Super Simple Machine Learning — Multiple Linear Regression Part 1

Bernadette Low Follow
Feb 19, 2018 · 13 min read

In this super long post, I cover Multiple Linear Regression describing briefly(lol) how it works and what criteria you need to take note of. So get yourself water and snacks, cause this will take a while.

The bulk of the basic concepts were covered in my Simple Linear Regression posts, which can be found here. I had intended to quickly cover MLR in one post, but there's really too many things to address. Sigh.

Multiple Linear Regression is Simple Linear Regression, but with more Relationships (More Xs. More Exes. Ha ha.)

It's like when your sister had a baby who was once the sole contributor to all the noise in the house, but then she had a couple more and now all three are contributing to the noise.

$$Y = \beta_0 + \beta_1 X$$

becomes

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3$$

A multiple linear regression with 2 more variables, making that 3 babies in total. Too many babies.

The multiple linear regression explains the relationship between **one continuous dependent variable** (*y*) and **two or more independent variables** (*x*1, *x*2, *x*3… etc)**.**

Note that it says **CONTINUOUS** dependant variable. Since *y* is the sum of *beta*, *beta*1 *x*1, *beta*2 *x*2 *etc etc*, the resulting *y* will be a number, a continuous variable, instead of a "yes", "no" answer (categorical).

For example, with linear regression, I would be trying to find out **how much Decibels of** noise is being produced, and not if it's noisy or not (Noisy | Not).

> *To find categorical variables (e.g "*yes*" or "*no*", "*1*" or "*0*"), logistic regression would be used. I'll cover this next time.*

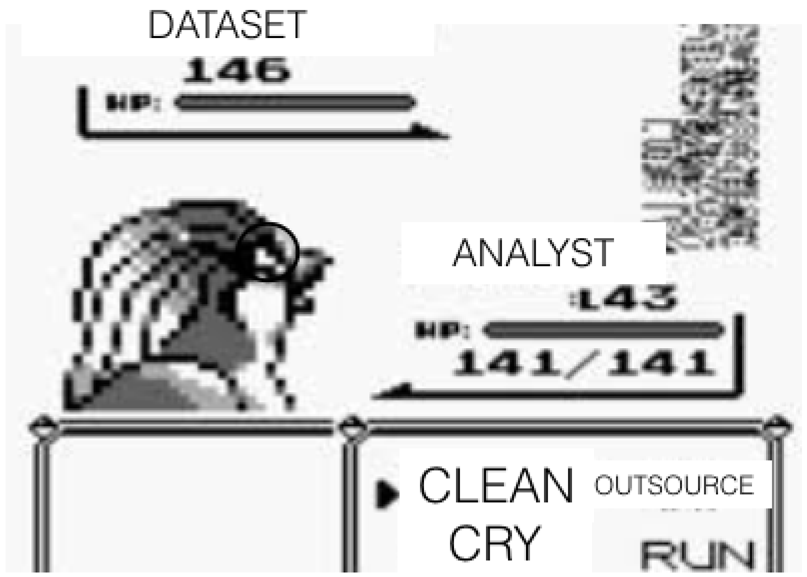
Let's Start with the Data

Datasets to try your code on can be found everywhere.

sklearn has toy datasets for different types of algorithms (regression, classification etc) which are great for practice. Kaggle also has real-life datasets.

Note that in the wild, when you do encounter a dataset, it is going to be ugly AF. It's going to have missing values, erroneous entries, wrongly formatted columns, irrelevant variables… and even after cleaning it, maybe your p-values look terrible and your R squared is too low. You'll need to select good features, try different algorithms, tune your hyperparameters, add a time lag, transform the column's data….

It's not that straightforward in real-life to run a model, and that's why people get paid a lot to do this, so that they can fund their scalp treatment to recover from stress-induced hair loss.

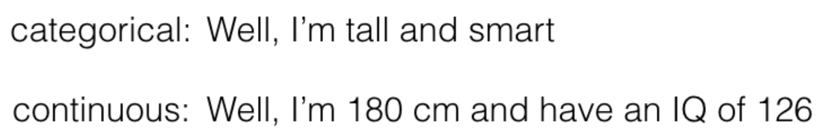


Exploring a dataset is like surfing along the coast of Cinnabar Island… you'll find MissingNo(s). *badum tss*

Categorical Variables >>> Continuous Variables

Due to the nature of the regression equation, **your *x* variables have to be continuous as well**. Thus, you'll need to look into changing your categorical variables into continuous ones.

Continuous variables are simply put, running numbers. Categorical variables are categories.

