COMP3220: Document Processing and Semantic Technologies Knowledge Graph Construction

Rolf Schwitter

Rolf.Schwitter@mq.edu.au

COMP3220 Topics: Week 7-13

Week 7: Semantic Technologies

Week 8: RDF, RDF Schema and SPARQL

Week 9: DBpedia and Wikidata

Week 10: Ontologies

Week 11: Rule Languages

Week 12: Recent Trends in Semantic Technologies

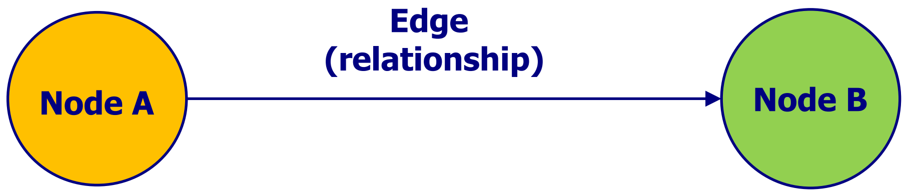
Week 13: Revision

Today’s Agenda

- What is a Knowledge Graph?
- Knowledge Graph Construction
- Publicly available Knowledge Graphs
- Use Cases
- What is spaCy?
- Constructing a Knowledge Graph in Python

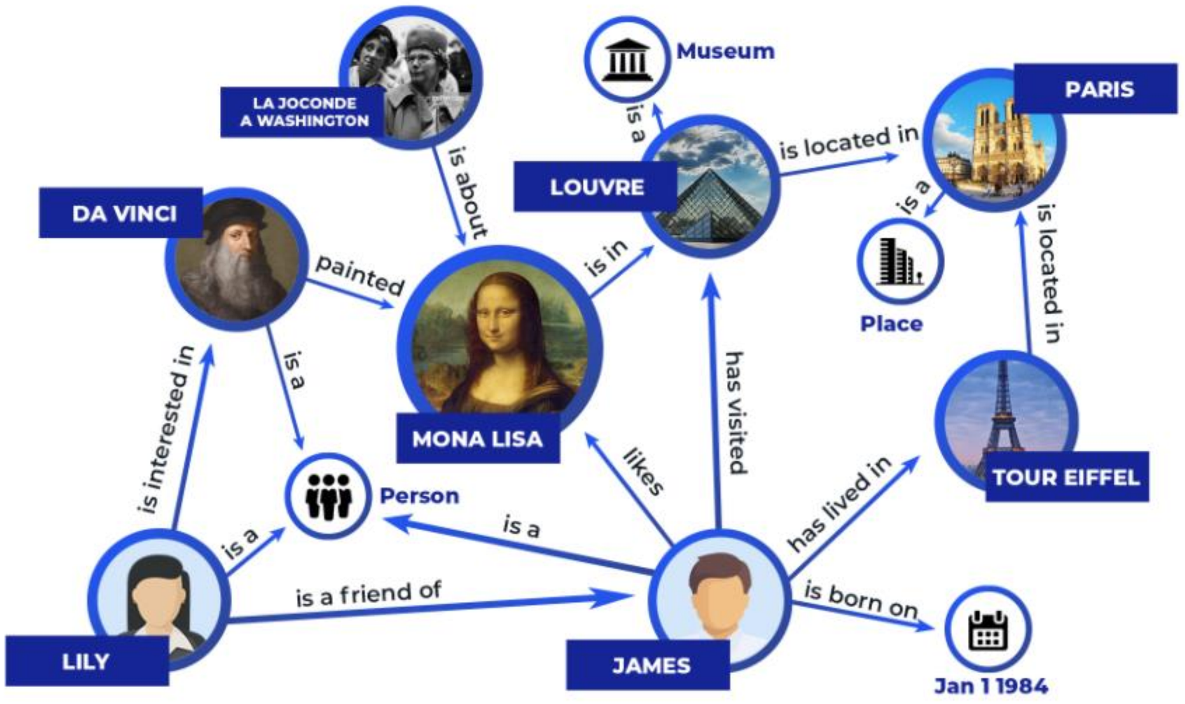
What is a Knowledge Graph?

- A knowledge graph is a model of a knowledge domain.
- A knowledge graph consists of connected entities which can be people, locations, organizations, or events.
- We can define a knowledge graph as a directed labelled graph in which labels have well-defined meanings.
- A directed labelled graph consists of nodes, edges and labels.



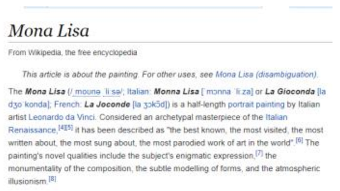
- The smallest knowledge graph consists of two nodes that are connected by an edge; also known as a triple.

