COMS BC1016

Introduction to Computational Thinking and Data Science

Lecture 4: Functions and Charts

Sept. 3, 2025

February 4, 2026

Office Hours

Office hours started this week!

- **Monday**:
  - **Elena Lukac**, 1:30-3pm in Milstein 503
  - Eysa Lee, 3-5pm in Milstein 512
- **Tuesday**: **Nami Jain**, 4-5:30pm in Milstein 503
- **Wednesday**: **Madeline Gutierrez**, 5:30-7pm in Milsten 503
- **Thursday**: **Sathya Raman**, 4-5:30pm in Milstein 503

Elena and Nami are from the Wednesday labs

Madeline and Sathya are from the Thursday labs

Notes about the assignments

- Homeworks and labs may use functions we haven’t covered in class
  - For example, HW 1 has you use the `join` function (which we haven’t and won’t cover in class)
- Lectures we’ll focus on concepts, and labs/homework will be practicing and applying these concepts
  - Programming is all about practice and trying things out yourself!

Functions and Methods

Functions vs Methods

- **Functions** can be run independently, while **methods** are associated with an object

Table object

Function	Method

`max(1, 5)`	`skyscrapers = Table.read_table(’skyscrapers.csv’)`
	`skyscrapers.num_rows`

method

Functions vs Methods

- It's not just about whether there's a dot!

Array object

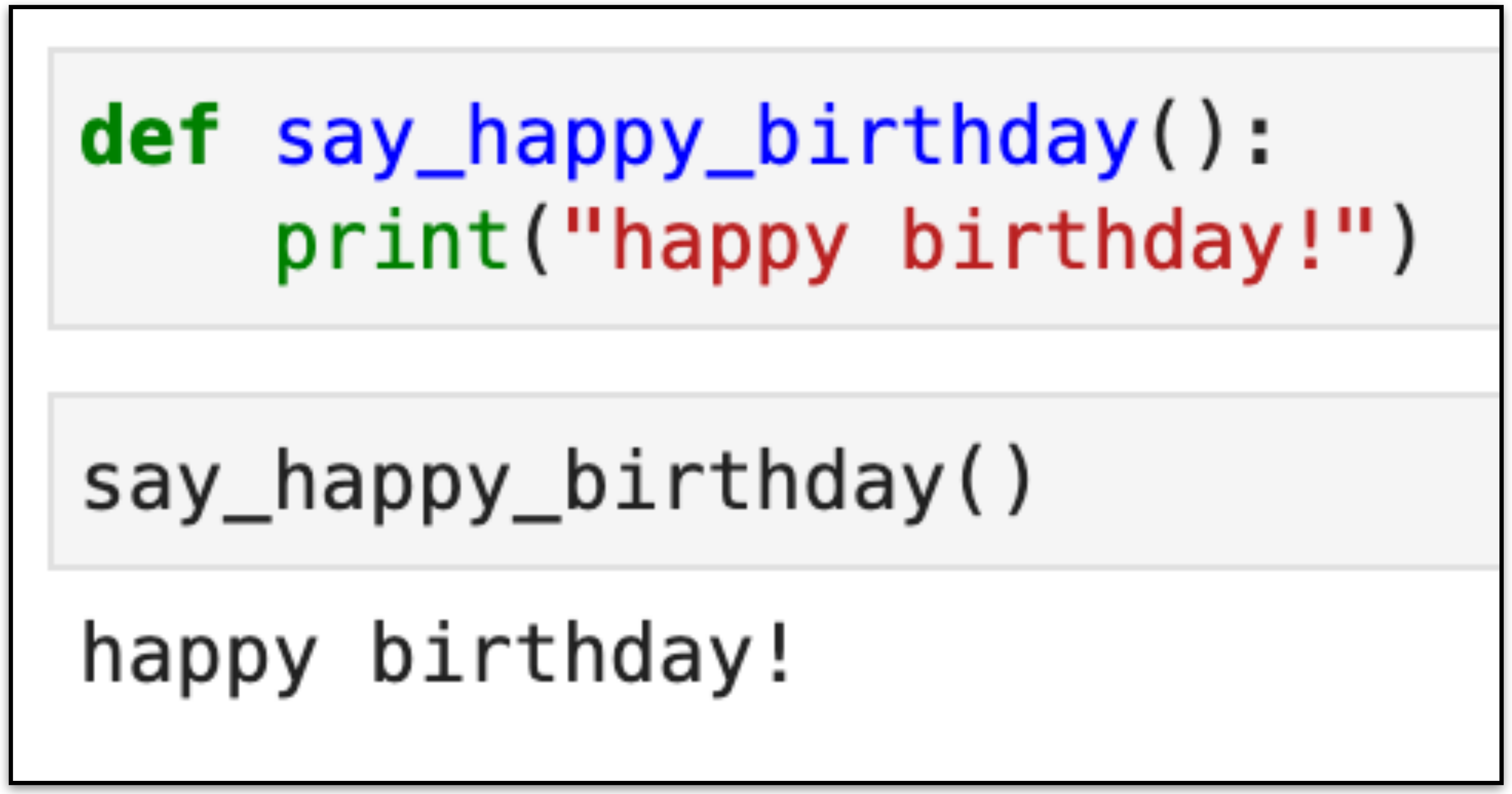
Function	Method

`np.average(make_array(1, 2, 3))`	`my_array = make_array(1, 2, 3)`
	`my_array.item(0)`

NumPy library (not object!)

Defining functions

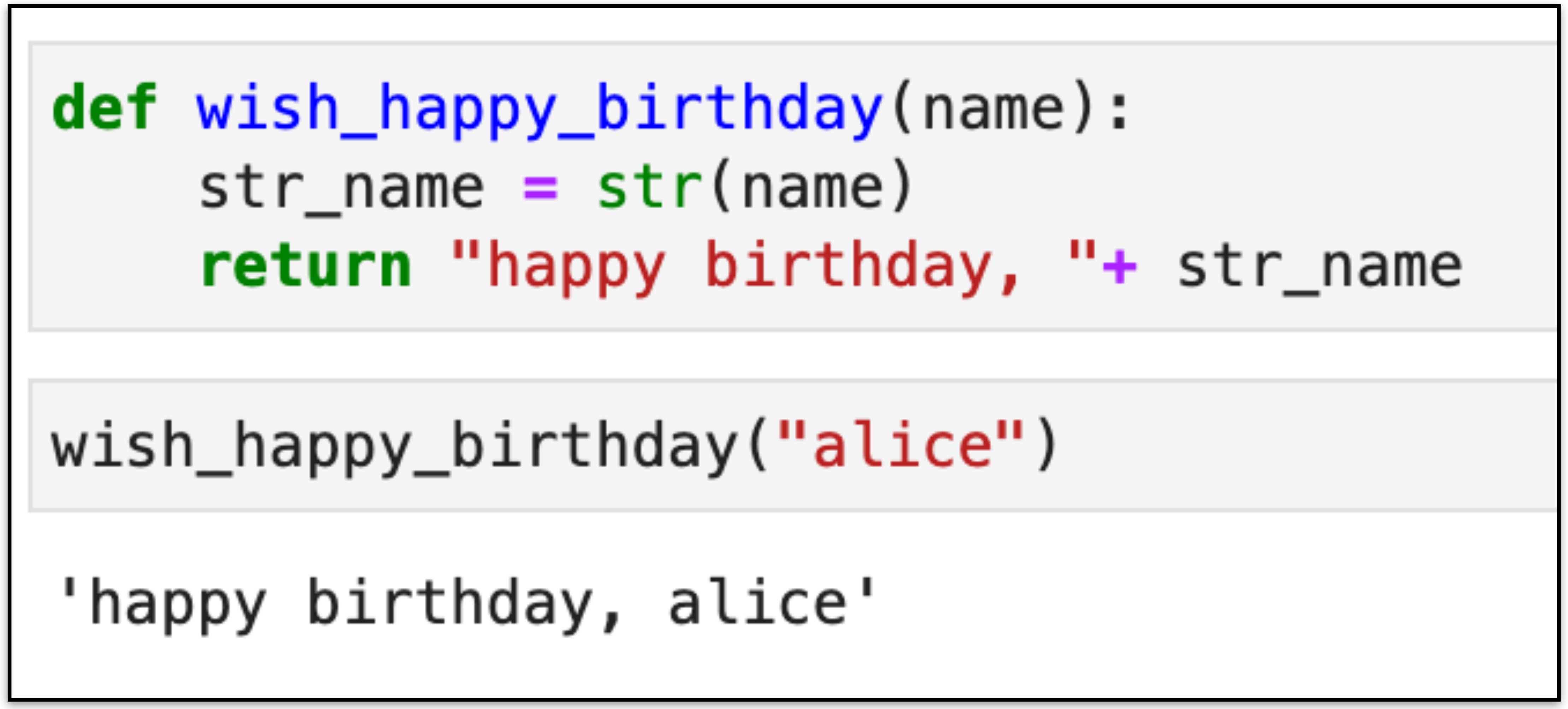
- Use `def` to define your own function!
  - The code you want to execute in the function starts on a new line with a single indent
  - You can optionally use `return` to have the function output a specific value



```
def say_happy_birthday():
    print("happy birthday!")
```

```
say_happy_birthday()
```

```
happy birthday!
```



