Case Study 3 - Creating a Shiny App

Jack Schnair

Introduction

Desired outcome of the case study. In this R markdown file I will analyze and discuss the dataset I used for this case study. I will also create a shiny app that uses linear regression on the dataset. The user will be able to choose what columns in the dataset are the indepnedent and dependent variables.

What data was collected?

The Brickset dataset is a collection of most, if not all, lego sets ever sold. It is updated and maintained by a community of lego collectors. There are 17 columns in the dataset, but the most important are the name, MSRP, number of pieces, and number owned.

Why this dataset?

I have always loved legos, ever since I was a kid, and am excited to be working with lego data. I haven't actually bought a set in years, but it feels nostalgic to be thinking about them. I could use any sort of product sales data for this study, so I might as well use something I like.

Libraries

```r
#install.packages("tidyverse")
library(tidyverse)
```

Import the dataset

```r
legoData = read.csv("sets.csv")
```

Clean the data

- Remove columns with no MSRP (This is over half the data)
  - I only have use for sets with prices
  - As a side effect, this removes most sets before the year 2000
- Replace N/A value with 0 for minifigures
- Filter out sets with less than 20 purchases
- Filter out the outlier sets that cost more than $200

```r
# I must reread the data in due to a knitting error
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
legoData = read.csv("sets.csv")


# This df without the outliers will be used for regression
# Dropping na for pieces also makes it so non-set products are not counted.
df <- legoData %>%
  drop_na(USD_MSRP, Pieces) %>%
  filter(Owned >= 20, USD_MSRP < 200) %>%
  mutate(Minifigures = ifelse(is.na(Minifigures), 0, Minifigures))
```

Analyzing the data

The question most companies are interested in is "what makes us the most money?" So I'm going to mostly look at what makes a profitable lego set. Unfortunately, this data only accounts for collectors that are part of the Brickset community. This means that many of the sets that are meant for children may not be represented.

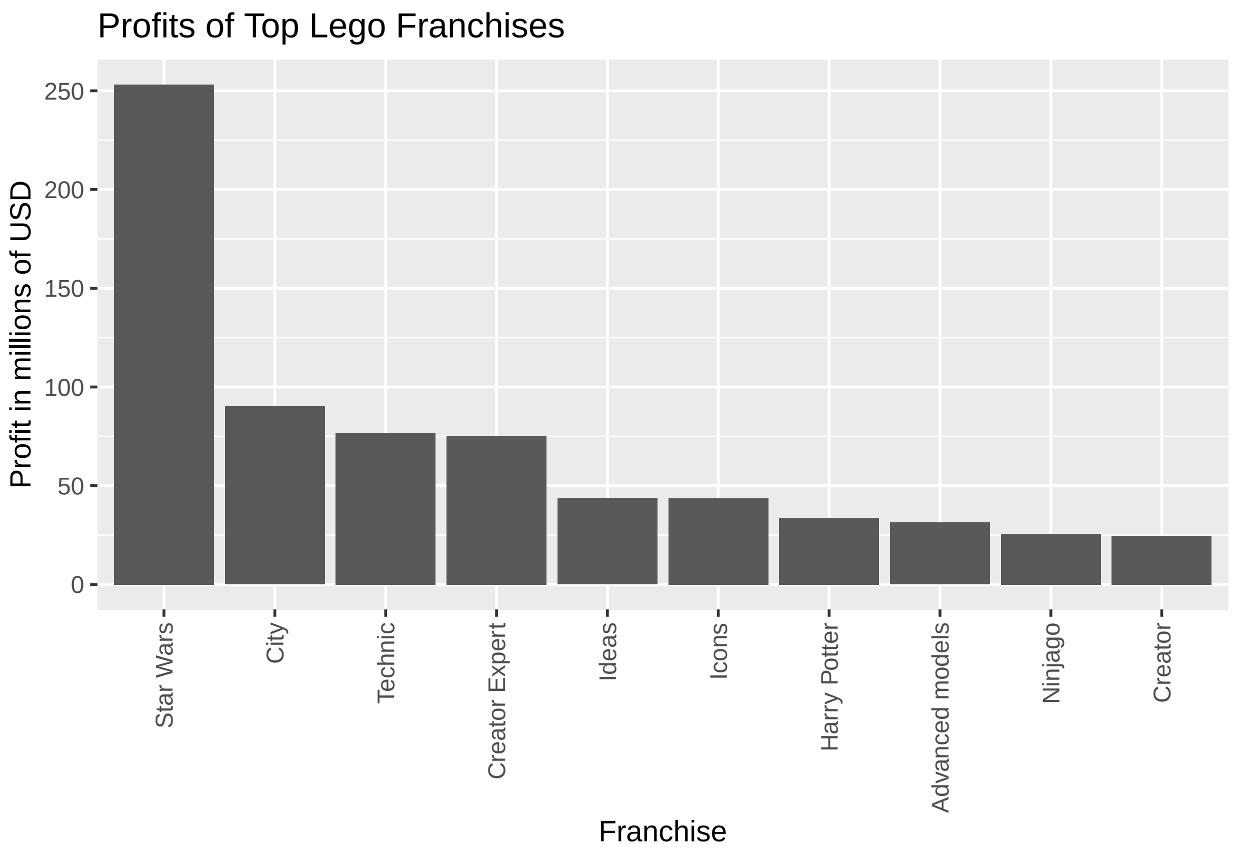
What are the most profitable franchises?

I will find out which lego franchises generate the most profit. This will be based on the quantity purchased and the MSRP of each set.

```
theme_df <- legoData %>%
  drop_na(Owned, USD_MSRP) %>%
  mutate(profit = Owned * USD_MSRP) %>%
  group_by(Theme) %>%
  summarise(count = n(), profit_Millions_USD = sum(profit)/1000000) %>%
  arrange(desc(profit_Millions_USD))

theme_chart = head(theme_df, 10)

head(theme_df, 10)
```

```
## # A tibble: 10 x 3
##    Theme            count profit_Millions_USD
##    <chr>            <int>               <dbl>
##  1 Star Wars          454               253.
##  2 City               527                90.1
##  3 Technic            202                76.9
##  4 Creator Expert      35                75.3
##  5 Ideas               46                43.8
##  6 Icons               41                43.7
##  7 Harry Potter        57                33.7
##  8 Advanced models     27                31.4
##  9 Ninjago            286                25.6
## 10 Creator            224                24.7
```

```
# Make a bar chart of the most profitable franchises

theme_chart %>% ggplot(aes(x=reorder(Theme, -profit_Millions_USD), y=profit_Millions_USD)) +
  theme(axis.text.x = element_text(angle=90, vjust = 0.5, hjust = 1)) +
  geom_bar(stat = "identity") +
  ylab("Profit in millions of USD") + xlab("Franchise") +
  labs(title = "Profits of Top Lego Franchises")
```



Based on this chart, Star Wars towers over every other franchise in profit. Lego seems to understand this, as the total number of Star Wars sets is in the top 3 franchises with the most sets. If Lego, as a company, wants to maximize their profits, they should create more sets from the most popular franchises.

