custom transformers + patsy & statsmodels

Lecture 17

Dr. Colin Rundel

Custom sklearn transformers

FunctionTransformer

The simplest way to create a new transformer is to use `FunctionTransformer()` from the preprocessing submodule which allows for converting a Python function into a transformer.

```python
from sklearn.preprocessing import FunctionTransformer
X = pd.DataFrame({"x1": range(1,6), "x2": range(5, 0, -1)})
```

```python
log_transform = FunctionTransformer(np.log)
lt = log_transform.fit(X)
lt
```

FunctionTransformer(func=<ufunc 'log'>)

```python
lt.transform(X)
```

```
       x1      x2
0  0.0000  1.6094
1  0.6931  1.3863
2  1.0986  1.0986
3  1.3863  0.6931
4  1.6094  0.0000
```

```python
lt.get_params()
```

```
{'accept_sparse': False, 'check_inverse': True,
'feature_names_out': None, 'func': <ufunc
'log'>, 'inv_kw_args': None, 'inverse_func':
None, 'kw_args': None, 'validate': False}
```

```python
dir(lt)
```

```
['__annotations__', '__class__', '__delattr__',
'__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__',
'__getstate__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__le__',
'__lt__', '__module__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__setstate__', '__sizeof__',
'__sklearn_clone__', '__sklearn_is_fitted__',
'__sklearn_tags__', '__str__',
'__subclasshook__', '__weakref__',
'_build_request_for_signature',
'_check_feature_names', '_check_input',
'_check_inverse_transform',
'_check_n_features', '_doc_link_module',
'_doc_link_template',
'_doc_link_url_param_generator',
'_get_default_requests', '_get_doc_link'
```

Input types

```
def interact(X, y = None):
  return np.c_[X, X[:,0] * X[:,1]]
X = pd.DataFrame({"x1": range(1,6), "x2": range(5, 0, -1)})
Z = np.array(X)
```

```
FunctionTransformer(
  interact
).fit_transform(X)
```

```
pandas.errors.InvalidIndexError: (slice(None,
None, None), 0)
```

```
FunctionTransformer(
  interact
).fit_transform(Z)
```

```
array([[1, 5, 5],
       [2, 4, 8],
       [3, 3, 9],
       [4, 2, 8],
       [5, 1, 5]])
```

```
FunctionTransformer(
  interact, validate=True
).fit_transform(X)
```

```
array([[1, 5, 5],
       [2, 4, 8],
       [3, 3, 9],
       [4, 2, 8],
       [5, 1, 5]])
```

```
FunctionTransformer(
  interact, validate=True
).fit_transform(Z)
```

```
array([[1, 5, 5],
       [2, 4, 8],
       [3, 3, 9],
       [4, 2, 8],
       [5, 1, 5]])
```

Build your own transformer

For a more full featured transformer, it is possible to construct it as a class that inherits from `BaseEstimator` and `TransformerMixin` classes from the `base` submodule.

```python
from sklearn.base import BaseEstimator, TransformerMixin

class scaler(BaseEstimator, TransformerMixin):
  def __init__(self, m = 1, b = 0):
    self.m = m
    self.b = b

  def fit(self, X, y=None):
    return self

  def transform(self, X, y=None):
    return X*self.m + self.b
```

```python
X = pd.DataFrame({
  "x1": range(1,6),
  "x2": range(5, 0, -1)}
); X
```

```
   x1  x2
0   1   5
1   2   4
2   3   3
3   4   2
4   5   1
```

```python
double = scaler(2)
double.get_params()
```

```
{'b': 0, 'm': 2}
```

```python
double.fit_transform(X)
```

```
   x1  x2
0   2  10
1   4   8
2   6   6
3   8   4
4  10   2
```

```python
double.set_params(b=-3).fit_transform(X)
```

```
   x1  x2
0  -1   7
1   1   5
2   3   3
3   5   1
4   7  -1
```

What else do we get?

```
print(
  np.array(dir(double))
)
```

```
['__class__' '__delattr__' '__dict__' '__dir__' '__doc__' '__eq__' '__format__'
 '__ge__' '__getattribute__' '__getstate__' '__gt__' '__hash__' '__init__'
 '__init_subclass__' '__le__' '__lt__' '__module__' '__ne__' '__new__'
 '__reduce__' '__reduce_ex__' '__repr__' '__setattr__' '__setstate__'
 '__sizeof__' '__sklearn_clone__' '__sklearn_tags__' '__str__'
 '__subclasshook__' '__weakref__' '_build_request_for_signature'
 '_check_feature_names' '_check_n_features' '_doc_link_module'
 '_doc_link_template' '_doc_link_url_param_generator' '_get_default_requests'
 '_get_doc_link' '_get_metadata_request' '_get_param_names' '_get_tags'
 '_more_tags' '_repr_html_' '_repr_html_inner' '_repr_mimebundle_'
 '_sklearn_auto_wrap_output_keys' '_validate_data' '_validate_params' 'b' 'fit'
 'fit_transform' 'get_metadata_routing' 'get_params' 'm' 'set_output'
 'set_params' 'transform']
```

Demo - Interaction Transformer

Useful methods

We employed a couple of special methods that are worth mentioning in a little more detail.

- `validate_data()` & `_check_feature_names()` are functions from `sklearn.base.validate` they are responsible for setting and checking the `n_features_in_` and the `feature_names_in_` attributes respectively.
- In general one or both is run during `fit()` with `reset=True` in which case the respective attribute will be set.
- Later, in `tranform()` one or both will again be called with `reset=False` and the properties of `X` will be checked against the values in the attribute.
- These are worth using as they promote an interface consistent with sklearn and also provide convenient error checking with useful warning / error messages.