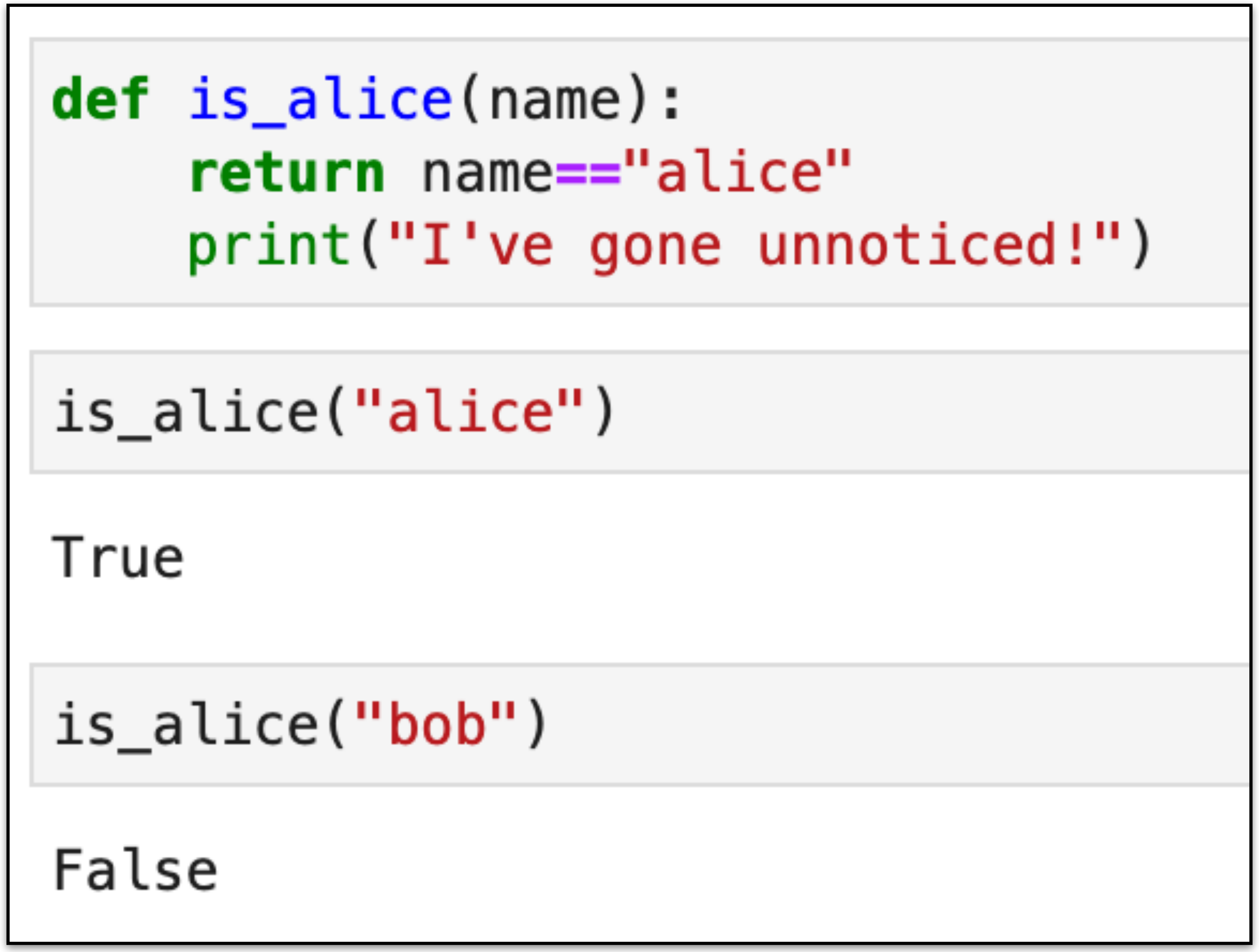
```
def wish_happy_birthday(name):
    str_name = str(name)
    return "happy birthday, "+ str_name
```

```
wish_happy_birthday("alice")
```

```
'happy birthday, alice'
```

Tips for writing functions

- Avoid naming your function something that already exists
- `return` will immediately exit a function
    - Typically goes at the end
- Variables defined *inside* the function only exist within the function
    - If you try to access it outside of the function you'll get an error!



```
def is_alice(name):
    return name=="alice"
    print("I've gone unnoticed!")
```

```
is_alice("alice")
```

