COMS BC1016

Introduction to Computational Thinking and Data Science

Lecture 4: Tables and Functions

February 2, 2026

Office Hours

Office hours begin 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

Homework

- HW 1 was released today
  - ZIP can be downloaded from the assignment page on Courseworks
  - Due next week Wednesday but can be submitted up to 5 days late (10% off per day)
- You'll be submitting your .ipynb file to Gradescope (via Courseworks)
  - If you run into technical issues with submission, you may email your assignment to me (along with a short explanation what's going wrong so we can fix it)
    - This only applies to HW 1 while we work out any technical issues

Course Outline

- **Exploration**
  - Introduction to Python
  - Working with data

Weeks 1-6

- **Inference**
  - Probability
  - Statistics

Weeks 7-11

- **Prediction**
  - Machine Learning
  - Regression and Classification

Weeks 12-14

Course Outline

- **Exploration** — **Weeks 1-6**
  - Discover patterns
  - Articulate insights
- **Inference** — **Weeks 7-11**
  - Make reliable conclusions about the world
  - Statistics is useful
- **Prediction** — **Weeks 12-14**
  - Informed guesses about unseen data!

Basics of Programming

Computational Thinking

- Apart from learning the syntax, programming requires thinking about how to formulate your task into steps your program can execute
- It helps to think about what basic operations do you know you can do
  - With numbers and arrays of numbers, you have basic arithmetic, computing the average, finding the max/min, …
  - With Tables, we can filter, sort, do basic array operations, …
  - As we do more examples, we'll see more operations. But for now, we work with what we have!
- With this in mind, break down the problem into smaller steps
  - Can I rewrite the task in terms of operations I know how to do?

Computational Thinking

Skyscraper Example: (we will do this in code later in the class)

- What city has the tallest steel building?

Computational Thinking

Skyscraper Example: (we will do this in code later in the class)

- What city has the tallest steel building?

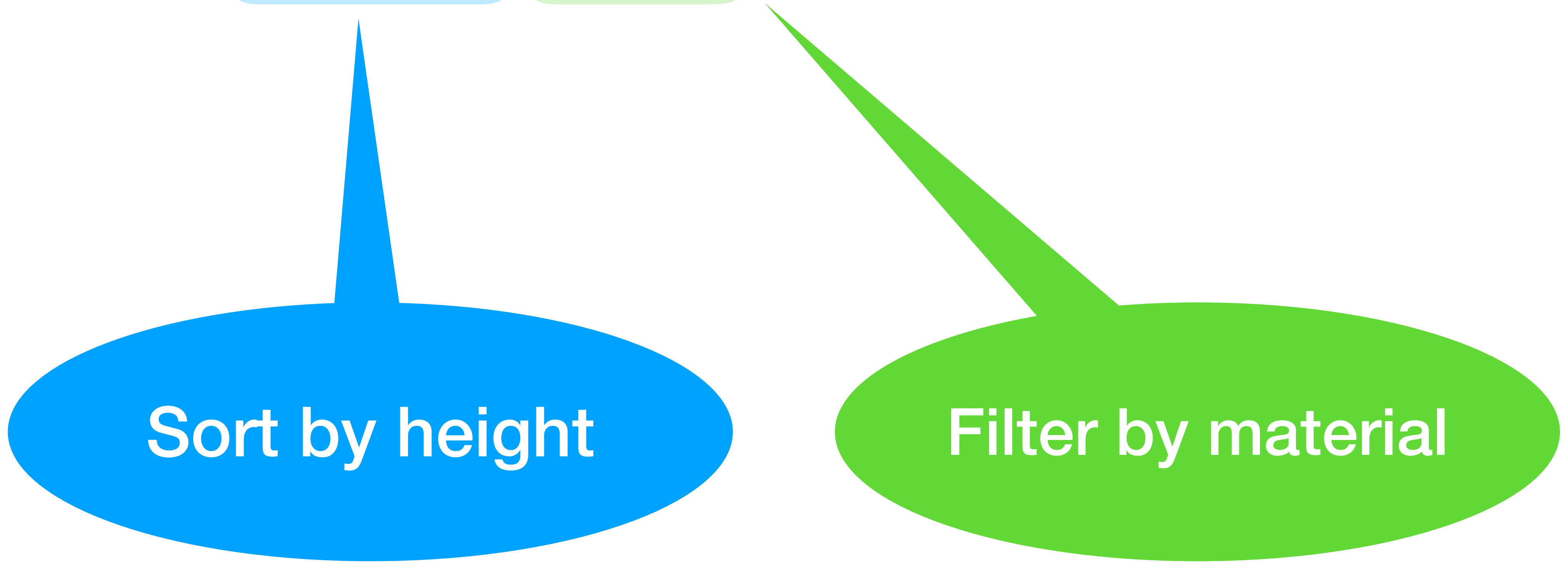
Sort by height

name	material	city	height	completed
One World Trade Center	mixed/composite	New York City	541.3	2014
Willis Tower	steel	Chicago	442.14	1974
432 Park Avenue	concrete	New York City	425.5	2015

Computational Thinking

Skyscraper Example: (we will do this in code later in the class)

- What city has the tallest steel building?



name	material	city	height	completed
~~One World Trade Center~~	~~mixed/composite~~	~~New York City~~	~~541.3~~	~~2014~~
Willis Tower	steel	Chicago	442.14	1974
~~432 Park Avenue~~	~~concrete~~	~~New York City~~	~~425.5~~	~~2015~~

Computational Thinking

Skyscraper Example: (we will do this in code later in the class)

- What city has the tallest steel building?

