Linear Regression

what is predictive modelling?

If somebody asks you to guess how Virat Kohli is going to perform in next cricket match against Australia, you can take a guess *[Assuming you are cricket follower]*. This guess is based on your knowledge of his past performances in these or similar conditions and against this kind of team.

Predictive modelling gives this process a formal framework. It gives you tools to extract mathematical equations/rules from the past data to predict future results. Before you draw any conclusions about **predictive modeling** being some kind of black magic, lets list down its limitations:

1. The models *[equations/rules]* are dependent on the past data that you have. If data is bad , your predictive models are also going to be bad.
2. Every model will have errors associated with its predictions. Better the model, lesser the error, but it will never be an exact estimate for all practical purposes.
3. Model will be good only until underlying factors on which it was based on , do not change behavior. For example: A predictive model which was built to predict a particular share performance in good economic conditions will perform rather poorly in recession.

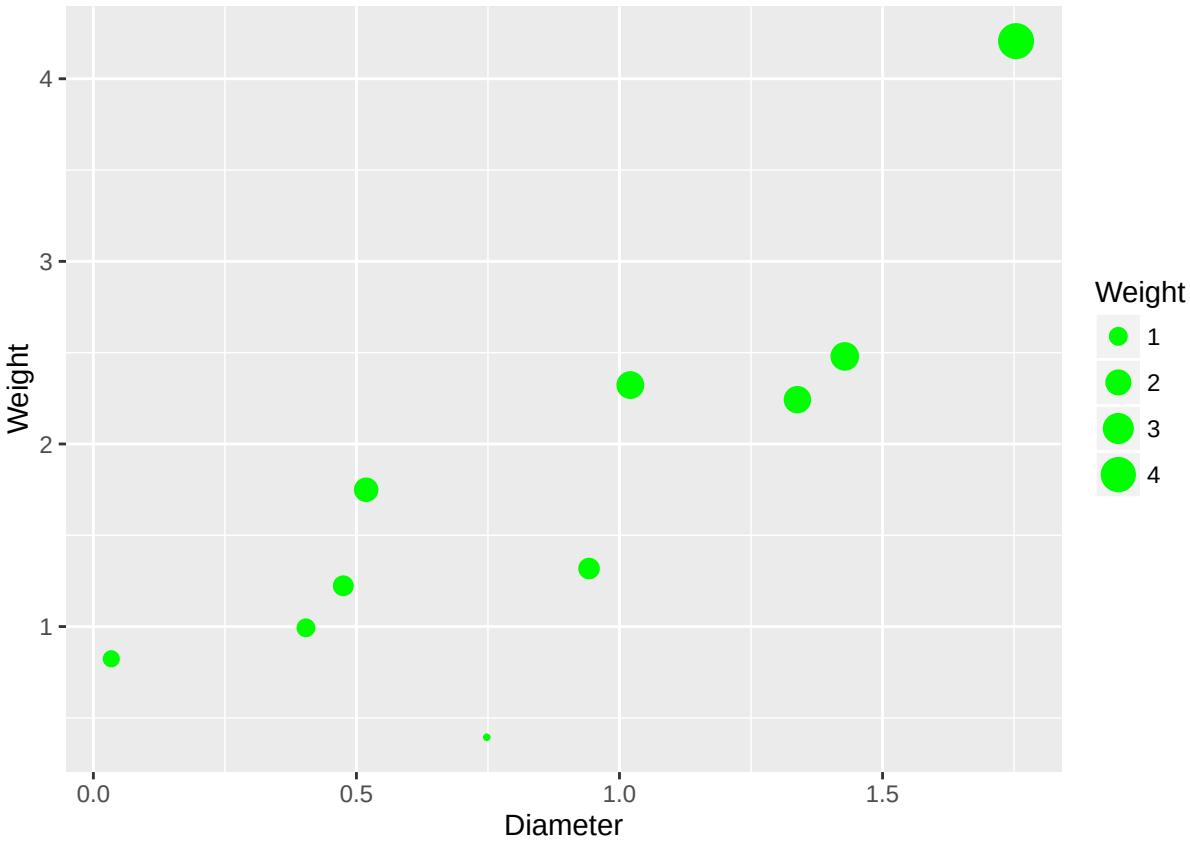
Before we right away jump in to predictive modelling and start extracting those said equations, lets first figure out what really leads us there:

Correlation

When we say that, given past data we can predict future results/outcomes, we are essentially relying on our hunch that we can observe some other factor which affects the outcome and we can leverage that information. For example: when you pick up an apple and guess it weight, you are betting on your assumption that weight of an apple is dependent on its size *[or diameter]*.

In other words , you are assuming that weight of that apple is **correlated** with its diameter.

```
## Warning: package 'ggplot2' was built under R version 3.2.4
```



As you can see in the figure, you were not really wrong. To make this more concrete we need to figure out a way to quantify this *correaltion* . Before doing that, we also need to understand different kind of correlations. As observed in the figure above, weight of apple goes up as its diameter increases. On looking more closely you find out that, increase in weight of apples is happening in *possibly* constant multiples of increase in diameter. This is called **linear correlation**. There can be other form of correlations as well. (Figure 1)

What do we mean by these *linear* and *other* forms of correlations is that one variable *[lets say y]* can be written as a function of another *[lets say x]*

- Linear Correlation : $y = ax + b$
- Exponential Correlation : y= a*exp(x) + b

quantifying correlation coefficient (Figure 2)

This formula is designed to measure strength of **linear** correlation. it has following properties:

- $-1 < r < 1$
- It takes -ve values of negative correlation and +ve values of +ve correlation *Note: -ve correlation between x and y means , when x increases y decreases and vice versa*
- Correlation is strong when absolute value of r is close to 1, it is weak if the absolute value is close to zero
- Value of r doesn't change if you linearly transform any or both of the variables. Meaning, correlation between x and y will be same as correlation between (ax+b) and (Ay+C).
- As emphasized above , it can be used to measure linear correlation only

But this formula being capable of measuring only linear correlation is not limited. look at this equation again:

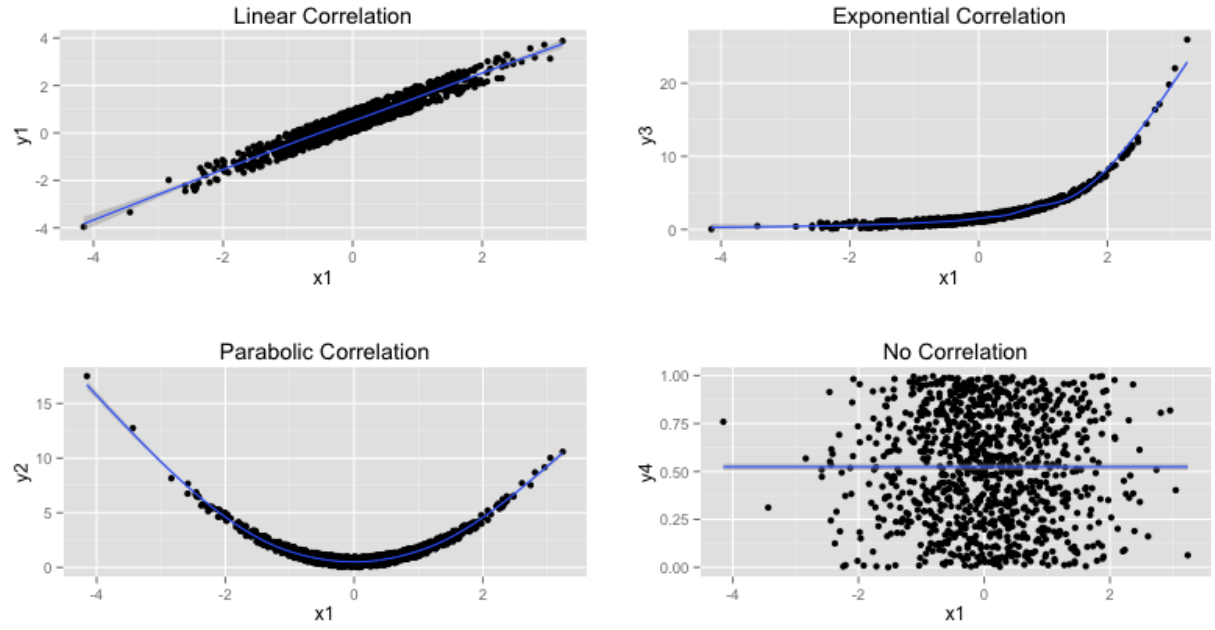


Figure 1: Linear and Non-Linear Correlation

$$r = \frac{\sum_{i=1}^{n} \left(\left(x_i - \overline{x} \right) \left(y_i - \overline{y} \right) \right)}{\sqrt{\sum_{i=1}^{n} \left(x_i - \overline{x} \right)^2 \sum_{i=1}^{n} \left(y_i - \overline{y} \right)^2}}$$

Figure 2: correlation formula

- y= a*e^x + b

consider that x' = e^x . You can write the above as this :

- y= a*x' + b

which is same as saying that y and x' are linearly correlated. In a similar manner, if you observe that y and x are non-linearly related , you can apply a suitable transformation to either of x and y and make the relationship linear.You can then measure the strength of correlation between y and x' *[transformed variable]*

Correlation and Causation (Figure 3)

Causation is when a particular factor is the reason for change in another factor. For example number of people buying sun-screen in the city and city's temperature are going to be correlated. Also there is direct causation. Temperatures going up *[Hot Sunny Weather]* is driving sales of sun-screen. However if two factors are correlated , that doesn't guarantee that there will be causation. For example look at following figure :

Figure 3: Correlation and Causation

Clearly you can see that ice-cream sales and shark attacks are correlated as far as the data can be seen. However that doesn't mean that ice-cream sales are cause of shark attacks. In fact rising temperature cause more people to buy ice-cream and also it causes people to go to beaches in larger numbers *[and sometimes subsequently get attacked by sharks]* . This clarifies two things:

- Clearly correlation doesn't necessarily means causation
- However it does indicate that correlated factors might have a common underlying cause *[Again, not always neccessary]*

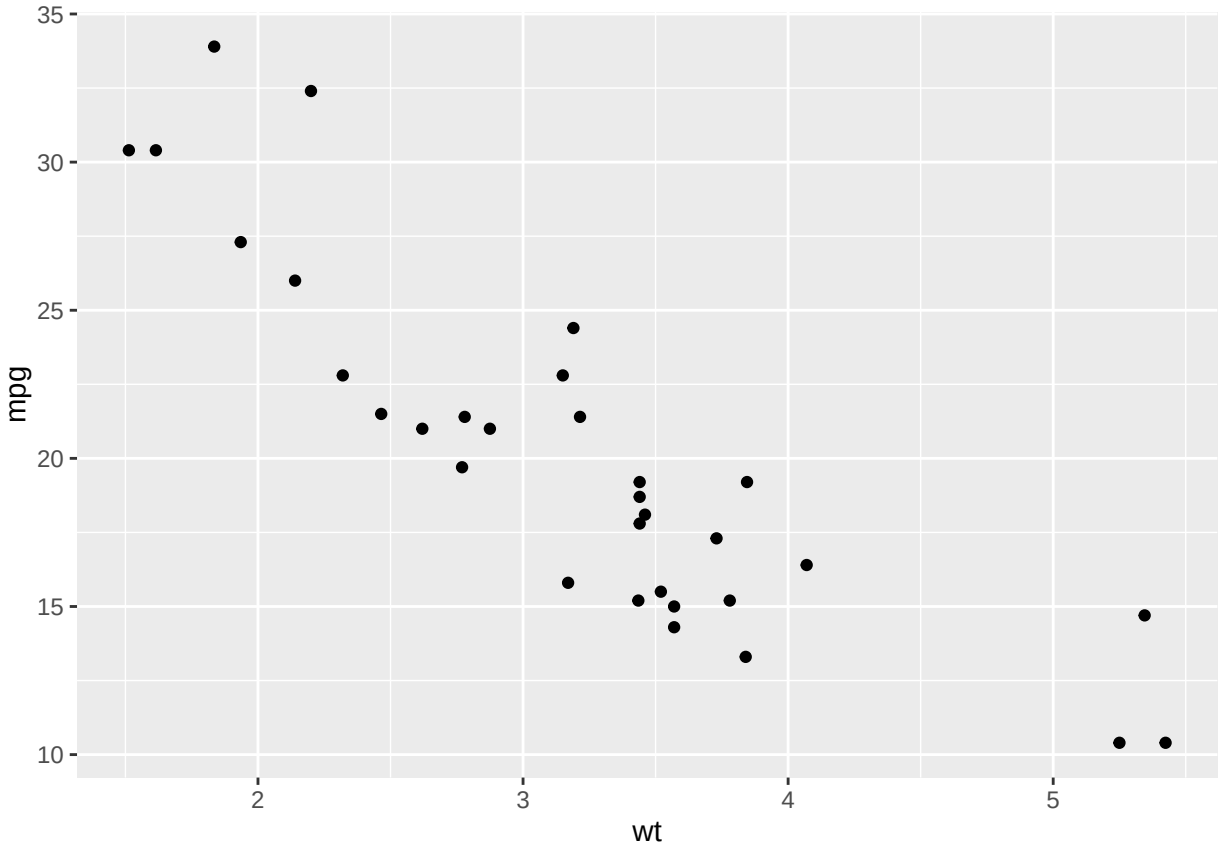
Finding Correlation in R : Correlation Coefficient and Scatter Plots

```
cor.test(mtcars$mpg,mtcars$wt)
```

```
##
##  Pearson's product-moment correlation
##
## data:  mtcars$mpg and mtcars$wt
## t = -9.559, df = 30, p-value = 1.294e-10
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.9338264 -0.7440872
## sample estimates:
##       cor
## -0.8676594
```

This tells you that correlation exists and it is significant. Also the sign of correlation coefficient is -ve. Which tells you that as weight of a vehicle increases , mileage goes down. Lets look at that visuall with the help of a scatter plot.

```
ggplot(mtcars,aes(x=wt,y=mpg))+geom_point()
```



This tells you the same story, although without numbers.

Simple Linear Regression

Well, correlation coefficient lets us figure whether a pair of variables are affecting each other and if they are then how strong is that effect. However , what we eventually want is an equation which can be used to predict value of y [my response/target/outcome], given a value of x [my input/predictor].

As we have seen earlier , if two variables are linearly correlated we can essential draw line through their scatter plot which depicts the relationship between them. But the problem is we can draw many lines, and until now have no clue as to which one to chose finally. (Figure 4)

However , you can observe few crucial things here :

- It's impossible to come up with a line which passes through all the points, in other words ; whatever line equation you come up with, there are going to be errors associated with it.
- Take general equation of a line ; $y = \beta_0 + \beta_1 * x$, what is changing between these lines in the figure is the value of the parameters b0 and b1 . We need to find out such values of these parameters for which error is minimum.

In the figure below, red points on the line are your predictions for values of y given some values of x, whereas blue points are the actuals observed for those values of x. (Figure 5)

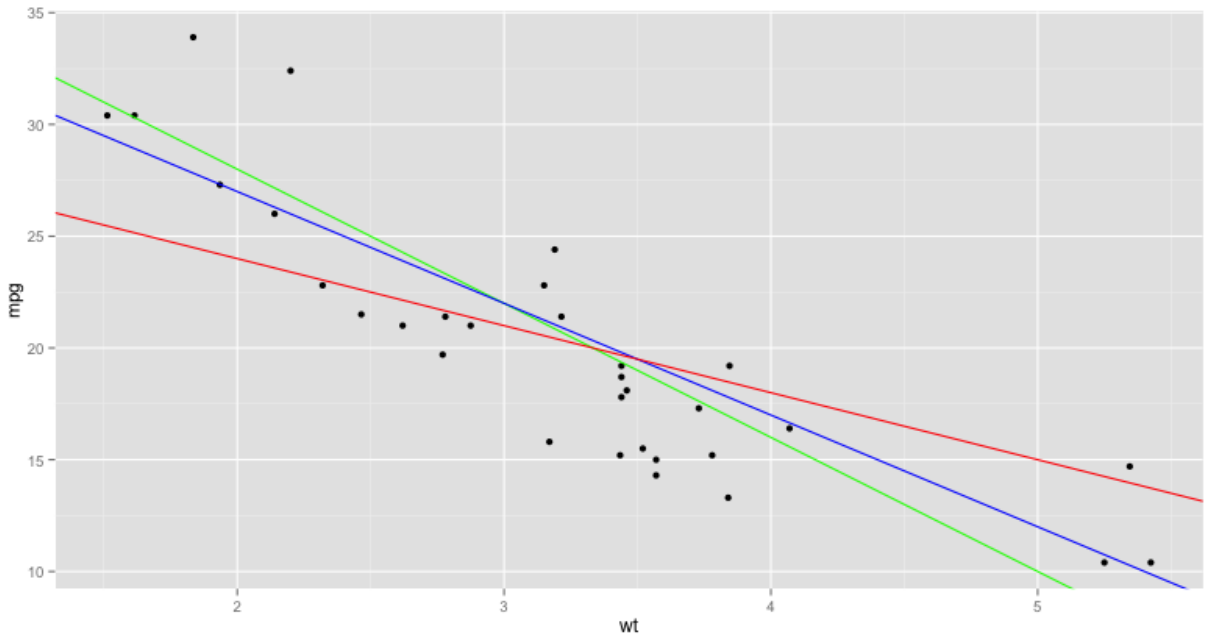


Figure 4: Which Lines to Pick

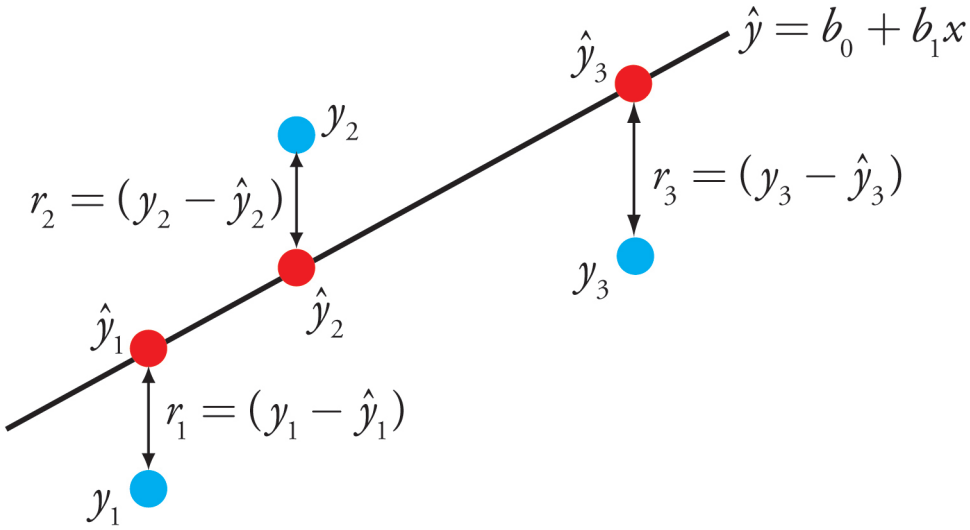


Figure 5: Errors in Predictions

After looking at the figure above and the crucial observations that we mentioned, you must have realized that actual value of y can be written as addition of predicted and associated error.

$$y_i = \hat{y_i} + e_i$$

$$y_i = \beta_0 + \beta_1 * x_i + e_i$$

from here we can see that:

$$e_i = y_i - \beta_0 - \beta_1 * x_i$$

What we need to minimize is the collective error for entire data.

$$\sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 * x_i)^2$$

Now if we want to minimize the above quantity w.r.t. β_0 and β_1 then we can differentiate that equation and put it equal to zero to find values of β_0 and β_1 for which $\sum_{i=1}^{n} e_i^2$ is minimum. Resulting equations are also called normal equations. Here they are:

$$-2 * \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 * x_i) = 0$$

and

$$-2 * \sum_{i=1}^{n} x_i * (y_i - \beta_0 - \beta_1 * x_i) = 0$$

upon solving them you'll get result for β_0 and β_1 which can also be written like this:

$$\beta_1 = r_{xy} * (s_x / s_y)$$

and

$$\beta_0 = \bar{y} - \beta_1 * \bar{x}$$

Don't get intimidated by all these equations, at no stage as a business analyst you'll need to create or solve these equations.Software *[SAS or R or python]* will do this for you. Don't worry and read on.

