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.

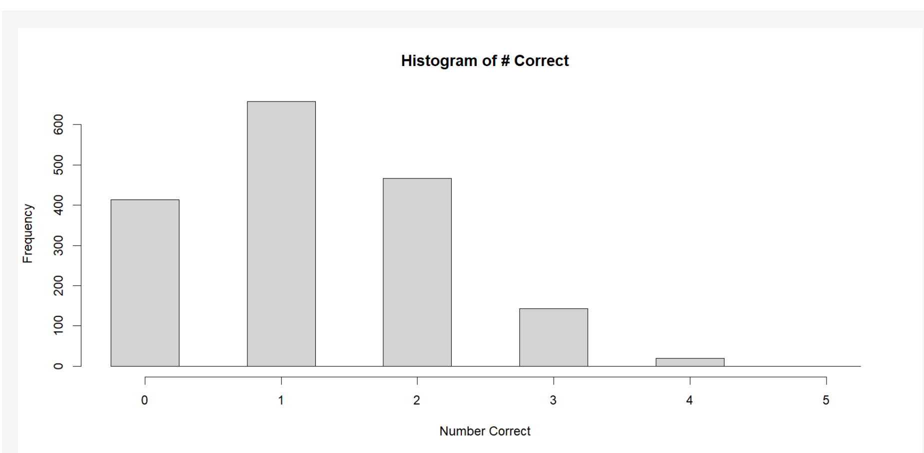
What's the point? How can I best predict the number of card suits the next person will get right?

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?

- 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")`. - 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. - 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).

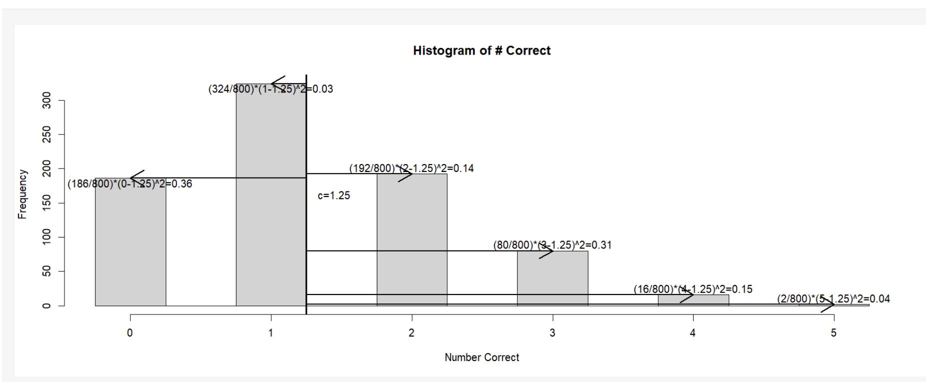
Relating Explanatory Variables in Prediction

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.

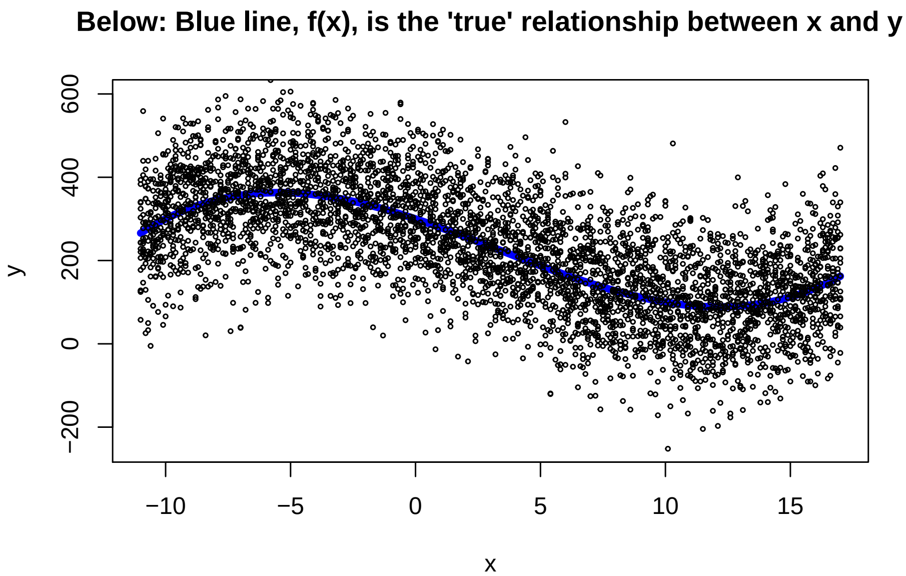
When we considered just Y by itself and used squared error loss, we know that $E(Y) = \mu$ minimizes

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

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

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

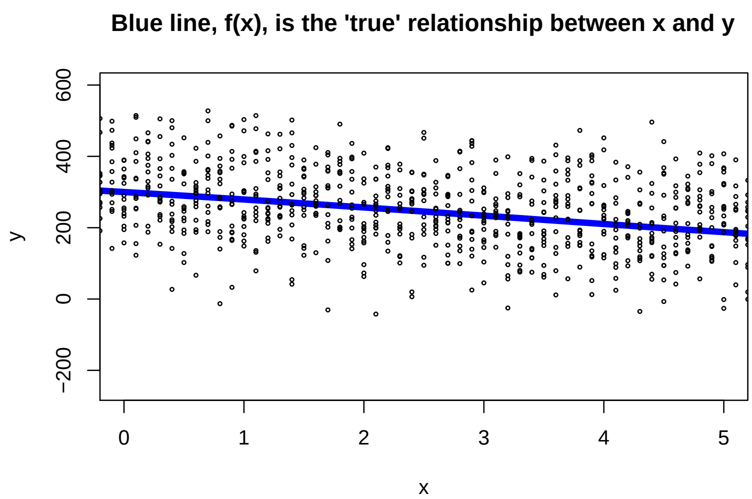
We can call this true unknown value $E(Y|x) = f(x)$. That is, the average value of Y will now be considered as a function of x.

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

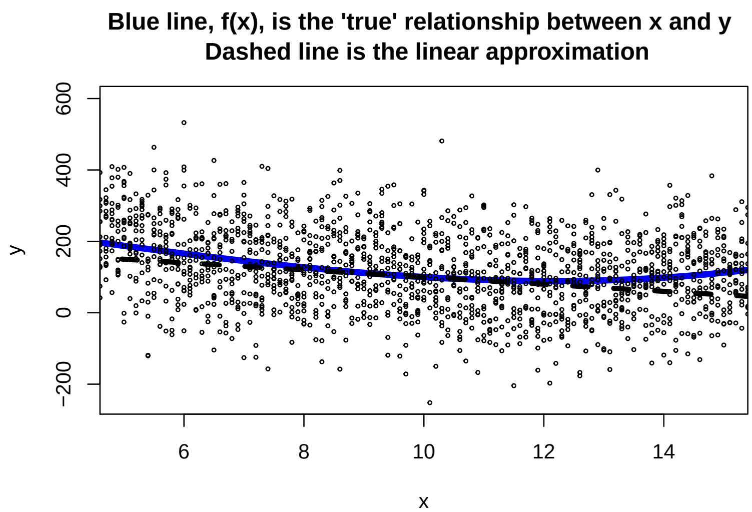
$$g(y_1, ..., y_n|x_1, ..., x_n) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}(x_i)) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

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.

