# Programming Principles in Python (CSCI 503/490)

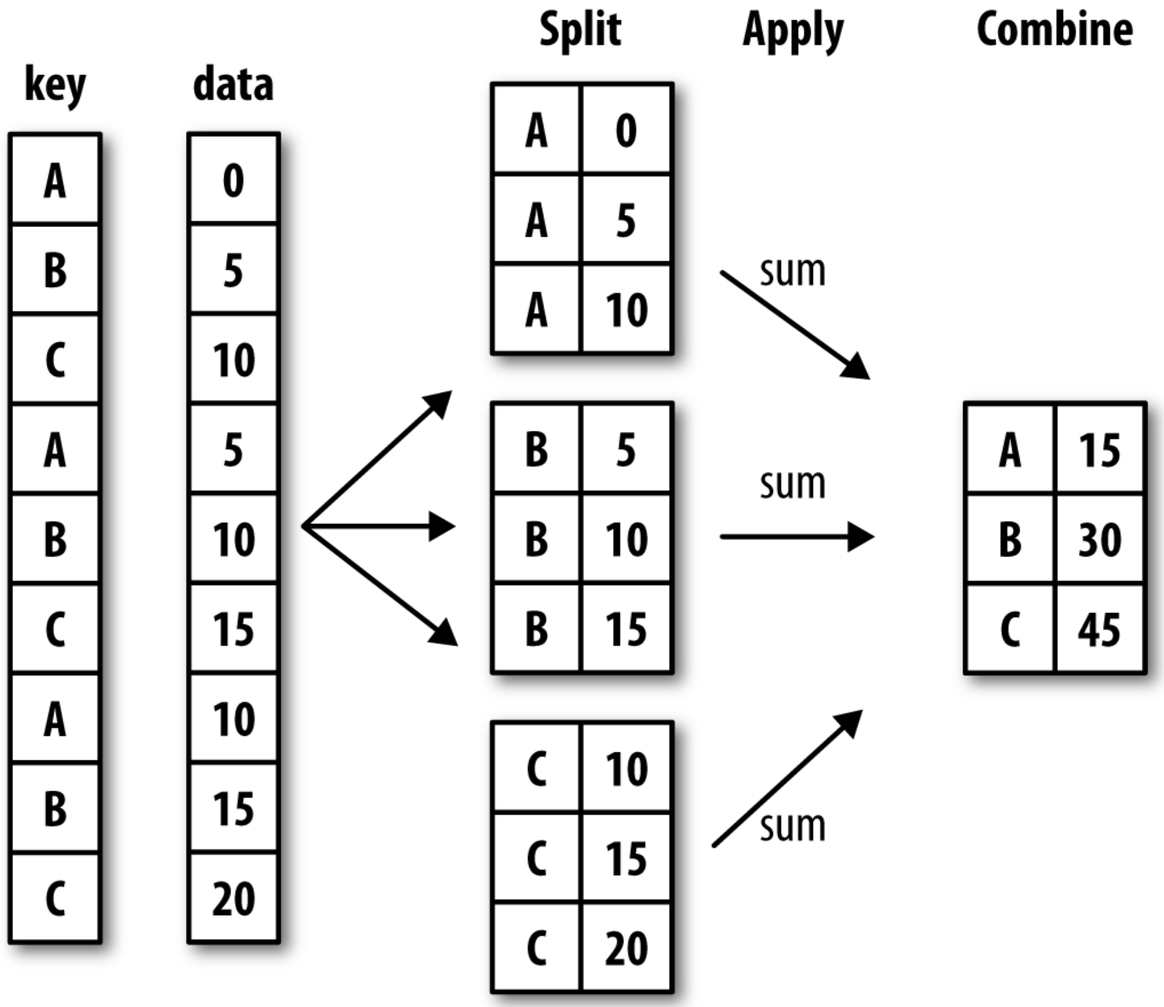
## Data Visualization

Dr. David Koop

# Reading and Writing Files Using polars & pandas

- polars uses `read_*/write_*`
- pandas uses `read_*/to_*`
- Many options available, format dependent!
- polars csv example:
  - `df = pl.read_csv(<fname>)`
  - `df.write_csv(<fname>)`
- pandas csv example:
  - `dfa = pd.read_csv(<fname>)`
  - `dfa.to_csv(<fname>)`
- Both pandas and polars can read/write to the cloud (e.g. S3)

# Split-Apply-Combine




[W. McKinney, Python for Data Analysis]

# Split-Apply-Combine

- Polars:
  - `df.group_by('Island').agg(pl.col('Length').mean())`
  - `df.group_by('Island').agg(pl.col('Length','Depth').mean())`
  - `df.group_by('Island').agg(pl.col('Length').min().alias('LMin'), pl.col('Length').max().alias('LMax'))`
- Pandas:
  - `dfa.groupby('Island')['Length (mm)'].mean()`
  - `dfa.groupby('Island')[['Length','Depth']].mean()`
  - `dfa.groupby('Island').agg({'Length': ['min','max']})`
  - `dfa.groupby('Island').agg(LMin=('Length','min') LMax=('Length','max'))`

# 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 |

# String and DateTime Accessors

- String: `.str`
  - Apply various string methods to string-like columns
- Datetime: `.dt`
  - Apply various datetime methods to datetime-like columns

# List Columns and Explode

- Both libraries support lists
  - pandas treats them as objects but can use `str` accessor to perform some operations like slicing
  - polars incorporates them in the schema (e.g. `pl.List(pl.String)`) and uses the list accessor for a variety of operations
- `explode` takes each item from a column of lists and creates multiple copies for each row, one row for each item in the list

# Join

- Like a database join, allows you to combine two different dataframes into one
- Need to identify the columns on which to join
- polars:
  - `df1.join(df2, on=<column(s)>)`
- pandas:
  - `dfa1.merge(dfa2, on=<column(s)>)`
  - If you want to join on an **index** column, use `left/right_index`
- Also various methods for how to do the join:
  - `inner`, `outer`, `left`, `right`, `anti`

# Assignment 8

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

# Final Exam

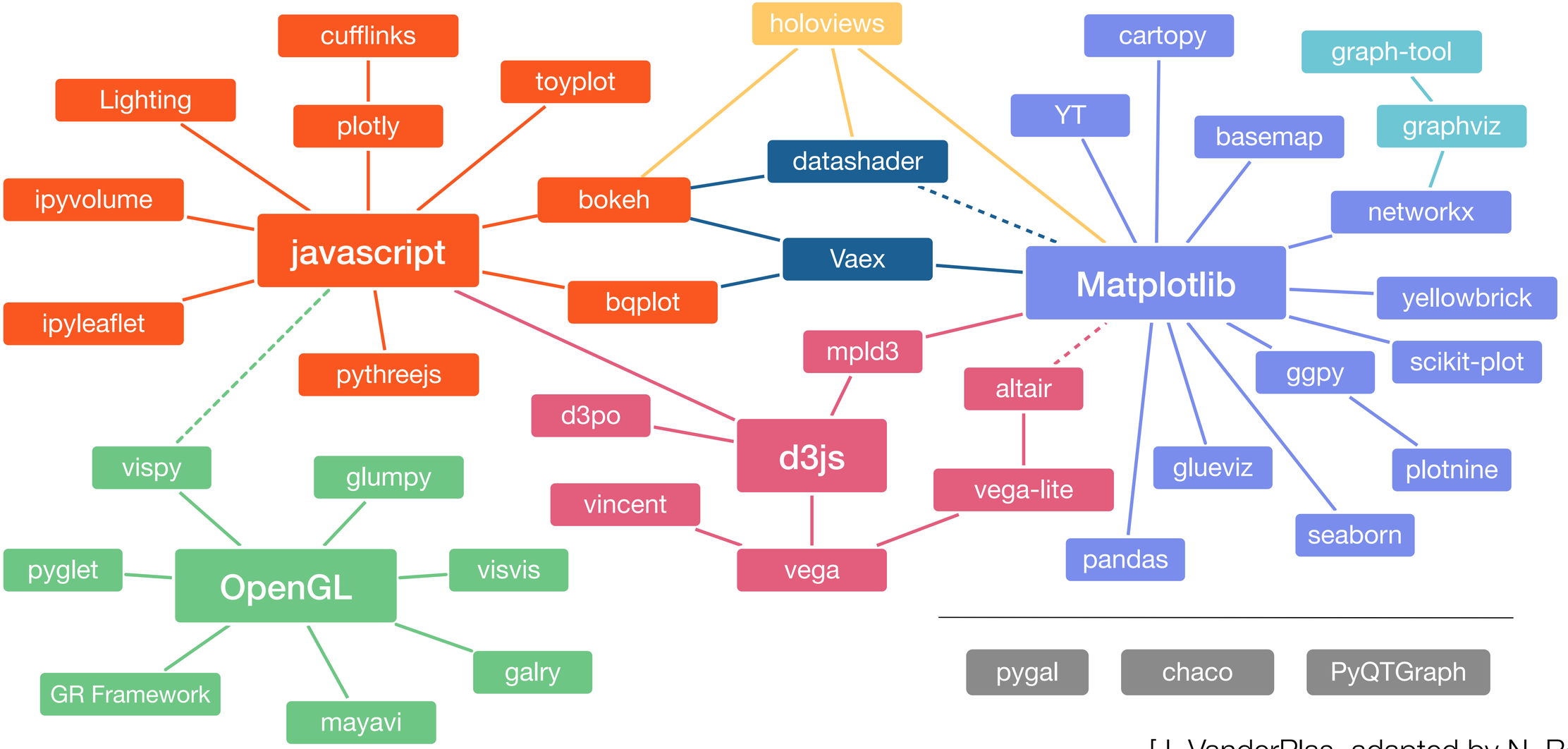
- In-Person (CSCI 503-1 & 490): Wednesday, May 6, **8:00**-9:50am in PM 203
- Online (CSCI 503-2): Wednesday, May 6
- **More comprehensive** than Test 2
- Expect questions from topics covered on Test 1 and 2
- Expect questions from the last few weeks of class (concurrency, structure pattern matching, data, visualization, machine learning)
- Similar format

