SciComp With Py

Getting Started with Machine Learning: Covariance, Correlation, and Curve Fitting

Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

- Standard Deviation, Covariance, & Correlation
- Data Modeling with Curve Fitting

Standard Deviation, Covariance, & Correlation

Standard Deviation

- Suppose there is a certain population (e.g., the petals of all roses in your garden, the salaries of all employees in a company, etc.)
- We can either take measurements of the entire population or take a sample of the population
- Suppose that we want to measure how spread out the numbers are; if we measure the entire population, we have population standard deviation; if we measure just a sample, we have sample standard deviation

Population Standard Deviation

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

number of elements in population

population's mean

Sample Standard Deviation

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

number of elements in sample

sample's mean

Population and Sample STD in Numpy

```
>>> np.std([5, 4, 9, 2, 12, 7])
3.304037933599835
```

population STD

```
>>> np.std([5, 4, 9, 2, 12, 7], ddof=1)
3.619392214170771
```

sample STD

Population and Sample STD in Numpy

```
>>> np.std([5, 4, 9, 2, 12, 7])
3.3040379335998349
>>> np.std([9, 2, 5, 4, 12, 7], ddof=1)
3.6193922141707713
>>> lst = [5, 4, 9, 2, 12, 7]
>>> m = np.mean(lst)
>>> s = [(xi - m)**2 for xi in lst]
>>> math.sqrt(sum(s)/len(s))
3.304037933599835
>>> math.sqrt(sum(s)/(len(s)-1))
3.6193922141707713
```

population STD

sample STD

Covariance and Correlation

- Covariance and correlation are mathematical means of measuring how the values of two variables are related to each other
- Example 1: T is the temperature inside a beehive; N is the number of the fanning worker bees
- Example 2: A is a person's age; M is the amount of money a person spends per year

Measuring Covariance

- Given two vectors of values, V1 and V2, of the same size where V1 consists of the values of the first variable (say X) and V2 consists of the values of the second variable (say Y)
- Compute the means of V1 and V2
- Compute DV1 as the vector of the deviations of each value in V1 from V1's mean
- Compute DV2 as the vector of the deviations of each value in V2 from V2's mean
- Compute the dot product between DV1 and DV2 and divide by the number of elements in DV1

Computing Covariance

pylab is another library that comes in handy in addition to numpy

```
import pylab as plb

def dev_from_mean(x):
  xmean = plb.mean(x)
  return [xi - xmean for xi in x]


def covariance(x, y):
  n = len(x)
  x_dev = dev_from_mean(x)
  y_dev = dev_from_mean(y)
  return plb.dot(x_dev, y_dev) / (n-1)
```

Interpreting Covariance

- Covariance can be positive or negative (this is the so called inverse covariance)
- A small covariance close to 0 means that there is not much relationship between the two variables
- But what if covariance is large?
- How large does it have to be for us to say that the two variables are correlated?

Enter Correlation

- Correlation is covariance divided by the standard deviations of both variables
- Correlation makes it easier for us to interpret the degree of relationship between two variables
- A correlation of 0 means there is no relationship
- A correlation of 1 means that the two variables are directly related
- A correlation of -1 means that the two variables are inversely related

Computing Correlation

standard deviation of a numpy vector

```
def correlation(x, y):
  stdx = x.std()
  stdy = y.std()
  return covariance(x, y) / stdx / stdy
```

Correlation and Causality

Correlation does not necessarily imply causality

Determining Causality

Causality is determined by experiment and correlation tells you what experiments you may want to run

Problem

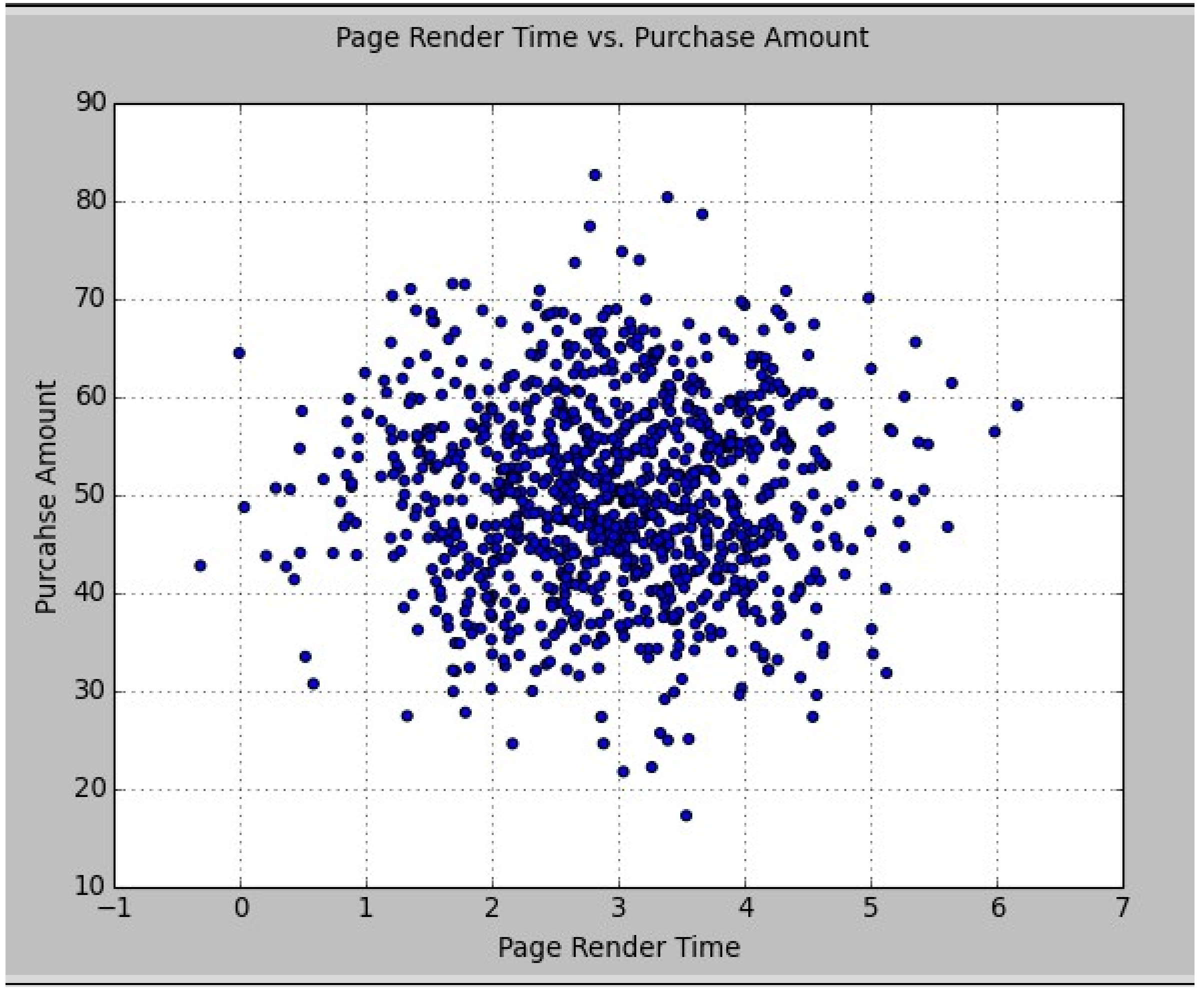
You are a data scientist at an e-commerce company. The company is interested in finding a correlation between page rendition time (how fast a page displays to customer) and the amount of money a customer spends on that page. As the data are being collected, you start playing with simulated data to understand the types of feasible relationships between the two variables.