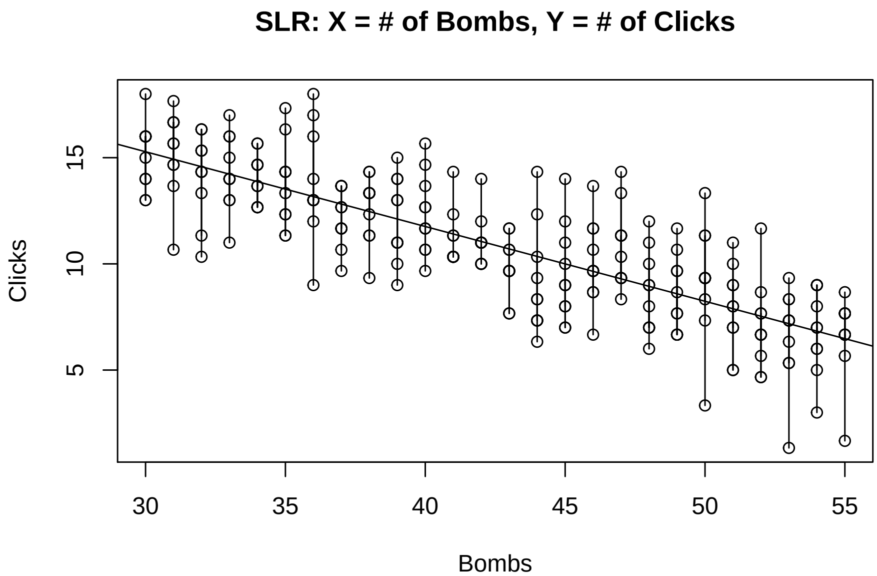
Linear Regression Model

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!

```r
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.

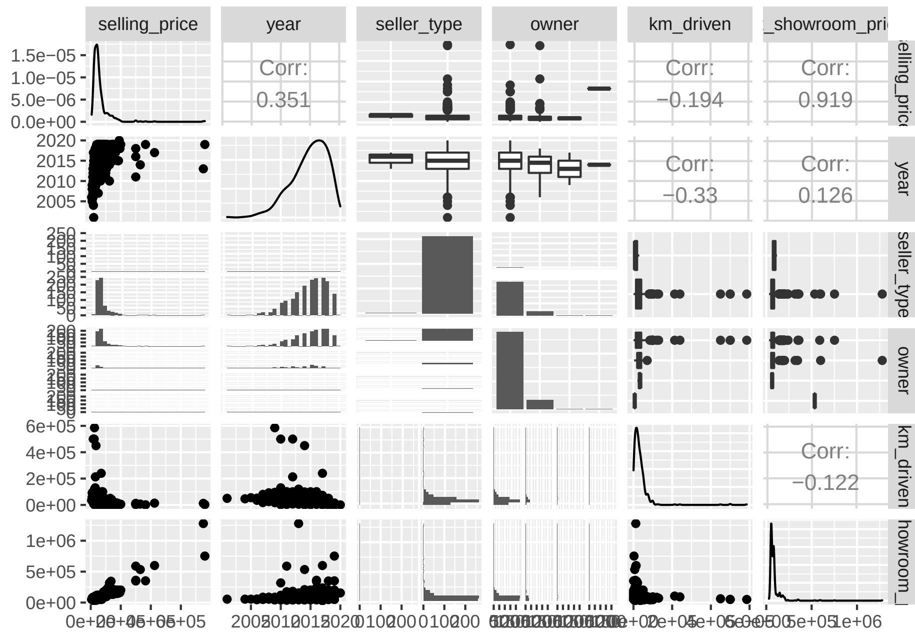
```
summary(bikeData)
```

```
##      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
```

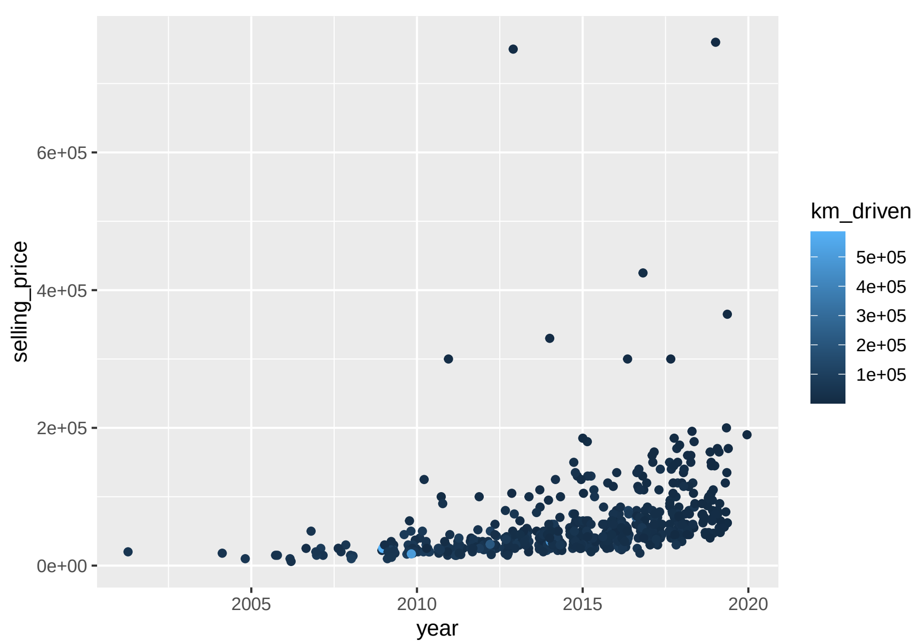
```
summarize(group_by(bikeData, owner),
         mean = mean(selling_price),
         median = median(selling_price),
         sd = sd(selling_price),
         IQR = IQR(selling_price))
```

```
## # 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
```

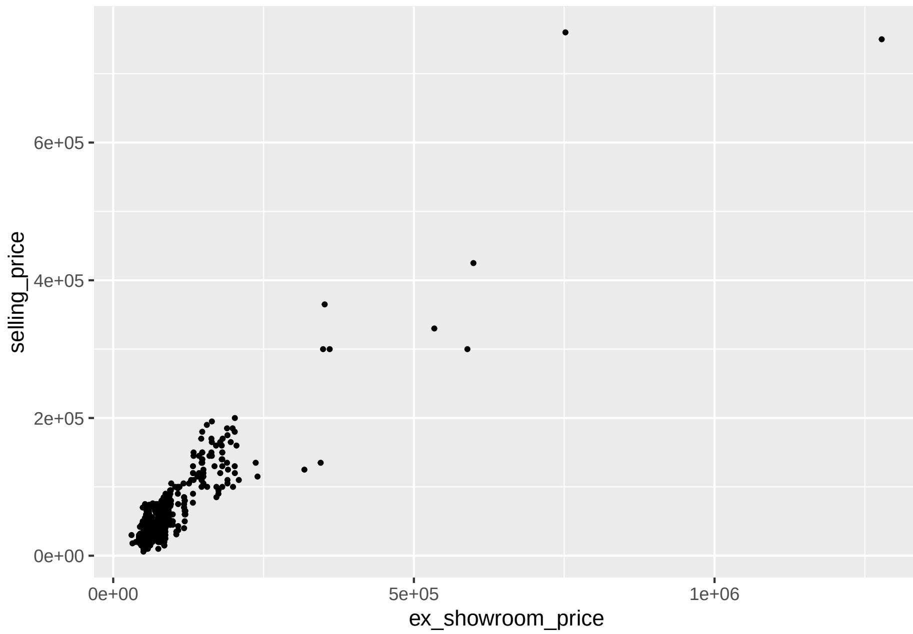
```
library(GGally)
ggpairs(select(bikeData, -name))
```



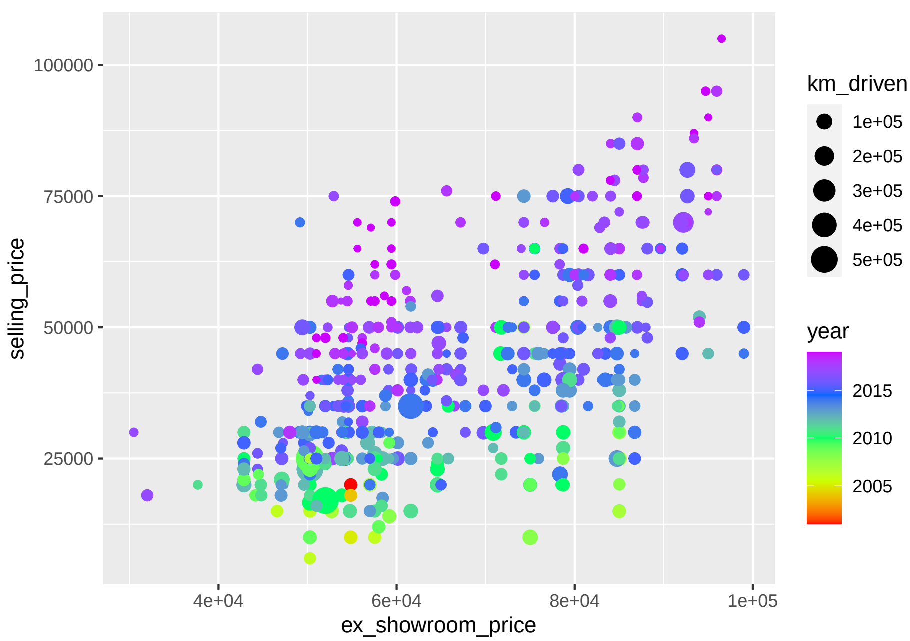