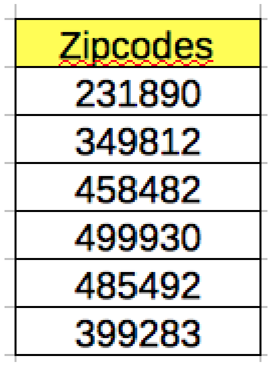


categorical: Well, I'm tall and smart

continuous: Well, I'm 180 cm and have an IQ of 126

It gets slightly confusing when your categorical variables appear continuous at first.

For example, what if there was a column of zip codes or phone numbers?



Zipcodes
231890
349812
458482
499930
485492
399283

Continuous at first glance, but actually categorical.

Every zip code represents a unique address and every phone number is just a unique contact number. Increasing/decreasing this number does not have any meaning—they are merely identifiers with no intrinsic numerical value, and hence, are **considered categorical**.

Categorical Variables are also referred to as **discrete** or **qualitative** variables. There are 3 types:

- **Nominal** : more than 2 types. e.g color
- **Dichotomous**: 2 types e.g yes or no
- **Ordinal**: more than 2 types, but have a rank/order e.g Below average, Average, Above Average

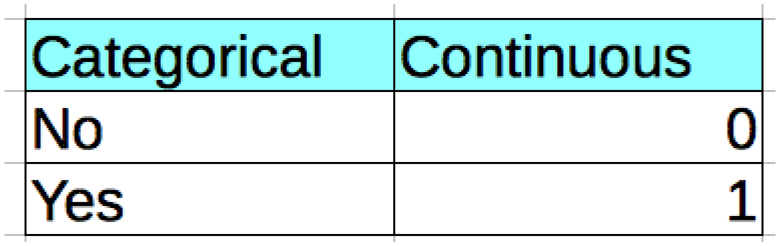
What Do We Do With Them?

There are several ways to change a categorical data into continuous variables that can be used in regression.

Label Encoder:

For Dichotomous Variables

The simple solution for **dichotomous variables** would be to change it into binary—"1" means "yes" and "0" means "no" or vice versa. Your label should start at 0.

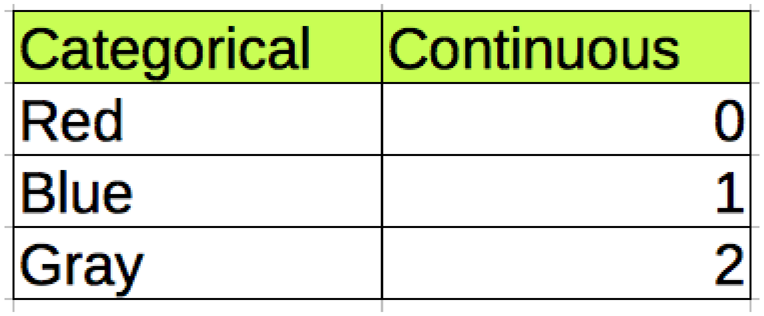


Categorical	Continuous
No	0
Yes	1

Nominal and Ordinal variables are slightly more troublesome.

For Nominal Variables

Let's say you have 3 different colours: Red, Blue and Grey. Following the concept above, you label Red as 0, Blue as 1 and Grey as 2.



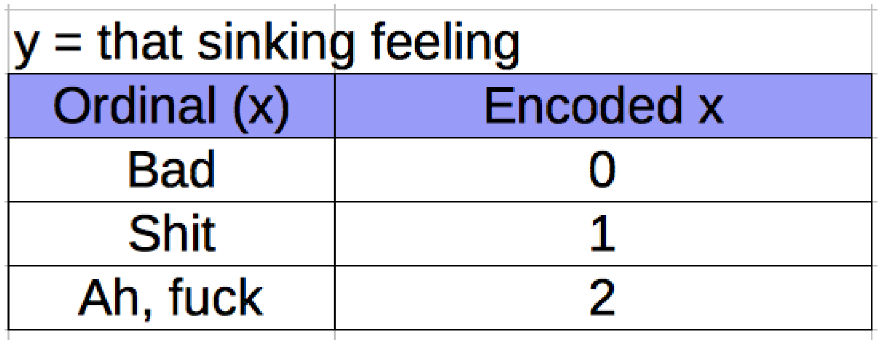
Categorical	Continuous
Red	0
Blue	1
Gray	2

The problem with doing this is that it implies that Grey is of a higher level than Red and Blue and Blue is of a higher level that Red, which if you consider all three just colours with equal "value", none of these colours should be of higher level than the other.

