CIS 1100

Information Representation
& Data Visualization
(Lecture One)

Python
Spring 2025
University of Pennsylvania

Updates and Reminders

- Apply to be a TA by 11:59pm on April 18th
  - link is on Ed
  - no late days accepted :)
- HW8 Released on Course Website
  - Due Wednesday, April 16th
  - HW8 and HW9 Part 1 are the only assignments remaining on which you can use a late token.
- Midterm 2 grades out extremely soon

Questions?

Today: Data Visualization

A whirlwind tour...

1. What is the stuff that makes up a graphic?
2. When is the graphic useful and truthful?
3. How can we make useful and truthful graphics with Pandas?

The Challenge: Information Representation

Basically, symbolism! What can it mean when I use **X** to represent something?

We'll talk about this in terms of:

- data types
- graphics & graphical markers (visual symbols)

> "I will tell you a terrible secret: language is punishment. Language must encompass all things and in it all things must again transpire according to guilt and the degree of guilt." — *Malina* by Ingebord Bachmann

> "And meaning, after all, is a kind of luck—some things just shine with it, and no one knows why." — *Priestdaddy* by Patricia Lockwood

Representation: Types

An `int` is a data type for integral (whole) numbers.

The typical interpretation of an `int` is a **quantity**: I have `10` eggs in my refrigerator, or there are `103` students in this class.

(L11) Write as many things as you can think of that an `int` can be used to represent. (Feel free to brainstorm with a partner.)

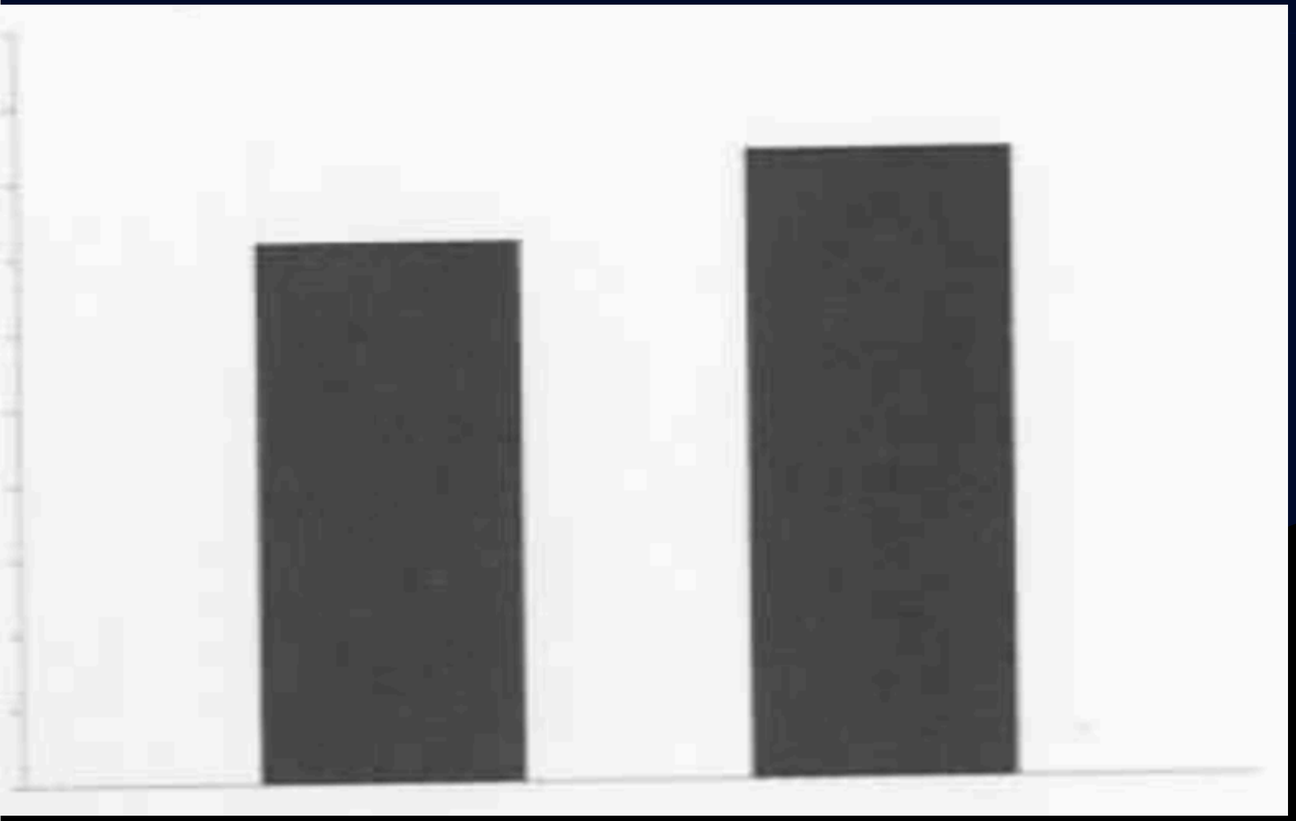
What Did You Come Up With?

`2158983500` (which can have a few meanings...)

`1100`

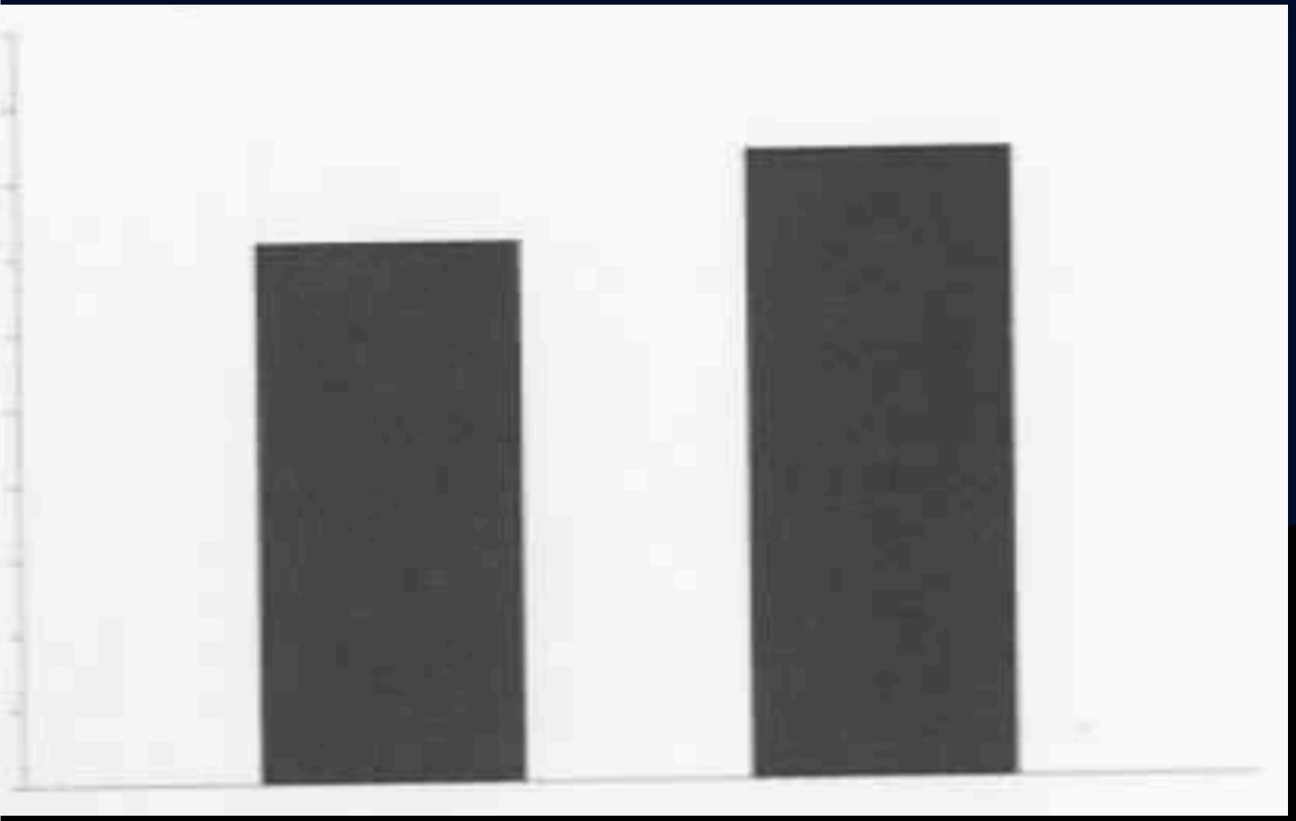
`-1`

Exercise (C12)



Briefly: *When you look at the two dark rectangles below, what do you notice and what meanings come to mind?*

Bars



The following can all convey meaning:

- heights, and differences between them
- weight (width) and contrast from the background
- position:
  - along the x-axis, separation
  - along the y-axis, alignment at the bottom

Marks & Channels

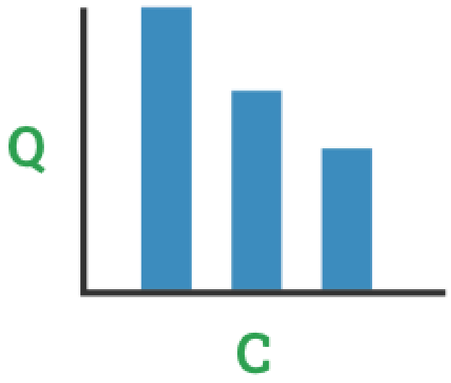
Marks are the geometric shapes that make up a graphic

- the stuff you draw with PennDraw commands like `rectangle` or `line` or `point`

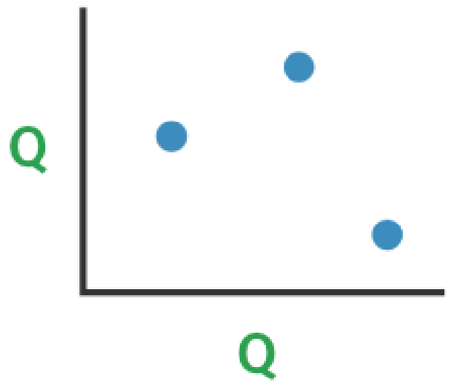
Channels are the ways that we modify the marks, including:

- positions, size, area, or tilt/angle (i.e. parameters of the `pd` calls themselves)
- color and thickness, which are changed by separate calls to `set_pen_color` and `set_pen_radius`

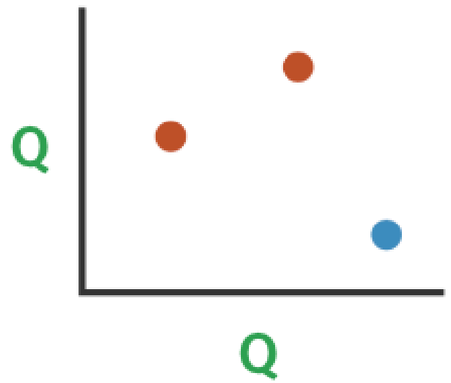
A Blatantly Plagiarized Example Courtesy of COMS 4995 at Columbia