Simulated Solution: Low Covariance

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


fig1 = plt.figure(1)
fig1.suptitle('Page Render Time vs. Purchase Amount')
plt.xlabel('Page Render Time')
plt.ylabel('Purcahse Amount')
plt.grid()
plt.scatter(pageRenderTime, purchaseAmount)


print('covar = %f' % covariance(pageRenderTime,purchaseAmount))
plt.show()
```



covar = 0.038134

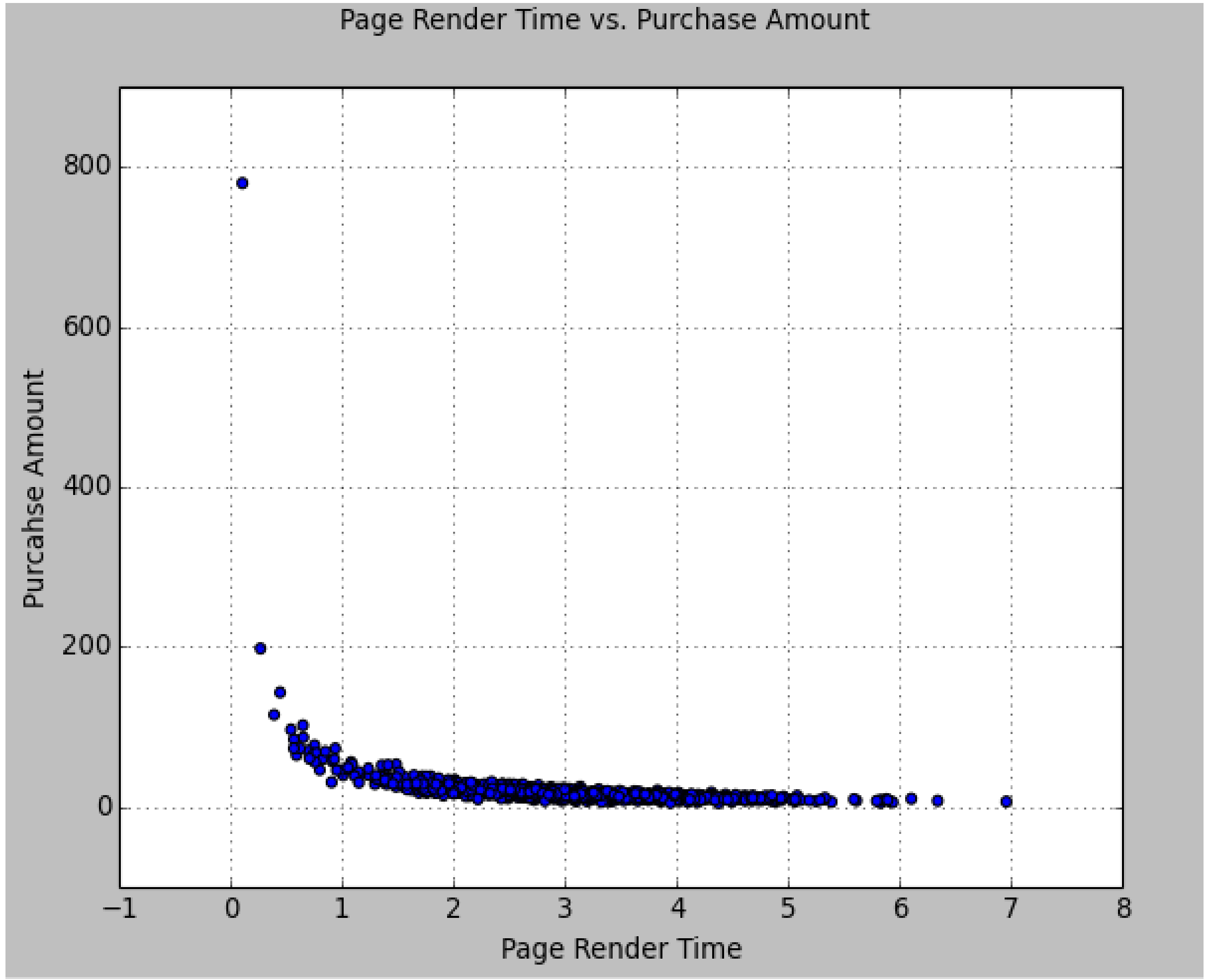
source in covar_correl.py

Simulated Solution: Large Inverse Covariance