- Get the city
- Sort by height
- Filter by material

name	material	city	height	completed
~~One World Trade Center~~	~~mixed/composite~~	~~New York City~~	~~541.3~~	~~2014~~
Willis Tower	steel	Chicago	442.14	1974
~~432 Park Avenue~~	~~concrete~~	~~New York City~~	~~425.5~~	~~2015~~

Computational Thinking

- Computer programs will do *exactly* what you tell them to do
  - They can't anticipate what you *meant*, and they won't know to do anything that you don't explicitly tell them to do
    - For example, let's say you want your code to do different things based on the weather conditions of the day (e.g., rain, sun, clouds, snow, …). A very common mistake would be to write your code to assume there could only be a single weather condition. Your code won't account for multiple weather conditions unless it's explicitly told to.
  - If there is any ambiguity, it may choose for you (and not necessarily how you want it to) or it'll throw an error
- You also need to explicitly run any cell that you want Python to execute!

Bugs and Error Messages

- **Bugs** are unintended (typically bad) behavior
- Sometimes Python will catch mistakes and explicitly tell you in the form of an **error message**
  - The tricky ones are the ones that Python *doesn't* catch and you have to figure out yourself
- You typically look for bugs through **testing** (trying different inputs and seeing if your code both does what you want and doesn't do what you don't want)
- Bugs and errors are *extremely normal* and a huge part of programming is learning how to fix these!

Error Messages

Python will tell you **where** it ran into an issue

```
[1]: num_elem = 3
     numelem + 1

---------------------------------------------------------------------------
NameError                                 Traceback
Cell In[1], line 2
      1 num_elem = 3
----> 2 numelem + 1

NameError: name 'numelem' is not defined
```

Errors *also* have types!
Sometimes the names are descriptive, but other times you need to look up what it means

Python will also give you a short description of the problem

Error Messages

Sometimes the error messages may look complicated

```
# Why doesn't this line work?
# Hint: Look at what data type select returns!
np.average(skyscrapers.select('height'))
```

```
---------------------------------------------------------------------------
UFuncTypeError                            Traceback (most recent call last)
Cell In[18], line 3
      1 # Why doesn't this line work?
      2 # Hint: Look at what data type select returns!
----> 3 np.average(skyscrapers.select('height'))

File /opt/conda/lib/python3.12/site-packages/numpy/lib/function_base.py:520, in aver
age(a, axis, weights, returned, keepdims)
    517     keepdims_kw = {'keepdims': keepdims}
    519 if weights is None:
--> 520     avg = a.mean(axis, **keepdims_kw)
    521     avg_as_array = np.asanyarray(avg)
    522     scl = avg_as_array.dtype.type(a.size/avg_as_array.size)

File /opt/conda/lib/python3.12/site-packages/numpy/core/_methods.py:118, in _mean(a,
axis, dtype, out, keepdims, where)
    115         dtype = mu.dtype('f4')
    116         is_float16_result = True
--> 118 ret = umr_sum(arr, axis, dtype, out, keepdims, where=where)
    119 if isinstance(ret, mu.ndarray):
    120     with _no_nep50_warning():

UFuncTypeError: ufunc 'add' did not contain a loop with signature matching types (dt
ype('<U6'), dtype('<U6')) -> None
```

Error Messages

Sometimes the error messages may look complicated

```
# Why doesn't this line work?
# Hint: Look at what data type select returns!
np.average(skyscrapers.select('height'))
```

```
---------------------------------------------------------------------------
UFuncTypeError                            Traceback (most recent call last)
Cell In[18], line 3
      1 # Why doesn't this line work?
      2 # Hint: Look at what data type select returns!
----> 3 np.average(skyscrapers.select('height'))

File /opt/conda/lib/python3.12/site-packages/numpy/li...ion_base.py:520, in aver
age(a, axis, weights, returned, keepdims)
    517     keepdims_kw = {'keepdims': keepdims}
    519 if weights is None:
--> 520     avg = a.mean(axis, **keepdims_kw)
    521     avg_as_array = np.asanyarray(av
    522     scl = avg_as_array.dtype.type(

File /opt/conda/lib/python3.12/site-package
axis, dtype, out, keepdims, where)
    115         dtype = mu.dtype('f4')
    116         is_float16_result = True
--> 118 ret = umr_sum(arr, axis, dtype, out, keepdims, where=where)
    119 if isinstance(ret, mu.ndarray):
    120     with _no_nep50_warning():

UFuncTypeError: ufunc 'add' did not contain a loop wi
ype('<U6'), dtype('<U6')) -> None
```

