Programming Principles in Python (CSCI 503/490)

Data Visualization

Dr. David Koop

Quiz

DataFrame Filtering

- polars:
  - `df['pop'] > 2 # boolean Series`
  - `df.filter(pl.col('pop') > 2) # subset of dataframe`
- pandas:
  - `dfa['pop'] > 2 # boolean Series`
  - `dfa[dfa['pop'] > 2] # subset of dataframe`
  - `dfa.query('pop > 2') # subset of dataframe`
- Multiple criteria, use `&`, `|`, and `~`; remember parentheses!
  - `df.filter((pl.col('year') < 2002) & (pl.col('pop') > 2))`
  - `dfa[(dfa['year'] < 2002) & (dfa['pop'] > 2)]`

Dataframe Statistics

- describe: overview (difference between numeric and non-numeric data
- unique: unique values (array in pandas!)
- value_counts: counts for each value
- min, max, median…

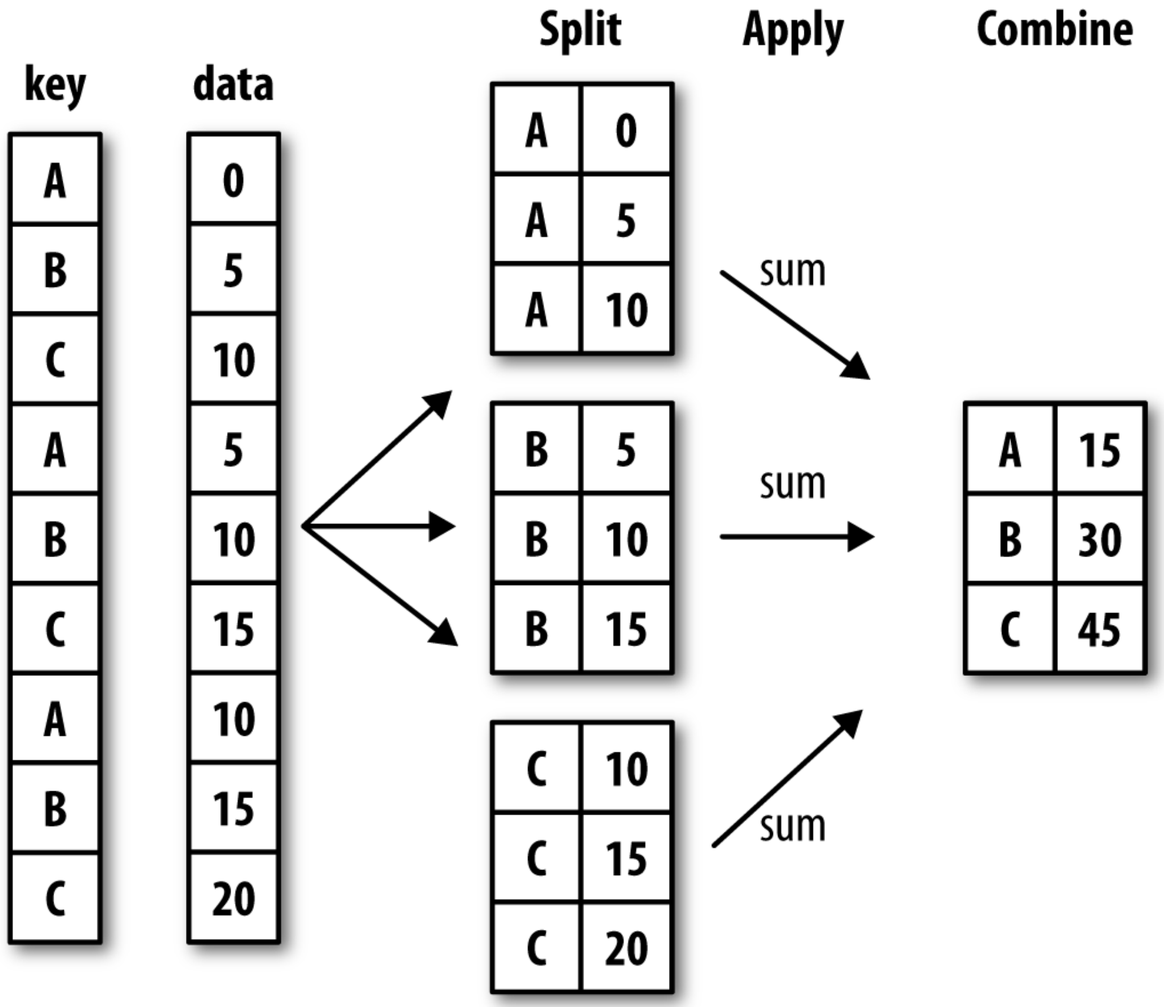
Derived Data

- 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
    ```

Assignment 8

- Out Soon…
- Last Assignment
- Data and Visualization

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'))`

Split-Apply-Combine

- Similar to Map (split+apply) Reduce (combine) paradigm
- The Pattern:
  1. **Split** the data by some grouping variable
  2. **Apply** some function to each group independently
  3. **Combine** the data into some output dataset
- The apply step is usually one of:
  - Aggregate
  - Transform
  - Filter

[T. Brandt]

Group By

- Polars: `group_by`, Pandas: `groupby`
- `group_by` method creates a `GroupBy` object
- `group_by` **does not compute** anything until there is an aggregate step
- Sizes of groups:
  - `df.group_by('Island').agg(pl.len()) # DataFrame`
  - `dfa.groupby('Island').size() # Series`
- Can iterate through the groups (names and dataframes):
  -
```
for name, gdf in df.group_by('Island'):
    display(name, gdf)
```

Aggregation

- Single Column:
  - `df.group_by('Island').agg(pl.col('Length (mm)').mean())`
  - `dfa.groupby('Island')['Length (mm)'].mean()`
- pandas returns a Series, polars returns a DataFrame
- List of Values:
  - `df.group_by('Island').agg(pl.col('Length (mm)'))`
  - `dfa.groupby('Island')['Length (mm)'].apply(list)`

Aggregation (Multiple Columns)

- Multiple columns in an aggregation
  - `df.group_by('Island').agg(pl.col('Length','Depth').mean())`
  - `dfa.groupby('Island')[['Length','Depth']].mean()`
- Multiple aggregations for a column
  - `df.group_by('Island').agg(pl.col('Length').min().alias('LMin'), pl.col('Length').max().alias('LMax'))`
  - `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]

