`SciPy` (linear regression, 1-D and 2-D interpolation)

ESS 116 | Fall 2024

Prof. Henri Drake, Prof. Jane Baldwin, and Prof. Michael Pritchard

(Modified from Ethan Campbell and Katy Christensen's materials for UW's Ocean 215)

What we’ll cover in this lesson

1. **`SciPy`: linear regression**

2. `SciPy`: 1-D and 2-D interpolation/regridding

The SciPy (Scientific Python) package

`scipy.cluster`	Vector quantization / Kmeans
`scipy.constants`	Physical and mathematical constants
`scipy.fftpack`	Fourier transform
`scipy.integrate`	Integration routines
`scipy.interpolate`	Interpolation
`scipy.io`	Data input and output
`scipy.linalg`	Linear algebra routines
`scipy.ndimage`	n-dimensional image package
`scipy.odr`	Orthogonal distance regression
`scipy.optimize`	Optimization
`scipy.signal`	Signal processing
`scipy.sparse`	Sparse matrices
`scipy.spatial`	Spatial data structures and algorithms
`scipy.special`	Any special mathematical functions
`scipy.stats`	Statistics

Useful constant values (e.g. gravitational constant, Stefan-Boltzmann constant) and unit conversions (e.g. nautical miles to miles)

Differential equation solvers

We'll use this module for 1-D and 2-D interpolation

Read and write odd file formats (e.g. MATLAB files)

Filtering, Fourier/spectral analysis

We'll use this module for linear regression
Also available: statistical tests (t-test, chi-squared test)

API reference: https://docs.scipy.org/doc/scipy/reference/index.html

Image credit: scipy-lectures.org

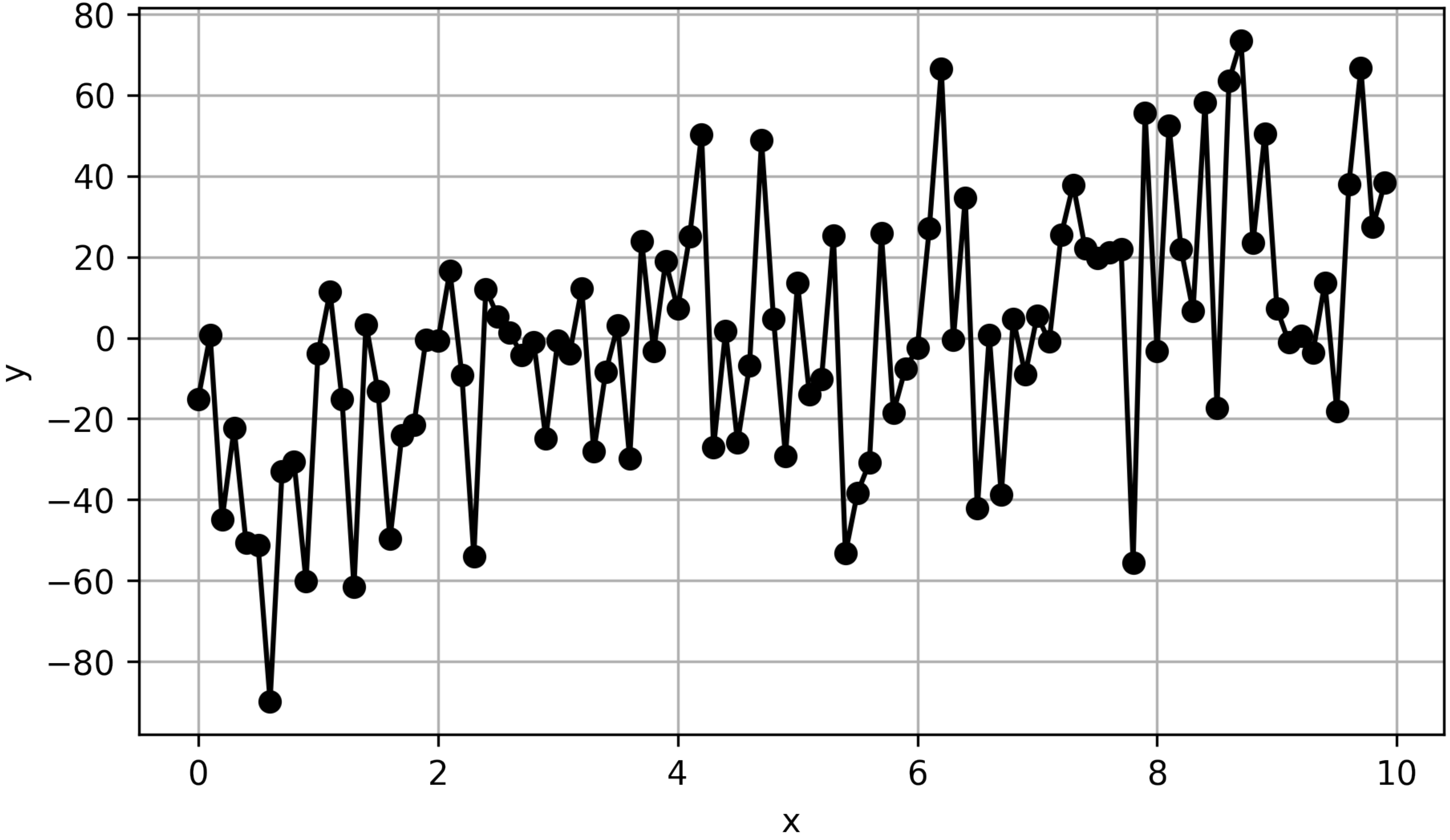
Loading `scipy` modules

```
from scipy import stats
from scipy import interpolate
```

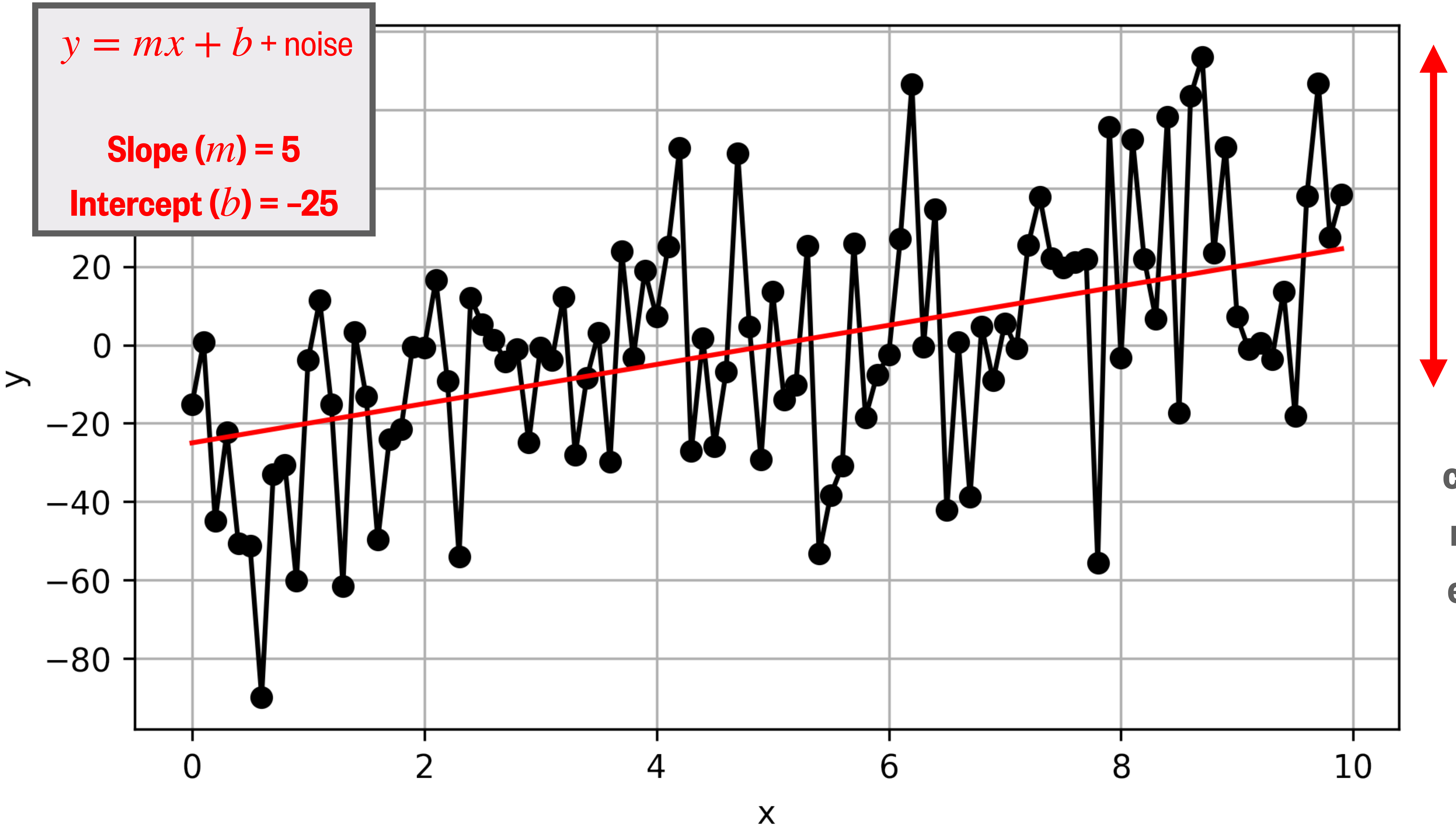
Loading `scipy` modules

```
from scipy import stats, interpolate
```

Does this noisy data have a trend?



