Lec 18 - patsy + statsmodels

Statistical Computing and Computation

Sta 663 | Spring 2022

Dr. Colin Rundel

patsy

patsy

> `patsy` is a Python package for describing statistical models (especially linear models, or models that have a linear component) and building design matrices. It is closely inspired by and compatible with the formula mini-language used in R and S.
>
> ...
>
> Patsy's goal is to become the standard high-level interface to describing statistical models in Python, regardless of what particular model or library is being used underneath.

Formulas

```
from patsy import ModelDesc

ModelDesc.from_formula("y ~ a + a:b + np.log(x)")
```

```
## ModelDesc(lhs_termlist=[Term([EvalFactor('y')])],
##           rhs_termlist=[Term([]),
##                         Term([EvalFactor('a')]),
##                         Term([EvalFactor('a'), EvalFactor('b')]),
##                         Term([EvalFactor('np.log(x)')])])
```

```
ModelDesc.from_formula("y ~ a*b + np.log(x) - 1")
```

```
## ModelDesc(lhs_termlist=[Term([EvalFactor('y')])],
##           rhs_termlist=[Term([EvalFactor('a')]),
##                         Term([EvalFactor('b')]),
##                         Term([EvalFactor('a'), EvalFactor('b')]),
##                         Term([EvalFactor('np.log(x)')])])
```

Model matrix

```
from patsy import demo_data, dmatrix, dmatrices
```

```
data = demo_data("y", "a", "b", "x1", "x2")
data
```

```
## {'a': ['a1', 'a1', 'a2', 'a2', 'a1', 'a1', 'a2'
```

```
pd.DataFrame(data)
```

```
##      a   b        x1        x2         y
## 0   a1  b1  1.764052 -0.103219  1.494079
## 1   a1  b2  0.400157  0.410599 -0.205158
## 2   a2  b1  0.978738  0.144044  0.313068
## 3   a2  b2  2.240893  1.454274 -0.854096
## 4   a1  b1  1.867558  0.761038 -2.552990
## 5   a1  b2 -0.977278  0.121675  0.653619
## 6   a2  b1  0.950088  0.443863  0.864436
## 7   a2  b2 -0.151357  0.333674 -0.742165
```

```
dmatrix("a + a:b + np.exp(x1)", data)
```

```
## DesignMatrix with shape (8, 5)
##   Intercept  a[T.a2]  a[a1]:b[T.b2]  a[a2]:b[T.
##           1        0              0
##           1        0              1
##           1        1              0
##           1        1              0
##           1        0              0
##           1        0              1
##           1        1              0
##           1        1              0
##   Terms:
##     'Intercept' (column 0)
##     'a' (column 1)
##     'a:b' (columns 2:4)
##     'np.exp(x1)' (column 4)
```

Note the `T.` in `a[T.a2]` is there to indicate treatment coding (i.e. typical dummy coding)

Model matrices

```
y, x  = dmatrices("y ~ a + a:b + np.exp(x1)", data)
```

```
y
```

```
## DesignMatrix with shape (8, 1)
##          y
##    1.49408
##   -0.20516
##    0.31307
##   -0.85410
##   -2.55299
##    0.65362
##    0.86444
##   -0.74217
##   Terms:
##     'y' (column 0)
```

```
x
```

```
## DesignMatrix with shape (8, 5)
##   Intercept  a[T.a2]  a[a1]:b[T.b2]  a[a2]:b[T.
##           1        0              0
##           1        0              1
##           1        1              0
##           1        1              0
##           1        0              0
##           1        0              1
##           1        1              0
##           1        1              0
##   Terms:
##     'Intercept' (column 0)
##     'a' (column 1)
##     'a:b' (columns 2:4)
##     'np.exp(x1)' (column 4)
```

as DataFrames

```
dmatrix("a + a:b + np.exp(x1)", data, return_type='dataframe')
```

```
##    Intercept  a[T.a2]  a[a1]:b[T.b2]  a[a2]:b[T.b2]  np.exp(x1)
## 0        1.0      0.0            0.0            0.0    5.836039
## 1        1.0      0.0            1.0            0.0    1.492059
## 2        1.0      1.0            0.0            0.0    2.661096
## 3        1.0      1.0            0.0            1.0    9.401725
## 4        1.0      0.0            0.0            0.0    6.472471
## 5        1.0      0.0            1.0            0.0    0.376334
## 6        1.0      1.0            0.0            0.0    2.585938
## 7        1.0      1.0            0.0            1.0    0.859541
```

Formula Syntax

Code	Description	Example
`+`	unions terms on the left and right	`a+a` ⇒ `a`
`-`	removes terms on the right from terms on the left	`a+b-a` ⇒ `b`
`:`	constructs interactions between each term on the left and right	`(a+b):c` ⇒ `a:c + b:c`
`*`	short-hand for terms and their interactions	`a*b` ⇒ `a + b + a:b`
`/`	short-hand for left terms and their interactions with right terms	`a/b` ⇒ `a + a:b`
`I()`	used for calculating arithmetic calculations	`I(x1 + x2)`
`Q()`	used to quote column names, e.g. columns with spaces or symbols	`Q('bad name!')`

Examples

```
dmatrix("x:y", demo_data("x","y","z"))
```

```
## DesignMatrix with shape (5, 2)
##   Intercept      x:y
##           1 -1.72397
##           1  0.38018
##           1 -0.14814
##           1 -0.23130
##           1  0.76682
##   Terms:
##     'Intercept' (column 0)
##     'x:y' (column 1)
```

```
dmatrix("x*y", demo_data("x","y","z"))
```

```
## DesignMatrix with shape (5, 4)
##   Intercept       x        y      x:y
##           1 1.76405 -0.97728 -1.72397
##           1 0.40016  0.95009  0.38018
##           1 0.97874 -0.15136 -0.14814
##           1 2.24089 -0.10322 -0.23130
##           1 1.86756  0.41060  0.76682
##   Terms:
##     'Intercept' (column 0)
##     'x' (column 1)
##     'y' (column 2)
##     'x:y' (column 3)
```

```
dmatrix("x/y", demo_data("x","y","z"))
```

```
## DesignMatrix with shape (5, 3)
##   Intercept       x      x:y
##           1 1.76405 -1.72397
##           1 0.40016  0.38018
##           1 0.97874 -0.14814
##           1 2.24089 -0.23130
##           1 1.86756  0.76682
##   Terms:
##     'Intercept' (column 0)
##     'x' (column 1)
##     'x:y' (column 2)
```

```
dmatrix("x*(y+z)", demo_data("x","y","z"))
```

```
## DesignMatrix with shape (5, 6)
##   Intercept       x        y       z      x:y     x:z
##           1 1.76405 -0.97728 0.14404 -1.72397 0.25410
##           1 0.40016  0.95009 1.45427  0.38018 0.58194
##           1 0.97874 -0.15136 0.76104 -0.14814 0.74486
##           1 2.24089 -0.10322 0.12168 -0.23130 0.27266
##           1 1.86756  0.41060 0.44386  0.76682 0.82894
##   Terms:
##     'Intercept' (column 0)
##     'x' (column 1)
##     'y' (column 2)
##     'z' (column 3)
##     'x:y' (column 4)
##     'x:z' (column 5)
```

