Principles of statistical learning

ECO 395M

James Scott (UT-Austin)

Reference: Introduction to Statistical Learning, Chapters 1-2

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. Intro to classification

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." We've already seen a simple example (OLS with one x feature)

Introduction to predictive modeling

A useful way to frame this problem is to think that y and x are related like this:

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

where:

- y_i is a scalar *outcome* or *target* variable
- $x_i = (x_{i1}, x_{i2}, \ldots x_{iP})$ is a vector of features
- f is an unknown function

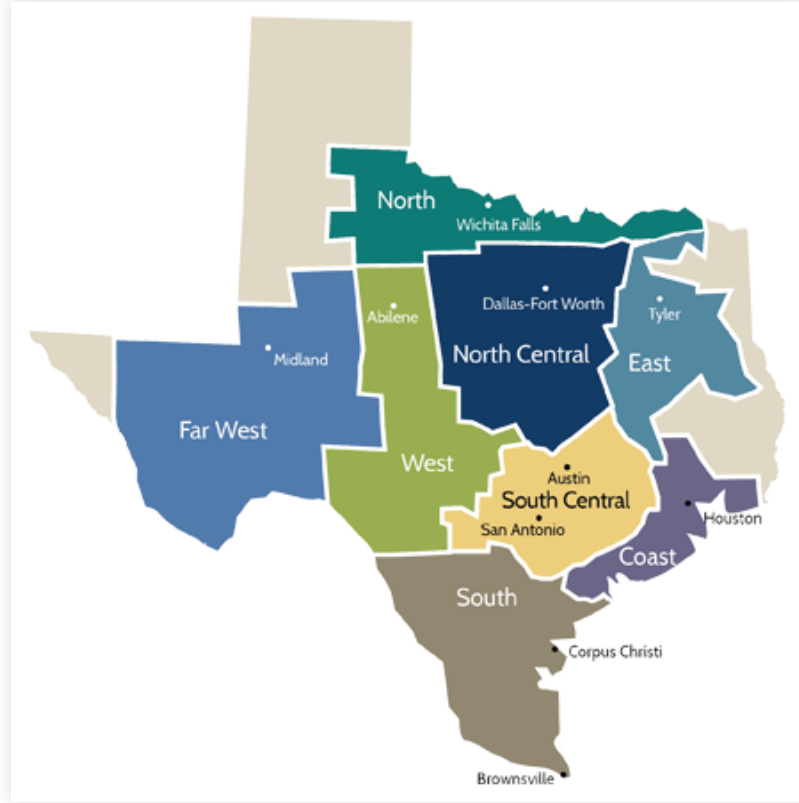
Our main purpose is to *learn* $f(x)$ from the observed data.

Example: predicting electricity demand

Example: predicting electricity demand

ERCOT operates the electricity grid for 75% of Texas.

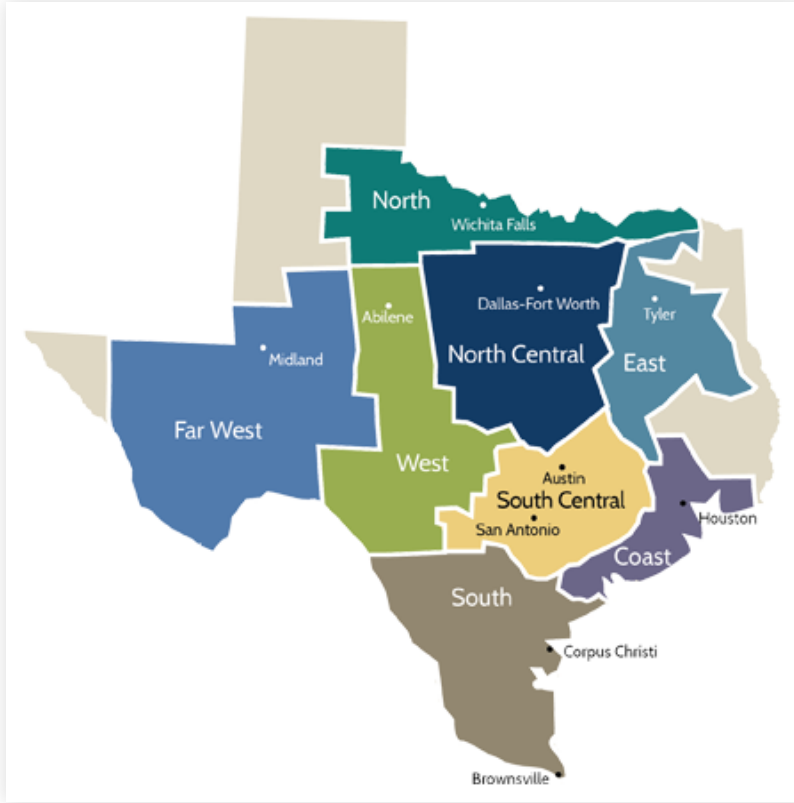
The 8 ERCOT regions are shown at right.



Example: predicting electricity demand

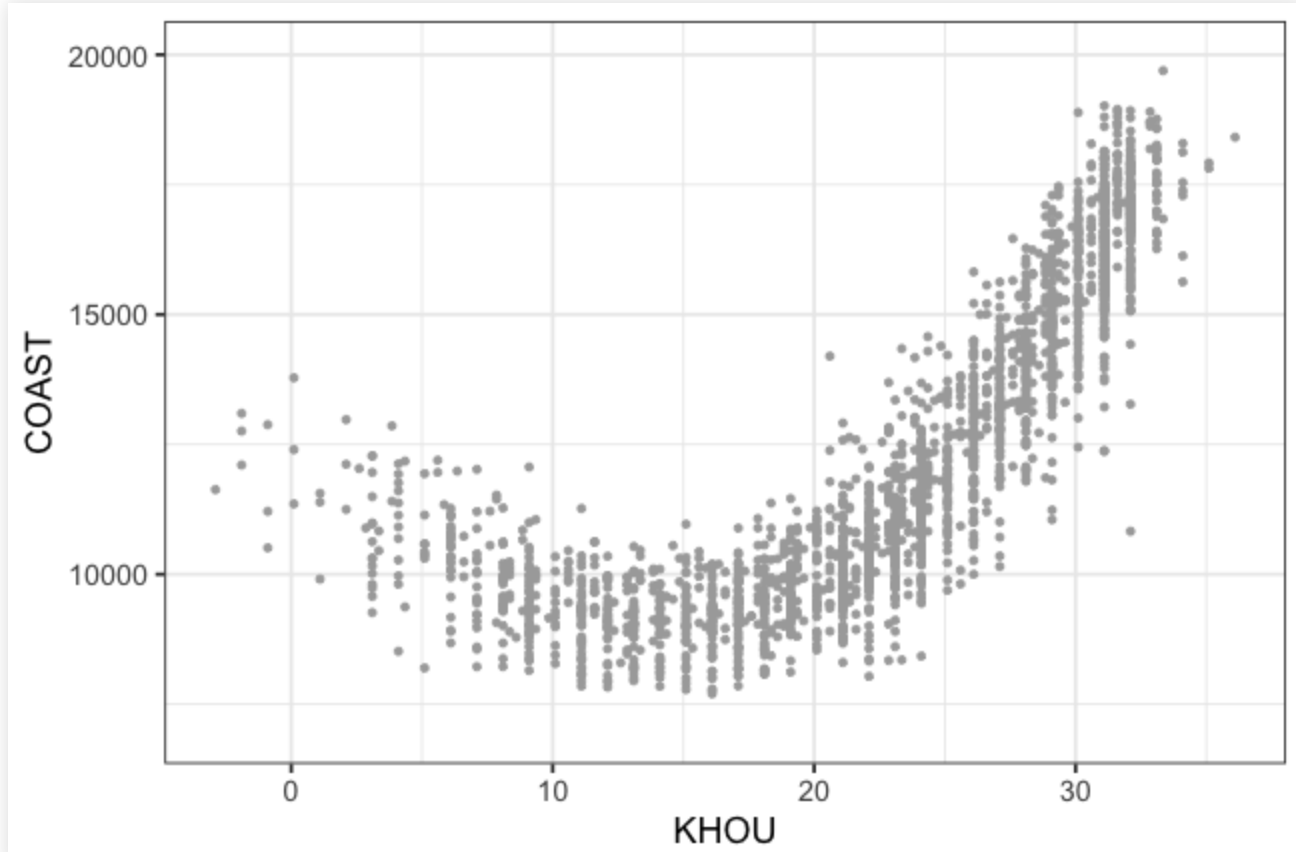
We'll focus on a basic prediction task:

- y = demand (megawatts) in the Coast region at 3 PM, every day from 2010-2016.
- x = average daily tempature measured at Houston's Hobby Airport (degrees Celsius)
- Date sources: scraped from the ERCOT website and the National Weather Service

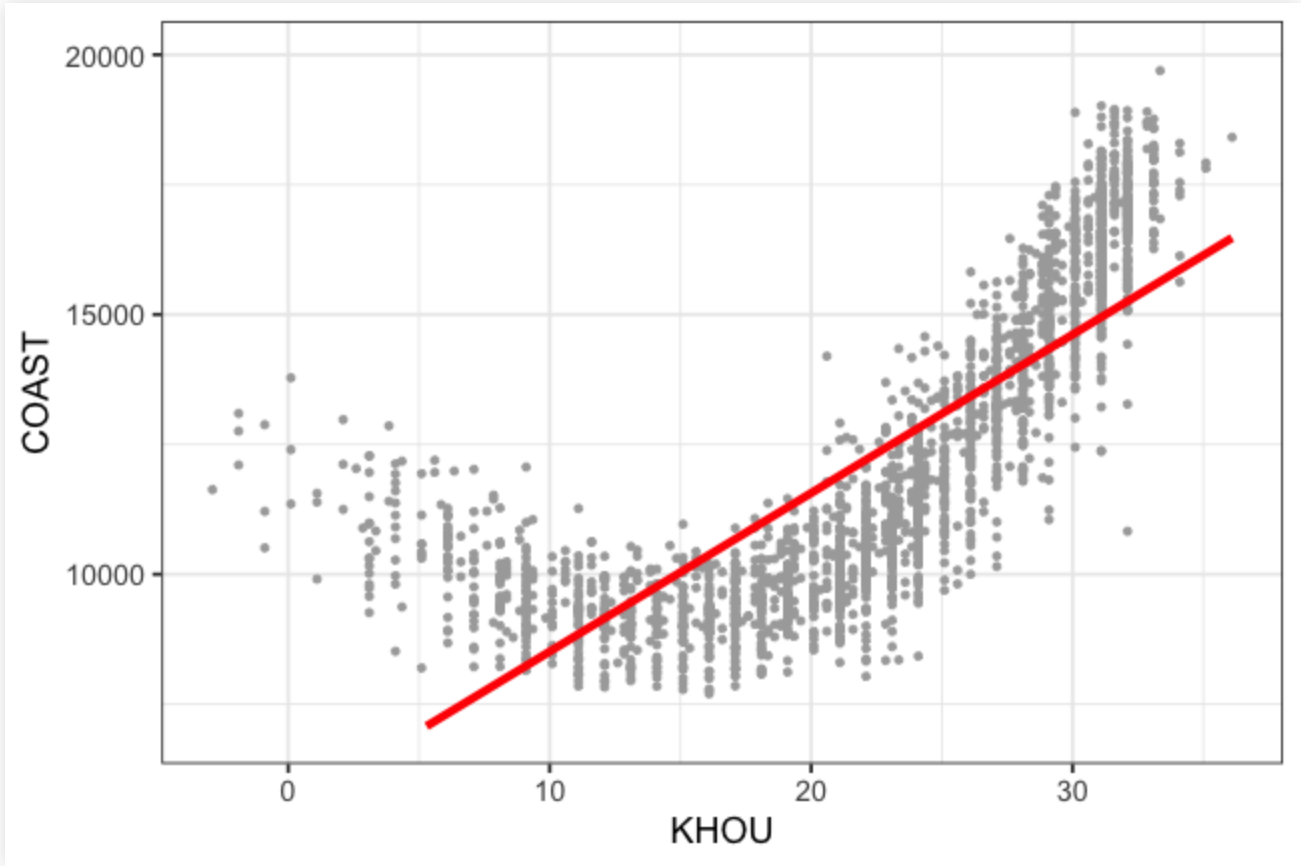


Demand versus temperature

```
ggplot(data = loadhou) +
  geom_point(mapping = aes(x = KHOU, y = COAST), color='darkgrey') +
  ylim(7000, 20000)
```

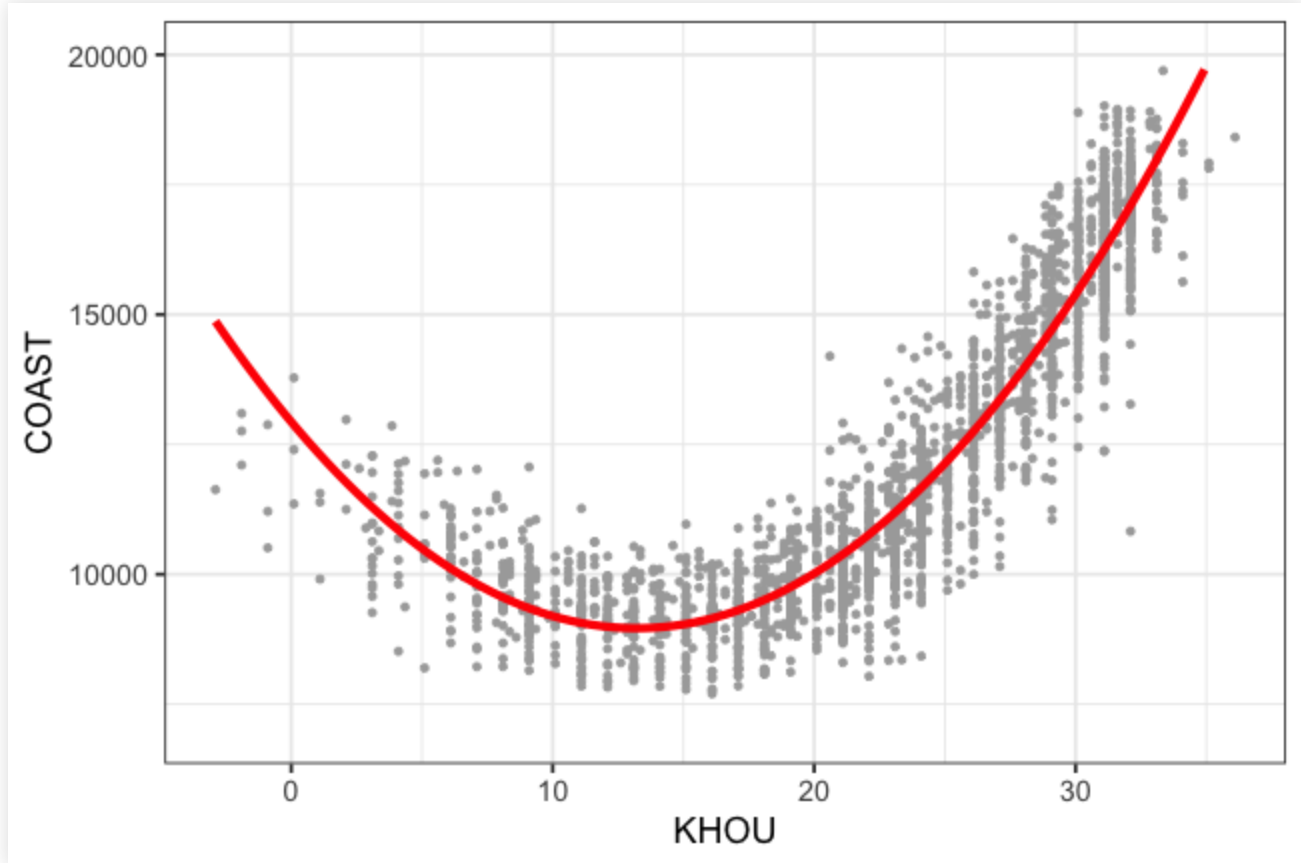


A linear model?



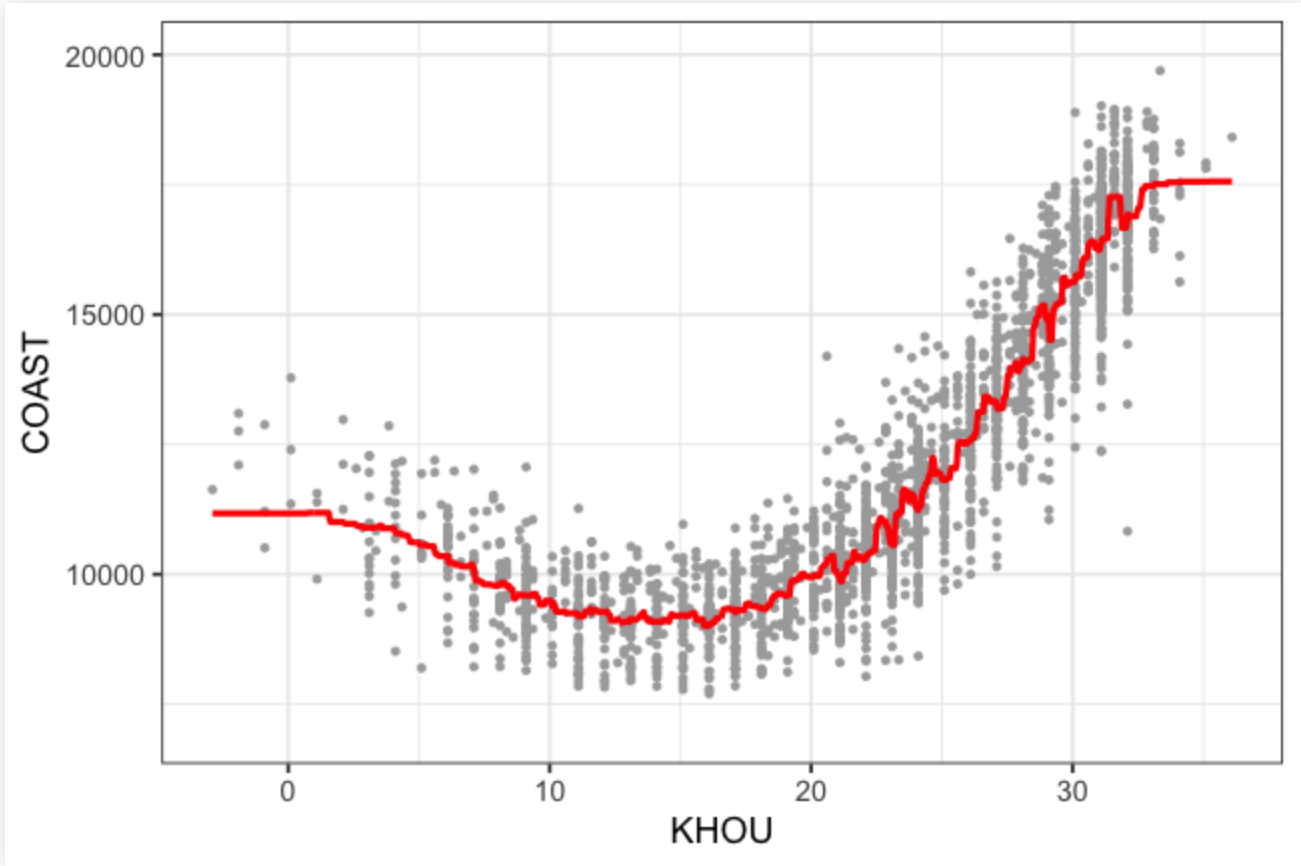
$$f(x) = \beta_0 + \beta_1 x$$

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 the prediction for y?
- What about if $x = 30$?

How do we estimate f?

Our *training data* consists of pairs

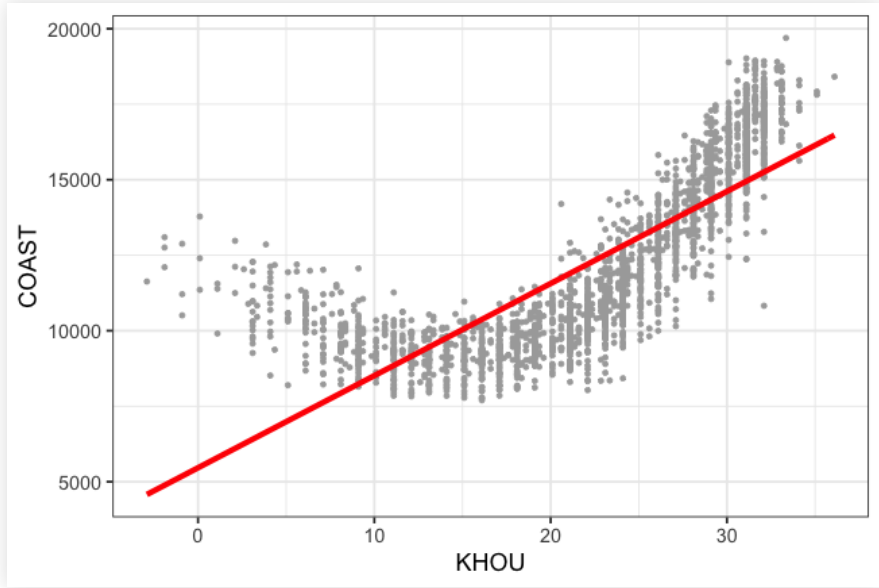
$$D_{\text{tr}} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

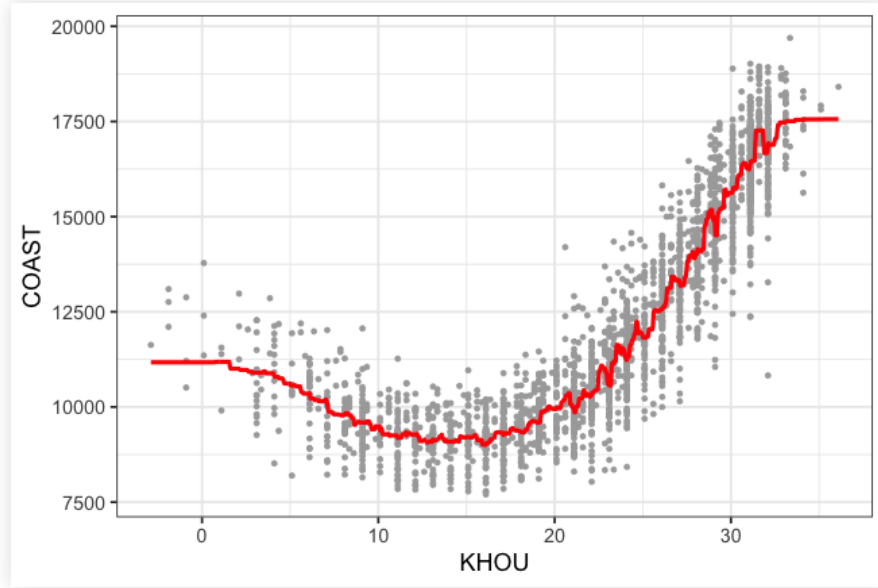
We then use some statistical method to estimate $f(x)$. Here "statistical" just means "we apply some recipe to the data."

