameters** (i.e. β's) to estimate.

- Rather, the estimate for $f(x)$ is defined by a particular *algorithm* applied to the data set.
- Note for the mathematically rigorous: $f(x)$ is defined *point-wise*, that is, by applying the same recipe at any point x.)

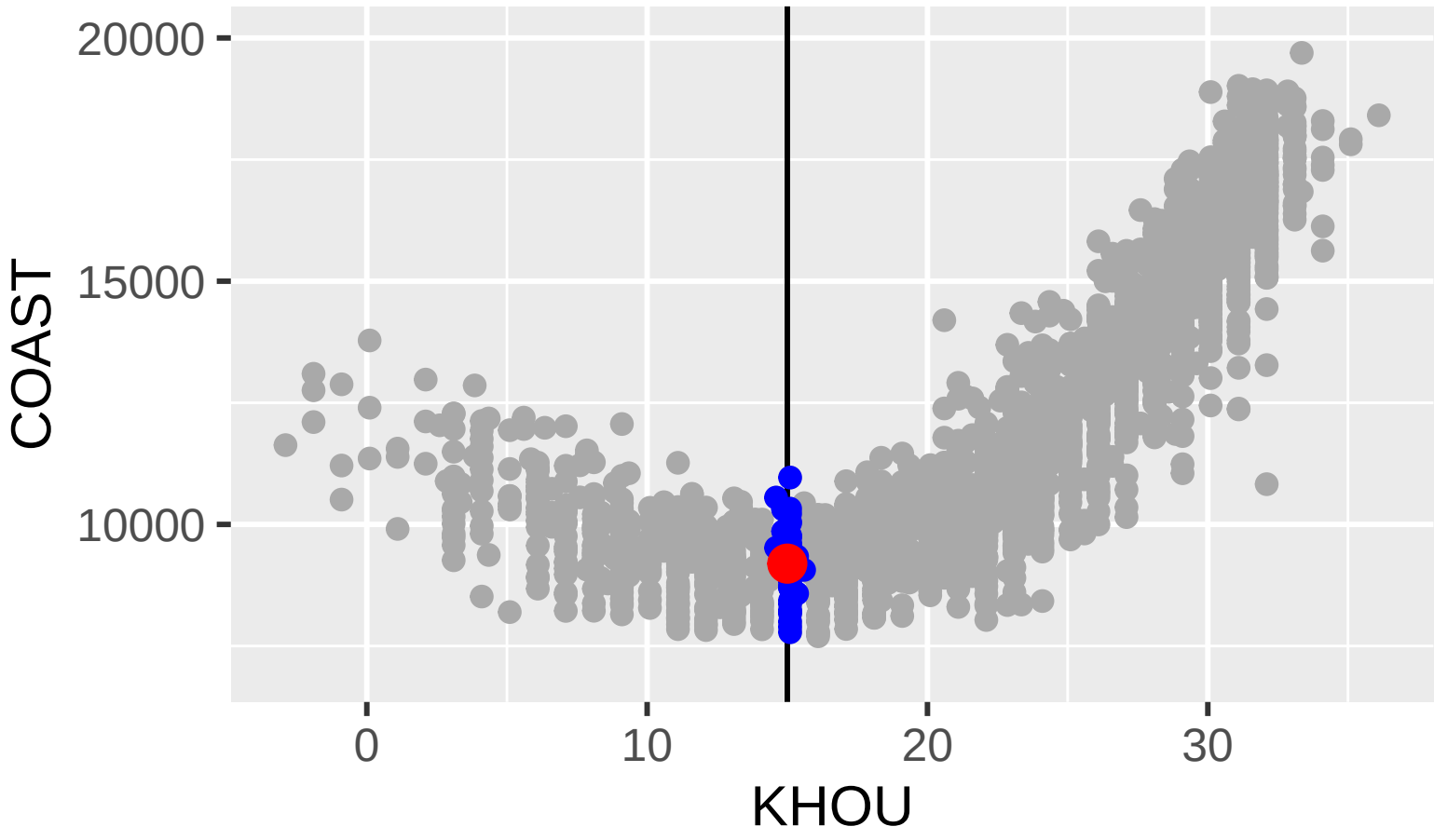
At x=5



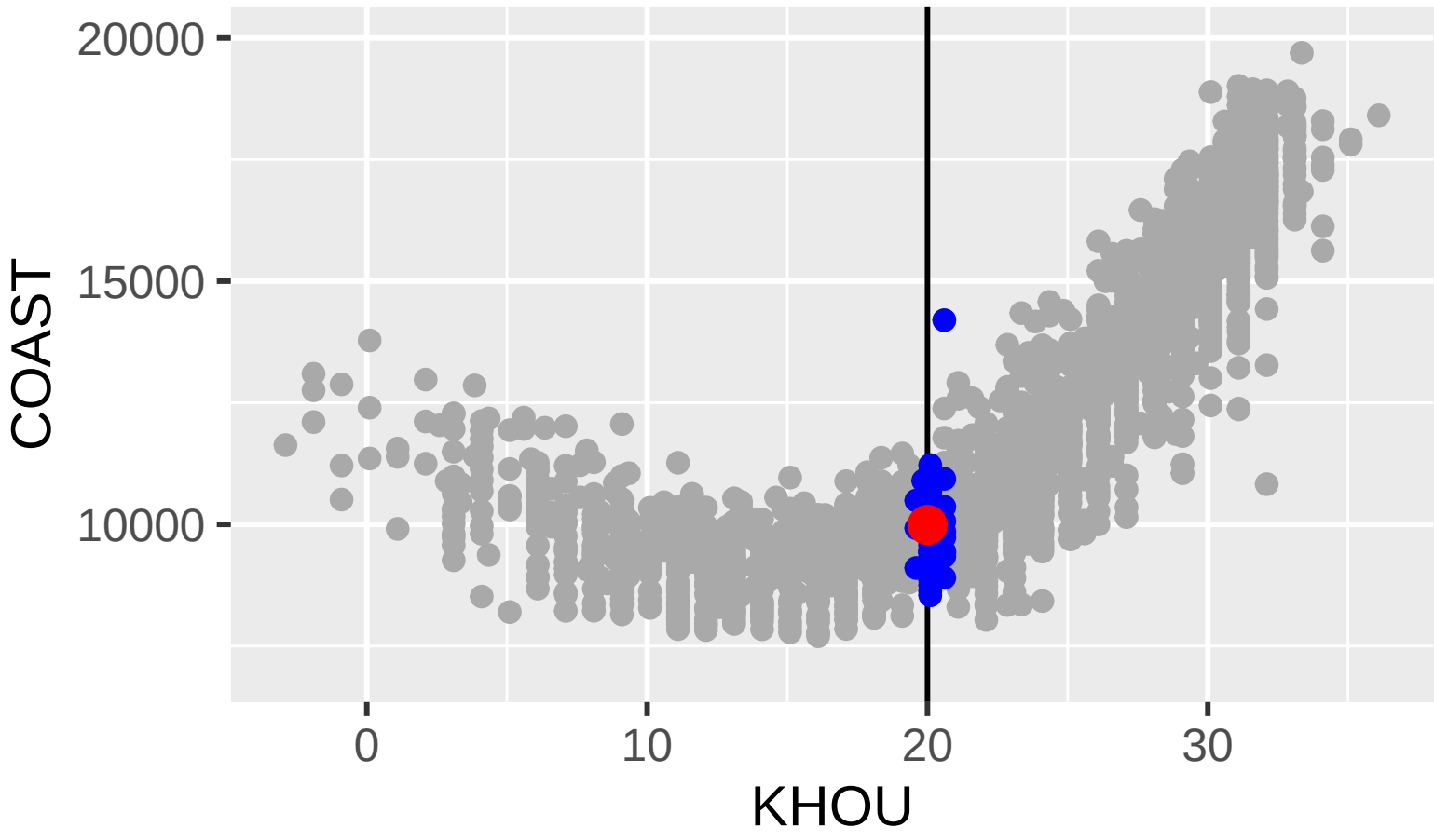
At x=10



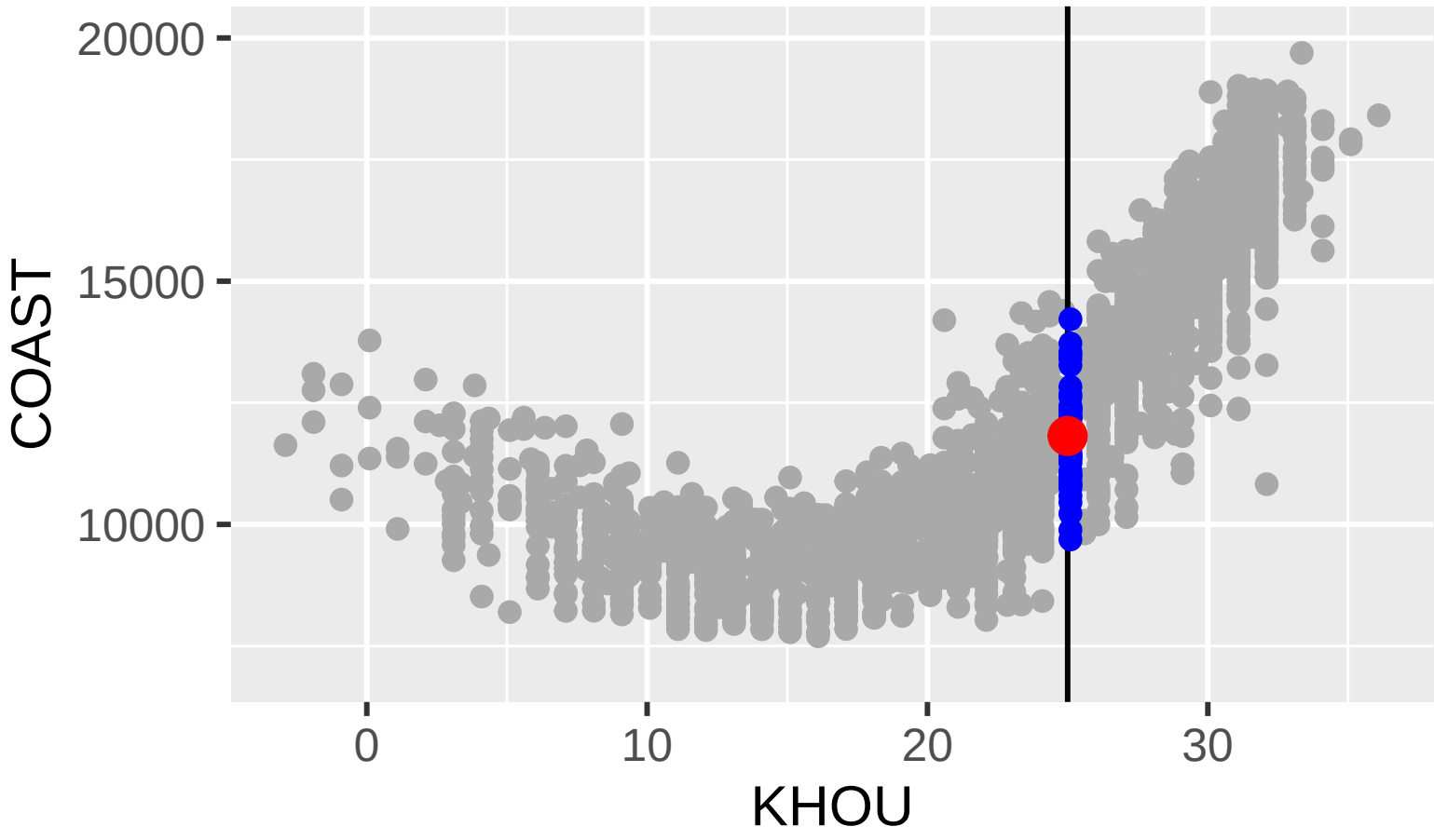
At x=15



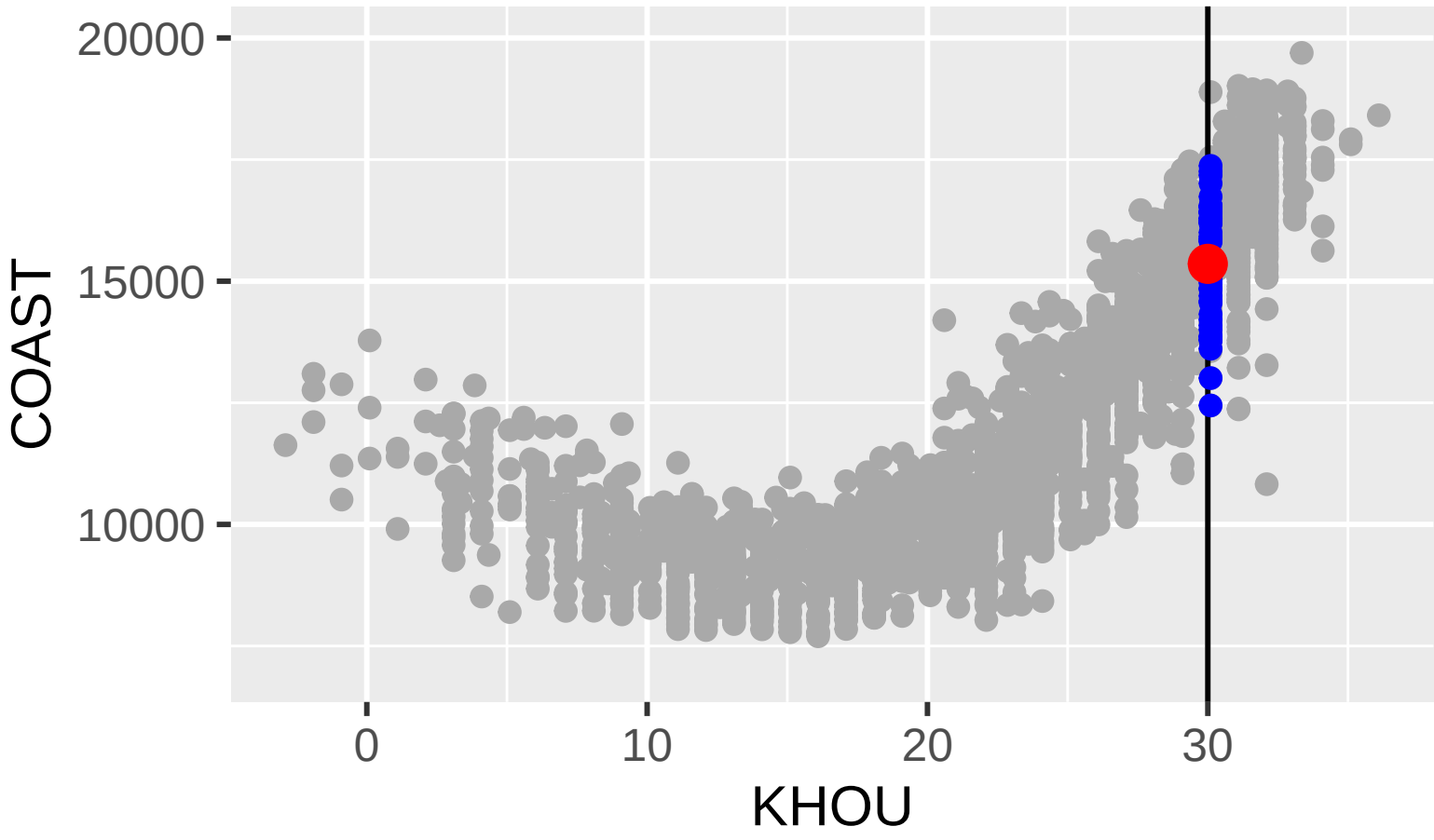
At x=20



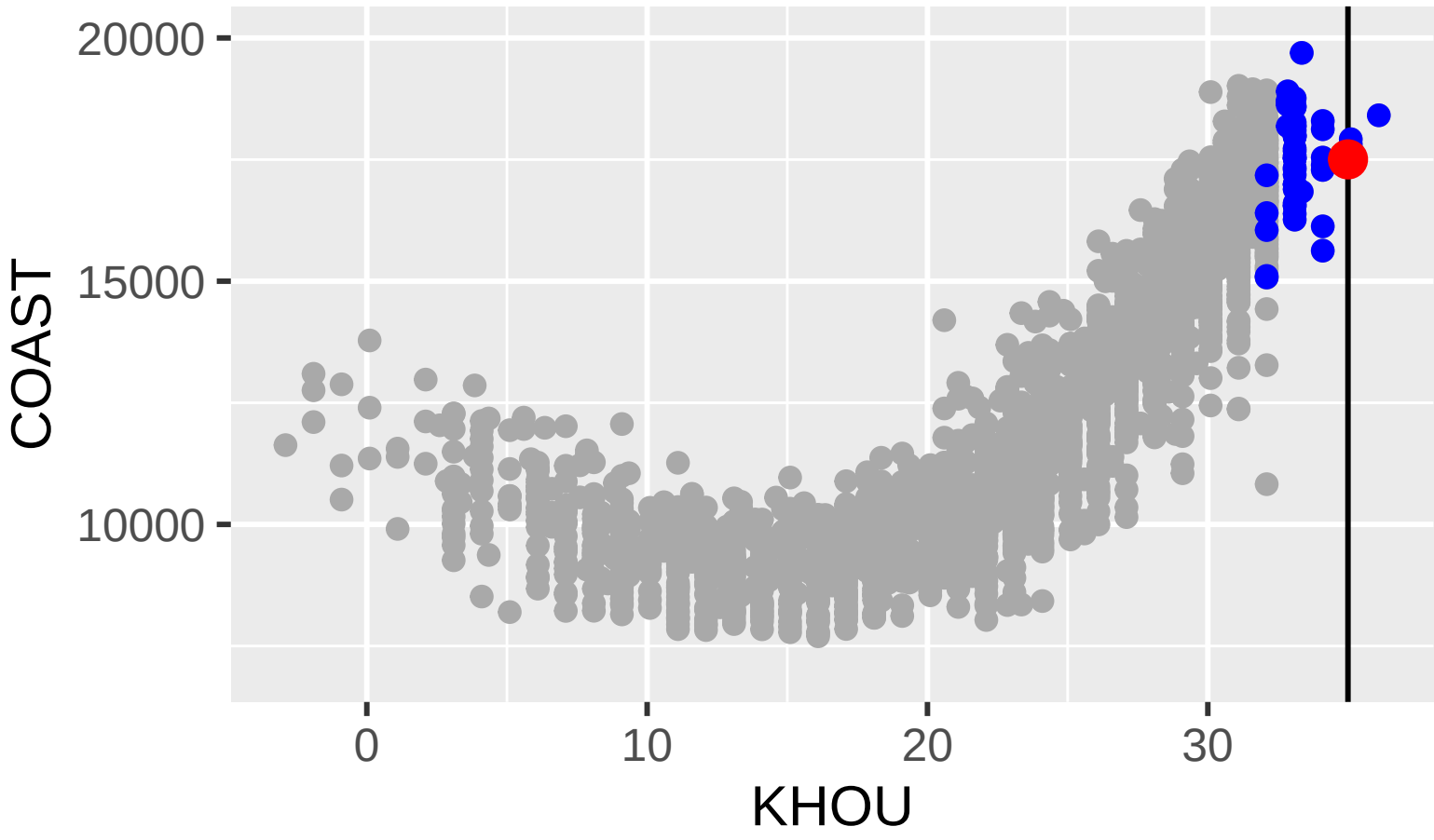
At x=25



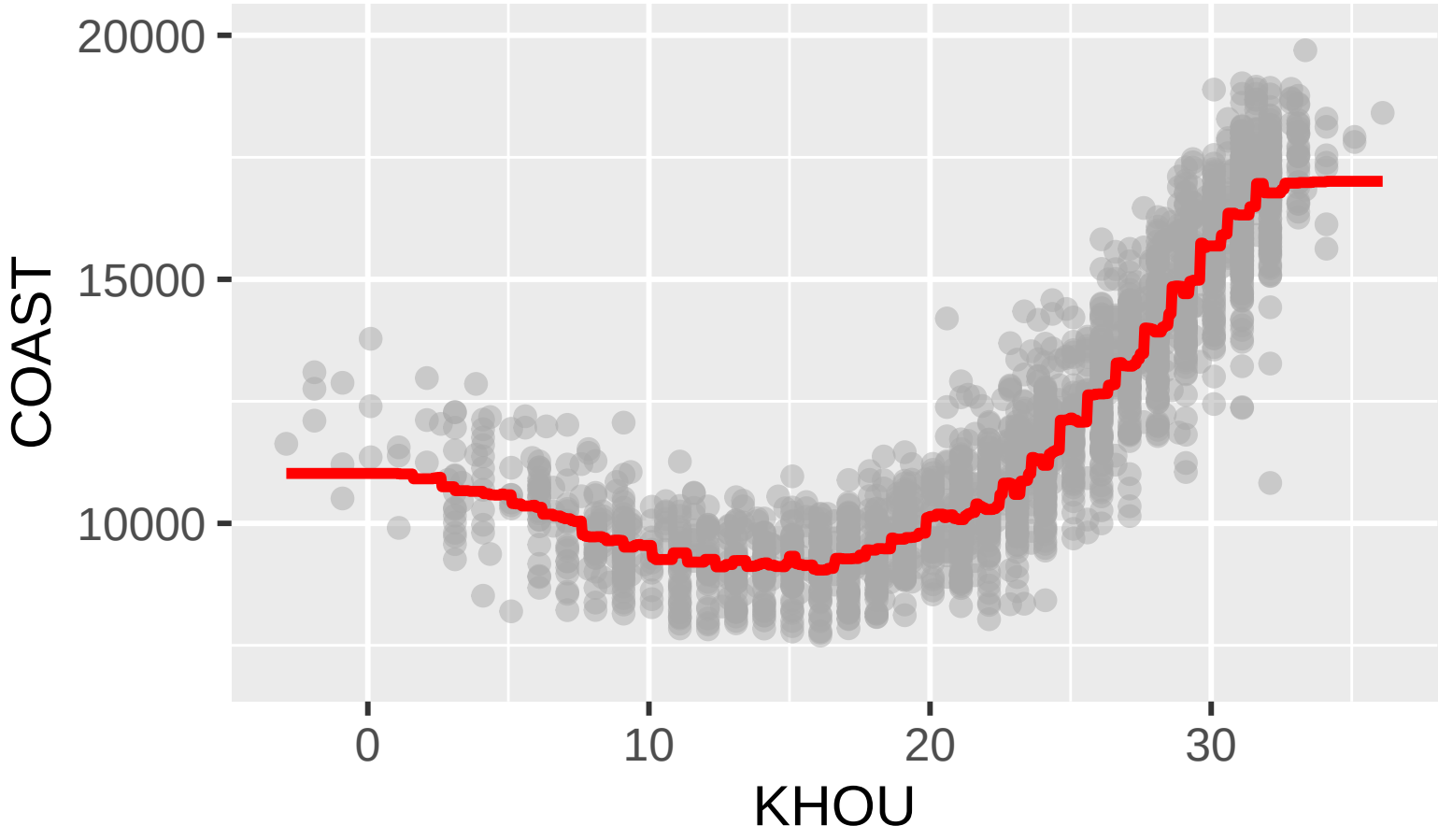
At x=30



At x=35



The predictions across all x values



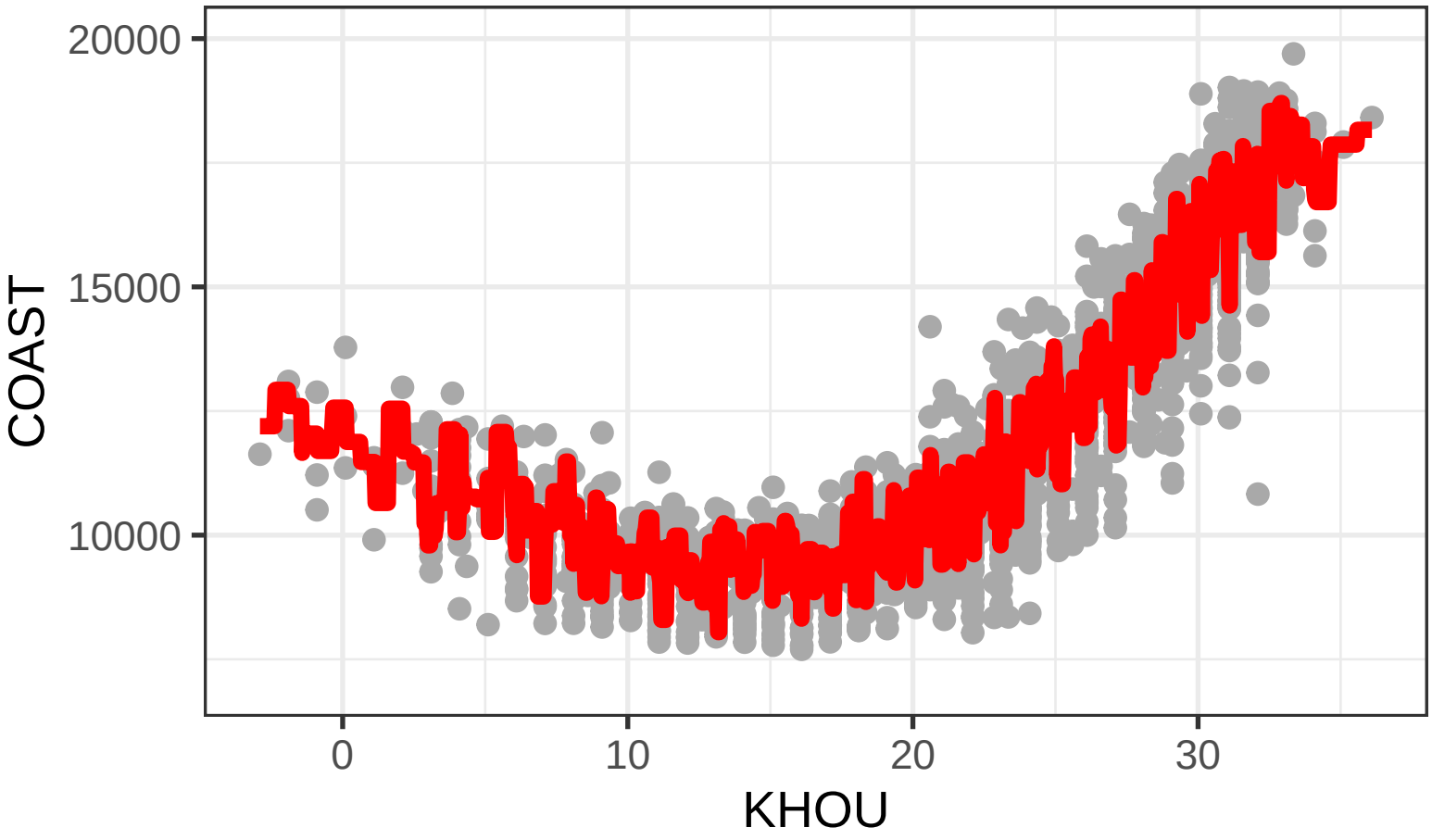
Two questions

This procedure raises two obvious questions:

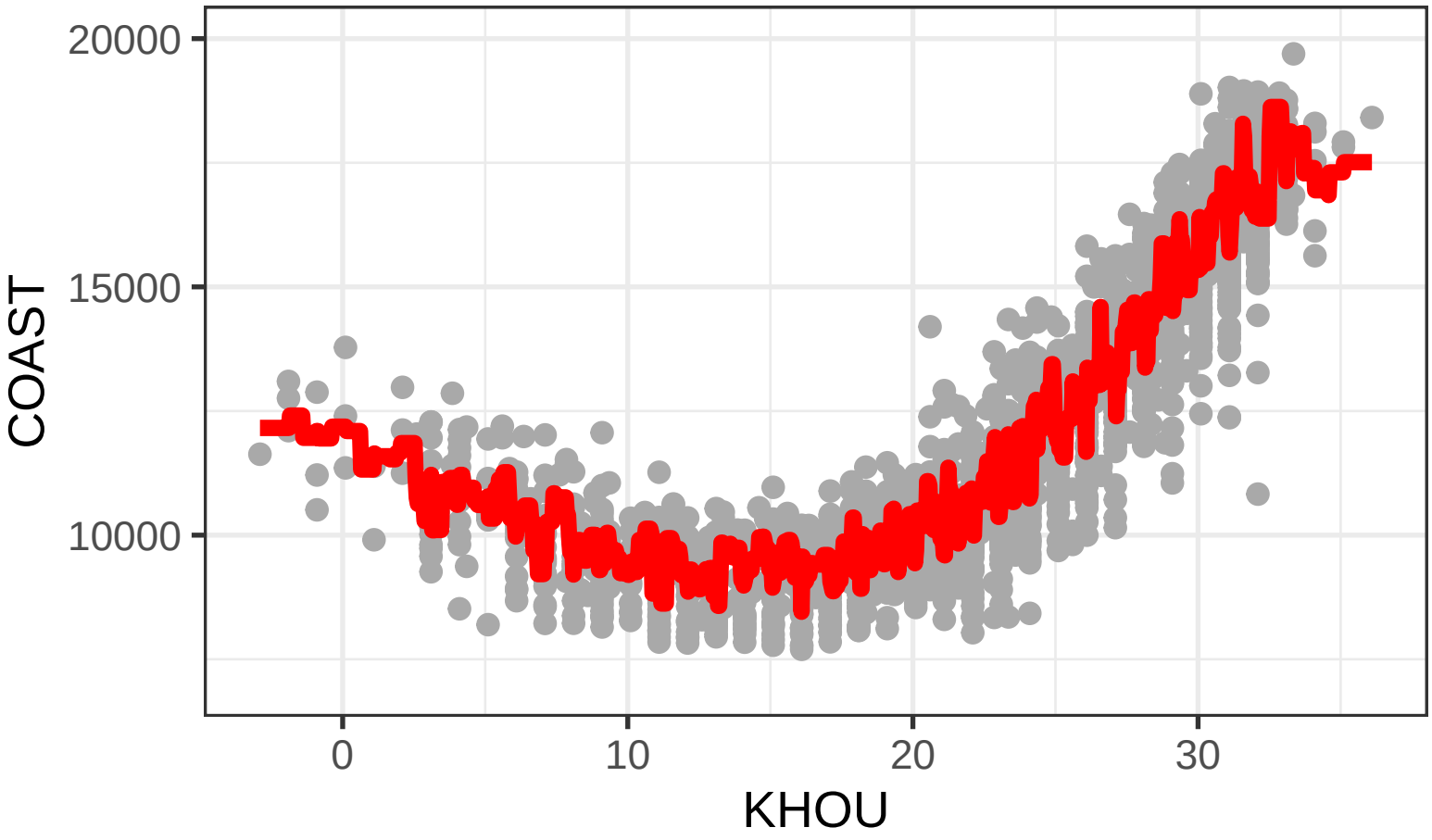
1. So why average the nearest $K = 50$ neighbors? Why not $K = 2$, or $K = 200$?
2. And if we're free to pick any value of K we like, how should we choose?

Let's take the true coder's approach: mess with stuff and see what happens!

