Programming Principles in Python (CSCI 503/490)

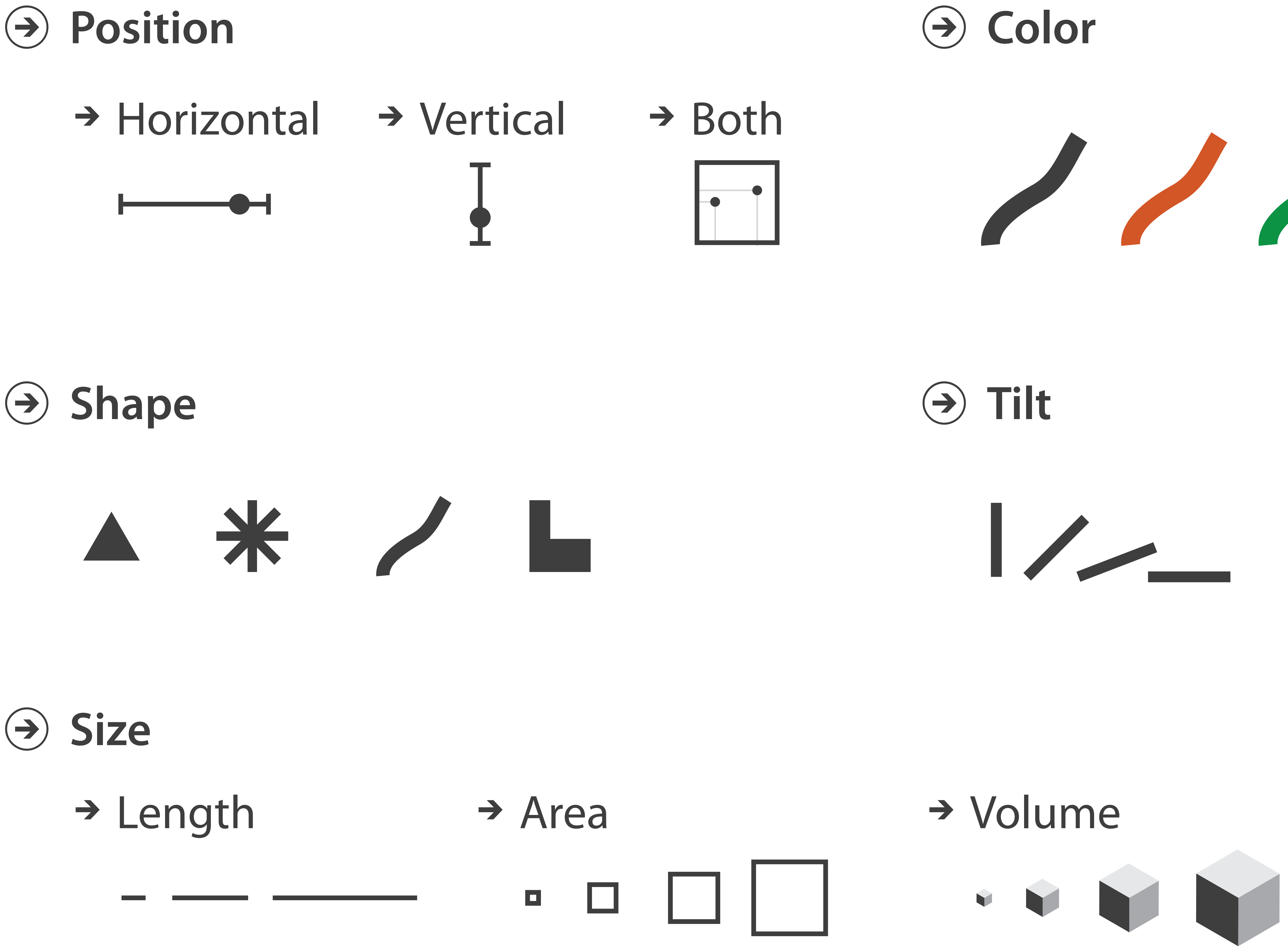
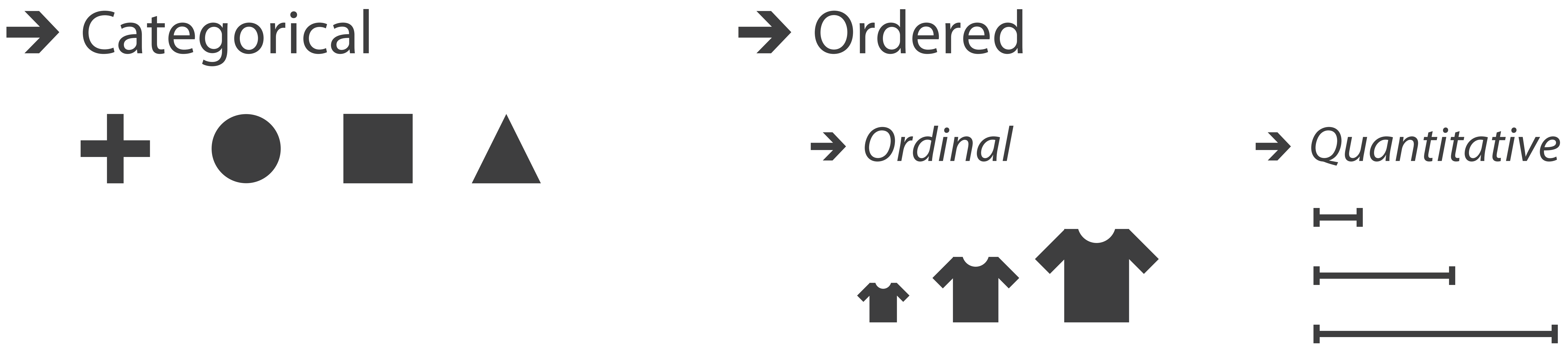
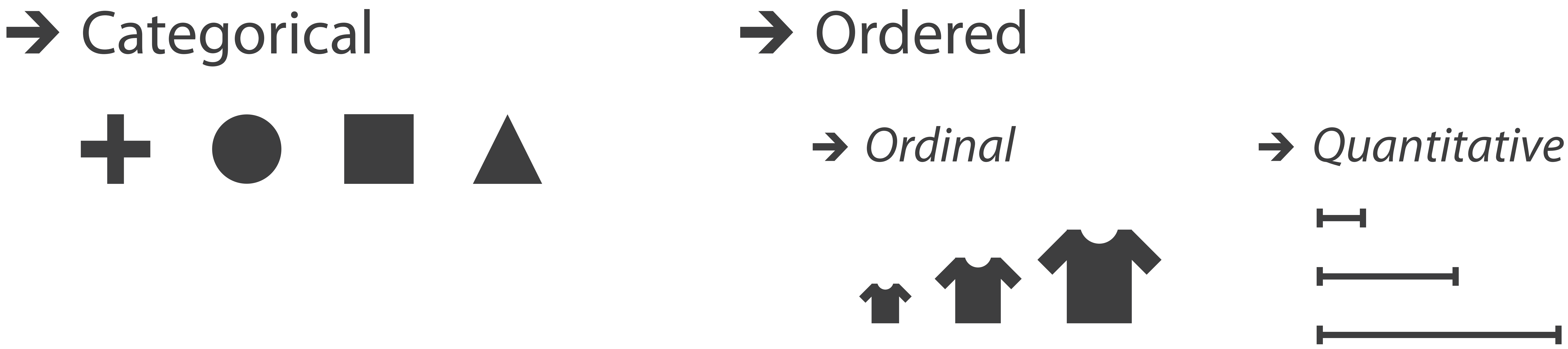
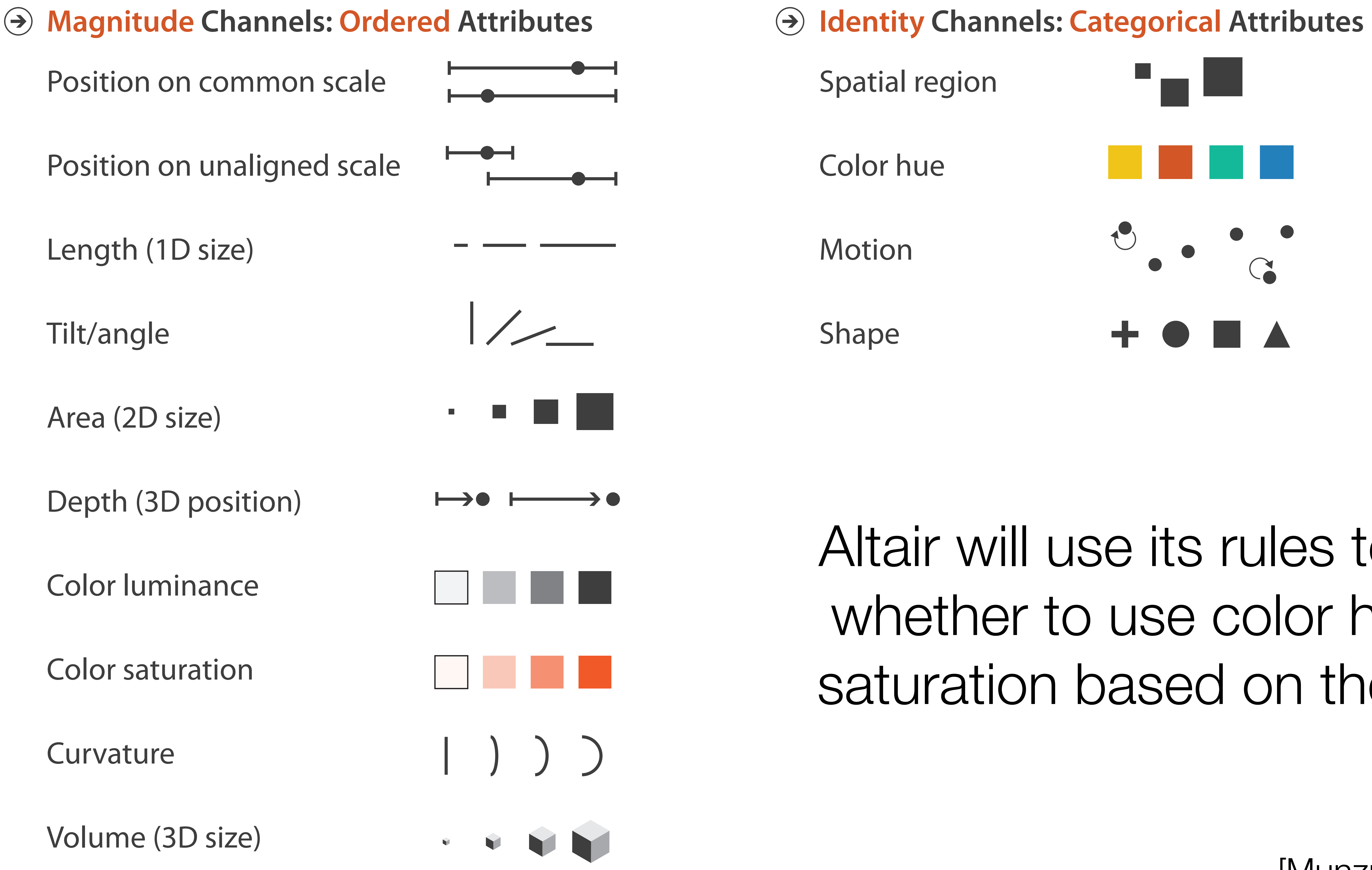
