StatsModels
Statistics in Python

Ordinary Least Squares

Link to Notebook GitHub

In [1]:
```
from __future__ import print_function
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.sandbox.regression.predstd import wls_prediction

np.random.seed(9876789)
```

OLS estimation

Artificial data:

In [2]:
```
nsample = 100
x = np.linspace(0, 10, 100)
X = np.column_stack((x, x**2))
beta = np.array([1, 0.1, 10])
e = np.random.normal(size=nsample)
```

Our model needs an intercept so we add a column of 1s:

In [3]:
```
X = sm.add_constant(X)
y = np.dot(X, beta) + e
```

Fit and summary:

In [4]:
```
model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:
Model:                            OLS   Adj. R-squared:
Method:                 Least Squares   F-statistic:                    4
Date:                Sun, 01 Feb 2015   Prob (F-statistic):             2
Time:                        09:32:32   Log-Likelihood:
No. Observations:                 100   AIC:
Df Residuals:                      97   BIC:
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Con
------------------------------------------------------------------------------
const          1.3423      0.313      4.292      0.000       0.722
x1            -0.0402      0.145     -0.278      0.781      -0.327
x2            10.0103      0.014    715.745      0.000       9.982
==============================================================================
Omnibus:                        2.042   Durbin-Watson:
Prob(Omnibus):                  0.360   Jarque-Bera (JB):
Skew:                           0.234   Prob(JB):
Kurtosis:                       2.519   Cond. No.
==============================================================================

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

Quantities of interest can be extracted directly from the fitted model. Typ examples:

In [5]:
```
print('Parameters: ', results.params)
print('R2: ', results.rsquared)
```

```
Parameters:  [  1.3423  -0.0402  10.0103]
R2:  0.999987936503
```

OLS non-linear curve but linear in parame

We simulate artificial data with a non-linear relationship between x and y:

In [6]:
```
nsample = 50
sig = 0.5
x = np.linspace(0, 20, nsample)
X = np.column_stack((x, np.sin(x), (x-5)**2, np.ones(nsample)))
beta = [0.5, 0.5, -0.02, 5.]

y_true = np.dot(X, beta)
y = y_true + sig * np.random.normal(size=nsample)
```

Fit and summary:

In [7]:
```
res = sm.OLS(y, X).fit()
print(res.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:
Model:                            OLS   Adj. R-squared:
Method:                 Least Squares   F-statistic:
Date:                Sun, 01 Feb 2015   Prob (F-statistic):
Time:                        09:32:33   Log-Likelihood:
No. Observations:                  50   AIC:
Df Residuals:                      46   BIC:
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Con
------------------------------------------------------------------------------
x1             0.4687      0.026     17.751      0.000         0.416
x2             0.4836      0.104      4.659      0.000         0.275
x3            -0.0174      0.002     -7.507      0.000        -0.022
const          5.2058      0.171     30.405      0.000         4.861
==============================================================================
Omnibus:                        0.655   Durbin-Watson:
Prob(Omnibus):                  0.721   Jarque-Bera (JB):
Skew:                           0.207   Prob(JB):
Kurtosis:                       3.026   Cond. No.
==============================================================================

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

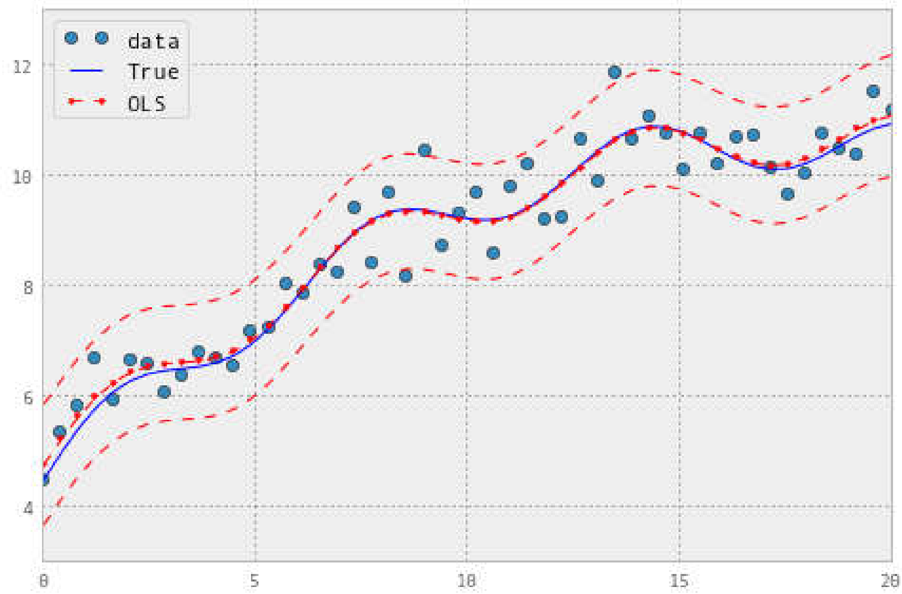
Extract other quantities of interest:

In [8]:
```
print('Parameters: ', res.params)
print('Standard errors: ', res.bse)
print('Predicted values: ', res.predict())
```

```
Parameters:  [ 0.4687  0.4836 -0.0174  5.2058]
Standard errors:  [ 0.0264  0.1038  0.0023  0.1712]
Predicted values:  [  4.7707   5.2221   5.6362   5.9866   6.2564   6.4
   6.6243   6.6518   6.7138   6.8341   7.0262   7.2905   7.6149   7.97
   8.3446   8.6876   8.9764   9.19     9.3187   9.3659   9.3474   9.28
   9.2217   9.1775   9.1834   9.2571   9.4044   9.6181   9.879   10.15
  10.4266  10.6505  10.8063  10.8795  10.8683  10.7838  10.6483  10.49
  10.3452  10.2393  10.1957  10.2249  10.3249  10.4808  10.6678  10.85
  11.0101  11.1058]
```

Draw a plot to compare the true relationship to OLS predictions. Confidence
the `wls_prediction_std` command.

In [9]:
```
prstd, iv_l, iv_u = wls_prediction_std(res)

fig, ax = plt.subplots(figsize=(8,6))

ax.plot(x, y, 'o', label="data")
ax.plot(x, y_true, 'b-', label="True")
ax.plot(x, res.fittedvalues, 'r--.', label="OLS")
ax.plot(x, iv_u, 'r--')
ax.plot(x, iv_l, 'r--')
ax.legend(loc='best');
```



OLS with dummy variables

We generate some artificial data. There are 3 groups which will be mod omitted/benchmark category.

In [10]:
```
nsample = 50
groups = np.zeros(nsample, int)
groups[20:40] = 1
groups[40:] = 2
#dummy = (groups[:,None] == np.unique(groups)).astype(float)

dummy = sm.categorical(groups, drop=True)
x = np.linspace(0, 20, nsample)
# drop reference category
X = np.column_stack((x, dummy[:,1:]))
X = sm.add_constant(X, prepend=False)

beta = [1., 3, -3, 10]
y_true = np.dot(X, beta)
e = np.random.normal(size=nsample)
y = y_true + e
```

Inspect the data:

In [11]:
```
print(X[:5,:])
print(y[:5])
print(groups)
print(dummy[:5,:])
```

```
[[ 0.      0.      0.      1.    ]
 [ 0.4082  0.      0.      1.    ]
 [ 0.8163  0.      0.      1.    ]
 [ 1.2245  0.      0.      1.    ]
 [ 1.6327  0.      0.      1.    ]]
[  9.2822  10.5048  11.8439  10.3851  12.3794]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 2 2 2 2 2 2 2 2 2 2]
[[ 1.  0.  0.]
 [ 1.  0.  0.]
 [ 1.  0.  0.]
 [ 1.  0.  0.]
 [ 1.  0.  0.]]
```

