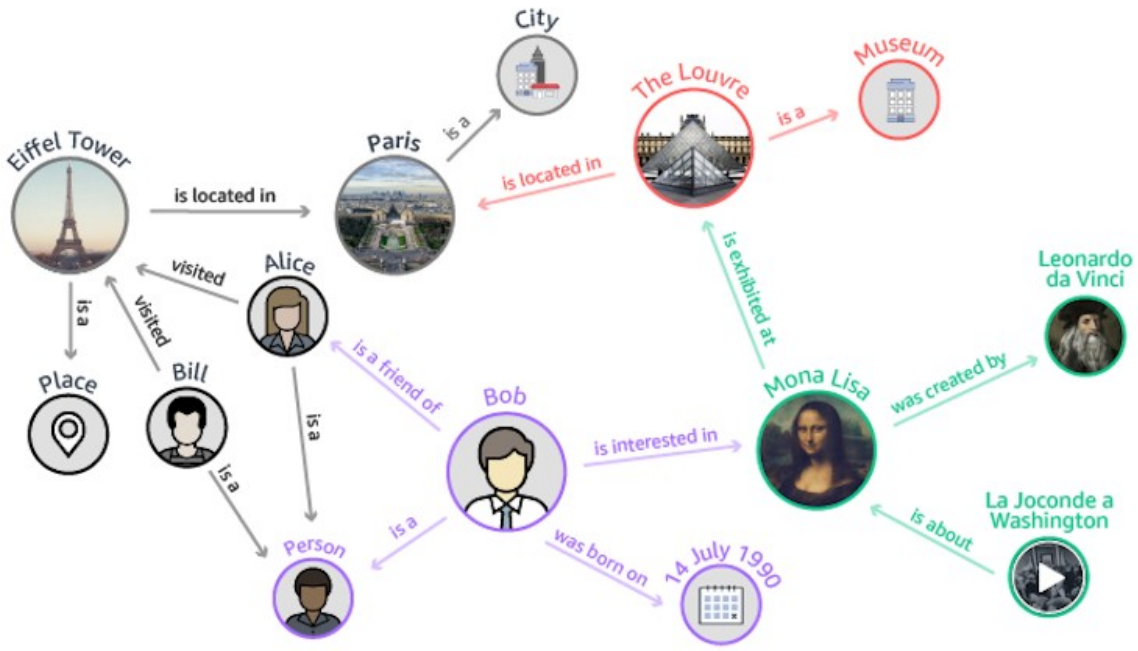
NLP based Knowledge Graph using Spacy, Neo4j

For the past decade or so, Knowledge Graphs have been sneaking into our daily lives, be it through voice assistants (such as Alexa, Siri, or Google Assistant), intuitive search results, or even personalized shopping experiences on Amazon. We constantly interact with Knowledge Graphs on a daily basis.
Connecting data adds context and improves outcomes. The best doctors do their homework. If a doctor takes the time to review the reason for the appointment – your medical conditions, your medical history, any lab tests you've had, and your vital signs – that visit will be far more productive. The doctor has context: the context of you.

A knowledge graph is an interconnected dataset enriched with semantics so we can reason about the underlying data and use it confidently for complex decision-making.



source: https://www.kaggle.com/ferdzso/knowledge-graph-analysis-with-node2vec

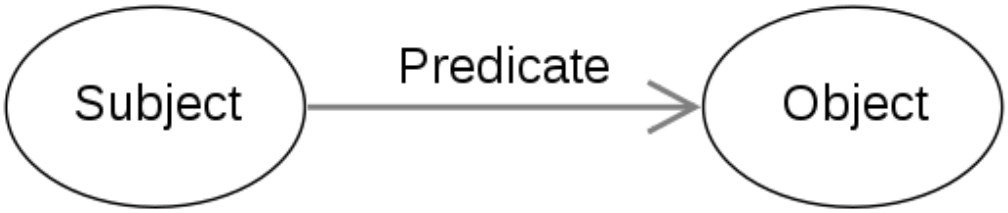
Applications of Knowledge Graph

- Search Systems
- Recommendation systems
- Question Answers

Creating Knowledge Graph

Knowledge graph can be generated in following ways

- Word Co occurrence : The nodes are put closer in the graph based on their co-occurrences.
- RDO Triples: a triple is a set of three entities that codifies a statement about semantic data in the form of subject–predicate–object expressions (e.g., "Bob is 35", or "Bob knows John").
- SVO Triples : Identities Subjects, verbs, Objects triples to create nodes and edges



