Statsmodels: Econometric and Statistical Modeling with Python

Skipper Seabold[§*], Josef Perktold[‡]

Abstract—*Statsmodels* is a library for statistical and econometric analysis in Python. This paper discusses the current relationship between statistics and Python and open source more generally, outlining how the *statsmodels* package fills a gap in this relationship. An overview of *statsmodels* is provided, including a discussion of the overarching design and philosophy, what can be found in the package, and some usage examples. The paper concludes with a look at what the future holds.

Index Terms—statistics, econometrics, R

Introduction

Statsmodels (http://statsmodels.sourceforge.net/) is a library for statistical and econometric analysis in Python[1]. Its intended audience is both theoretical and applied statisticians and econometricians as well as Python users and developers across disciplines who use statistical models. Users of R, Stata, SAS, SPSS, NLOGIT, GAUSS or MATLAB for statistics, financial econometrics, or econometrics who would rather work in Python for all its benefits may find *statsmodels* a useful addition to their toolbox. This paper introduces *statsmodels* and is aimed at the researcher who has some prior experience with Python, NumPy/SciPy [SciPy][2].

On a historical note, *statsmodels* was started by Jonathan Taylor, a statistician now at Stanford, as part of SciPy under the name *models*. Eventually, *models* was removed from SciPy and became part of the NIPY neuroimaging project [NIPY] in order to mature. Improving the *models* code was later accepted as a SciPy-focused project for the Google Summer of Code 2009 and again in 2010. It is currently distributed as a SciKit, or add-on package for SciPy.

The current main developers of *statsmodels* are trained as economists with a background in econometrics. As such, much of the development over the last year has focused on econometric applications. However, the design of *statsmodels* follows a consistent pattern to make it user-friendly and easily extensible by developers from any discipline. New contributions and ongoing work are making the code more useful for common statistical modeling needs. We hope that continued efforts will result in a package useful for all types of statistical modeling.

* Corresponding author: jsseabold@gmail.com
§ American University
‡ CIRANO, University of North Carolina Chapel Hill

The State of the Union: Open Source and Statistics

Currently R is the open source language of choice for applied statistics. In applied econometrics, proprietary software pacakges such as Gauss, MATLAB, Stata, SAS, and NLOGIT remain the most popular and are taught in most graduate programs. However, there is a growing call for the use of open source software in economic research due in large part to its reliability, transparency, and the paradigm it offers for workflow and innovation [YaltaYalta], [YaltaLucchetti]. In particular R is increasing in popularity as evidenced by the recent textbooks by Cryer and Chan (2008), Kleiber and Zeileis (2008), and Vinod (2008). Gretl is another notable open source alternative for econometrics [Gretl].

However, there are those who would like to see Python become the language of choice for economic research and applied econometrics. Choirat and Seri's "Econometrics with Python" is the first publication of which we are aware that openly advocates the use of Python as the language of choice for econometricians [ChoiratSeri]. Bilina and Lawford express similar views [BilinaLawford]. Further, John Stachurski has written a Python-based textbook, *Economic Dynamics: Theory and Computation* [Stachurski], and Alan Isaac's "Simulating Evolutionary Games: a Python-Based Introduction" showcases Python's abilities for implementing agent-based economic models [Isaac].

In depth arguments for the choice of Python are beyond the scope of this paper; however, Python is well known for its simple syntax, gentle learning curve, and large standard library. Beyond this, it is a language for much more than statistics and can be the one toolbox for researchers across discplines. A few examples of statistics-related packages that are outside of the main numpy/scipy code are packages for Markov Chain Monte Carlo and Bayesian statistics [PyMC], machine learning and multivariate pattern analysis [scikits-learn], [PyMVPA], neuroimaging [NIPY] and neuroscience time series [NITIME], visualization [Matplotlib], [Enthought], and efficient handling of large datasets [PyTables].

