

# Programming Principles in Python (CSCI 503/490)

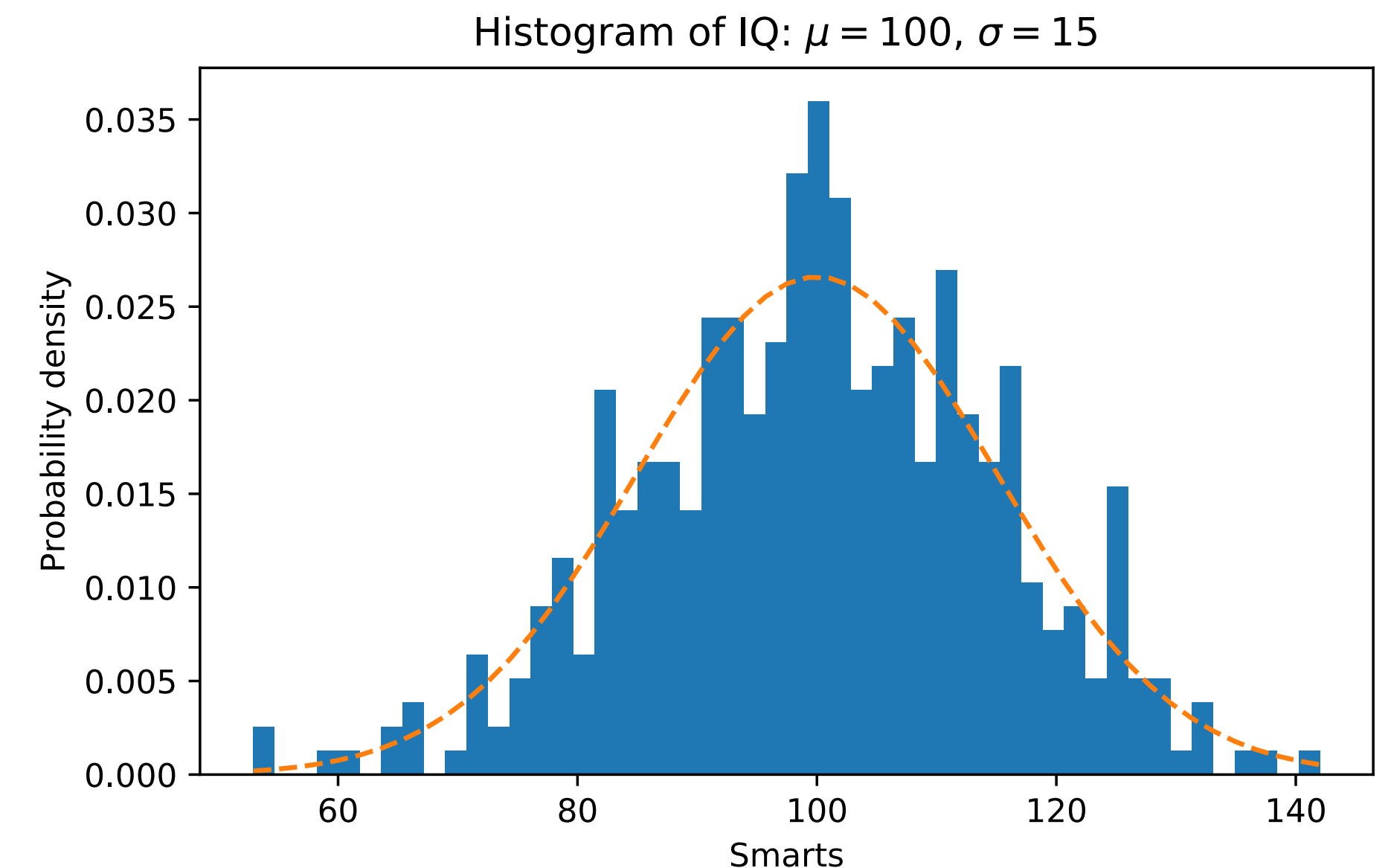
---

Machine Learning

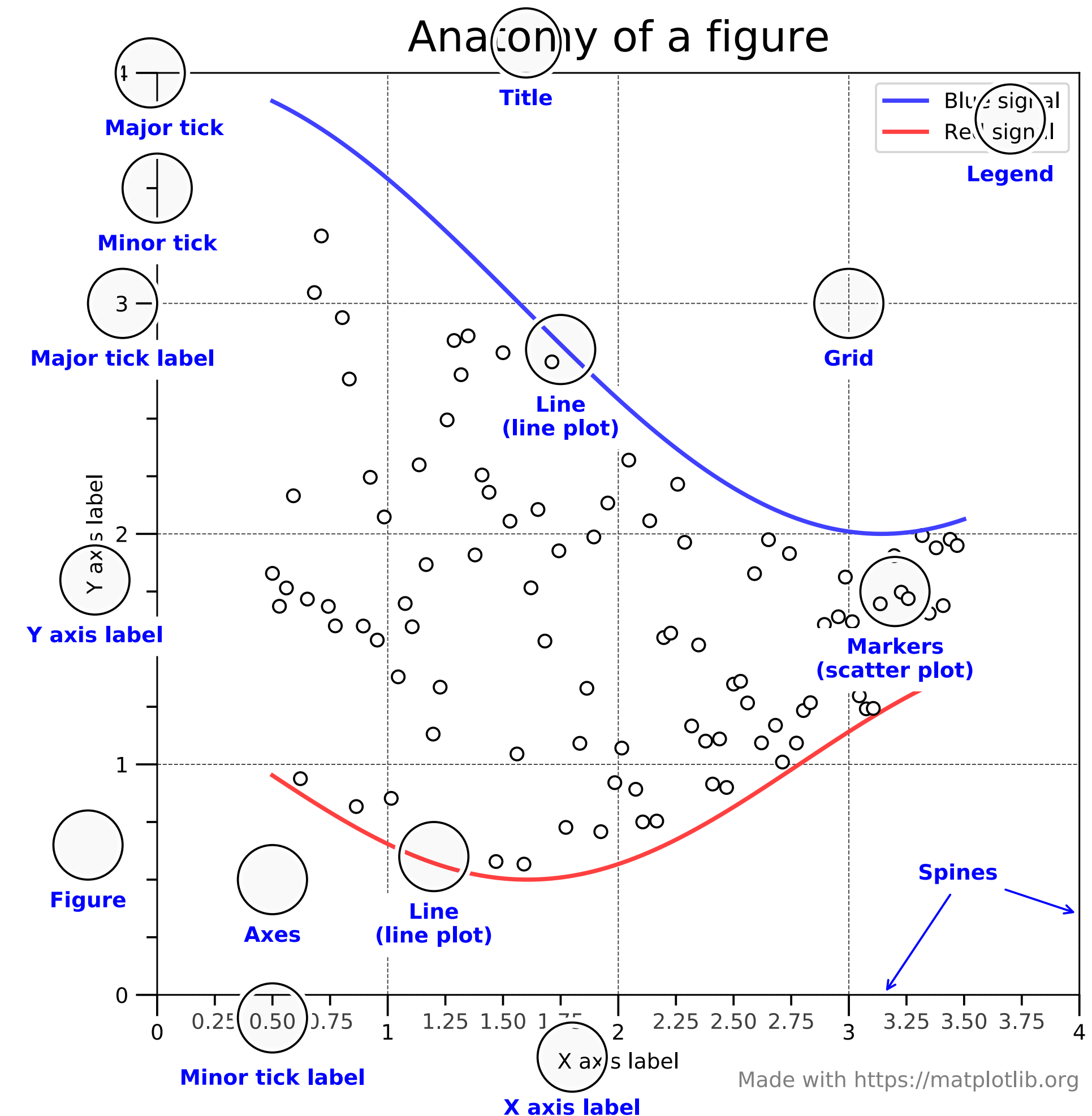
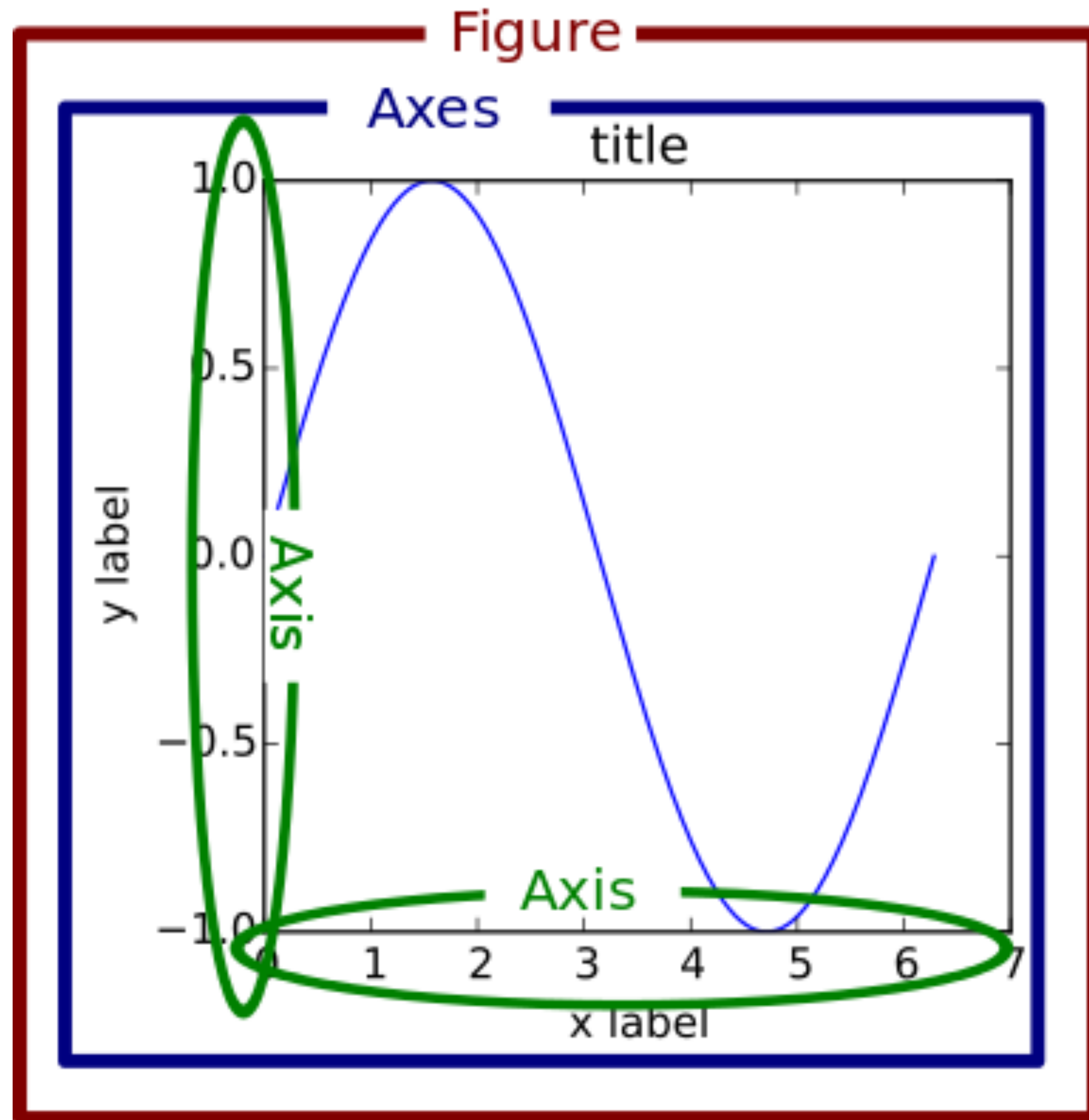
Dr. David Koop

