Modeling Toolkit

Article 2: Linear Regression

Andrew Freeman

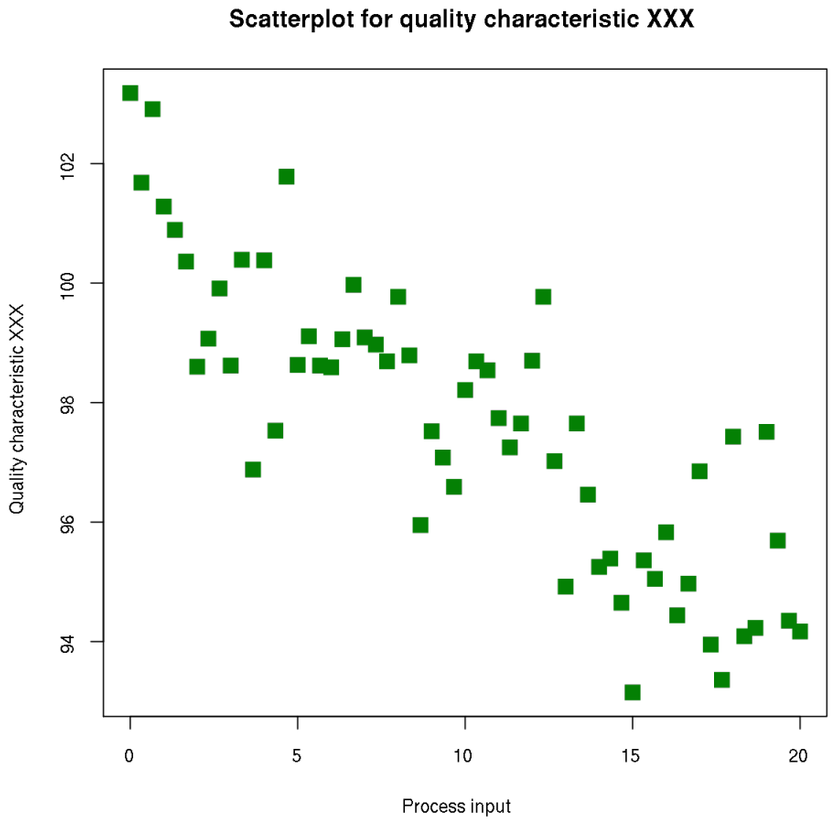
March 31, 2022

1 Introduction

Welcome back to the Modeling Toolkit! In this article, we will explore linear regression, one of the most ubiquitous tools in data science. It underlies a lot of more complex models and is a great starting point.

2 Intuition

We start with the basic ideas behind linear regression. No math, just concepts.
Imagine someone gave you a scatterplot that looked like this



Now, they ask you to give them a function relating the x and y variables. How would you do it? The simplest way would be to just pick a number. No matter what x you give me, I'm going to predict 98. That's not going to do well, but it's pretty easy. If you wanted to do better, you might try drawing a line through the data. More complex functions would probably be too difficult to derive by hand (for example, you could draw a line connecting each point in the plot, but you wouldn't be able to write this as a function in a reasonable amount of time). A line though, you could do fairly easily. You would just draw the line the best you can through the data and then dig through your high school math memories to remember how to write the equation of a line. This turns out to be a great idea. And in fact, the math isn't that horrible as we will soon see. But first, you might be thinking that this is a bad example. Of course a line fits this data; it was chosen to prove a point. What if our data is non-linear? For example, what if we look at physics and have distance in terms of acceleration squared? Will we need to derive a quadratic regression? And then a cubic and so on? It turns out that the answer is no. All we are doing is drawing a line between two variables; in theory, we don't care what those variables are. So rather than plot distance vs. acceleration, why don't we plot distance vs. acceleration_squared? Then, we can draw our line. When we actually go to interpret our function, we'll have to keep in mind that we did this, but that's fine. This idea extends to any function of the **x** values, and this is a crucial point. Linear regression is **not** linear in terms of the **x** variables. Rather, it is linear in terms of the **coefficients**. That means that each x gets it's own coefficeient, even if x represents some quantity squared.

3 Defining the Model

Using the framework from Article 1, let's define a linear regression:

3.1 Function Class

The function class for (ordinary) least squares (often abbreviated OLS) is

$$\mathcal{F} = a_0 + \sum_{n=1}^{n} a_j x_j$$

for $a_0, a_i \in \mathbb{R}$.

