Programming Principles in Python (CSCI 503/490)

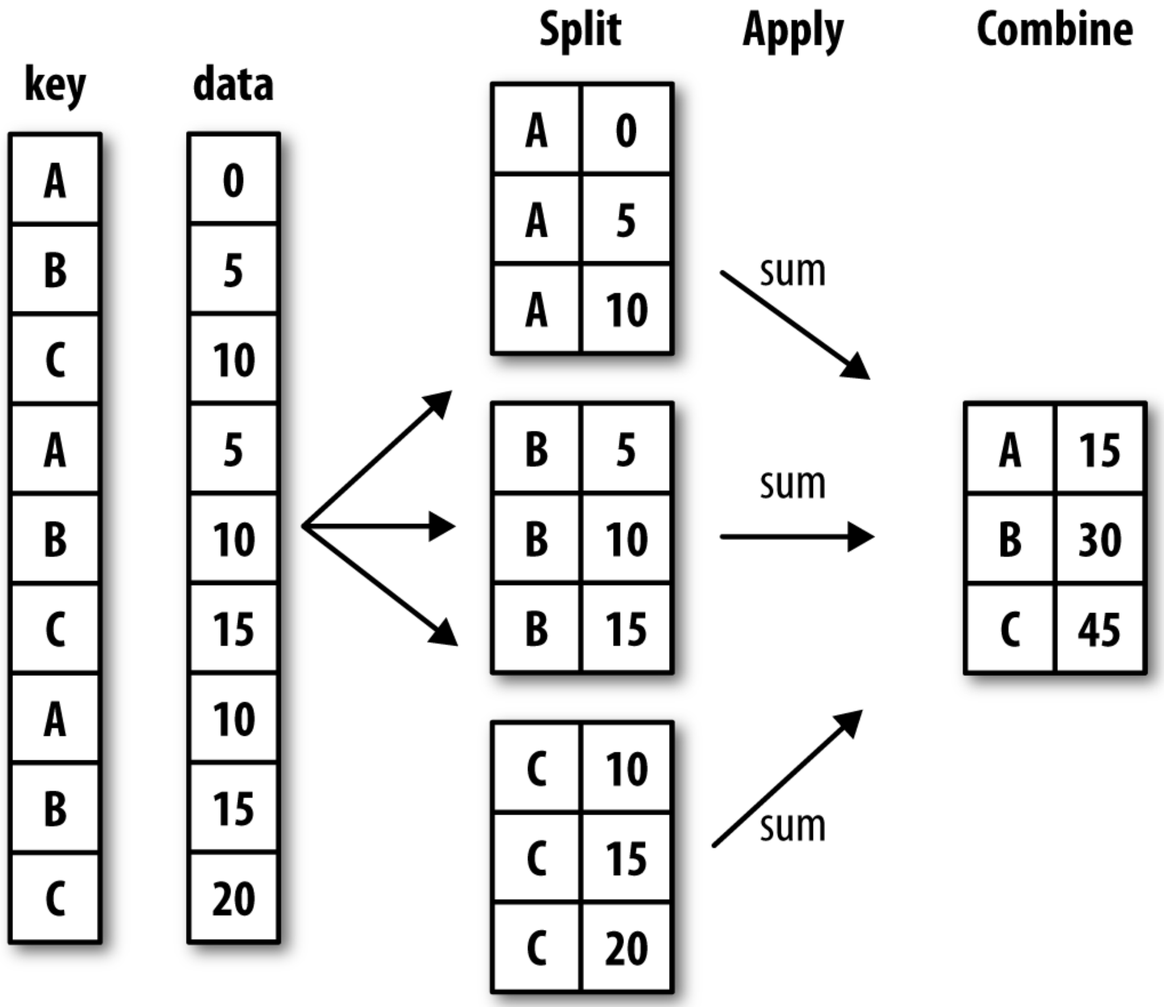
Data Visualization

Dr. David Koop

Derived Data

- Create new columns from existing columns
- pandas
  - `dfa["CulmenRatio"] = dfa['CLength'] / dfa['CDepth'] # Mut!`
  - `dfa = dfa.assign(CulmenRatio= dfa['CLength'] / dfa['CDepth'])`
- polars
  - ```
    df.with_columns(
            (df['CLength'] / df['CDepth']).alias('CulmenRatio'))
    ```
- Note that operations are computed in a vectorized manner
- Similarities to functional paradigm (map/filter):
  - specify the operation once, on entire column/frame
  - no loops
    ```

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'))`

Assignment 8

- Out Soon…
- Last Assignment
- Data and Visualization
- Same Energy Data

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]

Problem: Variables stored in both rows & columns

Mexico Weather, Global Historical Climatology Network

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

[H. Wickham, 2014]

Problem: Variables stored in both rows & columns

Mexico Weather, Global Historical Climatology Network

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

Variable in columns: day; Variable in rows: tmax/tmin

[H. Wickham, 2014]

Melting + Pivot

id	date	element	value
MX17004	2010-01-30	tmax	27.8
MX17004	2010-01-30	tmin	14.5
MX17004	2010-02-02	tmax	27.3
MX17004	2010-02-02	tmin	14.4
MX17004	2010-02-03	tmax	24.1
MX17004	2010-02-03	tmin	14.4
MX17004	2010-02-11	tmax	29.7
MX17004	2010-02-11	tmin	13.4
MX17004	2010-02-23	tmax	29.9
MX17004	2010-02-23	tmin	10.7

(a) Molten data

id	date	tmax	tmin
MX17004	2010-01-30	27.8	14.5
MX17004	2010-02-02	27.3	14.4
MX17004	2010-02-03	24.1	14.4
MX17004	2010-02-11	29.7	13.4
MX17004	2010-02-23	29.9	10.7
MX17004	2010-03-05	32.1	14.2
MX17004	2010-03-10	34.5	16.8
MX17004	2010-03-16	31.1	17.6
MX17004	2010-04-27	36.3	16.7
MX17004	2010-05-27	33.2	18.2