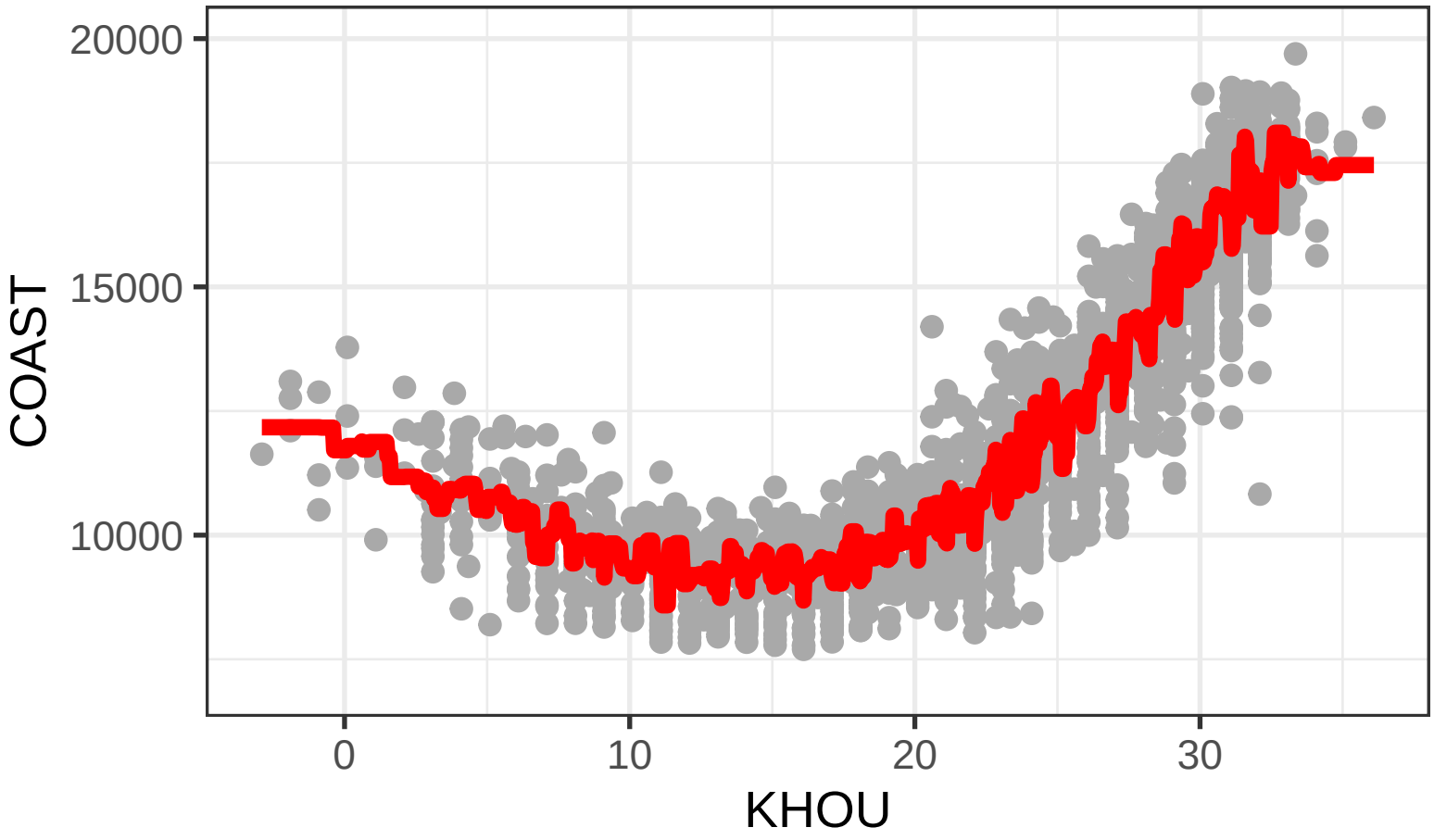
K=2



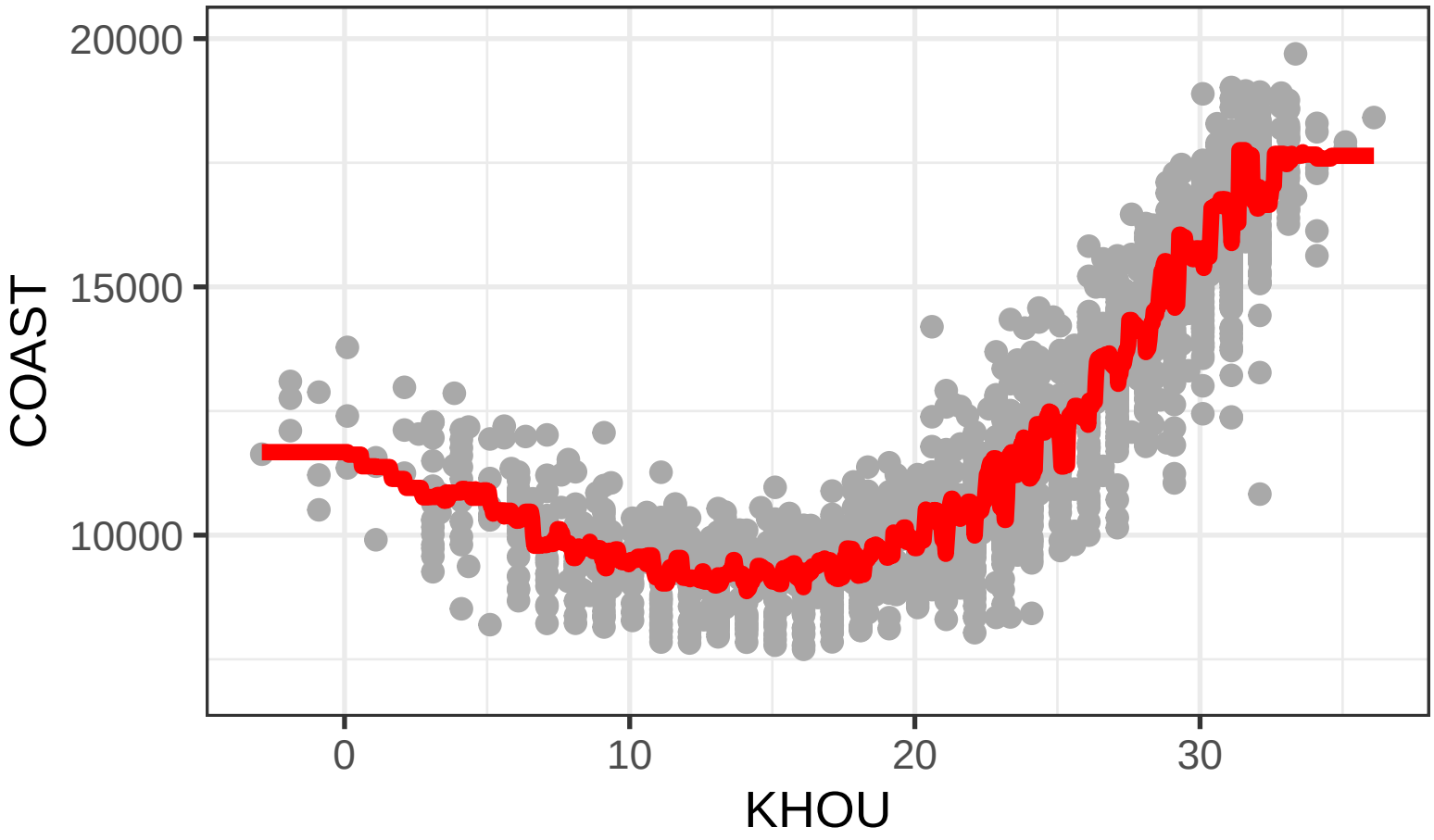
K=5



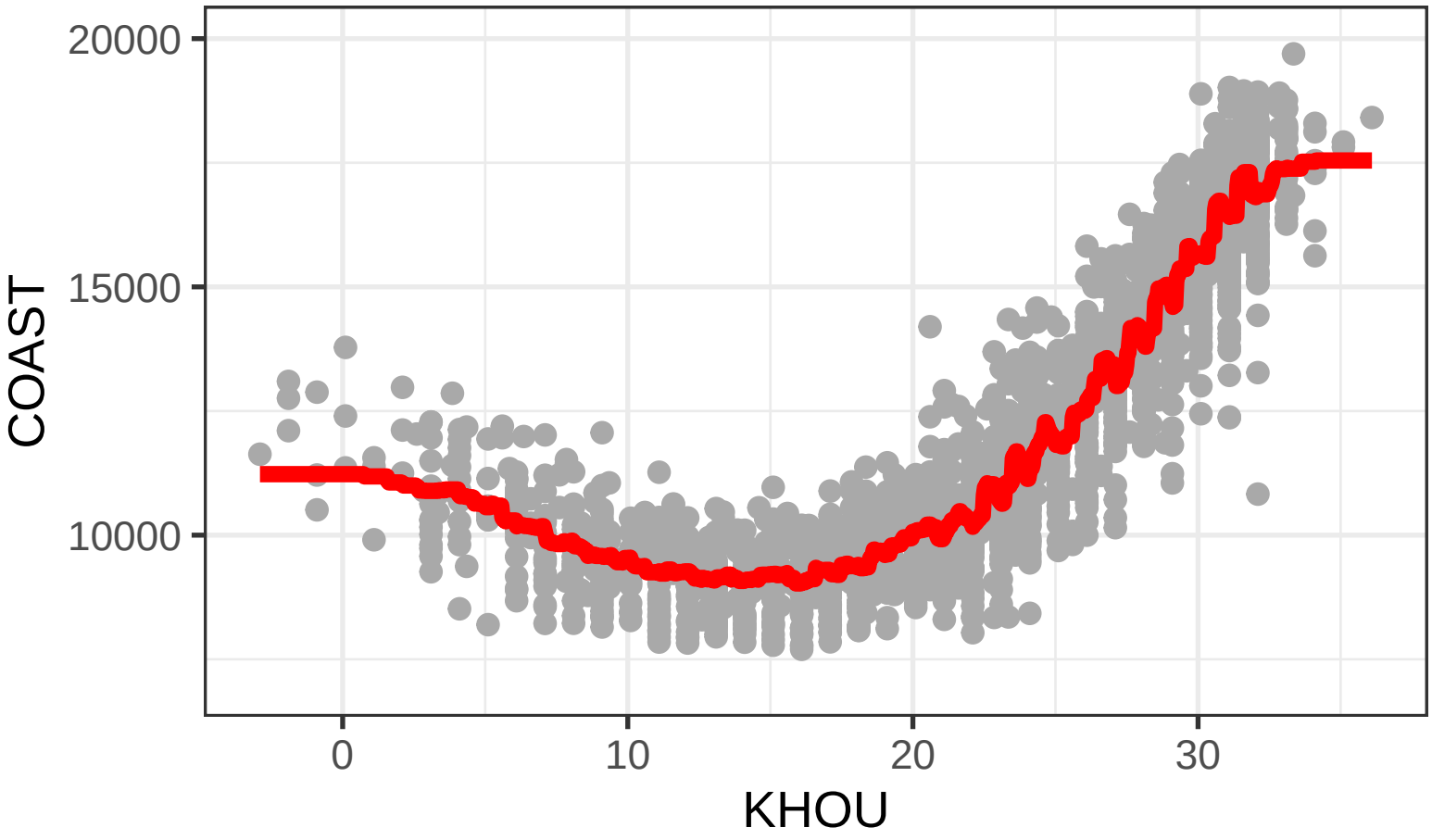
K=10



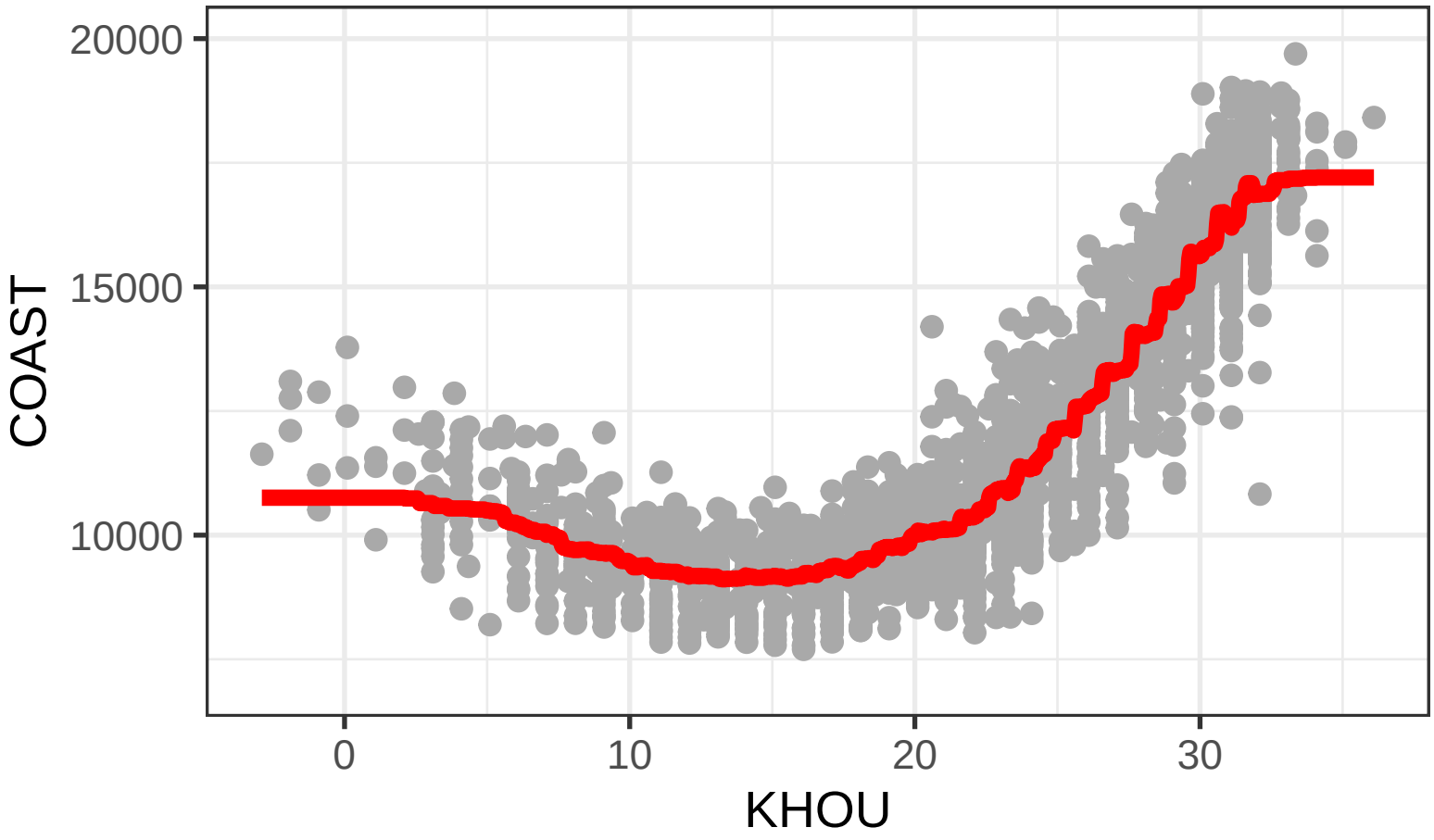
K=20



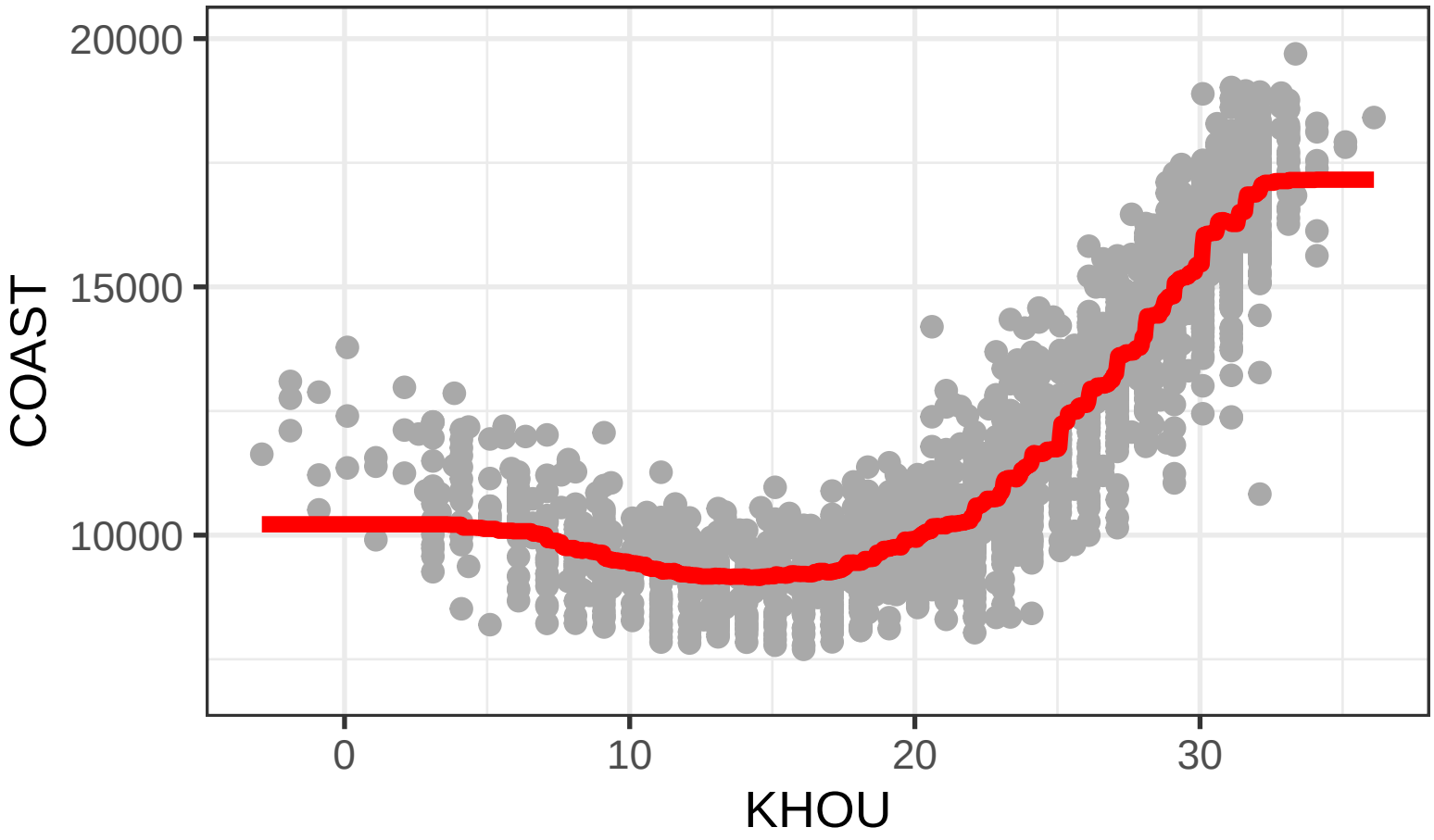
K=50



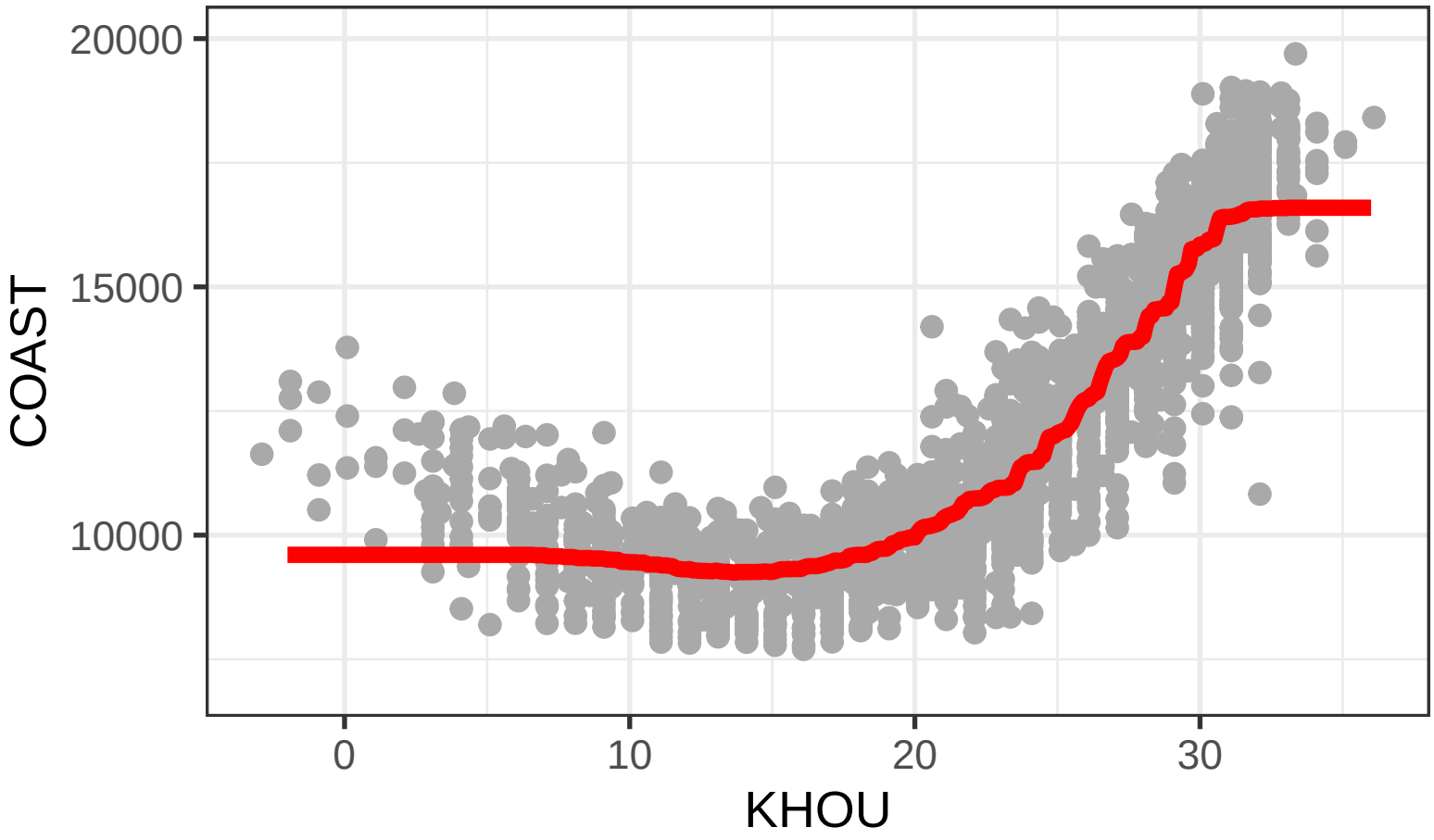
K=100



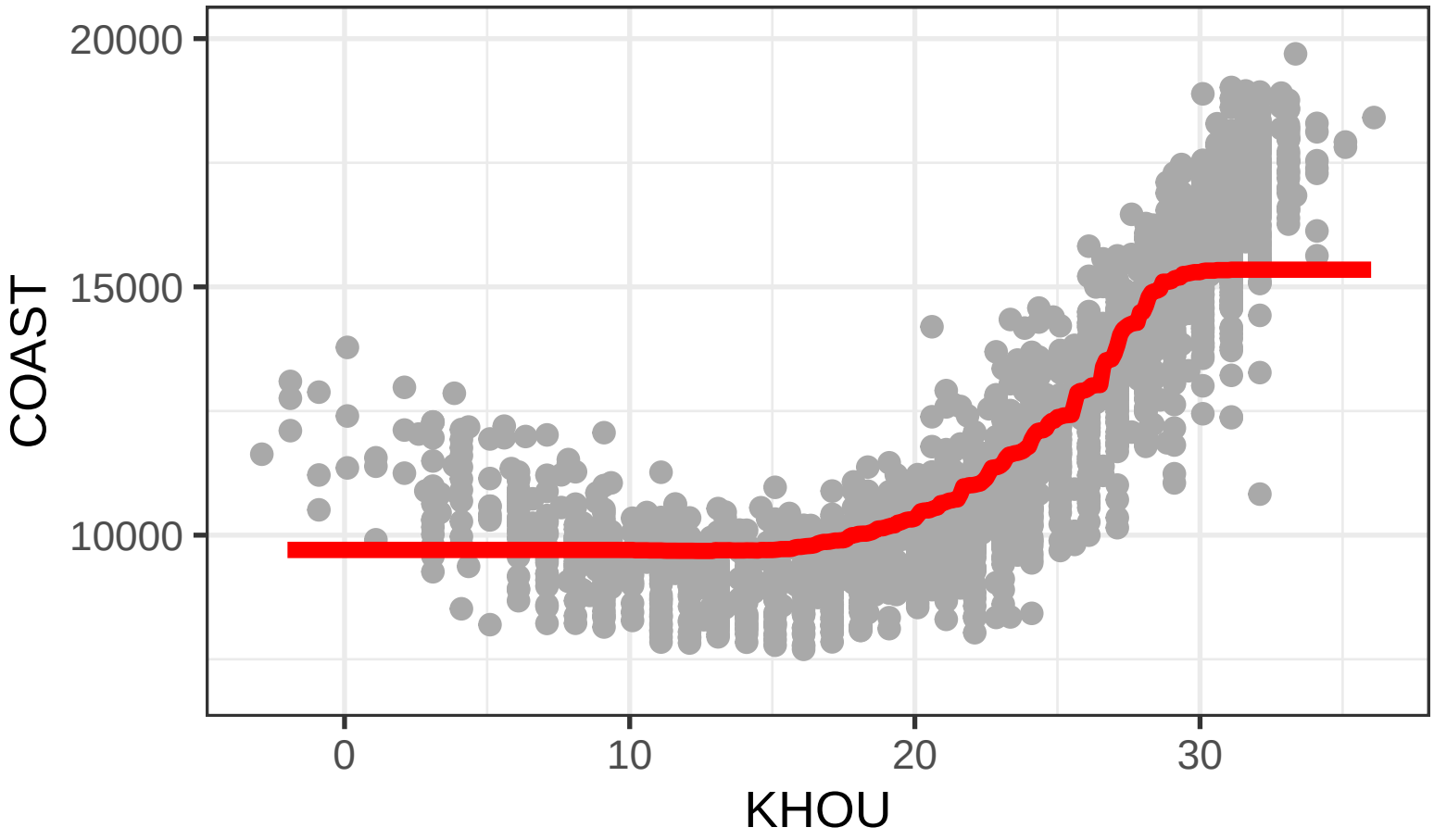
K=200



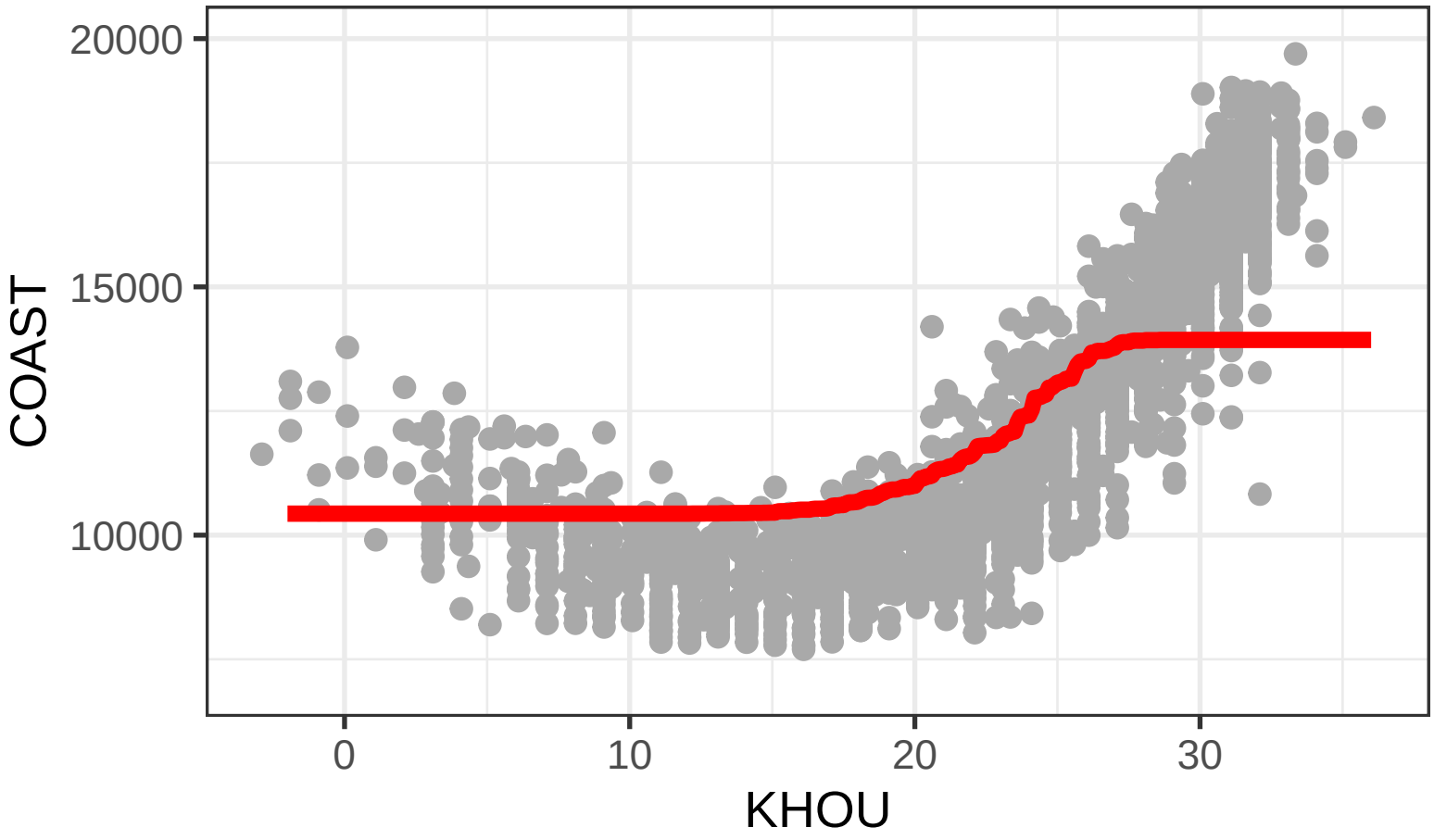
K=500



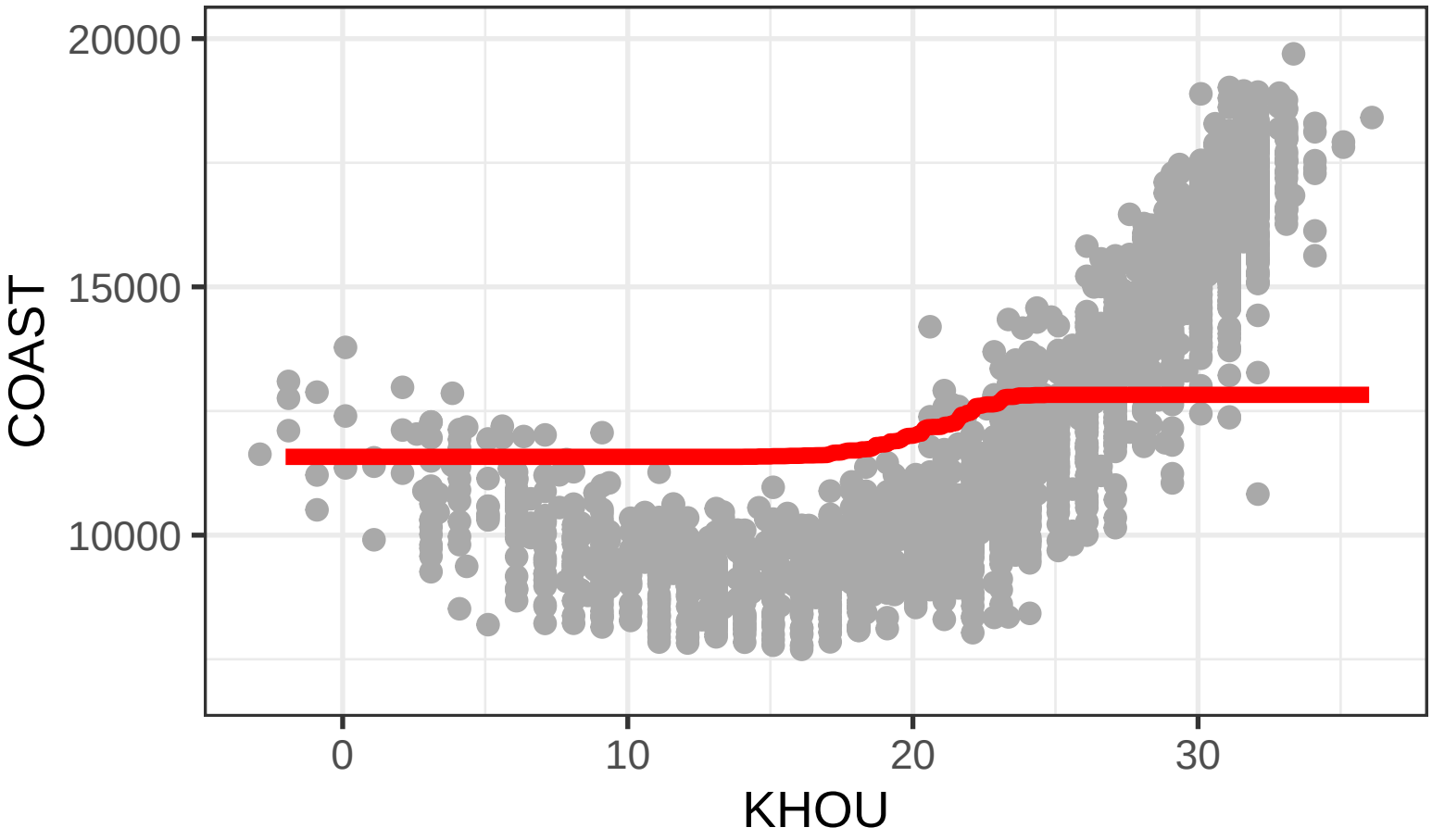
K=1000



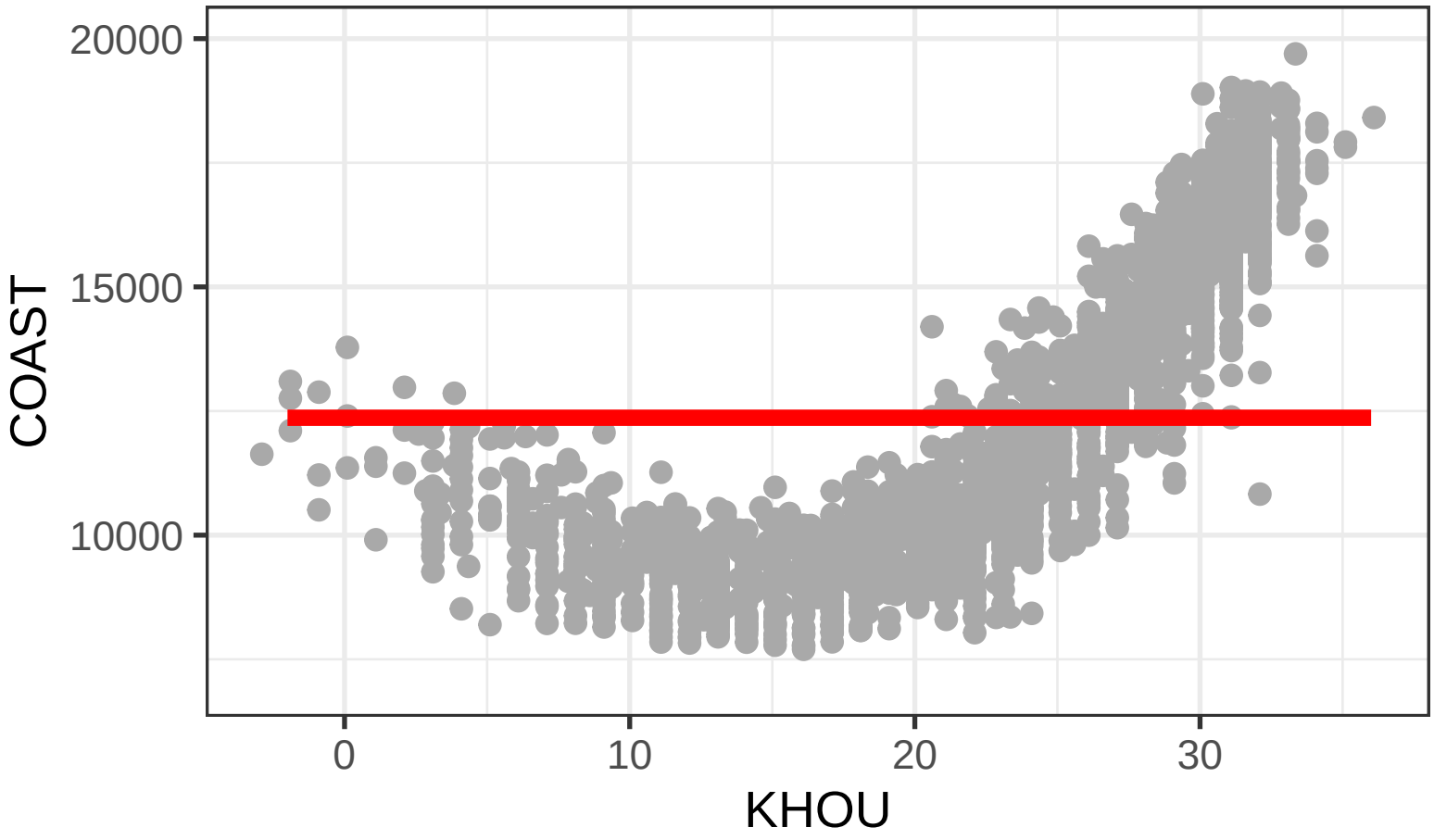
K=1500



K=2000



K=2357



Complexity, generalization, and interpretion

Smaller values of K give more flexible, but less stable function estimates:

- they can capture very fine-scale structure in $f(x)$, because they're only averaging points from a small neighborhood...
- but they can also confuse noise for signal!

Larger values of K give less flexible, but more stable function estimates:

