Linear models

Friday, September 20

Content

1. The statsmodels package

`statsmodels` is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration.

The online documentation is hosted at [statsmodels.org (https://www.statsmodels.org/stable/index.html)](https://www.statsmodels.org/stable/index.html)

It covered:

- Linear Regression
- Generalized Linear Models
- Generalized Estimating Equations
- Generalized Additive Models (GAM)
- Robust Linear Models
- Linear Mixed Effects Models
- Regression with Discrete Dependent Variable
- Generalized Linear Mixed Effects Models
- ANOVA
- Time Series analysis `tsa`
- Time Series Analysis by State Space Methods statespace
- Vector Autoregressions `tsa.vector_ar`
- Methods for Survival and Duration Analysis
- Statistics `stats`
- Nonparametric Methods nonparametric
- Generalized Method of Moments `gmm`
- Contingency tables
- Multiple Imputation with Chained Equations
- Multivariate Statistics multivariate
- Empirical Likelihood emplike
- Other Models miscmodels
- Distributions
- Graphics
- Input-Output iolib
- Tools
- The Datasets Package
- Sandbox
- Working with Large Data Sets
- Optimization

`statsmodels` works smoothly with the `pandas` in a way that DataFrame is the dataset form it supports by default.

Anaconda has installed `statsmodels` module by default. Before using the functions and classes inside, we need to import the `statsmodels.api` and `statsmodels.formula.api` .

In [1]:

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
import numpy as np
import matplotlib.pyplot as plt
```

The output of `statsmodels` is similiar to the output of functions in `R`. We start with the most widely used and elementary statistical methods : ordinary least square.

2. OLS

2.1. How to fit a dataset and see the result

We use the dataset `Guerry` provided by `statsmodel`. This dataset contains socio-economic variables for 86 departments (district) of France in 1830. It contains variables such as:

- Population per crime against persons
- Population per crime against property
- Literacy rates
- Donations to charity and to the clergy
- Per capita spending on Royal Lottery ...

The full variable description is available [here (https://vincentarelbundock.github.io/Rdatasets/doc/HistData/Guerry.html)](https://vincentarelbundock.github.io/Rdatasets/doc/HistData/Guerry.html).

This is a default dataset for R, so we get it with the command `sm.datasets.get_rdataset().data`

In [2]:

```
import pandas as pd
# Load data
dat = sm.datasets.get_rdataset("Guerry", "HistData").data
# list of the variables
dat.head(5)
```

Out[2]:

	dept	Region	Department	Crime_pers	Crime_prop	Literacy	Donations	Infants	Suicides
0	1	E	Ain	28870	15890	37	5098	33120	35039
1	2	N	Aisne	26226	5521	51	8901	14572	12831
2	3	C	Allier	26747	7925	13	10973	17044	114121
3	4	E	Basses-Alpes	12935	7289	46	2733	23018	14238
4	5	E	Hautes-Alpes	17488	8174	69	6962	23076	16171

5 rows × 23 columns

Below, we fit an OLS model of the relationship between lottery spending per capita, the literacy rate, and the population (logged).

In [3]:

```
# Fit regression model (using the natural log of one of the regressors)
model = smf.ols('Lottery ~ Literacy + np.log(Pop1831)', data=dat)
results = model.fit()
```

To see the results, we need an additional step:

In [4]:

```
# Inspect the results
print(results.summary())
```

```
                            OLS Regression Results
======================================================================
==========
Dep. Variable:                Lottery   R-squared:
0.348
Model:                            OLS   Adj. R-squared:
0.333
Method:                 Least Squares   F-statistic:
22.20
Date:                Thu, 19 Sep 2019   Prob (F-statistic):
1.90e-08
Time:                        19:27:38   Log-Likelihood:
-379.82
No. Observations:                  86   AIC:
765.6
Df Residuals:                      83   BIC:
773.0
Df Model:                           2
Covariance Type:            nonrobust
======================================================================
===============
                      coef    std err          t      P>|t|      [0.
025      0.975]
----------------------------------------------------------------------
---------------
Intercept         246.4341     35.233      6.995      0.000     176.
358     316.510
Literacy           -0.4889      0.128     -3.832      0.000      -0.
743      -0.235
np.log(Pop1831)   -31.3114      5.977     -5.239      0.000     -43.
199     -19.424
======================================================================
==========
Omnibus:                        3.713   Durbin-Watson:
2.019
Prob(Omnibus):                  0.156   Jarque-Bera (JB):
3.394
Skew:                          -0.487   Prob(JB):
0.183
Kurtosis:                       3.003   Cond. No.
702.
======================================================================
==========