```
pageRenderTime = np.random.normal(3.0, 1.0, 1000)
purchaseAmount = np.random.normal(50.0, 10.0, 1000) /pageRenderTime

fig1 = plt.figure(1)
fig1.suptitle('Page Render Time vs. Purchase Amount')
plt.xlabel('Page Render Time')
plt.ylabel('Purcahse Amount')
plt.grid()
plt.scatter(pageRenderTime, purchaseAmount)

print(covariance(pageRenderTime, purchaseAmount))
plt.show()
```



covar = -10.8016783288

source in covar_correl_02.py

Simulated Solution: Using Numpy

Py

```
pageRenderTime = np.random.normal(3.0, 1.0, 1000)
purchaseAmount = np.random.normal(50.0, 10.0, 1000) /pageRenderTime

print('covar=%f' % covariance(pageRenderTime, purchaseAmount))
print('correl=%f' % correlation(pageRenderTime, purchaseAmount))

## here is how you can compute correlation and covariance w/ numpy.
print('Numpy Correlation')
print(np.corrcoef(pageRenderTime, purchaseAmount))
print('Numpy Covariance')
print(np.cov(pageRenderTime, purchaseAmount))
```

Output

```
covar=-11.346364
correl=-0.461441
Numpy Correlation
[[ 1.         -0.46097982]
 [-0.46097982  1.        ]]
Numpy Covariance
[[   0.95645996  -11.34636365]
 [ -11.34636365  633.40687855]]
```

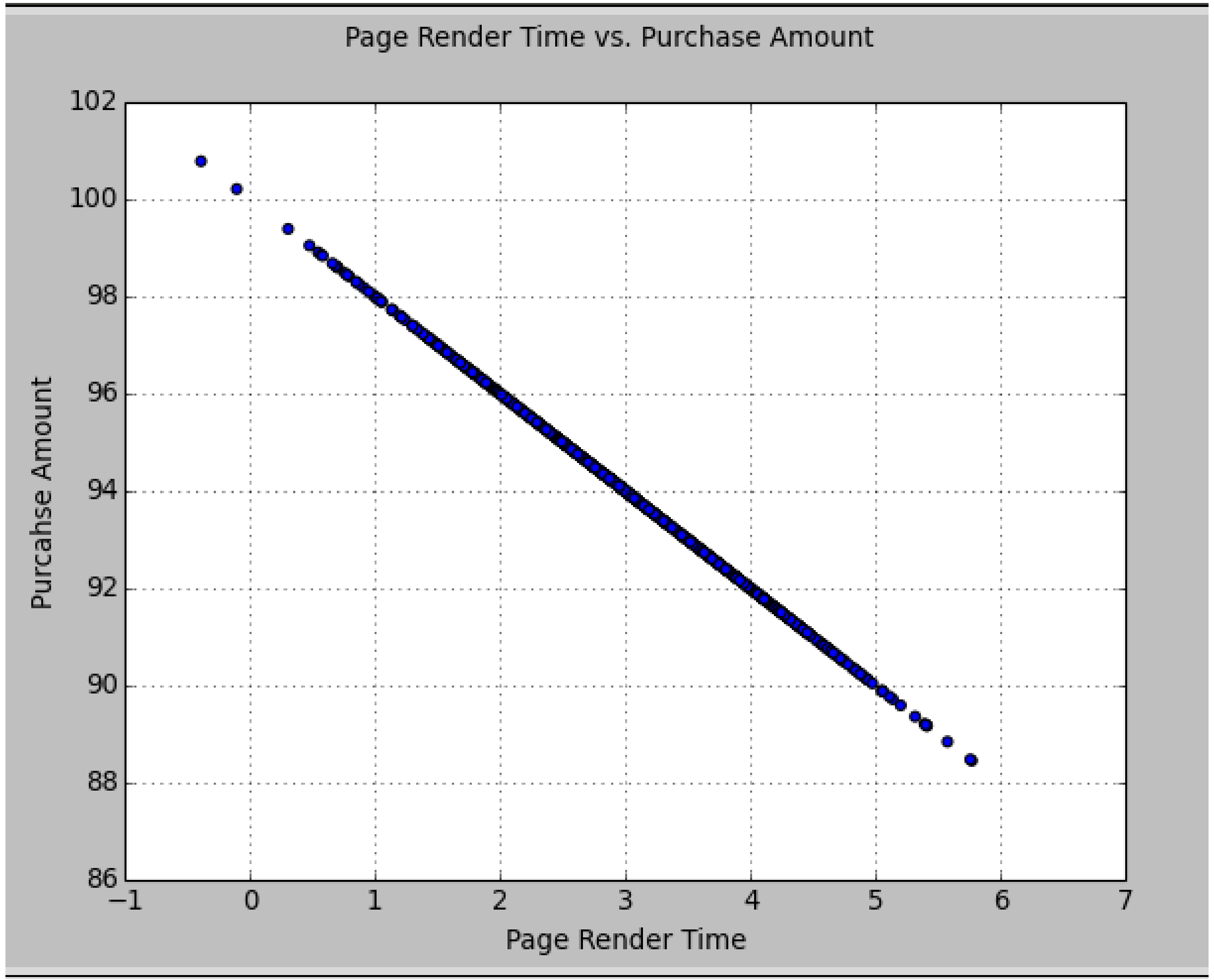
source in covar_correl_03.py

Simulated Solution: Perfect Inverse Correlation

