UMD DATA605 - Big Data Systems

Course Intro
Big Data
Data Models

Dr. GP Saggese
gsaggese@umd.edu

with thanks to Prof.
Alan Sussman
Amol Deshpande

UMD DATA605 - Big Data Systems

Course Intro

Big Data

Data Models

Goals

- Learn to model and reason about data
- Learn how to process and manipulate data in different ways
- Introduce a variety of data models
  - Decide which data model is appropriate for different applications
- Learn to use a variety of data management systems
  - Decide which system is appropriate for a data management scenario
- Learn how to build data processing pipelines
- Learn how to build a big-data system end-to-end
  - Class project!
  - Team work

Todos

- DATA605 - ELMS / Canvas site
  - Make sure to enable notifications
  - How to get in touch with me / TA
- DATA605 - Schedule
- Clone DATA605 - GitHub repo
- Begin setup of your computing environment
  - Install Docker on your laptop
  - Instructions are in the class repo

Tools We Will Learn To Use

- **Languages**: mainly Python
- **Dev tools**
  - Bash / Linux OS
  - Git (understand data model, branching)
  - GitHub (PRs, work with issues)
  - Code editors (PyCharm, Visual Studio, vim / emacs)
  - Data science libs (pydata stack: numpy, Pandas, sklearn, matplotlib)
  - Jupyter notebooks
  - Docker
  - Unit testing framework
- **Big data tools**
  - ETL pipelines
  - Relational DBs (PostgreSQL)
  - NoSQL DBs (HBase, MongoDB, Couchbase, Redis)
  - Graph DBs (Neo4j, OrientDB, AllegroGraph, GraphX, Giraph)
  - Computing framework (Hadoop, Spark, Dask, Storm, Spark Streaming)
  - Workflow manager (Airflow)
  - Cloud Services (AWS)
- **Tutorials** for all tools we use for the class project

Soft Skills to Succeed in the Workplace

- **Goal**: model the class project to prepare you for the workplace
- Skills that people out from college often don’t have
  - Work in a team
  - Design software architecture (e.g., OOP, Agile, Design Patterns)
  - Comment your own code
  - Write external documentation (e.g., tutorials, manuals, how-tos)
  - Write code so that other people can understand (including future-you)
  - Read other people’s code
  - Follow code conventions (e.g., PEP8, Google Code)
  - Be clear in communications (e.g., in emails, Slack)
  - File a bug report
  - Reproduce (aka repro) a bug
  - Have a sense of CS constants
  - Have a sense of how an OS works (e.g., virtual memory, processes)

Class Project

- DATA605 - Class project
  - Build an end-to-end big data system importing data, computing, orchestrating the tasks
  - Teams of 4-5 students
    - GP and the TA will create well-balanced teams
  - Review 2 examples of complete projects
  - 5 deliverables with deadlines graded individually (20% of final grade)
  - Every class ~1 hour of group work with GP + TA
  - Peer evaluation of the project
  - Grad and undergrad students will be assigned final grades separately
  - Will re-evaluate based on how projects go

- Student survey
  - Collect some information about you to organize groups for the class project

- Check out an example of Big Data System
  - Sorrentum project and Git repo

Why I Am Interested in Big Data

- https://www.linkedin.com/in/gpsaggese/
  - Feel free to connect on LinkedIn so we can stay in touch
- Open-source project I’ve started
  - Sorrentum
  - GitHub

Your Turn

- Introduce yourself
- Tell us something about yourself
  - E.g., hobby, peculiar habits, ...

UMD DATA605 - Big Data Systems

Course Intro

Big Data

Data Models

Data Science

- **Promises of data science** (DS)
  - Give a competitive advantages
  - Make better strategic and tactical business decisions
  - Optimize business processes
  - Detect strategic blind spots
- **Data science is not new**, it was called:
  - Decision support
  - Business intelligence
  - Predictive analytics
  - Operation research
  - ...
- What has changed is that **learning and applying DS is easy**
  - No need for consulting groups
  - Tools are open-source
    - E.g., Python + pydata stack (numpy, scipy, Pandas, sklearn)
  - Large data sets
  - Cheap computing (e.g., AWS, Google Cloud)

Motivation: Data Overload

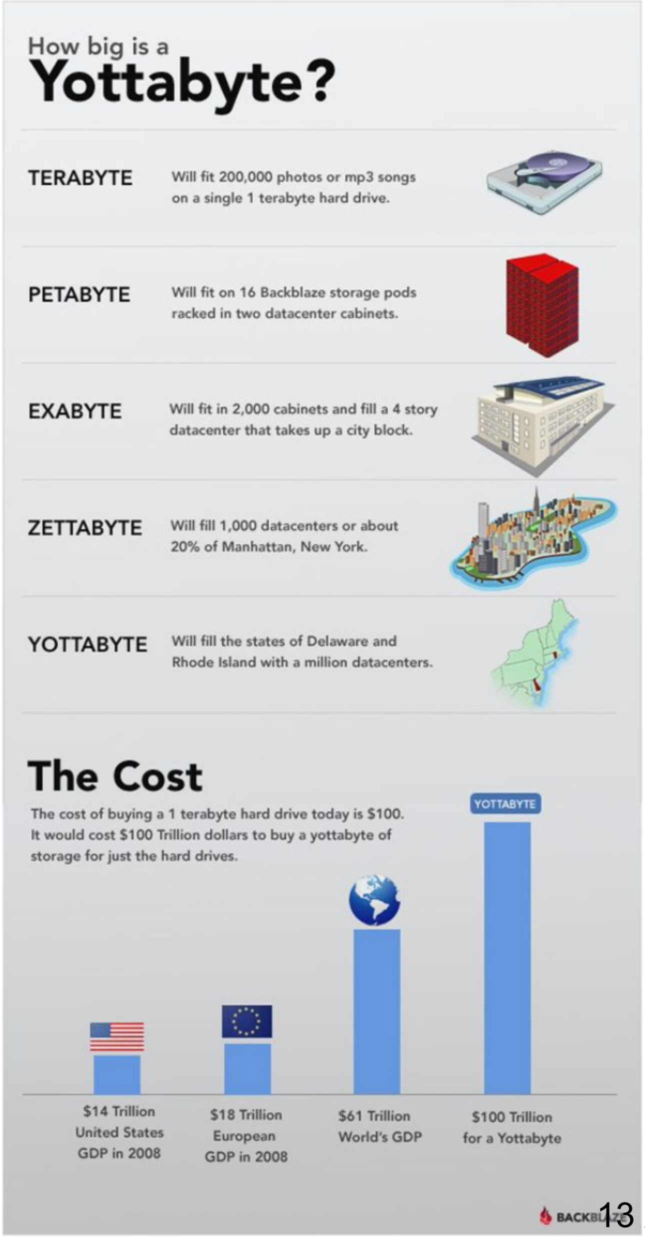
- McKinsey, 2013: "data science is the **number one catalyst for economic growth**"
  - Growing amount of data
  - Cloud computing
  - Build machine models to make better business decisions

- **Explosion of data in every domain**
  - Sensing devices / networks monitor processes 24/7
    - E.g., temperature of your room, your vital signs, pollution in the air
  - Increasingly sophisticated smart-phones (6.5b smart phones, 80% of the world population)
  - Internet and social networks make it easy to publish data
  - Internet of Things (IoT), everything is connected to the internet
    - E.g., power supply, toasters
  - Scientific experiments and simulations produce enormous volumes of data
  - [Datafication](): turn all aspects of life into data
    - E.g., what you like/enjoy turned into a stream of your "likes"

- **Challenges**
  - How to handle increasing amount data?
  - How to extract interesting actionable insights and scientific knowledge?