Intercept Examples

```
dmatrix("x", demo_data("x","y","z"))
```

```
## DesignMatrix with shape (5, 2)
##   Intercept        x
##           1  1.76405
##           1  0.40016
##           1  0.97874
##           1  2.24089
##           1  1.86756
##   Terms:
##     'Intercept' (column 0)
##     'x' (column 1)
```

```
dmatrix("x-1", demo_data("x","y","z"))
```

```
## DesignMatrix with shape (5, 1)
##         x
##   1.76405
##   0.40016
##   0.97874
##   2.24089
##   1.86756
##   Terms:
##     'x' (column 0)
```

```
dmatrix("-1 + x", demo_data("x","y","z"))
```

```
## DesignMatrix with shape (5, 1)
##         x
##   1.76405
```

```
dmatrix("x+0", demo_data("x","y","z"))
```

```
## DesignMatrix with shape (5, 1)
##         x
##   1.76405
##   0.40016
##   0.97874
##   2.24089
##   1.86756
##   Terms:
##     'x' (column 0)
```

```
dmatrix("x-0", demo_data("x","y","z"))
```

```
## DesignMatrix with shape (5, 2)
##   Intercept        x
##           1  1.76405
##           1  0.40016
##           1  0.97874
##           1  2.24089
##           1  1.86756
##   Terms:
##     'Intercept' (column 0)
##     'x' (column 1)
```

```
dmatrix("x - (-0)", demo_data("x","y","z"))
```

```
## DesignMatrix with shape (5, 1)
##         x
##   1.76405
```

Design Info

One of the keep features of the design matrix object is that it retains all the necessary details (including stateful transforms) that are necessary to apply to new data inputs (e.g. for prediction).

```
d = dmatrix("a + a:b + np.exp(x1)", data, return_type='dataframe')
d.design_info
```

```
## DesignInfo(['Intercept',
##              'a[T.a2]',
##              'a[a1]:b[T.b2]',
##              'a[a2]:b[T.b2]',
##              'np.exp(x1)'],
##             factor_infos={EvalFactor('a'): FactorInfo(factor=EvalFactor('a'),
##                                          type='categorical',
##                                          state=<factor state>,
##                                          categories=('a1', 'a2')),
##                           EvalFactor('b'): FactorInfo(factor=EvalFactor('b'),
##                                          type='categorical',
##                                          state=<factor state>,
##                                          categories=('b1', 'b2')),
##                           EvalFactor('np.exp(x1)'): FactorInfo(factor=EvalFactor('np.exp(x1)'),
##                                          type='numerical',
##                                          state=<factor state>,
```

Stateful transforms

```
data = {"x1": np.random.normal(size=10)}
new_data = {"x1": np.random.normal(size=10)}
```

```
d = dmatrix("scale(x1)", data)
d
```

```
## DesignMatrix with shape (10, 2)
##   Intercept  scale(x1)
##           1   -1.66439
##           1    1.31641
##           1    1.51247
##           1   -0.85749
##           1    0.40329
##           1   -1.27297
##           1   -0.50679
##           1    0.37778
##           1    0.18438
##           1    0.50732
##   Terms:
##     'Intercept' (column 0)
##     'scale(x1)' (column 1)
```

```
np.mean(d, axis=0)
```

```
pred = dmatrix(d.design_info, new_data)
pred
```

```
## DesignMatrix with shape (10, 2)
##   Intercept  scale(x1)
##           1   -0.01379
##           1    0.64039
##           1    1.73684
##           1   -0.76110
##           1   -0.39034
##           1    0.90875
##           1    0.64408
##           1   -0.06190
##           1    0.08917
##           1   -0.06990
##   Terms:
##     'Intercept' (column 0)
##     'scale(x1)' (column 1)
```

```
np.mean(pred, axis=0)
```

scikit-lego PatsyTransformer

If you would like to use a Patsy formula in a scikitlearn pipeline, it is possible via the `PatsyTransformer` from the `scikit-lego` library.

```
from sklego.preprocessing import PatsyTransformer

df = pd.DataFrame({
  "y": [2, 2, 4, 4, 6],
  "x": [1, 2, 3, 4, 5],
  "a": ["yes", "yes", "no", "no", "yes"]
})

X, y = df[["x", "a"]], df[["y"]].values
```

```
pt = PatsyTransformer("x*a + np.log(x)")
pt.fit_transform(X)
```

```
## DesignMatrix with shape (5, 5)
##   Intercept  a[T.yes]  x  x:a[T.yes]  np.log(x)
##           1         1  1           1    0.0000
##           1         1  2           2    0.69315
##           1         0  3           0    1.09861
##           1         0  4           0    1.38629
```

```
make_pipeline(
  PatsyTransformer("x*a + np.log(x)"),
  StandardScaler()
).fit_transform(X)
```

```
## array([[ 0.      ,  0.8165 , -1.41421, -0.3235 ,
##         [ 0.      ,  0.8165 , -0.70711,  0.21567,
##         [ 0.      , -1.22474,  0.     , -0.86266,
##         [ 0.      , -1.22474,  0.70711, -0.86266,
```

Exercise 1

Using patsy fit a linear regression model to the `books` data that includes an interaction term between `cover` and `volume`.

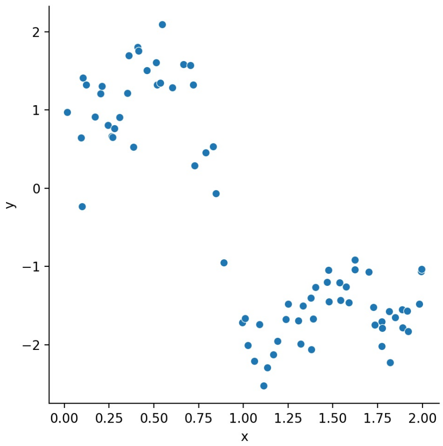
```
books = pd.read_csv("https://sta663-sp22.github.io/slides/data/daag_books.csv")
```

B-splines

Patsy also has support for B-splines and other related models.

```
d = pd.read_csv("data/d1.csv")

sns.relplot(x="x", y="y", data=d)
```




```
y, X = dmatrices("y ~ bs(x, df=6)", data=d)
X
```

```
## DesignMatrix with shape (75, 7)
##   Columns:
##     ['Intercept',
##      'bs(x, df=6)[0]',
##      'bs(x, df=6)[1]',
##      'bs(x, df=6)[2]',
##      'bs(x, df=6)[3]',
##      'bs(x, df=6)[4]',
##      'bs(x, df=6)[5]']
##   Terms:
##     'Intercept' (column 0), 'bs(x, df=6)' (colu
##   (to view full data, use np.asarray(this_obj))
```