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors
is correctly specified.
```

2.2. When the dataset is not in `DataFrame`

The dataset above is provided by the `statsmodels` package, hence it is in a format it supports. However, in many situations, the dataset is not constructed yet. In this case, we can use `numpy` arrays.

In [5]:

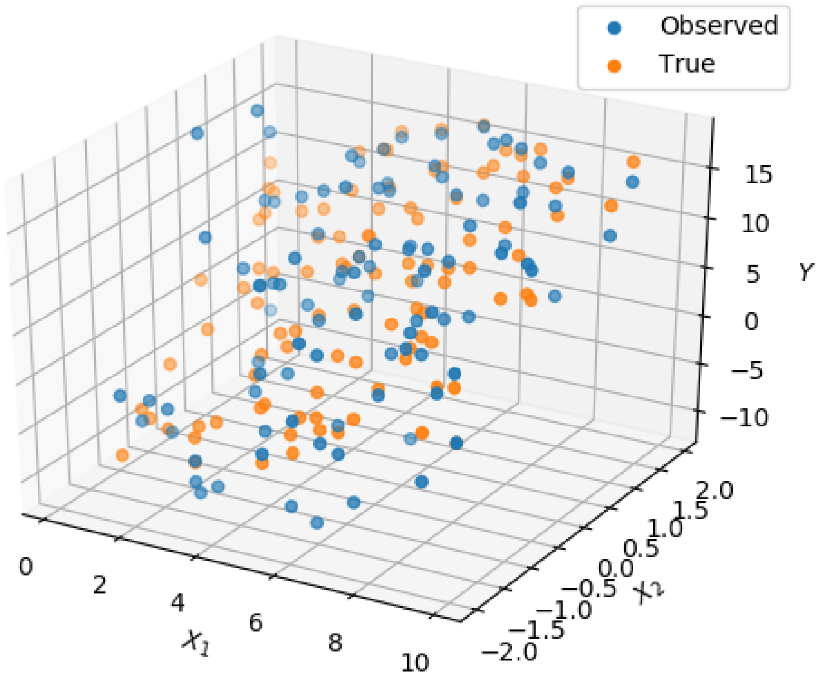
```
import numpy as np
np.random.seed(123)

# Generating artificial data (2 regressors + constant)
# X's are uniformly distributed (between 0 and 10 for X_1 and between -2 and 2 f
or X_2)
nobs = 100
X1 = np.random.uniform(low=0.0, high=10.0, size=nobs)
X2 = np.random.uniform(low=-2, high=2, size=nobs)
X = sm.add_constant(np.array([X1, X2]).T)          # Note that we need to tr
anspose the 2 x 100 matrix of variables

