Search Medium Write

Image by author

Member-only story

NLP with Python: Knowledge Graph

SpaCy, Sentence segmentation, Part-Of-Speech tagging, Dependency parsing, Named Entity Recognition, and more...

Mauro Di Pietro · Following

Published in Towards Data Science · 14 min read · Apr 19

423 1

Summary

In this article, I will show how to build a Knowledge Graph with Python and Natural Language Processing.

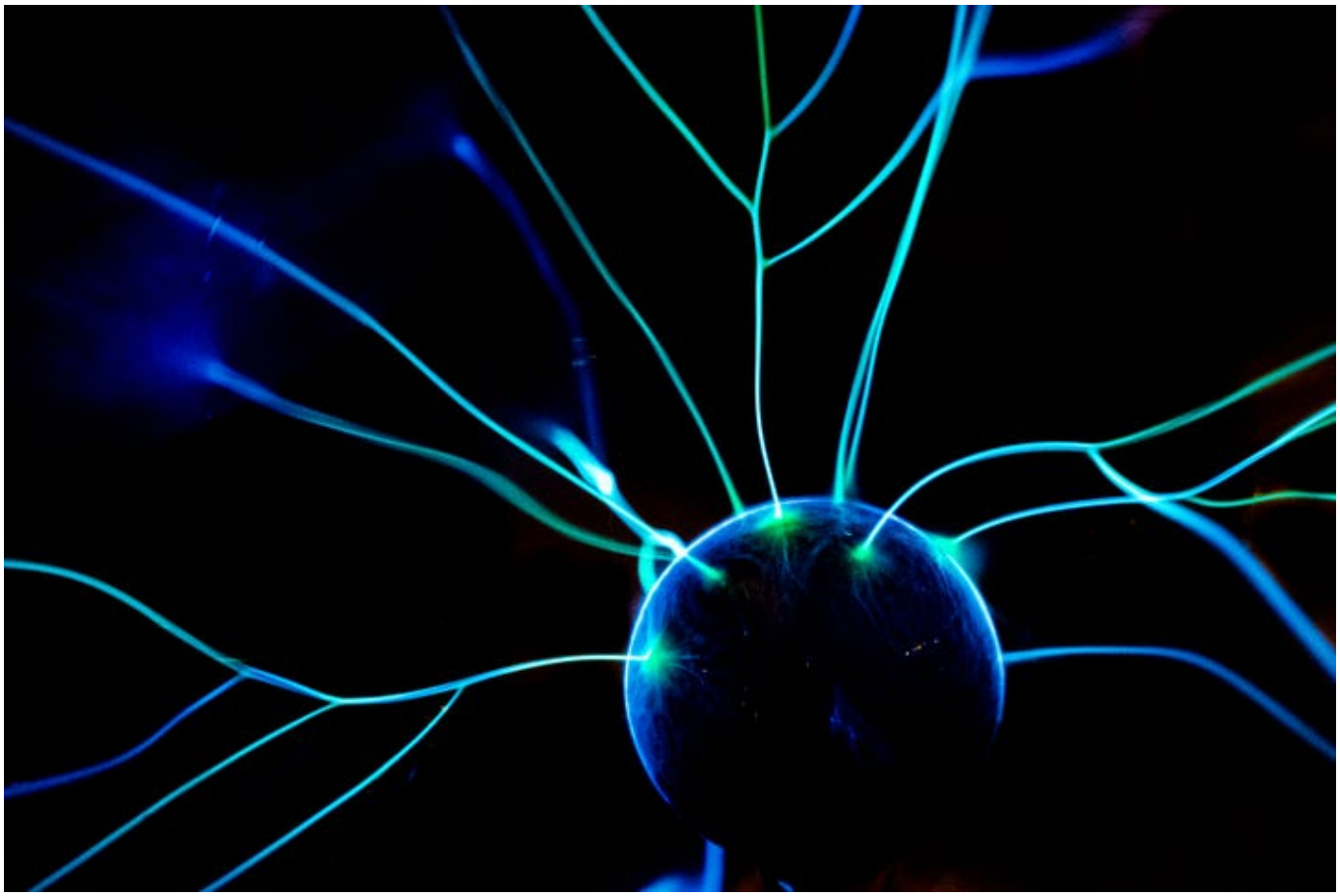


Photo by Moritz Kindler on Unsplash

A **network graph** is a mathematical structure to show relations between points that can be visualized with undirected/directed graph structures. It's a form of database that maps linked nodes.

A **knowledge base** is a unified repository of information from different sources, like *Wikipedia*.

A **Knowledge Graph** is a knowledge base that uses a graph-structured data model. To put it in simple words, it's a particular type of network graph that shows qualitative relationships between real-world entities, facts, concepts and events. The term "Knowledge Graph" was used for the first time by *Google* in 2012 to introduce their model.

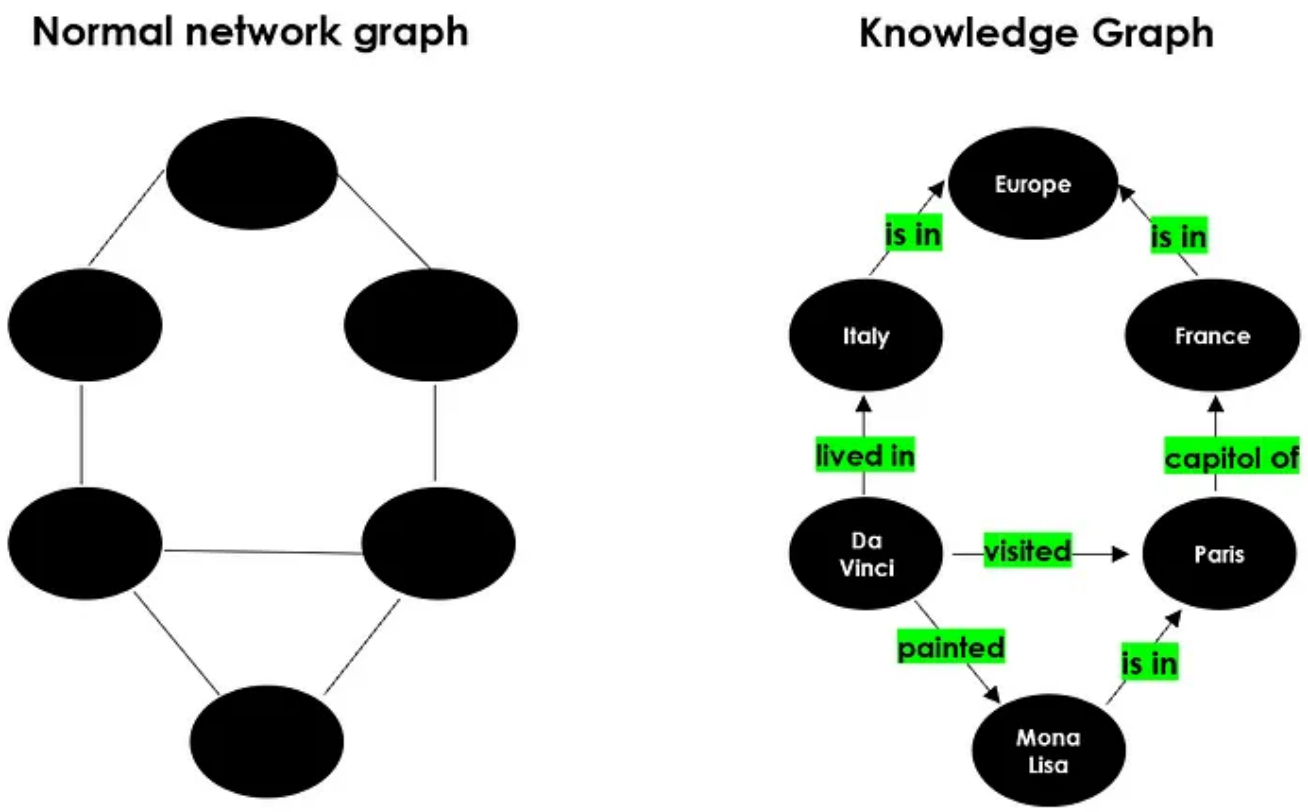


Image by author

Currently, most companies are building Data Lakes, a central database in which they toss raw data of all types (i.e. structured and unstructured) taken from different sources. Therefore, people need tools to make sense of all those pieces of different information. Knowledge Graphs are becoming popular as they can simplify exploration of large datasets and insight discovery. To put it in another way, a Knowledge Graph connects data and associated metadata, so it can be used to build a comprehensive representation of an organization's information assets. For instance, a Knowledge Graph might replace all the piles of documents you have to go through in order to find one particular information.

