Question 1 - Confidence Intervals and Hypothesis Testing

Previously, we worked with a company who was making phone cases using a new 3D printing method. Because 3D printing is not a perfect process, management wanted to know how many samples they should produce to get a good approximation for the failure probability of the process.

This was several weeks ago, and the engineers have now begun testing the new 3D printing process. We are no longer concerned with whether a print is a "success" or "failure" - we are going to be more precise now. Although they will do more tests later, the engineers have performed an initial test run of 20 successful units (that is, they keep producing units until they have 20 "successful" prints).

For this question, we are going to assume that the 3D print thickness is normally distributed.

In [1]:
```
test.20.units <- read.csv("test20units.csv")$x
head(test.20.units)
```

1.15709584471467 0.963530182860391 1.05631284113373 1.0832862604961 1.0604268323141
1.00938754839085

We have decided that, for the 3D printing process to be considered a success, we are going to require the following facts to be true (with a confidence level $\alpha = 0.01$):

- The mean print thickness must be 1mm
- The confidence interval for the mean must be $\leq$ 0.1mm wide

We will test these facts by determining whether the corresponding values are within the 99% confidence intervals for our estimates, and also whether the wideness of our confidence intervals fits. That is, we will calculate a two-sided 99% confidence interval for the mean of the print thickness and see if 1mm is within that range, and whether that range has a total width $\leq$ 0.1mm wide.

For this question, since we both have a small sample size and also don't know the standard deviation of the population (we are using the estimate from the sample), use the t-distribution rather than the normal distribution

In [3]:
```
conf.interval.mean.t <- function(data, alpha) {
    mean <- mean(data)
    t <- qt(p = 0.005,df = length(data)-1)
    std <- sd(data)
    sqt <- sqrt(length(data))

    lower_bound <- mean - ((t*std)/sqt)
    upper_bound <- mean + ((t*std)/sqt)

    return(c("Lower Bound = ",lower_bound,"Upper Bound = ",upper_bound))
}
```

In [4]:
```
ci.mean.20 <- conf.interval.mean.t(test.20.units, 0.01)
ci.mean.20
```

'Lower Bound = ' '1.1231640483385' 'Upper Bound = ' '0.955219955546044'

The two citerias (The mean print thickness must be 1mm and The confidence interval for the mean must be ≤0.1mm wide) established are not met. The desired mean of 1mm thickness is within the confidence interval of 0.96 and 1.12. However, the confidence interval is not less than 0.1mm wide from the lower to upper bound.

It is important to consider both whether the value is within the confidence interval and how wide the confidence interval. This is so as if the value is within the confidence interval then there is a 99% probability that the confidence interval will contain the true mean print thickness. The wideness of the confidence interval quantifies how uncertain we are about our estimate. A wide interval shows a high uncertainty whereas a low interval shows a low uncertainty.

Question 1b

The engineers have developed two alternative methods for 3D printing the phone cases. The engineers are now trying to minimize "warping" (which is where phone units are not printed flat and are instead warped slightly - this warping is measured in millimeters).

We have two datasets (one for each of the two new printing methods), where each dataset contains warping measurements from 50 prints produced using that dataset's 3D printing method. From our many tests with the original method, we know that the mean warping measurement for the original printing method is very close to 0.1mm (so we will use that as our estimate for the mean of the original population).

The engineers have asked us to determine whether either of their new 3D printing methods are superior to the original method. To do this, we could just calculate the mean of each dataset's measurements and compare that to the 0.1mm mean of the original method, but this does not take into account the variance introduced by sampling from the population. Instead, we will use hypothesis testing.

For each of the two new printing methods, each with datasets method1.50.units and method2.50.units respectively, we need to test the hypothesis that the mean of that method's dataset is **less** than the mean of the original population (i.e., less than 0.1mm). So, for each of the two printing methods, do the following (there are four cells below - the first cell imports the data, and the next three cells correspond to the following three points):

- Create a hypothesis test (defining the null hypothesis and alternative hypothesis). Note that this should be a **one-sided** test.
- Calculate a p-value using a t.test
- Interpret this p-value, using a confidence level $\alpha = 0.01$

When interpreting the p-value, keep in mind that the goal is to determine whether each of the new methods is better than the original method (in terms of warping).

In [5]:
```
method1.50.units <- read.csv("method1_50units.csv")$x
method2.50.units <- read.csv("method2_50units.csv")$x
```

Step 1:Defining null hypothesis and alternative hypothesis for one-sided test

$H_0 : \mu \geqslant \mu_0$

$H_1 : \mu \leq \mu_0$

Step 2: Calculating P value using t-test

In [7]:
```
pop_mean <- 0.1
stddev1 <- sd(method1.50.units)
stddev2 <- sd(method2.50.units)
n1 <- sqrt(length(method1.50.units))
n2 <- sqrt(length(method2.50.units))

#getting t value
tvalue_method1 <- (mean(method1.50.units) - pop_mean)/(stddev1/n1)
# converting to p value
pvalue1 <- pt((tvalue_method1),df= 50 -1)

#getting t value
tvalue_method2 <- (mean(method2.50.units) - pop_mean)/(stddev2/n2)
# converting to p value
pvalue2 <- pt((tvalue_method2),df= 50 -1)


paste0('P value for method 1: ',pvalue1)
paste0('P value for method 2: ',pvalue2)
```

'P value for method 1: 0.00417628050226874'

'P value for method 2: 0.678562406130603'

Step 3: Interpretting p value using confidence level = 0.01

In [8]:
```
confidence_level <- 0.01

if(pvalue1 > confidence_level){
cat("Fail to reject null hypothesis, Method 1 is superior")

    } else {
    cat("Reject null hypothesis, Initial method is superior compared to method 1")}
```

```
Reject null hypothesis, Initial method is superior compared to method 1
```

In [9]:
```
if(pvalue2 > confidence_level){
cat("Fail to reject null hypothesis, Method 2 is superior")

    } else {
    cat("Reject null hypothesis,Initial method is superior compared to method 2  ")}
```

```
Fail to reject null hypothesis, Method 2 is superior
```

Question 1.c

When we use $\alpha = 0.01$ for a single hypothesis test, we expect to find a "false positive" (that is, we reject the null hypothesis when we should not have) in around 1 in 100 experiments.

However, in Question 1.c, we did two hypothesis tests (one for each of our two methods). We will consider the implications of this below.

Please do the following:

- Assuming that the probability of finding a false positive is 1% for each experiment, calculate the probability of finding at least 1 false positive overall (for two hypothesis tests).
- Calculate the probability of finding at least 1 false positive overall, if we tested 100 methods instead of just two.
- Briefly discuss something you think data scientists could do to reduce the risk of false positives when performing multiple hypothesis tests at once. Justify your answer.

