Applied Data Science - Notes

Dom Hutchinson

February 17, 2021

Contents

1 Introduction

Remark 1.1 - *Types of Data*
Data comes in many forms including, but not limited to, the following

- Dense & Sparse data.
- Structured/Relational Data.
- Numerical; Categorical; Ordinal; or Boolean.
- Test (Emails, Tweets, Articles).
- Records (User-Level Data, Timestamped Event Data, Log Files).
- Geo-Based Location Data.
- Data-Time Data.
- Network Data.
- Sensor Data.
- Images and Video.
- Audio and Music.

Remark 1.2 - *Big & Small Data*
Whether a dataset is big or small depends on the computational-resources available, and thus will vary over time. Here are some ways to evaluate this

	Big Data	**Small Data**
Data Condition	Always unstructured, not read for analysis, many relational database tables that need to be merged.	Ready for analysis, flat file, no need to merge tables.
Location	Cloud, offshore, SQL server etc.	Database, local PC.
Data Size	Over 50k variables, over 50k individuals, random samples, unstructured	File that is in a spreadsheet, that can be viewed on a few sheets of paper.
Data Purpose	No intended purpose.	Intended purpose for data collection.

Remark 1.3 - *What is Data Science?*
Data-Driven Science. An interdisciplinary field about scientific processes and systems to extract knowledge or insights from data in various forms.

Data science incorporates fields from: Mathematics, Computer Science; &, Domain Expertise.

Remark 1.4 - *Motivating Applications*
Data science is motivated by its applications. Here are some examples of such applications

- *Amazon* use recommender systems to suggest products to customers.
- *Energy Companies* use data science to try and predict future usage of customers, so that resources can be applied efficiently.

- *Agriculture* use sensors in fields to collect data in order to monitor crops and predict weather.
- *Healthcare* use sensors in homes to monitor the health of people over long periods of time (especially when the person cannot go to the hospital).

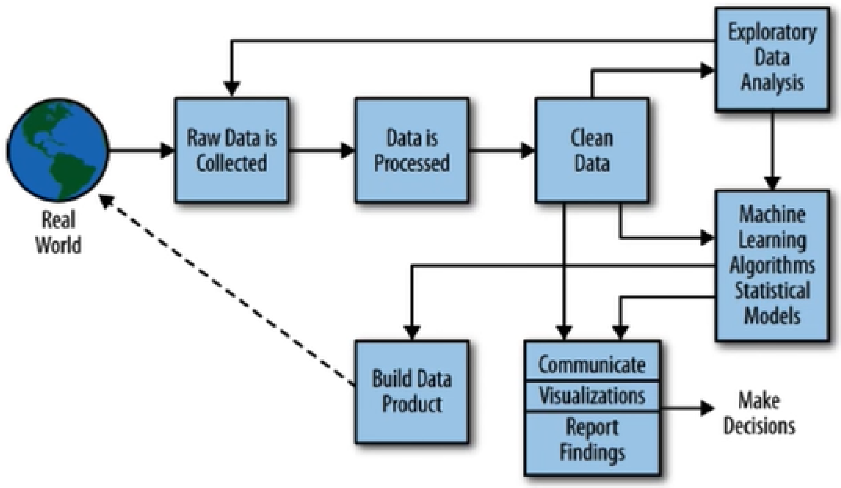
Real World
Raw Data is Collected
Data is Processed
Clean Data
Exploratory Data Analysis
Machine Learning Algorithms Statistical Models
Build Data Product
Communicate Visualizations Report Findings
Make Decisions

Figure 1: The pipeline for approaching problems in data science.

Proposition 1.1 - *Data Science Pipeline*
See `Figure 1` for a pipeline for approaching data science problems.

2 Data Ingress & Pre-Processing

2.1 Data Structures

Proposition 2.1 - *Native Python Data Structures*
Here are some data structures which are native to python and are popular in data science

- `list` - List of elements of varying types.
- `set` - List of unique elements of varying types.
- `dict` - Key-Value pairings.

Proposition 2.2 - *Non-Native Python Data Structures*
Here are some data structures which are not native to python but are popular in data science.

- `np.array`.
- `pandas.DataFrame`.

2.2 Data Formats

Definition 2.1 - *Object Persistence*
Object Persistence is the process of ensuring that the objects which are created are kept through multiple sessions. This comes in two stages

i). *Serialisation* - Translating data structures or objects from memory into a format which can be stored.

ii). *Deserialisation* - The inverse. Translating data structures which have been stored, into memory.

Remark 2.1 - *Bespoke Serialisation & Deserialisation*
Bespoke serialisation and deserialisation methods can be crafted manually. (e.g. Instantiating an output file; Writing each element of a list to a different line in the file; Closing the file.)