Knowledge Graphs are considered part of the Natural Language Processing landscape because, in order to build "knowledge", you must go through a

process called "**semantic enrichment**". Since nobody wants to do that manually, we need machines and NLP algorithms to perform this task for us.

I will present some useful Python code that can be easily applied in other similar cases (just copy, paste, run) and walk through every line of code with comments so that you can replicate this example (link to the full code below).

DataScience_ArtificialIntelligence_Utils/example_knowledge_graph.ipynb at master ·...

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com

I will parse Wikipedia and extract a page that shall be used as the dataset of this tutorial (link below).

Russo-Ukrainian War - Wikipedia

The Russo-Ukrainian War is an ongoing international conflict between Russia, alongside Russian-backed separatists, and...

en.wikipedia.org

In particular, I will go through:

- Setup: read packages and data with web scraping with *Wikipedia-API*.
- NLP with *SpaCy*: Sentence segmentation, POS tagging, Dependency parsing, NER.
- Extraction of Entities and their Relations with *Textacy*.

- Network Graph building with *NetworkX*.
- Timeline Graph with *DateParser*.

Setup

First of all, I need to import the following libraries:

```
## for data
import pandas as pd  #1.1.5
import numpy as np  #1.21.0

## for plotting
import matplotlib.pyplot as plt  #3.3.2

## for text
import wikipediaapi  #0.5.8
import nltk  #3.8.1
import re

## for nlp
import spacy  #3.5.0
from spacy import displacy
import textacy  #0.12.0

## for graph
import networkx as nx  #3.0 (also pygraphviz==1.10)

## for timeline
import dateparser #1.1.7
```

Wikipedia-api is the Python wrapper that easily lets you parse Wikipedia pages. I shall extract the page I want, excluding all the “notes” and “bibliography” at the bottom:

United States

On 28 April 2022, US President Joe Biden asked Congress for an additional $33 billion to assist Ukrain Ukraine.[394] On 5 May, Ukraine's Prime Minister Denys Shmyhal announced that Ukraine had received financial aid from Western countries since the start of Russia's invasion on 24 February.[395] On 21 May providing $40 billion in new military and humanitarian foreign aid to Ukraine, marking a historically large defense spending to counter the Russian war effort exceeded the first 5 years of war costs in Afghanist weapons delivered to the Ukrainian war front suggest a closer combat scenario with more casualties.[39 strength in Ukraine" with increased arms shipments and a record-breaking $3 billion military aid packag

Russian military suppliers

After expending large amounts of heavy weapons and munitions over months, the Russian Federation from Iran, deliveries of tanks and other armoured vehicles from Belarus, and reportedly planned to trad ballistic missiles from Iran.[399][400][401][402]

China may be providing Russia technology it needs for high-tech weapons, and the United States sanc to Russian mercenary forces fighting in Ukraine.[403]

STOP

See also

- Outline of the Russo-Ukrainian War
- List of conflicts in Europe
- List of invasions and occupations of Ukraine
- List of ongoing armed conflicts
- List of wars involving Russia
- List of wars involving Ukraine
- Modern history of Ukraine
- New generation warfare
- Russia under Vladimir Putin

Notes

a. ^ Self-declared republic since 7 April 2014; annexation by Russia declared on 30 September 2022.
b. ^ Self-declared republic since 27 April 2014; annexation by Russia declared on 30 September 2022.
c. ^ For further details, see Belarusian involvement in the 2022 Russian invasion of Ukraine.

from Wikipedia

We can simply write the name of the page:

```
topic = "Russo-Ukrainian War"

wiki = wikipediaapi.Wikipedia('en')
page = wiki.page(topic)
```

```
txt = page.text[:page.text.find("See also")]
txt[0:500] + " ..."
```

```
"The Russo-Ukrainian War is an ongoing international conflict between Russia, alongside Russian-backed separatists, and Ukraine, which began in Februa
ry 2014. Following Ukraine's Revolution of Dignity, Russia annexed Crimea from Ukraine and supported pro-Russian separatists fighting the Ukrainian mi
litary in the Donbas war. The first eight years of conflict also included naval incidents, cyberwarfare, and heightened political tensions. In Februar
y 2022, Russia launched a full-scale invasion of Uk ..."
```

In this usecase, I will try to map historical events by identifying and extracting subjects-actions-objects from the text (so the action is the relation).

NLP

In order to build a Knowledge Graph, we need first to identify entities and their relations. Therefore, we need to process the text dataset with NLP techniques.

Currently, the most used library for this type of task is *SpaCy*, an open-source software for advanced NLP that leverages *Cython* (C+Python). *SpaCy* uses pre-trained language models to tokenize the text and transform it into an object commonly called "document", basically a class that contains all the annotations predicted by the model.

```
#python -m spacy download en_core_web_sm

nlp = spacy.load("en_core_web_sm")
doc = nlp(txt)
```

The first output of the NLP model is **Sentence segmentation**: the problem of deciding where a sentence begins and ends. Usually, it's done by splitting

paragraphs based on punctuation. Let's see how many sentences *SpaCy* split the text into:

```
# from text to a list of sentences
lst_docs = [sent for sent in doc.sents]
print("tot sentences:", len(lst_docs))
```

```
tot sentences: 372
```

Image by author

Now, for each sentence, we are going to extract entities and their relations. In order to do that, first we need to understand **Part-of-Speech (POS) tagging:** the process of labeling each word in a sentence with its appropriate grammar tag. Here's the full list of possible tags (as of today):

> - **ADJ***: adjective, e.g. big, old, green, incomprehensible, first*
> - **ADP***: adposition (preposition/postposition) e.g. in, to, during*
> - **ADV***: adverb, e.g. very, tomorrow, down, where, there*
> - **AUX***: auxiliary, e.g. is, has (done), will (do), should (do)*
> - **CONJ***: conjunction, e.g. and, or, but*
> - **CCONJ***: coordinating conjunction, e.g. and, or, but*
> - **DET***: determiner, e.g. a, an, the*
> - **INTJ***: interjection, e.g. psst, ouch, bravo, hello*
> - **NOUN***: noun, e.g. girl, cat, tree, air, beauty*
> - **NUM***: numeral, e.g. 1, 2017, one, seventy-seven, IV, MMXIV*
> - **PART***: particle, e.g. 's, not*
> - **PRON***: pronoun, e.g I, you, he, she, myself, themselves, somebody*
> - **PROPN***: proper noun, e.g. Mary, John, London, NATO, HBO*

- **PUNCT***: punctuation, e.g. ., (,), ?*
- **SCONJ***: subordinating conjunction, e.g. if, while, that*
- **SYM***: symbol, e.g. $, %, §, ©, +, −, ×, ÷, =, :), emojis*
- **VERB***: verb, e.g. run, runs, running, eat, ate, eating*
- **X***: other, e.g. sfpksdpsxmsa*
- **SPACE***: space, e.g.*

POS tagging alone is not enough, the model also tries to understand the relationship between pairs of words. This task is called **Dependency (DEP) parsing**. Here's the full list of possible tags (as of today):

- **ACL***: clausal modifier of noun*
- **ACOMP***: adjectival complement*
- **ADVCL:** *adverbial clause modifier*
- **ADVMOD:** *adverbial modifier*
- **AGENT:** *agent*
- **AMOD:** *adjectival modifier*
- **APPOS:** *appositional modifier*
- **ATTR:** *attribute*
- **AUX:** *auxiliary*
- **AUXPASS:** *auxiliary (passive)*
- **CASE:** *case marker*
- **CC***: coordinating conjunction*
- **CCOMP:** *clausal complement*
- **COMPOUND:** *compound modifier*
- **CONJ:** *conjunct*
- **CSUBJ:** *clausal subject*
- **CSUBJPASS:** *clausal subject (passive)*
- **DATIVE:** *dative*
- **DEP:** *unclassified dependent*

