# Logistic Regression with scikit-learn

This is an example of logistic regression in Python with the scikit-learn module (http://scikit-learn.org/), performed for an assignment (https://github.com/ajschumacher/gadsdc/tree/master/logistic_assignment) with my General Assembly Data Science class (https://generalassemb.ly/education/data-science).

## Dataset

The dataset I chose is the affairs dataset (http://statsmodels.sourceforge.net/stable/datasets/generated/fair.html) that comes with Statsmodels (http://statsmodels.sourceforge.net/). It was derived from a survey of women in 1974 by Redbook magazine, in which married women were asked about their participation in extramarital affairs. More information about the study is available in a 1978 paper (http://fairmodel.econ.yale.edu/rayfair/pdf/1978a200.pdf) from the Journal of Political Economy.

## Description of Variables

The dataset contains 6366 observations of 9 variables:

- `rate_marriage` : woman's rating of her marriage (1 = very poor, 5 = very good)
- `age` : woman's age
- `yrs_married` : number of years married
- `children` : number of children
- `religious` : woman's rating of how religious she is (1 = not religious, 4 = strongly religious)
- `educ` : level of education (9 = grade school, 12 = high school, 14 = some college, 16 = college graduate, 17 = some graduate school, 20 = advanced degree)
- `occupation` : woman's occupation (1 = student, 2 = farming/semi-skilled/unskilled, 3 = "white collar", 4 = teacher/nurse/writer/technician/skilled, 5 = managerial/business, 6 = professional with advanced degree)
- `occupation_husb` : husband's occupation (same coding as above)
- `affairs` : time spent in extra-marital affairs

## Problem Statement

I decided to treat this as a classification problem by creating a new binary variable `affair` (did the woman have at least one affair?) and trying to predict the classification for each woman.

Skipper Seabold, one of the primary contributors to Statsmodels, did a similar classification in his Statsmodels demo (https://github.com/jseabold/pydc) at a Statistical Programming DC Meetup (http://www.meetup.com/stats-prog-dc/events/173693192/). However, he used Statsmodels for the classification (whereas I'm using scikit-learn), and he treated the occupation variables as continuous (whereas I'm treating them as categorical).

# Import modules

In [1]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from patsy import dmatrices
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import train_test_split
from sklearn import metrics
from sklearn.cross_validation import cross_val_score
```

# Data Pre-Processing

First, let's load the dataset and add a binary `affair` column.

In [2]:
```
# load dataset
dta = sm.datasets.fair.load_pandas().data

