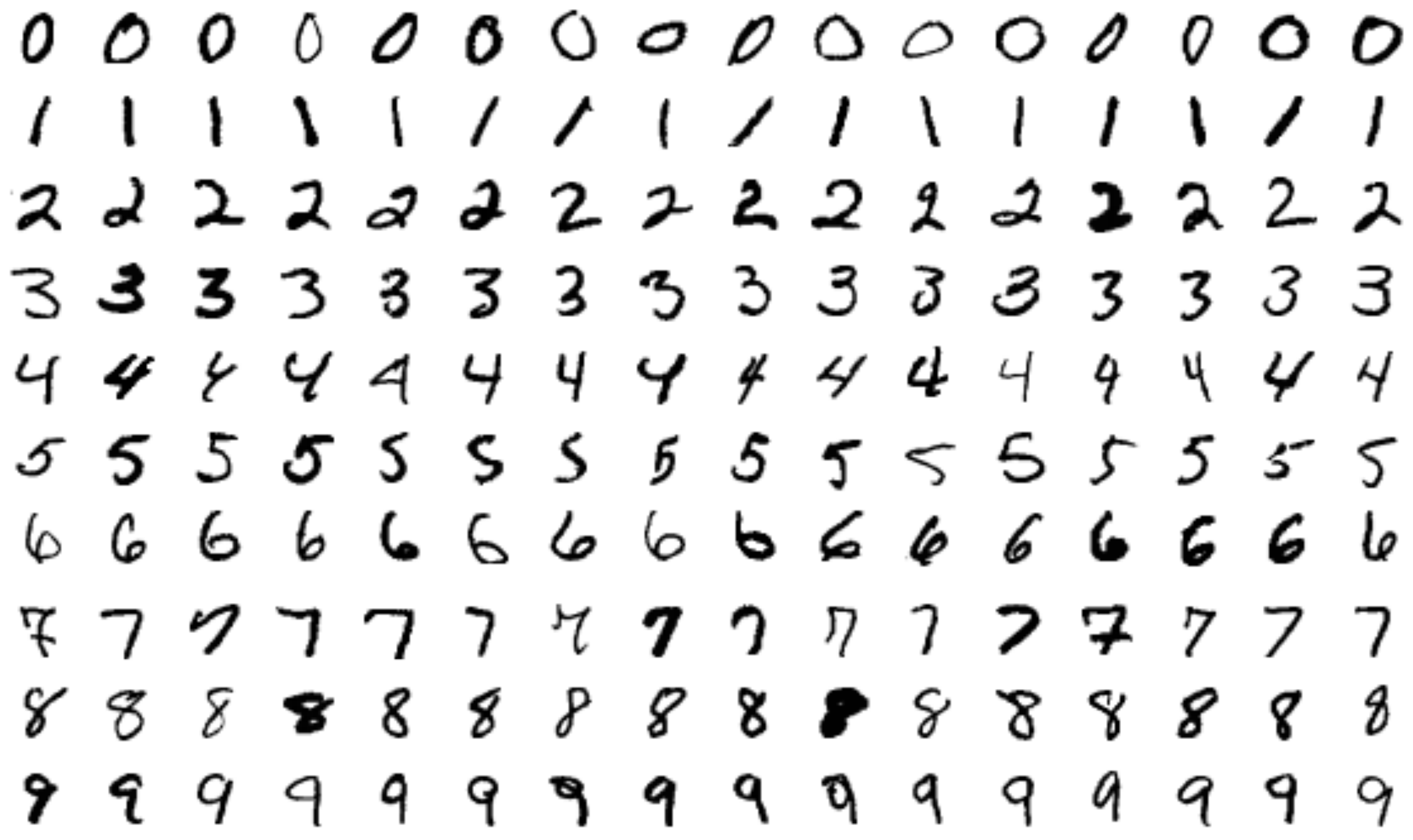
Programming Principles in Python (CSCI 503/490)

Review

Dr. David Koop

Tasks Machine Learning can Help With

- Identifying the zip code from handwritten digits on an envelope

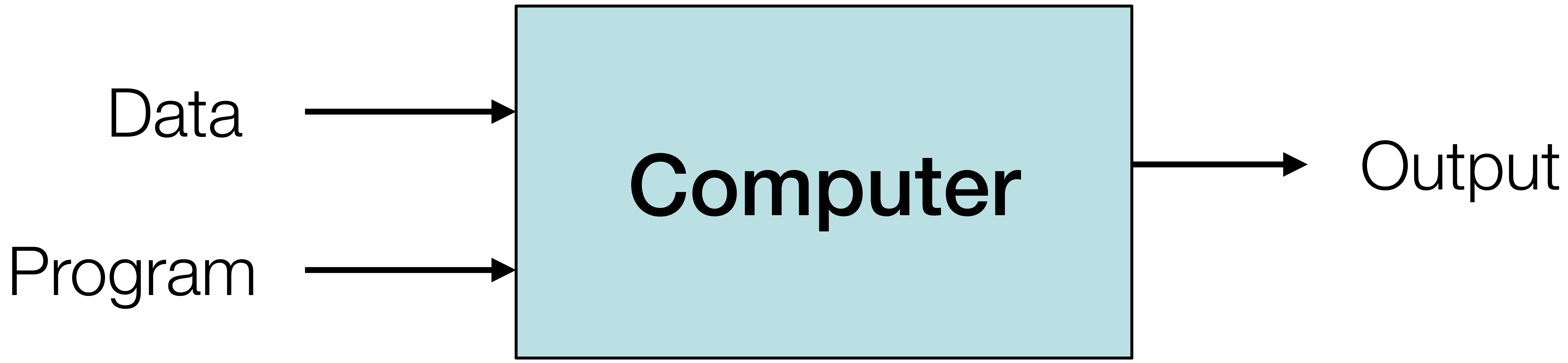


- Detecting fraudulent activity in credit card transactions
- Identifying topics in a set of blog posts
- Grouping customers with similar preferences

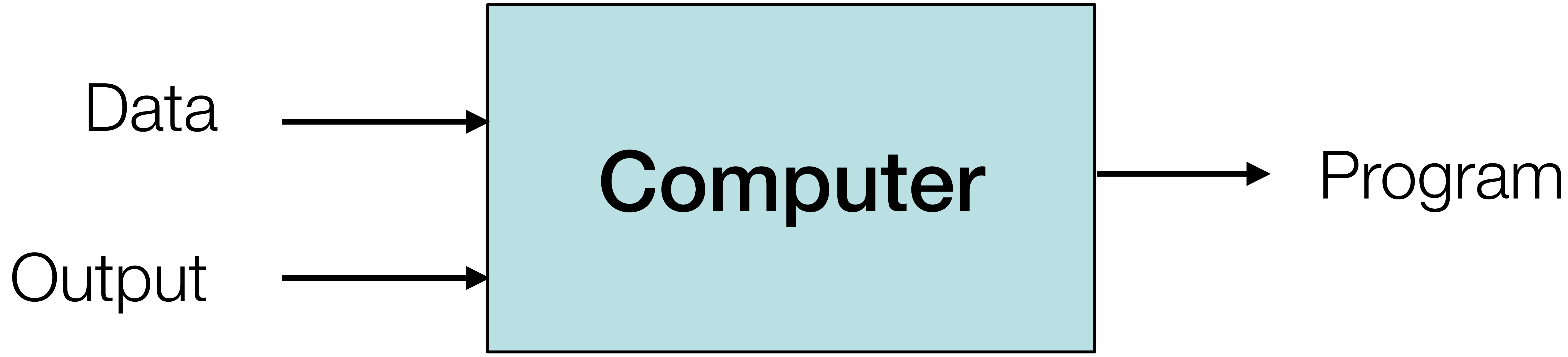
[A. Müller & S. Guido, Introduction to Machine Learning with Python, J. Steppan (MNIST image)]

Machine Learning

- Traditional Programming



- Machine Learning



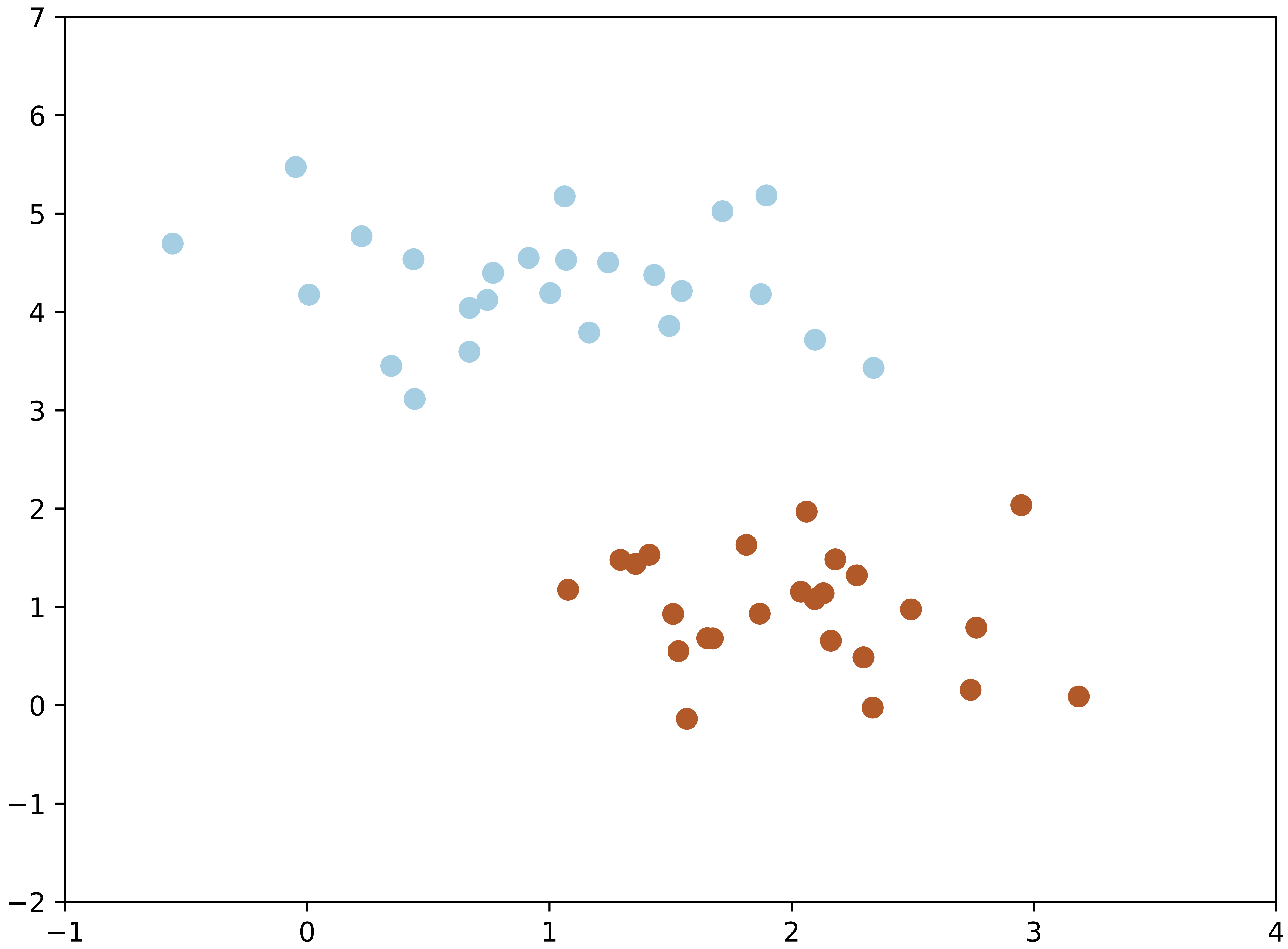
[P. Domingos]

Types of Learning

- Supervised (inductive) learning
  - Training data includes desired outputs
- Unsupervised learning
  - Training data does not include desired outputs
- Semi-supervised learning
  - Training data includes a few desired outputs
- Reinforcement learning
  - Rewards from sequence of actions

