Linear Regression Model in R

Ariel Felices

6/7/2022

# Building and analysing linear regression model in R

This project is for people who are interested in building a linear regression model on real world data set and analyze the model’s performance in R.

In this project, we will learn: 1. How to load and clean a real world data set. 2. How to build a linear regression model and create various plots to analyze the model’s performance. 3. How to predict future values using this model.

We don’t need to be a data science expert to complete this project, but we should be familiar with basic ggplot commands.

### Load R libraries

library(sjPlot)

## Warning: package 'sjPlot' was built under R version 4.1.3

## #refugeeswelcome

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(sjlabelled)

## Warning: package 'sjlabelled' was built under R version 4.1.3

##   
## Attaching package: 'sjlabelled'

## The following object is masked from 'package:dplyr':  
##   
## as\_label

library(sjmisc)

## Warning: package 'sjmisc' was built under R version 4.1.3

## Install package "strengejacke" from GitHub (`devtools::install\_github("strengejacke/strengejacke")`) to load all sj-packages at once!

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.1.3

##   
## Attaching package: 'ggplot2'

## The following object is masked from 'package:sjlabelled':  
##   
## as\_label

theme\_set(theme\_sjplot())

## Task 1 - Load dataset and summarize it

Our dataset file is called cars.csv, which contains cars characteristics and their price in the U.S. We will load this file in R as a data frame using read.csv() function.

data <- read.csv("cars.csv", header = TRUE, stringsAsFactors = FALSE)  
# To ensure our character columns are not converted to factors by  
# default, we have set the parameter stringsAsFactors to False.

Let’s look at the first few rows of the data set using the head function.

head(data)

## Make Model Year Engine.Fuel.Type Engine.HP Engine.Cylinders  
## 1 BMW 1 Series M 2011 premium unleaded (required) 335 6  
## 2 BMW 1 Series 2011 premium unleaded (required) 300 6  
## 3 BMW 1 Series 2011 premium unleaded (required) 300 6  
## 4 BMW 1 Series 2011 premium unleaded (required) 230 6  
## 5 BMW 1 Series 2011 premium unleaded (required) 230 6  
## 6 BMW 1 Series 2012 premium unleaded (required) 230 6  
## Transmission.Type Driven\_Wheels Number.of.Doors  
## 1 MANUAL rear wheel drive 2  
## 2 MANUAL rear wheel drive 2  
## 3 MANUAL rear wheel drive 2  
## 4 MANUAL rear wheel drive 2  
## 5 MANUAL rear wheel drive 2  
## 6 MANUAL rear wheel drive 2  
## Market.Category Vehicle.Size Vehicle.Style highway.MPG  
## 1 Factory Tuner,Luxury,High-Performance Compact Coupe 26  
## 2 Luxury,Performance Compact Convertible 28  
## 3 Luxury,High-Performance Compact Coupe 28  
## 4 Luxury,Performance Compact Coupe 28  
## 5 Luxury Compact Convertible 28  
## 6 Luxury,Performance Compact Coupe 28  
## city.mpg Popularity MSRP  
## 1 19 3916 46135  
## 2 19 3916 40650  
## 3 20 3916 36350  
## 4 18 3916 29450  
## 5 18 3916 34500  
## 6 18 3916 31200

We can see various columns like Make, Model, Engine HP and Cylinders. This will print the first six rows of our dataset. The last column is the price, which will be the response variable for our model. Let’s look at the data types of each column using the string function.

str(data)