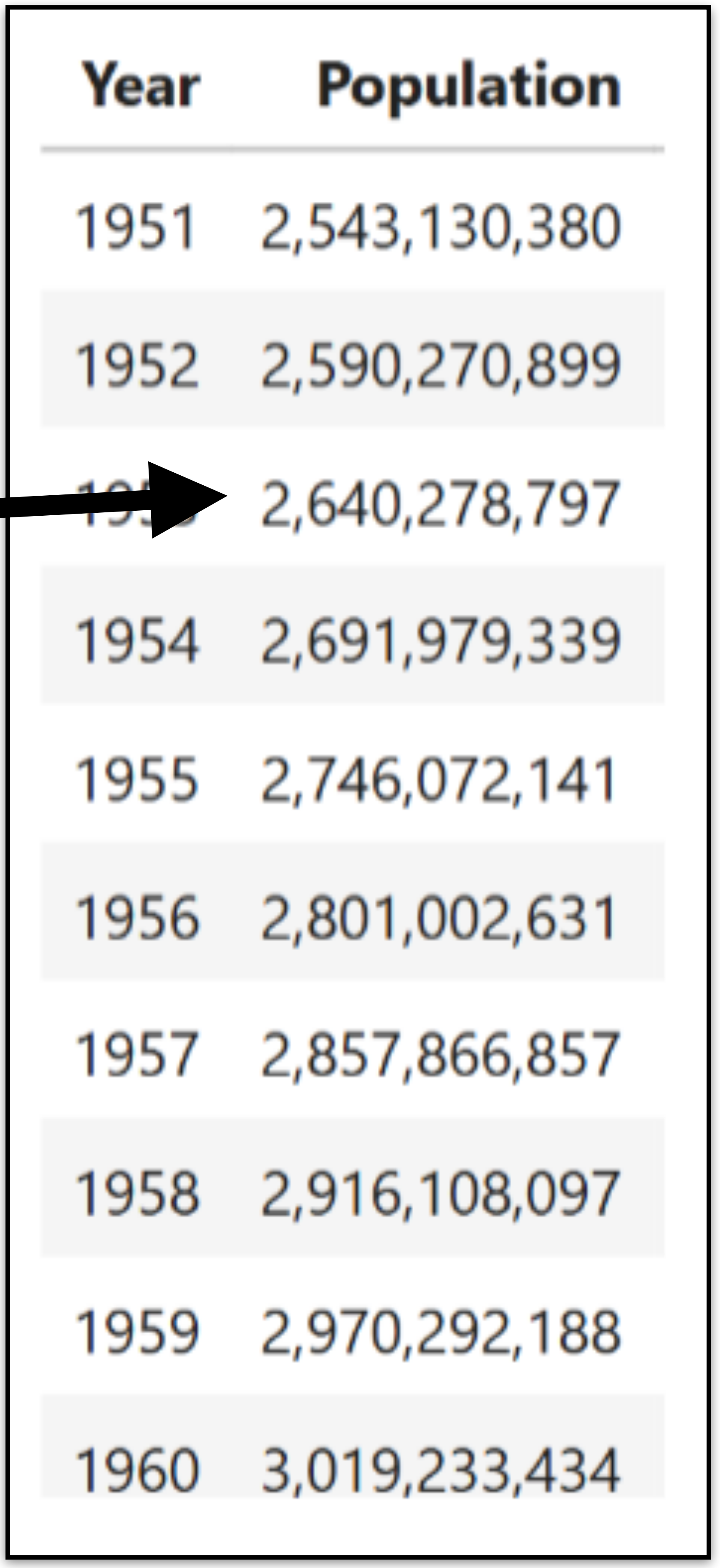
```
True
```

```
is_alice("bob")
```

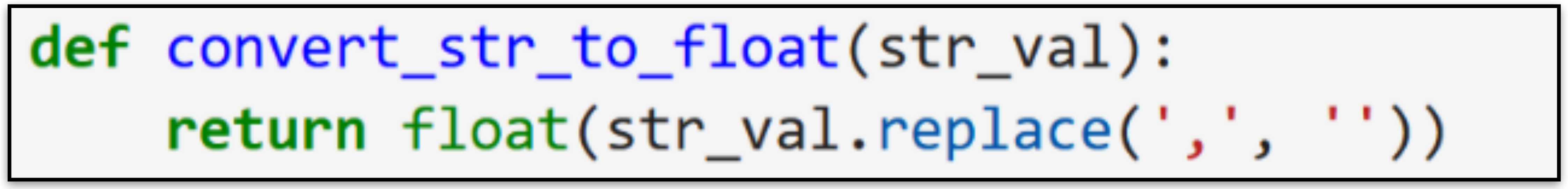
```
False
```

Example: Converting Strings to Numbers

- Sometimes when you import data it might be interpreted as a string instead of a number
    - Notice the `,`?
- To analyze this, we might need to convert that string to a numerical value
- How can we do that?



Year	Population
1951	2,543,130,380
1952	2,590,270,899
1953	2,640,278,797
1954	2,691,979,339
1955	2,746,072,141
1956	2,801,002,631
1957	2,857,866,857
1958	2,916,108,097
1959	2,970,292,188
1960	3,019,233,434



```python
def convert_str_to_float(str_val):
    return float(str_val.replace(',', ''))
```

Example: Converting Strings to Numbers

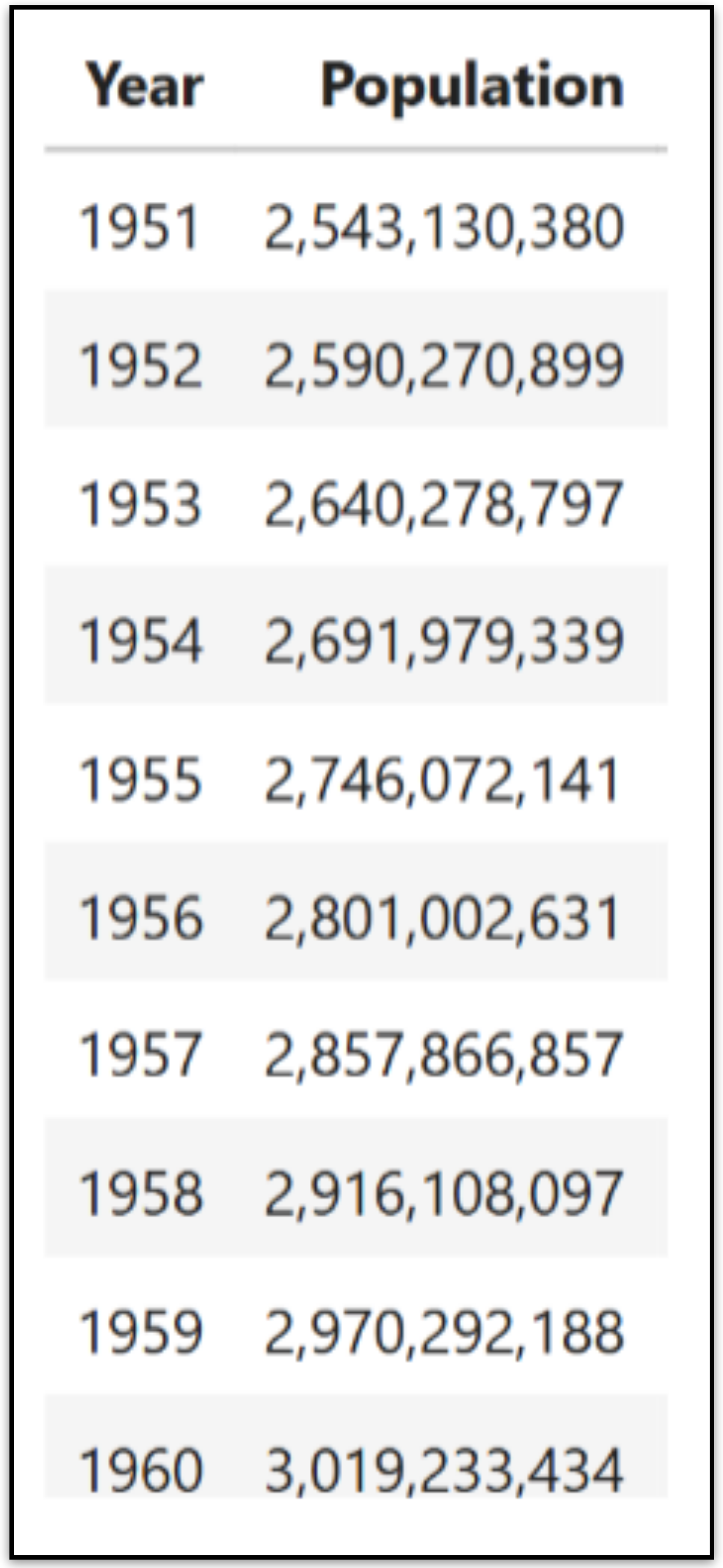
Once we define a function `convert_str_to_float`, two options for converting this:

1. Manually apply the function to each item

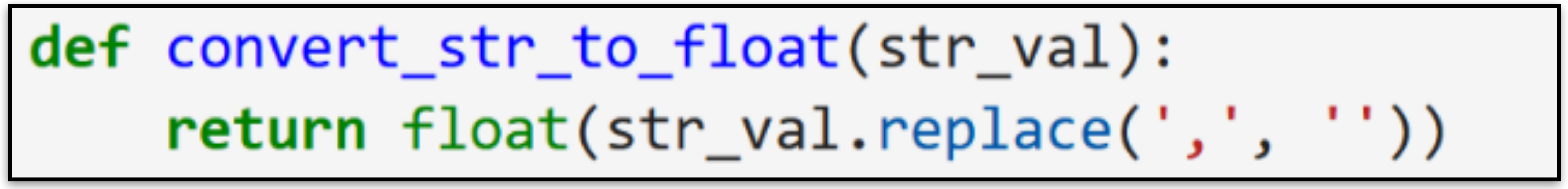
```
item0 = 
tbl.column(‘Population’).item(0)

convert_str_to_float(item0)
```