`check_is_fitted()`

This is another useful helper function from `sklearn.utils` - it is fairly simplistic in that it checks for the existence of a specified attribute. If no attribute is given then it checks for any attributes ending in `_` that do not begin with `__`.

Again this is useful for providing a consistent interface and useful error / warning messages.

See also the other `check*()` functions in `sklearn.utils`.

Other custom estimators

If you want to implement your own custom modeling function it is possible, there are different Mixin base classes in `sklearn.base` that provide the common core interface.

Class	Description
`base.BiclusterMixin`	Mixin class for all bicluster estimators
`base.ClassifierMixin`	Mixin class for all classifiers
`base.ClusterMixin`	Mixin class for all cluster estimators
`base.DensityMixin`	Mixin class for all density estimators
`base.RegressorMixin`	Mixin class for all regression estimators
`base.TransformerMixin`	Mixin class for all transformers
`base.OneToOneFeatureMixin`	Provides get_feature_names_out for simple transformers

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

```python
from patsy import ModelDesc
```

```python
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)')])])
```

```python
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)')])])
```

This general syntax is known as Wilkinson Notation and come from Wilkinson and Rogers. “Symbolic description of

Model matrix

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

```python
data = demo_data(
  "y", "a", "b", "x1", "x2"
)
pd.DataFrame(data)
```

```
    a   b      x1      x2       y
0  a1  b1  1.7641 -0.1032  1.4941
1  a1  b2  0.4002  0.4106 -0.2052
2  a2  b1  0.9787  0.1440  0.3131
3  a2  b2  2.2409  1.4543 -0.8541
4  a1  b1  1.8676  0.7610 -2.5530
5  a1  b2 -0.9773  0.1217  0.6536
6  a2  b1  0.9501  0.4439  0.8644
7  a2  b2 -0.1514  0.3337 -0.7422
```

```python
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.b2]  np.exp(x1)
          1        0              0              0     5.83604
          1        0              1              0     1.49206
          1        1              0              0     2.66110
          1        1              0              1     9.40173
          1        0              0              0     6.47247
          1        0              1              0     0.37633
          1        1              0              0     2.58594
          1        1              0              1     0.85954
  Terms:
    'Intercept' (column 0)
    'a' (column 1)
    'a:b' (columns 2:4)
    'np.exp(x1)' (column 4)
```

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.b2]  np.exp(x1)
          1        0              0              0     5.83604
          1        0              1              0     1.49206
          1        1              0              0     2.66110
          1        1              0              1     9.40173
          1        0              0              0     6.47247
          1        0              1              0     0.37633
          1        1              0              0     2.58594
          1        1              0              1     0.85954
  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      5.8360
1        1.0      0.0            1.0            0.0      1.4921
2        1.0      1.0            0.0            0.0      2.6611
3        1.0      1.0            0.0            1.0      9.4017
4        1.0      0.0            0.0            0.0      6.4725
5        1.0      0.0            1.0            0.0      0.3763
6        1.0      1.0            0.0            0.0      2.5859
7        1.0      1.0            0.0            1.0      0.8595
```

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 arithmetic calculations	`I(x1 + x2)`
`Q()`	used to quote column names, e.g. columns with spaces or symbols	`Q('bad name!')`
`C()`	used for categorical data coding	`C(a, Treatment('a2'))`

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 (-1)

```
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
  0.40016
  0.97874
  2.24089
  1.86756
  Terms:
    'x' (column 0)
```

Intercept Examples (0)

```
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
  0.40016
  0.97874
  2.24089
  1.86756
  Terms:
    'x' (column 0)
```

Design Info

One of the key 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>,
                                         num_columns=1)},
           term_codings=OrderedDict([(Term([]),
```

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    0.16735
          1    0.24109
          1   -1.16663
          1    0.73554
          1    1.04560
          1   -0.55675
          1   -0.55099
          1    0.38141
          1   -1.89695
          1    1.60033
  Terms:
    'Intercept' (column 0)
    'scale(x1)' (column 1)
```

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

```
array([1., 0.])
```

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

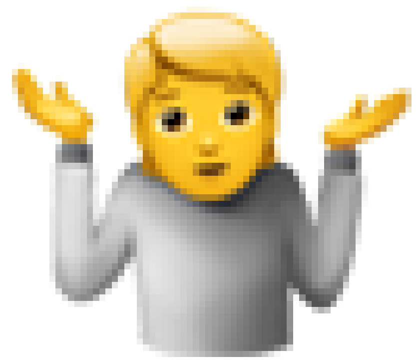
```
DesignMatrix with shape (10, 2)
  Intercept  scale(x1)
          1   -0.25337
          1   -0.15131
          1   -0.33295
          1   -0.47826
          1   -0.23705
          1    0.46020
          1    1.07162
          1   -0.55143
          1    1.51838
          1   -1.19011
  Terms:
    'Intercept' (column 0)
    'scale(x1)' (column 1)
```

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

```
array([ 1.    , -0.0144])
```

scikit-learn + Patsy

The state of affairs here is a bit of a mess at the moment - previously the `sklego` package implemented a `PatsyTransformer` class that has since been deprecated in favor of the `FormulaicTransformer` which uses the `formulaic` package for formula handling.



A PatsyTransformer

```python
from patsy import dmatrix, build_design_matrices
from sklearn.utils.validation import check_is_fitted
from sklearn.base import BaseEstimator, TransformerMixin

class PatsyTransformer(TransformerMixin, BaseEstimator):
    def __init__(self, formula):
        self.formula = formula

    def fit(self, X, y=None):
        m = dmatrix(self.formula, X)
        assert np.array(m).shape[0] == np.array(X).shape[0]
        self.design_info_ = m.design_info
        return self

    def transform(self, X):
        check_is_fitted(self, 'design_info_')
        return build_design_matrices([self.design_info_], X)[0]
```

```python
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
```

```python
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    0.00000
          1         1  2           2    0.69315
          1         0  3           0    1.09861
          1         0  4           0    1.38629
          1         1  5           5    1.60944
  Terms:
    'Intercept' (column 0)
    'a' (column 1)
    'x' (column 2)
    'x:a' (column 3)
    'np.log(x)' (column 4)
```

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

```
array([[ 0.    ,  0.8165, -1.4142, -0.3235, -1.6845],
       [ 0.    ,  0.8165, -0.7071,  0.2157, -0.4651],
       [ 0.    , -1.2247,  0.    , -0.8627,  0.2483],
       [ 0.    , -1.2247,  0.7071, -0.8627,  0.7544],
       [ 0.    ,  0.8165,  1.4142,  1.8332,  1.1469]])
```

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.

