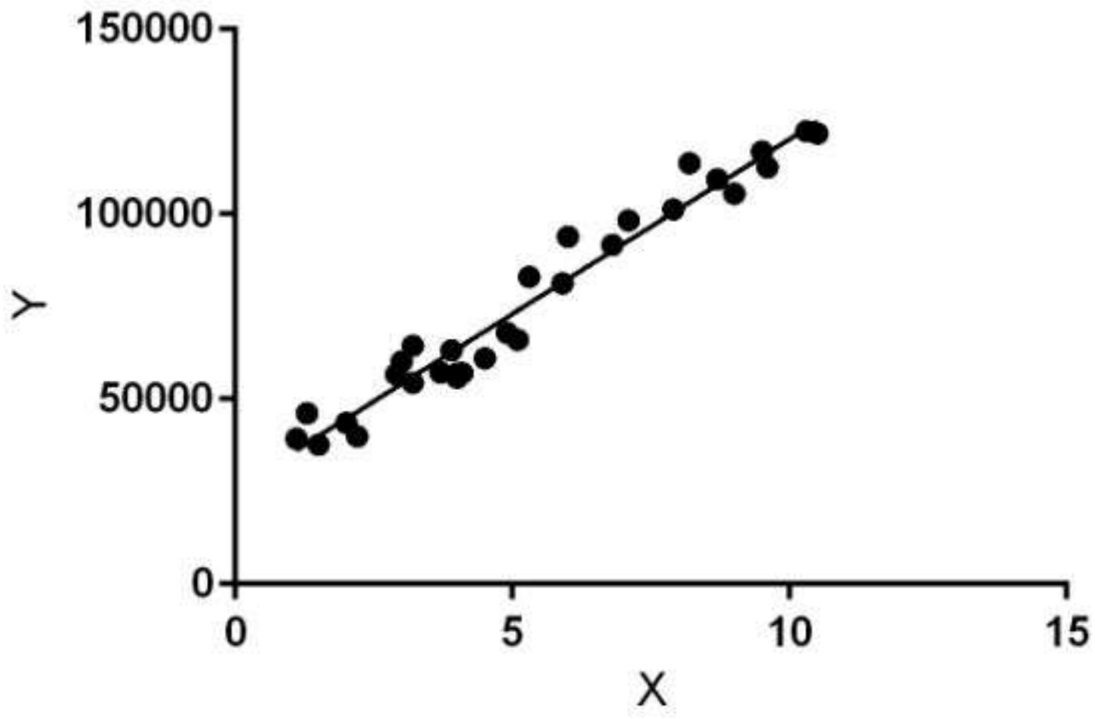
1. Explain the linear regression algorithm in detail.

Ans:

Linear regression is perhaps one of the most well known and well understood algorithms in statistics and machine learning.

Linear Regression is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

Hypothesis function for Linear Regression :

$$y = \theta_1 + \theta_2 . x$$

While training the model we are given :
x: input training data (univariate – one input variable(parameter))
y: labels to data (supervised learning)
When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ_1 and θ_2 values.
θ_1: intercept
θ_2: coefficient of x
Once we find the best θ_1 and θ_2 values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.
How to update θ_1 and θ_2 values to get the best fit line ?
Cost Function (J):
By achieving the best-fit regression line, the model aims to predict y value such that the error difference between predicted value and true value is minimum. So, it is very important to update the θ_1 and θ_2 values, to reach the best value that minimize the error between predicted y value (pred) and true y value (y).

$$minimize\frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

$$J = \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

Cost function(J) of Linear Regression is the **Root Mean Squared Error (RMSE)** between predicted y value (pred) and true y value (y).
Gradient Descent:
To update θ_1 and θ_2 values in order to reduce Cost function (minimizing RMSE value) and achieving the best fit line the model uses Gradient Descent. The idea is to start with random θ_1 and θ_2 values and then iteratively updating the values, reaching minimum cost.

Multiple Linear Regression is Simple Linear Regression, but with more Relationships

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 *x2 etc etc*, the resulting *y* will be a number, a continuous variable, instead of a “yes”, “no” answer (categorical).

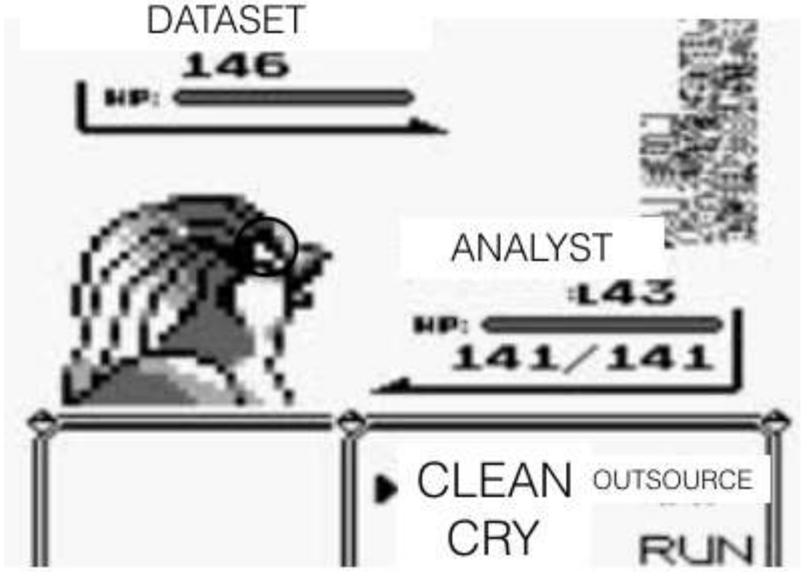
For example, with linear regression, I would be trying to find out **how much Decibels of** noise is being produced, and not if it’s noisy or not (Noisy | Not).

*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

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.

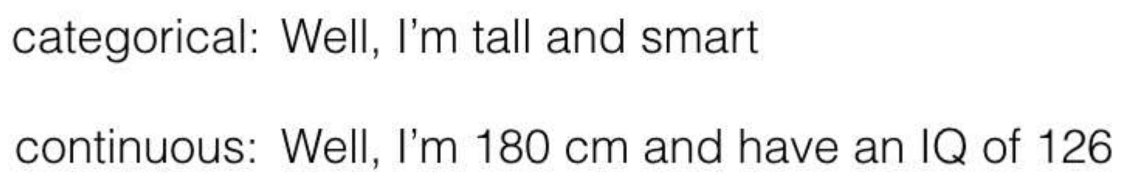


Exploring a dataset is like surfing along the coast of Cinnabar Island… you'll find MissingNo(s). *badum tss*

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.

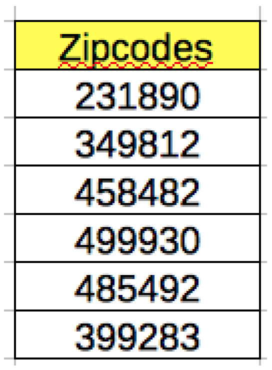


categorical: Well, I'm tall and smart

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

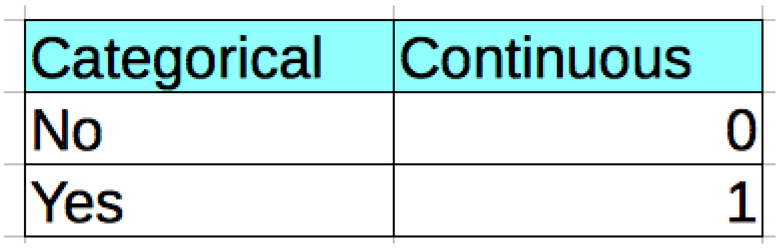
What Do We Do With Them?

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.

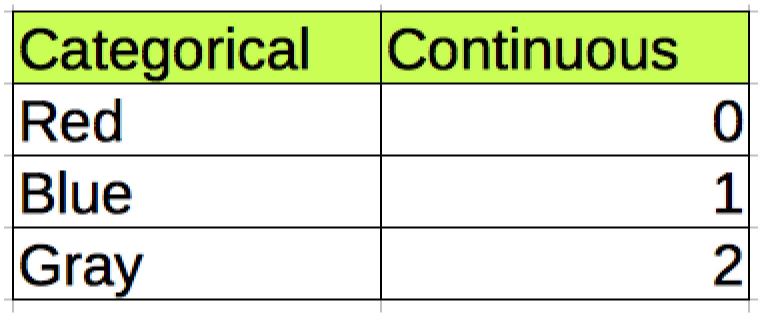


Categorical	Continuous
No	0
Yes	1

. **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.



