Prediction!

Contents

Ok, so I think I start off a bit loose to get people comfortable and bring out some playing cards.

- I'll point to someone and ask them to guess the suit of the next card I show.
- Reshuffle and repeat giving them five cards/guesses.
- Then I'll repeat with another three-four students.
- # correctly guessed will be noted somewhere
- They've done a similar thing before but now an emphasis on guessing the next # of correct.

What's the point? How can I best predict the number of card suits the next person will get right?

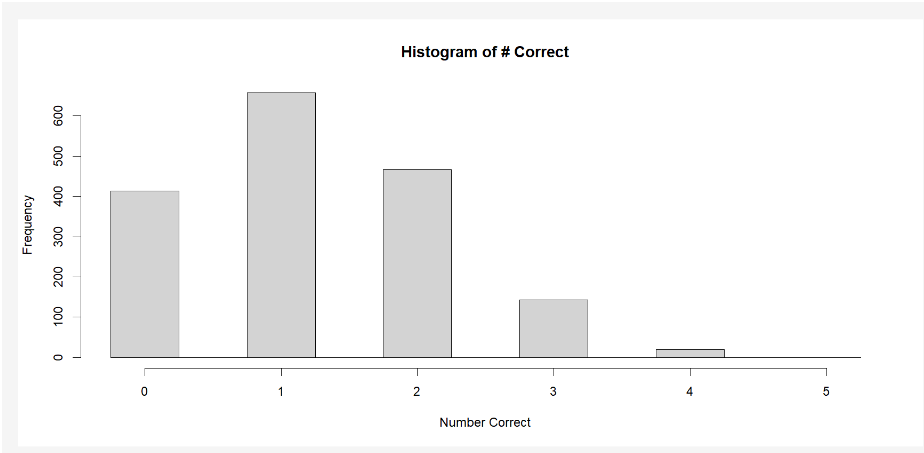
Prediction

Goal: Predict a new value of a variable

- Ex: Another student will be guessing. Define $Y = \#$ of card suits guessed correctly from the five. What should we guess/predict for the next value of Y?

App

- Chat with them.
- Lead them to talk about a sample.
- Simulate values of Y using an app (in the repo, use `shiny::runGitHub("caryAcademy", username = "jbpost2", subdir = "CardSim", ref = "main")`. app
- Lead them to ideas of using something like the sample mean or median as the predicted value.
- Why something like the sample mean or sample median? What are we really trying to do? Find a value that is 'close' to the most values, i.e. something in the center being the most logical thing to do.



Loss function

Let's assume we have a sample of n people that each guessed five cards. Call these values $y_1, y_2, \ldots, y_n$.

Need: A way to quantify how well our prediction is doing... Suppose there is some best prediction, call it c. How do we measure the quality of c?

- Using the idea that we want something 'close' to all points, we find a way to compare each point to our prediction.
- Think about things like:

$$y_1 - c, (y_1 - c)^2, |y_1 - c|$$

$$\sum_{i=1}^{n}(y_i - c), \sum_{i=1}^{n}(y_i - c)^2, \sum_{i=1}^{n}|y_i - c|$$

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - c), \frac{1}{n}\sum_{i=1}^{n}(y_i - c)^2, \frac{1}{n}\sum_{i=1}^{n}|y_i - c|$$

- Quick app to look at how the measures work in the app. app
- In the end an objective function (mean squared error here) must be created to minimize that uses a 'Loss function', and we'll talk about why we'll use the common squared error loss:

$$g(y_1, ..., y_n) = \frac{1}{n}\sum_{i=1}^{n}L(y_i, c) = \frac{1}{n}\sum_{i=1}^{n}(y_i - c)^2$$

Can we choose an 'optimal' value for c to minimize this function? Calculus to the rescue!

Steps to minimize a function with respect to c:

1. Take the derivative with respect to c
2. Set the derivative equal to 0
3. Solve for c to obtain the potential maximum or minimum
4. Check to see if you have a maximum or minimum (or neither)

Answer comes out to be $\bar{y}$ as the minimizer.

Big wrap: This means that the sample mean is the best prediction when using squared error loss (root mean square error).

Using a Population Distribution

Rather than using sample data, suppose we think about the theoretical distribution for $Y = \#$ of card suits guessed correctly from the five. What might we use here? What assumptions do we need to make this distribution reasonable?

- $Y \sim Bin(5, 0.25)$ assuming we have independent and identical trials
- This gives

$$p_Y(y) = P(Y = y) = \binom{n}{y} p^y (1-p)^{n-y} = \binom{5}{y} 0.25^y 0.75^{5-y}$$

  for $y = 0, 1, 2, ..., n$ or $y = 0, 1, 2, 3, 4, 5$

Is there an optimal value c for the **expected value** of the loss function?

That is, can we minimize (as a function of c) $E\left[(Y-c)^2\right]$?

- I think I'll start them out on this one as theory leads to more difficult ideas and I'm not sure if you did general expected values.
- In the end though we end up with np or 1.25.
- I'll show/discuss that this works generally for any distribution $p_Y(y)$ that has a mean
- Discuss the relationship with this and a sample (1/n weight for each point vs $p_Y(y)$ weight for each.)
- **Big idea:** This implies that μ is the best predictor to use if you are considering minimizing the expected squared error loss.

HW for after day 1:

- Give them a data set and have them find the mean in R and note that the prediction they would use is that.
- Have them play minesweeper and record the data appropriately. Ask them to produce a best guess for their # of overall bombs.

- Give them a partial derivative question to practice on.

This would be the 2nd (short day) material.

- Recap the big idea of prediction:
  - Need to quantify how well we are doing (squared error loss and MSE)
  - Sample mean is optimal if we have a sample
  - Given a theoretical distribution, expected squared error loss is optimized at the mean of the distribution
- Introduce next material with minesweeper, because it is now browser based, nostalgia on my part, and it seems somewhat fun https://minesweeper.online/game/938135731
  - Each student will be assigned a certain number of mines for the board (15x40).
  - They'll click on the first square just below the smiley face. They'll continue to click down one block at a time until they hit a bomb.
  - They'll record in a shared spreadsheet their number of blocks down the first bomb appears.
  - Each person should play 10 games and put their data in.
- Now we'll discuss how we could predict the number of blocks until the first bomb as a function of the number of bombs.
- We'll read in the data to R and do some plotting (I'm not sure what the relationship will be exactly but I'd guess not super linear).

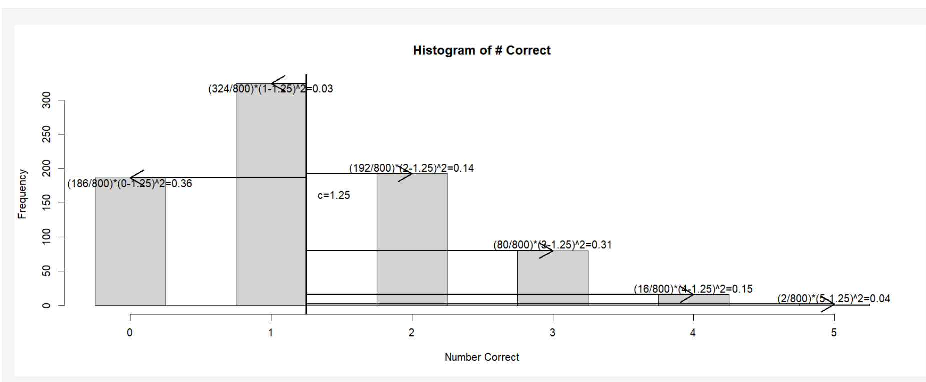
Relating Explanatory Variables in Prediction

Y is a random variable and we'll consider the x values fixed (we'll denote this as $Y|x$). We hope to learn about the relationship between Y and x.

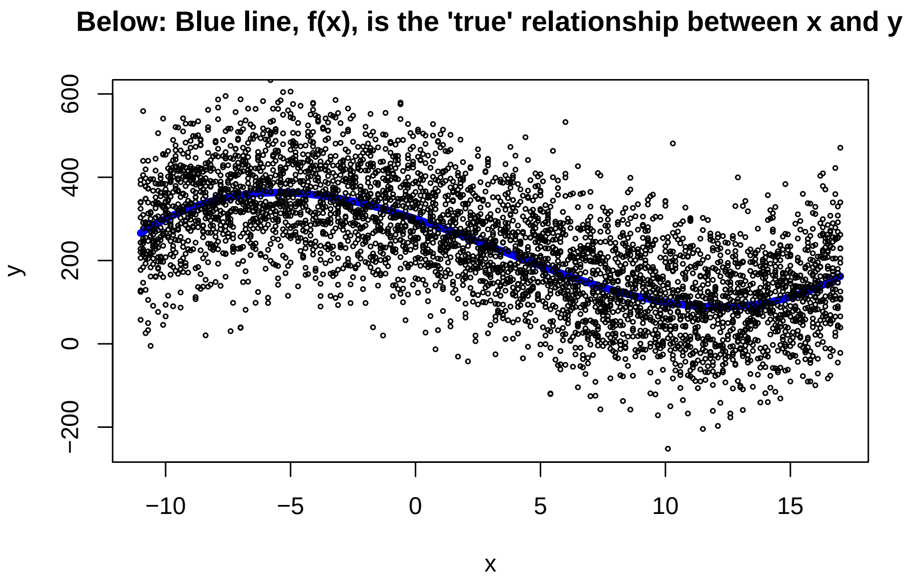
When we considered just Y by itself and used squared error loss, we know that $E(Y) = \mu$ minimizes

$$E\left[(Y-c)^2\right]$$

as a function of c. Given data, we used $\hat{\mu} = \bar{y}$ as our prediction.



Harder (and more interesting) problem is to consider predicting a (response) variable Y as a function of an explanatory variable x.



Now that we have an x, $E(Y|x)$ will minimize

$$E\left[(Y-c)^2|x\right]$$

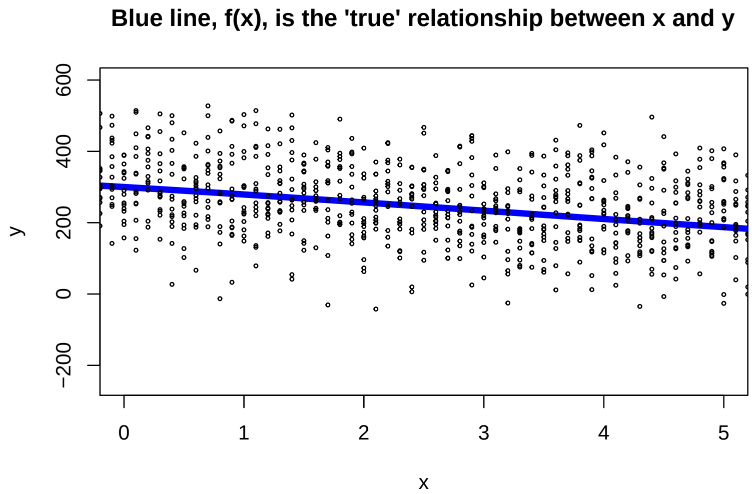
We can call this true unknown value $E(Y|x) = f(x)$. That is, the average value of Y will now be considered as a function of x.

