Programming Principles in Python (CSCI 503/490)

Data Visualization & Machine Learning

Dr. David Koop

Different Data Layouts

	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

Initial Data

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

Transpose

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

Tidy Data

[H. Wickham, 2014]

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

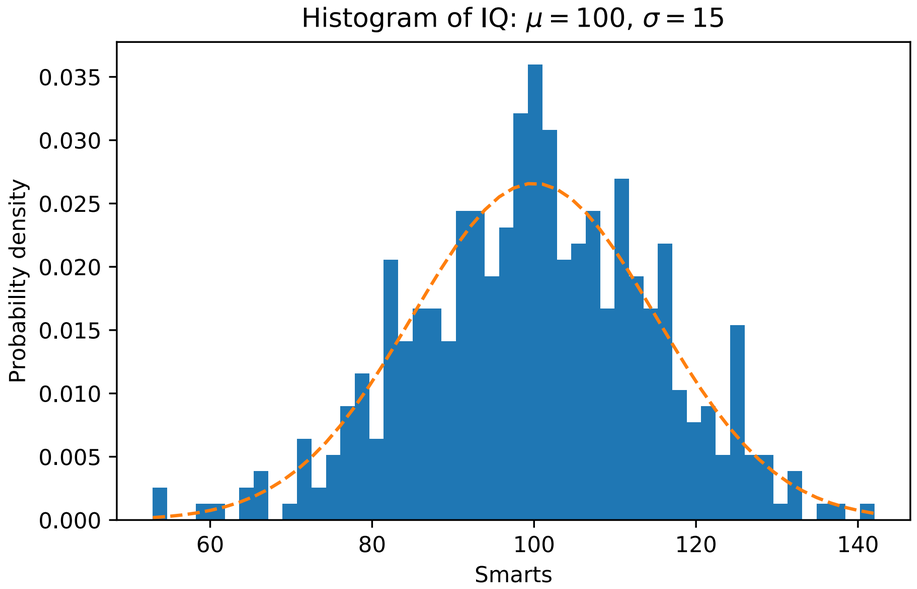
Visualization Goals

- "The purpose of visualization is **insight**, not pictures" – B. Schneiderman

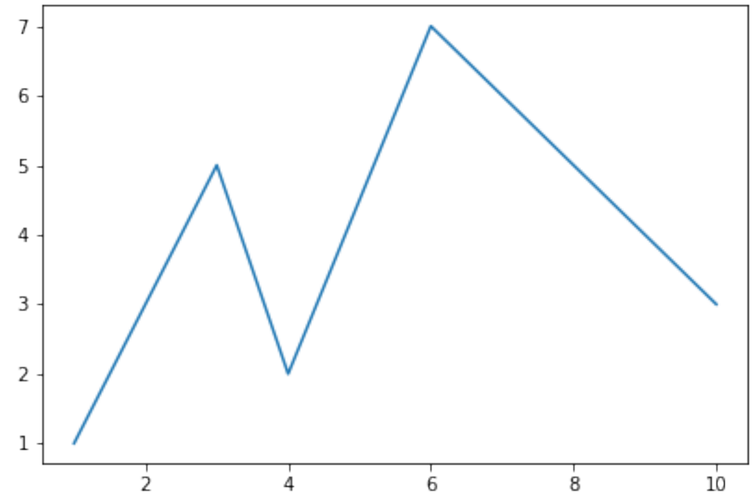
- Identify patterns, trends
- Spot outliers
- Find similarities, correlation

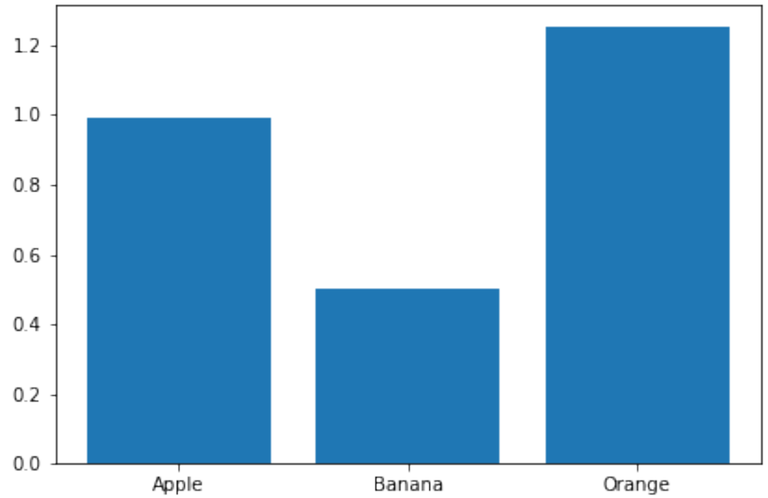
matplotlib

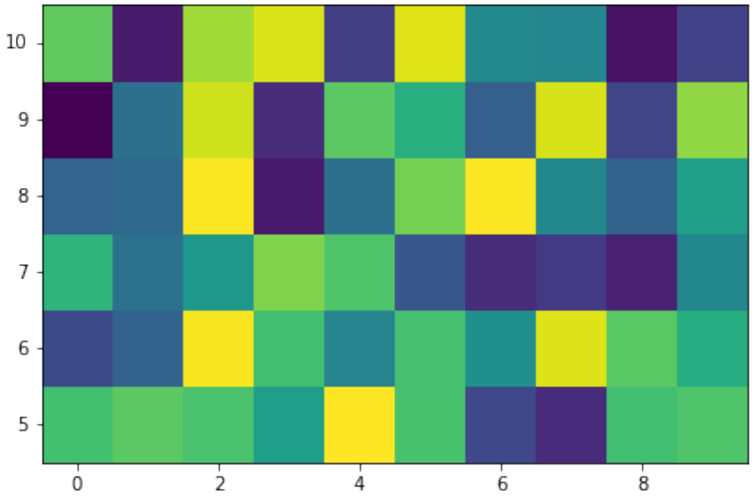
- Strengths:
  - Designed like Matlab
  - Many rendering backends
  - Can reproduce almost any plot
  - Proven, well-tested
- Weaknesses:
  - API is imperative
  - Not originally designed for the web
  - Dated styles

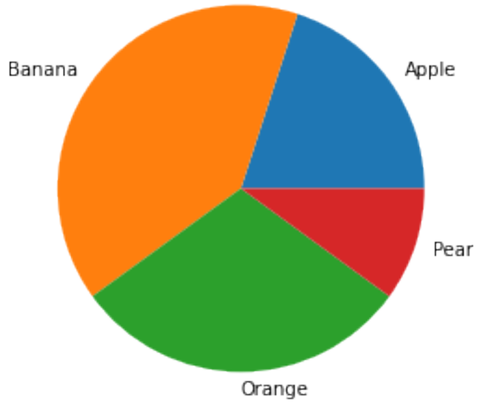


Many different types of charts









Many different types of charts

- Bar chart
  - `plt.bar(['Apple','Banana','Orange'],[0.99,0.50,1.25])`
- Grid Heatmap
  - `plt.pcolormesh(x, y, Z)`
- Pie chart:
  -
```
plt.pie([20,40,30,10],
        labels=['Apple','Banana','Orange','Pear'])
```