Reckless labelling can lead to disastrous consequences (The Office S3 E1)

For Ordinal Variables

“Ranking” with labels would work better for Ordinal variables, since they do have a rank and should be given different weightage.



y = that sinking feeling

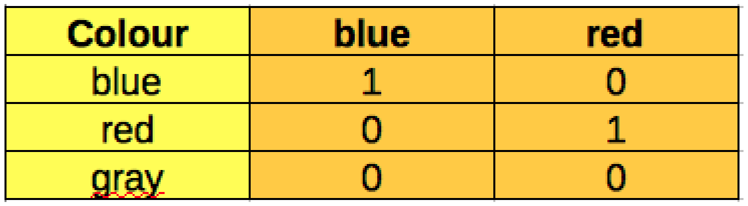
Ordinal (x)	Encoded x
Bad	0
Shit	1
Ah, fuck	2

When facing nominal variables that **should not have different weightage**, one hot encoding is preferred.

One Hot Encoding/Creating dummy variables:

In order not to give categories that are on an even playing field any unequal values, we use one hot encoding. This is done by creating dummy variables, which means creating more “*x*”s. These would be fake/dummy variables because they are placeholders for your actual variable and were created by you yourself.

It’s easy. For every level your variable is, just create a new x for each level.

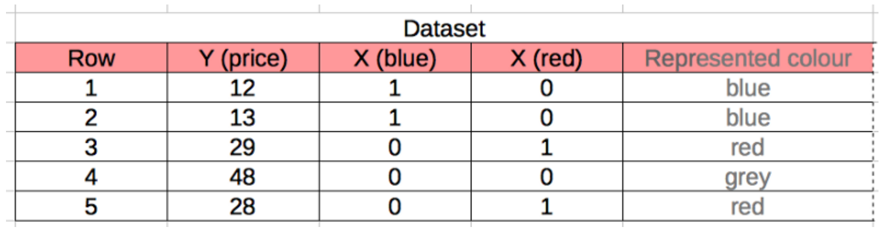


Colour	blue	red
blue	1	0
red	0	1
gray	0	0

easy peasy

Wait… What about Grey?

If your variable can only be 3 colours, then you should only be using 2 dummy variables. Grey becomes the reference category, in the case that your *X*(blue) and *X*(red) are both 0, then by default the variable would be Grey.



Dataset				
Row	Y (price)	X (blue)	X (red)	Represented colour
1	12	1	0	blue
2	13	1	0	blue
3	29	0	1	red
4	48	0	0	grey
5	28	0	1	red

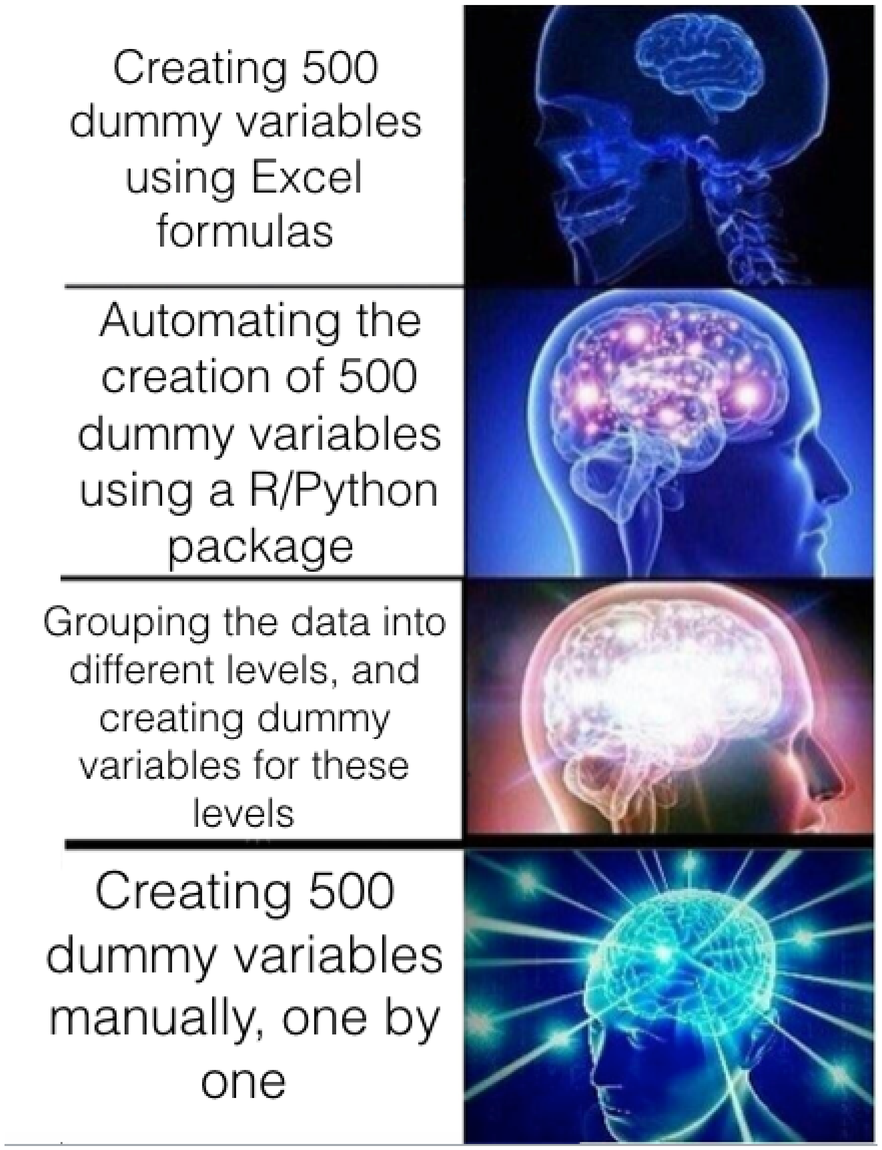
Does it matter which variable you choose to exclude and use as a reference category?

