CovarianceCorrelation

December 20, 2016

1 Covariance and Correlation

Covariance measures how two variables vary in tandem from their means.

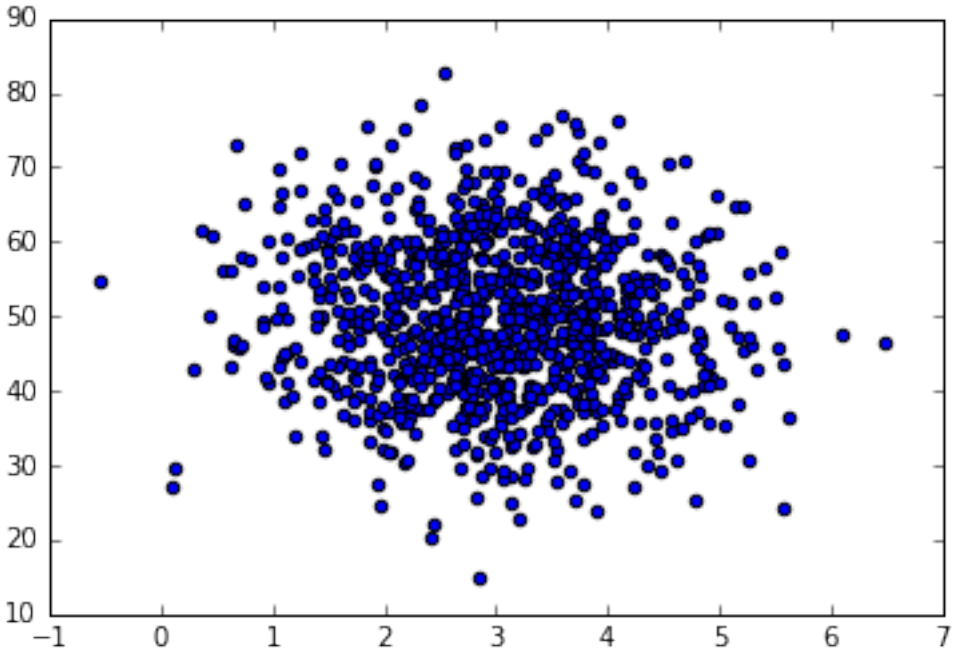
For example, let's say we work for an e-commerce company, and they are interested in finding a correlation between page speed (how fast each web page renders for a customer) and how much a customer spends.

numpy offers covariance methods, but we'll do it the "hard way" to show what happens under the hood. Basically we treat each variable as a vector of deviations from the mean, and compute the "dot product" of both vectors. Geometrically this can be thought of as the angle between the two vectors in a high-dimensional space, but you can just think of it as a measure of similarity between the two variables.

First, let's just make page speed and purchase amount totally random and independent of each other; a very small covariance will result as there is no real correlation:

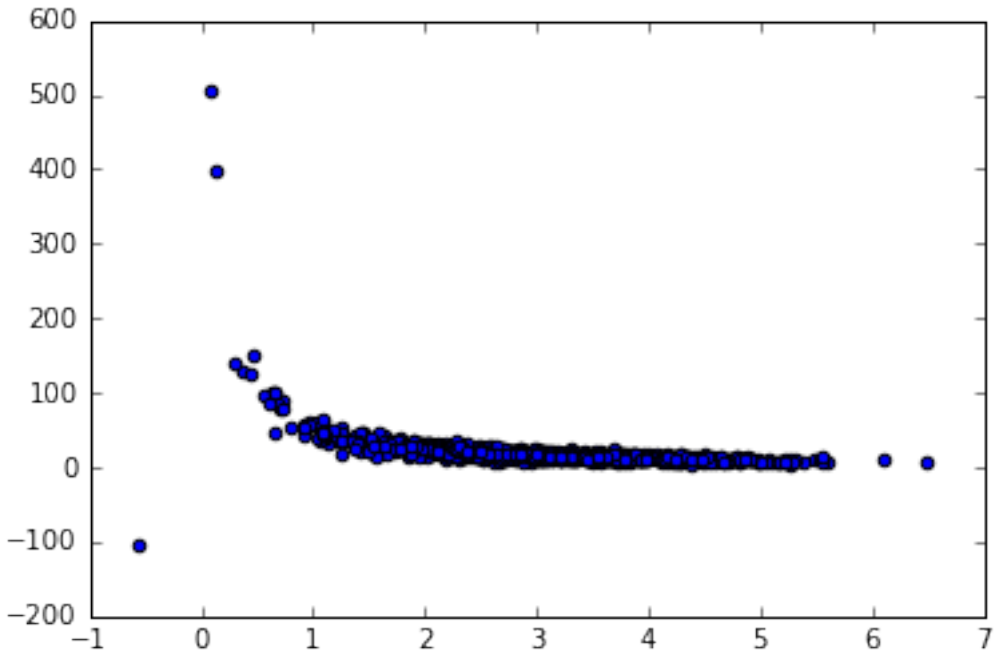
In [1]:
```
%matplotlib inline

import numpy as np
from pylab import *

def de_mean(x):
    xmean = mean(x)
    return [xi - xmean for xi in x]

def covariance(x, y):
    n = len(x)
    return dot(de_mean(x), de_mean(y)) / (n-1)

pageSpeeds = np.random.normal(3.0, 1.0, 1000)
purchaseAmount = np.random.normal(50.0, 10.0, 1000)

scatter(pageSpeeds, purchaseAmount)

covariance (pageSpeeds, purchaseAmount)
```