```python
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. Specifically it uses `endog` and `exog` to refer to `y` and `x` variable(s) respectively.

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.

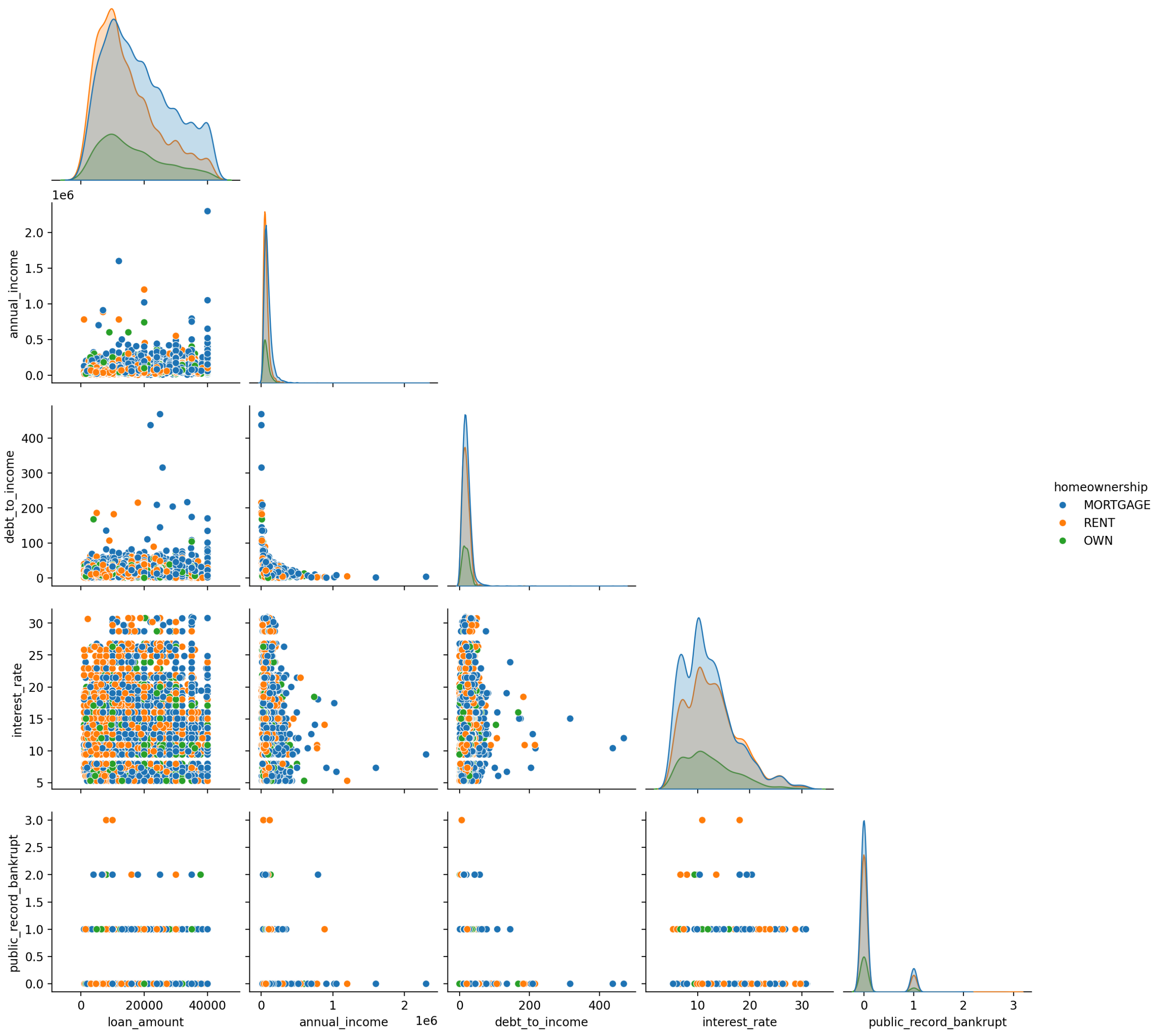
```
loans = pd.read_csv("data/openintro_loans.csv")
loans
```