No. But the best practice would be to use the category that happens most often (e.g if 70% of the dataset is grey, then grey would be the reference category).

Combine Levels:

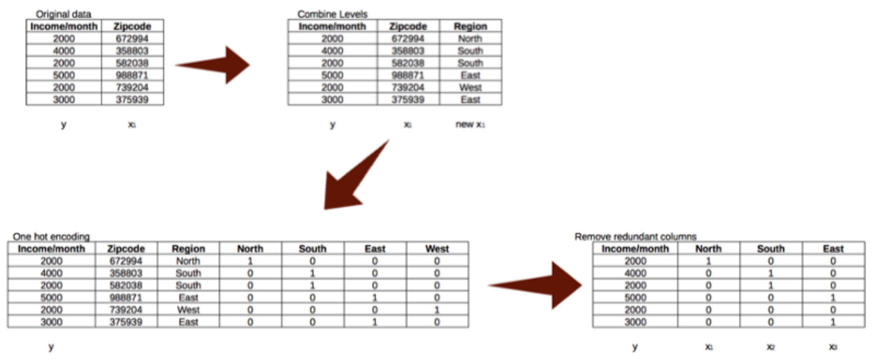
Going back to the column of zip codes, let’s say you have 500 rows of clients, all with their own unique zip codes. Does it make sense to hot encode 500 different zip codes?

That’s going to add 500 columns to your dataset, cluttering your model and blessing you with a migraine.


Creating 500 dummy variables using Excel formulas
Automating the creation of 500 dummy variables using a R/Python package
Grouping the data into different levels, and creating dummy variables for these levels
Creating 500 dummy variables manually, one by one

It is also absolutely pointless since every zip code is unique. It adds no insight to your model because how would you use such information to predict an outcome on new data? If you need to predict the income level of someone living at zip code 323348, how the heck would your model handle that if no such zip code was in your dataset? It has never seen this zip code before and cannot tell you anything about it.

What you *can do* is to **transform** it into something that you could be used to classify future data, such as grouping these zip codes by their **areas** (this data would not be in the dataset but needs to be from domain knowledge or research).



Original data

Income/month	Zipcode
2000	672994
4000	358803
2000	582038
5000	988871
2000	739204
3000	375939
y	x_1

Combine Levels

Income/month	Zipcode	Region
2000	672994	North
4000	358803	South
2000	582038	South
5000	988871	East
2000	739204	West
3000	375939	East
y	x_1	new x_1

One hot encoding

Income/month	Zipcode	Region	North	South	East	West
2000	672994	North	1	0	0	0
4000	358803	South	0	1	0	0
2000	582038	South	0	1	0	0
5000	988871	East	0	0	1	0
2000	739204	West	0	0	0	1
3000	375939	East	0	0	1	0
y						

Remove redundant columns

Income/month	North	South	East
2000	1	0	0
4000	0	1	0
2000	0	1	0
5000	0	0	1
2000	0	0	0
3000	0	0	1
y	x_1	x_2	x_3

what is this, tables for ants?! Apologies for the tiny font.

So instead of **500** different zip codes, you get **4** regions, North, South, East or West (OR depending on how specific you want to get, it could be actual areas like Hougang, Yishun, Bedok, Orchard etc. these are names of areas in Singapore).

That means if new data comes in where you need to predict the outcome, you can predict the *y* based on which area the new zip code falls into.

