

[301] Regression

(and numpy)

Tyler Caraza-Harter

Learning Objectives Today

History of regression

Drawing a fit line

Finding the slope/intercept w/ least squares method

Numpy introduction

Using `numpy.linalg.lstsq`

Definition

regression definition



All

Books

Images

News

Videos

More

Settings

Tools

About 141,000,000 results (0.41 seconds)

Dictionary

Search for a word



re·gres·sion

/rə'greʃ(ə)n/

noun

1. a return to a former or less developed state.

???

2. **STATISTICS**

a measure of the relation between the mean value of one variable (e.g., output) and corresponding values of other variables (e.g., time and cost).

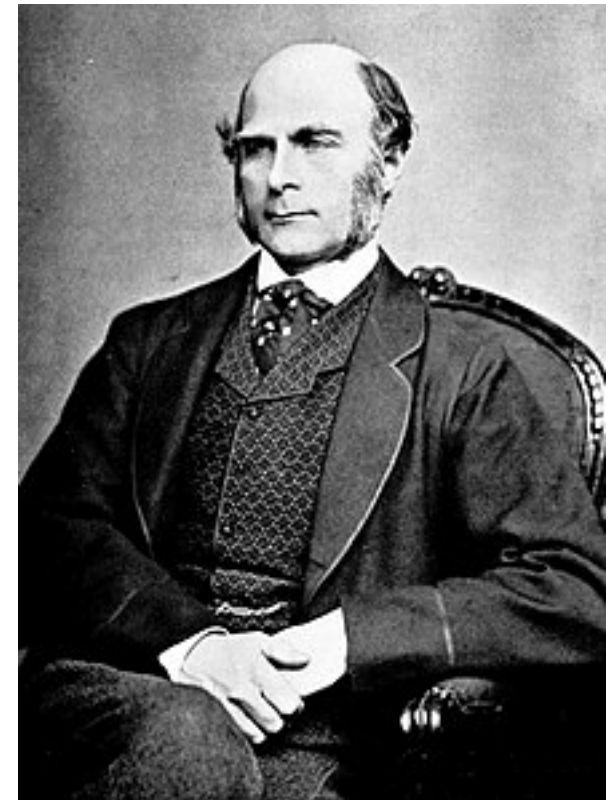
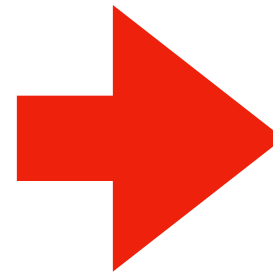


Translations, word origin, and more definitions

Feedback

History of Regression

Francis Galton

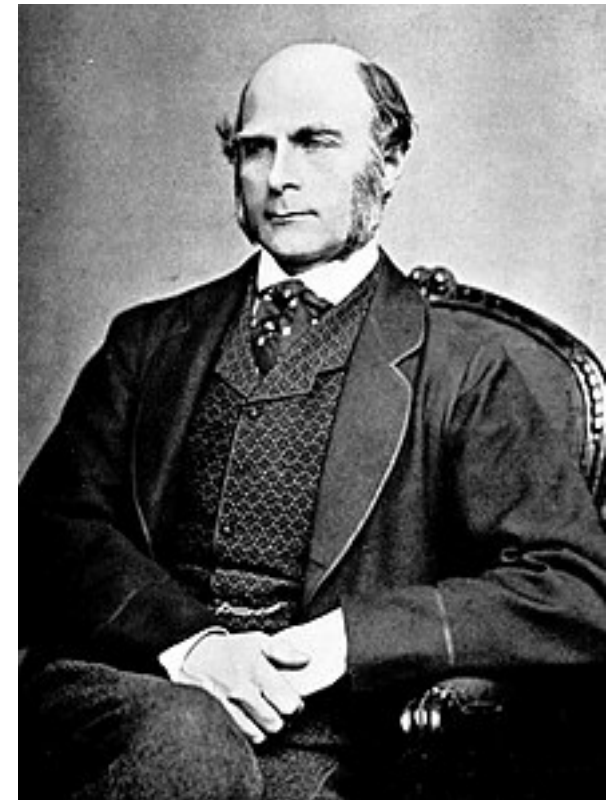
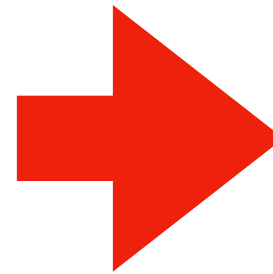


https://en.wikipedia.org/wiki/Francis_Galton

Question: what is the relationship between a parent's and child's height (both as adults)

History of Regression

Francis Galton

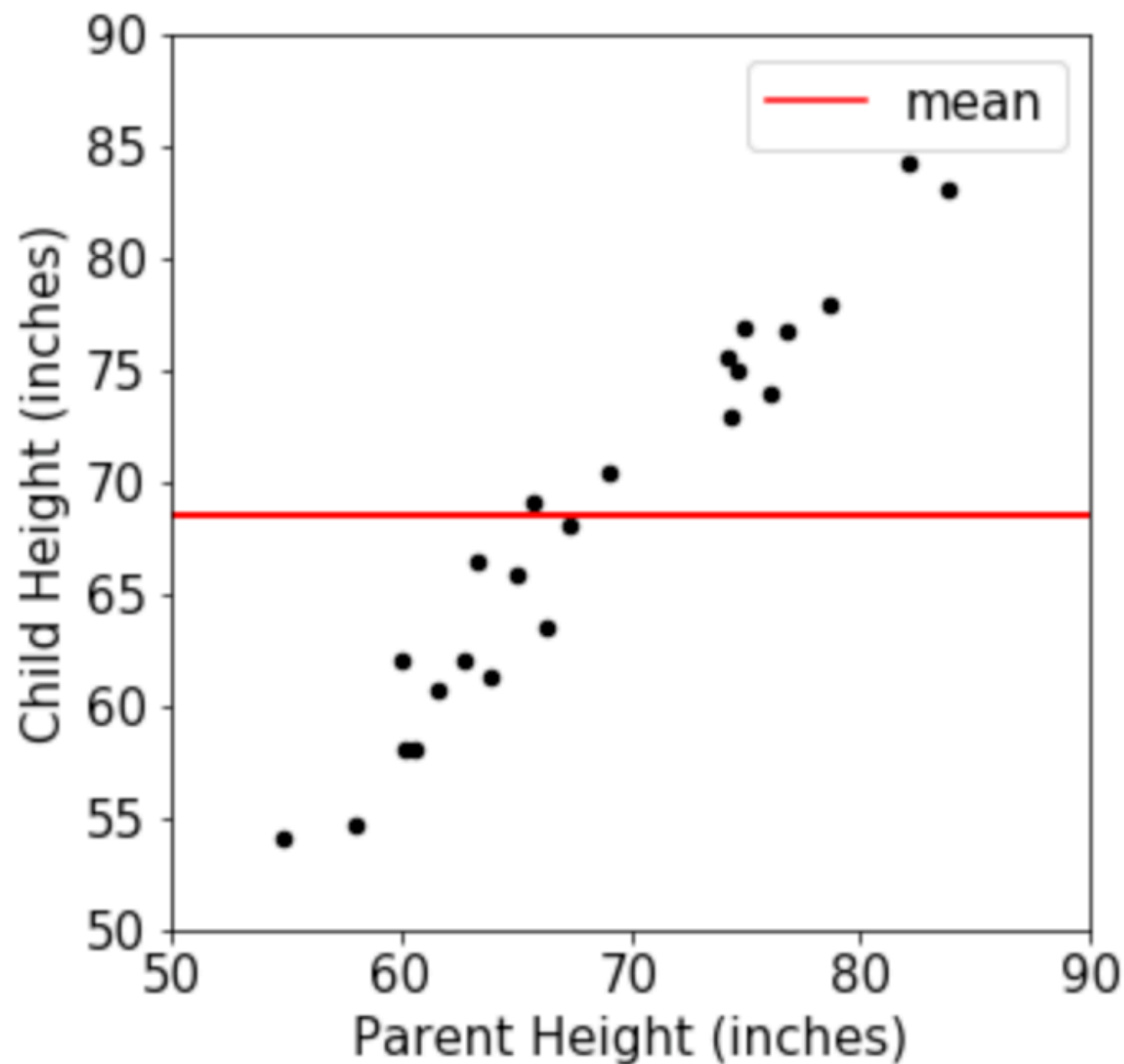


https://en.wikipedia.org/wiki/Francis_Galton

Question: what is the relationship between a parent's and child's height (both as adults)

What kind of plot should we make?

Result you might expect

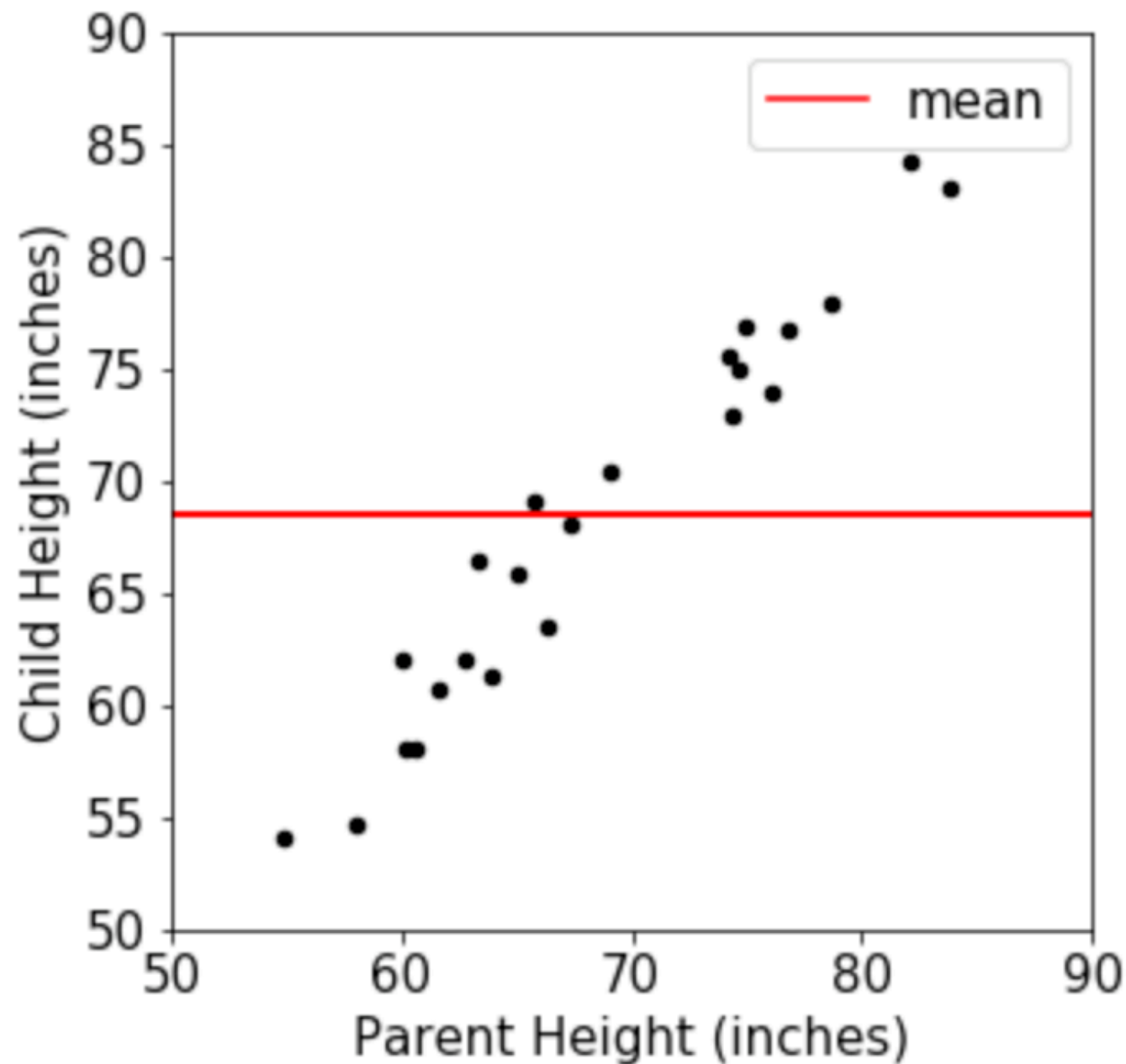


Observation:

- child height equals parent height (plus some noise)

Note: all these height plots contain fake data

Result you might expect



Observation:

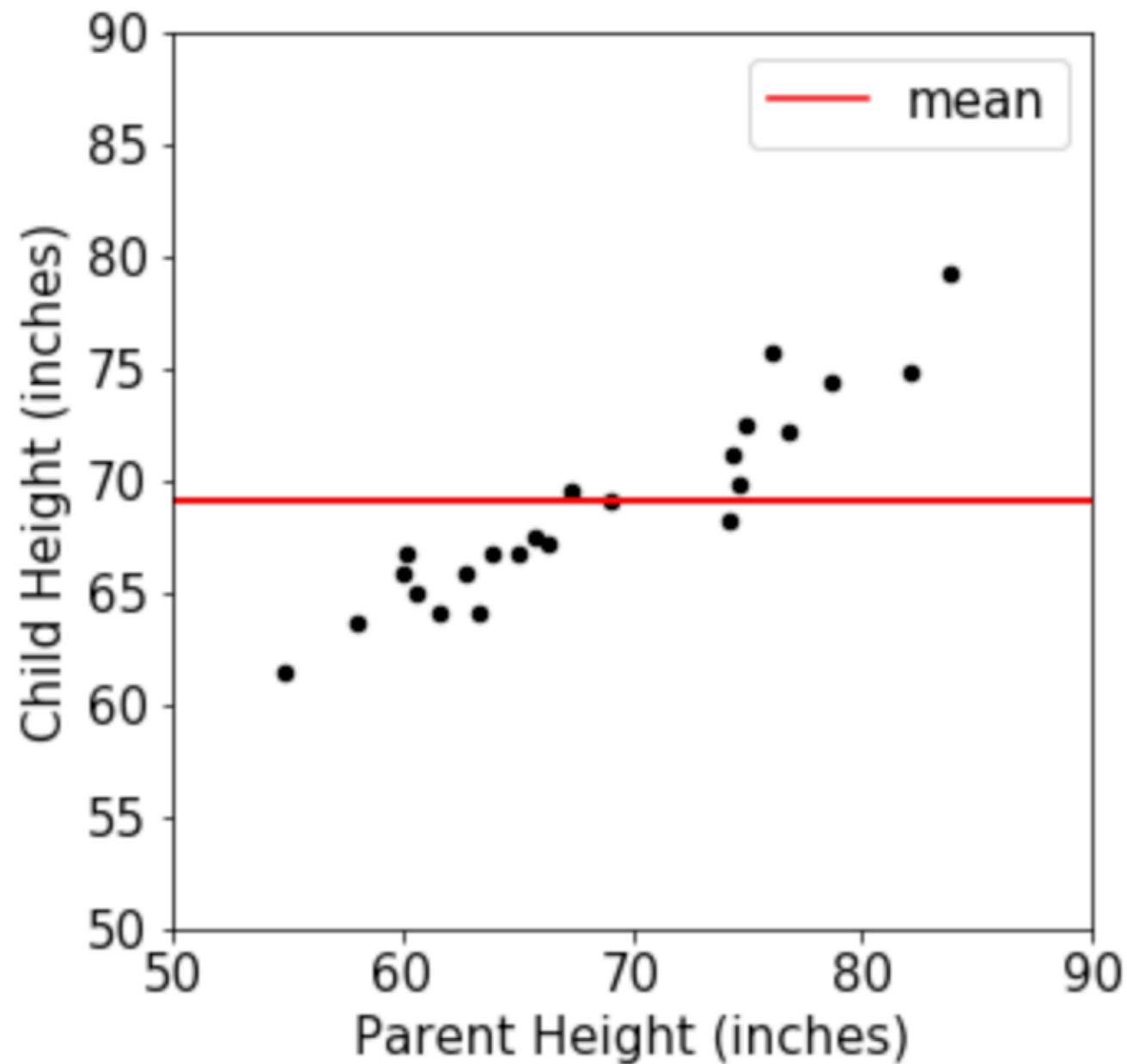
- child height equals parent height (plus some noise)

What about other factors?

- height of other parent
- nutrition
- etc

Note: all these height plots contain fake data

More realistic results

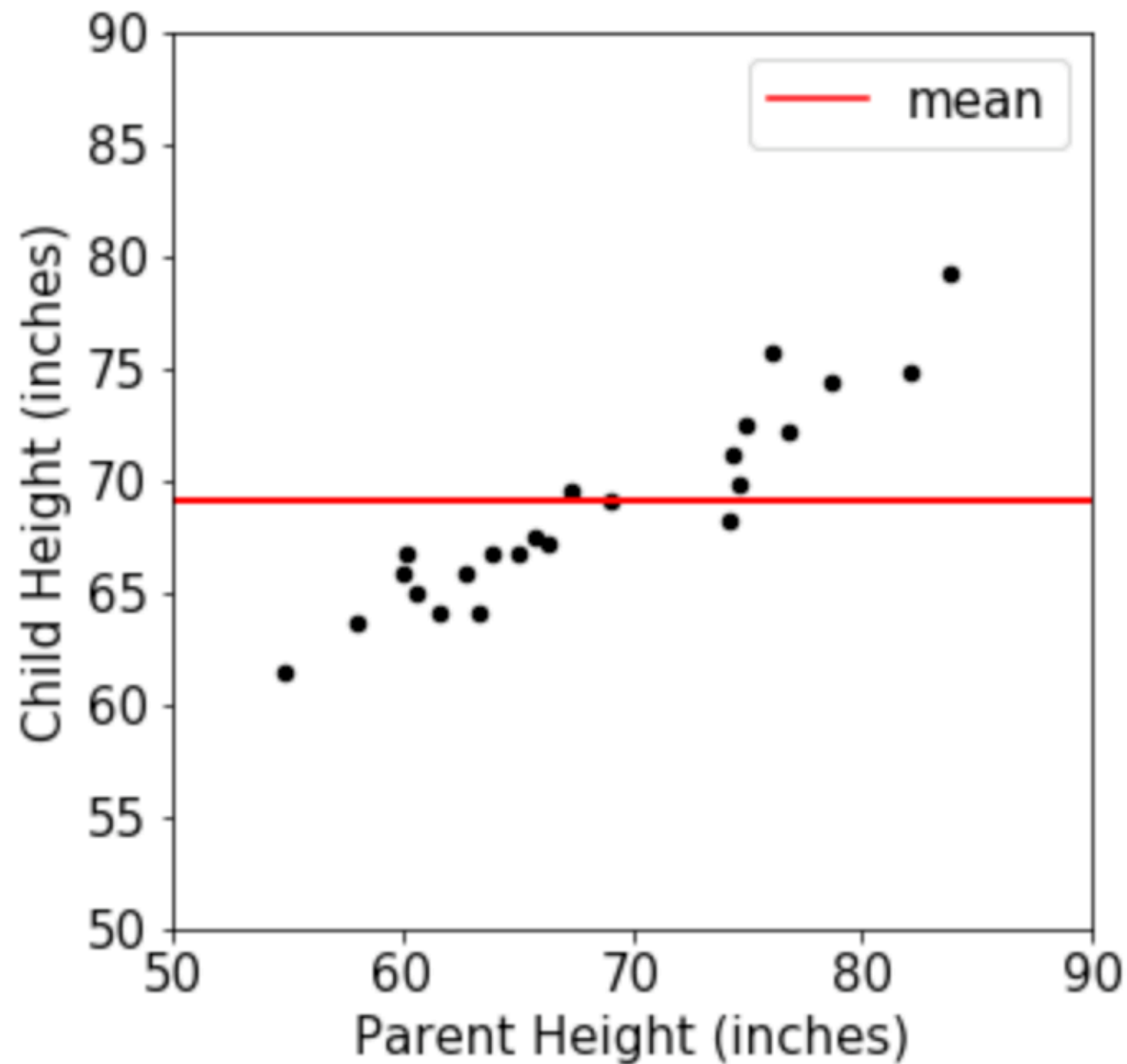


Observation:

- heights are correlated
- tall parents tend to have shorter children
- short parents tend to have taller children

Note: all these height plots contain fake data

More realistic results



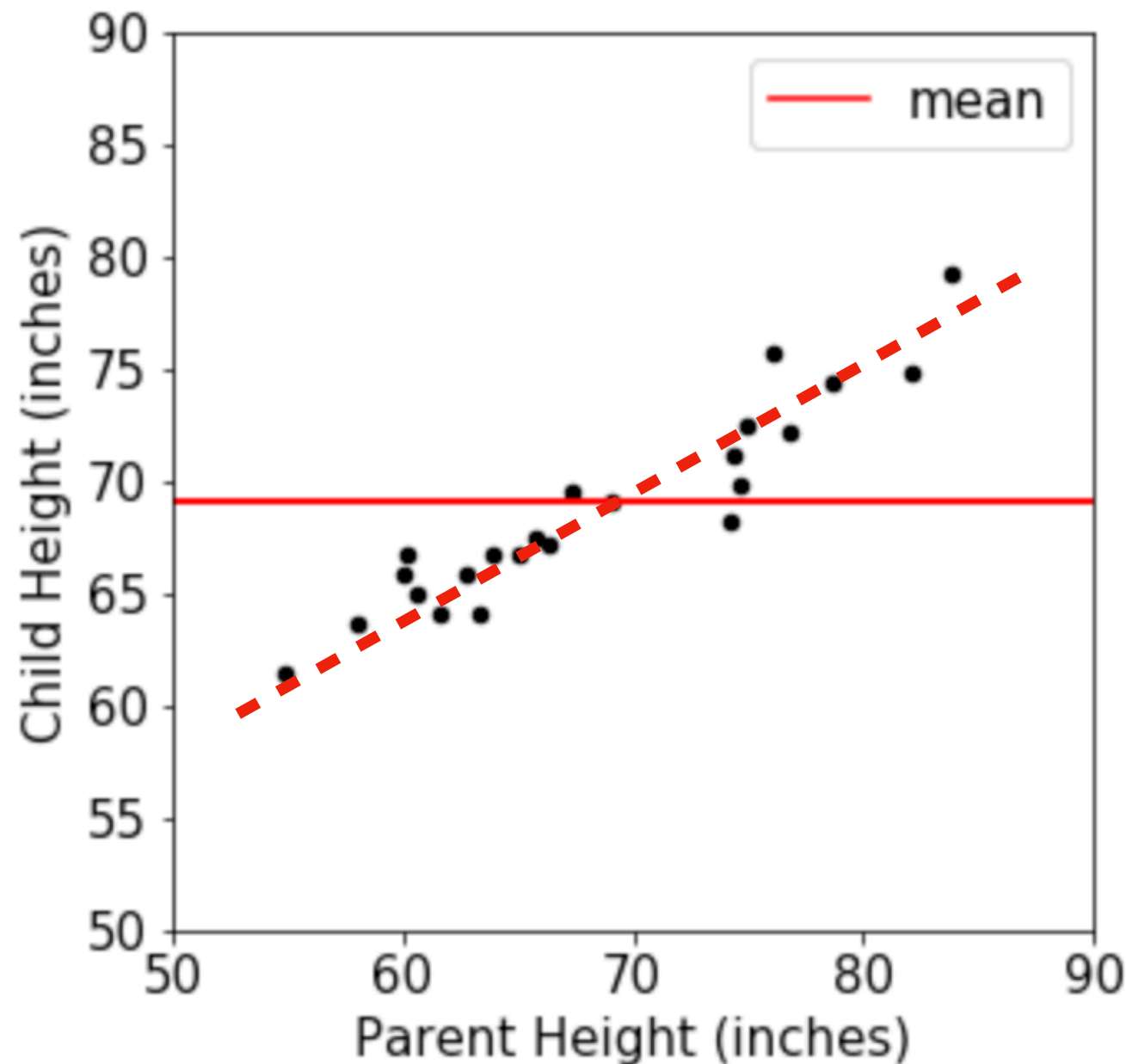
Observation:

- heights are correlated
- tall parents tend to have shorter children
- short parents tend to have taller children

Galton referred to this phenomenon as “regression to the mean”.

Note: all these height plots contain fake data

More realistic results



Observation:

- heights are correlated
- tall parents tend to have shorter children
- short parents tend to have taller children

Galton referred to this phenomenon as “regression to the mean”.

Nowadays, “**regression**” can refer to any fitting of a line to the points.