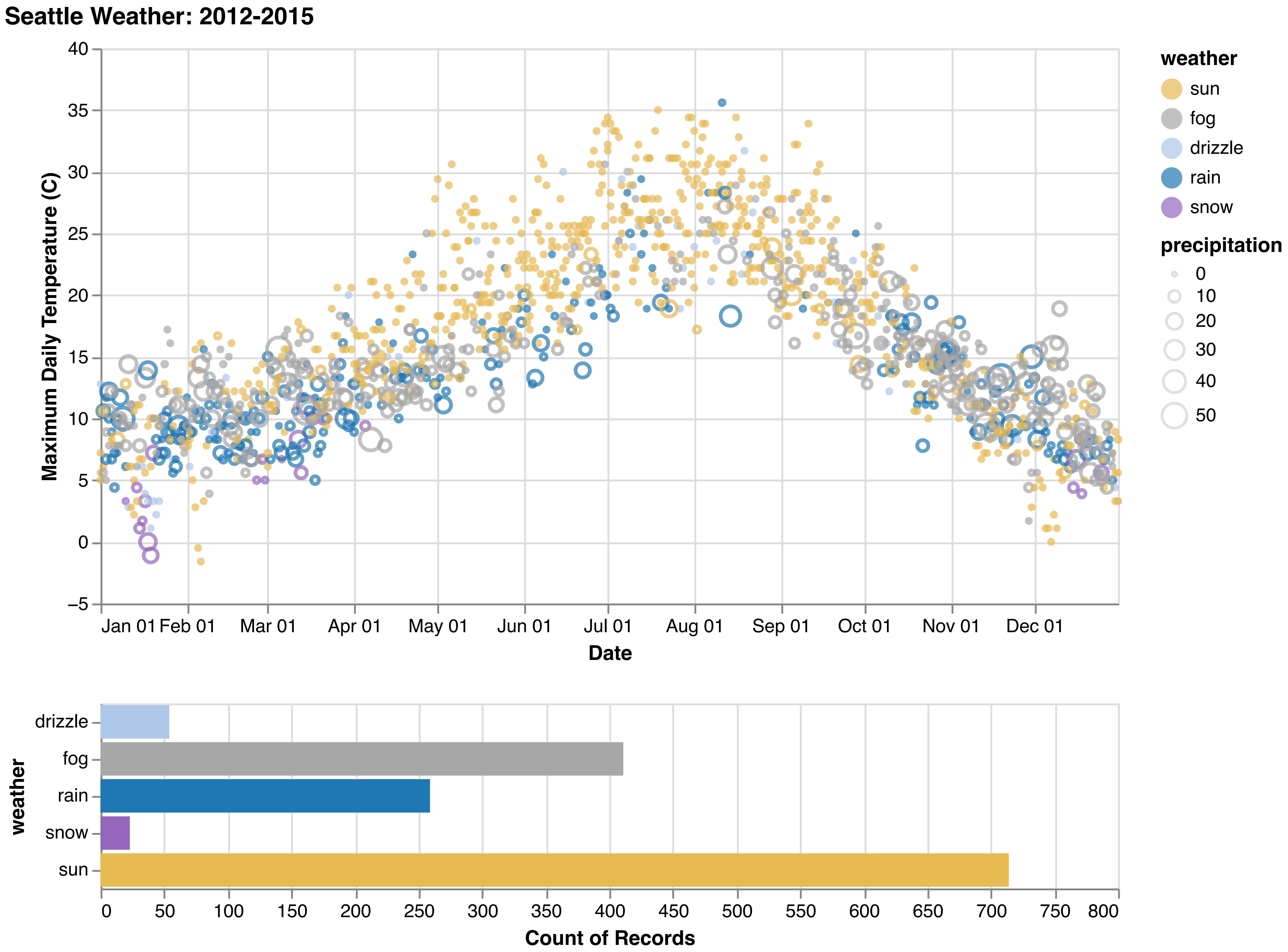
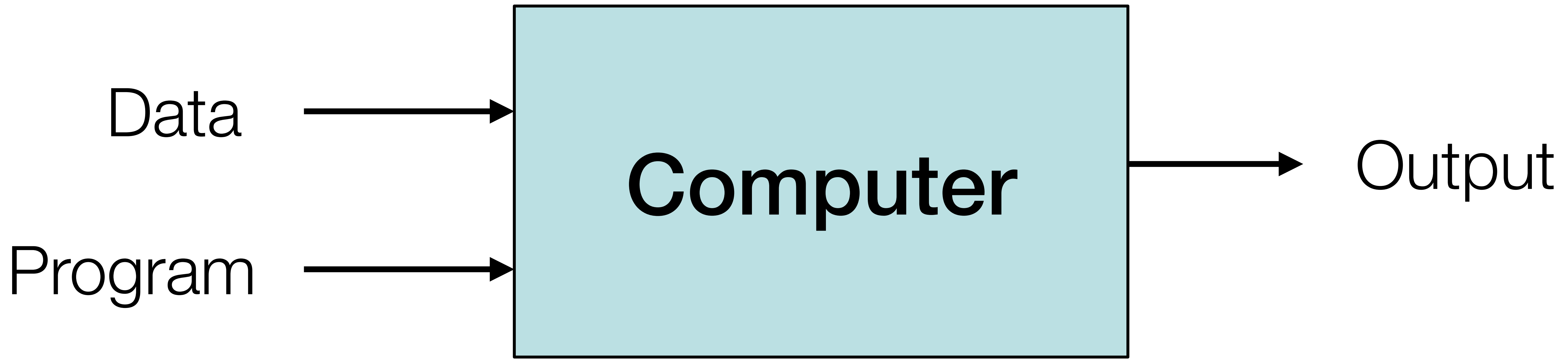
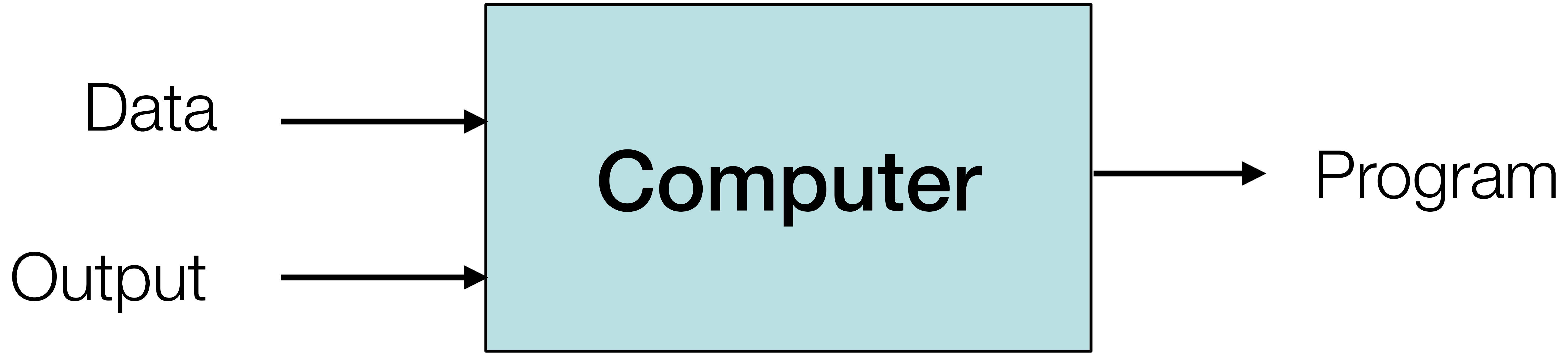
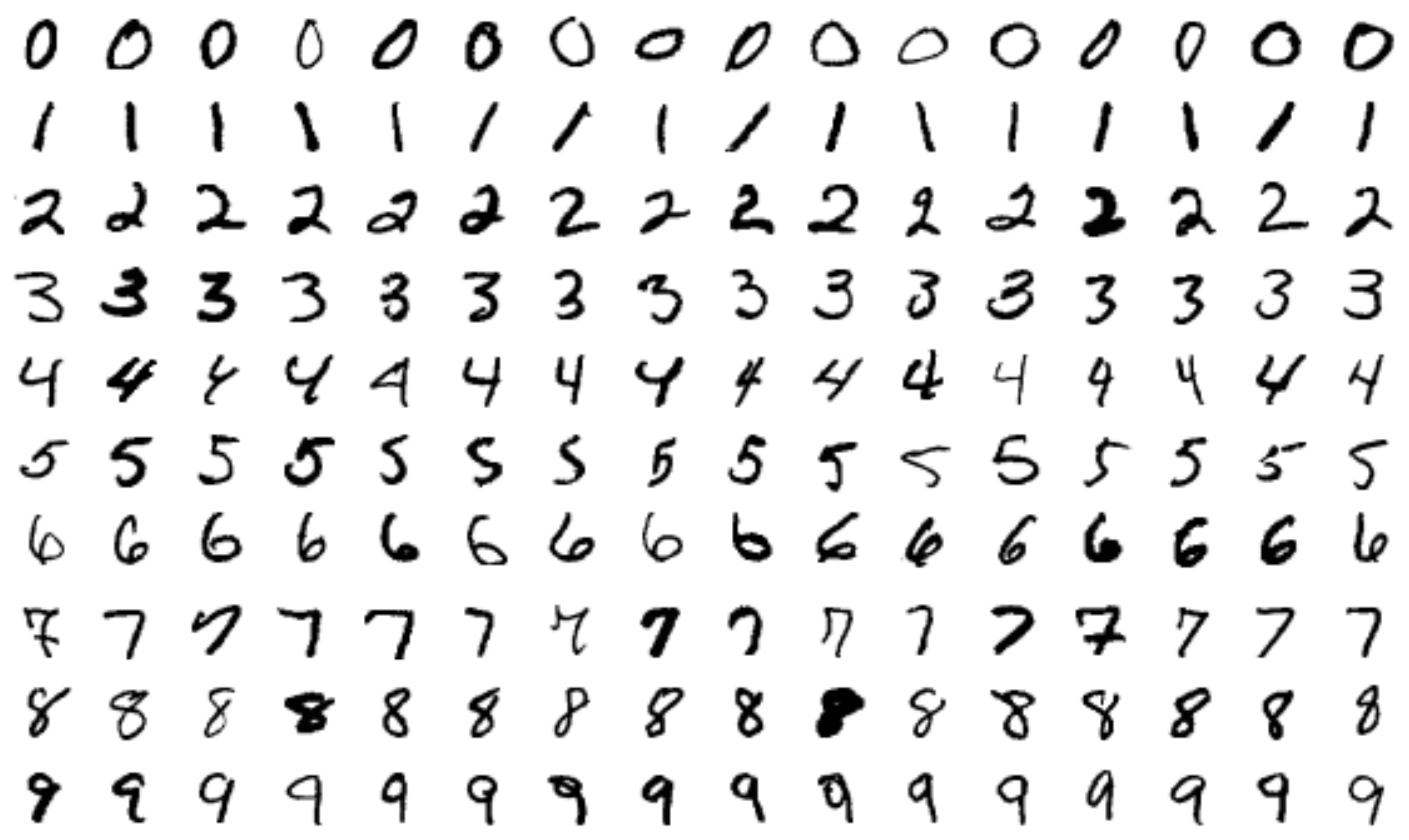
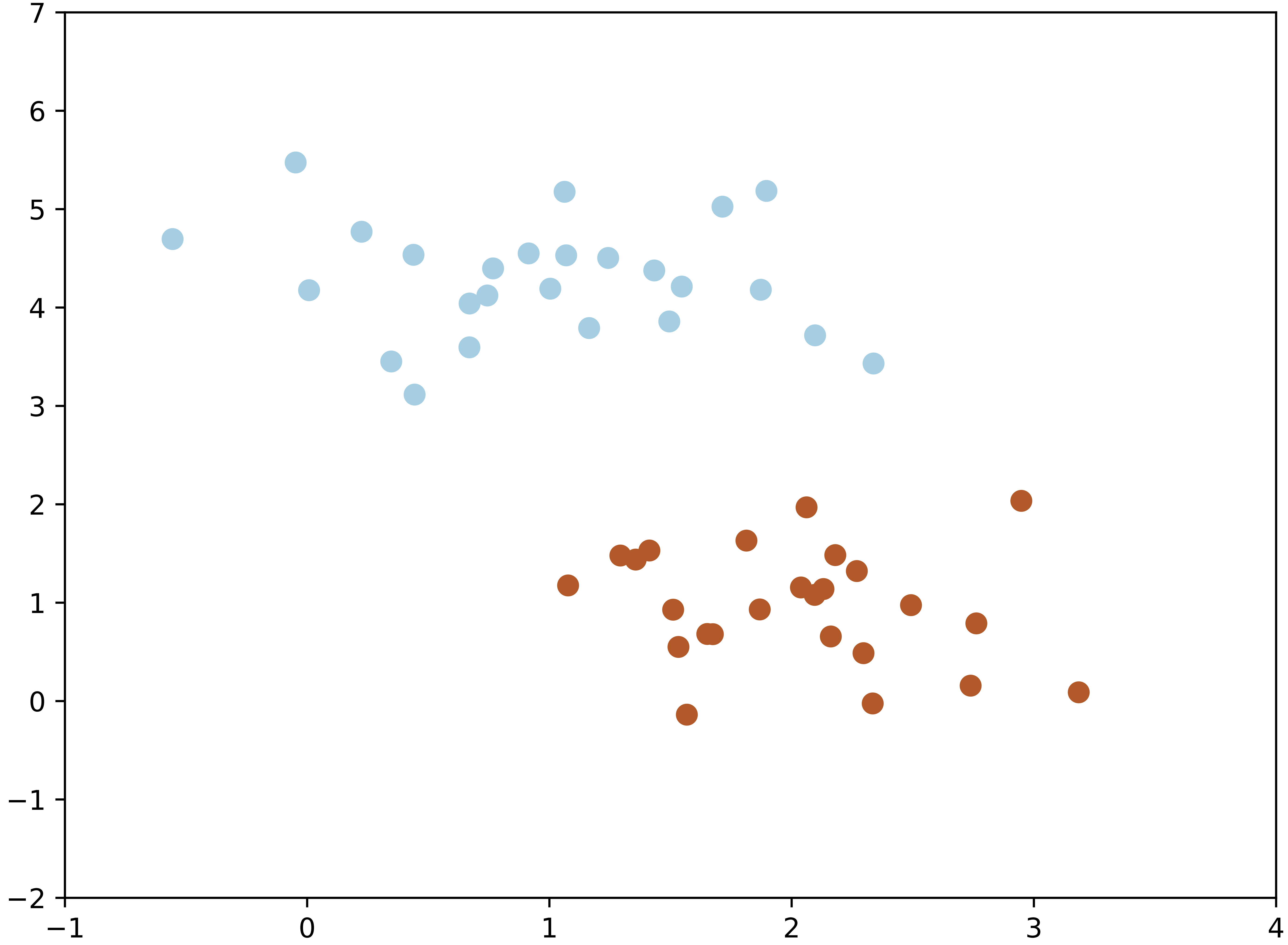
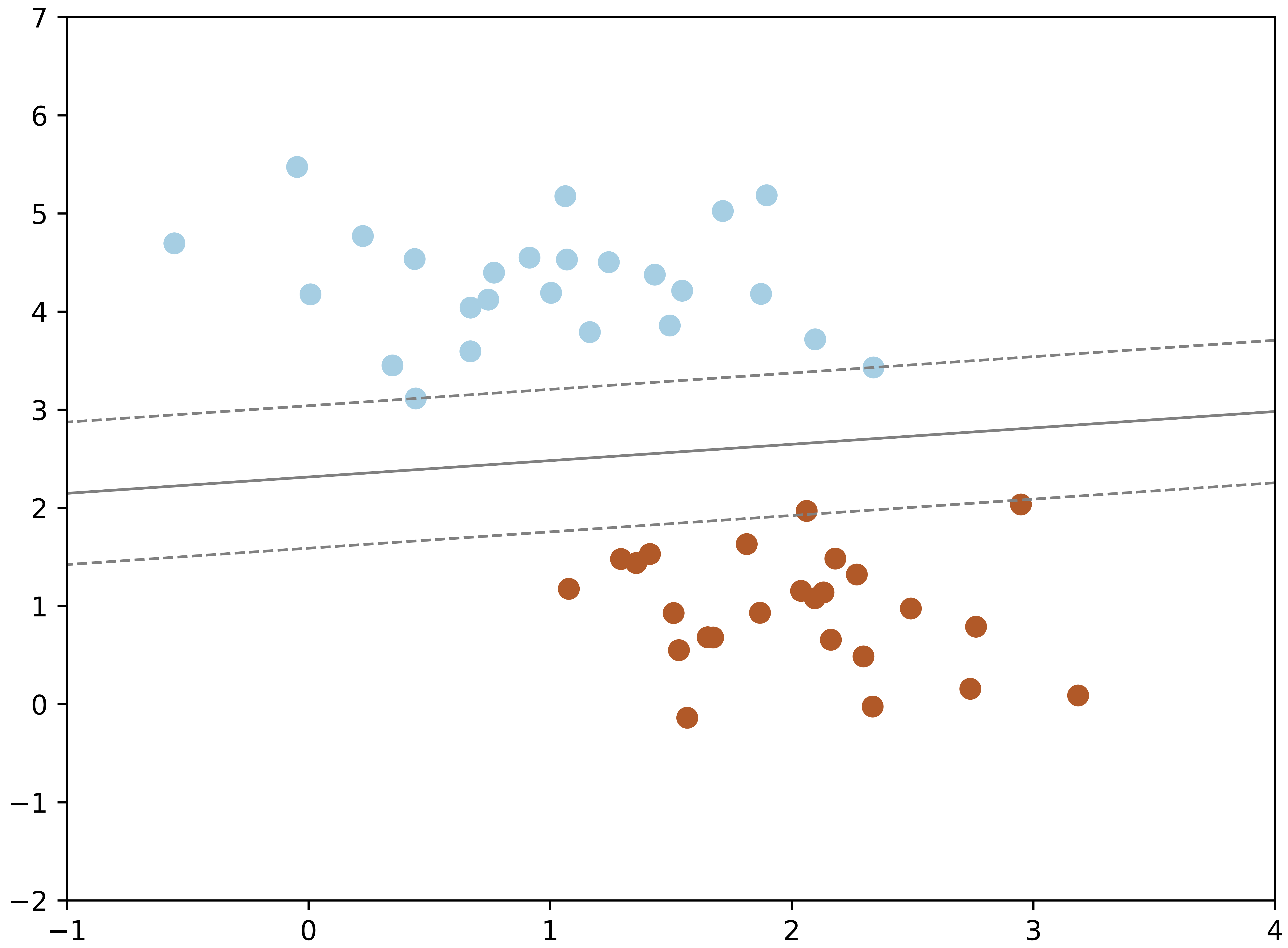
