DATA SCIENCE PROJECT

REPORT

A KNOWLEDGE BASED RECOMMENDATION SYSTEM
FOR PROJECT RISK MANAGEMENT:ONTOLGOY LEARNING APPROACH

Realized by :

LAKHAL Abir
HMAIDI Mehdi
BELAIDI Nada
MAGHREBI Racha
CHARAABI Khaled
KHECHANA Fares

University Year : 2021 - 2022

Project description

Project Risk Management (PRM), a knowledge-intensive process with processes and tools, requires effective management of risk-related knowledge that can assist in making the right decision by managers.

The premise of this work is to address the problem of OL that could be applied to text analysis as a way to build PRM ontology which will be integrated further in an recommendation system that predict and provide a real time personnalized recommendation with respect to user profile (project manager, risk owner, risk action owner, etc) a targeted answer for a risk-related request. This ontology

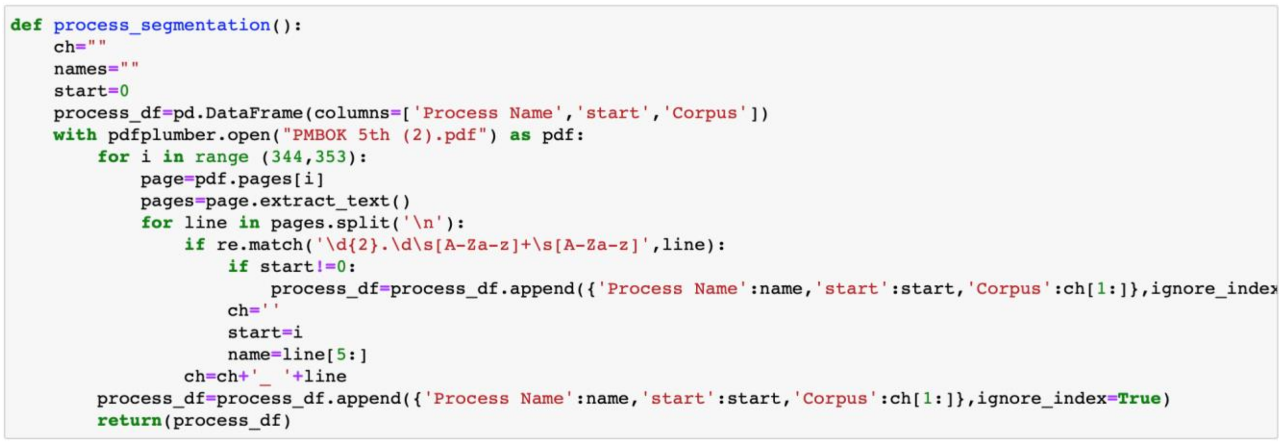
will be learnt from an unstructured text which called " PMBOK + PMI's standard for PRM" as a reference guideline that embodies a set of knowledge and recommendations respectively.

A generic framework, that is reusable and can be shared, serve as a guide to manage PRs according to PMBOK 5th An ontology based recommendation system for PRM supporting the retrieval, representation and recommendation of knowledge!

Chapter1: Data extraction and analysis

1.Text segmentation:

We start by segmenting our text data. It is a precursor to text retrieval, automatic summarization, information retrieval ; language modeling and natural language processing . In written texts, it is the process of identifying the boundaries between words, phrases, or some other linguistic meaningful units, such as sentences or topics. The term separated from such processing is useful to help humans reading texts, and are mainly used to assist computers to do some artificial processes as fundamental units, such as NLP, and IR. We extract the PDF file by process.



```
def process_segmentation():
    ch=""
    names=""
    start=0
    process_df=pd.DataFrame(columns=['Process Name','start','Corpus'])
    with pdfplumber.open("PMBOK 5th (2).pdf") as pdf:
        for i in range (344,353):
            page=pdf.pages[i]
            pages=page.extract_text()
            for line in pages.split('\n'):
                if re.match('\d{2}.\d\s[A-Za-z]+\s[A-Za-z]',line):
                    if start!=0:
                        process_df=process_df.append({'Process Name':name,'start':start,'Corpus':ch[1:]},ignore_index
                    ch=''
                    start=i
                    name=line[5:]
                ch=ch+'_ '+line
        process_df=process_df.append({'Process Name':name,'start':start,'Corpus':ch[1:]},ignore_index=True)
        return(process_df)
```

Then we extract the output, input and the techniques and tools each one in a separate dataframe with their annotation .

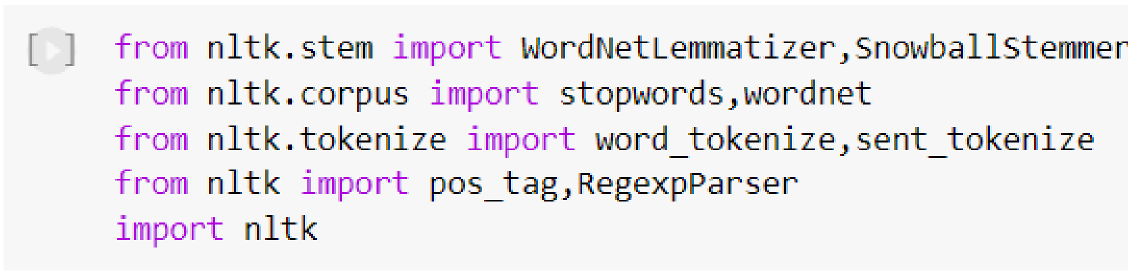
2.Text-preprocessing:

Since we have textual PDF data , we are in need of specific processing to be able to extract meanings and learn from this data in order to make it more useful. Thus, we proceeded by using the Natural Language Processing which is known as NLP. The techniques of the 11 preparation of NLP data covers several stages, the result of which is cleaned and formatted text data.

NLP stands for Natural Language Processing, which is a part of Computer Science, Human language, and Artificial Intelligence. It is the technology that is used by machines to understand, analyse, manipulate, and interpret human's languages. It helps developers to organize knowledge for performing tasks such as translation, automatic summarization, Named Entity Recognition (NER), speech recognition, relationship extraction, and topic segmentation

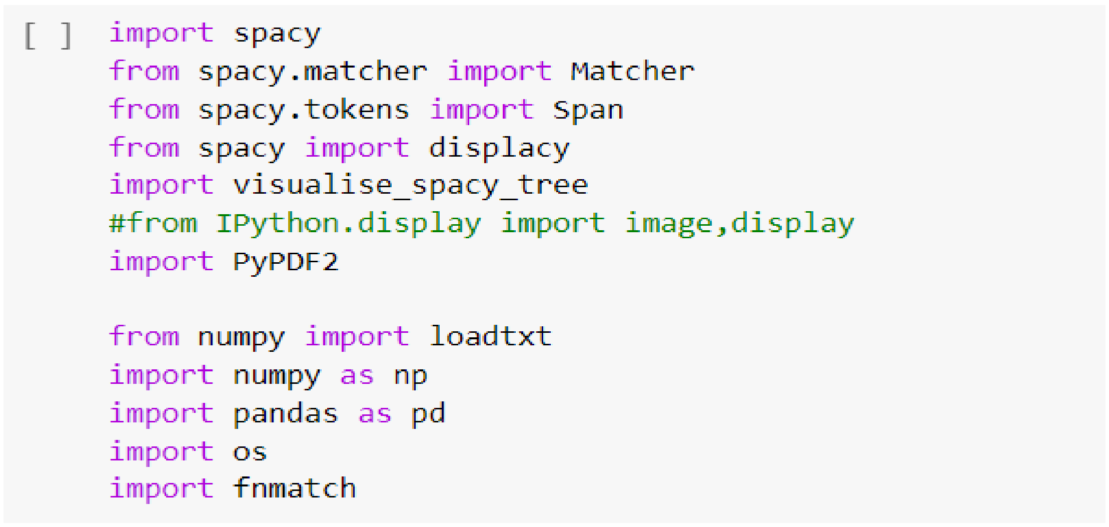
1.1 Libraries importing :

