CoVarience and CoRelation

August 21, 2021

[32]:
```python
import numpy as np
from pylab import *
from scipy.stats import norm
import matplotlib.pyplot as plt
```

[33]:
```python
#return vector that represent devation from mean of datapoint
def devation_from_mean(a_vectorList):

    #calculate mean of datapoint
    mean_of_datapoint=mean(a_vectorList)

    #go through each datapoint in the list and substract datpoint from the mean␣
 ↪and create another list
    return [datapoint-mean_of_datapoint for datapoint in a_vectorList]
```

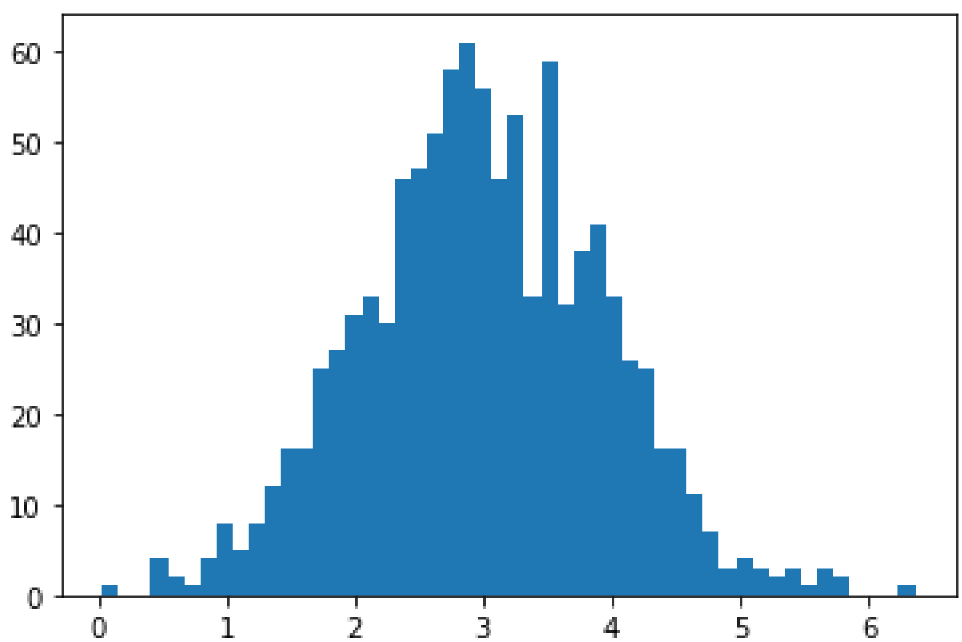
[34]:
```python
def covariance(a_vector,b_vector):

    length_of_vector=len(a_vector)

    a_devation_from_mean_vector=devation_from_mean(a_vector)
    b_devation_from_mean_vector=devation_from_mean(b_vector)

     #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.
    return dot(a_devation_from_mean_vector,b_devation_from_mean_vector)/
 ↪length_of_vector-1
```

[35]:
```python
#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.
```

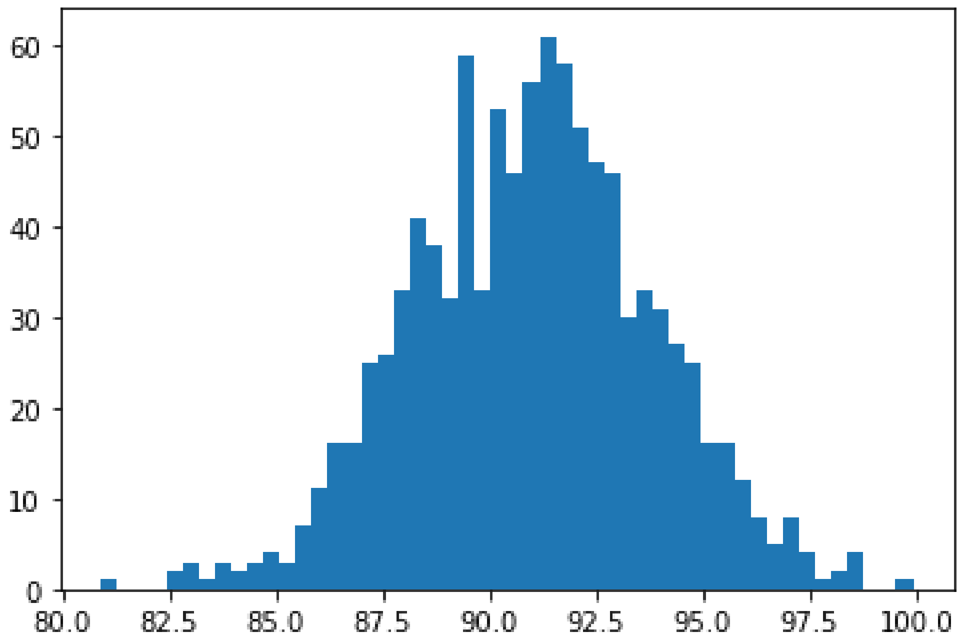
```
#First, let's just make page speed and purchase amount totally random and␣
↪independent of each other;
#a very small covariance will result as there is no real correlation:

#generating normally distrubuted random data
pageSpeed=np.random.normal(3.0,1.0,1000)
purchaseAmount = np.random.normal(50.0,10.0,1000)
```

