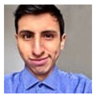
towardsdatascience.com

A Simple Guide to Linear Regression using Python - Towards Data Science

The PyCoach

11–13 minutes

Learn the core concepts of machine learning while building a linear regression model in Python



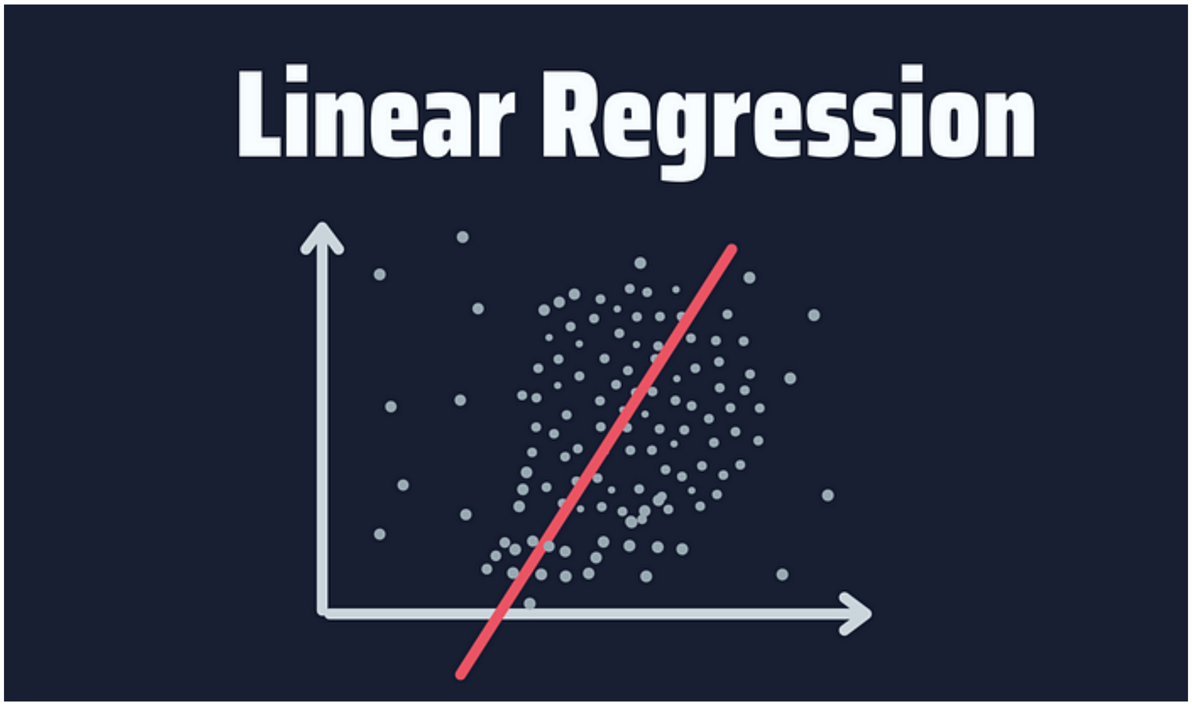


Image by author

One of the first machine learning algorithms every data scientist should learn is linear regression. This simple model helps us grasp core concepts of machine learning such as recognizing the dependent and independent variables, building a model, and understand the math and statistics behind a model.

There are 2 common ways to make linear regression in Python — using the statsmodel and sklearn libraries. Both are great options and have their pros and cons.

In this guide, I will show you how to make a linear regression using both of them, and also we will learn all the core concepts behind a linear regression model.

Table of Contents

What is Linear Regression?

Linear regression is an approach for modeling the relationship between two (simple linear regression) or more variables (multiple linear regression). In simple linear regression, one variable is considered the predictor or independent variable, while the other variable is viewed as the outcome or dependent variable.