```
## an example of strong negative correlation
pageRenderTime = np.random.normal(3.0, 1.0, 1000)
purchaseAmount = 100 - pageRenderTime * 2


fig1 = plt.figure(1)
fig1.suptitle('Page Render Time vs. Purchase Amount')
plt.xlabel('Page Render Time')
plt.ylabel('Purcahse Amount')
plt.grid()
plt.scatter(pageRenderTime, purchaseAmount)


print('covar=%f' % covariance(pageRenderTime, purchaseAmount))
print('correl=%f' % correlation(pageRenderTime, purchaseAmount))
plt.show()
```



covar=-2.015354

correl=-1.001001

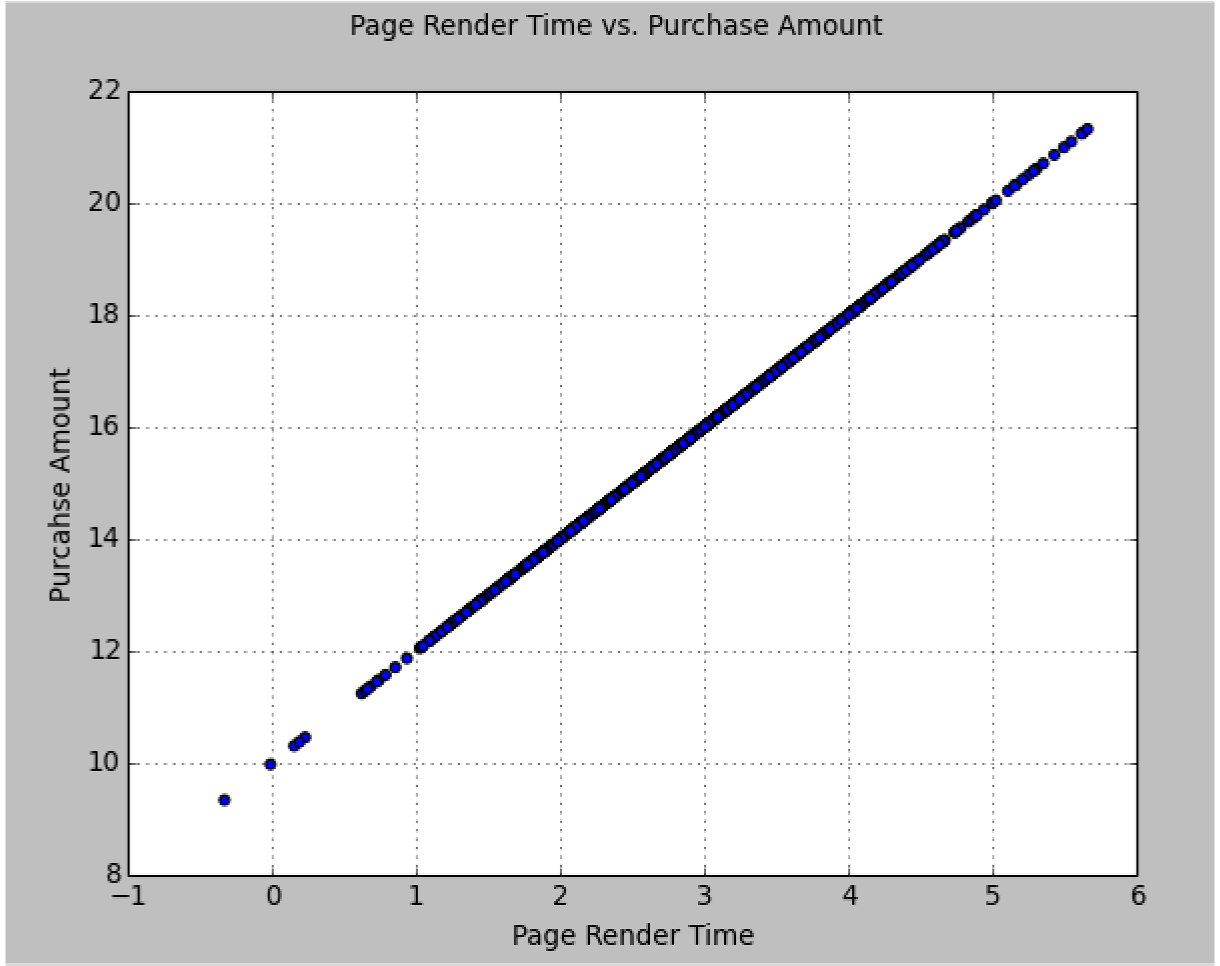
source in covar_correl_04.py

Simulated Solution: Perfect Direct Correlation

```
## an example of perfect direct correlation
pageRenderTime = np.random.normal(3.0, 1.0, 1000)
purchaseAmount = 10 + pageRenderTime * 2

fig1 = plt.figure(1)
fig1.suptitle('Page Render Time vs. Purchase Amount')
plt.xlabel('Page Render Time')
plt.ylabel('Purcahse Amount')
plt.grid()
plt.scatter(pageRenderTime, purchaseAmount)

print('covar=%f' % covariance(pageRenderTime, purchaseAmount))
print('correl=%f' % correlation(pageRenderTime, purchaseAmount))
plt.show()
```



covar=1.847916

correl=1.001001

source in covar_correl_05.py

Data Modeling with Curve Fitting

Prediction of Additional Server Need

Our hypothetical web startup, which sells the service of providing ML algorithms via HTTP. With the increasing success of our company, the demand for better infrastructure also increases the necessity to serve all incoming web requests successfully. We don't want to allocate too many resources because of the costs. On the other hand, we'll lose money if we haven't reserved enough resources for serving all incoming requests. The question now is, when will we hit the limit of our current infrastructure, which we estimated being 100,000 requests per hour. We'd like to know in advance when we need additional servers in the cloud to serve all the incoming requests successfully without paying for unused ones.

Ch. 1, p. 19. Richert & Coeho. “Building ML Systems with Py.”

First 15 Lines Of Data File WEB_TRAFFIC.TSV

1	2272
2	nan
3	1386
4	1365
5	1488
6	1337
7	1883
8	2283
9	1335
10	1025
11	1139
12	1477
13	1203
14	1311
15	1299

Number of web server requests per hour for 1 month

Reading TSV File

