The Complete Beginner's Guide to Machine Learning: Multiple Linear Regression in 4 Lines of Code!

Conquer the basics of multiple linear regression (and backward elimination!) and use your data to predict the future!

Anne Bonner Follow

Apr 6 · 19 min read ★

Photo by PublicDomainPictures via Pixabay

Being able to predict the future is awesome.

You might want to predict how well a stock will do based on some other information that you just happen to have.

It might help you to know if how often you bathe and how many cats you have relates to how long you'll live.

You might want to figure out if there's a relationship between a man who 1.) calls his mom more than three times a day, 2.) refers to another man as "bro," 3.) has never done his own laundry and above average divorce rates.

Multiple linear regression might be for you!

GIF via GIPHY

Multiple linear regression is fun because it looks at the relationships within a bunch of information. Instead of just looking at how **one** thing relates to another thing (simple linear regression), you can look at the relationship between a lot of different things and the thing you want to predict.

A **linear regression model** is a statistical model that's frequently used in data science. It's also one of the basic building blocks of machine learning! **Multiple linear regression** (MLR/multiple regression) is a statistical technique. It can use several variables to predict the outcome of a different variable. The **goal of multiple regression is to model the linear relationship between your independent variables and your dependent variable**. It looks at how multiple independent variables are related to a dependent variable.

I'm going to assume that you know a little bit about simple linear regression. If you don't, check out this article on building a simple

linear regressor. It will give you a quick (and fun) walk-through of the basics.

Simple linear regression is what you can use when you have one independent variable and one dependent variable. **Multiple linear regression** is what you can use when you have a bunch of different independent variables!

Multiple regression analysis has three main uses.

- You can look at the strength of the effect of the independent variables on the dependent variable.
- You can use it to ask how much the dependent variable will change if the independent variables are changed.
- You can also use it to predict trends and future values.

Let's do that one!