What is `bs(x)[i]`?

```
bs_df = (
  dmatrix("bs(x, df=6)", data=d, return_type="dataframe")
  .drop(["Intercept"], axis = 1)
  .assign(x = d["x"])
  .melt(id_vars="x")
)

sns.relplot(x="x", y="value", hue="variable", kind="line", data = bs_df, aspect=1.5)
```

Fitting a model

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression(fit_intercept=False).fit(X,y)
lm.coef_
```

```
## array([[ 1.28955, -1.69132,  3.17914, -5.3865 , -1.18284, -3.8488 , -2.42867]])
```

```
plt.figure(layout="constrained")
sns.lineplot(x=d["x"], y=lm.predict(X).ravel(), color="k")
sns.scatterplot(x="x", y="y", data=d)
plt.show()
```

sklearn SplineTransformer

```
from sklearn.preprocessing import SplineTransform

p = make_pipeline(
  SplineTransformer(
    n_knots=6,
    degree=3,
    include_bias=True
  ),
  LinearRegression(fit_intercept=False)
).fit(
  d[["x"]], d["y"]
)
```

```
plt.figure()
sns.lineplot(x=d["x"], y=p.predict(d[["x"]]).rave
sns.scatterplot(x="x", y="y", data=d)
plt.show()
```

Why different?

For patsy the number of splines is determined by `df` while for sklearn this is determined by `n_knots + degree - 1`.

```
p = p.set_params(splinetransformer__n_knots = 5).fit(d[["x"]], d["y"])

plt.figure(layout="constrained")
sns.lineplot(x=d["x"], y=p.predict(d[["x"]]).ravel(), color="k")
sns.scatterplot(x="x", y="y", data=d)
plt.show()
```

but that is not the whole story, if we examine the bases we also see they differ slightly between implementations:

```
bs_df = pd.DataFrame(
  SplineTransformer(n_knots=6, degree=3, include_bias=True).fit_transform(d[["x"]]),
  columns = ["bs["+ str(i) +"]" for i in range(8)]
).assign(
  x = d.x
).melt(
  id_vars = "x"
)

sns.relplot(x="x", y="value", hue="variable", kind="line", data = bs_df, aspect=1.5)
```

statsmodels

statsmodels

> 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. An extensive list of result statistics are available for each estimator. The results are tested against existing statistical packages to ensure that they are correct.

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

`statsmodels` uses slightly different terminology for refering to `y` / dependent / response and `x` / independent / explanatory variables. Specificially it uses `endog` to refer to the `y` and `exog` to refer to the `x` variable(s).

This is particularly important when using the main API, less so when using the formula API.

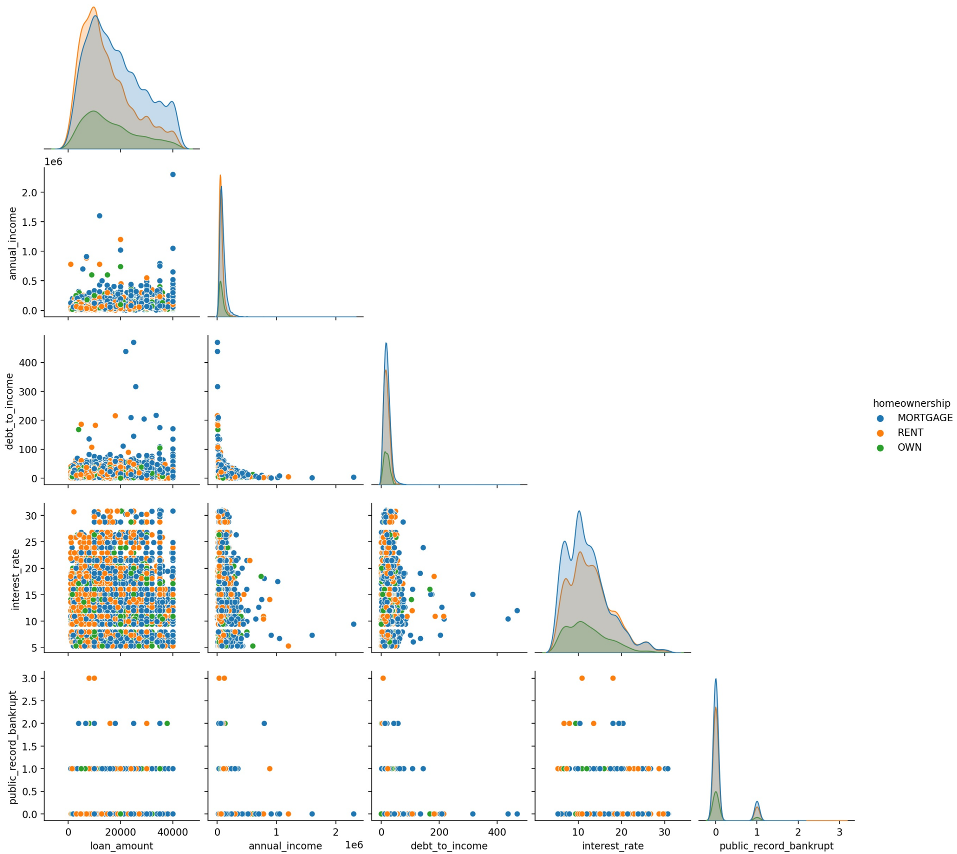
OpenIntro Loans data

> This data set represents thousands of loans made through the Lending Club platform, which is a platform that allows individuals to lend to other individuals. Of course, not all loans are created equal. Someone who is a essentially a sure bet to pay back a loan will have an easier time getting a loan with a low interest rate than someone who appears to be riskier. And for people who are very risky? They may not even get a loan offer, or they may not have accepted the loan offer due to a high interest rate. It is important to keep that last part in mind, since this data set only represents loans actually made, i.e. do not mistake this data for loan applications!

For the full data dictionary see here. We have removed some of the columns to make the data set more reasonably sized and also droped any rows with missing values.

```
loans = pd.read_csv("data/openintro_loans.csv")
loans
```

```
##      state  emp_length  term homeownership  annual_income  ... loan_amount  grade  interest_rate  public_record_bankrupt  loan_status
## 0       NJ           3    60      MORTGAGE        90000.0  ...       28000      C          14.07                       0      Current
## 1       HI          10    36          RENT        40000.0  ...        5000      C          12.61                       1      Current
## 2       WI           3    36          RENT        40000.0  ...        2000      D          17.09                       0      Current
## 3       PA           1    36          RENT        30000.0  ...       21600      A           6.72                       0      Current
## 4       CA          10    36          RENT        35000.0  ...       23000      C          14.07                       0      Current
## ...    ...         ...   ...           ...            ...  ...         ...    ...            ...                     ...          ...
## 9177    TX          10    36          RENT       108000.0  ...       24000      A           7.35                       1      Current
## 9178    PA           8    36      MORTGAGE       121000.0  ...       10000      D          19.03                       0      Current
## 9179    CT          10    36      MORTGAGE        67000.0  ...       30000      E          23.88                       0      Current
## 9180    WI           1    36      MORTGAGE        80000.0  ...       24000      A           5.32                       0      Current
## 9181    CT           3    36          RENT        66000.0  ...       12800      B          10.91                       0      Current
## 
## [9182 rows x 16 columns]
```



homeownership
MORTGAGE
RENT
OWN
loan_amount
annual_income
debt_to_income
interest_rate
public_record_bankrupt
1e6

OLS

```
y = loans["loan_amount"]
X = loans[["homeownership", "annual_income", "debt_to_income", "interest_rate", "public_record_bankrupt"]

model = sm.OLS(endog=y, exog=X)
```

```
## ValueError: Pandas data cast to numpy dtype of object. Check input data with np.asarray(data).
```

What do you think the issue is here?

The error occurs because `X` contains mixed types - specifically we have categorical data columns which cannot be directly converted to a numeric dtype so we need to take care of the dummy coding for statsmodels (with this interface).

