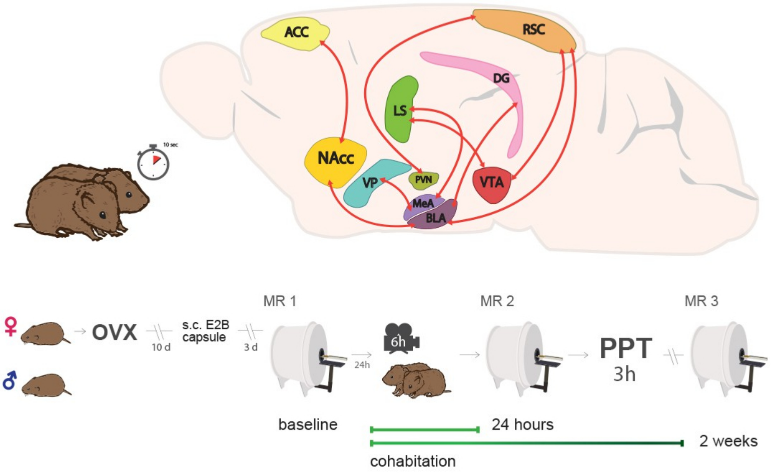
NBpy: Network-based (R)Statistics in Python

Ljuba, Waleed, & Zeus

July 30, 2021

Context

Longitudinal brain networks

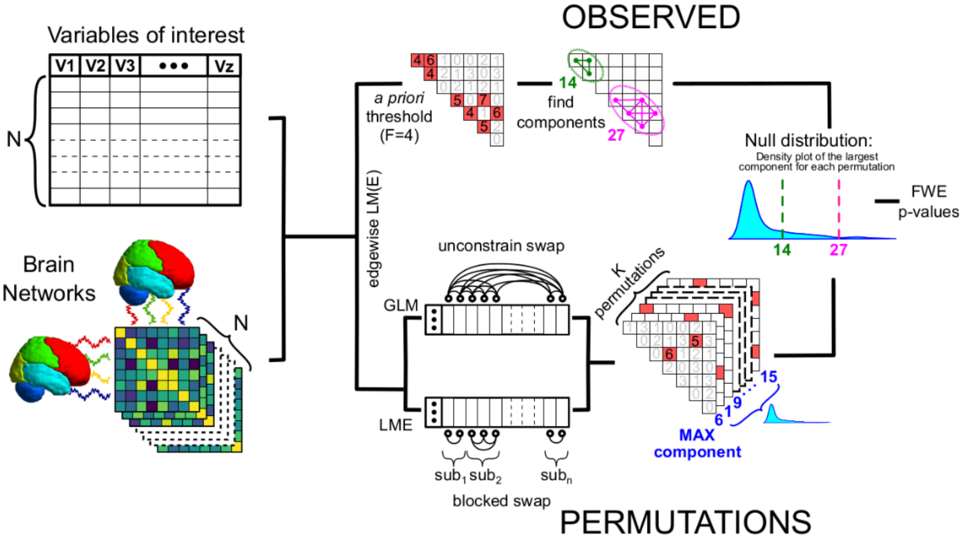


(López-Gutierrez et al., *eLife*, 2021)

Context

Prairie voles

Network Based Statistics framework



(Gracia-Tabuenca & Alcauter, *bioRxiv*, 2021)

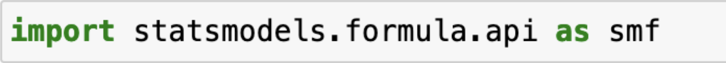
NBR

- First CRAN release 0.1.2 (March 2020)
  - `https://cran.r-project.org/package=NBR`
  - $>$6k downloads (`https://cranlogs.r-pkg.org/badges/grand-total/NBR`)
  - Computing efficiency is the biggest challenge, according to the mails from various users.

