```
import statsmodels.api as sm
import pandas as pd
from patsy import dmatrices
import matplotlib.pyplot as plt
from statsmodels.sandbox.regression.predstd import wls_prediction_std

%matplotlib inline

#original version of the combined turnstile and weather data
input_filename = "/Users/rclement2/Projects/workspace/udacity/data_science1/turnstile_data_master_with_weather.csv"
df = pd.read_csv(input_filename)
```

```
def OLS_sub(df):
    '''
    Ordinary least squares (OLS) or linear least squares is a method for estimating the unknown
    parameters in a linear regression

    Calculations using different features in the OLS equation
    Features are listed in the dmatrices input below
    '''

    #y, X = dmatrices('ENTRIESn_hourly ~ Hour + rain + UNIT', data=df, return_type='dataframe') # base

    #y, X = dmatrices('ENTRIESn_hourly ~ Hour + rain + fog + UNIT', data=df, return_type='dataframe') # 1

    #y, X = dmatrices('ENTRIESn_hourly ~ Hour + rain + mintempi + maxtempi + UNIT', \
    #                 data=df, return_type='dataframe') # 2

    #y, X = dmatrices('ENTRIESn_hourly ~ Hour + rain + mintempi + maxtempi + meanpressurei + meanwindspdi + UNIT', \
    #                 data=df, return_type='dataframe') # 3

    #y, X = dmatrices('ENTRIESn_hourly ~ Hour + rain + EXITSn_hourly + UNIT', \
    #                 data=df, return_type='dataframe') # 4


    #df = df.set_index([pd.to_datetime(df['DATEn'] + ' ' + df['TIMEn'])])
    #df['dayofweek'] = df.index.weekday
    #y, X = dmatrices('ENTRIESn_hourly ~ Hour + rain + dayofweek + UNIT', \
    #                 data=df, return_type='dataframe') # 5

    # This is the option went with for the last R^2 calculation. It seemed to be reasonable in terms of the
    # info you may have to make predictions of the number of entries.
    y, X = dmatrices('ENTRIESn_hourly ~ Hour + rain + DATEn + UNIT', \
                     data=df, return_type='dataframe') # 6
    #print y[:10]
    #print X[:10]

    mod = sm.OLS(y, X)  # describe the model
    res = mod.fit()  # fit model

    return y, X, res
```

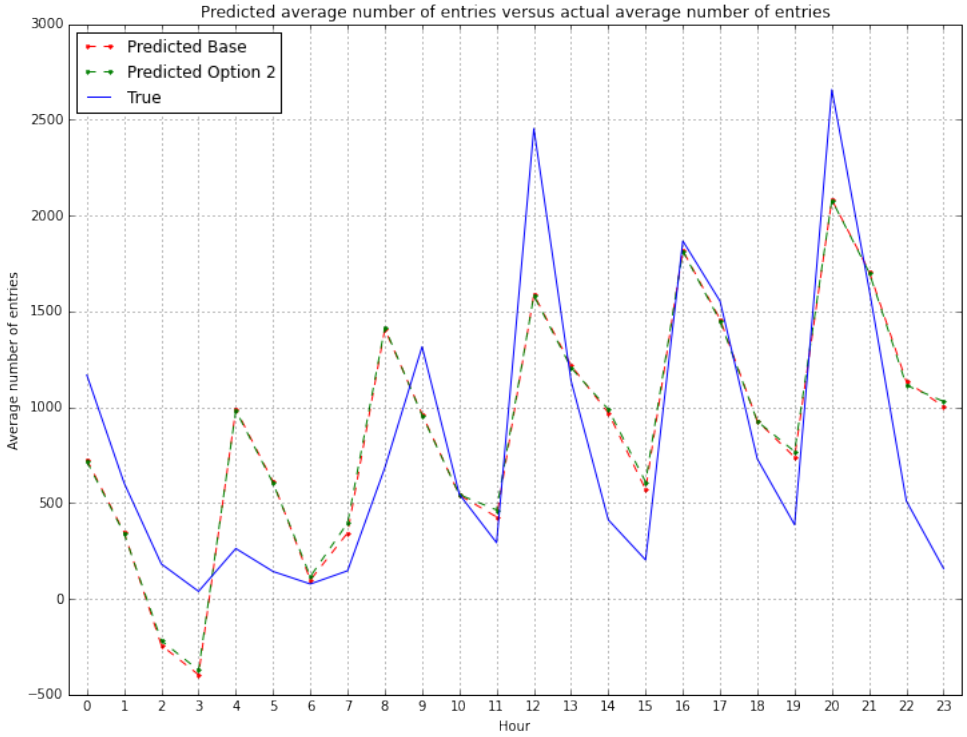
```
y, X, res = OLS_sub(df)

#print res.summary()  # summarize model
#print('Parameters: ', res.params) # parameters
print('R2: ', res.rsquared) # R^2 calculated
#print('Standard errors: ', res.bse) # Standard error
#print('Predicted values: ', res.predict()) # Predicted values
print('Predicted values: ', (res.predict()).size) # Number of predicted values for entries by hour
print('Actual values: ', len(y.index)) # Number of actual values for entries by hour
```

```
('R2: ', 0.4713825035908572)
('Predicted values: ', 131951)
('Actual values: ', 131951)
```

