Statistical Modelling With Python: The Three Must Know “S-Modules”

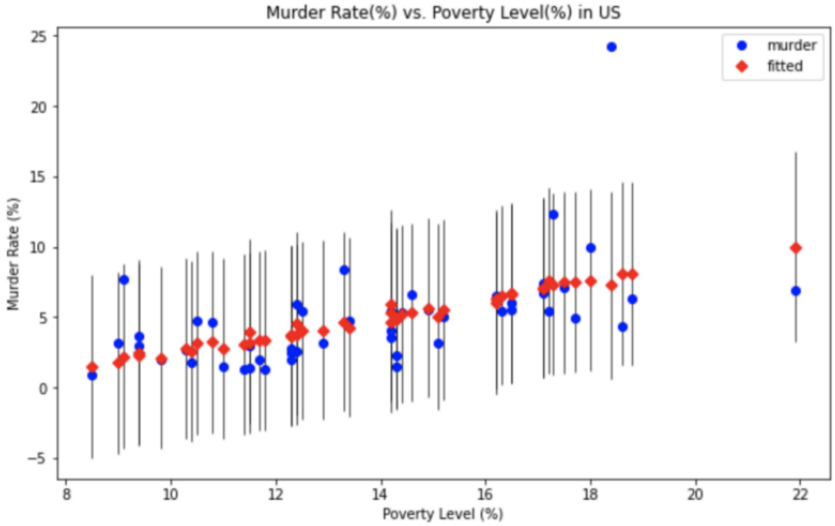

Fig. 1. The plot of the Murder Rate (%) versus Poverty Level (%) in the US by using the **regressionplots.plot_fit** sub-module of the **graphics** sub-module. Figure created by the author for educational purposes.

Introduction

Python is among the most important and efficient multitasking software that exists and it offers the user many libraries and modules to perform different tasks. In the context of data science, statistical modelling is an important part of the everyday job of a data scientist and Python has different libraries and modules that can perform efficient statistical calculations and visualisation.

In this article, I discuss the most important Python libraries/modules that you must know and keep always on your side when you need to perform statistical modelling and calculations. The libraries that I discuss below, are in my opinion the best ones, and I call them the “**S-modules**” because they all initiate with the letter “S” to reflect their statistical nature. In this article, I assume that the reader knows Python and its most important libraries.

1. Statsmodels

If there is a statistical library/module in Python that one must know, that module is Statsmodels. It is probably my favorite Python module when I need to perform statistical calculations. This library/module is based on the SciPy Python library and it is a complete module that allows the user to perform numerous operations for statistical analysis.

This Python module is usually called with the following Python syntax:

[In] **import** statsmodels.api **as** sm

You can note that this module has the api extension which is slightly different from other Python libraries and modules. Statsmodels in many cases has different nomenclature and syntax from other Python modules. For example, if one wants to analyse data for statistical purposes, Statsmodels uses **endog** and **exog** terminology for *x* and *y* variables. endog is an abbreviation of the word *endogenous* that essentially means caused by factors within a system. On the other hand, exog is an abbreviation of the word *exogeneous* that essentially means caused by factors outside a system. So one should keep in mind this terminology while dealing with the Statsmodels module documentation.

What can the user do with Statsmodels in statistical modelling? As I already mentioned above, with this module the user can practically do almost everything in this field. In the case of multivariate linear regression, the OLS sub-module allows the user to perform multivariate linear regression with the Ordinary Least Square(OLS) method. If the user wants to perform Weighted Least Square(WLS), then the WLS sub-module is the one to use. If the user wants to perform, Generalised Least Square (GLS) sub-module is the one to use. In addition, in the context of regression, Statsmodels can perform, ANOVA tests, regression with discrete dependent variable *y,* linear mixed effect models, etc.

For pure statistic calculations and analysis, Statsmodel has a sub-module called **stats** that allows the user to perform several statistics tests. This sub-module is usually called with the following Python syntax:

```
[In] from statsmodels import stats
```

With the **stats** sub-module one can perform numerous statistical tests based on the specific problem that one encounters. Just to mention a few, with the stats sub-module you can perform different Chi-Square tests for goodness of fit, Anderson-Darling test, Ramsey's RESET test, Omnibus test for normality, etc.

A complete statistical module like Statsmodels could not be left without a graphic sub-module. Indeed, this module also includes the **graphics** sub-module for plotting and visualisation of statistical results. At this point, it is very useful that I give a specific example of how one can use Statsmodels for the graphic visualisation of statistical results.