## 'data.frame': 11914 obs. of 16 variables:  
## $ Make : chr "BMW" "BMW" "BMW" "BMW" ...  
## $ Model : chr "1 Series M" "1 Series" "1 Series" "1 Series" ...  
## $ Year : int 2011 2011 2011 2011 2011 2012 2012 2012 2012 2013 ...  
## $ Engine.Fuel.Type : chr "premium unleaded (required)" "premium unleaded (required)" "premium unleaded (required)" "premium unleaded (required)" ...  
## $ Engine.HP : int 335 300 300 230 230 230 300 300 230 230 ...  
## $ Engine.Cylinders : int 6 6 6 6 6 6 6 6 6 6 ...  
## $ Transmission.Type: chr "MANUAL" "MANUAL" "MANUAL" "MANUAL" ...  
## $ Driven\_Wheels : chr "rear wheel drive" "rear wheel drive" "rear wheel drive" "rear wheel drive" ...  
## $ Number.of.Doors : int 2 2 2 2 2 2 2 2 2 2 ...  
## $ Market.Category : chr "Factory Tuner,Luxury,High-Performance" "Luxury,Performance" "Luxury,High-Performance" "Luxury,Performance" ...  
## $ Vehicle.Size : chr "Compact" "Compact" "Compact" "Compact" ...  
## $ Vehicle.Style : chr "Coupe" "Convertible" "Coupe" "Coupe" ...  
## $ highway.MPG : int 26 28 28 28 28 28 26 28 28 27 ...  
## $ city.mpg : int 19 19 20 18 18 18 17 20 18 18 ...  
## $ Popularity : int 3916 3916 3916 3916 3916 3916 3916 3916 3916 3916 ...  
## $ MSRP : int 46135 40650 36350 29450 34500 31200 44100 39300 36900 37200 ...

We can see there are 11,914 rows and 16 characteristics as columns. The data types of each columns are correct, which is due to the parameter stringsAsFactors set to False. Lastly, we can summarize each column using the summary() function, which gives descriptive statistics of each column.

summary(data)

## Make Model Year Engine.Fuel.Type   
## Length:11914 Length:11914 Min. :1990 Length:11914   
## Class :character Class :character 1st Qu.:2007 Class :character   
## Mode :character Mode :character Median :2015 Mode :character   
## Mean :2010   
## 3rd Qu.:2016   
## Max. :2017   
##   
## Engine.HP Engine.Cylinders Transmission.Type Driven\_Wheels   
## Min. : 55.0 Min. : 0.000 Length:11914 Length:11914   
## 1st Qu.: 170.0 1st Qu.: 4.000 Class :character Class :character   
## Median : 227.0 Median : 6.000 Mode :character Mode :character   
## Mean : 249.4 Mean : 5.629   
## 3rd Qu.: 300.0 3rd Qu.: 6.000   
## Max. :1001.0 Max. :16.000   
## NA's :69 NA's :30   
## Number.of.Doors Market.Category Vehicle.Size Vehicle.Style   
## Min. :2.000 Length:11914 Length:11914 Length:11914   
## 1st Qu.:2.000 Class :character Class :character Class :character   
## Median :4.000 Mode :character Mode :character Mode :character   
## Mean :3.436   
## 3rd Qu.:4.000   
## Max. :4.000   
## NA's :6   
## highway.MPG city.mpg Popularity MSRP   
## Min. : 12.00 Min. : 7.00 Min. : 2 Min. : 2000   
## 1st Qu.: 22.00 1st Qu.: 16.00 1st Qu.: 549 1st Qu.: 21000   
## Median : 26.00 Median : 18.00 Median :1385 Median : 29995   
## Mean : 26.64 Mean : 19.73 Mean :1555 Mean : 40595   
## 3rd Qu.: 30.00 3rd Qu.: 22.00 3rd Qu.:2009 3rd Qu.: 42231   
## Max. :354.00 Max. :137.00 Max. :5657 Max. :2065902   
##

We can see statistics value such as mean and median are calculated for numeric columns but not for character columns.

## Task 2 - Clean our dataset

In this task, we will clean our dataset that we loaded in the previous task. Prior to building any machine learning model, our data sets should be accurate, reliable and robust. To achieve this, we should remove leading or trailing whitespaces in all cells, remove columns with high number of missing values and remove rows containing missing values.

# Remove the leading or trailing whitespaces in all cells.  
# We can only do this operation on character columns. Thus, we  
# should identify all character columns in our data set.  
cols <- names(data[vapply(data, is.character, logical(1))])

Here we’re collecting the names of all columns that returned True for the function is.character. Next, we will apply the function trimws() to remove leading and trailing whitespaces in these character columns.

data[, cols] <- lapply(data[,cols], trimws)

Missing values are always present in real world data sets, so we should know to deal with them. For this project, we will delete them. By default R will convert empty values to missing value equivalent NA but sometimes we can have manual entry for missing values as N/A our data set.