(b) 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

Unpivot/Melt

- Two sets of columns to identify:
  - Value vars: columns to unpivot: `on` / `value_vars` (`None` → all not specified)
  - Index vars: columns to keep: `index` / `id_vars`
- Polars: unpivot
  - `wdf.unpivot(index=['id','year','month','element'])`
- Pandas: melt
  - `wdfa.melt(id_vars=['id','year','month','element'])`

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

Pivot

- **Three** sets of columns to identify:
  - Columns: columns to pivot: `on` / `columns`
  - Index: columns to keep: `index`
  - Values: column to fill new columns: `values`
- Polars:
  -
```
wdf_up.pivot('element',
             index=['id','year','month','variable'],
             values='value')
```
- Pandas:
  -
```
wdfa_melt.pivot(columns='element',
                index=['id','year','month','variable'],
                values='value')
```

Melting + Pivot

id	date	element	value
MX17004	2010-01-30	tmax	27.8
MX17004	2010-01-30	tmin	14.5
MX17004	2010-02-02	tmax	27.3
MX17004	2010-02-02	tmin	14.4
MX17004	2010-02-03	tmax	24.1
MX17004	2010-02-03	tmin	14.4
MX17004	2010-02-11	tmax	29.7
MX17004	2010-02-11	tmin	13.4
MX17004	2010-02-23	tmax	29.9
MX17004	2010-02-23	tmin	10.7

(a) Molten data

id	date	tmax	tmin
MX17004	2010-01-30	27.8	14.5
MX17004	2010-02-02	27.3	14.4
MX17004	2010-02-03	24.1	14.4
MX17004	2010-02-11	29.7	13.4
MX17004	2010-02-23	29.9	10.7
MX17004	2010-03-05	32.1	14.2
MX17004	2010-03-10	34.5	16.8
MX17004	2010-03-16	31.1	17.6
MX17004	2010-04-27	36.3	16.7
MX17004	2010-05-27	33.2	18.2

(b) Tidy data

[H. Wickham, 2014]

String Methods

- Can do many of the same methods used for single strings on entire columns
- Requires `.str` prefix before calling the method
  - polars: `df['Species'].str.split('(')`
  - pandas: `dfa['Species'].str.split('(')`
- Also can extract from a list
  - polars: `df['Species'].str.split('(').list[0]`