- **DET:** *determiner*
- **DOBJ:** *direct object*
- **EXPL:** *expletive*
- **INTJ:** *interjection*
- **MARK:** *marker*
- **META:** *meta modifier*
- **NEG:** *negation modifier*
- **NOUNMOD:** *modifier of nominal*
- **NPMOD:** *noun phrase as adverbial modifier*
- **NSUBJ:** *nominal subject*
- **NSUBJPASS:** *nominal subject (passive)*
- **NUMMOD:** *number modifier*
- **OPRD:** *object predicate*
- **PARATAXIS:** *parataxis*
- **PCOMP:** *complement of preposition*
- **POBJ:** *object of preposition*
- **POSS:** *possession modifier*
- **PRECONJ***: pre-correlative conjunction*
- **PREDET:** *pre-determiner*
- **PREP:** *prepositional modifier*
- **PRT:** *particle*
- **PUNCT:** *punctuation*
- **QUANTMOD:** *modifier of quantifier*
- **RELCL:** *relative clause modifier*
- **ROOT:** *root*
- **XCOMP:** *open clausal complement*

Let’s make an example to understand POS tagging and DEP parsing:

```
# take a sentence
i = 3
lst_docs[i]
```

```
In February 2022, Russia launched a full-scale invasion of Ukraine.
```

Let's check the POS and DEP tags predicted by the NLP model:

```
for token in lst_docs[i]:
    print(token.text, "-->", "pos: "+token.pos_, "|", "dep: "+token.dep_, "")
```

```
In --> pos: ADP | dep: prep
February --> pos: PROPN | dep: pobj
2022 --> pos: NUM | dep: nummod
, --> pos: PUNCT | dep: punct
Russia --> pos: PROPN | dep: nsubj
launched --> pos: VERB | dep: ROOT
a --> pos: DET | dep: det
full --> pos: ADJ | dep: amod
- --> pos: PUNCT | dep: punct
scale --> pos: NOUN | dep: compound
invasion --> pos: NOUN | dep: dobj
of --> pos: ADP | dep: prep
Ukraine --> pos: PROPN | dep: pobj
. --> pos: PUNCT | dep: punct
```

Image by author

SpaCy provides also a graphic tool to visualize those annotations:

```
from spacy import displacy

displacy.render(lst_docs[i], style="dep", options={"distance":100})
```

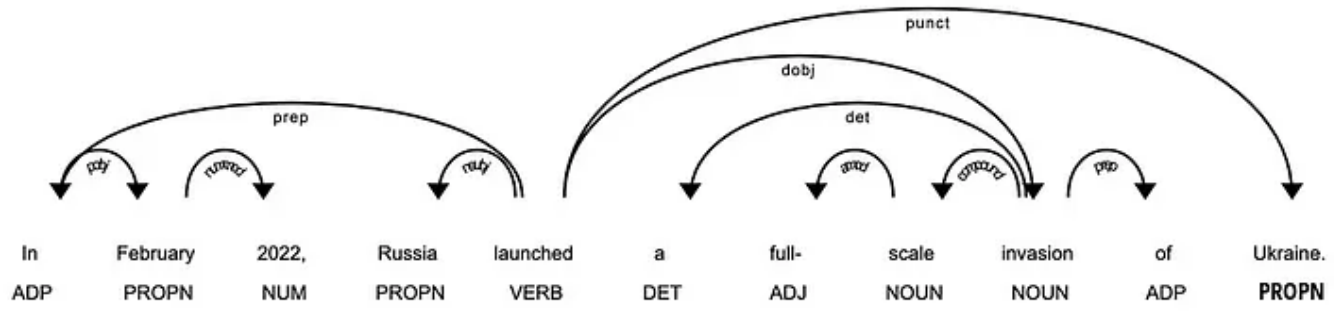


Image by author

The most important token is the verb (*POS=VERB*) because is the root (*DEP=ROOT*) of the meaning in a sentence.

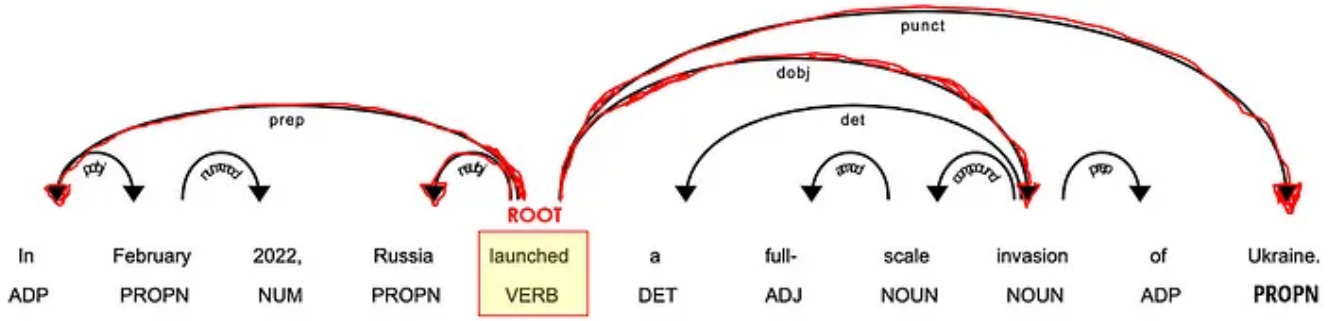


Image by author

Auxiliary particles, like adverbs and adpositions (*POS=ADV/ADP*), are often linked to the verb as modifiers (*DEP=*mod*), as they can modify the meaning of the verb. For instance, "*travel to*" and "*travel from*" have different meanings even though the root is the same ("*travel*").

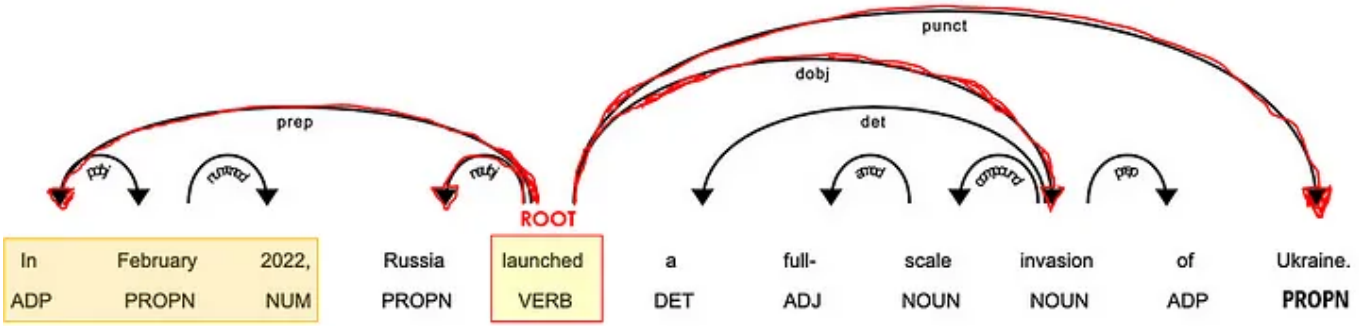


Image by author

Among the words linked to the verb, there must be some nouns (*POS=PROPN/NOUN*) that work as the subject and object (*DEP=nsubj/*obj*) of the sentence.

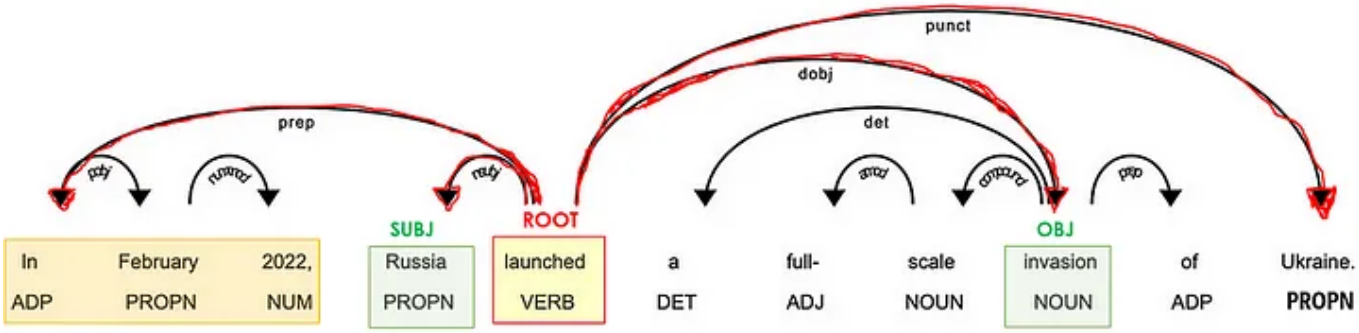


Image by author

Nouns are often near an adjective (*POS=ADJ*) that acts as a modifier of their meaning (*DEP=amod*). For instance, in "*good person*" and "*bad person*" the adjectives give opposite meanings to the noun *"person"*.