Given observed Y's and x's, we can estimate this function as $\hat{f}(x)$ (think $\bar{y}$ from before). This $\hat{f}(x)$ will minimize

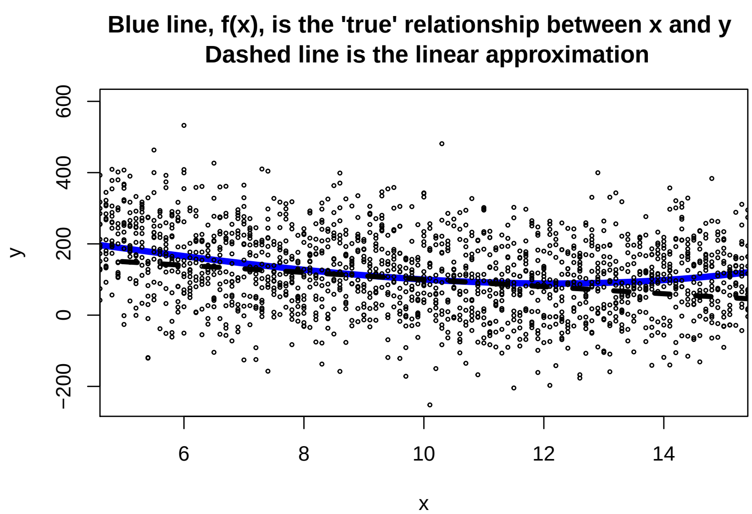
$$g(y_1, ..., y_n|x_1, ..., x_n) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}(x_i)) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

Approximating $f(x)$

Although the true relationship is most certainly nonlinear, we may be ok approximating the relationship linearly. For example, consider the same plot as above but between 0 and 5 only:



That's pretty linear. Consider plot between 5 and 15:



Line still does a reasonable job and is often used as a basic approximation.

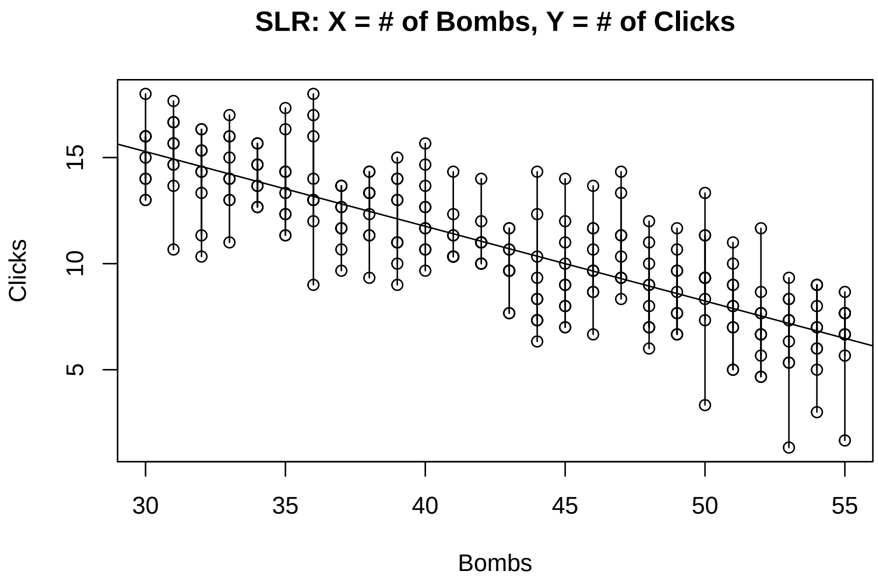
Linear Regression Model

The (fitted) linear regression model uses $\hat{f}(x) = \hat{\beta}_0 + \hat{\beta}_1 x$. This means we want to find the optimal values of $\hat{\beta}_0$ and $\hat{\beta}_1$ from:

$$g(y_1, ..., y_n | x_1, ..., x_n) = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$$

This equation is often called the 'sum of squared errors (or residuals)' or the 'residual sum of squares'. The model for the data, $E(Y|x) = f(x) = \beta_0 + \beta_1 x$ is called the Simple Linear Regression (SLR) model.

I'll have code at the ready to update and rerender this plot using their data from minesweeper.



Calculus allows us to find the 'least squares' estimators, $\hat{\beta}_0$ and $\hat{\beta}_1$ in a nice closed-form!

Do they know partial derivatives? I'm not sure. I think we'll be running low on time here anyway, so maybe I'll just talk about the idea of how to get them, set up the equations and then just give the answers.

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \bar{x}\hat{\beta}_1$$

Then we'll jump into R and find the values for the minesweeper data and use it to predict by "hand" (plugging it in manually in R).

Ok, day 3 here. Get into R and do some model fitting and predicting. Start with a quick recap here

Fitting a Linear Regression Model in R

Recap: Our goal is to predict a value of Y while including an explanatory variable x. We are assuming we have a sample of (x_i, y_i) pairs, $i = 1, ..., n$.

The Simple Linear Regression (SLR) model can be used:

$$\hat{f}(x_i) = \hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

where

- y_i is our response for the i^{th} observation
- x_i is the value of our explanatory variable for the i^{th} observation
- β_0 is the y intercept
- β_1 is the slope

The best model to use if we consider squared error loss has

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \bar{x}\hat{\beta}_1$$

called the 'least squares estimates'.

Data Intro

This dataset contains information about used motorcycles and their cost.

From the information page: This data can be used for a lot of purposes such as price prediction to exemplify the use of linear regression in Machine Learning. The columns in the given dataset are as follows:

- name
- selling price
- year
- seller type
- owner
- km driven
- ex showroom price

The data are available to download from this URL:
https://www4.stat.ncsu.edu/~online/datasets/bikeDetails.csv

I think the exploration and modeling would be better to do live and ask them for input as we go through it. I'll put some stuff here and then we can talk about it.

Read in Data and Explore!

```
library(tidyverse)
bikeData <- read_csv("https://www4.stat.ncsu.edu/~online/datasets/bikeDetails.csv")
bikeData <- bikeData %>% tidyr::drop_na()
select(bikeData, selling_price, year, km_driven, ex_showroom_price, name, everything())
```

```
## # A tibble: 626 x 7
##    selling_price  year km_driven ex_showroom_price name        seller_type owner
##            <dbl> <dbl>     <dbl>             <dbl> <chr>       <chr>       <chr>
##  1        150000  2018     12000            148114 Royal Enfi~ Individual  1st ~
##  2         65000  2015     23000             89643 Yamaha Faz~ Individual  1st ~
##  3         18000  2010     60000             53857 Honda CB T~ Individual  1st ~
##  4         78500  2018     17000             87719 Honda CB H~ Individual  1st ~
##  5         50000  2016     42000             60122 Bajaj Disc~ Individual  1st ~
##  6         35000  2015     32000             78712 Yamaha FZ16 Individual  1st ~
##  7         28000  2016     10000             47255 Honda Navi  Individual  2nd ~
##  8         80000  2018     21178             95955 Bajaj Aven~ Individual  1st ~
##  9        365000  2019      1127            351680 Yamaha YZF~ Individual  1st ~
## 10         25000  2012     55000             58314 Suzuki Acc~ Individual  1st ~
## # ... with 616 more rows
```

Our 'response' variable here is the `selling_price` and we could use the variable `year`, `km_driven`, or `ex_showroom_price` as the explanatory variable. Let's make some plots and summaries to explore.

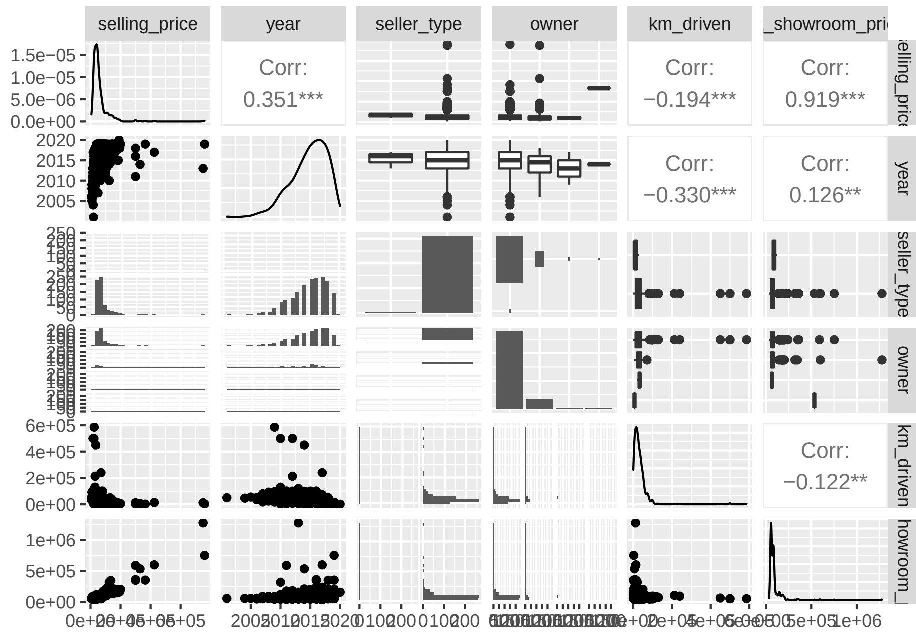
```
summary(bikeData)
```

```
##      name           selling_price         year      seller_type       
##  Length:626         Min.   :  6000   Min.   :2001   Length:626        
##  Class :character   1st Qu.: 30000   1st Qu.:2013   Class :character  
##  Mode  :character   Median : 45000   Median :2015   Mode  :character  
##                     Mean   : 59445   Mean   :2015                     
##                     3rd Qu.: 65000   3rd Qu.:2017                     
##                     Max.   :760000   Max.   :2020                     
##     owner             km_driven      ex_showroom_price
##  Length:626         Min.   :   380   Min.   :  30490  
##  Class :character   1st Qu.: 13031   1st Qu.:  54852  
##  Mode  :character   Median : 25000   Median :  72753  
##                     Mean   : 32672   Mean   :  87959  
##                     3rd Qu.: 40000   3rd Qu.:  87032  
##                     Max.   :585659   Max.   :1278000  
```