  - pandas: `dfa['Species'].str.split('(').str[0]`
- Many, many more (see documentation):
  - pandas: link
  - polars: link

Datetime Support

- Python has datetime library to support dates and times
- pandas has a Timestamp data type that functions somewhat similarly
- polars has a Datetime date type that functions somewhat similarly
- Can convert timestamps
  - `pl.to_datetime` and `pl.str.to_datetime`: directed, require format
  - `pd.to_datetime`: versatile, can often guess format from a string
- Like string methods, also a `.dt` accessor for datetime methods/properties
  - polars: `df['date'].dt.year()`,
  - pandas: `dfa['date'].dt.year`

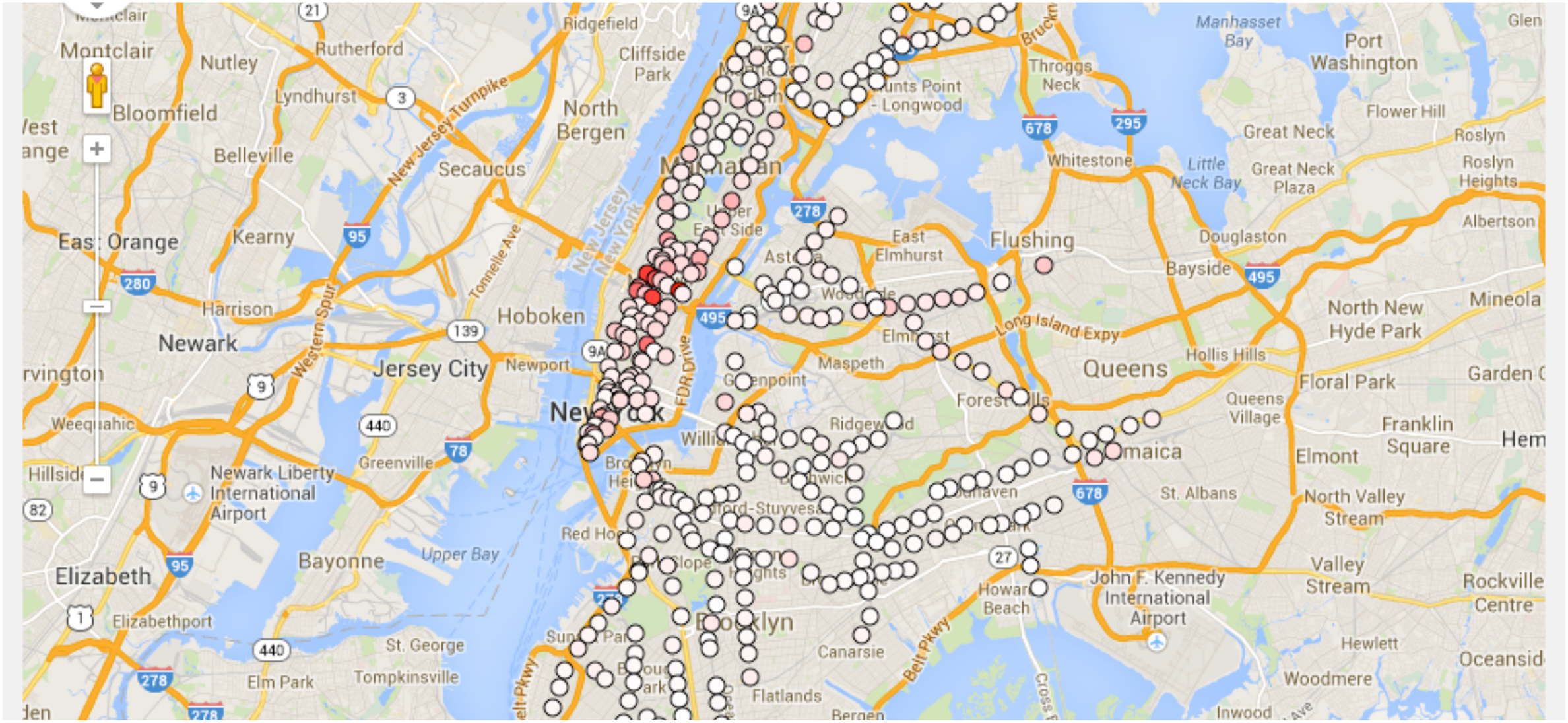
Data Exploration through Visualization

Transportation Data - NYC MTA

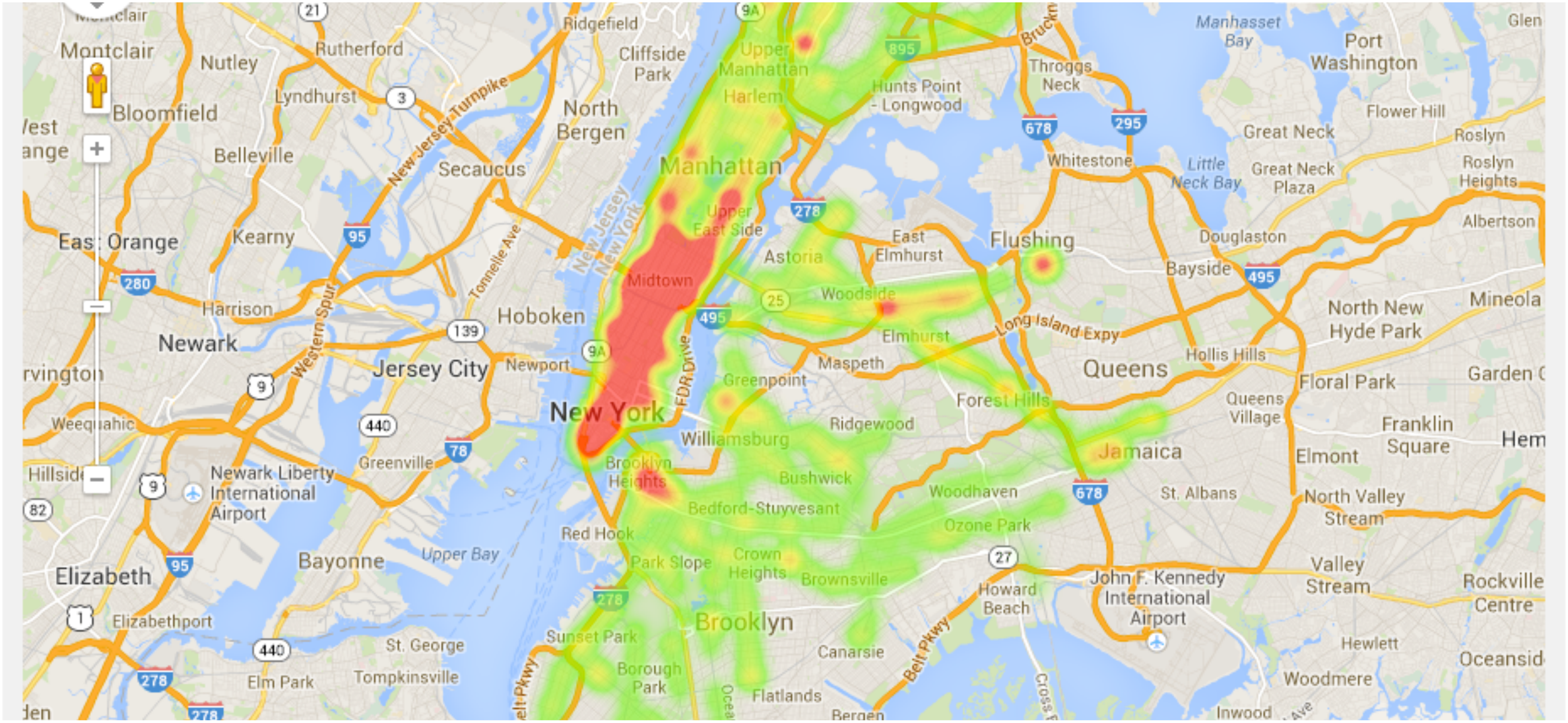
MTA Fare Data Exploration

	REMOTE	STATION	FF	SEN/DIS	7-D AFAS UNL	D AFAS/RMF U	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

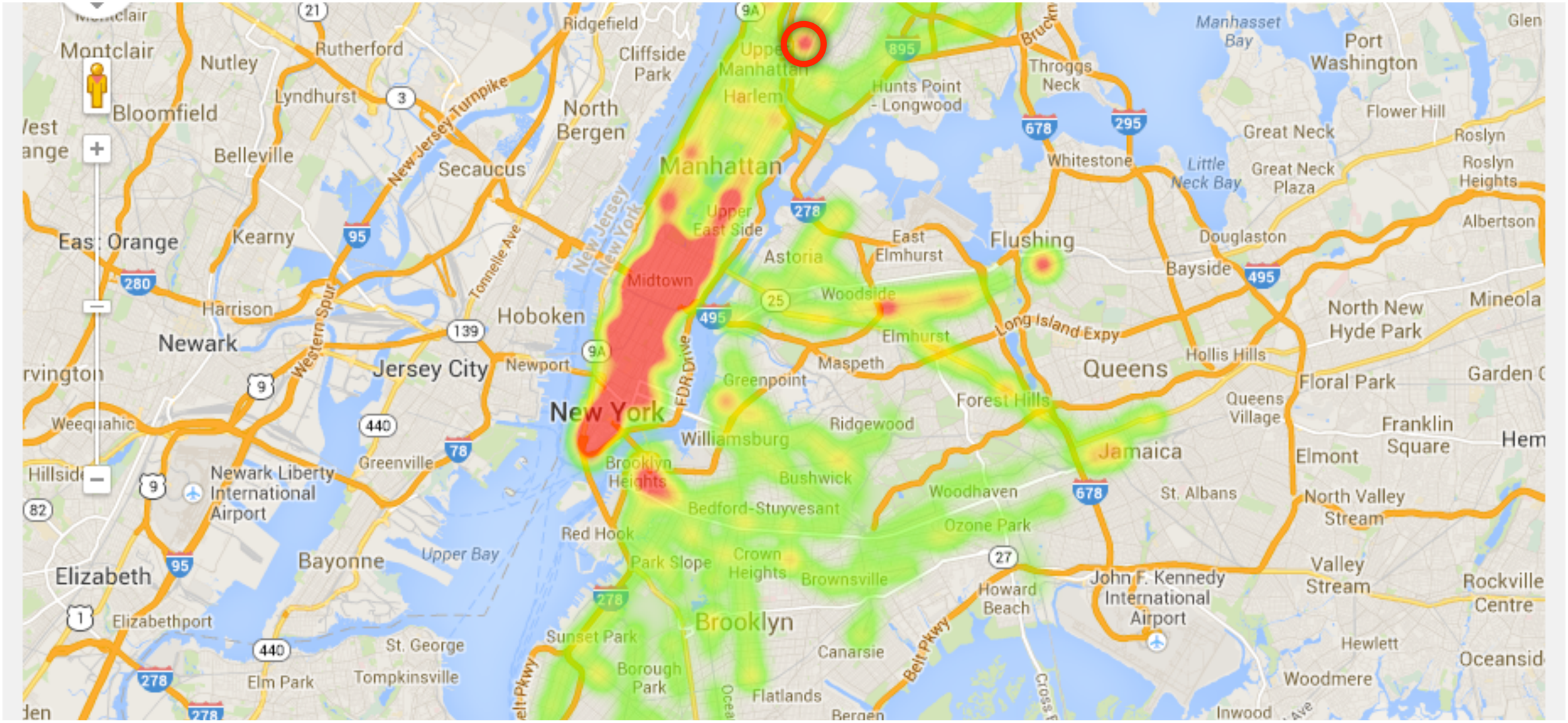
MTA Fare Data Exploration



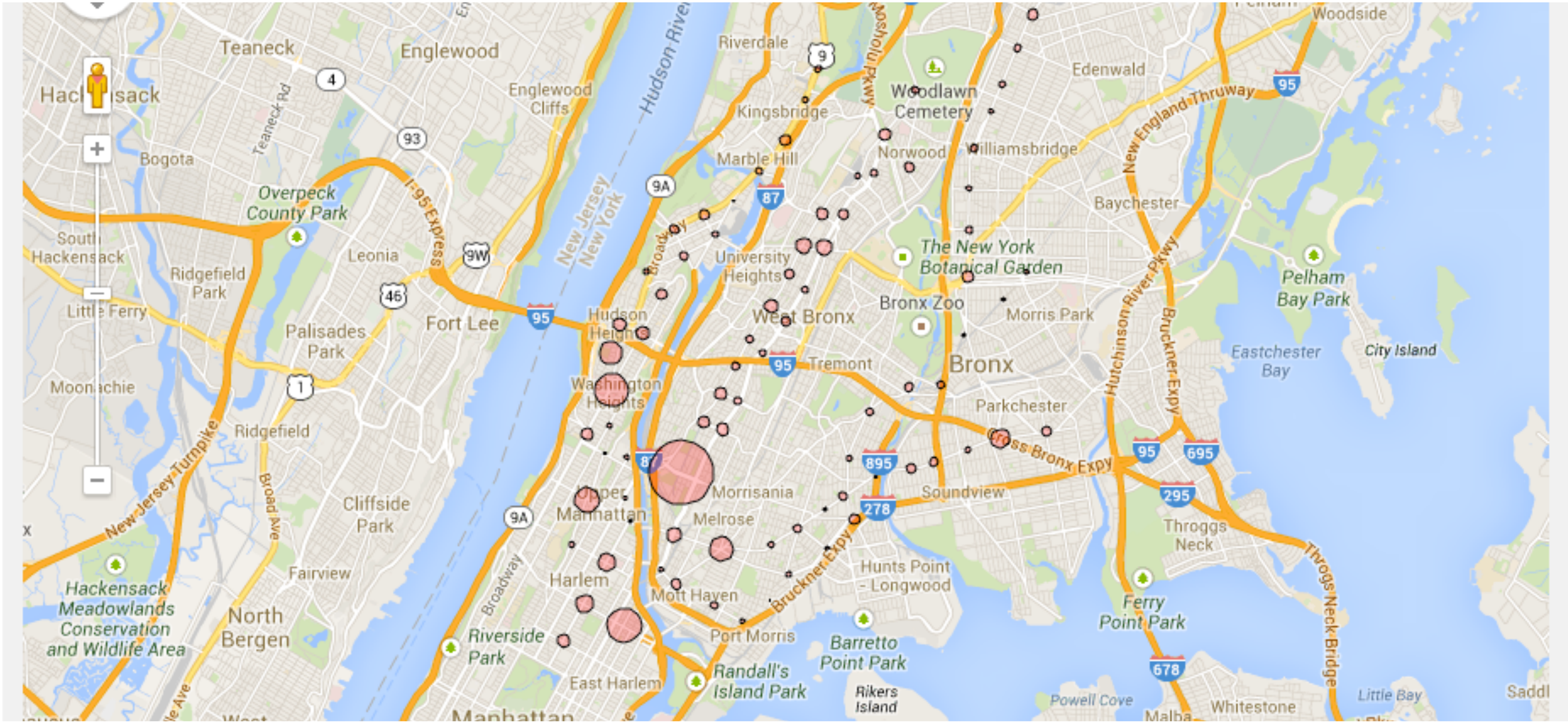
MTA Fare Data Exploration



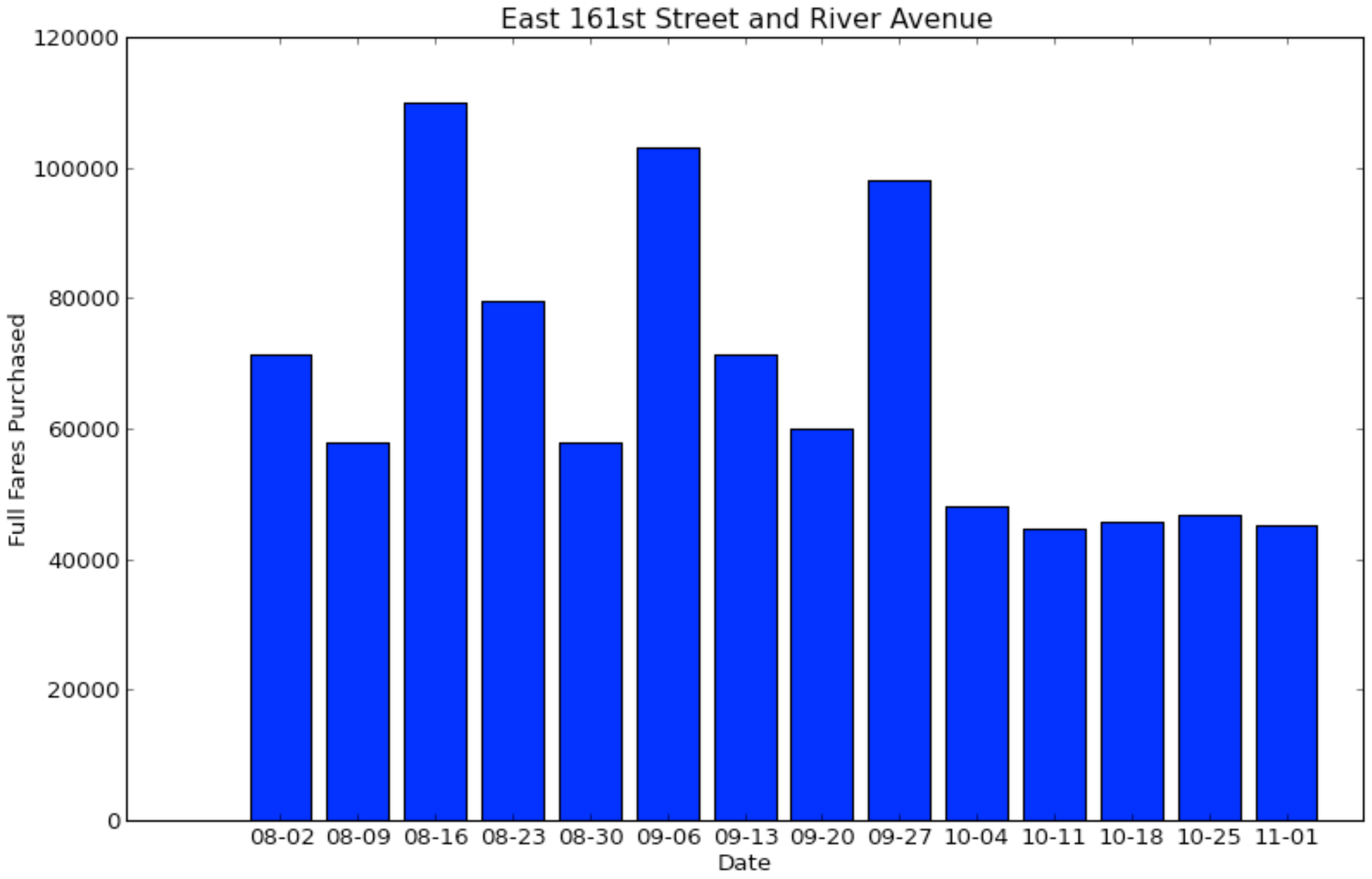
MTA Fare Data Exploration



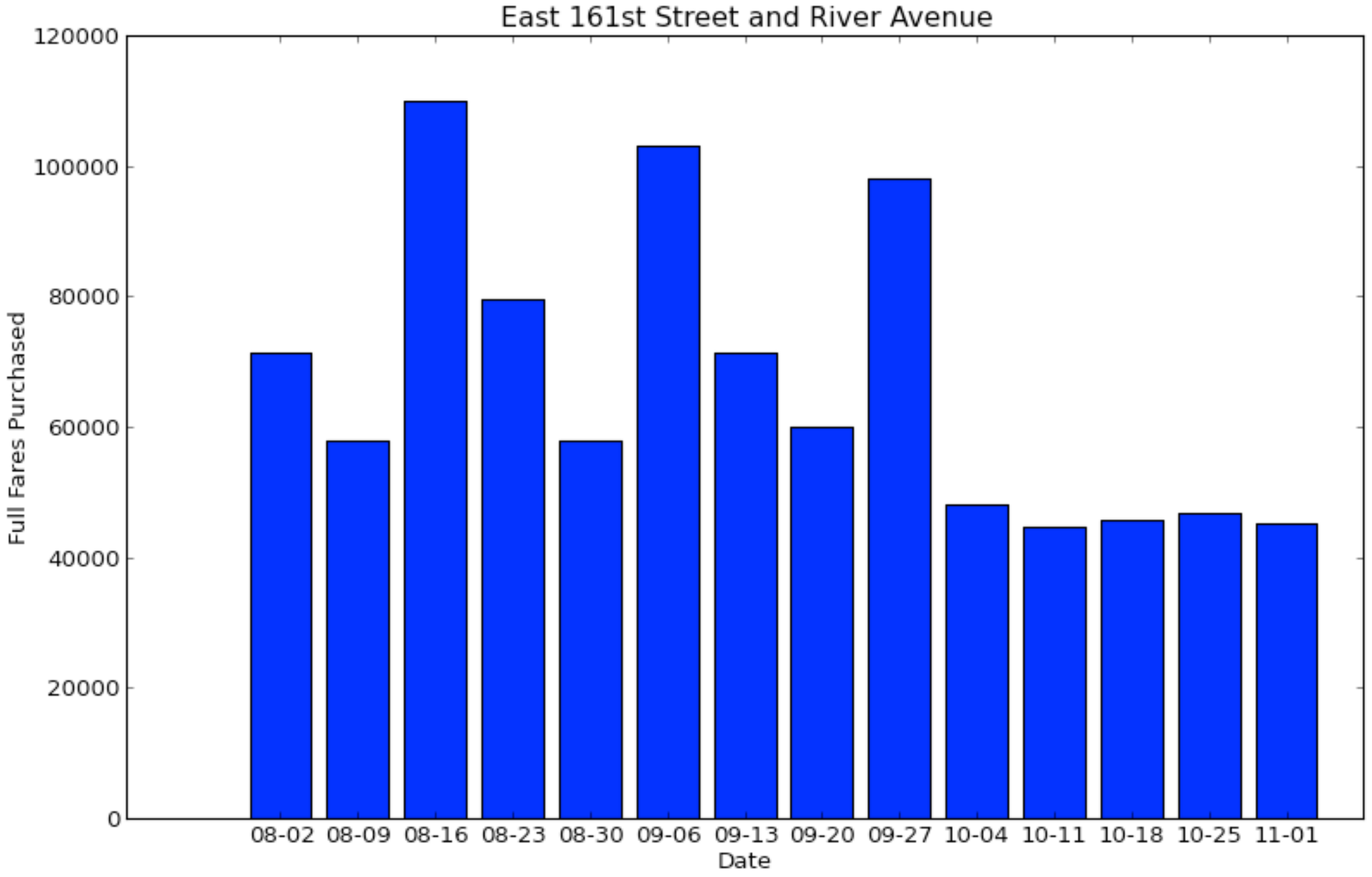
MTA Fare Data Exploration



MTA Fare Data Exploration



MTA Fare Data Exploration

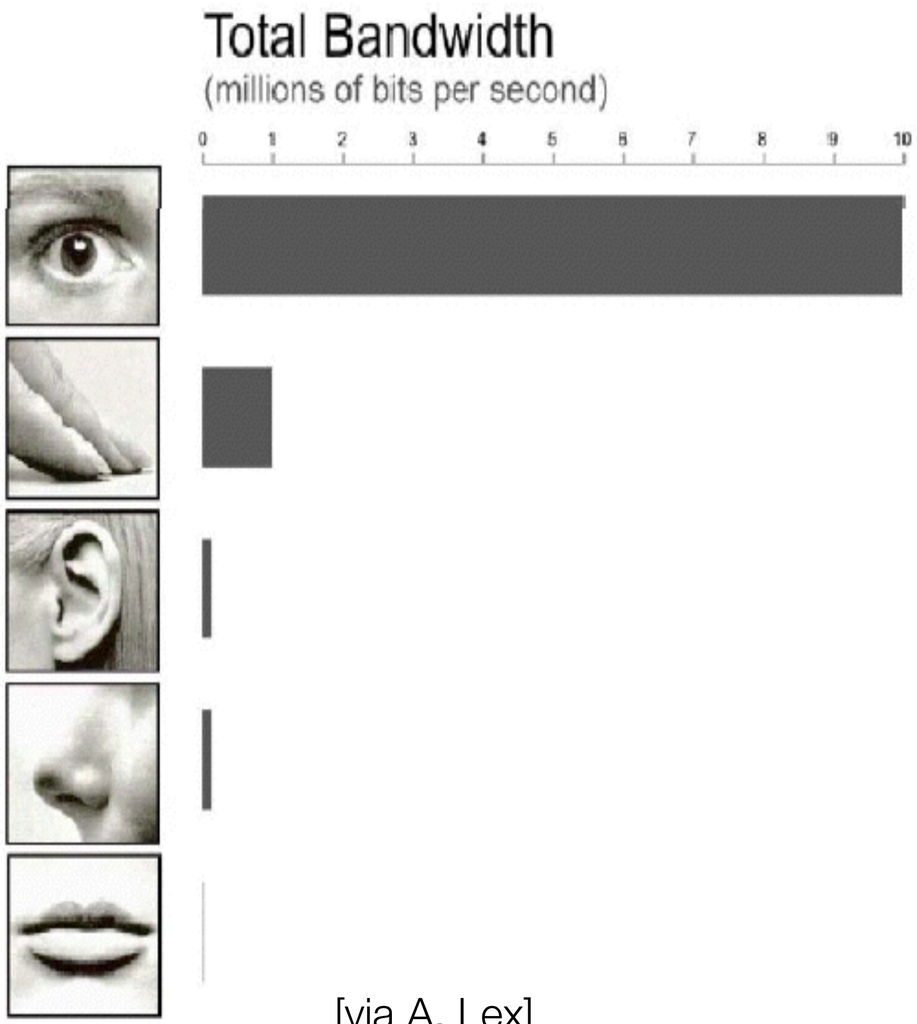


Definition of Visualization

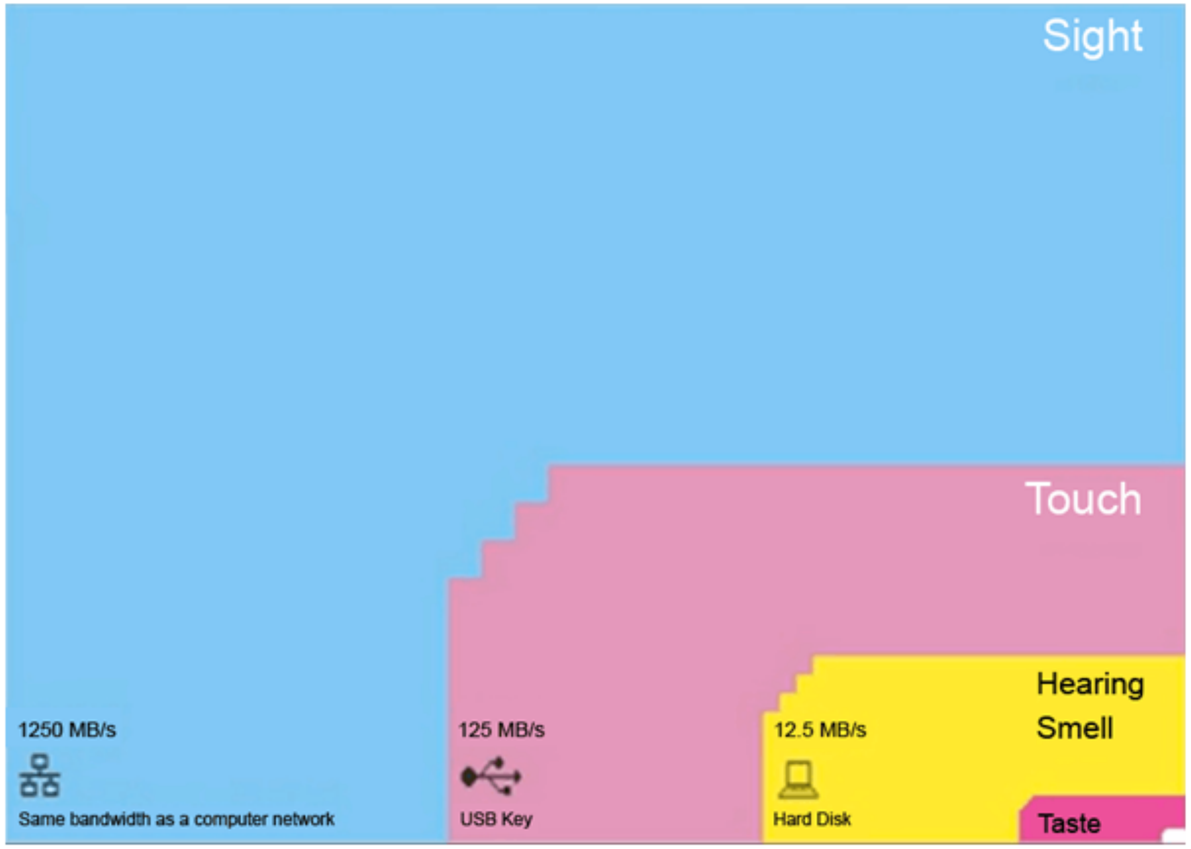
“Computer-based visualization systems provide visual representations of datasets designed to help people carry out tasks more effectively”

— 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]

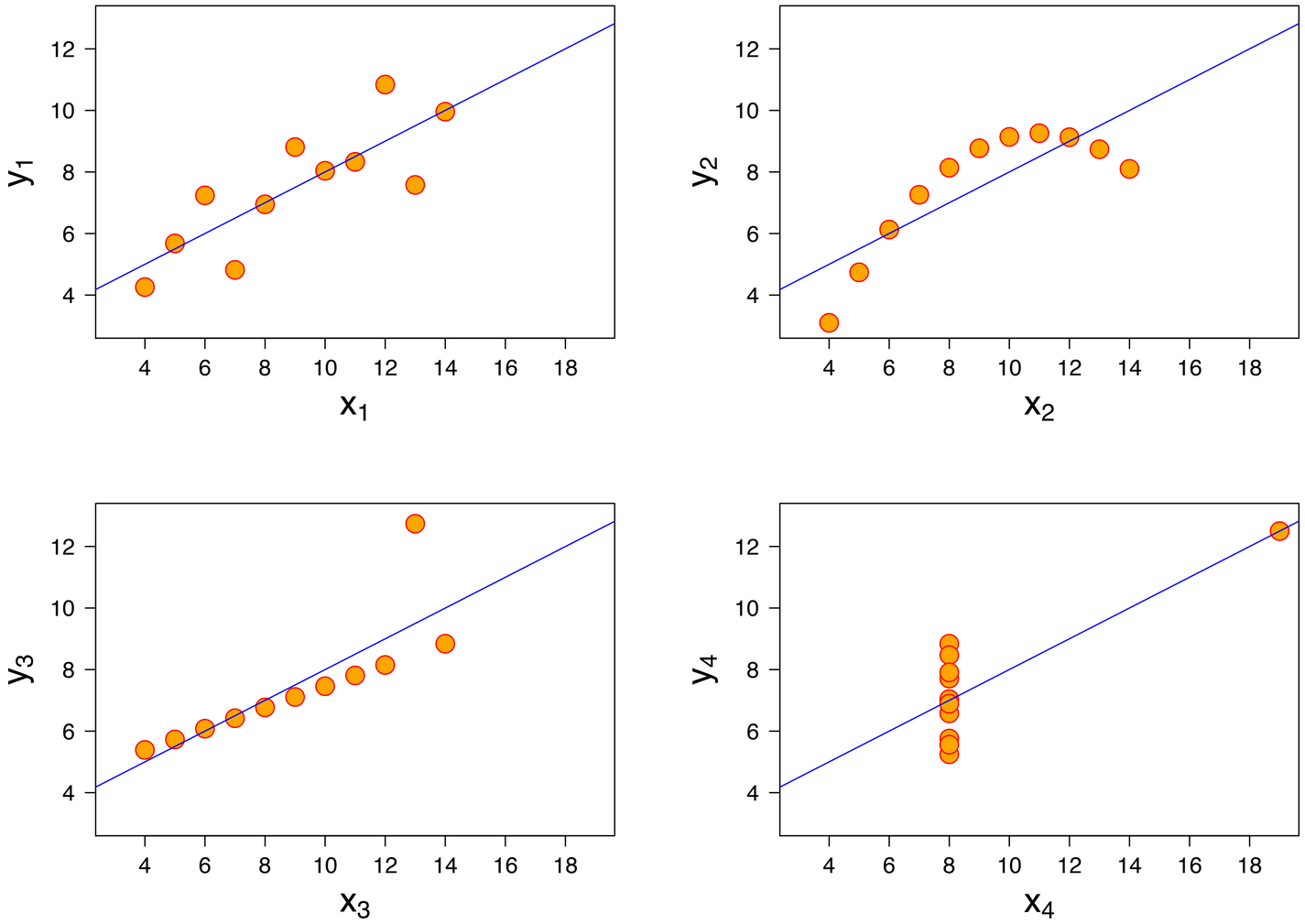
Why Visual?

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

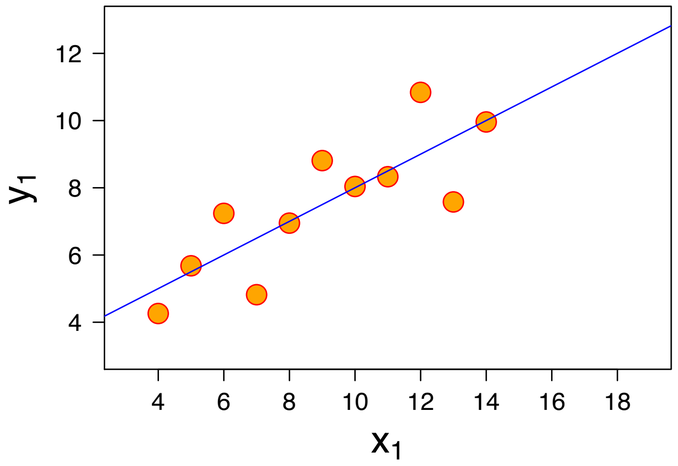
[F. J. Anscombe]

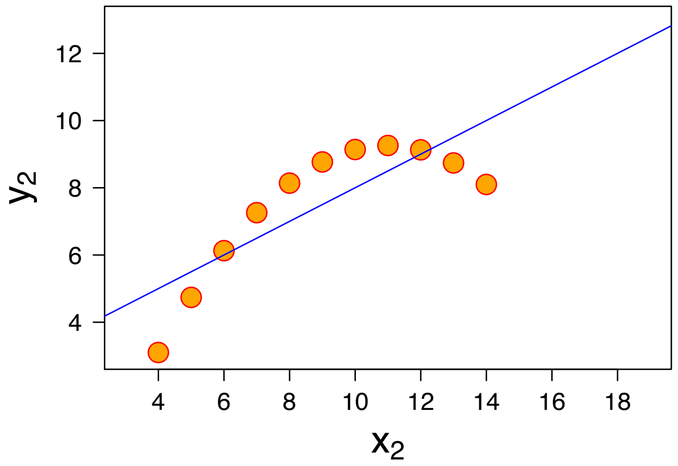
Why Visual?

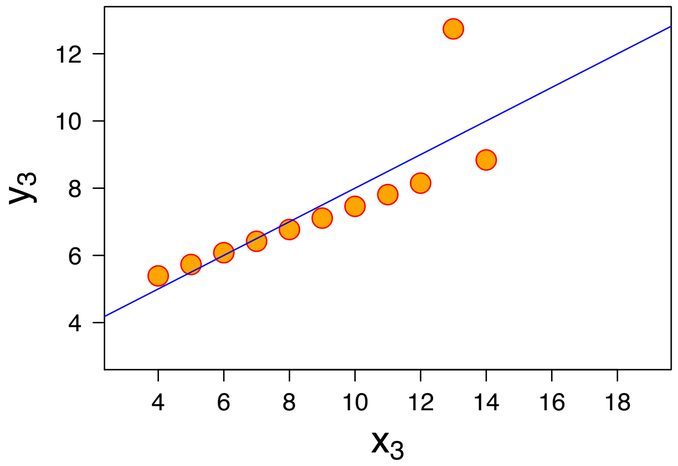


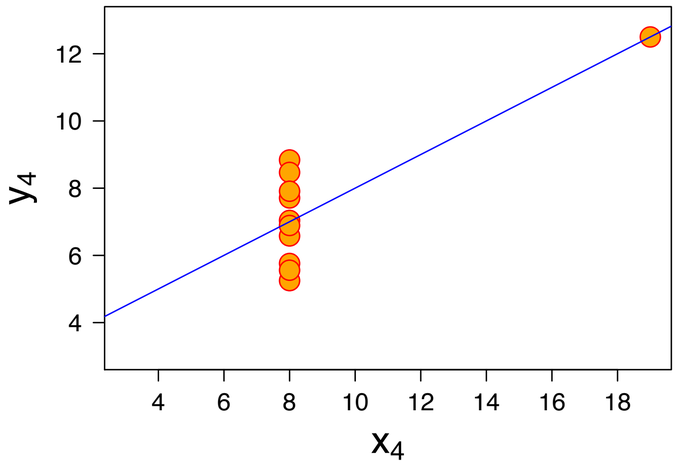
[F. J. Anscombe]

Why Visual?