Altair

- ```
  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:

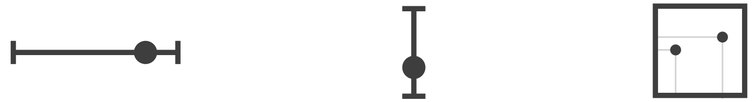
Points **Lines** **Areas**

- 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]

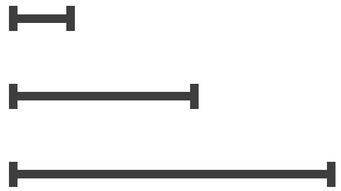
Data Attributes and Altair Types

➔ Categorical

➔ Ordered

➔ *Ordinal*

➔ *Quantitative*



[Munzner (ill. Maguire), 2014]

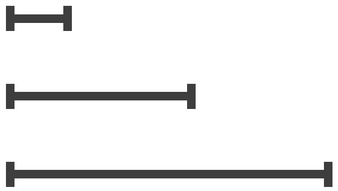
Data Attributes and Altair Types

➔ Categorical

➔ Ordered

➔ *Ordinal*

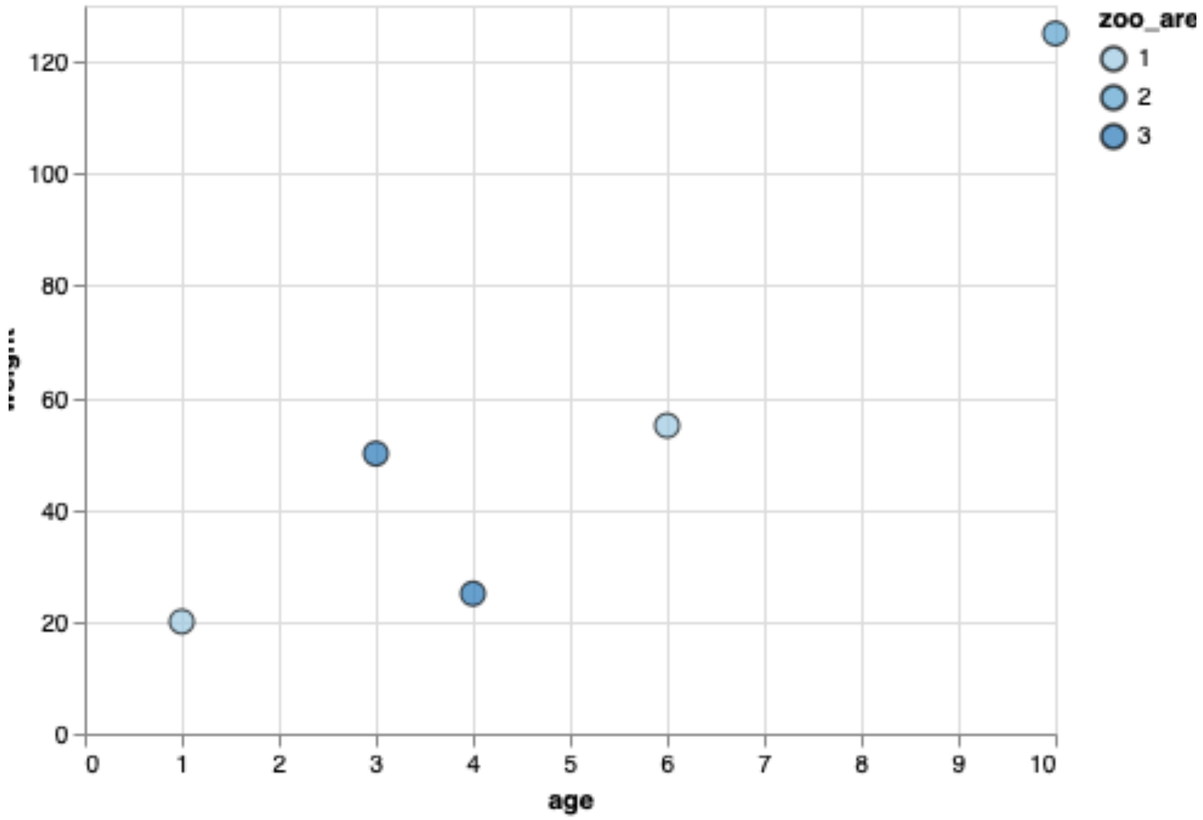
➔ *Quantitative*



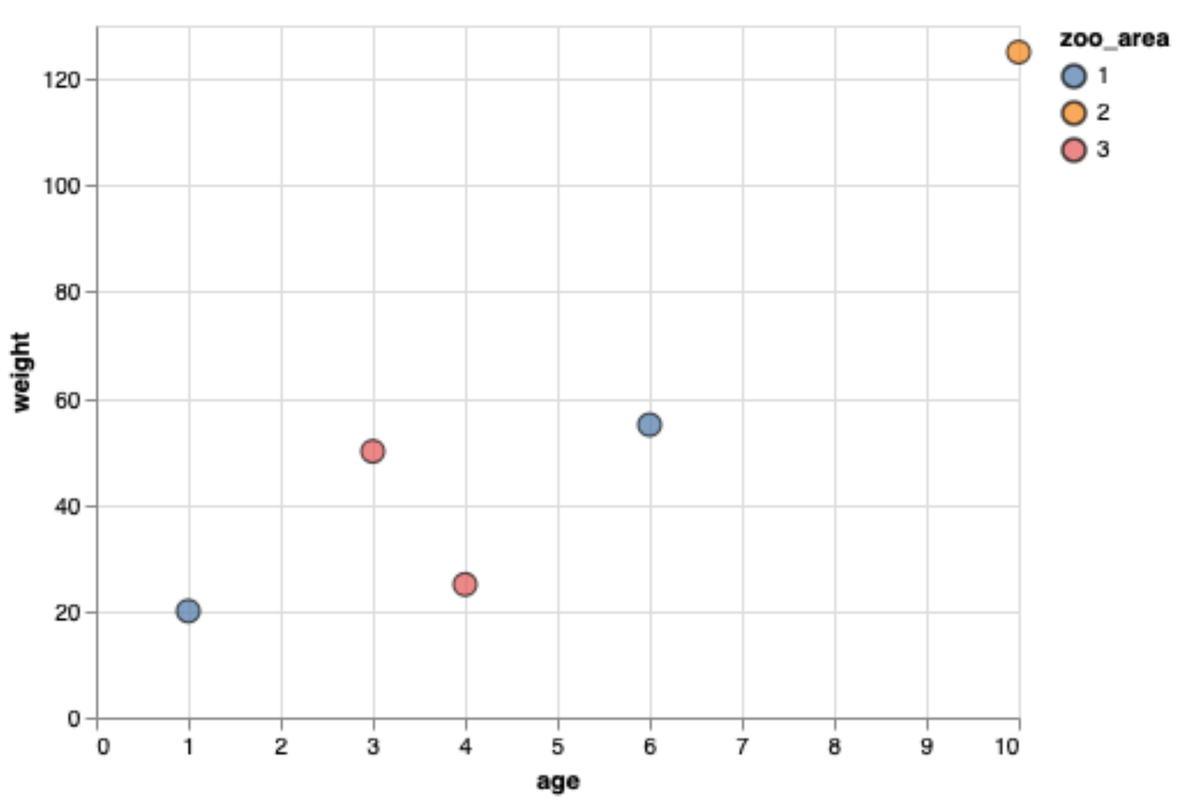
- Categorical data = Nominal (N)
- Ordinal data = Ordinal (O)
- Quantitative data = Quantitative (Q)
- Temporal data = Temporal (T)

[Munzner (ill. Maguire), 2014]

Specifying the Type



`zoo_area:O`

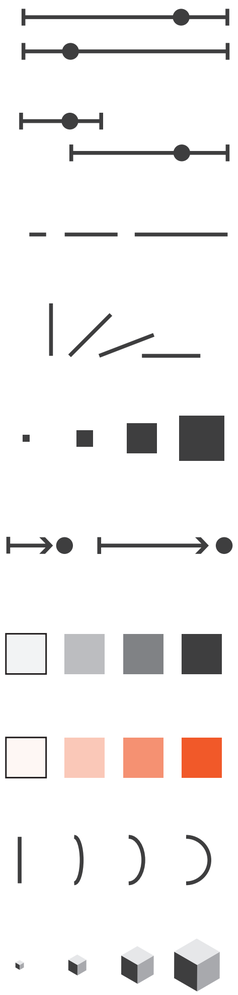


`zoo_area:N`

Different Channels for Different Attribute Types

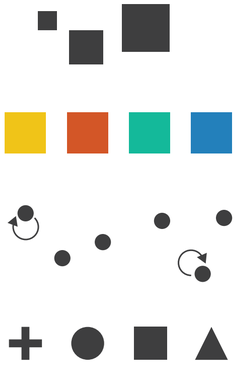
