

Programming Principles in Python (CSCI 503/490)

Visualization

Dr. David Koop

String Methods

- Can do many of the same methods used for single strings on entire columns
- Requires `.str` prefix before calling the method
 - `violations.value.str.strip().str.split(' - Comments:')`
- Also helps when extracting from a list
 - `comments.str[1]`

Support for Datetime

- Python has datetime library to support dates and times
- pandas has a Timestamp data type that functions somewhat similarly
- Pandas can convert timestamps
 - `pd.to_datetime`: versatile, can often guess format
- Like string methods, also a `.dt` accessor for datetime methods/properties
- With a timestamp, filtering based on datetimes becomes easier
 - `df[df['Inspection Date'] > '2021']`

Method chaining in pandas

- Tom Augspurger's [post](#)
- [Effective Pandas](#) book by Matt Harrison
- Functions written for chaining, and pipe allows custom functions
- ```
def read(fp):
 df = (pd.read_csv(fp)
 .rename(columns=str.lower)
 .drop('unnamed: 36', axis=1)
 .pipe(extract_city_name)
 .pipe(time_to_datetime, ['dep_time', 'arr_time',
 'crs_arr_time', 'crs_dep_time'])
 .assign(fl_date=lambda x: pd.to_datetime(x['fl_date']),
 dest=lambda x: pd.Categorical(x['dest']),
 origin=lambda x: pd.Categorical(x['origin']),
 tail_num=lambda x: pd.Categorical(x['tail_num']),
 unique_carrier=lambda x: pd.Categorical(x['unique_carrier']),
 cancellation_code=lambda x: pd.Categorical(x['cancellation_code'])))
 return df
```

# Assignment 8

---

- Last Assignment
- Data and Visualization
- Pokemon Data

# Data Exploration through Visualization



# Transportation Data - NYC MTA





# MTA Fare Data Exploration

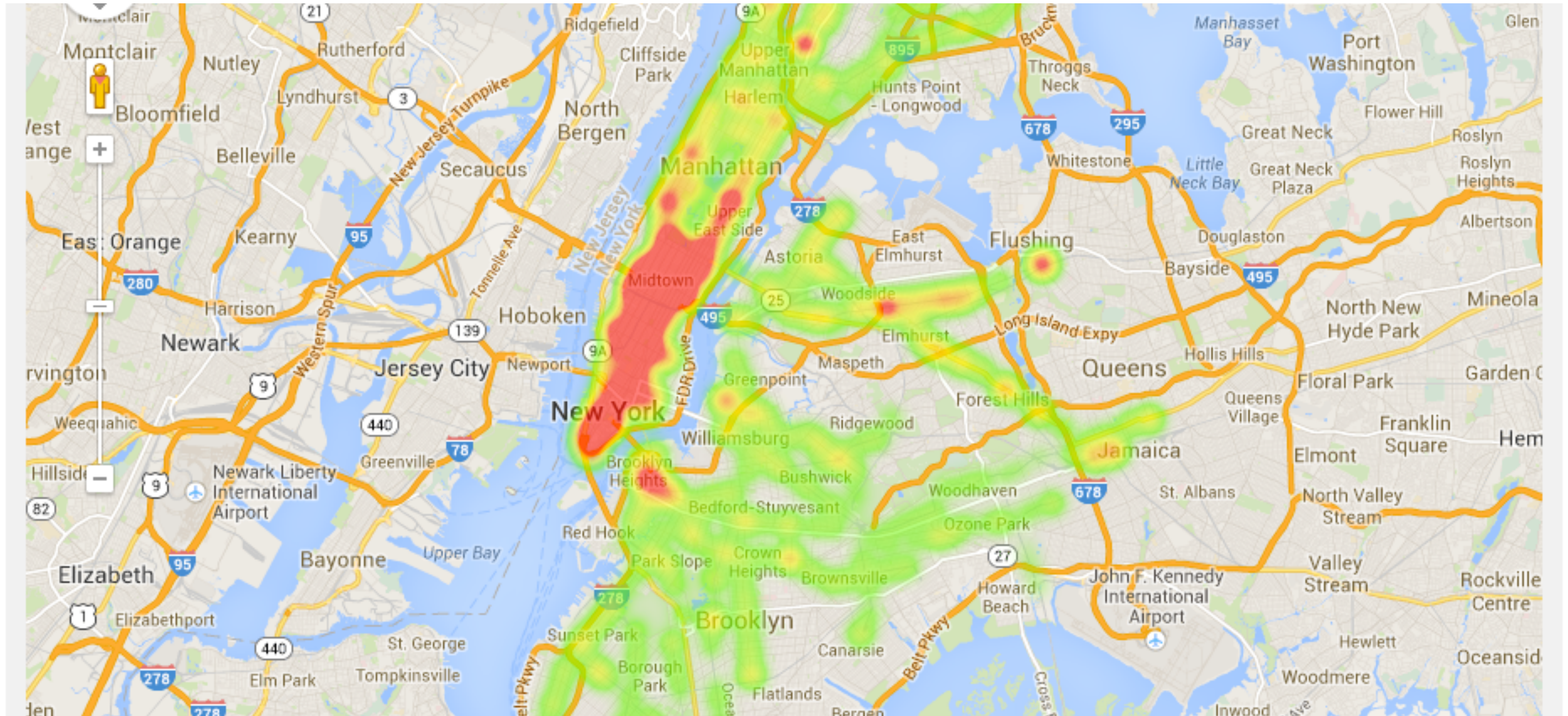
|    | REMOTE | STATION                     | FF ▼     | SEN/DIS  | 7-D AFAS UNL | D AFAS/RMF I | JOINT RR TKT | 7-D UNL  | 30-D UNL |
|----|--------|-----------------------------|----------|----------|--------------|--------------|--------------|----------|----------|
| 1  | R011   | 42ND STREET & 8TH AVENUE    | 00228985 | 00008471 | 00000441     | 00001455     | 00000134     | 00033341 | 00071255 |
| 2  | R170   | 14TH STREET-UNION SQUARE    | 00224603 | 00011051 | 00000827     | 00003026     | 00000660     | 00089367 | 00199841 |
| 3  | R046   | 42ND STREET & GRAND CENTRAL | 00207758 | 00007908 | 00000323     | 00001183     | 00003001     | 00040759 | 00096613 |
| 4  | R012   | 34TH STREET & 8TH AVENUE    | 00188311 | 00006490 | 00000498     | 00001279     | 00003622     | 00035527 | 00067483 |
| 5  | R293   | 34TH STREET - PENN STATION  | 00168768 | 00006155 | 00000523     | 00001065     | 00005031     | 00030645 | 00054376 |
| 6  | R033   | 42ND STREET/TIMES SQUARE    | 00159382 | 00005945 | 00000378     | 00001205     | 00000690     | 00058931 | 00078644 |
| 7  | R022   | 34TH STREET & 6TH AVENUE    | 00156008 | 00006276 | 00000487     | 00001543     | 00000712     | 00058910 | 00110466 |
| 8  | R084   | 59TH STREET/COLUMBUS CIRCLE | 00155262 | 00009484 | 00000589     | 00002071     | 00000542     | 00053397 | 00113966 |
| 9  | R020   | 47-50 STREETS/ROCKEFELLER   | 00143500 | 00006402 | 00000384     | 00001159     | 00000723     | 00037978 | 00090745 |
| 10 | R179   | 86TH STREET-LEXINGTON AVE   | 00142169 | 00010367 | 00000470     | 00001839     | 00000271     | 00050328 | 00125250 |
| 11 | R023   | 34TH STREET & 6TH AVENUE    | 00134052 | 00005005 | 00000348     | 00001112     | 00000649     | 00031531 | 00075040 |
| 12 | R029   | PARK PLACE                  | 00121614 | 00004311 | 00000287     | 00000931     | 00000792     | 00025404 | 00065362 |
| 13 | R047   | 42ND STREET & GRAND CENTRAL | 00100742 | 00004273 | 00000185     | 00000704     | 00001241     | 00022808 | 00068216 |



The map displays the distribution of 1000 points across the New York City metropolitan area. The points are represented by small circles, with a color gradient from white to red. The highest concentration of points is in the lower Manhattan area, particularly in the Midtown and Lower East Side regions. The points are also distributed across the other boroughs, including Queens, Brooklyn, and the Bronx, but with a lower density. The map includes labels for various cities, towns, and villages, as well as major highways and water bodies. The points are distributed across the entire map, with a higher density in the lower Manhattan area.

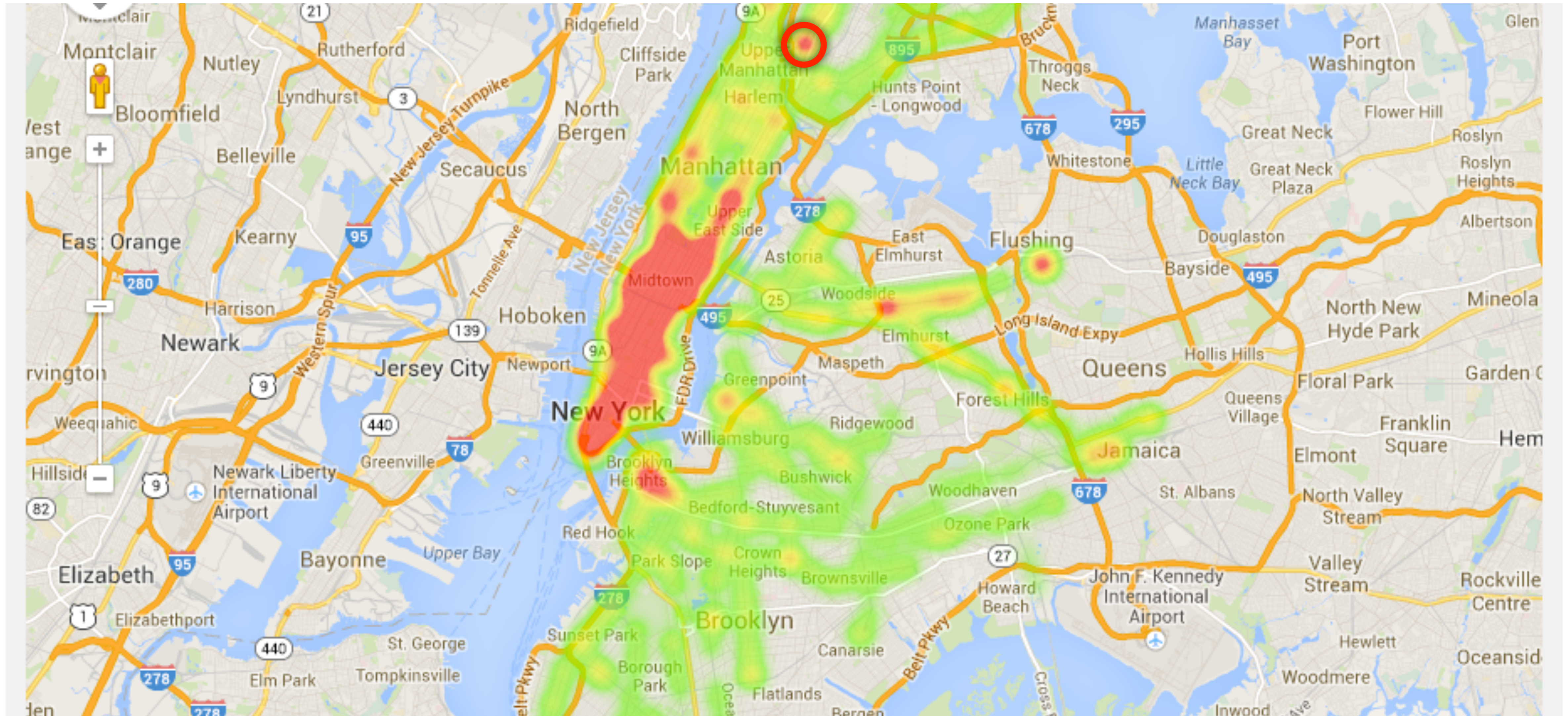


# MTA Fare Data Exploration





# MTA Fare Data Exploration



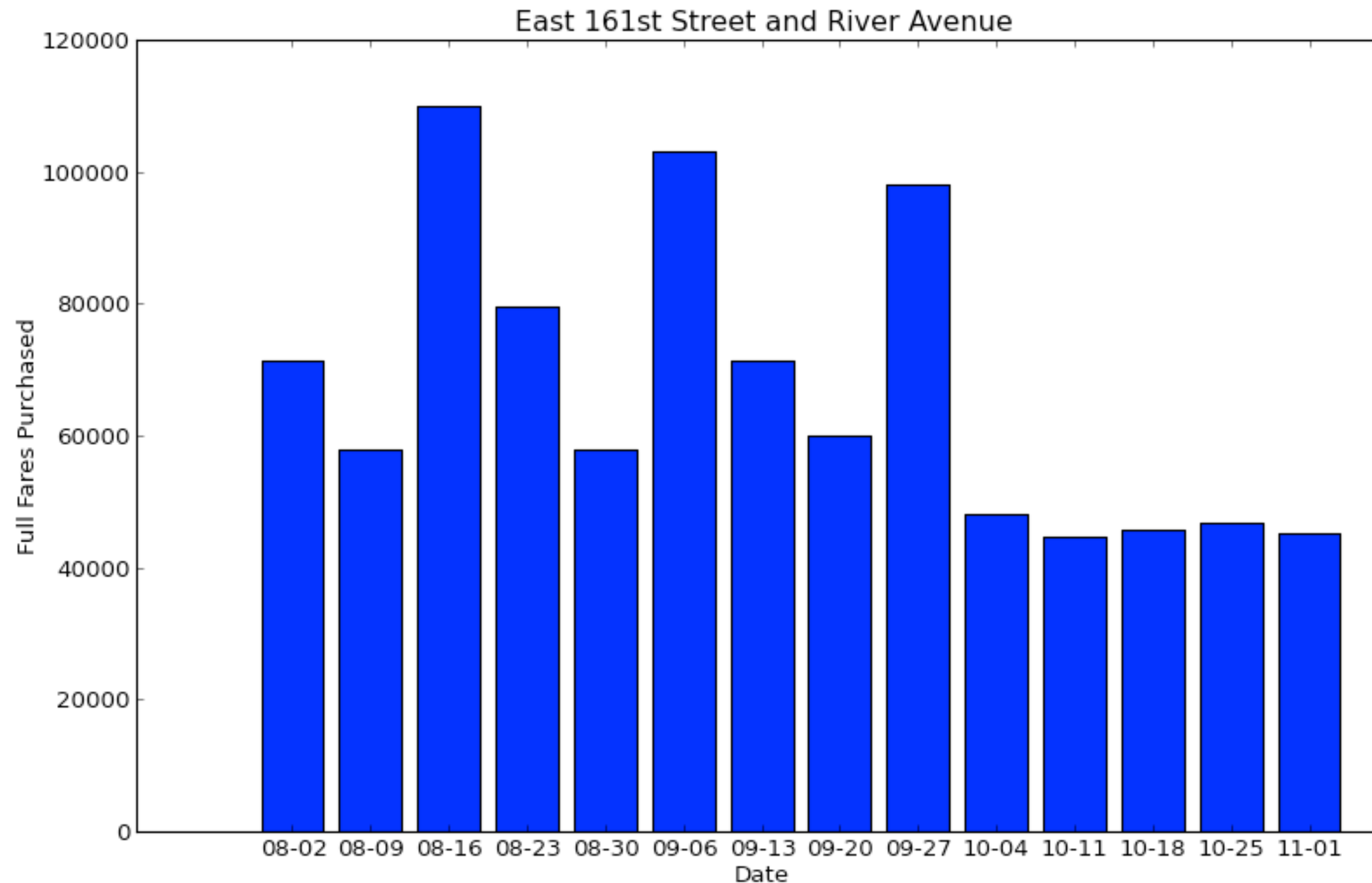


This map illustrates the distribution of parks and green spaces across New York City. Key features include:

- Parks and Green Spaces:** Numerous parks are labeled, including Central Park, Prospect Park, the Bronx Zoo, and the New York Botanical Garden. The density of green spaces is higher in the central and northern parts of the city.
- Major Highways:** Major highways are shown in orange, including I-95, I-87, I-278, and I-678.
- Water Bodies:** The Hudson River, Harlem River, and Eastchester Bay are labeled.
- Neighborhoods:** Various neighborhoods are labeled, including Teaneck, Englewood, Riverdale, Kingsbridge, Marble Hill, University Heights, West Bronx, Tremont, Parkchester, Soundview, Throgs Neck, and Little Bay.
- Infrastructure:** The map shows the New York City Subway system, with lines 4, 9, 9A, 9W, 46, 1, 9A, 8, 895, 278, 95, 695, 295, and 678.