This data has a linear trend and random noise

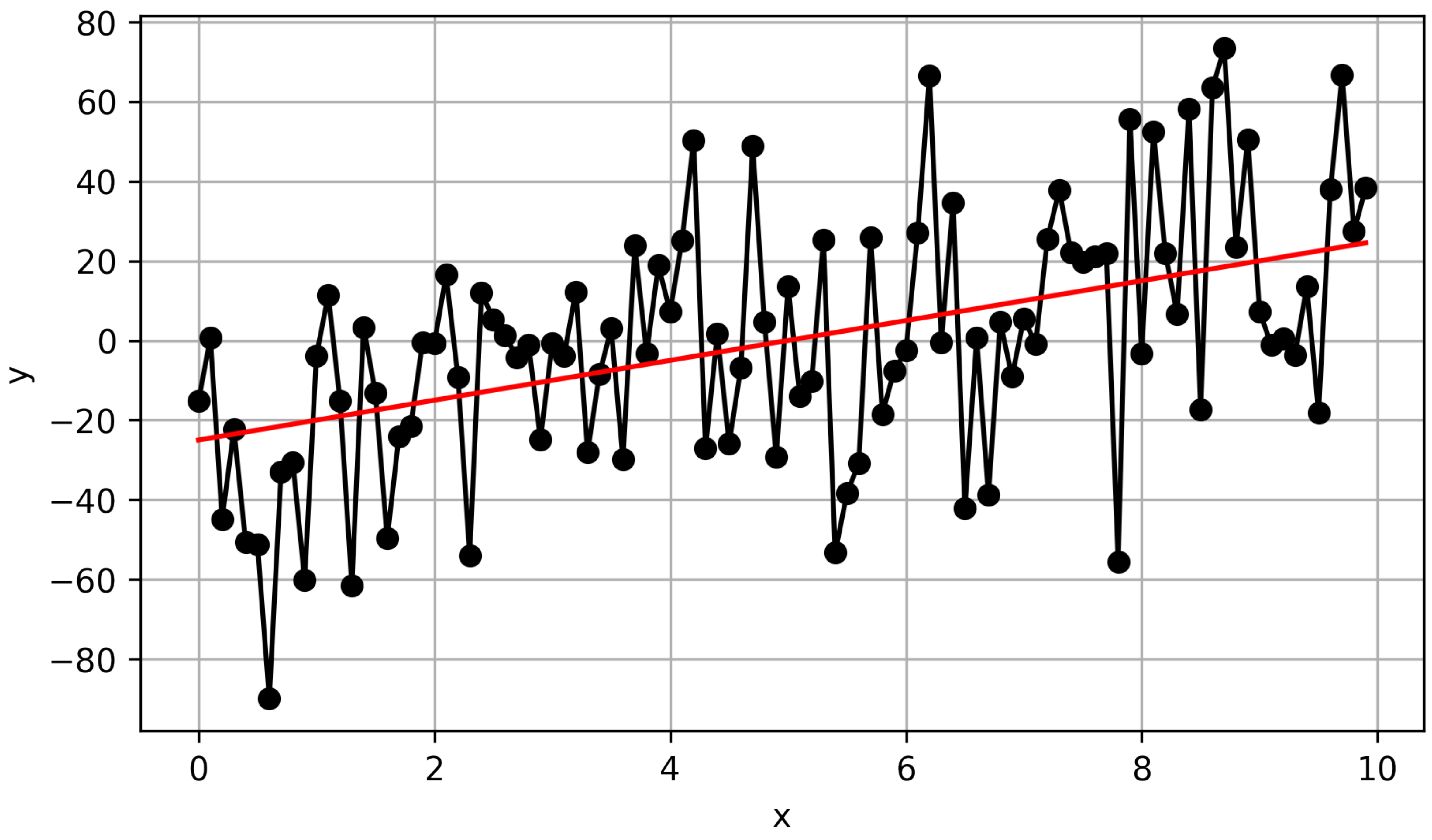


Random variations (“noise”)

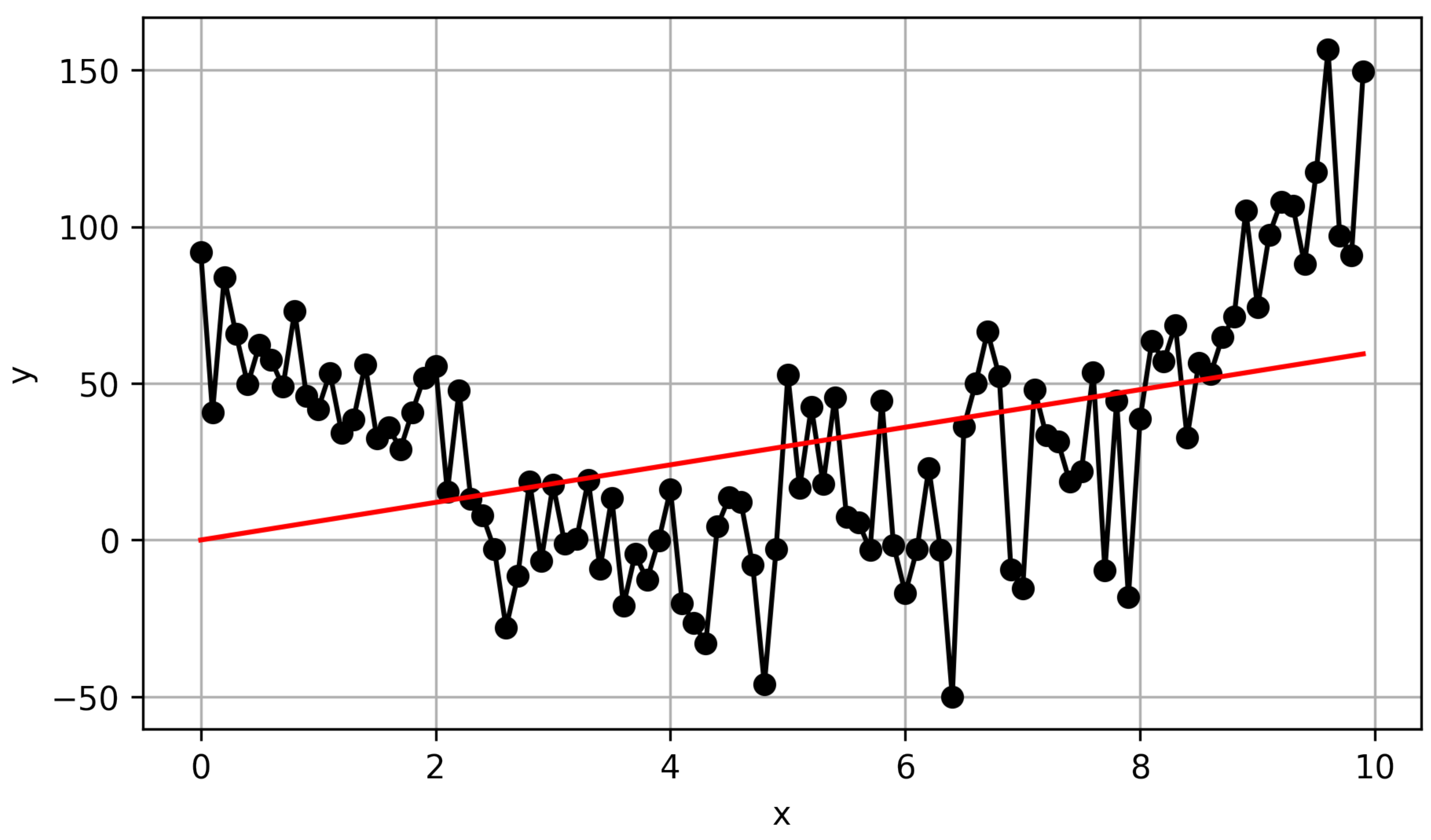
“Noise” can come from measurement error or actual processes of interest

Regression relates one (or more) predictor variables to a dependent variable, and it requires assuming a “model”