[56]:
```
plt.hist(pageSpeed,50)
```

[56]:
```
(array([ 1.,  0.,  0.,  4.,  2.,  1.,  4.,  8.,  5.,  8., 12., 16., 16.,
        25., 27., 31., 33., 30., 46., 47., 51., 58., 61., 56., 46., 53.,
        33., 59., 32., 38., 41., 33., 26., 25., 16., 16., 11.,  7.,  3.,
         4.,  3.,  2.,  3.,  1.,  3.,  2.,  0.,  0.,  0.,  1.]),
 array([0.02920481, 0.15592906, 0.28265332, 0.40937757, 0.53610182,
        0.66282608, 0.78955033, 0.91627458, 1.04299883, 1.16972309,
        1.29644734, 1.42317159, 1.54989585, 1.6766201 , 1.80334435,
        1.93006861, 2.05679286, 2.18351711, 2.31024137, 2.43696562,
        2.56368987, 2.69041413, 2.81713838, 2.94386263, 3.07058689,
        3.19731114, 3.32403539, 3.45075965, 3.5774839 , 3.70420815,
        3.83093241, 3.95765666, 4.08438091, 4.21110517, 4.33782942,
        4.46455367, 4.59127793, 4.71800218, 4.84472643, 4.97145069,
        5.09817494, 5.22489919, 5.35162345, 5.4783477 , 5.60507195,
        5.73179621, 5.85852046, 5.98524471, 6.11196897, 6.23869322,
        6.36541747]),
 <BarContainer object of 50 artists>)
```

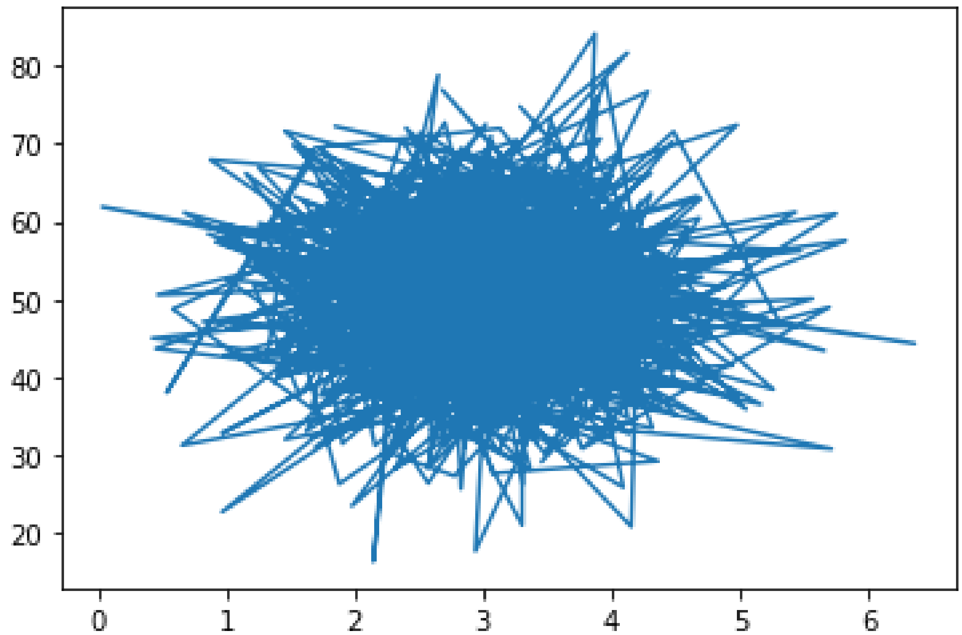


```
[61]: plt.hist(purchaseAmount,50)
      plt.show()
```