```
import scipy as sp
import sys

data = sp.genfromtxt(sys.argv[1], delimiter='\t')

print data[:10]
print data.shape

x = data[:,0]
y = data[:,1]

print x[:10]
print y[:10]
```

Reading delimited data file

Getting values of columns into arrays

Displaying First 10 Key-Val Pairs From TSV File

```
$ python read_data.py web_traffic.tsv
[[  1.00000000e+00   2.27200000e+03]
 [  2.00000000e+00              nan]
 [  3.00000000e+00   1.38600000e+03]
 [  4.00000000e+00   1.36500000e+03]
 [  5.00000000e+00   1.48800000e+03]
 [  6.00000000e+00   1.33700000e+03]
 [  7.00000000e+00   1.88300000e+03]
 [  8.00000000e+00   2.28300000e+03]
 [  9.00000000e+00   1.33500000e+03]
 [  1.00000000e+01   1.02500000e+03]]
(743, 2)
[  1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]
[ 2272.    nan  1386.  1365.  1488.  1337.  1883.  2283.  1335.  1025.]
```

Cleaning NAN Values

```
>>> y = np.array([10, 20, np.nan, 40, 50])
>>> x = np.array([1, 2, 3, 4, 5])
>>> x[~sp.isnan(y)]
array([1, 2, 4, 5])
```

Cleaning NAN Values from the Data

```
data = sp.genfromtxt(sys.argv[1], delimiter='\t')

x = data[:,0] ## hours
y = data[:,1] ## server hits

if sp.sum(sp.isnan(y)) > 0:
    x = x[~sp.isnan(y)]
    y = y[~sp.isnan(y)]
```

Checking if there are nan values and getting rid of non-values.

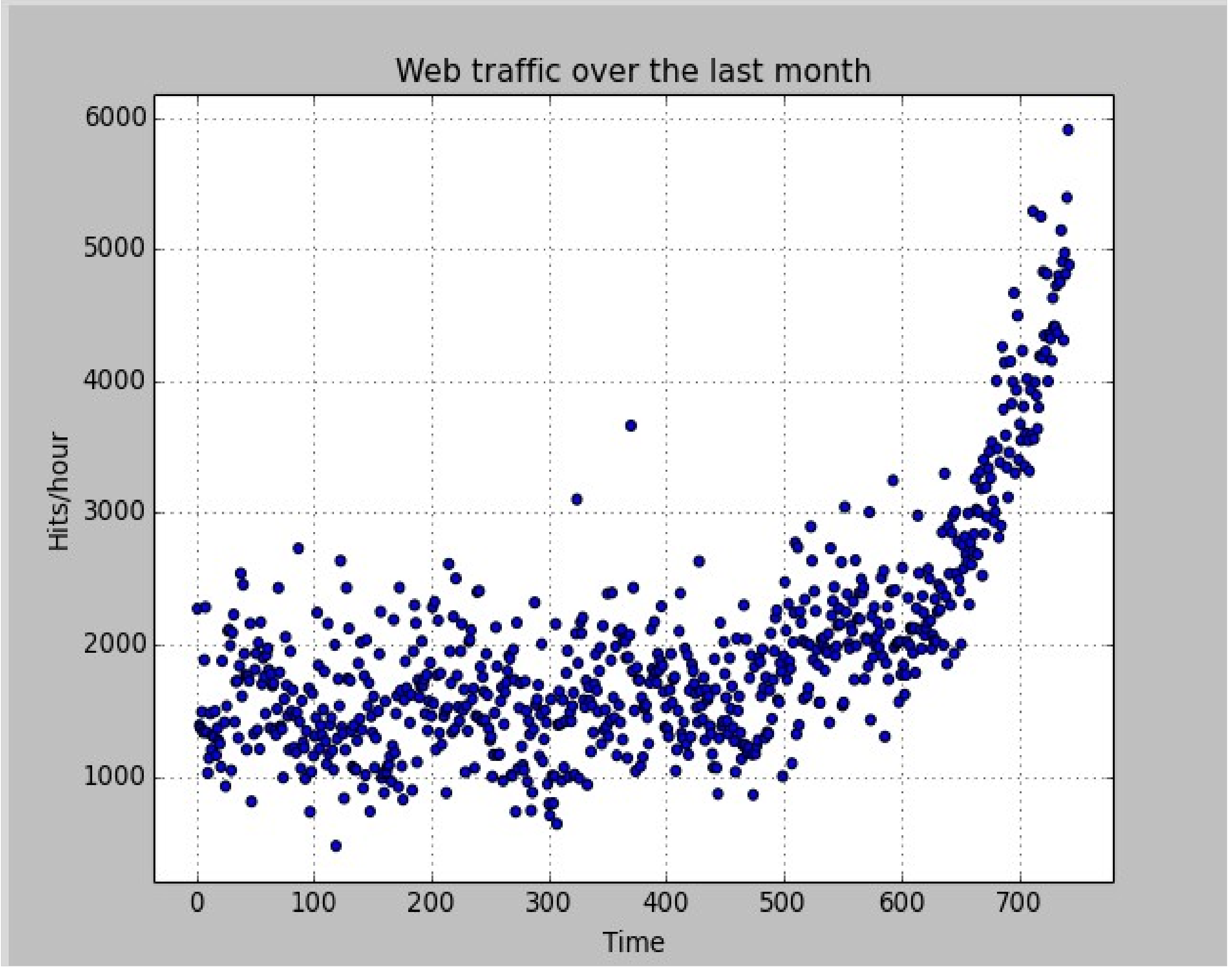
Scatter Plotting Data