First, you might ask what ordinary means above. There are derivations of linear regression that use fancy techniques and are more complicated but are still consider "linear regression". To avoid ambiguity, we often call the technique used in this article "ordinary", but when I don't specify, assume "linear regression" means OLS.
Let's look at our function class more. In the simplest form of OLS, we have just one predictor x. Therefore, our function is $a_0 + a_1 x$. If you remember your high school math, you can think of this as $y = mx + b$ where m and b are coefficients we can change to make our function fit the data. If we want to use more predictors, we just add more coefficients: $y = a_0 + a_1 x_1 + ... + a_n x_n$. This is called multivariate regression.

3.2 Score Function

For OLS, we use the least squares score function,

$$E_{\mathbf{Xy}}(\mathbf{y} - a_0 - \sum_{j=1}^{n} a_j \mathbf{x}_j)^2$$

This is for our population; however, we don't have the whole population. If we did, why would we bother to model it? Instead, we substitute the expected value for a summation:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - a_0 - \sum_{j=1}^{n} a_j x_{ij})^2$$

This function is also called the Residual Sum of Squares (RSS). A residual is the difference between our prediction and the true value, $y - \hat{y}$. Note that here $\hat{y} = a_0 + \sum_{j=1}^{n} x_j$, so we can equivalently write

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2$$

and we see where the name comes from.
N is the number of observations we have. As an example, imagine we have the following data:

$$y = 0, \hat{y} = 1$$

$$y = 1, \hat{y} = 2$$

$$y = 1, \hat{y} = 1$$

Then, we have $N = 3$ and our loss is

$$L(\boldsymbol{X}, \mathbf{y}) = (0-1)^2 + (1-2)^2 + (1-1)^2 = 2$$

Note that since we don't know what $\boldsymbol{X}$ is here, we may have different predictions for data with the same y. Our goal will be to minimize this loss over all of the data we have.
Before we get to the search method, we're going to rewrite this loss in a vectorized form.
First, some notation. We recall $\mathbf{y}$ and $\boldsymbol{X}$ from Article 1 to signify the vector of responses and matrix of predictors. In $\boldsymbol{X}$, each column is a new variable. It can be helpful to remember $\mathbf{y}$ and $\boldsymbol{X}$ as a table where each row is one observation

$$\begin{pmatrix} y_1 & x_{11} & \dots & x_{n1} \\ \vdots & \vdots & & \vdots \\ y_N & x_{1N} & \dots & x_{nN} \end{pmatrix}$$

Then, we just separate them out.

$$\begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \begin{pmatrix} x_{11} & \dots & x_{n1} \\ \vdots & & \vdots \\ x_{1N} & \dots & x_{nN} \end{pmatrix}$$

Next, let's consider vectorizing our loss function. Rather than sum up over each entry, can we do this using matrices? First, let's combine our a_i values into a vector that we call β. Now we have

$$\beta = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}$$

Then, take $\boldsymbol{X}$ and add a column of 1s to it (which we denote $\boldsymbol{X}_1$, so we have

$$\begin{pmatrix} 1 & x_{11} & \dots & x_{n1} \\ \vdots & \vdots & & \vdots \\ 1 & x_{1N} & \dots & x_{nN} \end{pmatrix}$$

Now, if we multiply $\boldsymbol{X}_1\beta$, we get a vector of $\hat{y}$:

$$\boldsymbol{X}_1\beta = \begin{pmatrix} 1 & x_{11} & \dots & x_{n1} \\ \vdots & \vdots & & \vdots \\ 1 & x_{1N} & \dots & x_{nN} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} a_0 + x_{11}a_1 + ... + x_{1n}a_n = \hat{y_1} \\ \vdots \\ a_0 + x_{N1}a_1 + ... + x_{Nn}a_n = \hat{y_n} \end{pmatrix}$$

Now, we can write our residuals as

$$\mathbf{y} - \boldsymbol{X}_1\beta$$

Next, we want the sum of the squares of the residuals (divided by N). Here's where vector properties make this really nice: recall that for a vector $\boldsymbol{a}$, we can write

$$\boldsymbol{a}^T\boldsymbol{a} = a_1^2 + a_2^2 + ... + a_n^2$$

In other words, this *is* the sum of squares. So, we get

$$(\mathbf{y} - \boldsymbol{X}_1\beta)^T(\mathbf{y} - \boldsymbol{X}_1\beta)$$

What about the $\frac{1}{N}$? Well, we can throw it out because it is a monotonic function. Monotonic means that if you increase x, y must increase (decrease for a negative monotonic function) or stay the same. Examples include a linear function or an exponential function. A function that is not monotonic is the sine function. Why does this allow us to throw out the $\frac{1}{N}$? We can do this because we know it won't affect our answer; if b minimizes our function with the $\frac{1}{N}$, b will still minimize the modified function. As a hard example, think of buying bottles of soda from the store (as in the classic math problem). Bottles cost \$2 each. You pay the minimum amount of \$0 when you buy none. If you have to buy at least one, you minimize your cost at 1. Now say that bottles are half off and we multiply our cost by $\frac{1}{2}$. We'll still minimize our total at the same points, we just will pay less now. The same idea holds here.
So, we wish to minimize