- they can't adapt as much to wiggles in $f(x)$, because they're averaging points over a larger neighborhood.
- but this makes them less prone to confusing noise for signal.

You probably got the sense from the pictures that there's a "happy medium" somewhere. **How should we find it?**

Measuring accuracy

Let's go with a simple principle: choose the model that makes the most accurate predictions, on average.

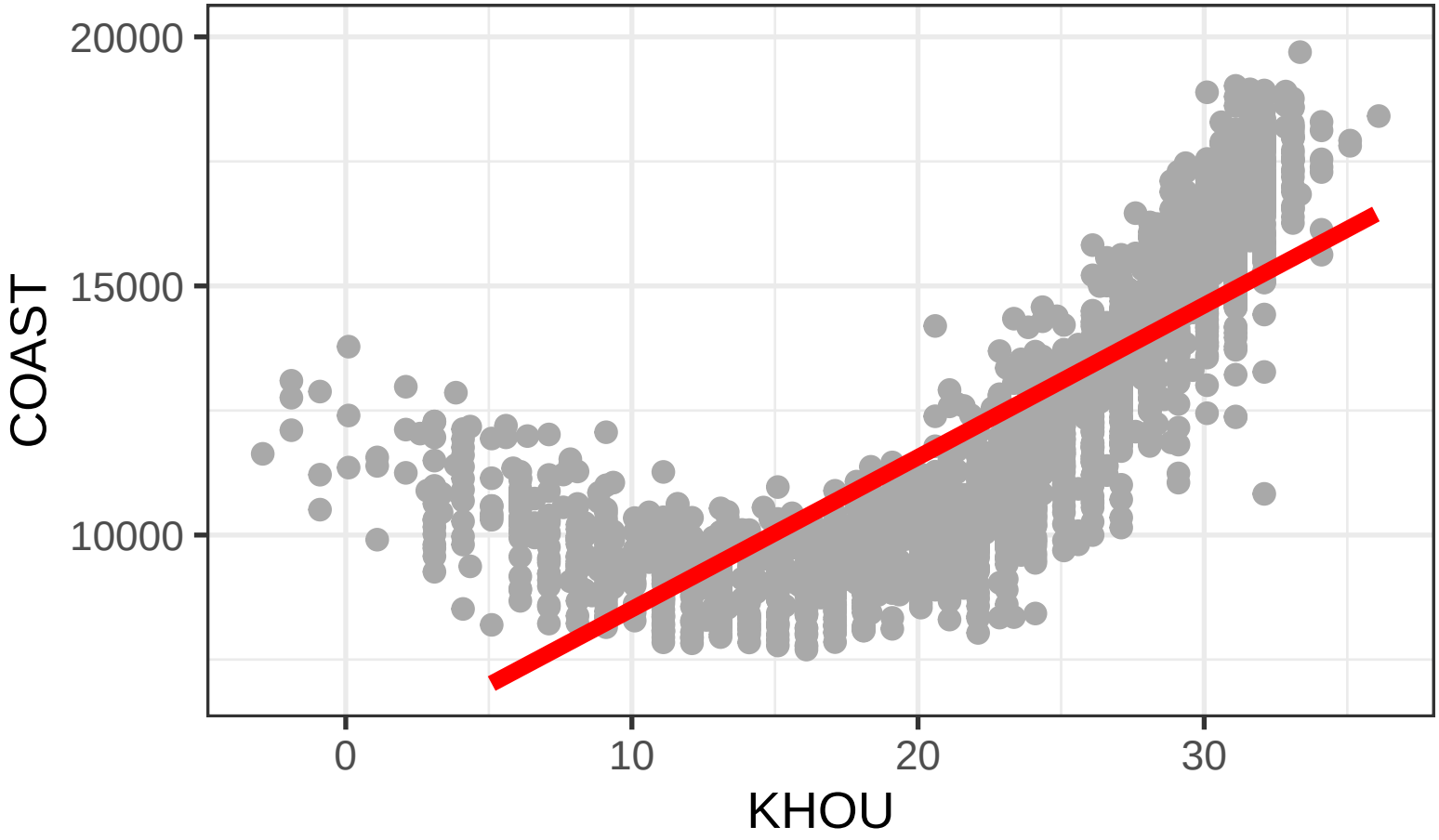
A standard measure of (in)accuracy is the root mean-squared error:

$$\mathrm{RMSE}_{\mathrm{in}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2}$$

This measures, on average, how large are the errors made by the model on the training data. (OLS minimizes this quantity over the set of linear functions.)

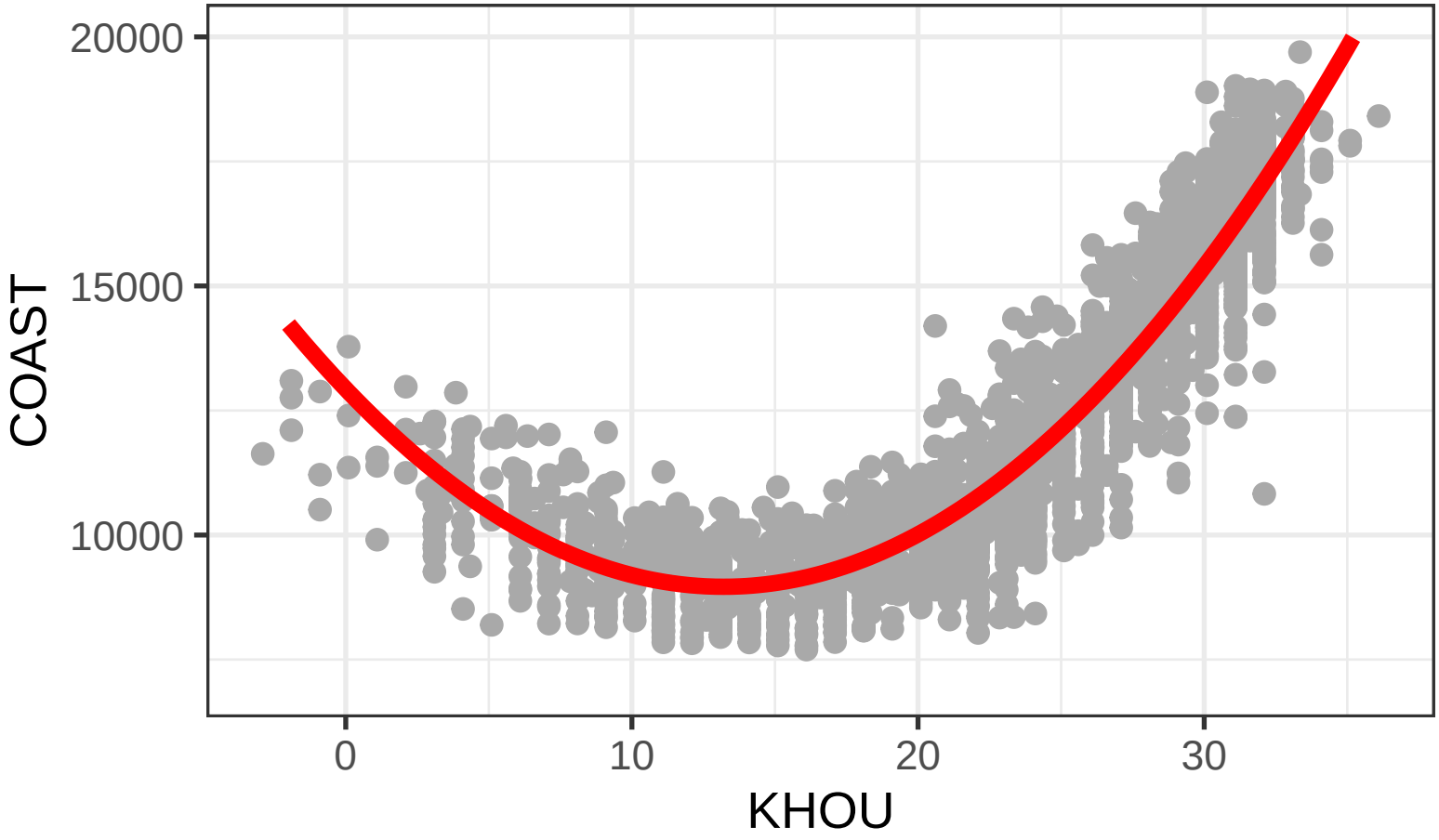
RMSE is just one possible error metric, but it's pretty popular. (You'll also see RMSE without the square root, or MSE, but it's hard to interpret.)

