2nd half overview

CSE 314A
DCDS 510

Open Data Sets

- Quality and quantity of open data exploding
- Mature datasets are often provided with interactive dashboards and spreadsheet exports.
- API (Application Programming Interface)
  - Usually gets you more fine grained and complete data
- Web scraping: last resort but sometimes necessary

Real-world data is sourced from many sources & people

Common Open Data Sources

- US Census
- World Bank Open Data
- Social Media: Twitter and Yelp
- Movies: OMDb,
- Sports: Sports-Reference
- FiveThirtyEight
- Yahoo! Finance API
- St Louis Regional Data Exchange
- FEC (data), FEC (API), MEC
- Science: Nature, PLoS One
- ML/AI: Kaggle, UCI ML Repository

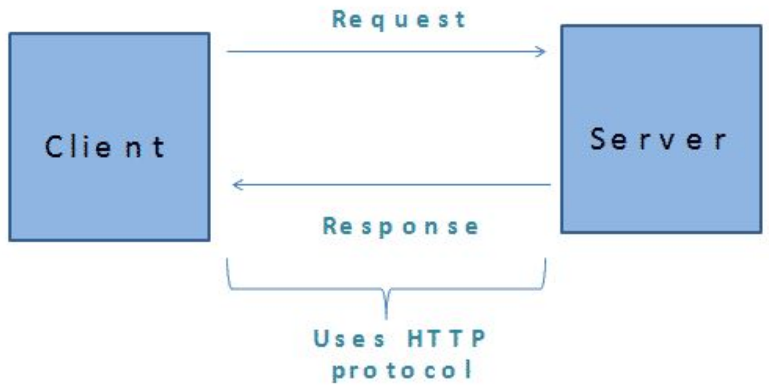
Other closed/personal data sources

- Business data:
  - Accounting, marketing, finance, inventory
- Organizational Data:
  - Contacts, personal information
- Science/Social Science Data:
  - Subjects, results, conditions
- Public Data
  - Freedom of Information Act (FOIA)

- Your own experiment/survey

Anatomy of an API

- **Application Programming Interface** - Terminology
- A reminder of how the Internet works →
- Stereotypically:
  - You tell the API what you want in a URL
  - The server responds with some JSON



API Clients in Python

- Requests package
- Provide common functionality that helps abstract away some web development knowledge (response handling, parsing, pagination, etc)
  - Prepackaged clients, e.g.: Google API Python Client, Sportsipy
  - Build your own client

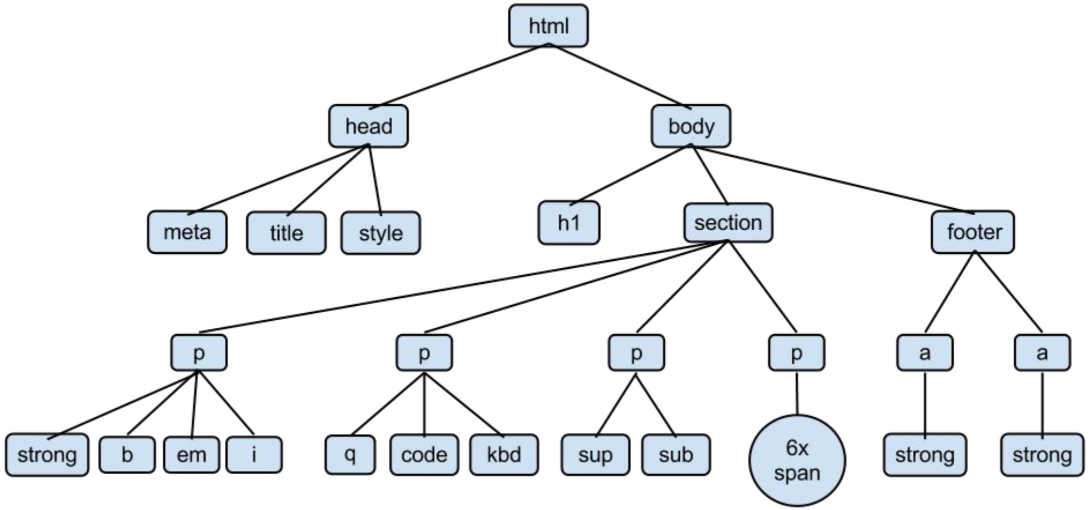
Web Scraping/Crawling - when all else fails?