$$(\mathbf{y} - \boldsymbol{X}_1\beta)^T(\mathbf{y} - \boldsymbol{X}_1\beta)$$

You may see this written more succinctly as

$$||\mathbf{y} - \boldsymbol{X}_1\beta||^2$$

where $||\cdot||$ is the 2-norm of the vector and $||\mathbf{x}|| = \sqrt{\mathbf{x}^T\mathbf{x}}$.

3.3 Optimization Search Method

For linear regression, the search method is very easy to implement, since we can find a closed form solution! I'll first derive it mathematically, and then try to explain it geometrically.
We wish to minimize

$$(\mathbf{y} - \boldsymbol{X}_1\beta)^T(\mathbf{y} - \boldsymbol{X}_1\beta)$$

Going back to high school calculus, you may remember that we can try to minimize (or maximize) a function by looking at its critical points, or places where it has a derivative of 0. **This is a very important idea for a lot of machine learning techniques**, so it may be worthwhile to touch up on this idea (i.e. skim this).
So, let's find the derivative, using vector calculus, which sounds scary but really isn't too bad, especially here. We want

$$\frac{d}{d\beta}(\mathbf{y} - \boldsymbol{X}_1\beta)^T(\mathbf{y} - \boldsymbol{X}_1\beta)$$

First, expand

$$= \frac{d}{d\beta}\mathbf{y}^T\mathbf{y} - \frac{d}{d\beta}(\boldsymbol{X}_1\beta)^T\mathbf{y} - \frac{d}{d\beta}\mathbf{y}^T(\boldsymbol{X}_1\beta) + \frac{d}{d\beta}(\boldsymbol{X}_1\beta)^T(\boldsymbol{X}_1\beta)$$

Note that

$$(\boldsymbol{X}_1\beta)^T\mathbf{y} = (\mathbf{y}^T(\boldsymbol{X}_1\beta))^T$$

But this is a vector times a vector, so these values are scalars. The transpose of a scalar is a scalar (i.e. $5^T = 5$), so we can simplify. We also expand out the last term

$$= \frac{d}{d\beta}\mathbf{y}^T\mathbf{y} - 2\frac{d}{d\beta}(\boldsymbol{X}_1\beta)^T\mathbf{y} + \frac{d}{d\beta}\beta^T\boldsymbol{X}_1^T\boldsymbol{X}_1\beta$$

Our first term has no β, so its derivative is 0. Our second can be worked on similarly to a normal derivative, and we get $-2\boldsymbol{X}_1^T\mathbf{y}$. Finally, we use a result from vector calculus to say that

$$\frac{d}{d\beta}\beta^T\boldsymbol{X}_1^T\boldsymbol{X}_1\beta = 2\boldsymbol{X}_1^T\boldsymbol{X}_1\beta$$

So

$$\frac{d}{d\beta}(\mathbf{y} - \boldsymbol{X}_1\beta)^T(\mathbf{y} - \boldsymbol{X}_1\beta) = -2\boldsymbol{X}_1^T\mathbf{y} + 2\boldsymbol{X}_1^T\boldsymbol{X}_1\beta$$

To find a potential minimum, set this to zero:

$$-2\boldsymbol{X}_1^T\mathbf{y} + 2\boldsymbol{X}_1^T\boldsymbol{X}_1\beta = 0$$

$$\boldsymbol{X}_1^T\boldsymbol{X}_1\beta = \boldsymbol{X}_1^T\mathbf{y}$$

Remembering that we "divide" matrices by inversion,

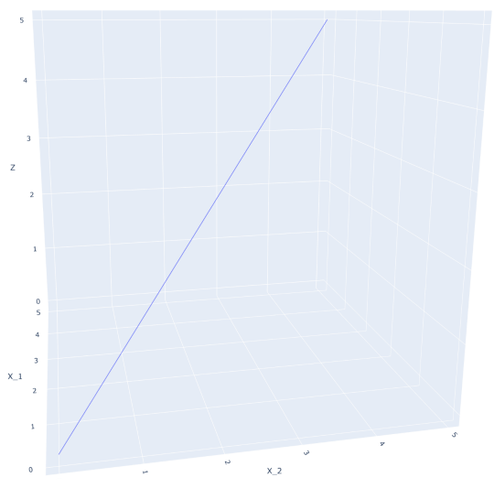
$$\beta = (\boldsymbol{X}_1^T\boldsymbol{X}_1)^{-1}\boldsymbol{X}_1^T\mathbf{y}$$

assuming that $(\boldsymbol{X}_1^T\boldsymbol{X}_1)^{-1}$ exists.