However, there are limitations to bespoke methods:

- Methods are specific to each use case (not standardised).
- Methods may not be robust.
- Methods require testing against many test cases.
- Object metadata is not encoded.

These limitations are rarely a problem in very controlled environments.

Definition 2.2 - *Comma-Separated Values (CSV)*
Comma-Separated Values (CSV) files are well suited to tabular data (inc. matrices). Each line of a *CSV File* stores one row of the table, and each element in a row is separated by a comma.

CSV Files are generally very readable, as well as time- and space-efficient for tabular data.

Remark 2.2 - *Using CSV Files*
As *CSV Files* are very popular, there are many library methods which can interact with them. Including reading & writing to and from memory (e.g. `pandas.read_csv`).

Definition 2.3 - *JavaScript Object Notation (JSON)*
JavaScript Object Notation[1] (JSON) is a standardised syntax for storing & exchanging data, used for serialisation. JSON uses text files which are human-readable and `dict`-like (i.e. have value-key pairs). JSON is designed to be simple so is a very robust language & suits many purposes.[2]

Limitations of JSON are that a specific conversion process may be required to convert a JSON file into objects in memory, and JSON files can become large due to key repetition.

Definition 2.4 - *Hierarchical Data Format (HDF5)*
Hierarchical Data Format (HDF5) is a standardised format for serialisation. HDF5 is a binary format and tries to mimic file system-like access. HDF5 files have the following three components

i). *Datasets* - Array-like collections of data. Thousands of datasets can be stored in a single file, and can be categorised and tagged however you want.

ii). *Groups* - Folder-like structures which groups datasets & other groups.

iii). *Metadata* - Information which pertains to all the datasets. (e.g. author, edit data & version).

[1]JSON is not just compatible with JavaScript and is common for many languages & APIs
[2]`https://jsonlint.com/` is a useful site for JSON validation.

HDF5 files are ideal for large numerical data sets, and can easily be manipulated by `numpy`. HDF5 files support a variety of transparent storage features (inc. compression, error-detection and chunked I/O), which `numpy.array` do not.[3]

Proposition 2.3 - *HDF5 files vs Traditional File Systems*
There are a few key differences between *HDF5 Files* and *Traditional File Systems.*

i). *HDF5 Files* are portable, as the entire structure is contained in the file independent of the underlying file system.[4]

ii). Datasets in *HDF5 Files* are all homogeneous hyper-rectangular numerical arrays, whereas files in traditional file system can be anything.

iii). Metadata can be added to groups in *HDF5 Files.* This is not possible in traditional file systems.

Remark 2.3 - *Other Standardised Serialisation Method*
XML, *Protocol Buffers* & YAML are other popular standardised serialisation methods.

2.3 Web-Scraping & APIs

Remark 2.4 - *Terms of Use*
Web Scraping should be done within the website's terms of use.

Definition 2.5 - *Web Scraping*
Web Scraping is the practice of collecting data from websites. This can take many forms, but typically involves taking a raw webpage and parsing the desired data.

Proposition 2.4 - *Approaching Web Scraping*
To perform *Web Scraping* successfully, you need a good idea of how a webpage is structure. Typically webpages are based around a `html` file, which are well structured[5]. Identifying combinations of tags, classes & ids in the `html` file can help locate the desired data.

It is harder, sometimes impossible, to navigate poorly designed websites as they are less structured and inconsistent.

Remark 2.5 - *Tools for Web Scraping*
Web Scraping technologies need to be tolerant to several artefacts of real-world data (known as *"wrangling"*) as-well-as errors in the website.

Some popular tools for *Web Scraping* are

- `BeautifulSoup` - A python library for parsing `XML` & `HTML`.
- `scrapy` - A python library. Generally faster than `BeautifulSoup`.
- `Selenium` - A web-browser plugin, generally used to test web services.

[3]`http://docs.h5py.org/en/latest/quick.html` provides a quick-start guide to using HDF5 files in python.
[4]However, it does depend on the HDF5 library.
[5]Webpages are often interpreted to have a tree structure, with each tag being a node

Definition 2.6 - *Web APIs*
Web APIs greatly simplify *Web Scraping* by provide a portal for explicit data acquisition, and are generally less prone to the issues which arise when *Web Scraping.*[6]

- The code running *Web APIs* is optimised for data requesting & retrieval. It does not waste time on visualisation or aesthetics. This means the bandwidth required for an *API* request is much lower than for a similiar *Web Scraping* process (As images etc. do not need to be loaded).
- *Web API* querying is robust, reliable, well maintained and documented with a static schema (`HTML`-based *Web Scraping* is not).
- *Web APIs* use standardised *Serialisation Tools* (e.g. JSON).
- *Web APIs* have already extracted and organised the desired data, however this does mean the user can only access what the operator will allow. This is much better than `HTML`-based *Web Scraping* where you rely upon fickle naming conventions of tags.