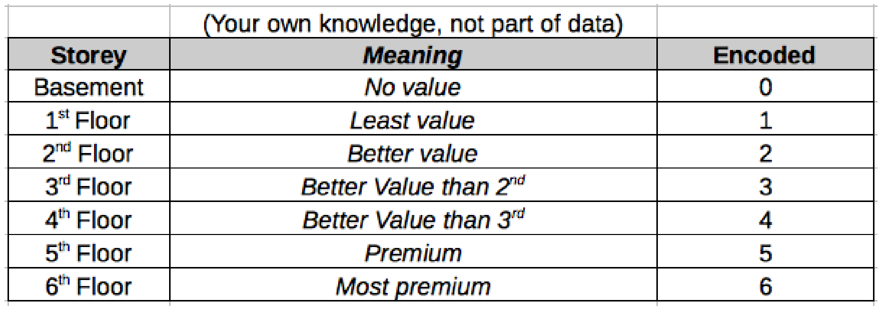
Meaningful labels:

One thing to always keep in mind is not to label blindly.

It has to make sense.

For example, when encoding ordinal variables (categorical variables with rank), you must ensure that the rank value corresponds to the actual significance of each rank (which can also be seen as its relationship with the dependant variable).

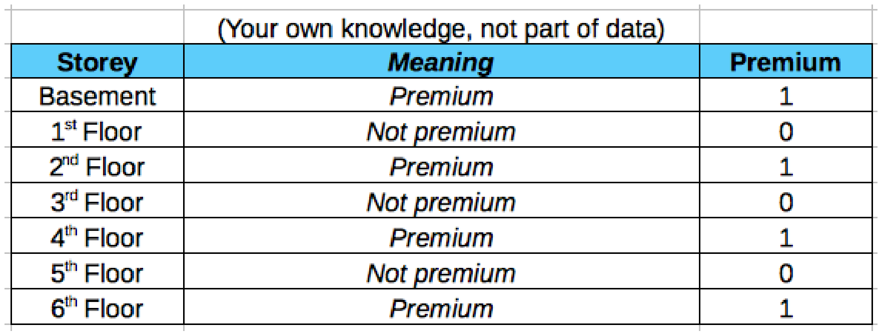
For example, if you are selling a house, and $y = price$, while one of the x variables is the floor the apartment is on, encoding the floors as integers make sense if the floors increase in price as it increases in level:



	(Your own knowledge, not part of data)	
Storey	***Meaning***	**Encoded**
Basement	*No value*	0
1st Floor	*Least value*	1
2nd Floor	*Better value*	2
3rd Floor	*Better Value than 2nd*	3
4th Floor	*Better Value than 3rd*	4
5th Floor	*Premium*	5
6th Floor	*Most premium*	6

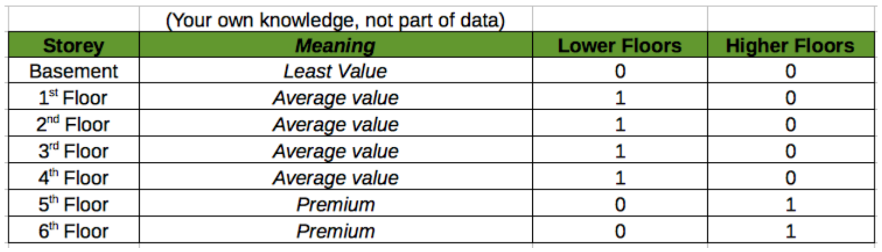
Encoding it this way is appropriate to show its relationship with the Y variable (price)

However, if the value does not increase accordingly, perhaps one hot encoding and combining the levels would be more appropriate:



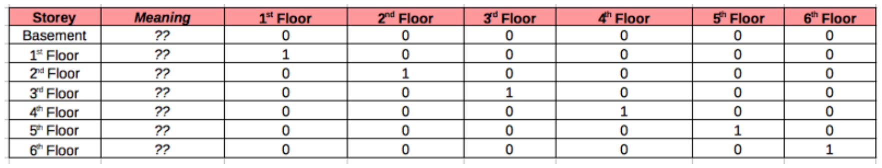
	(Your own knowledge, not part of data)	
Storey	***Meaning***	**Premium**
Basement	*Premium*	1
1st Floor	*Not premium*	0
2nd Floor	*Premium*	1
3rd Floor	*Not premium*	0
4th Floor	*Premium*	1
5th Floor	*Not premium*	0
6th Floor	*Premium*	1

Or



	(Your own knowledge, not part of data)		
Storey	***Meaning***	**Lower Floors**	**Higher Floors**
Basement	*Least Value*	0	0
1st Floor	*Average value*	1	0
2nd Floor	*Average value*	1	0
3rd Floor	*Average value*	1	0
4th Floor	*Average value*	1	0
5th Floor	*Premium*	0	1
6th Floor	*Premium*	0	1

Or if you have no idea what the relationship is:



Storey	***Meaning***	**1st Floor**	**2nd Floor**	**3rd Floor**	**4th Floor**	**5th Floor**	**6th Floor**
Basement	*??*	0	0	0	0	0	0
1st Floor	*??*	1	0	0	0	0	0
2nd Floor	*??*	0	1	0	0	0	0
3rd Floor	*??*	0	0	1	0	0	0
4th Floor	*??*	0	0	0	1	0	0
5th Floor	*??*	0	0	0	0	1	0
6th Floor	*??*	0	0	0	0	0	1

One hot encode everything!!!! There are packages that can do this for you, don’t worry.