```
import scipy as sp
import matplotlib.pyplot as plt
data = sp.genfromtxt('web_traffic.tsv', delimiter='\t')

x = data[:,0]
y = data[:,1]

x = x[~sp.isnan(y)]
y = y[~sp.isnan(y)]

plt.scatter(x, y)
plt.title('Web traffic over the last month')
plt.xlabel('Time')
plt.ylabel('Hits/hour')
plt.autoscale(tight=True)
plt.grid()
plt.show()
```



source in scatter_plot.py

Two Basic Questions & One Comment

- Q1: Which model fits the data best?
- Q2: How well does the model extrapolate into the future?
- Comment: Data modeling is more of an art than a science

Curve Fitting

- Curve fitting is a method of constructing curves that best fit series of data points
- Interpolation is a method of constructing new data points within a set of known data points
- Extrapolation is a method of constructing new data points outside a set of known data points
- Regression analysis is a method of constructing curves that best fit data points and assigning uncertainty to each found curve

Calculating Model Error

A model is a function of 1 argument, because we want to map hours to server hits

An integer array with hour numbers, e.g., 1, 2, 3, 4, 5...

An integer array with real sever hits for the hour numbers in x, e.g. 202, nan, 443, 145, ...

```
def error(f, x, y):
    return sp.sum((f(x) - y) ** 2)
```

Error is the sum of the squared differences between the model predictions, i.e. f(x), and the real data, i.e., y

Polyfitting: Modeling Data with N-Degree Polys

- If we have 2D data, polyfitting is a reasonable first approach
- We can start by modeling the data points with lines (1^{st} deg polys)
- Then we can go on to parabolas (2^{nd} degree polys)
- Then we can go to cubics (3^{rd} degree polys)
- Higher degree polys can also be tried

SciPy's POLYFIT() Function

array with x values

array with y values

poly degree (e.g., 1 for a line)

```
pcoeffs, error, rank, sv, rcond = sp.polyfit(x, y, pd, full=True)
```

array of poly coefficients

error of the found poly curve

Interpreting Line Model Parameters

Py

```
pcoeffs, err, rank, sv, rcond = sp.polyfit(x, y, 1, full=True)
print('Model coeffs: %s' % pcoeffs)
```

Output

```
Model coeffs: [   2.59619213  989.02487106]
```

Line Equation

$$f(x) = 2.59619213x + 989.02487106$$

Obtaining Line Model Function

array with x values

array with y values

poly degree

```
f1 = sp.poly1d(sp.polyfit(x, y, 1))
```

Comparing Errors

Py

```python
pcoeffs, err, rank, sv, rcond = sp.polyfit(x, y, 1, full=True)
f1 = sp.poly1d(sp.polyfit(x, y, 1))
print('Model coeffs: %s' % pcoeffs)
print('Model error: %s' % err)
print('Linear error: %s' % error(f1, x, y))
```

Output

```
Model coeffs: [   2.59619213  989.02487106]
Model error: [  3.17389767e+08]
Linear error: 317389767.34
```

Linear Interpolation: Generating Line Values

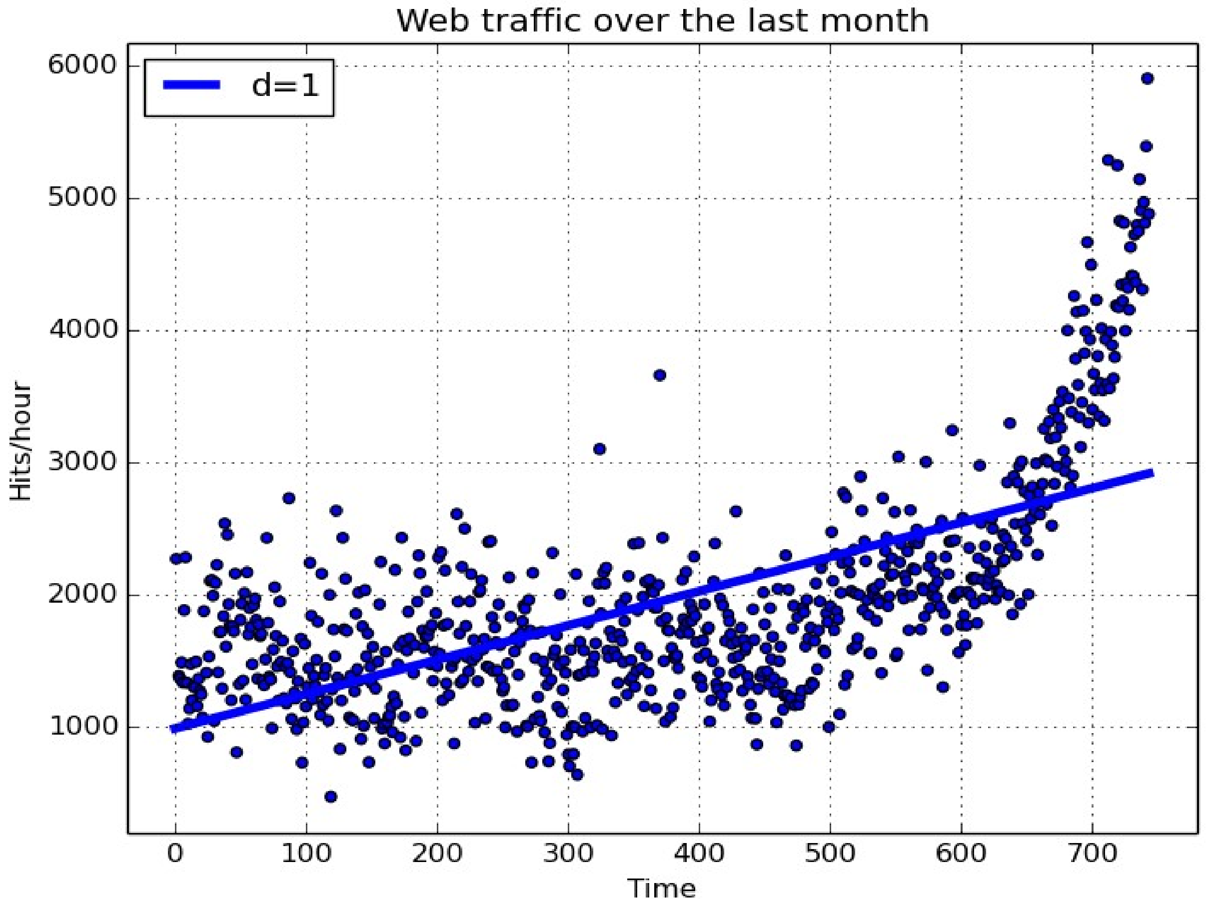
return an array of n values in from start to stop

