Fitting models using R-style formulas

Since version 0.5.0, `statsmodels` allows users to fit statistical models using R-style formulas. Internally, `statsmodels` uses the patsy [https://patsy.readthedocs.io/en/latest/] package to convert formulas and data to the matrices that are used in model fitting. The formula framework is quite powerful; this tutorial only scratches the surface. A full description of the formula language can be found in the `patsy` docs:

- Patsy formula language description [https://patsy.readthedocs.io/en/latest/]

Loading modules and functions

```
In [1]: import statsmodels.api as sm

In [2]: import statsmodels.formula.api as smf

In [3]: import numpy as np

In [4]: import pandas
```

Notice that we called `statsmodels.formula.api` in addition to the usual `statsmodels.api`. In fact, `statsmodels.api` is used here only to load the dataset. The `formula.api` hosts many of the same functions found in `api` (e.g. OLS, GLM), but it also holds lower case counterparts for most of these models. In general, lower case models accept `formula` and `df` arguments, whereas upper case ones take `endog` and `exog` design matrices. `formula` accepts a string which describes the model in terms of a `patsy` formula. `df` takes a pandas [https://pandas.pydata.org/] data frame.

`dir(smf)` will print a list of available models.

Formula-compatible models have the following generic call signature: `(formula, data, subset=None, *args, **kwargs)`

OLS regression using formulas

To begin, we fit the linear model described on the Getting Started [gettingstarted.html] page. Download the data, subset columns, and list-wise delete to remove missing observations:

```
In [5]: df = sm.datasets.get_rdataset("Guerry", "HistData").data

In [6]: df = df[['Lottery', 'Literacy', 'Wealth',
'Region']].dropna()

In [7]: df.head()
Out[7]:
   Lottery  Literacy  Wealth Region
0       41        37      73      E
1       38        51      22      N
2       66        13      61      C
3       80        46      76      E
4       79        69      83      E
```

Fit the model:

```
In [8]: mod = smf.ols(formula='Lottery ~ Literacy + Wealth +
Region', data=df)

In [9]: res = mod.fit()

In [10]: print(res.summary())
                            OLS Regression Results
======================================================================

Dep. Variable:                Lottery   R-squared:
0.338
Model:                            OLS   Adj. R-squared:
0.287
Method:                 Least Squares   F-statistic:
6.636
Date:                Tue, 02 Feb 2021   Prob (F-statistic):
1.07e-05
Time:                        07:06:45   Log-Likelihood:
-375.30
```

```
No. Observations:                  85   AIC:
764.6
Df Residuals:                      78   BIC:
781.7
Df Model:                           6
Covariance Type:            nonrobust
=======================================================================

                  coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------
-----------
Intercept      38.6517      9.456      4.087      0.000      19.826
57.478
Region[T.E]   -15.4278      9.727     -1.586      0.117     -34.793
3.938
Region[T.N]   -10.0170      9.260     -1.082      0.283     -28.453
8.419
Region[T.S]    -4.5483      7.279     -0.625      0.534     -19.039
9.943
Region[T.W]   -10.0913      7.196     -1.402      0.165     -24.418
4.235
Literacy       -0.1858      0.210     -0.886      0.378      -0.603
0.232
Wealth          0.4515      0.103      4.390      0.000       0.247
0.656
=======================================================================

Omnibus:                        3.049   Durbin-Watson:
1.785
Prob(Omnibus):                  0.218   Jarque-Bera (JB):
2.694
Skew:                          -0.340   Prob(JB):
0.260
Kurtosis:                       2.454   Cond. No.
371.
=======================================================================


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

Categorical variables

Looking at the summary printed above, notice that `patsy` determined that elements of *Region* were text strings, so it treated *Region* as a categorical variable. `patsy`'s default is also to include an intercept, so we automatically dropped one of the *Region* categories.

If *Region* had been an integer variable that we wanted to treat explicitly as categorical, we could have done so by using the `C()` operator:

```
In [11]: res = smf.ols(formula='Lottery ~ Literacy + Wealth +
C(Region)', data=df).fit()

In [12]: print(res.params)
Intercept          38.651655
C(Region)[T.E]    -15.427785
C(Region)[T.N]    -10.016961
C(Region)[T.S]     -4.548257
C(Region)[T.W]    -10.091276
Literacy           -0.185819
Wealth              0.451475
dtype: float64
```

Examples more advanced features `patsy`'s categorical variables function can be found here: Patsy: Contrast Coding Systems for categorical variables [contrasts.html]

Operators

We have already seen that "~" separates the left-hand side of the model from the right-hand side, and that "+" adds new columns to the design matrix.

Removing variables

The "-" sign can be used to remove columns/variables. For instance, we can remove the intercept from a model by:

```
In [13]: res = smf.ols(formula='Lottery ~ Literacy + Wealth +
C(Region) -1 ', data=df).fit()

In [14]: print(res.params)
C(Region)[C]    38.651655
C(Region)[E]    23.223870
```

```
C(Region)[N]    28.634694
C(Region)[S]    34.103399
C(Region)[W]    28.560379
Literacy        -0.185819
Wealth           0.451475
dtype: float64
```

Multiplicative interactions

":" adds a new column to the design matrix with the product of the other two columns. "*" will also include the individual columns that were multiplied together:

```
In [15]: res1 = smf.ols(formula='Lottery ~ Literacy : Wealth - 1',
data=df).fit()

In [16]: res2 = smf.ols(formula='Lottery ~ Literacy * Wealth - 1',
data=df).fit()

In [17]: print(res1.params)
Literacy:Wealth    0.018176
dtype: float64

In [18]: print(res2.params)
Literacy           0.427386
Wealth             1.080987
Literacy:Wealth   -0.013609
dtype: float64
```

Many other things are possible with operators. Please consult the patsy docs [https://patsy.readthedocs.io/en/latest/formulas.html] to learn more.

Functions

You can apply vectorized functions to the variables in your model:

```
In [19]: res = smf.ols(formula='Lottery ~ np.log(Literacy)',
data=df).fit()

In [20]: print(res.params)
Intercept           115.609119
```

```
np.log(Literacy)    -20.393959
dtype: float64
```

Define a custom function:

```
In [21]: def log_plus_1(x):
   ....:     return np.log(x) + 1.0
   ....:

In [22]: res = smf.ols(formula='Lottery ~ log_plus_1(Literacy)',
data=df).fit()

In [23]: print(res.params)
Intercept               136.003079
log_plus_1(Literacy)    -20.393959
dtype: float64
```

Namespaces

Notice that all of the above examples use the calling namespace to look for the functions to apply. The namespace used can be controlled via the `eval_env` keyword. For example, you may want to give a custom namespace using the `patsy:patsy.EvalEnvironment` or you may want to use a "clean" namespace, which we provide by passing `eval_func=-1`. The default is to use the caller's namespace. This can have (un)expected consequences, if, for example, someone has a variable names `C` in the user namespace or in their data structure passed to `patsy`, and `C` is used in the formula to handle a categorical variable. See the Patsy API Reference [https://patsy.readthedocs.io/en/latest/API-reference.html] for more information.

Using formulas with models that do not (yet) support them

Even if a given `statsmodels` function does not support formulas, you can still use `patsy`'s formula language to produce design matrices. Those matrices can then be fed to the fitting function as `endog` and `exog` arguments.

To generate `numpy` arrays:

```
In [24]: import patsy

In [25]: f = 'Lottery ~ Literacy * Wealth'

In [26]: y, X = patsy.dmatrices(f, df, return_type='matrix')

In [27]: print(y[:5])
[[41.]
 [38.]
 [66.]
 [80.]
 [79.]]

In [28]: print(X[:5])
[[1.000e+00 3.700e+01 7.300e+01 2.701e+03]
 [1.000e+00 5.100e+01 2.200e+01 1.122e+03]
 [1.000e+00 1.300e+01 6.100e+01 7.930e+02]
 [1.000e+00 4.600e+01 7.600e+01 3.496e+03]
 [1.000e+00 6.900e+01 8.300e+01 5.727e+03]]
```

`y` and `X` would be instances of `patsy.DesignMatrix` which is a subclass of `numpy.ndarray`.

To generate pandas data frames:

```
In [29]: f = 'Lottery ~ Literacy * Wealth'

In [30]: y, X = patsy.dmatrices(f, df, return_type='dataframe')

In [31]: print(y[:5])
   Lottery
0     41.0
1     38.0
2     66.0
3     80.0
4     79.0

In [32]: print(X[:5])
   Intercept  Literacy  Wealth  Literacy:Wealth
0        1.0      37.0    73.0           2701.0
1        1.0      51.0    22.0           1122.0
2        1.0      13.0    61.0            793.0
3        1.0      46.0    76.0           3496.0
4        1.0      69.0    83.0           5727.0
```

```
In [33]: print(sm.OLS(y, X).fit().summary())
                            OLS Regression Results
======================================================================

Dep. Variable:                Lottery   R-squared:
0.309
Model:                            OLS   Adj. R-squared:
0.283
Method:                 Least Squares   F-statistic:
12.06
Date:                Tue, 02 Feb 2021   Prob (F-statistic):
1.32e-06
Time:                        07:06:45   Log-Likelihood:
-377.13
No. Observations:                  85   AIC:
762.3
Df Residuals:                      81   BIC:
772.0
Df Model:                           3
Covariance Type:            nonrobust
======================================================================

                      coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------
---------------
Intercept          38.6348     15.825      2.441      0.017
7.149      70.121
Literacy           -0.3522      0.334     -1.056      0.294
-1.016       0.312
Wealth              0.4364      0.283      1.544      0.126
-0.126       0.999
Literacy:Wealth    -0.0005      0.006     -0.085      0.933
-0.013       0.012
======================================================================

Omnibus:                        4.447   Durbin-Watson:
1.953
Prob(Omnibus):                  0.108   Jarque-Bera (JB):
3.228
Skew:                          -0.332   Prob(JB):
0.199
Kurtosis:                       2.314   Cond. No.
1.40e+04
======================================================================


Notes:
```

```
[1] Standard Errors assume that the covariance matrix of the errors
is correctly specified.
[2] The condition number is large, 1.4e+04. This might indicate that
there are
strong multicollinearity or other numerical problems.
```