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

A quick comparison





Parametric:

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

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:

$$L(f) = \sum_{i=1}^{N} (y_i - f(x_i))^2$$

$$= \sum_{i=1}^{N} (y_i - \beta_0 - \beta_1 x_i)^2$$

3. Find the parameters that minimize the loss function.

Estimating k-nearest neighbors

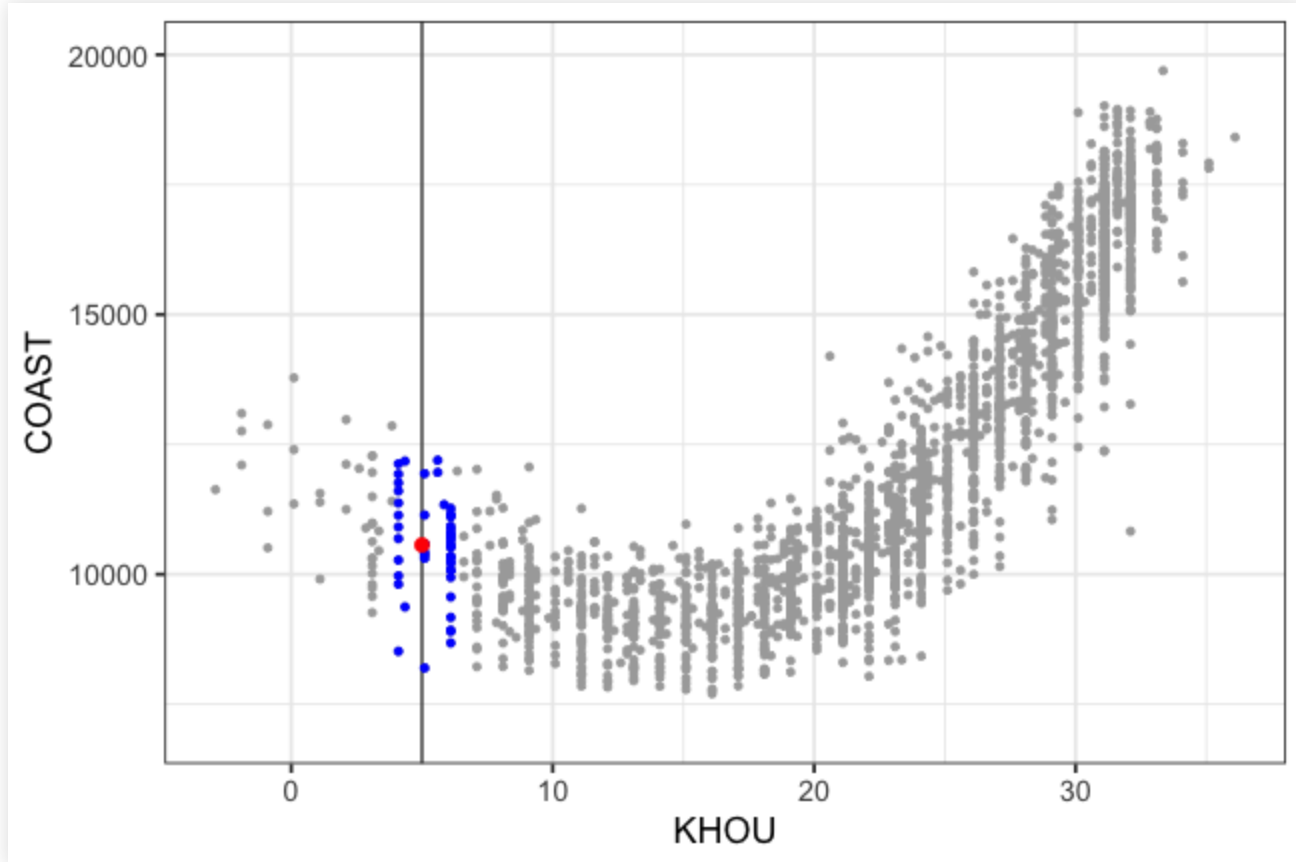
Suppose we have our training data in the form of (x_i, y_i) pairs.

Now we want to predict y at some new point $x^\star$.

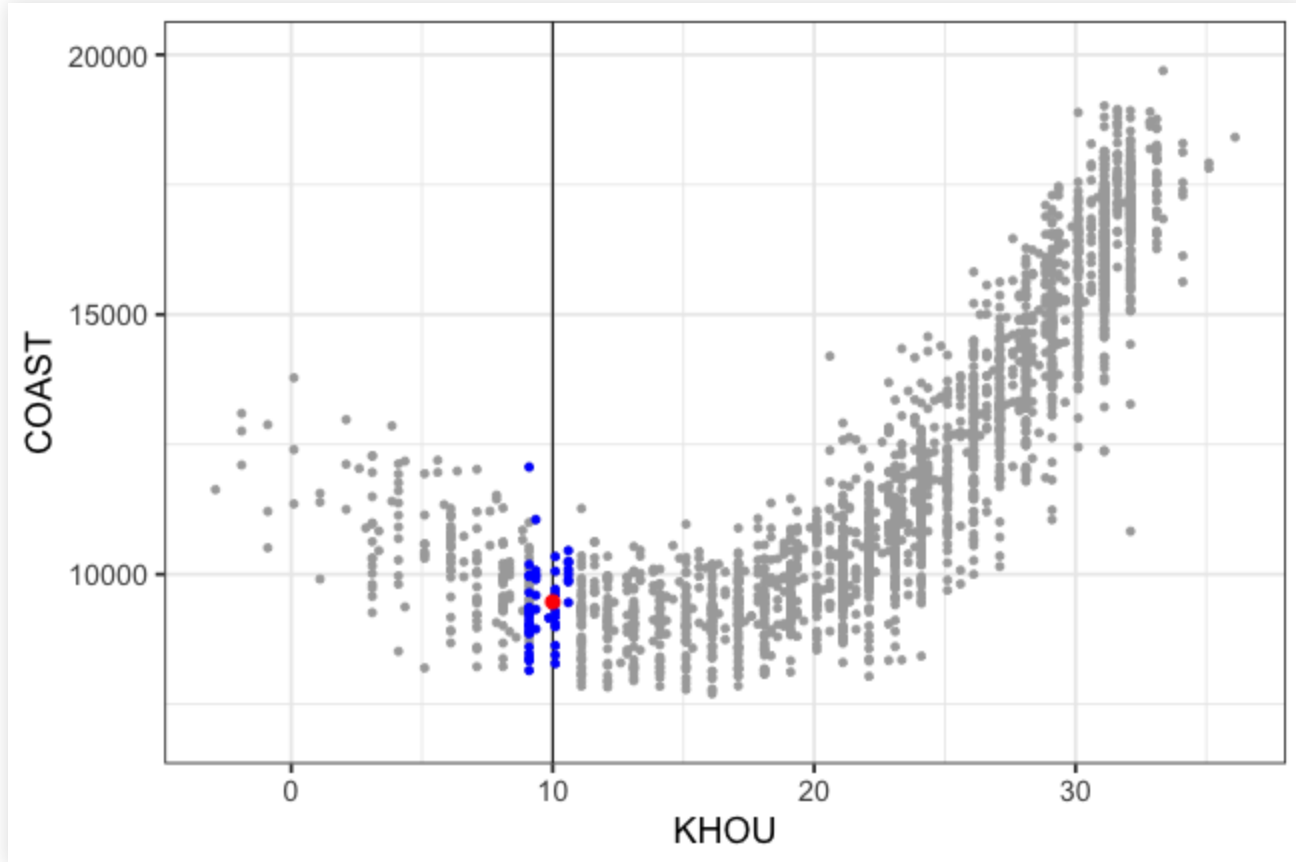
1. Pick the K points in the training data whose x_i values are closest to $x^\star$.
2. Average the y_i values for those points and use this average to estimate $f(x^\star)$.

There are no parameters (i.e. the β's in a linear model) 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.)