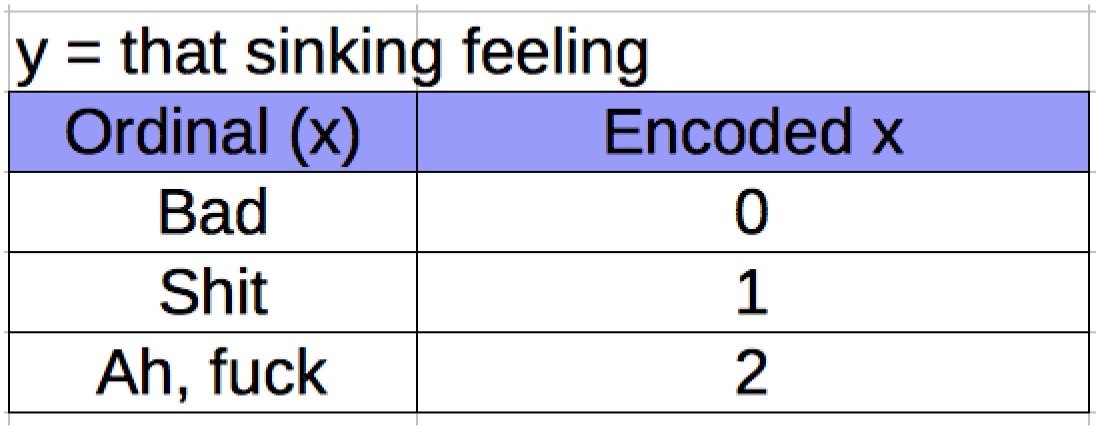
Categorical	Continuous
Red	0
Blue	1
Gray	2

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

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

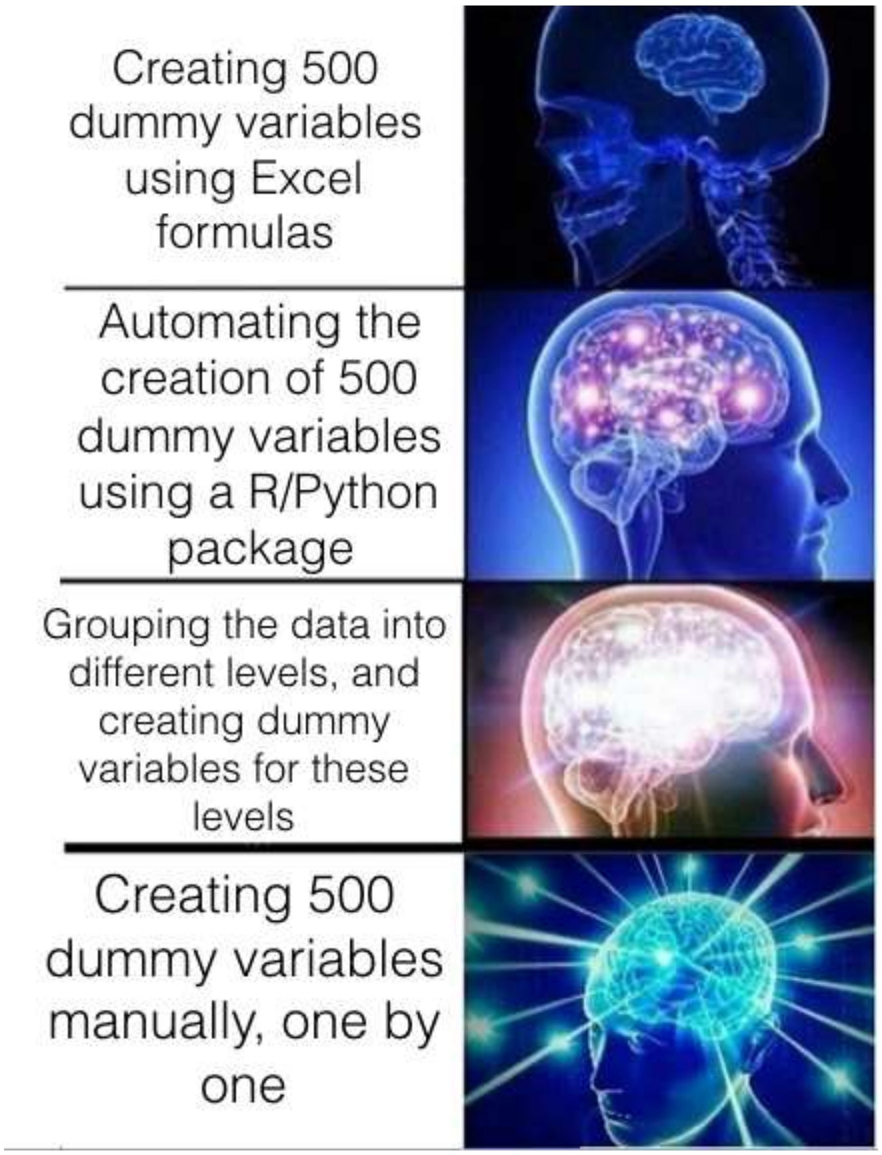
Does it matter which variable you choose to exclude and use as a reference category?

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:

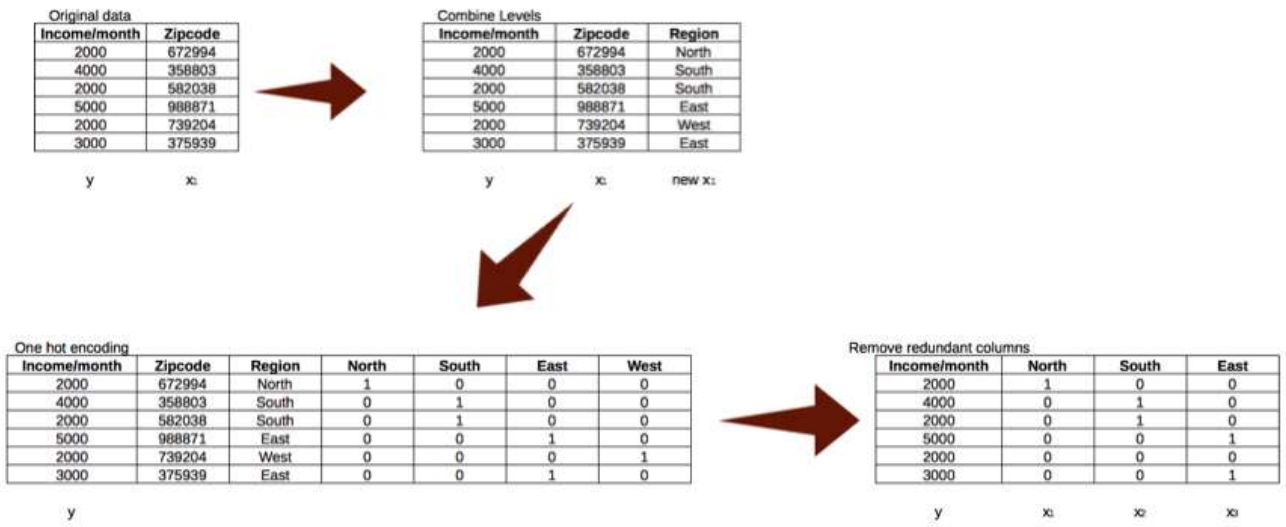
Going back to the column of zip codes, let's say you have 500 rows of clients, all with their own unique zip codes. Does it make sense to hot encode 500 different zip codes?

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

Combine Levels

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

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

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

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:

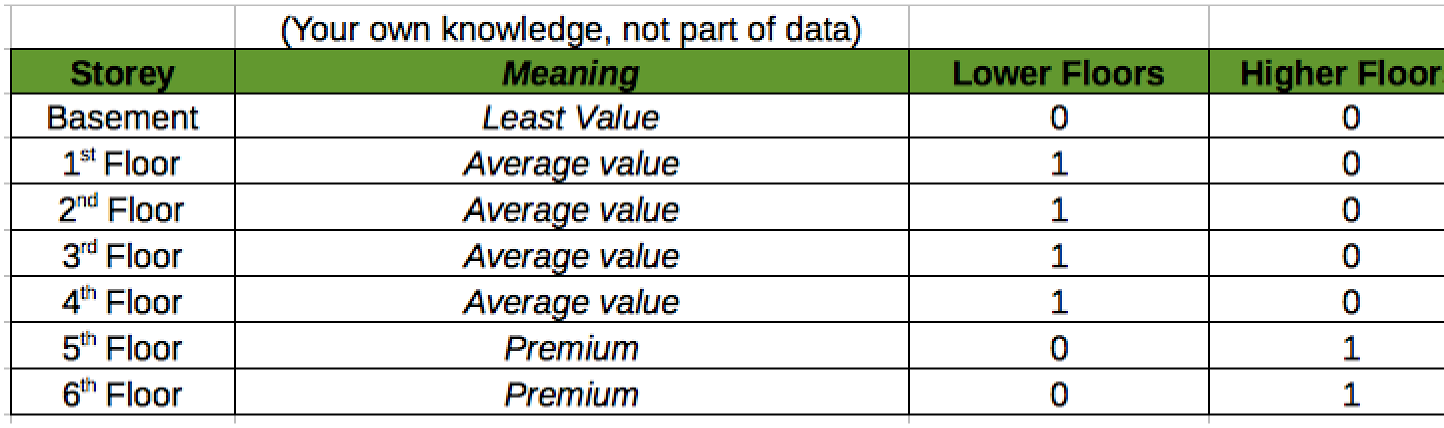
	(Your own knowledge, not part of data)	
Storey	***Meaning***	**Encoded**
Basement	*No value*	0
1st Floor	*Least value*	1
2nd Floor	*Better value*	2
3rd Floor	*Better Value than 2nd*	3
4th Floor	*Better Value than 3rd*	4
5th Floor	*Premium*	5
6th Floor	*Most premium*	6