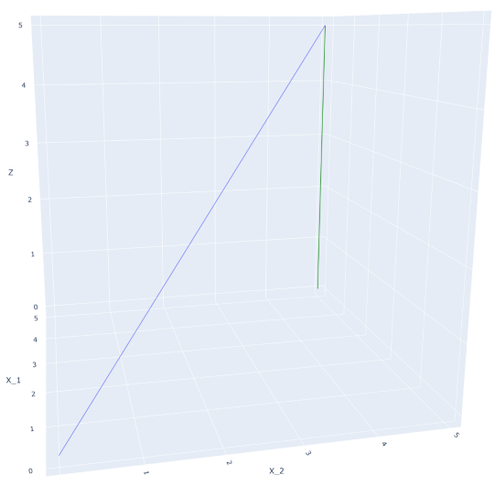
To verify that this is indeed the minimum, we would need to differentiate again and check the sign; this is left out here to simplify.
That's it; we have an explicit formula for our answer! A couple of quick notes: first, you can derive this from the summation form of the loss function using regular calculus, and you will get the exact same thing.
Second, when *does* $(\boldsymbol{X}_1^T\boldsymbol{X}_1)^{-1}$ exist? This exists when the rank of $\boldsymbol{X}$ is n where n is the number of predictors we are using. In practice, this means that we have more observations than predictors (our matrix is longer than it is wide) and our predictors are linearly independent. This just means that I cannot write one predictor as a function of another predictor (i.e. one predictor is $\mathbf{x}$ and another is $3\mathbf{x}$). If this is the case, we will have an infinite number of solutions and we need to use regularization (our next article) to find the best β.

3.4 Geometric Intuition

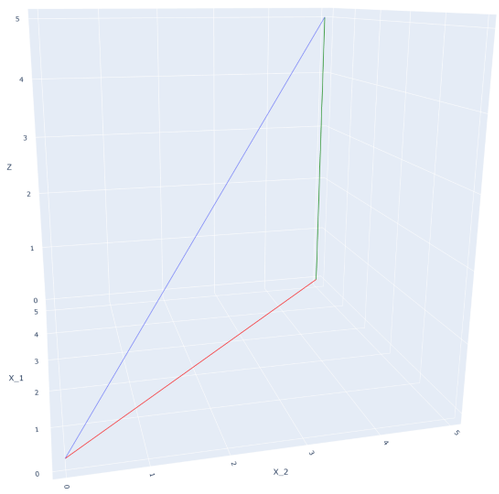
Seeing the math is great, but I tend to understand concepts better visually. So, let's look at geometrically why our answer makes sense. First, let's look at our $\mathbf{y}$ vector in three dimensions:



Our $\mathbf{y}$ vector is given in blue; this is the true value of our observation. Note our axes: we have $\mathbf{x}_1$ and $\mathbf{x}_2$, which are the two predictors we are using. Then, we have $\mathbf{z}$, the dimension(s) of $\mathbf{y}$ that are explained by variables we do not have, including noise. Since we do not have the variables represented by $\mathbf{z}$, we cannot plot anything in $\mathbf{z}$. Therefore, our best guess for $\mathbf{y}$ will have to lie in the $\mathbf{x}_1$-$\mathbf{x}_2$ plane (the plane where $\mathbf{z} = 0$). So what is the best guess in this plane? Intuitively, it should be the line that is cloeset to $\mathbf{y}$. A basic result of geometry holds that the shortest line between a plane and a vector is orthogonal to the plane. This means that we should be able to draw a line straight up from our solution to the true solution. Let's drop a line from $\mathbf{y}$ to our plane:



This green line is the visual representation of our error: it represents the least possible error we could hope for. It also allows us to define our solution:



This red vector is our desired solution, since it is orthogonal to our error. But how do we find the equation for this red line? First, recall that this red line is of the form $\boldsymbol{X}\beta$ where $\boldsymbol{X}$ has x_1 and x_2 and we need to find β. Next, we note that our error (the green line) is the difference between our prediction and the true value, $\mathbf{y} - \boldsymbol{X}\beta$. By design, this vector is orthogonal to our solution:

$$\boldsymbol{X}\beta \perp (\mathbf{y} - \boldsymbol{X}\beta)$$

You may recall from geometry that if two vectors are orthogonal, their dot product is 0:

$$(\boldsymbol{X}\beta)^T(\mathbf{y} - \boldsymbol{X}\beta) = 0$$

Distributing and moving terms over:

$$\beta^T\boldsymbol{X}^T\mathbf{y} = \beta^T\boldsymbol{X}^T\boldsymbol{X}\beta$$