Method	R^2	Feature
OLS	0.458	Hour + rain + UNIT
OLS	0.458	Hour + rain + fog + UNIT
OLS	0.459	Hour + rain + mintempi + maxtempi + UNIT
OLS	0.459	Hour + rain + mintempi + maxtempi + meanpressurei + meanwindspdi + UNIT
OLS	0.621	Hour + rain + EXITs_hourly + UNIT
OLS	0.463	Hour + rain + dayofweek + UNIT
OLS	0.471	Hour + rain + DATEn + UNIT

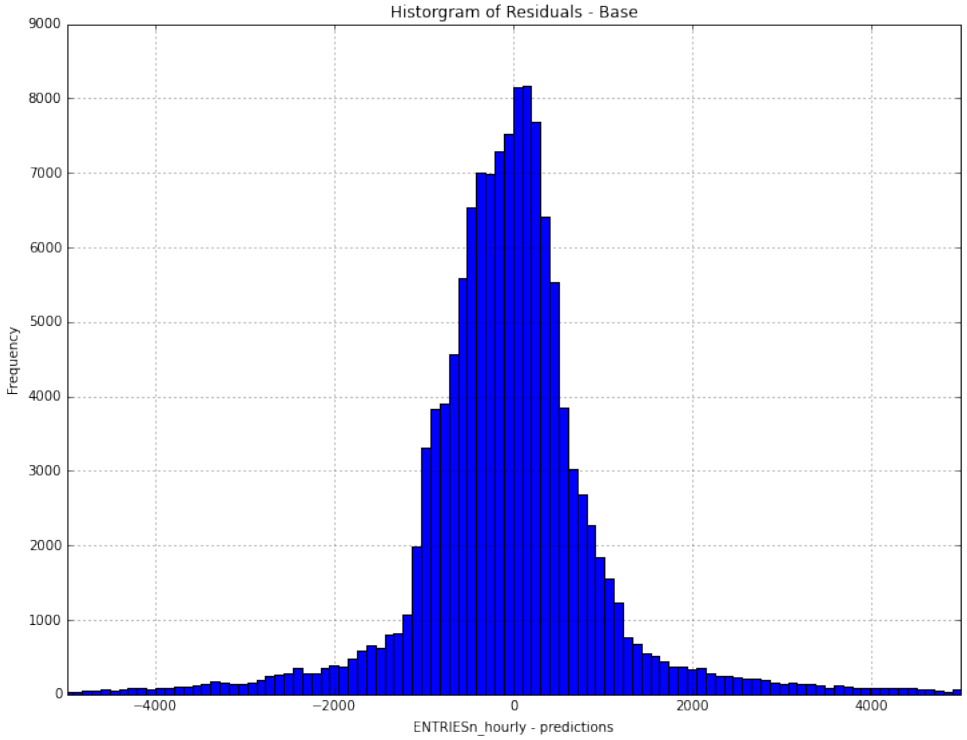
```
'''
Using OLS take a look at the effect of including various features and see what the impact is on R^2.
Since there are in excess of 130,000 values the mean number of entries per hour (actual and predicted)
are then plotted using a line plot to see what the correlation is
'''

#prep the features
y1, X1 = dmatrices('ENTRIESn_hourly ~ Hour + rain + UNIT', data=df, return_type='dataframe') # base
y2, X2 = dmatrices('ENTRIESn_hourly ~ Hour + rain + DATEn + UNIT', data=df, return_type='dataframe') # 6

#use OLS
mod1 = sm.OLS(y1, X1)  # describe the model 1
mod2 = sm.OLS(y2, X2)  # describe the model 2
res1 = mod1.fit()  # fit model 1
res2 = mod2.fit()  # fit model 2

#results
print('Model 1 R2: ', res1.rsquared)
print('Model 2 R2: ', res2.rsquared)
print '\n'

#get data ready for plotting
df['fitted1'] = res1.fittedvalues
df['fitted2'] = res2.fittedvalues
grouped = df.groupby(['Hour'])['ENTRIESn_hourly', 'fitted1', 'fitted2'].mean()

#setup the plots
fig, ax = plt.subplots(figsize=(12,9))

ax.plot(grouped.index, grouped.fitted1, 'r--.', label="Predicted Base")
ax.plot(grouped.index, grouped.fitted2, 'g--.', label="Predicted Option 2")
ax.plot(grouped.index, grouped['ENTRIESn_hourly'], 'b-', label="True")

ax.set_xlabel('Hour')
ax.set_ylabel('Average number of entries')
ax.set_title('Predicted average number of entries versus actual average number of entries')
start, end = ax.get_xlim()
ax.set_xlim(-.5, end-1.5)
plt.xticks(range(0, int(end-1), 1))
plt.grid(True)
ax.legend(loc='best')
plt.show()
```