A Knowledge Graph



Knowledge Graph Construction

- Massive amount of unstructured data on the web.
- We need an efficient way to represent this data.
- A knowledge graph is a large network of linked data.
- Knowledge graphs are constructed from knowledge bases.
- Knowledge bases gather their information from free text on web pages and other content (e.g. from databases).
- Knowledge graph construction pipeline:



Mona Lisa

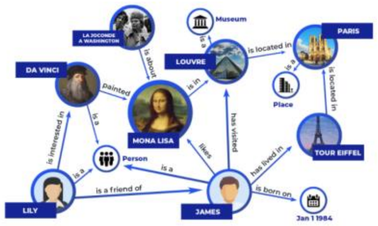
From Wikipedia, the free encyclopedia

This article is about the painting. For other uses, see Mona Lisa (disambiguation).

The ***Mona Lisa*** (/ˌmoʊnə ˈliːsə/; Italian: ***Monna Lisa*** [ˈmɔnna ˈliːza] or ***La Gioconda*** [la dʒoˈkonda]; French: ***La Joconde*** [la ʒɔkɔ̃d]) is a half-length portrait painting by Italian artist Leonardo da Vinci. Considered an archetypal masterpiece of the Italian Renaissance,[4][5] it has been described as "the best known, the most visited, the most written about, the most sung about, the most parodied work of art in the world".[6] The painting's novel qualities include the subject's enigmatic expression,[7] the monumentality of the composition, the subtle modelling of forms, and the atmospheric illusionism.[8]

Free Text

Knowledge Base



Knowledge Graph

Publicly Available Knowledge Graphs

- DBpedia and Wikidata are publicly available knowledge graphs:
  - https://www.dbpedia.org/
  - https://www.wikidata.org/wiki/Wikidata:Main_Page
- These knowledge graphs can be queried using standard query languages such as SPARQL.
- For example, Wikidata contains background information about most topics of the world.
- Wikidata currently contains 97,225,327 items (26.03.2022).
- Here is a link to a Wikidata page with information about Nicole Kidman: https://www.wikidata.org/wiki/Q37459

Use Cases

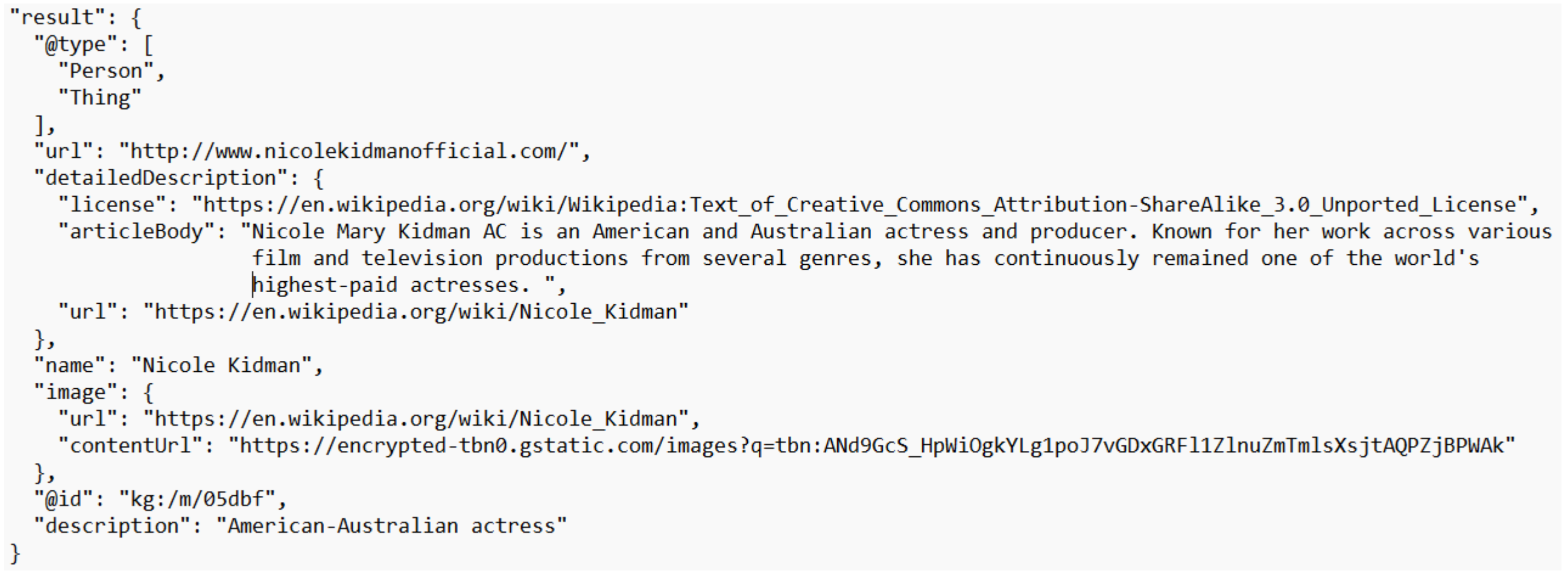
- Question answering is one of the most used applications of knowledge graphs.
- Storing information of research; companies use knowledge graphs to store information for building models (risk management, process monitoring, etc.).
- Netflix uses a knowledge graph to store vast amount of information for its recommender system.
- Google uses a knowledge graph to enhance its search engine's results with information gathered from a variety of sources. The information is presented to users in a knowledge panel next to the search results.

