Data Dive Week 5: Regression in Python

Note that this notebook borrows heavily from the online resources (https://github.com/cs109/2015lab4) for CS109 at Harvard University (http://cs109.github.io/2015/pages/videos.html).

This week we take a look at some basic statistical concepts, with a particular focus on regression models. As we covered in the lecture portion() of this week’s class, linear regression is used to model and predict continuous outcomes. Time permitting, we'll also discuss logistic regression, which is used to model binary outcomes.

Though the DataCamp course covered for homework used the numpy package for linear regression, we'll also touch upon statsmodels and scikit-learn in today's exercise.

```python
In [1]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
%matplotlib inline
```
Purpose of linear regression

Given a dataset $X$ and $Y$, linear regression can be used to:

- Build a **predictive model** to predict future values of $X_i$ without a $Y$ value.
- Model the **strength of the relationship** between each dependent variable $X_i$ and $Y$
  - Sometimes not all $X_i$ will have a relationship with $Y$
  - Need to figure out which $X_i$ contributes most information to determine $Y$

A brief recap

[Linear Regression](http://en.wikipedia.org/wiki/Linear_regression) is a method to model the relationship between a set of independent variables $X$ (also knowns as explanatory variables, features, predictors) and a dependent variable $Y$. This method assumes the relationship between each predictor $X$ is linearly related to the dependent variable $Y$.

$$Y = \beta_0 + \beta_1X + \epsilon$$

where $\epsilon$ is considered as an unobservable random variable that adds noise to the linear relationship. This is the simplest form of linear regression (one variable), we’ll call this the simple model.

- $\beta_0$ is the intercept of the linear model
- Multiple linear regression is when you have more than one independent variable
  - $X_1, X_2, X_3, \ldots$

$$Y = \beta_0 + \beta_1X_1 + \ldots + \beta_pX_p + \epsilon$$

StreetEasy Rentals Data Set

Data from this week’s exercise comes from StreetEasy (www.streeteasy.com), an online platform for real estate listings in New York City. The sample provided covers 5,000 listings for homes for rent in Manhattan, Brooklyn, and Queens in June 2016 and provides several features of interest in modeling rents, including neighborhood, subway access, building amenities,

Note that this data is provided for instructional purposes only and is not intended to be representative of all listings on StreetEasy or all homes for rent in New York City. Any analysis conducted as part of this exercise does not reflect the opinion or endorsement of StreetEasy or any of its affiliates.

In [2]: se_df = pd.read_csv('https://grantmlong.com/data/streeteasy_rents_june2016.csv')
In [3]: se_df.head()

Out[3]:

<table>
<thead>
<tr>
<th>rental_id</th>
<th>building_id</th>
<th>rent</th>
<th>bedrooms</th>
<th>bathrooms</th>
<th>size_sqft</th>
<th>min_to_subway</th>
<th>floor</th>
<th>building_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1545</td>
<td>44518357</td>
<td>2550</td>
<td>0.0</td>
<td>1</td>
<td>480</td>
<td>9</td>
<td>2.0</td>
</tr>
<tr>
<td>1</td>
<td>2472</td>
<td>94441623</td>
<td>11500</td>
<td>2.0</td>
<td>2</td>
<td>2000</td>
<td>4</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>10234</td>
<td>87632265</td>
<td>3000</td>
<td>3.0</td>
<td>1</td>
<td>1000</td>
<td>4</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>2919</td>
<td>76909719</td>
<td>4500</td>
<td>1.0</td>
<td>1</td>
<td>916</td>
<td>2</td>
<td>51.0</td>
</tr>
<tr>
<td>4</td>
<td>2790</td>
<td>92953520</td>
<td>4795</td>
<td>1.0</td>
<td>1</td>
<td>975</td>
<td>3</td>
<td>8.0</td>
</tr>
</tbody>
</table>

**Data Exploration**

**Summarize and Plot a Histogram for the Target Variable**

Is there anything surprising or interesting about this data?