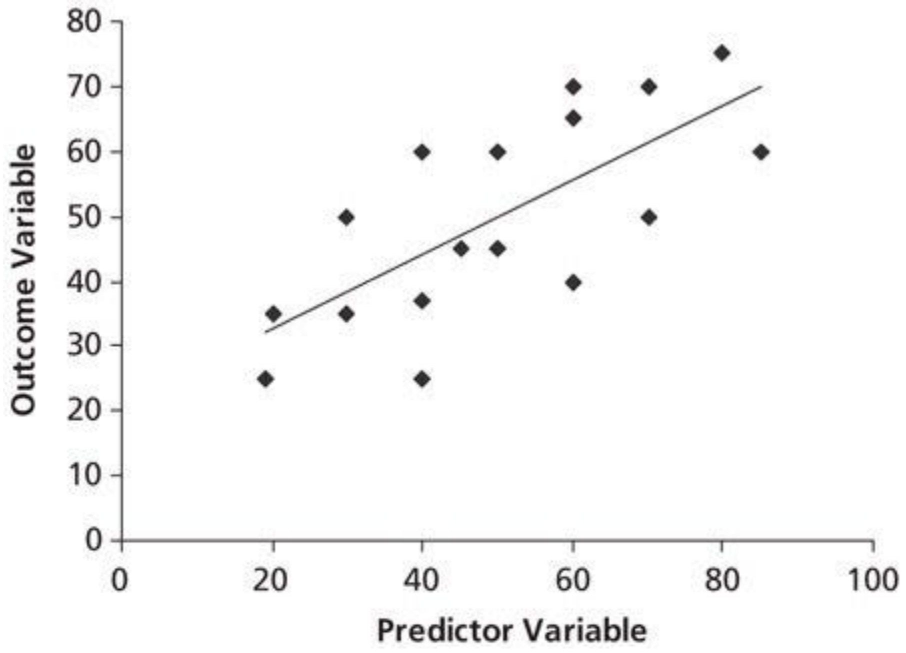
Here’s the linear regression equation:

$$Y = b_0 + b_1x_1 + b_2x_2 + ... + b_nx_n$$

[illegible]

where y is the dependent variable (target value), `x1, x2, … xn` the independent variable (predictors), `b0` the intercept, `b1, b2, ...` bn the coefficients and n the number of observations.

If the equation isn't clear, the picture below might help.



Credit: [Quora]

In the picture, you can see a linear relationship. That is, if one independent variable increases or decreases, the dependent variable will also increase or decrease.

Linear regression can be used to make simple predictions such as predicting exams scores based on the number of hours studied, the salary of an employee based on years of experience, and so on.

Enough theory! Let's learn how to make a linear regression in Python.

Linear Regression in Python

There are different ways to make linear regression in Python. The 2 most popular options are using the statsmodels and scikit-learn libraries.

First, let’s have a look at the data we’re going to use to create a linear model.

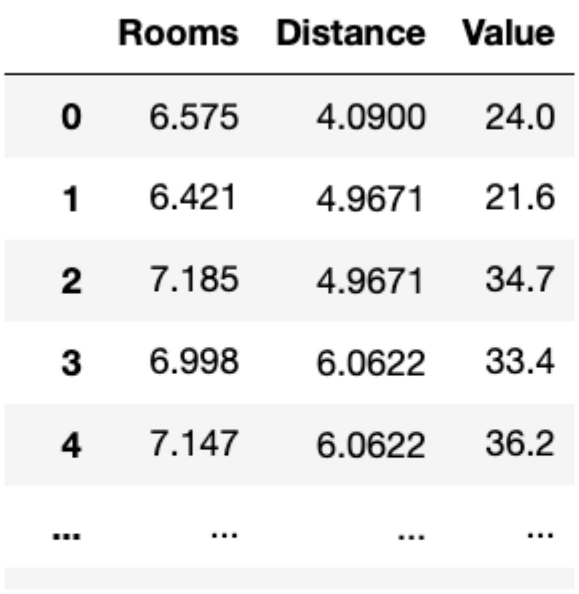
The Data

To make a linear regression in Python, we’re going to use a dataset that contains Boston house prices. The original dataset comes from the sklearn library, but I simplified it, so we can focus on building our first linear regression.

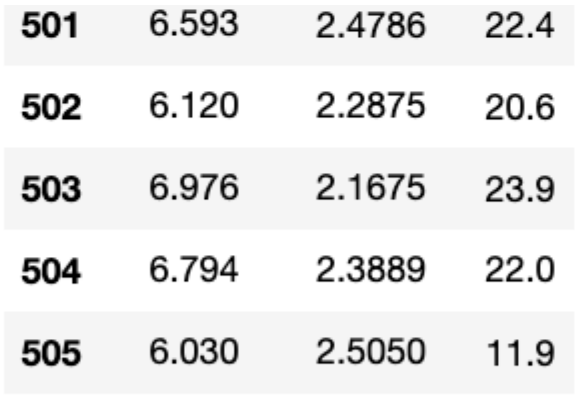
You can download this dataset on my Github or on Google Drive. Make sure to leave this CSV file in the same directory where your Python script is located.