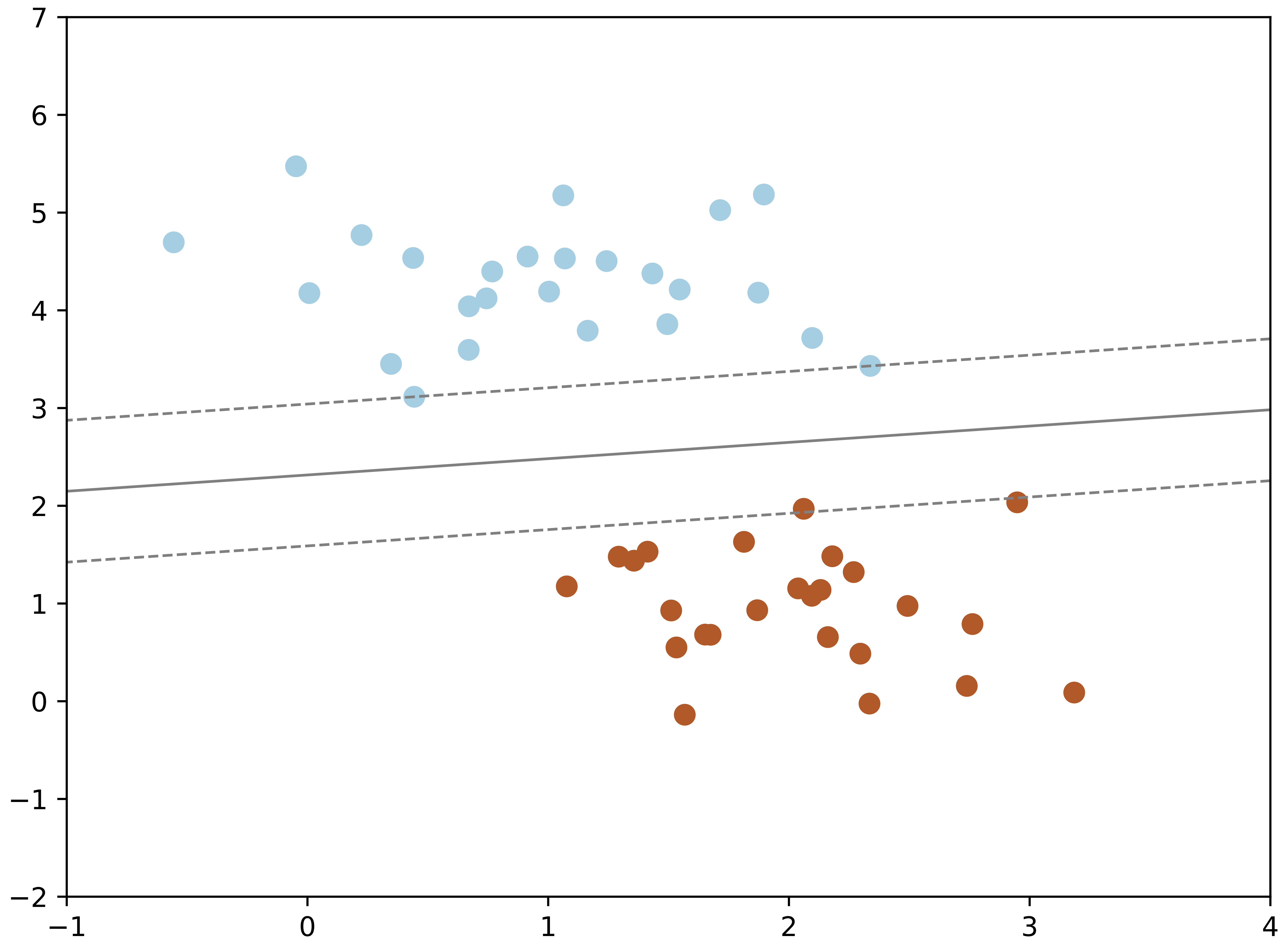
[P. Domingos]

Supervised Learning



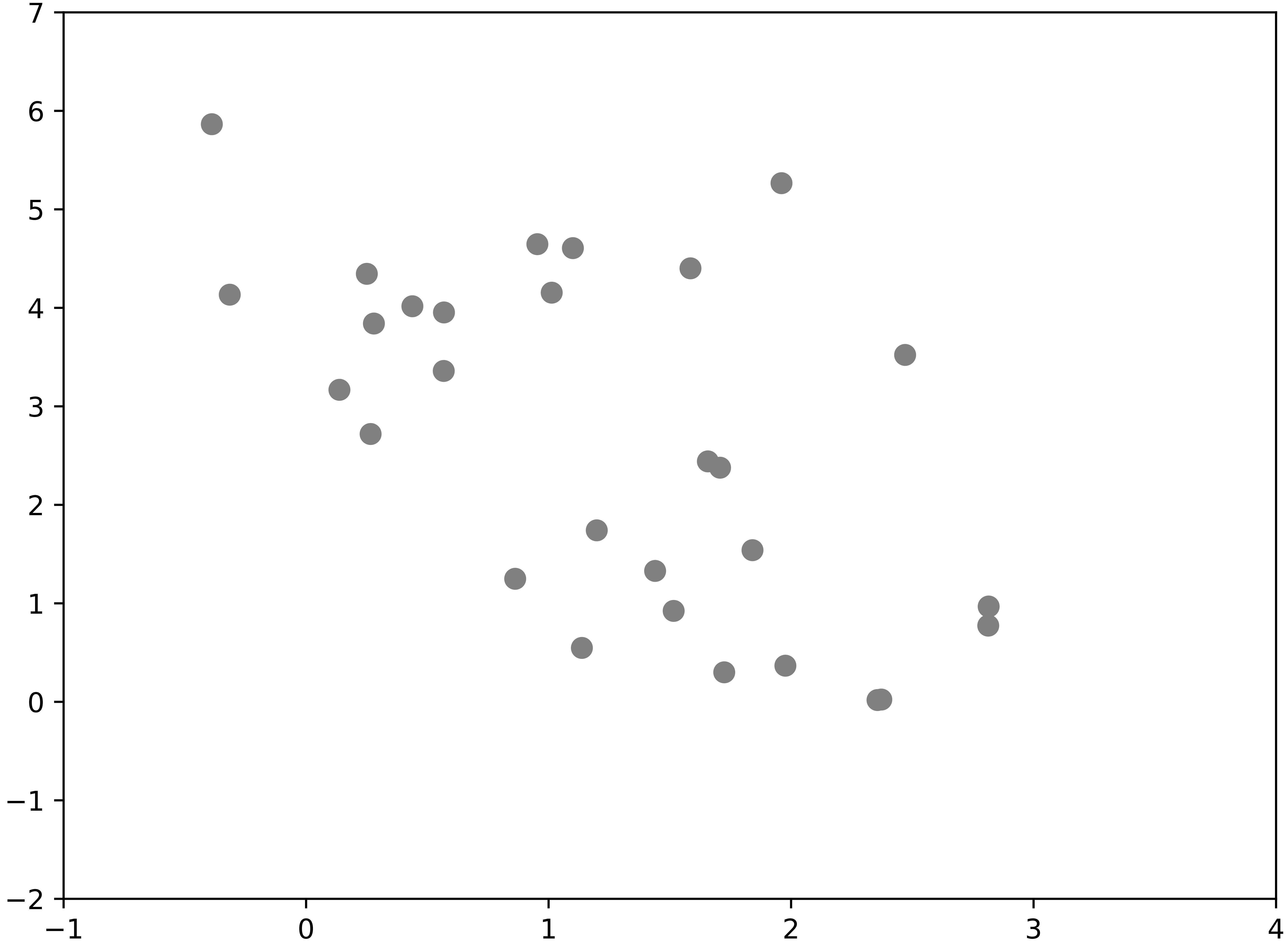
[J. VanderPlas]

Supervised Learning: Learned Algorithm (Fit)



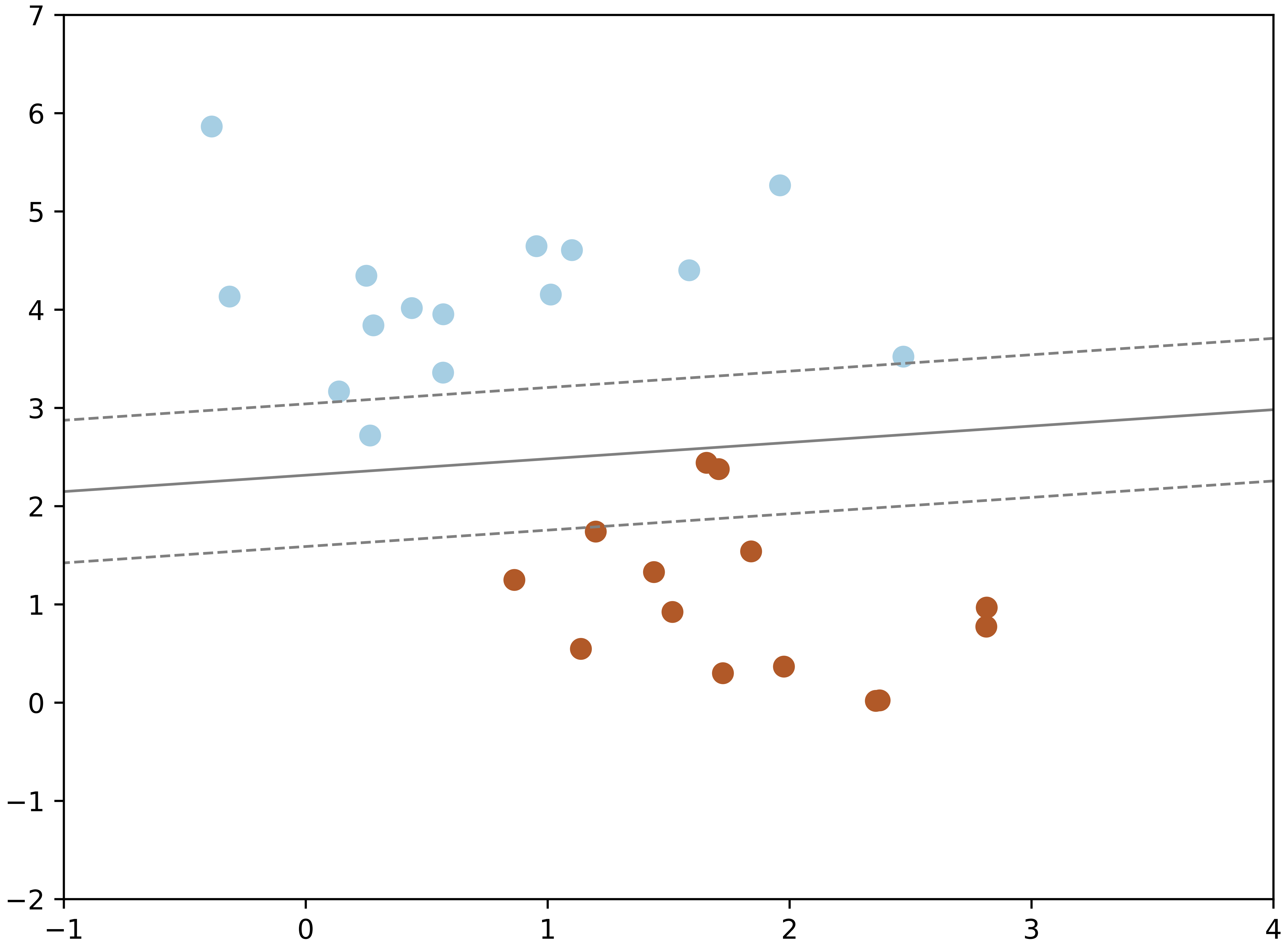
[J. VanderPlas]

Supervised Learning: Prediction



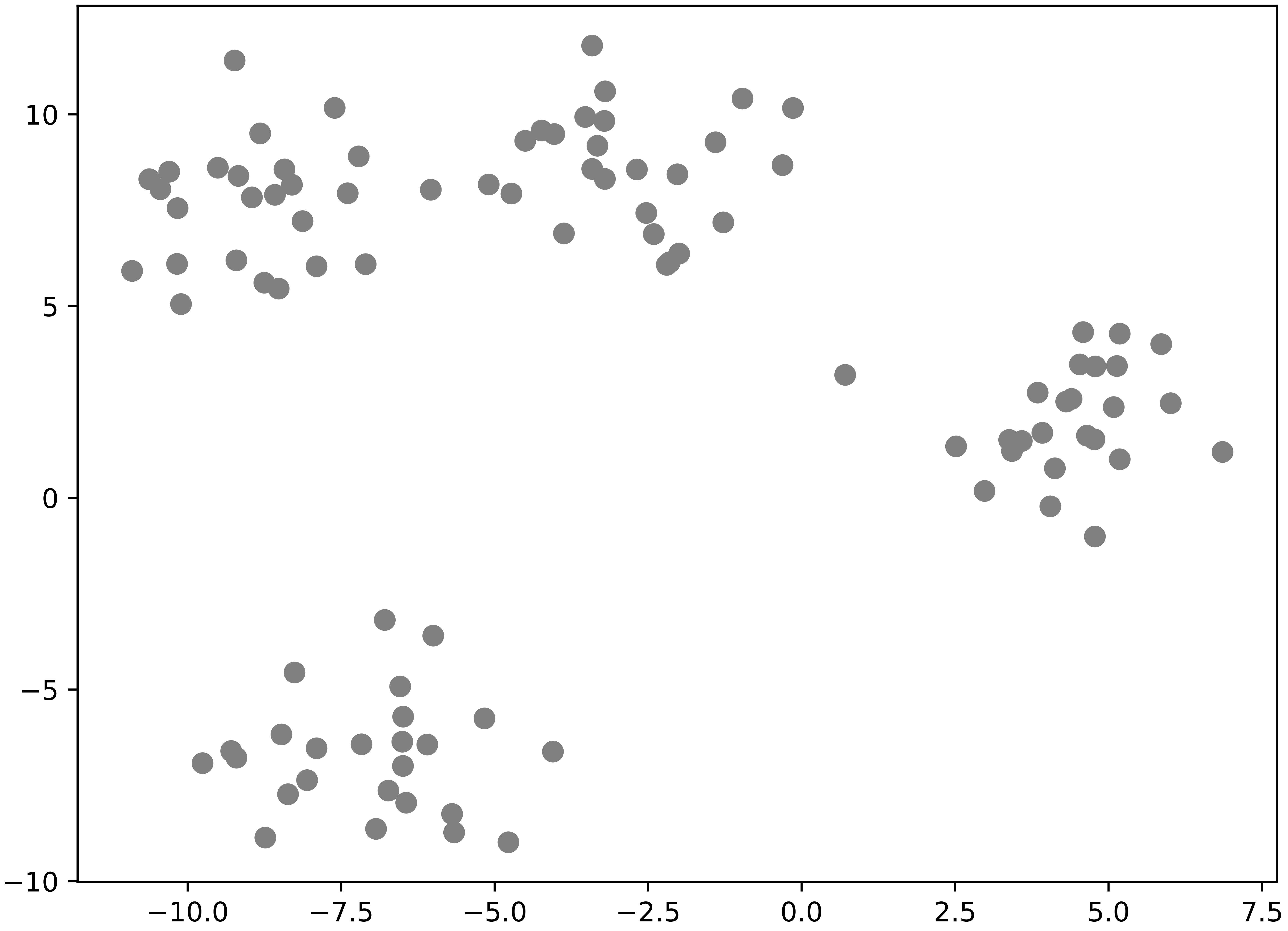
[J. VanderPlas]