# 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



# Anatomy of a Figure



[B. Solomon & matplotlib]

# Figure and Axes Objects

---

- pyplot is stateful, functions affect the "current" figure and axes
  - `plt.gcf()`: gets current figure
  - `plt.gca()`: gets current axes
  - Creates one if it doesn't exist!
- This is not aligned with object-based programming ideas
- Most methods in pyplot are translated to methods on the current axes (gca)
- We can instead call these directly, but first need to create them:
  - `fig, ax = plt.subplots()` # "constructor-like" method  
`ax.scatter([1, 3, 4, 6, 10], [1, 5, 2, 7, 3])`

# Multiple Figures

---

- subplots allows multiple axes in the same figure:
  - `fig, ax = plt.subplots(2, 2, figsize=(10, 10))` # rows, then columns
- `ax` is now a 2x2 numpy array
- Can put any type of visualization on each pair of axes
- ```
ax[0,0].plot([1,3,4,6,10],[1,5,2,7,3])
ax[0,1].bar(['Apple','Banana','Orange'],[0.99,0.50,1.25])
ax[1,0].pcolormesh(x, y, Z)
ax[1,1].pie([20,40,30,10],
            labels=['Apple','Banana','Orange','Pear'])
```

# 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.



# Altair Example

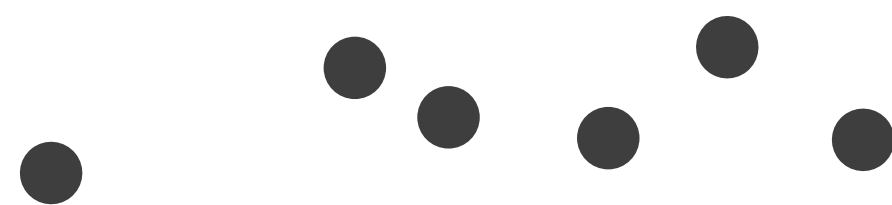
---

- ```
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:

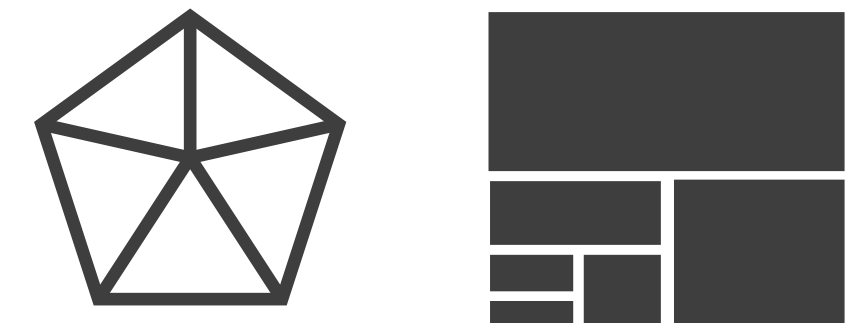
➔ **Points**



➔ **Lines**



➔ **Areas**



- 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

## ➔ Position

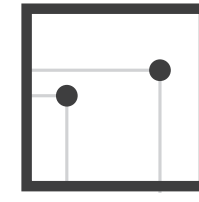
➔ Horizontal



➔ Vertical



➔ Both



## ➔ Color



## ➔ Shape



## ➔ Tilt

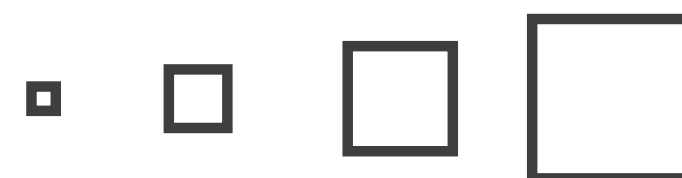


## ➔ Size

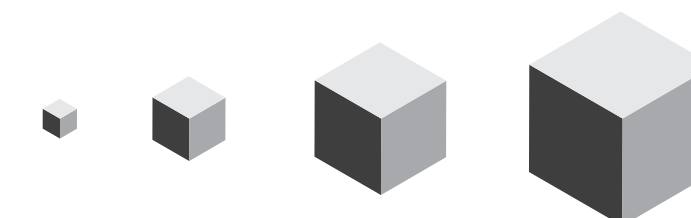
➔ Length



➔ Area



➔ Volume



[Munzner (ill. Maguire), 2014]

# Data Attributes and Altair Types

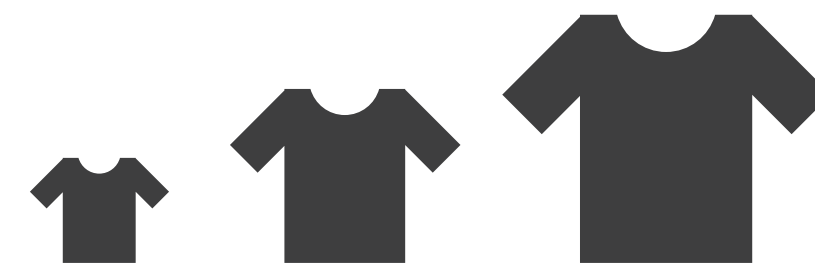
---

→ Categorical

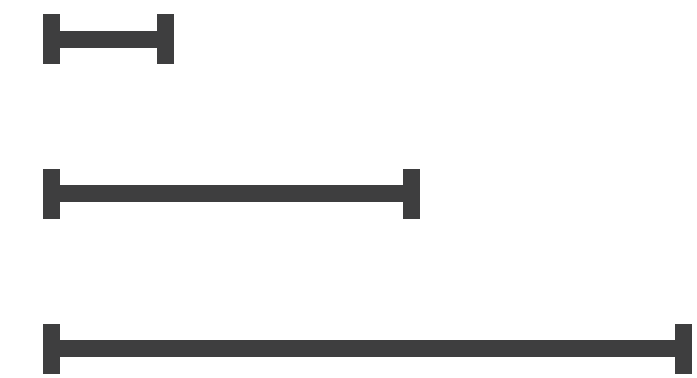


→ Ordered

→ *Ordinal*



→ *Quantitative*



# Data Attributes and Altair Types

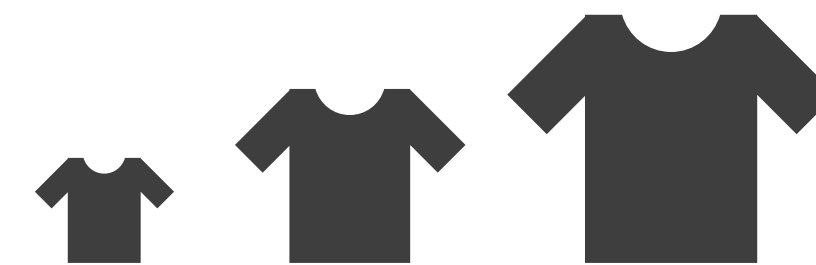
---

→ Categorical

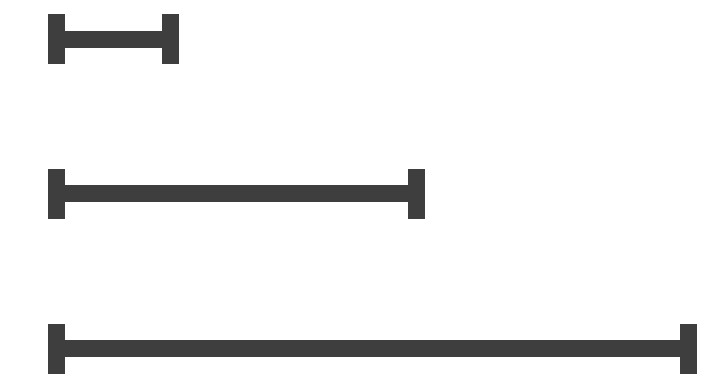


→ Ordered

→ *Ordinal*