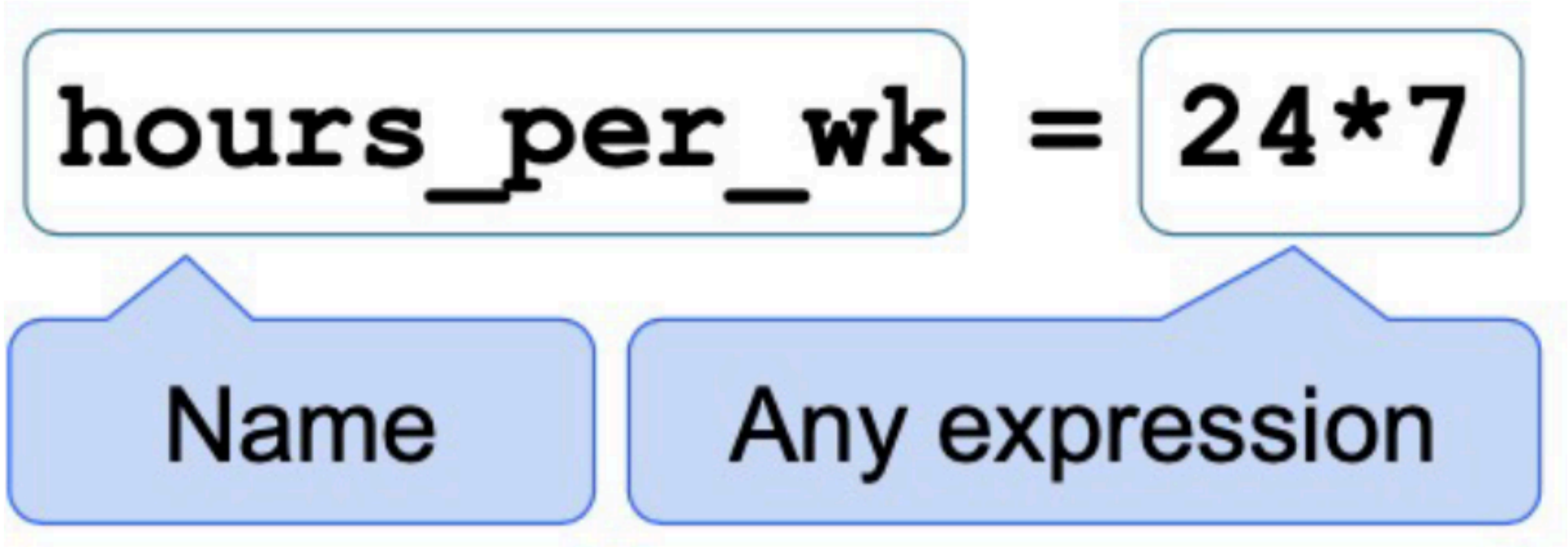
When in doubt, look at the line that caused the error

Walk through each part of the expression.
What does `skyscrapers.select('height')` do?
What type does it return?
What type is `np.average` expecting?

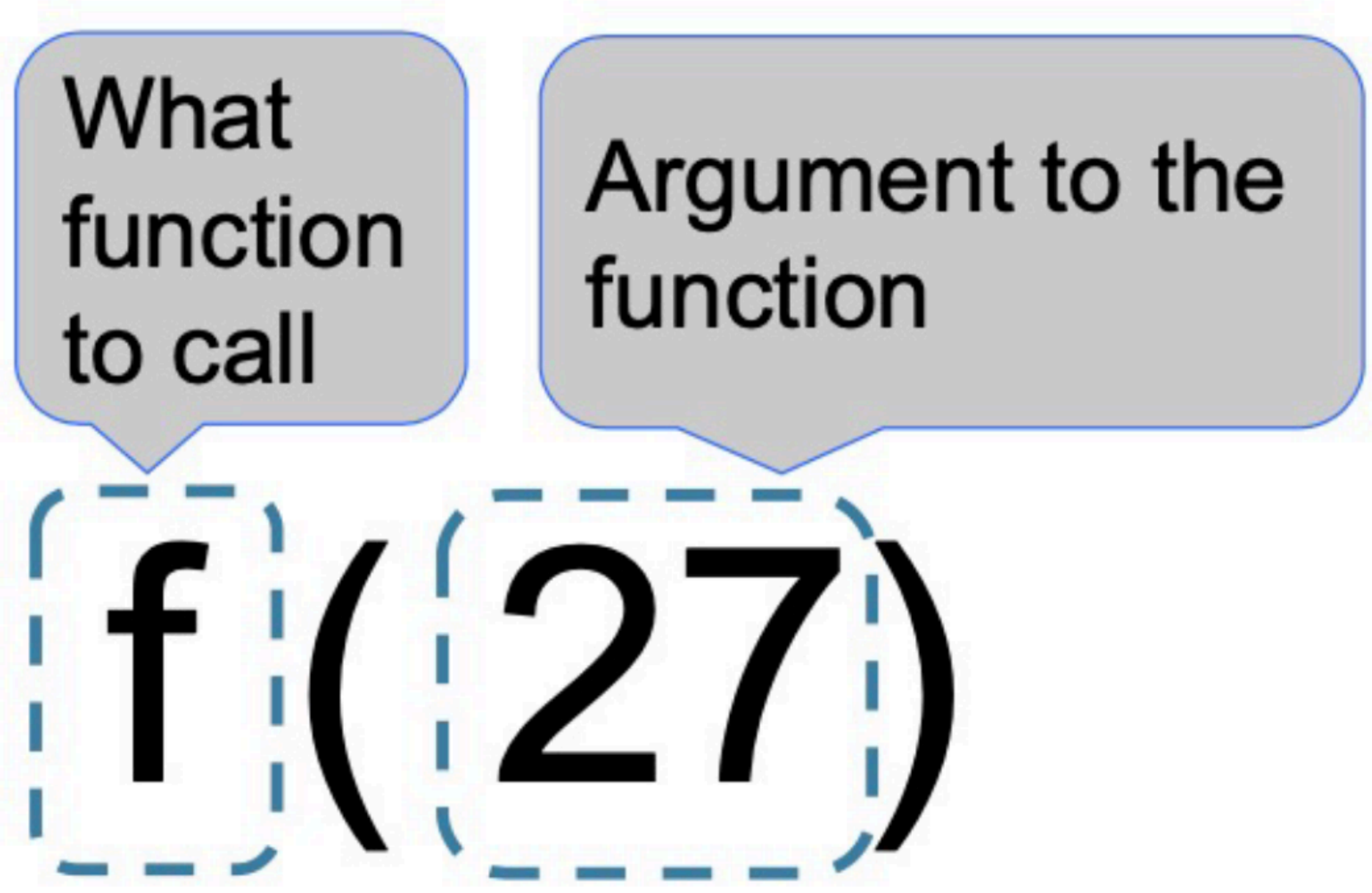
This is when it may be helpful to **test** each component separately and check that it's doing what you expect!