2. Use `apply` to this function to all values

```
tbl.apply(convert_str_to_float, 
‘Population’)
```



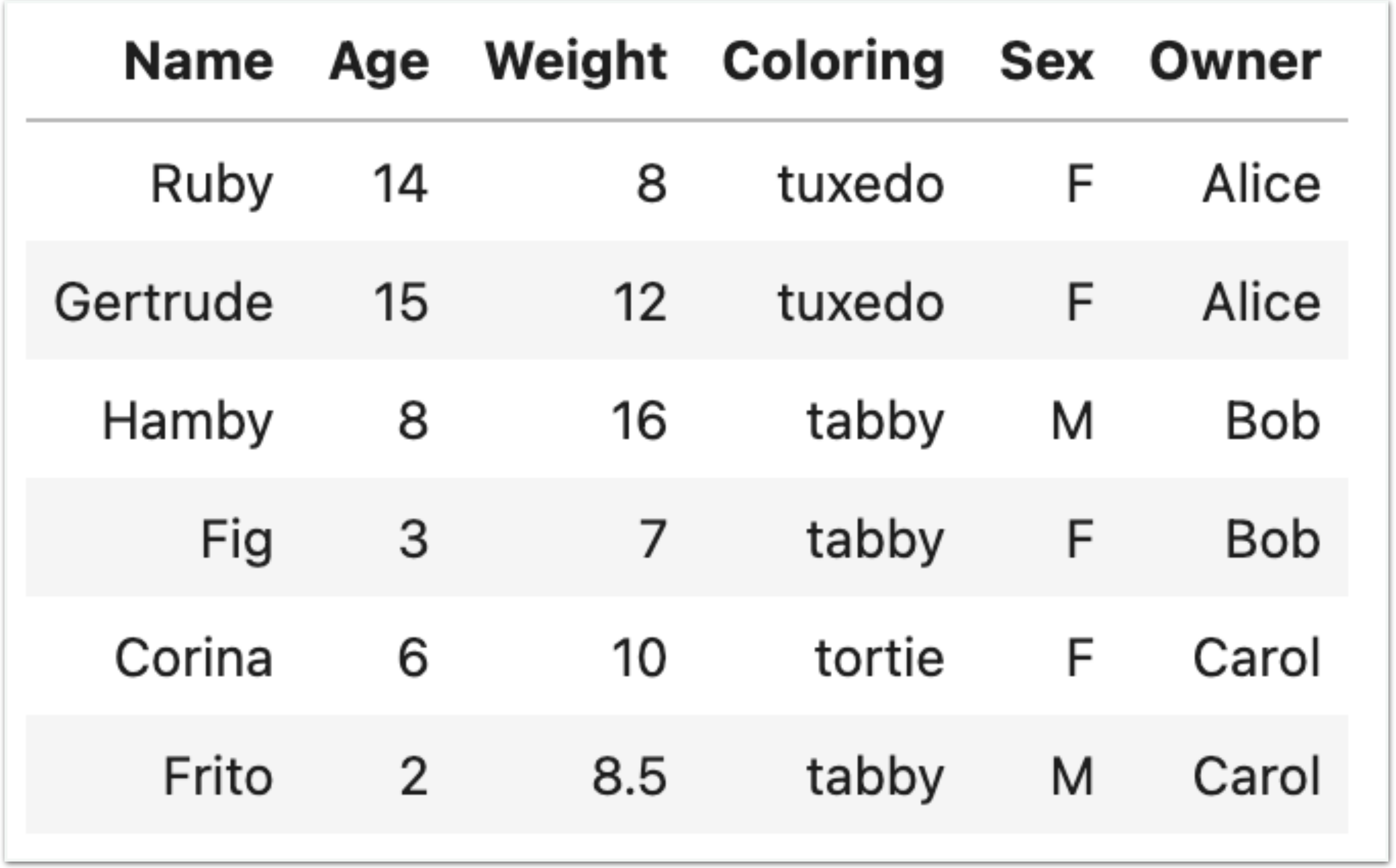
Year	Population
1951	2,543,130,380
1952	2,590,270,899
1953	2,640,278,797
1954	2,691,979,339
1955	2,746,072,141
1956	2,801,002,631
1957	2,857,866,857
1958	2,916,108,097
1959	2,970,292,188
1960	3,019,233,434



```
def convert_str_to_float(str_val):
    return float(str_val.replace(',', ''))
```

Example: Prof Lee’s 2025 Cat Census

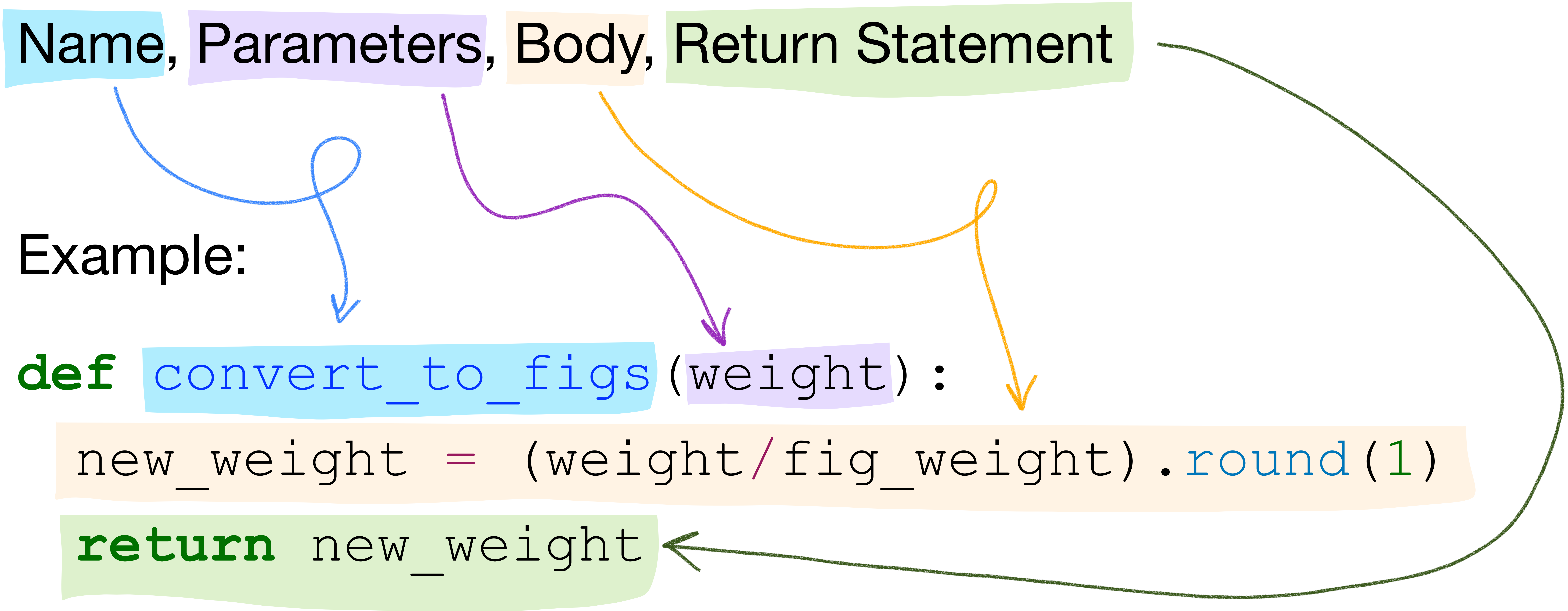
Professor Lee is in a cat picture group chat. In 2025 she collected data on the cats shared in this chat:



Name	Age	Weight	Coloring	Sex	Owner
Ruby	14	8	tuxedo	F	Alice
Gertrude	15	12	tuxedo	F	Alice
Hamby	8	16	tabby	M	Bob
Fig	3	7	tabby	F	Bob
Corina	6	10	tortie	F	Carol
Frito	2	8.5	tabby	M	Carol

What if she wanted to create a function to convert all of the cats’ weights into units of the smallest cat (Fig)?