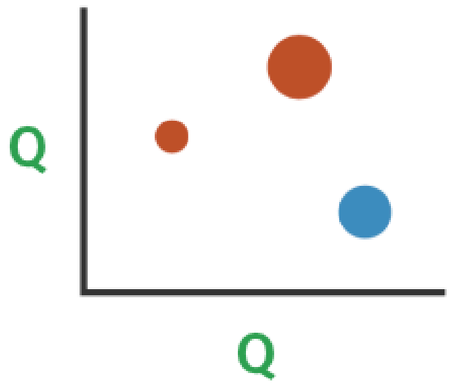
Mark: **line**



Mark: **point**

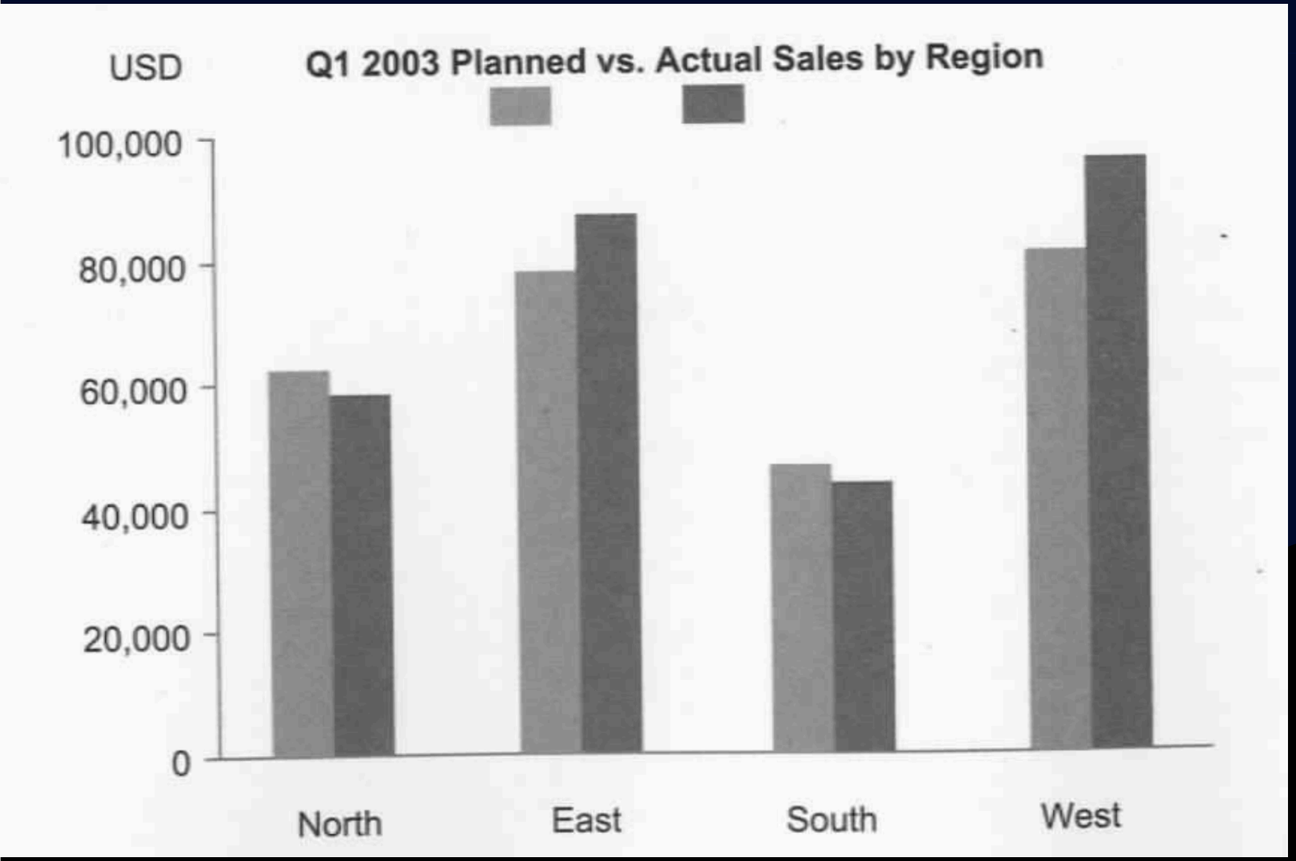


Mark: **point**
Channels:
Color: **C**



Mark: **point**
Channels:
Color: **C**
Size: **Q**

Exercise



In one or two words...

(S7) What does the height channel of a bar encode (represent)?

(S8) What does the width channel of a bar encode?

(S9) What does the x-position channel of a bar encode?

(S10) What does the color channel of a bar encode?

What is the stuff that makes up a graphic?

- We make **marks** on a graphic to indicate individual pieces of information (data!)
- The ways in which we choose to make the marks are the **channels.**
- Ideally, we modify the marks depending on the information they are used to represent
  - The differences between the properties (position, size, color, etc.) of two shapes should be used to convey differences between the underlying data points.

CIS 1100

When is a Graphic Useful and Truthful?

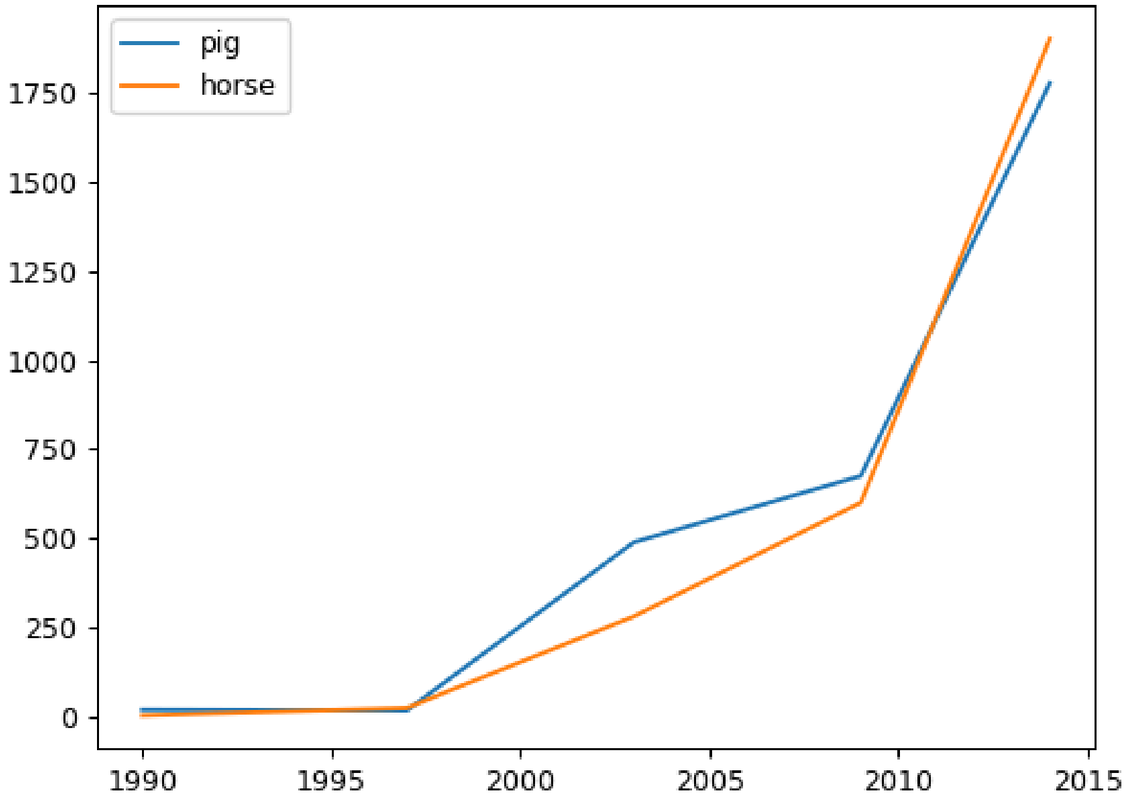
Python
Spring 2025
University of Pennsylvania

Making Good Choices

All that's needed for a graphic are marks & channels, but how do you pick them so that:

- the graphic reveals something that inspection of the underlying data would not?
- the graphic is quick to read and easy to interpret?
- the graphic is hard to misinterpret?
- the data are accurately & correctly represented?

Making Good Choices



The moral of the story is that the built-in plotting features for Pandas allow you to pretty quickly make reasonable choices.

```
df = pd.DataFrame({
   'pig': [20, 18, 489, 675, 1776],
   'horse': [4, 25, 281, 600, 1900]
   }, index=[1990, 1997, 2003, 2009, 2014])
lines = df.plot.line()
```

The defaults are often punishingly **boring**, but they usually acheive the aims listed on the previous slide.

More Fun... Bad Examples

Let's take a look at a few examples and critique them based on our goals:

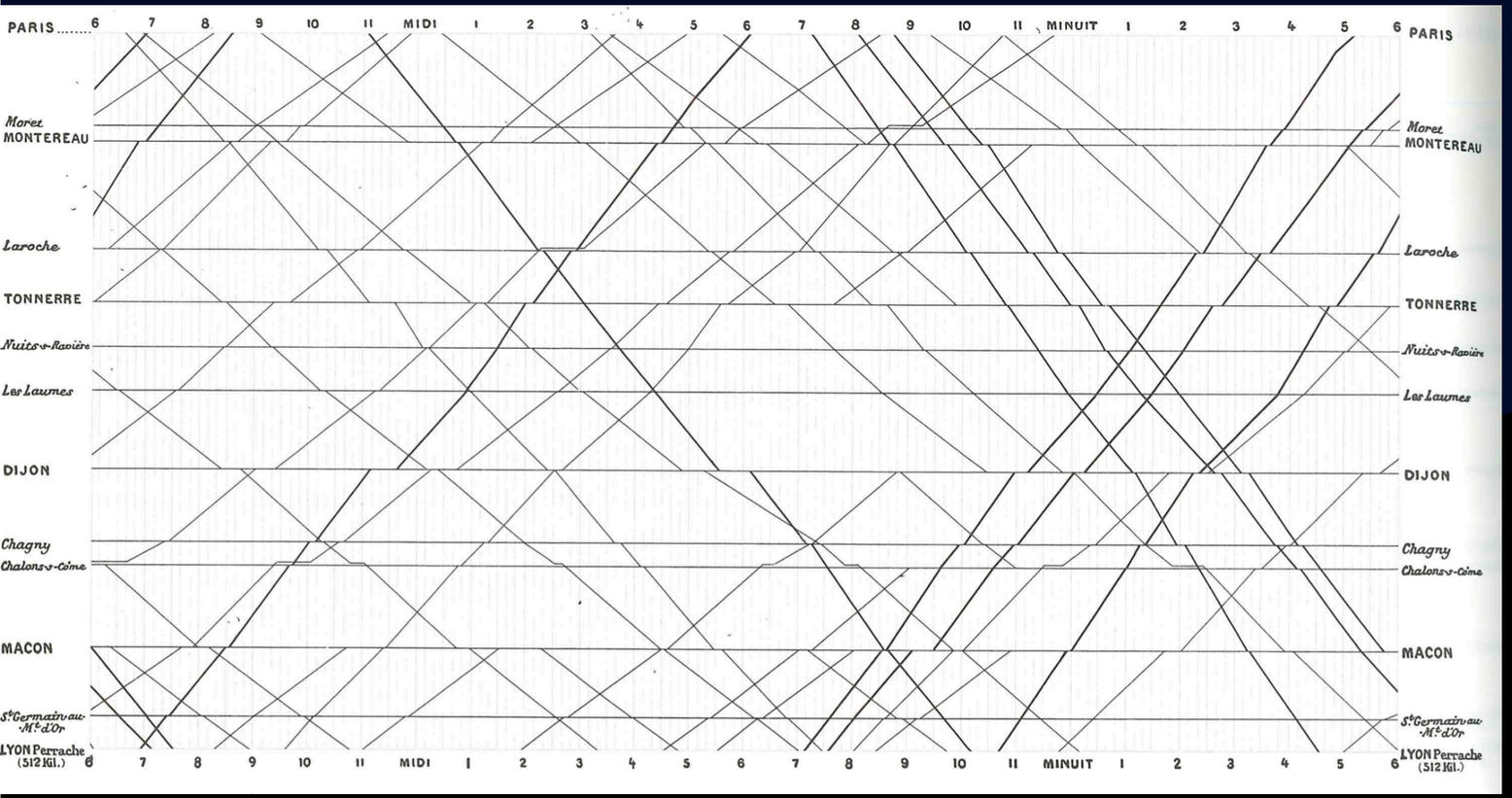
- (A) the graphic is not much more useful than just reading the underlying data "raw"
- (B) the graphic is hard to read or get *any* sense of
- (C) the graphic is easy to misinterpret
- (D) the data are presented in a misleading or inaccurate way

Pravda (M1)