Python implementation

R2Py

- **statsmodels** is a python framework designed for statistics, and allows users to fit statistical models using R-style formulas.
- Simply import statsmodels' R-API:

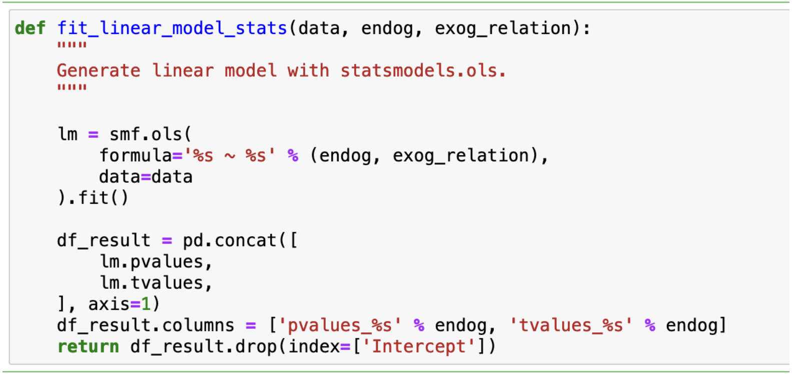


```python
import statsmodels.formula.api as smf
```

- And start R-hacking in Python, with API customization.

statsmodel implementation

- Statsmodel execution for 1000 randomly selected endog columns finished on average: 3.22 ± 0.73 [ms]



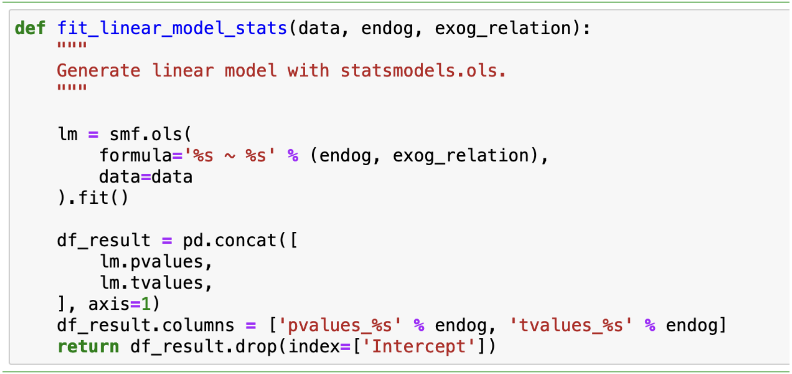
```
def fit_linear_model_stats(data, endog, exog_relation):
    """
    Generate linear model with statsmodels.ols.
    """

    lm = smf.ols(
        formula='%s ~ %s' % (endog, exog_relation),
        data=data
    ).fit()

    df_result = pd.concat([
        lm.pvalues,
        lm.tvalues,
    ], axis=1)
    df_result.columns = ['pvalues_%s' % endog, 'tvalues_%s' % endog]
    return df_result.drop(index=['Intercept'])
```

statsmodel

- Statsmodel execution for 1000 randomly selected endog columns finished on average: 3.22 ± 0.73 [ms]



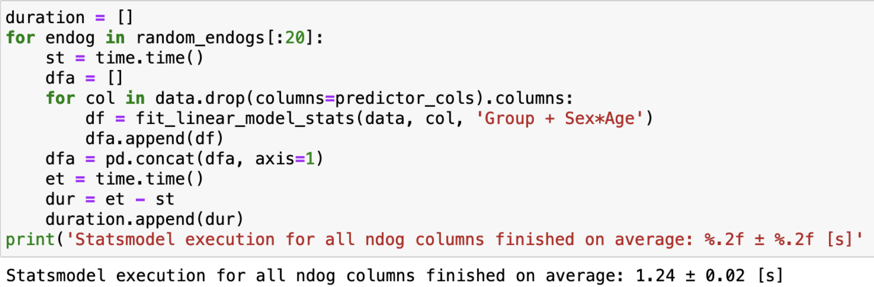
```
def fit_linear_model_stats(data, endog, exog_relation):
    """
    Generate linear model with statsmodels.ols.
    """

    lm = smf.ols(
        formula='%s ~ %s' % (endog, exog_relation),
        data=data
    ).fit()

    df_result = pd.concat([
        lm.pvalues,
        lm.tvalues,
    ], axis=1)
    df_result.columns = ['pvalues_%s' % endog, 'tvalues_%s' % endog]
    return df_result.drop(index=['Intercept'])
```

statsmodel

- If we run linear regression across all endog columns (interconnections)