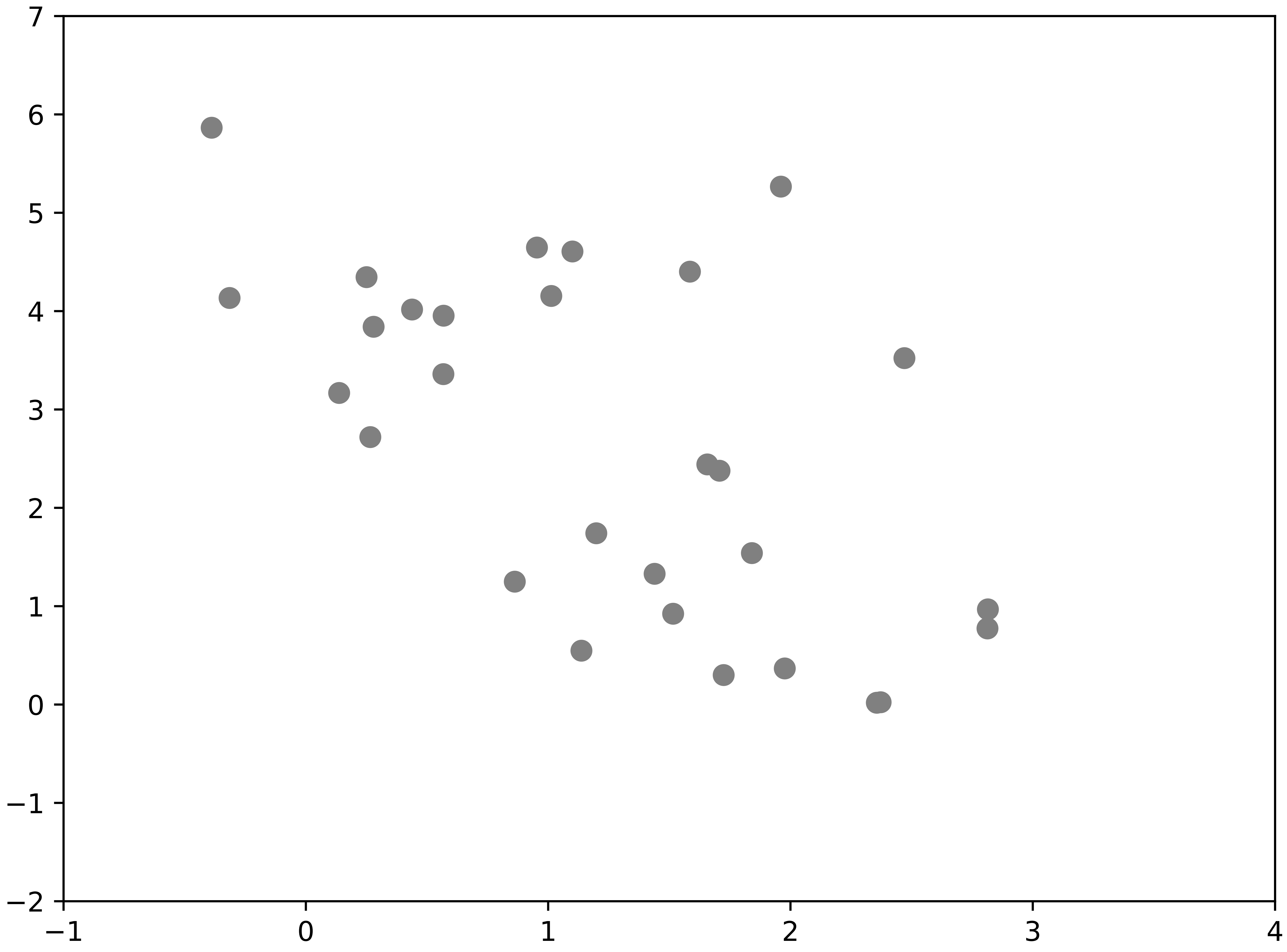
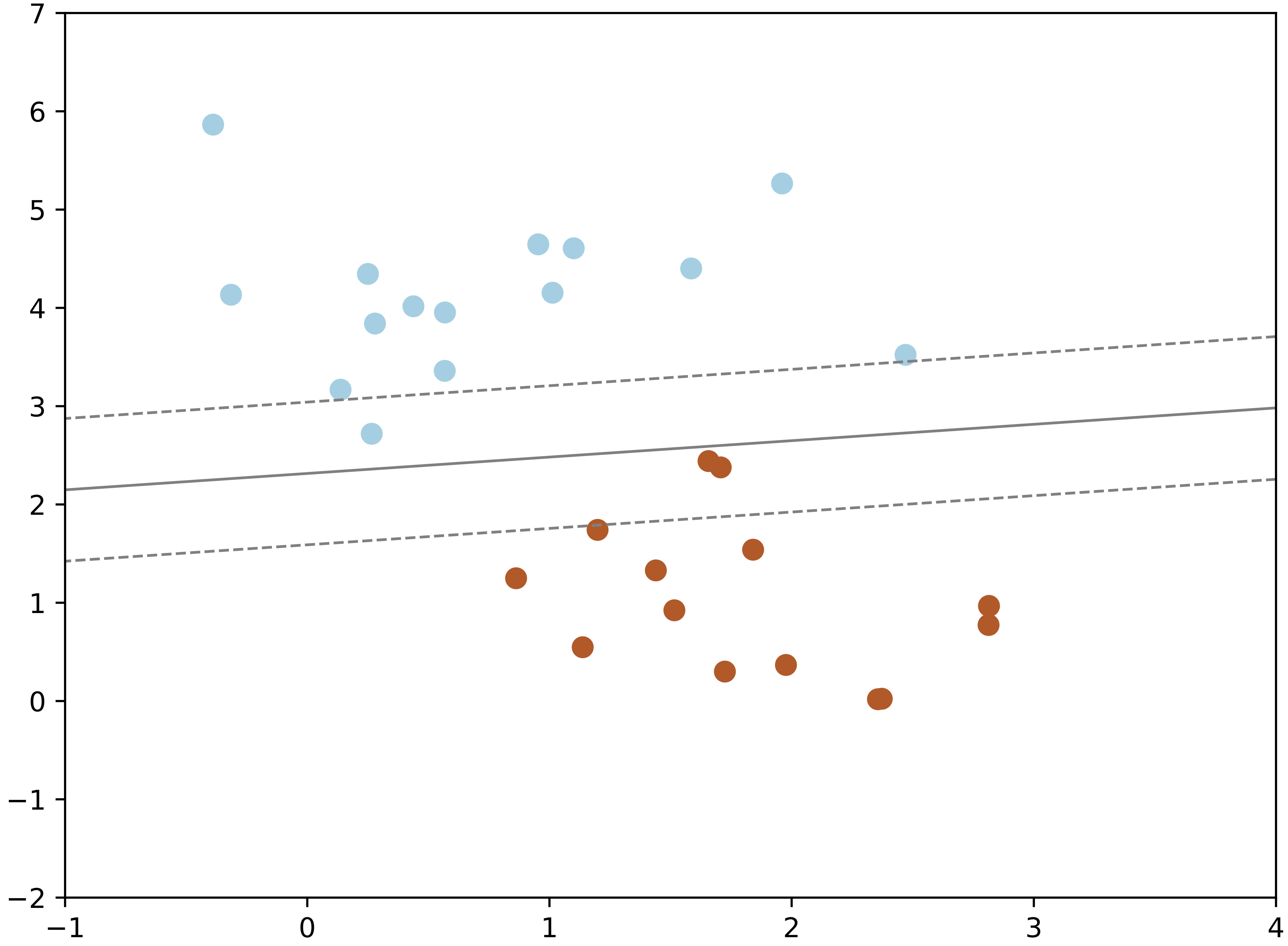
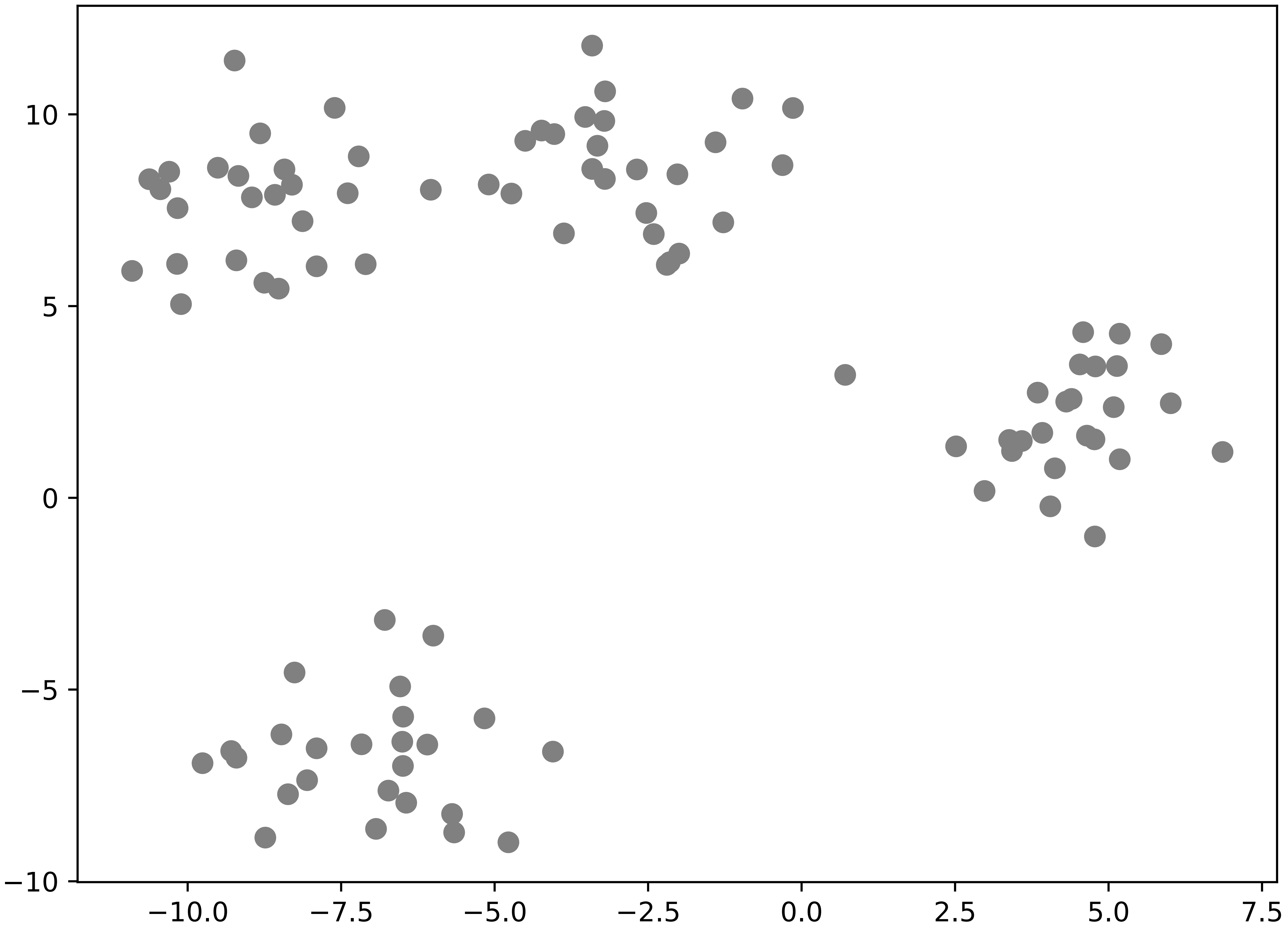
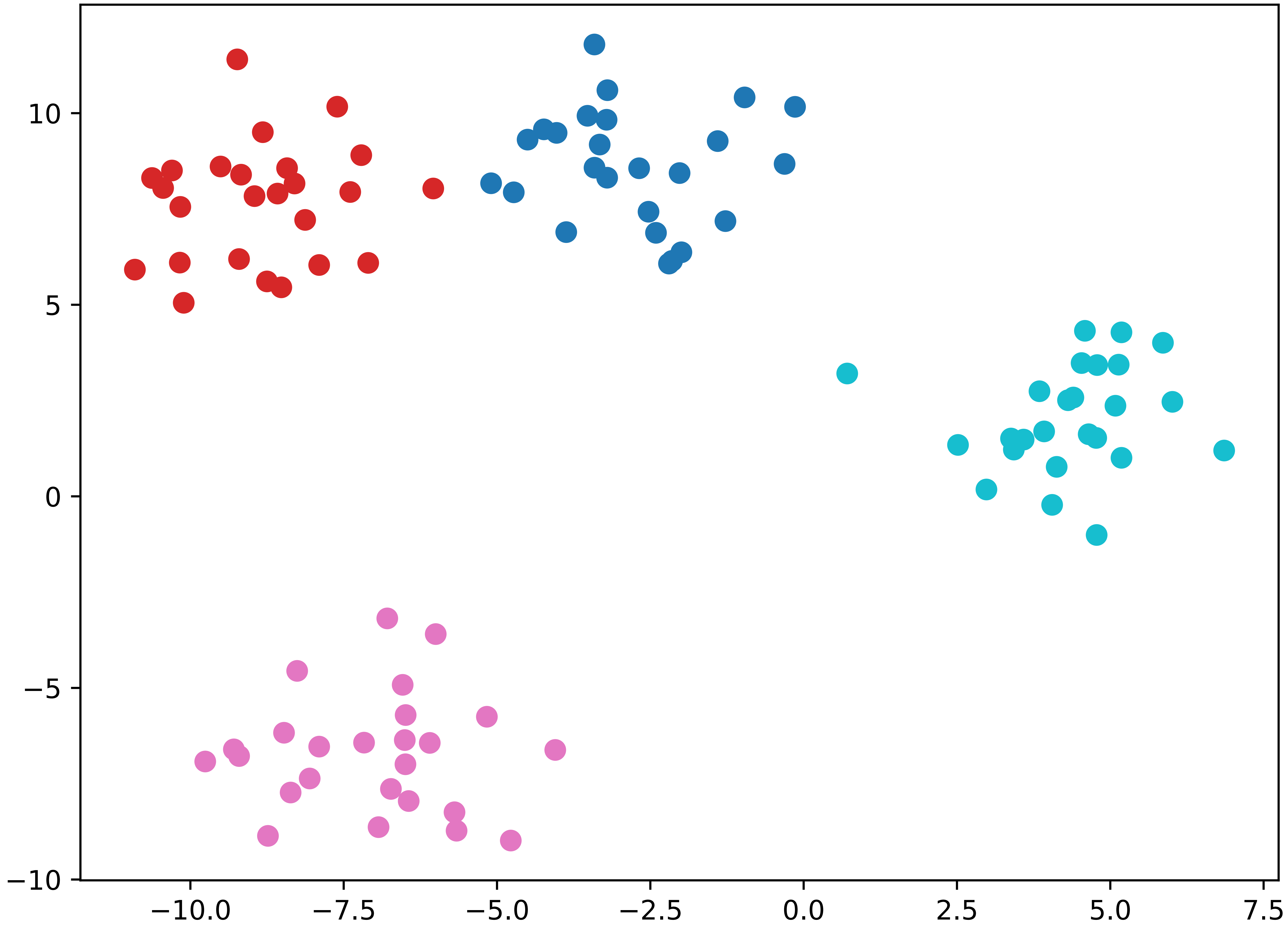
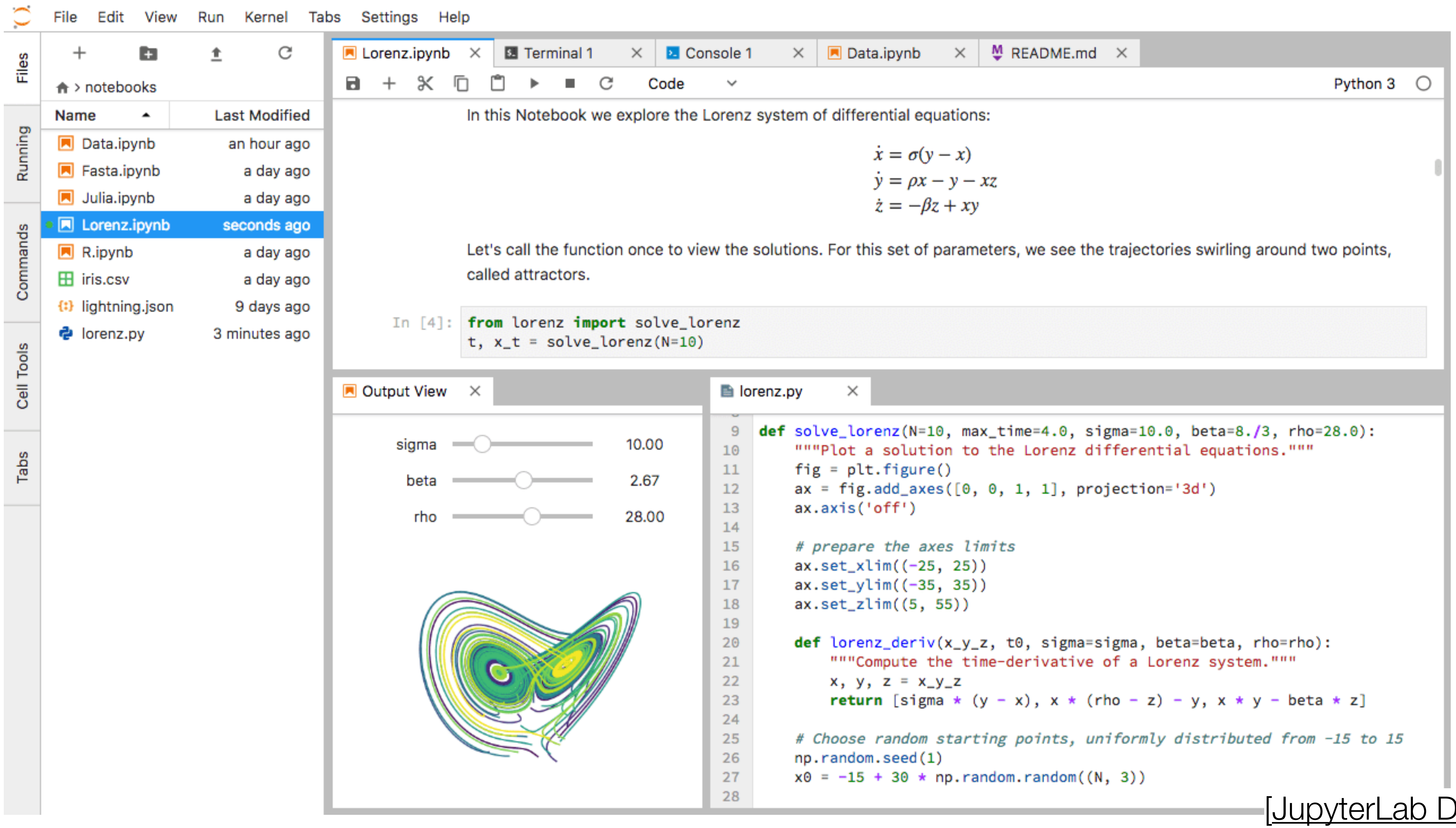
Machine Learning & Review