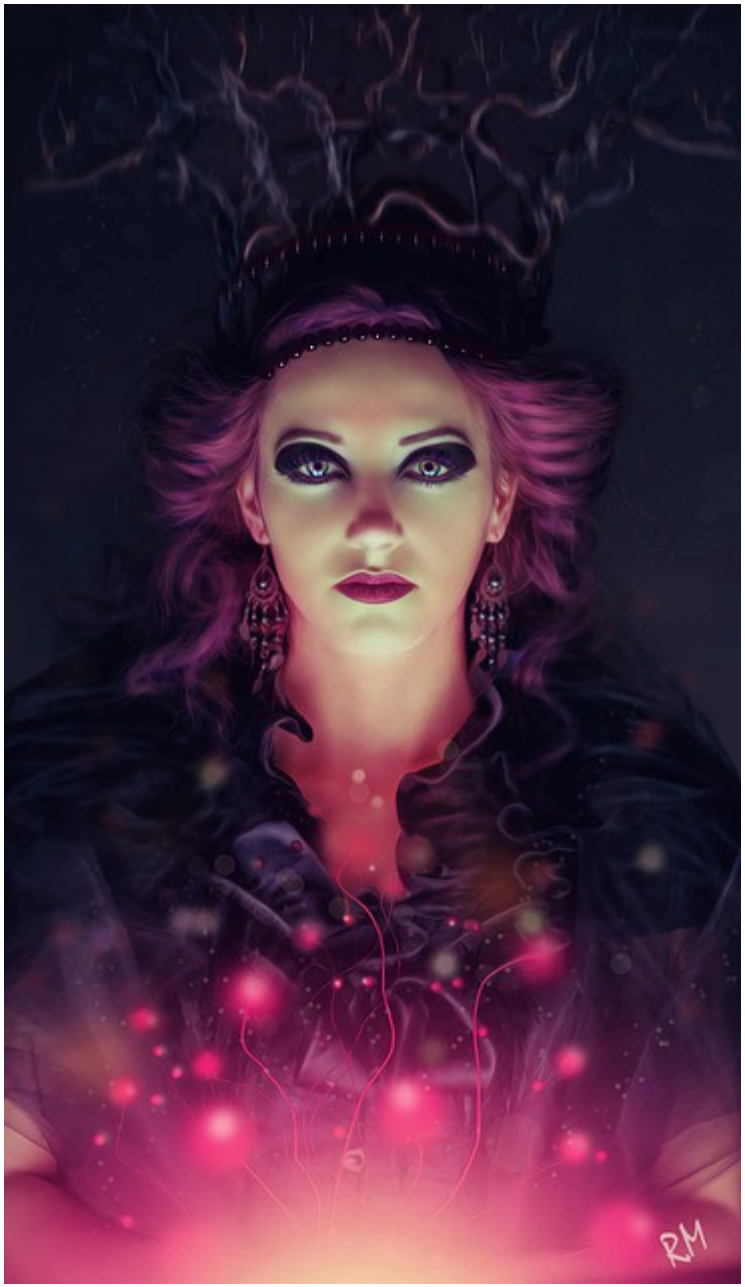


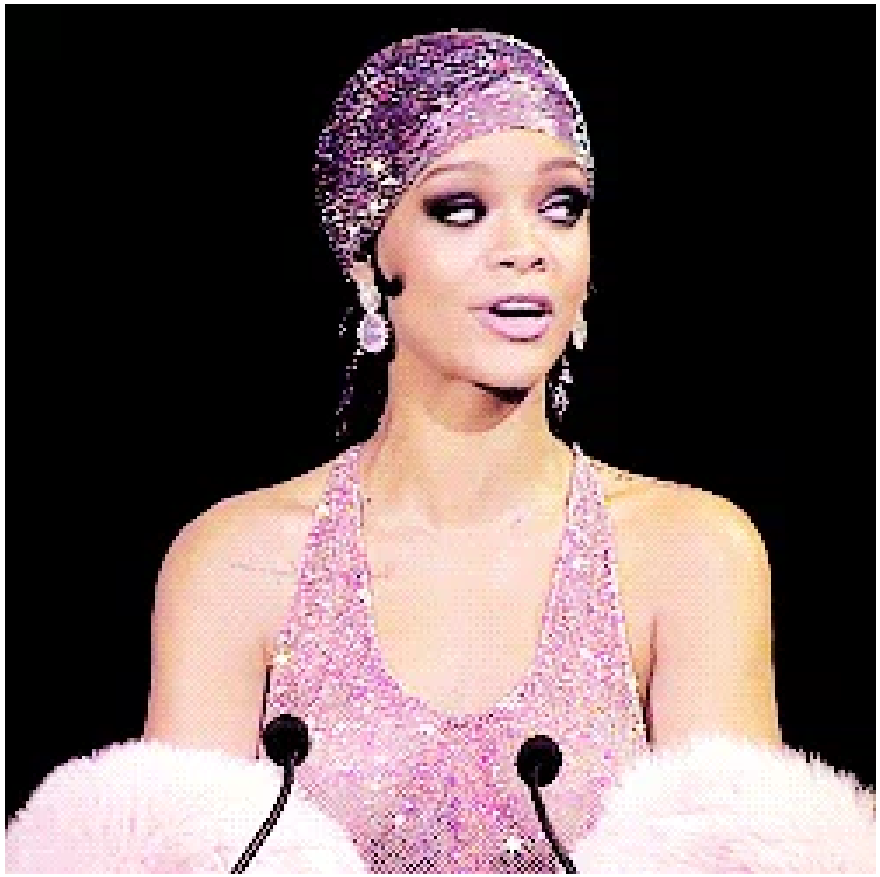
Image by RondellMelling via Pixabay

> *We're going to keep things super simple here so that multiple linear regression as a whole makes sense. I do want you to know that things can get a lot more complex than this in the real world.*

How do I begin?

