Visualization (Intro)

AUTHOR
Peter Ganong and Maggie Shi

PUBLISHED
October 2, 2024

Introduction to Vega-Lite

Roadmap

- Textbook
- What's different from `matplotlib`?
- What is Vega and Vega-Lite?
- First plot – image and then grammar

Citing our sources

This lecture closely follows an online textbook by Jeffrey Heer, Dominik Moritz, Jake VanderPlas, and Brock Craft.

https://idl.uw.edu/visualization-curriculum/

Declarative approaches to visualization.

Our old friend `matplotlib` is "imperative" meaning that you tell the computer what to do. Implement the visualization in terms of for-loops, low-level drawing commands.

Good graphics packages are *declarative*, we mean that you can provide a high-level specification of *what* you want in the visualization. Three inputs

- data
- graphical marks
- encoding channels

We are going to use Vega-Lite + Altair in this class.

What is Vega, Vega-Lite, and Altair?

A grammar of interactive graphics.

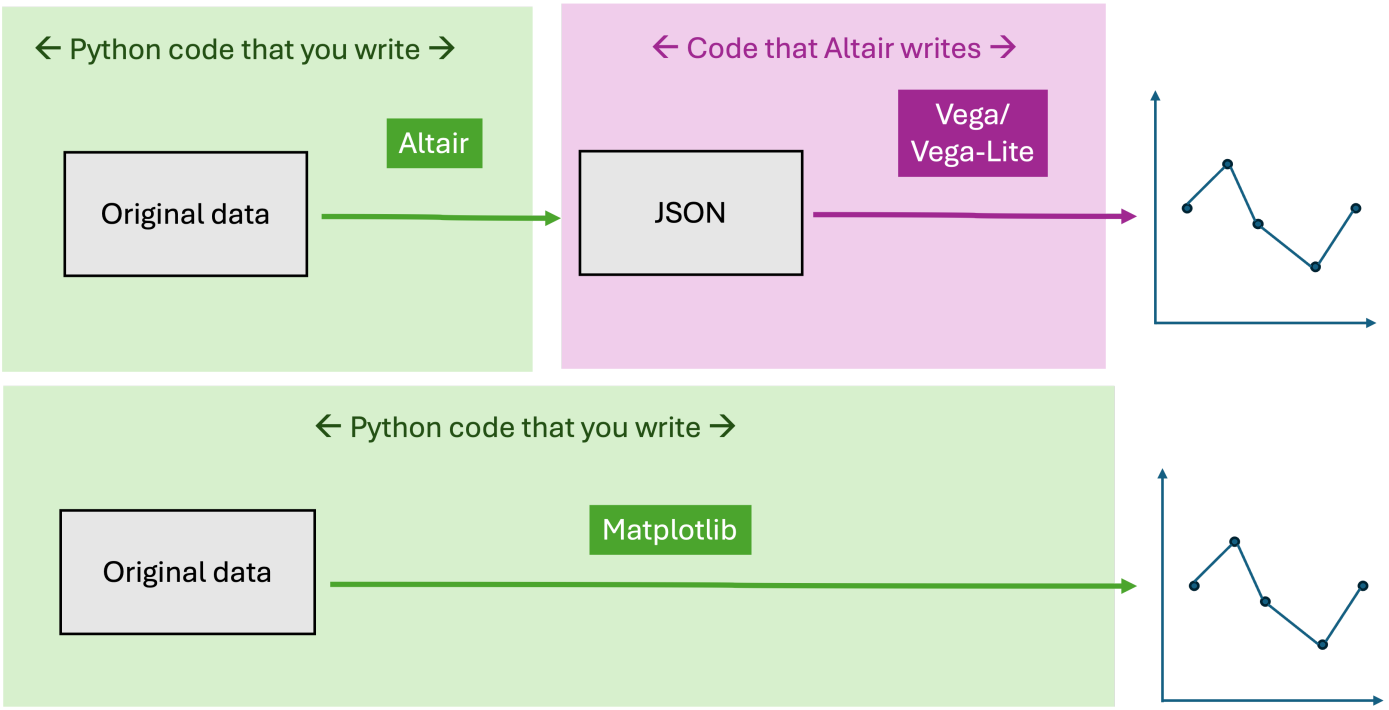
Just like English has a grammar which lets you write a sentence, graphics can have a grammar that let you make a plot.