Supervised Learning: Prediction



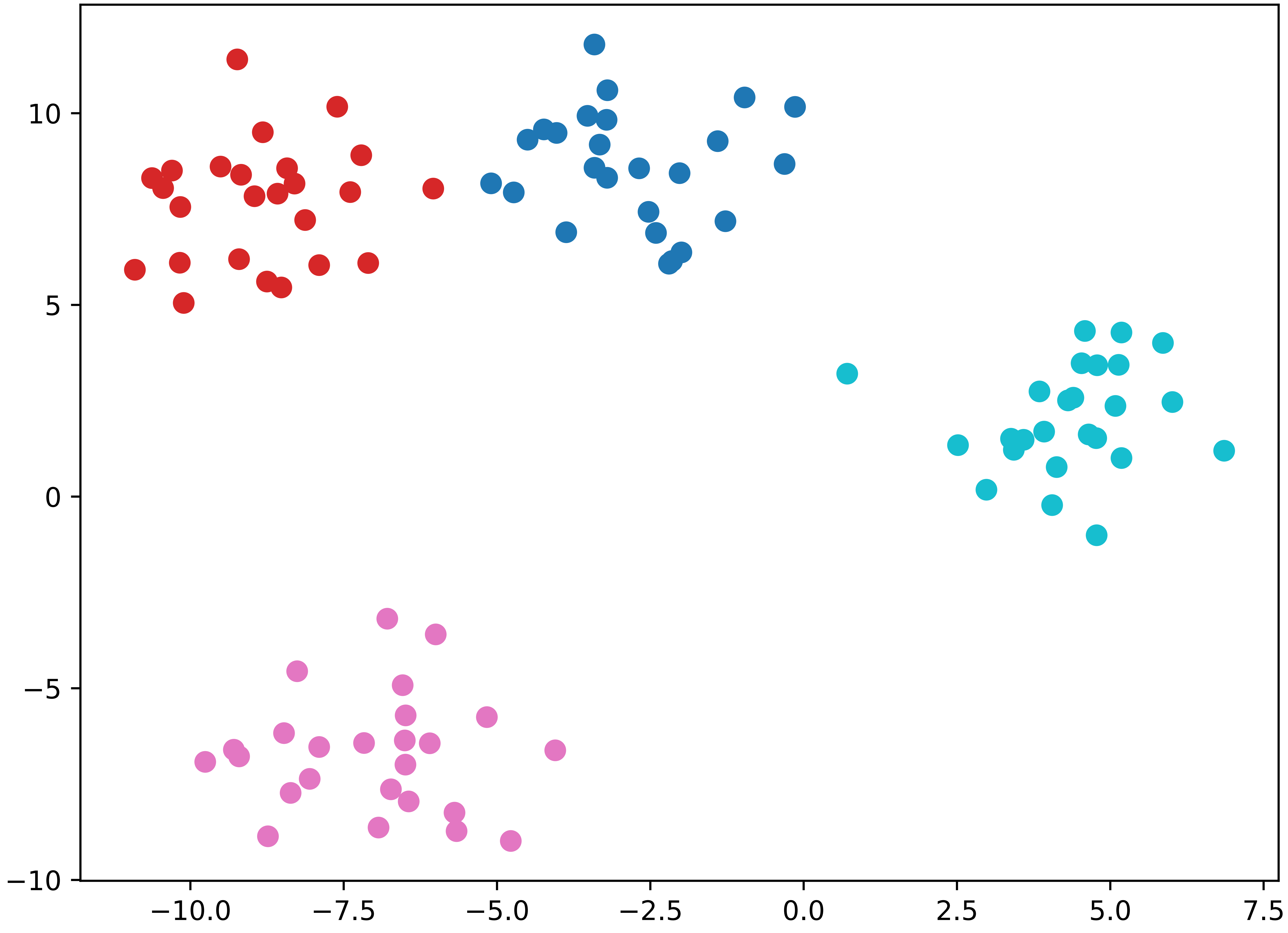
[J. VanderPlas]

Unsupervised Learning: Input



[J. VanderPlas]

Unsupervised Learning: Output



[J. VanderPlas]

scikit-learn entities

- Data: numpy matrices (also pandas series, data frames), process batches
- Estimator: all supervised & unsupervised algs implement **common** interface
  - estimator initialization does not do learning, only attaches parameters
  - `fit` does the learning, learned parameters exposed with trailing underscore
- Predictor: extends estimator with `predict` method
  - also provides `score` method to return value indicating prediction quality
- Transformer: help modify or filter data before learning
  - Preprocessing, feature selection, feature extraction, and dimensionality reduction vis `transform` method
  - Can combine `fit` and `transform` via `fit_transform`

[L. Buitinck et al.]

scikit-learn Template

1. Choose model class
2. Instantiate model
3. Fit model to data
4. Predict on new data

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(Xtrain, ytrain)
y_model = model.predict(Xtest)
```

5. (Check accuracy)

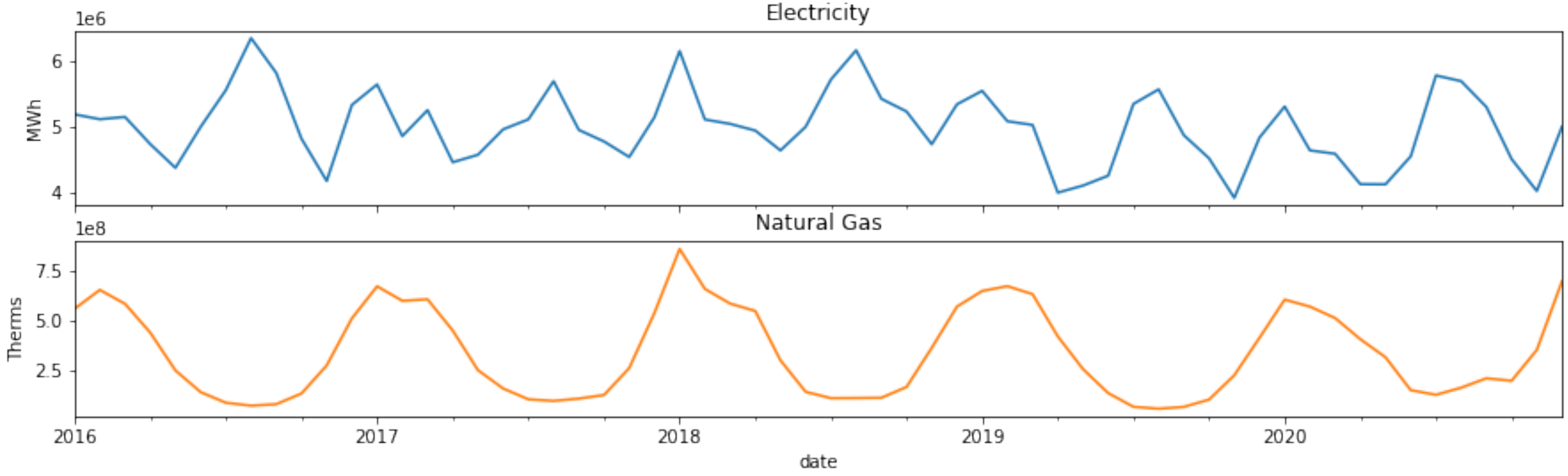
```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

Deep Learning

- Deep learning is tied to neural networks, attempting to mimic how human neurons work together
- Hierarchical with multiple layers
- Usually takes advantage of GPUs
- Frameworks:
  - pytorch
  - TensorFlow
  - keras
  - theano

Assignment 8

- Energy Data
- Data Manipulation using pandas
- Visualization using matplotlib and altair



Final Exam

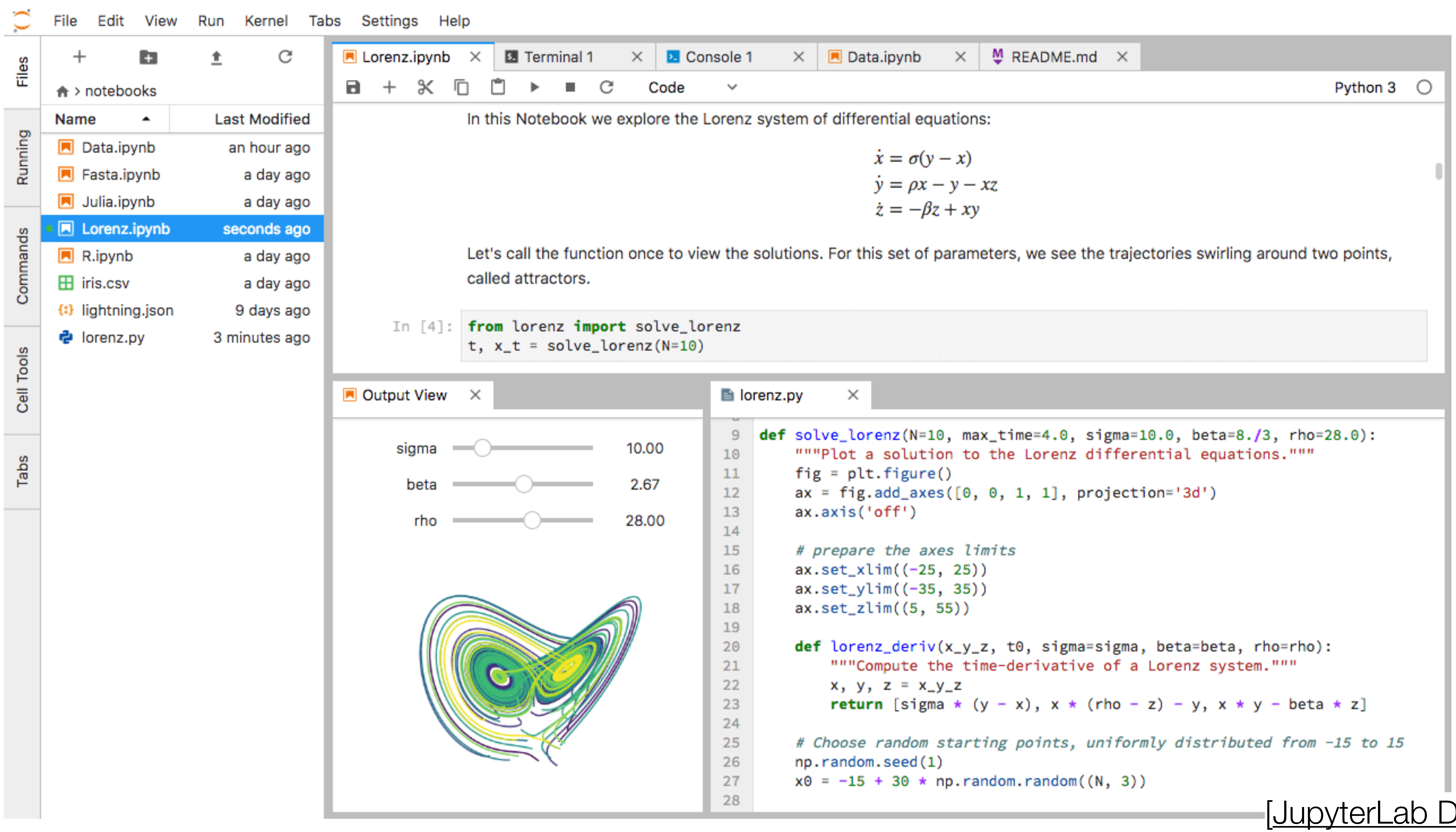
- Tuesday, December 7 at **12:00pm**-1:50pm in PM 153
- **More** comprehensive than Test 2
- Expect questions from topics covered on Test 1 and 2
- Expect questions from the last four weeks of class (concurrency, data, visualization, machine learning)
- Similar format

Questions?

