

Business 4720 - Class 2

Data Types and Data Quality

Joerg Evermann

Faculty of Business Administration
Memorial University of Newfoundland
jevermann@mun.ca



Unless otherwise indicated, the copyright in this material is owned by Joerg Evermann. This material is licensed to you under the [Creative Commons by-attribution non-commercial license \(CC BY-NC 4.0\)](#)

This Class

What You Will Learn:

- ▶ Different data types
- ▶ Data quality and provenance
- ▶ Data cleaning

Primitive Data Types

char	Individual Characters
string	A string of characters
byte	1 byte, $-128 \dots 127$ or one Ascii characters
int (16 bit)	"Short", Integer numbers, $-32,768 \dots 32,767$
int (32 bit)	"Long", Integer numbers, $-2,147,483,648 \dots 2,147,483,647$
int (64 bit)	Integer numbers, $-9,223,372,036,854,775,808 \dots$ $9,223,372,036,854,775$
float	Decimal numbers, 6 to 7 significant digits
double	Decimal numbers, 15 to 16 significant digits
boolean	Logical, true/false, 1 or 0

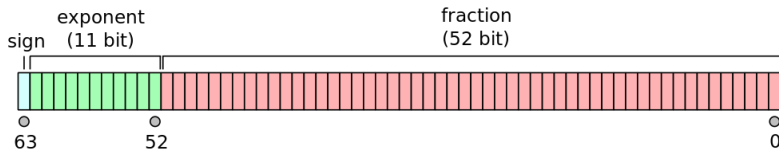
Not all tools use the same names, and not all tools make the same distinctions. For example, the R system uses `numeric` (which is actually a double type) and `integer` (which is a 32 bit integer).

Missing Values

- ▶ Depends on software tool
- ▶ Different meanings (such as not applicable, not available)

R	NA
Python	None
SQL	Null

Floating Point Numbers (IEEE 754 Standard)



https://commons.wikimedia.org/wiki/File:IEEE_754_Double_Floating_Point_Format.svg

$$(-1)^{\text{sign}}(1.b_{51}b_{50}\dots b_0)_2 \times 2^{\text{exp}-1023}$$

$$(-1)^{\text{sign}} \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i} \right) \times 2^{\text{exp}-1023}$$

float	4 bytes, 1 bit for sign, 8 bits for exponent, 23 bits for significand, $\pm 3.40282347e + 38$
double	8 bytes, 1 bit for sign, 11 bits for exponent, 52 bits for significand, $\pm 1.79769313486231570e + 308$

















Floating Point Serialization to Text

Idiosyncrasies

- ▶ Thousands separator, grouping
- ▶ Negatives in brackets
- ▶ "Scientific notation"

-1023476.56	
-1023476,56	some locales use comma as decimal sep
-1,023,476.56	some locales use comma for grouping
-1.023.475,56	some locales use comma as sep and points to group
(1,023,476.56)	some applications use brackets for neg
-1 023 476.56	some locales use space for grouping
-1.02347656e+06	"scientific notation"
-1023.47656e+03	also "scientific notation"
...	

WEIRD UNICODE MATH SYMBOLS AND THEIR MEANINGS

U+29CD		SHARK
U+23E7		TRAFFIC CIRCLE
U+2A33		WAG WAG WAG
U+2A7C		CONFUSED ALLIGATOR
U+299E		SNACK
U+2A04		DRINK REFILL
U+2B48		SNAKES OVER THERE
U+225D		DEFINITELY, FOR SURE
U+237C		LARRY POTTER
U+2A50		SPIDER CAUGHT WITH A CUP AND INDEX CARD
U+2A69		HASH TAG
U+2368		:/
U+2118		SNAKE
U+2AC1		USER PROFILE
U+232D		ROLLING DOUGH BETWEEN YOUR HANDS TO SHAPE IT INTO A BALL
U+2A13		INTEGRAL THAT AVOIDS A BEE ON THE WHITEBOARD

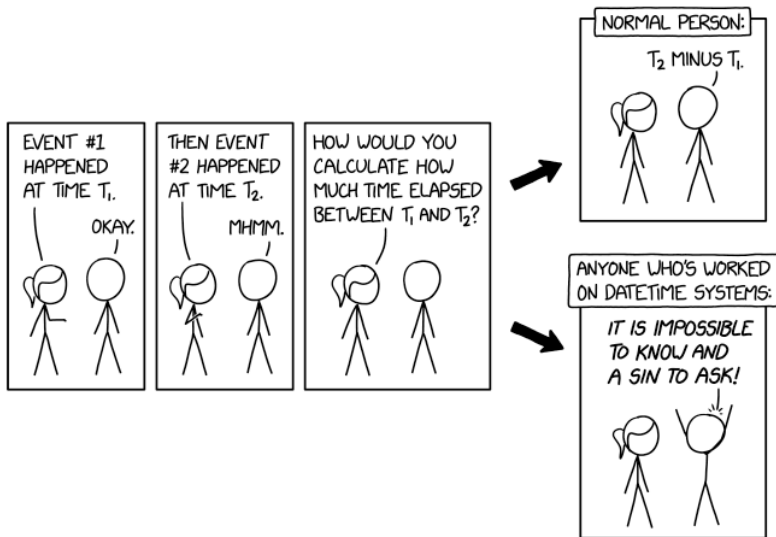
Hands-On Exercise

- ▶ Choose your favourite emoji
- ▶ Determine its Unicode number ("codepage")
- ▶ Determine its UTF-8 encoding