So lets say your software gave you the appropriate values of β_0 & β_1 and you have your predictive model equation ready. But think, that whatever variable data you pass to this mathematical framework you'll get some values of β_0 & β_1, does that ensure you have a **good** model? Not necessarily .

Imagine , in absence of any such equation, what is your best guess for y . Its $\bar{y}$ the average value. But with this guess there is error associated with each observation : $y_i - \bar{y}$. Writing this in collective form :

SST= Total sum of squares = Total Variability = $\sum_{i=1}^{n} (y_i - \bar{y})^2$

This is also called total variability in target. We intend to bring this down with our model. In other words, we want to explain this variability with our model. Lets try to understand this with this figure:(Figure 6)

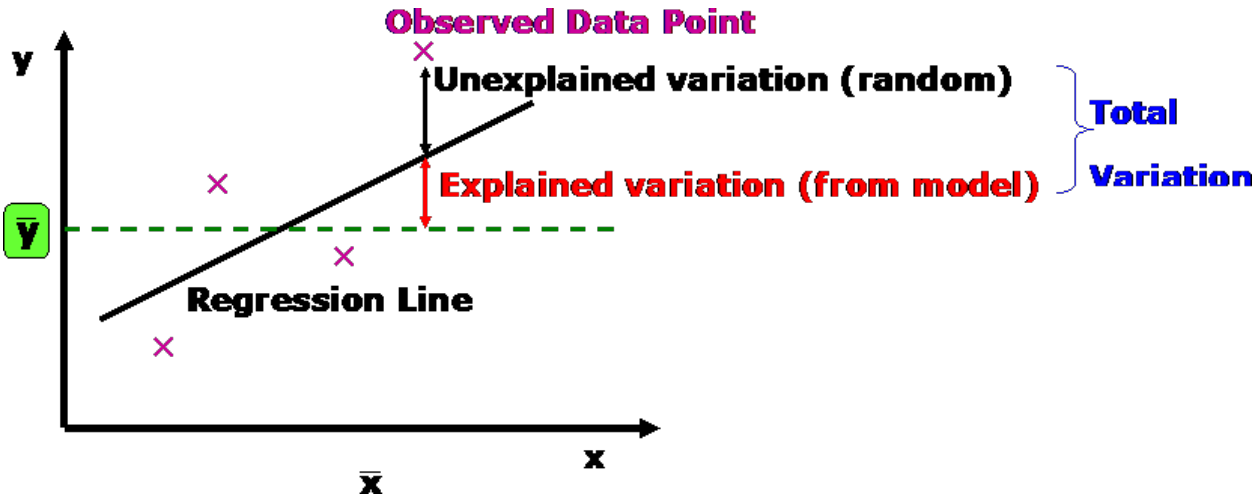


Figure 6: R square

Consider that our predicted values are $\hat{y_i}$. As discussed earlier , they don't explain entire variability in the target but a part instead. which is $\hat{y_i} - \bar{y}$.

SSR = Regression sum of squares = Explained Variability = $\sum_{i=1}^{n}(\hat{y_i} - \bar{y})^2$

You never get a perfect model. After prediction, each y_i has still error in prediction being equal to $y_i - \hat{y_i}$.

SSE = Error sum of squares = Unexplained Variability = $\sum_{i=1}^{n}(y_i - \hat{y_i})^2$

Ideally you'd want explained variability to be equal to total variability. That'll be a perfect model. Ratio between explained and total variability is defined as coefficient of determination, it is written as R^2. Its an indicator of how good your model is.

$$R^2 = SSR/SST$$

As you can see R^2's maximum value is 1 when SSR=SST . Also its minimum value would be 0 when SSR=0 . The closer to 1 , R^2's value is, better is your model.

OK, so now we can estimate parameters for our linear model equation, also we can measure how good our model is, but we haven't done anything about errors. Of course there is no way to predict them *[That's why they are errors!]*

What we can do is to make some kind of probabilistic estimate for them.

Assumption of Normality and its consequences

If we assume that $e_i \sim N(0, \sigma_i^2)$, then for each i_{th} Observation we can come up with some confidence interval around our prediction. For example we can say that each i_{th} response would lie in $y_i \pm 3 * \sigma_i$ with 99.7% confidence.

In addition to this quantification of errors , normality assumption allows us to build a very useful hypothesis. Before we discuss that, lets answer a question. We have seen this that if we supply some random y and x variable to this mathematical frame work, we'd get some non-zero estimate of β_0 & β_1 . Does that mean that y *[our response/target/DV]* really depends on x and it follows the equation $y = \beta_0 + \beta_1 * x$?

If y did not depend on x then estimate for β_1 should be zero. But we rarely see that with real data. Estimate might be close to zero but never exactly zero. So when should we conclude that parameter estimate for our β is significantly different from zero?

Now lets look at the hypothesis that the normality assumption enables us to build.

$$H_0 : \beta_i = 0$$
$$test - statistic : \frac{\beta_i}{S_{\beta_i}} \sim t - distribution$$

We can look at the p-values for the test, and if they turn out to be greater than alpha *[standard value 0.05]* , we can conclude that null hypothesis is true and parameter estimate is non-zero by chance and should be discarded.

Assumption of Homoscedasticity

We can estimate error variance σ_i for each x_i from our past data, because we very likely will have multiple observation for x_i. But we are building this model to predict values of y for not yet seen values of x, for such values of x there is no way for us to estimate error variance. We'll have to assume that error variance remains constant across all values of x . This assumption of constant variance across all values of predictor variable *[x]* is called homoscedasticity assumption.

Assumption of error independence

Errors from a model are those part of our response which we could not predict. Whatever part fell in a pattern became part of our model. Errors on the other hand are random and don't follow a pattern. If you plotted error with your target you should ideally see that there is no apparent pattern.

How to check validity of assumption and cosequences of violation

1. Normality Assumption: If errors don't follow normal distribution, we can not rely on p-values and confidence intervals for our predictions. However point estimates for target remain unaffected apart from the fact that parameter estimates might be non-zero by chance. We can check whether this is true or not by looking at qqplot for errors, histogram with kernel density curves for errors.
2. Homoscedasticity Assumption: If you plot errors with target and instead of a random cloud you see a funnel shape or some other pattern , that'd be an indication of hetroscedastic errors. One popular remedy for the same is to take log of response and use that instead. Taking log brings down scale of errors and of course scale of difference between them as well.
3. Independence Assumption: If error seem to follow some pattern with any of the predictor that indicates at non-linear relation between and y and that predictor. We should try appropriate variable transformation instead of using that variable as it is.

Multiple Linear Regression

Multiple linear regression is just an extension of simple linear regression. Although few additional issues come up due to extra variables but basic frame work remains same. Instead of one predictor variable x, you have multiple predictor variables $x_1, x_2, x_3.....x_p$. The target y can still be written as linear combination of these predictors:

$$y_i = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \beta_3 * x_3..... + \beta_p * x_p + e_i$$

Again for finding best values of parameters, we'll minimize error sum of squares $\sum_{i=1}^{n} e_i^2$. Instead of 2 linear equations, now we'll have (p+1) linear equations to solve. Other than that everything else remains pretty much same.

Multi-collinearity

multi-collinearity means, one or more of the predictor variables being linearly dependent on few other predictor variables. There is a very important consequence to this in MLR. Recall the test statistic for hypothesis which we built to figure whether parameter estimate for β_i is significantly different from zero. If that test statistic is close to zero, p-value becomes high and you discard the corresponding predictor variable.

This reliance on p-value to discard not-so-good variable is alright until multiple predictors came into picture and with them , came multicollinearity. Discussion, which is about to follow is a little technical, if you find it overwhelming, don't worry, in the end we'll walk you through a case study done in SAS which you'll be able to understand even if this discussion doesn't make sense to you.

Lets say you have a target y which you are trying model as linear combination of $x_1, x_2, x_3.....x_p$. Forget about y for a minute. Consider that we choose one of the predictor as target and rest as predictor for it. Let x_j is my target and $x_1, x_2, x_3...x_{j-1}, x_{j+1}.....x_p$ are predictors for it. I go on and build a linear regression model and find its coefficient of determination to be R_j^2. if R_j^2 is high that means x_j can be written as linear combination of $x_1, x_2, x_3...x_{j-1}, x_{j+1}.....x_p$. This also implies that x_j contains redundant information. Another repercussion of this comes from the test statistic of hypothesis $\beta_j = 0$.

Remember the test statistic $\frac{\beta_j}{S_{\beta_j}}$. The denominator is proportional to $\frac{1}{(1-R_j^2)}$. As R_j^2 goes close to 1, S_{β_j} goes close to infinity, which means test statistic $\frac{\beta_j}{S_{\beta_j}}$ goes close to zero. Which in turn makes p-values artificially high. And you might end up discarding the variable. Remember that, p-values here are going up, **not** because variable is not correlated enough to y, but because of multicollinearity present in the data.

Which implies if multicollinearity is present in the data, you can not rely on the p-values. We can measure multicollinarity effect caused by presence of each variable by looking at the corresponding $\frac{1}{(1-R_j^2)}$ values. This is also called Variance Inflation Factor because it is the factor by which variance of parameters estimate S_{β_j} goes up.

$$VIF = \frac{1}{(1 - R_j^2)}$$

We will first need to remove variables from our consideration which have high VIF values and then proceed to build our regression model.

Using Categorical Variables

The process that you have seen requires all the variables considered to be strictly numbers. Which is not really the case with real life data. We get mix up of both numeric and string characteristics in our data. Then how do we use categorical variables in our model?

We need to figure out some way to convert them to numbers. Blindly giving them 1,2,3,4.... doesn't makes sense, it also makes your model simply bad. Because the mathematical framework is going to treat these numbers as usual. So 6 will be treated as 2 times 3, whereas say the variable in question was city names, and you assigned 3 to Agra and 6 to Kochi, it might not makes sense to say the **name** "Kochi" is 2 times "Agra".

We can comment that if I increase my numeric predictor say x_i by amount Δt then my target will change by some multiple of this say $\beta_i * \Delta t$. Now categorical variable don't "change" like that. At one time they will take some fixed category as a value at another some other category as a value. And measuring "change" in that context doesn't really make sense.

Now lets think how categorical variables really affect your target. Lets say your target is risk score for heart deceases and you have a model w.r.t. age As this:

$Risk = \beta_0 + \beta_1 * Age$

You have another categorical variable "history" which takes value "both" when both of your parents have history of having heart decease. "one" when one of the parents has history of heart decease, "none" otherwise. We can hypothetically say that, if history="both", Risk goes up by 0.10 or 10%, if history="one" then Risk goes up by 0.05 and no effect if history="none". So we can have three different models to incorporate this categorical variable.

when history="both" then $Risk = \beta_0 + \beta_1 * Age + 0.1$. if history="one" then $Risk = \beta_0 + \beta_1 * Age + 0.05$ and when history="none" then simply $Risk = \beta_0 + \beta_1 * Age$. We can combine this if we consider two **dummy** variables, history_both=1 when history="both" and 0 otherwise. history_one=1 when history="one" and 0 otherwise.

$Risk = \beta_0 + \beta_1 * Age + 0.1 * history\_both + 0.05 * history\_one$

The lesson here is that we can convert a categorical variable which takes **n** distinct values to a set of **n-1** dummy variables as mentioned above and use them just like numeric variables in the regression process. You must be wondering why **n-1** . Create on your own a hypothetical categorical variable which takes values "a" or "b" or "c" randomly. Create 10 observations for that. Now create three dummy variables cat_a, cat_b and cat_c. Observe that you can write:

$cat\_c = 1 - cat\_a - cat\_b$

you can randomly switch places of cat_a, cat_b and cat_c, and this equation will still hold. Basically if you know values of **n-1** dummy variables , you can perfectly know what would be the value of n^{th} dummy variable. To avoid this situation of perfect multicollineraity between predictors we need to make only **n-1** dummy variables for a categorical variable which takes **n** distinct values or has **n** distinct categories.

Then the question comes which one should i drop OR *not* create a dummy variable for.Well, it doesn't really matter. However standard is to drop the category which is least frequent.

Model Validation

We have pretty much learned every thing which we needed to build a linear regression model. However it might not be enough still to achieve our goal , which was to come with an equation which enables us to forecast results for **yet unknown** or **future** values of predictors. Why? , because the parameter estimates that you got are for the data that you already have. How do you check if this model will perform well on the unseen data as well?

The answer is rather simple, randomly break your data in two samples and use one to build your model and check performance on the second one. We'll formally call them train and test datasets.

Although you'll come across few machine learning algorithms [Decision Trees], where they have cross validation inbuilt functions which utilize train data for tuning the model.

k fold cross validation : Underlying process is that , your train data is broken into k random parts. At a time a model is built on k-1 parts together and its performance [error] is checked on the left out part. This process is done k times , leaving one part [out of k parts] for measuring error. Final error is calculated as average of k iterations. This average error helps in tuning parameters of machine learning models. We'll implementation of cross validation when we reach to that part of the course.

For now we'll rely on breaking our data into two parts train and test.

Let me re-iterate why we need to do this. Reason is rather practical requirement than statistical. Ultimate goal of all this model building is to get an equation which can be used on unseen data to predict outcome of the business process.

Lets Summarise The Model Buidling Process

- Remove or impute missing values in the data
- Create dummy variables for the categorical variables
- Break your data in to three parts : train,validation and test. start building model on train
  - In the first run of your model building process check VIF for all variables.
  - Drop variable with high VIF (>5). Drop them one by one only **not all at once**. drop the one with highest VIF, run your process again and then pick again the variable with highest VIF and drop, keep on doing this until VIF is (<5) for all variables.
  - Once VIF is under control for all the remaining variables , start dropping variable one by one based on p-values. If p-value is **greater** than 0.05, drop that variable
  - You have your train model , once you have all remaining variables with p-values less than 0.05.
- Test this model's performance by calculating RMSE (Root mean square error) on test dataset.
- You can use RMSE calculated on test data to compare multiple models.

Case Study 1

We'll also learn syntax for R [which is surprisingly simple] while we are carrying out model building process.

Loan Smart is a lending advisory firm. Based on their client's characteristic and needed loan amount they advise them on which Financial Institution to apply for loan at. So far their recommendations have been based hunches business experience. Now they are trying to leverage power of data that they have collected so far.

They want to check whether given their client's characteristics , they can predict how much interest rates they will be offered by various financial institution. They want to run with proof of concept for this idea. They have given us data collected for one such financial institution ABC Capitals Ltd.

What we need to do is to figure out whether using that data we can predict interest rate offered to client. Lets import and look at the data

```
# Remember you need to set working directory to the folder which contains this data
ld=read.csv("loans data.csv",stringsAsFactors = FALSE)
library(dplyr)
glimpse(ld)
```

```
## Observations: 2,500
## Variables: 15
## $ ID                            (int) 81174, 99592, 80059, 15825, 331...
## $ Amount.Requested              (chr) "20000", "19200", "35000", "100...
## $ Amount.Funded.By.Investors    (chr) "20000", "19200", "35000", "997...
## $ Interest.Rate                 (chr) "8.90%", "12.12%", "21.98%", "9...
## $ Loan.Length                   (chr) "36 months", "36 months", "60 m...
## $ Loan.Purpose                  (chr) "debt_consolidation", "debt_con...
## $ Debt.To.Income.Ratio          (chr) "14.90%", "28.36%", "23.81%", "...
## $ State                         (chr) "SC", "TX", "CA", "KS", "NJ", "...
## $ Home.Ownership                (chr) "MORTGAGE", "MORTGAGE", "MORTGA...
## $ Monthly.Income                (dbl) 6541.67, 4583.33, 11500.00, 383...
## $ FICO.Range                    (chr) "735-739", "715-719", "690-694"...
## $ Open.CREDIT.Lines             (chr) "14", "12", "14", "10", "11", "...
## $ Revolving.CREDIT.Balance      (chr) "14272", "11140", "21977", "934...
## $ Inquiries.in.the.Last.6.Months (int) 2, 1, 1, 0, 0, 2, 0, 0, 1, 0, 0...
## $ Employment.Length             (chr) "< 1 year", "2 years", "2 years...
```

Variable names are self explanatory as to what they represent.In case you have any doubts , feel free to post on Q&A forum. Also Let me remind you at this point, a big chunk of predictive modeling is preparing your

data, getting it ready for the modeling process.

When you look at the data set , you'll find out Interest.Rate , Debt.To.Income.Ratio have been imported as characters due to "%" sign . Open.Credit.Lines and some othe supposedly numeric variables have been converted to character becuase of some character value present possibly. Lets convert them to numbers first.

```
ld=ld %>%
  mutate(Interest.Rate=as.numeric(gsub("%","",Interest.Rate)) ,
         Debt.To.Income.Ratio=as.numeric(gsub("%","",Debt.To.Income.Ratio)) ,
         Open.CREDIT.Lines=as.numeric(Open.CREDIT.Lines) ,
         Amount.Requested=as.numeric(Amount.Requested) ,
         Amount.Funded.By.Investors=as.numeric(Amount.Funded.By.Investors),
         Revolving.CREDIT.Balance=as.numeric(Revolving.CREDIT.Balance)
         )
glimpse(ld)
```

```
## Observations: 2,500
## Variables: 15
## $ ID                              (int) 81174, 99592, 80059, 15825, 331...
## $ Amount.Requested                (dbl) 20000, 19200, 35000, 10000, 120...
## $ Amount.Funded.By.Investors      (dbl) 20000.00, 19200.00, 35000.00, 9...
## $ Interest.Rate                   (dbl) 8.90, 12.12, 21.98, 9.99, 11.71...
## $ Loan.Length                     (chr) "36 months", "36 months", "60 m...
## $ Loan.Purpose                    (chr) "debt_consolidation", "debt_con...
## $ Debt.To.Income.Ratio            (dbl) 14.90, 28.36, 23.81, 14.30, 18....
## $ State                           (chr) "SC", "TX", "CA", "KS", "NJ", "...
## $ Home.Ownership                  (chr) "MORTGAGE", "MORTGAGE", "MORTGA...
## $ Monthly.Income                  (dbl) 6541.67, 4583.33, 11500.00, 383...
## $ FICO.Range                      (chr) "735-739", "715-719", "690-694"...
## $ Open.CREDIT.Lines               (dbl) 14, 12, 14, 10, 11, 17, 10, 12,...
## $ Revolving.CREDIT.Balance        (dbl) 14272, 11140, 21977, 9346, 1446...
## $ Inquiries.in.the.Last.6.Months (int) 2, 1, 1, 0, 0, 2, 0, 0, 1, 0, 0...
## $ Employment.Length               (chr) "< 1 year", "2 years", "2 years...
```

The variable Amount_Funded_By_Investors happens to have high correlation with Interest.Rate [our target], but we should still drop this variables because it contains information which will not be available at the time when we need to use this model.