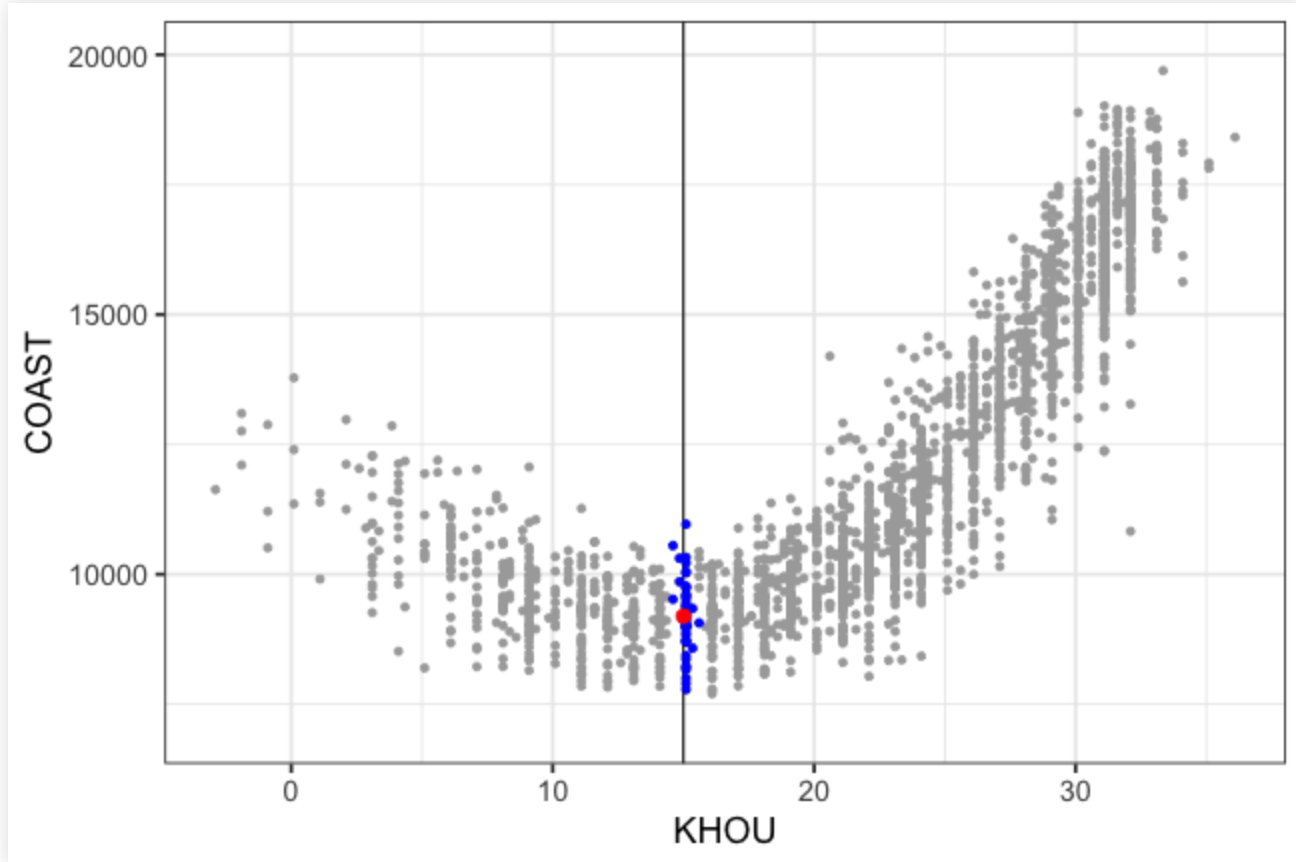
At x=5



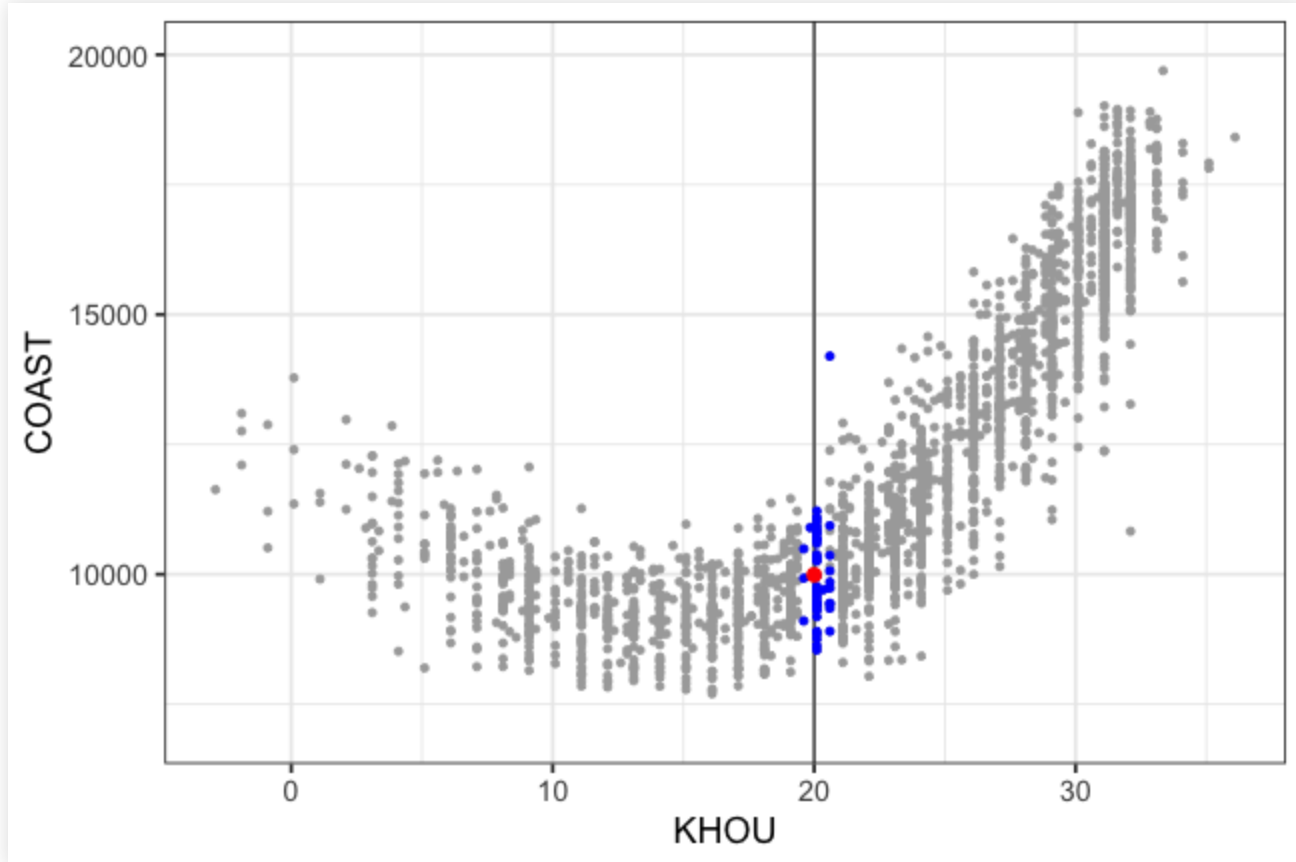
At x=10



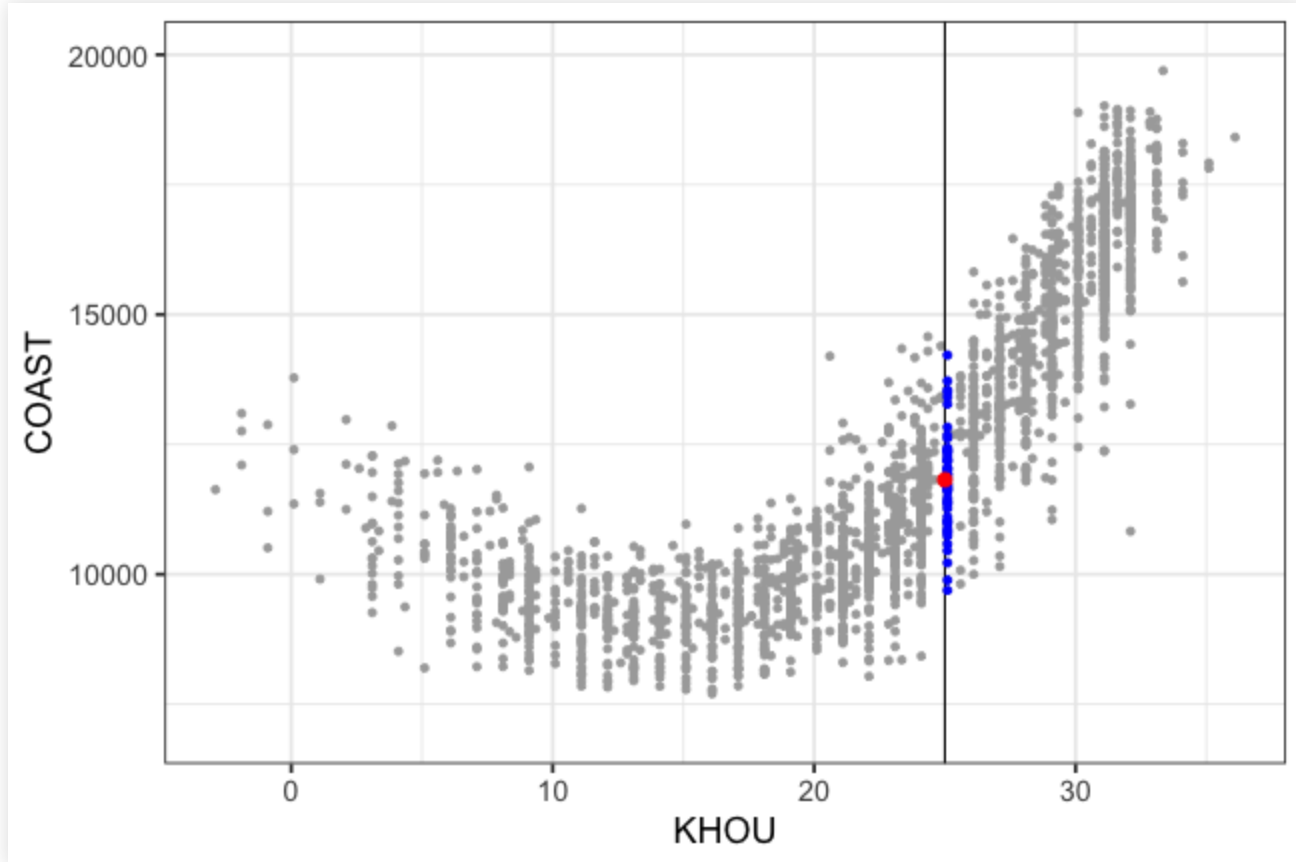
At x=15



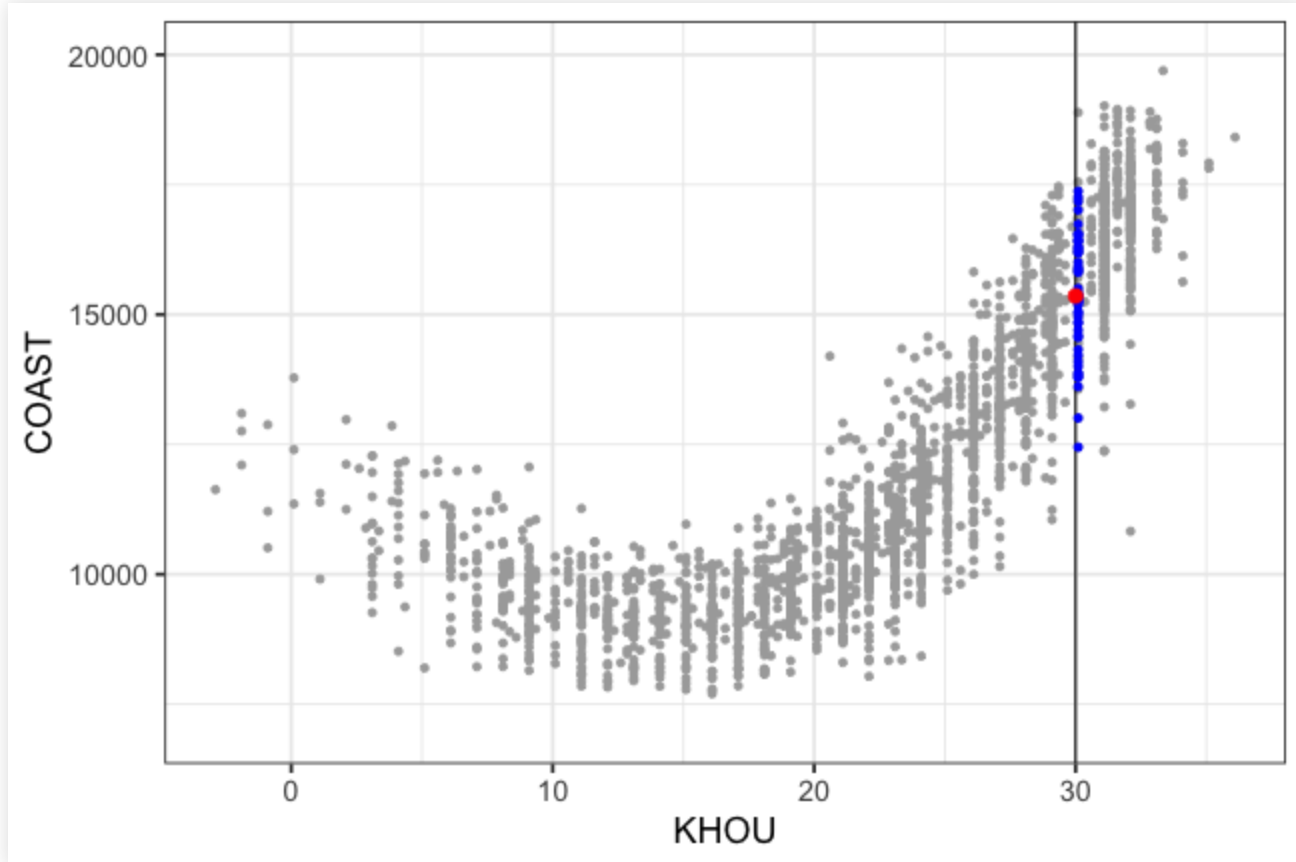
At x=20



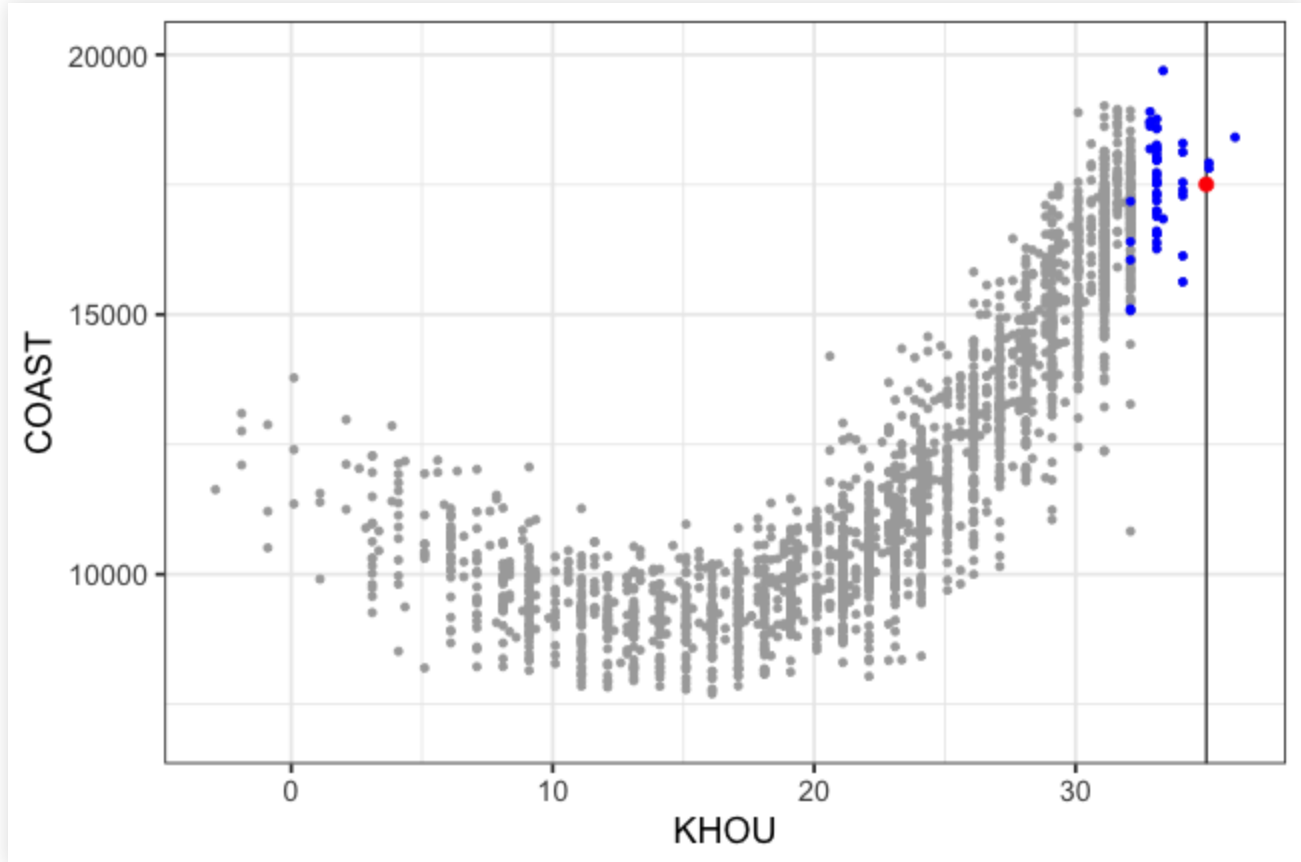
At x=25



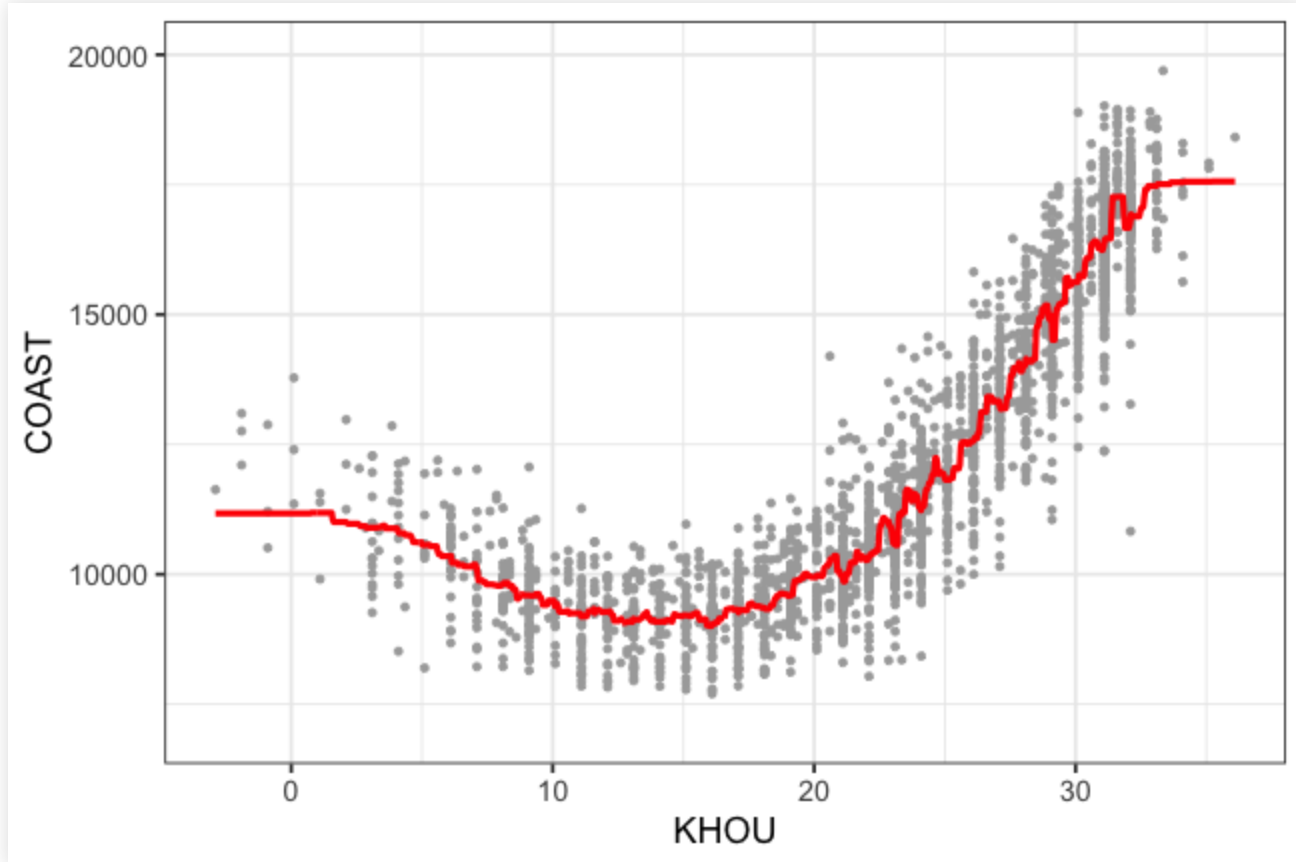
At x=30



At x=35



The predictions across all x values

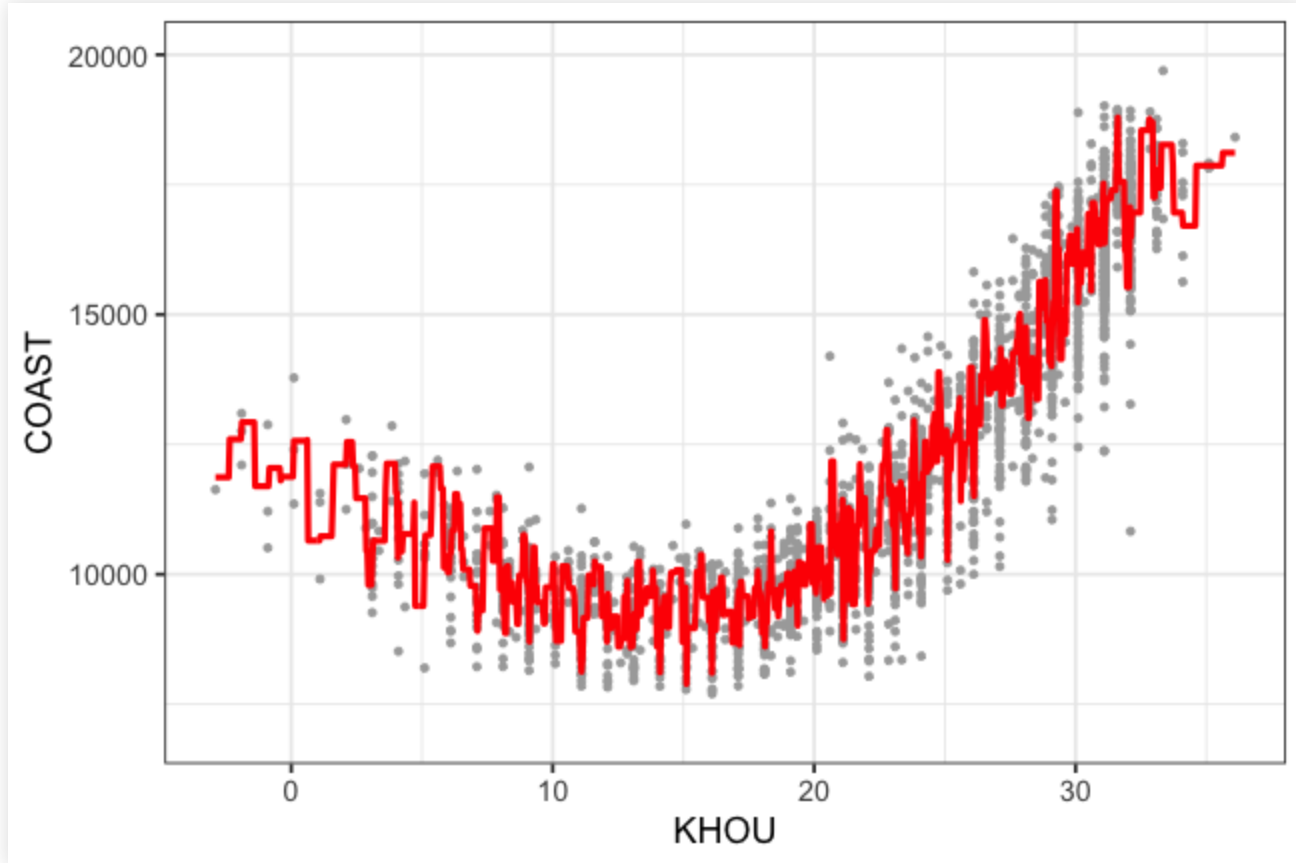


Two questions

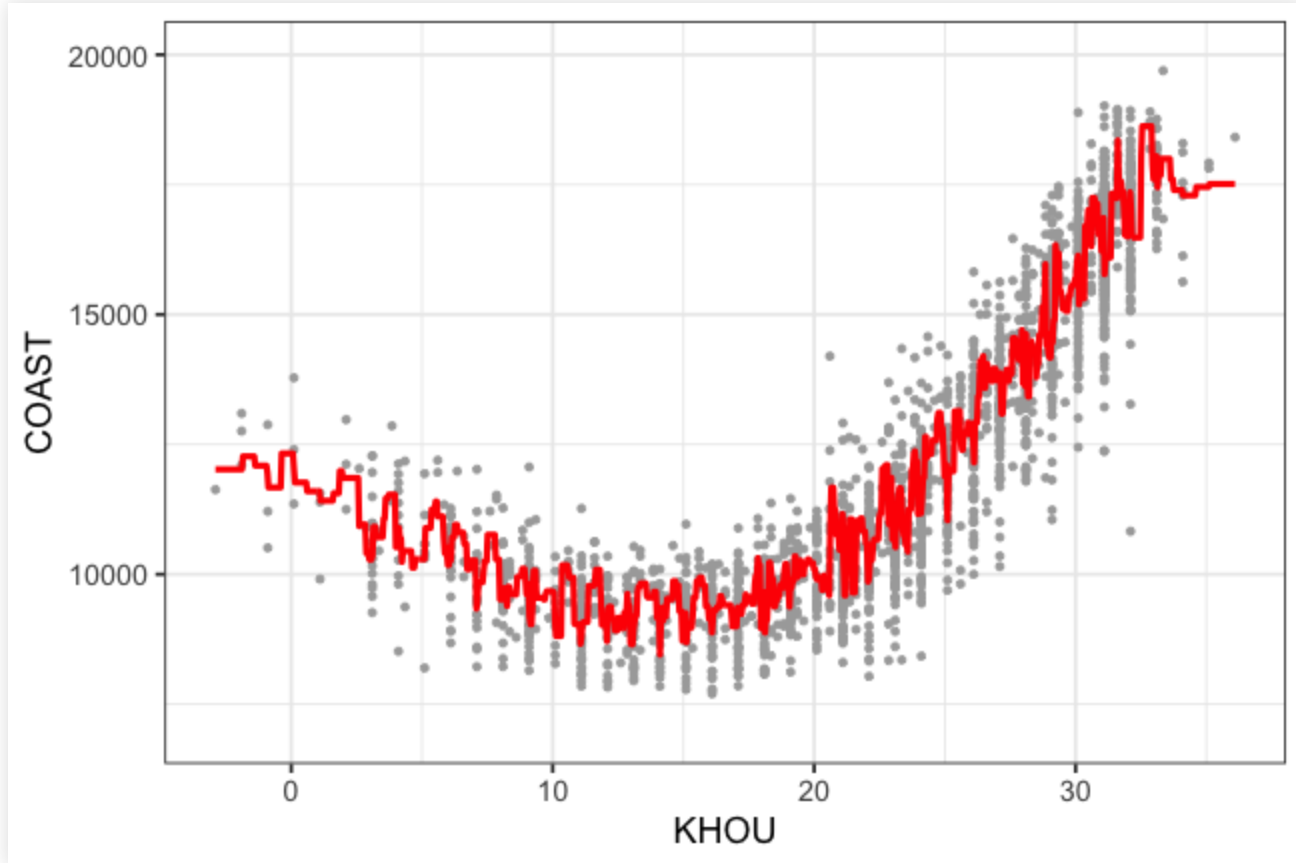
So why average the nearest $K = 50$ neighbors? Why not $K = 2$, or $K = 200$?

And if we're free to pick any value of K we like, how should we choose?