A good grammar should be easy to use and clear (unlike English...).

New vocabulary:

- Vega is a sophisticated grammar
- Vega-Lite is a more simple grammar
- JSON (JavaScript Object Notation) is used to record Vega and Vega-Lite specifications
- Altair is an API enabling Python to write Vega-Lite

The declarative approach w/ altair vs. imperative w/ matplotlib



- When using a **declarative approach**, your Python code tells `altair` how you want to visualize the data and it writes Vega/Vega-lite to "draw" the graph
- When using an **imperative approach**, your Python code tells `matplotlib` directly how to "draw" the graph

Other declarative graphics packages

#	package	what it is
1	`seaborn`	a wrapper for `matplotlib` to make it easier to use
2	`plotly`	Designed for apps and dashboards. Some features cost $.

#	package	what it is
3	`bokeh`	Designed for apps and dashboards.
4	`plotnine`	an exact clone of `ggplot2`

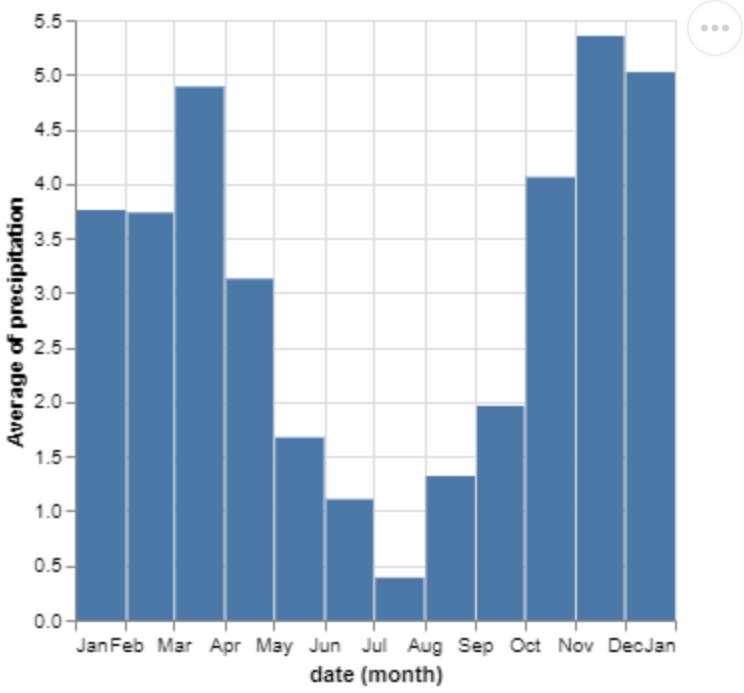
Why did we choose Altair and not one of these?

1. Makes beautiful plots, but under the hood it has all the problems that `matplotlib` does
2. We wanted something that was also good for static graphics. Not entirely free.
3. See above.
4. It doesn't make sense for your "main" plotting language in Python to be a crude port of something from R.

Installation

```
pip install altair
pip install vega_datasets
```

My first plot – image



source.

My first plot – grammar

```
{
  "data": {"url": "data/seattle-weather.csv"},
  "mark": "bar",
  "encoding": {
    "x": {"timeUnit": "month", "field": "date", "type": "ordinal"},
    "y": {"aggregate": "mean", "field": "precipitation"}
  }
}
```

- This is designed to be readable for both a human and for Vega. Pretty cool!

Discussion question: what does each line of text mean?

Why the emphasis on grammar?

- Packages for doing graphics and coding languages change over time
- We chose to teach a package with an underlying grammar because we are trying to foreground the conceptual aspects of data visualization
- This will hopefully teach you insights that are portable, even as which language you or your staff choose to work in change over time

Summary

- Use a declarative approach
- Grammar: Vega
- Gives a coherent conceptual representation underlying a plot

Introduction to Altair and datasets

What is Altair? + roadmap

Altair is a Python API (Application Programming Interface) that generates Vega-Lite specifications in JSON

Roadmap:

- Load package
- Load data

Imports and Renderer

```
import pandas as pd
import altair as alt
```

Depending on your environment, you may need to specify a renderer for Altair. If you are using the class-recommended workflow, you should not need to do anything extra. Otherwise, please read the documentation for Displaying Altair Charts. If that fails, post in Ed and bring your question to lab.