Scale of Data Size

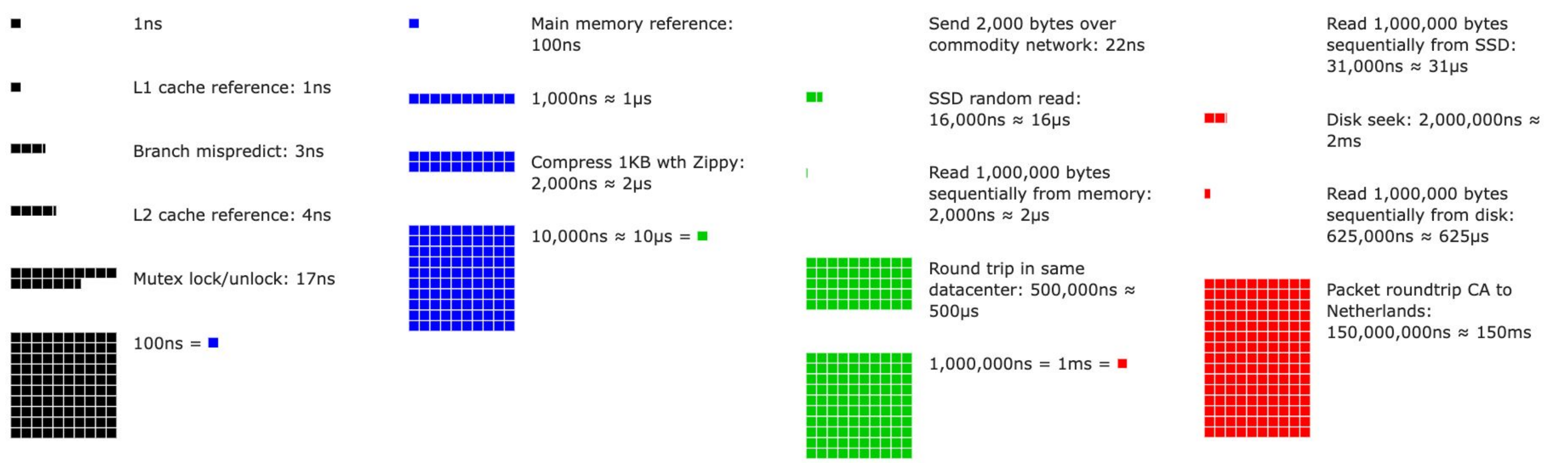
- Megabyte = 2^{10} \approx 10^{6} bytes
  - Typical English book
- Gigabyte = 10^{9} bytes
  - 1/2 hour of video
  - Wikipedia (compressed, without media): 22GB
- Terabyte = 1m MB
  - Human genome: ~1TB
  - 200,000 photos
  - LHC generates 100TB of data per day
  - $50 to buy 1TB HDD, $23 / mo on AWS S3
- Petabyte = 1000 TB
  - 13 years of HD video
  - 65 copies of Library of Congress
  - $250k / year on AWS S3
- Exabyte = 1m TB
  - Global yearly Internet traffic in 2004
- Zetabyte = 1b TB = 10^{21} bytes
  - Global yearly Internet traffic in 2016
- Yottabytes = 10^{24} bytes
  - 1TB today costs ~$100, Yottabyte costs $100t
- Brontobytes = 10^{27} bytes



Constants that everybody should know

From Latency Numbers Every Programmer Should Know (by year)

- A CPU running at 3GHz executes an instruction every 0.3ns
- L1 cache reference / register: 1ns
- L2 cache reference: 4ns
- Main memory reference: 100ns
- Send 1KB over network: 10ns
- Read 1MB from memory: 2us
- SSD random read: 16us
- Disk seek: 2ms
- Packet round-trip from CA to Netherland: 150ms



Big Data Applications

- **Personalized marketing**
  - Target each consumer instead of the consumers at large
  - E.g., Amazon personalizes suggestions using signals from:
    - Your shopping history
    - What you have searched for (or clicked, watched)
    - Other consumers and trends
    - Reviews (through NLP and sentiment analysis)
  - Brands want to understand how customers relate to products
    - Use sentiment analysis from
      - social media
      - on-line reviews
      - blogs
      - surveys
    - Positive, negative, neutral feeling
- E.g.,
  - In 2022, $600b spent on digital marketing
  - [50 Stats Showing The Power Of Personalization]

Big Data Applications

- **Mobile advertisement**
  - Mobile phones are ubiquitous
    - 80% of world population has one
  - Integration of on-line and off-line databases, e.g.,
    - GPS location
    - Search history, credit history, credit card transactions
  - E.g.,
    - You've bought a new house
    - You google questions about house renovations
    - You watch shows about renovations
    - Your phone tracks where you are
    - Google sends you coupons for the closest Home Depot

Big Data Applications

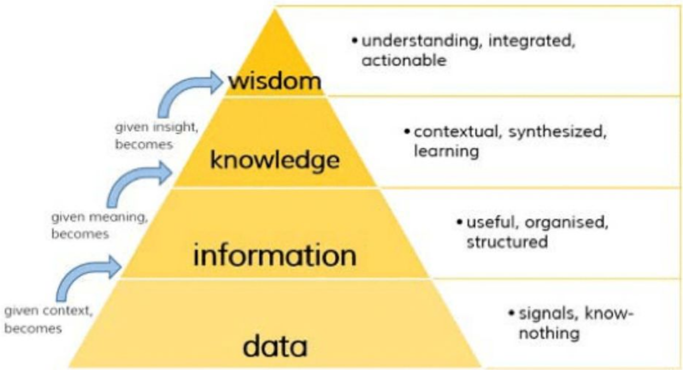
- **Biomedical data**
  - Personalized medicine
    - Patients can receive treatment tailored to them to maximize efficacy
    - Genetics, daily activities, environment, habits
  - Genome sequencing

Big Data Applications

- **Smart cities**
  - Interconnected mesh of sensors
    - E.g., traffic sensors, camera networks, satellites
  - Goals:
    - Monitor air pollution
    - Minimize traffic congestion
    - Optimal urban services
    - Maximize energy savings