Definition 2.7 - *RESTful APIs*
Representational State Transfer APIs (REST/RESTful) are a popular form of *Web API* and generally require an *API Key* to access data.

Request to *RESTful APIs* generally involves constructing a URL which contains your keys and the parameters of your query.

Remark 2.6 - *Regular Expression (RegEx)*
Regular Expression (ReGex) queries are useful for extracting data, either while web scraping or from API requests.[7] Python has the `re` library for *RegEx* queries, two popular methods from this library are

i). `re.match` - Attempts to match a *RegEx* pattern to the whole string.

ii). `re.search` - Searches for the first occurrence of a *RegEx* pattern in a string.

3 Privacy

3.1 IRL Examples

Remark 3.1 - *AOL Data Incident, 2006*
In 2006, *AOL Research* released to the public, for the purpose of research, a text file containing 20mn search terms from over 650k users, over a 3-month period.

The users were anonymised in the in the data, but personally identifiable information was present in many of the search terms.

AOL Research retracted the document shortly after publishing.

Remark 3.2 - *Netflix Prize, 2007*
In 2007, *Netflix* ran a competition where \$1mn was offered to anyone who was able to produce a system which outperformed their existing recommender system by at least 10%.

The training data offered by *Netflix* contained ¡user,movie,date,grade¿ and customer ids were anonymised.

[6]See `https://github.com/public-apis/public-apis` for a categorised list of public APIs.
[7]`http://pythex.org/` is a website which can perform *RegEx*.

It was later found that some users had their *Netflix* and *IMDb* accounts linked (so reviews were posted on both). This meant the training data could be partially de-anonymised. This led to lawsuits and *Netflix* cancelling future competitions.

Remark 3.3 - *Pay Attention*
The moral of these examples is that you have to be really careful and diligent when anonymising data as there may be non-obvious ways for people to de-anonymise it. (Often through little fault of your own).

3.2 Security & Statistical Database

Definition 3.1 - *Statistical Database*
A *Statistical Database* is a database for statistics. They can come in several forms

- *Tabular Data* - Tables with counts or magnitudes.
- *Queryable Databases* - On-line databases which accept statistical queries (e.g. min, max, sum etc.)
- *Microdata* - Files where each record contains information about an individual.

Remark 3.4 - *Privacy Trade-Off*
Statistical Databases have to trade-off privacy and functionality. Generally, more privacy means less functionality.

Definition 3.2 - *Identifiers*
Identifiers are attributes that unambiguously identify someone (e.g. passport number, NI number, name etc.)

Definition 3.3 - *Quasi-Identifiers*
Quasi-Identifiers are attributes which identify someone, but there is some ambiguity. Often, having multiple *Quasi-Identifiers* are enough to confidently identify someone. (e.g. address, age, gender).

Definition 3.4 - *Confidential Attributes*
Confidential Attributes are attributes which contain sensitive information about an individual (e.g. salary, religion, medical conditions etc.)

Definition 3.5 - *Non-Confidential Attributes*
Non-Confidential Attributes are attributes which do not contain sensitive information about an individual.

Remark 3.5 - *Anonymisation, Identifiers & Quasi-Identifiers*
When anonymising a data set:

- *Identifiers* need to be suppressed from the data sets (i.e. removed).
- *Quasi-Identifiers* hold disclosure risk as they can often not be suppress due their high analytical value, but can often be linked with other non-anonymised datasets to de-anonymise your dataset.

Definition 3.6 - *Attribute Disclosure*
Attribute Disclousure is when the value of a confidential attribute is made available, and in doing so the individual is easier to identifier (compared to the value not being disclosed).

Definition 3.7 - *Identity Disclosure*
Identity Disclosure is when an entire record in an anonymised data set can be linked with that individual's identity.

Definition 3.8 - *Membership Disclosure*
Membership Disclosure is whether or not data about an individual is contained in a dataset.

Definition 3.9 - *External Attack on a Table*
An *External Attack* on a table occurs when another dataset is released which contains data which comprises more table, or can be combined with your table to comprise others.

Definition 3.10 - *Internal Attack on a Table*
An *Internal Attack* on a table occurs when those within the table (i.e. with access to it and know which data represents them) are able to deduce which data belongs to other people by a process of elimination.

Only really possible when the table has few respondents.

Definition 3.11 - *Dominance Attack on a Table*
An *Internal Attack* on a table occurs a (or a few) respondents dominate the contribution in a particular cell of a magnitude table, these *Dominant Respondents* are then able to upper-bound the contributions from the rest due it being easier to differentiate other users from them.