Complexities

- ▶ Calendar formats (written)
 - ▶ DMY, YMD, MDY, YDM with different separators (".", "-", "/")
 - ▶ Not all software or data sets comply with standards
 - ▶ Difficult to parse and validate
- ▶ 12 hour (AM/PM) and 24 hour time formats
- ▶ Time zones
- ▶ Leap seconds, leap years
- ▶ Week numbering
- ▶ Precision (milliseconds, nanoseconds)
- ▶ Different written formats
- ▶ Arithmetic involving years, months, days

Dates and Times



Source: <https://xkcd.com> (CC license)

ISO 8601 and RFC 3339 Date Format

Calendar dates	YYYY-MM-DD
Ordinal dates	YYYY-DDD (not in RFC 3339)
Week dates	YYYY-Www-d (not in RFC 3339)
Times	Thh:mm:ss.sss (or Thhmmss.ss) Thh:mm:ss (or Thhmmss) Thh:mm.mmm or Thhmm.mmm Thh:mm or Thhmm Thh.hhh
Time Zones	<time>Z or <time>±hh:mm or (<time>±hhmm or <time>±hh)
Combined Periods	<date>T<time> PnYnMnDTnHnMnS or P<date>T<time>

Leap Year Rule

$(\text{year} \% 4 == 0) \text{ and } (\text{year} \% 100 != 0 \text{ or } \text{year} \% 400 == 0)$

The territory of Nunavut was created on April 1st, 1999.

- ▶ Express the date in RFC 3339
- ▶ Calculate the number of days since the creation of Nunavut
- ▶ Assume that a ceremony took place at 3PM that day in Iqaluit and express this date-time in RFC 3339
- ▶ Assume the ceremony lasted for 125 minutes and express this duration in RFC 3339

Complex/Structured Data Types

Python

- ▶ `list`, `[1, 2, "a", "b", 2]`, mutable, ordered
- ▶ `tuple`, `(1, 2, "a", "b", 2)`, immutable
- ▶ `set`, `{1, 2, "a", "b"}`, mutable, unordered
- ▶ `dict`, `{"make": "Ford", "year": 2023}`, mutable

R

- ▶ `list`, `list(1, 2, "a", "b", 2)`, mutable, ordered
- ▶ `vector`, `c(1, 2, 3)`, mutable, same primitive type
- ▶ `factor`, `as.factor(c("Hot", "Med", "Cold"))`
- ▶ `matrix`, `matrix(c(1, 2, 3, 4), nrow=2)`
- ▶ `array`, `array(c(1, 2, 3), c(4, 5, 6))`

Data Types in Analytics

Structured Data

- ▶ Tables
- ▶ Key-Value pairs
- ▶ Documents (JSON, XML)
- ▶ Graphs

Unstructured Data

- ▶ Text
- ▶ Image
- ▶ Video

Tables

- ▶ Columns, represent different variables
 - ▶ Each column is a vector, typically named
- ▶ Rows, represents values for different observations
- ▶ Cells, may be primitive or complex, e.g. sets or lists or tables

Name	Area	Population
Canada	9,984,670	38,781,292
Nigeria	923,768	223,804,632
Germany	357,600	83,294,633

CSV format (RFC 4180)

- ▶ Plain text, Ascii or UTF-8 Unicode
- ▶ One record per line
- ▶ Fields separated by delimiter (typically: ",")
- ▶ Fields must be primitive
- ▶ Optional header with column/field names
- ▶ Fields *may* be enclosed by double quotes (" " ")

```
"Name", "Area", "Population" <LF>
"Canada", "9984670", "38781292" <LF>
"Nigeria", "923768", "223804632" <LF>
"Germany", "357600", "83294633" <LF>
```

Potential Problems with CSV Files

- ▶ Different delimiters (";", "tab", ":", " ^ ", etc.)
- ▶ Different line endings/terminators (CR/LF for Windows, LF for Mac & Unix)
- ▶ Different or inconsistent quoting
- ▶ Different decimal and thousand delimiters for numerics (depending on locale)
- ▶ Different date formats (depending on locale)

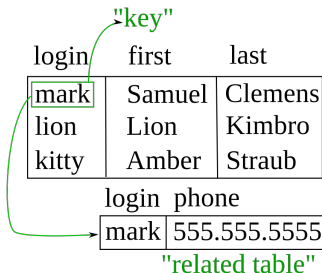
Hands-On Exercise

- ▶ Search the internet for a CSV file of the population and areas of all countries of the world
- ▶ Examine the CSV file and answer the following questions:
 - ▶ What is the delimiter?
 - ▶ Which fields are quoted, and how?
 - ▶ What is the line ending character(s)?
 - ▶ What is the number format?
 - ▶ What is the date format (if there are dates)?
- ▶ Import the CSV file into your favourite spreadsheet tool
 - ▶ Does it recognize all information correctly? If not, what is not imported well?
- ▶ Export the CSV file from your tool under a different name.
 - ▶ Do you get an identical file to the one you imported? If not, what has changed?