For the purposes of this post, you are now working for a venture capitalist.

Congratulations!



GIF via GIPHY

So here's the thing: you have a dataset in front of you with information on 50 companies. You have five columns that contain information about how much those companies spend on admin, research and development (R&D), and marketing, their location by state, and their profit for the most recent year. This dataset is anonymized, which means we don't know the names of these companies or any other identifying information.

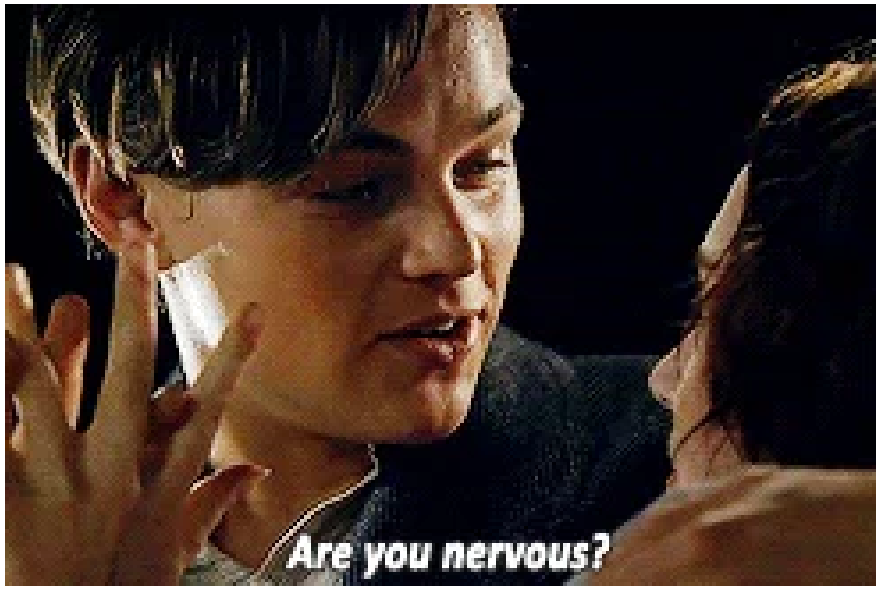
You've been hired to analyze this information and create a model. You need to inform the guy who hired you what kind of companies will make the most sense in the future to invest in. To keep things simple, let's say that your employer wants to make this decision based on last year's profit. This means that the profits column is your **dependent variable**. The other columns are the **independent variables**.

So you want to learn about the **dependent variable** (profit) based on the other categories of information you have.

The guy who hired you doesn't want to invest in these specific companies. He wants to use the information in this dataset as a sample. This sample will help him understand which of the companies he looks at in the future will perform better based on the same information.

Does he want to invest in companies that spend a lot on R&D? Marketing? Does he want to invest in companies that are based in Illinois? You need to help him create a set of guidelines. You're going to help him be able to say something along the lines of, "I'm interested in a company that's based in New York that spends very little on admin expenses but a lot on R&D."

You're going to come up with a model that will allow him to assess where and into which companies he wants to invest to maximize his profit.



GIF via GIPHY

> *Linear regression is great for correlation, but remember that* ***correlation and causation are not the same things!*** *You are not saying that one thing causes the other, you're finding which independent variables are strongly correlated to the dependent variable.*

There are some assumptions that absolutely have to be true:

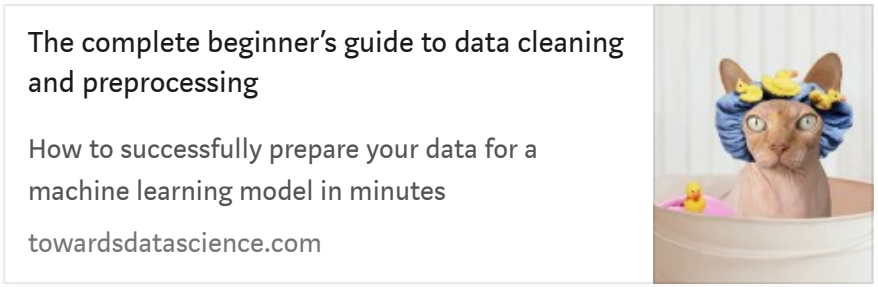
- There is a linear relationship between the dependent variable and the independent variables.
- The independent variables aren't too highly correlated with each other.
- Your observations for the dependent variable are selected independently and at random.