Dr. David Koop

Grammar of Graphics & Altair

- "Grammar of Graphics", L. Wilkinson
- "A Layered Grammar of Graphics" + ggplot, H. Wickham
- Vega: "Declarative language for creating, saving, and sharing interactive visualization designs"
- Vega-Lite: higher-level language than Vega, carefully crafted rules for defaults
- Altair: Python interface to Vega-Lite
  - "spend more time understanding your data and its meaning"
  - Specify the what, minimize the amount of code directing the how
  - Python can write JSON specification just as well as any other language
  - Bindings make it more Python-friendly, integrate with pandas, add support for Jupyter, etc.

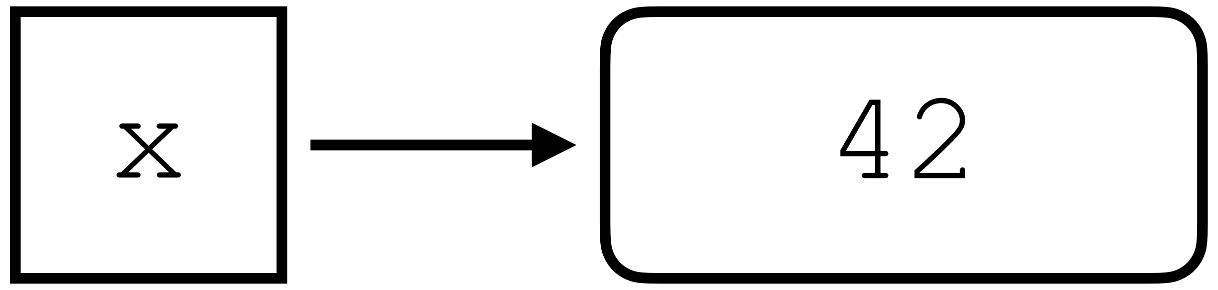
Basic Example

- ```
  import altair as alt
  import pandas as pd
  data = pd.DataFrame({'x': [1,3,4,6,10],'y': [1,5,2,7,3]})
  alt.Chart(data).mark_line().encode(x='x', y='y')
  ```
- Easiest to use data from a pandas data frame
  - Another option is a csv or json file
  - Can support geo_interface, too
- `Chart` is the basic unit
- Mark: `.mark_*()` indicates the geometry created for each data item
- Encode: `.encode()` allows visual properties to be set to data attributes
  ```

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

Encode via Visual Channels



[Munzner (ill. Maguire), 2014]

Data Attributes and Altair Types



[Munzner (ill. Maguire), 2014]

Data Attributes and Altair Types



- Categorical data = Nominal (N)
- Ordinal data = Ordinal (O)
- Quantitative data = Quantitative (Q)
- Temporal data = Temporal (T)

[Munzner (ill. Maguire), 2014]

Different Channels for Different Attribute Types



Altair will use its rules to pick whether to use color hue or saturation based on the type

[Munzner (ill. Maguire), 2014]

Interaction



Assignment 8

- Due Friday, Dec. 7
- Last Assignment
- Data and Visualization
- Same Illinois Unemployment Data
- Use polars or pandas
- Must use matplotlib or altair where directed

Final Exam

- Wednesday, Dec. 11, **8:00**-9:50am in PM 103
- **More comprehensive** than Test 2
- Expect questions from topics covered on Test 1 and 2
- Expect questions from the last few weeks of class (data, visualization, machine learning)
- Similar format

Machine Learning

- Traditional Programming



- Machine Learning



[P. Domingos]

Machine Learning

- Every machine learning algorithm has three components:
  - Representation
  - Evaluation
  - Optimization

Representation

- Decision trees
- Sets of rules / Logic programs
- Instances
- Graphical models (Bayes/Markov nets)
- Neural networks
- Support vector machines
- Model ensembles
- Etc.

[P. Domingos]

Evaluation

- Accuracy
- Precision and recall
- Squared error
- Likelihood
- Posterior probability
- Cost / Utility
- Margin
- Entropy
- K-L divergence
- Etc.

[P. Domingos]

Optimization

- Combinatorial optimization
  - E.g.: Greedy search
- Convex optimization
  - E.g.: Gradient descent
- Constrained optimization
  - E.g.: Linear programming

[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]

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]

Areas of Machine Learning

- Supervised learning
  - Decision tree induction
  - Rule induction
  - Instance-based learning
  - Bayesian learning
  - Neural networks
  - Support vector machines
  - Model ensembles
  - Learning theory
- Unsupervised learning
  - Clustering
  - Dimensionality reduction

[P. Domingos]

Supervised & Unsupervised Tasks

- Identifying the zip code from handwritten digits on an envelope (**supervised**)



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

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

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

- Started as a Google Summer of Code project! (D. Cournapeau, 2007)
- Rewritten by scientists at INRIA (France) in 2010
- Written in Python using numpy, some optimizations using C (cython)
- The "gold standard" for machine learning in python

[scikit-learn]

scikit-learn Principles

- Consistency: all objects share consistent, documented interface
- Inspection: parameters and parameter values determined by learning algorithms are stored and exposed as public attributes
- Non-proliferation of classes: only learning algs are classes, not datasets or parameters; easier to combine with other libraries
- Composition: create and reuse building blocks
- Sensible defaults: user-defined parameters should have meaningful defaults

[L. Buitinck et al.]

scikit-learn entities

- Data: numpy matrices (also pandas series, data frames), process batches
- Estimators: 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.]

Penguin Example

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

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)
    ```

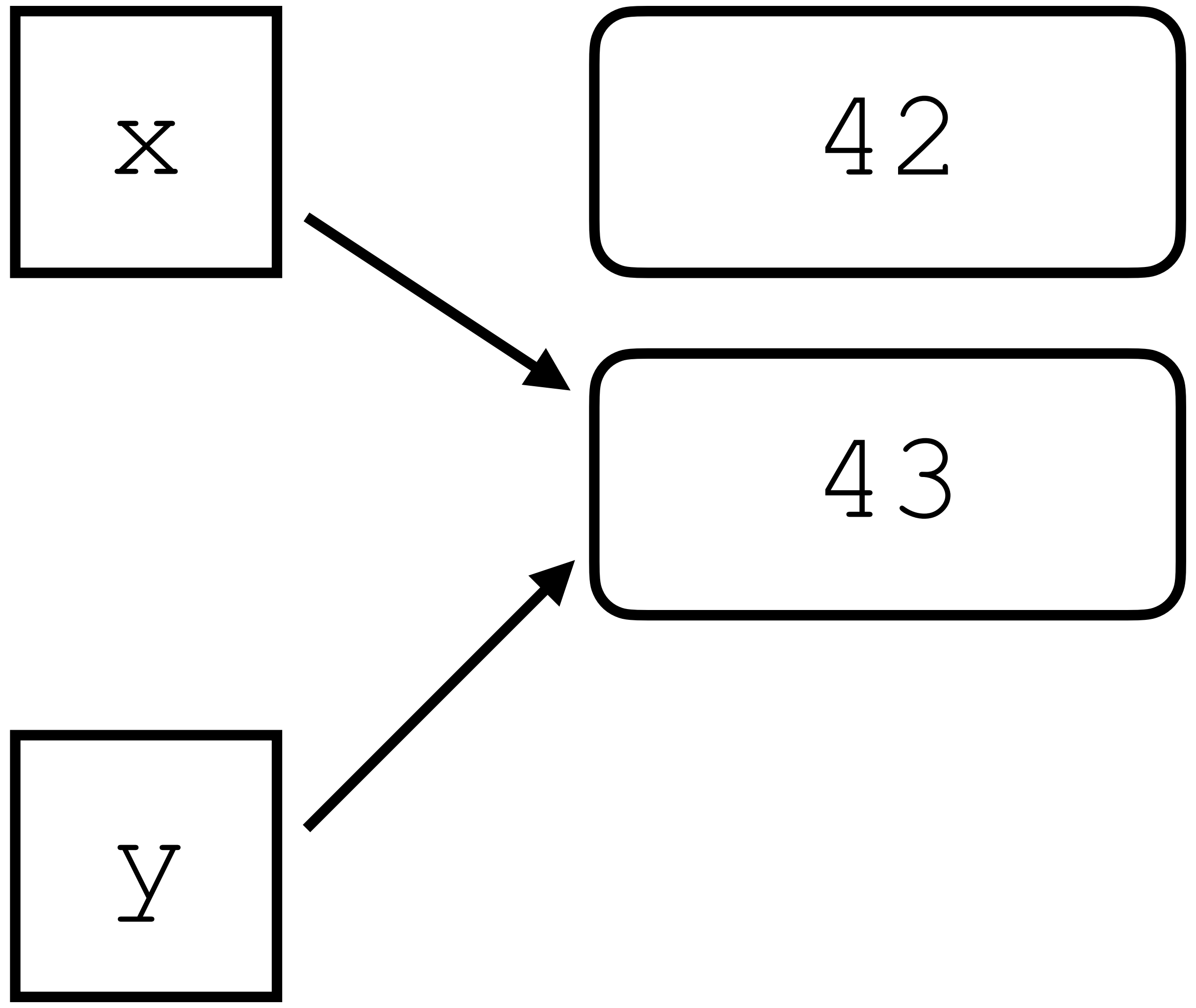
Assignment

- 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
```