Relational Databases

Characteristics

- ▶ Records ("rows") in tables ("relations")
- ▶ Columns/fields are typed
- ▶ Records are identified by "primary keys"
- ▶ Records can refer to other records, in the same or different relation/table



Advantages

- ▶ Normalization reduces redundancy, increases data integrity
- ▶ Enforcement of constraints such as types, referential integrity, non-nulls, etc. increases data integrity
- ▶ Intuitive schemas and queries

Prominent Examples

- ▶ *On-premises*: Oracle DBMS
- ▶ *Open source*: PostgreSQL
- ▶ *Cloud*: Amazon RDS, Google Cloud Database, Azure SQL

Key-Value Data Stores

Characteristics

- ▶ Records are sets of key-value pairs
- ▶ Key has multiple components (ordered list, "minor keys")
- ▶ Value is uninterpreted

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

<https://commons.wikimedia.org/wiki/File:KeyValue.PNG>

Advantages

- ▶ Fast retrieval/insertion/updating
- ▶ No relationships between entities/records
- ▶ Less memory use (does not store empty table cells)
- ▶ Untyped and no fixed schema increases flexibility
- ▶ Easy scalability and distribution

Prominent Examples

- ▶ *On-premises, open-source*: Apache Cassandra, Facebook RocksDB, Redis
- ▶ *Cloud*: AWS DynamoDB, Google LevelDB, Azure CosmosDB

JavaScript Object Notation (RFC 8259)

- ▶ Plain text, UTF-8
- ▶ Primitive Types:
 - ▶ Strings (in single or double quotations)
 - ▶ Number
 - ▶ Boolean
 - ▶ Null
- ▶ Structured Types:
 - ▶ Objects
 - ▶ unordered collection of zero or more name/value pairs
 - ▶ Delimited by "{" and "}"
 - ▶ Arrays
 - ▶ Ordered sequence of zero or more values
 - ▶ Delimited by "[" and "]"

JSON Example – Complex Object

```
{
  "Image": {
    "Width": 1060,
    "Height": 400,
    "Title": "Skyline of Iqualuit, Nunavut",
    "Url":
      "https://upload.wikimedia.org/wikipedia/commons/b/b4/Iqaluit_skylin
    "Legal": {
      "Copyrighted": true,
      "License": "GNU Free Documentation License",
      "Inception": "2010-03-24",
      "Author": "Aaron Lloyd"
    },
  },
}
```

JSON Example – List of Objects

```
[
  {
    "Latitude": 56.536389,
    "Longitude": -61.718889,
    "City": "Nain",
    "Province": "NL",
    "Postal": "A0P",
    "Country": "Canada"
  },
  {
    "Latitude": 53.512778,
    "Longitude": -60.135556,
    "City": "Sheshatshiu",
    "Province": "NL",
    "Postal": "A0P",
    "Country": "Canada"
  }
]
```

Document yourself in a JSON object

- ▶ Identify information about yourself, such as names, addresses, dates, relationships (work, school, uni), etc.
- ▶ Structure the information in JSON Objects and Arrays
- ▶ Use nested structures, e.g. objects in arrays, or arrays in objects, or objects in objects, etc.

What makes them special

- ▶ *Nested* key-value data store
- ▶ All keys are *strings*
- ▶ No fixed schema, increases flexibility

Applications

- ▶ Content management
- ▶ Catalogs and product data
- ▶ Log and event data (IoT, sensors)

Prominent Examples

- ▶ *On-premises*: MongoDB, ArangoDB
- ▶ *Open source*: Apache CouchDB
- ▶ *Cloud*: AWS DocumentDB, Azure CosmosDB

Extensible Markup Language – XML

- ▶ Document serialization format for structured data
- ▶ Nested **elements**
- ▶ Elements described by opening and closing **tag**
- ▶ Elements may have **attributes**, defined in opening tag
- ▶ Attributes can only hold simple data, elements can contain structured data.
- ▶ Arbitrary element and attribute names uniquely defined within **namespaces** (typically URI, Uniform Resource Identifier)
- ▶ Element and attribute are defined using **XML Schema**