Assignment Statements

- **Expressions** evaluate to a result
- **Statements** perform an action
- Assignment statement changes the meaning of the name to the left of the = symbol
  - The name is bound to a value

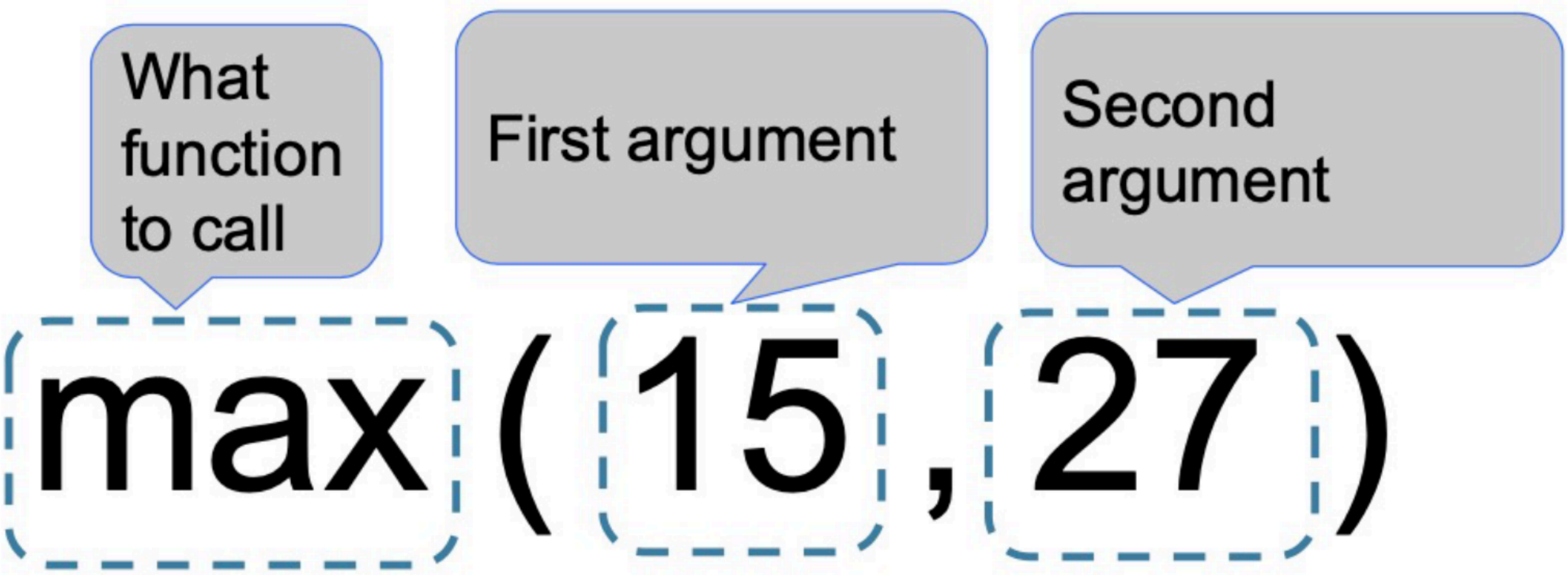


Anatomy of a Call Expression (Functions)



"Call f on 27."

Anatomy of a Call Expression (Functions)



Order of Operations

- Python typically evaluates in order of left-to-right and follows PEDMAS (parenthesis, exponentiation, division/multiplication, addition, subtraction)

- Anything inside parenthesis will be evaluated first (left-to-right)

    - If you're uncertain about order of operations and what to ensure an operation occurs in the order you intend, you can always add a parenthesis around the expression!

Tables

Tables

A **table** is a way of representing data sets, they're a sequence of labeled columns

- Each **row** is an **individual**

- Each **column** is an **attribute** of the individual

In this class, our columns will consist of a header name (string) and an array of values

Name	Age	Coloring	Favorite Food
Gertrude	15 yrs	Tuxedo	Milk
Ruby	14 yrs	Tuxedo	Potato chips
Corina	6 yrs	Dilute Tortoiseshell	Kibble
Frito	1 yr	Tabby	Cheese

Creating a `datascience` Tables

- Read from a CSV file
  - `Table.read_table(filename)`
- Create an empty table using `Table()`
  - Add elements to the Table using `.with_column`

Name	Description	Input	Output
`Table()`	Create an empty table, usually to extend with data (Ch 6)	None	An empty **Table**
`Table().read_table(filename)`	Create a table from a data file (Ch 6)	**string**: the name of the file	**Table** with the contents of the data file
`tbl.with_columns(name, values)` `tbl.with_columns(n1, v1, n2, v2,...)`	A table with an additional or replaced column or columns. `name` is a string for the name of a column, `values` is an array (Ch 6)	1. **string**: the name of the new column; 2. **array**: the values in that column	**Table**: a copy of the original Table with the new columns added

Creating `datascience` Tables

Create an empty table using `Table()`

Each column of a table is an array and `with_columns` creates a table with the array of values as a new column

```
Table().with_columns("Name", make_array("Gertrude",
"Ruby", "Corina", "Frito"))
```

Creating `datascience` Tables

`Table()` **creates an empty table**

`.with_columns()` adds a column

The first argument to `.with_columns` is the column name

ble us

Each olumn of a table is an a ay and `with_columns` cre y of value as a new column

```
Table().with_columns("Name", make_array("Gertrude",
"Ruby", "Corina", "Frito"))
```

… Followed by an array with the column values

Creating `datascience` Tables

`Table()` creates an empty table

`.with_columns()` adds a column

...ble us...