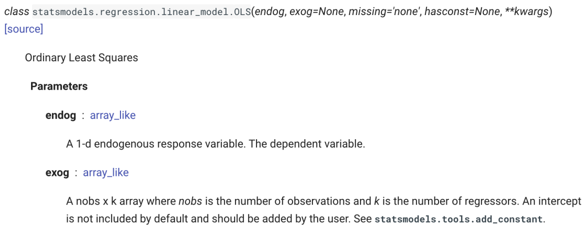


```
duration = []
for endog in random_endogs[:20]:
    st = time.time()
    dfa = []
    for col in data.drop(columns=predictor_cols).columns:
        df = fit_linear_model_stats(data, col, 'Group + Sex*Age')
        dfa.append(df)
    dfa = pd.concat(dfa, axis=1)
    et = time.time()
    dur = et - st
    duration.append(dur)
print('Statsmodel execution for all ndog columns finished on average: %.2f ± %.2f [s]'
```

```
Statsmodel execution for all ndog columns finished on average: 1.24 ± 0.02 [s]
```

statsmodel

- Can we improve performance by computing linear models across all input endog columns, by utilizing matrix computations?
- For that reason we will call Pythonic API to handle input data with statsmodels.OLS.



class `statsmodels.regression.linear_model.OLS`(*endog, exog=None, missing='none', hasconst=None, **kwargs*) [source]

Ordinary Least Squares

Parameters

endog : array_like

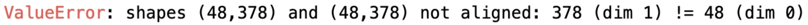
A 1-d endogenous response variable. The dependent variable.

exog : array_like

A nobs x k array where *nobs* is the number of observations and *k* is the number of regressors. An intercept is not included by default and should be added by the user. See `statsmodels.tools.add_constant`.

statsmodel

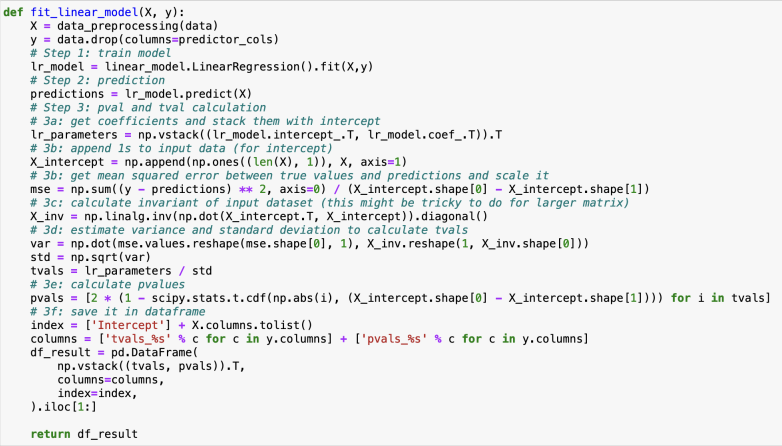
- Not possible to have an endog input with dimension higher than 1.



```
ValueError: shapes (48,378) and (48,378) not aligned: 378 (dim 1) != 48 (dim 0)
```

sklearn/numpy

- sklearn/numpy to the rescue.
- We can perform matrix/broadcasting operations on a lowest level of abstraction.