XML Example

XML

```
<People
  xmlns="https://www.example.com/peoples"
  xmlns:geo="http://www.example.com/geo"
  xmlns:hist="http://www.example.com/history">
  <GeneralInformation
    Name="Innu" Language="Innu-aimun">
    <geo:Location geo:Country="Canada"
      geo:Regions="Labrador, Quebec" />
  </GeneralInformation>
  <hist:History>
    <hist:Period hist:era="Pre-Colonial">
      <Description>
        Nomadic lifestyle, primarily
        hunting and fishing.
      </Description>
    </hist:Period>
```

XML

```
<hist:Period hist:era="Post-Colonial">
  <Description>
    Impact of colonization,
    including displacement and
    cultural changes.
  </Description>
</hist:Period>
</hist:History>
<Culture>
  <Traditions>
    <Tradition>
      Hunting and fishing as cultural
      and subsistence activities.
    </Tradition>
    <Tradition>
      Use of the tepee for temporary
      shelter.
    </Tradition>
  </Traditions>
```


XML

```
<Art>
  <Form>Drum making</Form>
  <Form>Clothing with intricate beadwork
</Form>
</Art>
</Culture>
<Challenges>
  Issues like land rights, cultural preservation
</Challenges>
</People>
```

XML Example – Key Points

- ▶ The `xmlns:geo` and `xmlns:hist` are namespace declarations. They are used to distinguish between geographical (geo) and historical (hist) data. Notice how element and attribute names may be prefixed by a namespace.
- ▶ The `xmlns` is the default namespace and applies to all elements and attributes without an explicit namespace.
- ▶ The `GeneralInformation` element has attributes for Name and Language. Attributes must be character strings
- ▶ The `Location` element includes attributes for Country and Regions, using the `geo` namespace.
- ▶ The `Location` does not include other elements and is defined with just one tag.
- ▶ Each `Period` element in the `History` element has an attribute `hist:era` to specify the era.
- ▶ Multiple elements with the same name can follow each other.

Equivalent JSON Example

```
{ "Innu": {  
  "GeneralInformation": {  
    "Location": {  
      "Country": "Canada",  
      "Regions": "Labrador, Quebec"  
    }  
  },  
  "History": {  
    "Period": [  
      {  
        "era": "Pre-Colonial",  
        "Description": "Nomadic lifestyle,  
          primarily hunting and fishing."  
      },  
      {  
        "era": "Post-Colonial",  
        "Description": "Impact of colonization, ..."  
      }  
    ]  
  },  
}
```

- ▶ Both are human and machine readable
- ▶ Both are system/language independent
- ▶ XML is more verbose/lengthy and very self-descriptive
- ▶ JSON is more compact/lightweight but not as descriptive
- ▶ XML supports more complex structures than JSON through
 - ▶ Attributes
 - ▶ Namespaces
 - ▶ More datatypes
- ▶ XML can be strictly defined using XML Schema, JSON always remains flexible

Document yourself in an XML document

- ▶ Identify information about yourself, such as names, addresses, dates, relationships (work, school, uni), etc.
- ▶ Structure the information in Elements and Attributes
- ▶ Use nested elements where appropriate

Characteristics

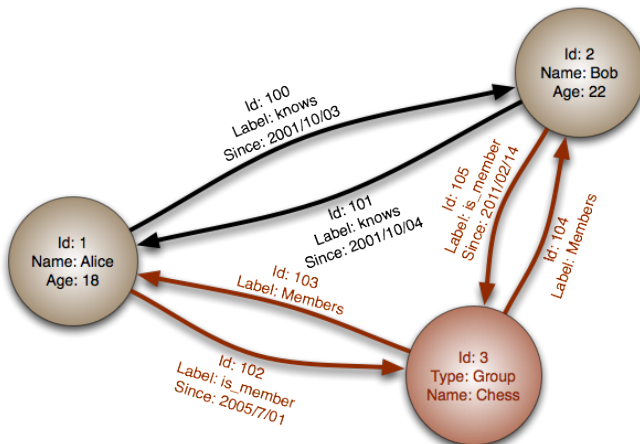
- ▶ Nodes (also called "vertices")
- ▶ Edges (also called "arcs", "relationships") that connect vertices
 - ▶ Directed or undirected
- ▶ Vertices and Edges may be typed

Applications

- ▶ Social networks
- ▶ Logistics networks
- ▶ Financial networks
- ▶ Biological networks
- ▶ Resource descriptions (RDF)

Property Graphs

- Vertices and Edges may have properties (e.g. key/value pairs of simple or complex data types, e.g. JSON objects)



RDF Graphs

- ▶ Resource Description Framework
- ▶ Subject – Verb/Predicate – Object triples



https://commons.wikimedia.org/wiki/File:Rdf_graph_for_Eric_Miller.png

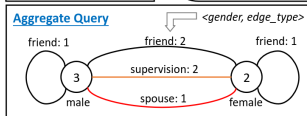
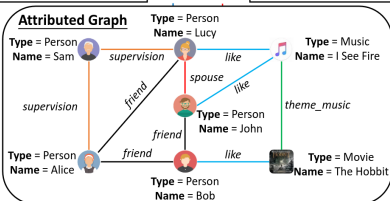
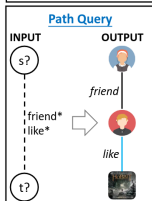
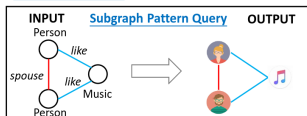
Graph Queries

