Statsmodels

Econometric and Statistical Modeling with Python

Skipper Seabold[1] Josef Perktold[2]

[1]Department of Economics
American University

[2]CIRANO
University of North Carolina at Chapel Hill

Python for Scientific Computing Conference, 2010

Outline

What is statsmodels?

- A library for statistical and econometric analysis in Python
- Useful for users of R, GAUSS or MATLAB as well as SAS, Stata, SPSS, NLOGIT, gretl, or eViews.
- Statistical, Financial Econometric, and Econometric models

Background and Overview

- Jonathan Taylor $\longrightarrow$ SciPy (models) $\longrightarrow$ NIPY$\longrightarrow$ GSoC
- Now distributed as a SciKit
- Main developers from economics
- Consistent design for general statistical modeling

State of the Union

- R for applied statistics
- Econometrics - mainly proprietary software
  - Proprietary: GAUSS, MATLAB (time series/macro), Stata, SAS, NLOGIT, etc.
  - FLOSS: R (Finance, theoretical), gretl (both GPL)

Python and Statistics

- Growing call for FLOSS in economic research and Python to be the language of choice for applied and theoretical econometrics
  - Choirat and Seri (2009), Bilina and Lawford (2009), Stachurski (2009), Isaac (2008)
- Finance/ SEC - Asset Backed Securities
  - "we are proposing to require, along with the prospectus filing [in XML], the filing of a computer program of the contractual cash flow provisions expressed as downloadable source code in Python"
- Related packages: PyMC, scikits-learn, PyMVPA, NIPY (nitime), matplotlib, PyTables, Biopython, Pyentropy, pandas, larry

Test-Driven Development

- TDD
- Reliability and Accuracy
- Transparency

Development Workflow

- Supports TDD
- Branches vs. Trunk
- Sandbox and code review
- Test results (R, Stata, SAS, Monte Carlo)

What is a Model?

- Object for data reduction
- Data: Endogenous and Exogenous
  - Terminology: Dependent/Independent, Regressand/Regressor, Response/Explanatory
- Statistical theory provides the relationship between the two
- Naturally leads to OO design

Implementation

- Base class: Model

```
class Model(object):
    def __init__(self, endog, exog=None):
        self.endog = endog
        if exog is not None:
            self.exog = exog
    def fit(self):
        ...
    def predict(self):
        ...
```

Implementation Con't

- Inheritance: LikelihoodModel

```
class LikelihoodModel(Model):
    def __init__(self, endog, exog=None):
        super(LikelihoodModel, self).__init__(endog,
                exog)
        self.initialize()
    def initialize(self):
        pass
    def fit(self, start_params=None, method='newton',
            maxiter=100, full_output=True,
            disp=True, fargs=(), callback=None,
            retall=False, **kwargs):
        ...
        return LikelihoodModelResults(self, ...)
```

Implementation Con't

- Results Objects

```
class LikelihoodModelResults(Results, LLMTests):
    def __init__(self, model, ...):
        self.model = model
        ...
```

Package Overview

- Main model modules
  - regression
  - glm
  - rlm
  - discretemod
  - contrast
- Convenience functions
  - Descriptive Statistics, SimpleTable, Foreign I/O, ...
- Datasets
- Examples

Regression Example

- Import conventions

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

- OLS: $Y = X\beta + \varepsilon$ where $\varepsilon \sim N\left(0, \sigma^2\right)$
- Notation: $params \equiv \beta$

```
>>> data = sm.datasets.longley.load()
>>> data.exog = sm.add_constant(data.exog)
>>> ols_model = sm.OLS(data.endog, data.exog)
>>> ols_results = ols_model.fit()
>>> ols_results.params
array([ 1.50618723e+01, -3.58191793e-02,
       -2.02022980e+00, -1.03322687e+00,
       -5.11041057e-02, 1.82915146e+03,
       -3.48225863e+06])
```

Regression Example Con't

- A peak inside the *RegressionResults* object

```
>>> [_ for _ in dir(ols_results) 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']
```

```
>>> print ols_results.summary()
     Summary of Regression Results
=================================================
| Dependent Variable:                        'y' |
| Model:                                     OLS|
| Method:                         Least Squares|
| Date:                        Tue, 22 Jun 2010|
| Time:                                11:21:43|
| # obs:                                   16.0|
| Df residuals:                             9.0|
| Df model:                                 6.0|
===============================================================================================
|               coefficient           std. error          t-statistic                   prob.|
-----------------------------------------------------------------------------------------------
| x1                15.0619              84.9149               0.1774                 0.8631|
| x2                -0.0358               0.0335              -1.0695                 0.3127|
| x3                -2.0202               0.4884              -4.1364               0.002535|
| x4                -1.0332               0.2143              -4.8220              0.0009444|
| x5                -0.0511               0.2261              -0.2261                 0.8262|
| x6              1829.1515             455.4785               4.0159               0.003037|
| const        -3482258.6346          890420.3836              -3.9108               0.003560|
===============================================================================================
|                        Models stats                                     Residual stats |
-----------------------------------------------------------------------------------------------
| R-squared:                         0.995479   Durbin-Watson:                    2.55949 |
| Adjusted R-squared:                0.992465   Omnibus:                         0.748615 |
| F-statistic:                        330.285   Prob(Omnibus):                   0.687765 |
| Prob (F-statistic):             4.98403e-10   JB:                              0.352773 |
| Log likelihood:                    -109.617   Prob(JB):                        0.838294 |
| AIC criterion:                      233.235   Skew:                            0.419984 |
| BIC criterion:                      238.643   Kurtosis:                         2.43373 |
```