Let’s have a look at this dataset. To do so, import pandas and run the code below.

```
import pandas as pd
df_boston = pd.read_csv('Boston House Prices.csv')
df_boston
```



	Rooms	Distance	Value
0	6.575	4.0900	24.0
1	6.421	4.9671	21.6
2	7.185	4.9671	34.7
3	6.998	6.0622	33.4
4	7.147	6.0622	36.2
...	...	...	...



501	6.593	2.4786	22.4
502	6.120	2.2875	20.6
503	6.976	2.1675	23.9
504	6.794	2.3889	22.0
505	6.030	2.5050	11.9

Image by author

There are 3 columns. The "Value" column contains the median value of owner-occupied homes in $1000's (this is what we want to predict, that is, our target value). The "Rooms" and "Distance" columns contain the average number of rooms per dwelling and weighted distances to five Boston employment centers (both are the predictors)

To sum it up, we want to predict home values based on the number of rooms a home has and its distance to employment centers.

Linear Regression with Statsmodels

Statsmodels is a module that helps us conduct statistical tests and estimate models. It provides an extensive list of results for each estimator.

If you have installed Python through Anaconda, you already have statsmodels installed. If not, you can install it either with conda or pip.

```
# pip
pip install statsmodels# conda
conda install -c conda-forge statsmodels
```

Once you have statsmodel installed, import it with the following line of code.

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

The first thing to do before creating a linear regression is to define the dependent and independent variables. We've already discussed them in the previous section. The dependent variable is the value we want to predict and is also known as the target value. On the other hand, the independent variable(s) is the predictor.

In our dataset we have 2 predictors, so we can use any or both of them.

Let's start with a simple linear regression. A simple linear regression estimates the relationship between **one independent variable** and **one dependent variable.**

Simple Linear Regression

For this example, I'll choose "Rooms" as our predictor/independent variable.

- Dependent variable: "Value"
- Independent variable: "Rooms"

Let's define the dependent and independent variables in our code as well.

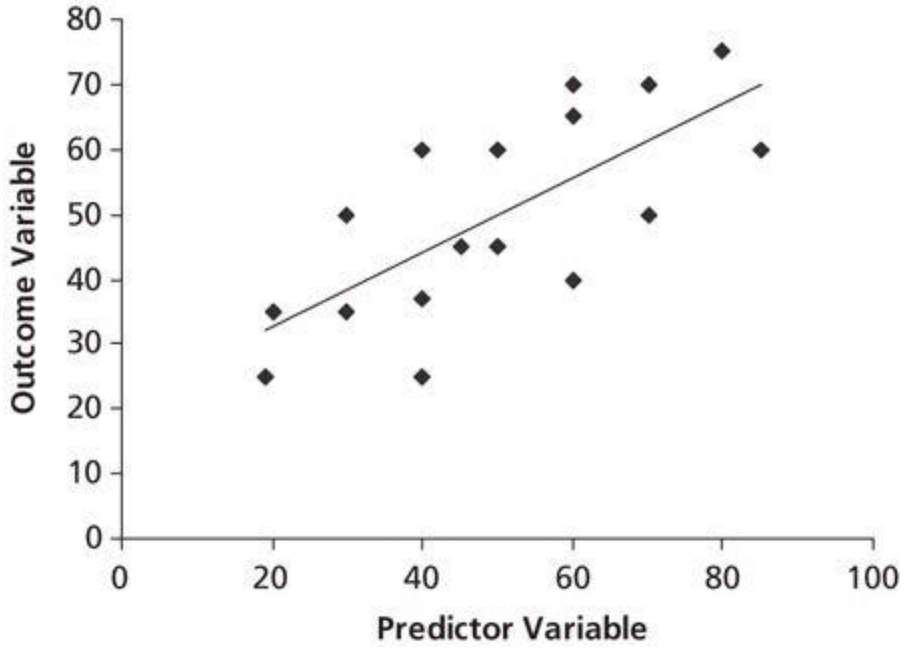
```
y = df_boston['Value'] # dependent variable
x = df_boston['Rooms'] # independent variable
```

Throughout this guide, I'll be using linear algebra notation — lower case letters will be used for vectors and upper case letters will be used for matrices.