Why Python?

- High-level, readable
- Productivity
- Large standard library
- Libraries, Libraries, Libraries
- What about Speed?
  - What speed are we measuring?
  - Time to code vs. time to execute

JupyterLab and Jupyter Notebooks



[JupyterLab Documentation]

Principles: Explicit Code

- Complex code isn't necessarily bad, but make sure you can't make it clearer
- Bad:

```
def make_complex(*args):
    x, y = args
    return dict(**locals())
```

- Good

```
def make_complex(x, y):
    return {'x': x, 'y': y}
```

[The Hitchhiker's Guide to Python]

Principles: Don't Repeat Yourself

- "Two or more, use a for" [Dijkstra]
- Rule of Three: [Roberts]
  - Don't copy-and-paste more than once
  - Refactor into methods
- Repeated code is harder to maintain
- Bad

```
f1 = load_file('f1.dat')
r1 = get_cost(f1)
f2 = load_file('f2.dat')
r2 = get_cost(f2)
f3 = load_file('f3.dat')
r3 = get_cost(f3)
```

- Good

```
for i in range(1,4):
    f = load_file(f'f{i}.dat')
    r = get_cost(f)
```

Expression Rules

- Involve
  - Literals (`1, "abc"`),
  - Variables (`a, my_height`), and
  - Operators (`+,-*,/,//,**`)
- Spaces are **irrelevant** within an expression
  - `a +              34 # ok`
- Standard precedence rules
  - Parentheses, exponentiation, mult/div, add/sub
  - **Left to right** at each level
- Also **boolean** expressions

Identifiers

- A sequence of letters, digits, or underscores, but…
- Also includes unicode "letters", spacing marks, and decimals (e.g. `Σ`)
- Must begin with a letter or underscore (_)
- Why not a number?
- Case sensitive (`a` is different from `A`)
- Conventions:
  - Identifiers beginning with an underscore (_) are reserved for system use
  - Use underscores (`a_long_variable`), **not** camel-case (`aLongVariable`)
  - Keep identifier names less than 80 characters
- Cannot be reserved words

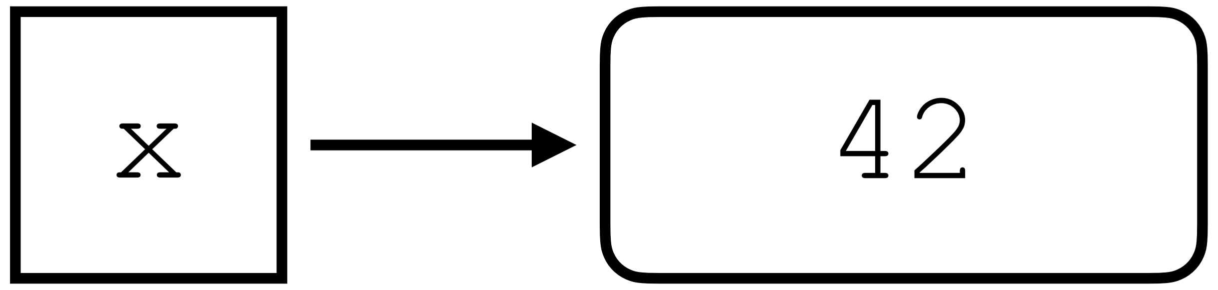
Types

- Don't worry about types, but think about types
- Variables can "change types"
  - ```
    a = 0
    a = "abc"
    a = 3.14159
    ```
- Actually, the name is being moved to a different value
- You can find out the type of the value stored at a variable `v` using `type(v)`
- Some literal types are determined by subtle differences
  - `1` vs `1.` (integer vs. float)
  - `1.43` vs `1.43j` (float vs. imaginary)
  - `'234'` vs `b'234'` (string vs. byte string)
    ```

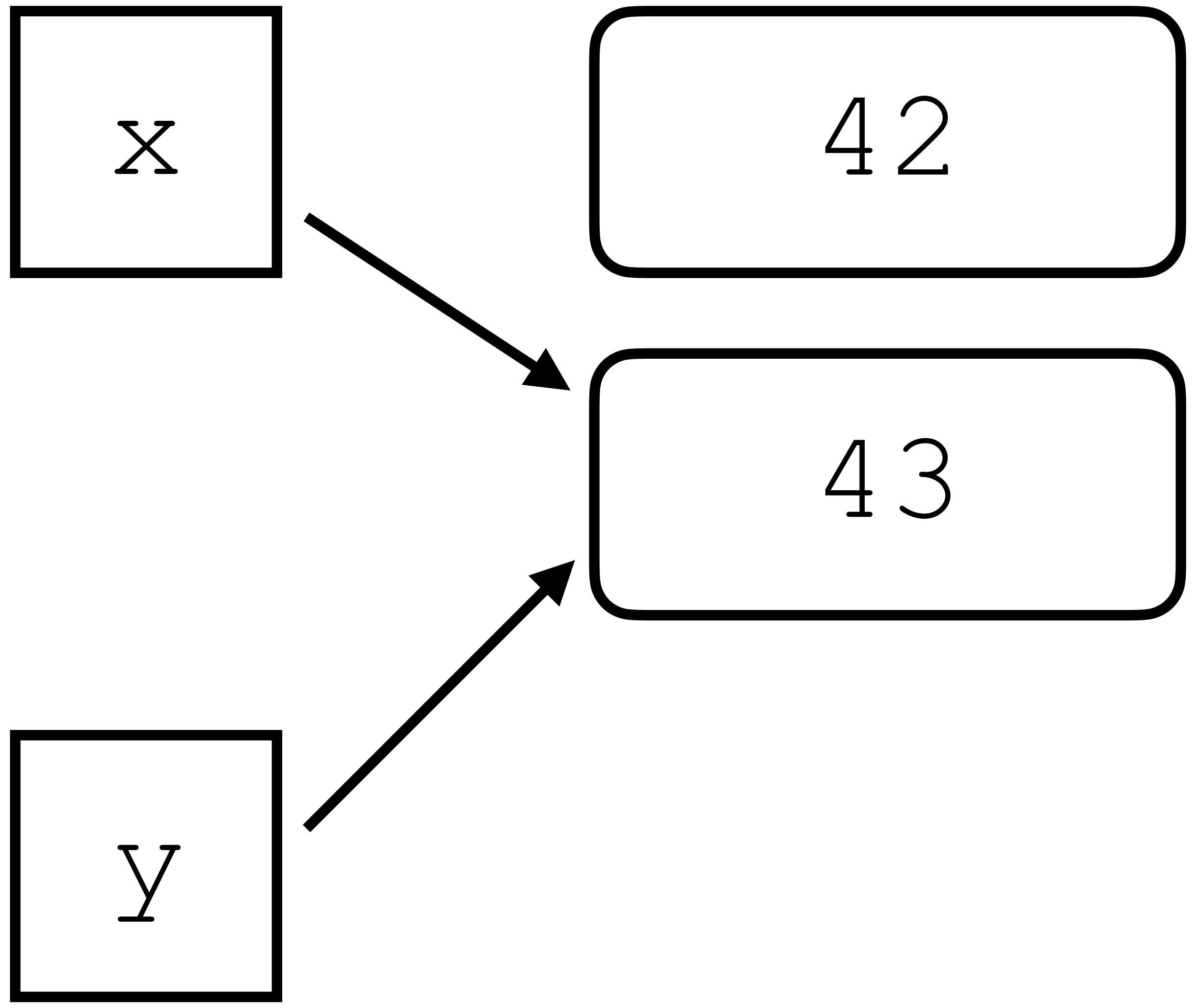
Assignment

- The = operator: `a = 34`; `c = (a + b) ** 2`
- Python variables are actually **pointers** to objects
- Also, augmented assignment: `+=`, `-=`, `*=`, `/=`, `//=`, `**=`

```
x = 42
```




```
x = x + 1
y = x
```



Boolean Expressions

- Type `bool`: `True` or `False`
- Note **capitalization**!
- Comparison Operators: `<`, `<=`, `>`, `>=`, `==`, `!=`
  - Double equals (==) checks for equal values,
  - Assignment (=) assigns values to variables
- Boolean operators: `not`, `and`, `or`
  - Different from many other languages (`!, &&, ||`)
- More:
  - `is`: exact same object (usually `a_variable is None`)
  - `in`: checks if a value is in a collection (`34 in my_list`)

if, else, elif, pass

- ```
  if a < 10:
      print("Small")
  else:
      if a < 100:
          print("Medium")
      else:
          if a < 1000:
              print("Large")
          else:
              print("X-Large")
  ```
- ```
  if a < 10:
      print("Small")
  elif a < 100:
      print("Medium")
  elif a < 1000:
      print("Large")
  else:
      print("X-Large")
  ```
- Indentation is critical so else-if branches can become unwieldy (elif helps)
- Remember colons and indentation
- `pass` can be used for an empty block

Loop Styles

- Loop-and-a-Half

```
d = get_data() # priming rd
while check(d):
    # do stuff
    d = get_data()
```

- Infinite-Loop-Break

```
while True:
    d = get_data()
    if check(d):
        break
    # do stuff
```

- Assignment Expression (Walrus)

```
while check(d := get_data):
    # do stuff
```

Functions

- Use `return` to return a value
-
```
def <function-name>(<parameter-names>):
    # do stuff
    return res
```
- Can return more than one value using commas
-
```
def <function-name>(<parameter-names>):
    # do stuff
    return res1, res2
```
- Use **simultaneous assignment** when calling:
  - `a, b = do_something(1,2,5)`
- If there is no return value, the function returns `None` (a special value)

Positional & Keyword Arguments

- Generally, any argument can be passed as a keyword argument
- ```
  def f(alpha, beta, gamma=1, delta=7, epsilon=8, zeta=2,
            eta=0.3, theta=0.5, iota=0.24, kappa=0.134):
      # …
  ```
- `f(5,6)`
- `f(alpha=7, beta=12, iota=0.7)`
  ```

Pass by object reference

- AKA passing object references by value
- Python doesn't allocate space for a variable, it just links identifier to a value
- **Mutability** of the object determines whether other references see the change
- Any immutable object will act like pass by value
- Any mutable object acts like pass by reference unless it is reassigned to a new value

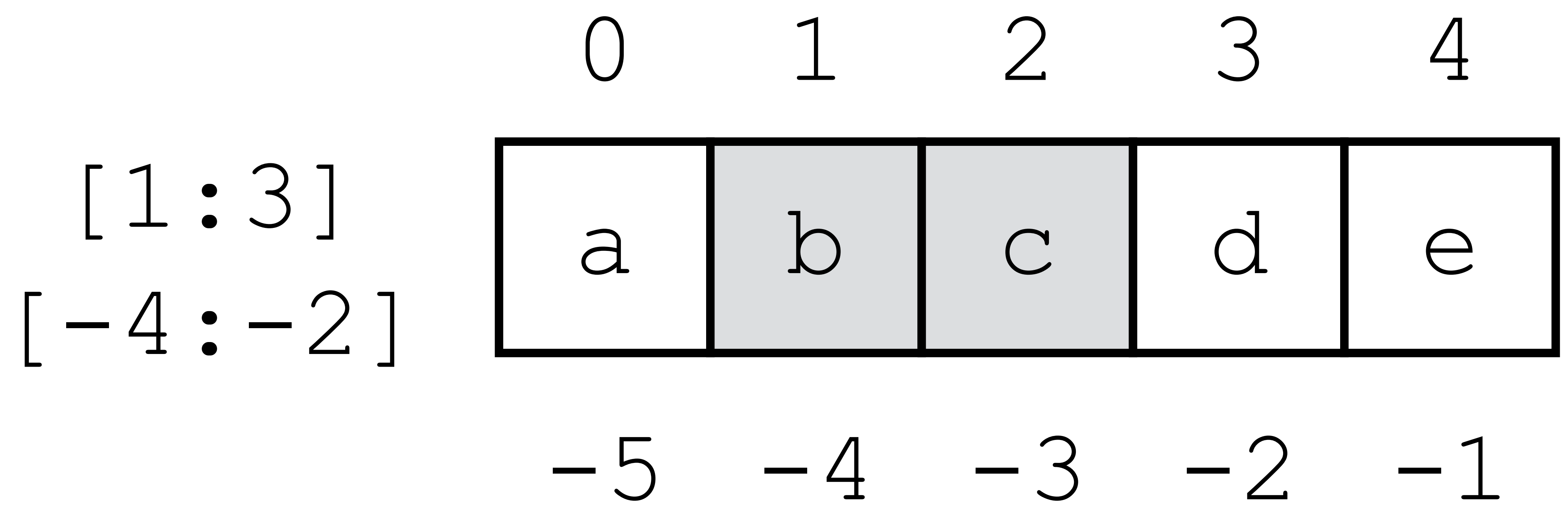
Sequences

- Strings `"abcde"`, Lists `[1, 2, 3, 4, 5]`, and Tuples `(1, 2, 3, 4, 5)`

- Defining a list: `my_list = [0, 1, 2, 3, 4]`
- But lists can store different types:
  - `my_list = [0, "a", 1.34]`
- Including other lists:
  - `my_list = [0, "a", 1.34, [1, 2, 3]]`
- Others are similar: tuples use parenthesis, strings are delineated by quotes (single or double)