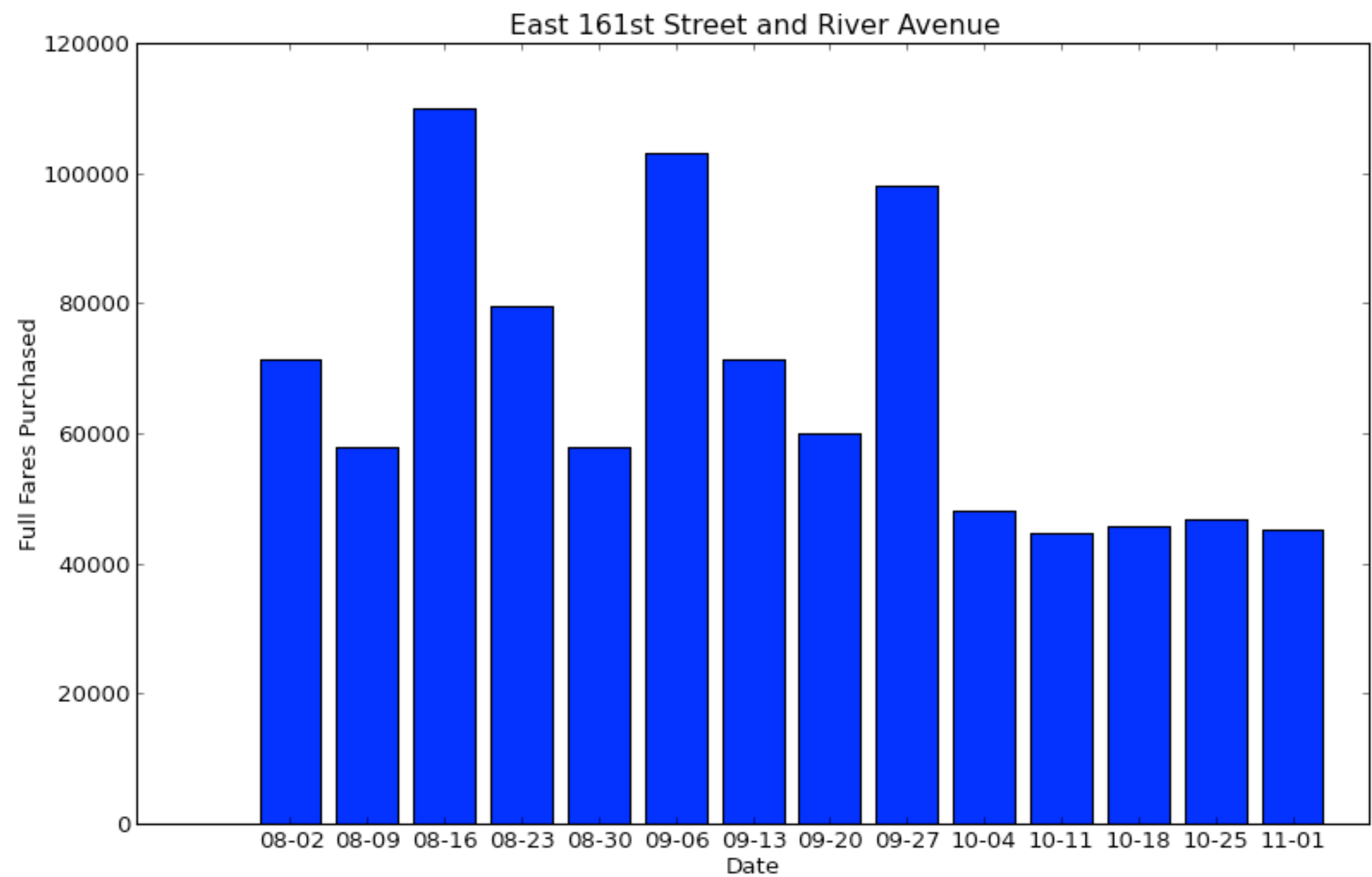


# MTA Fare Data Exploration





# MTA Fare Data Exploration



*New York Yankees*

AUGUST

| SUN                  | MON                   | TUE                   | WED                   | THU                   | FRI                   | SAT                   |
|----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| yankees.com          |                       |                       |                       | 1                     | 2<br>SD<br>10:10 YES  | 3<br>SD<br>8:40 YES   |
| 4<br>SD<br>4:10 YES  | 5<br>CHW<br>8:10 YES  | 6<br>CHW<br>8:10 MY9  | 7<br>CHW<br>8:10 YES  | 8                     | 9<br>DET<br>7:05 YES  | 10<br>DET<br>1:05 YES |
| 11<br>DET<br>TBA TBA | 12<br>LAA<br>7:05 YES | 13<br>LAA<br>7:05 YES | 14<br>LAA<br>7:05 YES | 15<br>LAA<br>1:05 YES | 16<br>BOS<br>7:10 MY9 | 17<br>BOS<br>4:05 FOX |
| 18<br>BOS<br>TBA TBA | 19                    | 20<br>TOR<br>7:05 MY9 | 21<br>TOR<br>7:05 YES | 22<br>TOR<br>1:05 YES | 23<br>TB<br>7:10 MY9  | 24<br>TB<br>7:10 YES  |
| 25<br>TB<br>1:40 YES | 26<br>TOR<br>7:07 YES | 27<br>TOR<br>7:07 YES | 28<br>TOR<br>7:07 YES | 29                    | 30<br>BAL<br>7:05 YES | 31<br>BAL<br>1:05 YES |

SEPTEMBER

| SUN                   | MON                  | TUE                         | WED                   | THU                   | FRI                   | SAT                   |
|-----------------------|----------------------|-----------------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 1<br>BAL<br>1:05 YES  | 2<br>CHW<br>1:05 YES | 3<br>CHW<br>7:05 YES        | 4<br>CHW<br>7:05 YES  | 5<br>BOS<br>7:05 YES  | 6<br>BOS<br>7:05 YES  | 7<br>BOS<br>1:05 FOX  |
| 8<br>BOS<br>TBA TBA   | 9<br>BAL<br>7:05 YES | 10<br>BAL<br>7:05 MY9       | 11<br>BAL<br>7:05 YES | 12<br>BAL<br>7:05 YES | 13<br>BOS<br>7:10 MY9 | 14<br>BOS<br>1:05 FOX |
| 15<br>BOS<br>TBA TBA  | 16                   | 17<br>TOR<br>7:07 MY9       | 18<br>TOR<br>7:07 YES | 19<br>TOR<br>7:07 YES | 20<br>SF<br>7:05 YES  | 21<br>SF<br>TBA TBA   |
| 22<br>SF<br>1:05 YES  | 23                   | 24<br>TB<br>7:05 MY9        | 25<br>TB<br>7:05 YES  | 26<br>TB<br>7:05 YES  | 27<br>HOU<br>8:10 YES | 28<br>HOU<br>TBA TBA  |
| 29<br>HOU<br>2:10 YES | 30                   | ALL GAMES ARE EASTERN TIME. |                       |                       |                       |                       |

• 2013 REGULAR SEASON SCHEDULE •



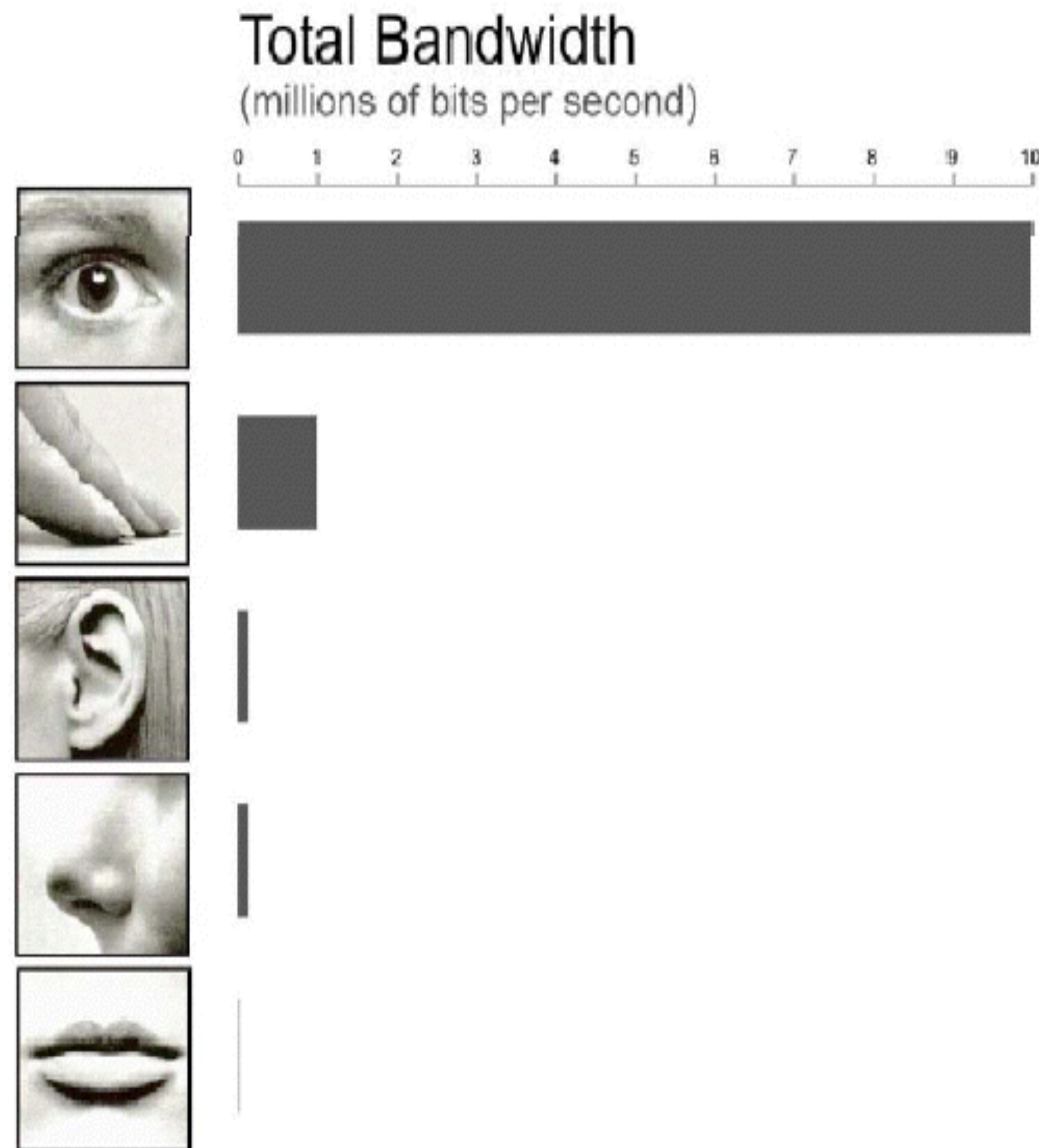
# Definition of Visualization

---

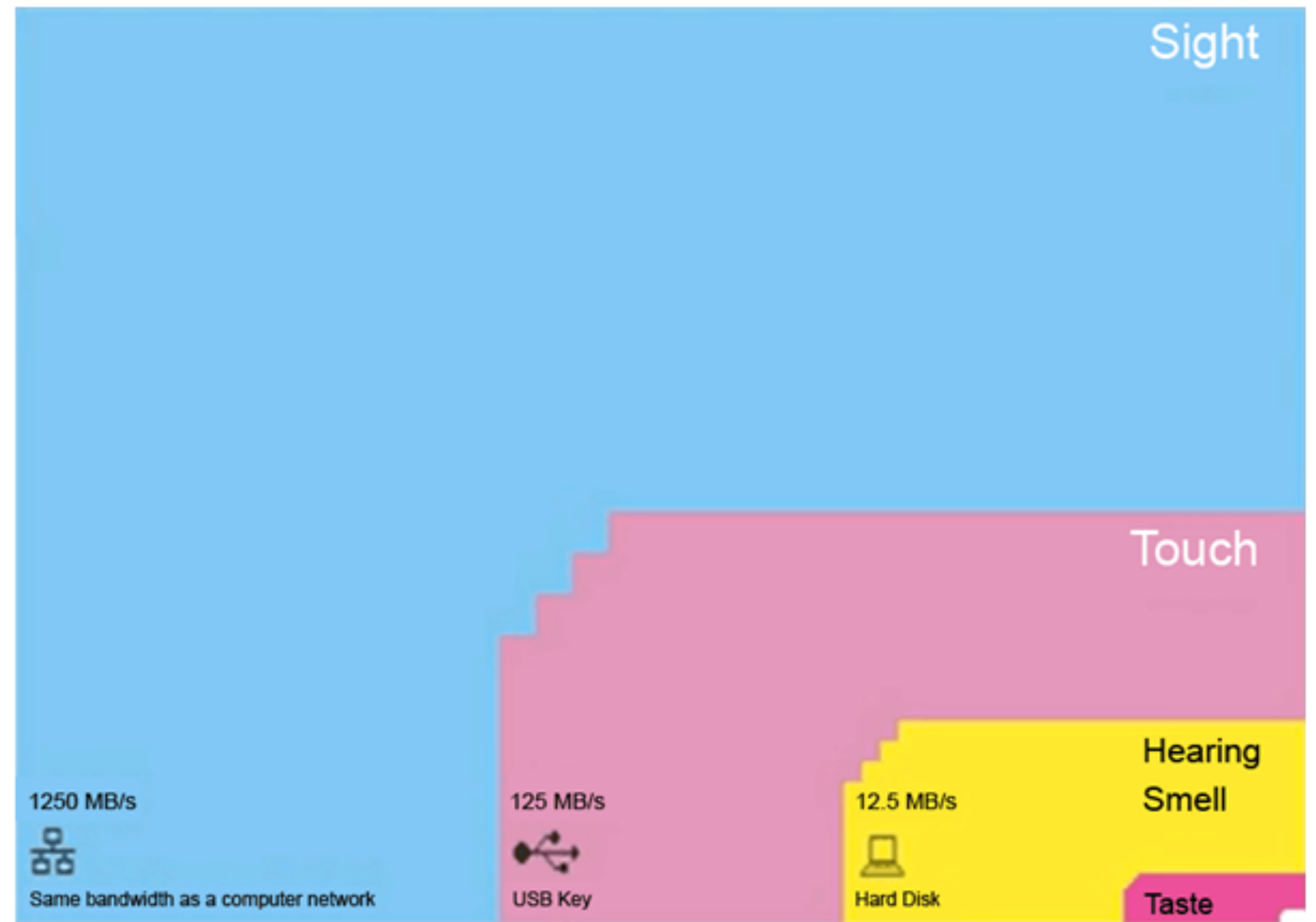
“Computer-based visualization systems provide visual representations of datasets designed to help people carry out tasks more effectively” — T. Munzner