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:

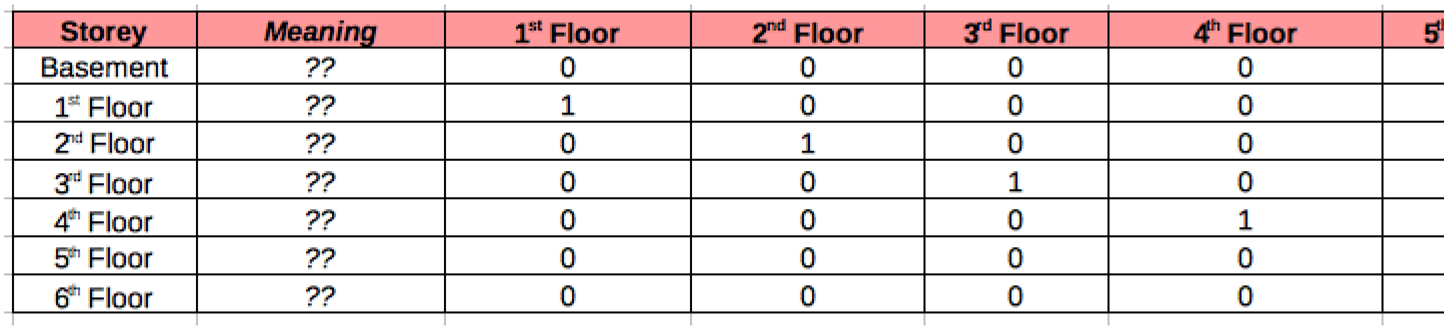
	(Your own knowledge, not part of data)	
Storey	***Meaning***	**Premium**
Basement	*Premium*	1
1st Floor	*Not premium*	0
2nd Floor	*Premium*	1
3rd Floor	*Not premium*	0
4th Floor	*Premium*	1
5th Floor	*Not premium*	0
6th Floor	*Premium*	1

Or



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

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 The 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

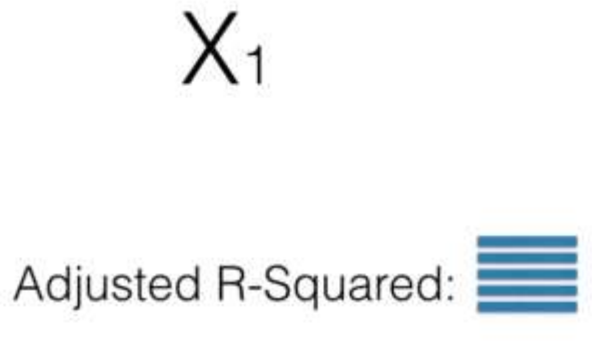
Let’s take for example the 3 babies screaming. If I were to model my regression equation to find out the decibels of noise being produced based on these 4 variables = baby 1, baby 2, baby 3, lamp (on or off) , it would not be a good model. This is because my spidey senses are telling me that a lamp should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

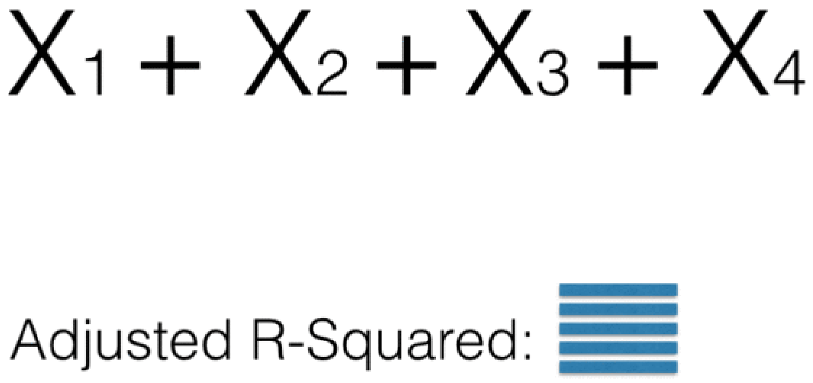
One other way to select features is to use the p-values. As we last discussed, p-values tell you how statistically significant the variable is. **Removing** variables with **high p-values** can cause your accuracy/R squared to increase, and even the p-values of the other variables to increase as well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

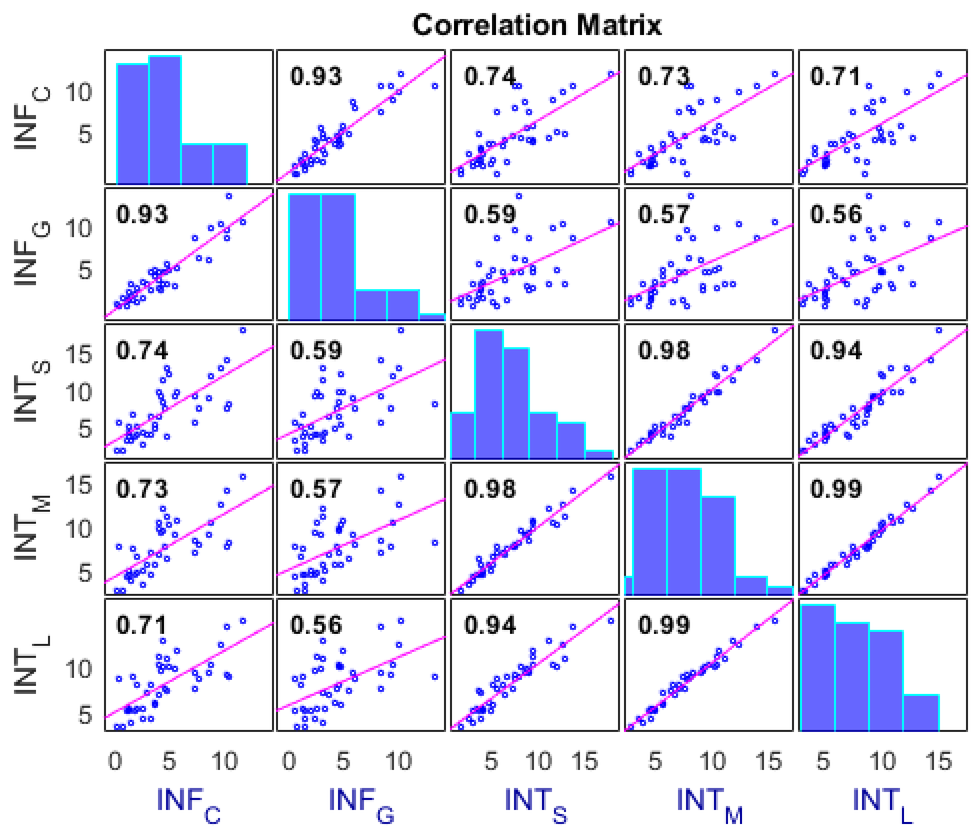
In stock market data, based on prices at a certain time, you can roughly guess what the prices will be in the future following that, showing the dependancy of the future instance on the previous one.

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_factory, 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).

$$\text{Since SST} = \text{SSE} + \text{SSR}$$
$$\text{SSR} = \text{SST} - \text{SSE}$$

$$\text{Explained Variance} = \text{SSR}$$
$$\text{Explained Variance} = [\text{SST} - \text{SSE}]$$

$$\text{R-squared} = \text{Explained Variance/Total Variance}$$
$$= [\text{SST} - \text{SSE}] \,/\, \text{SST}$$
$$= 1 - [\text{SSE/SST}]$$

SOS.

So R-squared essentially looks like this:

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

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.**

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

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

- *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.