Proposition 3.1 - *Combatting Table Attacks*
There are two main approaches to combatting *Table Attacks*

i). *Non-Perturbative* techniques which do not modify the values in the cells, but they may suppress or recode them (e.g. cell suppression, recoding of categorical attributes).

ii). *Perturbative* techniques do modify the values in the cells (e.g. controlled rounding).

Proposition 3.2 - *Combatting Database Attacks*
There are three main approaches to combatting attacks on databases, all focused around adjusting/controlling queries

i). *Query Input Perturbation* - Perturbation is applied to records on which queries are computed.

ii). *Query Output Perturbation* - Perturbation is applied to the results of a query.

iii). *Query Restriction* - The database refuses certain queries.

iv). *Camoflage* - Returns answers which have been made non-exact in some deterministic fashion.

Remark 3.6 - *Differential Attack*[8]

[8]Not proper name.

Suppose you know that someone is going to be changing doctors soon (and thus will be removed from the doctors database), if you copied the database before and after this person moves you could likely deduce details about the individual by comparing the two versions.

How do we prevent this?

Definition 3.12 - *Differential Privacy*
Consider a learner implements a summary statistic A (e.g. sum of all elements); an adversary proposes two datasets S and S' which differ by only one row; and a test set Q. The learner wants the summary statistic to have the same value if only one row has changed.

Summary statistic A is *ε-Differentially Private* iff

$$\left|\ln\left(\frac{\mathbb{P}(A(S)\in Q)}{\mathbb{P}(A(S')\in Q)}\right)\right|\leq\varepsilon$$

Note that *Differential Privacy* is a condition on the release mechanism of the data A, not on the data S, S' itself.

Remark 3.7 - *Differential Privacy in Practice*
In practice we avoid differential privacy issues by adding noise to our responses (either to all responses, or incorrectly responding to some queries).

3.3 k-Anonymity

Definition 3.13 - *k-Anonymity*
A dataset satisfies *k-Anonymity* if each combination of values of *Quasi-Identifier Attributes* in the table are shared by at least k records.

Measuring *k-Anonymity* requires a brute force approach which compares every pair of records in the table.

Proposition 3.3 - *Achieving k-Anonymity*
There are two main approaches to achieving *k-Anonymity*:

- *Suppression* - Replace the values of attributes with an asterisk.
- *Generalisation* - Values of attributes are replaced with a broader category. (e.g. age-ranges rather than exact age)

Proposition 3.4 - *k-Anonymity Attacks*
k-Anonymity does not protect against attribute disclosure attacks in general if the values of *Confidential Attributes* are very similar in the group of k records who share *Quasi-Identifier* values.

Here are two specific attacks that can happen

- *Homogeneity Attack* - If all records in the set of k identical records have the same *Sensitive Value*, then the value can likely be predicted (and would comprise all k).
- *Background Knowledge Attack* - If associations exist between one or more *Quasi-Identifier Attributes* and a *Sensitive Attribute* then the set of possible values of the *Sensitive Attribute* is reduced.

Definition 3.14 - *l-Diversity*
l-Diversity is an extension of *k-Anonymity.*

l-Diversity requires that the values of all *Confidential Attributes* within any group of k-anonymous records contain at least l clearly distinct values.

Proposition 3.5 - *Limitations of l-Diversity*
Here are two limitations of *l-Diversity*

- Not every value shows equal sensitivity. (A rare positive indicator for a disease may provide more information than a common negative indicator).
- It ensures diversity of sensitive values in each group, but it does not recognise that values may be semantically similar.

Definition 3.15 - *t-Closeness*
t-Closeness requires that the distribution of the confidential attribute within a group of k records is similar to the distribution of the confidential attribute in the entire data set (at most distance t between the two distributions).

Remark 3.8 - *Tools for Anonymisation*
Here are three popular tools for anonymisation

i). *ARX* - Applies k-anonymity, l-diversity and t-clossness in Java.

ii). *Argus* - Creates safe microdata files.

iii). *SdcMicro* - Provides disclosure control methods for anonymisation and risk-estimation.

3.4 Other Approaches

Definition 3.16 - *Privacy-Preserving Data Mining, PPDM*
Privacy-Preserving Data Mining (PPDM) seeks privacy for the data's owner when several owners wish to co-operate without giving away their data to each other

Definition 3.17 - *Private Information Retrieval*
Private Information Retrieval (PIR) seeks user privacy to allow the user of a database to retrieve information without the database knowing which item was retrieved.