```
X_dc = pd.get_dummies(X)
model = sm.OLS(endog=y, exog=X_dc)
```

Fitting and summary

```
res = model.fit()
print(res.summary())
```

```
##                             OLS Regression Results
## ==============================================================================
## Dep. Variable:            loan_amount   R-squared:                       0.135
## Model:                            OLS   Adj. R-squared:                  0.135
## Method:                 Least Squares   F-statistic:                     239.5
## Date:                Fri, 18 Mar 2022   Prob (F-statistic):         2.33e-285
## Time:                        09:18:54   Log-Likelihood:                -97245.
## No. Observations:                9182   AIC:                         1.945e+05
## Df Residuals:                    9175   BIC:                         1.946e+05
## Df Model:                           6
## Covariance Type:            nonrobust
## ============================================================================================
##                              coef    std err          t      P>|t|      [0.025      0.975]
## --------------------------------------------------------------------------------------------
## annual_income              0.0505      0.002     31.952      0.000       0.047       0.054
## debt_to_income            65.6641      7.310      8.982      0.000      51.334      79.994
## interest_rate            204.2480     20.448      9.989      0.000     164.166     244.330
## public_record_bankrupt -1362.3253    306.019     -4.452      0.000   -1962.191    -762.460
## homeownership_MORTGAGE  1.002e+04    357.245     28.048      0.000    9319.724    1.07e+04
## homeownership_OWN       8880.4144    422.296     21.029      0.000    8052.620    9708.209
## homeownership_RENT      7446.5385    351.641     21.177      0.000    6757.243    8135.834
## ==============================================================================
## Omnibus:                      481.833   Durbin-Watson:                   2.002
## Prob(Omnibus):                  0.000   Jarque-Bera (JB):              916.542
## Skew:                           0.391   Prob(JB):                    9.45e-200
## Kurtosis:                       4.336   Cond. No.                     6.17e+05
## ==============================================================================
```

Formula interface

Most of the modeling interfaces are also provided by `smf` (`statsmodels.formula.api`) in which case patsy is used to construct the model matrices.

```
model = smf.ols(
  "loan_amount ~ homeownership + annual_income + debt_to_income + interest_rate + public_record_bankrupt",
  data = loans
)
res = model.fit()
print(res.summary())
```

```
##                             OLS Regression Results
## ==============================================================================
## Dep. Variable:            loan_amount   R-squared:                       0.135
## Model:                            OLS   Adj. R-squared:                  0.135
## Method:                 Least Squares   F-statistic:                     239.5
## Date:                Fri, 18 Mar 2022   Prob (F-statistic):          2.33e-285
## Time:                        09:18:55   Log-Likelihood:                -97245.
## No. Observations:                9182   AIC:                         1.945e+05
## Df Residuals:                    9175   BIC:                         1.946e+05
## Df Model:                           6
## Covariance Type:            nonrobust
## ==========================================================================================
##                              coef    std err          t      P>|t|      [0.025      0.975]
## ------------------------------------------------------------------------------------------
## Intercept               1.002e+04    357.245     28.048      0.000    9319.724    1.07e+04
## homeownership[T.OWN]   -1139.5893    322.361     -3.535      0.000   -1771.489    -507.690
## homeownership[T.RENT]  -2573.4652    221.101    -11.639      0.000   -3006.873   -2140.057
## annual_income              0.0505      0.002     31.952      0.000       0.047       0.054
```

Result values and model parameters

```
res.params
```

```
## Intercept                      10020.003630
## homeownership[T.OWN]           -1139.589268
## homeownership[T.RENT]          -2573.465175
## annual_income                      0.050505
## debt_to_income                    65.664103
## interest_rate                    204.247993
## public_record_bankrupt         -1362.325291
## dtype: float64
```

```
res.bse
```

```
## Intercept                      357.244896
## homeownership[T.OWN]           322.361151
## homeownership[T.RENT]          221.101300
## annual_income                    0.001581
## debt_to_income                   7.310428
## interest_rate                   20.447644
## public_record_bankrupt         306.019080
## dtype: float64
```

```
res.rsquared
```

```
## 0.13542611095847512
```

```
res.aic
```

```
## 194503.99751598848
```

```
res.bic
```

```
## 194553.87251826216
```

```
res.predict()
```

```
## array([18621.86199, 11010.94015, 14346.14516, 1
##        18283.87492, 26719.61804, 18496.72337, 1
##        18845.26463, 20083.33976, ..., 19576.169
##        17161.96037, 18764.48833, 19252.9242 , 1
##        22144.19006, 21253.25932, 15934.34097, 1
```

Diagnostic plots

QQ Plot

```
plt.figure()
sm.graphics.qqplot(res.resid, line="s")
plt.show()
```

Leverage plot

```
plt.figure()
sm.graphics.plot_leverage_resid2(res)
plt.show()
```

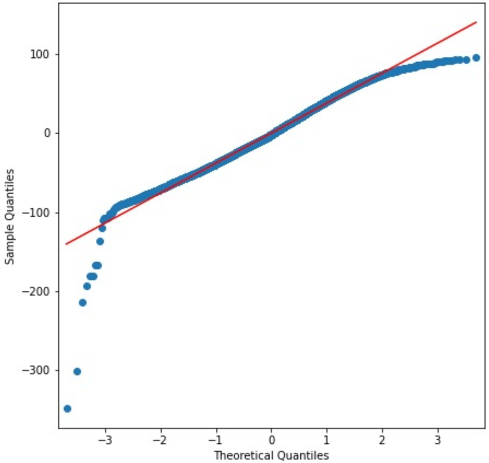
Alternative model

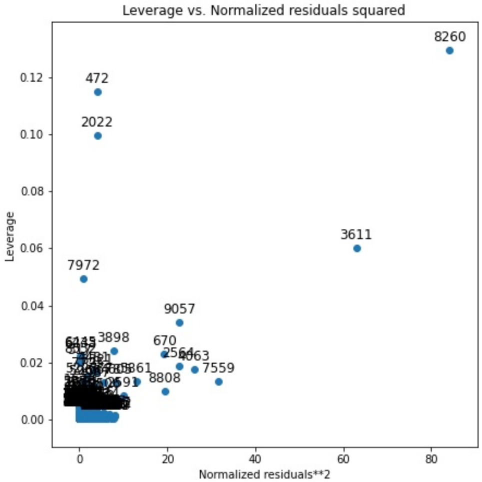
```
res = smf.ols(
  "np.sqrt(loan_amount) ~ homeownership + annual_income + debt_to_income + interest_rate + public_record_bankrupt",
  data = loans
).fit()
print(res.summary())
```