Magnitude Channels: Ordered Attributes

- 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

- Data and Visualization
- Work with polars or pandas
- Must use seaborn/matplotlib and altair (specified for problems)

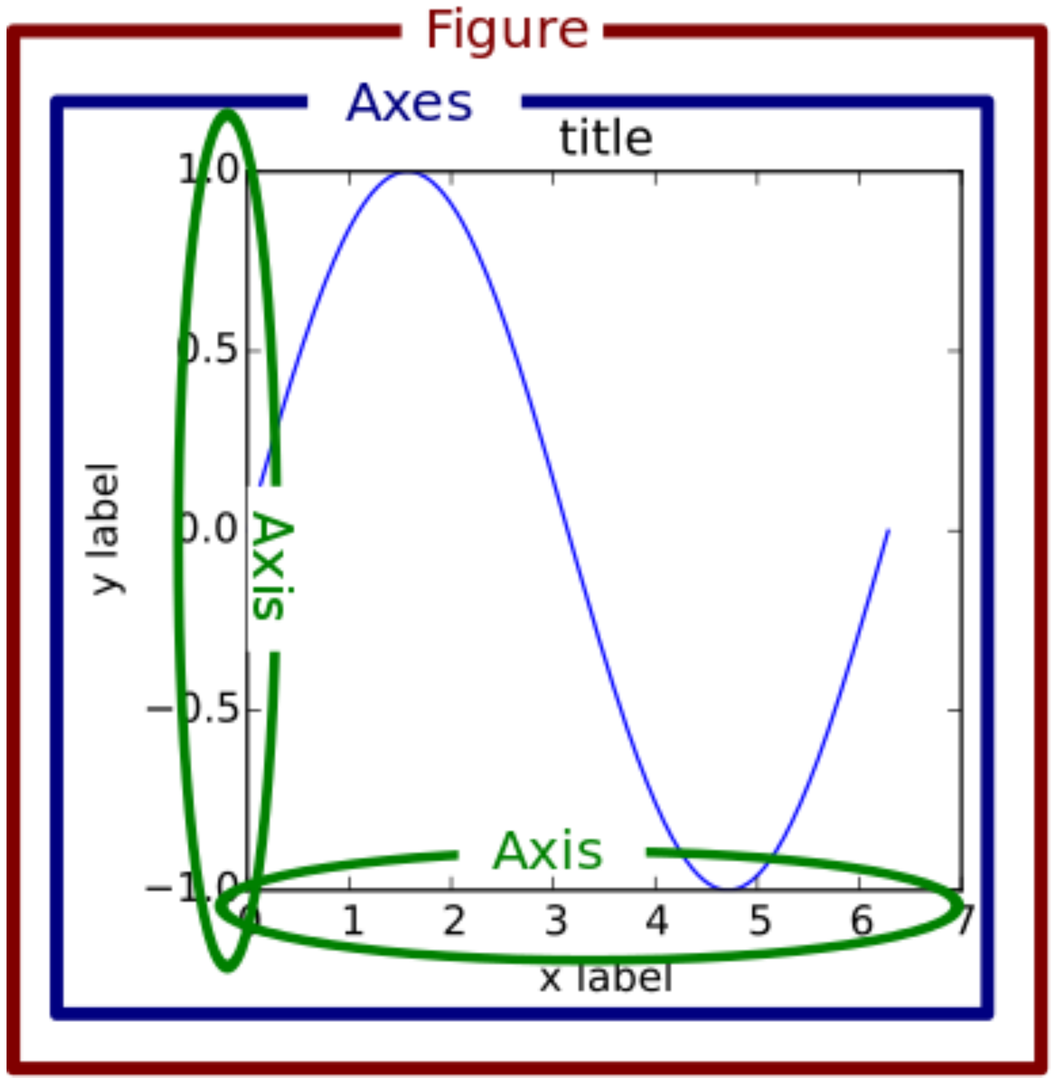
Final Exam

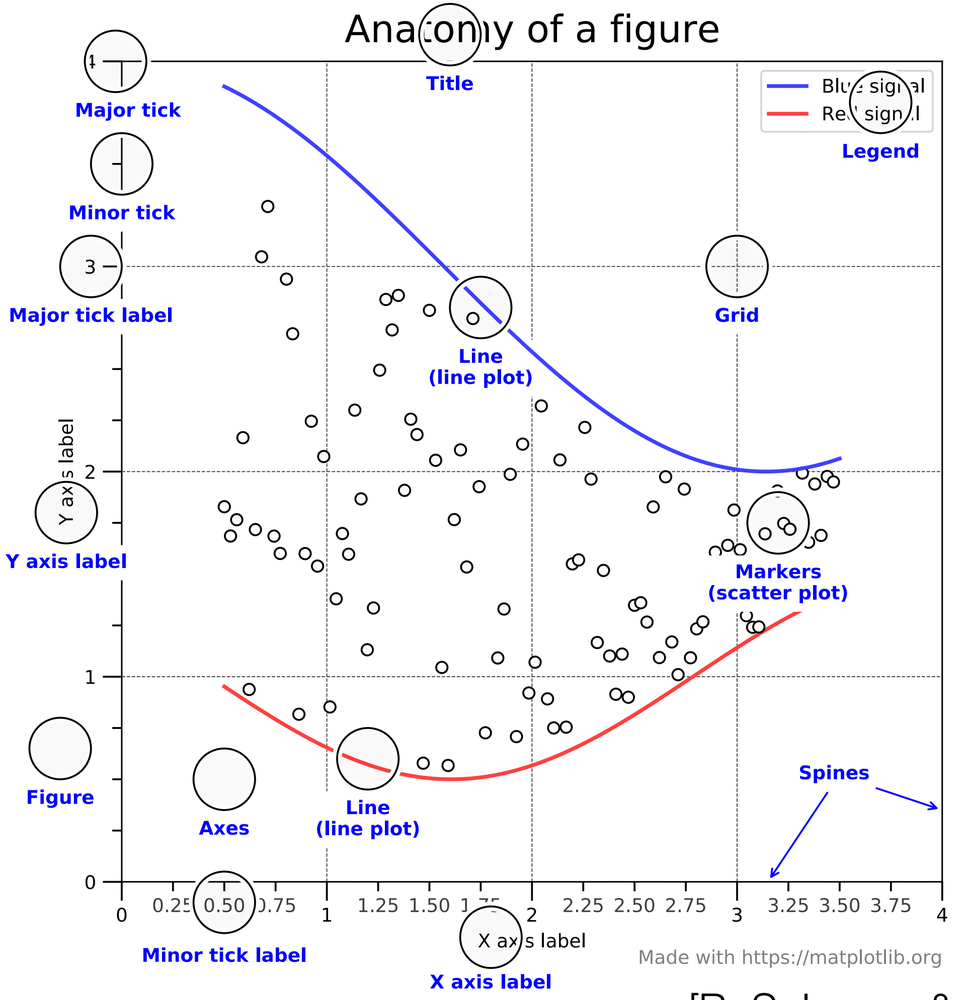
- Wednesday, December 10, **8:00**-9:50am in PM 103
- **More comprehensive** than Test 2
- Expect questions from topics covered on Test 1 and 2
- Expect questions from the last few weeks of class (concurrency, structure pattern matching, data, visualization, machine learning)
- Similar format

Adding Labels

- `plt.xlabel`: set x label
- `plt.ylabel`: set y label
- `plt.title`: set title
-
```
plt.plot([1,3,4,6,10],[1,5,2,7,3])
plt.xlabel('Age')
plt.ylabel('Number of Jumps')
plt.title('Kangaroo Jumps Today')
```

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])
```