Measuring accuracy: linear vs. quadratric



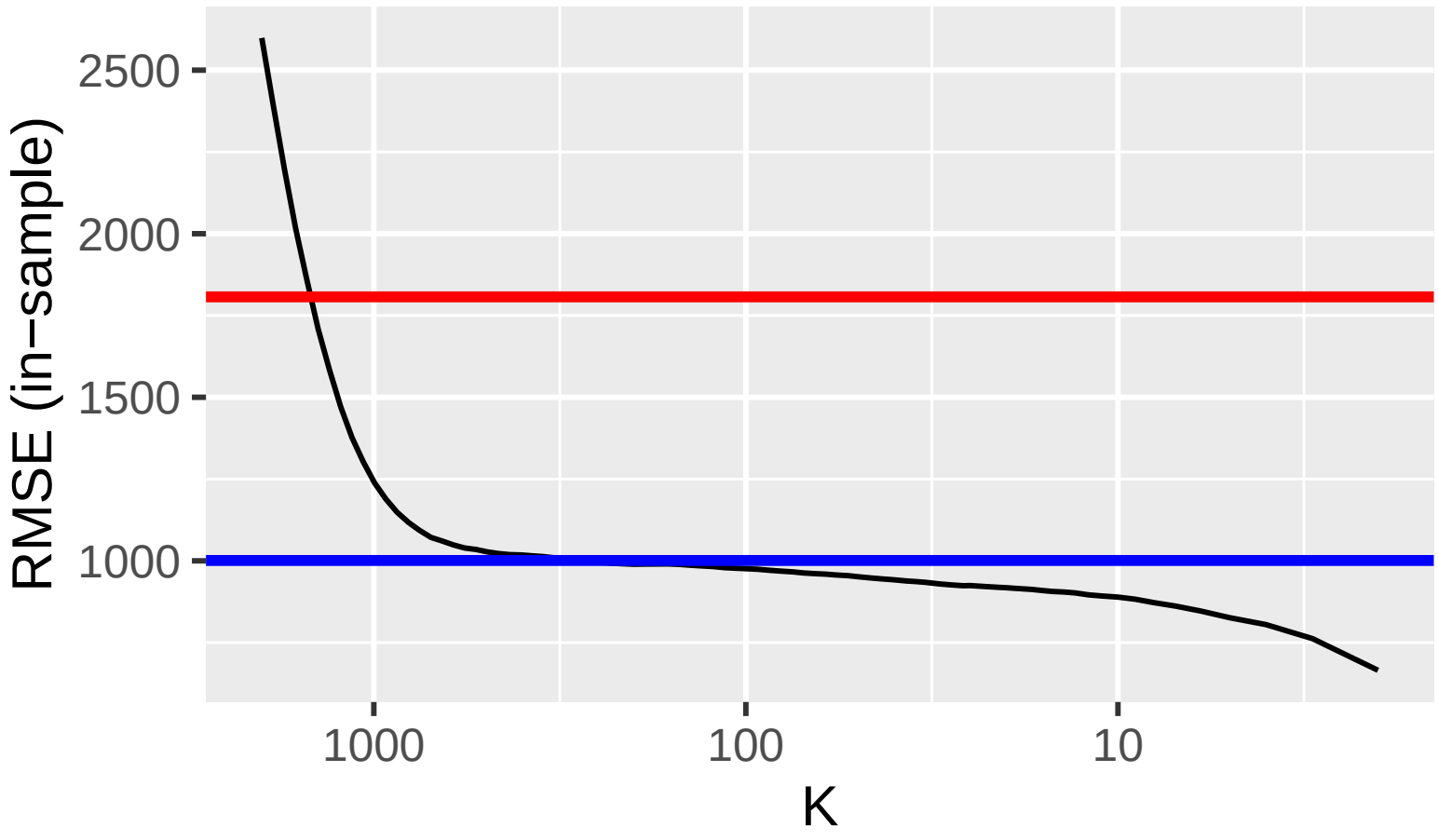
$RMSE = 1807$

Measuring accuracy: linear vs. quadratric

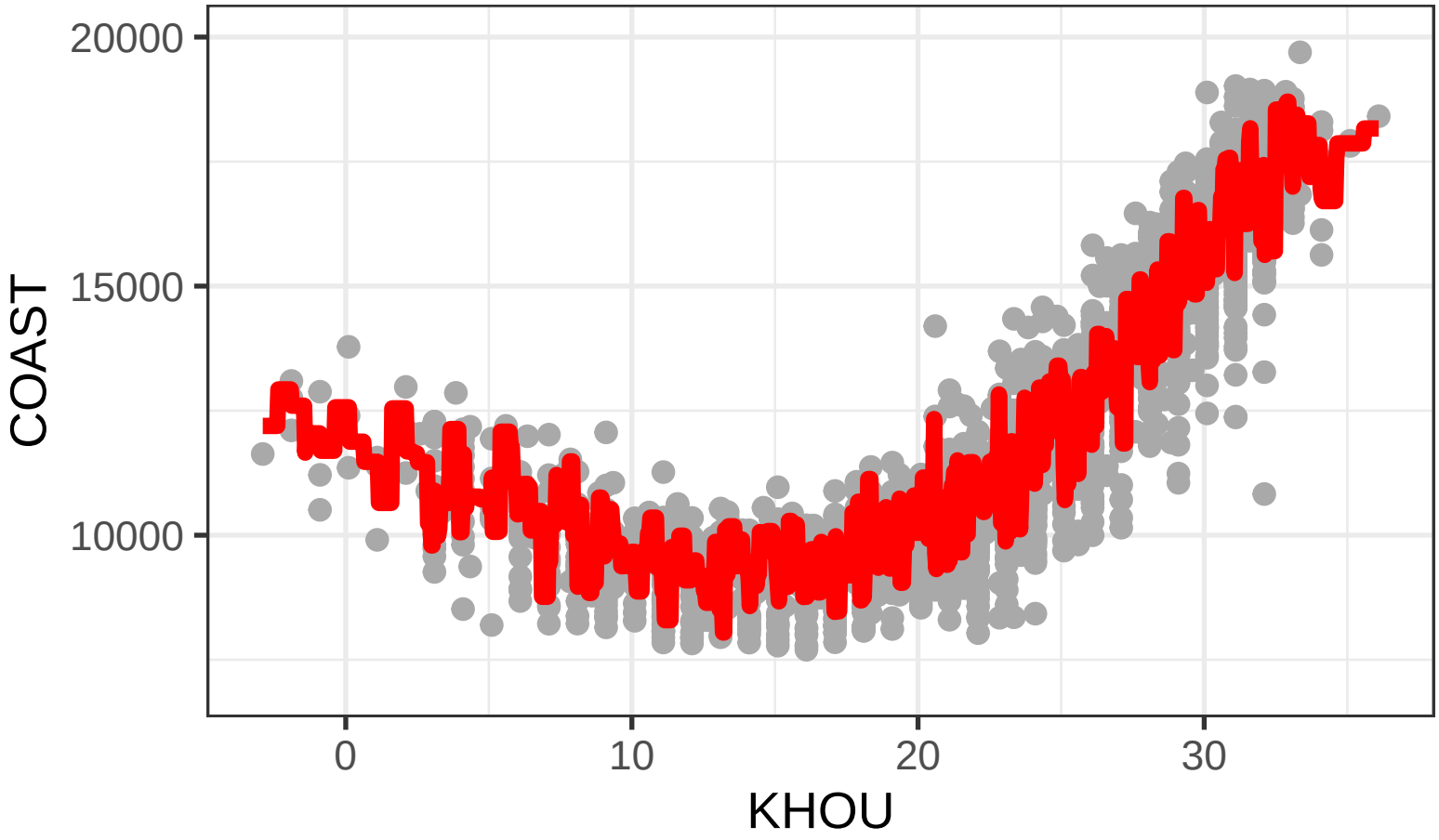


$$RMSE = 1001$$

Measuring accuracy: RMSE versus K for KNN

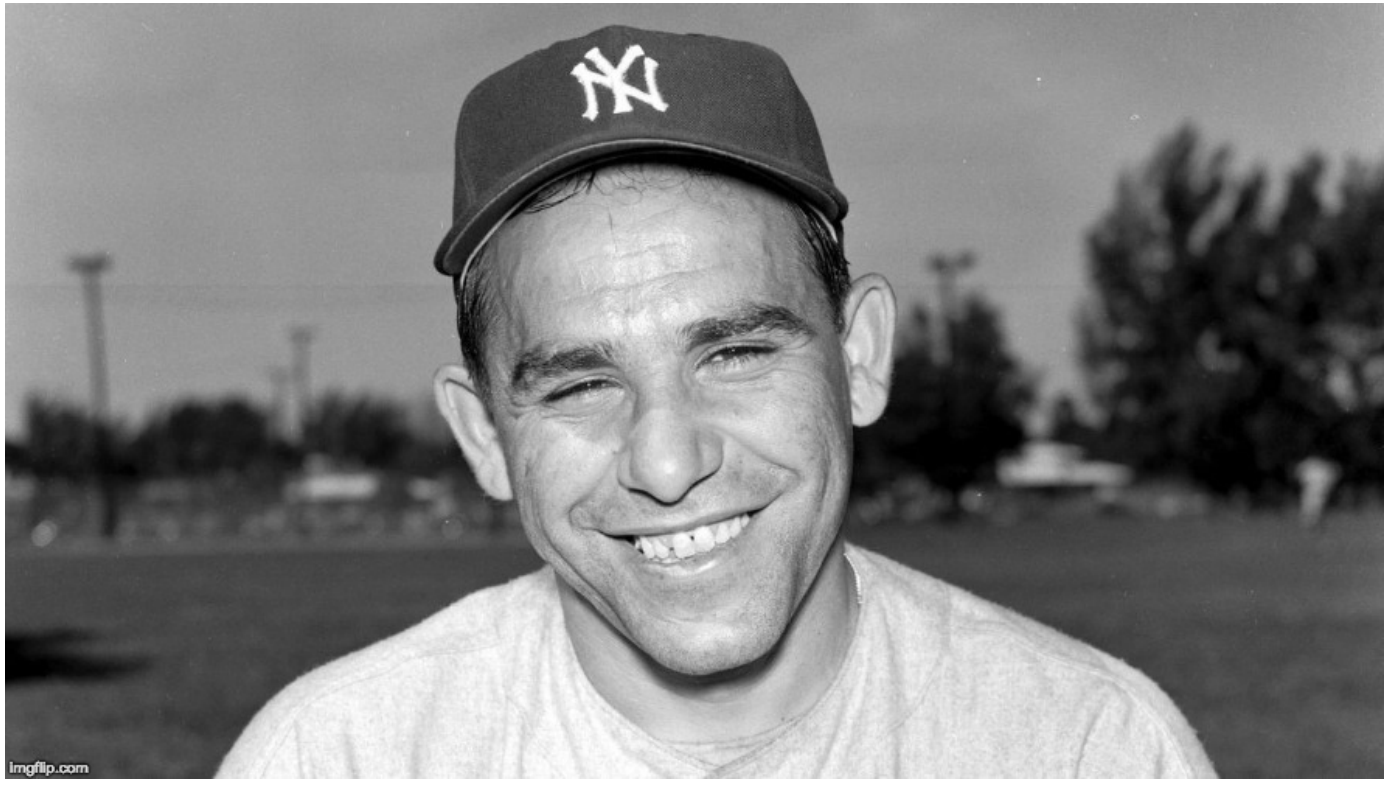


So we should pick K=2?

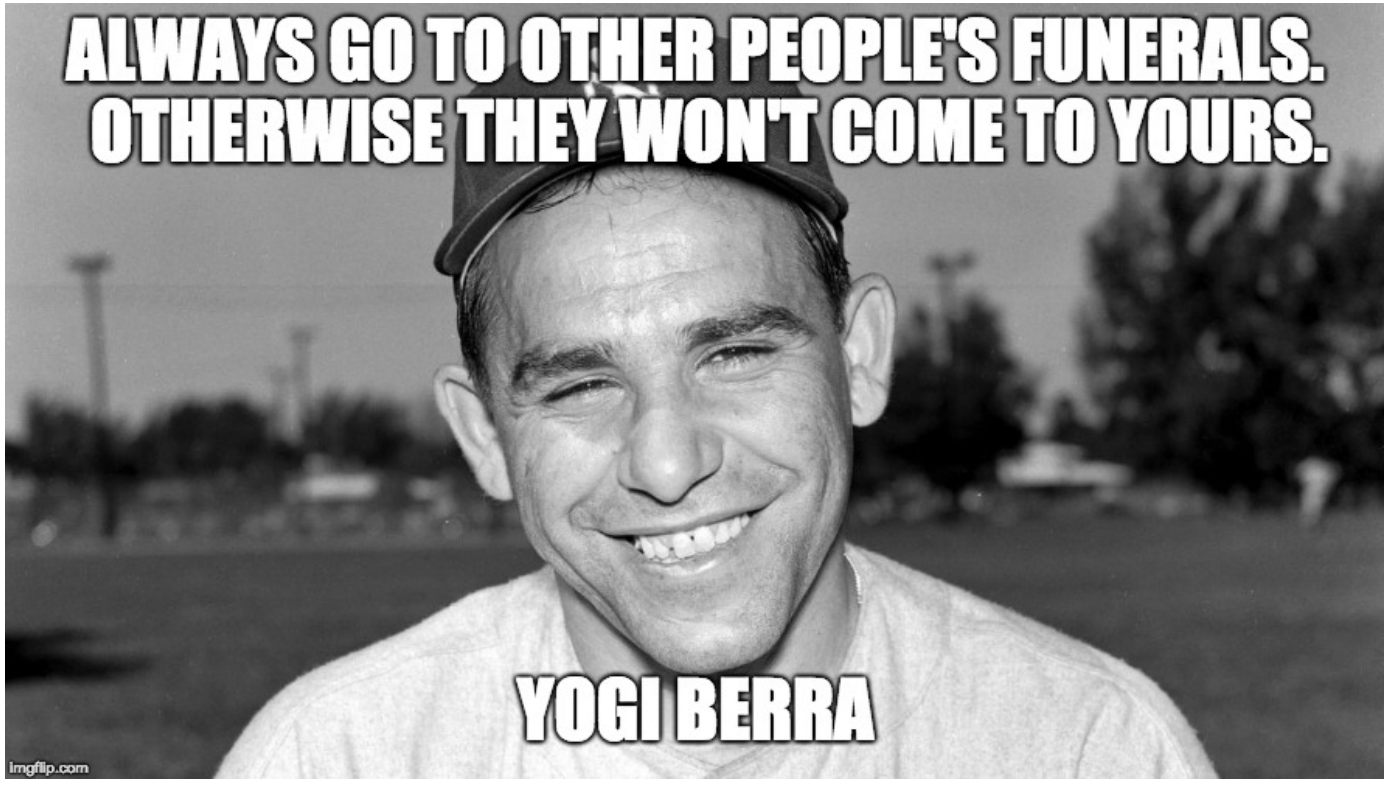


RMSE = 670

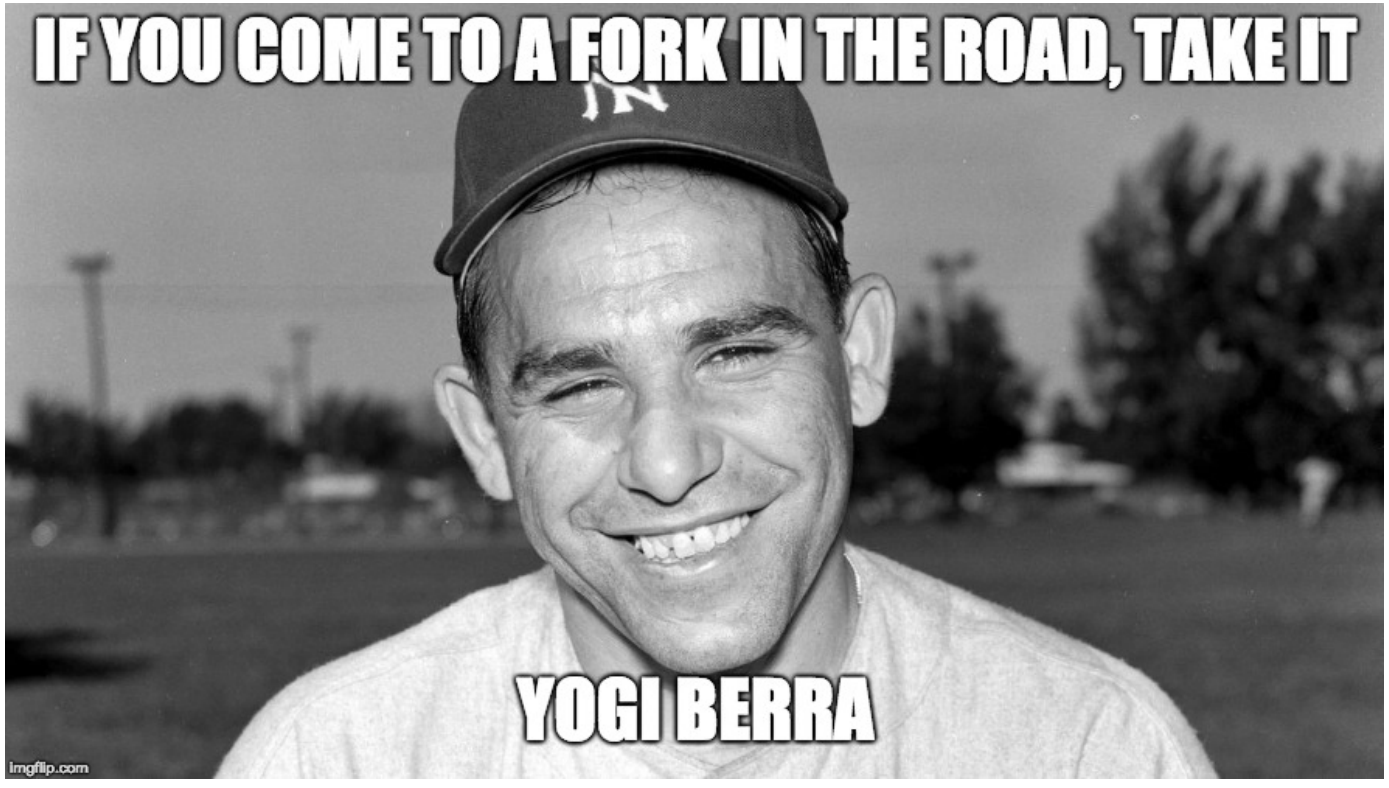
So we should pick K=2? Ask Yogi!



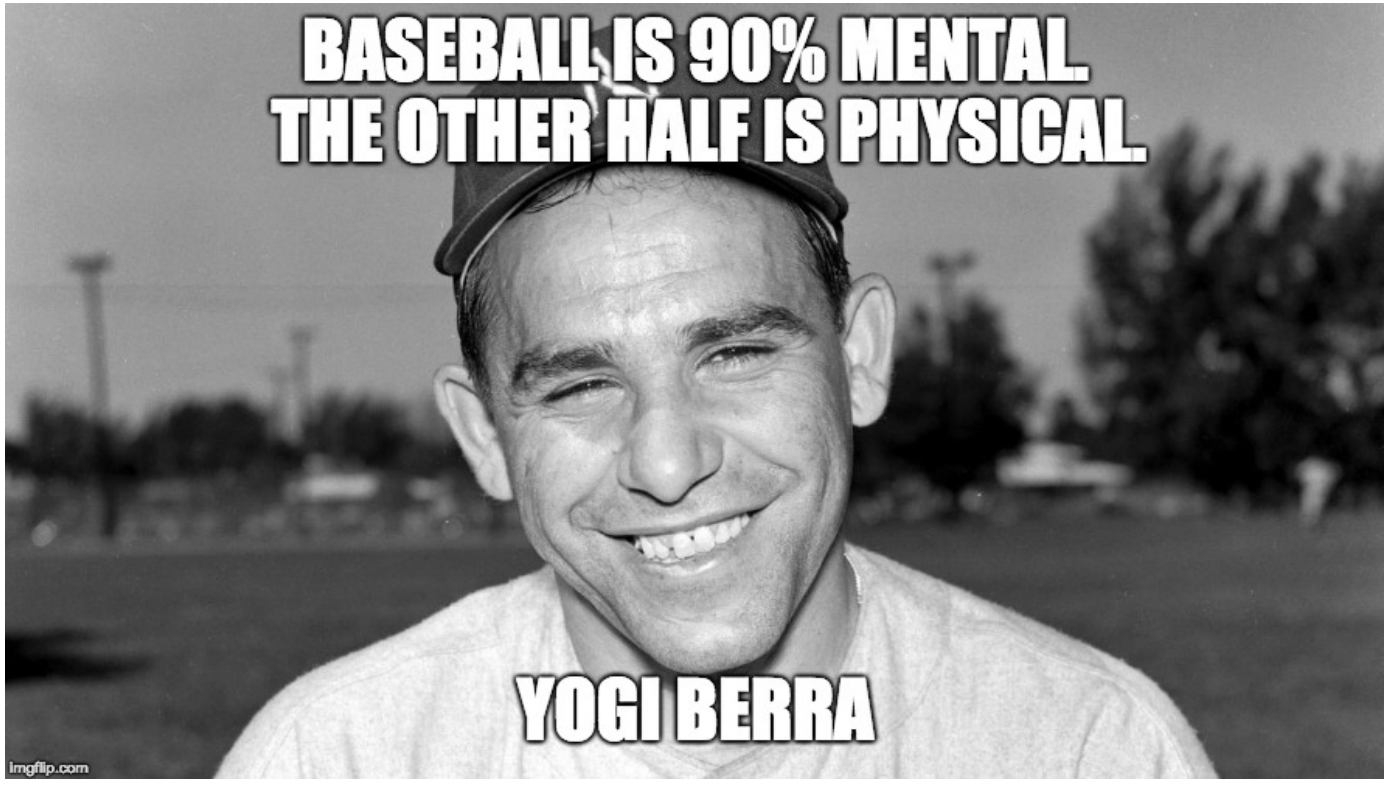
So we should pick K=2? Ask Yogi!



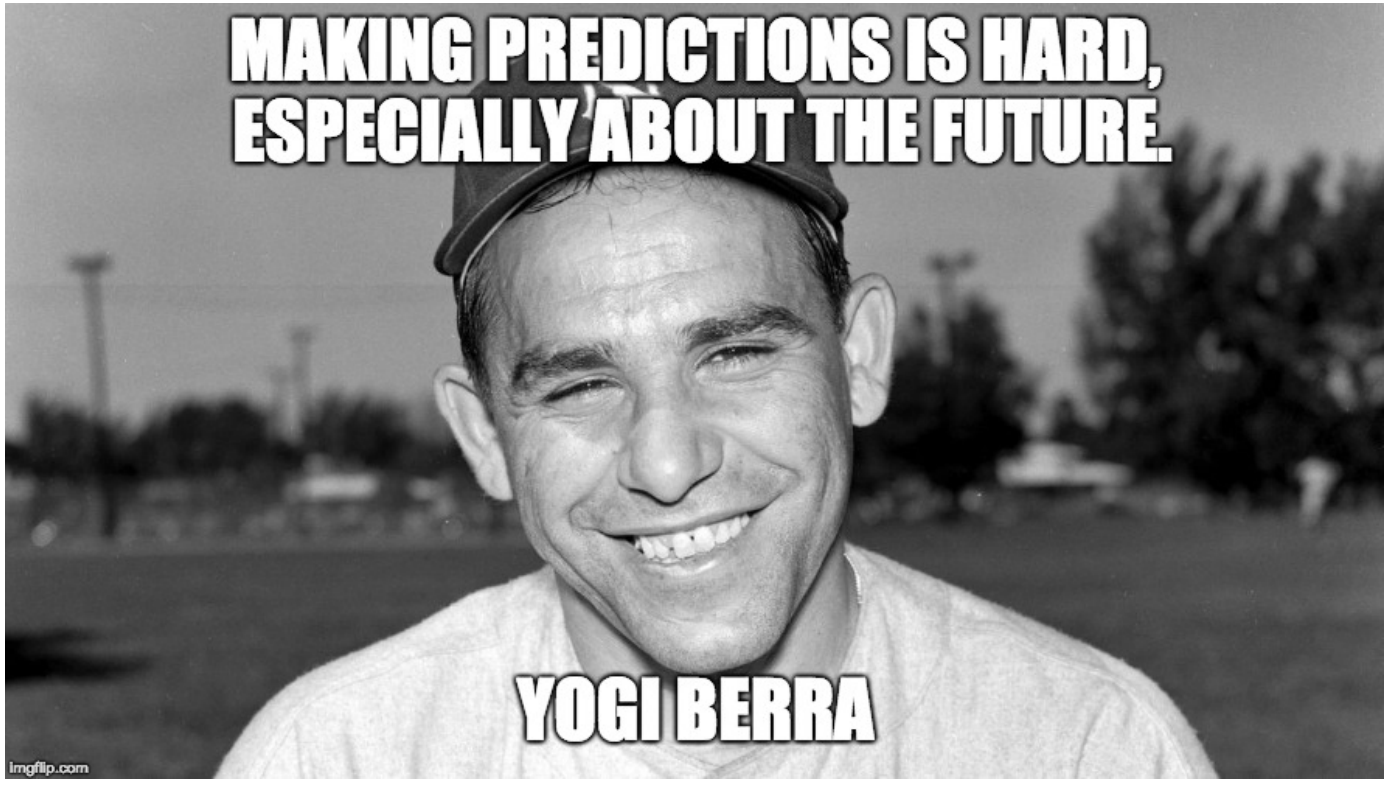
So we should pick K=2? Ask Yogi!



So we should pick K=2? Ask Yogi!



So we should pick K=2? Ask Yogi!



Out-of-sample accuracy

Making good predictions about the past isn't very impressive.

Our very complex (K=2) model earned a low RMSE by simply memorizing the random pattern of noise in the training data.

It's like getting a perfect score on the GRE when someone tells you what the questions are ahead of time: it doesn't predict anything about how well you'll do on the **next** test.

Out-of-sample accuracy

Key idea: what really matters is our prediction accuracy out-of-sample!

Suppose we have M **additional** observations $(x_i^\star, y_i^\star)$ for $i = 1, \ldots, M$.

- The important thing is that these are data points that we *did not use* to fit the model.
- We'll call this the "testing" data, to distinguish it from our original ("training") data.

The out-of-sample root mean-squared error is then:

$$\mathrm{RMSE}_{\mathrm{out}} = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(y_i^\star - f(x_i^\star))^2}$$

Using a train/test split

We don't have any "future data" to use to test our model. We just have our N original data points.

A simple solution is a train/test split. That is, we split our data set D into two subsets:

- A training set D_{in} of size $N_{in} < N$, to use for fitting the models under consideration.
- A testing set D_{out} of size N_{out}.
- $D = D_{in} \cup D_{out}$ and $N = N_{in} + N_{out}$, but $D_{in} \cap D_{out} = \emptyset$.
- No cheating! We use D_{in} *only* to fit the models, and D_{out} *only* to compare the out-of-sample accuracy of the models.