When someone comes with their loan application to Loan Smart , it does not contain information about amount funded by investors. This is a lesson for not falling in love with your data. Always think from a perspective of "At what point of the business process you are going to use this model" and then discard variables containing information which will not be available at that decision point.

```
ld = ld %>%
  select(-Amount.Funded.By.Investors)
```

lets see what we are going to do about categorical variables. Although fico_range is recorded as categorical variable in terms of ranges , we can convert it to a numeric variable by assigning value as average of the range. lets do that.

```
ld= ld %>%
  mutate(f1=as.numeric(substr(FICO.Range,1,3)),
         f2=as.numeric(substr(FICO.Range,5,7)),
         fico=0.5*(f1+f2)
         ) %>%
  select(-FICO.Range,-f1,-f2)
glimpse(ld)
```

```
## Observations: 2,500
## Variables: 14
## $ ID                              (int) 81174, 99592, 80059, 15825, 331...
## $ Amount.Requested                (dbl) 20000, 19200, 35000, 10000, 120...
## $ Interest.Rate                   (dbl) 8.90, 12.12, 21.98, 9.99, 11.71...
## $ Loan.Length                     (chr) "36 months", "36 months", "60 m...
## $ Loan.Purpose                    (chr) "debt_consolidation", "debt_con...
## $ Debt.To.Income.Ratio            (dbl) 14.90, 28.36, 23.81, 14.30, 18....
## $ State                           (chr) "SC", "TX", "CA", "KS", "NJ", "...
## $ Home.Ownership                  (chr) "MORTGAGE", "MORTGAGE", "MORTGA...
## $ Monthly.Income                  (dbl) 6541.67, 4583.33, 11500.00, 383...
## $ Open.CREDIT.Lines               (dbl) 14, 12, 14, 10, 11, 17, 10, 12,...
## $ Revolving.CREDIT.Balance        (dbl) 14272, 11140, 21977, 9346, 1446...
## $ Inquiries.in.the.Last.6.Months (int) 2, 1, 1, 0, 0, 2, 0, 0, 1, 0, 0...
## $ Employment.Length               (chr) "< 1 year", "2 years", "2 years...
## $ fico                            (dbl) 737, 717, 692, 697, 697, 672, 7...
```

In a similar fashion we can convert employment length to numbers as well.

```
ld=ld %>%
  mutate(el=ifelse(substr(Employment.Length,1,2)=="10",10,Employment.Length),
         el=ifelse(substr(Employment.Length,1,1)=="<",0,el),
         el=gsub("years","",el),
         el=gsub("year","",el),
         el=as.numeric(el)
         ) %>%
  select(-Employment.Length) %>%
  na.omit()

# You must have noticed that we have also removed obs with missing values.
# You could have imputed them if you want
```

Lets take a pause at this point and think over the decision of how we processed employment length here. Converting this to numbers using the logic above was a subjective decision. We can't really say that this is what we do in every case. We might have as well went on treating it as categorical variable and created dummy variables for it. We might have used some other logic to convert it to numeric by assigning some other numeric values. Each of these decisions would have resulted in a different model. Lets say we build models m1, m2, m3 based on all these decision separately. You can select one of these models to go forward with , based on performance on the test data. In this text however , we'll be focusing on just one iteration and in turn developing just one model. You can try out other models with different features/variables [arriving from different subjective decisions while preparing data] and see whether they perform better than the model developed in this iteration here. [Its not a challenge, your model can very well outperform the one developed here !]

Now lets resume and get to create dummy variables for our remaining categorical variables. We should ignore categories with very low count as well. So instead of **n-1** dummy variables , we at times have made far less dummy variables. You can choose not to ignore and make more dummy variables. That way you'll have one of your own iterations of this process and you can check whether your iteration performs better or not.

```
table(ld$Home.Ownership)
```

```
##
## MORTGAGE    OTHER      OWN     RENT
##     1098        5      186     1107
```

We should categories with small counts and consider only 3 categories , however we'll create 3 dummy variable for Home Ownership to see effect of doing so.

```
ld=ld %>%
  mutate(HW_RENT=as.numeric(Home.Ownership=="RENT"),
         HW_MORT=as.numeric(Home.Ownership=="MORTGAGE"),
         HW_OWN=as.numeric(Home.Ownership=="OWN")) %>%
  select(-Home.Ownership)
```

Note: If you do choose to go ahead and not ignore the very less frequent category and make more dummy variables accrodingly , you'll notice that these dummy variables will end up having high VIF. Why?

Consider a hypothetical example where a categorical variable takes 5 distinct values a ,b ,c, d, e. Among these e occurs very few times. We do not make 5 dummy variable because of this relation ship:

$cat\_a + cat\_b + cat\_c + cat\_d + cat\_e = 1$

Which implies that if we make 5 dummy variables , any one taken at a time will be perfect linear combination of the rest which is undesirable as discussed previously.

So we will go ahead and take e as base category , this being least frequent and make 4 dummy variables. But e was a rare category. Meaning dummy variable for e , if made would have been equal to 1 rarely, which means that rest 4 dummies will still add up to exactly one for most part of the data , hence the high VIF values. We'll that in our results also.

Next we look at variable Loan.Purpose

```
table(ld$Loan.Purpose)
```

```
##
##                car        credit_card debt_consolidation
##                 49                426               1261
##        educational   home_improvement              house
##                 14                144                 20
##     major_purchase            medical             moving
##                 98                 28                 28
##              other   renewable_energy     small_business
##                184                  4                 82
##           vacation            wedding
##                 20                 38
```

It has too many categories. There is no direct harm in considering to create dummy variables for n-1 fo them. That can be one iteration to try. Here we are going to make just three dummy variables for categories credit_card, debt_consolidation and other.

```
ld=ld %>%
  mutate(LP_cc=as.numeric(Loan.Purpose=="credit_card"),
         LP_dc=as.numeric(Loan.Purpose=="debt_consolidation"),
         LP_other=as.numeric(Loan.Purpose=="other")
         ) %>%
  select(-Loan.Purpose)
```

Next we look at Loan.Length.

```
table(ld$Loan.Length)
```

```
##
##         . 36 months 60 months
##         1      1861       534
```

```
ld=ld %>%
  mutate(LL_36=as.numeric(Loan.Length=="36 months")) %>%
```

```
  select(-Loan.Length)

table(ld$State)
```

```
##
##  AK  AL  AR  AZ  CA  CO  CT  DC  DE  FL  GA  HI  IA  IL  IN  KS  KY  LA
##  11  36  13  45 414  56  47  11   8 161  96  11   1  99   3  20  23  20
##  MA  MD  MI  MN  MO  MS  MT  NC  NH  NJ  NM  NV  NY  OH  OK  OR  PA  RI
##  70  66  43  37  33   1   7  60  15  89  11  31 243  68  21  28  93  15
##  SC  SD  TX  UT  VA  VT  WA  WI  WV  WY
##  24   4 170  16  75   5  54  26  12   4
```

Variable State takes too many values and if you look at frequencies , there is no dominant category as such. If you want to make dummy variable for the same , please do so. In this iteration we are going to ignore variable "State" and drop that.

```
ld= ld %>%
  select(-State)
```

Now we are done with preparing data , lets split the data.

```
set.seed(2)
s=sample(1:nrow(ld),0.7*nrow(ld))
ld_train=ld[s,]
ld_test=ld[-s,]
```

Now we'll start with our modeling process. We'll be using function `lm` for building our linear regression model. First argument to this function is the modeling equation which we are trying to model. second argument is the dataset.

Lets say our data set has response y and predcitor variables as a ,b and c. then we'll write the modeling equation as this.

$$y \sim a + b + c$$

If the a, b, and c are the only predictor variables in the the data and you want to use all of them in your modeling equation, you can simply write:

$$y \sim .$$

If you want to use all predictor present in the data except , say b, then you can write :

$$y \sim . - b$$

Lets begin. Our response here is Interest.Rate and rest of the variables now in `ld_train` are our predictor variable. Before you begin make sure that none of the variables in the dataset which you are going to use for modeling are of character type.

```
glimpse(ld_train)
```

```
## Observations: 1,677
## Variables: 17
## $ ID                          (int) 55717, 363, 97364, 28871, 75525...
## $ Amount.Requested            (dbl) 7500, 10000, 20800, 15250, 1750...
## $ Interest.Rate               (dbl) 9.76, 13.30, 6.62, 16.49, 14.09...
## $ Debt.To.Income.Ratio        (dbl) 14.96, 6.31, 15.64, 20.57, 34.8...
```

```
## $ Monthly.Income                 (dbl) 7083.33, 10000.00, 10833.33, 29...
## $ Open.CREDIT.Lines              (dbl) 11, 11, 11, 10, 6, 9, 9, 8, 7, ...
## $ Revolving.CREDIT.Balance       (dbl) 16449, 28653, 15192, 17931, 184...
## $ Inquiries.in.the.Last.6.Months (int) 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0...
## $ fico                           (dbl) 782, 662, 712, 692, 692, 692, 6...
## $ el                             (dbl) 10, 10, 10, 0, 1, 7, 1, 10, 1, ...
## $ HW_RENT                        (dbl) 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0...
## $ HW_MORT                        (dbl) 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1...
## $ HW_OWN                         (dbl) 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ LP_cc                          (dbl) 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0...
## $ LP_dc                          (dbl) 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0...
## $ LP_other                       (dbl) 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1...
## $ LL_36                          (dbl) 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1...
```

You can see that all the variables in data are numeric , lets start now. It doesnt make sense to include variable `ID` in the modelling process. Its just a id number for the transactions which should be ignored.

```
fit=lm(Interest.Rate~. -ID,data=ld_train)
```

Before we go start looking at model summary and p-value for the variables , we need to drop variables from the model which have high VIF. Remember that VIF is a mutual thing, so if you drop one variable, it might happen that VIF value for several other variables will go down. For this reason, always drop variables for high VIF , one by one, never in a chunk. For looking at VIF values for the variables , we'll use function `vif` from package `car`.

```
library(car)
vif(fit)
```

```
##                Amount.Requested           Debt.To.Income.Ratio
##                        1.601573                       1.364146
##                  Monthly.Income              Open.CREDIT.Lines
##                        1.402768                       1.344688
##       Revolving.CREDIT.Balance Inquiries.in.the.Last.6.Months
##                        1.318871                       1.045984
##                            fico                             el
##                        1.153243                       1.090594
##                         HW_RENT                        HW_MORT
##                      105.600550                     106.113545
##                         HW_OWN                           LP_cc
##                       29.936906                       1.644014
##                           LP_dc                        LP_other
##                        1.833158                       1.291791
##                           LL_36
##                        1.257889
```

We can see that all variables have VIF under 5 , except 3 variables HW_MORT , HW_RENT and HW_OWN, we will drop HW_MORT and see if this improves things.

```
fit=lm(Interest.Rate~. -ID - HW_MORT,data=ld_train)
vif(fit)
```

```
##                Amount.Requested           Debt.To.Income.Ratio
##                        1.600472                       1.362425
##                  Monthly.Income              Open.CREDIT.Lines
##                        1.402430                       1.341196
##       Revolving.CREDIT.Balance Inquiries.in.the.Last.6.Months
##                        1.316826                       1.045602
```

```
##                            fico                              el
##                        1.151463                        1.089516
##                         HW_RENT                           HW_OWN
##                        1.251635                        1.104588
##                           LP_cc                           LP_dc
##                        1.642075                        1.832890
##                        LP_other                           LL_36
##                        1.291791                        1.257672
```

All VIF values are under control.Now we look at the model summary , specifically p-values asscociated with the variables. We'll drop variables with high p-values [>0.05] one by one

```
summary(fit)
```

```
##
## Call:
## lm(formula = Interest.Rate ~ . - ID - HW_MORT, data = ld_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.3743 -1.3822 -0.1933  1.1733  9.7761
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    7.612e+01  1.140e+00  66.795  < 2e-16 ***
## Amount.Requested               1.601e-04  8.057e-06  19.867  < 2e-16 ***
## Debt.To.Income.Ratio           1.683e-03  7.770e-03   0.217  0.82858
## Monthly.Income                -1.839e-05  1.392e-05  -1.321  0.18661
## Open.CREDIT.Lines             -3.744e-02  1.285e-02  -2.914  0.00361 **
## Revolving.CREDIT.Balance      -4.814e-06  3.093e-06  -1.557  0.11977
## Inquiries.in.the.Last.6.Months 3.216e-01  4.201e-02   7.655 3.27e-14 ***
## fico                          -8.799e-02  1.514e-03 -58.136  < 2e-16 ***
## el                             1.239e-02  1.460e-02   0.849  0.39601
## HW_RENT                        2.529e-01  1.124e-01   2.250  0.02461 *
## HW_OWN                         4.803e-01  2.009e-01   2.390  0.01694 *
## LP_cc                         -5.136e-01  1.697e-01  -3.026  0.00252 **
## LP_dc                         -3.590e-01  1.357e-01  -2.646  0.00823 **
## LP_other                       1.406e-01  2.090e-01   0.673  0.50111
## LL_36                         -3.161e+00  1.343e-01 -23.543  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.048 on 1662 degrees of freedom
## Multiple R-squared:  0.7658, Adjusted R-squared:  0.7638
## F-statistic: 388.1 on 14 and 1662 DF,  p-value: < 2.2e-16
```

We'll start with dropping variable `Debt.To.Income.Ratio` which has highest p-value [0.9456]

```
fit=lm(Interest.Rate~. -ID - HW_MORT-Debt.To.Income.Ratio,data=ld_train)
summary(fit)
```

```
##
## Call:
## lm(formula = Interest.Rate ~ . - ID - HW_MORT - Debt.To.Income.Ratio,
##     data = ld_train)
##
```

```
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.379 -1.380 -0.192  1.175  9.784
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    7.617e+01  1.117e+00  68.214  < 2e-16 ***
## Amount.Requested               1.602e-04  8.038e-06  19.926  < 2e-16 ***
## Monthly.Income                -1.921e-05  1.339e-05  -1.434  0.15167
## Open.CREDIT.Lines             -3.647e-02  1.204e-02  -3.028  0.00250 **
## Revolving.CREDIT.Balance      -4.713e-06  3.057e-06  -1.542  0.12328
## Inquiries.in.the.Last.6.Months  3.213e-01  4.197e-02   7.654 3.29e-14 ***
## fico                          -8.804e-02  1.497e-03 -58.802  < 2e-16 ***
## el                             1.248e-02  1.459e-02   0.855  0.39255
## HW_RENT                        2.534e-01  1.124e-01   2.255  0.02425 *
## HW_OWN                         4.813e-01  2.008e-01   2.397  0.01665 *
## LP_cc                         -5.109e-01  1.692e-01  -3.019  0.00257 **
## LP_dc                         -3.564e-01  1.351e-01  -2.638  0.00842 **
## LP_other                       1.419e-01  2.088e-01   0.680  0.49690
## LL_36                         -3.161e+00  1.342e-01 -23.549  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.048 on 1663 degrees of freedom
## Multiple R-squared:  0.7658, Adjusted R-squared:  0.7639
## F-statistic: 418.2 on 13 and 1663 DF,  p-value: < 2.2e-16
```

el goes next. We'll subsequently keep on dropping variables until all remaining variables have p-values less than 0.05

```
fit=lm(Interest.Rate~. -ID - HW_MORT-Debt.To.Income.Ratio-el,data=ld_train)
summary(fit)
```

```
##
## Call:
## lm(formula = Interest.Rate ~ . - ID - HW_MORT - Debt.To.Income.Ratio -
##     el, data = ld_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.4347 -1.3595 -0.1682  1.1632  9.7796
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    7.625e+01  1.113e+00  68.520  < 2e-16 ***
## Amount.Requested               1.607e-04  8.013e-06  20.057  < 2e-16 ***
## Monthly.Income                -1.900e-05  1.339e-05  -1.419  0.15608
## Open.CREDIT.Lines             -3.659e-02  1.204e-02  -3.038  0.00242 **
## Revolving.CREDIT.Balance      -4.635e-06  3.055e-06  -1.517  0.12939
## Inquiries.in.the.Last.6.Months  3.211e-01  4.197e-02   7.650 3.37e-14 ***
## fico                          -8.806e-02  1.497e-03 -58.823  < 2e-16 ***
## HW_RENT                        2.331e-01  1.098e-01   2.123  0.03392 *
## HW_OWN                         4.659e-01  2.000e-01   2.330  0.01993 *
## LP_cc                         -5.018e-01  1.689e-01  -2.971  0.00301 **
## LP_dc                         -3.509e-01  1.349e-01  -2.600  0.00939 **
```

```
## LP_other                        1.428e-01  2.088e-01   0.684  0.49404
## LL_36                          -3.163e+00  1.342e-01 -23.568  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.048 on 1664 degrees of freedom
## Multiple R-squared:  0.7657, Adjusted R-squared:  0.764
## F-statistic: 453.1 on 12 and 1664 DF,  p-value: < 2.2e-16
```

```
fit=lm(Interest.Rate~. -ID - HW_MORT-Debt.To.Income.Ratio  -el- LP_other,data=ld_train)
summary(fit)
```

```
##
## Call:
## lm(formula = Interest.Rate ~ . - ID - HW_MORT - Debt.To.Income.Ratio -
##     el - LP_other, data = ld_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.4373 -1.3673 -0.1735  1.1635  9.8879
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   7.636e+01  1.101e+00  69.382  < 2e-16 ***
## Amount.Requested              1.606e-04  8.009e-06  20.049  < 2e-16 ***
## Monthly.Income               -1.923e-05  1.338e-05  -1.437 0.151030
## Open.CREDIT.Lines            -3.672e-02  1.204e-02  -3.050 0.002322 **
## Revolving.CREDIT.Balance     -4.588e-06  3.054e-06  -1.502 0.133188
## Inquiries.in.the.Last.6.Months  3.217e-01  4.195e-02   7.668 2.96e-14 ***
## fico                         -8.816e-02  1.489e-03 -59.187  < 2e-16 ***
## HW_RENT                       2.375e-01  1.096e-01   2.166 0.030455 *
## HW_OWN                        4.612e-01  1.998e-01   2.308 0.021128 *
## LP_cc                        -5.418e-01  1.584e-01  -3.420 0.000641 ***
## LP_dc                        -3.903e-01  1.220e-01  -3.198 0.001409 **
## LL_36                        -3.162e+00  1.342e-01 -23.567  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.047 on 1665 degrees of freedom
## Multiple R-squared:  0.7656, Adjusted R-squared:  0.7641
## F-statistic: 494.4 on 11 and 1665 DF,  p-value: < 2.2e-16
```

```
fit=lm(Interest.Rate~. -ID - HW_MORT-Debt.To.Income.Ratio
       - LP_other -el -Revolving.CREDIT.Balance ,data=ld_train)
summary(fit)
```

```
##
## Call:
## lm(formula = Interest.Rate ~ . - ID - HW_MORT - Debt.To.Income.Ratio -
##     LP_other - el - Revolving.CREDIT.Balance, data = ld_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.4386 -1.3677 -0.2023  1.1578  9.9224
##
```

```
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    7.633e+01  1.101e+00  69.340  < 2e-16 ***
## Amount.Requested               1.589e-04  7.935e-06  20.025  < 2e-16 ***
## Monthly.Income                -2.429e-05  1.296e-05  -1.875 0.061030 .
## Open.CREDIT.Lines             -4.080e-02  1.173e-02  -3.478 0.000519 ***
## Inquiries.in.the.Last.6.Months 3.217e-01  4.197e-02   7.665 3.02e-14 ***
## fico                          -8.808e-02  1.489e-03 -59.147  < 2e-16 ***
## HW_RENT                        2.529e-01  1.092e-01   2.316 0.020662 *
## HW_OWN                         4.607e-01  1.999e-01   2.305 0.021306 *
## LP_cc                         -5.641e-01  1.578e-01  -3.575 0.000360 ***
## LP_dc                         -3.939e-01  1.221e-01  -3.227 0.001273 **
## LL_36                         -3.166e+00  1.342e-01 -23.595  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.048 on 1666 degrees of freedom
## Multiple R-squared:  0.7653, Adjusted R-squared:  0.7639
## F-statistic: 543.2 on 10 and 1666 DF,  p-value: < 2.2e-16
```

```
fit=lm(Interest.Rate~. -ID - HW_MORT-Debt.To.Income.Ratio
       - LP_other -el -Revolving.CREDIT.Balance - HW_OWN,data=ld_train)
summary(fit)
```