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

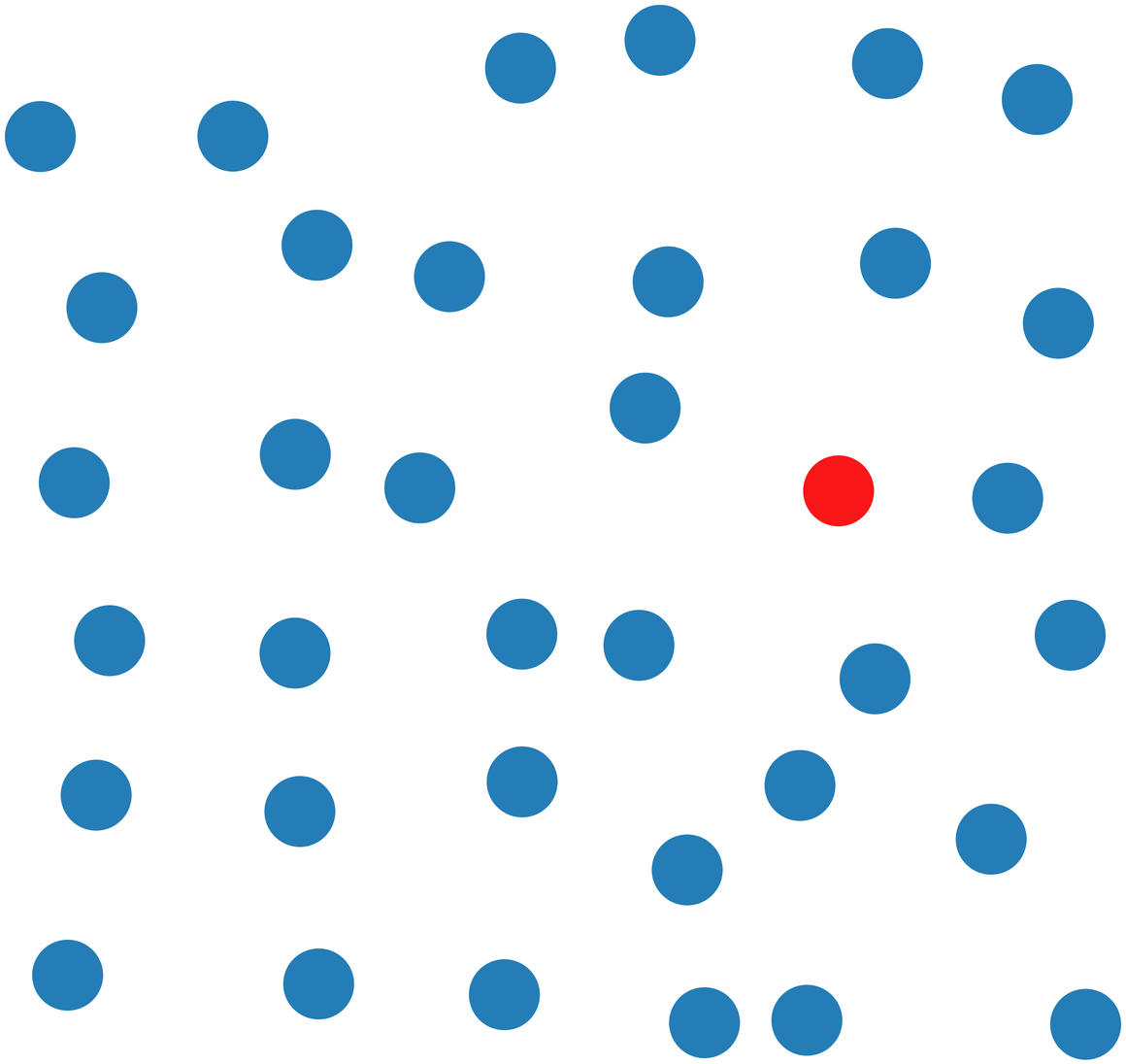
[F. J. Anscombe]

Visualization Goals

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

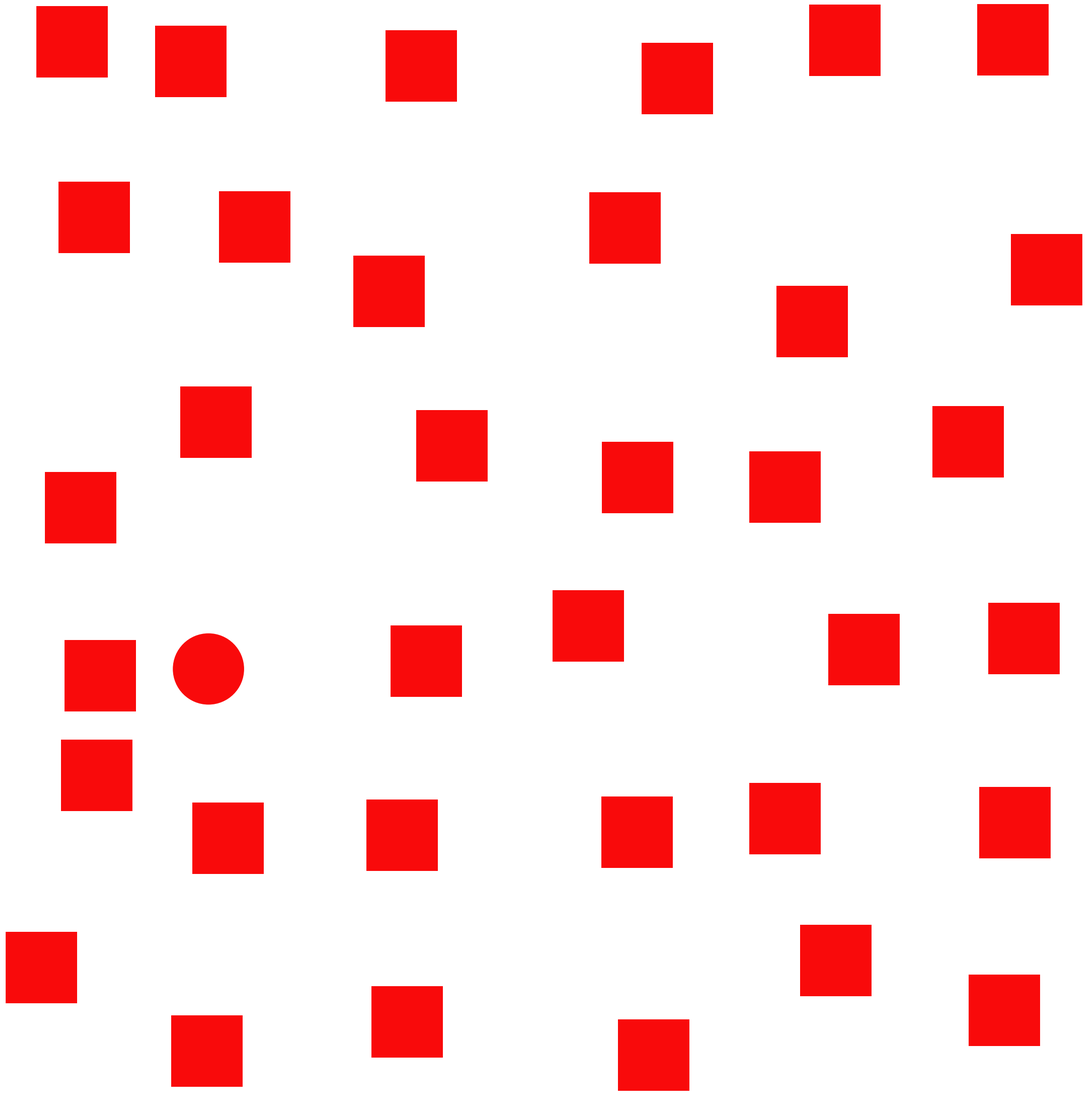
- Identify patterns, trends
- Spot outliers
- Find similarities, correlation

Visual Pop-out



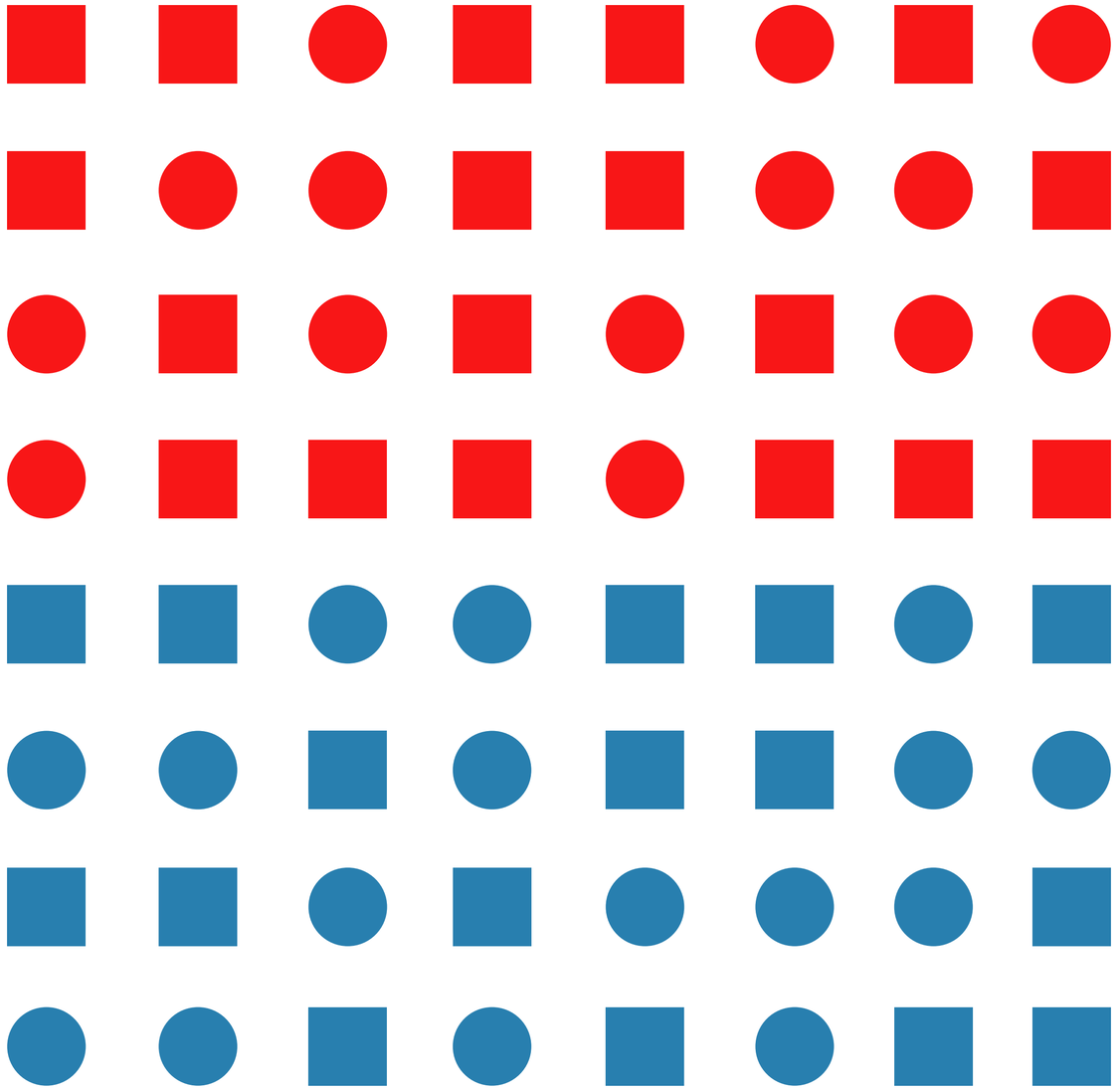
[C. G. Healey]

Visual Pop-out



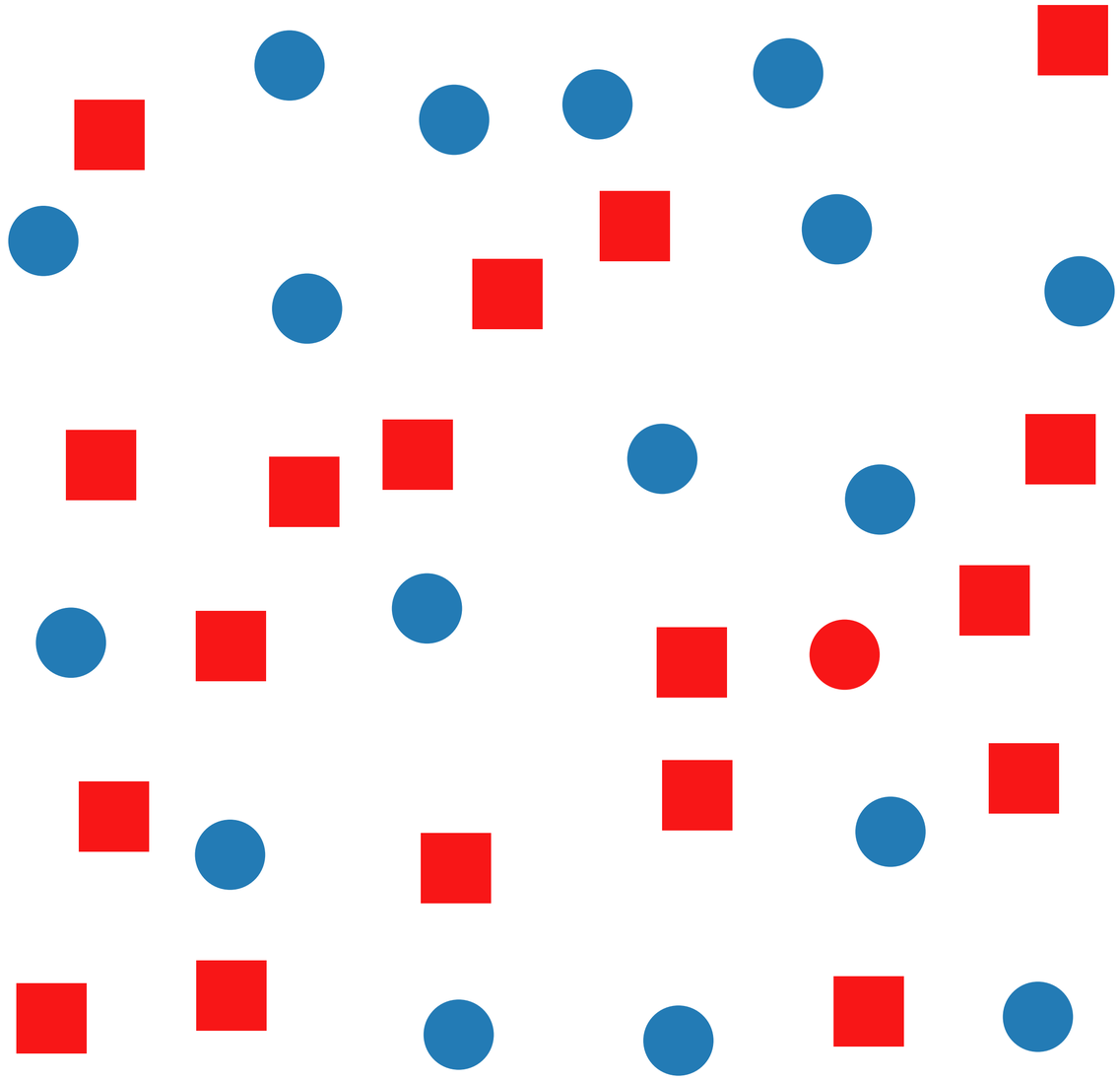
[C. G. Healey]

Visual Pop-out



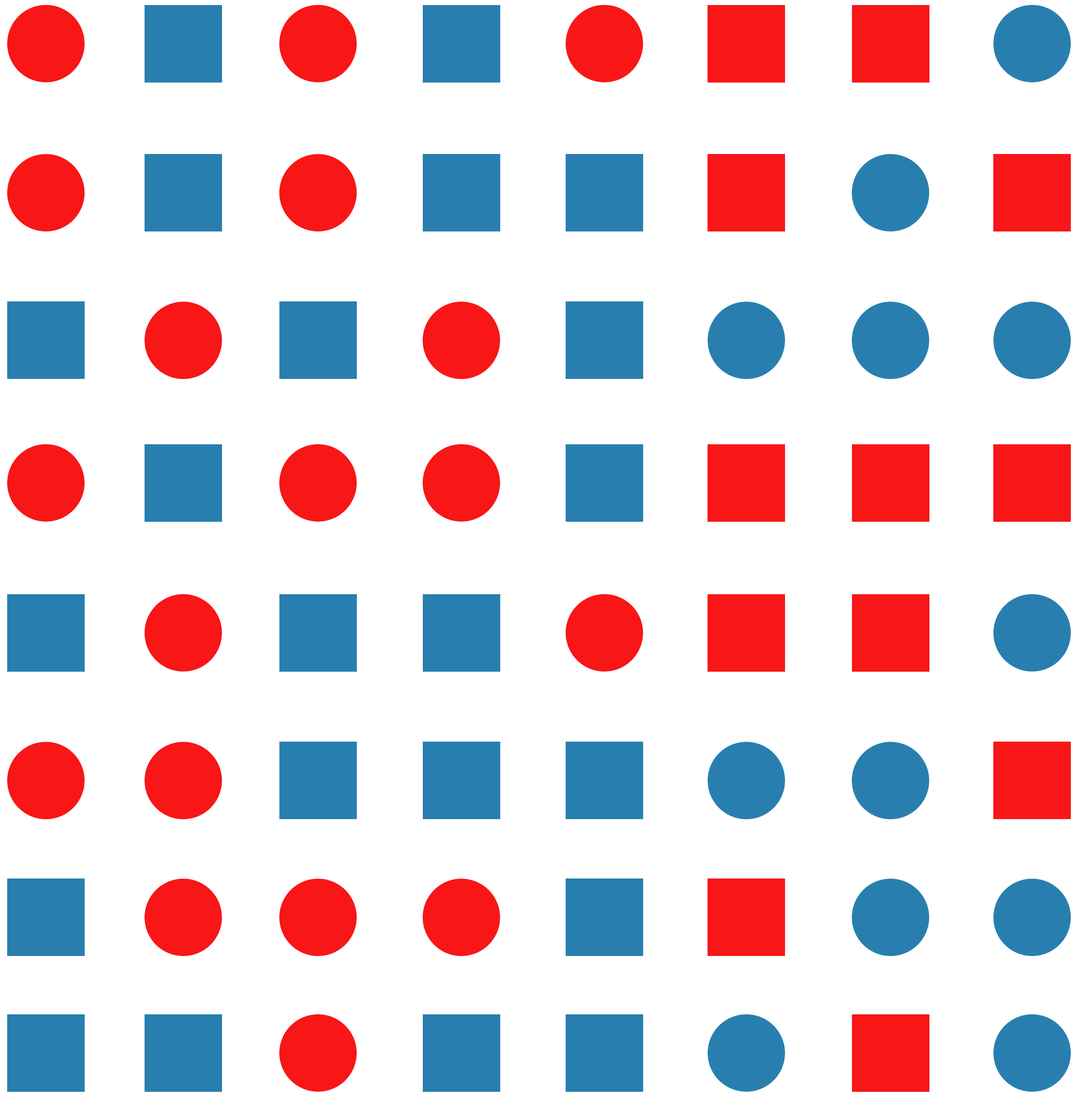
[C. G. Healey]

Visual Perception Limitations



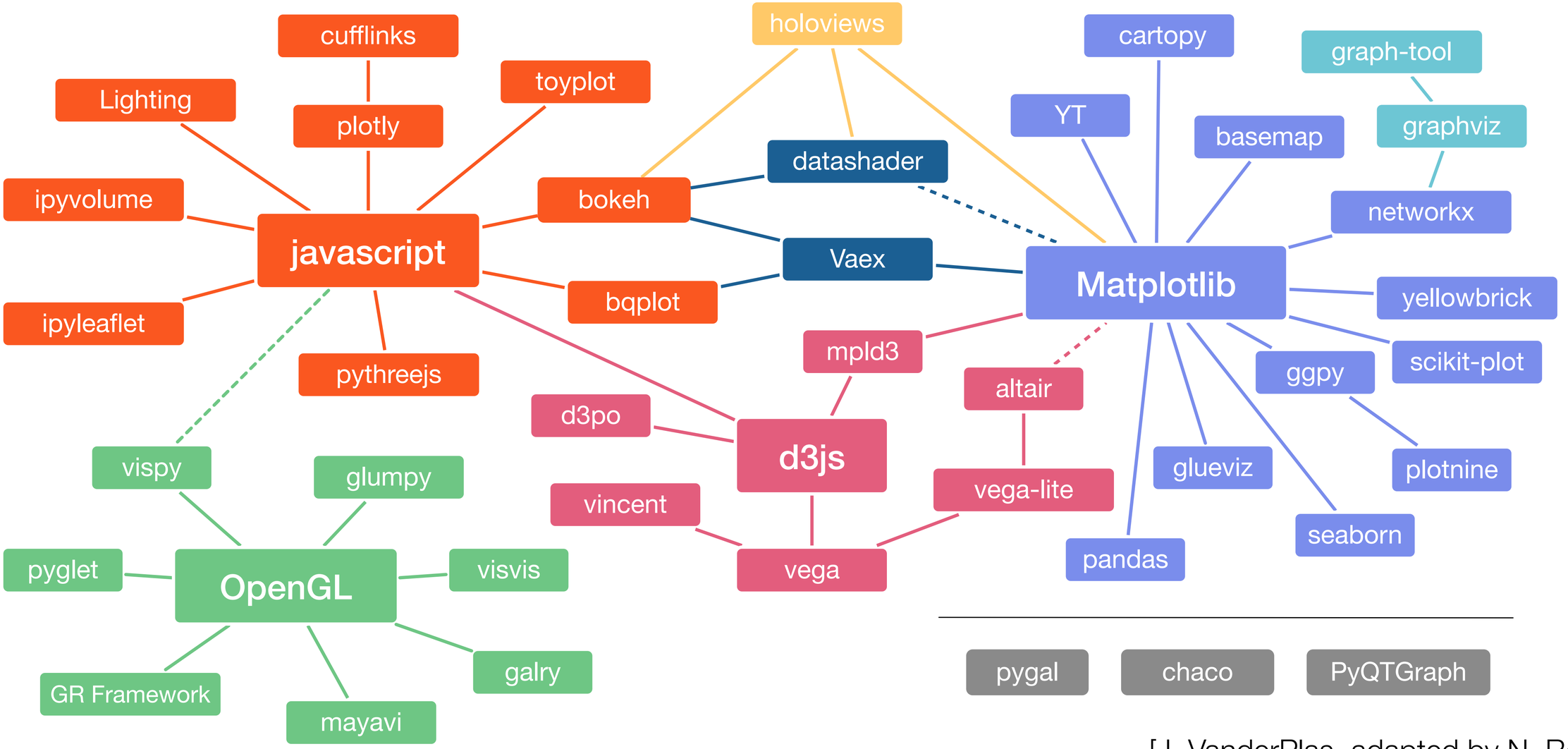
[C. G. Healey]

Visual Perception Limitations



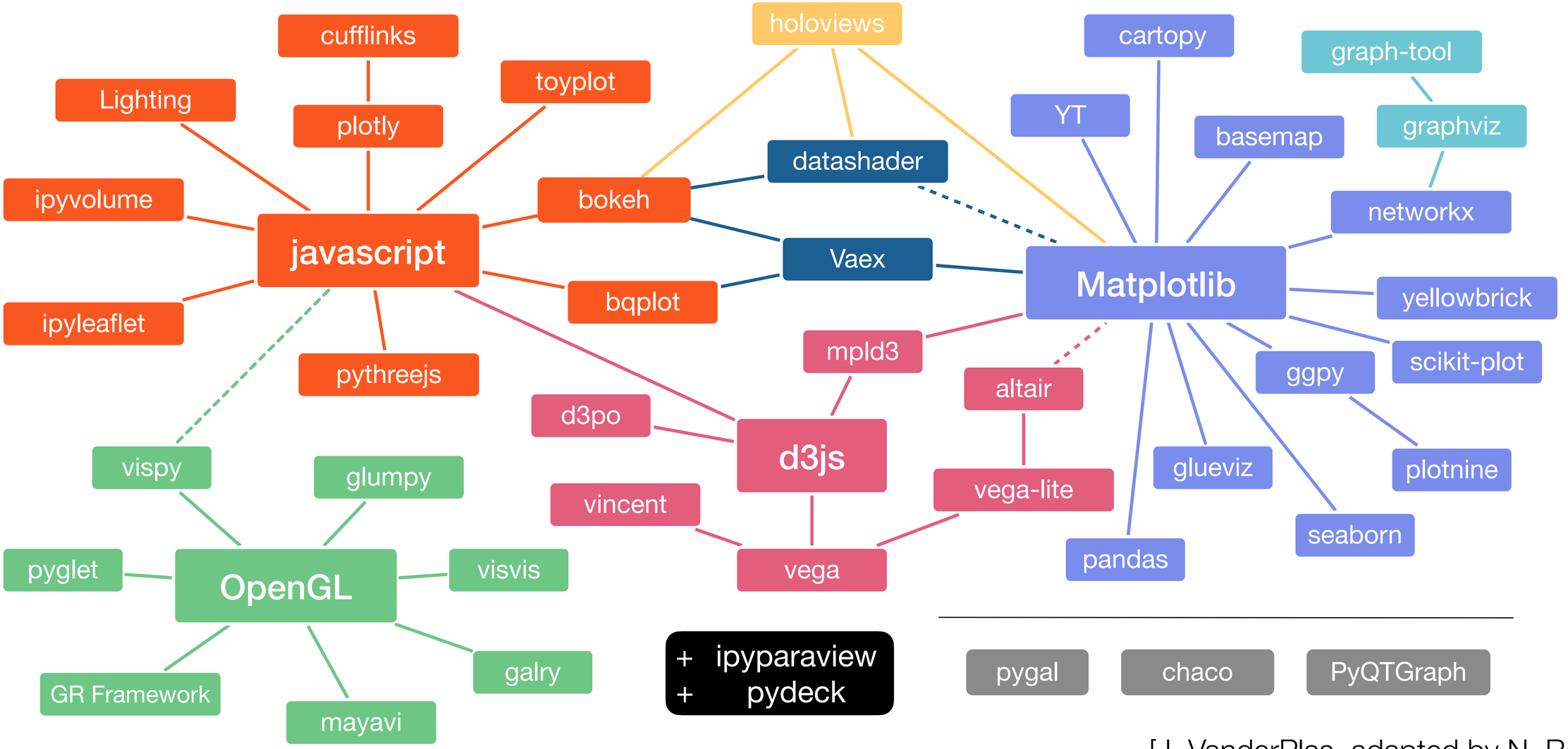
[C. G. Healey]

The Python Visualization Landscape



[J. VanderPlas, adapted by N. Rougier]

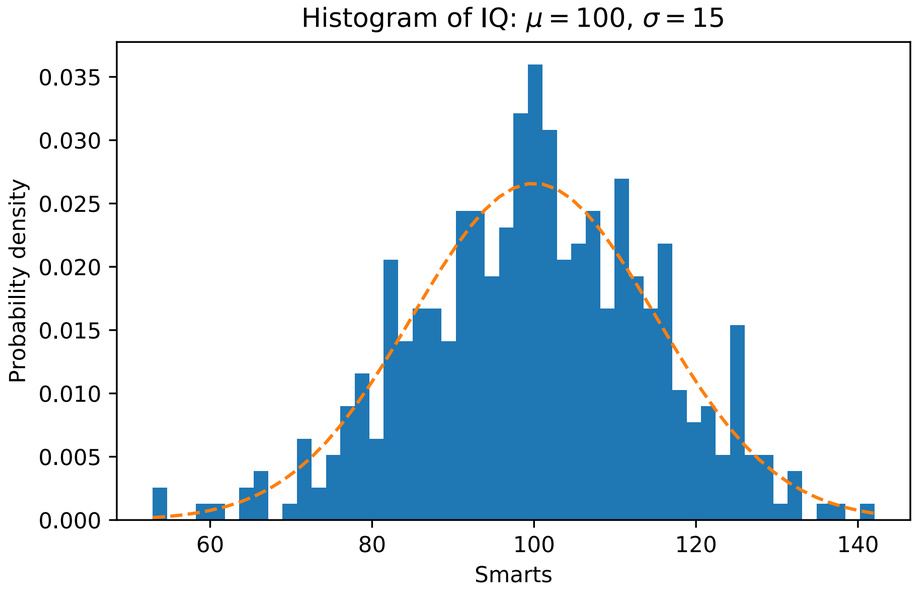
The Python Visualization Landscape



[J. VanderPlas, adapted by N. Rougier]

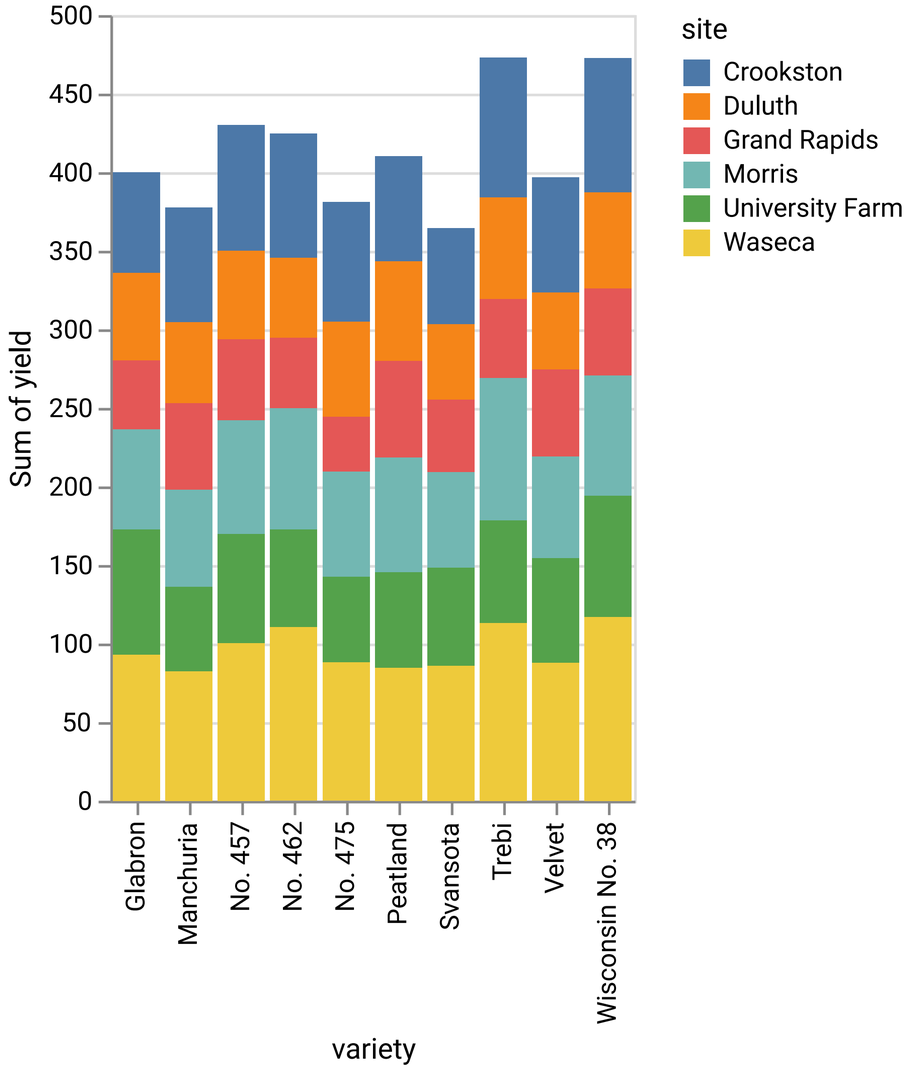
matplotlib

- 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

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