So let’s convert these manual entries to R’s version of missing values.

data[data=="N/A"] = NA

Now we can count the average number of missing values in each column using the mean() and is.na() function.

sapply(data, function(x) mean(is.na(x)))

## Make Model Year Engine.Fuel.Type   
## 0.0000000000 0.0000000000 0.0000000000 0.0000000000   
## Engine.HP Engine.Cylinders Transmission.Type Driven\_Wheels   
## 0.0057915058 0.0025180460 0.0000000000 0.0000000000   
## Number.of.Doors Market.Category Vehicle.Size Vehicle.Style   
## 0.0005036092 0.3140842706 0.0000000000 0.0000000000   
## highway.MPG city.mpg Popularity MSRP   
## 0.0000000000 0.0000000000 0.0000000000 0.0000000000

Here, sapply() is like a for loop, which goes to each column in our dataset and applies the functions to it. In the output we will notice the column “Market.Category” has high number off missing values, roughly 31.4%. It would be wise to remove this column in this case, but we should always be concerned about which column we are removing as it could be an important column.

data$Market.Category <- NULL

Lastly, we can remove rows that contain one or more empty values. This will ensure that our dataset is complete.

data <- data[complete.cases(data), ]

Here, the function complete.cases() returns rows that have no missing values.

head(data)

## Make Model Year Engine.Fuel.Type Engine.HP Engine.Cylinders  
## 1 BMW 1 Series M 2011 premium unleaded (required) 335 6  
## 2 BMW 1 Series 2011 premium unleaded (required) 300 6  
## 3 BMW 1 Series 2011 premium unleaded (required) 300 6  
## 4 BMW 1 Series 2011 premium unleaded (required) 230 6  
## 5 BMW 1 Series 2011 premium unleaded (required) 230 6  
## 6 BMW 1 Series 2012 premium unleaded (required) 230 6  
## Transmission.Type Driven\_Wheels Number.of.Doors Vehicle.Size Vehicle.Style  
## 1 MANUAL rear wheel drive 2 Compact Coupe  
## 2 MANUAL rear wheel drive 2 Compact Convertible  
## 3 MANUAL rear wheel drive 2 Compact Coupe  
## 4 MANUAL rear wheel drive 2 Compact Coupe  
## 5 MANUAL rear wheel drive 2 Compact Convertible  
## 6 MANUAL rear wheel drive 2 Compact Coupe  
## highway.MPG city.mpg Popularity MSRP  
## 1 26 19 3916 46135  
## 2 28 19 3916 40650  
## 3 28 20 3916 36350  
## 4 28 18 3916 29450  
## 5 28 18 3916 34500  
## 6 28 18 3916 31200

We will notice that the number of observations now stand at 11,815 and the number of columns stand at 15.

*Note:* We can also view our dataset by clicking on the data variable in the Environment pane in RStudio.

## Task 3 - Split into training and test set

In this task we will split our data set into training and test set to build our model. One of the most important steps in machine learning is to create in our model on a training set that is separate and distinct from the test set for which we will gauge its accuracy. Failure to do so will result in a model that may not generalize to unseen or future data set.

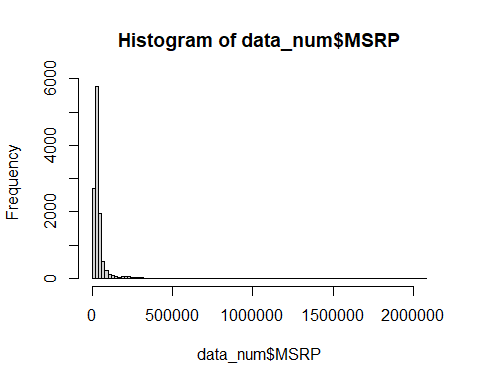
First we should select only numeric columns from our data set for a linear regression model. We can do this using select\_if() function.

data\_num <- data %>% select\_if(is.numeric)

This will select numeric columns and store the result in a data\_num variable. Our target variable for machine learning model is the price column, which is called MSRP.