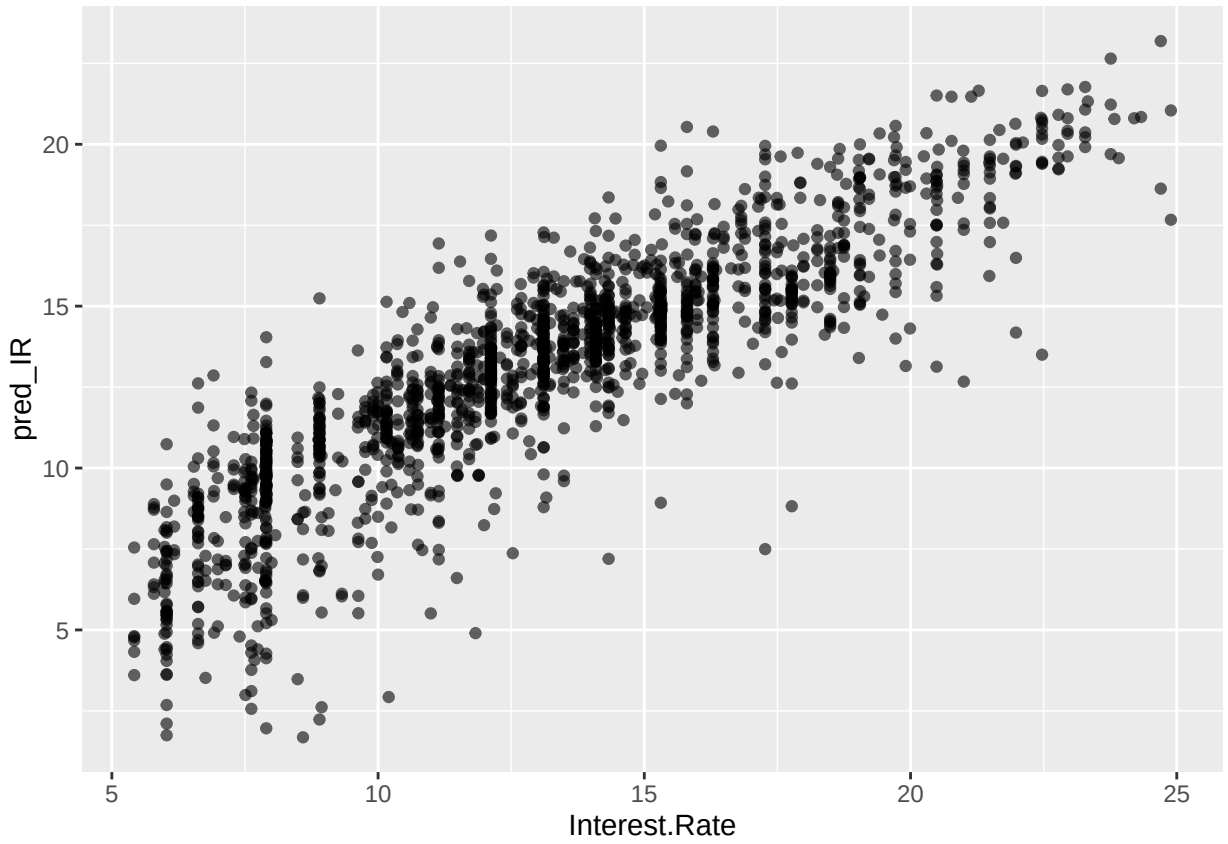
```
##
## Call:
## lm(formula = Interest.Rate ~ . - ID - HW_MORT - Debt.To.Income.Ratio -
##     LP_other - el - Revolving.CREDIT.Balance - HW_OWN, data = ld_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.4351 -1.3682 -0.1929  1.1799  9.8418
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    7.644e+01  1.101e+00  69.423  < 2e-16 ***
## Amount.Requested               1.588e-04  7.945e-06  19.982  < 2e-16 ***
## Monthly.Income                -2.672e-05  1.293e-05  -2.066 0.038959 *
## Open.CREDIT.Lines             -4.129e-02  1.175e-02  -3.515 0.000451 ***
## Inquiries.in.the.Last.6.Months 3.208e-01  4.202e-02   7.635 3.79e-14 ***
## fico                          -8.813e-02  1.491e-03 -59.108  < 2e-16 ***
## HW_RENT                        1.836e-01  1.051e-01   1.747 0.080784 .
## LP_cc                         -5.628e-01  1.580e-01  -3.562 0.000378 ***
## LP_dc                         -4.094e-01  1.220e-01  -3.355 0.000811 ***
## LL_36                         -3.147e+00  1.341e-01 -23.466  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.051 on 1667 degrees of freedom
## Multiple R-squared:  0.7645, Adjusted R-squared:  0.7633
## F-statistic: 601.4 on 9 and 1667 DF,  p-value: < 2.2e-16
```

```
fit=lm(Interest.Rate~. -ID - HW_MORT-Debt.To.Income.Ratio
       - LP_other -el -Revolving.CREDIT.Balance - HW_OWN- HW_RENT,data=ld_train)
summary(fit)
```

```
##
## Call:
## lm(formula = Interest.Rate ~ . - ID - HW_MORT - Debt.To.Income.Ratio -
##     LP_other - el - Revolving.CREDIT.Balance - HW_OWN - HW_RENT,
##     data = ld_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.3411 -1.3544 -0.1825  1.2225  9.7750
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   7.687e+01  1.075e+00  71.495  < 2e-16 ***
## Amount.Requested              1.582e-04  7.943e-06  19.914  < 2e-16 ***
## Monthly.Income               -2.930e-05  1.285e-05  -2.279 0.022767 *
## Open.CREDIT.Lines            -4.422e-02  1.163e-02  -3.801 0.000149 ***
## Inquiries.in.the.Last.6.Months  3.157e-01  4.194e-02   7.526 8.49e-14 ***
## fico                         -8.854e-02  1.474e-03 -60.081  < 2e-16 ***
## LP_cc                        -5.586e-01  1.581e-01  -3.534 0.000420 ***
## LP_dc                        -4.101e-01  1.221e-01  -3.358 0.000802 ***
## LL_36                        -3.139e+00  1.341e-01 -23.405  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.052 on 1668 degrees of freedom
## Multiple R-squared:  0.7641, Adjusted R-squared:  0.763
## F-statistic: 675.4 on 8 and 1668 DF,  p-value: < 2.2e-16
```

This is our final model with all variables being significant [p-value < 0.05]. How good is this model. Adj R-squared value for the model is 0.7713 which means that this models explains 77.13% variation in our response. You can quickly check this visually too by plotting predicted values and the original response.

```
ld_train %>%
  mutate(pred_IR=predict(fit,newdata=ld_train)) %>%
  ggplot(aes(x=Interest.Rate,y=pred_IR))+geom_point(alpha=0.6)
```



You can see that there is good overlap.

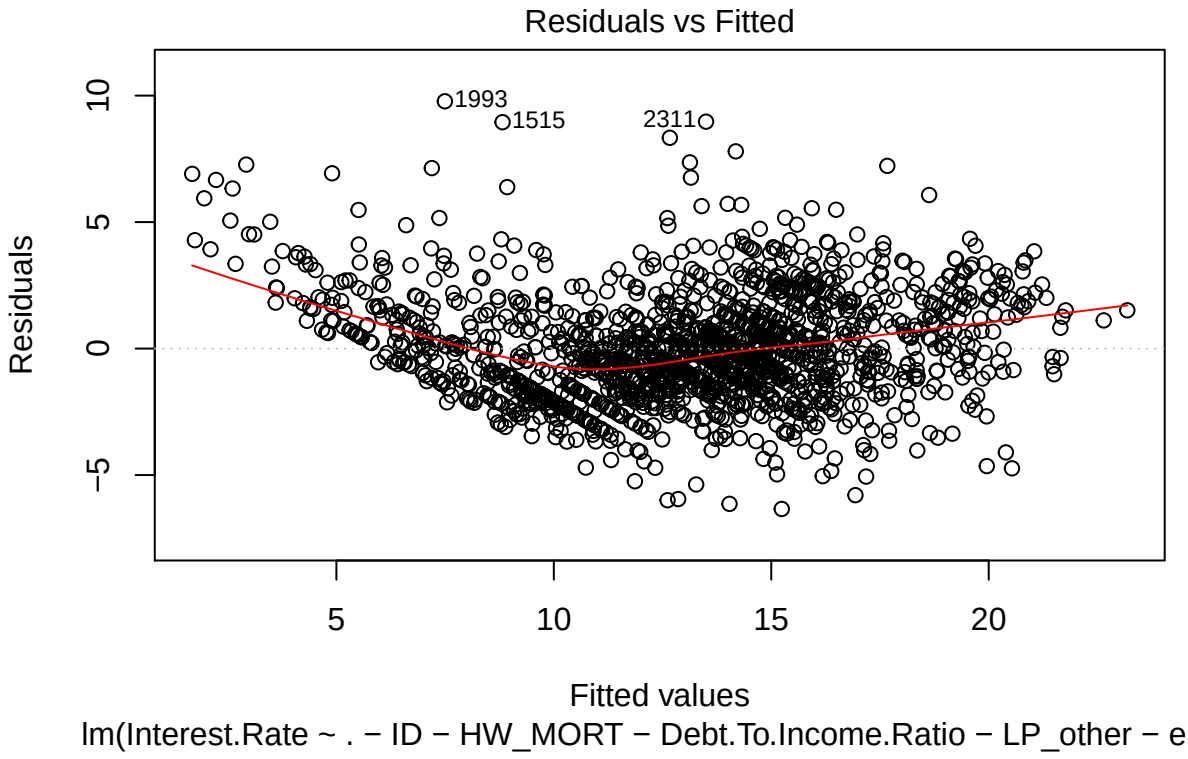
Our model equation from the model is :[taking down values from estimates from `summary(fit_train)`]. We are going to use collapse property of paste function. The result will need little modification but it saves us from noting down each value and writing prediction equation manually.

```
model_string=paste(fit$coefficients,names(fit$coefficients),sep="*",collapse = " + ")
strwrap(sub("*(Intercept)","",gsub("+ -","- ",model_string,fixed=TRUE)))
```

```
## [1] "76.8650907054335*() + 0.000158179982459964*Amount.Requested -"
## [2] "2.93000074294854e-05*Monthly.Income -"
## [3] "0.0442199732920645*Open.CREDIT.Lines +"
## [4] "0.315689634430113*Inquiries.in.the.Last.6.Months -"
## [5] "0.0885375016793099*fico - 0.558649920680204*LP_cc -"
## [6] "0.410066504084309*LP_dc - 3.1385200298849*LL_36"
```

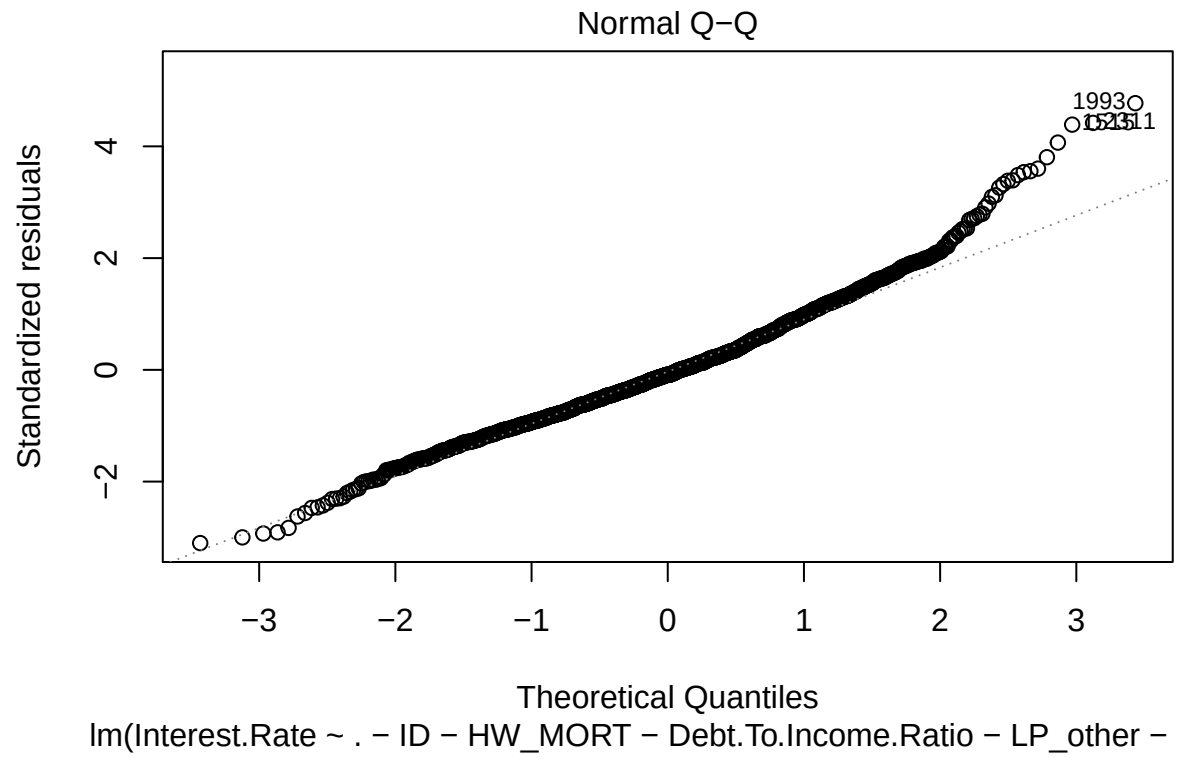
Next we look at different diagnostic plots to see if any of our linear regression assumptions are being violated and some other information regarding outliers.

```
plot(fit,which=1)
```



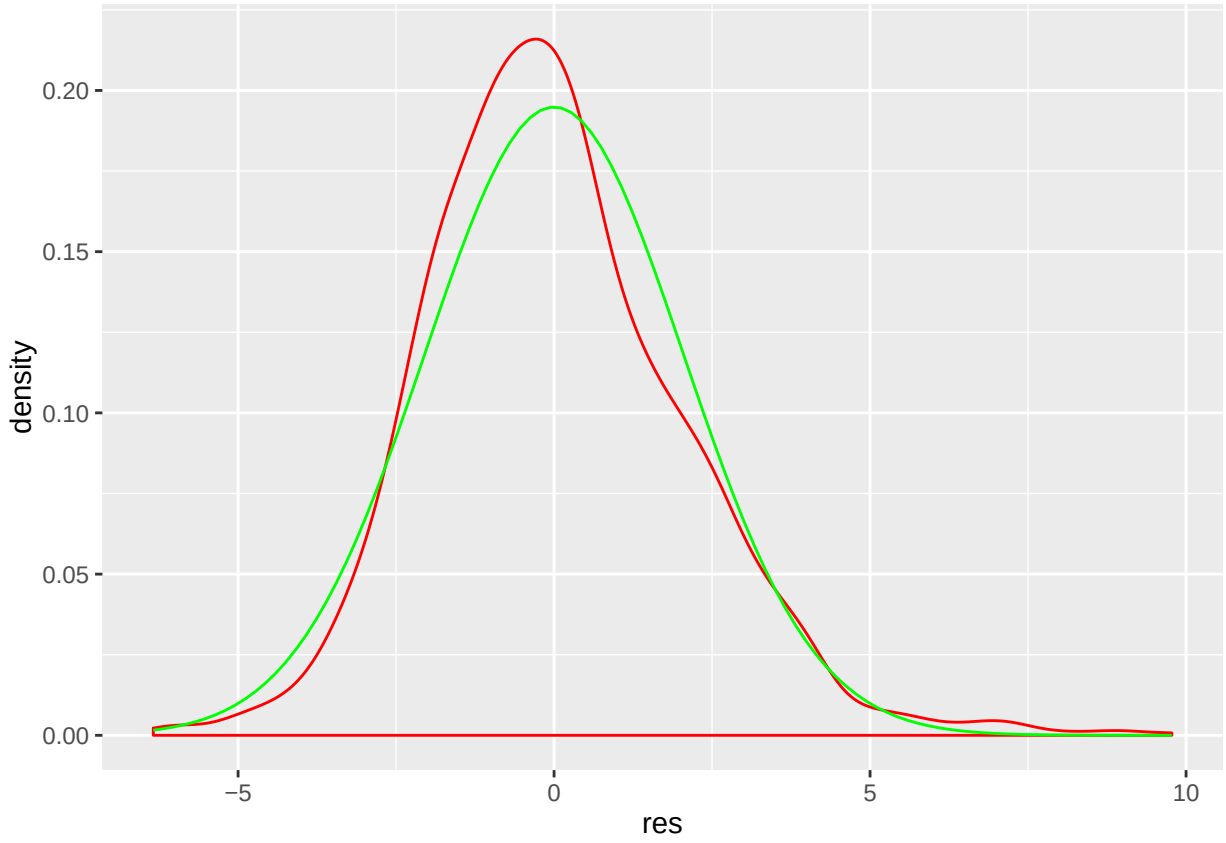
Ideally the red line the plot should be horizontal or close to horizontal . If it shows a significant pattern , it means we might have missed non-linear components in the model. If that is the case you might want to look at pair wise relation ship between response and each predictor and figure out if any of the predictors need to be transformed to make the relationship linear between response and that predictor.

```
plot(fit,which=2)
```