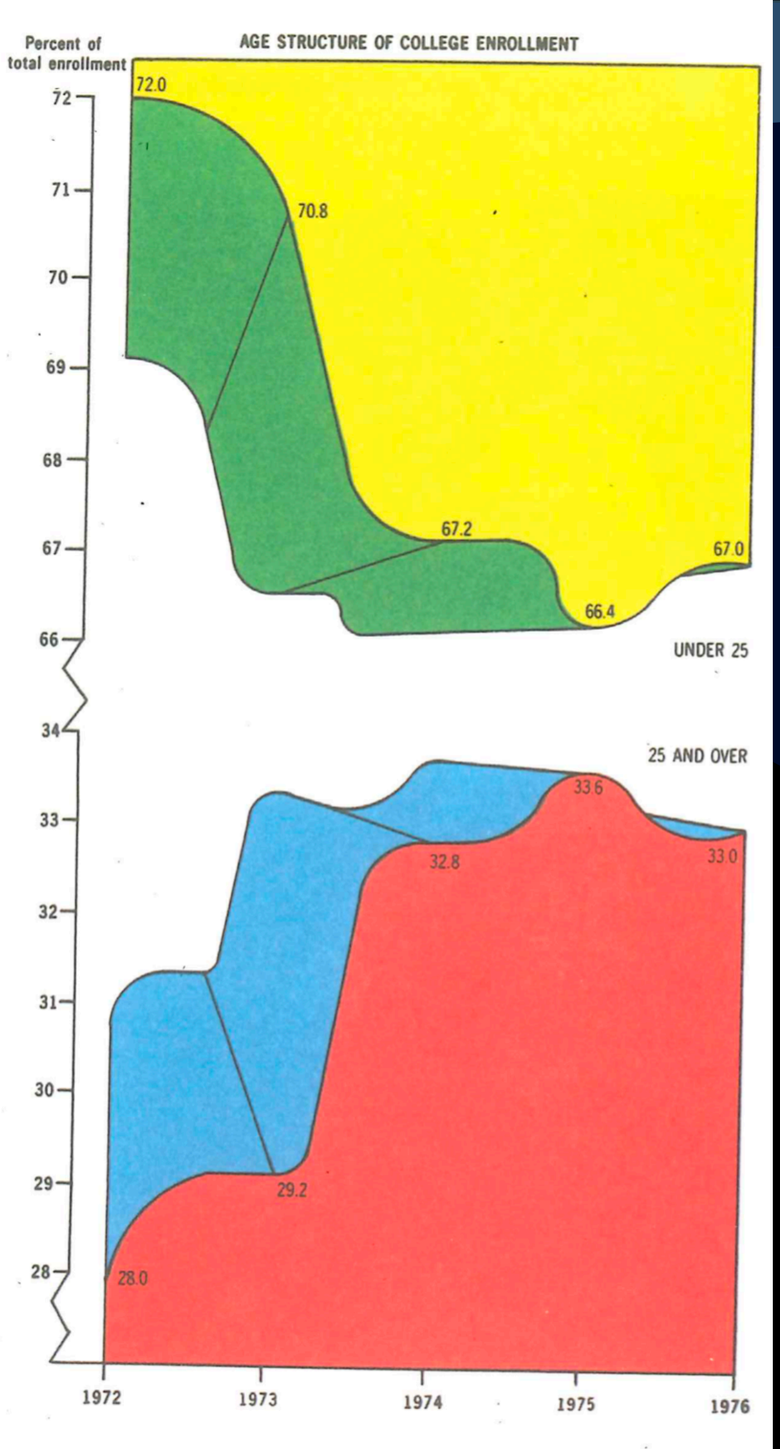
Рост продукции промышленности (1922 г. = 1).

- (A) the graphic is not much more useful than just reading the underlying data "raw"
- (B) the graphic is hard to read or get *any* sense of
- (C) the graphic is easy to misinterpret
- (D) the data are presented in a misleading or inaccurate way

Train Times (M2)



- (A) the graphic is not much more useful than just reading the underlying data "raw"
- (B) the graphic is hard to read or get *any* sense of
- (C) the graphic is easy to misinterpret
- (D) the data are presented in a misleading or inaccurate way

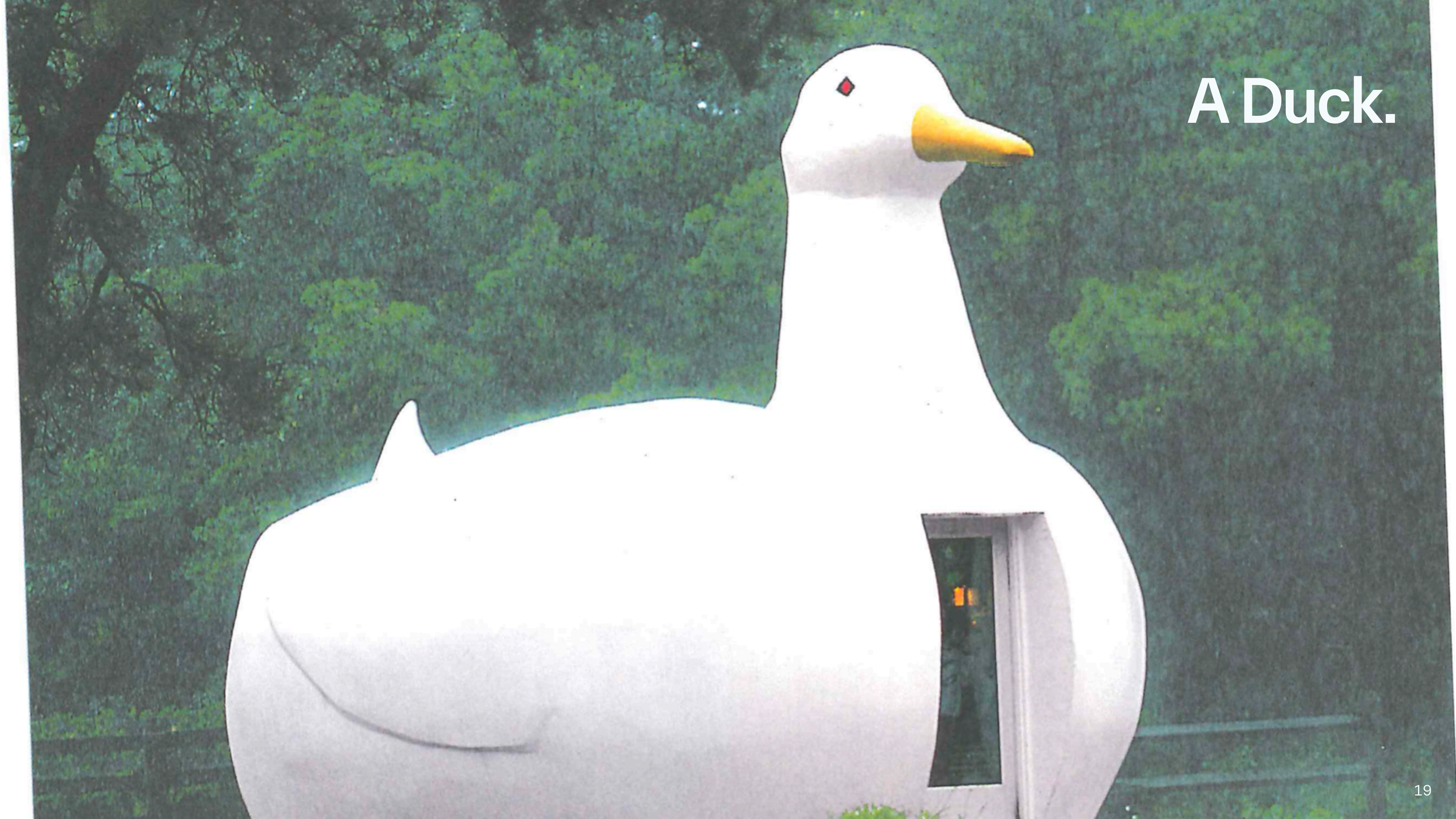


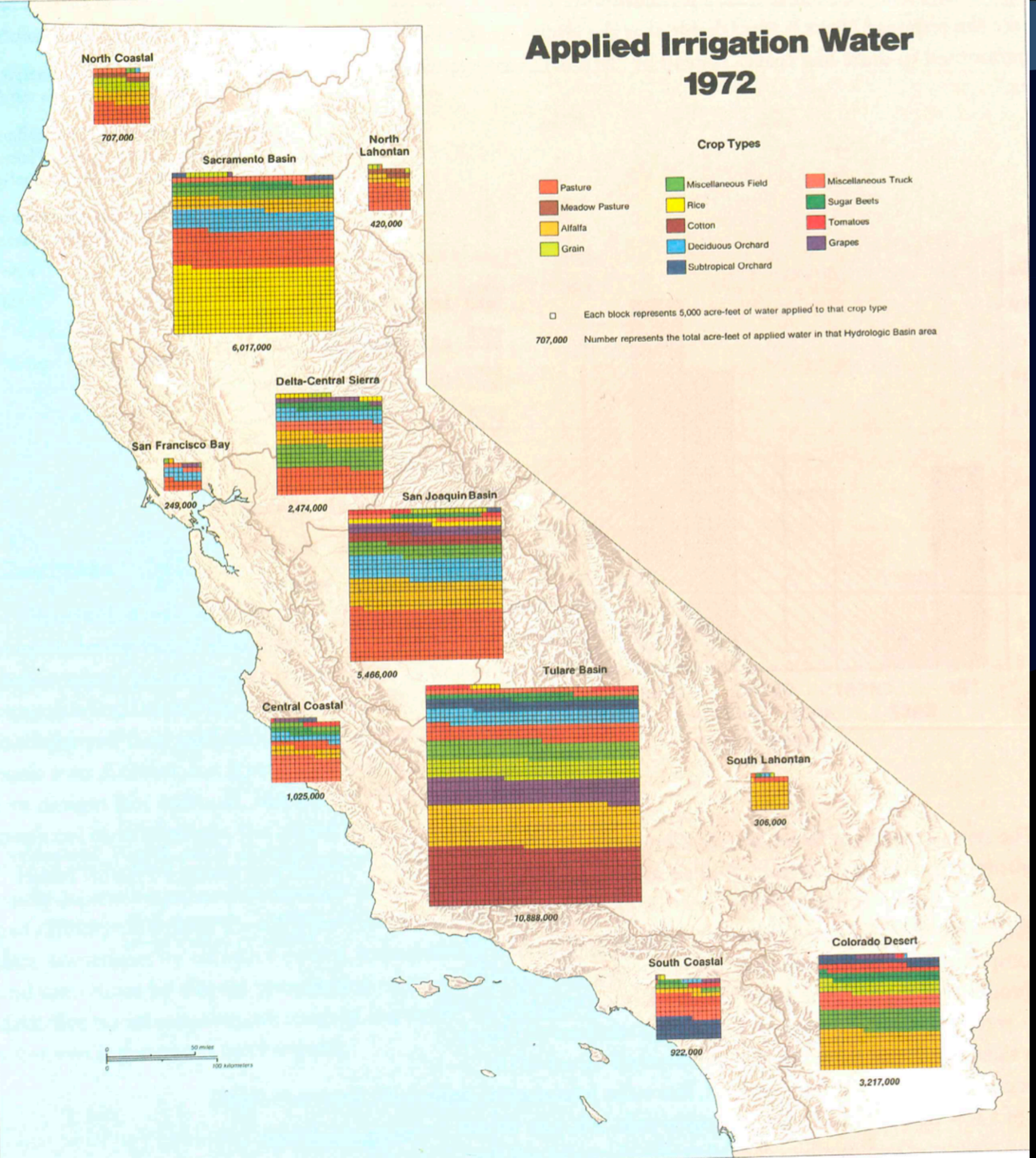
Student Age (M3)

- (A) the graphic is not much more useful than just reading the underlying data "raw"
- (B) the graphic is hard to read or get *any* sense of
- (C) the graphic is easy to misinterpret
- (D) the data are presented in a misleading or inaccurate way

Big Ideas

- Make sure the "ink" you're using actually conveys a piece of information.
- Numbers have *magnitude*. Series of numbers can have *direction.* Make sure to represent both concepts correctly.
- Don't lie, either by making up data or by changing the rules you use to present it.
- Don't make a duck. 🦆


A Duck.



Also A Duck.

"It is alright to decorate construction but never construct decoration."