→ *Quantitative*













- 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

➔ **Magnitude** Channels: **Ordered** Attributes

Position on common scale	
Position on unaligned scale	
Length (1D size)	
Tilt/angle	
Area (2D size)	
Depth (3D position)	
Color luminance	
Color saturation	
Curvature	
Volume (3D size)	

➔ **Identity** Channels: **Categorical** Attributes

Spatial region	
Color hue	
Motion	
Shape	

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

[Munzner (ill. Maguire), 2014]

# Assignment 8

---

- Due Friday, May 2
- Last Assignment
- Data and Visualization
- Use polars or pandas
- Must use matplotlib or altair where directed

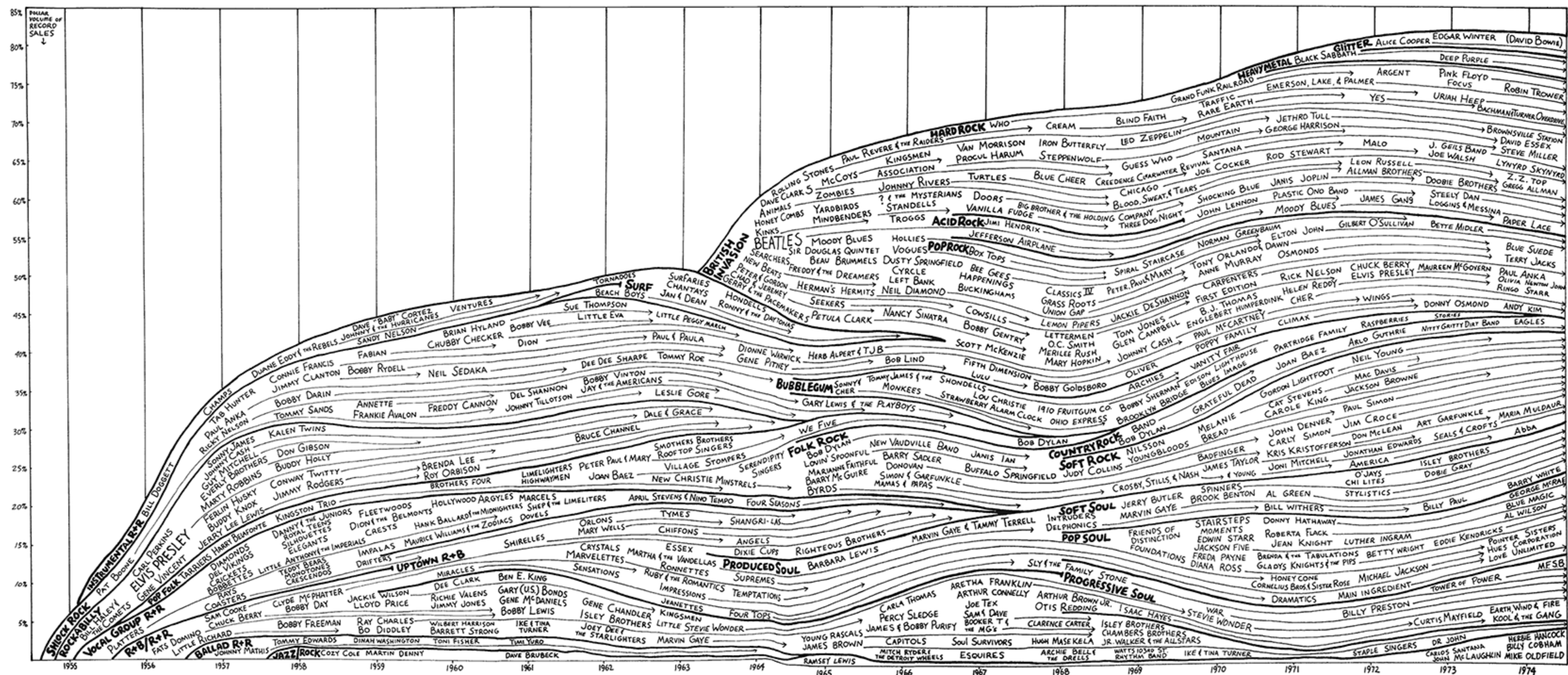
# Final Exam

---

- Monday, May 5, **12:00**-1:50pm 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



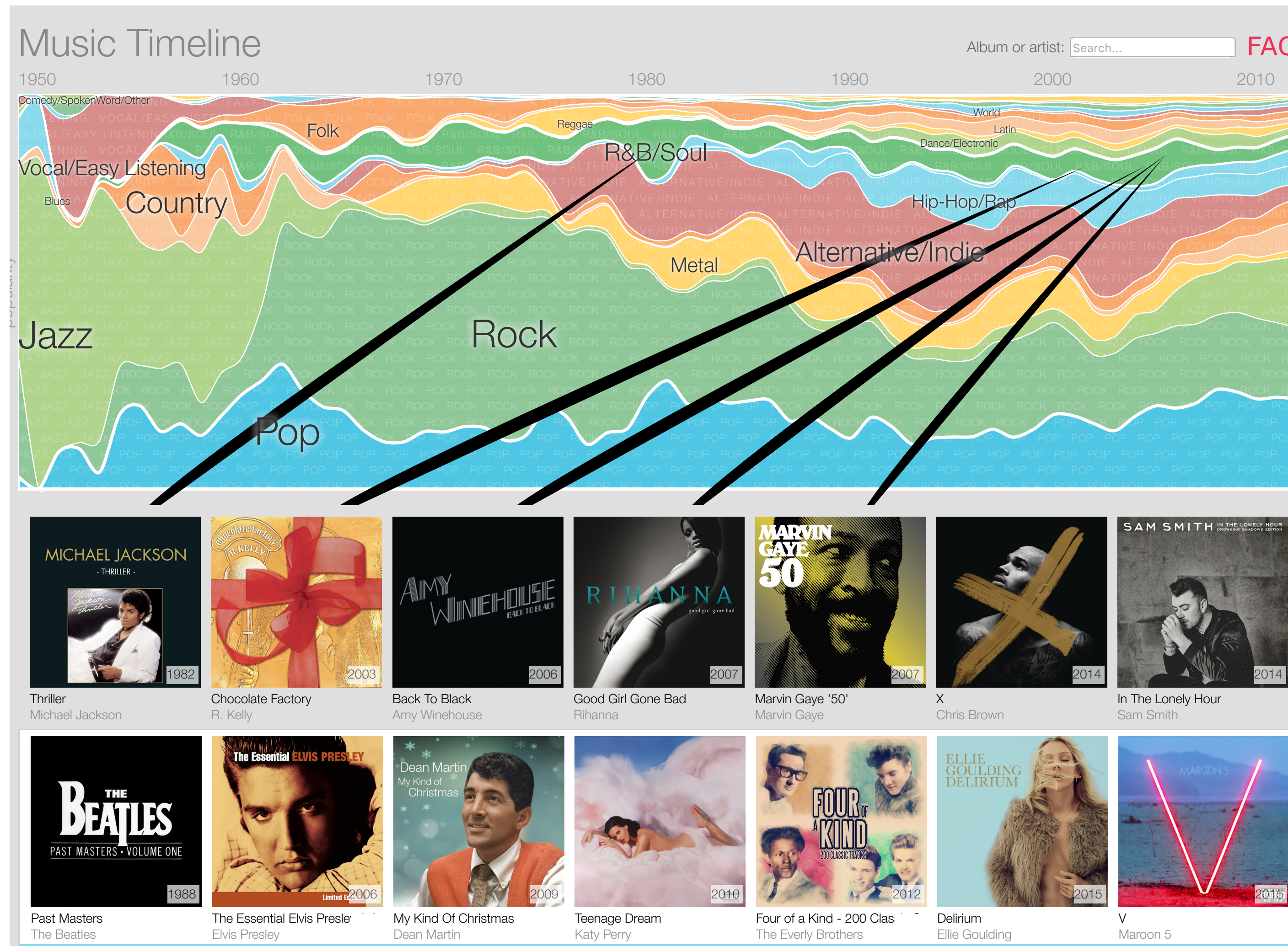
# Visualization



[Rock 'N' Roll is Here to Pay, R. Garofalo, 1977 (via Tufte)]



# Also Visualization, but with Interaction



[Music Timeline, Google Research (no working version)]