- ▶ Path queries: Reachability, shortest-path, regular path
- ▶ Subgraph queries: Exact match, approximate match
- ▶ Aggregate queries
- ▶ Similarity search: path-based approaches, graph embedding-based approaches
- ▶ Keyword search: tree-based semantics, subgraph-based semantics

Wang, Y., Li, Y., Fan, J., Ye, C., & Chai, M. (2021). A survey of typical attributed graph queries. *World Wide Web*, 24, 297-346.

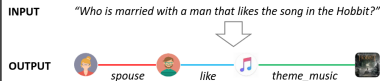
Graph Queries

Structured Query

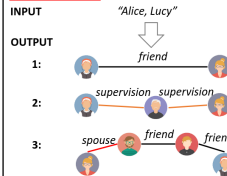


Unstructured Query

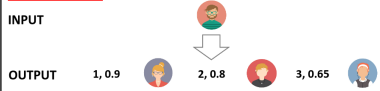
Natural Language Question Answering



Keyword Search



Similarity Search



Wang, Y., Li, Y., Fan, J., Ye, C., & Chai, M. (2021). A survey of typical attributed graph queries. *World Wide Web*, 24, 297-346.

Prominent Examples

- ▶ *On-premises*: ArangoDB, Neo4J, OrientDB
- ▶ *Open-source*: JanusGraph
- ▶ *Cloud*: AWS Neptune, Azure CosmosDB

PG-JSON Example

```
{
  "nodes": [
    {
      "id": 101,
      "labels": ["Person"],
      "properties": {"name": ["Alice"], "age": [15], "country": ["United
    ],
    {
      "id": 102,
      "labels": ["Person", "Student"],
      "properties": {"name": ["Bob"], "country": ["Japan", "Germany"]}
    }
  ],
  "edges": [
    {
      "from": 101,
      "to": 102,
      "undirected": true,
      "labels": ["sameSchool", "sameClass"],
      "properties": {"since": [2012]}
    },
    {
      "from": 102,
      "to": 101
    }
  ]
}
```

GraphSON Example

```
{
  "graph": {
    "mode": "NORMAL",
    "vertices": [
      {
        "name": "lop",
        "lang": "java",
        "_id": "3",
        "_type": "vertex"
      },
      {
        "name": "vadas",
        "age": 27,
        "_id": "2",
        "_type": "vertex"
      },
      {
        "name": "marko",
        "age": 29,
        "_id": "1",
        "_type": "vertex"
      },
      {
        "name": "peter"
```

GraphSON Example [cont'd]

```
"edges": [  
  {  
    "weight": 1,  
    "_id": "10",  
    "_type": "edge",  
    "_outV": "4",  
    "_inV": "5",  
    "_label": "created"  
  },  
  {  
    "weight": 0.5,  
    "_id": "7",  
    "_type": "edge",  
    "_outV": "1",  
    "_inV": "2",  
    "_label": "knows"  
  },  
  {  
    "weight": 0.4000000059604645,  
    "_id": "9",  
    "_type": "edge",  
    "_outV": "1",  
    "_inV": "3",  
    "_label": "created"
```

```
<http://www.w3.org/People/EM/contact#me>  
<http://www.w3.org/2000/10/swap/pim/contact#fullName>  
  "Eric Miller" .  
  
<http://www.w3.org/People/EM/contact#me>  
<http://www.w3.org/2000/10/swap/pim/contact#mailbox>  
  <mailto:e.miller123(at)example> .  
  
<http://www.w3.org/People/EM/contact#me>  
<http://www.w3.org/2000/10/swap/pim/contact#personalTitle>  
  "Dr." .  
  
<http://www.w3.org/People/EM/contact#me>  
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>  
  <http://www.w3.org/2000/10/swap/pim/contact#Person> .
```

Turtle Example (Terse RDF Triples)

```
@prefix eric:    <http://www.w3.org/People/EM/contact#> .
@prefix contact: <http://www.w3.org/2000/10/swap/pim/contact#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

eric:me contact:fullName "Eric Miller" .
eric:me contact:mailbox <mailto:e.miller123(at)example> .
eric:me contact:personalTitle "Dr." .
eric:me rdf:type contact:Person .
```


Document yourself in a Turtle

- ▶ Identify information about yourself, such as names, addresses, dates, relationships (work, school, uni), etc.
- ▶ Structure the information in Turtle triples
- ▶ Make up appropriate prefixes and appropriate verbs/predicates