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

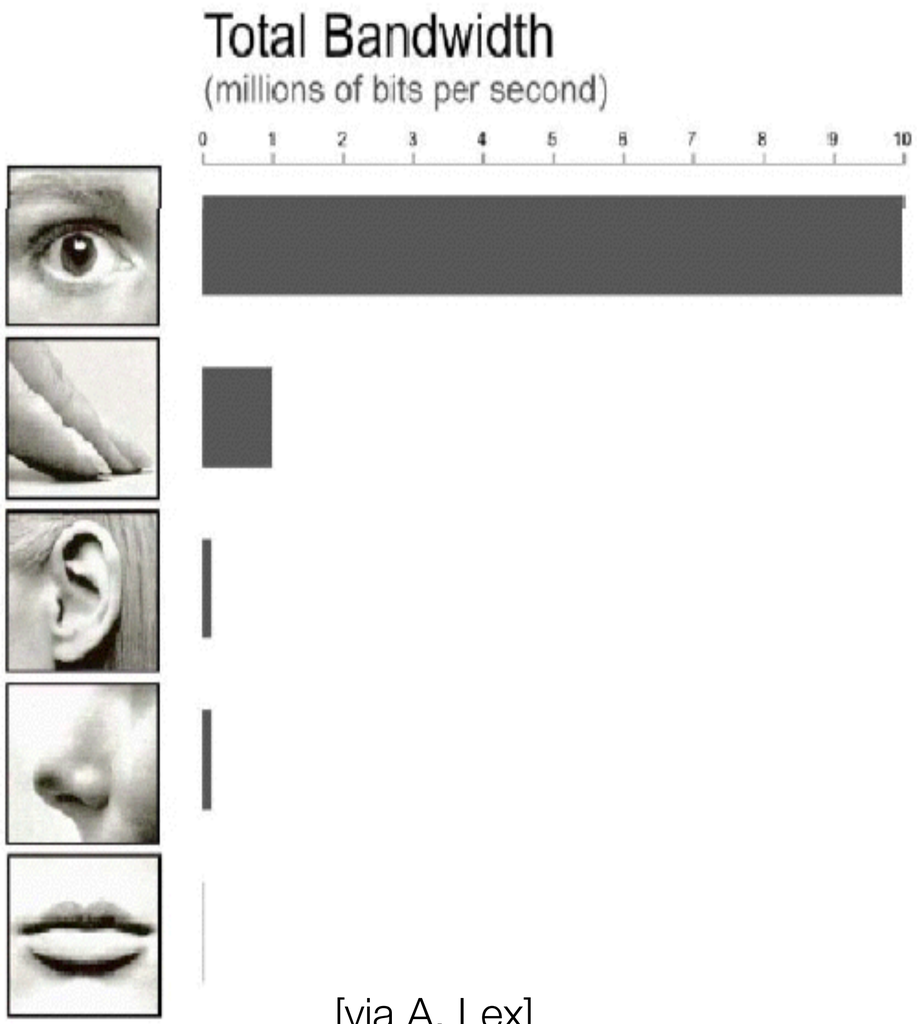
- Python has datetime library to support dates and times
- pandas has a Timestamp data type that functions somewhat similarly
- polars has a Datetime data type that functions somewhat similarly
- Can convert timestamps
  - `pl.to_datetime` and `pl.str.to_datetime`: directed, require format
  - `pd.to_datetime`: 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`

Definition of Visualization

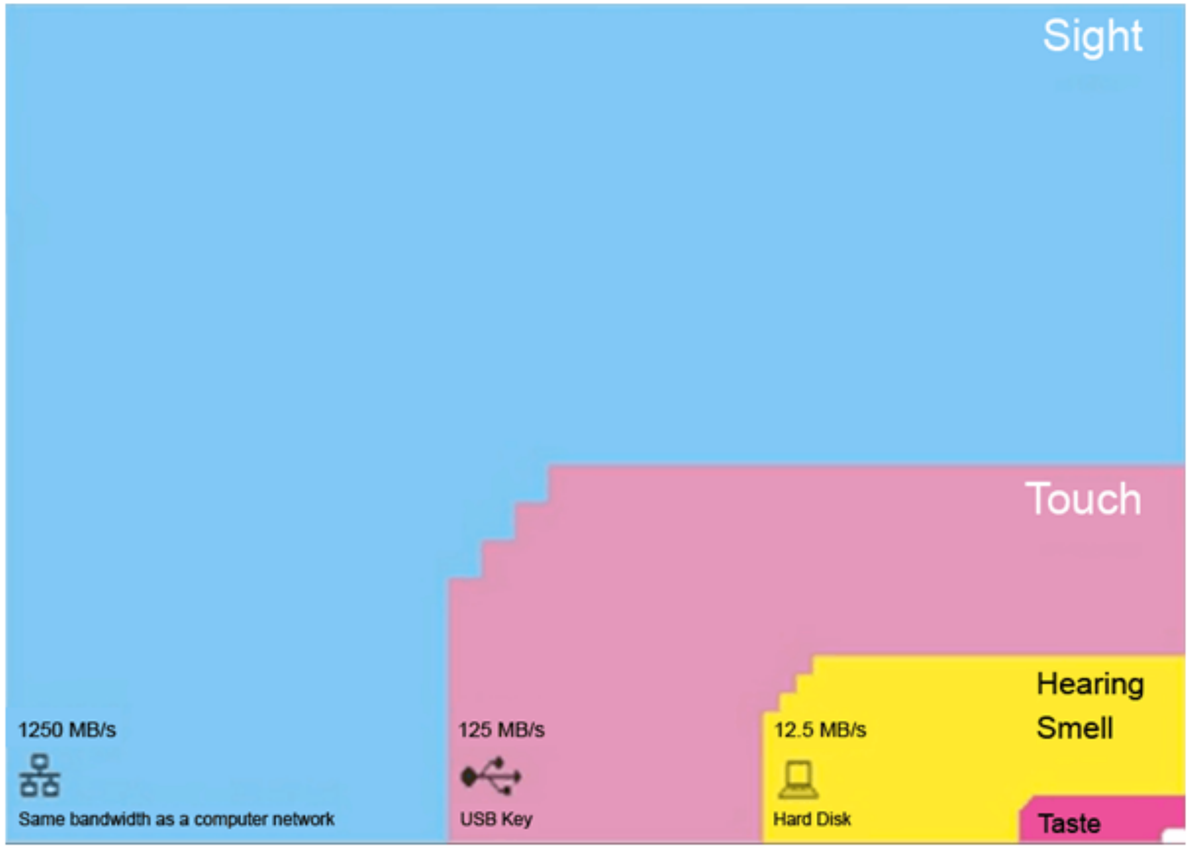
“Computer-based visualization systems provide visual representations of datasets designed to help people carry out tasks more effectively”

— 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]