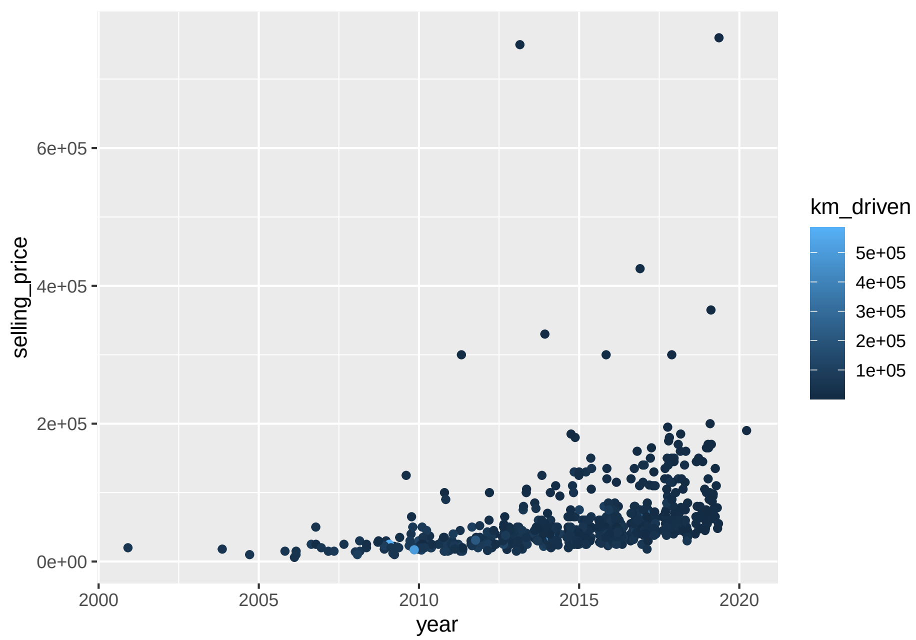
```
summarize(group_by(bikeData, owner),
        mean = mean(selling_price),
        median = median(selling_price),
        sd = sd(selling_price),
        IQR = IQR(selling_price))
```

```
## # A tibble: 4 x 5
##   owner         mean median      sd   IQR
##   <chr>        <dbl>  <dbl>   <dbl> <dbl>
## 1 1st owner  58432.   45000  51125. 35000
## 2 2nd owner  64795.   35000 104861. 30750
## 3 3rd owner  39333.   40000  17010. 17000
## 4 4th owner 330000   330000     NA      0
```

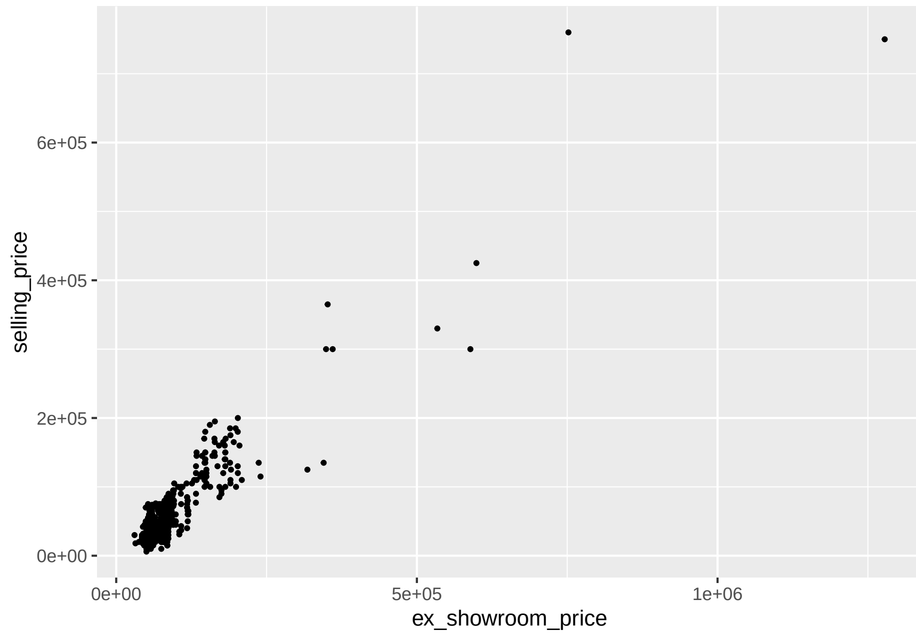
```
library(GGally)
ggpairs(select(bikeData, -name))
```



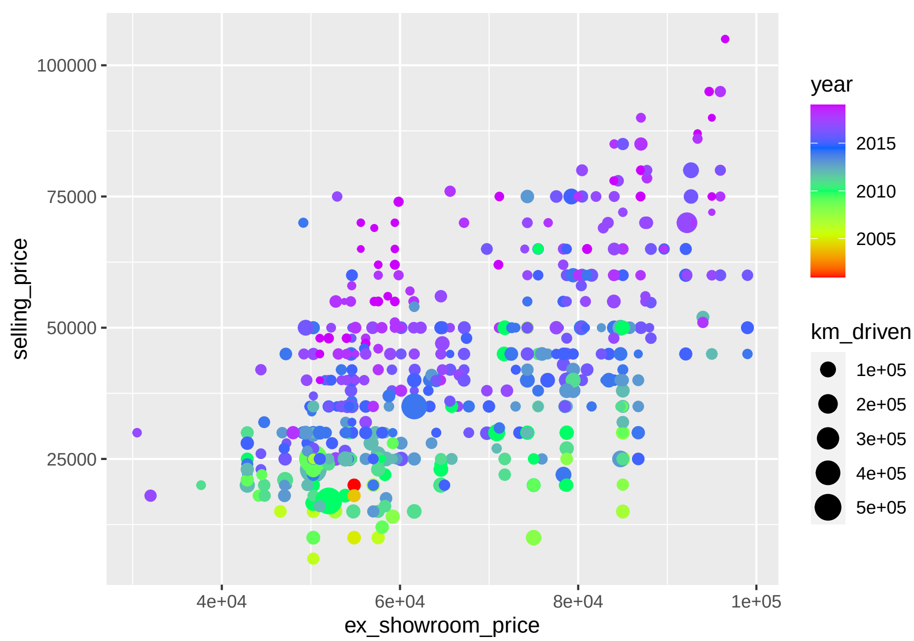
```
g <- ggplot(data = bikeData, aes(y = selling_price))
g + geom_jitter(aes(x = year, color = km_driven))
```




```
g + geom_point(aes(x = ex_showroom_price), size = 0.75)
```




```
g <- ggplot(data = filter(bikeData, ex_showroom_price < 100000), aes(y = selling_price))
g +
  geom_point(aes(x = ex_showroom_price, color = year, size = km_driven)) +
  scale_color_gradientn(colours = rainbow(5))
```



'Fitting' the Model

Basic *linear model* fits done with `lm()`. First argument is a `formula`:

$$response\ variable \sim modeling\ variable(s)$$

We specify the modeling variable(s) with a `+` sign separating variables. With SLR, we only have one variable on the right hand side.

```r
fit <- lm(selling_price ~ ex_showroom_price, data = bikeData)
fit
```

```
##
## Call:
## lm(formula = selling_price ~ ex_showroom_price, data = bikeData)
##
## Coefficients:
##       (Intercept)  ex_showroom_price
##        -3010.6984             0.7101
```

We can easily pull off things like the coefficients.

```r
coefficients(fit) #helper function
```

```
##       (Intercept) ex_showroom_price
##     -3010.6984021         0.7100588
```

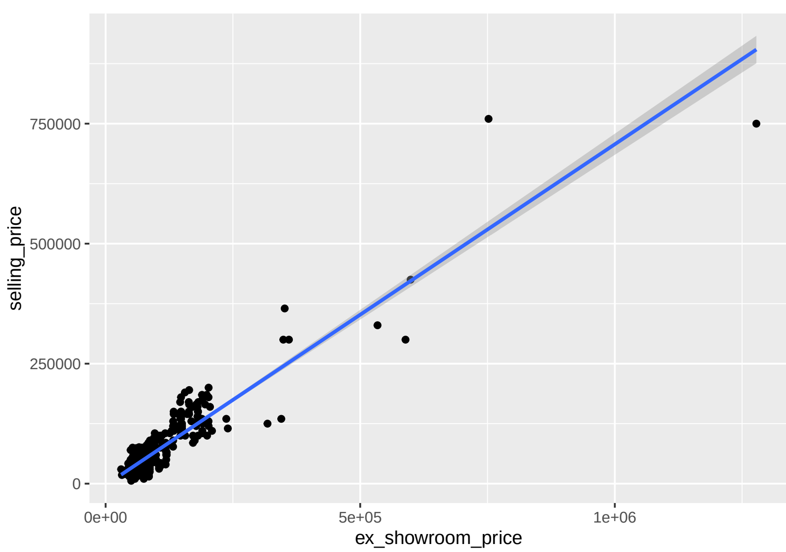
Manually predict for an `ex_showroom_price` of 50000:

```r
intercept <- coefficients(fit)[1]
slope <- coefficients(fit)[2]
intercept + slope * 50000
```

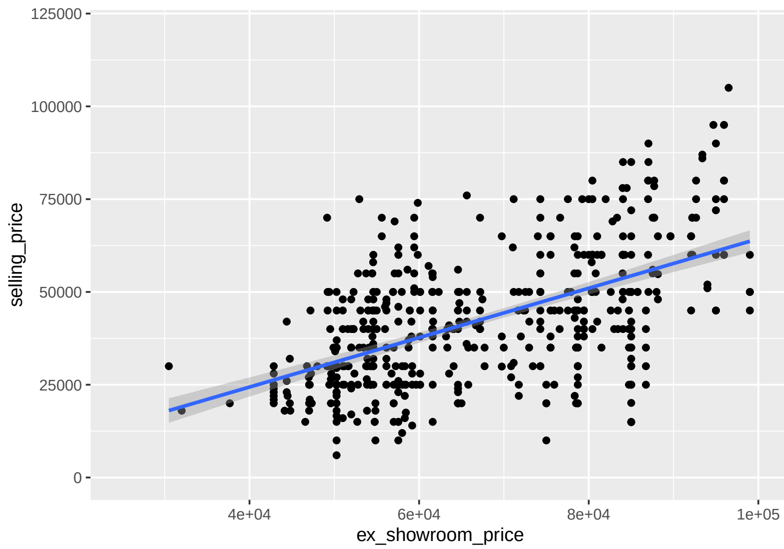
```
## (Intercept)
##    32492.24
```

We can also look at the fit of the line on the graph.

```r
ggplot(bikeData, aes(x = ex_showroom_price, y = selling_price)) +
  geom_point() +
  geom_smooth(method = "lm")
```




```r
ggplot(bikeData, aes(x = ex_showroom_price, y = selling_price)) +
  geom_point() +
  geom_smooth(method = "lm") +
  scale_x_continuous(limits = c(25000, 100000)) +
  scale_y_continuous(limits = c(0, 120000))
```



Predicting!

Can predict the `selling_price` for a given `ex_showroom_price` easily using the `predict()` function.

```
predict(fit, newdata = data.frame(ex_showroom_price = c(50000, 75000, 100000)))
```

```
##        1        2        3
## 32492.24 50243.71 67995.19
```

Error Assumptions

Although, not needed for prediction, we often assume that we observe our response variable Y as a function of the line plus random errors:

$$Y_i = \beta_0 + \beta_1 x_i + E_i$$

where the errors come from a Normal distribution with mean 0 and variance σ^2 ($E_i \overset{iid}{\sim} N(0, \sigma^2)$)

If we do this and use probability theory (maximum likelihood), we will get the same estimates for the slope and interceptas above!