Object-Based Plotting

- `fig, ax = plt.subplots() # "constructor-like" method`
  `ax.scatter([1,3,4,6,10],[1,5,2,7,3])`
- Use getters/setters for labels and title
  - `ax.set_xlabel('Age')`
  - `ax.set_ylabel('Number of Jumps')`
  - `ax.set_title('Kangaroo Jumps Today')`
- We can also call methods on the figure:
  - `fig.tight_layout() # reduce margins`

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'])
  ```
  ```

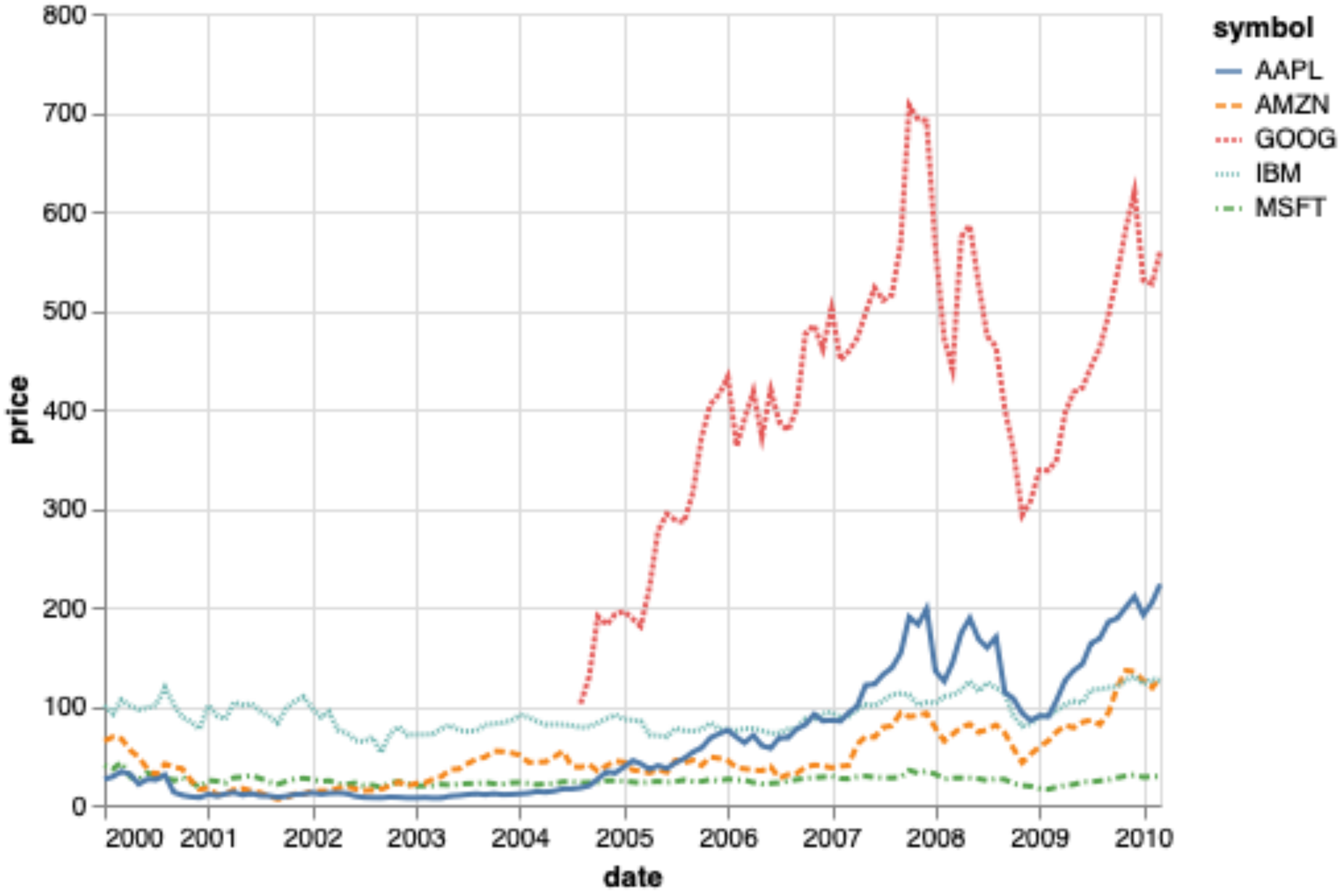
pandas Integration

- Can call many of these methods directly from pandas
- Handled through `kind` kwarg or `.plot` accessor
- It will try to guess a reasonable visualization, but may fail:
  - `fruit.plot()`
- Instead, specify x and y and other parameters:
  - `fruit.plot(kind='bar',x='name',y='price')`
  - `plt.bar(x='name',height='price',data=fruit) # SIMILAR`
  - `fruit.plot.scatter(x='price',y='count',c='name') # ERROR`
  -
```
colors = {'Apple': 'red','Orange': 'orange',
          'Banana': 'yellow','Pear': 'green'}
fruit.plot.scatter(x='price',y='count',
                   c=fruit['name'].map(colors))
```