# Interaction

---

- Grammar of Graphics, why not Grammar of Interaction?
- Vega-Lite/Altair is about **interactive** graphics
- Types of Interactions:
  - Selection
  - Zoom
  - Brushing



# Selection

---

- Selection is often used to initiate other changes
- User needs to select something to drive the next change
- What can be a selection target?
  - Items, links, attributes, (views)
- How?
  - mouse click, mouse hover, touch
  - keyboard modifiers, right/left mouse click, force
- Selection modes:
  - Single, multiple
  - Contiguous?

# Highlighting

---

- Selection is the user action
- Feedback is important!
- How? Change selected item's visual encoding
  - Change color: want to achieve visual popout
  - Add outline mark: allows original color to be preserved
  - Change size (line width)
  - Add motion: marching ants



# Highlighting

---

- Selection is the user action
- Feedback is important!
- How? Change selected item's visual encoding
  - Change color: want to achieve visual popout
  - Add outline mark: allows original color to be preserved
  - Change size (line width)
  - Add motion: marching ants



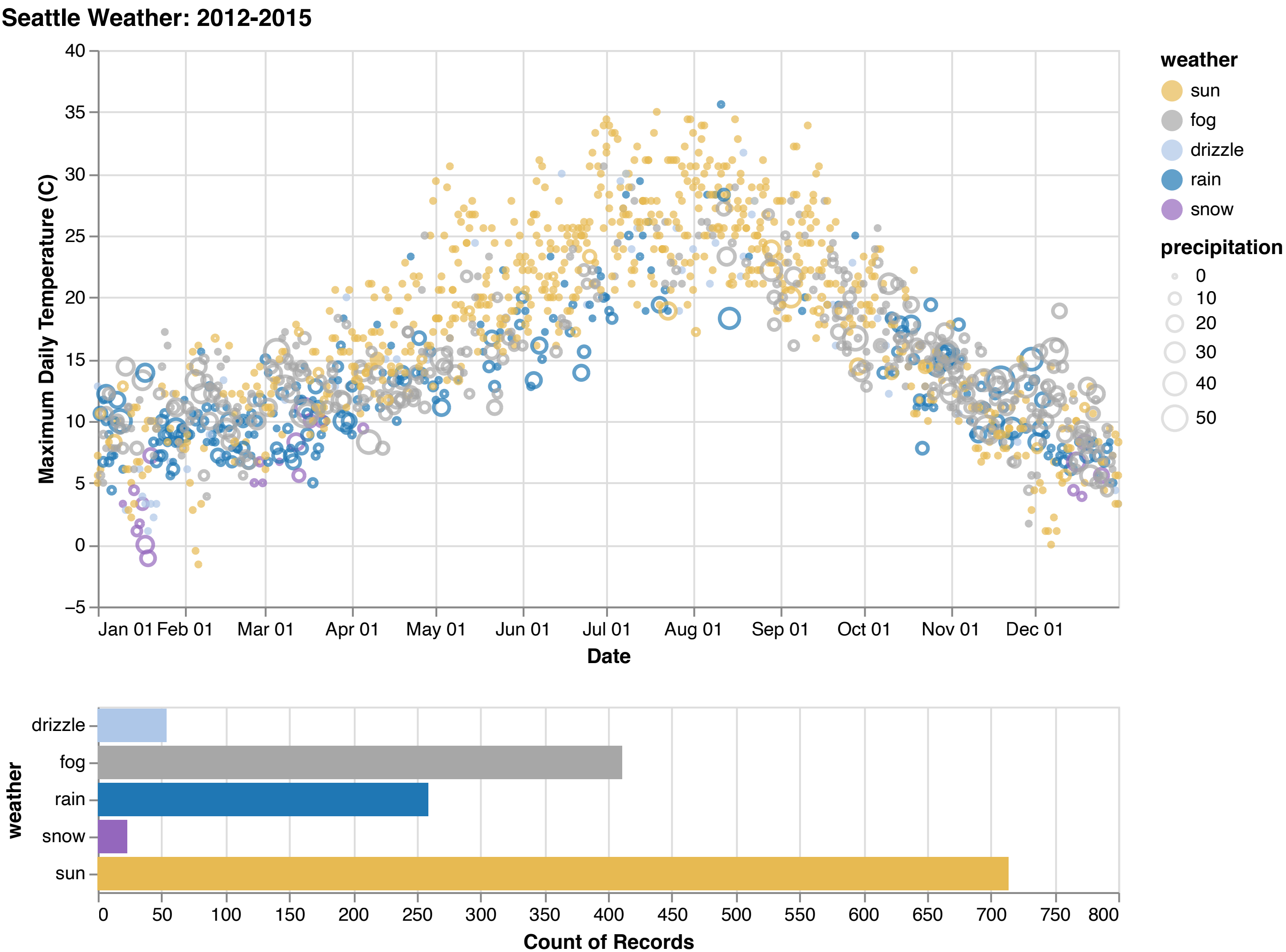


# Altair's Interactive Charts

---

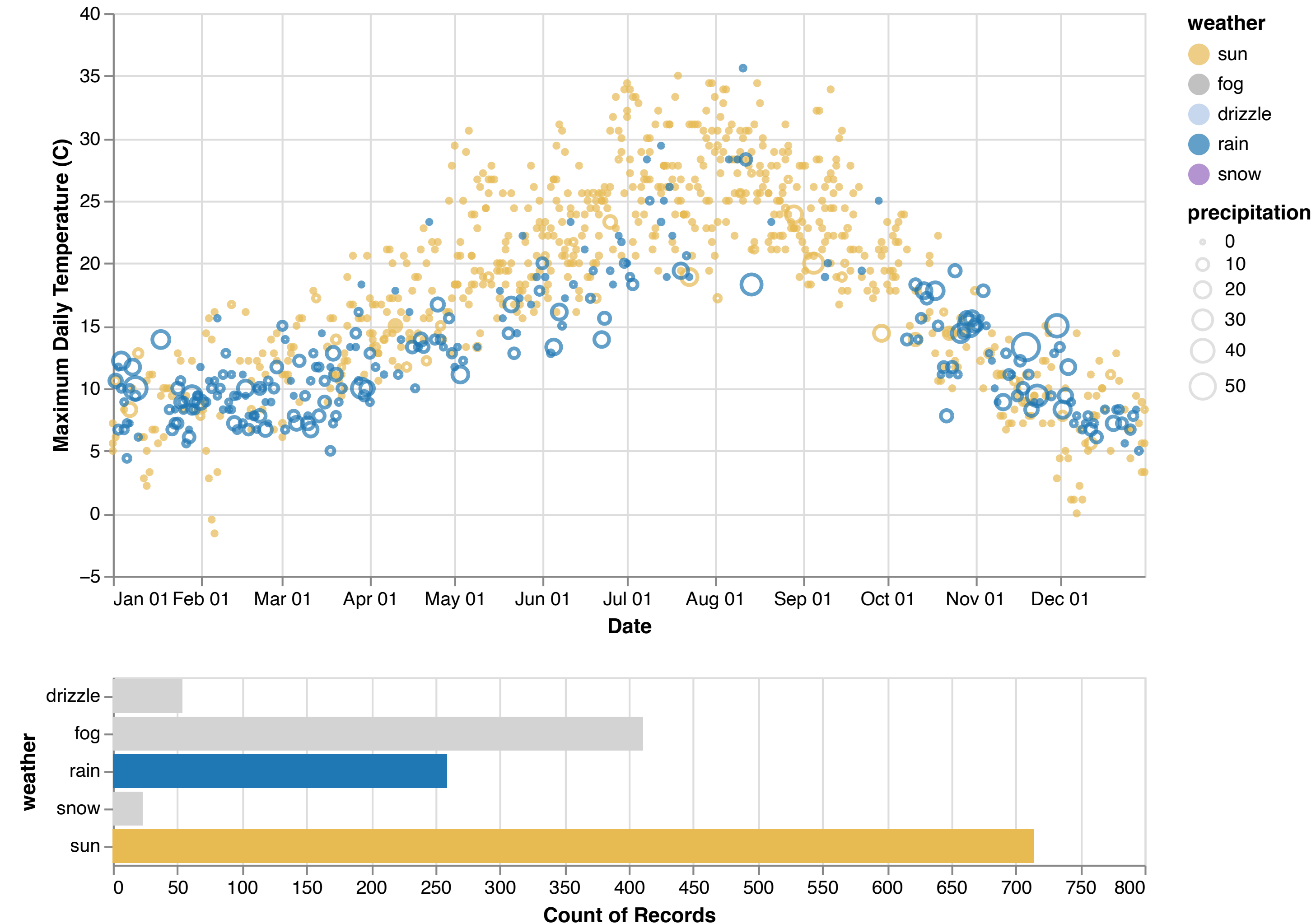
- <https://altair-viz.github.io/gallery/index.html#interactive-charts>

# Interaction



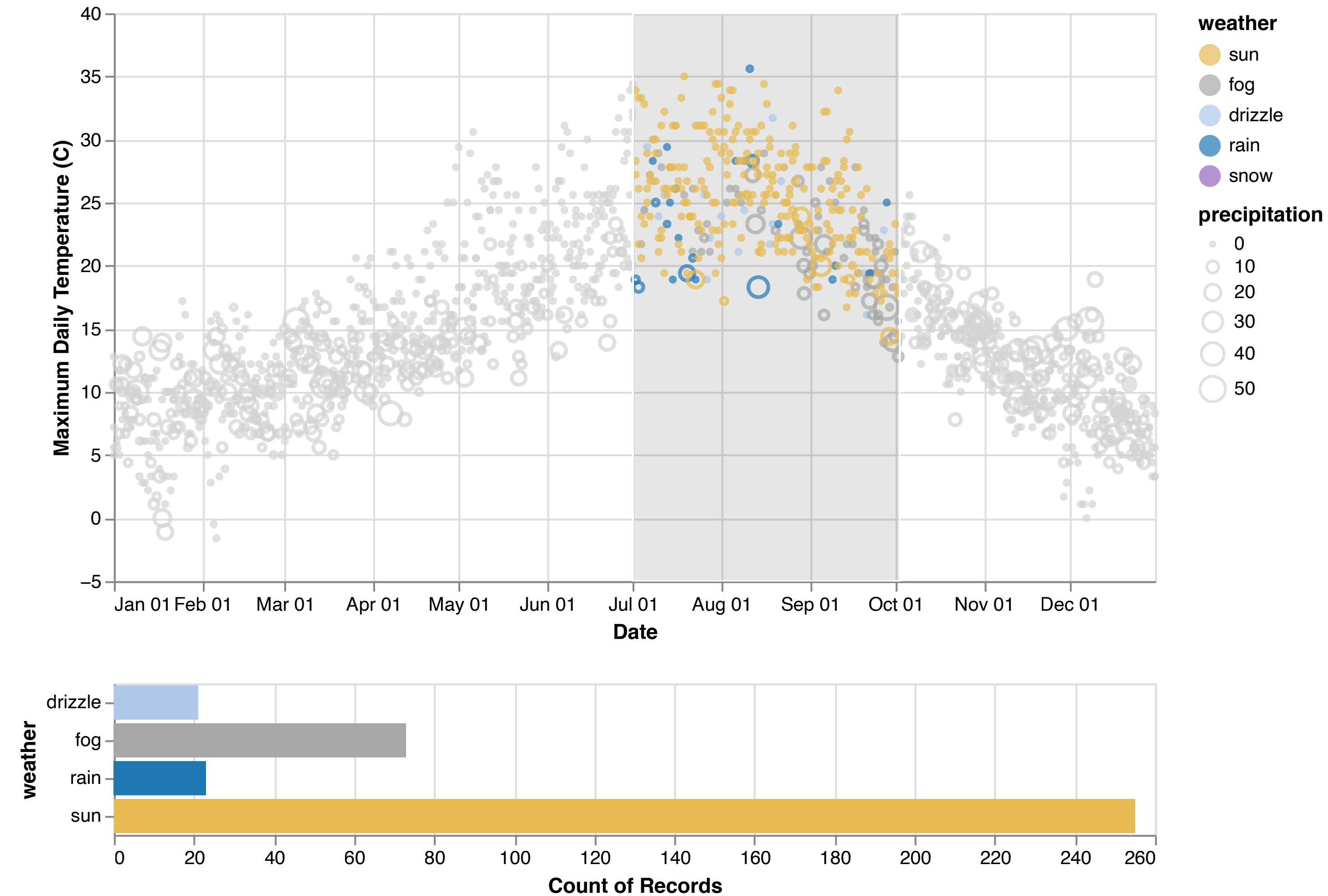
# Weather Selection: Rain vs. Sun

Seattle Weather: 2012-2015



# Date Selection: July-September Sun

Seattle Weather: 2012-2015



# Machine Learning in Python

# 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)]



# When to Use Machine Learning?

---

- ML is used when:
  - Human expertise does not exist (navigating on Mars)
  - Humans can't explain their expertise (speech recognition)
  - Models must be customized (personalized medicine)
  - Models are based on huge amounts of data (genomics)
- ML isn't always useful:
  - Calculating payroll...

[E. Alpaydin via [E. Eaton](#)]

# Questions when building a machine learning solution

---