This tells you whether you residuals are normal or not. You can visualise see this by plotting density plots as well.

```
df=data.frame(res=fit$residual)
ggplot(df,aes(x=res))+geom_density(color="red")+
  stat_function(fun=dnorm ,args = list(mean=mean(df$res),sd=sd(df$res)),color="green")
```

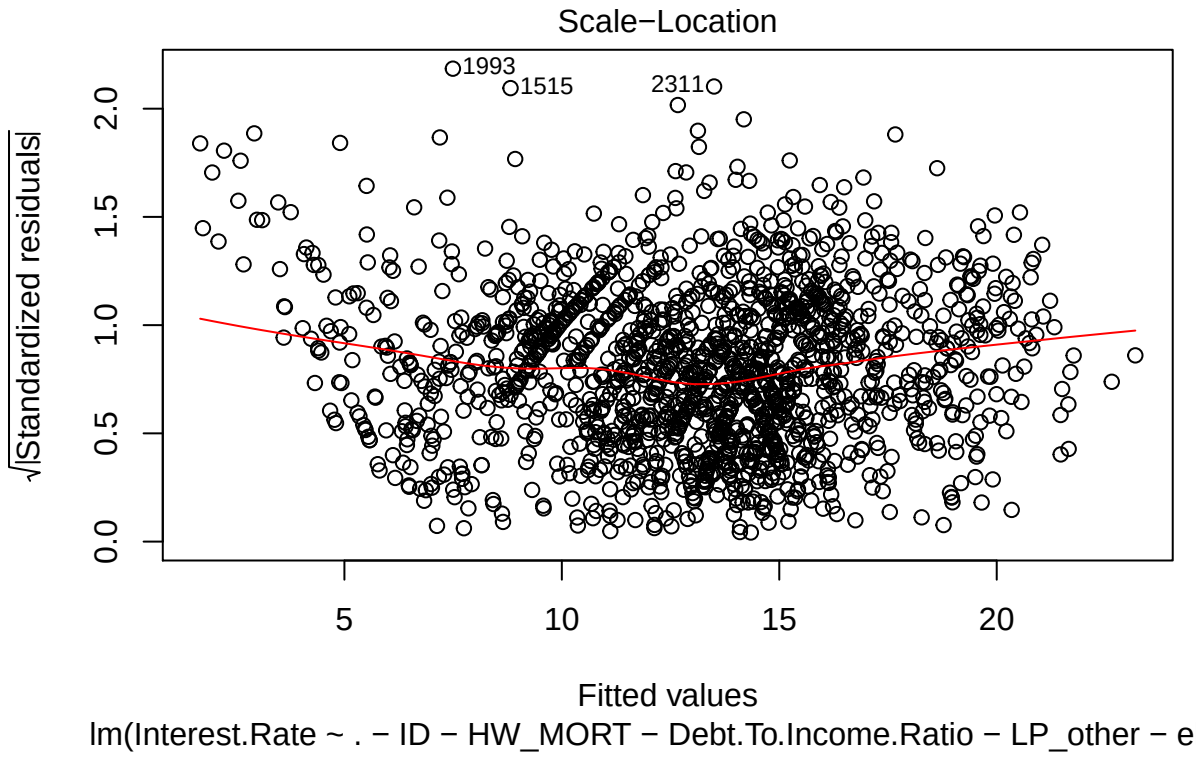



```
shapiro.test(fit$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  fit$residuals
## W = 0.98074, p-value = 3.153e-14
```

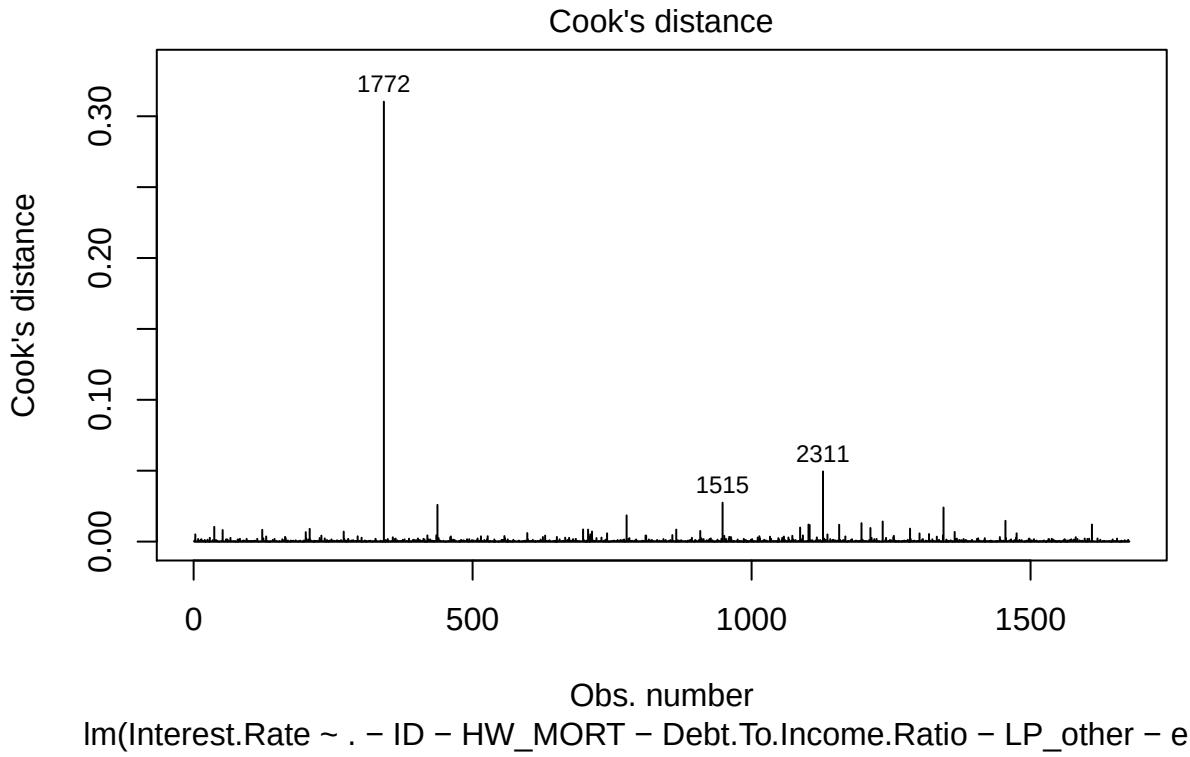
This tells you that there is significant deviation from normality in your residual which is confirmed by shapiro wilk test as well. What you can do next is to include more information to your data or log transform your response [you'll have to transform your predictors too with the response] and build this model again. I'll leave that as an exercise to you. Assuming that this doesn't affect our model much we'll move forward.

```
plot(fit,which=3)
```



This plot tells us whether our residuals are homoscedastic or not. If this plot has a pattern then that mean residuals are heteroscedastic. If that is the case which is not in our case , you can again log transform response and rebuild your model. Why log transform of response in these deviations of residuals? Becuase it brings down scale of response and in turn for residuals. Effect of scale coming down is differences being less prominent.

```
plot(fit,which=4)
```



This gives us cook's distance for all the observations. If cook's distance for any observation is greater than 1 , you should consider removing that observation from your modeling process and rebuild the model. In our cook's distance is less than 1 for all the observations.[dont worry about high cook's distance value for observation number 2311, it still is less than 1]

Cook's distance basically tells how much leverage [effect] one observation has on your model equation. High value of cook's distance [>1] hints at that observation being very different from rest of the data or in other words, that observation being an outlier.

Done with the diagnostic plots.But we havent yet done the crucial part of testing here. Lets look at rmse value for test

```
mean((ld_test$Interest.Rate-predict(fit,newdata=ld_test))**2) %>%
sqrt()
```

```
## [1] 2.010822
```

We see that model performance [RMSE] on test data comes out to be ~2.02. We can check performance of other models with this and then choose the best one having least RMSE.

Overfit & Underfit : Performance on train and test should not be vastly different. [Not true for high bias algortihm such as Decision Trees]. Models which perform very well on train data and not so well on test data are said to be overfitting.

On the other hand models which perform much better on test in comparison to train are called to be underfit. This is not a bad thing as long as you select models based on performance on test data.

How ever overfitting and underfitting of the model gives you hint on how you can improve your model.

There are many other observation you can make from that model equation. For example :

- For high amount requested , interest rates are going to be higher as well
- If someone has been applying for loan at too many places [results in high number of inquiries], they'll get higher interest rates

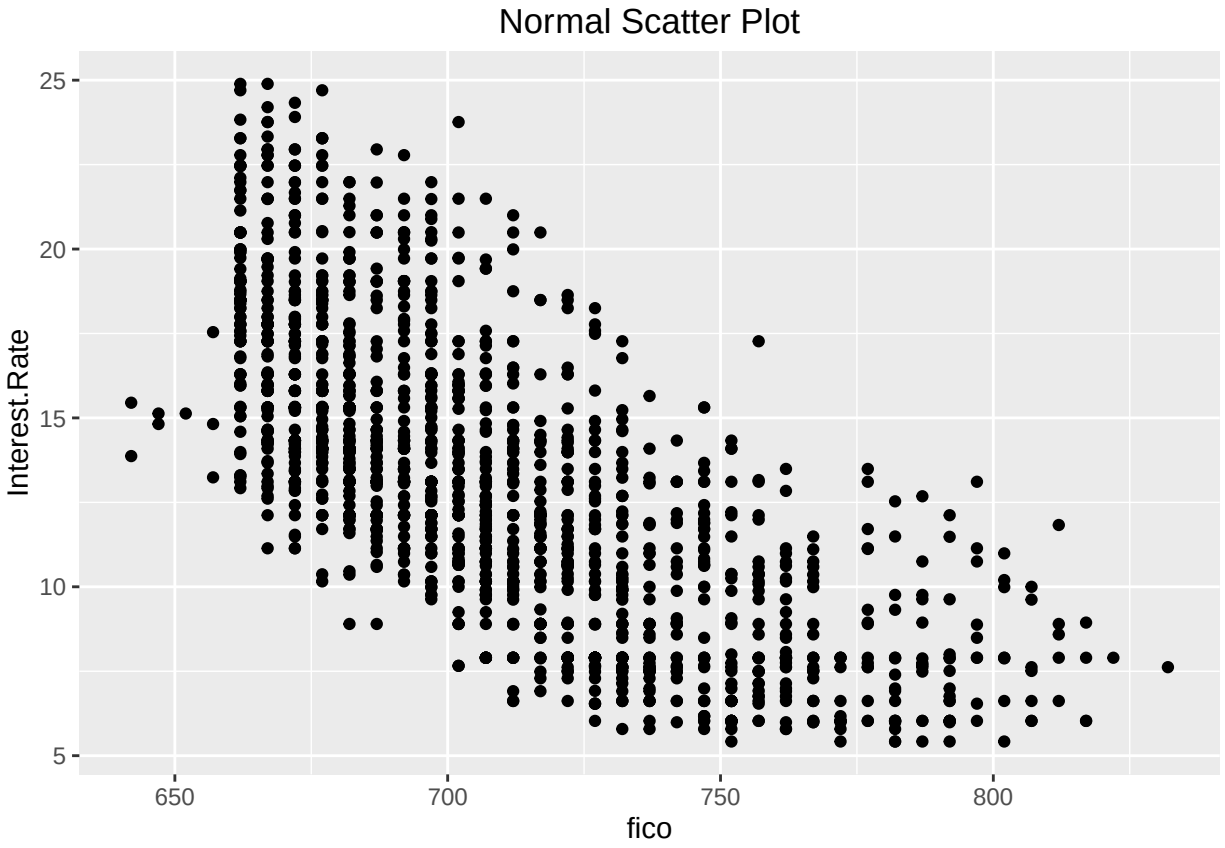
- Where as if you have high fico score , your interest rates are going to be lower.
- If you are applying for a loan for 36 months , your interest will be lower by almost 3% in comparisons to 60 months loan

with these and many more observations , you can come up with good recommendations for your clients.

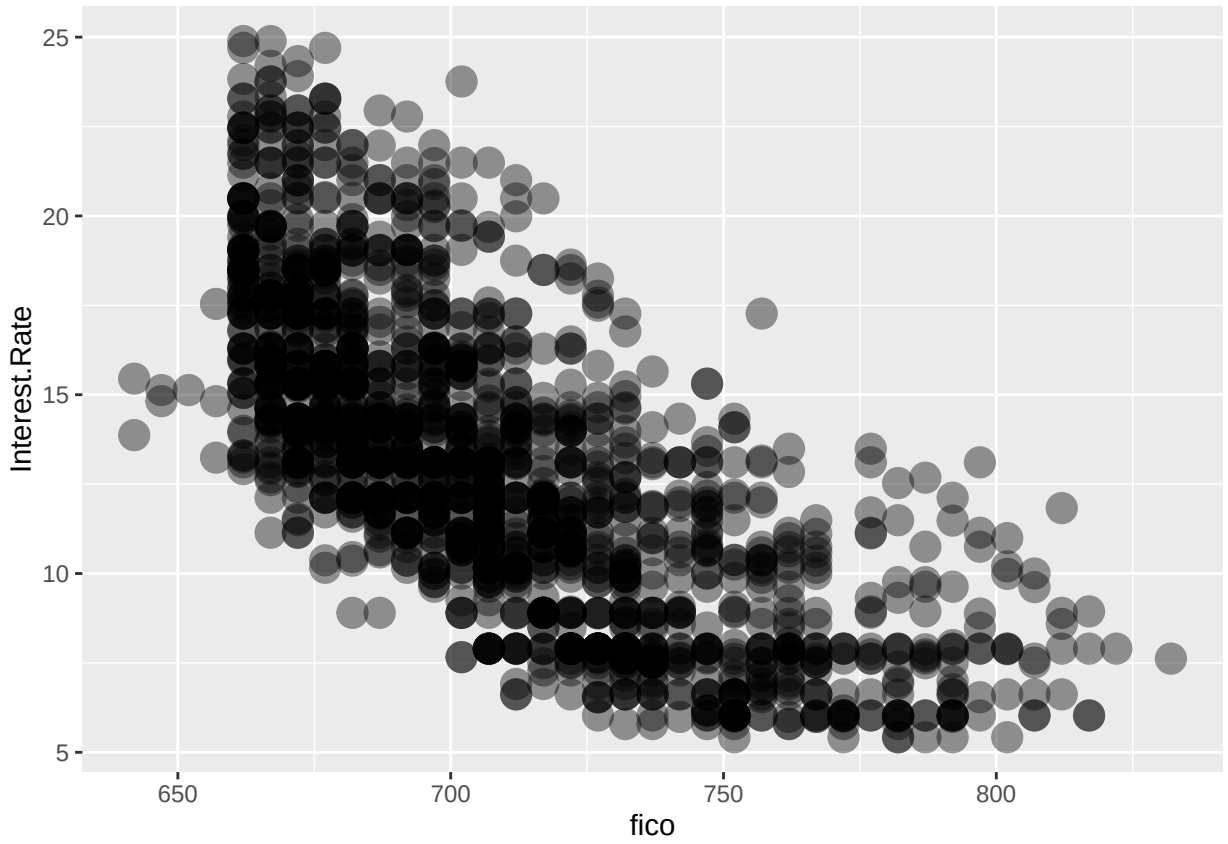
Remember , what we went through here is one of the possible iterations. You can experiment with your own variable transformation etc and come up with different models which might perform better.

Extra Notes on Scatter Plots For Large Data

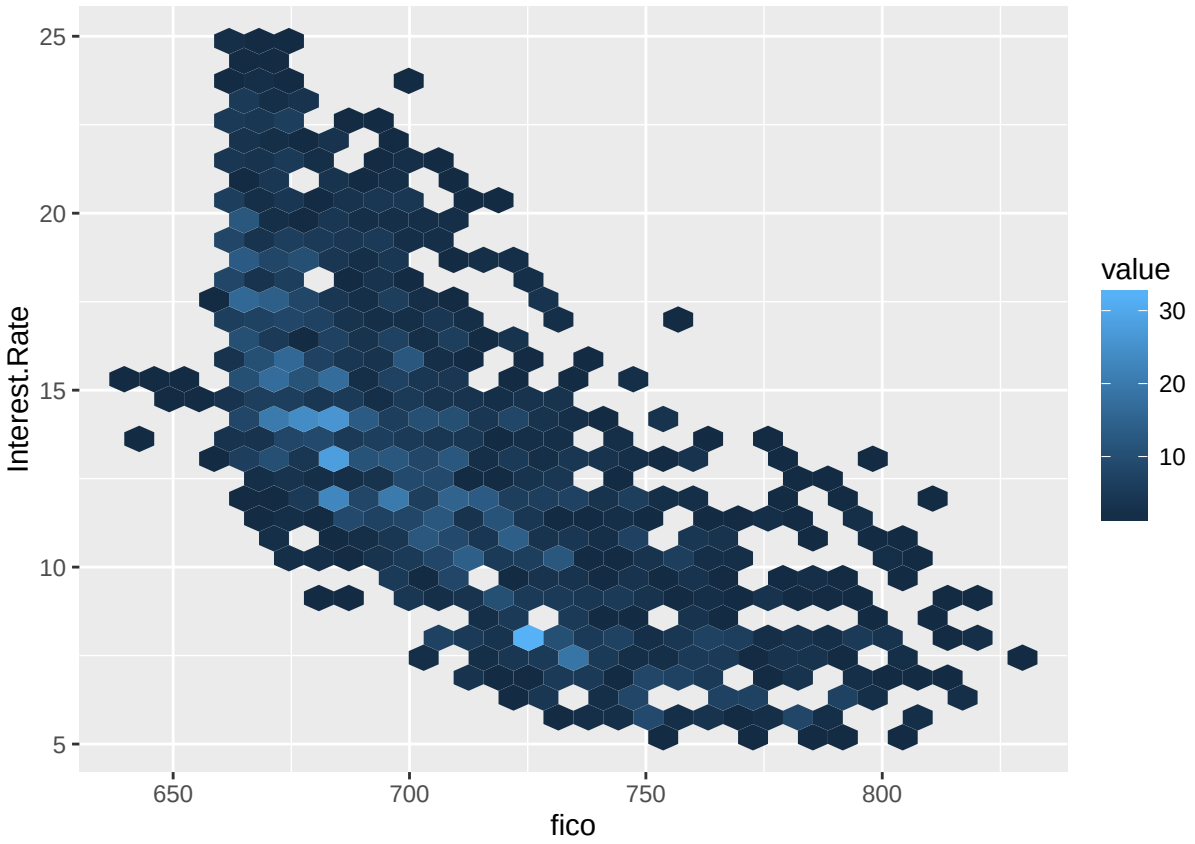
We have seen that as datasets get large , scatter plots start to look like blobs and not much can be deciphered. This happens due to overlapping of points. There is a way we can see what part of the scatter plots are dense and focus on only dense parts to identify a pattern if there exists one. Here are few ways to do the same.