# Why do we visualize data?



[via A. Lex]



[T. Nørretranders]



# Why Visual?

| I    |       | II   |      | III  |       | IV   |       |
|------|-------|------|------|------|-------|------|-------|
| x    | y     | x    | y    | x    | y     | x    | y     |
| 10.0 | 8.04  | 10.0 | 9.14 | 10.0 | 7.46  | 8.0  | 6.58  |
| 8.0  | 6.95  | 8.0  | 8.14 | 8.0  | 6.77  | 8.0  | 5.76  |
| 13.0 | 7.58  | 13.0 | 8.74 | 13.0 | 12.74 | 8.0  | 7.71  |
| 9.0  | 8.81  | 9.0  | 8.77 | 9.0  | 7.11  | 8.0  | 8.84  |
| 11.0 | 8.33  | 11.0 | 9.26 | 11.0 | 7.81  | 8.0  | 8.47  |
| 14.0 | 9.96  | 14.0 | 8.10 | 14.0 | 8.84  | 8.0  | 7.04  |
| 6.0  | 7.24  | 6.0  | 6.13 | 6.0  | 6.08  | 8.0  | 5.25  |
| 4.0  | 4.26  | 4.0  | 3.10 | 4.0  | 5.39  | 19.0 | 12.50 |
| 12.0 | 10.84 | 12.0 | 9.13 | 12.0 | 8.15  | 8.0  | 5.56  |
| 7.0  | 4.82  | 7.0  | 7.26 | 7.0  | 6.42  | 8.0  | 7.91  |
| 5.0  | 5.68  | 5.0  | 4.74 | 5.0  | 5.73  | 8.0  | 6.89  |

[F. J. Anscombe]



# Why Visual?

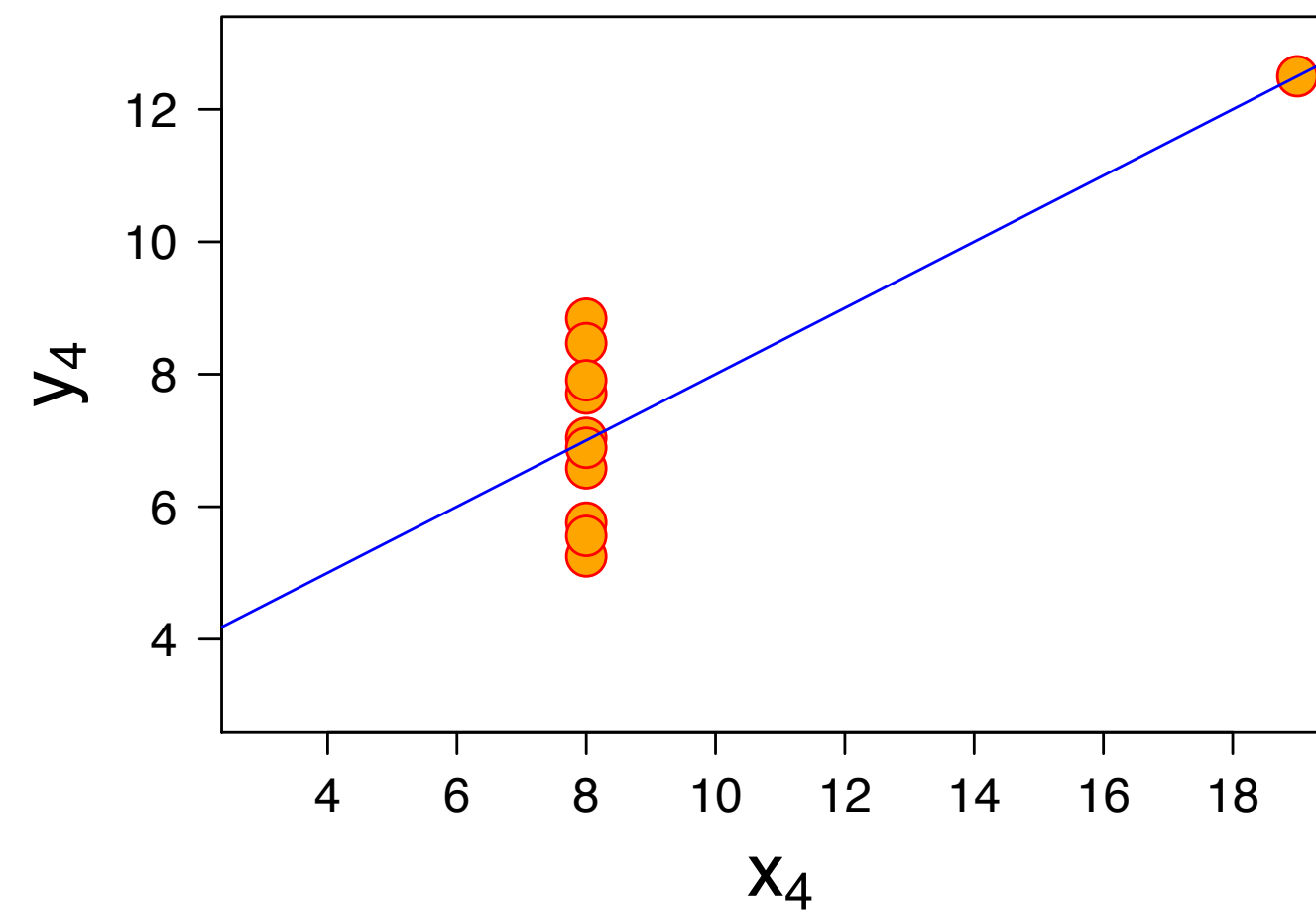
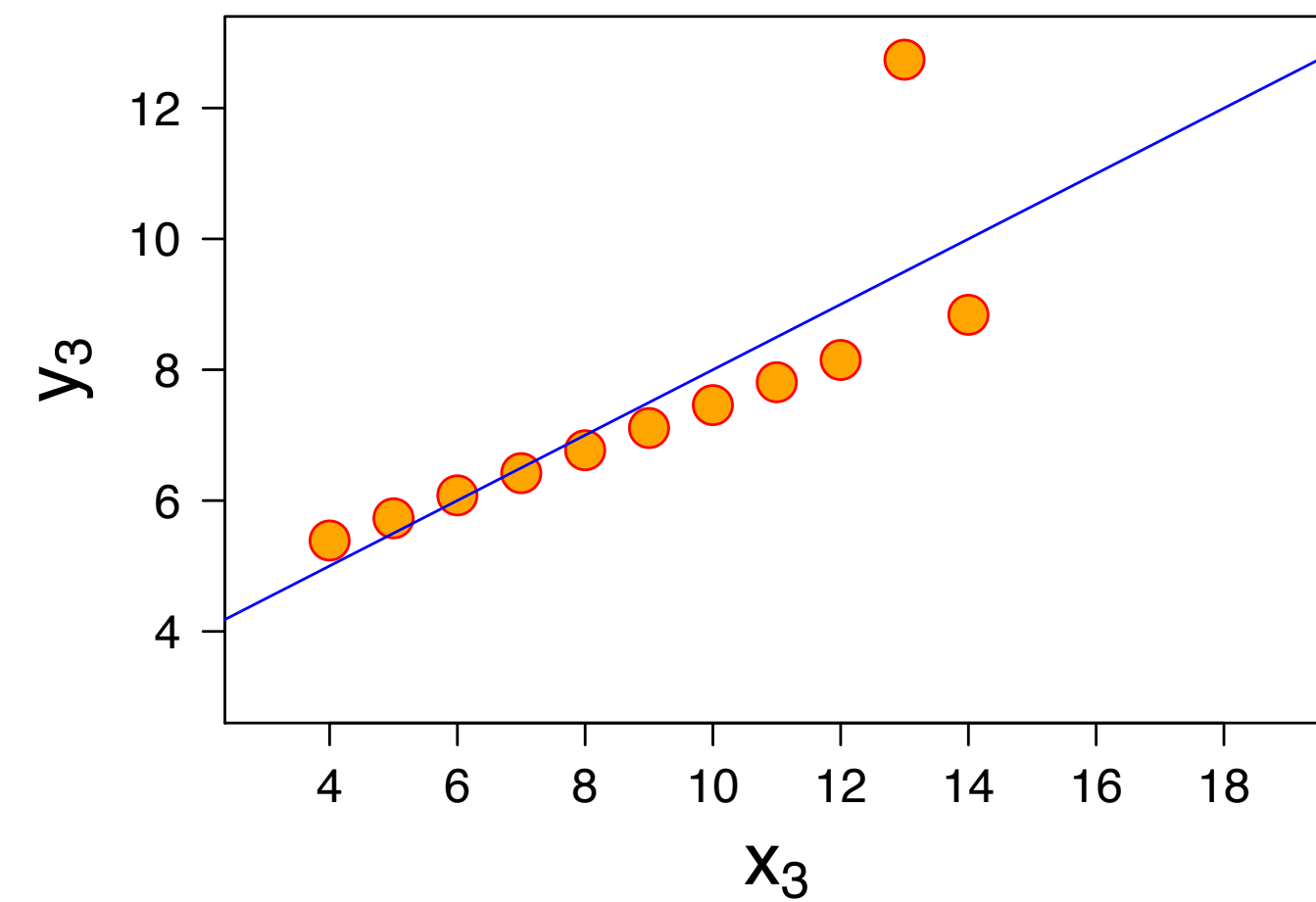
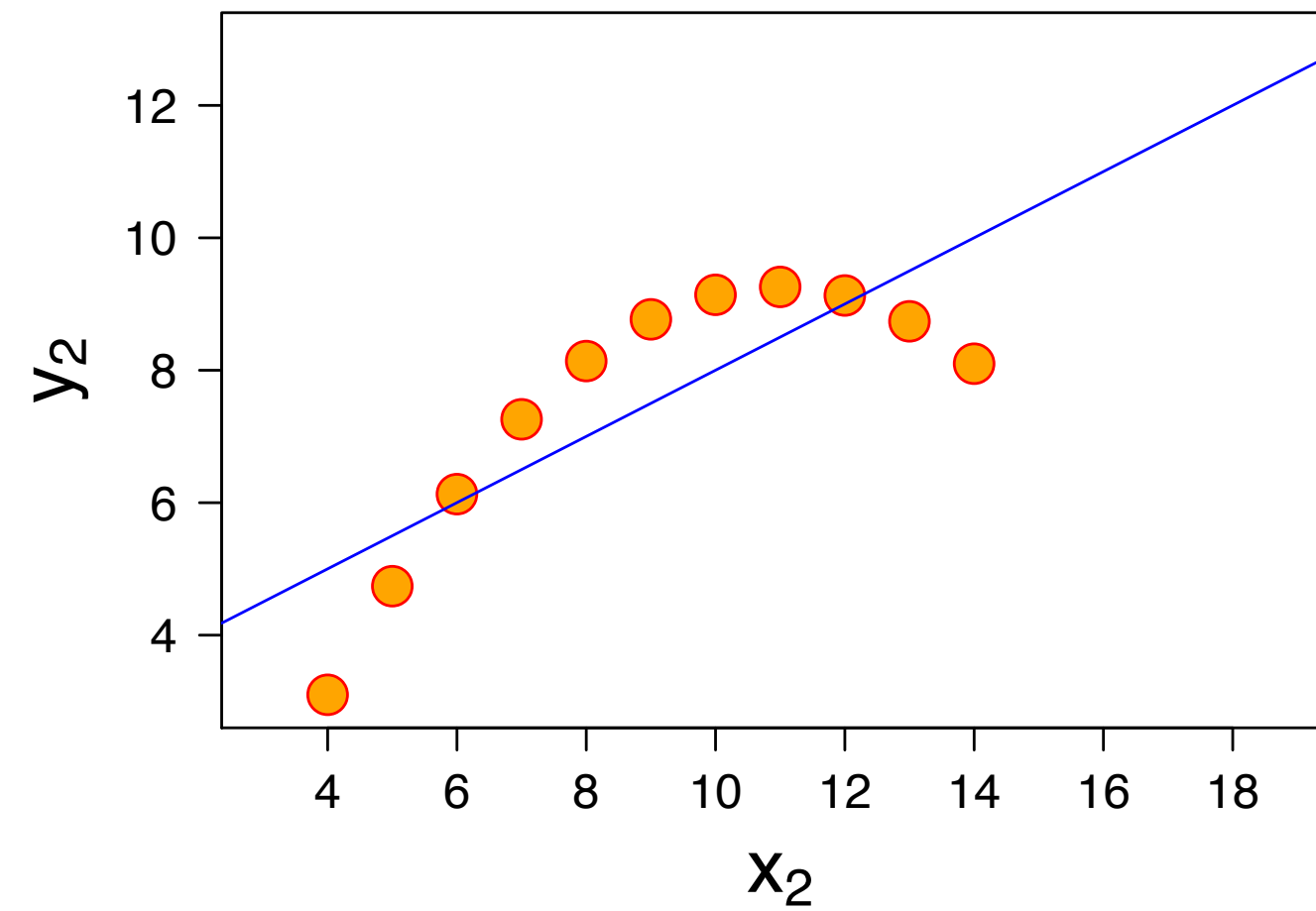
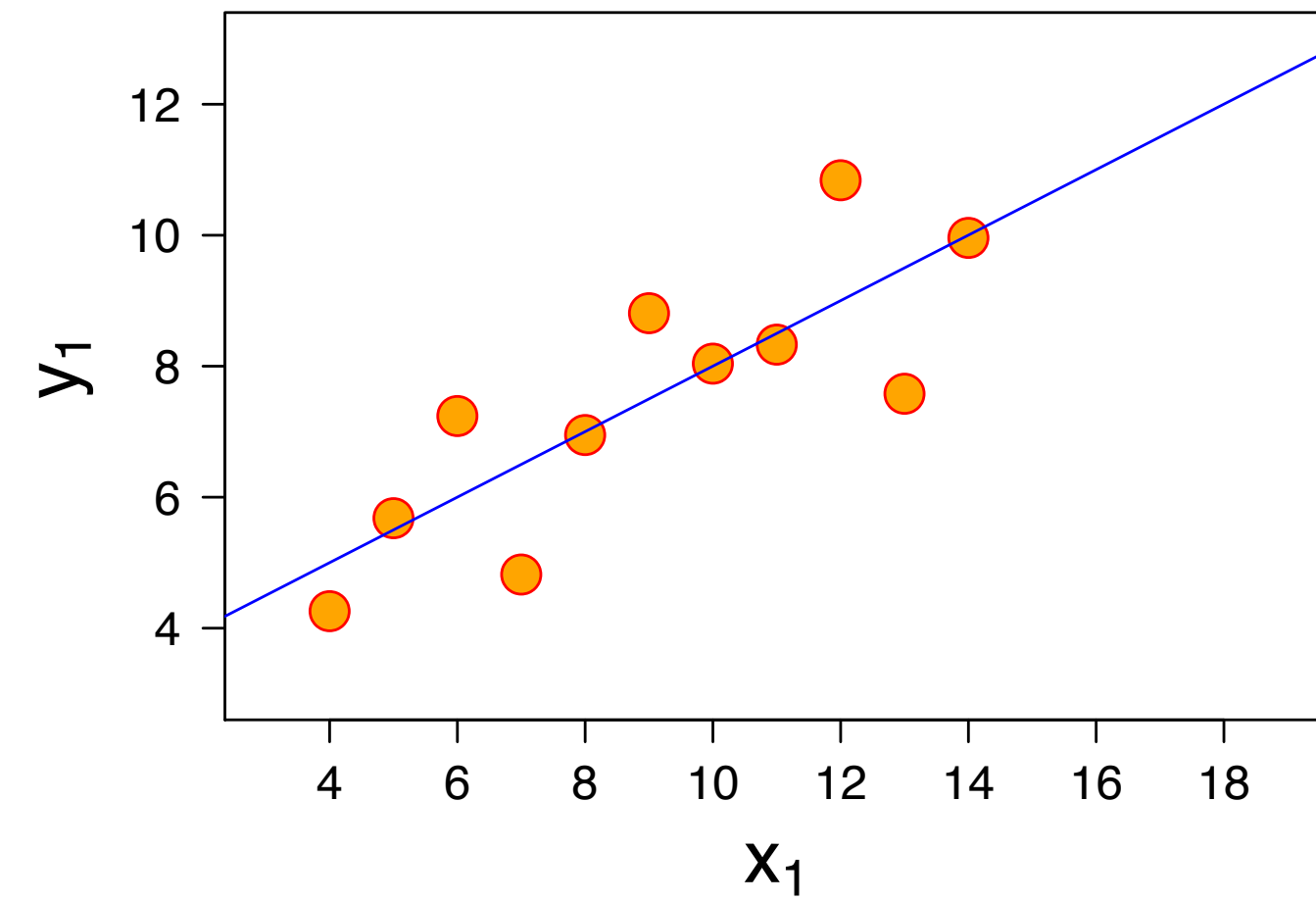
| I    |       | II   |      | III  |       | IV   |       |
|------|-------|------|------|------|-------|------|-------|
| x    | y     | x    | y    | x    | y     | x    | y     |
| 10.0 | 8.04  | 10.0 | 9.14 | 10.0 | 7.46  | 8.0  | 6.58  |
| 8.0  | 6.95  | 8.0  | 8.14 | 8.0  | 6.77  | 8.0  | 5.76  |
| 13.0 | 7.58  | 13.0 | 8.74 | 13.0 | 12.74 | 8.0  | 7.71  |
| 9.0  | 8.81  | 9.0  | 8.77 | 9.0  | 7.11  | 8.0  | 8.84  |
| 11.0 | 8.33  | 11.0 | 9.26 | 11.0 | 7.81  | 8.0  | 8.47  |
| 14.0 | 9.96  | 14.0 | 8.10 | 14.0 | 8.84  | 8.0  | 7.04  |
| 6.0  | 7.24  | 6.0  | 6.13 | 6.0  | 6.08  | 8.0  | 5.25  |
| 4.0  | 4.26  | 4.0  | 3.10 | 4.0  | 5.39  | 19.0 | 12.50 |
| 12.0 | 10.84 | 12.0 | 9.13 | 12.0 | 8.15  | 8.0  | 5.56  |
| 7.0  | 4.82  | 7.0  | 7.26 | 7.0  | 6.42  | 8.0  | 7.91  |
| 5.0  | 5.68  | 5.0  | 4.74 | 5.0  | 5.73  | 8.0  | 6.89  |