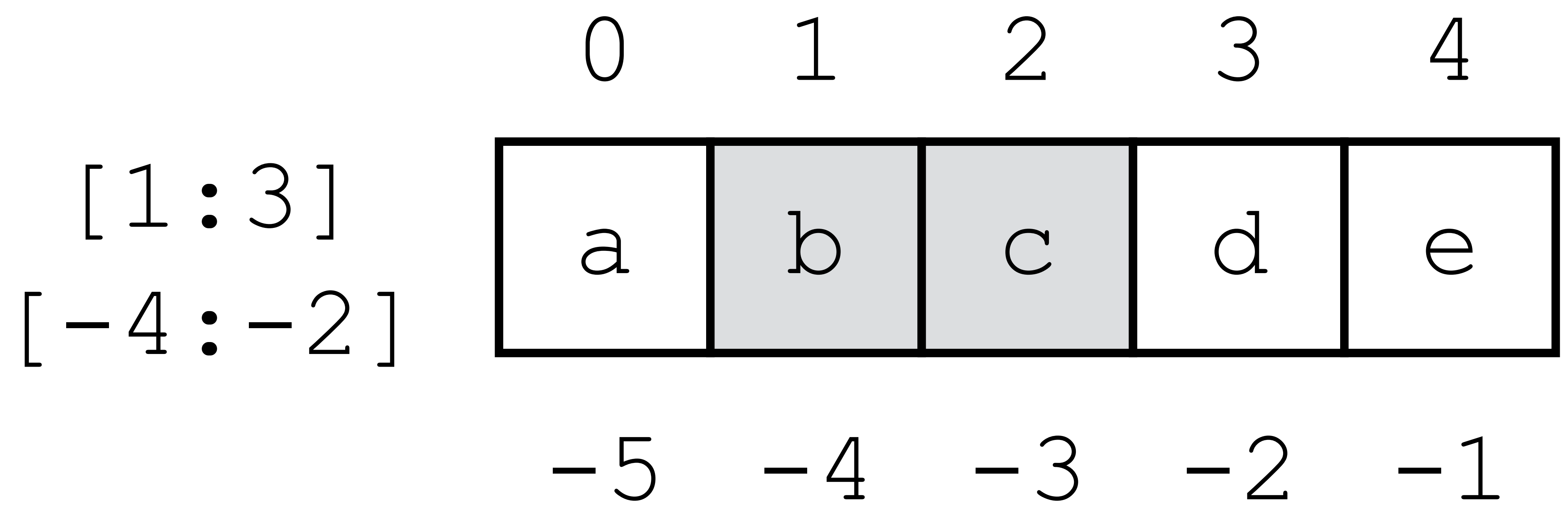
- Assignment Expression (Walrus)

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

Functions

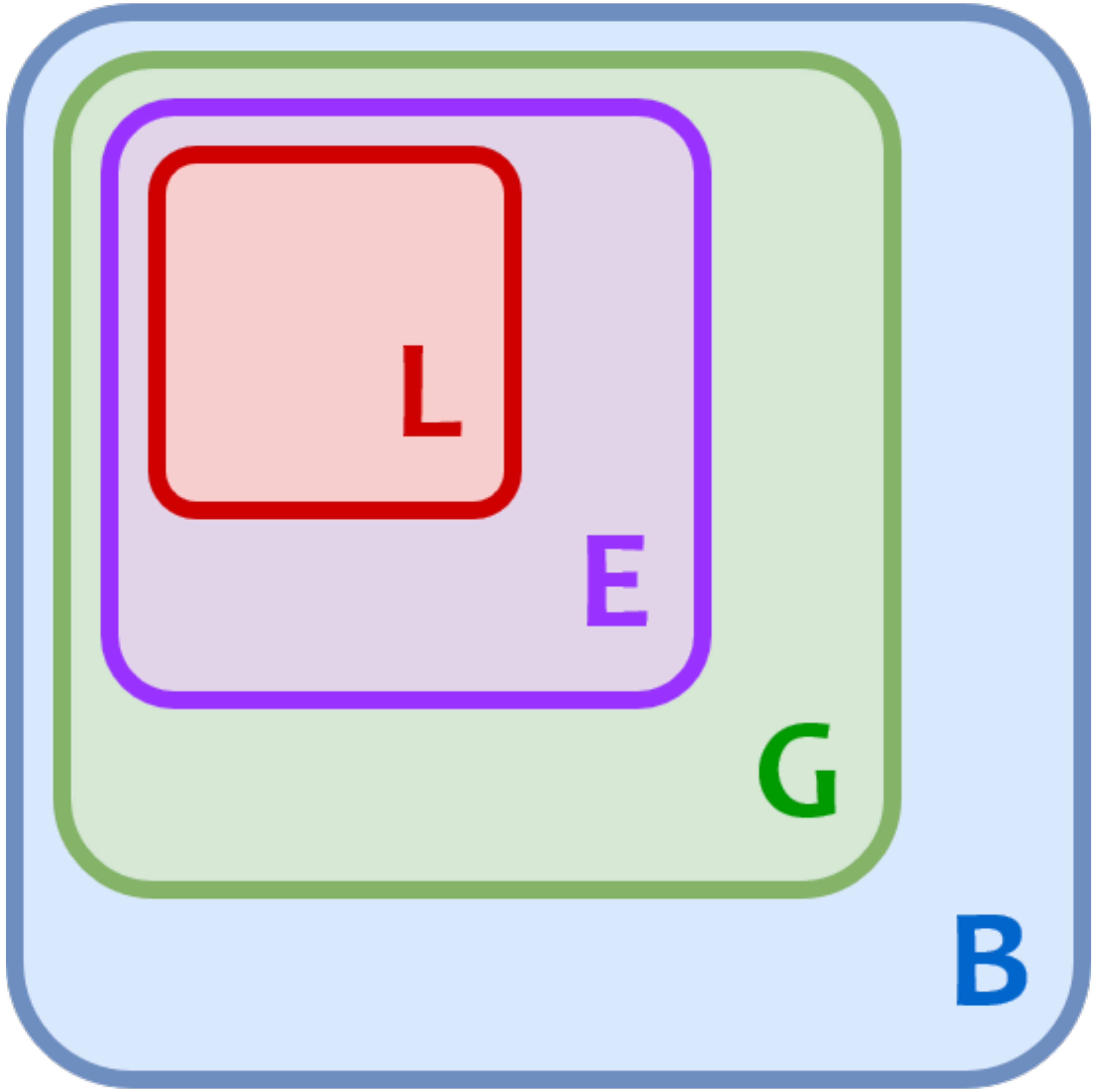
- Use `return` to return a value
-
```
def <function-name>(<parameter-names>):
    # do stuff
    return res
```
- Can return more than one value using commas
-
```
def <function-name>(<parameter-names>):
    # do stuff
    return res1, res2
```
- Use **simultaneous assignment** when calling:
    - `a, 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]

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):`

```
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

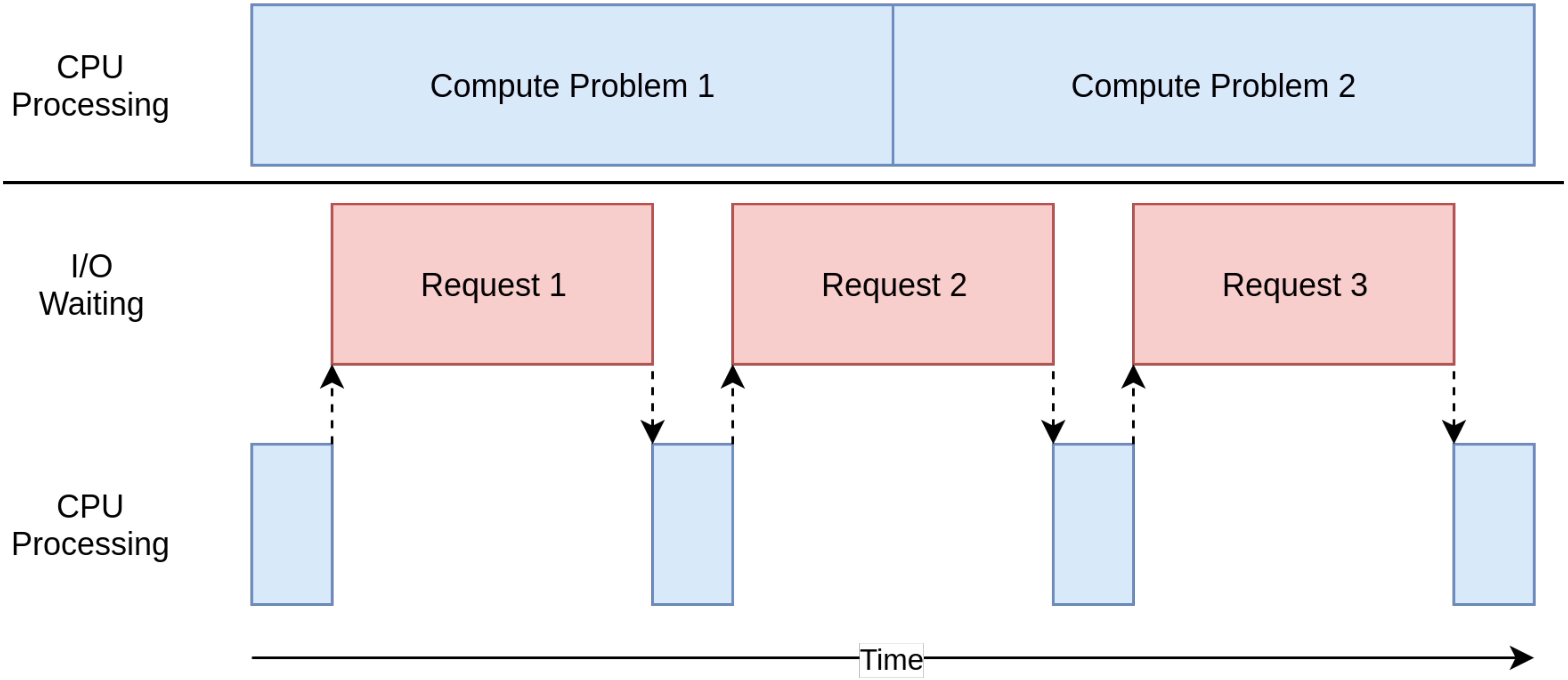
- Dynamic Typing: variable's type can change (what Python does)
- Static Typing: compiler enforces types, variable types generally don't change
- Duck Typing: check method/attribute existence, not type
- Python is a dynamically-typed language (and plans to remain so)
- …but it has recently added more 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]

Structural Pattern Matching

- Besides literal cases, match statements can be used to
  - differentiate structure
  - assign values
  - differentiate class instances
- Example:
- ```
  match sys.argv:
      case [_, "commit"]:
          print("Committing")
      case [_, 'add', fname]:
          print("Adding file", fname)
  ```
  ```

Patterns

- Sequence Pattern:

```
match sys.argv:
    case [_, "commit"]:
        print("Committing")
    case [_, 'add', *fnames]:
        print("Adding files", fnames)
```

- Or and As Pattern:

```
match command.split():
    case ["go", ("north" | "south" | "east" | "west") as d]:
        current_room = current_room.neighbor(d)
```

- Mapping Pattern
- Class Pattern

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`

pandas DataFrame

```
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

pandas DataFrame

```
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

pandas DataFrame

```
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

pandas DataFrame

```
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']`

pandas DataFrame

```
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']`

pandas DataFrame

```
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
	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']`

pandas DataFrame

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

Column Names

Row: `df.loc[2]`

Index

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

Missing Data

	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

Column: `df['Island']`

polars DataFrame

shape: (344, 10)

studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
str	i64	str	str	str	str	str	str	str	f64
"PAL0708"	1	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N1A1"	"Yes"	"11/11/07"	39.1
"PAL0708"	2	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N1A2"	"Yes"	"11/11/07"	39.5
"PAL0708"	3	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N2A1"	"Yes"	"11/16/07"	40.3
"PAL0708"	4	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N2A2"	"Yes"	"11/16/07"	null
"PAL0708"	5	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N3A1"	"Yes"	"11/16/07"	36.7
…	…	…	…	…	…	…	…	…	…
"PAL0910"	120	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N38A2"	"No"	"12/1/09"	null
"PAL0910"	121	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N39A1"	"Yes"	"11/22/09"	46.8
"PAL0910"	122	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N39A2"	"Yes"	"11/22/09"	50.4
"PAL0910"	123	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N43A1"	"Yes"	"11/22/09"	45.2
"PAL0910"	124	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N43A2"	"Yes"	"11/22/09"	49.9

polars DataFrame

Column Names & Types

shape: (344, 10)

studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
str	i64	str	str	str	str	str	str	str	f64
"PAL0708"	1	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N1A1"	"Yes"	"11/11/07"	39.1
"PAL0708"	2	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N1A2"	"Yes"	"11/11/07"	39.5
"PAL0708"	3	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N2A1"	"Yes"	"11/16/07"	40.3
"PAL0708"	4	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N2A2"	"Yes"	"11/16/07"	null
"PAL0708"	5	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N3A1"	"Yes"	"11/16/07"	36.7
…	…	…	…	…	…	…	…	…	…
"PAL0910"	120	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N38A2"	"No"	"12/1/09"	null
"PAL0910"	121	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N39A1"	"Yes"	"11/22/09"	46.8
"PAL0910"	122	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N39A2"	"Yes"	"11/22/09"	50.4
"PAL0910"	123	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N43A1"	"Yes"	"11/22/09"	45.2
"PAL0910"	124	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N43A2"	"Yes"	"11/22/09"	49.9

polars DataFrame

Column Names & Types

shape: (344, 10)

studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
str	i64	str	str	str	str	str	str	str	f64
"PAL0708"	1	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N1A1"	"Yes"	"11/11/07"	39.1
"PAL0708"	2	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N1A2"	"Yes"	"11/11/07"	39.5
"PAL0708"	3	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N2A1"	"Yes"	"11/16/07"	40.3
"PAL0708"	4	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N2A2"	"Yes"	"11/16/07"	null
"PAL0708"	5	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N3A1"	"Yes"	"11/16/07"	36.7
…	…	…	…	…	…	…	…	…	…
"PAL0910"	120	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N38A2"	"No"	"12/1/09"	null
"PAL0910"	121	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N39A1"	"Yes"	"11/22/09"	46.8
"PAL0910"	122	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N39A2"	"Yes"	"11/22/09"	50.4
"PAL0910"	123	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N43A1"	"Yes"	"11/22/09"	45.2
"PAL0910"	124	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"					49.9