- What question(s) am I trying to answer? Do I think the data collected can answer that question?
- What is the best way to phrase my question(s) as a machine learning problem?
- Have I collected enough data to represent the problem I want to solve?
- What features of the data did I extract, and will these enable the right predictions?
- How will I measure success in my application?

[A. Müller & S. Guido]

# Machine Learning Workflow Overview

---

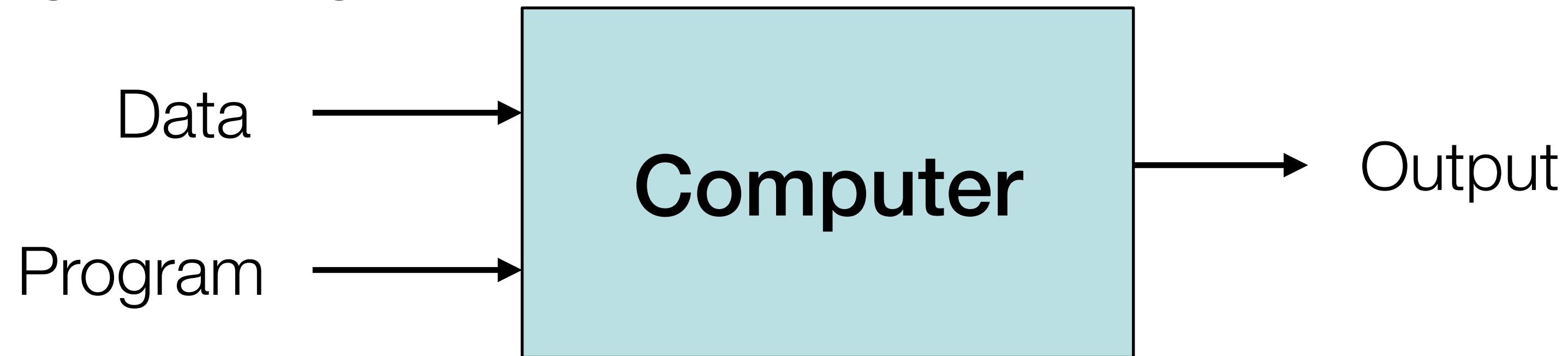
1. Should I use ML on this problem?
  - Is there a pattern to detect? Can I solve it analytically? Do I have data?
2. Gather and organize data.
  - Preprocessing, cleaning, visualizing.
3. Establishing a baseline.
4. Choosing a model, loss, regularization, ...
5. Optimization (could be simple, could be a Phd...).
6. Hyperparameter search.
7. Analyze performance & mistakes, and iterate back to step 4 (or 2).

[R. Grosse et al.]

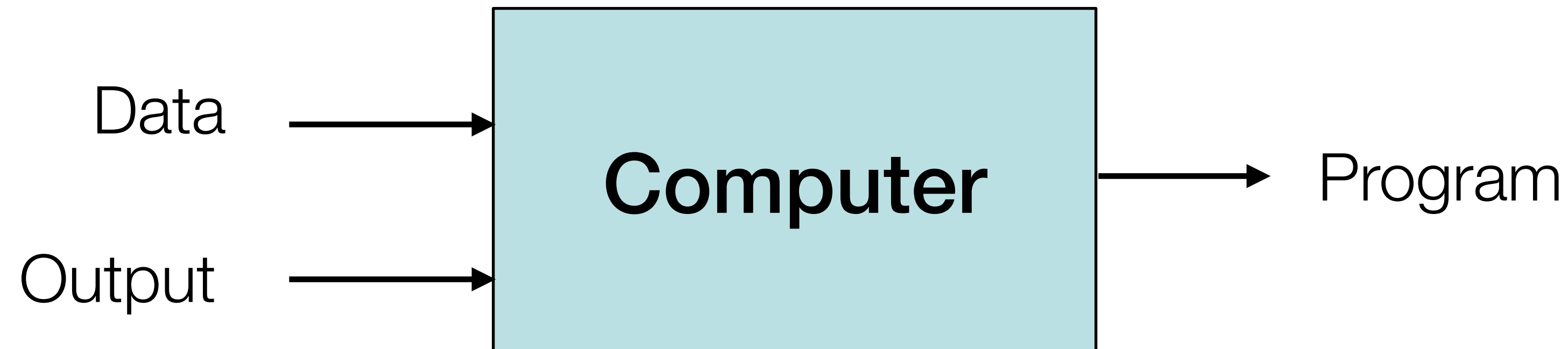
# Machine Learning

---

- 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