4 Storage & Management

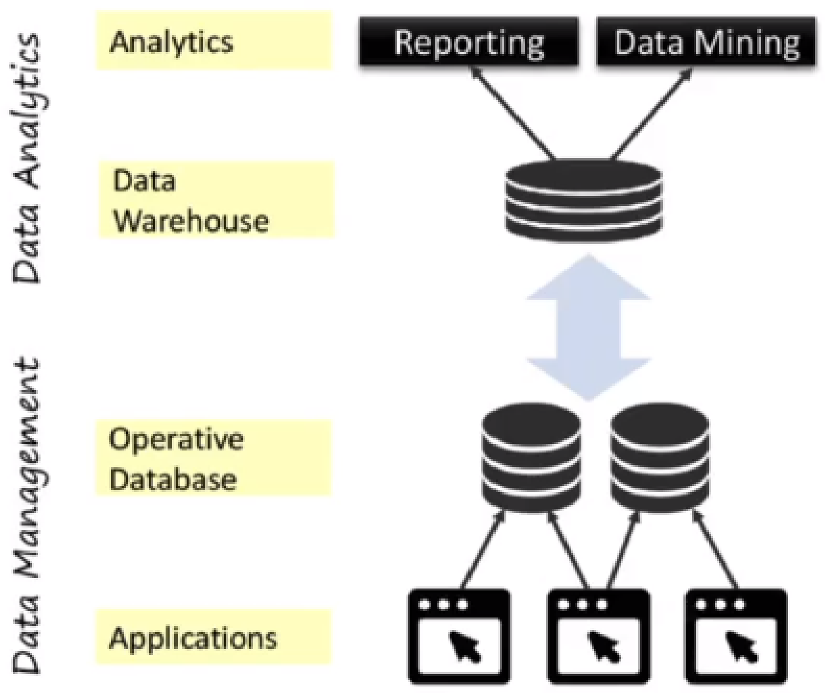

Figure 2: Typical data architecture.

Proposition 4.1 - *Typical Data Architecture*
Organisation tend to have two main data stores

i). A *Data Warehouse* which contains copies of historic data. Data analysis is performed on this data as it is not critical to the operation of the organisation. *Data Warehouses* usually allow data analysts read-only access.

ii). Several *Operative Databases* which the organisation's core products use and rely on. Data analysis should not be performed on this databases directly as it could interfere with customers (or worse, break the company's products).

See `Figure 2` for an illustration.

Definition 4.1 - *Datalake*
TODO

4.1 Storage Technologies

Definition 4.2 - *BLOB Storage*
A *Binary Large Object* (BLOB) storage is typically used for large, unstructured files (e.g. images & backup files).

Due to being unstructured, *BLOB* files are not quick to query and are best used just as a record of the data.

BLOB files are generally accessed using a REST-API or an SDK, they are not mountable as a disk and drivers need to be build into applications in order for the app to acess the *BLOB Storage*. *Spark* is a tool used by many datalakes for *BLOB* storage.

Data storage providers typically offer two tiers of *BLOB* storage

i). *Hot Access* - Data is accessed frequently.

ii). *Cold Access* - Data is not accessed very often. (Cheaper, but less availability).

Definition 4.3 - *Disk Storage*
Disk Storage[9] are an alternative form of storage, but are essentially a facade for *BLOB-backed storage* and tend to be up to 1TB in size.

Disk Storage can be mounted to a virtual machine. Once mounted and formatted, the file is accessible like any other local file, but with greater throughput than standard-file-storage (60/sec per blob)[10]

Disk Storage is ideal when you want to move applications which you were using locally into cloud storage, and you want the data to be decoupled from the virtual machine[11]. But, only one person can access a *Disk Storage* at a time.

Definition 4.4 - *File Storage*
File Storage[12] uses bytes across a network and a using a protocol (e.g SMB), this creates the illusion of the data being a local files. This means several instances of an application can access the data at the same time.

File Storage can be upto 5TB each and has a maximum throughput of 60MB/sec across a share[13]. Throughput is better in Local-Area networks.

File Storage is backed by *BLOB Storage* and thus is just as durable.

Capabilities such as concurrency and file locking can be introduced into *File Storage*, availability depends on the protocol being used.

4.2 Databases

Remark 4.1 - *Database Classification*
There are two features typically used to classify databases

i). The model they use to store data.

ii). Their "Consistency/Availability" trade-off.

