EASC2410 Lecture 15

Statistics Basics: Covariance, Correlation, Regression and Curve Fitting

Dr. Binzheng Zhang
Department of Earth Sciences

Review of Lecture 14:

- Properties of Normal distributions
  - Mode, Mean, Median, Standard Deviation
  - Continuous distributions
- How to use Seaborn to visualize distributions of a dataset

In Lecture 15, you will learn:

- learn basic concepts of correlations, covariance, regression
- use Python to fit linear models for data sets
- use Python to fit non-linear models for data sets
- understanding the quality of the fitting

Bivariate Statistics

Until now, we've worked with data that does not depend on other data. But there are many examples in the Earth Sciences where we are interested in the **dependence between two types of data**. This is called *bivariate statistics*

For example, the distance from the ridge crest versus age gives you a spreading rate. The depth in a sediment core versus age gives you a sedimentation rate. The ratio of the radioactive form of carbon, 14C, to a stable form, 12 C, is a function of the age of the material being dated.

In another example, we already saw that the difference in travel times of the P and S seismic waves is related to distance from the source to the receiver. These examples rely on the use of bivariate statistics to get at the desired quantities.

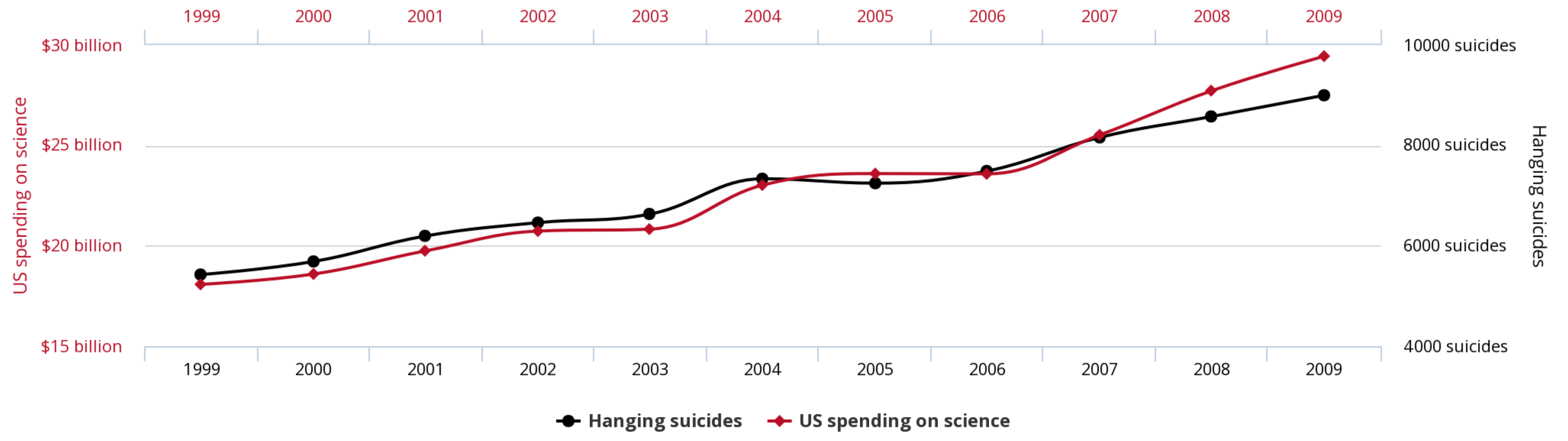
Correlation: one of the **most widely used** — and **widely misunderstood** — **statistical concepts**.

Definition: "correlation" refers to a mutual relationship or association between quantities.

Importance:

- Correlation can help in predicting one quantity from another
- Correlation can (but often does not, as we will see in some examples below) indicate the presence of a causal relationship
- Correlation is used as a basic quantity and foundation for many other modeling techniques

US spending on science, space, and technology
correlates with
Suicides by hanging, strangulation and suffocation



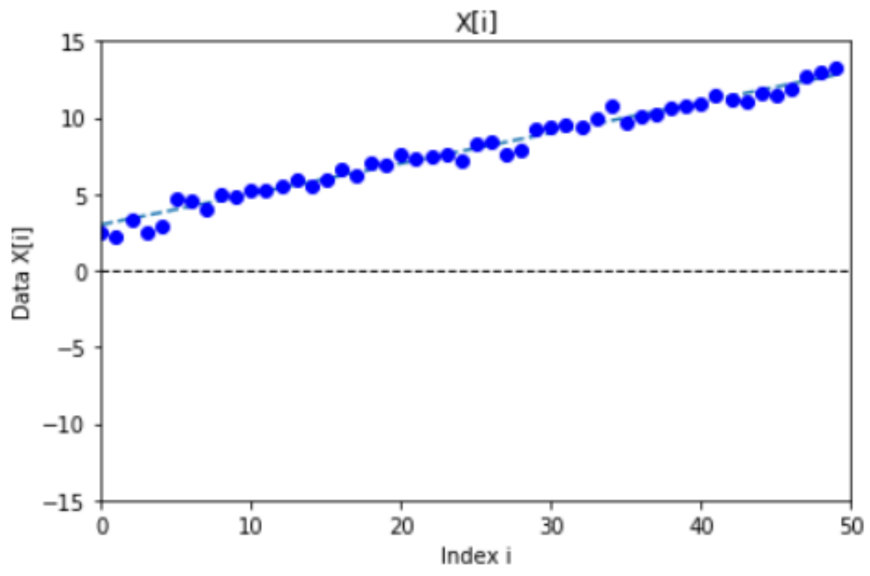
tylervigen.com

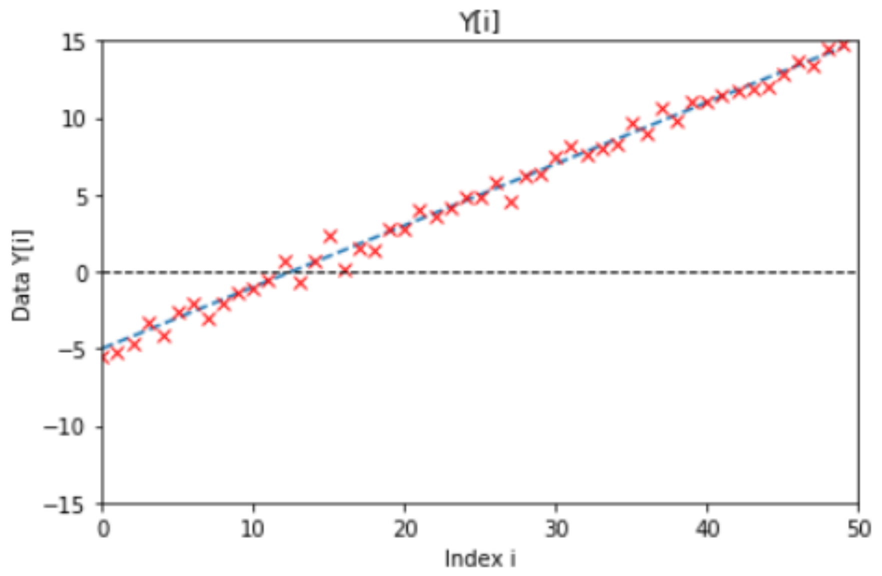
Again, Correlation is not necessarily Causality

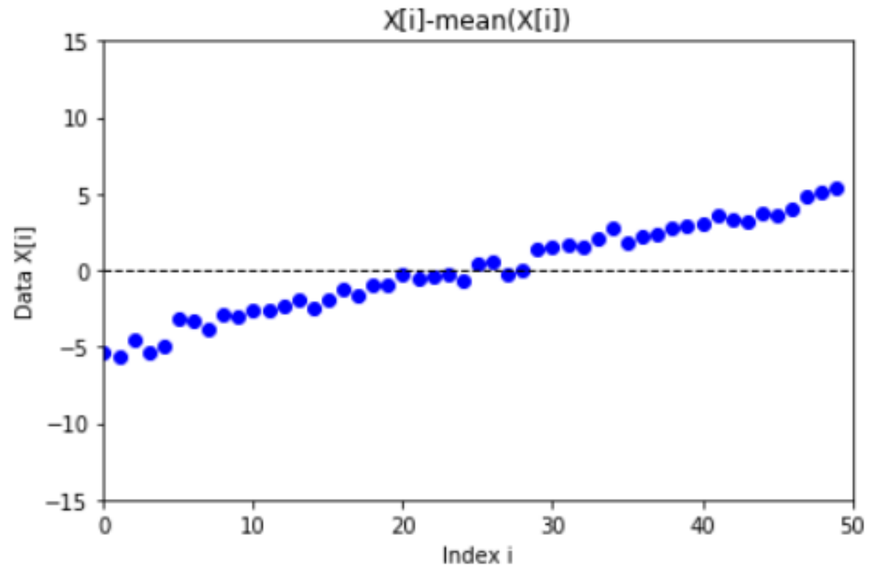
Covariance between two variables (datasets)

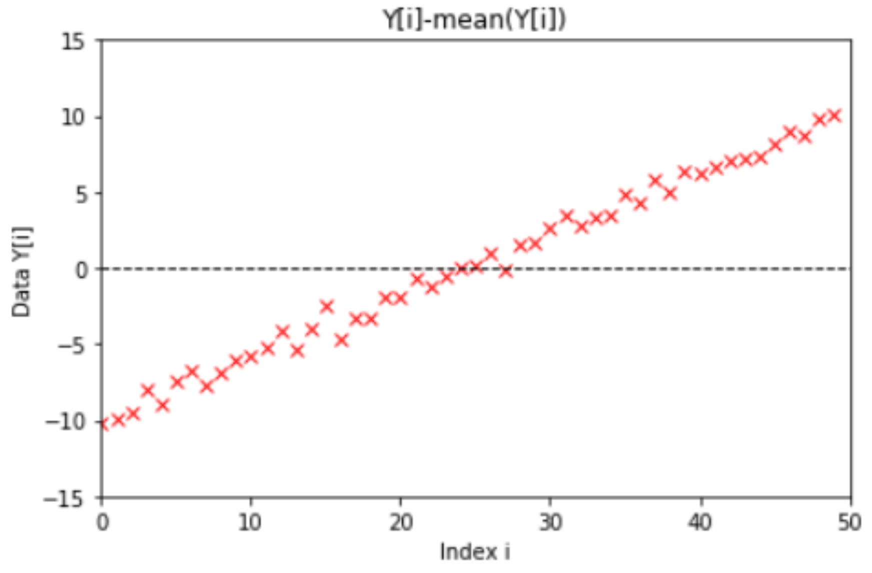
Definition: "covariance" in data science is basically a statistical measure of *association between two datasets X and Y*.