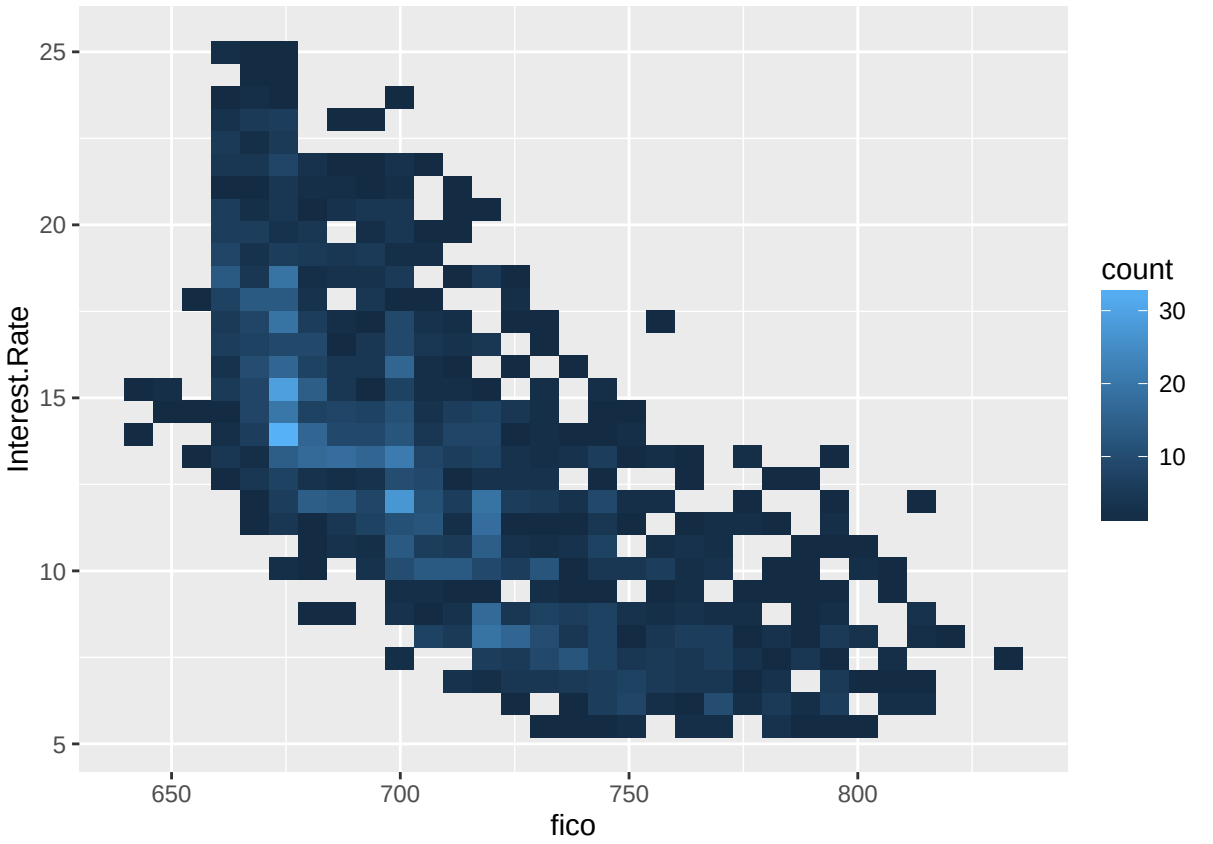

```
# Scatter Plot with density mapped
ggplot(ld_temp,aes(x=fico,y=Interest.Rate))+geom_point(alpha=0.4,size=5)
```




```
#scatter plot with hex binning or 2d binning , you'll need to install package "hexbin" for the same
ggplot(ld_temp,aes(x=fico,y=Interest.Rate))+stat_binhex()
```




```
ggplot(ld_temp,aes(x=fico,y=Interest.Rate))+stat_bin2d()
```



Case Study 2

Online news , blogs and analysis journals are now a days have become chief source of information for many and are leaving traditional TV/News Paper behind everyday. Great thing about things online is that their "performance" can be tracked in real time. Generated data from this tracking can be used for various purposes. In the particular problem that we are going to solve today relates to using this data to make articles better and interesting. But how to measure "betterness" or "popularity" of an article? How many times that article has been shared in the social media can be taken as one big measure of popularity.

This problem's dataset summarizes a heterogeneous set of features about articles published by Mashable in a period of two years. The goal is to predict the number of shares in social media networks (popularity)

Data File : OnlineNewsPopularity.csv

Note: Download updated data.zip file from LMS if you dont find these files in your data folder.

Attribute Information: Attribute Information: 1. url: URL of the article

2. n_tokens_title: Number of words in the title
3. n_tokens_content: Number of words in the content
4. n_unique_tokens: Rate of unique words in the content
5. n_non_stop_words: Rate of non-stop words in the content
6. n_non_stop_unique_tokens: Rate of unique non-stop words in the content
7. num_hrefs: Number of links
8. num_self_hrefs: Number of links to other articles published by Mashable

9. num_imgs: Number of images
10. num_videos: Number of videos
11. average_token_length: Average length of the words in the content
12. num_keywords: Number of keywords in the metadata
13. kw_min_min: Worst keyword (min. shares)
14. kw_max_min: Worst keyword (max. shares)
15. kw_avg_min: Worst keyword (avg. shares)
16. kw_min_max: Best keyword (min. shares)
17. kw_max_max: Best keyword (max. shares)
18. kw_avg_max: Best keyword (avg. shares)
19. kw_min_avg: Avg. keyword (min. shares)
20. kw_max_avg: Avg. keyword (max. shares)
21. kw_avg_avg: Avg. keyword (avg. shares)
22. self_reference_min_shares: Min. shares of referenced articles in Mashable
23. self_reference_max_shares: Max. shares of referenced articles in Mashable
24. self_reference_avg_sharess: Avg. shares of referenced articles in Mashable
25. LDA_00: Closeness to LDA topic 0
26. LDA_01: Closeness to LDA topic 1
27. LDA_02: Closeness to LDA topic 2
28. LDA_03: Closeness to LDA topic 3
29. LDA_04: Closeness to LDA topic 4
30. global_subjectivity: Text subjectivity
31. global_sentiment_polarity: Text sentiment polarity
32. global_rate_positive_words: Rate of positive words in the content
33. global_rate_negative_words: Rate of negative words in the content
34. rate_positive_words: Rate of positive words among non-neutral tokens
35. rate_negative_words: Rate of negative words among non-neutral tokens
36. avg_positive_polarity: Avg. polarity of positive words
37. min_positive_polarity: Min. polarity of positive words
38. max_positive_polarity: Max. polarity of positive words
39. avg_negative_polarity: Avg. polarity of negative words
40. min_negative_polarity: Min. polarity of negative words
41. max_negative_polarity: Max. polarity of negative words
42. title_subjectivity: Title subjectivity
43. title_sentiment_polarity: Title polarity
44. abs_title_subjectivity: Absolute subjectivity level

45. abs_title_sentiment_polarity: Absolute polarity level
46. shares: Number of shares (target)
47. Day of the week when article was published
48. Data Channel for the Article

*_shares : Average Share Numbers for different characteristic values

sp_feat1, sp_feat2 : Properietary features provided by the business

Lets start:

```
setwd("~/Dropbox/March onwards/CBAP with R/Data/")
md=read.csv("OnlineNewsPopularity.csv",stringsAsFactors = F)

#alternatively large csv files can be read faster with fread in data.table
library(data.table)
md=data.frame(fread("OnlineNewsPopularity.csv"))
# we dont need to use any option like stringsAsFactors because by default
# character columns are read as character in this function
```

Lets have a look at our data

```
library(dplyr)
glimpse(md)
```

```
## Observations: 39,644
## Variables: 60
## $ url                          (chr) "http://mashable.com/2013/01/07/a...
## $ n_tokens_title               (int) 12, 9, 9, 9, 13, 10, 8, 12, 11, 1...
## $ n_tokens_content             (int) 219, 255, 211, 531, 1072, 370, 96...
## $ n_unique_tokens              (dbl) 0.6635945, 0.6047431, 0.5751295, ...
## $ n_non_stop_words             (dbl) 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ n_non_stop_unique_tokens     (dbl) 0.8153846, 0.7919463, 0.6638655, ...
## $ num_hrefs                    (int) 4, 3, 3, 9, 19, 2, 21, 20, 2, 4, ...
## $ num_self_hrefs               (int) 2, 1, 1, 0, 19, 2, 20, 20, 0, 1, ...
## $ num_imgs                     (int) 1, 1, 1, 1, 20, 0, 20, 20, 0, 1, ...
## $ num_videos                   (int) 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ...
## $ average_token_length         (dbl) 4.680365, 4.913725, 4.393365, 4.4...
## $ num_keywords                 (int) 5, 4, 6, 7, 7, 9, 10, 9, 7, 5, 8,...
## $ kw_min_min                   (int) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ kw_max_min                   (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ kw_avg_min                   (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ kw_min_max                   (int) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ kw_max_max                   (int) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ kw_avg_max                   (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ kw_min_avg                   (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ kw_max_avg                   (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ kw_avg_avg                   (dbl) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ self_reference_min_shares    (dbl) 496, 0, 918, 0, 545, 8500, 545, 5...
## $ self_reference_max_shares    (dbl) 496, 0, 918, 0, 16000, 8500, 1600...
## $ self_reference_avg_sharess   (dbl) 496.000, 0.000, 918.000, 0.000, 3...
## $ LDA_00                       (dbl) 0.50033120, 0.79975569, 0.2177922...
## $ LDA_01                       (dbl) 0.37827893, 0.05004668, 0.0333344...
## $ LDA_02                       (dbl) 0.04000468, 0.05009625, 0.0333514...
## $ LDA_03                       (dbl) 0.04126265, 0.05010067, 0.0333335...
```

```
## $ LDA_04                       (dbl) 0.04012254, 0.05000071, 0.6821882...
## $ global_subjectivity          (dbl) 0.5216171, 0.3412458, 0.7022222, ...
## $ global_sentiment_polarity    (dbl) 0.09256198, 0.14894781, 0.3233333...
## $ global_rate_positive_words   (dbl) 0.04566210, 0.04313725, 0.0568720...
## $ global_rate_negative_words   (dbl) 0.013698630, 0.015686275, 0.00947...
## $ rate_positive_words          (dbl) 0.7692308, 0.7333333, 0.8571429, ...
## $ rate_negative_words          (dbl) 0.2307692, 0.2666667, 0.1428571, ...
## $ avg_positive_polarity        (dbl) 0.3786364, 0.2869146, 0.4958333, ...
## $ min_positive_polarity        (dbl) 0.10000000, 0.03333333, 0.1000000...
## $ max_positive_polarity        (dbl) 0.7000000, 0.7000000, 1.0000000, ...
## $ avg_negative_polarity        (dbl) -0.3500000, -0.1187500, -0.466666...
## $ min_negative_polarity        (dbl) -0.6000000, -0.1250000, -0.800000...
## $ max_negative_polarity        (dbl) -0.2000000, -0.1000000, -0.133333...
## $ title_subjectivity           (dbl) 0.5000000, 0.0000000, 0.0000000, ...
## $ title_sentiment_polarity     (dbl) -0.1875000, 0.0000000, 0.0000000,...
## $ abs_title_subjectivity       (dbl) 0.00000000, 0.50000000, 0.5000000...
## $ abs_title_sentiment_polarity (dbl) 0.1875000, 0.0000000, 0.0000000, ...
## $ shares                       (int) 593, 711, 1500, 1200, 505, 855, 5...
## $ weekday                      (chr) "Monday", "Monday", "Monday", "Mo...
## $ data_channel                 (chr) "entertainment", "business", "bus...
## $ nnsw_share                   (dbl) 3363.712, 2274.789, 2660.598, 509...
## $ numk_share                   (dbl) 3010.249, 2478.100, 3490.664, 353...
## $ ntt_share                    (dbl) 3393.260, 3397.781, 3397.781, 339...
## $ kw_min_share                 (dbl) 2004.557, 2004.557, 2004.557, 200...
## $ mnp_share                    (dbl) 4380.948, 3100.186, 2639.936, 329...
## $ nv_shares                    (dbl) 2891.471, 2891.471, 2891.471, 289...
## $ minnp_shares                 (dbl) 3387.131, 2789.048, 3221.353, 338...
## $ nsh_shares                   (dbl) 3705.609, 3599.222, 3599.222, 302...
## $ num_im_shares                (dbl) 2573.598, 2573.598, 2573.598, 257...
## $ num_hrefs_shares             (dbl) 3146.210, 3232.165, 3232.165, 270...
## $ sp_feat1                     (dbl) -615.9291, 1582.9984, 4625.6907, ...
## $ sp_feat2                     (dbl) -1244.9805, 5125.8428, 1086.7130,...
```

Lets first create dummy variable for categorical variables `weekday` and `data_channel`. We'll start with looking at what values these variables take and their frequencies.

```
table(md$weekday)
```

```
##
##    Friday    Monday  Saturday    Sunday  Thursday   Tuesday Wednesday
##      5701      6661      2453      2737      7267      7390      7435
```

```
md=md %>%
  mutate(
         wd_Friday=as.numeric(weekday=="Friday"),
         wd_Monday=as.numeric(weekday=="Monday"),
         wd_Sunday=as.numeric(weekday=="Sunday"),
         wd_Thursday=as.numeric(weekday=="Thursday"),
         wd_Tuesday=as.numeric(weekday=="Tuesday"),
         wd_Wednesday=as.numeric(weekday=="Wednesday")
        ) %>%
  select(-weekday)

table(md$data_channel)
```

```
##
```

```
##      business entertainment     lifestyle         other        socmed
##          6258          7057          2099          6134          2323
##          tech         world
##          7346          8427
```

We might have discussed in the class, that in practice we take the category which is least frequent as base. However theoratically taking any category as base doesnt make a difference. In this particular case we will not use the least frequent category as base , but rather chose category "other" base .Reason being , that all other category names are meaningful. At the end of the day, however , this choice doesnt make a difference as far as mathematics involved is concerned because all that matter is that you make `n-1` dummy variables when categorical variable involved has `n` distinct values.

```
md=md %>%
  mutate(
        dc_bus=as.numeric(data_channel=="business"),
        dc_ent=as.numeric(data_channel=="entertainment"),
        dc_life=as.numeric(data_channel=="lifestyle"),
        dc_socmed=as.numeric(data_channel=="socmed"),
        dc_tech=as.numeric(data_channel=="tech"),
        dc_world=as.numeric(data_channel=="world")
      ) %>%
  select(-data_channel)
```

We have not done anything with the variable `url`. One kind of features that we can create is to check if url contains some specific keywords or not. We have selected few keywords to check , you can include more of your own if you want.

```
words=c("google","twitter","facebook","apple",
        "video","new","app","iphone","social","world","2014")
for(word in words){
  md[,paste0("word_",word)]=NA
  for(i in 1:nrow(md)){
    md[,paste0("word_",word)][i]=as.numeric(length(grep(word,md$url[i]))>0)
  }
}
md=md[,-1]
```

We'll start with our modelling process now.

We'll keep some part of data for testing our model. Setting a seed makes our results reproducible , when a random process is involved.

```
set.seed(2)
s=sample(1:nrow(md),0.8*nrow(md))
md_train=md[s,]
md_test=md[-s,]
```

Lets start building our model

```
fit=lm(shares~.,data=md_train)
```

Before we start dropping variables based on p-values or any other significnace measure , we need to remove variables with high VIF values as discussed in the class.

```
library(car)
sort(vif(fit),decreasing = T)[1:10]
```

```
##                  LDA_03                  LDA_04                  LDA_02
##            1.054767e+09            1.007107e+09            9.539390e+08
```

```
##                   LDA_00                   LDA_01         n_non_stop_words
##             8.373366e+08             5.805794e+08             3.816725e+05
##          n_unique_tokens n_non_stop_unique_tokens      rate_positive_words
##             1.745836e+04             1.065067e+04             3.816249e+02
##      rate_negative_words
##             2.640243e+02
```

We'll drop variables with high VIF one by one untile all variables have VIF <5.

```
fit=lm(shares~.-LDA_03,data=md_train)
sort(vif(fit),decreasing = T)[1:10]
```

```
##            n_unique_tokens   n_non_stop_unique_tokens
##                17352.03287                10567.54243
##            n_non_stop_words        rate_positive_words
##                 4437.79743                   21.82742
## self_reference_avg_sharess        rate_negative_words
##                   21.12004                   19.01322
##                 kw_avg_min                 kw_max_min
##                   12.36810                   12.32359
##       average_token_length                 kw_avg_avg
##                   11.98855                   11.02887
```

```
fit=lm(shares~.-LDA_03-n_unique_tokens,data=md_train)
sort(vif(fit),decreasing = T)[1:10]
```

```
##            n_non_stop_words   n_non_stop_unique_tokens
##                2626.251872               2623.831553
## self_reference_avg_sharess        rate_positive_words
##                  21.118237                  18.563776
##        rate_negative_words                 kw_avg_min
##                  16.751359                  12.367254
##                 kw_max_min       average_token_length
##                  12.323114                  11.069952
##                 kw_avg_avg  self_reference_max_shares
##                  11.027706                   9.564423
```

```
fit=lm(shares~.-LDA_03-n_unique_tokens-n_non_stop_words,data=md_train)
sort(vif(fit),decreasing = T)[1:10]
```

```
## self_reference_avg_sharess        rate_positive_words
##                  21.113532                  18.313784
##        rate_negative_words                 kw_avg_min
##                  16.553904                  12.367146
##                 kw_max_min                 kw_avg_avg
##                  12.323025                  11.027680
##       average_token_length  self_reference_max_shares
##                  11.012537                   9.561410
##  global_sentiment_polarity      avg_negative_polarity
##                   7.852565                   7.789074
```

```
fit=lm(shares~.-LDA_03-n_unique_tokens
       -n_non_stop_words
       -self_reference_avg_sharess,data=md_train)
sort(vif(fit),decreasing = T)[1:10]
```

```
##       rate_positive_words       rate_negative_words
```

```
##                 18.311880                 16.552471
##               kw_avg_min               kw_max_min
##                12.366655                12.321515
##               kw_avg_avg     average_token_length
##                11.027453                11.012502
## global_sentiment_polarity    avg_negative_polarity
##                 7.852552                 7.789012
##                 dc_world               kw_max_avg
##                 7.113409                 6.822215
```

```
fit=lm(shares~.-LDA_03-n_unique_tokens
       -n_non_stop_words
       -self_reference_avg_sharess
       -rate_positive_words,data=md_train)
sort(vif(fit),decreasing = T)[1:10]
```

```
##                kw_avg_min                kw_max_min
##                 12.365854                 12.320802
##                kw_avg_avg       rate_negative_words
##                 11.015263                  8.769939
##  global_sentiment_polarity     avg_negative_polarity
##                  7.852492                  7.788372
##                  dc_world               kw_max_avg
##                  7.111351                  6.816404
## global_rate_negative_words                   dc_tech
##                  6.446044                  6.232057
```

```
fit=lm(shares~.-LDA_03-n_unique_tokens
       -n_non_stop_words
       -self_reference_avg_sharess
       -rate_positive_words
       -kw_avg_min,data=md_train)
sort(vif(fit),decreasing = T)[1:10]
```

```
##                kw_avg_avg       rate_negative_words
##                 10.344654                  8.769848
##  global_sentiment_polarity     avg_negative_polarity
##                  7.851676                  7.788361
##                  dc_world               kw_max_avg
##                  7.108346                  6.552682
## global_rate_negative_words                   dc_tech
##                  6.445904                  6.227078
##     avg_positive_polarity                     LDA_02
##                  6.015267                  5.881956
```

```
fit=lm(shares~.-LDA_03-n_unique_tokens
       -n_non_stop_words
       -self_reference_avg_sharess
       -rate_positive_words
       -kw_avg_min-kw_avg_avg,data=md_train)
sort(vif(fit),decreasing = T)[1:10]
```

```
##       rate_negative_words  global_sentiment_polarity
##                  8.769412                  7.848483
##     avg_negative_polarity                  dc_world
##                  7.787270                  6.921365
```

```
## global_rate_negative_words                    dc_tech
##                    6.445884                   6.109705
##      avg_positive_polarity      min_negative_polarity
##                    6.014431                   5.795292
##                      LDA_04                     dc_bus
##                    5.724221                   5.691880
```

```
fit=lm(shares~.-LDA_03-n_unique_tokens
       -n_non_stop_words
       -self_reference_avg_sharess
       -rate_positive_words
       -kw_avg_min-kw_avg_avg
       -rate_negative_words,data=md_train)
sort(vif(fit),decreasing = T)[1:10]
```

```
##     avg_negative_polarity global_sentiment_polarity
##                  7.765578                   7.289767
##                  dc_world                    dc_tech
##                  6.917681                   6.109083
##     avg_positive_polarity                     LDA_04
##                  5.854440                   5.723826
##     min_negative_polarity                     dc_bus
##                  5.705331                   5.690173
##                    LDA_02                     LDA_00
##                  5.662139                   5.305756
```

```
fit=lm(shares~.-LDA_03-n_unique_tokens
       -n_non_stop_words
       -self_reference_avg_sharess
       -rate_positive_words
       -kw_avg_min-kw_avg_avg
       -rate_negative_words
       -avg_negative_polarity,data=md_train)
sort(vif(fit),decreasing = T)[1:10]
```

```
##                  dc_world global_sentiment_polarity
##                  6.917290                   6.477676
##                   dc_tech                     LDA_04
##                  6.108317                   5.720821
##                    dc_bus     avg_positive_polarity
##                  5.687931                   5.679878
##                    LDA_02                     LDA_00
##                  5.657695                   5.305360
##                kw_max_max                kw_min_min
##                  4.941222                   3.852872
```

```
fit=lm(shares~.-LDA_03-n_unique_tokens
       -n_non_stop_words
       -self_reference_avg_sharess
       -rate_positive_words
       -kw_avg_min-kw_avg_avg
       -rate_negative_words
       -avg_negative_polarity
       -dc_world,data=md_train)
sort(vif(fit),decreasing = T)[1:10]
```

```
## global_sentiment_polarity     avg_positive_polarity
##                   6.477568                  5.679701
##                kw_max_max                    LDA_04
##                   4.907400                  4.725131
##                    LDA_00                    dc_bus
##                   4.694847                  4.029956
##                kw_min_min                   dc_tech
##                   3.852832                  3.669442
##                kw_avg_max              wd_Wednesday
##                   3.477458                  3.306595
```

```
fit=lm(shares~.-LDA_03-n_unique_tokens
       -n_non_stop_words
       -self_reference_avg_sharess
       -rate_positive_words
       -kw_avg_min-kw_avg_avg
       -rate_negative_words
       -avg_negative_polarity
       -dc_world
       -global_sentiment_polarity,data=md_train)
sort(vif(fit),decreasing = T)[1:10]
```

```
##            kw_max_max                LDA_04                LDA_00
##              4.907400              4.724700              4.693558
##                dc_bus avg_positive_polarity            kw_min_min
##              4.029878              3.930132              3.852800
##               dc_tech            kw_avg_max          wd_Wednesday
##              3.669441              3.477128              3.306552
##            wd_Tuesday
##              3.299710
```