GLM Example

- $Y = g(X\beta) + \varepsilon$ where, in this case, $Y \sim B(\cdot)$ and g^{-1} is the link function such that $\mu_y = g^{-1}(X\beta)$
- Jeff Gill's STAR data

```
>>> data = sm.datasets.star98.load()
>>> data.exog = sm.add_constant(data.exog)
>>> links = sm.families.links
>>> glm_bin = sm.GLM(data.endog, data.exog,
...                  family=sm.families.Binomial(link=
...                  links.logit)
>>> trials = data.endog.sum(axis=1)
>>> glm_results = glm_bin.fit(data_weights=trials)
```

GLM Example Con't

- Look at interquartile difference in predicted success between groups

```
>>> means = data.exog.mean(axis=0)
>>> means25 = means.copy()
>>> means75 = means.copy()
>>> from scipy.stats import scoreatpercentile as sap
>>> means25[0] = sap(data.exog[:,0], 25)
>>> means75[0] = sap(data.exog[:,0], 75)
>>> resp25 = glm_bin.predict(means25)
>>> resp75 = glm_bin.predict(means75)
>>> print "%4.2f percent" % ((resp75-resp25)*100)
-11.88 percent
```

Robust Standard Errors

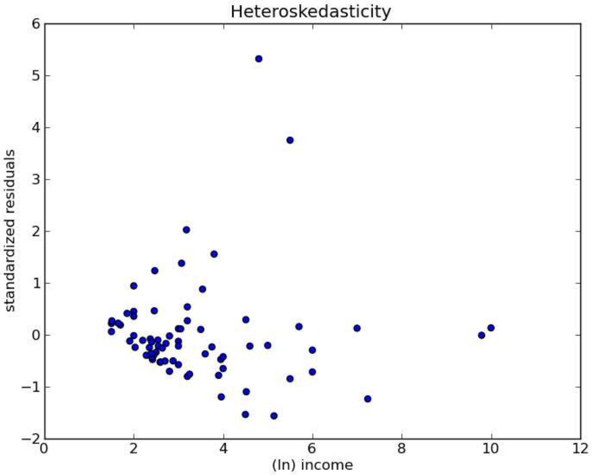
- Bill Greene's credit card data model.

$$AVGEXP = \beta_1 + \beta_2 AGE + \beta_3 INC + \beta_4 INC^2 + \beta_5 OWNRENT$$

```
>>> data = sm.datasets.ccard.load()
>>> data.exog = sm.add_constant(data.exog)
>>> ols_fit = sm.OLS(data.endog, data.exog).fit()
```

- Problem: variance of errors might be assumed to increase with income (though we might not know exact functional form).
- Consequence: standard errors are underestimated.

Robust Standard Errors con't



Robust Standard Errors Con't

- White (1980) Robust Standard Errors: HC0

$$SE(\beta) = \sqrt{diag\left((X'X)^{-1}X'\hat{\varepsilon}_i^2X(X'X)^{-1}\right)}$$

- Small sample analogues MacKinnon and White (1985): HC1

$$SE(\beta) = \sqrt{diag\left(\frac{n}{n-k-1}(X'X)^{-1}X'\hat{\varepsilon}_i^2X(X'X)^{-1}\right)}$$

```
>>> ols_fit.HC1_se
array([ 3.42264107,  92.12260235,  7.19902694,
       95.56573144, 220.79495237])
>>> ols_fit.bse
array([ 5.51471653,  80.36595035,  7.46933695,
       82.92232357, 199.35166485])
>>> ols_fit.t()
array([-0.55883453,   2.91599895,  -2.00778788,
        0.33695279,  -1.18958883])
>>> ols_fit.params/ols_fit.HC1_se
array([-0.90041987,   2.54386026,  -2.08317656,
        0.29237372,  -1.07405768])
```

Linear Restrictions Example

- Consider the following static investment function for a macro economy

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

- Suppose we believe that investors care *only* about real interest rates, that the marginal propensity to invest is unity, and that there is no linear time trend.