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

[F. J. Anscombe]



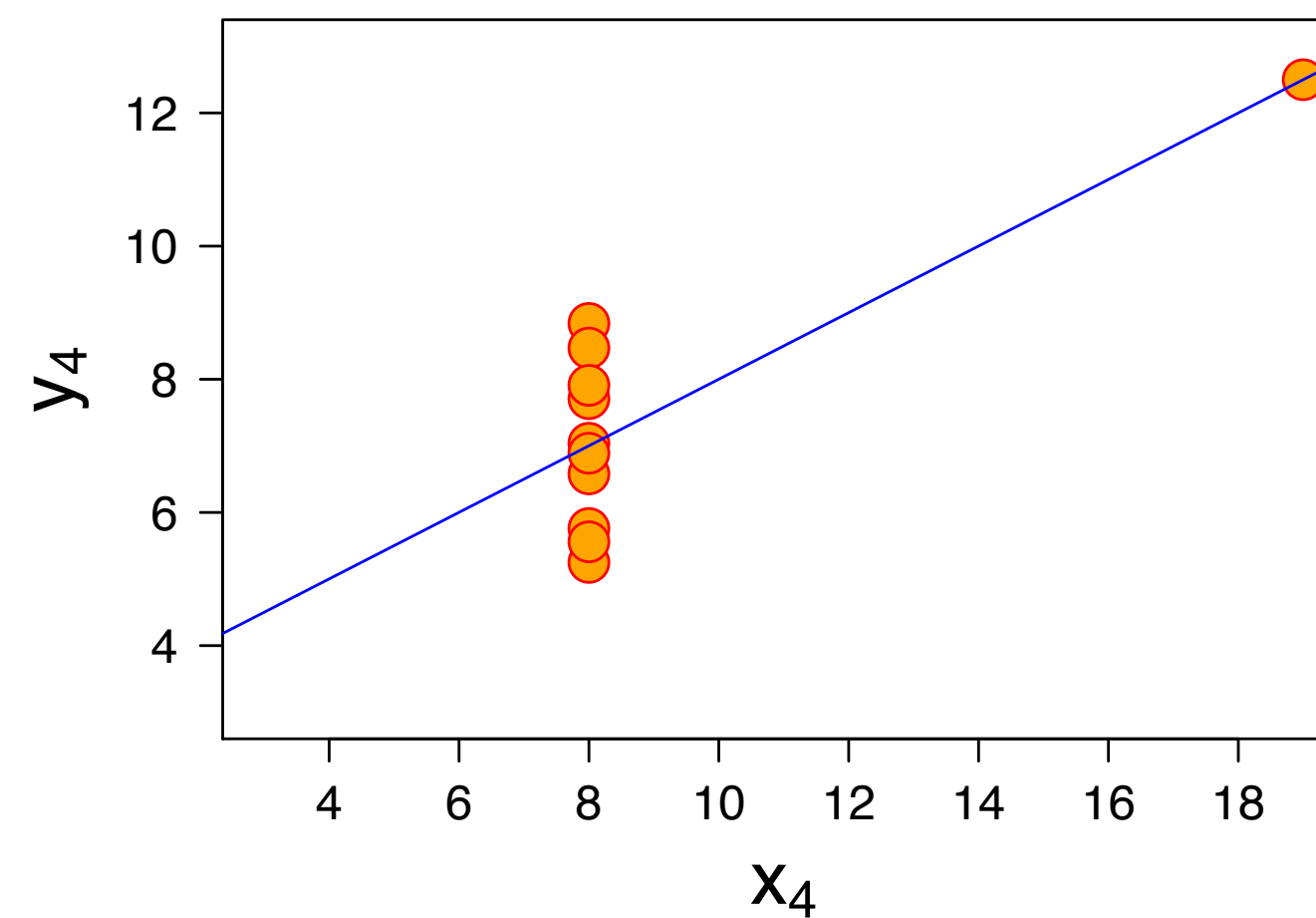
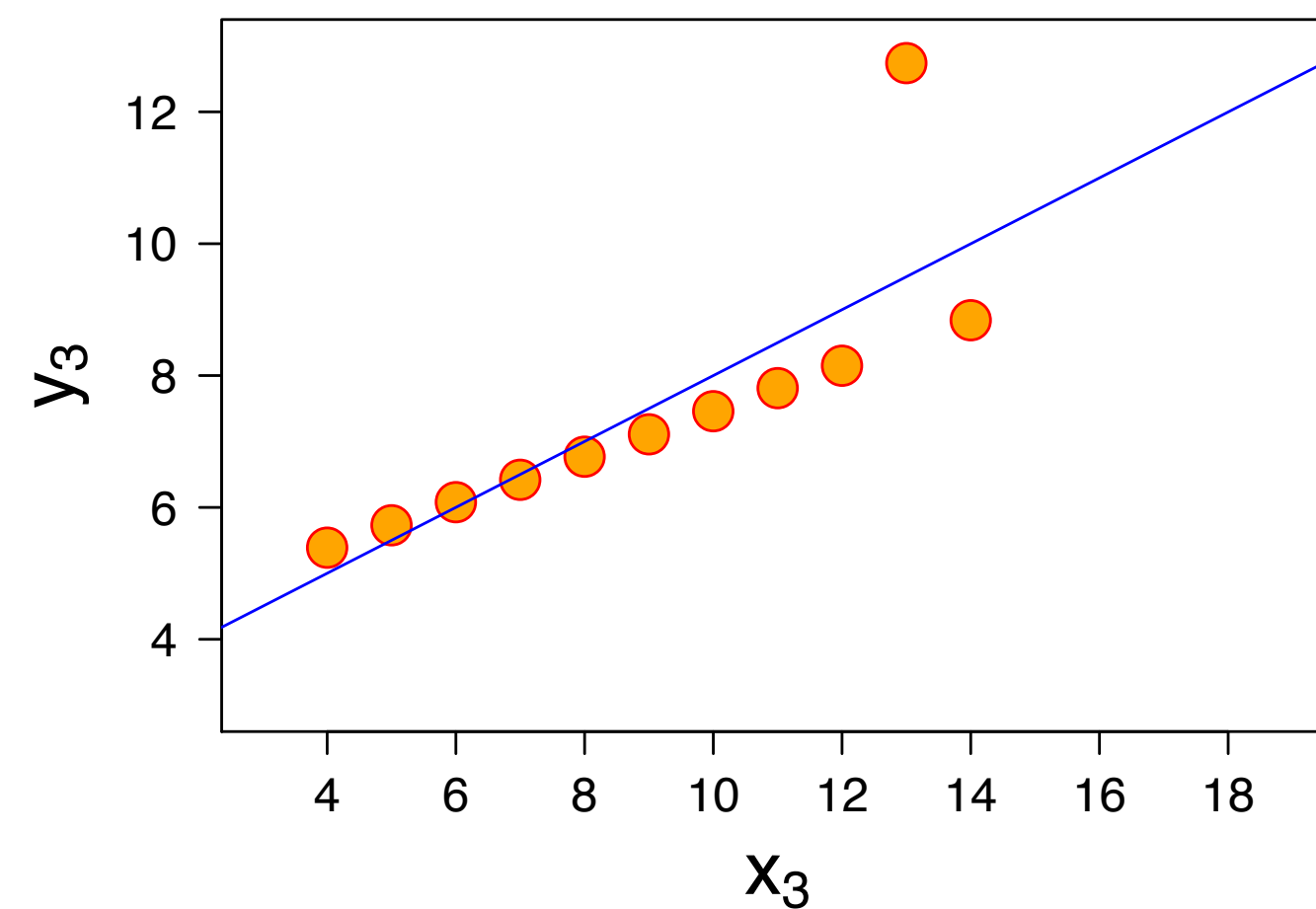
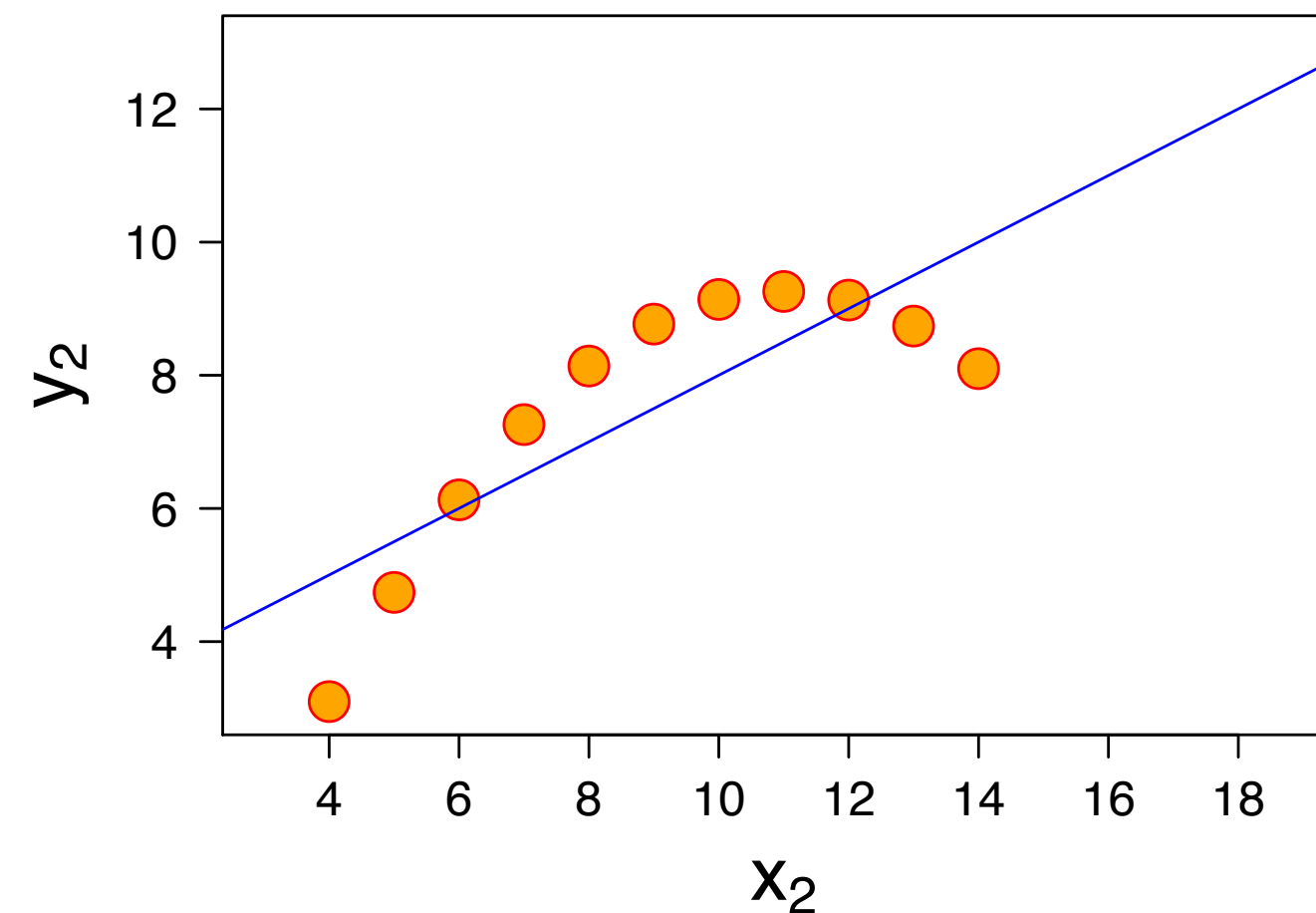
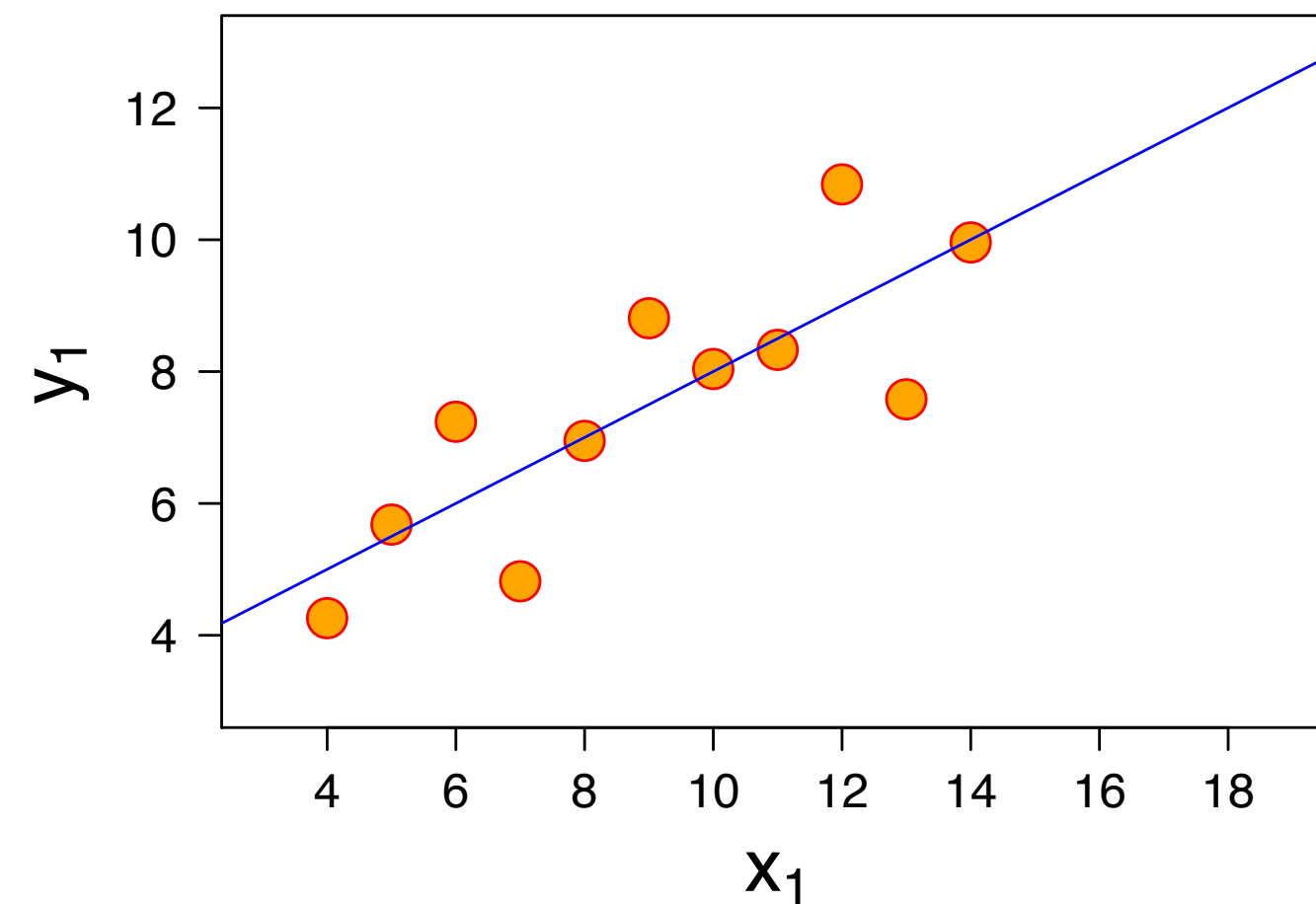
# Why Visual?