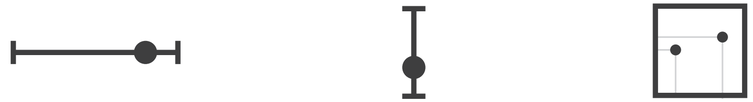
string	description
'p'	pentagon
'P'	plus (filled)
'*'	star
'h'	hexagon1
'H'	hexagon2
'+'	plus
'x'	x
'X'	x (filled)
'D'	diamond
'd'	thin_diamond
'|'	vline
'_'	hline

Markers

[Documentation (Notes Section)]

Data is Encoded via Visual Channels

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



- **Color**

- **Shape**

- **Tilt**

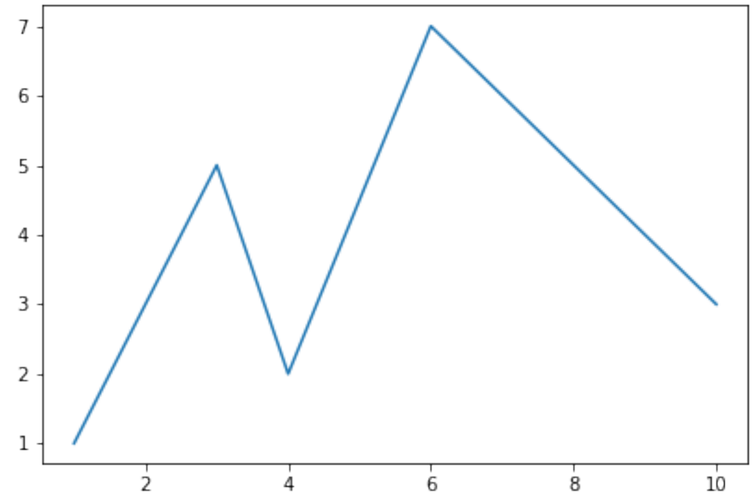
- **Size**
  - Length
  - Area
  - Volume

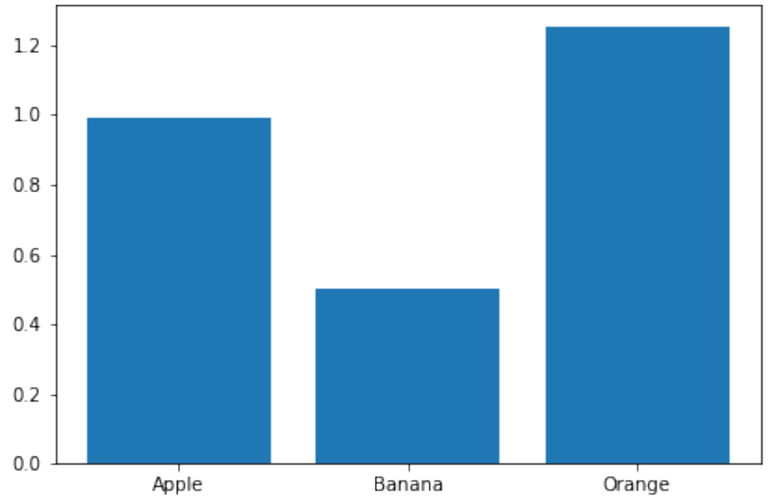
[Munzner (ill. Maguire), 2014]

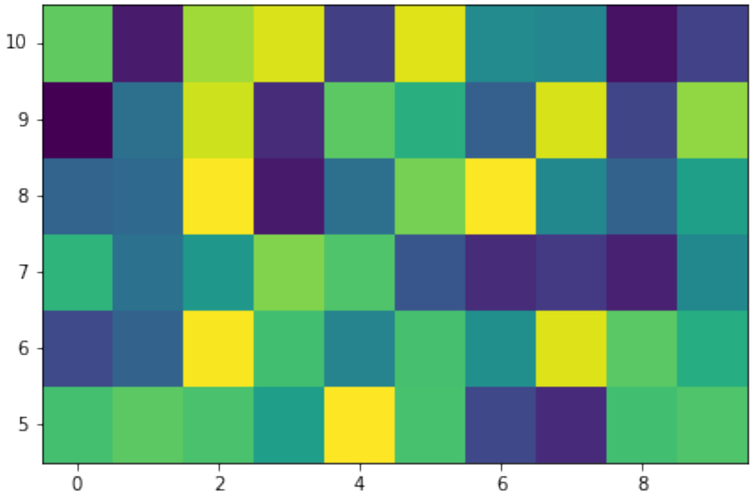
Encoding Data Attributes via Channels

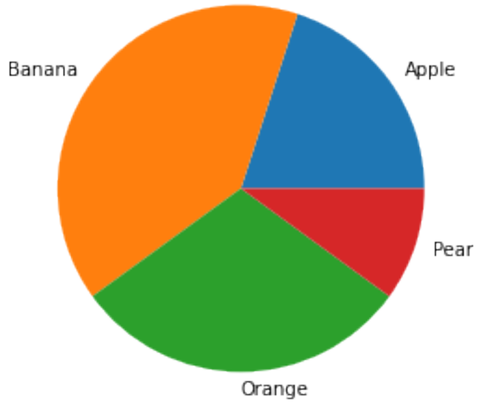
-
```
data = {'age': [1,3,4,6,10],
        'num_jumps': [1,5,2,7,3],
        'weight': [20,50,25,55,25],
        'num_scoops': [3,2,4,2,3]}
plt.scatter('age','num_jumps',c='num_scoops',s='weight',
            data=data)
```
- `data` is a dictionary that contains information about each data item (first 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'])
```

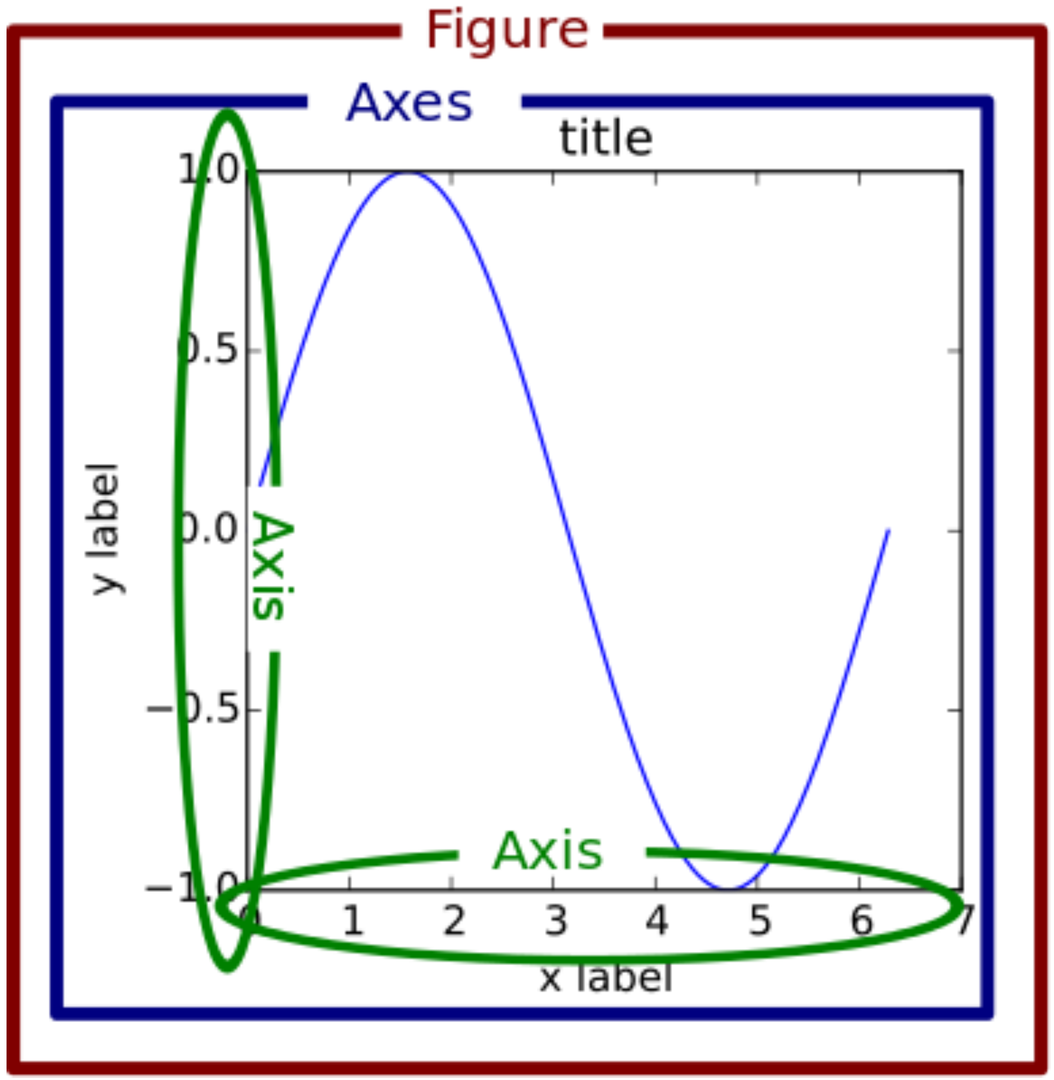