Google Knowledge Graph Search API

- Sample request:

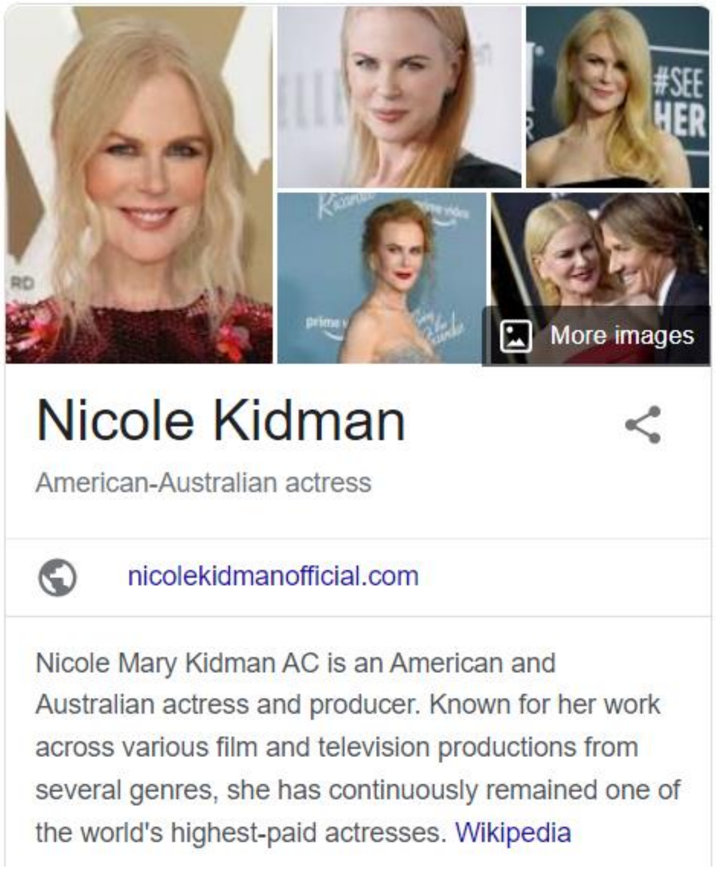
```
https://kgsearch.googleapis.com/v1/entities:search? \
query=nicole+kidman&key=API_KEY&limit=1&indent=True
```

- Partial JSON-LD output:



```
"result": {
  "@type": [
    "Person",
    "Thing"
  ],
  "url": "http://www.nicolekidmanofficial.com/",
  "detailedDescription": {
    "license": "https://en.wikipedia.org/wiki/Wikipedia:Text_of_Creative_Commons_Attribution-ShareAlike_3.0_Unported_License",
    "articleBody": "Nicole Mary Kidman AC is an American and Australian actress and producer. Known for her work across various
                    film and television productions from several genres, she has continuously remained one of the world's
                    highest-paid actresses. ",
    "url": "https://en.wikipedia.org/wiki/Nicole_Kidman"
  },
  "name": "Nicole Kidman",
  "image": {
    "url": "https://en.wikipedia.org/wiki/Nicole_Kidman",
    "contentUrl": "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcS_HpWiOgkYLg1poJ7vGDxGRFl1ZlnuZmTmlsXsjtAQPZjBPWAk"
  },
  "@id": "kg:/m/05dbf",
  "description": "American-Australian actress"
}
```

Knowledge Panel in Google Chrome



Nicole Kidman

American-Australian actress

nicolekidmanofficial.com

Nicole Mary Kidman AC is an American and Australian actress and producer. Known for her work across various film and television productions from several genres, she has continuously remained one of the world's highest-paid actresses. Wikipedia

What is spaCy?

spaCy

- spaCy is a free, open-source library for advanced natural language processing in Python:

  https://spacy.io/
- spaCy can be used to build information extraction systems, natural language understanding systems and to pre-process text for machine learning.
- It excels at large-scale information extraction tasks.
- It features pretrained neural models for tagging, entity recognition and dependency parsing.

