17-Regression-I-Linear

November 8, 2016

1 Linear Models

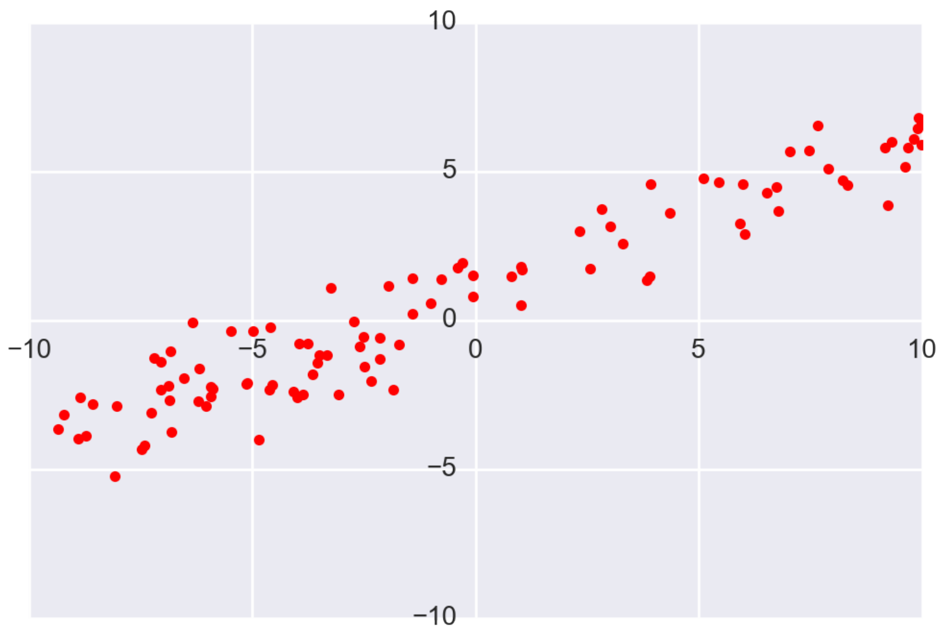
In 1886 Francis Galton published his observations about how random factors affect outliers.

This notion has come to be called "regression to the mean" because unusually large or small phenomena, after the influence of random events, become closer to their mean values (less extreme).

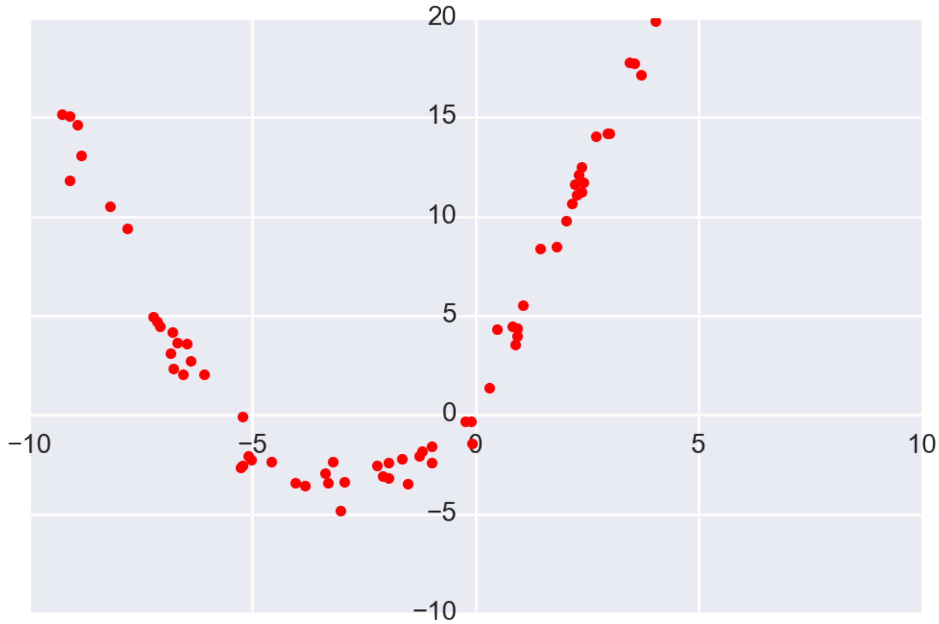
The most common form of machine learning is **regression**, which means constructing an equation that describes the relationships among variables.

It is a form of supervised learning: whereas **classification** deals with predicting categorical features (labels or classes), **regression** deals with predicting continuous features (real values).

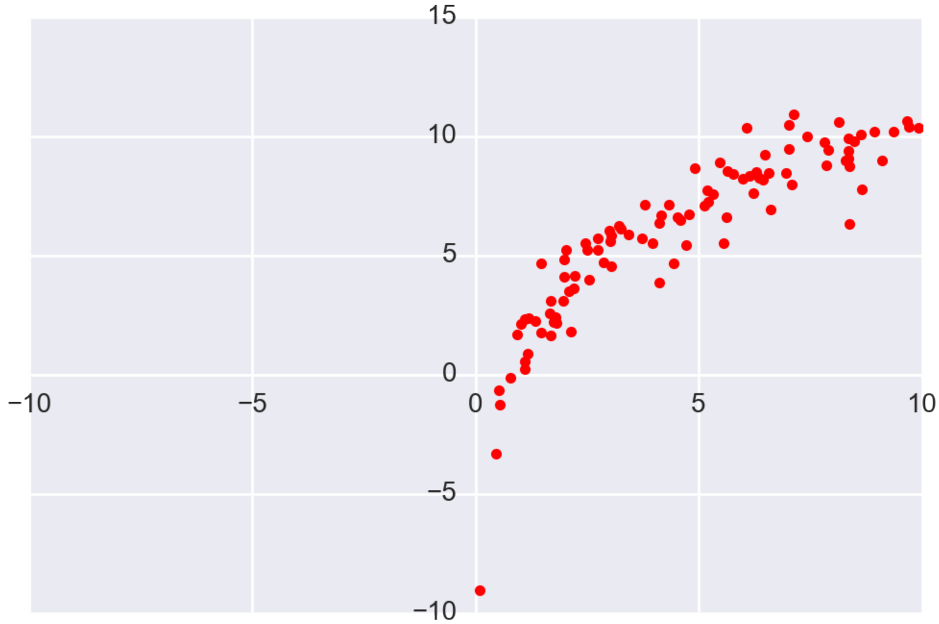
For example, we may look at these points and decide to model them using a line.



We may look at these points and decide to model them using a quadratic function.