Statsmodels has also a sub-module called **datasets** where one can choose defoult pre-loaded datasets for statistical modelling. In this article, I choose the *statescrime dataset* as a matter of example which is a dataset that includes many parameters related to crimes in the US. I run the following Python code to show few entries of the statecrime dataset as shown in Fig. 2 below:

[In]: **import** statsmodels.api **as** sm

[In]: **import** matplotlib.pyplot **as** plt

[In]: data = sm.datasets.statecrime.load_pandas().data

[In]: data.head(10)

[Out]:

state	violent	murder	hs_grad	poverty	single	white	urban
Alabama	459.9	7.1	82.1	17.5	29.0	70.0	48.65
Alaska	632.6	3.2	91.4	9.0	25.5	68.3	44.46
Arizona	423.2	5.5	84.2	16.5	25.7	80.0	80.07
Arkansas	530.3	6.3	82.4	18.8	26.3	78.4	39.54
California	473.4	5.4	80.6	14.2	27.8	62.7	89.73
Colorado	340.9	3.2	89.3	12.9	21.4	84.6	76.86
Connecticut	300.5	3.0	88.6	9.4	25.0	79.1	84.83
Delaware	645.1	4.6	87.4	10.8	27.6	71.9	68.71
District of Columbia	1348.9	24.2	87.1	18.4	48.0	38.7	100.00
Florida	612.6	5.5	85.3	14.9	26.6	76.9	87.44

Fig. 2. First ten rows of the statecrime Pandas dataset.

Now suppose I want to use the dataset in Fig. 2 to perform a multivariate linear regression of the poverty rate vs. murder rate and hs_grad and plot the result of **only** poverty rate versus murder rate. To achieve this, I use **graphics.plot_fit** sub-module with the following Python code:

[In]: y = data['murder'] # Select the murder rate column from the data dataset as the exonog variable.

[In]: X = data[['poverty', 'hs_grad']].copy() # Create the design matrix by using the poverty rate and hs_grad columns of data

[In]: X['constant'] = 1 # Add a constant term of to the design matrix as the intercept.

[In]: model = sm.OLS(y, X) # Perform ordinary OLS on the data

[In]: results = model.fit()

[In]: fig, ax = plt.subplots(figsize = (10, 6))
fig = plot_fit(results, 0, ax=ax, vlines = True)
ax.set_ylabel("Murder Rate (%)")
ax.set_xlabel("Poverty Level (%)")
ax.set_title("Murder Rate vs. Poverty Level in US")

[Out]: plt.show()

If you run the code above, you will get exactly Fig. 1 shown at the top of this article. The vertical lines represent the residuals of the linear regression of the fitted data to the true data (murder data). As you can see, the **graphics.regressionplots.plot_fit** sub-module performs multiple linear regression with two predictors (in the case considered in this article) and it plots the results for only one predictor. This example indicates the many options that the Statsmodels module offers to the user. There are many others that I do not discuss here.

2. Stats

SciPy is the Python library for scientific calculations. This library has many modules and sub-modules that make it one of the best libraries for scientific computation. SciPy has a module called **stats** that is responsible for statistical modelling and calculations. This module is a predecessor of the **Statsmodels** module where the latter is an extension of the former. Both these modules are complementary to each other and in many cases, they also have overlapping functions with not necessarily the same calling syntax.

The **stats** module allows the user to perform different statistical calculations such as summary statistics, frequency statistics, calculate correlation functions, calculate univariate and multivariate probability distributions (continuous and discrete), Monte-Carlo simulations, generate random variables, etc.

Even in this section, to illustrate the performance of SciPy, I give a specific example. Here I use the same data as in my previous article, where the x-array is the GDP per capita of different countries and y-array is the life satisfaction value. With these arrays, I use **stats** module to calculate the *Pearson correlation coefficient (r)* and its p-value by using the following Python code:

```
[In]: import numpy as np
[In]: from scipy import stats
[In]: x = np.array([ 56755.72171242,  44178.04737774,  40991.80813814,   8814.00098681, 43585.51198178,  13574.1718307
[In]: y = np.array([7.3, 7.1, 6.9, 6.4, 7.4, 6.5, 6.3, 6.7, 7.6, 5.7, 7.6, 6.5, 7.0, 5.4, 5.6, 7.5, 7.0, 7.2, 6.0, 5.9, 5.9, 5.9, 6.9, 6.5, 7.4,
[In]: (r, p_value) = stats.pearsonr(x, y)
[Out]: (0.7202871953226558, 3.426556470064707e-07)
```

As you can see from the Python code above, the Pearson correlation coefficient coincides with that calculated in my previous article. The *two-sided* p-value calculated with the **pearsonr** sub-module is valid under the assumption that x is drawn from a normal distribution, a condition that does not necessarily hold for x in this example.