matplotlib tutorials

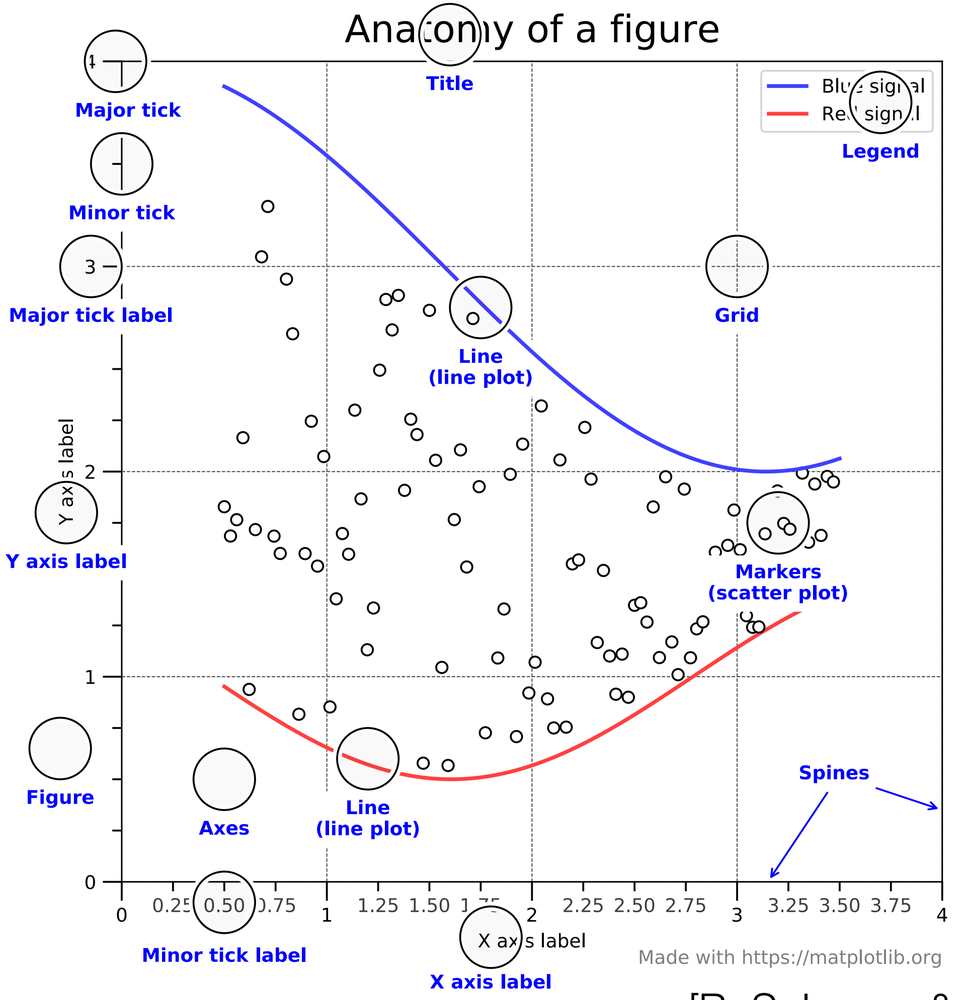
- https://matplotlib.org/stable/tutorials/index.html
- https://github.com/rougier/matplotlib-tutorial

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'])
  ```
  ```

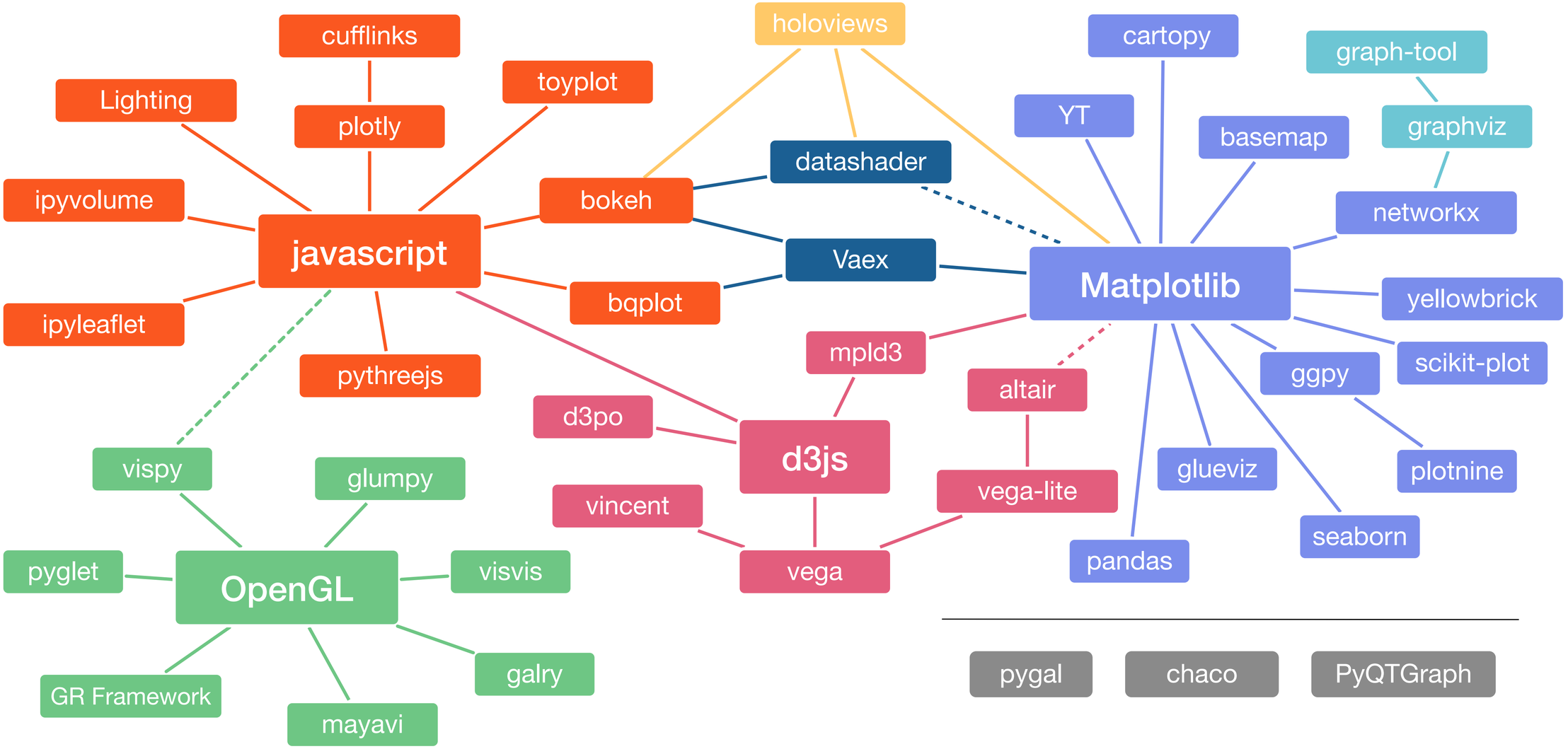
pandas Integration

- 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))
```

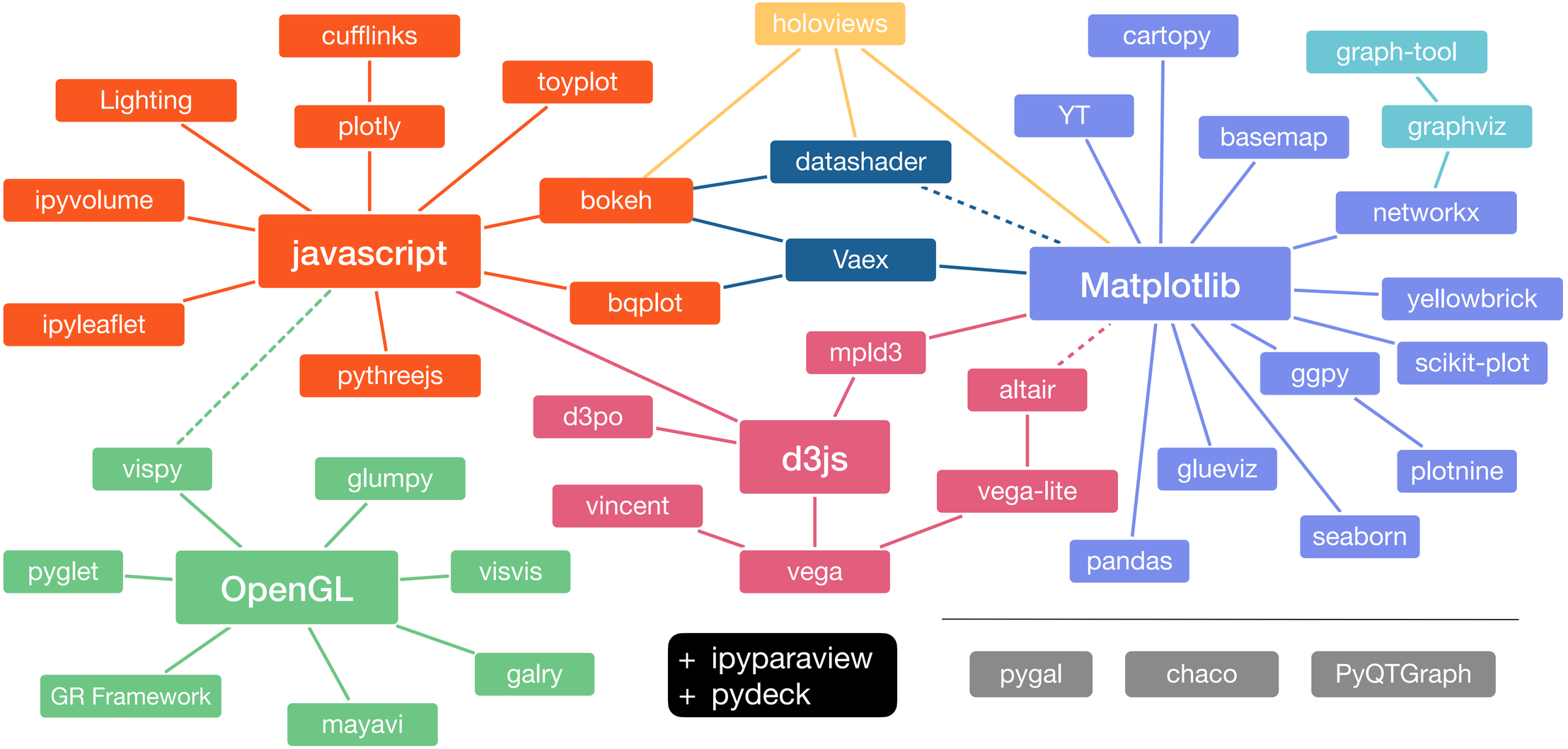
Extensions & Other Directions

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

The Python Visualization Landscape



The Python Visualization Landscape



History of Vega-Lite & 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

- Altair: Python interface to Vega-Lite (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

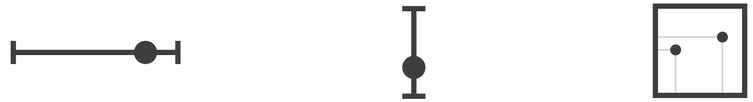
- **Marks** are the basic graphical elements in a visualization
- Marks classified by dimensionality:

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**

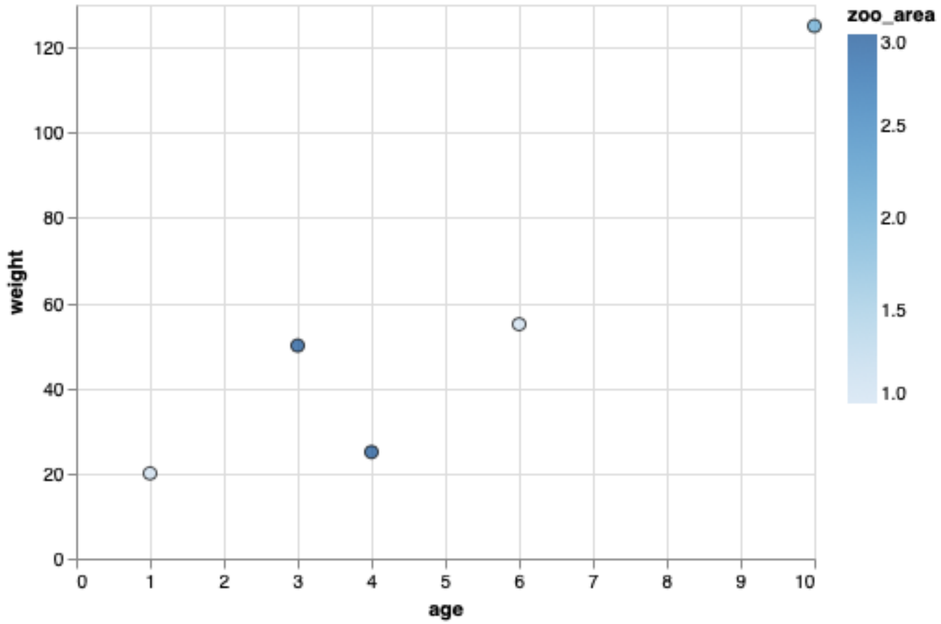
- **Tilt**

- **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!