We imported the **nltk** library, which is a software library in Python allowing automatic language processing, so we can use WordNetLemmatizer, StopWords, word_tokenize, pos_tag..



```
[ ]  from nltk.stem import WordNetLemmatizer,SnowballStemmer
     from nltk.corpus import stopwords,wordnet
     from nltk.tokenize import word_tokenize,sent_tokenize
     from nltk import pos_tag,RegexpParser
     import nltk
```

We also imported spacy, which is a Python software library for automatic language processing, to use the Matcher, os, fnmatch..



```
[ ]  import spacy
     from spacy.matcher import Matcher
     from spacy.tokens import Span
     from spacy import displacy
     import visualise_spacy_tree
     #from IPython.display import image,display
     import PyPDF2

     from numpy import loadtxt
     import numpy as np
     import pandas as pd
     import os
     import fnmatch
```

We downloaded the medium English pipeline optimized for CPU from spacy.

```
[ ]  pip install spacy download en_core_web_md
```

1.2 Regex:

We used the regular expression function to extract only the alphabetic expressions from the text (Remove all the other characters).

```
import regex as re

def typo4(text):
    new=text.replace("."," ")
    return new
def Split(text):
    x=re.findall('.[^A-Z]*', text)
    x = ' '.join(x)
    return x
```

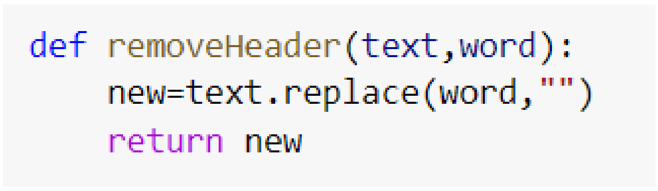
1.2 Stopwords :

Many words have little semantic interest, this is the reason for which the stop words must be deleted. Indeed, stopwords are all the most common words in a language. NLTK library has a list of stopwords for many languages and also tried removing stopwords using their POS in the sentence.

```
def stopwords(s):
    doc=nlp(s)
    tokens = [token.text for token in doc]
    filtered = [token.text for token in doc if token.is_stop == False]
    return filtered
```

1.4 Remove Header:

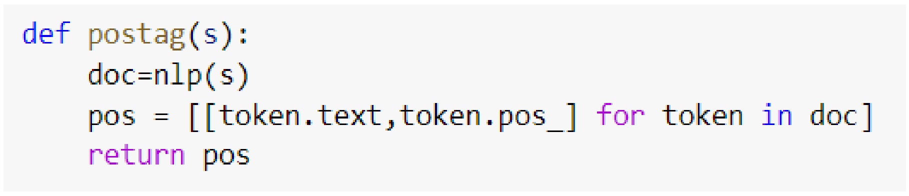
This function removes the page header from the text.



```python
def removeHeader(text,word):
    new=text.replace(word,"")
    return new
```

1.5 POSTAG:

It may not be possible to manually provide the correct POS tag for every word for large texts. So, instead, we will find out the correct POS tag for each word and map it.



```python
def postag(s):
    doc=nlp(s)
    pos = [[token.text,token.pos_] for token in doc]
    return pos
```

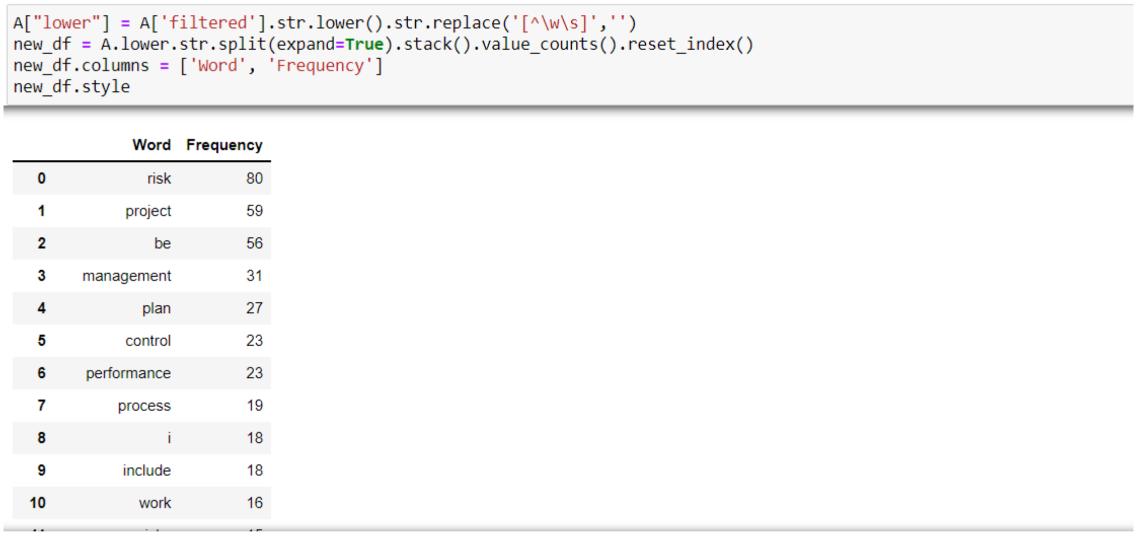
1.6 Lemmatization:

It's a Natural Language Processing technique that is used to prepare text, words, and documents for further processing. It designates a lexical treatment given to a text with a view to its analysis. This processing consists in applying to the occurrences of lexemes subject to inflection a coding referring to their common lexical entry, which is designated under the term of lemma.

```
lemmatizer=WordNetLemmatizer()
def lemm(s):
    l=[]
    for word,tag in pos_tag(word_tokenize(s)):
        wntag=tag[0].lower()
        wntag=wntag if wntag in ['a','r','n','v'] else None
        lemma=lemmatizer.lemmatize(word,wntag) if wntag else word
        l.append(lemma)
    return l
```