# 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**)

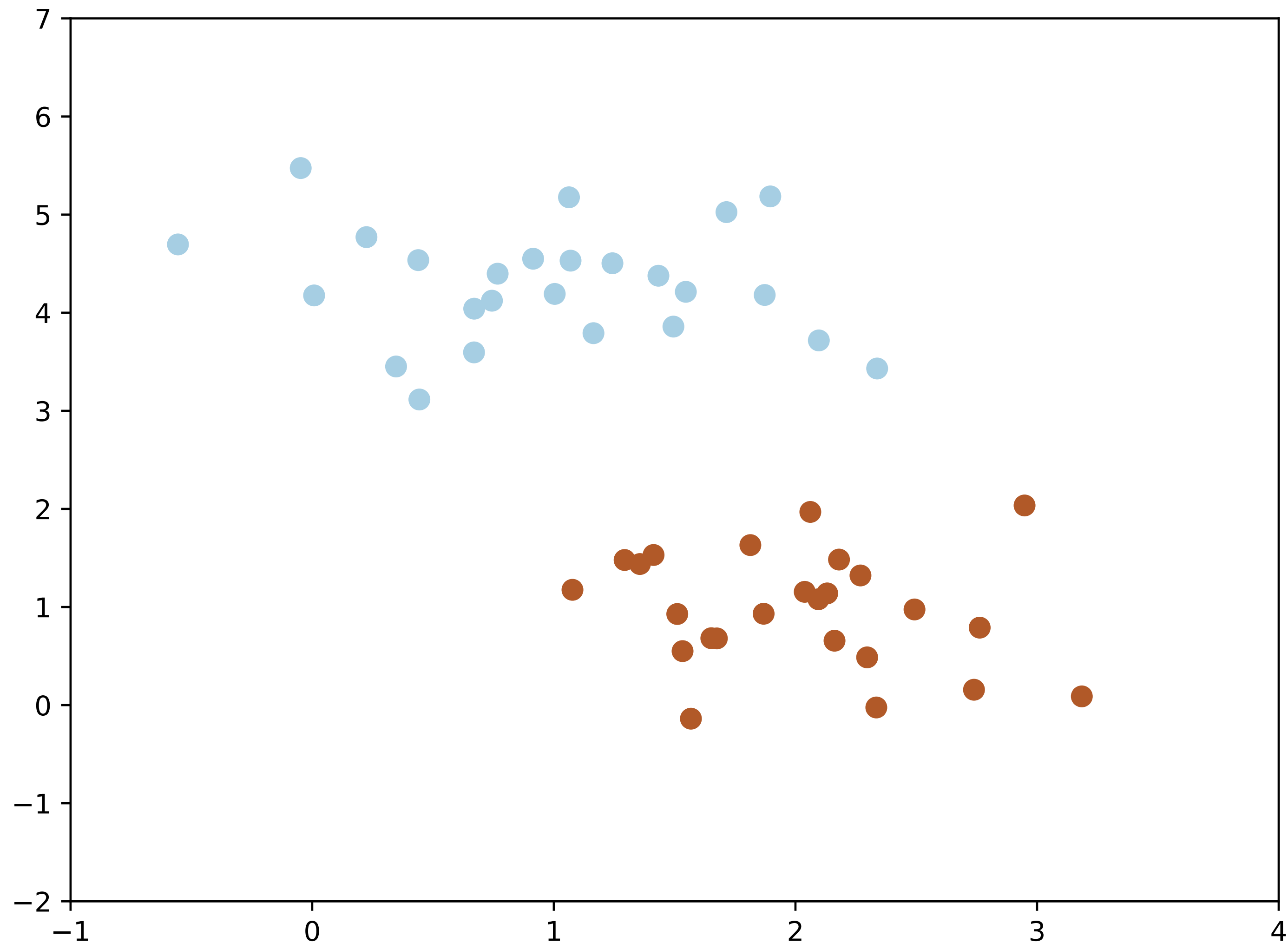


- Detecting fraudulent activity in credit card transactions (**supervised**)
- Identifying topics in a set of blog posts (**unsupervised**)
- Grouping customers with similar preferences (**unsupervised**)

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

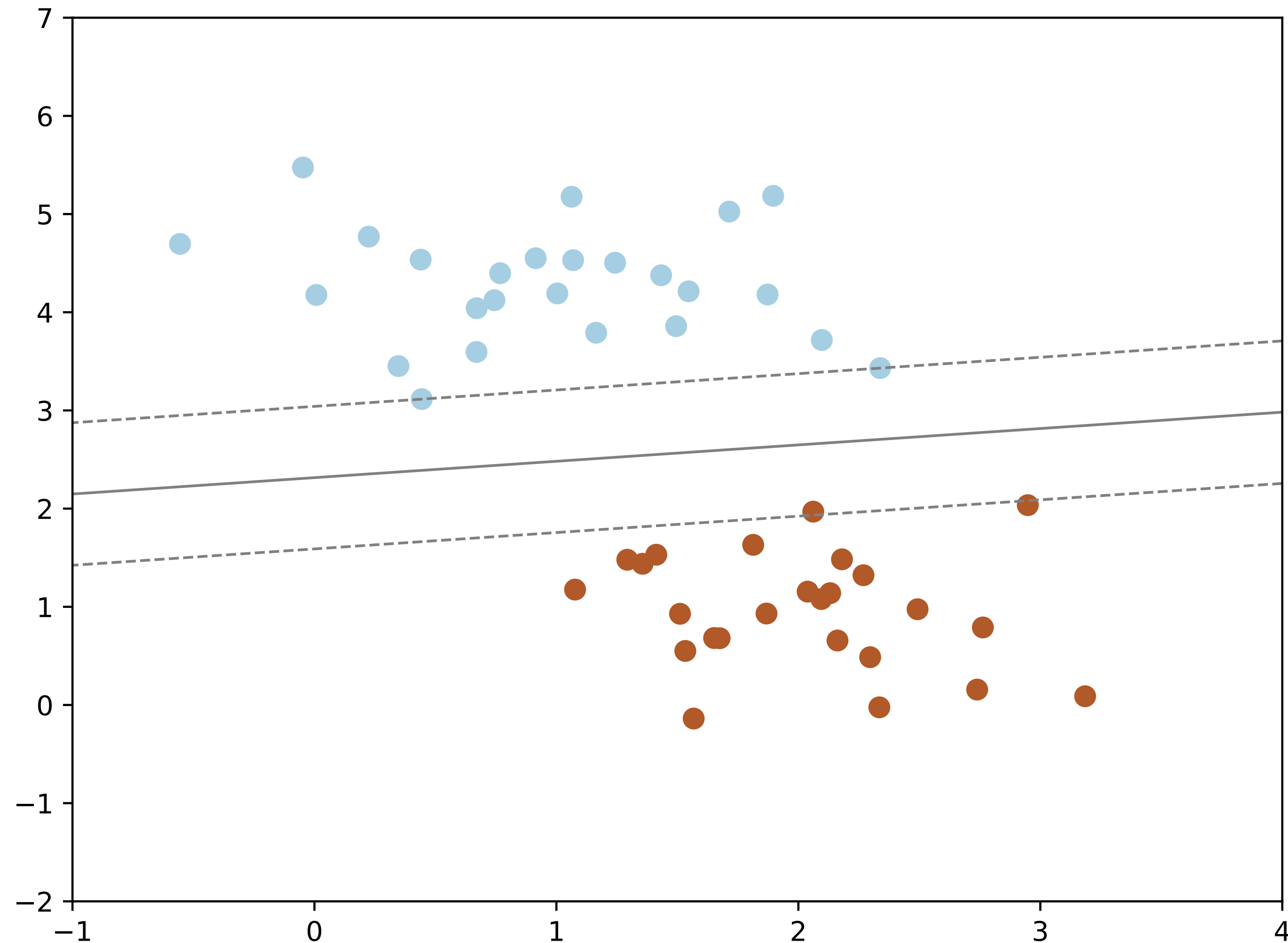
# Supervised Learning

---



[J. VanderPlas]

# Supervised Learning: Learned Algorithm (Fit)

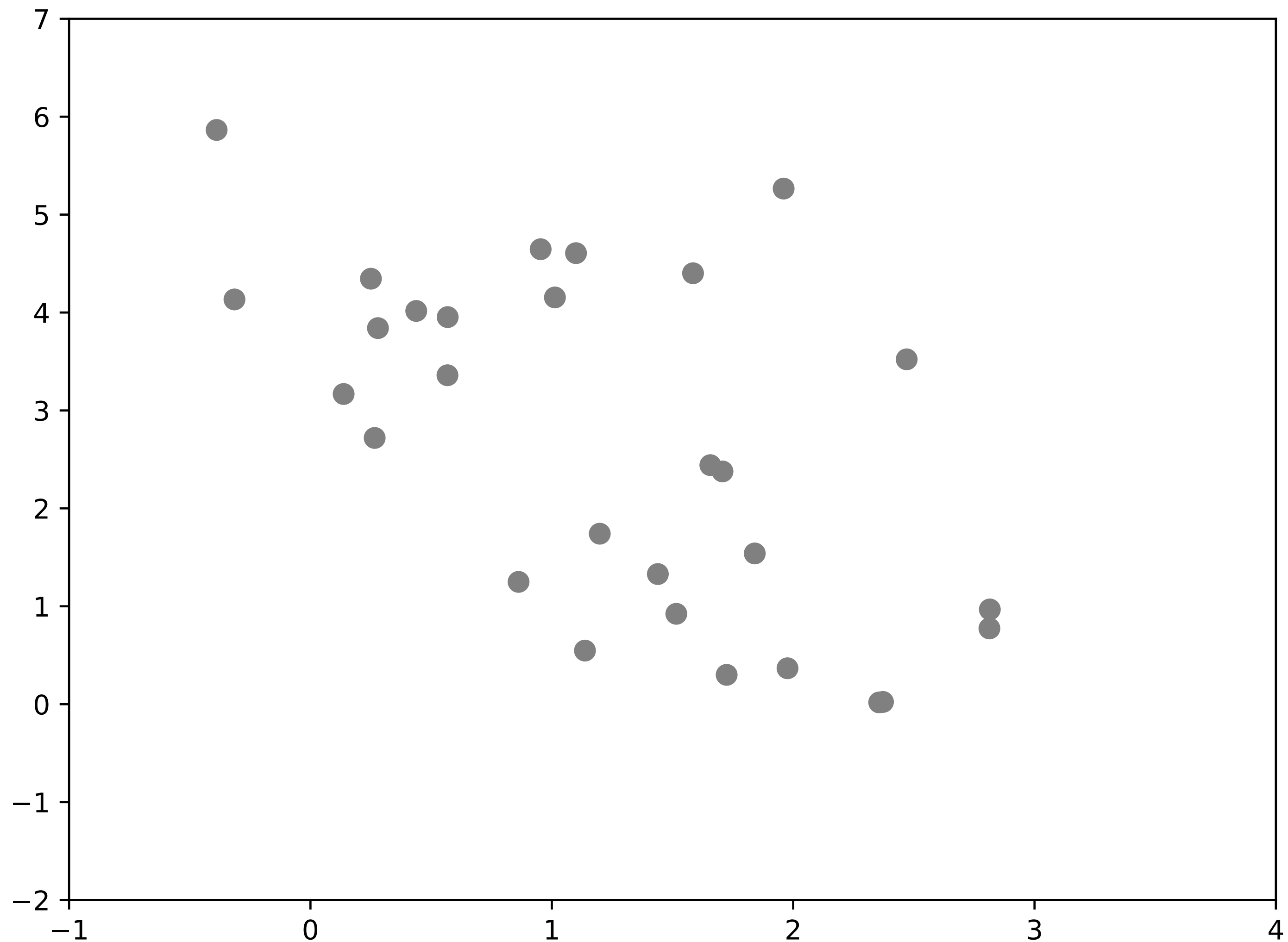


[J. VanderPlas]



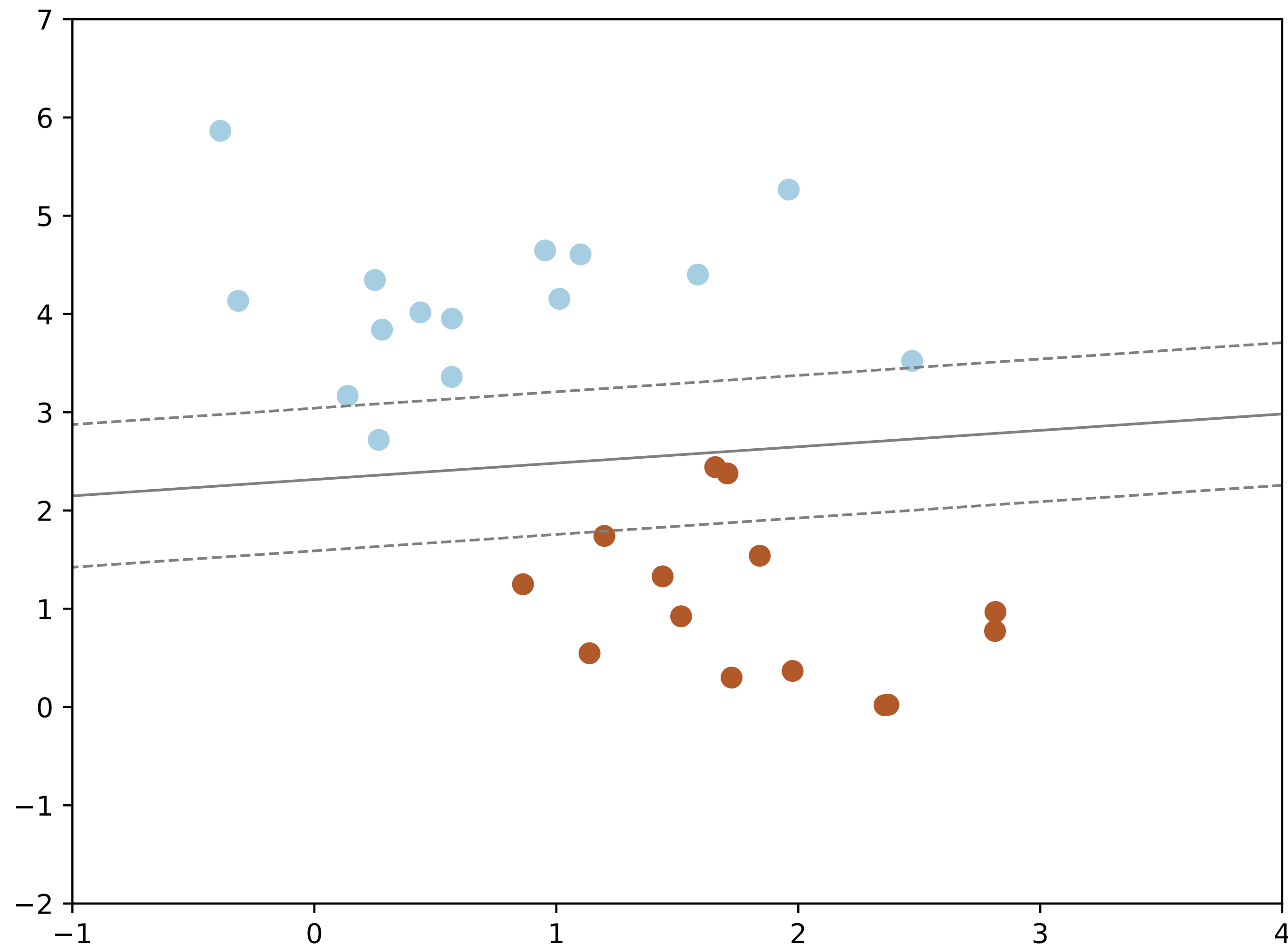
# Supervised Learning: Prediction

---



[J. VanderPlas]

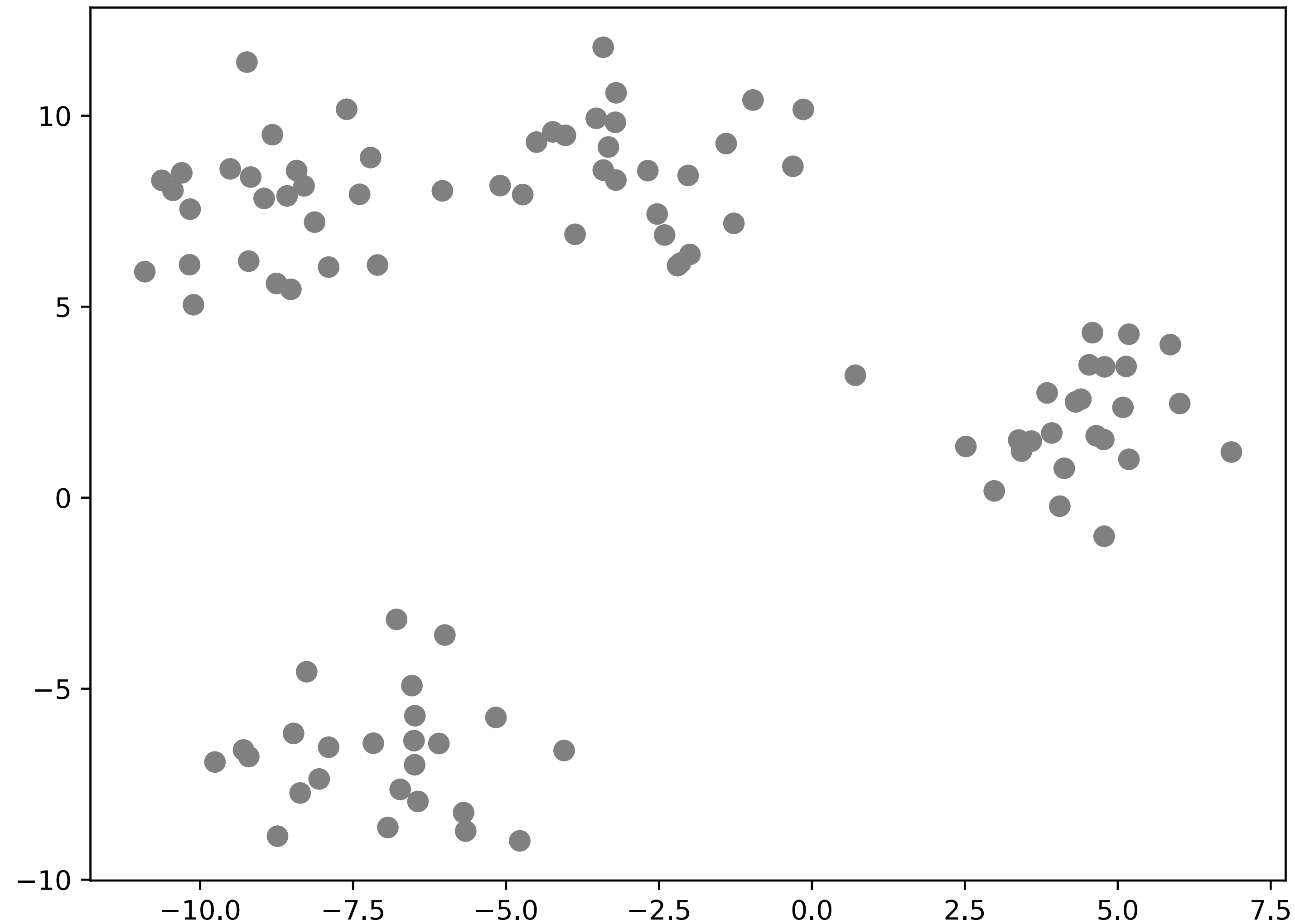
# Supervised Learning: Prediction



[J. VanderPlas]

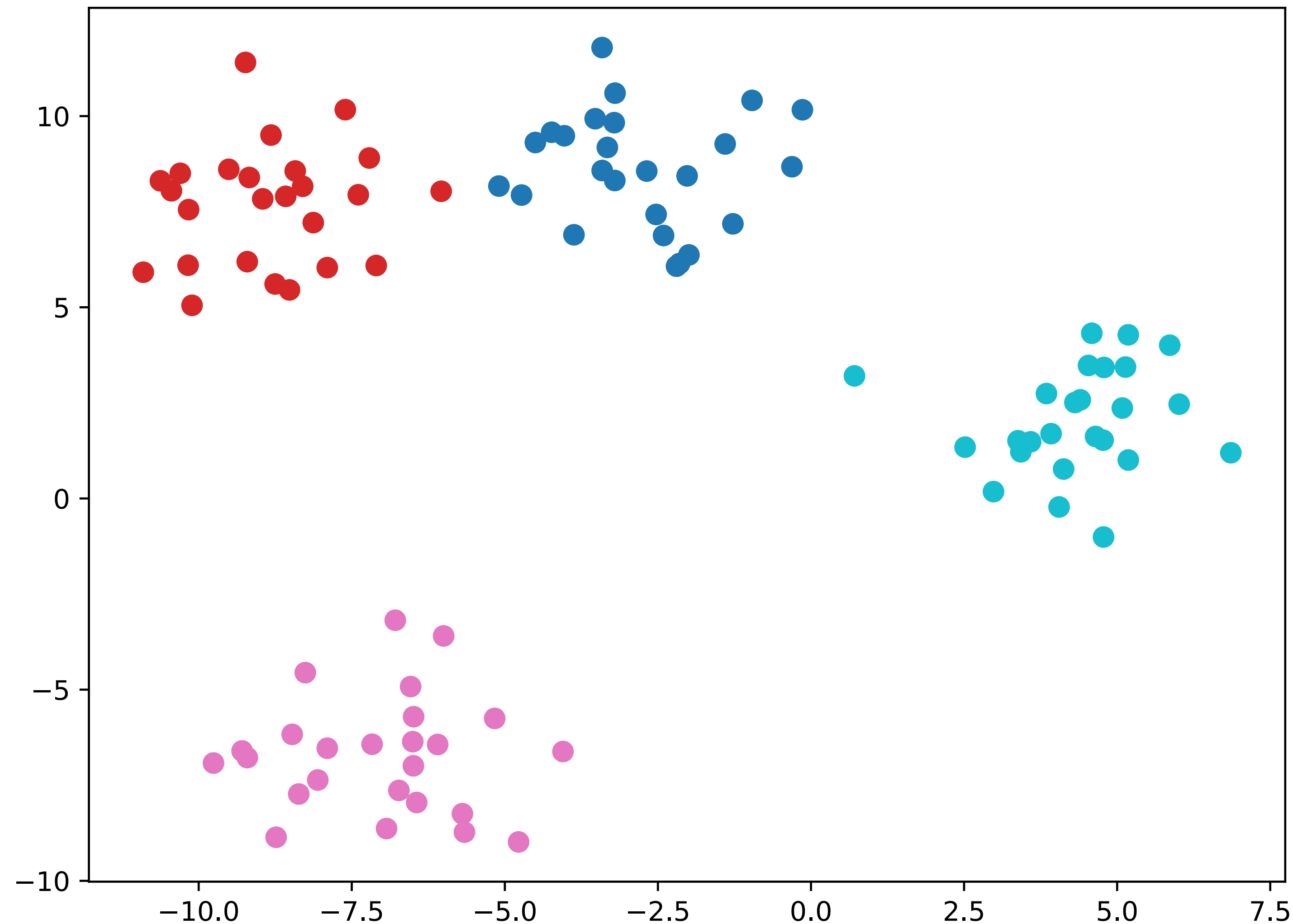
# Unsupervised Learning: Input

---



[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](https://scikit-learn.org)]

# 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 via `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