Resale Value

There is data for the USD_MSRP of each set and the current price. I'm curious in which sets or kind of sets have increased in value the most.

```
price_change_df <- legoData %>%
  select(Name, Year, Theme, USD_MSRP, Current_Price) %>%
  drop_na(USD_MSRP, Current_Price) %>%
  mutate(price_change = Current_Price - USD_MSRP) %>%
  arrange(desc(price_change))

percent_change_df <- legoData %>%
  select(Name, Year, Theme, USD_MSRP, Current_Price) %>%
  drop_na(USD_MSRP, Current_Price) %>%
  mutate(percent_change = (Current_Price/USD_MSRP)*100) %>%
  arrange(desc(percent_change))
```

```
head(price_change_df, 10)
```

```
##                                        Name Year           Theme USD_MSRP
## 1                          Market Street 2007 Advanced models    89.99
## 2                         Grand Carousel 2009 Advanced models   249.99
## 3                            Cafe Corner 2007 Advanced models   139.99
## 4                          Death Star II 2005       Star Wars   269.99
## 5                           Green Grocer 2008 Advanced models   149.99
## 6     Republic Dropship with AT-OT Walker 2009       Star Wars   249.99
## 7  Ultimate Collector's Millennium Falcon 2007       Star Wars   499.99
## 8                   Super Star Destroyer  2011       Star Wars   399.99
## 9                       Imperial Flagship 2010 Advanced models   179.99
## 10                      The Mountain Cave 2017       Minecraft   249.99
##    Current_Price price_change
## 1       3099.990     3010.000
## 2       2380.792     2130.802
## 3       1850.000     1710.010
## 4       1800.000     1530.010
## 5       1500.000     1350.010
## 6       1439.511     1189.521
## 7       1645.936     1145.946
## 8       1499.990     1100.000
## 9       1275.374     1095.384
## 10      1300.000     1050.010
```

The largest changes in price are more expected than the percentage increases. Most of the largest price increases are either Star Wars or Advanced Models. The advanced models make sense as a collectors item because that line of legos is marketed towards adults as "serious builds". There appears to be a "golden age" around 2005 - 2010 where expensive lego sets were not largely manufactored. Since then, there has been a collector boom and those rare sets have greatly increased in value.

```r
head(percent_change_df, 10)
```

```
##                          Name Year                   Theme USD_MSRP
## 1                 Cheerleader 2010 Collectable Minifigures     1.99
## 2             Spartan Warrior 2010 Collectable Minifigures     1.99
## 3                       Pilot 2011 Collectable Minifigures     2.99
## 4                  Kimono Girl 2011 Collectable Minifigures     2.99
## 5                   Gladiator 2011 Collectable Minifigures     2.99
## 6               Market Street 2007         Advanced models    89.99
## 7                       Ehlek 2007                Bionicle     9.99
## 8  BARC Speeder with Sidecar 2013               Star Wars    24.99
## 9                   Toa Jaller 2007                Bionicle     9.99
## 10                  Toa Nuparu 2007                Bionicle     9.99
##    Current_Price percent_change
## 1       365.0000      18341.709
## 2       165.1400       8298.492
## 3       153.4500       5132.107
## 4       138.6000       4635.452
## 5       125.0000       4180.602
## 6      3099.9900       3444.816
## 7       200.0000       2002.002
## 8       480.0758       1921.072
## 9       175.0000       1751.752
## 10      175.0000       1751.752
```

The largest percentage increases are largely "Collectible Minifigures". If a person bought some of these Minifigures for 2 or 3 dollars each when they were released, they would be making profits of 4000% - 18300%. The change in price isn't as large as some of the higher value sets, but since the price of investment was so low, the increase in value is much higher.

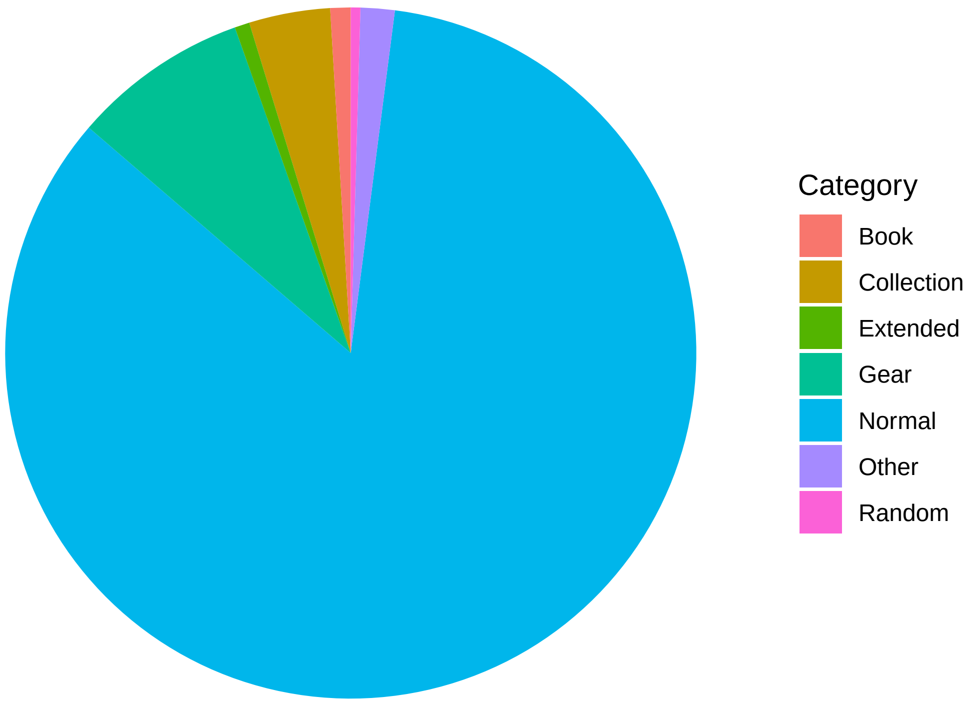
Interestingly, the set “Market Street” is one of the sets that increased most in both flat price and in percentage. I did some research of my own, and I think that used copies of this set still have about a 400% increase in price, but to get the full value, you would have to be holding onto a mint condition copy.