What we get from the normality assumption (if reasonable) is the knowledge of the distribution of our estimators ($\hat{\beta}_0$ and $\hat{\beta}_1$).

What does knowing the distribution allow us to do? We can create confidence intervals or conduct hypothesis tests.

Discuss basic ideas/point of each method.

- Get standard error (SE) for prediction

```
predict(fit, newdata = data.frame(ex_showroom_price = c(50000, 75000, 100000)), se.fit = TRUE)
```

```
## $fit
##        1        2        3
## 32492.24 50243.71 67995.19
##
## $se.fit
##         1         2         3
## 1054.7005  960.2046  958.4166
##
```

```
## $df
## [1] 624
##
## $residual.scale
## [1] 23694.8
```

- Get confidence interval for mean response

```
predict(fit, newdata = data.frame(ex_showroom_price = c(50000, 75000, 100000)),
        se.fit = TRUE, interval = "confidence")
```

```
## $fit
##        fit      lwr      upr
## 1 32492.24 30421.05 34563.44
## 2 50243.71 48358.09 52129.34
## 3 67995.19 66113.07 69877.30
##
## $se.fit
##         1         2         3
## 1054.7005  960.2046  958.4166
##
## $df
## [1] 624
##
## $residual.scale
## [1] 23694.8
```

- Get prediction interval for new response

```
predict(fit, newdata = data.frame(ex_showroom_price = c(50000, 75000, 100000)),
        se.fit = TRUE, interval = "prediction")
```

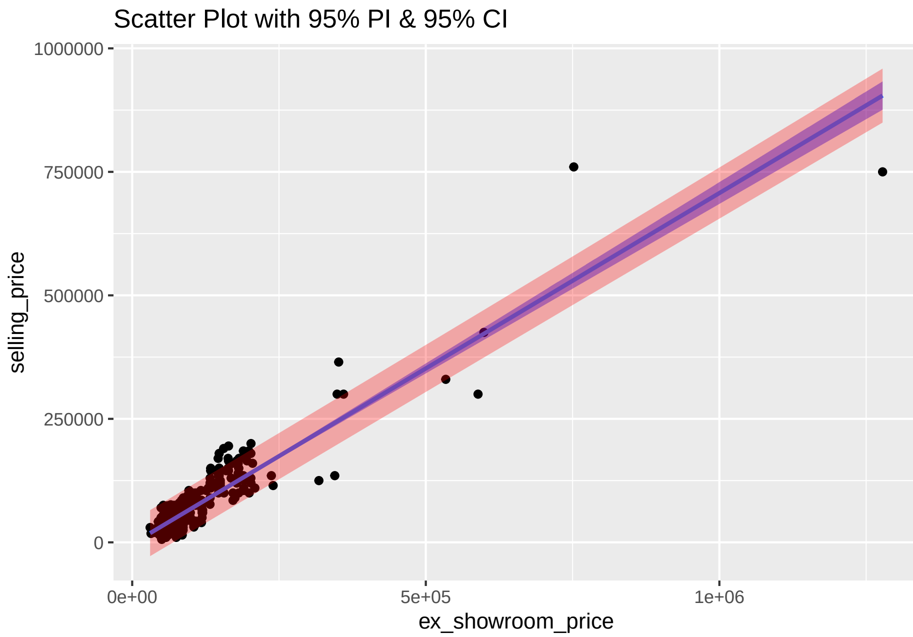
```
## $fit
##        fit        lwr       upr
## 1 32492.24 -14085.045  79069.53
## 2 50243.71   3674.309  96813.12
## 3 67995.19  21425.922 114564.45
##
## $se.fit
##         1         2         3
## 1054.7005  960.2046  958.4166
##
## $df
## [1] 624
##
## $residual.scale
## [1] 23694.8
```

- Can see the confidence and prediction bands on the plot:

```
library(ciTools)
bikeData <- add_pi(bikeData, fit, names = c("lower", "upper"))

ggplot(bikeData, aes(x = ex_showroom_price, y = selling_price)) +
  geom_point() +
    geom_smooth(method = "lm", fill = "Blue") +
    geom_ribbon(aes(ymin = lower, ymax = upper), alpha = 0.3, fill = "Red") +
```

```
    ggtitle("Scatter Plot with 95% PI & 95% CI")
```



For HW have them jump into R and run the code to read in the minesweeper data. Then they could use `lm()` to fit a model and predict.

Multiple Linear Regression

We can add in more than one explanatory variable using the `formula` for `lm()`. The ideas all follow through!

Just show new SSE type equation.

```
fit <- lm(selling_price ~ ex_showroom_price + year + km_driven, data = bikeData)
fit
```

```
##
## Call:
## lm(formula = selling_price ~ ex_showroom_price + year + km_driven,
##     data = bikeData)
##
## Coefficients:
##       (Intercept)  ex_showroom_price               year          km_driven
##        -9.429e+06          6.863e-01          4.679e+03         -1.053e-02
```

To predict we now need to specify values for all the explanatory variables.

```
data.frame(ex_showroom_price = c(50000, 75000),
                                 year = c(2010, 2011),
                                 km_driven = c(15000, 10000))
```

```
##   ex_showroom_price year km_driven
## 1             50000 2010     15000
## 2             75000 2011     10000
```

```
predict(fit, newdata = data.frame(ex_showroom_price = c(50000, 75000),
                                  year = c(2010, 2011),
```

```
                                 km_driven = c(15000, 10000)),
        se.fit = TRUE, interval = "confidence")
```

```
## $fit
##        fit       lwr      upr
## 1 11118.83  7914.815 14322.85
## 2 33007.56 30202.482 35812.63
##
## $se.fit
##        1        2
## 1631.552 1428.402
##
## $df
## [1] 622
##
## $residual.scale
## [1] 19011.31
```

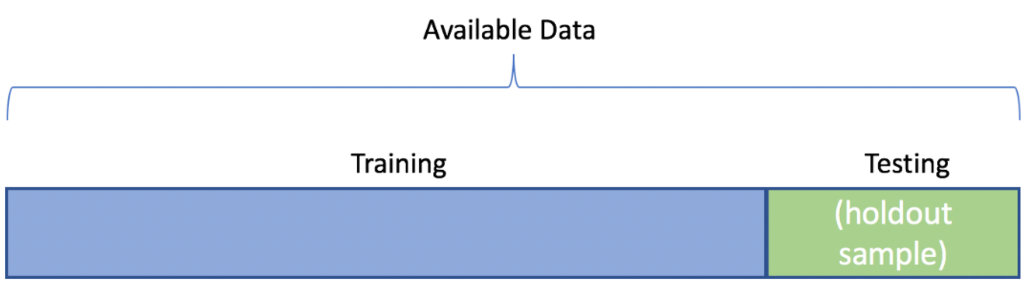
Difficult to visualize the model fit though!

Evaluating Model Accuracy

Which model is better? Ideally we want a model that can predict **new** data better, not the data we've already seen. We need a **test** set to predict on. We also need to quantify what me mean by better!

Training and Test Sets

We can split the data into a **training set** and **test set**.



- On the training set we can fit (or train) our models. The data from the test set isn't used at all in this process.
- We can then predict for the test set observations (for the combinations of explanatory variables seen in the test set). Can then compare the predicted values to the actual observed responses from the test set.

Let's jump into R and fit our SLR model and compare it to an MLR model.

Split data randomly:

```
set.seed(1)
numObs <- nrow(bikeData)
index <- sample(1:numObs, size = 0.7*numObs, replace = FALSE)
train <- bikeData[index, ]
test <- bikeData[-index, ]
```

Fit the models on the training data only.

```r
fitSLR <- lm(selling_price ~ ex_showroom_price , data = train)
fitMLR <- lm(selling_price ~ ex_showroom_price + year + km_driven, data = train)
```

Predict on the test set.

```r
predSLR <- predict(fitSLR, newdata = test)
predMLR <- predict(fitMLR, newdata = test)
tibble(predSLR, predMLR, test$selling_price)
```

```
## # A tibble: 188 x 3
##    predSLR predMLR `test$selling_price`
##      <dbl>   <dbl>                <dbl>
##  1  58966.  73988.                78500
##  2 244701. 258749.               365000
##  3  80221.  58088.                40000
##  4 101463. 114943.               150000
##  5  90603. 103928.               120000
##  6  28477.  39802.                42000
##  7  37743.  53770.                60000
##  8  40588.  56071.                45000
##  9  32173.  34042.                28000
## 10 101463. 114974.               140000
## # ... with 178 more rows
```

Root Mean Square Error

Which is better?? Can use squared error loss to evaluate! (Square root of the mean squared error loss is often reported instead and is called RMSE or Root Mean Square Error.)

```r
sqrt(mean((predSLR - test$selling_price)^2))
```

```
## [1] 23026.97
```

```r
sqrt(mean((predMLR - test$selling_price)^2))
```

```
## [1] 17439.81
```

MLR fit does much better at predicting!

Another Modeling Approach (k Nearest Neighbors)

Day 4:

kNN relate to idea with minesweeper data about how they predicted for their individual piece.

Will need to use k = 10, 30, 50, ...

Intro and Recap

Recap: Our previous goal was to predict a value of Y while including an explanatory variable x. With that x, we said $E(Y|x)$ will minimize

$$E\left[(Y-c)^2|x\right]$$

We called this true unknown value $E(Y|x) = f(x)$.

Given observed Y's and x's, we can estimate this function as $\hat{f}(x)$ (with SLR we estimated it with $\hat{\beta}_0 + \hat{\beta}_1 x$). This $\hat{f}(x)$ will minimize

$$g(y_1, ..., y_n|x_1, ..., x_n) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}(x_i)) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

What other things could we consider for $f(x)$???

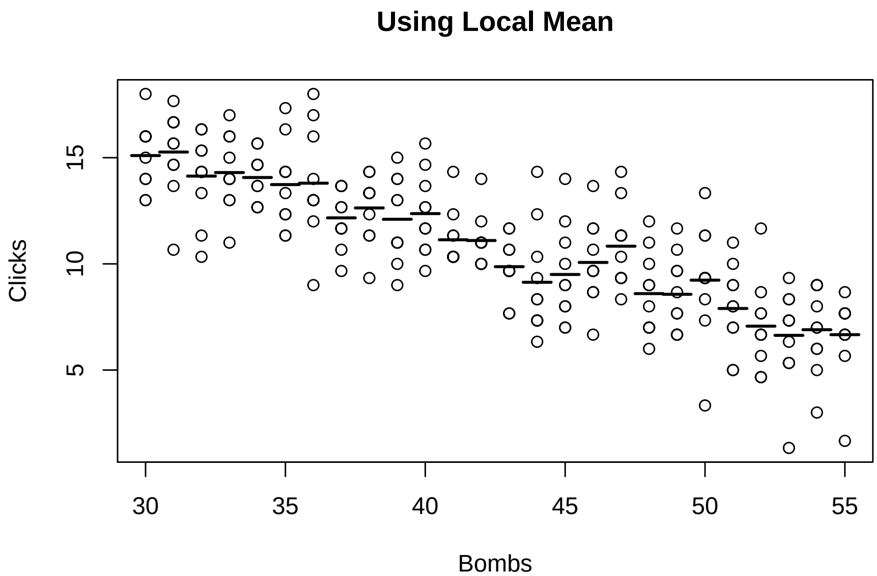
Consider the minesweeper data we collected previously.