We can create a histogram of its values to see the distribution.

hist(data\_num$MSRP, breaks = 100)



The histogram tells us that there are some outliers in our column as the majority of cars have the price in the region of 0 in x-axis.

These outliers can cause issues in your model. So we will filter our dataset to include cars with price range between 15,000 and 50,000.

data\_num <- data\_num %>%   
 filter(MSRP>15000) %>%   
 filter(MSRP<50000)

head(data\_num)

## Year Engine.HP Engine.Cylinders Number.of.Doors highway.MPG city.mpg  
## 1 2011 335 6 2 26 19  
## 2 2011 300 6 2 28 19  
## 3 2011 300 6 2 28 20  
## 4 2011 230 6 2 28 18  
## 5 2011 230 6 2 28 18  
## 6 2012 230 6 2 28 18  
## Popularity MSRP  
## 1 3916 46135  
## 2 3916 40650  
## 3 3916 36350  
## 4 3916 29450  
## 5 3916 34500  
## 6 3916 31200

Now our dataset contains around 8000 cars and 8 columns.

Now let’s split our dataset into training and test set. To get consistent results and to make sure our partitions are reproducible, we set the seed to any integer.

Next, we will select 80% of our data set as training and remaining 20% as test. To do so, we will get the number of rows that account for 80%. We will use the floor() function to round up the calculation to an integer.

set.seed(123)  
size <- floor(0.8 \* nrow(data\_num))

Next, we will use the sample() function to randomly select 80% of rows from our dataset and store the row numbers or indices.

train\_ind <- sample(seq\_len(nrow(data\_num)), size = size)

To get the training set, we can filter our dataset to include the row numbers.

train <- data\_num[train\_ind, ]

head(train)

## Year Engine.HP Engine.Cylinders Number.of.Doors highway.MPG city.mpg  
## 2463 2016 288 6 4 22 15  
## 2511 2007 242 6 4 23 16  
## 2227 1994 182 8 2 15 12  
## 526 2016 170 4 2 35 24  
## 4291 2016 175 4 4 36 26  
## 2986 2016 287 6 4 23 16  
## Popularity MSRP  
## 2463 155 43660  
## 2511 1439 26395  
## 2227 258 27736  
## 526 3105 36600  
## 4291 640 27145  
## 2986 5657 37800

To get the test set, we can filter our dataset to ignore the row numbers.

test <- data\_num[-train\_ind, ]

head(test)

## Year Engine.HP Engine.Cylinders Number.of.Doors highway.MPG city.mpg  
## 3 2011 300 6 2 28 20  
## 7 2012 300 6 2 26 17  
## 10 2013 230 6 2 27 18  
## 12 2013 230 6 2 28 19  
## 14 2013 230 6 2 28 19  
## 15 2013 230 6 2 28 19  
## Popularity MSRP  
## 3 3916 36350  
## 7 3916 44100  
## 10 3916 37200  
## 12 3916 31500  
## 14 3916 37200  
## 15 3916 31500

We will notice that our training set contains 6392 observations, and your test contained roughly 1600 observations.

## Task 4 - Fit linear regression model and interpret model summary statistics

In this task, we will build a linear regression model and interpret model summary statistics.

A linear regression model is a model that assumes a linear relationship between the predictors and the response variable. This means that the response variable can be calculated from a linear combination of the predictors.

In our numeric dataset, the response variable is *MSRP* column while the remaining columns serves as the predictors.

Our aim is to build a model to predict the *MSRP* column, using the car characteristics such as *Engine HP*, *number of doors*, et cetera.