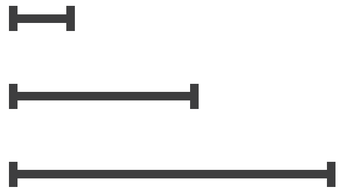
Data Attributes and Altair Types

➔ Categorical

➔ Ordered

➔ *Ordinal*

➔ *Quantitative*



[Munzner (ill. Maguire), 2014]

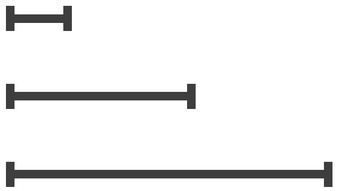
Data Attributes and Altair Types

➔ Categorical

➔ Ordered

➔ *Ordinal*

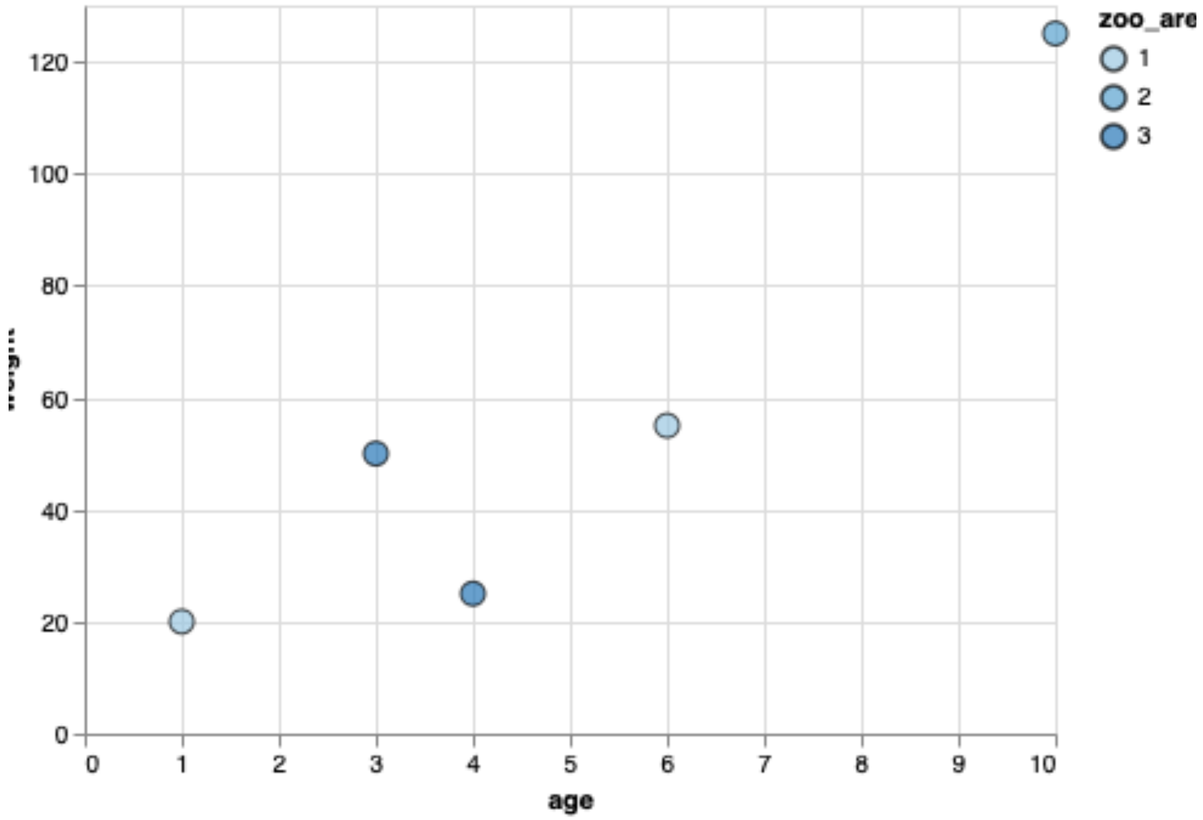
➔ *Quantitative*



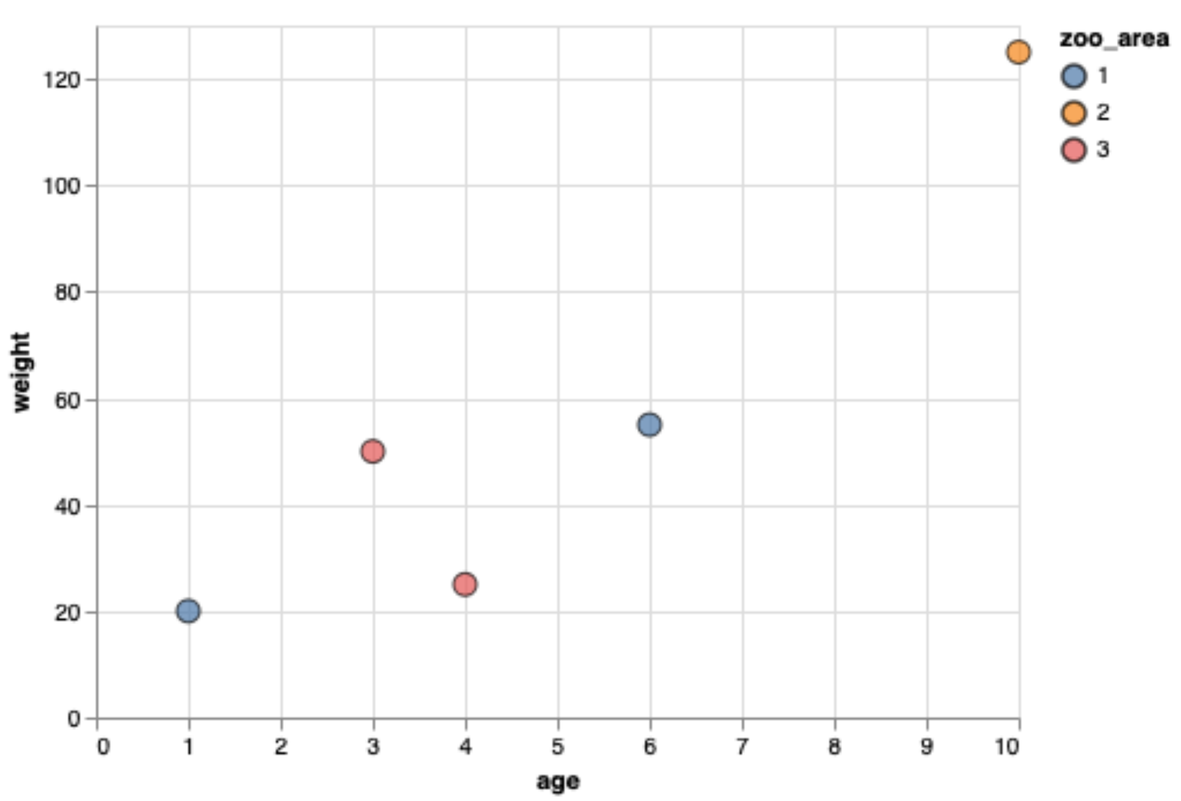
- Categorical data = Nominal (N)
- Ordinal data = Ordinal (O)
- Quantitative data = Quantitative (Q)
- Temporal data = Temporal (T)

[Munzner (ill. Maguire), 2014]

Specifying the Type



`zoo_area:O`



`zoo_area:N`

Different Channels for Different Attribute Types

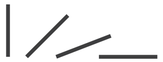
Magnitude Channels: Ordered Attributes

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

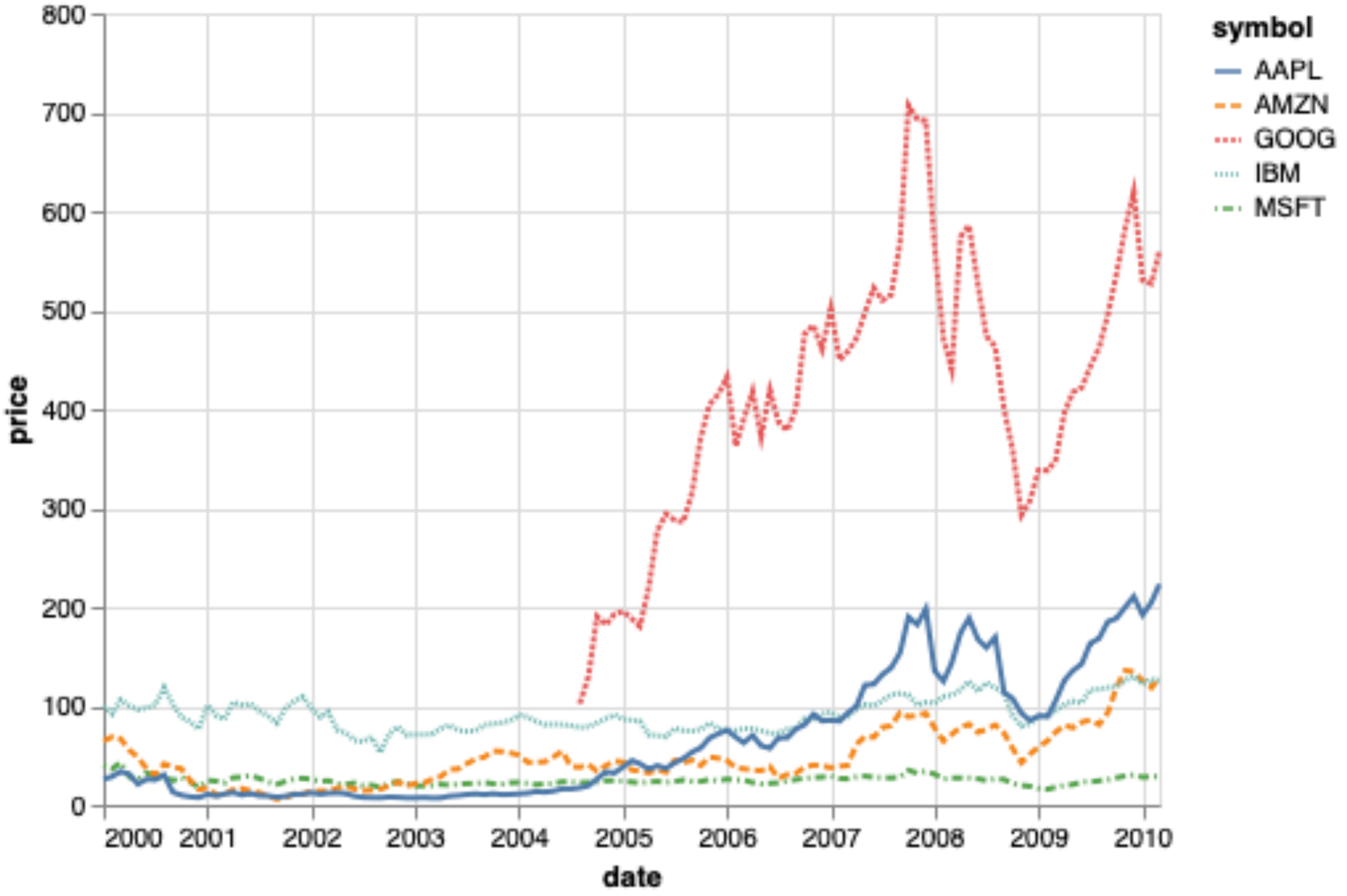
Motion

Shape

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

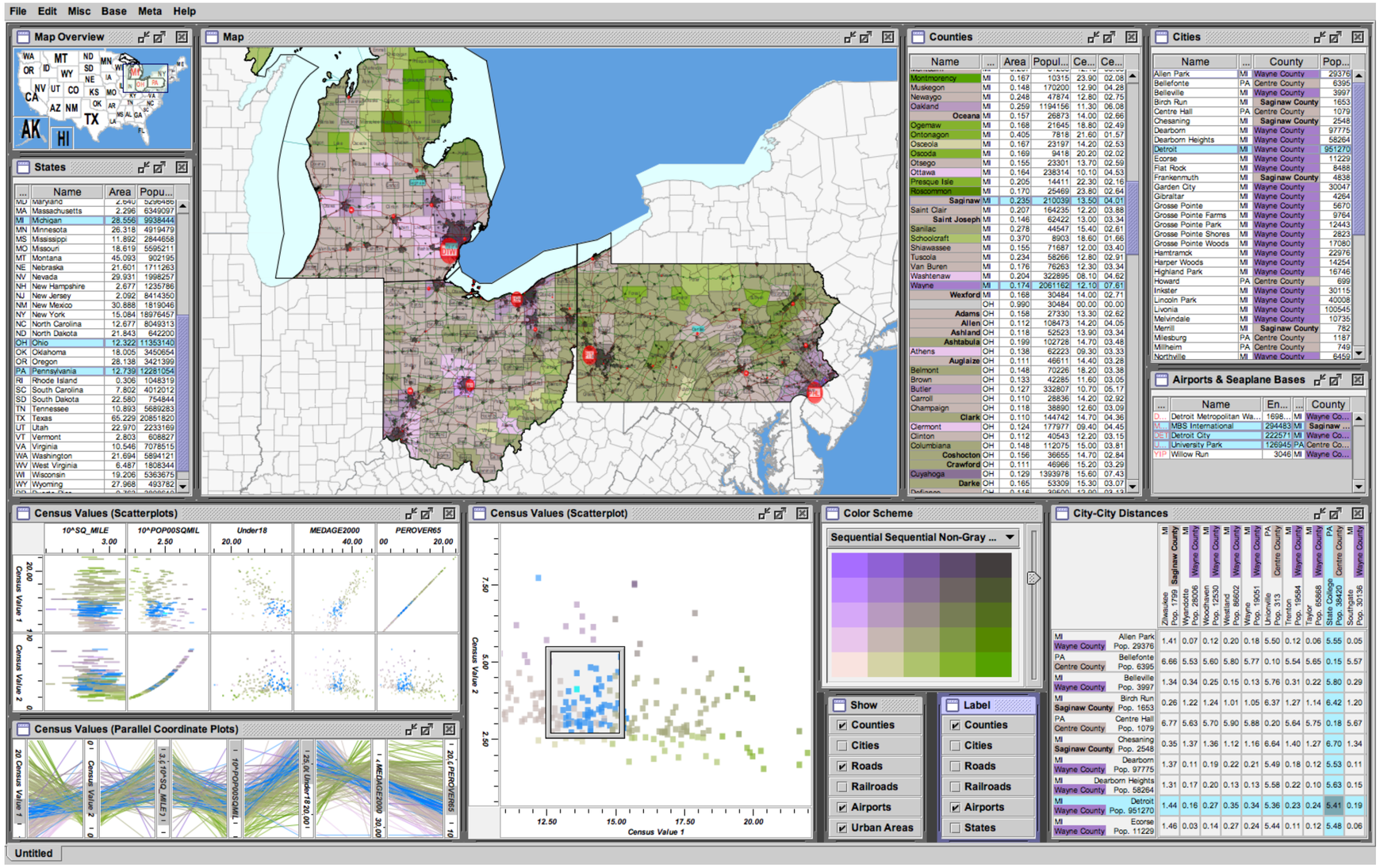