Constructing a Knowledge Graph in Python

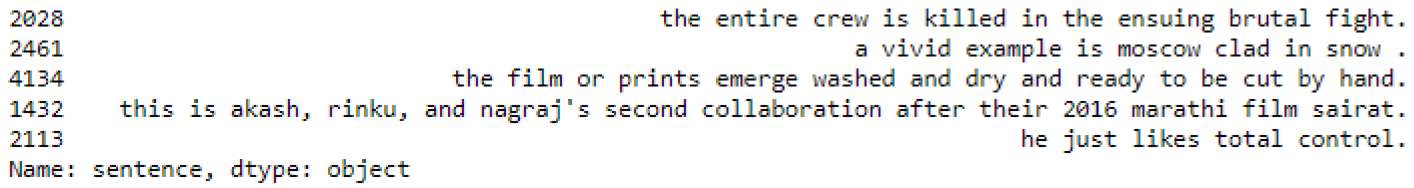
```
# Based on Python code written by Prateek Joshi, adapted for
# COMP3220 (Document Processing and Semantic Technologies).
# Libraries
import pandas as pd

# A Python library for graphs and networks.
import networkx as nx

# Matplotlib is a low level graph plotting library.
import matplotlib.pyplot as plt
%matplotlib inline

# Progression bar
from tqdm import tqdm

pd.set_option('display.max_colwidth', 200)
```

spaCy

```
# spaCy is an open-source software library for advanced
# natural language processing.

import spacy
from spacy.matcher import Matcher

# The en_core_web_sm model for English was trained on written
# web text (blogs, news, comments), and includes vocabulary,
# vectors, syntax and entities.

nlp = spacy.load('en_core_web_sm')
```

Load csv file

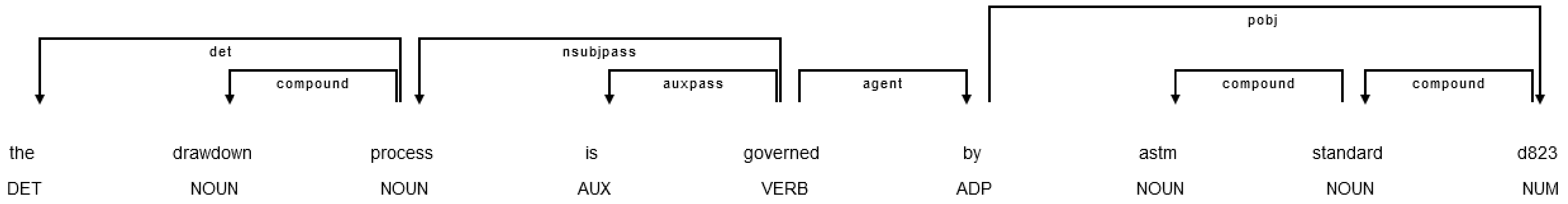
```
# Reads csv file that contains more than 4300 sentences
# extracted from 500 Wikipedia articles.
# Each of these sentences contains exactly two entities:
# one subject and one object.

candidate_sentences = pd.read_csv("wiki_sentences_v2.csv")

candidate_sentences['sentence'].sample(5)
```



```
2028                                            the entire crew is killed in the ensuing brutal fight.
2461                                                           a vivid example is moscow clad in snow .
4134                                   the film or prints emerge washed and dry and ready to be cut by hand.
1432    this is akash, rinku, and nagraj's second collaboration after their 2016 marathi film sairat.
2113                                                                           he just likes total control.
Name: sentence, dtype: object
```

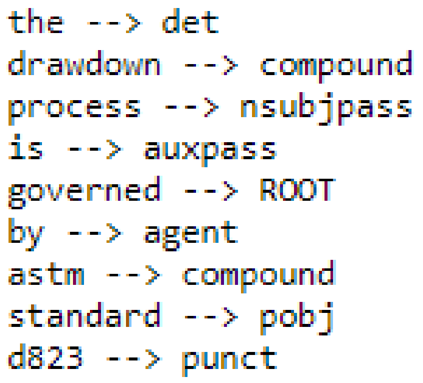
Visualising the Dependency Parse

```
# Let's display the dependency parse of a sentence.
from spacy import displacy

doc = nlp("the drawdown process is governed by astm standard d823")

options = {"compact": True}
displacy.render(doc, style="dep", options=options, jupyter=True)
```