1.8 TF_IDF:

FIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

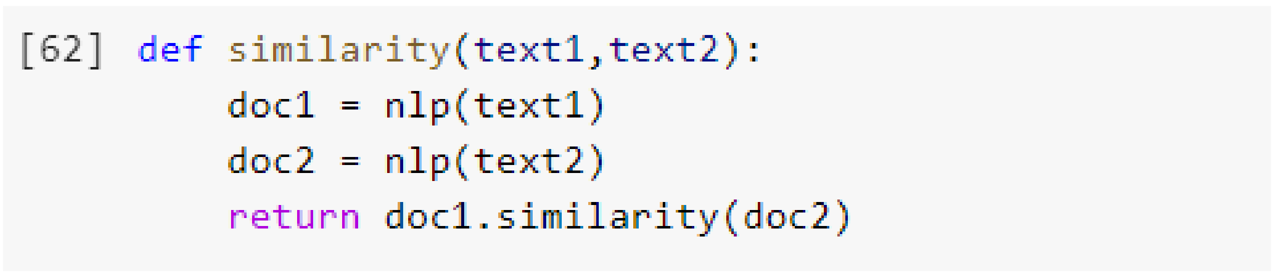


```
A["lower"] = A['filtered'].str.lower().str.replace('[^\w\s]','')
new_df = A.lower.str.split(expand=True).stack().value_counts().reset_index()
new_df.columns = ['Word', 'Frequency']
new_df.style
```

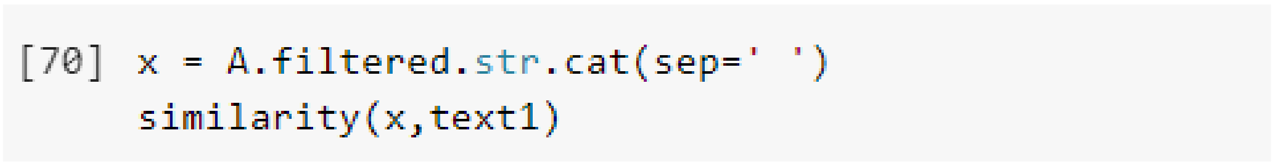
	Word	Frequency
0	risk	80
1	project	59
2	be	56
3	management	31
4	plan	27
5	control	23
6	performance	23
7	process	19
8	i	18
9	include	18
10	work	16

1.9 Similarity:

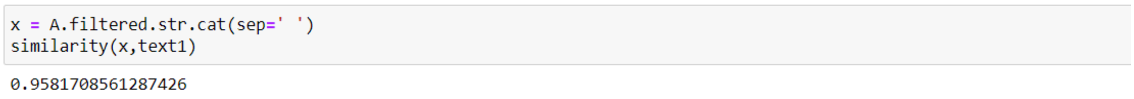
Text Similarity is one of the essential techniques of NLP which is being used to find the **closeness** between two chunks of text by it's meaning or by surfaces.
We used it in order to find out whether the text we started off with changed it’s meaning or not.
Our similarity rate was equal to 0.95.



```
[62] def similarity(text1,text2):
         doc1 = nlp(text1)
         doc2 = nlp(text2)
         return doc1.similarity(doc2)
```



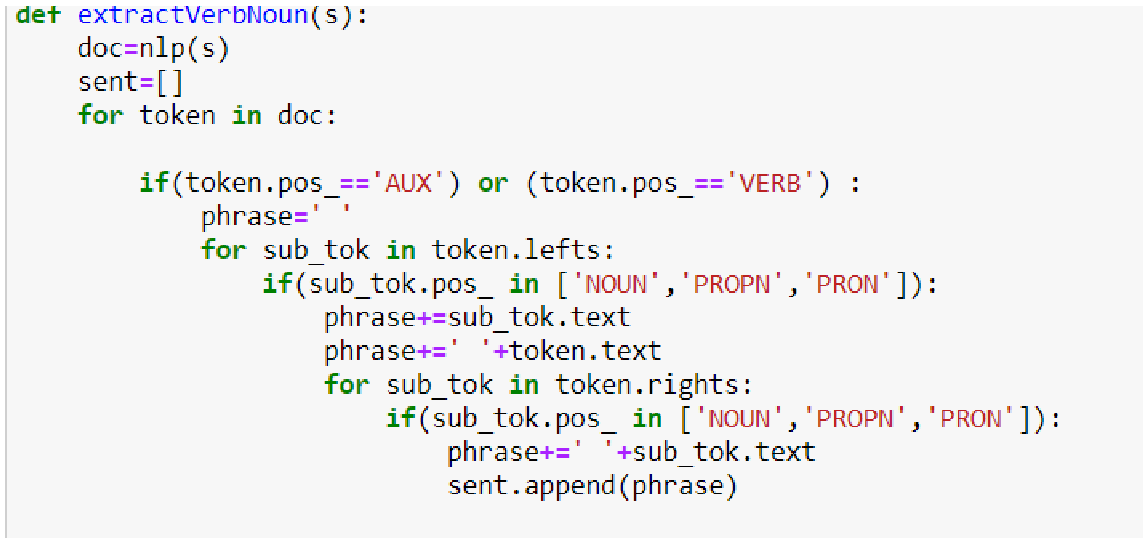
```
[70] x = A.filtered.str.cat(sep=' ')
     similarity(x,text1)
```



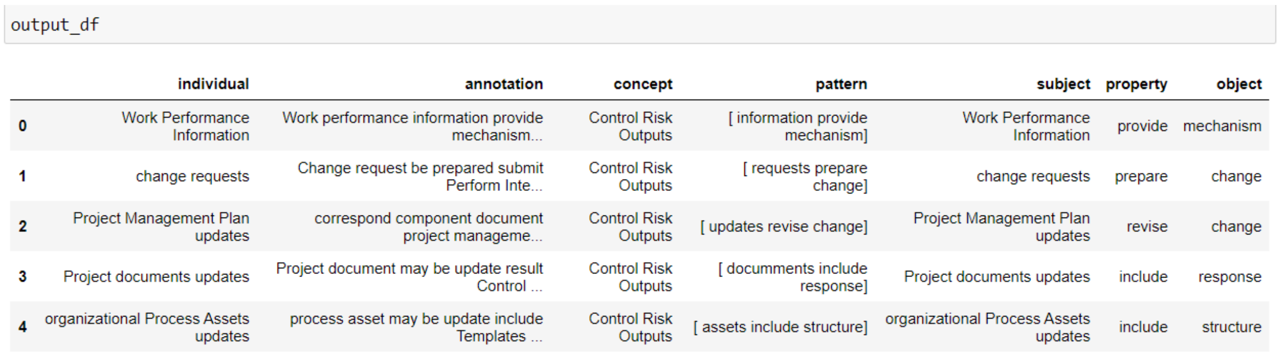
```
x = A.filtered.str.cat(sep=' ')
similarity(x,text1)

0.9581708561287426
```

1.10 Triplet extraction:

Triples are a way to represent information from a text sentence in fewer words without losing the context. There are different techniques for getting this information before you represent it as triples, and the techniques depend on the kind of data being read as input.



```
def extractVerbNoun(s):
    doc=nlp(s)
    sent=[]
    for token in doc:

        if(token.pos_=='AUX') or (token.pos_=='VERB') :
            phrase=' '
            for sub_tok in token.lefts:
                if(sub_tok.pos_ in ['NOUN','PROPN','PRON']):
                    phrase+=sub_tok.text
                    phrase+=' '+token.text
                    for sub_tok in token.rights:
                        if(sub_tok.pos_ in ['NOUN','PROPN','PRON']):
                            phrase+=' '+sub_tok.text
                            sent.append(phrase)
```