Fitting the model

Now it’s time to fit the model. To explain to you what fitting a model means, consider the following generic equation used for simple linear regression.

$y = ax + b$



Credit: Quora

Fitting the model means finding the optimal values of a and b, so we obtain a line that best fits the data points. A model that is well-fitted produces more accurate outcomes, so only after fitting the model, we can predict the target value using the predictors.

Now let’s fit a model using statsmodels. First, we add a constant before fitting a model (sklearn adds it by default) and then we fit the model using the .fit() method.

```
x = sm.add_constant(x1) # adding a constant
lm = sm.OLS(y,x).fit() # fitting the model
```

“lm” stands for linear model and represents our fitted model. This variable will help us predict our target value.

```
>>> lm.predict(x)0      25.232623
1      24.305975
2      31.030253
3      29.919727
4      31.231138
         ...
501    24.603318
502    20.346831
503    27.822178
504    26.328552
505    19.661029
```

The code above predicts home values (printed output) based on the data inside the “Room” column.

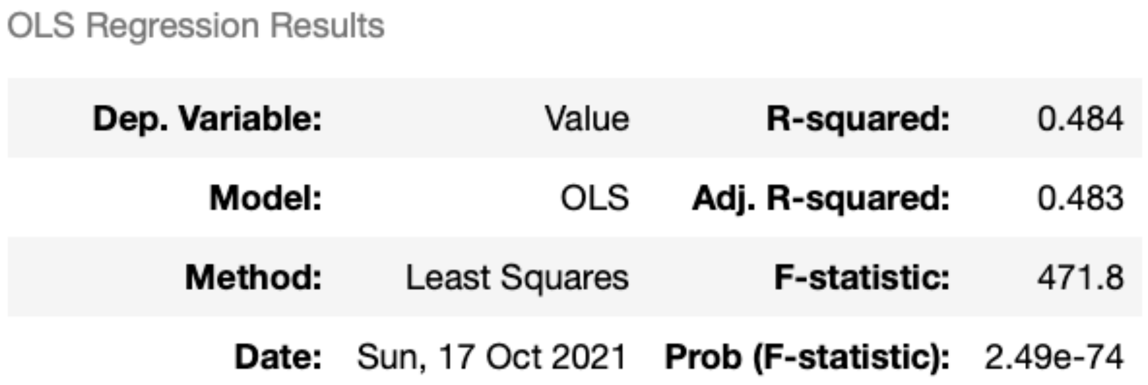
The Regression Table

Although we could predict the target values, the analysis isn’t done yet. We need to know how this linear model performs. The regression table can help us with that. This table provides an extensive list of results that reveal how good/bad is our model.

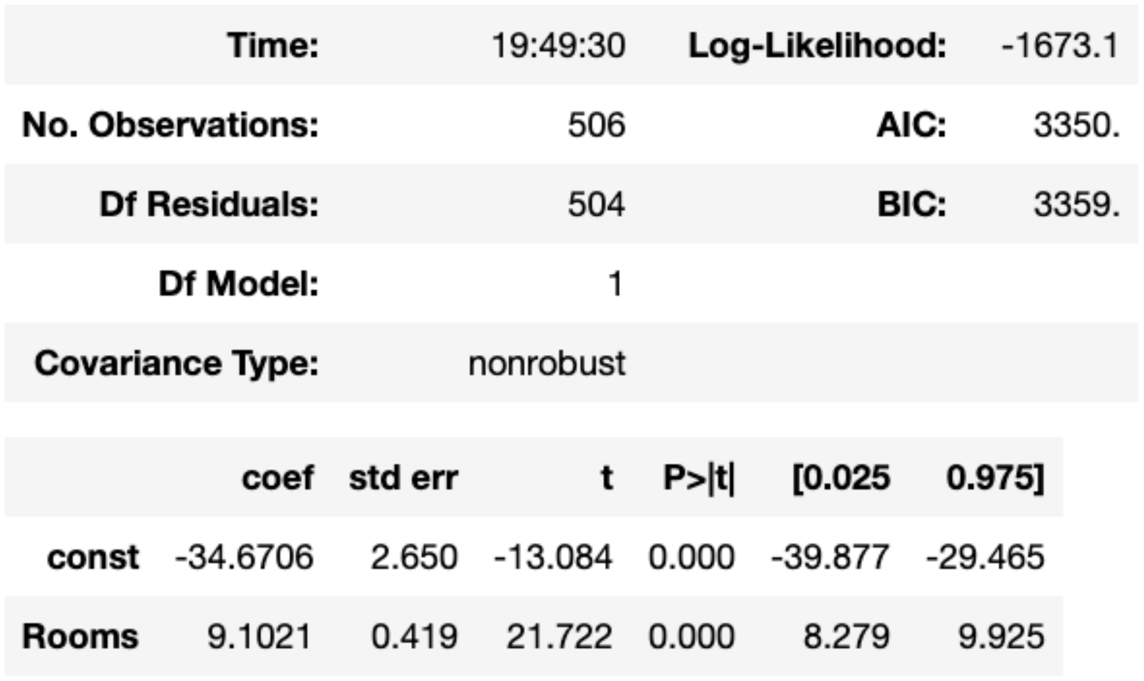
To obtain the regression table run the code below:

```
lm.summary()
```