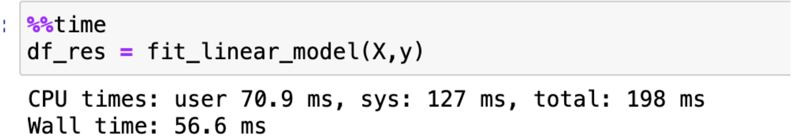


```
def fit_linear_model(X, y):
    X = data_preprocessing(data)
    y = data.drop(columns=predictor_cols)
    # Step 1: train model
    lr_model = linear_model.LinearRegression().fit(X,y)
    # Step 2: prediction
    predictions = lr_model.predict(X)
    # Step 3: pval and tval calculation
    # 3a: get coefficients and stack them with intercept
    lr_parameters = np.vstack((lr_model.intercept_.T, lr_model.coef_.T)).T
    # 3b: append 1s to input data (for intercept)
    X_intercept = np.append(np.ones((len(X), 1)), X, axis=1)
    # 3b: get mean squared error between true values and predictions and scale it
    mse = np.sum((y - predictions) ** 2, axis=0) / (X_intercept.shape[0] - X_intercept.shape[1])
    # 3c: calculate invariant of input dataset (this might be tricky to do for larger matrix)
    X_inv = np.linalg.inv(np.dot(X_intercept.T, X_intercept)).diagonal()
    # 3d: estimate variance and standard deviation to calculate tvals
    var = np.dot(mse.values.reshape(mse.shape[0], 1), X_inv.reshape(1, X_inv.shape[0]))
    std = np.sqrt(var)
    tvals = lr_parameters / std
    # 3e: calculate pvalues
    pvals = [2 * (1 - scipy.stats.t.cdf(np.abs(i), (X_intercept.shape[0] - X_intercept.shape[1]))) for i in tvals]
    # 3f: save it in dataframe
    index = ['Intercept'] + X.columns.tolist()
    columns = ['tvals_%s' % c for c in y.columns] + ['pvals_%s' % c for c in y.columns]
    df_result = pd.DataFrame(
        np.vstack((tvals, pvals)).T,
        columns=columns,
        index=index,
    ).iloc[1:]

    return df_result
```

sklearn/numpy

- If we run linear regression across all endog columns (interconnections)
- We achieve improvement of calculating linear regression models for all input endog columns from 1.24 seconds to 56.6ms

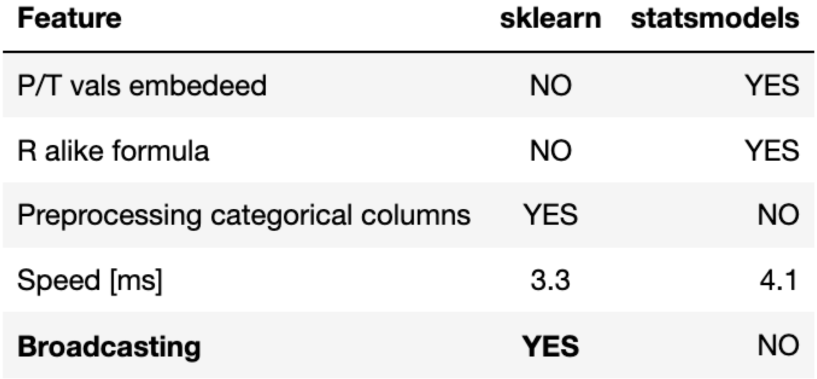


```
%%time
df_res = fit_linear_model(X,y)
```

```
CPU times: user 70.9 ms, sys: 127 ms, total: 198 ms
Wall time: 56.6 ms
```

sklearn/numpy

- Although statsmodel has better support for statistical calculations (ie. faster computation of p and t vals), p and t vals calculation is embedeed into package.



Feature	sklearn	statsmodels
P/T vals embedeed	NO	YES
R alike formula	NO	YES
Preprocessing categorical columns	YES	NO
Speed [ms]	3.3	4.1
Broadcasting	**YES**	NO

Conclusion

Conclusion

Conclusion

- R implementation has reach statistical libraries.
- On contrary, Python requires implementation of p and t values.
- However, it is faster to load bigger datasets with Python and it allows broadcasting/matrix operations.
- **Improvements**:
  - Instead of **sklearn** we can utilize **numpy.linalg.lstsq** library to compute coefficients of linear regression.
  - Improve calculation of **FWE** strength values.