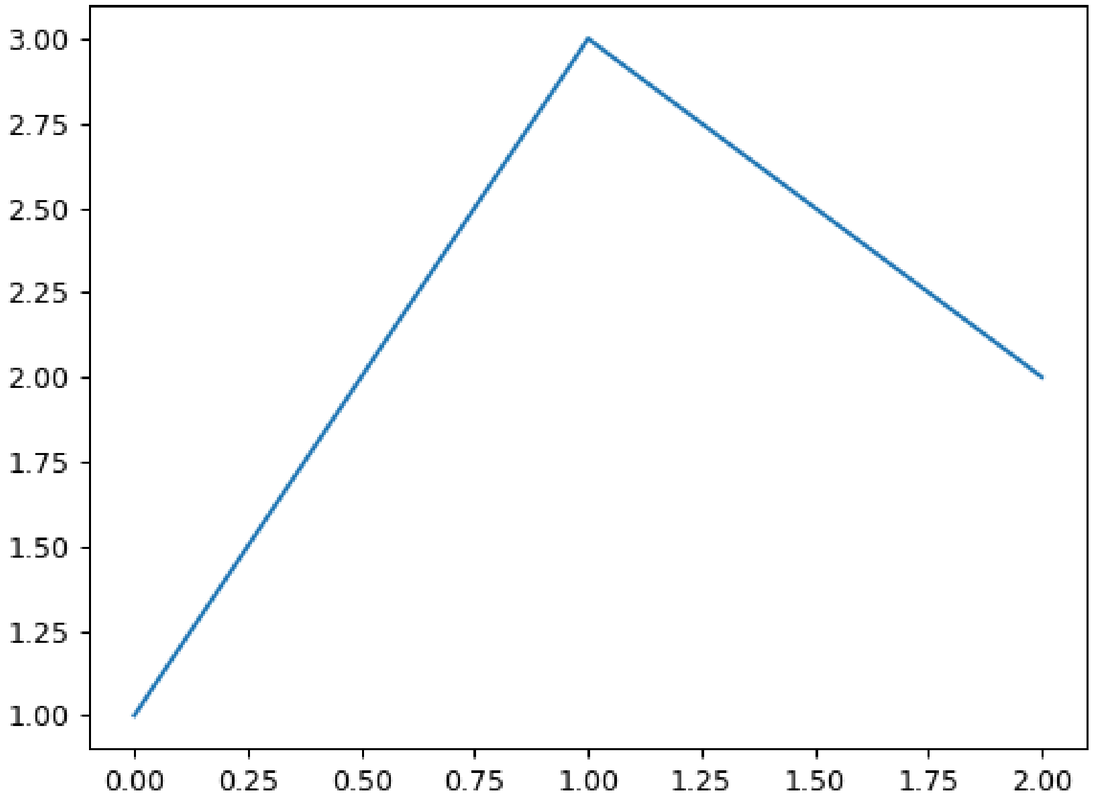
Making Useful and Truthful Graphics

	pig	horse
1990	20	4
1997	18	25
2003	489	281
2009	675	600
2014	1776	1900

Pandas documentation gives us a nice set of examples using this DataFrame.

```
df = pd.DataFrame({
    'pig': [20, 18, 489, 675, 1776],
    'horse': [4, 25, 281, 600, 1900]
    }, index=[1990, 1997, 2003, 2009, 2014])
```

Line Plots

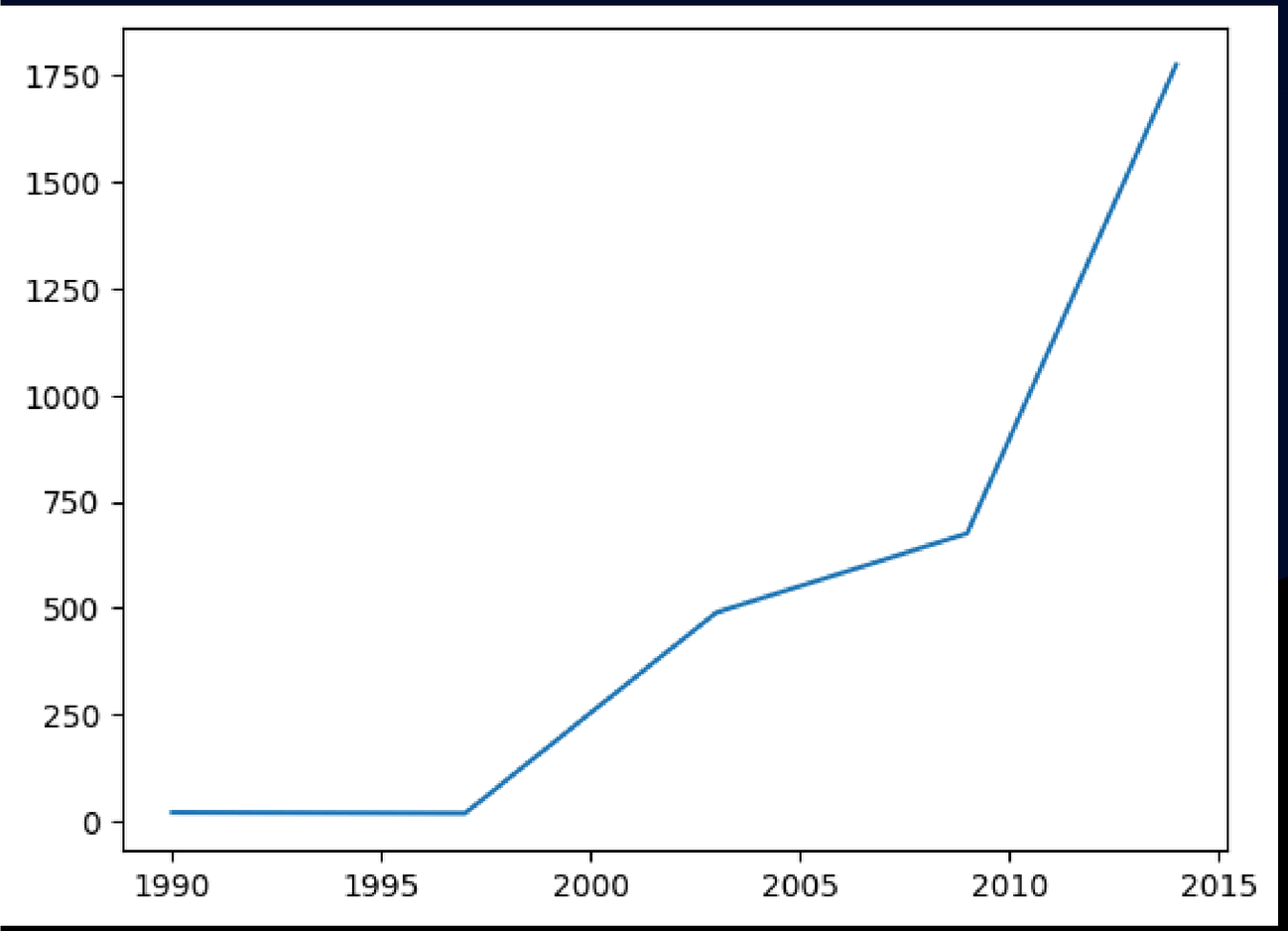


Marks: lines!

Channels: x/y position of segment endpoints

(usually choose x position as *independent variable* and y position as *dependent variable*)

Line Plots



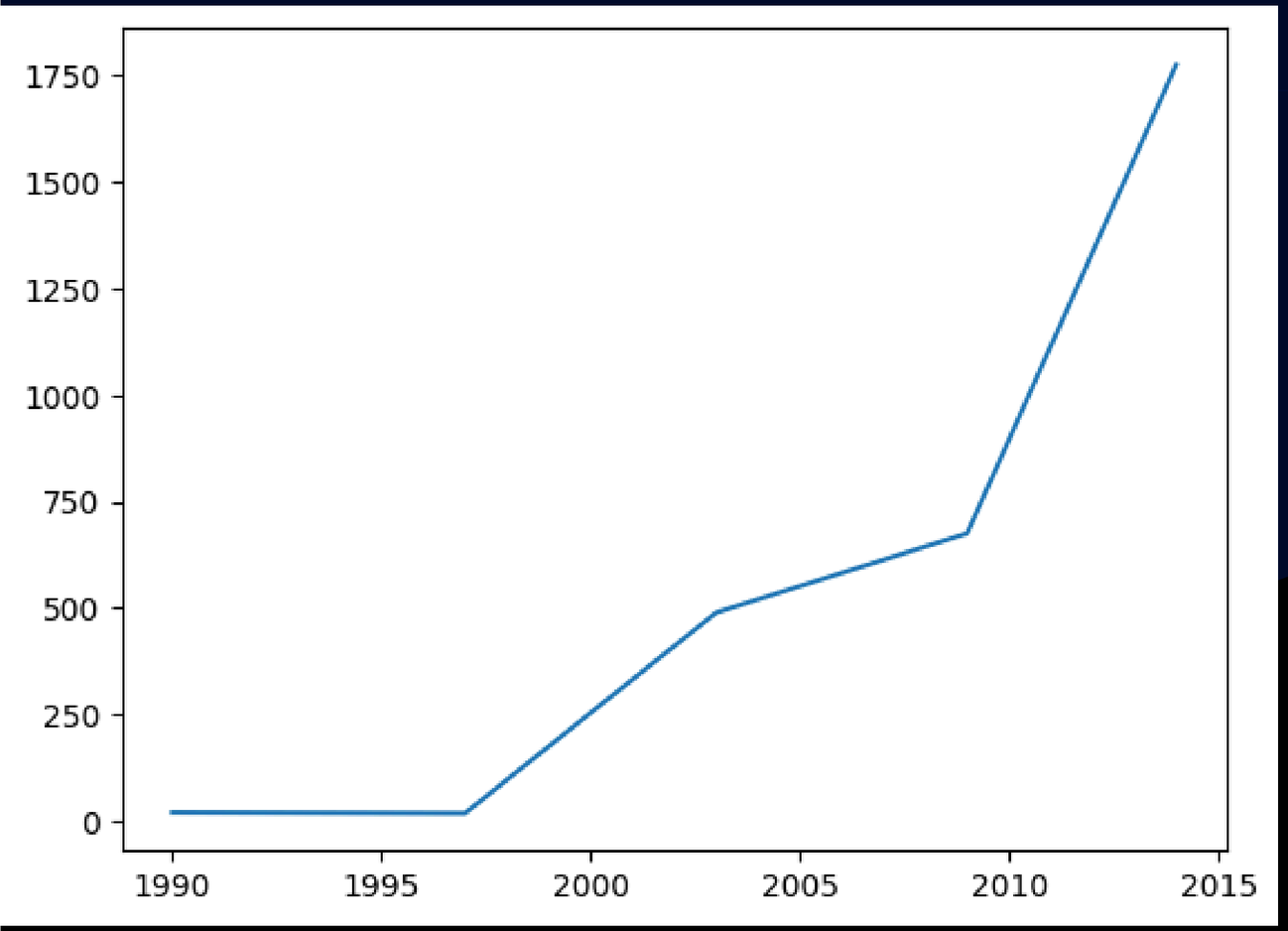
Marks: lines!