And we may look at these points and decide to model them using a logarithmic function.



Clearly, none of these datasets agrees perfectly with the proposed model. So the question arises:

How do we find the **best** linear function (or quadratic function, or logarithmic function) given the data?

Framework.

This problem has been studied extensively in the field of statistics. Certain terminology is used:

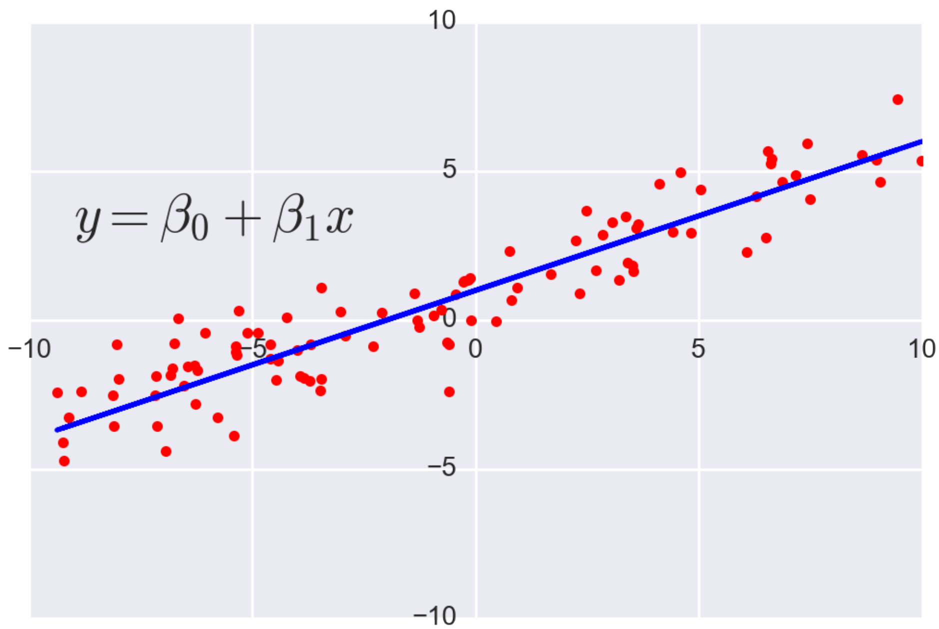
- Some values are referred to as "independent," and
- Some values are referred to as "dependent."

The basic regression task is: given a set of independent variables and the associated dependent variables, estimate the parameters of a model (such as a line, parabola, etc) that describes how the dependent variables are related to the independent variables.

The dependent variables are collected into a matrix X, which is called the **design matrix.**

The independent variables are collected into an **observation** vector $\mathbf{y}$.

The parameters of the model (for any kind of model) are collected into a **parameter** vector β.



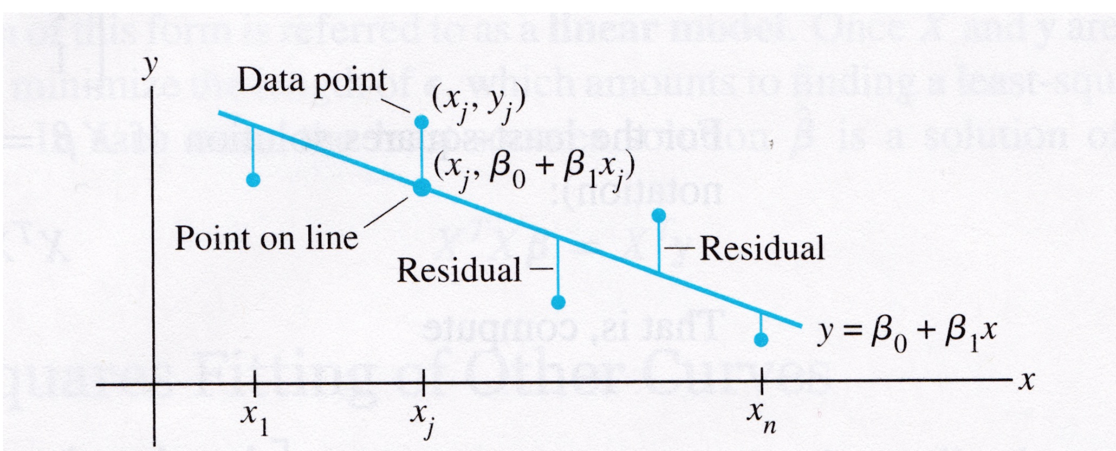
1.1 Least-Squares Lines

The first kind of model we'll study is a linear equation, $y = \beta_0 + \beta_1 x$.

Experimental data often produce points $(x_1, y_1), \ldots, (x_n, y_n)$ that seem to lie close to a line.

We want to determine the parameters β_0, β_1 that define a line that is as "close" to the points as possible.

Suppose we have a line $y = \beta_0 + \beta_1 x$. For each data point (x_j, y_j), there is a point $(x_j, \beta_0 + \beta_1 x_j)$ that is the point on the line with the same x-coordinate.



We call y_j the **observed** value of y and $\beta_0 + \beta_1 x_j$ the **predicted** y-value.

The difference between an observed y-value and a predicted y-value is called a **residual**.

There are several ways of measure how "close" the line is to the data.

The usual choice is to sum the squares of the residuals.

The **least-squares line** is the line $y = \beta_0 + \beta_1 x$ that minimizes the sum of squares of the residuals.

The coefficients β_0, β_1 of the line are called **regression coefficients.**

A least-squares problem.

If the data points were on the line, the parameters β_0 and β_1 would satisfy the equations

$$\begin{aligned}
\beta_0 + \beta_1 x_1 &= y_1 \\
\beta_0 + \beta_1 x_2 &= y_2 \\
\beta_0 + \beta_1 x_3 &= y_3 \\
&\vdots \\
\beta_0 + \beta_1 x_n &= y_n
\end{aligned}$$

We can write this system as

$$X\beta = \mathbf{y}, \quad \text{where } X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Of course, if the data points don't actually lie exactly on a line,

... then there are no parameters β_0, β_1 for which the predicted y-values in $X\beta$ equal the observed y-values in $\mathbf{y}$,

... and $X\beta = \mathbf{y}$ has no solution.