Anatomy of a Function



Name, Parameters, Body, Return Statement

Example:

```
def convert_to_figs(weight):
  new_weight = (weight/fig_weight).round(1)
  return new_weight
```

Example: Prof Lee’s Cat Census

Once we’ve defined `convert_to_figs`, two options for converting each element:

1. Manually apply the function to each item

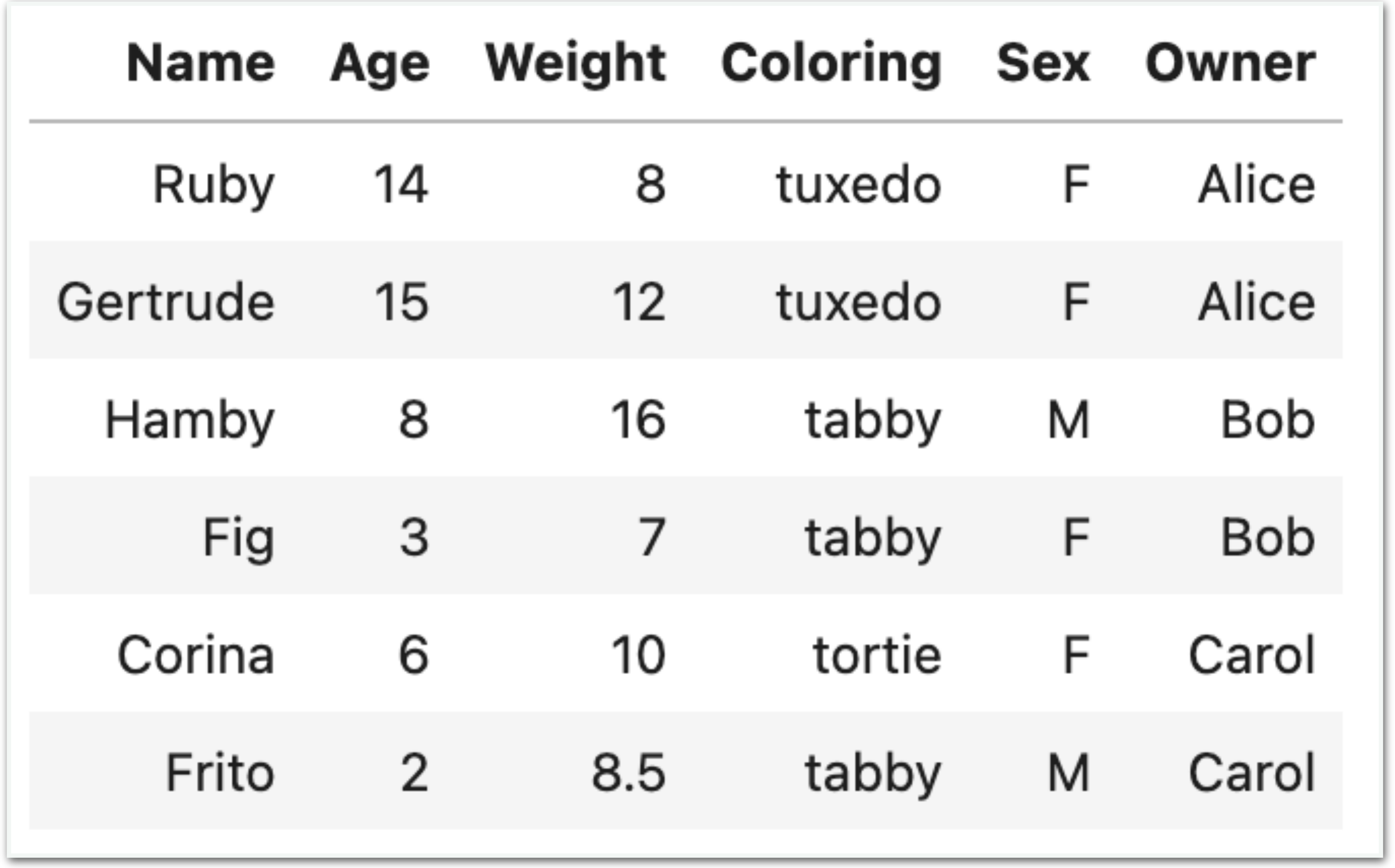
```
item0 =
tbl.column(‘Weight’).item(0)
```

```
convert_to_figs(item0)
```

2. Use **`apply`** to apply the function to all values in the column

```
tbl.apply(convert_to_figs,’Weight’)
```

Returns an array with convert_to_figs called on each element in the 'Weight' column



Name	Age	Weight	Coloring	Sex	Owner
Ruby	14	8	tuxedo	F	Alice
Gertrude	15	12	tuxedo	F	Alice
Hamby	8	16	tabby	M	Bob
Fig	3	7	tabby	F	Bob
Corina	6	10	tortie	F	Carol
Frito	2	8.5	tabby	M	Carol

Charts

Types of Attributes

- Attributes are the names of columns in tables
- All values in a column should be the same type and comparable to each other
  - **Numerical:** Values are on a numerical scale (e.g., years)
    - Values are ordered
    - Differences are meaningful
  - **Categorical:** Each value is from a fixed inventory (e.g., material)
    - May not have an ordering
    - Categories are either the same or different