# Visualization Goals

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

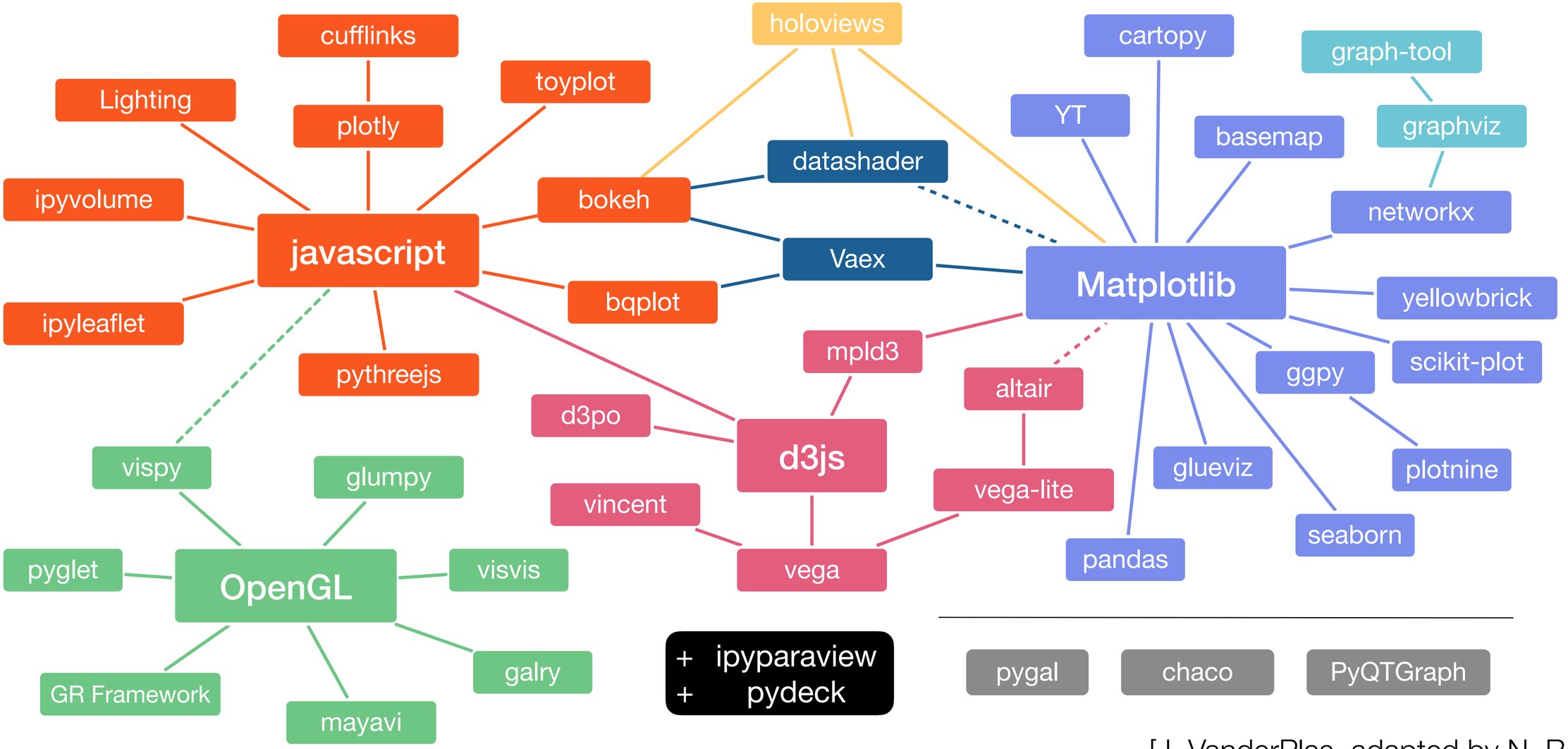
- Identify patterns, trends
- Spot outliers
- Find similarities, correlation

# The Python Visualization Landscape




[J. VanderPlas, adapted by N. Rougier]

# The Python Visualization Landscape




[J. VanderPlas, adapted by N. Rougier]

# Matplotlib History

- "In the beginning was matplotlib" – J. VanderPlas
- Started by John D. Hunter, a neurobiologist ~2003
- John tragically passed away in 2012, community-led now

- Before Python, John had Perl scripts that called C++ mathematical programs that wrote data files that were plotted using Matlab (then gnuplot)
- Sought a solution that was Matlab users would be more comfortable with
  - Imports "hidden" by importing into the global namespace
  - pylab mode: match terminology of Matlab (at the cost of overriding core python functions/definitions)

# Lots of Changes Since

- pylab is "strongly discouraged nowadays and deprecated." [docs]
- stateful plotting using pyplot still exists, but…
- also object-oriented methods to build and customize plots now
- Integrated output in JupyterLab
- Many derivative libraries (e.g. seaborn) that build on matplotlib core
- Can use more directly from pandas

# Basic Example

- `import matplotlib.pyplot as plt`
  `plt.plot([1,5,2,7,3])`
- Default is line plot
- x-values are implicit (`range(5)`)
- Can add x-values
  - `plt.plot([1,3,4,6,10],[1,5,2,7,3])`
- Can change type of plot
  - `plt.scatter([1,3,4,6,10],[1,5,2,7,3])`
  - `plt.plot([1,3,4,6,10],[1,5,2,7,3],'o') # format string`

# Plot Formats

- Can specify color, marker, and linestyle in format string
  - `plt.plot([1,3,4,6,10],[1,5,2,7,3],'ro—')`
- Can also specify these via keyword arguments:
  - 
```
plt.plot([1,3,4,6,10],[1,5,2,7,3],
         color='red', marker='s', linestyle='dashed')
```
- Other keyword arguments, too:
  - 
```
plt.plot([1,3,4,6,10],[1,5,2,7,3],
         color='red', marker='s', linestyle='dashed',
         linewidth=3, markersize=12)
```

# Format Reference

| string | color |
|---|---|
| 'b' | blue |
| 'g' | green |
| 'r' | red |
| 'c' | cyan |
| 'm' | magenta |
| 'y' | yellow |
| 'k' | black |
| 'w' | white |

Color shortcuts

| string | description |
|---|---|
| '-' | solid |
| '--' | dashed |
| '-.' | dash-dot |
| ':' | dotted |

Line Styles

| string | description |
|---|---|
| '.' | point |
| ',' | pixel |
| 'o' | circle |
| 'v' | triangle_down |
| '^' | triangle_up |
| '<' | triangle_left |
| '>' | triangle_right |
| '1' | tri_down |
| '2' | tri_up |
| '3' | tri_left |
| '4' | tri_right |
| '8' | octagon |
| 's' | square |

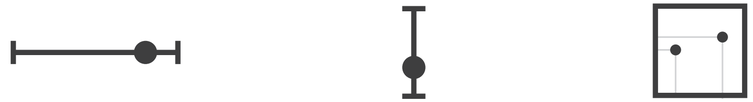
| string | description |
|---|---|
| 'p' | pentagon |
| 'P' | plus (filled) |
| '*' | star |
| 'h' | hexagon1 |
| 'H' | hexagon2 |
| '+' | plus |
| 'x' | x |
| 'X' | x (filled) |
| 'D' | diamond |
| 'd' | thin_diamond |
| '\|' | vline |
| '_' | hline |