Indexing & Slicing

- Positive or negative indices can be used at any step
- `my_str = "abcde"; my_str[1] + my_str[-4]# "bb"`
- `my_list = [1,2,3,4,5]; my_list[3:-1] # [4]`
- Implicit indices
  - `my_tuple = (1,2,3,4,5); my_tuple[-2:] # (4,5)`
  - `my_tuple[:3] # (1,2,3)`



Tuples

- Tuples are immutable sequences
- We've actually seen tuples a few times already
  - Simultaneous Assignment
  - Returning Multiple Values from a Function
- Python allows us to omit parentheses when it's clear
  - `b, a = a, b         # same as (b, a) = (a, b)`
  - `t1 = a, b           # don't normally do this`
  - `c, d = f(2, 5, 8) # same as (c, d) = f(2, 5, 8)`
  - `t2 = f(2, 5, 8)     # don't normally do this`

Dictionary

- AKA associative array or map
- Collection of key-value pairs
  - Keys must be unique
  - Values need not be unique
- Syntax:
  - Curly brackets {} delineate start and end
  - Colons separate keys from values, commas separate pairs
  - `d = {'DeKalb': 783, 'Kane': 134, 'Cook': 1274, 'Will': 546}`
- No type constraints
  - `d = {'abc': 25, 12: 'abc', ('Kane', 'IL'): 123.54}`

Collections

- A dictionary is **not** a sequence
- Sequences are **ordered**
- Conceptually, dictionaries need no order
- A dictionary is a **collection**
- Sequences are also collections
- All collections have length (`len`), membership (`in`), and iteration (loop over values)
- Length for dictionaries counts number of key-value **pairs**
  - Pass dictionary to the `len` function
  - ```
    d = {'abc': 25, 12: 'abc', ('Kane', 'IL'): 123.54}
    len(d) # 3
    ```
    ```

List Comprehension

- ```
  output = []
  for d in range(5):
      output.append(d ** 2 - 1)
  ```

- Rewrite as a map:
  - `output = [d ** 2 - 1 for d in range(5)]`
- Can also filter:
  - `output = [d for d in range(5) if d % 2 == 1]`
- Combine map & filter:
  - `output = [d ** 2 - 1 for d in range(5) if d % 2 == 1]`
  ```

Short-Circuit Evaluation

- Automatic, works left to right according to order of operations (and before or)
- Works for `and` and `or`
- and:
  - if **any** value is `False`, stop and return `False`
  - ```
    a, b = 2, 3
    a > 3 and b < 5
    ```
- or:
  - if **any** value is `True`, stop and return `True`
  - ```
    a, b, c = 2, 3, 7
    a > 3 or b < 5 or c > 8
    ```

Strings

- Remember strings are sequences of characters
- Strings are collections so have `len`, `in`, and iteration
  - ```
    s = "Huskies"
    len(s); "usk" in s; [c for c in s if c == 's']
    ```
- Strings are sequences so have
  - indexing and slicing: `s[0], s[1:]`
  - concatenation and repetition: `s + " at NIU"; s * 2`
- Single or double quotes `'string1'`, `"string2"`
- Triple double-quotes: `"""A string over many lines"""`
- Escaped characters: `'\n'` (newline) `'\t'` (tab)
    ```

Regular Expressions

- AKA regex
- A syntax to better specify how to decompose strings
- Look for patterns rather than specific characters
- `"31" in "The last day of December is 12/31/2016."`
- May work for some questions but now suppose I have other lines like: `"The last day of September is 9/30/2016."`
- …and I want to find dates that look like:
- `{digits}/{digits}/{digits}`
- Cannot search for every combination!
- `\d+/\d+/\d+ # \d is a` **`character class`**

Reading & Writing Files

- Can iterate through the file (think of the file as a collection of lines):
  - ```
    f = open('huck-finn.txt', 'r')
    for line in f:
        if 'Huckleberry' in line:
            print(line.strip())
    ```
- For writing, with statement does "enter" and "exit": don't need to call `outf.close()`
  - ```
    with open('output.txt', 'w') as outf:
        for k, v in counts.items():
            outf.write(k + ': ' + v + '\n')
    ```

Command Line Interfaces (CLIs)

- Prompt:
  - $
  - 

- Commands
  - `$ cat <filename>`
  - `$ git init`
- Arguments/Flags: (options)
  - `$ python -h`
  - `$ head -n 5 <filename>`
  - `$ git branch fix-parsing-bug`

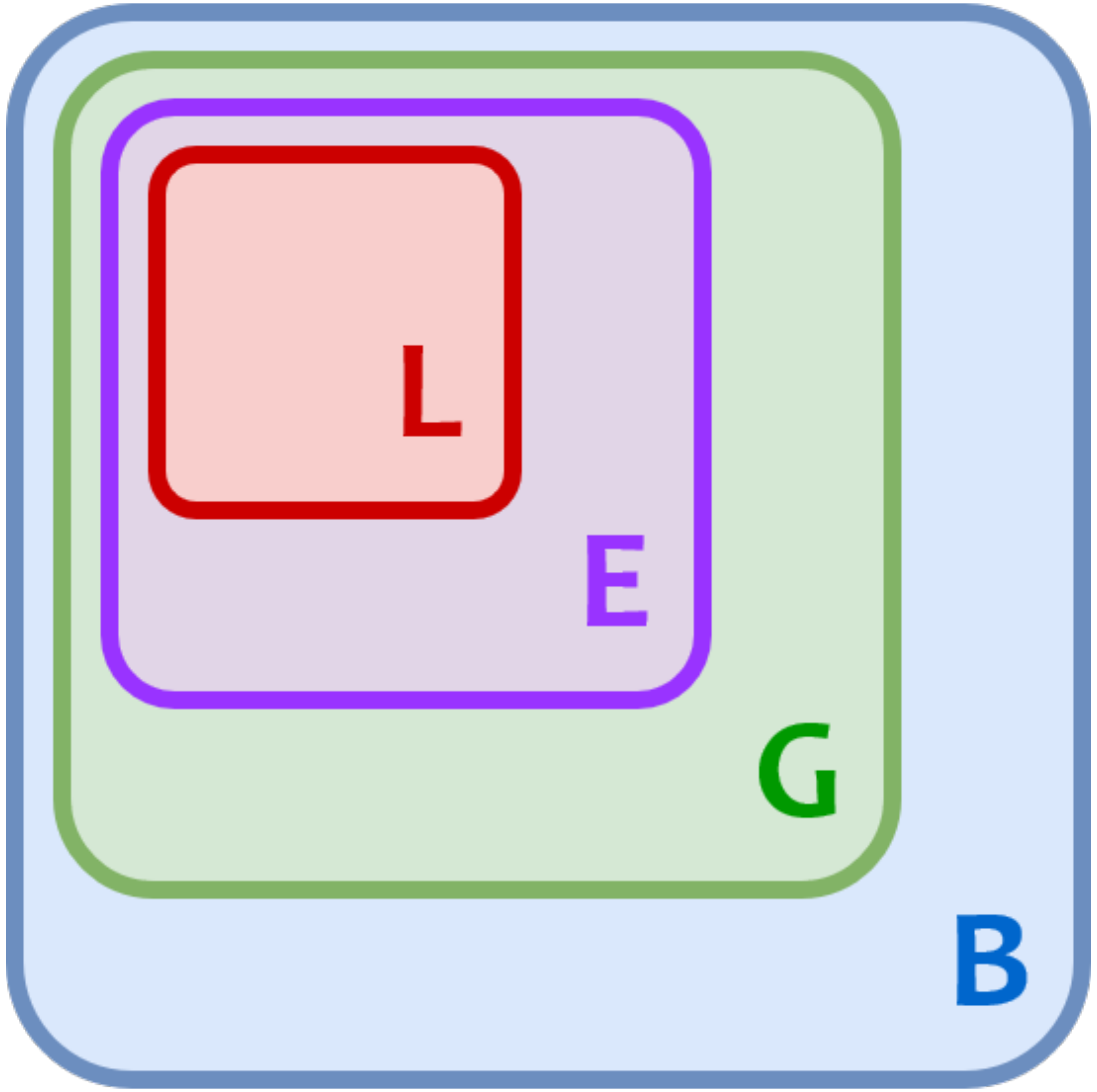
Modules and Packages

- Python allows you to import code from other files, even your own
- A **module** is a collection of definitions
- A **package** is an organized collection of modules
- Modules can be
  - a separate python file
  - a separate C library that is written to be used with Python
  - a built-in module contained in the interpreter
  - a module installed by the user (via conda or pip)
- All types use the same import syntax

[RealPython]

Namespaces

- Namespace is basically a dictionary with names and their values
- Accessing namespaces
  - `__builtins__`, `globals()`, `locals()`
- Examine contents of a namespace: `dir(<namespace>)`
- Python checks for a name in the sequence: local, enclosing, global, builtins
- To access names in outer scopes, use `global` (global) and `nonlocal` (enclosing) declarations



[RealPython]

Array Operations

- `a = np.array([1,2,3])`
  `b = np.array([6,4,3])`
- (Array, Array) Operations (**Element-wise**)
  - Addition, Subtraction, Multiplication
  - `a + b # array([7, 6, 6])`
- (Scalar, Array) Operations (**Broadcasting**):
  - Addition, Subtraction, Multiplication, Division, Exponentiation
  - `a ** 2 # array([1, 4, 9])`
  - `b + 3 # array([9, 7, 6])`

Array Slicing

- 2D+: comma separated indices as shorthand:
  - `arr2 = np.array([[1.5,2,3,4],[5,6,7,8]])`
  - `a[1:2,1:3]`
  - `a[1:2,:] # works like in single-dimensional lists`
- Can combine index and slice in different dimensions
  - `a[1,:] # gives a row`
  - `a[:,1] # gives a column`
- Slicing vs. indexing produces different shapes!
  - `a[1,:] # 1-dimensional`
  - `a[1:2,:] # 2-dimensional`

Object-Oriented Programming Concepts

- Abstraction: simplify, hide implementation details, don't repeat yourself
- Encapsulation: represent an entity fully, keep attributes and methods together
- Inheritance: reuse (don't reinvent the wheel), specialization
- Polymorphism: methods are handled by a single interface with different implementations (overriding)

Classes and Instances in Python

- Class Definition:
  - ```
    class Vehicle:
        def __init__(self, make, model, year, color):
            self.make = make
            self.model = model
            self.year = year
            self.color = color

        def age(self):
            return 2021 - self.year
    ```
- Instances:
  - `car1 = Vehicle('Toyota', 'Camry', 2000, 'red')`
  - `car2 = Vehicle('Dodge', 'Caravan', 2015, 'gray')`
    ```

Subclass

- Just put superclass(-es) in parentheses after the class declaration
- ```
  class Car(Vehicle):
      def __init__(self, make, model, year, color, num_doors):
          super().__init__(make, model, year, color)
          self.num_doors = num_doors

      def open_door(self):
          …
  ```
- `super()` is a special method that locates the base class
  - Constructor should call superclass constructor
  - Extra arguments should be initialized and extra instance methods
  ```

Typing

- Dynamic Typing: variable's type can change (what Python does)
- Static Typing: compiler enforces types, variable types generally don't change
- Duck Typing: check method/attribute existence, not type
- Python is a dynamically-typed language (and plans to remain so)
- …but it has recently added more support for type hinting/annotations that allow **static type checking**
- Type annotations change **nothing** at runtime!

[RealPython, G. A. Hjelle]

Dealing with Errors

- Can explicitly check for errors at each step
  - Check for division by zero
  - Check for invalid parameter value (e.g. string instead of int)
- Sometimes all of this gets in the way and can't be addressed succinctly
  - Too many potential errors to check
  - Cannot handle groups of the same type of errors together
- Allow programmer to determine when and how to handle issues
  - Allow things to go wrong and handle them instead
  - Allow errors to be propagated and addressed once

Try, Except, Else, and Finally

- ```
  b = 3
  a = 0
  try:
      c = b / a
  except ZeroDivisionError:
      print("Division failed")
      c = 0
  else:
      print("Division succeeded", c)
  finally:
      print("This always runs")
  ```
  ```

Debugging

- print statements
- logging library
- pdb
- Extensions for IDEs (e.g. PyCharm)
- JupyterLab Debugger Support

