# Week 9 - Modeling through Statsmodels, Sklearn

# Customization vs Rapid Development

Building our own models is great!

- Understand the assumptions
- Get EXACTLY what you need

Unfortunately, it takes a lot of time!

# Statsmodels

# Importing Statsmodels

We can import `statsmodels` in one of two ways:

1. With support for R-style formulas:

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

2. Using pre-built numpy arrays as inputs:

```
import statsmodels.api as sm
```

We will focus on option (1)

# Preparing a Dataset

When using formulas, we prepare our dataset by importing the data into a Pandas `DataFrame`. We should take care that each of our variables has a name with

1. **No spaces**
2. No symbols
3. Made up of letters and numbers (also can't have a number as the first character)

# Preparing a Dataset

Our code so far might look something like:

```
import statsmodels.formula.api as sm
from sqlalchemy import create_engine
import pandas as pd, numpy as np

engine = create_engine(
  'mysql+mysqlconnector://viewer:@dadata.cba.edu:3306/ACS'
          )

SELECT = """SELECT AVG(hhincome) AS hhincome, year,
    statefip
  FROM ACS
  GROUP BY year, statefip
  ORDER BY year, statefip"""

data = pd.read_sql(SELECT, engine)
```

# Implementing a Model

The first thing we might try is a simple linear regression:

```
reg = sm.ols("hhincome ~ year", data=data).fit()
print(reg.summary())
```

Or, I might want to try regressing year on the logged average household incomes:

```
reg = sm.ols("np.log(hhincome) ~ year", data=data).fit()
print(reg.summary())
```

# Advancing our Model

It might be useful to create state-level fixed effects by including dummy variables for the states in our `statefip` column.

```
reg = sm.ols("np.log(hhincome) ~ year + C(statefip)",
        data=data).fit()
print(reg.summary())
```

The `C()` command indicates that we would like to consider the `statefip` variable as a **C**ategorical variable, not a numeric variable.

# Additional Transformations

Sometimes we want to include transformed variables in our model:

```
# Square a variable using the I() function for
#   mathematical transformations
reg = sm.ols("np.log(hhincome) ~ age + I(age**2)",
        data=data).fit()
```

```
# Combine variables using the I() function for
#   mathematical transformations
reg = sm.ols("np.log(hhincome) ~ I(age-education-5)",
        data=data).fit()
```

# Robust Modeling

If we want to utilize robust standard errors, we can update our regression results:

```
reg = sm.ols("np.log(hhincome) ~ year + C(statefip)",
        data=data).fit()
# Use White's (1980) Standard Error
reg.get_robustcov_results(cov_type='HC0')
print(reg.summary())

-----------------------------------------------------------

reg = sm.ols("np.log(hhincome) ~ year + C(statefip)",
        data=data).fit()
# Use Cluster-robust Standard Errors
reg.get_robustcov_results(cov_type='cluster',
        groups=data['statefip']) # Need to specify groups
print(reg.summary())
```

# Robust Modeling

Below are some of the covariance options that we have:

1. `HC0`: White's (1980) Heteroskedasticity robust standard errors
2. `HC1`, `HC2`, `HC3`: MacKinnon and White's (1985) alternative robust standard errors, with `HC3` being designed for improved performance in small samples
3. `cluster`: Cluster robust standard errors
4. `hac-panel`: Panel robust standard errors

# Time Series Models

We have multiple time series options available.

To implement an ARIMA(1,1,0) model:

```
from statsmodels.tsa.arima_model import ARIMA

y = data.loc[data['statefip']==31, ['hhincome','year']]
y.index=pd.to_datetime(y.year)
reg = ARIMA(y['hhincome'], order=(1,1,0)).fit()
print(reg.summary())
```

# Time Series Models

To implement a VAR model:

```
from statsmodels.tsa.vector_ar.var_model import VAR

y = data.loc[data['statefip']==31, ['hhincome','year']]
y.index=pd.to_datetime(y.year)
reg = VAR(y['hhincome']).fit()
print(reg.summary())
```

The VAR model will optimize its own order (number of lags included) based on information criteria estimates.

# Modeling Discrete Outcomes

If we have a binary dependent variable, we are able to use either Logit or Probit models to estimate the effect of exogenous variables on our outcome of interest. To fit a Logit model:

```
from scipy import stats
stats.chisqprob = lambda chisq,
                    df: stats.chi2.sf(chisq, df)
# Previous lines fix a temporary problem between
#   statsmodels and scipy's chi square distribution
data = pd.read_csv("auto-mpg.csv")
data['highMPG'] = (data['mpg']>30).astype(np.int)

myformula="highMPG ~ cylinders + displacement + weight"
model= sm.Logit.from_formula(myformula, data=data).fit()
```

# Modeling Discrete Outcomes

When modeling count data, we have options such as [Poisson]() and [Negative Binomial]() models.

```
from scipy import stats
stats.chisqprob = lambda chisq,
                    df: stats.chi2.sf(chisq, df)
# Previous lines fix a temporary problem between
#   statsmodels and scipy's chi square distribution
data = pd.read_csv("auto-mpg.csv")

myformula="mpg ~ cylinders + displacement + weight"
model= sm.Poisson.from_formula(myformula, data=data).fit()
```

# scikit-learn

# Predictive Modeling

What `statsmodels` does for regression analysis, `sklearn` does for predictive analytics and machine learning.

- Likely the most popular machine learning library currently in use
- Has a standard API to make using the library VERY simple.

# Decision Tree Classification (and Regression)

[Classification](#) and [Regression](#) Trees (CARTs) are the standard jumping-off point for exploring machine learning. They are very easy to implement in `sklearn`:

```
from sklearn import tree
from sklearn.metrics import accuracy_score

clf = tree.DecisionTreeClassifier()
clf = clf.fit(x, y)

pred = clf.pred(new_xs)

print(accuracy_score(new_ys, pred))
```

# Support Vector Machines

We also implement Support Vector Machines for both classification and regression:

```
from sklearn import svm
from sklearn.metrics import accuracy_score

clf = svm.SVC()
clf = clf.fit(x, y)

pred = clf.pred(new_xs)

print(accuracy_score(new_ys, pred))
```

Can you see the API pattern yet?

# Random Forest Models

Again, available in both classification and regression flavors, these models are aggregations of many randomized Decision Trees.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

clf = RandomForestClassifier(n_estimators=100)
clf = clf.fit(x, y)

pred = clf.pred(new_xs)

print(accuracy_score(new_ys, pred)
```

There MUST be a pattern here...

# Data Preprocessing

Many other tools are also available to aid in the data cleaning process through `sklearn`. Some of these are:

- Principal Component Analysis (PCA)
- Factor Analysis
- Many Cross-Validation Algorithms
- Hyperparameter Tuning
  - Finding the correct parameters for a decision tree or random forest, for example
- Model Evaluation Tools

# For Lab Today

Work on the homework assignment. You will practice using both `statsmodels` and `sklearn` in order to apply regression and predictive models to data.

You will want to make sure that you collect the data before you leave the lab, since the homework will require data from the databases available through the data server here in Mammel Hall.