YOUR ANSWER HERE

Given the probability of finding a false positive is 0.01, the probabability of not finding a false positive will be 1-0.01 = 0.99.

The probability of finding at least 1 false potive overall for both the hypothesis tests are: 1- probability of not finding any false positive over both the tests

Hence: 1 - (0.99 x 0.99) = 0.0199

Probability of finding at least 1 false positive overall if we tested 100 methods instead of just two: 1-(0.99^{100}) = 0.6339

False positive (type 1 error) occurs when we reject a true null hypothesis. Data scientists can reduce the risk of false positives (type 1 error) when performing multiple hypothesis test at once by choosing a smaller significance level. This is so as lower value of significance level makes it harder to reject the null hypothesis. As a result, if the null hypothesis is false, it may be more difficult to reject using a low value for significance test.

Question 1.d

The engineers have decided, for other reasons than warping, to use one of the two new 3D printing methods instead of the old one. They would now like us to determine which of these two new printing methods is superior.

So, we need to test two hypotheses:

- That method 1 is superior to method 2 (i.e. method1 has a **lower** mean warping than method2)
- That method 2 is superior to method 1 (i.e. method2 has a **lower** mean warping than method1)

For each of methods 1 and 2, please do the following (there are three cells below, corresponding to the the following three points):

- Create a hypothesis test (defining the null and alternative hypotheses) which tests whether the chosen method is superior to the alternative method (i.e. either method 1 is superior to method 2, or method 2 is superior to method 1). Note that this should be a **one-sided** test.
- Calculate a p-value
- Interpret the p-value, using a confidence level $\alpha = 0.01$

Step 1:Defining null hypothesis and alternative hypothesis for one-sided test

First hypothesis testing is to test whether method 1 is superior to method 2 (i.e. method1 has a lower mean warping than method2)

$H_0 : \mu_1 \leq \mu_2$

$H_1 : \mu_1 \geqslant \mu_2$

Second hypothesis testing is to test whether method 2 is superior to method 1 (i.e. method2 has a lower mean warping than method1)

$H_0 : \mu_2 \leq \mu_1$

$H_1 : \mu_2 \geqslant \mu_1$

Step 2: Calculating P value using t-test

In [10]:
```
mean1 <- mean(method1.50.units)
mean2 <- mean(method2.50.units)
stddev1 <- sd(method1.50.units)
stddev2 <- sd(method2.50.units)
n1 <- sqrt(length(method1.50.units))
n2 <- sqrt(length(method2.50.units))

#getting t value
tvalue_method1 <- (mean1 - mean2)/(stddev1/n1)
# converting to p value


pvalue_1 <- pt(-abs(tvalue_method1),df= 50 -1)



#getting t value
tvalue_method2 <- (mean1 - mean2)/(stddev1/n1)
# converting to p value
pvalue_2 <- pt(abs(tvalue_method2),df= 50 -1)


paste0('P value for method 1: ',pvalue_1)
paste0('P value for method 2: ',pvalue_2)
```

'P value for method 1: 0.000665206789150935'

'P value for method 2: 0.999334793210849'

Step 3: Interpretting p value using confidence level = 0.01

In [11]:
```
confidence_level <- 0.01

if(pvalue_1 > confidence_level){
cat("Fail to reject null hypothesis, Method 1 is superior than method 2")

    } else {
    cat("Reject null hypothesis,Method 2 is superior than method 1 ")}
```

```
Reject null hypothesis,Method 2 is superior than method 1
```

In [12]:
```
confidence_level <- 0.01

if(pvalue_2 > confidence_level){
cat("Fail to reject null hypothesis, Method 2 is superior than method 1")

    } else {
    cat("Reject null hypothesis, Method 1 is superior than method 2")}
```

```
Fail to reject null hypothesis, Method 2 is superior than method 1
```

Question 1.e

If you completed Question 1.e correctly, you should have found that the two p-values calculated (one for each of the two methods) sum to 1. Or, put another way, each p-value is $1-$ the other p-value.

Explain why this should be the case, justifying your answer.

In [13]:
```
paste0("P value of testing which two new printing methods is superior (question 1e):  ", pvalue
```

```
'P value of testing which two new printing methods is superior (question 1e):  1'
```

Both the p values calculated should add up to 1 as it is a symmetric distribution.

Question 2 - Logistic Regression

The engineers are now doing the first production run, which is going to take a while. One of your colleagues has challenged you to a game of chess while you wait for the test run to complete.

The game is being played by correspondance (meaning that you aren't playing over a board, and can do other things while you play). Your opponent neglected to tell you that they are a former Australian chess champion, and you have unfortunately found yourself in a tricky position.

Your opponent is now beginning to gloat, and has asked you to resign so as to not waste the time of such a great chess player as them (they're only kidding but it still hurts). You think that maybe you can use data science to determine whether the position is lost (meaning you should resign) or drawn (meaning you should keep playing and teach your colleague a lesson).

One of your other colleagues suggested that, since your opponent is so good, they surely should be able to win in ten turns or less. Your opponent has agreed to these rules - if they cannot beat you in ten turns, the game will be declared a draw.

You've found a dataset which contains thousands of chess positions where white has a rook and a king, and black has a king. Each entry in the original dataset also includes whether the position is winnable by white some number of turns (labelled by the number of turns, e.g. "one", "two", ..., "fifteen", "sixteen") or not (labelled "draw", since black can only draw with a king). In each position, it is black's turn to play.

The code cell below imports the dataset and does a small transformation on it, converting it from a 17-class problem ("draw", "one", "two", ..., "sixteen") to a binary problem ("loss" or "draw"), where a position is considered a draw if either black can force a draw, or if white cannot win within ten moves (with optimal play on both sides). Each entry in this transformed dataset contains the following information:

- The white king's file (a, b, c, d)
- The white king's rank (1, 2, 3, 4)

- The white rook's file (a, b, ..., h)
- The white rook's rank (1, 2, ..., 8)
- The black king's file (a, b, ..., h)
- The black king's rank (1, 2, ..., 8)
- The optimal end result of this position for black ("loss" if the position can be won by white in less than 10 moves, and "draw" otherwise).

Notice that the white king's rank and file are always within 1-4 and a-d respectively. It appears that whoever prepared this dataset rotated the board so the white king was always in this quadrant, probably to reduce the number of variables in our model. This would be something we would have to worry about if we wanted to make this model work for any possible chess position; in any case, don't worry about this for the purposes of this assignment as it won't be an issue (your opponent's king happens to be in this quadrant so it's not a problem).