Note that we are seeking the β that minimizes the sum of squared residuals, ie,

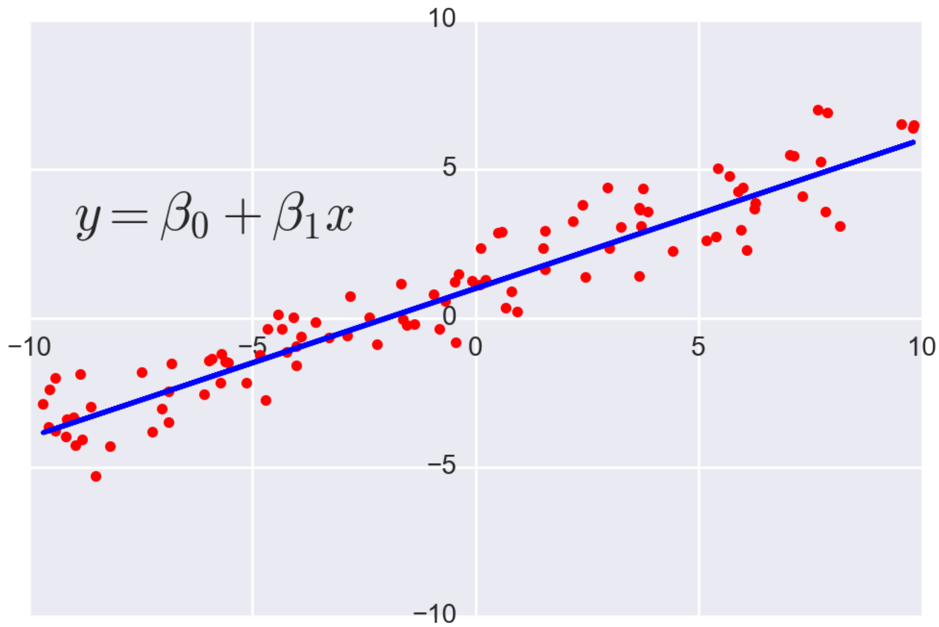
$$\sum_i (\beta_0 + \beta_1 x_i - y_i)^2$$

$$= \|X\beta - \mathbf{y}\|^2$$

This is key: **the sum of squares of the residuals** is **exactly** the **square of the distance between the vectors $X\beta$ and $\mathbf{y}$**.

This is a least-squares problem, $A\mathbf{x} = \mathbf{b}$, with different notation.

Computing the least-squares solution of $X\beta = \mathbf{y}$ is equivalent to finding the β that determines the least-squares line.



Now, to obtain the least-squares line, find the least-squares solution to $X\beta = \mathbf{y}$.

From linear algebra we know that the least squares solution of $X\beta = \mathbf{y}$ is given by the solution of the **normal equations**:

$$X^T X\beta = X^T \mathbf{y}$$

We also know that the normal equations **always** have at least one solution.

And if $X^T X$ is invertible, there is a unique solution that is given by:

$$\beta = (X^T X)^{-1} X^T \mathbf{y}$$

1.2 The General Linear Model

Another way that the inconsistent linear system is often written is to collect all the residuals into a **residual vector.**

Then an exact equation is

$$y = X\beta + \epsilon$$

Any equation of this form is referred to as a **linear model.**

In this formulation, the goal is to find the β so as to minimize the length of ϵ, ie, $\|\epsilon\|$.

In some cases, one would like to fit data points with something other than a straight line.

In cases like this, the matrix equation is still $X\beta = \mathbf{y}$, but the specific form of X changes from one problem to the next.

1.3 Least-Squares Fitting of Other Models

Most models have parameters, and the objection of **model fitting** is to to fix those parameters. Let's talk about model parameters.

In model fitting, the parameters are the unknown. A central question for us is whether the model is *linear* in its parameters.

For example, the model $y = \beta_0 e^{-\beta_1 x}$ is **not** linear in its parameters. The model $y = \beta_0 e^{-2x}$ **is** linear in its parameters.

For a model that is linear in its parameters, an observation is a linear combination of (arbitrary) known functions.

In other words, a model that is linear in its parameters is

$$y = \beta_0 f_0(x) + \beta_1 f_1(x) + \cdots + \beta_n f_n(x)$$

where $f_0, \ldots, f_n$ are known functions and $\beta_0, \ldots, \beta_k$ are parameters.

Example. Suppose data points $(x_1, y_1), \ldots, (x_n, y_n)$ appear to lie along some sort of parabola instead of a straight line. Suppose we wish to approximate the data by an equation of the form

$$y = \beta_0 + \beta_1 x + \beta_2 x^2.$$

Describe the linear model that produces a "least squares fit" of the data by the equation.

Solution. The ideal relationship is $y = \beta_0 + \beta_1 x + \beta_2 x^2$.

Suppose the actual values of the parameters are $\beta_0, \beta_1, \beta_2$. Then the coordinates of the first data point satisfy the equation

$$y_1 = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \epsilon_1$$

where ϵ_1 is the residual error between the observed value y_1 and the predicted y-value.

Each data point determines a similar equation:

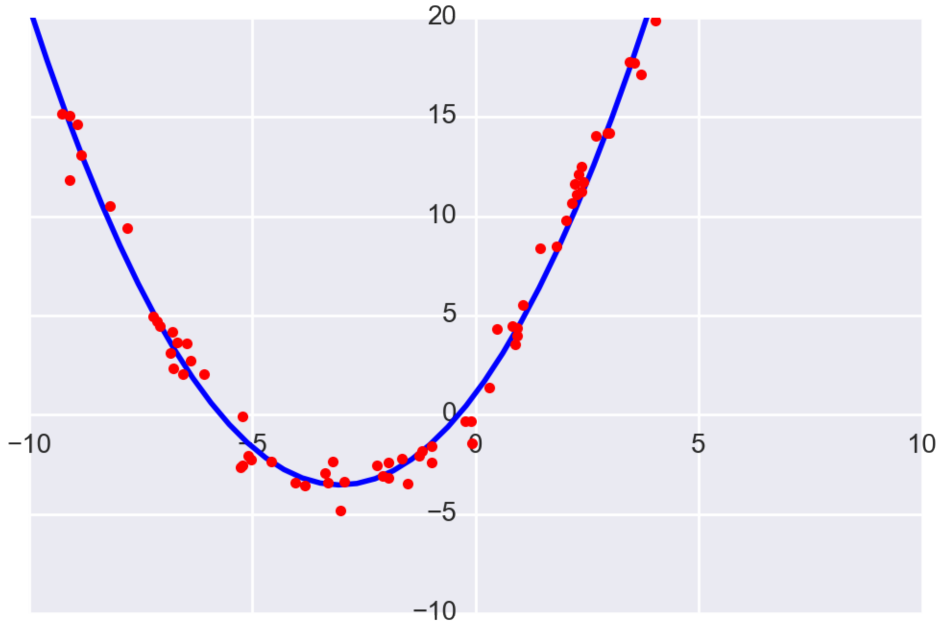
$$\begin{aligned} y_1 &= \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \epsilon_1 \\ y_2 &= \beta_0 + \beta_1 x_2 + \beta_2 x_2^2 + \epsilon_2 \\ &\vdots \\ y_n &= \beta_0 + \beta_1 x_n + \beta_2 x_n^2 + \epsilon_n \end{aligned}$$

Clearly, this system can be written as $\mathbf{y} = X\beta + \epsilon$.