- Packages
    - BeautifulSoup
    - Scrapy
    - Selenium
    - Some API clients
    - Pandas

Web Scraping Ethics and Etiquette

- Be aware of Terms of Service and proprietary information
- Be aware of your network traffic and server load
- Follow the robots.txt

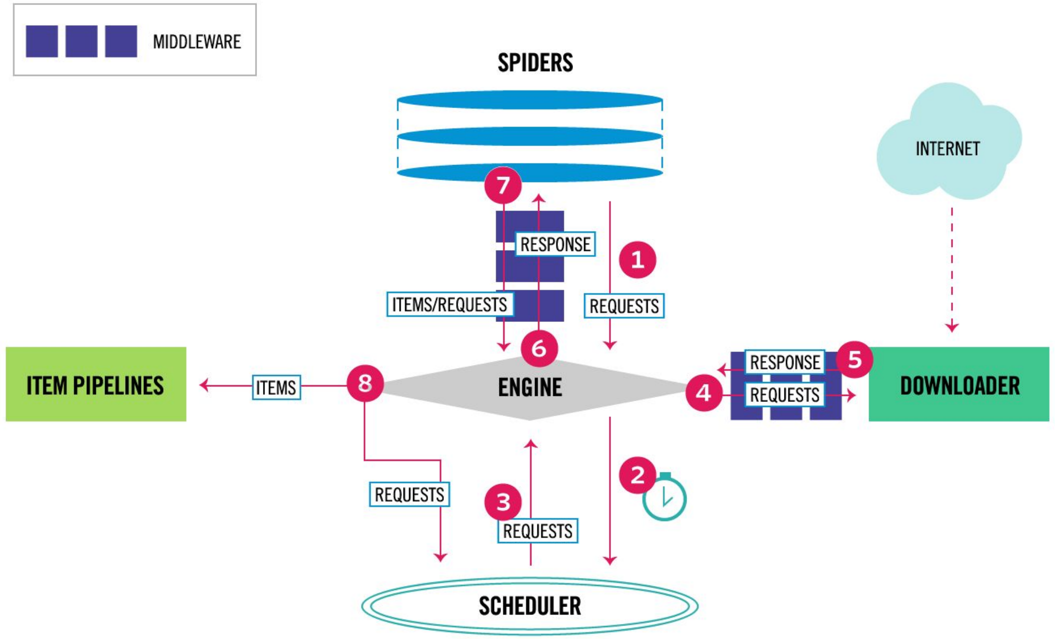
Beautiful Soup - Parsing your HTML



Scrapy -

https://docs.scrapy.org/en/latest/

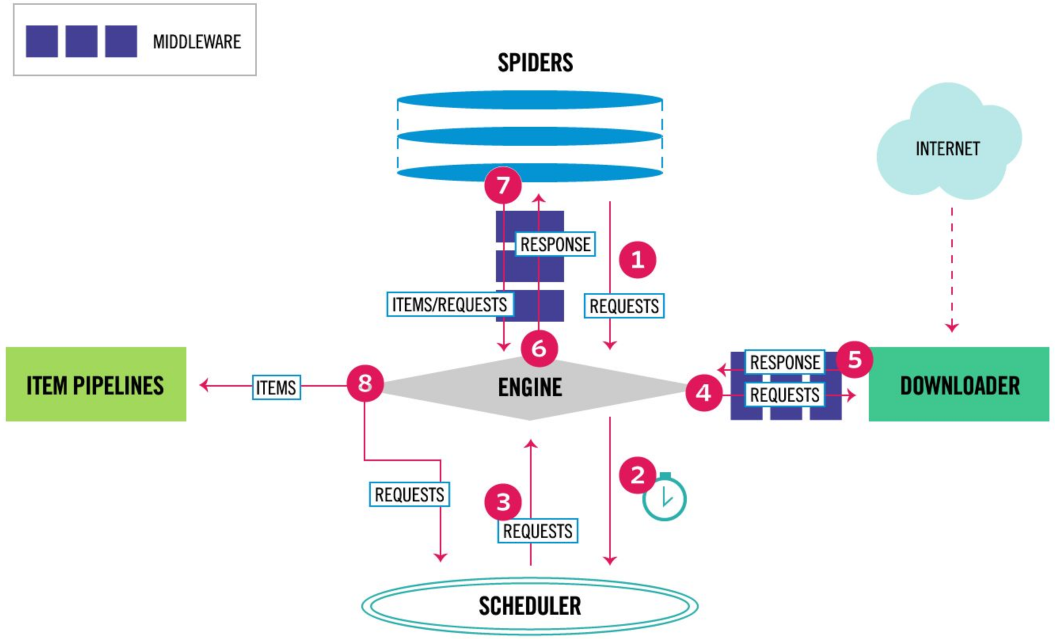
Scrapy is an application framework for crawling web sites and extracting structured data which can be used for a wide range of useful applications, like data mining, information processing or historical archival.