Using a train/test split

The R code for this is pretty simple. For example, here's out-of-sample RMSE calculated for KNN-25 model:

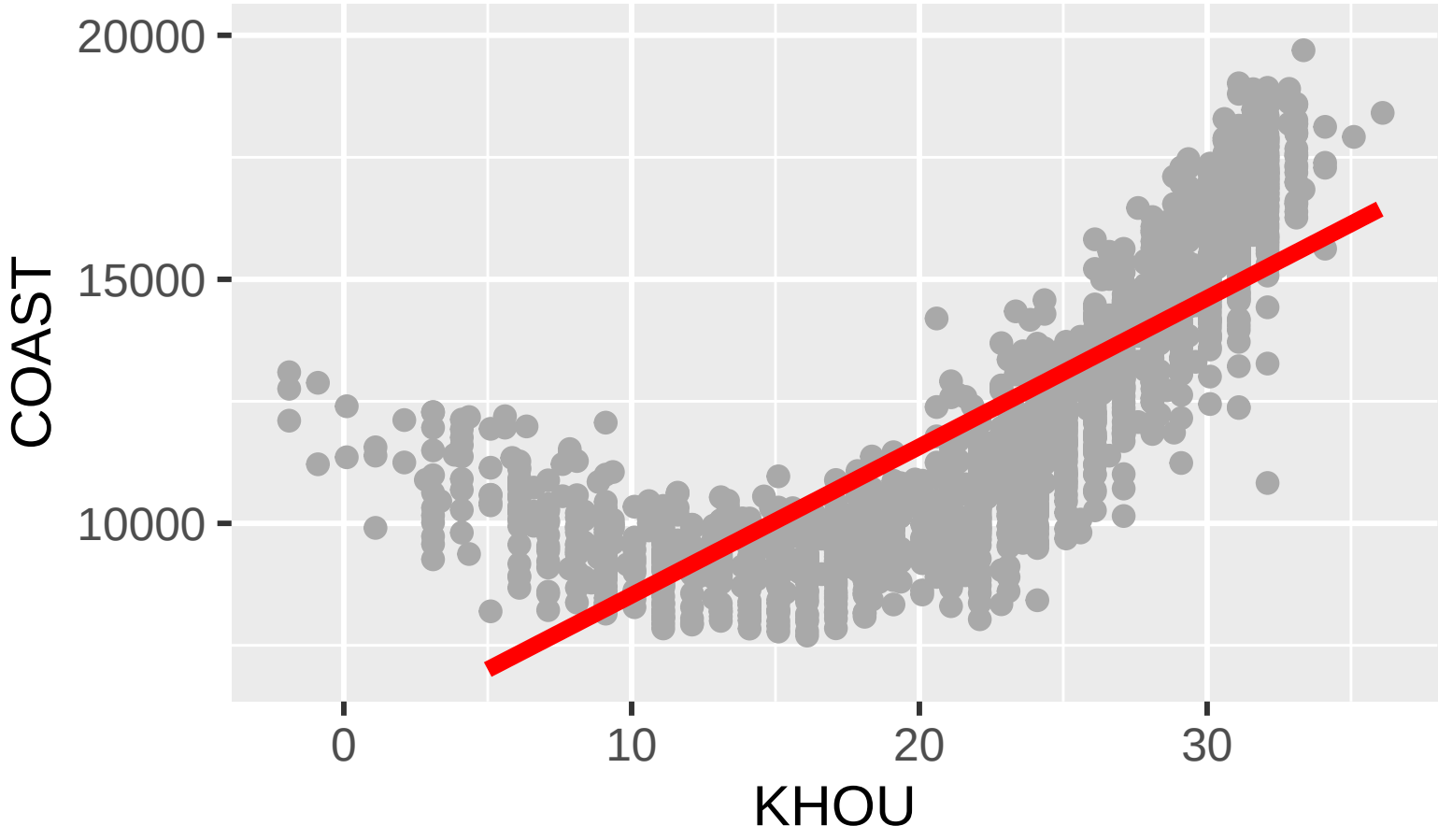
```
loadhou_split =  initial_split(loadhou, prop=0.8)
loadhou_train = training(loadhou_split)
loadhou_test  = testing(loadhou_split)

# train the model and calculate RMSE on the test set
knn_model = knnreg(COAST ~ KHOU, data=loadhou_train, k = 25)
modelr::rmse(knn_model, loadhou_test)
```

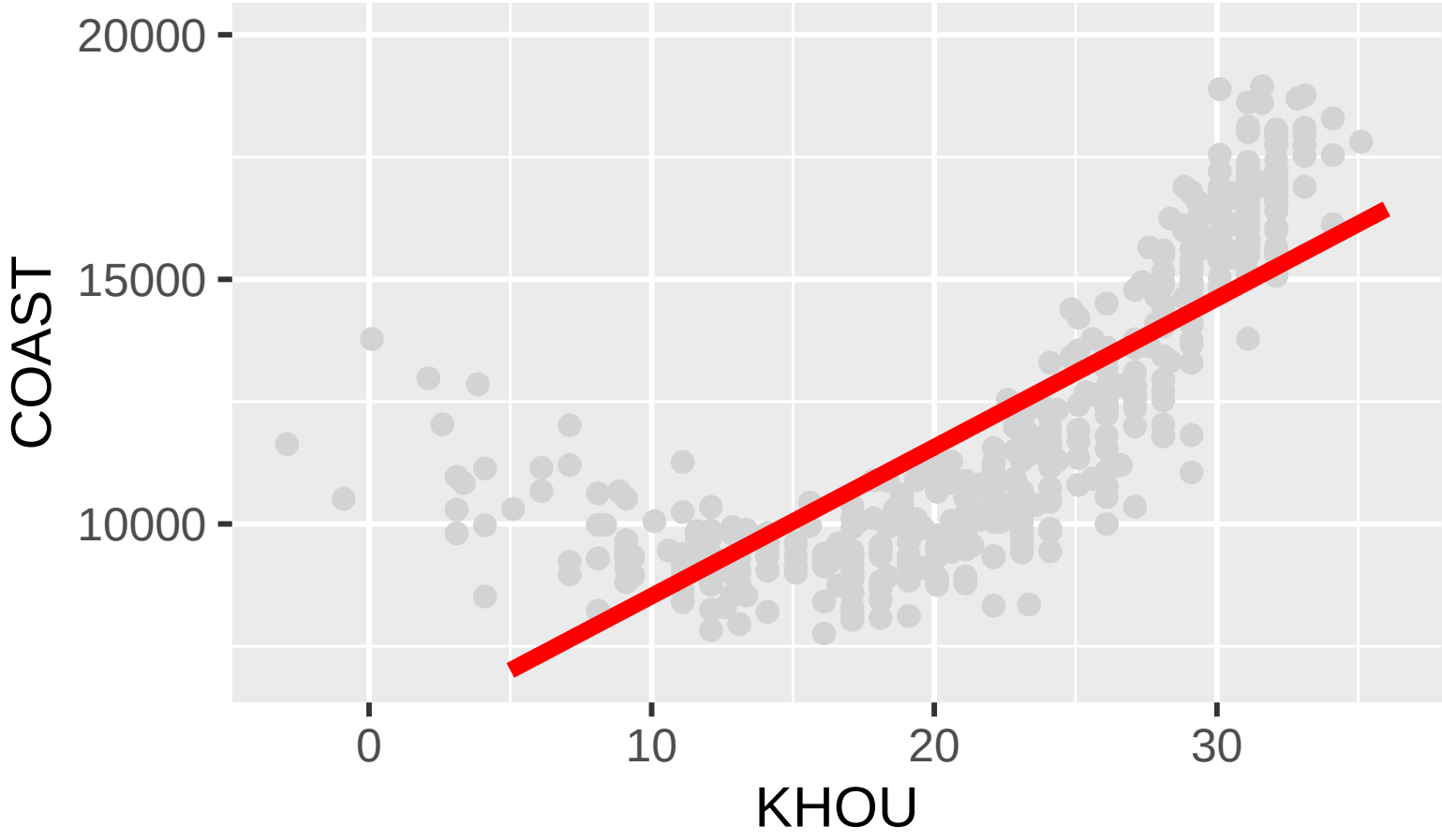
```
## [1] 1023.188
```

So let's check performance across a variety of choices for K, versus the linear and quadratic models.

Linear model: train

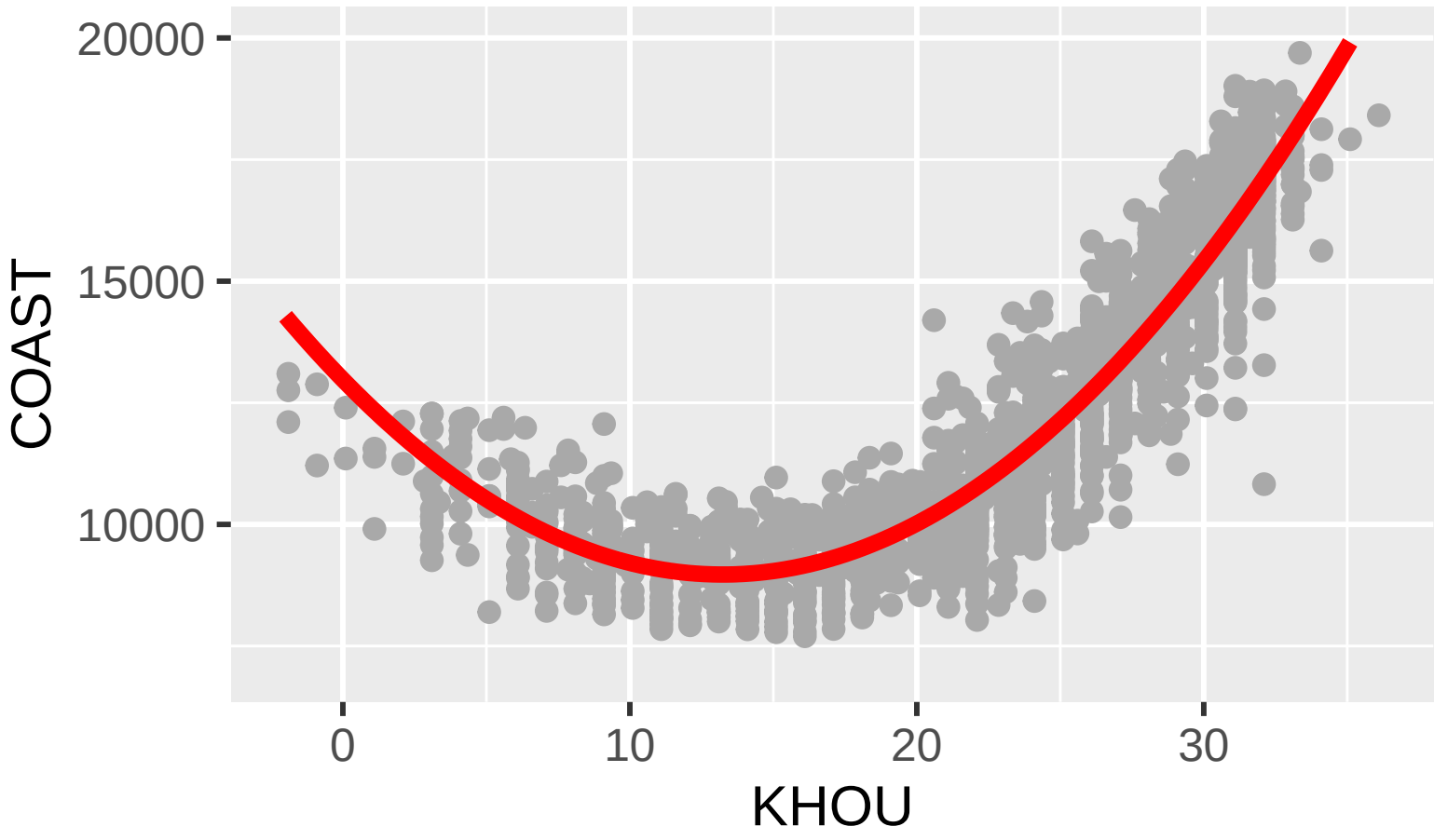


Linear model: test

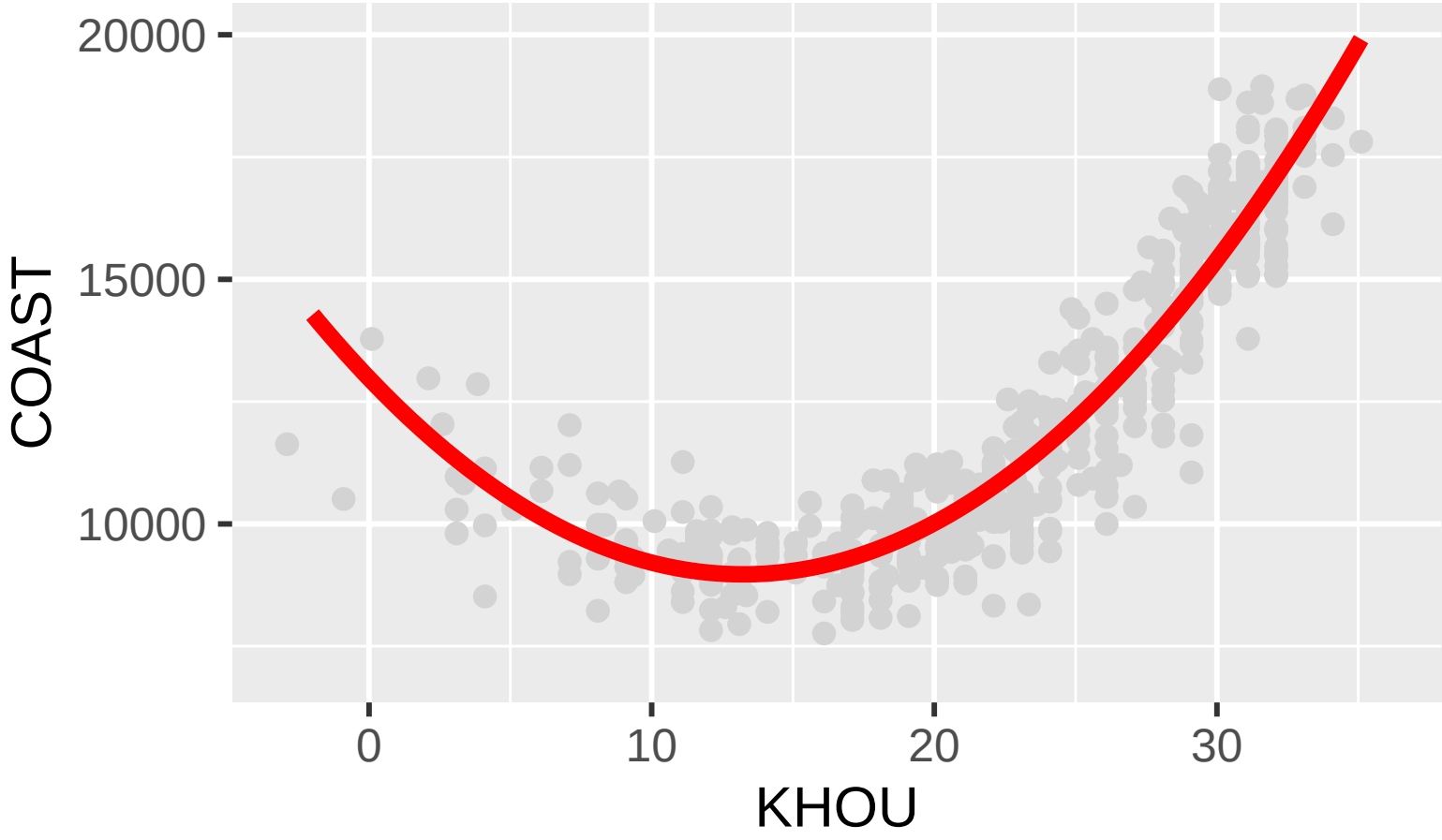


$$RMSE_{out} = 1869$$

Quadratic model: train

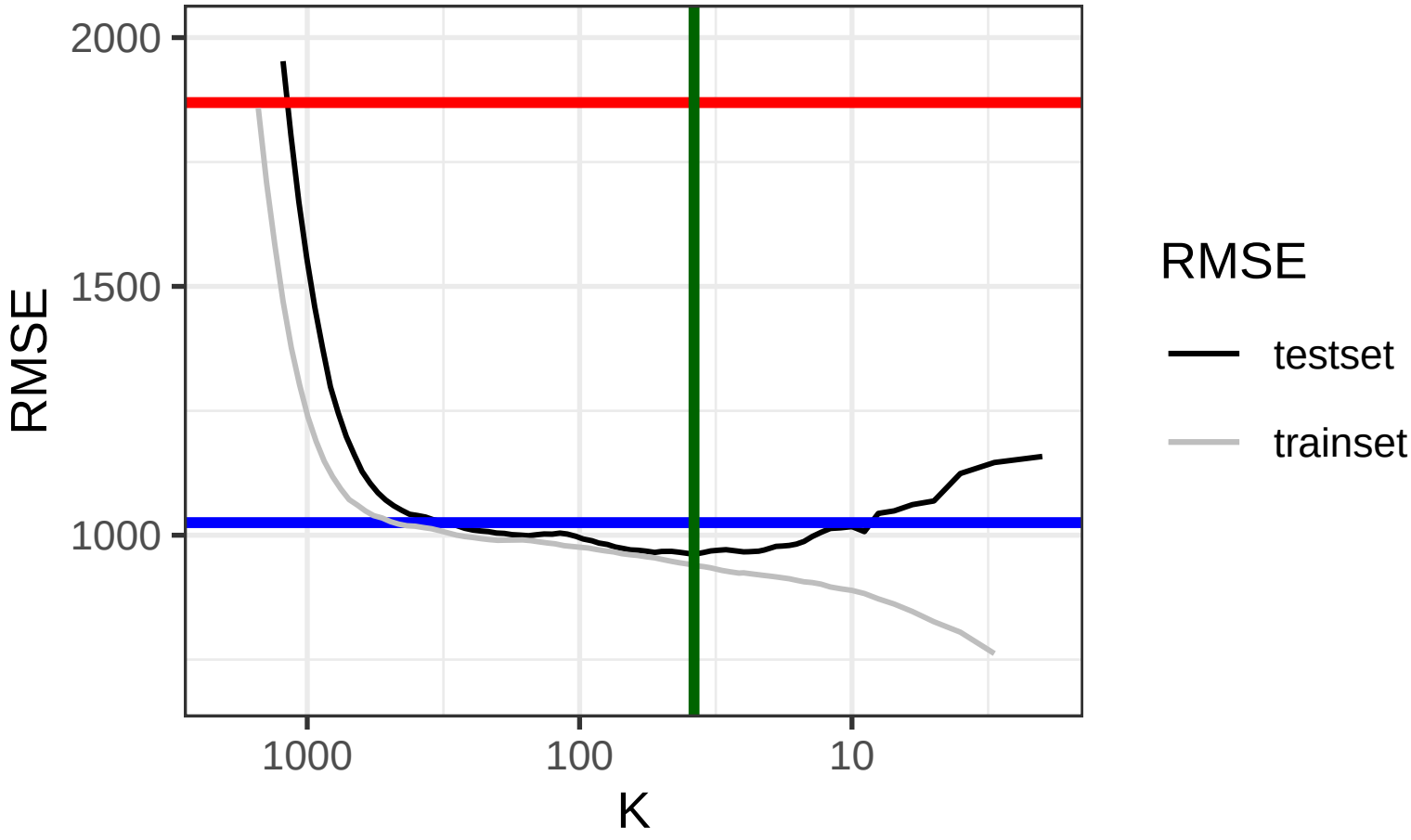


Quadratic model: test



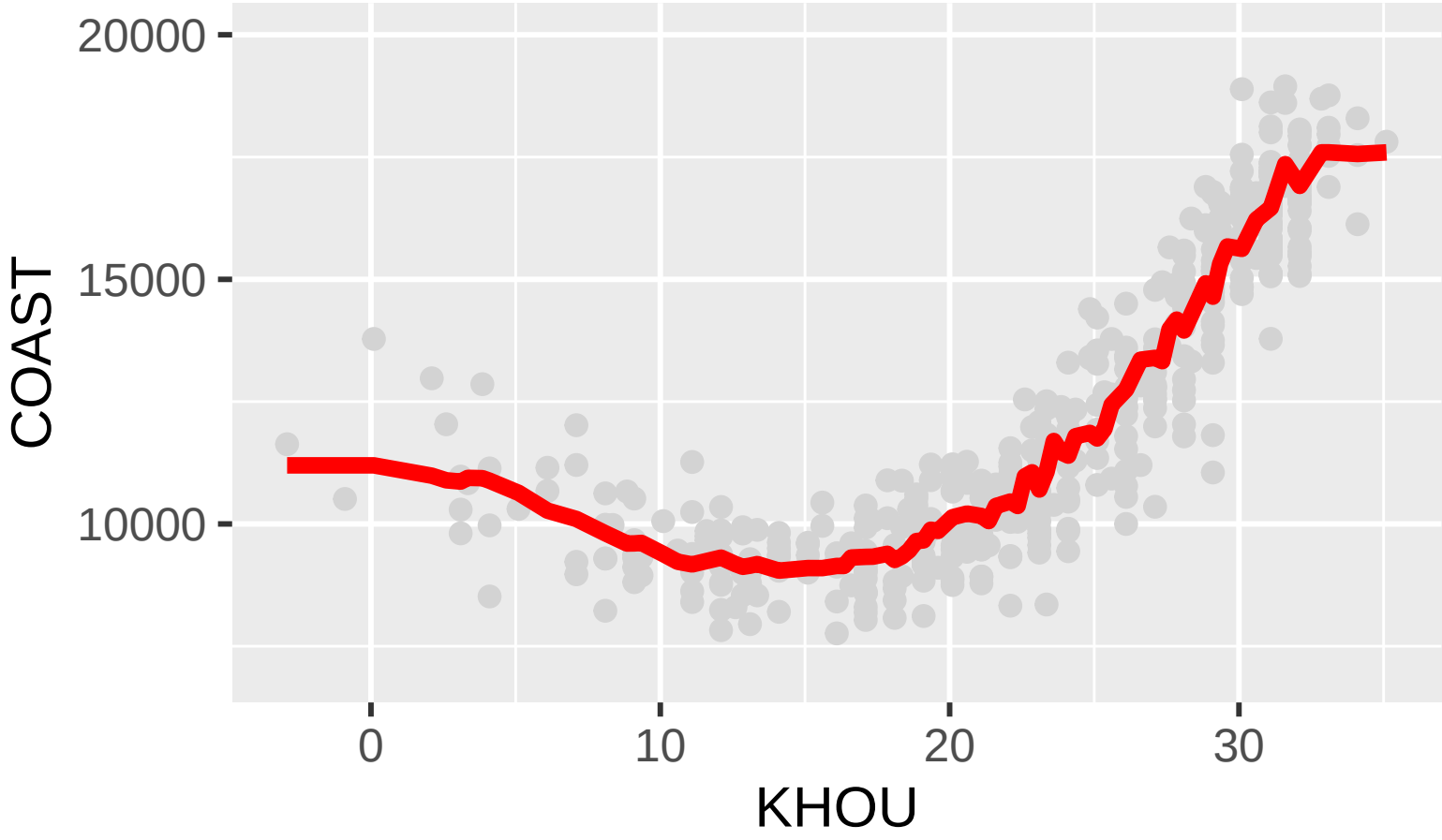
$RMSE_{out} = 1025$

K-nearest neighbors: test



Not too simple, not too complex... This plot illustrates the **bias-variance trade-off**, one of the key ideas of this course.