One can go more in detail and calculate either the critical *z* or *t* values. For example, in the case of linear regression, the critical *z* or *t* values are calculated for the linear regression coefficients, namely for the intercept and slope. Suppose one is interested to calculate the critical values at a 95% confidence level. In this case, one needs to run the following Python code:

```
[In]: n=38                    # Number of degrees of freedom
[In]: alpha=0.05              # Significance level
[In]: t_critical = stats.t.ppf(1-alpha/2, n-2) # ppf is the inverse function of cdf
```

```
[In]: print("Model t_critical: ", t_critical)
[Out]: Model t_critical:  2.0280940009804502 # Critical value of T statistic at alpha=0.05
```

You can see that to calculate the critical t value I used the **stats** sub-module **t.ppf().** This function is the inverse function of the cumulative distribution function and it calculates the *t* values for various values of the confidence level expressed as percentage values. You can see on my GitHub page that in the case of the linear regression over the x and y arrays, the t-score value is *t_s = 6.23*, which is larger than *t_critical = 2.02*, and thus one rejects the null hypothesis that *x* and *y* are uncorrelated.

3. Seaborn

Seaborn is another excellent Python library for statistical data visualization. This Python library is mostly built on top of the **Matplotlib** library and it also includes many of Pandas library functionalities. Seaborn is exclusively used to visualise statistical data and plots. It can be easily installed either with the PyPi command **pip install seaborn** or with Anaconda distribution through the command **conda install seaborn.** In general, seaborn requires different library dependencies such as NumPy, SciPy, Matplotlib, and Pandas. If these libraries are not present in your Python distribution, then they will be installed together with seaborn.

With seaborn you can:

1. Visualise statistical relationships
2. Visualise distributions of data
3. Visualise regression models
4. Plotting categorical data
5. Controll graphics aestetichs
6. Build multi-plot grids
7. Choose color palettes

Seaborn has numerous options to plot and visualise statistical results. As a matter of example, consider the Pandas dataset in Fig. 2 in section 2. Suppose that I want to look for any possible linear relationship between the poverty rate and murder rate. To see this, I run the following Python code and show the result in Fig. 3 below:

```
[In]: import seaborn as sns
[In]: sns.set(style = "whitegrid")
[In]: fig = sns.lmplot(x="poverty", y="murder", data=data)
[Out]:
```

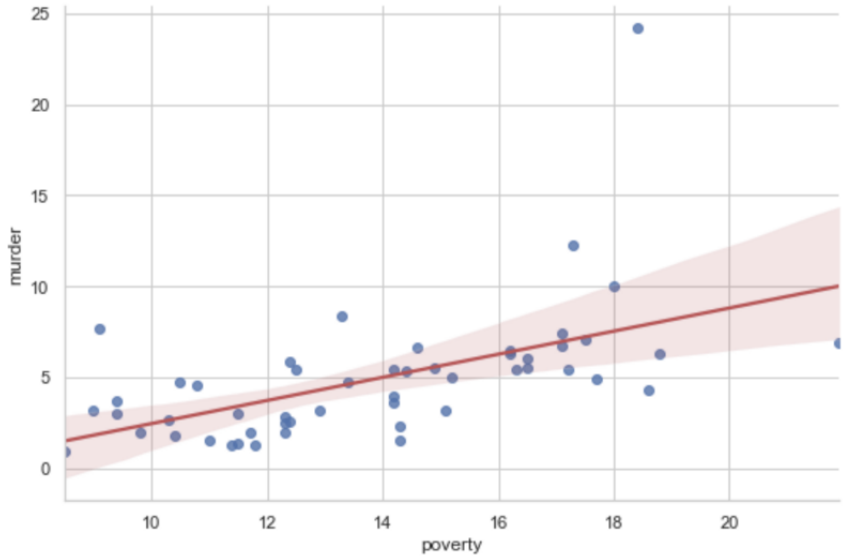

Fig. 3. The plot of the murder rate(%) versus poverty rate(%) in the US by using the **sns.lmplot** sub-module of the **seaborn** library. Image created by the author for educational purposes.

As you can see, with seaborn one can use many different options this library offers to the user and create excellent final results. Clearly, here I do not show these many other functionalities of seaborn, and I invite the reader to experiment with them.

Conclusions

In this article, I discussed the three most important statistical modules that exist in Python. These are the modules that you should always keep on your side if you want to perform statistical modelling and calculations. Each of them has almost unique features and in most cases, they are also complementary to each other. Statsmoldels should be your number one module for statistical modelling and seaborn should be your number two since it allows you to create figures of statistical modelling without the user needing to make any calculations or write additional Python codes to make these calculations.

If you liked my article, please share it with your friends that might be interested in this topic and cite/refer to my article in your research studies. Do not forget to subscribe for other related topics that will post in the future.

By Damian Ejlli on September 28, 2021.

Canonical link

Exported from Medium on October 8, 2021.