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

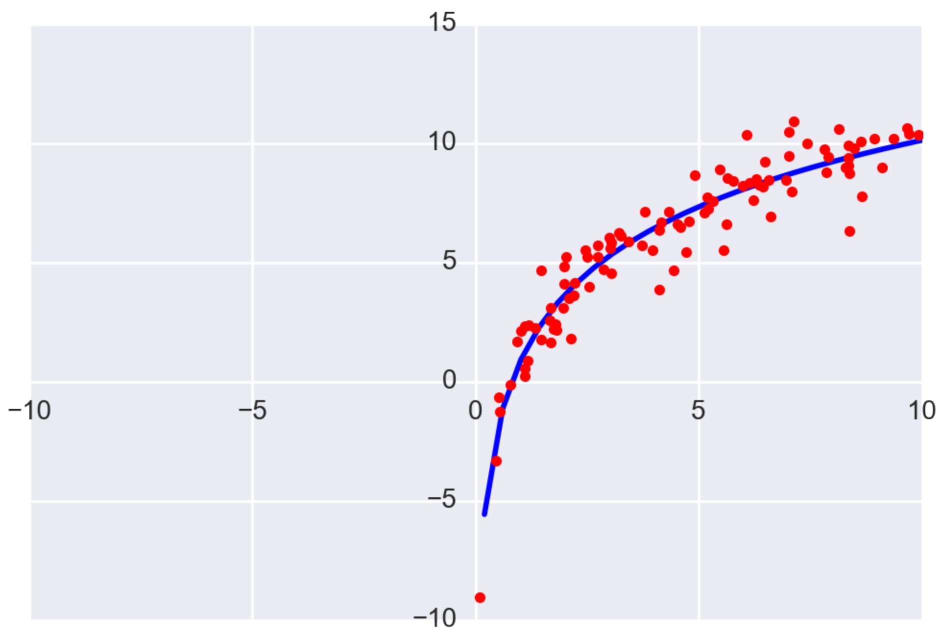
In [49]:
```
import numpy as np
import matplotlib.pyplot as plt
import laUtilities as ut
#
# Input data are in the vectors xquad and yquad
#
# estimate the parameters of the linear model
```

```
#
m = np.shape(xquad)[0]
X = np.array([np.ones(m),xquad,xquad**2]).T
beta = np.linalg.inv(X.T @ X) @ X.T @ yquad
#
# plot the results
#
ax = ut.plotSetup(-10,10,-10,20)
ut.centerAxes(ax)
xplot = np.linspace(-10,10,50)
yestplot = beta[0]+beta[1]*xplot+beta[2]*xplot**2
ax.plot(xplot,yestplot,'b-',lw=2)
ax.plot(xquad,yquad,'ro',markersize=4)
print('')
```




```
In [50]: import numpy as np
         import matplotlib.pyplot as plt
         import laUtilities as ut
         #
         # Input data are in the vectors xlog and ylog
         #
         # estimate the parameters of the linear model
         #
```

```
m = np.shape(xlog)[0]
X = np.array([np.ones(m),np.log(xlog)]).T
beta = np.linalg.inv(X.T @ X) @ X.T @ ylog
#
# plot the results
#
ax = ut.plotSetup(-10,10,-10,15)
ut.centerAxes(ax)
xplot = np.linspace(-10,10,50)
yestplot = beta[0]+beta[1]*np.log(xplot)
ax.plot(xplot,yestplot,'b-',lw=2)
ax.plot(xlog,ylog,'ro',markersize=4)
print('')
```



1.4 Multiple Regression

Suppose an experiment involves two independent variables – say, u and v, – and one dependent variable, y. A simple equation for predicting y from u and v has the form

$$y = \beta_0 + \beta_1 u + \beta_2 v$$

Since there is more than one independent variable, this is called **multiple regression.**

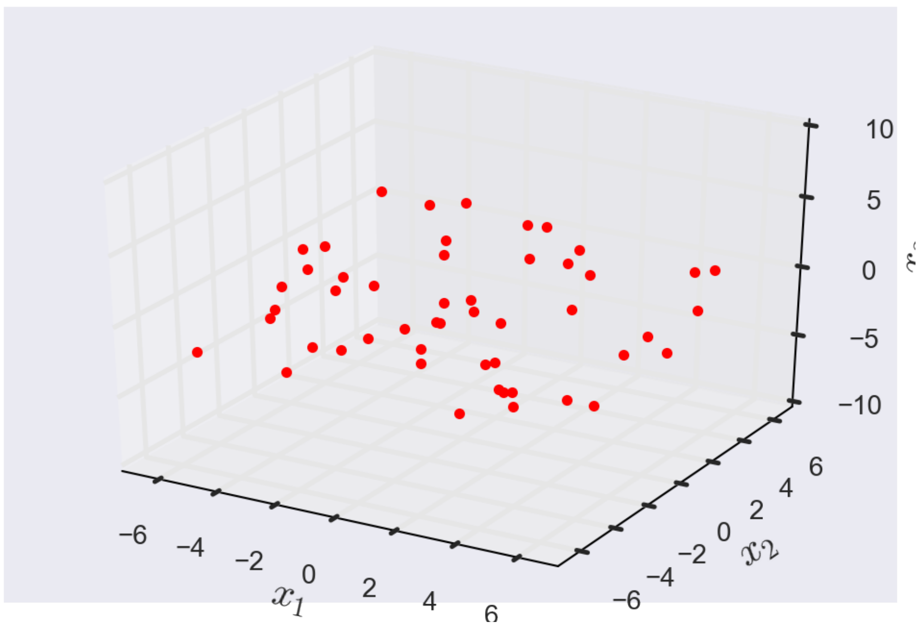
A more general prediction equation might have the form

$$y = \beta_0 + \beta_1 u + \beta_2 v + \beta_3 u^2 + \beta_4 uv + \beta_5 v^2$$

A least squares fit to equations like this is called a **trend surface.**
In general, a linear model will arise whenever y is to be predicted by an equation of the form

$$y = \beta_0 f_0(u, v) + \beta_1 f_1(u, v) + \cdots + \beta_k f_k(u, v)$$

with $f_0, \ldots, f_k$ any sort of known functions and β_0, ..., β_k unknown weights.
Let's take an example. Here are a set of points in $\mathbb{R}^3$:



Example. In geography, local models of terrain are constructed from data $(u_1, v_1, y_1), \ldots, (u_n, v_n, y_n)$ where u_j, v_j, and y_j are latitude, longitude, and altitude, respectively.