Scrapy -

https://docs.scrapy.org/en/latest/

Even though Scrapy was originally designed for web scraping, it can also be used to extract data using APIs or as a general purpose web crawler.



Selenium

- Designed for automated UI/UX testing
- Most useful for Javascript-focused web browsing
  - Hidden/Dynamic Link navigation
  - Manual clicking processes
- Actually uses browser engines
- Requires some OS-level setup/drivers

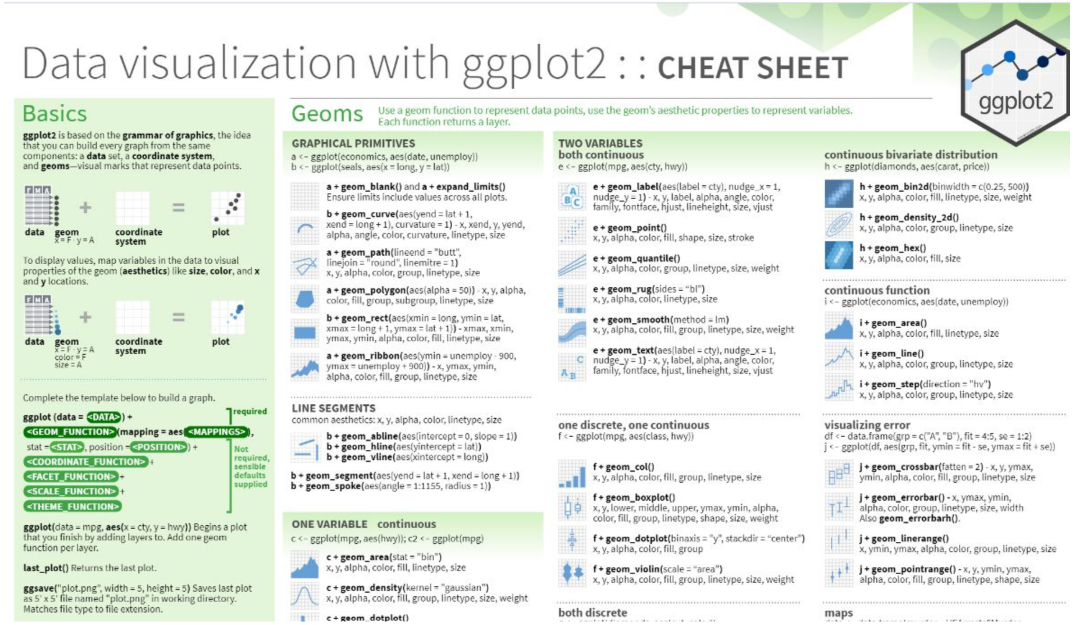
Pandas

- Extracts tables directly from HTML
- Easy, convenient
- Still may require some manual pruning
- Won’t help for data outside of tables

More on storing/caching data

- No go-to answer
- Pandas provides flexible options
- Common approach - store a response and work from that response
  - VCRpy records “cassettes” - designed for testing
- Keep memory limitations in mind, cache as you go
- Focus on what helps you in your development process
  - Don’t repeat yourself
  - Choose good example cases