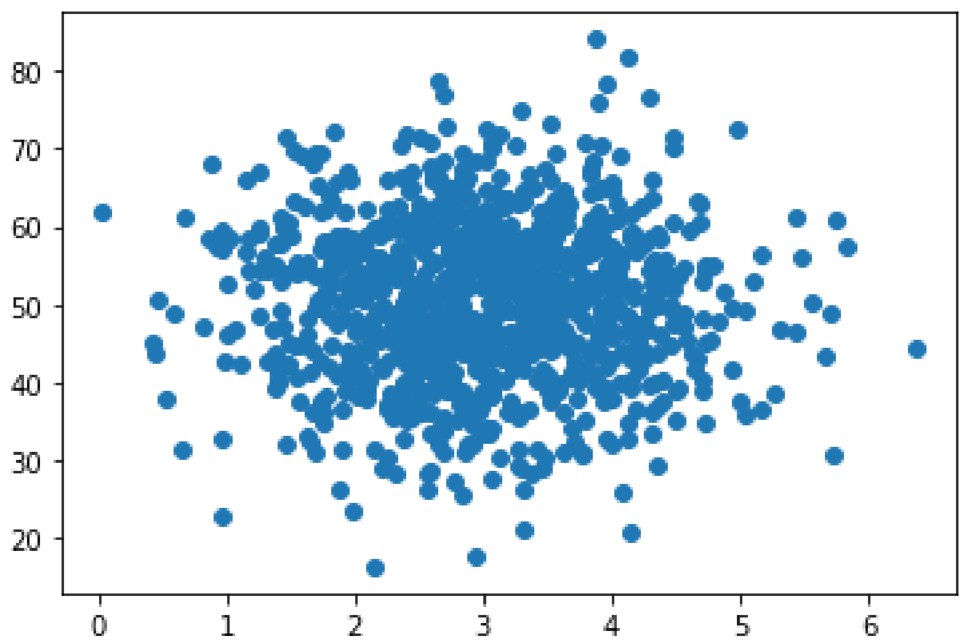

```
[36]: plt.plot(pageSpeed,purchaseAmount)
```

```
[36]: [<matplotlib.lines.Line2D at 0x7fcdd6267100>]
```



[37]:
```
scatter(pageSpeed, purchaseAmount)
```

[37]: <matplotlib.collections.PathCollection at 0x7fcdd6229d60>



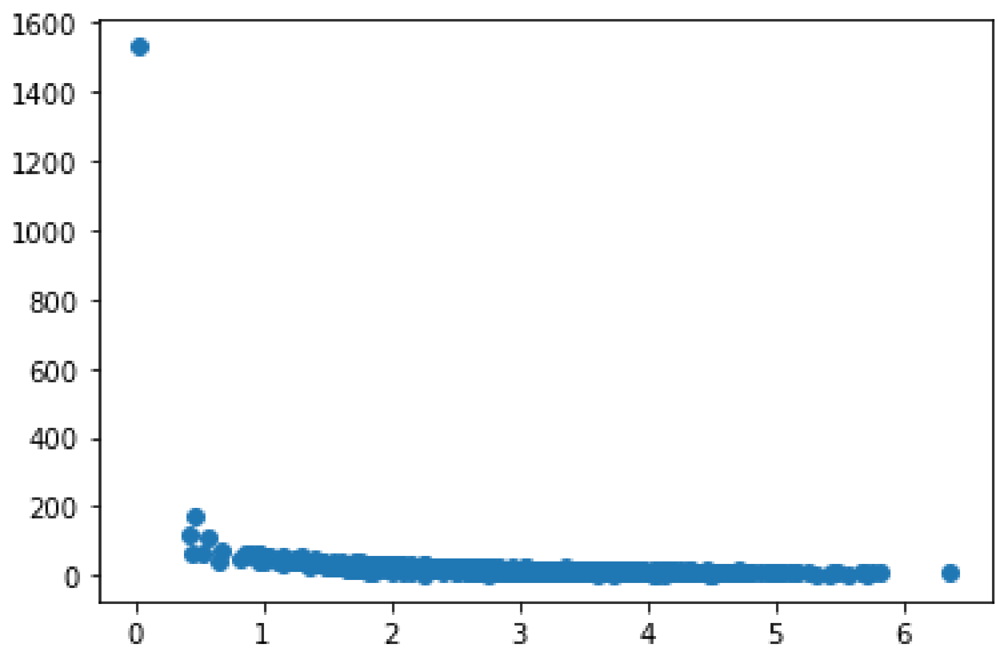
[38]:
```python
##a very small covariance will result as there is no real correlation:
covariance(pageSpeed,purchaseAmount)
```