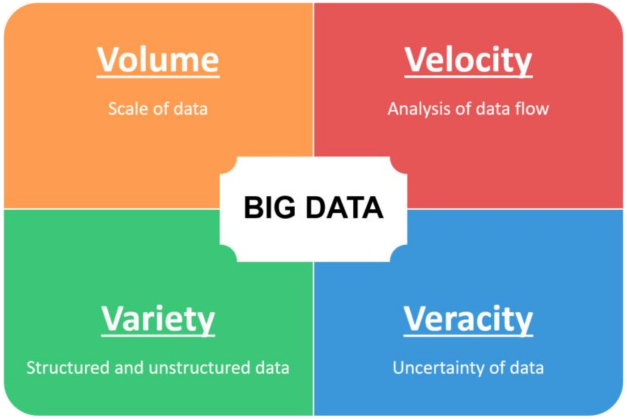
Goal of Data Science

- Goal of data science: from data to wisdom
  - Data (raw bytes)
  - Information (organized, structured)
  - Knowledge (learning)
  - Wisdom (actionable, understanding)
- Combine streams of big data to generate new data
  - New data can be "big data" itself
- Insights enable decisions and actions



Four V’s of Big Data

- Big data often exhibits certain characteristics and qualities
- Volume
  - Vast amount of data is generated
- Variety
  - Different forms
  - E.g., text, images, voice, geospatial data
- Velocity
  - Speed at which data is generated
- Veracity
  - Biases, noise, abnormality in data
  - Uncertainty, trustworthiness
- (Valence)
  - Connectedness of big data in the form of graphs
- (Value)
  - Challenges of big data can distract from focusing on how big data benefits an organization



Four V's of Big Data

- **Volume**
  - Exponentially increasing amount of data
  - Every day 2.5 exabytes (1m of TB) of data is generated
    - 90% of all the data in the world was generated in the last 2 years
    - Total amount of stored data doubles every 1.2 years
  - Twitter: 500M tweets / day (2022)
  - Google processes 8.5b queries / day (2022)
  - Facebook generates 4PB of data / day (2022)
  - Walmart: 2.5PB of unstructured data / hour (2022)

- **Variety**
  - Different form
    - Structured data (e.g., spreadsheets, relational data)
    - Semi-structured data (e.g., natural language text, sales receipts, your class notes)
    - Unstructured data (e.g., photos, videos)
  - Different formats (e.g., flat files, CSV, XML, JSON)

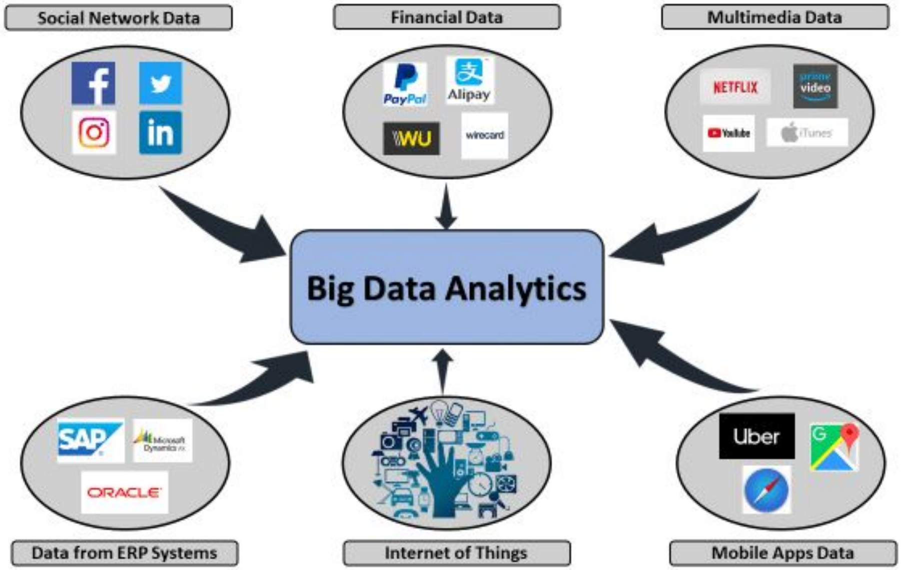
Four V's of Big Data

- **Velocity**
  - Relates to the speed at which data is generated
    - E.g., sensors generating data streams
  - Sometimes data can be processed off-line
  - Real-time analytics requires data to be consumed as fast as it is generated

- **Veracity**
  - Relates to data quality
  - How to remove noise and bad data?
  - How to fill in missing values?
  - What is an outlier?
  - How do you decide what data to trust?

Sources of Big Data

- We can distinguish Big Data in terms of its source
  - Machines
  - People
  - Organizations



Sources of Big Data: Machines

- **Machines** generate data
  - Real-time sensors (e.g., sensors on Boeing 787)
  - Cars
  - Website tracking
  - Personal health trackers
  - Scientific experiments
- Pros
  - Highly structured
- Cons
  - Can't be easily moved, but need to be computed in-situ or in centralized fashion
  - Streaming, not batch

Sources of Big Data: People

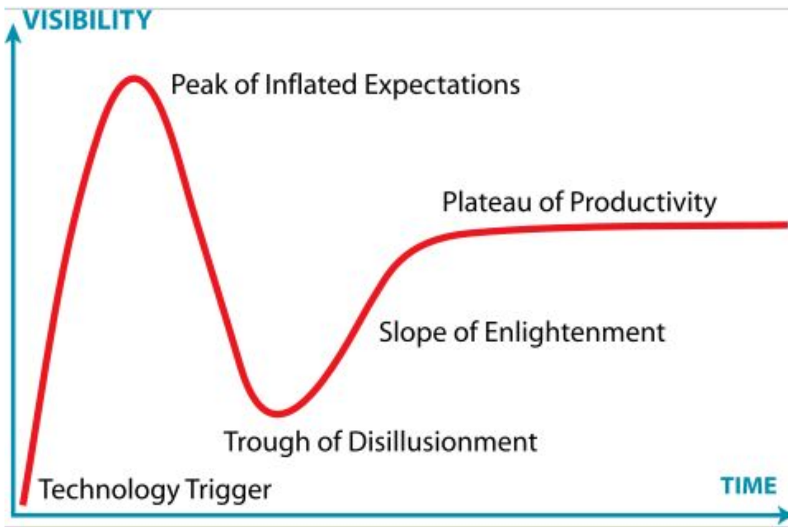
- **People** and their activities generate data
  - Social media (e.g., Facebook, Instagram, Twitter, LinkedIn)
  - Video sharing (e.g., YouTube, TikTok)
  - Blogging and commenting on a website
  - Internet searches
  - Text messages (e.g., SMS, Whatsapp, Signal, Telegram)
  - Personal documents (e.g., Google Docs, emails)
- Pros
  - Highly valuable for business intelligence
  - Allow personalization
- Cons
  - Typically unstructured data
    - Text, images
  - It takes investment before you can reap the value
    - Acquire → Store → Clean → Retrieve → Process → Insights
  - Surveillance capitalism