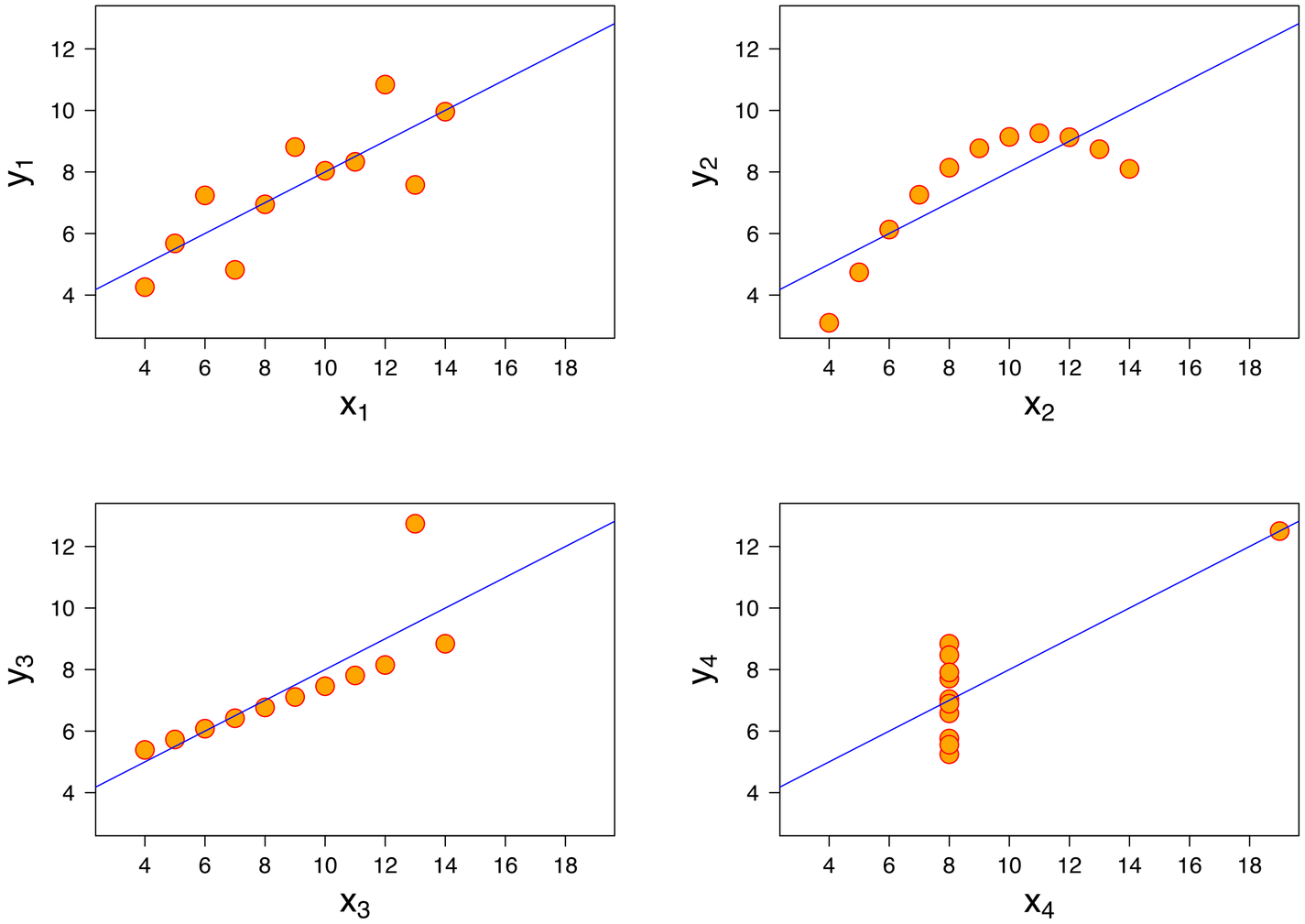
Why Visual?

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

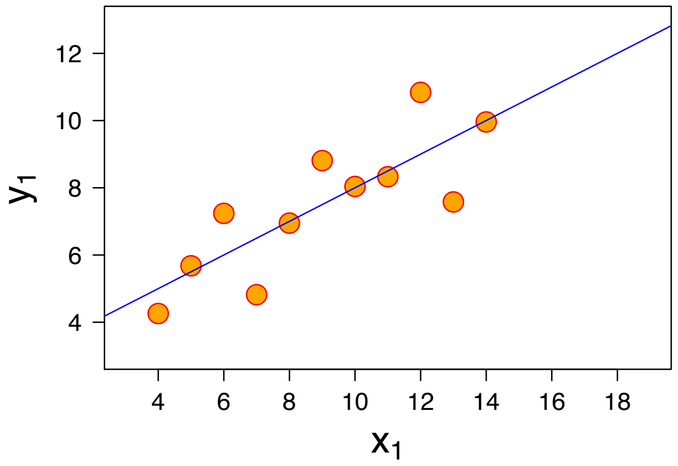
[F. J. Anscombe]

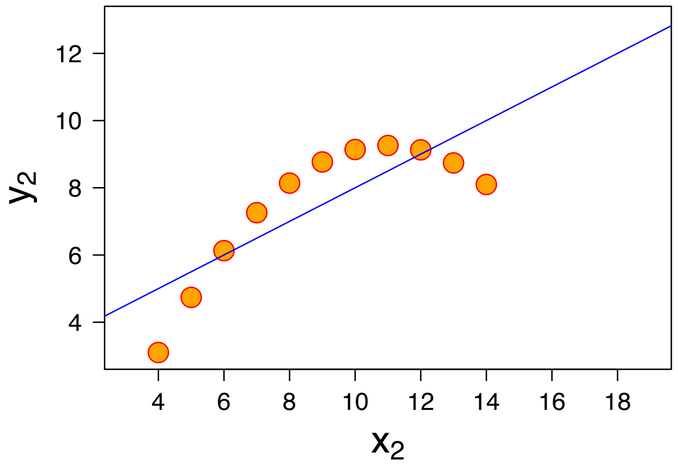
Why Visual?

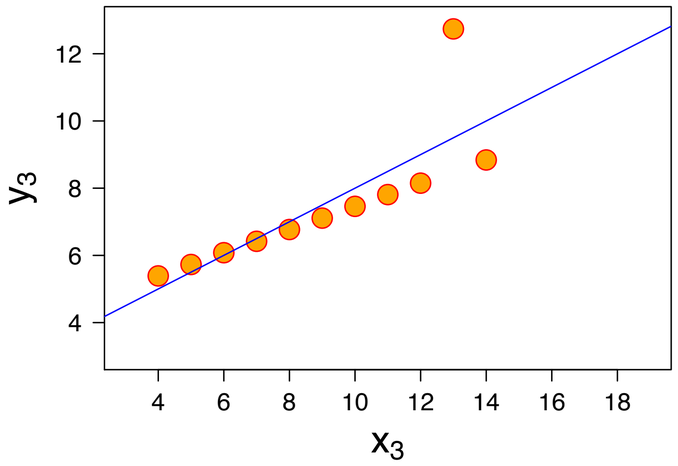


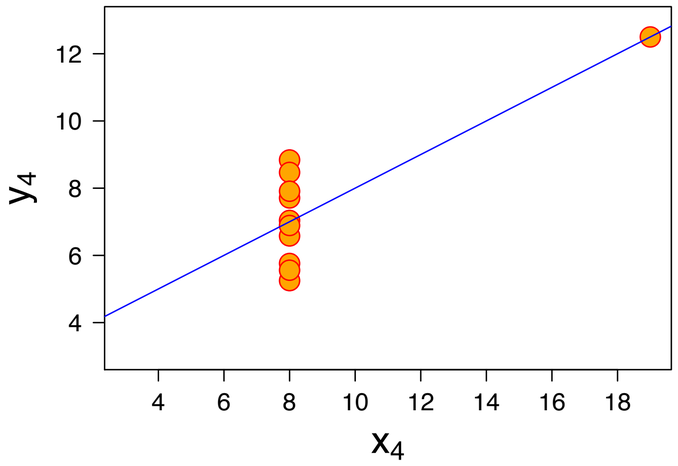
[F. J. Anscombe]

Why Visual?