- Regression residuals are normally distributed.

You need to check that these assumptions are true before you proceed and build your model. We're totally skipping past that here. Make sure that if you're doing this in the real world, you aren't just blindly following this tutorial. Those assumptions need to be correct when you're building your regression!

Dummy variables

If you aren't familiar with the concept of dummy variables, check out this article on data cleaning and preprocessing. It has some simple code that we can go ahead and copy and paste here.



So we've already decided that "profit" is our dependent variable (**y**) and the others are our independent variables (**X**). We've also decided that what we want is a linear regression model. What about that column of states? "State" is a categorical variable, not a numerical variable. We need our independent variables to be numbers, not words. What do we do?

Photo by 3dman_eu via Pixabay

Let's create a dummy variable!

If you looked at the information in the locations column, you might see that all of the companies that are being examined are based in two states. For the purposes of this explanation, let's say all of our companies are located in either New York or Minnesota. That means that we'll want to turn this one column of information into two columns of 1s and 0s. (If you want to learn more about why we're doing that, check out that article on simple linear regression. It explains why this would be the best way to arrange our data.)

So how do we populate those columns? Basically, we'll turn each state into its own column. If a company is located in New York, it will have a **1** in the "New York" column and a **0** in the "Minnesota" column. If you were using more states, you'd have a 1 in the New York column, and,

for example, a 0 in the "California" column, a zero in the "Illinois" column, a 0 in the Arkansas column, and so on. We won't be using the original "locations" column anymore because we won't need it!

These 1s and 0s are basically working as a light switch. 1 is "on" or "yes" and 0 is "off" or "nope."

Beware the dummy variable trap

You never want to include both variables at the same time.

Why is that?

You'd be duplicating a variable. The first variable (d1) is always equal to 1 minus the second variable (d2). (**d1 = 1-d2**) When one variable predicts another, it's called **multicollinearity**. As a result, the model wouldn't be able to distinguish the results of d1 from the results of d2. You can't have the constant and both dummy variables at the same time. If you have nine variables, include eight of them. (If you have two sets of dummy variables, then you have to do this for each set.)

What is the P-value?

You're going to want to be familiar with the concept of a P-value. That's definitely going to come up.

The P-value is the probability of getting a sample like ours (or more extreme than ours) if the null hypothesis is true.

It gives a value to the weirdness of your sample. If you have a large P-value, then you probably won't change your mind about the null hypothesis. A large value means that it wouldn't be at all surprising to

get a sample like yours if the hypothesis is true. As the P-value gets smaller, you should probably start to ask yourself some questions. You might want to change your mind and maybe even reject the hypothesis.

> *The **null hypothesis** is the official way to refer to the claim (hypothesis) that's on trial here. It's the default position where there's just no association among the groups that are being tested. In every experiment, you're looking for an effect among the groups that are being tested. Unfortunately, there's always the possibility that there's no effect (or no difference) between the groups. That lack of difference is called the null hypothesis.*

> *It's like if you were doing a trial of a drug that doesn't work. In that trial, there just wouldn't be a difference between the group that took the drug and the rest of the population. The difference would be null.*

> *You always assume that the null hypothesis is true until you have evidence that it isn't.*

Let's keep moving!