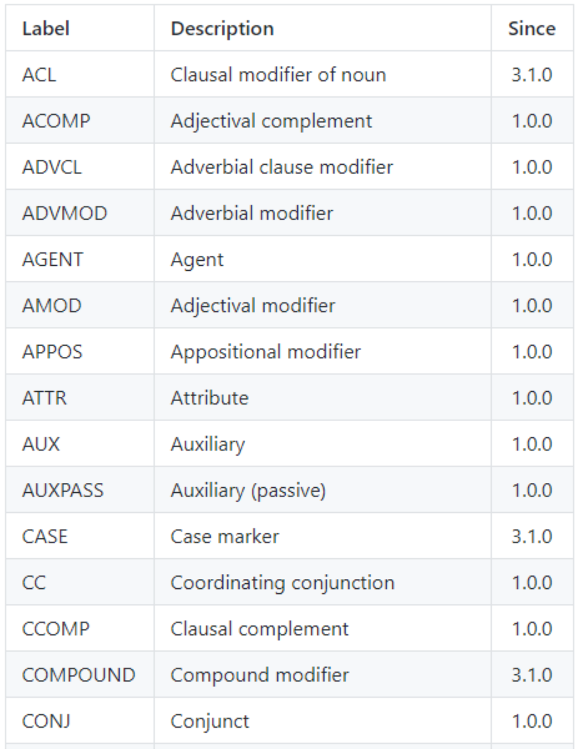
Inspecting the Subject and Object

```
# Let's check the subject and object of this sentence.

for tok in doc:
  print(tok.text, "-->", tok.dep_)
```



```
the --> det
drawdown --> compound
process --> nsubjpass
is --> auxpass
governed --> ROOT
by --> agent
astm --> compound
standard --> pobj
d823 --> punct
```

Some ClearNLP Dependency Labels



Label	Description	Since
ACL	Clausal modifier of noun	3.1.0
ACOMP	Adjectival complement	1.0.0
ADVCL	Adverbial clause modifier	1.0.0
ADVMOD	Adverbial modifier	1.0.0
AGENT	Agent	1.0.0
AMOD	Adjectival modifier	1.0.0
APPOS	Appositional modifier	1.0.0
ATTR	Attribute	1.0.0
AUX	Auxiliary	1.0.0
AUXPASS	Auxiliary (passive)	1.0.0
CASE	Case marker	3.1.0
CC	Coordinating conjunction	1.0.0
CCOMP	Clausal complement	1.0.0
COMPOUND	Compound modifier	3.1.0
CONJ	Conjunct	1.0.0

https://github.com/clir/clearnlp-guidelines/blob/master/md/specifications/dependency_labels.md

Get Entities

```
def get_entities(sent):

  ent1 = ""               # Variable for storing the subject.
  ent2 = ""               # Variable for storing the object.

  prv_tok_dep = ""        # Variable for dependency tag of previous
                          # token in the sentence.
  prv_tok_text = ""       # Variable for previous token in the sentence.

  prefix = ""             # Variable for storing compounds.
  modifier = ""           # Variable for storing modifiers.
```

Get Entities (Workshop Task)

```
# Loop through the tokens in the sentence.
## YOUR CODE GOES HERE.
  # Check if a token is a punctuation mark or not.
  ## YOUR CODE GOES HERE.
    # Check if a token is a compound one or not.
    ## YOUR CODE GOES HERE.
      # If yes, then store the token in the prefix variable.
      ## YOUR CODE GOES HERE.
      # Check if the previous token was also a compound one.
      ## YOUR CODE GOES HERE.
        # If yes, then update the prefix variable.
        ## YOUR CODE GOES HERE.
```

Get Entities (Workshop Task)

```
# Check if a token is a modifier or not.
## YOUR CODE GOES HERE.
  # If yes, then store the token in the modifier variable.
  ## YOUR CODE GOES HERE.
  # Check if the previous token was a compound one.
  ## YOUR CODE GOES HERE.
    # If yes, then update the modifier variable.
    ## YOUR CODE GOES HERE.
```

Get Entities (Workshop Task)

```
# Check if a token is the subject.
## YOUR CODE GOES HERE.
  # If yes, then concatenate the modifier, prefix, and token
  # and assign the result to the subject variable (ent1).
  ## YOUR CODE GOES HERE.
  # Reset the following variables: prefix, modifier,
  # prv_tok_dep, and prv_tok_text.
  ## YOUR CODE GOES HERE.
```

Get Entities (Workshop Task)

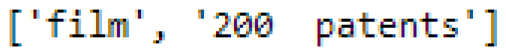
```
# Check if a token is the object.
   ## YOUR CODE GOES HERE.
     # If yes, then concatenate the modifier, prefix, and token
     # and assign the result to the modifier variable (ent2).
     ## YOUR CODE GOES HERE.
```

Get Entities (Workshop Task)

```
        # Update the variable for the dependency tag
        # for the previous token.
        ## YOUR CODE GOES HERE.
        # Update the variable for the previous token in the sentence.
        ## YOUR CODE GOES HERE.

    return [ent1.strip(), ent2.strip()]
```

Test Function

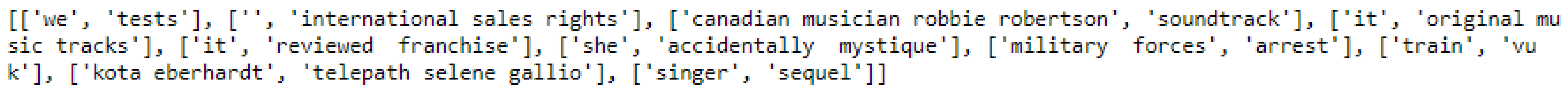
```
# Test function:

print(get_entities("the film had 200 patents"))
```



```
['film', '200  patents']
```