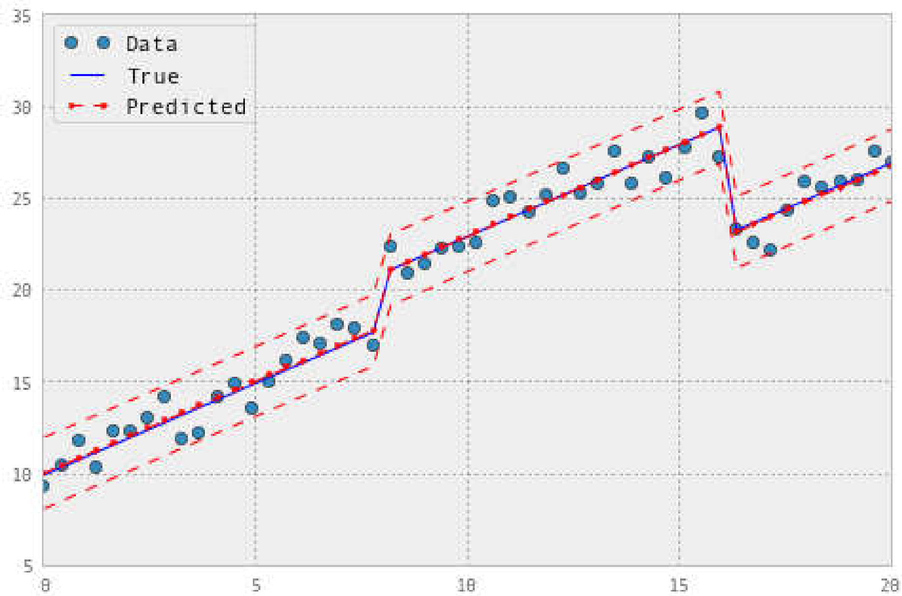
Fit and summary:

In [12]:
```
res2 = sm.OLS(y, X).fit()
print(res.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:
Model:                            OLS   Adj. R-squared:
Method:                 Least Squares   F-statistic:
Date:                Sun, 01 Feb 2015   Prob (F-statistic):
Time:                        09:32:35   Log-Likelihood:
No. Observations:                  50   AIC:
Df Residuals:                      46   BIC:
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Con
------------------------------------------------------------------------------
x1             0.4687      0.026     17.751      0.000       0.416
x2             0.4836      0.104      4.659      0.000       0.275
x3            -0.0174      0.002     -7.507      0.000      -0.022
const          5.2058      0.171     30.405      0.000       4.861
==============================================================================
Omnibus:                        0.655   Durbin-Watson:
Prob(Omnibus):                  0.721   Jarque-Bera (JB):
Skew:                           0.207   Prob(JB):
Kurtosis:                       3.026   Cond. No.
==============================================================================

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

Draw a plot to compare the true relationship to OLS predictions:

In [13]:
```
prstd, iv_l, iv_u = wls_prediction_std(res2)

fig, ax = plt.subplots(figsize=(8,6))

ax.plot(x, y, 'o', label="Data")
ax.plot(x, y_true, 'b-', label="True")
ax.plot(x, res2.fittedvalues, 'r--.', label="Predicted")
ax.plot(x, iv_u, 'r--')
ax.plot(x, iv_l, 'r--')
legend = ax.legend(loc="best")
```



Joint hypothesis test

F test

We want to test the hypothesis that both coefficients on the dummy variables
An F test leads us to strongly reject the null hypothesis of identical constant ir

In [14]:
```
R = [[0, 1, 0, 0], [0, 0, 1, 0]]
print(np.array(R))
print(res2.f_test(R))
```

```
[[0 1 0 0]
 [0 0 1 0]]
<F test: F=array([[ 145.4927]]), p=1.28344196173e-20, df_denom=46, df_
```

You can also use formula-like syntax to test hypotheses

In [15]:
```
print(res2.f_test("x2 = x3 = 0"))
```

```
<F test: F=array([[ 145.4927]]), p=1.28344196173e-20, df_denom=46, df_
```

Small group effects

If we generate artificial data with smaller group effects, the T test can no long

In [16]:
```
beta = [1., 0.3, -0.0, 10]
y_true = np.dot(X, beta)
y = y_true + np.random.normal(size=nsample)

res3 = sm.OLS(y, X).fit()
```

In [17]:
```
print(res3.f_test(R))
```

```
<F test: F=array([[ 1.2249]]), p=0.303186441063, df_denom=46, df_num=2
```

In [18]:
```
print(res3.f_test("x2 = x3 = 0"))
```

```
<F test: F=array([[ 1.2249]]), p=0.303186441063, df_denom=46, df_num=2
```

Multicollinearity

The Longley dataset is well known to have high multicollinearity. That is, the e is problematic because it can affect the stability of our coefficient estin specification.

In [19]:
```
from statsmodels.datasets.longley import load_pandas
y = load_pandas().endog
X = load_pandas().exog
X = sm.add_constant(X)
```

Fit and summary:

In [20]:

```
ols_model = sm.OLS(y, X)
ols_results = ols_model.fit()
print(ols_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 TOTEMP   R-squared:
Model:                            OLS   Adj. R-squared:
Method:                 Least Squares   F-statistic:
Date:                Sun, 01 Feb 2015   Prob (F-statistic):
Time:                        09:32:37   Log-Likelihood:
No. Observations:                  16   AIC:
Df Residuals:                       9   BIC:
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Con
------------------------------------------------------------------------------
const      -3.482e+06    8.9e+05     -3.911      0.004      -5.5e+06 -
GNPDEFL       15.0619     84.915      0.177      0.863      -177.029
GNP           -0.0358      0.033     -1.070      0.313        -0.112
UNEMP         -2.0202      0.488     -4.136      0.003        -3.125
ARMED         -1.0332      0.214     -4.822      0.001        -1.518
POP           -0.0511      0.226     -0.226      0.826        -0.563
YEAR        1829.1515    455.478      4.016      0.003       798.788
==============================================================================
Omnibus:                        0.749   Durbin-Watson:
Prob(Omnibus):                  0.688   Jarque-Bera (JB):
Skew:                           0.420   Prob(JB):
Kurtosis:                       2.434   Cond. No.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
[2] The condition number is large, 4.86e+09. This might indicate that
strong multicollinearity or other numerical problems.

/usr/local/lib/python2.7/dist-packages/scipy/stats/stats.py:1206: User
  int(n))
```

Condition number

One way to assess multicollinearity is to compute the condition number. Va
The first step is to normalize the independent variables to have unit length:

In [21]:

```
norm_x = X.values
for i, name in enumerate(X):
    if name == "const":
        continue
    norm_x[:,i] = X[name]/np.linalg.norm(X[name])
norm_xtx = np.dot(norm_x.T,norm_x)
```

Then, we take the square root of the ratio of the biggest to the smallest eigen

In [22]:
```
eigs = np.linalg.eigvals(norm_xtx)
condition_number = np.sqrt(eigs.max() / eigs.min())
print(condition_number)
```

```
56240.8689371
```

Dropping an observation

Greene also points out that dropping a single observation can have a dramati

In [23]:
```
ols_results2 = sm.OLS(y.ix[:14], X.ix[:14]).fit()
print("Percentage change %4.2f%%\n"*7 % tuple([i for i in (ols_re
```

```
Percentage change -13.35%
Percentage change -236.18%
Percentage change -23.69%
Percentage change -3.36%
Percentage change -7.26%
Percentage change -200.46%
Percentage change -13.34%
```

We can also look at formal statistics for this such as the DFBETAS -- a stanc changes when that observation is left out.

In [24]:
```
infl = ols_results.get_influence()
```

In general we may consider DBETAS in absolute value greater than \(2/\sqrt{N

In [25]:
```
2./len(X)**.5
```

Out[25]: 0.5

In [26]:
```
print(infl.summary_frame().filter(regex="dfb"))
```

```
    dfb_const  dfb_GNPDEFL   dfb_GNP  dfb_UNEMP  dfb_ARMED   dfb_POP
0   -0.016406    -0.234566 -0.045095  -0.121513  -0.149026  0.211057
1   -0.020608    -0.289091  0.124453   0.156964   0.287700 -0.161890
2   -0.008382     0.007161 -0.016799   0.009575   0.002227  0.014871
3    0.018093     0.907968 -0.500022  -0.495996   0.089996  0.711142 -
4    1.871260    -0.219351  1.611418   1.561520   1.169337 -1.081513 -
5   -0.321373    -0.077045 -0.198129  -0.192961  -0.430626  0.079916
6    0.315945    -0.241983  0.438146   0.471797  -0.019546 -0.448515 -
7    0.015816    -0.002742  0.018591   0.005064  -0.031320 -0.015823 -
8   -0.004019    -0.045687  0.023708   0.018125   0.013683 -0.034770
9   -1.018242    -0.282131 -0.412621  -0.663904  -0.715020 -0.229501
10   0.030947    -0.024781  0.029480   0.035361   0.034508 -0.014194 -
11   0.005987    -0.079727  0.030276  -0.008883  -0.006854 -0.010693 -
12  -0.135883     0.092325 -0.253027  -0.211465   0.094720  0.331351
13   0.032736    -0.024249  0.017510   0.033242   0.090655  0.007634 -
14   0.305868     0.148070  0.001428   0.169314   0.253431  0.342982 -
15  -0.538323     0.432004 -0.261262  -0.143444  -0.360890 -0.467296

[16 rows x 7 columns]
```