

x

-

(<http://play.google.com/store/apps/details?id=com.analyticsvidhya.android>)

End of Decade Sale: Flat 20% OFF on courses | **Use Code: EODS20** - Enroll Today

([https://courses.analyticsvidhya.com/collections/?utm\\_source=flashstrip&utm\\_medium=blog](https://courses.analyticsvidhya.com/collections/?utm_source=flashstrip&utm_medium=blog))



LOGIN / REGISTER ([HTTPS://ID.ANALYTICSVIDHYA.COM/ACCOUNTS/LOGIN/?](https://id.analyticsvidhya.com/accounts/login/)

NEXT=[HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2019/10/HOW-TO-BUILD-KNOWLEDGE-GRAPH-TEXT-USING-SPACY/](https://www.analyticsvidhya.com/blog/2019/10/how-to-build-knowledge-graph-text-using-spacy/))



**Analytics Vidhya**

Learn everything about analytics

([https://www.analyticsvidhya.com/myfeed/?utm-](https://www.analyticsvidhya.com/myfeed/?utm-source=blog&utm-medium=top-icon/)

source=blog&utm-medium=top-icon/)



([https://courses.analyticsvidhya.com/collections?utm\\_source=blog&utm\\_medium=top-right](https://courses.analyticsvidhya.com/collections?utm_source=blog&utm_medium=top-right))

[ENTERTAINMENT \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/ENTERTAINMENT/\)](https://www.analyticsvidhya.com/blog/category/entertainment/)

[GRAPHS & NETWORKS \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/GRAPHS-NETWORKS/\)](https://www.analyticsvidhya.com/blog/category/graphs-networks/)

[INTERMEDIATE \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/INTERMEDIATE/\)](https://www.analyticsvidhya.com/blog/category/intermediate/)

[NLP \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/NLP/\)](https://www.analyticsvidhya.com/blog/category/nlp/)

[PROJECT \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/PROJECT/\)](https://www.analyticsvidhya.com/blog/category/project/)

[PYTHON \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/PYTHON-2/\)](https://www.analyticsvidhya.com/blog/category/python-2/)

[TECHNIQUE \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/TECHNIQUE/\)](https://www.analyticsvidhya.com/blog/category/technique/)

[TEXT \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/TEXT/\)](https://www.analyticsvidhya.com/blog/category/text/)

[UNSTRUCTURED DATA \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/UNSTRUCTURED-DATA/\)](https://www.analyticsvidhya.com/blog/category/unstructured-data/)

## Knowledge Graph – A Powerful Data Science Technique to Mine Information from Text (with Python code)



Bootcamp

**PRATEEK JOSHI (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/AUTHOR/PJOSHI15/), OCTOBER 14, 2019 [LOGIN TO BOOKMARK THIS ART...](#)**

## Overview

- Knowledge graphs are one of the most fascinating concepts in data science
- Learn how to build a knowledge graph using text from Wikipedia pages
- We will be working hands-on in Python to build our knowledge graph using the popular spaCy library

## Introduction

Lionel Messi needs no introduction. Even folks who don't follow football have heard about the brilliance of one of the greatest players to have graced the sport. Here's his Wikipedia page:



# Lionel Messi



From Wikipedia, the free encyclopedia

*"Messi" redirects here. For other uses, see [Messi \(disambiguation\)](#).*

*This article uses [Spanish naming customs](#): the first or paternal family name is Messi and the second or maternal family name is Cuccittini.*

**Lionel Andrés Messi Cuccittini**<sup>[note 1]</sup> (Spanish pronunciation: [ljoˈnel anˈdrez ˈmesi] (listen);<sup>[A]</sup> born 24 June 1987) is an Argentine professional footballer who plays as a forward and captains both Spanish club Barcelona and the Argentine national team. Often considered the best player in the world and widely regarded as one of the greatest players of all time, Messi has won a record six FIFA Ballon d'or/Best FIFA Men's Player awards,<sup>[note 2]</sup> and a record six European Golden Shoes. He has spent his entire professional career with Barcelona, where he has won a club-record 34 trophies, including ten La Liga titles, four UEFA Champions League titles and six Copas del Rey. A prolific goalscorer and a creative playmaker, Messi holds the records for most goals in La Liga (420), a La Liga and European league season (50), most hat-tricks in the UEFA

## Lionel Messi



Messi with Argentina at the 2018 FIFA World

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/messi.png>).

Quite a lot of information there! We have text, tons of [hyperlinks](#), and even an [audio clip](#). That's a lot of relevant and potentially useful information on a single page. The possibilities of putting this into a use case are endless.

However, there is a slight problem. This is not an ideal source of data to feed to our machines. Not in its current form anyway.

Can we find a way to make this text data readable for machines? Essentially, can we transform this text data into something that can be used by the machines and also can be interpreted easily by us?

Yes, we can! We can do it with the help of **Knowledge Graphs (KG)**, one of the most fascinating concepts in [data science](https://courses.analyticsvidhya.com/courses/introduction-to-data-science-2/?utm_source=blog&utm_medium=how-to-build-knowledge-graph-text-using-spacy) ([https://courses.analyticsvidhya.com/courses/introduction-to-data-science-2/?utm\\_source=blog&utm\\_medium=how-to-build-knowledge-graph-text-using-spacy](https://courses.analyticsvidhya.com/courses/introduction-to-data-science-2/?utm_source=blog&utm_medium=how-to-build-knowledge-graph-text-using-spacy)). I have been blown away by the sheer potential and applications of knowledge graphs and I am sure you will as well.

## Bootcamp

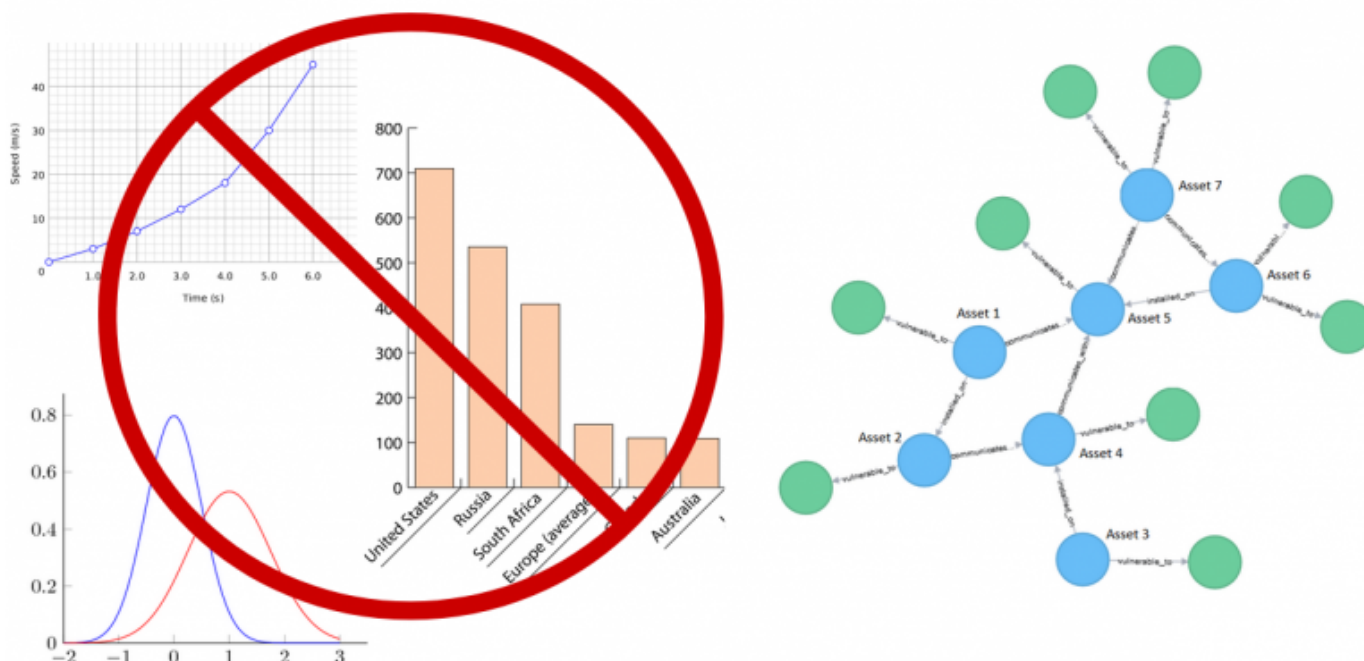
In this article, you will learn what knowledge graphs are, why they're useful, and then we'll dive into code by building our own knowledge graph on data we'll extract from Wikipedia.