Here, a **linear model** seems appropriate

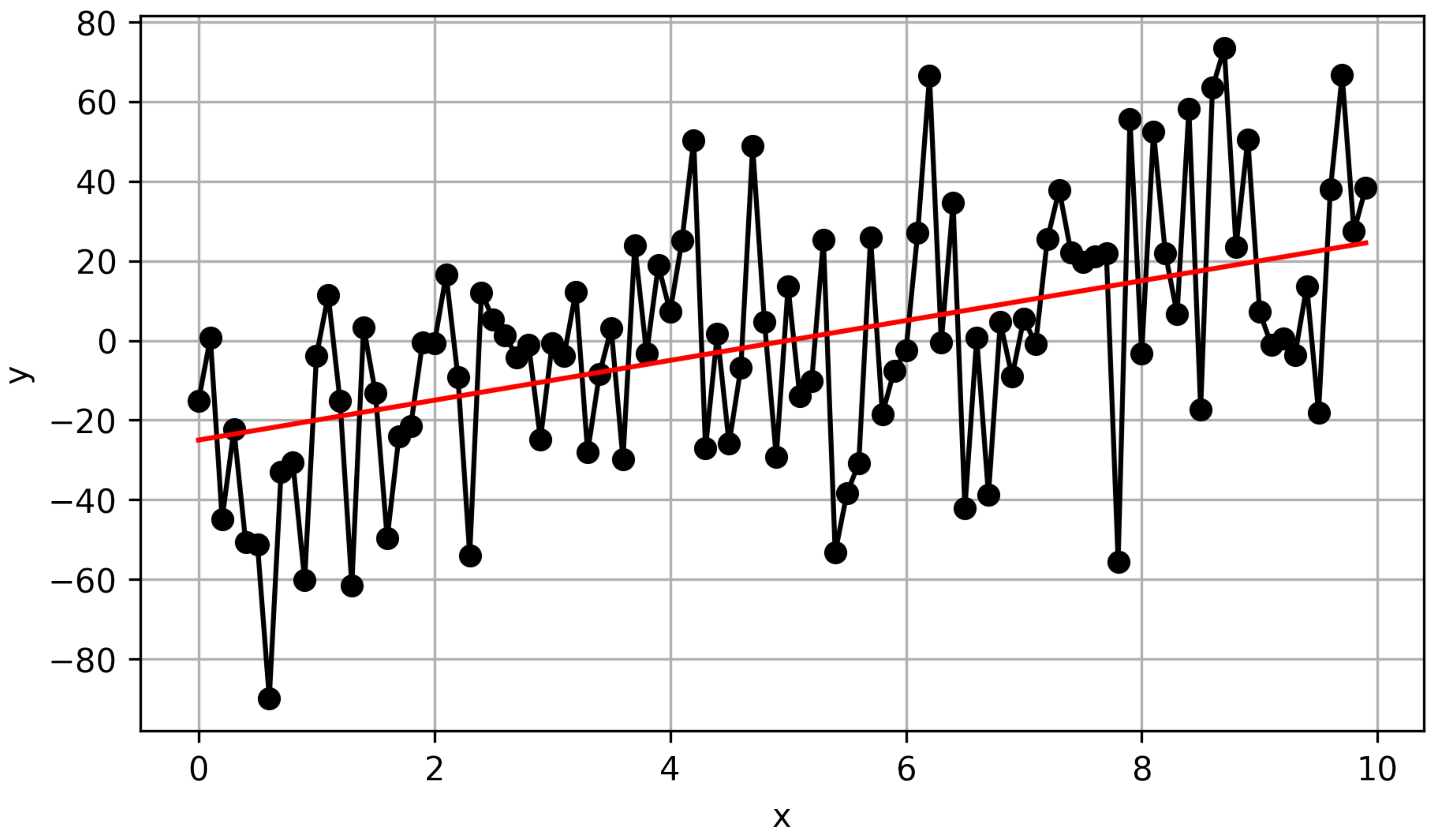


Here, a linear model is inappropriate
(a **quadratic model** would be better)



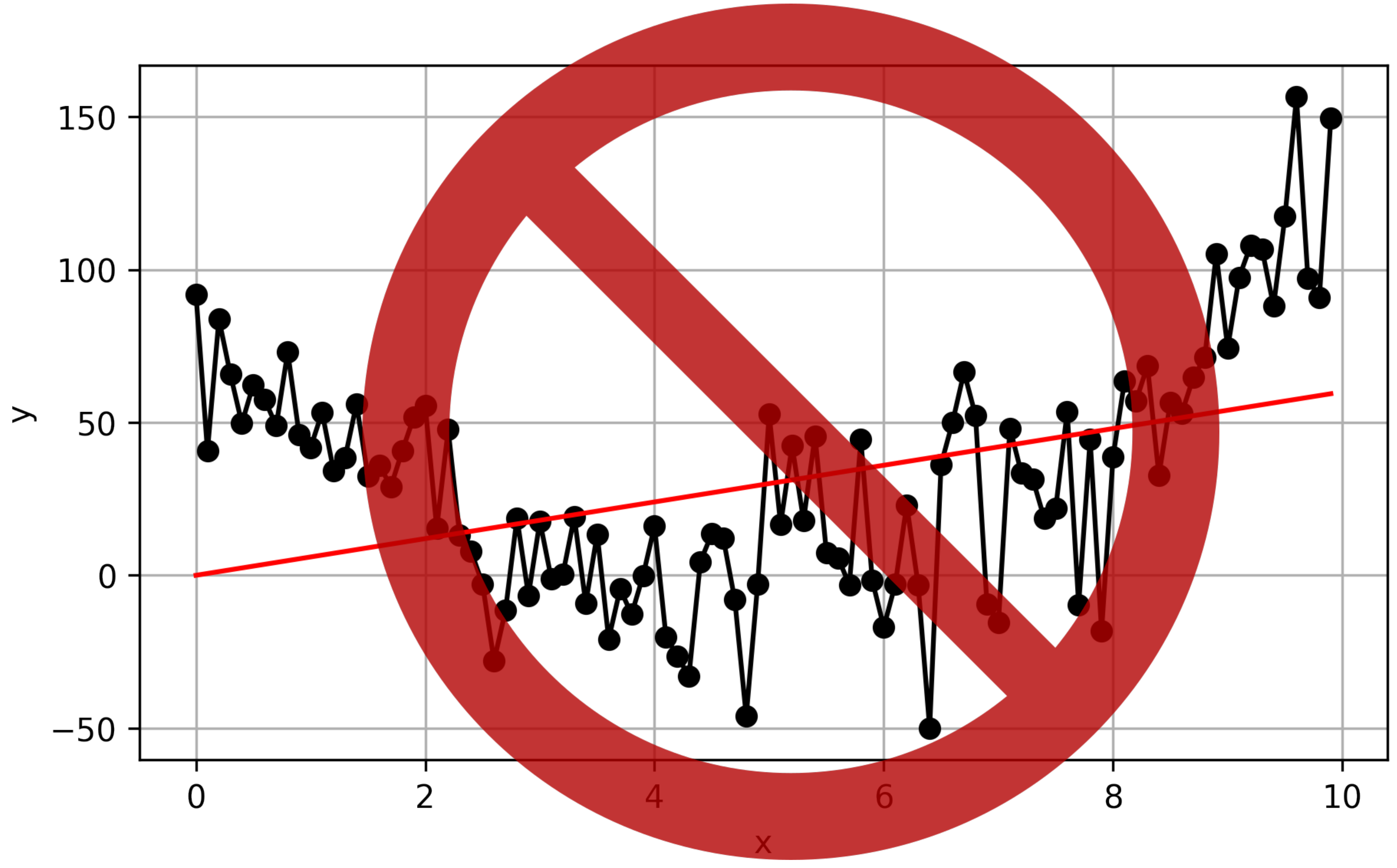
Regression relates one (or more) predictor variables to a dependent variable, and it requires assuming a "model"