# add "affair" column: 1 represents having affairs, 0 represents not
dta['affair'] = (dta.affairs > 0).astype(int)
```

# Data Exploration

In [3]:
```
dta.groupby('affair').mean()
```

Out[3]:

| affair | rate_marriage | age | yrs_married | children | religious | educ | occupation | occupation_hu |
|---|---|---|---|---|---|---|---|---|
| **0** | 4.329701 | 28.390679 | 7.989335 | 1.238813 | 2.504521 | 14.322977 | 3.405286 | 3.8337 |
| **1** | 3.647345 | 30.537019 | 11.152460 | 1.728933 | 2.261568 | 13.972236 | 3.463712 | 3.8845 |

We can see that on average, women who have affairs rate their marriages lower, which is to be expected. Let's take another look at the `rate_marriage` variable.

In [4]:
```
dta.groupby('rate_marriage').mean()
```

Out[4]:

| rate_marriage | age | yrs_married | children | religious | educ | occupation | occupation_husb | a |
|---|---|---|---|---|---|---|---|---|
| **1** | 33.823232 | 13.914141 | 2.308081 | 2.343434 | 13.848485 | 3.232323 | 3.838384 | 1.2 |
| **2** | 30.471264 | 10.727011 | 1.735632 | 2.330460 | 13.864943 | 3.327586 | 3.764368 | 1.6 |
| **3** | 30.008056 | 10.239174 | 1.638469 | 2.308157 | 14.001007 | 3.402820 | 3.798590 | 1.3 |
| **4** | 28.856601 | 8.816905 | 1.369536 | 2.400981 | 14.144514 | 3.420161 | 3.835861 | 0.6 |
| **5** | 28.574702 | 8.311662 | 1.252794 | 2.506334 | 14.399776 | 3.454918 | 3.892697 | 0.3 |

An increase in `age`, `yrs_married`, and `children` appears to correlate with a declining marriage rating.

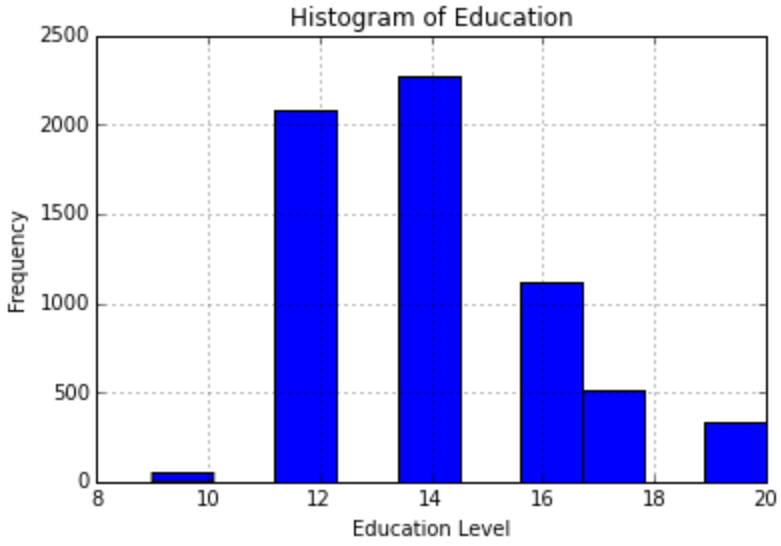
# Data Visualization

In [5]:
```
# show plots in the notebook
%matplotlib inline
```

Let's start with histograms of education and marriage rating.

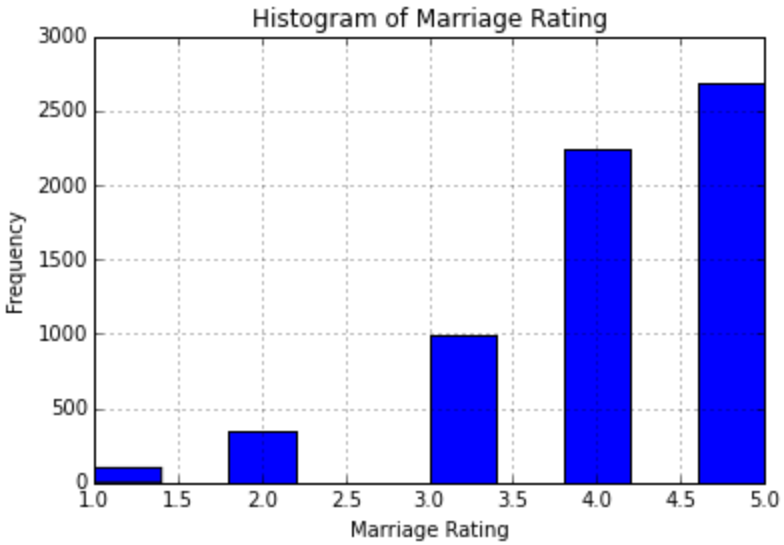
In [6]:
```
# histogram of education
dta.educ.hist()
plt.title('Histogram of Education')
plt.xlabel('Education Level')
plt.ylabel('Frequency')
```

Out[6]: `<matplotlib.text.Text at 0x16e48ac8>`




In [7]:
```
# histogram of marriage rating
dta.rate_marriage.hist()
plt.title('Histogram of Marriage Rating')
plt.xlabel('Marriage Rating')
plt.ylabel('Frequency')
```

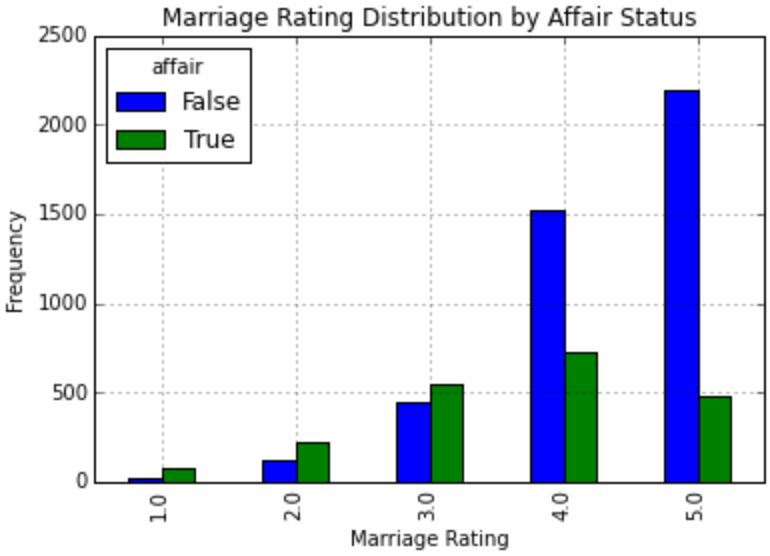
Out[7]: <matplotlib.text.Text at 0x16eac550>




Let's take a look at the distribution of marriage ratings for those having affairs versus those not having affairs.

In [8]:
```
# barplot of marriage rating grouped by affair (True or False)
pd.crosstab(dta.rate_marriage, dta.affair.astype(bool)).plot(kind='bar')
plt.title('Marriage Rating Distribution by Affair Status')
plt.xlabel('Marriage Rating')
plt.ylabel('Frequency')
```

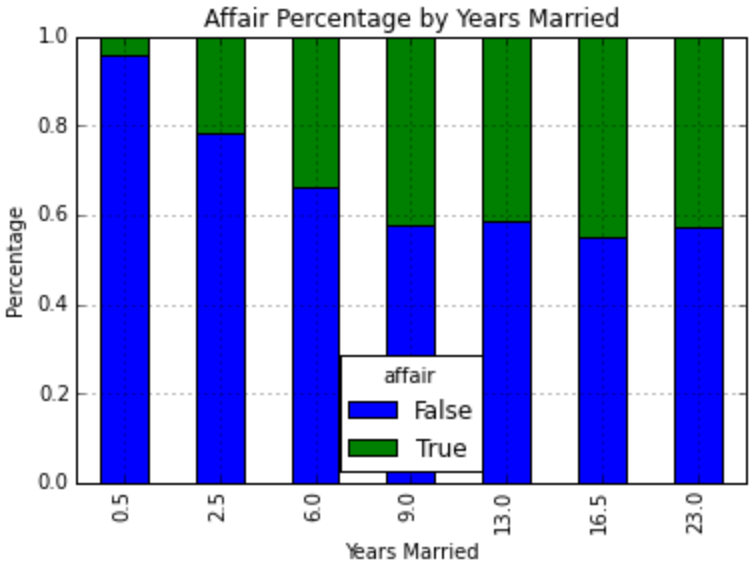
Out[8]: <matplotlib.text.Text at 0x1710c5f8>




Let's use a stacked barplot to look at the percentage of women having affairs by number of years of marriage.

In [9]:
```
affair_yrs_married = pd.crosstab(dta.yrs_married, dta.affair.astype(bool))
affair_yrs_married.div(affair_yrs_married.sum(1).astype(float), axis=0).plot(kind='bar'
plt.title('Affair Percentage by Years Married')
plt.xlabel('Years Married')
plt.ylabel('Percentage')
```

Out[9]: <matplotlib.text.Text at 0x17d83a20>




# Prepare Data for Logistic Regression

To prepare the data, I want to add an intercept column as well as dummy variables for `occupation` and `occupation_husb`, since I'm treating them as categorial variables. The dmatrices function from the patsy module (http://patsy.readthedocs.org/en/latest/) can do that using formula language.

In [10]:
```
# create dataframes with an intercept column and dummy variables for
# occupation and occupation_husb
y, X = dmatrices('affair ~ rate_marriage + age + yrs_married + children + \
                  religious + educ + C(occupation) + C(occupation_husb)',
                  dta, return_type="dataframe")
print X.columns
```