Out[1]:
```
-0.52925537535873524
```



Now we'll make our fabricated purchase amounts an actual function of page speed, making a very real correlation. The negative value indicates an inverse relationship; pages that render in less time result in more money spent:

```
In [2]: purchaseAmount = np.random.normal(50.0, 10.0, 1000) / pageSpeeds

        scatter(pageSpeeds, purchaseAmount)

        covariance (pageSpeeds, purchaseAmount)

Out[2]: -11.312423794287247
```



But, what does this value mean? Covariance is sensitive to the units used in the variables, which makes it difficult to interpret. Correlation normalizes everything by their standard deviations, giving you an easier to understand value that ranges from -1 (for a perfect inverse correlation) to 1 (for a perfect positive correlation):

```
In [3]: def correlation(x, y):
            stddevx = x.std()
            stddevy = y.std()
            return covariance(x,y) / stddevx / stddevy  #In real life you'd check for divide by

        correlation(pageSpeeds, purchaseAmount)
```

```
Out[3]: -0.46775563114087165
```

numpy can do all this for you with numpy.corrcoef. It returns a matrix of the correlation coefficients between every combination of the arrays passed in:

```
In [4]: np.corrcoef(pageSpeeds, purchaseAmount)
```

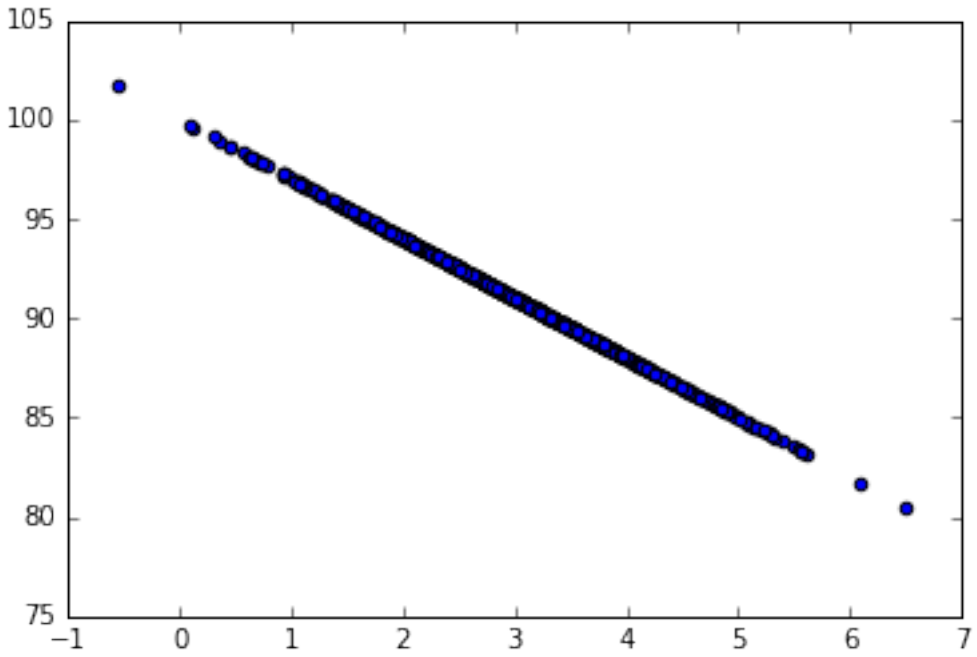
```
Out[4]: array([[ 1.        , -0.46728788],
               [-0.46728788,  1.        ]])
```

(It doesn't match exactly just due to the math precision available on a computer.)

We can force a perfect correlation by fabricating a totally linear relationship (again, it's not exactly -1 just due to precision errors, but it's close enough to tell us there's a really good correlation here):

```
In [5]: purchaseAmount = 100 - pageSpeeds * 3

        scatter(pageSpeeds, purchaseAmount)

        correlation (pageSpeeds, purchaseAmount)
```

```
Out[5]: -1.0010010010010013
```



Remember, correlation does not imply causality!

1.1 Activity

numpy also has a numpy.cov function that can compute Covariance for you. Try using it for the pageSpeeds and purchaseAmounts data above. Interpret its results, and compare it to the results from our own covariance function above.

```
In [ ]:
```