## Table of Contents

1. What is a Knowledge Graph?
2. How to Represent Knowledge in a Graph?
  - Sentence Segmentation
  - Entities Extraction
  - Relations Extraction
3. Build a Knowledge Graph from Text Data

## What is a Knowledge Graph?

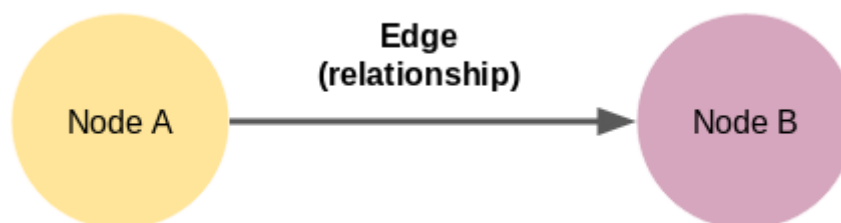
Let's get one thing out of the way – we will see the term “graphs” a lot in this article. We do not mean bar charts, pie charts, and line plots when I say graphs. Here, we are talking about interconnected entities which can be people, locations, organizations, or even an event.



([https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/graph\\_not\\_plots.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/graph_not_plots.png)).  
Bootcamp

*We can define a graph as a set of nodes and edges.*

Take a look at the figure below:



[https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/graph\\_link.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/graph_link.png)

Node A and Node B here are two different entities. These nodes are connected by an edge that represents the relationship between the two nodes. Now, this is the smallest knowledge graph we can build – it is also known as a **triple**.

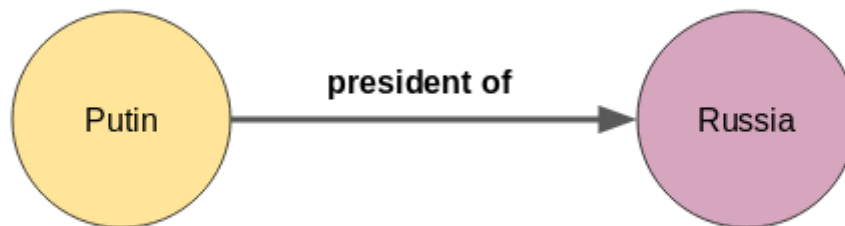
Knowledge Graph's come in a variety of shapes and sizes. For example, the knowledge graph of Wikidata had 59,910,568 nodes by October 2019.

## How to Represent Knowledge in a Graph?

Before we get started with building Knowledge Graphs, it is important to understand how information or knowledge is embedded in these graphs.

Let me explain this using an example. If Node A = Putin and Node B = Russia, then it is quite likely that the edge would be "president of":

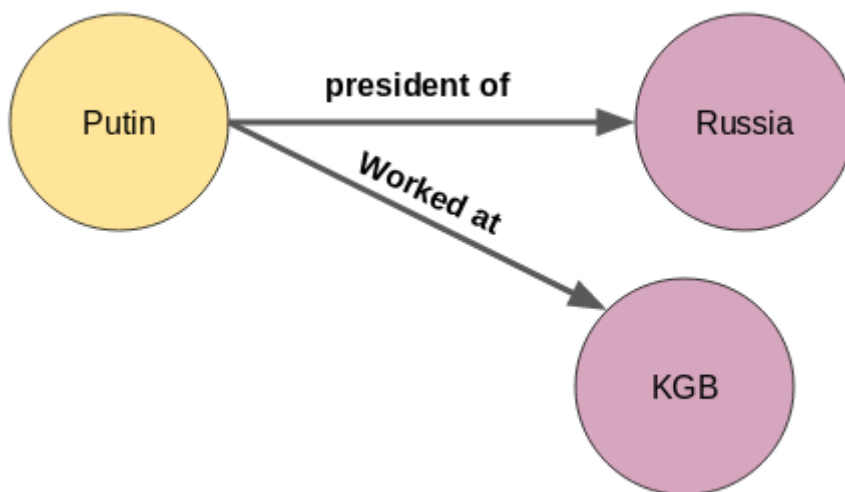




([https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/putin\\_1.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/putin_1.png)).

**A node or an entity can have multiple relations as well.** Putin is not only the President of Russia, he also worked for the Soviet Union's security agency, KGB. But how do we incorporate this new information about Putin in the knowledge graph above?

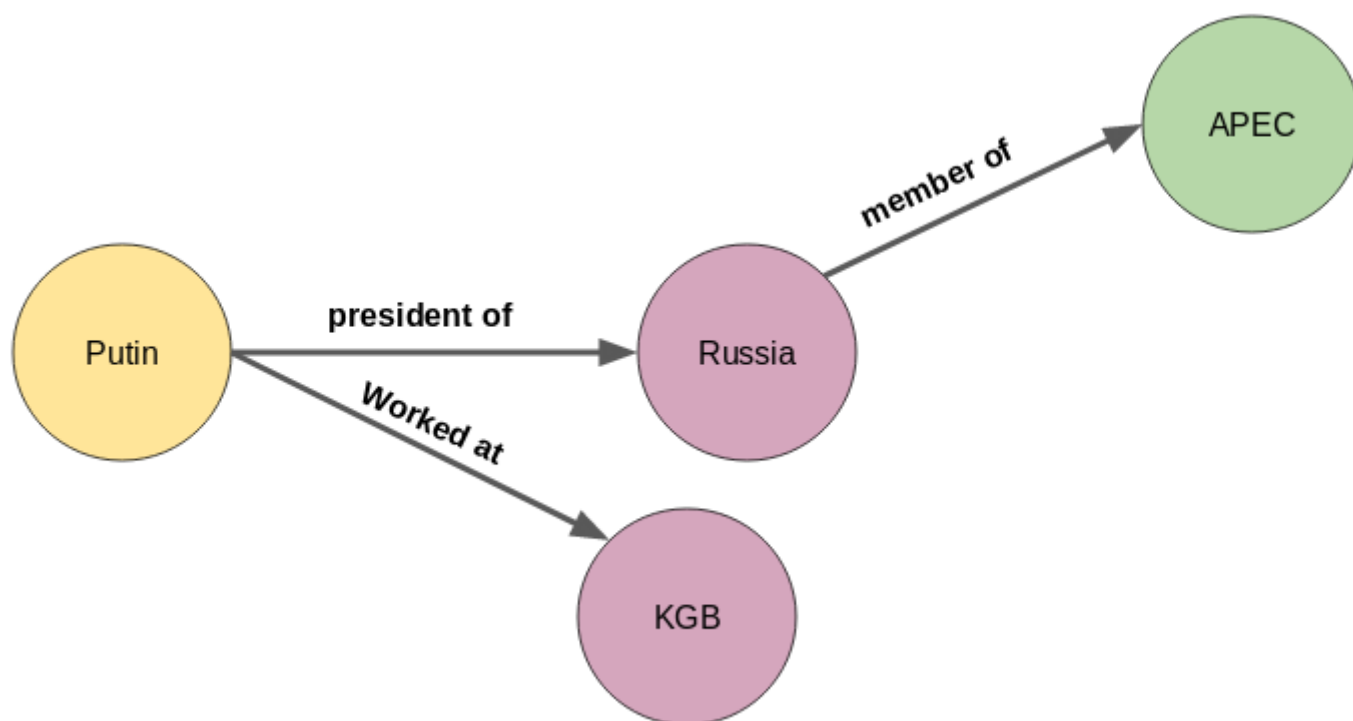
It's actually pretty simple. Just add one more node for the new entity, KGB:



(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/KGB.png>).

The new relationships can emerge not only from the first node but from any node in a knowledge graph as shown below:





(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/APEC.png>).

Russia is a member of the Asia Pacific Economic Cooperation (APEC).

Identifying the entities and the relation between them is not a difficult task for us. However, **manually building a knowledge graph is not scalable**. Nobody is going to go through thousands of documents and extract all the entities and the relations between them!

That's why machines are more suitable to perform this task as going through even hundreds or thousands of documents is child's play for them. But then there is another challenge – machines do not understand natural language. This is where Natural Language Processing (NLP) comes into the picture.