In [4]: _ = se_df.rent.hist(bins=25)
   _ = plt.xlabel('Monthly Rent')
   _ = plt.ylabel('Count')
Feature Exploration

*If we wanted to try to create a model to price any given apartment, what variables might be the most important?*

- How many variables are at our disposal?
- Which are binary? Categorical? Continuous?
- Which variable make most sense to use from an intuitive standpoint?
- Identify which variable has the highest correlation with
In [6]:
print(list(se_df))
print()
se_df.describe()

['\xef\xbfb\xfbrental_id', 'building_id', 'rent', 'bedrooms', 'bathrooms', 'size_sqft', 'min_to_subway', 'floor', 'building_age_yrs', 'no_fee', 'has_roofdeck', 'has_washer_dryer', 'has_doorman', 'has_elevator', 'has_dishwasher', 'has_patio', 'has_gym', 'neighborhood', 'submarket', 'borough']

Out[6]:

<table>
<thead>
<tr>
<th></th>
<th>rental_id</th>
<th>building_id</th>
<th>rent</th>
<th>bedrooms</th>
<th>bathrooms</th>
<th>size_sqft</th>
<th>min_to_subway</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>5000.000000</td>
<td>5.000000e+03</td>
<td>5000.000000</td>
<td>5000.000000</td>
<td>5000.000000</td>
<td>5000.000000</td>
<td>500</td>
</tr>
<tr>
<td>mean</td>
<td>5526.909400</td>
<td>5.122007e+07</td>
<td>4536.920800</td>
<td>1.395700</td>
<td>1.321600</td>
<td>920.101464</td>
<td>920.101400</td>
</tr>
<tr>
<td>std</td>
<td>3263.692417</td>
<td>2.802283e+07</td>
<td>2929.838953</td>
<td>0.961018</td>
<td>0.565542</td>
<td>440.150464</td>
<td>440.150400</td>
</tr>
<tr>
<td>min</td>
<td>1.000000</td>
<td>7.107000e+03</td>
<td>1250.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>250.000000</td>
<td>250.000000</td>
</tr>
<tr>
<td>25%</td>
<td>2699.750000</td>
<td>2.699811e+07</td>
<td>2750.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>633.000000</td>
<td>633.000000</td>
</tr>
<tr>
<td>50%</td>
<td>5456.500000</td>
<td>5.069894e+07</td>
<td>3600.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>800.000000</td>
<td>800.000000</td>
</tr>
<tr>
<td>75%</td>
<td>8306.000000</td>
<td>7.572064e+07</td>
<td>5200.000000</td>
<td>2.000000</td>
<td>2.000000</td>
<td>1094.000000</td>
<td>1094.000000</td>
</tr>
<tr>
<td>max</td>
<td>11349.000000</td>
<td>9.998721e+07</td>
<td>20000.000000</td>
<td>5.000000</td>
<td>5.000000</td>
<td>4800.000000</td>
<td>4800.000000</td>
</tr>
</tbody>
</table>

In [7]:
se_df.corr()]['rent'].sort_values(ascending=False)[1:]

Out[7]:
size_sqft                       0.808784
bathrooms                       0.733644
bedrooms                        0.531200
floor                           0.272351
has_elevator                    0.120822
has_doorman                      0.099106
has_dishwasher                   0.096631
has_washer_dryer                 0.096136
has_gym                          0.093737
has_roofdeck                     0.081286
has_patio                        0.058260
min_to_subway                    0.003652
building_id                      -0.001235
no_fee                           -0.091769
building_age_yrs                 -0.122302
rental_id                        -0.142801
Name: rent, dtype: float64
Scatterplots

- Create a scatterplot of `size_sqft`, `bathrooms`, and `floor`.
- Describe the relationship you see? Is it positive or negative? Linear? Non-linear?

In [8]: `_ = se_df.plot.scatter('size_sqft', 'rent')`

![Scatterplot of size_sqft vs rent](image1)

In [9]: `_ = se_df.plot.scatter('bathrooms', 'rent')`

![Scatterplot of bathrooms vs rent](image2)
In [10]: _ = se_df.plot.scatter('floor', 'rent')

In [ ]:

**Modeling**

**Single Variable Linear Regression with size_sqft**

- Use `numpy` to fit a simple linear regression
  - Calculate the slope and intercept using the `polyfit` function
  - Print the slope and intercept. How would you interpret these two numbers?
  - Based on this data, how much would you expect a 700 square foot apartment to cost?