We need to figure out which columns we want to keep and which we want to toss. If you just chuck a bunch of stuff into your model, it won't be a good one. It definitely won't be reliable! (Also, at the end of the day, you need to be able to explain your model to the guy who hired you to create this thing. You're only going to want to explain the variables that actually predict something!)

There are essentially five methods of building a multiple linear regression model.

1. Chuck Everything In and Hope for the Best
2. Backward Elimination
3. Forward Selection
4. Bidirectional Elimination
5. Score Comparison

You'll almost certainly hear about Stepwise Regression as well. Stepwise regression is most commonly used as another way of saying bidirectional elimination (method 4). Sometimes when people use that phrase they're referring to a combination of methods 2, 3, and 4. (That's the idea behind bidirectional elimination as well.)

Method 1 (Chuck Everything In): Okay. That isn't the official name for this method (but it should be). Occasionally you'll need to build a model where you just throw in all your variables. You might have some kind of prior knowledge. You might have a particular framework you need to use. You might have been hired by someone who's insisting that you do that. You might want to prepare for backward elimination. It's a real option, so I'm including it here.

Method 2 (backward elimination): This has a few basic steps.

1. First, you'll need to set a significance level for which data will stay in the model. For example, you might want to set a significance

level of 5% (SL = 0.05). This is important and can have real ramifications, so give it some thought.

2. Next, you'll fit the full model with all possible predictors.
3. You'll consider the predictor with the highest P-value. If your P-value is greater than your significance level, you'll move to step four, otherwise, you're done!
4. Remove that predictor with the highest P-value.
5. Fit the model without that predictor variable. If you just remove the variable, you need to refit and rebuild the model. The coefficients and constants will be different. When you remove one, it affects the others.
6. Go back to step 3, do it all over, and keep doing that until you come to a point where even the highest P-value is < SL. Now your model is ready. All of the variables that are left are less than the significance level.

(After we go through these concepts, I'll walk you through an example of backward elimination so you can see it in action! It's definitely confusing, but if you really look at what's going on, you'll get the hang of it.)

Method 3 (forward selection): This is way more complex than just reversing backward elimination.

1. Choose your significance level (SL = 0.05).
2. Fit all possible simple regression models and select the one with the lowest P-value.

3. Keep this variable and fit all possible models with one extra predictor added to the one you already have. If we selected a simple linear regressor with one variable, now we'd select all of them with two variables. That means all possible two variable linear regressions.

4. Find the predictor with the lowest P-value. If P < Sl, go back to step 3. Otherwise, you're done!

We can stop when P<SL is no longer true, or there are no more P-values that are less than the significance level. It means that the variable is not significant anymore. **You won't keep the current model, though**. You'll keep the previous one because, in the final model, your variable is insignificant.

Method 3 (bidirectional elimination): This method combines the previous two!

1. Select a significance level to enter and a significance level to stay (SLENTER = 0.05, SLSTAY = 0.05).

2. Perform the next step of forward selection where you add the new variable. You need to have your P-value be less than SLENTER.

3. Now perform all of the steps of backward elimination. The variables must have a P-value less than SLSTAY in order to stay.

4. Now head back to step two, then move forward to step 3, and so on until no new variables can enter and no new variables can exit.

You're done!

Method 4 (score comparison): Here, you're going to be looking at all possible methods. You'll look at a comparison of the scores for all of the possible methods. This is definitely the most resource-consuming approach!

1. Select a criterion of goodness of fit (for example, Akaike criterion)
2. Construct all possible regression models
3. Select the one with the best criterion

Fun fact: if you have 10 columns of data, you'll wind up with 1,023 models here. You'd better be ready to commit if you're going to go this route!

Ummm, what?

If you're just getting started with machine learning, statistics, or data science, that all looks like it will be an insane amount of code. It's not!

So much of what you need to do with a machine learning model is all ready to go with the amazing libraries out there. You'll need to do the tough parts where you decide what information is important and what kind of models you'll want to use. It's also up to you to interpret the results and be able to communicate what you've built. However, the code itself is very doable.