Vega_datasets formatted for Pandas

We will often use datasets from the vega-datasets repository. Some of these datasets are directly available as Pandas data frames:

```
from vega_datasets import data as vega_data
cars = vega_data.cars()
cars.head()
```

	Name	Miles_per_Gallon	Cylinders	Displacement	Horsepower	Weight_in_lbs	Acceleration	Year	Origin
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	1970-01-01	USA
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	1970-01-01	USA
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	1970-01-01	USA
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0	1970-01-01	USA
4	ford torino	17.0	8	302.0	140.0	3449	10.5	1970-01-01	USA

Vega_datasets formatted for JSON

```
#URL if you want
vega_data.cars.url
```

```
'https://cdn.jsdelivr.net/npm/vega-datasets@v1.29.0/data/cars.json'
```

What you will see if you go to this link


```
{
   "Name":"chevrolet chevelle malibu",
   "Miles_per_Gallon":18,
   "Cylinders":8,
   "Displacement":307,
   "Horsepower":130,
   "Weight_in_lbs":3504,
   "Acceleration":12,
```

```
    "Year":"1970-01-01",
    "Origin":"USA"
  }, ...
```

Looks less familiar (and more repetitive). Just use `pd.read_json(data.cars.url)` to convert to tabular

Weather Data

Statistical visualization in Altair begins with "tidy" data frames. Here, we'll start by creating a simple data frame (`df`) containing the average precipitation (`precip`) for a given `city` and `month`:

```
df = pd.DataFrame({
    'city': ['Seattle', 'Seattle', 'Seattle', 'New York', 'New York', 'New York', 'Chicago', 'Chi
    'month': ['Apr', 'Aug', 'Dec', 'Apr', 'Aug', 'Dec', 'Apr', 'Aug', 'Dec'],
    'precip': [2.68, 0.87, 5.31, 3.94, 4.13, 3.58, 3.62, 3.98, 2.56]
})

df
```

	city	month	precip
0	Seattle	Apr	2.68
1	Seattle	Aug	0.87
2	Seattle	Dec	5.31
3	New York	Apr	3.94
4	New York	Aug	4.13
5	New York	Dec	3.58
6	Chicago	Apr	3.62
7	Chicago	Aug	3.98
8	Chicago	Dec	2.56

Summary

- Altair is an API that enables Python to "speak" in Vega-Lite's grammar
- Datasets: cars, weather

Building a first chart

Building a first chart: roadmap

- incrementally build our first chart
- then build our first aggregated chart

The `Chart` object

```
chart = alt.Chart(df)
```

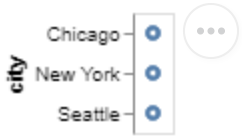
A single point

```
alt.Chart(df).mark_point()
```

Actually this is many points all located in the same place

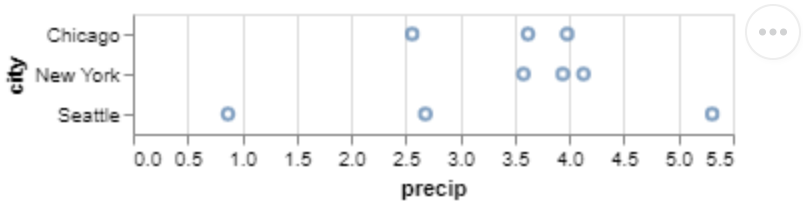
One point per city on y-axis

```
alt.Chart(df).mark_point().encode(
  alt.Y('city')
)
```



xy coordinates

```
alt.Chart(df).mark_point().encode(
  alt.X('precip'),
  alt.Y('city')
)
```

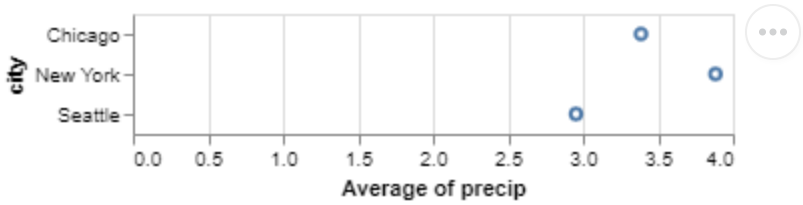


Process: Code is super-duper readable.