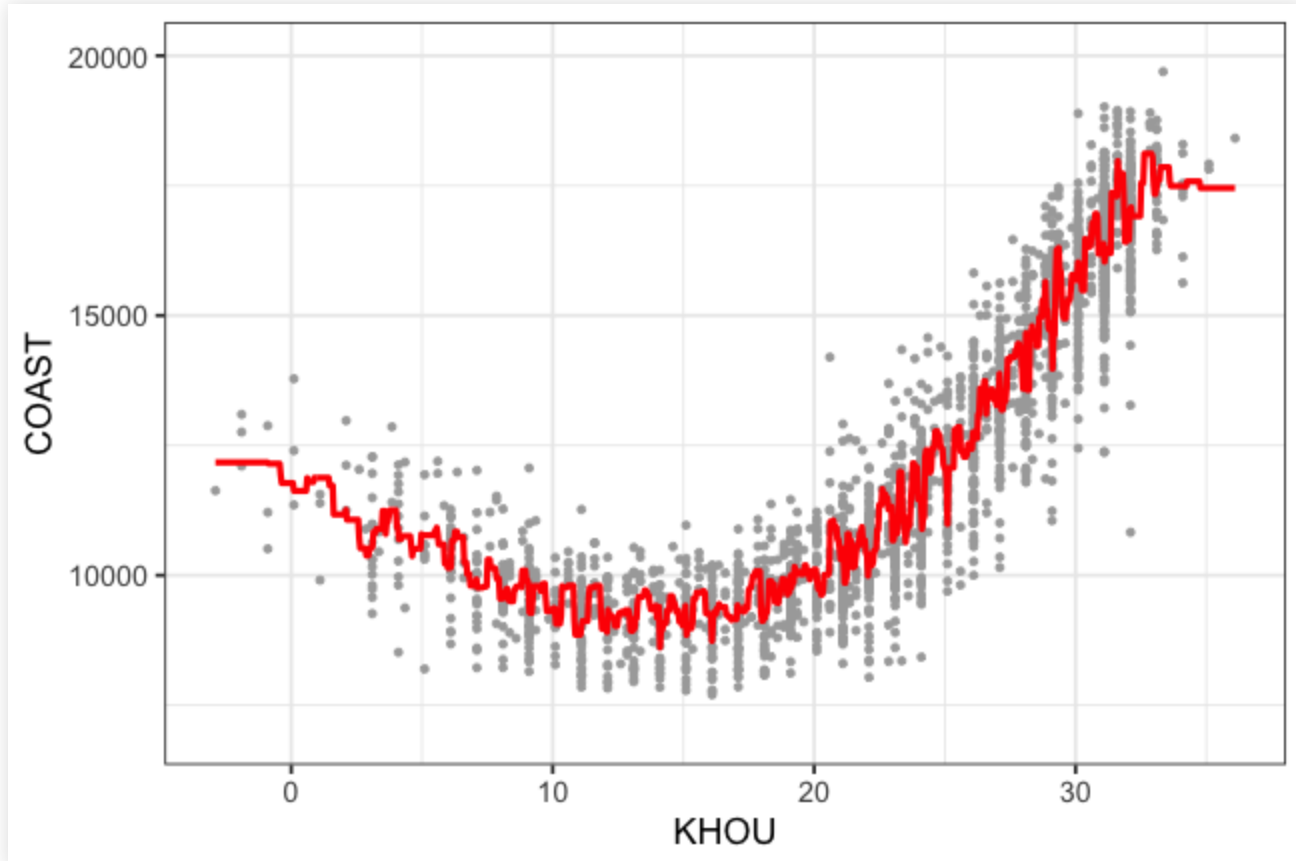
K=2



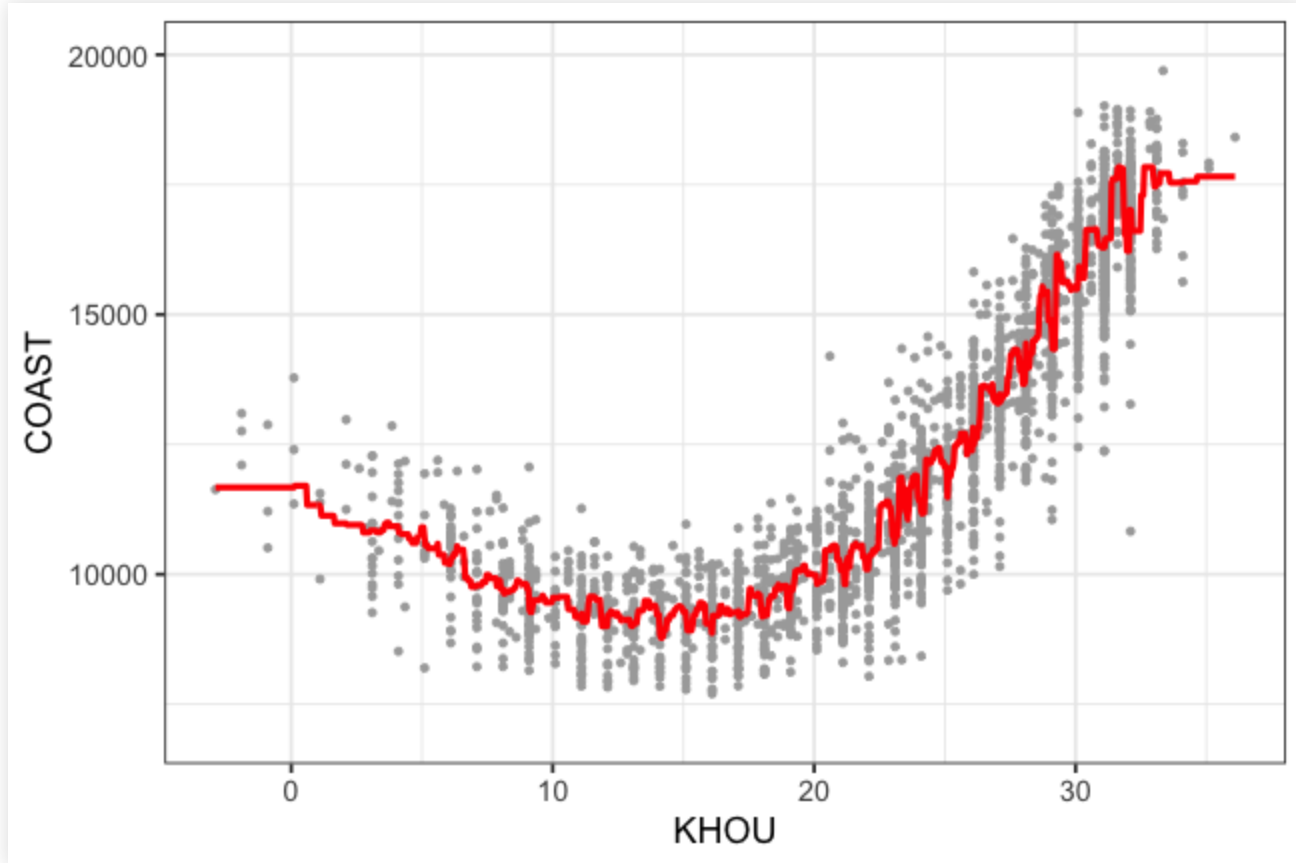
K=5



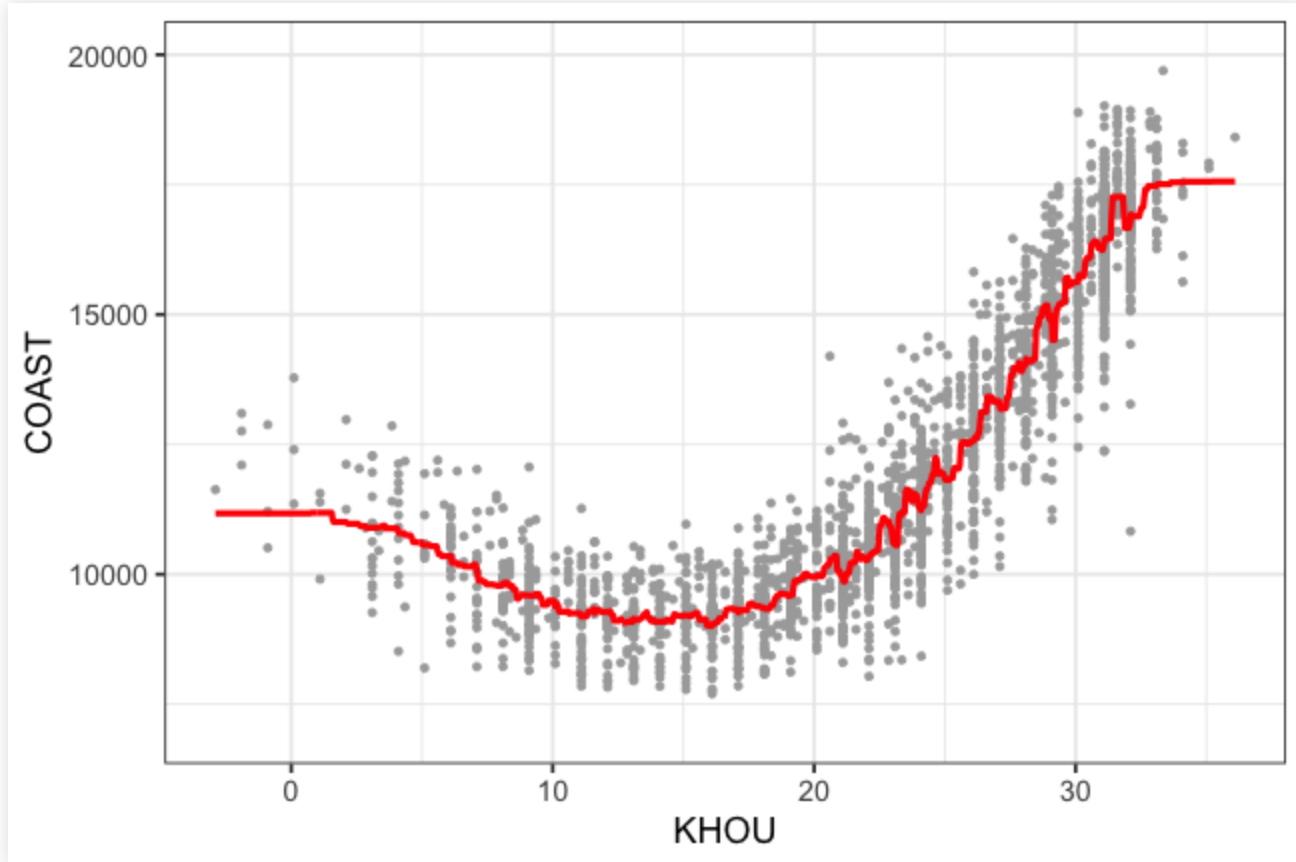
K=10



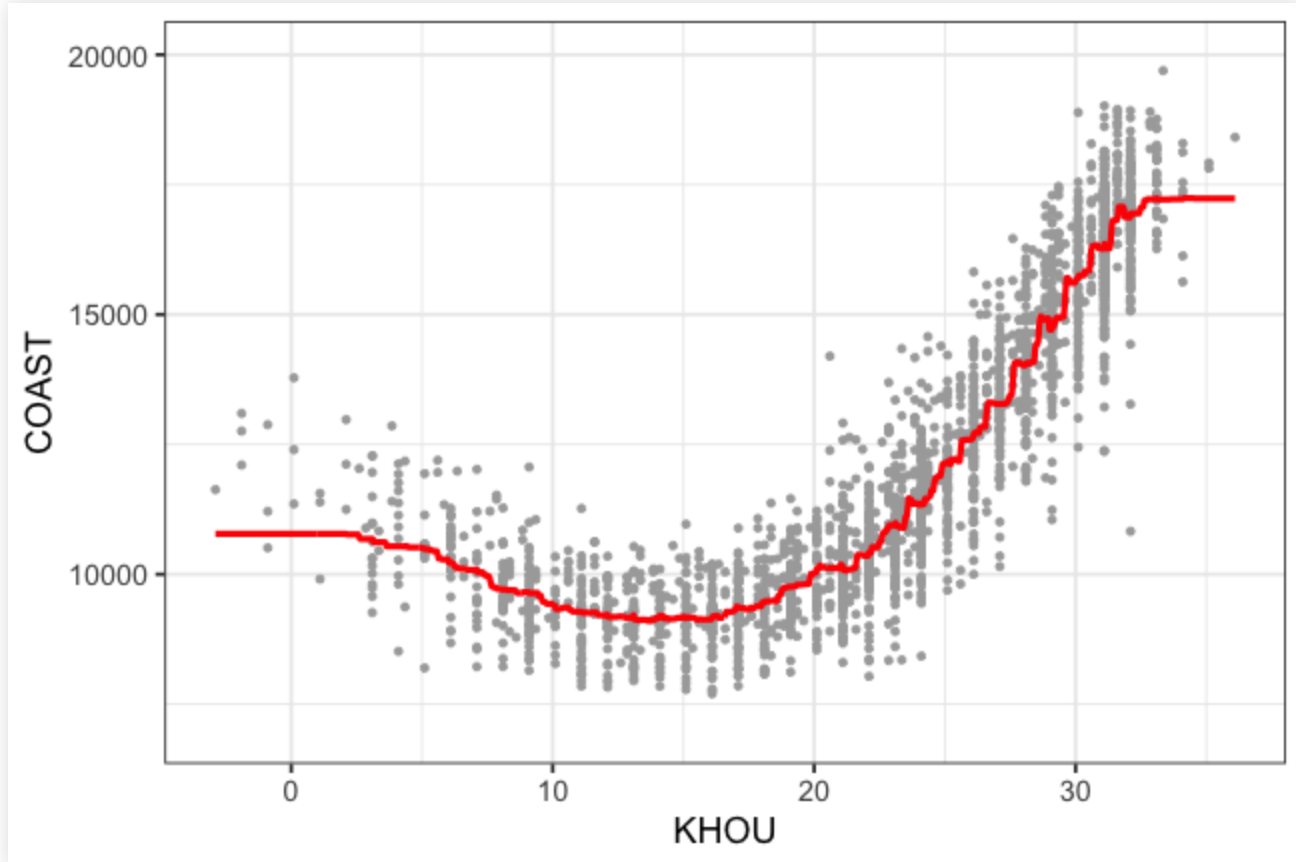
K=20



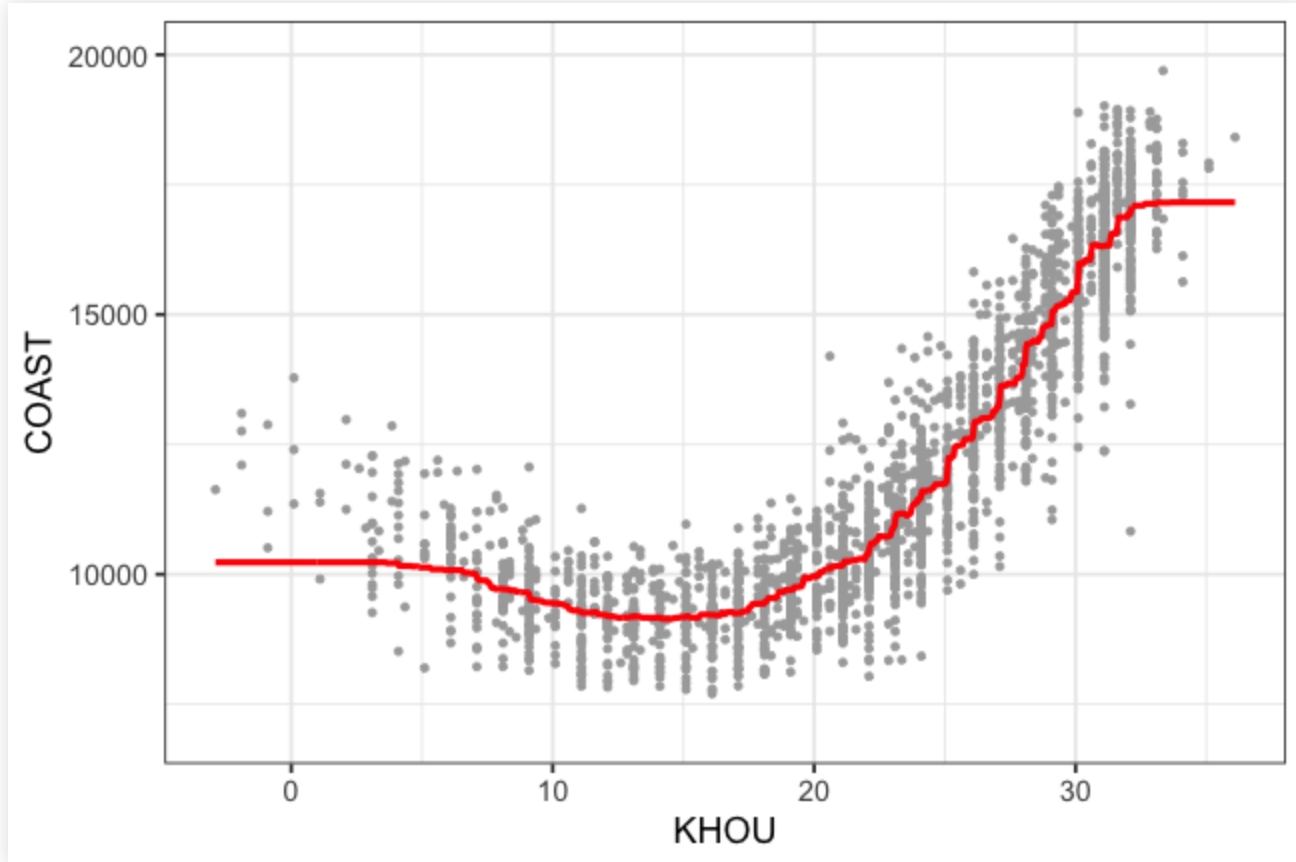
K=50



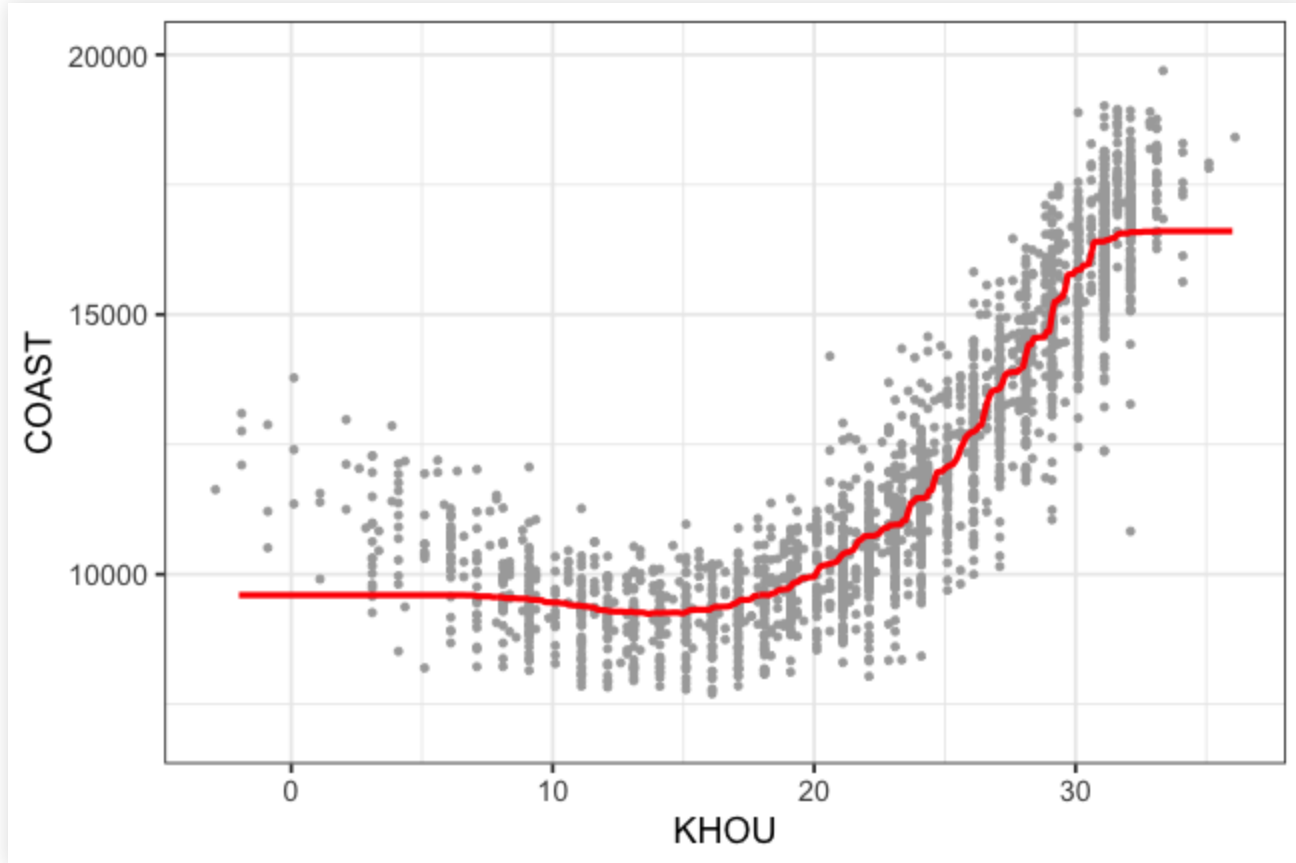
K=100



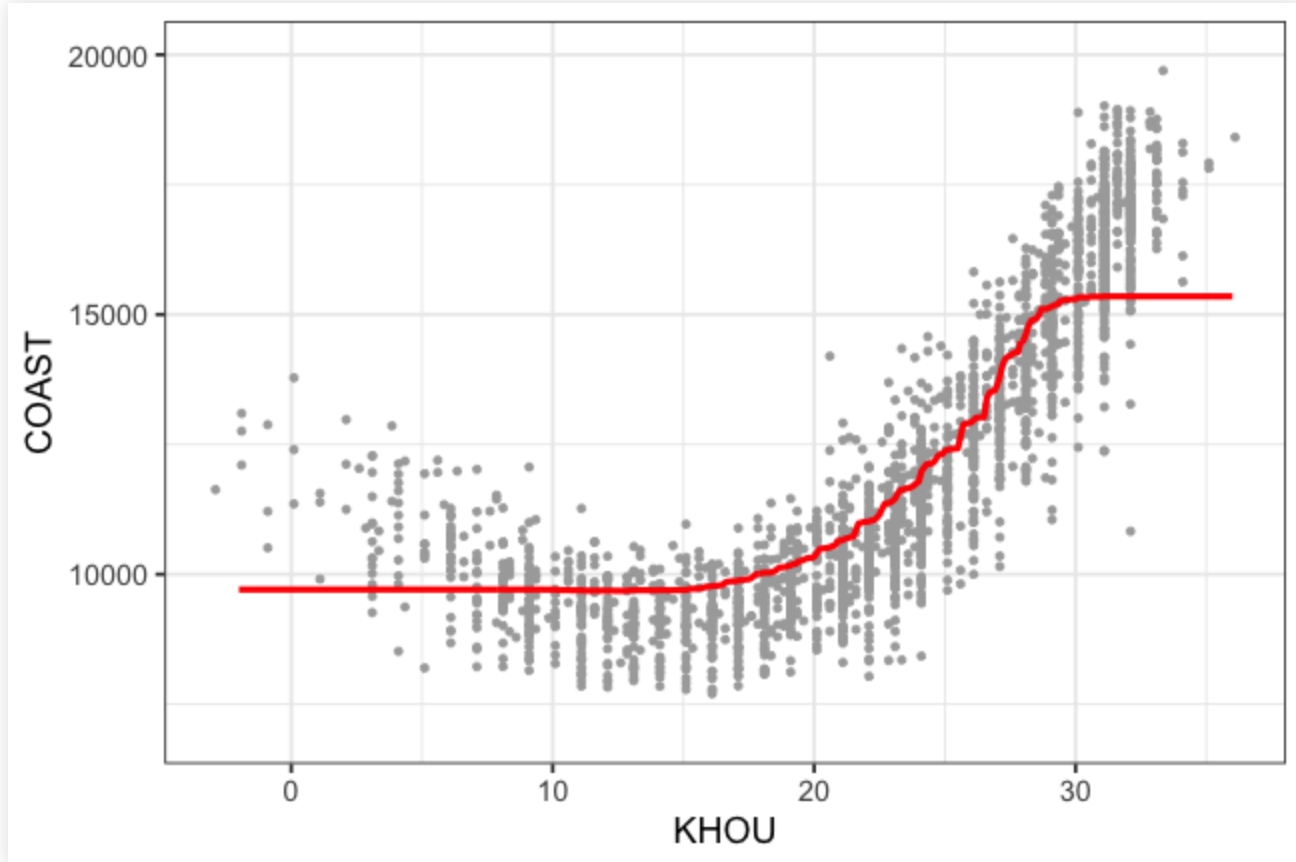
K=200



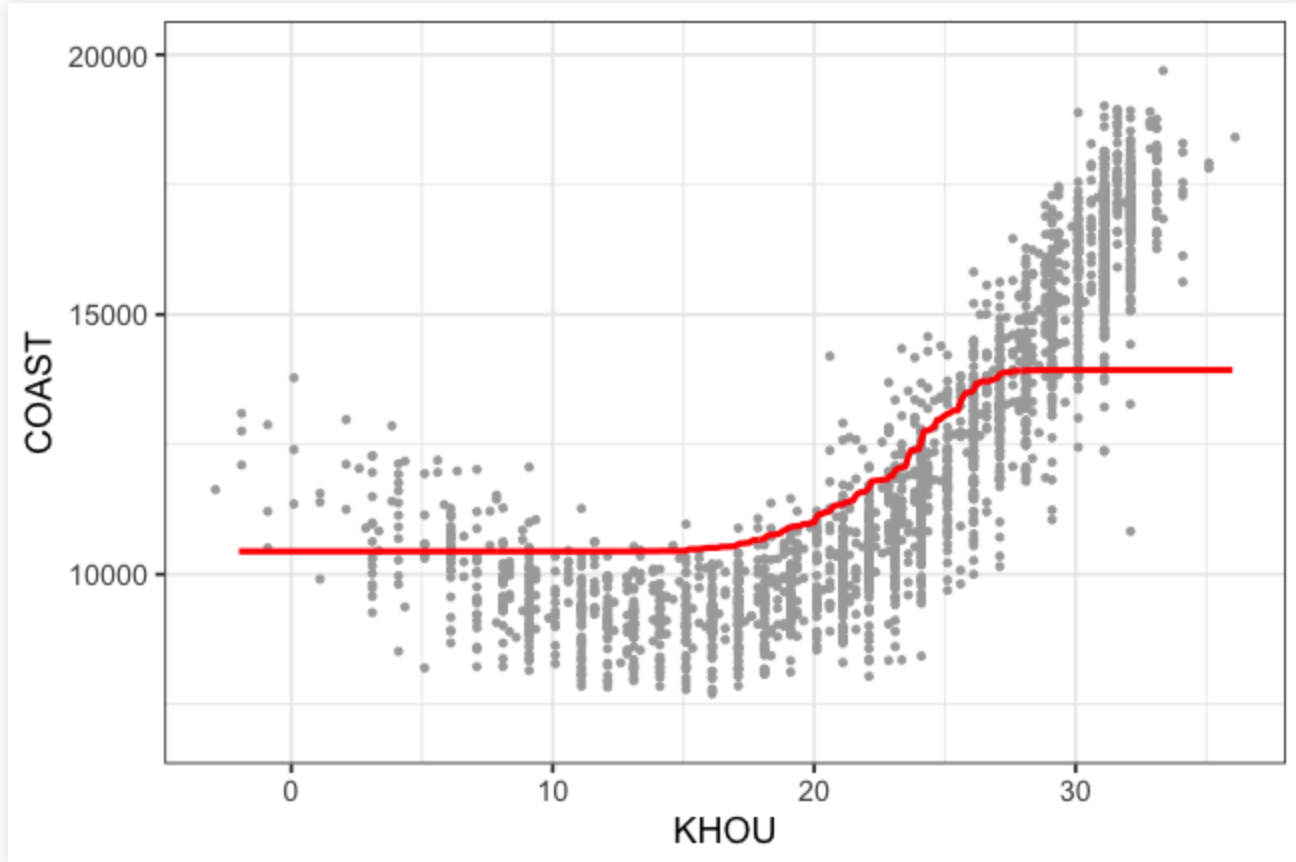
K=500



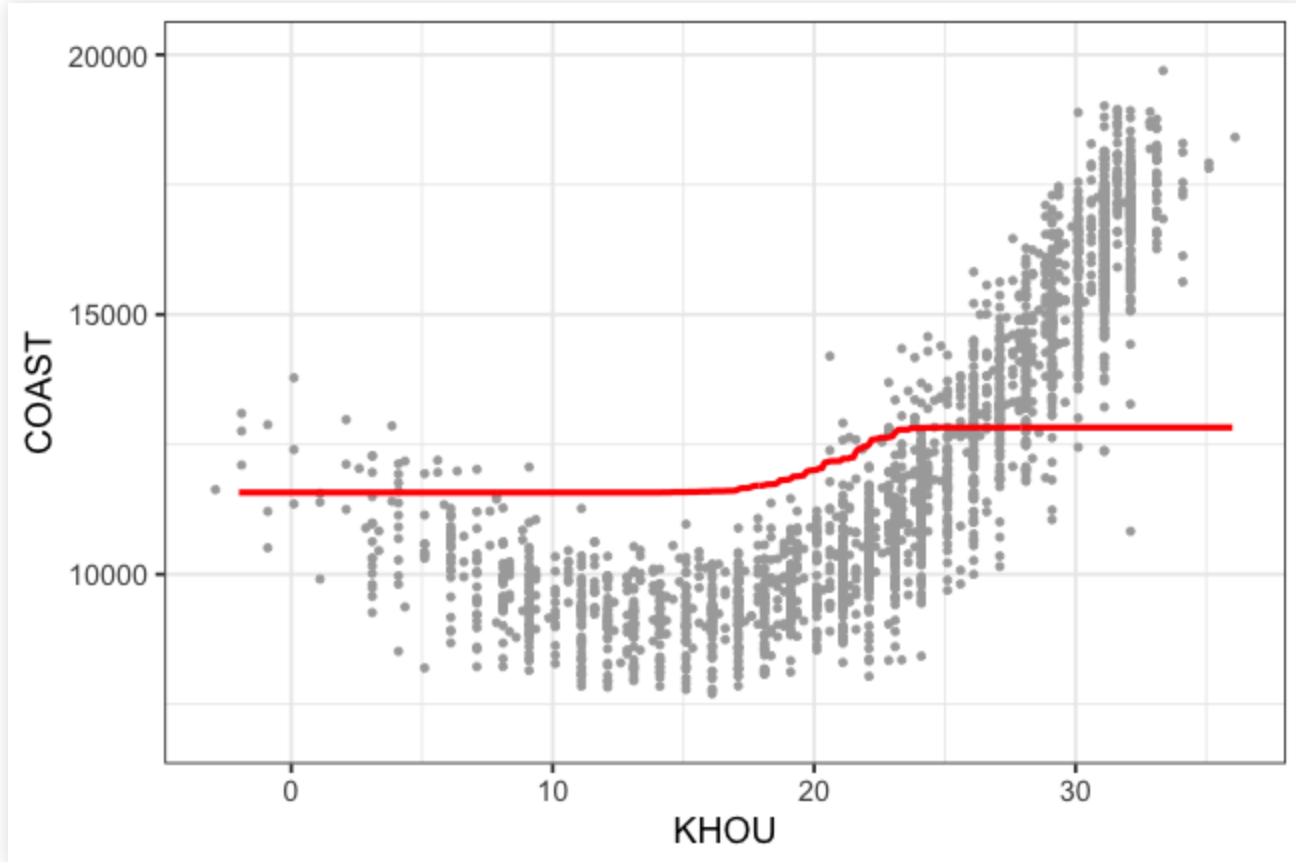
K=1000



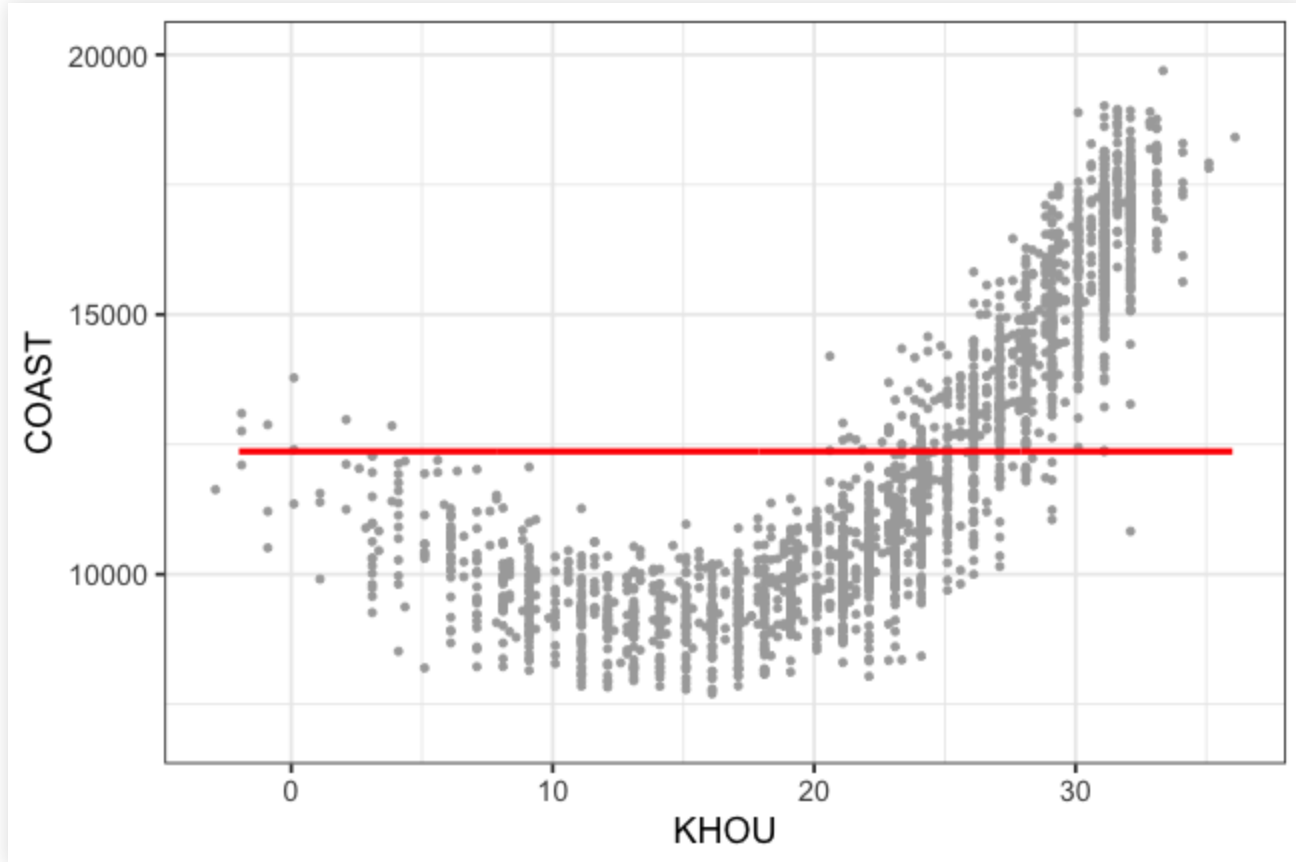
K=1500



K=2000



K=2357



Complexity, generalization, and interpretion

As we have seen in the examples above, there are lots of options in estimating $f(x)$.