Markers

[Documentation (Notes Section)]

# Data is Encoded via Visual Channels

- **Position**
  - Horizontal
  - Vertical
  - Both



- **Color**

- **Shape**

- **Tilt**

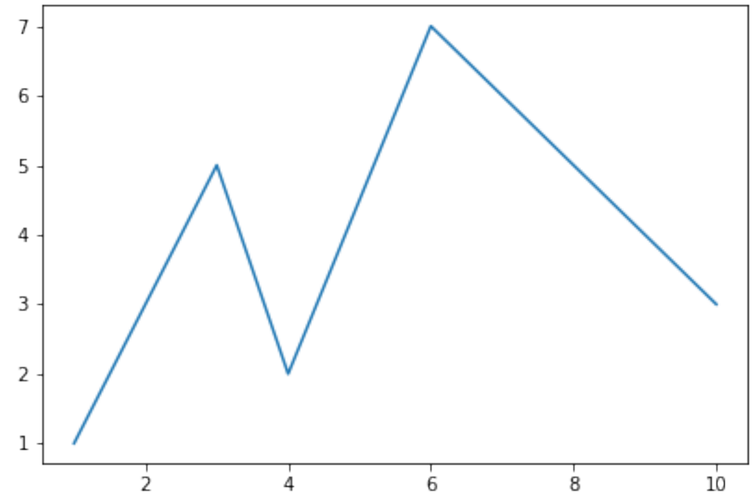
- **Size**
  - Length
  - Area
  - Volume

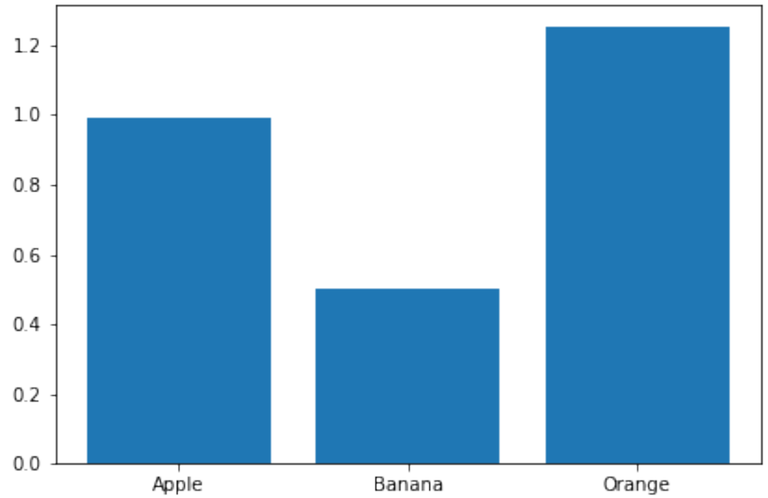
[Munzner (ill. Maguire), 2014]

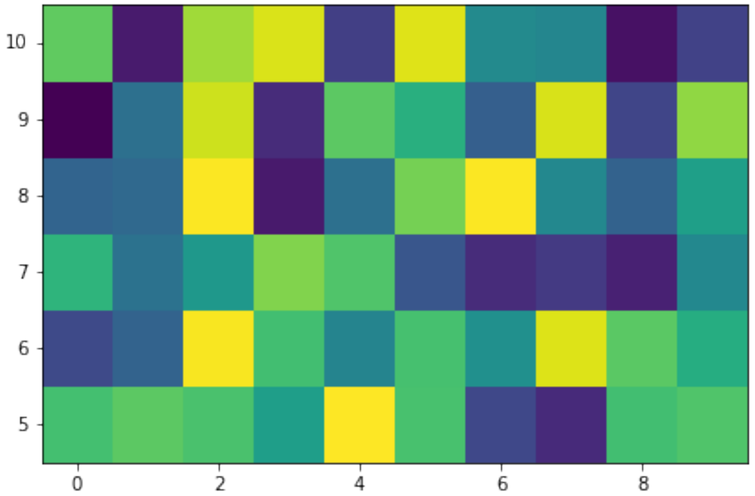
# Encoding Data Attributes via Channels

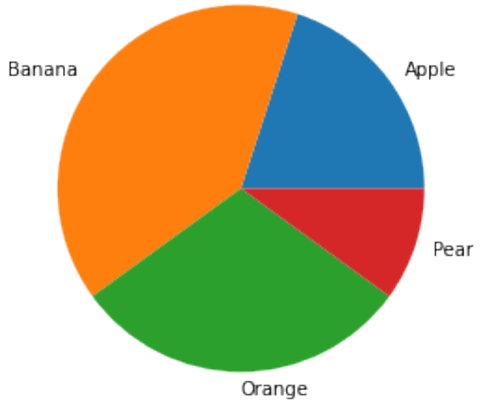
- 
```
data = {'age': [1,3,4,6,10],
        'num_jumps': [1,5,2,7,3],
        'weight': [20,50,25,55,25],
        'num_scoops': [3,2,4,2,3]}
plt.scatter('age','num_jumps',c='num_scoops',s='weight',
            data=data)
```
- `data` is a dictionary that contains information about each data item (first animal has `age=1, num_jumps=1, weight=20, num_scoops=3`)
- `x` and `y` are referenced as parts of the array
- `s` is marker size
- `c` is color and numbers are mapped to colors

# 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'])
```

# 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))
```

# matplotlib tutorials

- https://matplotlib.org/stable/tutorials/index.html
- https://github.com/rougier/matplotlib-tutorial

# History of Vega-Lite & Vega-Altair

- "Grammar of Graphics", L. Wilkinson
- "A Layered Grammar of Graphics", H. Wickham
- ggplot: plotting library for R
- Vega: similar idea for Javascript/JSON (U. Washington, A. Satyanarayan)
  - "Declarative language for creating, saving, and sharing interactive visualization designs"
  - More focus on interaction and reactive signals
  - Separation between specification and runtime
- Vega-Lite: higher-level language than Vega (U. Washington, D. Moritz)
  - uses carefully designed rules to default settings

# History of Vega-Lite & Altair

- Vega-Altair: Python interface to Vega-Lite (U. Washington, J. VanderPlas)
  - "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.
- Vega Fusion (J. Mease)
  - Scaling to larger datasets
  - Serverside scaling

# 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:

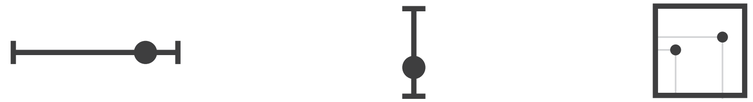
**Points** **Lines** **Areas**

- 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**

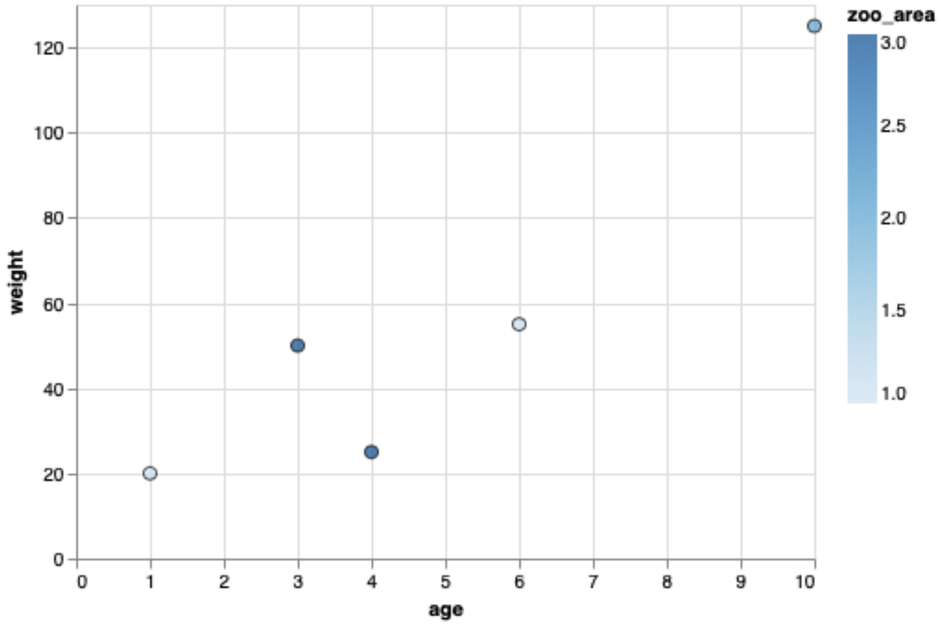
**Size**

- Length
- Area
- Volume

[Munzner (ill. Maguire), 2014]

# Easily Explore Different Encodings

```
data = pd.DataFrame({
    'age': [1,3,4,6,10],
    'weight': [20,50,25,55,125],
    'zoo_area': [1,3,3,1,2],
    'num_scoops': [3,2,4,2,3]
})
alt.Chart(data).mark_point(
    filled=True, size=50,
    stroke='black',strokeWidth=1
).encode(
    x='age',
    y='weight',
    color='zoo_area'
)
```



Problem: zoo_area is not a continuous value,
nor is it ordered in any way!