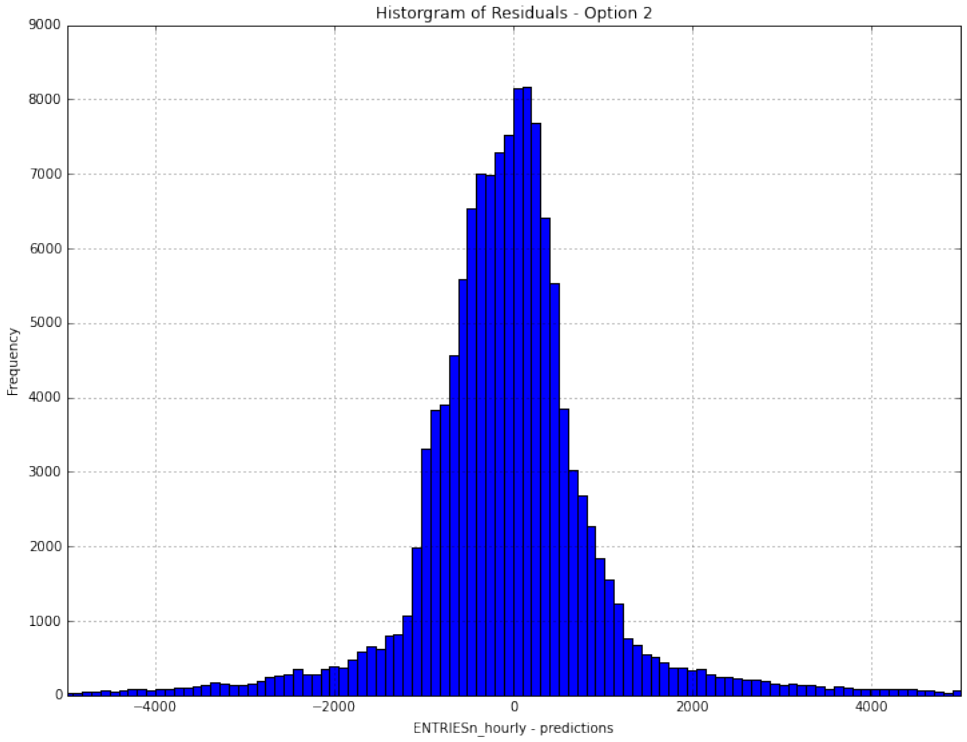
```
('Model 1 R2: ', 0.4575613638472098)
('Model 2 R2: ', 0.4713825035908572)
```




```
def plot_residuals(df, predictions, title):
    '''
    Using the same methods that we used to plot a histogram of entries
    per hour for our data, we can also make a histogram of the residuals.
    (that is, the difference between the original hourly entry data and the predicted values).

    Reading a bit on this webpage might be useful: http://www.itl.nist.gov/div898/handbook/pri/section2/pri24.htm
    '''

    #setup the plot
    plt.figure(figsize=(12,9))
    plt.grid(True)
    axHist = plt.axes()

    # the histogram plot:
    axHist.hist((df['ENTRIESn_hourly'] - res.fittedvalues), bins=500)
    Start, End = axHist.get_ylim()
    axHist.set_ylim(0, End)
    axHist.set_xlabel('ENTRIESn_hourly - predictions')
    axHist.set_ylabel('Frequency')
    axHist.set_title('Historgram of Residuals - ' + title)
    axHist.set_xlim( (-5000, 5000) )
    plt.show()
```

```
plot_residuals(df, res1, 'Base') #base plot of residuals using Model 1 ('Model 1 R2: ', 0.4575613638472098)
plot_residuals(df, res2, 'Option 2') #option 2 from Model 2 which was adding day of the week ('Model 2 R2: ', 0.4713825035
```


Historgram of Residuals - Base
Frequency
ENTRIESn_hourly - predictions
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
−4000
−2000
0
2000
4000


Historgram of Residuals - Option 2
Frequency
ENTRIESn_hourly - predictions
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
−4000
−2000
0
2000
4000