We hope that *statsmodels* too can become an integral a part of the Scientific Python community and serve as a step in the direction of Python becoming a serious open source language for statistics. Towards this end, others are working on an R-like formula framework to help users specify and manipulate models [charlton], and packages like pandas [pandas] (discussed in these proceedings) and larry [larry] are providing flexible data structures and routines for data analysis currently lacking in NumPy.

Statsmodels: Development and Design

It should not be the case that different conclusions can be had from the same data depending on the choice of statistical software or its version; however, this is precisely what Altman and MacDonald (2003) find [AltmanMcDonald]. Given the importance of numerical accuracy and replicablity of research and the multitude of software choices for statistical analysis, the development of *statsmodels* follows a process to help ensure accurate and transparent results. This process is known as Test-Driven Development (TDD). In its strictest form, TDD means that tests are written before the functionality which it is supposed to test. While we do not often take the strict approach, there are several layers in our development process that ensure that our results are correct versus existing software (often R, SAS, or Stata). Any deviations from results in other software are noted in the test suite.

First, we employ a distributed version control system in which each developer has his own copy of the code, a branch, to make changes outside of the main codebase. Once a model is specified, early changes, such as working out the best API or bug hunting, take place in the main branch's, or trunk's, sandbox directory so that they can be tried out by users and developers who follow the trunk code. Tests are then written with results taken from another statistical package or Monte Carlo simulation when it is not possible to obtain results from elsewhere. After the tests are written, the developer asks for a code review on our mailing list (http://groups.google.ca/group/pystatsmodels). When all is settled, the code becomes part of the main codebase. Periodically, we release the software in the trunk for those who just want to download a tarball or install from PyPI, using *setuptools*' easy_install. This workflow, while not foolproof, helps make sure our results are and remain correct. If they are not, we are able to know why and document discrepancies resulting in the utmost transparency for end users. And if all else fails, looking at the source code is trivial to do (and encouraged!).

The design of the package itself is straightforward. The main idea behind the design is that a model is itself an object to be used for data reduction. Data can be both endogenous and exogenous to a model, and these constituent parts are related to each other through statistical theory. This statistical relationship is usually justified by an appeal to discipline-specific theory. Note that in place of endogenous and exogenous, one could substitute the terms dependent and independent variables, regressand and regressors, response and explanatory variables, etc., respectively, as you prefer. We maintain the endogenous-exogenous terminology throughout the package, however.

With this in mind, we have a base class, *Model*, that is intended to be a template for parametric models. It has two main attributes *endog* and *exog*[3] and placeholders for *fit* and *predict* methods. *LikelihoodModel* is a subclass of *Model* that is the workhorse for the regression models. All *fit* methods are expected to return some results class. Towards this end, we also have a base class *Results* and *LikelihoodModelResults* which inherits from *Results*. The result objects have attributes and methods that contain common post-estimation results and statistical tests. Further, these are computed lazily, meaning that they are not computed until the user asks for them so that those who are only interested in, say, the fitted parameters are not slowed by computation of extraneous results. Every effort is made to ensure that the constructors of each sublcass of *Model*, the call signatures of its methods, and the post-estimation results are consistent throughout the package.

Package Overview

Currently, we have five modules in the main codebase that contain statistical models. These are *regression* (least squares regression models), *glm* (generalized linear models), *rlm* (robust linear models), *discretemod* (discrete choice models), and *contrast* (contrast analysis). *Regression* contains generalized least squares (*GLS*), weighted least squares (*WLS*), and ordinary least squares (*OLS*). *Glm* contains generalized linear models with support for six common exponential family distributions and at least ten standard link functions. *Rlm* supports M-estimator type robust linear models with support for eight norms. *Discretemod* includes several discrete choice models such as the *Logit*, *Probit*, Multinomial Logit (*MNLogit*), and *Poisson* within a maximum likelihood framework. *Contrast* contains helper functions for working with linear contrasts. There are also tests for heteroskedasticity, autocorrelation, and a framework for testing hypotheses about linear combinations of the coefficients.

