# [301] Regression

Tyler Caraza-Harter

# Learning Objectives Today

History of regression

Drawing a fit line

Finding the slope/intercept w/ least squares method

Numpy introduction

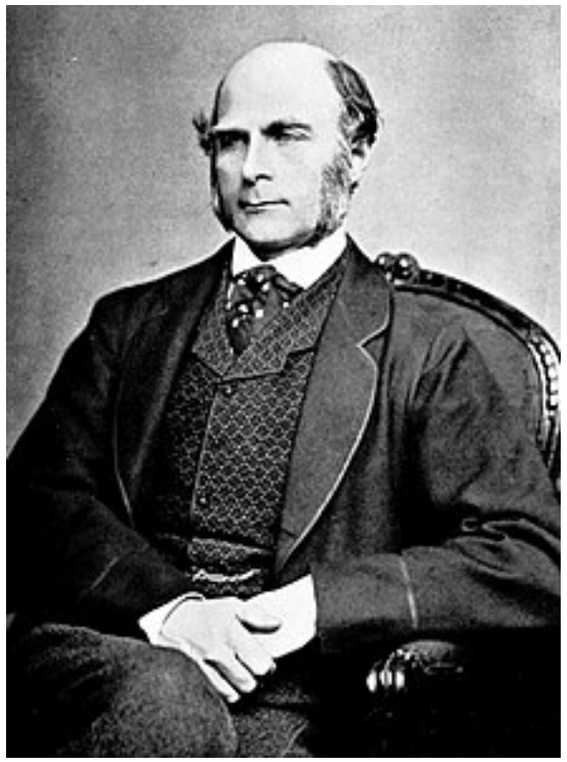
Using numpy.linalg.lstsq

Advanced:

- non-linear relations
- more variables

# History of Regression

**Francis Galton**

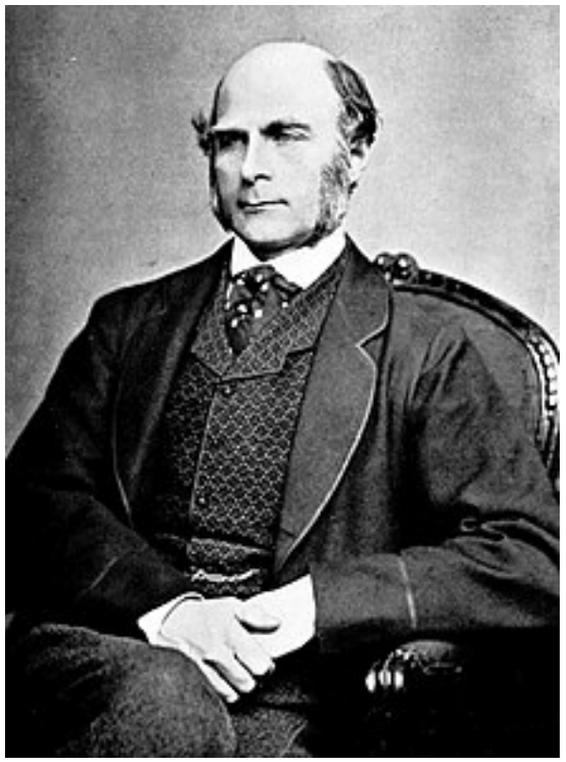


https://en.wikipedia.org/wiki/Francis_Galton

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

# History of Regression

**Francis Galton**

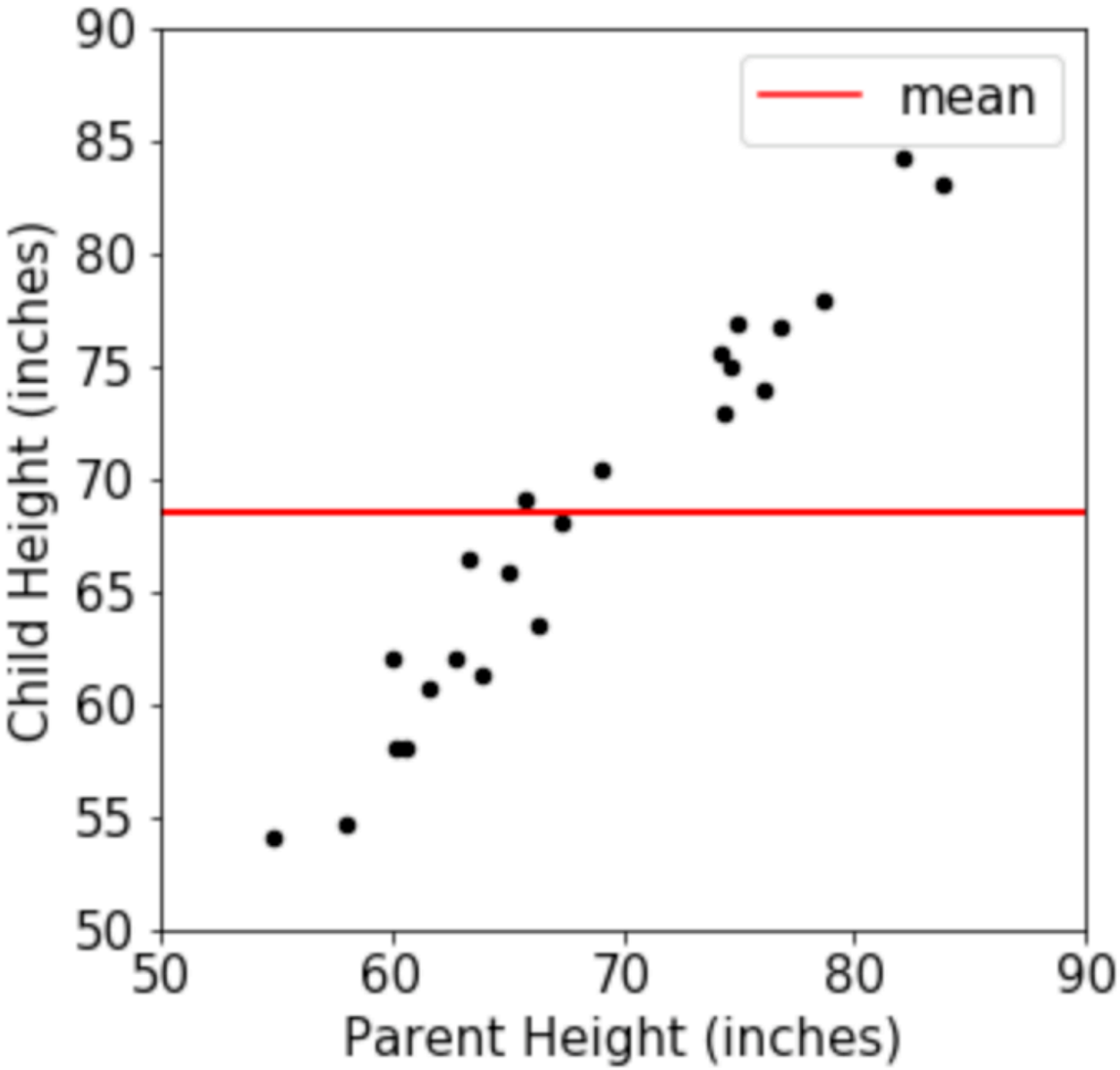


https://en.wikipedia.org/wiki/Francis_Galton

**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

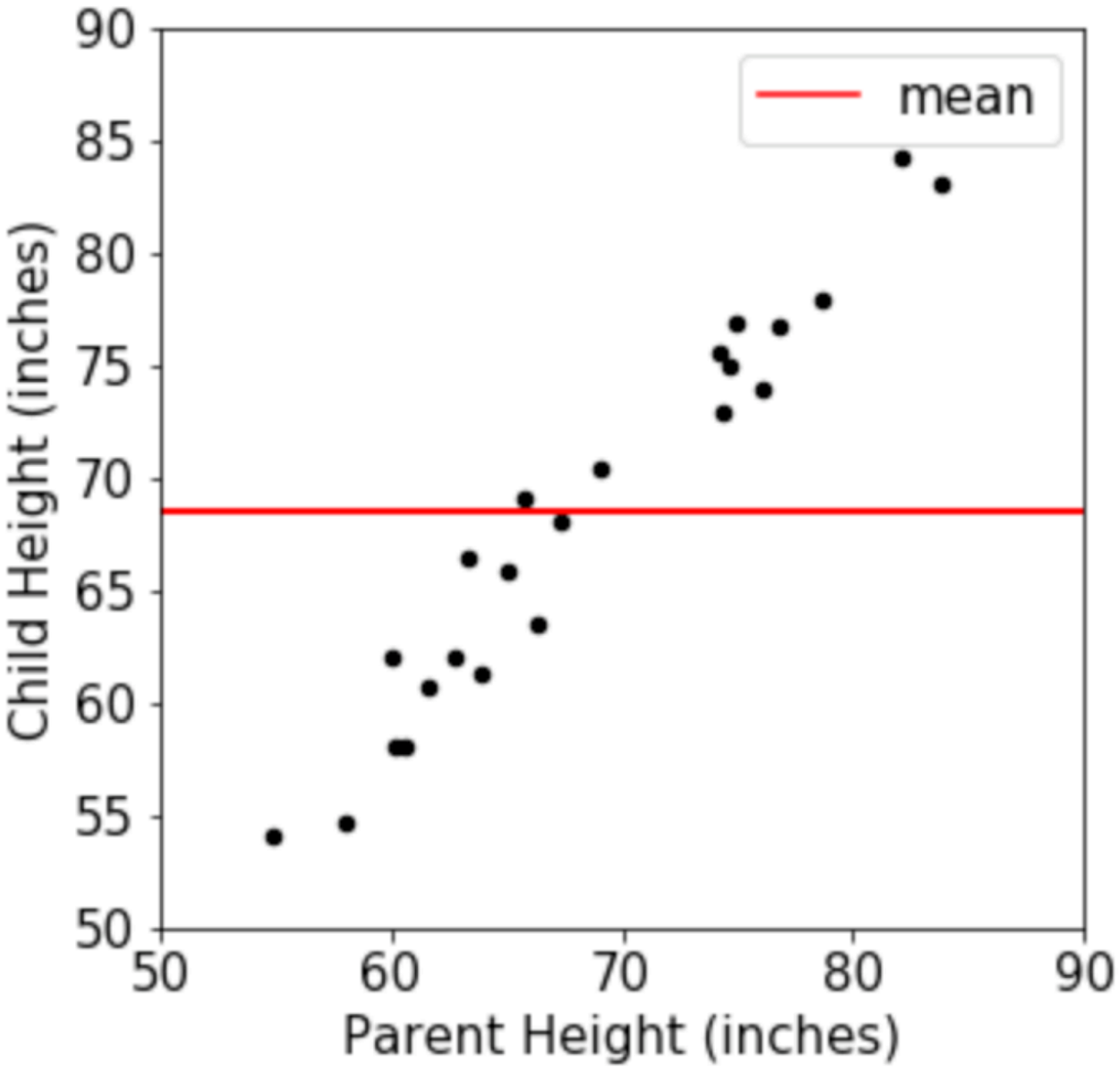



Observation:

- child height equals parent height (plus some noise)

**Note:** all these height plots contain fake data

# Result you might expect




Observation:

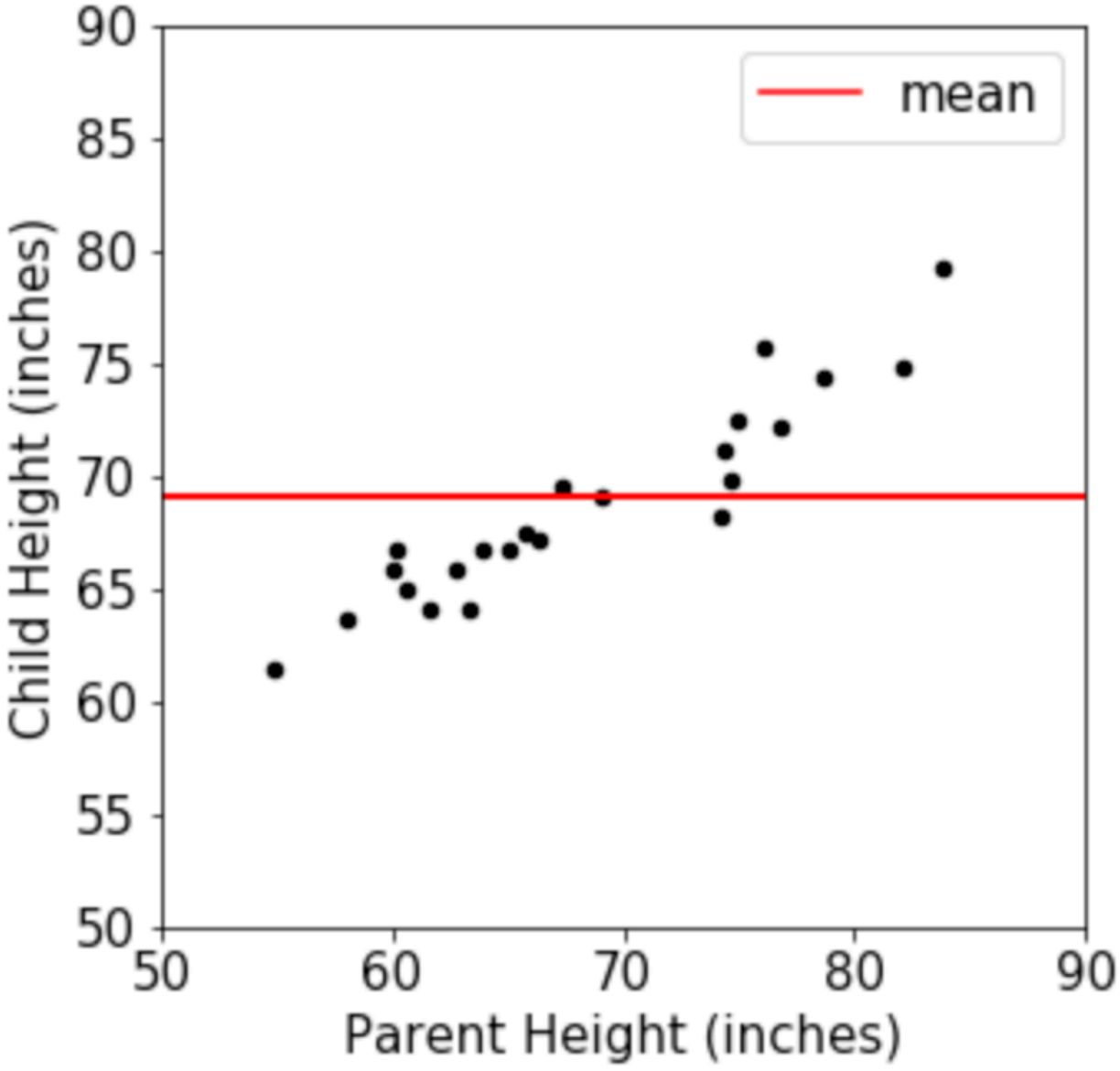
- child height equals parent height (plus some noise)

What about other factors?

- height of other parent
- nutrition
- etc

**Note:** all these height plots contain fake data

# More realistic results

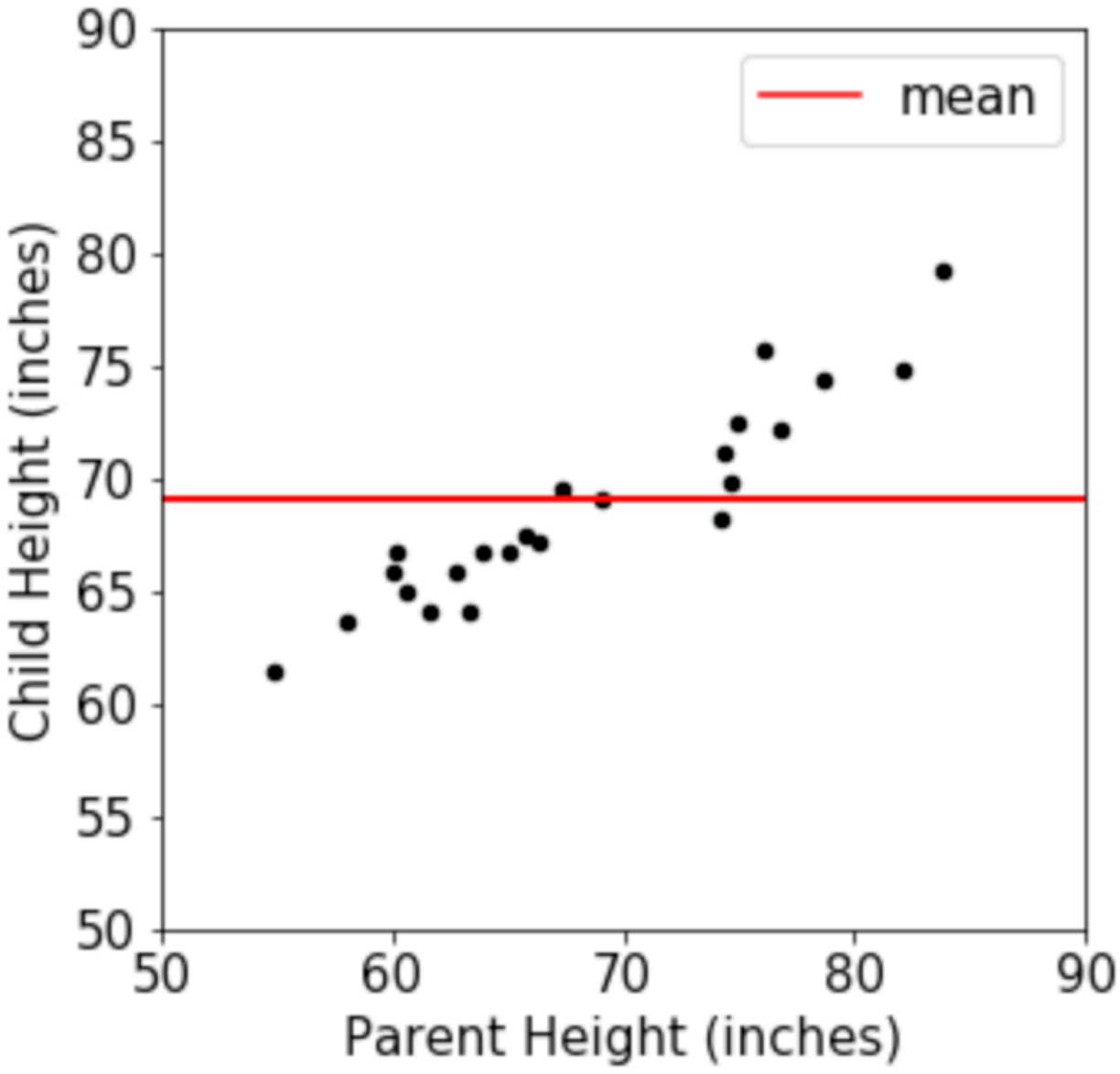



Observation:

- 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



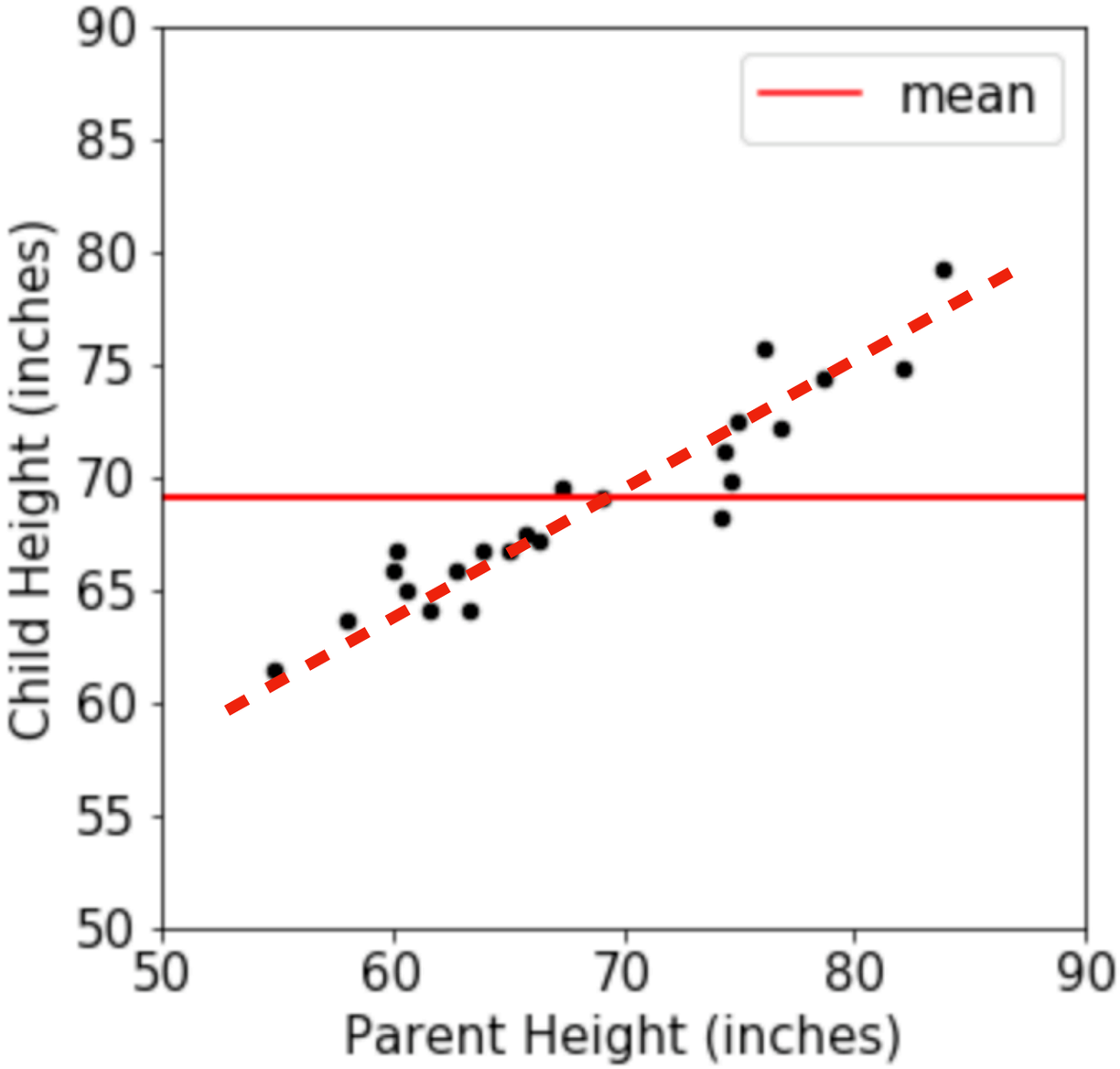

Observation:

- 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

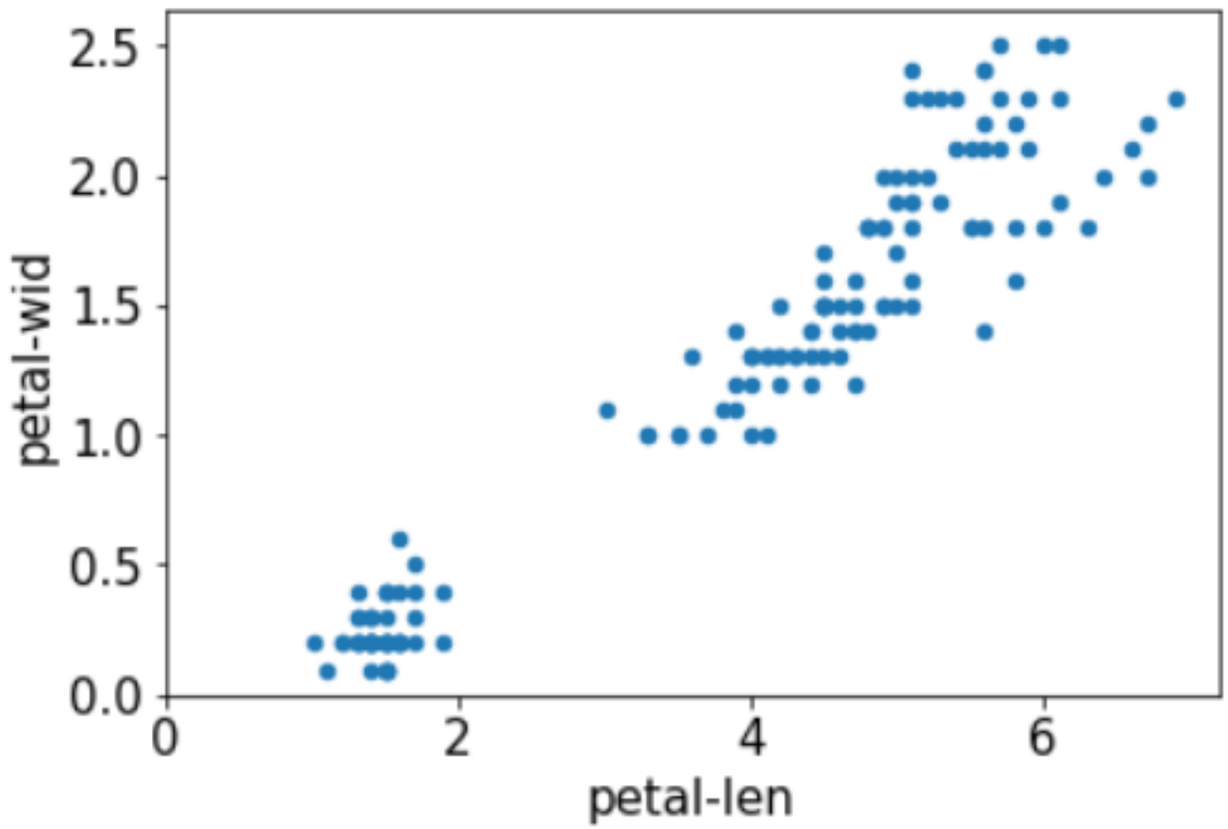
Drawing a fit line

Finding the slope/intercept w/ least squares method

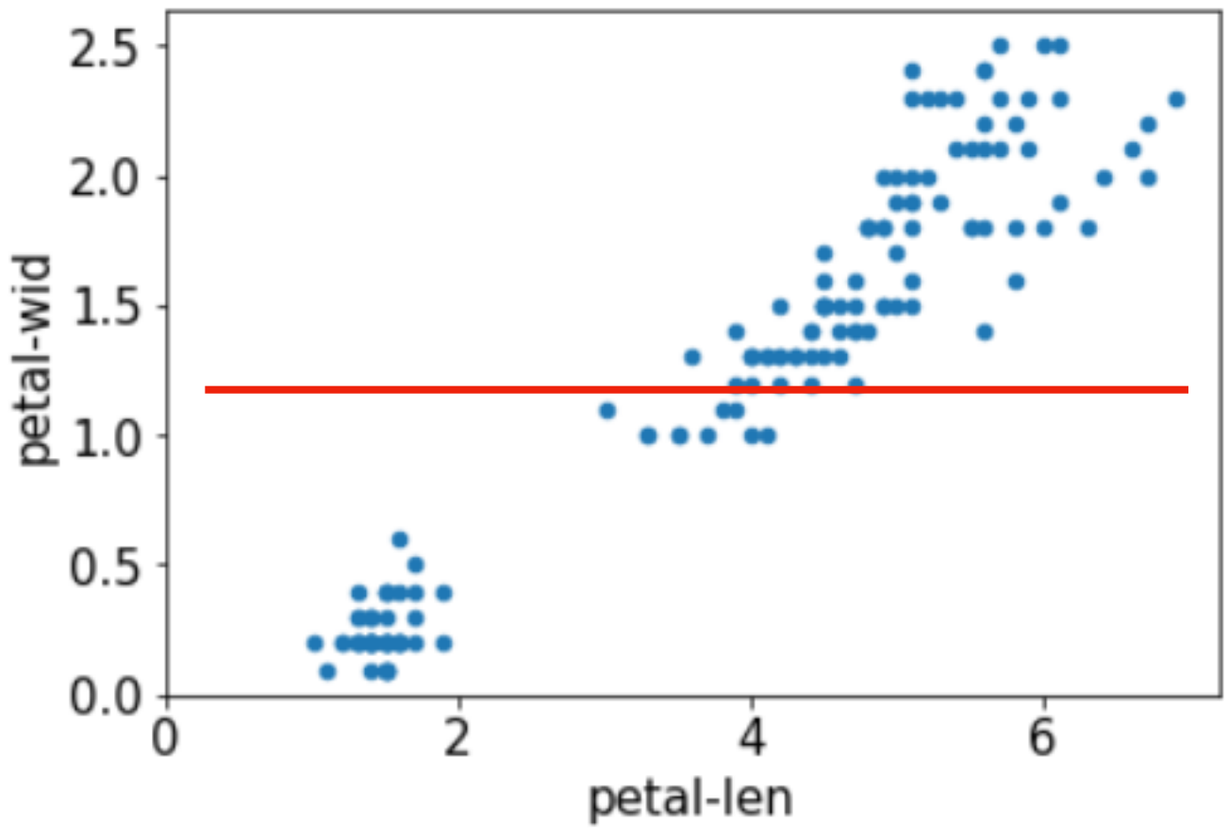
Numpy introduction

Using numpy.linalg.lstsq

# Demo 1: annotate Iris data

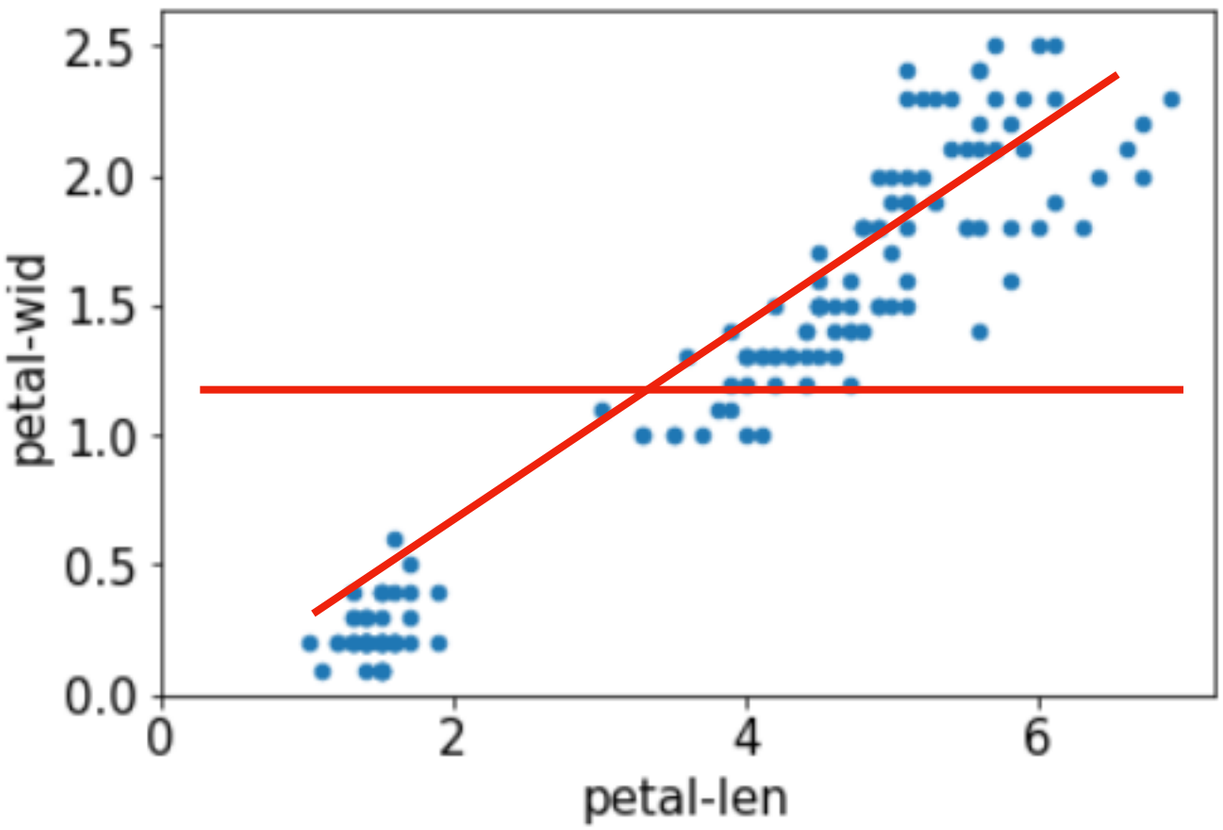


# Demo 1: annotate Iris data




**Annotation 1: mean line**

# Demo 1: annotate Iris data




**Annotation 1: mean line**

**Annotation 2: fit line**

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

# Learning Objectives Today

History of regression

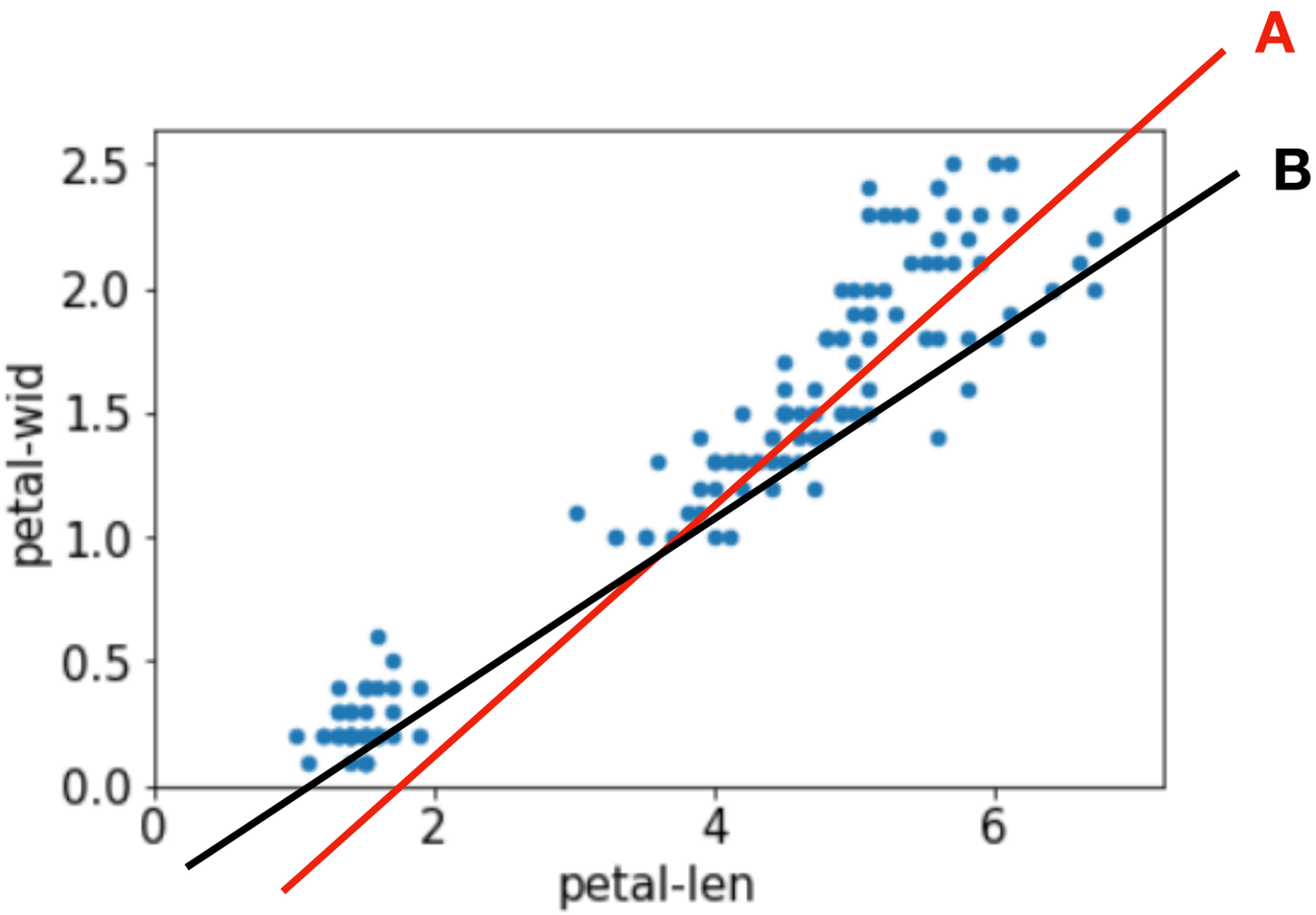
Drawing a fit line

Finding the slope/intercept w/ least-squares method

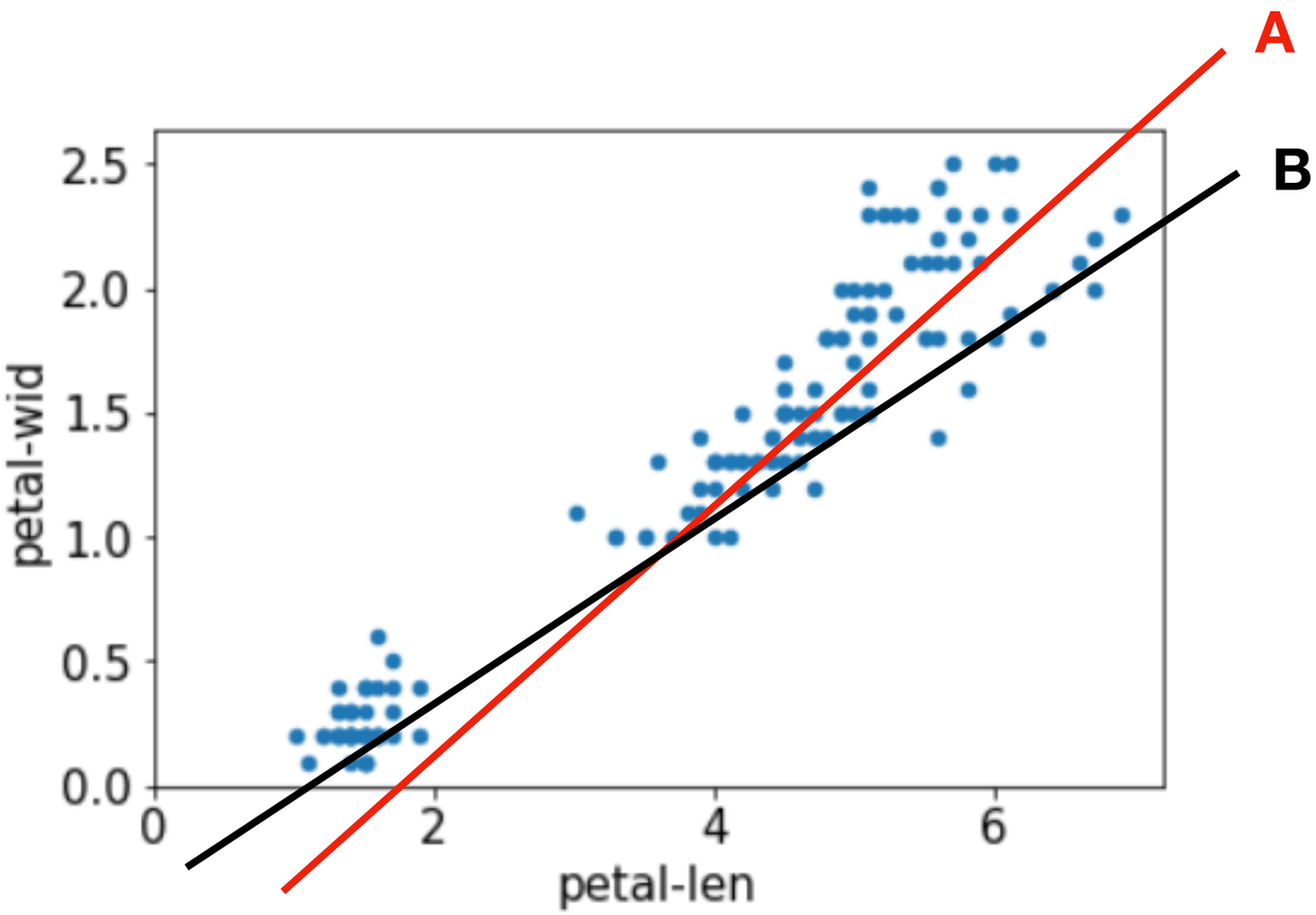
Numpy introduction

Using numpy.linalg.lstsq

# Which fit line is better?

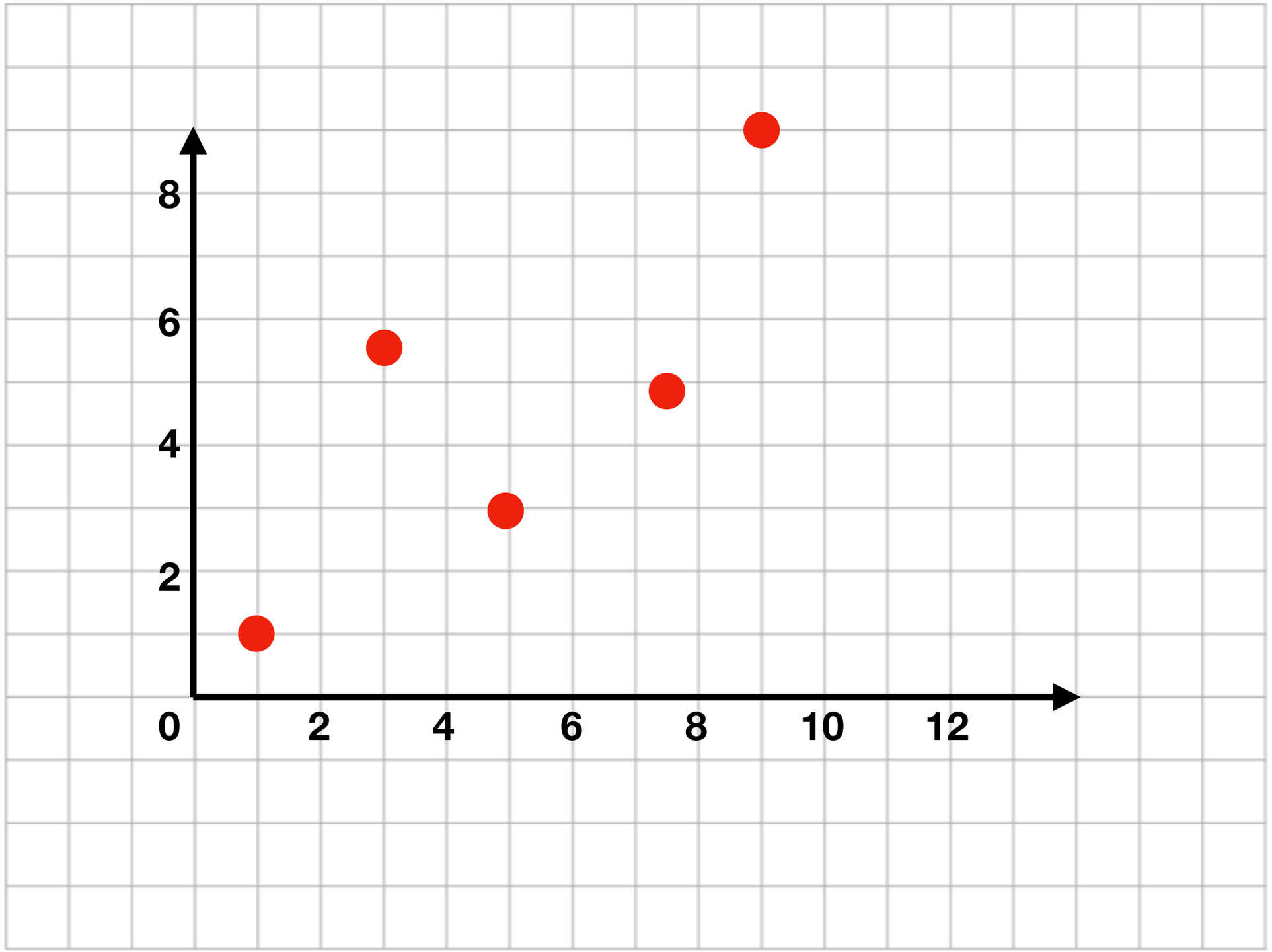


# Which fit line is better?

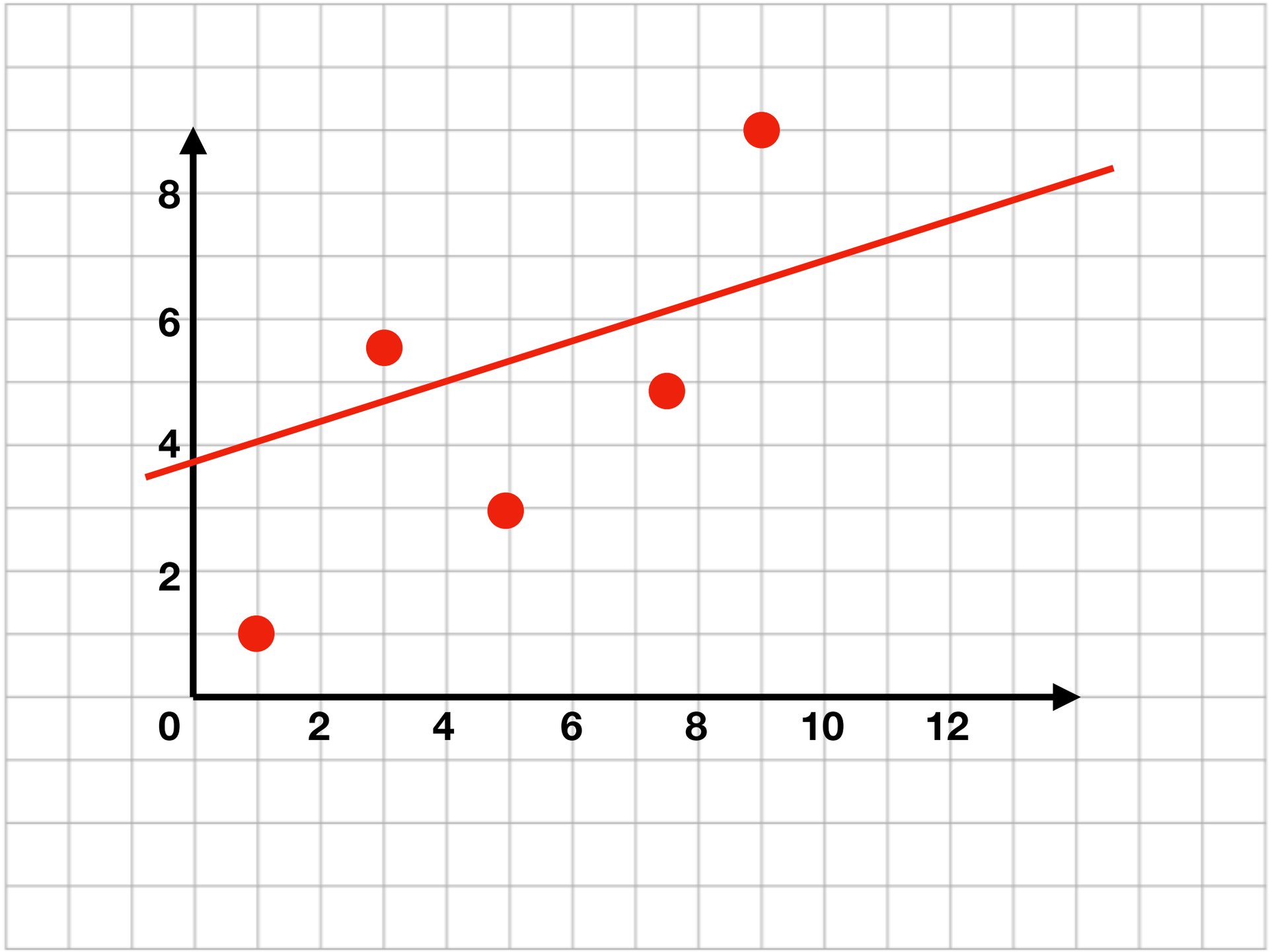