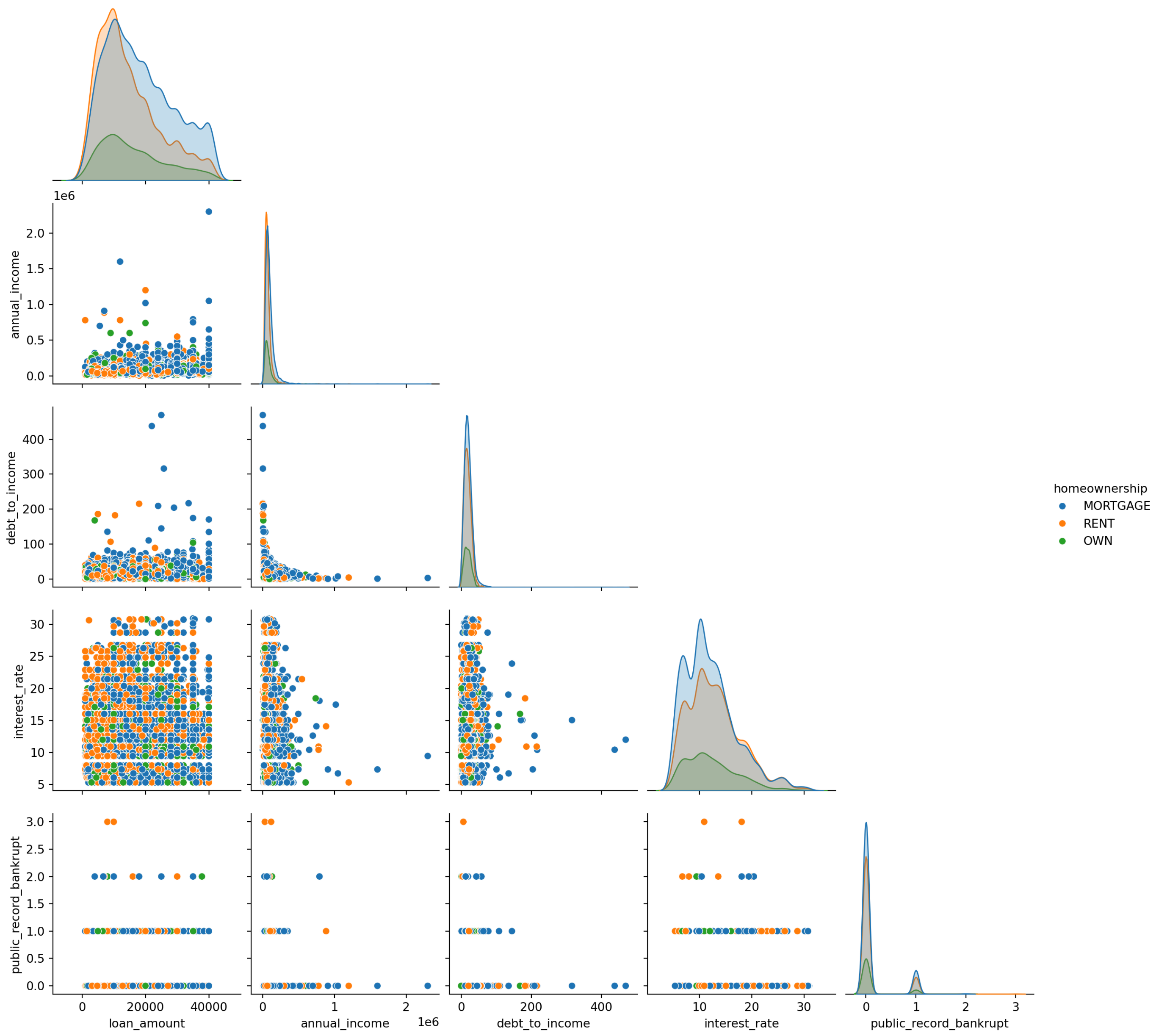
```
      state  emp_length  term homeownership  annual_income  ... loan_amount  grade  interest_rate  \
0        NJ           3    60      MORTGAGE        90000.0  ...       28000      C          14.07
1        HI          10    36          RENT        40000.0  ...        5000      C          12.61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

```
      public_record_bankrupt  loan_status
0                          0      Current
1                          1      Current
```

```
print(loans.columns)
```

```
Index(['state', 'emp_length', 'term', 'homeownership', 'annual_income', 'verified_income',
       'debt_to_income', 'total_credit_limit', 'total_credit_utilized',
'num_cc_carrying_balance',
       'loan_purpose', 'loan_amount', 'grade', 'interest_rate', 'public_record_bankrupt',
       'loan_status'],
      dtype='object')
```


annual_income
debt_to_income
interest_rate
public_record_bankrupt
loan_amount
homeownership
MORTGAGE
RENT
OWN
1e6



homeownership
MORTGAGE
RENT
OWN
annual_income
debt_to_income
interest_rate
public_record_bankrupt
loan_amount
annual_income
debt_to_income
interest_rate
public_record_bankrupt
1e6
1e6

OLS

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

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)
```

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

```
X_dc.dtypes
```

```
annual_income                float64
debt_to_income               float64
interest_rate                float64
public_record_bankrupt         int64
homeownership_MORTGAGE          bool
homeownership_OWN               bool
homeownership_RENT              bool
dtype: object
```

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

```python
model
```

```
<statsmodels.regression.linear_model.OLS object at 0x156a4d520>
```

```python
np.array(dir(model))
```

```
array(['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
       '__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__',
       '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__',
       '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
       '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
       '__weakref__', '_check_kwargs', '_data_attr', '_df_model', '_df_resid',
       '_fit_collinear', '_fit_ridge', '_fit_zeros', '_formula_max_endog',
       '_get_init_kwds', '_handle_data', '_init_keys', '_kwargs_allowed',
       '_setup_score_hess', '_sqrt_lasso', 'data', 'df_model', 'df_resid',
       'endog', 'endog_names', 'exog', 'exog_names', 'fit', 'fit_regularized',
       'from_formula', 'get_distribution', 'hessian', 'hessian_factor',
       'information', 'initialize', 'k_constant', 'loglike', 'nobs',
       'pinv_wexog', 'predict', 'rank', 'score', 'weights', 'wendog', 'wexog',
       'whiten'], dtype='<U18')
```