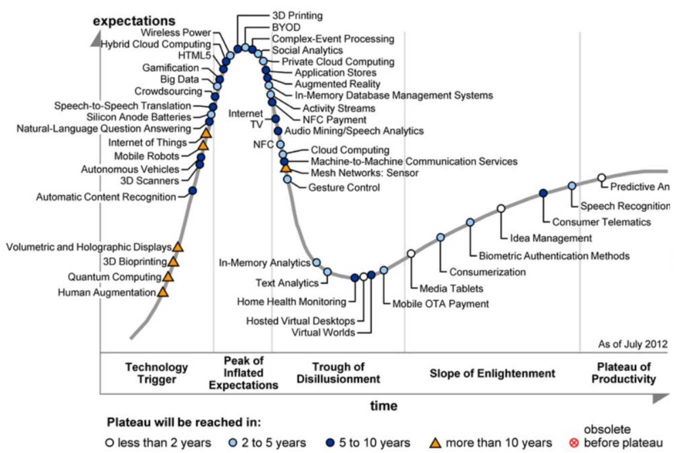
Sources of Big Data: Orgs

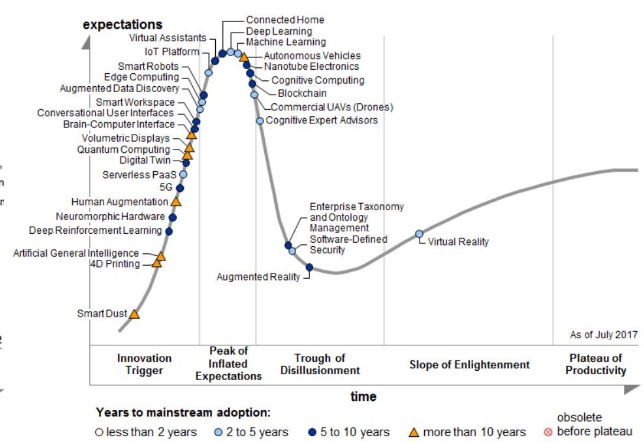
- **Organizations** generate data
  - Commercial transactions
  - Credit cards
  - E-commerce
  - Banking
  - Financial information
  - Medical records
  - Transactions
  - Clicks on a website
- Pros
  - Highly structured
- Cons
  - Store every event in the past to predict the future
  - Stored in “data silos” with different data models
    - Each department has its own data system
    - Additional complexity, missing opportunities, data is outdated / not visible
    - Cloud computing helps (e.g., data lakes, data warehouses)

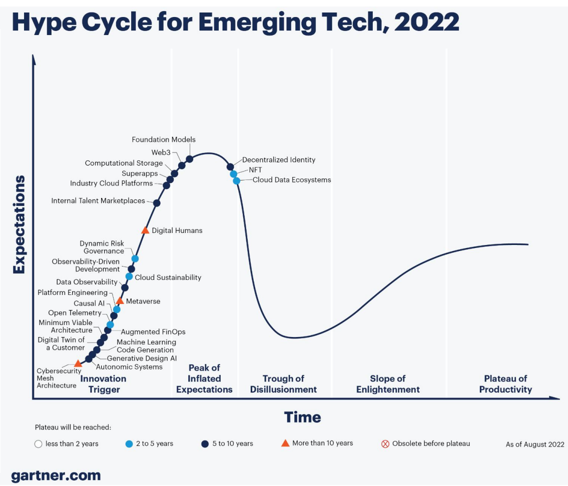
Big Data + Data Science to the Rescue

- *Big data* and *data science* often used synonymously
  - Aka data analytics, data mining, business intelligence, predictive analytics
  - Loosely used for any process where interesting information are inferred from data
- Data scientist called the “sexiest job” of the 21st century
  - The term has becoming very muddled at this point









Is Data Science Just Hype?

- **No**
  - Extracting insights and knowledge from data:
    - Is very important
    - Will continue to increase in importance
  - Big data techniques are revolutionizing the world in many domains
    - E.g., education, food supply, disease epidemics, …
- **But**
  - Not much different from what statisticians have been doing for many years

- **What is different? What has changed?**
  - Much more data is digitally available than ever before
  - Easy-to-use programming frameworks (e.g., Hadoop, Spark) = much easier to analyze it
  - Cloud computing
  - Often large-scale data + simple algorithms > small data + complex algorithms

Key Shifts Before / After Big-Data

1) Datasets: small, curated, clean → large, uncurated, messy
   - Before:
     - Statistics based on small, carefully collected random samples
     - Costly and careful planning for experiments
     - Hard to do fine-grained analysis
   - Today:
     - Easily collect huge volume of data
     - Feed it into algorithms
     - Usually the signal is strong enough to overcome the noise
2) Causation → Correlation
   - Goal: figure out what caused what
   - Causation very hard to figure out
   - Today: give up causation for correlation
     - Find out that two things are correlated and often that's enough
3) "Datafication" = process of converting abstract things into concrete data
   - E.g., "sitting posture" is datafied by capturing information with 100's of sensors placed in your seat
   - Your preference is datafied into a stream of your likes

- From: [Rise of Big Data](), 2013

Examples: Election Prediction

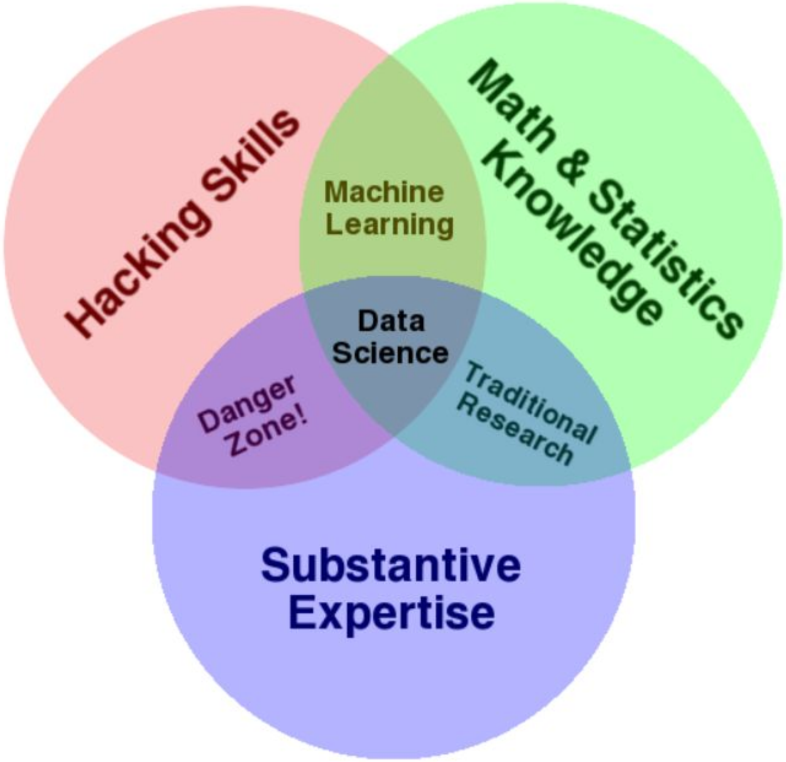
- Nate Silver and the 2012 Elections
  - 49 out of 50 in calling each state in 2008 US elections
  - 50 out of 50 in 2012 US elections
  - Didn't work that well in 2016
- Some reasons why he got things right
  - Many sources of data, irrespective of quality
  - Incorporation of historical accuracy
  - Use of statistical models
  - Understanding correlations
  - Monte-Carlo simulations to compute the probabilities of electoral college
  - Focus on probabilities instead of predictions
  - Great communication and presentation skills