```
##                              OLS Regression Results
## ================================================================================
## Dep. Variable:     np.sqrt(loan_amount)   R-squared:                       0.132
## Model:                              OLS   Adj. R-squared:                  0.132
## Method:                   Least Squares   F-statistic:                     232.7
## Date:                  Fri, 18 Mar 2022   Prob (F-statistic):          1.16e-277
## Time:                          09:18:58   Log-Likelihood:                -46429.
## No. Observations:                  9182   AIC:                         9.287e+04
## Df Residuals:                      9175   BIC:                         9.292e+04
## Df Model:                             6
## Covariance Type:              nonrobust
## ==========================================================================================
##                              coef    std err          t      P>|t|      [0.025      0.975]
## ------------------------------------------------------------------------------------------
## Intercept                 95.4915      1.411     67.687      0.000      92.726      98.257
## homeownership[T.OWN]      -4.4495      1.273     -3.495      0.000      -6.945      -1.954
## homeownership[T.RENT]    -10.4225      0.873    -11.937      0.000     -12.134      -8.711
## annual_income              0.0002   6.24e-06     30.916      0.000       0.000       0.000
## debt_to_income             0.2720      0.029      9.421      0.000       0.215       0.329
## interest_rate              0.8911      0.081     11.035      0.000       0.733       1.049
## public_record_bankrupt    -4.6899      1.208     -3.881      0.000      -7.059      -2.321
## ==============================================================================
## Omnibus:                      178.498   Durbin-Watson:                   2.011
## Prob(Omnibus):                  0.000   Jarque-Bera (JB):              333.598
```

```
plt.figure()
sm.graphics.qqplot(res.resid, line="s")
plt.show()
```

```
plt.figure()
sm.graphics.plot_leverage_resid2(res)
plt.show()
```





Bushtail Possums

> Data representing possums in Australia and New Guinea. This is a copy of the data set by the same name in the DAAG package, however, the data set included here includes fewer variables.
>
> `pop` - Population, either `Vic` (Victoria) or `other` (New South Wales or Queensland).

```
possum = pd.read_csv("data/possum.csv")
possum
```

```
##      site    pop sex  age  head_l  skull_w  total_l  tail_l
## 0       1    Vic   m  8.0    94.1     60.4     89.0    36.0
## 1       1    Vic   f  6.0    92.5     57.6     91.5    36.5
## 2       1    Vic   f  6.0    94.0     60.0     95.5    39.0
## 3       1    Vic   f  6.0    93.2     57.1     92.0    38.0
## 4       1    Vic   f  2.0    91.5     56.3     85.5    36.0
## ..    ...    ...  ..  ...     ...      ...      ...     ...
## 99      7  other   m  1.0    89.5     56.0     81.5    36.5
## 100     7  other   m  1.0    88.6     54.7     82.5    39.0
## 101     7  other   f  6.0    92.4     55.0     89.0    38.0
## 102     7  other   m  4.0    91.5     55.2     82.5    36.5
## 103     7  other   f  3.0    93.6     59.9     89.0    40.0
```

Logistic regression models (GLM)

```
y = pd.get_dummies( possum["pop"] )
X = pd.get_dummies( possum.drop(["site","pop"], axis=1) )

model = sm.GLM(y, X, family = sm.families.Binomial())
```

```
## MissingDataError: exog contains inf or nans
```

Behavior for dealing with missing data can be handled via `missing`, possible values are `"none"`, `"drop"`, and `"raise"`.

```
model = sm.GLM(y, X, family = sm.families.Binomial(), missing="drop")
```

Fit and summary

```
res = model.fit()
print(res.summary())
```

```
##                  Generalized Linear Model Regression Results
## ==============================================================================
## Dep. Variable:       ['Vic', 'other']   No. Observations:                  102
## Model:                            GLM   Df Residuals:                       95
## Model Family:                Binomial   Df Model:                            6
## Link Function:                  Logit   Scale:                          1.0000
## Method:                          IRLS   Log-Likelihood:                -31.942
## Date:                Fri, 18 Mar 2022   Deviance:                       63.885
## Time:                        09:19:01   Pearson chi2:                     154.
## No. Iterations:                     7   Pseudo R-squ. (CS):             0.5234
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## age            0.1373      0.183      0.751      0.453      -0.221       0.495
## head_l        -0.1972      0.158     -1.247      0.212      -0.507       0.113
## skull_w       -0.2001      0.139     -1.443      0.149      -0.472       0.072
## total_l        0.7569      0.176      4.290      0.000       0.411       1.103
## tail_l        -2.0698      0.429     -4.820      0.000      -2.912      -1.228
## sex_f         40.0148     13.077      3.060      0.002      14.385      65.645
## sex_m         38.5395     12.941      2.978      0.003      13.175      63.904
## ==============================================================================
```

Success vs failure

Note endog can be 1d or 2d for binomial models - in the case of the latter each row is interpreted as [success, failure]. s

```
y = pd.get_dummies( possum["pop"], drop_first = True)
X = pd.get_dummies( possum.drop(["site","pop"], axis=1) )

res = sm.GLM(y, X, family = sm.families.Binomial(), missing="drop").fit()
print(res.summary())
```

```
##                  Generalized Linear Model Regression Results
## ==============================================================================
## Dep. Variable:                  other   No. Observations:                  102
## Model:                            GLM   Df Residuals:                       95
## Model Family:                Binomial   Df Model:                            6
## Link Function:                  Logit   Scale:                          1.0000
## Method:                          IRLS   Log-Likelihood:                -31.942
## Date:                Fri, 18 Mar 2022   Deviance:                       63.885
## Time:                        09:19:02   Pearson chi2:                     154.
## No. Iterations:                     7   Pseudo R-squ. (CS):             0.5234
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## age           -0.1373      0.183     -0.751      0.453      -0.495       0.221
## head_l         0.1972      0.158      1.247      0.212      -0.113       0.507
## skull_w        0.2001      0.139      1.443      0.149      -0.072       0.472
## total_l       -0.7569      0.176     -4.290      0.000      -1.103      -0.411
## tail_l         2.0698      0.429      4.820      0.000       1.228       2.912
## sex_f        -40.0148     13.077     -3.060      0.002     -65.645     -14.385
```

Fit and summary