NLP Considerations for creating knowledge Graph

- Named Entity Recognition: A named entity is a “real-world object” that’s assigned a name – for example, a person, a country, a product or a book title.
- SVO/SPO triples : In a regular SVO (subject-verb-object) sentence, the following dependencies are what you need:
    - *root(ROOT, verb)*
    - *nsubj(verb, **subject**)*
    - *dobj(verb, **object**)*
-

Using Spacy for identifying Named Entities and SVO

spaCy is a free, open-source library for advanced Natural Language Processing (NLP) in Python.spaCy can recognize various types of named entities in a document, by asking the model for a prediction. Named entities are available as the ents property of a Doc:

```
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")

for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

Using Neo4j building knowledge graph

Neo4j is a native graph database, built from the ground up to leverage not only data but also data relationships. Neo4j connects data as it's stored, enabling queries never before imagined, at speeds never thought possible.

Using Neo4j Cypher Query Language

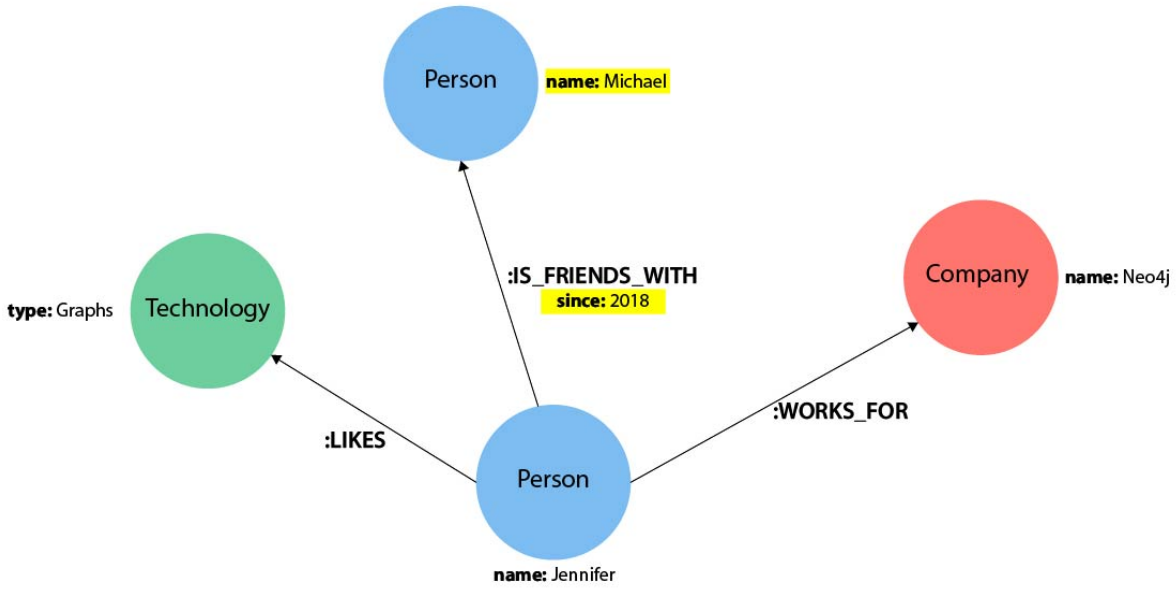
Cypher is Neo4j's graph query language that allows users to store and retrieve data from the graph database. Neo4j wanted to make querying graph data easy to learn, understand, and use for everyone, but also incorporate the power and functionality of other standard data access languages.
Cypher's syntax provides a visual and logical way to match patterns of nodes and relationships in the graph. It is a declarative, SQL-inspired language for describing visual patterns in graphs using ASCII-Art syntax. It allows us to state what we want to select, insert, update, or delete from our graph data without a description of exactly how to do it.

```
//data stored with this direction
```

```
CREATE (p:Person)-[:LIKES]->(t:Technology)

//query relationship backwards will not return results
MATCH (p:Person)<-[:LIKES]-(t:Technology)

//better to query with undirected relationship unless sure of direction
MATCH (p:Person)-[:LIKES]-(t:Technology)
```



Graph Embeddings using Neo4J

The starting point for all machine learning is to turn your data into vectors/embeddings (if they don't already have them). Graph embeddings are useful when the data is in the form of the graph and other data is related to each other.

Node2Vec

Node2Vec is a node embedding algorithm that computes a vector representation of a node based on random walks in the graph. The neighborhood is sampled through random walks.

Using a number of random neighborhood samples, the algorithm trains a single hidden layer neural network. The neural network is trained to predict the likelihood that a node will occur in a walk based on the occurrence of another node.

```
CALL gds.beta.node2vec.stream(
  graphName: String,
  configuration: Map
) YIELD
  nodeId: Integer,
  embedding: List of Float
```

References:

- https://neo4j.com/
- https://spacy.io/