In addition to the models and the related post-estimation results and tests, *statsmodels* includes a number of convenience classes and functions to help with tasks related to statistical analysis. These include functions for conveniently viewing descriptive statistics, a class for creating publication quality tables, and functions for translating foreign datasets, currently only Stata's binary *.dta* format, to numpy arrays.

The last main part of the package is the datasets. There are currently fourteen datasets that are either part of the public domain or used with express consent of the original authors. These datasets follow a common pattern for ease of use, and it is trivial to add additional ones. The datasets are used in our test suite and in examples as illustrated below.

Examples

All of the following examples use the datasets included in *statsmodels*. The first example is a basic use case of the OLS model class to get a feel for the rest of the package, using Longley's 1967 dataset [Longley] on the US macro economy. Note that the Longley data is known to be highly collinear (it has a condition number of 456,037), and as such it is used to test accuracy of least squares routines than to examine any economic theory. First we need to import the package. The suggested convention for importing *statsmodels* is

```
>>> import scikits.statsmodels as sm
```

Numpy is assumed to be imported as:

```
>>> import numpy as np
```

Then we load the example dataset.

```
>>> longley = sm.datasets.longley
```

The datasets have several attributes, such as descriptives and copyright notices, that may be of interest; however, we will just load the data.

```
>>> data = longley.load()
```

Many of the *Dataset* objects have two attributes that are helpful for tests and examples -*endog* and *exog*- though the whole dataset is available. We will use them to construct an *OLS* model instance. The constructor for *OLS* is

```
def __init__(self, endog, exog)
```

It is currently assumed that the user has cleaned the dataset and that a constant is included, so we first add a constant and then instantiate the model.

```
>>> data.exog = sm.add_constant(data.exog)
>>> longley_model = sm.OLS(data.endog, data.exog)
```

We are now ready to fit the model, which returns a *RegressionResults* class.

```
>>> longley_res = longley_model.fit()
>>> type(longley_res)
<class 'sm.regression.RegressionResults'>
```

By default, the least squares models use the pseudoinverse to compute the parameters that solve the objective function.

```
>>> params = np.dot(np.linalg.pinv(data.exog),
                    data.endog)
```

The instance *longley_res* has several attributes and methods of interest. The first is the fitted values, commonly β in the general linear model, $Y = X\beta$, which is called *params* in *statsmodels*.

```
>>> longley_res.params
array([  1.50618723e+01, -3.58191793e-02,
    -2.02022980e+00,  -1.03322687e+00,
    -5.11041057e-02,   1.82915146e+03,
    -3.48225863e+06])
```

Also available are

```
>>> [_ for _ in dir(longley_res) if not
     _.startswith('_')]
['HC0_se', 'HC1_se', 'HC2_se', 'HC3_se', 'aic',
'bic', 'bse', 'centered_tss', 'conf_int',
'cov_params', 'df_model', 'df_resid', 'ess',
'f_pvalue', 'f_test', 'fittedvalues', 'fvalue',
'initialize', 'llf', 'model', 'mse_model',
'mse_resid', 'mse_total', 'nobs', 'norm_resid',
'normalized_cov_params', 'params', 'pvalues',
'resid', 'rsquared', 'rsquared_adj', 'scale', 'ssr',
'summary', 't', 't_test', 'uncentered_tss', 'wresid']
```

All of the attributes and methods are well-documented in the docstring and in our online documentation. See, for instance, help(longley_res). Note as well that all results objects carry an attribute *model* that is a reference to the original model instance that was fit whether or not it is instantiated before fitting.