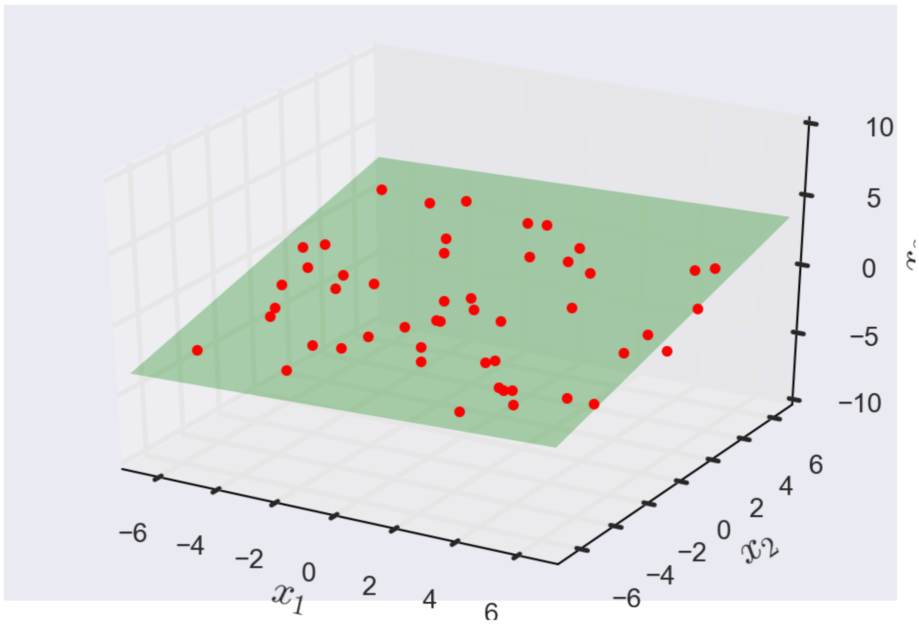
Let's describe the linear models that gives a least-squares fit to such data. The solution is called the least-squares *plane.*

Solution. We expect the data to satisfy these equations:

$$y_1 = \beta_0 + \beta_1 u_1 + \beta_2 v_1 + \epsilon_1$$
$$y_1 = \beta_0 + \beta_1 u_2 + \beta_2 v_2 + \epsilon_2$$
$$\vdots$$
$$y_1 = \beta_0 + \beta_1 u_n + \beta_2 v_n + \epsilon_n$$

This system has the matrix for $\mathbf{y} = X\beta + \epsilon$, where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}, \quad X = \begin{bmatrix} 1 & u_1 & v_1 \\ 1 & u_2 & v_2 \\ \vdots & \vdots & \vdots \\ 1 & u_n & v_n \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}, \quad \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$



This example shows that the linear model for multiple regression has the same abstract form as the model for the simple regression in the earlier examples.

We can see that there the general principle is the same across all the different kinds of linear models.

Once X is defined properly, the normal equations for β have the same matrix form, no matter how many variables are involved.

Thus, for any linear model where X^TX is invertible, the least squares $\hat{\beta}$ is given by $(X^TX)^{-1}X^T\mathbf{y}$.

1.5 Measuring the fit of a regression model and R^2

Given any X and $\mathbf{y}$, the above algorithm will produce an output $\hat{\beta}$.

But how do we know whether the data is in fact well described by the model?

The most common measure of fit is R^2.

R^2 measures the fraction of the variance of $\mathbf{y}$ that can be explained by the model $X\hat{\beta}$.

The variance of $\mathbf{y}$ is Var($\mathbf{y}$) $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y})^2$ where: $\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$

For any given n, we can equally work with just

$$\sum_{i=1}^{n}(y_i - \overline{y})^2$$

which is called the **Total Sum of Squares** (TSS).

Now to measure the quality of git of a model, we break TSS down into two components.

For any given $\mathbf{x}_i$, the prediction made by the model is $\hat{y}_i = \mathbf{x}_i^T \beta$.

Therefore, the residual ϵ is $y_i - \hat{y}_i$, and the part that the model "explains" is $\hat{y}_i - \overline{y_i}$.

Then it turns out that the total sum of squares is exactly equal to the sum of squares of the residuals plus the sum of squares of the explained part.

In other words:

$$\text{TSS} = \text{SSR} + \text{ESS},$$

where Residual Sum of Squares (RSS) is:

$$\text{RSS} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2 ,$$

and Explained Sum of Squares (ESS) is:

$$\text{ESS} = \sum_{i=1}^{n} \left(\hat{y}_i - \overline{y} \right)^2 ,$$

Now, a good fit is one in which the model explains a large part of the variance of $\mathbf{y}$.

So the measure of fit R^2 is defined as:

$$R^2 \quad = \quad \frac{\text{ESS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}} \tag{1}$$

$0 \leq R^2 \leq 1$; the closer the value of R^2 is to 1 the better the fit of the regression; small values of RSS imply that the residuals are small and therefore we have a better fit.