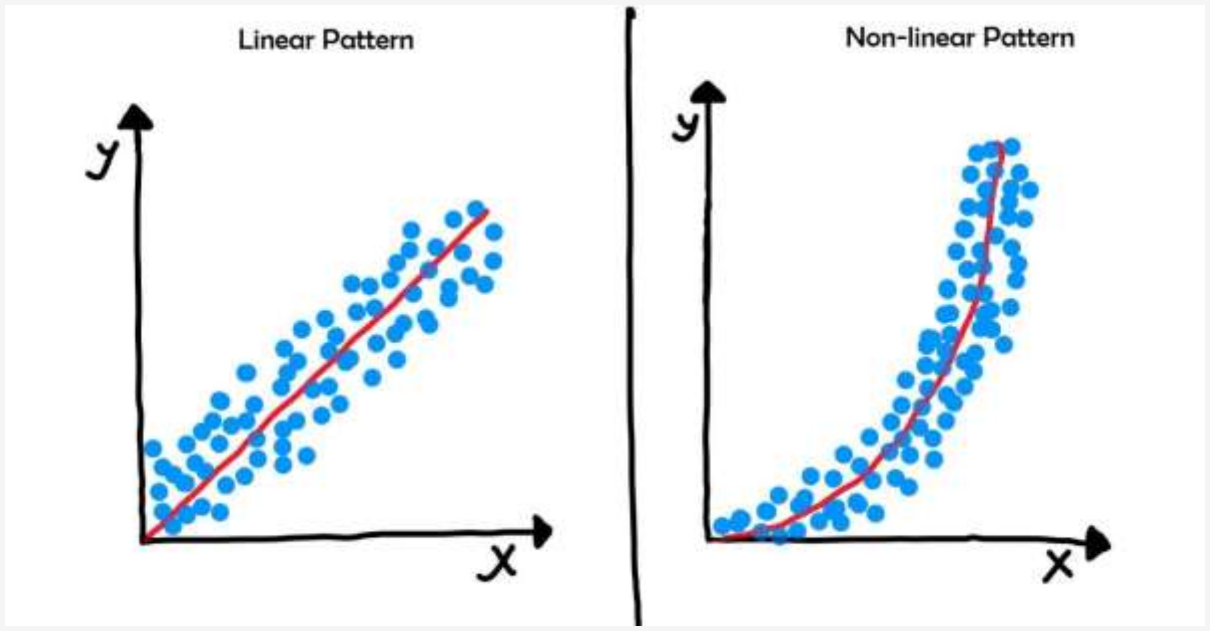
2. What are the assumptions of linear regression regarding residuals?

Ans:

In regression analysis, **residual** (e) is nothing but error term of each data point which is the difference between the observed value of the dependent variable (y) and the predicted value (ŷ) is called the Each data point has one **residual**. **Residual** = Observed value - Predicted value. e = y - ŷ Both the sum and the **mean** of the residuals are equal to zero.

There is a *linear relationship* between X and Y:

- X and Y should display some sort of a linear relationship; otherwise, there is no use of fitting a linear model between them.

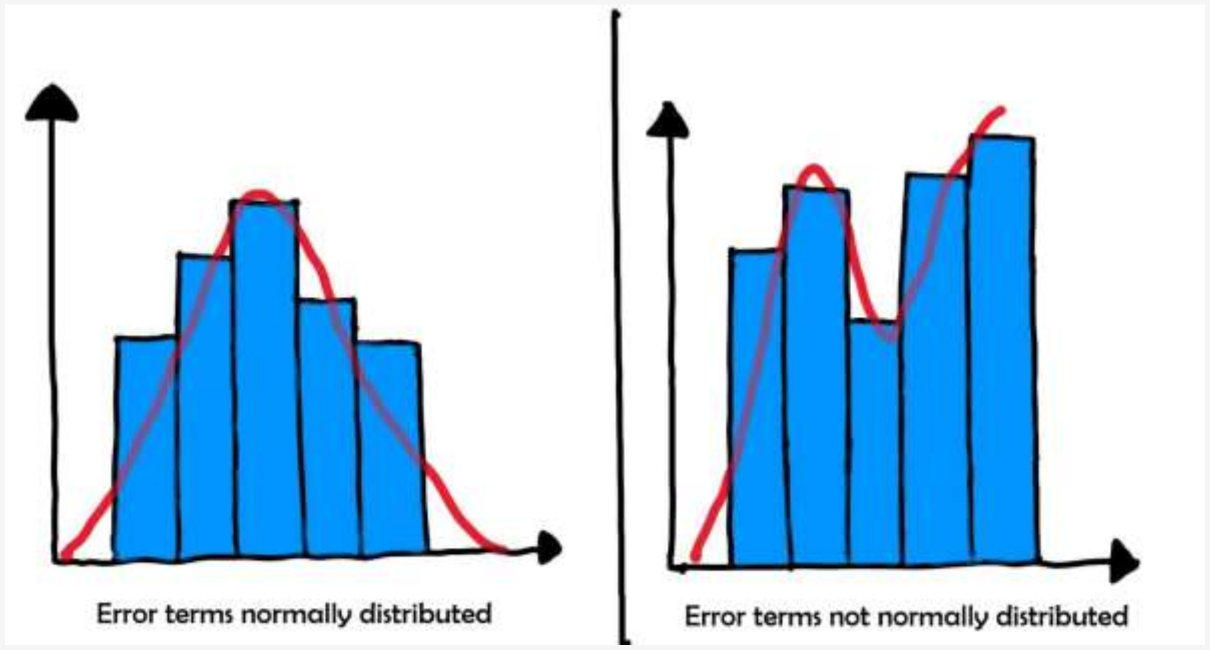


Error terms are *normally distributed* with mean zero(not X, Y):

- There is no problem if the error terms are not normally distributed if you just wish to fit a line and not make any further interpretations.
- But if you are willing to make some inferences on the model that you have built (you will see this in the coming segments), you need to have a notion of the distribution of the error terms. One particular repercussion of the error terms not being normally

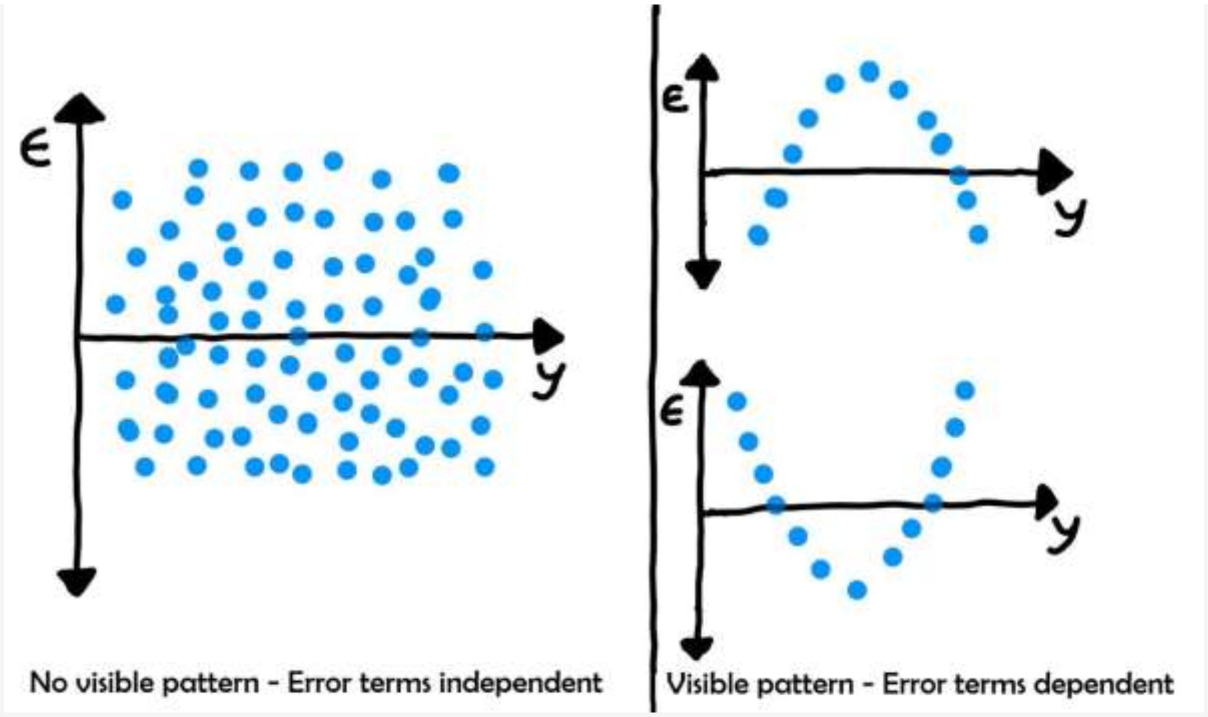
distributed is that the p-values obtained during the hypothesis test to determine the significance of the coefficients become unreliable. (You'll see this in a later segment.)