In [14]:
```
N = 250

set.seed(42)

raw.df <- read.csv("krkopt.data")
names(raw.df) <- c("wk_file", "wk_rank", "wr_file", "wr_rank", "bk_file",
                   "bk_rank", "result")

draws <- c("draw", "eleven", "twelve", "thirteen", "fourteen", "fifteen", "sixteen")

draw.idx <- raw.df$result %in% draws
loss.idx <- !draw.idx

draw.df <- raw.df[draw.idx,]
loss.df <- raw.df[loss.idx,]

draw.df$result = 1
loss.df$result = 0

draw.df <- draw.df[sample(nrow(draw.df), N),]
loss.df <- loss.df[sample(nrow(loss.df), N),]
```

```
loss.df <- loss.df[sample(nrow(loss.df), N),]

raw.df <- rbind(draw.df, loss.df)
raw.df <- raw.df[sample(nrow(raw.df)),]
head(raw.df)
```

A data.frame: 6 x 7

	wk_file	wk_rank	wr_file	wr_rank	bk_file	bk_rank	result
	<fct>	<int>	<fct>	<int>	<fct>	<int>	<dbl>
21704	b	1	d	5	e	8	1
1752	d	1	c	8	d	8	1
7079	c	2	d	2	a	8	0
5875	d	2	c	7	h	2	0
11016	c	1	f	8	g	4	1
5448	c	2	b	5	e	1	0

Before we can use our dataset, we need to preprocess it to make it suitable for use in a logistic regression. At the moment, our dataset consists of six feature variables, all of which are categorical, as well as one label variable (which is binary). We need to perform "one-hot encoding" on all of these categorical variables (i.e. all of the categorical variables, but not the binary label variable - think about why it would be useless to do one-hot encoding on a binary variable, if we're avoiding the dummy variable trap as discussed below).

One-hot encoding is the process of taking a categorical feature (with k possible values it can take) and converting it to a set of k binary values (in practice we use $k-1$, to avoid the "dummy variable trap"). It is important to do this because, when categorical variables are encoded as a single integer (e.g. "red" = 1, "green" = 2, "blue" = 3, ...), a logistic regression (and many other modelling techniques) will fit the feature as though "red" is closer to "green" than it is to "blue" (since 1 is closer to 2 than to 3). So, we split the values up into individual variables to avoid this.

In fact, R's built-in logistic regression automatically performs one-hot encoding on factor variables prior to running the algorithm (but we're going to ignore that and implement the encoding ourselves instead). To learn more about one-hot encoding, Wikipedia has a good page on it at https://en.wikipedia.org/wiki/Dummy_variable_(statistics) (https://en.wikipedia.org/wiki/Dummy_variable_(statistics)) (note that this technique goes by several names, including one-hot encoding, dummy variables, indicator variables, and a few others listed on the Wikipedia page).

When we perform one-hot encoding, we take the dataset and, for each column we wish to encode:

- Find how many different values the factor variable can take, using the levels() function; call this number k
- Create $k - 1$ new columns in the data-frame, corresponding to each level of the factor. For each row, set the value to 1 if the original variable's value corresponded to the column, and 0 otherwise. If the variable took the last value of the factor (the k-th value), just set them all to 0. We use $k - 1$ columns to avoid the "dummy variable trap" (https://en.wikipedia.org/wiki/Dummy_variable_(statistics) (https://en.wikipedia.org/wiki/Dummy_variable_(statistics)))
- Don't forget to delete the original factor column from the data frame when you're done

So, prepare our dataset as follows:

1. Write a function to.factors, which takes the variables listed below and returns a data-frame identical to df, except that the columns listed in col.names have been converted to factor columns (you are allowed to use R's built-in as.factor() function here)
    - df is a data frame for which we wish to convert some variables to factors
    - col.names is a vector of strings, where each string corresponds to the name of a column in the data-frame which we want to convert to a factor column using the to.factor() function

2. Write a function one.hot.encode, which takes two variables (df and to.encode) and returns a data-frame identical to df, except that the factor columns listed in to.encode have been replaced with one-hot encoding columns as described above
    - df is a data frame which we wish to apply one-hot encoding to
    - to.encode is a vector of strings, where each string corresponds to the name of a column in the data-frame which we want to convert from a factor column to a one-hot encoding

In [15]:

```
to.factors <- function(df,col.names) {
    no_col <- length(col.names)
    for(i in 1:no_col){
        df[,col.names[i]] <- factor(df[,col.names[i]])
        }
        return(df)

}

one.hot <- function(df,to.encode){
    for(name in to.encode){
        level <-levels(df[,name])
        length <- nlevels(df[,name])-1
        for(i in 1:length){
            column_names <- paste0(name,"_",level[i])
            df[column_names] <- ifelse(df[name] == level[i],1,0)
        }
    }
    data <- df[,!(names(df)%in% to.encode)]
    return(data)
}
```

In [16]:
```
preprocess <- function(df, factor.names) {
    fact.df <- to.factors(df, factor.names)
    one.hot.df <- one.hot(fact.df, factor.names)

    return (one.hot.df)
}

factor.names <- c("wk_file", "wk_rank", "wr_file", "wr_rank", "bk_file", "bk_rank")
df.student.preprocessed <- preprocess(raw.df, factor.names)

head(df.student.preprocessed)
```

A data.frame: 6 x 35

	result	wk_file_a	wk_file_b	wk_file_c	wk_rank_1	wk_rank_2	wk_rank_3	wr_file_a	wr_file_b	wr_file_c	...	bk_file_e	l
	<dbl>	<dbl[,1]>	<dbl[,1]>	<dbl[,1]>	<dbl[,1]>	<dbl[,1]>	<dbl[,1]>	<dbl[,1]>	<dbl[,1]>	<dbl[,1]>	...	<dbl[,1]>	<
21704	1	0	1	0	1	0	0	0	0	0	...	1	
1752	1	0	0	0	1	0	0	0	0	1	...	0	
7079	0	0	0	1	0	1	0	0	0	0	...	0	
5875	0	0	0	0	0	1	0	0	0	1	...	0	
11016	1	0	0	1	1	0	0	0	0	0	...	0	
5448	0	0	0	1	0	1	0	0	1	0	...	1	

In [17]:
```
df.processed <- read.csv("krkopt_preprocessed.csv")

head(df.processed)
```