```
res = model.fit()
print(res.summary())
```

```
##                  Generalized Linear Model Regression Results
## ==============================================================================
## Dep. Variable:       ['Vic', 'other']   No. Observations:                  102
## Model:                            GLM   Df Residuals:                       95
## Model Family:                Binomial   Df Model:                            6
## Link Function:                  Logit   Scale:                          1.0000
## Method:                          IRLS   Log-Likelihood:                -31.942
## Date:                Fri, 18 Mar 2022   Deviance:                       63.885
## Time:                        09:19:03   Pearson chi2:                     154.
## No. Iterations:                     7   Pseudo R-squ. (CS):             0.5234
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## age            0.1373      0.183      0.751      0.453      -0.221       0.495
## head_l        -0.1972      0.158     -1.247      0.212      -0.507       0.113
## skull_w       -0.2001      0.139     -1.443      0.149      -0.472       0.072
## total_l        0.7569      0.176      4.290      0.000       0.411       1.103
## tail_l        -2.0698      0.429     -4.820      0.000      -2.912      -1.228
## sex_f         40.0148     13.077      3.060      0.002      14.385      65.645
## sex_m         38.5395     12.941      2.978      0.003      13.175      63.904
```

Formula interface

```
res = smf.glm(
  "pop ~ sex + age + head_l + skull_w + total_l + tail_l-1",
  data = possum,
  family = sm.families.Binomial(),
  missing="drop"
).fit()
print(res.summary())
```

```
##                  Generalized Linear Model Regression Results
## ==============================================================================
## Dep. Variable:     ['pop[Vic]', 'pop[other]']   No. Observations:          102
## Model:                         GLM   Df Residuals:                   95
## Model Family:             Binomial   Df Model:                        6
## Link Function:               Logit   Scale:                      1.0000
## Method:                       IRLS   Log-Likelihood:            -31.942
## Date:             Fri, 18 Mar 2022   Deviance:                   63.885
## Time:                     09:19:03   Pearson chi2:                 154.
## No. Iterations:                  7   Pseudo R-squ. (CS):         0.5234
## Covariance Type:         nonrobust
## ======================================================================
##               coef    std err          z      P>|z|      [0.025      0.975]
## ----------------------------------------------------------------------
## sex[f]      40.0148     13.077      3.060      0.002      14.385      65.645
## sex[m]      38.5395     12.941      2.978      0.003      13.175      63.904
```

sleepstudy data

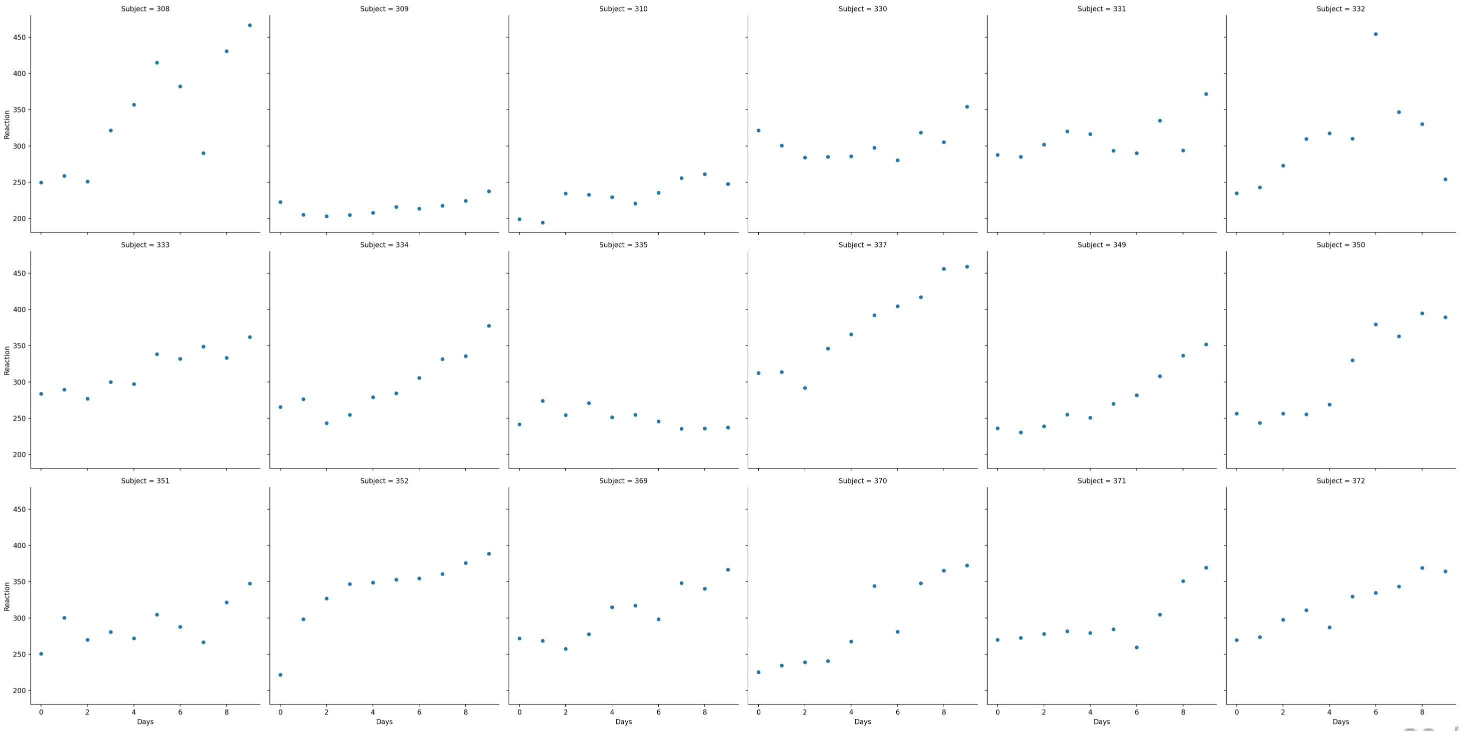
> These data are from the study described in Belenky et al. (2003), for the most sleep-deprived group (3 hours time-in-bed) and for the first 10 days of the study, up to the recovery period. The original study analyzed speed (1/(reaction time)) and treated day as a categorical rather than a continuous predictor.
>
> The average reaction time per day (in milliseconds) for subjects in a sleep deprivation study. Days 0-1 were adaptation and training (T1/T2), day 2 was baseline (B); sleep deprivation started after day 2.

```
sleep = pd.read_csv("data/sleepstudy.csv")
sleep
```

```
##      Reaction  Days  Subject
## 0    249.5600     0      308
## 1    258.7047     1      308
## 2    250.8006     2      308
## 3    321.4398     3      308
## 4    356.8519     4      308
## ..        ...   ...      ...
## 175  329.6076     5      372
```

These data come from the `sleepstudy` dataset in the `lme4` R package