If we assume that β is not zero, then

$$\boldsymbol{X}^T\mathbf{y} = \boldsymbol{X}^T\boldsymbol{X}\beta$$

And solving out,

$$\beta = (\boldsymbol{X}^T\boldsymbol{X})^{-1}(\boldsymbol{X}^T\mathbf{y})$$

4 Implementation

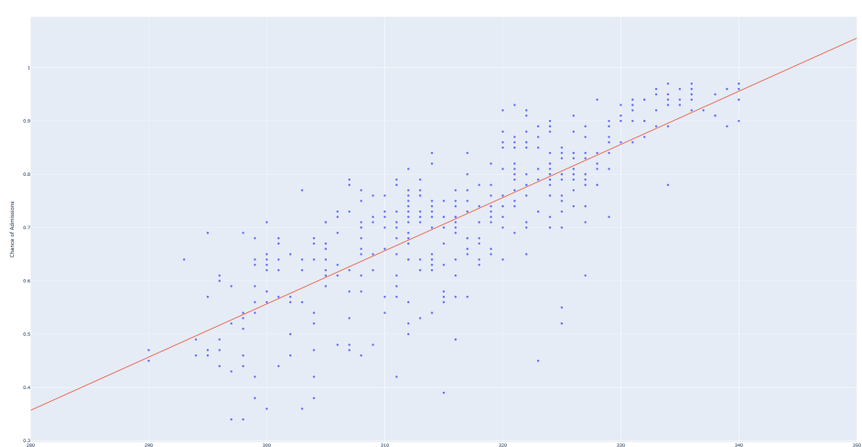
Let's write our own linear regression model. Since we have a closed form expression, this is really easy.
For this example, I'm using this graduate school admissions data. We'll do two models: one predicting Chance of Admit using just GRE Score and one where we also use TOEFL Score.
Now let's write out function to calculate linear regression.
Import the data

```
import pandas as pd

def linear_regression(df, x_cols, y_col, intercept = True):
    # Get our y column as a vector
    y = df[y_col]

    # If intercept, add our column of ones
    if intercept:
        X = np.append(np.ones((len(df),1)),
                      df[x_cols].to_numpy(), axis = 1)
    else:
        X = df[x_cols].to_numpy()

    # beta = (X^TX)^-1 X^TY
    beta = np.dot(
        np.linalg.inv(np.dot(np.transpose(X),X)), # (X^TX)^-1
        np.dot(np.transpose(X),y)) #X^TY
    return beta
```

In our function, we pass a list of columns by name for $\boldsymbol{X}$ and a single string for our $\mathbf{y}$ column. We can also choose to enable or disable our intercept. Note that the above code will work with any number of predictors we choose (as long as $\boldsymbol{X}^T\boldsymbol{X}$ is invertible - more on this shortly). Now let's build our models:

```
import numpy as np
import plotly.express as px
import plotly.io as pio
import plotly.graph_objects as go
pio.renderers.default = "browser"
```

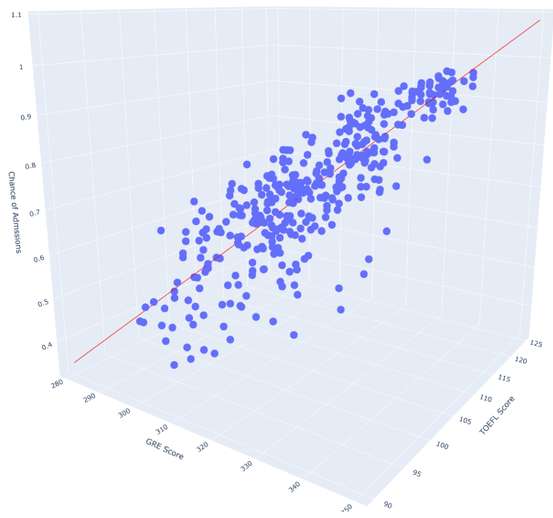
```
# Read in data
df = pd.read_csv('Data/Admission_Predict.csv')
df.dropna(inplace=True)

# Define variables
x_cols = ['GRE Score']
y_col = ['Chance of Admit ']

# Get beta
beta = linear_regression(df, x_cols, y_col)

# Plot the data
fig = px.scatter(df, x = x_cols[0], y = y_col)

# Add our fitted line
fig.add_trace(go.Scatter(
    x = np.linspace(280,350), # range of GRE scores
    y = np.linspace(280,350)*beta[1]+beta[0]) # y = beta_0 + beta_1*x_1
    )

# Update axis titles
fig.update_xaxes(title_text='GRE Score')
fig.update_yaxes(title_text='Chance of Admissions')
fig.show()
```