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, 21 Feb 2025   Prob (F-statistic):          2.33e-285
Time:                        09:18:11   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]
------------------------------------------------------------------------------
```

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.

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

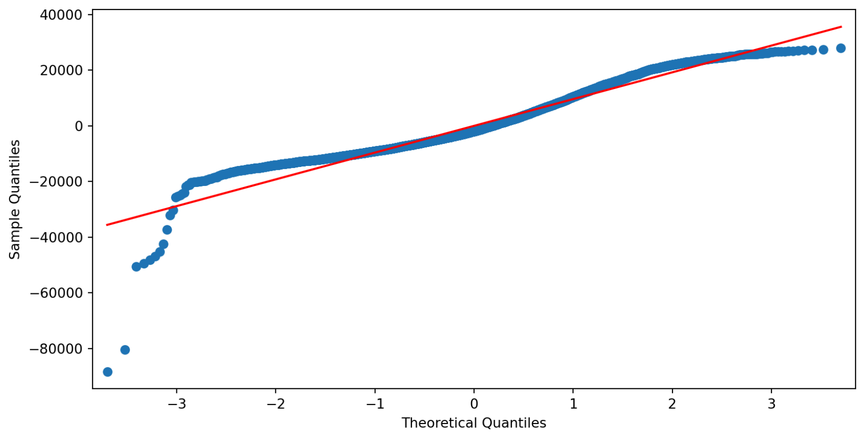
```
                            OLS Regression Results
==============================================================================
Dep. Variable:            loan_amount   R-squared:                       0.135
Model:                            OLS   Adj. R-squared:                  0.135
Method:                 Least Squares   F-statistic:                     239.5
Date:                Fri, 21 Feb 2025   Prob (F-statistic):          2.33e-285
Time:                        09:18:11   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
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
==============================================================================
Omnibus:                      481.833   Durbin-Watson:                   2.002
```

Result values and model parameters

Diagnostic plots

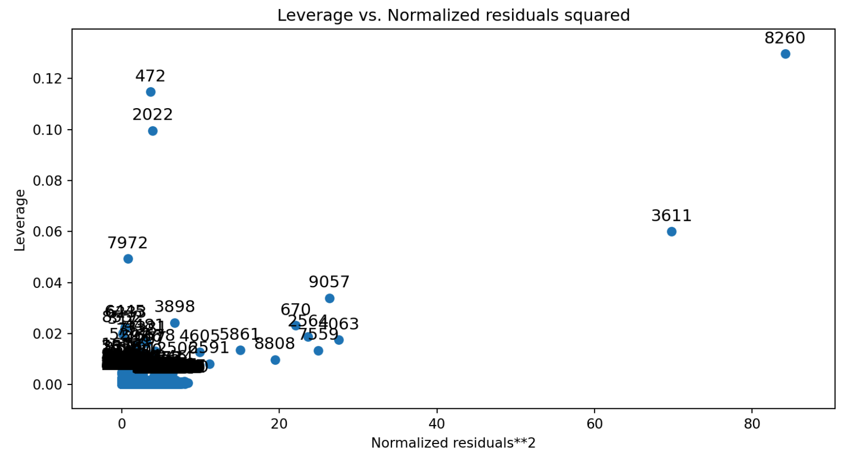
QQ Plot

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



Leverage plot

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



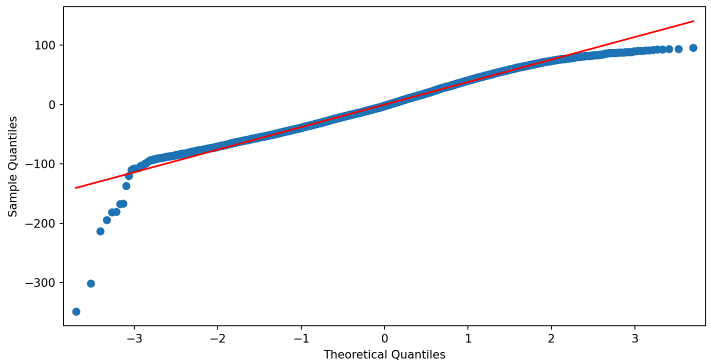
Alternative model

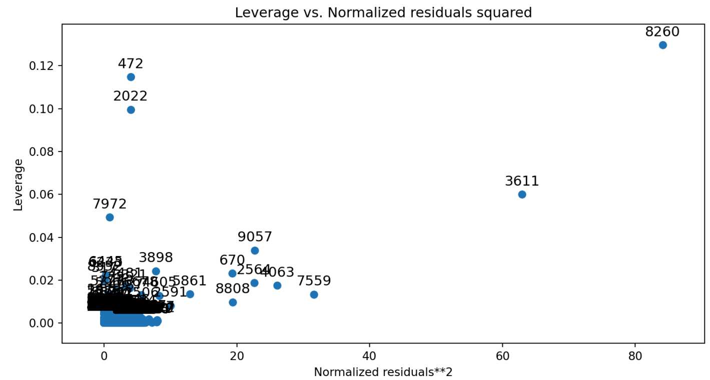
```
res = smf.ols(
  "np.sqrt(loan_amount) ~ homeownership + annual_income + debt_to_income + interest_rate + public_record_bankrupt",
  data = loans
).fit()
print(res.summary())
```

```
                            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, 21 Feb 2025   Prob (F-statistic):          1.16e-277
Time:                          09:18:12   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
```

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

```python
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.

```python
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

[104 rows x 8 columns]
```

Logistic regression models (GLM)

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

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

```
statsmodels.tools.sm_exceptions.MissingDataError: exog contains inf or nans
```

What went wrong this time?

Missing values

Missing values can be handled via `missing` argument, possible values are `"none"`, `"drop"`, and `"raise"`.

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

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].