[Munzner (ill. Maguire), 2014]

Multiple Views in Visualization



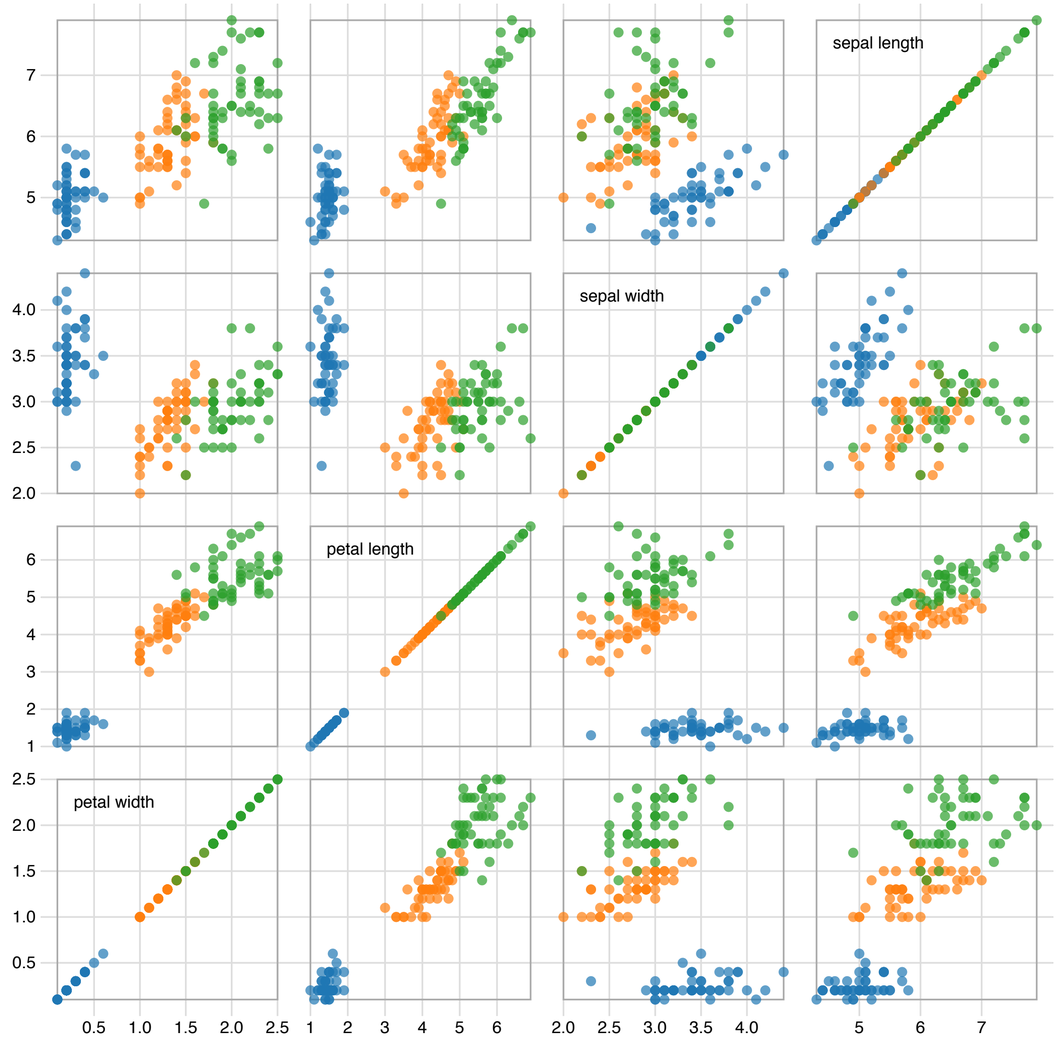
[Altair]

Multiple Views in Visualization



[Improvise, Weaver, 2004]

Multiple Views in Visualization

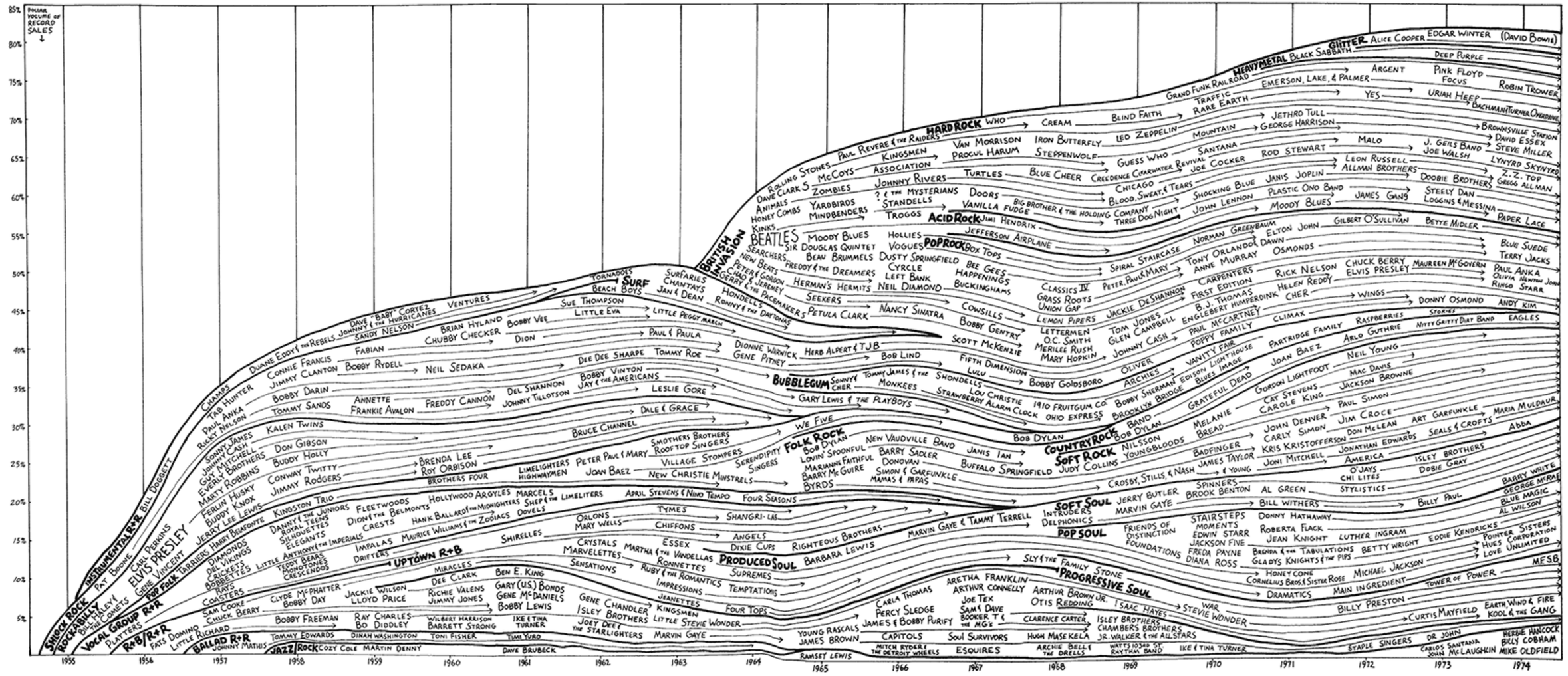


[M. Bostock]

Altair Supports Concatenation, Layering, & Repetition

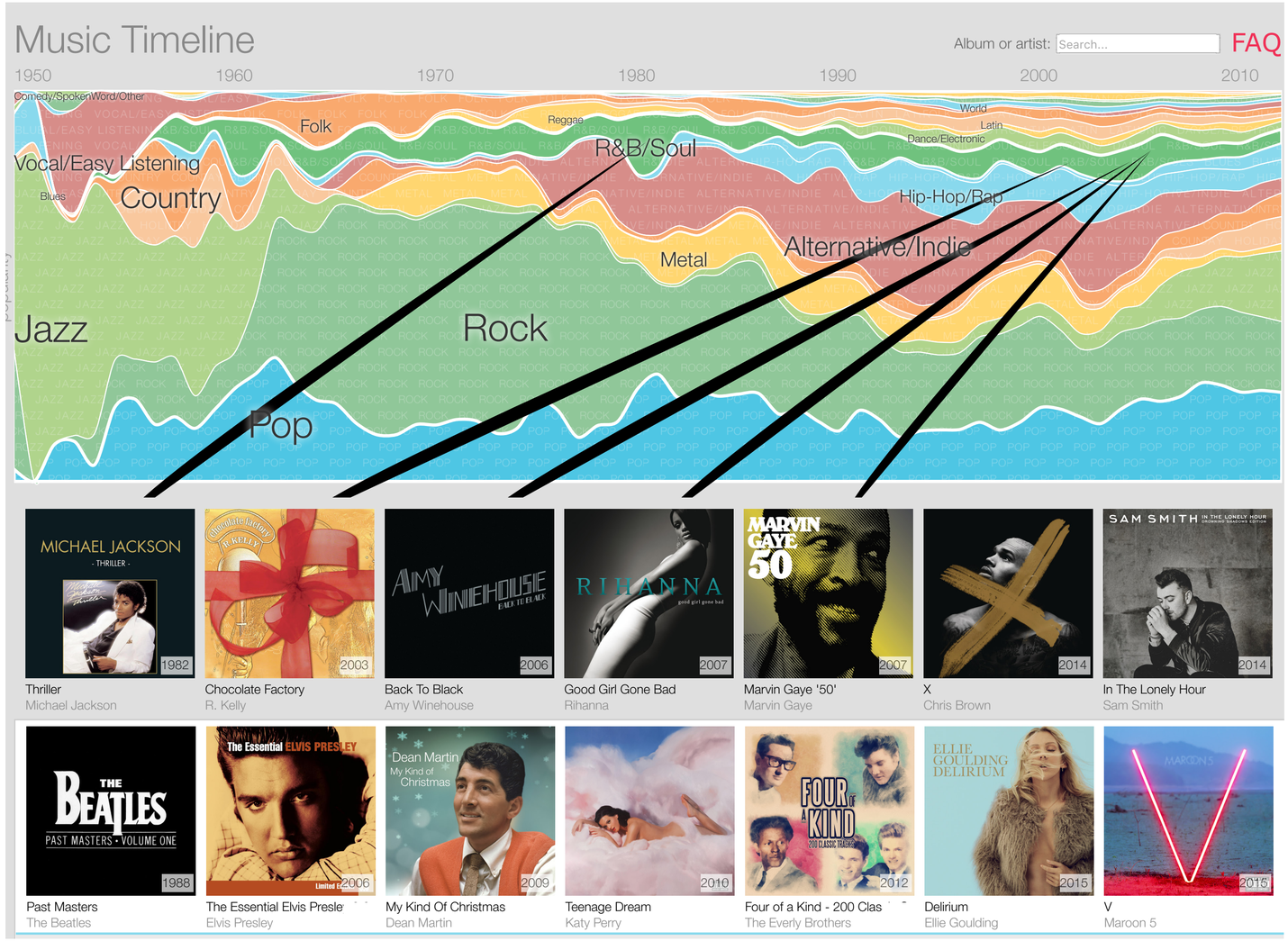
- Layering:
  - + Operator
- Concatenation:
  - Horizontal: | operator
  - Vertical: & operator
- Repetition
  - Use of .repeat for layout
  - Reference repeated variables in the encoding

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

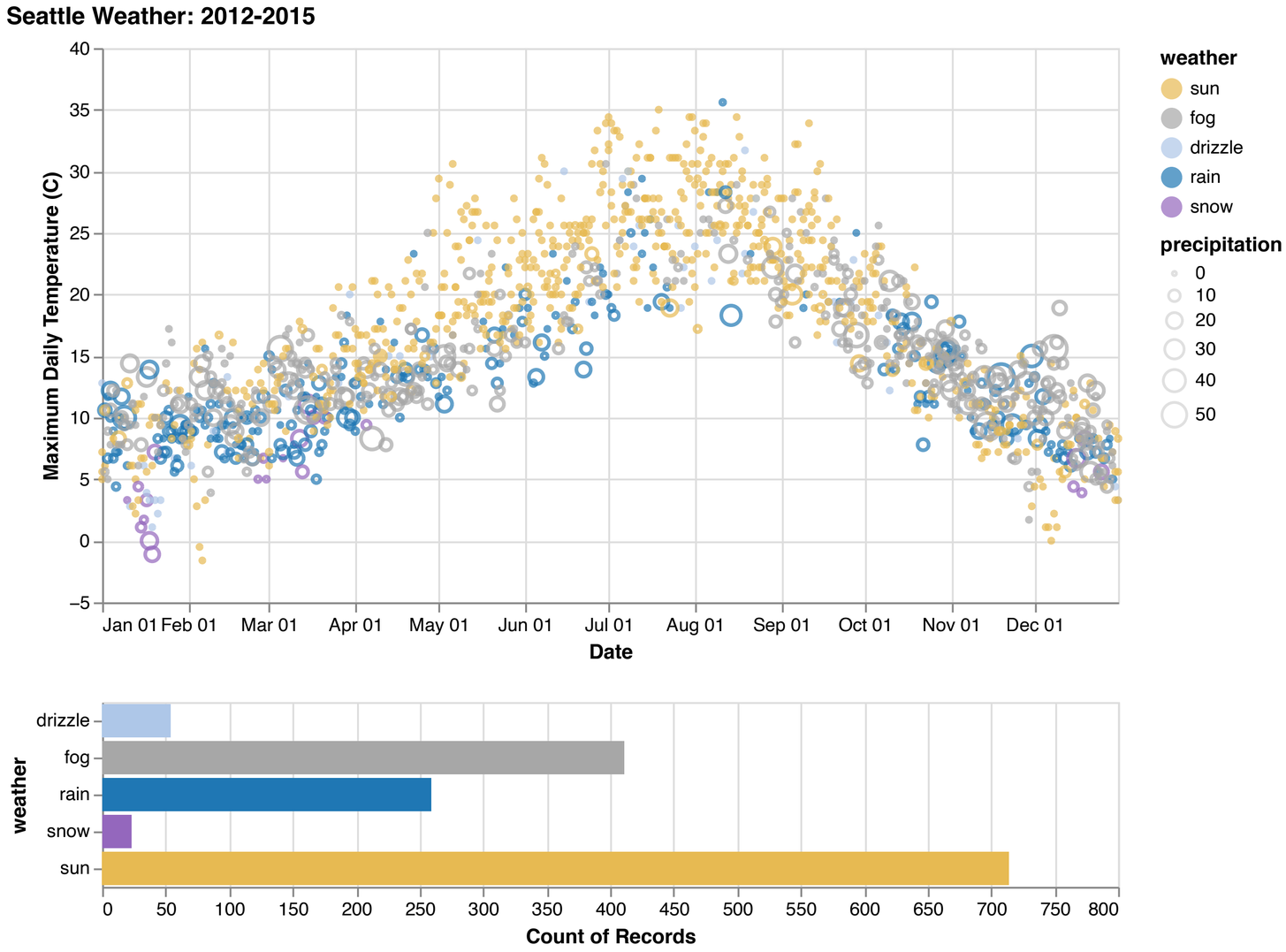
Highlighting

- 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

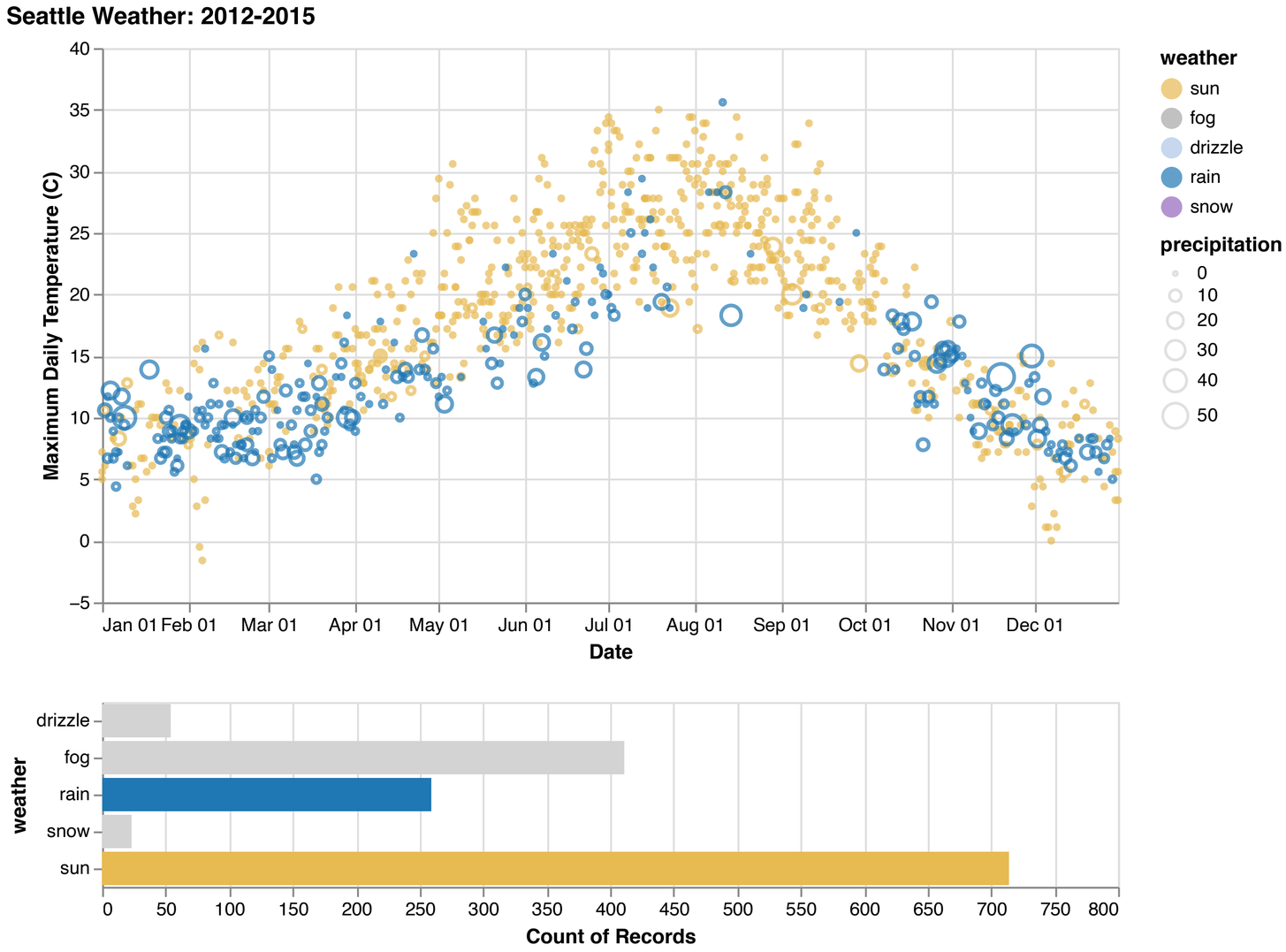
Altair's Interactive Charts

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

Interaction



Weather Selection: Rain vs. Sun



Date Selection: July-September Sun