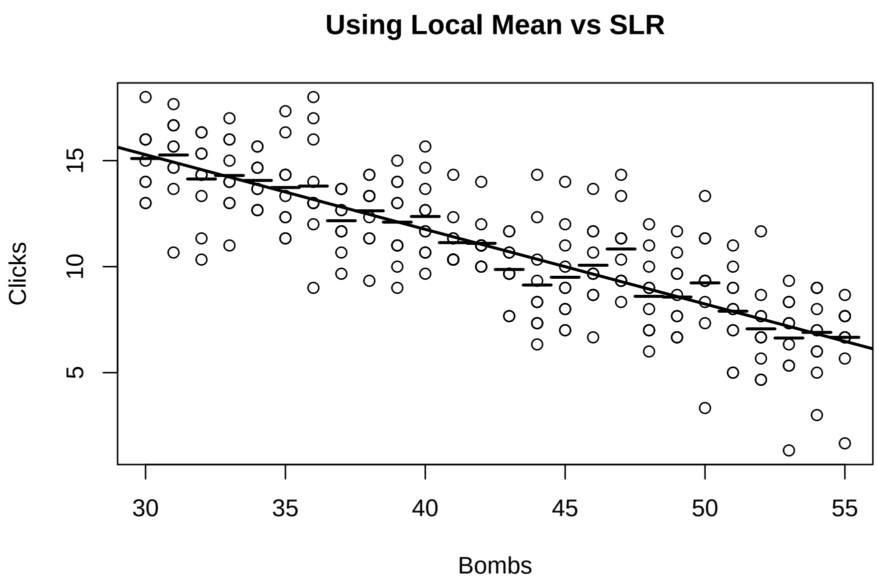
Recall the minesweeper data we collected. In your homework, you were asked to determine a prediction of the number of blocks you could click before hitting a bomb for **your** given number of bombs. You were each creating your estimate of $f(x)$ for your x value!

Let's visualize that idea and compare it to the SLR fit!

I'd need the data here but this is what it would look like. I won't show the code here.

```
## # A tibble: 26 x 2
##    bombs  mean
##    <int> <dbl>
##  1    30  15.1
##  2    31  15.3
##  3    32  14.1
##  4    33  14.3
##  5    34  14.1
##  6    35  13.7
##  7    36  13.8
##  8    37  12.2
##  9    38  12.6
## 10    39  12.1
## # ... with 16 more rows
```





This is the idea of k Nearest Neighbors (kNN) for predicting a numeric response!

kNN

To predict a value of our (numeric) response kNN uses the **average of the k 'closest' responses**. For numeric data, we usually use Euclidean distance ($d(x_1, x_2) = \sqrt{(x_1 - x_2)^2}$) to determine the closest values.

- Large k implies more rigid (possibly *underfit* but lower variance prediction).
- Smaller k implies less rigid (possible *overfit* with high variance in prediction)

Let's check out this app.

For the minesweeper data, we had many values at the same x (# of bombs). That's why we considered using only 10, 30, 50, ... Otherwise, we have ties and then things get tricky!

Choosing the Value of k

How do we choose which k value to use? We can do a similar training vs test set idea. Fit the models (one model for each k) and predict on the test set. The model with the lowest Root Mean Squared Error (RMSE) on the test set can be chosen!

kNN Models for `selling_price` from the Bike Dataset

Previously, we fit the SLR model using the `ex_showroom_price` to predict our `selling_price` of motorcycles. We'll refit this using the training data here.

```
fitSLR <- lm(selling_price ~ ex_showroom_price, data = train)
fitSLR
```

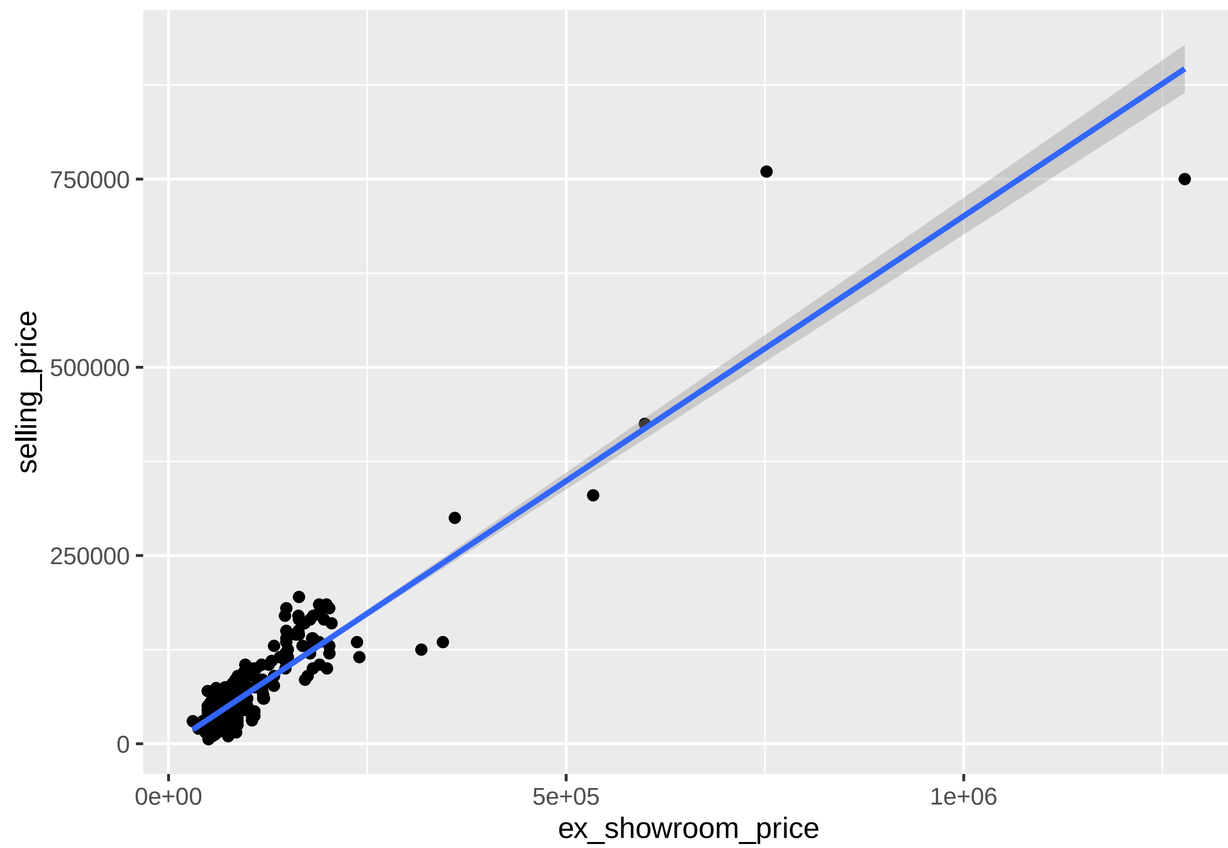
```
##
## Call:
## lm(formula = selling_price ~ ex_showroom_price, data = train)
##
## Coefficients:
##       (Intercept)  ex_showroom_price
##        -2756.9143             0.7036
```

```
summary(fitSLR)
```

```
##
## Call:
## lm(formula = selling_price ~ ex_showroom_price, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -146502  -11685   -2135   10303  233602
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -2.757e+03  1.663e+03  -1.658    0.098 .
## ex_showroom_price  7.036e-01  1.359e-02  51.779   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23990 on 436 degrees of freedom
## Multiple R-squared:  0.8601, Adjusted R-squared:  0.8598
## F-statistic:  2681 on 1 and 436 DF,  p-value: < 2.2e-16
```

```
ggplot(train, aes(x = ex_showroom_price, y = selling_price)) +
  geom_point() +
  geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Obtain the prediction on the test set.