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

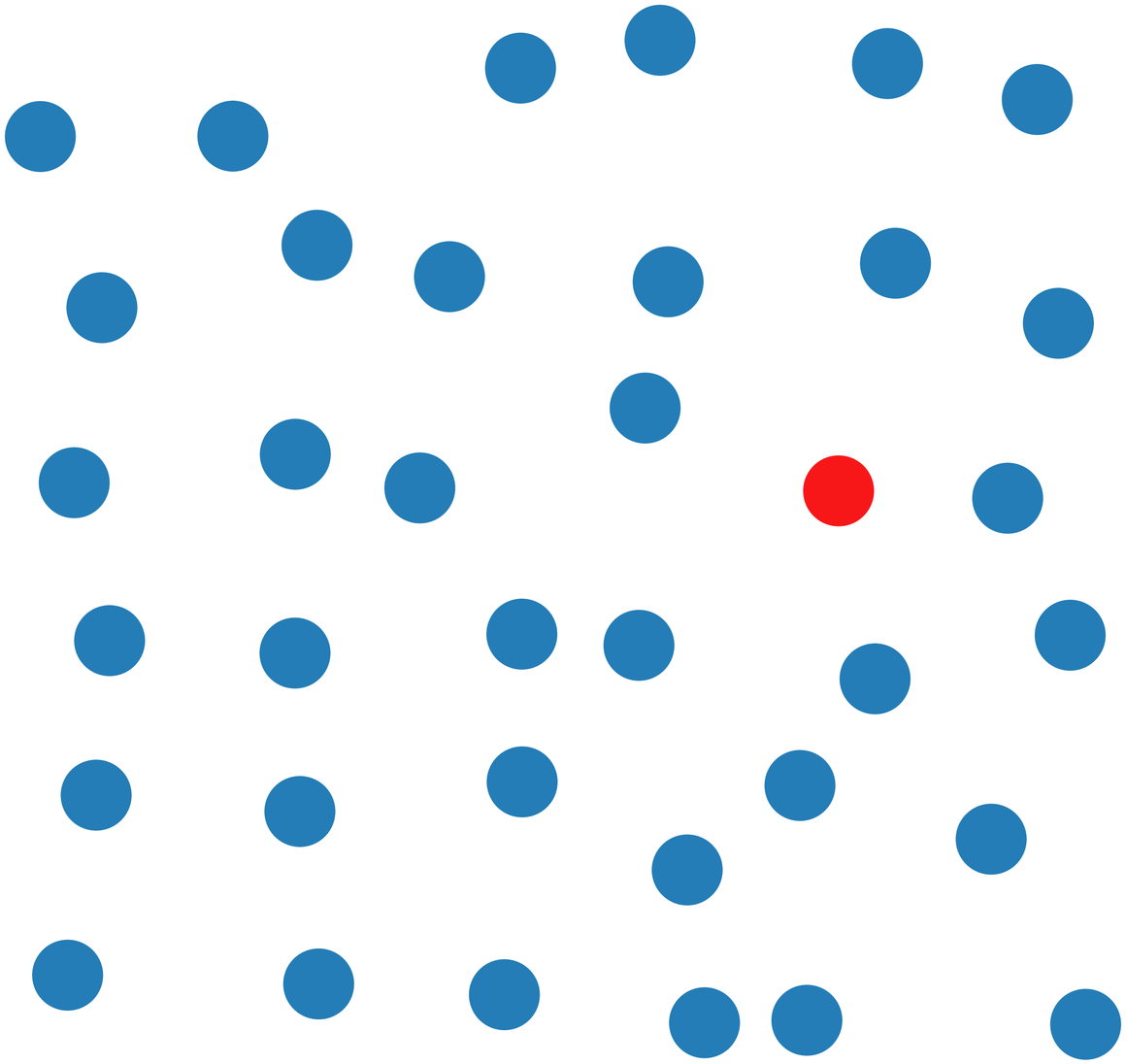
[F. J. Anscombe]

Visualization Goals

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

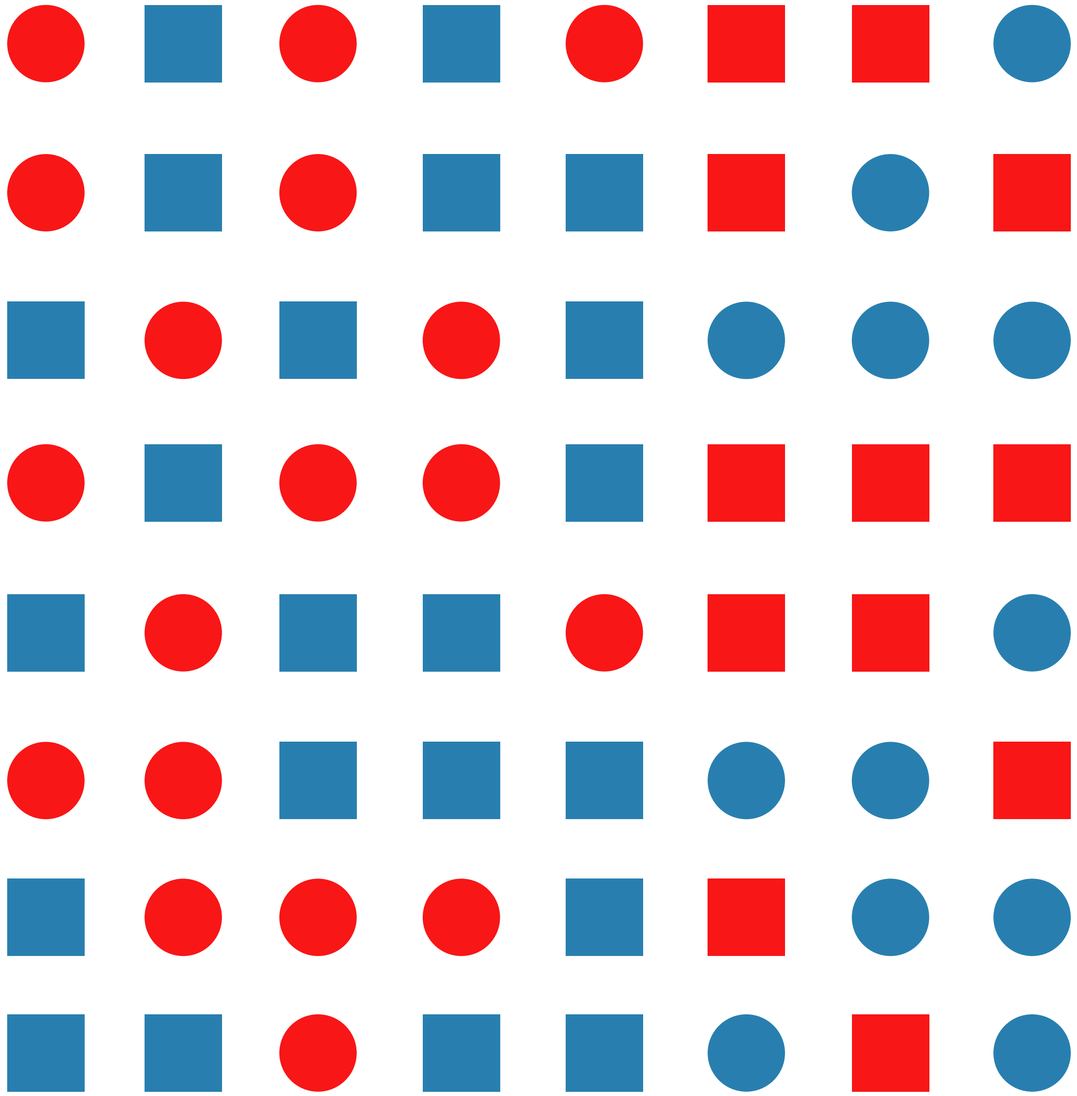
- Identify patterns, trends
- Spot outliers
- Find similarities, correlation