Pretty good fit! Let's add in TOEFL score:

```
# Define variables
x_cols = ['GRE Score', 'TOEFL Score']
y_col = ['Chance of Admit ']

# Get beta
beta = linear_regression(df, x_cols, y_col)

# Plot the data
fig = px.scatter_3d(df, x = df[x_cols[0]], y = df[x_cols[1]], z = y_col[0])

# Add in the fitted line
fig.add_trace(go.Scatter3d(x = np.linspace(280,350), #GRE Score range
                           y = np.linspace(90,125), #TOEFL Score range
                           z = np.linspace(280,350)*beta[1]+np.linspace(90,125)*beta[2]+be
                           mode = 'lines', marker=dict(
                                                color='red',
                                            line=dict(
                                                color='red')
                                            )
                           )
              )

# Label axes
fig.update_layout(scene = dict(
                    xaxis_title='GRE Score',
                    yaxis_title='TOEFL Score',
                    zaxis_title='Chance of Admssions'))
```

```
fig.show()
```



It may be kind of tough to see from the static image, but the result is a better fit than we got just from GRE. If we wanted to actually evaluate our model, we would need to look at various metrics of fitting, such as RSS. We would also probably want to use cross validation if we wanted to use this model for predictions, but for now we're okay with just showing the proof of concept.

This is an important note: adding an additional variable to a linear regression model **will not make your training predictions worse**. It can cause you to overfit your data, but your bias will always decrease. Why is this? Recall that OLS finds the best $\hat{\mathbf{y}} = \boldsymbol{X}_1\beta = \beta_0 + \beta_1\mathbf{x}_1 + ... + \beta_n\mathbf{x}_n$. If we now add $\beta_{n+1}\mathbf{x}_{n+1}$, we can make $\beta_{n+1} = 0$ and do exactly as well as without the new $\mathbf{x}_{n+1}$. So to anthropomorphize the model, if the new variable does nothing to improve our RSS, the model will just choose not to use it and make $\beta_{n+1} = 0$. **Caveat:** Adding $\mathbf{x}_{n+1}$ does not mean that you will have the same $\beta_0, ..., \beta_n$ (in fact you probably won't), just that your RSS will not increase.

Our model works great, but in practice, we should use models that have been thoroughly built and vetted, such as scikit-learn. This also has the advantage of being able to calculate regularized linear regression models, which we will discuss in our next article.

4.1 Singular X

Above, we noted that our code will not run when our $\boldsymbol{X}^T\boldsymbol{X}$ is invertible (non-singular). This makes sense; if our matrix is not invertible, then we can't solve for it's inverse. When is this true? When the rank of $\boldsymbol{X}$ is equal to the number of columns of $\boldsymbol{X}$. This means that none of the columns are linear combinations of the others. So for instance, we might want to regress on GRE ($\mathbf{x}_1$), TOEFL ($\mathbf{x}_2$), and a new variable $\mathbf{x}_3$ we make up that averages GRE and TOEFL. But now $0.5\mathbf{x}_1 + 0.5\mathbf{x}_2 = \mathbf{x}_3$, so our matrix is singular and our method will not work (quick note: running our algorithm with an intercept and $\mathbf{x}_3$ will in fact work because of rounding point errors, but we shouldn't use it. Running it without an intercept will throw an error as expected). What do we do if we're deadset on adding this new variable? We need a way to make $\boldsymbol{X}^T\boldsymbol{X}$ non-singular. As it turns out, we'll be able to do this with regularization.

5 Model Assumptions

What assumptions underlie linear regression? Some of these were made explicit, while others were not, but these are important things to know:

- Linearity: There exists a linear relationship between $\mathbf{y}$ and $\boldsymbol{X}$. This one is obvious enough
- Independence: Our observations (a single y with its corresponding xs) are independent of each other. That means that the result of one observation shouldn't influence other observations. If this is violated, your regression probably won't make sense. Consider the following example: every morning, we add a dollar to our rainy day fund if it's sunny outside. So our $\mathbf{x}$ is whether or not it's sunny, and our y is the total amount we have in our fund. Clearly, y_{i+1} depends on y_i, so we can't really fit a model to this: if I tell you that today is sunny, you have no way of predicting the amount in the fund without knowing previous results. To circumvent this, we could add the amount in the fund yesterday as a variable, and now our observations are independent since a single observation tells us all we need to know to predict y
- Homoscedasticity: A big word meaning that our errors are constant everywhere. Basically, if our errors are of order 1 ($O(1)$) at low x values and order 100 ($O(100)$) at high x values, we can't expect our model to do well because we ask it to minimize the average error. If this value isn't constant, it's only going to try to fit well in the areas where we have high errors, but we want it to fit well everywhere over x.