Channels: x/y position of segment endpoints

```
df["pig"].plot.line()
```

	pig	horse
1990	20	4
1997	18	25
2003	489	281
2009	675	600
2014	1776	1900

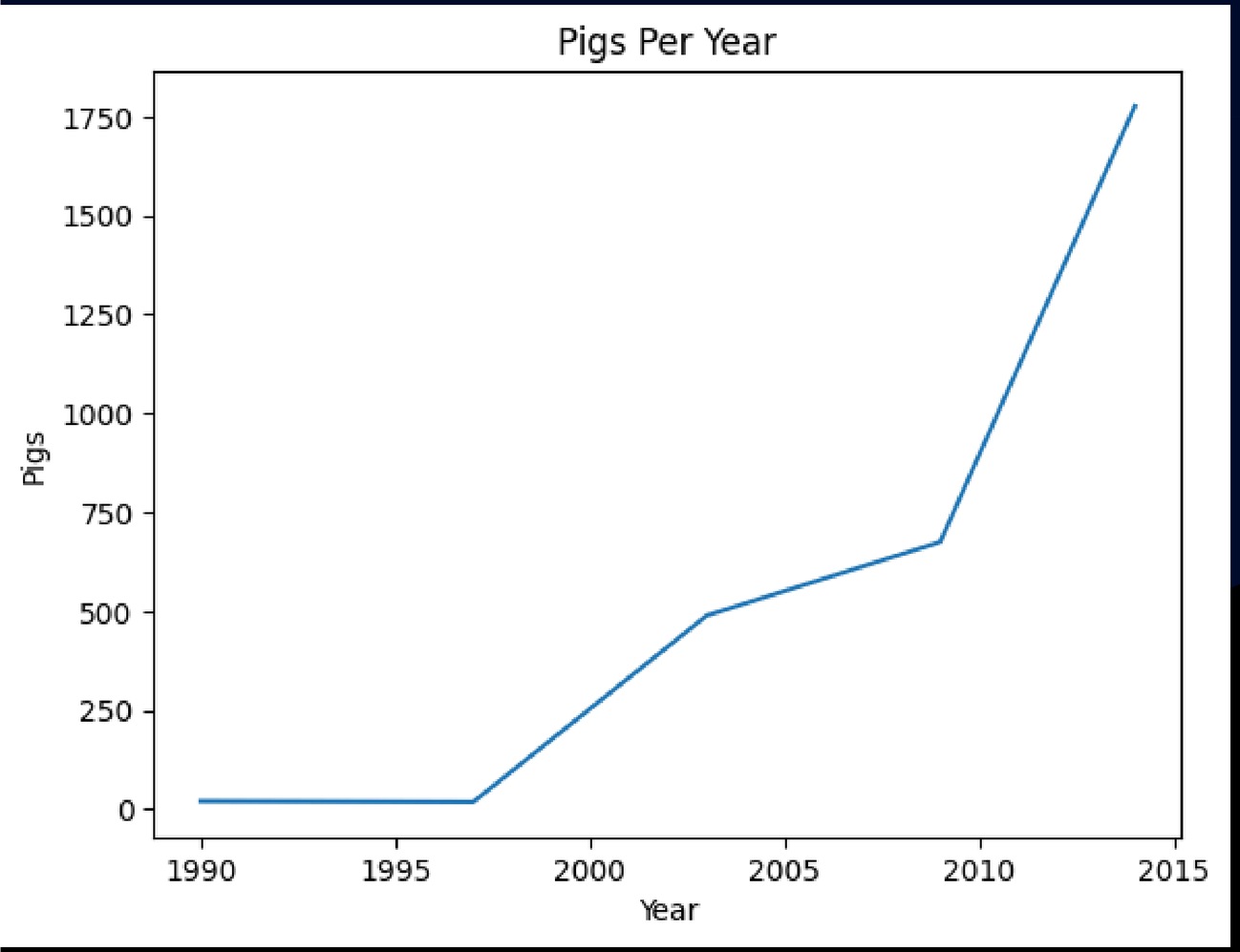
Common Features:



When using `.plot.xyz`...

1. Plotting a Series usually infers the **index** as the range to plot against (here, "year" as an integer.)
2. Axes use standard tick marks and are scaled to use the minimum & maximum values in the range.
3. Nothing else comes for free.

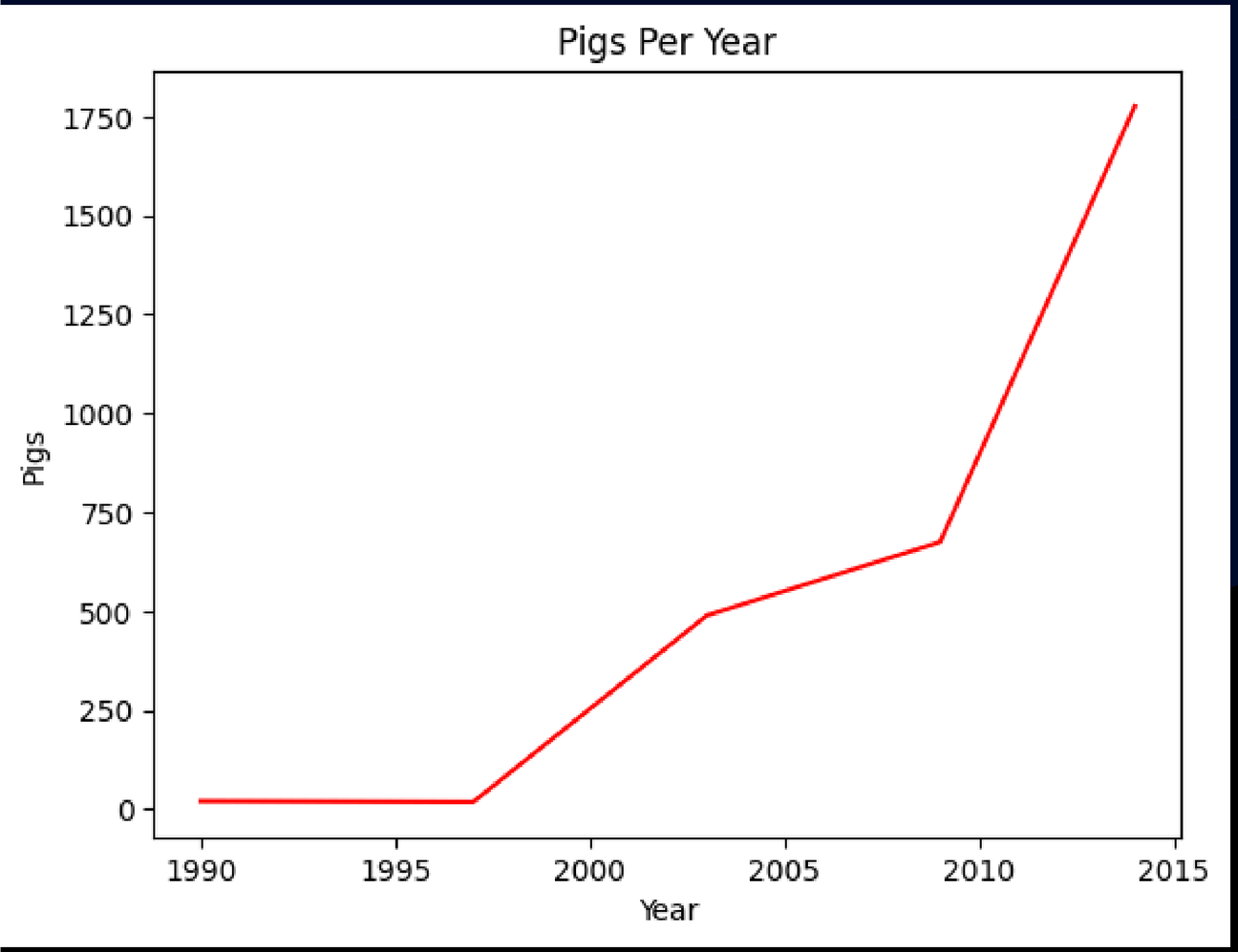
Line Plots



Add titles and axes labels using keyword arguments with predictable names

```
df["pig"].plot.line(title="Pigs Per Year",
                    xlabel="Year",
                    ylabel="Pigs")
```

Line Plots

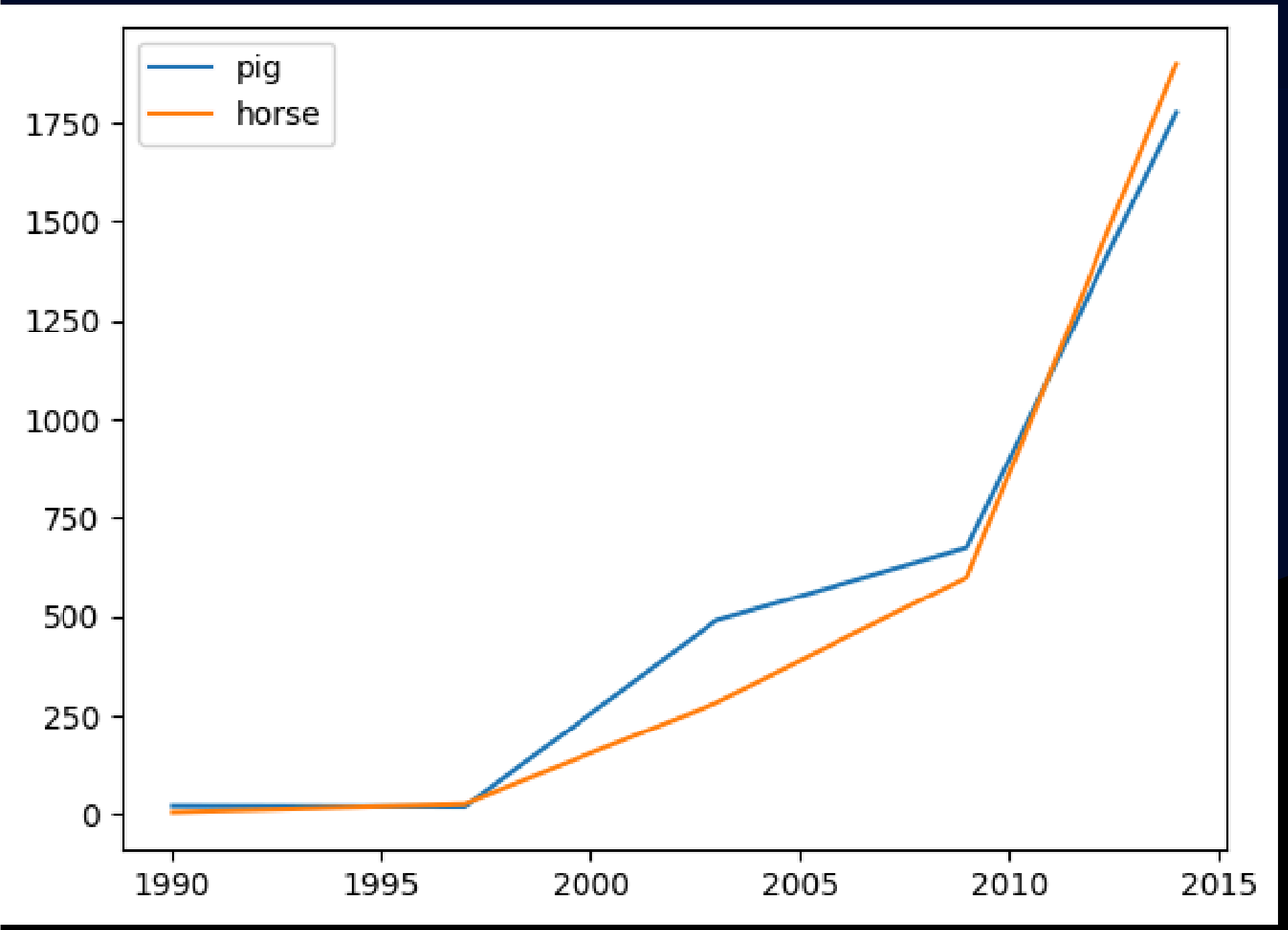


Do the same for color, too.

```
df["pig"].plot.line(title="Pigs Per Year",
                    xlabel="Year",
                    ylabel="Pigs",
                    color="red")
```

Check out all the keyword argument options here.