$$Cov(X,Y) = \frac{1}{N}\Sigma_{i=1}^{N}(X[i]-\mu_X)(Y[i]-\mu_Y)$$

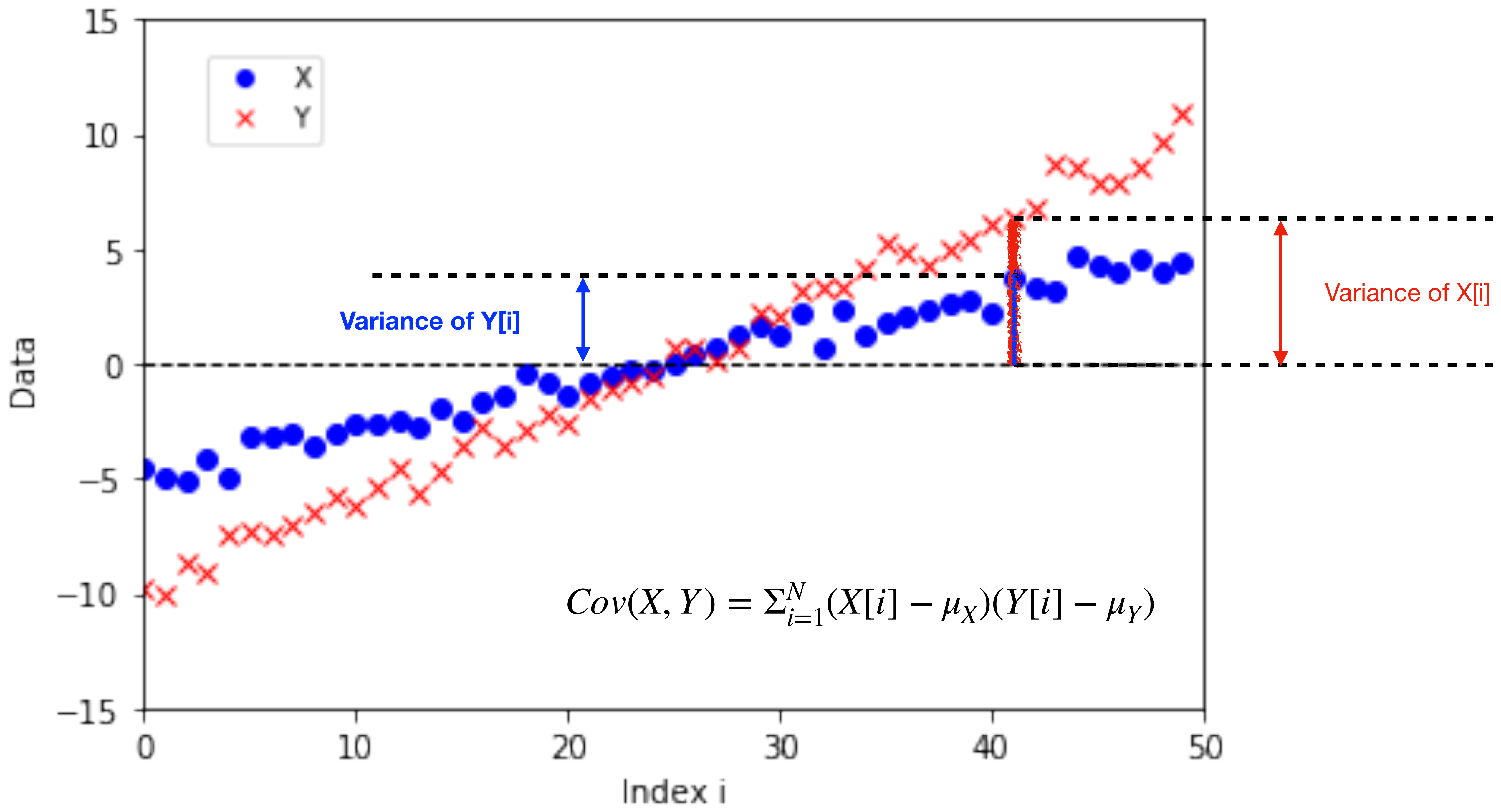








Covariance between two variables (datasets)

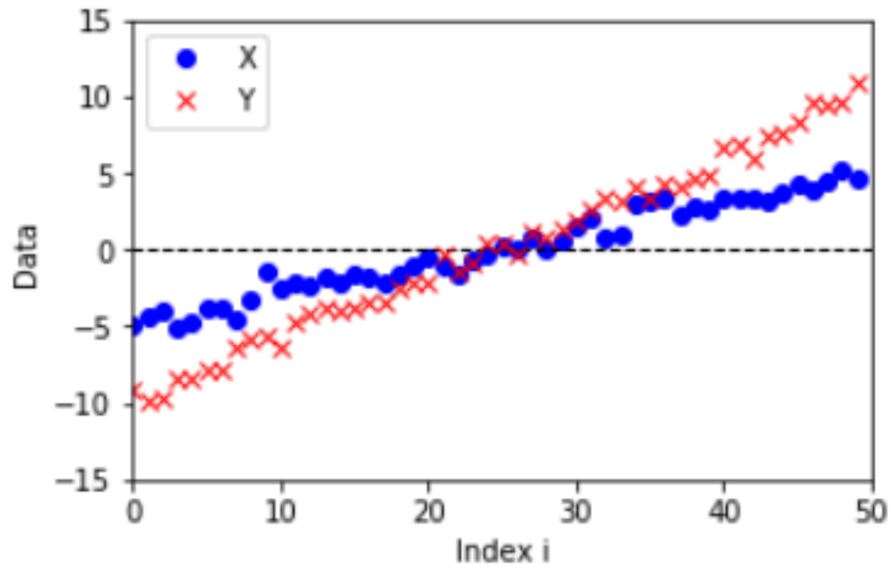


$$Cov(X, Y) = \Sigma_{i=1}^{N}(X[i] - \mu_X)(Y[i] - \mu_Y)$$

The mathematical meaning of Covariance between the dataset X and Y is very clear. In this dataset, the variance of X[i] and Y[i] either increase together (for i>25) or decrease together (for i<=25). In that sense, the two data sets show the behavior of co-variate, or one is dependent on the other.

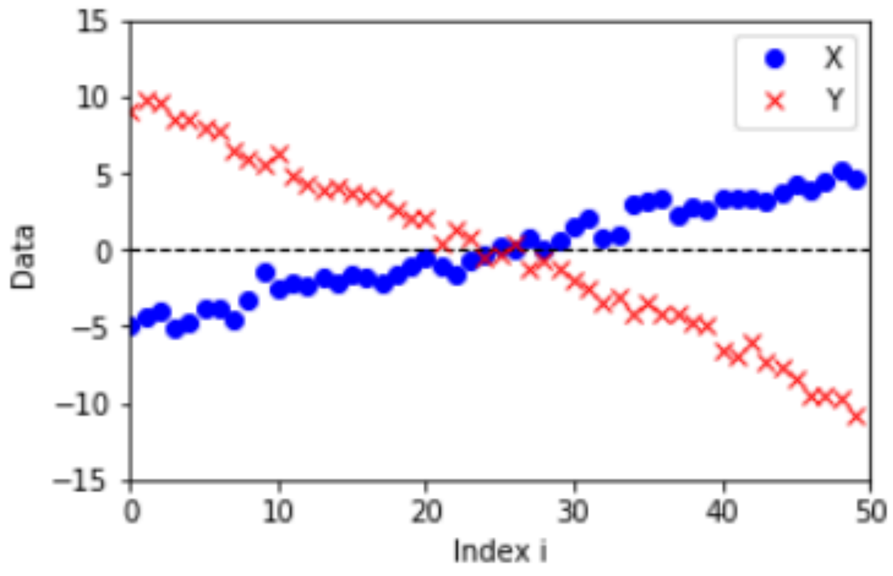
Covariance between two variables (datasets)

Cov(X,Y)>0



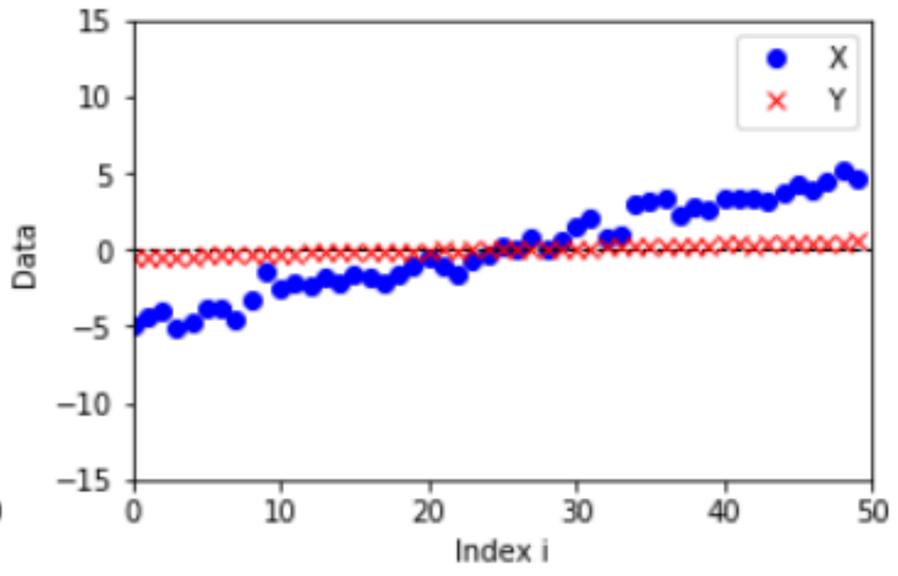
X and Y vary in the same direction

Cov(X,Y)<0



X and Y vary in the opposite direction

Cov(X,Y)=0



X and Y almost independent

The problem with Covariance

The problem with covariance is that it keeps the scale of the variables X and Y, and therefore can take on any value. This makes interpretation difficult and comparing covariances to each other impossible. For example, Cov(X, Y) = 5.2 and Cov(Z, Q) = 3.1 tell us that these pairs are positively associated, but it is difficult to tell whether the relationship between X and Y is stronger than Z and Q without looking at the means and distributions of these variables. This is where correlation becomes useful — by standardizing covariance by some measure of variability in the data, it produces a quantity that has intuitive interpretations and consistent scale.