To build a linear regression model, we will use the *lm()* function. We will focus on two parameters - **the model equation** and **the dataset** to be used. Model equation can be written in terms of column names while the dataset we will be using will be the training dataset.

model <- lm(MSRP ~ ., data = train)

Here we’re specifying the MSRP column as the response variable while all other columns (represented by dot, .) are predictors. The model is built and stored in a variable called model. To see the summary of our model, we can use the *summary()* function on the model.

summary(model)

##   
## Call:  
## lm(formula = MSRP ~ ., data = train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -18468.7 -3769.3 -684.6 3423.8 20456.2   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -3.000e+05 3.978e+04 -7.541 5.32e-14 \*\*\*  
## Year 1.543e+02 1.989e+01 7.759 9.93e-15 \*\*\*  
## Engine.HP 1.118e+02 1.756e+00 63.659 < 2e-16 \*\*\*  
## Engine.Cylinders -1.201e+03 9.944e+01 -12.081 < 2e-16 \*\*\*  
## Number.of.Doors 2.526e+02 8.925e+01 2.830 0.00467 \*\*   
## highway.MPG -1.170e+02 2.905e+01 -4.028 5.68e-05 \*\*\*  
## city.mpg 1.634e+02 2.554e+01 6.398 1.69e-10 \*\*\*  
## Popularity -2.246e-01 4.952e-02 -4.536 5.85e-06 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 5495 on 6384 degrees of freedom  
## Multiple R-squared: 0.5953, Adjusted R-squared: 0.5949   
## F-statistic: 1342 on 7 and 6384 DF, p-value: < 2.2e-16

The *Call* shows the function call used to compute the regression model. The *Residuals* provide the quick view of the distribution of the residuals, which by definition have a mean zero.

The critical part of the summary are the *Coefficients*. This shows the regression **beta coefficients** and their statistical significance. Predictor variables that are significantly associated with outcome variable are marked by stars. The higher the number of stars, the most significant predictors are.

For a given predictor value, the coefficient, which is called the *Estimate*, can be interpreted as the average effect on the response variable of 1 unit increase in predictor, given that all other predictors are fixed.

For example, if the engine horsepower increases by one unit and all other predictors are kept constant, the price of the car will increase by 111 units.

The *Residual Standard Error (RSE)*, *R- squared*, and the *F-statistics* are metrics that are used to check how well the model fits to our data.

*Residual Standard Error (RSE)* corresponds to the prediction error in our training set and represents roughly the average difference between the observed values and the predicted values by the model.

In this model, the *Residual Standard Error (RSE)* is 5495. That means on average, you can expect a deviation of 5495 in the price prediction.

The *R-squared* ranges from 0 to 1 and represents the proportion of the variation in the response variable that can be explained by the model predictor variables.

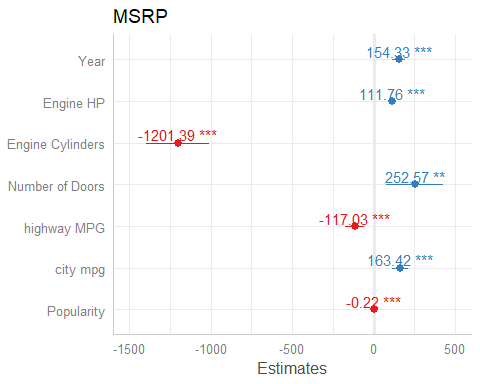
The higher the *R-squared* value, the better the model is. However, a problem with the *R-squared* is that it will always increase when more predictors are added to the model even if those predictors are only weakly associated with the outcome or the response variable.

A solution is to adjust the *R-squared* value by taking into account the number of predictor variables. The adjustment in the *Adjusted R-squared* value in the summary output is a correction for the number of predictor variables included in the model. So we mainly considere the *Adjusted R-squared* value.

Our value is 0.59 which is good. The *F-statistic* gives the overall significance of the model. It assesses whether at least one predictor value or variable has a non zero coefficient. The P-value of less than 10 to the power -16 shows that the model is highly significant.

We can also plot the estimates for a better visual interpretation. We can use the plot\_model() function for this purpose.

plot\_model(model, show.values = TRUE, value.offset = 0.2)

 This plot shows us the coefficients and the significance value.

Lastly, we can build a linear regression model by explicitly specifying the predictors that we want. For example, we may wish to include only three predictors rather than all from your numeric dataset. The syntax is similar, but we explicitly type out the predictors names.

model2 <- lm(MSRP ~ Engine.HP + highway.MPG + Engine.Cylinders,  
 data = train)

Now, a second model has been created.