You will obtain this table:


OLS Regression Results

Dep. Variable:	Value	**R-squared:**	0.484
Model:	OLS	**Adj. R-squared:**	0.483
Method:	Least Squares	**F-statistic:**	471.8
Date:	Sun, 17 Oct 2021	**Prob (F-statistic):**	2.49e-74



Time:	19:49:30	**Log-Likelihood:**	-1673.1
No. Observations:	506	**AIC:**	3350.
Df Residuals:	504	**BIC:**	3359.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-34.6706	2.650	-13.084	0.000	-39.877	-29.465
Rooms	9.1021	0.419	21.722	0.000	8.279	9.925

Image by author

The table is titled “OLS Regression Results.” OLS stands for Ordinary Least Squares and this is the most common method to estimate linear regression.

Let’s have a look at some important results in the first and second tables.

- Dep. Variable: This is the dependent variable (in our example “Value” is our target value)
- R-squared: Takes values from 0 to 1. R-squared values close to 0 correspond to a regression that explains none of the variability of the data, while values close to 1 correspond to a regression that explains the entire variability of the data. **The r-squared obtained is telling us that the number of rooms explains 48.4% of the variability in house values.**
- Coef: These are the coefficients (a, b) we’ve seen in the model equation before.

- Std error: Represents the accuracy of the prediction. The lower the standard error, the better prediction.
- t, P>t (p-value): The t scores and p-values are used for hypothesis test. The “Rooms” variable has a statistically significant p-value. Also, we can say at a 95% percent confidence level that the value of “Rooms” is between 8.279 to 9.925.

Linear Regression Equation

From the table above, let’s use the coefficients (coef) to create the linear equation and then plot the regression line with the data points.

```
# Rooms coef: 9.1021
# Constant coef: - 34.6706# Linear equation: 𝑦 = 𝑎𝑥 + 𝑏
y_pred = 9.1021 * x['Rooms'] - 34.6706
```

where y_pred (also known as yhat) is the predicted value of y (the dependent variable) in the regression equation.

Linear Regression Plot

To plot the equation let’s use seaborn.

```
import seaborn as sns
import matplotlib.pyplot as plt# plotting the data points
sns.scatterplot(x=x['Rooms'], y=y)#plotting the line
sns.lineplot(x=x['Rooms'],y=y_pred, color='red')#axes
plt.xlim(0)
plt.ylim(0)
plt.show()
```

The code above produces the following plot.