# True coefficients
beta = [1, 1, 5]

# Normally distributed error
e = np.random.normal(loc=0.0, scale = 5, size=nobs)

# Observed y's
y = np.dot(X, beta) + e

# True y's
y_true = np.dot(X, beta)

# Fit regression model
results = sm.OLS(y, X).fit()

# Inspect the results
print(results.summary())
```

```
                            OLS Regression Results
====================================================================
==========
Dep. Variable:                      y   R-squared:
0.609
Model:                            OLS   Adj. R-squared:
0.601
Method:                 Least Squares   F-statistic:
75.69
Date:                Thu, 19 Sep 2019   Prob (F-statistic):
1.57e-20
Time:                        19:28:11   Log-Likelihood:
-295.69
No. Observations:                 100   AIC:
597.4
Df Residuals:                      97   BIC:
605.2
Df Model:                           2
Covariance Type:            nonrobust
====================================================================
==========
                 coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------
----------
const          1.8251      1.080      1.690      0.094      -0.318
3.968
x1             0.7702      0.193      3.981      0.000       0.386
1.154
x2             4.9872      0.417     11.960      0.000       4.160
5.815
====================================================================
==========
Omnibus:                        6.909   Durbin-Watson:
1.675
Prob(Omnibus):                  0.032   Jarque-Bera (JB):
9.264
Skew:                          -0.299   Prob(JB):
0.00973
Kurtosis:                       4.366   Cond. No.
13.1
====================================================================
==========

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors
is correctly specified.
```

We can see the relationship between our variables in a 3D plot. See the [Notebook on the basicas of data handling (https://github.com/arnauddyevre/Python-for-Social-Scientists/tree/master/statistics_and_econometrics/data_handling)](https://github.com/arnauddyevre/Python-for-Social-Scientists/tree/master/statistics_and_econometrics/data_handling) for a refresher about 3D-plotting.

In [6]:

```
from mpl_toolkits.mplot3d import Axes3D  # noqa: F401 unused import
%matplotlib nbagg

np.random.seed(123)

fig = plt.figure(1)
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X1, X2, y, label="Observed")
ax.set_xlabel('$X_1$')
ax.set_ylabel('$X_2$')
ax.set_zlabel('$Y$')

ax.scatter(X1, X2, y_true, label="True")

plt.legend()
plt.show()
```



Of course, we can create a dataset and make it supported by `statsmodels` . Details can be found here: [adding a dataset (https://www.statsmodels.org/stable/dev/dataset_notes.html?highlight=statsmodels%20datasets#adding-a-dataset-an-example)](https://www.statsmodels.org/stable/dev/dataset_notes.html?highlight=statsmodels%20datasets#adding-a-dataset-an-example)

2.3. Non-linear least squares with `Scipy`

Any non-linear model can be simply estimated using `SciPy` . Let's say we want to estimated the following model:

$$y_i = f(x_i, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \alpha e^{\beta x_i} + \varepsilon_i$$
$$\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$$

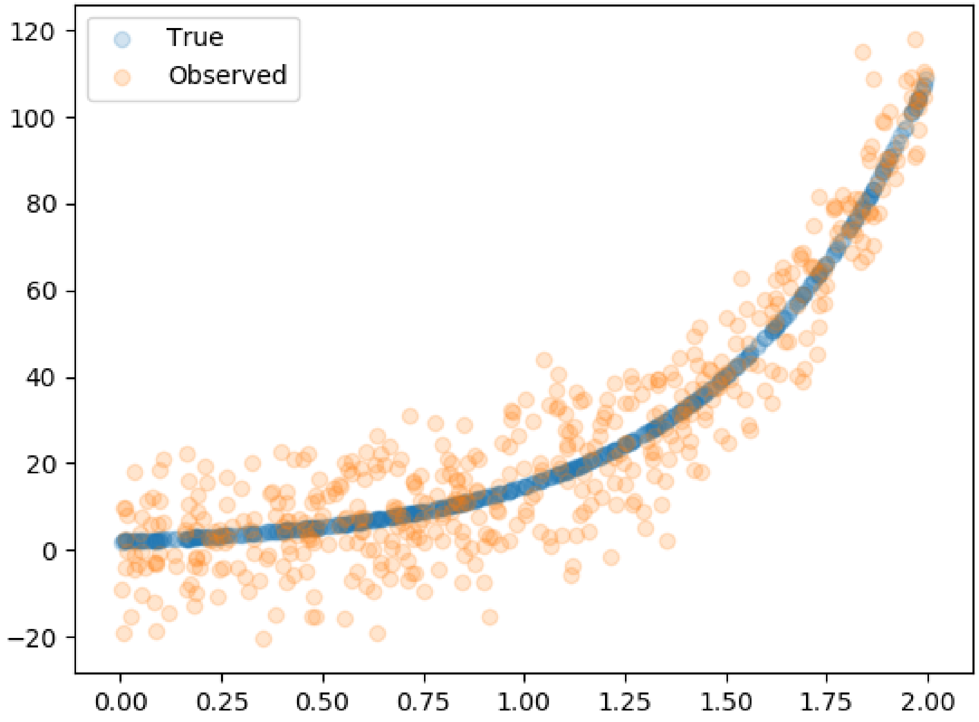
The objective function we will minimise (through numerical methods) is thus:

$$\min_{\alpha,\beta,\sigma} \quad \sum_{i=0}^{N} \left(y_i - \alpha e^{\beta x_i}\right)^2$$

We generate some data with some added noise. We will estimate our model on this data.

In [7]:

```
nobs = 500

x_obs = np.random.uniform(low=0.0, high=2.0, size=nobs)

α_true = 2
β_true = 2

ϵ = np.random.normal(loc=0.0, scale = 10, size=nobs)

βX = β_true*x_obs
y_true = α_true*np.exp(βX)
y_observed = α_true*np.exp(βX) + ϵ

# Plots
plt.figure(2)
plt.scatter(x_obs, y_true, alpha = 0.2, label="True")
plt.scatter(x_obs, y_observed, alpha = 0.2, label="Observed")
plt.legend()
```



Out[7]:

```
<matplotlib.legend.Legend at 0x1c19ee2e10>
```

To solve this problem, we first write the function calculating the residuals from observations (x_i, y_i), given parameters (α, β):

$$res(x_i, y_i, \alpha, \beta)_i = y_i - \alpha e^{\beta x_i}$$

and we initialise a vector of parameters $(\alpha_0, \beta_0) = (0, 0)$.

In [8]:

```
# Residual function
def res(params, x, y):
    temp = params[1]*x
    return y - params[0]*np.exp(temp)

# Initial values of parameters
α_0 = 0
β_0 = 0
params_0 = [α_0, β_0]
```

We can now calculate the residuals and solve the problem with SciPy's `least_squares()` function.

In [9]:

```
from scipy.optimize import least_squares

results_nlls = least_squares(res, params_0, args=(x_obs, y_observed))
print("α =", results_nlls.x[0])
print("β =", results_nlls.x[1])
```

```
α = 1.9286859713395337
β = 2.023991340502911
```

These results are fairly close to the true values!

2.4. Wald's test

Besides the fitting, `statsmodels` also supports many statsitical testing methods. Here, we show how to use *Wald's test* in `statsmodels`.

Again, we consider the dataset `Guerry`.

We want to analyse the effect of *Wealth* and *Literacy* on the _Crime*pers* and test:

> whether the coeffcients of *Wealth* and *Literacy* are the same.

In [10]:

```
formula = 'Crime_pers ~ Wealth + Literacy'
results = smf.ols(formula, dat).fit()
hypotheses = '(Wealth = Literacy)'
f_test = results.f_test(hypotheses)
print(f_test)
```

```
<F test: F=array([[0.03467668]]), p=0.8527291641569565, df_denom=83,
df_num=1>
```

3. Generalised linear model

Generalized linear models in `statsmodels` currently supports estimation using the one-parameter exponential families.

What is it?

The statistical model for each observation i is assumed to be

$Y_i \sim F_{EDM}(\cdot|\theta, \phi, w_i)$ and $\mu_i = E[Y_i|x_i] = g^{-1}(x_i' \beta)$.

where g is the link function and $F_{EDM}(\cdot|\theta, \phi, w)$ is a distribution of the family of exponential dispersion models (EDM) with natural parameter θ, scale parameter ϕ and weight :math: `w` . Its density is given by

$$f_{EDM}(y|\theta, \phi, w) = c(y, \phi, w) \exp\left(\frac{y\theta - b(\theta)}{\phi} w \right).$$

It follows that $\mu = b'(\theta)$ and $Var[Y|x] = \frac{\phi}{w} b''(\theta)$. The inverse of the first equation gives the natural parameter as a function of the expected value $\theta(\mu)$ such that

$$Var[Y_i|x_i] = \frac{\phi}{w_i} v(\mu_i)$$

with $v(\mu) = b''(\theta(\mu))$. Therefore it is said that a GLM is determined by link function g and variance function $v(\mu)$ alone (and x of course).

Note that while ϕ is the same for every observation y_i and therefore does not influence the estimation of β, the weights w_i might be different for every y_i such that the estimation of β depends on them.

Examples: binomial response $B(n, p)$

$\mu = E[Y|x] = np$
$v(\mu) = \mu - \frac{\mu^2}{n}$
$\theta(\mu) = \log \frac{p}{1-p}$
$b(\theta) = n \log(1 + e^{\theta})$

Real data

Here we use the Star98 dataset which was taken with permission from Jeff Gill (2000) *Generalized linear models: A unified approach*

In [11]:

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

import pandas as pd

star98 = sm.datasets.star98.load(as_pandas=True)
star98.data.head(5)
```