```
g <- ggplot(data = bikeData, aes(y = selling_price))
g + geom_jitter(aes(x = year, color = km_driven))
```




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




```r
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
```

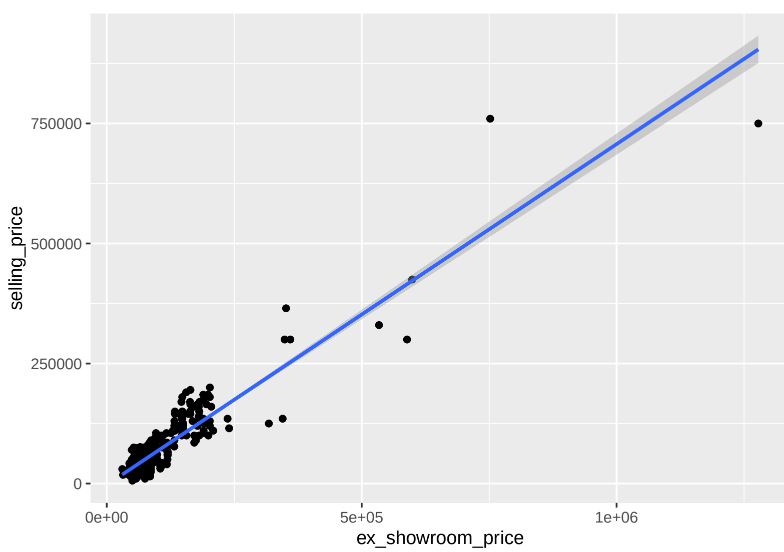
Manually predict for an `ex_showroom_price` of 50000:

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

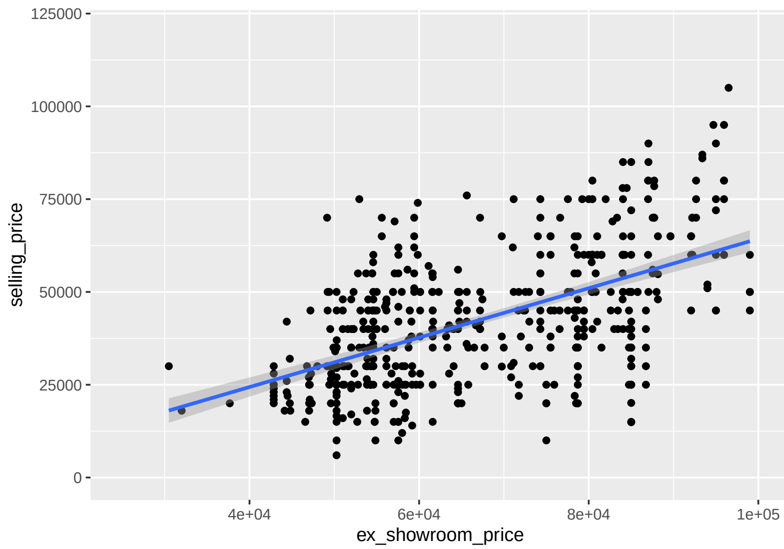
```
## (Intercept)
##    32492.24
```

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.

- 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

```r
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

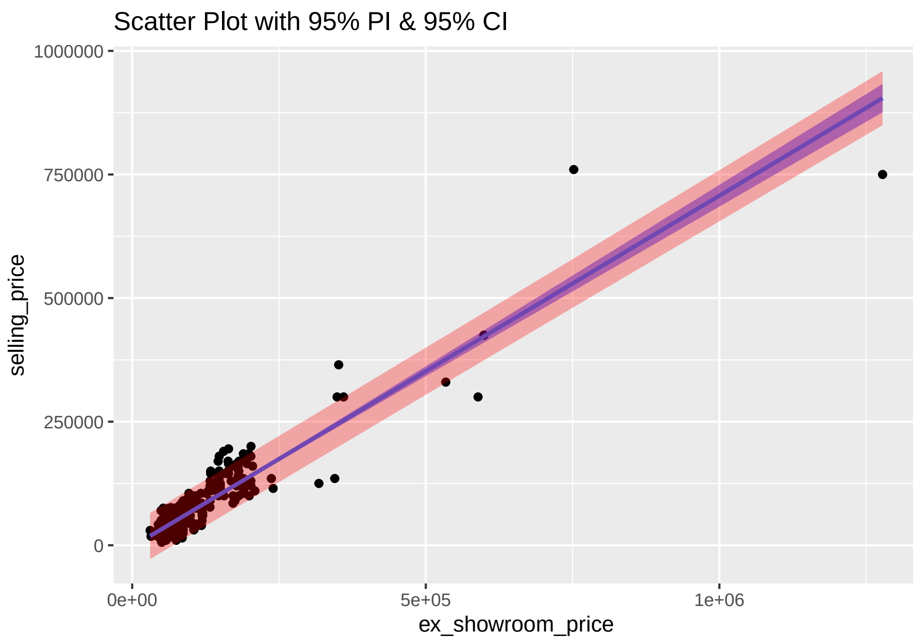
```r
predict(fit, newdata = data.frame(ex_showroom_price = c(50000, 75000, 100000)),
        se.fit = TRUE, interval = "prediction")
```

```
## $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:

```r
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!

```
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
```

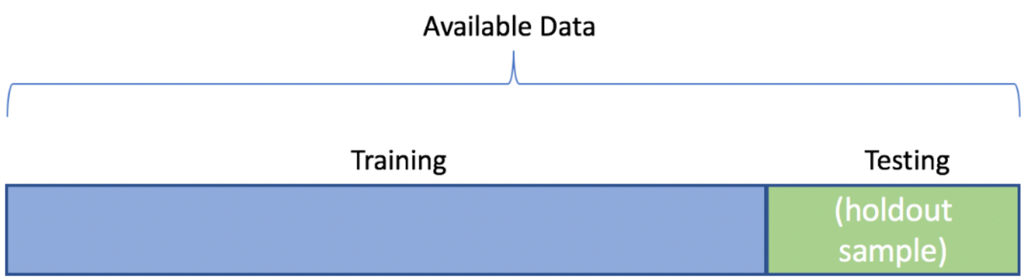
Difficult to visualize the model fit though!