```
sp.linspace(start, stop, n)
```

```
>>> sp.linspace(0, 1, 2)
array([ 0.,  1.])
>>> sp.linspace(0, 1, 3)
array([ 0. ,  0.5,  1. ])
>>> sp.linspace(0, 1, 4)
array([ 0.        ,  0.33333333,  0.66666667,  1.        ])
```

Plotting Line Model on Data

```
plt.scatter(x, y)
plt.title('Web traffic over the last month')
plt.xlabel('Time')
plt.ylabel('Hits/hour')
plt.autoscale(tight=True)
plt.grid()
xvals = sp.linspace(0, x[-1], 1000)
plt.plot(xvals, f1(xvals),  linewidth=4)
plt.legend(['d=%d' % f1.order], loc='upper left')
plt.show()
```



source in web_data_poly_deg1_fit.py

Obtaining Quadratic Model

Py

```
f1 = sp.poly1d(sp.polyfit(x, y, 1))
f2 = sp.poly1d(sp.polyfit(x, y, 2))
print('Linear model error: %s' % error(f1, x, y))
print('Quadratic model error: %s' % error(f2, x, y))
```

Output

```
Linear model error: 317389767.34
Quadratic model error: 179983507.878
```

Interpreting Quadratic Model

Py

```
pcoeffs, err, rank, sv, rcond = sp.polyfit(x, y, 2, full=True)
print('Quadratic model coeffs: %s' % pcoeffs)
```

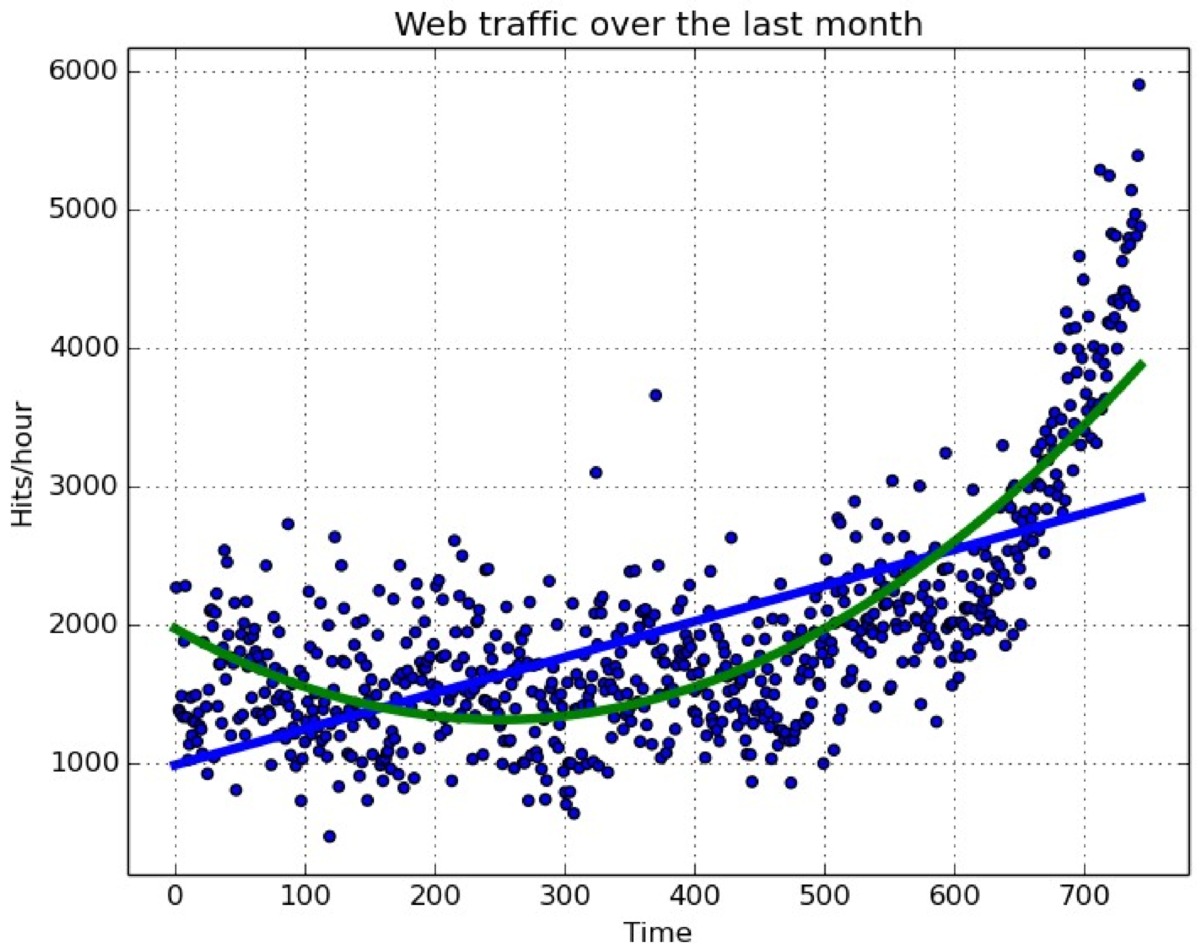
Output

```
Quadratic model coeffs: [  1.05322215e-02  -5.26545650e+00   1.97476082e+03]
```

```
f(x) = 0.0105322215 * x**2 - 5.26545650 * x + 1974.76082
```

Plotting Two Models on Data