Out[11]:

	NABOVE	NBELOW	LOWINC	PERASIAN	PERBLACK	PERHISP	PERMINTE	AVYRSEXP
0	452.0	355.0	34.39730	23.299300	14.235280	11.411120	15.91837	14.70646
1	144.0	40.0	17.36507	29.328380	8.234897	9.314884	13.63636	16.08324
2	337.0	234.0	32.64324	9.226386	42.406310	13.543720	28.83436	14.59559
3	395.0	178.0	11.90953	13.883090	3.796973	11.443110	11.11111	14.38939
4	8.0	57.0	36.88889	12.187500	76.875000	7.604167	43.58974	13.90568

5 rows × 22 columns

In [12]:

```
print(sm.datasets.star98.NOTE)
```

```
::

    Number of Observations - 303 (counties in California).

    Number of Variables - 13 and 8 interaction terms.

    Definition of variables names::

        NABOVE   - Total number of students above the national media
n for the
                   math section.
        NBELOW   - Total number of students below the national media
n for the
                   math section.
        LOWINC   - Percentage of low income students
        PERASIAN - Percentage of Asian student
        PERBLACK - Percentage of black students
        PERHISP  - Percentage of Hispanic students
        PERMINTE - Percentage of minority teachers
        AVYRSEXP - Sum of teachers' years in educational service div
ided by the
                number of teachers.
        AVSALK   - Total salary budget including benefits divided by
the number
                   of full-time teachers (in thousands)
        PERSPENK - Per-pupil spending (in thousands)
        PTRATIO  - Pupil-teacher ratio.
        PCTAF    - Percentage of students taking UC/CSU prep courses
        PCTCHRT  - Percentage of charter schools
        PCTYRRND - Percentage of year-round schools

        The below variables are interaction terms of the variables d
efined
        above.

        PERMINTE_AVYRSEXP
        PEMINTE_AVSAL
        AVYRSEXP_AVSAL
        PERSPEN_PTRATIO
        PERSPEN_PCTAF
        PTRATIO_PCTAF
        PERMINTE_AVTRSEXP_AVSAL
        PERSPEN_PTRATIO_PCTAF
```

Now, we use genelized linear model to analyze the number of students above the national median for the math section

In [13]:

```
data = sm.datasets.star98.load(as_pandas=False)
data.exog = sm.add_constant(data.exog, prepend=False)
glm_binom = sm.GLM(data.endog, data.exog, family=sm.families.Binomial())
res = glm_binom.fit()
print(res.summary())
```