Numerical Attributes

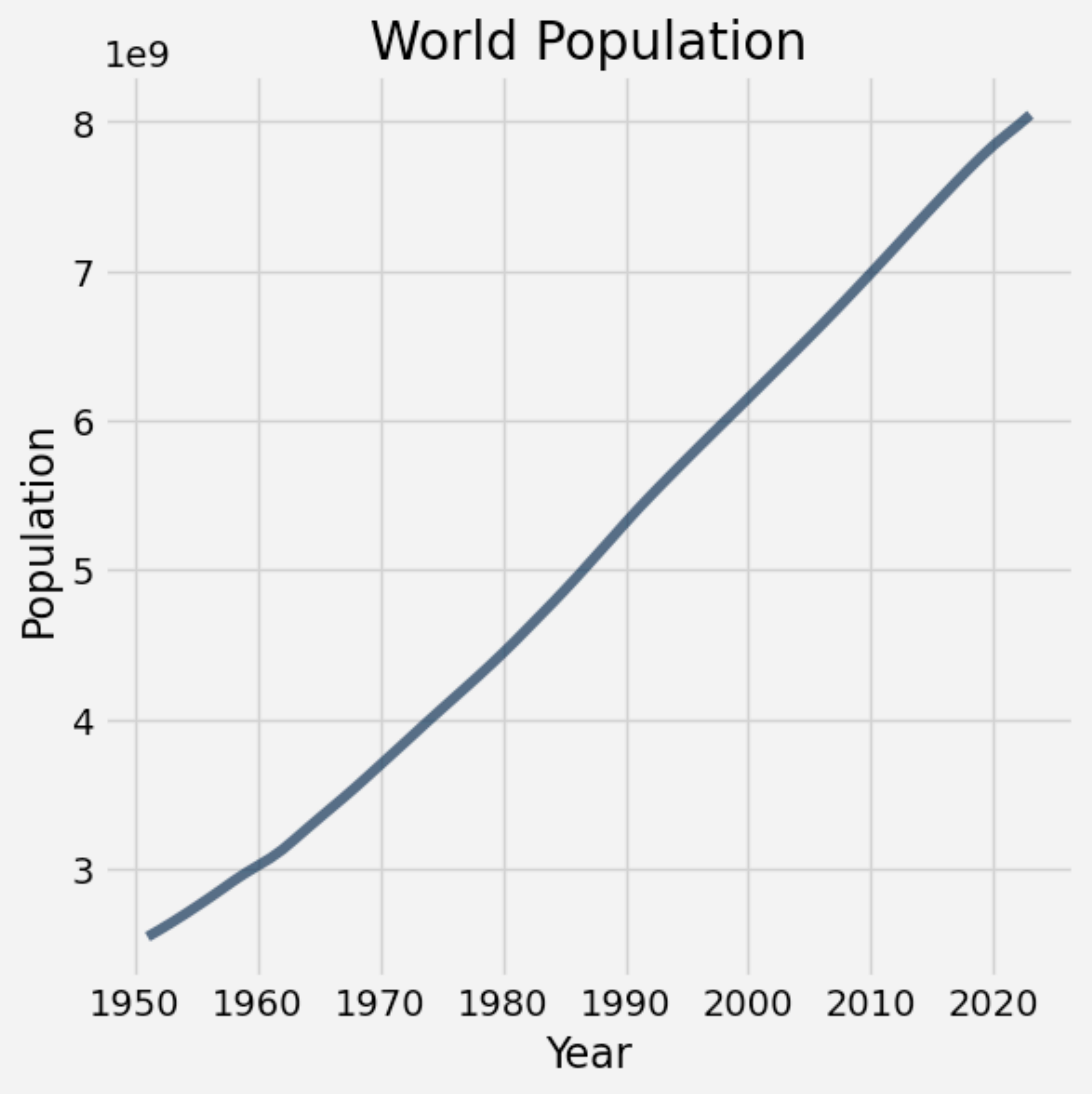
Values that are numbers are not necessarily numerical

- Sometimes people use numbers instead of strings to represent categories
- Example: In US census data, `SEX` code is (0, 1, 2)
  - Arithmetic on these “numbers” is meaningless
  - The variable `SEX` is still categorical even though numbers were used for the categories