Substance: *Seattle exhibits both the least-rainiest and most-rainiest months!*

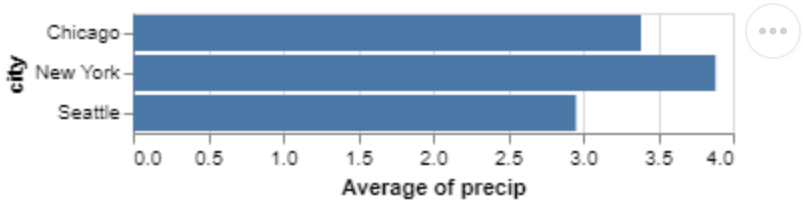
Data Transformation: Aggregation

```
alt.Chart(df).mark_point().encode(
  alt.X('average(precip)'),
  alt.Y('city')
)
```



Bar plot

```
alt.Chart(df).mark_bar().encode(
    alt.X('average(precip)'),
    alt.Y('city')
)
```



Syntax: Understanding Altair's shorthands.

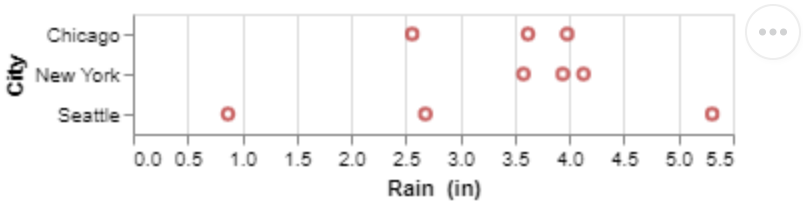
Three ways to say the same idea

```
# what we will continue to use
alt.X('average(precip)')

# shorter
x = 'average(precip)'

#longer
alt.X(aggregate='average', field='precip', type='quantitative')
```

Customizing a plot – colors and labels

```
alt.Chart(df).mark_point(color='firebrick').encode(
  alt.X('precip', axis=alt.Axis(title='Rain  (in)')),
```

```
  alt.Y('city', axis=alt.Axis(title='City')),
)
```



Customizing a visualization – log scale

```
alt.Chart(df).mark_point(color='firebrick').encode(
  alt.X('precip', scale=alt.Scale(type='log'), axis=alt.Axis(title='Log-Scaled Values')),
  alt.Y('city', axis=alt.Axis(title='City')),
)
```

Building a first chart: summary

- Everything begins with a `Chart(data)`
- Every `Chart` needs a `mark`
- Every `Chart` needs guidance how to encode the data in terms of `mark`s
- Simple chart formatting: `mark_point(color='firebrick')`, `axis=alt.Axis(title=...)`, `scale=alt.Scale(type='log')`

Data Transformation: Do-pair-share

Make a bar plot showing the **lowest** rainfall for each city in the dataset.