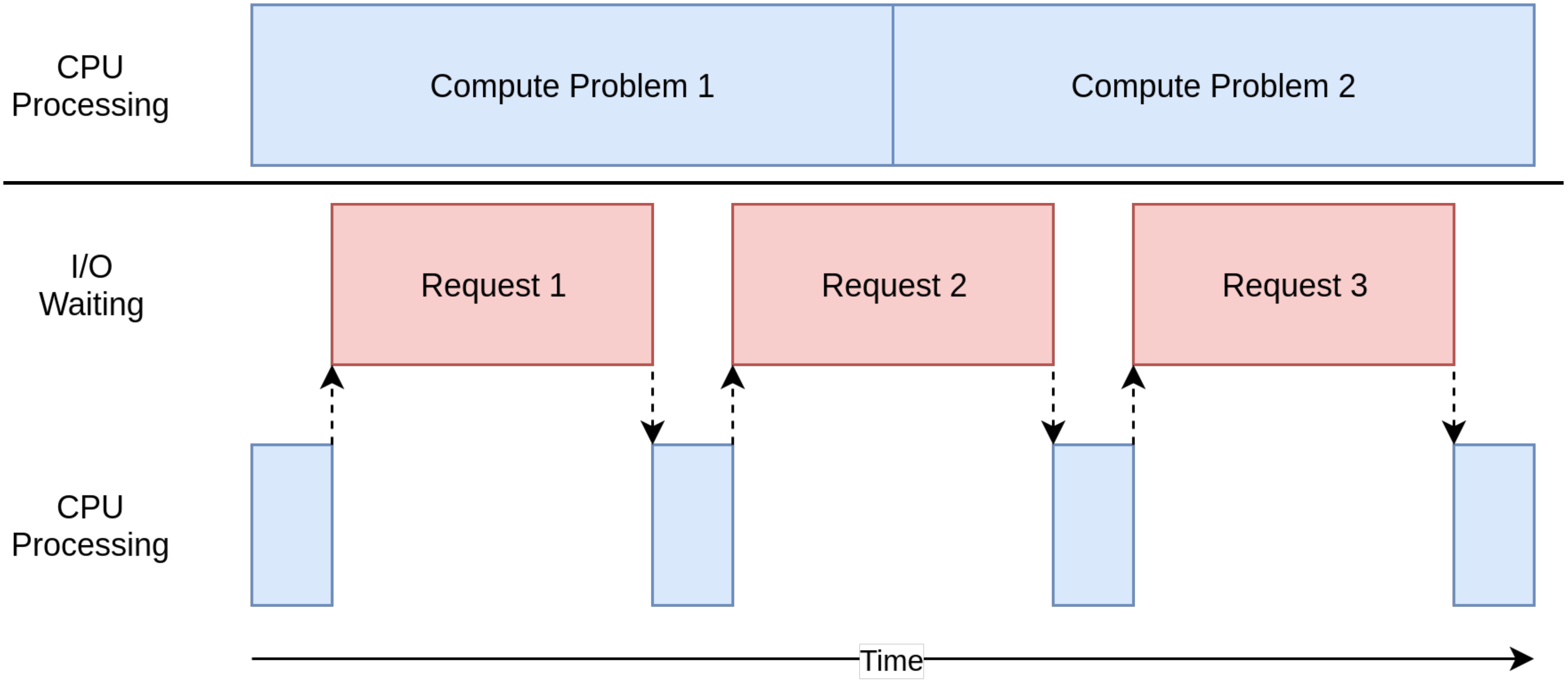
Testing

- If statements
- Assert statements
- Unit Testing
- Integration Testing

Python Modules for Working with the Filesystem

- In general, cross-platform! (Linux, Mac, Windows)
- `os`: translations of operating system commands
- `shutil`: better support for file and directory management
- `fnmatch`, `glob`: match filenames, paths
- `os.path`: path manipulations
- `pathlib`: object-oriented approach to path manipulations, also includes some support for matching paths

Concurrency: CPU-Bound vs. I/O-Bound



[J. Anderson]

Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

Column Names

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

Column Names

Index

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

Column Names

Index

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Column: `df['Island']`

Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

Column Names

Row: `df.loc[2]`

Index

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Column: `df['Island']`

Data Frame

```python
df = pd.read_csv('penguins_lter.csv')
```

Column Names

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
341	PAL0910	122	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Row: `df.loc[2]`

Index

Cell: `df.loc[341,'Species']`

Column: `df['Island']`

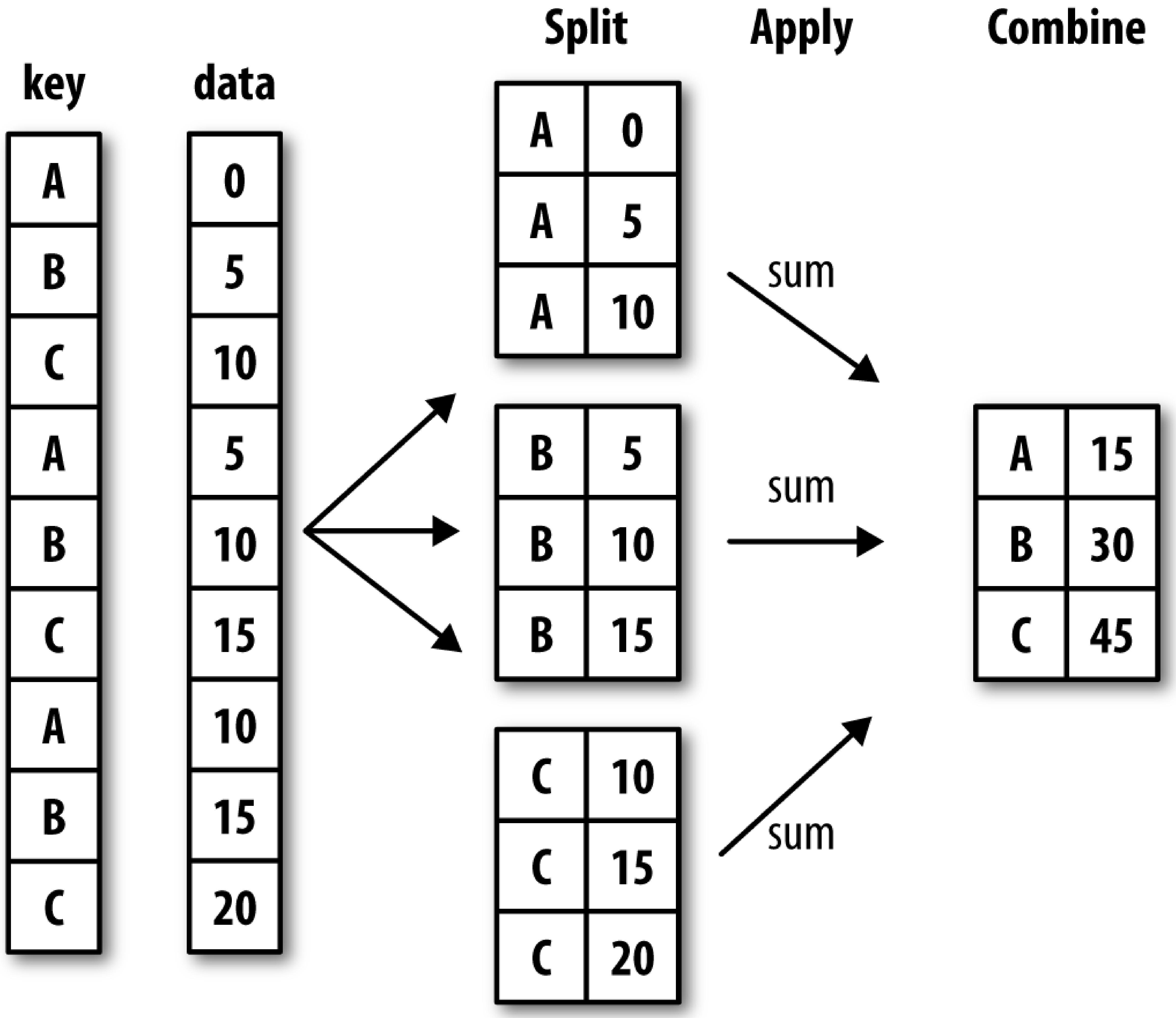
Data Frame

```
df = pd.read_csv('penguins_lter.csv')
```

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	
...	...	...	...	...	...	...	...	...	...	...
339	PAL0910	120	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N38A2	No	12/1/09	NaN
340	PAL0910	121	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A1	Yes	11/22/09	46.8
			Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N39A2	Yes	11/22/09	50.4
342	PAL0910	123	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A1	Yes	11/22/09	45.2
343	PAL0910	124	Gentoo penguin (Pygoscelis papua)	Anvers	Biscoe	Adult, 1 Egg Stage	N43A2	Yes	11/22/09	49.9

344 rows × 17 columns

Aggregation: Split-Apply-Combine



[W. McKinney, Python for Data Analysis]

Tidy Data: Melt

- Want to keep each observation separate (tidy), aka pivot_longer

	location	Temperature	Jan-2010	Feb-2010	Mar-2010
0	CityA	Predict	30	45	24
1	CityB	Actual	32	43	22

```
df.melt(id_vars=["location", "Temperature"],
        var_name="Date", value_name="Value")
```

	location	Temperature	Date	Value
0	CityA	Predict	Jan-2010	30
1	CityB	Actual	Jan-2010	32
2	CityA	Predict	Feb-2010	45
3	CityB	Actual	Feb-2010	43
4	CityA	Predict	Mar-2010	24
5	CityB	Actual	Mar-2010	22

[AB Abhi]

Tidy Data: Pivot

- Sometimes, we have data that is given in "long" format and we would like "wide" format (aka pivot_wider)
- Long format: column names are data values…
- Wide format: more like spreadsheet format
- Example:

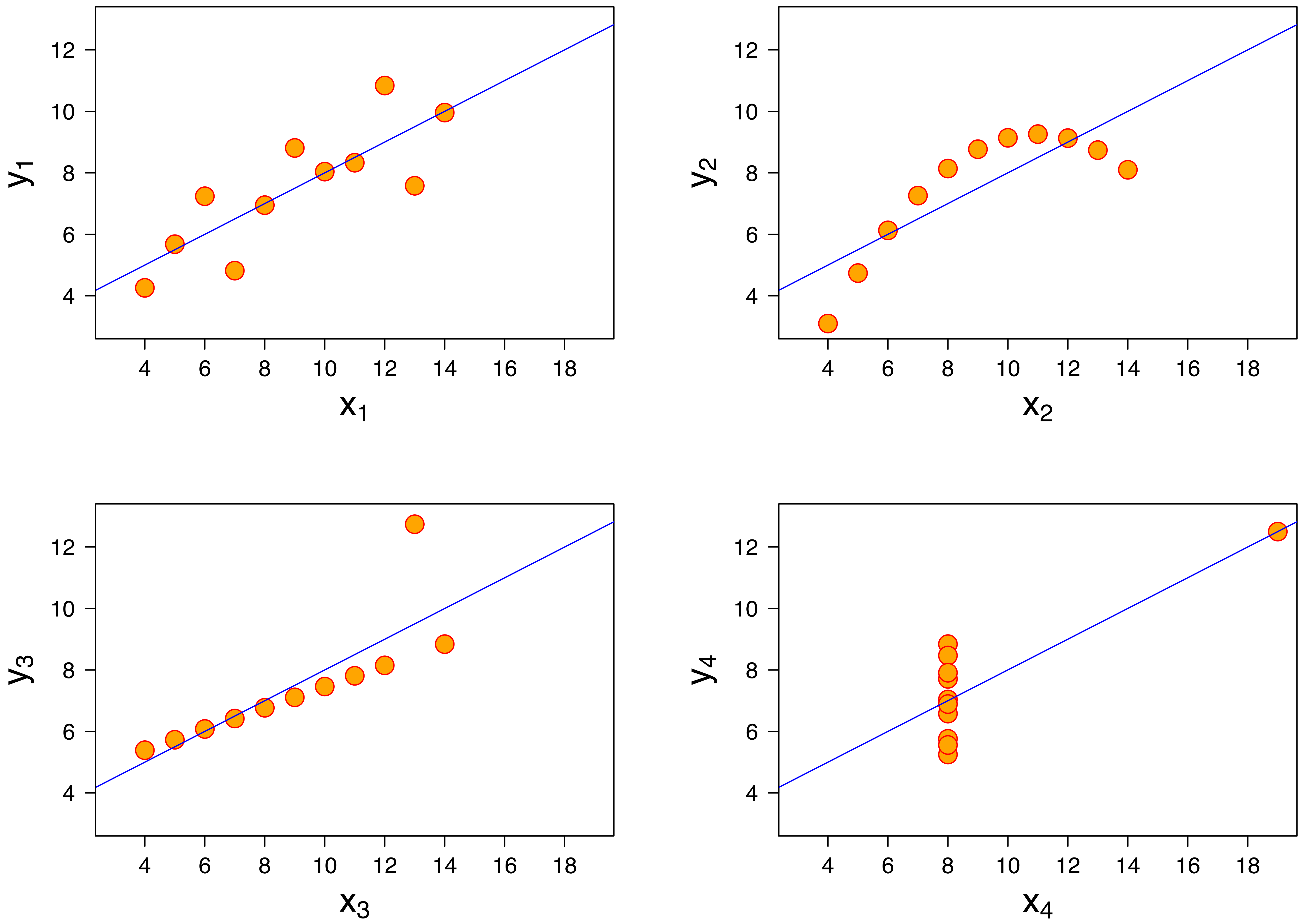
```
        date     item     value
0 1959-03-31  realgdp  2710.349
1 1959-03-31     infl     0.000
2 1959-03-31    unemp     5.800
3 1959-06-30  realgdp  2778.801
4 1959-06-30     infl     2.340
5 1959-06-30    unemp     5.100
6 1959-09-30  realgdp  2775.488
7 1959-09-30     infl     2.740
8 1959-09-30    unemp     5.300
9 1959-12-31  realgdp  2785.204
```

```
.pivot('date', 'item', 'value')
```

```
item        infl   realgdp  unemp
date
1959-03-31  0.00  2710.349    5.8
1959-06-30  2.34  2778.801    5.1
1959-09-30  2.74  2775.488    5.3
1959-12-31  0.27  2785.204    5.6
1960-03-31  2.31  2847.699    5.2
```