Now all VIF values are below 5 . We can proceed with the modelling process , in subsequent steps we'll not use the variables which we excluded in the step above.

```
md_train=md_train %>%
  select(-LDA_03,-n_unique_tokens,
       -n_non_stop_words,
       -self_reference_avg_sharess,
       -rate_positive_words,
       -kw_avg_min-kw_avg_avg,
       -rate_negative_words,
       -avg_negative_polarity,
       -dc_world,
       -global_sentiment_polarity)
```

```
fit=lm(shares~.,data=md_train)
summary(fit)
```

```
##
## Call:
## lm(formula = shares ~ ., data = md_train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -86490  -3435   -134   3198 229955
##
```

```
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  -7.945e+03  1.505e+03  -5.278 1.32e-07 ***
## n_tokens_title               -2.332e+01  1.777e+01  -1.313 0.189354
## n_tokens_content              8.922e-02  9.867e-02   0.904 0.365876
## n_non_stop_unique_tokens      8.553e+00  8.637e+00   0.990 0.322042
## num_hrefs                    -4.427e+00  3.926e+00  -1.128 0.259476
## num_self_hrefs                2.210e+00  9.504e+00   0.233 0.816096
## num_imgs                     -1.108e+00  4.942e+00  -0.224 0.822660
## num_videos                   -7.523e+00  8.911e+00  -0.844 0.398556
## average_token_length          5.650e+01  5.764e+01   0.980 0.327042
## num_keywords                  3.751e+01  2.759e+01   1.359 0.174048
## kw_min_min                   -1.864e+00  8.843e-01  -2.107 0.035096 *
## kw_max_min                    1.029e-02  3.024e-02   0.340 0.733730
## kw_avg_min                   -6.619e-02  1.763e-01  -0.376 0.707285
## kw_min_max                   -6.194e-04  6.584e-04  -0.941 0.346775
## kw_max_max                   -3.094e-04  3.238e-04  -0.956 0.339306
## kw_avg_max                   -5.396e-04  4.612e-04  -1.170 0.242023
## kw_min_avg                   -4.547e-02  4.158e-02  -1.093 0.274216
## kw_max_avg                   -5.542e-02  1.373e-02  -4.037 5.42e-05 ***
## kw_avg_avg                    4.537e-01  7.888e-02   5.751 8.94e-09 ***
## self_reference_min_shares     2.391e-03  1.841e-03   1.299 0.194024
## self_reference_max_shares     7.504e-04  9.081e-04   0.826 0.408578
## LDA_00                        4.522e+02  2.612e+02   1.731 0.083476 .
## LDA_01                        1.920e+02  1.992e+02   0.964 0.335090
## LDA_02                       -8.575e+00  2.034e+02  -0.042 0.966381
## LDA_04                        3.827e+01  2.418e+02   0.158 0.874267
## global_subjectivity           9.667e+02  4.342e+02   2.226 0.026010 *
## global_rate_positive_words   -6.105e+03  2.414e+03  -2.529 0.011428 *
## global_rate_negative_words    4.150e+03  3.596e+03   1.154 0.248546
## avg_positive_polarity         3.419e+02  5.980e+02   0.572 0.567456
## min_positive_polarity        -2.082e+03  5.990e+02  -3.476 0.000509 ***
## max_positive_polarity        -6.441e+01  2.306e+02  -0.279 0.779978
## min_negative_polarity         3.776e+01  1.652e+02   0.229 0.819187
## max_negative_polarity         5.146e+02  3.862e+02   1.333 0.182702
## title_subjectivity            1.568e+02  1.494e+02   1.049 0.294057
## title_sentiment_polarity      2.363e+01  1.360e+02   0.174 0.862006
## abs_title_subjectivity        3.928e+02  1.983e+02   1.981 0.047640 *
## abs_title_sentiment_polarity -8.449e+01  2.154e+02  -0.392 0.694804
## nnsw_share                    2.863e-01  9.631e-03  29.727  < 2e-16 ***
## numk_share                   -1.623e-01  1.594e-01  -1.018 0.308839
## ntt_share                     3.880e-01  1.687e-01   2.301 0.021415 *
## kw_min_share                  1.762e-01  3.784e-01   0.466 0.641462
## mnp_share                     1.564e-01  4.782e-02   3.272 0.001071 **
## nv_shares                     1.914e-01  4.619e-02   4.144 3.42e-05 ***
## minnp_shares                  8.990e-02  6.197e-02   1.451 0.146890
## nsh_shares                    2.216e-01  7.473e-02   2.966 0.003024 **
## num_im_shares                 1.184e-01  3.759e-02   3.149 0.001641 **
## num_hrefs_shares              1.509e-01  3.256e-02   4.633 3.61e-06 ***
## sp_feat1                      2.355e+00  8.430e-03 279.307  < 2e-16 ***
## sp_feat2                      7.926e-03  1.579e-02   0.502 0.615642
## wd_Friday                    -8.369e+01  1.518e+02  -0.551 0.581398
## wd_Monday                     5.286e+01  1.487e+02   0.355 0.722260
## wd_Sunday                     7.227e+01  1.744e+02   0.414 0.678533
```

```
## wd_Thursday                -2.001e+02  1.470e+02  -1.361 0.173493
## wd_Tuesday                 -1.224e+02  1.466e+02  -0.835 0.403804
## wd_Wednesday               -8.869e+01  1.467e+02  -0.604 0.545534
## dc_bus                      1.329e+01  1.741e+02   0.076 0.939173
## dc_ent                     -1.296e+02  1.240e+02  -1.045 0.295980
## dc_life                    -3.383e+02  1.826e+02  -1.853 0.063883 .
## dc_socmed                  -2.348e+01  1.701e+02  -0.138 0.890228
## dc_tech                    -6.417e+01  1.553e+02  -0.413 0.679421
## word_google                -2.015e+02  1.814e+02  -1.111 0.266774
## word_twitter                1.457e+02  1.892e+02   0.770 0.441375
## word_facebook               2.332e+02  1.954e+02   1.194 0.232599
## word_apple                  6.764e+02  2.656e+02   2.546 0.010886 *
## word_video                 -2.258e+02  1.983e+02  -1.139 0.254781
## word_new                    6.798e+01  1.976e+02   0.344 0.730872
## word_app                   -2.754e+02  1.776e+02  -1.551 0.120972
## word_iphone                 5.686e+02  2.647e+02   2.148 0.031724 *
## word_social                 1.798e+00  2.709e+02   0.007 0.994704
## word_world                 -7.478e+01  2.655e+02  -0.282 0.778201
## word_2014                  -1.462e+02  7.901e+01  -1.851 0.064185 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5601 on 31644 degrees of freedom
## Multiple R-squared:  0.7471, Adjusted R-squared:  0.7465
## F-statistic:  1335 on 70 and 31644 DF,  p-value: < 2.2e-16
```

You can see that there are lot of variables which are not significant according to p-values. We can drop them one by one. There is no automated function in R to drop variables from the model based on p-values. However we have a function called `step` which drops variable based on AIC value. We can first run `step` function which removes most of the insignificant variables , we can then drop the reminaing ones based on p-values. Lets do that.

```
fit=step(fit)
```

This steps takes some time to run, dropping noisy variables one by one. As number of variables in model go down, it may become a little faster. Have patience and let it run.

```
summary(fit)
```

```
##
## Call:
## lm(formula = shares ~ kw_min_min + kw_avg_max + kw_max_avg +
##     kw_avg_avg + self_reference_min_shares + LDA_00 + global_subjectivity +
##     global_rate_positive_words + min_positive_polarity + max_negative_polarity +
##     abs_title_subjectivity + nnsw_share + ntt_share + mnp_share +
##     nv_shares + minnp_shares + nsh_shares + num_im_shares + num_hrefs_shares +
##     sp_feat1 + wd_Thursday + dc_life + word_apple + word_app +
##     word_iphone + word_2014, data = md_train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -86861  -3447   -122   3206 229793
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                -7.364e+03  6.425e+02 -11.461  < 2e-16 ***
```

```
## kw_min_min                 -1.430e+00  5.491e-01  -2.605 0.009192 **
## kw_avg_max                 -8.736e-04  3.448e-04  -2.534 0.011289 *
## kw_max_avg                 -4.555e-02  1.019e-02  -4.471 7.82e-06 ***
## kw_avg_avg                  3.976e-01  5.288e-02   7.520 5.60e-14 ***
## self_reference_min_shares   3.181e-03  1.610e-03   1.976 0.048161 *
## LDA_00                      5.208e+02  1.288e+02   4.044 5.26e-05 ***
## global_subjectivity         1.325e+03  3.362e+02   3.939 8.19e-05 ***
## global_rate_positive_words -5.529e+03  2.213e+03  -2.498 0.012482 *
## min_positive_polarity      -1.829e+03  4.785e+02  -3.822 0.000133 ***
## max_negative_polarity       5.587e+02  3.601e+02   1.551 0.120850
## abs_title_subjectivity      3.090e+02  1.693e+02   1.825 0.068036 .
## nnsw_share                  2.858e-01  9.591e-03  29.800  < 2e-16 ***
## ntt_share                   2.849e-01  1.494e-01   1.907 0.056526 .
## mnp_share                   1.473e-01  4.700e-02   3.135 0.001721 **
## nv_shares                   1.780e-01  4.259e-02   4.180 2.92e-05 ***
## minnp_shares                8.954e-02  5.546e-02   1.614 0.106441
## nsh_shares                  2.361e-01  7.363e-02   3.206 0.001348 **
## num_im_shares               1.211e-01  3.306e-02   3.662 0.000251 ***
## num_hrefs_shares            1.381e-01  2.845e-02   4.853 1.22e-06 ***
## sp_feat1                    2.355e+00  8.410e-03 280.028  < 2e-16 ***
## wd_Thursday                -1.497e+02  8.147e+01  -1.837 0.066198 .
## dc_life                    -2.887e+02  1.431e+02  -2.017 0.043741 *
## word_apple                  6.766e+02  2.650e+02   2.553 0.010690 *
## word_app                   -2.811e+02  1.772e+02  -1.587 0.112633
## word_iphone                 5.522e+02  2.638e+02   2.093 0.036326 *
## word_2014                  -1.617e+02  7.388e+01  -2.188 0.028677 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5600 on 31688 degrees of freedom
## Multiple R-squared:  0.7469, Adjusted R-squared:  0.7467
## F-statistic:  3596 on 26 and 31688 DF,  p-value: < 2.2e-16
```

Function `formula` on the model object will give you variables remaining in the model, then we can drop the variables which are still with p-values higher than 0.10.

```
formula(fit)
```

```
## shares ~ kw_min_min + kw_avg_max + kw_max_avg + kw_avg_avg +
##     self_reference_min_shares + LDA_00 + global_subjectivity +
##     global_rate_positive_words + min_positive_polarity + max_negative_polarity +
##     abs_title_subjectivity + nnsw_share + ntt_share + mnp_share +
##     nv_shares + minnp_shares + nsh_shares + num_im_shares + num_hrefs_shares +
##     sp_feat1 + wd_Thursday + dc_life + word_apple + word_app +
##     word_iphone + word_2014
```

```
fit=lm(shares ~ kw_min_min + kw_avg_max + kw_max_avg + kw_avg_avg +
    self_reference_min_shares + LDA_00 + global_subjectivity +
    global_rate_positive_words + min_positive_polarity  +
    abs_title_subjectivity + nnsw_share + ntt_share + mnp_share +
    nv_shares + minnp_shares + nsh_shares + num_im_shares + num_hrefs_shares +
    sp_feat1 + wd_Thursday + dc_life + word_apple + word_app +
    word_iphone + word_2014,data=md_train)
summary(fit)
```

```
##
```

```
## Call:
## lm(formula = shares ~ kw_min_min + kw_avg_max + kw_max_avg +
##     kw_avg_avg + self_reference_min_shares + LDA_00 + global_subjectivity +
##     global_rate_positive_words + min_positive_polarity + abs_title_subjectivity +
##     nnsw_share + ntt_share + mnp_share + nv_shares + minnp_shares +
##     nsh_shares + num_im_shares + num_hrefs_shares + sp_feat1 +
##     wd_Thursday + dc_life + word_apple + word_app + word_iphone +
##     word_2014, data = md_train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -86825  -3449   -123   3203 229803
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -7.323e+03  6.420e+02 -11.407  < 2e-16 ***
## kw_min_min                  -1.447e+00  5.490e-01  -2.636 0.008382 **
## kw_avg_max                  -8.896e-04  3.446e-04  -2.581 0.009844 **
## kw_max_avg                  -4.562e-02  1.019e-02  -4.477 7.58e-06 ***
## kw_avg_avg                   3.976e-01  5.288e-02   7.519 5.65e-14 ***
## self_reference_min_shares    3.115e-03  1.609e-03   1.936 0.052912 .
## LDA_00                       5.175e+02  1.288e+02   4.019 5.86e-05 ***
## global_subjectivity          1.232e+03  3.310e+02   3.724 0.000197 ***
## global_rate_positive_words -5.455e+03  2.213e+03  -2.466 0.013687 *
## min_positive_polarity      -1.897e+03  4.765e+02  -3.981 6.88e-05 ***
## abs_title_subjectivity       3.085e+02  1.693e+02   1.822 0.068457 .
## nnsw_share                   2.858e-01  9.591e-03  29.799  < 2e-16 ***
## ntt_share                    2.843e-01  1.494e-01   1.903 0.057066 .
## mnp_share                    1.248e-01  4.470e-02   2.792 0.005235 **
## nv_shares                    1.770e-01  4.259e-02   4.156 3.25e-05 ***
## minnp_shares                 9.932e-02  5.510e-02   1.802 0.071484 .
## nsh_shares                   2.326e-01  7.360e-02   3.160 0.001581 **
## num_im_shares                1.211e-01  3.306e-02   3.664 0.000248 ***
## num_hrefs_shares             1.401e-01  2.842e-02   4.930 8.26e-07 ***
## sp_feat1                     2.355e+00  8.410e-03 280.037  < 2e-16 ***
## wd_Thursday                 -1.498e+02  8.147e+01  -1.839 0.065904 .
## dc_life                     -2.850e+02  1.431e+02  -1.991 0.046469 *
## word_apple                   6.768e+02  2.650e+02   2.554 0.010660 *
## word_app                    -2.802e+02  1.772e+02  -1.582 0.113709
## word_iphone                  5.508e+02  2.638e+02   2.088 0.036779 *
## word_2014                   -1.572e+02  7.383e+01  -2.129 0.033264 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5600 on 31689 degrees of freedom
## Multiple R-squared:  0.7468, Adjusted R-squared:  0.7466
## F-statistic:  3740 on 25 and 31689 DF,  p-value: < 2.2e-16
```