This is why “correlation coefficient” trumps

Correlation coefficients between two variables (Pearson)

Pearson is the most widely used correlation coefficient. Pearson correlation measures the linear association between continuous variables. In other words, this coefficient quantifies the degree to which a relationship between two variables can be described by a line. Remarkably, while correlation can have many interpretations, the same formula developed by Karl Pearson over **120 years ago** is still the most widely used today.

The original formula for correlation, developed by Pearson himself, uses raw data and the means of two variables X and Y

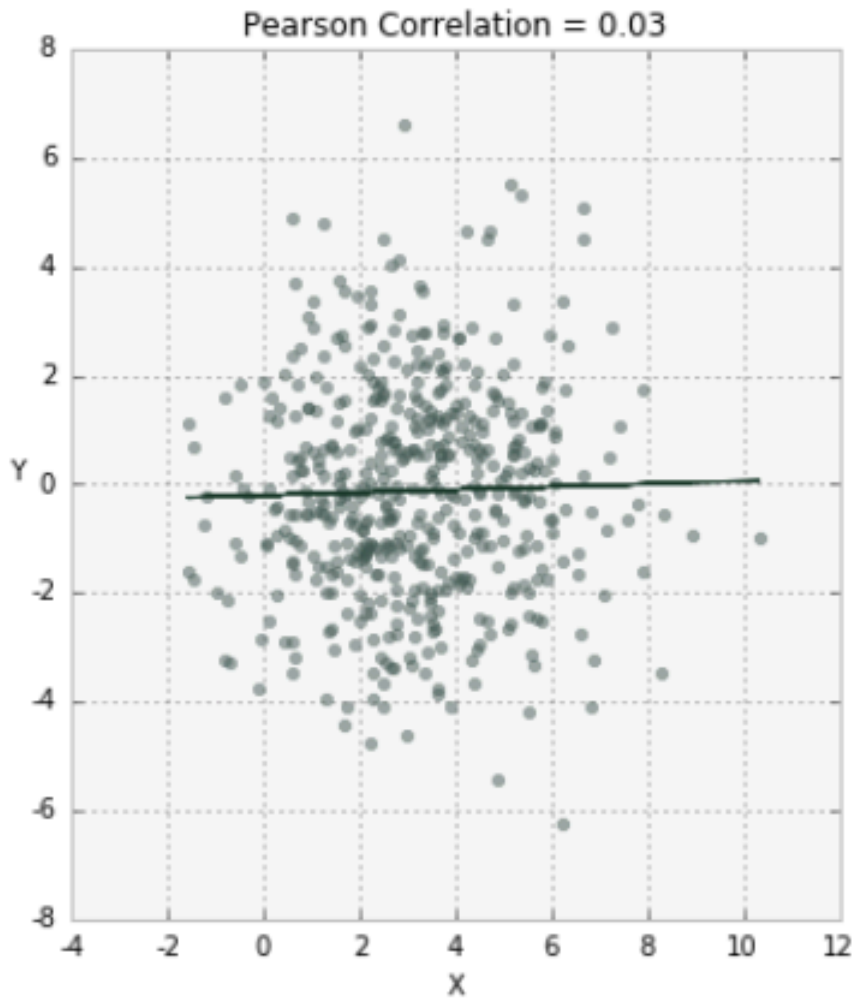
$$\rho_{X,Y} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} = \frac{\Sigma_{i=1}^{N}(X[i]-\mu_X)(Y[i]-\mu_Y)}{\sqrt{\Sigma_{i=1}^{N}(X[i]-\mu_X)^2 \Sigma_{i=1}^{N}(Y[i]-\mu_Y)}^{2}}$$

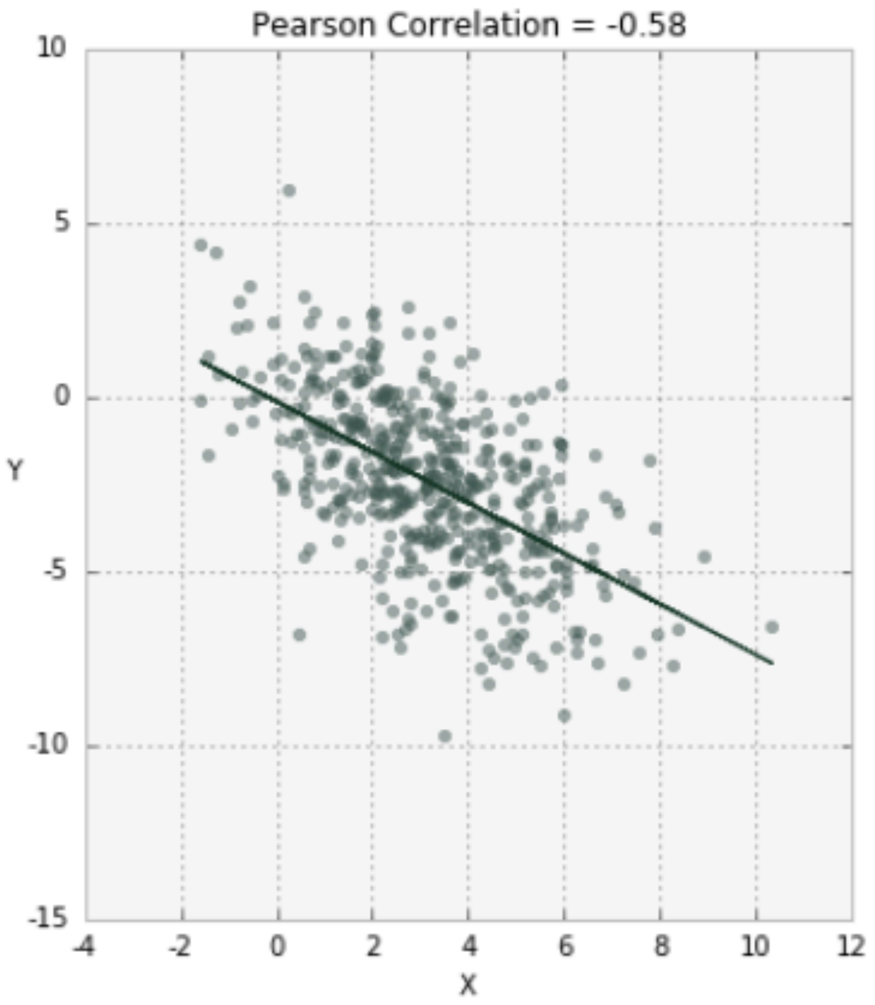
In this formulation, raw observations are centered by subtracting their means and **re-scaled** by a measure of standard deviations. In other words, $\rho_{X,Y}$ is called "**Dimensionless**". Now we can compare between different data sets.

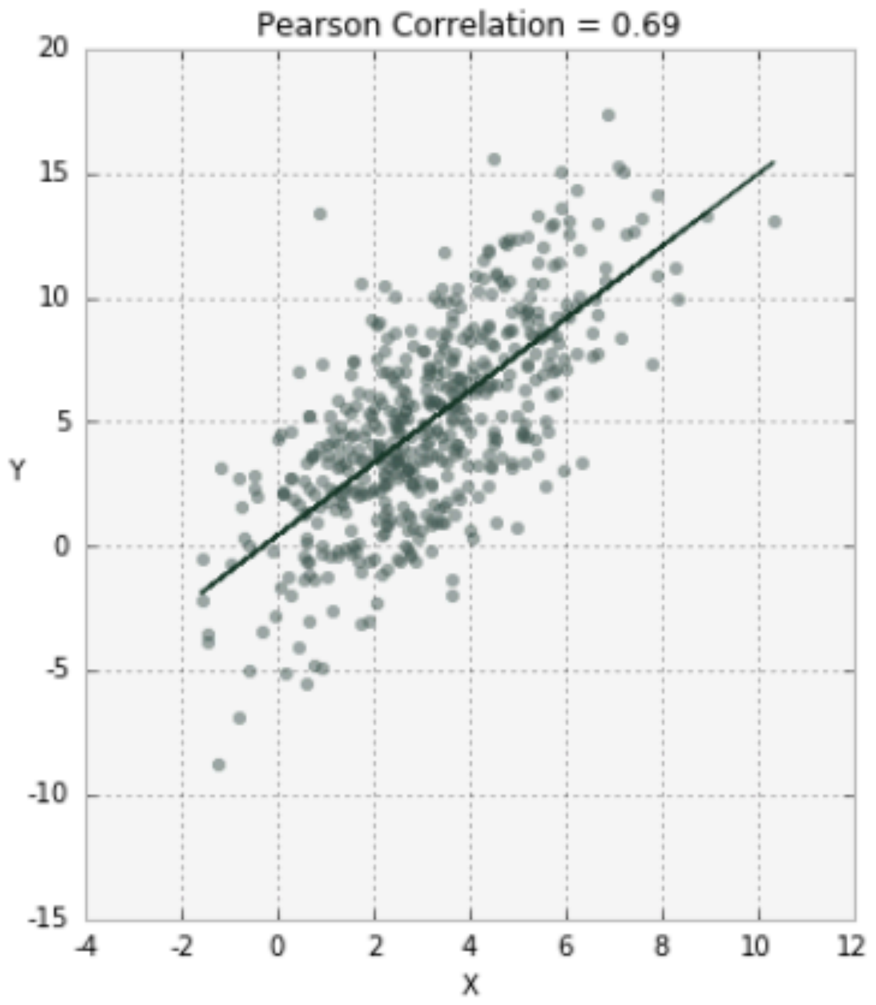
The figure below shows three examples of Pearson correlation. The closer ρ is to 1, the more an increase in one variable associates with an increase in the other. On the other hand, the closer ρ is to -1, the increase in one variable would result in decrease in the other. Note that if X and Y are independent, then ρ is close to 0, but not vice versa! In other words, Pearson correlation can be small even if there is a strong relationship between two variables. We will see shortly how this can be the case.