Note: all these height plots contain fake data

Learning Objectives Today

History of regression

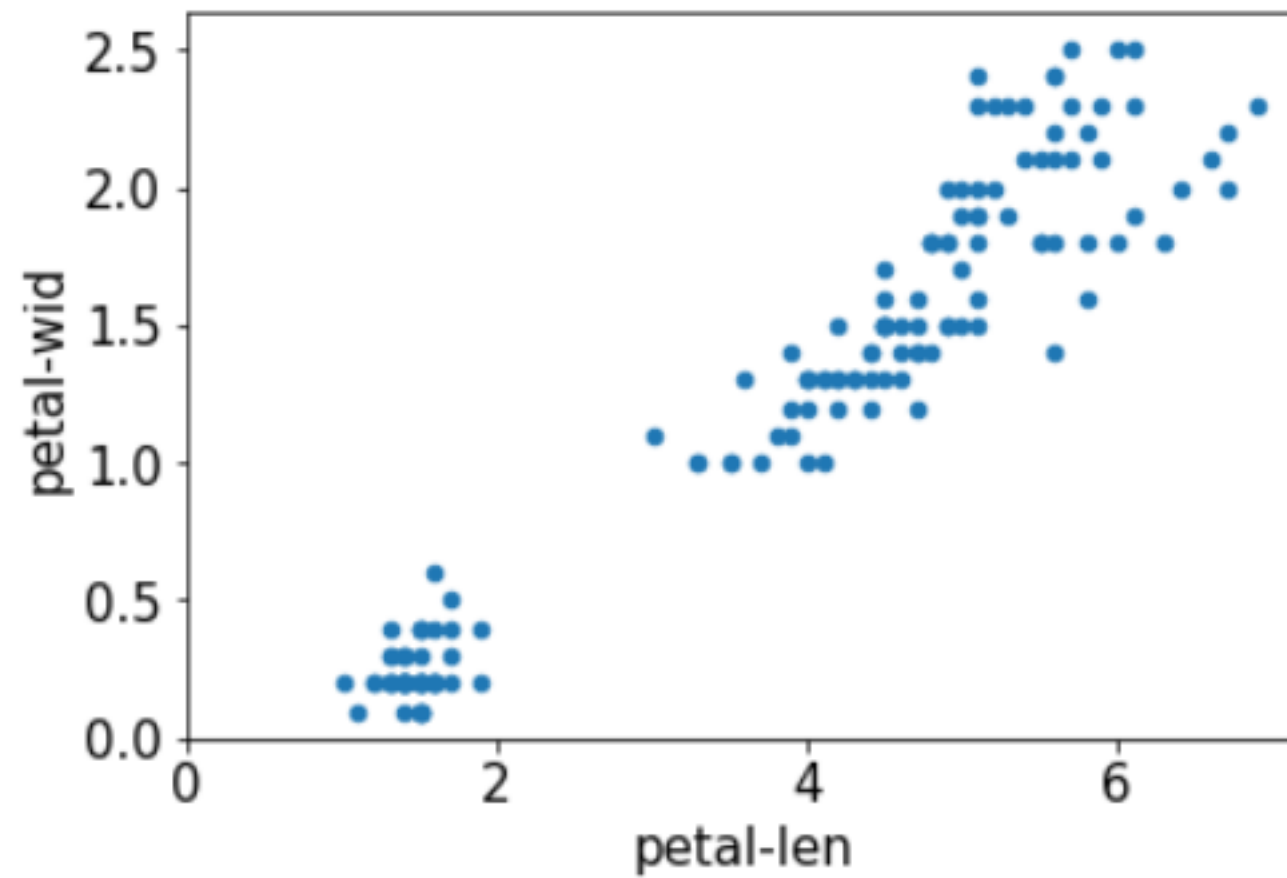
Drawing a fit line

Finding the slope/intercept w/ least squares method

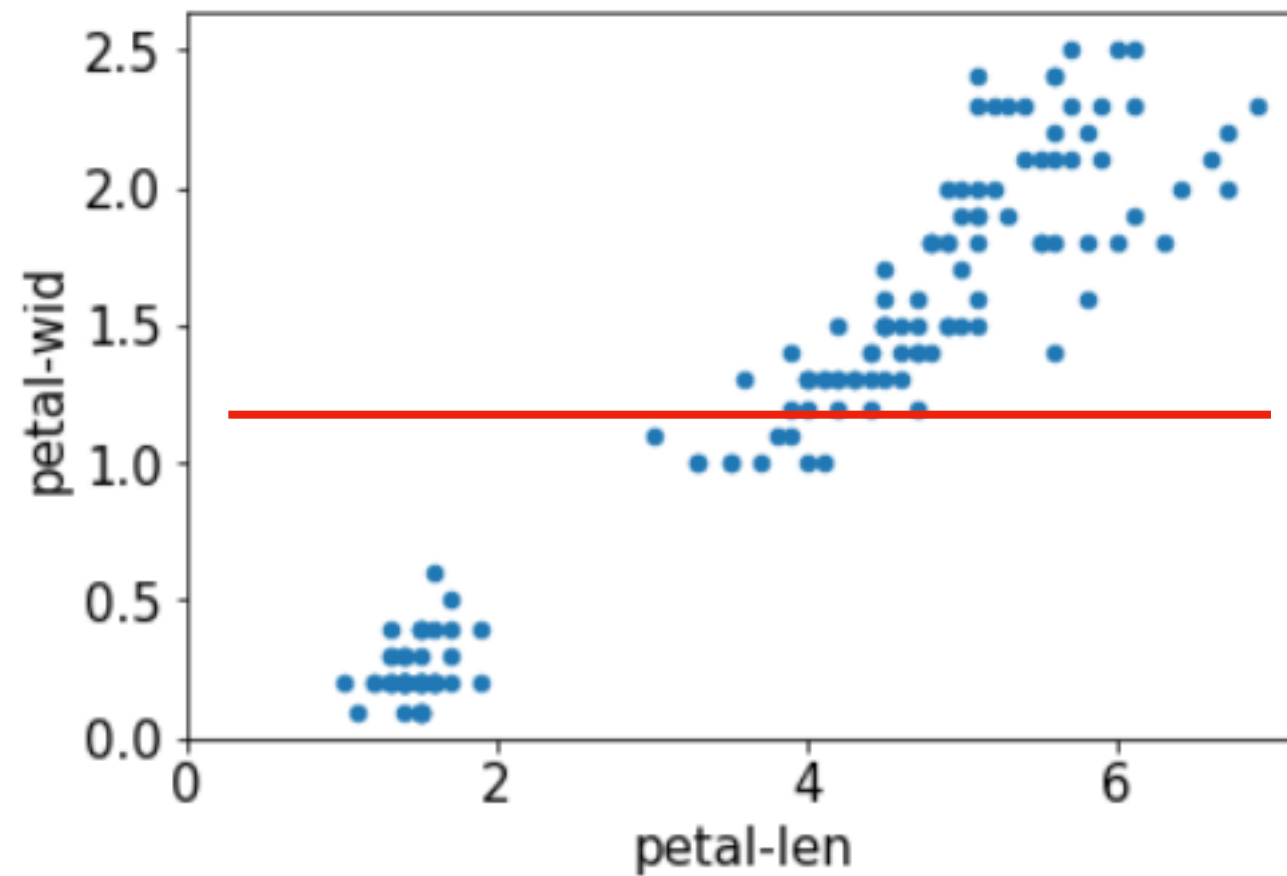
Numpy introduction

Using `numpy.linalg.lstsq`

Demo 1: annotate Iris data

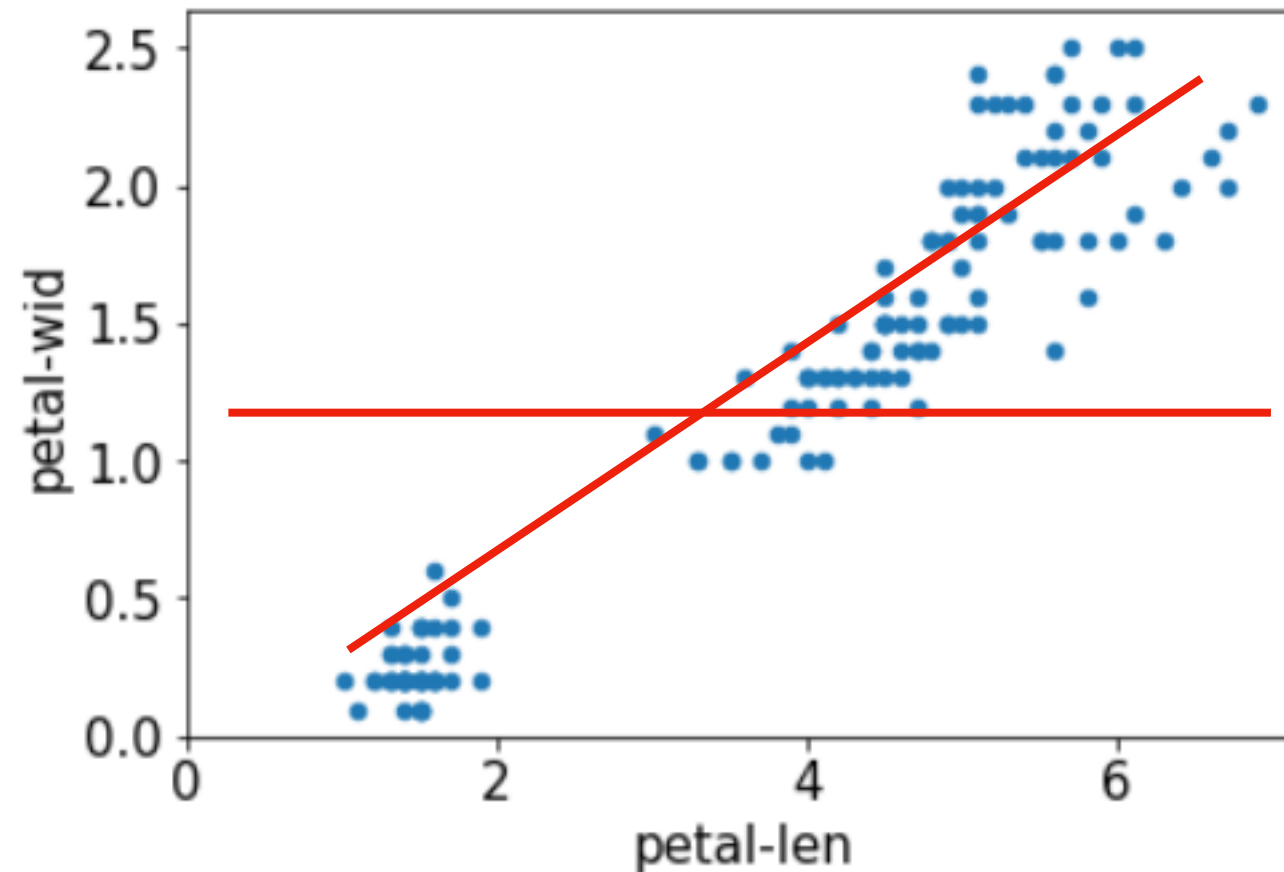


Demo 1: annotate Iris data



Annotation 1: mean line

Demo 1: annotate Iris data



Annotation 1: mean line

Annotation 2: fit line

- assume slope=1/3
- y-intercept=0

*not necessarily
the best line*

Learning Objectives Today

History of regression

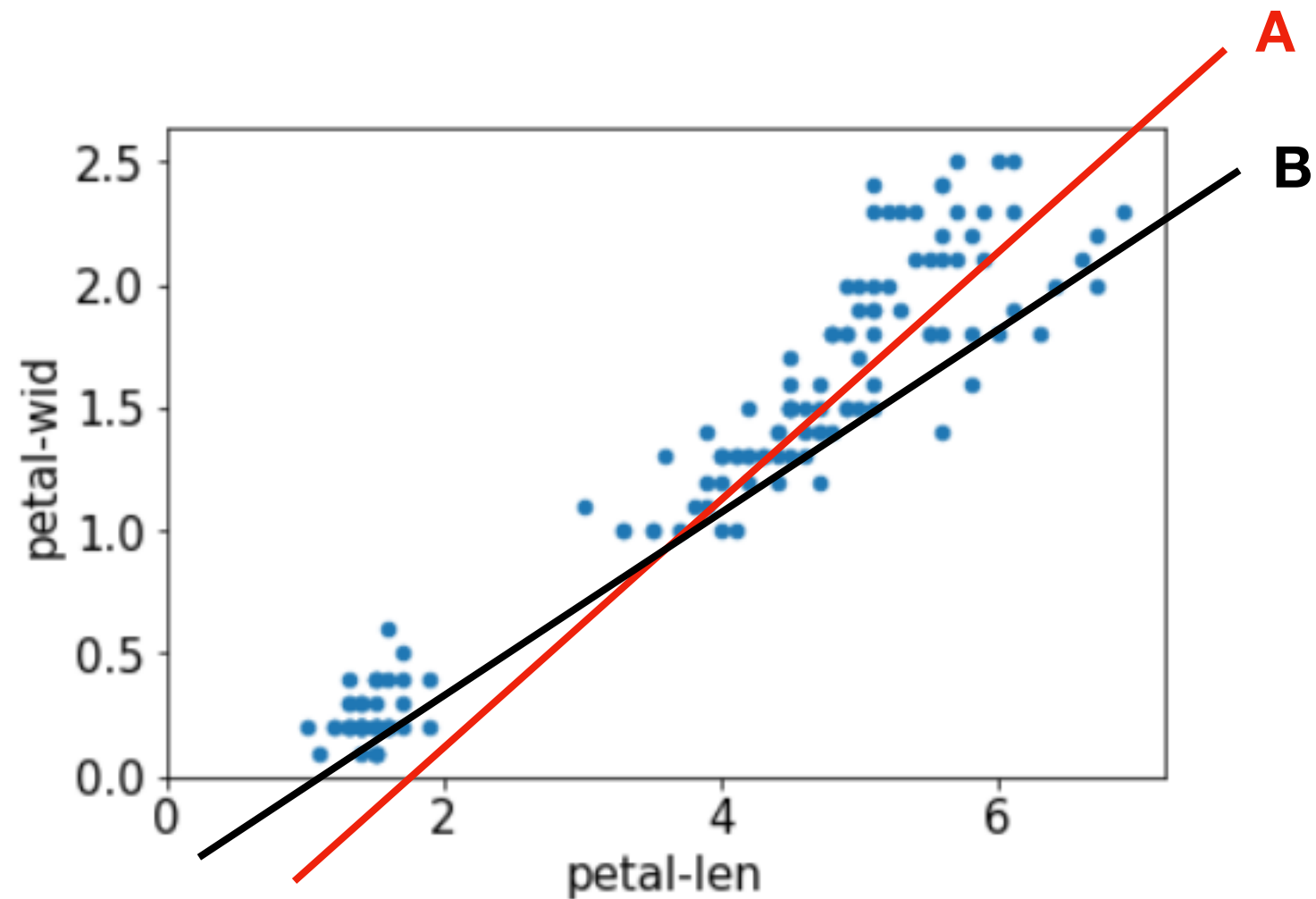
Drawing a fit line

Finding the slope/intercept w/ least-squares method

Numpy introduction

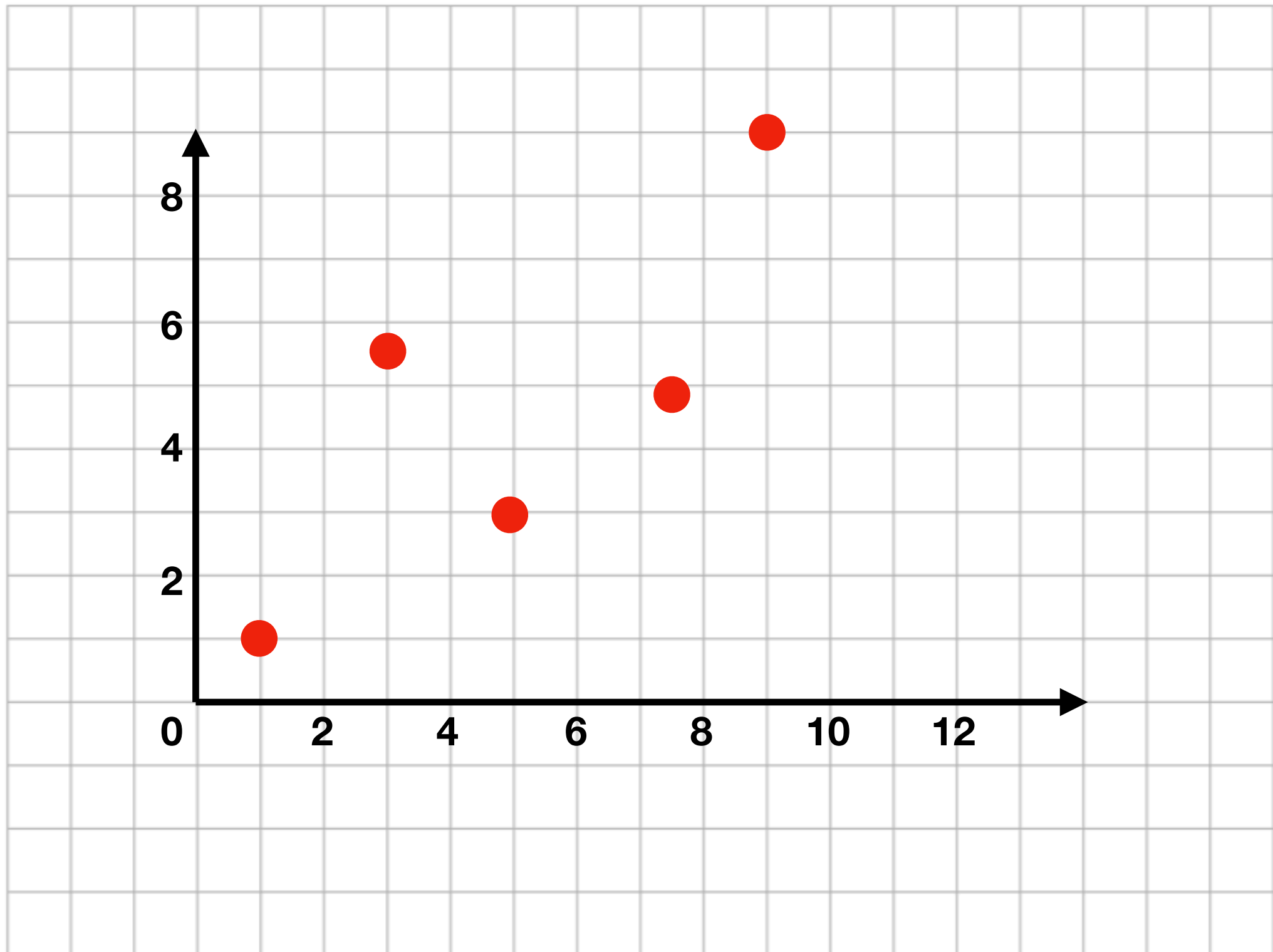
Using `numpy.linalg.lstsq`

Which fit line is better?