[9]AKA *Block Storage*.
[10]Throughput is lower than *BLOB Storage*.
[11]As, otherwise, if the machine was to die then you would loose your data too!
[12]AKA *Network Attached Storage* (NAS).
[13]Slower than BLOB & Disk

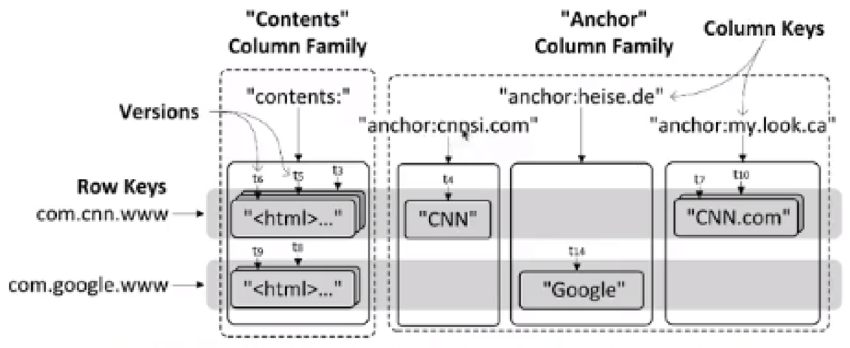

Figure 3: Example of data may be structured in a *Colum Store.*

Definition 4.5 - *Relational Database*
TODO

Definition 4.6 - *Database Models*
Databases typically use one of the following five models to store & organise data.

i). *Key-Value Stores* - A dictionary (keys map to values in a deterministic fashion). These databases only support get & put operations[14], and all data is stored as a *BLOB.*

Key-Value Stores are *schemaless* meaning all assumptions about the structure of the data is implicitly encoded into the application log (and not explicitly defined by a data-definition language).

Key-Value Stores are easy to partition, query, have low latency and high throughput.

ii). *Wide-Column Stores* - A distributed, multi-level, sorted map. This means that *Wide Column Stores* have several layers of "keys": the first layer, called "Row Keys", identify rows which contain a given *key-value pair*; The second layer is known as "column keys".[15] (Columns can be grouped into "Column Families". Values in the same column family and same row are stored on the same disk).

Wide-Column Stores are space efficient as null values take zero space, and store data in a lexicographic order wrt their keys. Good compression rate allows for efficient querying of row subsets. Versions of each cell can be stored.

iii). *Document Stores* - An implementation of *Key-Value Stores* but each value (known as a "document") has to be in a semi-structured format (e.g. JSON). An entire document can be fetched using its unique key. *Document Stores* allow for more complex queries than *Key-Value Stores*, such as: retrieval of parts of documents; aggregation; query-by-example; and, full-text search.

iv). *Graph Databases* - Store data sources in nodes and uses edges to define relationships between data sources. This is really good for complex relationships, but not very scalable as *Graph Partitioning* is NP-Hard.

v). *Search Engines*[16] - Essentially an inverted index, with some meta-data and query optimisation. Often will link from a value to the databases which contain that value.

[14]There is no support for operations beyond simple CRUD (Create-Read-Update-Delete)
[15]See `Figure 3`.
[16]Not strictly a database

Remark 4.2 - *Choosing Database Model*
When choosing which model of database to use, we are generally trading between the models ability to handle "size" and "complexity". Generally *Key-Value Stores* are best for the simplest datasets, then *Column Stores*, then *Document Stores* and finally, *Graph Databases* are typically best for handling complex datasets (but do no scale well).

A *Relational Database* is the default model to use for most projects, as they perform well for both size & complexity up to reasonably large projects.

Theorem 4.1 - *CAP Theorem*
The *CAP Theorem*[17] states that a distributed system or database can guarantee at most two of the following three properties at any one time

- *Consistency* (C) - All users have the same view on the data at all times, with minimal *Latency.*
- *Availability* (A) - All users can always read and write to the database.
- *Partition-Tolerant* (P) - The system works well across physical network partitions (even if a machine fails or the cluster is split).

Proposition 4.2 - *Database Consistency-Availability Trade-Off*
Databases fulfil one of the following

- *Available & Partition Tolerant* (AP). (Never a relational database).
- *Consistent & Partition Tolerant* (CP). (Never a relational database).
- *Not Partition Tolerant* (CA).

Definition 4.7 - *Sharding*
Sharding specifies how data should be partitioning data across several nodes. This is an important technique in data storage as data is often stored in distributed networks.

Here are some popular approachs to *Sharding*

- *Hashing* - Apply a hash function to the data to determine which partition to place it in. Data-pieces which the same hash will be stored on the same machine. *Hashing* creates balanced partitions, but has no data locality.
- *Range* - Data-pieces are group according to whether their value falls into a given range. This allows for easier range search and sorting, but is not necessarily balanced.
- *Entity-Group* - Explicitly define how to co-locate data. This allows for easy access (equivalent to a single node), but you cannot change the partition after the fact.

When implementing these approaches we want the partitions to be as balanced as possible, for maximal efficiency.

Definition 4.8 - *Replication*
Replication is the practice of replicating data so that it is safe should a machine fail. The difficultly in *Replication* is *Sychronisation.*

Here are the two approaches for when to update data

[17]Proved by Gilbert and Lynch in 2002.