Extract Entity Pairs: [Subject, Object]

```
# Extracts these entity pairs (subject, object) for all the sentences.

entity_pairs = []

for i in tqdm(candidate_sentences["sentence"]):
    entity_pairs.append(get_entities(i))

print(entity_pairs[10:20])
```



```
[['we', 'tests'], ['', 'international sales rights'], ['canadian musician robbie robertson', 'soundtrack'], ['it', 'original mu
sic tracks'], ['it', 'reviewed  franchise'], ['she', 'accidentally  mystique'], ['military  forces', 'arrest'], ['train', 'vu
k'], ['kota eberhardt', 'telepath selene gallio'], ['singer', 'sequel']]
```

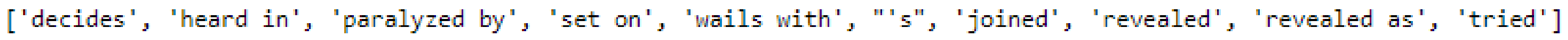
Extract Predicates (Workshop Task)

```
# Extracts the predicates from the sentences using spaCy's
# rule-based pattern matcher:
# https://spacy.io/usage/rule-based-matching

def get_predicate(sent):

  # YOUR CODE GOES HERE

  return(predicate)

# Test function:

print(get_predicate("John completed the task"))
```

```
completed
```

Get all the Predicates

```
# Get the predicates from all the Wikipedia sentences:

predicates = [get_predicate(i) for i in
              tqdm(candidate_sentences['sentence'])]

print(predicates[:10])
```



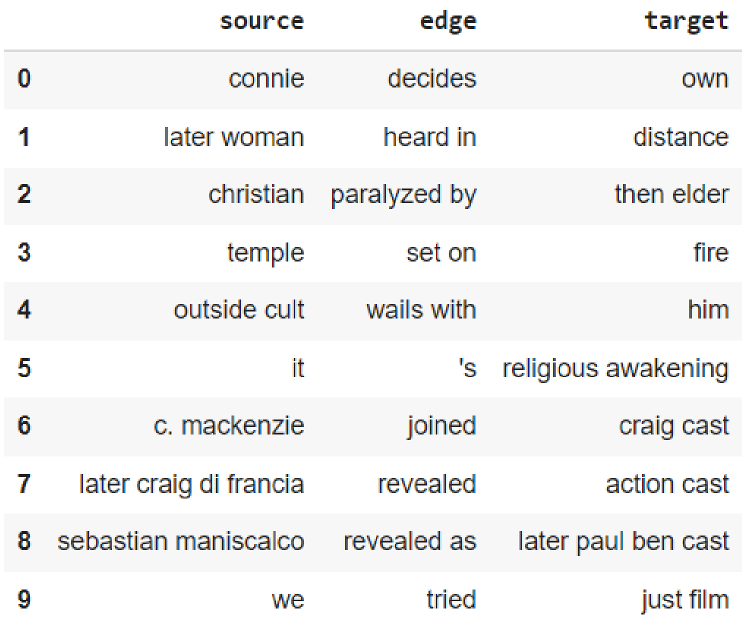
```
['decides', 'heard in', 'paralyzed by', 'set on', 'wails with', "'s", 'joined', 'revealed', 'revealed as', 'tried']
```

Extract Subject and Objects

```
# Extract subjects form entity pairs.
subjects = [i[0] for i in entity_pairs]

# Extract objects from entity pairs.
objects = [i[1] for i in entity_pairs]
```