**We need a metric to evaluate how good a fit is**

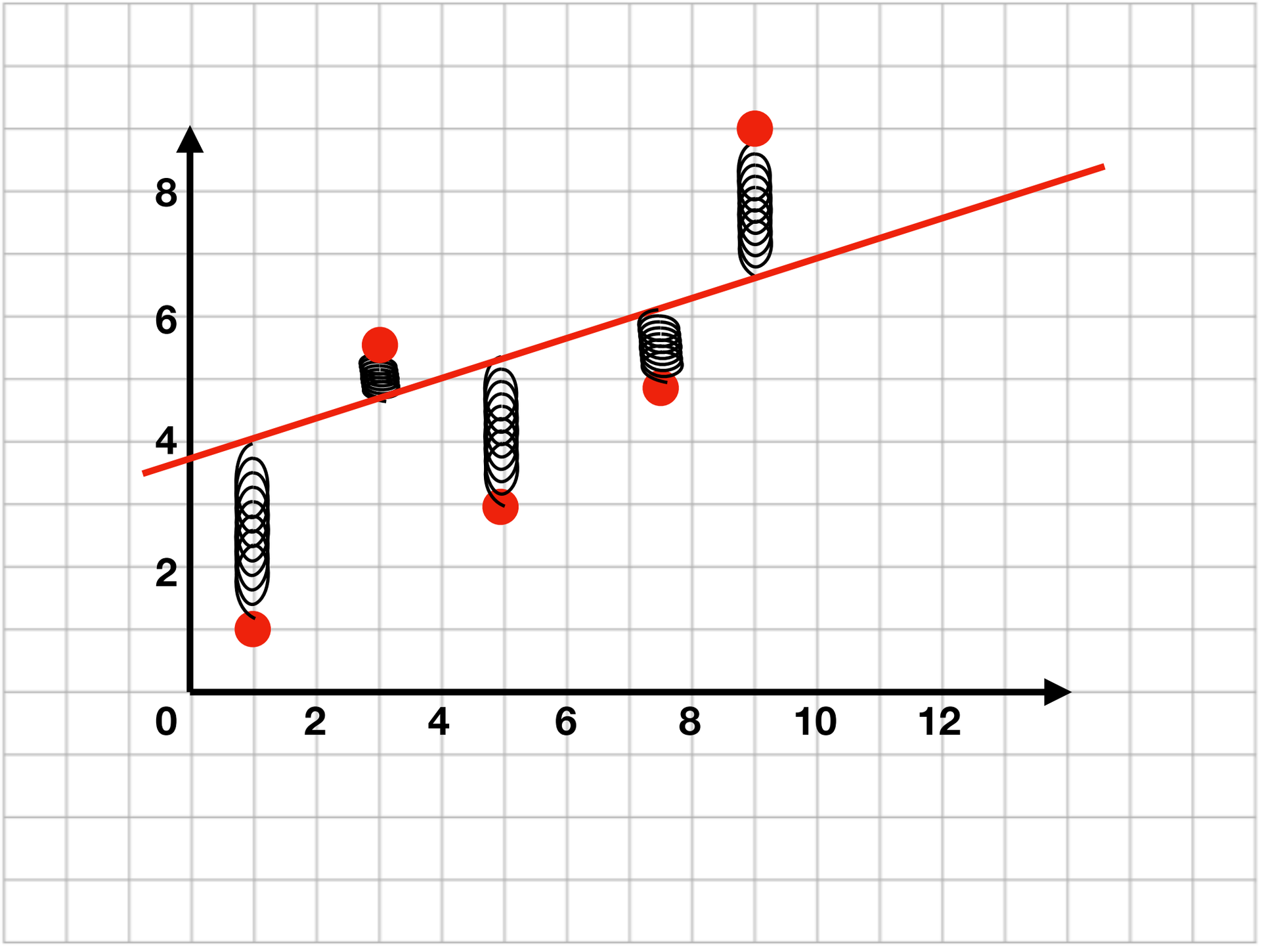
# Intuition: Springs



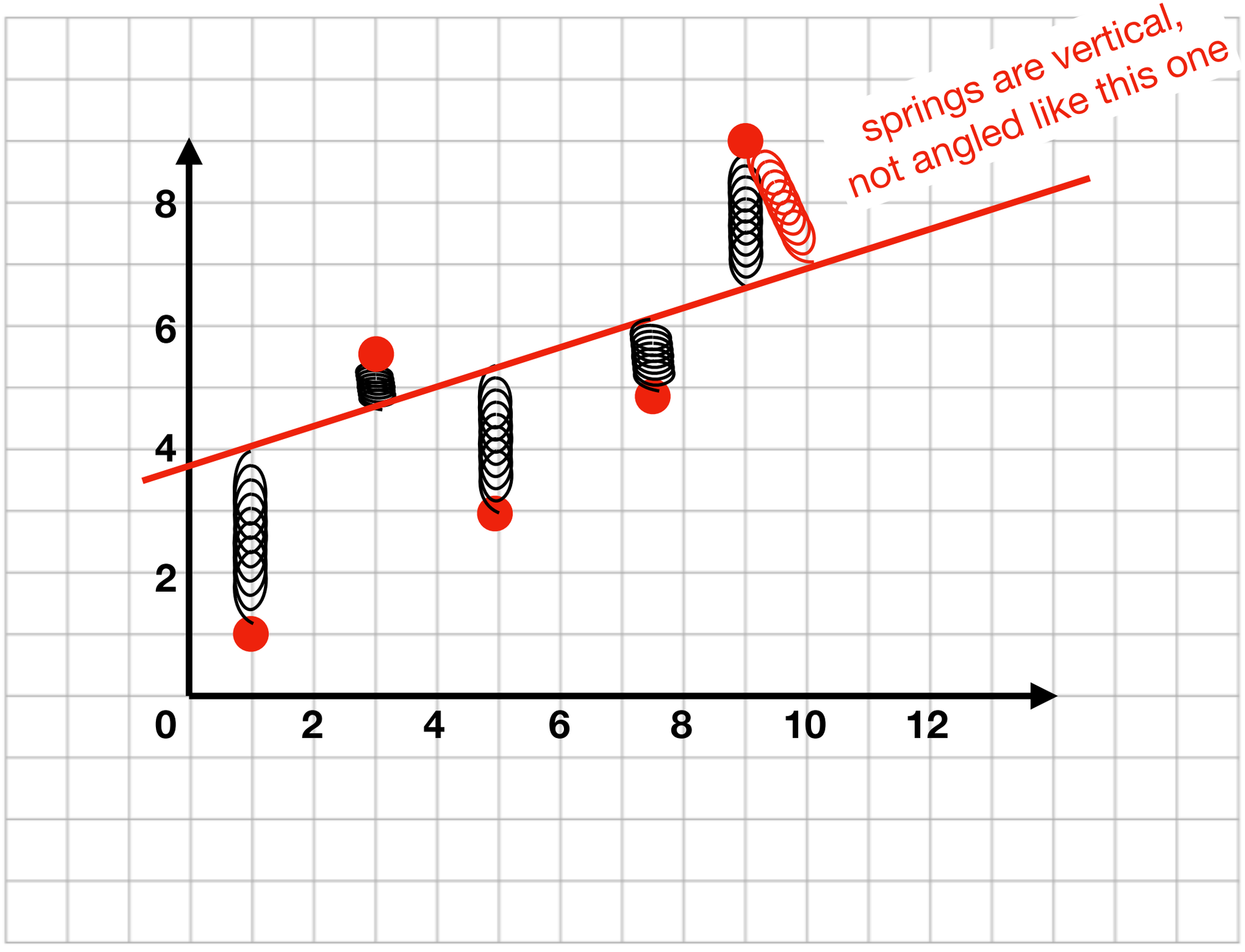
# Intuition: Springs



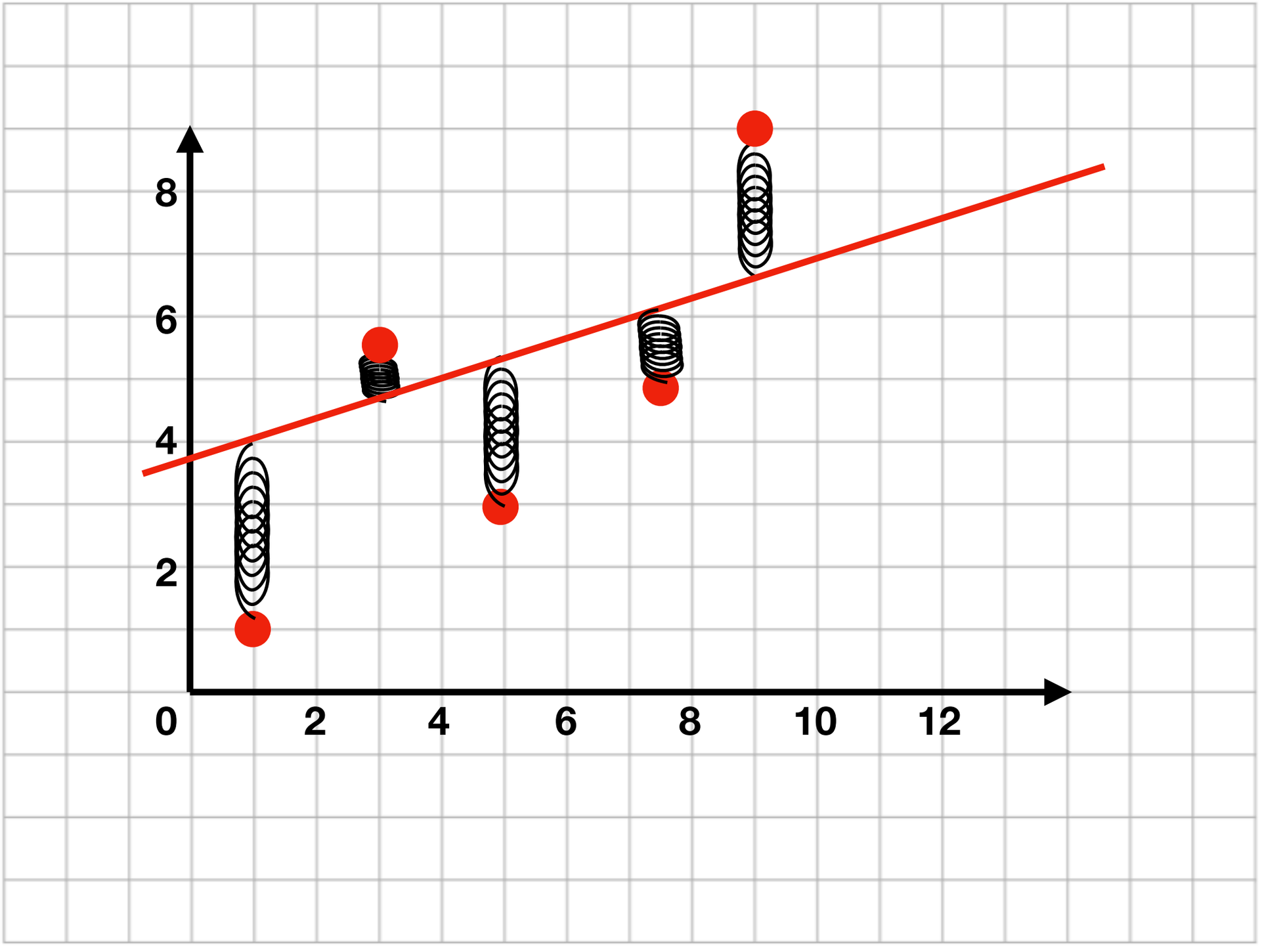
# Intuition: Springs



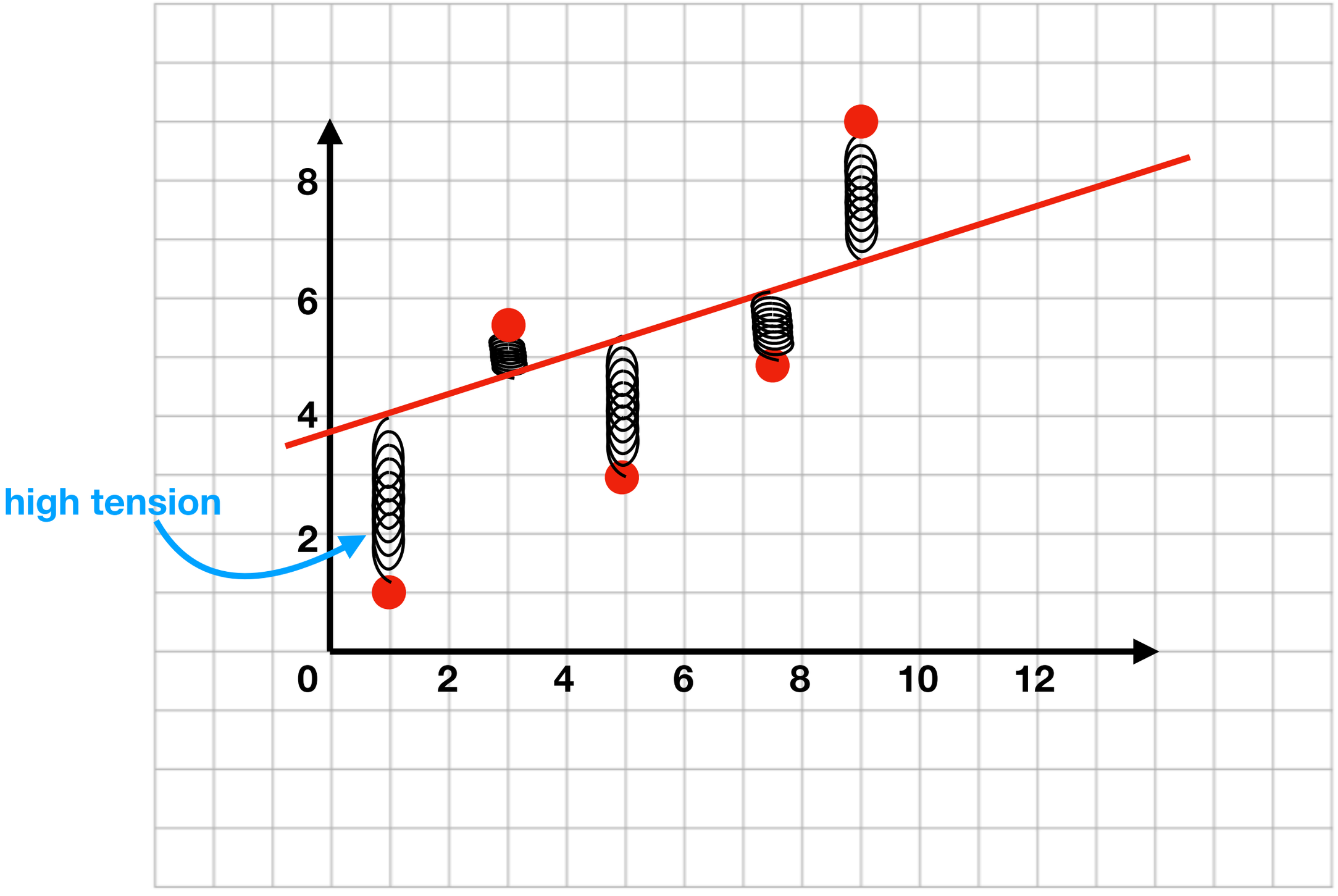
# Intuition: Springs



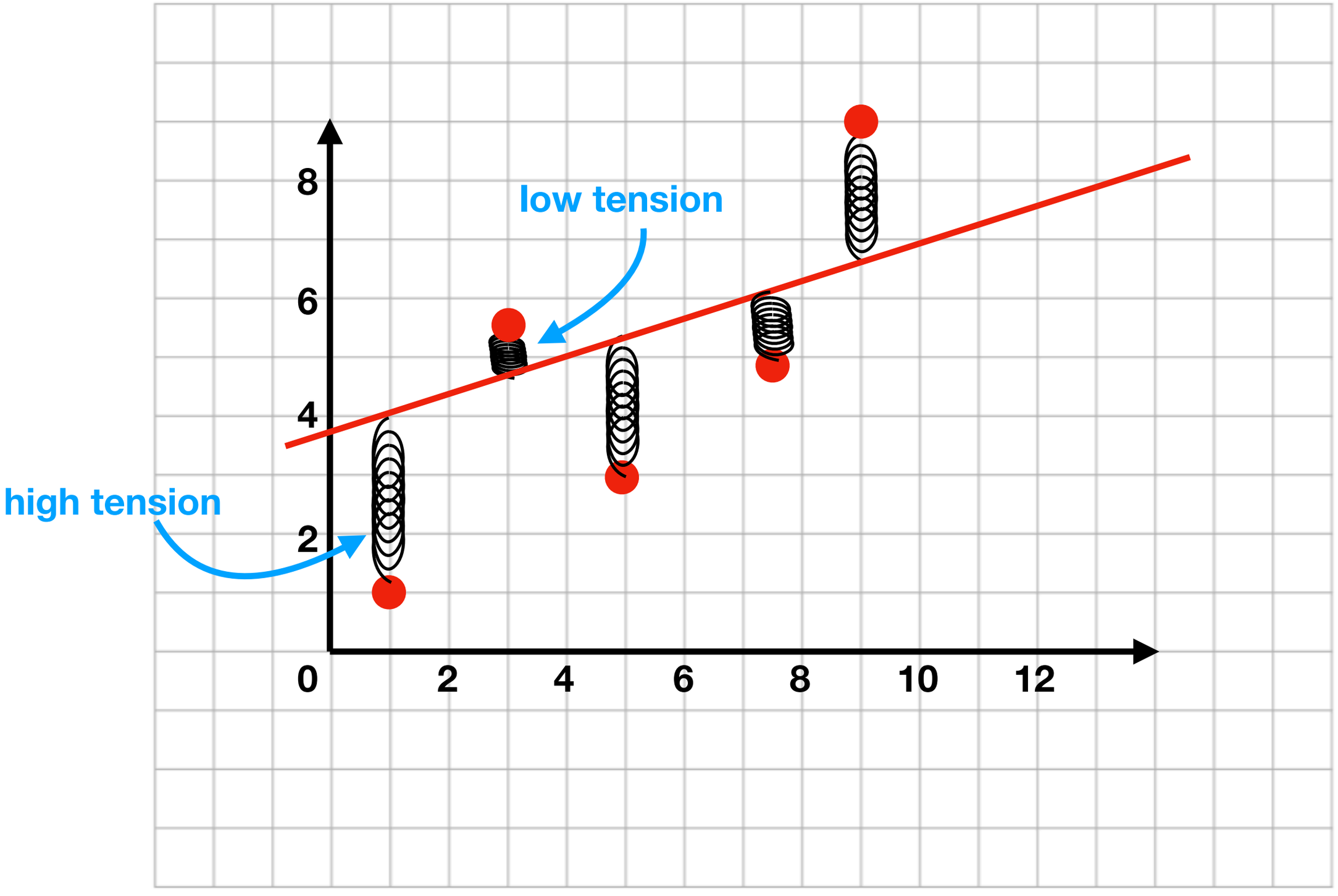
# Intuition: Springs



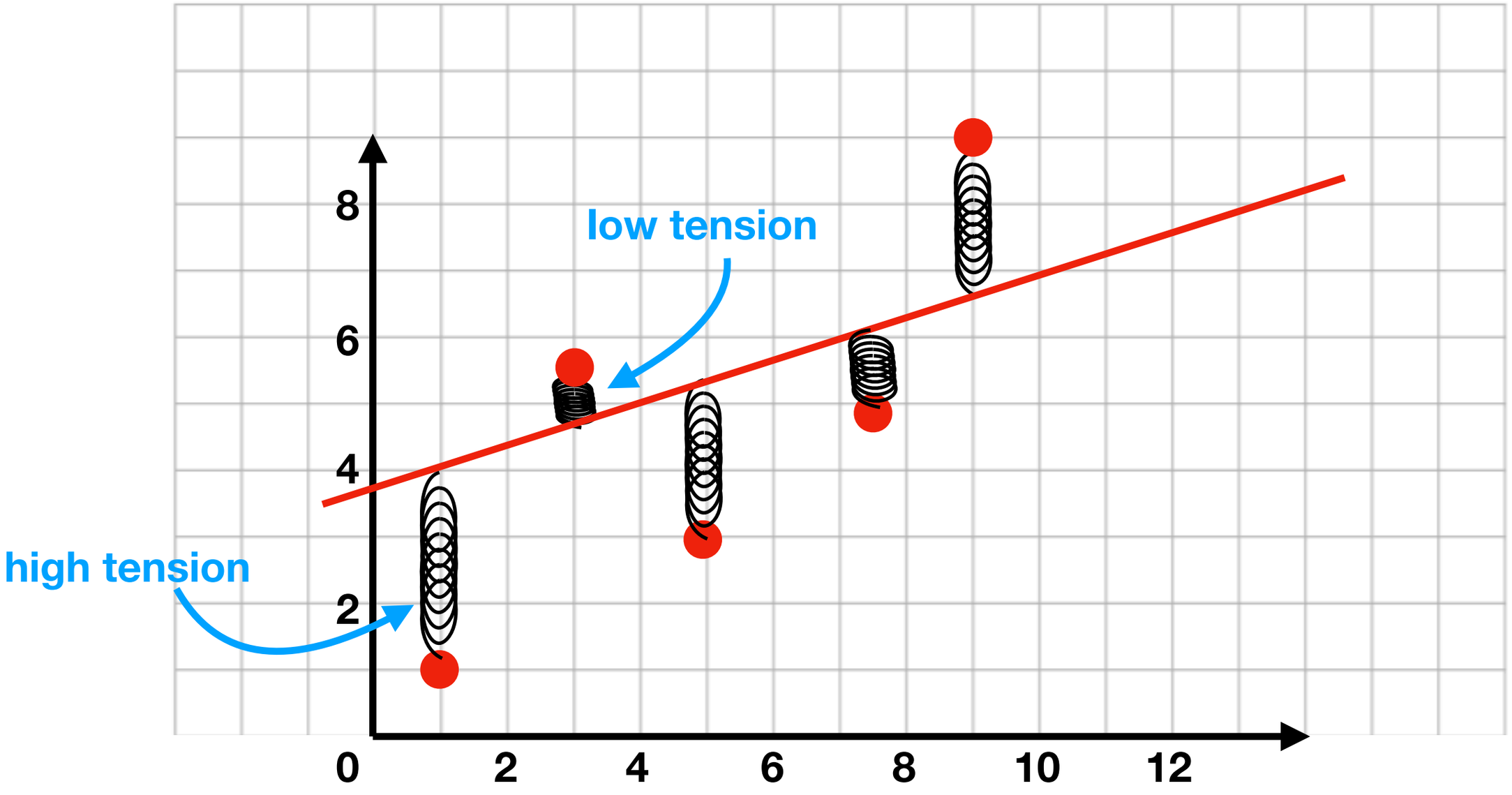
# Intuition: Springs



# Intuition: Springs

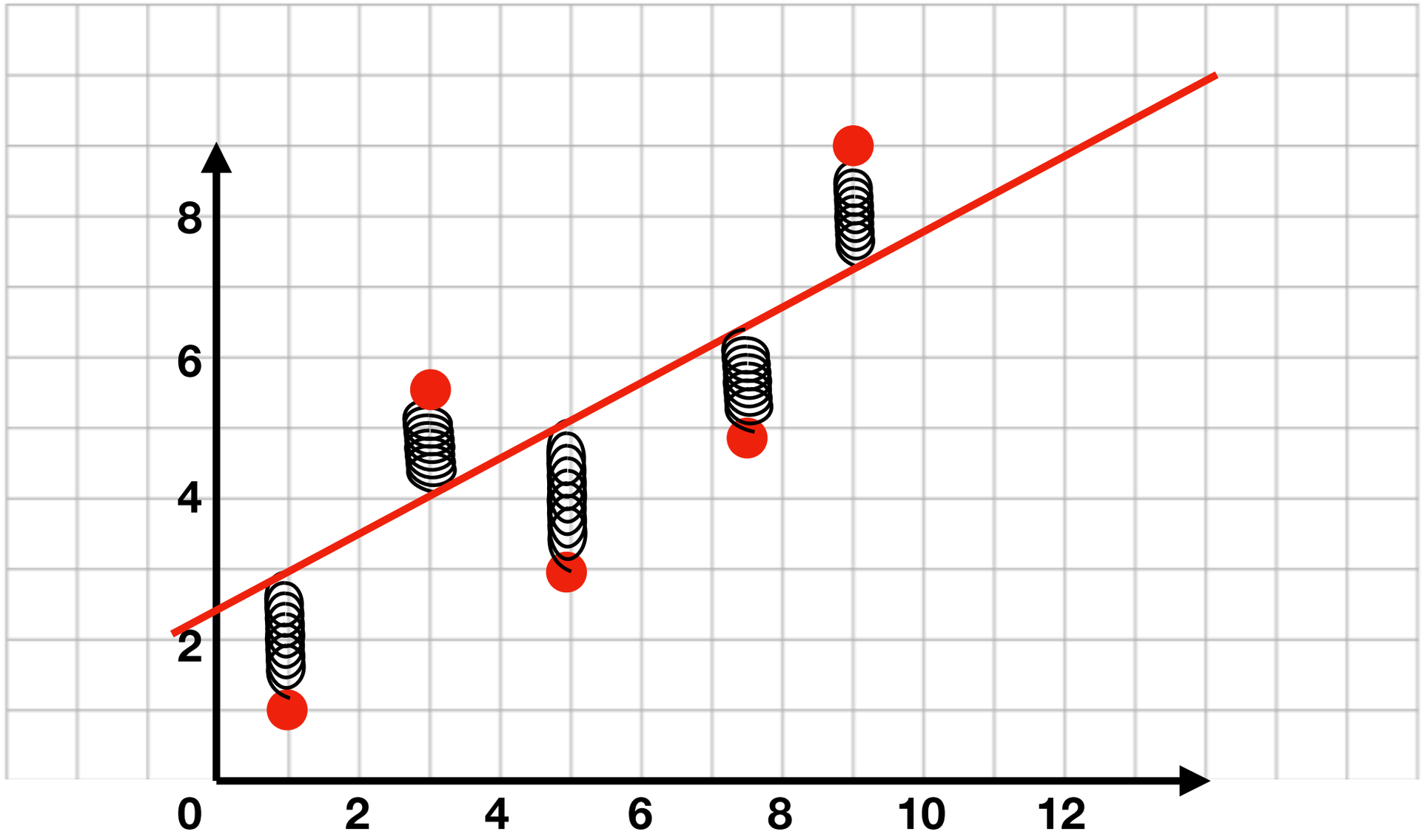


# Intuition: Springs



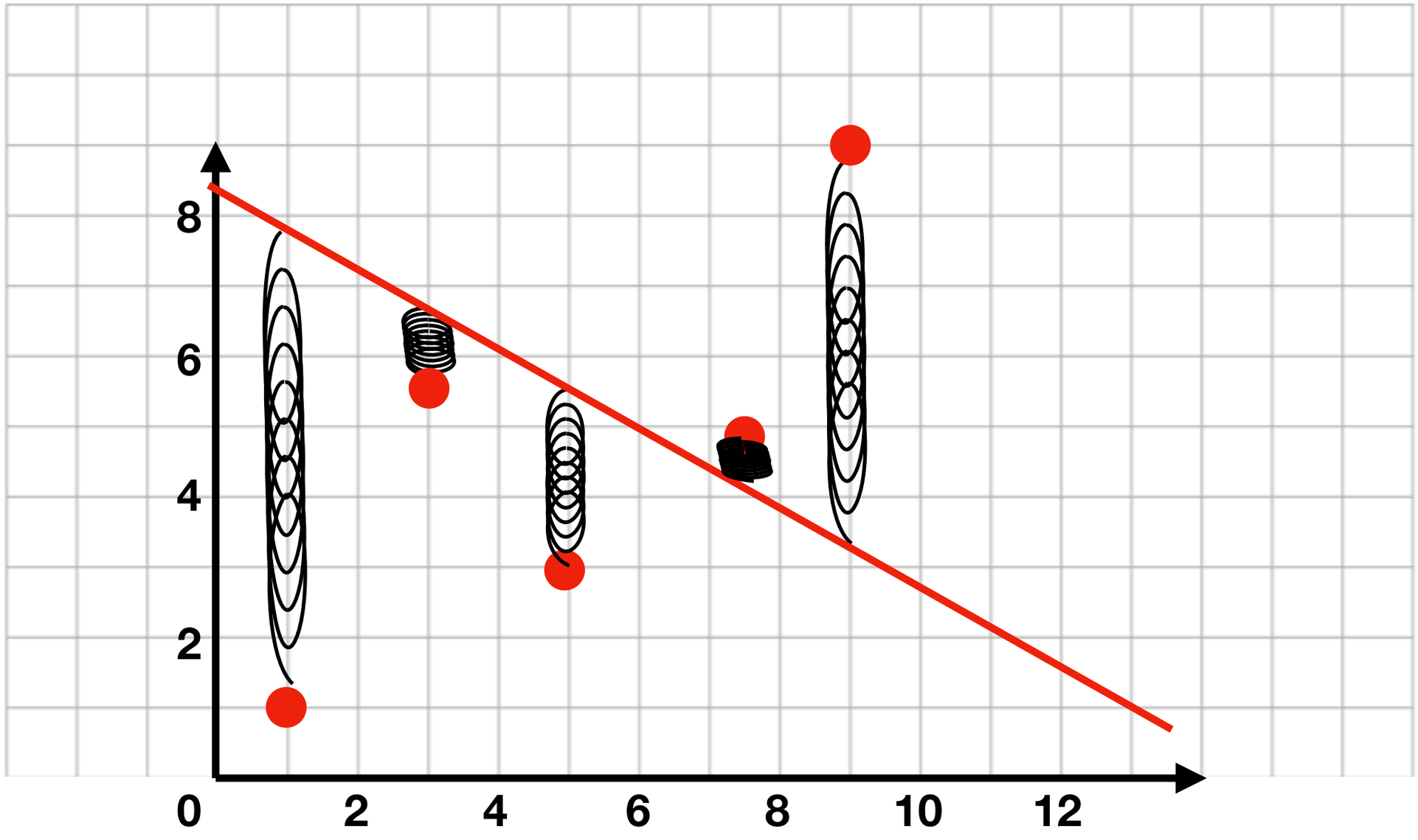

**The best line minimizes total tension across springs**

# Intuition: Springs



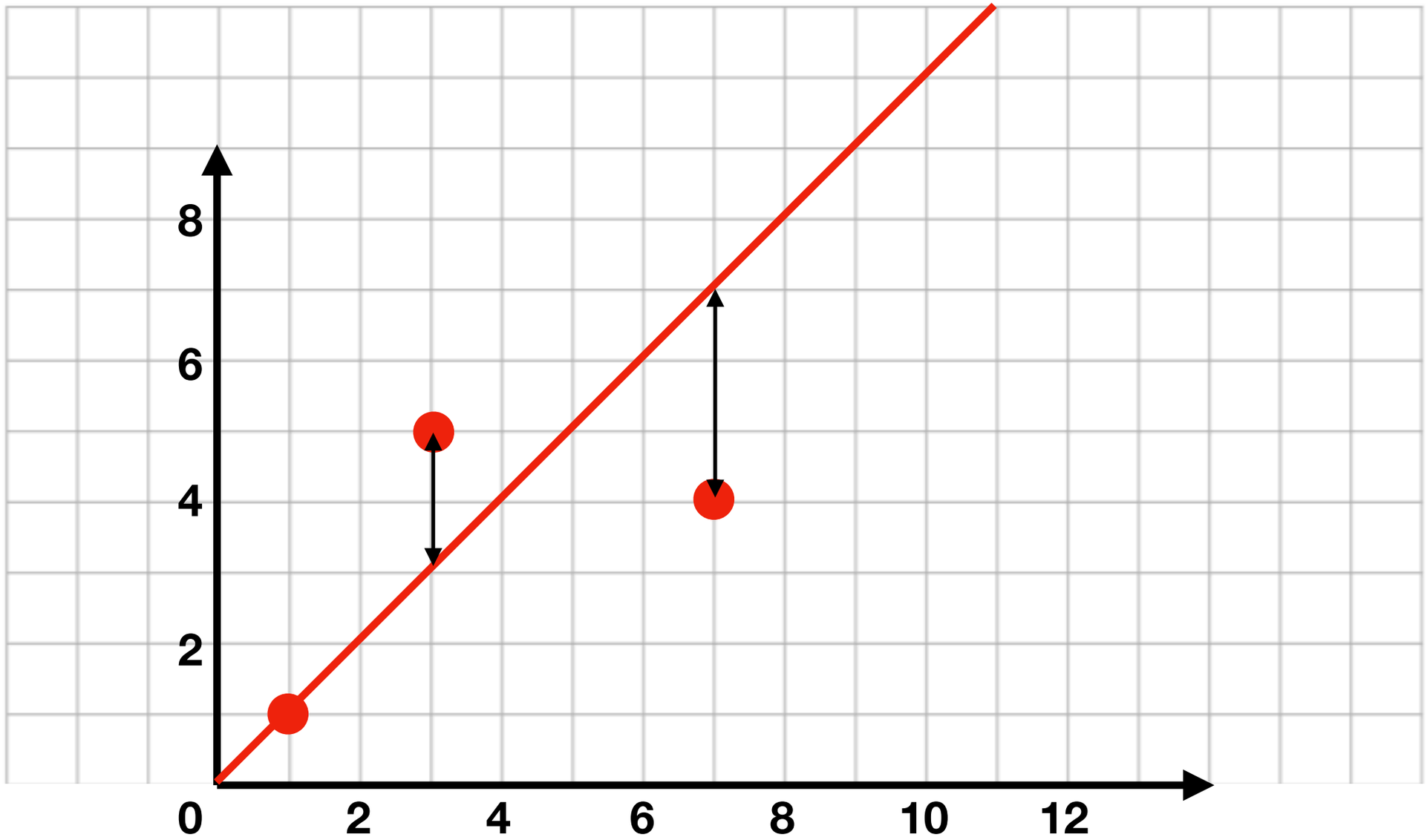

**Good fit with low overall tension**

# Intuition: Springs



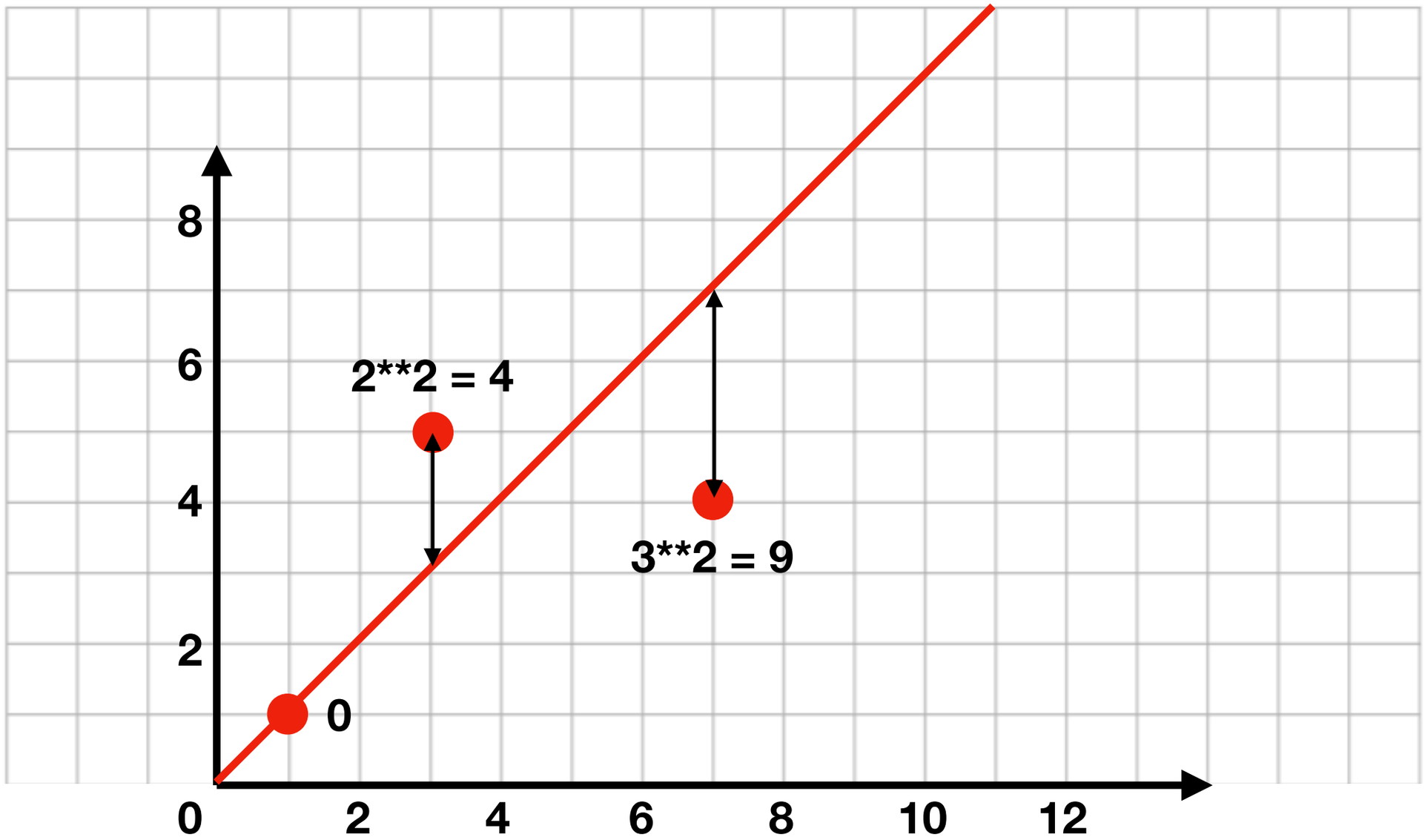

**Bad fit with high overall tension**

# Intuition: Springs



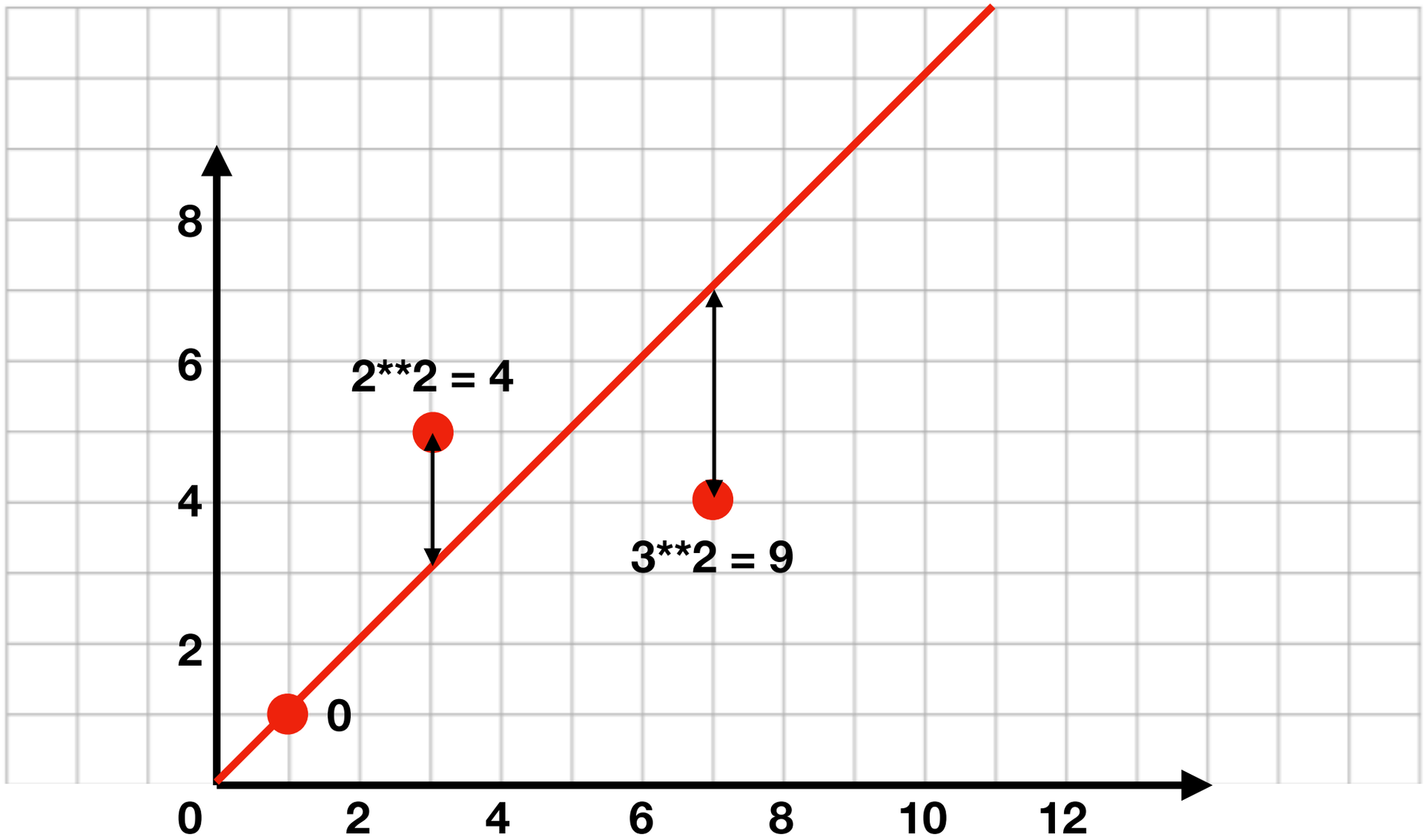