```
Index([u'Intercept', u'C(occupation)[T.2.0]', u'C(occupation)[T.3.0]', u'C(occupation)
[T.4.0]', u'C(occupation)[T.5.0]', u'C(occupation)[T.6.0]', u'C(occupation_husb)[T.2.
0]', u'C(occupation_husb)[T.3.0]', u'C(occupation_husb)[T.4.0]', u'C(occupation_husb)
[T.5.0]', u'C(occupation_husb)[T.6.0]', u'rate_marriage', u'age', u'yrs_married', u'ch
ildren', u'religious', u'educ'], dtype='object')
```

The column names for the dummy variables are ugly, so let's rename those.

In [11]:
```
# fix column names of X
X = X.rename(columns = {'C(occupation)[T.2.0]':'occ_2',
                        'C(occupation)[T.3.0]':'occ_3',
                        'C(occupation)[T.4.0]':'occ_4',
                        'C(occupation)[T.5.0]':'occ_5',
                        'C(occupation)[T.6.0]':'occ_6',
                        'C(occupation_husb)[T.2.0]':'occ_husb_2',
                        'C(occupation_husb)[T.3.0]':'occ_husb_3',
                        'C(occupation_husb)[T.4.0]':'occ_husb_4',
                        'C(occupation_husb)[T.5.0]':'occ_husb_5',
                        'C(occupation_husb)[T.6.0]':'occ_husb_6'})
```

We also need to flatten `y` into a 1-D array, so that scikit-learn will properly understand it as the response variable.

In [12]:
```
# flatten y into a 1-D array
y = np.ravel(y)
```

# Logistic Regression

Let's go ahead and run logistic regression on the entire data set, and see how accurate it is!

In [13]:
```
# instantiate a logistic regression model, and fit with X and y
model = LogisticRegression()
model = model.fit(X, y)