```python
sns.relplot(x="Days", y="Reaction", col="Subject", col_wrap=6, data=sleep)
```



Random intercept model

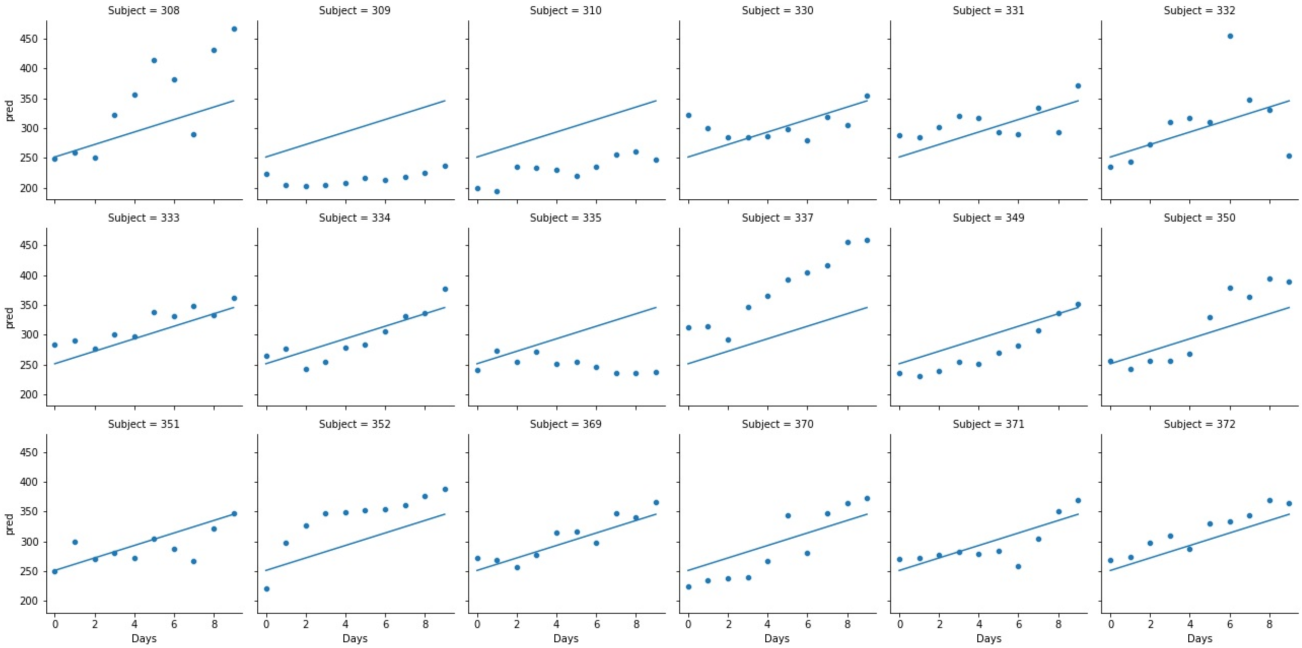
```python
me_rand_int = smf.mixedlm(
  "Reaction ~ Days", data=sleep, groups=sleep["Subject"],
  subset=sleep.Days >= 2
)
res_rand_int = me_rand_int.fit(method=["lbfgs"])
print(res_rand_int.summary())
```

```
##          Mixed Linear Model Regression Results
## ========================================================
## Model:             MixedLM Dependent Variable: Reaction
## No. Observations:  180     Method:             REML
## No. Groups:        18      Scale:              960.4529
## Min. group size:   10      Log-Likelihood:     -893.2325
## Max. group size:   10      Converged:          Yes
## Mean group size:   10.0
## --------------------------------------------------------
##            Coef.   Std.Err.   z    P>|z|  [0.025  0.975]
## --------------------------------------------------------
## Intercept  251.405    9.747 25.793 0.000 232.302 270.509
## Days        10.467    0.804 13.015 0.000   8.891  12.044
## Group Var 1378.232   17.157
## ========================================================
```

```r
summary(
  lmer(Reaction ~ Days + (1|Subject), data=sleepstudy)
)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: Reaction ~ Days + (1 | Subject)
##    Data: sleepstudy
##
## REML criterion at convergence: 1786.5
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.2257 -0.5529  0.0109  0.5188  4.2506
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  Subject  (Intercept) 1378.2   37.12
##  Residual              960.5   30.99
## Number of obs: 180, groups:  Subject, 18
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept) 251.4051     9.7467   25.79
## Days         10.4673     0.8042   13.02
##
## Correlation of Fixed Effects:
##      (Intr)
## Days -0.371
```

Predictions



The prediction is only taking into account the fixed effects here, not the group random effects.

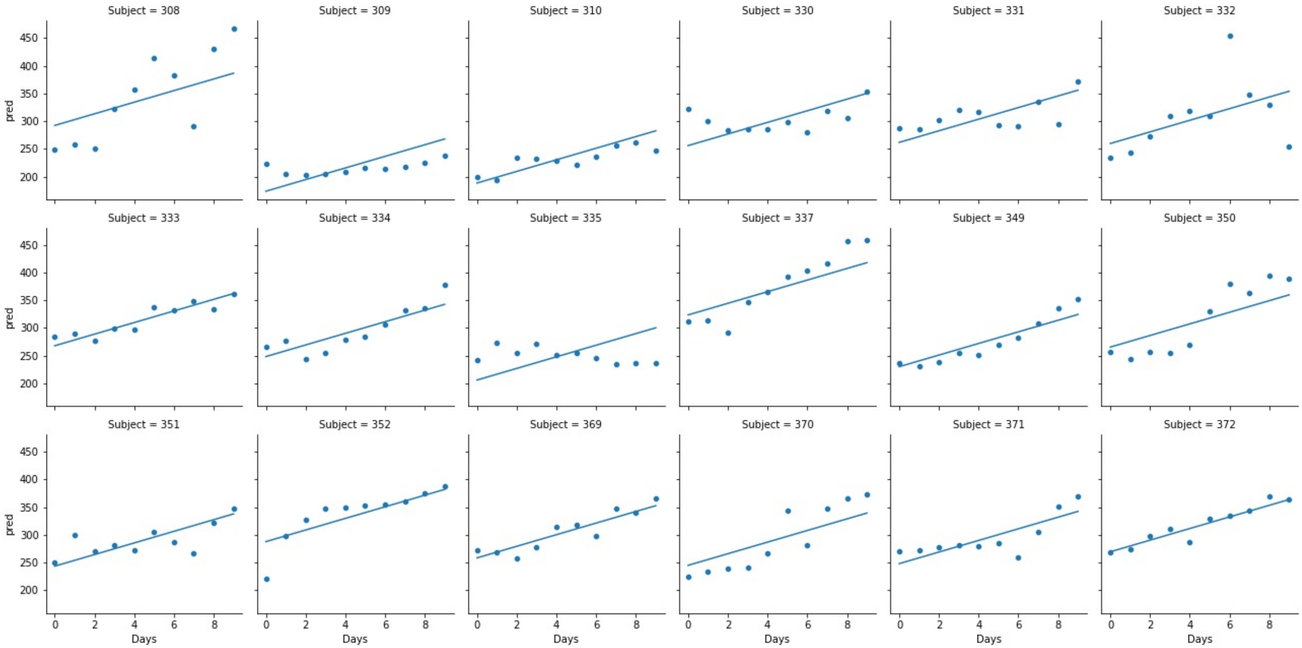
Recovering random effects for prediction

```
# Dictionary of random effects estimates
re = res_rand_int.random_effects

# Multiply each RE by the random effects design matrix for each group
rex = [np.dot(me_rand_int.exog_re_li[j], re[k]) for (j, k) in enumerate(me_rand_int.group_labels)]