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.

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.

```
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.

```
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 100749. 113099.               150000
##  2  59739.  60915.                65000
##  3  58390.  72928.                78500
##  4  39035.  45467.                50000
##  5  52073.  53538.                35000
##  6  64166.  78343.                80000
##  7  79576.  57387.                40000
##  8 100749. 112955.               150000
##  9  89924. 102497.               120000
## 10  36247.  25302.                25000
## # ... 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] 27840.6
```

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

```
## [1] 23374.61
```

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 $f(x)$???

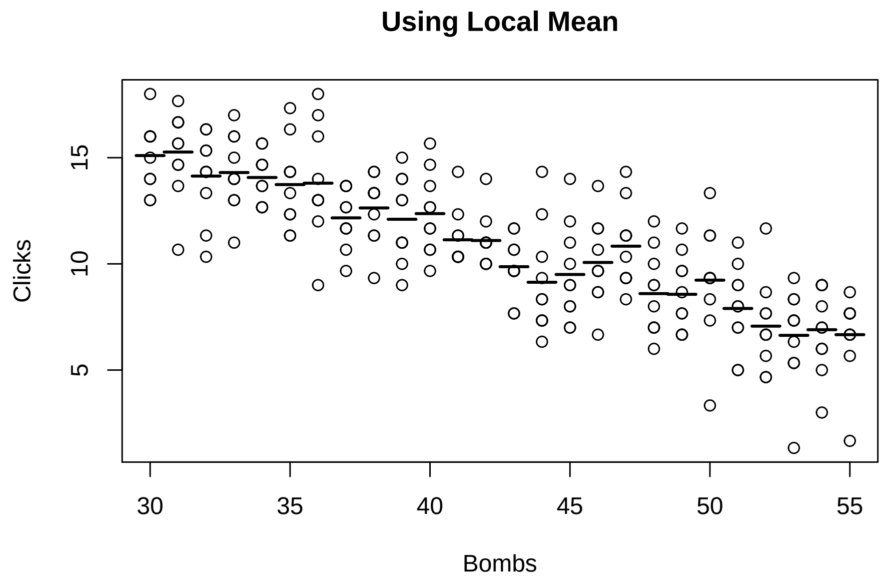
Recall the minesweeper data we collected. In your homework, you were asked to determine a prediction of the number of blocks you could click before hitting a bomb for **your** given number of bombs. You were each creating your estimate of $f(x)$ for your x value!

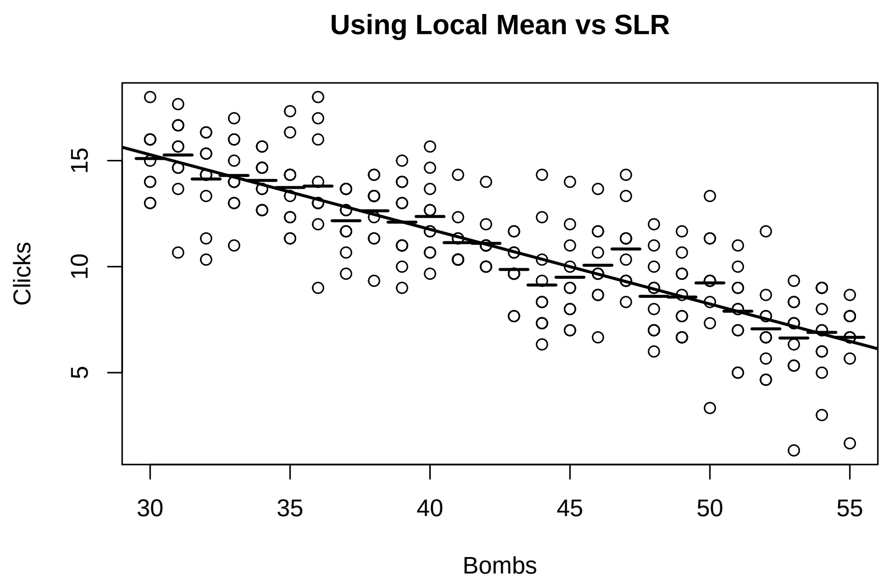
Let's visualize that idea and compare it to the SLR fit!

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

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # 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. **Link app here**

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.

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

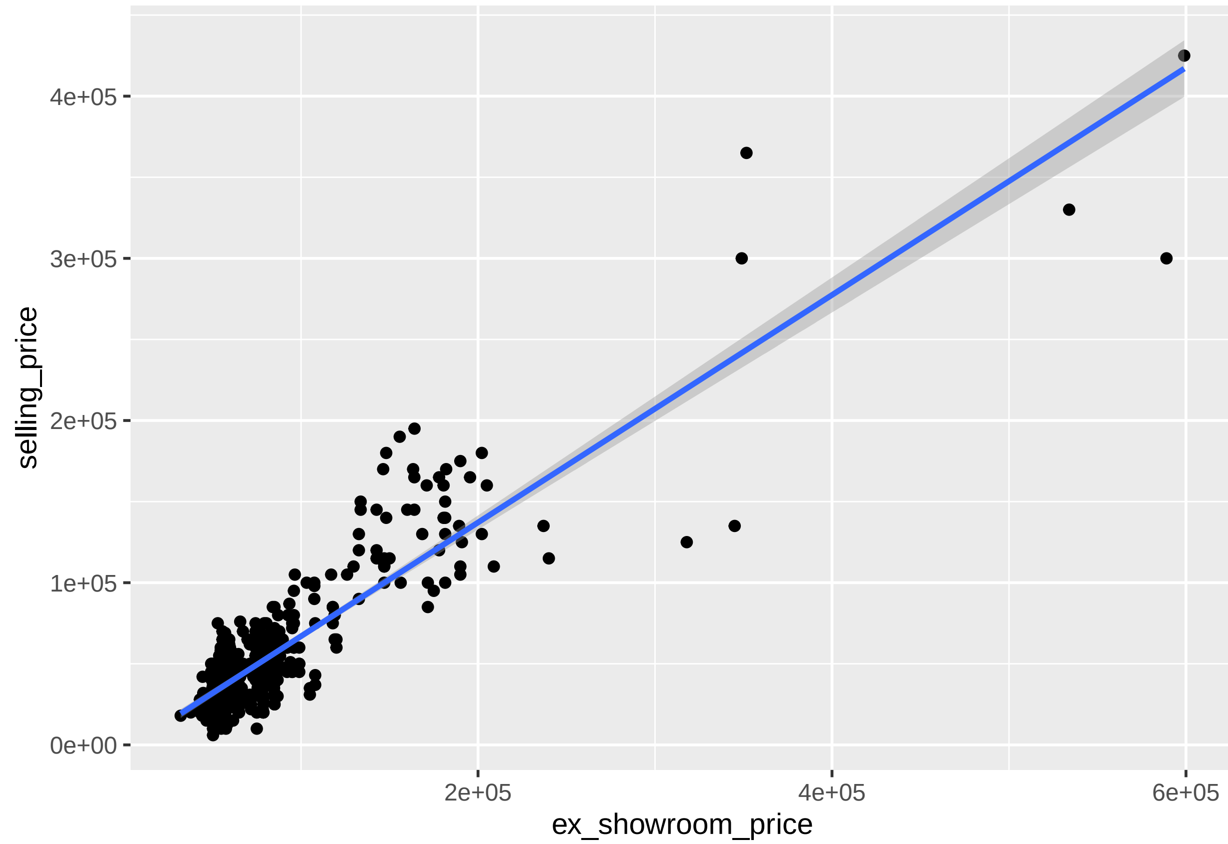
```
##
## Call:
## lm(formula = selling_price ~ ex_showroom_price, data = train)
##
## Coefficients:
##       (Intercept)  ex_showroom_price
##        -3132.4598             0.7014
```