Extensions & Other Directions

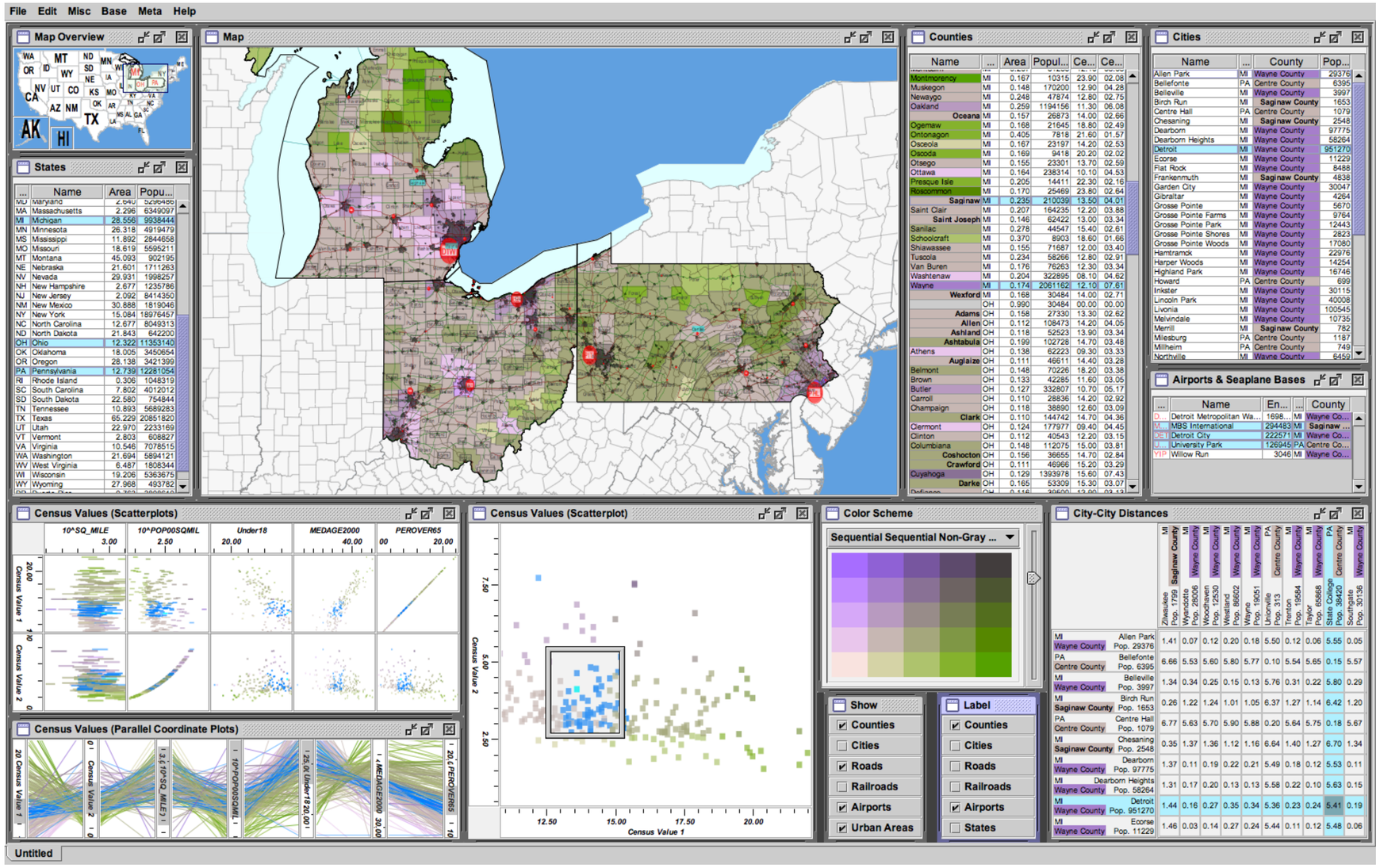
- Seaborn:
  - ```
    import seaborn as sns
    sns.scatterplot(x='price',y='count',hue='name', data=fruit)
    ```
    ```

Multiple Views in Visualization



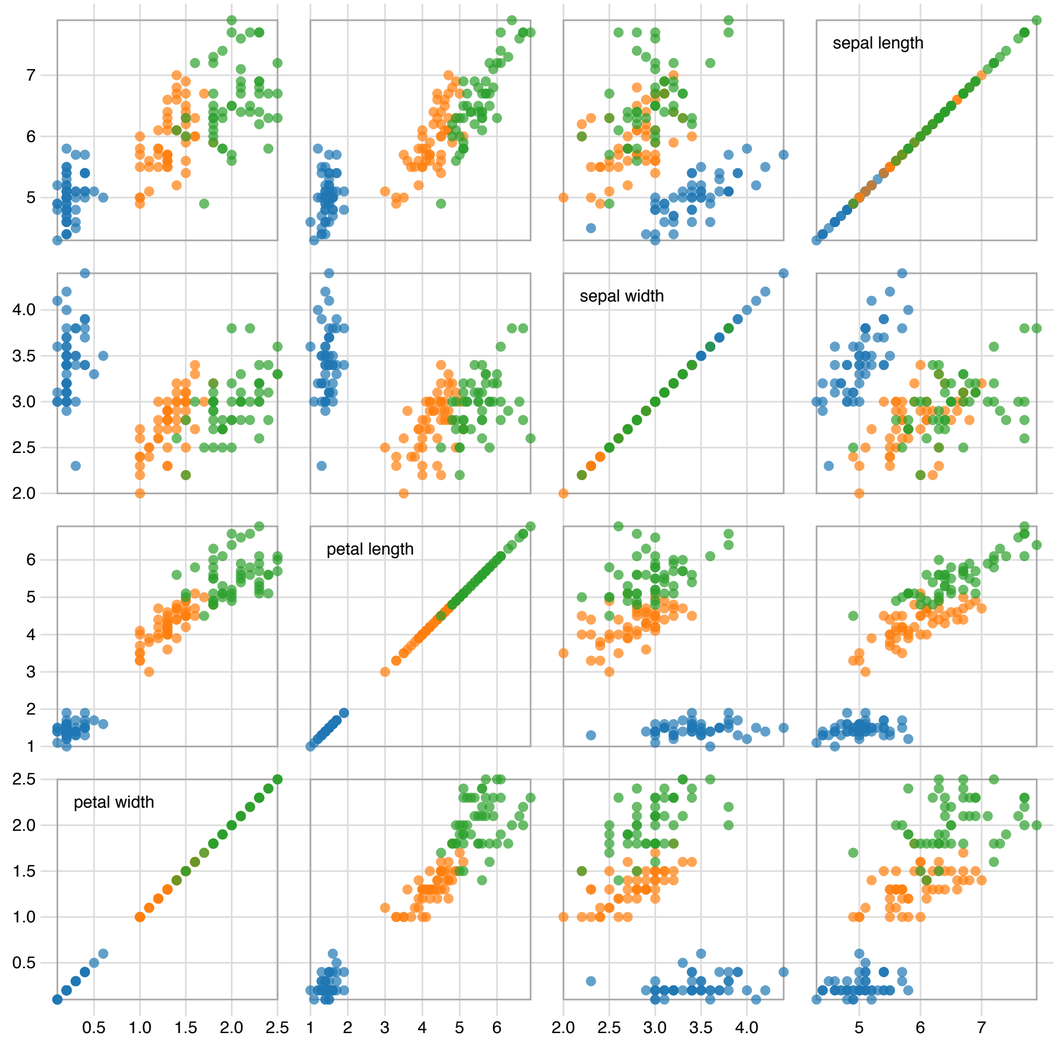
[Altair]

Multiple Views in Visualization



[Improvise, Weaver, 2004]

Multiple Views in Visualization