**To build a knowledge graph from the text, it is important to make our machine understand natural language.** This can be done by using NLP techniques such as sentence segmentation, dependency parsing, parts of speech tagging, and entity recognition. Let's discuss these in a bit more detail.

## Sentence Segmentation

**The first step in building a knowledge graph is to split the text document or article into sentences.** Then, we will shortlist only those sentences in which there is exactly 1 subject and 1 object. Let's look at a sample text below👇

Bootcamp

*"Indian tennis player Sumit Nagal moved up six places from 135 to a career-best 129 in the latest men's singles ranking. The 22-year-old recently won the ATP Challenger tournament. He made his Grand Slam debut against Federer in the 2019 US Open. Nagal won the first set."*

Let's split the paragraph above into sentences:

1. *Indian tennis player Sumit Nagal moved up six places from 135 to a career-best 129 in the latest men's singles ranking*
2. *The 22-year-old recently won the ATP Challenger tournament*
3. *He made his Grand Slam debut against Federer in the 2019 US Open*
4. *Nagal won the first set*

Out of these four sentences, we will shortlist the second and the fourth sentences because each of them contains 1 subject and 1 object. In the second sentence, "22-year-old" is the subject and the object is "ATP Challenger tournament". In the fourth sentence, the subject is "Nagal" and "first set" is the object:


Sentence	Subject	Object
The 22-year-old recently won ATP Challenger tournament.	22-year-old	ATP Challenger tournament
Nagal won the first set.	Nagal	first set

(<https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/entities.png>).

The challenge is to make your machine understand the text, especially in the cases of multi-word objects and subjects. For example, extracting the objects in both the sentences above is a bit tricky. Can you think of any method to solve this problem?

## Entities Extraction

The extraction of a single word entity from a sentence is not a tough task. We can easily do this with the help of parts of speech (POS) tags. The nouns and the proper nouns would be our entities.

However, when an entity spans across multiple words, then POS tags alone are not sufficient. We need to parse the dependency tree of the sentence. You can read more about dependency parsing in the following article. 

Bootcamp



- [Introduction to Information Extraction using Python and spaCy](https://www.analyticsvidhya.com/blog/2019/09/introduction-information-extraction-python-spacy/?utm_source=blog&utm_medium=how-to-build-knowledge-graph-text-using-spacy)  
([https://www.analyticsvidhya.com/blog/2019/09/introduction-information-extraction-python-spacy/?utm\\_source=blog&utm\\_medium=how-to-build-knowledge-graph-text-using-spacy](https://www.analyticsvidhya.com/blog/2019/09/introduction-information-extraction-python-spacy/?utm_source=blog&utm_medium=how-to-build-knowledge-graph-text-using-spacy))

Let's get the dependency tags for one of the shortlisted sentences. I will use the popular spaCy library for this task:

```
1 import spacy
2 nlp = spacy.load('en_core_web_sm')
3
4 doc = nlp("The 22-year-old recently won ATP Challenger tournament.")
5
6 for tok in doc:
7     print(tok.text, "...", tok.dep_)
```

[n/prateekjoshi565/6439d3d8996aeb490b9d816fa0226e6c/raw/8c768382cd489545f01b4789bf9e62e98baa1c8f/kg\\_dep\\_parse\\_eg1.py](https://gist.github.com/prateekjoshi565/6439d3d8996aeb490b9d816fa0226e6c/raw/8c768382cd489545f01b4789bf9e62e98baa1c8f/kg_dep_parse_eg1.py)  
kg\_dep\_parse\_eg1.py ([https://gist.github.com/prateekjoshi565/6439d3d8996aeb490b9d816fa0226e6c#file-kg\\_dep\\_parse\\_eg1-py](https://gist.github.com/prateekjoshi565/6439d3d8996aeb490b9d816fa0226e6c#file-kg_dep_parse_eg1-py)) hosted with ❤ by GitHub (<https://github.com>)

### Output:

```
The ... det
22-year ... amod
- ... punct
old ... nsubj
recently ... advmod
won ... ROOT
ATP ... compound
Challenger ... compound
tournament ... dobj
. ... punct
```

The subject (*nsubj*) in this sentence as per the dependency parser is “old”. That is not the desired entity. We wanted to extract “22-year-old” instead.

The dependency tag of “22-year” is *amod* which means it is a modifier of “old”. Hence, we should define a rule to extract such entities.



*The **rule** can be something like this — **extract the subject/object along with its modifiers and also extract the punctuation marks between them.***

But then look at the object (*dobj*) in the sentence. It is just “tournament” instead of “ATP Challenger tournament”. Here, we don’t have the modifiers but compound words.

Compound words are those words that collectively form a new term with a different meaning. Therefore, we can update the above rule to — **extract the subject/object along with its modifiers, compound words and also extract the punctuation marks between them.**

In short, we will use dependency parsing to extract entities.

## Extract Relations

Entity extraction is half the job done. **To build a knowledge graph, we need edges to connect the nodes (entities) to one another.** These edges are the relations between a pair of nodes.

Let’s go back to the example in the last section. We shortlisted a couple of sentences to build a knowledge graph:

Sentence
The 22-year-old recently won ATP Challenger tournament.
Nagal won the first set.

([https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/info\\_extract.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/info_extract.png)).

Can you guess the relation between the subject and the object in these two sentences?

Both sentences have the same relation – “won”. Let’s see how these relations can be extracted. We will again use dependency parsing:

```
1 doc = nlp("Nagal won the first set.")
```

Bootcamp

```

2
3 for tok in doc:
4     print(tok.text, "...", tok.dep_)

```

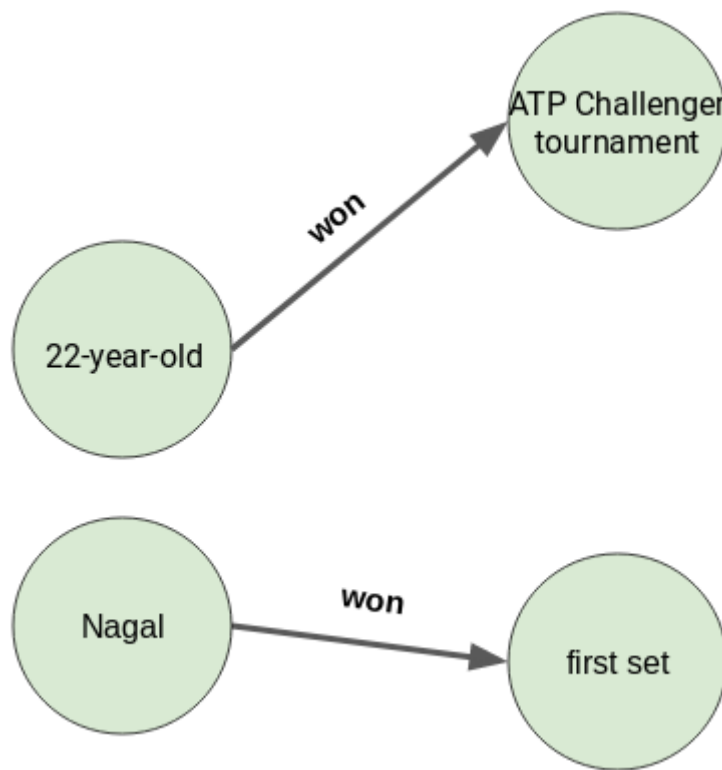
/prateekjoshi565/1fbd666170bd79960811b9a3099909f7/raw/467dadfd498701cc3585d6c9d88ade8ed883975b/kg\_dep\_parse\_eg2.py)  
 kg\_dep\_parse\_eg2.py ([https://gist.github.com/prateekjoshi565/1fbd666170bd79960811b9a3099909f7#file-kg\\_dep\\_parse\\_eg2-py](https://gist.github.com/prateekjoshi565/1fbd666170bd79960811b9a3099909f7#file-kg_dep_parse_eg2-py)) hosted with ❤ by GitHub (<https://github.com>)

## Output:

*Nagal ... nsubj*  
*won ... ROOT*  
*the ... det*  
*first ... amod*  
*set ... dobj*  
*. ... punct*

To extract the relation, we have to find the ROOT of the sentence (which is also the verb of the sentence). Hence, the relation extracted from this sentence would be “won”.