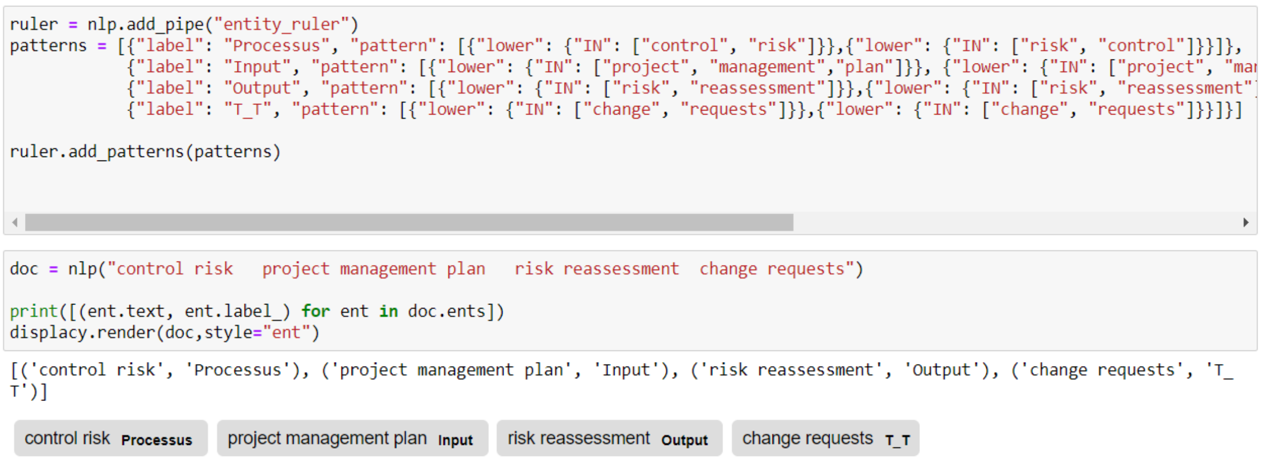


```
output_df
```

	individual	annotation	concept	pattern	subject	property	object
0	Work Performance Information	Work performance information provide mechanism...	Control Risk Outputs	[information provide mechanism]	Work Performance Information	provide	mechanism
1	change requests	Change request be prepared submit Perform Inte...	Control Risk Outputs	[requests prepare change]	change requests	prepare	change
2	Project Management Plan updates	correspond component document project manageme...	Control Risk Outputs	[updates revise change]	Project Management Plan updates	revise	change
3	Project documents updates	Project document may be update result Control ...	Control Risk Outputs	[documments include response]	Project documents updates	include	response
4	organizational Process Assets updates	process asset may be update include Templates ...	Control Risk Outputs	[assets include structure]	organizational Process Assets updates	include	structure

1.11 Rule based matching:

Compared to using regular expressions on raw text, spaCy's rule-based matcher engines and components not only let you find the words and phrases you're looking for – they also give you access to the tokens within the document and their relationships.



```
ruler = nlp.add_pipe("entity_ruler")
patterns = [{"label": "Processus", "pattern": [{"lower": {"IN": ["control", "risk"]}},{"lower": {"IN": ["risk", "control"]}}]},
            {"label": "Input", "pattern": [{"lower": {"IN": ["project", "management","plan"]}}, {"lower": {"IN": ["project", "ma
            {"label": "Output", "pattern": [{"lower": {"IN": ["risk", "reassessment"]}},{"lower": {"IN": ["risk", "reassessment"]
            {"label": "T_T", "pattern": [{"lower": {"IN": ["change", "requests"]}},{"lower": {"IN": ["change", "requests"]}}]}]

ruler.add_patterns(patterns)
```

```
doc = nlp("control risk   project management plan   risk reassessment  change requests")

print([(ent.text, ent.label_) for ent in doc.ents])
displacy.render(doc,style="ent")
```

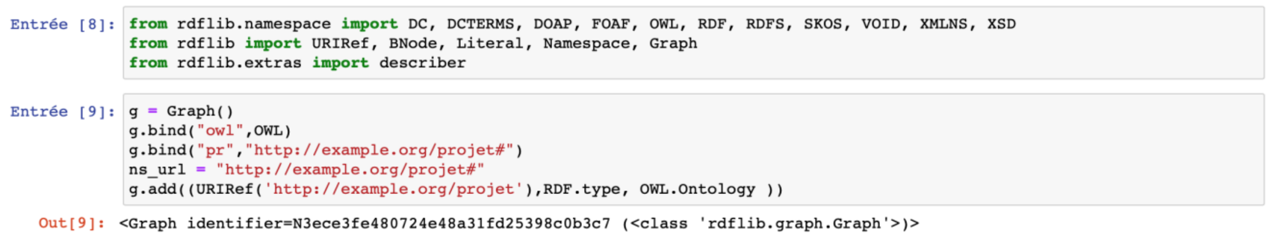
```
[('control risk', 'Processus'), ('project management plan', 'Input'), ('risk reassessment', 'Output'), ('change requests', 'T_
T')]
```

control risk **Processus** project management plan **Input** risk reassessment **Output** change requests **T_T**

Chapter2: Ontology

1.OWL components building process

Using RDFLIB, we convert our obtained dataframes from the pre-processing process into OWL format. Then we add the properties and the classes into triples format while keeping the relationships between them. Beside we add prefixes to entities in order to differentiate between them:

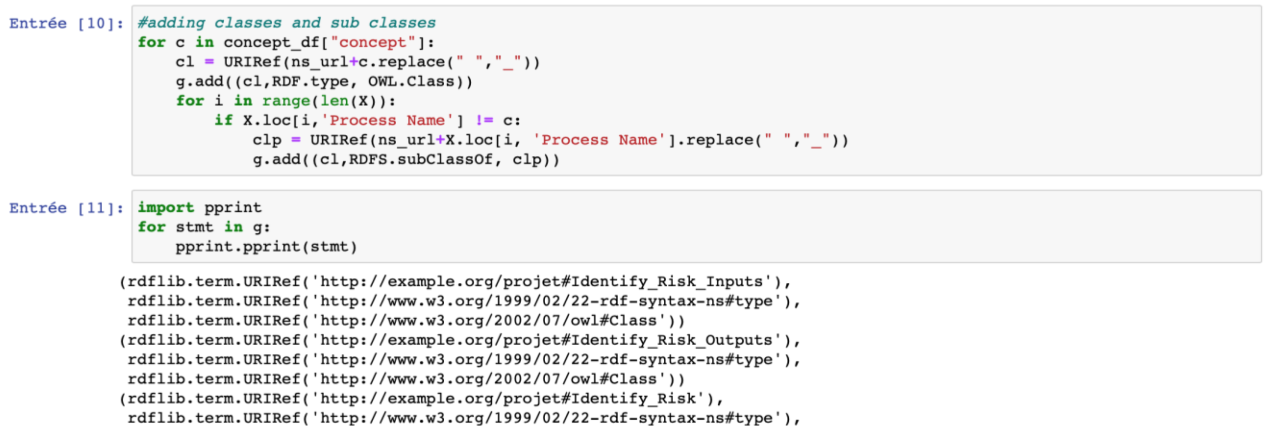


```
Entrée [8]: from rdflib.namespace import DC, DCTERMS, DOAP, FOAF, OWL, RDF, RDFS, SKOS, VOID, XMLNS, XSD
            from rdflib import URIRef, BNode, Literal, Namespace, Graph
            from rdflib.extras import describer

Entrée [9]: g = Graph()
            g.bind("owl",OWL)
            g.bind("pr","http://example.org/projet#")
            ns_url = "http://example.org/projet#"
            g.add((URIRef('http://example.org/projet'),RDF.type, OWL.Ontology ))

   Out[9]: <Graph identifier=N3ece3fe480724e48a31fd25398c0b3c7 (<class 'rdflib.graph.Graph'>)>
```