```r
summary(fitSLR)
```

```
##
## Call:
## lm(formula = selling_price ~ ex_showroom_price, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -109968  -11974   -1779   10033  121479
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -3.132e+03  1.804e+03  -1.736   0.0832 .
## ex_showroom_price  7.014e-01  1.717e-02  40.849   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21700 on 436 degrees of freedom
## Multiple R-squared:  0.7928, Adjusted R-squared:  0.7924
## F-statistic:  1669 on 1 and 436 DF,  p-value: < 2.2e-16
```

```r
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")
```

```
## Warning: package 'caret' was built under R version 3.5.3
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

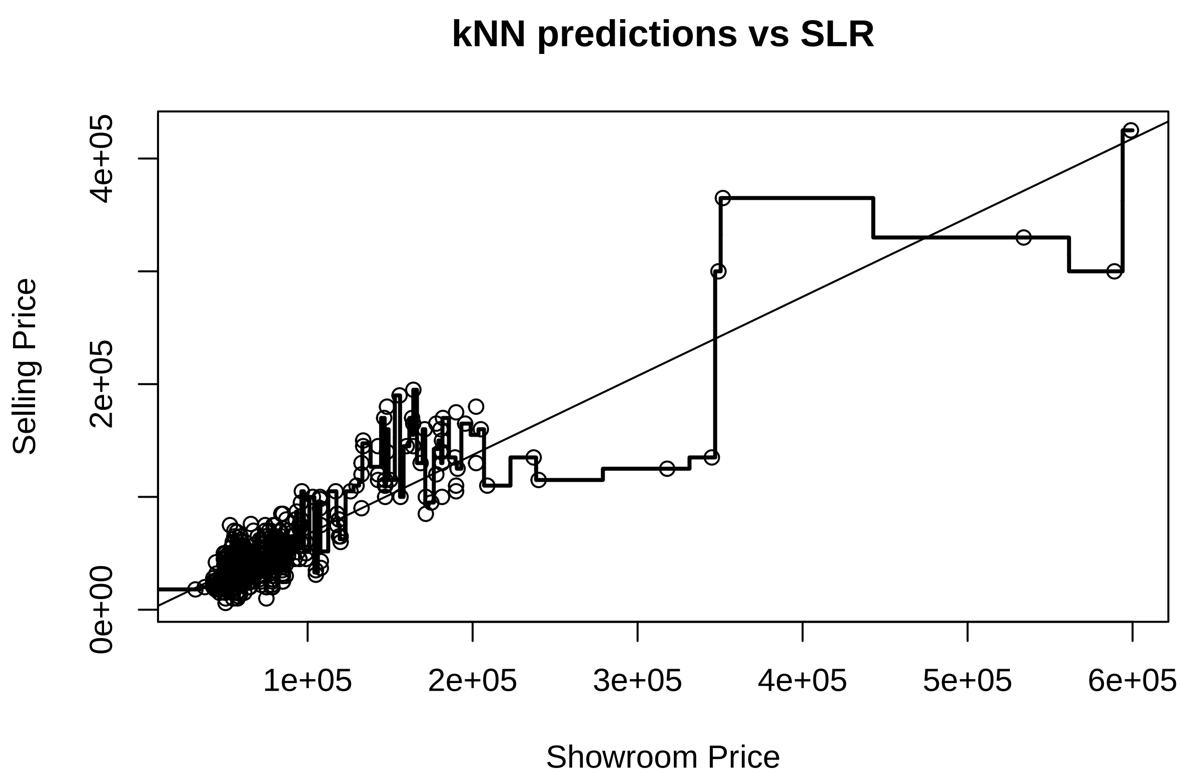
```
## The following object is masked from 'package:purrr':
##
##     lift
```

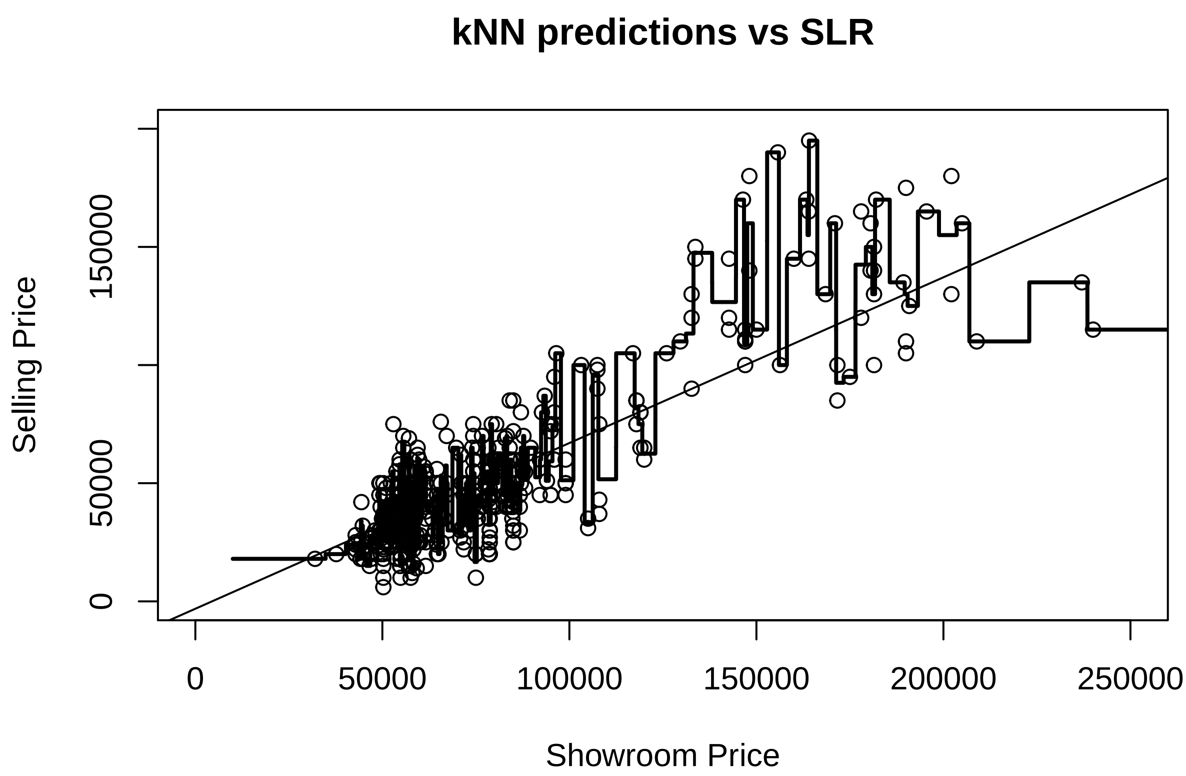
```
k <- 1
kNNFit1 <- train(selling_price ~ ex_showroom_price,
      data = train,
      method = "knn",
      tuneGrid = data.frame(k =k)
      )
kNNFit1
```

```
## 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
##   23995.06  0.7606492  15197.3
##
## Tuning parameter 'k' was held constant at a value of 1
```