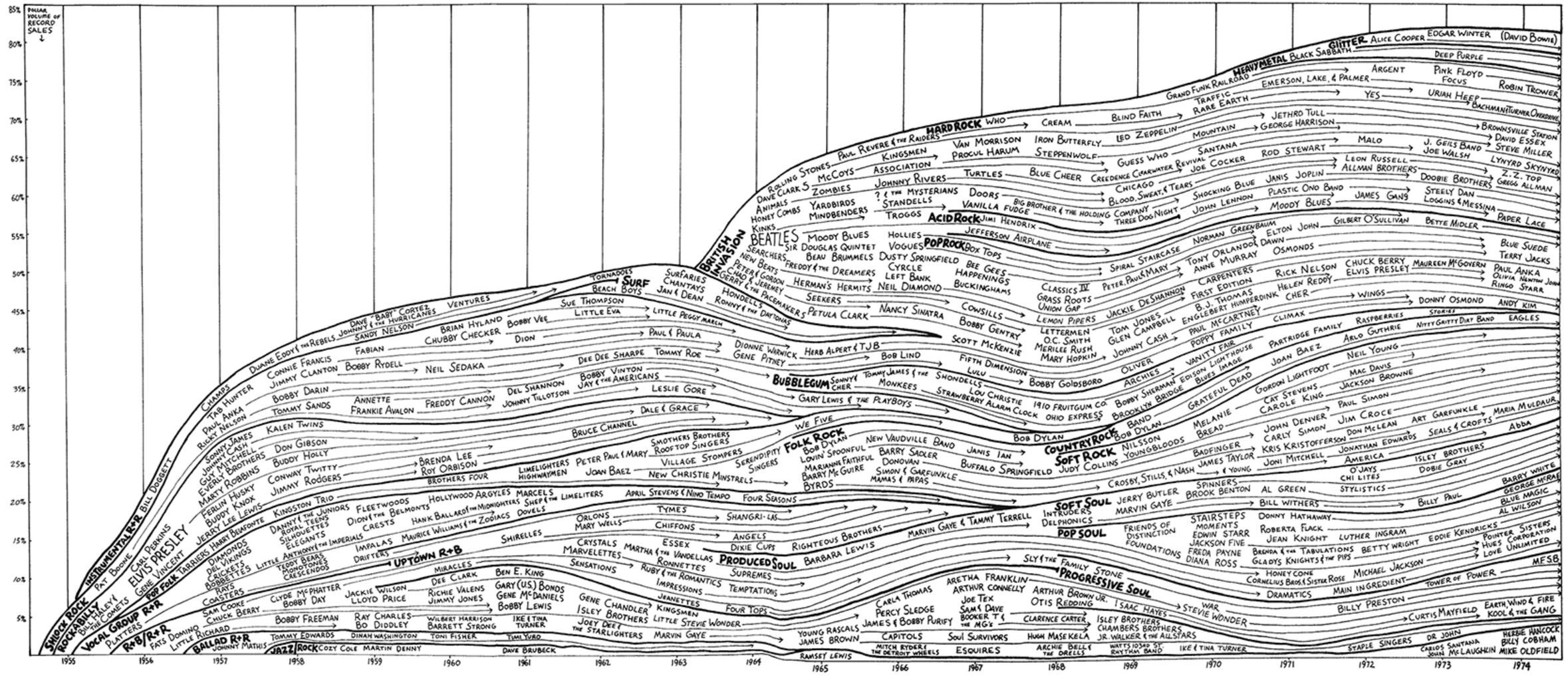


[M. Bostock]

Altair Supports Concatenation, Layering, & Repetition

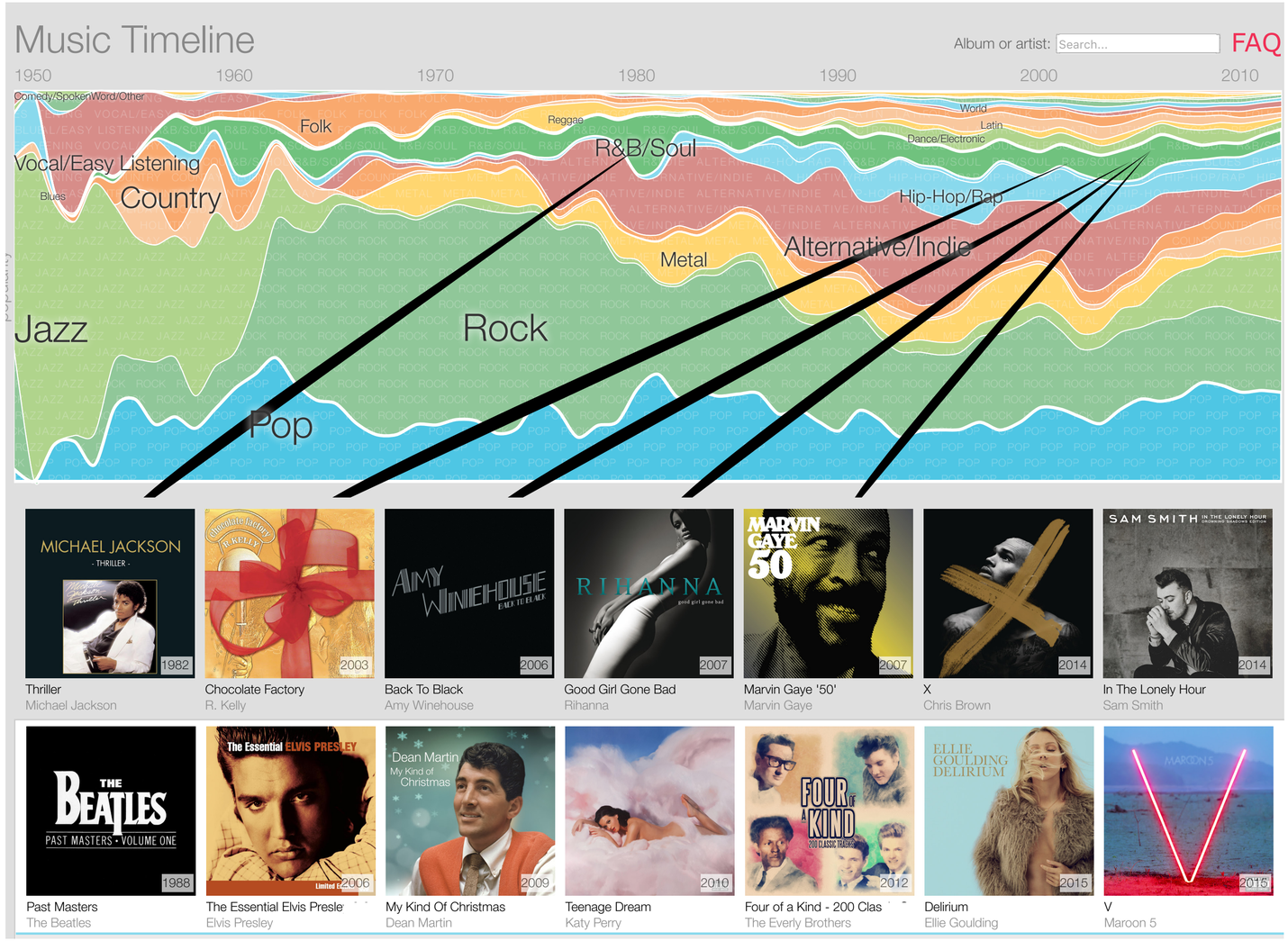
- Layering:
  - + Operator
- Concatenation:
  - Horizontal: | operator
  - Vertical: & operator
- Repetition
  - Use of .repeat for layout
  - Reference repeated variables in the encoding