[F. J. Anscombe]



# Why Visual?



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

[F. J. Anscombe]



# Visualization Goals

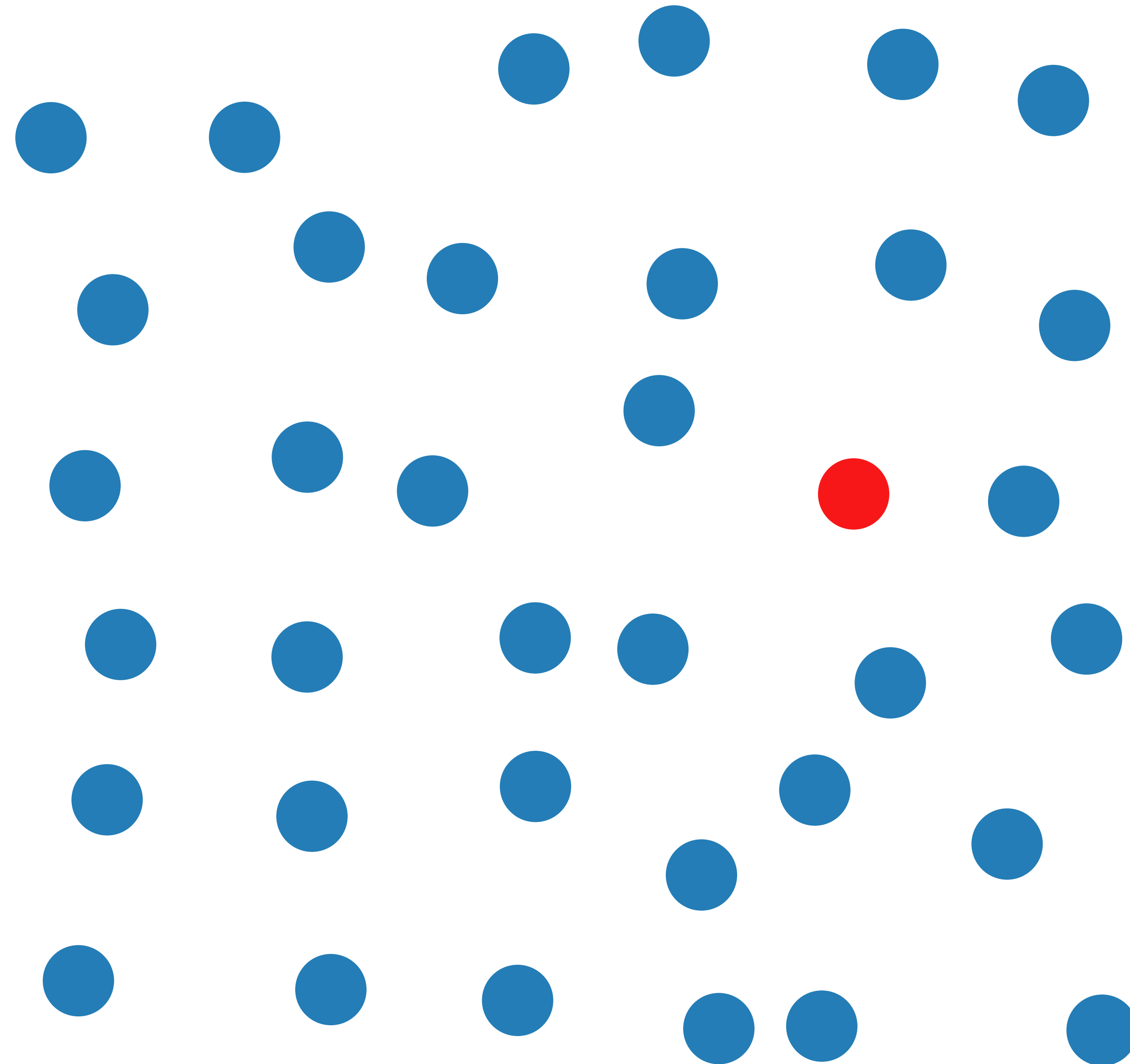
---

- "The purpose of visualization is **insight**, not pictures" – B. Shneiderman
- Identify patterns, trends
- Spot outliers
- Find similarities, correlation



# Visual Pop-out

---

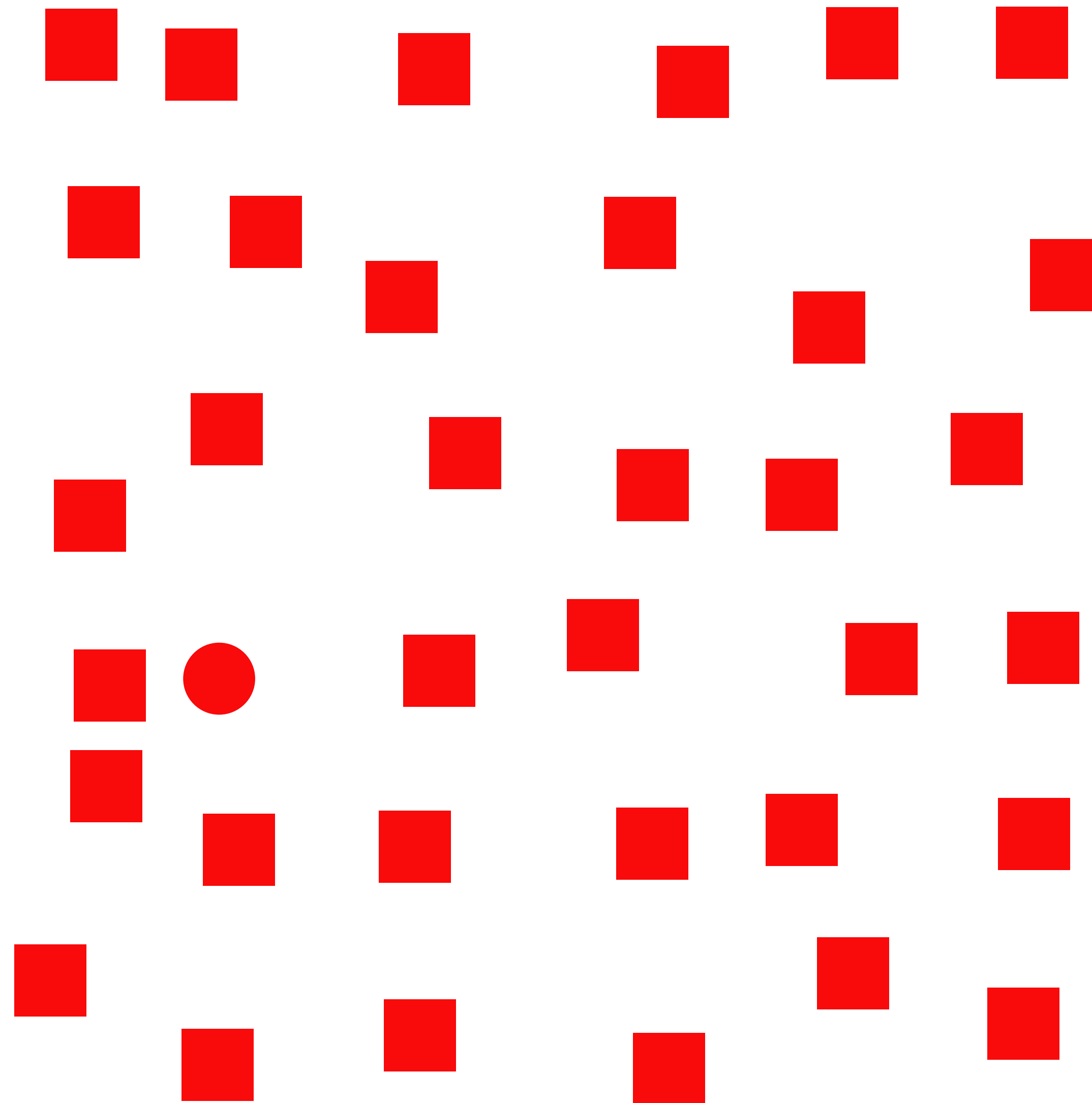


[C. G. Healey]



# Visual Pop-out

---

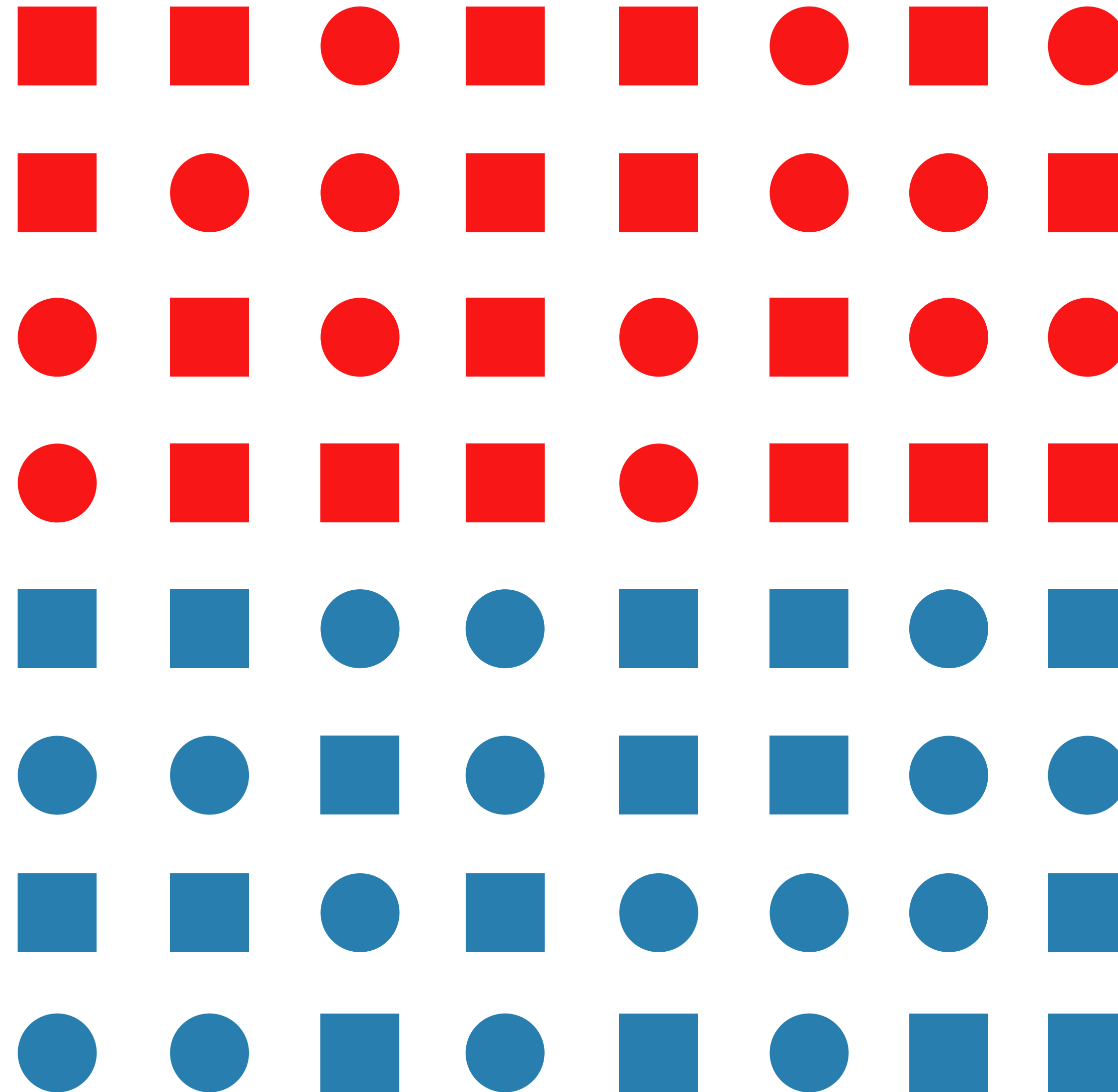


[C. G. Healey]