1.6 OLS in Practice

```
In [53]: X, y = datasets.make_regression(n_samples=100, n_features=20, n_informativ
         print(X.shape, y.shape)
```

```
(100, 20) (100,)
```

```
In [54]: model = sm.OLS(y, X)
         results = model.fit()
         print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.969
Model:                            OLS   Adj. R-squared:                  0.961
Method:                 Least Squares   F-statistic:                     123.8
Date:                Tue, 08 Nov 2016   Prob (F-statistic):           1.03e-51
Time:                        12:32:12   Log-Likelihood:                -468.30
No. Observations:                 100   AIC:                             976.6
Df Residuals:                      80   BIC:                             1029.
```

```
Df Model:                          20
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
x1            12.5673      3.471      3.620      0.001         5.659    19.476
x2            -3.8321      2.818     -1.360      0.178        -9.440     1.776
x3            -2.4197      3.466     -0.698      0.487        -9.316     4.477
x4             2.0143      3.086      0.653      0.516        -4.127     8.155
x5            -2.6256      3.445     -0.762      0.448        -9.481     4.230
x6             0.7894      3.159      0.250      0.803        -5.497     7.076
x7            -3.0684      3.595     -0.853      0.396       -10.224     4.087
x8            90.1383      3.211     28.068      0.000        83.747    96.529
x9            -0.0133      3.400     -0.004      0.997        -6.779     6.752
x10           15.2675      3.248      4.701      0.000         8.804    21.731
x11           -0.2247      3.339     -0.067      0.947        -6.869     6.419
x12            0.0773      3.546      0.022      0.983        -6.979     7.133
x13           -0.2452      3.250     -0.075      0.940        -6.712     6.222
x14           90.0179      3.544     25.402      0.000        82.966    97.070
x15            1.6684      3.727      0.448      0.656        -5.748     9.085
x16            4.3945      2.742      1.603      0.113        -1.062     9.851
x17            8.7918      3.399      2.587      0.012         2.028    15.556
x18           73.3771      3.425     21.426      0.000        66.562    80.193
x19           -1.9139      3.515     -0.545      0.588        -8.908     5.080
x20           -1.3206      3.284     -0.402      0.689        -7.855     5.214
==============================================================================
Omnibus:                        5.248   Durbin-Watson:                   2.018
Prob(Omnibus):                  0.073   Jarque-Bera (JB):                4.580
Skew:                           0.467   Prob(JB):                        0.101
Kurtosis:                       3.475   Cond. No.                         2.53
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
```

The R^2 value is very good. We can see that the linear model does a very good job of predicting the observations y_i.

However, some of the independent variables may not contribute to the accuracy of the prediction.

Note that each parameters of an independent variable has an associated confidence interval.

If a coefficient is not distinguishable from zero, then we cannot assume that there is any relationship between the independent variable and the observations.

In other words, if the confidence interval for the parameter includes zero, the associated independent variable may not have any predictive value.

```
In [55]: print('Confidence Intervals: {}'.format(results.conf_int()))
         print('Parameters: {}'.format(results.params))
```

```
Confidence Intervals: [[  5.65891465  19.47559281]
 [ -9.44032559   1.77614877]
 [ -9.31636359   4.47701749]
 [ -4.12661379   8.15524508]
 [ -9.4808662    4.22965424]
 [ -5.49698033   7.07574692]
 [-10.22359973   4.08684835]
 [ 83.74738375  96.52928603]
 [ -6.77896356   6.75226985]
 [  8.80365396  21.73126149]
 [ -6.86882065   6.4194618 ]
 [ -6.97868351   7.1332267 ]
 [ -6.71228582   6.2218515 ]
 [ 82.96557061  97.07028228]
 [ -5.74782503   9.08465366]
 [ -1.06173893   9.85081724]
 [  2.02753258  15.5561241 ]
 [ 66.56165458  80.19256546]
 [ -8.90825108   5.0804296 ]
 [ -7.85545335   5.21424811]]
Parameters: [  1.25672537e+01  -3.83208841e+00  -2.41967305e+00   2.01431564e+00
  -2.62560598e+00   7.89383294e-01  -3.06837569e+00   9.01383349e+01
  -1.33468527e-02   1.52674577e+01  -2.24679428e-01   7.72715974e-02
  -2.45217158e-01   9.00179264e+01   1.66841432e+00   4.39453916e+00
   8.79182834e+00   7.33771100e+01  -1.91391074e+00  -1.32060262e+00]
```

```
In [56]: CIs = results.conf_int()
         notSignificant = (CIs[:,0] < 0) & (CIs[:,1] > 0)
         notSignificant
```

```
Out[56]: array([False,  True,  True,  True,  True,  True,  True, False,  True,
                False,  True,  True,  True, False,  True,  True, False, False,
                 True,  True], dtype=bool)
```

```
In [57]: Xsignif = X[:,~notSignificant]
         Xsignif.shape
```

```
Out[57]: (100, 6)
```

By eliminating independent variables that are not significant, we help avoid overfitting.

```
In [58]: model = sm.OLS(y, Xsignif)
         results = model.fit()
         print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.965
```

```
Model:                            OLS   Adj. R-squared:                  0.963
Method:                 Least Squares   F-statistic:                     437.1
Date:                Tue, 08 Nov 2016   Prob (F-statistic):           2.38e-66
Time:                        12:32:12   Log-Likelihood:                -473.32
No. Observations:                 100   AIC:                             958.6
Df Residuals:                      94   BIC:                             974.3
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
x1            11.9350      3.162      3.775      0.000         5.657    18.213
x2            90.5841      2.705     33.486      0.000        85.213    95.955
x3            14.3652      2.924      4.913      0.000         8.560    20.170
x4            90.5586      3.289     27.535      0.000        84.028    97.089
x5             8.3185      3.028      2.747      0.007         2.307    14.330
x6            71.9119      3.104     23.169      0.000        65.749    78.075
==============================================================================
Omnibus:                        9.915   Durbin-Watson:                   2.056
Prob(Omnibus):                  0.007   Jarque-Bera (JB):               11.608
Skew:                           0.551   Prob(JB):                      0.00302
Kurtosis:                       4.254   Cond. No.                         1.54
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
```

1.7 Real Data: California Housing

```
In [59]: ca = pd.read_table("data/cal_housing.data", sep=',')

         attributes = ['longitude',
                       'latitude',
                       'housingMedianAge',
                       'totalRooms',
                       'totalBedrooms',
                       'population',
                       'households',
                       'medianIncome',
                       'medianHouseValue']

         ca.columns = attributes

         ca.info()
         ca.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 20639 entries, 0 to 20638
Data columns (total 9 columns):
longitude           20639 non-null float64
latitude            20639 non-null float64
housingMedianAge    20639 non-null float64
totalRooms          20639 non-null float64
totalBedrooms       20639 non-null float64
population          20639 non-null float64
households          20639 non-null float64
medianIncome        20639 non-null float64
medianHouseValue    20639 non-null float64
dtypes: float64(9)
memory usage: 1.4 MB
```