- *Asynchronous* (lazy) - Data is only read on request, this means writing is fast but some replicas may not have been updated (and become stale).
- *Synchronous* (eager) - Data is consistent across users, but read-write is slower as the database need to co-ordinate data between users.

Here are the two approaches for where to store data

- *Primary Copy* - A single node accepts write requests and all other nodes just replicate this node. This is quick and easy to keep consistent, but creates a single point of failure.
- *Update Anywhere* - Any node can accept write requests, and will then propagate these requests to the rest of the network. This approach is fast to write, but requires co-ordination. This approach is more resilient to hardware failure

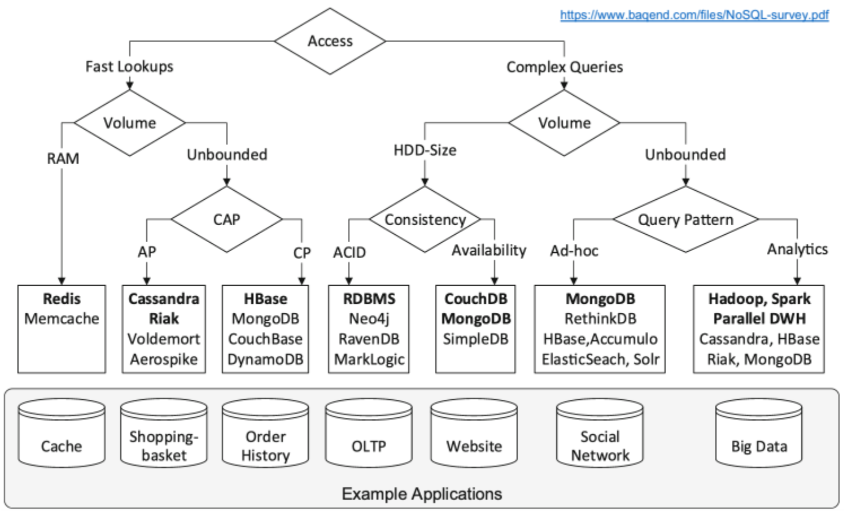

Figure 4: Flow chart to determine which data storage technology to use, given your requirements.

4.2.1 Graph Databases

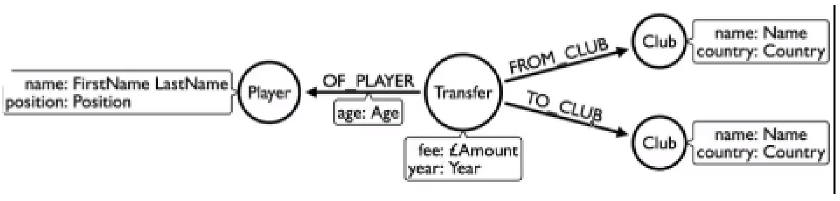

Figure 5: Example of a graph model for a database storing details about football transfers.

Definition 4.9 - *Graph Model*
A graph model can be used to represent data dependencies in a database

- *Nodes* are used to represent datafields or tables. Nodes are labelled with the fields which are unique to them.
- *Edges* are used to show which nodes depend on each other, often with directionality. Edges are labelled with the fields which are shared (ie connect) between two nodes.

See `Figure 4.5` for an example.

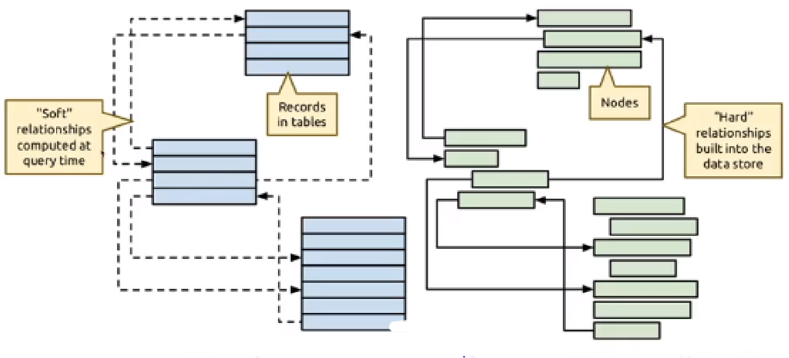
"Soft" relationships computed at query time
Records in tables
Nodes
"Hard" relationships built into the data store

Figure 6: Comparison of the data-storage architectures for Relation Databases (Left) compared to Graph Databases (Right).

Proposition 4.3 - *Relational vs Graph Database*
Here are some comparisons between *Relational* and *Graph Databases*:

- In a *Relational Database* relationships are "soft" and only computed at query time.
- In a *Graph Database* relationships are "hard" as the relationships themselves are stored on the disk.

See `Figure 4.6`.