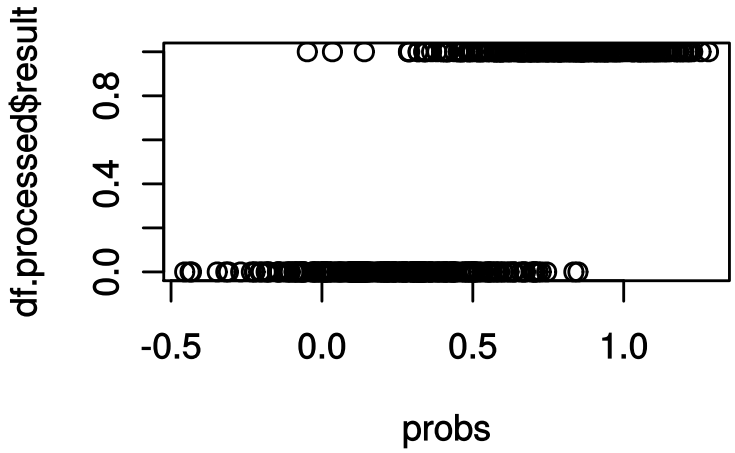
A data.frame: 6 x 35

result	wk_file_a	wk_file_b	wk_file_c	wk_rank_1	wk_rank_2	wk_rank_3	wr_file_a	wr_file_b	wr_file_c	...	bk_file_e	bk_file_f
<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	...	<int>	<int>
0	0	0	0	0	1	0	0	0	0	...	0	0
1	0	0	1	1	0	0	1	0	0	...	0	0
0	0	0	0	0	0	0	0	0	0	...	0	0
0	0	1	0	1	0	0	0	0	0	...	0	0
0	0	0	1	0	0	1	1	0	0	...	0	1
1	1	0	0	1	0	0	0	0	0	...	0	0

In [18]:
```
log_regress <- glm(df.processed)
probs <- predict(log_regress,type = 'response')
preds <- round(probs)
```

In [19]:
```
plot(df.processed$result ~ probs)
table(preds, df.processed$result)
```

```
preds   0   1
    0 210  34
    1  40 216
```



Question 2.b

The above model seems to do reasonably well, but let's see if we can do better using some variable selection.

You've been discussing your model with some of your colleagues, and they have made several suggestions:

- One colleague has suggested that it does not matter where the rook or the white king is (**only** where the black king is); that is, we only need to consider the following variables:
  - bk_rank_1, bk_rank_2, ..., bk_rank_7
  - bk_file_a, bk_file_b, ..., bk_file_g
- Another colleague has suggested that it only matters where the white rook and king are (and **not** where the black king is); that is, we only need to consider the following variables:
  - wr_rank_1, wr_rank_2, ..., wr_rank_7
  - wr_file_a, wr_file_b, ..., wr_rank_g,
  - wk_rank_1, wk_rank_2, wk_rank_3
  - wk_file_a, wk_file_b, wk_file_c
- Your opponent has heard what you've been up to, and they have suggested that it only matters where the white rook is (not the black or white king king). You don't trust them, but you'll give it a try... you only need to consider the following variables:
  - wr_rank_1, wr_rank_2, wr_rank_3
  - wr_file_a, wr_file_b, wk_file_c

For each of these three proposals, create a linear model which only contains those variables. Then calculate the accuracy of each of these models using the evaluate function provided.

In [20]:
```
evaluate <- function(fit, truth) {
    probs <- predict(fit, type="response")
    preds <- probs > 0.5
    accuracy <- mean(preds == truth)

    return (accuracy)
}
```

In [21]:
```
colnames(df.processed)
```

'result' 'wk_file_a' 'wk_file_b' 'wk_file_c' 'wk_rank_1' 'wk_rank_2' 'wk_rank_3' 'wr_file_a' 'wr_file_b' 'wr_file_c' 'wr_file_d' 'wr_file_e' 'wr_file_f' 'wr_file_g' 'wr_rank_1' 'wr_rank_2' 'wr_rank_3' 'wr_rank_4' 'wr_rank_5' 'wr_rank_6' 'wr_rank_7' 'bk_file_a' 'bk_file_b' 'bk_file_c' 'bk_file_d' 'bk_file_e' 'bk_file_f' 'bk_file_g' 'bk_rank_1' 'bk_rank_2' 'bk_rank_3' 'bk_rank_4' 'bk_rank_5' 'bk_rank_6' 'bk_rank_7'

In [22]:
```
black_fit <- glm(result~bk_rank_1+bk_rank_2+bk_rank_3+bk_rank_4+bk_rank_5+
                 bk_rank_6+bk_rank_7+bk_file_a+bk_file_b+bk_file_c+bk_file_d+
                 bk_file_e+bk_file_f+bk_file_g,
                 data = df.processed, family = binomial())

whiterooking_fit <- glm(result~wr_rank_1+wr_rank_2+wr_rank_3+wr_rank_4+
                        wr_rank_5+wr_rank_6+wr_rank_7+wr_file_a+wr_file_b+wr_file_c+
                        wr_file_d+wr_file_e+wr_file_f+wr_file_g+wk_rank_1+wk_rank_2+
                        wk_rank_3+wk_file_a+wk_file_b+
                        wk_file_c,data =df.processed,family = binomial())

white_fit <- glm(result~wr_rank_1+wr_rank_2+wr_rank_3+wr_rank_4+wr_rank_5+
                 wr_rank_6+wr_rank_7+wr_file_a+wr_file_b+wr_file_c+wr_file_d+
                 wr_file_e+wr_file_f+wr_file_g,data=df.processed,family=binomial())


paste0('Accuracy of model with black king only: ',evaluate(black_fit,df.processed$result))
paste0('Accuracy of model with white rook and king:',evaluate(whiterooking_fit,
                                                              df.processed$result))
paste0('Accuracy of model with white rook:',evaluate(white_fit,df.processed$result))
```

'Accuracy of model with black king only: 0.798'

'Accuracy of model with white rook and king: 0.686'

'Accuracy of model with white rook:0.588'

We can also use an automated, step-wise process to determine remove variables, using R's built-in step() function.

In [23]:
```
to.stepwise <- function(df, k) {
    fit <- glm(result ~ ., family="binomial", data=df)
    return (step(fit, k=k))
}
```

In [24]:
```
fit.aic <- to.stepwise(df.processed, k =2)
fit.bic <- to.stepwise(df.processed, k = log(nrow(df.processed)))


# your code here
```

```
- wk_rank_3   1    307.97 373.97
- wr_file_g   1    308.06 374.06
- wr_rank_1   1    308.16 374.16
- bk_file_b   1    308.61 374.61
- wr_file_e   1    308.70 374.70
- wr_file_a   1    309.00 375.00
<none>              307.33 375.33
- wr_rank_7   1    309.51 375.51
- wk_rank_2   1    310.34 376.34
- wr_file_c   1    310.48 376.48
- bk_rank_4   1    310.60 376.60
- bk_rank_6   1    310.88 376.88
- bk_file_a   1    311.25 377.25
- wr_rank_5   1    311.57 377.57
- wr_rank_3   1    313.41 379.41
- wk_file_c   1    313.96 379.96
- bk_rank_7   1    316.65 382.65
- wk_file_a   1    317.17 383.17
- bk_rank_5   1    317.43 383.43
```

In [25]:
```
evaluate(fit.aic, df.processed$result)
evaluate(fit.bic, df.processed$result)
```

0.86

0.856

AIC and BIC models limit the level of overfitting by removing the unimportant features one by one to find the correct set of features. There are less parameters when comparing BIC to AIC method due to BIC's higher penalty compared to AIC.