Here, a **linear model** seems appropriate

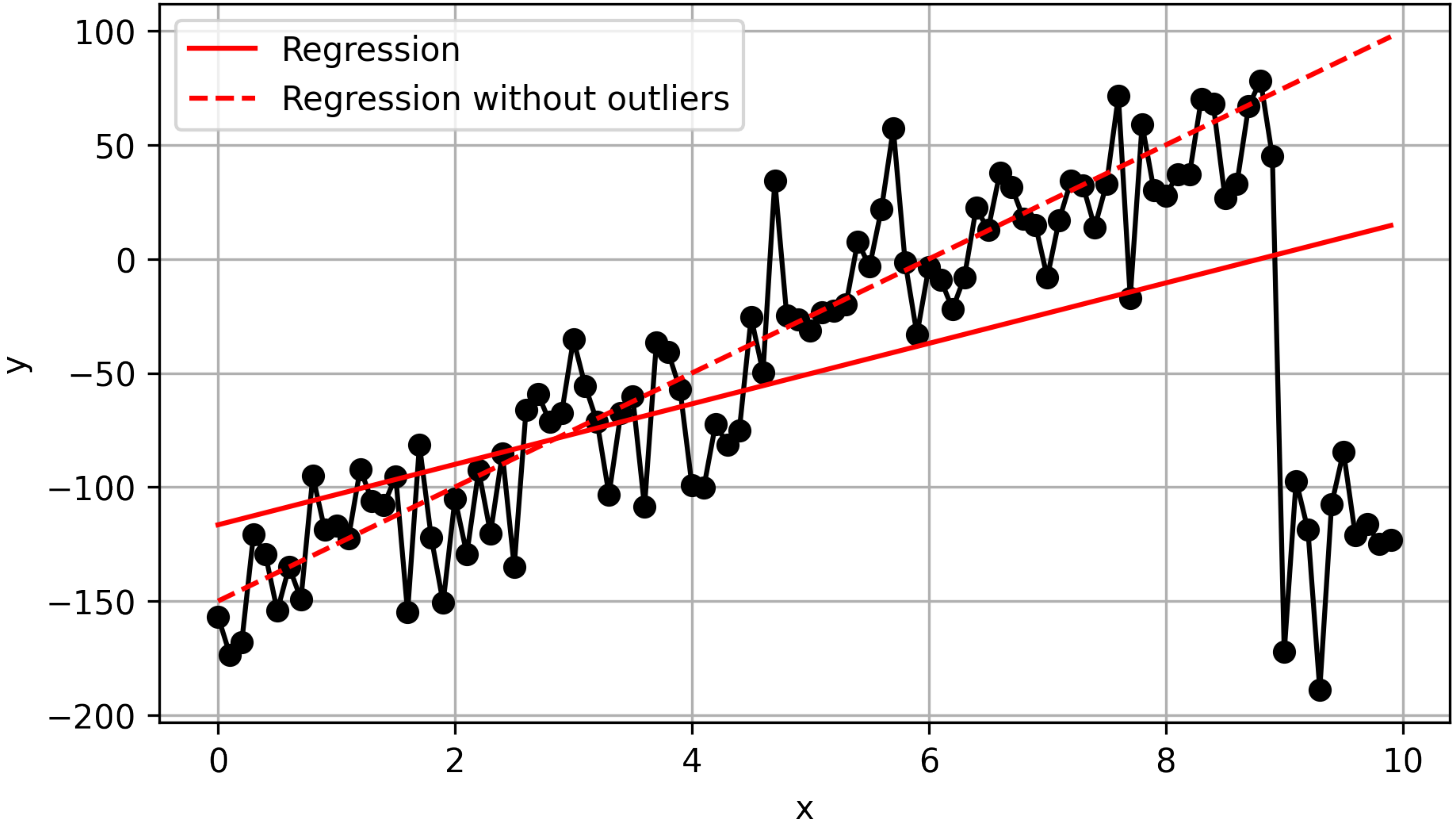


Here, a linear model is inappropriate

(a **quadratic model** would be better)



Regression works by minimizing the square of the errors, so it's sensitive to outliers



The regression line gets "pulled" towards outliers

Linear regression in SciPy

Correlation coefficient (*r*)

Standard error

Two-sided *p*-value

```
slope, intercept, rvalue, pvalue, stderr
  = stats.linregress(x,y)
```

1-D NumPy arrays of the same length

If you don't need a function output, you can give it to a "throwaway" underscore

These output variables will be ignored

```
slope, intercept, _, _, stderr
  = stats.linregress(x,y)
```

Correlation coefficient (*r* value) for a linear regression

Important: the r value is not typically used!

Instead, we use r^2, which represents the **goodness of fit**,
the proportion of variance (σ^2) in the dependent variable (y) that can be predicted
from the independent variable (x) by the linear regression model.

- $r^2 = 1.0$ means 100% of variance is explained by the regression (i.e. the data is a straight line)
- $r^2 = 0.5$ means 50% of variance is explained by the regression
- $r^2 = 0.0$ means 0% of variance is explained by the regression (a very poor fit)

p value for a linear regression

The p-value represents the probability of obtaining the given regression slope if the null hypothesis were correct (i.e. the actual slope was zero).

- If $p < 0.10$, the null hypothesis of no slope can be rejected at the 90% confidence level.
- If $p < 0.05$, the null hypothesis of no slope can be rejected at the 95% confidence level.
- If $p < 0.01$, the null hypothesis of no slope can be rejected at the 99% confidence level.

Caution: p-values are frequently misused in science.
Small p-values can be found even when the chosen model is inappropriate.

Linear regression results

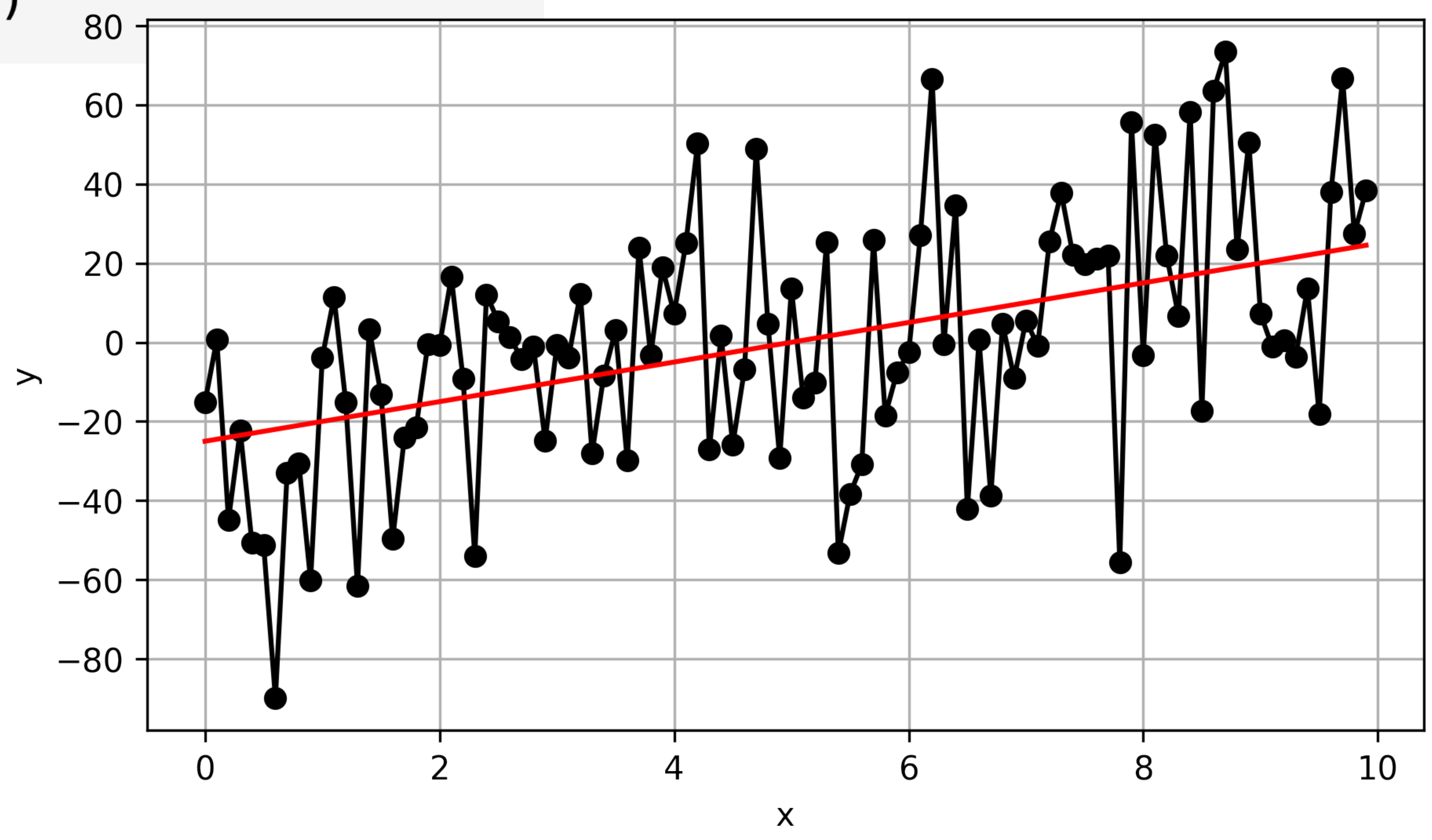
```python
slope, intercept, rvalue, pvalue, stderr = stats.linregress(x,y)

print('The slope is',round(slope,2))
print('The y-intercept is',round(intercept,2))
print('The r-value is',round(rvalue,2))
print('The p-value is',pvalue)
print('The standard error is',round(stderr,2))
```

```
The slope is 5.77
The y-intercept is -28.7
The r-value is 0.53
The p-value is 1.779535447617004e-08
The standard error is 0.94
```