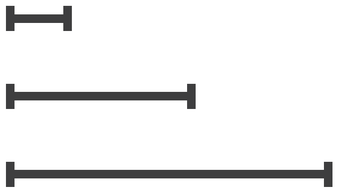
# Data Attributes and Altair Types

➔ Categorical

➔ Ordered

➔ *Ordinal*

➔ *Quantitative*



[Munzner (ill. Maguire), 2014]

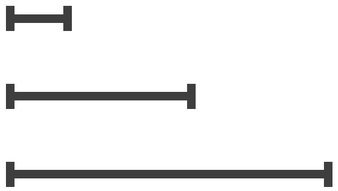
# Data Attributes and Altair Types

➔ Categorical

➔ Ordered

➔ *Ordinal*

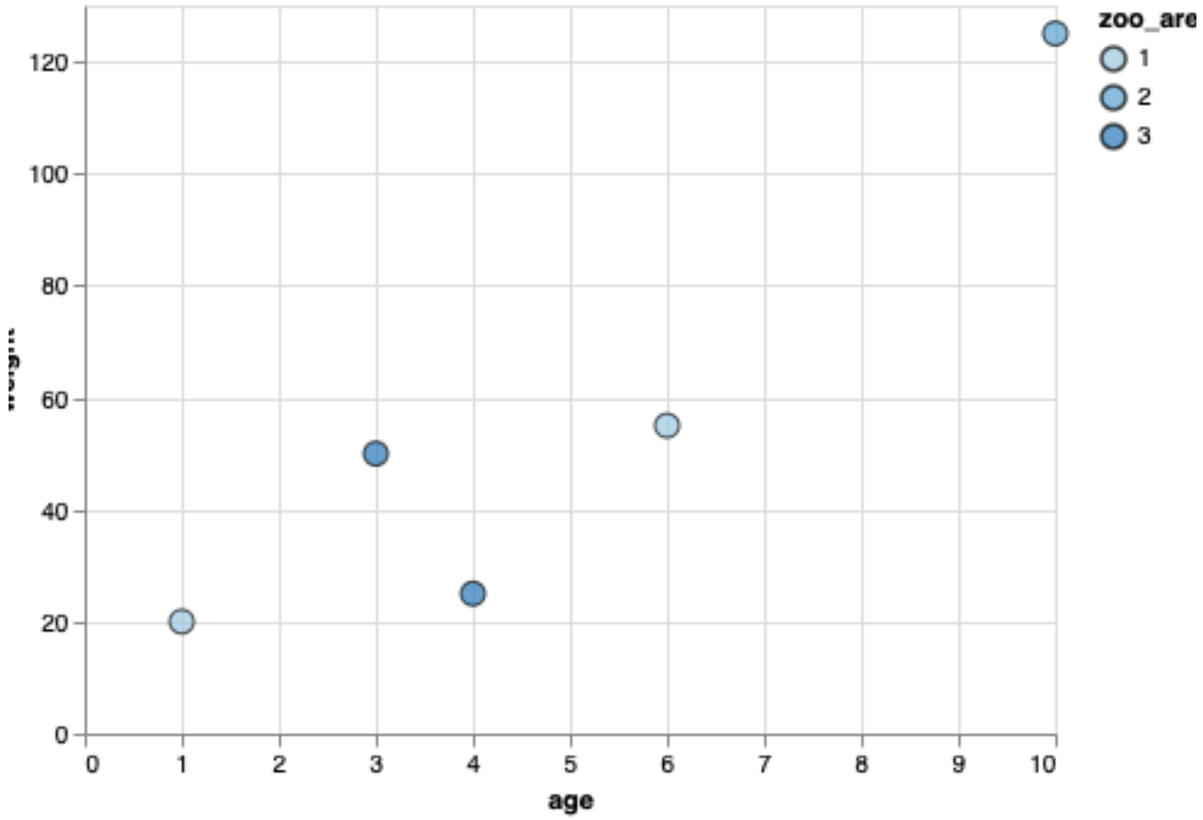
➔ *Quantitative*



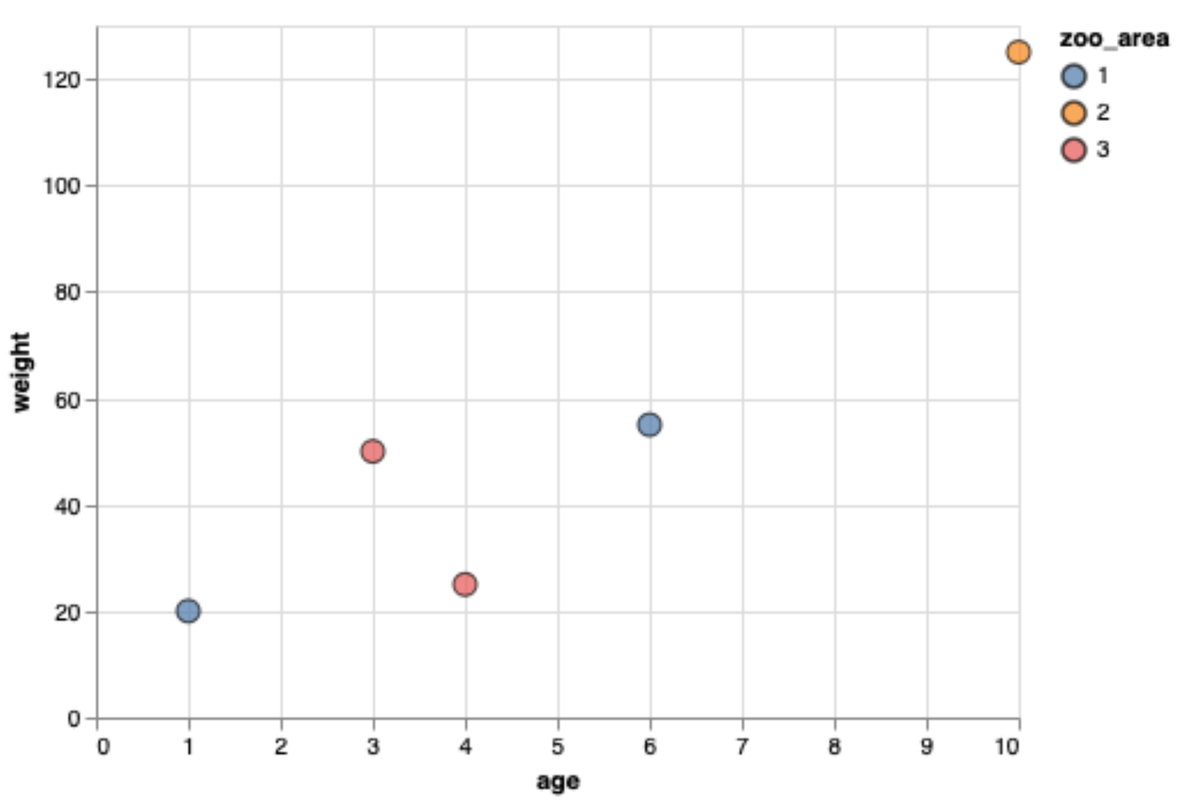
- Categorical data = Nominal (N)
- Ordinal data = Ordinal (O)
- Quantitative data = Quantitative (Q)
- Temporal data = Temporal (T)

[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)

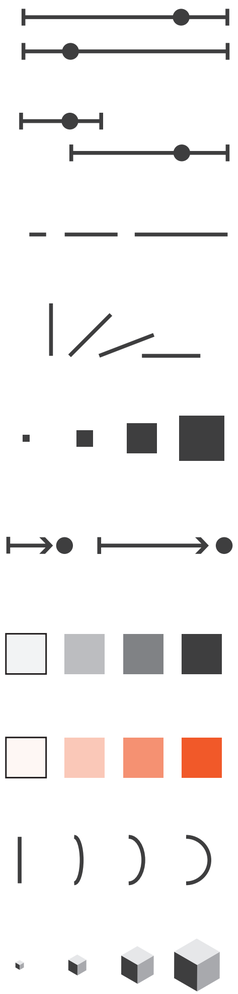
Depth (3D position)