Our second example borrows from Jeff Gill's *Generalized Linear Models: A Unified Approach* [Gill]. We fit a Generalized Linear Model where the endogenous variable has a binomial distribution, since the syntax differs somewhat from the other models. Gill's data comes from the 1998 STAR program in California, assessing education policy and outcomes. The endogenous variable here has two columns. The first specifies the number of students above the national median score for the math section of the test per school district. The second column specifies the number of students below the median. That is, *endog* is (number of sucesses, number of failures). The explanatory variables for each district are measures such as the percentage of low income families, the percentage of minority students and teachers, the median teacher salary, the mean years of teacher experience, per-pupil expenditures, the pupil -teacher ratio, the percentage of student taking college credit courses, the percentage of charter schools, the percent of schools open year round, and various interaction terms. The model can be fit as follows

```
>>> data = sm.datasets.star98.load()
>>> data.exog = sm.add_constant(data.exog)
>>> glm_bin = sm.GLM(data.endog, data.exog,
                     family=sm.families.Binomial())
```

Note that you must specify the distribution family of the endogenous variable. The available families in *scikits.statsmodels.families* are *Binomial*, *Gamma*, *Gaussian*, *InverseGaussian*, *NegativeBinomial*, and *Poisson*.

The above examples also uses the default canonical logit link for the Binomial family, though to be explicit we could do the following

```
>>> links = sm.families.links
>>> glm_bin = sm.GLM(data.endog, data.exog,
        family=sm.families.Binomial(link=
        links.logit))
```

We fit the model using iteratively reweighted least squares, but we must first specify the number of trials for the endogenous variable for the Binomial model with the endogenous variable given as (success, failure).

```
>>> trials = data.endog.sum(axis=1)
>>> bin_results = glm_bin.fit(data_weights=trials)
>>> bin_results.params
array([ -1.68150366e-02,   9.92547661e-03,
    -1.87242148e-02,  -1.42385609e-02,
     2.54487173e-01,   2.40693664e-01,
     8.04086739e-02,  -1.95216050e+00,
    -3.34086475e-01,  -1.69022168e-01,
     4.91670212e-03,  -3.57996435e-03,
    -1.40765648e-02,  -4.00499176e-03,
    -3.90639579e-03,   9.17143006e-02,
     4.89898381e-02,   8.04073890e-03,
     2.22009503e-04,  -2.24924861e-03,
     2.95887793e+00])
```

Since we have fit a GLM with interactions, we might be interested in comparing interquartile differences of the response between groups. For instance, the interquartile difference between the percentage of low income households per school district while holding the other variables constant at their mean is

```
>>> means = data.exog.mean(axis=0) # overall means
>>> means25 = means.copy() # copy means
>>> means75 = means.copy()
```

We can now replace the first column, the percentage of low income households, with the value at the first quartile using scipy.stats and likewise for the 75th percentile.

```
>>> from scipy.stats import scoreatpercentile as sap
>>> means25[0] = sap(data.exog[:,0], 25)
>>> means75[0] = sap(data.exog[:,0], 75)
```

And compute the fitted values, which are the inverse of the link function at the linear predicted values.

```
>>> lin_resp25 = glm_bin.predict(means25)
>>> lin_resp75 = glm_bin.predict(means75)
```

Therefore the percentage difference in scores on the standardized math tests for school districts in the 75th percentile of low income households versus the 25th percentile is

```
>>> print "%4.2f percent" % ((lin_resp75-
                             lin_resp25)*100)
-11.88 percent
```

The next example concerns the testing of joint hypotheses on coefficients and is inspired by a similar example in Bill Greene's *Econometric Analysis* [Greene]. Consider a simple static investment function for a macro economy

$$\ln I_t = \beta_1 + \beta_2 \ln Y_t + \beta_3 i_t + \beta_4 \Delta p_t + \beta_5 t + \varepsilon_t \qquad (1)$$

In this example, (log) investment, I_t is a function of the interest rate, i_t, inflation, Δp_t, (log) real GDP, Y_t, and possibly follows a

linear time trend, t. Economic theory suggests that the following model may instead be correct

$$\ln I_t = \beta_1 + \ln Y_t + \beta_3 (i_t - \Delta p_t) + \varepsilon_t \quad (2)$$