Diversity of Products

```
products_df <- legoData %>%
  drop_na(Category) %>%
  group_by(Category) %>%
  summarise(count = n()) %>%
  mutate(percent = count/nrow(legoData)*100) %>%
  arrange(desc(percent))

products_df
```

```
## # A tibble: 7 x 3
##   Category   count percent
##   <chr>      <int>   <dbl>
## 1 Normal     12586  84.3
## 2 Gear        1232   8.25
## 3 Collection   567   3.80
## 4 Other        238   1.59
## 5 Book         142   0.951
## 6 Extended     104   0.696
## 7 Random        67   0.449
```

```
products_df %>% ggplot(aes(x="", y=count, fill=Category)) +
  geom_bar(stat = "identity", width=1) +
  coord_polar("y", start=0) +
  theme_void()
```



As I expected, the vast majority products are regular sets, but I was surprised to see how large a market there is for lego gear. Looking at the data, I believe gear is defined as products made by lego that aren't actually Legos. For example, gear can be video games, pens, watches, or clothing. It goes to show how much of a cultural impact Legos have had.

Explain the Algorithm - Linear Regression

Regression is an algorithm used to find the relationship between a dependent variable and one or more independent variables. They may also be called response and predictor variables respectively. I will mostly focus on linear regression, which uses just one independent variable. We are not only learning about the outcome (dependent variable), but the individual drivers that cause it. For example, in the lego dataset I can determine what effects the price more, the number of pieces or the number of minifigures. Logistical Regression is also very useful, but does not work well with this dataset. I could try to predict what the theme of the set is, but there is almost no correlation with any other variables that would work.

The outcome may either be continuous or discrete. When the outcomes is continuous, we can predict what it may be. When it is discrete, we can measure the probability of it happening.

For linear regression specifically, the dependent variable must be continuous and the independent variables can either be continuous, discrete, or categorical. Continuous variables can be infinitely many numbers. Discrete variables can be one a finite set of numbers. Categorical variables are non-numberic data such as gender or theme in the case of the lego dataset.

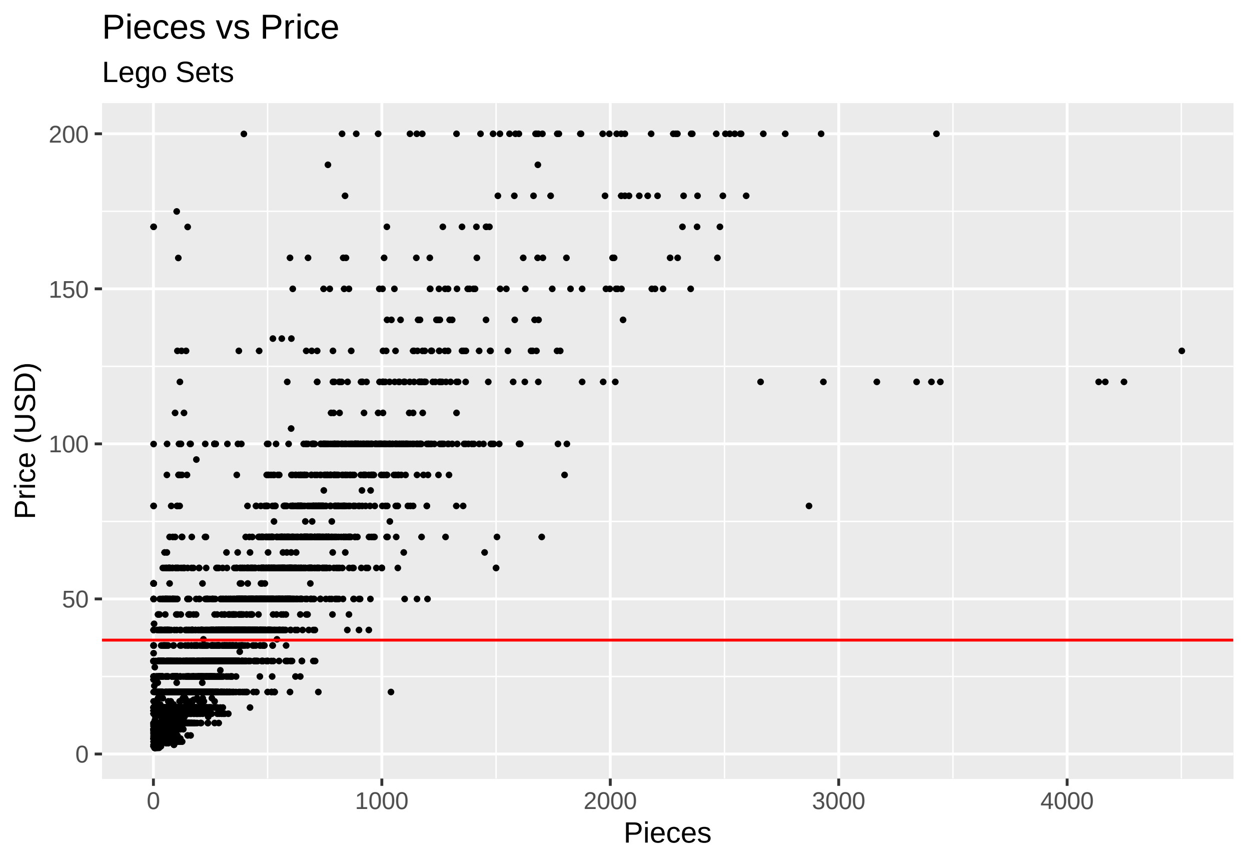
The following is the equation for the linear regression model:

$$\hat{y}_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

The intercept and slope are unknown. The first order of business is to solve for B1 by finding the least squares. Before we do this, we must understand what the sum of squares means.

The sum of squares is a measurement of how well fit a regression line is. The total sum of squares is the measurement of the distance of each point from the mean of the dependent variable squared. I am going to model the price of a lego set based on the number of pieces. The graph would look like this:

```r
avg_price = mean(df$USD_MSRP)

# Scatter plot of pieces vs price
p <- df %>%
  ggplot(aes(x = Pieces, y = USD_MSRP)) +
  geom_point(size = 0.5) +
  labs(title = "Pieces vs Price", subtitle = "Lego Sets") +
  xlab("Pieces") + ylab("Price (USD)")

# Add horizontal mean y line
p1 <- p + geom_hline(yintercept = avg_price, colour = "Red")

p1
```



The red line is the mean price of a lego set, so each square is calculated using the distance of each point from it. This is the calculation in r for the total sum of squares (SST):

```
SST = sum((df$USD_MSRP - mean(df$USD_MSRP))^2)
SST
```

```
## [1] 6637830
```

The total sum of squares is the maximum sum of squares we could possibly have. The goal of linear regression is to minimize the sum of squares. The least squares criterion is represented by the following equation:

$$min \sum(y_i - \hat{y}_i)^2$$

Y is the dependent variable and "Y hat" is the predicted value of the variable. We want the the difference of these two values to be as small as possible. The regression line will be the line that gets this minimum sum of squares.

We can find one point along the regression line by finding the centroid. The centroid is the point where x = mean(x) and y = mean(y). For example:

```
x_val = mean(df$Pieces)
y_val = mean(df$USD_MSRP)
print(paste("Centroid: (",
            round(x_val, digits = 4),
            ", ",
```

```r
            round(y_val, digits = 4),
            ")"))
```

```
## [1] "Centroid: ( 324.0236 ,  36.7238 )"
```

This means that the average number of pieces is about 346 and the average price of a lego set is about 37 dollars.