```
predSLR <- predict(fitSLR, newdata = test)
```

Let's now fit the kNN model using a few values of k.

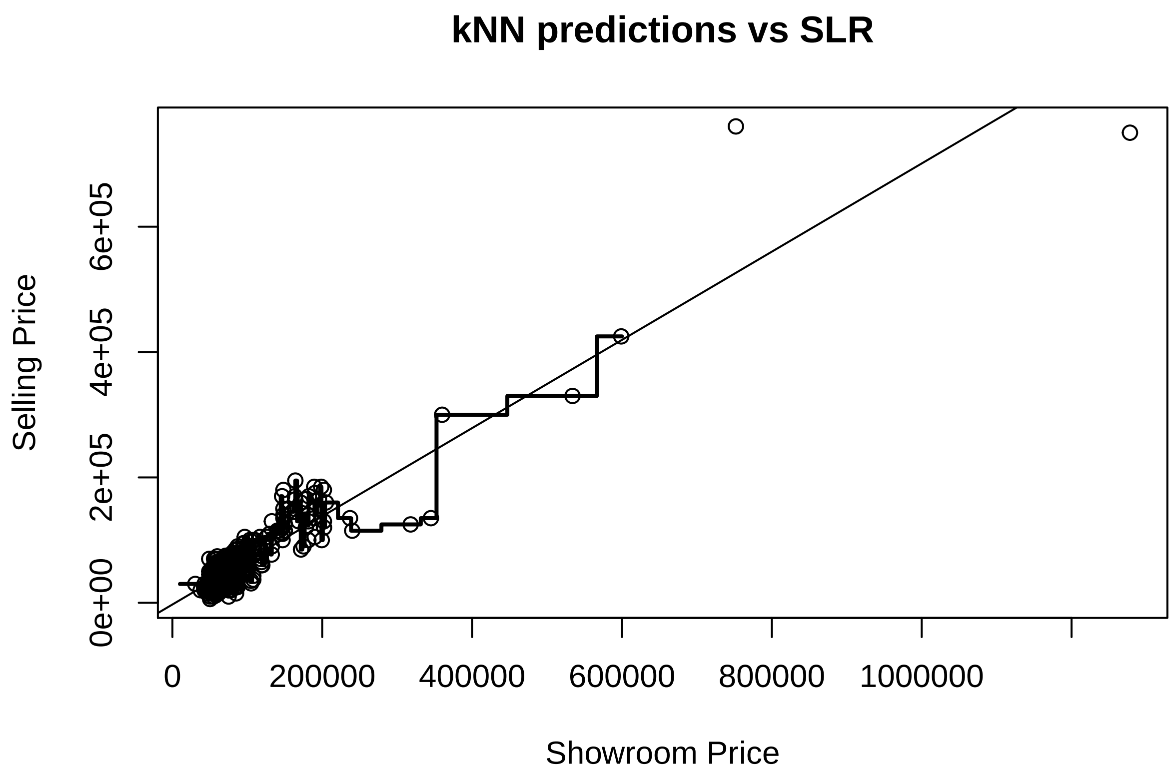
$k = 1$:

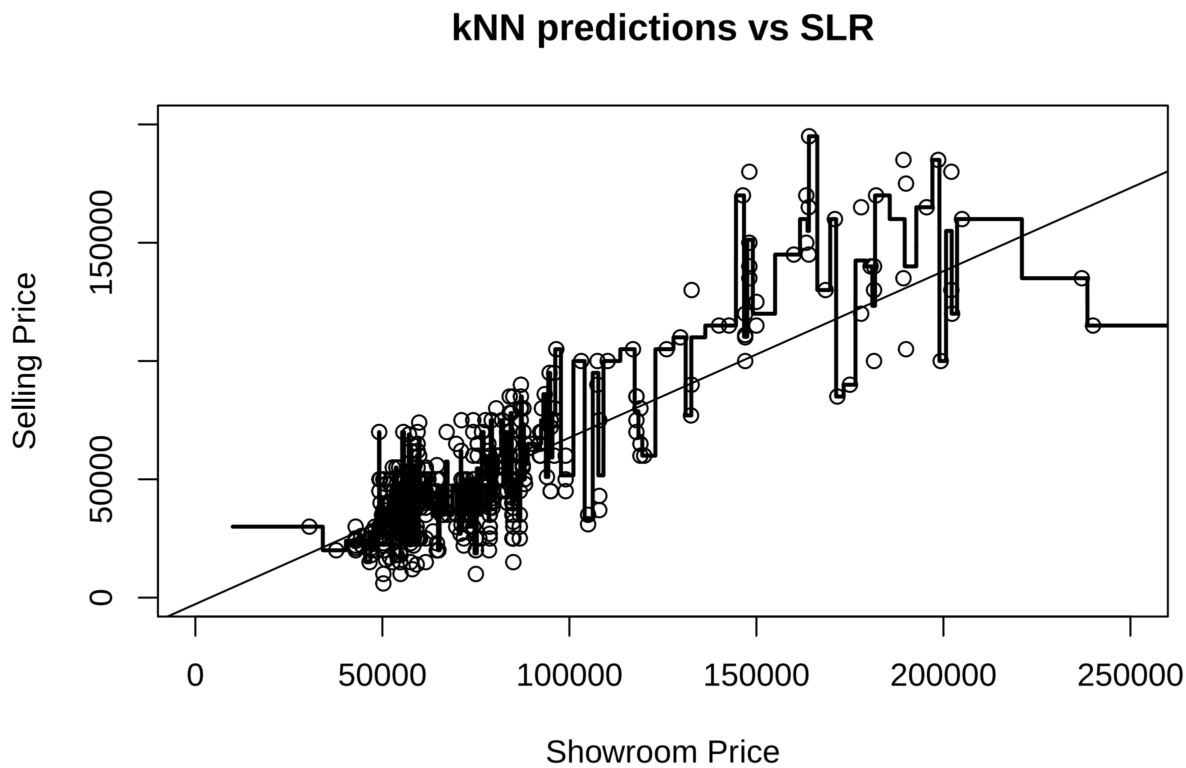
```
library("caret")
k <- 1
kNNFit1 <- train(selling_price ~ ex_showroom_price,
      data = train,
      method = "knn",
      tuneGrid = data.frame(k =k)
      )
kNNFit1
```

```
## k-Nearest Neighbors
##
## 438 samples
##   1 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 438, 438, 438, 438, 438, 438, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
```

```
##   29919.47  0.7849003  16569.5
##
## Tuning parameter 'k' was held constant at a value of 1
```

```r
predkNN1 <- predict(kNNFit1, newdata = test)
```



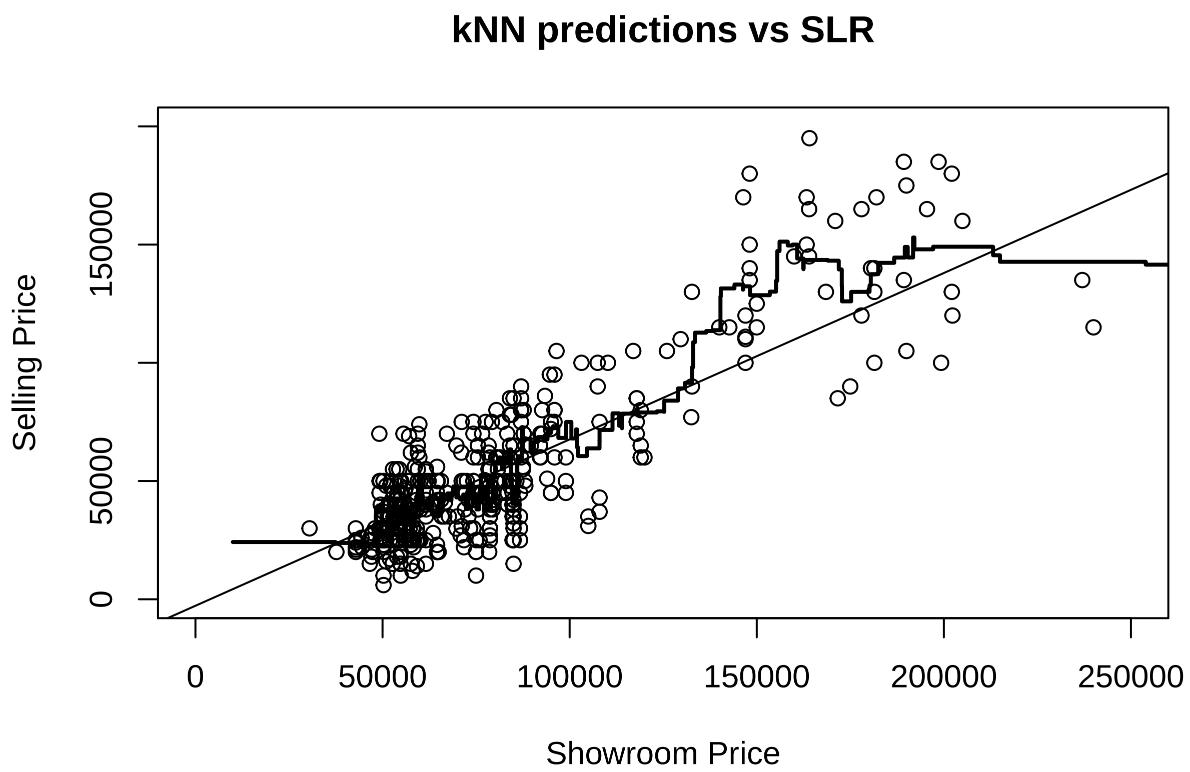


$k = 10$:

```
k <- 10
kNNFit10 <- train(selling_price ~ ex_showroom_price,
      data = train,
      method = "knn",
      tuneGrid = data.frame(k =k)
      )
kNNFit10
```

```
## k-Nearest Neighbors
##
## 438 samples
##   1 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 438, 438, 438, 438, 438, 438, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   36256.39  0.7045166  16586.28
##
## Tuning parameter 'k' was held constant at a value of 10
```

```r
predkNN10 <- predict(kNNFit10, newdata = test)
```

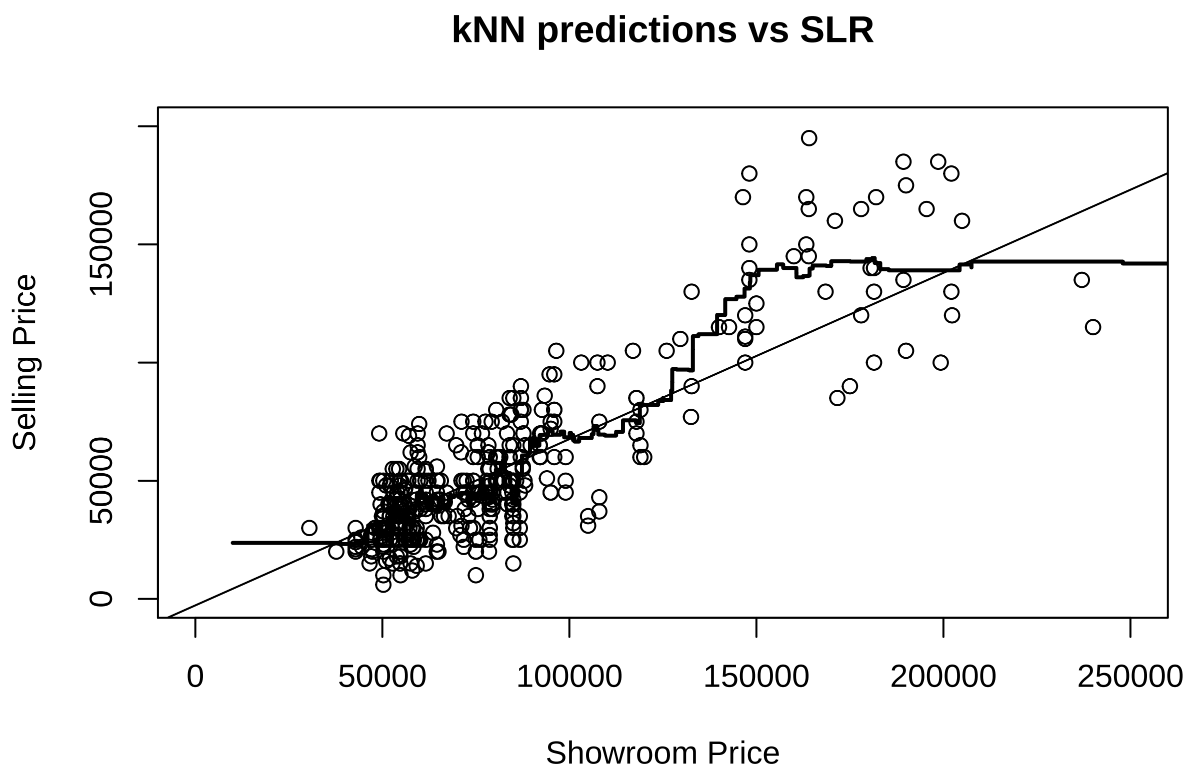


$k = 20$:

```r
k <- 20
kNNFit20 <-train(selling_price ~ ex_showroom_price,
      data = train,
      method = "knn",
      tuneGrid = data.frame(k =k)
      )
kNNFit20
```

```
## k-Nearest Neighbors
##
## 438 samples
##   1 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 438, 438, 438, 438, 438, 438, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   38713.86  0.6601001  16633.84
##
## Tuning parameter 'k' was held constant at a value of 20
```