In terms of the (1) this implies that $\beta_3 + \beta_4 = 0$, $\beta_2 = 1$, and $\beta_5 = 0$. This can be implemented in *statsmodels* using the *macrodata* dataset. Assume that *endog* and *exog* are given as in (1)

```
>>> inv_model = sm.OLS(endog, exog).fit()
```

Now we need to make linear restrictions in the form of $R\beta = q$

```
>>> R = [[0,1,0,0,0],[0,0,1,1,0],[0,0,0,0,1]]
>>> q = [1,0,0]
```

$R\beta = q$ implies the hypotheses outlined above. We can test the joint hypothesis using an F test, which returns a *ContrastResults* class

```
>>> Ftest = inv_model.f_test(R,q)
>>> print Ftest
<F test: F=array([[ 194.4428894]]),
p=[[  1.27044954e-58]], df_denom=197, df_num=3>
```

Assuming that we have a correctly specified model, given the high value of the F statistic, the probability that our joint null hypothesis is true is essentially zero.

As a final example we will demonstrate how the *SimpleTable* class can be used to generate tables. SimpleTable is also currently used to generate our regression results summary. Continuing the example above, one could do

```
>>> print inv_model.summary(yname="lninv",
            xname=["const","lnY","i","dP","t"])
```

To build a table, we could do:

```
>>> gdpmean = data.data['realgdp'].mean()
>>> invmean = data.data['realinv'].mean()
>>> gdpstd = data.data['realgdp'].std()
>>> invstd = data.data['realinv'].std()
>>> mydata = [[gdpmean, gdpstd],[invmean,
        invstd]]
>>> myheaders = ["Mean", "Std Dev."]
>>> mystubs = ["Real GDP", "Real Investment"]
>>> tbl = sm.iolib.SimpleTable(mydata,
        myheaders, mystubs, title =
        "US Macro Data", data_fmts=['%4.2f'])
>>> print tbl
         US Macro Data
================================
                 Mean  Std Dev.
--------------------------------
Real GDP        7221.17 3207.03
Real Investment 1012.86  583.66
--------------------------------
```

LaTeX output can be generated with something like

```
>>> fh = open('./tmp.tex', 'w')
>>> fh.write(tbl.as_latex_tabular())
>>> fh.close()
```

While not all of the functionality of *statsmodels* is covered in the above, we hope it offers a good overview of the basic usage from model to model. Anything not touched on is available in our documentation and in the examples directory of the package.

Conclusion and Outlook

Statsmodels is very much still a work in progress, and perhaps the most exciting part of the project is what is to come. We currently have a good deal of code in our sandbox that is being cleaned up, tested, and pushed into the main codebase as part of the Google Summer of Code 2010. This includes models for time-series analysis, kernel density estimators and nonparametric regression, panel or longitudinal data models, systems of equation models, and information theory and maximum entropy models.

We hope that the above discussion gives some idea of the appoach taken by the project and provides a good overview of what is currently offered. We invite feedback, discussion, or contributions at any level. If you would like to get involved, please join us on our mailing list available at http://groups.google.com/group/pystatsmodels or on the scipy-user list. If you would like to follow along with the latest development, the project blog is http://scipystats.blogspots.com and look for release announcements on the scipy-user list.

All in all, we believe that the future for Python and statistics looks bright.

Acknowledgements

In addition to the authors of this paper, many others have contributed to the codebase. Thanks are due to Jonathan Taylor and contributors to the models code while it was in SciPy and NIPY. Thanks also go to those who have provided code, constructive comments, and discussion on the mailing lists.

References

[AltmanMcDonald] M. Altman and M.P. McDonald. 2003. "Replication with Attention to Numerical Accuracy." *Political Analysis*, 11.3, 302-7.

[BilinaLawford] R. Bilina and S. Lawford. July 4, 2009. *Python for Unified Research in Econometrics and Statistics*, July 4, 2009. Available at SSRN: http://ssrn.com/abstract=1429822

[charlton] Charlton. Available at https://github.com/charlton

[ChoiratSeri] C. Choirat and R. Seri. 2009. "Econometrics with Python." *Journal of Applied Econometrics*, 24.4, 698-704.