We only need two points to form a line, so now we just need one more. To get the other, we will need to go back to calculating B1. The equation is the following:

$$b_1 = \Sigma(x_i - \bar{x}_i)(y_i - \bar{y}_i) / \Sigma(x_i - \bar{x}_i)^2$$

The equation is basically taking the deviation of x and y multiplied and dividing it by the deviation of x squared. This is how the slope is found. In this example it's about 0.0765.

The intercept (b0) is found using the following equation:

$$b_0 = \bar{y} - b_1\bar{x}$$

Just plug in the means of x and y and the slope! In this example, the slope is just over 13.1371.

This is how it is done by hand. It's much easier in R, and also more accurate!

Simple Linear Regression

Simply use the function lm with the dependent variable and independent variable separated by a ~ in that order. You can get the fitted values from the fitted() function, the residuals from the resid() function and the summary from the summary function. I will not show them because the data isn't visually appealing on a pdf.

```r
lr = lm(USD_MSRP ~ Pieces, data=df)
lr
```

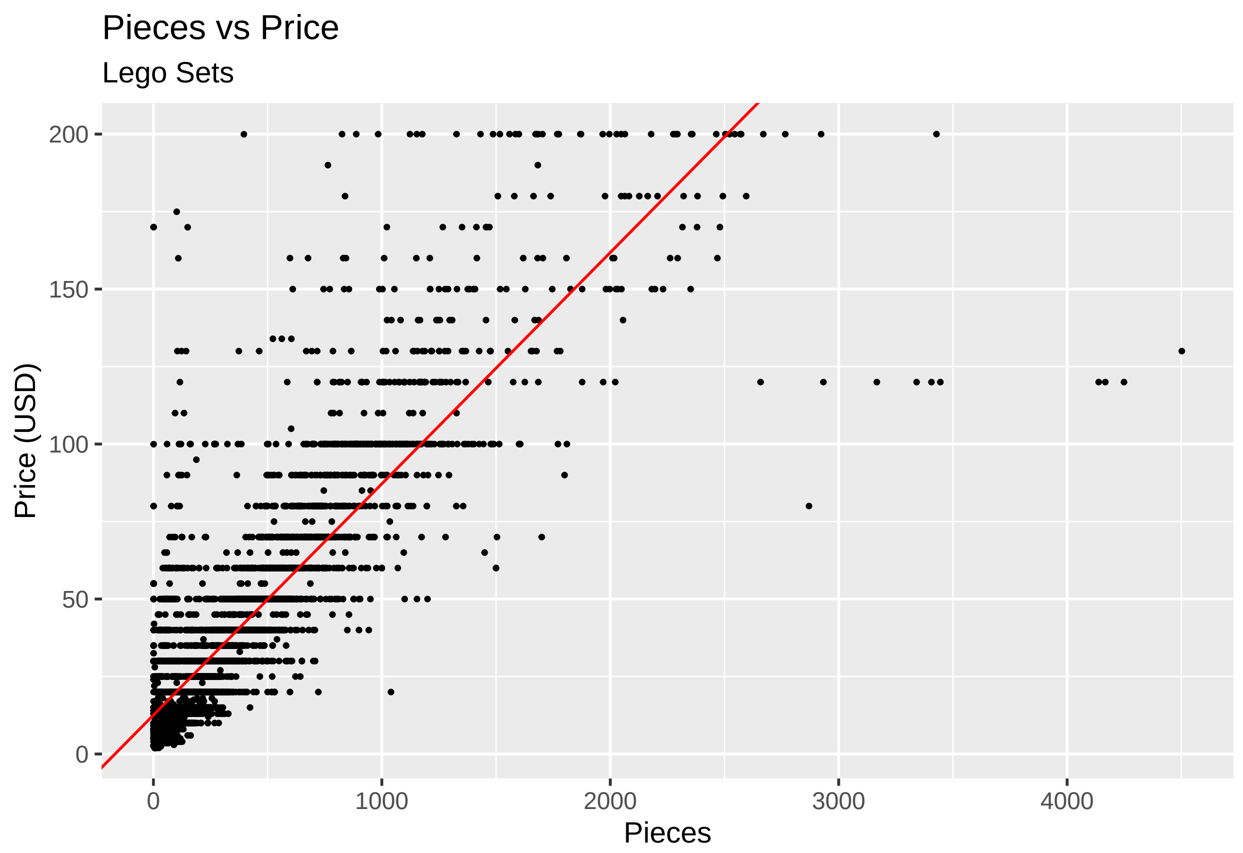
```
##
## Call:
## lm(formula = USD_MSRP ~ Pieces, data = df)
##
## Coefficients:
## (Intercept)       Pieces
##    12.54249      0.07463
```

```r
summary(lr)
```

```
##
## Call:
## lm(formula = USD_MSRP ~ Pieces, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -218.529   -9.418   -4.344    5.358  157.855
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.254e+01  3.424e-01   36.63   <2e-16 ***
## Pieces      7.463e-02  6.527e-04  114.34   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 19.14 on 5050 degrees of freedom
## Multiple R-squared:  0.7214, Adjusted R-squared:  0.7213
## F-statistic: 1.307e+04 on 1 and 5050 DF,  p-value: < 2.2e-16
```

See how easily the intercept and slope can be found with a simple R function! The intercept and slope can be accessed through lr[1]$coefficients[x] where x is 1 and 2 respectively Now let's actually graph the regression line:

```r
simplePlot = p + geom_abline(intercept = lr[1]$coefficients[1],
                             slope = lr[1]$coefficients[2], color="red")
simplePlot
```



There are over 5000 points of data on this graph so it can be hard to evaluate if the regression line is accurate or not. To calculate how accurate it is, we can find the correlation coefficient

```r
cor(df$USD_MSRP, df$Pieces)
```

```
## [1] 0.8493296
```

0.85 is a really strong correlation, so we can be confident about this model. To really see if our model is accurate though, we should train and test it. First I will cover multiple linear regression though.