Color luminance

Color saturation

Curvature

Volume (3D size)



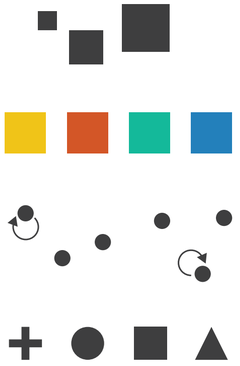
**Identity Channels: Categorical Attributes**

Spatial region

Color hue

Motion

Shape



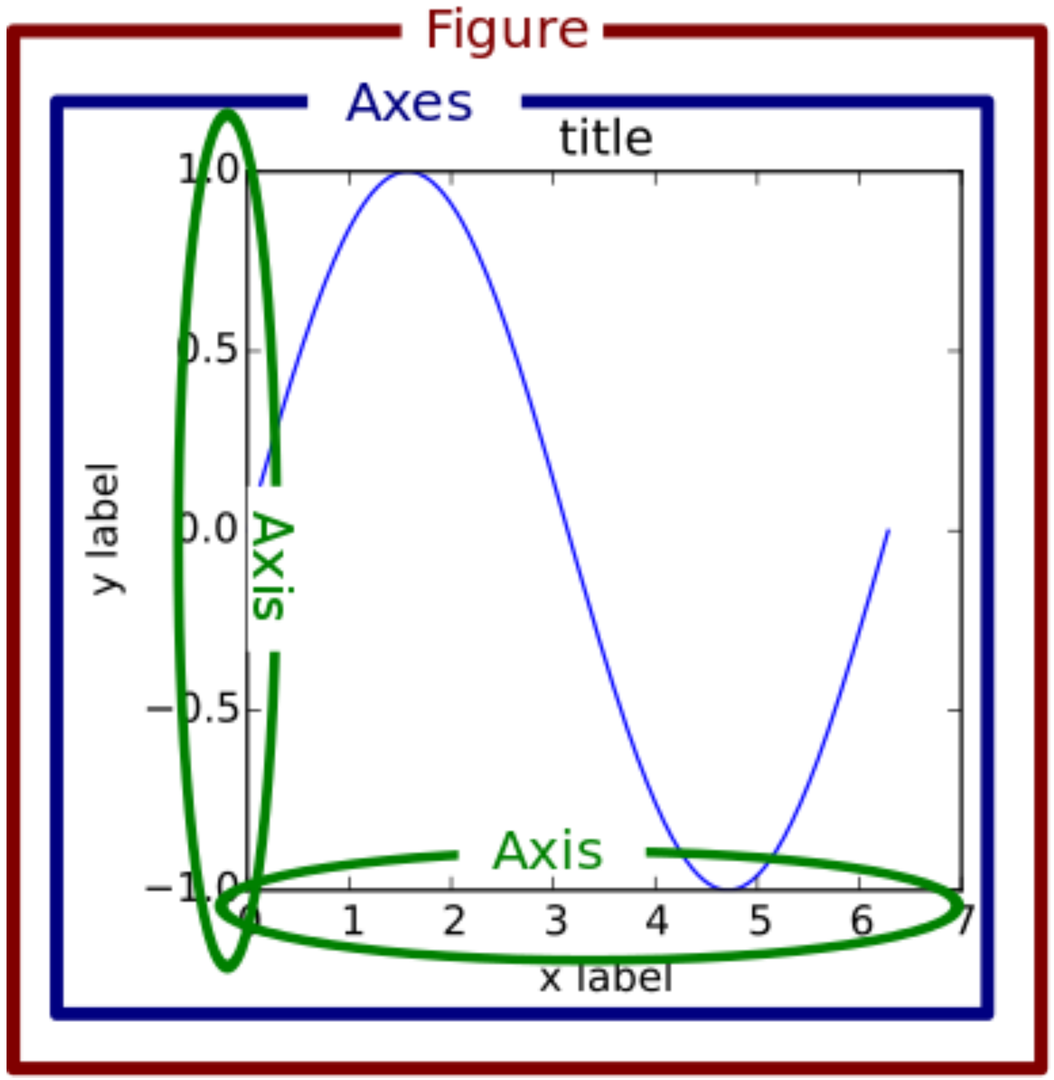
Altair will use its rules to pick whether to use color hue or saturation based on the type

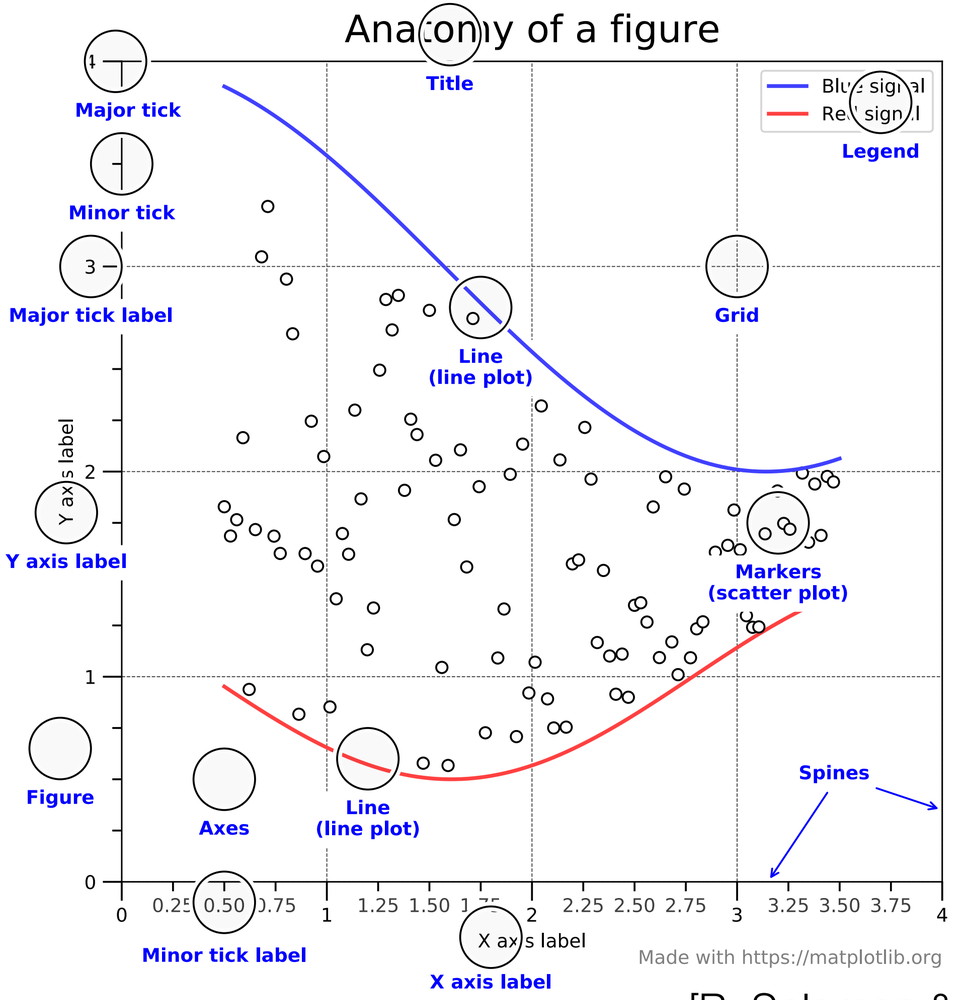
[Munzner (ill. Maguire), 2014]