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

res = sm.GLM(y, X, family = sm.families.Binomial(), missing="drop").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, 21 Feb 2025   Deviance:                       63.885
Time:                        09:18:13   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

```python
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, 21 Feb 2025   Deviance:                       63.885
Time:                                09:18:13   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
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
==============================================================================
```

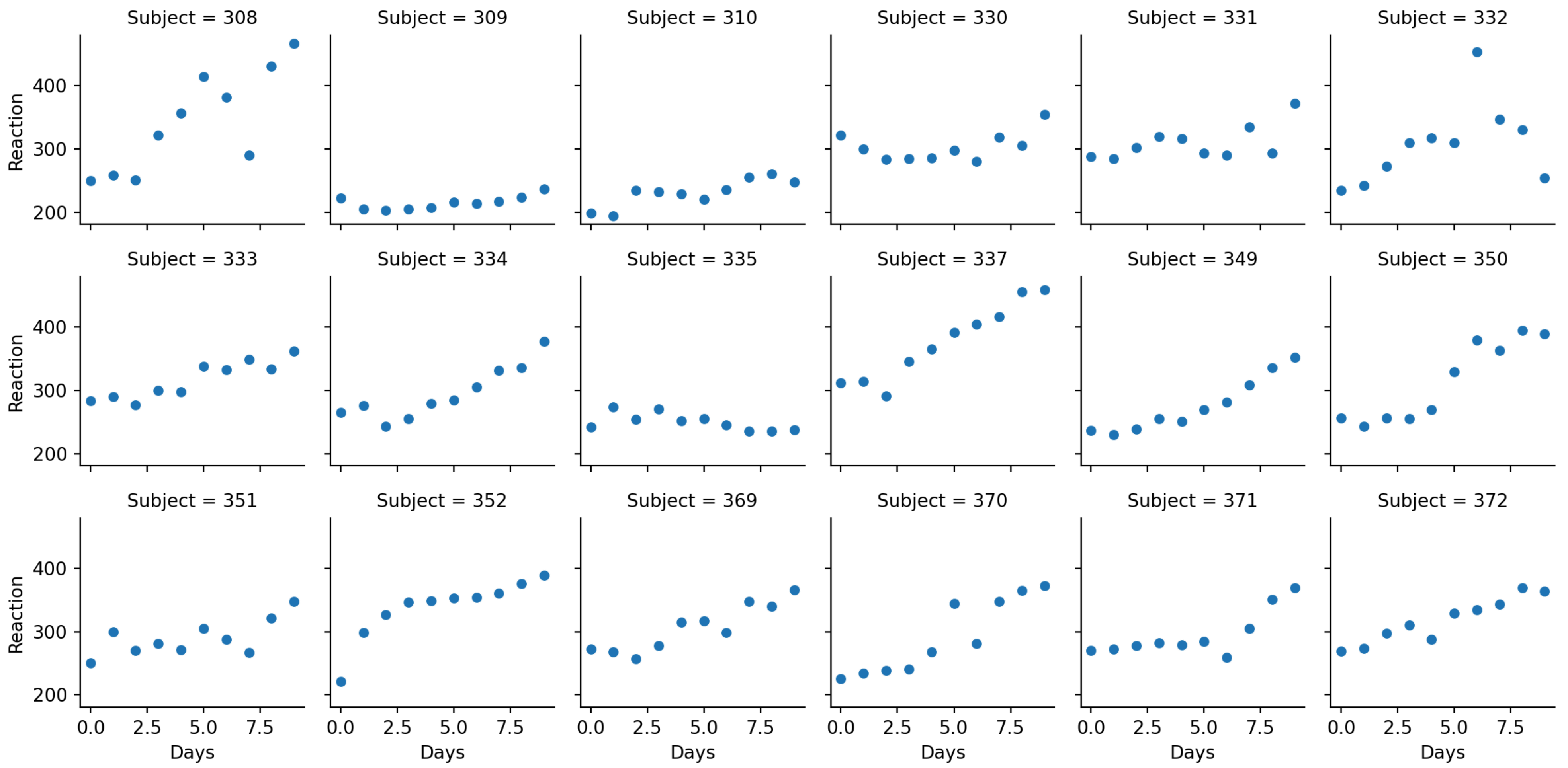
sleepstudy data

> 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.

```python
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
176  334.4818     6      372
177  343.2199     7      372
178  369.1417     8      372
179  364.1236     9      372

[180 rows x 3 columns]
```

```python
g = sns.relplot(x="Days", y="Reaction", col="Subject", col_wrap=6, data=sleep, height=2)
```



Random intercept model

```
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())
```

