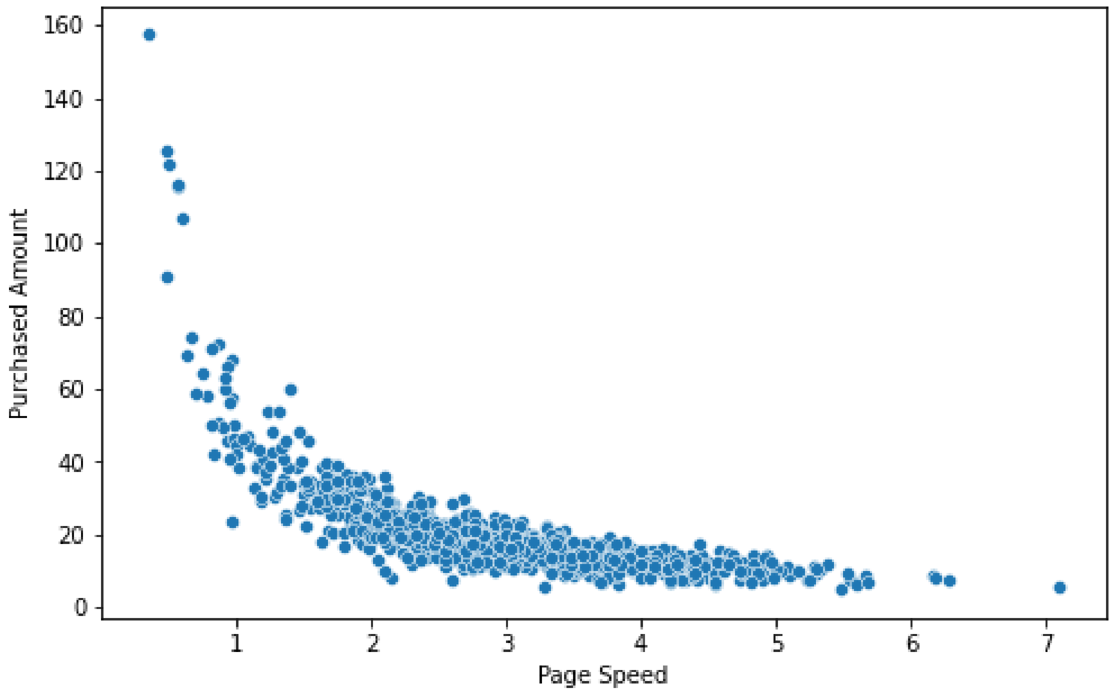
PolynomialRegression

September 2, 2021

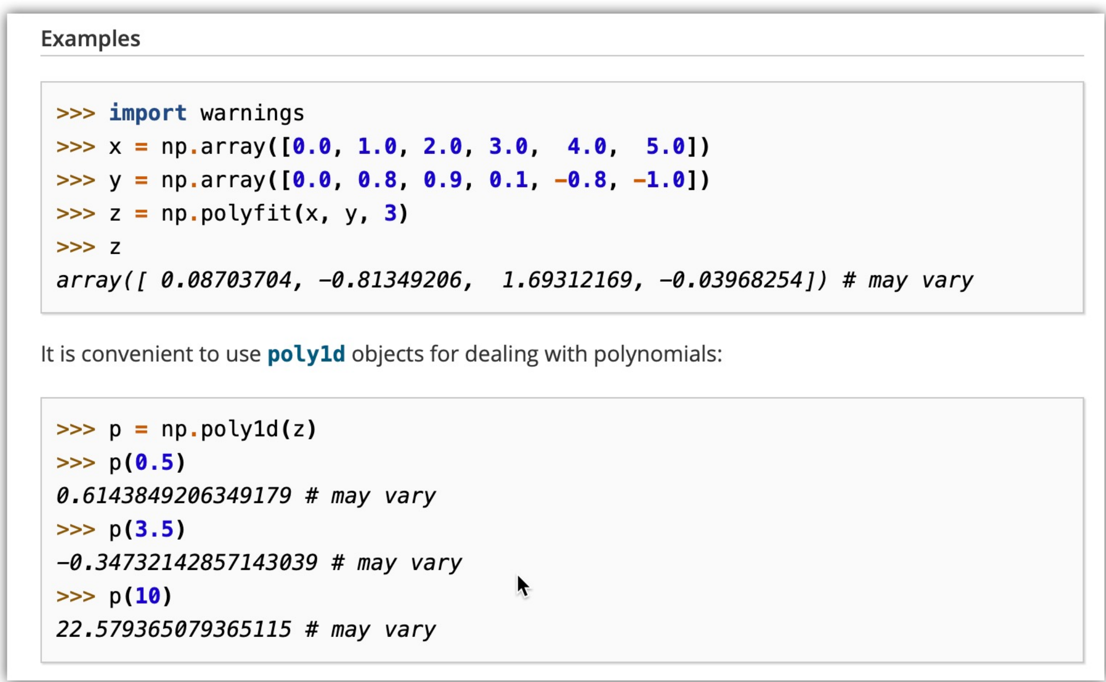
[4]:
```
%matplotlib inline
from pylab import *
import numpy as np
import seaborn as sns
import pandas as pd

np.random.seed(2)
#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
#generating normally distrubuted random data

#Normally distributed dat with mean 3 i.e data centered arround 3 with standard␣
↪deviation 1 and
#total 1000 data point
pageSpeeds = np.random.normal(3.0, 1.0, 1000)
#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) / pageSpeeds
```