Line Plots

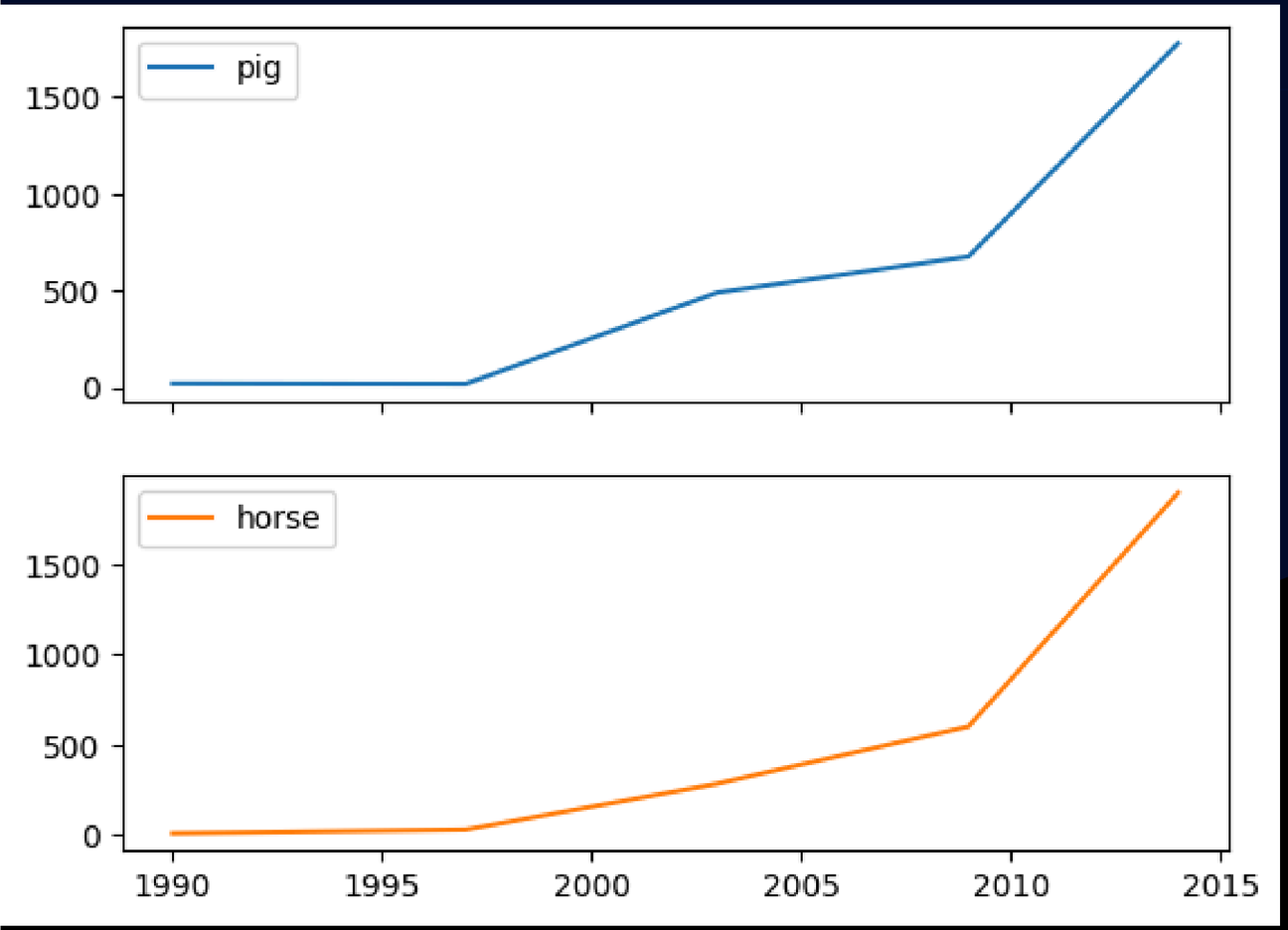


DataFrames contain multiple Series, which can be plotted together against the same variable.

```
df.plot.line()
```

Now there's a new channel for color, which encodes animal type.

Line Plots & Small Multiples



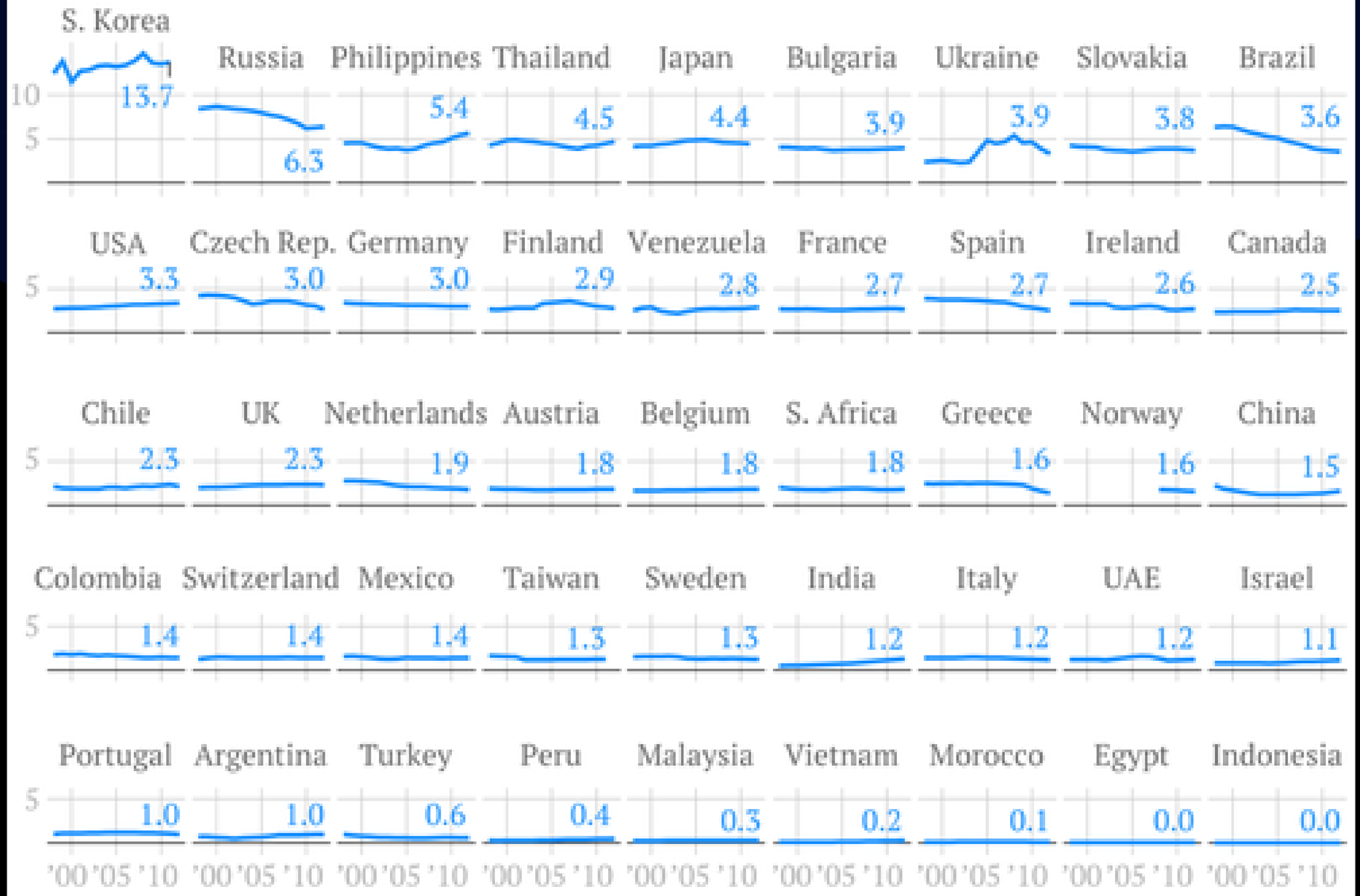
Lines on top of each other are good when Series are measured on a similar scale. You could also produce **small multiples** of the same kind of plot.

```
df.plot.line(subplots=True)
```

Color & y-position now both encode animal type.