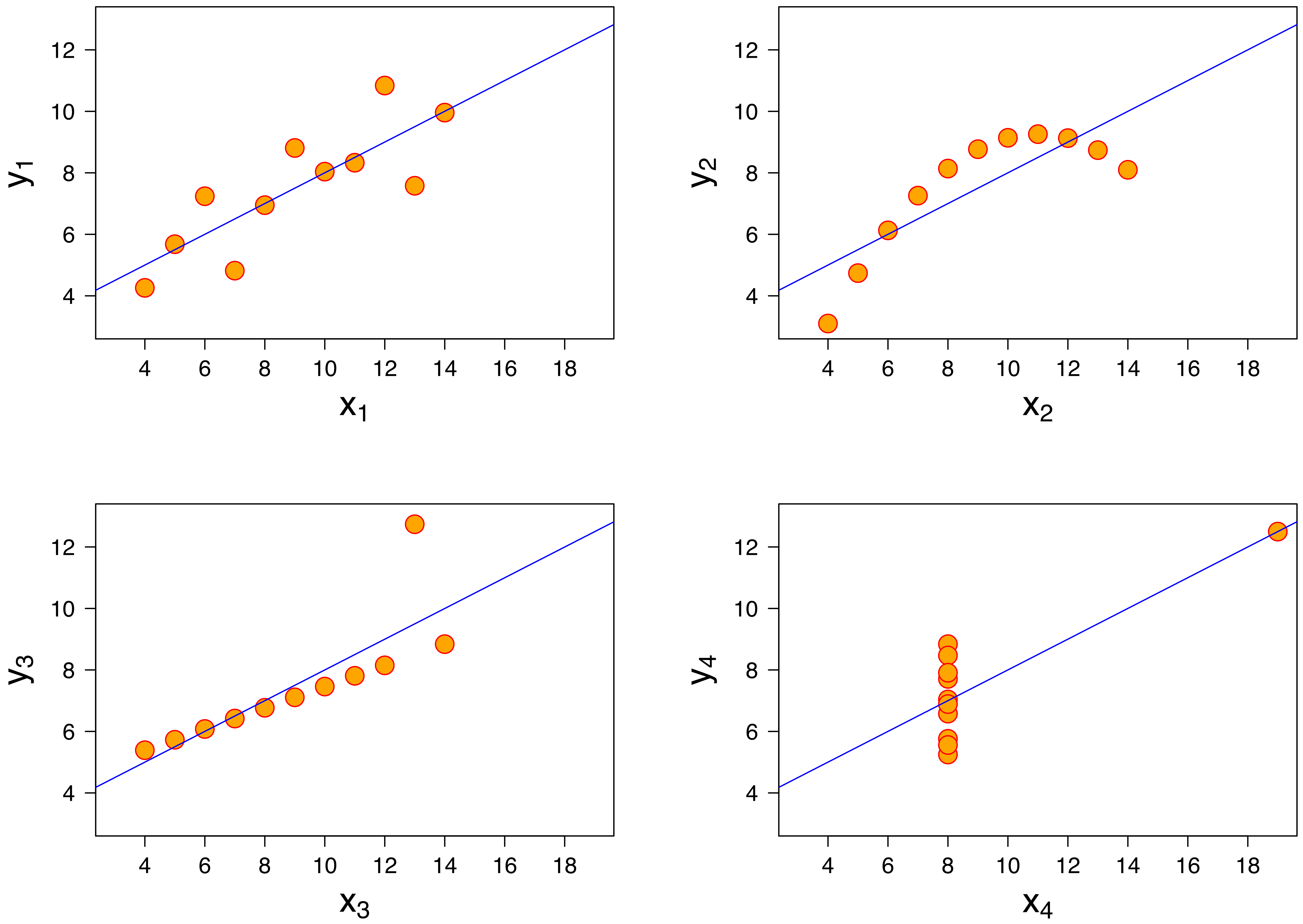
[W. McKinney, Python for Data Analysis]

Visualizing Data



[F. J. Anscombe]

Visualizing Data

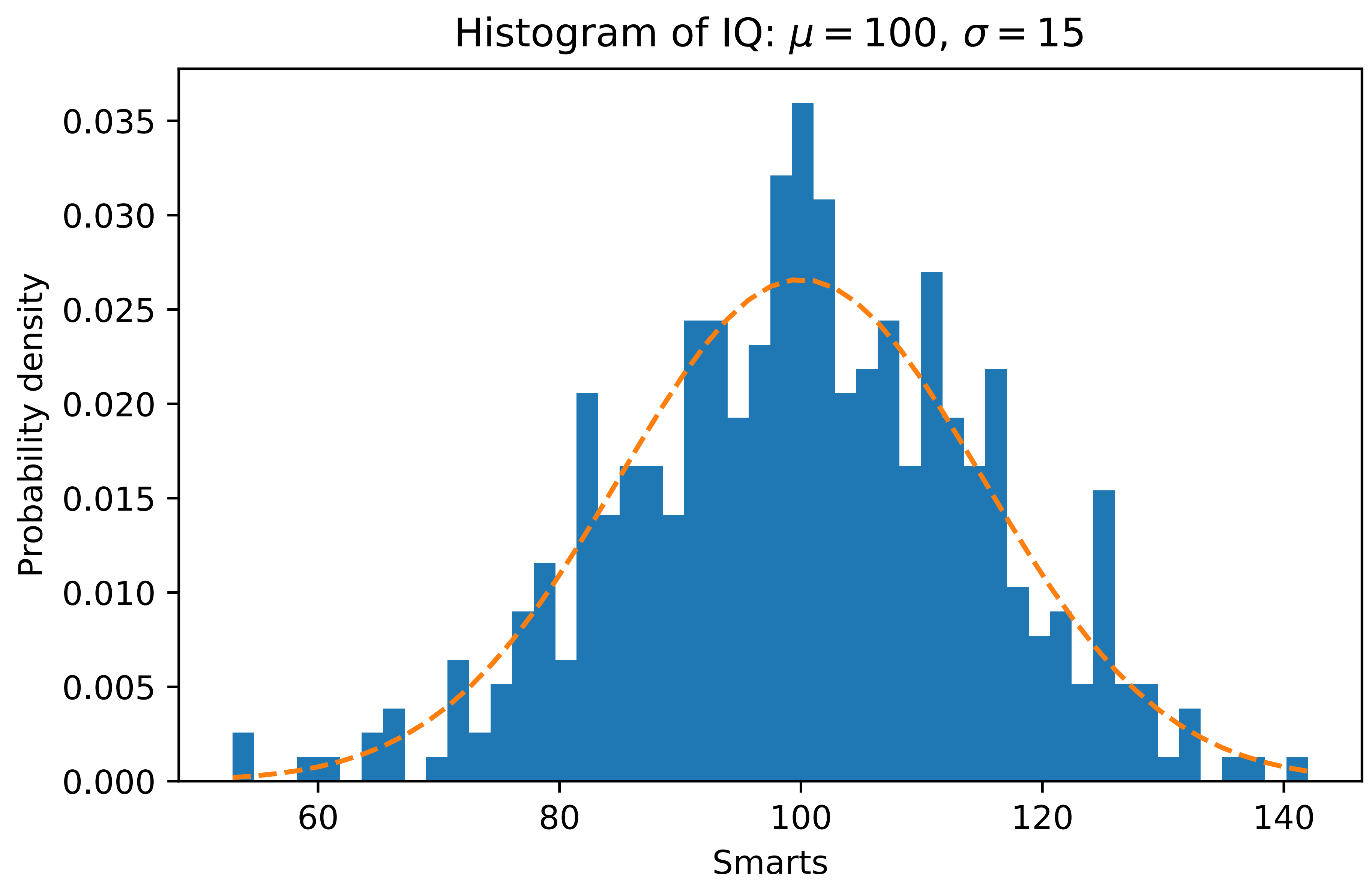


Mean of x	9
Variance of x	11
Mean of y	7.50
Variance of y	4.122
Correlation	0.816

[F. J. Anscombe]

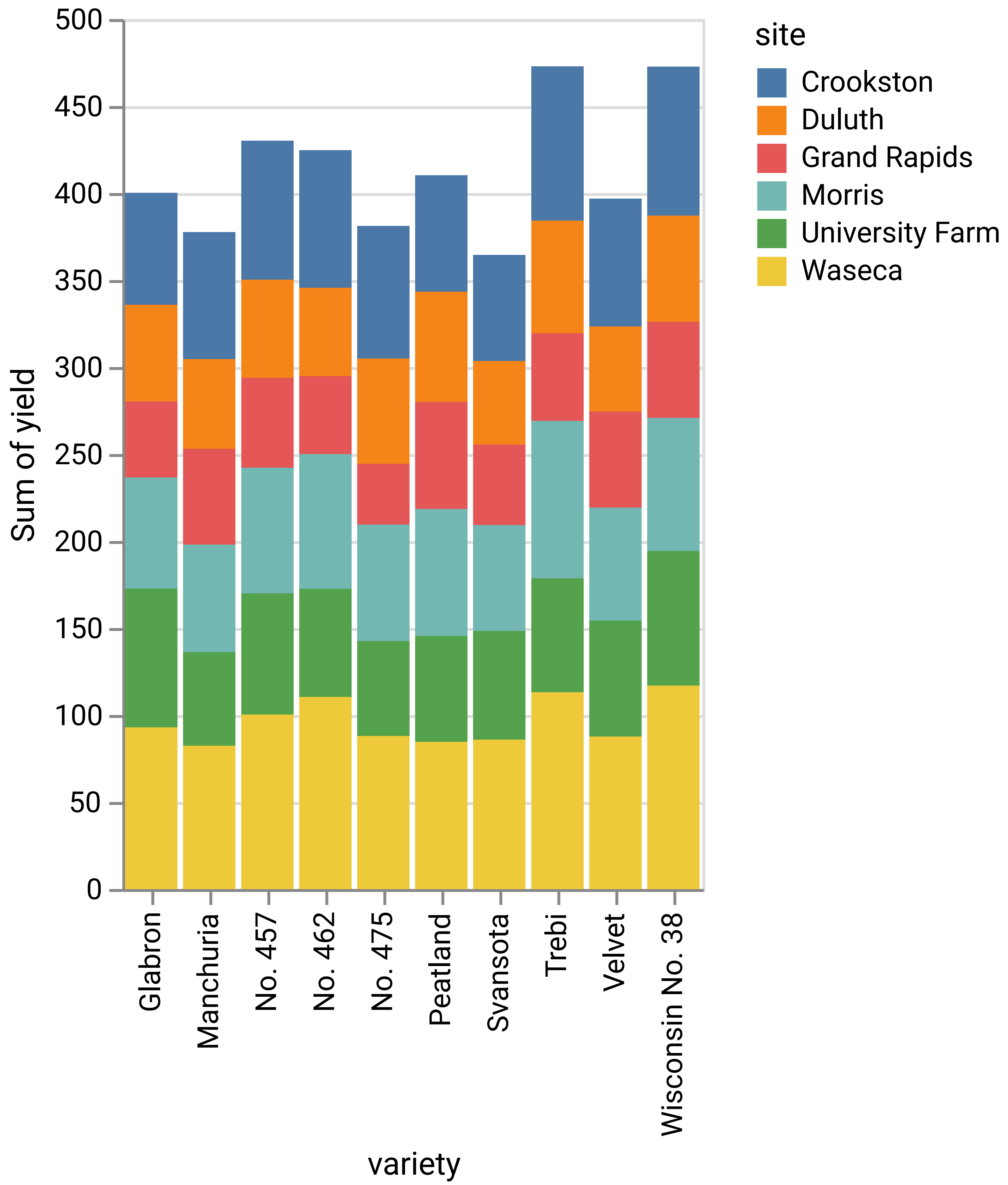
matplotlib

- Strengths:
  - Designed like Matlab
  - Many rendering backends
  - Can reproduce almost any plot
  - Proven, well-tested
- Weaknesses:
  - API is imperative
  - Not originally designed for the web
  - Dated styles