Correlation coefficients between two variables (Pearson)

Examples:





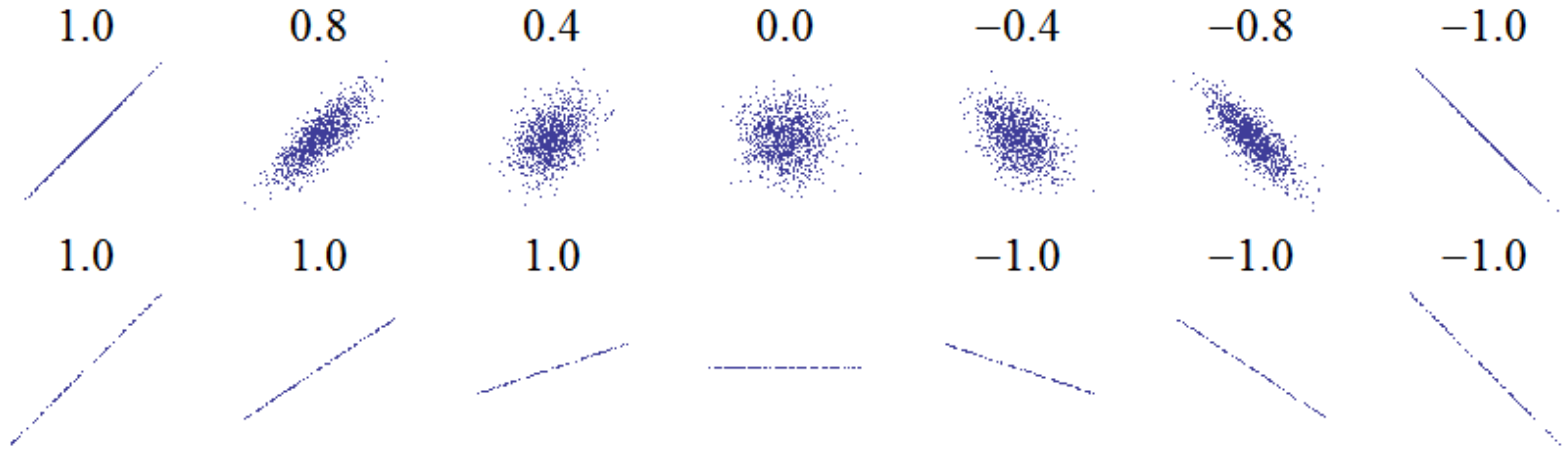


zero correlation
$p=0$

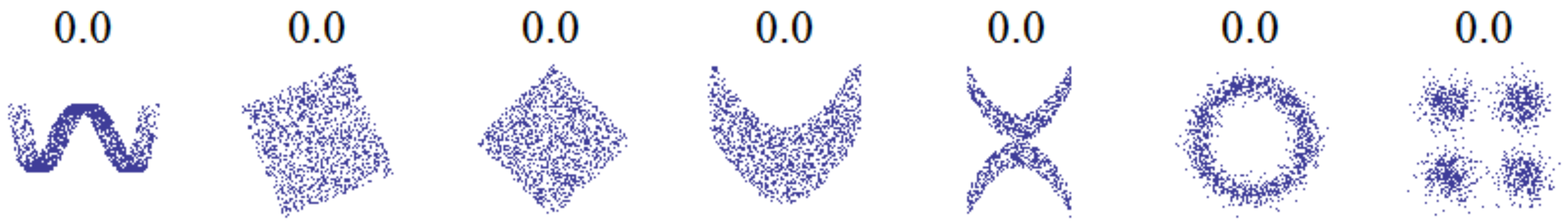
negative correlation
$p<0$

positive correlation
$p>0$

Correlation coefficients between two variables (non-linear cases)



The Pearson correlation coefficient describe only the linear dependence of two statistical variables

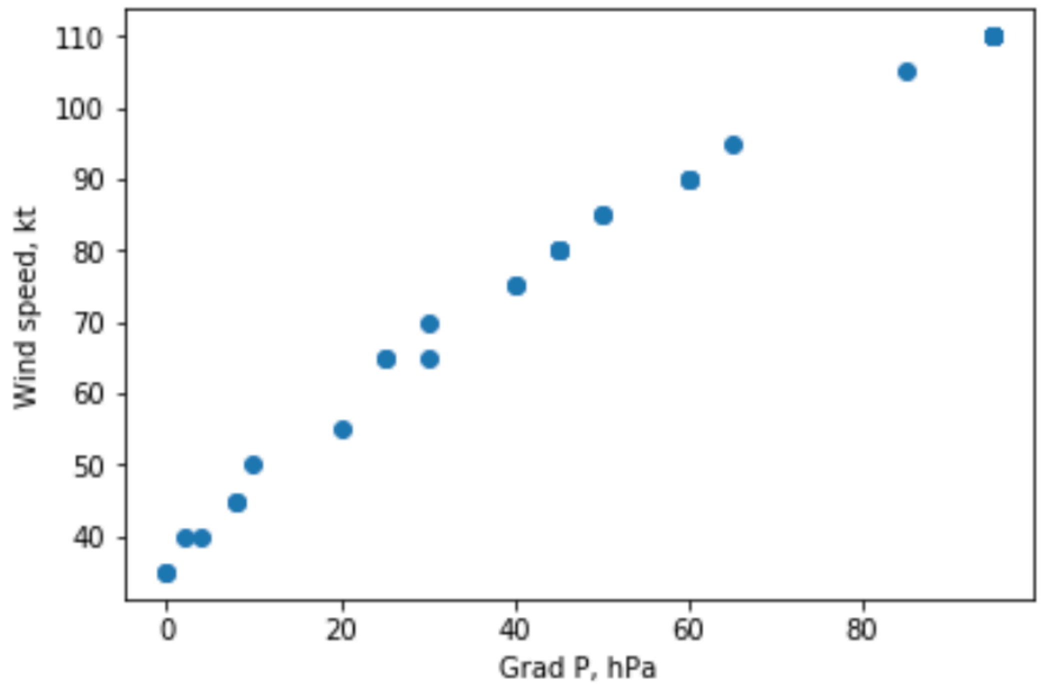


Cannot tell you anything about non-linear patterns. Solution: non-linear correlations, or machine learning

Calculate Correlation coefficients using Pandas

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

file = "datasets/mangkhut.xlsx" # file name and location
mg = pd.read_excel(file) # load file into pandas data frame

print(mg.head()) # print the head()

mg.rename(columns={'Pressure (hPa)':'Pres', 'Wind (kt)':'Wind'},inplace = True) # rename column names

mg = mg[mg.Wind>0] # filtering all the zero wind speeds

mg['gradP'] = mg['Pres'].max() - mg['Pres'] # create a new column which is the pressure gradient

plt.scatter(mg.gradP, mg.Wind) # make a plot showing the relationship between gradP and Wind
plt.xlabel('Grad P, hPa')
plt.ylabel('Wind speed, kt')
plt.show()

c1 = mg['Wind'].corr(mg['gradP'])
print('The correlation coefficient between "Wind Speed" and "gradP" is', c1)
```

```
   Year  Month  Day  Hour   Lat   Long  Pressure (hPa)  Wind (kt)  Class
0  2018      9    6    12  11.8  170.2            1008          0      2
1  2018      9    6    18  12.0  169.2            1008          0      2
2  2018      9    7     0  12.0  167.8            1008          0      2
3  2018      9    7     6  12.3  166.6            1006          0      2
4  2018      9    7    12  12.7  165.4            1000         35      3
The correlation coefficient between "Wind Speed" and "gradP" is 0.98919618097888
```



Syntax: .corr( )

The .corr( ) function

The .corr( ) function computes correlation coefficients between column data sets in Pandas data frames. For example let’s take a look at the Mangkhut data (data frame named mg):

```
mg.head()
```

	Year	Month	Day	Hour	Lat	Long	Pres	Wind	Class	gradP
4	2018	9	7	12	12.7	165.4	1000	35	3	0
5	2018	9	7	18	13.0	163.9	1000	35	3	0
6	2018	9	8	0	13.6	162.3	998	40	3	2
7	2018	9	8	6	14.3	160.6	996	40	3	4
8	2018	9	8	12	14.5	159.2	992	45	3	8

Let’s try the .corr( ) function for the data frame called mg:

```
mg.corr()
```

	Year	Month	Day	Hour	Lat	Long	Pres	Wind	Class	gradP
Year	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Month	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Day	NaN	NaN	1.000000	-0.139432	0.828020	-0.992509	-0.369698	0.388957	0.396614	0.369698
Hour	NaN	NaN	-0.139432	1.000000	-0.040382	0.043045	-0.012507	-0.006051	-0.025402	0.012507
Lat	NaN	NaN	0.828020	-0.040382	1.000000	-0.814642	0.174553	-0.143231	0.056052	-0.174553
Long	NaN	NaN	-0.992509	0.043045	-0.814642	1.000000	0.403475	-0.425452	-0.436836	-0.403475
Pres	NaN	NaN	-0.369698	-0.012507	0.174553	0.403475	1.000000	-0.989196	-0.718066	-1.000000
Wind	NaN	NaN	0.388957	-0.006051	-0.143231	-0.425452	-0.989196	1.000000	0.802666	0.989196
Class	NaN	NaN	0.396614	-0.025402	0.056052	-0.436836	-0.718066	0.802666	1.000000	0.718066
gradP	NaN	NaN	0.369698	0.012507	-0.174553	-0.403475	-1.000000	0.989196	0.718066	1.000000

We’ve got lot’s of NaNs in the correlation coefficient, Why?

The .corr( ) function

Because the standard deviation for ‘Year’, ‘Month’ is zero! (recall the correlation coefficient formula)

Let’s drop the ‘Year’, ‘Month’, ‘Day’, ‘Hour’ columns and try the .corr( ) function again:

```python
mg.drop(['Year', 'Month', 'Day', 'Hour'], axis=1).corr()
```

	Lat	Long	Pres	Wind	Class	gradP
Lat	1.000000	-0.814642	0.174553	-0.143231	0.056052	-0.174553
Long	-0.814642	1.000000	0.403475	-0.425452	-0.436836	-0.403475
Pres	0.174553	0.403475	1.000000	-0.989196	-0.718066	-1.000000
Wind	-0.143231	-0.425452	-0.989196	1.000000	0.802666	0.989196
Class	0.056052	-0.436836	-0.718066	0.802666	1.000000	0.718066
gradP	-0.174553	-0.403475	-1.000000	0.989196	0.718066	1.000000

The “axis” option:

axis : *{0 or ‘index’, 1 or ‘columns’}, default 0*
Whether to drop labels from the index (0 or ‘index’) or columns (1 or ‘columns’).

Recall the .describe( ) function which gives a statistical summary of your data frame:

```python
mg.drop(['Year', 'Month', 'Day', 'Hour'], axis=1).describe()
```

	Lat	Long	Pres	Wind	Class	gradP
count	39.000000	39.000000	39.000000	39.000000	39.000000	39.000000
mean	16.033333	135.987179	944.051282	83.589744	4.641026	55.948718
std	2.885840	16.434434	34.987929	25.827075	0.742938	34.987929
min	12.700000	108.300000	905.000000	35.000000	3.000000	0.000000
25%	14.100000	123.200000	905.000000	65.000000	5.000000	27.500000
50%	14.600000	135.200000	950.000000	85.000000	5.000000	50.000000
75%	17.700000	148.300000	972.500000	110.000000	5.000000	95.000000
max	23.200000	165.400000	1000.000000	110.000000	5.000000	95.000000

Regression and Curve Fitting

Regression

Regression analysis models the relationships between a response variable and one or more predictor variables. Use a regression model to understand how changes in the predictor values are associated with changes in the response mean. You can also use regression to **make predictions** based on the values of the predictors.

Curve Fitting

Curve fitting is the process of specifying the model that provides the best fit to the specific curves in your dataset. Curved relationships between variables are not as straightforward to fit and interpret as linear relationships.

Difference between Regression and Curve Fitting

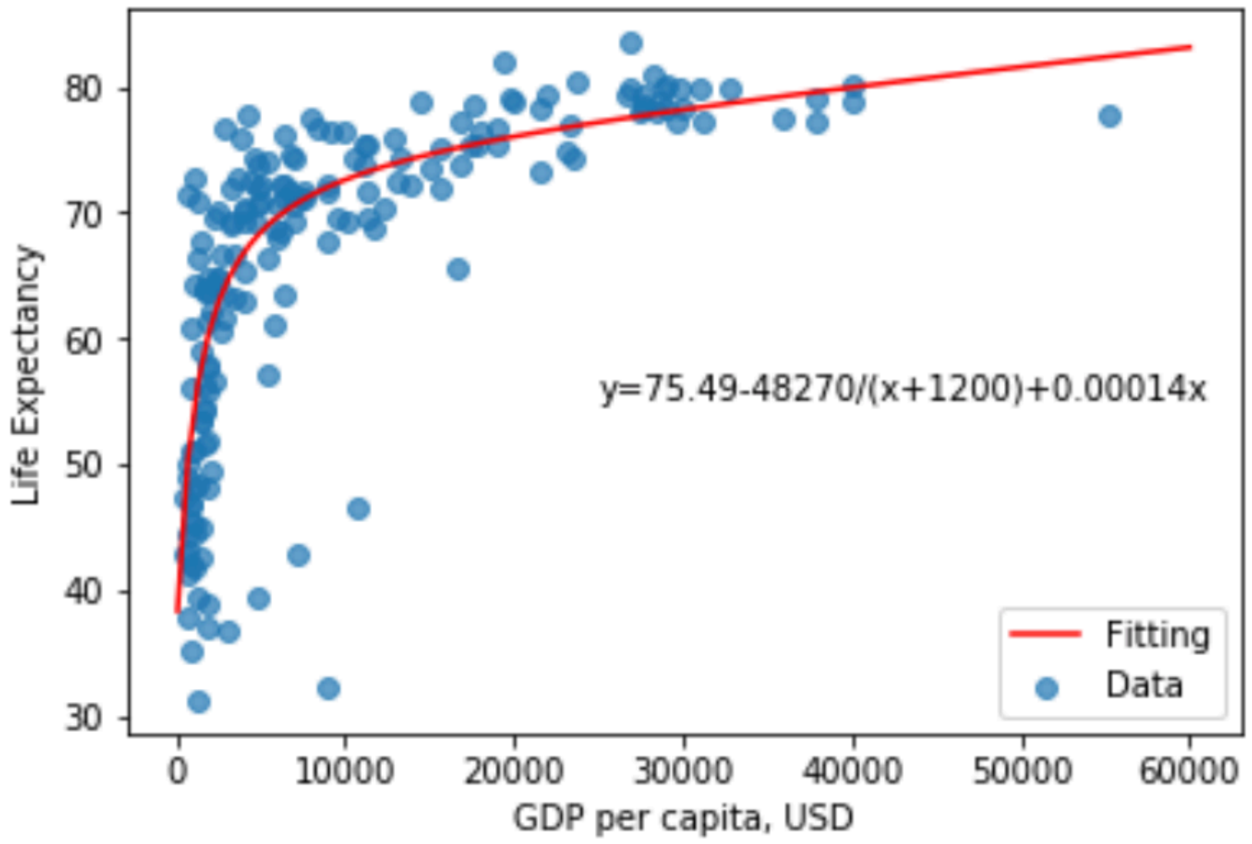
Curve fitting is one of the most common things you'll do as an experimental physicist or pretty much any hard science. You gather a set of data, you visualize it, create a fit and build a model around that fit so you can interpolate. Majority of the time, if not every time, you know exactly what parameters are in the dataset as they correspond to some physical event. Building fits help you extract a mathematical equation that will dictate how the event will act in the future given the parameters are the same. Since you know the parameters (and in the event you know how the event was setup), you can tailor your errors and uncertainties more carefully. This can include human error, instrument error etc…
In the other way around, if you already have a theorised model, you can use curve fitting from experimental data to extract an equation and verify the theory that was derived without any data. This is where theoretical and experimental scientists play together.

Regression is a far more loaded term and has a lot of connections to machine learning. Admittedly, curve fitting also sounds simpler. It's not. Regression analysis is most commonly used in forecasting and building predictions. It deals with the relationship between the independent variable and the dependent variables and **how** the dependent variables change when the independent variable is changed. More often than curve fitting, correlation does not always mean causation in regression analysis. On the more complex side, regression analysis can deal with "messier" and unstructured data (machine learning), but we won't go into that as it's beyond the scope of this course topic.

Recall in-class practice 06

5. Histgram and scatter plots

- In the following cell, you've already given two numpy arrays named **age** and **gdp**, which are the life expectancy and the GDP per capita for quite a few countries. Let's use those two numpy arrays to try some data analysis
- Make a pie plot using pie() to show the distributions of life expectancy - does it make sense at all?
- Make a histgram using hist() with 20 bins to show the distribution of the life expectancy - what do you learn?
- Now make a line plot using **plot(gdp,age)** to show the relationship between GDP per capita and life expectancy
- Does your line plot look nice? How to fix it? (hint: either use markers without lines or sort your numpy array before plotting)
- Now try the scatter() function to show your data points
- change your xscale to be logarithm using xscale('log')
- In the same plot, make the following curve using the following equation $y = 75.49 - \frac{48270}{x+1200} + 0.00014x$. How does this function look like? Note: **x** should be a 1-D numpy array from, say, 500 to 50000, using 100 points (e.g., you can use the linspace() function)



In this example, you used plt.scatter( ) to show the relationship between GDP per capita and life expectancy. Followed by a given function y(x):

$$y(x) = 75.49 - \frac{48270}{x + 1200} - 0.00014x$$

Which “pass” right through the cluster of data points. This is basically a fitted curve for the data set you are showing here.

Let’s learn how to compute such a curve using Python (HW problem).

How to do curve fitting in Python?

The **curve_fit( )** function from the **scipy** module:

```
from scipy.optimize import curve_fit # import function curve_fit
```

Syntax of the **curve_fit( )** function:

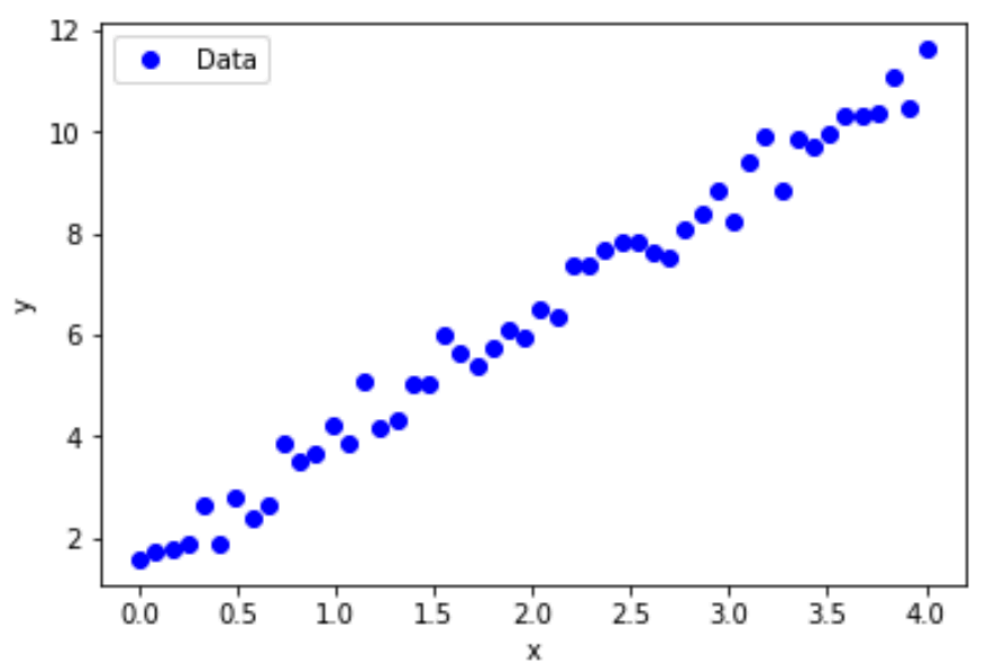
scipy.optimize.curve_fit(f, xdata, ydata, p0=None, sigma=None, absolute_sigma=False, check_finite=True, bounds=(-inf, inf), method=None, jac=None, //kwargs)

a complete documentation is here: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html

Let's try a simple example to learn how to do linear curve fitting, first generate random data distributed along a line

$$y = 2.5x + 1.3$$

```
from scipy.optimize import curve_fit # import function curve_fit

# define a linear function
def func(x, a, b):
    return a *x + b

xdata = np.linspace(0, 4, 50) # x-axis
y = func(xdata, 2.5, 1.3) # data is func(x)
y_noise = 0.5 * np.random.normal(size=xdata.size) # generate
ydata = y + y_noise # add the noise to the ydata
plt.plot(xdata, ydata, 'bo', label='Data')
```

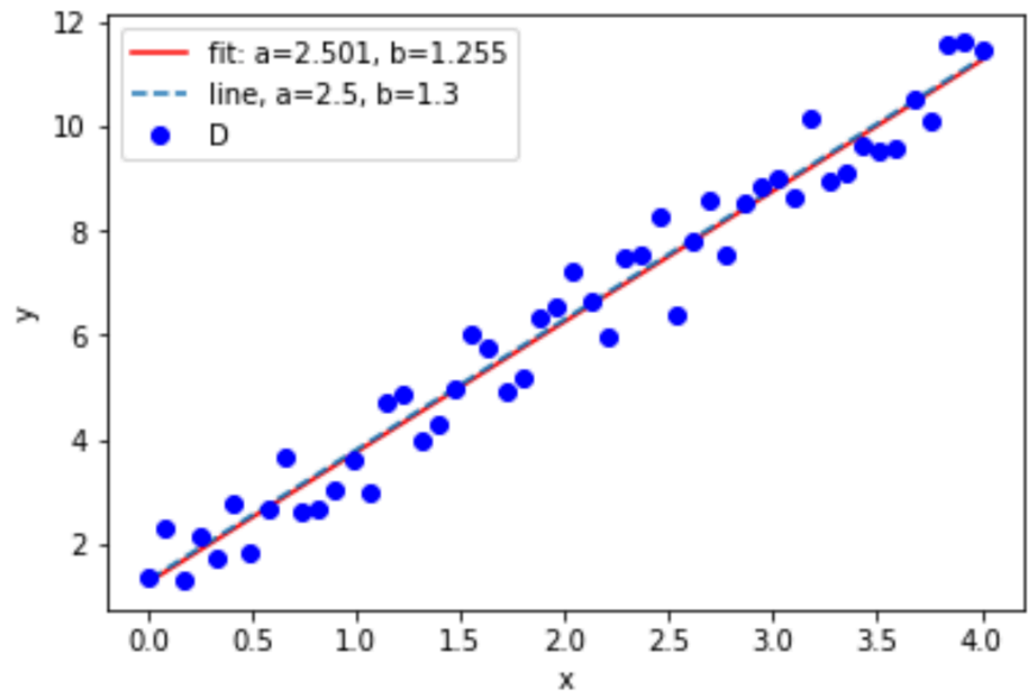


Now let's see if the curve fitting process can recover the line y=2.5x+1.3

Linear curve fitting in Python

Let's call the curve_fit( ) function and feed the linear function, the xdata and ydata as inputs:

```python
popt, pcov = curve_fit(func, xdata, ydata) # curve fitting, popt has the a,b,c calculated by SciPy

plt.plot(xdata, func(xdata, *popt), 'r-', label='fit: a=%5.3f, b=%5.3f' % tuple(popt)) # Plot the results
plt.plot(xdata, y, '--', label='line, a=2.5, b=1.3') # plot the theoretical line
plt.plot(xdata, ydata, 'bo', label='D') # plot the original data

plt.legend()
plt.show()
```



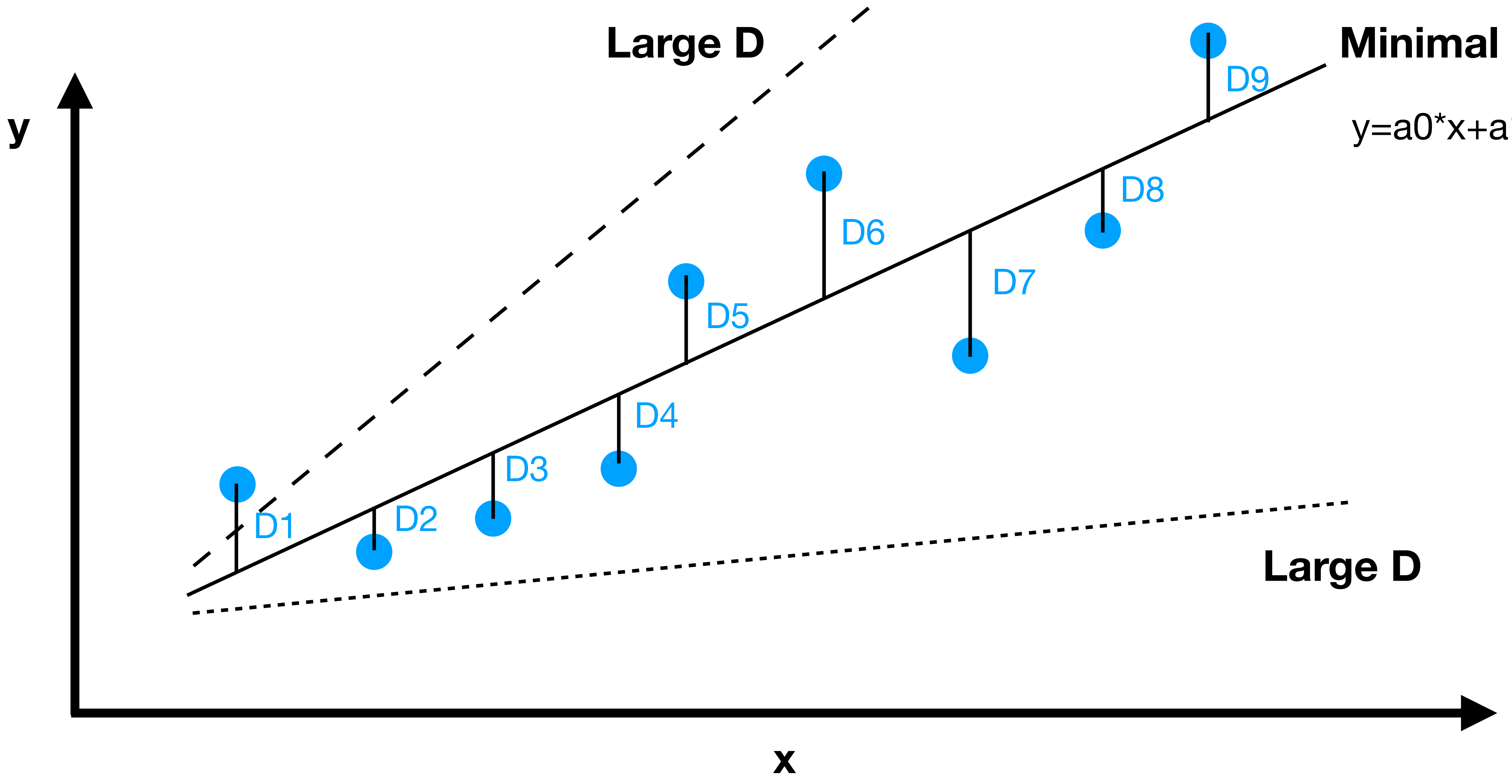
The line is recovered quite well!