Proposition 4.4 - *Neo4J*
Neo4J is a piece of *Graph Database* management software.

Here is an example of a *Neo4J* query[18]

```
MATCH (player:Player) WHERE player.name = ''Cristiano Ronaldo''
RETURN player;
```

In this query `player` is a variable we create, `Player` is the type of node that `player` is and `name` is a property of this variable.

We can before logical queries too. The following query returns people who have been a coach and a player.

```
MATCH (n) WHERE n:Player AND n:Coach RETURN n;
```

Proposition 4.5 - *Traversals*
Traversals are a feature which are unique to *Graph Databases*. *Traversals* define what relationship between several nodes to consider, and we can make queries on all these nodes[19].

The query below fetches all transfers away from Man Utd[20]

[18]The equivalent SQL query is `SELECT * FROM players AS player WHERE player.name = ''Cristiano Ronaldo'';`

[19]Similar to *Joins* in relational databases.

[20]The equivalent SQL query is not as clean `SELECT from.name FROM transfers AS t JOIN clubs AS clubFrom ON t.from_club_id =clubFrom.id WHERE from.name="Manchester United";`

```
    MATCH (from:Club) <- [:FROM_CLUB]-(transfer:Transfer)
    WHERE from.name="Manchester United"
```

We can extend this query to be all transfer from Man Utd to Real Madrid[21]

`MATCH (from:Club) <- [:FROM_CLUB]-(transfer:Transfer)-[:TO`$_C LUB]->(to:Club)$
`WHERE from.name="Manchester United"` This query nicely shows how `<-[]-` and `-[]->` define the direction of data flow. We can extend this to find any path, including cycles.

Proposition 4.6 - *Length of Relationship*
Neo4j allows for variable length relationships. With `-[:__*n..]->` notation, where meaning a chain of at least `n` relationships.

The following statement defines a relationship where a player leaves a club and then after some number of transfers returns to the club.

`(c:Club)<-[f:FROM`$_C LUB]-(sale:Transfer)-[:NEXT*1..]->(rebuy:Transfer)-[t:TO_C LUB]->(c)$

4.3 Data Wrangling

Definition 4.10 - *Data Wrangling*
Data Wrangling is the process of getting to know your data and getting it into a format which you are comfortable with. This should get you interested in the project.

A big part of *Data Wrangling* is detecting inaccurate or malformed records and then correcting or removing them. (Data cleaning, validation, debugging, transforming, formatting, pre-processing).

Data Wrangling is an iterative process, you will often return to after "evaluating your model" in order to tweak the data some more.

Remark 4.3 - *Data Used*
Some *Data Wrangling* tasks need to be performed on the whole data set (e.g. checking & validation), but others can be performed with a random, representative subset (e.g. exploring)[22].

Remark 4.4 - *Strategies for Data Wrangling*
Here are some things to consider when *Data Wrangling*

- *Type Screening* - Ensuring data is the type you expect and want.
- *Range Check* - Check data is within legal ranges or categories.
- *Illegal Values* - Check data takes legal values.
- *Multi-Column Validation* - Data in different columns is consistent.
- *Check Unique/Distinct Values* - Check that unique values are reasonable. (Not typos).
- *Remove Duplicates* - Remove duplicate records.

[21]The SQL for this would require two joins
[22]You may want to seek out some outliers, especially when your project is interest in outlier cases.

Remark 4.5 - *Tools*
There are several useful tools for *Data Wrangling*, including

- Data Wrangler `http://vis.stanford.edu/wrangler/`
- Pandas.
- Regular Expressions.

Remark 4.6 - *NoSQL Databases*
NoSQL Databases (e.g. MongoDB, Neo4j) are not as strongly typed as SQL databases.

Remark 4.7 - *Missing Data*
Missing data is common and can occur for many reasons

- Data may be sensitive and people choose to withhold.
- Data may be deemed irrelevant and skipped.
- Data may be censored.
- Communication link may fail (battery dies).
- Data collection may be aborted (boredom).
- Data may be corrupted.
- Data may not be known by the respondent.

Remark 4.8 - *Informative Missing Data*
Sometimes the fact that data is missing can be very informative. It can indicate that the technique/equipment used to take a reading was changed, potentially due to a change in circumstances (A patient went into/out of intensive case).

Imputing values in this case would loose this information.

Remark 4.9 - *Dealing with Missing Data*
Here are some approaches which can be taken to deal with missing data

- Cross reference against validated external information.
- Impute values from nearest neighbours.
- Impute value from mean/median/mode.
- Impute from a model-based calculation.
- Perform multiple imputations.
- Use a “Missing Data” indicator.

5 Transformation & Integration

6 Exploration & Visualisation

7 Deployment

8 Sharing & Privacy