- The assumption of normality is made, as it has been observed that the error terms generally follow a **normal distribution with mean equal to zero** in most cases.



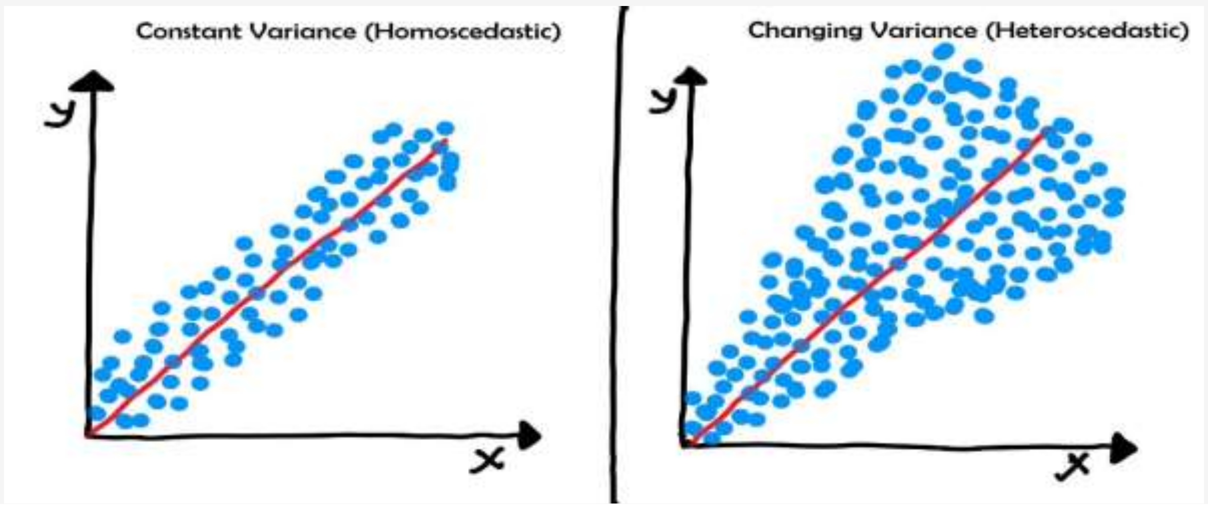
Error terms are *independent* of each other:

- The error terms should not be dependent on one another (like in a time-series data wherein the next value is dependent on the previous one).



Error terms have *constant variance* (homoscedasticity):

- The variance should not increase (or decrease) as the error values change.
- Also, the variance should not follow any pattern as the error terms change.



3. What is the coefficient of correlation and the coefficient of determination?

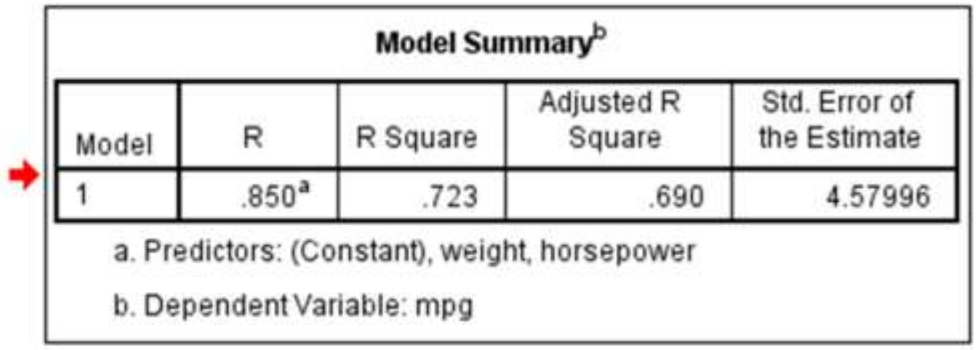
Ans:

Coefficient of Correlation: is the degree of relationship between two variables say x and y. It can go between -1 and 1. 1 indicates that the two variables are moving in unison. They rise and fall together and have perfect correlation. -1 means that the two variables are in perfect opposites. One goes up and other goes down, in perfect negative way. Any two variables in this universe can be argued to have a correlation value. If they are not correlated then the correlation value can still be computed which would be 0. The correlation value always lies between -1 and 1 (going thru 0 – which means no correlation at all – perfectly not related). Correlation can be rightfully explalined for simple linear regression – because you only have one x and one y variable. For multiple linear regression R is computed, but then it is difficult to explain because we have multiple variables invovled here. Thats why R square is a better term. You can explain R square for both simple linear regressions and also for multiple linear regressions.

Coefficient of correlation is "R" value which is given in the summary table in the Regression output. **R square is also called coefficient of determination.** Multiply R times R to get the R square value. In other words **Coefficient of Determination is the square of Coefficeint of Correlation.**

R square or coeff. of determination shows percentage variation in y which is explained by all the x variables together. Higher the better. It is always between 0 and 1. It can never be negative – since it is a squared value. the **coefficient of determination**, denoted R^2 or r^2 and pronounced "R squared", is the proportion of the variance in the dependent variable that is predictable from the independent variable(s).

It is easy to explain the R square in terms of regression. It is not so easy to explain the R in terms of regression.



Model Summary[b]

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.850[a]	.723	.690	4.57996

a. Predictors: (Constant), weight, horsepower

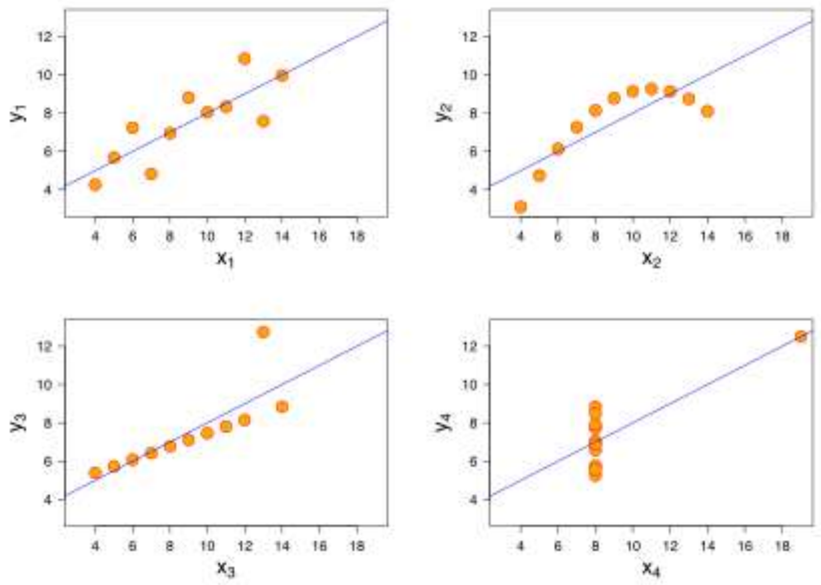
b. Dependent Variable: mpg

Coefficient of Correlation is the R value i.e. .850 (or 85%). Coefficient of Determination is the R square value i.e. .723 (or 72.3%). R square is simply square of R i.e. R times R.

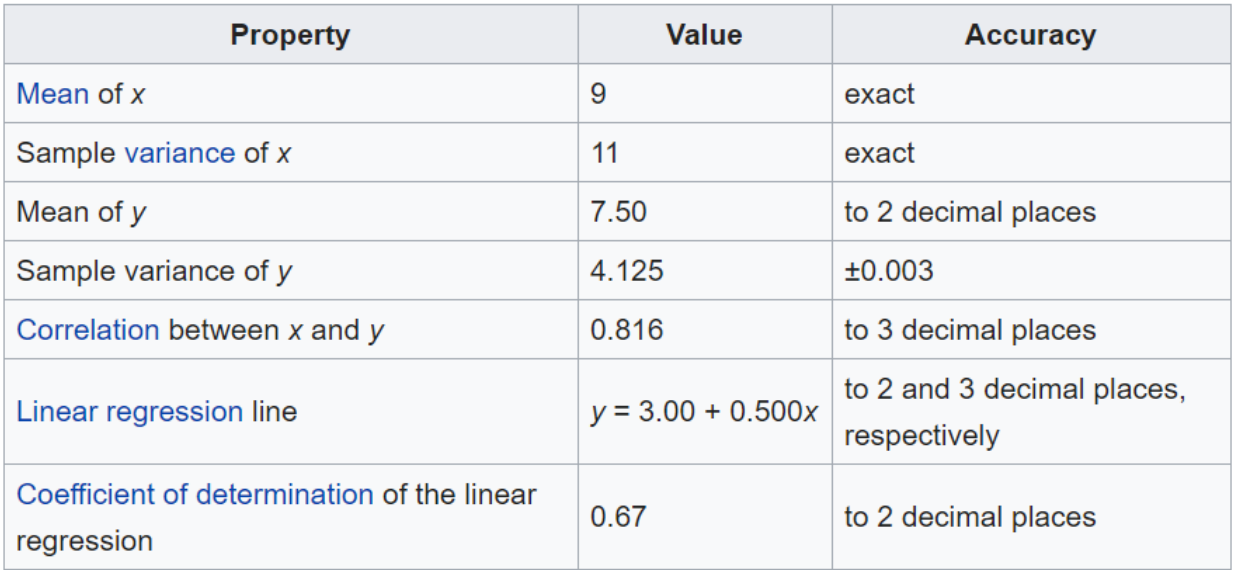
4. Explain the Anscombe's quartet in detail.