**Tension is defined as distance squared**

# Intuition: Springs




**Tension is defined as distance squared**

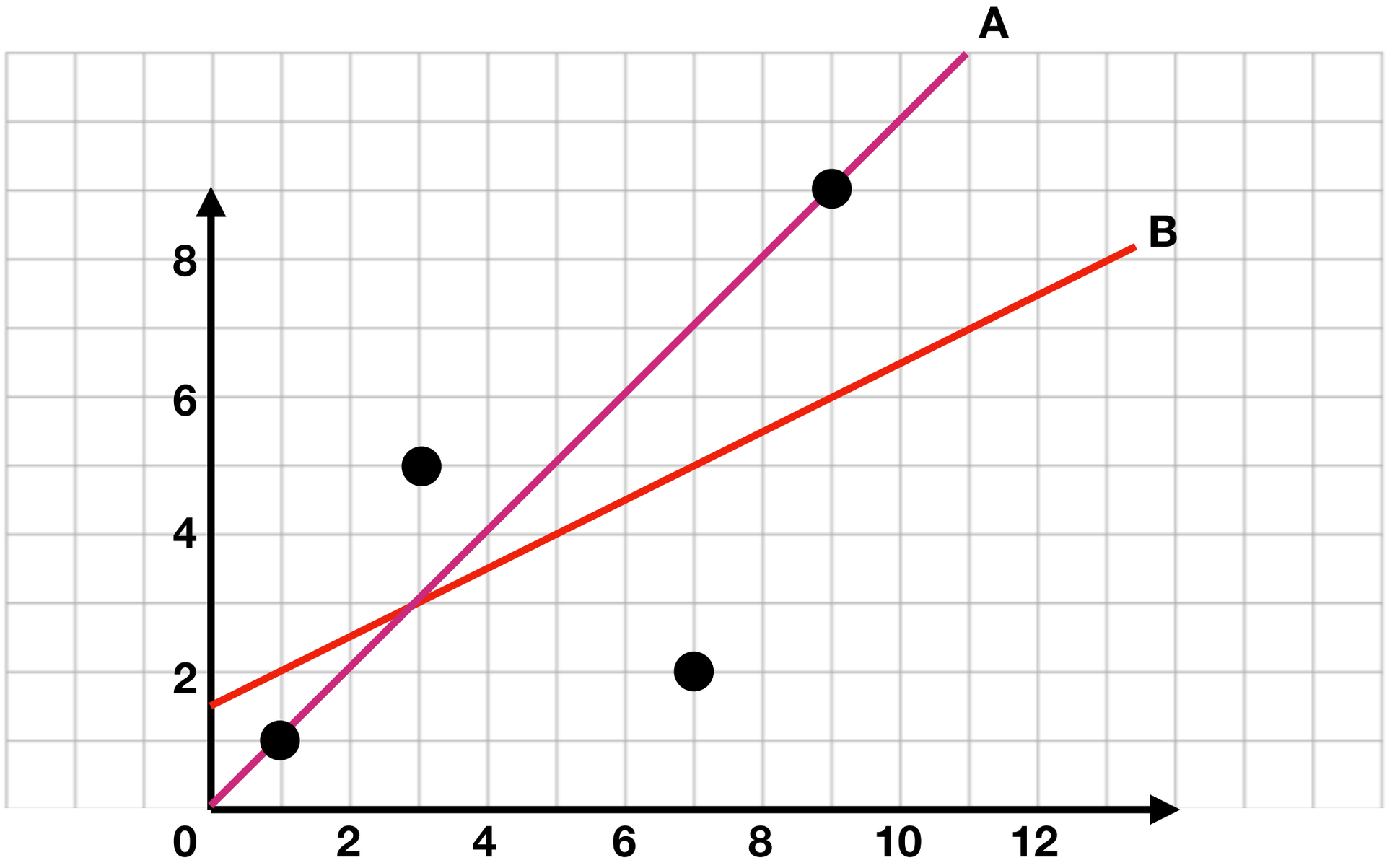
# Intuition: Springs




**Tension is defined as distance squared**

**Total: 4+9 = 13**

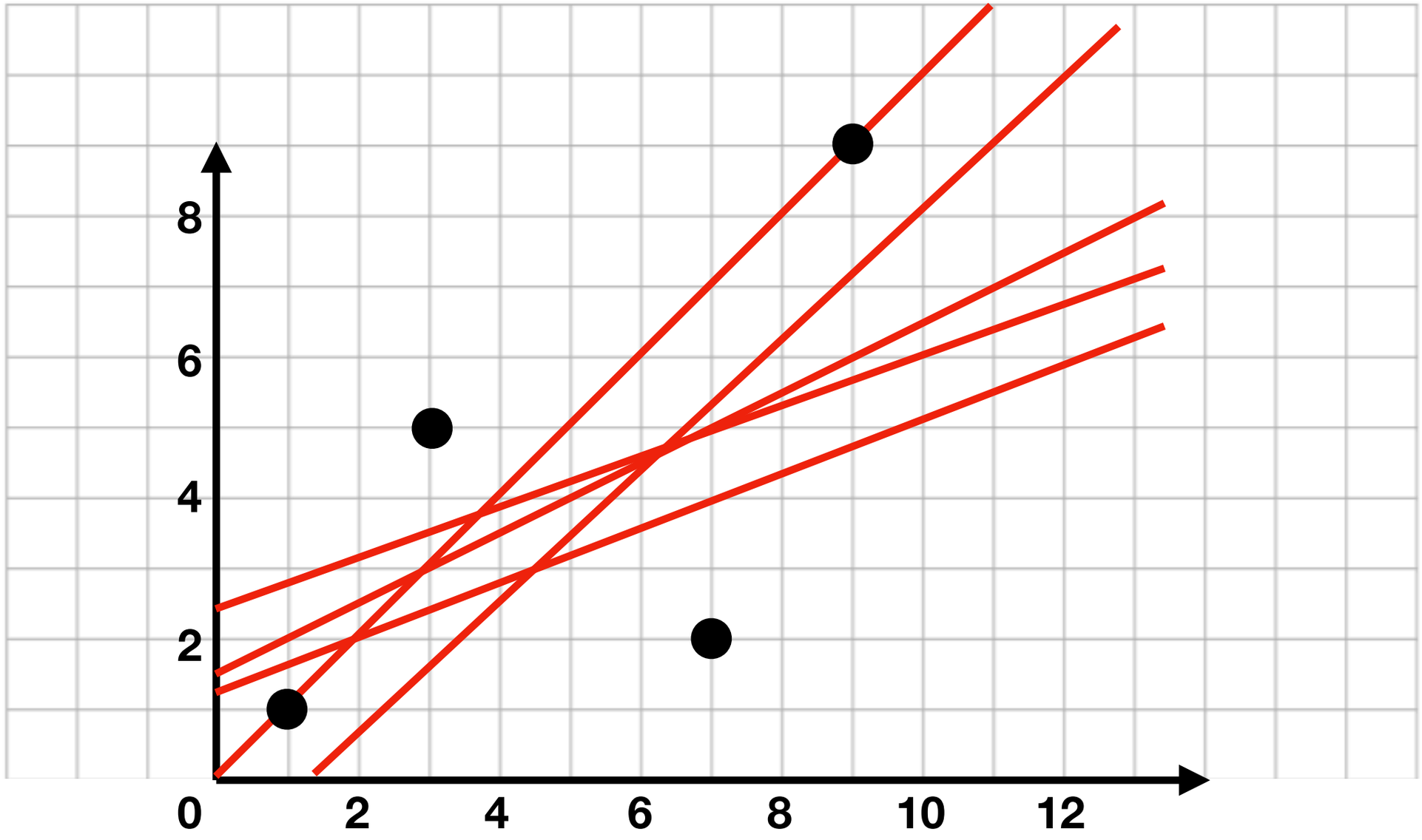
# Practice




**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

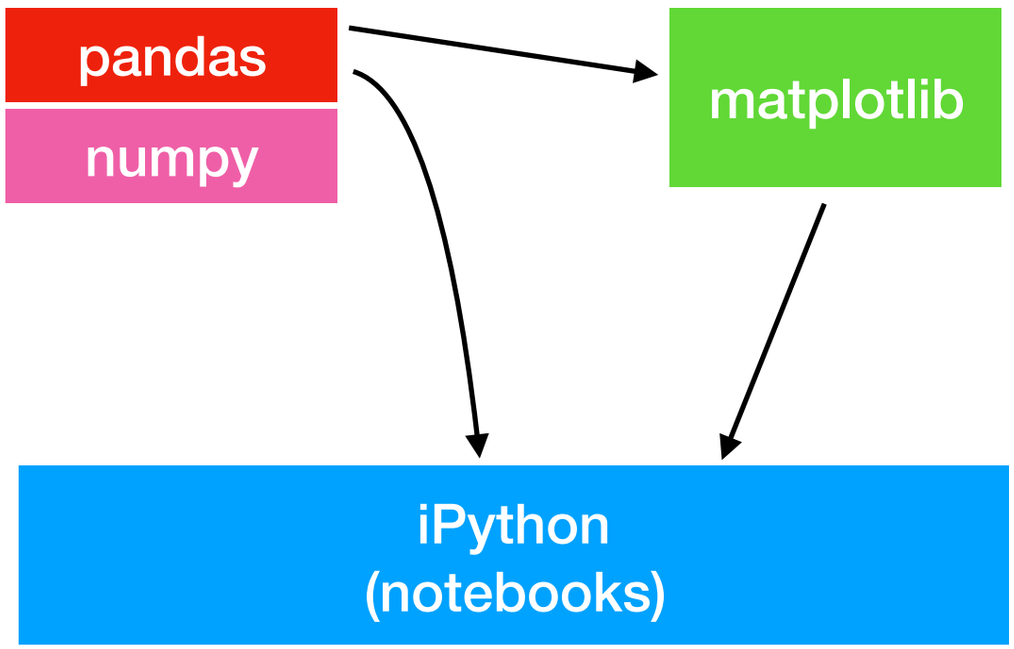
# Modules we've learned this semester

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

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

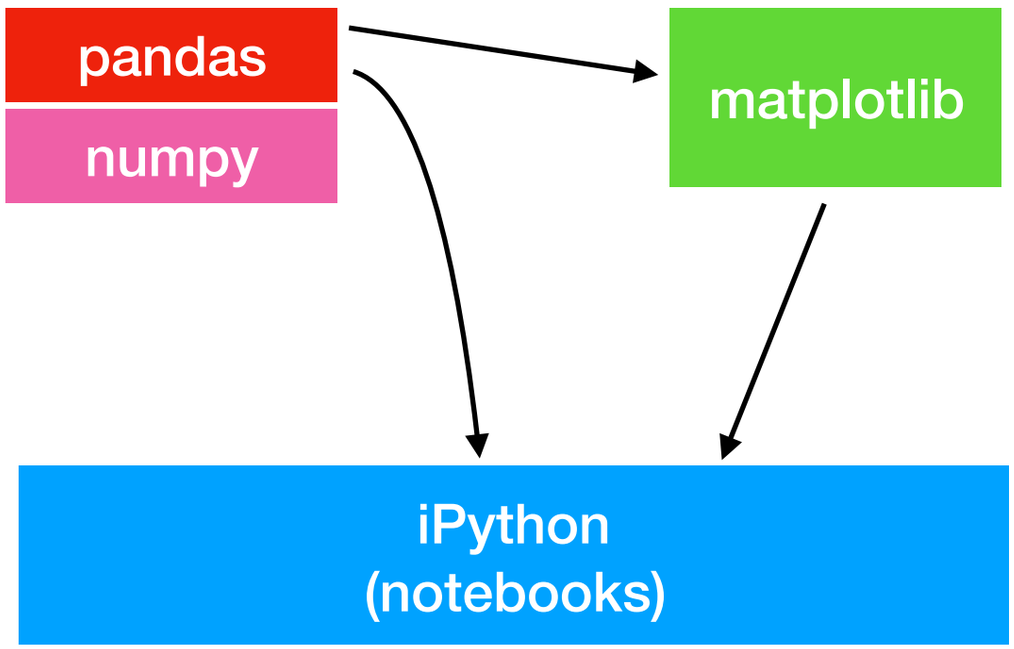
# Modules we've learned this semester

- 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
```

# numpy

```
import numpy as np

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

**array is the core data structure of numpy**

# numpy

```
import numpy as np

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

it can be initialized
from a Python list

array 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]
```