## Task 5 - Plot and analyse model residuals

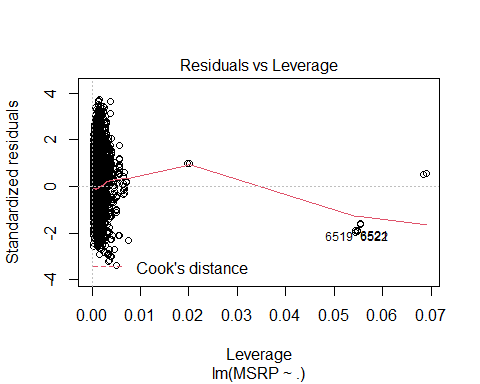
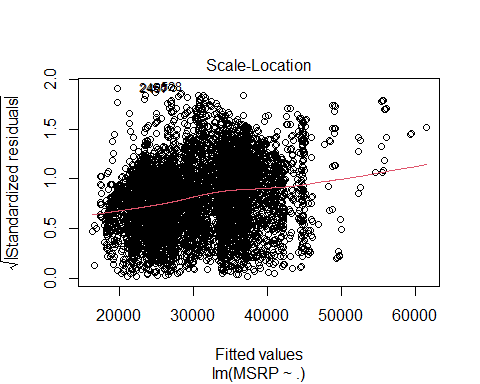
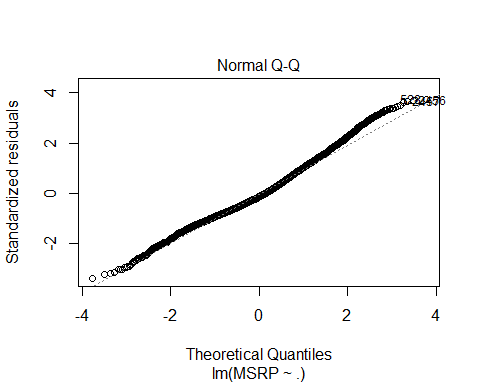
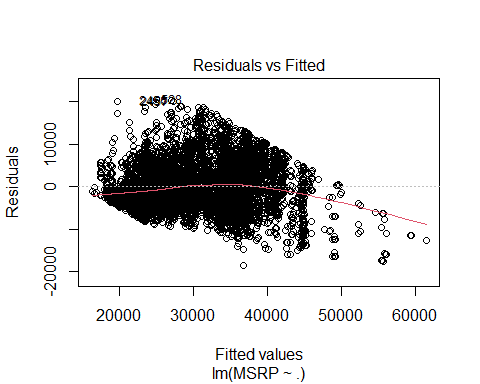
In this task, we will plot and analyze the model residuals. Residuals could show how poorly a model represents data. Residuals are leftover values of the response variable after a fitting a model to data and they could reveal unexplained patterns in the data by the fitted model. Using this information, not only could we check if linear regression assumptions are met, but we could improve our model as well.

The plot function in R can work on linear regression models. It creates 4 diagnostic plots, each showing the residuals in different ways. To show all 4 plots in one plot, you can change the options of plotting.

par(mfrow=c(2,2))

This will set the plotting pane to include 4 plots as 2 rows and 2 columns. Next simply call the plot function on the model.

plot(model)

 This will create 4 plots. Now let’s investigate each of them.

We’ll often see numbers next to some points in each plots. These are extreme values based on each criterion, and are identified by the row numbers in the data set.

The first plot, **Residuals vs fitted** plot, shows if residuals have non-linear patterns. Fitted values are on the X axis and the residuals (that is, how far the fitted values are from the observed values) are on the Y axis. There could be a non linear relationship between predictor variables and the response variable, and the pattern could show up in this plot if the model doesn’t capture the non-linear relationship. If we find equally spaced residuals around the horizontal line without distinct patterns, that is a good indication we don’t have non linear relationships. In our plot we don’t see any distinctive patterns.

The second plot, **Normal Q-Q** plot, shows if residuals are normally distributed. Do the residuals follow a straight line, or do they deviate severely? It’s good if residuals are lined well on the straight dashed line, but in reality, we will see some deviations. In our plot we don’t see much deviation until towards the end, where some data points are deviating.