Column: `df['Island']`

polars DataFrame

Column Names & Types

Row: `df[2]`

shape: (344, 10)

studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
str	i64	str	str	str	str	str	str	str	f64
"PAL0708"	1	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N1A1"	"Yes"	"11/11/07"	39.1
"PAL0708"	2	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N1A2"	"Yes"	"11/11/07"	39.5
"PAL0708"	3	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N2A1"	"Yes"	"11/16/07"	40.3
"PAL0708"	4	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N2A2"	"Yes"	"11/16/07"	null
"PAL0708"	5	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N3A1"	"Yes"	"11/16/07"	36.7
…	…	…	…	…	…	…	…	…	…
"PAL0910"	120	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N38A2"	"No"	"12/1/09"	null
"PAL0910"	121	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N39A1"	"Yes"	"11/22/09"	46.8
"PAL0910"	122	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N39A2"	"Yes"	"11/22/09"	50.4
"PAL0910"	123	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N43A1"	"Yes"	"11/22/09"	45.2
"PAL0910"	124	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"					49.9

Column: `df['Island']`

polars DataFrame

Column Names & Types

shape: (344, 10)

studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
str	i64	str	str	str	str	str	str	str	f64
"PAL0708"	1	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N1A1"	"Yes"	"11/11/07"	39.1
"PAL0708"	2	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N1A2"	"Yes"	"11/11/07"	39.5
"PAL0708"	3	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N2A1"	"Yes"	"11/16/07"	40.3
"PAL0708"	4	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N2A2"	"Yes"	"11/16/07"	null
"PAL0708"	5	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N3A1"	"Yes"	"11/16/07"	36.7
…	…	…	…	…	…	…	…	…	…
"PAL0910"	120	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N38A2"	"No"	"12/1/09"	null
		Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N39A1"	"Yes"	"11/22/09"	46.8
"PAL0910"	122	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N39A2"	"Yes"	"11/22/09"	50.4
"PAL0910"	123	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N43A1"	"Yes"	"11/22/09"	45.2
"PAL0910"	124	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"					49.9

Row: `df[2]`

Cell: `df['Species'][341]`

Column: `df['Island']`

polars DataFrame

Column Names & Types

Row: `df[2]`

Cell: `df['Species'][341]`

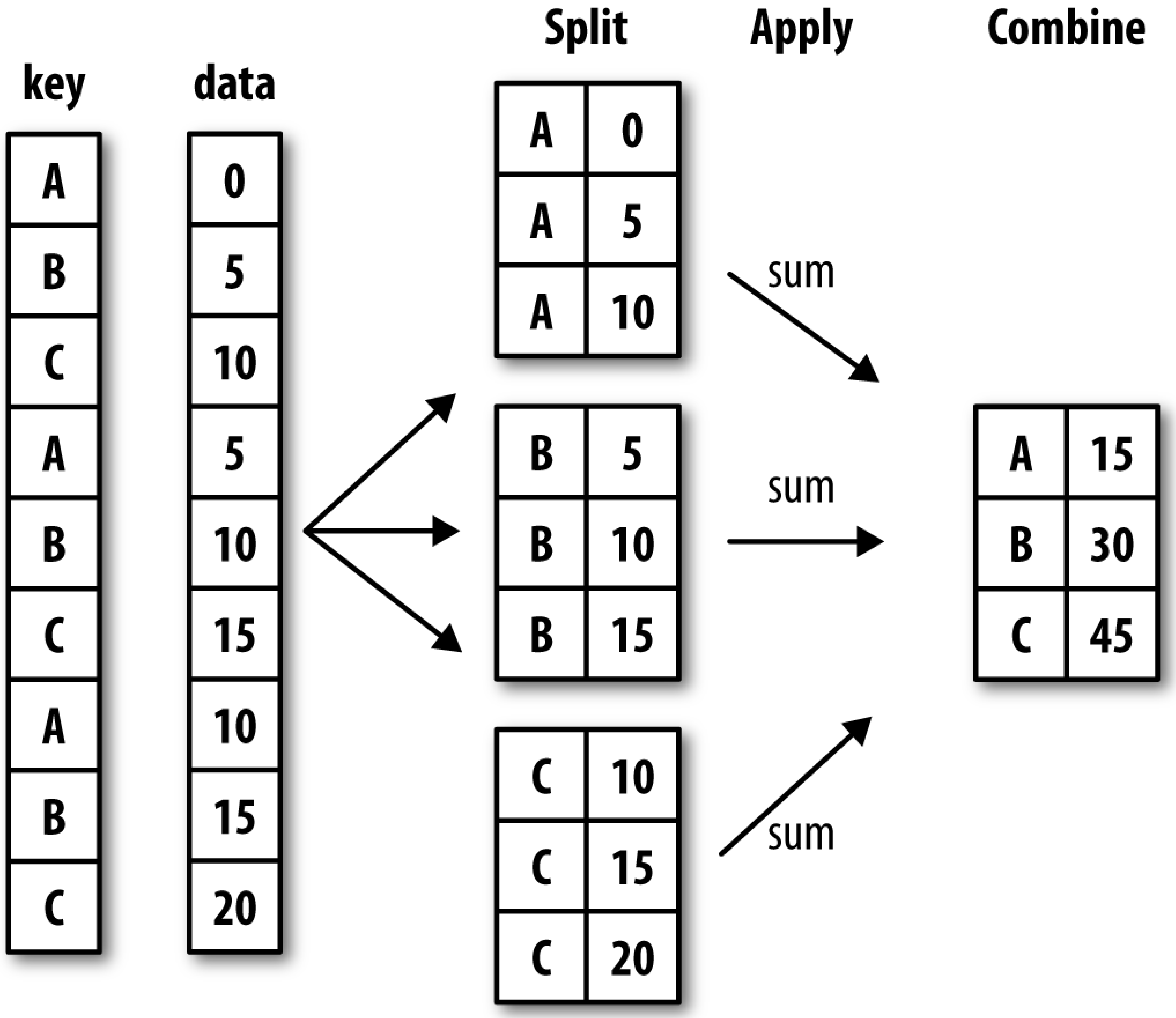
Missing Data

Column: `df['Island']`

shape: (344, 10)

studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)
str	i64	str	str	str	str	str	str	str	f64
"PAL0708"	1	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N1A1"	"Yes"	"11/11/07"	39.1
"PAL0708"	2	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N1A2"	"Yes"	"11/11/07"	39.5
"PAL0708"	3	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N2A1"	"Yes"	"11/16/07"	40.3
"PAL0708"	4	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N2A2"	"Yes"	"11/16/07"	null
"PAL0708"	5	"Adelie Penguin (Pygoscelis ade…	"Anvers"	"Torgersen"	"Adult, 1 Egg Stage"	"N3A1"	"Yes"	"11/16/07"	
…	…	…	…	…	…	…	…	…	…
"PAL0910"	120	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N38A2"	"No"	"12/1/09"	null
		Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N39A1"	"Yes"	"11/22/09"	46.8
"PAL0910"	122	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N39A2"	"Yes"	"11/22/09"	50.4
"PAL0910"	123	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"	"Adult, 1 Egg Stage"	"N43A1"	"Yes"	"11/22/09"	45.2
"PAL0910"	124	"Gentoo penguin (Pygoscelis pap…	"Anvers"	"Biscoe"					49.9

Aggregation: Split-Apply-Combine



[W. McKinney, Python for Data Analysis]

Unpivot/Melt

- Many columns (wider) become two columns (longer): one with column name (variable), other with value

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
str	i32	i32	str	f64	f64	f64	f64	f64	f64	f64	f64
"MX000017004"	1955	4	"tmax"	31.0	31.0	31.0	32.0	33.0	32.0	32.0	33.0
"MX000017004"	1955	4	"tmin"	15.0	15.0	16.0	15.0	16.0	16.0	16.0	16.0
"MX000017004"	1955	5	"tmax"	31.0	31.0	31.0	30.0	30.0	30.0	31.0	31.0
"MX000017004"	1955	5	"tmin"	20.0	16.0	16.0	15.0	15.0	15.0	16.0	16.0
"MX000017004"	1955	6	"tmax"	30.0	29.0	28.0	27.0	28.0	26.0	23.0	27.0
…	…	…	…	…	…	…	…	…	…	…	…
"MX000017004"	2011	2	"tmin"	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
"MX000017004"	2011	3	"tmax"	NaN	NaN	NaN	NaN	33.2	NaN	NaN	NaN
"MX000017004"	2011	3	"tmin"	NaN	NaN	NaN	NaN	14.8	NaN	NaN	NaN
"MX000017004"	2011	4	"tmax"	NaN	35.0	NaN	NaN	NaN	NaN	NaN	NaN
"MX000017004"	2011	4	"tmin"	NaN	16.8	NaN	NaN	NaN	NaN	NaN	NaN

id	year	month	element	variable	value
str	i32	i32	str	str	f64
"MX000017004"	1955	4	"tmax"	"d1"	31.0
"MX000017004"	1955	4	"tmin"	"d1"	15.0
"MX000017004"	1955	5	"tmax"	"d1"	31.0
"MX000017004"	1955	5	"tmin"	"d1"	20.0
"MX000017004"	1955	6	"tmax"	"d1"	30.0
…	…	…	…	…	…
"MX000017004"	2011	2	"tmin"	"d31"	NaN
"MX000017004"	2011	3	"tmax"	"d31"	36.5
"MX000017004"	2011	3	"tmin"	"d31"	17.0
"MX000017004"	2011	4	"tmax"	"d31"	NaN
"MX000017004"	2011	4	"tmin"	"d31"	NaN

Unpivot/Melt

- Many columns (wider) become two columns (longer): one with column name (variable), other with value

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
str	i32	i32	str	f64	f64	f64	f64	f64	f64	f64	f64
"MX000017004"	1955	4	"tmax"	31.0	31.0	31.0	32.0	33.0	32.0	32.0	33.0
"MX000017004"	1955	4	"tmin"	15.0	15.0	16.0	15.0	16.0	16.0	16.0	16.0
"MX000017004"	1955	5	"tmax"	31.0	31.0	31.0	30.0	30.0	30.0	31.0	31.0
"MX000017004"	1955	5	"tmin"	20.0	16.0	16.0	15.0	15.0	15.0	16.0	16.0
"MX000017004"	1955	6	"tmax"	30.0	29.0	28.0	27.0	28.0	26.0	23.0	27.0
…	…	…	…	…	…	…	…	…	…	…	…
"MX000017004"	2011	2	"tmin"	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
"MX000017004"	2011	3	"tmax"	NaN	NaN	NaN	NaN	33.2	NaN	NaN	NaN
"MX000017004"	2011	3	"tmin"	NaN	NaN	NaN	NaN	14.8	NaN	NaN	NaN
"MX000017004"	2011	4	"tmax"	NaN	35.0	NaN	NaN	NaN	NaN	NaN	NaN
"MX000017004"	2011	4	"tmin"	NaN	16.8	NaN	NaN	NaN	NaN	NaN	NaN

id	year	month	element	variable	value
str	i32	i32	str	str	f64
"MX000017004"	1955	4	"tmax"	"d1"	31.0
"MX000017004"	1955	4	"tmin"	"d1"	15.0
"MX000017004"	1955	5	"tmax"	"d1"	31.0
"MX000017004"	1955	5	"tmin"	"d1"	20.0
"MX000017004"	1955	6	"tmax"	"d1"	30.0
…	…	…	…	…	…
"MX000017004"	2011	2	"tmin"	"d31"	NaN
"MX000017004"	2011	3	"tmax"	"d31"	36.5
"MX000017004"	2011	3	"tmin"	"d31"	17.0
"MX000017004"	2011	4	"tmax"	"d31"	NaN
"MX000017004"	2011	4	"tmin"	"d31"	NaN

Pivot

- Inverse of unpivot: two columns (longer) become many columns (wider) one column becomes column names (variable), other becomes values

id	year	month	element	variable	value
str	i32	i32	str	str	f64
"MX000017004"	1955	4	"tmax"	"d1"	31.0
"MX000017004"	1955	4	"tmin"	"d1"	15.0
"MX000017004"	1955	5	"tmax"	"d1"	31.0
"MX000017004"	1955	5	"tmin"	"d1"	20.0
"MX000017004"	1955	6	"tmax"	"d1"	30.0
…	…	…	…	…	…
"MX000017004"	2011	2	"tmin"	"d31"	NaN
"MX000017004"	2011	3	"tmax"	"d31"	36.5
"MX000017004"	2011	3	"tmin"	"d31"	17.0
"MX000017004"	2011	4	"tmax"	"d31"	NaN
"MX000017004"	2011	4	"tmin"	"d31"	NaN

id	year	month	variable	tmax	tmin
str	i32	i32	str	f64	f64
"MX000017004"	1955	4	"d1"	31.0	15.0
"MX000017004"	1955	5	"d1"	31.0	20.0
"MX000017004"	1955	6	"d1"	30.0	16.0
"MX000017004"	1955	7	"d1"	27.0	15.0
"MX000017004"	1955	8	"d1"	23.0	14.0
…	…	…	…	…	…
"MX000017004"	2010	12	"d31"	NaN	NaN
"MX000017004"	2011	1	"d31"	NaN	NaN
"MX000017004"	2011	2	"d31"	NaN	NaN
"MX000017004"	2011	3	"d31"	36.5	17.0
"MX000017004"	2011	4	"d31"	NaN	NaN