Examples: Google Flu Trends

- Google Flu Trends
  - Early warning of flu outbreaks by analyzing search queries
    - What terms people searched for (45 search terms used to create a single model)
    - IP to determine location
    - Application of “collective intelligence”
  - Predict regional outbreaks of flu up to 1 or 2 weeks ahead of CDC
    - 5% to 20% of US population contract flu every year and 40k deaths
    - Earlier warnings allow prevention and control
  - Service in activity from 2008 to 2015
- Caveat: accuracy not as good any more
  - Google claimed 97% accuracy
  - Out of sample accuracy lower (overshot CDC data by 30%)
  - People search about flu without knowing how to diagnose flu (e.g., people searching for “fever” and “cough”)
  - Google Flu Trends: The Limits of Big Data

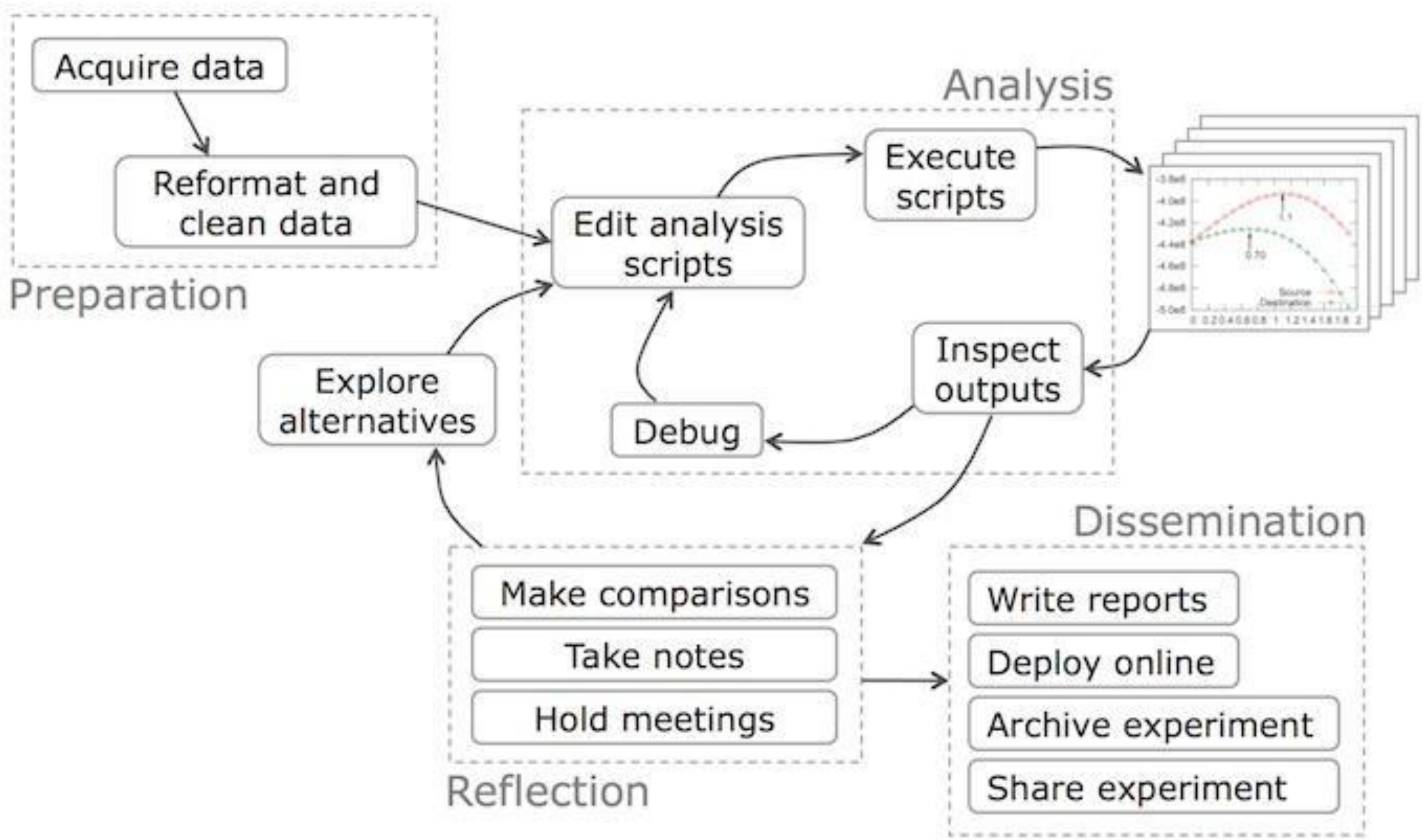
Data Scientist

- Very ambiguous and ill-defined term



- From [Drew Conway's Venn Diagram]

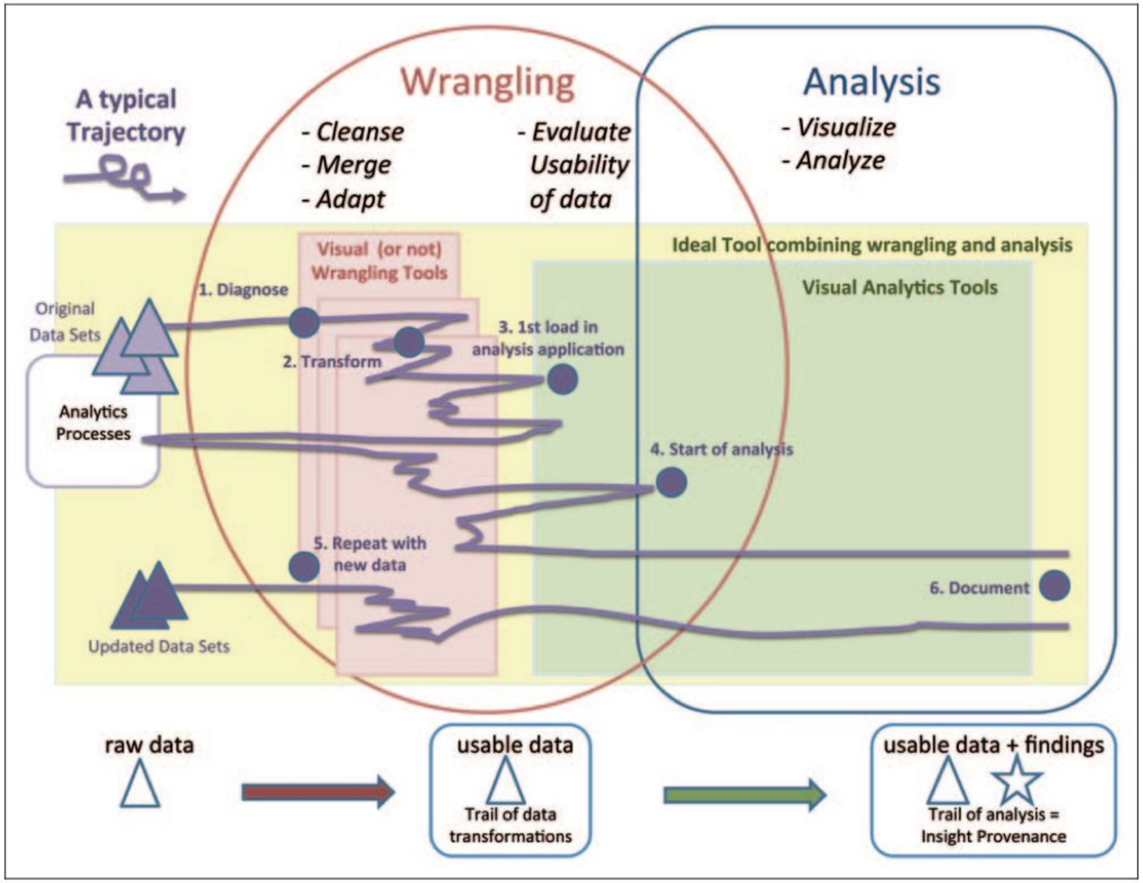
Typical Data Science Workflow



- From [Data Science Workflow]()