Some methods are very flexible, and some are not… why might we ever choose a less flexible model?

1. Simple, more restrictive methods are usually easier to interpret

2. More importantly, it is often the case that simple models make *more accurate predictions* than very complex ones.

Measuring accuracy

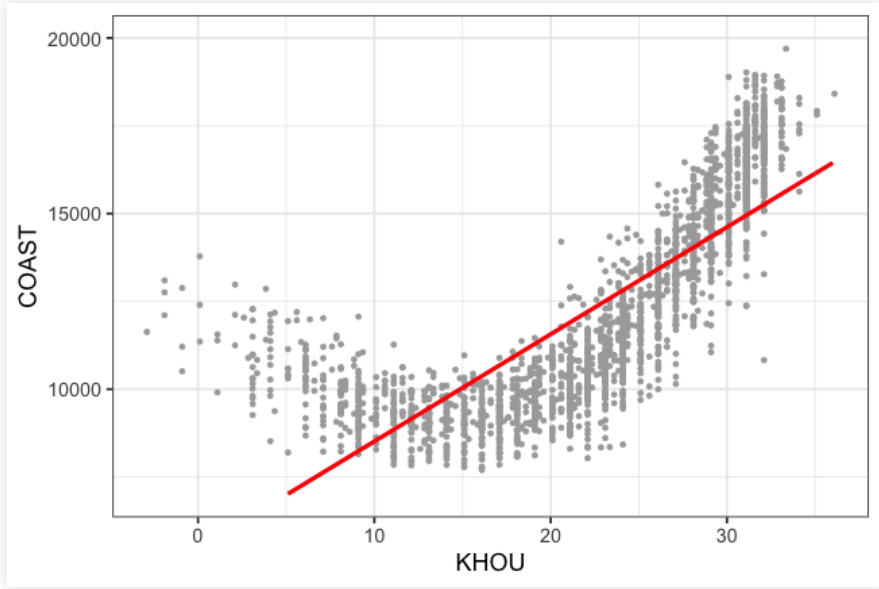
Let's turn to a deceptively subtle question: how accurate is each of these models?

A standard measure of accuracy is the root mean-squared error, or RMSE:

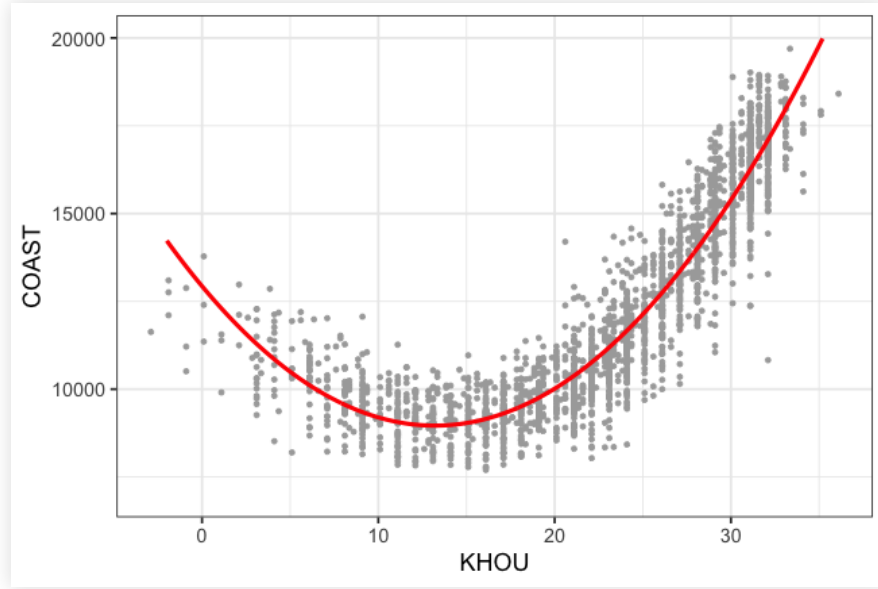
$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2}$$

This measures, on average, how large are the "mistakes" (errors) made by the model on the training data. (OLS minimizes this quantity.)

Measuring accuracy: linear vs. quadratric

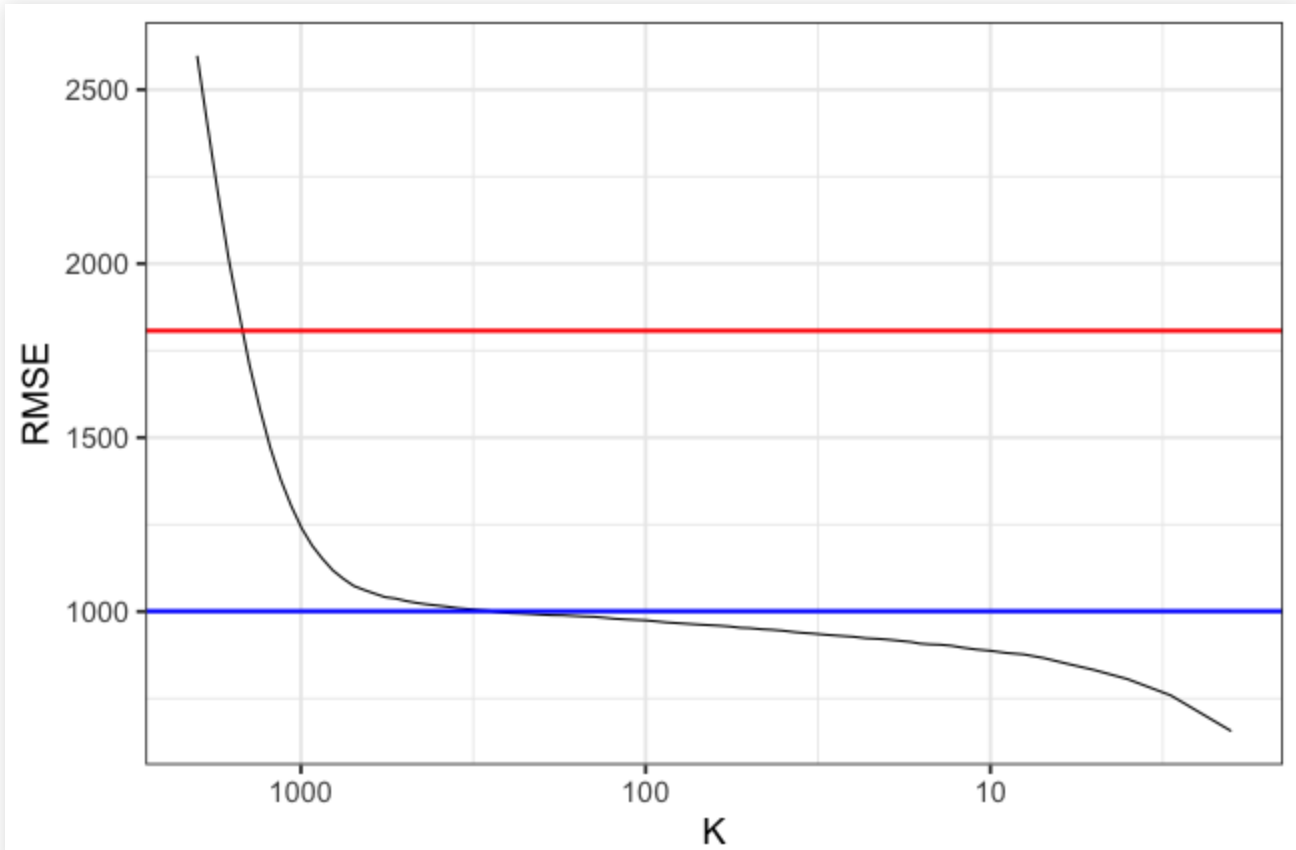


$RMSE = 1807$

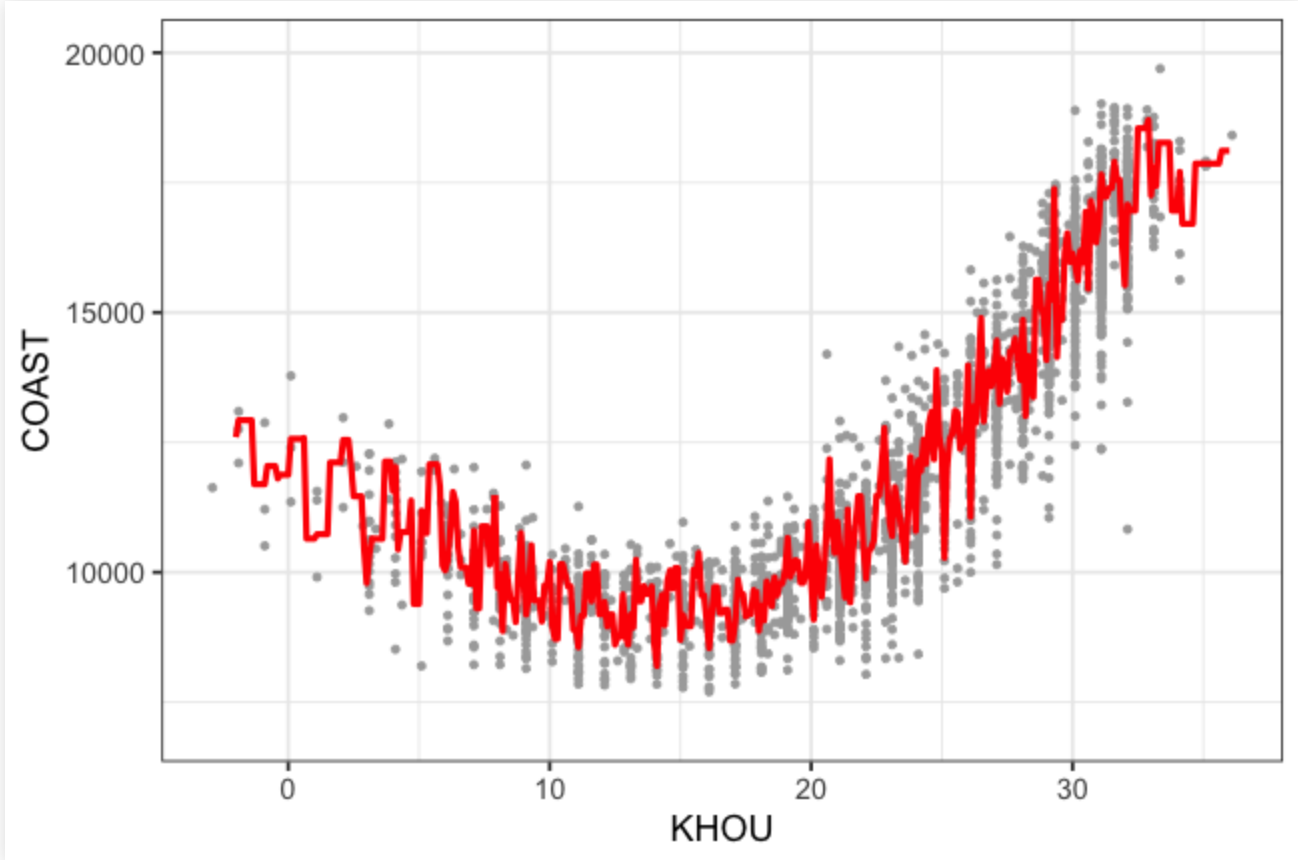


$RMSE = 1001$

Measuring accuracy: RMSE versus K

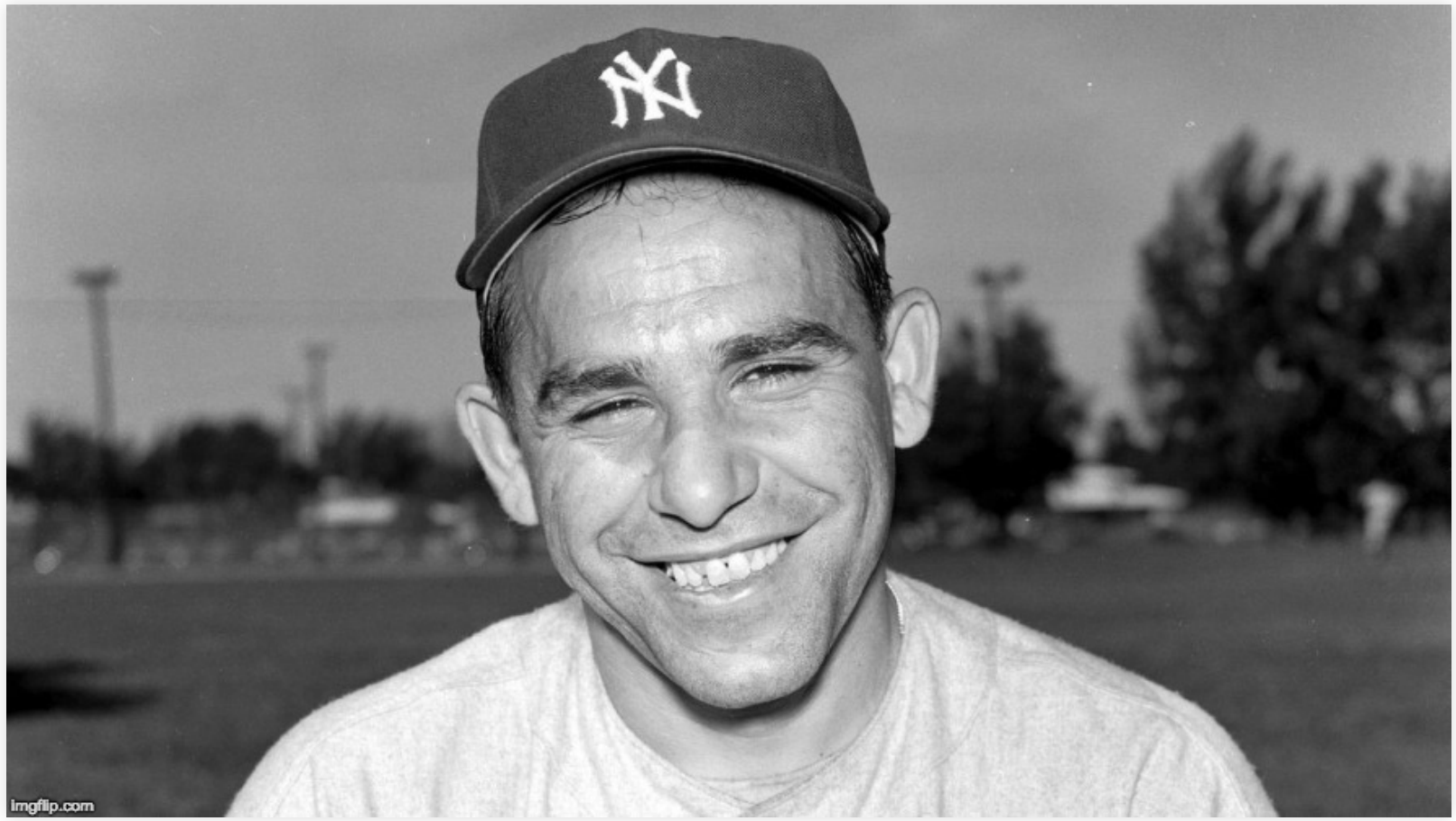


So we should pick K=2, right?

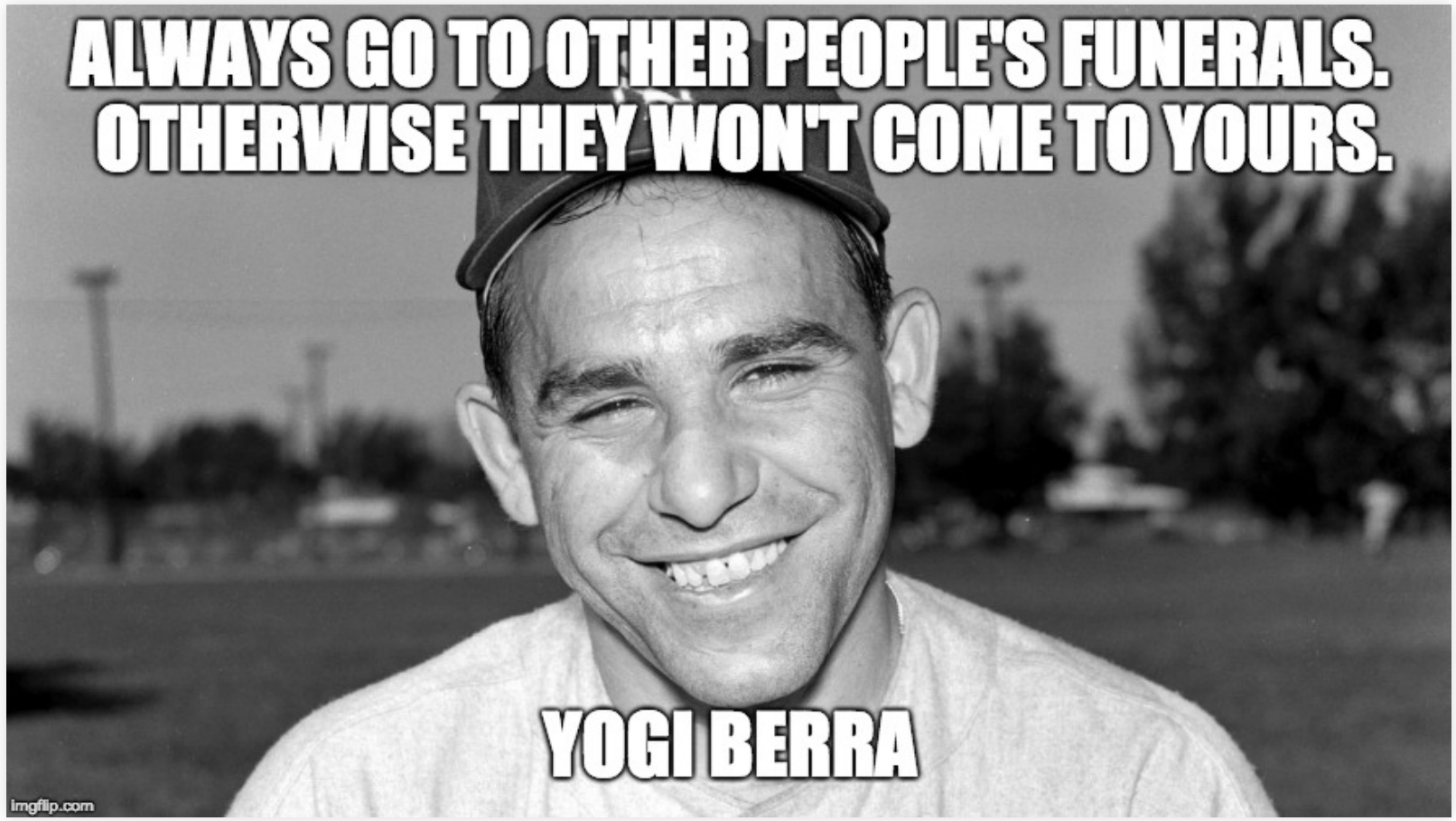


RMSE = 669

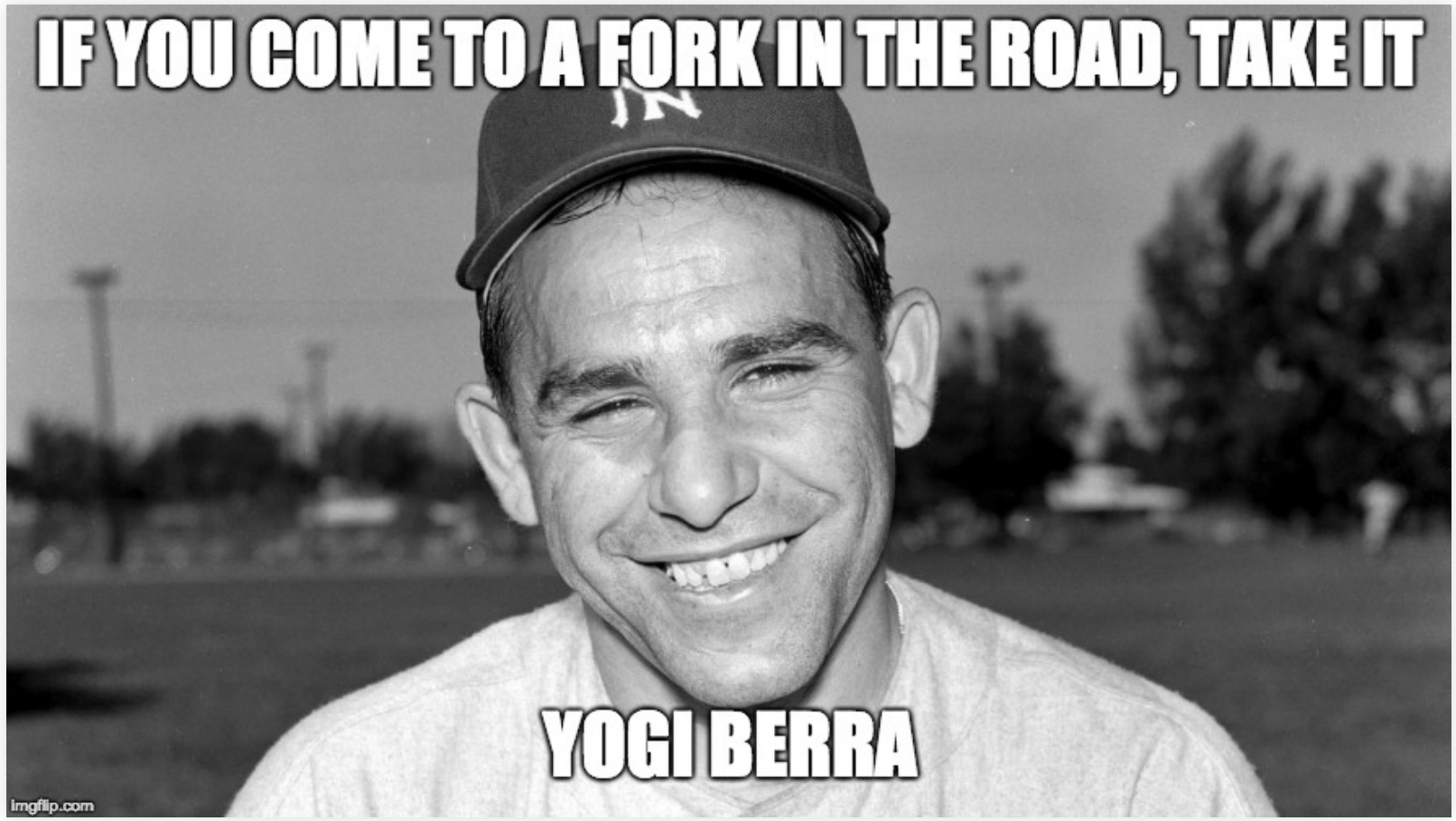
So we should pick K=2? Ask Yogi!