GIF via GIPHY

Let me show you!

Backward elimination is the fastest and the best method to start with, so that's what I'm going to walk you through after we build the quick and easy multiple linear regression model.

First, let's prepare our dataset. Let's say we have a .csv file called "startups.csv" that contains the information we talked about earlier. We'll say it has 50 companies and columns for R&D spending, admin spending, marketing spending, what state the company is located in (let's say, New York, Minnesota, and California), and one column for last year's profit.

It's a good idea to import your libraries right away.

```
# Importing the libraries
import numpy as np
```

```
import matplotlib.pyplot as plt
import pandas as pd
```

Now we can go ahead and copy and paste the code from that data cleaning and preparation article! We're definitely going to want to change the name of our dataset to ours. I'm calling it 'startups.csv.' We'll adjust a couple of other tiny details as well. Profit (y) is still our last column, so we'll continue to remove that with [:, :-1]. We'll make a little adjustment to grab our independent variables with [:, 4]. Now we have a vector of the dependent variable (y) and a matrix of independent variables that contains everything except the profits (X). We want to see if there is a linear dependency between the two!

```
dataset = pd.read_csv('startups.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
```

Now we need to encode the categorical variable. We can use label encoder and one hot encoder to create dummy variables. (We can copy and paste this from that other article too! Make sure you're grabbing the right information and you don't encode the dependent variable.) You're going to change the index of the column in both spots [:, 3] and [:, 3] again, and replace the index in one hot encoder too [3].

```
from sklearn.preprocessing import LabelEncoder,
OneHotEncoder
labelencoder = LabelEncoder()
```

```
X[:, 3] = labelencoder.fit_transform(X[:, 3])
onehotencoder = OneHotEncoder(categorical_features = [3])
X = onehotencoder.fit_transform(X).toarray()
```

You're ready to go! Our one column of information is now three columns, each of which corresponds to one state!

What about avoiding the dummy variable trap? You don't actually need to do that with our libraries! It's all taken care of for you here with the libraries that we're choosing to use. However, if you ever want or need to run that code, it's simple! You can do that with one line right after you encode your data.

```
X=X[:, 1:]
```

What does that do? It removes the first column from X. Putting the **1** there means that we want to take all of the columns starting at index 1 to the end. You won't take the first column. For some libraries, you'll need to take one column away manually to be sure your dataset won't contain redundancies.

Now let's split our training and testing data. The most common split is an 80/20 split, which means 80% of our data would go to training our model and 20% would go to testing it. Let's do that here!

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
  test_size = 0.2, random_state = 0)
```

What about feature scaling?

We don't need to do feature scaling here! The library will take care of that for us.

Photo by Gift Habeshaw on Unsplash

Multiple linear regression time!

We'll import linear regression from Scikit-Learn. (That makes a little sense, doesn't it?)

```
from sklearn.linear_model import LinearRegression
```

Now we'll introduce our regressor. We'll create an object of the class LinearRegression and we'll fit the object to our training set. We want to apply this to both our X_train and y_train.

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Now let's test the performance of our multiple linear regressor!

(We won't plot a graph here because we'd need five dimensions to do that. If you're interested in plotting a graph with a simple linear regressor, check out this article on building a simple linear regressor.)

We'll create the vector of predictions (y_pred). We can use the regressor with the predict method to predict the observations of the test set (X_test).