```
plt.scatter(x, y)
plt.title('Web traffic over the last month')
plt.xlabel('Time')
plt.ylabel('Hits/hour')
plt.autoscale(tight=True)
plt.grid()
xvals = sp.linspace(0, x[-1], 1000)
plt.plot(xvals, f1(xvals), linewidth=4, color='b')
plt.plot(xvals, f2(xvals), linewidth=4, color='g')
plt.show()
```



source in web_data_poly_deg2_fit.py

Obtaining Higher Degree Polynomial Models

```
f1 = sp.poly1d(sp.polyfit(x, y, 1))
f2 = sp.poly1d(sp.polyfit(x, y, 2))
f3   = sp.poly1d(sp.polyfit(x, y, 3))
f10  = sp.poly1d(sp.polyfit(x, y, 10))
f100 = sp.poly1d(sp.polyfit(x, y, 100))
```

Computing Error for All Models

Error d=1: 317389767.34

Error d=2: 179983507.878

Error d=3: 139350144.032

Error d=10: 121942326.364

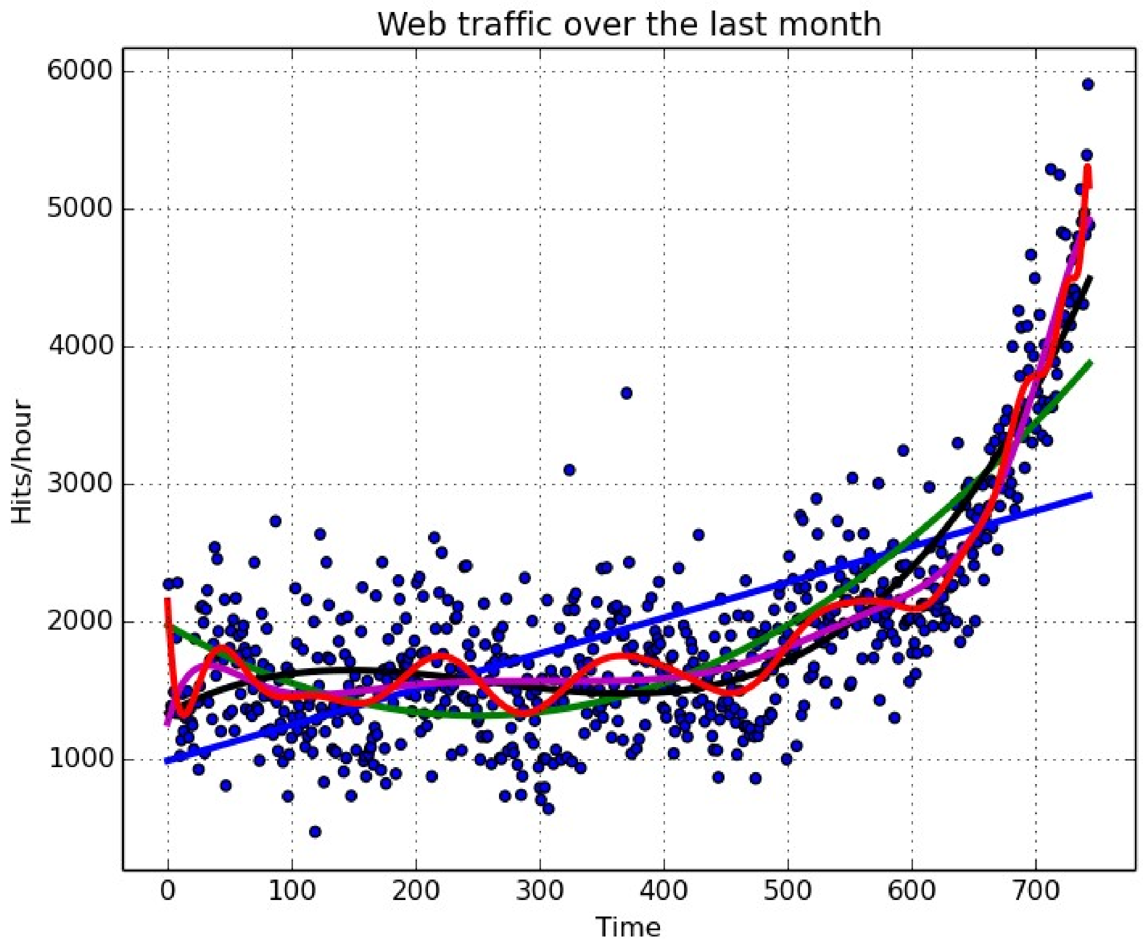
Error d=100: 109452416.424

Overfitting & Underfitting

- Overfitting occurs when a model matches the data too closely: this is typically revealed by a wildly oscillating behavior of a graph
- Underfitting occurs when a model matches the data too losely: this is typically revealed by too smooth a graph that misses many point clusters

Plotting All Models on Data

```
xvals = sp.linspace(0, x[-1], 1000)
plt.plot(xvals, f1(xvals), linewidth=3, color='b')
plt.plot(xvals, f2(xvals), linewidth=3, color='g')
plt.plot(xvals, f3(xvals), linewidth=3, color='k')
plt.plot(xvals, f10(xvals), linewidth=3, color='m')
plt.plot(xvals, f100(xvals), linewidth=3, color='r')
plt.show()
```



source in web_data_poly_deg3_10_100_fit.py

Predicting When The Server Will Get 100,000 Hits

solve the poly

```
>>> print(f2)
        2
0.01053 x - 5.265 x + 1975
>>> print(f2 - 100000)
        2
0.01053 x - 5.265 x - 9.803e+04
>>> from scipy.optimize import fsolve
>>> max_val = fsolve(f2 - 100000, 800)
>>> max_val
array([ 3310.95905176])
>>> max_val[0]/(7*24)
19.708089593829524
```

Quadratic model predicts 100,000 server hits in about 19.7 weeks (5 months)

References

W. Richert & L. Coelho. “Building ML Systems with Python”, Ch. 1, Pack, 2013.