Line and Scatter Plots

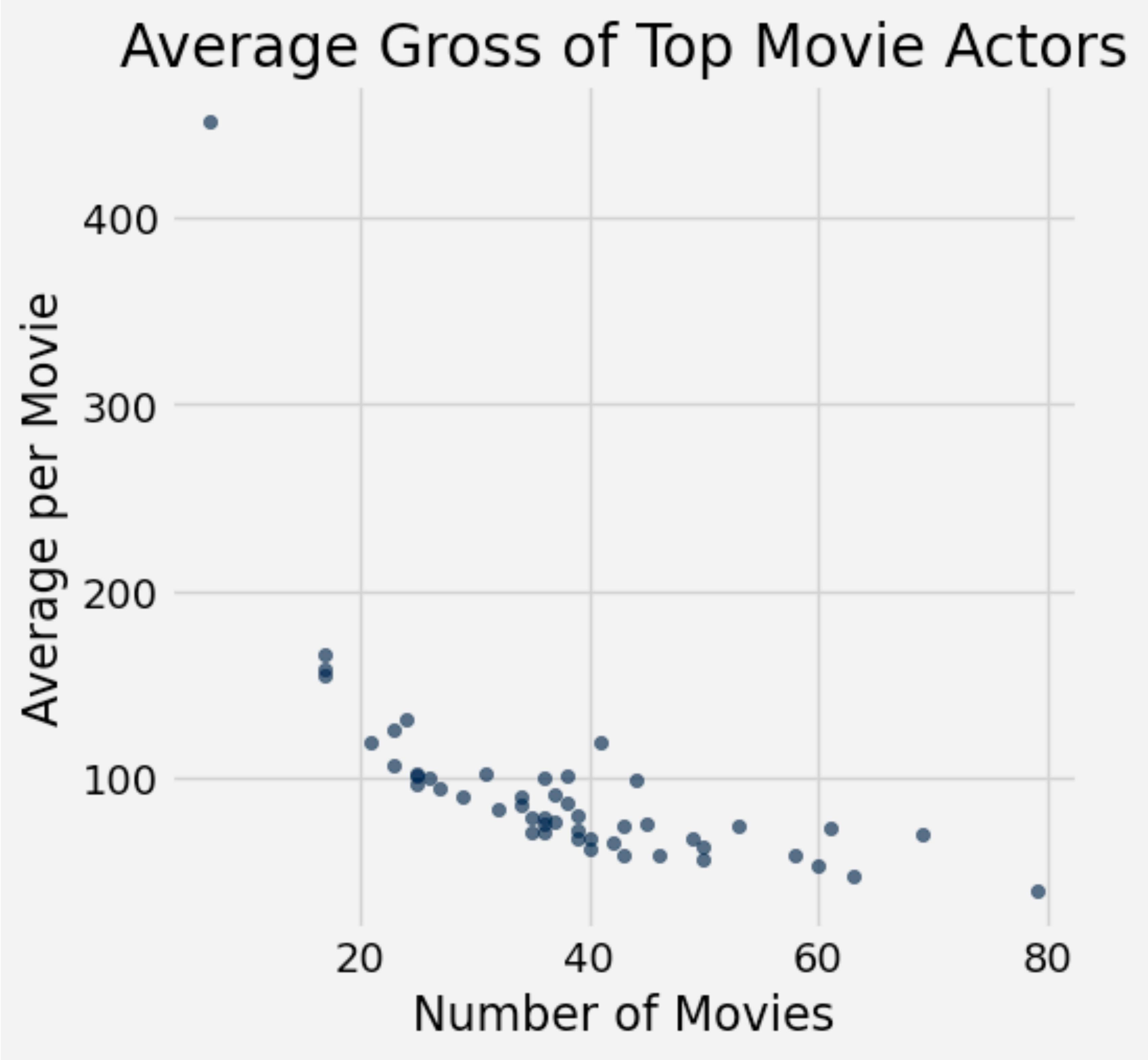
Line Plot

`plot`



Scatter Plot

`scatter`



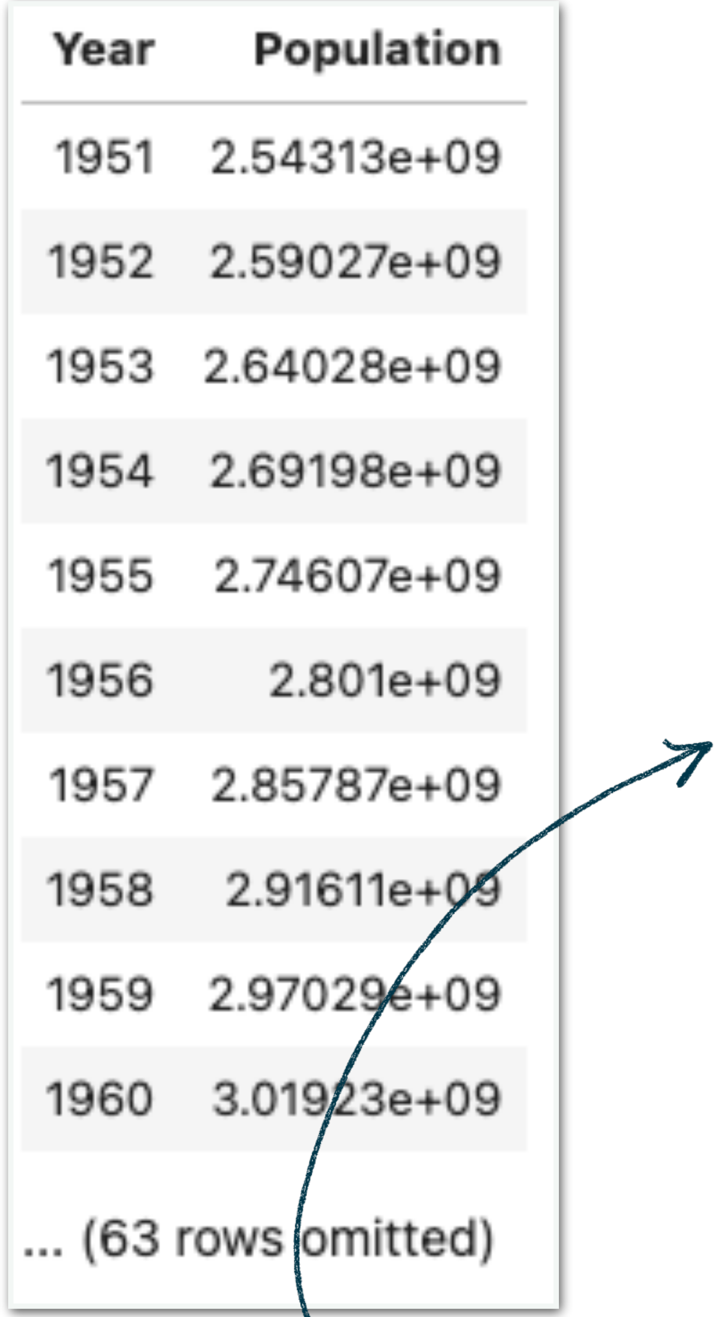
Line Plots

Line plots: good for sequential data if

- x-axis has an order (e.g., time, years, distance)
- sequential differences in y value are meaningful
- there’s only one y-value for each x-value

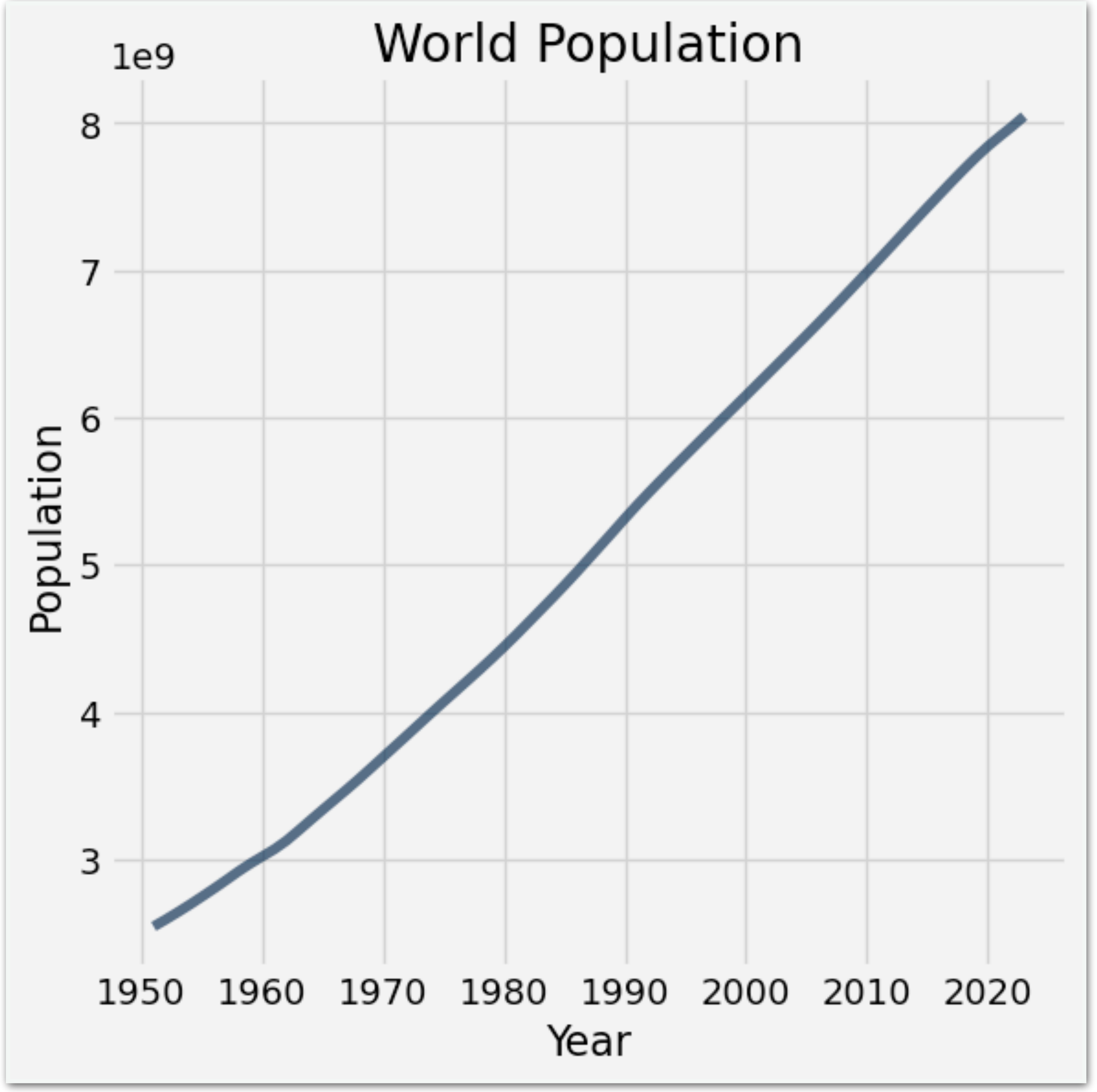
```
tbl.plot(x_axis, y_axis)
```

```
– pop_data.plot(‘Year’, ‘Population’)
```



Year	Population
1951	2.54313e+09
1952	2.59027e+09
1953	2.64028e+09
1954	2.69198e+09
1955	2.74607e+09
1956	2.801e+09
1957	2.85787e+09
1958	2.91611e+09
1959	2.97029e+09
1960	3.01923e+09

... (63 rows omitted)