Finally, the knowledge graph from these two sentences will be like this:



([https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/sample\\_KG.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/sample_KG.png))

Bootcamp

## Build a Knowledge Graph from Text Data

Time to get our hands on some code! Let's fire up our Jupyter Notebooks (or whatever IDE you prefer).

We will build a knowledge graph from scratch by using the text from a set of movies and films related to Wikipedia articles. I have already extracted around 4,300 sentences from over 500 Wikipedia articles. Each of these sentences contains exactly two entities – one subject and one object. You can download these sentences from [here \(https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/wiki\\_sentences\\_v2.csv\)](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/wiki_sentences_v2.csv).

I suggest using Google Colab for this implementation to speed up the computation time.

### Import Libraries

```
1  import re
2  import pandas as pd
3  import bs4
4  import requests
5  import spacy
6  from spacy import displacy
7  nlp = spacy.load('en_core_web_sm')
8
9  from spacy.matcher import Matcher
10 from spacy.tokens import Span
11
12 import networkx as nx
13
14 import matplotlib.pyplot as plt
15 from tqdm import tqdm
16
17 pd.set_option('display.max_colwidth', 200)
18 %matplotlib inline
```

/prateekjoshi565/8ad786c305c874b77563515a10bf605a/raw/4b5f47f8b96023c6e48a25abc0b289675598d175/kg\_import\_libraries.py)  
kg\_import\_libraries.py (https://gist.github.com/prateekjoshi565/8ad786c305c874b77563515a10bf605a#file-kg\_import\_libraries-  
py) hosted with ❤ by GitHub (https://github.com)



Bootcamp

## Read Data

Read the CSV file containing the Wikipedia sentences:

```
1 # import wikipedia sentences
2 candidate_sentences = pd.read_csv("wiki_sentences_v2.csv")
3 candidate_sentences.shape
```

com/prateekjoshi565/a809b60656159bc0c106d2e13827d15c/raw/300bb31a8eced9db3aec01d2b11d5582ae374a0e/kg\_read\_data.py)  
kg\_read\_data.py (https://gist.github.com/prateekjoshi565/a809b60656159bc0c106d2e13827d15c#file-kg\_read\_data-py) hosted  
with ❤ by GitHub (https://github.com)

**Output: (4318, 1)**

Let's inspect a few sample sentences:

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

**Output:**

```
3480                                     for a time, it did seem as if mr.
683                                     however, none of these earliest pornographic films are known to have survived.
3313                                     the drawdown process is governed by astm standard d823.
250                                     some pronounced trends have marked horror films.
461    toronto's hot docs founded by filmmaker paul jay, is the leading north american documentary film festival.
Name: sentence, dtype: object
```

([https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/Screenshot\\_1.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/Screenshot_1.png)).

Let's check the subject and object of one of these sentences. Ideally, there should be one subject and one object in the sentence:

```
1 doc = nlp("the drawdown process is governed by astm standard d823")
2
3 for tok in doc:
4     print(tok.text, "...", tok.dep_)
```

com/prateekjoshi565/6b32055e5a96032eee7126216c7238d0/raw/ca17351d40126cafe91e340536421f9b7c077405/kg\_dep\_parse.py)  
kg\_dep\_parse.py (https://gist.github.com/prateekjoshi565/6b32055e5a96032eee7126216c7238d0#file-kg\_dep\_parse-py) hosted  
with ❤ by GitHub (https://github.com)

**Output:**



Bootcamp

```

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

```

([https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/Screenshot\\_2.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/Screenshot_2.png)).

Perfect! There is only one subject ('process') and only one object ('standard'). You can check for other sentences in a similar manner.

## Entity Pairs Extraction

*To build a knowledge graph, the most important things are the nodes and the edges between them.*

These nodes are going to be the entities that are present in the Wikipedia sentences. Edges are the relationships connecting these entities to one another. We will extract these elements in an unsupervised manner, i.e., we will use the grammar of the sentences.

**The main idea is to go through a sentence and extract the subject and the object as and when they are encountered.** However, there are a few challenges — an entity can span across multiple words, eg., “red wine”, and the dependency parsers tag only the individual words as subjects or objects.

So, I have created a function below to extract the subject and the object (entities) from a sentence while also overcoming the challenges mentioned above. I have partitioned the code into multiple chunks for your convenience:

```

1  def get_entities(sent):
2      ## chunk 1
3      ent1 = ""
4      ent2 = ""
5
6      prv_tok_dep = ""    # dependency tag of previous token in the sentence
7      prv_tok_text = ""   # previous token in the sentence

```



Bootcamp

```
8
9     prefix = ""
10    modifier = ""
11
12    #####
13
14    for tok in nlp(sent):
15        ## chunk 2
16        # if token is a punctuation mark then move on to the next token
17        if tok.dep_ != "punct":
18            # check: token is a compound word or not
19            if tok.dep_ == "compound":
20                prefix = tok.text
21                # if the previous word was also a 'compound' then add the current word to it
22                if prv_tok_dep == "compound":
23                    prefix = prv_tok_text + " " + tok.text
24
25            # check: token is a modifier or not
26            if tok.dep_.endswith("mod") == True:
27                modifier = tok.text
28                # if the previous word was also a 'compound' then add the current word to it
29                if prv_tok_dep == "compound":
30                    modifier = prv_tok_text + " " + tok.text
31
32        ## chunk 3
33        if tok.dep_.find("subj") == True:
34            ent1 = modifier + " " + prefix + " " + tok.text
35            prefix = ""
36            modifier = ""
37            prv_tok_dep = ""
38            prv_tok_text = ""
39
40        ## chunk 4
41        if tok.dep_.find("obj") == True:
42            ent2 = modifier + " " + prefix + " " + tok.text
43
44        ## chunk 5
45        # update variables
46        prv_tok_dep = tok.dep_
47        prv_tok_text = tok.text
48    #####
```



```
49
50     return [ent1.strip(), ent2.strip()]
```

om/prateekjoshi565/6833da973d65338216d0f6b99755d120/raw/209c8734023763493514f58d49d77c7b09c57e69/kg\_get\_entities.py)  
kg\_get\_entities.py (https://gist.github.com/prateekjoshi565/6833da973d65338216d0f6b99755d120#file-kg\_get\_entities-py)  
hosted with ❤ by GitHub (https://github.com)

Let me explain the code chunks in the function above:

### Chunk 1

I have defined a few empty variables in this chunk. *prv\_tok\_dep* and *prv\_tok\_text* will hold the dependency tag of the previous word in the sentence and that previous word itself, respectively. *prefix* and *modifier* will hold the text that is associated with the subject or the object.

### Chunk 2

Next, we will loop through the tokens in the sentence. We will first check if the token is a punctuation mark or not. If yes, then we will ignore it and move on to the next token. If the token is a part of a compound word (dependency tag = "compound"), we will keep it in the *prefix* variable. A compound word is a combination of multiple words linked to form a word with a new meaning (example – "Football Stadium", "animal lover").

As and when we come across a subject or an object in the sentence, we will add this *prefix* to it. We will do the same thing with the modifier words, such as "nice shirt", "big house", etc.

### Chunk 3

Here, if the token is the subject, then it will be captured as the first entity in the *ent1* variable. Variables such as *prefix*, *modifier*, *prv\_tok\_dep*, and *prv\_tok\_text* will be reset.

### Chunk 4

Here, if the token is the object, then it will be captured as the second entity in the *ent2* variable. Variables such as *prefix*, *modifier*, *prv\_tok\_dep*, and *prv\_tok\_text* will again be reset.

### Chunk 5

Once we have captured the subject and the object in the sentence, we will update the previous token and its dependency tag.

Let's test this function on a sentence:



Bootcamp



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

**Output:** ['film', '200 patents']

Great, it seems to be working as planned. In the above sentence, 'film' is the subject and '200 patents' is the object.