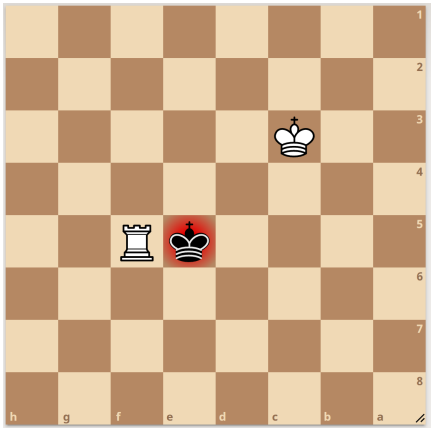
All of my colleagues were wrong. Despite black king being the most important parameter based on one colleague, other positions (white rook and white king) do have relevance too as AIC and BIC models have included parameters these variables too.

BIC result consisted the most parameters (10 out of 17 parameters)for black king (bk_file_a , bk_file_c , bk_file_d , bk_file_e , bk_file_f , bk_file_g , bk_rank_1 , bk_rank_2 , bk_rank_5 , bk_rank_7). Hence, it is important to have the black king in the right spot to determine the outcome of the game.

Question 2c

You have finished your model, and have decided to use the AIC model (which should be saved as a variable named "fit.aic"). You now want to apply your model to the chess position you find yourself in against your colleague.

You are in the following position (it's your move, i.e. black's move):



Note: the above image will only show if you include the file "chess_pos.png" (contained in the "data.zip" file you've been given on Moodle) in the same folder as this Jupyter notebook, and then re-run this Markdown cell by putting it into edit mode and then running it. Alternatively, you can just look at the file itself.

This position has the following variable values:

- wk_file = c
- wk_rank = 3
- wr_file = f
- wr_rank = 5
- bk_file = e
- bk_rank = 5

You will need to convert this position into a dataframe of one row with the one-hot encoding we used above, and then pass this into the predict function using the model "fit.aic" and using type="response" for the predict function to get a probability.

Do this in the cell below and see what result you get. Then, in the Markdown cell below that, state whether your model indicates that you truly should resign in this position (i.e. the position is lost) or whether your opponent was bluffing (i.e. the position does not seem to be winnable in less than 10 moves).

In [26]:
```
# your code here

# creating a dataframe called empty df which takes out result column
empty_df <- df.processed[,!names(df.processed) %in% "result"][FALSE,]


# for columns wk_file_c, wk_rank_3,wr_file_f,wr_rank_5, bk_file_e and bk_rank_5
# making the first row as value 1
empty_df[1,'wk_file_c'] <- 1
empty_df[1,'wk_rank_3'] <- 1
empty_df[1,'wr_file_f'] <- 1
empty_df[1,'wr_rank_5'] <- 1
empty_df[1,'bk_file_e'] <- 1
empty_df[1,'bk_rank_5'] <- 1


# for columns which are empty replace it with value 0
empty_df[is.na(empty_df)] <- 0

#
probability <- predict(fit.aic,empty_df,type = 'response')
prediction <- ifelse(probability >0.5,1,0)

paste0("the prediction value:",prediction)
```

'the prediction value:1'

YOUR ANSWER HERE

Given that my prediction value was 1, we can say that my opponent was bluffing (i.e. the position does not seem to be winnable in less than 10 moves).

Question 2d

In our code above (and also in the code in Question 3), we train our models using all of our training data, and also evaluate the models on our training data. Describe a problem with this approach, and give an example of how we could fix this (other than leave-one-out cross-validation, which is used in Question 3).

YOUR ANSWER HERE

There is a high probability of overfitting the data given the introduction of a biased evaluation of our model. This is so as the training data used to build the model is used to assess the fit of the data our model was based upon.

To fix this, we can use k*l-fold cross validation with the data split into 'k' sets. The outer sets will be the training sets and used as a validation set, whereas the inner training sets will be used to build the model. This can be repeated for multiple hyperparameters and the validation set can be used to determine the best hyperparameters for the inner training sets.

Question 3 - Linear Regression

Just as you give your opponent the finishing move that draws your chess game (which is no small feat against an Australian champion - you have the admiration of your colleagues), the engineers come running into your office in a panic. They tell you that their deadline for finishing their 3D printing process is tomorrow, but that they are still having trouble solving a problem with their printing process.

They have printed 50 test cases, with varying degrees of success. They have found that a large number of their cases turn out very rough to the touch; they are concerned that management will see this and scrap the 3D printing project altogether (and their jobs with it!).

You calmly ask them to send over all the data they have, and get to work developing a linear regression model to try to identify which component(s) of the process are causing the roughness problem.

In [28]:
```
data.3d <- read.csv("data.csv")
data.3d <- data.3d[,-c(11, 12)]
head(data.3d)
```