Always ensure that no matter how you transform your categorical variables, make sure you look back at it and ask yourself “does that make sense?”

Sometimes there is no right answer, so the question then becomes "does that make **more** sense?"

which might potentially become, "but what makes any sense?" and lead into "what is sense?" which becomes "what is?" and then segue into a week spent in existential nihilism.

Bottom line is: *Know Thy Data*

Feature Selection

Having too many variables could potentially cause your model to become less accurate, especially if certain variables have no effect on the outcome or have a significant effect on other variables.

Variables that have no significant effect or high collinearity can ruin the model

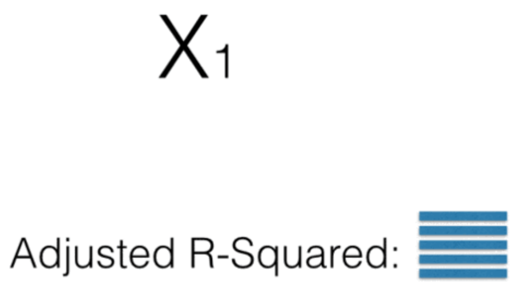
Let’s take for example the 3 babies screaming. If I were to model my regression equation to find out the decibels of noise being produced based on these 4 variables = baby 1, baby 2, baby 3, lamp (on or off) , it would not be a good model. This is because my spidey senses are telling me that a lamp should not contribute to noise, so any possible correlation between lamp and noise would be spurious and inaccurate.

> *Basic step of Feature Selection: Use your Common and/or Business Sense(s)*

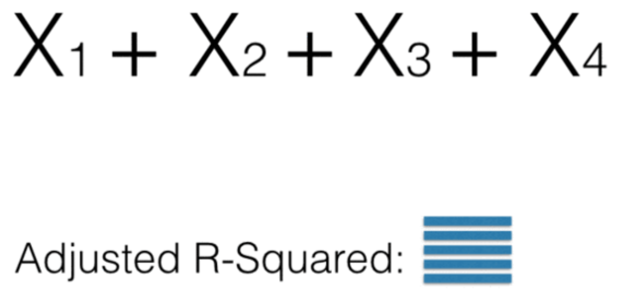
One other way to select features is to use the p-values. As we last discussed, p-values tell you how statistically significant the variable is. **Removing** variables with **high p-values** can cause your accuracy/R squared to increase, and even the p-values of the other variables to increase as well—and that’s a good sign.

This action of omitting variables is part of stepwise regression. There are 3 ways to do this:

- **Forward Selection**: Start with 0. Run the model over and over, trying each variable. Find the variable that gives the best metric (E.g p-values, Adjusted R-squared, SSE, % accuracy) and stick with it. Run model again with the chosen variable, trying one of the remaining variables at each time and sticking with the best one. Repeat process until the adding does not improve the model any more.



- **Backward Elimination**: Start with all variables. Try model out multiple times, excluding one variable at each time. Remove variable that causes the model to improve the most when it is left out. Repeat process without the removed variable, until the metric(s) you are judging on can no longer improve.



- **Bidirectional Elimination**: Do a forward selection, but then do backward eliminations as well at some stages. So you can be at a stage where you've added X1, X2, X5 and X8, then do elimination by taking out X2. This is a good example.

There are R and Python packages created by amazing people that automate these processes to choose the "best model" based on a particular metric (e.g adjusted R Squared or AIC or p-values).Here are some I found.

NOTE: Stepwise regression is a quick and fast way to get better scoring models, especially when you're running simple models. It's widely used, but also **widely criticised** as being inaccurate. I've always been taught to use stepwise, so I would love to hear your opinions about it.

An alternative to stepwise regression would be the LASSO (Least Absolute Shrinkage and Selection Operator) method, which I will cover next time. Or you can read about it here.

Correlation and Collinearity

Checking for collinearity helps you get rid of variables that are skewing your data by having a **significant relationship** with another variable.

Correlation between variables describe the **relationship** between two variables. If they are **extremely correlated**, then they are **collinear**.

Autocorrelation occurs when a variable's data affects another instance of that same variable (same column, different row). Linear regression only works if there is little or no autocorrelation in the dataset, and each instance is independent of each other. If instances are autocorrelated then your residuals are not independent from each other, and will show a pattern. This usually occurs in time series datasets, so I will go into more details when I cover Time Series Regression.