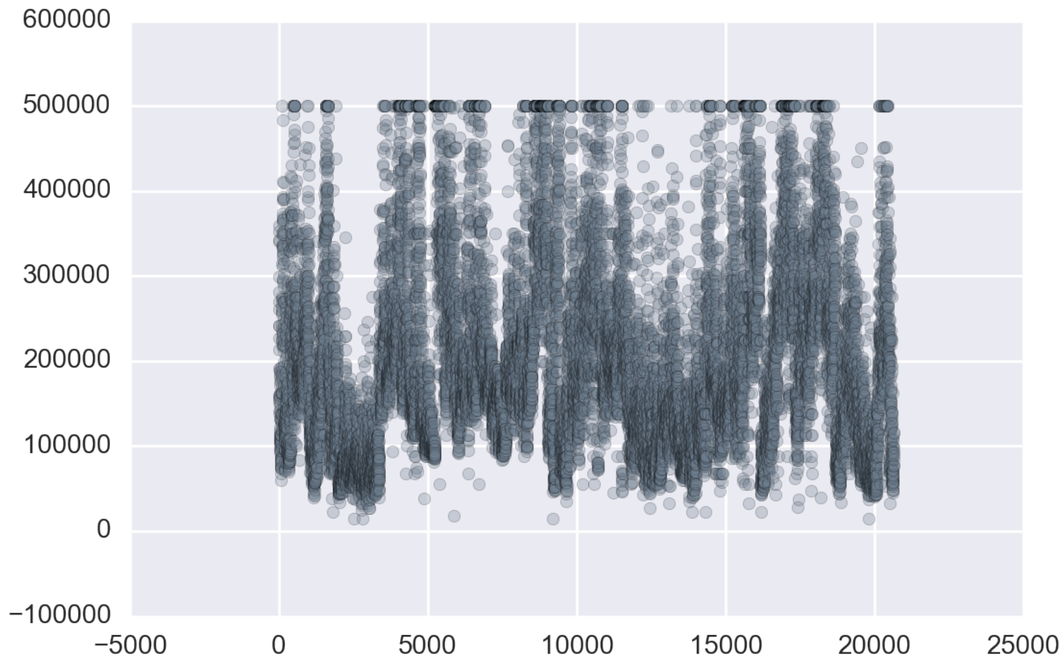
```
Out[59]:   longitude  latitude  housingMedianAge  totalRooms  totalBedrooms  \
        0    -122.22     37.86              21.0      7099.0         1106.0
        1    -122.24     37.85              52.0      1467.0          190.0
        2    -122.25     37.85              52.0      1274.0          235.0
        3    -122.25     37.85              52.0      1627.0          280.0
        4    -122.25     37.85              52.0       919.0          213.0
        5    -122.25     37.84              52.0      2535.0          489.0
        6    -122.25     37.84              52.0      3104.0          687.0
        7    -122.26     37.84              42.0      2555.0          665.0
        8    -122.25     37.84              52.0      3549.0          707.0
        9    -122.26     37.85              52.0      2202.0          434.0

           population  households  medianIncome  medianHouseValue
        0      2401.0      1138.0        8.3014          358500.0
        1       496.0       177.0        7.2574          352100.0
        2       558.0       219.0        5.6431          341300.0
        3       565.0       259.0        3.8462          342200.0
        4       413.0       193.0        4.0368          269700.0
        5      1094.0       514.0        3.6591          299200.0
        6      1157.0       647.0        3.1200          241400.0
        7      1206.0       595.0        2.0804          226700.0
        8      1551.0       714.0        3.6912          261100.0
        9       910.0       402.0        3.2031          281500.0
```

```
In [60]: X_CA_H = ca[['longitude','latitude','housingMedianAge','totalRooms',
                      'totalBedrooms','population','households','medianIncome']]
         print('Complete dataset shape is {}'.format(X_CA_H.shape))
         print('Sample median house values:')
         print(ca.medianHouseValue.head())
         y_CA_H = ca.medianHouseValue;
         plt.scatter(range(len(y_CA_H)), y_CA_H, c="slategray", alpha=0.3, linewidt
```

```
Complete dataset shape is (20639, 8)
Sample median house values:
```

```
0    358500.0
1    352100.0
2    341300.0
3    342200.0
4    269700.0
Name: medianHouseValue, dtype: float64
```

```
Out[60]: <matplotlib.collections.PathCollection at 0x11ec97668>
```

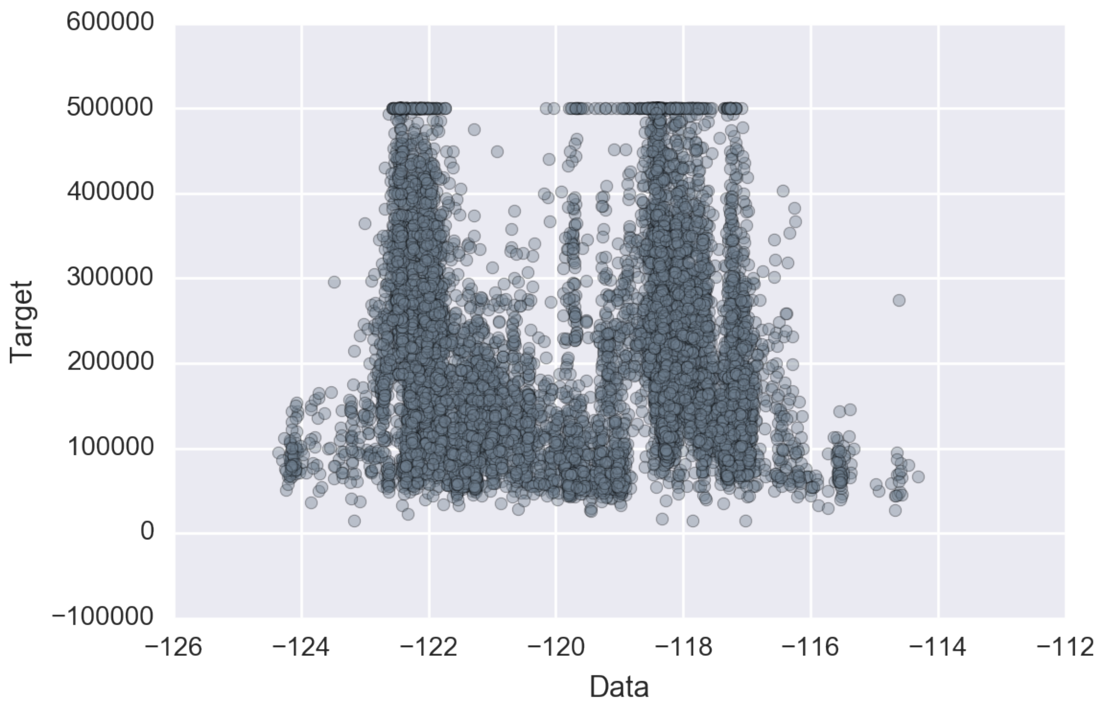



```
In [61]: X_CA_H, y_CA_H = utils.shuffle(X_CA_H, y_CA_H, random_state=1)
         X_CA_H_train, X_CA_H_test, y_CA_H_train, y_CA_H_test = cross_validation.tr
             X_CA_H, y_CA_H, test_size=0.4, random_state=0)
         print((X_CA_H_train.shape), y_CA_H_train.shape)
         print((X_CA_H_test.shape), y_CA_H_test.shape)
```