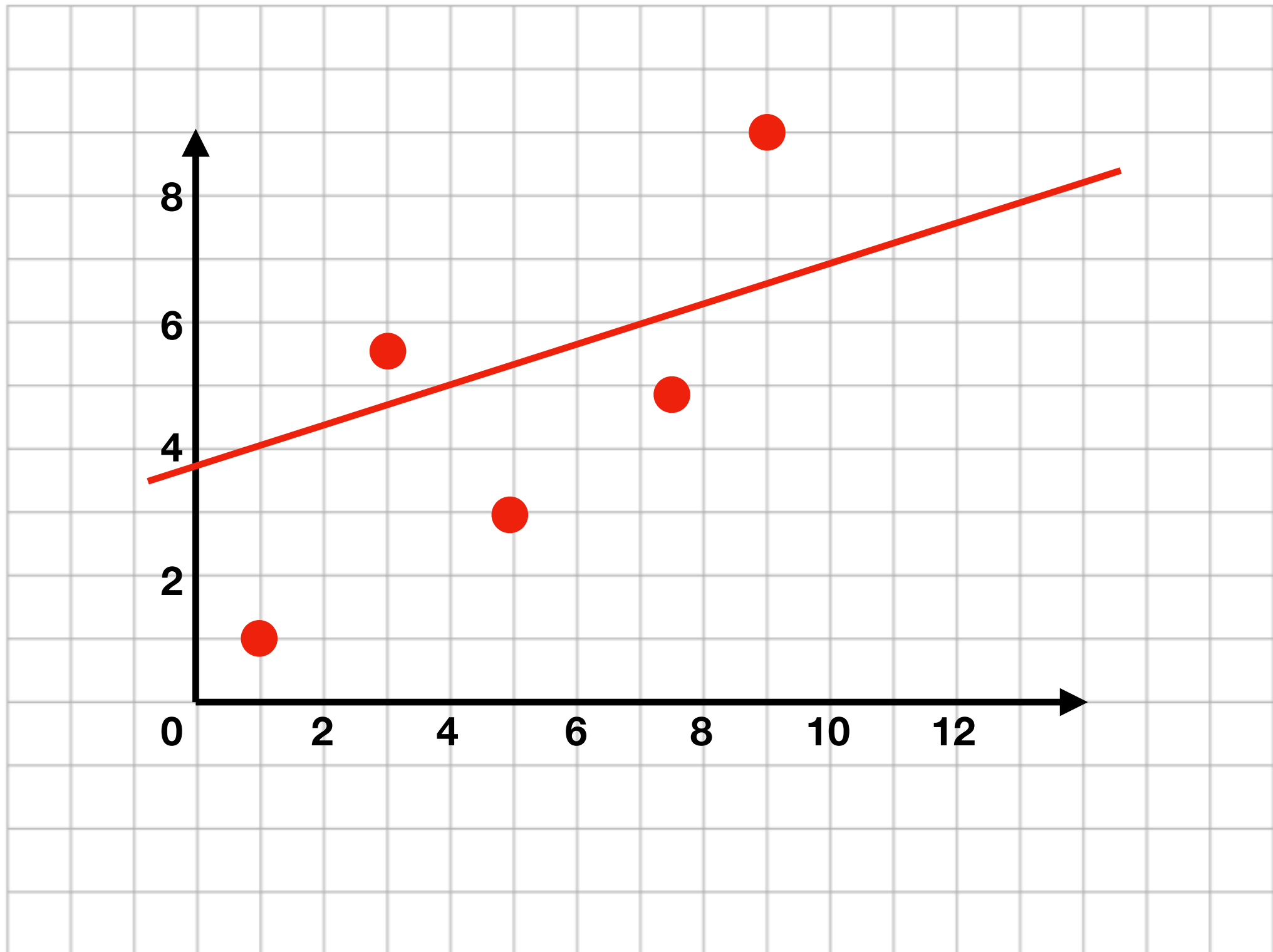


We need a metric to evaluate how good a fit is

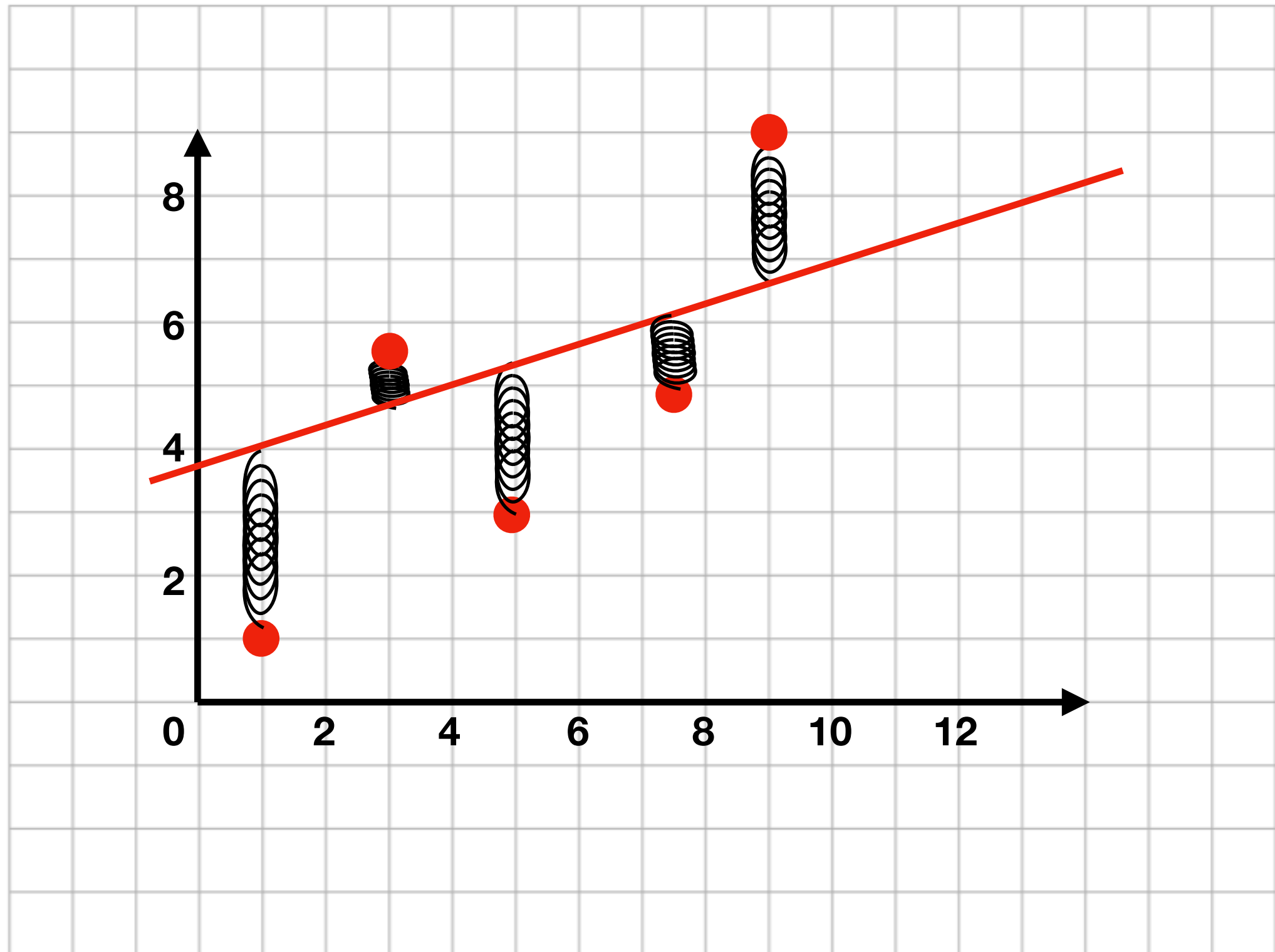
Intuition: Springs



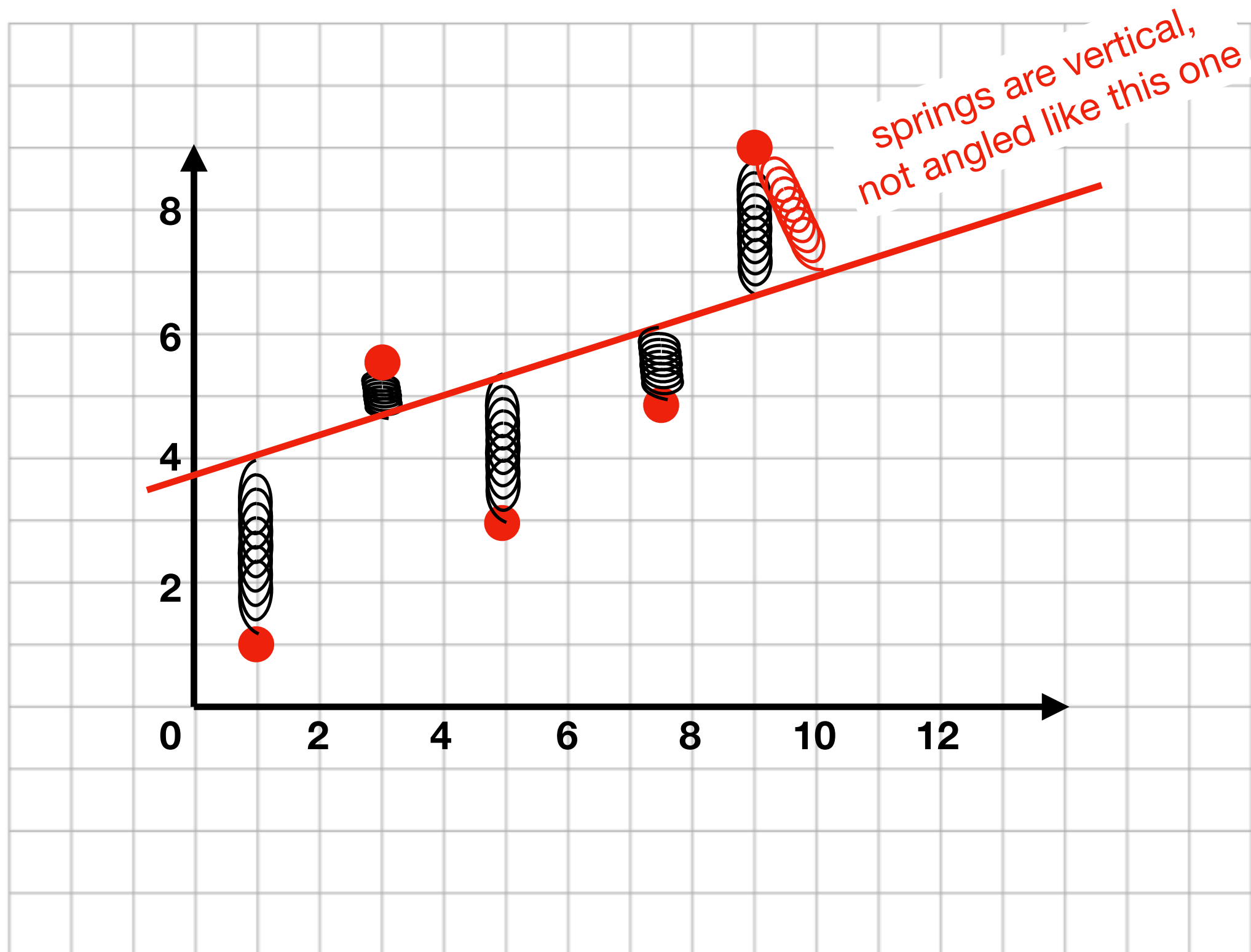
Intuition: Springs



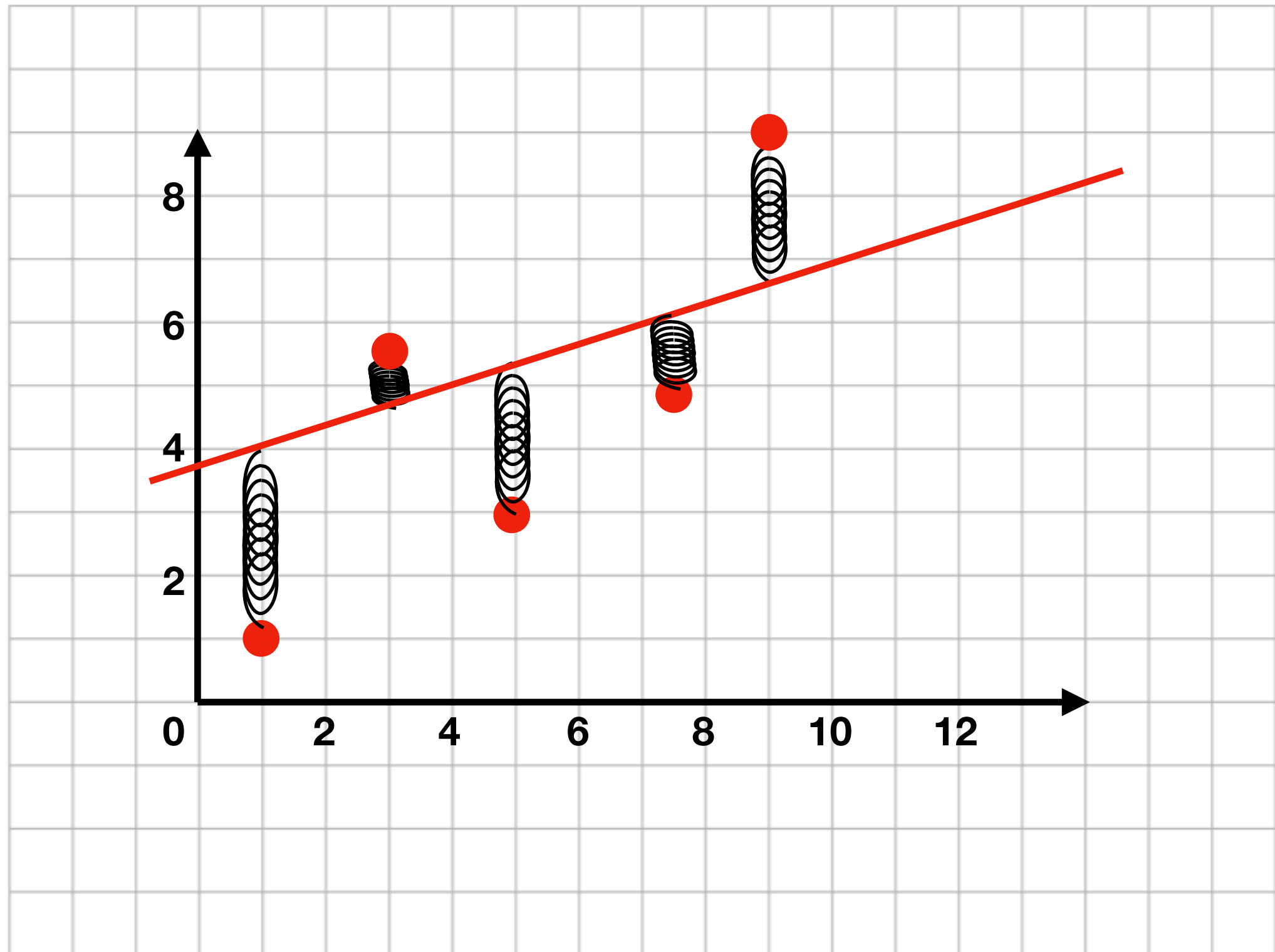
Intuition: Springs



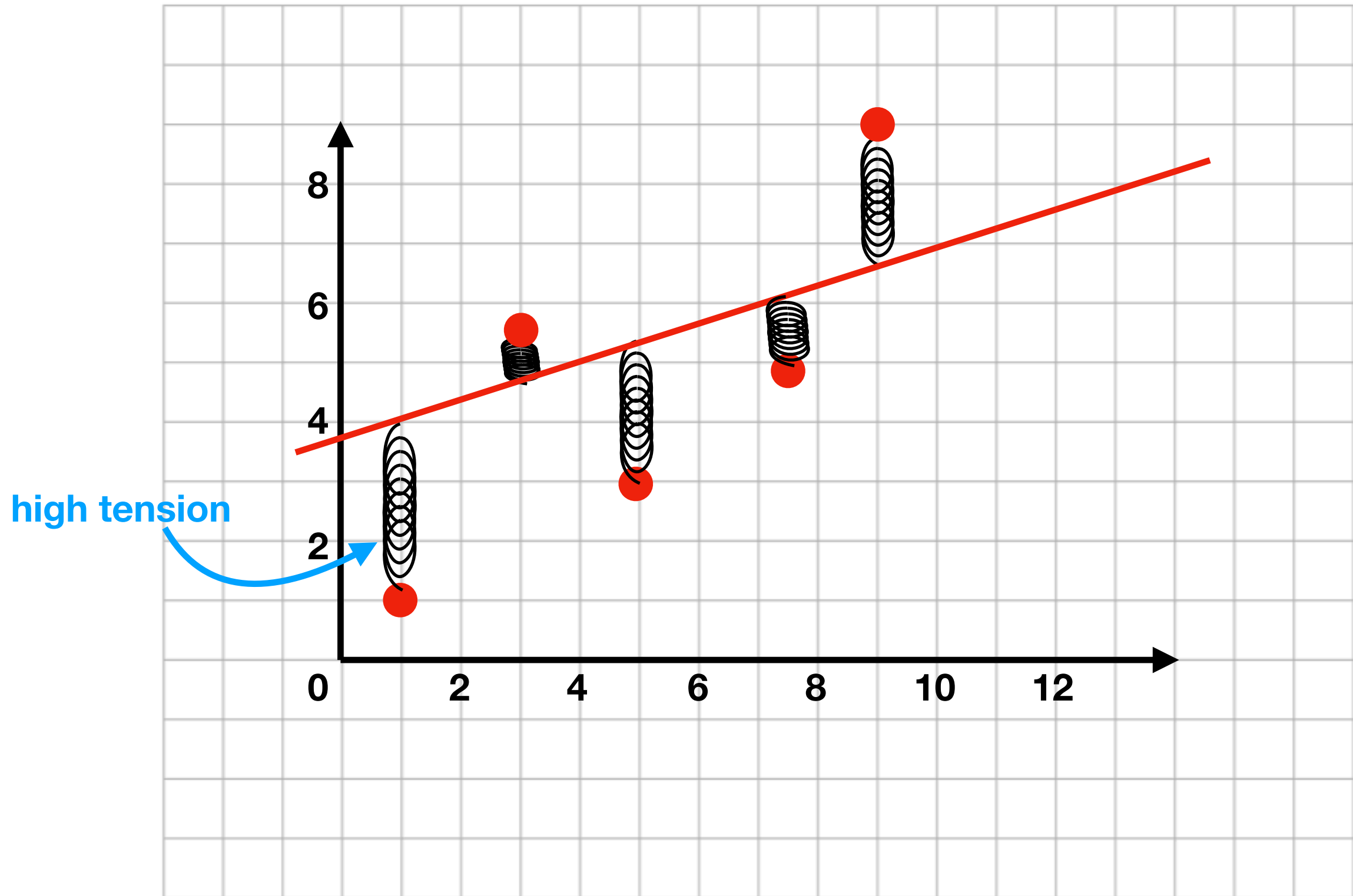
Intuition: Springs



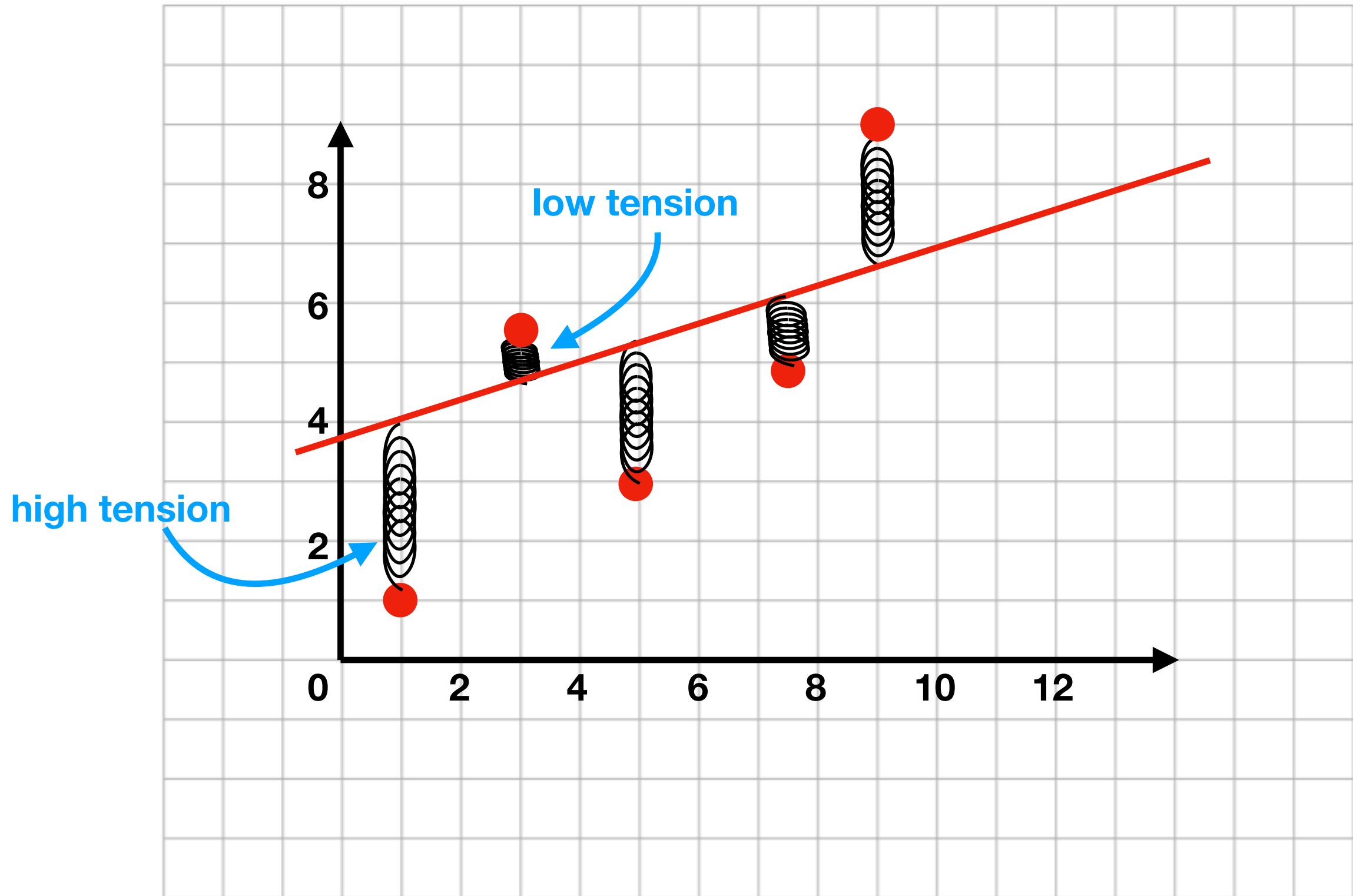
Intuition: Springs



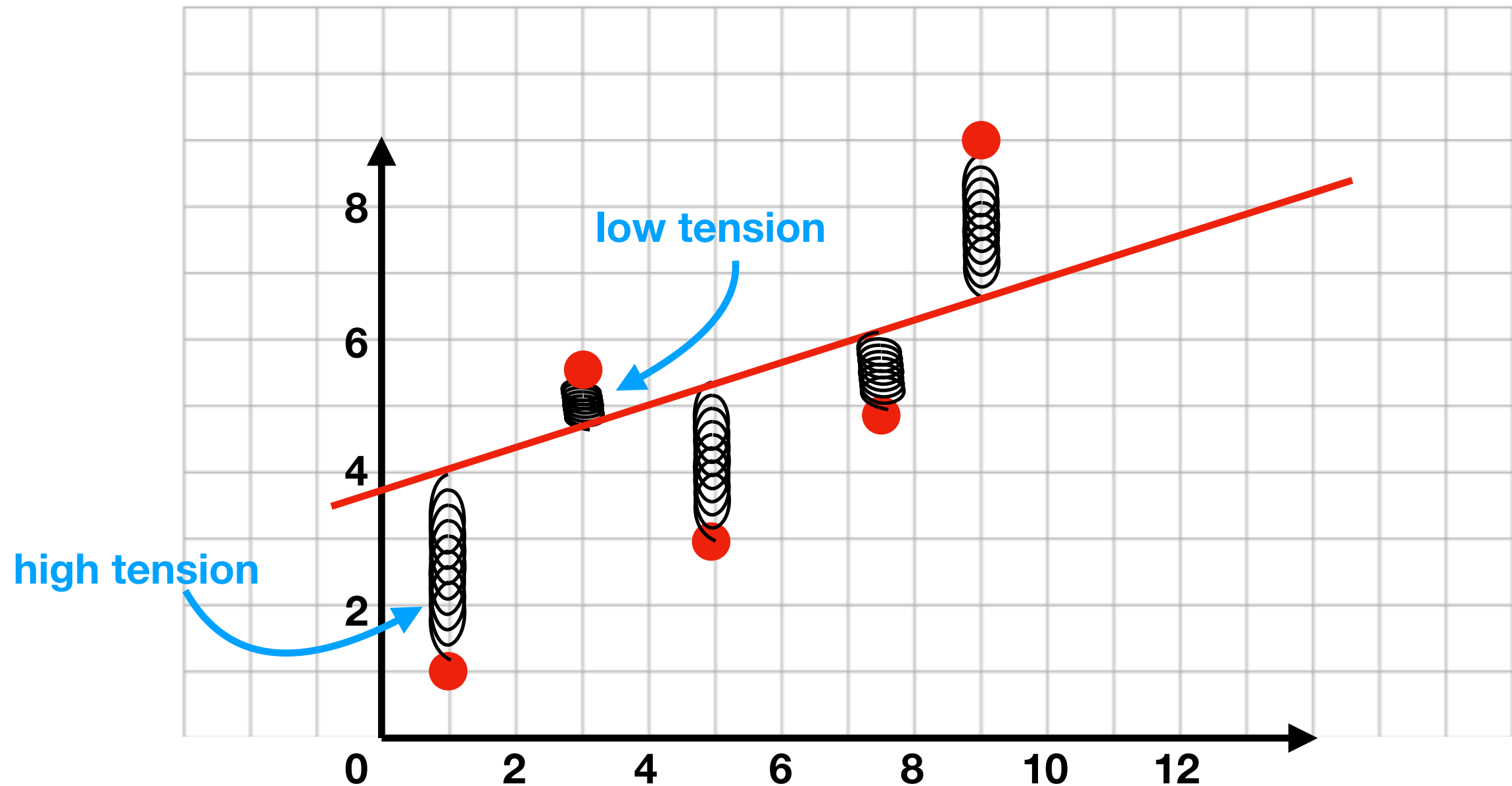
Intuition: Springs



Intuition: Springs

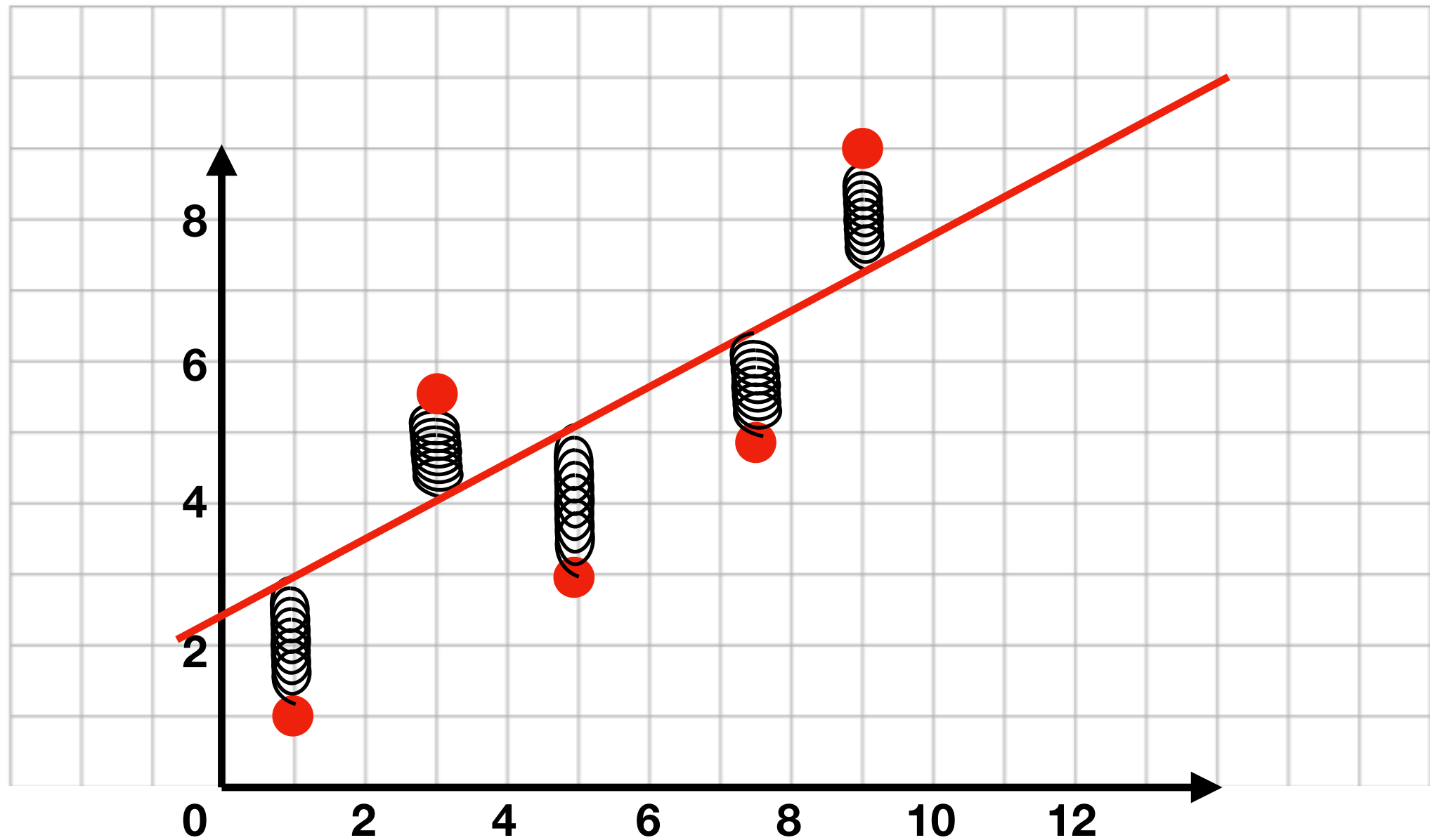


Intuition: Springs



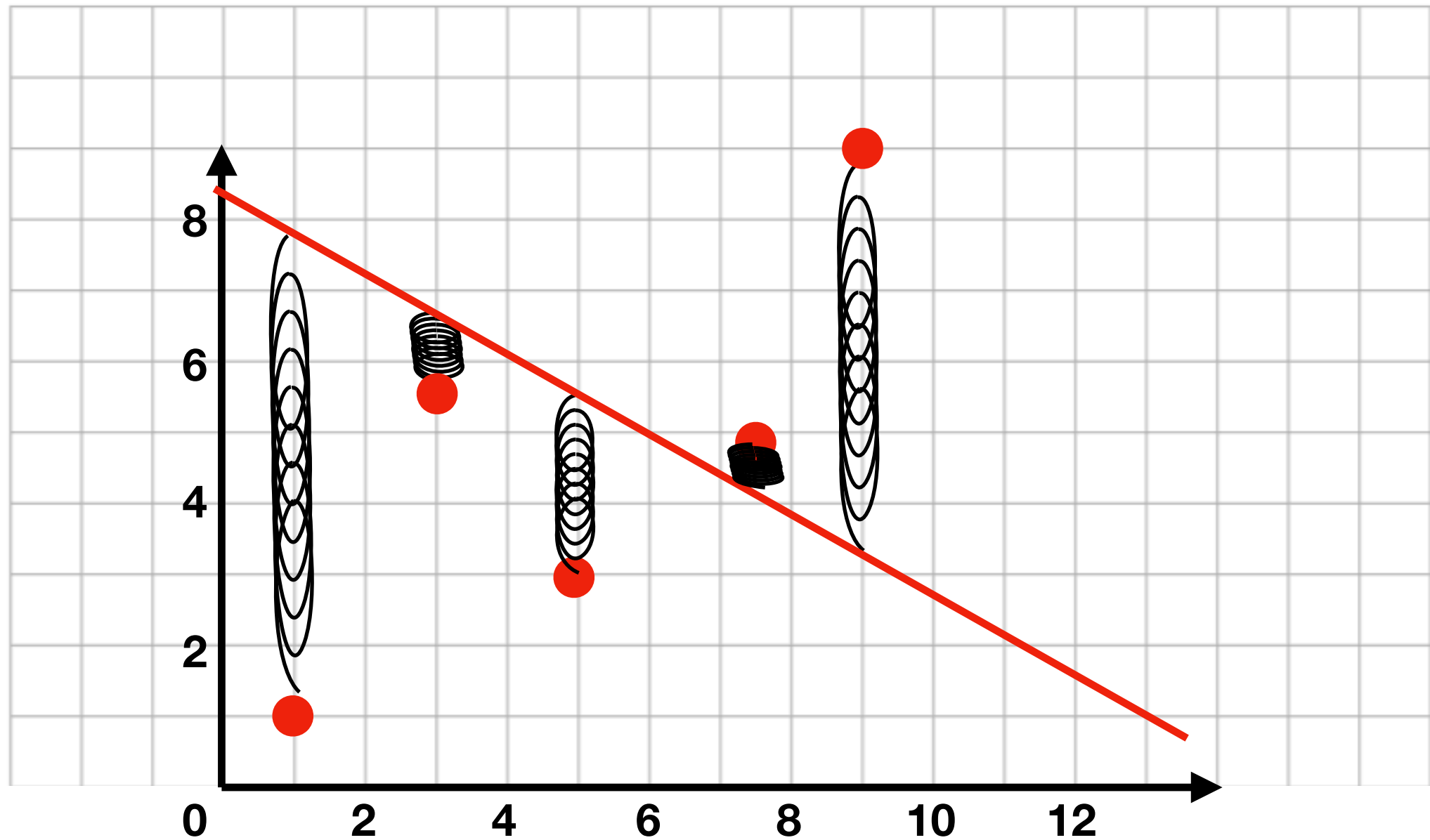
The best line minimizes total tension across springs