Hint: Altair's aggregation methods are here

Multiple Views

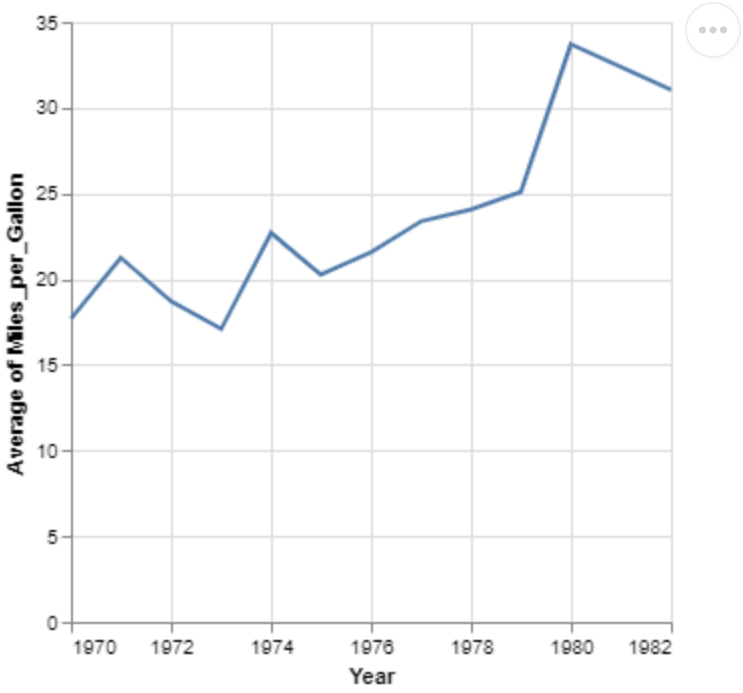
Multiple Views: roadmap

- introduce `mark_line()`
- line + `mark_circle()` on one panel

- multiple panels

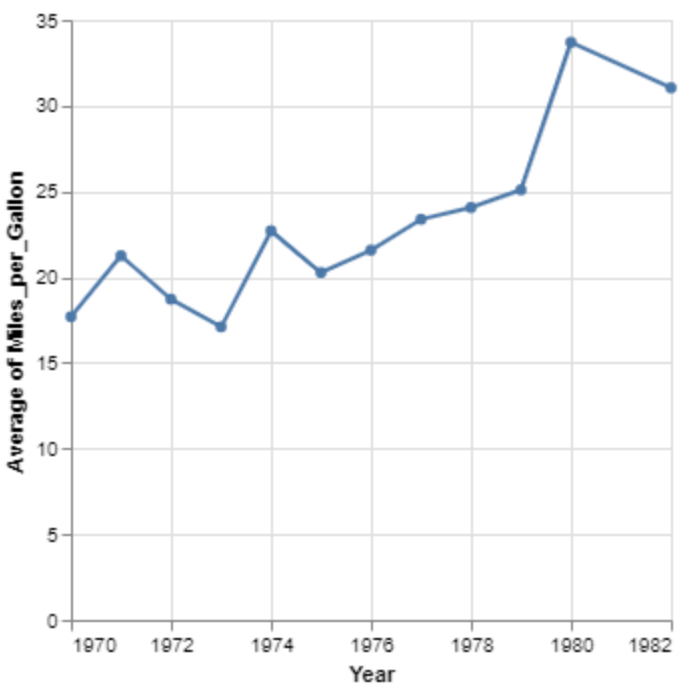
`mark_line()`

```
alt.Chart(cars).mark_line().encode(
    alt.X('Year'),
    alt.Y('average(Miles_per_Gallon)')
)
```



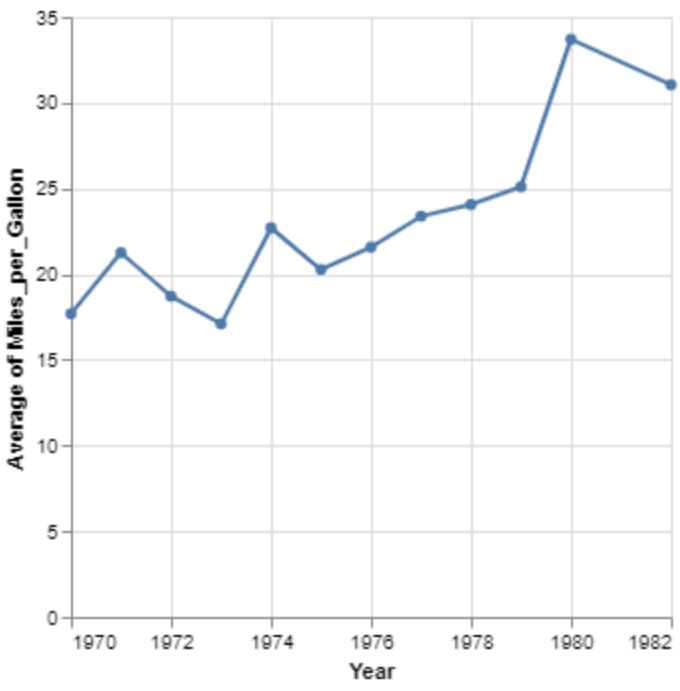
Multiple Marks – the long way

```
line = alt.Chart(cars).mark_line().encode(
    alt.X('Year'),
    alt.Y('average(Miles_per_Gallon)')
)

point = alt.Chart(cars).mark_circle().encode(
    alt.X('Year'),
    alt.Y('average(Miles_per_Gallon)')
)

line + point
```



Multiple Marks – the short way

```
mpg = alt.Chart(cars).mark_line().encode(
    alt.X('Year'),
    alt.Y('average(Miles_per_Gallon)')
)

mpg + mpg.mark_circle()
```