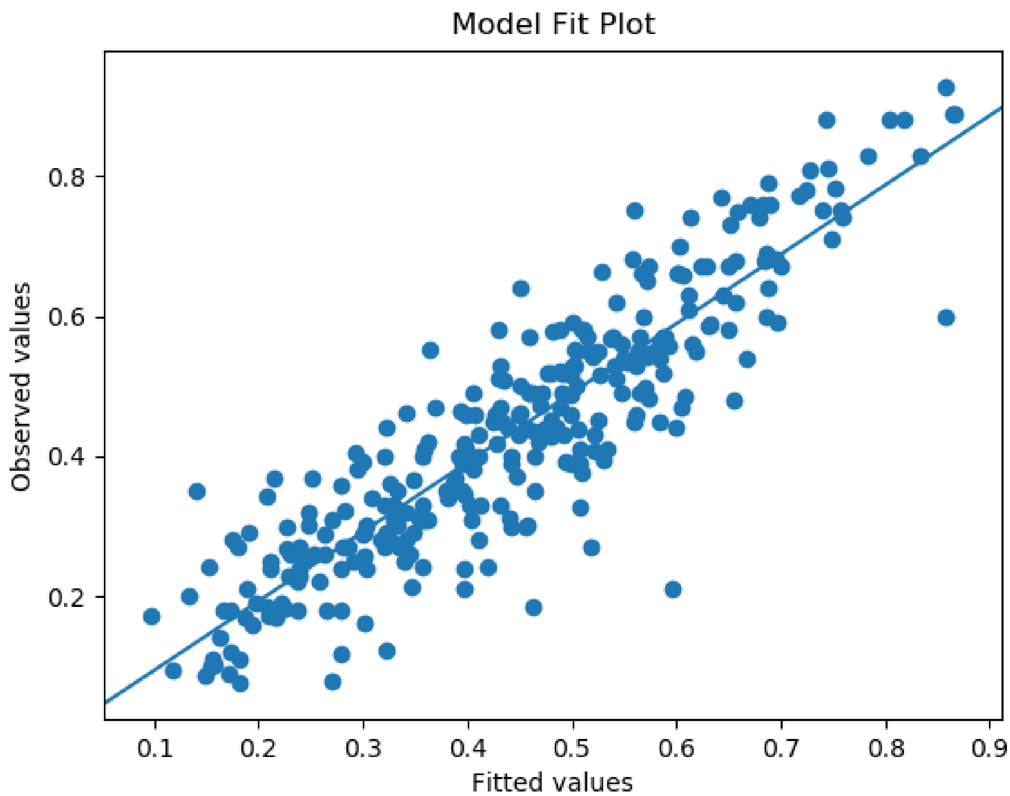
```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:           ['y1', 'y2']   No. Observations:                  303
Model:                            GLM   Df Residuals:                      282
Model Family:                Binomial   Df Model:                           20
Link Function:                  logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -2998.6
Date:            Thu, 19 Sep 2019   Deviance:                       4078.8
Time:                        19:28:30   Pearson chi2:                 4.05e+03
No. Iterations:                     5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1            -0.0168      0.000    -38.749      0.000      -0.018      -0.016
x2             0.0099      0.001     16.505      0.000       0.009       0.011
x3            -0.0187      0.001    -25.182      0.000      -0.020      -0.017
x4            -0.0142      0.000    -32.818      0.000      -0.015      -0.013
x5             0.2545      0.030      8.498      0.000       0.196       0.313
x6             0.2407      0.057      4.212      0.000       0.129       0.353
x7             0.0804      0.014      5.775      0.000       0.053       0.108
x8            -1.9522      0.317     -6.162      0.000      -2.573      -1.331
x9            -0.3341      0.061     -5.453      0.000      -0.454      -0.214
x10           -0.1690      0.033     -5.169      0.000      -0.233      -0.105
x11            0.0049      0.001      3.921      0.000       0.002       0.007
x12           -0.0036      0.000    -15.878      0.000      -0.004      -0.003
x13           -0.0141      0.002     -7.391      0.000      -0.018      -0.010
x14           -0.0040      0.000     -8.450      0.000      -0.005      -0.003
x15           -0.0039      0.001     -4.059      0.000      -0.006      -0.002
x16            0.0917      0.015      6.321      0.000       0.063       0.120
x17            0.0490      0.007      6.574      0.000       0.034       0.064
x18            0.0080      0.001      5.362      0.000       0.005       0.011
```

```
x19             0.0002   2.99e-05      7.428      0.000       0.000
0.000
x20            -0.0022      0.000     -6.445      0.000      -0.003
-0.002
const           2.9589      1.547      1.913      0.056      -0.073
5.990
======================================================================
==========
```