K-nearest neighbors: at the optimal k



$$\text{RMSE}_{out} = 962$$

Take-home lessons

- In general, $RMSE_{out} > RMSE_{in}$. That is, the estimate of RMSE from the training set is an over-optimistic assessment of how big your errors will be for future data.
- For very complex models, $RMSE_{in}$ can be *wildly* optimistic.
- The best model is usually one that balances simplicity with explanatory power.
- Estimating $RMSE_{out}$ using a train-test split of the original data set is a simple way to help us from going too far wrong.

In-class exercise

Download the `loudhou` data set and starter R script. Get a feel for how the code behaves:

1. Make a train/test split.
2. Train on the training set.
3. Predict on the testing set.
4. Plot the results.

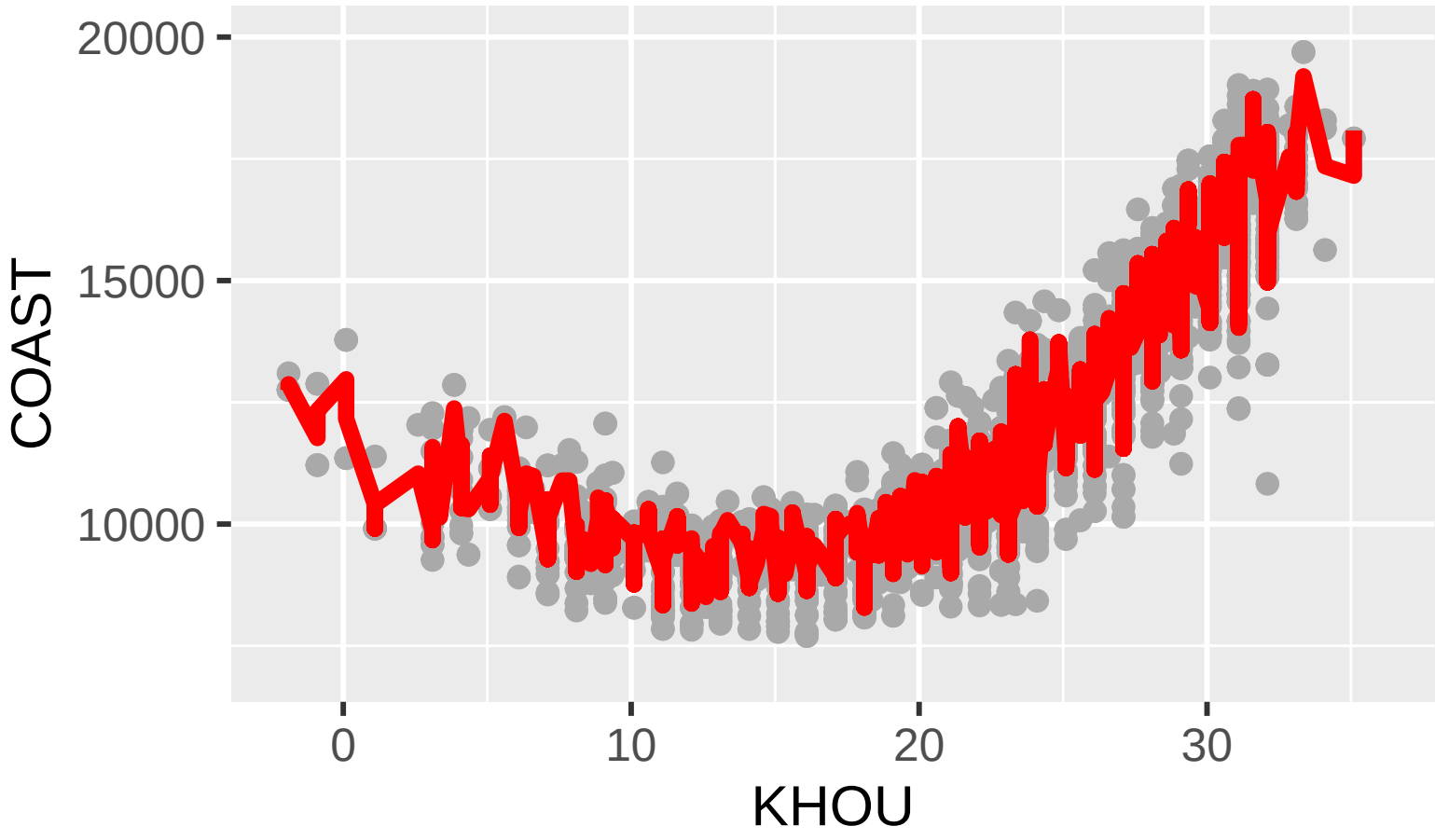
In-class exercise

Then make an informal investigation of the *bias* and *variance* of the KNN estimator:

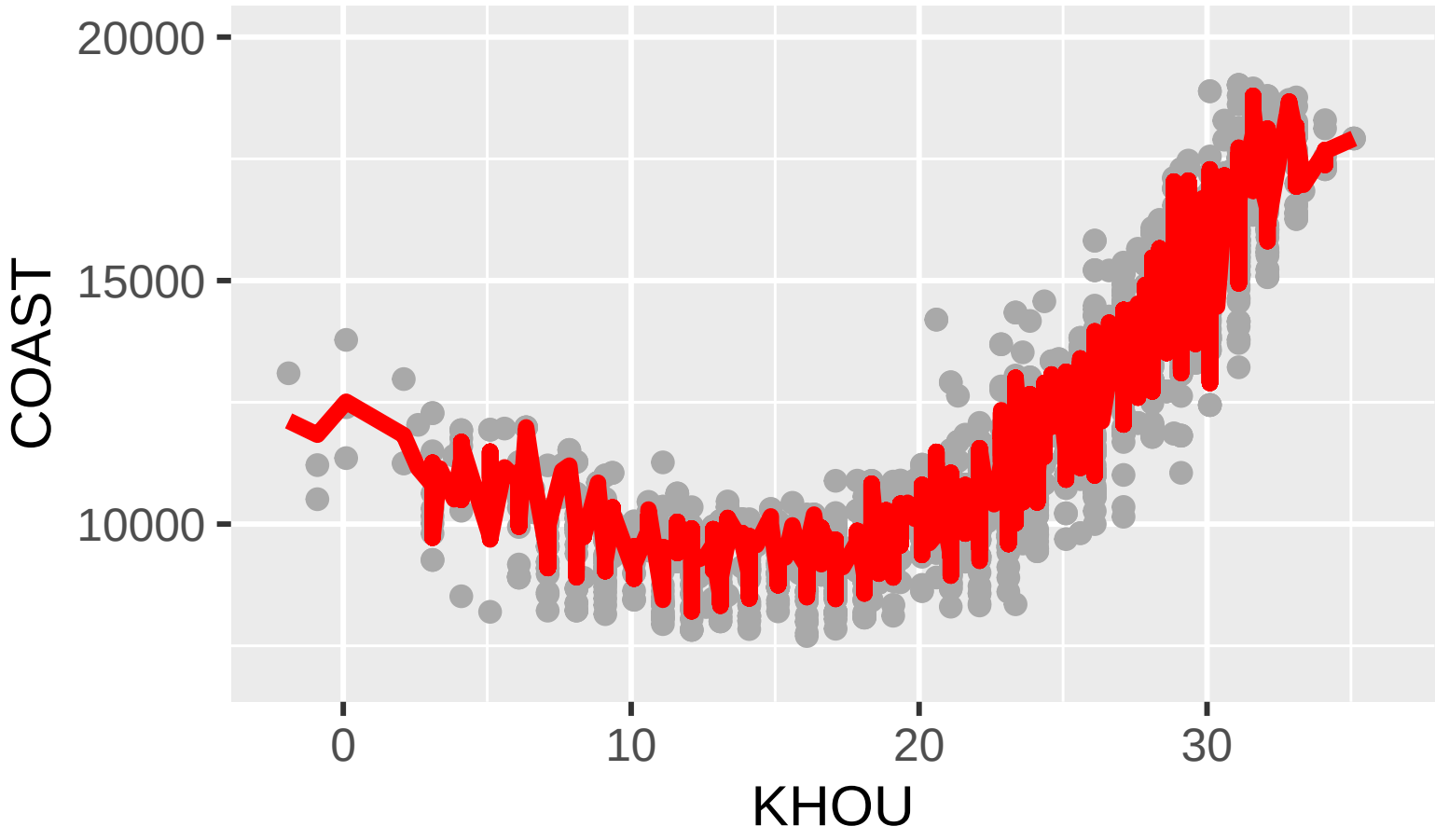
1. Repeatedly take bootstrap samples of the full data set, and train a $K = 3$ model on each small training set. Plot the fit to the training set. How stable are they from sample to sample? How do they behave at the endpoints, i.e. at very low and very high temperatures?
2. Now do the same thing, except training a $K = 75$ model on each bootstrap sample. How stable are the estimates from sample to sample? And how do they behave at the endpoints?

Hint: keep the x and y limits constant across plots, e.g. by adding the layers `+ ylim(7000, 20000) + xlim(0,36)` or whatever limits seem appropriate.