Visual Pop-out



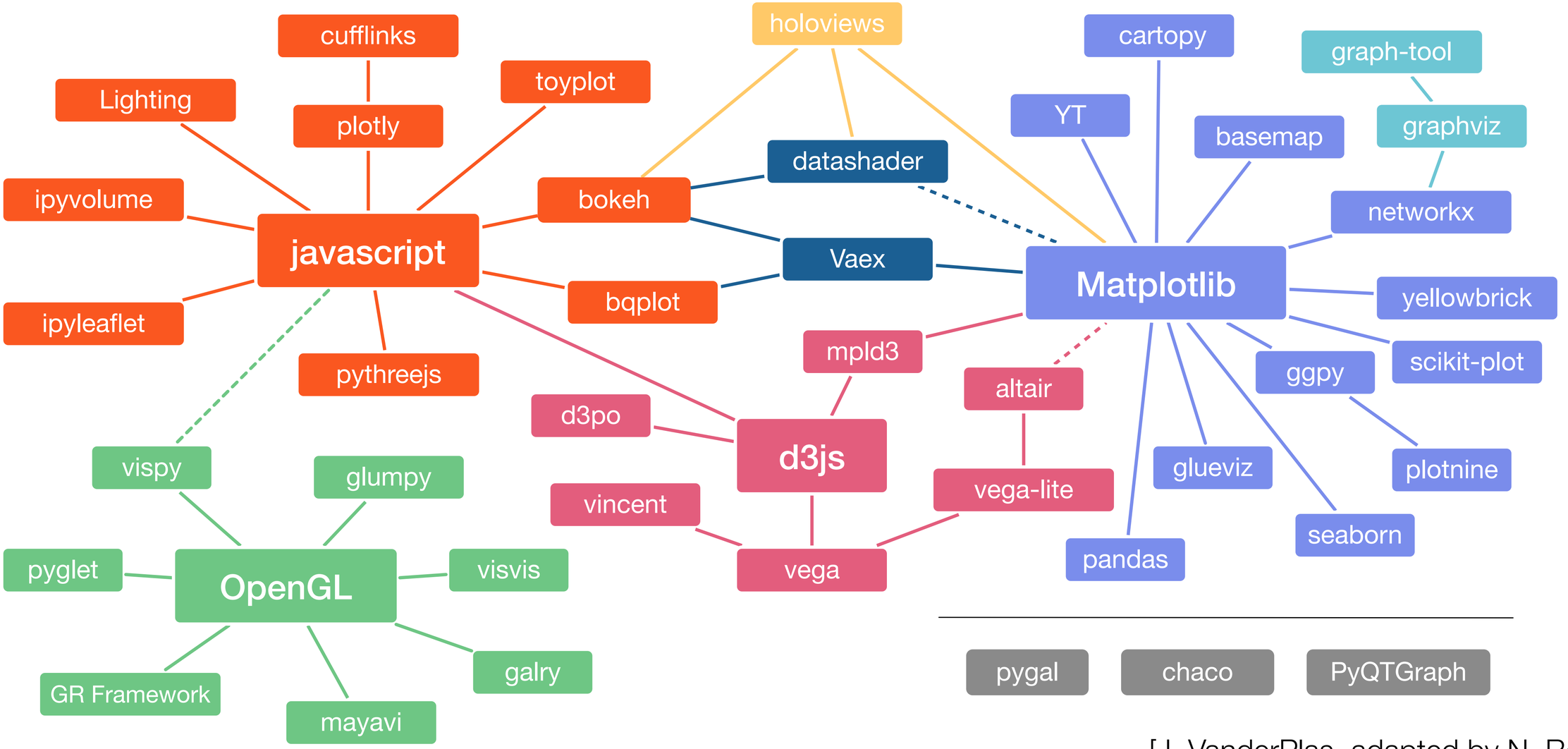
[C. G. Healey]

Visual Perception Limitations



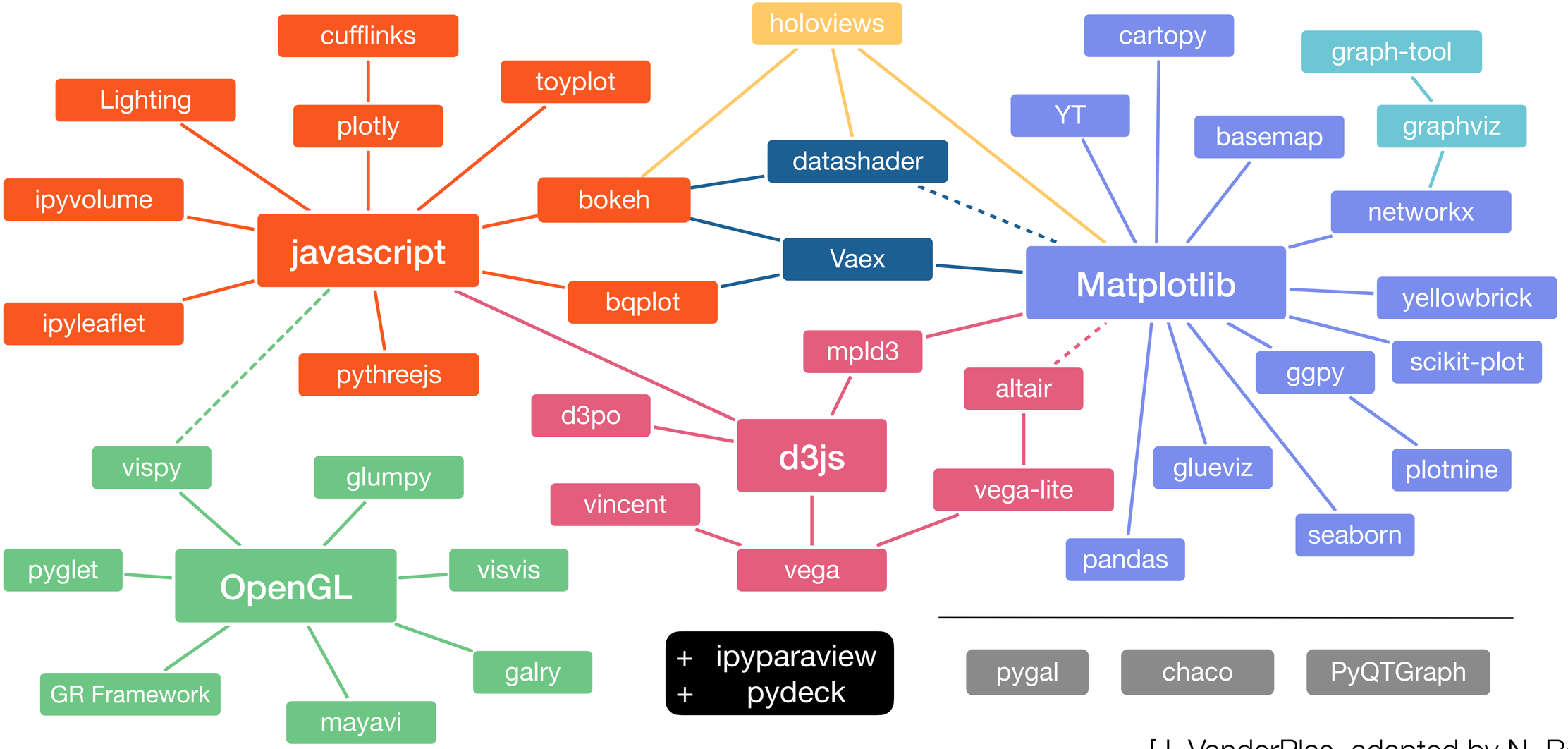
[C. G. Healey]