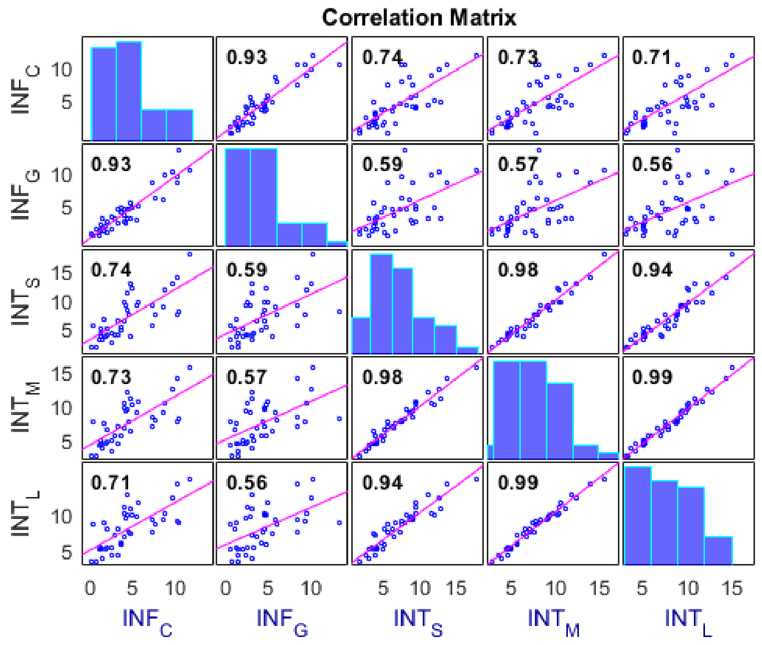
In stock market data, based on prices at a certain time, you can roughly guess what the prices will be in the future following that, showing the dependancy of the future instance on the previous one.

Multicollinearity exists when two or more of the predictors (*x variables*) in a regression model are moderately or highly correlated (different column). When one of **our predictors is able to strongly predict another predictor or have weird relationships with each other** (maybe $x2 = x3$ or $x2 = 2(x3) + x4$), then your regression equation is going to be a mess.

Multicollinearity: One x variable = Orange colour, Other x variable = number of oranges. They would be highly correlated as the number of oranges affects the orange-ness of the juice. Orange you pleased with this simple analogy?

Why is multicollinearity an issue with regression? Well, the regression equation is the **best fit line to represent the effects of your predictors and the dependant variable**, and **does not include the effects of one predictor on another**.

Having high collinearity (correlation of 1.00) between predictors will affect your coefficients and the accuracy, plus its ability to reduce the SSE (sum of squared errors—that thing you need to minimise with your regression).



Example of a correlation plot taken from https://www.mathworks.com/help/econ/corrplot.html comparing 5 variables with each other

The simplest method to detect collinearity would be to plot it out in graphs or to view a correlation matrix to check out pairwise correlation (correlation between 2 variables).

If you have two variables that are highly correlated, your best course of action is to just remove one of them.

Top 4 Signs that Collinearity is Affecting Your Regression Life —You'll be Shocked by #3!

1. When your coefficient (the *b* in *bx*) is *negative*, but you know from common sense and business knowledge that the effect of the variable should be *positive.* Or when the coefficient is *too small* for a variable that should have a *bigger effect*.
2. When you run one model with that *x* variable and you run another without that *x* variable, and the coefficients for these models are *drastically* different.
3. When the *t*-tests for each of the individual slopes are non-significant, but the overall *F*-test is significant. This is because multicollinearity causes some variables to seem useless, so lowering the t-stat, but has no effect on the F-statistics which takes an overall view.
4. When your VIF is off the charts is 5 or more

What in tarnations is VIF?!

VIF : Variance Inflation Factor

VIF is a measure of how much the **variance of the coefficient derived from the model is inflated by collinearity**. It helps detect multicollinearity that you cannot catch just by eyeballing a pairwise correlation plot and even detects strong relations between 3 variables and more.

It is calculated by taking the **ratio of [the variance of all of the coefficients] divided by [the variance of that one variable's coefficient** when it is the only variable in the model].

VIF = 1 : no correlation between that predictor and the other variables

VIF = 4 : Suspicious, needs to be looked into

VIF = 5-10 : "Houston, we have a problem." Look into it or drop the variable.

```
Finding VIF in Python:

from statsmodels.stats.outliers_influence import
variance_inflation_factor

y, X = dmatrices('y ~ x1 + x2', dataset,
return_type='dataframe')

vif_df = pd.Dataframe()
vif_df["vif"] = [variance_inflation_factor(X.values, i) for
i in range(X.shape[1])]

vif_df["features"] = X.columns
print(vif_df)

Finding VIF in R

 reg <- lm(y ~ x1+x2,x3, data=dataset)
 summary(reg)

 VIF(lm(x1 ~ x2+x3, data=dataset))
 VIF(lm(x2 ~ x1+x3, data=dataset))
 VIF(lm(x3 ~ x2+x1, data=dataset))
```

Adjusted R Squared

We learnt about R Squared in the last post. To recap, it measures h**ow well the regression line fits with the actual results**. Let’s dive deeper.