Mutliple Linear Regression

It's possible to perform linear regression with multiple independent variables. It cannot be done with more than one dependent variable though. In the R programming language, you can simply add variables to the linear model function. For example, do the following to add number of Minifigures as a variable: lm(USD_MSRP ~ Pieces, data=df) -> lm(USD_MSRP ~ Pieces+Minifigures, data=df)

```
lr2 = lm(USD_MSRP ~ Pieces+Minifigures, data=df)
lr2
```

```
##
## Call:
## lm(formula = USD_MSRP ~ Pieces + Minifigures, data = df)
##
## Coefficients:
## (Intercept)       Pieces  Minifigures
##      8.7892       0.0696       2.5455
```

```
summary(lr)
```

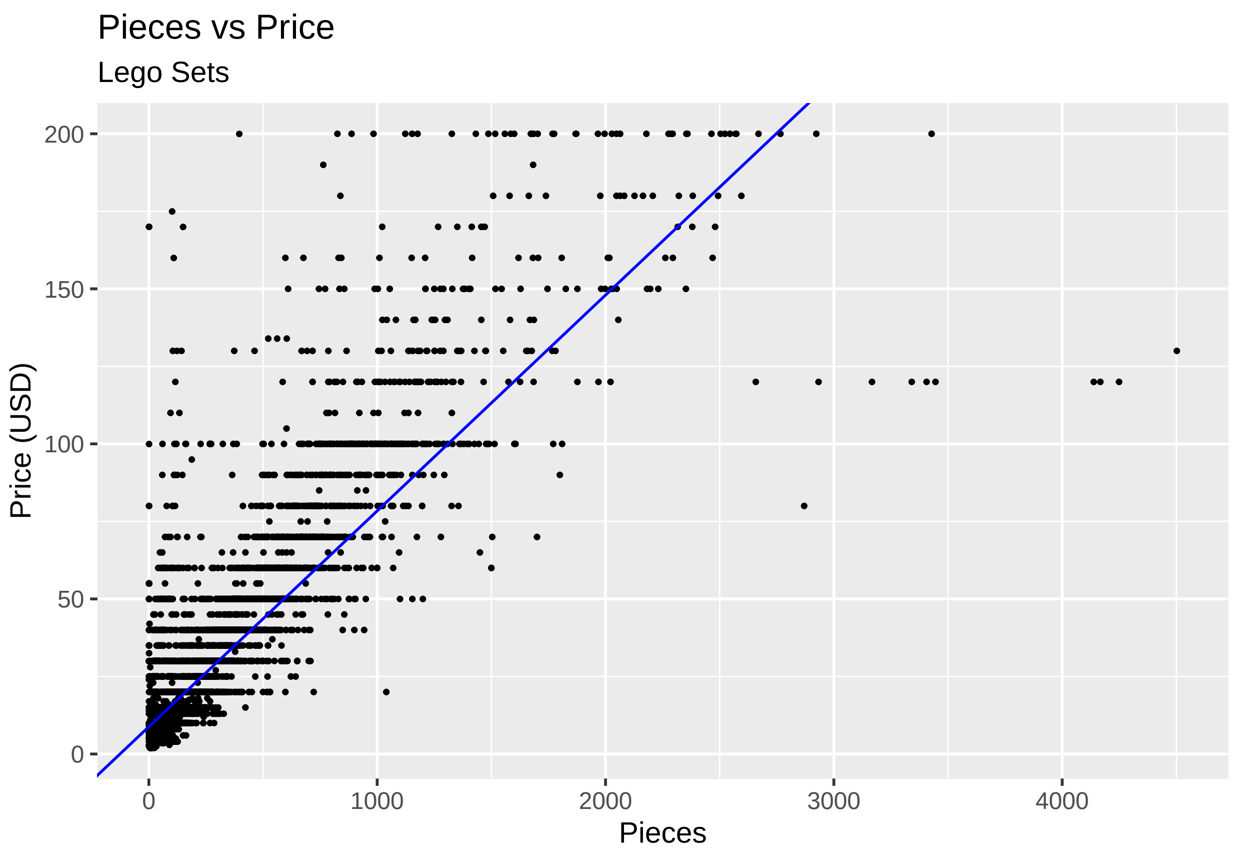
```
##
## Call:
## lm(formula = USD_MSRP ~ Pieces, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -218.529   -9.418   -4.344    5.358  157.855
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.254e+01  3.424e-01   36.63   <2e-16 ***
## Pieces      7.463e-02  6.527e-04  114.34   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 19.14 on 5050 degrees of freedom
## Multiple R-squared:  0.7214, Adjusted R-squared:  0.7213
## F-statistic: 1.307e+04 on 1 and 5050 DF,  p-value: < 2.2e-16
```

You'll notice that the summary has changed for multiple linear regression:

Coefficients: Intercept Pieces Minifigures Simple: 20.74566 0.04744
Multiple: 11.10572 0.04182 5.91149

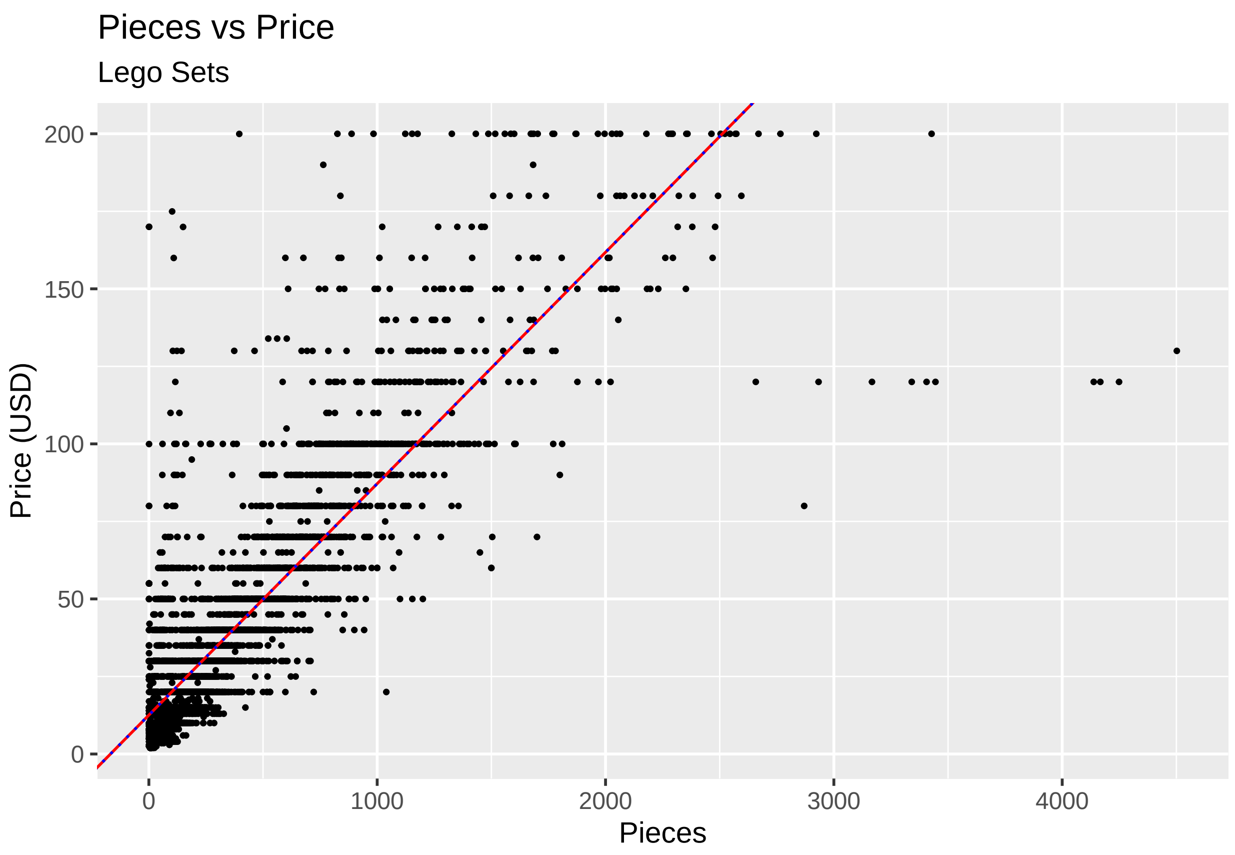
Here is how it looks plotted:

```
multiplePlot = p + geom_abline(intercept = lr2[1]$coefficients[1],
                               slope = lr2[1]$coefficients[2], color="blue")
multiplePlot
```