**indexing**

**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))
```

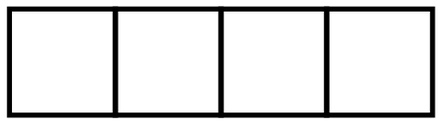
```
numpy.ndarray
```

**why is it called an ndarray?**

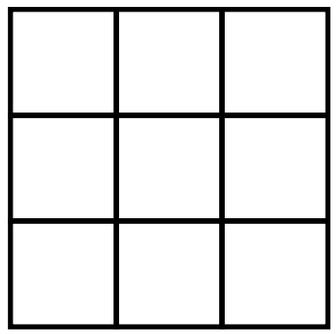
# numpy

```
import numpy as np

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

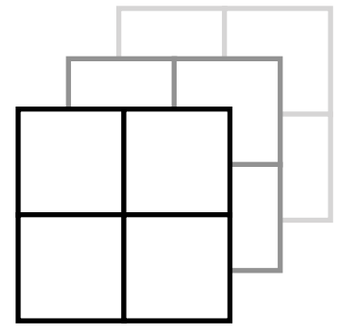
**1-dimensional array**



**2-dimensional array**



**3-dimensional array**



# numpy

```
import numpy as np

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

# numpy

```
import numpy as np

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

a.reshape(     )
```

# 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

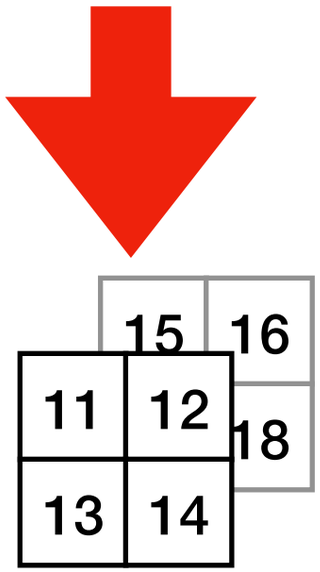
```
import numpy as np

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**

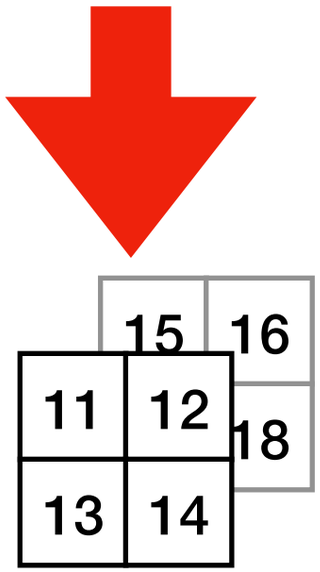
# numpy

```
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))
```

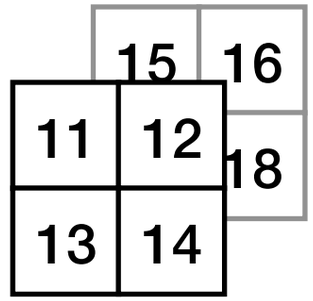



**default fill order:**

- layers
- rows
- columns

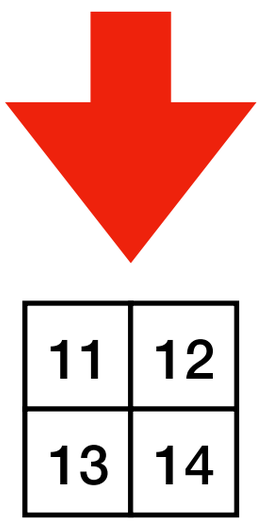
# numpy

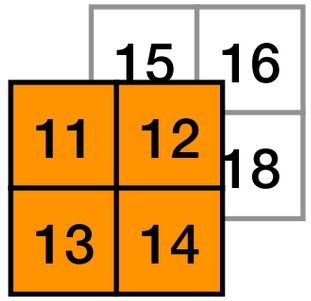
```
import numpy as np

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

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



# numpy

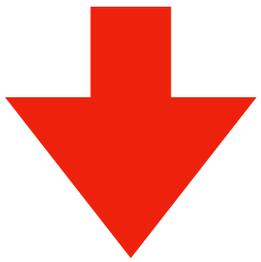
```
import numpy as np

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

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

b[0]
```



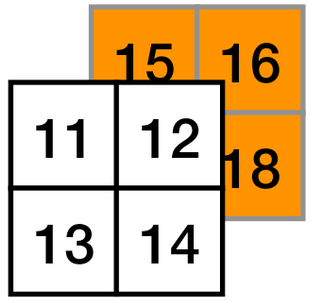



# numpy

```
import numpy as np

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

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

b[1]
```

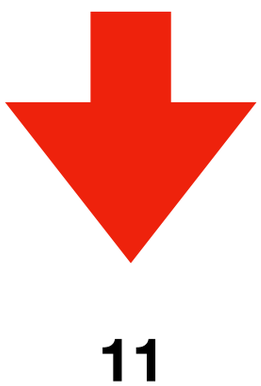


| 15 | 16 |
|---|---|
| 17 | 18 |

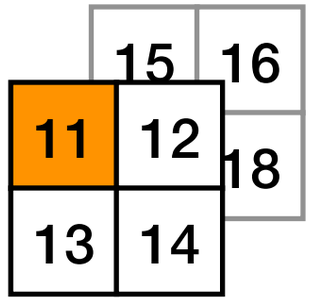


# numpy

```
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]
```



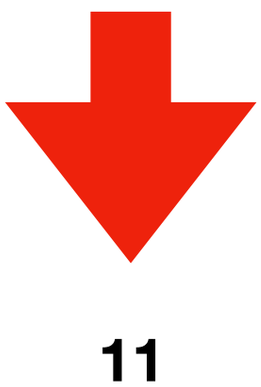
11



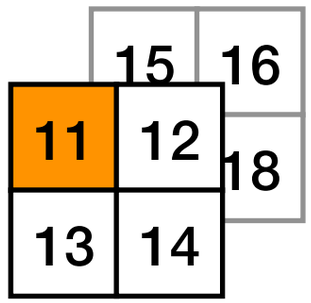

**indexing:** `ndarray[layer, row, col]`

# 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



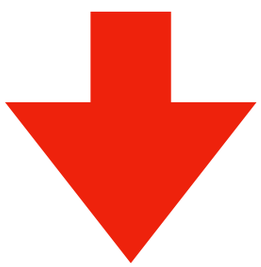

**indexing:** `ndarray[layer, row, col]`

contrast with indexing into a list of lists of lists:

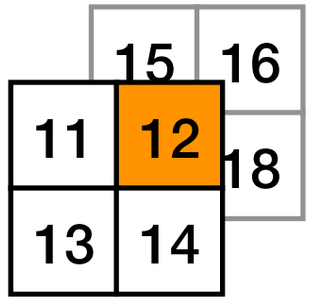
`data[layer][row][col]`

# 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]
```



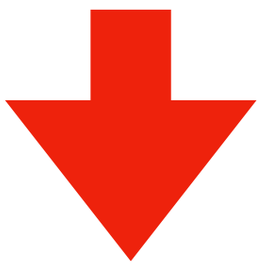
12



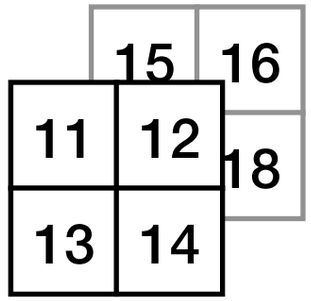

**indexing:** `ndarray[layer, row, col]`

# 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]
```



???

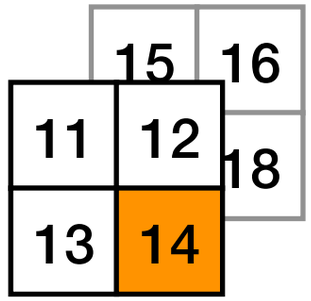



**indexing:** `ndarray[layer, row, col]`

# 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]
```

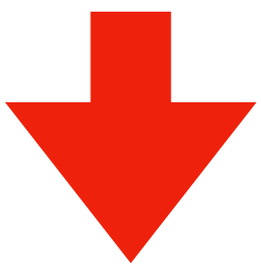
14



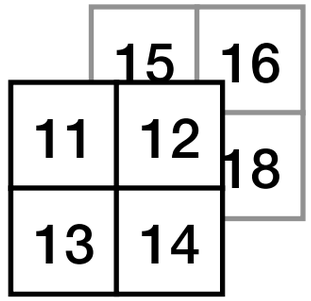

**indexing:** `ndarray[layer, row, col]`

# 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]
```



???




**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”]
```

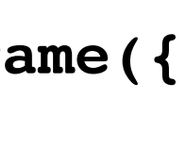
# Pandas and numpy

```
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 |

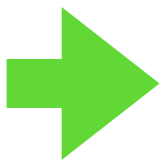
you've been using numpy arrays without knowing it!

```
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,)
```

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

# Learning Objectives Today

History of regression

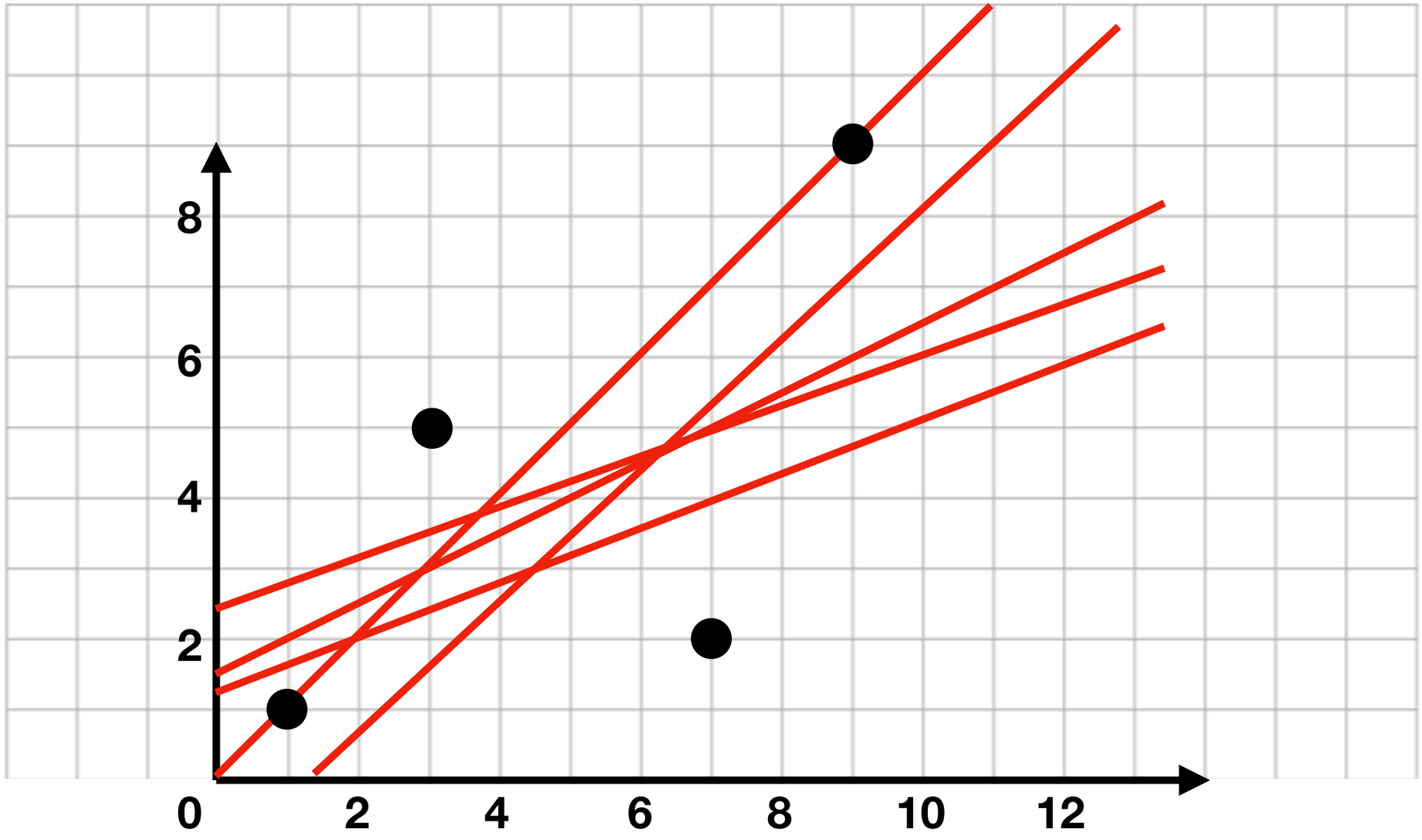
Drawing a fit line

Finding the slope/intercept w/ least squares method

Numpy introduction

Using numpy.linalg.lstsq

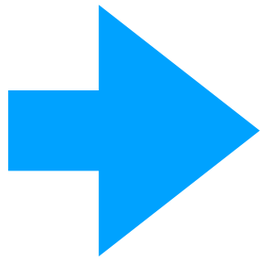
# Use numpy to solve this!




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

# Example data

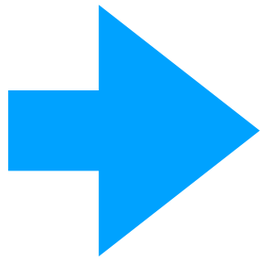
```
df = DataFrame({
     "x": [1,2,3,4],
     "y": [2,5,6,5]
})
```



| | x | y |
|---|---|---|
| **0** | 1 | 2 |
| **1** | 2 | 5 |
| **2** | 3 | 6 |
| **3** | 4 | 5 |

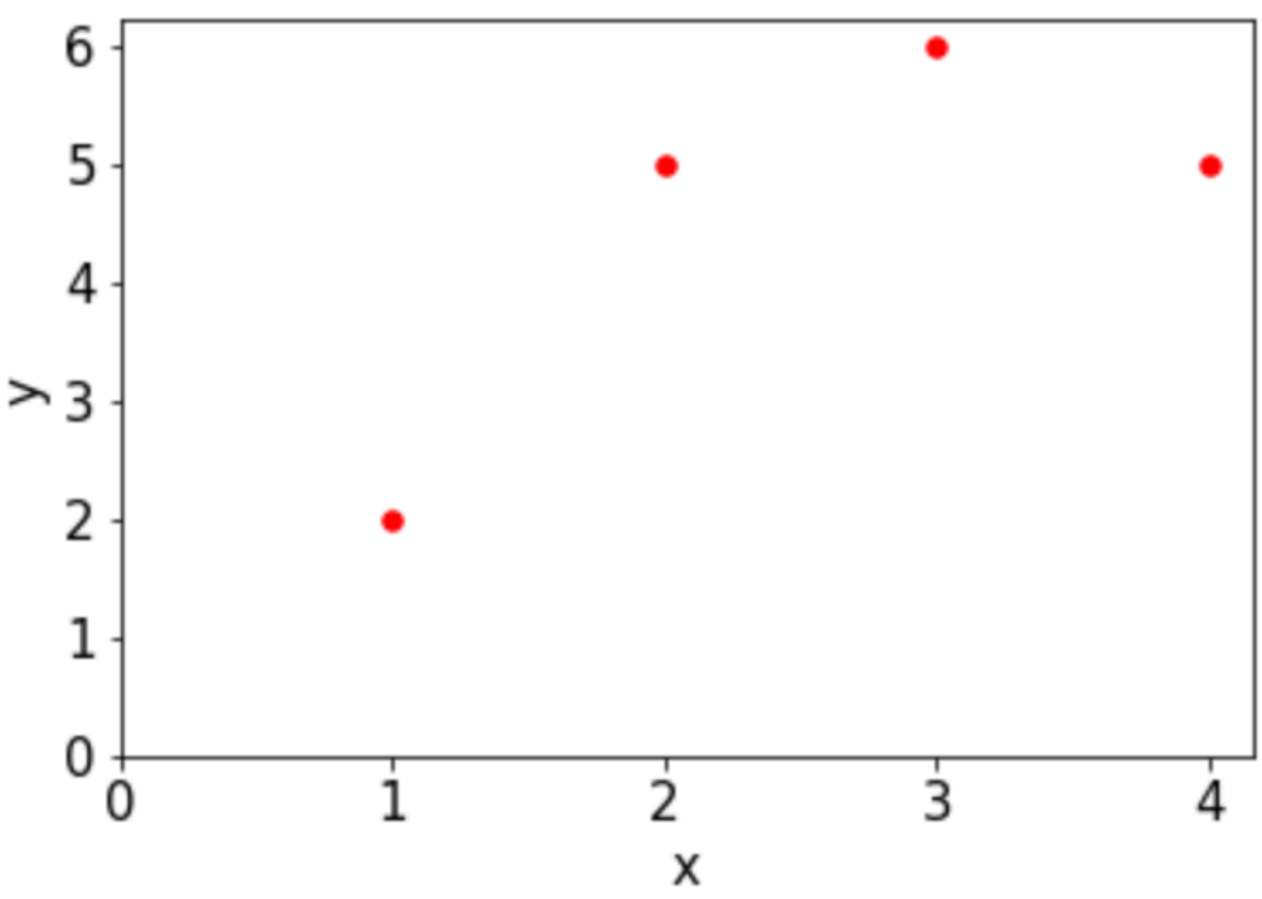
# Example data

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



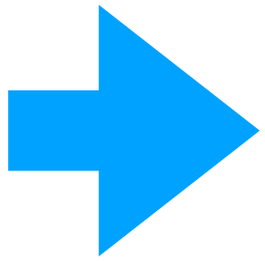
| | x | y |
|---|---|---|
| **0** | 1 | 2 |
| **1** | 2 | 5 |
| **2** | 3 | 6 |
| **3** | 4 | 5 |

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



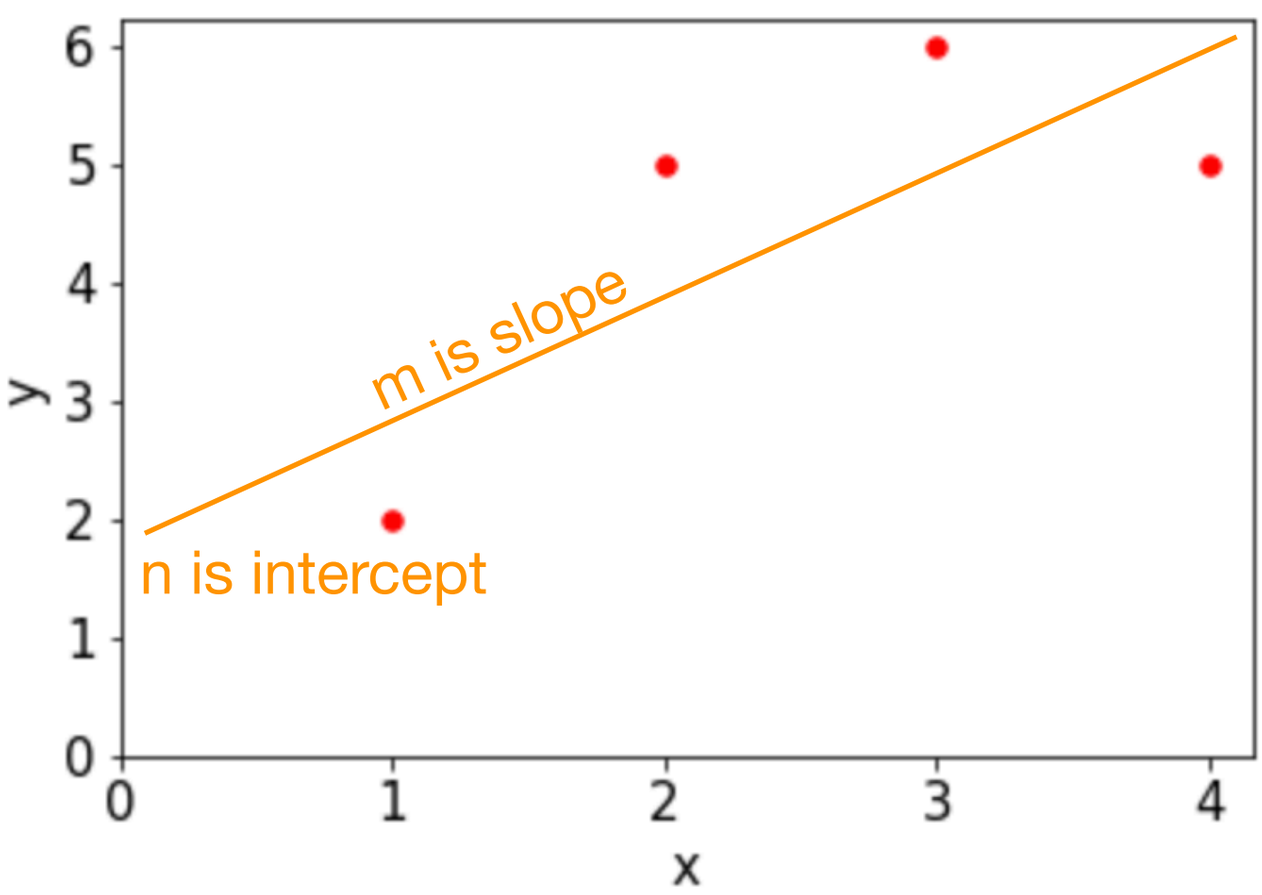
# Example data