Intuition: Springs



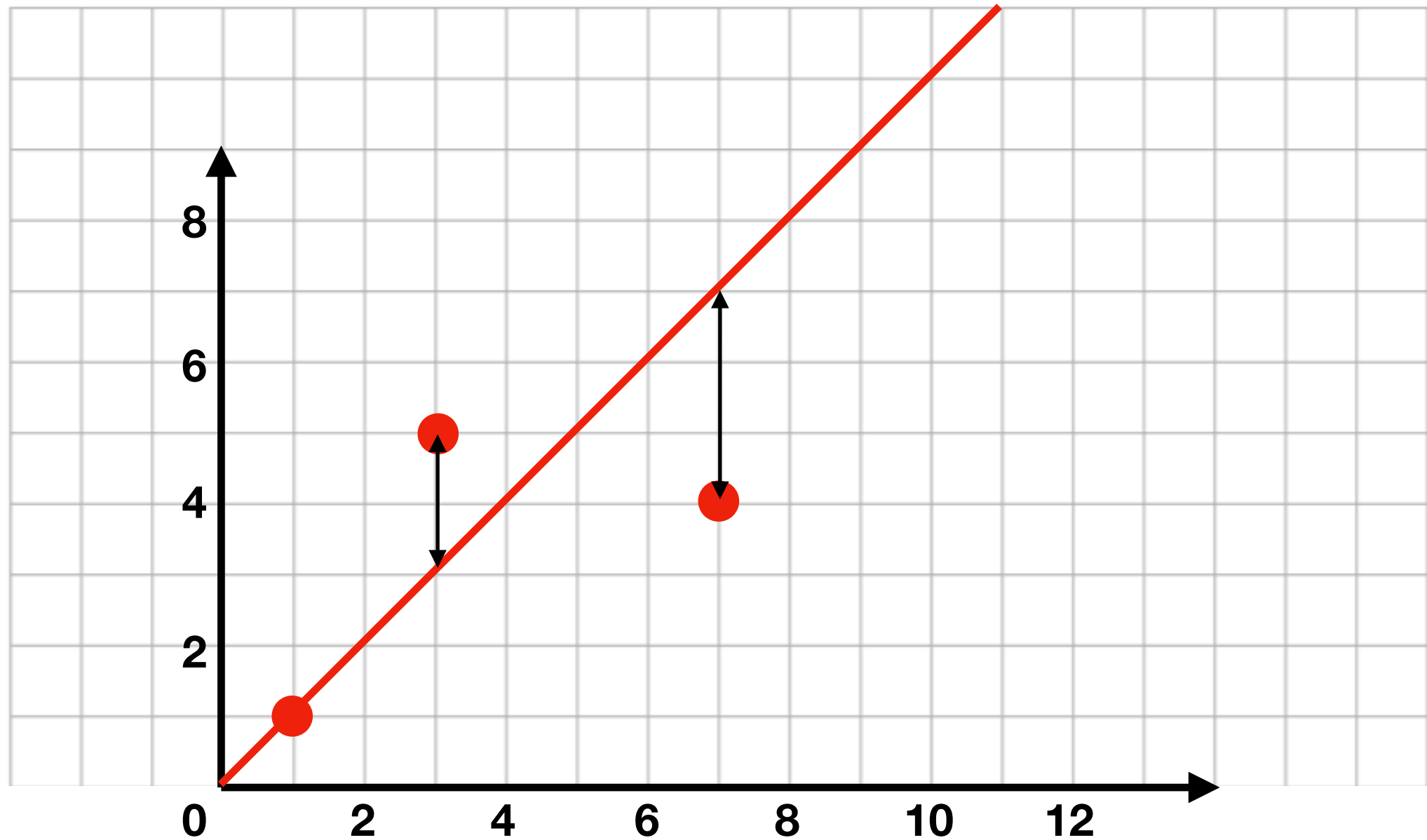
Good fit with low overall tension

Intuition: Springs



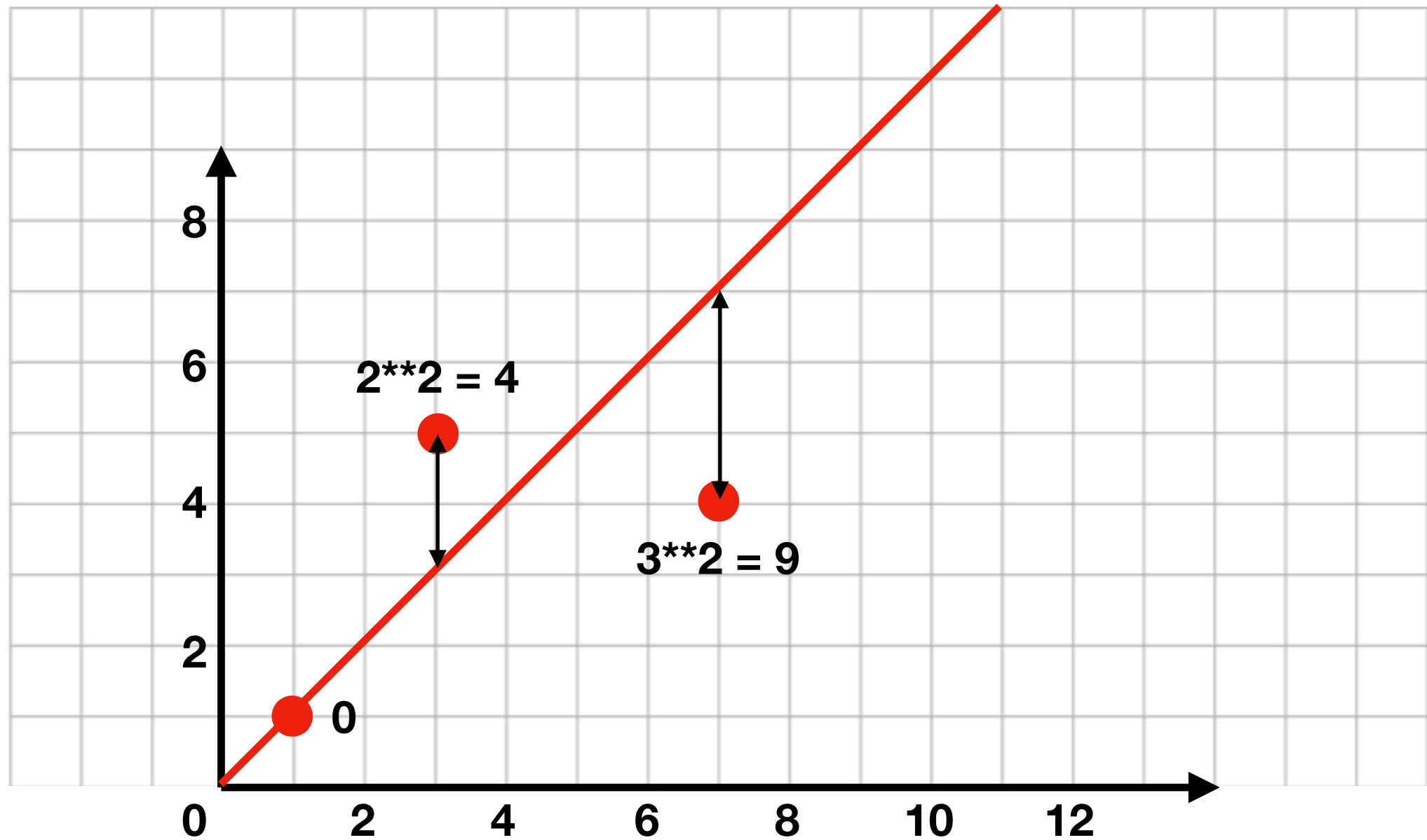
Bad fit with high overall tension

Intuition: Springs



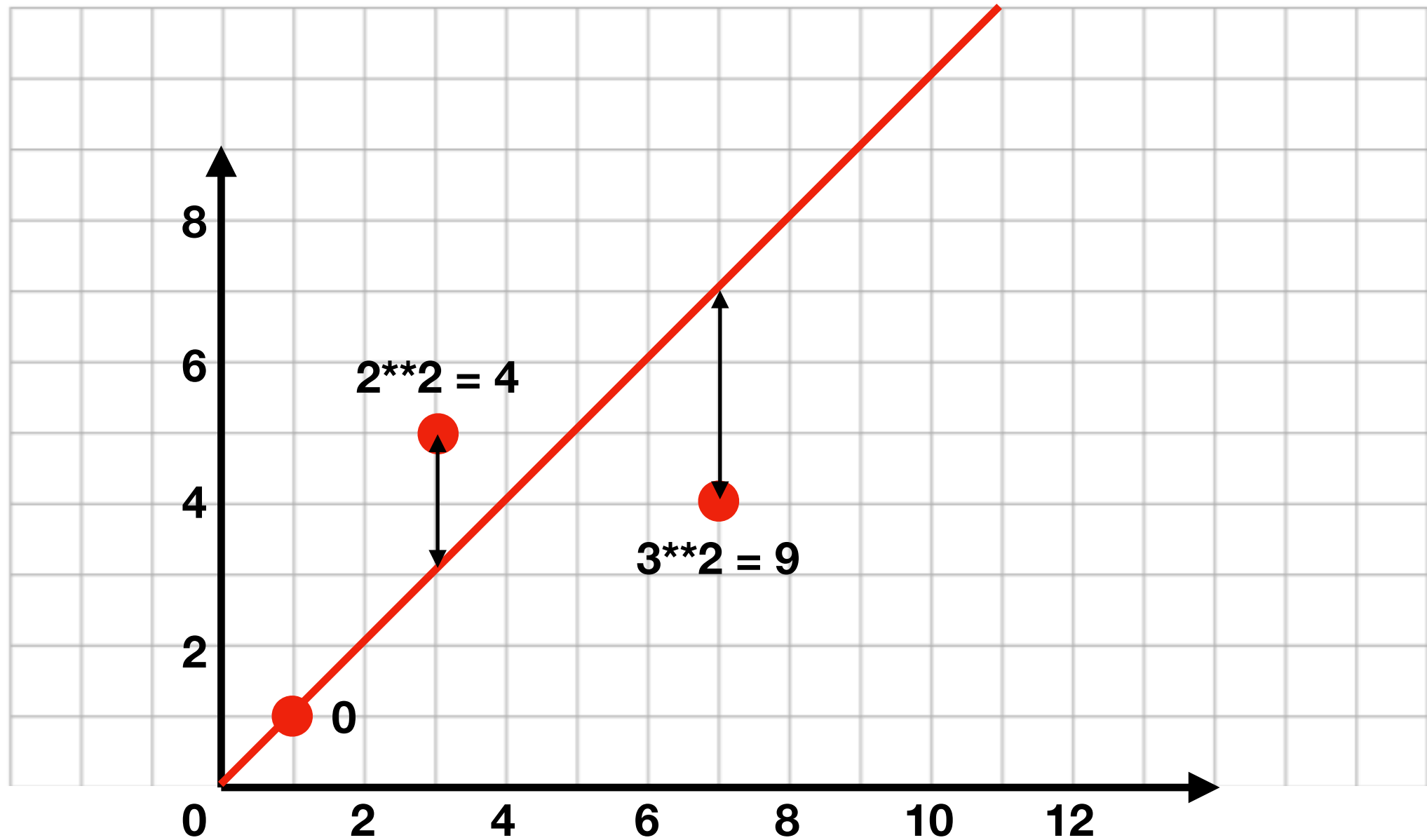
Tension is defined as distance squared

Intuition: Springs



Tension is defined as distance squared

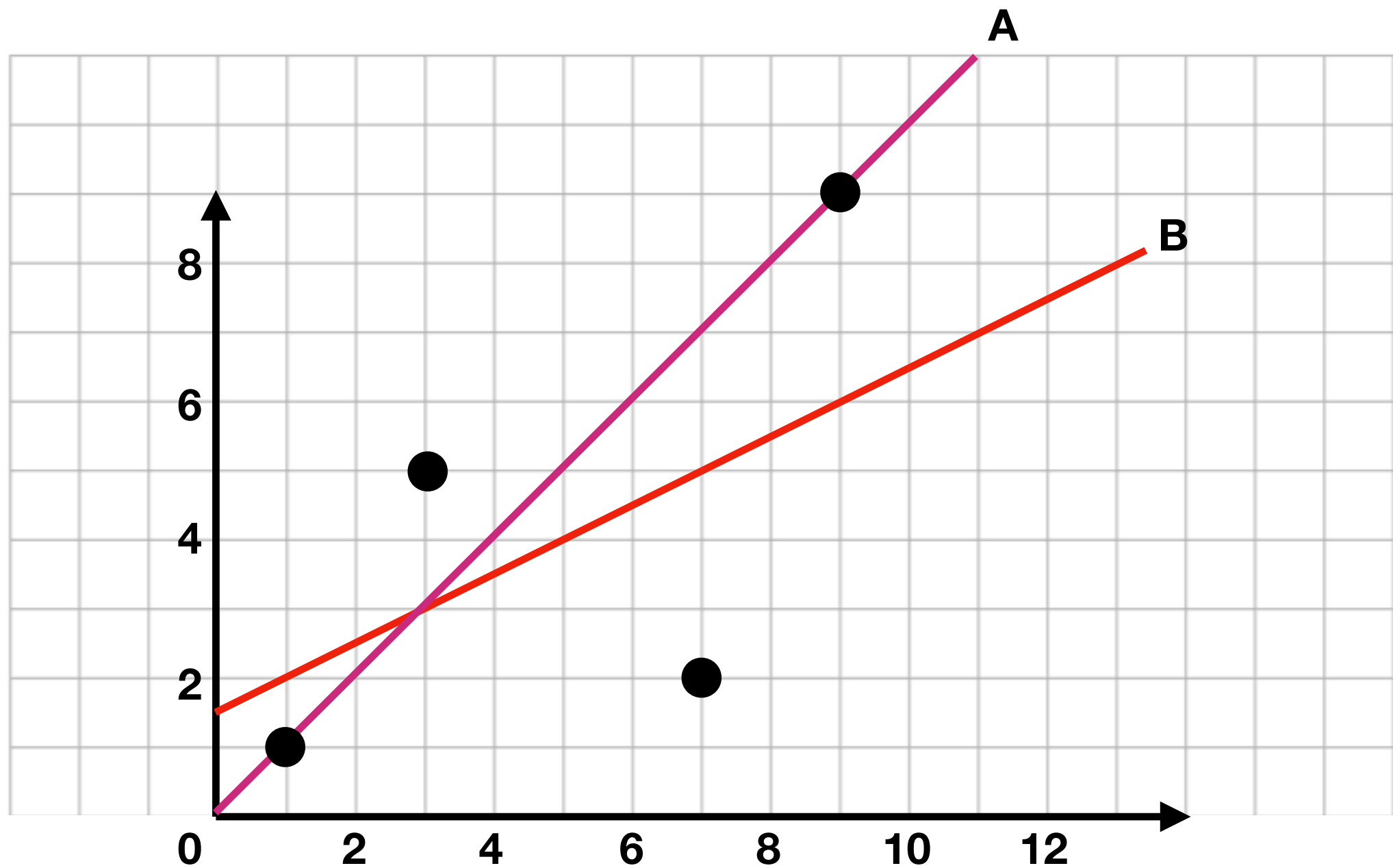
Intuition: Springs



Tension is defined as distance squared

Total: $4+9 = 13$

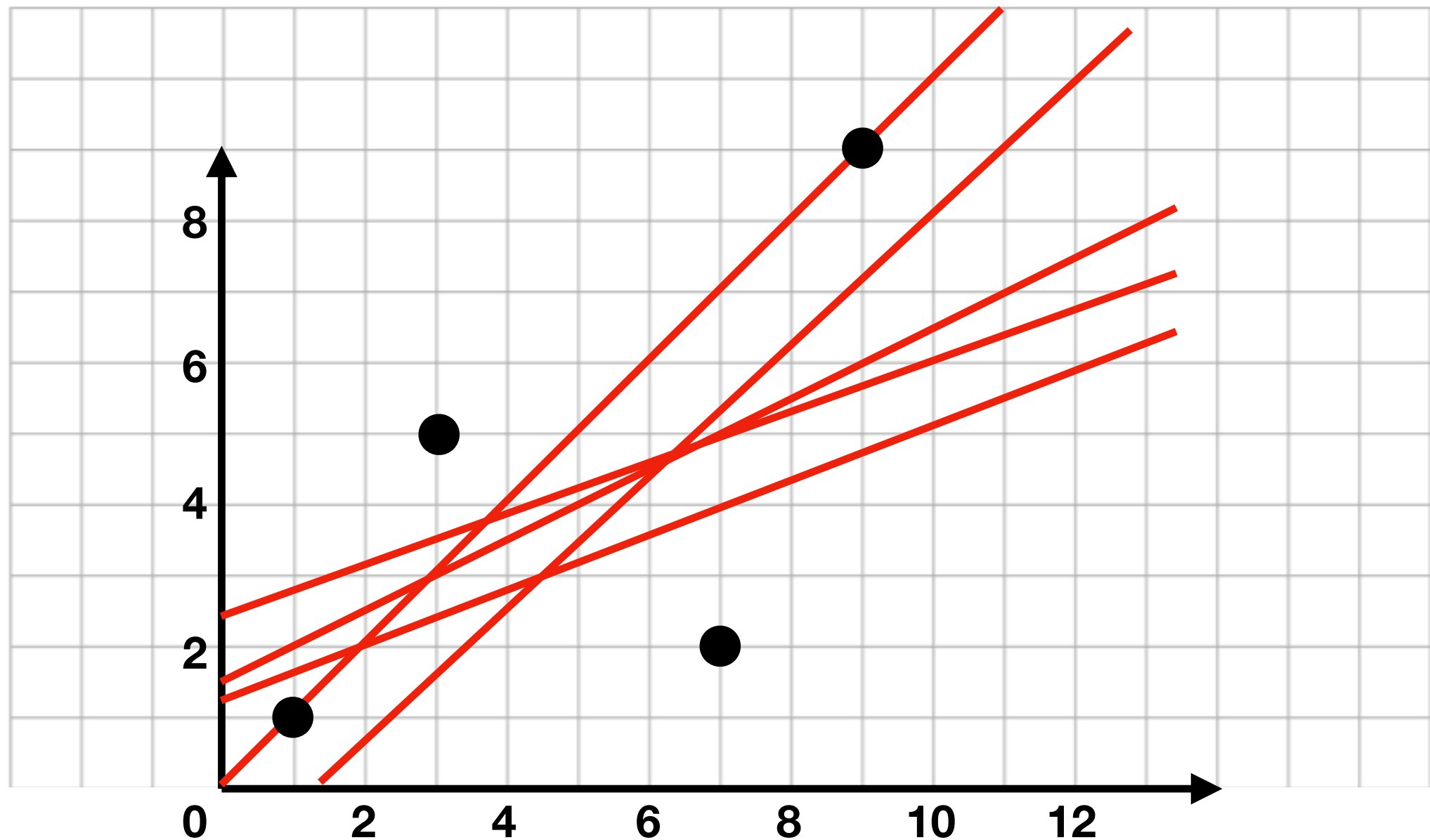
Practice



How much tension is in line A?

How much tension is in line B?

Optimization



There are many possible fit lines, but we want the one with the **minimal tension**.

Rather than crunch the numbers ourselves, we'll use a function from the **numpy** module

Learning Objectives Today

History of regression

Drawing a fit line

Finding the slope/intercept w/ least squares method

Numpy introduction

Using `numpy.linalg.lstsq`

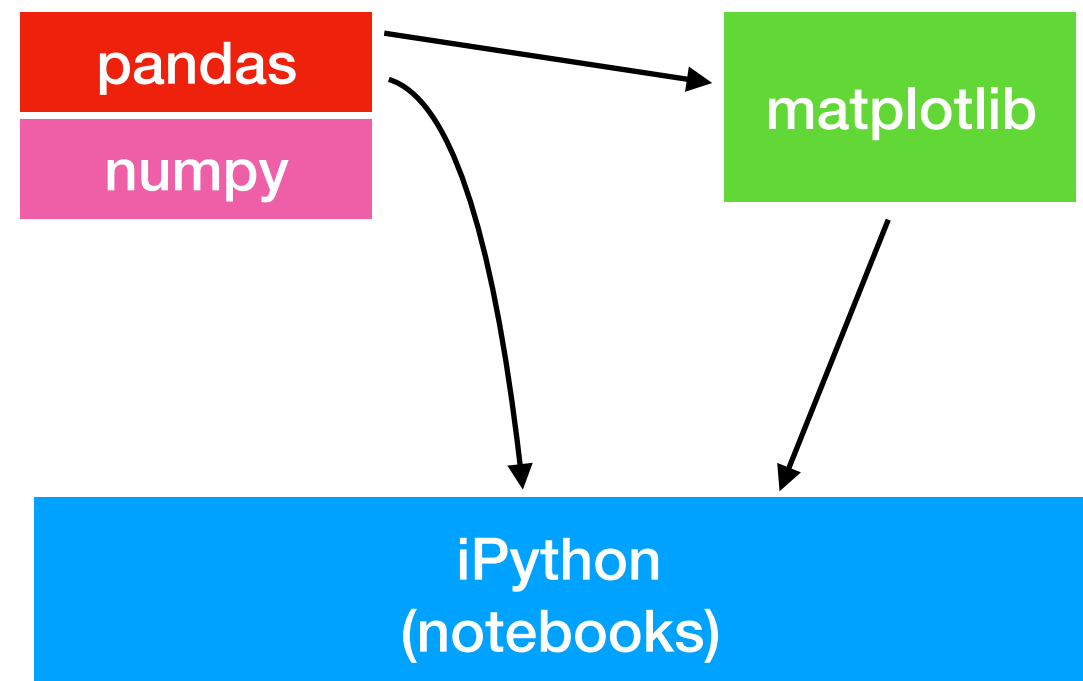
Modules we've learned this semester

- math
- collections
- json
- csv
- sys
- os
- copy
- recordclass
- requests
- bs4 (BeautifulSoup)
- pandas
- sqlite3
- matplotlib
- **numpy**  today

numpy is the second most popular
installed Python package after django
(by some measures)

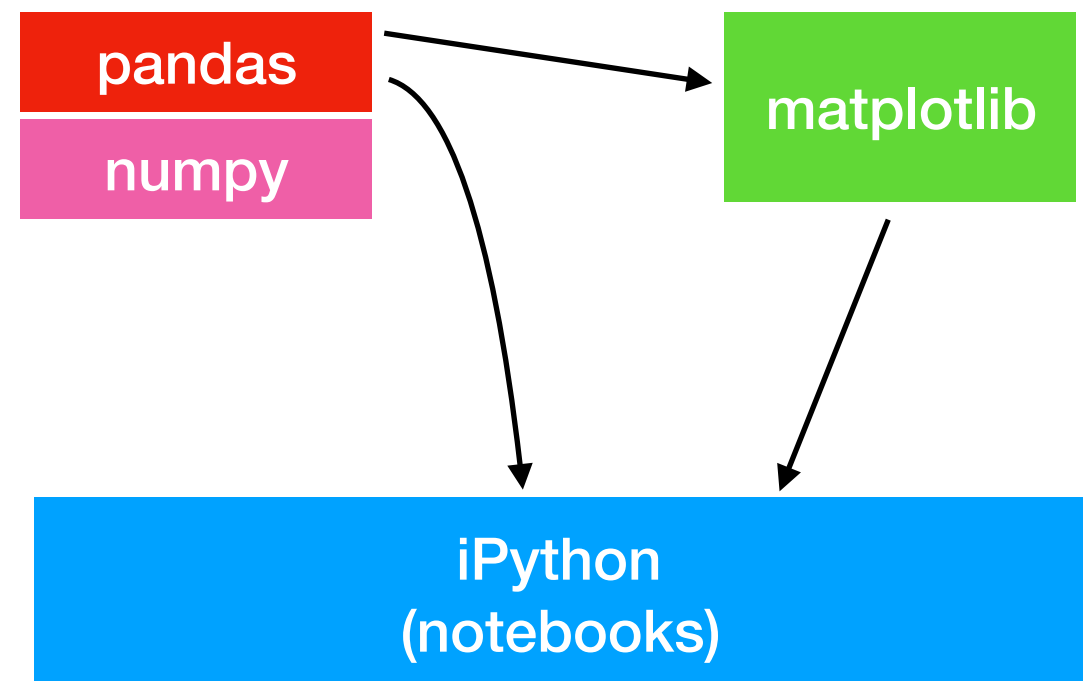
Modules we've learned this semester

- math
- collections
- json
- csv
- sys
- os
- copy
- recordclass
- requests
- bs4 (BeautifulSoup)
- pandas
- sqlite3
- matplotlib
- **numpy** ← today



Modules we've learned this semester

- math
- collections
- json
- csv
- sys
- os
- copy
- recordclass
- requests
- bs4 (BeautifulSoup)
- pandas
- sqlite3
- matplotlib
- **numpy** ← today



pandas Series and DataFrames use numpy, so you've been using it too without realizing it

numpy

```
import numpy as np
```

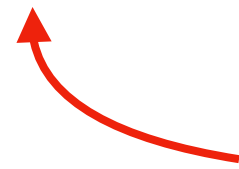


conventional alias

numpy

```
import numpy as np
```

```
a = np.array([10, 20, 30])
```



**ndarray (created by this call)
is the core data structure of numpy**

numpy

```
import numpy as np
```

it can be initialized
from a **Python list**



```
a = np.array([10, 20, 30])
```



ndarray (created by this call)
is the core data structure of numpy

numpy

```
import numpy as np
```

```
a = np.array([10, 20, 30])
```

```
a[1]
```



indexing

20

numpy

```
import numpy as np
```

```
a = np.array([10, 20, 30])
```

```
a[-1]
```



negative works

30

numpy

```
import numpy as np
```

```
a = np.array([10, 20, 30])
```

```
a[1:]
```



slicing

```
array([20, 30])
```

numpy

```
import numpy as np
```

```
a = np.array([10, 20, 30])
```

```
a + 1
```



element-wise ops

```
array([11, 21, 31])
```

numpy

```
import numpy as np
```

```
a = np.array([10, 20, 30])
```

```
a + a
```



element-wise ops

```
array([20, 40, 60])
```

numpy

```
import numpy as np
```

```
a = np.array([10, 20, 30])
```

```
a * a
```



element-wise ops

```
array([100, 400, 900])
```

numpy

```
import numpy as np
```

```
a = np.array([10, 20, 30])
```

```
print(type(a))
```



```
numpy.ndarray
```

numpy

```
import numpy as np
```

```
a = np.array([10, 20, 30])
```

```
print(type(a))
```



`numpy.ndarray`



why is it called an ndarray?

numpy

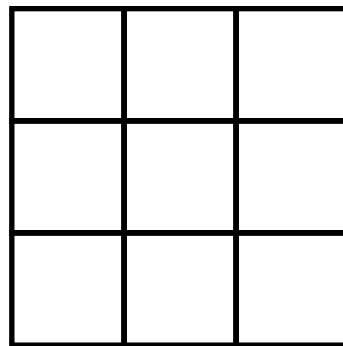
```
import numpy as np
```

```
a = np.array([10, 20, 30])
```

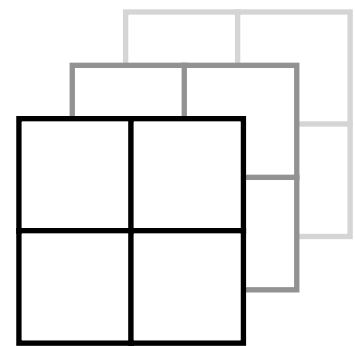
1-dimensional array



2-dimensional array



3-dimensional array



numpy

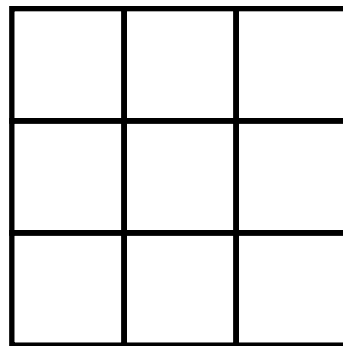
```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

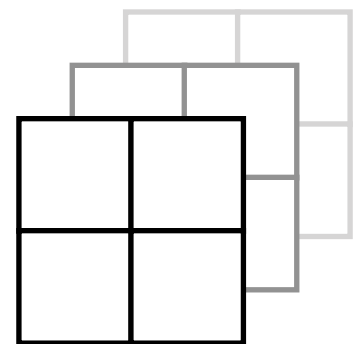
1-dimensional array



2-dimensional array



3-dimensional array



numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
a.reshape( )
```

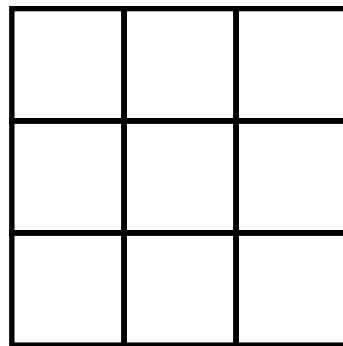
shape tuple



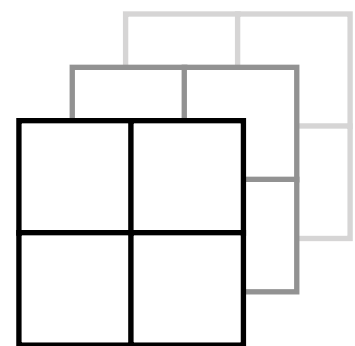
1-dimensional array



2-dimensional array



3-dimensional array

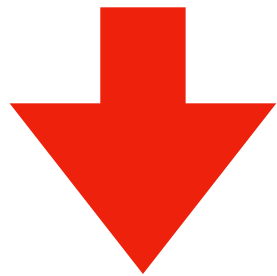


numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
a.reshape((2,4))
```



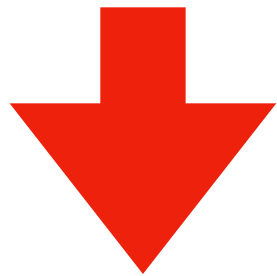
11	12	13	14
15	16	17	18

numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
a.reshape((4,2))
```



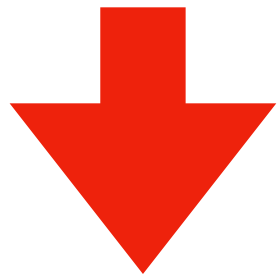
11	12
13	14
15	16
17	18

numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
a.reshape((4,2))
```



11	12
13	14
15	16
17	18

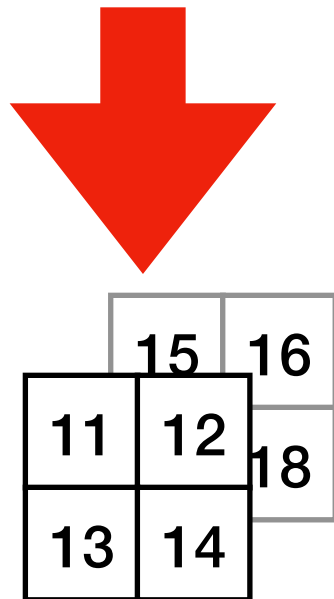
**note that reshape fills in
row-by-row first by default**

numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
a.reshape((2,2,2))
```

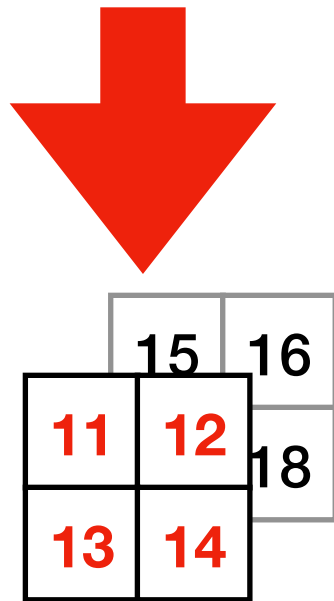


numpy

```
import numpy as np
```

```
a = np.array([11, 12, 13, 14, 15, 16, 17, 18])
```

```
a.reshape((2, 2, 2))
```



default fill order:

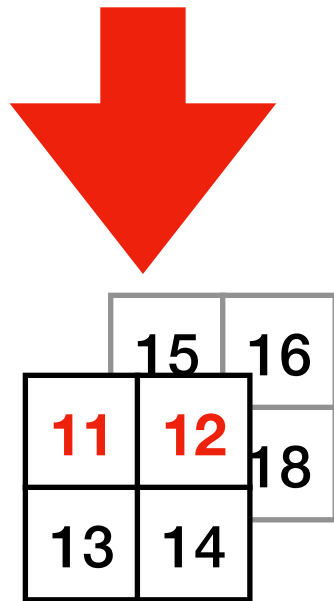
- layers
- rows
- columns

numpy

```
import numpy as np
```

```
a = np.array([11, 12, 13, 14, 15, 16, 17, 18])
```

```
a.reshape((2, 2, 2))
```



default fill order:

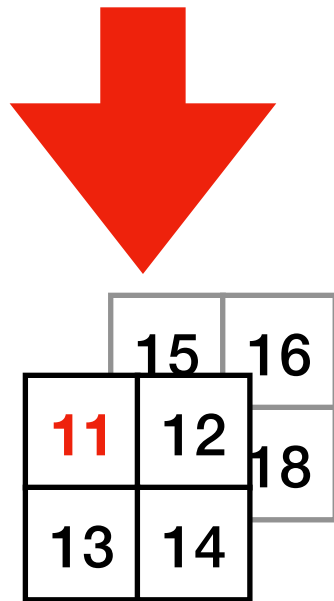
- layers
- **rows**
- columns

numpy

```
import numpy as np
```

```
a = np.array([11, 12, 13, 14, 15, 16, 17, 18])
```

```
a.reshape((2, 2, 2))
```



default fill order:

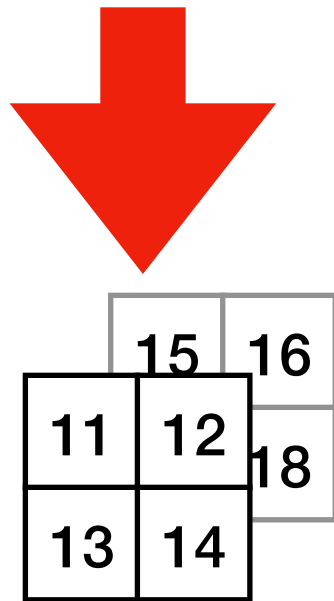
- layers
- rows
- **columns**

numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
a.reshape((2,2,2))
```



default fill order:

- layers
- rows
- columns

numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
b = a.reshape((2,2,2))
```

	15	16
11	12	18
13	14	

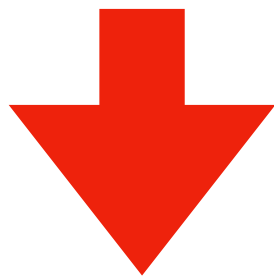
numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
b = a.reshape((2,2,2))
```

```
b[0]
```



11	12
13	14

	15	16
11	12	18
13	14	

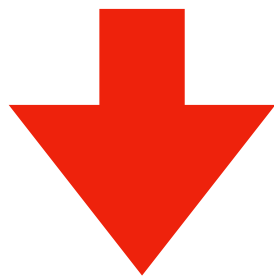
numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
b = a.reshape((2,2,2))
```

```
b[1]
```



15	16
17	18

11	12	15	16
13	14	17	18

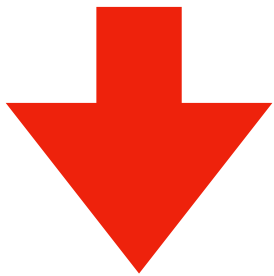
numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
b = a.reshape((2,2,2))
```

```
b[0,0,0]
```



11

	15	16
11	12	18
13	14	

indexing: ndarray[layer, row, col]

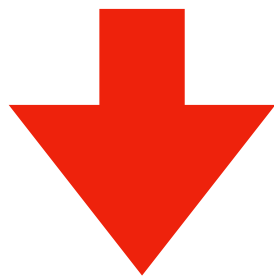
numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
b = a.reshape((2,2,2))
```

```
b[0,0,0]
```



11

	15	16
11	12	18
13	14	

indexing: `ndarray[layer, row, col]`

contrast with indexing into a list of lists of lists:

`data[layer][row][col]`

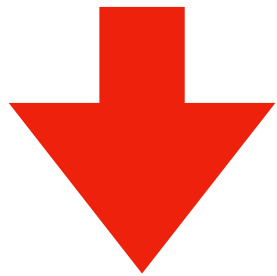
numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
b = a.reshape((2,2,2))
```

```
b[0,0,1]
```



12

	15	16
11	12	18
13	14	

indexing: ndarray[layer, row, col]

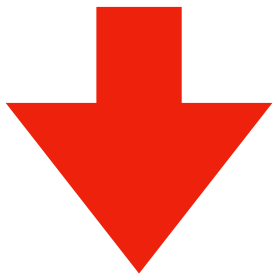
numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
b = a.reshape((2,2,2))
```

```
b[0,1,1]
```



???

	15	16
11	12	18
13	14	

indexing: ndarray[layer, row, col]

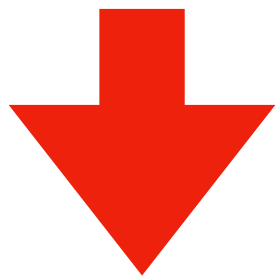
numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
b = a.reshape((2,2,2))
```

```
b[0,1,1]
```



14

	15	16
11	12	18
13	14	

indexing: ndarray[layer, row, col]

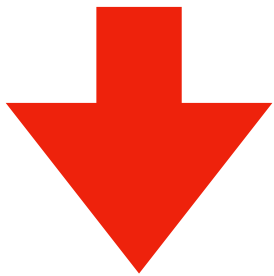
numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
b = a.reshape((2,2,2))
```

```
b[1,1,1]
```



???