$y = mx + b$ + noise

Slope (m) = 5

Intercept (b) = –25



What if your x-values are `datetime` objects?

```python
import matplotlib.dates as mdates

t = np.array([datetime(2020,1,1),
              datetime(2020,2,1),
              datetime(2020,3,1)])

t_as_numbers = mdates.date2num(t)

print(t_as_numbers)
```

```
[737425. 737456. 737485.]
```

`linregress()` can't handle an array of `datetime` objects as x-values 😡

This converts `datetime` objects to numbers representing "days since 0001-01-01 plus one", which `linregress()` can handle 😌

What we'll cover in this lesson

1. `SciPy`: linear regression

2. `SciPy`: 1-D and 2-D interpolation/regridding

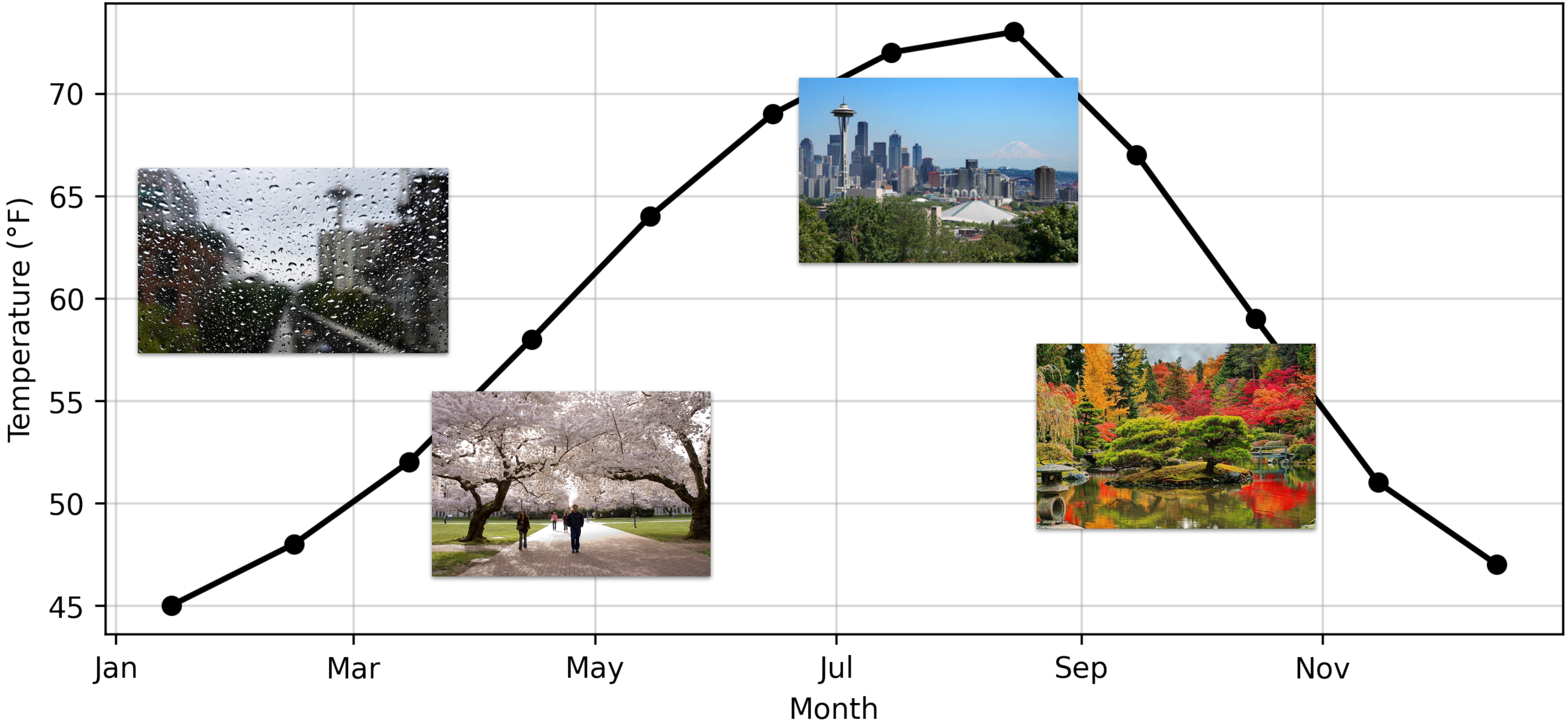
What is interpolation?

Definition: Interpolation allows you to estimate unknown values of a variable based on known values of the variable.

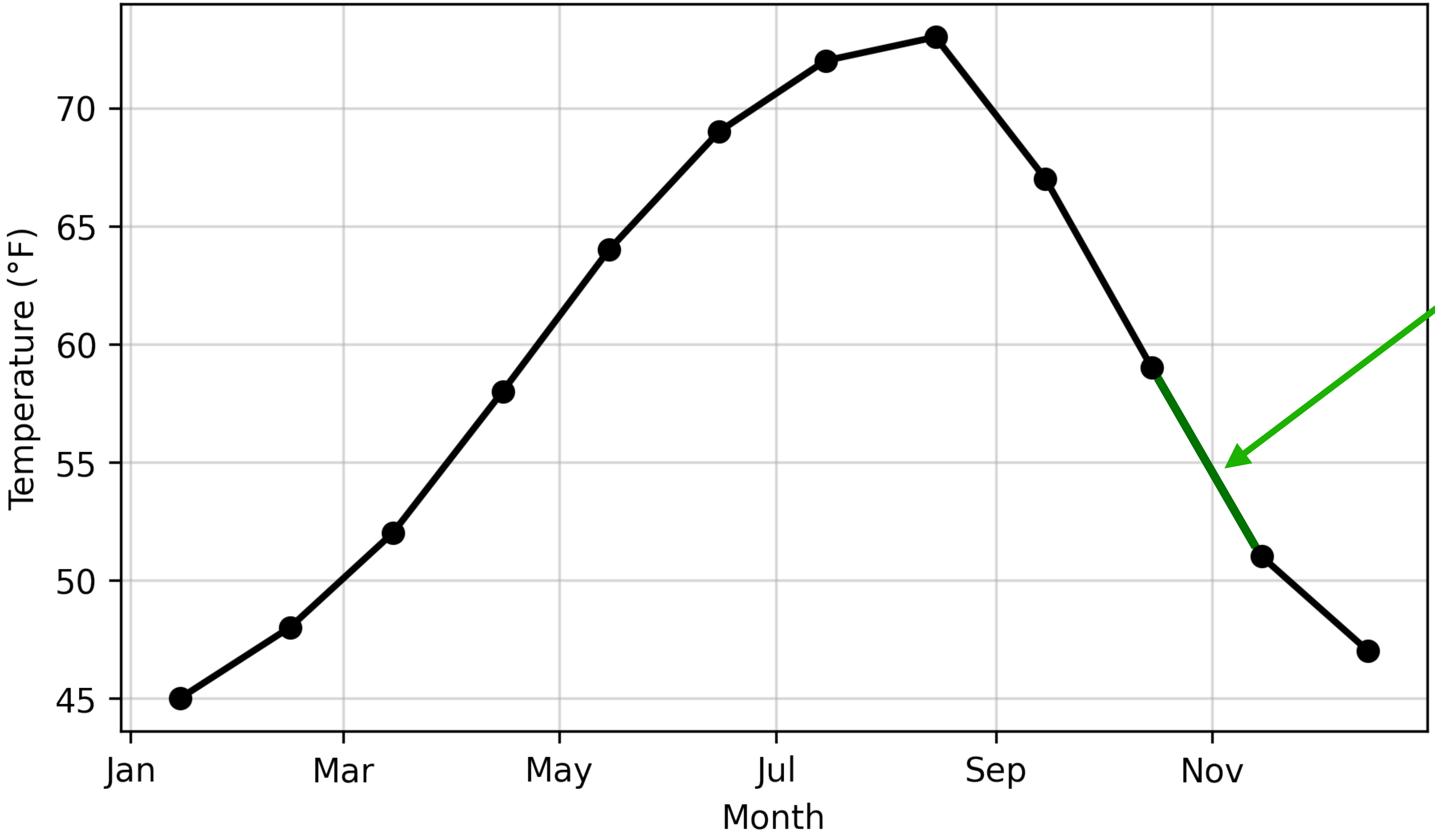
Values of a variable can be unknown because...

- They weren't measured frequently enough in time or space.
- They weren't measured at the right times or locations or on the right grid.
- The data are missing, perhaps because an instrument temporarily stopped measuring.