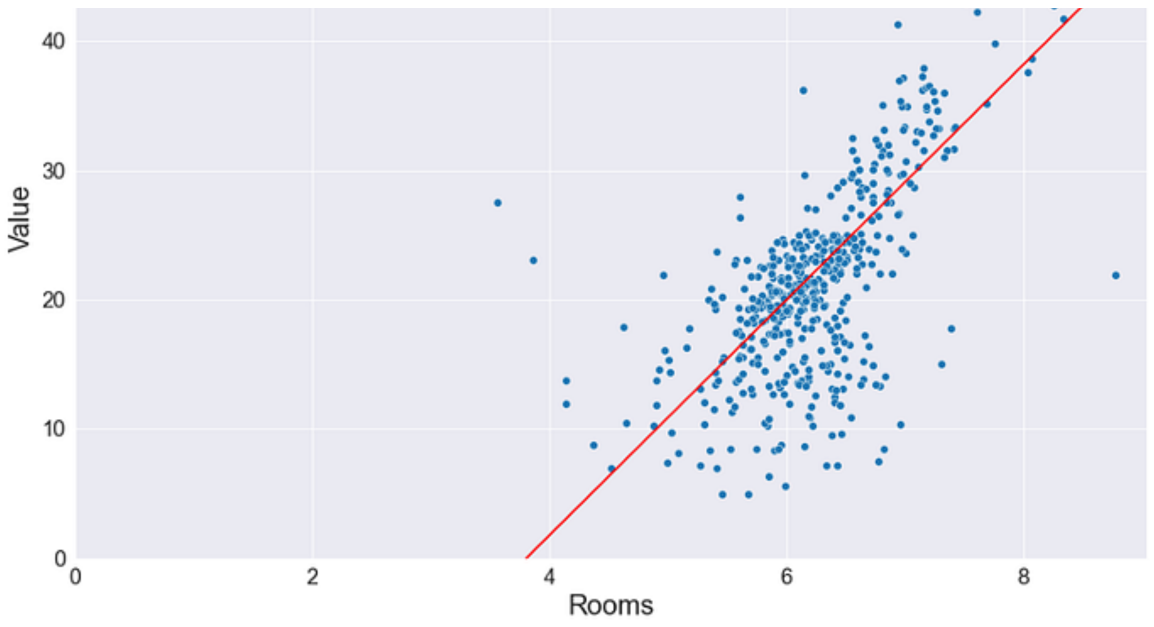


Image by author

The red plot is the linear regression we built using Python. We can say that this is the line that best fits the blue data points.

Congratulation! You've just built your first simple linear regression in Python. If you're up for a challenge, check how to make a multiple linear regression.

Multiple Linear Regression

Now that you already know the core concepts of linear regression, we can easily create a multiple linear regression.

Let's start by setting the dependent and independent variables. In this case, we're going to use 2 independent variables.

- Dependent variable: "Value"
- Independent variable: "Rooms" and "Distance"

Let's define the dependent and independent variables in our code as well.

y = df_boston['Value'] # dependent variable

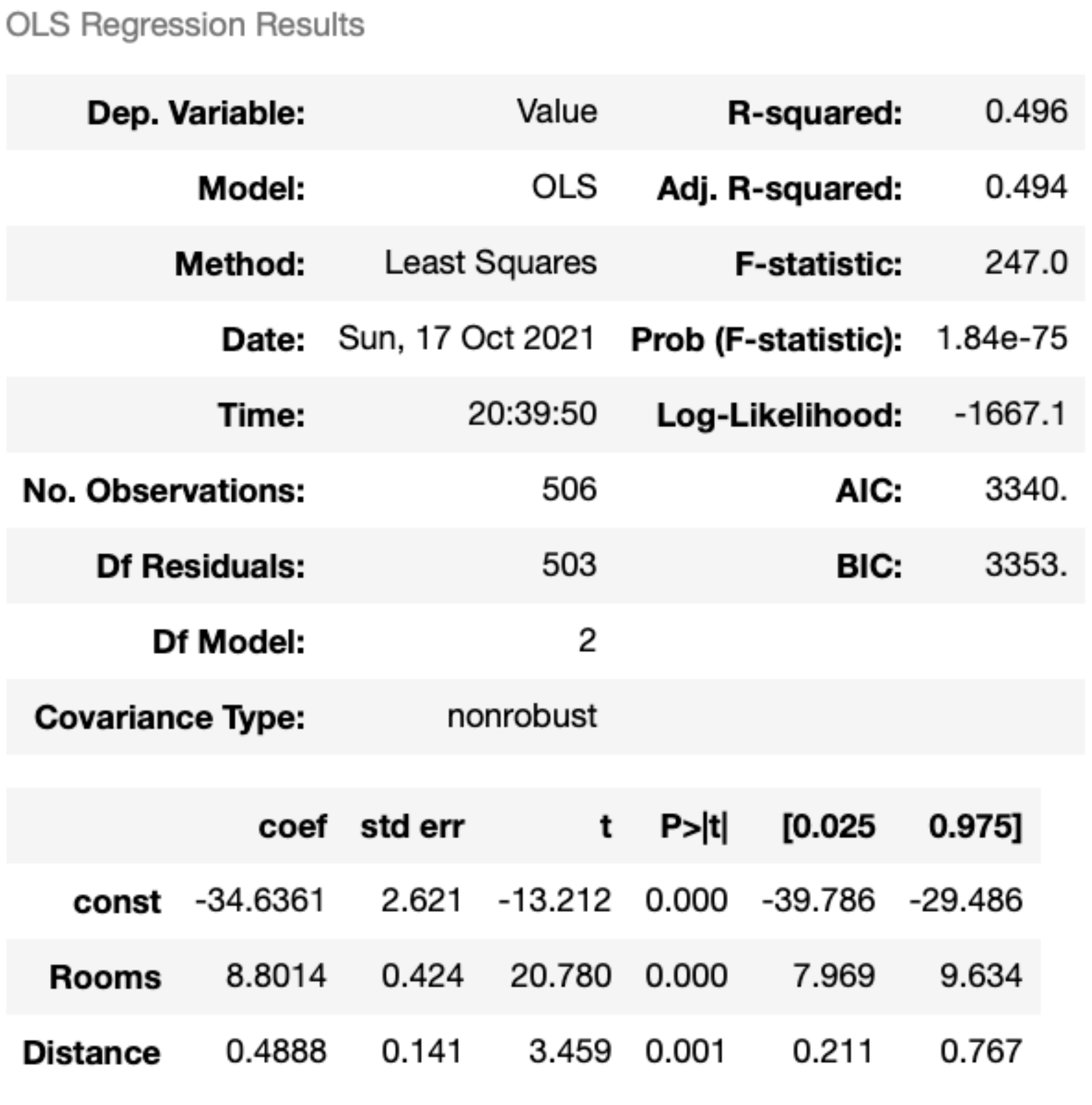
```
X = df_boston[['Rooms', 'Distance']] # independent variable
```