Where Data Scientist Spends Most Time

- ['Janitor Work' in Data Science]
- [Research Directions in Data Wrangling]
  - Estimates that 80-90% of the work is in data cleaning and wrangling



What a Data Scientist Should Know

- From: [How to hire a data scientist]
- Data grappling skills
  - How to move data around and manipulate it with some programming language
  - Scripting languages (e.g., Python)
  - Data storage tools like relational databases, key-value stores
  - Programming frameworks like SQL, Hadoop, Spark, etc.
- Data visualization experience
  - How to draw informative pictures of data
  - Many tools (e.g., D3.js, plotting libraries)
  - Harder question is knowing what to draw
- Knowledge of statistics
  - E.g., error-bars, confidence intervals
  - Python libraries; Matlab; R
- Experience with forecasting and prediction
  - Basic machine learning techniques
- Communication skills
  - Communicate the findings

We focus here!!!

UMD DATA605 - Big Data Systems

Course Intro

Big data

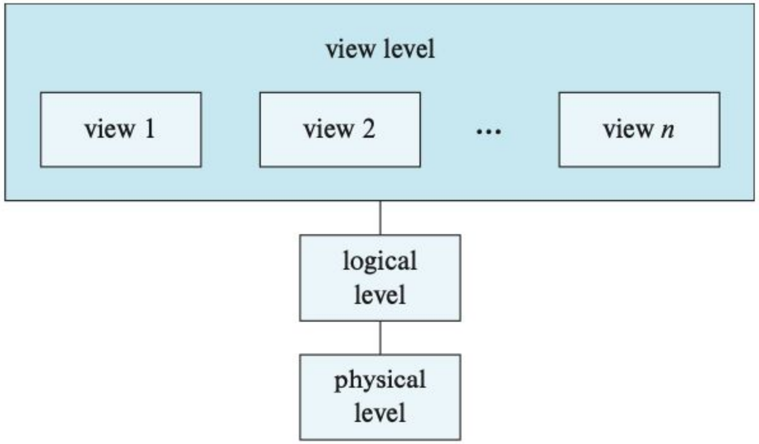
Data Models

Data Models

- **Data modeling**
  - = process of representing and capturing the structure and properties of real-world entities
  - Process of abstraction: real-world → representation
- **Data model**
  - = description of how data is represented (e.g., relational, key-value) and accessed (e.g., allowed operations, how to query)
  - E.g., schema in a DB describe a specific collection of data, using a given data model
- **Why do we need data model?**
  - Need to know the structure of the data (to some extent) to be able to write general purpose code
  - Allow to share data across programs, organizations, systems
  - Need to integrate information from multiple sources
  - Preprocess data to make access efficient (e.g., building an index on a data field)

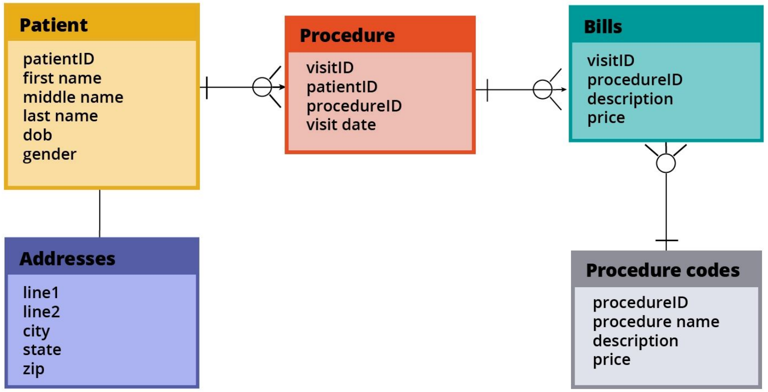
Multiple Layers of Data Modeling

- **Physical layer**
  - How is the data physically stored
  - How to represent complex data structures (e.g., B-trees for indexing)
- **Logical layer**
  - Type of information stored
  - Entities
  - Entity attributes
  - Relationships among the above
- **Views**
  - Restrict information flow
  - Security and/or ease-of-use



Data Models: Logical Layer

- **Modeling constructs**
  - A collection of concepts used to represent the structure in the data, e.g.,
    - Types of entities
    - Entity attributes
    - Types of relationships between entities
    - Types of relationships between attributes



- **Integrity constraints**
  - Constraints to ensure data integrity
    - Goal: avoid errors and data inconsistencies
    - E.g., a field can’t be empty, is an integer
- **Manipulation constructs**
  - E.g., insert, query, update, delete data

Examples of Data Models

- We will cover:
  - Relational model (SQL)
  - Entity-relationship (ER) model
  - XML
  - Object-oriented (OO)
  - Object-relational
  - RDF
  - Property graph

- Serialization formats are also data models
  - CSV
  - Parquet
  - JSON
  - Protocol Buffer
  - Avro / Thrift
  - Python Pickle

Good Data Models

- We would like a data model to be:
  - Expressive
    - Capture real-world data well
  - Easy to use
  - Good performance
- Tension between the above characteristics
  - More powerful models
    - Can represent more datasets
    - Harder to use, to query
    - Less efficient
- The evolution of data modeling tools is an attempt to capture the structure in the data
  - Structured data → Relational DBs
  - Semi-structured web data → XML
  - Unstructured data → NoSQL DBs

Data Independence

- **Logical data independence**
  - Can change the representation of data without changing programs that operate on it
- **Physical data independence**
  - Can change the layout of data on disk and programs won't change
    - Index the data
    - Partition / distribute / replicate the data
    - Compress the data
    - Sort the data

Databases: A Brief History

- **1960s: Early beginning**
  - Computers finally become attractive technology
  - Enterprises start adopting computers
  - Most applications initially used their own data stores
    - Each application had its own format
    - Data was basically unavailable to other programs
  - **Database**: term coined in military information systems to denote "shared data banks" by multiple applications
    - Define a data format
    - Store it as a "data dictionary" (schema)
    - Implement general-purpose "database management" software to access data
  - Issues:
    - How to write data dictionaries?
    - How to access data?
    - Disadvantages of integration: integrity, security, privacy concerns
    - Who controls the data?