Its formula is:

R-squared = Explained variation / Total variation

Explained Variance = Sum of Squares due to Regression (SSR) [do no confuse with RSS (Residual Sum of Squares) which is also called SSE]

Sum of Squares Total (SST) = SSR + SSE (Sum of Squared Errors)

You can look at it as SSE being the “*Unexplained Variation*”. Remember that SSE is the errors between your actual (*y*) and your predicted (*y-hat*)

Total Variation = SST (Sum of Squares Total). The average squared difference between Each Variable and the Overall Mean of the Variable (y- ybar).

Since SST = SSE + SSR
SSR = SST - SSE

Explained Variance = SSR
Explained Variance = [SST - SSE]

R-squared = Explained Variance/Total Variance
= [SST - SSE] / SST
=1 - [SSE/SST]

sos.

So R-squared essentially looks like this:

$$r^2 = 1 - \frac{\text{SS Error}}{\text{SS Total}} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

http://blog.minitab.com/blog/statistics-and-quality-data-analysis/r-squared-sometimes-a-square-is-just-a-square

Now that that's clarified, let's check out Adjusted R-Squared.

Why adjust it?

Because your numerator is the **SUM of variances**, adding more predictors (*x* variables) will **always increase** the R-squared, making it inaccurate as the predictors increase.

The **adjusted R-squared** on the other hand, **accounts for the increase in number of predictors.**

$$R^2 = 1 - \frac{SS_{residuals}}{SS_{total}}$$

$$\text{Adjusted R}^2 = 1 - \frac{SS_{residuals} / (n - K)}{SS_{total} / (n - 1)}$$

https://www.graphpad.com/guides/prism/7/curve-fitting/index.htm?reg_interpreting_the_adjusted_r2.htm

- *n* refers to the number of data points (E.g rows in your dataset)
- *k* refers to the number of x variables

Because of the nature of the equation, the Adjusted R-Squared should always be lower than or equal the R-Squared.

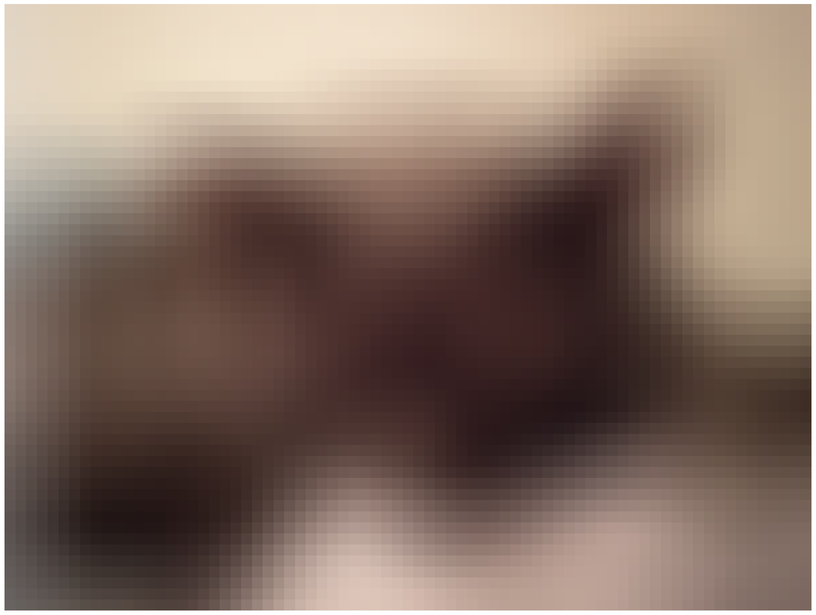
How to evaluate what's good and what's not?

A good way to use Adjusted R square is to perhaps run a few iterations of the model with different variables (e.g do a stepwise regression). You can see how the adjusted R squared increases and maybe hits an optimal point before decreasing when more variables are added. You should then keep it at those variables that gave you the optimal adjust R squared.

Any variables that are able to predict the outcome variable significantly, and thus improve your accuracy, will lead to a higher Adjusted R-Squared.

That's all folks! For now.

Take a breather. Congrats on reaching the end of this post!



Take this cute photo - of a sleeping cat that I pet-sit—as your reward.

The next one will cover the actual modelling using gretl and python, and more ways of checking accuracy.

Till next time!