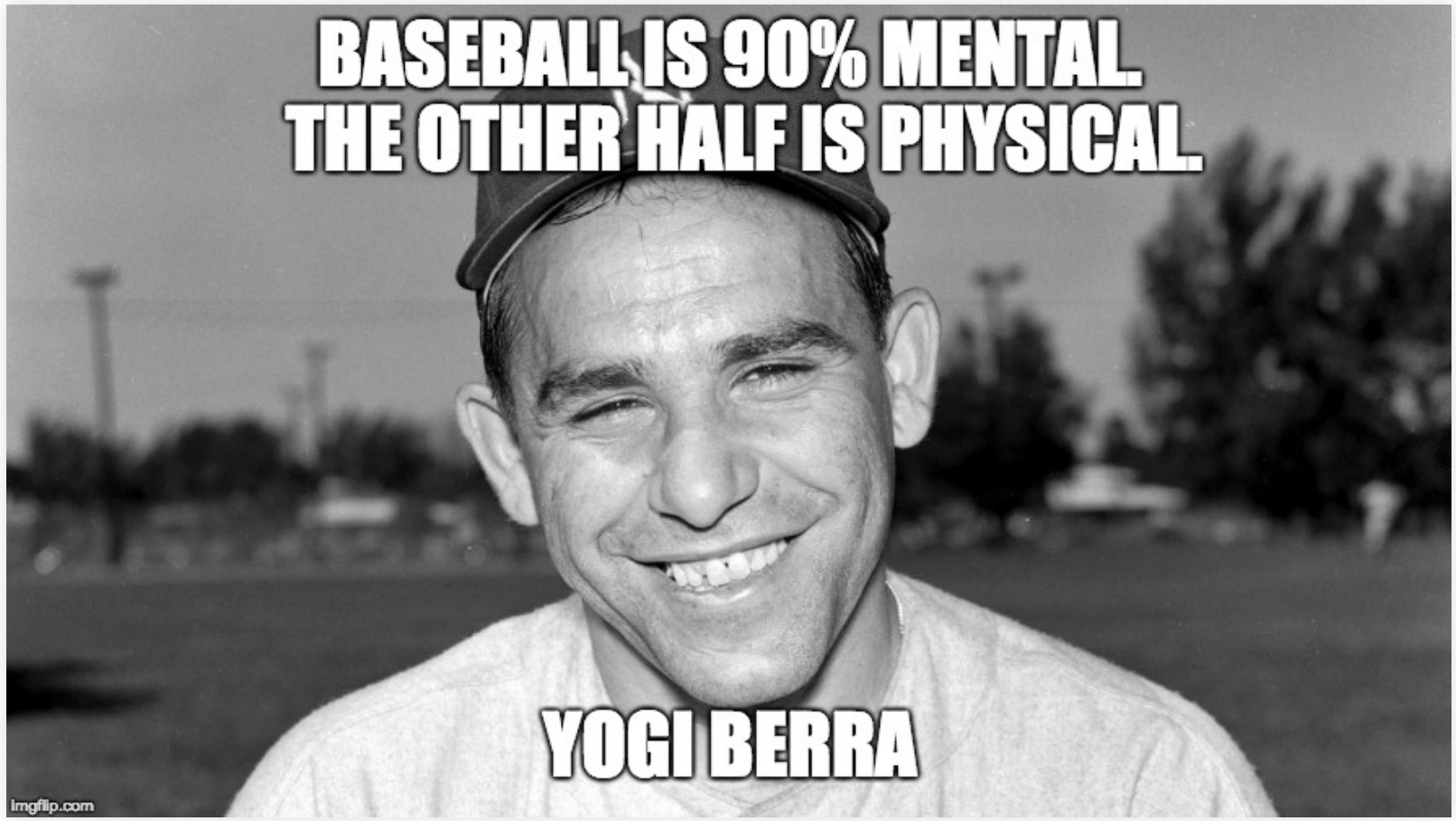
So we should pick K=2? Ask Yogi!



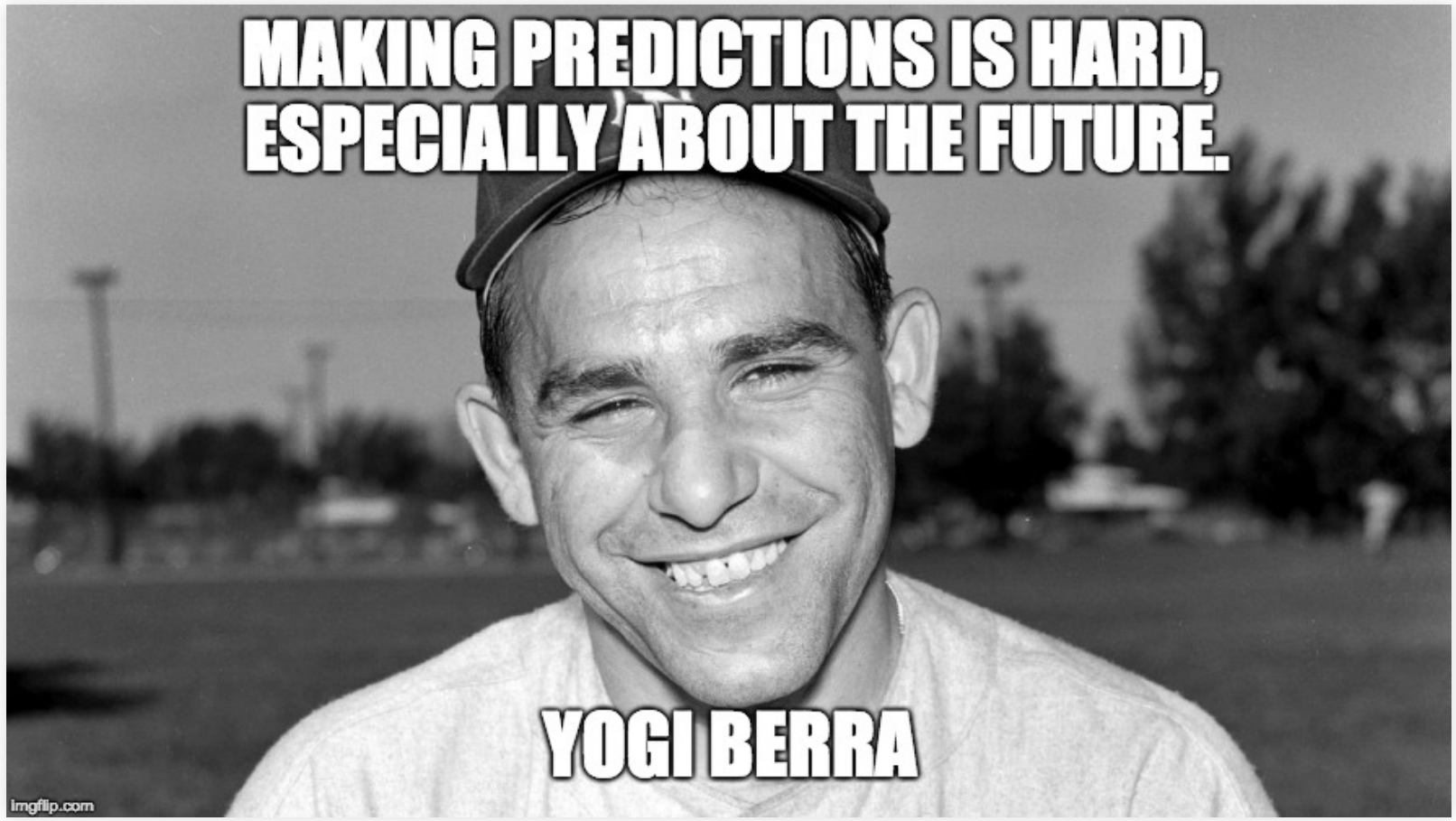
So we should pick K=2? Ask Yogi!



So we should pick K=2? Ask Yogi!



So we should pick K=2? Ask Yogi!



Out-of-sample accuracy

Make 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 in the future.

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 $i = 1, \ldots, M$, which we *did not use* to fit the model. We'll call this the "testing" data, to distinguish it from our original ("training") data.

What really matters is the model's out-of-sample root mean-squared error:

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

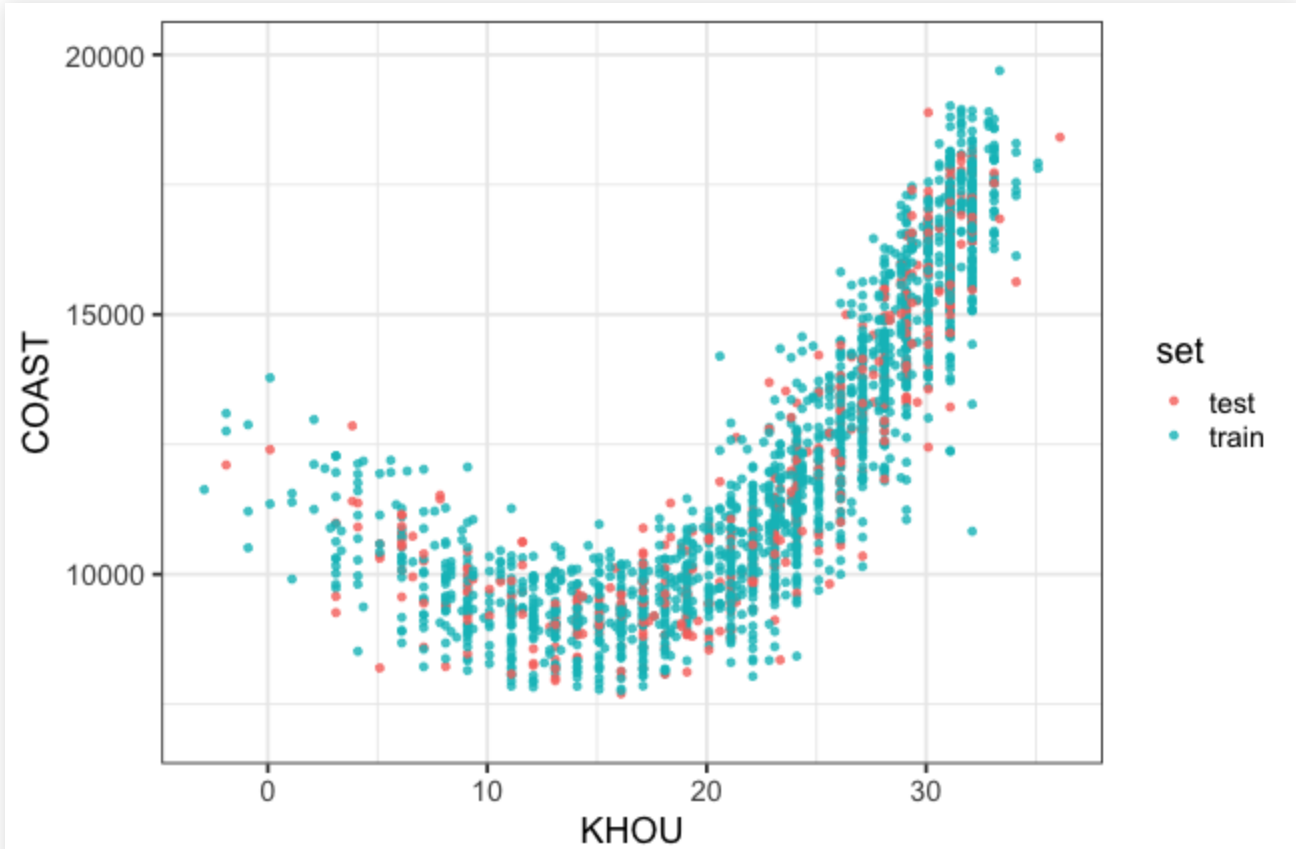
Using a train/test split

We don't have any "future data" to use to test our model. We just have our N original data points. So we have to fake it by splitting our data set D into two subsets:

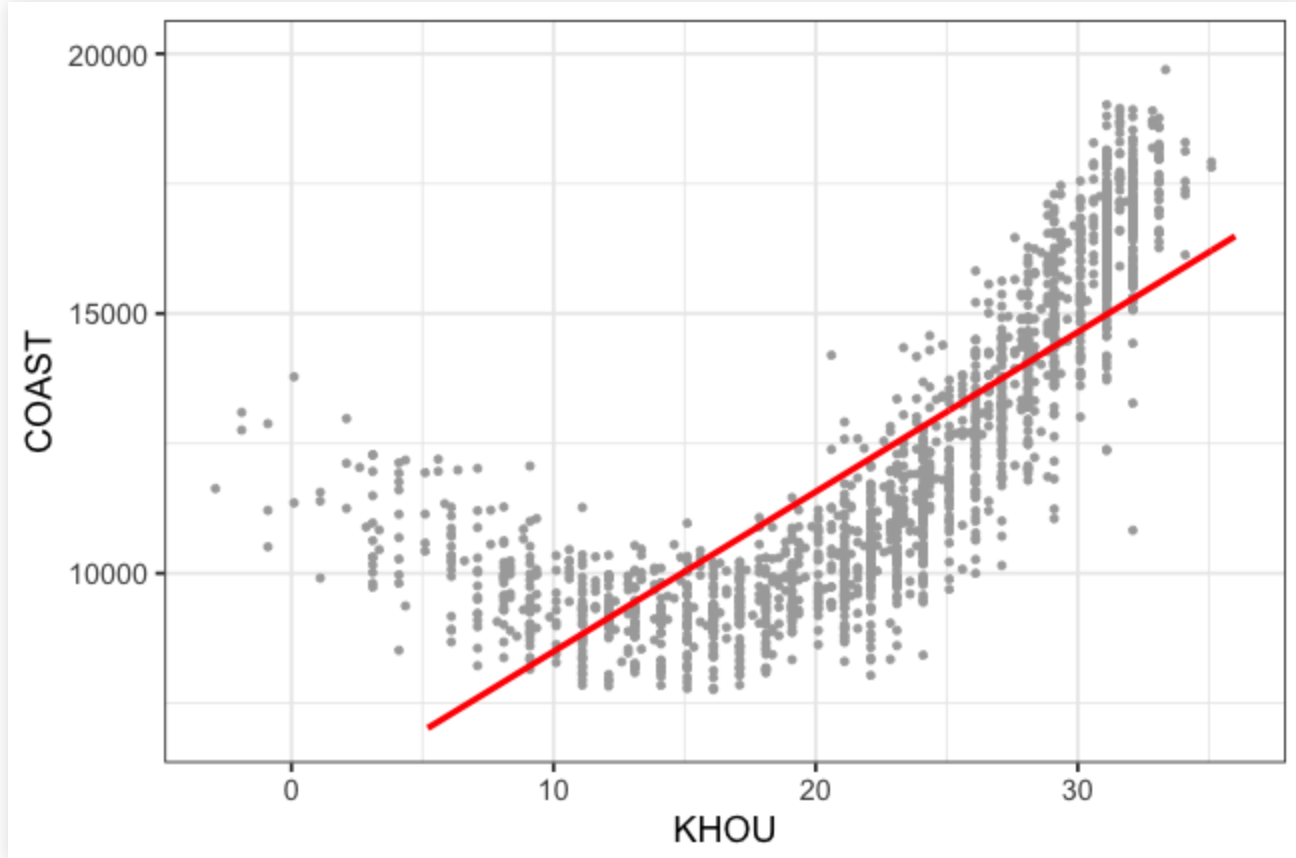
- A training set D_{in} of size $N_{in} < N$, to use for fitting the models under consideration.
- A testing set D_{out} of size N_{out}.
- $D = D_{in} \cup D_{out}$ and $N = N_{in} + N_{out}$, but $D_{in} \cap D_{out} = \emptyset$. No cheating!

We use D_{in} *only* to fit the models, and D_{out} *only* to compare the out-of-sample predictive performance of the models.