2. Add classes and sub-classes :

The ontology class attribute can be used to associate your class to the given one.

We use the top-down approach to develop the classes and subclasses hierachy . we start with creating classes for the general concepts, our six processes Then we specialize each one by its inputs,outputs and tools.



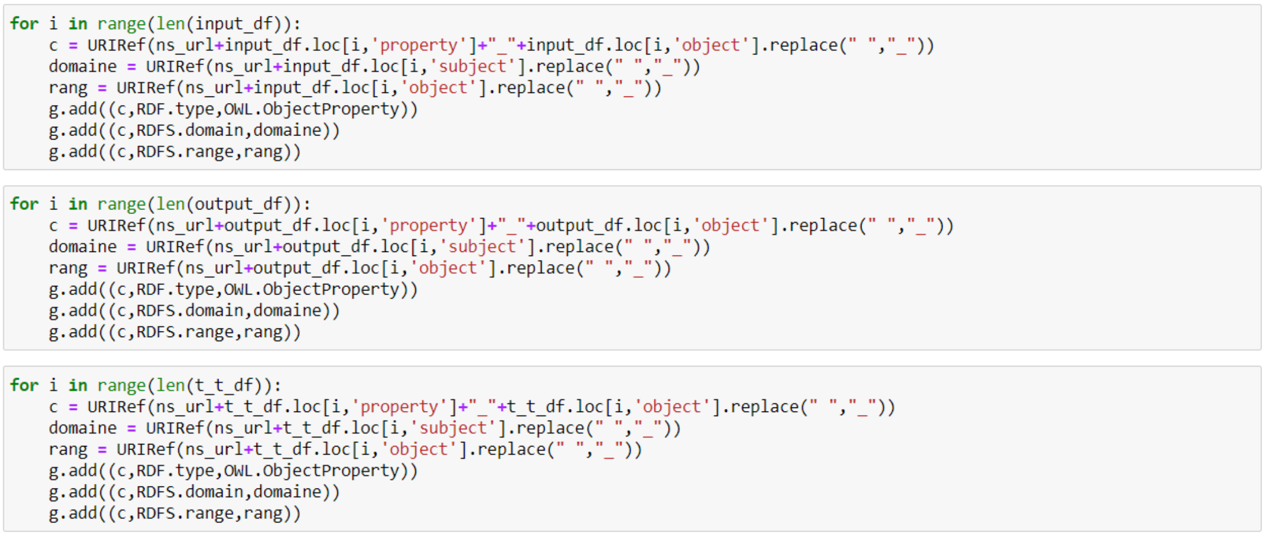
```
Entrée [10]: #adding classes and sub classes
             for c in concept_df["concept"]:
                 cl = URIRef(ns_url+c.replace(" ","_"))
                 g.add((cl,RDF.type, OWL.Class))
                 for i in range(len(X)):
                     if X.loc[i,'Process Name'] != c:
                         clp = URIRef(ns_url+X.loc[i, 'Process Name'].replace(" ","_"))
                         g.add((cl,RDFS.subClassOf, clp))

Entrée [11]: import pprint
             for stmt in g:
                 pprint.pprint(stmt)

            (rdflib.term.URIRef('http://example.org/projet#Identify_Risk_Inputs'),
             rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'),
             rdflib.term.URIRef('http://www.w3.org/2002/07/owl#Class'))
            (rdflib.term.URIRef('http://example.org/projet#Identify_Risk_Outputs'),
             rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'),
             rdflib.term.URIRef('http://www.w3.org/2002/07/owl#Class'))
            (rdflib.term.URIRef('http://example.org/projet#Identify_Risk'),
             rdflib.term.URIRef('http://www.w3.org/1999/02/22-rdf-syntax-ns#type'),
```

3.add data proporties :

the classes alone will not provide enough information to build our ontology. Once we have defined the classes, we must describe the internal structure of concepts.in this case we have to add data proporties for each concept

4.Add object proporties :

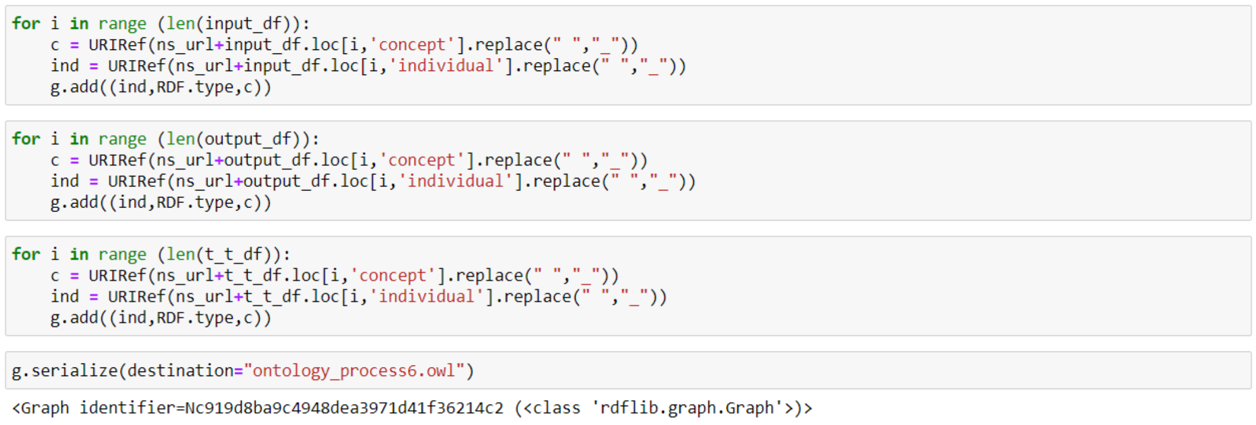
Object properties connect two individuals a subject and object with a predicate, while with data properties the predicate connects a single subject with some form of attribute data.



```
for i in range(len(input_df)):
    c = URIRef(ns_url+input_df.loc[i,'property']+"_"+input_df.loc[i,'object'].replace(" ","_"))
    domaine = URIRef(ns_url+input_df.loc[i,'subject'].replace(" ","_"))
    rang = URIRef(ns_url+input_df.loc[i,'object'].replace(" ","_"))
    g.add((c,RDF.type,OWL.ObjectProperty))
    g.add((c,RDFS.domain,domaine))
    g.add((c,RDFS.range,rang))
```

```
for i in range(len(output_df)):
    c = URIRef(ns_url+output_df.loc[i,'property']+"_"+output_df.loc[i,'object'].replace(" ","_"))
    domaine = URIRef(ns_url+output_df.loc[i,'subject'].replace(" ","_"))
    rang = URIRef(ns_url+output_df.loc[i,'object'].replace(" ","_"))
    g.add((c,RDF.type,OWL.ObjectProperty))
    g.add((c,RDFS.domain,domaine))
    g.add((c,RDFS.range,rang))
```

```
for i in range(len(t_t_df)):
    c = URIRef(ns_url+t_t_df.loc[i,'property']+"_"+t_t_df.loc[i,'object'].replace(" ","_"))
    domaine = URIRef(ns_url+t_t_df.loc[i,'subject'].replace(" ","_"))
    rang = URIRef(ns_url+t_t_df.loc[i,'object'].replace(" ","_"))
    g.add((c,RDF.type,OWL.ObjectProperty))
    g.add((c,RDFS.domain,domaine))
    g.add((c,RDFS.range,rang))
```



```
for i in range (len(input_df)):
    c = URIRef(ns_url+input_df.loc[i,'concept'].replace(" ","_"))
    ind = URIRef(ns_url+input_df.loc[i,'individual'].replace(" ","_"))
    g.add((ind,RDF.type,c))
```