Example: climatological high temperatures in Seattle

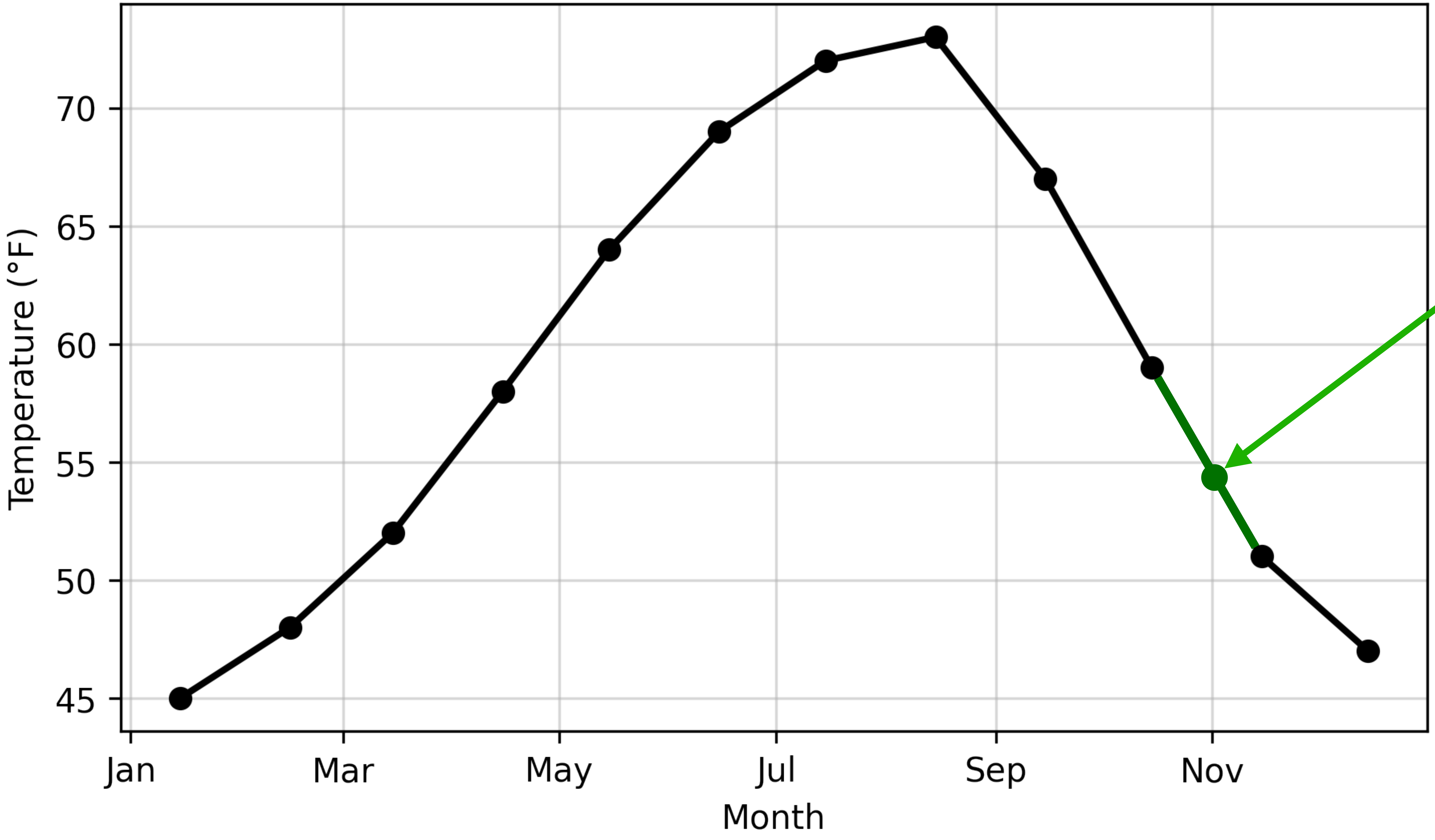


Example: climatological high temperatures in Seattle



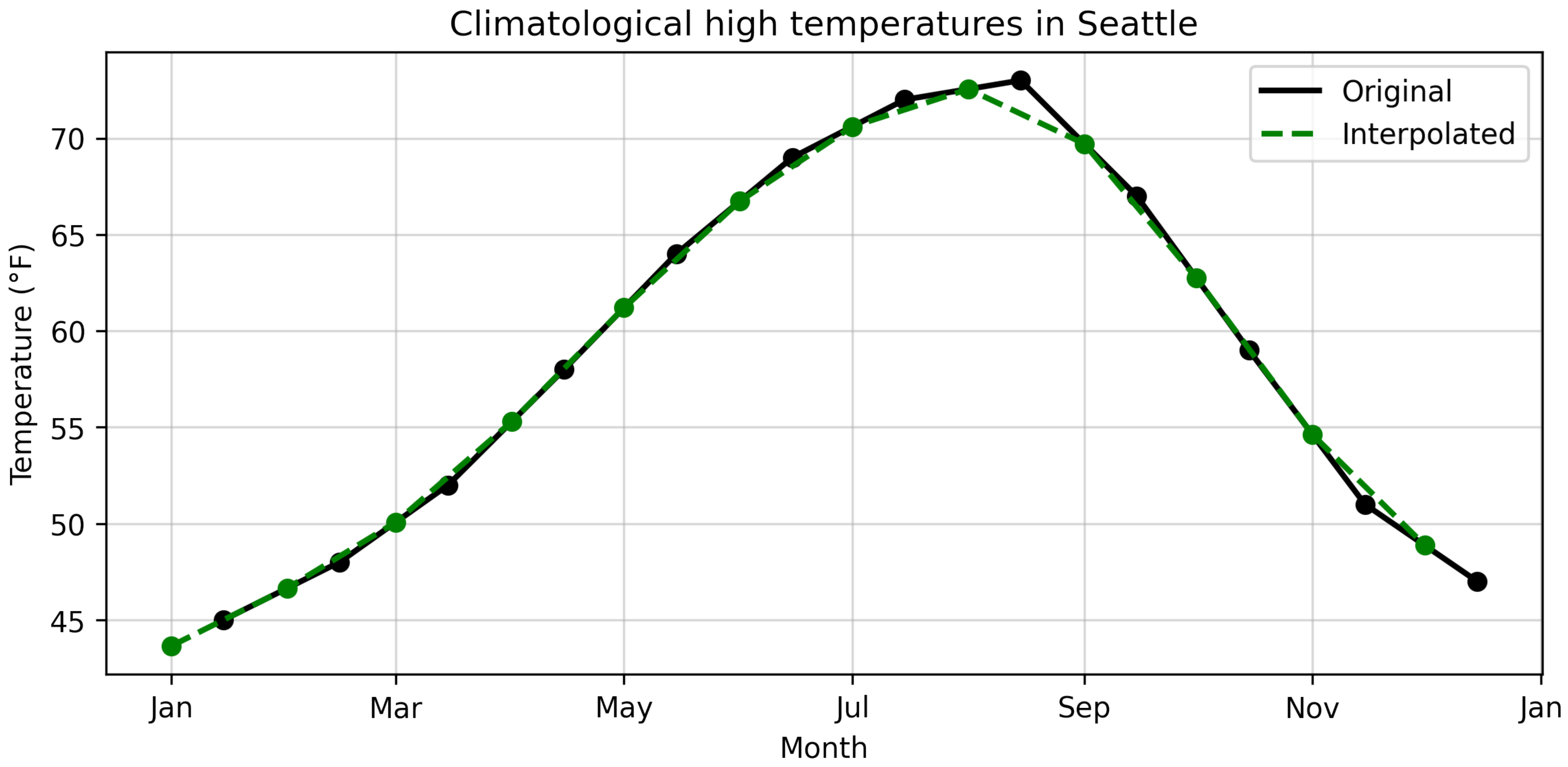
This is linear interpolation!

What if we wanted the climatological temperature on November 1?