# Visual Pop-out

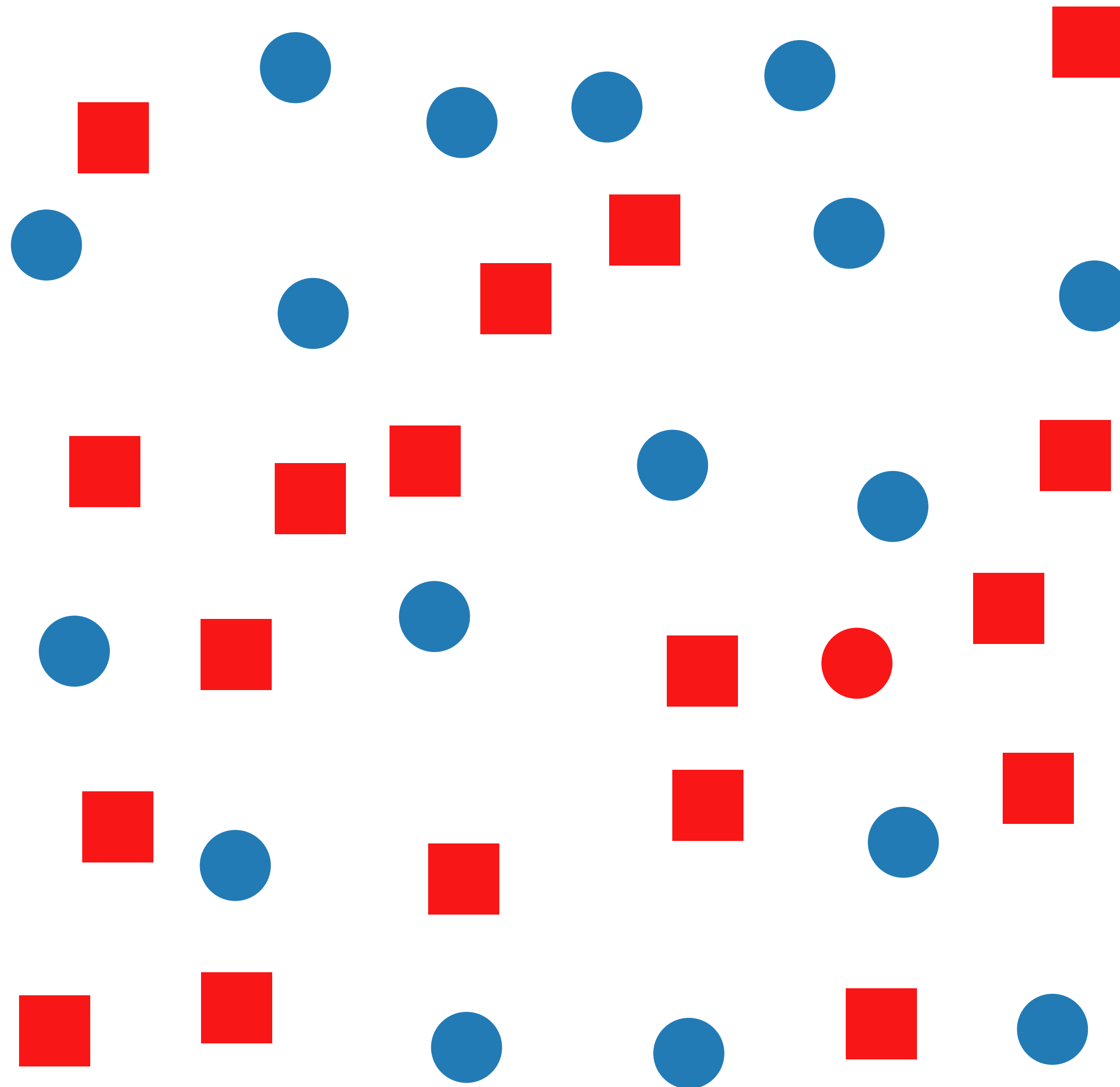
---



[C. G. Healey]

# Visual Perception Limitations

---

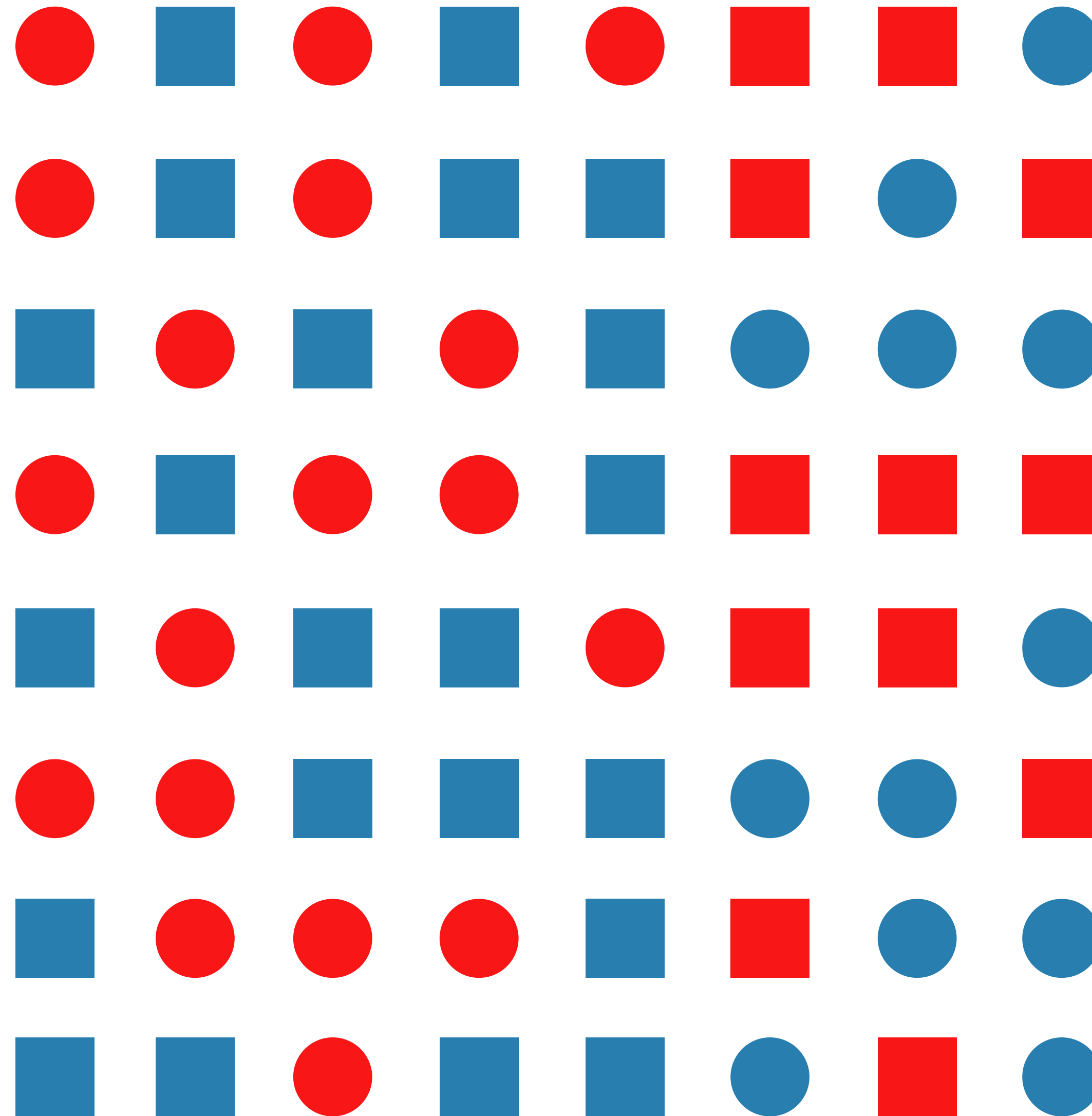


[C. G. Healey]



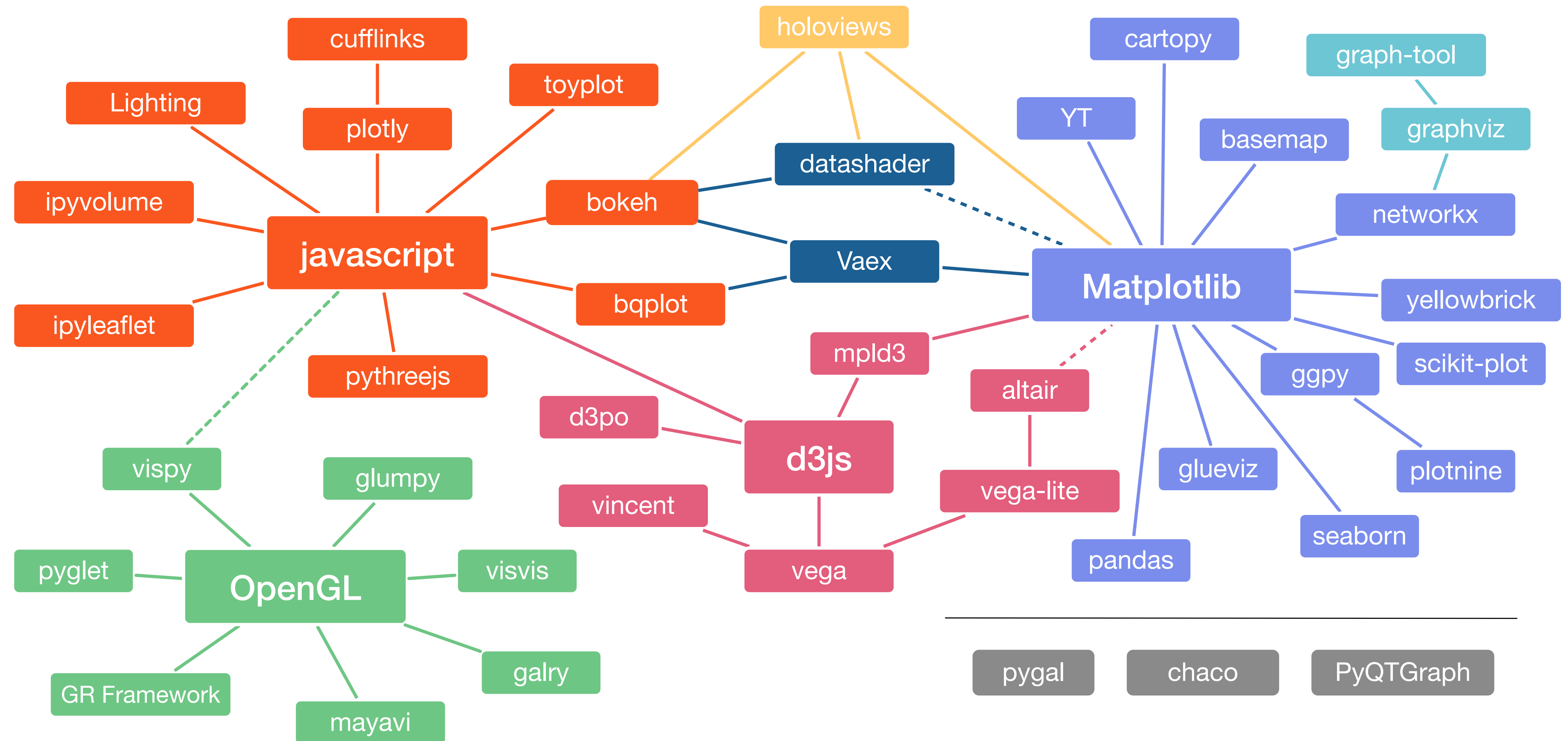
# Visual Perception Limitations

---



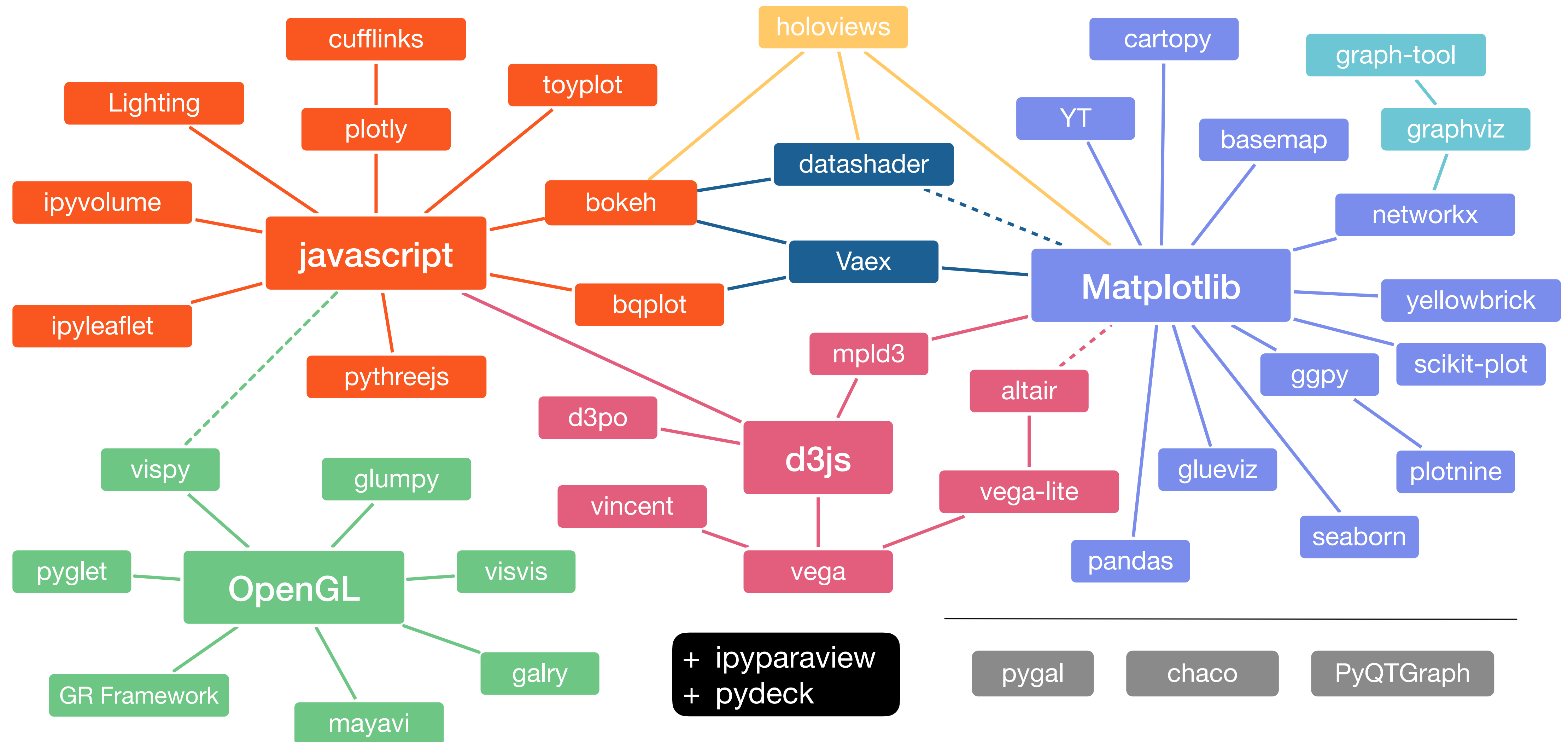
[C. G. Healey]