[CryerChan] J.D. Cryer and K.S. Chan. 2008. *Time Series Analysis: with Applications in R*, Springer.

[Enthought] Enthought Tool Suite. Available at http://code.enthought.com/.

[Gill] J. Gill. 2001. *Generalized Linear Models: A Unified Approach*. Sage Publications.

[Greene] W. H. Greene. 2003. *Econometric Analysis* 5th ed. Prentice Hall.

[Gretl] Gnu Regression, Econometrics, and Time-series Library: gretl. Available at http://gretl.sourceforge.net/.

[Isaac] A.G. Isaac. 2008. "Simulating Evolutionary Games: a Python- Based Introduction." *Journal of Artificial Societies and Social Simulation*. 11.3.8. Available at http://jasss.soc.surrey.ac.uk/11/3/8.html

[KleiberZeileis] C. Kleiber and A. Zeileis. 2008. *Applied Econometrics with R*, Springer.

[Longley] J.W. Longley. 1967. "An Appraisal of Least Squares Programs for the Electronic Computer from the Point of View of the User." *Journal of the American Statistical Association*, 62.319, 819-41.

[Matplotlib] J. Hunter, *et al.* Matplotlib. Available at http://matplotlib.sourceforge.net/index.html.

1. The examples reflect the state of the code at the time of writing. The main model API is relatively stable; however, recent refactoring has changed the organization of the code. See online documentation for the current usage.

2. Users who wish to learn more about NumPy can do so at http://www.scipy.org/Tentative_NumPy_Tutorial, http://www.scipy.org/Numpy_Example_List, or http://mentat.za.net/numpy/intro/intro.html. For those coming from R or MATLAB, you might find the following helpful: http://mathesaurus.sourceforge.net/ and http://www.scipy.org/NumPy_for_Matlab_Users

3. The *exog* attribute is actually optional, given that we are developing support for (vector) autoregressive processes in which all variables could at times be thought of as "endogenous".

[larry] K.W. Goodman. Larry: Labeled Arrays in Python. Available at http://larry.sourceforge.net/.

[NIPY] NIPY: Neuroimaging in Python. Available at http://nipy.org.

[NITIME] Nitime: time-series analysis for neuroscience. Available at http://nipy.org/nitime

[pandas] W. McKinney. Pandas: A Python Data Analysis Library. Available at http://pandas.sourceforge.net/.

[PyMC] C. Fonnesbeck, D. Huard, and A. Patil, PyMC: Pythonic Markov chain Monte Carlo. Available at http://code.google.com/p/pymc/

[PyMVPA] M. Hanke, *et. al.* PyMVPA: Multivariate Pattern Analysis in Python. Available at http://www.pymvpa.org/.

[PyTables] F. Alted, I. Vilata, *et al.* PyTables: Hierarchical Datasets in Python. Available at http://www.pytables.org.

[scikits-learn] Pedregosa, F. *et al.* `scikits.learn`: machine learning in Python. Available at http://scikits-learn.sourceforge.net.

[SciPy] T. Oliphant, *et al.* SciPy. Available at http://www.scipy.org.

[Stachurski] J. Stachurski. 2009. *Economic Dynamics: Theory and Computation*. MIT Press.

[Vinod] H.D. Vinod. 2008. *Hands-on Intermediate Econometrics Using R*, World Scientific Publishing.

[YaltaLucchetti] A.T. Yalta and R. Lucchetti. 2008. "The GNU/Linux Platform and Freedom Respecting Software for Economists." *Journal of Applied Econometrics*, 23.3, 279-86.

[YaltaYalta] A.T. Yalta and A.Y. Yalta. 2010. *Should Economists Use Open Source Software for Doing Research*. Computational Economics, 35, 371-94.