```
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)

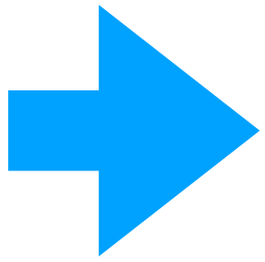



**we want a formula like this:**

$$m*x + n = y$$

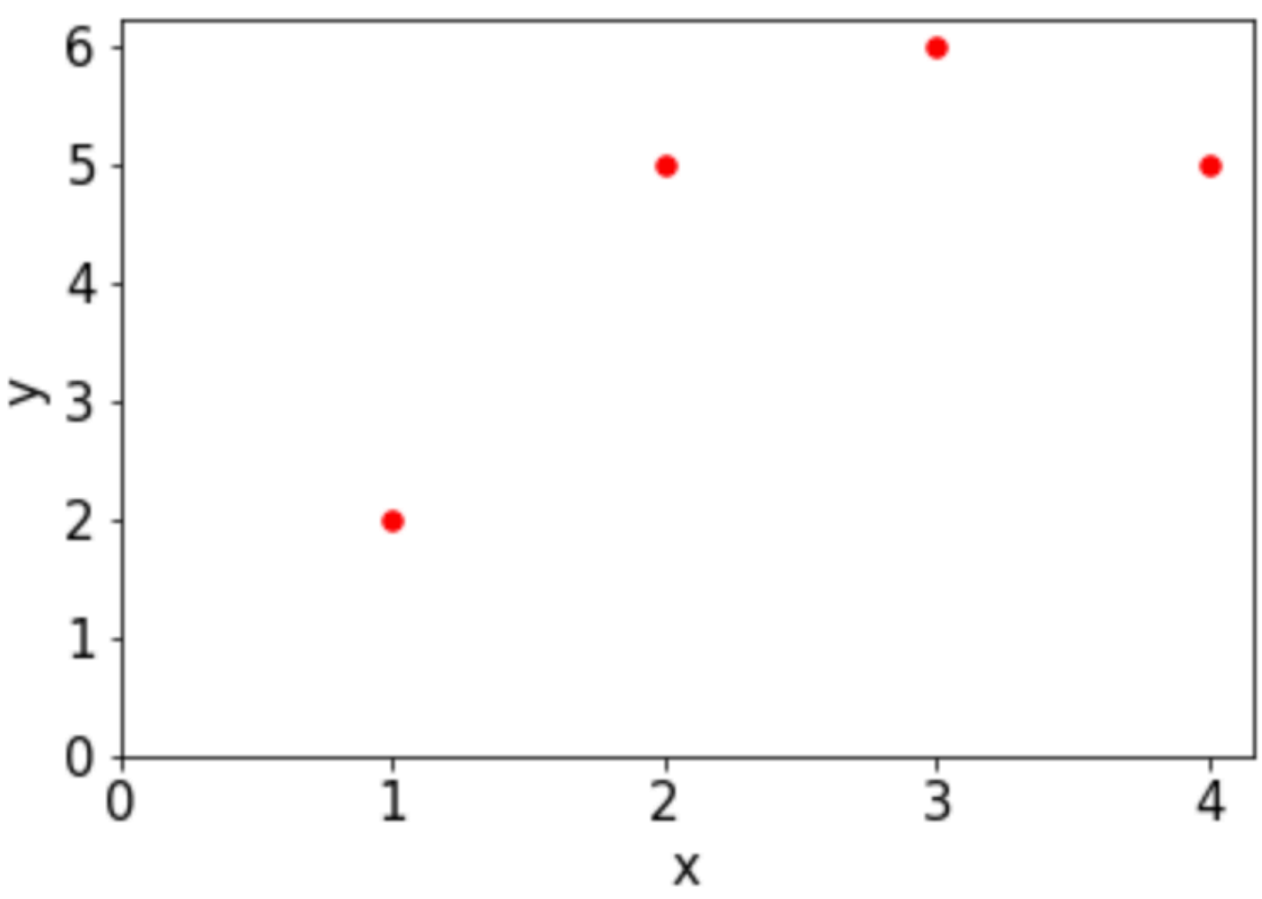
# Example data

```
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)

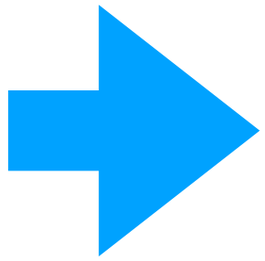



**we want a formula like this:**

$$m*x + n = y$$

# Example data

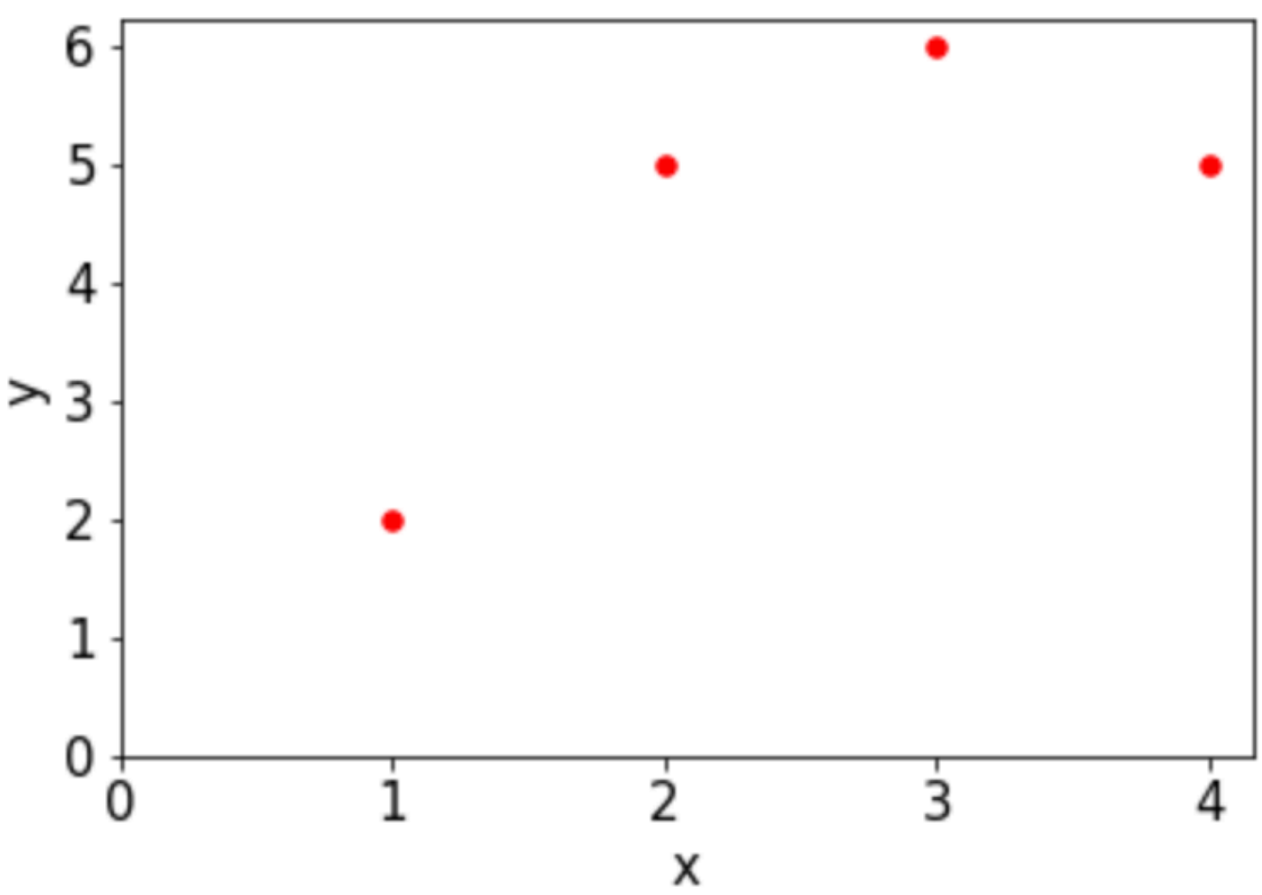
```
df = DataFrame({
     "x": [1,2,3,4],
     "y": [2,5,6,5]
})
```



*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)

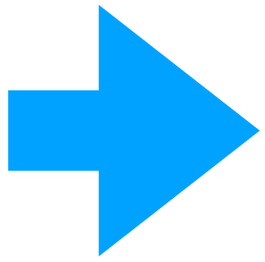



**we want a formula like this:**

$$m*x + n = y$$

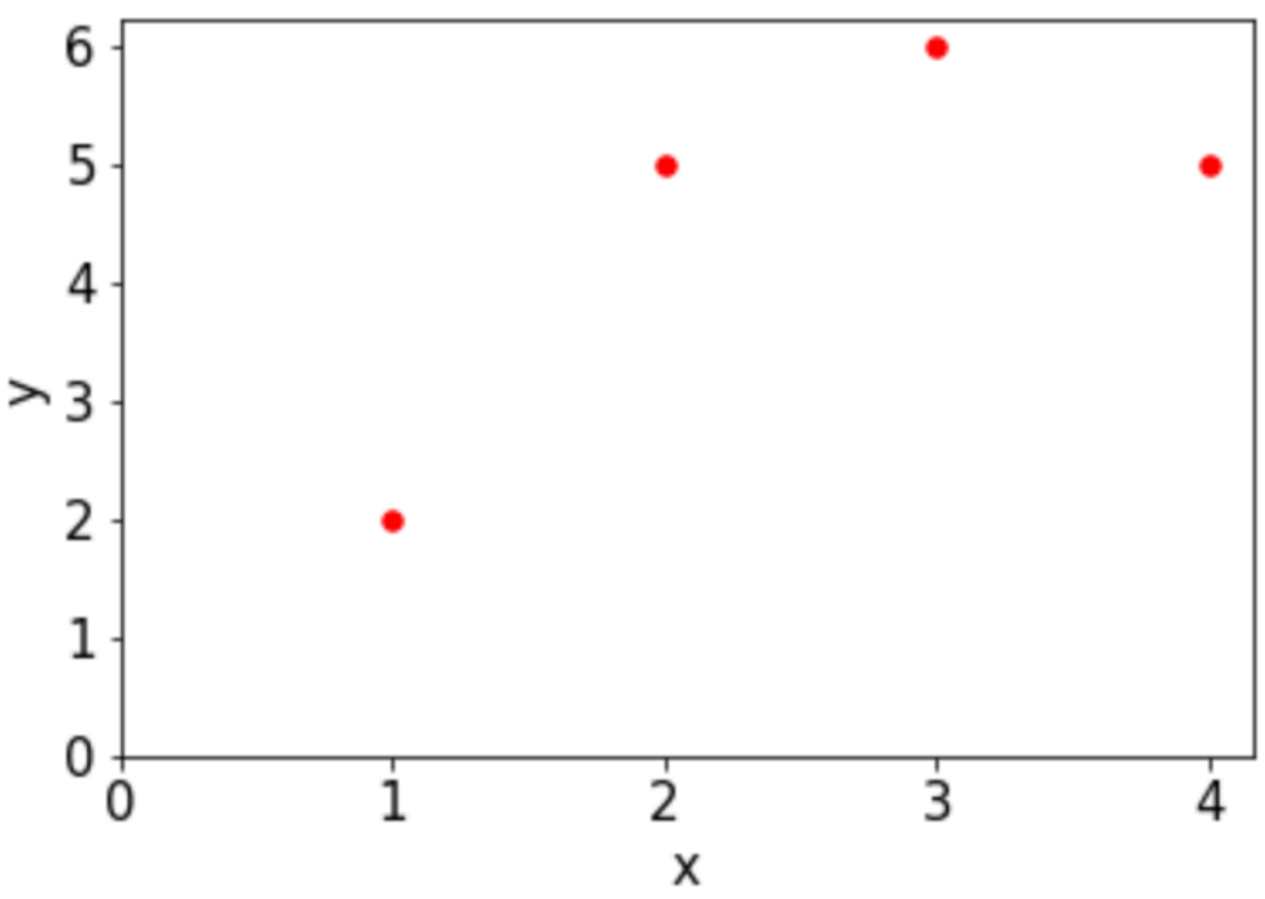
# Example data

```
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)

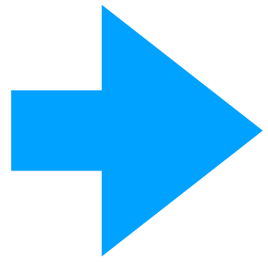



**we want a formula like this:**

$$m*x + n = y$$

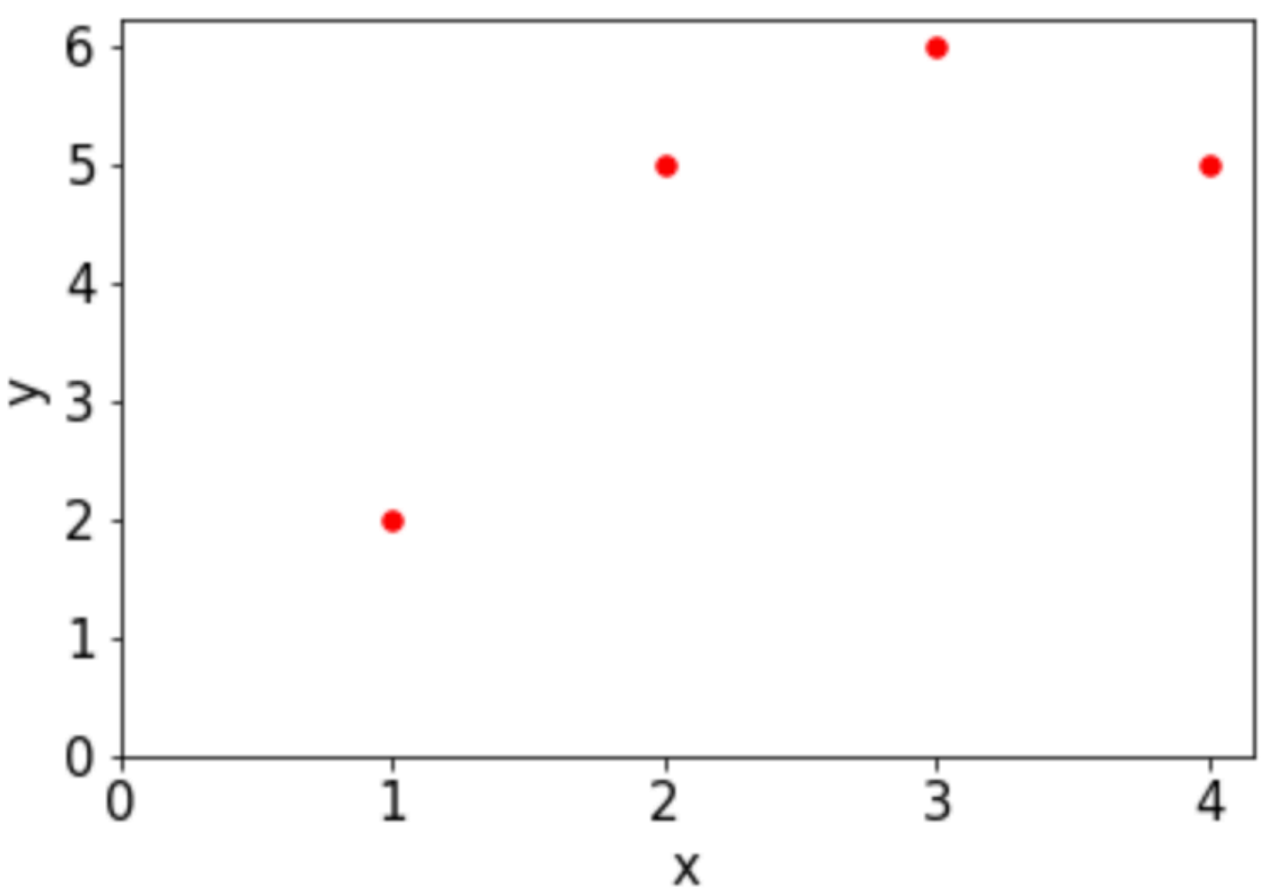
# Example data

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



| m * x | + n ≈ y |
|---|---|
| 1 | 2 |
| 2 | 5 |
| 3 | 6 |
| 4 | 5 |

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

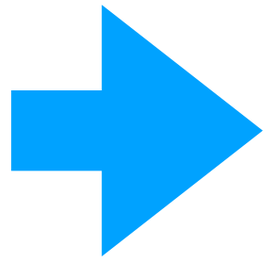



**we want a formula like this:**

$$m*x + n = y$$

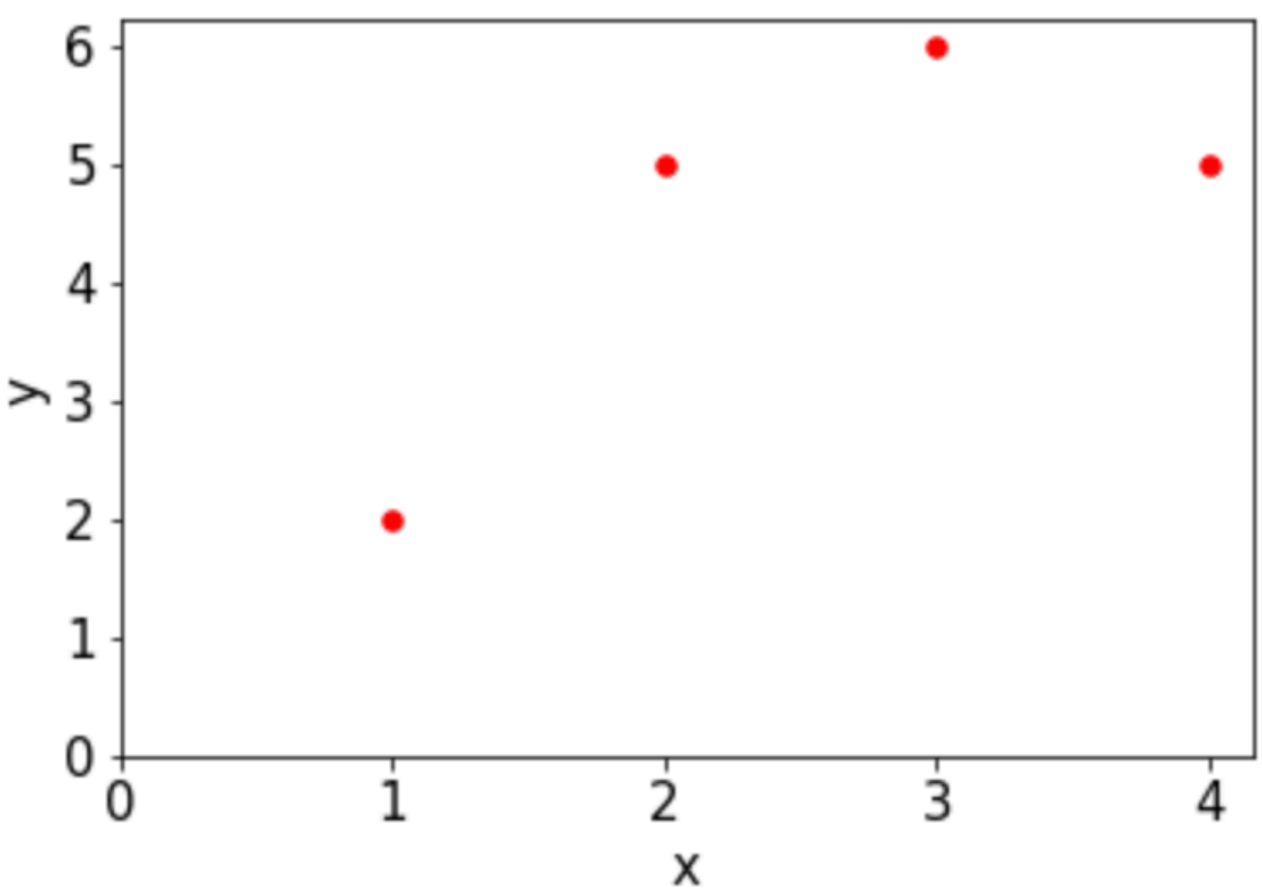
# Example data

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



| m * | x | + n * | 1 | ≈ | y |
|---|---|---|---|---|---|
| | 1 | | 1 | | 2 |
| | 2 | | 1 | | 5 |
| | 3 | | 1 | | 6 |
| | 4 | | 1 | | 5 |