Also, we can see the fit plot of the glm model

In [14]:

```
nobs = res.nobs
y = data.endog[:,0]/data.endog.sum(1)
yhat = res.mu

from statsmodels.graphics.api import abline_plot

from matplotlib import pyplot as plt

fig, ax = plt.subplots()
ax.scatter(yhat, y)
line_fit = sm.OLS(y, sm.add_constant(yhat, prepend=True)).fit()
abline_plot(model_results=line_fit, ax=ax)


ax.set_title('Model Fit Plot')
ax.set_ylabel('Observed values')
ax.set_xlabel('Fitted values');
```



4. Two-stage least squares

4.1. Endogeneity

Endogeneity issues are at the central of the quantitative research in the social science. That is to say, when we use the linear regression, the dependent variable might actually affect the explaintionary variable. And once this happens, the estimates from the OLS could be largely biased.

For example, there is a two-way relationship between the institutions and the economic outcomes:

- better institutions will output labor force of higher quality which boost the economic development
- richer countries/cities can afford better institutions

To eliminate such endogeneity, two-stage least square method is one tool used by many social scientists. The idea is to find an *instrument variable* that is

- correlated with the explaintionary variable
- not correlated with the dependent variable

4.2. Real data: Acemoglu et al. (2001)

As an example, we will use the data set from Daron Acemoglu, Simon Johnson, and James A Robinson. *The colonial origins of comparative development: an empirical investigation*. The American Economic Review, 91(5):1369–1401, 2001.

In this paper, Acemoglu et al. (2001) want to study the effect of the institution quality on the economic outcomes.

The data set could be downloaded from [Quant Econ (https://lectures.quantecon.org/)](https://lectures.quantecon.org/)

In [15]:

```
import pandas as pd

# Import and select the data
df4 = pd.read_stata('https://github.com/QuantEcon/QuantEcon.lectures.code/raw/master/ols/maketable4.dta')
df4 = df4[df4['baseco'] == 1]

df4.head(5)
```

Out[15]:

	shortnam	africa	lat_abst	rich4	avexpr	logpgp95	logem4	asia	loghjypl	baseco
1	AGO	1.0	0.136667	0.0	5.363636	7.770645	5.634789	0.0	-3.411248	1.0
3	ARG	0.0	0.377778	0.0	6.386364	9.133459	4.232656	0.0	-0.872274	1.0
5	AUS	0.0	0.300000	1.0	9.318182	9.897972	2.145931	0.0	-0.170788	1.0
11	BFA	1.0	0.144444	0.0	4.454545	6.845880	5.634789	0.0	-3.540459	1.0
12	BGD	0.0	0.266667	0.0	5.136364	6.877296	4.268438	1.0	-2.063568	1.0

Acemoglu et al. (2001) use:

- economic outcome: *logpgp95* , log GDP per capita in 1995, adjusted for exchange rates
- institution quality: *avexpr* , an index of protection against expropriation on average over 1985-95
- instrument variable: *logem4* , settler mortality rates

In [16]:

```
import statsmodels.sandbox.regression.gmm as gmm

model = gmm.IV2SLS(endog=df4['logpgp95'], exog=df4['avexpr'], instrument=df4['lo
gem4'])
result = model.fit()
print(result.summary())
```

```
                          IV2SLS Regression Results
====================================================================
==========
Dep. Variable:               logpgp95   R-squared:
0.976
Model:                         IV2SLS   Adj. R-squared:
0.975
Method:                     Two Stage   F-statistic:
nan
                        Least Squares   Prob (F-statistic):
nan
Date:                Thu, 19 Sep 2019
Time:                        19:28:40
No. Observations:                  64
Df Residuals:                      63
Df Model:                           1
====================================================================
==========
                 coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------
----------
avexpr         1.2468      0.026     47.531      0.000       1.194
1.299
====================================================================
==========
Omnibus:                        0.340   Durbin-Watson:
2.052
Prob(Omnibus):                  0.844   Jarque-Bera (JB):
0.474
Skew:                          -0.152   Prob(JB):
0.789
Kurtosis:                       2.707   Cond. No.
1.00
====================================================================
==========
```