[38]: -0.9722325925601741

[39]:
```python
#Now we'll make our fabricated purchase amounts an actual function of page␣
↪speed, making a very real correlation.
#making purchaseAmount function of pageSpeed
purchaseAmount = np.random.normal(50.0, 10.0, 1000) / pageSpeed
```

[41]:
```python
scatter(pageSpeed, purchaseAmount)
#The negative value indicates an inverse relationship; pages that render in␣
↪less time result in more money spent:
covariance (pageSpeed, purchaseAmount)
```

[41]: -12.917921299873793



[42]:
```python
#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):
```

[46]:
```
def corelation(a_vector,b_vector):
    stddev_a_vector=a_vector.std()
    stddev_b_vector=b_vector.std()
    return covariance(a_vector,b_vector)/(stddev_a_vector * stddev_b_vector)
```

[47]:
```
#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):
corelation(pageSpeed,purchaseAmount)
```

[47]:
```
-0.28036399177805077
```

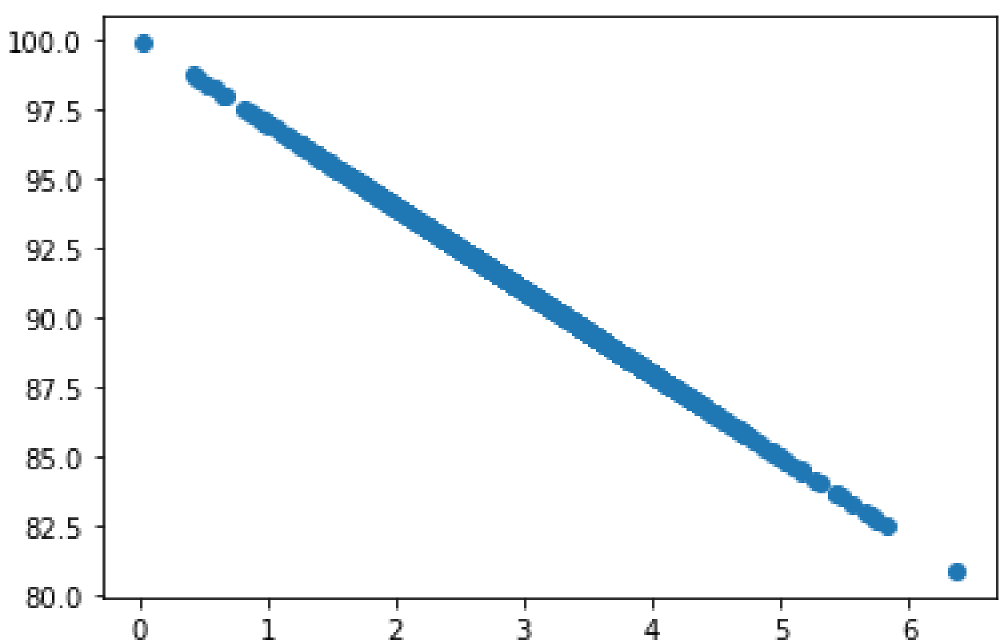
[48]:
```
#umpy can do all this for you with numpy.corrcoef. It returns a matrix of the␣
 ↪correlation coefficients between every combination of the arrays passed in:

np.corrcoef(pageSpeed,purchaseAmount)
```

[48]:
```
array([[ 1.        , -0.2586605],
       [-0.2586605,  1.        ]])
```

[49]:
```
#We can force a perfect correlation by fabricating a totally linear relationship
#making purchaseAmount function of pageSpeed
purchaseAmount = 100-pageSpeed*3
```

[50]:
```
scatter(pageSpeed,purchaseAmount)
```

[50]:
```
<matplotlib.collections.PathCollection at 0x7fcdb96a8910>
```



[51]:
```
corelation(pageSpeed,purchaseAmount)
```

[51]: -1.3775310518076287

[52]:
```
np.corrcoef(pageSpeed,purchaseAmount)
```

[52]:
```
array([[ 1., -1.],
       [-1.,  1.]])
```

[53]:
```
#calculation co-varience
np.cov(pageSpeed,purchaseAmount)
```

[53]:
```
array([[ 0.88381339, -2.65144018],
       [-2.65144018,  7.95432055]])
```

[55]:
```
```

```
zsh:1: command not found: conda
```

[63]:
```
! conda install nbconvert
```

```
zsh:1: command not found: conda
```

[ ]:
```
```