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#"
xmlns:eric="http://www.w3.org/People/EM/contact#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#m"
    <contact:fullName>Eric Miller</contact:fullName>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#m"
    <contact:mailbox rdf:resource="mailto:e.miller123(at)example"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#m"
    <contact:personalTitle>Dr.</contact:personalTitle>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#m"
    <rdf:type rdf:resource="http://www.w3.org/2000/10/swap/pim/cont"
  </rdf:Description>
</rdf:RDF>
```

Example Text Analysis Tasks

- ▶ Named entity recognition
- ▶ Document clustering/similarity detection
- ▶ Co-reference analysis
- ▶ Relationship, fact, event extraction
- ▶ Sentiment analysis

Approaches

- ▶ Symbolic (1950s – 1990s)
- ▶ Statistical (1990s – 2010s)
- ▶ Neural networks (present)

Business Applications

- ▶ Marketing
- ▶ Customer relationship management
- ▶ Finance
- ▶ ...

Hands-On Exercise

- 1 Identify a specific business problem that can be addressed by analyzing text data
- 2 What text data would you need to address the problem?
- 3 What would you wish to do with the text data?
- 4 Where might you get this text data?

Regular Expressions (Regex)

- ▶ Important tool in analyzing text
- ▶ Sequence of characters specifying a match pattern in text
- ▶ Example (matches any number):