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



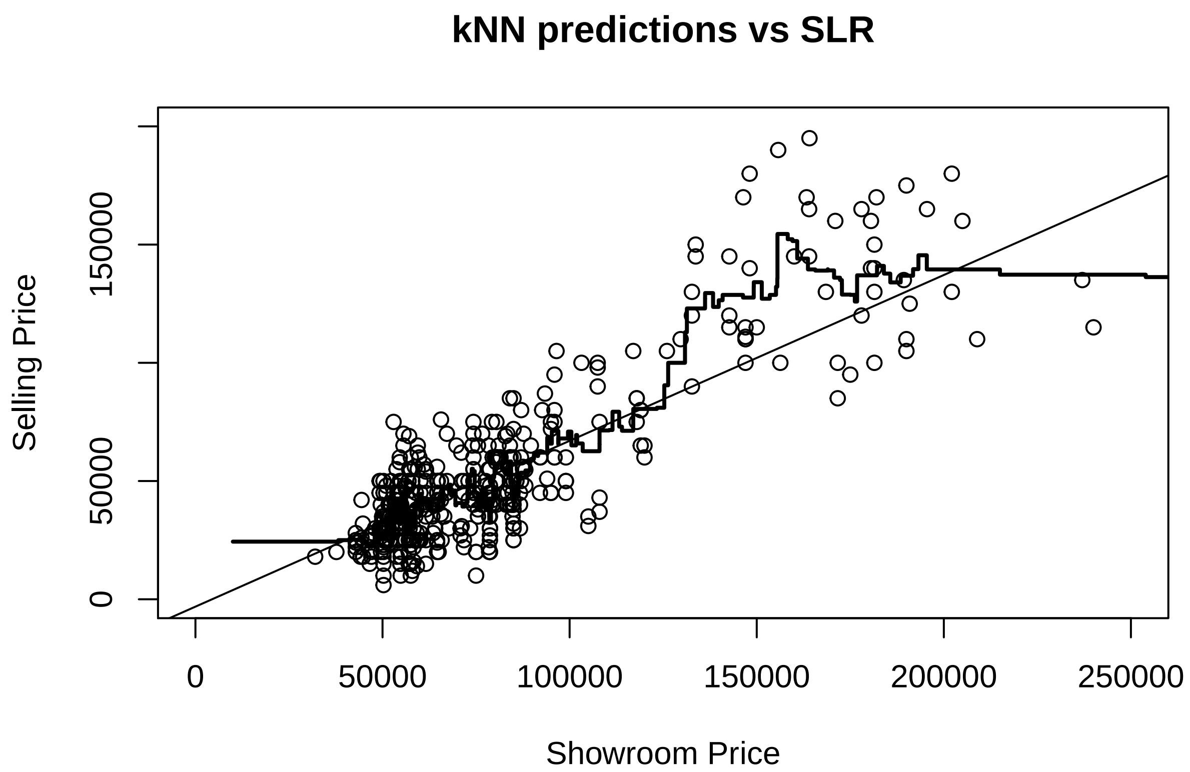


$k = 10$:

```
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
##   27190.57  0.7274929  16045.91
##
## Tuning parameter 'k' was held constant at a value of 10
```

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

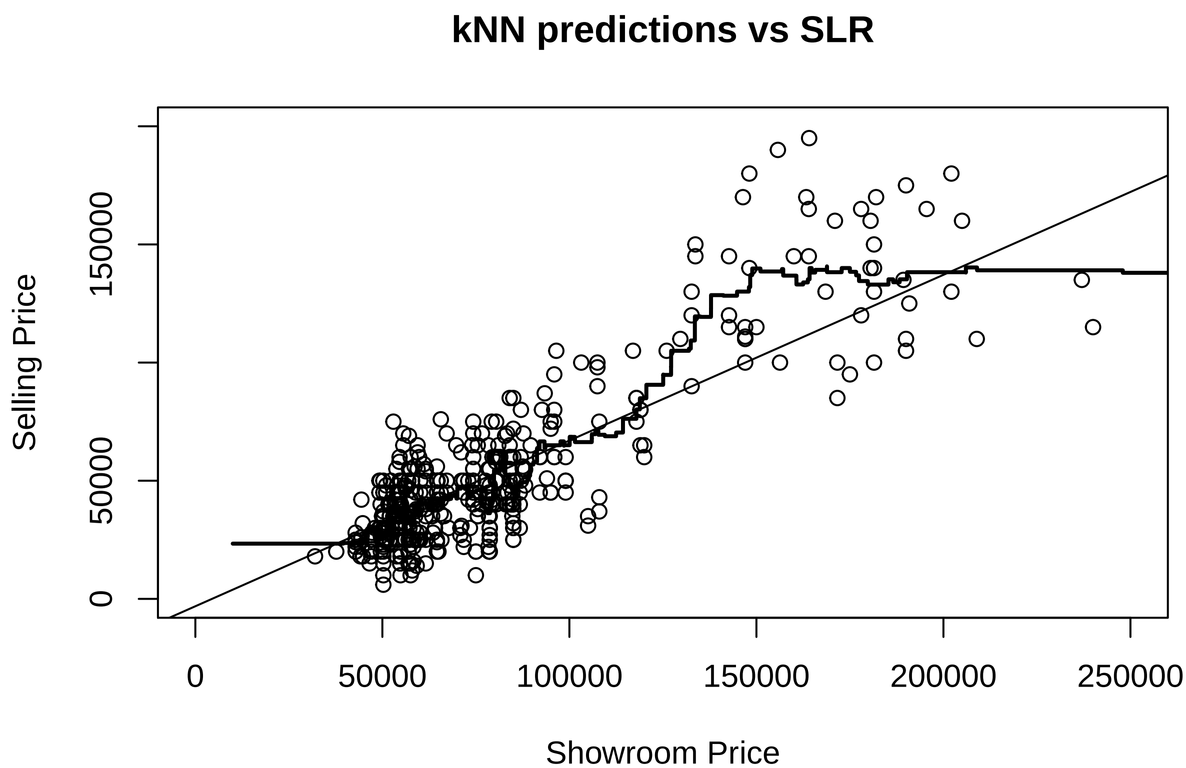


$k = 20$:

```
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
##   24376.62  0.7327945  14873.03
##
## Tuning parameter 'k' was held constant at a value of 20
```