```
y_pred = regressor.predict(X_test)
```

That's it! Four lines of code and you've built a multiple linear regressor!

GIF via GIPHY

Now we can see the ten predicted profits! You can print them any time with a simple `print(y_pred)` . We can easily compare them by taking a look at the predictions and then comparing them to the actual results. If you were to take a look, you'd see that some are incredibly accurate and the rest are pretty darn good. Nice work!

There is definitely some linear dependency between our dependent and independent variables. We can clearly see a strong linear relationship

between the two.

Congratulations!! You now know how to make a multiple linear regressor in Python!

Want to keep going?

Things are about to get more challenging!

What if some of the variables have a lot of impact on our dependent variable and some are statistically insignificant? We can definitely find out which are the variables that have the highest impact on the dependent variable. We'll want to find a team of variables that all have a definite effect, positive or negative.

Let's use **backward elimination**!

We need to prepare something specific for backward elimination. We want a library stats model, so let's import statsmodels.formula.api. That's a little long to have to keep retyping, so we'll make a shortcut using sm.

```
import statsmodels.formula.api as sm
```

We need to add a column of ones in our matrix of features of independent variables because of the way it works with the constant. (Our model needs to take into account our constant b0. In most libraries it's included, but not in the stats model that we're using. We'll

add a column of ones so our stats model will understand the formula correctly.)

This starts pretty simply. We'll use .append because we want to append.

(Love Python ❤)

We have our matrix of features X. The values argument is perfect for us because it's an array. We'll input a matrix of 50 lines and one column with 1s inside. We can create that with Numpy's np.ones. We'll need to specify the numbers of lines and columns we want (50,1). We need to convert the array into the integer type to make this work, so we'll use .astype(int). Then we need to decide if we're adding a line or a column (line = 0, column = 1), so we'll say axis = 1 for a column!

We want this column to be located at the beginning of our dataset. What do we do? Let's add matrix X to the column of 50 ones, rather than the other way around. We can do that with values = X.

```
X = np.append(arr = np.ones((50, 1)).astype(int), values =
X, axis = 1)
```

Let's do this!

We want to create a new matrix of our optimal features (X_opt). These features are the ones that are statistically significant. The ones that have a high impact on the profit. This will be the matrix containing the team of optimal features with high impact on the profit.

We'll need to initialize it. We can remove the variables that are not statistically significant one by one. We'll do this by removing the index at each step. First take all the indexes of the columns in X, separated by commas [0,1,2,3,4,5].

If you look back at the methods earlier, you'll see that we first need to select our significance level, which we talked about earlier. Then we need to fit the model!

We aren't going to take the regressor we built. We're using a new library, so now we need a new fit to our future optimal matrix. We'll create a new regressor (our last one was from the linear regression library). Our new class will be ordinary least squares (OLS). We'll need to call the class and specify some arguments. (You can check out the official documentation here.) For our arguments, we'll need an endog (our dependent variable) and an exog (our X_opt, which is just our matrix of features (X) with the intercept, which isn't included by default). In order to fit it we'll just use a .fit()!

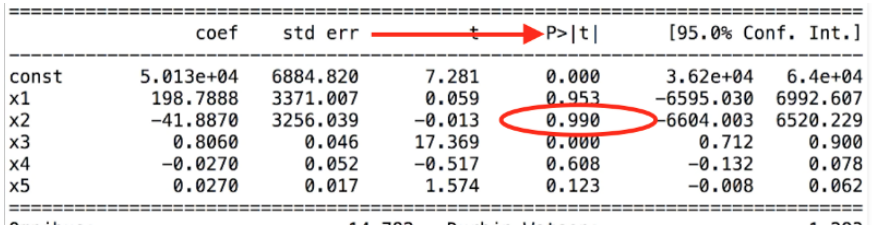
```
X_opt = X[:, [0, 1, 2, 3, 4, 5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
```

Now we've initialized X_opt!

Now let's look at our P-values! How do we look for the predictor with the highest P-values? We'll take our regressor object and call the function .summary().