```r
predkNN20 <- predict(kNNFit20, newdata = test)
```

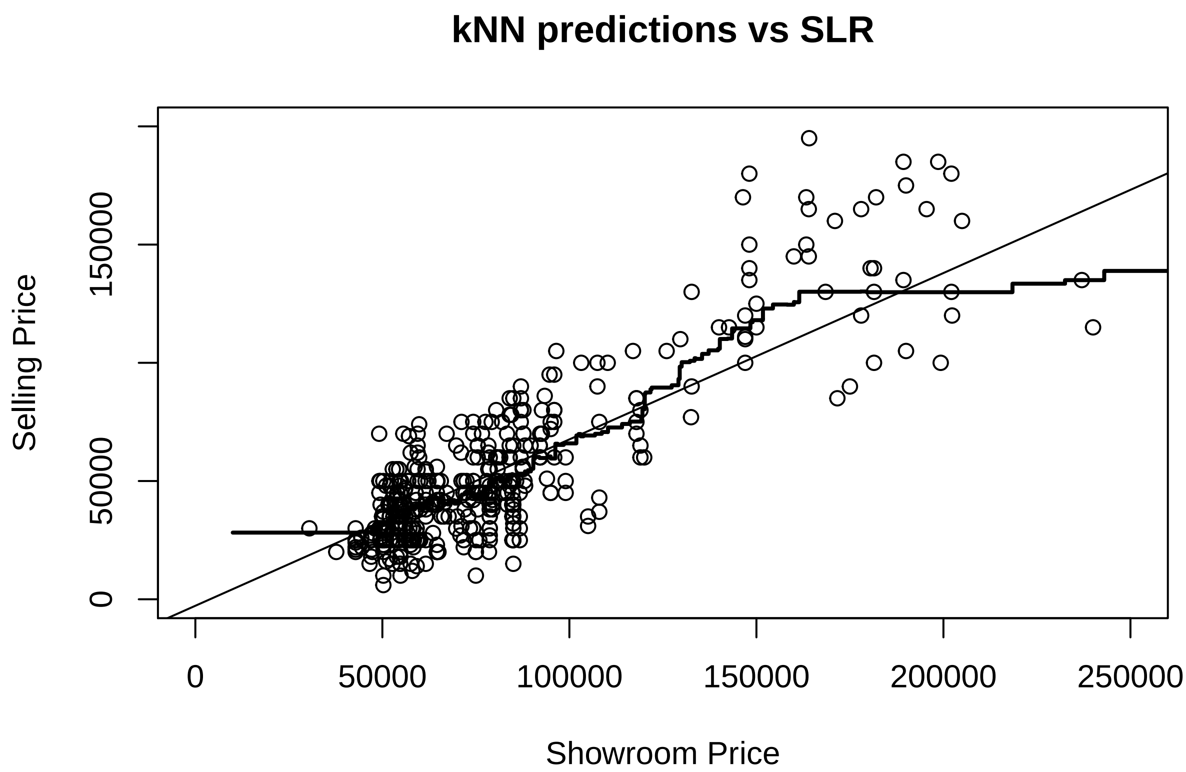


$k = 50$:

```r
k <- 50
kNNFit50 <- train(selling_price ~ ex_showroom_price,
      data = train,
      method = "knn",
      tuneGrid = data.frame(k =k)
      )
kNNFit50
```

```
## k-Nearest Neighbors
##
## 438 samples
##   1 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 438, 438, 438, 438, 438, 438, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   41931.03  0.6175174  17212.5
##
## Tuning parameter 'k' was held constant at a value of 50
```

```
predkNN50 <- predict(kNNFit50, newdata = test)
```

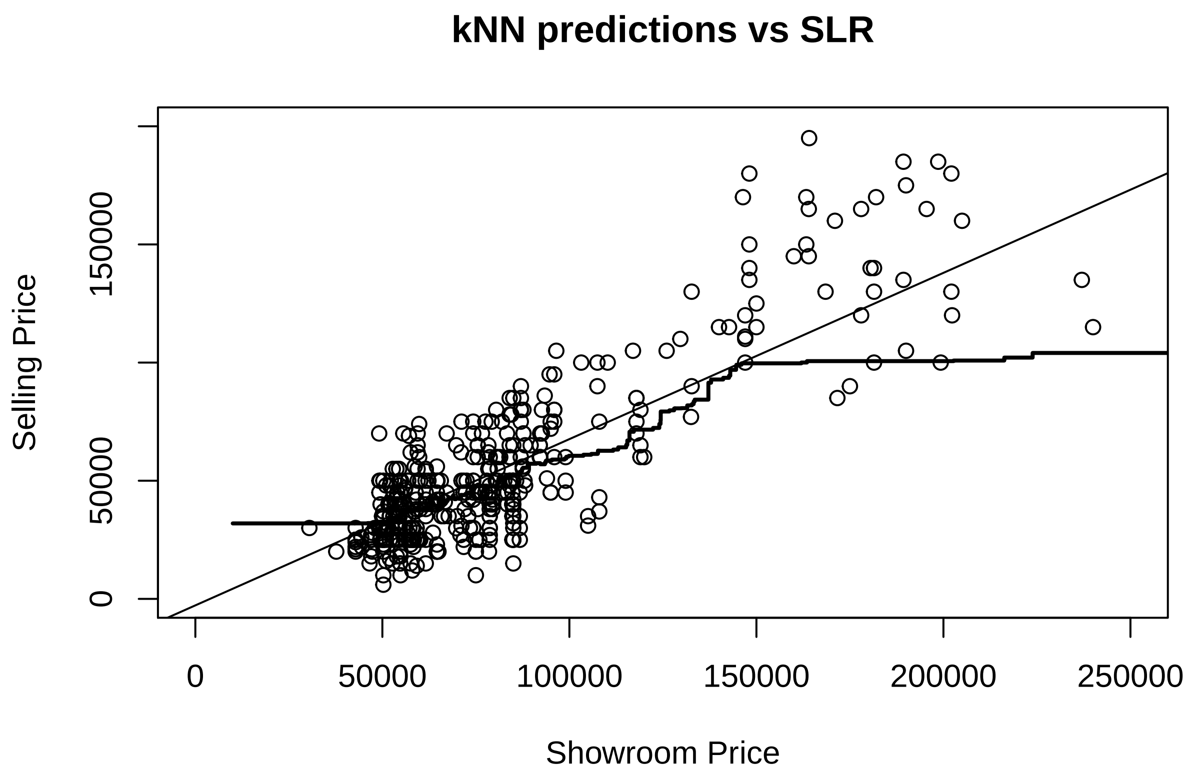


$k = 100$:

```
k <- 100
kNNFit100 <- train(selling_price ~ ex_showroom_price,
      data = train,
      method = "knn",
      tuneGrid = data.frame(k =k)
      )
kNNFit100
```

```
## k-Nearest Neighbors
##
## 438 samples
##   1 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 438, 438, 438, 438, 438, 438, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   56678.71  0.5195777  20813.64
##
## Tuning parameter 'k' was held constant at a value of 100
```

```r
predkNN100 <- predict(kNNFit100, newdata = test)
```



Compare test set RMSE!

```r
RMSE <- function(pred, test){sqrt(mean((pred-test)^2))}
SLR <- RMSE(predSLR, test$selling_price)
kNN1 <- RMSE(predkNN1, test$selling_price)
kNN10 <- RMSE(predkNN10, test$selling_price)
kNN20 <- RMSE(predkNN20, test$selling_price)
kNN50 <- RMSE(predkNN50, test$selling_price)
kNN100 <- RMSE(predkNN100, test$selling_price)

data.frame(method = c("SLR", "kNN1", "kNN10", "kNN20", "kNN50", "kNN100"),
           RMSE = c(SLR, kNN1, kNN10, kNN20, kNN50, kNN100))
```

```
##   method     RMSE
## 1    SLR 23026.97
## 2   kNN1 27897.53
## 3  kNN10 25660.12
## 4  kNN20 26407.66
## 5  kNN50 28344.37
## 6 kNN100 34072.25
```

Ok, of course we don't want to do this manually in real life… What we actually do:

1. Split the data into training and test sets
2. Choose a 'best' model for a given method (MLR, kNN, etc.) using the training set
    - This requires us to have a method to choose using only the training data!

3. Compare the best model from each method on the test set to see how they do
4. Refit the chosen model on the full data set. This model would then be what you would use for future predictions.

R makes it easy! To choose a kNN model we can run code like this:

```r
k <- 1:100
kNNFit <- train(selling_price ~ ex_showroom_price,
      data = train,
      method = "knn",
      tuneGrid = data.frame(k =k),
      trControl = trainControl(method = "cv", number = 10)
      )
kNNFit
```

```
## k-Nearest Neighbors
##
## 438 samples
##   1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 393, 394, 393, 394, 394, 395, ...
## Resampling results across tuning parameters:
##
##   k    RMSE      Rsquared   MAE
##     1  25904.73  0.7737071  15818.28
##     2  25056.89  0.8070520  15245.05
##     3  25850.39  0.8166904  14899.56
##     4  27503.92  0.8033103  15312.44
##     5  28770.16  0.7938322  15619.93
##     6  28553.09  0.8027001  15747.58
##     7  29835.75  0.7891728  16092.20
##     8  30720.22  0.7806901  16162.82
##     9  31313.01  0.7728020  16270.00
##    10  31994.24  0.7622684  16432.57
##    11  32348.47  0.7582698  16391.57
##    12  32603.39  0.7575916  16324.25
##    13  32920.38  0.7522707  16290.40
##    14  33088.32  0.7506262  16254.19
##    15  33263.18  0.7461206  16318.78
##    16  33618.11  0.7410228  16462.95
##    17  33867.10  0.7389925  16557.38
##    18  34287.07  0.7317341  16668.59
##    19  34277.44  0.7303175  16658.02
##    20  34417.47  0.7264348  16706.93
##    21  34548.00  0.7233476  16759.79
##    22  34794.77  0.7199672  16794.70
##    23  34810.98  0.7200562  16761.74
##    24  34993.35  0.7175039  16787.08
##    25  35171.71  0.7150430  16784.03
##    26  35189.16  0.7161863  16805.63
##    27  35331.23  0.7135606  16851.94
##    28  35365.14  0.7132777  16862.88
##    29  35385.36  0.7122885  16831.37
```