# The Python Visualization Landscape



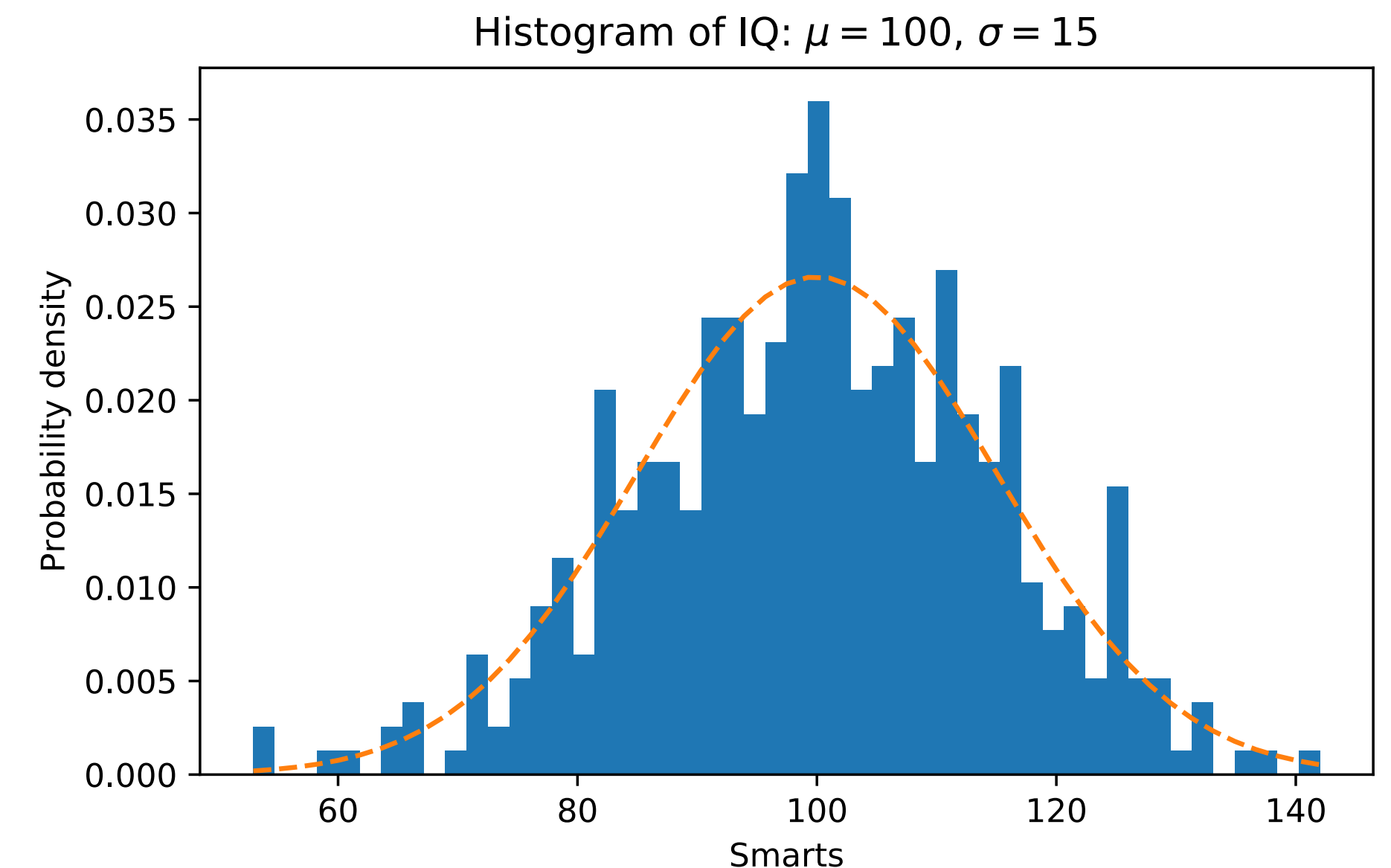


# The Python Visualization Landscape



# matplotlib

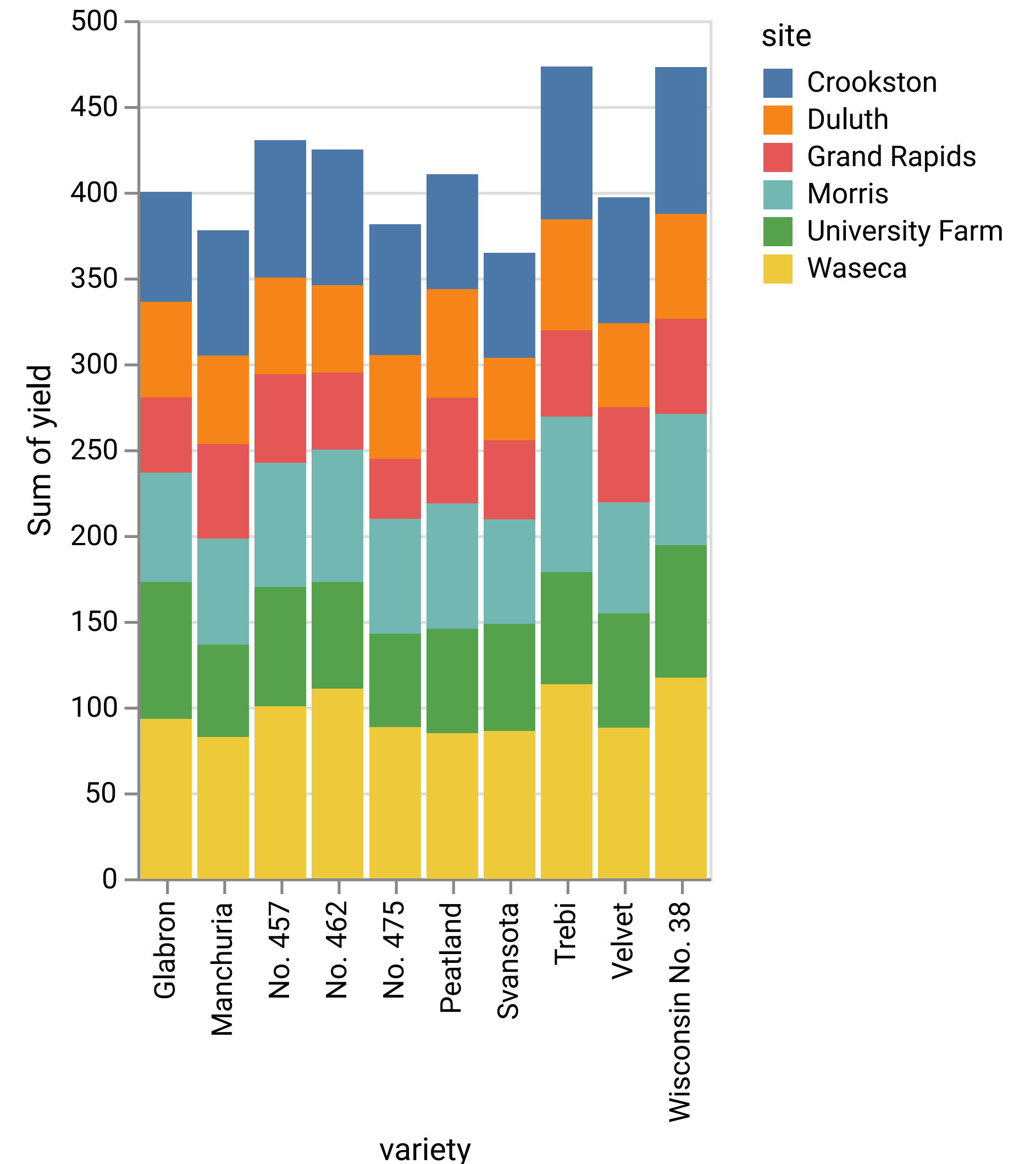
- Strengths:
  - Designed like Matlab
  - Many rendering backends
  - Can reproduce almost any plot
  - Proven, well-tested
- Weaknesses:
  - API is imperative
  - Not originally designed for the web
  - Dated styles





# 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



# 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

# matplotlib tutorials

---

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



# 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         |

Markers

| 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         |

# Data is Encoded via Visual Channels

## ➔ Position

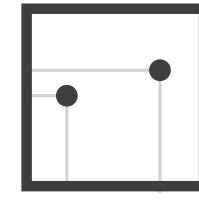
➔ Horizontal



➔ Vertical



➔ Both



## ➔ Color



## ➔ Shape



## ➔ Tilt

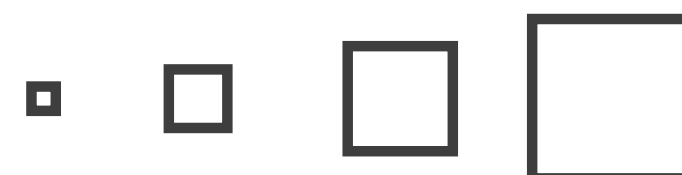


## ➔ Size

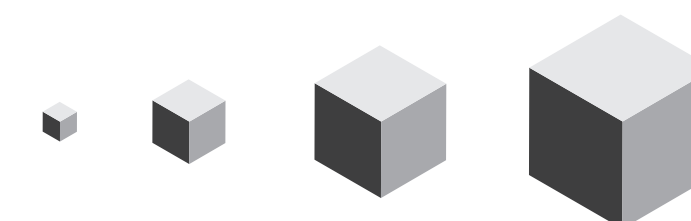
➔ Length



➔ Area



➔ Volume



[Munzner (ill. Maguire), 2014]