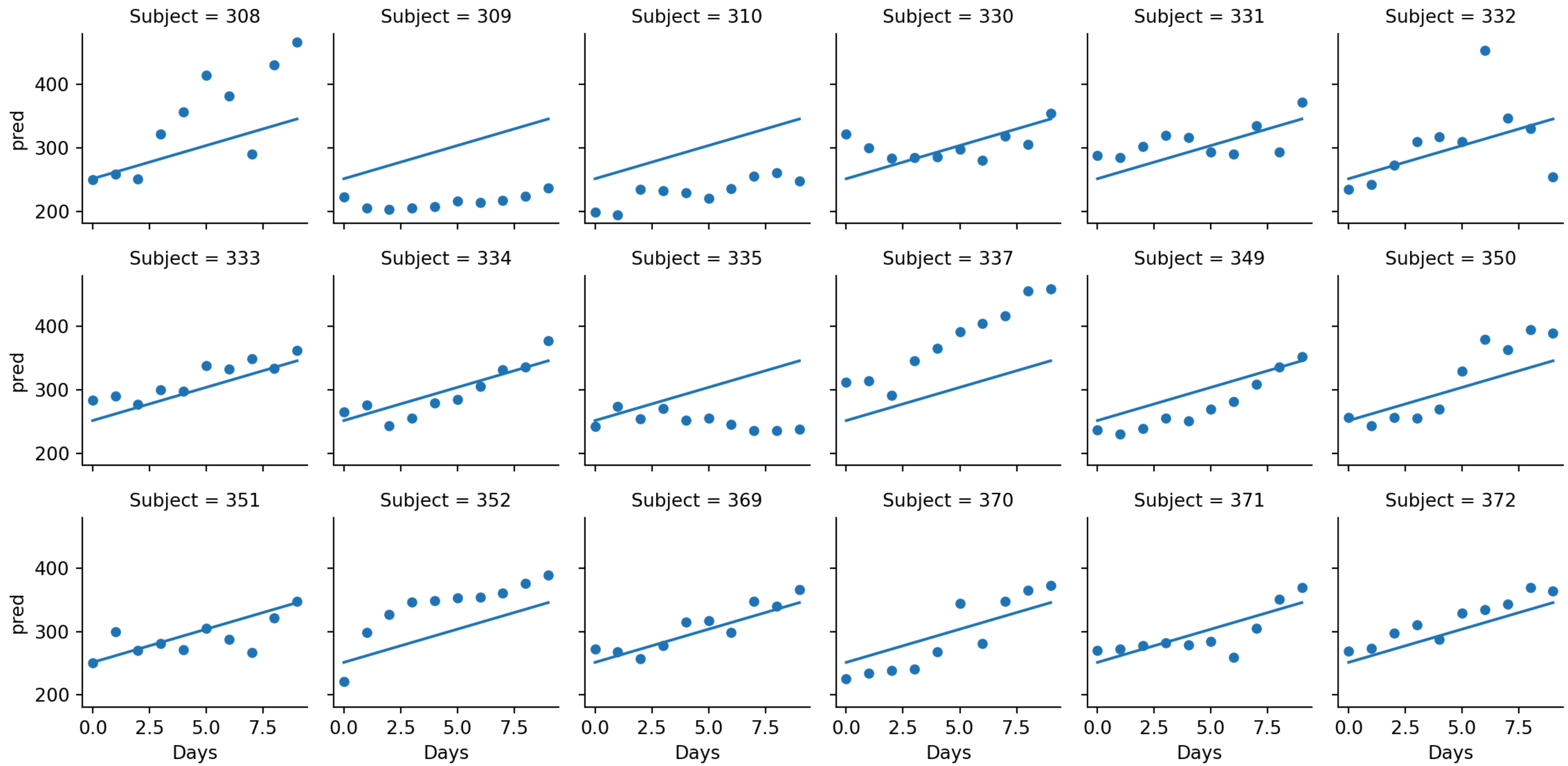
```
         Mixed Linear Model Regression Results
========================================================
Model:               MixedLM Dependent Variable: Reaction
No. Observations:    180     Method:             REML
No. Groups:          18      Scale:              960.4529
Min. group size:     10      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
========================================================
```