	15	16
11	12	18
13	14	

indexing: ndarray[layer, row, col]

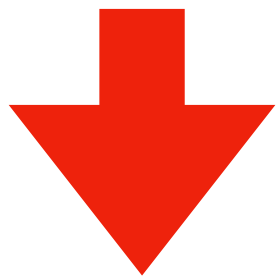
numpy

```
import numpy as np
```

```
a = np.array([11,12,13,14,15,16,17,18])
```

```
b = a.reshape((2,2,2))
```

```
b[1,1,1]
```



18

	15	16
11	12	18
13	14	

indexing: ndarray[layer, row, col]

Pandas and numpy

```
df = DataFrame({"a": [1, 2], "b": [3, 4]})
```

	a	b
0	1	3
1	2	4

```
s = df["a"]
```

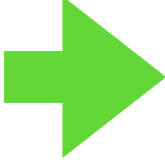
Pandas and numpy

```
df = DataFrame({"a": [1, 2], "b": [3, 4]})
```

	a	b
0	1	3
1	2	4

```
s = df["a"]
```

`df.values`  `array([[1, 3],
[2, 4]])`

`s.values`  `array([3, 4])`

Pandas and numpy

```
df = DataFrame({"a": [1, 2], "b": [3, 4]})
```

	a	b
0	1	3
1	2	4

you've been using
numpy arrays without
knowing it!

```
s = df["a"]
```

```
type(df.values) → numpy.ndarray
```


```
type(s.values) → numpy.ndarray
```

Pandas and numpy

```
df = DataFrame({"a": [1, 2], "b": [3, 4]})
```

	a	b
0	1	3
1	2	4

```
s = df["a"]
```

df.shape  (2, 2)


s.shape  (2,)

Pandas and numpy

```
df = DataFrame({"a": [1, 2], "b": [3, 4]})
```

	a	b
0	1	3
1	2	4

```
s = df["a"]
```

`df.shape`  `(2, 2)`

`s.shape`  `(2,)`

`(2,)` is a tuple with
one number in it

Learning Objectives Today

History of regression

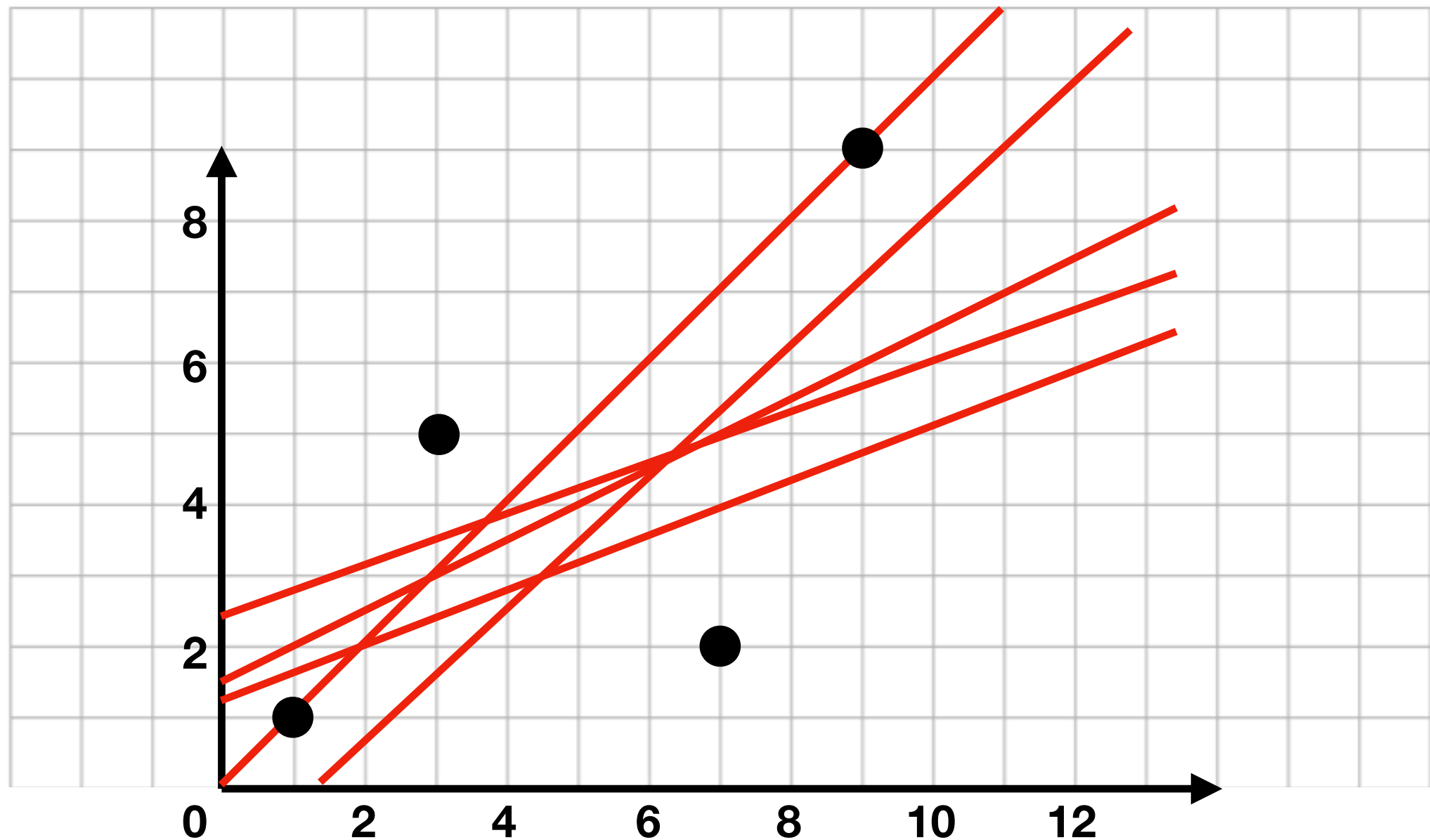
Drawing a fit line

Finding the slope/intercept w/ least squares method

Numpy introduction

Using `numpy.linalg.lstsq`

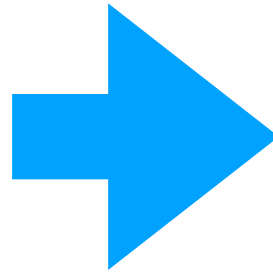
Use numpy to solve this!



There are many possible fit lines, but we want the one with the **minimal tension**.

Example data

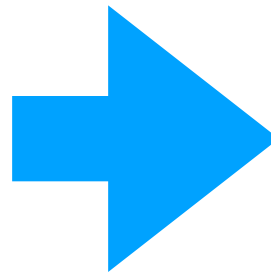
```
df = DataFrame({  
    "x": [1,2,3,4],  
    "y": [2,5,6,5]  
})
```



	x	y
0	1	2
1	2	5
2	3	6
3	4	5

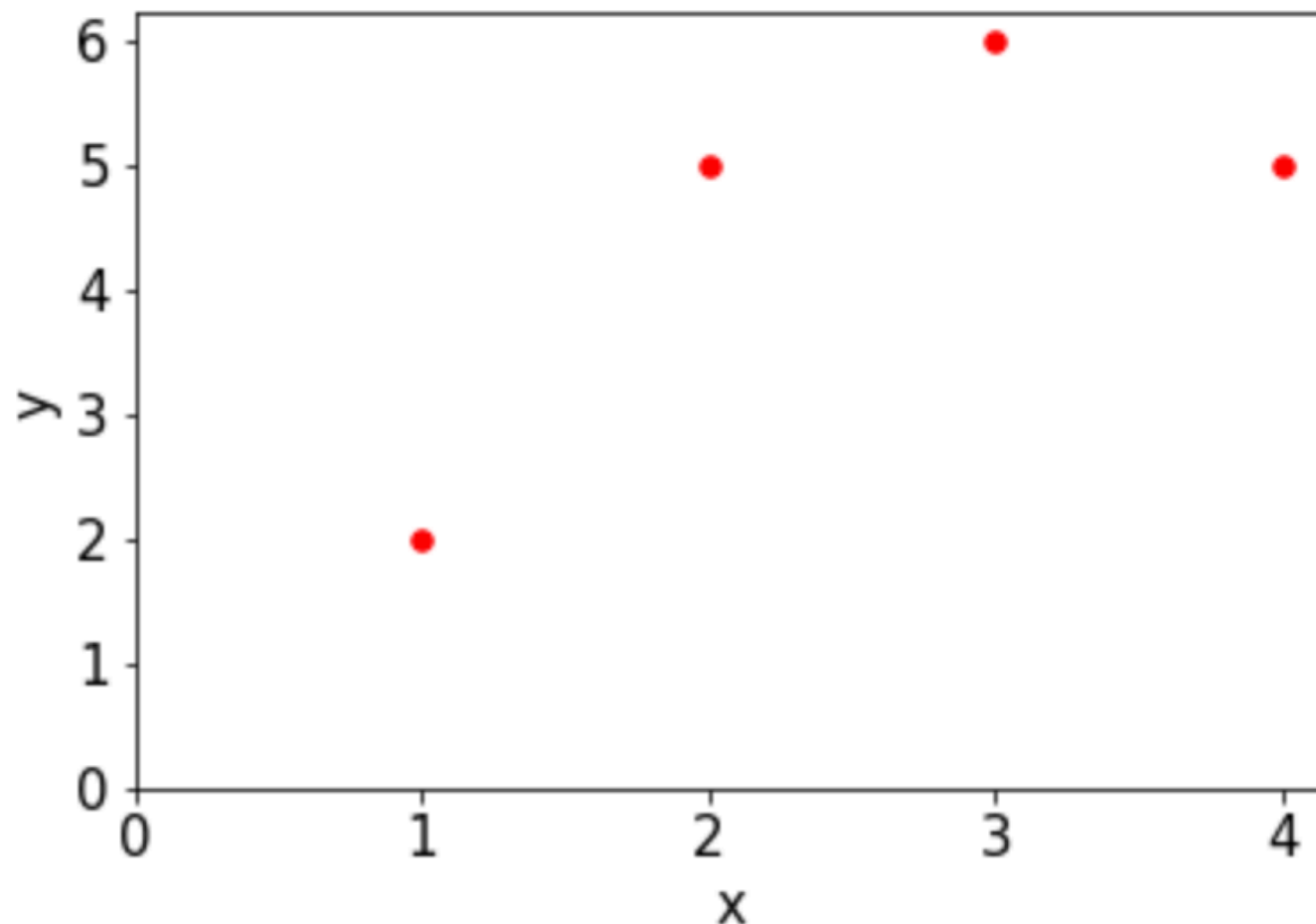
Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "y": [2,5,6,5]  
})
```



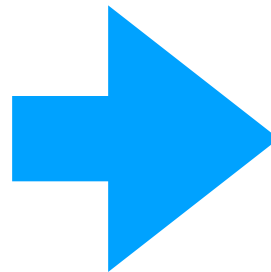
	x	y
0	1	2
1	2	5
2	3	6
3	4	5

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```



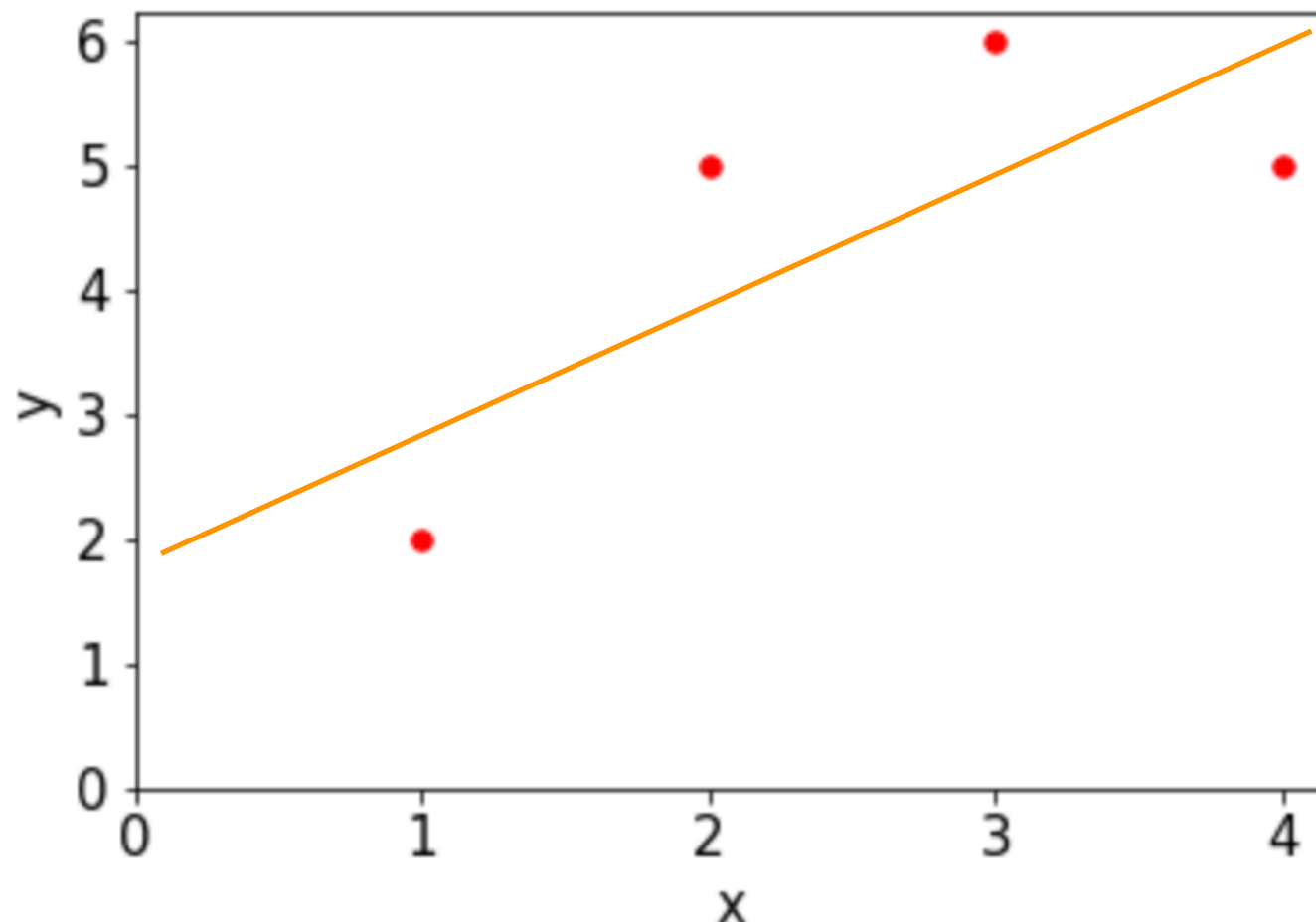
Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "y": [2,5,6,5]  
})
```



	x	y
0	1	2
1	2	5
2	3	6
3	4	5

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```

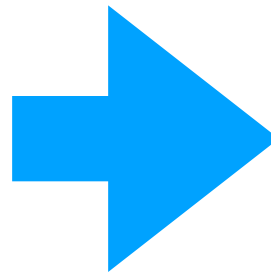


we want a formula like this:

$$m \cdot x + n = y$$

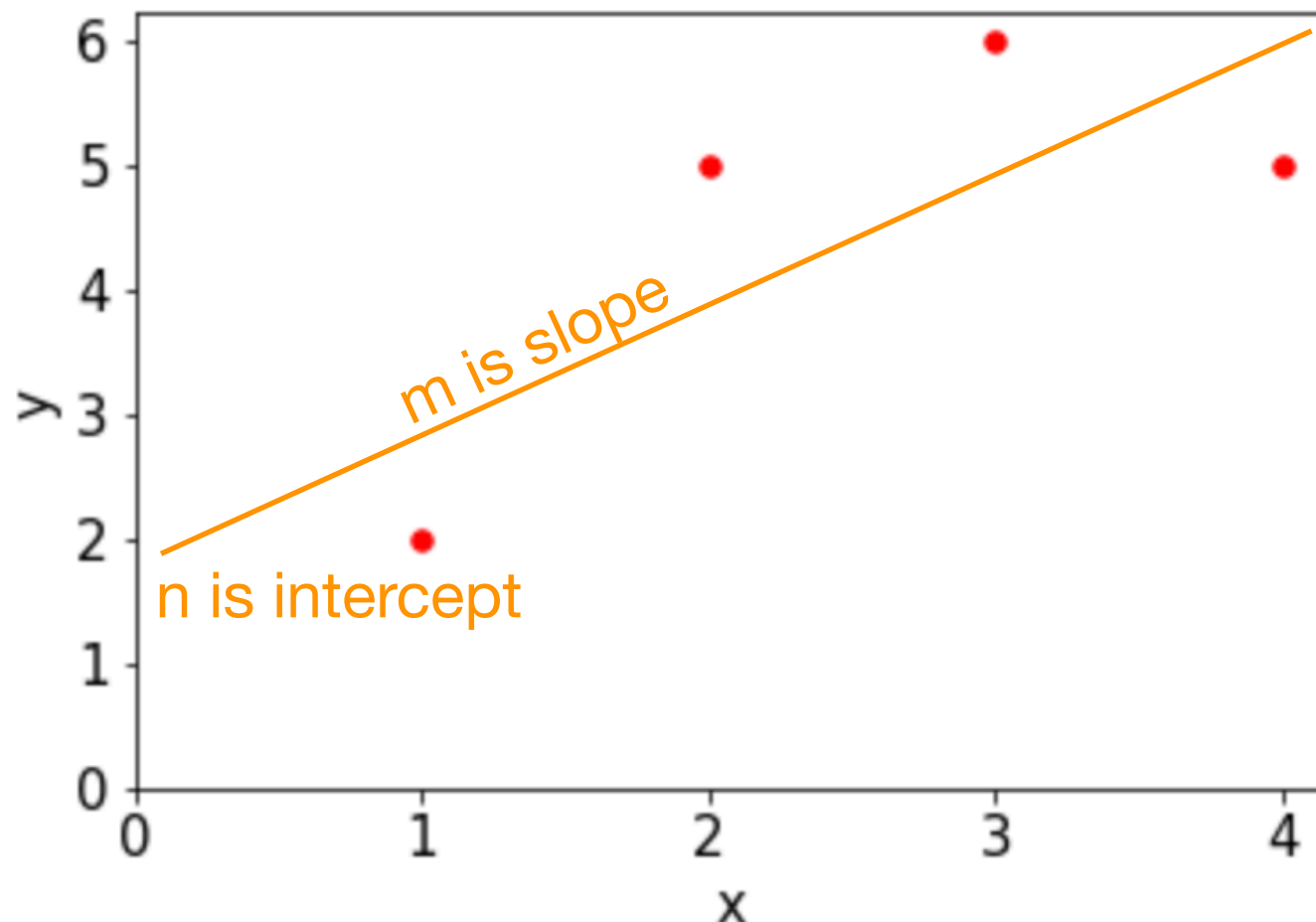
Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "y": [2,5,6,5]  
})
```



	x	y
0	1	2
1	2	5
2	3	6
3	4	5

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```

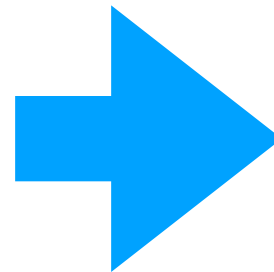


we want a formula like this:

$$m \cdot x + n = y$$

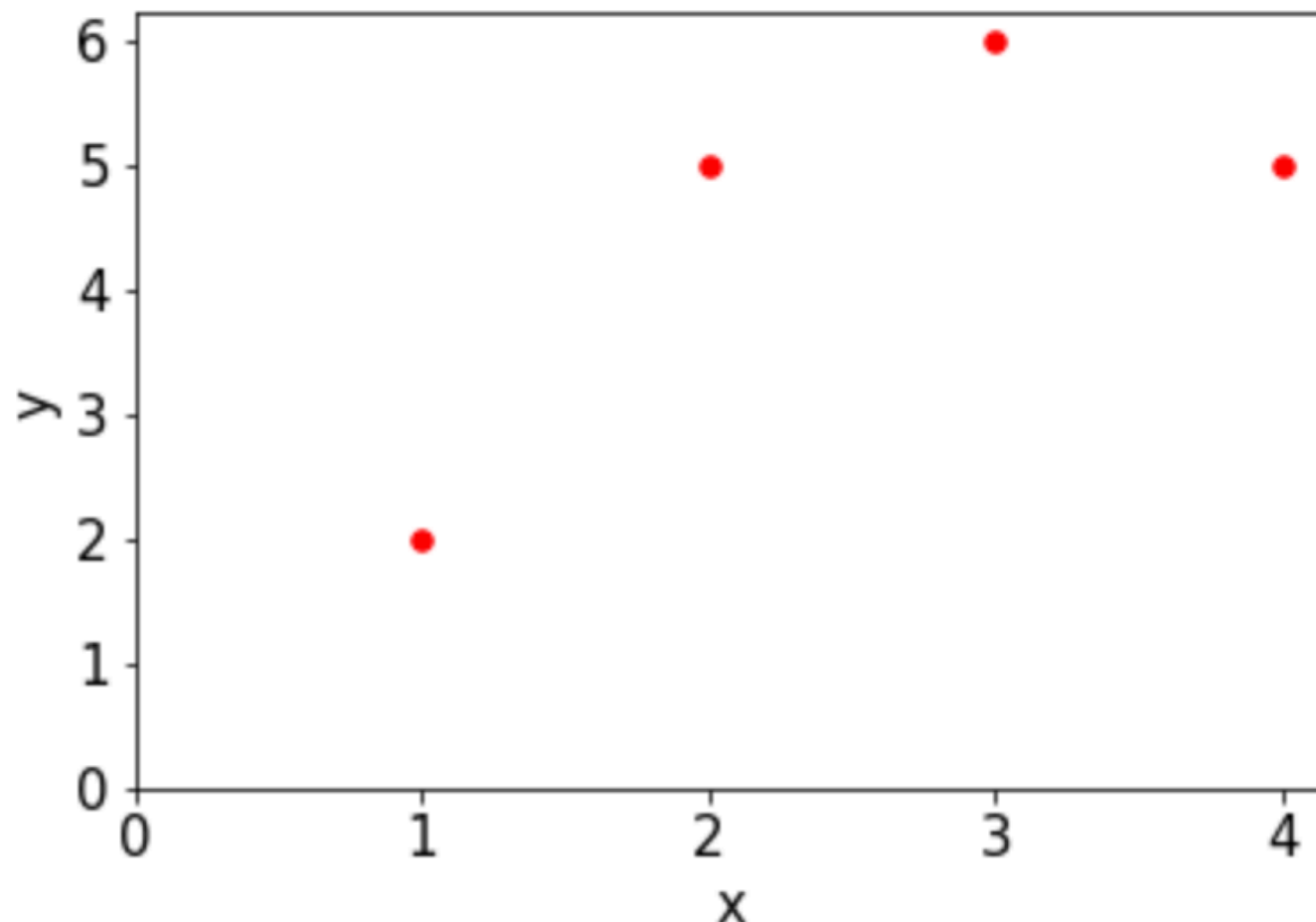
Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "y": [2,5,6,5]  
})
```



	x	y
0	1	2
1	2	5
2	3	6
3	4	5

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```

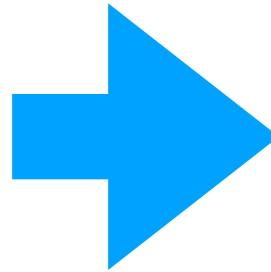


we want a formula like this:

$$m \cdot x + n = y$$

Example data

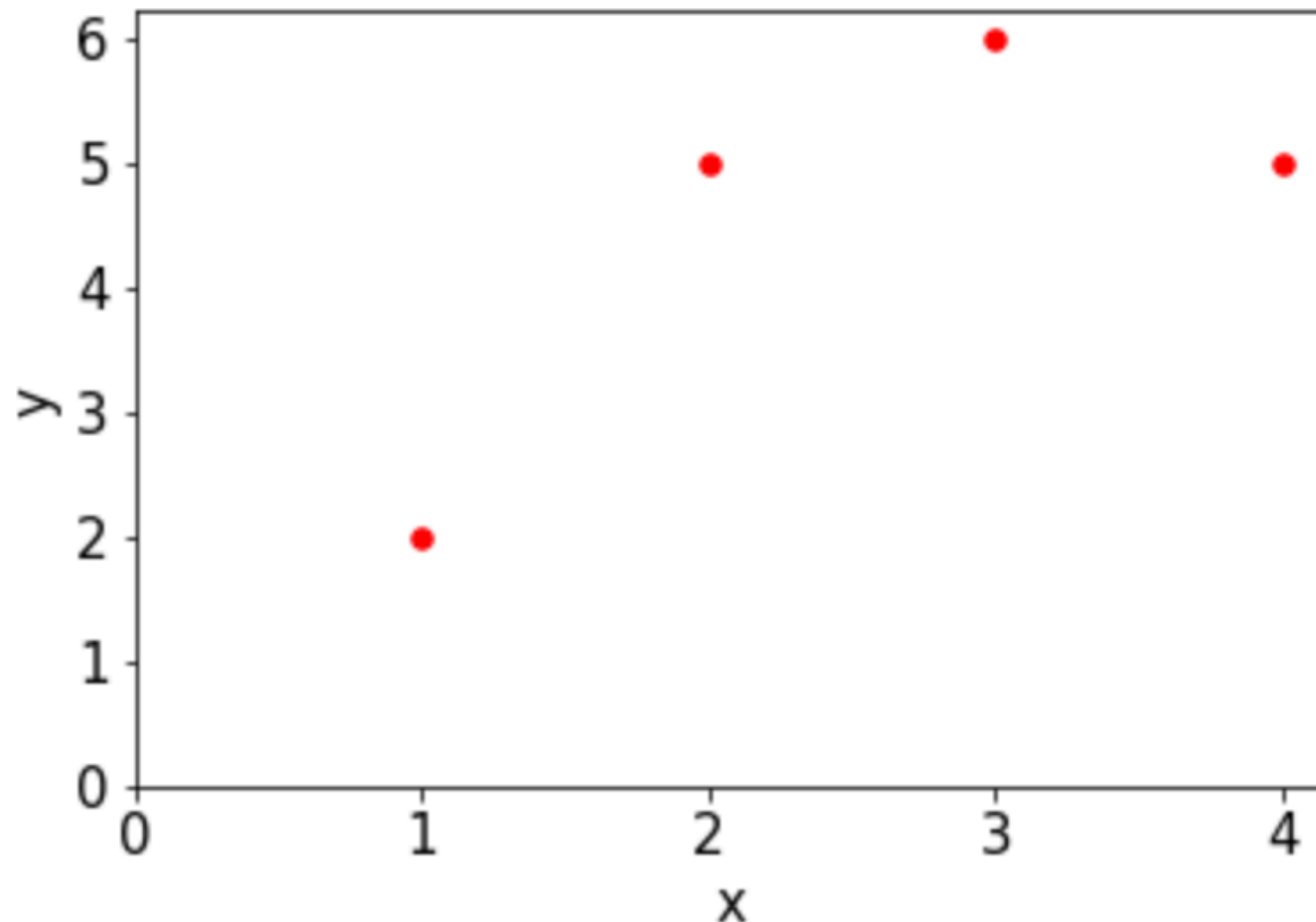
```
df = DataFrame({  
    "x": [1,2,3,4],  
    "y": [2,5,6,5]  
})
```



cut

	x	y
0	1	2
1	2	5
2	3	6
3	4	5

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```

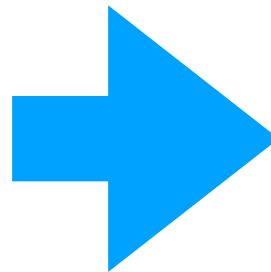


we want a formula like this:

$$m \cdot x + n = y$$

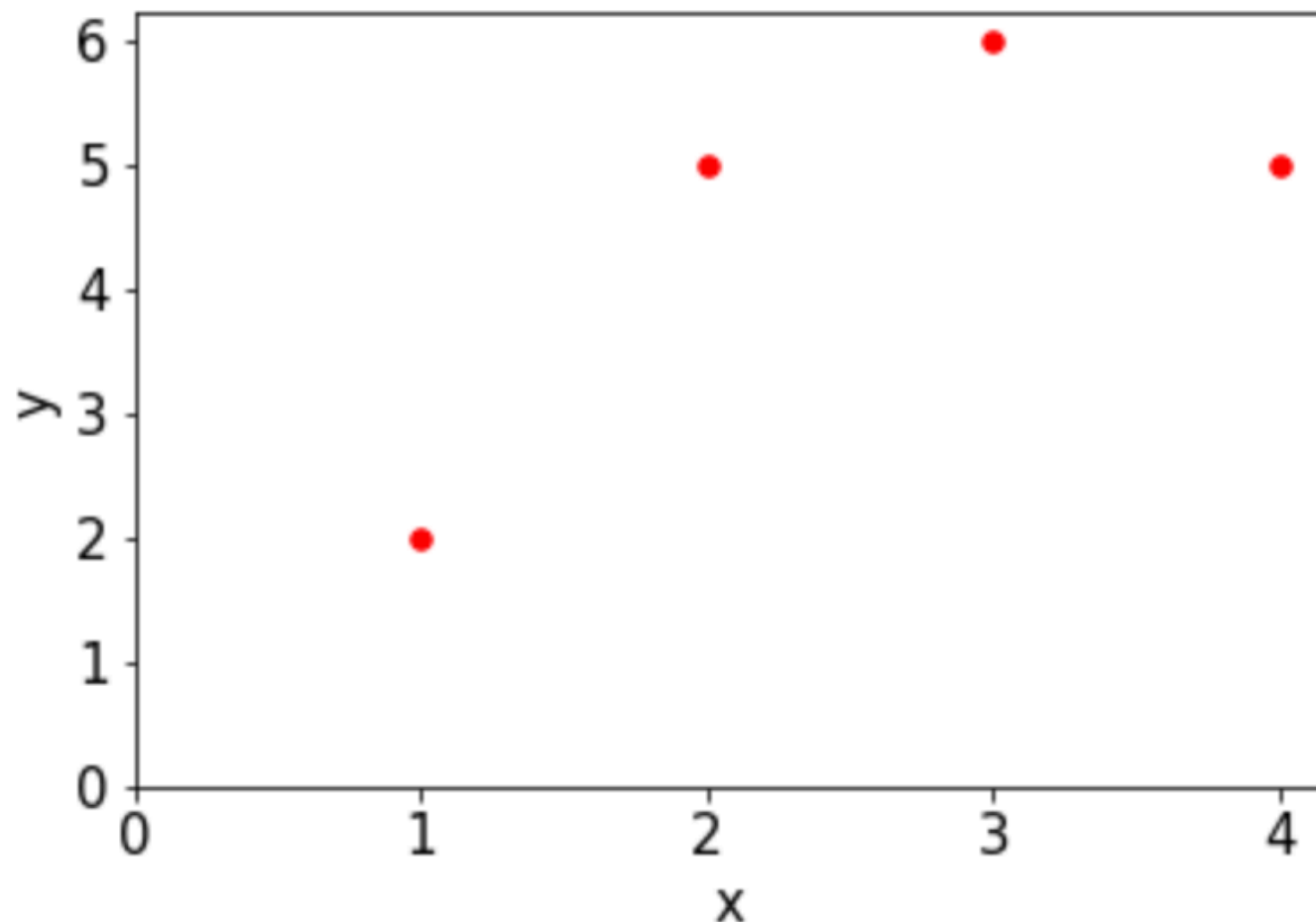
Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "y": [2,5,6,5]  
})
```



x	y
1	2
2	5
3	6
4	5

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```

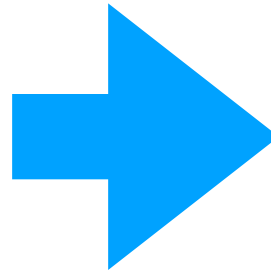


we want a formula like this:

$$m \cdot x + n = y$$

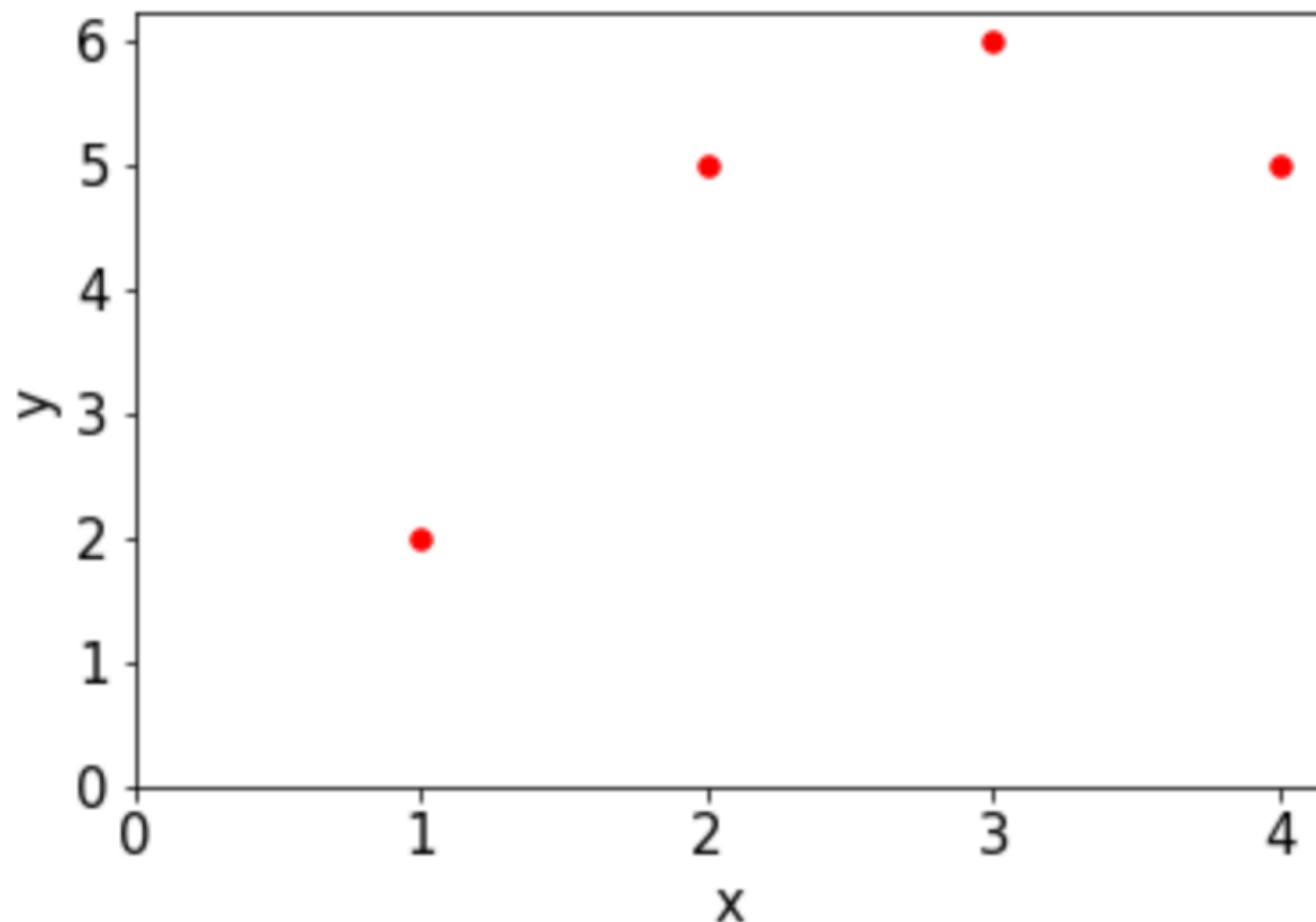
Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "y": [2,5,6,5]  
})
```



m^*	x	$+ n \approx$	y
	1		2
	2		5
	3		6
	4		5

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```

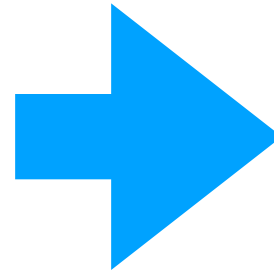


we want a formula like this:

$$m^*x + n = y$$

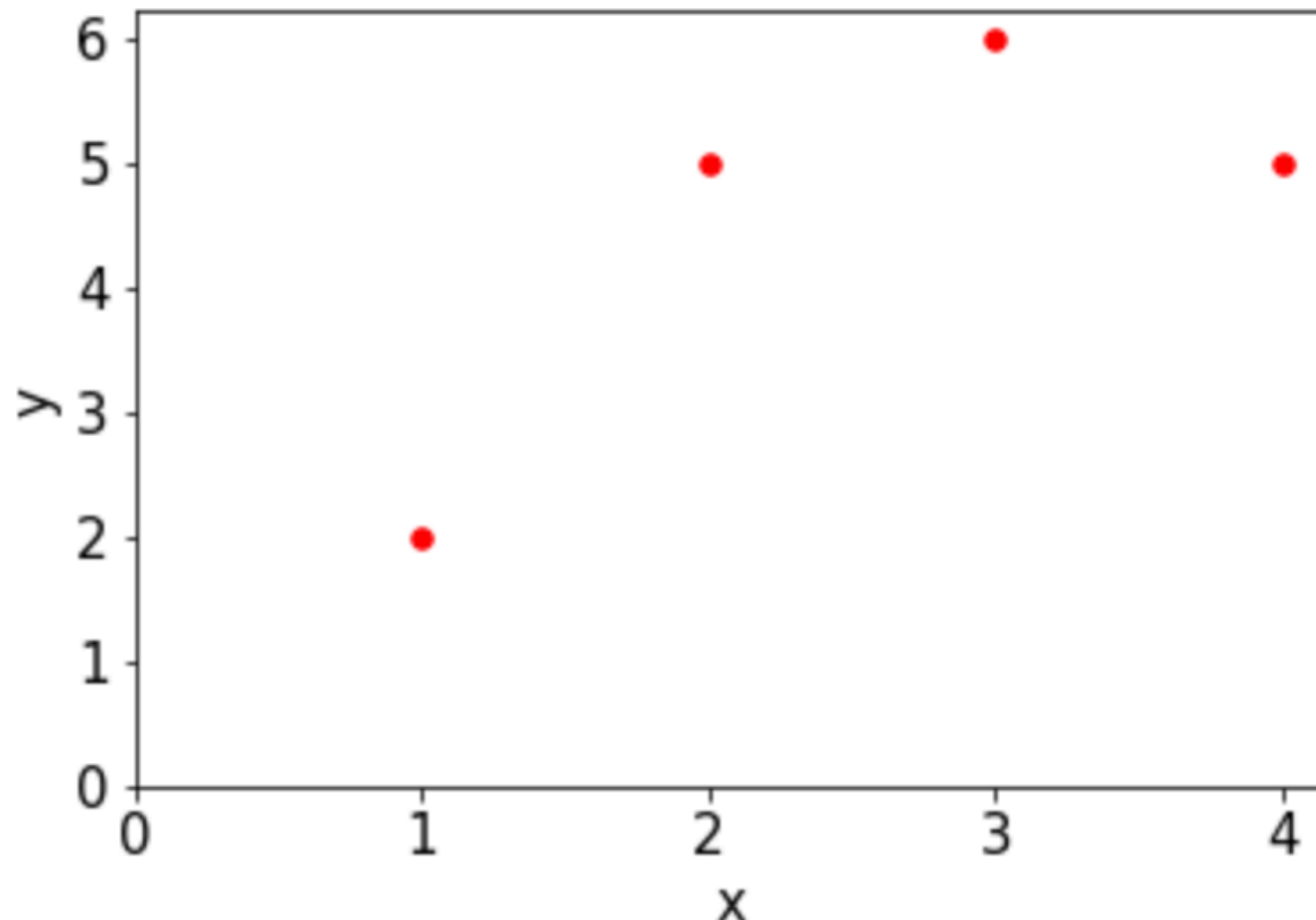
Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "y": [2,5,6,5]  
})
```



m^*	x	$+ n \approx$	y
	1		2
	2		5
	3		6
	4		5

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```



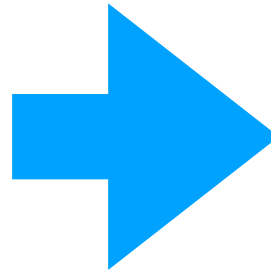
we want a formula like this:

$$m^*x + n = y$$

*when numpy solves for
a coefficient, it needs
to be multiplied by
some column in our data*

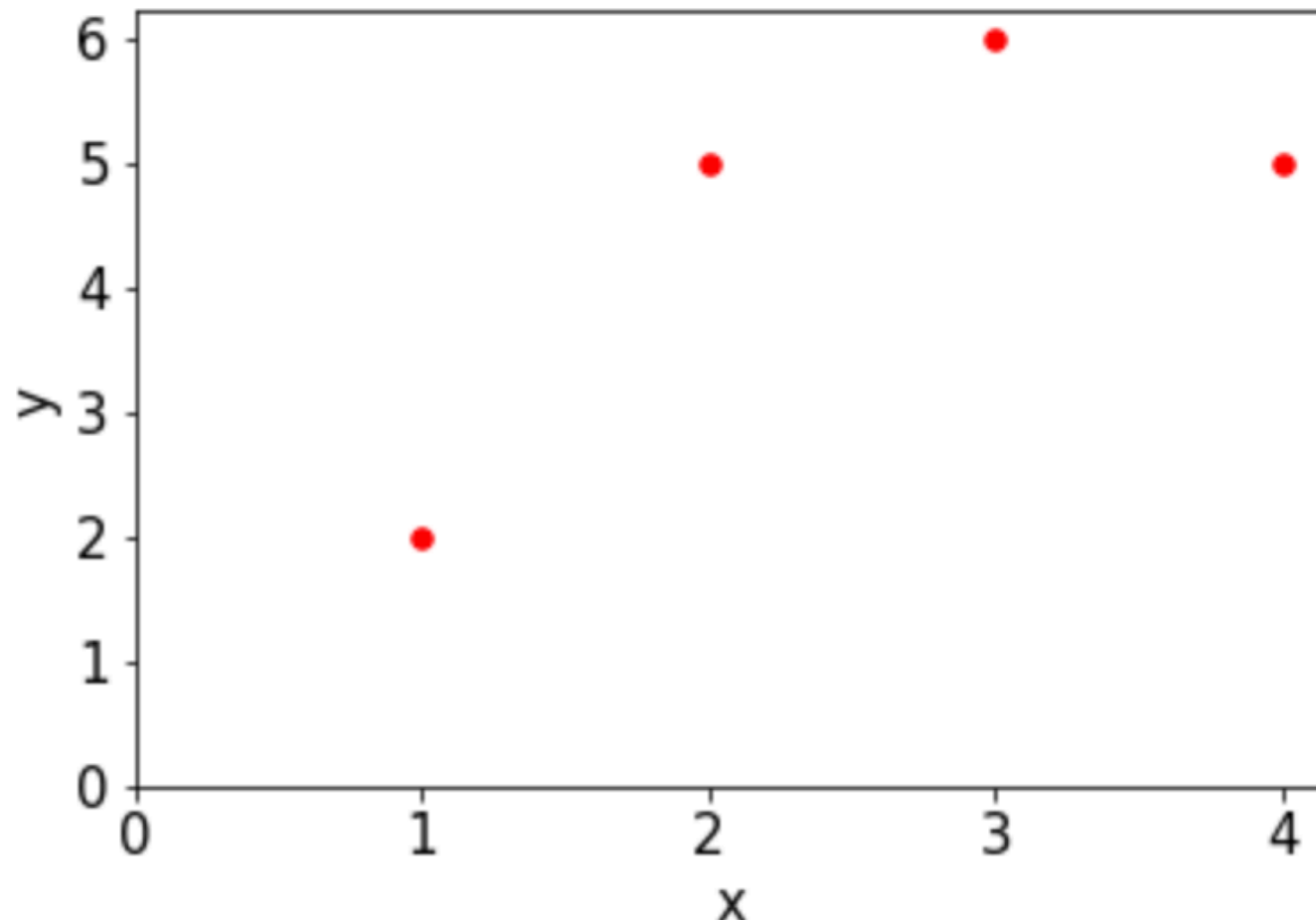
Example data

```
df = DataFrame({  
    "x": [1, 2, 3, 4],  
    "1": [1, 1, 1, 1],  
    "y": [2, 5, 6, 5]  
})
```



$$m * \begin{array}{|c|} \hline x \\ \hline 1 \\ 2 \\ 3 \\ 4 \\ \hline \end{array} + n * \begin{array}{|c|} \hline 1 \\ \hline 1 \\ 1 \\ 1 \\ 1 \\ \hline \end{array} \approx \begin{array}{|c|} \hline y \\ \hline 2 \\ 5 \\ 6 \\ 5 \\ \hline \end{array}$$

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```



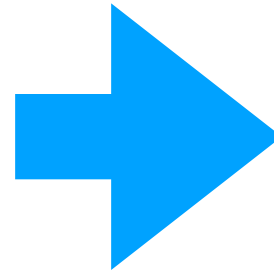
we want a formula like this:

$$m * x + n * 1 = y$$

*when numpy solves for
a coefficient, it needs
to be multiplied by
some column in our data*

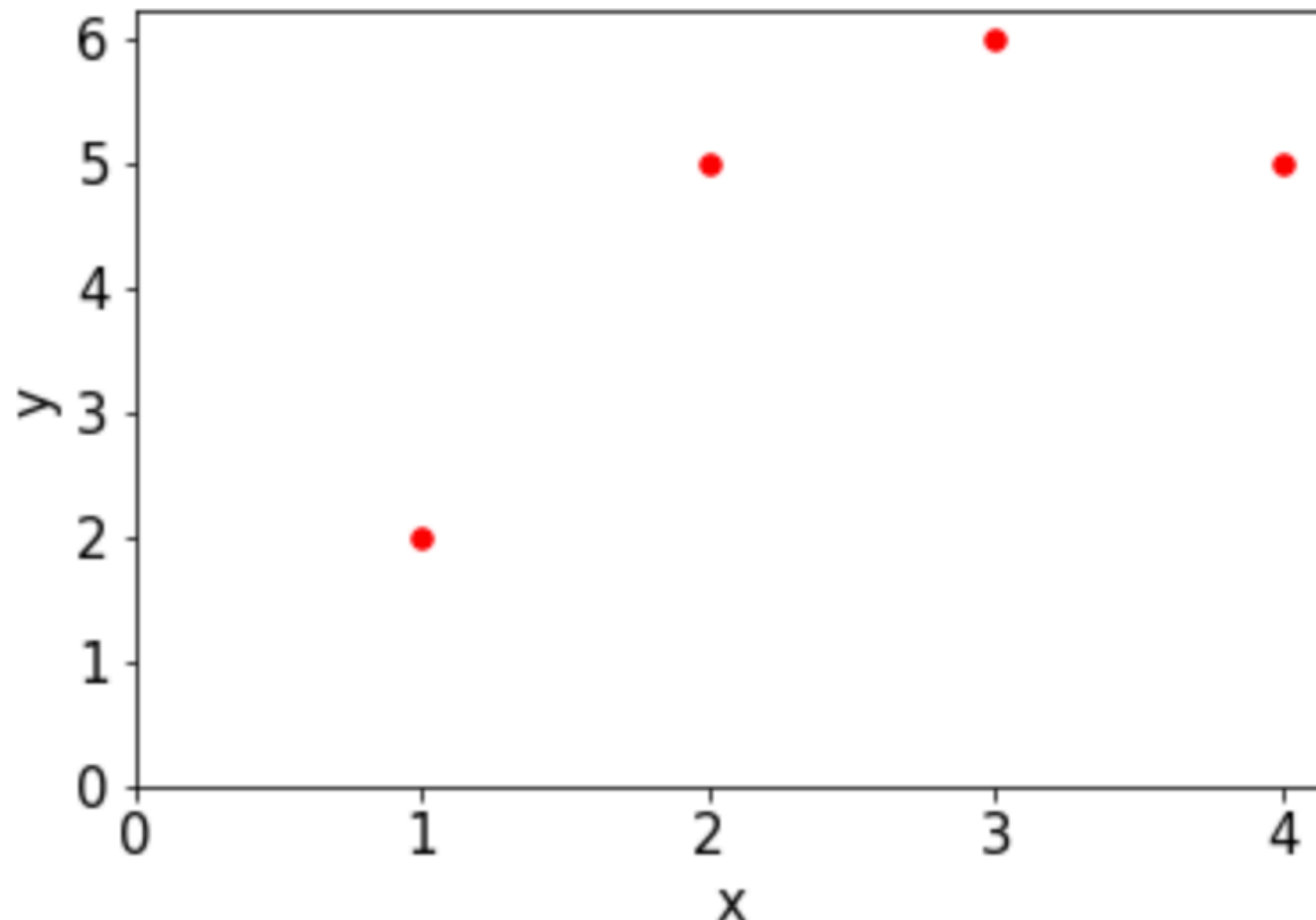
Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



$$m * \begin{array}{|c|} \hline x \\ \hline 1 \\ 2 \\ 3 \\ 4 \\ \hline \end{array} + n * \begin{array}{|c|} \hline 1 \\ \hline 1 \\ 1 \\ 1 \\ \hline \end{array} \approx \begin{array}{|c|} \hline y \\ \hline 2 \\ 5 \\ 6 \\ 5 \\ \hline \end{array}$$

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```



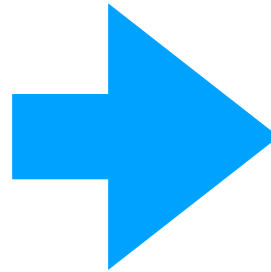
we want a formula like this:

$$m * x + n * 1 = y$$

*when numpy solves for
a coefficient, it needs
to be multiplied by
some column in our data*

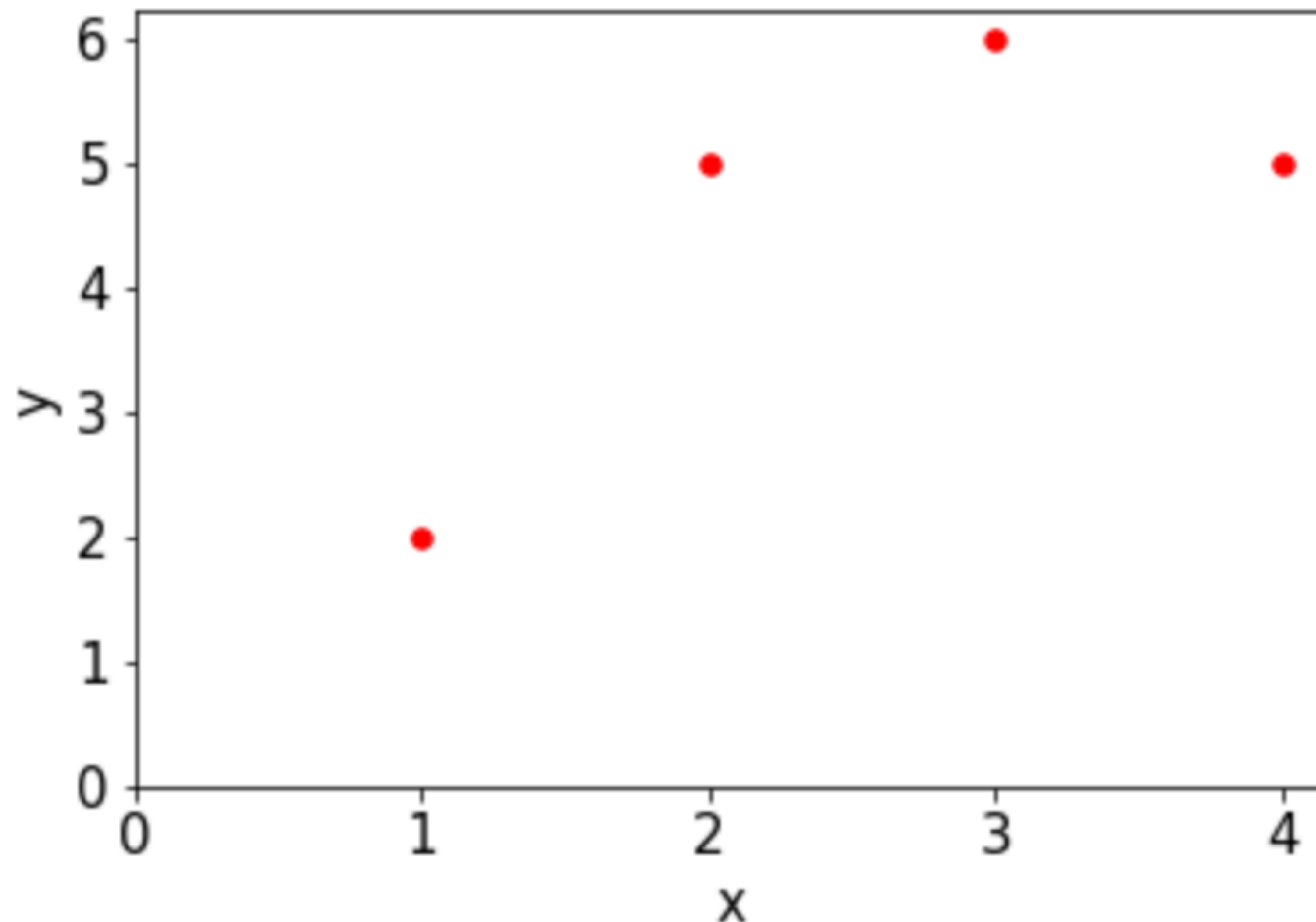
Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



$$m * \begin{array}{|c|} \hline x \\ \hline 1 \\ 2 \\ 3 \\ 4 \\ \hline \end{array} + n * \begin{array}{|c|} \hline 1 \\ \hline 1 \\ 1 \\ 1 \\ \hline \end{array} \approx \begin{array}{|c|} \hline y \\ \hline 2 \\ 5 \\ 6 \\ 5 \\ \hline \end{array}$$

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```

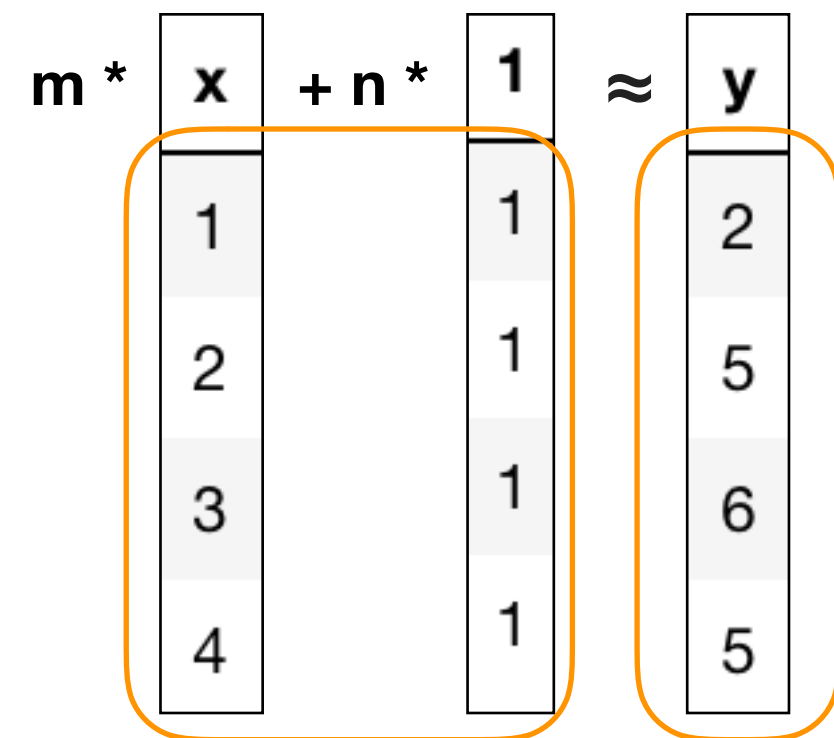
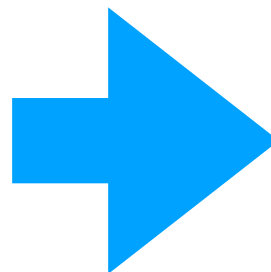


we want a formula like this:

$$m * x + n * 1 = y$$

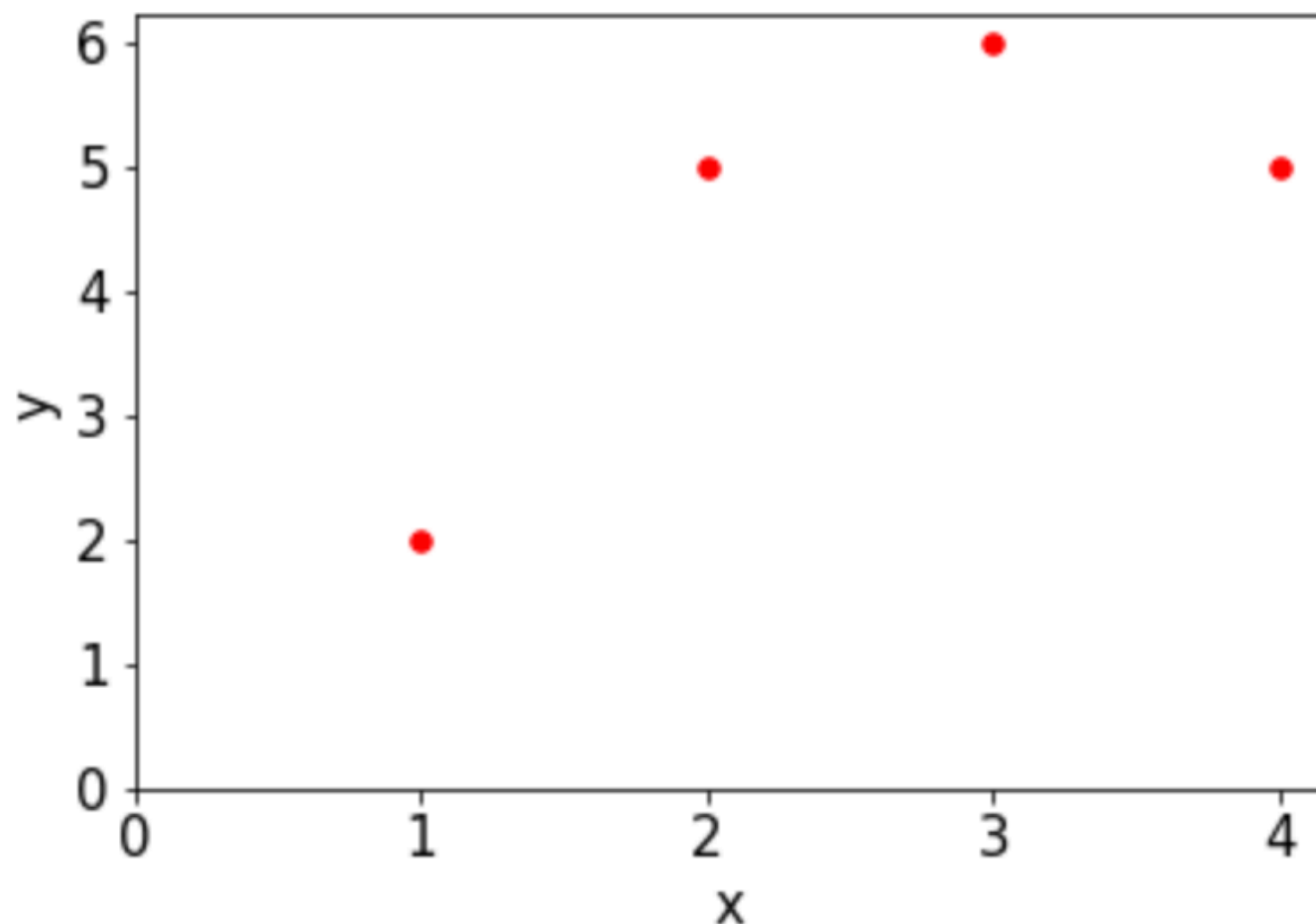
Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



given these inputs, as ndarrays...

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```

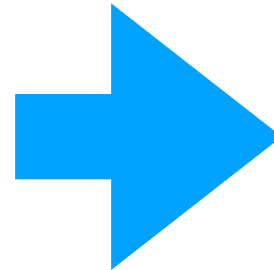


we want a formula like this:

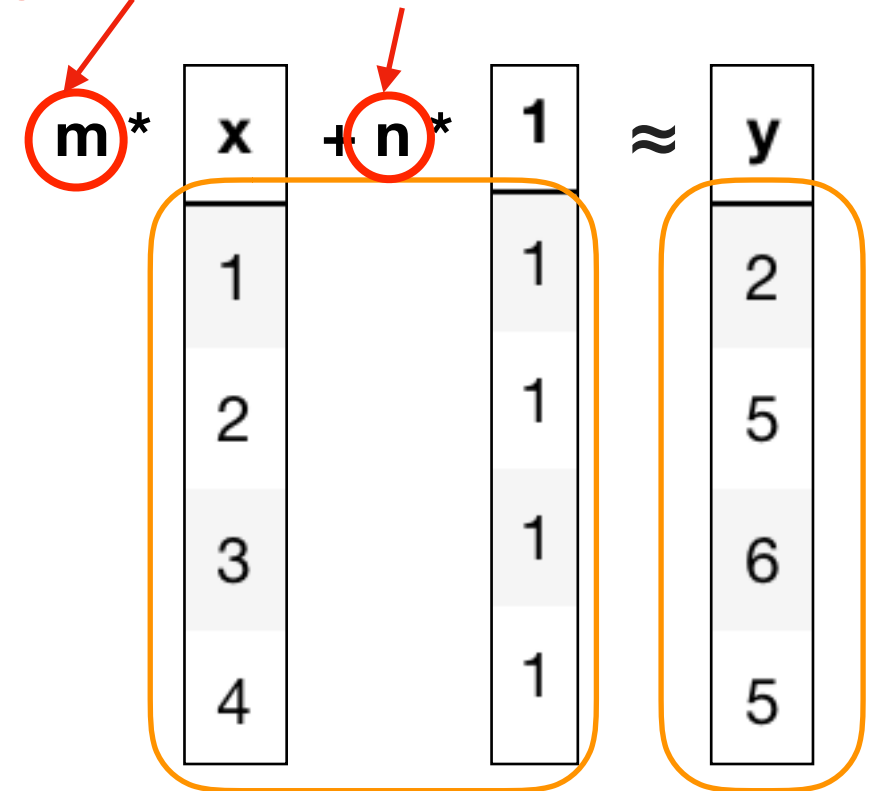
$$m \cdot x + n \cdot 1 = y$$

Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```

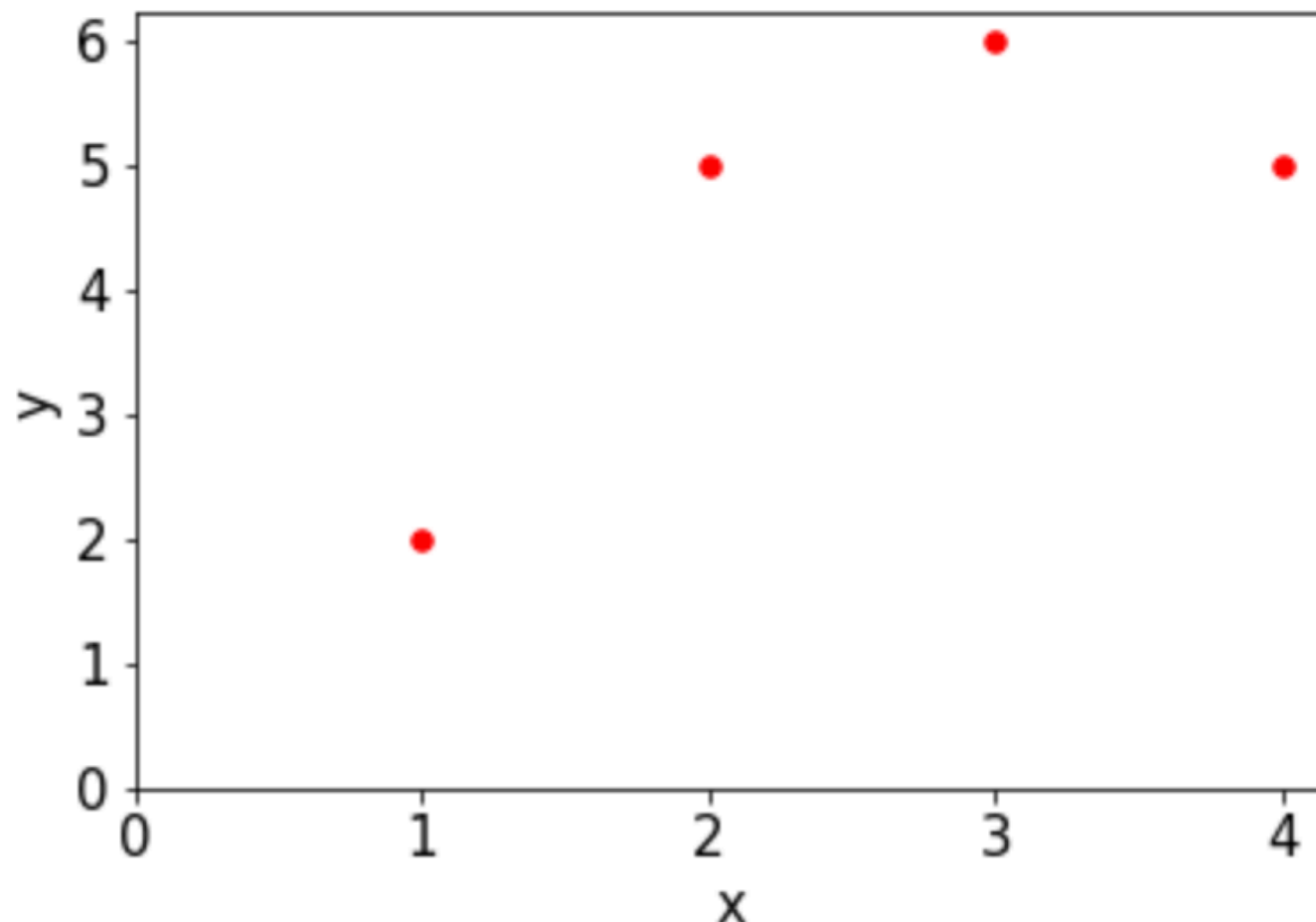


...numpy will give us these coefficients



given these inputs, as ndarrays...