Each ...olumn of a table is an a... ...ay and `with_columns` cre... ...y of value... as a new column

The first argument to `.with_columns` is the column name

```
Table().with_columns("Name", make_array("Gertrude",
"Ruby", "Corina", "Frito"))
```

… Followed by an array with the column values

Name
Gertrude
Ruby
Corina
Frito

Creating `datascience` Tables

Create an empty table using `Table()`

Each column of a table is an array and `with_columns` creates a table with the array of values as a new column

```
Table().with_columns("Name", make_array("Gertrude",
"Ruby", "Corina", "Frito"))
```

Name
Gertrude
Ruby
Corina
Frito

Creating `datascience` Tables

Create an empty table using `Table()`

Each column of a table is an array and `with_columns` creates a table with the array of values as a new column

```
Table().with_columns(“Name”, make_array(“Gertrude”,
“Ruby”, “Corina”, “Frito”),
“Age”, make_array(15,14,6,1))
```

We can add more columns with a comma and following this same pattern

Name	Age
Gertrude	15
Ruby	14
Corina	6
Frito	1

More Ways to Create Tables

Create a new table from an existing table. Let `tbl` be a table and `c`, `c1`, `c2` be column names or indices

- Create a table with only the specified columns
  `tbl.select(c1 ,c2, …)`

- Copy the original table but *without* specified columns
  `tbl.drop(c1, c2, …)`

- Copy the original table but only with individuals in specified rows
  `tbl.take(row_indices)`

Note that all of these produce `Table`s

More Ways to Create Tables

Create a new table from an existing table. Let `tbl` be a table and `c`, `c1`, `c2` be column names or indices

- Copy the original table but sorted by column `c`

```
tbl.sort(c[, descending=False])
```

- Copy the original table but only with individuals where their value in `c` meets some predicate

```
tbl.where(c, predicate)
```

Note that all of these produce `Table`s

Filtering

https://www.data8.org/sp22/python-reference.html

`Table.where` Predicates

Any of these predicates can be negated by adding `not_` in front of them, e.g. `are.not_equal_to(Z)` or `are.not_containing(S)`.

Predicate	Description
`are.equal_to(Z)`	Equal to `Z`
`are.not_equal_to(Z)`	Not equal to `Z`
`are.above(x)`	Greater than `x`
`are.above_or_equal_to(x)`	Greater than or equal to `x`
`are.below(x)`	Less than `x`
`are.below_or_equal_to(x)`	Less than or equal to `x`
`are.between(x,y)`	Greater than or equal to `x` and less than `y`
`are.between_or_equal_to(x,y)`	Greater than or equal to `x`, and less than or equal to `y`
`are.contained_in(A)`	Is a substring of `A` (if `A` is a string) or an element of `A` (if `A` is a list/array)
`are.containing(S)`	Contains the string `S`
`are.strictly_between(x,y)`	Greater than `x` and less than `y`

Table Methods

Recall each column in a Table is an array

- `column` takes a label or index and returns an **array**

  `tbl.column(c)`

- Array methods work on data in the columns
  - e.g., `sum`, `min`, `max`, `average`

A Useful Table Method: `group`

`group` counts the number of rows of each category in a column

- Optionally takes in a function as a second argument and applies to other columns

```
chess_games.group('winner')
```

winner	count
black	9107
draw	950
white	10001

```
wins_and_moves = chess_games.select('victory_status','turns')
wins_and_moves.group('victory_status', max)
```

victory_status	turns max
draw	259
mate	222
outoftime	349
resign	218

Operating on Tables

Organize table entries by values in column `c`:

- `tbl.group(c)`
- `tbl.group(c, func)`

Apply a function `func` to all entries in a column `c`:

- `tbl.apply(func, c)`

Skyscraper Exercise (filter, sort array operations)

We’re going to try to answer some questions using the our dataset on skyscrapers in the US

1. What’s the tallest building in Los Angeles?

2. What city has the tallest steel building?

3. Which type of construction (concrete, mixed/composite, or steel) has the highest average skyscraper height?

4. What’s the tallest building completed in the year you were born?

Skyscraper Exercise (filter, sort array operations)

We're going to try to answer some questions using the our dataset on skyscrapers in the US

1. What's the tallest building in Los Angeles?
2. What city has the tallest steel building?
3. Which type of construction (concrete, mixed/composite, or steel) has the highest average skyscraper height?
4. What's the tallest building completed in the year you were born?

Skyscraper Exercise (filter, sort array operations)

We’re going to try to answer some questions using the our dataset on skyscrapers in the US

1. What’s the tallest building in Los Angeles?
2. What city has the tallest steel building?
3. Which … ction (concret… site, or steel) has the highest a… scraper height?
4. What’s the tallest building completed in the year you were born?

Sort by height

Filter by location

Skyscraper Exercise (filter, sort array operations)

We're going to try to answer some questions using the our dataset on skyscrapers in the US

1. What's the tallest building in Los Angeles?
2. What city has the tallest steel building?
3. Which type of construction (concrete, mixed/composite, or steel) has the highest average skyscraper height?
4. What's the tallest building completed in the year you were born?

Skyscraper Exercise (filter, sort array operations)

We're going to try to answer some questions using the our dataset on skyscrapers in the US

1. What's the tallest building in Los Angeles?
2. What city has the tallest steel building?
3. Which type of construction (co /composite, or steel) has the highest average skyscraper hei
4. What's the ta mpleted in the year you were born?