# 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'])
  ```
  ```

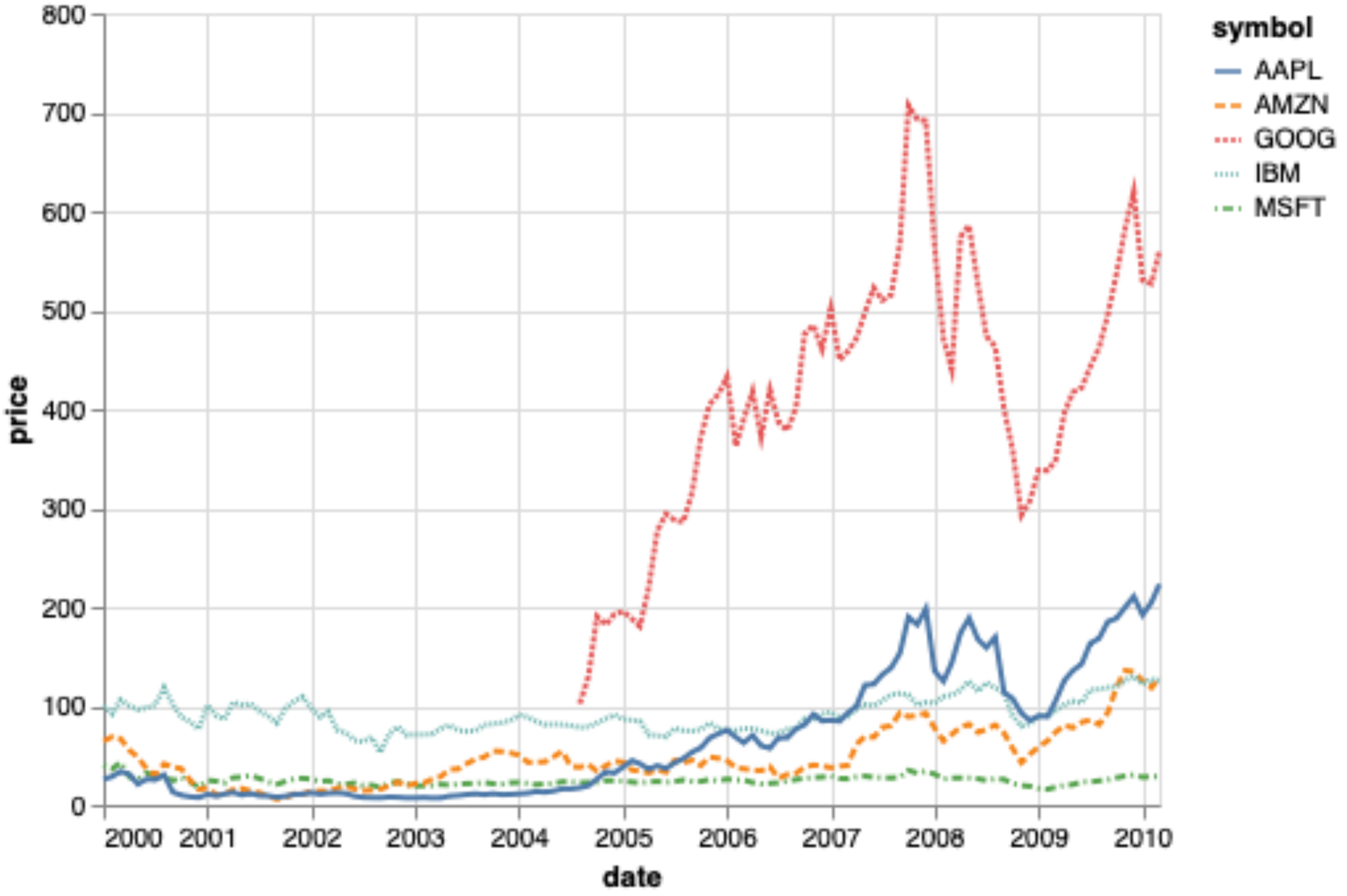
# pandas Integration

- 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

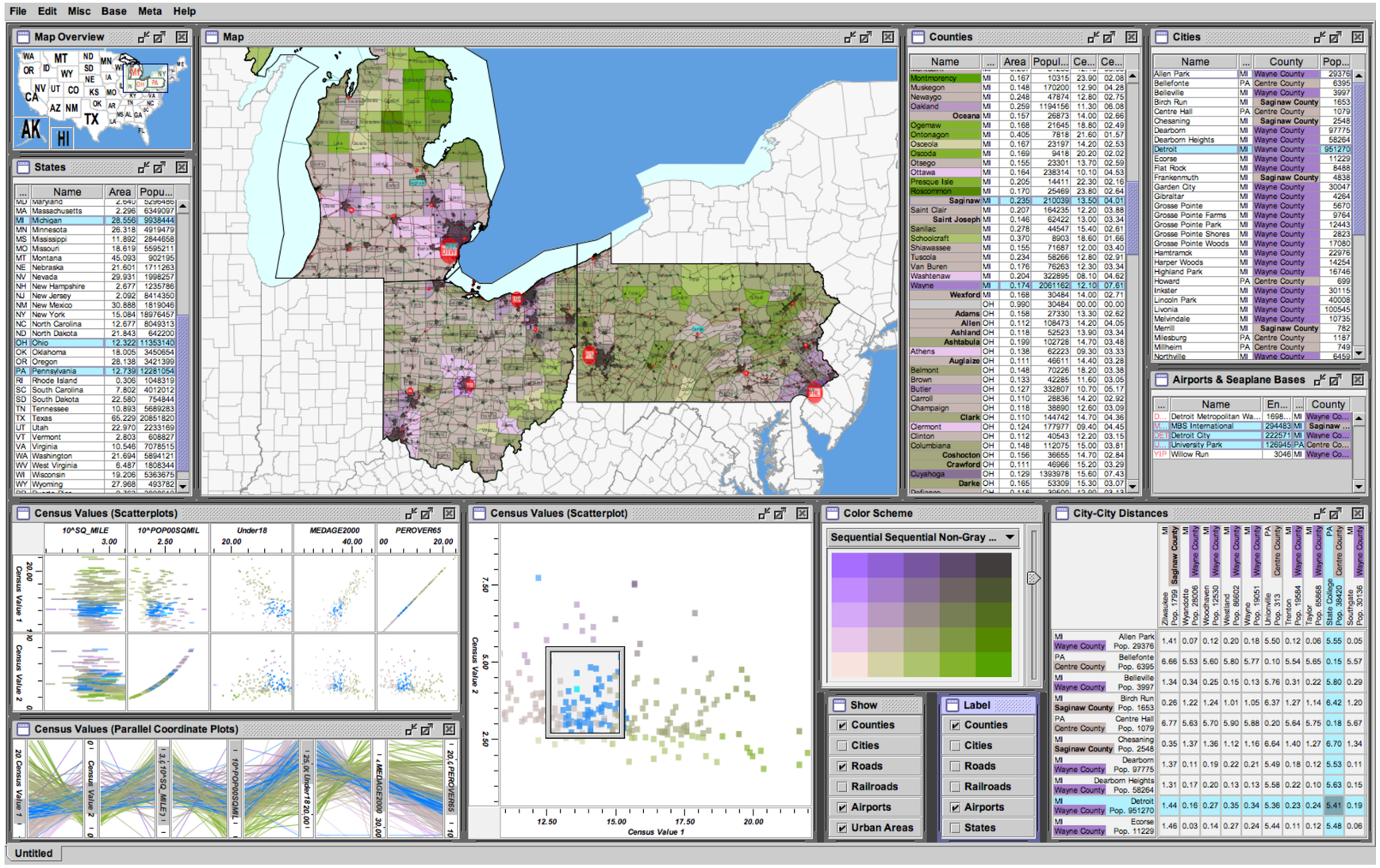
- Seaborn:
  - ```
    import seaborn as sns
    sns.scatterplot(x='price',y='count',hue='name', data=fruit)
    ```
    ```

# Multiple Views in Visualization



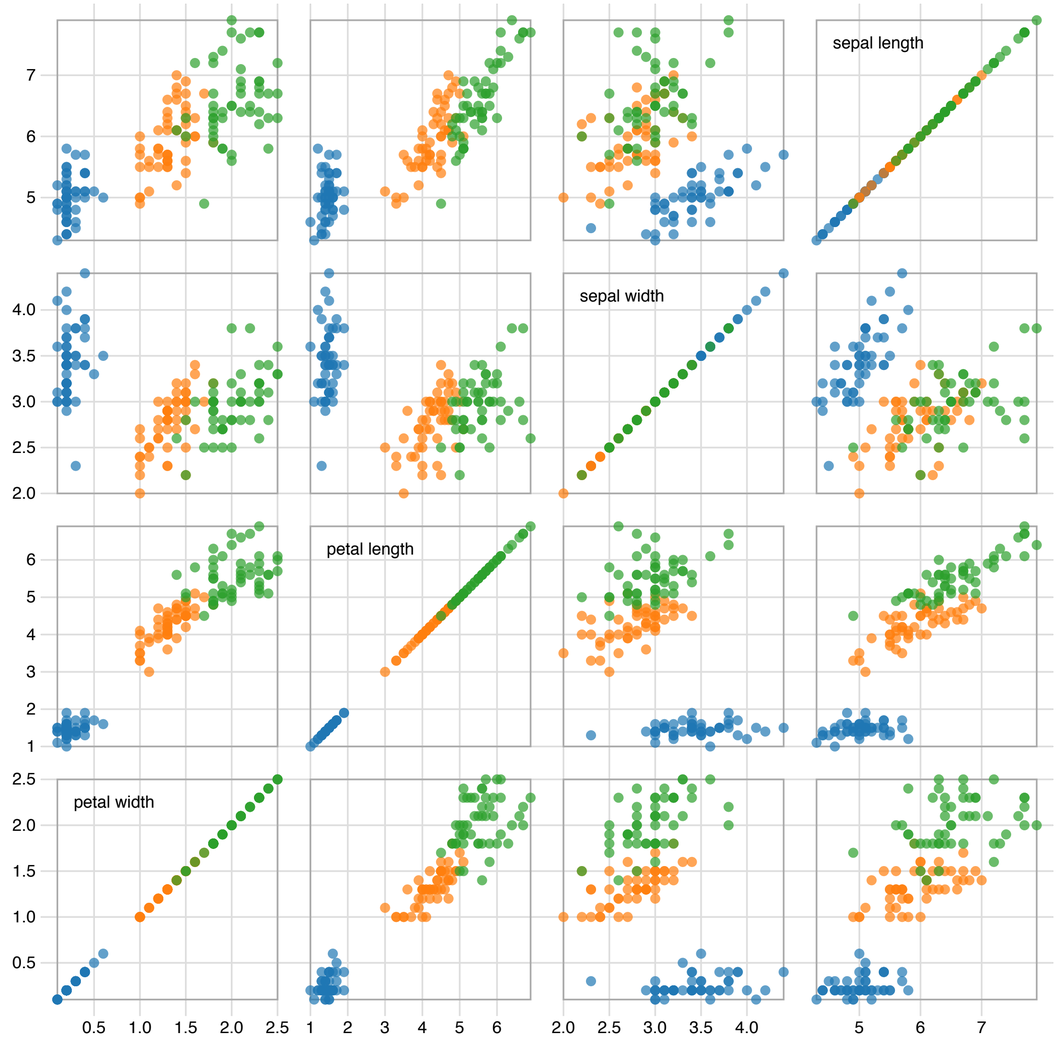