The Python Visualization Landscape



[J. VanderPlas, adapted by N. Rougier]

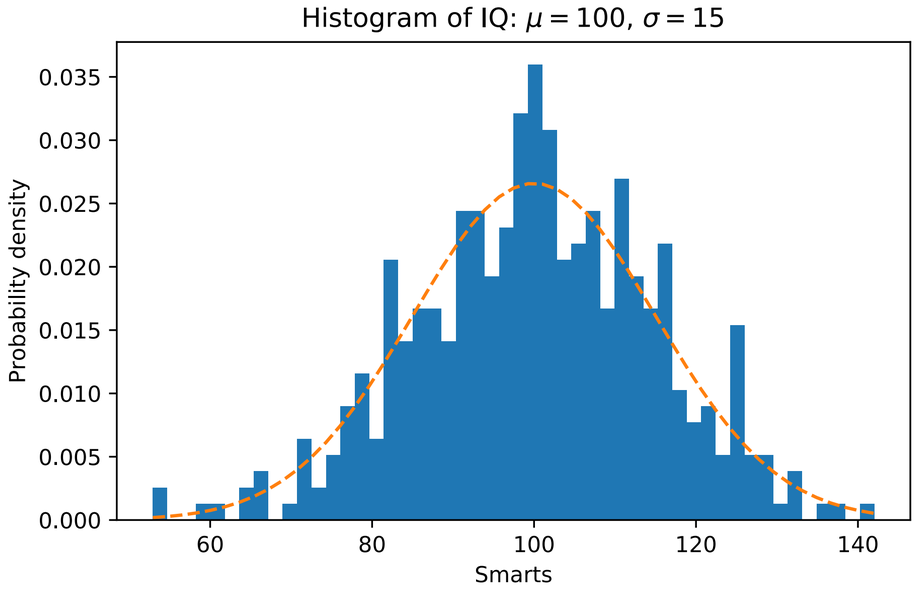
The Python Visualization Landscape



[J. VanderPlas, adapted by N. Rougier]

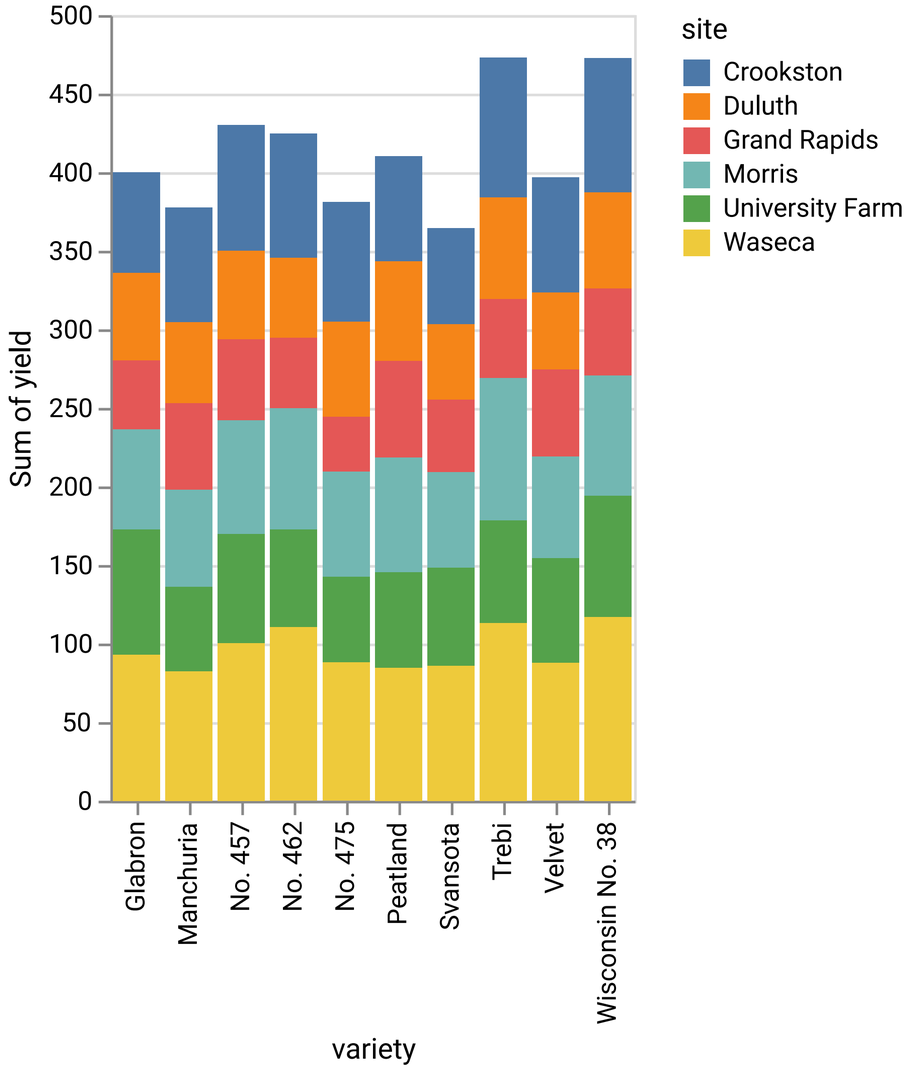
matplotlib

- 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

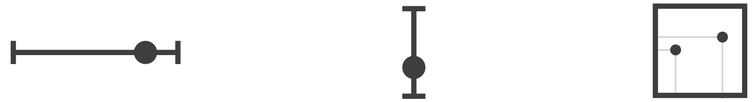
Markers

[Documentation (Notes Section)]