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

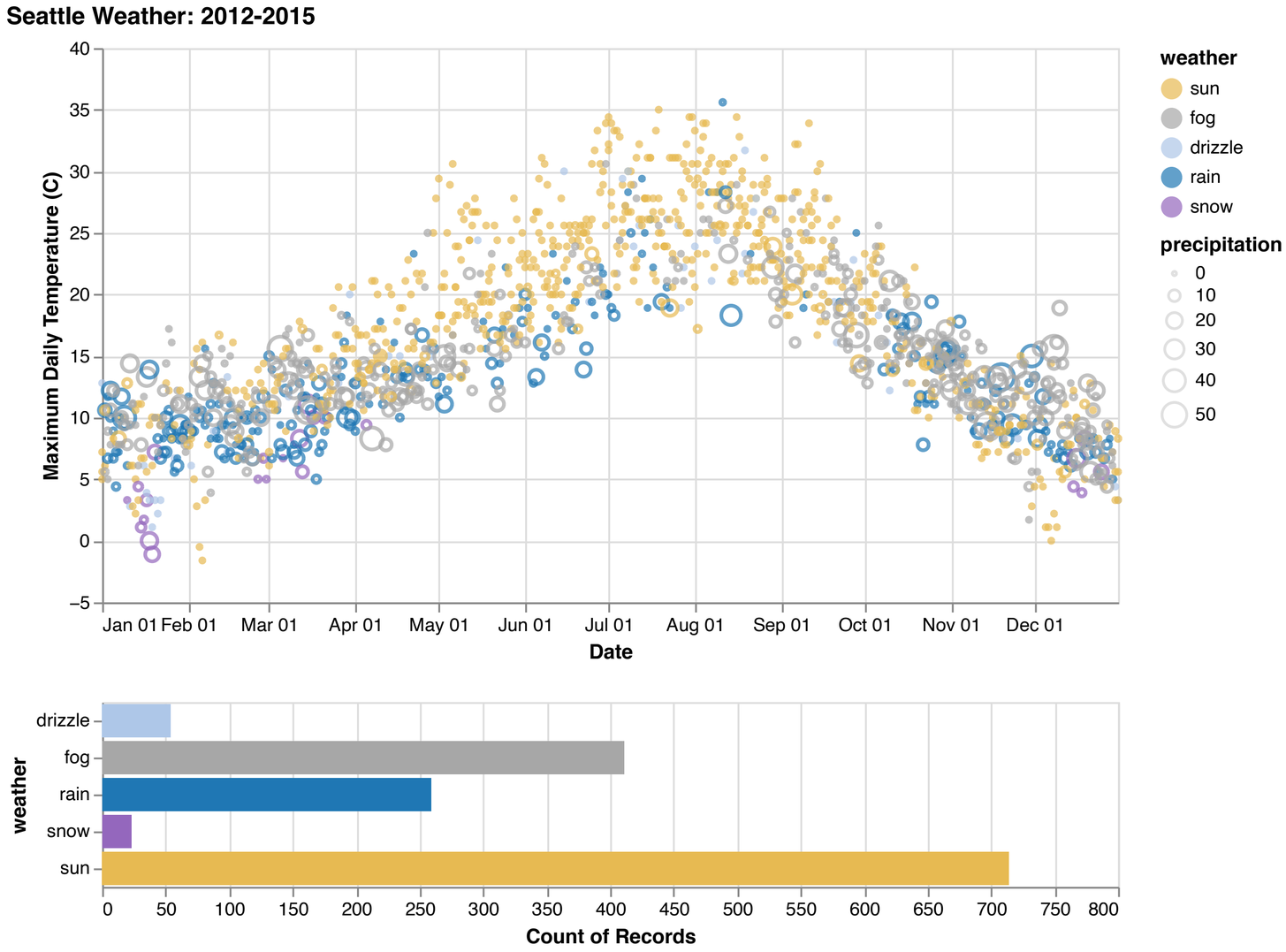
Highlighting

- 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

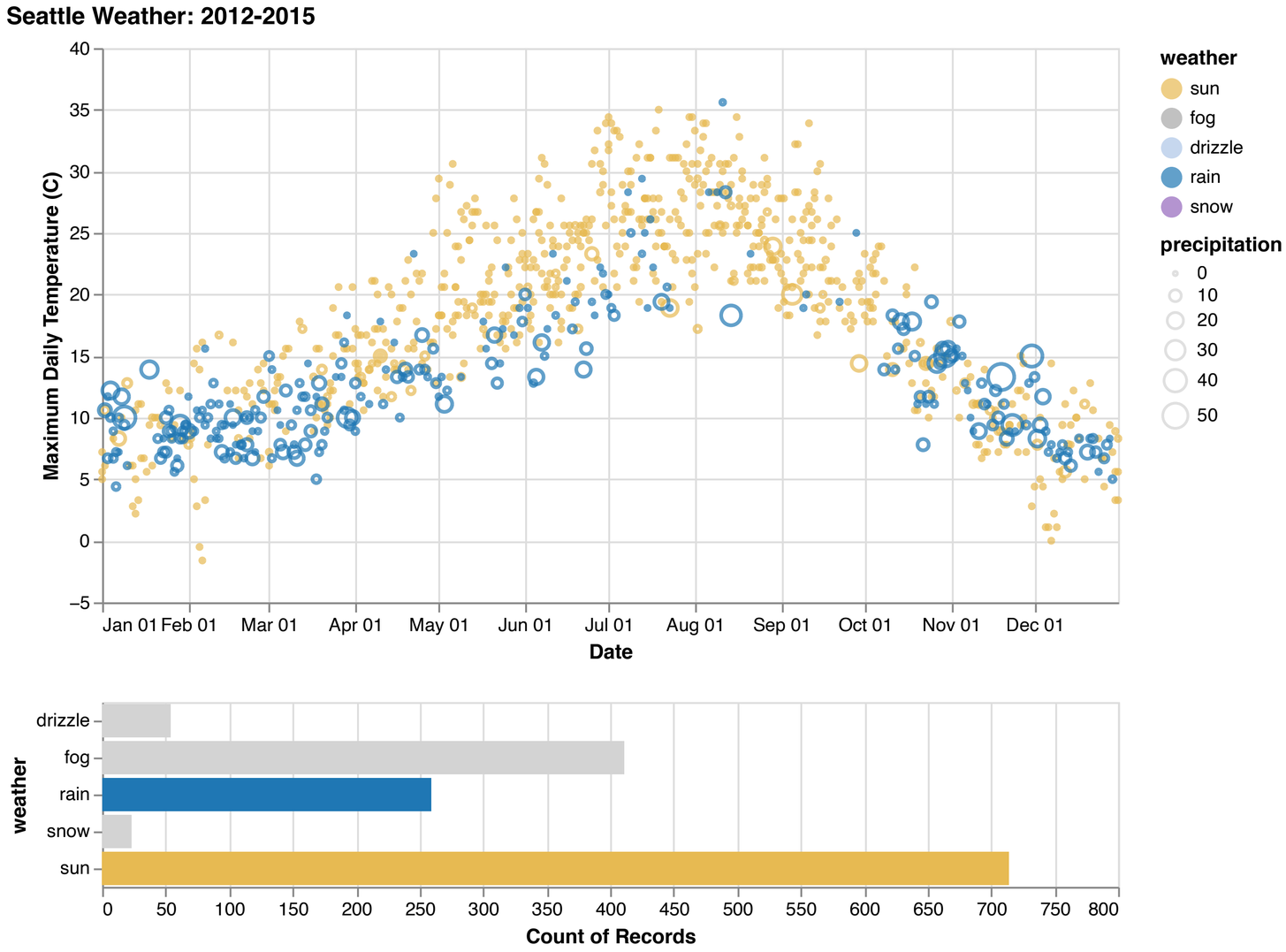
Altair's Interactive Charts

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

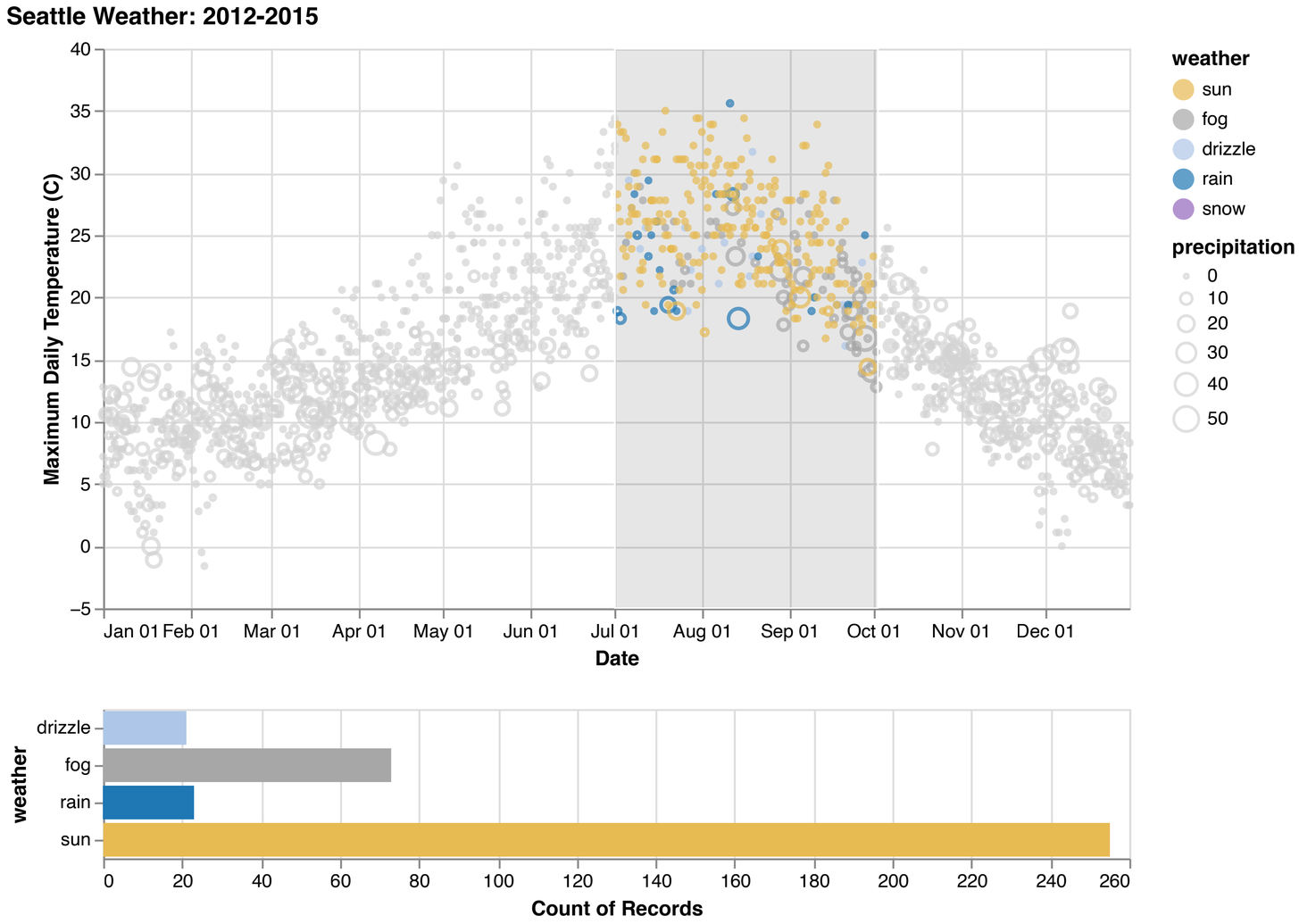
Interaction



Weather Selection: Rain vs. Sun