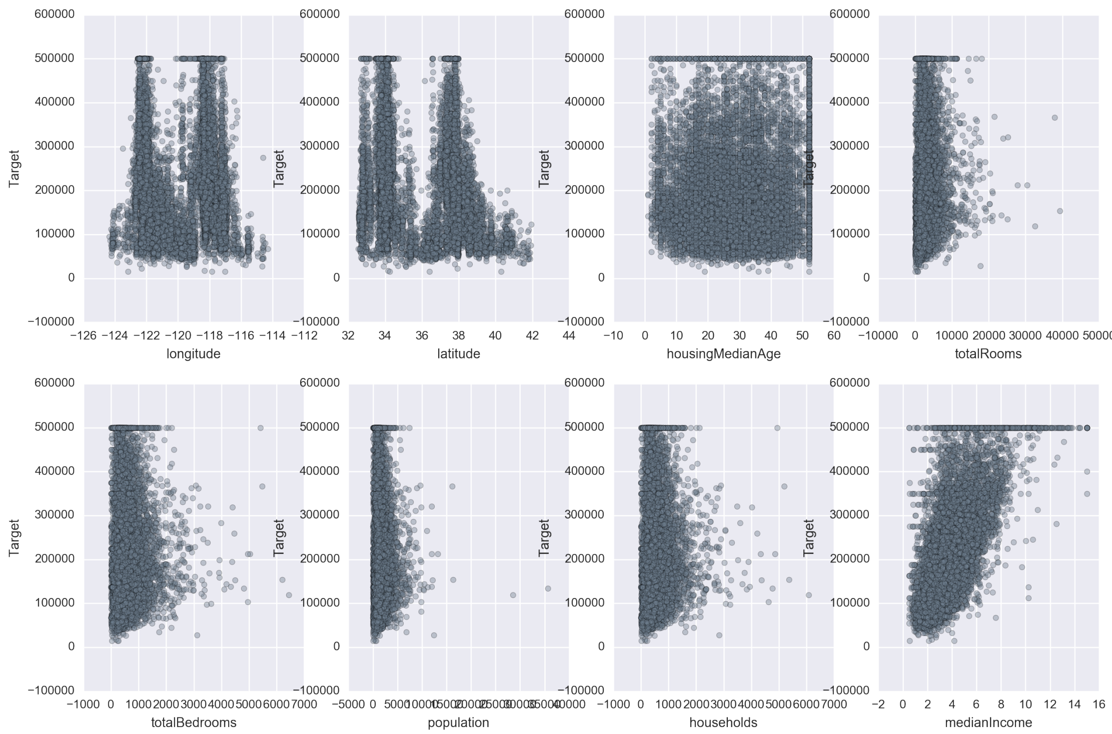
```
(12383, 8) (12383,)
(8256, 8) (8256,)
```

```
In [62]: print(X_CA_H_train[X_CA_H_train.columns[0]].head(10))
         subX_train = X_CA_H_train[X_CA_H_train.columns[0]]
         subX_test = X_CA_H_test[X_CA_H_test.columns[0]]
         plt.scatter(subX_train, y_CA_H_train, c="slategray", alpha=0.4, linewidths
         # plt.scatter(subX_test, y_test, c="seagreen", alpha=0.2, linewidths=0.3)
         plt.xlabel('Data')
         plt.ylabel('Target');
```

```
17929   -121.96
5579    -118.30
7206    -118.18
12271   -116.99
12254   -117.02
8054    -118.18
17197   -119.75
1435    -122.01
193     -122.25
7964    -118.19
Name: longitude, dtype: float64
```




```
In [63]: fig, axes = plt.subplots(2,4,figsize=(15,10))

         for i in range(8):
             plt_i = i // 4
             plt_j = i % 4
             subX_train = X_CA_H_train[X_CA_H_train.columns[i]]
             # plt.subplot(2, 4, 1 + i)
             axes[plt_i][plt_j].scatter(subX_train, y_CA_H_train, c="slategray", al
             #plt.scatter(subX_test, y_test)
             axes[plt_i][plt_j].set_xlabel(X_CA_H_train.columns[i])
             axes[plt_i][plt_j].set_ylabel('Target');
```




```
In [64]: model = sm.OLS(y_CA_H_train, X_CA_H_train)
         results = model.fit()
         print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:      medianHouseValue   R-squared:                       0.900
Model:                           OLS   Adj. R-squared:                  0.900
Method:                Least Squares   F-statistic:                 1.393e+04
Date:               Tue, 08 Nov 2016   Prob (F-statistic):               0.00
Time:                       12:32:17   Log-Likelihood:            -1.5658e+05
No. Observations:              12383   AIC:                         3.132e+05
Df Residuals:                  12375   BIC:                         3.132e+05
Df Model:                          8
Covariance Type:           nonrobust
================================================================================
                      coef    std err          t      P>|t|      [95.0% Conf. Int.
--------------------------------------------------------------------------------
longitude       -2146.8860    140.225    -15.310      0.000     -2421.748 -1872.02
latitude        -8194.2243    445.094    -18.410      0.000     -9066.677 -7321.77
housingMedianAge 1883.6351     58.158     32.388      0.000      1769.636  1997.63
totalRooms        -13.9877      1.088    -12.856      0.000       -16.120   -11.85
totalBedrooms      68.2261      9.785      6.973      0.000        49.047    87.40
population        -39.7481      1.473    -26.977      0.000       -42.636   -36.86
```

```
households         137.4371     10.524     13.059      0.000       116.807   158.06
medianIncome      4.567e+04    452.493    100.922      0.000      4.48e+04  4.66e+0
==============================================================================
Omnibus:                     2777.677   Durbin-Watson:                   1.977
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             9539.379
Skew:                           1.114   Prob(JB):                         0.00
Kurtosis:                       6.677   Cond. No.                     3.03e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
[2] The condition number is large, 3.03e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```