Ans:

Anscombe's quartet comprises four data sets that have nearly identical simple descriptive statistics, yet have very different distributions and appear very different when graphed. Each dataset consists of eleven (x,y) points. They were constructed in 1973 by the statistician Francis Anscombe to demonstrate both the importance of graphing data before analyzing it and the effect of outliers and other influential observations on statistical properties. He described the article as being intended to counter the impression among statisticians that "numerical calculations are exact, but graphs are rough."



All four sets are identical when examined using simple summary statistics, but vary considerably when graphed

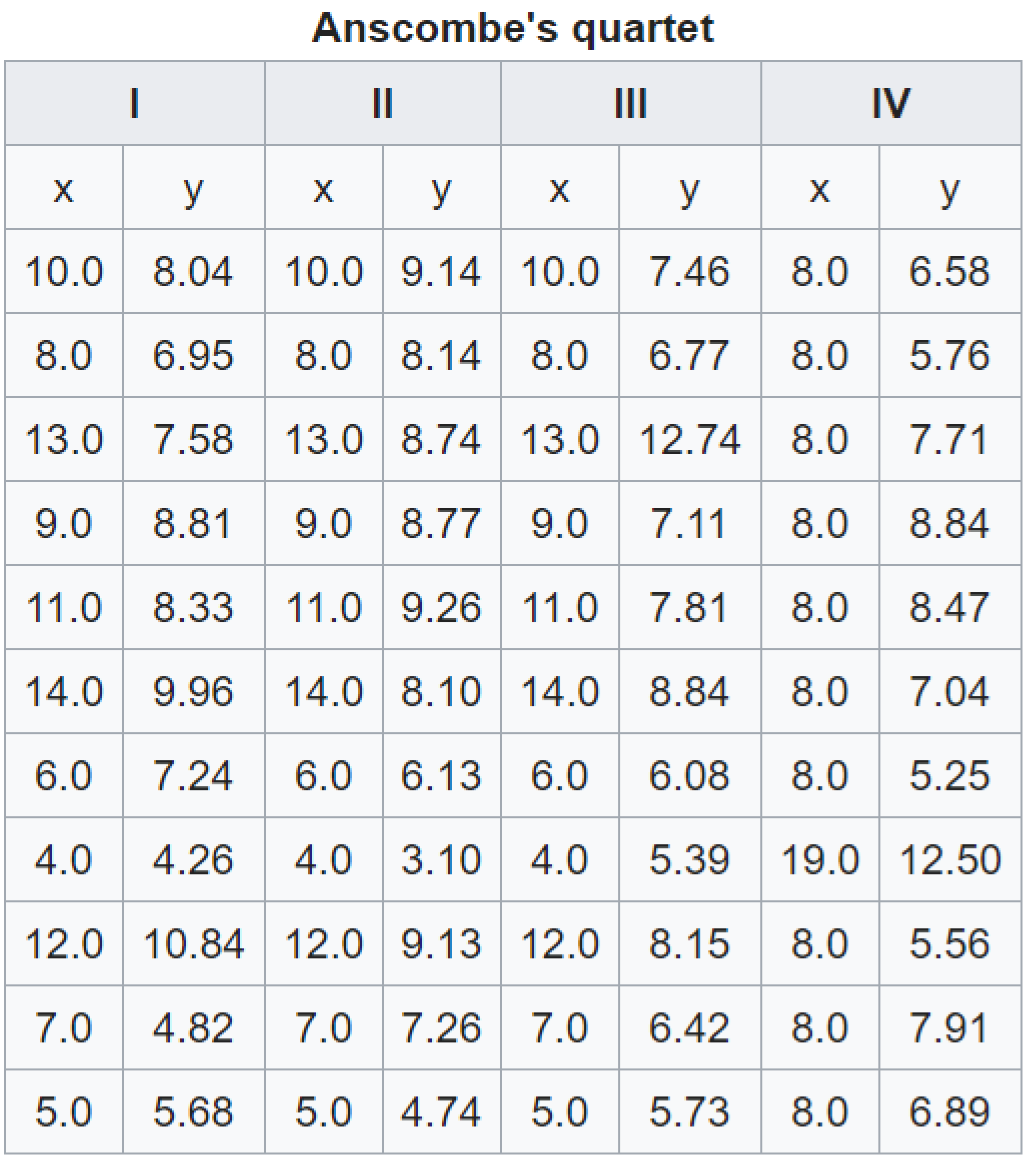


Property	Value	Accuracy
Mean of x	9	exact
Sample variance of x	11	exact
Mean of y	7.50	to 2 decimal places
Sample variance of y	4.125	±0.003
Correlation between x and y	0.816	to 3 decimal places
Linear regression line	$y = 3.00 + 0.500x$	to 2 and 3 decimal places, respectively
Coefficient of determination of the linear regression	0.67	to 2 decimal places

- The first scatter plot (top left) appears to be a simple linear relationship, corresponding to two variables correlated and following the assumption of normality.
- The second graph (top right) is not distributed normally; while a relationship between the two variables is obvious, it is not linear, and the Pearson correlation coefficient is not relevant. A more general regression and the corresponding coefficient of determination would be more appropriate.
- In the third graph (bottom left), the distribution is linear, but should have a different regression line (a robust regression would have been called for). The calculated regression is offset by the one outlier which exerts enough influence to lower the correlation coefficient from 1 to 0.816.
- Finally, the fourth graph (bottom right) shows an example when one high-leverage point is enough to produce a high correlation coefficient, even though the other data points do not indicate any relationship between the variables.

The quartet is still often used to illustrate the importance of looking at a set of data graphically before starting to analyze according to a particular type of relationship, and the inadequacy of basic statistic properties for describing realistic datasets.

The datasets are as follows. The x values are the same for the first three datasets.



Anscombe's quartet

I		II		III		IV	
x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

5. What is Pearson’s R?

Ans:

Pearson's correlation coefficient is the covariance of the two variables divided by the product of their standard deviations. The form of the definition involves a "product moment", that is, the mean (the first moment about the origin) of the product of the mean-adjusted random variables; hence the modifier product-moment in the name.

Pearson's correlation coefficient when applied to a population is commonly represented by the Greek letter ρ (rho) and may be referred to as the population correlation coefficient or the population Pearson correlation coefficient. Given a pair of random variables (X,Y), the formula for ρ] is

$$\boxed{\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} \quad \textbf{(Eq.1)}}$$

where:

- cov is the covariance
- σ_X is the standard deviation of X
- σ_Y is the standard deviation of Y

The formula for ρ can be expressed in terms of mean and expectation. Since

$$\operatorname{cov}(X,Y) = \operatorname{E}[(X-\mu_X)(Y-\mu_Y)],$$[7]

the formula for ρ can also be written as

$$\boxed{\rho_{X,Y} = \frac{\operatorname{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \quad \textbf{(Eq.2)}}$$

where:

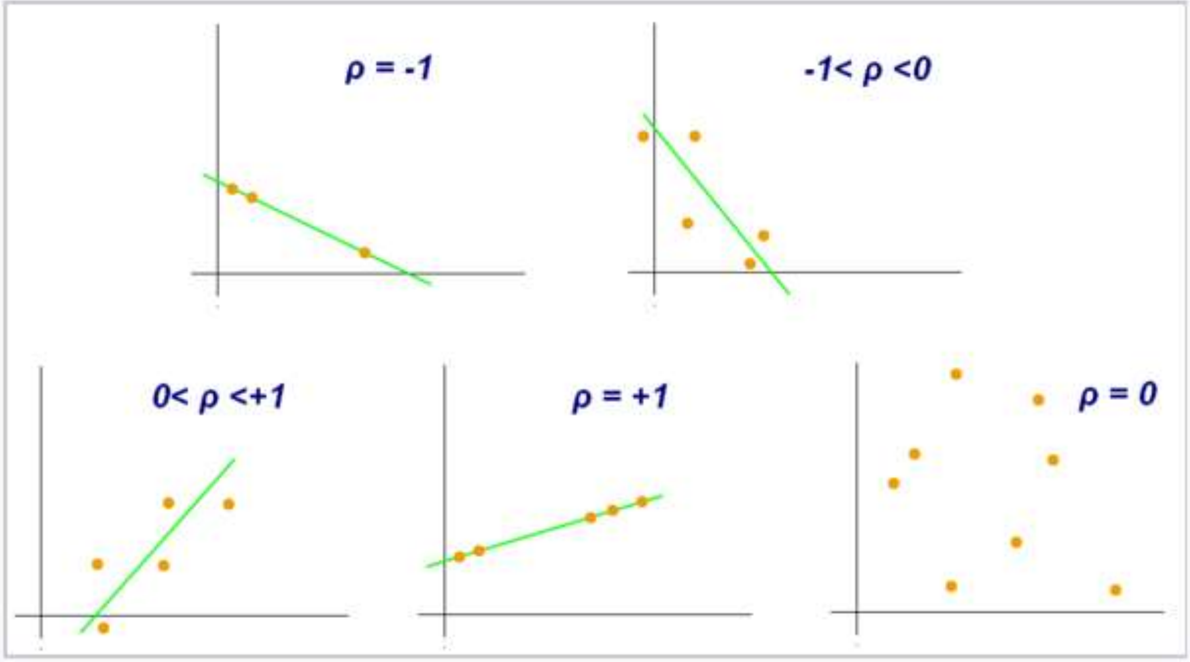
- σ_Y and σ_X are defined as above
- μ_X is the mean of X
- μ_Y is the mean of Y
- E is the expectation.

Pearson's correlation coefficient when applied to a sample is commonly represented by r_{xy} and may be referred to as the *sample correlation coefficient* or the *sample Pearson correlation coefficient*. We can obtain a formula for r_{xy} by substituting estimates of the covariances and variances based on a sample into the formula above. Given paired data $\{(x_1,y_1),\ldots,(x_n,y_n)\}$ consisting of n pairs, r_{xy} is defined as:

$$\boxed{r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \quad \textbf{(Eq.3)}}$$

where:

- n is sample size
- x_i, y_i are the individual sample points indexed with *i*
- $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ (the sample mean); and analogously for $\bar{y}$



Examples of scatter diagrams with different values of correlation coefficient (ρ)

6. What is scaling? Why is scaling performed? What is the difference between normalized scaling and standardized scaling?

Scaling is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step

Why Scaling

Most of the times, your dataset will contain features highly varying in magnitudes, units and range. But since, most of the machine learning algorithms use Eucledian distance between two data points in their computations, this is a problem.

If left alone, these algorithms only take in the magnitude of features neglecting the units. The results would vary greatly between different units, 5kg and 5000gms. The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes.

To supress this effect, we need to bring all features to the same level of magnitudes. This can be acheived by scaling.

How to Scale Features

There are four common methods to perform Feature Scaling.

1. Standardisation:

Standardisation replaces the values by their Z scores.

$$x' = \frac{x - \bar{x}}{\sigma}$$

This redistributes the features with their mean **μ = 0** and standard deviation **σ =1** . sklearn.preprocessing.scale helps us implementing standardisation in python.

2. Mean Normalisation:

$$x' = \frac{x - \text{mean}(x)}{\max(x) - \min(x)}$$

This distribution will have values between **-1 and 1**with **μ=0**.

Standardisation and **Mean Normalization** can be used for algorithms that assumes zero centric data like **Principal Component Analysis(PCA).**

3. Min-Max Scaling:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

This scaling brings the value between 0 and 1.

4. Unit Vector:

$$x' = \frac{x}{||x||}$$

Scaling is done considering the whole feature vecture to be of unit length.

Min-Max Scaling and **Unit Vector** techniques produces values of range [0,1]. When dealing with features with hard boundaries this is quite useful. For example, when dealing with image data, the colors can range from only 0 to 255.

Normalization typically means rescales the values into a range of [0,1]. *Standardization* typically means rescales data to have a mean of 0 and a standard deviation of 1 (unit variance).

.

7. You might have observed that sometimes the value of VIF is infinite. Why does this happen?

Ans

An infinite VIF value indicates that the corresponding variable may be expressed exactly by a linear combination of other variables (which show an infinite VIF as well).

These VIFs tell you there is perfect collinearity: you have completely redundant variables.

The VIF gives how much the variance of the coefficient estimate is being inflated by collinearity. If the VIF for a variable is 16 the associated standard error is four times as large as it would be if its VIF was 1. In such a case, the coefficient would have to be 4 times as large to be statistically significant at a given significance the level.

The VIF can be conceived as related to the R-squared of a particular predictor variable regressed on all other includes predictor variables.:

VIF of X1 = 1/(1 - R-squared of X1 on all other Xs).

If you only have 1 X or that X is orthogonal with all the other Xs; then

VIF = 1/(1-0) = 1 - so no variance inflation

If two Xs are perfectly correlated

VIF = 1/(1-1)= 1/0 = infinity that is the estimate is as imprecise as it can be.

8. What is the Gauss-Markov theorem?

Gauss Markov Theorem

The **Gauss Markov theorem** tells us that if a certain set of assumptions are met, the ordinary least squares estimate for regression coefficients gives you the *best linear unbiased estimate (BLUE)* possible.

Gauss Markov Assumptions

There are five Gauss Markov assumptions (also called *conditions*):

Linearity: the parameters we are estimating using the OLS method must be themselves linear.

Random: our data must have been randomly sampled from the population.

Non-Collinearity: the regressors being calculated aren't perfectly correlated with each other.

Exogeneity: the regressors aren't correlated with the error term.

Homoscedasticity: no matter what the values of our regressors might be, the error of the variance is constant.

Purpose of the Assumptions

The **Gauss Markov assumptions** guarantee the validity of ordinary least squares for estimating regression coefficients.

Checking how well our data matches these assumptions is an important part of estimating regression coefficients. When you know where these conditions are violated, you may be able to plan ways to change your experiment setup to help your situation fit the ideal Gauss Markov situation more closely.

In practice, the Gauss Markov assumptions are **rarely all met perfectly**, but they are still useful as a benchmark, and because they show us what 'ideal' conditions would be. They also allow us to pinpoint problem areas that might cause our estimated regression coefficients to be inaccurate or even unusable.

The Gauss-Markov Assumptions In Algebra

We can summarize the Gauss-Markov Assumptions succinctly in algebra, by saying that a linear regression model represented by

$yi = xi' \beta + \varepsilon i$

and generated by the ordinary least squares estimate is the best linear unbiased estimate (BLUE) possible if

- $E\{\varepsilon_i\} = 0, i = 1, \ldots, N$
- $\{\varepsilon_1 \ldots\ldots \varepsilon_n\}$ and $\{x_1 \ldots.., x_N\}$ are independent
- $\text{cov}\{\varepsilon_i, \varepsilon_j\} = 0, i, j = 1, \ldots., N\ I \neq j.$
- $V\{\varepsilon_1 = \sigma^2, i= 1, \ldots.N$

The first of these assumptions can be read as “The expected value of the error term is zero.”. The second assumption is collinearity, the third is exogeneity, and the fourth is homoscedasticity.

9. Explain the gradient descent algorithm in detail.
   Gradient Descent is an optimization algorithm used for minimizing the cost function in various machine learning algorithms. It is basically used for updating the parameters of the learning model.

Types of gradient Descent:

1. **Batch Gradient Descent:** This is a type of gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive. Hence if the number of training examples is large, then batch gradient descent is not preferred. Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent.
2. **Stochastic Gradient Descent:** This is a type of gradient descent which processes 1 training example per iteration. Hence, the parameters are being updated even after one iteration in which only a single example has been processed. Hence this is quite faster than batch gradient descent. But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the number of iterations will be quite large.
3. **Mini Batch gradient descent:** This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent. Here b examples where b<m are processed per iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations.

Variables used:

Let m be the number of training examples.
Let n be the number of features.
Note: if b == m, then mini batch gradient descent will behave similarly to batch gradient descent.

Algorithm for batch gradient descent :

Let hθ(x) be the hypothesis for linear regression. Then, the cost function is given by:
Let Σ represents the sum of all training examples from i=1 to m.

```
$J_{train}(\theta) = (1/2m)\ \Sigma(\ h_\theta(x^{(i)})\ -\ y^{(i)})^2$

Repeat {
 $\theta j = \theta j$ – (learning rate/m) * $\Sigma(\ h_\theta(x^{(i)})\ -\ y^{(i)})x_j^{(i)}$
    For every j =0 …n
}
```

Where xj(i) Represents the jth feature of the ith training example. So if *m* is very large, then the derivative term fails to converge at the global minimum.