```python
print('popt = ',popt)
print('--------------------------------')
print('pcov=')
print(pcov)
print('--------------------------------')
print('According to the fitting results:')
print("a =", popt[0], "+/-", pcov[0,0]**0.5)
print("b =", popt[1], "+/-", pcov[1,1]**0.5)
```

This is the results and quality of the fitting:

```
popt =  [2.50122391 1.25467496]
-------------------------------
pcov=
[[ 0.00464825 -0.0092965 ]
 [-0.0092965   0.02504364]]
-------------------------------
According to the fitting results:
a = 2.5012239055196464 +/- 0.06817809186401011
b = 1.254674962636234З +/- 0.15825183931755307
```

How's curve fitting done: Least square fitting to a data set



$$D = (D1^2 + D2^2 + D3^2 + D4^2 + D5^2 + D6^2 + D7^2 + D8^2 + D9^2)^{1/2}$$

The idea of least square fitting is simple: find a line that minimizes D

$$a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - \left(\sum x_i\right)^2} \qquad a_0 = \frac{\sum y_i - a_1 \sum x_i}{n} = \bar{y} - a_1\bar{x}$$

How to interpret the quality (goodness) of the curve fitting

```python
popt, pcov = curve_fit(func, xdata, ydata) # curve fitting, popt has the a,b,c calculated by SciPy

```

After calling the curve_fit( ) function, we obtain two variables: **popt** and **pcov**:

```python
print('popt = ',popt)
print('--------------------------------')
print('pcov=')
print(pcov)
print('--------------------------------')
print('According to the fitting results:')
print("a =", popt[0], "+/-", pcov[0,0]**0.5)
print("b =", popt[1], "+/-", pcov[1,1]**0.5)
```

```
popt =  [2.50122391 1.25467496]
--------------------------------
pcov=
[[ 0.00464825 -0.0092965 ]
 [-0.0092965   0.02504364]]
--------------------------------
According to the fitting results:
a = 2.5012239055196464 +/- 0.06817809186401011
b = 1.2546749626362343 +/- 0.15825183931755307
```

popt is basically the optimal parameters **a** and **b** in your function defined as **func:**

- **a** = popt[0] and **b** = popt[1]
- these are the “best” values that Python thinks based on some criteria like the least square rule

```python
# define a linear function
def func(x, a, b):
    return a *x + b
```

pcov is basically the covariance matrix of all the **a** and **b** values Python tried during the curve fitting:

- **it’s a 2x2 matrix**

$$pcov = \begin{bmatrix} cov(a,a) & cov(a,b) \\ cov(b,a) & cov(b,b) \end{bmatrix}$$

recall:

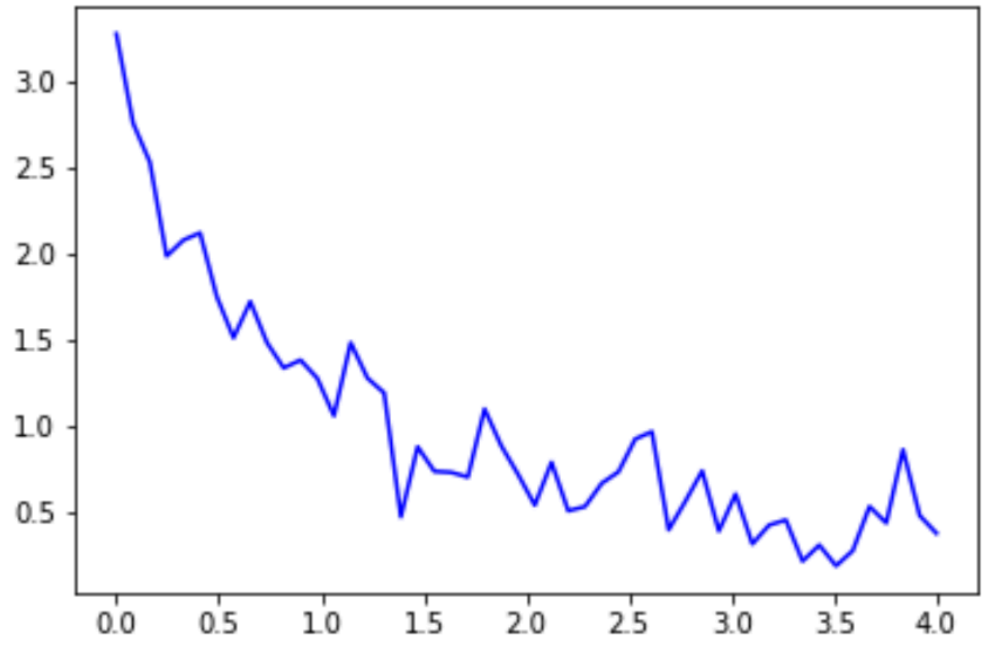
$$Cov(X,Y) = \frac{1}{N}\Sigma_{i=1}^{N}(X[i]-\mu_X)(Y[i]-\mu_Y)$$

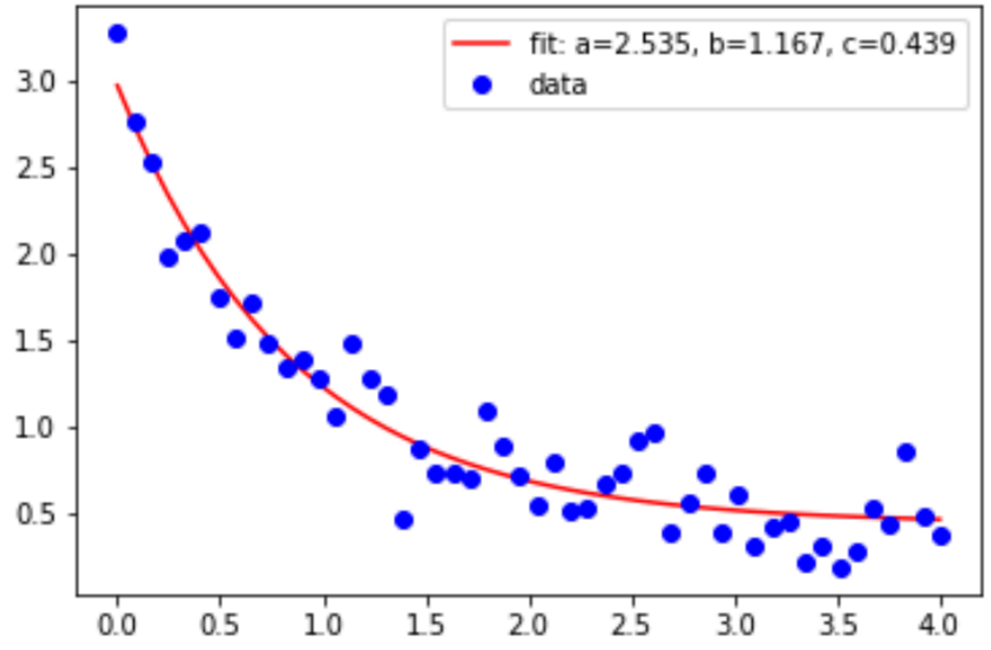
- the cov(a,a) and cov(b,b) basically tells you the standard deviation of the fitting parameters”

Nonlinear curve fitting in Python

The curve fitting process for a non-linear function is basically the same as the linear example we've seen. Let's first generate an exponential function with random perturbations

```python
# first let's define an exponential function
def func(x, a, b, c):
    return a * np.exp(-b * x) + c

xdata = np.linspace(0, 4, 50) # x-axis
y = func(xdata, 2.5, 1.3, 0.5) # data is func(x)

y_noise = 0.2 * np.random.normal(size=xdata.size) # generate some noise at each data point
ydata = y + y_noise # add the noise to the ydata

plt.plot(xdata, ydata, 'b-', label='data')
```

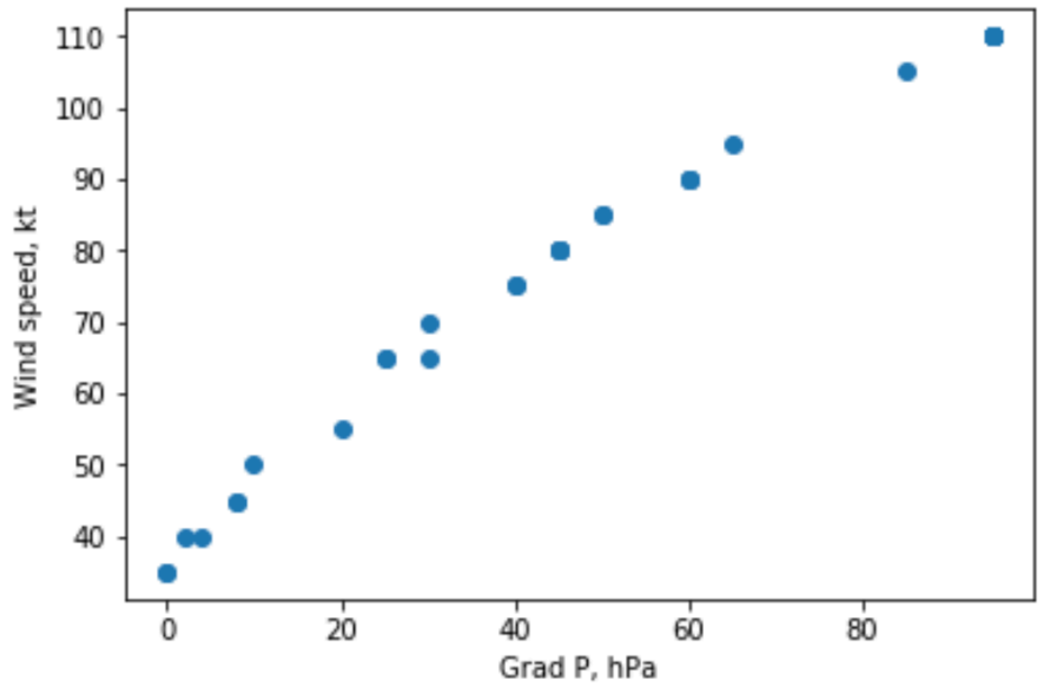