# Add the fixed and random terms to get the overall prediction
rex = np.concatenate(rex)
y_hat = res_rand_int.predict() + rex
```

Based on code provide on stack overflow.


Subject = 308
Subject = 309
Subject = 310
Subject = 330
Subject = 331
Subject = 332
Subject = 333
Subject = 334
Subject = 335
Subject = 337
Subject = 349
Subject = 350
Subject = 351
Subject = 352
Subject = 369
Subject = 370
Subject = 371
Subject = 372
pred
Days
200
250
300
350
400
450
0
2
4
6
8

Random intercept and slope model

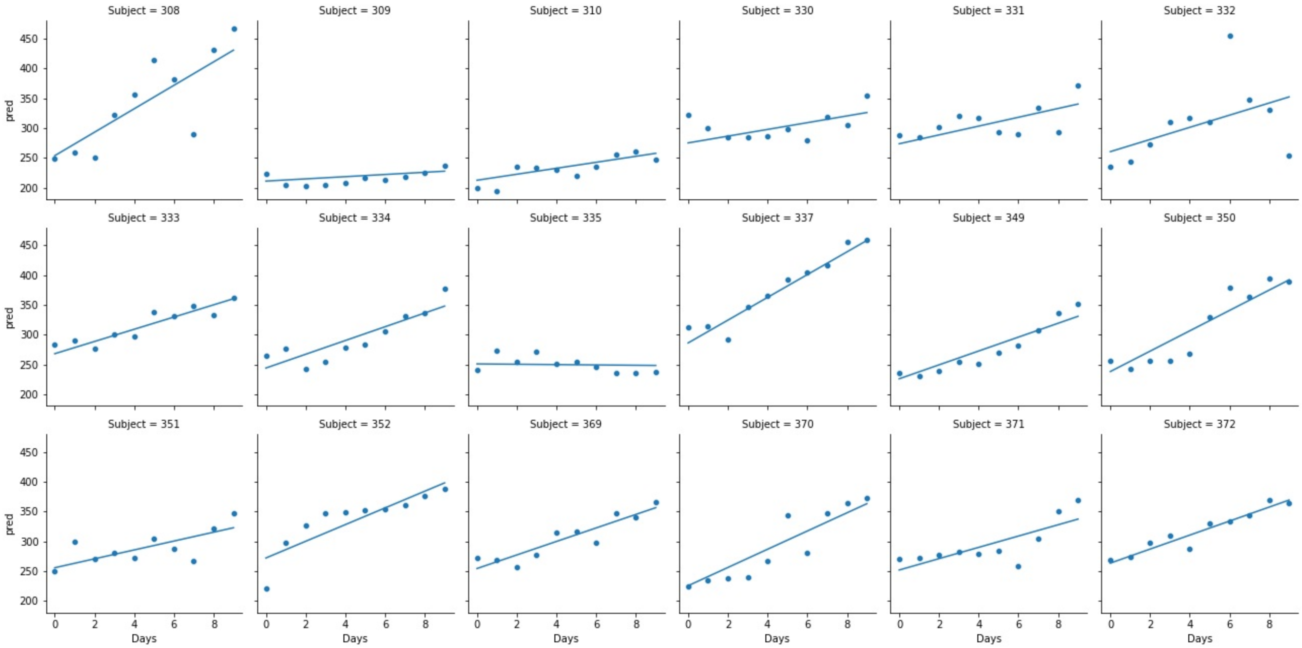
```
me_rand_sl= smf.mixedlm(
  "Reaction ~ Days", data=sleep, groups=sleep["Subject"],
  subset=sleep.Days >= 2,
  re_formula="~Days"
)
res_rand_sl = me_rand_sl.fit(method=["lbfgs"])
print(res_rand_sl.summary())
```

```
##              Mixed Linear Model Regression Results
## ================================================================
## Model:                MixedLM   Dependent Variable:   Reaction
## No. Observations:     180       Method:               REML
## No. Groups:           18        Scale:                654.9412
## Min. group size:      10        Log-Likelihood:       -871.814
## Max. group size:      10        Converged:            Yes
## Mean group size:      10.0
## ----------------------------------------------------------------
##                      Coef.  Std.Err.   z    P>|z|  [0.025  0.975
## ----------------------------------------------------------------
## Intercept           251.405    6.825 36.838 0.000 238.029 264.78
## Days                 10.467    1.546  6.771 0.000   7.438  13.49
## Group Var           612.089   11.881
## Group x Days Cov      9.605    1.820
## Days Var             35.072    0.610
## ================================================================
```

```
summary(
  lmer(Reaction ~ Days + (Days|Subject), data=sleepstudy)
)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: Reaction ~ Days + (Days | Subject)
##    Data: sleepstudy
##
## REML criterion at convergence: 1743.6
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.9536 -0.4634  0.0231  0.4634  5.1793
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  Subject  (Intercept) 612.10   24.741
##           Days         35.07    5.922   0.07
##  Residual             654.94   25.592
## Number of obs: 180, groups:  Subject, 18
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  251.405      6.825  36.838
## Days          10.467      1.546   6.771
##
## Correlation of Fixed Effects:
##      (Intr)
## Days -0.138
```

Prediction



We are using the same approach described previously to obtain the RE estimates and use them in the predictions.

t-test and z-test for equality of means

```
cm = sm.stats.CompareMeans(
  sm.stats.DescrStatsW( books.weight[books.cover == "hb"] ),
  sm.stats.DescrStatsW( books.weight[books.cover == "pb"] )
)

print(cm.summary())
```

```
##                            Test for equality of means
## ==============================================================================
##                  coef    std err          t      P>|t|      [0.025      0.975]
## ------------------------------------------------------------------------------
## subset #1    168.3036    136.636      1.232      0.240    -126.880     463.487
## ==============================================================================
```

```
print(cm.summary(use_t=False))
```

```
##                            Test for equality of means
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## subset #1    168.3036    136.636      1.232      0.218     -99.497     436.104
## ==============================================================================
```

```
print(cm.summary(usevar="unequal"))
```

```
##                            Test for equality of means
## ==============================================================================
##                  coef    std err          t      P>|t|      [0.025      0.975]
## ------------------------------------------------------------------------------
## subset #1    168.3036    136.360      1.234      0.239    -126.686     463.293
```

Contigency tables

Below are data from the GSS and a survery of Duke students in a intro stats class - the question asked about how concerned the respondent was about the effect of global warming on polar ice cap melt.

```
gss = pd.DataFrame({"US": [454, 226], "Duke": [56,32]}, index=["A great deal", "Not a great deal"])
gss
```

```
##                    US  Duke
## A great deal      454    56
## Not a great deal  226    32
```

```
tbl = sm.stats.Table2x2(gss.to_numpy())
print(tbl.summary())
```

```
##                 Estimate   SE   LCB    UCB  p-value
## --------------------------------------------------
## Odds ratio        1.148        0.723 1.823   0.559
## Log odds ratio    0.138 0.236 -0.325 0.601   0.559
## Risk ratio        1.016        0.962 1.074   0.567
## Log risk ratio    0.016 0.028 -0.039 0.071   0.567
## --------------------------------------------------
```