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

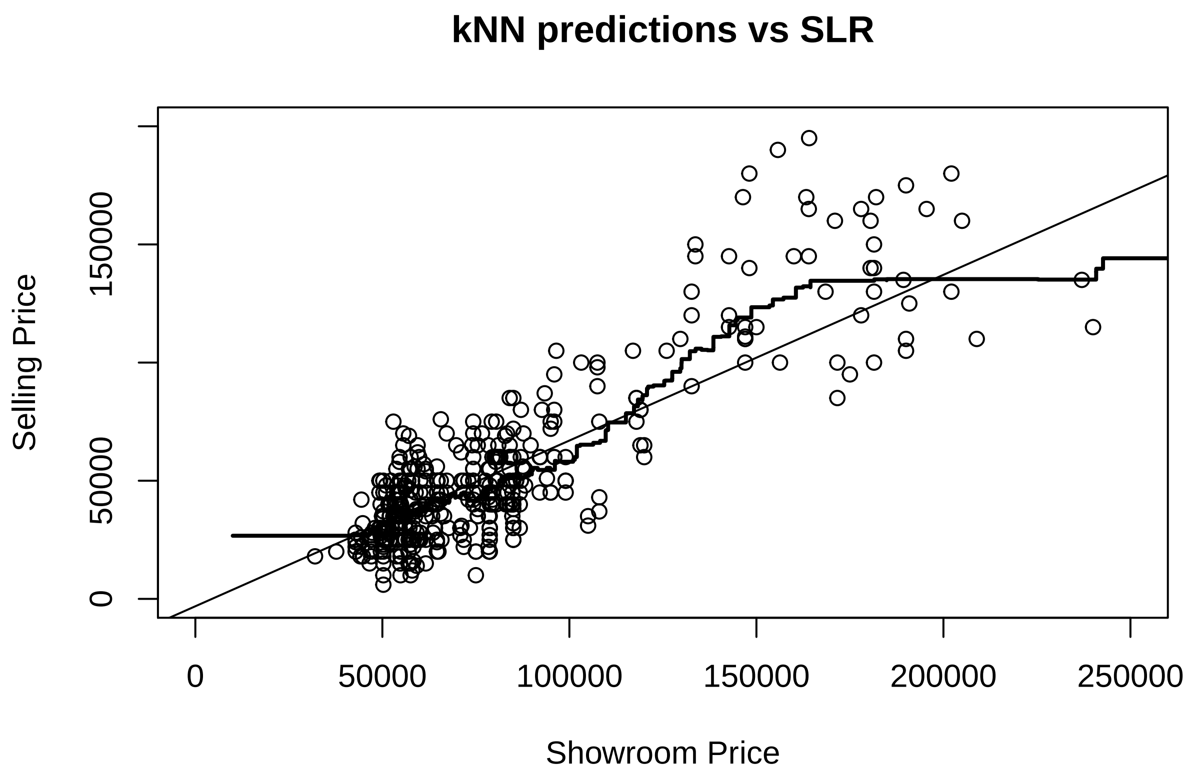


$k = 50$:

```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
##   25970.85  0.7269515  15167.21
##
## Tuning parameter 'k' was held constant at a value of 50
```

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

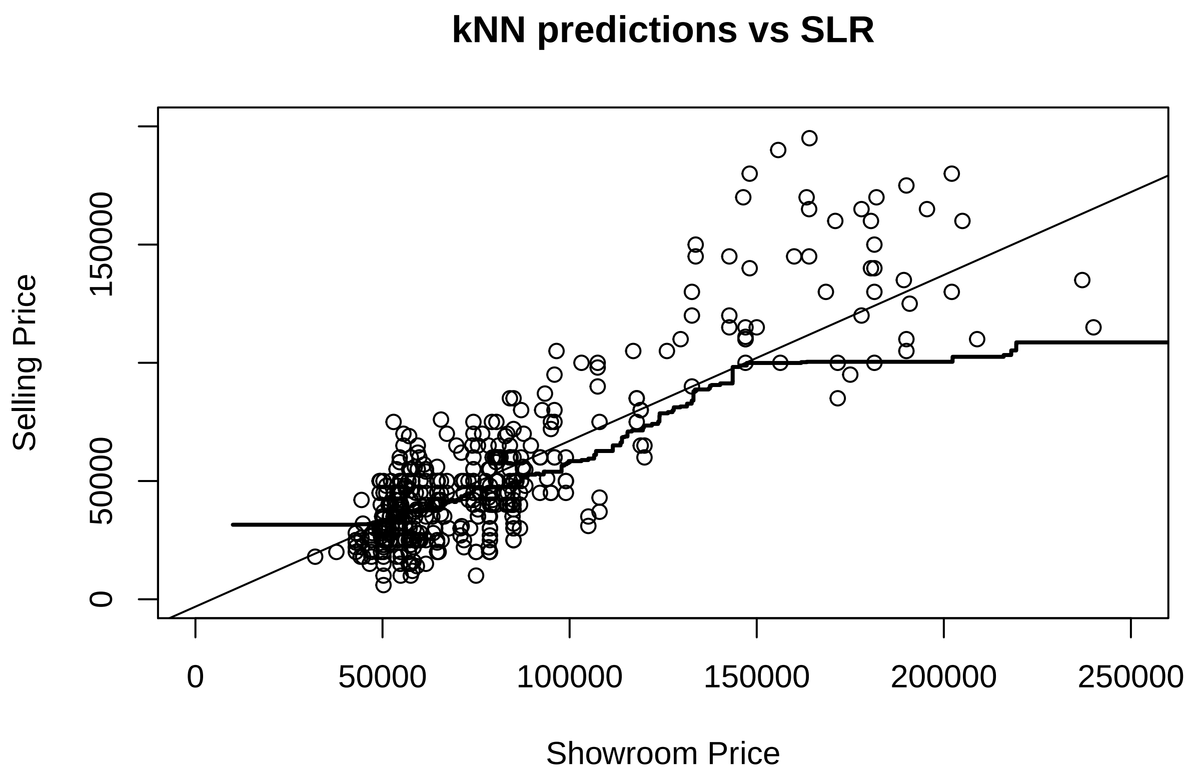


$k = 100$:

```r
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
##   32864.01  0.7052464  17514.27
##
## 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 27840.60
## 2   kNN1 38426.66
## 3  kNN10 57192.25
## 4  kNN20 61807.92
## 5  kNN50 65410.10
## 6 kNN100 70511.86
```

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:

```
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, 395, 395, 394, 395, 394, ...
## Resampling results across tuning parameters:
##
##   k    RMSE      Rsquared   MAE
##     1  21791.19  0.8255668  14210.50
##     2  20782.31  0.8191957  13937.01
##     3  20995.61  0.8143527  13817.78
##     4  20749.55  0.8173147  13758.75
##     5  21038.07  0.8142272  14035.86
##     6  21510.43  0.8097114  14467.62
##     7  22098.16  0.8042000  14627.56
##     8  22504.33  0.8005286  14669.61
##     9  22877.16  0.7966801  14785.62
##    10  23200.59  0.7898105  14802.47
##    11  23582.20  0.7827674  14941.16
##    12  23537.03  0.7831586  14919.37
##    13  23592.47  0.7837519  14842.92
##    14  23736.89  0.7818278  14865.71
##    15  24103.96  0.7746132  14924.28
##    16  24088.63  0.7759797  14859.08
##    17  24185.28  0.7746521  14922.38
##    18  24247.40  0.7735124  14975.27
##    19  24354.65  0.7717801  15033.22
##    20  24590.25  0.7676895  15121.85
##    21  24593.15  0.7672238  15079.50
##    22  24631.29  0.7653932  15041.76
##    23  24735.65  0.7629428  15007.55
##    24  24768.37  0.7628850  14961.07
##    25  24728.38  0.7643760  14913.60
##    26  24720.40  0.7647586  14963.02
##    27  24648.75  0.7682739  14921.69
##    28  24733.92  0.7668029  14992.23
##    29  24768.96  0.7667417  14926.99
```

```
## 30 24790.60 0.7668187 15002.75
## 31 24794.68 0.7682305 14984.37
## 32 24813.57 0.7698235 14965.24
## 33 24903.82 0.7696879 15054.05
## 34 24881.14 0.7704787 15121.08
## 35 24936.45 0.7691981 15131.56
## 36 24970.41 0.7686061 15129.71
## 37 24995.03 0.7676471 15134.50
## 38 25022.80 0.7677180 15114.04
## 39 25175.24 0.7651497 15151.41
## 40 25213.43 0.7645103 15189.17
## 41 25262.68 0.7638752 15217.27
## 42 25336.92 0.7636167 15266.03
## 43 25398.05 0.7630132 15309.07
## 44 25386.27 0.7637307 15270.05
## 45 25431.92 0.7642146 15291.61
## 46 25520.34 0.7639196 15352.53
## 47 25530.97 0.7666179 15284.97
## 48 25603.50 0.7667640 15266.34
## 49 25617.05 0.7691948 15294.00
## 50 25762.90 0.7674965 15369.59
## 51 25825.95 0.7691776 15394.02
## 52 25934.43 0.7688300 15396.09
## 53 26058.52 0.7680946 15446.35
## 54 26246.79 0.7653493 15457.64
## 55 26372.74 0.7658096 15510.02
## 56 26444.81 0.7653888 15533.50
## 57 26659.06 0.7635288 15646.53
## 58 26626.82 0.7665449 15610.01
## 59 26793.17 0.7644656 15664.50
## 60 26918.21 0.7655529 15729.03
## 61 27039.12 0.7645784 15750.49
## 62 27101.31 0.7659216 15779.22
## 63 27327.46 0.7655766 15898.79
## 64 27364.95 0.7683210 15892.68
## 65 27405.19 0.7704313 15890.13
## 66 27581.50 0.7681817 15982.45
## 67 27769.85 0.7673199 16093.19
## 68 27908.39 0.7671792 16148.90
## 69 28048.51 0.7637428 16188.47
## 70 28142.57 0.7636680 16232.98
## 71 28283.13 0.7619680 16294.08
## 72 28381.09 0.7634655 16324.80
## 73 28457.10 0.7649634 16385.45
## 74 28564.43 0.7647628 16447.17
## 75 28728.52 0.7612627 16506.13
## 76 28829.53 0.7619253 16579.31
## 77 28933.02 0.7621430 16672.18
## 78 29004.64 0.7622336 16710.35
## 79 29180.24 0.7611944 16772.38
## 80 29303.24 0.7610363 16824.51
## 81 29412.15 0.7608901 16889.26
## 82 29494.77 0.7614794 16927.23
## 83 29605.61 0.7612174 16966.20
```

```
##    84  29691.34  0.7616438  17030.23
##    85  29825.79  0.7595197  17084.52
##    86  29967.33  0.7591394  17160.67
##    87  30082.33  0.7596716  17220.50
##    88  30237.84  0.7579103  17312.90
##    89  30345.25  0.7580539  17374.06
##    90  30622.68  0.7584970  17519.13
##    91  30762.00  0.7586696  17609.22
##    92  30978.05  0.7573555  17721.23
##    93  31102.51  0.7575753  17796.24
##    94  31296.32  0.7581402  17934.13
##    95  31544.28  0.7573465  18033.62
##    96  31710.51  0.7579952  18127.32
##    97  31827.47  0.7552275  18141.07
##    98  31908.02  0.7541961  18164.55
##    99  32002.40  0.7526284  18200.99
##   100  32132.82  0.7500400  18254.19
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 4.
```