Here is the original regression line with the multiple regression line plotted over it. They are nearly the same. This is because the number of minifigures in a set has a much lower correlation with the price than the number of pieces does. When they are both considered, the number of pieces will have much more weight in the model.

```
p3 <- p + geom_abline(intercept = lr[1]$coefficients[1],
                        slope = lr[1]$coefficients[2], color="red") +
      geom_abline(intercept = lr[1]$coefficients[1],
                    slope = lr[1]$coefficients[2], color="blue", linetype = "dotted")
p3
```



Training a Model

We need to split the data into train and test data. We will create the model based on the train data and then test it on the test data. I made the test data very small for visibility later on.

```
set.seed(1234)
splitLego = caret::createDataPartition(y = df$Year, p = 0.95, list=F)
trainLego = df[splitLego,]
testLego = df[!row.names(df) %in% row.names(trainLego),]
testLego = df[-splitLego,]
```

We'll create a linear regression line just like we did earlier, but this time for the training data

```
lr = lm(USD_MSRP ~ Pieces, data=trainLego)
lr
```

```
##
## Call:
## lm(formula = USD_MSRP ~ Pieces, data = trainLego)
##
## Coefficients:
## (Intercept)       Pieces
##    12.56347      0.07439
```

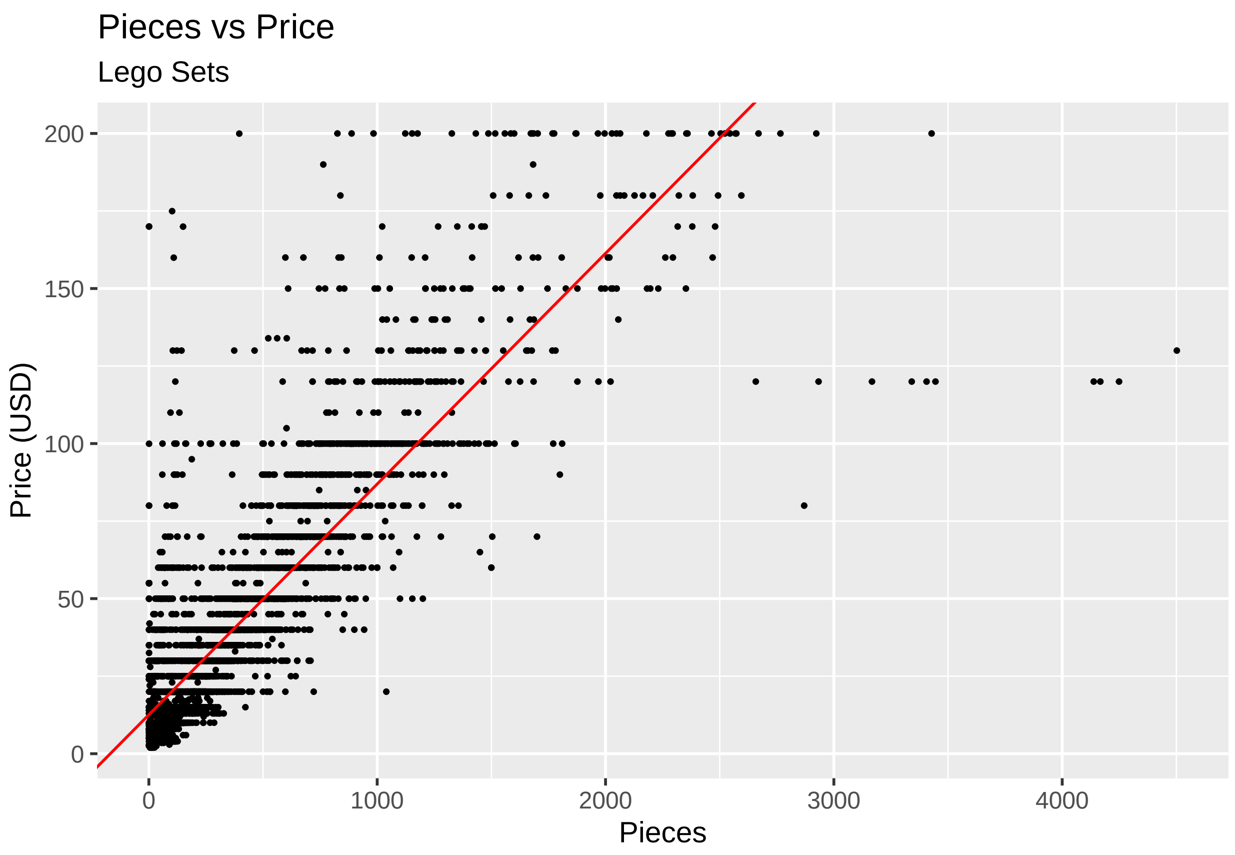
```
summary(lr)
```

```
##
## Call:
## lm(formula = USD_MSRP ~ Pieces, data = trainLego)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -217.466   -9.401   -4.295    5.387  157.352
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.256e+01  3.506e-01   35.84   <2e-16 ***
## Pieces      7.439e-02  6.699e-04  111.05   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 19.12 on 4799 degrees of freedom
## Multiple R-squared:  0.7199, Adjusted R-squared:  0.7198
## F-statistic: 1.233e+04 on 1 and 4799 DF,  p-value: < 2.2e-16
```

Notice that it's slightly different than the original regression line since we're using a subset of the data. Now we can plot this new line

```
p1 = p + geom_abline(intercept = lr[1]$coefficients[1],
                     slope = lr[1]$coefficients[2], color="red")
p1
```