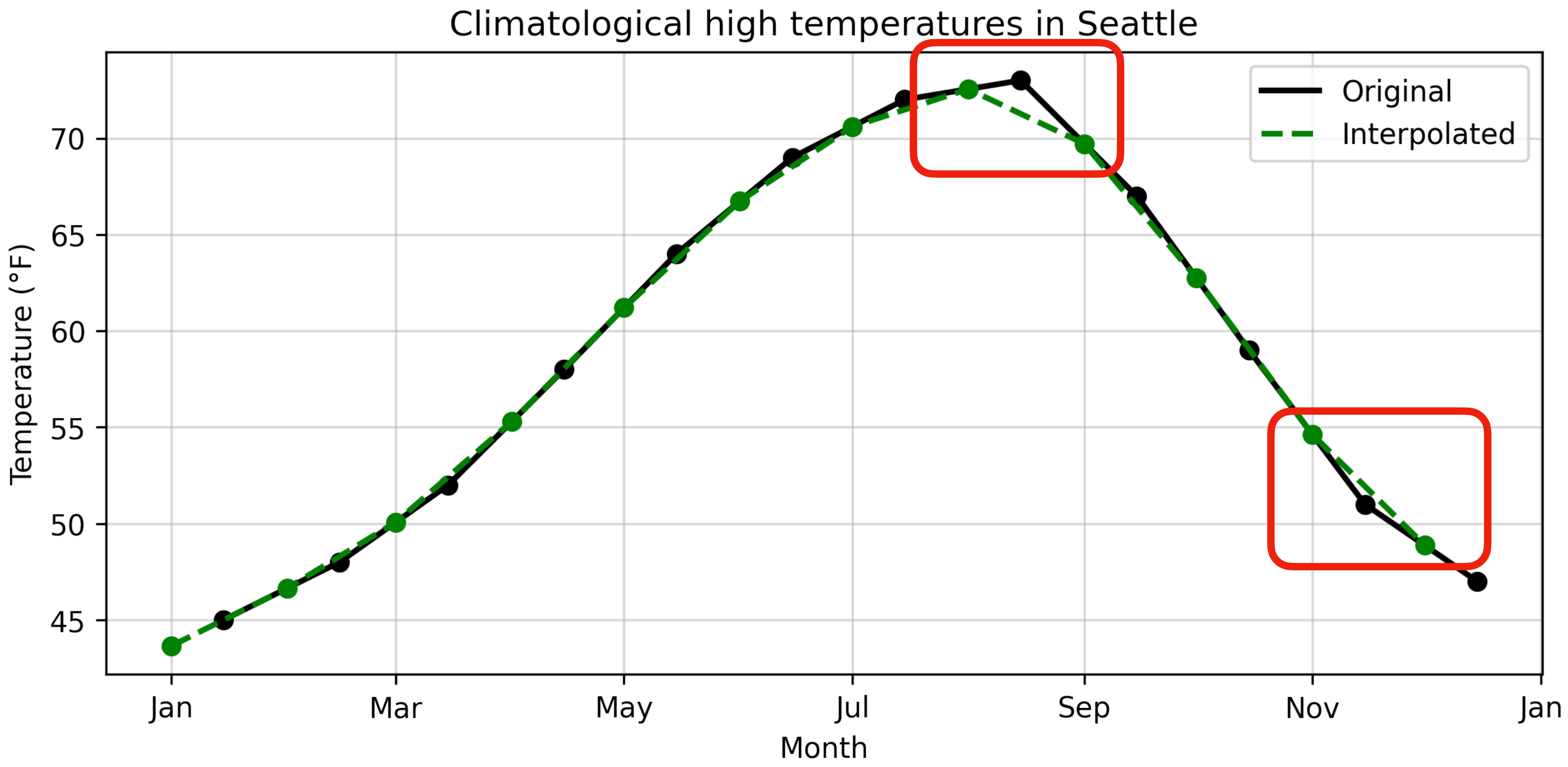


We'd estimate it using the straight line between the Oct. 15 and Nov. 15 points!

Interpolated (“regridded”) from 15th of each month to 1st of each month…



Interpolation and regridding can come with a loss in accuracy



1-D interpolation in SciPy is a two-step process

```
interp_func = interpolate.interp1d(x,y,
                                   kind='linear',
                                   bounds_error=False,
                                   fill_value=np.NaN)

y_new = interp_func(x_new)
```

API reference: SciPy interp1d()

1-D interpolation in SciPy is a two-step process

This is a function, but you can choose its name

Original x- and y-values (1-D arrays)

```
interp_func = interpolate.interp1d(x,y,
                                   kind='linear',
                                   bounds_error=False,
                                   fill_value=np.NaN)
```

Other options: `'nearest'`, `'quadratic'`, `'cubic'`, `etc.`

If points in `x_new` are outside `x`, set to `False` to avoid an error

Other option: `'extrapolate'`

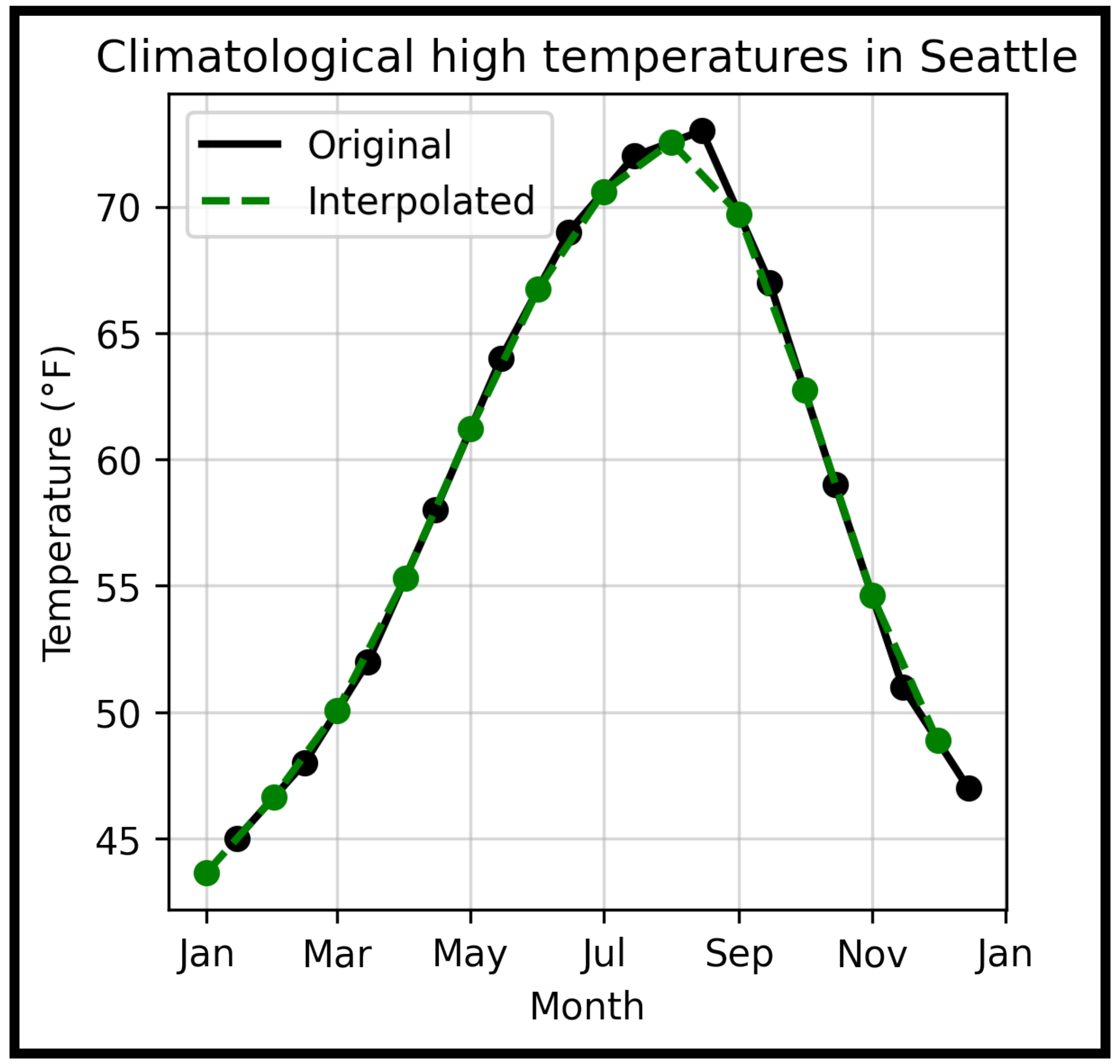
```
y_new = interp_func(x_new)
```

Interpolated y-values (1-D array)

Set of x-values to interpolate to (1-D array)

Interpolating to/from x-values that are `datetime` arrays

Example scenario:




```
import matplotlib.dates as mdates

interp_func =
    interpolate.interp1d(mdates.date2num(x),y)

y_new = interp_func(mdates.date2num(x_new))
```