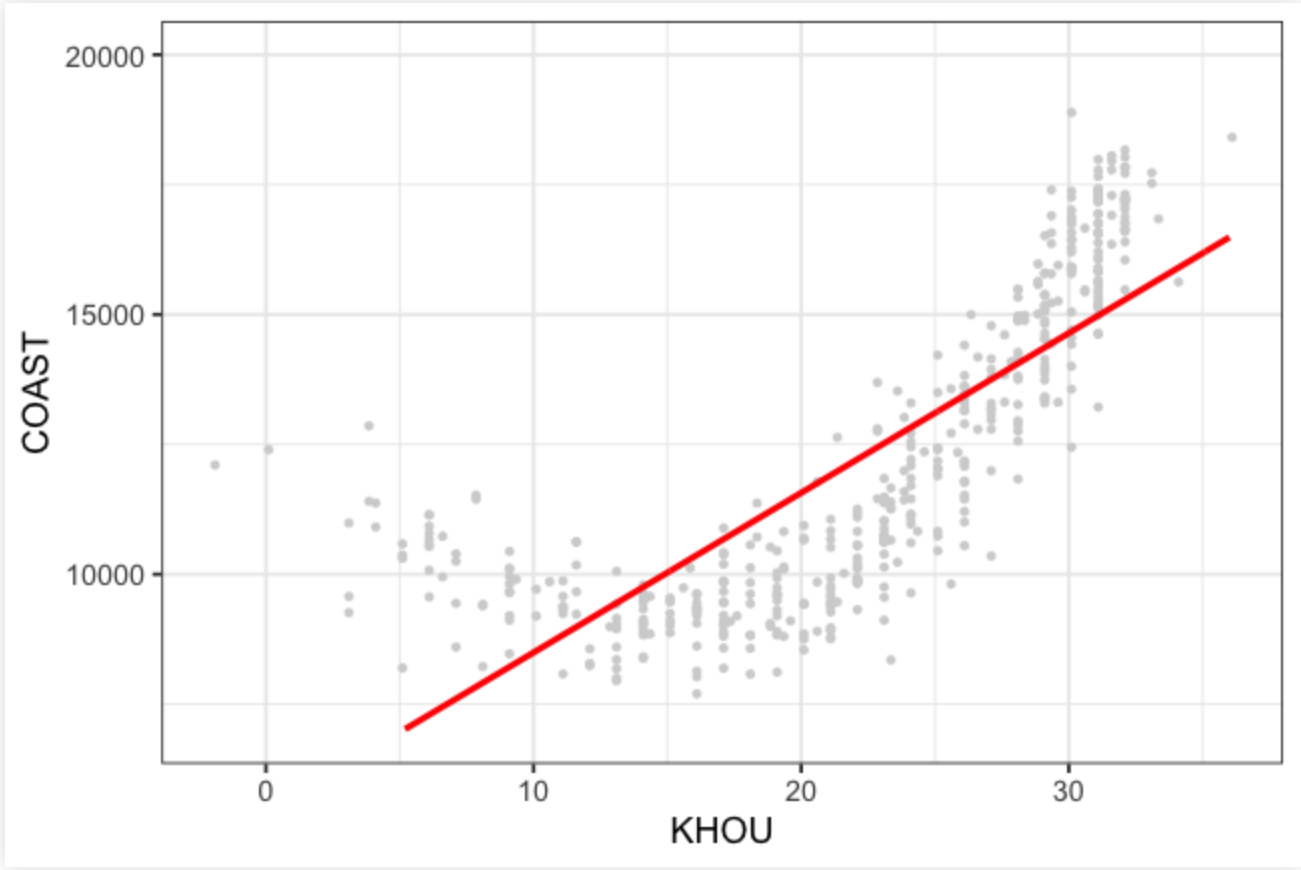
On our ERCOT data



Linear model: train

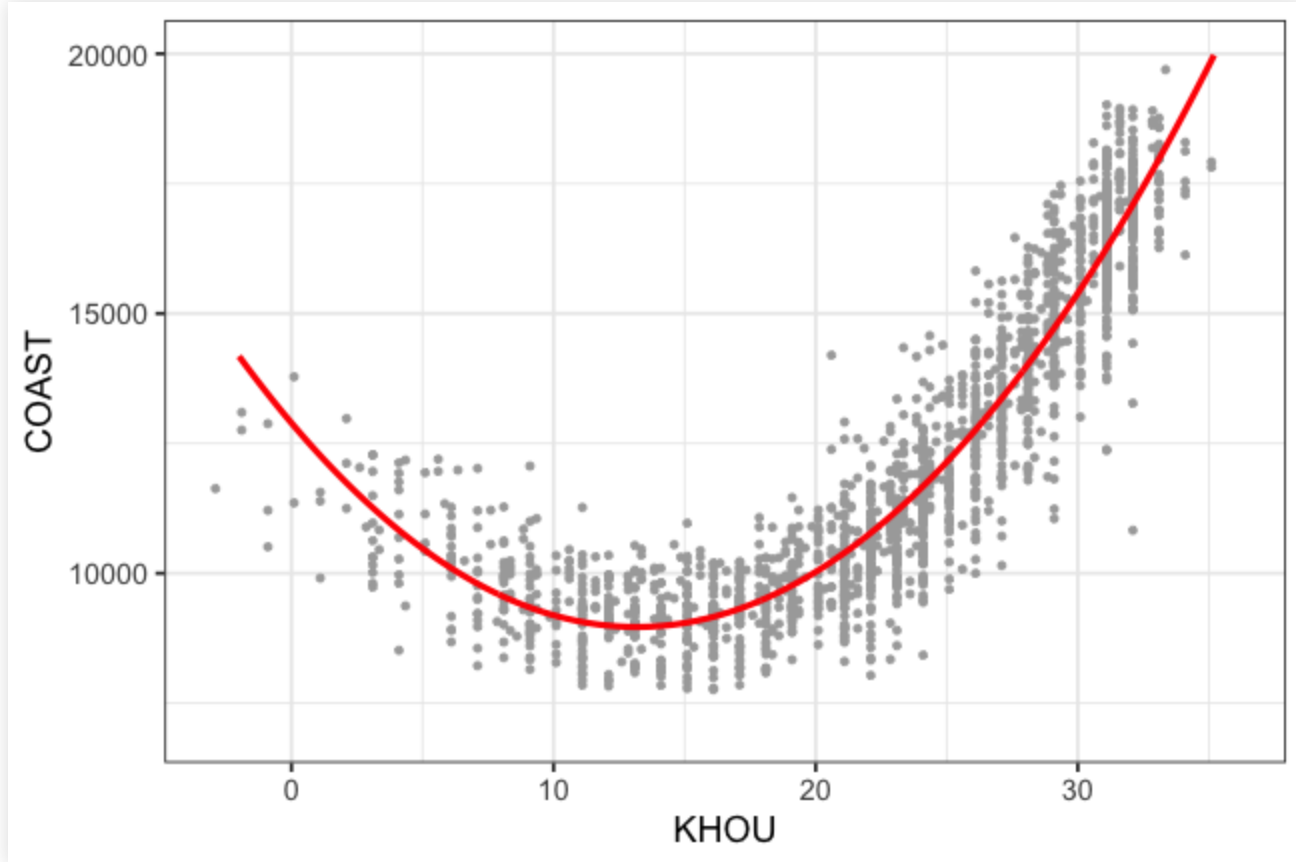


Linear model: test

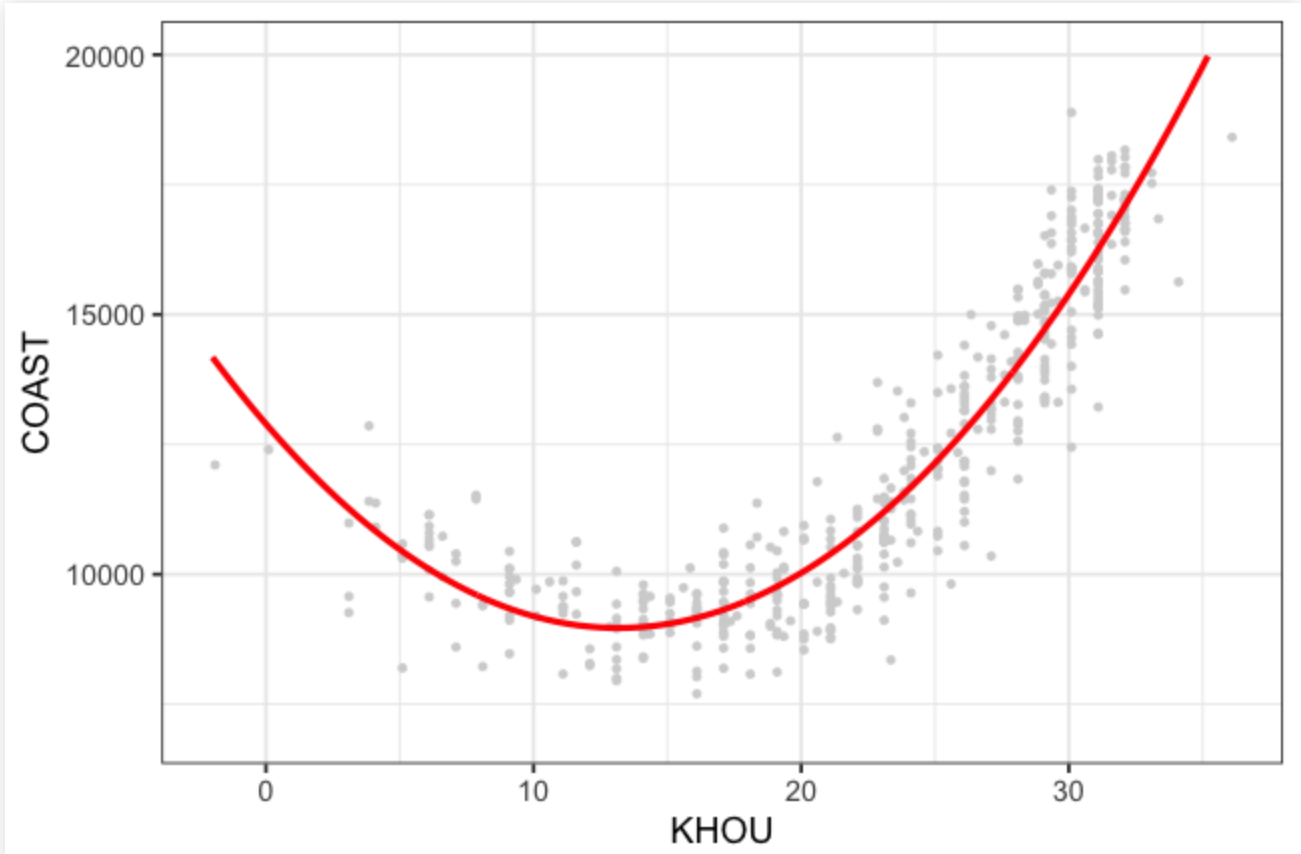


$$RMSE_{out} = 1818$$

Quadratic model: train

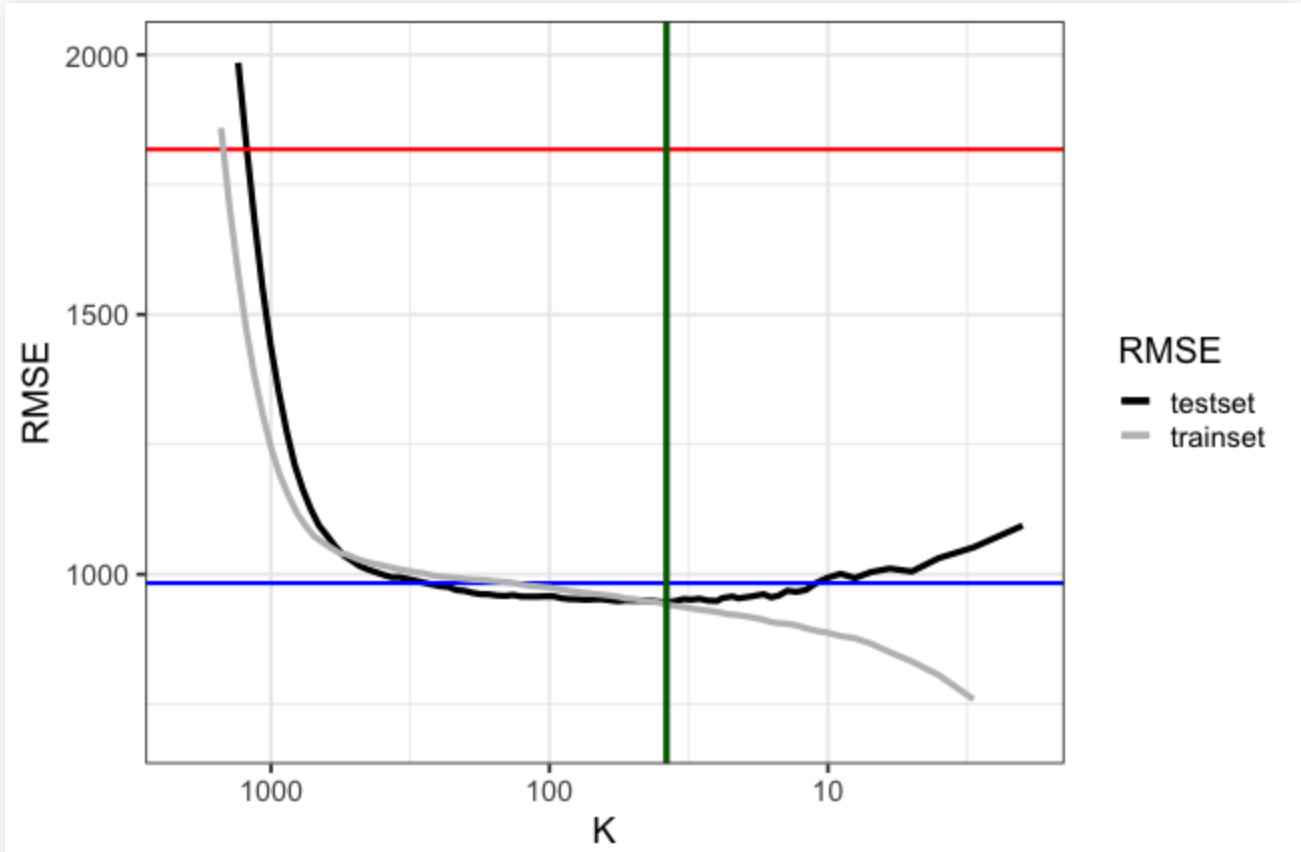


Quadratic model: test



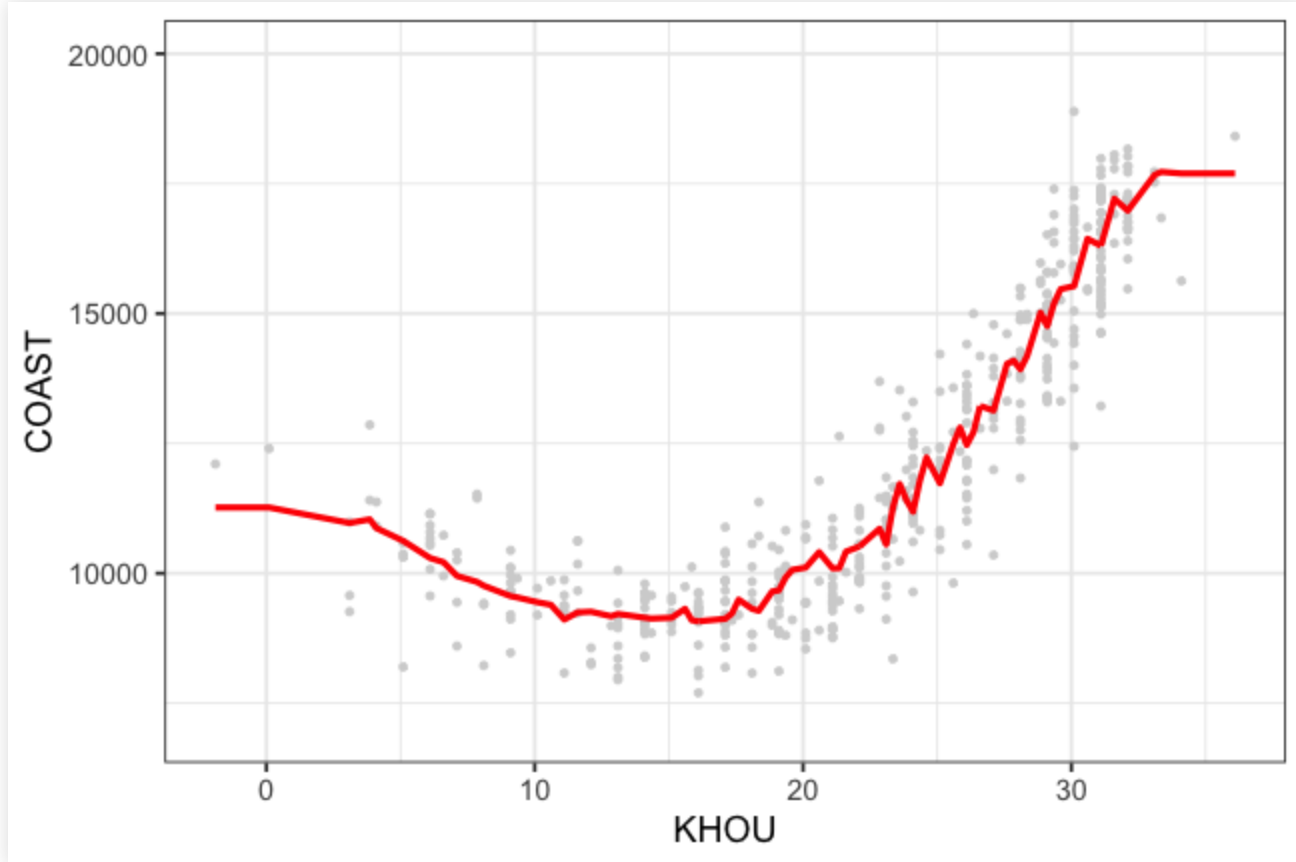
$$RMSE_{out} = 983$$

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



$$\text{RMSE}_{out} = 946$$

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 will 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.

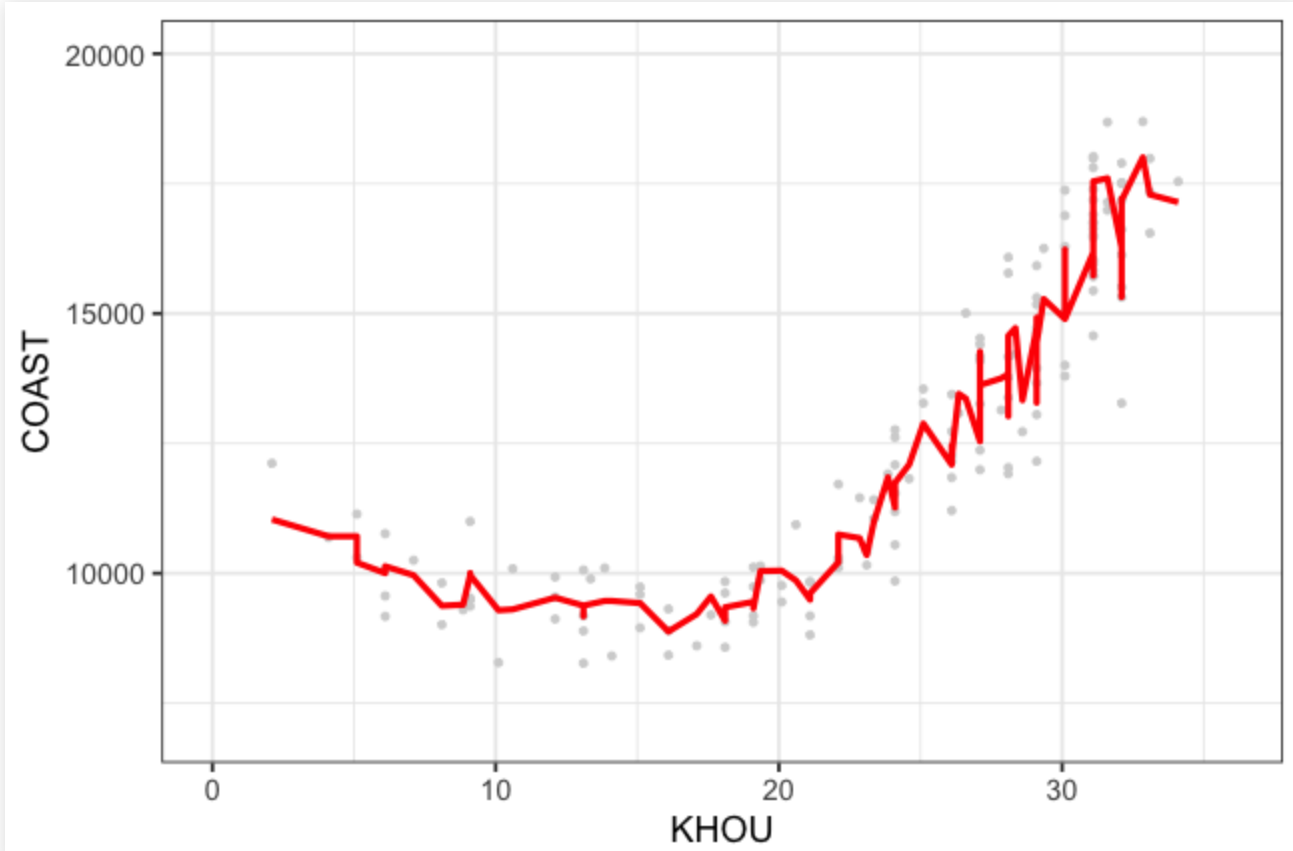
In-class exercise

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

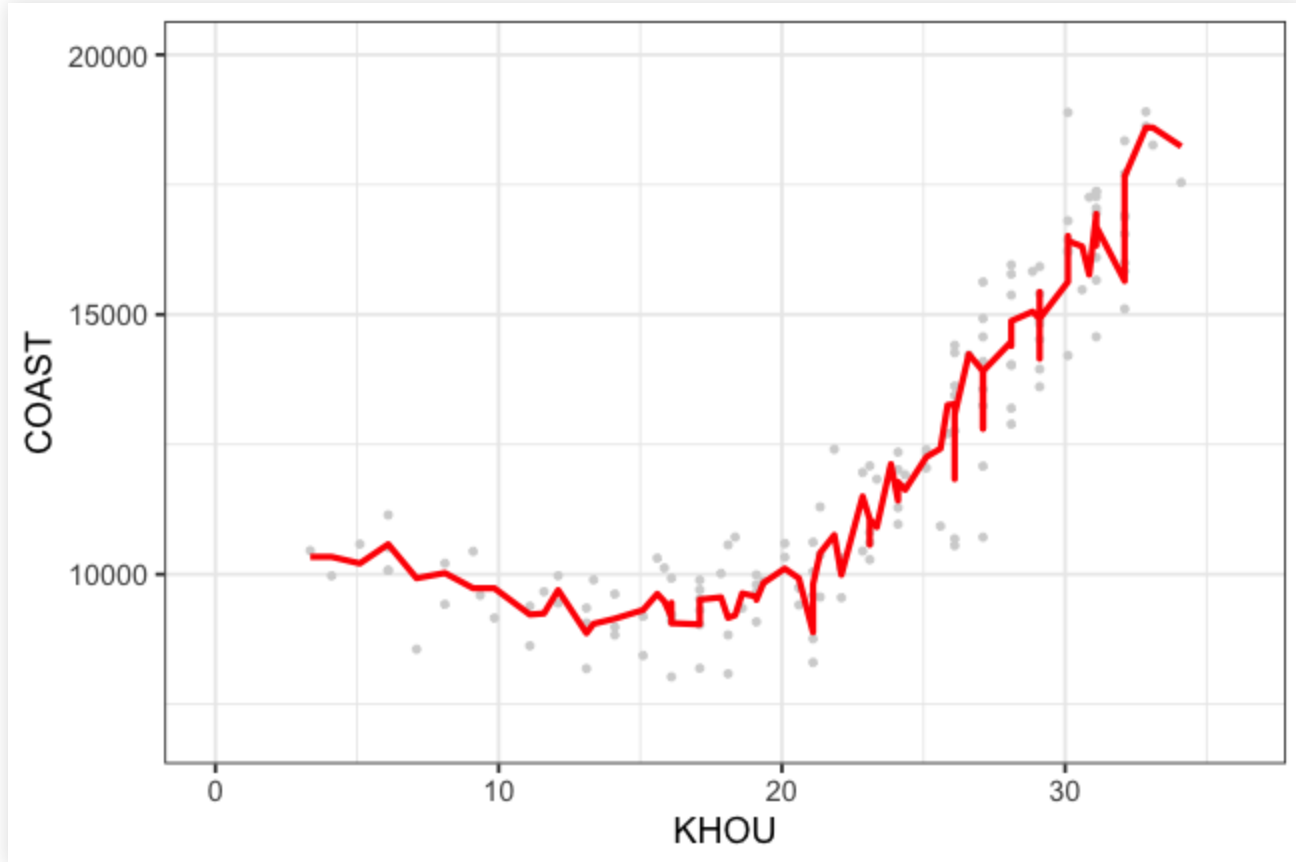
  1. Repeatedly sample sets of size N=150 from the full data set, and train a $K = 3$ model on each small training set. Plot the fit to the training set. How stable are they from sample to sample? How do they behave at the endpoints, i.e. at very low and very high temperatures?
  2. Now do the same thing, except training a $K = 75$ model on each small training set of size $N = 150$. 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.