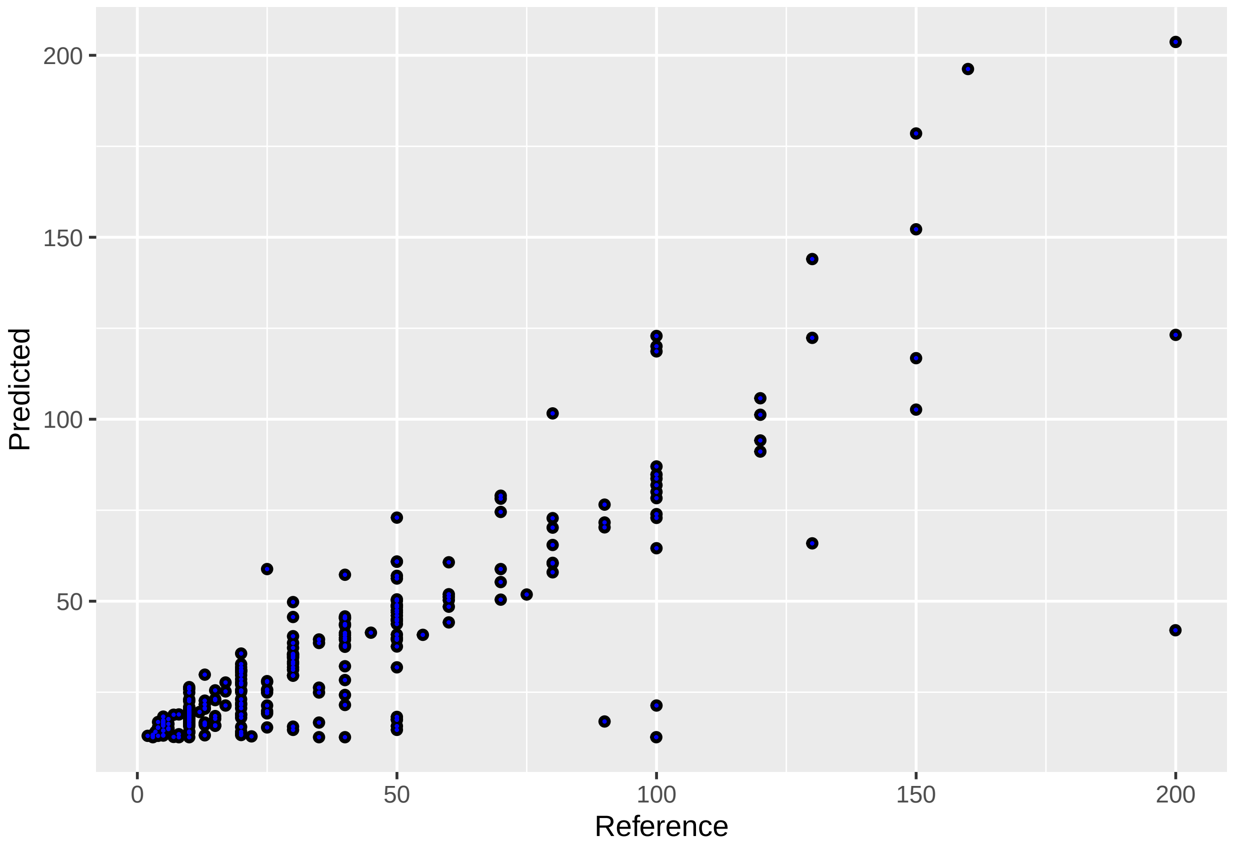
We can now predict the test data using this model. This is what the model predicts that the price of each test set to be.

```
head(predict(lr, newdata=testLego))
```

```
##        12        15        26        36        46        52
## 12.71225 12.63786 14.42316 17.62183 13.15857 17.39866
```

```
predPrice = data.frame(predict(lr, newdata=testLego))
names(predPrice)[1] = 'Predicted'
predPrice$Reference = testLego[,c('USD_MSRP')]
qplot(Reference, Predicted, data=predPrice) + geom_point(colour = "blue", size = 0.1)
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



We can go back and mathematically see if the model has improved by calculating various sums of squares.

Again, the total sum of squares is the worst the model can be and is:

```
SST = sum((df$USD_MSRP - mean(df$USD_MSRP))^2)
SST
```

```
## [1] 6637830
```

First we want the predicted residual sum of squares (PRESS), which is the difference between the predicted results and the actual values.

```
PRESS = sum((predPrice$Reference - predPrice$Predicted)^2)
PRESS
```

```
## [1] 96031.06
```

This is less than 15% of the total sum of squares, so the model has improved!

Next we calculate the root mean sqaured error of prediction (RMSEP), which is the averaege difference between the actual values and the predicted ones.

```
RMSEP = sqrt(PRESS/nrow(df))
RMSEP
```

```
## [1] 4.359877
```

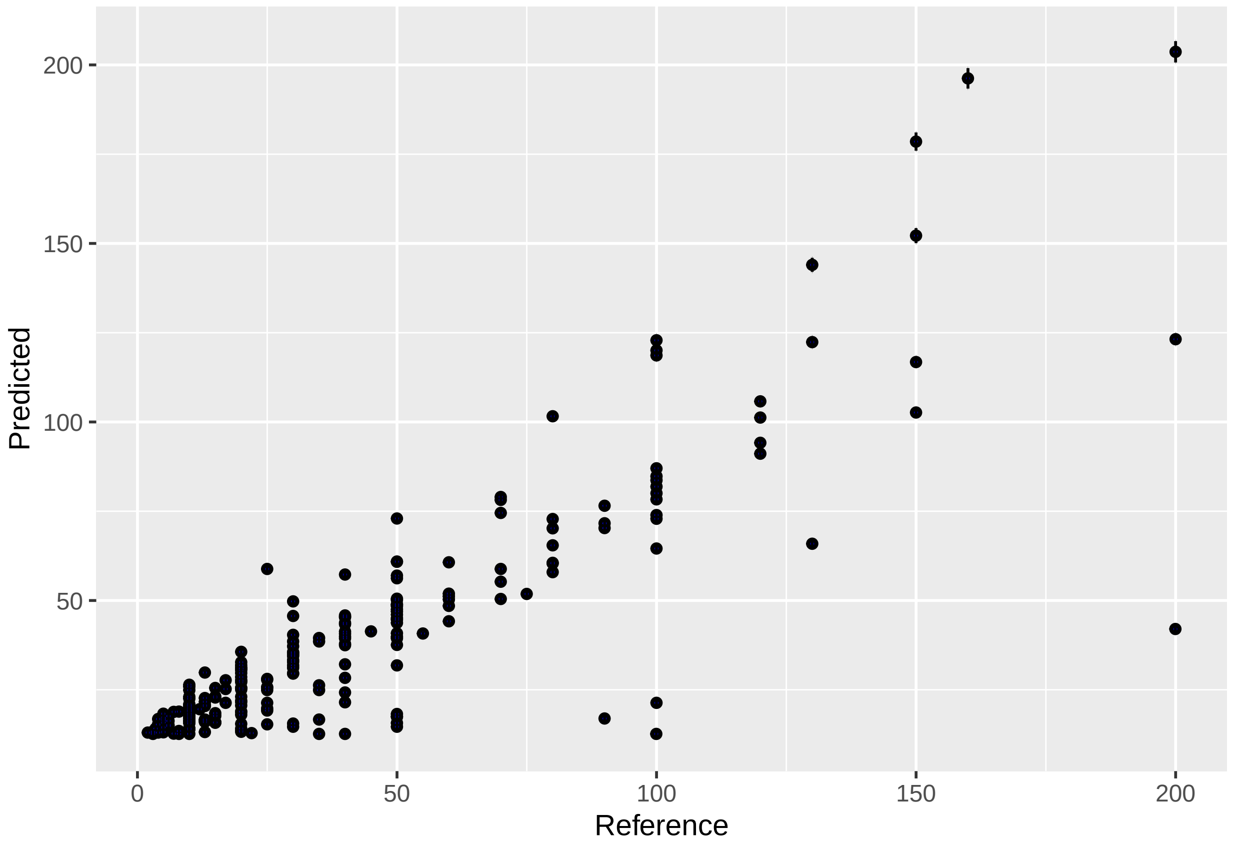
This means that on average, we can predict what the price of the lego set will be within $5.

Here I will graph the confidnece intervals of each prediction. The scale of this data is very large, so it is hard to see. Look to the top right of the plot to see good examples of the intervals on the little dots. The dataframe also provides a good insight into how it works. You can see that it usually predicts within a dollar of error on each side.