Now we can use this function to extract these entity pairs for all the sentences in our data:

```
1 entity_pairs = []
2
3 for i in tqdm(candidate_sentences["sentence"]):
4     entity_pairs.append(get_entities(i))
```

[/prateekjoshi565/241aee30cc95aaf54a3533c2ec0f0b40/raw/c91a57e8234501dce75adf2ce54c3e2989b19cdb/kg\\_extract\\_entities.py](https://gist.github.com/prateekjoshi565/241aee30cc95aaf54a3533c2ec0f0b40/raw/c91a57e8234501dce75adf2ce54c3e2989b19cdb/kg_extract_entities.py)  
[kg\\_extract\\_entities.py \(https://gist.github.com/prateekjoshi565/241aee30cc95aaf54a3533c2ec0f0b40#file-kg\\_extract\\_entities-py\)](https://gist.github.com/prateekjoshi565/241aee30cc95aaf54a3533c2ec0f0b40#file-kg_extract_entities-py)  
 hosted with ❤ by GitHub (<https://github.com>)

The list *entity\_pairs* contains all the subject-object pairs from the Wikipedia sentences. Let's have a look at a few of them:

```
entity_pairs[10:20]
```

**Output:**

```
[['we', 'tests'],
 ['global', 'international sales rights'],
 ['robbie robertson', 'soundtrack'],
 ['it', 'original music tracks'],
 ['it', 'reviewed franchise'],
 ['she', 'accidentally mystique'],
 ['military forces', 'arrest'],
 ['train', 'vuk'],
 ['kota eberhardt', 'telepath selene gallio'],
 ['singer', 'sequel']]
```

([https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/Screenshot\\_3.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/Screenshot_3.png))

As you can see, there are a few pronouns in these entity pairs such as 'we', 'it', 'she', etc. We'd like to have proper nouns or nouns instead. Perhaps we can further improve the *get\_entities()* function to filter out pronouns. For the time being, let's leave it as it is and move on to the relation extraction part.

## Relation / Predicate Extraction



Bootcamp

This is going to be a very interesting aspect of this article. **Our hypothesis is that the predicate is actually the main verb in a sentence.**

For example, in the sentence – “Sixty Hollywood musicals were released in 1929”, the verb is “released in” and this is what we are going to use as the predicate for the triple generated from this sentence.

The function below is capable of capturing such predicates from the sentences. Here, I have used spaCy’s rule-based matching:

```
1  def get_relation(sent):
2
3      doc = nlp(sent)
4
5      # Matcher class object
6      matcher = Matcher(nlp.vocab)
7
8      #define the pattern
9      pattern = [{'DEP': 'ROOT'},
10                 {'DEP': 'prep', 'OP': "?"},
11                 {'DEP': 'agent', 'OP': "?"},
12                 {'POS': 'ADJ', 'OP': "?"}]
13
14      matcher.add("matching_1", None, pattern)
15
16      matches = matcher(doc)
17      k = len(matches) - 1
18
19      span = doc[matches[k][1]:matches[k][2]]
20
21      return(span.text)
```

[https://gist.github.com/prateekjoshi565/8751194336f170234a32ca852d729ae2/raw/876ce04e418e8b52ca39335cab275f87ef24fb78/kg\\_get\\_relation.py](https://gist.github.com/prateekjoshi565/8751194336f170234a32ca852d729ae2/raw/876ce04e418e8b52ca39335cab275f87ef24fb78/kg_get_relation.py)  
kg\_get\_relation.py (https://gist.github.com/prateekjoshi565/8751194336f170234a32ca852d729ae2#file-kg\_get\_relation-py)  
hosted with ❤ by GitHub (https://github.com)

The pattern defined in the function tries to find the ROOT word or the main verb in the sentence. Once the ROOT is identified, then the pattern checks whether it is followed by a preposition (‘prep’) or an agent word. If yes, then it is added to the ROOT word.

Let me show you a glimpse of this function:



```
get_relation("John completed the task")
```

**Output:** completed

Similarly, let's get the relations from all the Wikipedia sentences:

```
relations = [get_relation(i) for i in tqdm(candidate_sentences['sentence'])]
```

Let's take a look at the most frequent relations or predicates that we have just extracted:

```
pd.Series(relations).value_counts()[:50]
```

**Output:**

is	401
was	324
are	99
released on	92
were	91
include	80
's	45
released	41
became	40
composed by	35
have	32
become	32
has	31
included	28
released in	27
been	26
received	25
had	25
considered	24
be	24
made	22
produced	21
used	19
called	17
written by	16

([https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/Screenshot\\_4.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/Screenshot_4.png)).

It turns out that relations like “A **is** B” and “A **was** B” are the most common relations. However, there are quite a few relations that are more associated with the overall theme – “the ecosystem around movies”. Some of the examples are “composed by”, “released in”, “produced”, “written by” and a few more.

Bootcamp

## Build a Knowledge Graph

We will finally create a knowledge graph from the extracted entities (subject-object pairs) and the predicates (relation between entities).

Let's create a dataframe of entities and predicates:

```
1 # extract subject
2 source = [i[0] for i in entity_pairs]
3
4 # extract object
5 target = [i[1] for i in entity_pairs]
6
7 kg_df = pd.DataFrame({'source':source, 'target':target, 'edge':relations})
```

[n/prateekjoshi565/e53be0c2500f5e0338c32fc997aaa970/raw/9ee7a183d4b3b90b52527e6ccba82b8268cac9d2/kg\\_extract\\_nodes.py](https://gist.github.com/prateekjoshi565/e53be0c2500f5e0338c32fc997aaa970/raw/9ee7a183d4b3b90b52527e6ccba82b8268cac9d2/kg_extract_nodes.py)  
[kg\\_extract\\_nodes.py \(https://gist.github.com/prateekjoshi565/e53be0c2500f5e0338c32fc997aaa970#file-kg\\_extract\\_nodes-py\)](https://gist.github.com/prateekjoshi565/e53be0c2500f5e0338c32fc997aaa970#file-kg_extract_nodes-py)  
 hosted with ❤ by GitHub (<https://github.com>)

Next, we will use the *networkx* library to create a network from this dataframe. The nodes will represent the entities and the edges or connections between the nodes will represent the relations between the nodes.

**It is going to be a directed graph.** In other words, the relation between any connected node pair is not two-way, it is only from one node to another. For example, “John **eats** pasta”:

```
1 # create a directed-graph from a dataframe
2 G=nx.from_pandas_edgelist(kg_df, "source", "target",
3                             edge_attr=True, create_using=nx.MultiDiGraph())
```

[n/prateekjoshi565/177e18602c7e03f96c573a746e8a2806/raw/2c47b391bb9aade70f1f1a8b875f5053d6a6b8b5/kg\\_create\\_graph\\_1.py](https://gist.github.com/prateekjoshi565/177e18602c7e03f96c573a746e8a2806/raw/2c47b391bb9aade70f1f1a8b875f5053d6a6b8b5/kg_create_graph_1.py)  
[kg\\_create\\_graph\\_1.py \(https://gist.github.com/prateekjoshi565/177e18602c7e03f96c573a746e8a2806#file-kg\\_create\\_graph\\_1-py\)](https://gist.github.com/prateekjoshi565/177e18602c7e03f96c573a746e8a2806#file-kg_create_graph_1-py)  
 hosted with ❤ by GitHub (<https://github.com>)