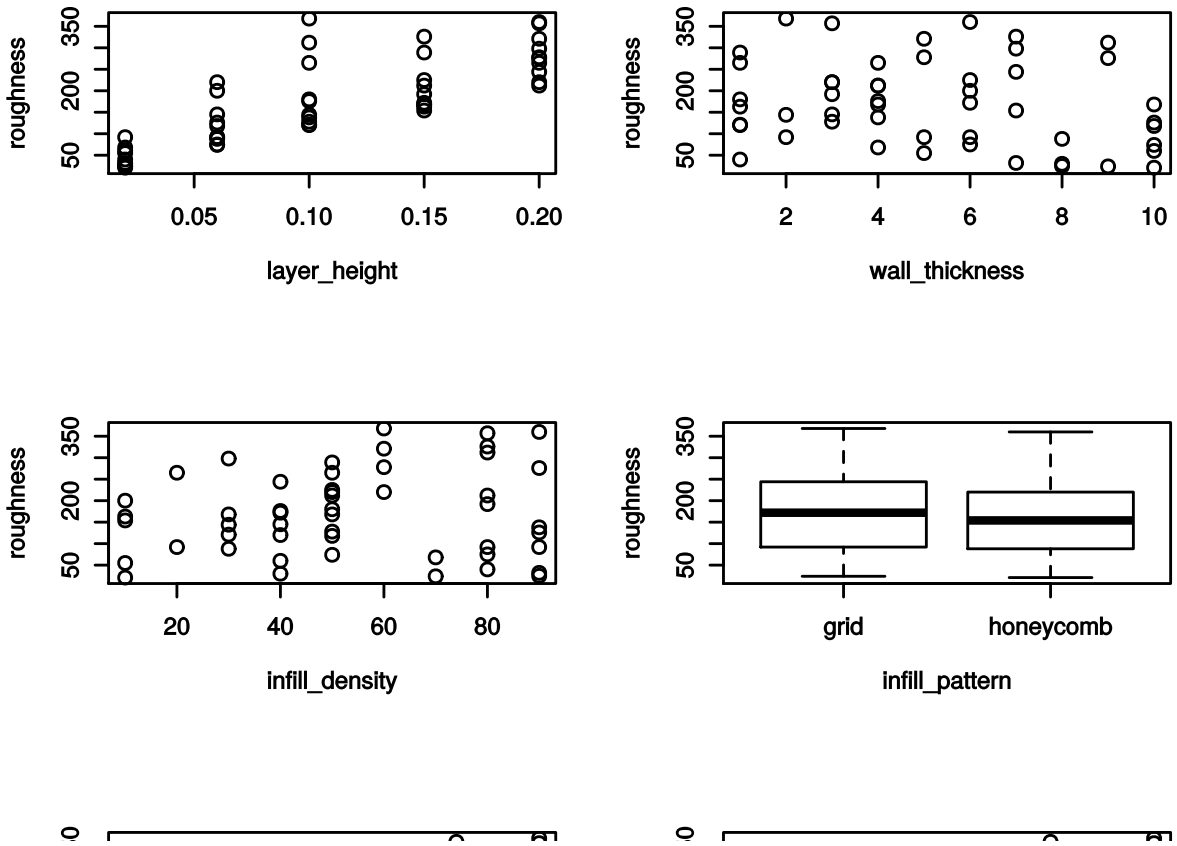
A data.frame: 6 x 10

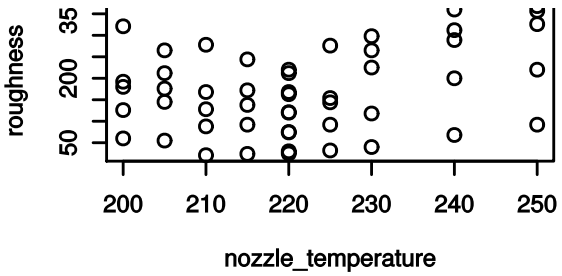
layer_height	wall_thickness	infill_density	infill_pattern	nozzle_temperature	bed_temperature	print_speed	material	fan_speed
<dbl>	<int>	<int>	<fct>	<int>	<int>	<int>	<fct>	<int>
0.02	8	90	grid	220	60	40	abs	0
0.02	7	90	honeycomb	225	65	40	abs	25
0.02	1	80	grid	230	70	40	abs	50
0.02	4	70	honeycomb	240	75	40	abs	75
0.02	6	90	grid	250	80	40	abs	100
0.02	10	40	honeycomb	200	60	40	pla	0

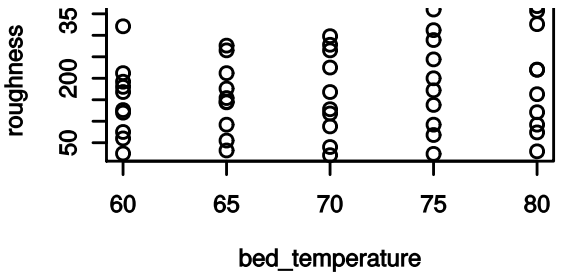
First things first - we need to take a look at our data. Write a function "plot.variables" which takes two arguments - var.x and var.y. It then plots var.x vs var.y. Make sure your plots have a reasonable label (e.g. var.x vs var.y, substituting in the actual values of var.x and var.y) as well as reasonable x-axis and y-axis labels (e.g. var.x on the x-axis and var.y on the y-axis, again substituting the actual values of var.x and var.y).

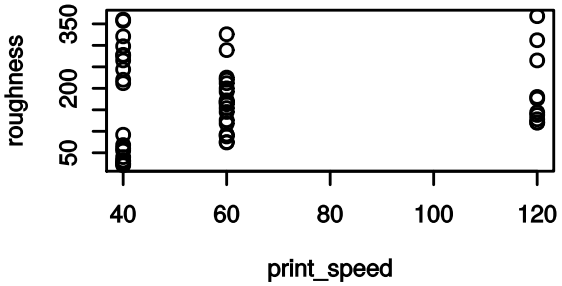
In [29]:
```
plot.variables <- function(var1, var2) {
    var_1 = data.3d[[var1]]
    var_2 = data.3d[[var2]]
    model <- plot(var_1, var_2, xlab = var1, ylab = var2)
    return(model)

}
```

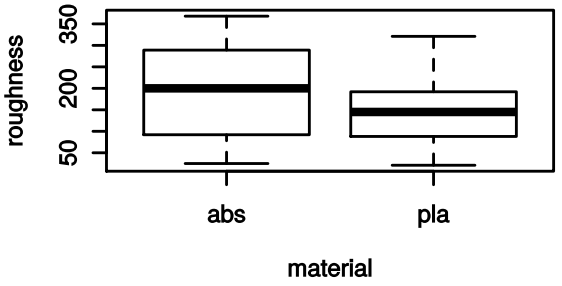
In [30]:
```
par(mfrow=c(3, 2))

options(repr.plot.width=6, repr.plot.height=6)

for (name in names(data.3d)) {
    if (name != "roughness") {
        plot.variables(name, "roughness")
    }
}
```

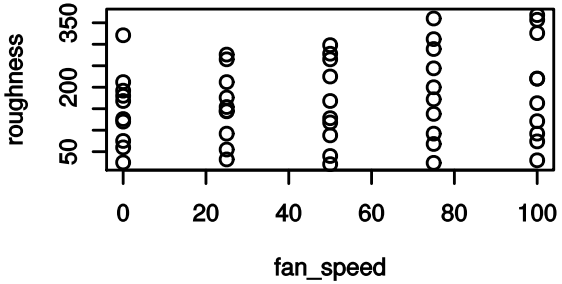



roughness
50
200
350
200
210
220
230
240
250
nozzle_temperature


roughness
50
200
350
60
65
70
75
80
bed_temperature


roughness
50
200
350
40
60
80
100
120
print_speed


roughness
50
200
350
abs
pla
material


roughness
50
200
350
0
20
40
60
80
100
fan_speed

Now we have an idea of what our dataset looks like, we can start working on creating a linear model. Using R's built-in lm() function, we create a linear model of roughness vs all the other variables in our dataset. This model is saved in a variable called "fit.lm"