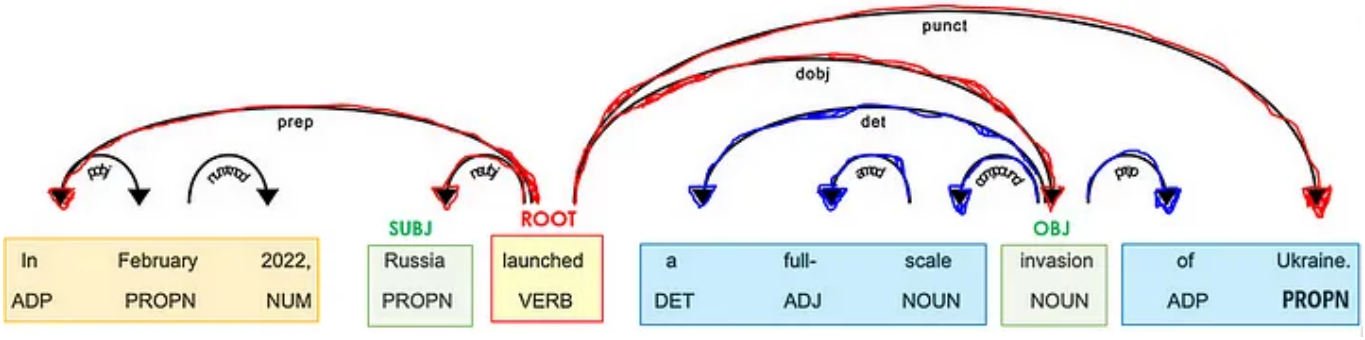


Image by author

Another cool task performed by *SpaCy* is **Named Entity Recognition (NER)**. A named entity is a "real-world object" (i.e. person, country, product, date) and models can recognize various types in a document. Here's the full list of possible tags (as of today):

> - **PERSON:** *people, including fictional.*
> - **NORP:** *nationalities or religious or political groups.*
> - **FAC:** *buildings, airports, highways, bridges, etc.*
> - **ORG:** *companies, agencies, institutions, etc.*
> - **GPE:** *countries, cities, states.*
> - **LOC:** *non-GPE locations, mountain ranges, bodies of water.*
> - **PRODUCT:** *objects, vehicles, foods, etc. (Not services.)*
> - **EVENT:** *named hurricanes, battles, wars, sports events, etc.*
> - **WORK_OF_ART:** *titles of books, songs, etc.*
> - **LAW:** *named documents made into laws.*
> - **LANGUAGE:** *any named language.*
> - **DATE:** *absolute or relative dates or periods.*
> - **TIME:** *times smaller than a day.*
> - **PERCENT:** *percentage, including "%".*
> - **MONEY:** *monetary values, including unit.*
> - **QUANTITY:** *measurements, as of weight or distance.*
> - **ORDINAL:** *"first", "second", etc.*
> - **CARDINAL:** *numerals that do not fall under another type.*

Let's see our example:

```
for tag in lst_docs[i].ents:
    print(tag.text, f"({tag.label_})")
```

```
February 2022 (DATE)
Russia (GPE)
Ukraine (GPE)
```

Image by author

or even better with *SpaCy* graphic tool:

```
displacy.render(lst_docs[i], style="ent")
```

Image by author

That is useful in case we want to add several attributes to our Knowledge Graph.

Moving on, using the tags predicted by the NLP model, we can extract entities and their relations.

Entity & Relation Extraction

The idea is very simple but the implementation can be tricky. For each sentence, we're going to extract the subject and object along with their modifiers, compound words, and punctuation marks between them.

This can be done in 2 ways:

1. **Manually**, you can start from the baseline code, which probably must be slightly modified and adapted to your specific dataset/usecase.

```
def extract_entities(doc):
    a, b, prev_dep, prev_txt, prefix, modifier = "", "", "", "", "", ""
    for token in doc:
        if token.dep_ != "punct":
            ## prexif --> prev_compound + compound
            if token.dep_ == "compound":
                prefix = prev_txt +" "+ token.text if prev_dep == "compound" els

            ## modifier --> prev_compound + %mod
            if token.dep_.endswith("mod") == True:
                modifier = prev_txt +" "+ token.text if prev_dep == "compound" e

            ## subject --> modifier + prefix + %subj
            if token.dep_.find("subj") == True:
                a = modifier +" "+ prefix + " "+ token.text
                prefix, modifier, prev_dep, prev_txt = "", "", "", ""

            ## if object --> modifier + prefix + %obj
            if token.dep_.find("obj") == True:
                b = modifier +" "+ prefix +" "+ token.text

            prev_dep, prev_txt = token.dep_, token.text

    # clean
    a = " ".join([i for i in a.split()])
    b = " ".join([i for i in b.split()])
    return (a.strip(), b.strip())


# The relation extraction requires the rule-based matching tool,
# an improved version of regular expressions on raw text.
def extract_relation(doc, nlp):
    matcher = spacy.matcher.Matcher(nlp.vocab)
    p1 = [{'DEP':'ROOT'},
          {'DEP':'prep', 'OP':"?"},
          {'DEP':'agent', 'OP':"?"},
          {'POS':'ADJ', 'OP':"?"}]
    matcher.add(key="matching_1", patterns=[p1])
    matches = matcher(doc)
    k = len(matches) - 1
    span = doc[matches[k][1]:matches[k][2]]
    return span.text
```

Let’s try it out on this dataset and check out the usual example:

```
## extract entities
lst_entities = [extract_entities(i) for i in lst_docs]

## example
lst_entities[i]
```

```
('2022 Russia', 'full scale Ukraine')
```

```
## extract relations
lst_relations = [extract_relation(i,nlp) for i in lst_docs]

## example
lst_relations[i]
```

```
'launched'
```

```
## extract attributes (NER)
lst_attr = []
for x in lst_docs:
    attr = ""
    for tag in x.ents:
        attr = attr+tag.text if tag.label_=="DATE" else attr+""
    lst_attr.append(attr)

## example
lst_attr[i]
```

```
'February 2022'
```

2. Alternatively, you can use ***Textacy***, a library built on top of *SpaCy* for extending its core functionalities. This is much more user-friendly and in general more accurate.

```
## extract entities and relations
dic = {"id":[], "text":[], "entity":[], "relation":[], "object":[]}

for n,sentence in enumerate(lst_docs):
    lst_generators = list(textacy.extract.subject_verb_object_triples(sentence))
    for sent in lst_generators:
        subj = "_".join(map(str, sent.subject))
        obj  = "_".join(map(str, sent.object))
        relation = "_".join(map(str, sent.verb))
        dic["id"].append(n)
        dic["text"].append(sentence.text)
        dic["entity"].append(subj)
        dic["object"].append(obj)
        dic["relation"].append(relation)


## create dataframe
dtf = pd.DataFrame(dic)

## example
dtf[dtf["id"]==i]
```

	id	text	entity	relation	object
3	3	In February 2022, Russia launched a full-scale...	Russia	launched	scale_invasion

Image by author

Let's extract also the attributes using NER tags (i.e. dates):

```
## extract attributes
attribute = "DATE"
```

```
dic = {"id":[], "text":[], attribute:[]}

for n,sentence in enumerate(lst_docs):
    lst = list(textacy.extract.entities(sentence, include_types={attribute}))
    if len(lst) > 0:
        for attr in lst:
            dic["id"].append(n)
            dic["text"].append(sentence.text)
            dic[attribute].append(str(attr))
    else:
        dic["id"].append(n)
        dic["text"].append(sentence.text)
        dic[attribute].append(np.nan)

dtf_att = pd.DataFrame(dic)
dtf_att = dtf_att[~dtf_att[attribute].isna()]

## example
dtf_att[dtf_att["id"]==i]
```

	id	text	date
3	3	In February 2022, Russia launched a full-scale...	February 2022

Image by author

Now that we have extracted “knowledge”, we can build the graph.