Filter by material

Sort by height

Get the city

Skyscraper Exercise (filter, sort array operations)

We're going to try to answer some questions using the our dataset on skyscrapers in the US

1. What's the tallest building in Los Angeles?

2. What city has the tallest steel building?

3. Which type of construction (concrete, mixed/composite, or steel) has the highest average skyscraper height?

4. What's the tallest building completed in the year you were born?

Skyscraper Exercise (filter, sort array operations)

We're going to try to answer some questions using the our dataset on skyscrapers in the US

Group by type and compute
the average height of each type

1. What's
2. What city has the tallest steel building?
3. Which type of construction (concrete, mixed/composite, or steel) has the highest average skyscraper height?
4. What's the tallest building completed in the year you were born?

Sort by height

Skyscraper Exercise (filter, sort array operations)

We're going to try to answer some questions using the our dataset on skyscrapers in the US

1. What's the tallest building in Los Angeles?
2. What city has the tallest steel building?
3. Which type of construction (concrete, mixed/composite, or steel) has the highest average skyscraper height?
4. What's the tallest building completed in the year you were born?

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!

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

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

```
True
```

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

```
False
```

Example: Prof Lee's Cat Census

Professor Lee is in a cat picture group chat. She has 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

Attribute Types

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

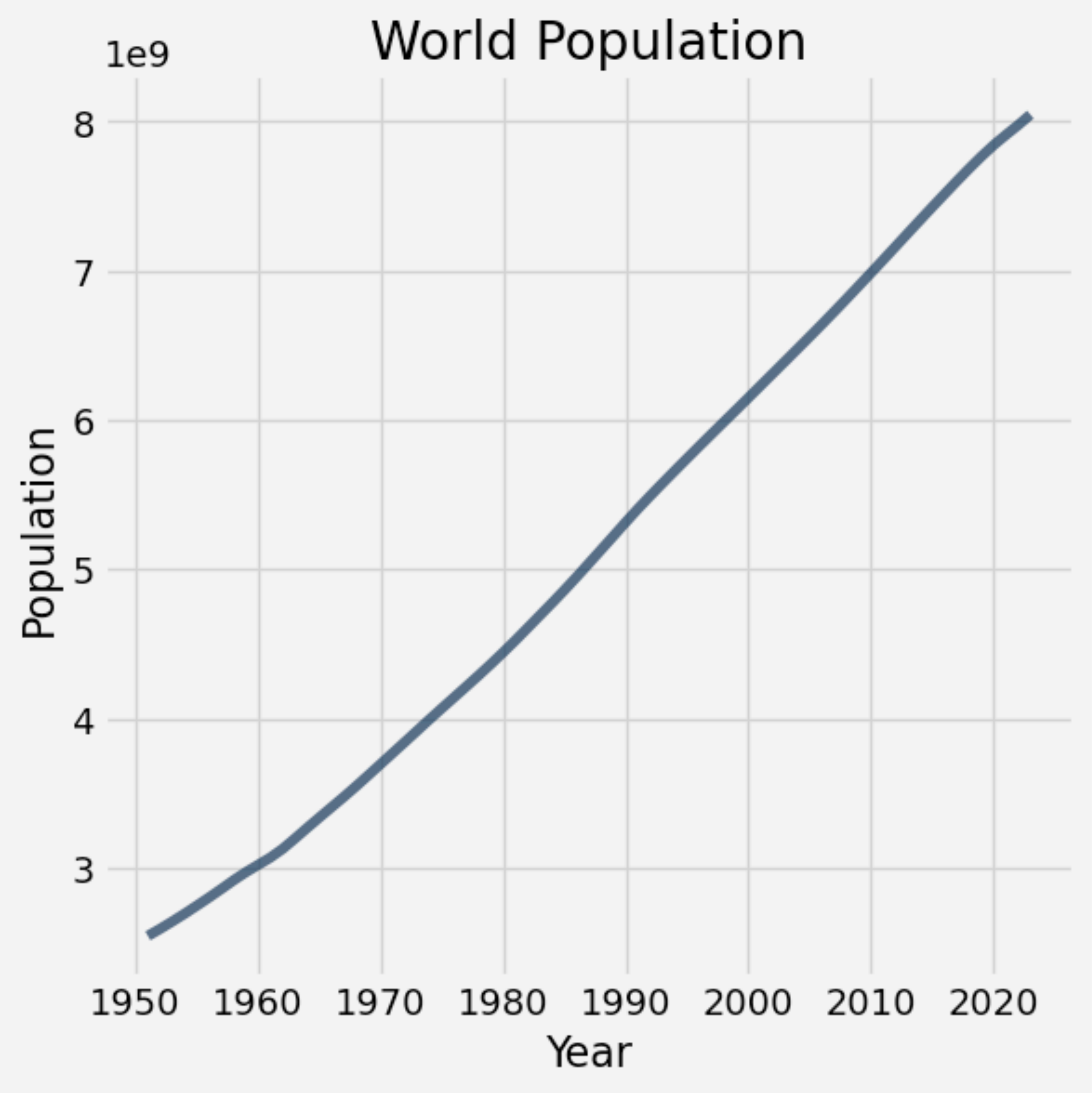
Values that are numbers are not necessarily numerical