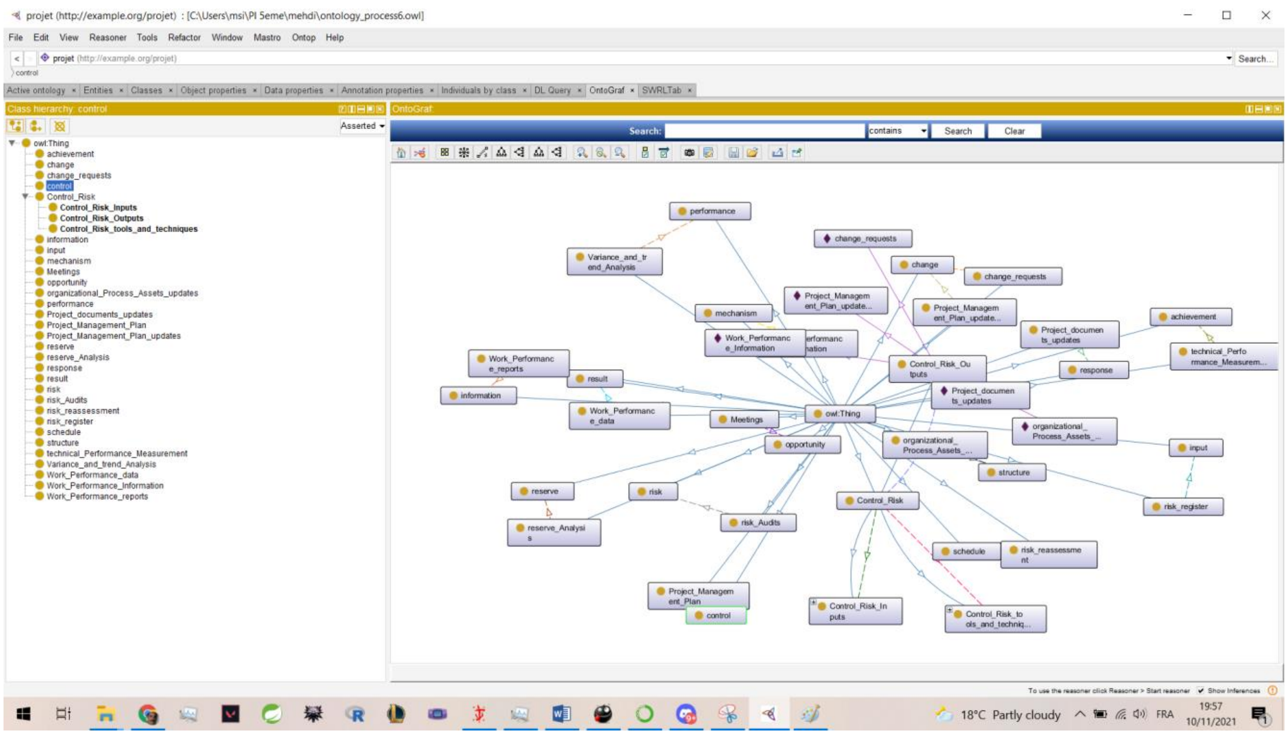
```
for i in range (len(output_df)):
    c = URIRef(ns_url+output_df.loc[i,'concept'].replace(" ","_"))
    ind = URIRef(ns_url+output_df.loc[i,'individual'].replace(" ","_"))
    g.add((ind,RDF.type,c))
```

```
for i in range (len(t_t_df)):
    c = URIRef(ns_url+t_t_df.loc[i,'concept'].replace(" ","_"))
    ind = URIRef(ns_url+t_t_df.loc[i,'individual'].replace(" ","_"))
    g.add((ind,RDF.type,c))
```

```
g.serialize(destination="ontology_process6.owl")
```

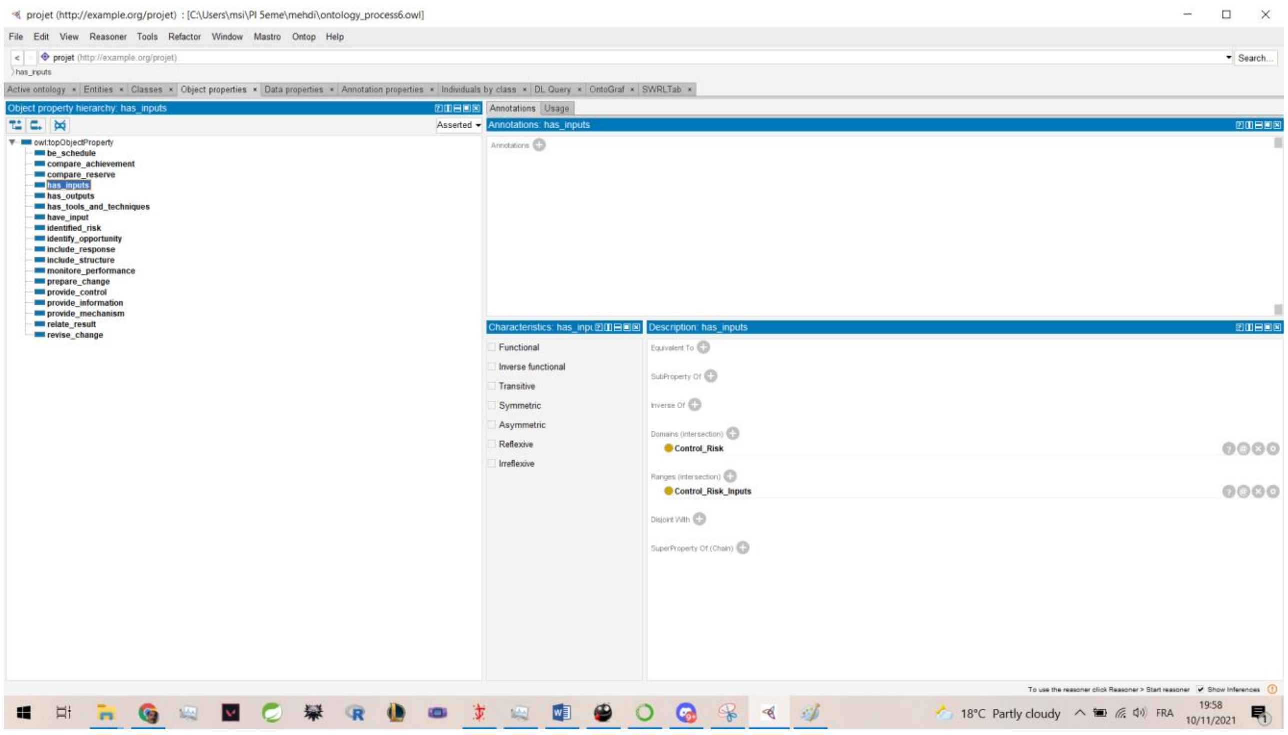
```
<Graph identifier=Nc919d8ba9c4948dea3971d41f36214c2 (<class 'rdflib.graph.Graph'>)>
```