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

- In terms of the first model, this implies

$$\begin{aligned} \beta_3 + \beta_4 &= 0 \\ \beta_2 &= 1 \\ \beta_5 &= 0 \end{aligned}$$

Linear Restrictions Example Con't

- In terms of linear restrictions we have

$$R\beta = q$$

where

$$R = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } q = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Linear Restrictions Example Con't

```
>>> data = sm.datasets.macrodata.load()
>>> endog = np.log(data.data['realinv'][1:])
>>> exog = data.data[['realgdp','tbilrate',
...                   'infl']][1:].view((float,3))
>>> exog[:,0] = np.log(exog[:,0])
>>> exog = sm.add_constant(exog, prepend=True)
>>> from scikits.statsmodels.sandbox.tsa.stattools\
...       import add_trend # function will be moved
>>> exog = add_trend(exog, trend='t')
>>> inv_model = sm.OLS(endog, exog).fit()
```

Linear Restrictions Example Con't

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

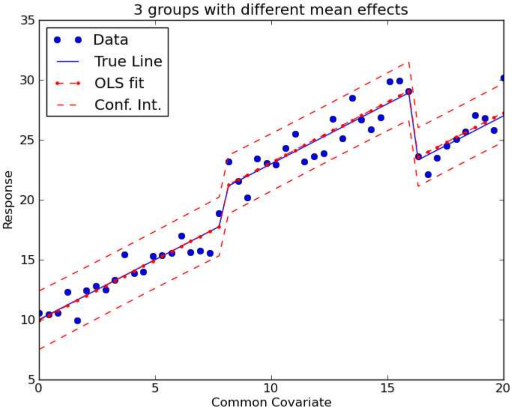
- F-test with $H_0 : R\beta = \hat{q}$.

```
>>> 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>
```

- $\therefore$ we can reject the null hypothesis that $R\beta = q$.

Linear Restrictions as ANCOVA

- We have three groups of subjects that share a common covariate.
- Want to test that the mean effect on the three groups is the same.



Linear Restrictions as ANCOVA con't

- Define R for linear model with group dummies and a constant

$$Y_i = \beta_1 X_i + \beta_2 Group1 + \beta_3 Group2 + \beta_4 + \varepsilon$$

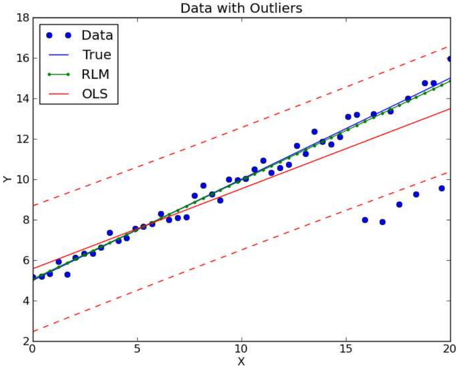
```
>>> ancova_model = sm.OLS(y,X).fit()
>>> R = [[0,1,0,0],[0,0,1,0]]
>>> print ancova_model.f_test(R)
<F test: F=array([[ 91.69986847]]),
 p=[[  8.90826383e-17]], df_denom=46, df_num=2>
```

- Or as ANOVA, f_test is the same as scipy.stats.f_oneway

```
>>> anova_model = sm.OLS(y,X[:,1:]).fit()
>>> anova_model.f_test(R[:,1:])
<F test: F=array([[ 122.07800238]]),
 p=[[  2.43974538e-19]], df_denom=47, df_num=2>
>>> from scipy.import stats
>>> stats.f_oneway(y[:20],y[20:40],y[40:])
(122.07800238379976, 2.4397453793959e-19)
```

RLM Example

```
>>> norms = sm.robust.norms
>>> rlm_model = sm.RLM(y,X,M=norms.HuberT).fit()
```



Outlook

- The sandbox and GSoC 2010
  - Time series analysis and dynamic models, Panel data models, Nonparametric regression and kernel density estimators, System of equation models, and Maximum entropy estimators
- What else?
  - R-like formula framework
  - Statistics-oriented data structures and analysis
- Want to get involved?
  - Mailing list: http://groups.google.ca/group/pystatsmodels or scipy-user
  - Documentation: http://statsmodels.sourceforge.net/
  - Blog: http://scipystats.blogspot.com/

Summary

- Statsmodels is a library for statistical and econometric modeling in Python.
- Python is becoming a popular choice for statistical programming.
- A foundation for continuing development of statistics with the Python community.

For Further Reading I

J. Stachurski.
Economic Dynamics: Theory and Computation.
MIT Press, 2009.

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

C. Choirat and R. Seri.
"Econometrics with Python."
Journal of Applied Econometrics. 24(4):698–704, 2009.

For Further Reading II

A. Isaac.
"Simulating Evolutionary Games: A Python-Based Introduction."
Journal of Artificial Societies and Social Simulation. 11(38), 2008.
Available at http://jasss.soc.surrey.ac.uk/11/3/8.html