Converts `datetime` objects into numbers of days

Types of interpolation

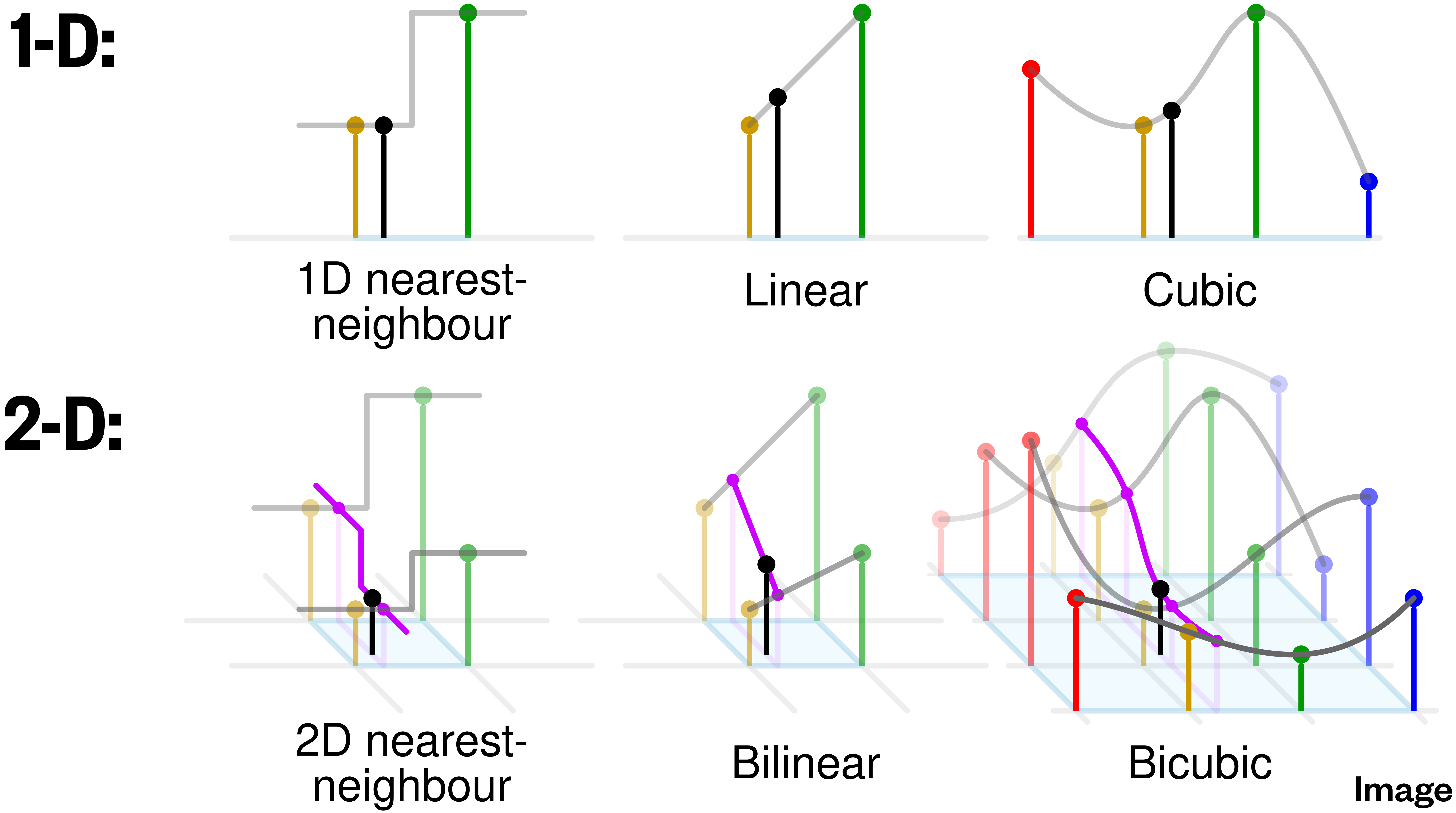


Image credit: C.M.G. Lee

2-D interpolation (a.k.a. 2-D regridding)

You have:

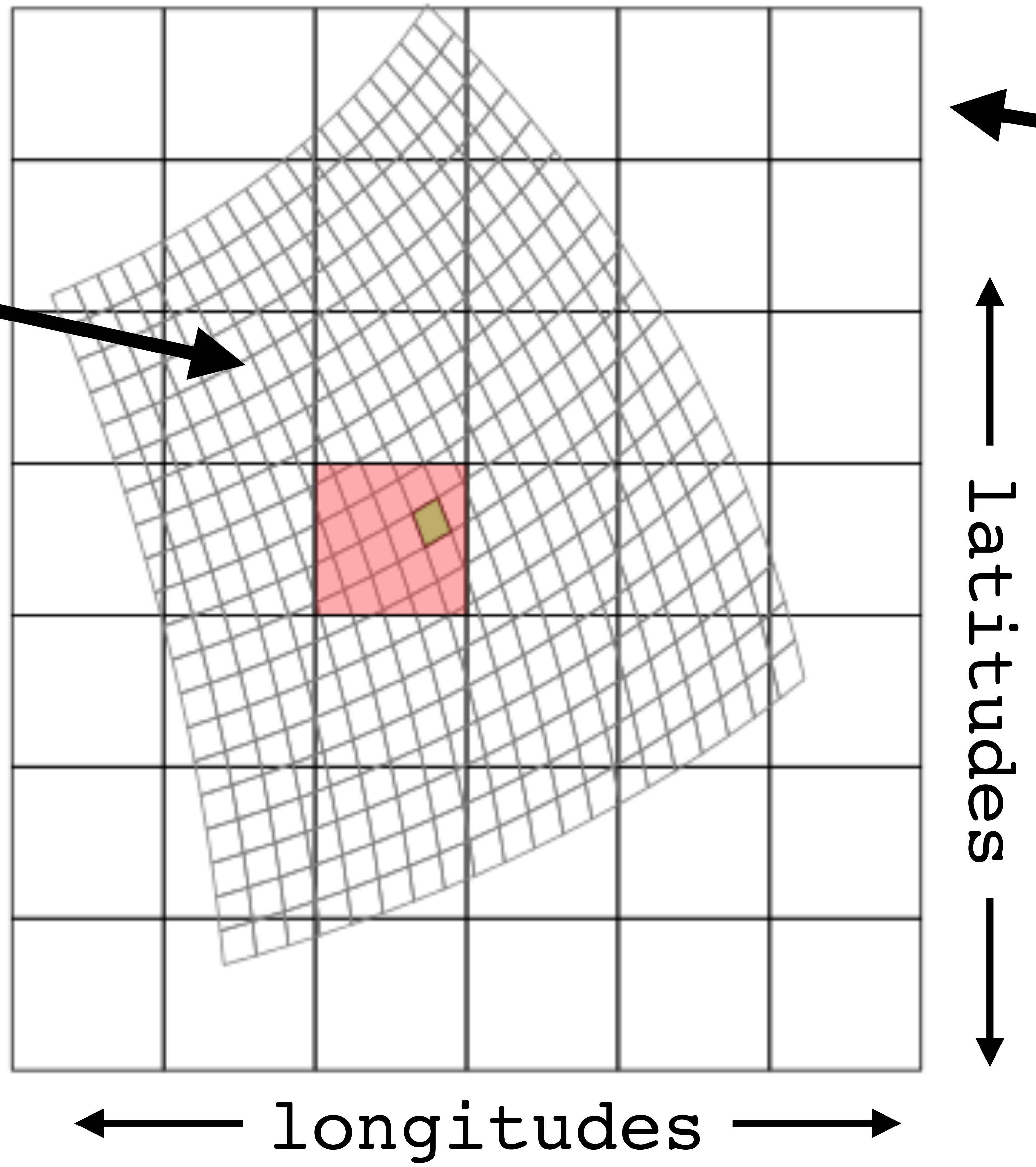
An irregular grid

(`lat` and `lon`

or other coordinates are usually

2-D arrays)

```
plt.pcolormesh()
plt.contourf()
```
xarray's `.sel()`



You want:

A regular grid

(`lat` and `lon`

can be represented

as 1-D coordinates)

```
plt.pcolormesh()
plt.contourf()
```
xarray's `.sel()`

For more information on regridding, see [Climate Data Guide's "Regridding Overview"]

Image credit: [Lu et al. (2018)]

2-D interpolation in SciPy is a three-step process

```
x_coord = np.linspace(start,end,num_x_points)
y_coord = np.linspace(start,end,num_y_points)


x_grid,y_grid = np.meshgrid(x_coord,y_coord)


z_gridded = interpolate.griddata((x_flat,y_flat),
                                 z_flat,
                                 (x_grid,y_grid),
                                 method='linear')
```

API references: NumPy meshgrid() and SciPy griddata()

2-D interpolation in SciPy is a three-step process

Regularly-spaced 1-D coordinate arrays

These values determine your new grid domain

```
x_coord = np.linspace(start,end,num_x_points)
y_coord = np.linspace(start,end,num_y_points)
```

"Meshed" (stacked) 2-D versions of the 1-D coordinate arrays – compatible with `plt.pcolormesh()`, `plt.contourf()`

```
x_grid,y_grid = np.meshgrid(x_coord,y_coord)
```

Steps #1 and #2 are optional if you already have a new x- and y-grid

2-D array of the z-parameter values, interpolated to the new x- and y-coordinates – compatible with `plt.pcolormesh()`, `plt.contourf()`

```
z_gridded = interpolate.griddata((x_flat,y_flat),
                                 z_flat,
                                 (x_grid,y_grid),
                                 method='linear')
```

1-D arrays of the original irregular x- and y-locations and z-parameter data – incompatible with `plt.pcolormesh()`, `plt.contourf()`

Note: if the original arrays are 2-D, you have to flatten them first, e.g.:

```
z_flat = z_original.flatten()
```

Other interpolation methods: `'nearest'`, `'cubic'`