Data is Encoded via Visual Channels

Position

- Horizontal
- Vertical
- Both



Color

Shape

Tilt

Size

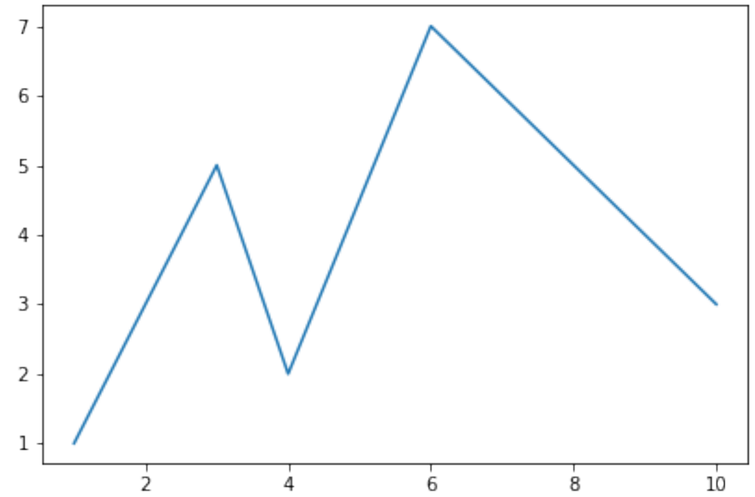
- Length
- Area
- Volume

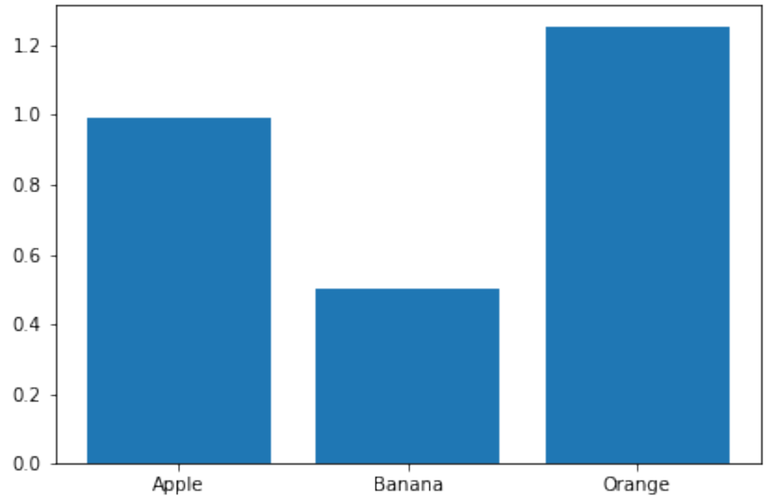
[Munzner (ill. Maguire), 2014]

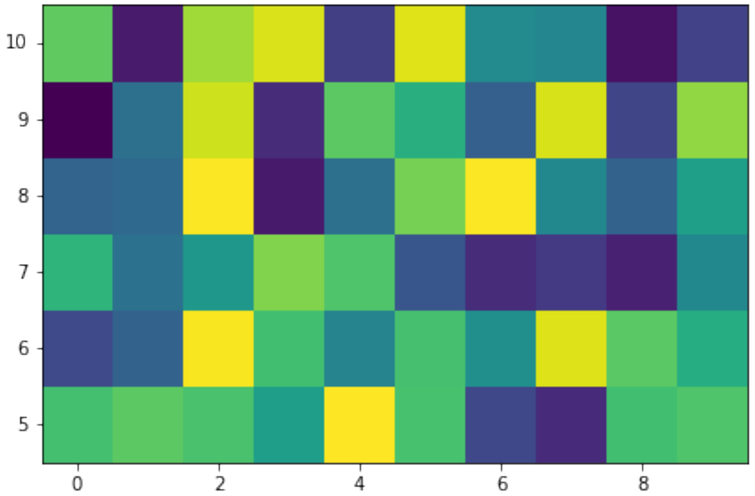
Encoding Data Attributes via Channels

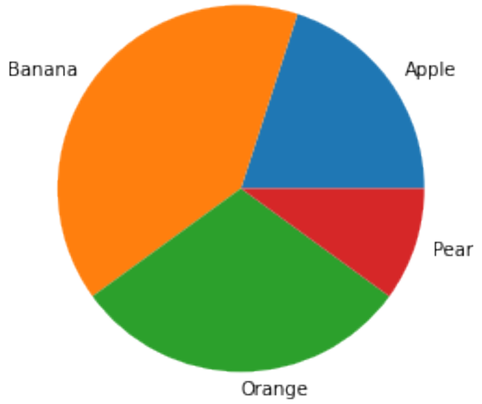
-
```
data = {'age': [1,3,4,6,10],
        'num_jumps': [1,5,2,7,3],
        'weight': [20,50,25,55,25],
        'num_scoops': [3,2,4,2,3]}
plt.scatter('age','num_jumps',c='num_scoops',s='weight',
            data=data)
```
- `data` is a dictionary that contains information about each data item (first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

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 & 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:

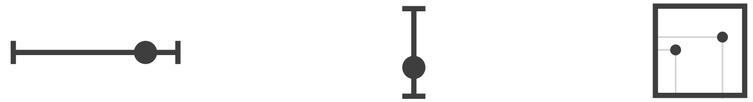
Points **Lines** **Areas**

- 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

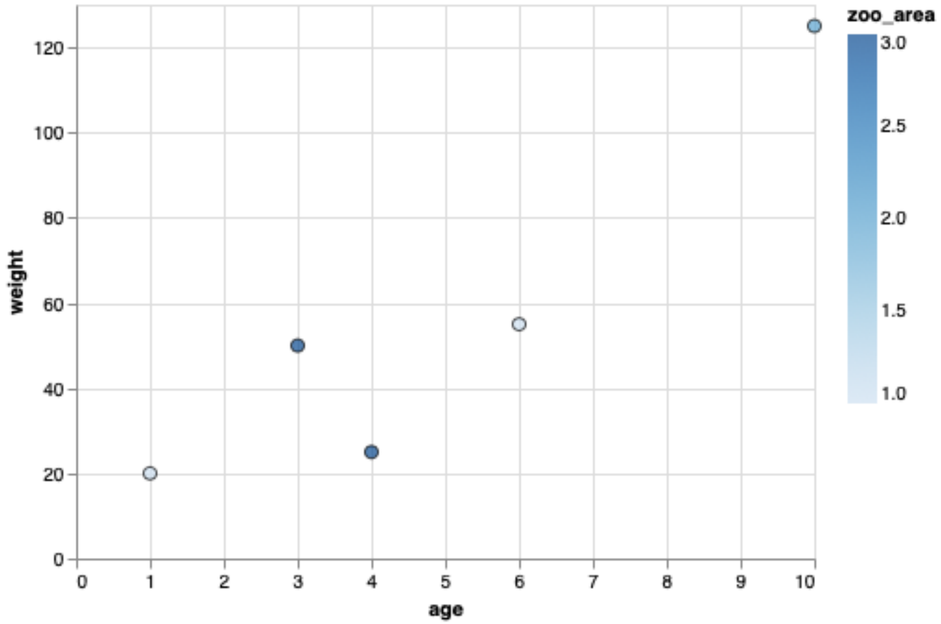
Size

- 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!

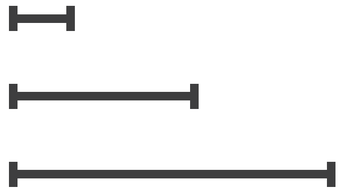
Data Attributes and Altair Types

➔ Categorical

➔ Ordered

➔ *Ordinal*

➔ *Quantitative*



[Munzner (ill. Maguire), 2014]