```
predPrice$lwr = predict(lr, newdata=testLego, interval = "confidence")[,2]
predPrice$upr = predict(lr, newdata=testLego, interval = "confidence")[,3]
head(predPrice)
```

```
##    Predicted Reference      lwr      upr
## 12  12.71225      6.99 12.02656 13.39794
## 15  12.63786      9.99 11.95136 13.32436
## 26  14.42316      4.99 13.75561 15.09072
## 36  17.62183      5.99 16.98573 18.25793
## 46  13.15857     12.99 12.47770 13.83945
## 52  17.39866     49.99 16.76048 18.03685
```

```
qplot(Reference, Predicted, data=predPrice) +
  geom_point(colour = "blue", size = 0.01) +
  geom_errorbar(aes(ymin = lwr,ymax = upr))
```

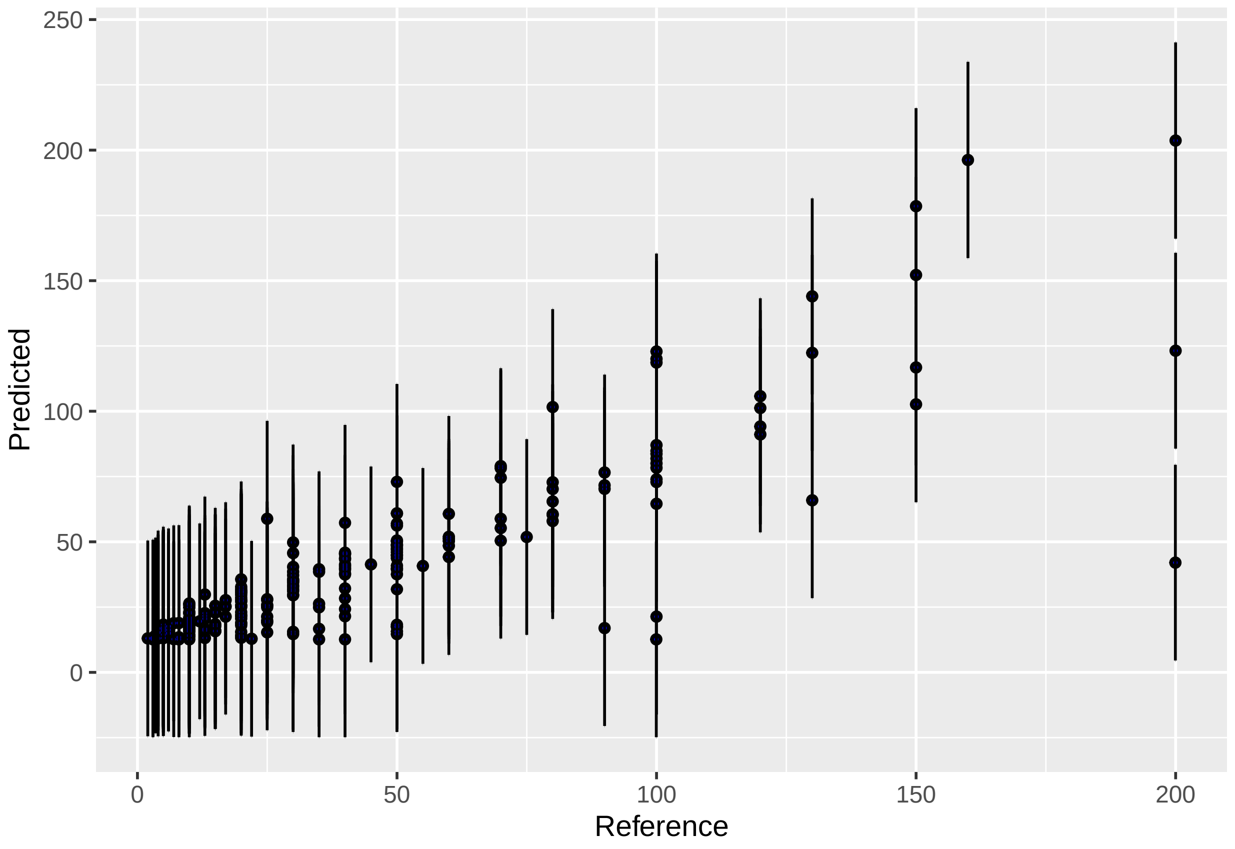


This wide tolerance is much easier to see on this plot. This is far more likely to be correct, but less useful for good predictions.

```
predPrice$lwr = predict(lr, newdata=testLego, interval = "prediction")[,2]
predPrice$upr = predict(lr, newdata=testLego, interval = "prediction")[,3]
head(predPrice)
```

```
##    Predicted Reference       lwr      upr
## 12  12.71225      6.99 -24.76947 50.19397
## 15  12.63786      9.99 -24.84388 50.11960
## 26  14.42316      4.99 -23.05823 51.90456
## 36  17.62183      5.99 -19.85902 55.10268
## 46  13.15857     12.99 -24.32306 50.64021
## 52  17.39866     49.99 -20.08222 54.87955
```

```
qplot(Reference, Predicted, data=predPrice) + geom_point(colour = "blue", size = 0.1) +
  geom_errorbar(aes(ymin = lwr,ymax = upr))
```



Conclusion

Hopefully this was a sufficient explanation of linear regression. I enjoyed going in depth into the actual equations that can be done by hand. It provides really good context to what r is doing behind the scenes and helped me understand the algorithm better overall.

You may explore regression of the data set yourself by following this link: https://posit.cloud/content/7242487

The shiny app is deployed on posit cloud and can be accessed by anyone.