4. ontology visualization with protégé of process 6 :


projet (http://example.org/projet) : [C:\Users\msi\PI 5eme\mehdi\ontology_process6.owl]

Class hierarchy: control

- owl:Thing
  - achievement
  - change
  - change_requests
  - control
  - Control_Risk
    - **Control_Risk_Inputs**
    - **Control_Risk_Outputs**
    - **Control_Risk_tools_and_techniques**
  - information
  - input
  - mechanism
  - Meetings
  - opportunity
  - organizational_Process_Assets_updates
  - performance
  - Project_documents_updates
  - Project_Management_Plan
  - Project_Management_Plan_updates
  - reserve
  - reserve_Analysis
  - response
  - result
  - risk
  - risk_Audits
  - risk_reassessment
  - risk_register
  - schedule
  - structure
  - technical_Performance_Measurement
  - Variance_and_trend_Analysis
  - Work_Performance_data
  - Work_Performance_Information
  - Work_Performance_reports


projet (http://example.org/projet) : [C:\Users\msi\PI 5eme\mehdi\ontology_process6.owl]

Object property hierarchy: has_inputs

- owl:topObjectProperty
  - **be_schedule**
  - **compare_achievement**
  - **compare_reserve**
  - **has_inputs**
  - **has_outputs**
  - **has_tools_and_techniques**
  - **have_input**
  - **identified_risk**
  - **identify_opportunity**
  - **include_response**
  - **include_structure**
  - **monitore_performance**
  - **prepare_change**
  - **provide_control**
  - **provide_information**
  - **provide_mechanism**
  - **relate_result**
  - **revise_change**

Characteristics: has_inputs

- Functional
- Inverse functional
- Transitive
- Symmetric
- Asymmetric
- Reflexive
- Irreflexive

Description: has_inputs

Equivalent To

SubProperty Of

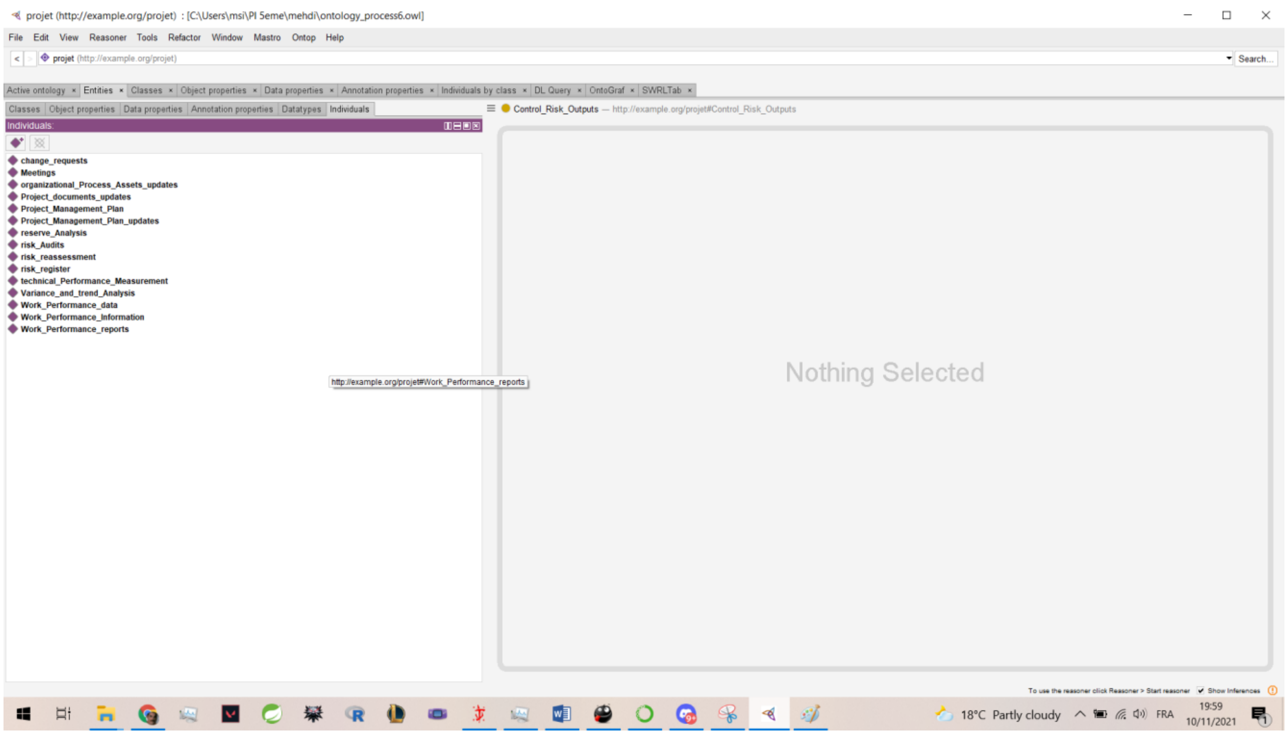
Inverse Of

Domains (intersection)
- **Control_Risk**

Ranges (intersection)
- **Control_Risk_Inputs**

Disjoint With

SuperProperty Of (Chain)



projet (http://example.org/projet) : [C:\Users\msi\PI 5eme\mehdi\ontology_process6.owl]

Control_Risk_Outputs — http://example.org/projet#Control_Risk_Outputs

Individuals:

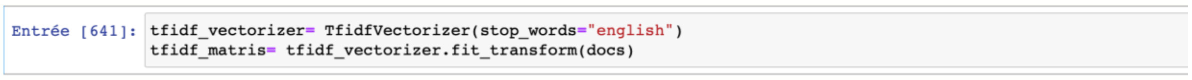
- change_requests
- Meetings
- organizational_Process_Assets_updates
- Project_documents_updates
- Project_Management_Plan
- Project_Management_Plan_updates
- reserve_Analysis
- risk_Audits
- risk_reassessment
- risk_register
- technical_Performance_Measurement
- Variance_and_trend_Analysis
- Work_Performance_data
- Work_Performance_Information
- Work_Performance_reports

http://example.org/projet#Work_Performance_reports

Nothing Selected

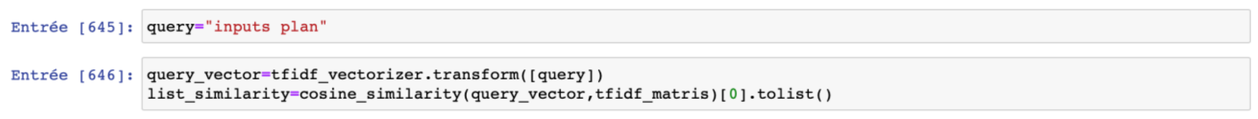
Chapter3 : Recommendation system implementation & deployment

In this step, we will explain how we implemented and deployed our recommendation system.
Creation of the model based on TF-IDF :

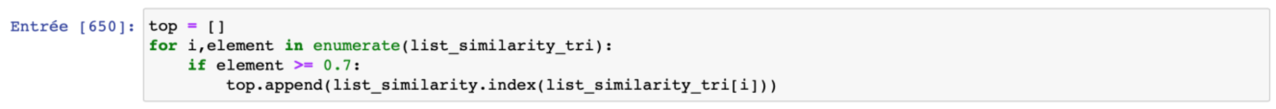


```
Entrée [641]: tfidf_vectorizer= TfidfVectorizer(stop_words="english")
              tfidf_matris= tfidf_vectorizer.fit_transform(docs)
```

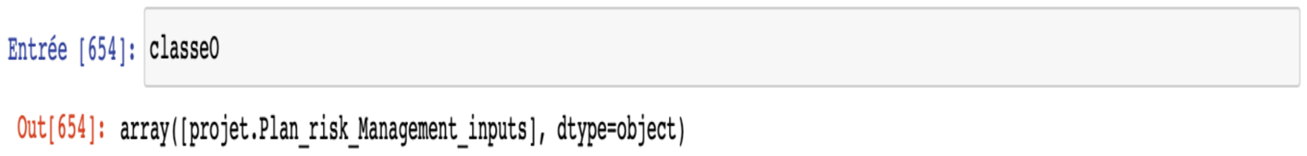
We used TF-IDF to create our model to compare query with our ontology classes.



```
Entrée [645]: query="inputs plan"

Entrée [646]: query_vector=tfidf_vectorizer.transform([query])
              list_similarity=cosine_similarity(query_vector,tfidf_matris)[0].tolist()
```



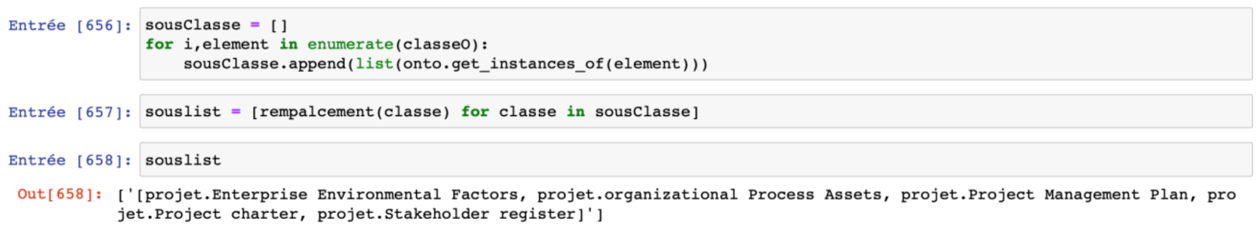
```
Entrée [650]: top = []
              for i,element in enumerate(list_similarity_tri):
                  if element >= 0.7:
                      top.append(list_similarity.index(list_similarity_tri[i]))
```



```
Entrée [654]: classe0

Out[654]: array([projet.Plan_risk_Management_inputs], dtype=object)
```

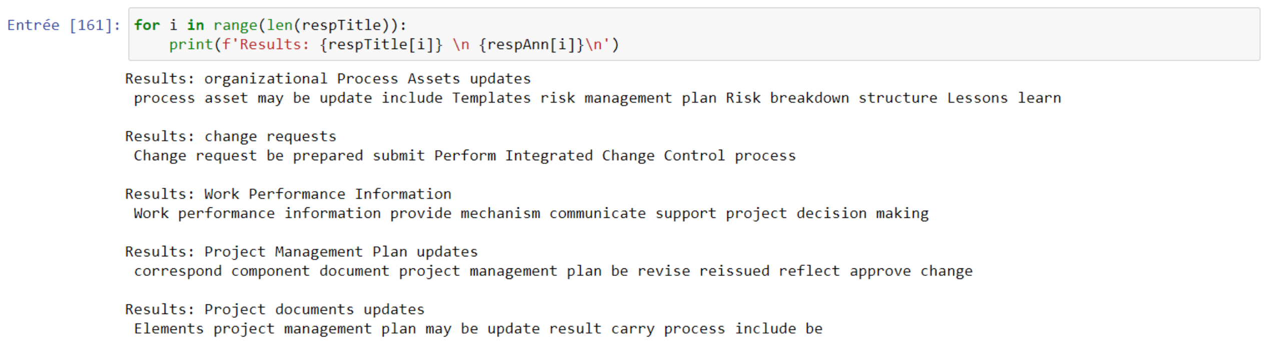
Here’s an example of similarity between our query and the ontology class with a similarity higher than 70%.

Here’s the ‘Inputs’ of ‘Plan Risk Management’ extracted from the ontology :



```
Entrée [656]: sousClasse = []
              for i,element in enumerate(classeO):
                  sousClasse.append(list(onto.get_instances_of(element)))

Entrée [657]: souslist = [rempalcement(classe) for classe in sousClasse]

Entrée [658]: souslist
  Out[658]: ['[projet.Enterprise Environmental Factors, projet.organizational Process Assets, projet.Project Management Plan, pro
            ject.Project charter, projet.Stakeholder register]']
```

This is an example of the final output of our recommendation system :



```
Entrée [161]: for i in range(len(respTitle)):
                  print(f'Results: {respTitle[i]} \n {respAnn[i]}\n')

              Results: organizational Process Assets updates
               process asset may be update include Templates risk management plan Risk breakdown structure Lessons learn

              Results: change requests
               Change request be prepared submit Perform Integrated Change Control process

              Results: Work Performance Information
               Work performance information provide mechanism communicate support project decision making

              Results: Project Management Plan updates
               correspond component document project management plan be revise reissued reflect approve change

              Results: Project documents updates
               Elements project management plan may be update result carry process include be
```

General conclusion

In this project, we used different technics and tools such as NLP for the preprocessing phase ,as well as a lot of research, to create a model that will make it easier for the client, who wants to lunch a project,to discover easily the risks that he can face .