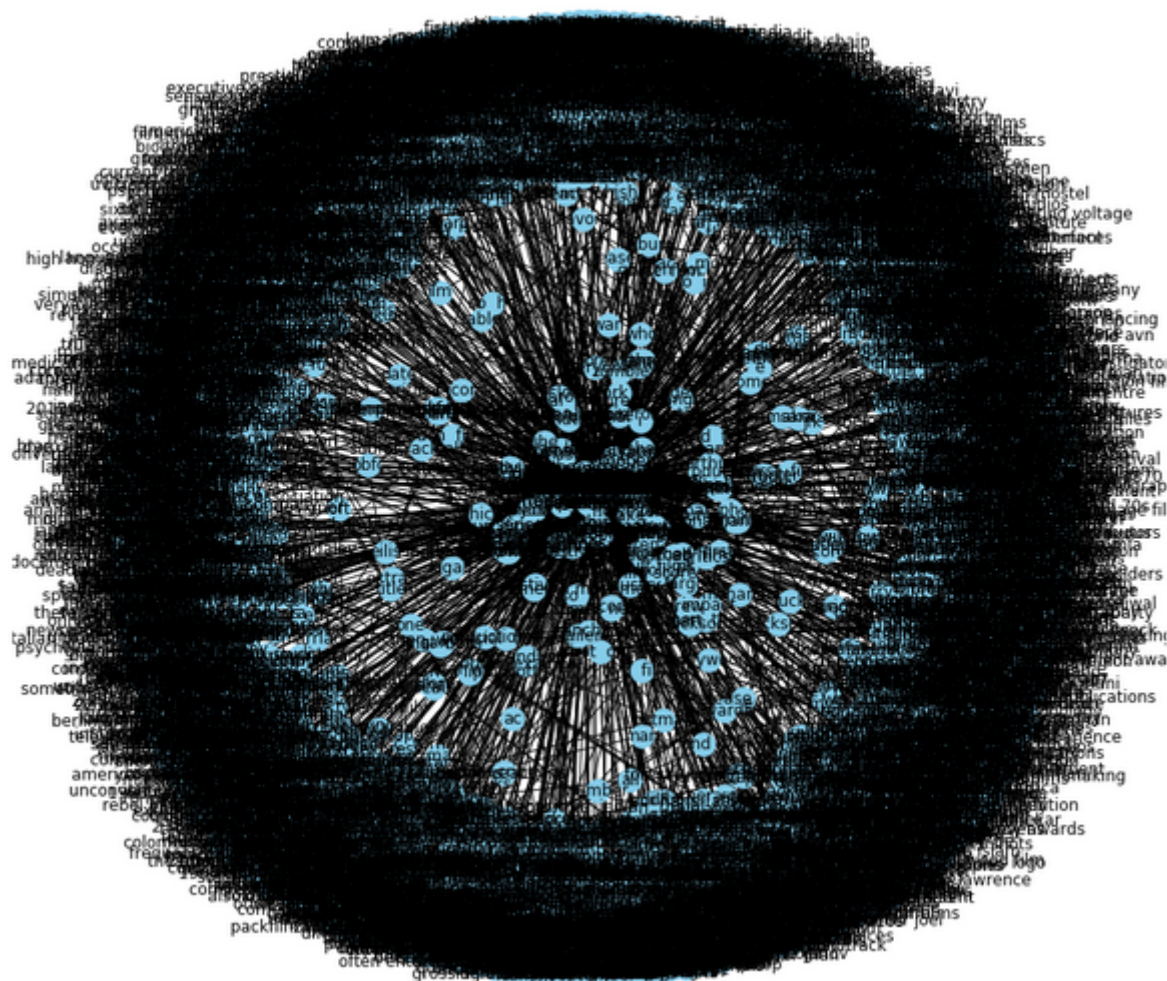
Let's plot the network:

```
1 plt.figure(figsize=(12,12))
2
3 pos = nx.spring_layout(G)
4 nx.draw(G, with_labels=True, node_color='skyblue', edge_cmap=plt.cm.Blues, pos = pos)
5 plt.show()
```

[kg\\_create\\_graph\\_2.py \(https://gist.github.com/prateekjoshi565/88c1afa052f1c34f5954bb781ad74aee#file-kg\\_create\\_graph\\_2-py\)](https://gist.github.com/prateekjoshi565/88c1afa052f1c34f5954bb781ad74aee#file-kg_create_graph_2-py)  
 Bootcamp

om/prateekjoshi565/88c1afa052f1c34f5954bb781ad74aee/raw/15bd7c36383860371f76b8a1f1681a7c96acf20f/kg\_create\_graph\_2.py)  
 hosted with ❤ by GitHub (<https://github.com>)

## Output:



([https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/all\\_relations\\_graph.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/all_relations_graph.png)).

Well, this is not exactly what we were hoping for (still looks quite a sight though!).

It turns out that we have created a graph with all the relations that we had. It becomes really hard to visualize a graph with these many relations or predicates.

So, it's advisable to use only a few important relations to visualize a graph. I will take one relation at a time. Let's start with the relation "composed by":

```
1 G=nx.from_pandas_edgelist(kg_df[kg_df['edge']=="composed by"], "source", "target",
```

Bootcamp

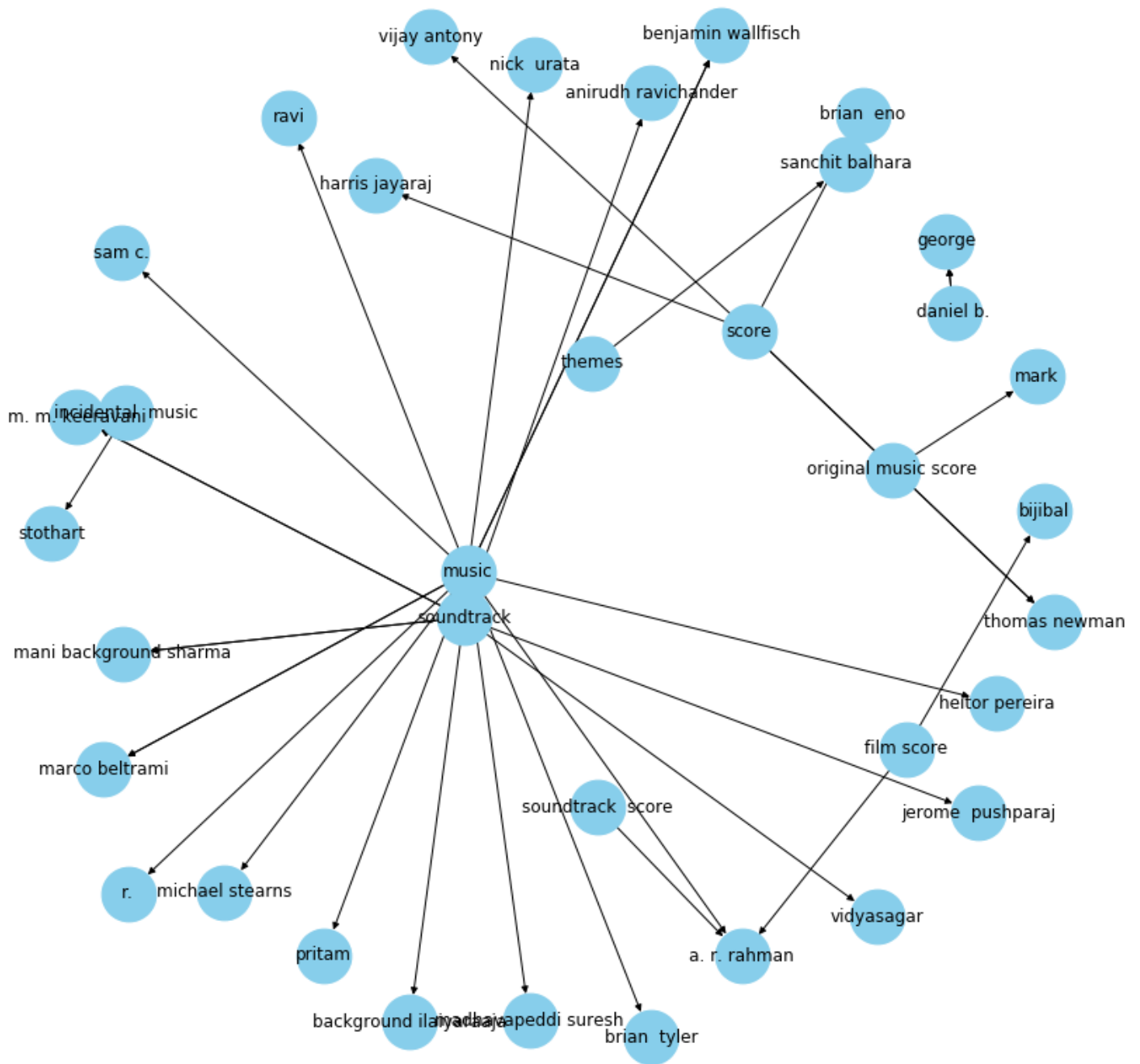
```
2             edge_attr=True, create_using=nx.MultiDiGraph())
3
4 plt.figure(figsize=(12,12))
5 pos = nx.spring_layout(G, k = 0.5) # k regulates the distance between nodes
6 nx.draw(G, with_labels=True, node_color='skyblue', node_size=1500, edge_cmap=plt.cm.Blues, pos = pos)
7 plt.show()
```

[/prateekjoshi565/14ab0085b276333be12a4463c42ddb5e/raw/416d1caa74fae9fd799177932fe659a674b44f1b/kg\\_create\\_graph\\_3.py](https://gist.github.com/prateekjoshi565/14ab0085b276333be12a4463c42ddb5e/raw/416d1caa74fae9fd799177932fe659a674b44f1b/kg_create_graph_3.py))  
kg\_create\_graph\_3.py ([https://gist.github.com/prateekjoshi565/14ab0085b276333be12a4463c42ddb5e#file-kg\\_create\\_graph\\_3-py](https://gist.github.com/prateekjoshi565/14ab0085b276333be12a4463c42ddb5e#file-kg_create_graph_3-py)) hosted with ❤ by GitHub (<https://github.com>)

**Output:**



Bootcamp



([https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/composed\\_by.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/composed_by.png))

That's a much cleaner graph. Here the arrows point towards the composers. For instance, A.R. Rahman, who is a renowned music composer, has entities like "soundtrack score", "film score", and "music" connected to him in the graph above. ^

Bootcamp

Let's check out a few more relations.