```
df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```

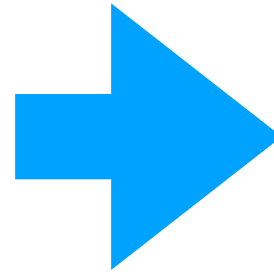


we want a formula like this:

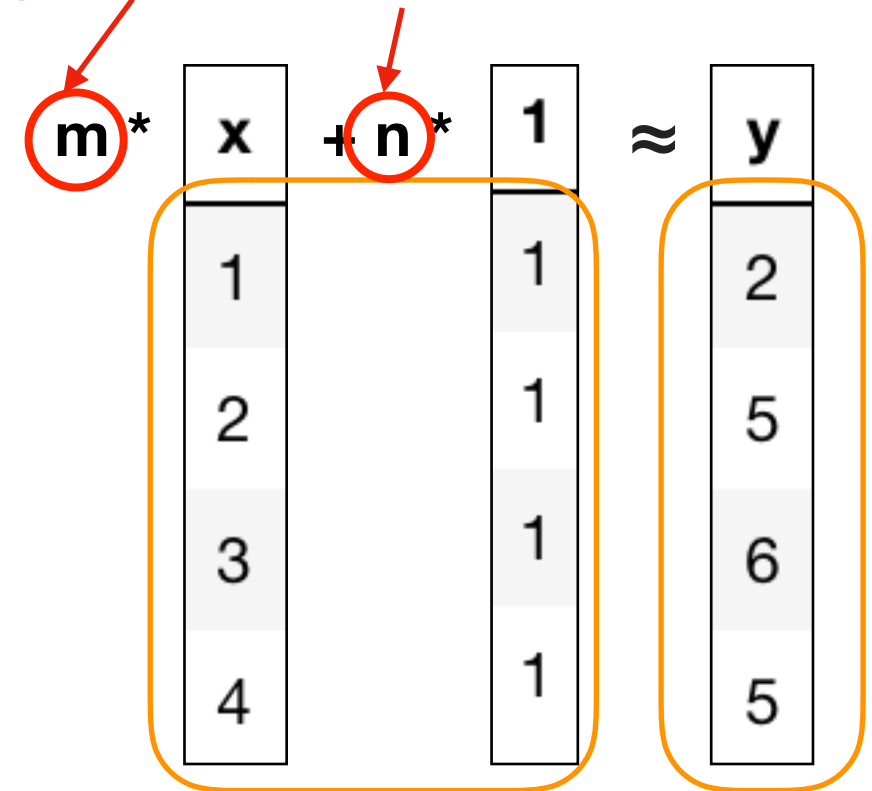
$$m^*x + n^*1 = y$$

Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



...numpy will give us these coefficients



given these inputs, as ndarrays...

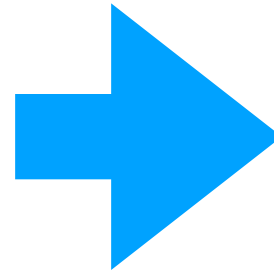
we want a formula like this:

$$m^*x + n^*1 = y$$

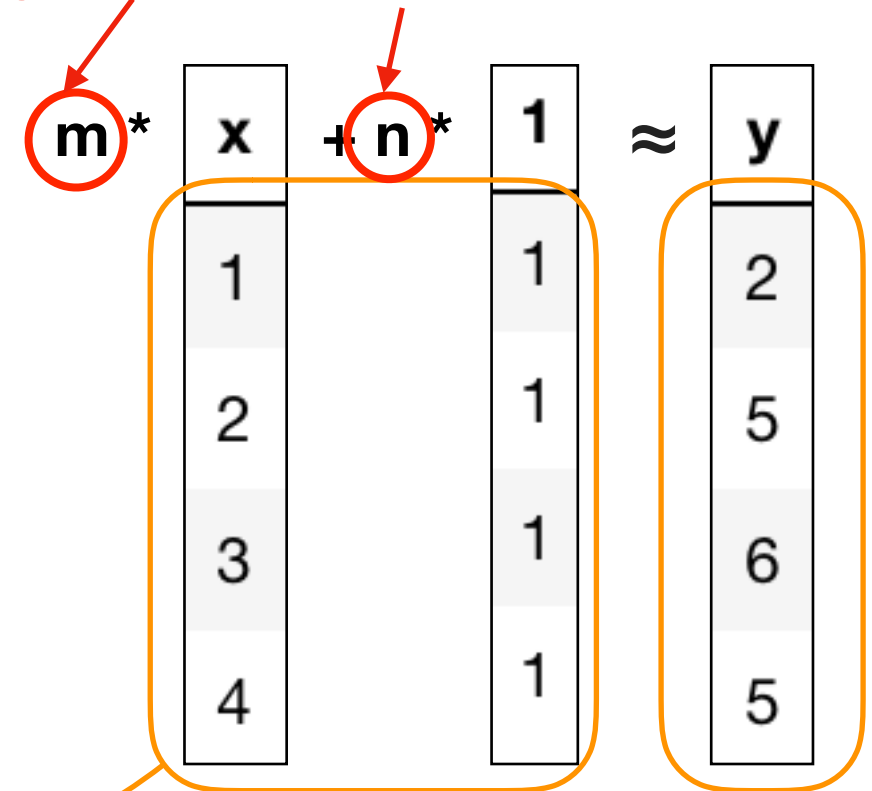
```
np.linalg.lstsq(
```

Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



...numpy will give us these coefficients



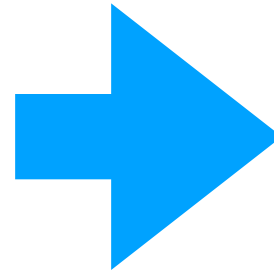
`np.linalg.lstsq(df[["x", "1"]],)`

we want a formula like this:

$$m^*x + n^*1 = y$$

Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



...numpy will give us these coefficients

$$\begin{matrix} m^* & x & + & n^* & 1 & \approx & y \end{matrix}$$

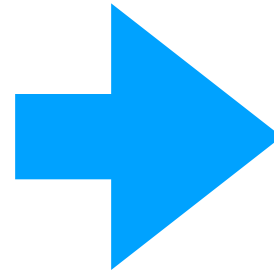
1	2	1	2
2	1	5	
3	1	6	
4	1	5	

The diagram illustrates the linear regression equation $m^*x + n^*1 \approx y$. The coefficients m^* and n^* are circled in red, with arrows pointing to the text "...numpy will give us these coefficients". The data is organized into three vertical columns: the first column contains the values of x (1, 2, 3, 4), the second column contains the values of the constant term 1 (1, 1, 1, 1), and the third column contains the values of y (2, 5, 6, 5). The first two columns are grouped together by an orange bracket, and the third column is grouped by another orange bracket. Arrows from the `df[["x", "1"]]` and `df["y"]` parts of the code below point to the first and third columns respectively.

```
np.linalg.lstsq(df[["x", "1"]], df["y"], )
```

Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



...numpy will give us these coefficients

$$\begin{matrix} m^* & x & + & n^* & 1 & \approx & y \end{matrix}$$

1	2	1	2
2	1	5	
3	1	6	
4	1	5	

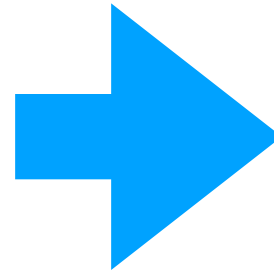
The diagram illustrates the linear regression equation $m^*x + n^*1 \approx y$. The coefficients m^* and n^* are circled in red. The matrix representation of the data is shown as a table with columns x , 1 , and y . The first two columns are grouped by an orange box, and the third column is also grouped by an orange box. Arrows point from the x and 1 columns to the `df[["x", "1"]]` argument in the `np.linalg.lstsq` function call below.

```
np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)
```

rcond is required, but
not important for us

Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



...numpy will give us these coefficients

$$\begin{matrix} m^* & x & + & n^* & 1 & \approx & y \end{matrix}$$

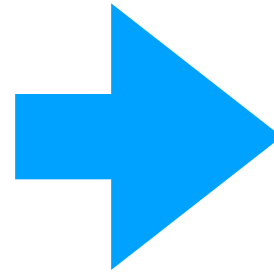
1	2	1	2
2	1	5	
3	1	6	
4	1	5	

The diagram illustrates the linear regression equation $m^*x + n^*1 \approx y$. The coefficients m^* and n^* are circled in red, with arrows pointing to the text "...numpy will give us these coefficients". The matrix representation of the data is shown below the equation, with the columns x , 1 , and y labeled. The values in the matrix are: $x = [1, 2, 3, 4]$, $1 = [1, 1, 1, 1]$, and $y = [2, 5, 6, 5]$. The matrix is enclosed in an orange rounded rectangle.

```
(array([1., 2.]), array([4.]), 2, array([5.77937881, 0.77380911]))  
  
res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)
```

Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



...numpy will give us these coefficients

$$\begin{matrix} m^* & \begin{matrix} x \\ 1 \end{matrix} & + & n^* & \begin{matrix} 1 \end{matrix} & \approx & \begin{matrix} y \end{matrix} \end{matrix}$$

1	2	1	1	2
2	1	1	5	
3	1	6		
4	1	5		

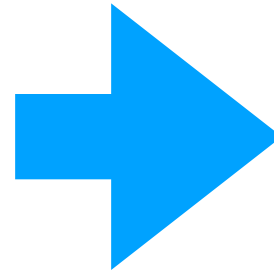
```
(array([1., 2.]), array([4.]), 2, array([5.77937881, 0.77380911]))
```

```
res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)
```

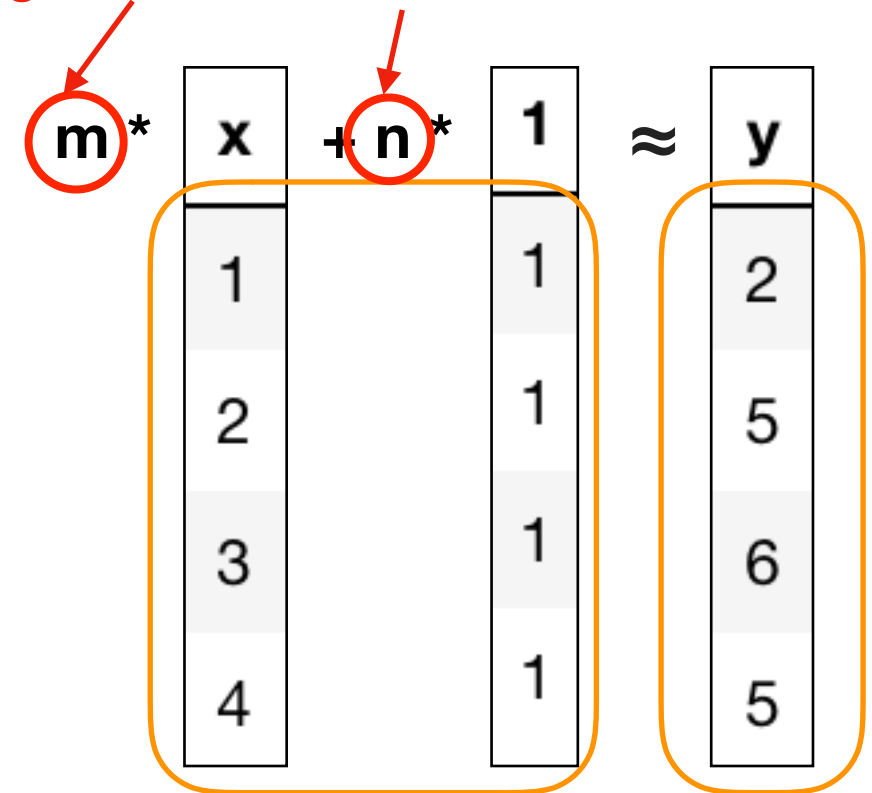
```
# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)
```

Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



...numpy will give us these coefficients



```
(array([1., 2.]), array([4.]), 2, array([5.77937881, 0.77380911]))
```

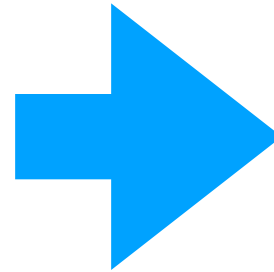
```
res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)
```

```
# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)
```

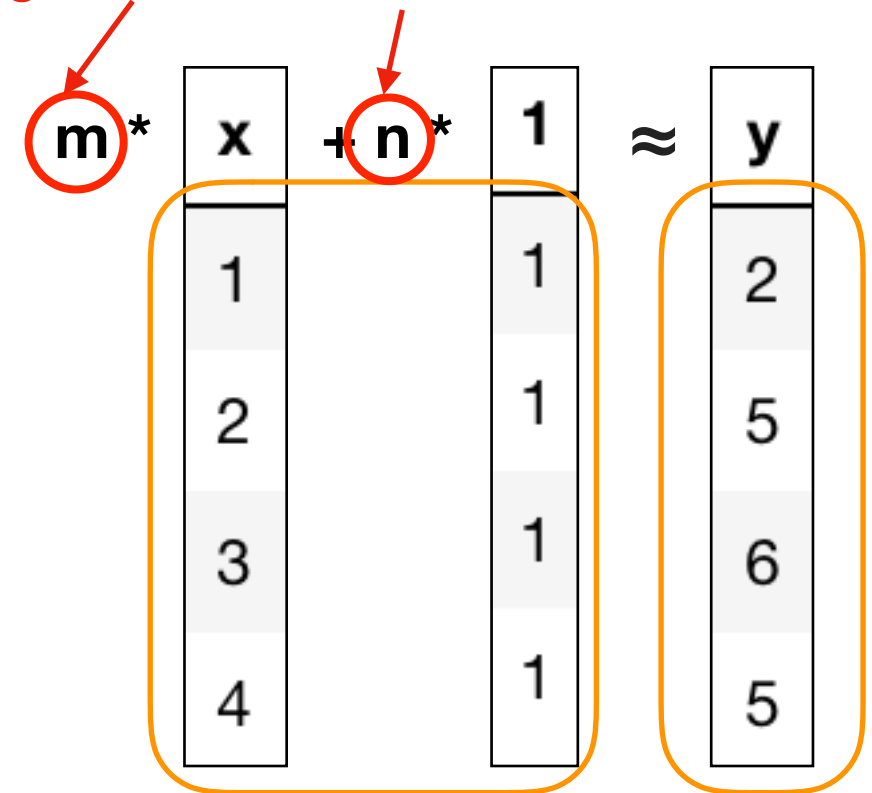
```
coefficients = res[0] # coefficients is (m,n)
```

Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



...numpy will give us these coefficients



```
(array([1., 2.]), array([4.]), 2, array([5.77937881, 0.77380911]))
```

```
res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)
```

```
# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)
```

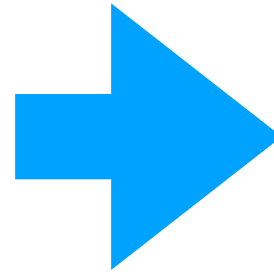
```
coefficients = res[0] # coefficients is (m,n)
```

```
m = coefficients[0]
```

```
n = coefficients[1]
```


Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



...numpy will give us these coefficients

$$\begin{array}{c} \textcircled{m}^* \\ \downarrow \end{array} \begin{array}{|c|} \hline x \\ \hline 1 \\ 2 \\ 3 \\ 4 \\ \hline \end{array} + \begin{array}{c} \textcircled{n}^* \\ \downarrow \end{array} \begin{array}{|c|} \hline 1 \\ \hline 1 \\ 1 \\ 1 \\ \hline \end{array} \approx \begin{array}{|c|} \hline y \\ \hline 2 \\ 5 \\ 6 \\ 5 \\ \hline \end{array}$$

```
(array([1., 2.]), array([4.]), 2, array([5.77937881, 0.77380911]))
```

```
res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)
```

```
# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)
```

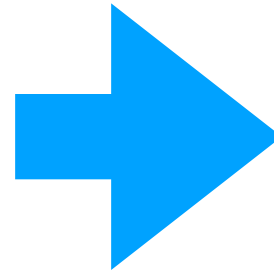
```
coefficients = res[0] # coefficients is (m,n)
```

```
m = coefficients[0] # slope is 1
```

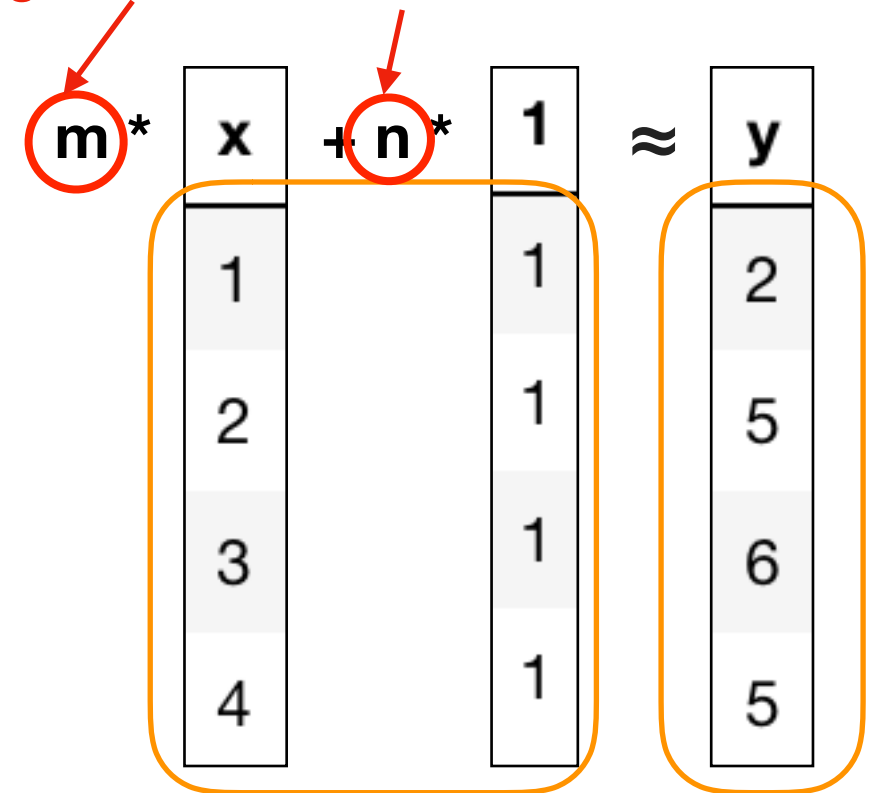
```
n = coefficients[1] # intercept is 2
```

Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



...numpy will give us these coefficients

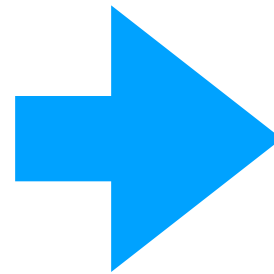


$$y = 1 * x + 2 * 1$$

```
(array([1., 2.]), array([4.]), 2, array([5.77937881, 0.77380911]))  
  
res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)  
  
# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)  
coefficients = res[0] # coefficients is (m,n)  
m = coefficients[0] # slope is 1  
n = coefficients[1] # intercept is 2
```

Example data

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```



...numpy will give us these coefficients

$$\begin{array}{c} \textcircled{m}^* \\ \downarrow \end{array} \begin{array}{|c|} \hline x \\ \hline 1 \\ 2 \\ 3 \\ 4 \\ \hline \end{array} + \begin{array}{c} \textcircled{n}^* \\ \downarrow \end{array} \begin{array}{|c|} \hline 1 \\ \hline 1 \\ 1 \\ 1 \\ \hline \end{array} \approx \begin{array}{|c|} \hline y \\ \hline 2 \\ 5 \\ 6 \\ 5 \\ \hline \end{array}$$

```
(array([1., 2.]), array([4.]), 2, array([5.77937881, 0.77380911]))
```

```
res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)  
  
# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)  
coefficients = res[0] # coefficients is (m,n)  
m = coefficients[0] # slope is 1  
n = coefficients[1] # intercept is 2
```

```
df = DataFrame({  
    "x": [1,2,3,4],  
    "1": np.ones(4),  
    "y": [2,5,6,5]  
})
```

```
res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)  
  
# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)  
coefficients = res[0] # coefficients is (m,n)  
m = coefficients[0] # slope is 1  
n = coefficients[1] # intercept is 2
```

```

df = DataFrame({
    "x": [1,2,3,4],
    "1": np.ones(4),
    "y": [2,5,6,5]
})

res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)

# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)
coefficients = res[0] # coefficients is (m,n)
m = coefficients[0] # slope is 1
n = coefficients[1] # intercept is 2