```
[+-]?(\d+(\.\d*)?|\.\d+)([eE][+-]?\d+)?.
```

https://en.wikipedia.org/wiki/Regular_expression

Basic RegEx

Metacharacter	Description
<code>^</code>	Matches start of text
<code>.</code>	Matches any character; matches the dot character within brackets
<code>[]</code>	Matches any of the characters in the brackets; - can be used to specify ranges
<code>[^]</code>	Matches any character not in the brackets
<code>\$</code>	Matches the end of text
<code>()</code>	Marked subexpression
<code>\n</code>	Matches the n-th marked subexpression
<code>*</code>	Matches the preceding element zero or more times
<code>{m,n}</code>	Matches the preceding element at least m and not more than n times

All others are treated as literals

Basic RegEx Examples

RegEx	Matches
.at	"hat", "cat", "bat", "4at", etc.
[hc]at	"hat", "cat"
[^b]	all strings matched by .at except "bat"
[^bc]	all strings matched by .at except "bat" and "cat"
^[bc]at	"bat" and "cat" at start of text
[bc]at\$	"bat" and "cat" at end of text
\[.]	any single character surrounded by [and], e.g. "[a]", "[7]", etc.
s.*	character "s" followed by zero or more characters, e.g. "s", "saw", "s3w96.7", etc.

Metacharacter	Description
?	Matches preceding element zero or one time
+	Matches preceding element one or more times
	Matches either the expression before or after the choice operator

Extended RegEx Examples

RegEx	Matches
[hc]?at	"at", "hat", "cat"
[hc]*at	"at", "hat", "cat", "chat", "chchchat", etc.
[hc]+at	"hat", "cat", "chat", "chchchat", etc.
cat dog	"cat" or "dog"

Different Types of RegEx

Differences

- ▶ Posix BRE requires () { } to be escaped, ERE does not
- ▶ Perl RegEx is a popular "dialect"
- ▶ Some RegEx dialects provide characterclasses

	Perl/Vim	ASCII	Posix
Digits	\d	[0-9]	[:digit:]
Non-digits	\D	[^0-9]	
Lowercase letters	\l	[a-z]	[:lower:]
Uppercase letters	\u	[A-Z]	[:upper:]
Alphanumeric chars	\w	[A-Za-z0-9_]	
Non-word chars	\W	[^A-Za-z0-9_]	
Whitespace	\s	[\t\r\n\v\f]	[:space:]
Non-whitespace	\S	[^ \t\r\n\v\f]	

1 Specify a RegEx to match Canadian postal codes

`https://www.canadapost-postescanada.ca/cpc/en/support/articles/addressing-guidelines/postal-codes.page`

2 Specify a RegEx to match a RFC 3339 date with timezone, such as "2023-11-14T20:42:53-04:30"

Levenshtein Distance

- ▶ Text similarity
- ▶ Type of string–edit distance
- ▶ Counts insertion, deletion, substitution operations to transform one text into the other
- ▶ May use differential cost for the operations

$$\text{lev}(a, b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \text{lev}(\text{tail}(a), \text{tail}(b)) & \text{if } \text{head}(a) = \text{head}(b), \\ 1 + \min \begin{cases} \text{lev}(\text{tail}(a), b) \\ \text{lev}(a, \text{tail}(b)) \\ \text{lev}(\text{tail}(a), \text{tail}(b)) \end{cases} & \text{otherwise} \end{cases}$$

https://en.wikipedia.org/wiki/Levenshtein_distance

Hands-On Exercise

Determine the Levenshtein distances between the following:

- 1 Last five digits of your student number and "12345"
- 2 The words "Nunavut" and "Nunatsiavut"
- 3 The words "Inuktitut" and "Innuttitut"
- 4 The words "Mikak" and "Micoock"

Vector Formats

- ▶ Examples are SVG, EPS, PDF
- ▶ Describe images in terms of graphics primitives such as rectangles, curves, polygons
- ▶ Infinitely scalable

Raster formats

- ▶ Describe images in terms of pixels and their colours
- ▶ Lossy compression formats such as PNG, JPEG
- ▶ Lossless compression formats such as TIFF
- ▶ Based on specific colorspace such as RGB (3 bytes) or CMYK (4 bytes) and resolution
- ▶ Conceptually a $3 \times X \times Y$ array of RGB values between $0 \dots 255$ (or $0 \dots 1$)

Image Analysis



https://commons.wikimedia.org/wiki/File:Persian_sand_CAT.jpg

Tasks

- ▶ Object detection and counting
- ▶ Object classification
- ▶ Image segmentation
- ▶ Image retrieval
- ▶ ...

Applications

- ▶ Robotics
- ▶ Character and handwriting recognition (process automation)
- ▶ Security (identity verification, fraud detection, etc.)
- ▶ Manufacturing (defect detection, etc.)
- ▶ ...

Hands-On Exercise

- 1 Identify a specific business problem that can be addressed by analyzing image data
- 2 What image data would you need to address the problem?
- 3 What would you wish to do with the image data?
- 4 Where might you get this image data?

Codecs and Container Formats

- ▶ Conceptually a series of image *frames* in raster image format, i.e. a $T \times 3 \times X \times Y$ array of RGB values between $0 \dots 255$ (or $0 \dots 1$)
- ▶ Codecs ("coder/decoder") such as H.264, H.265, AVC, AV1 describe how frames are encoded in computer bytes and compressed
- ▶ Containers such as MPEG-4, Matroska, AVI, VOB, WebM describe how multiple video, audio, and text (subtitle) streams are arranged in a file

Typical Tasks

- ▶ Object detection
- ▶ Object recognition
- ▶ Object motion detection
- ▶ Object or background dynamic masking/blurring
- ▶ Event detection and classification (errors, exceptions)
- ▶ Activity detection and classification

Hands-On Exercise

- 1 Identify a specific business problem that can be addressed by analyzing video data
- 2 What video data would you need to address the problem?
- 3 What would you wish to do with the video data?
- 4 Where might you get this video data?

- ▶ Data about data
- ▶ Embedded in data or external
- ▶ Examples
 - ▶ Authorship & ownership
 - ▶ Licensing & legal information
 - ▶ Information about collection (when, who, where, how, what)