- 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

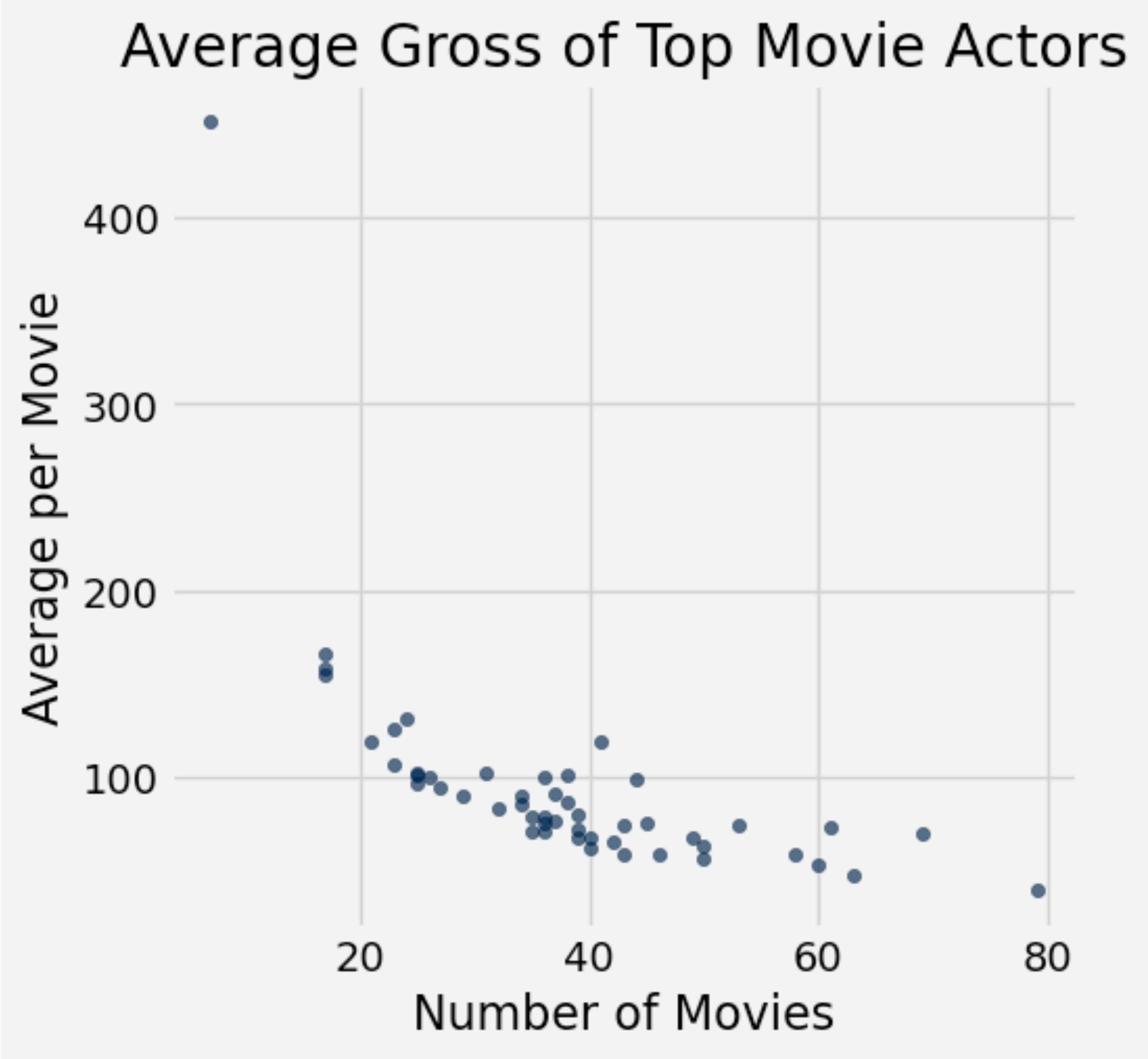
Line Plot

`plot`



Scatter Plot

`scatter`



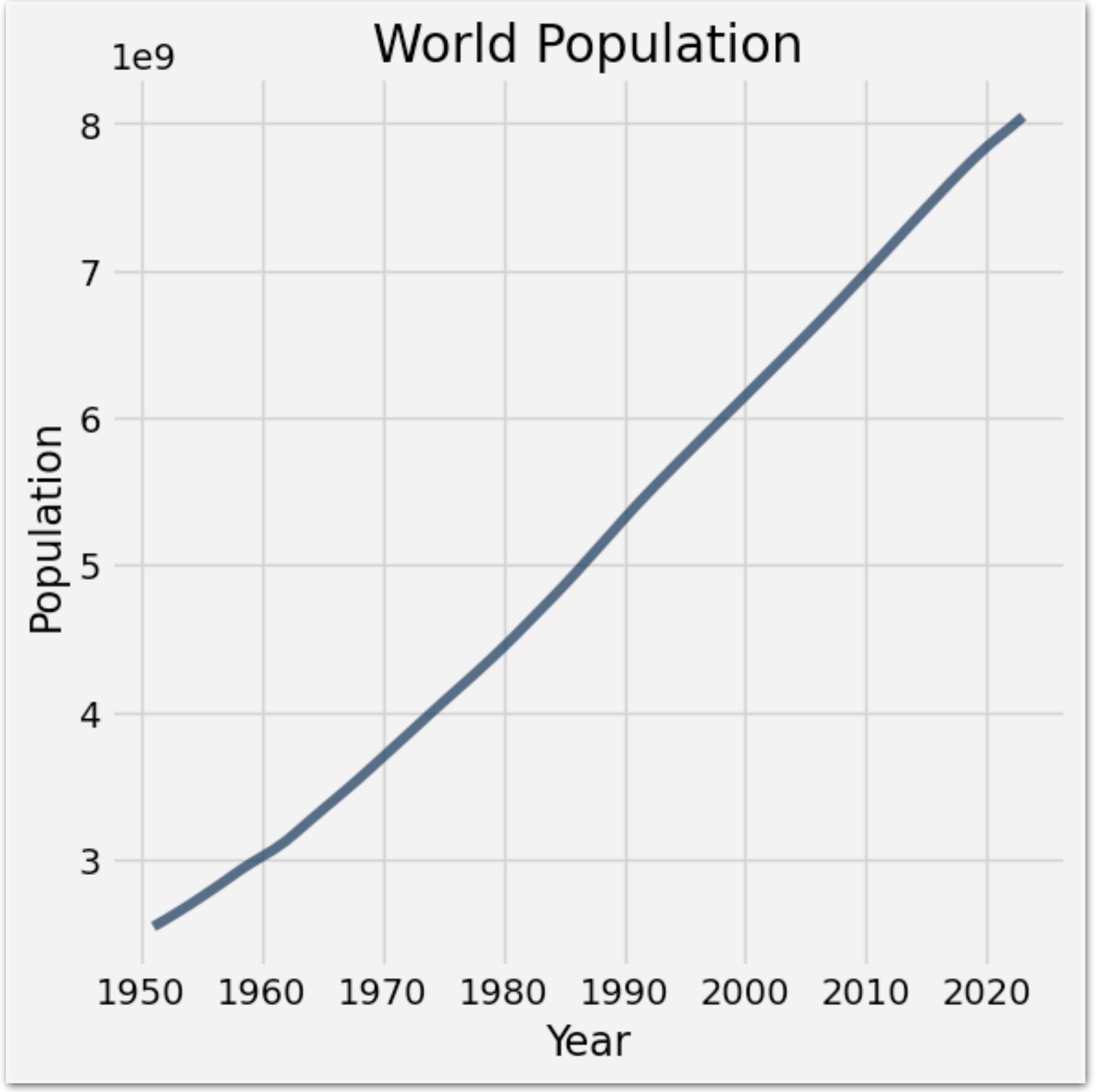
Line Plots

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

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)