```

df:

	x	1	y
0	1	1.0	2
1	2	1.0	5
2	3	1.0	6
3	4	1.0	5

```

df = DataFrame({
    "x": [1,2,3,4],
    "1": np.ones(4),
    "y": [2,5,6,5]
})

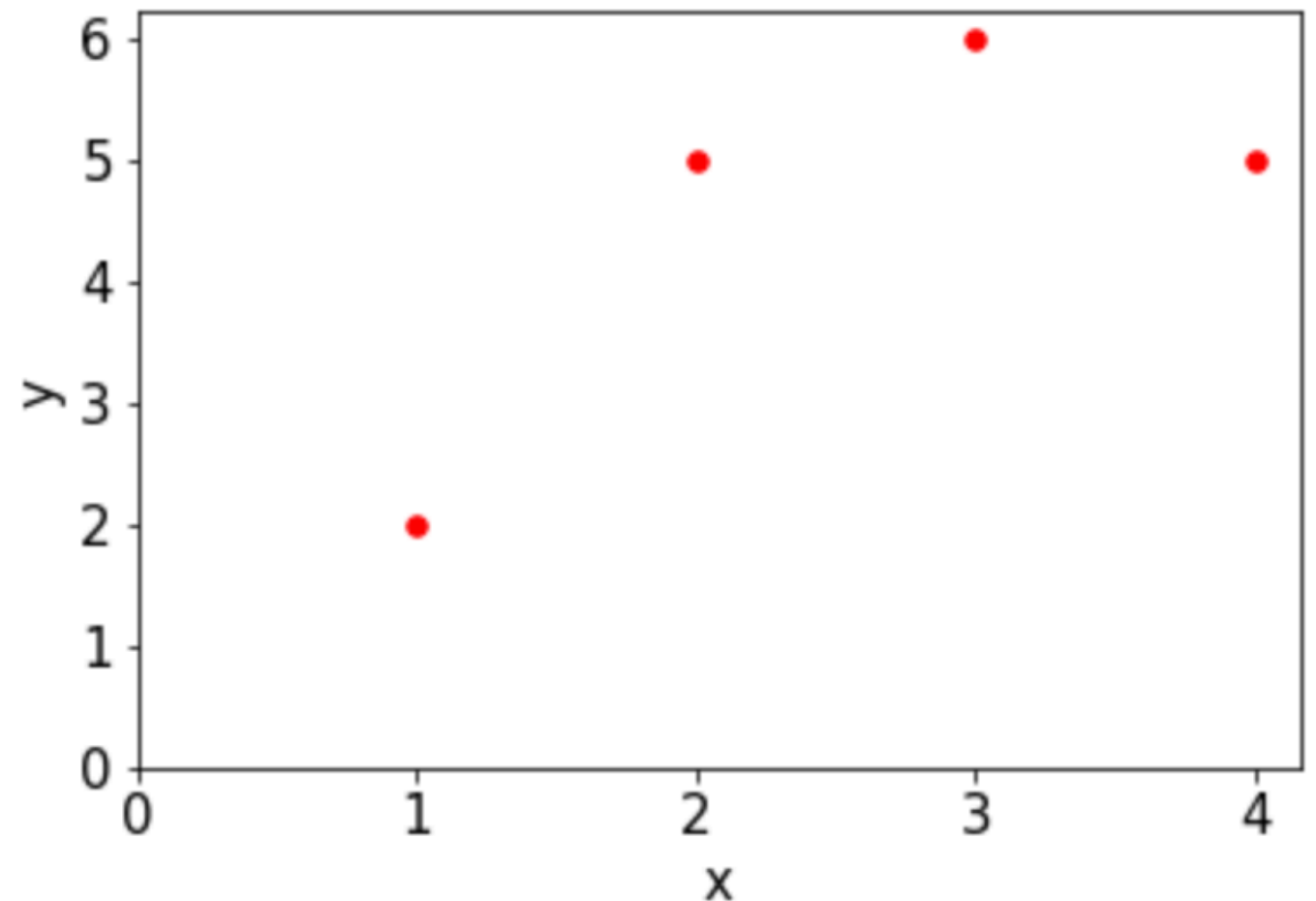
res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)

# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)
coefficients = res[0] # coefficients is (m,n)
m = coefficients[0] # slope is 1
n = coefficients[1] # intercept is 2
ax = df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)

```

df:

	x	1	y
0	1	1.0	2
1	2	1.0	5
2	3	1.0	6
3	4	1.0	5



```

df = DataFrame({
    "x": [1,2,3,4],
    "1": np.ones(4),
    "y": [2,5,6,5]
})

res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)

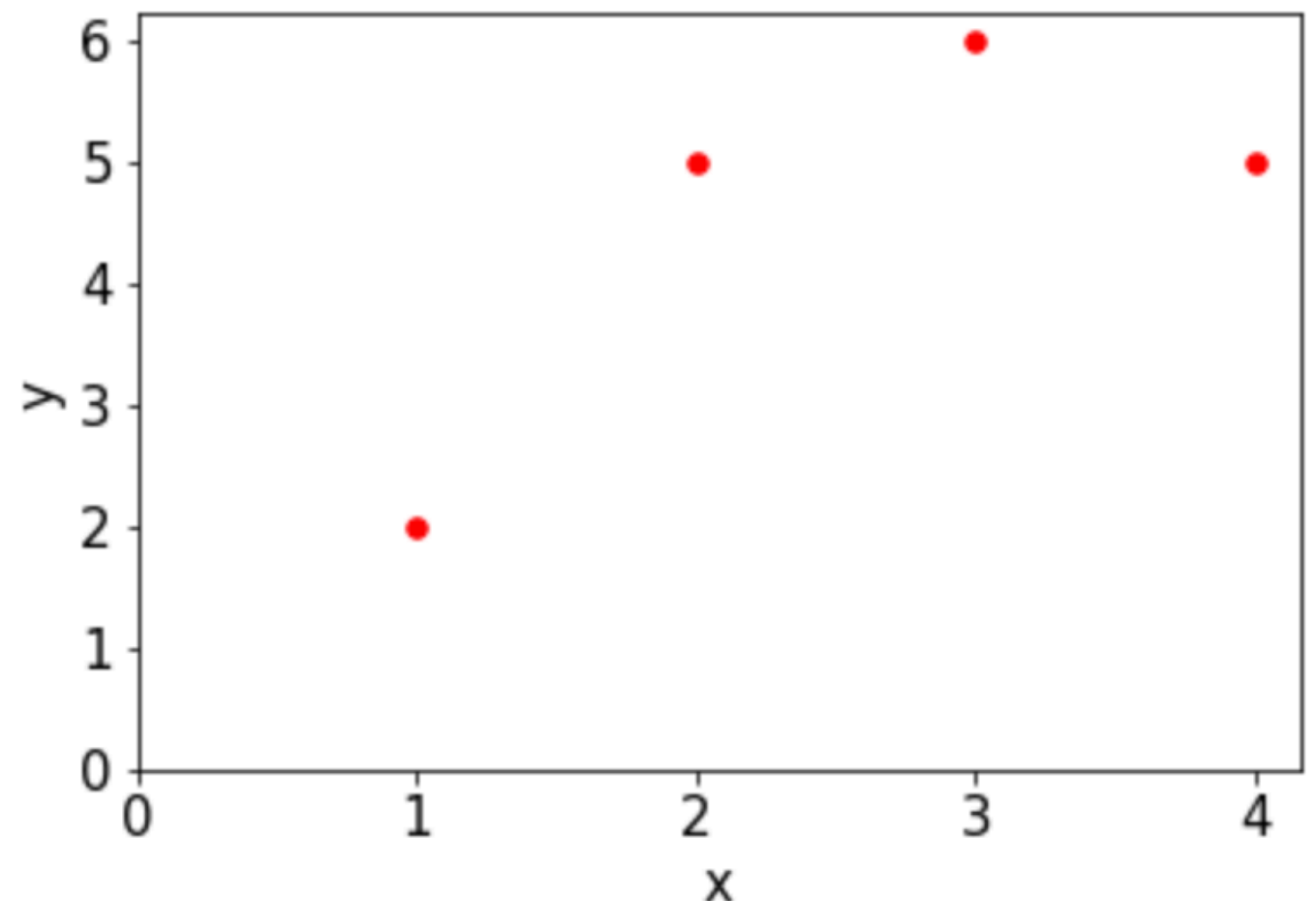
# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)
coefficients = res[0] # coefficients is (m,n)
m = coefficients[0] # slope is 1
n = coefficients[1] # intercept is 2
ax = df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)

df["fit"] = df["x"] * m + n

```

df:

	x	1	y	fit
0	1	1.0	2	3.0
1	2	1.0	5	4.0
2	3	1.0	6	5.0
3	4	1.0	5	6.0



```

df = DataFrame({
    "x": [1,2,3,4],
    "1": np.ones(4),
    "y": [2,5,6,5]
})

res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)

# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)
coefficients = res[0] # coefficients is (m,n)
m = coefficients[0] # slope is 1
n = coefficients[1] # intercept is 2
ax = df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)

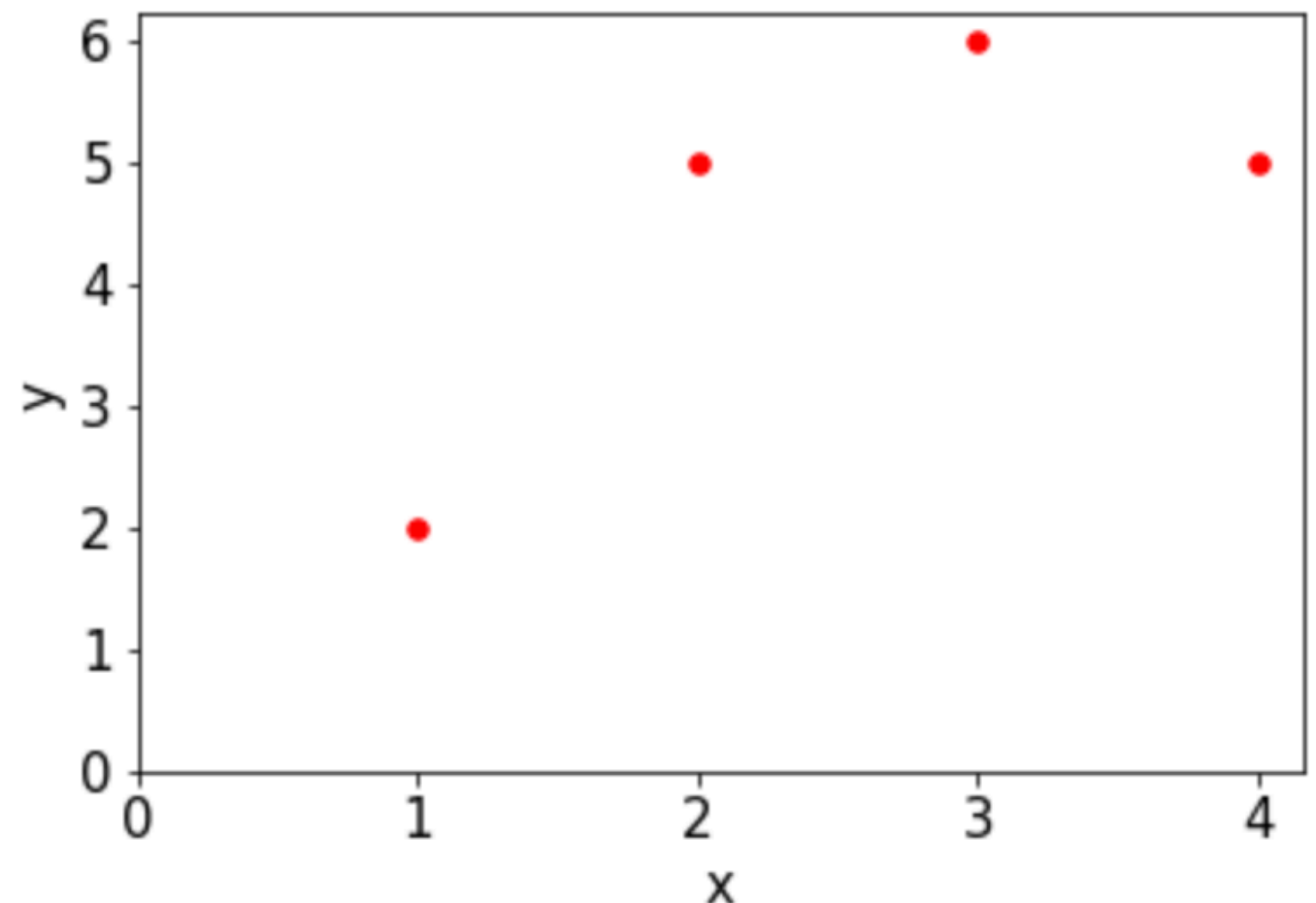
df["fit"] = df["x"] * m + n

```

df:

	x	1	y	fit
0	1	1.0	2	3.0
1	2	1.0	5	4.0
2	3	1.0	6	5.0
3	4	1.0	5	6.0

scatter data
fit line




```

df = DataFrame({
    "x": [1,2,3,4],
    "1": np.ones(4),
    "y": [2,5,6,5]
})

res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)

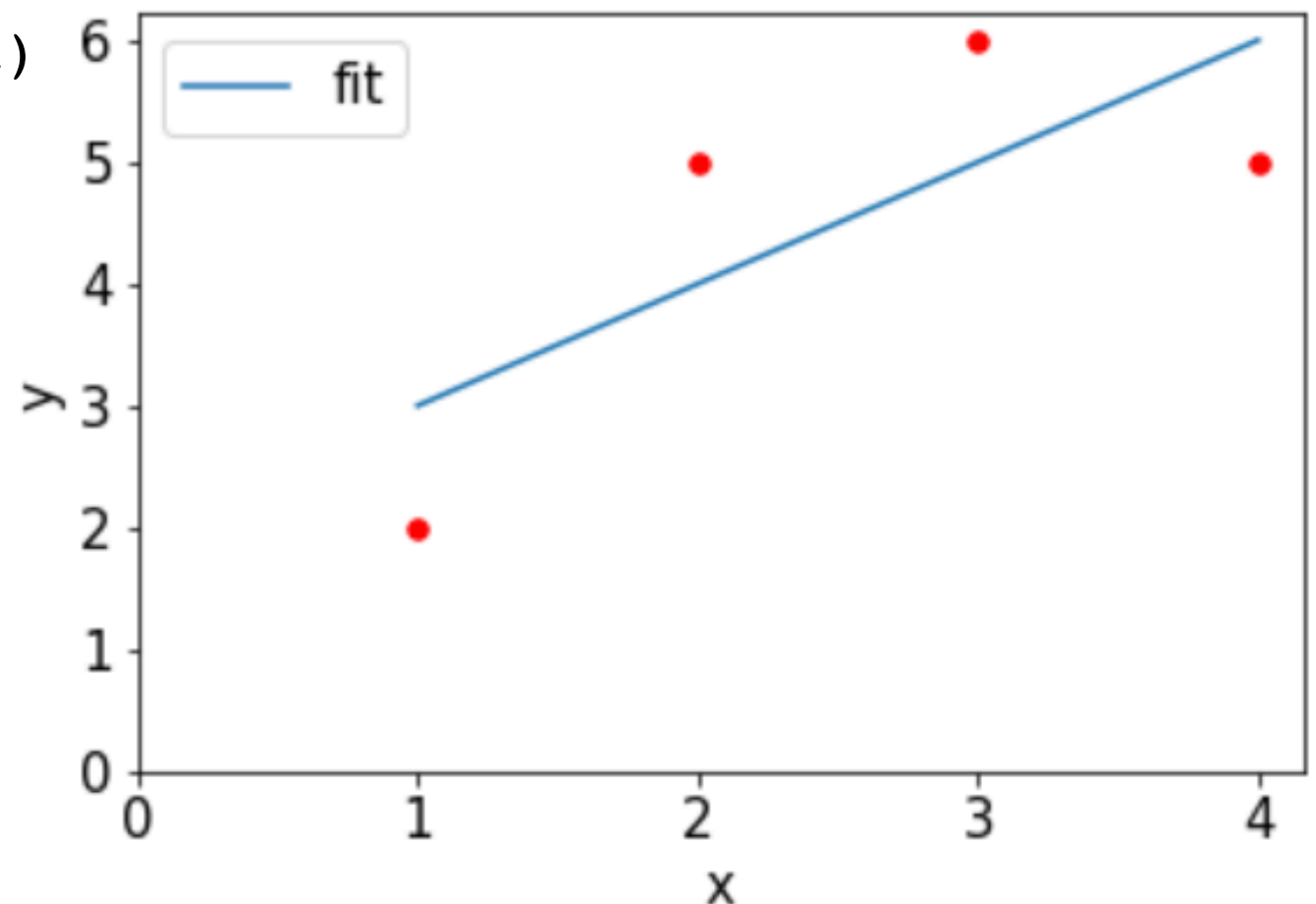
# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)
coefficients = res[0] # coefficients is (m,n)
m = coefficients[0] # slope is 1
n = coefficients[1] # intercept is 2
ax = df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)

df["fit"] = df["x"] * m + n
df.plot.line(x='x', y='fit', ax=ax)

```

df:

	x	1	y	fit
0	1	1.0	2	3.0
1	2	1.0	5	4.0
2	3	1.0	6	5.0
3	4	1.0	5	6.0



```

df = DataFrame({
    "x": [1,2,3,4],
    "y": [2,5,6,5]
})
df["1"] = 1

res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)

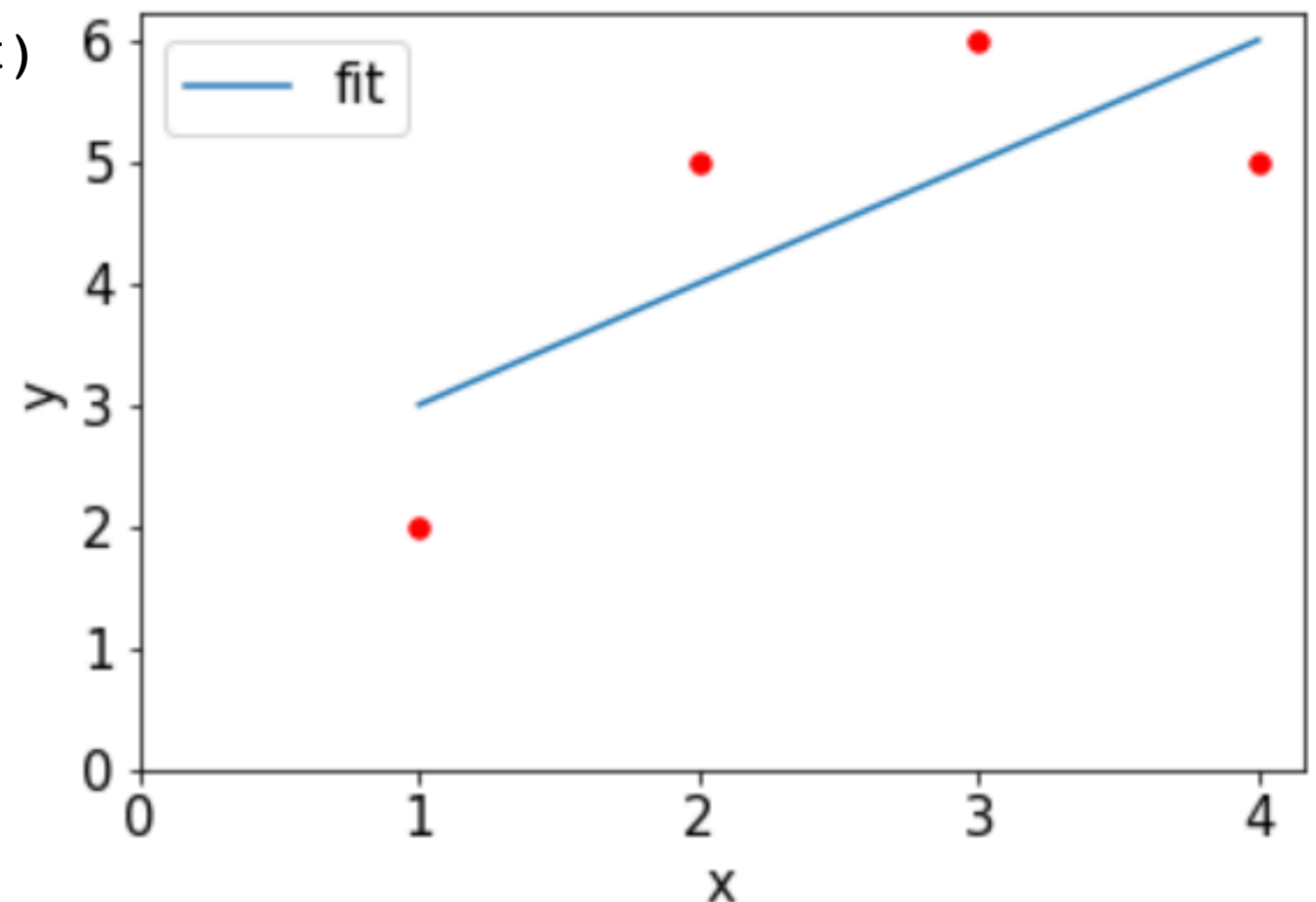
# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)
coefficients = res[0] # coefficients is (m,n)
m = coefficients[0] # slope is 1
n = coefficients[1] # intercept is 2
ax = df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)

df["fit"] = df["x"] * m + n
df.plot.line(x='x', y='fit', ax=ax)

```

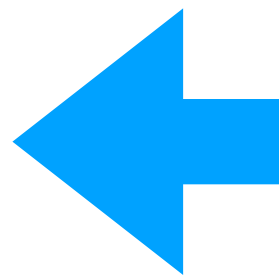
df:

	x	1	y	fit
0	1	1.0	2	3.0
1	2	1.0	5	4.0
2	3	1.0	6	5.0
3	4	1.0	5	6.0



```
df = DataFrame({
    "x": [1,2,3,4],
    "y": [2,5,6,5]
})
```

```
df["1"] = 1
```



this is a complete example. You can copy/paste and just change the input data

```
res = np.linalg.lstsq(df[["x", "1"]], df["y"], rcond=None)
```

```
# res is a tuple: (COEFFICIENTS, VALUE, VALUE, VALUE)
```

```
coefficients = res[0] # coefficients is (m,n)
```

```
m = coefficients[0] # slope is 1
```

```
n = coefficients[1] # intercept is 2
```

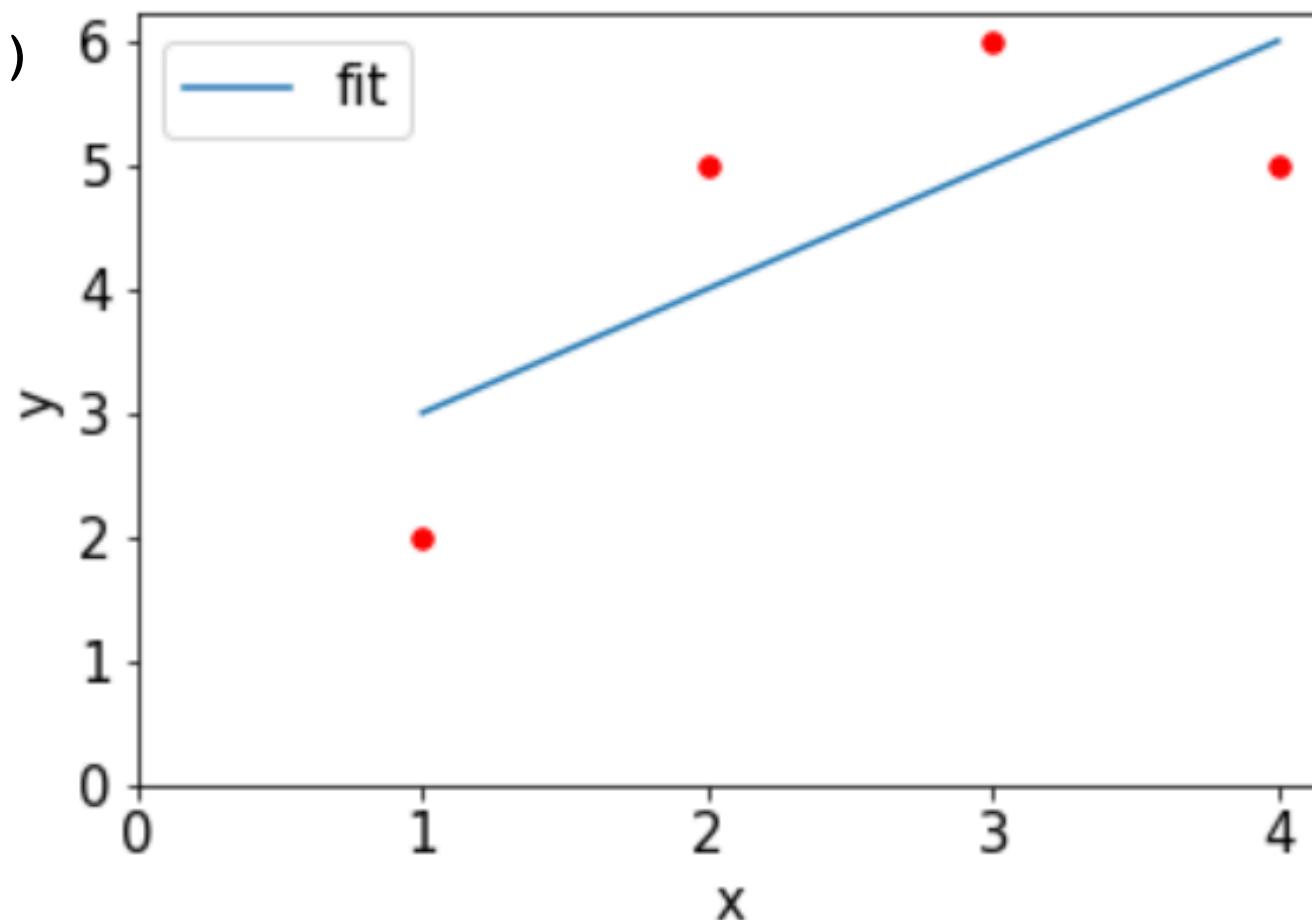
```
ax = df.plot.scatter(x='x', y='y', c='red', s=30, xlim=0, ylim=0)
```

```
df["fit"] = df["x"] * m + n
```

```
df.plot.line(x='x', y='fit', ax=ax)
```

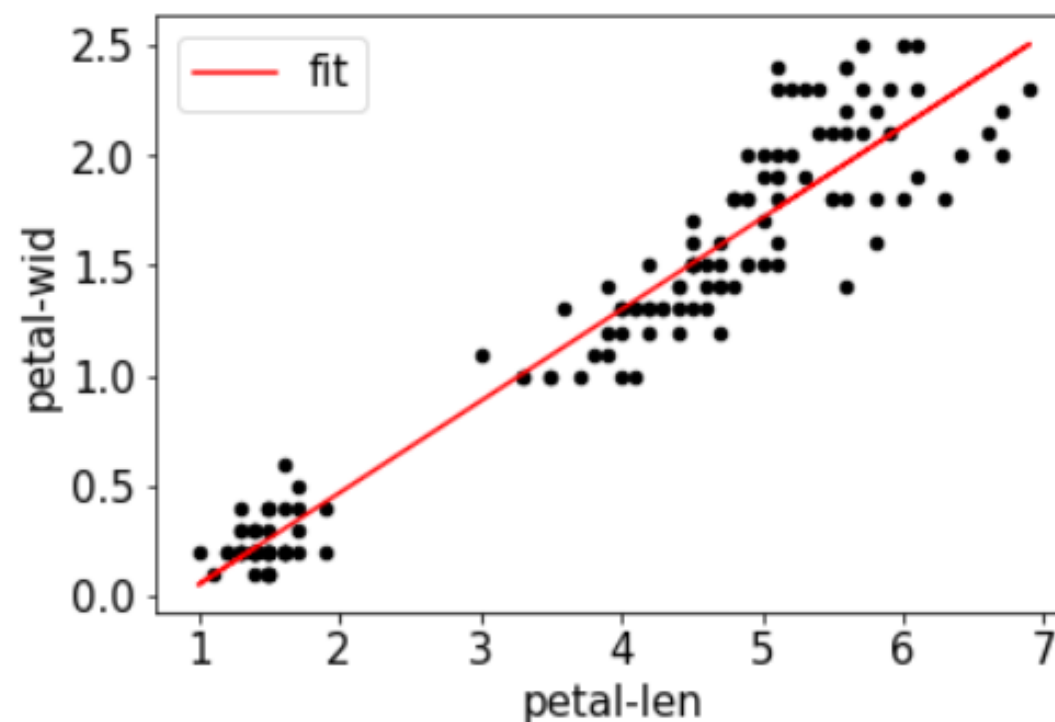
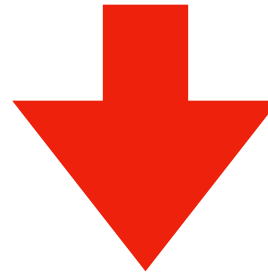
df:

	x	1	y	fit
0	1	1.0	2	3.0
1	2	1.0	5	4.0
2	3	1.0	6	5.0
3	4	1.0	5	6.0



Demo 2: draw real fit line on Iris data

	sepal-len	sepal-wid	petal-len	petal-wid	name
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa



Demo 3: fit line to S&P 500 returns

	year	return	tot	log10(tot)
0	1970	1.0401	1.040100	0.017075
1	1971	1.1431	1.188938	0.075159
2	1972	1.1898	1.414599	0.150633
3	1973	0.8534	1.207219	0.081786
4	1974	0.7353	0.887668	-0.051750
5	1975	1.3720	1.217880	0.085605