# check the accuracy on the training set
model.score(X, y)
```

Out[13]:
```
0.72588752748978946
```

73% accuracy seems good, but what's the null error rate?

In [14]:
```
# what percentage had affairs?
y.mean()
```

Out[14]:
```
0.32249450204209867
```

Only 32% of the women had affairs, which means that you could obtain 68% accuracy by always predicting "no". So we're doing better than the null error rate, but not by much.

Let's examine the coefficients to see what we learn.

In [15]:
```
# examine the coefficients
pd.DataFrame(zip(X.columns, np.transpose(model.coef_)))
```

Out[15]:

| | 0 | 1 |
|---|---|---|
| **0** | Intercept | [1.48988372957] |
| **1** | occ_2 | [0.188045598942] |
| **2** | occ_3 | [0.498926222393] |
| **3** | occ_4 | [0.25064662018] |
| **4** | occ_5 | [0.838982982602] |
| **5** | occ_6 | [0.833921262629] |
| **6** | occ_husb_2 | [0.190546828287] |
| **7** | occ_husb_3 | [0.297744578502] |
| **8** | occ_husb_4 | [0.161319424129] |
| **9** | occ_husb_5 | [0.187683007035] |
| **10** | occ_husb_6 | [0.193916860892] |
| **11** | rate_marriage | [-0.70312052711] |
| **12** | age | [-0.0584177439191] |
| **13** | yrs_married | [0.10567679013] |
| **14** | children | [0.0169195866351] |
| **15** | religious | [-0.371135218074] |
| **16** | educ | [0.0040161519299] |

Increases in marriage rating and religiousness correspond to a decrease in the likelihood of having an affair. For both the wife's occupation and the husband's occupation, the lowest likelihood of having an affair corresponds to the baseline occupation (student), since all of the dummy coefficients are positive.

# Model Evaluation Using a Validation Set

So far, we have trained and tested on the same set. Let's instead split the data into a training set and a testing set.

In [16]:
```
# evaluate the model by splitting into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0
model2 = LogisticRegression()
model2.fit(X_train, y_train)
```

Out[16]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, penalty='l2', random_state=None, tol=0.0001)
```

We now need to predict class labels for the test set. We will also generate the class probabilities, just to take a look.

In [17]:
```
# predict class labels for the test set
predicted = model2.predict(X_test)
print predicted
```

```
[ 1.  0.  0. ...,  0.  0.  0.]
```

In [18]:
```
# generate class probabilities
probs = model2.predict_proba(X_test)
print probs
```

```
[[ 0.3514255   0.6485745 ]
 [ 0.90952541  0.09047459]
 [ 0.72576645  0.27423355]
 ...,
 [ 0.55736908  0.44263092]
 [ 0.81213879  0.18786121]
 [ 0.74729574  0.25270426]]
```

As you can see, the classifier is predicting a 1 (having an affair) any time the probability in the second column is greater than 0.5.

Now let's generate some evaluation metrics.

In [19]:
```
# generate evaluation metrics
print metrics.accuracy_score(y_test, predicted)
print metrics.roc_auc_score(y_test, probs[:, 1])
```

```
0.729842931937
0.74596198609
```

The accuracy is 73%, which is the same as we experienced when training and predicting on the same data.

We can also see the confusion matrix and a classification report with other metrics.

In [20]:
```
print metrics.confusion_matrix(y_test, predicted)
print metrics.classification_report(y_test, predicted)
```

```
[[1169  134]
 [ 382  225]]
             precision    recall  f1-score   support

        0.0       0.75      0.90      0.82      1303
        1.0       0.63      0.37      0.47       607

avg / total       0.71      0.73      0.71      1910
```

# Model Evaluation Using Cross-Validation

Now let's try 10-fold cross-validation, to see if the accuracy holds up more rigorously.

In [21]:
```
# evaluate the model using 10-fold cross-validation
scores = cross_val_score(LogisticRegression(), X, y, scoring='accuracy', cv=10)
print scores
print scores.mean()
```

```
[ 0.72100313  0.70219436  0.73824451  0.70597484  0.70597484  0.72955975
  0.7327044   0.70440252  0.75157233  0.75      ]
0.724163068551
```

Looks good. It's still performing at 73% accuracy.

## Predicting the Probability of an Affair

Just for fun, let's predict the probability of an affair for a random woman not present in the dataset. She's a 25-year-old teacher who graduated college, has been married for 3 years, has 1 child, rates herself as strongly religious, rates her marriage as fair, and her husband is a farmer.

In [22]:
```
model.predict_proba(np.array([1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 3, 25, 3, 1, 4,
                              16]))
```

Out[22]:
```
array([[ 0.77472334,  0.22527666]])
```

The predicted probability of an affair is 23%.

## Next Steps

There are many different steps that could be tried in order to improve the model:

- including interaction terms
- removing features
- regularization techniques
- using a non-linear model