```
regressor_OLS.summary()
```

Now we can see a table with some very useful information about our model! We can see the adjusted R-squared values and our P-values. The lower the p-value, the more significant your independent variable will be with respect to your dependent variable. Here, we're looking for the highest one. That's easy to see.



	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
const	5.013e+04	6884.820	7.281	0.000	3.62e+04	6.4e+04
x1	198.7888	3371.007	0.059	0.953	-6595.030	6992.607
x2	-41.8870	3256.039	-0.013	0.990	-6604.003	6520.229
x3	0.8060	0.046	17.369	0.000	0.712	0.900
x4	-0.0270	0.052	-0.517	0.608	-0.132	0.078
x5	0.0270	0.017	1.574	0.123	-0.008	0.062

Now let's remove it!

We can copy and paste our code from above and remove index 2. That will look like this:

```
X_opt = X[:, [0, 1, 3, 4, 5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
```

Just keep going until you don't have any P-values that are higher than the SL value you chose. Remember that you always want to look at the

original matrix in order to choose the correct index! You're using the columns in your original matrix (X), not in X_opt.

You might get to the point where you have a P-value that's incredibly close to the SL value that you chose. For example, we chose 0.050 and here's 0.060.

const	4.698e+04	2689.933	17.464	0.000	4.16e+04	5.24e+04
x1	0.7966	0.041	19.266	0.000	0.713	0.880
x2	0.0299	0.016	1.927	0.060	-0.001	0.061

That's a tough situation because the value that you chose could have been anything. If you want to thoroughly follow your framework, you'll need to remove that index. But there are other metrics that can help make more sense of whether or not we want to do that. We could add other metrics, like a criterion, that can help us decide if we really want to make that choice. There's also a lot of information right in the summary here, like the R-squared value, that can help us make our decision.

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.948
Model:	OLS	**Adj. R-squared:**	0.943
Method:	Least Squares	**F-statistic:**	205.0
Date:	Fri, 05 Apr 2019	**Prob (F-statistic):**	2.90e-28
Time:	13:31:15	**Log-Likelihood:**	-526.75
No. Observations:	50	**AIC:**	1064.
Df Residuals:	45	**BIC:**	1073.
Df Model:	4		
Covariance Type:	nonrobust		

So let's say we ran backward elimination until the end and we're left with only the index for the R&D spending column.

```
X_opt = X[:, [0, 1, 3, 4, 5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [0, 1, 3, 5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [0, 3, 5]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
X_opt = X[:, [0, 3]]
regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
regressor_OLS.summary()
```

If we've been following our model carefully, that means that we now know that R&D spending is a powerful predictor for our dependent variable! The conclusion here is that the data that can predict profits with the highest impact is composed of only one category: R&D spending!

	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
const	4.903e+04	2537.897	19.320	0.000	4.39e+04	5.41e+04
x1	0.8543	0.029	29.151	0.000	0.795	0.913

You did it! You used multiple linear regression and backward elimination! You figured out that looking at R&D spending will give you the best sense of what a company's profits will be!

You're amazing!

GIF via GIPHY

As always, if you're doing anything cool with this information, let people know about it in the responses below or reach out any time on LinkedIn @annebonnerdata!

You might want to check out some of these articles too:

Getting started with Git and GitHub: the complete beginner's guide

Git and GitHub basics for the curious and completely confused (plus the easiest way to...

towardsdatascience.com

How to effortlessly create a website for free with GitHub

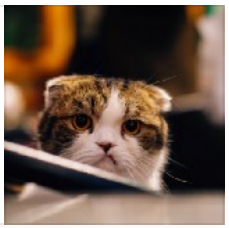
Getting started with GitHub Pages: the unbelievably quick and easy guide for creating a...

towardsdatascience.com

Getting Started With Google Colab

A Basic Tutorial for the Frustrated and Confused

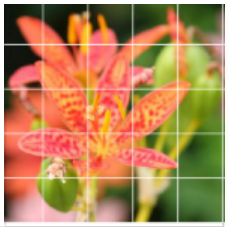
towardsdatascience.com



How to build an image classifier with greater than 97% accuracy

A clear and complete blueprint for success

medium.freecodecamp.org



Thanks for reading!