Next we drop variable `word_app`

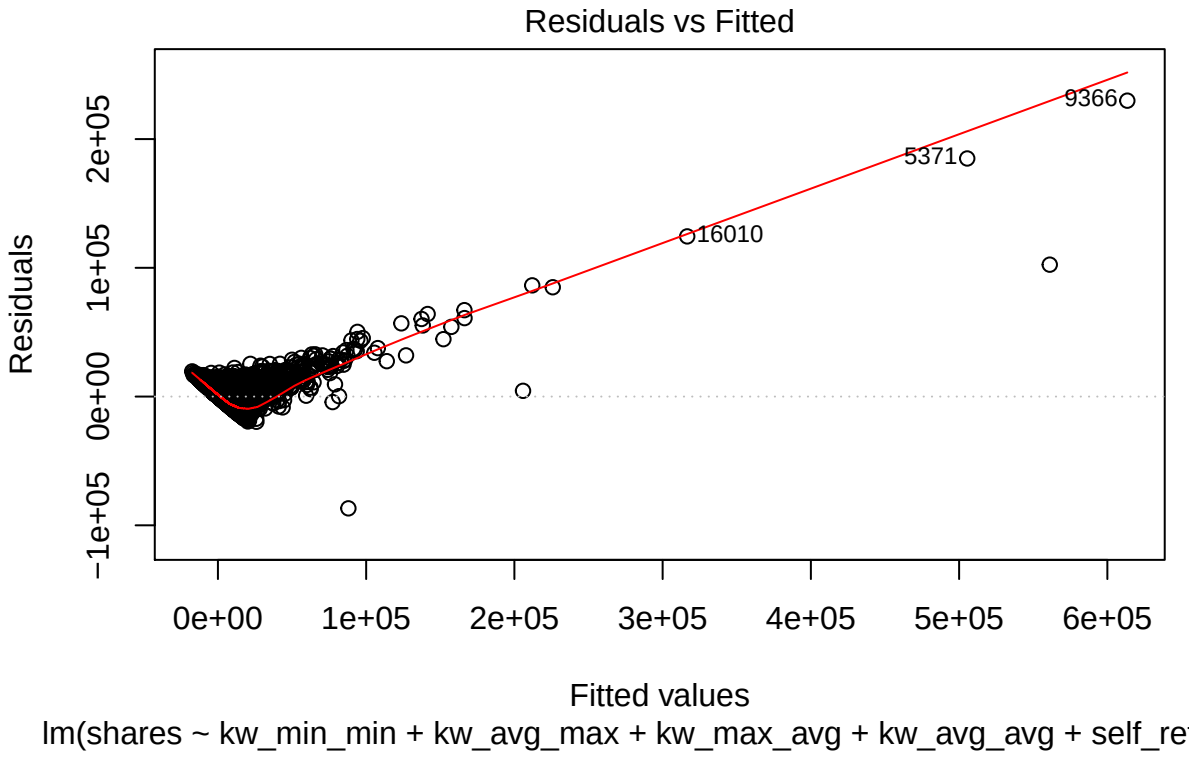
```
fit=lm(shares ~ kw_min_min + kw_avg_max + kw_max_avg + kw_avg_avg +
    self_reference_min_shares + LDA_00 + global_subjectivity +
    global_rate_positive_words + min_positive_polarity  +
    abs_title_subjectivity + nnsw_share + ntt_share + mnp_share +
    nv_shares + minnp_shares + nsh_shares + num_im_shares + num_hrefs_shares +
```

```
    sp_feat1 + wd_Thursday + dc_life + word_apple +
    word_iphone + word_2014,data=md_train)
summary(fit)
```

```
##
## Call:
## lm(formula = shares ~ kw_min_min + kw_avg_max + kw_max_avg +
##     kw_avg_avg + self_reference_min_shares + LDA_00 + global_subjectivity +
##     global_rate_positive_words + min_positive_polarity + abs_title_subjectivity +
##     nnsw_share + ntt_share + mnp_share + nv_shares + minnp_shares +
##     nsh_shares + num_im_shares + num_hrefs_shares + sp_feat1 +
##     wd_Thursday + dc_life + word_apple + word_iphone + word_2014,
##     data = md_train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -86812  -3449   -127   3202 229831
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -7.325e+03  6.420e+02 -11.409  < 2e-16 ***
## kw_min_min                  -1.456e+00  5.490e-01  -2.652 0.008015 **
## kw_avg_max                  -8.857e-04  3.446e-04  -2.570 0.010172 *
## kw_max_avg                  -4.552e-02  1.019e-02  -4.468 7.92e-06 ***
## kw_avg_avg                   3.971e-01  5.288e-02   7.511 6.03e-14 ***
## self_reference_min_shares    3.125e-03  1.609e-03   1.942 0.052114 .
## LDA_00                       5.188e+02  1.288e+02   4.029 5.61e-05 ***
## global_subjectivity          1.231e+03  3.310e+02   3.719 0.000200 ***
## global_rate_positive_words -5.467e+03  2.213e+03  -2.471 0.013478 *
## min_positive_polarity      -1.890e+03  4.765e+02  -3.967 7.29e-05 ***
## abs_title_subjectivity       3.077e+02  1.693e+02   1.817 0.069188 .
## nnsw_share                   2.858e-01  9.591e-03  29.798  < 2e-16 ***
## ntt_share                    2.817e-01  1.494e-01   1.885 0.059417 .
## mnp_share                    1.248e-01  4.471e-02   2.792 0.005242 **
## nv_shares                    1.775e-01  4.258e-02   4.169 3.07e-05 ***
## minnp_shares                 9.905e-02  5.511e-02   1.798 0.072264 .
## nsh_shares                   2.330e-01  7.360e-02   3.166 0.001549 **
## num_im_shares                1.211e-01  3.306e-02   3.664 0.000249 ***
## num_hrefs_shares             1.403e-01  2.842e-02   4.935 8.07e-07 ***
## sp_feat1                     2.355e+00  8.410e-03 280.027  < 2e-16 ***
## wd_Thursday                 -1.508e+02  8.147e+01  -1.851 0.064199 .
## dc_life                     -2.924e+02  1.431e+02  -2.044 0.040965 *
## word_apple                   4.072e+02  2.029e+02   2.007 0.044809 *
## word_iphone                  5.385e+02  2.637e+02   2.042 0.041118 *
## word_2014                   -1.567e+02  7.383e+01  -2.123 0.033750 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5600 on 31690 degrees of freedom
## Multiple R-squared:  0.7468, Adjusted R-squared:  0.7466
## F-statistic:  3895 on 24 and 31690 DF,  p-value: < 2.2e-16
```

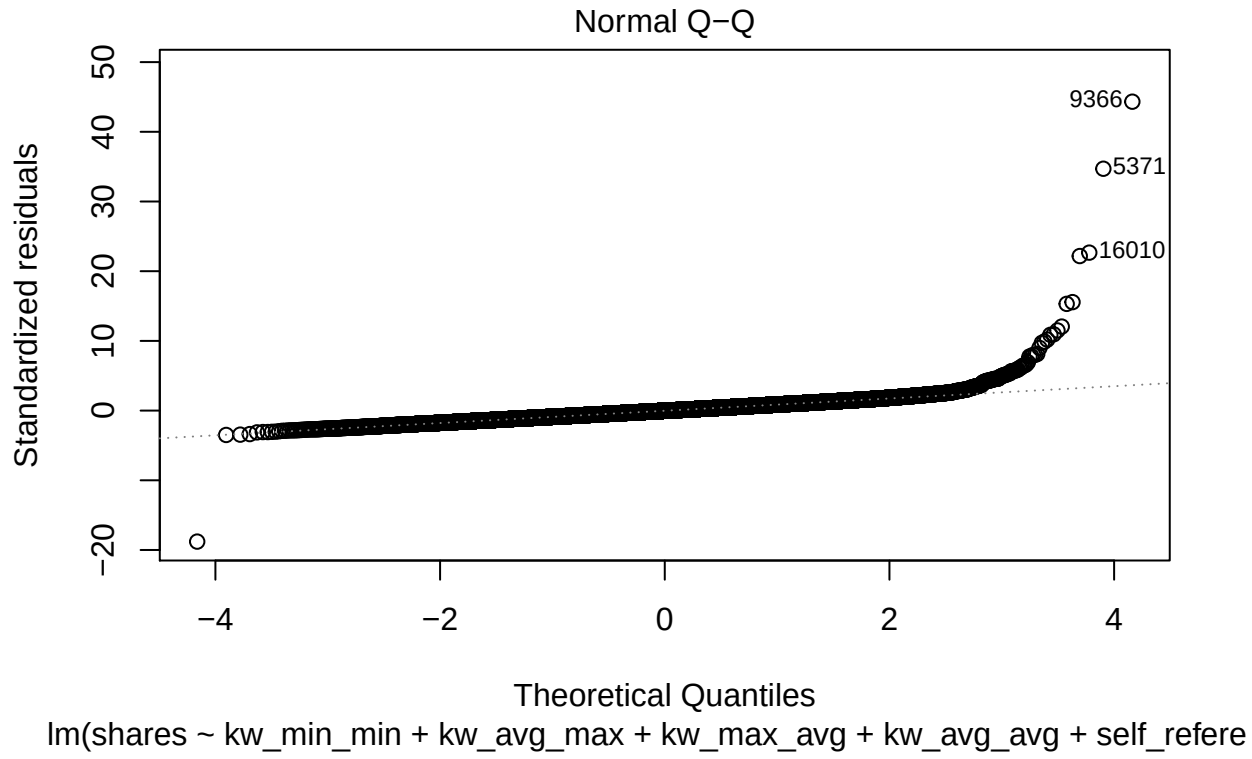
Now all the variables in our model are significant. Lets look at model diagnostic plots

```
plot(fit,which=1)
```



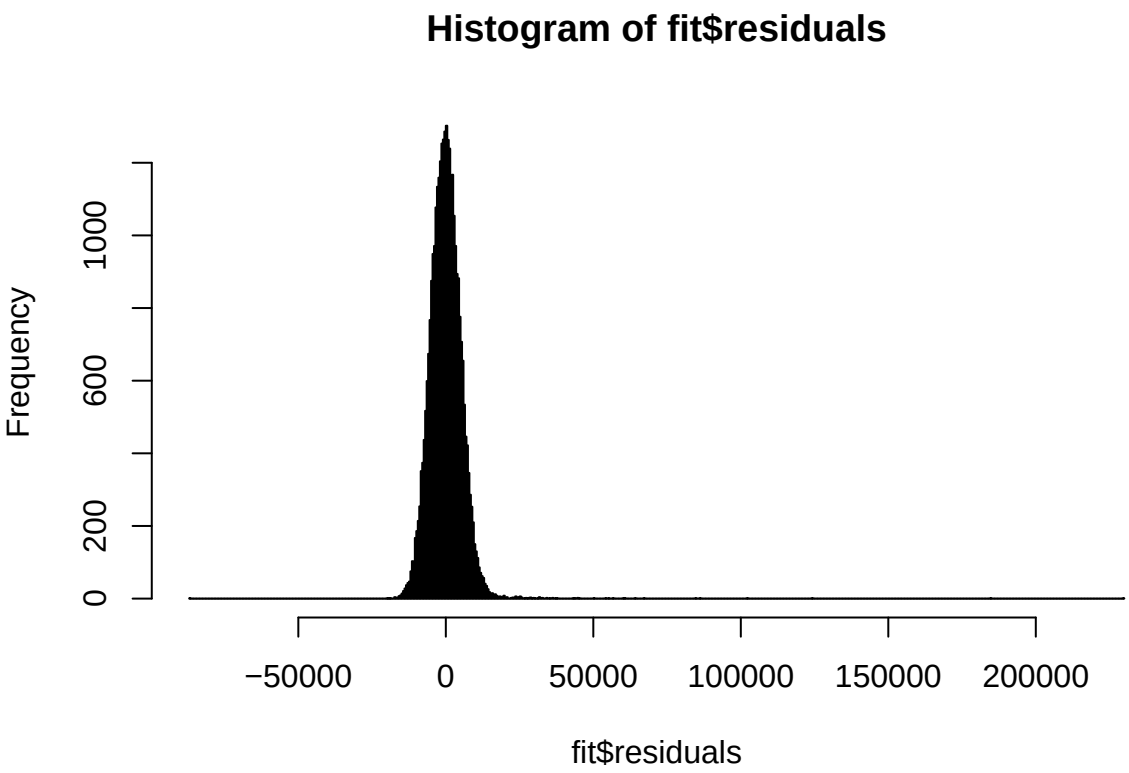
This ideally should be random and devoid of any pattern. If you see a pattern as you see here , you should look at pair wise plot of residuals with variables in the model and see any of these need transformation to achieve linearity with the response variable

```
plot(fit,which=2)
```

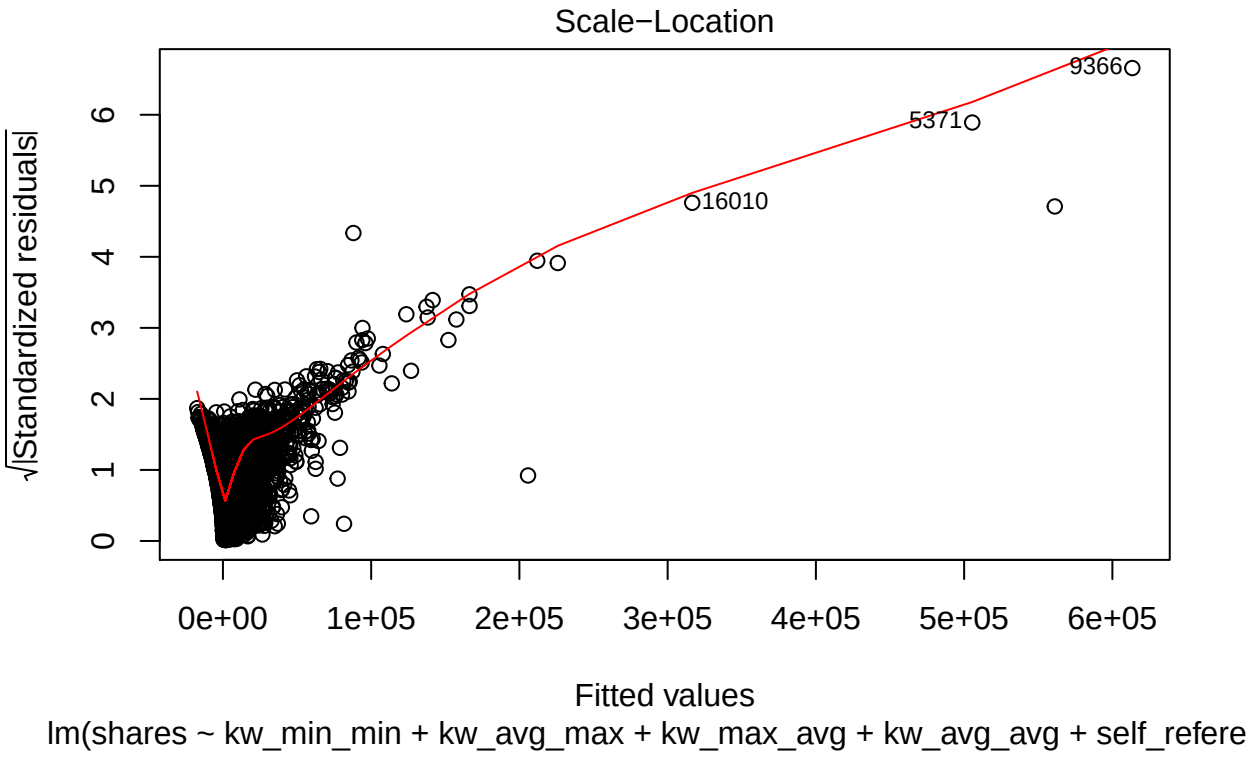


This tells us that , residual are mostly normal with slight deviation, you can confirm this by looking at histogram of residual also.

```
hist(fit$residuals,breaks=1000)
```

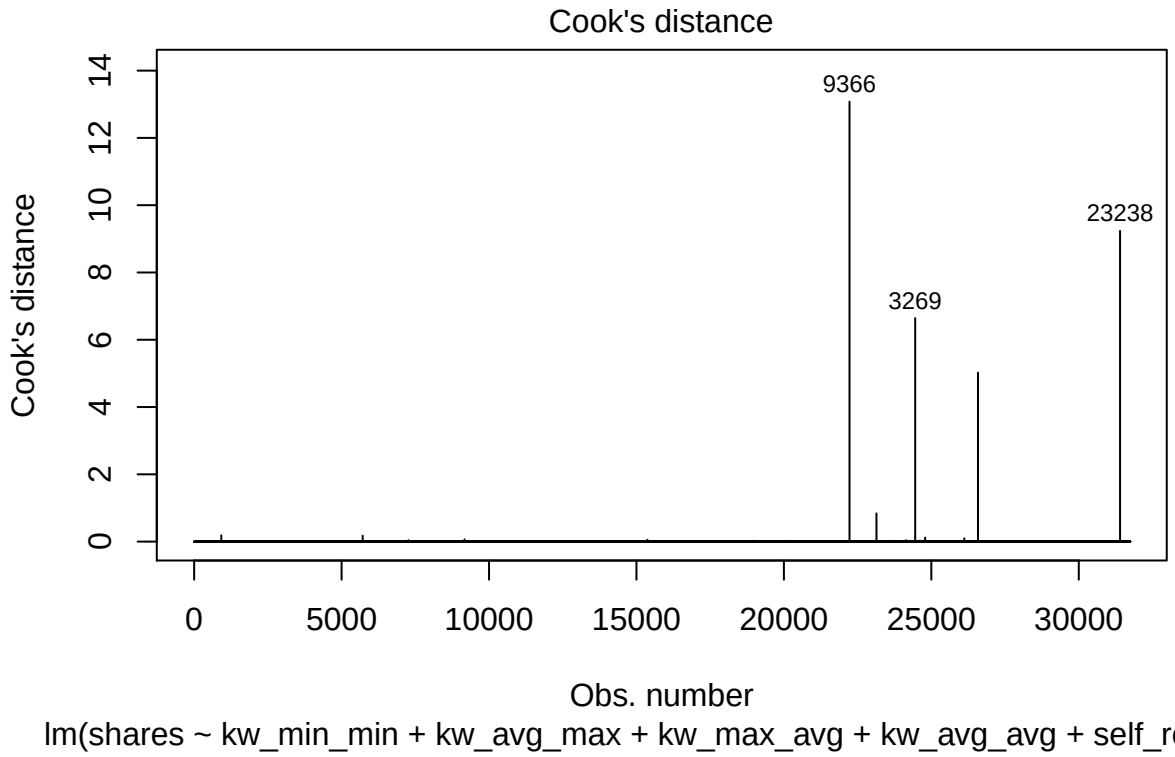


```
plot(fit,which=3)
```



This plot shows whether assumption of constant variance holds or not, you can see that there seem to be some outliers towards the higher side of fitted values, because of that assumption of constant variance is being violated . You can rectify this by finding out which observations are outliers. We will do this by looking at cook's distance plot for all the observation. Wherever cook's distance exceed one, that observation will conidered as an outlier.

```
plot(fit,which=4)
```



You can see observation numbers for which cook's distance is high, repeating the modeling process after removing these outliers is left to you.

Next step is to check performance of our model on the test data. We can prediction for test data using `predict` function.

```
predicted=predict(fit,newdata=md_test)
RMSE=(predicted-md_test$shares)**2 %>%
  mean() %>%
  sqrt()
RMSE
```

```
## [1] 6105.746
```

RMSE can be used for benchmarking of your model. If you build another model , say using different features , or techniques [such as DTrees, RandomForest etc], you can compare RMSEs for them with the one that you got here and select the model for production which has best RMSE.

Note: Remeber that when you are comparing RMSEs for models, train and test data for them should be having same observations.

RMSE as a stand alone value doesnt hold much meaning and there is no set standard, against which you can compare and conclude that its a good/bad RMSE value that you got.

We'll conlude here. You can play around with linear regression model with the other datasets shared with you. Contact if you face any issue.

Prepared By : Lalit Sachan

Contact : lalit.sachan@edvancer.in

In case of any doubts or clarification please take to QA forum on LMS