Network Graph

The standard Python library to create and manipulate graph networks is *NetworkX*. We can create the graph starting from the whole dataset but, if there are too many nodes, the visualization will be messy:

```
## create full graph
G = nx.from_pandas_edgelist(dtf, source="entity", target="object",
                            edge_attr="relation",
                            create_using=nx.DiGraph())
```

```
## plot
plt.figure(figsize=(15,10))

pos = nx.spring_layout(G, k=1)
node_color = "skyblue"
edge_color = "black"

nx.draw(G, pos=pos, with_labels=True, node_color=node_color,
        edge_color=edge_color, cmap=plt.cm.Dark2,
        node_size=2000, connectionstyle='arc3,rad=0.1')

nx.draw_networkx_edge_labels(G, pos=pos, label_pos=0.5,
                             edge_labels=nx.get_edge_attributes(G,'relation'),
                             font_size=12, font_color='black', alpha=0.6)
plt.show()
```

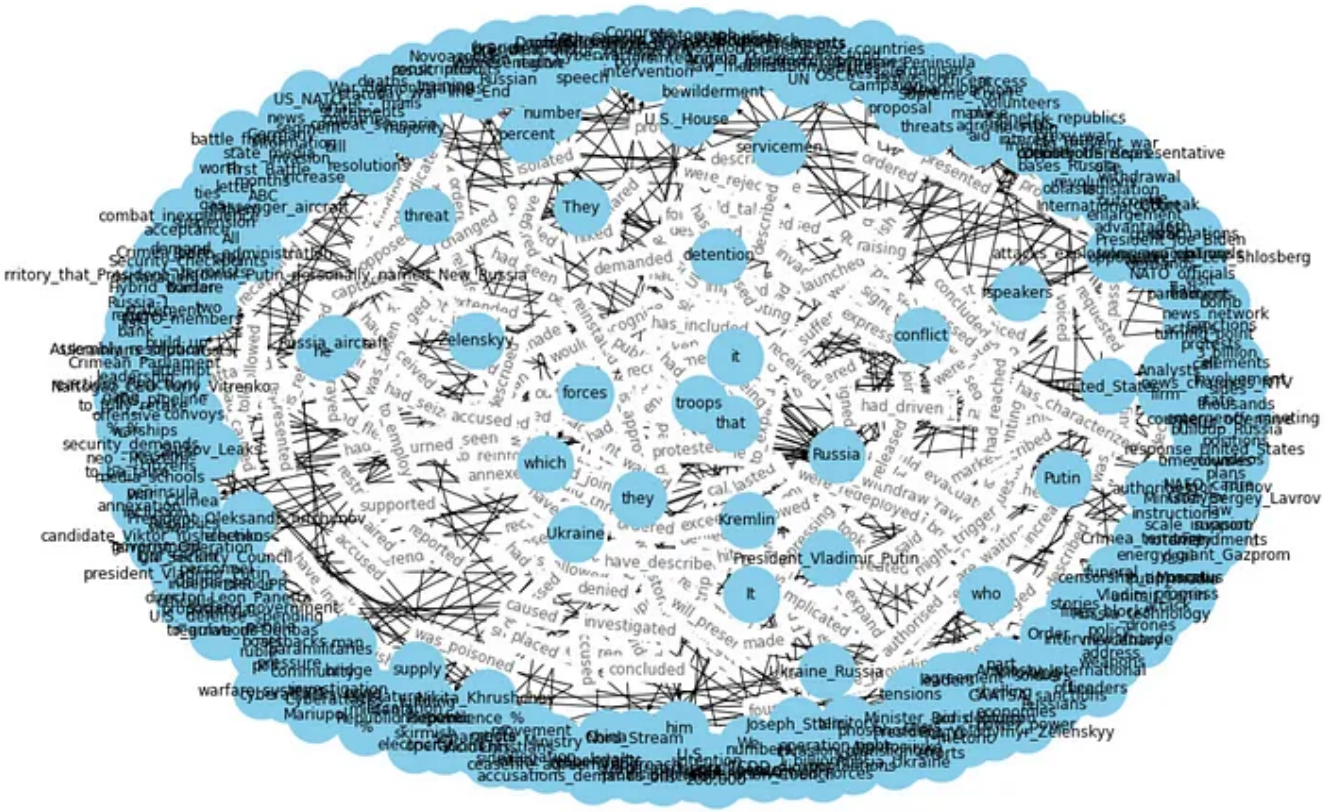


Image by author

Knowledge Graphs make it possible to see how everything is related at a big picture level, but like this is quite useless… so better to apply some filters

based on the information we are looking for. For this example, I shall take only the part of the graph involving the most frequent entity (basically the most connected node):

```
dtf["entity"].value_counts().head()
```

```
Russia     32
Putin      11
it         10
Ukraine     9
they        9
```

Image by author

```
## filter
f = "Russia"
tmp = dtf[(dtf["entity"]==f) | (dtf["object"]==f)]


## create small graph
G = nx.from_pandas_edgelist(tmp, source="entity", target="object",
                            edge_attr="relation",
                            create_using=nx.DiGraph())


## plot
plt.figure(figsize=(15,10))

pos = nx.nx_agraph.graphviz_layout(G, prog="neato")
node_color = ["red" if node==f else "skyblue" for node in G.nodes]
edge_color = ["red" if edge[0]==f else "black" for edge in G.edges]

nx.draw(G, pos=pos, with_labels=True, node_color=node_color,
        edge_color=edge_color, cmap=plt.cm.Dark2,
        node_size=2000, node_shape="o", connectionstyle='arc3,rad=0.1')

nx.draw_networkx_edge_labels(G, pos=pos, label_pos=0.5,
                             edge_labels=nx.get_edge_attributes(G,'relation'),
```

```
                        font_size=12, font_color='black', alpha=0.6)
plt.show()
```

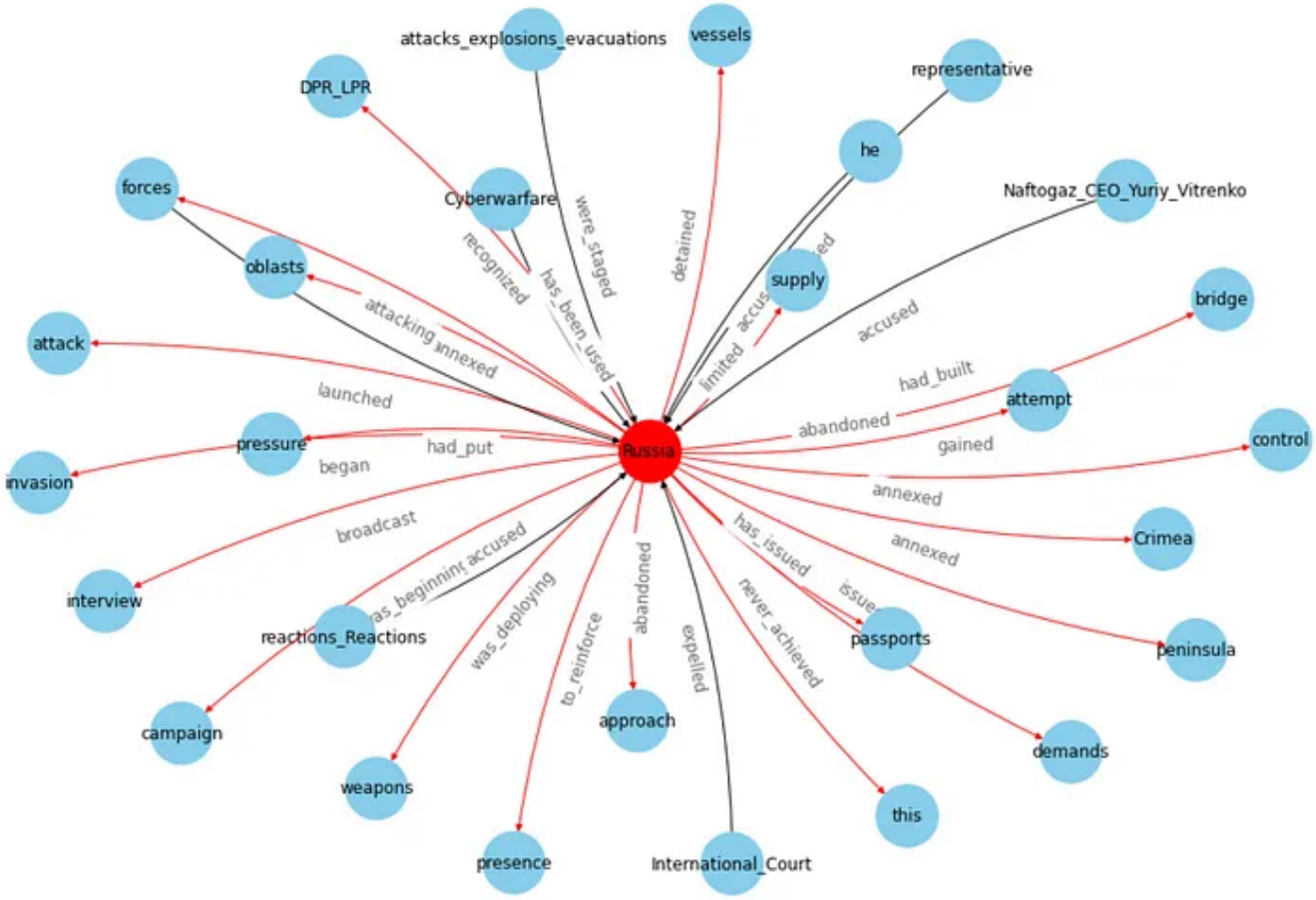


Image by author

That's better. And if you want to make it 3D, use the following code:

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(15,10))
ax = fig.add_subplot(111, projection="3d")
pos = nx.spring_layout(G, k=2.5, dim=3)

nodes = np.array([pos[v] for v in sorted(G) if v!=f])
center_node = np.array([pos[v] for v in sorted(G) if v==f])

edges = np.array([(pos[u],pos[v]) for u,v in G.edges() if v!=f])
```

```
center_edges = np.array([(pos[u],pos[v]) for u,v in G.edges() if v==f])

ax.scatter(*nodes.T, s=200, ec="w", c="skyblue", alpha=0.5)
ax.scatter(*center_node.T, s=200, c="red", alpha=0.5)

for link in edges:
    ax.plot(*link.T, color="grey", lw=0.5)
for link in center_edges:
    ax.plot(*link.T, color="red", lw=0.5)

for v in sorted(G):
    ax.text(*pos[v].T, s=v)
for u,v in G.edges():
    attr = nx.get_edge_attributes(G, "relation")[(u,v)]
    ax.text(*((pos[u]+pos[v])/2).T, s=attr)

ax.set(xlabel=None, ylabel=None, zlabel=None,
       xticklabels=[], yticklabels=[], zticklabels=[])
ax.grid(False)
for dim in (ax.xaxis, ax.yaxis, ax.zaxis):
    dim.set_ticks([])
plt.show()
```

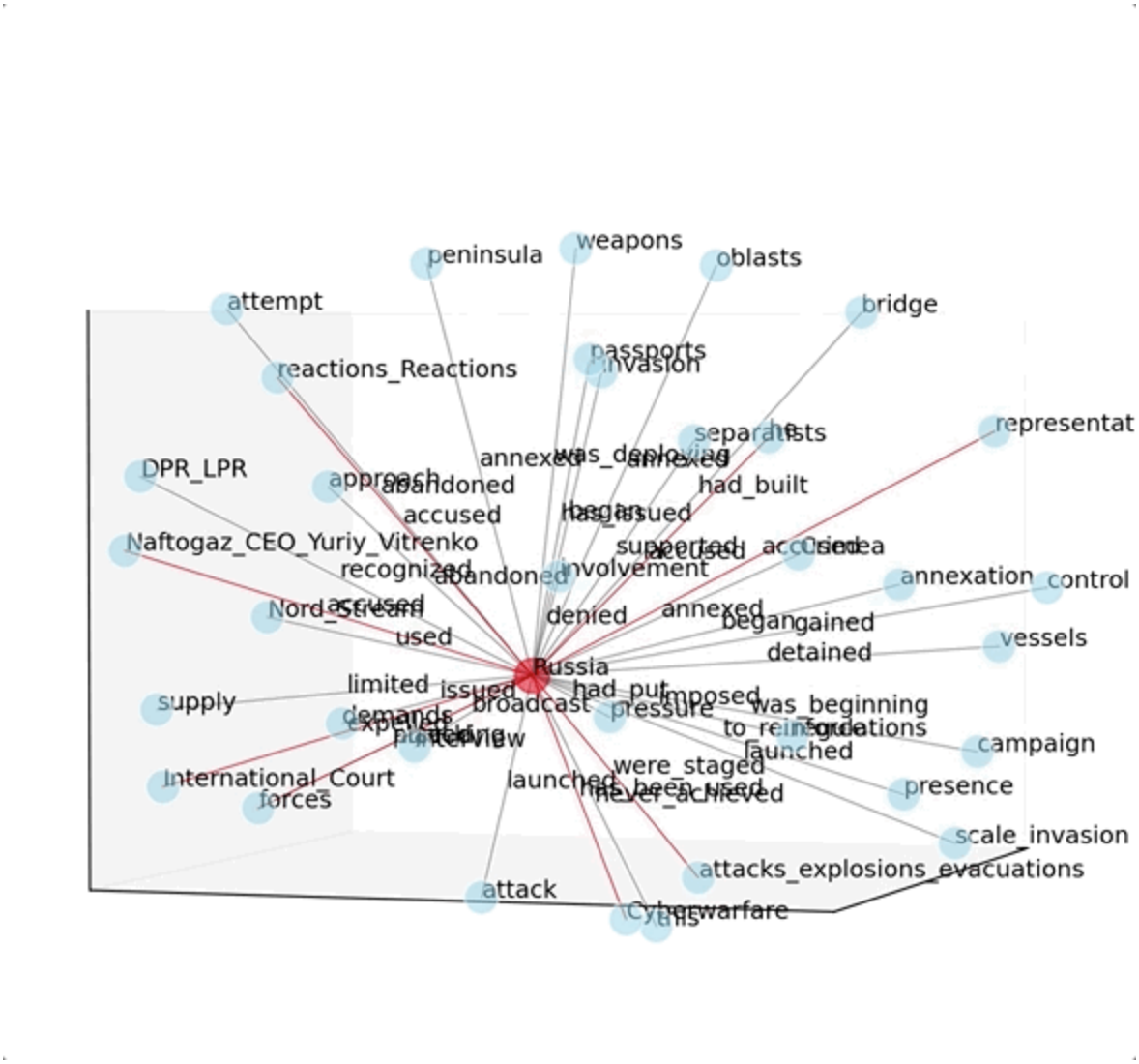


Image by author

Please note that a graph might be useful and nice to see, but it's not the main focus of this tutorial. The most important part of a Knowledge Graph is the "knowledge" (text processing), then results can be shown on a dataframe, a graph, or a different plot. For instance, I could use the dates recognized with NER to build a Timeline graph.

Timeline Graph

First of all, I have to transform the strings identified as a "date" to datetime format. The library ***DateParser*** parses dates in almost any string format commonly found on web pages.

```
def utils_parsetime(txt):
    x = re.match(r'.*([1-3][0-9]{3})', txt) #<--check if there is a year
    if x is not None:
        try:
            dt = dateparser.parse(txt)
        except:
            dt = np.nan
    else:
        dt = np.nan
    return dt
```

Let's apply it to the dataframe of attributes:

```
dtf_att["dt"] = dtf_att["date"].apply(lambda x: utils_parsetime(x))

## example
dtf_att[dtf_att["id"]==i]
```

	id	text	date	dt
3	3	In February 2022, Russia launched a full-scale...	February 2022	2022-02-13

Image by author

Now, I shall join it with the main dataframe of entities-relations:

```
tmp = dtf.copy()
tmp["y"] = tmp["entity"]+" "+tmp["relation"]+" "+tmp["object"]

dtf_att = dtf_att.merge(tmp[["id","y"]], how="left", on="id")
dtf_att = dtf_att[~dtf_att["y"].isna()].sort_values("dt",
```

```
                    ascending=True).drop_duplicates("y", keep='first')
dtf_att.head()
```

id	text	date	dt	y
24	After the dissolution of the Soviet Union (USS...	1991	1991-04-13	Ukraine_Russia maintained ties
29	In the years after the dissolution of the USSR...	1993	1993-04-13	Bloc_countries joined NATO
60	According to the original treaty on the divisi...	1997	1997-04-13	it would_evacuate units
28	In 1999, Russia was one of the signatories of ...	1999	1999-04-13	which reaffirmed right
292	In June 2014, several Russian state news outle...	2004	2004-04-13	Ukraine was_using phosphorus

Image by author

Finally, I can plot the timeline. As we already know, a full plot probably won't be useful:

```
dates = dtf_att["dt"].values
names = dtf_att["y"].values
l = [10,-10, 8,-8, 6,-6, 4,-4, 2,-2]
levels = np.tile(l, int(np.ceil(len(dates)/len(l))))[:len(dates)]

fig, ax = plt.subplots(figsize=(20,10))
ax.set(title=topic, yticks=[], yticklabels=[])

ax.vlines(dates, ymin=0, ymax=levels, color="tab:red")
ax.plot(dates, np.zeros_like(dates), "-o", color="k", markerfacecolor="w")

for d,l,r in zip(dates,levels,names):
    ax.annotate(r, xy=(d,l), xytext=(-3, np.sign(l)*3),
                textcoords="offset points",
                horizontalalignment="center",
                verticalalignment="bottom" if l>0 else "top")

plt.xticks(rotation=90)
plt.show()
```

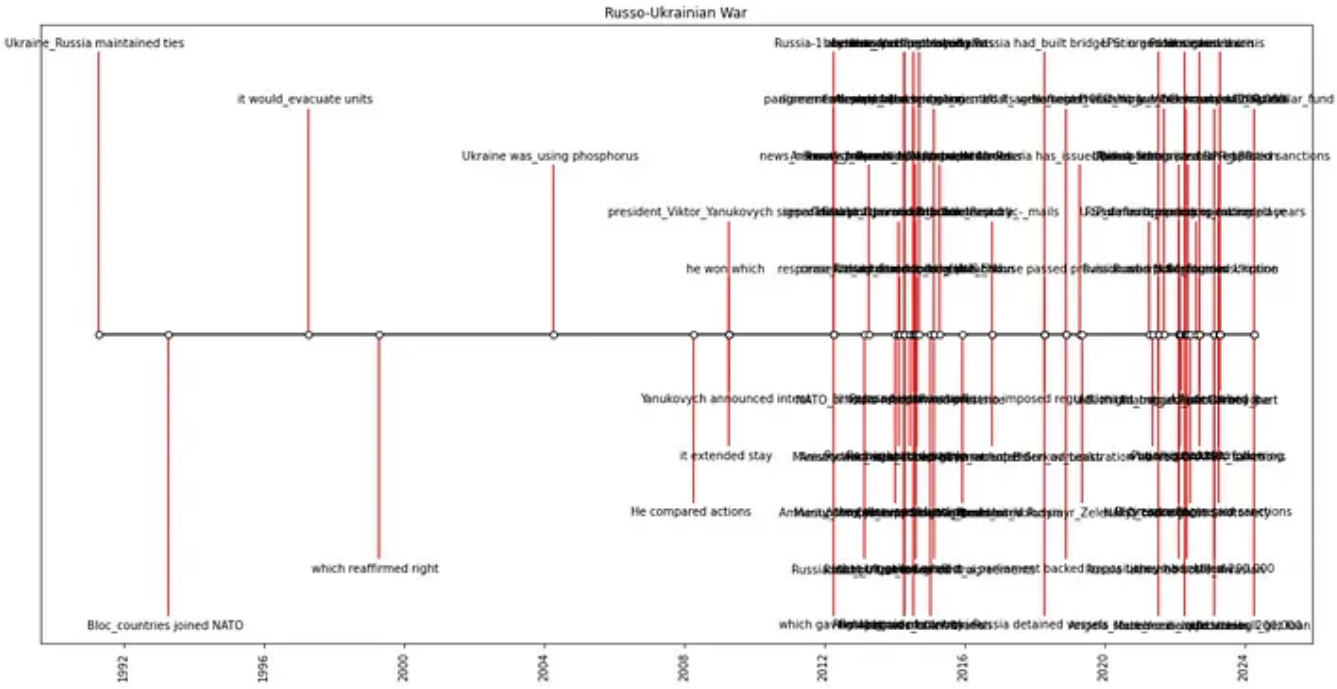


Image by author

So it's better to filter a specific time:

```
yyyy = "2022"
dates = dtf_att[dtf_att["dt"]>yyyy]["dt"].values
names = dtf_att[dtf_att["dt"]>yyyy]["y"].values
l = [10,-10, 8,-8, 6,-6, 4,-4, 2,-2]
levels = np.tile(l, int(np.ceil(len(dates)/len(l))))[:len(dates)]

fig, ax = plt.subplots(figsize=(20,10))
ax.set(title=topic, yticks=[], yticklabels=[])

ax.vlines(dates, ymin=0, ymax=levels, color="tab:red")
ax.plot(dates, np.zeros_like(dates), "-o", color="k", markerfacecolor="w")

for d,l,r in zip(dates,levels,names):
    ax.annotate(r, xy=(d,l), xytext=(-3, np.sign(l)*3),
                textcoords="offset points",
                horizontalalignment="center",
                verticalalignment="bottom" if l>0 else "top")