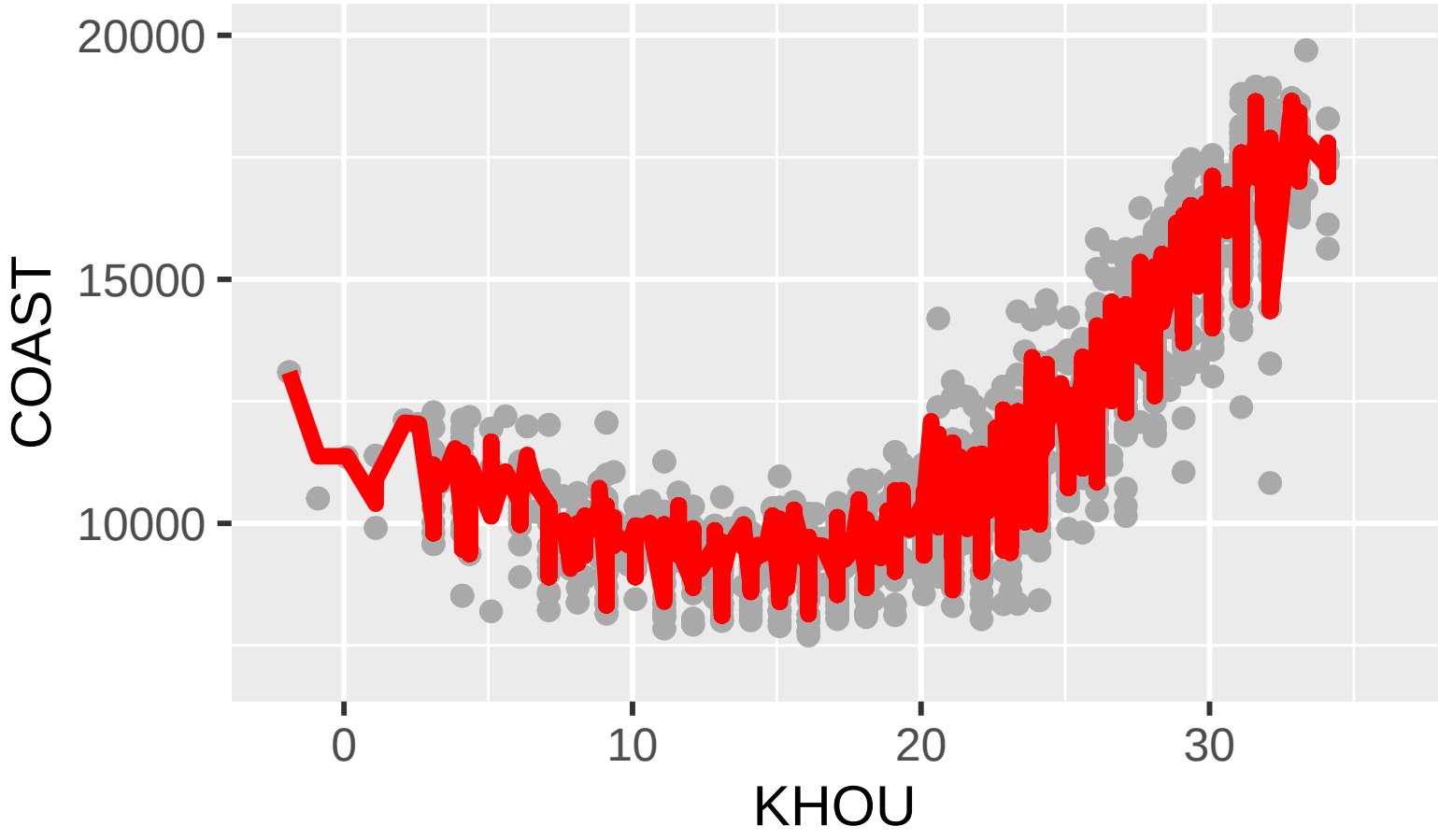
K=3: sample 1



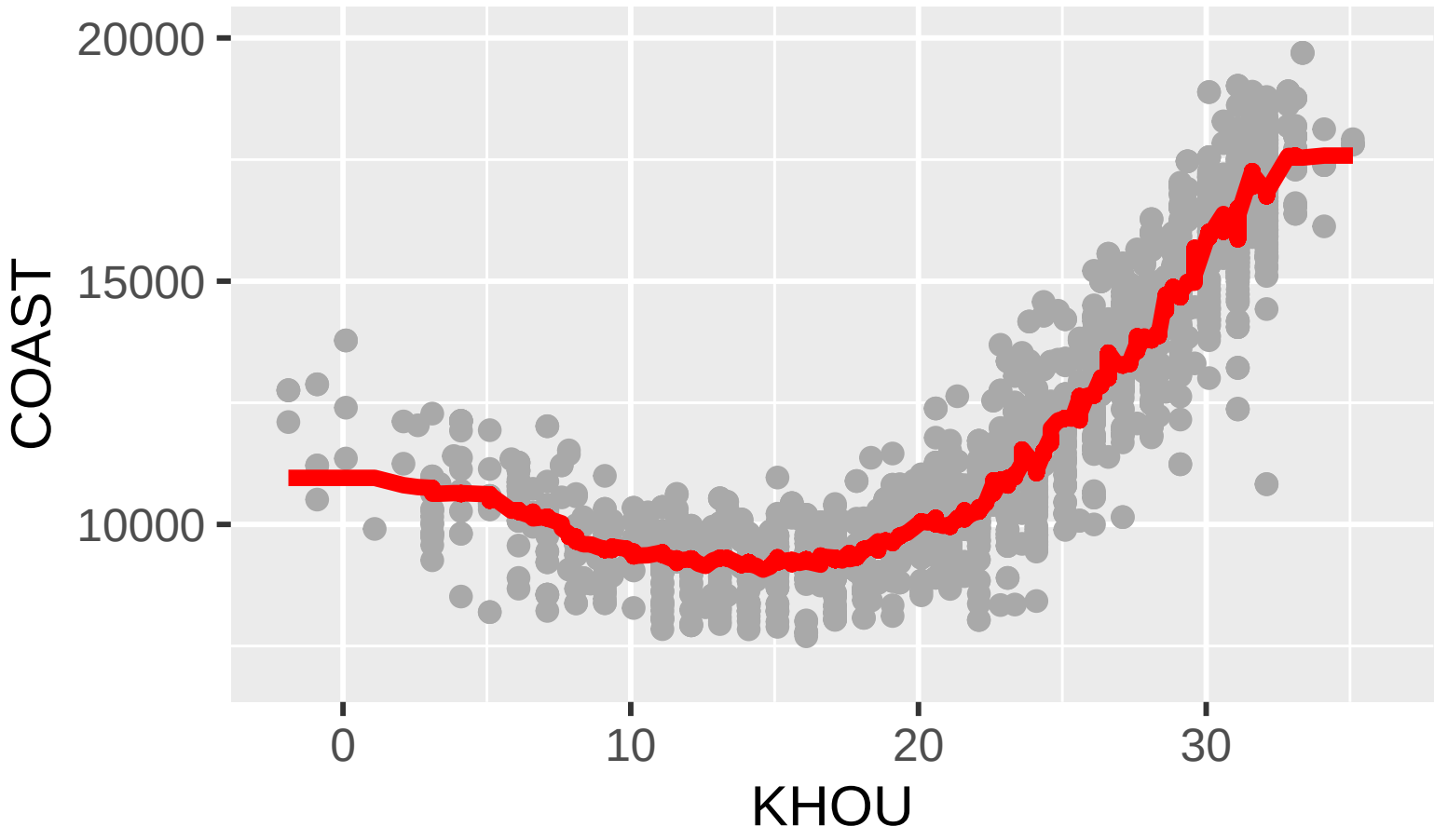
K=3: sample 2



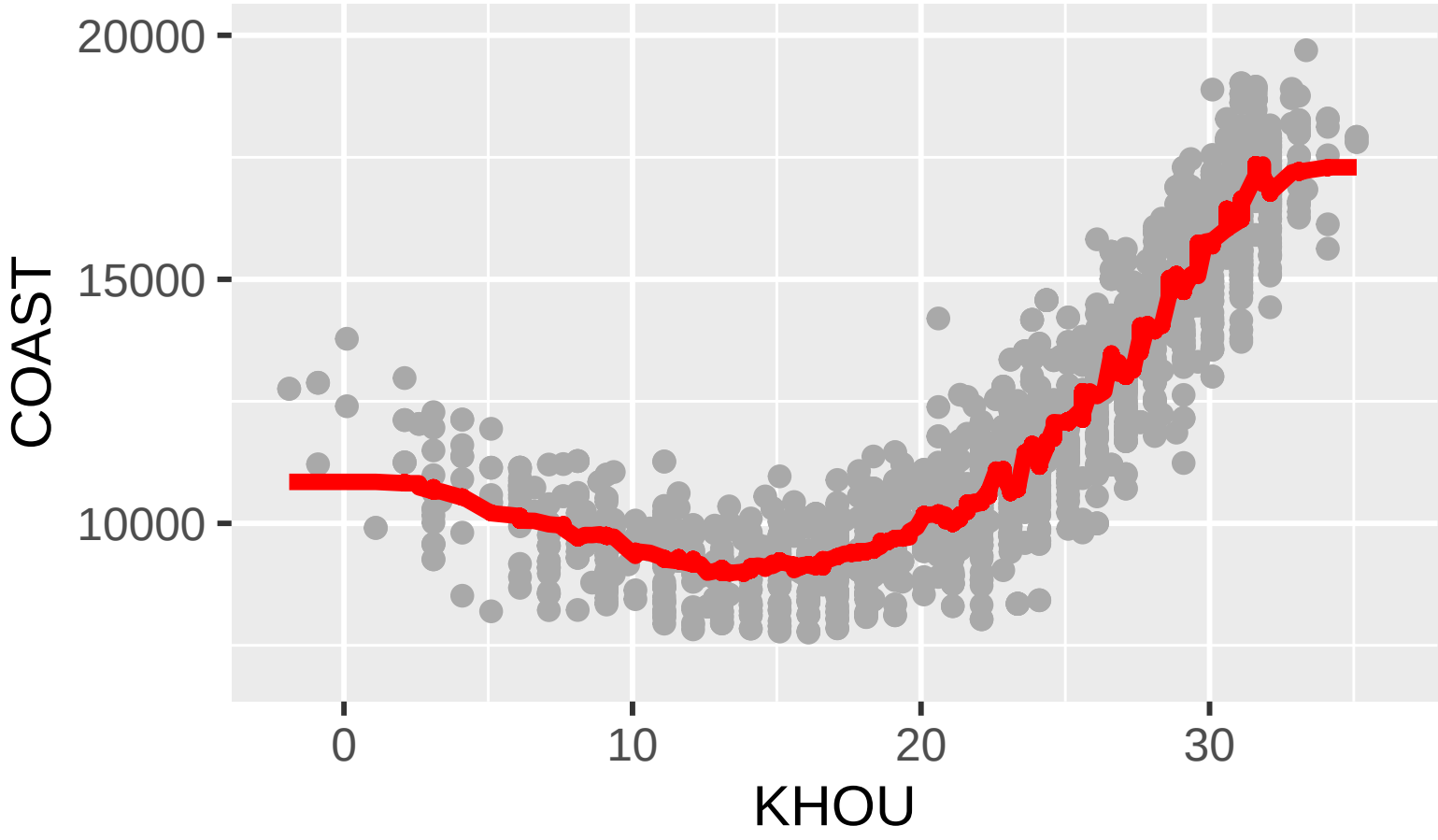
K=3: sample 3



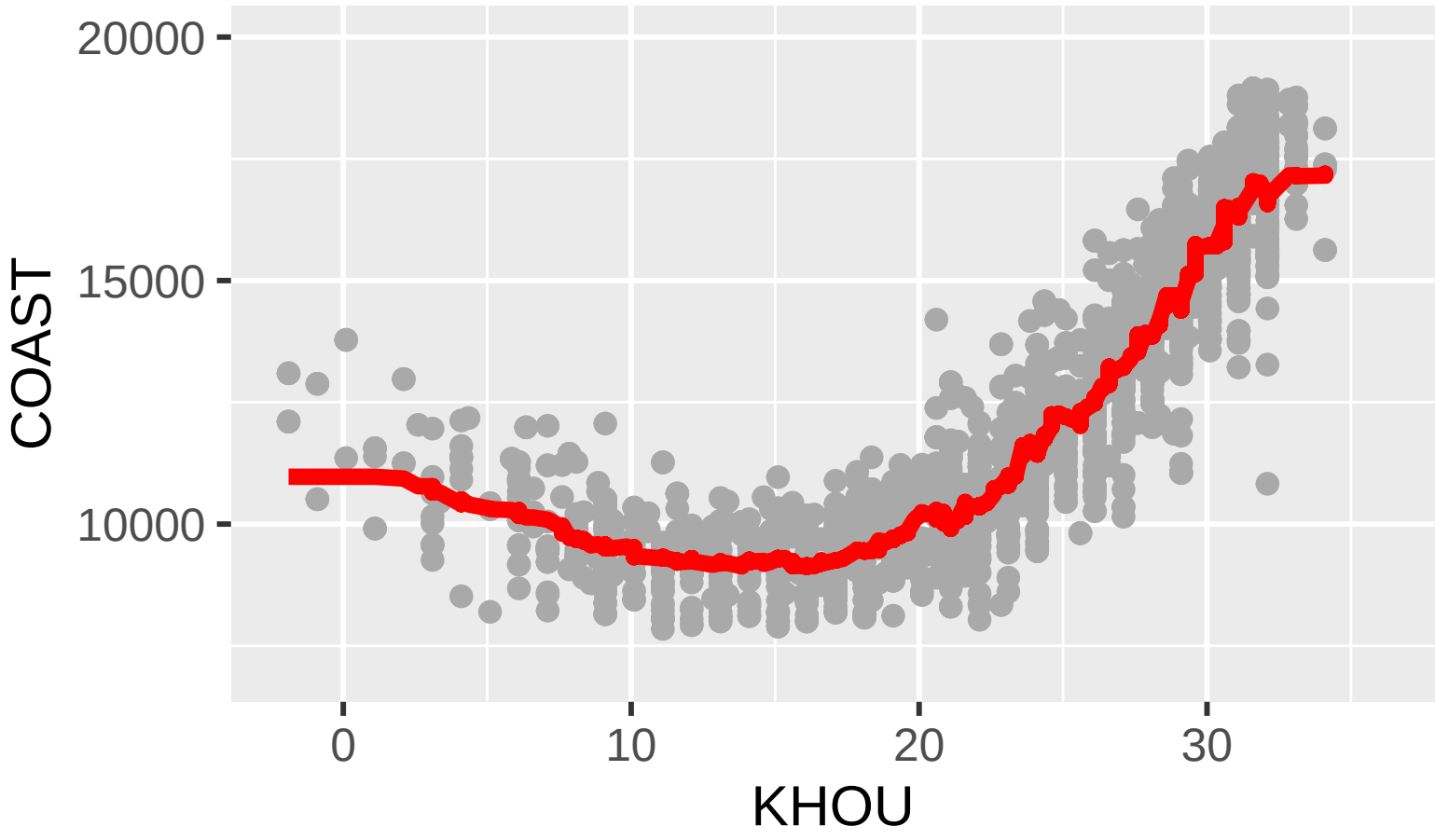
K=75: sample 1



K=75: sample 2



K=75: sample 3



Bias-variance trade-off

High K = high bias, low variance:

- We estimate $f(x)$ using many points, some of which might be far away from x. These far-away points bias the prediction; their values of $f(x)$ are slightly off on average.
- But more data points means lower variance—less chance of memorizing random noise.

Low K = low bias, high variance:

- We estimate $f(x)$ using only points that are *very close* to x. Far-away x points don't bias the prediction with their "slightly off" y values.
- But fewer data points means higher variance—more chance of memorizing random noise.

Bias-variance trade-off

Let's take a deeper look at prediction error.

- Let $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ be your training data.
- Suppose that $y = f(x) + e$, where $E(e) = 0$ and $\text{var}(e) = \sigma^2$.
- Let $\hat{f}(x)$ be the function estimate from your training data.
- Let $x^\star$ be a new x point, and let $y^\star$ be the corresponding outcome.
- $x^\star$ is fixed (not a random variable), but the training data and the future outcome $y^\star$ are both random.

Define the expected squared prediction error at $x^\star$ as:

$$MSE^\star = E\left\{\left(y^\star - \hat{f}(x^\star)\right)^2\right\}$$

Bias-variance trade-off

For any random variable A, $E(A^2) = \text{var}(A) + E(A)^2$. So:

$$
\begin{aligned}
MSE^\star &= E\left\{\left(y^\star - \hat{f}(x^\star)\right)^2\right\} \\
&= \text{var}\left\{y^\star - \hat{f}(x^\star)\right\} + \left(E\left\{y^\star - \hat{f}(x^\star)\right\}\right)^2 \\
&= \text{var}\left\{f(x^\star) + e^\star - \hat{f}(x^\star)\right\} + \left(E\left\{f(x^\star) + e^\star - \hat{f}(x^\star)\right\}\right)^2 \\
&= \sigma^2 + \text{var}\left\{\hat{f}(x^\star)\right\} + \left(E\left\{f(x^\star) - \hat{f}(x^\star)\right\}\right)^2 \\
&= \sigma^2 + (\text{Estimation variance}) + (\text{Squared estimation bias})
\end{aligned}
$$

Bias-variance trade-off

$$MSE^\star = \sigma^2 + \text{var}\left\{\hat{f}(x^\star)\right\} + \left(E\left\{f(x^\star) - \hat{f}(x^\star)\right\}\right)^2$$

First, consider σ^2.

- This is the intrinsic variability of the data: remember, $y = f(x) + e$, and $\text{var}(e) = \sigma^2$.
- How can we make this term smaller?

Bias-variance trade-off

$$MSE^\star = \sigma^2 + \text{var}\left\{\hat{f}(x^\star)\right\} + \left(E\left\{f(x^\star) - \hat{f}(x^\star)\right\}\right)^2$$

Next, consider $\text{var}\left\{\hat{f}(x^\star)\right\}$.

- This is the variance of our estimate $\hat{f}(x)$: remember, our estimate is random, because the data is random.
- How can we make this term smaller?

Bias-variance trade-off

$$MSE^\star = \sigma^2 + \text{var}\left\{\hat{f}(x^\star)\right\} + \left(E\left\{f(x^\star) - \hat{f}(x^\star)\right\}\right)^2$$

Finally, consider $\left(E\left\{f(x^\star) - \hat{f}(x^\star)\right\}\right)^2$.

- This is the bias of our estimate $\hat{f}(x)$: remember, our estimate doesn't necessarily equal the true $f(x)$, even on average.
- How can we make this term smaller?

Bias-variance trade-off

That's why it's a trade-off!

- Smaller estimation variance generally requires a *less complex* model—intuitively, one that "wiggles less" from sample to sample. (Think K=75 on our `loadhou` example.)
- Smaller bias generally requires a *more complex* model—one that can "wiggle more," to adapt to the true function. (Think K=3 on our `loadhou` example.)
- Models that "wiggle more" can adapt to more kinds of functions, but they're also more prone to memorizing random noise.

Much of the rest of the semester is about finding estimates with the right amount of wiggle!

Measuring model accuracy, revisited

Recall our definition of "true" out-of-sample MSE:

$$\text{MSE}^\star = E\left\{\left(y^\star - \hat{f}(x^\star)\right)^2\right\}$$

As we've seen, a simple way to estimate this quantity is to train our model $\hat{f}$ on D_{in} and to calculate average performance on D_{out}:

$$\widehat{\text{MSE}} = \frac{1}{N_{\text{out}}} \sum_{i=1}^{N_{\text{out}}} \left(y_i - \hat{f}(x_i)\right)^2$$

The key word here is **estimate.** There are two sources of randomness in our estimate:

- $\hat{f}(x)$, the function estimate from D_{in}.
- the specific (x_i, y_i) pairs that end up in D_{out}.

Measuring model accuracy, revisited

So let's see what happens if we try this process for ten different random train/test splits:

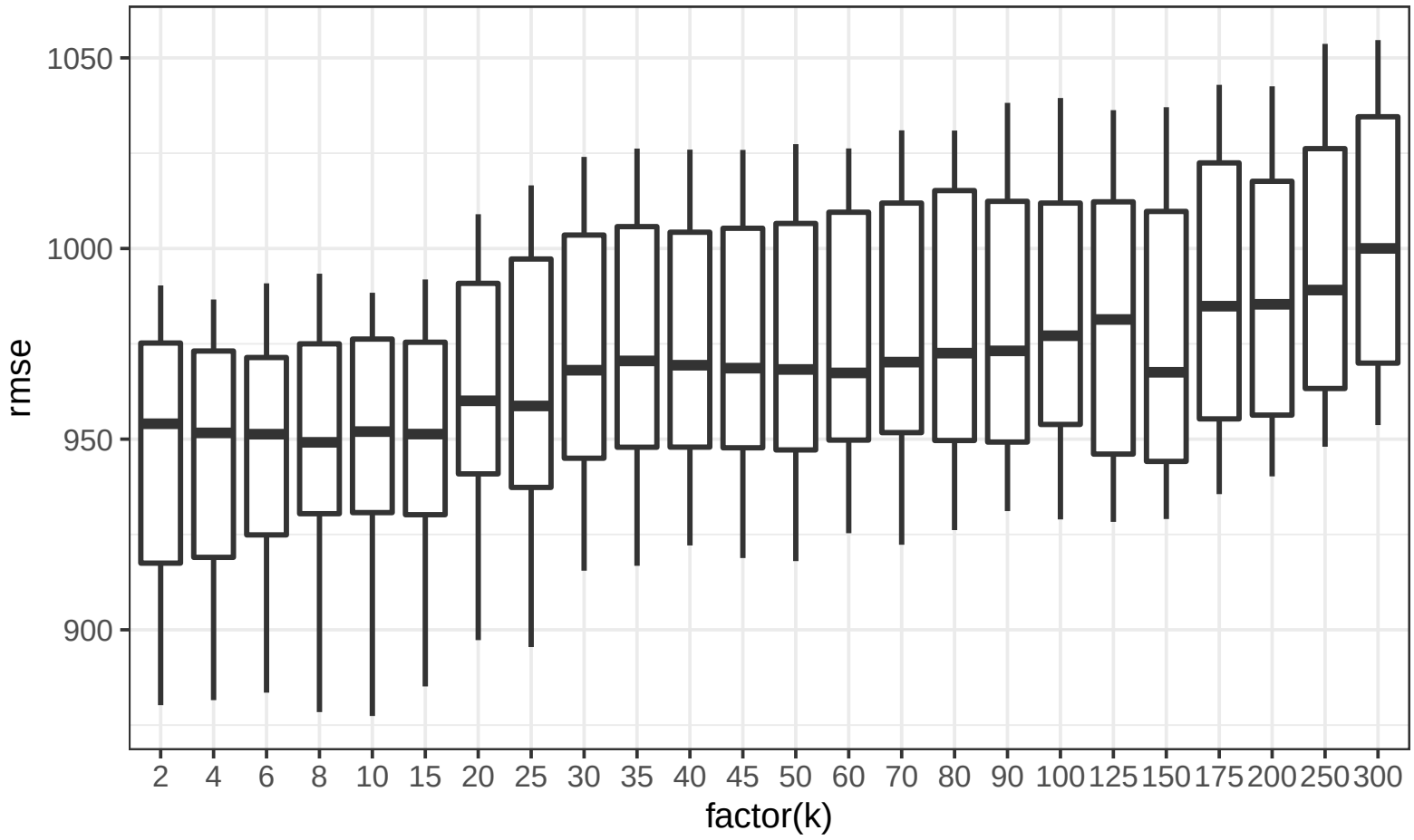
```
rmse_out = foreach(i=1:10, .combine='c') %do% {
  loadhou_split =  initial_split(loadhou, prop=0.8)
  loadhou_train = training(loadhou_split)
  loadhou_test  = testing(loadhou_split)