Databases: A Brief History

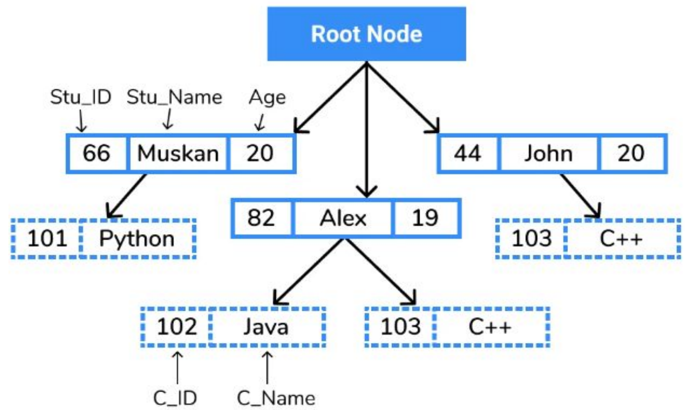
- Birth of "hierarchical model" and "network model"
  - Both allowed connecting records of different types
  - E.g., connect accounts with customers
  - Network model attempted to be very general and flexible
- IBM designed IMS hierarchical database in 1966 for the Apollo space program
  - Predates *hard disks*
  - Still around today!
  - *.. more than 95 percent of the top Fortune 1000 companies use IMS to process 50 billion transactions a day and manage 15 million gigabytes of critical business data* (from IBM Website on IMS)
- Cons:
  - Hierarchical / network models exposed too much of the internal data (e.g., structures / pointers to the users)
  - Leaky abstraction

Relational, Hierarchical, Network model

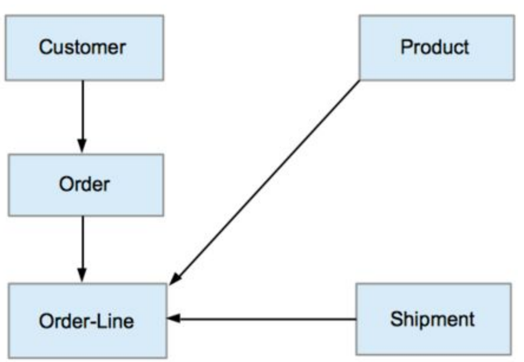
- **Relational model**
  - Data is represented as tuples grouped in relation
  - Omnipresent SQL

Customer ID	Tax ID	Name	Address	[More fields…]
1234567890	555-5512222	Ramesh	323 Southern Avenue	…
2223344556	555-5523232	Adam	1200 Main Street	…
3334445563	555-5533323	Shweta	871 Rani Jhansi Road	…
4232342432	555-5325523	Sarfaraz	123 Maulana Azad Sarani	…

- **Hierarchical model**
  - Data is organized into a tree-like structure
    - Each record has one parent record and many children
    - Records connected through links
  - Resurgence in 1990s with XML DBs



- **Network model**
  - Data is organized in a graph
    - Each record can have multiple parent and child records
  - Resurgence in 2010s with graph DBs

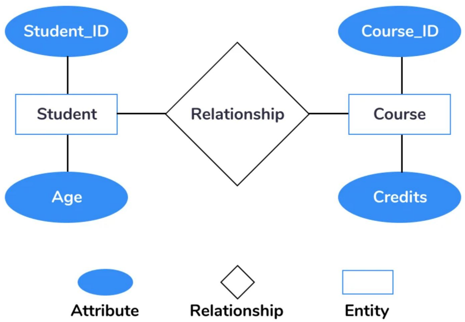


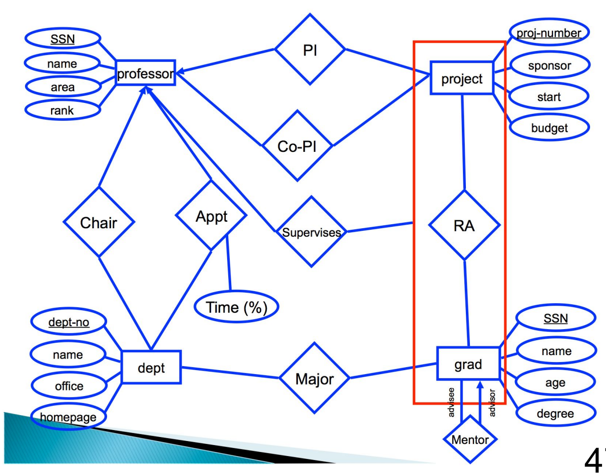
Databases: A Brief History

- **1970s: Relational model**
  - Set Theory, First-order predicate logic
    - Ted Codd developed the Relational Model
  - Elegant, formal model
    - Provided almost complete *data independence*
    - Users didn't need to worry about how the data was stored, processed
  - High level query language
    - SQL based on relational algebra
  - Notion of *normal forms*
    - Allowed one to reason about
    - Remove redundancies
- Influential projects: INGRES (UC Berkeley), System R (IBM)
  - Didn't care about IMS/IDMS compatibility (as IBM had to)
- Many debates between Relational Model vs Network Model proponents

Entity-Relationship Model

- 1976: Peter Chen proposed "Entity-Relationship Model"
- Data model describing knowledge in terms of entities and relationships
- **Entities** are physical or logical objects that can be uniquely identified
  - "Nouns"
- **Relationships** between entities
  - "Verbs"
- An ER model can be mapped onto a relational DB
  - Entities, relationships -> tables





Databases: A Brief History

- **1980s: Widespread acceptance of relational model**
  - SQL emerged as a standard, in large part because of IBM's backing
  - Enriching the expressive power of relational model
    - Set-valued attributes, aggregation, etc.
- Late 80's
  - Object-oriented DB
    - Store objects instead of tables
    - Get around *impedance mismatch* between programming languages and databases
  - Object-relational DB
    - Allow user-defined types
    - Get many benefits of object-oriented while keeping the essence of relational model
  - No expressive difference from pure relational model

Object-Oriented

- OOP is a data model
  - Object behavior is described through data (stored as fields) and code (in the form of methods)
- Composition
  - Aka `has-a` relationships
  - E.g., an Employee class has an Address class
- Inheritance
  - Aka `is-a` relationships
  - E.g., an Employee class derives from a Person class
- Polymorphism
  - Code executed depends on the class of the object
  - One interface, many implementations
  - E.g., `draw()` method on a Circle vs Square object, both descending from Shape class
- Encapsulation
  - E.g., private vs public fields / members
  - Prevents external code from being concerned with inner workings of an object