Since writing is an important role in any movie, I would like to visualize the graph for the "written by" relation:

```
1 G=nx.from_pandas_edgelist(kg_df[kg_df['edge']=="written by"], "source", "target",
2                             edge_attr=True, create_using=nx.MultiDiGraph())
3
4 plt.figure(figsize=(12,12))
5 pos = nx.spring_layout(G, k = 0.5)
6 nx.draw(G, with_labels=True, node_color='skyblue', node_size=1500, edge_cmap=plt.cm.Blues, pos = pos)
7 plt.show()
```

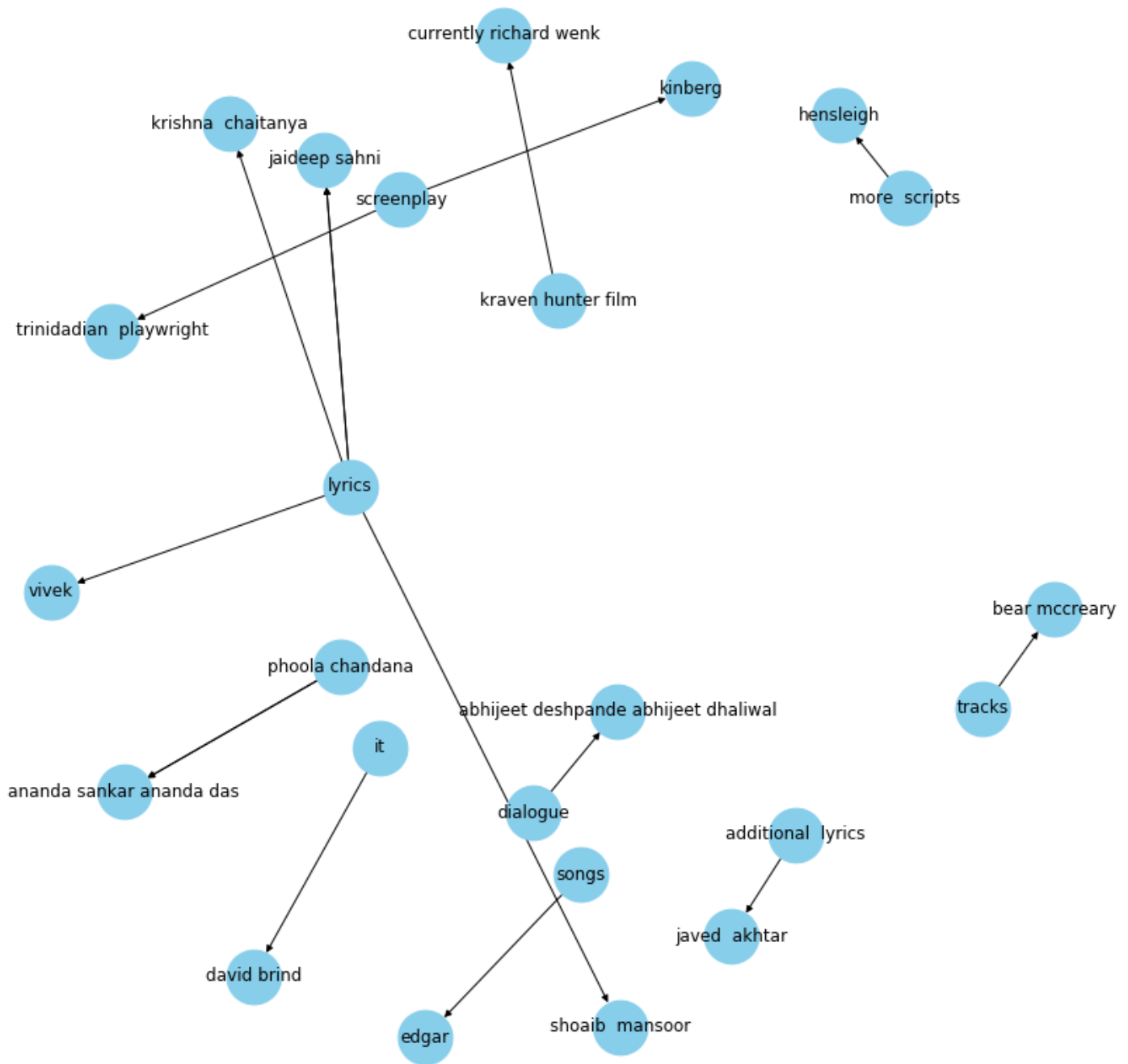
[/prateekjoshi565/a56a27328122eb8c0727999ace512213/raw/406d347d3a5ebc09bb8cb689ff75e2c3406379fe/kg\\_create\\_graph\\_4.py](https://gist.github.com/prateekjoshi565/a56a27328122eb8c0727999ace512213/raw/406d347d3a5ebc09bb8cb689ff75e2c3406379fe/kg_create_graph_4.py)  
[kg\\_create\\_graph\\_4.py \(https://gist.github.com/prateekjoshi565/a56a27328122eb8c0727999ace512213#file-kg\\_create\\_graph\\_4-py\)](https://gist.github.com/prateekjoshi565/a56a27328122eb8c0727999ace512213#file-kg_create_graph_4-py) hosted with ❤ by [GitHub \(https://github.com\)](https://github.com)

**Output:**



Bootcamp





([https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/written\\_by.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/written_by.png)).

Awesome! This knowledge graph is giving us some extraordinary information. Guys like Javed Akhtar, Krishna Chaitanya, and Jaideep Sahni are all famous lyricists and this graph beautifully captures this relationship.

Let's see the knowledge graph of another important predicate, i.e., the "released in":