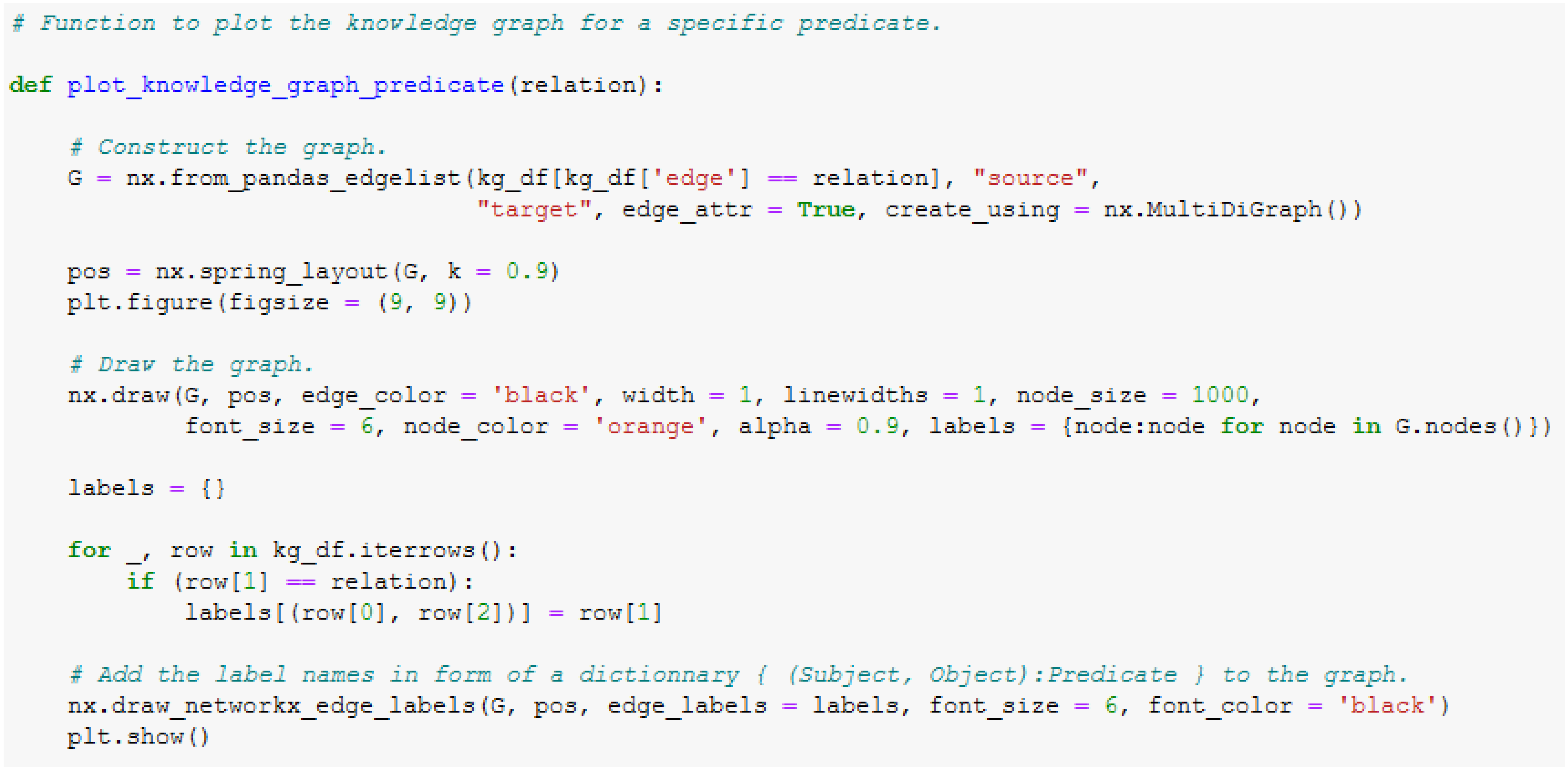
Build a DataFrame for KG Construction

```
# Build a pandas DataFrame of source, edge, target for
# the knowledge graph construction

kg_df = pd.DataFrame({'source':subjects, 'edge':predicates,
                      'target':objects})

kg_df.head(10)
```



	source	edge	target
0	connie	decides	own
1	later woman	heard in	distance
2	christian	paralyzed by	then elder
3	temple	set on	fire
4	outside cult	wails with	him
5	it	's	religious awakening
6	c. mackenzie	joined	craig cast
7	later craig di francia	revealed	action cast
8	sebastian maniscalco	revealed as	later paul ben cast
9	we	tried	just film