Now let's add a constant and fit the model.

```
X = sm.add_constant(X) # adding a constant
lm = sm.OLS(y, X).fit() # fitting the model
```

Let's have a look at the results.

```
lm.summary()
```



OLS Regression Results

Dep. Variable:	Value	**R-squared:**	0.496
Model:	OLS	**Adj. R-squared:**	0.494
Method:	Least Squares	**F-statistic:**	247.0
Date:	Sun, 17 Oct 2021	**Prob (F-statistic):**	1.84e-75
Time:	20:39:50	**Log-Likelihood:**	-1667.1
No. Observations:	506	**AIC:**	3340.
Df Residuals:	503	**BIC:**	3353.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-34.6361	2.621	-13.212	0.000	-39.786	-29.486
Rooms	8.8014	0.424	20.780	0.000	7.969	9.634
Distance	0.4888	0.141	3.459	0.001	0.211	0.767

Image by author

The r-squared increased a bit. Also, there's a new line in the second table that represents the parameters for the "Distance" variable. The analysis of this table is similar to the simple linear regression, but if you have any questions, feel free to let me know in the comment section.

Linear Regression with sklearn

Scikit-learn is the standard machine learning library in Python and it can also help us make either a simple linear regression or a multiple linear regression.

Since we deeply analyzed the simple linear regression using statsmodels before, now let's make a multiple linear regression with sklearn.

First, let's install sklearn. If you have installed Python through Anaconda, you already have sklearn installed. If not, you can install it either with conda or pip.

```
# pip
pip install scikit-learn# conda
conda install -c conda-forge scikit-learn
```

Now let's import linear_model from the sklearn library.

from sklearn import linear_model

The dependent and independent variables will be the following.

```
y = df_boston['Value'] # dependent variable
X = df_boston[['Rooms', 'Distance']] # independent variable
```

Now we have to fit the model (note that the order of arguments in the fit method using sklearn is different from statsmodels)

```
lm = linear_model.LinearRegression()
lm.fit(X, y) # fitting the model
```

Similarly to statsmodels, we use the predict method to predict the target value in sklearn.

```
lm.predict(X)
```

However, unlike statsmodels we don't get a summary table using `.summary()`. Instead, we have to call each element one by one.

```
>>> lm.score(X, y)
0.495>>> lm.coef_
array([8.80141183, 0.48884854])>>> lm.intercept_
-34.636050175473315
```

The results are the same as the table we obtained with statsmodels.

Note that we didn't split the data into training and test for the sake of simplicity. Splitting the data before building the model is a popular approach to avoid overfitting. That'll be a topic for a future article, so stay tuned!

That's it! You've just learned how to make a simple and multiple linear regression in Python. You can find all the code written in this guide on my Github.