 - ▶ Information about processing (when, who, where, how, what)
 - ▶ Technical information (e.g. encoding, serialization format, etc.)

Hands-On Exercise

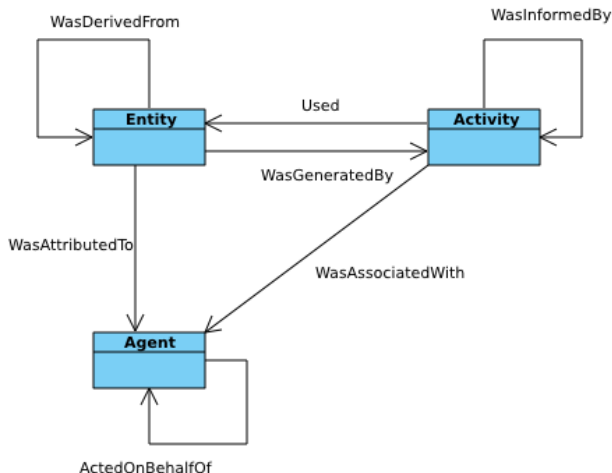
- 1 With your cell phone camera, take a selfie
- 2 Identify the meta-data for this photo

Dimensions

- ▶ Accuracy (error rate)
- ▶ Availability (cost, ease of retrieval or collection, licensing)
- ▶ Completeness (omissions, bias)
- ▶ Conformity (with external standards)
- ▶ Consistency (no contradictions)
- ▶ Integrity (relationships within the data)
- ▶ Precision (measurement precision of values)
- ▶ Relevance (usefulness)
- ▶ Reliability (consistency of repeated data points)
- ▶ Timeliness (latency, currency, "age")
- ▶ Traceability (auditable provenance, verifiable source)

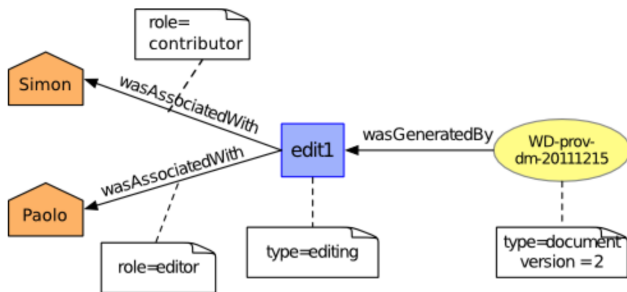
Based in part on Richard Y. Wang & Diane M. Strong (1996) Beyond Accuracy: What Data Quality Means to Data Consumers, Journal of Management Information Systems, 12:4, 5-33,
DOI: 10.1080/07421222.1996.11518099

Data Provenance and Validity



https://commons.wikimedia.org/wiki/File:Prov_dm-essentials.png

Data Provenance and Validity [cont'd]



[urlhttps://www.w3.org/TR/prov-dm/#dfn-provenance](https://www.w3.org/TR/prov-dm/#dfn-provenance)

Collection

- ▶ How was the data collected? What errors could have happened?
- ▶ Who collected the data? Is it a trustworthy source?
- ▶ When were the data collected? Are they still valid?
- ▶ Are all the data collected? Are the data biased?
- ▶ Can the collection be verified/audited/repeated?

Processing

- ▶ How was the data processed? What mistakes could have been made?
- ▶ Was anything omitted or added?
- ▶ Who processed the data? Is it a trustworthy party?
- ▶ Can the processing be verified/audited/repeated?

- ▶ What do different data fields mean?
- ▶ What are the units of measure?
- ▶ What is the level of aggregation?
- ▶ Were data sources combined? Are the different sources consistent with each other and of the same quality?
- ▶ Are the data accurate? How high are the error rates and the levels of precision?
- ▶ Can the data be validated? What are the validation rules for the data? Was the data validated?
- ▶ How can errors be detected and/or corrected?
- ▶ Are the data usable in a technical and legal way?

Hands-On Exercises

- 1 Identify data on the consumer price index (excluding living and transportation expenses) for Newfoundland & Labrador for the last 10 years
 - ▶ How was it collected? By who? When?
 - ▶ How was it processed? By who? What was done to it?
 - ▶ Is there meta-data available for it?
 - ▶ How do you assess the quality of the data on the data quality dimensions?
 - ▶ Under what license is it available to you to use?
- 2 Identify some IoT devices or sensors in your household
 - ▶ What information can they measure?
 - ▶ How and when is the information being collected? By who?
 - ▶ How could the information be erroneous or biased?
 - ▶ How would you assess the quality of the data?

Overview

- ▶ Critical step in the data analysis process
- ▶ Identification and rectification of errors and inconsistencies
- ▶ Improve data quality

Data Cleaning

- ▶ Critical step in the data analysis process
- ▶ Identify and "fix" errors and inconsistencies
- ▶ Improve data quality

Activities

- 1 **Auditing:** Identify anomalies and inconsistencies
- 2 **Validation:** Ensure data conforms to rules and constraints
- 3 **Cleaning:** Transform and correct data
- 4 **Duplicate Removal:** Ensure uniqueness of data.
- 5 **Harmonization:** Merge datasets from different sources and ensure consistent formats and scales.
- 6 **Standardization:** Bring data into a standard format.
- 7 **Quality Assessment:** Ensure cleaning has been effective.

Data Validation

- ▶ Coding/serialization rules, e.g. with Regex
 - ▶ **Example** Are all phone numbers of the format:
`^([0-9]{3})[-]?[0-9]{3}[-]?[0-9]{4}$`
- ▶ Data type constraints
 - ▶ *Example:* Are all sales prices numbers?
- ▶ Range constraints
 - ▶ *Examples:* Are prices > 0 ? Are sales number < 1000 ?
- ▶ Cross-field validation
 - ▶ *Example:* If province is NL, then area code must be 709
- ▶ Uniformity of measures/scales
 - ▶ *Example:* All weights must be in kg, not pounds

- ▶ Uniqueness constraints
 - ▶ Real duplicates and synonyms
 - ▶ *Example:* Rebekah Uqi Williams
(Commissioner of Nunavut (2020–2021))
 - ▶ **Abbreviations:**
Rebekah U. Williams; Rebekah Williams, R.U. Williams
 - ▶ **Order:**
Williams, Rebekah Uqi; Williams, Rebekah U.; Williams, R.
 - ▶ **Spelling:**
Rebekah; Rebecca; Rebeccah; Rebeckah; Rebecka
 - ▶ **Misspellings:**
Reebkah, Rebkah, Wililams, Willaims, . . .
 - ▶ . . .

- ▶ **Data Transformation**, into proper format or structure.
 - ▶ One row for each observation, case, event, . . .
 - ▶ Requires case or event identifiers
- ▶ **Data Imputation**, replacing missing values with estimated or default values, or removing missing values
 - ▶ Different meanings of missing values
 - ▶ Removal may bias data
 - ▶ Estimating values may be error-prone
- ▶ **Data Correction**, or removal of erroneous data.
 - ▶ Requires access to correct data

Important

Cleaning, transformation, and correction of data is *subjective* and requires *expert knowledge* of the data, the validation rules, the metadata, and the application domain.

The 80/20 Rule of Data Science

Cleaning, transformation, and correction of data takes 80% of the time, data analysis takes 20% of the time.