Historical context - ggplot2 (R Tidyverse)



Data visualization with ggplot2 : : CHEAT SHEET

ggplot2

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.

data **geom** x = F · y = A + **coordinate system** = **plot**

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

data **geom** x = F · y = A color = F size = A + **coordinate system** = **plot**

Complete the template below to build a graph.

```
ggplot (data = <DATA>) +                                   required
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
 stat = <STAT>, position = <POSITION>) +                   Not
<COORDINATE_FUNCTION> +                                    required,
<FACET_FUNCTION> +                                         sensible
<SCALE_FUNCTION> +                                         defaults
<THEME_FUNCTION>                                           supplied
```

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

last_plot() Returns the last plot.

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank() and **a + expand_limits()**
Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend = "butt", linejoin = "round", linemitre = 1**)**
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(alpha = 50)**)** - x, y, alpha, color, fill, group, subgroup, linetype, size

b + geom_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1)**)** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)**)** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept = 0, slope = 1)**)**
b + geom_hline(aes(yintercept = lat)**)**
b + geom_vline(aes(xintercept = long)**)**

b + geom_segment(aes(yend = lat + 1, xend = long + 1)**)**
b + geom_spoke(aes(angle = 1:1155, radius = 1)**)**

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin"**)**
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()

TWO VARIABLES
both continuous

e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_point()
x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile()
x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)**
x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)**
x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

one discrete, one continuous

f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot()
x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, group, linetype, size, weight

both discrete

continuous bivariate distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density_2d()
x, y, alpha, color, group, linetype, size

h + geom_hex()
x, y, alpha, color, fill, size

continuous function

i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv"**)**
x, y, alpha, color, group, linetype, size

visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))

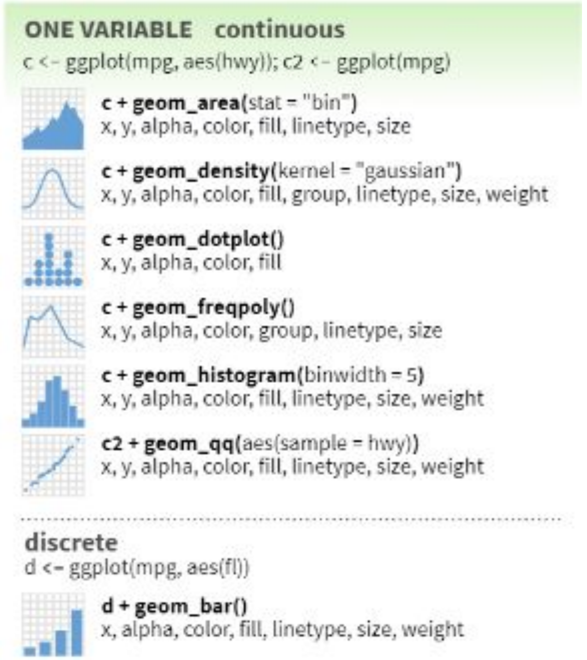
j + geom_crossbar(fatten = 2**)** - x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar() - x, ymax, ymin, alpha, color, group, linetype, size, width
Also **geom_errorbarh()**.

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange() - x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps



ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin"**)**
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly()
x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)**)**
x, y, alpha, color, fill, linetype, size, weight