In [31]:
```
fit.lm <- lm(formula =  roughness~., data = data.3d)
```

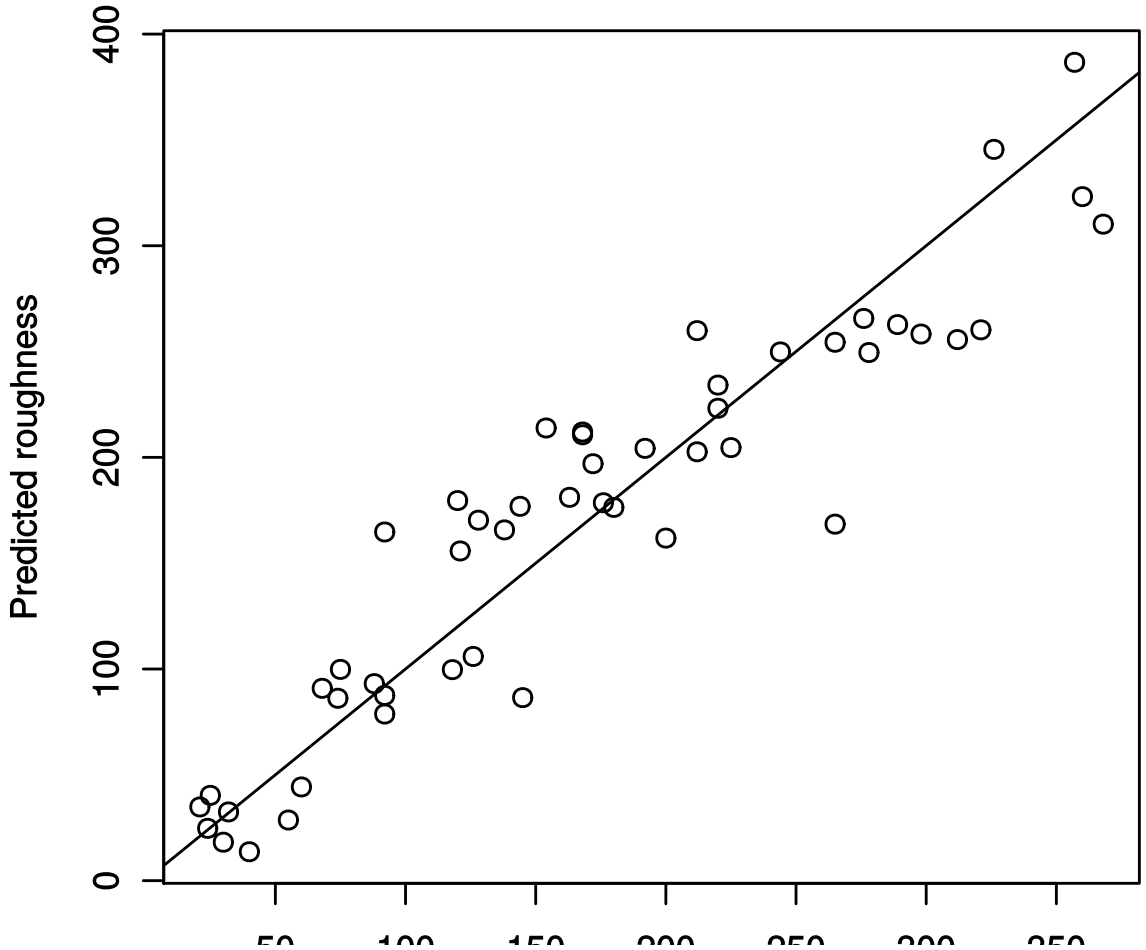
In [32]:
```
fit.lm
```

```
Call:
lm(formula = roughness ~ ., data = data.3d)

Coefficients:
            (Intercept)             layer_height          wall_thickness
              -2.371e+03                1.269e+03                2.334e+00
         infill_density  infill_patternhoneycomb      nozzle_temperature
              -4.231e-02               -1.255e-01                1.506e+01
        bed_temperature             print_speed             materialpla
              -1.613e+01                6.496e-01                2.985e+02
             fan_speed
                    NA
```

Below we are creating a plot of the predicted roughness (using our linear model) on the x-axis vs the actual roughness on the y-axis. This is to get an idea of how our estimates compare with the true roughness values using a graph.

In [33]:
```
predict_values <- predict(fit.lm)
```

In [34]:
```
plot(data.3d$roughness,predict_values,xlab="Actual roughness",ylab="Predicted roughness")
abline(a=0,b=1)
```



The graph seems to indicate that our model does reasonably well, but we would like to have more mathematically-sound way of determining how close our model gets, on average. This is done using mean squared error and mean absolute error.

In [35]:
```
install.packages("Metrics")
library(Metrics) # library to use MSE and MAE function
```

```
Installing package into '/home/nbuser/R'
(as 'lib' is unspecified)
```

In [36]:
```
mean.sq.error <- function(model, df) {
    values <-mse(predict(model),df$roughness)
    return(values)
}
```

In [37]:
```
mean.abs.error <- function(model, df) {
    values <- mae(df$roughness, predict(model))
    return(values)
    }
```

In [38]:
```
mean.sq.error(fit.lm, data.3d)
mean.abs.error(fit.lm, data.3d)
```

1199.35220414104

27.6021083464572

We need to narrow down what could be the cause for print roughness; we will use some variable selection to do this. Step wise variable selection will be performed with values of k for the AIC (k = 2) and BIC (k = log(N) for N data points in our dataset).

In [39]:
```
to.stepwise <- function(df, k) {
    fit <- lm(roughness ~ ., data=df)
    return (step(fit, k=k))
}
```

In [40]:
```
fit.lm.aic <- to.stepwise(data.3d,k=2)

fit.lm.bic <- to.stepwise(data.3d,k= log(nrow(data.3d)))
```

```
                     Df Sum of Sq    RSS    AIC
- wall_thickness      1      1651  61667 379.35
<none>                             60016 381.90
- print_speed         1     14651  74667 388.91
- bed_temperature     1     37456  97472 402.24
- material            1     39029  99045 403.04
- nozzle_temperature  1     54242 114258 410.18
- layer_height        1    307270 367285 468.57

Step:  AIC=379.35
roughness ~ layer_height + nozzle_temperature + bed_temperature +
    print_speed + material

                     Df Sum of Sq    RSS    AIC
<none>                             61667 379.35
- print_speed         1     13210  74877 385.14
- bed_temperature     1     36693  98360 398.78
- material            1     38454 100121 399.67
- nozzle_temperature  1     53228 114895 406.55
- layer_height        1    314770 376437 465.88
```