The average amount of liquor consumed by a person of drinking age

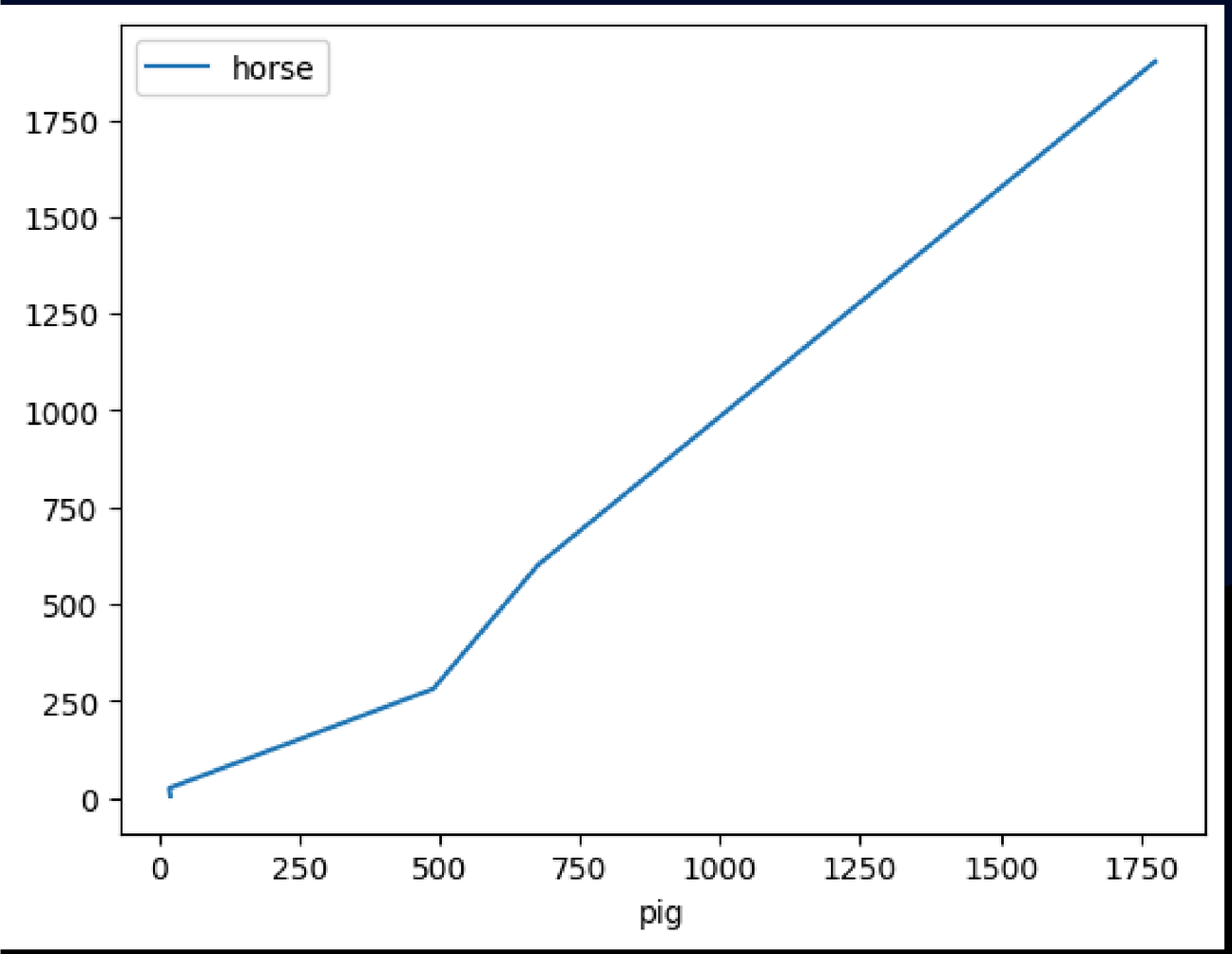
Shots per week of any spirit



Quartz | Ritchie King

Data: Euromonitor

Line Plots

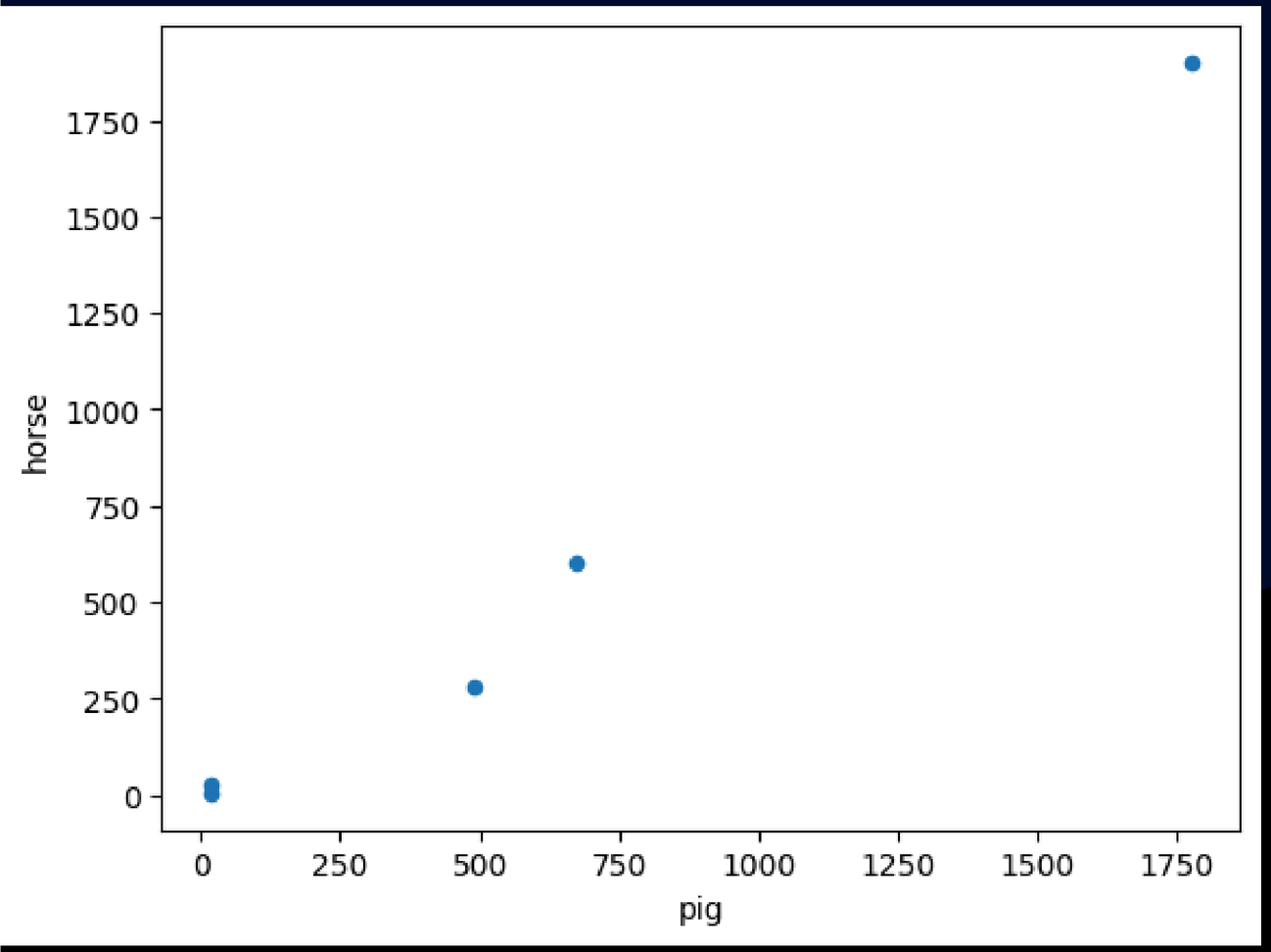


You can also replace the use of the index as the default x variable.

```
df.plot.line(x='pig', y='horse')
```

Pig—Horse Correlation, anyone?

Scatter Plots



Line Plots imply a kind of continuous change between points. Makes sense for *time* but not in every context.

```
df.plot.scatter(x='pig', y='horse')
```

Scatter Plots are good for showing individual points in isolation.

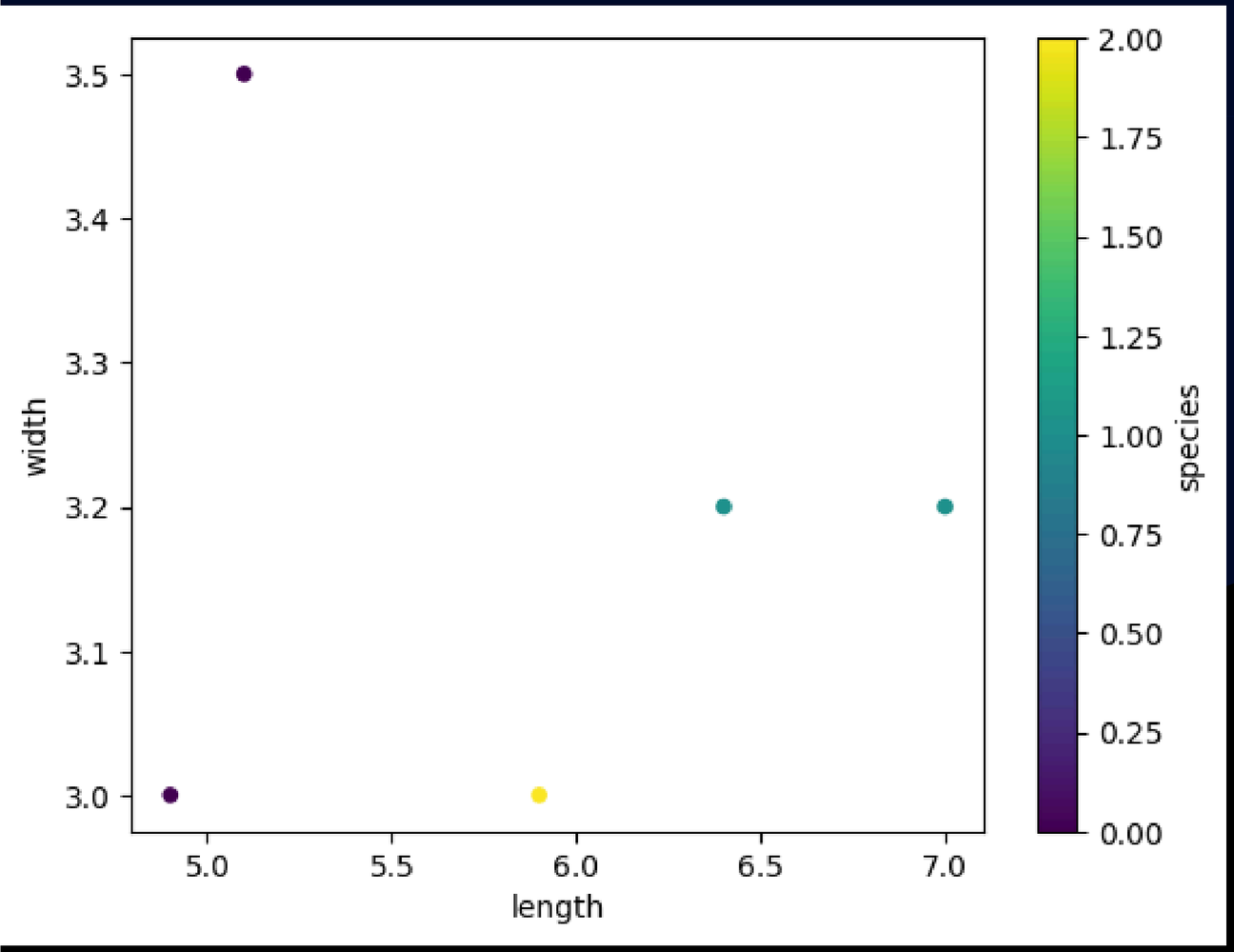
A Big Virtue of Scatter Plots: Color!

	length	width	species
0	5.1	3.5	0
1	4.9	3.0	0
2	7.0	3.2	1
3	6.4	3.2	1
4	5.9	3.0	2

A different dataset for different species of Iris flowers.

```
irises = pd.DataFrame(
         [ [5.1, 3.5, 0],
           [4.9, 3.0, 0],
           [7.0, 3.2, 1],
           [6.4, 3.2, 1],
           [5.9, 3.0, 2] ],
         columns=['length', 'width', 'species']
)
```

Scatter Plots

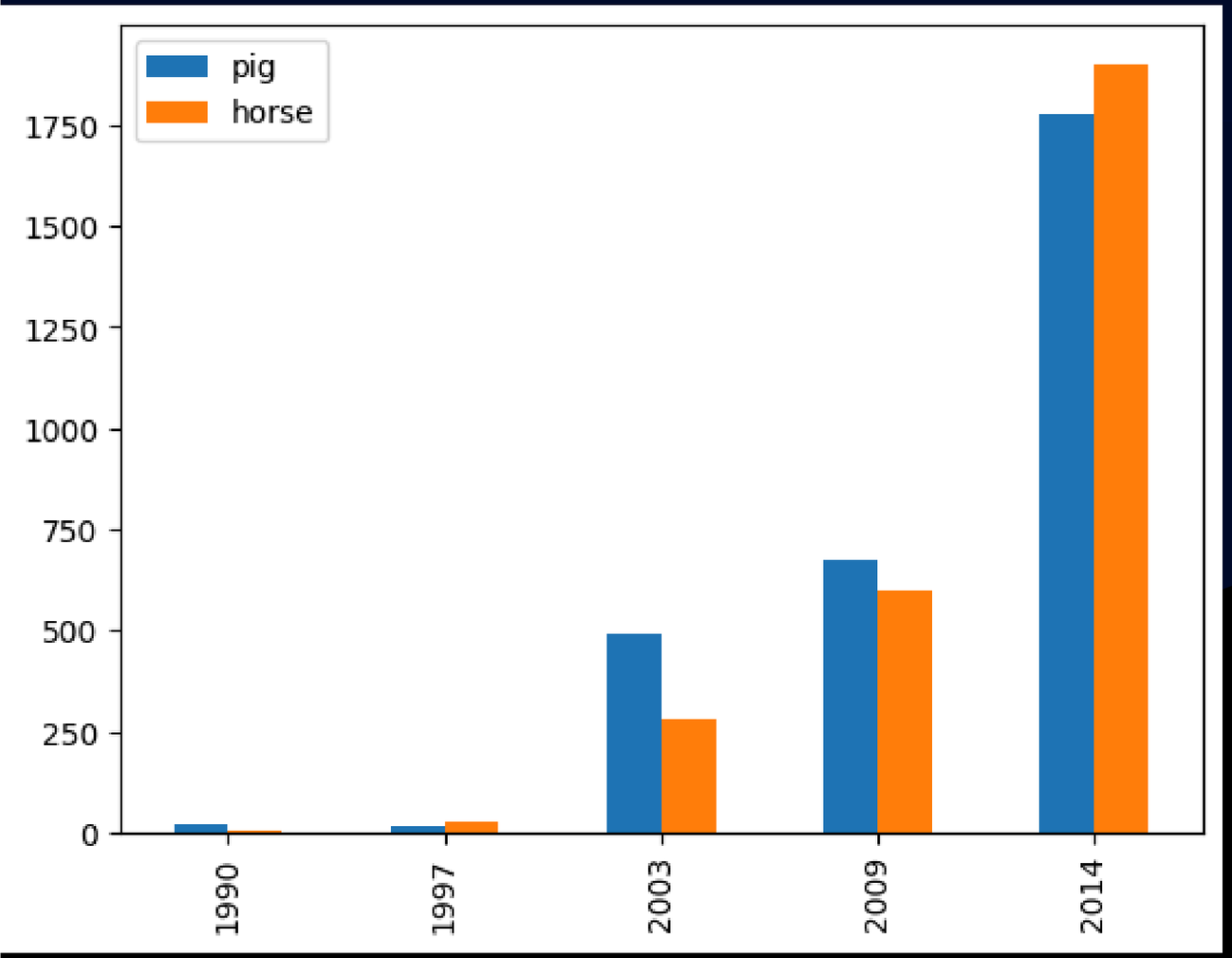


Scatter Plots can be especially effective for seeing how data from different categories can compare.

```
irises.plot.scatter(x='length',
                    y='width',
                    c='species',
                    colormap='viridis')
```

The color bar leaves a bit to be desired since the "species" variable is not actually continuous, but we'll ignore for now.