[Altair]

# Multiple Views in Visualization




[Improvise, Weaver, 2004]

# Multiple Views in Visualization

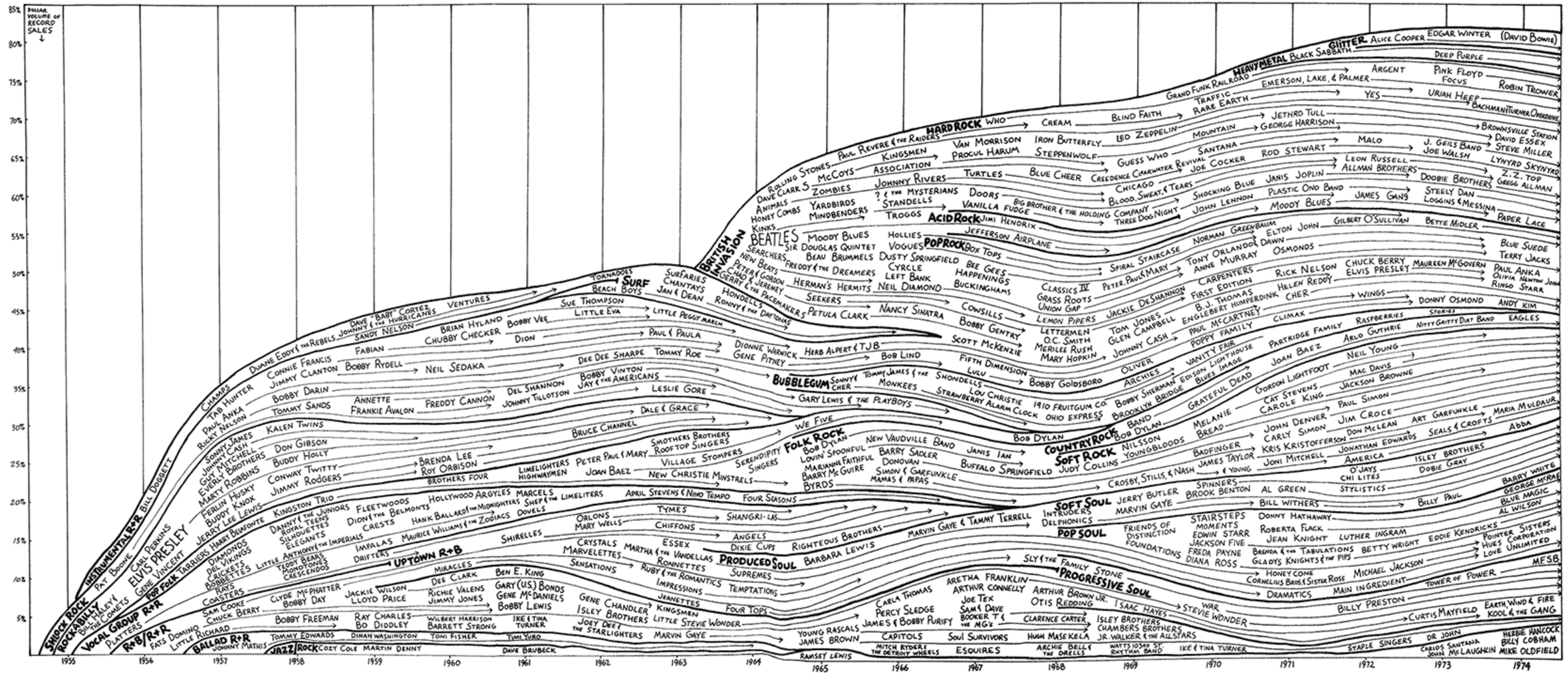



[M. Bostock]

# Altair Supports Concatenation, Layering, & Repetition

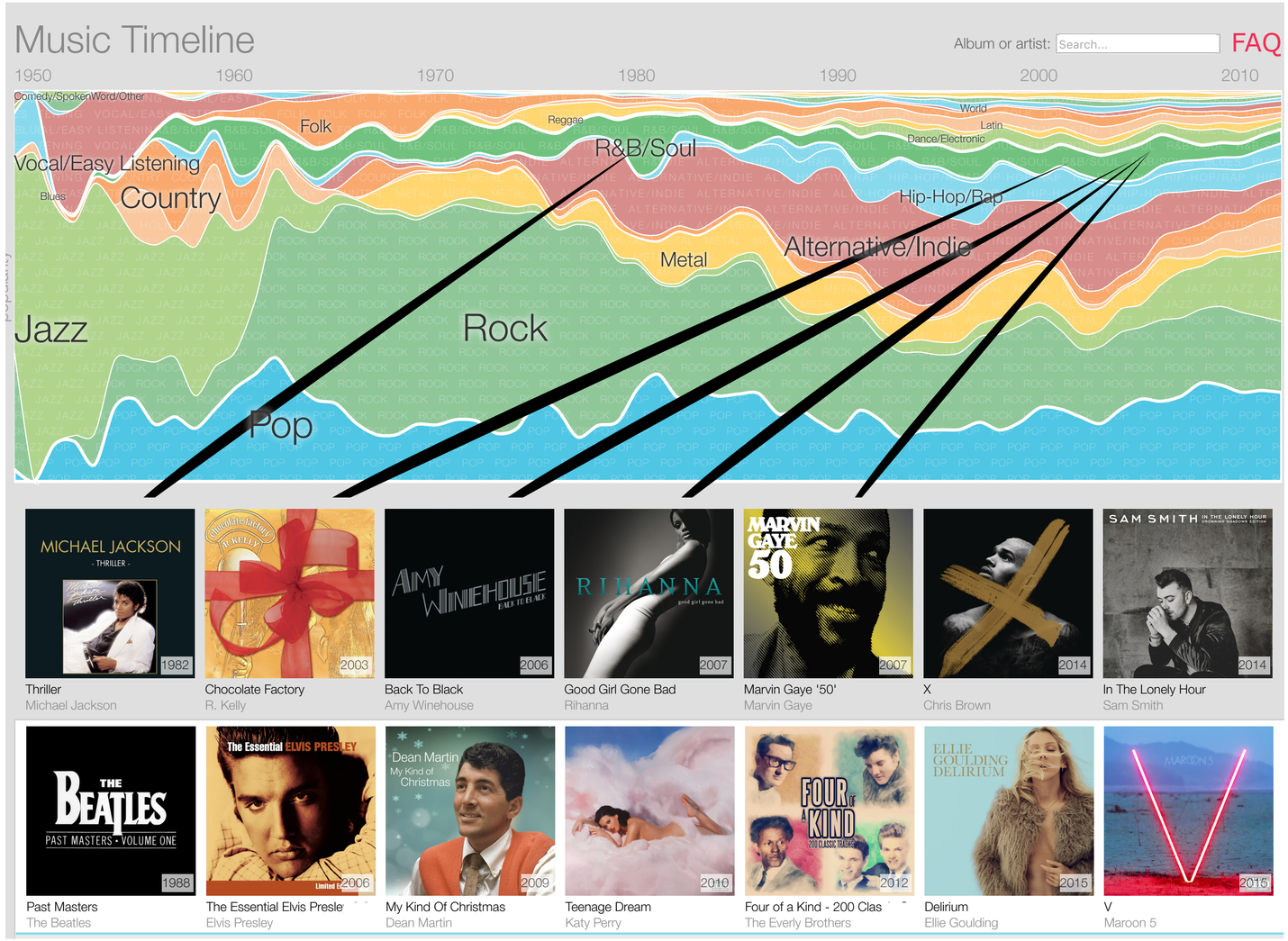
- Layering:
  - + Operator
- Concatenation:
  - Horizontal: | operator
  - Vertical: & operator
- Repetition
  - Use of .repeat for layout