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

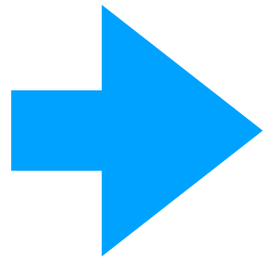



**we want a formula like this:**

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

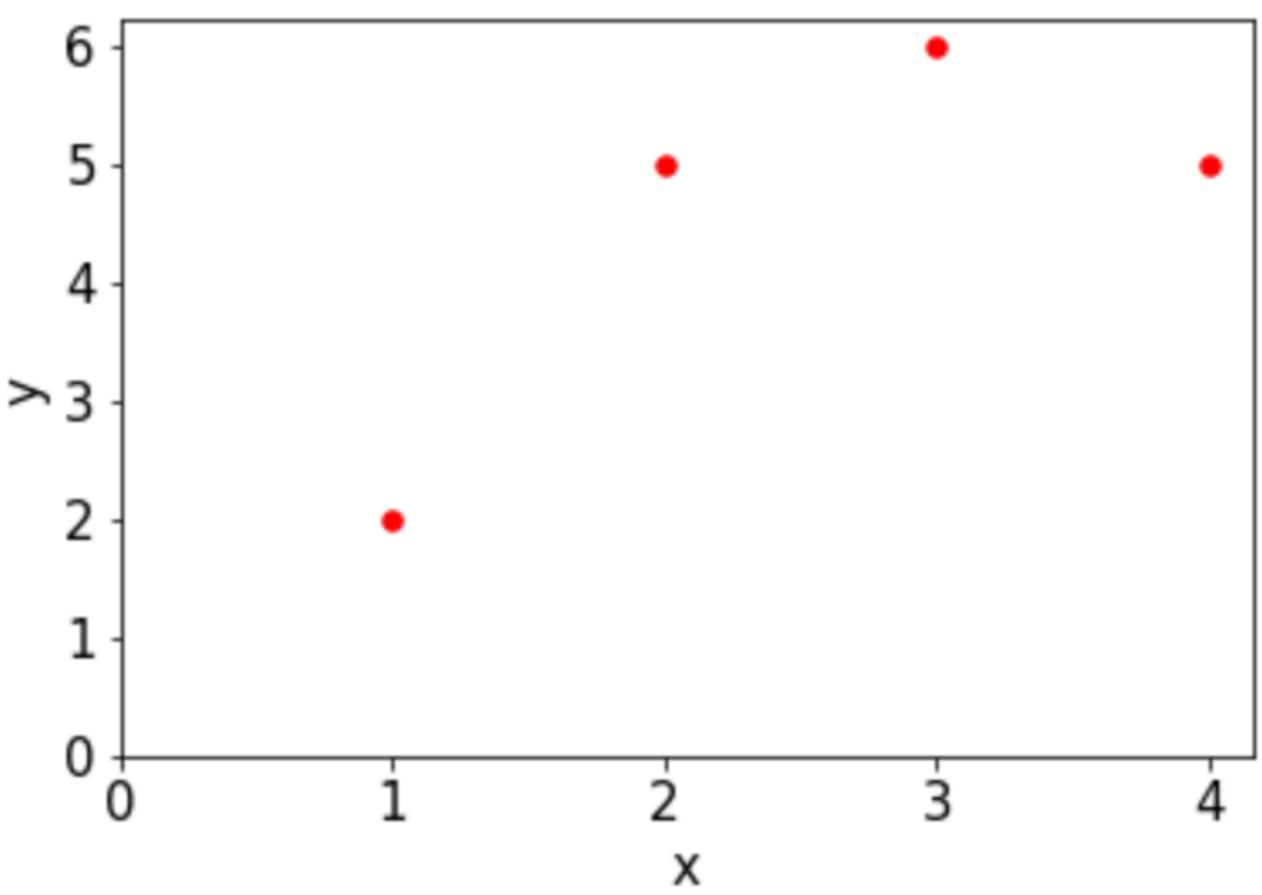
# Example data

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



| m * | x | + n * | 1 | ≈ | y |
|---|---|---|---|---|---|
| | 1 | | 1 | | 2 |
| | 2 | | 1 | | 5 |
| | 3 | | 1 | | 6 |
| | 4 | | 1 | | 5 |

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

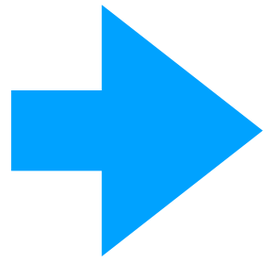



**we want a formula like this:**

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

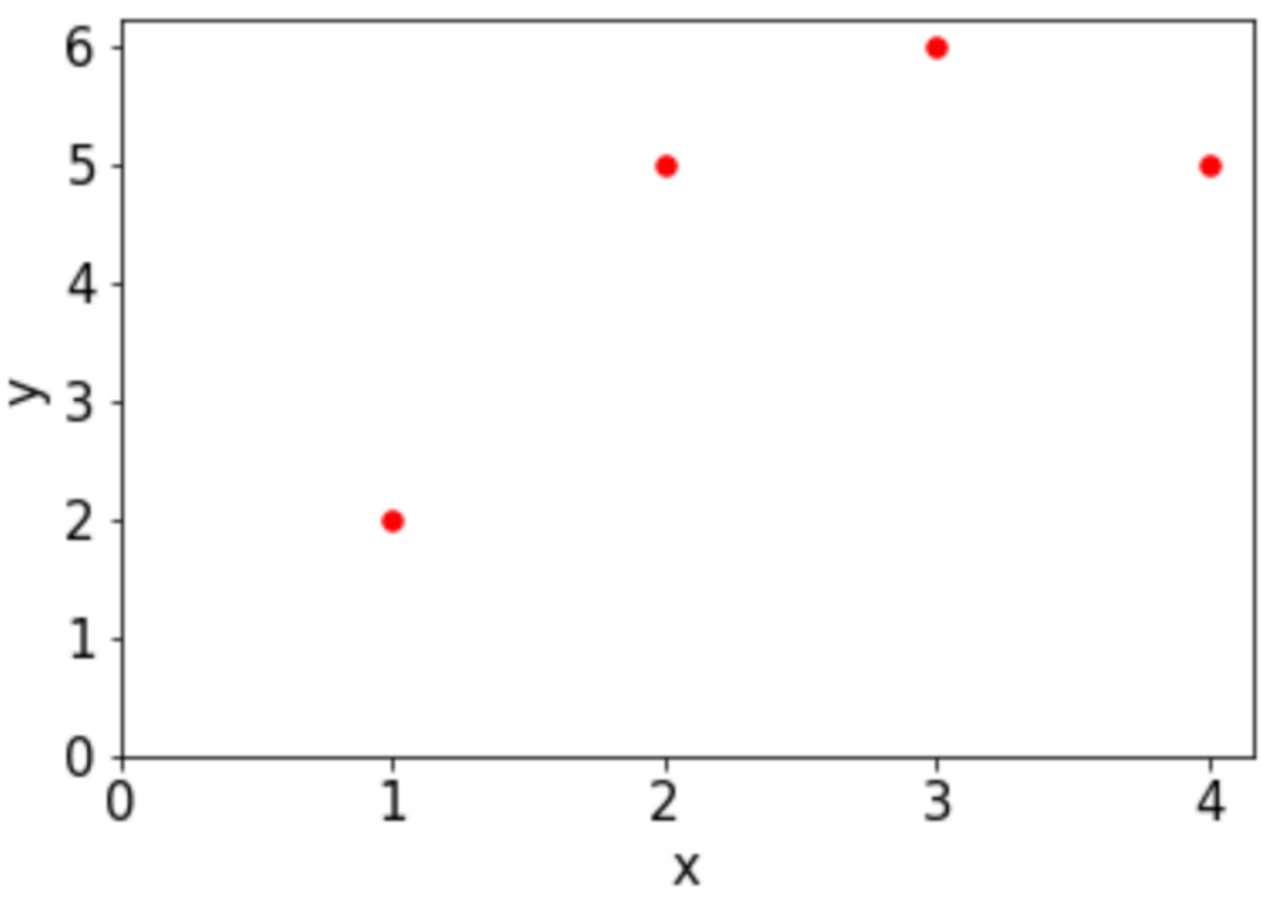
# Example data

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



| m * | x | + n * | 1 | ≈ | y |
|---|---|---|---|---|---|
| | 1 | | 1 | | 2 |
| | 2 | | 1 | | 5 |
| | 3 | | 1 | | 6 |
| | 4 | | 1 | | 5 |

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

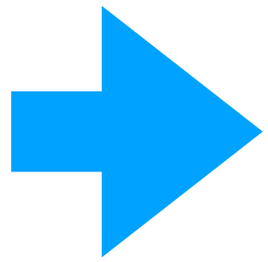


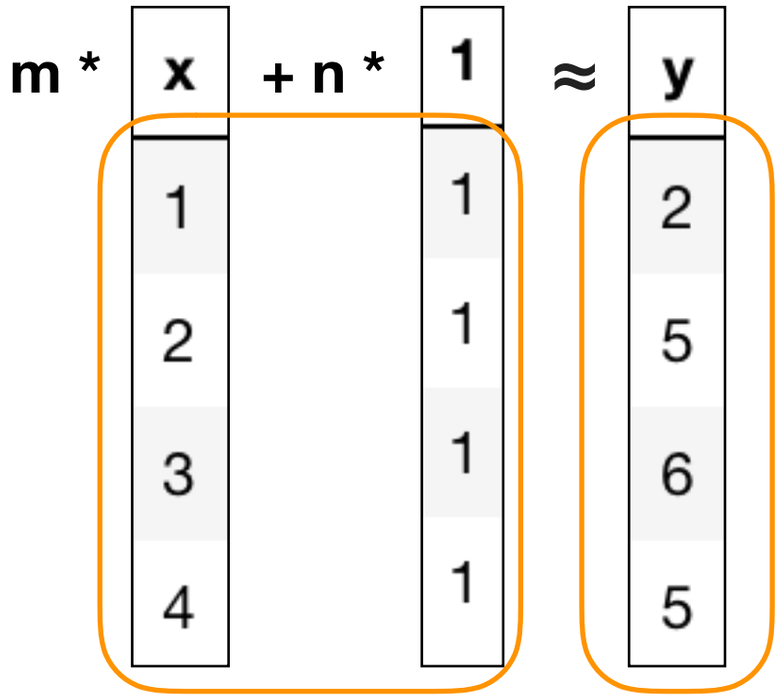

**we want a formula like this:**

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

# Example data

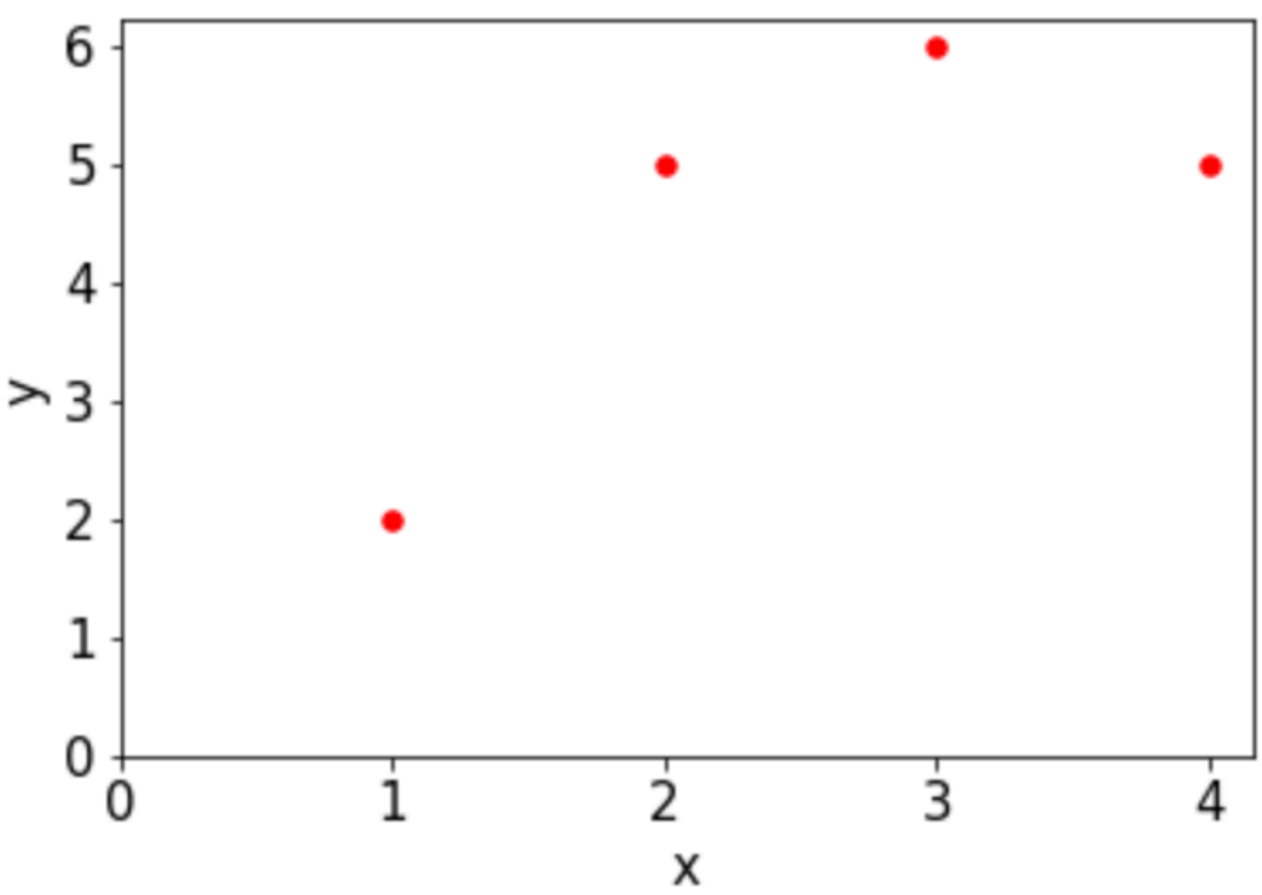
```
df = DataFrame({
     "x": [1,2,3,4],
     "1": np.ones(4),
     "y": [2,5,6,5]
})
```






given these inputs, as ndarrays…

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

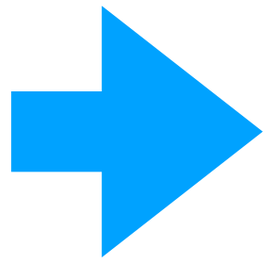



**we want a formula like this:**

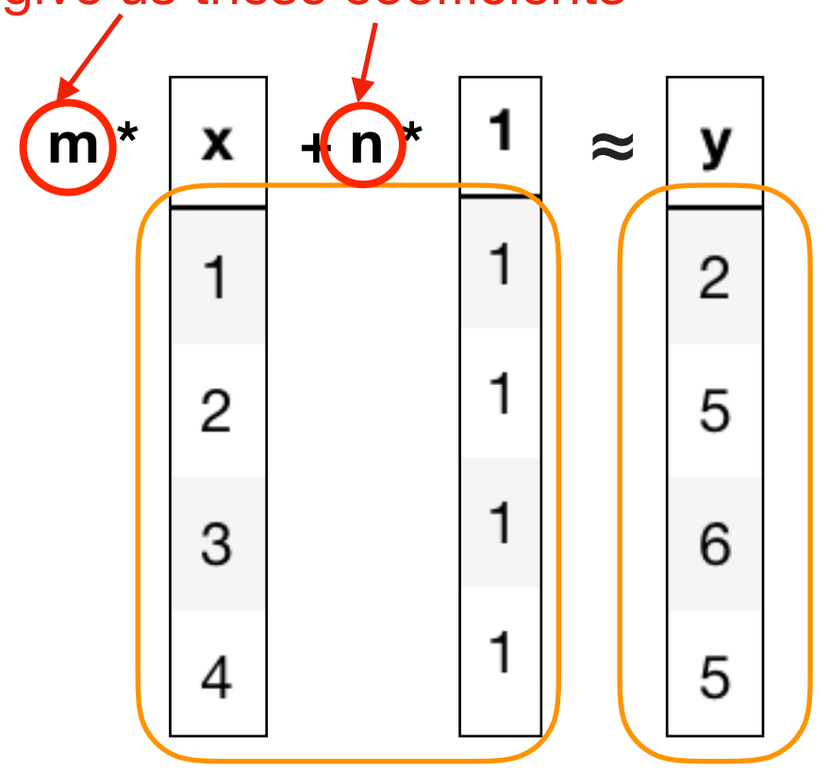
$$m*x + n*1 = y$$

# Example data

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

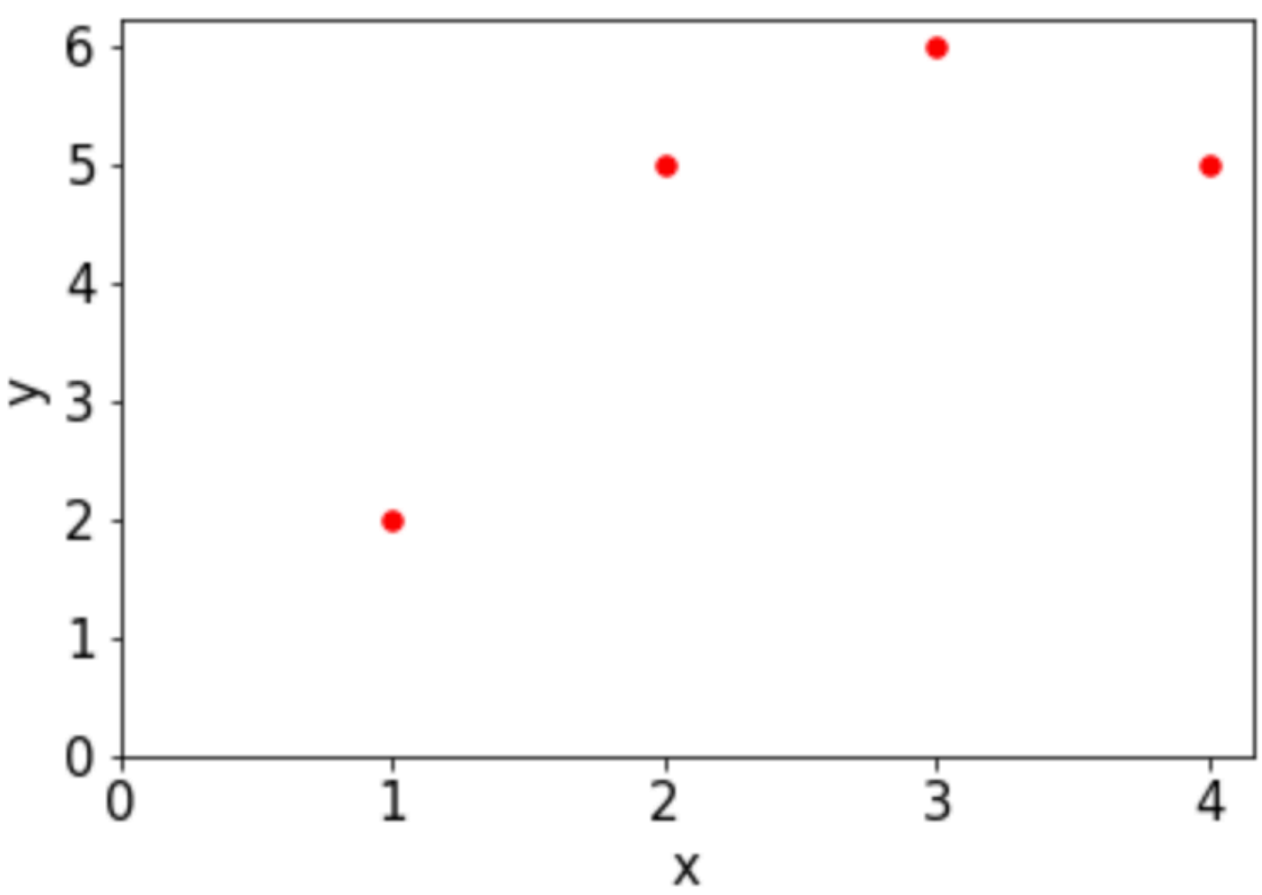


…numpy will give us these coefficients




given these inputs, as ndarrays…

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




**we want a formula like this:**

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