In [11]: beta, alpha = np.polyfit(se_df.size_sqft, se_df.rent, 1)
print('beta: %0.3f, alpha: %0.1f.' % (beta, alpha))
print()
beta: 5.384, alpha: -416.6.
()

In [12]: print('Based on this analysis a 700 square foot apartment would rent for $%0.2f'
   % (beta * 700 + alpha))
   
   Based on this analysis a 700 square foot apartment would rent for $335 1.98
Visualize the relationship.

- Plot the fitted line along with the scatter plot.
- Is this line a good fit?

In [13]:

```python
_ = plt.plot(se_df.size_sqft, se_df.rent, color='blue', marker='.', markersize=12, linestyle='none')
_ = plt.plot([0, 4000], [alpha, alpha + beta * 4000], color='red', linewidth=4)
_ = plt.title('')
```

Calculate the predicted rent and residual for each observation.

- Create columns in the se_df dataframe for rent_predicted and rent_residual
- Does this appear to fall in line with the assumptions we've described?

In [14]:

```python
se_df['rent_predicted'] = se_df['rent'] * beta + alpha
se_df['rent_residual'] = se_df['rent_predicted'] - se_df['rent']
```
Using `statsmodels` for Single Variable Linear Regression

- Use `statsmodels` to fit a simple linear regression with `size_sqft`.
  - Output the regression results.
  - Describe how this output compares to our $\alpha$ and $\beta$ from `numpy`. 

In [15]: `se_df['rent_residual'].hist(bins=20)`

Out[15]: `<matplotlib.axes._subplots.AxesSubplot at 0x11c287610>`
# Add a constant to our existing dataframe for modeling purposes
se_df = sm.add_constant(se_df)

est = sm.OLS(se_df['rent'],
             se_df[['const', 'size_sqft']]).fit()

print(est.summary())
OLS Regression Results

=======================================================================

Dep. Variable: rent R-squared: 0.654
Model: OLS Adj. R-squared: 0.654
Method: Least Squares F-statistic: 9453.
Date: Wed, 03 Oct 2018 Prob (F-statistic): 0.00
Time: 00:37:11 Log-Likelihood: -44353.
No. Observations: 5000 AIC: 8.871e+04
Df Residuals: 4998 BIC: 8.872e+04
Df Model: 1
Covariance Type: nonrobust

=======================================================================

 coef std err t P>|t| [95.0% Conf. Int.]
--------- ------- ------ ----- ------ ---------------------
const -416.5609 56.478 -7.376 0.000 -527.282 -305.840
size_sqft 5.3836 0.055 97.224 0.000 5.275 5.492

=======================================================================

Omnibus: 870.478 Durbin-Watson: 2.006
Prob(Omnibus): 0.000 Jarque-Bera (JB): 7186.761
Skew: 0.594 Prob(JB):
Kurtosis: 8.752 Cond. No. 2.36e+03

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.36e+03. This might indicate that there are strong multicollinearity or other numerical problems.
Using *statsmodels* for Multiple Linear Regression

- Still using *statsmodels*, add some variables to our existing regression. Can you get a better prediction?
  - Add a one or two variables at a time. What happens to our $R^2$?
  - Which variables are most significant? How does this change as we add more predictors?
  - With regression with many predictors, create a histogram of the residuals. How does this compare to the single variable case?
    - *Note: use can access the residuals using the* `est.resid` *attribute of the regression results.*
est = sm.OLS(se_df['rent'],
          se_df[['const', 'size_sqft', 'bathrooms', 'floor', 'has_door']]
        ).fit()

print(est.summary())
OLS Regression Results
=======================================================================
            dep. variable: rent          R-squared: 0.716
            model: OLS          Adj. R-squared: 0.716
            method: Least Squares  F-statistic: 3151.
            date: Wed, 03 Oct 2018  Prob (F-statistic): 0.00
            time: 00:37:11         log-likelihood: -43859.
            no. observations: 5000  AIC: 8.773e+04
            df residuals: 4995     BIC: 8.776e+04
            df model: 4
            covariance type: nonrobust
=======================================================================