In [41]:
```
mean.sq.error(fit.lm, data.3d)
mean.sq.error(fit.lm.aic, data.3d)
mean.sq.error(fit.lm.bic, data.3d)

mean.abs.error(fit.lm, data.3d)
mean.abs.error(fit.lm.aic, data.3d)
mean.abs.error(fit.lm.bic, data.3d)
```

1199.35220414104

1233.34041020578

1233.34041020578

27.6021083464572

27.7549872191008

27.7549872191008

Both the AIC and BIC models identify the same formula, and therefore remove the same variables:

```
roughness ~ layer_height + nozzle_temperature + bed_temperature + print_speed + materia
l
```

In [42]:
```
summary(fit.lm)
```

```
Call:
lm(formula = roughness ~ ., data = data.3d)

Residuals:
    Min      1Q  Median      3Q     Max 
-72.746 -24.332  -1.641  20.304  96.552 

Coefficients: (1 not defined because of singularities)
                          Estimate Std. Error t value Pr(>|t|)    
(Intercept)             -2.371e+03  3.716e+02  -6.379 1.25e-07 ***
layer_height             1.269e+03  8.765e+01  14.483  < 2e-16 ***
wall_thickness           2.334e+00  2.189e+00   1.066  0.29259    
infill_density          -4.231e-02  2.341e-01  -0.181  0.85742    
infill_patternhoneycomb -1.255e-01  1.128e+01  -0.011  0.99117    
nozzle_temperature       1.506e+01  2.529e+00   5.953 5.05e-07 ***
bed_temperature         -1.613e+01  3.251e+00  -4.962 1.27e-05 ***
print_speed              6.496e-01  2.060e-01   3.153  0.00302 ** 
materialpla              2.985e+02  5.836e+01   5.114 7.78e-06 ***
fan_speed                       NA         NA      NA       NA    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 38.24 on 41 degrees of freedom
Multiple R-squared:  0.8752,    Adjusted R-squared:  0.8509 
F-statistic: 35.95 on 8 and 41 DF,  p-value: 3.834e-16
```

In [43]:
```
summary(fit.lm.aic)
```

```
Call:
lm(formula = roughness ~ layer_height + nozzle_temperature + 
    bed_temperature + print_speed + material, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-74.084 -26.500  -1.662  22.585  92.356 

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)    
(Intercept)        -2310.7356   353.2009  -6.542 5.38e-08 ***
layer_height        1246.5353    83.1780  14.986  < 2e-16 ***
nozzle_temperature    14.7774     2.3979   6.163 1.95e-07 ***
bed_temperature      -15.8078     3.0895  -5.117 6.55e-06 ***
print_speed            0.5538     0.1804   3.070  0.00366 ** 
materialpla          294.1610    56.1586   5.238 4.38e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 37.44 on 44 degrees of freedom
Multiple R-squared:  0.8717,    Adjusted R-squared:  0.8571 
F-statistic: 59.78 on 5 and 44 DF,  p-value: < 2.2e-16
```

In [44]:
```
summary(fit.lm.bic)
```

```
Call:
lm(formula = roughness ~ layer_height + nozzle_temperature + 
    bed_temperature + print_speed + material, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-74.084 -26.500  -1.662  22.585  92.356 

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)    
(Intercept)        -2310.7356   353.2009  -6.542 5.38e-08 ***
layer_height        1246.5353    83.1780  14.986  < 2e-16 ***
nozzle_temperature    14.7774     2.3979   6.163 1.95e-07 ***
bed_temperature      -15.8078     3.0895  -5.117 6.55e-06 ***
print_speed            0.5538     0.1804   3.070  0.00366 ** 
materialpla          294.1610    56.1586   5.238 4.38e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 37.44 on 44 degrees of freedom
Multiple R-squared:  0.8717,    Adjusted R-squared:  0.8571 
F-statistic: 59.78 on 5 and 44 DF,  p-value: < 2.2e-16
```

We note that the variables removed by AIC and BIC have large p values. Variables like wall_thickness, infill_density, infill_patternhoneycomb which isnt statistically significant has been removed. These 3 values have large p values as their p values are larger than alpha (0.05).

It is a problem that we are training and evaluating on the same dataset

One way to mitigate this problem is by using "leave-one-out cross validation", or LOOCV (https://en.wikipedia.org/wiki/Cross-validation_(statistics)#Leave-one-out_cross-validation) (https://en.wikipedia.org/wiki/Cross-validation_(statistics)#Leave-one-out_cross-validation)). In LOOCV, for a dataset of size N we train N different models, where each model trains on all but one of the points in our dataset. We then evaluate the squared error from each model predicting the point we left out (hence the name "leave-one-out" CV), and average the final result.

In the code cell below, we've implemented a function "cv.loo", which takes as argument a dataset "df" and returns the average **squared prediction error** from LOOCV done on that dataset.

In [46]:
```
cv.loo <- function (df) {
    square_prediction_error <- numeric(50)

    for (number in 1:50){
        test_df<- df[number,]
        train_df <- df[-c(number),]
        value <- test_df$roughness

        linear_model <- lm(roughness~.,data = train_df)
        prediction <- predict(linear_model,newdata= test_df)

        square_error <- (value-prediction)**2
        square_prediction_error[number] <- square_error


    }
    mean_square_error = mean(square_prediction_error)
    return(mean_square_error)

}
```

In [47]:
```
cv.loo(data.3d)
```

```
Warning message in predict.lm(linear_model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
```

```
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
```

```
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”Warning message in predict.lm(linear_
model, newdata = test_df):
“prediction from a rank-deficient fit may be misleading”
```

1831.22985575416

The LOOCV error is higher than training MSE as the test data in LOOCV doesnt consist of the training data set hence it is more likely to get error.

We can reduce MSE by increasing the size of the sample.

With reference to the original model, the variables that do not appear to be related to roughness is wall_thickness, infill_density, infill_patternhoneycomb and fan_speed due it's large p value making it statistically insignificant. On the other hand, the variables that are related to roughness is layer_height , nozzle_temperature , bed_temperature , print_speed and material.

We can determine what to do to reduce the roughness of 3d prints by observing the coefficient of the statistically significant variables individually from the AIC summary. To reduce the roughness, we can reduce the layer height by 1 micron which will lead to a decrease in roughness by 1246.53 unit (given coefficient of 1246.53). We can also increase the nozzle temperature by 1 celcius which will lead to a decline in roughness by 14.77 unit (given coefficient of 14.77). An increase in bed temperature would lower the roughness by 15.80(given coefficient of -15.80)