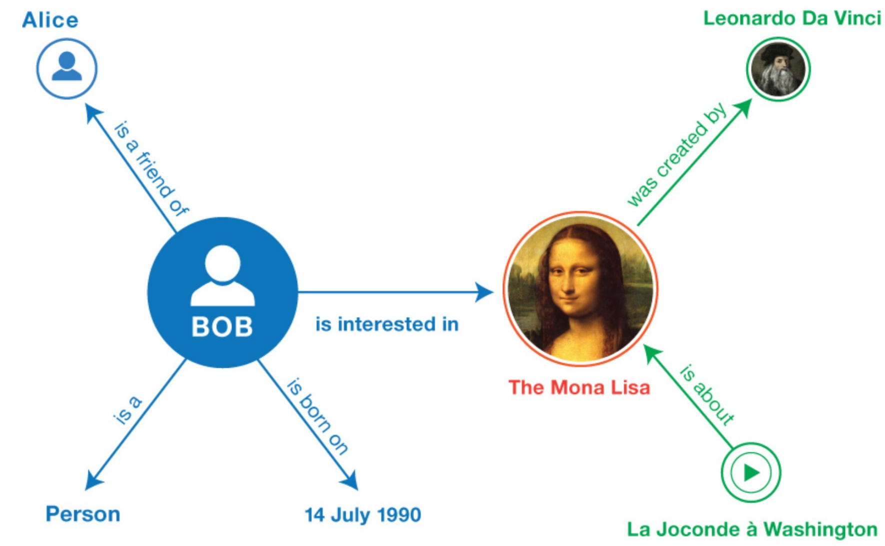
Databases: A Brief History

- **Late 90's-today**
- Web emerges
- XML: eXtensible Markup Language
  - Intended for *semi-structured* data
  - Tree-like structure
  - Flexible schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited by XMLSpy -->
 <CATALOG>
     <CD>
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
    </CD>
    <CD>
      <TITLE>Hide your heart</TITLE>
      <ARTIST>Bonnie Tyler</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>CBS Records</COMPANY>
      <PRICE>9.90</PRICE>
      <YEAR>1988</YEAR>
     </CD>
     ...
```

Resource Description Framework

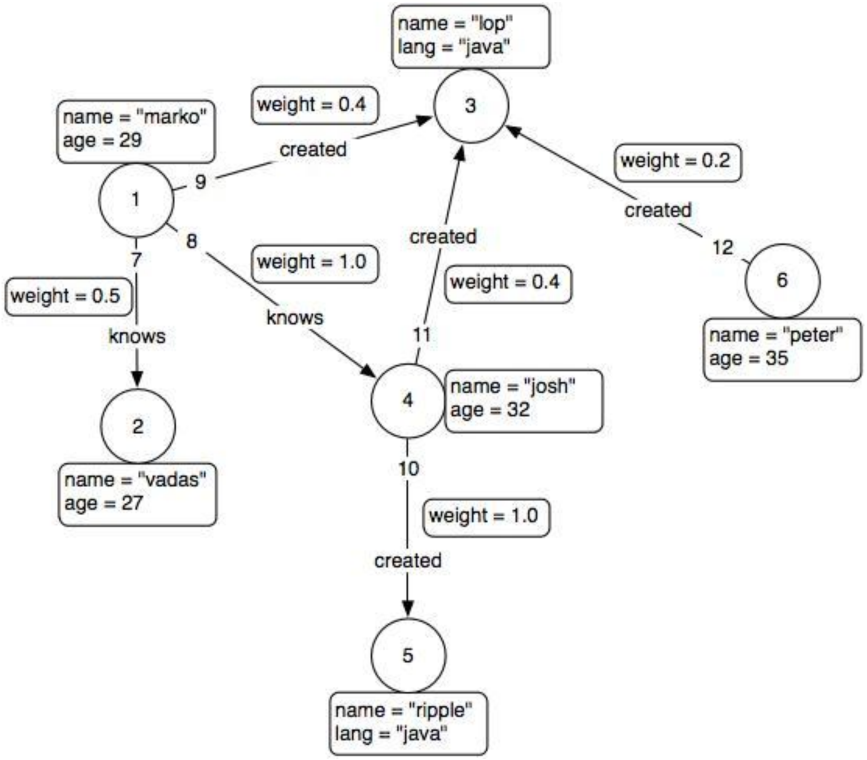
- Aka RDF
- Key construct: a "subject-predicate-object" triple
  - E.g.,
    - subject=sky
    - predicate=has-the-color
    - object=blue
- Can be mapped to a labeled, directed multi-graph
  - More general than a tree
- Typically stored in:
  - Relational DBs
  - Dedicated "triple-stores" DBs




```
01    <http://example.org/bob#me>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person> .
02    <http://example.org/bob#me> <http://xmlns.com/foaf/0.1/knows>
<http://example.org/alice#me> .
03    <http://example.org/bob#me> <http://schema.org/birthDate>
"1990-07-04"^^<http://www.w3.org/2001/XMLSchema#date> .
04    <http://example.org/bob#me>
<http://xmlns.com/foaf/0.1/topic_interest>
<http://www.wikidata.org/entity/Q12418> .
05    <http://www.wikidata.org/entity/Q12418>
<http://purl.org/dc/terms/title> "Mona Lisa" .
06    <http://www.wikidata.org/entity/Q12418>
<http://purl.org/dc/terms/creator>
<http://dbpedia.org/resource/Leonardo_da_Vinci> .
07
<http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B81
3C5F9AA4D619> <http://purl.org/dc/terms/subject>
<http://www.wikidata.org/entity/Q12418> .
```

Property Graph Model

- Graph:
    - with vertices and edges
    - with properties associated with each edge and vertex
- Typically stored in:
    - Relational DBs
    - Graph DBs



Serialization Formats

- Programs need to send data to each other (on the network, on disk)
  - E.g., Remote Procedure Calls
  - Several recent technologies based around schemas
    - JSON, YAML, Protocol Buffer
- Serialization formats are data models

JSON

- JSON = JavaScript Object Notation
- Data is nested dictionaries and arrays
- Very similar to XML
  - More human-readable
  - Less boilerplate
  - Executable in JavaScript (and Python)


```
{
   "firstName": "John",
   "lastName": "Smith",
   "isAlive": true,
   "age": 25,
   "height_cm": 167.6,
   "address": {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": "10021-3100"
   },
   "phoneNumbers": [
      {
      "type": "home",
      "number": "212 555-1234”
      },
      {
      "type": "office",
      "number": "646 555-4567"
      }
   ],
   "children": [],
   "spouse": null
}
```

Protocol Buffers

- Developed by Google
- Open-source
- Represent data structures in a
  - Language agnostic
  - Platform agnostic
  - Versioning
- Schema is mostly relational
  - Optional fields
  - Types
  - Default values
  - Structures
  - Arrays
- Schema specified using a .proto file
- Compiled by protoc to produce C++, Java, or Python code to initialize, read, serialize objects

```
message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    optional string number = 1;
    optional PhoneType type = 2;
  }

  repeated PhoneNumber phones = 4;
}
```

```
import addressbook_pb2
person = addressbook_pb2.Person()
person.id = 1234
person.name = "John Doe"
person.email = "jdoe@example.com"
phone = person.phones.add()
phone.number = "555-4321"
   phone.type =
   addressbook_pb2.Person.HOME
```

Serialization Formats

- Avro
  - Richer data structures
  - JSON-specified schema

```
{
        "namespace": "example.avro",
        "type": "record",
        "name": "User",
        "fields": [
                {"name": "name", "type": "string"},
                {"name": "favorite_number", "type": ["int", "null"]},
                {"name": "favorite_color", "type": ["string", "null"]}
        ]
 }
```

- Thrift
  - Developed by Facebook
  - Now Apache project
  - More languages supported
  - Supports exceptions and sets

Class Project: TODOs

- Clone / look DATA605 - Class project
- Look at Sorrentum GitHub repo
  - Two project examples
    - Price data from Binance
    - Reddit data
- Complete the student survey
- Take a look at list of projects
- By next lesson:
  - Dr Saggese + TA create teams and assign a project