The third plot, **Scale-Location** plot, shows if residuals are spread equally along the ranges of predictors. This is how we can check the assumption of equal variance. It’s good if we see a horizontal line with equally or randomly spread points. In our model the residuals appear randomly spread.

The last plot, **Residuals vs Leverage** plot, helps us to find influential cases in our data set. These cases could be extreme cases against the regression line and can alter the results if we exclude them from our model. In this plot, patterns are not relevant. We should watch out for outlying values at the upper right corner or the lower right corner. Those spots are the places where the cases can be influential against the regression line. Look for cases outside of a dash line, the Cook’s distance. When cases are outside of the Cook’s distance, meaning they have high Cook’s distance scores, the cases are influential to the regression results. In our model, we can see observation number 6519 and 6522 are far beyond the Cook’s distance lines. These are influential cases and will alter our model if we remove them.

The 4 plots show potential problematic cases with the row numbers of the data in our data set. If some cases are identified across all 4 plots, we might want to take a closer look at them. Is there anything special about those points? Or could they just be simply errors in data entry? Our current model might not be the best way to understand our data, and w may need to revisit the model building step. We can try including or excluding predictors and see if the diagnostic plots improve.

## Task 6 - Predict future values and calculate model error metrics

In this final task we will predict future values and calculate model error metrics.

Using a regression model, you can predict future values and utilize these predictions in your business. For example, these predictions could be the number of sales in the next month or the amount of rain to fall tomorrow. Using the predictive values and the observed values, you can also assess your model’s performance and calculate error metrics such as mean absolute error and root mean squared error. Great, let’s get into it! You will predict the MSRP values of the test data set and compare it with the observed MSRP values. You can use the predict() function with the parameters - model and test data.

test$pred <- predict(model, newdata = test)

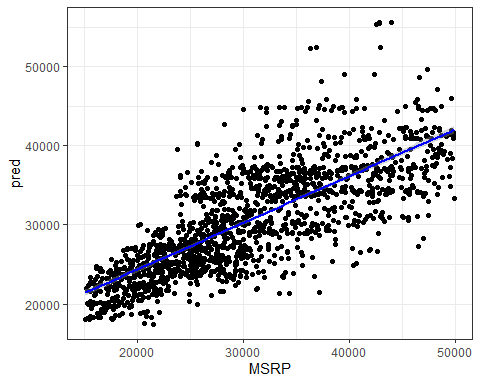
We are storing the predicted values in a new column called pred in the test data set.

Next, we can plot the predicted and observed MSRP values using ggplot. But before we do that, we have to reset our plotting pane to one plot per pane.

par(mfrow=c(1,1))

ggplot(test, aes(x = MSRP, y = pred)) +  
 geom\_point() +   
 geom\_smooth(method = 'lm', color = "blue") +  
 theme\_bw()

## `geom\_smooth()` using formula 'y ~ x'

 On the X axis, you can see the observed MSRP value and on the Y axis it, the predicted values. The blue line is a regression line between the predicted and observed values.

Next, we will calculate the error metrics of the linear model. we will first find the error, which is each observed value subtracted from the respective predicted value.

error <- test$pred - test$MSRP

We would calculate two error metrics. The first is **root mean square error** or **RMSE**.

RMSE is a good measure of how accurately the model predicts the response and it is the most important criteria for fit if the main purpose of the model is prediction.

rmse <- sqrt(mean(error^2))  
rmse

## [1] 5546.37

Our model’s RMSE value is 5546.37 which is fine, given that the range of our MSRP value in our dataset is between 15,000 and 50,000.

The second error metric is **mean absolute error** or **MAE**.

MAE measures the average magnitude of the errors in our predictions without considering their direction.

mae <- mean(abs(error))  
mae

## [1] 4401.405

Our model’s mean absolute error (MAE) is 4401 which means that on average we would expect an error magnitude of 4401 in our predictions. This error can either be positive or negative.

In RMSE, since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable. But from an interpretation standpoint, mean absolute error (MAE) is better.

**Note:** Other ways to compute for MAE is by loading the library “Metrics” and perform the function below: - mae(observed, predicted) Where: **mae**: function for MAE **observed**: observed values **predicted**: predicted values

### End of project