discrete

d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

Python Visualization in 2022

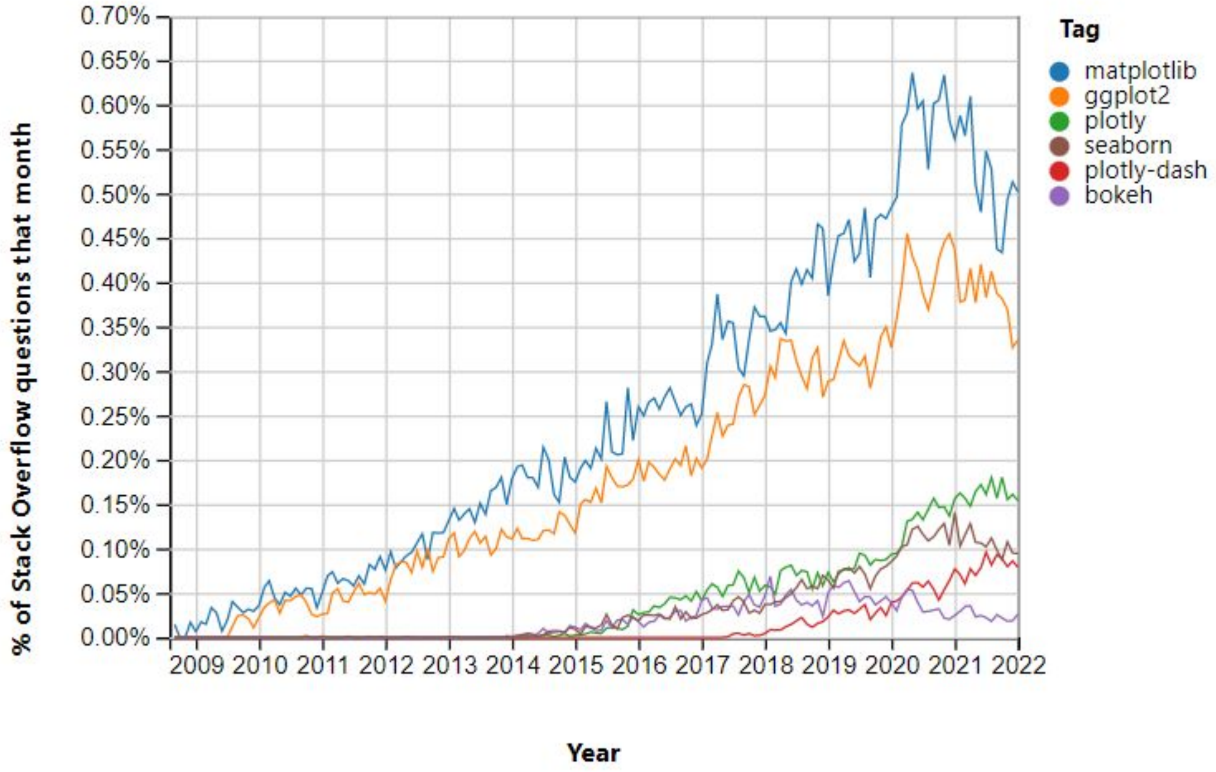
plotly + Dash

matplotlib

(HoloViz)

- Libraries can be used by Jupyter notebooks and Pandas API

Stack Overflow Trends



Viz-first data analysis

- When presenting findings
  - Keeps you focused on outcomes
  - Evaluate hypotheses
  - Keep your supervisors happy
- When exploring data
  - Narrows down possibilities
  - Fosters ongoing thinking about data structure and domain particularities
  - Familiarizes you with the data if it is external
- Simply more fun and less discouraging
- Answers a lot of philosophical questions up front
  - Resource: Fundamentals of Data Visualization by Claus O Wilke

Feeding your plot - Quality in, Quality Out

- Modern plotting libraries can do much of the data processing for common visualizations
- Organized data allows for single-command plotting
  - More on this next week - for now use preorganized data in hw/demos
  - Fake your data if your data is unorganized or unavailable!

Basic plotting (Plotly)

- Raw Figure objects - Just fancy dictionaries!
  - Use when customizing, examining structure, or testing
- Plotly Express
  - Works best (but not only)with “tidy” data – but it essentially means that you have one row per observation.
  - “Wide” data support added in last year or two.
- Use templates
- Minimize tinkering
- Base level interactivity

Dashboard basics (Dash)

- Works with Flask under the hood (easily deployable – relatively)
- Uses HTML to arrange components
  - Dash Bootstrap Components can make your app look pretty and provide some supplemental functionality
- Uses “callbacks” to update the dashboard on user actions
  - Just functions – reads attributes from HTML components and updates them.
- Callbacks are chainable – you can use use them to update other input components as well as your figure data