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

```
- pop_data.plot('Year', 'Population')
```

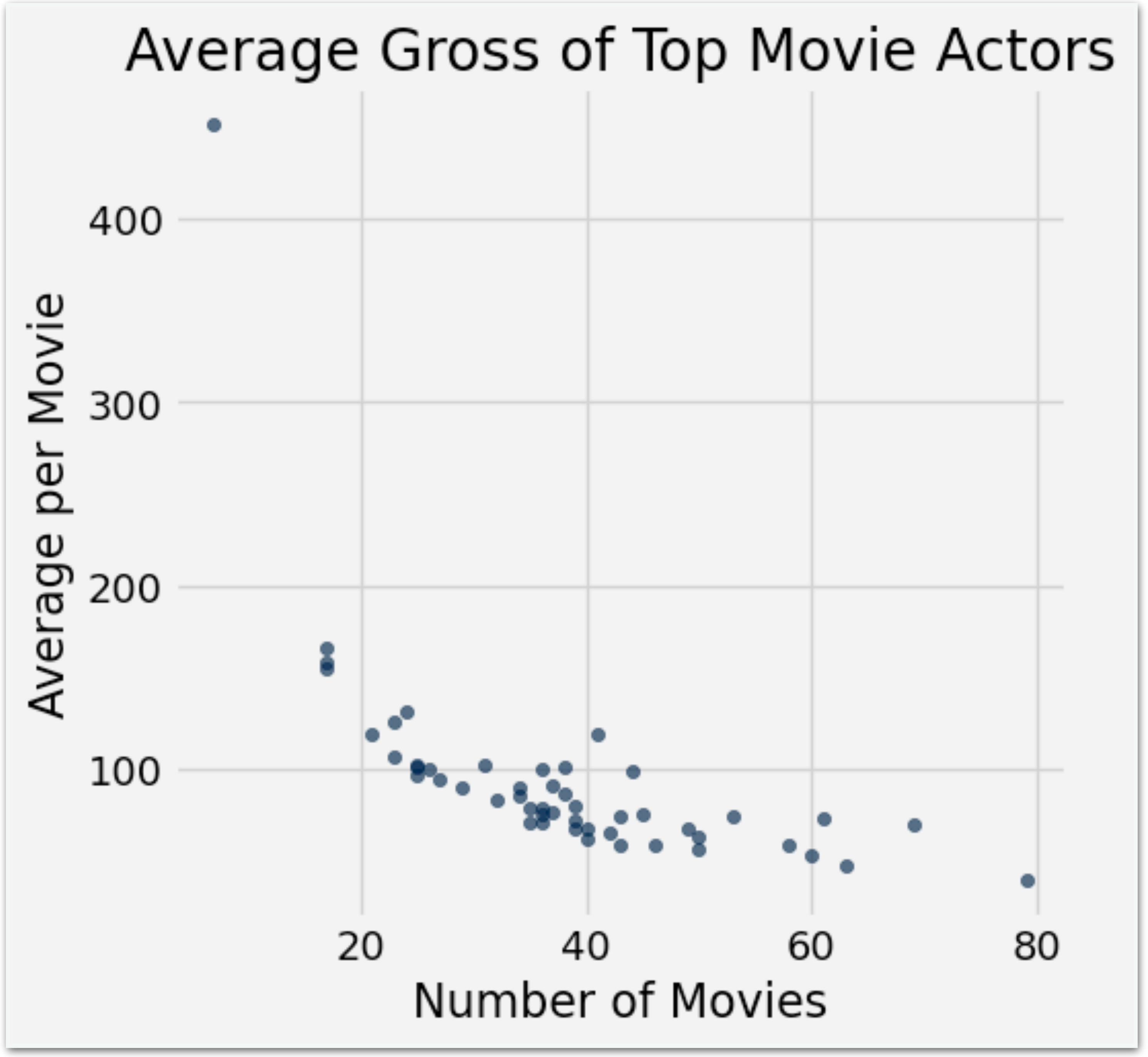
Scatter Plots

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 (HW 1 Released)
  - Tables (Part 2)
- Wednesday
  - Charts & Visualization