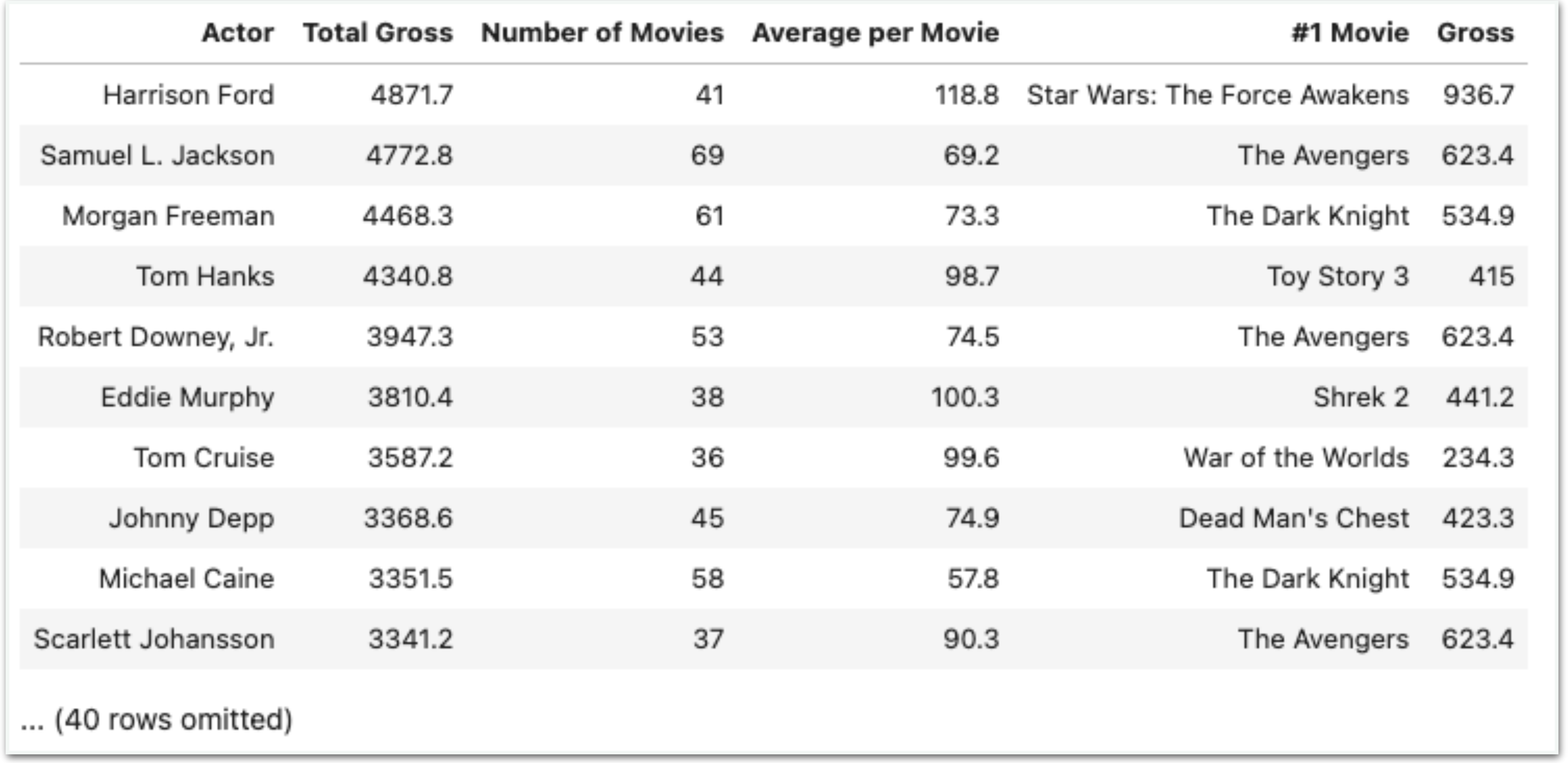
y-axis

x-axis

Scatter Plots

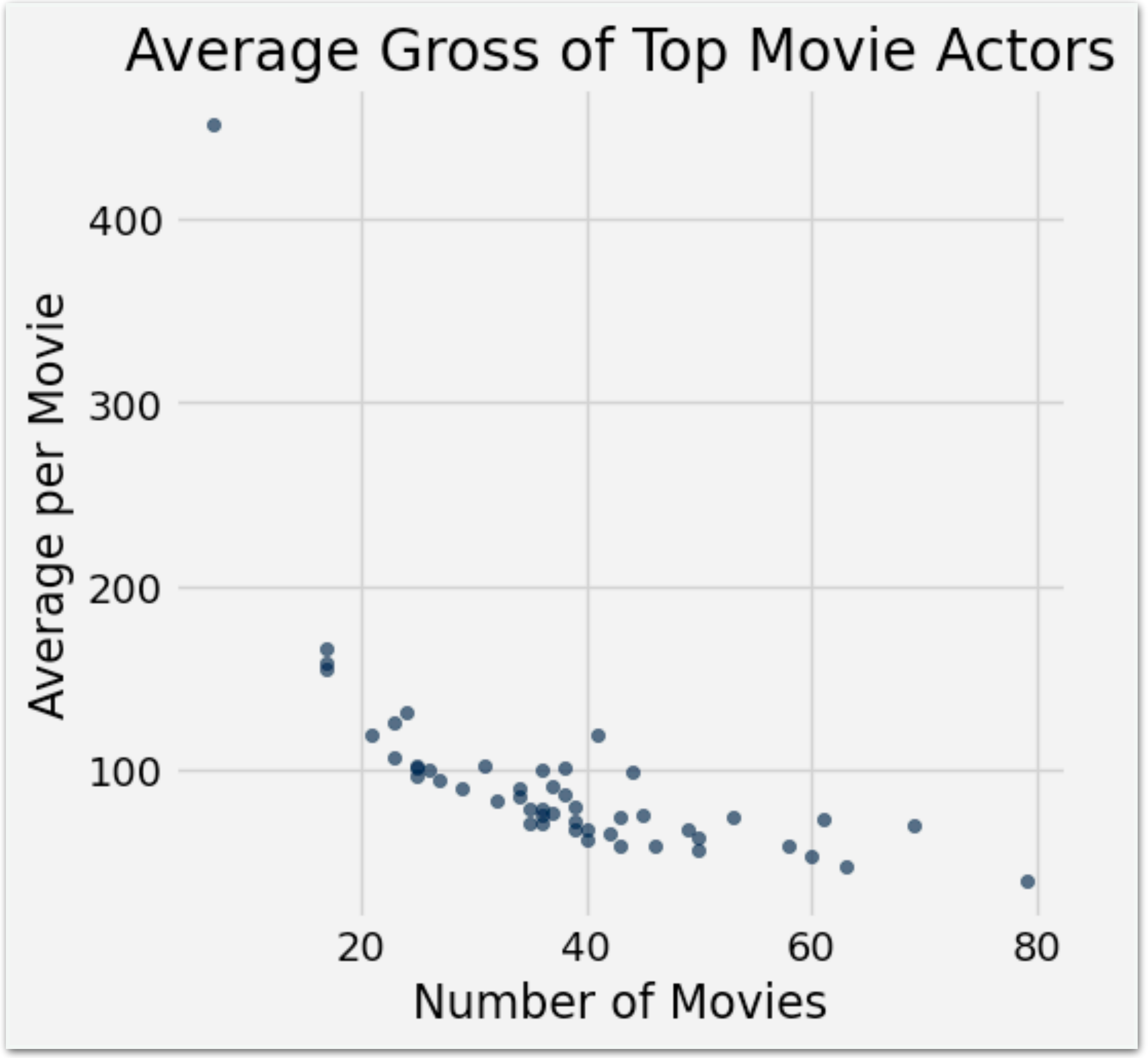
Scatter plots: good for non-sequential quantitative data

- Great for looking for associations



Actor	Total Gross	Number of Movies	Average per Movie	#1 Movie	Gross
Harrison Ford	4871.7	41	118.8	Star Wars: The Force Awakens	936.7
Samuel L. Jackson	4772.8	69	69.2	The Avengers	623.4
Morgan Freeman	4468.3	61	73.3	The Dark Knight	534.9
Tom Hanks	4340.8	44	98.7	Toy Story 3	415
Robert Downey, Jr.	3947.3	53	74.5	The Avengers	623.4
Eddie Murphy	3810.4	38	100.3	Shrek 2	441.2
Tom Cruise	3587.2	36	99.6	War of the Worlds	234.3
Johnny Depp	3368.6	45	74.9	Dead Man's Chest	423.3
Michael Caine	3351.5	58	57.8	The Dark Knight	534.9
Scarlett Johansson	3341.2	37	90.3	The Avengers	623.4

... (40 rows omitted)



```
tbl.scatter(x_axis, y_axis)
```

```
– actor.scatter(‘Number of Movies’, ‘Average per Movie’)
```

Line Plots vs Scatter Plots

- **Line plots** are good for sequential data if
  - x-axis has an order (e.g., time, years, distance)
  - sequential differences in y value are meaningful
  - there's only one y-value for each x-value
- Use **scatter plot** for non-sequential quantitative data
  - great for looking for associations

Next Class

- Today
  - Functions
  - Charts & Visualizations
- Monday
  - Histograms and Bar Charts