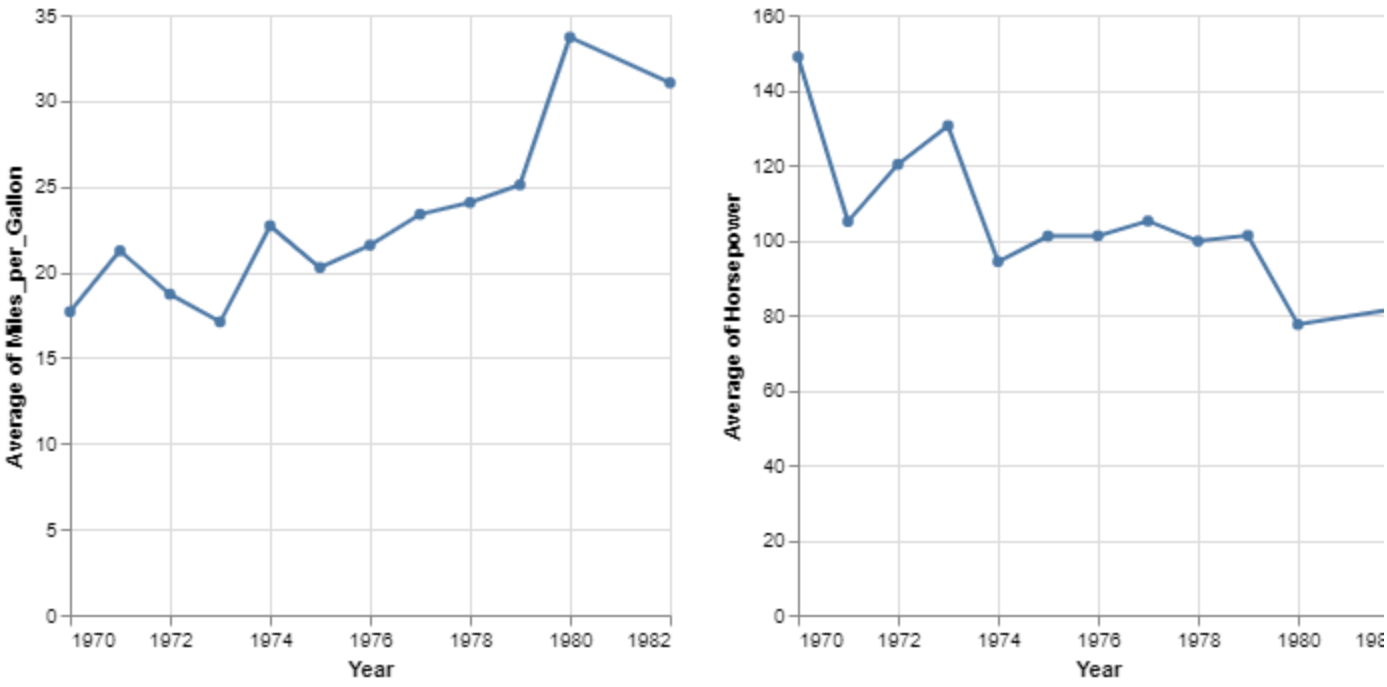


Multiple Marks – the shortest way

```
alt.Chart(cars).mark_line(point=True).encode(
    alt.X('Year'),
    alt.Y('average(Miles_per_Gallon)')
)
```

Multiple Panels

```
hp = alt.Chart(cars).mark_line().encode(
    alt.X('Year'),
    alt.Y('average(Horsepower)')
)

(mpg + mpg.mark_circle()) | (hp + hp.mark_circle())
```



Multiple Views – summary

- `mark_line() + mark_point()` or `mark_line(point=True)`
- `plot1 | plot2` for side-by-side

Note: this is just a preview, we will do a lot more on multiple views in lecture 6 (textbook's chapter 5).

Under the hood: JSON

roadmap

- Beauty of JSON
- Teach how Altair writes JSON via three cumulative examples
- More syntax for working with JSON
- In-class exercise

The beauty of JSON

"JSON Schema is the vocabulary that enables JSON data consistency, validity, and interoperability at scale."