[47]:
```
plt.figure(figsize=(8,5))
sns.scatterplot(x=pageSpeeds,y=purchaseAmount)
plt.xlabel('Page Speed') #x label
plt.ylabel('Purchased Amount') #y label
plt.show()
```




```
[56]: from IPython.display import Image
      Image(filename='/Users/subhasish/Downloads/Unknown1.jpg')
```

[56]:



Examples

```
>>> import warnings
>>> x = np.array([0.0, 1.0, 2.0, 3.0,  4.0,  5.0])
>>> y = np.array([0.0, 0.8, 0.9, 0.1, -0.8, -1.0])
>>> z = np.polyfit(x, y, 3)
>>> z
array([ 0.08703704, -0.81349206,  1.69312169, -0.03968254]) # may vary
```

It is convenient to use **poly1d** objects for dealing with polynomials:

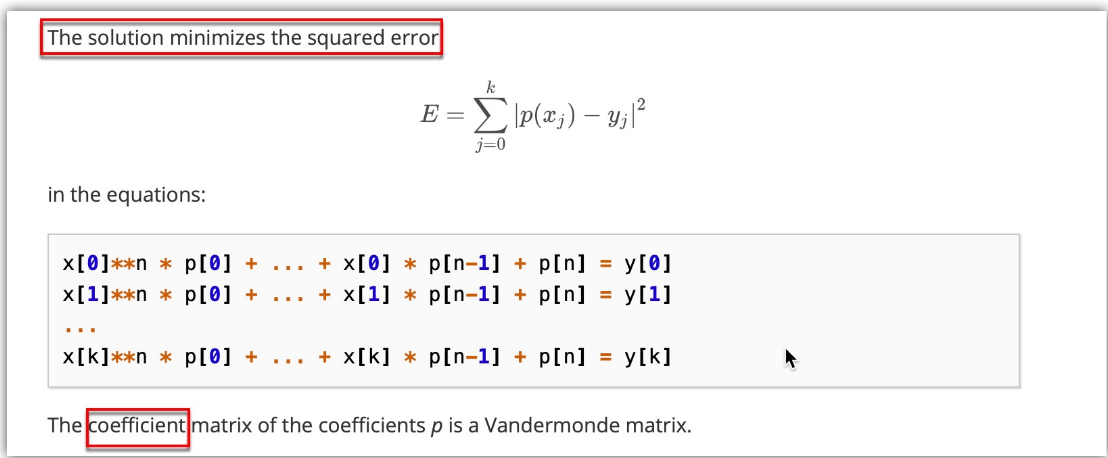
```
>>> p = np.poly1d(z)
>>> p(0.5)
0.6143849206349179 # may vary
>>> p(3.5)
-0.34732142857143039 # may vary
>>> p(10)
22.579365079365115 # may vary
```

[48]:
```python
#Fit a polynomial p(x) = p[0] * x**deg + ... + p[deg] of degree deg to points␣
↪(x, y).
#Returns a vector of coefficients p that minimises the squared error in the␣
↪order deg, deg-1, … 0.

#Parameters
#    xarray_like, shape (M,)
#        x-coordinates of the M sample points (x[i], y[i]).
#    yarray_like, shape (M,) or (M, K)
#        y-coordinates of the sample points.
#    degint
#        Degree of the fitting polynomial

#Returns
#        Polynomial coefficients, highest power first.
z=np.polyfit(x,y,4)
z
```