- Normality of errors: The errors, on top of being homoscedastic, are normally distributed. This assumption comes into play with the way that we define our loss (using expectations). Consider an example where our error can be -1 with probability 0.99 and 1000 with probability 0.01 (that is, $y = \beta\mathbf{x} + c$ where $c = -1$ or 1000). This error term is clearly not normal. If I asked you to fit this, you would almost definitely want to assume $c = -1$. However, to minimize the average expected error we would instead want to set c to be 9.01, which would almost always give us the wrong answer. This is an extreme example, but hopefully it kind of explains the statistical underpinnings of this assumption

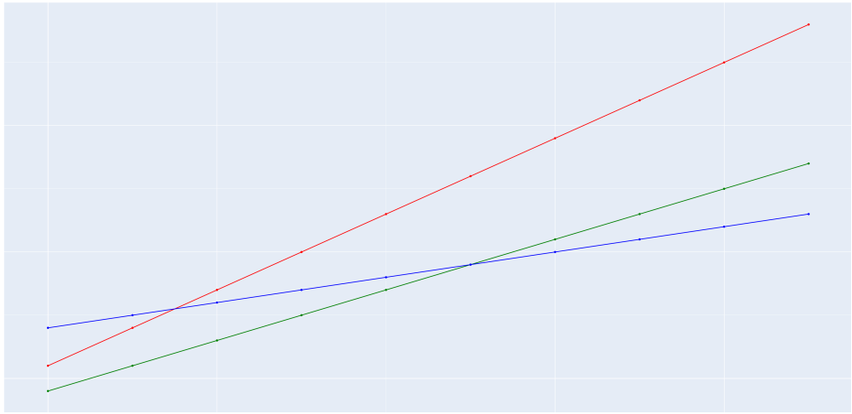
In practice, these are all important to consider, and we can check these using statistical tests or graphically. However, more often than not, these assumptions will hold when you want to fit a linear model, and if you cross-validate your model and get good results, you're probably okay.
There's one more assumption that is important if we want to interpret our model. If you're only interested in prediction, feel free to ignore this assumption.

- X Independence: The different variables we use to make our prediction (our xs) are independent of one another

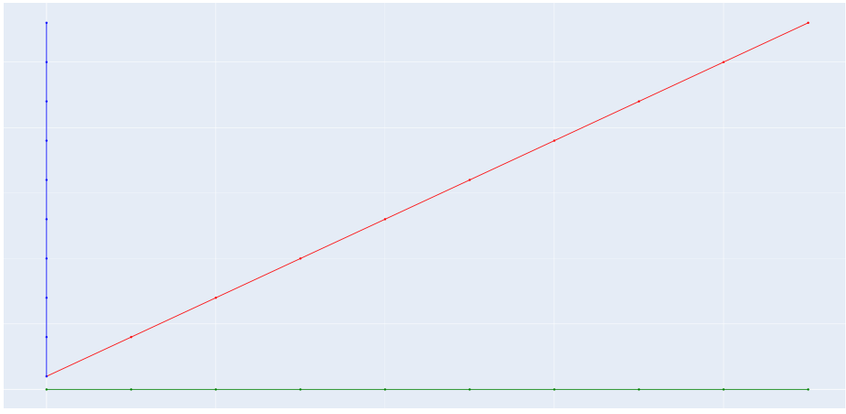
First, a very important note: **Independence and non-singulariy are NOT the same thing**. Two variables can be linear combinations of each other and be independent, and two dependent variables need not be linear combinations of each other. That said, why do we need this assumption? And why don't we need it if we only care about prediction?
Take a very stupid modelling scenario: we want to predict the perimeter of a right triangle given the lengths of the three legs. Obviously, we should get