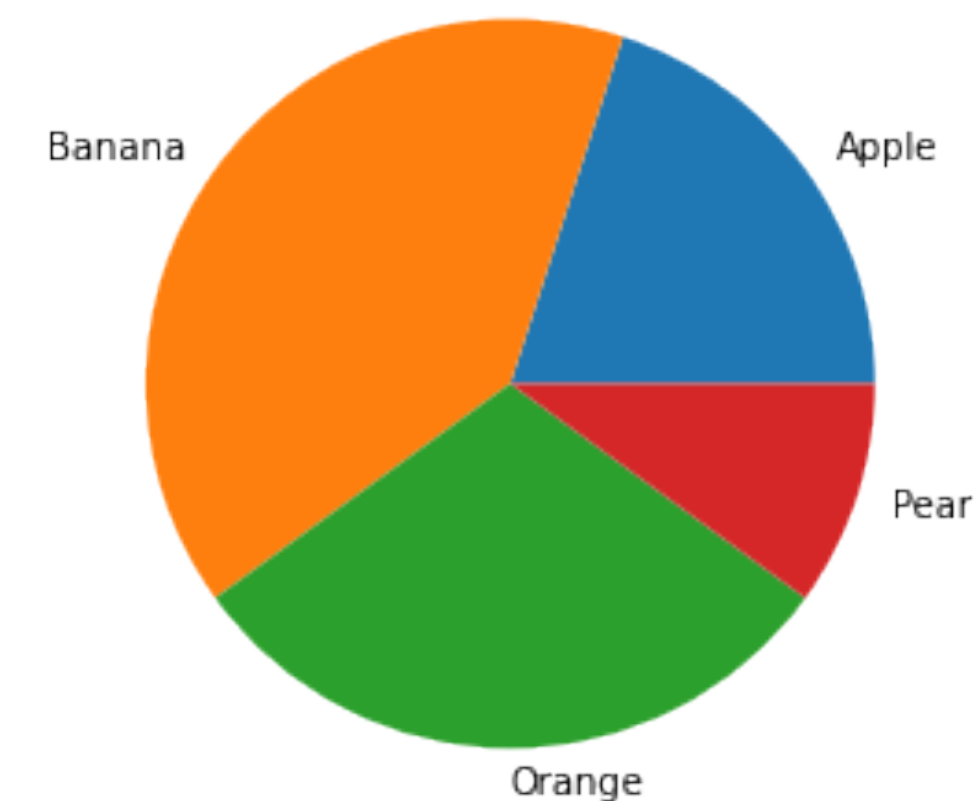
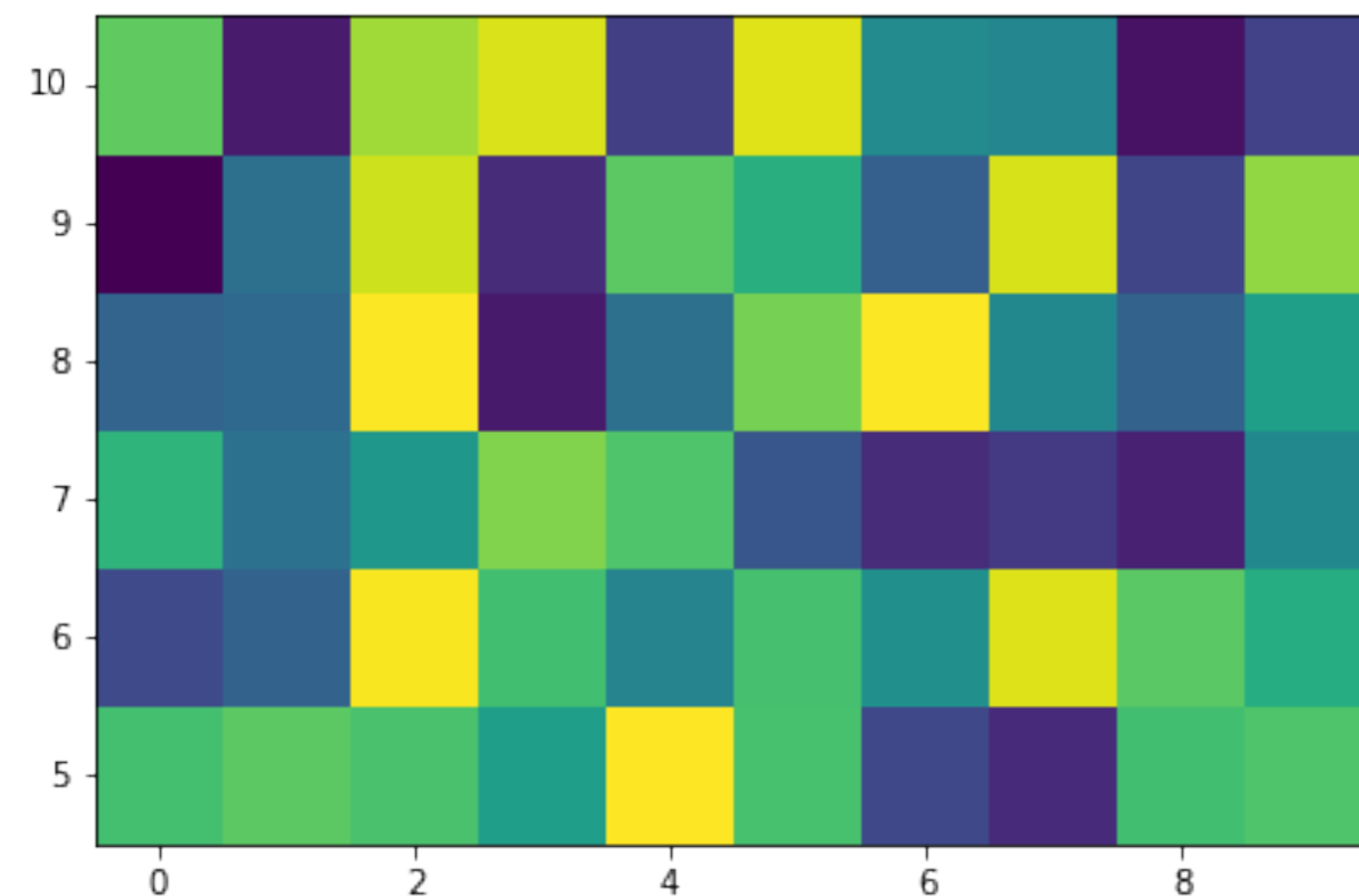
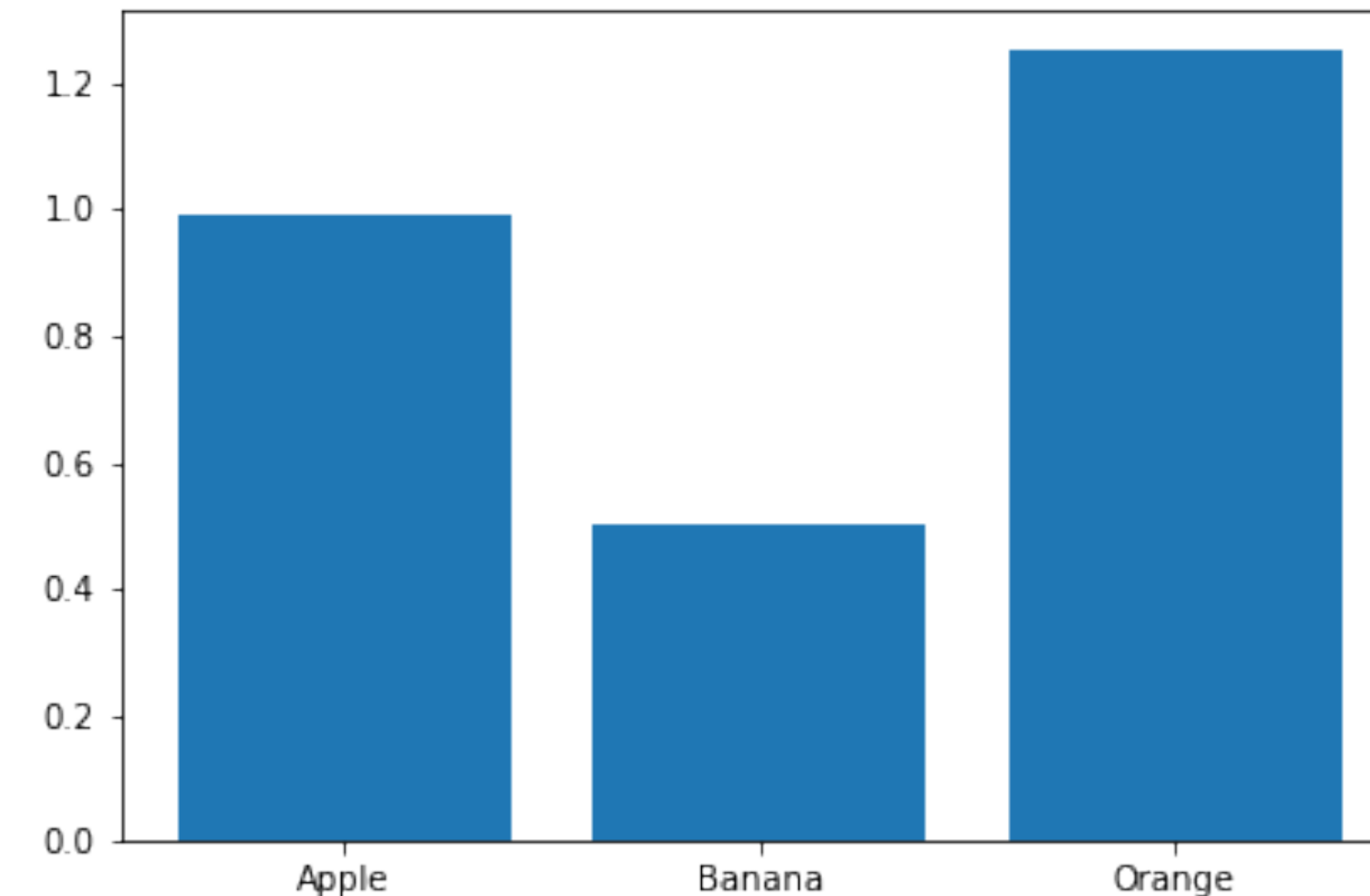
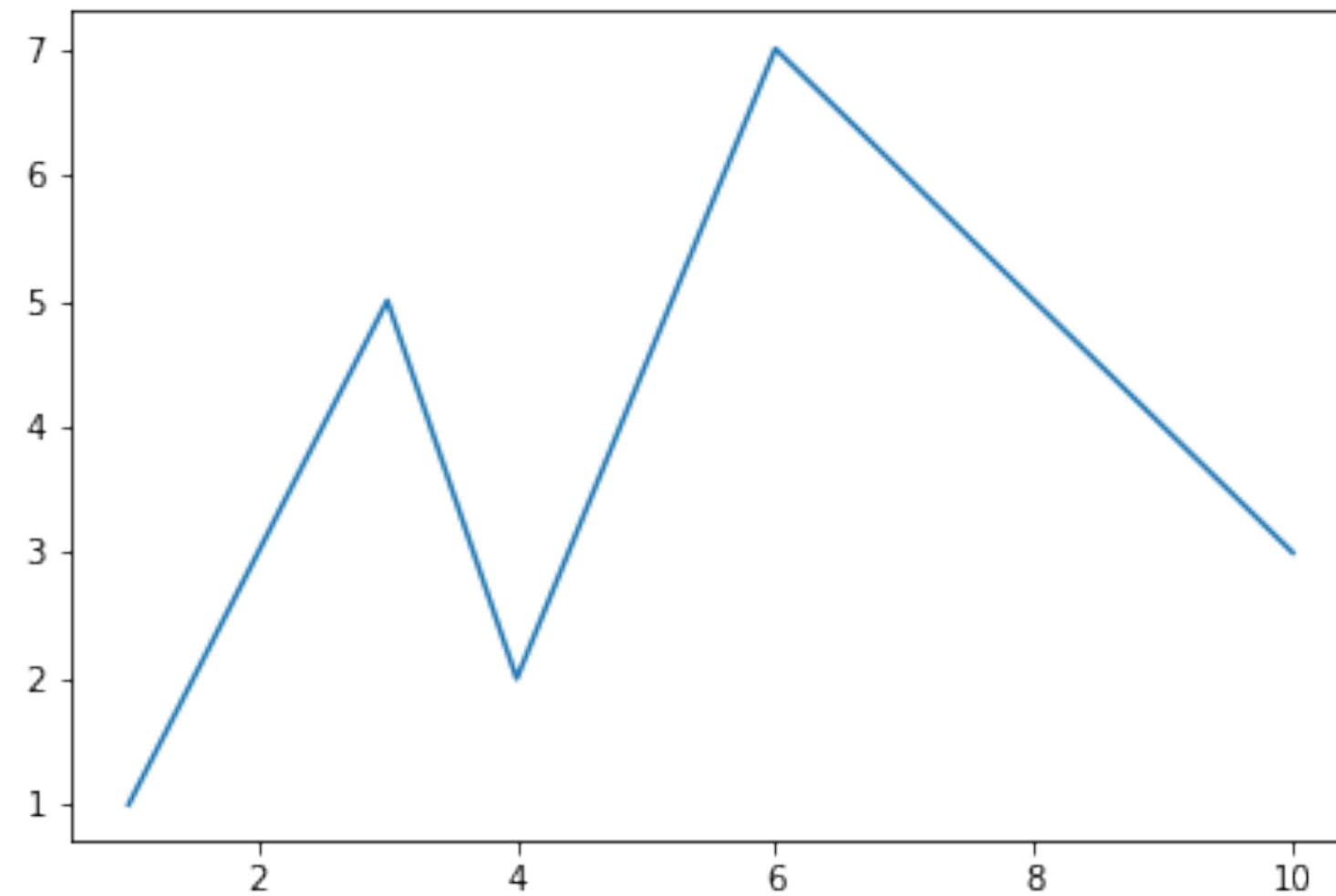


# Encoding Data Attributes via Channels

---

- ```
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:

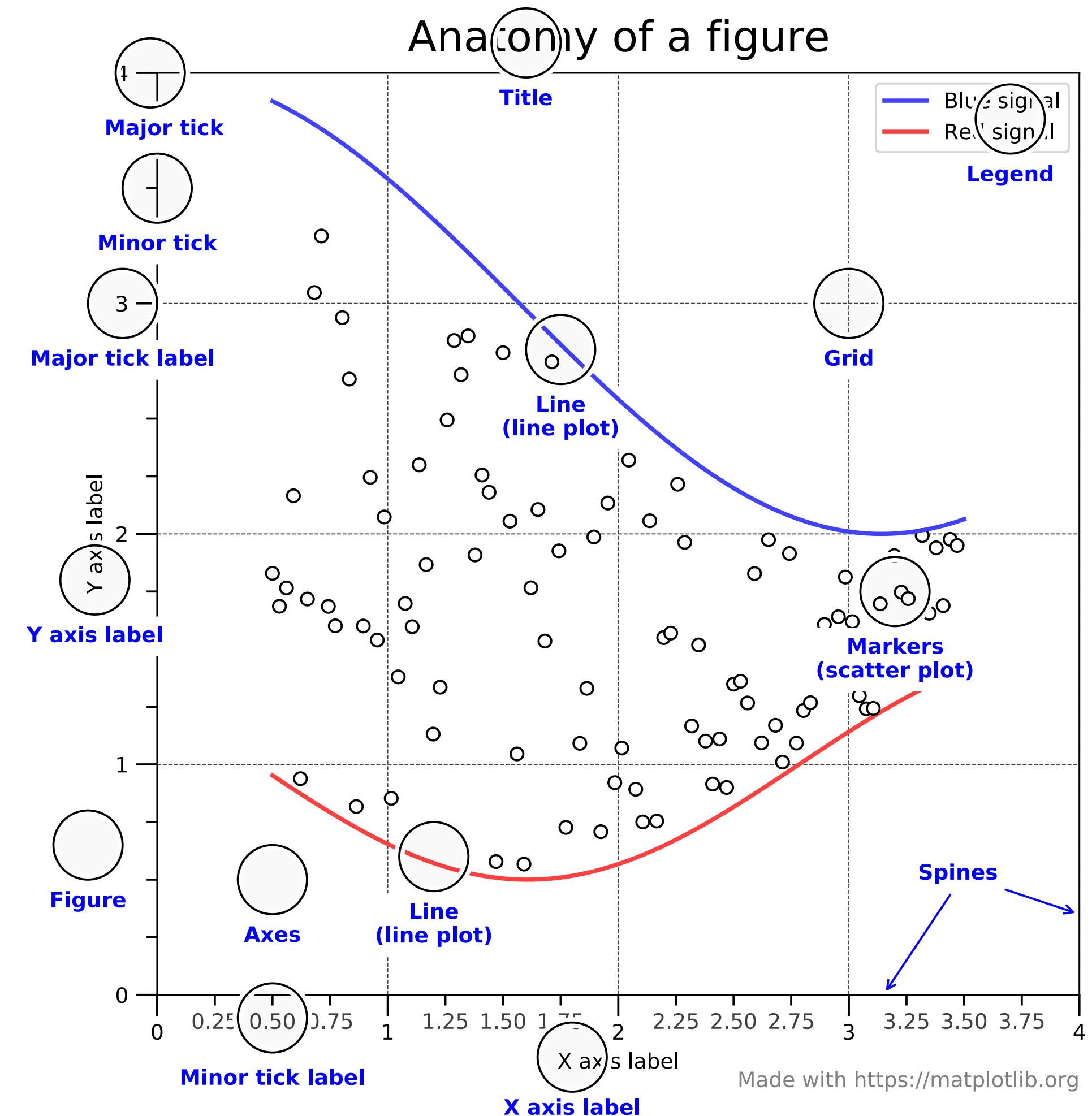
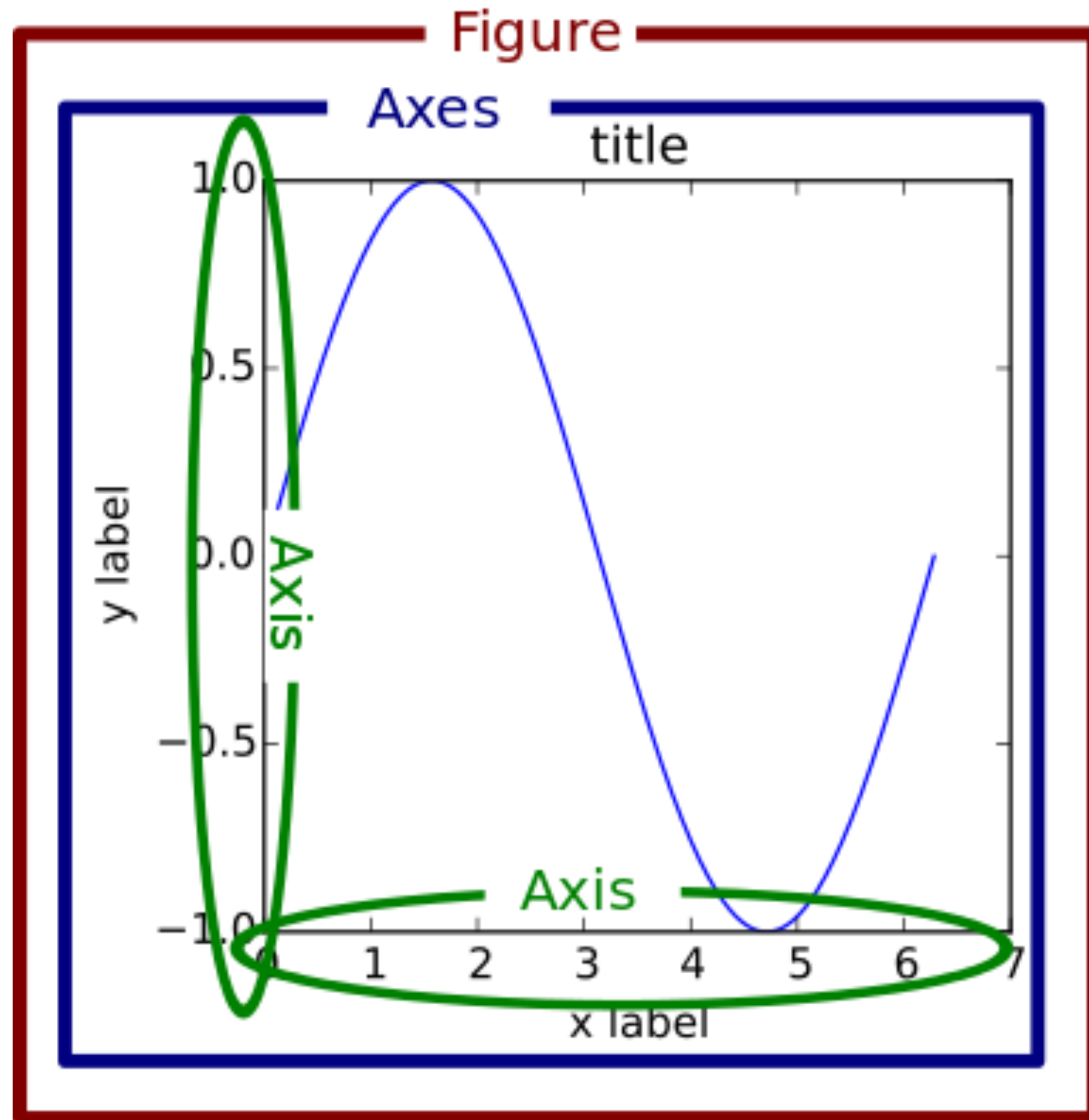
- `plt.pie([20, 40, 30, 10],
 labels=['Apple', 'Banana', 'Orange', 'Pear'])`

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)`