Documents that are

- hierarchical (unlike tables)
- interpretable for humans and computers

https://vega.github.io/schema/vega-lite/v5.json

Example from the schema

```
"Mark": {
  "description": "All types of primitive marks.",
  "enum": [
    "arc",
    "area",
    "bar",
    "image",
    "line",
    "point",
    "rect",
    "rule",
    "text",
    "tick",
    "trail",
    "circle",
    "square",
    "geoshape"
  ],
  "type": "string"
```

example 1: snippet from Altair

Altair can best be thought of as a language translator. It writes JSON for you. More specifically, it writes JSON which complies with the grammar rules laid out by Vega-Lite.

```
#python altair object. will explain :Q in next lecture
x = alt.X('average(precipitation):Q')
print(x.to_json())
```


```
{
  "aggregate": "average",
  "field": "precipitation",
  "type": "quantitative"
}
```


example 2: longer snippet from Altair

```
chart = alt.Chart().mark_point().encode(
    alt.X('average(precipitation):Q'),
    alt.Y('city:O')
)
print(chart.to_json())
```


```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v5.20.1.json",
  "config": {
    "view": {
      "continuousHeight": 300,
      "continuousWidth": 300
    }
  },
  "data": {
    "name": "empty"
  },
  "datasets": {
    "empty": [
      {}
    ]
  },
  "encoding": {
    "x": {
      "aggregate": "average",
      "field": "precipitation",
      "type": "quantitative"
    },
    "y": {
      "field": "city",
      "type": "ordinal"
    }
  },
  "mark": {
    "type": "point"
```

```
  }
}
```

Example 3: add `df` to make the bar plot from earlier in lecture

```
chart = alt.Chart(df).mark_point().encode(
    alt.X('average(precip)'),
    alt.Y('city')
)
print(chart.to_json())
```


```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v5.20.1.json",
  "config": {
    "view": {
      "continuousHeight": 300,
      "continuousWidth": 300
    }
  },
  "data": {
    "name": "data-8e72c2f67818e64f2c6d729f1a903405"
  },
  "datasets": {
    "data-8e72c2f67818e64f2c6d729f1a903405": [
      {
        "city": "Seattle",
        "month": "Apr",
        "precip": 2.68
      },
      {
        "city": "Seattle",
        "month": "Aug",
        "precip": 0.87
      },
      {
        "city": "Seattle",
        "month": "Dec",
        "precip": 5.31
      },
      {
        "city": "New York",
        "month": "Apr",
        "precip": 3.94
      },
      {
        "city": "New York",
        "month": "Aug",
        "precip": 4.13
      },
```

```
        {
          "city": "New York",
          "month": "Dec",
          "precip": 3.58
        },
        {
          "city": "Chicago",
          "month": "Apr",
          "precip": 3.62
        },
        {
          "city": "Chicago",
          "month": "Aug",
          "precip": 3.98
        },
        {
          "city": "Chicago",
          "month": "Dec",
          "precip": 2.56
        }
      ]
    },
    "encoding": {
      "x": {
        "aggregate": "average",
        "field": "precip",
        "type": "quantitative"
      },
      "y": {
        "field": "city",
        "type": "nominal"
      }
    },
    "mark": {
      "type": "point"
    }
}
```


syntax: `keys()`

```
import json
chart_as_string = chart.to_json()
chart = json.loads(chart_as_string)
chart.keys()
```

```
dict_keys(['$schema', 'config', 'data', 'datasets', 'encoding', 'mark'])
```

extract dataset

```
chart.get("datasets")
```


```
{'data-8e72c2f67818e64f2c6d729f1a903405': [{'city': 'Seattle',
   'month': 'Apr',
   'precip': 2.68},
  {'city': 'Seattle', 'month': 'Aug', 'precip': 0.87},
  {'city': 'Seattle', 'month': 'Dec', 'precip': 5.31},
  {'city': 'New York', 'month': 'Apr', 'precip': 3.94},
  {'city': 'New York', 'month': 'Aug', 'precip': 4.13},
  {'city': 'New York', 'month': 'Dec', 'precip': 3.58},
  {'city': 'Chicago', 'month': 'Apr', 'precip': 3.62},
  {'city': 'Chicago', 'month': 'Aug', 'precip': 3.98},
  {'city': 'Chicago', 'month': 'Dec', 'precip': 2.56}]}
```


in-class exercise

1. extract `chart`'s encoding.
2. extract just the encoding for `x`

summary

JSON creates documents that are interpretable for humans and computers.

The JSON schema enforces the grammar rules.