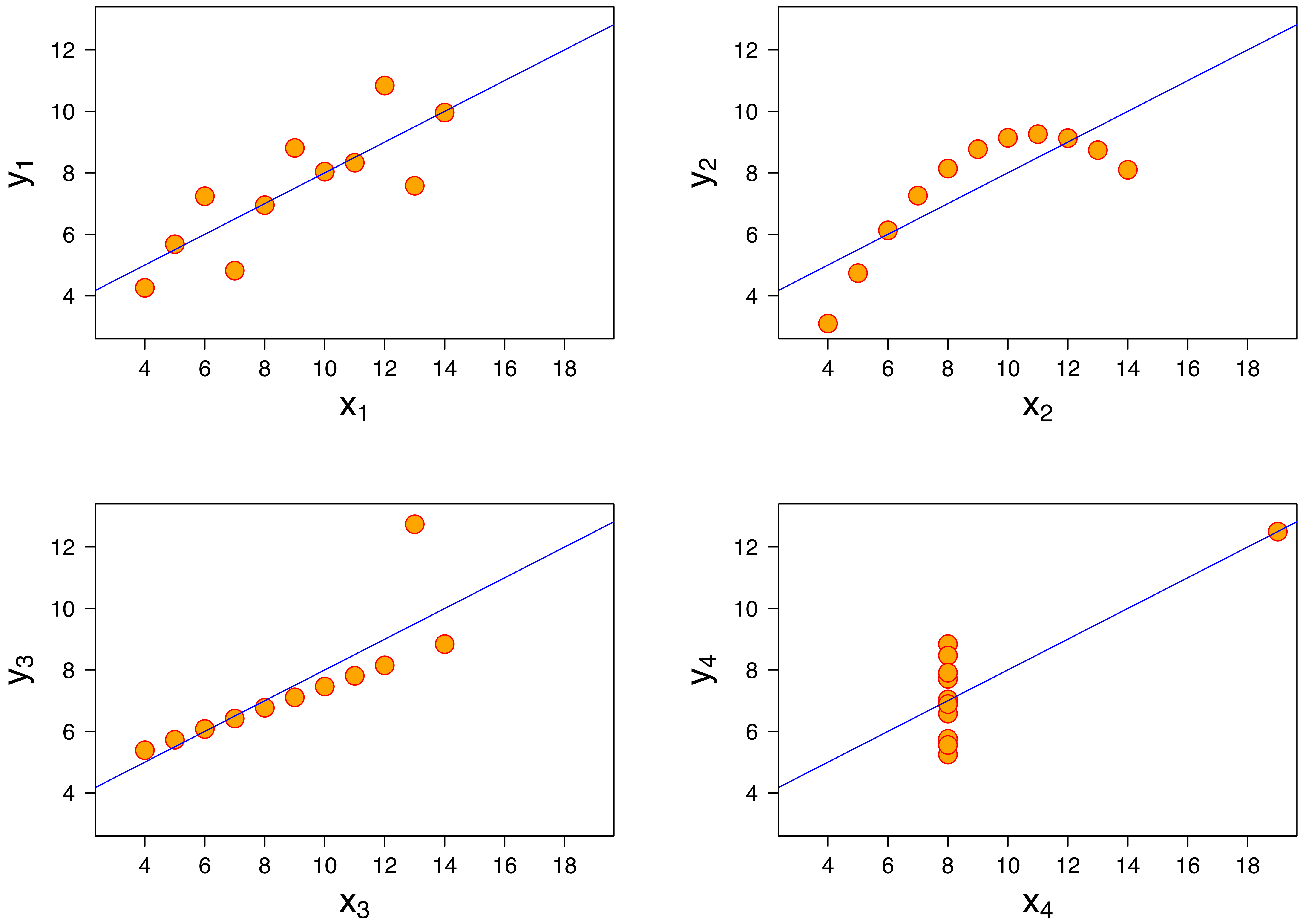
Pivot

- Inverse of unpivot: two columns (longer) become many columns (wider) one column becomes column names (variable), other becomes values

id	year	month	element	variable	value
str	i32	i32	str	str	f64
"MX000017004"	1955	4	"tmax"	"d1"	31.0
"MX000017004"	1955	4	"tmin"	"d1"	15.0
"MX000017004"	1955	5	"tmax"	"d1"	31.0
"MX000017004"	1955	5	"tmin"	"d1"	20.0
"MX000017004"	1955	6	"tmax"	"d1"	30.0
…	…	…	…	…	…
"MX000017004"	2011	2	"tmin"	"d31"	NaN
"MX000017004"	2011	3	"tmax"	"d31"	36.5
"MX000017004"	2011	3	"tmin"	"d31"	17.0
"MX000017004"	2011	4	"tmax"	"d31"	NaN
"MX000017004"	2011	4	"tmin"	"d31"	NaN

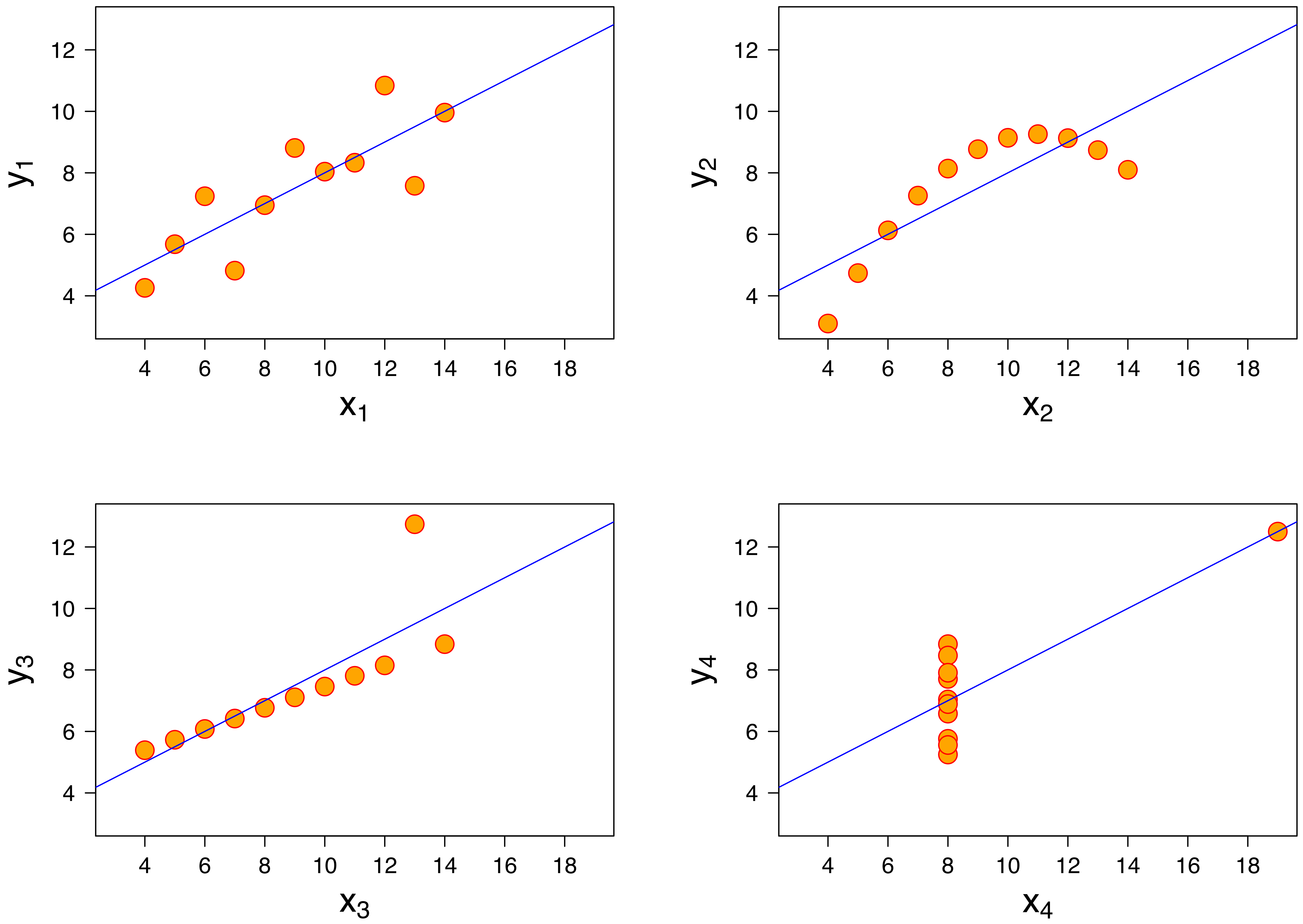
id	year	month	variable	tmax	tmin
str	i32	i32	str	f64	f64
"MX000017004"	1955	4	"d1"	31.0	15.0
"MX000017004"	1955	5	"d1"	31.0	20.0
"MX000017004"	1955	6	"d1"	30.0	16.0
"MX000017004"	1955	7	"d1"	27.0	15.0
"MX000017004"	1955	8	"d1"	23.0	14.0
…	…	…	…	…	…
"MX000017004"	2010	12	"d31"	NaN	NaN
"MX000017004"	2011	1	"d31"	NaN	NaN
"MX000017004"	2011	2	"d31"	NaN	NaN
"MX000017004"	2011	3	"d31"	36.5	17.0
"MX000017004"	2011	4	"d31"	NaN	NaN

Visualizing Data



[F. J. Anscombe]

Visualizing Data

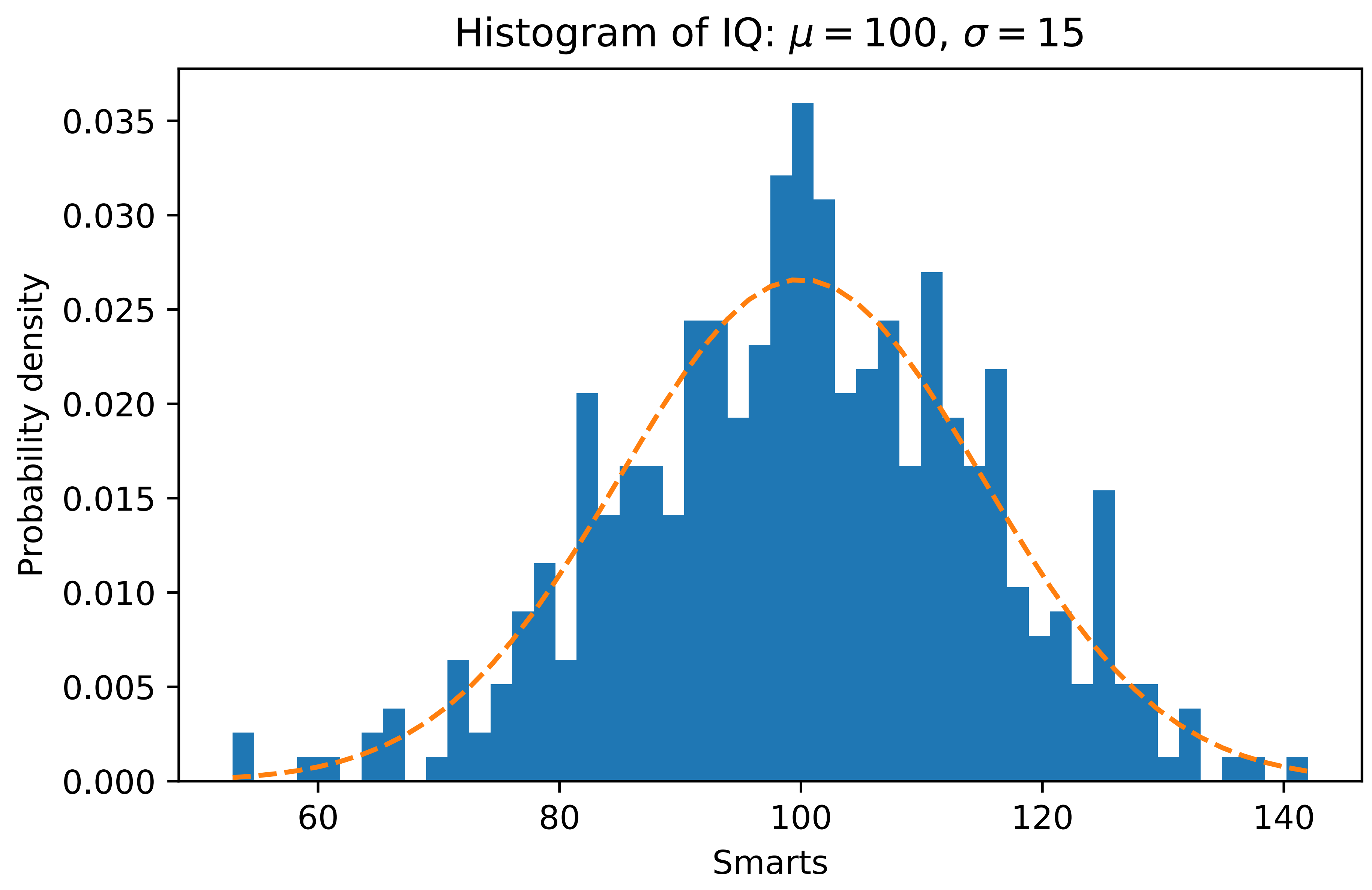


Mean of x	9
Variance of x	11
Mean of y	7.50
Variance of y	4.122
Correlation	0.816

[F. J. Anscombe]