plt.xticks(rotation=90)
plt.show()
```

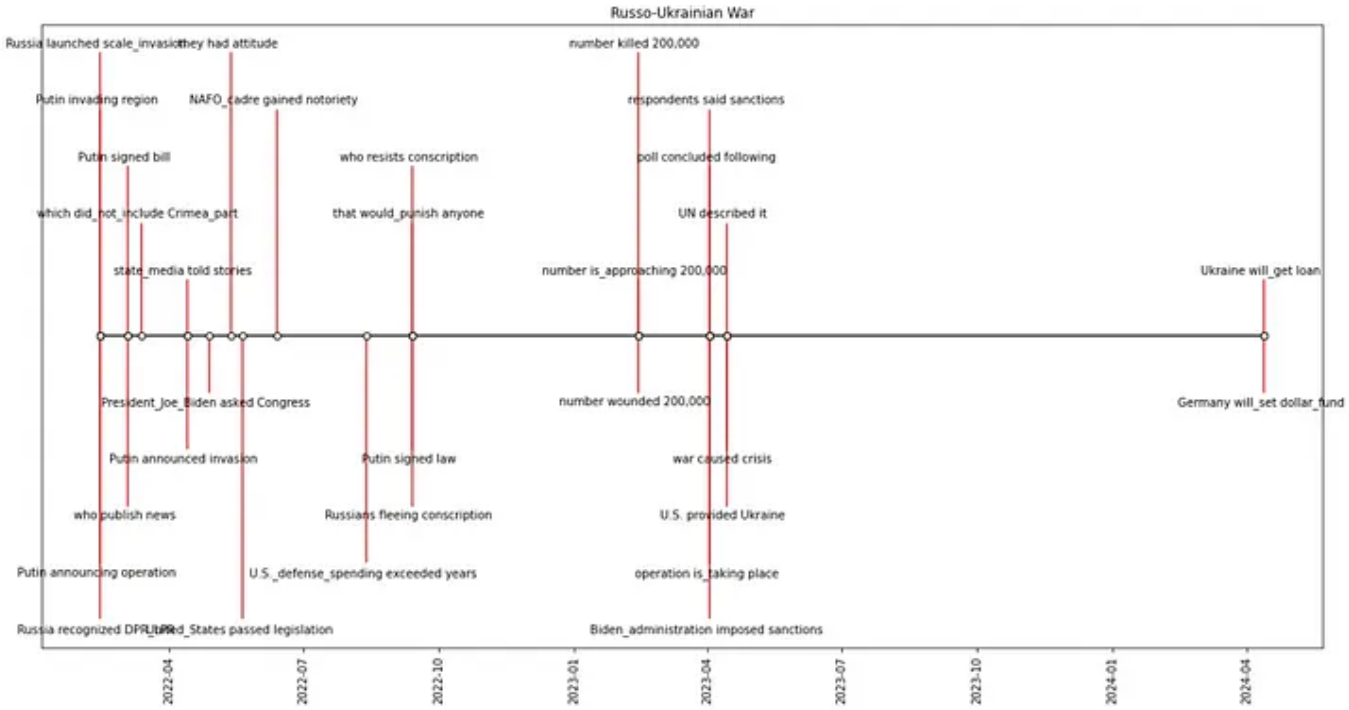


Image by author

As you can see, once the “knowledge” has been extracted, you can plot it any way you like.

Conclusion

This article has been a tutorial about **how to build a Knowledge Graph with Python.** I used several NLP techniques on data parsed from Wikipedia to extract “knowledge” (i.e. entities and relations) and stored it in a Network Graph object.

Now you understand why companies are leveraging NLP and Knowledge Graphs to map relevant data from multiple sources and find insights useful for the business. Just imagine how much value can be extracted by applying this kind of models on all documents (i.e. financial reports, news, tweets) related to a single entity (i.e. Apple Inc). You could quickly understand all the facts, people, and companies directly connected to that entity. And then, by extending the network, even the information not directly connected to starting entity (A — > B — > C).

I hope you enjoyed it! Feel free to contact me for questions and feedback or just to share your interesting projects.

👉 Let's Connect 👈

> *This article is part of the series* ***NLP with Python****, see also:*

Text Summarization with NLP: TextRank vs Seq2Seq vs BART

Natural Language Processing with Python, Gensim, Tensorflow, Transformers

towardsdatascience.com

Text Classification with NLP: Tf-Idf vs Word2Vec vs BERT

Preprocessing, Model Design, Evaluation, Explainability for Bag-of-Words, Word Embedding, Language models

towardsdatascience.com

Text Analysis & Feature Engineering with NLP

Language Detection, Text Cleaning, Length, Sentiment, Named-Entity Recognition, N-grams Frequency, Word Vectors, Topic…

towardsdatascience.com

BERT for Text Classification with NO model training

Use BERT, Word Embedding, and Vector Similarity when you don't have a labeled training set

towardsdatascience.com

AI Chatbot with NLP: Speech Recognition + Transformers

Build a talking ChatBot with Python and have a conversation with your AI

towardsdatascience.com

NLP Python Artificial Intelligence Machine Learning Deep Learning

Written by Mauro Di Pietro

Following

3.4K Followers · Writer for Towards Data Science

Italian, Data Scientist, Financial Analyst, Good Reader, Bad Writer

More from Mauro Di Pietro and Towards Data Science

Mauro Di Pietro in Towards Data Science

Document Parsing with Python & OCR

Detect and extract text, figures, tables from any type of document with Computer Vision

✦ · 7 min read · Jul 13, 2022

152 4

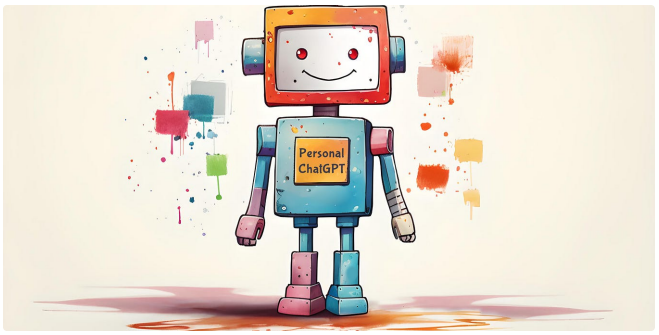
Antonis Makropoulos in Towards Data Science

How to Build a Multi-GPU System for Deep Learning in 2023

This story provides a guide on how to build a multi-GPU system for deep learning and…

10 min read · Sep 17

549 11



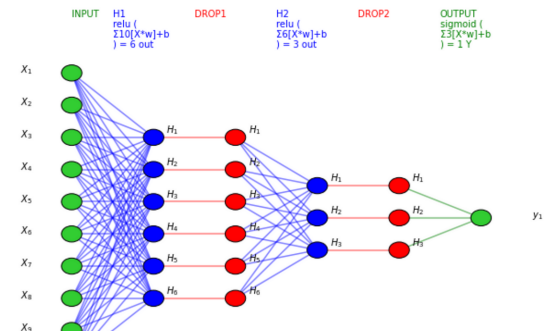
Robert A. Gonsalves in Towards Data Science

Your Own Personal ChatGPT

How you can fine-tune OpenAI's GPT-3.5 Turbo model to perform new tasks using you…

✦ · 15 min read · Sep 8

595 7



Mauro Di Pietro in Towards Data Science

Deep Learning with Python: Neural Networks (complete tutorial)

Build, Plot & Explain Artificial Neural Networks with TensorFlow

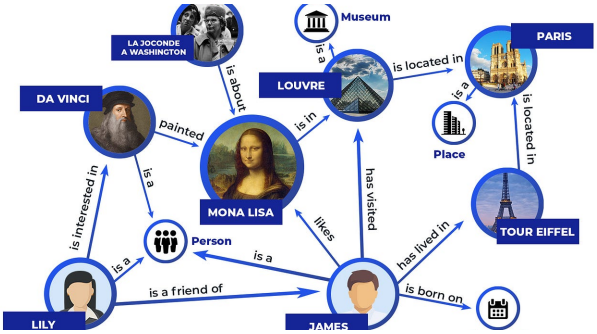
✦ · 14 min read · Dec 17, 2021

682 9

See all from Mauro Di Pietro

See all from Towards Data Science

Recommended from Medium



0xdevshah in AI Skunks

Knowledge Graphs: A Comprehensive Analysis

Knowledge graphs are becoming increasingly important in NLP due to their ability to mode...

8 min read · Apr 9

63

Alla Chepurova in DeepPavlov

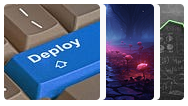
Improving Knowledge Graph Completion with Generative LM...

Combining both internal LM knowledge and external data from KG

13 min read · Sep 5

36

Lists



Predictive Modeling w/ Python

20 stories · 452 saves

Natural Language Processing

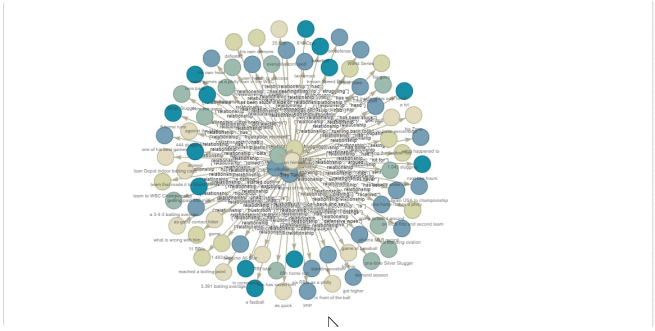
669 stories · 283 saves

Practical Guides to Machine Learning

10 stories · 519 saves

ChatGPT

21 stories · 179 saves



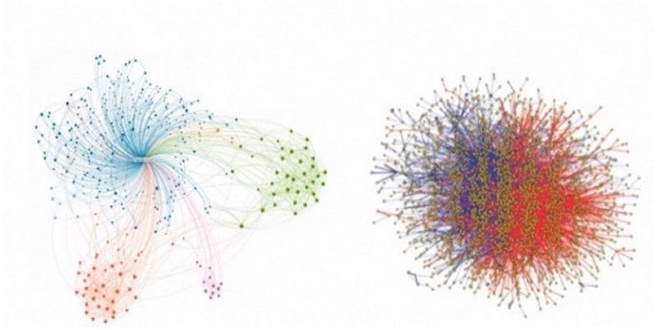
Wenqi Glantz in Better Programming

7 Query Strategies for Navigating Knowledge Graphs With...

Exploring NebulaGraph RAG Pipeline with the Philadelphia Phillies

✦ · 17 min read · 4 days ago

501 4



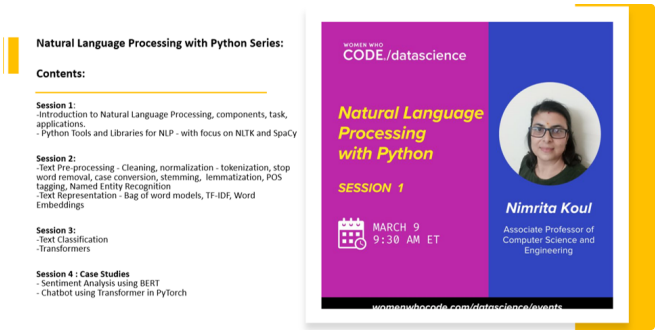
Lei Wang in graphscope

Graphs and Graph Applications

In this post, we will introduce basic concepts of graphs, and some typical applications of...

7 min read · May 22

7

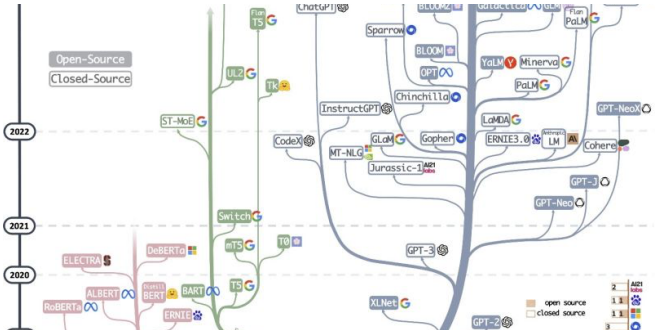


Nimrita Koul

NLP with Python Part 2 NLTK

This is the second article in the series of articles on Natural Language Processing...

5 min read · Apr 5



Haifeng Li

A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14

See more recommendations