                          coef  std err     t     P>|t|    [95.0% Conf. Int.]
-----------------------------------------------------------------------
const                -1362.8677  59.283    -22.99    0.000   -1479.089 -1246.647
size_sqft             4.0150   0.079    50.975    0.000     3.861    4.169
bathrooms            1254.2537 61.633    20.350    0.000   1133.427 1375.081
floor                 47.6704   2.128    22.400    0.000    43.498    51.843
has_doorman           272.7467  53.074     5.139    0.000   168.699  376.794
-----------------------------------------------------------------------

 Omnibus: 1234.155 Durbin-Watson: 2.009
Prob(Omnibus): 0.000 Jarque-Bera (JB): 9467.982
Skew: 0.968 Prob(JB): 0.00
Kurtosis: 9.458 Cond. No. 3.31e+03

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.31e+03. This might indicate that there are strong multicollinearity or other numerical problems.
In [18]:
est = sm.OLS(se_df['rent'],
            se_df.drop(['neighborhood', 'submarket', 'borough', 'rent',
                        'rent_predicted', 'rent_residual'], axis=1))

            .fit()

print(est.summary())
OLS Regression Results

=======================================================================
=======
Dep. Variable: rent R-squared: 0.734
Model: OLS Adj. R-squared: 0.733
Method: Least Squares F-statistic: 858.1
Date: Wed, 03 Oct 2018 Prob (F-statistic): 0.00
Time: 00:37:11 Log-Likelihood: -43700.
No. Observations: 5000 AIC: 8.743e+04
Df Residuals: 4983 BIC: 8.754e+04
Df Model: 16
Covariance Type: nonrobust
=======================================================================

 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const             -777.1793    100.937     -7.700      0.000      -975.059  -579.299
rental_id        -0.0396      0.007     -5.949      0.000        -0.053  -0.027
building_id       -1.79e-07   7.68e-07     -0.233      0.816     -1.68e-06  1.33e-06
bedrooms          -448.8758     34.356    -13.066      0.000      -516.228  -381.524
bathrooms         1337.9998     62.256     21.492      0.000     1215.951  1460.048
size_sqft            4.6522      0.089     52.028      0.000         4.477  4.828
min_to_subway      -16.5954      4.129     -4.019      0.000       -24.690  -8.501
floor               38.1177      2.218     17.185      0.000        33.769  42.466
building_age_yrs    -4.2090      0.597     -7.047      0.000       -5.380  -3.038
no_fee             -77.5812     45.713     -1.697      0.090      -167.198  12.036
has_roofdeck        43.9167     82.388      0.533      0.594      -117.959  205.433
has_washer_dryer   149.3765     74.678      2.000      0.046         2.974  295.779
has_dishwasher      40.3748     71.617      0.564      0.573      -100.025  180.775
has_patio           -32.7547    105.289     -0.311      0.756      -239.167  173.657
Using `sklearn` for Multiple Linear Regression

`sklearn` is among the most popular packages for machine learning, and it's one we'll be using throughout the rest of the semester. It’s syntax and functionality is a little different, but it gives us a little more flexibility around accessing and using the output, and also plays nice with modeling options beyond linear regression.
What can you do with a LinearRegression object?

Main functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lm.fit()</td>
<td>Fit a linear model</td>
</tr>
<tr>
<td>lm.predict()</td>
<td>Predict Y using the linear model with estimated coefficients</td>
</tr>
<tr>
<td>lm.score()</td>
<td>Returns the coefficient of determination ($R^2$). A measure of how well observed outcomes are replicated by the model, as the proportion of total variation of outcomes explained by the model</td>
</tr>
</tbody>
</table>

What output can you get?