```python
popt, pcov = curve_fit(func, xdata, ydata) # curve fitting, popt has the a,b,c calculated by SciPy

plt.plot(xdata, func(xdata, *popt), 'r-', label='fit: a=%5.3f, b=%5.3f, c=%5.3f' % tuple(popt)) # Plot

plt.plot(xdata, ydata, 'bo', label='data') # now plot the dat set

plt.legend()
```

Put it all together: The Typhoon Manghkut

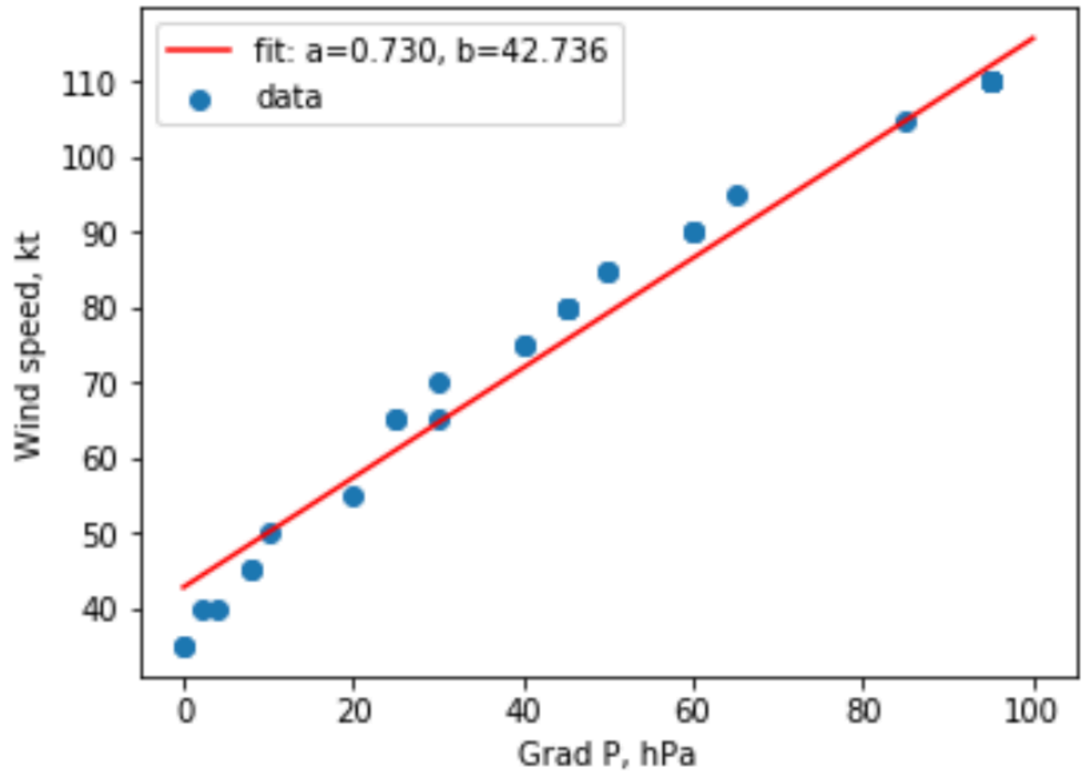
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

file = "datasets/mangkhut.xlsx" # file name and location
mg = pd.read_excel(file) # load file into pandas data frame

print(mg.head()) # print the head()

mg.rename(columns={'Pressure (hPa)':'Pres', 'Wind (kt)':'Wind'},inplace = True) # rename column names

mg = mg[mg.Wind>0] # filtering all the zero wind speeds

mg['gradP'] = mg['Pres'].max() - mg['Pres'] # create a new column which is the pressure gradient

plt.scatter(mg.gradP, mg.Wind) # make a plot showing the relationship between gradP and Wind
plt.xlabel('Grad P, hPa')
plt.ylabel('Wind speed, kt')
plt.show()

c1 = mg['Wind'].corr(mg['gradP'])
print('The correlation coefficient between "Wind Speed" and "gradP" is', c1)
```



Now let's compute the relationship between GradP and Wind Speed

Linear Fit (y=ax+b) Results:

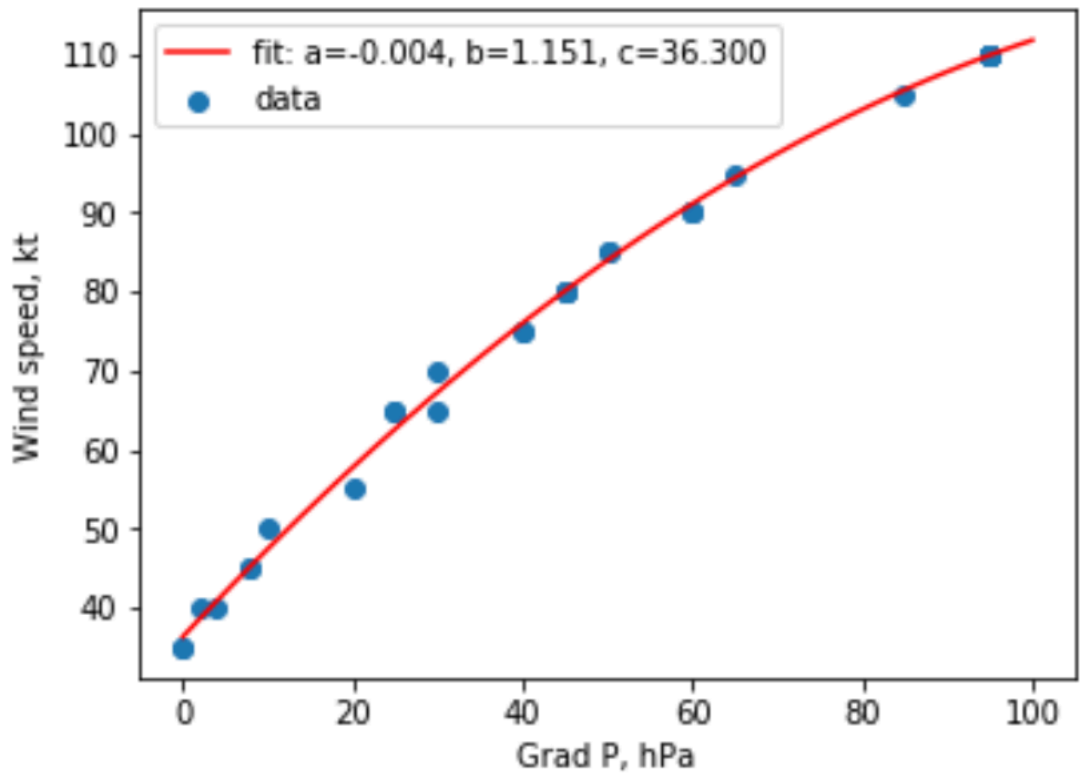
```
def func_linear(x, a, b):
    return a * x + b

plt.figure(figsize=(12,4))
# linear fit
popt, pcov = curve_fit(func_linear, mg['gradP'], mg['Wind'])

plt.subplot(1,2,1)
plt.scatter(mg.gradP, mg.Wind,label='data') # make a plot showing the relationship between gradP and Wind
plt.xlabel('Grad P, hPa')
plt.ylabel('Wind speed, kt')

x=np.linspace(0,100,50)

plt.plot(x, func_linear(x, *popt), 'r-',label='fit: a=%5.3f, b=%5.3f' % tuple(popt))
plt.legend()
```



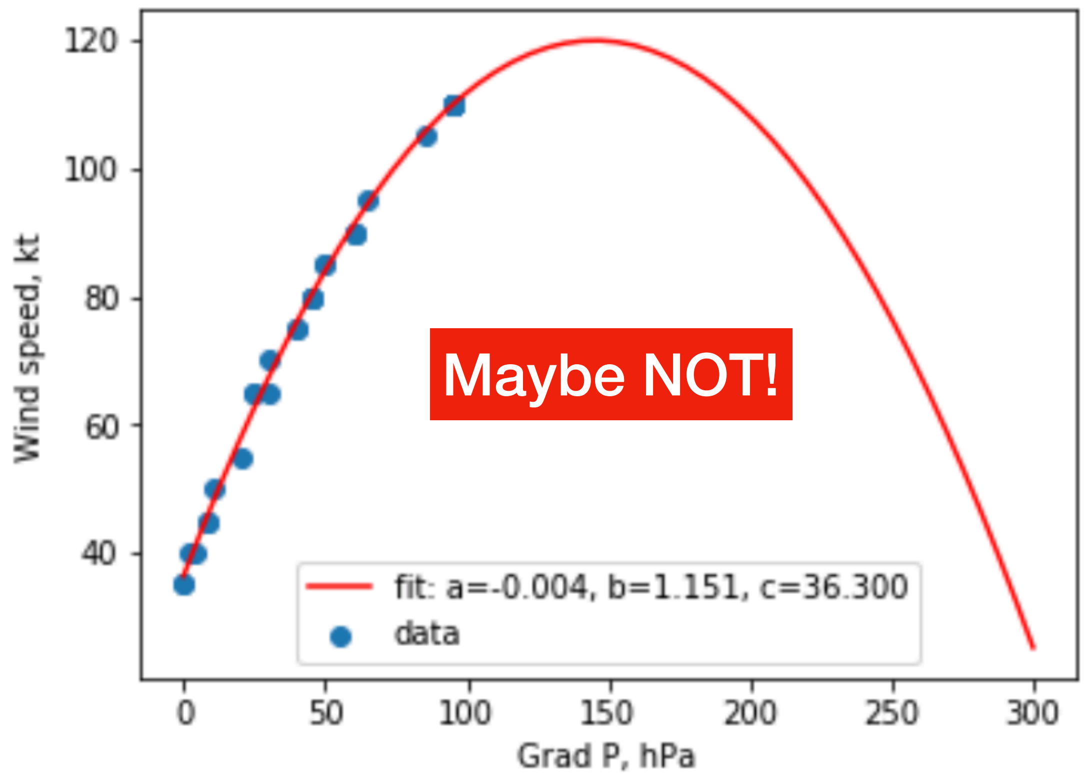
Now let's try a quadratic curve fitting between GradP and Wind Speed

Quadratic Fit ($y=ax^2+bx+c$) Results:

```
# quadratic fit
def func_quadratic(x, a, b, c):
    return a * x**2 + b*x + c

popt, pcov = curve_fit(func_quadratic, mg['gradP'], mg['Wind'])

plt.subplot(1,2,2)
plt.scatter(mg.gradP, mg.Wind,label='data') # make a plot showing the relationship between gradP and Wind
plt.xlabel('Grad P, hPa')
plt.ylabel('Wind speed, kt')

plt.plot(x, func_quadratic(x, *popt), 'r-',label='fit: a=%5.3f, b=%5.3f, c=%5.3f' % tuple(popt))
plt.legend()
plt.show()
```



Is it necessary a good fit?


Maybe NOT!
fit: a=-0.004, b=1.151, c=36.300
data
Wind speed, kt
Grad P, hPa