Bootcamp

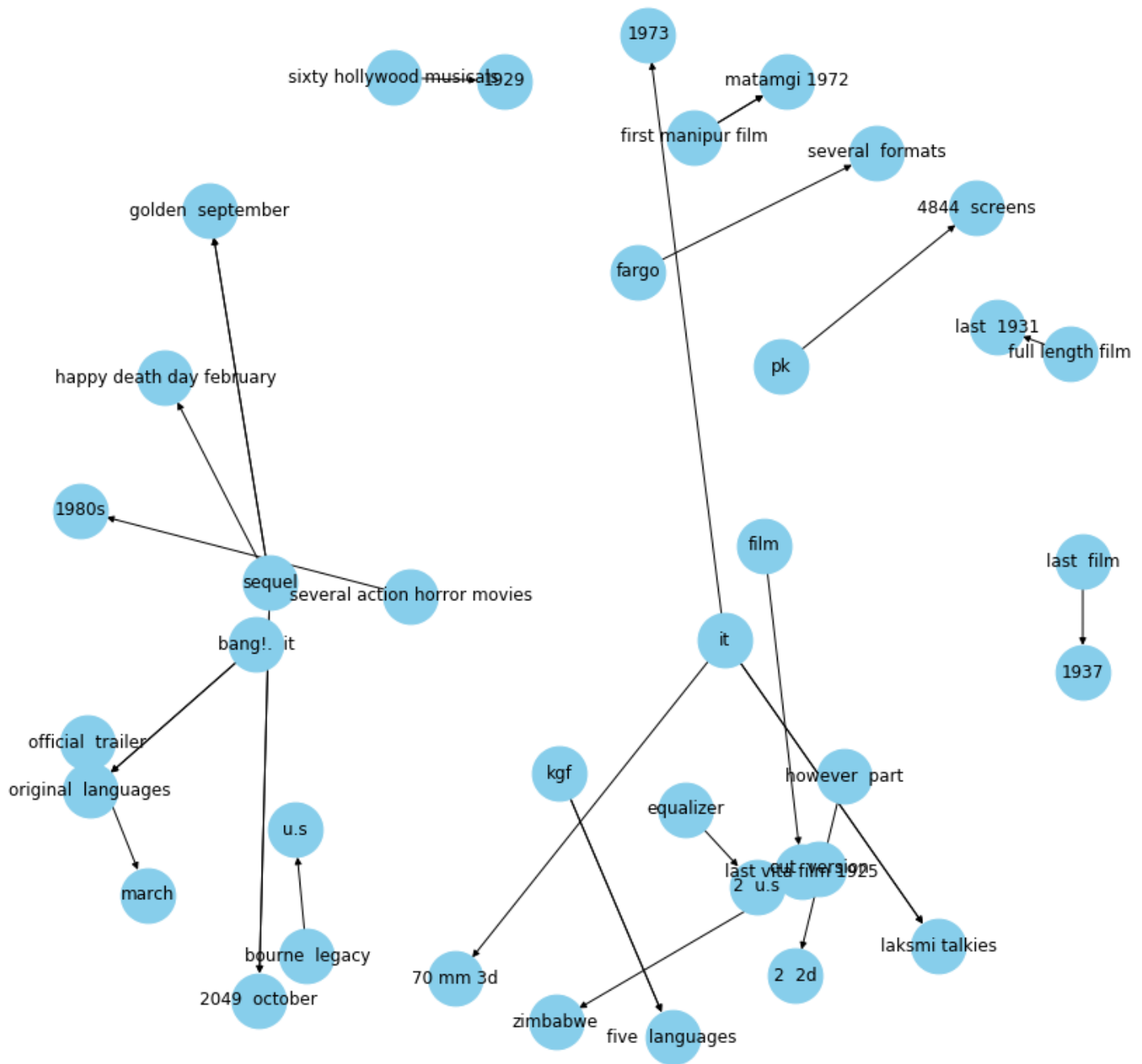
```
1 G=nx.from_pandas_edgelist(kg_df[kg_df['edge']=="released in"], "source", "target",
2                             edge_attr=True, create_using=nx.MultiDiGraph())
3
4 plt.figure(figsize=(12,12))
5 pos = nx.spring_layout(G, k = 0.5)
6 nx.draw(G, with_labels=True, node_color='skyblue', node_size=1500, edge_cmap=plt.cm.Blues, pos = pos)
7 plt.show()
```

n/prateekjoshi565/09e330170de52e1a38fd5a9680eaa222/raw/3fdd605b5e93c1442f7dbfae29b4b2159f9db216/kg\_create\_graph\_5.py)  
kg\_create\_graph\_5.py ([https://gist.github.com/prateekjoshi565/09e330170de52e1a38fd5a9680eaa222#file-kg\\_create\\_graph\\_5-py](https://gist.github.com/prateekjoshi565/09e330170de52e1a38fd5a9680eaa222#file-kg_create_graph_5-py)) hosted with ❤ by GitHub (<https://github.com>)

**Output:**



Bootcamp



([https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/released\\_in.png](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/10/released_in.png)).

I can see quite a few interesting information in this graph. For example, look at this relationship – “several action horror movies **released in** the 1980s” and “pk **released on** 4844 screens”. These are facts and it shows us that we can mine such facts from just text. That’s quite amazing!

Bootcamp

## End Notes

In this article, we learned how to extract information from a given text in the form of triples and build a knowledge graph from it.

However, we restricted ourselves to use sentences with exactly 2 entities. Even then we were able to build quite informative knowledge graphs. Imagine the potential we have here!

I encourage you to explore this field of information extraction more to learn extraction of more complex relationships. In case you have any doubt or you want to share your thoughts, please feel free to use the comments section below.

You can also read this article on Analytics Vidhya's Android APP



[\(\(//play.google.com/store/apps/details?id=com.analyticsvidhya.android&utm\\_source=blog\\_article&utm\\_campaign=blog&pcampaignid=MKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1\)\)](https://play.google.com/store/apps/details?id=com.analyticsvidhya.android&utm_source=blog_article&utm_campaign=blog&pcampaignid=MKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1)

### Share this:

 (<https://www.analyticsvidhya.com/blog/2019/10/how-to-build-knowledge-graph-text-using-spacy/?share=linkedin&nb=1>)

 (<https://www.analyticsvidhya.com/blog/2019/10/how-to-build-knowledge-graph-text-using-spacy/?share=facebook&nb=1>)

 (<https://www.analyticsvidhya.com/blog/2019/10/how-to-build-knowledge-graph-text-using-spacy/?share=twitter&nb=1>)

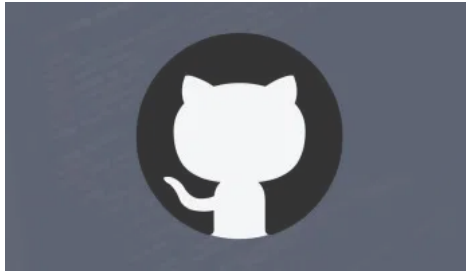
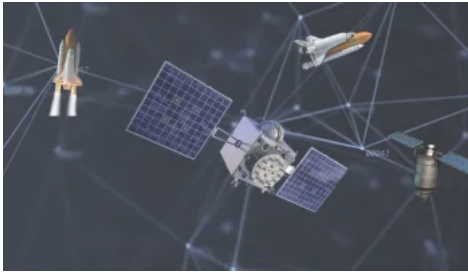
 (<https://www.analyticsvidhya.com/blog/2019/10/how-to-build-knowledge-graph-text-using-spacy/?share=pocket&nb=1>)

 (<https://www.analyticsvidhya.com/blog/2019/10/how-to-build-knowledge-graph-text-using-spacy/?share=reddit&nb=1>)

## Related Articles



Bootcamp



(<https://www.analyticsvidhya.com/blog/2019/11/graph-feature-extraction-deepwalk/>).

Learn How to Perform Feature Extraction from Graphs using DeepWalk

(<https://www.analyticsvidhya.com/blog/2019/11/graph-feature-extraction-deepwalk/>)

November 6, 2019

In "Algorithm"

[open-source-data-science-projects/](https://www.analyticsvidhya.com/blog/2019/11/6-open-source-data-science-projects/)).

6 Exciting Open Source Data Science Projects you Should Start Working on Today

(<https://www.analyticsvidhya.com/blog/2019/11/6-open-source-data-science-projects/>)

November 4, 2019

In "Computer Vision"

[information-extraction-python-spacy/](https://www.analyticsvidhya.com/blog/2019/09/introduction-information-extraction-python-spacy/)).

How Search Engines like Google Retrieve Results: Introduction to Information Extraction using Python and spaCy

(<https://www.analyticsvidhya.com/blog/2019/09/introduction-information-extraction-python-spacy/>)

September 23, 2019

In "Beginner"

**TAGS :** [DEPENDENCY TREES \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/DEPENDENCY-TREES/\)](https://www.analyticsvidhya.com/blog/tag/dependency-trees/), [GRAPHS \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/GRAPHS/\)](https://www.analyticsvidhya.com/blog/tag/graphs/), [NATURAL LANGUAGE PROCESSING \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/NATURAL-LANGUAGE-PROCESSING/\)](https://www.analyticsvidhya.com/blog/tag/natural-language-processing/), [NATURAL LANGUAGE UNDERSTANDING \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/NATURAL-LANGUAGE-UNDERSTANDING/\)](https://www.analyticsvidhya.com/blog/tag/natural-language-understanding/), [NLP \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/NLP/\)](https://www.analyticsvidhya.com/blog/tag/nlp/), [PART OF SPEECH \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/PART-OF-SPEECH/\)](https://www.analyticsvidhya.com/blog/tag/part-of-speech/), [PYTHON \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/PYTHON/\)](https://www.analyticsvidhya.com/blog/tag/python/), [SPACY \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/SPACY/\)](https://www.analyticsvidhya.com/blog/tag/spacy/), [TEXT DATA \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/TEXT-DATA/\)](https://www.analyticsvidhya.com/blog/tag/text-data/), [UNSTRUCTURED DATA \(HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/UNSTRUCTURED-DATA/\)](https://www.analyticsvidhya.com/blog/tag/unstructured-data/)