  - Reference repeated variables in the encoding

# 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

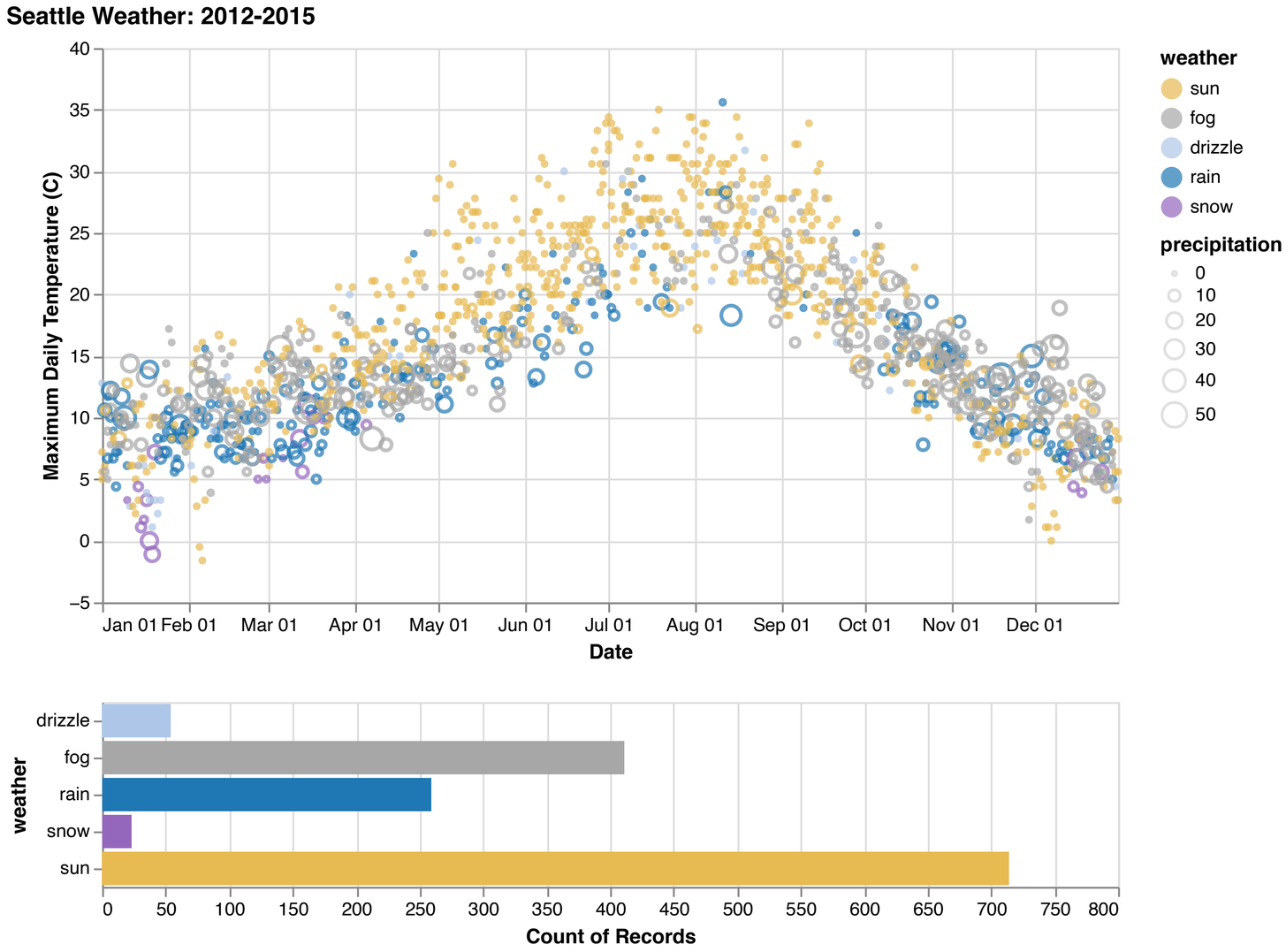
# Highlighting

- 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

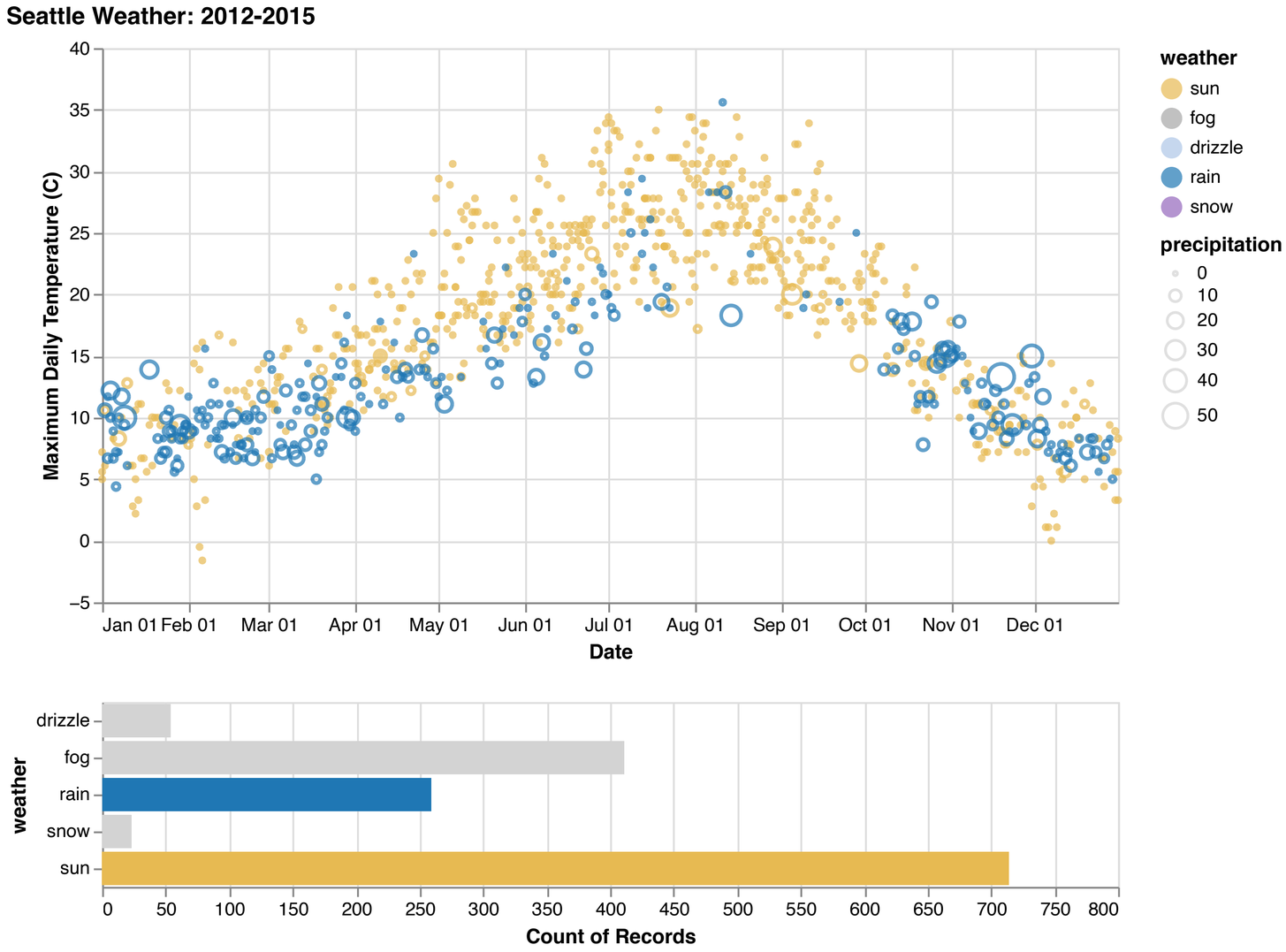
# Altair's Interactive Charts

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

# Interaction



# Weather Selection: Rain vs. Sun



# Date Selection: July-September Sun