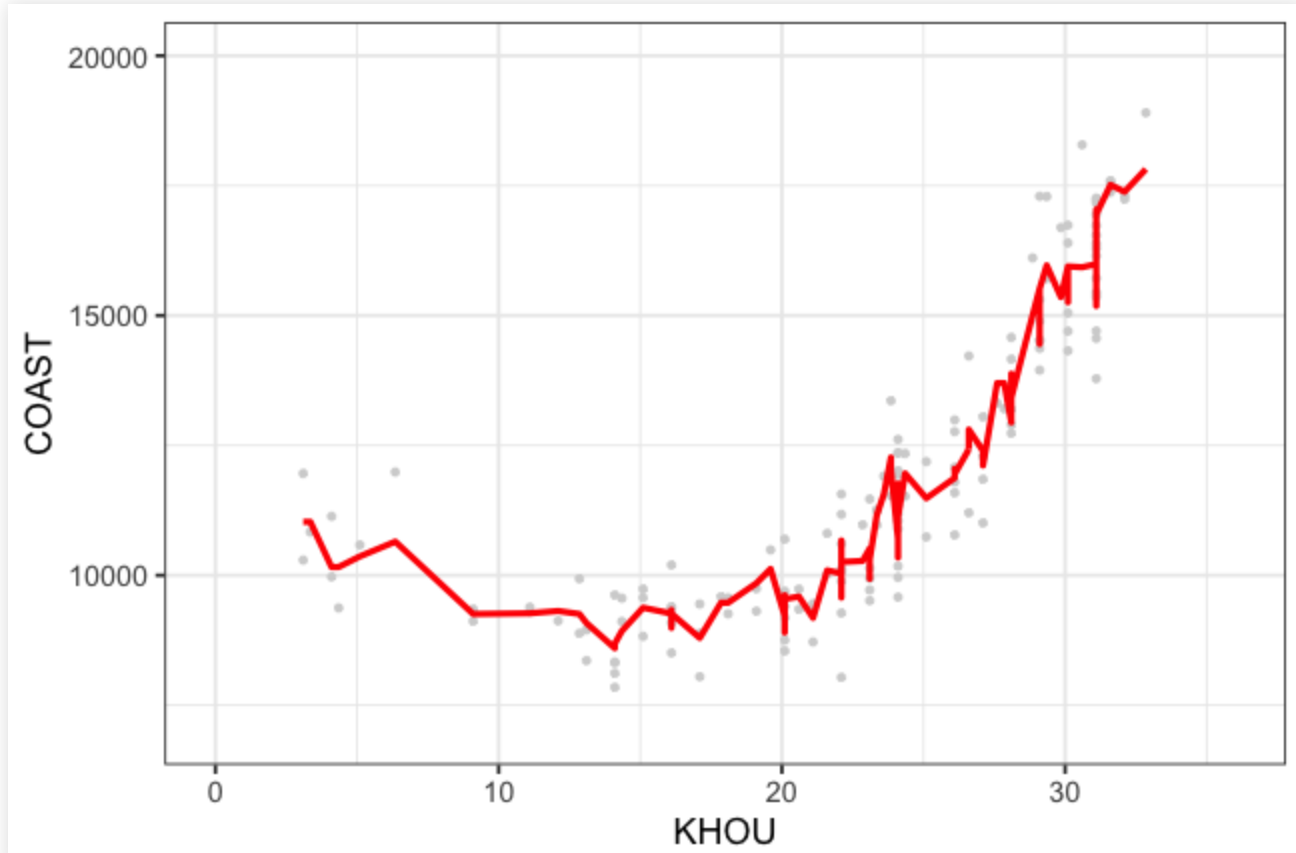
K=3: sample 1



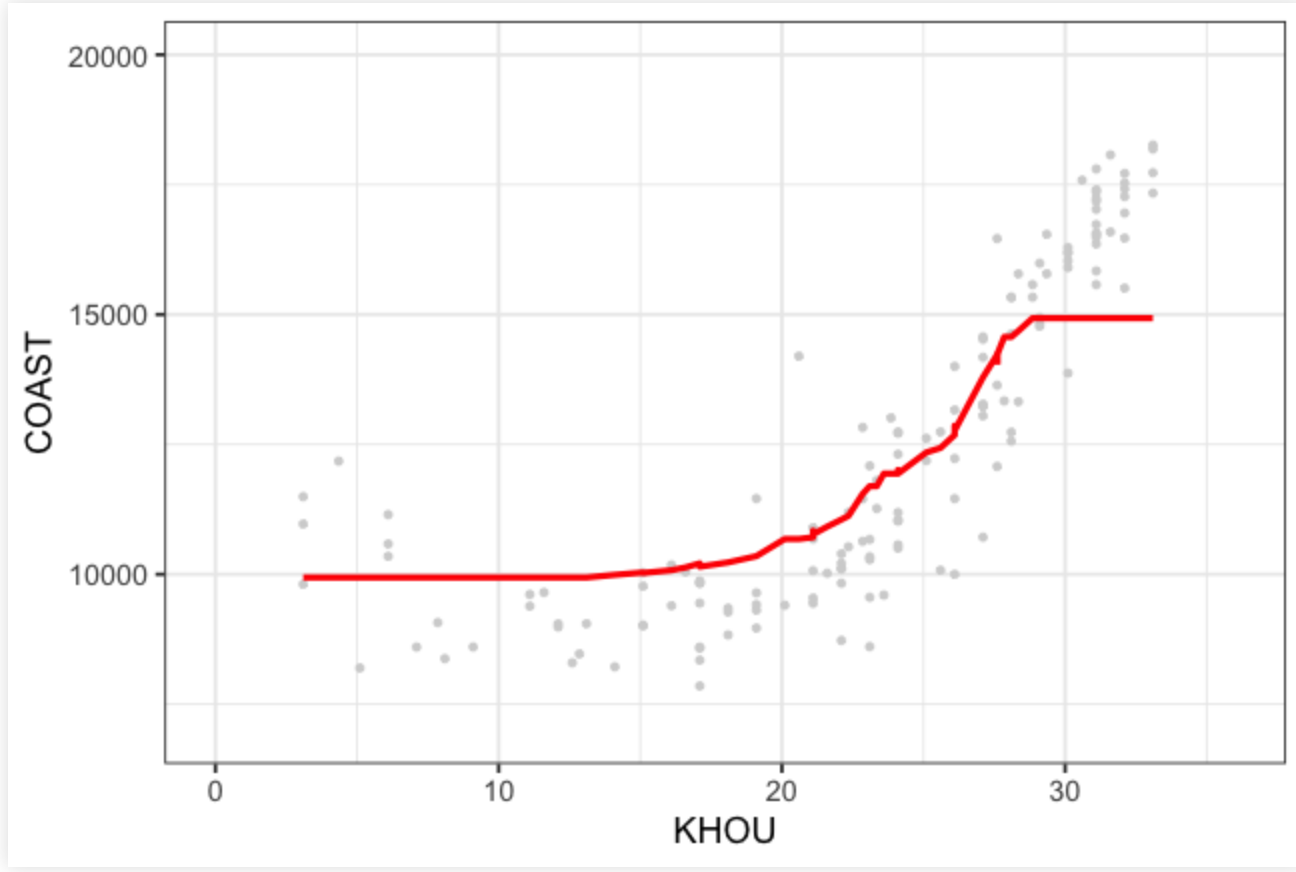
K=3: sample 2



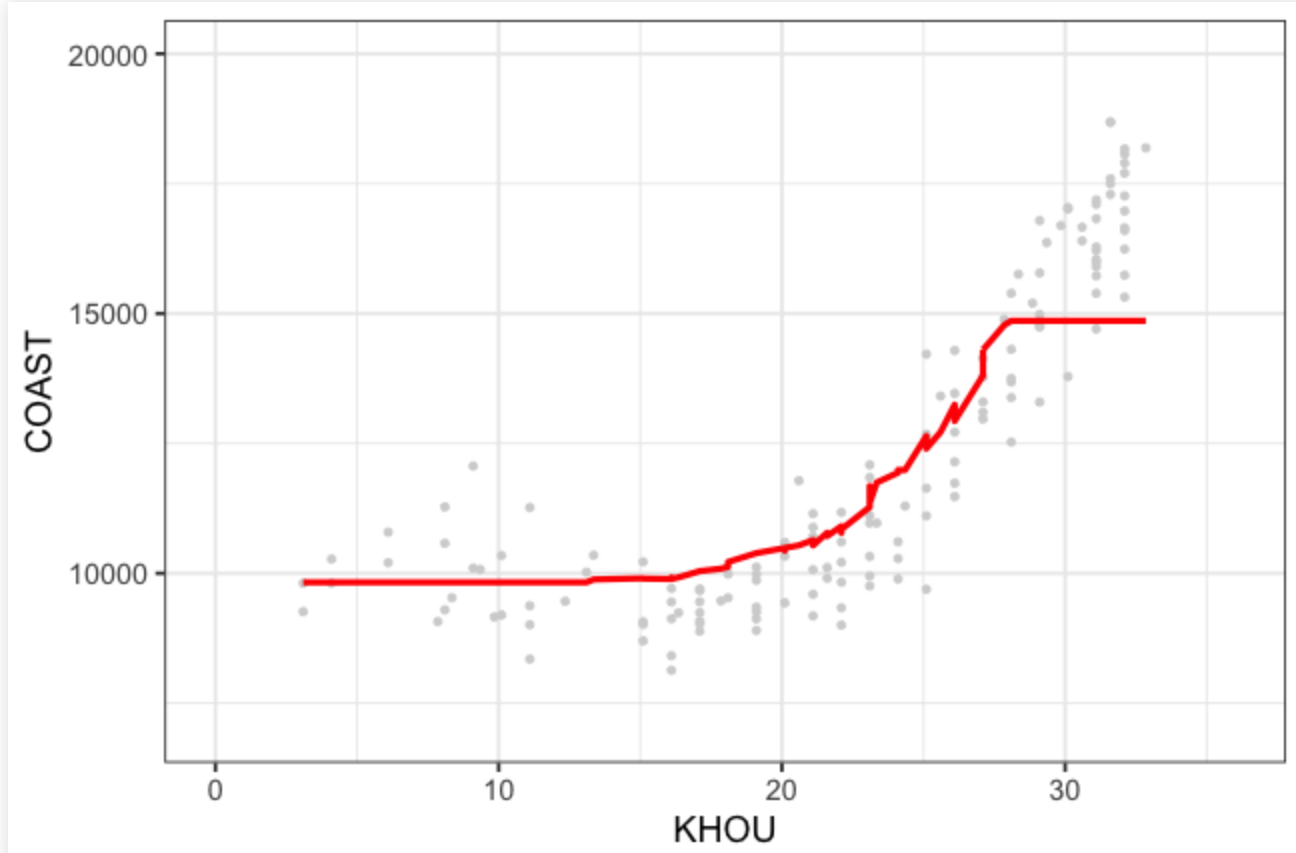
K=3: sample 3



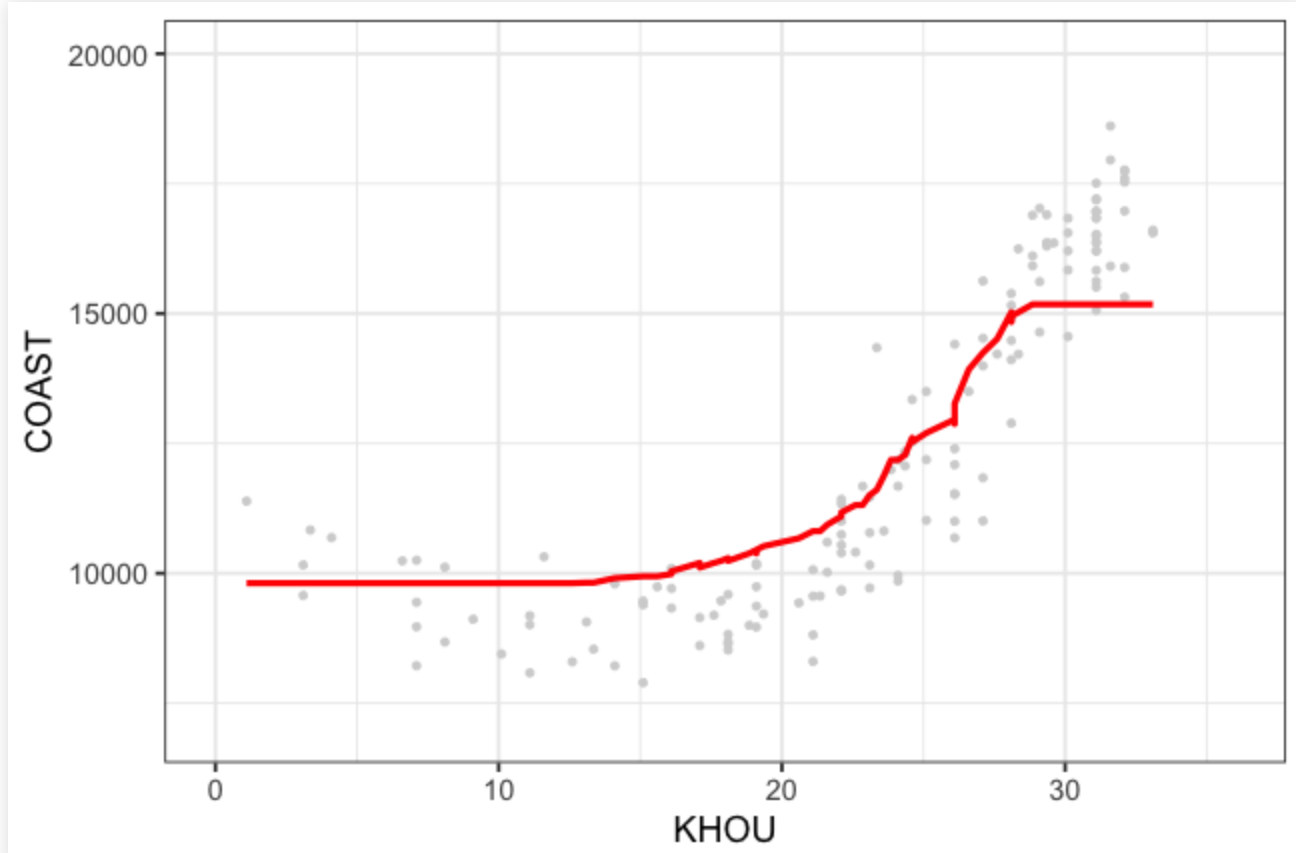
K=75: sample 1



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 arising from your training data.
- Let $x^\star$ be some future x point, and let $y^\star$ be the corresponding outcome.
- $x^\star$ is fixed a priori, but the training data and the future outcome $y^\star$ are 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

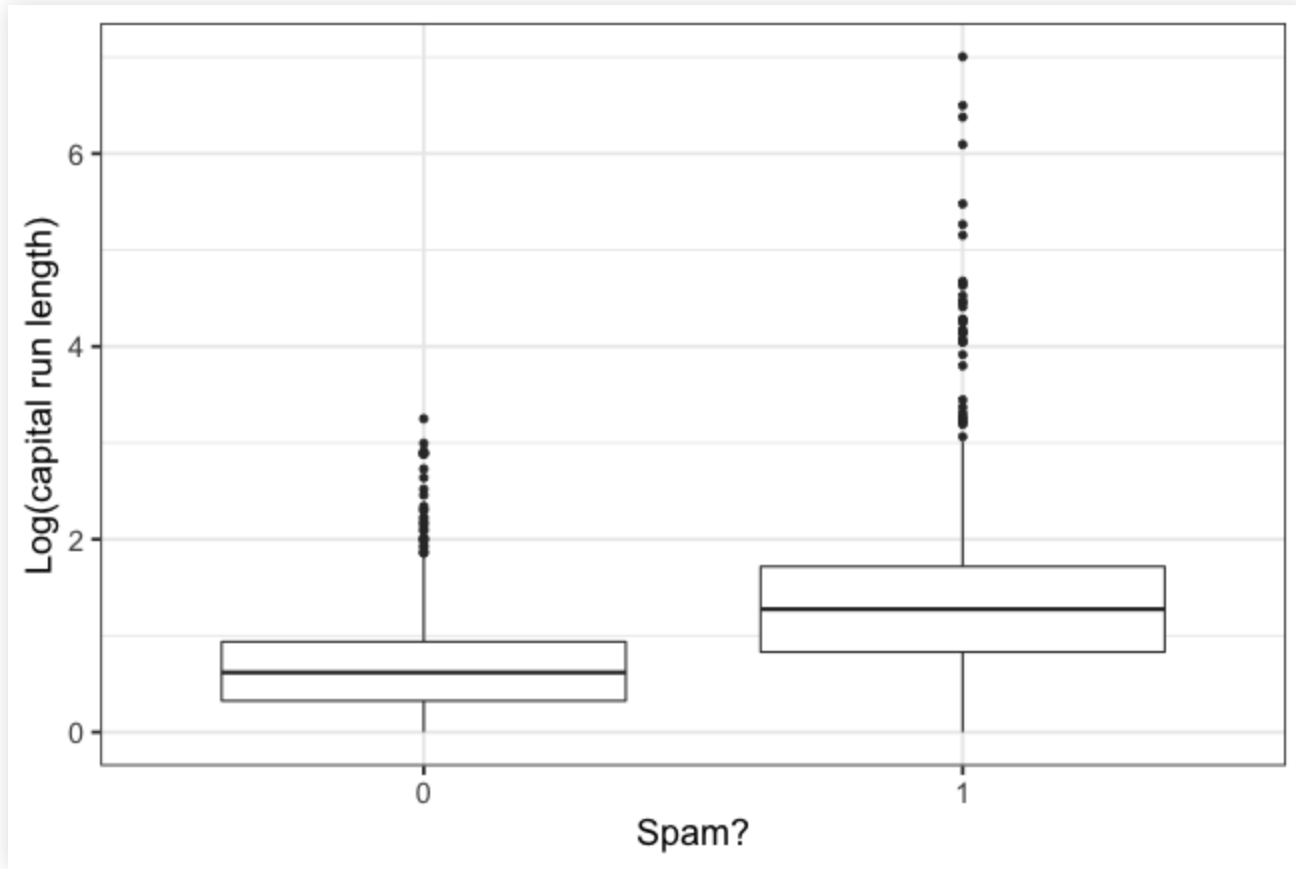
That's why it's a trade-off!

- 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!

Classification

Classification is the term used when we want to predict a target variable y that is categorical (win/lose, sick/healthy, pay/default, good/bad…). For example, the plot below shows the longest run length of capital letters for a sample of spam and non-spam e-mails:



Classification

In this context, our approach is similar, but different in some specific ways. If y is binary (0/1), then our assumed model is:

$$P(y = 1 \mid x) = f(x)$$

In general, if $y \in \{1, \ldots, K\}$, then we have a *multi-class classification* problem, and our goal is to estimate

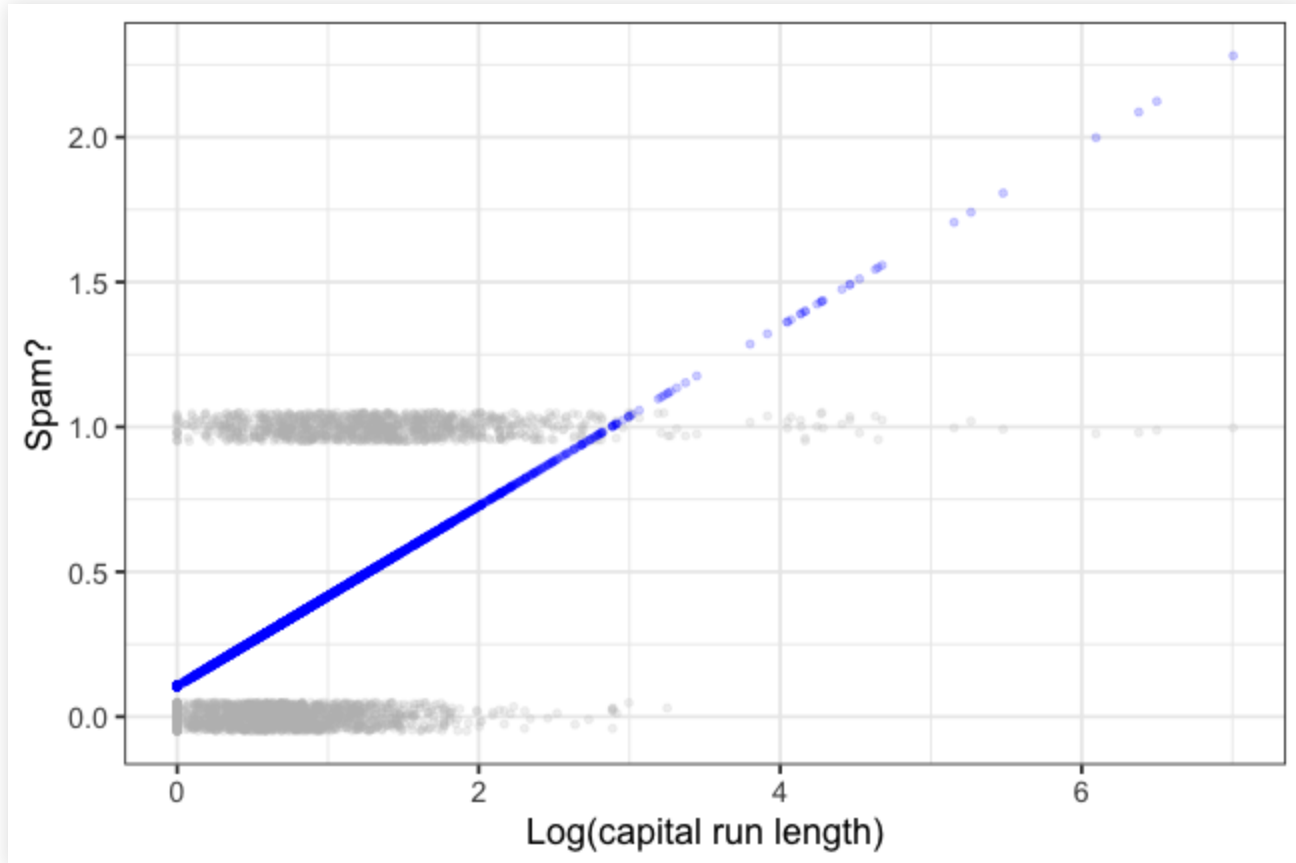
$$P(y = k \mid x) = f_k(x)$$

for each category k. Note there is an implicit constraint:

$$\sum_{k=1}^{K} f_k(x) = 1$$

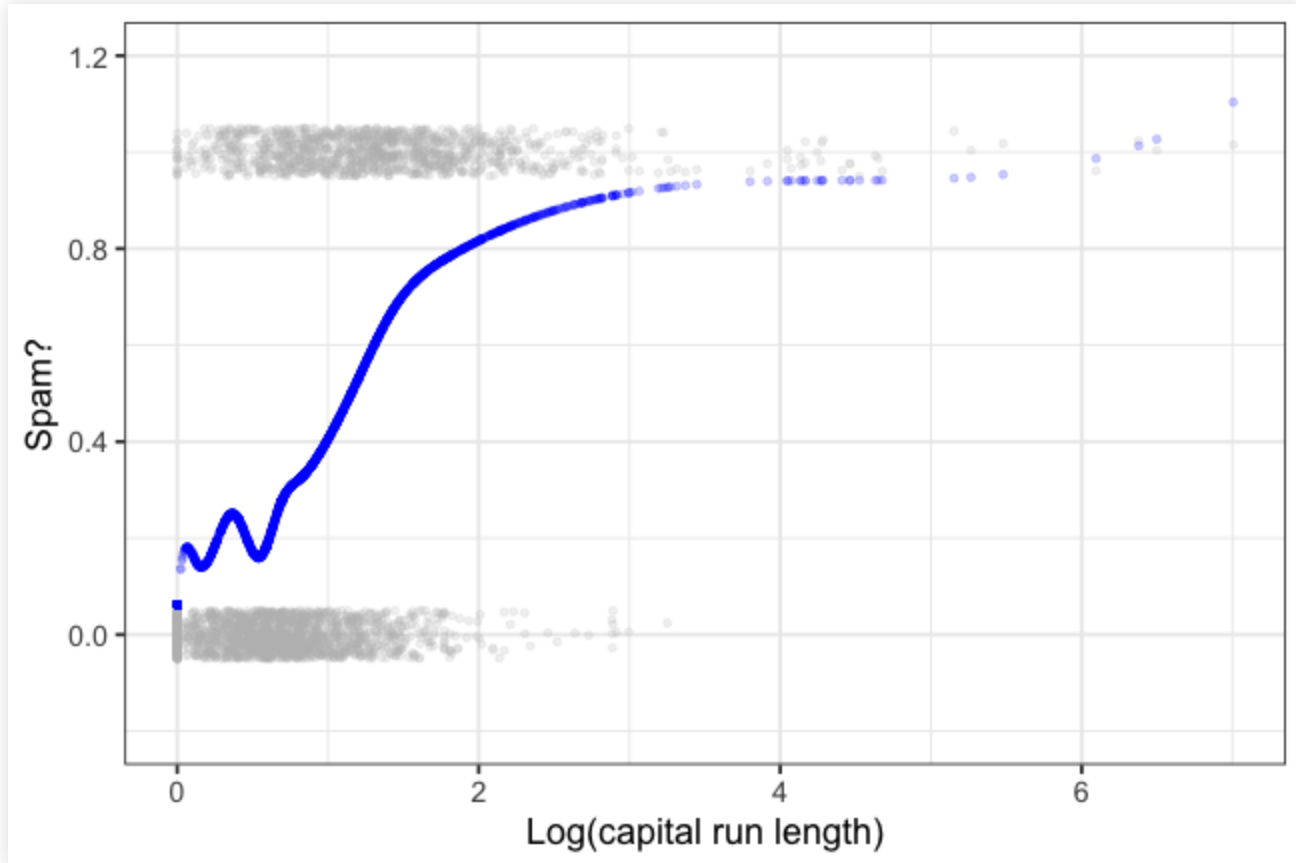
Classification

For example, in the spam classification problem, here is a linear estimate for $f(x)$. It behaves somewhat reasonably, but also has some obviously undesirable features.



Classification

Here's a very different fit:



Classification: from probabilities to predictions

In binary classification, a very natural prediction rule is to threshold the probabilities:

$$\hat{y} = \begin{cases} 1 & \text{if } f(x) \geq t \\ 0 & \text{if } f(x) < t \end{cases}$$

A natural choice is $t = 0.5$, although this might not always be appropriate.

In multi-class problems, the extension of this idea is to predict using the "highest probability" class:

$$\hat{y} = \arg_k \max f_k(x) \, .$$

Classification: measuring accuracy

What differs from numerical prediction is our *measure of accuracy*: we are no longer dealing with a numerical outcome variable.

One common approach is to measure the accuracy of a model's predictions using the raw *error rate* on the training sample:

$$ER = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

Here $I(\cdot)$ is the indicator function, taking the value 1 if its argument is true, and 0 otherwise.

Classification: out-of-sample error rate

Just like with numerical prediction, we can define an out-of-sample version of the error rate:

$$ER_{out} = \frac{1}{m} \sum_{i=1}^{m} I(y_i^\star \neq \hat{y}_i^\star)$$

where $(x_1^\star, y_1^\star), \ldots, (x_m^\star, y_m^\star)$ is a collection of m "future" (out-of-sample) data points that weren't used to train the model.

As before, the practical way to estimate this quantity is to use a train/test split of the original data set.