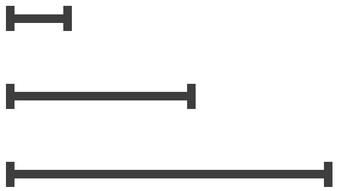
Data Attributes and Altair Types

➔ Categorical

➔ Ordered

➔ *Ordinal*

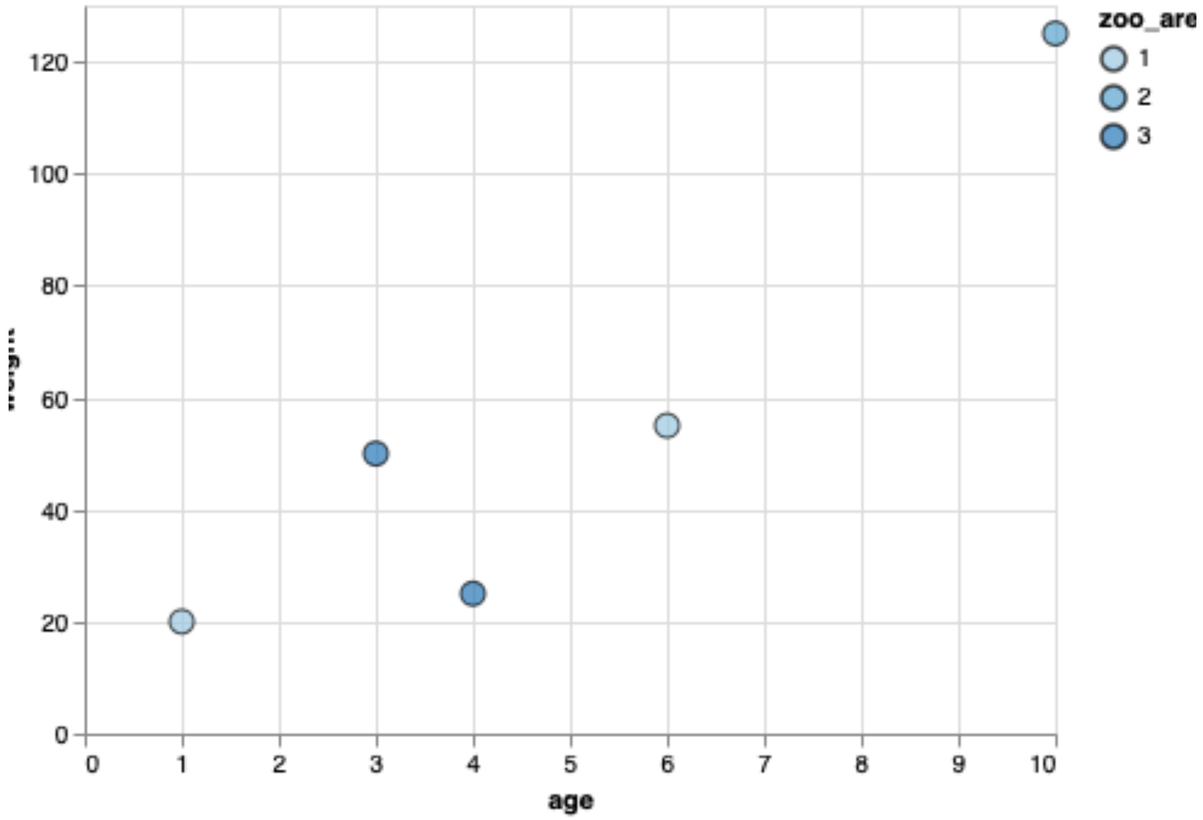
➔ *Quantitative*



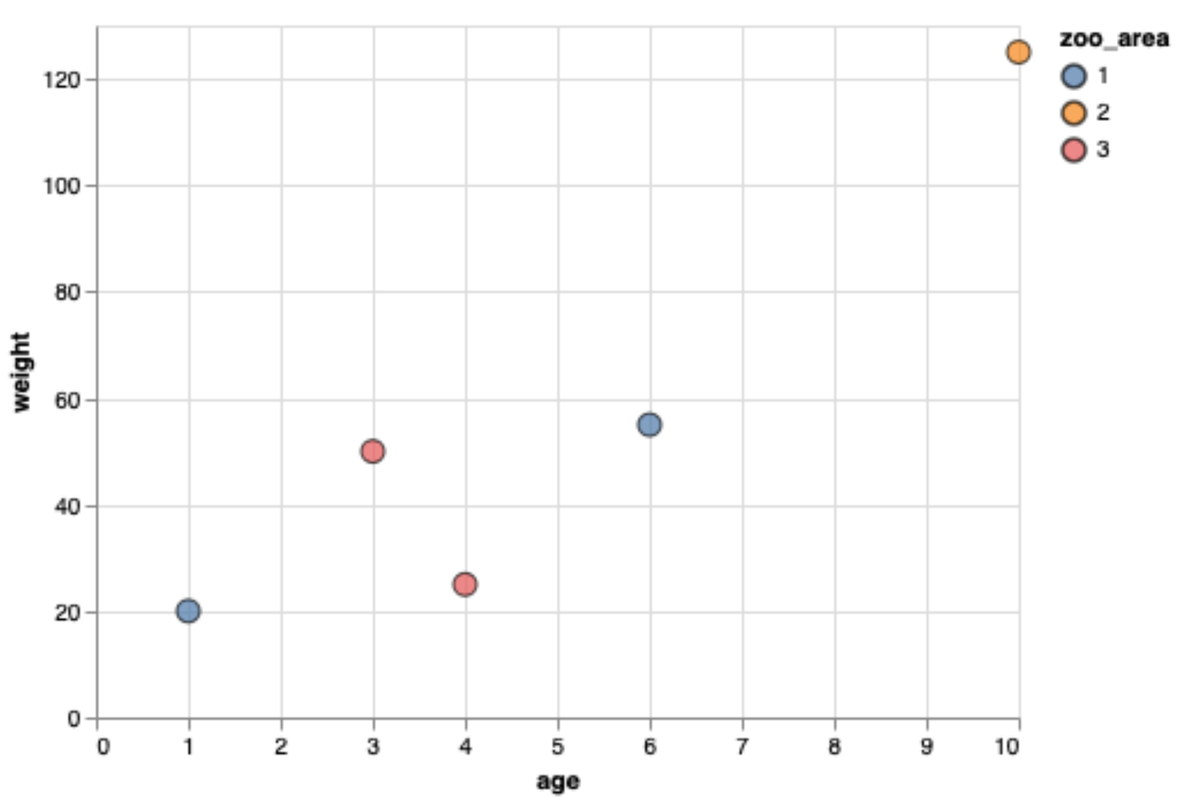
- Categorical data = Nominal (N)
- Ordinal data = Ordinal (O)
- Quantitative data = Quantitative (Q)
- Temporal data = Temporal (T)

[Munzner (ill. Maguire), 2014]

Specifying the Type



`zoo_area:O`

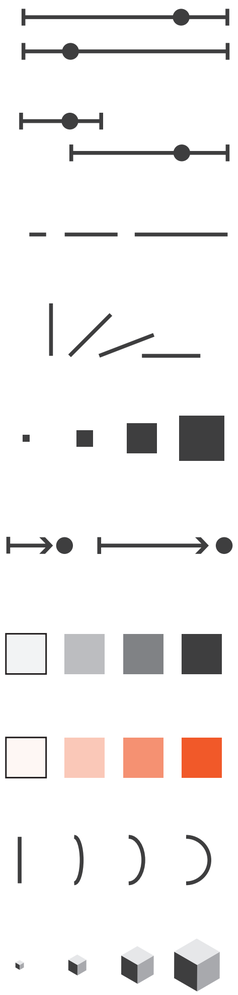


`zoo_area:N`

Different Channels for Different Attribute Types

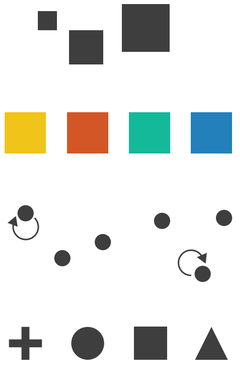
Magnitude Channels: Ordered Attributes

- 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]