Altair

- Declarative Visualization
  - Specify **what** instead of how
  - Separate specification from execution
- Based on VegaLite which is browser-based
- Strengths:
  - Declarative visualization
  - Web technologies
- Drawbacks:
  - Moving data between Python and JS
  - Sometimes longer specifications

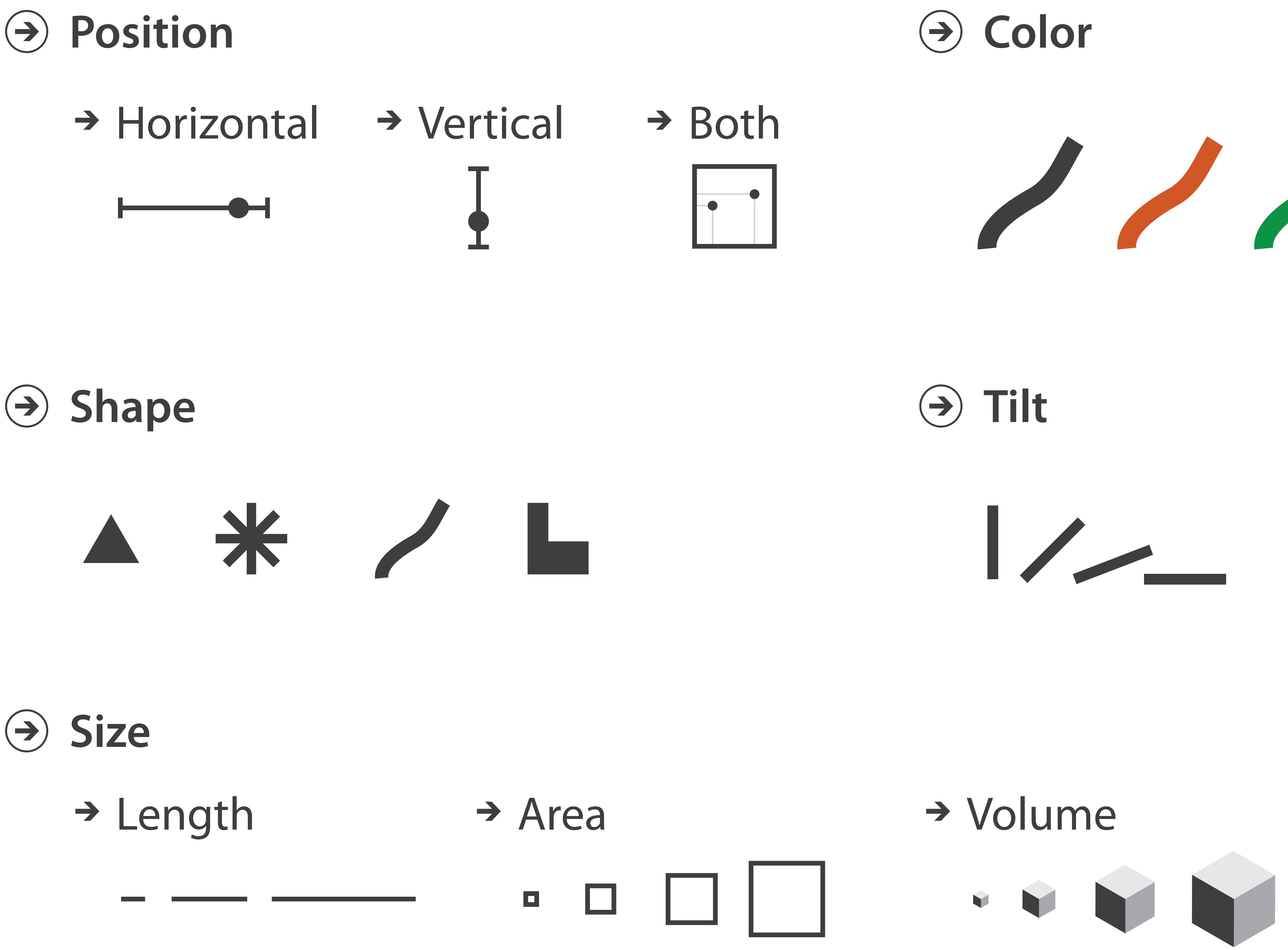


Visual Marks

- **Marks** are the basic graphical elements in a visualization
- Marks classified by dimensionality:

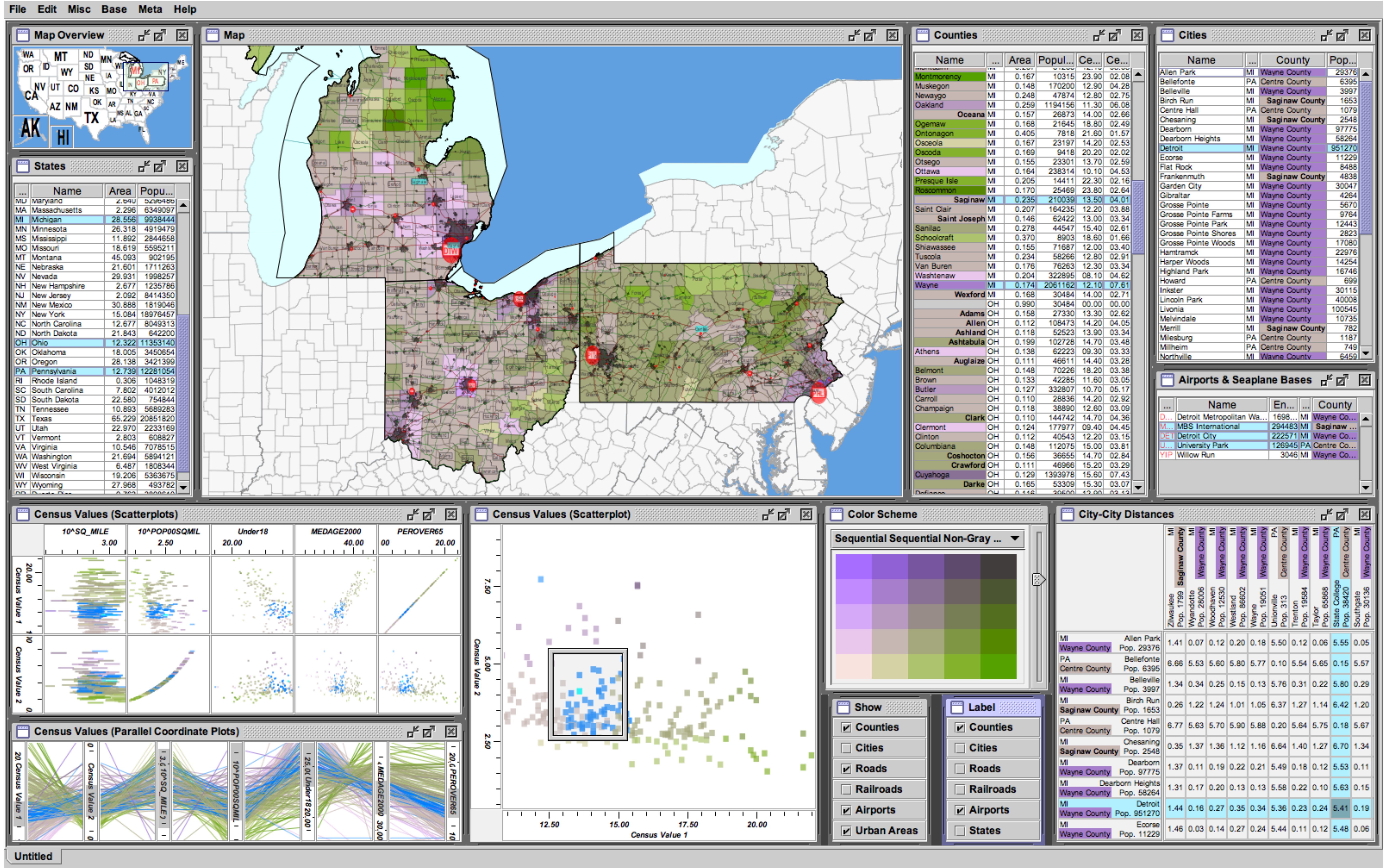
- Also can have surfaces, volumes
- Think of marks as a mathematical definition, or if familiar with tools like Adobe Illustrator or Inkscape, the path & point definitions
- Altair: area, bar, circle, geoshape, image, line, point, rect, rule, square, text, tick
  - Also compound marks: boxplot, errorband, errorbar

Data is Encoded via Visual Channels



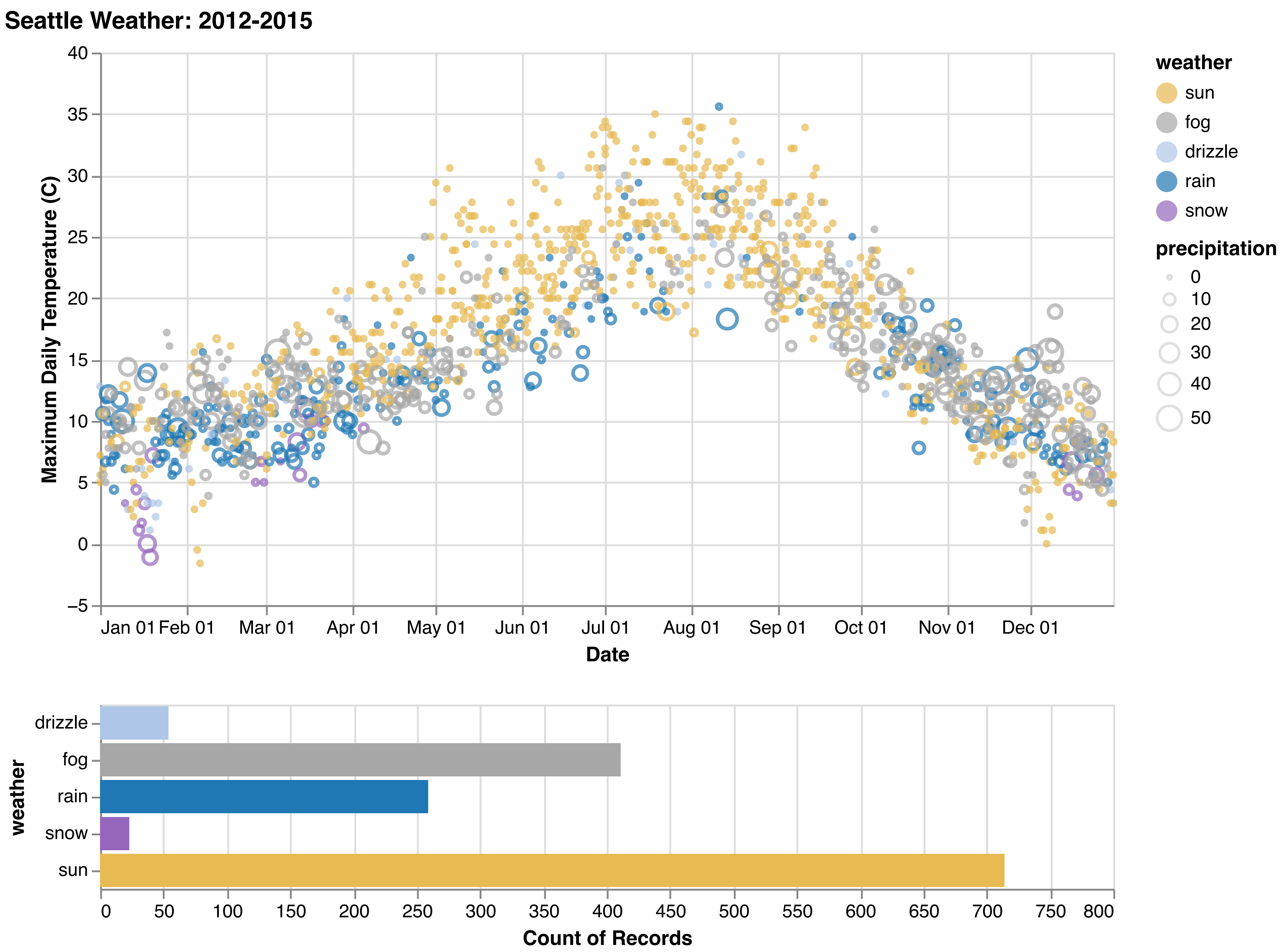
[Munzner (ill. Maguire), 2014]

Multiple Views



[Improvise, Weaver, 2004]

Interaction



Questions?