Bar Plots

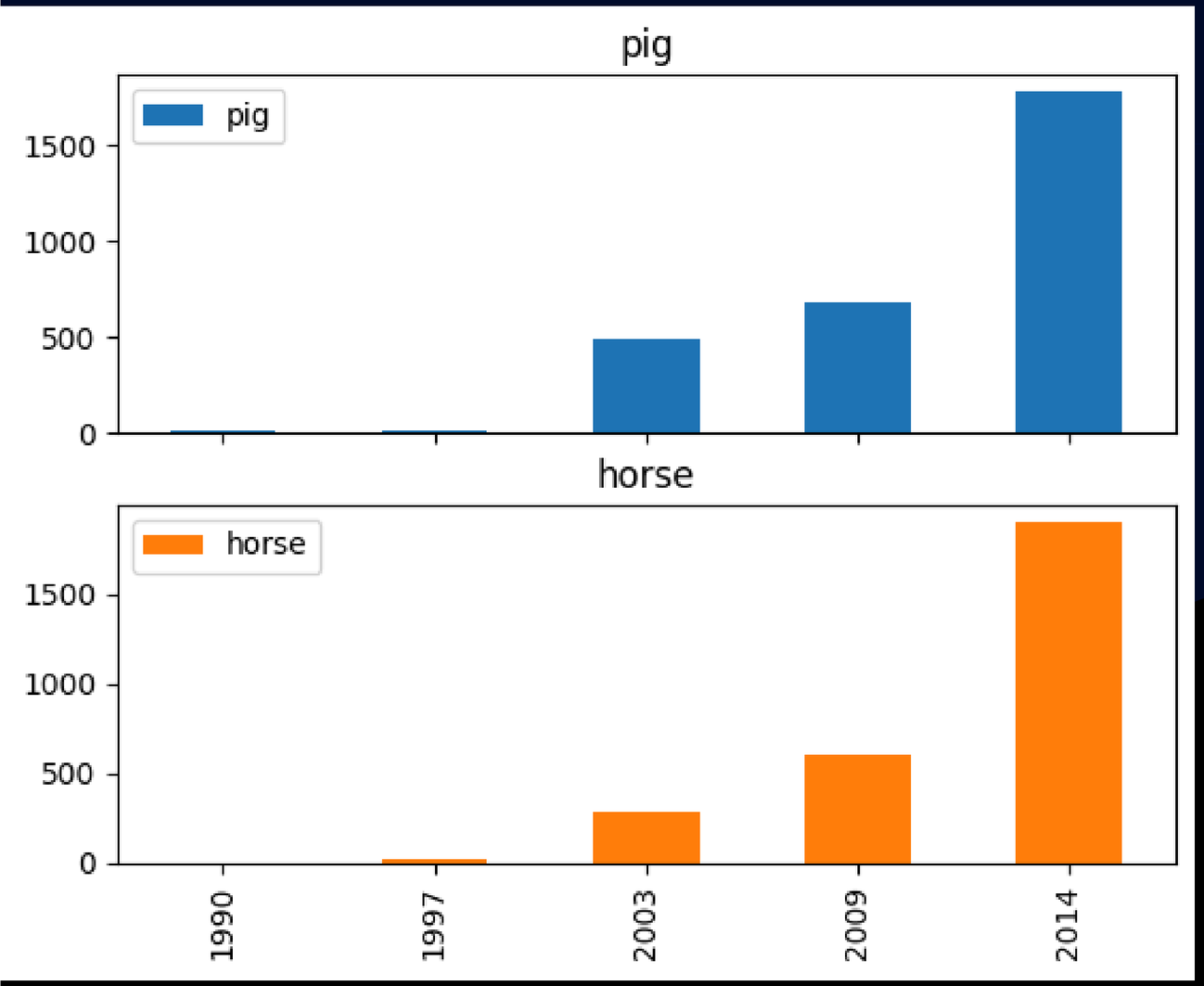


Back to pigs and horses! Bars are nice for quick comparisons between two categories in a particular dimension.

```
df.plot.bar()
```

Encodes similar information to the double Line Plot.

Bar Plots

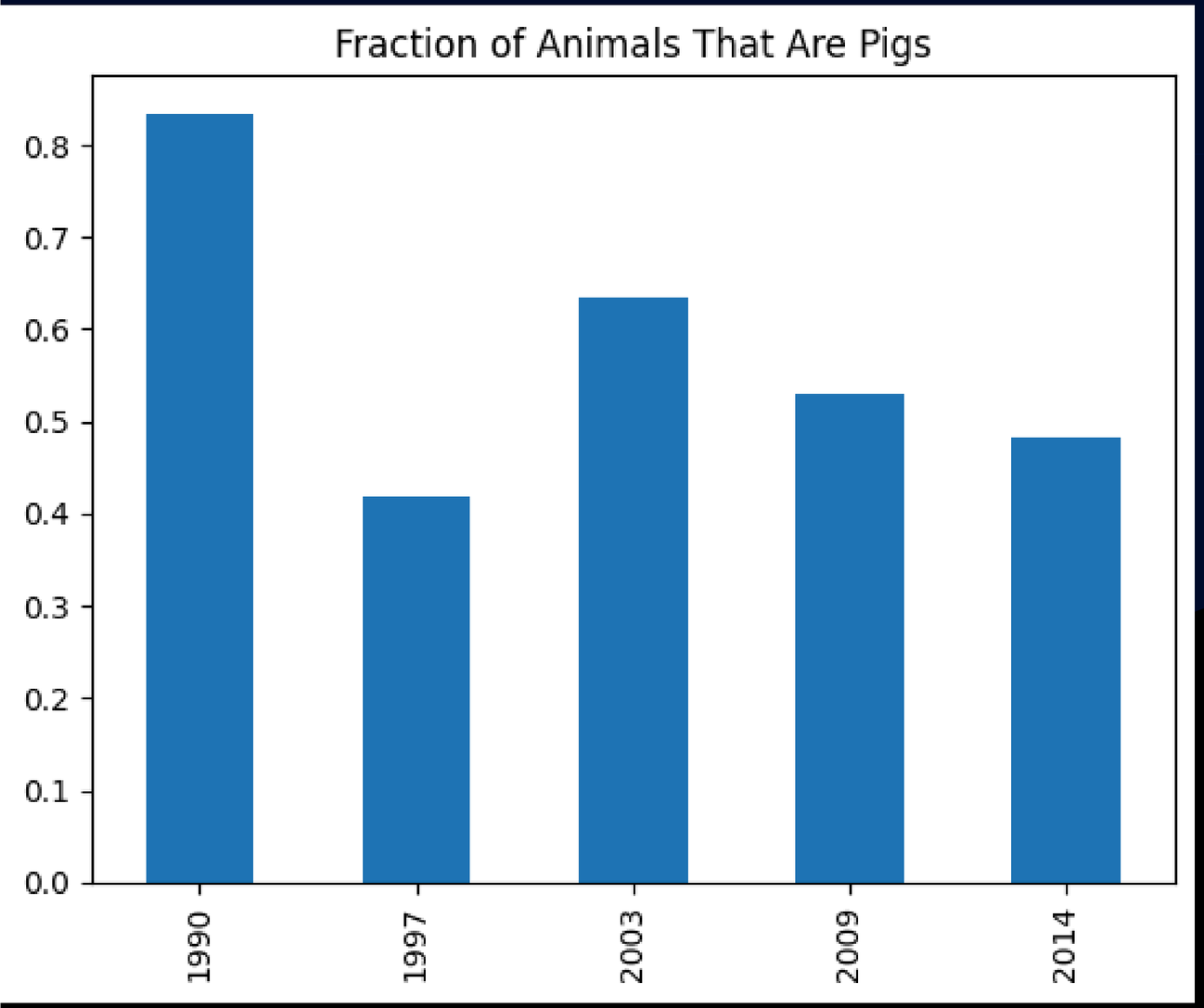


Can do small multiples again, but beware of axis consistency.

```
df.plot.bar(subplots=True)
```

Works well for comparing trends in pigs and horses, but much harder to see year vs. year.

Bar Plots

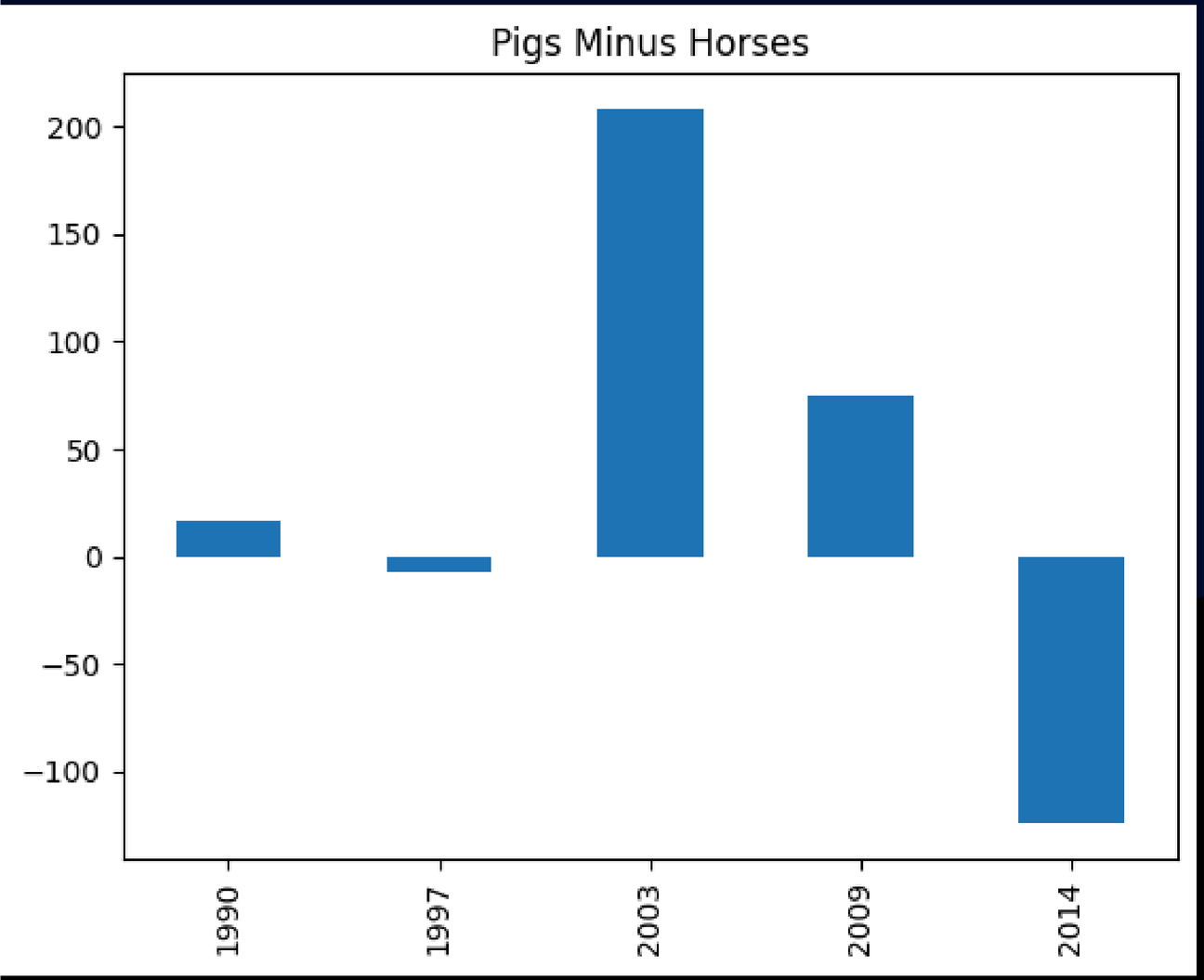


Can also create new Series to represent with the same marks.

```
pct_pig = df["pig"] / (df["pig"] + df["horse"])
pct_pig.plot.bar(title="Fraction of Animals That Are Pigs")
```

If the differences between the heights of the paired bars were all that you cared about, then you can express that in *relative* terms...

Bar Plots



Or *absolute* terms...

```
pig_diff = df["pig"] - df["horse"]
pig_diff.plot.bar(title="Pigs Minus Horses")
```

This chart probably needs a line at `y=0`—kinda hard to see if a bar is negative or positive.