Exporting your images and deploying your dashboards

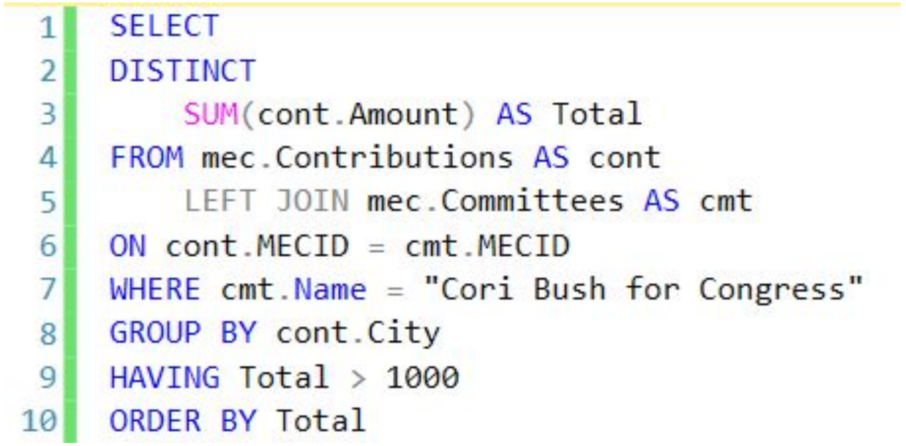
- Images
    - PNG button on dashboard
    - I/O methods on Figure objects
        - Always use vector formats if possible – .svg, .pdf, etc
    - Post-processing
        - Inkscape - free and open source
        - Adobe Illustrator
- Deploying Dashboards – tricky but doable
    - The Magical Guide

The future: 3D dataviz, VR/AR, and data storytelling

https://www.tiktok.com/@the.data.guy/video/7043589231066860846

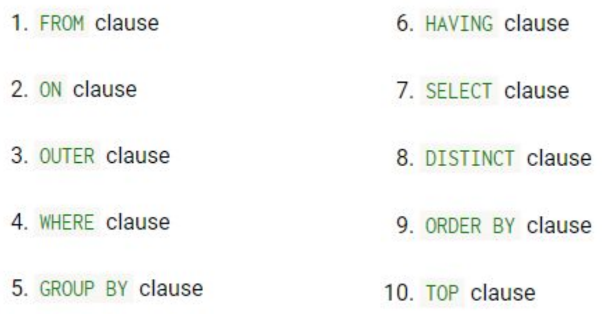
https://www.tiktok.com/@the.data.guy/video/7028635623833488646

SQL from 10,000 feet -



```
SELECT
DISTINCT
    SUM(cont.Amount) AS Total
FROM mec.Contributions AS cont
    LEFT JOIN mec.Committees AS cmt
ON cont.MECID = cmt.MECID
WHERE cmt.Name = "Cori Bush for Congress"
GROUP BY cont.City
HAVING Total > 1000
ORDER BY Total
```

SQL order of execution - SELECT

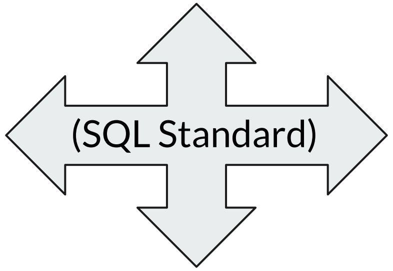


1. FROM clause
2. ON clause
3. OUTER clause
4. WHERE clause
5. GROUP BY clause
6. HAVING clause
7. SELECT clause
8. DISTINCT clause
9. ORDER BY clause
10. TOP clause

CRUD and ACID

- CRUD operations
  - Create
  - Read
  - Update
  - Delete
- ACID transactions
  - Atomicity - All or nothing
  - Consistency - Data is correct
  - Isolation - Independent of other transactions
  - Durability - changes are persistent

RDBMS systems



Differences:

- User Permissions/Security
- Programmatic features
- Some architecture/indexing decisions
- Secret Sauce

NoSQL - Think “Nonrelational”

- Key-Value Databases (common for caching [Redis])
- Document Databases (MongoDB) - think JSON/XML
- Graph Databases

Key Considerations:

- “Schema-less”
- Horizontal Scalability
- Performance gains at cost of ACID compliance