Date Selection: July-September Sun



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]

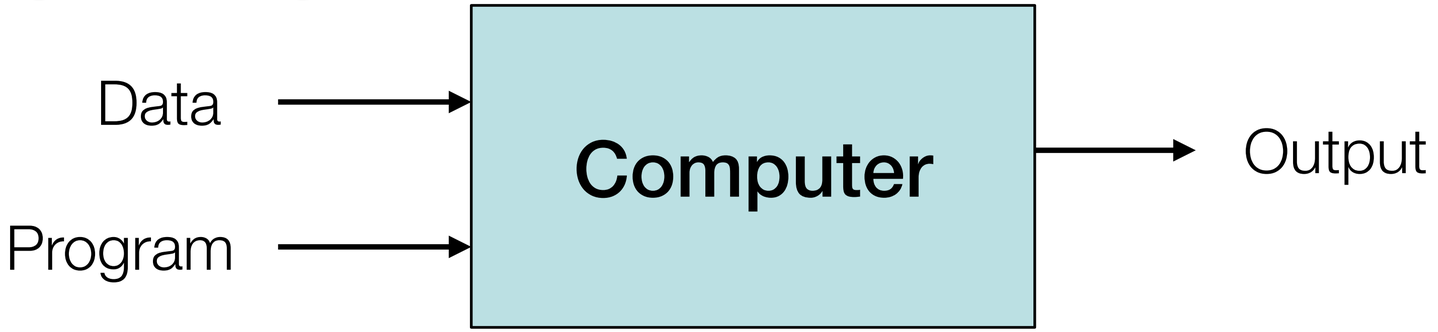
Machine Learning Workflow Overview

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).

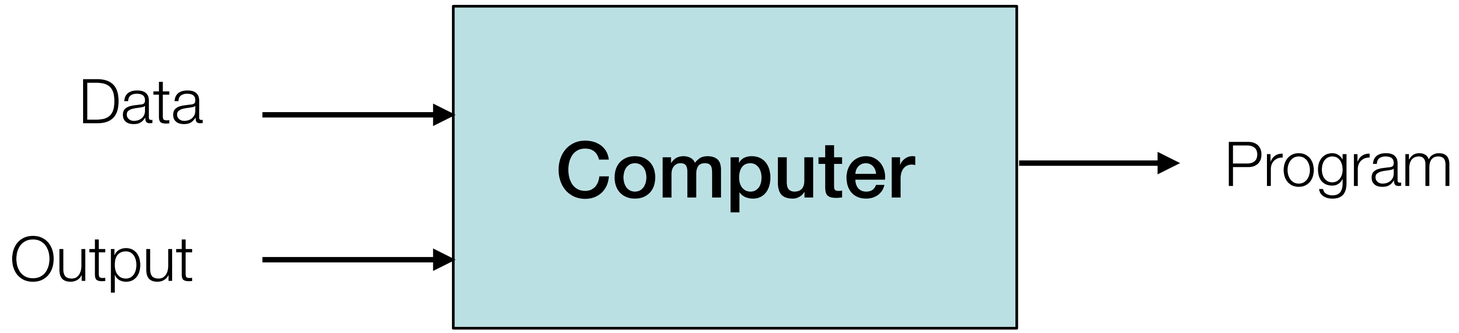
[R. Grosse et al.]

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

[P. Domingos]