  # train the model and calculate RMSE on the test set
  knn_model = knnreg(COAST ~ KHOU, data=loadhou_train, k = 25)
  modelr::rmse(knn_model, loadhou_test)
}
rmse_out
```

```
##  [1]  961.7983  969.6035  965.1566  968.6203
##  [5]  998.3412  950.1040  971.5591 1006.6980
##  [9]  974.1025  994.5908
```

Measuring model accuracy, revisited

Let's see this across multiple values of K.



Measuring model accuracy, revisited

This is a big lesson: any estimate of RMSE is subject to error!

- Our $\widehat{\mathrm{RMSE}}$ differs from one train/test split to the next.
- This is intuitive: $\widehat{\mathrm{RMSE}}$ just an estimate of something unknown ($\mathrm{RMSE}^\star$), and all estimates of unknown quantities have variance.
- In fact, the variability of $\widehat{\mathrm{RMSE}}$ across different train/test splits *for fixed K* can be pretty large, compared to the differences across a wide range of K.

Before we selected K using a single train/test split and picked the K with the lowest RMSE. **Should we modify our approach?**

Averaging across multiple splits

An obvious solution:

- Just average $\widehat{\mathrm{RMSE}}$ for all your models across multiple train/test splits.
- This will reduce the variance of $\widehat{\mathrm{RMSE}}$, compared with using a single train/test split.
- This is perfectly valid, and we'll sometimes use it when training the model is pretty cheap.

Averaging across multiple splits

An obvious solution:

- Just average $\widehat{\mathrm{RMSE}}$ for all your models across multiple train/test splits.
- This will reduce the variance of $\widehat{\mathrm{RMSE}}$, compared with using a single train/test split.
- This is perfectly valid, and we'll sometimes use it when training the model is pretty cheap.

But while perfectly sensible, this obvious solution is a bit inefficient.

- Two randomly sampled testing sets can overlap a lot.
- Overlap means that $\widehat{\mathrm{RMSE}}_1$ and $\widehat{\mathrm{RMSE}}_2$ calculated from these two testing sets are highly correlated.
- Remember: averaging quantities with high covariance is less efficient than averaging those with low covariance.

K-fold cross validation

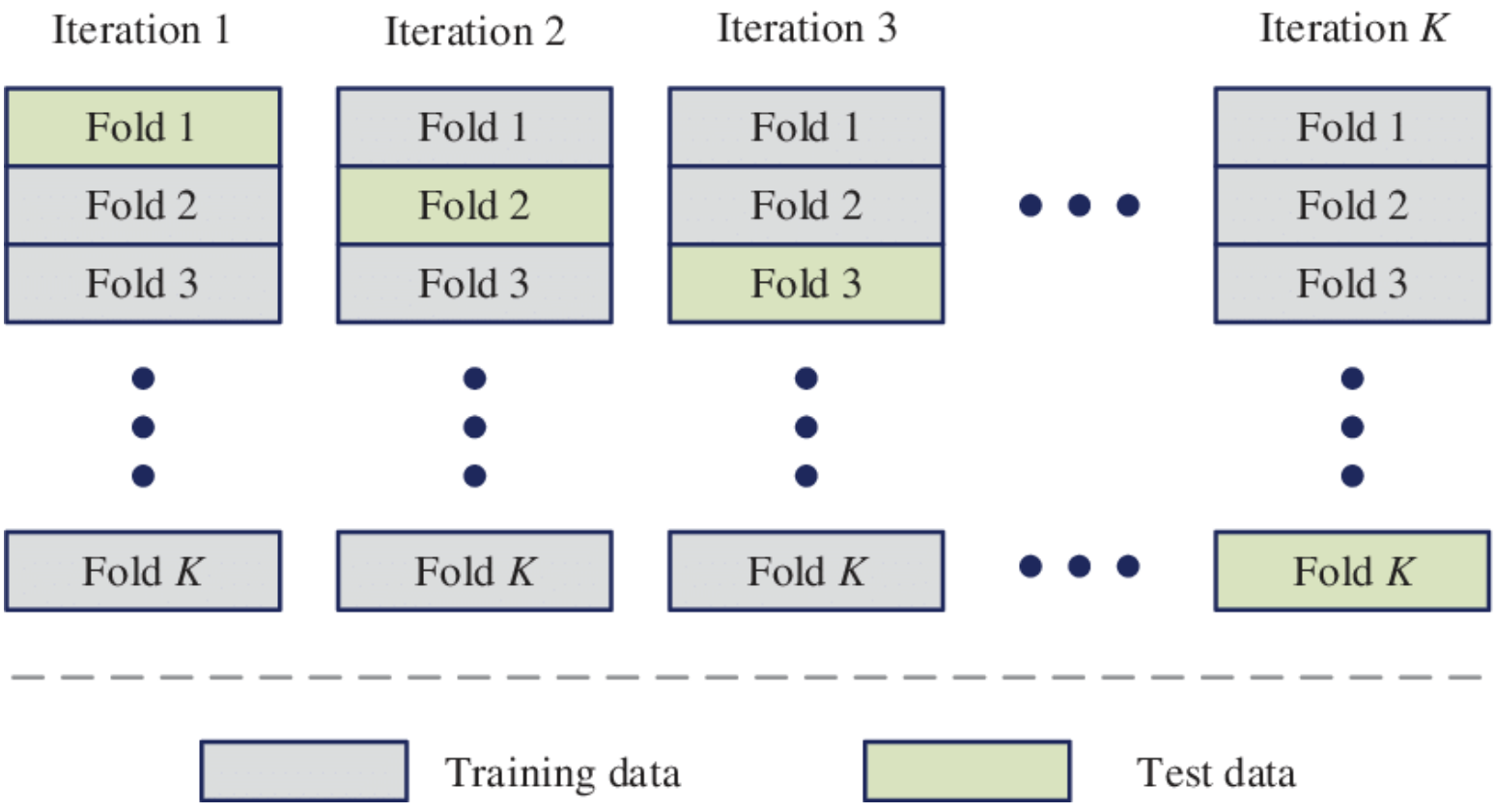
A more efficient solution is K-fold cross-validation:

1. Randomly divide the data set into K nonoverlapping groups, or *folds*, of roughly equal size.
2. For fold k $= 1$ to K:
   - Fit the model using all data points **not in** fold k.
   - For all points (y_i, x_i) **in** fold k, predict $\hat{y}_i$ using the fitted model.
   - Calculate $\widehat{\mathrm{RMSE}}_k$, the average error on fold k.
3. Calculate the cross-validated error rate as:

$$\mathrm{CV}_{(K)} = \frac{1}{K} \sum_{k=1}^{K} \widehat{\mathrm{RMSE}}_k$$

Let's draw a picture to build intuition for this procedure.

K-fold cross validation



The split of the data into folds is still random, but in a way that minimizes the overlap between each test set.

K-fold cross validation

A few notes:

- Typical values of K are 5 or 10. There's not a lot of deep theory here; those are just values that work well in practice.
- All candidate models should be fit on the same set of folds. (That is, don't create a different split to evaluate different models.)
- The K in K-fold cross validation is not the same K as the K in K-nearest neighbors. (The only thing they share in common is that K is just generic notation for an integer.)
- If $K = N$, i.e. the size of the data set, the resulting procedure is called "leave-one-out" cross validation (LOOCV). We'll talk about this later.

Let's go back to `loadhou.R` to see cross validation in action.

K-fold cross validation

There are two typical ways to select a model using cross validation:

1. The "min" rule: choose the model with the best cross-validated error.
2. The "1SE" rule: choose the simplest model whose cross-validated error is within one standard error of the minimum.

For each model, we estimate the standard error of that model's cross-validated RMSE as:

$$\text{stderr} \approx \frac{\text{sd}\left(\widehat{\text{RMSE}}_1, \widehat{\text{RMSE}}_2, \ldots, \widehat{\text{RMSE}}_K\right)}{\sqrt{K}}$$

(A decent approximation even though the $\widehat{\text{RMSE}}_k$'s aren't IID.)

Summary

Nonparametric models (like KNN):

- can't be written down in terms of simple math functions
- can adapt to complex functions, but have to be reined in somehow, so that they don't overfit
- usually have "tuning" parameters (like K in KNN) that govern the trade-off between bias and variance.

Out-of-sample performance is the true test of a model:

- In-sample RMSE is too optimistic, often wildly so.
- Cross-validation using K=5 or K=10 is a practical way to quantify out-of-sample performance.
- For close calls, the 1SE rule is a widely accepted way to actually pick the model. *Simplicity is a virtue*, and the 1SE rule finds the simplest model that doesn't forfeit any statistically noticeable gains in performance.