```
## 30 35449.53 0.7108309 16833.85
## 31 35556.21 0.7107566 16852.93
## 32 35549.46 0.7123548 16848.74
## 33 35652.32 0.7121986 16845.50
## 34 35798.56 0.7084975 16959.81
## 35 35859.72 0.7068521 16980.25
## 36 35932.48 0.7058858 16988.45
## 37 35989.22 0.7050570 16990.60
## 38 36072.06 0.7053125 17044.79
## 39 36267.88 0.7023702 17080.37
## 40 36278.49 0.7039122 17116.17
## 41 36338.89 0.7051274 17152.85
## 42 36417.57 0.7044305 17197.94
## 43 36489.55 0.7054950 17178.37
## 44 36710.31 0.7046807 17287.47
## 45 36811.80 0.7051789 17294.24
## 46 36850.80 0.7063547 17289.71
## 47 36965.92 0.7067176 17354.09
## 48 37111.03 0.7061536 17394.91
## 49 37288.64 0.7039797 17443.22
## 50 37444.91 0.7002947 17500.34
## 51 37692.81 0.6973852 17581.45
## 52 37857.57 0.6957976 17649.23
## 53 38045.83 0.6930176 17695.22
## 54 38205.98 0.6909715 17791.70
## 55 38233.19 0.6939117 17830.28
## 56 38359.42 0.6933284 17898.97
## 57 38442.89 0.6945296 17921.76
## 58 38651.31 0.6915580 18007.98
## 59 38847.64 0.6889681 18118.40
## 60 38986.35 0.6906318 18176.17
## 61 39064.30 0.6908630 18178.68
## 62 39144.81 0.6897898 18230.25
## 63 39333.04 0.6895490 18331.68
## 64 39495.53 0.6872812 18395.71
## 65 39633.12 0.6846900 18404.26
## 66 39746.99 0.6842957 18449.01
## 67 39799.84 0.6864295 18497.58
## 68 39924.16 0.6853453 18541.09
## 69 39980.10 0.6864826 18558.58
## 70 40158.83 0.6824437 18610.91
## 71 40249.58 0.6821368 18657.83
## 72 40344.77 0.6828577 18698.59
## 73 40428.59 0.6844805 18777.18
## 74 40510.00 0.6852928 18817.00
## 75 40585.02 0.6851775 18836.98
## 76 40694.82 0.6851035 18912.83
## 77 40833.42 0.6838674 18932.22
## 78 40908.10 0.6832563 18943.97
## 79 40999.55 0.6826276 18968.80
## 80 41112.59 0.6823271 19052.37
## 81 41175.25 0.6831190 19070.73
## 82 41325.35 0.6803447 19128.99
## 83 41395.72 0.6816859 19164.62
```

```
##    84  41486.06  0.6821614  19210.48
##    85  41570.79  0.6823958  19271.42
##    86  41696.75  0.6804796  19352.22
##    87  41808.97  0.6794509  19421.28
##    88  41875.45  0.6785051  19432.63
##    89  41966.92  0.6772777  19493.21
##    90  42038.66  0.6761105  19510.26
##    91  42126.68  0.6752747  19551.52
##    92  42191.36  0.6755212  19620.99
##    93  42352.12  0.6744606  19715.64
##    94  42419.87  0.6745983  19767.47
##    95  42599.98  0.6733391  19885.68
##    96  42661.41  0.6726663  19898.98
##    97  42784.86  0.6731938  19962.98
##    98  42968.79  0.6763404  20124.66
##    99  43200.93  0.6733748  20201.59
##   100  43437.70  0.6728062  20330.94
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 2.
```

The chosen model can then be used to predict just as before.

```r
predkNN <- predict(kNNFit, newdata = test)
postResample(predkNN, test$selling_price)
```

```
##         RMSE     Rsquared          MAE
## 2.119984e+04 8.137631e-01 1.396988e+04
```

The same process can be used to fit and predict for an SLR or MLR model.

```r
SLRFit <- train(selling_price ~ ex_showroom_price,
                data = train,
                method = "lm",
                trControl = trainControl(method = "cv", number = 10)
                )
SLRFit
```

```
## Linear Regression
##
## 438 samples
##   1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 394, 394, 394, 395, 393, 394, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   25238.82  0.7870342  15841.65
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
predSLR <- predict(SLRFit, newdata = test)
postResample(predSLR, test$selling_price)
```

```
##         RMSE     Rsquared          MAE
```

```
## 2.302697e+04 7.806227e-01 1.608487e+04
```

Multiple Predictors

Just like SLR can include multiple explanatory variables, we can include multiple explanatory variables with kNN (they must all be numeric unless you develop or use a 'distance' measure that is appropriate for categorical data).

With all numeric explanatory variables, we often use Euclidean distance as our distance metric. For instance, with two explanatory variables x_1 and x_2:

$$d(x_1, x_2) = \sqrt{(x_{11} - x_{12})^2 + (x_{21} - x_{22})^2}$$

The same model notation from before can be used:

$$respons\ variable \sim explanatory\ variable1 + explanatory\ variable2 + ...$$

Along with the same kind of R code to fit the model:

```
k <- 1:100
kNNFit <- train(selling_price ~ ex_showroom_price + km_driven + year,
      data = train,
      method = "knn",
      tuneGrid = data.frame(k =k),
      trControl = trainControl(method = "cv", number = 10)
      )
kNNFit
```

```
## k-Nearest Neighbors
##
## 438 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 394, 396, 396, 393, 395, 392, ...
## Resampling results across tuning parameters:
##
##   k    RMSE      Rsquared   MAE
##     1  26485.95  0.7453587  17249.50
##     2  25888.44  0.7974705  15931.94
##     3  25684.42  0.8149551  15075.93
##     4  26854.17  0.8165682  14911.70
##     5  28273.68  0.8024523  14932.50
##     6  29545.25  0.7891693  15034.19
##     7  28915.83  0.8102074  14804.87
##     8  29869.35  0.7994379  14952.79
##     9  30501.79  0.7908927  15043.70
##    10  31209.53  0.7795996  15126.41
##    11  31745.91  0.7726032  15234.65
##    12  32128.38  0.7654519  15295.04
##    13  32137.88  0.7702724  15150.58
##    14  32441.58  0.7645767  15112.73
##    15  32515.00  0.7650274  14969.01
##    16  32817.06  0.7597333  14990.08
##    17  32910.42  0.7590637  15005.94
```

```
## 18 33056.63 0.7576681 14935.64
## 19 33323.63 0.7531680 14965.07
## 20 33608.82 0.7485403 15030.60
## 21 33835.81 0.7448975 15112.90
## 22 33957.59 0.7442549 15114.15
## 23 34132.02 0.7411115 15203.17
## 24 34238.33 0.7402716 15189.22
## 25 34394.95 0.7369021 15263.48
## 26 34592.74 0.7335659 15309.38
## 27 34695.74 0.7325211 15338.03
## 28 34779.08 0.7306775 15359.30
## 29 34809.76 0.7306110 15371.64
## 30 34893.29 0.7292591 15396.23
## 31 34969.68 0.7284088 15424.85
## 32 35040.27 0.7292750 15427.96
## 33 35118.72 0.7293264 15496.50
## 34 35231.24 0.7275798 15544.48
## 35 35375.63 0.7257651 15605.65
## 36 35452.09 0.7248865 15626.76
## 37 35507.85 0.7242370 15639.91
## 38 35581.71 0.7240194 15641.15
## 39 35627.25 0.7245175 15670.03
## 40 35799.36 0.7221820 15759.47
## 41 35902.12 0.7208025 15804.86
## 42 36006.19 0.7200243 15820.42
## 43 36144.54 0.7196128 15866.79
## 44 36236.38 0.7201557 15917.49
## 45 36288.40 0.7217948 15961.05
## 46 36404.06 0.7218532 16012.10
## 47 36592.27 0.7187822 16108.94
## 48 36729.54 0.7189574 16148.60
## 49 36863.14 0.7180808 16192.81
## 50 37029.52 0.7169798 16272.73
## 51 37206.23 0.7154082 16331.80
## 52 37404.28 0.7128655 16395.54
## 53 37516.55 0.7126394 16438.28
## 54 37673.75 0.7115667 16501.46
## 55 37823.50 0.7109514 16563.38
## 56 37980.86 0.7107232 16632.11
## 57 38145.50 0.7090360 16711.17
## 58 38283.84 0.7091538 16774.36
## 59 38470.17 0.7066650 16846.63
## 60 38637.72 0.7057442 16912.71
## 61 38734.97 0.7059584 16962.66
## 62 38895.09 0.7036605 17037.56
## 63 39043.80 0.7031642 17092.54
## 64 39192.85 0.7017078 17145.90
## 65 39330.27 0.7008909 17203.78
## 66 39423.44 0.7013463 17244.20
## 67 39538.86 0.7009596 17302.29
## 68 39639.10 0.7008519 17352.73
## 69 39811.10 0.6986129 17402.53
## 70 39928.35 0.6986864 17473.43
## 71 40031.45 0.6983731 17540.50
```

```
##    72  40139.67  0.6985453  17592.70
##    73  40290.28  0.6976118  17650.03
##    74  40384.39  0.6973126  17690.66
##    75  40526.40  0.6957159  17752.31
##    76  40622.61  0.6952652  17808.15
##    77  40783.47  0.6939245  17877.57
##    78  40893.98  0.6939233  17934.27
##    79  41004.64  0.6933813  17990.99
##    80  41145.02  0.6914472  18053.38
##    81  41264.74  0.6905914  18106.26
##    82  41358.27  0.6900105  18164.05
##    83  41455.17  0.6896838  18202.36
##    84  41574.55  0.6884006  18240.99
##    85  41672.40  0.6889835  18281.36
##    86  41776.43  0.6883291  18342.09
##    87  41881.80  0.6880274  18403.97
##    88  42007.78  0.6869543  18482.50
##    89  42108.39  0.6867971  18528.35
##    90  42234.53  0.6857172  18586.45
##    91  42329.68  0.6858546  18645.08
##    92  42443.63  0.6854015  18715.68
##    93  42572.72  0.6841337  18780.32
##    94  42659.67  0.6843586  18828.77
##    95  42738.99  0.6842556  18881.76
##    96  42844.06  0.6838632  18926.54
##    97  42946.69  0.6832781  18982.89
##    98  43038.41  0.6835034  19032.13
##    99  43115.98  0.6836748  19066.24
##   100  43216.09  0.6824380  19117.57
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 3.
```

The chosen model can then be used to predict just as before.

```r
predkNN <- predict(kNNFit, newdata = test)
postResample(predkNN, test$selling_price)
```

```
##         RMSE     Rsquared          MAE
## 2.728151e+04 7.113783e-01 1.552835e+04
```

Just for reference: let's compare this to the MLR output.

```r
MLRFit <- train(selling_price ~ ex_showroom_price + km_driven + year,
      data = train,
      method = "lm",
      trControl = trainControl(method = "cv", number = 10)
      )
MLRFit
```

```
## Linear Regression
##
## 438 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
```

```
## Summary of sample sizes: 395, 394, 393, 394, 395, 394, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   19664.54  0.8836404  11735.8
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
predMLR <- predict(MLRFit, newdata = test)
postResample(predMLR, test$selling_price)
```

```
##         RMSE     Rsquared          MAE
## 1.743981e+04 8.741091e-01 1.158801e+04
```

Note: Practical use of kNN says we should usually standardize (center to have mean 0 and scale to have standard deviation 1) our numeric explanatory variables. Why?

Day 5

Competition!

Time to put what we've learned into practice! Kaggle is a site that hosts competitions around predicting a response (either a numeric response or predicting the category that an observation might belong to).

Housing Prices

Let's go check out our competition: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview

Use the starter files to come up with some models!