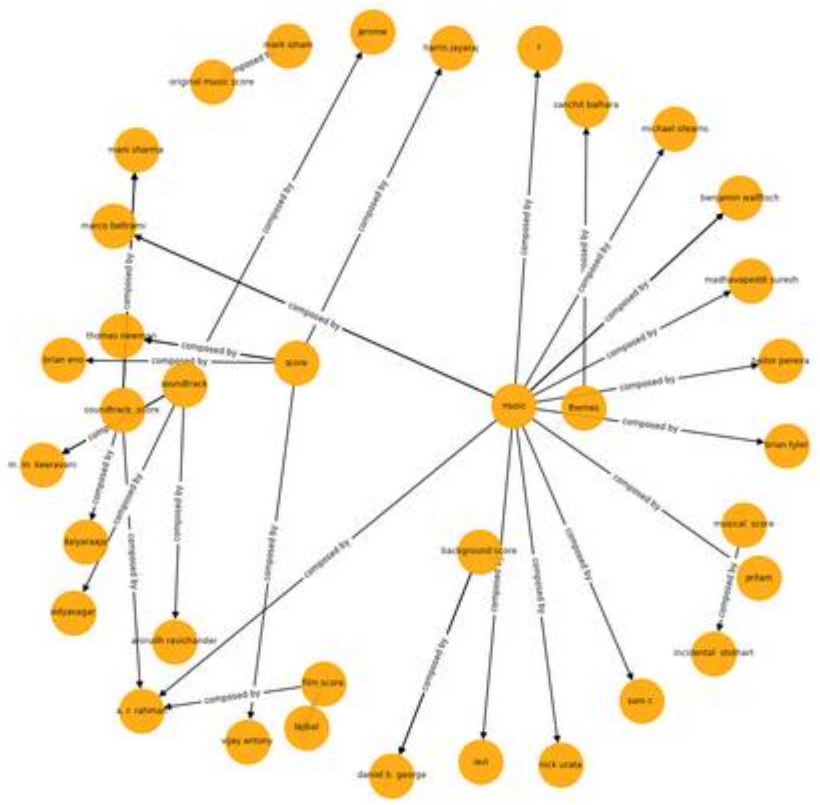
Function for Plotting a Predicate



```python
# Function to plot the knowledge graph for a specific predicate.

def plot_knowledge_graph_predicate(relation):

    # Construct the graph.
    G = nx.from_pandas_edgelist(kg_df[kg_df['edge'] == relation], "source",
                                "target", edge_attr = True, create_using = nx.MultiDiGraph())

    pos = nx.spring_layout(G, k = 0.9)
    plt.figure(figsize = (9, 9))

    # Draw the graph.
    nx.draw(G, pos, edge_color = 'black', width = 1, linewidths = 1, node_size = 1000,
            font_size = 6, node_color = 'orange', alpha = 0.9, labels = {node:node for node in G.nodes()})

    labels = {}

    for _, row in kg_df.iterrows():
        if (row[1] == relation):
            labels[(row[0], row[2])] = row[1]

    # Add the label names in form of a dictionnary { (Subject, Object):Predicate } to the graph.
    nx.draw_networkx_edge_labels(G, pos, edge_labels = labels, font_size = 6, font_color = 'black')
    plt.show()
```

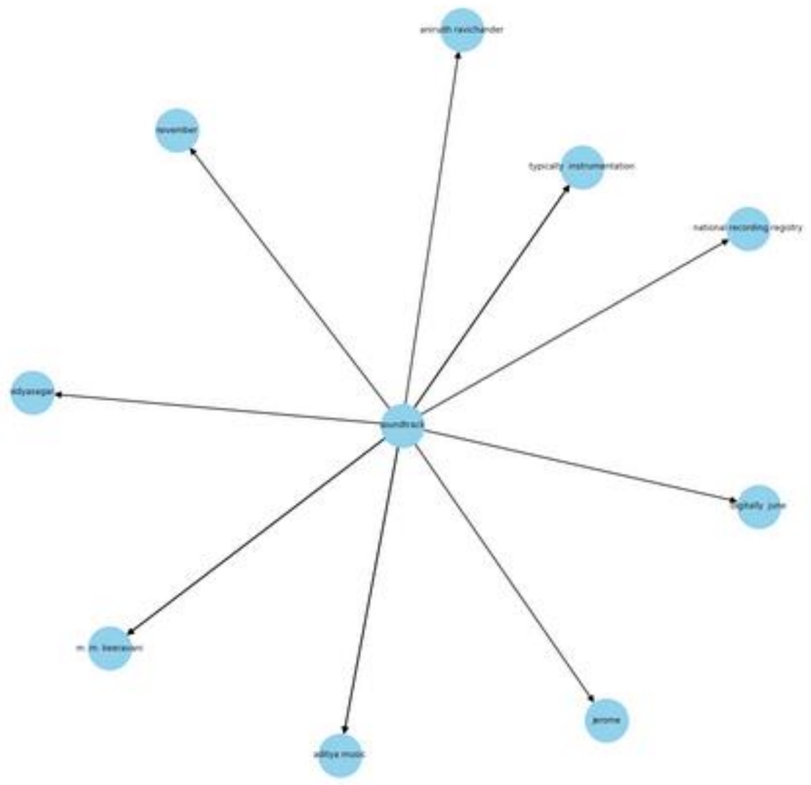
Knowledge Graph for a Specific Predicate

```
plot_knowledge_graph_predicate('composed by')
```



Knowledge Graph for a Specific Entity

```
plot_knowledge_graph_entity('soundtrack')
```



Take-Home Messages

- Knowledge graphs are a form of (semantic) networks, usually limited to a specific domain, and managed as a graph.
- The smallest knowledge graph consists of two nodes that are connected by an edge; also known as a triple.
- DBpedia and Wikidata are publicly available knowledge graphs.
- These knowledge graphs can be queried (if formalised) using standard query languages such as SPARQL.
- spaCy is an open source industrial-strength natural language processing library in Python.
- Note: The KG that we built so far in Python does not have a "formal" foundation; it is still difficult to use it for reasoning.