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lm.coef_</td>
<td>Estimated coefficients</td>
</tr>
<tr>
<td>lm.intercept_</td>
<td>Estimated intercept</td>
</tr>
</tbody>
</table>

Fit a linear model

The `lm.fit()` function estimates the coefficients the linear regression using least squares.
# Use sensible subset of predictors to fit linear regression model

dependent_vars = ['bedrooms', 'bathrooms', 'min_to_subway', 'floor', 'building_age_yrs', 'no_fee', 'has_roofdeck', 'has_washer_dryer', 'has_doorman', 'has_elevator', 'has_dishwasher', 'has_patio', 'has_gym']

X = se_df[dependent_vars]

lm.fit(X, se_df.rent)

# notice fit_intercept=True and normalize=True
# How would you change the model to not fit an intercept term?

Out[23]:

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

**Estimated intercept and coefficients**

Let's look at the estimated coefficients from the linear model using `lm.intercept_` and `lm.coef_`.

After we have fit our linear regression model using the least squares method, we want to see what are the estimates of our coefficients $\beta_0, \beta_1, ..., \beta_{13}$:

$$\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_{13}$$

In [24]:

print('Estimated intercept coefficient:', lm.intercept_)
print('Number of coefficients:', len(lm.coef_))

('Estimated intercept coefficient:', -515.50403608104898)
('Number of coefficients:', 13)
# The coefficients

```
pd.DataFrame(lm.coef_, index=dependent_vars, columns=['Est. Coefficient'])
```

<table>
<thead>
<tr>
<th></th>
<th>Est. Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>bedrooms</td>
<td>417.730687</td>
</tr>
<tr>
<td>bathrooms</td>
<td>3171.816130</td>
</tr>
<tr>
<td>min_to_subway</td>
<td>-22.630204</td>
</tr>
<tr>
<td>floor</td>
<td>49.132507</td>
</tr>
<tr>
<td>building_age_yrs</td>
<td>-1.488370</td>
</tr>
<tr>
<td>no_fee</td>
<td>-344.120712</td>
</tr>
<tr>
<td>has_roofdeck</td>
<td>-4.475219</td>
</tr>
<tr>
<td>has_washer_dryer</td>
<td>191.475476</td>
</tr>
<tr>
<td>has_doorman</td>
<td>-41.004582</td>
</tr>
<tr>
<td>has_elevator</td>
<td>400.797299</td>
</tr>
<tr>
<td>has_dishwasher</td>
<td>147.357561</td>
</tr>
<tr>
<td>has_patio</td>
<td>-83.967351</td>
</tr>
<tr>
<td>has_gym</td>
<td>-96.097033</td>
</tr>
</tbody>
</table>

## Predict Prices

We can calculate the predicted prices ($\hat{Y}_i$) using `lm.predict`.

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_1 + ... + \hat{\beta}_{13} X_{13}$$

```
# first five predicted prices
lm.predict(X)[0:5]
```

```
array([2629.06488344, 6479.31777692, 3710.34863476, 5994.05257186, 3764.12600492])
```
Let's plot the true prices compared to the predicted prices to see they disagree, we saw this exactly before but this is how you access the predicted values in using \texttt{sklearn}.

```python
In [28]: _ = plt.scatter(se_df['rent'], lm.predict(X))
   _ = plt.xlabel("Rents: $Y_i$")
   _ = plt.ylabel("Predicted rents: $\hat{Y}_i$")
   _ = plt.title("Rents vs Predicted Rents: $Y_i$ vs $\hat{Y}_i$")
   _ = plt.plot([0, 20000], [0, 20000], linewidth=4, color='red')
```
Residual sum of squares

Let’s calculate the residual sum of squares

\[ S = \sum_{i=1}^{N} r_i = \sum_{i=1}^{N} (y_i - (\beta_0 + \beta_1 x_i))^2 \]

```python
In [29]: print('%0.2f' % np.sum((se_df['rent'] - lm.predict(X)) ** 2))
17730479762.04
```

Bonus Round: Feature Engineering

Our original data set featured information on borough, submarket, neighborhood - all different ways of slices up the city in geographic terms.

- To what extent do you think models will return different results across different boroughs?
- How might you include some or all of these geographic areas in the model?