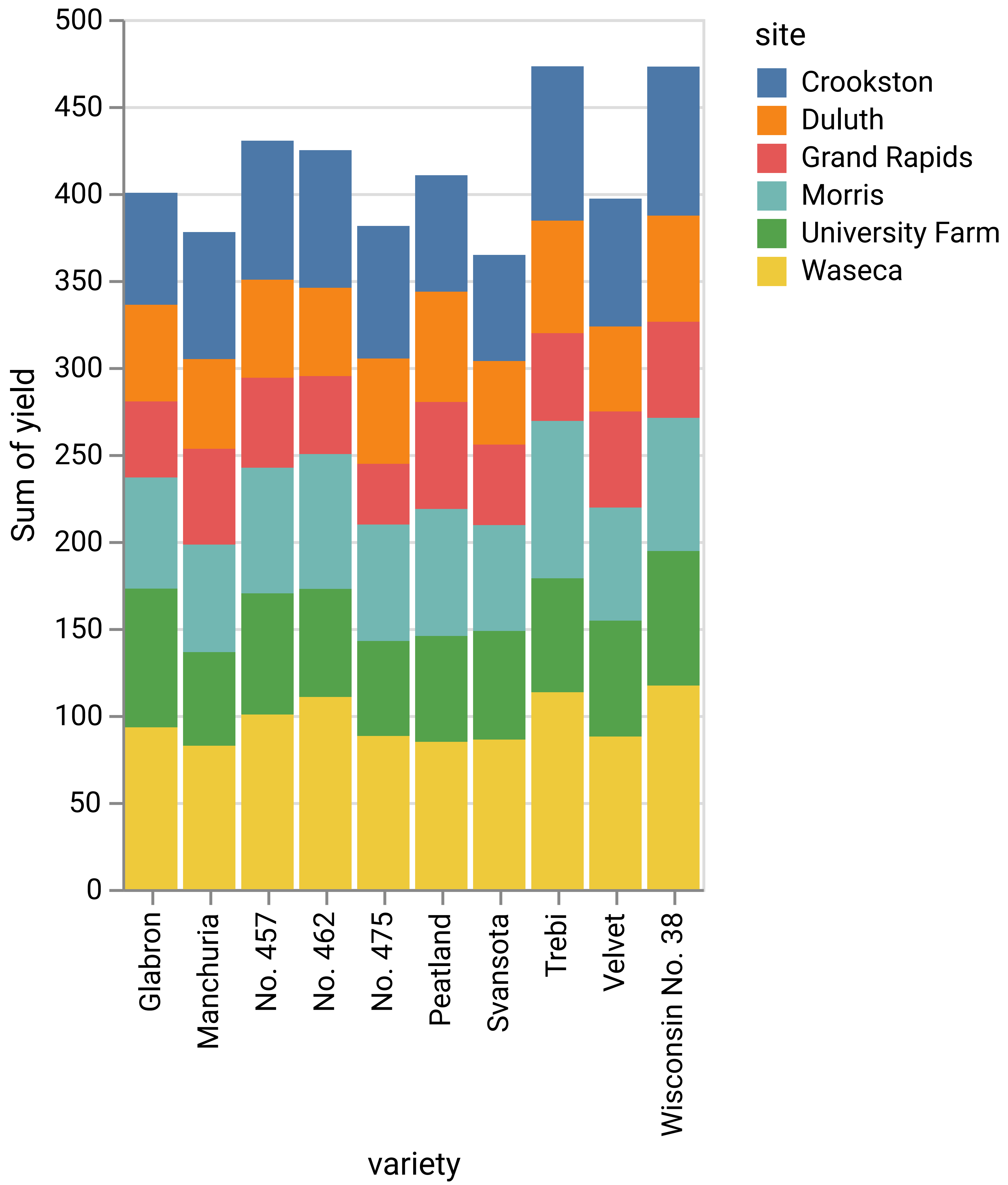
matplotlib

- 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

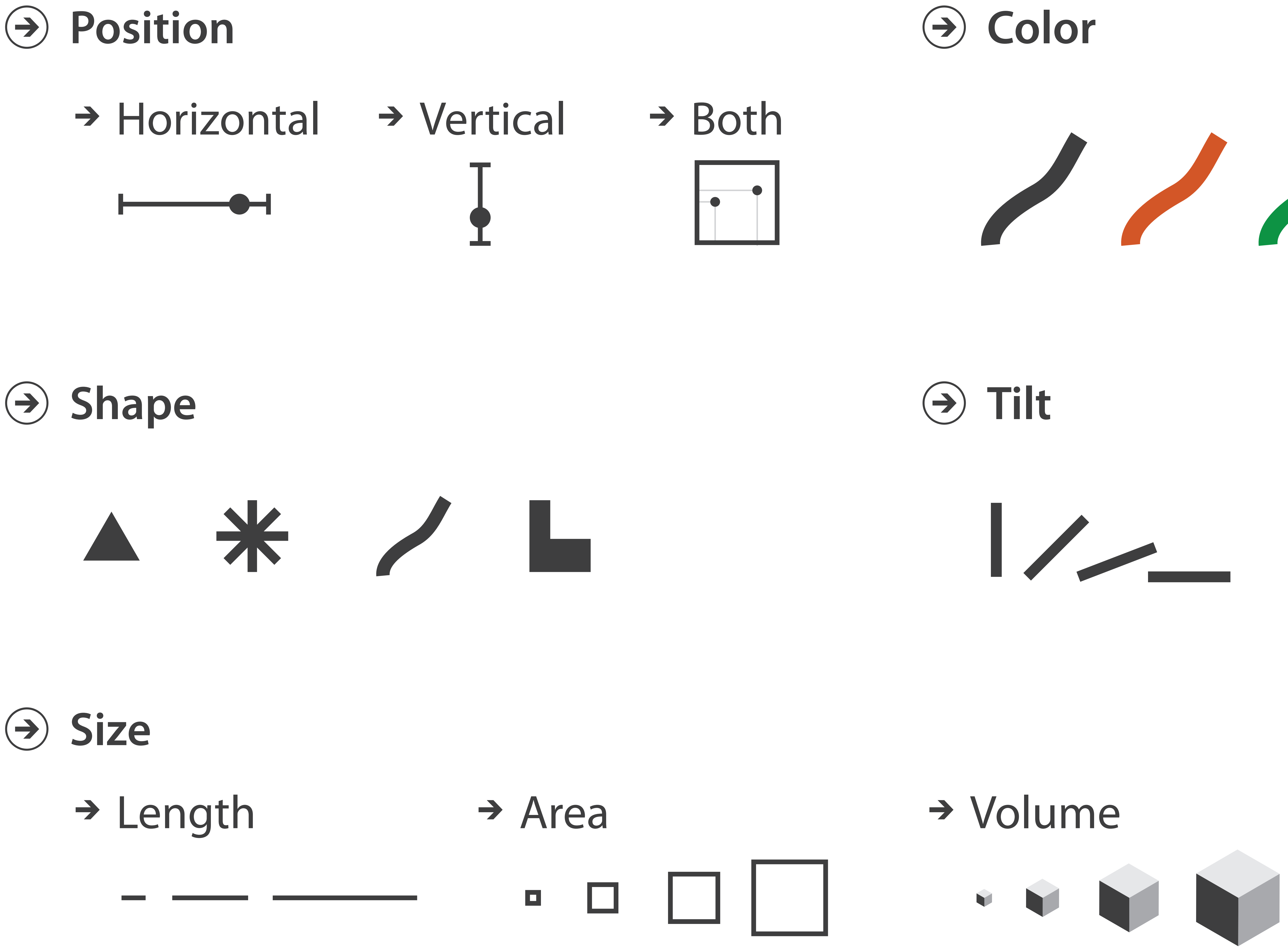


Visual Marks

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

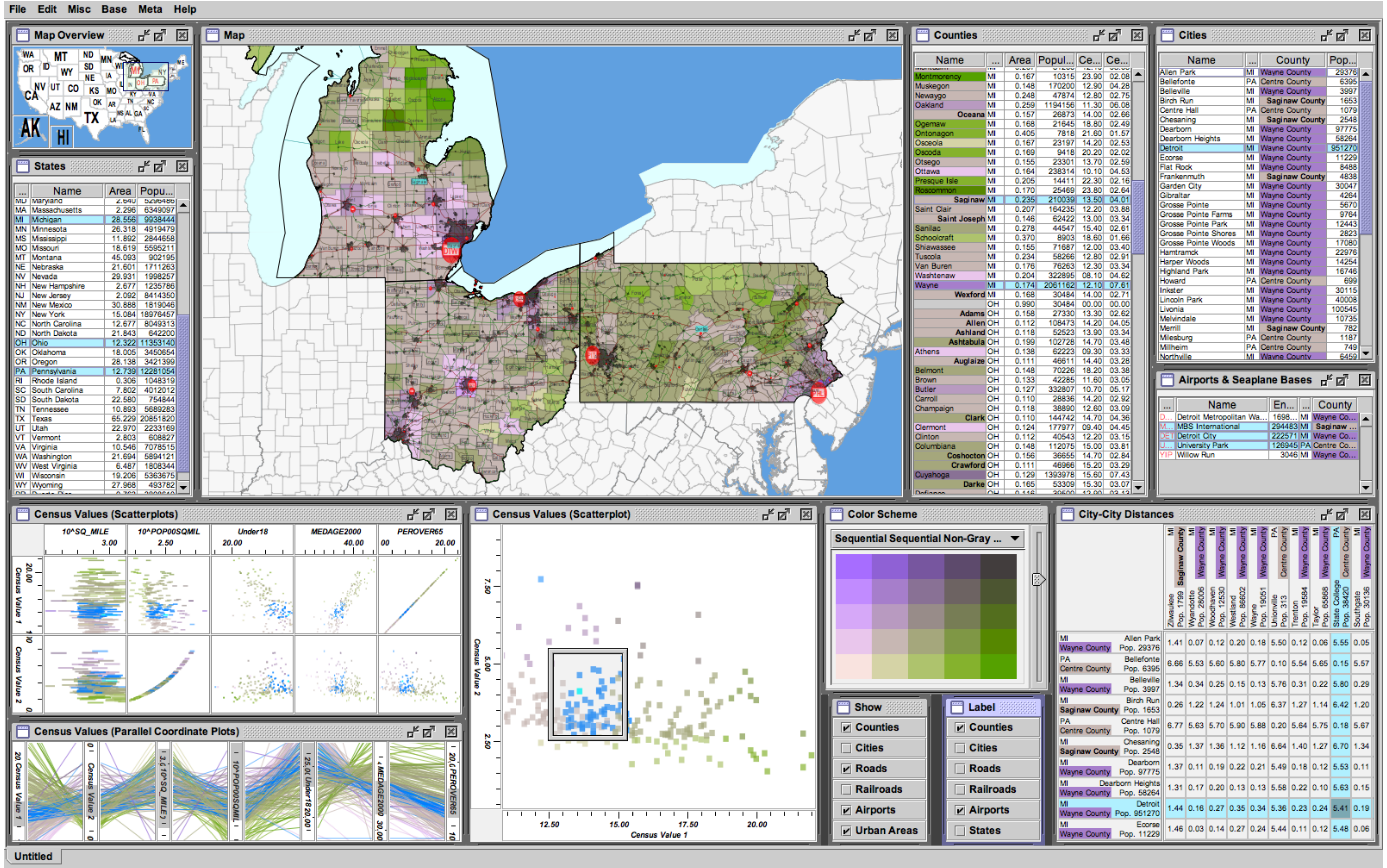
- Also can have surfaces, volumes
- Think of marks as a mathematical definition, or if familiar with tools like Adobe Illustrator or Inkscape, the path & point definitions
- Altair: area, bar, circle, geoshape, image, line, point, rect, rule, square, text, tick
  - Also compound marks: boxplot, errorband,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

Data is Encoded via Visual Channels



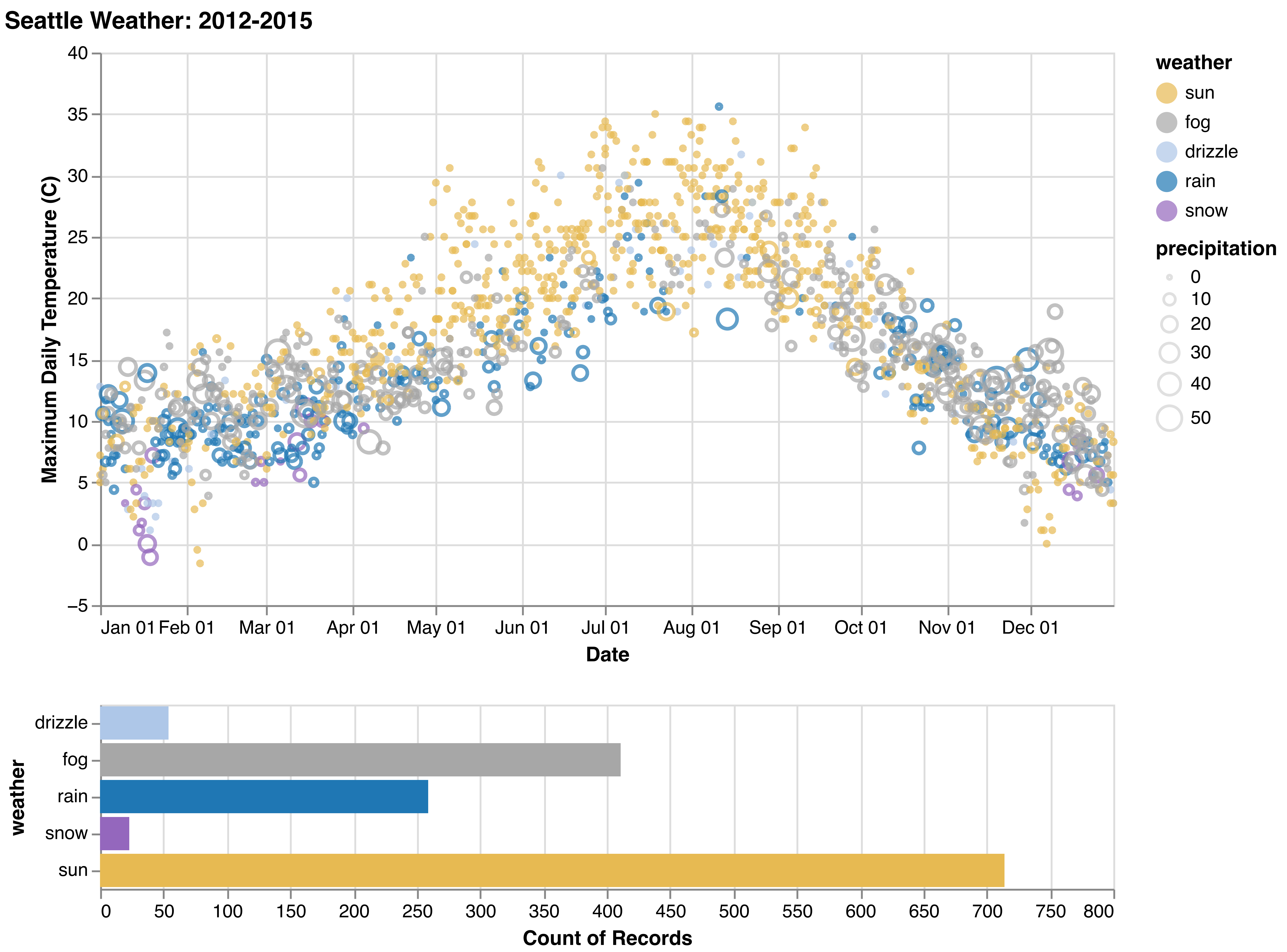
[Munzner (ill. Maguire), 2014]

Multiple Views



[Improvise, Weaver, 2004]

Interaction



Questions?

Final Exam

- Wednesday, Dec. 11, **8:00**-9:50am in PM 103
- **More comprehensive** than Test 2
- Expect questions from topics covered on Test 1 and 2
- Expect questions from the last few weeks of class (data, visualization, machine learning)
- Similar format