Algorithm for stochastic gradient descent:

1) Randomly shuffle the data set so that the parameters can be trained evenly for each type of data.
2) As mentioned above, it takes into consideration one example per iteration.

```
Hence,

Let $(x^{(i)},y^{(i)})$ be the training example
$Cost(θ, (x^{(i)},y^{(i)})) = (1/2) Σ( hθ(x^{(i)})  - y^{(i)})^2$

$J_{train}(θ) = (1/m) Σ Cost(θ, (x^{(i)},y^{(i)}))$

Repeat {

For i=1 to m{

         $θ_j = θ_j – (learning rate) * Σ( h_θ(x^{(i)})  - y^{(i)})x_j^{(i)}$
        For every j =0 …n

                }
}
```

Algorithm for mini batch gradient descent:

Say b be the no of examples in one batch, where b < m. Assume b = 10, m = 100; **Note:** However we can adjust the batch size. It is generally kept as power of 2. The reason behind it is because some hardware such as GPUs achieve better run time with common batch sizes such as power of 2.

```
Repeat {
 For i=1,11, 21,…..,91


    Let Σ be the summation from i to i+9 represented by k.


    $θ_j = θ_j – (learning rate/size of (b) ) * Σ( h_θ(x^{(k)})  - y^{(k)})x_j^{(k)}$
        For every j =0 …n

}
```

Convergence trends in different variants of Gradient Descents:

In case of Batch Gradient Descent, the algorithm follows a straight path towards the minimum. If the cost function is convex, then it converges to a global minimum and if the cost function is not convex, then it converges to a local minimum. Here the learning rate is typically held constant.

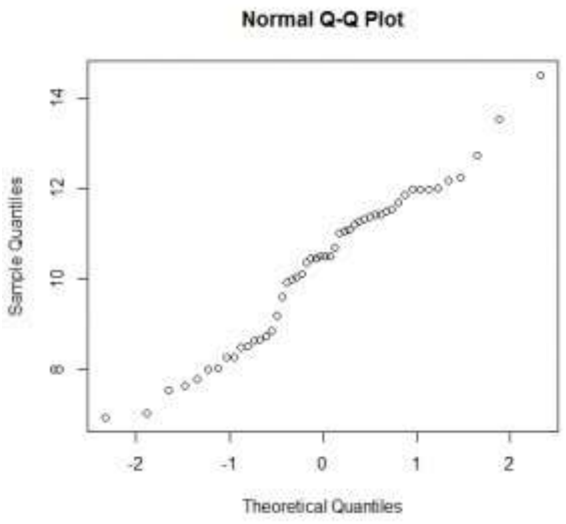
In case of stochastic gradient Descent and mini-batch gradient descent, the algorithm does not converge but keeps on fluctuating around the global minimum. Therefore in order to make it converge, we have to slowly change the learning rate. However the convergence of Stochastic gradient descent is much noisier as in one iteration, it processes only one training example.

10. What is a Q-Q plot? Explain the use and importance of a Q-Q plot in linear regression.

Ans:

The Q-Q plot, or quantile-quantile plot, is a graphical tool to help us assess if a set of data plausibly came from some theoretical distribution such as a Normal or exponential. For example, if we run a statistical analysis that assumes our dependent variable is Normally distributed, we can use a Normal Q-Q plot to check that assumption. It's just a visual check, not an air-tight proof, so it is somewhat subjective. But it allows us to see at-a-glance if our assumption is plausible, and if not, how the assumption is violated and what data points contribute to the violation.

A Q-Q plot is a scatterplot created by plotting two sets of quantiles against one another. If both sets of quantiles came from the same distribution, we should see the points forming a line that's roughly straight. Here's an example of a Normal Q-Q plot when both sets of quantiles truly come from Normal distributions.



Use and Importance in Linear Regression:

The process of statistical modeling involves three distinct stages: formulating a model, fitting the model to data, and checking the model. Often, the third stage suggests a reformulation of the model that leads to a repetition of the entire cycle and, one hopes, an improved model. In this section we discuss techniques that can be used to check the model. The Q-Q plot used as Regression diagnostic aid for checking the model

One type of diagnostic aid is the probability plot, a graph of the residuals versus the expected order statistics of the standard normal distribution. This graph is also called a Q-Q Plot because it plots quantiles of the data versus quantiles of a distribution. The Q-Q plot may be constructed using raw, standardized or jack-knifed residuals, although I recommend the latter.

The first step in constructing a Q-Q plot is to order the residuals from smallest to largest, so r(i)r(i) is the ii-th smallest residual. The quantity r(i)r(i) is called an order statistic. The smallest value is the first order statistic and the largest out of nn is the nn-th order statistic.

The next step is to imagine taking a sample of size nn from a standard normal distribution and calculating the order statistics, say z(i)z(i). The expected values of these order statistics are sometimes called rankits. A useful approximation to the ii-th rankit in a sample of size nn is given by

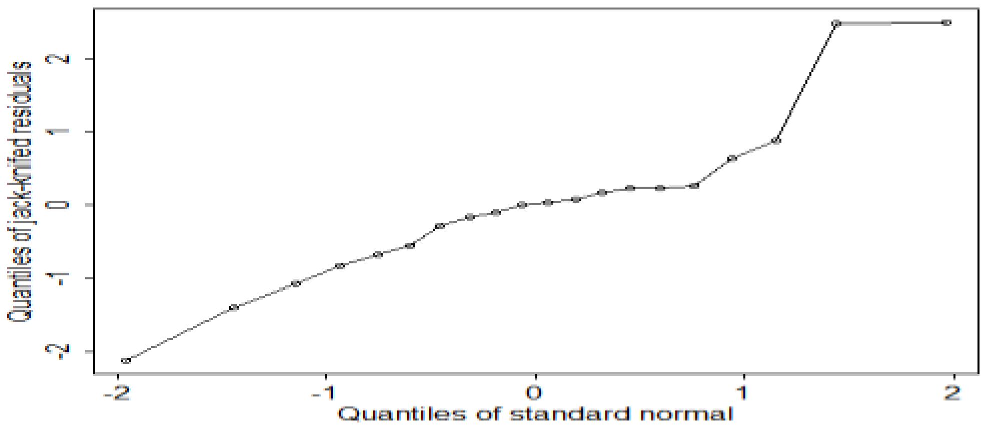
E(z(i))≈Φ−1[(i−3/8)/(n+1/4)]E(z(i))≈Φ−1[(i−3/8)/(n+1/4)]

where Φ−1Φ−1 denotes the inverse of the standard normal distribution function. An alternative approximation proposed by Filliben (1975) uses Φ−1[(i−0.3175)/(n+0.365)]Φ−1[(i−0.3175)/(n+0.365)] except for the first and last rankits, which are estimated as Φ−1(1−0.51/n)Φ−1(1−0.51/n) and Φ−1(0.51/n)Φ−1(0.51/n), respectively. The two approximations give very similar results.

If the observations come from a normal distribution we would expect the observed order statistics to be reasonably close to the rankits or expected order statistics. In particular, if we plot the order statistics versus the rankits we should get approximately a straight line.

Figure shows a Q-Q plot of the jack-knifed residuals from the analysis of covariance model fitted to the program effort data. The plot comes very close to a straight line, except possibly for the upper tail, where we find a couple of residuals somewhat larger than expected.

Q-Q Plot of Residuals From Analysis of Covariance Model
of CBR Decline by Social Setting and Program Effort



In general, Q-Q plots showing curvature indicate skew distributions, with downward concavity corresponding to negative skewness (long tail to the left) and upward concavity indicating positive skewness. On the other hand, S-shaped Q-Q plots indicate heavy tails, or an excess of extreme values, relative to the normal distribution.

Filliben (1975) has proposed a test of normality based on the linear correlation between the observed order statistics and the rankits and has published a table of critical values. The 5% points of the distribution of rr for n=10(10)100n=10(10)100 are shown below. You would reject the hypothesis of normality if the correlation is less than the critical value. Note than to accept normality we require a very high correlation coefficient.

nn	10	20	30	40	50	60	70	80	90	100
rr	.917	.950	.964	.972	.977	.980	.982	.984	.985	.987

The Filliben test is closely related to the Shapiro-Francia approximation to the Shapiro-Wilk test of normality. These tests are often used with standardized or jack-knifed residuals, although the fact that the residuals are correlated affects the significance levels to an unknown extent. For the program effort data in Figure.the Filliben correlation is a respectable 0.966. Since this value exceeds the critical value of 0.950 for 20 observations, we conclude that we have no evidence against the assumption of normally distributed residuals.