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
## 4.504931e+04 8.224044e-01 1.768072e+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: 395, 394, 393, 394, 395, 394, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   21397.87  0.8006818  15128.29
##
## 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
```

```
## 2.784060e+04 8.845011e-01 1.689184e+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:

```r
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, 394, 396, 394, 394, 395, ...
## Resampling results across tuning parameters:
##
##   k    RMSE      Rsquared   MAE
##     1  23640.50  0.7573258  16460.14
##     2  21836.42  0.7857472  14837.83
##     3  21019.79  0.7938294  13982.71
##     4  20922.74  0.8033729  13642.88
##     5  21341.90  0.8057949  13772.31
##     6  21567.20  0.8101463  13833.95
##     7  21779.42  0.8111850  13944.40
##     8  21996.73  0.8120830  13937.33
##     9  22049.90  0.8107130  13867.25
##    10  21997.21  0.8132944  13782.32
##    11  22144.41  0.8109825  13807.93
##    12  22260.30  0.8090794  13795.21
##    13  22320.04  0.8104700  13740.63
##    14  22409.88  0.8108319  13750.92
##    15  22582.47  0.8074337  13847.22
##    16  22587.63  0.8073978  13802.44
##    17  22495.87  0.8096101  13765.12
```

```
## 18  22502.31  0.8097275  13758.29
## 19  22635.53  0.8080655  13750.78
## 20  22703.92  0.8063917  13735.66
## 21  22734.59  0.8063965  13705.90
## 22  22780.80  0.8055256  13706.06
## 23  22786.35  0.8063212  13692.50
## 24  22901.19  0.8046647  13756.52
## 25  23064.28  0.8030037  13840.15
## 26  22945.98  0.8057186  13810.70
## 27  23050.27  0.8033385  13845.49
## 28  23095.08  0.8033021  13836.57
## 29  23065.82  0.8038240  13818.44
## 30  23068.04  0.8043034  13809.20
## 31  22987.24  0.8058844  13817.12
## 32  22949.61  0.8065536  13841.69
## 33  22932.02  0.8076682  13844.54
## 34  22967.88  0.8075008  13866.61
## 35  22986.66  0.8070167  13913.72
## 36  23017.12  0.8068164  13940.93
## 37  23014.74  0.8072177  13935.62
## 38  23053.59  0.8069973  13955.12
## 39  23121.92  0.8058505  13976.88
## 40  23188.98  0.8054436  13985.18
## 41  23242.65  0.8044281  14024.84
## 42  23307.17  0.8038732  14041.14
## 43  23347.23  0.8034174  14046.71
## 44  23400.94  0.8030069  14088.13
## 45  23419.83  0.8037873  14093.62
## 46  23457.56  0.8045055  14118.09
## 47  23534.76  0.8044310  14155.88
## 48  23550.50  0.8055000  14164.25
## 49  23608.15  0.8060852  14190.60
## 50  23680.20  0.8061208  14218.41
## 51  23792.80  0.8055961  14248.96
## 52  23863.67  0.8061005  14237.46
## 53  23941.24  0.8057372  14268.26
## 54  24012.78  0.8064626  14283.52
## 55  24088.04  0.8066189  14314.38
## 56  24223.90  0.8058238  14380.16
## 57  24332.59  0.8057464  14429.05
## 58  24432.58  0.8056225  14483.27
## 59  24471.53  0.8072791  14499.97
## 60  24590.55  0.8072288  14528.39
## 61  24706.51  0.8068890  14549.87
## 62  24824.34  0.8063164  14603.47
## 63  24974.46  0.8056767  14662.43
## 64  25129.51  0.8052308  14721.67
## 65  25210.99  0.8059404  14761.42
## 66  25212.34  0.8079921  14782.19
## 67  25361.68  0.8069921  14852.17
## 68  25522.10  0.8053265  14928.02
## 69  25551.98  0.8074049  14938.67
## 70  25684.36  0.8067109  14999.57
## 71  25801.00  0.8067485  15048.63
```

```
##    72  25923.95  0.8063366  15110.98
##    73  26049.21  0.8058486  15179.51
##    74  26150.98  0.8067401  15223.84
##    75  26287.68  0.8064546  15284.45
##    76  26389.27  0.8070254  15340.69
##    77  26516.86  0.8066690  15397.88
##    78  26623.45  0.8070039  15458.81
##    79  26744.78  0.8069820  15507.65
##    80  26888.79  0.8060954  15576.07
##    81  27036.57  0.8051313  15656.25
##    82  27172.33  0.8047743  15716.55
##    83  27291.07  0.8049043  15774.40
##    84  27401.84  0.8049215  15823.50
##    85  27546.69  0.8047449  15907.97
##    86  27671.29  0.8047395  15966.98
##    87  27798.12  0.8042159  16043.73
##    88  27926.29  0.8039591  16102.04
##    89  28070.78  0.8032271  16188.30
##    90  28197.69  0.8026942  16240.57
##    91  28308.91  0.8023809  16305.39
##    92  28448.75  0.8020104  16380.77
##    93  28545.18  0.8022831  16424.13
##    94  28663.66  0.8023173  16485.83
##    95  28784.19  0.8023103  16538.29
##    96  28914.83  0.8019965  16600.57
##    97  29030.81  0.8015904  16665.41
##    98  29131.86  0.8007397  16711.59
##    99  29247.84  0.8006306  16767.39
##   100  29350.81  0.8008318  16827.80
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 4.
```

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
## 4.474815e+04 8.233754e-01 1.722657e+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: 394, 393, 396, 394, 394, 395, ...
## Resampling results:
##
##   RMSE      Rsquared  MAE
##   17256.27  0.877712  11293.37
##
## 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
## 2.337461e+04 9.215234e-01 1.191222e+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!