[48]:
```
array([   0.54005597,   -8.85641318,   52.25378374, -135.34422815,
        147.6050662 ])
```

[57]:
```python
from IPython.display import Image
Image(filename='/Users/subhasish/Downloads/Unknown2.jpg')
```

[57]:



The solution minimizes the squared error

$$E = \sum_{j=0}^{k} |p(x_j) - y_j|^2$$

in the equations:

```
x[0]**n * p[0] + ... + x[0] * p[n-1] + p[n] = y[0]
x[1]**n * p[0] + ... + x[1] * p[n-1] + p[n] = y[1]
...
x[k]**n * p[0] + ... + x[k] * p[n-1] + p[n] = y[k]
```

The coefficient matrix of the coefficients *p* is a Vandermonde matrix.

[49]:
```python
#Example of poly1d
#The polynomial’s coefficients, in decreasing powers,
#For example, poly1d([1, 2, 3]) returns an object that represents Xsqure2+2X+1
p = np.poly1d([1, 2, 3])
print(np.poly1d(p))
```

```
   2
```

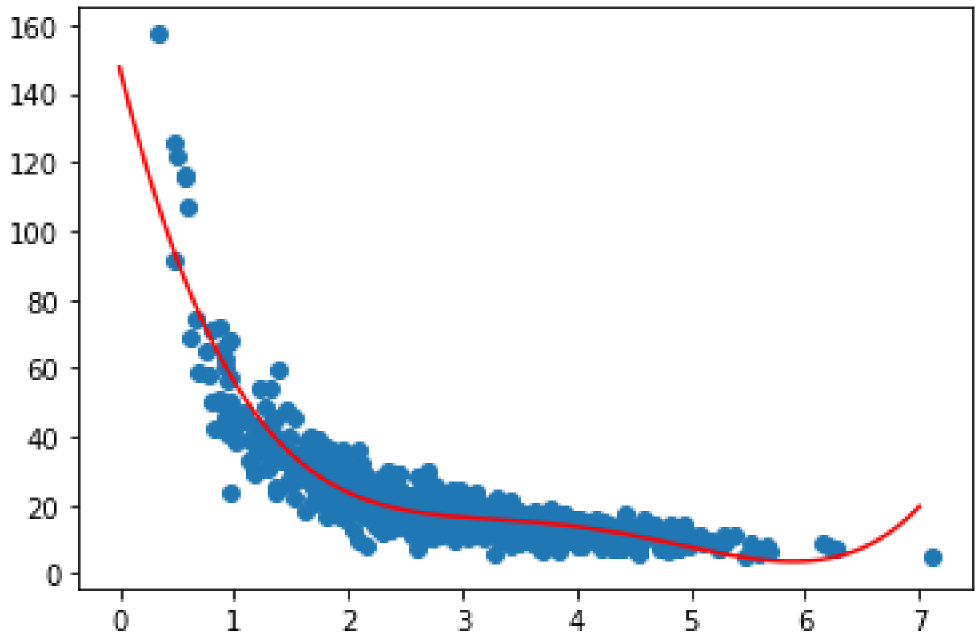
1 x + 2 x + 3

[50]:
```
#It is convenient to use poly1d objects for dealing with polynomials:

p=np.poly1d(z)
p(0.5)
```

[50]: 91.92309990686252

[51]:
```
#numpy has a handy polyfit function we can use, to let us construct an␣
↪nth-degree polynomial model
#of our data that minimizes squared error.
#Let's try it with a 4th degree polynomial:

#x, y axis is array of pageSpeed and Purchase Amount
x = np.array(pageSpeeds)
y = np.array(purchaseAmount)

#np.polyfit(x,y,4) -> we want 4th Degree of Polynomial Fit to this data

#The polynomial’s coefficients, in decreasing powers,
#For example, poly1d([1, 2, 3]) returns an object that represents Xsqure2+2X+1
p4=np.poly1d(np.polyfit(x,y,4))
```

[52]:
```
#We'll visualize our original scatter plot, together with a plot of our␣
↪predicted values using the polynomial
#for page speed times ranging from 0-7 seconds:

import matplotlib.pyplot as plt

#100 data point between 0-7 sec
xp = np.linspace(0, 7, 100)
plt.scatter(x, y)
#P4(xp) -> Predicted Y value for each xp value
#plotting x and y [x=xp] and [y=p4(xp)]
plt.plot(xp, p4(xp), c='r')
plt.show()
```



[45]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y, p4(x))
r2
```

[45]: 0.8293766396303073

[ ]:
```

```