$$P = (1)a + (1)b + (1)c$$

The interpretation from this in the standard modelling sense is that if we increase the length of a by 1, the perimeter will increase by 1. But we know that this isn't true: if we increase a, we must also increase c by the Pythagorean Theorem. This highlights the often implicit assumption: if we increase a by 1, P will increase by 1 *if we hold everything else constant*. But we can't do that in this case because c depends on a. This is one problem with dependent variables in interpretations. However, note that our model is perfectly accurate, so we can still use it to make predictions.
Another problem is assigning importance. Imagine we are fitting a model $\mathbf{y}$ with predictors $\mathbf{x}_1, \mathbf{x}_2$ and $\mathbf{x}_3 = \mathbf{x}_1 + \mathbf{x}_2 + d$ where d is some unknown constant. That is, $\mathbf{x}_3$ is not constructed, but we know it is the combination of $\mathbf{x}_1$, $\mathbf{x}_2$, and something else. Strictly speaking, we can't fit this by OLS, but we can use our regularization method (that we will discuss next article). We can also try to fit the model anyways and if we have some numeric instability (rounding errors), we may get a fit anyways. Now the question is how important are $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$? There isn't really a good answer here since any importance assigned to $\mathbf{x}_3$ implicitly assigns importance to $\mathbf{x}_1$ and $\mathbf{x}_2$, but our model won't know that when it fits β so we can't just use those coefficients. If we want to interpret our values, we can orthogonalize our values. Assuming $\mathbf{x}_1$ and $\mathbf{x}_2$ are already independent, we can try to create $\tilde{\mathbf{x}}_3 = \mathbf{x}_3 - \mathbf{x}_1 - \mathbf{x}_2$. Now, $\tilde{\mathbf{x}}_3$ is independent of our other varaibles and we can directly interpret our results. To show this graphically, we can take a rough approach using vectors. We are trying to predict the red line using the green and the blue lines:



Can you solve this just by looking at it? Probably not. What about this one:



Now you can do it immediately! The blue line gives you the y value and the green gives you the x value. Why is this easier? Because now each of our predictors gives us one distinct dimension of our prediction and there is no interaction between the two variables. Note that the math on this is not rigorous; you could solve the first case above if you wanted to (hence the statement that we can still use colliniear variables for predictions), but its much harder to interpret.
Finally, a semi-related and interesting note on confounding variables and interpretation.

6 Linear Regression Characteristics

Last but not least, we mention a very important characteristic of linear regression: **linear regression is scale and shift invariant**, assuming we use an intercept. This means that if I fit a model with x or with $2x + 4$, I will get the same exact fit (just with β scaled and shifted). A quick proof is nice to show this and can be found in the appendix, but feel free to skip it if you're willing to take my word. This means that for linear regression, as long as we include an intercept (which you almost always want to do unless you have a very specific reason not to), it doesn't matter if we normalize our data.

7 Summary

Let's summarize our model: linear regression works by finding the line through our data with the smallest residual sum of squares (RSS). We have a closed form solution for this line, which makes linear regression fast and easy to implement. When should you use linear regression? Whenever you think your target function might be linear in its coefficients (remember, we can apply any functions we wish to our variables before fitting and still use linear regression). Linear regression is often a good place to start since it's the easiest model to fit. The downside to this is that it often is not the best fitting model and you can do better. Further, linear regression may over fit if you have a lot of predictors and cannot fit a model with a linear combination of variables. A corollary of this is that OLS cannot fit a model with more predictors than observations, a scenario that is fairly common in some fields, such as genetics. In our next article, we will look at regularization, a method to fix these last two problems as well as to reduce variance.

8 Appendix: Invariance Proof

Note that any linear mapping (i.e. $a * x$) can be represented by a matrix, call it $\boldsymbol{A}$. Then, we can take

$$\tilde{X}_1 = \boldsymbol{X}_1\boldsymbol{A}$$

Then, we get

$$\begin{aligned}
\tilde{\beta} &= (\tilde{X}_1^T\tilde{X}_1)^{-1}\tilde{X}_1^T\mathbf{y} \\
&= ((\boldsymbol{X}_1\boldsymbol{A})^T\boldsymbol{X}_1\boldsymbol{A})^{-1}(\boldsymbol{X}_1\boldsymbol{A})^T\mathbf{y} \\
&= (\boldsymbol{A}^T\boldsymbol{X}_1^T\boldsymbol{X}_1\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{X}_1^T\mathbf{y} \\
&= \boldsymbol{A}^{-1}(\boldsymbol{X}_1^T\boldsymbol{X}_1)^{-1}\boldsymbol{A}^{-T}\boldsymbol{A}^T\boldsymbol{X}_1^T\mathbf{y} \\
&= \boldsymbol{A}^{-1}(\boldsymbol{X}_1^T\boldsymbol{X}_1)^{-1}\boldsymbol{X}_1^T\mathbf{y} \\
&= \boldsymbol{A}^{-1}\beta
\end{aligned}$$

Therefore, we can quickly convert between the two and we'll get the same fit since $\boldsymbol{X}_1\beta = \tilde{X}_1\tilde{\beta}$. Why do we need the y intercept? If we don't have one, we are assuming that it is 0. Clearly, if I fit a model and then move everything to the right $\tilde{x} = x + 100$, my model will change if they both have to go through the origin.