lme4 version

```
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
```

Predictions

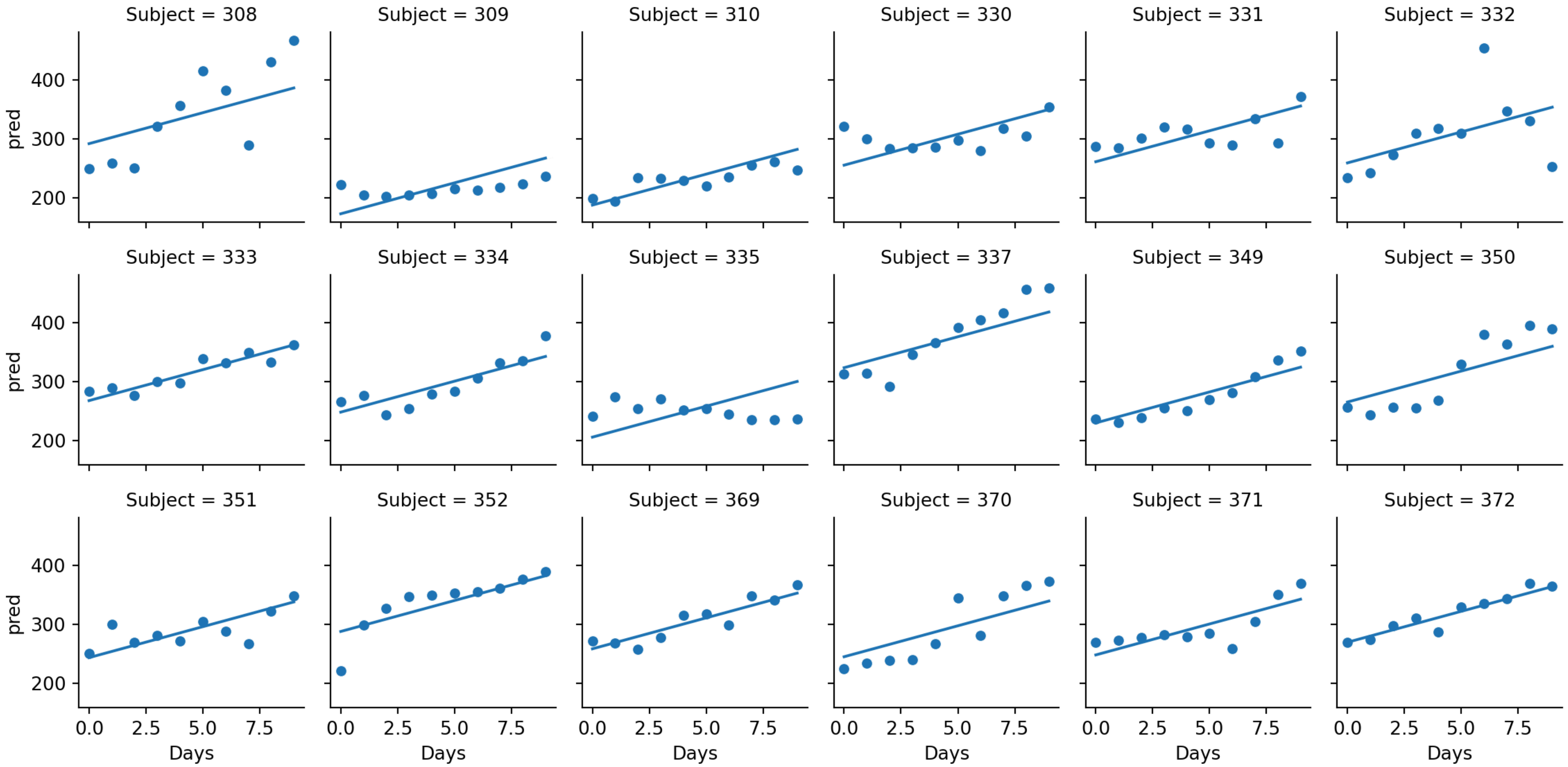


Recovering random effects for prediction

```
# Multiply each RE by the random effects design matrix for each group
rex = [
  np.dot(
    me_rand_int.exog_re_li[j],
    res_rand_int.random_effects[k]
  )
  for (j, k) in enumerate(me_rand_int.group_labels)
]
rex[0]
```

```
array([40.7838, 40.7838, 40.7838, 40.7838, 40.7838, 40.7838, 40.7838, 40.7838,
       40.7838, 40.7838])
```

```

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


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
200
300
400
Days
0.0
2.5
5.0
7.5

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())
```

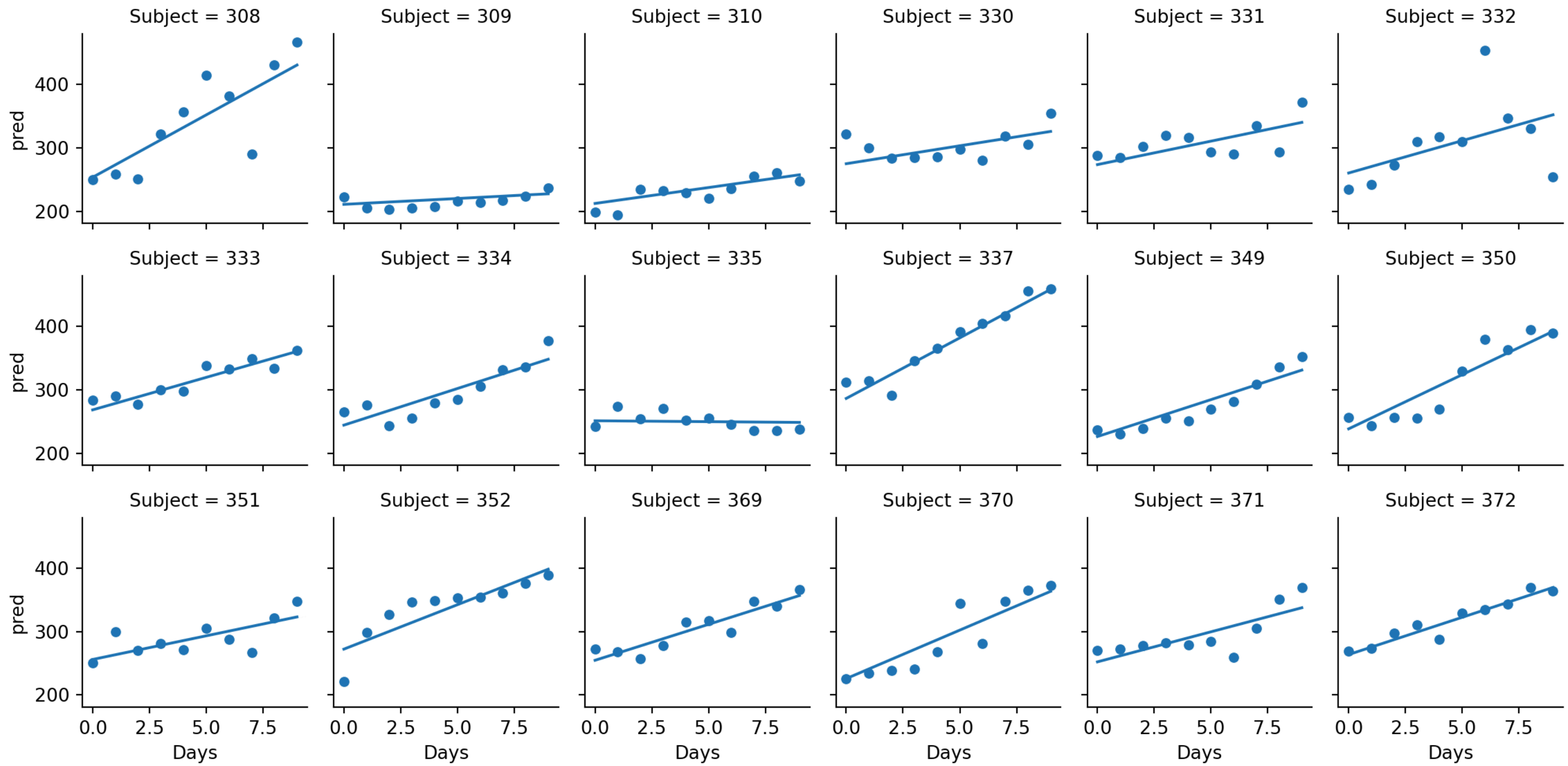
```
            Mixed Linear Model Regression Results
==============================================================
Model:                MixedLM   Dependent Variable:   Reaction
No. Observations:     180       Method:               REML
No. Groups:           18        Scale:                654.9412
Min. group size:      10        Log-Likelihood:       -871.8141
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 0.000 238.029 264.781
Days              10.467    1.546  6.771 0.000   7.438  13.497
Group Var        612.089   11.881
Group x Days Cov   9.605    1.820
Days Var          35.072    0.610
==============================================================
```

lme4 version

```r
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
```

Prediction



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