# **Nicola PHD Project Code Documentation**

Table of Content

[**Nicola PHD Project Code Documentation** 1](#_Toc164345938)

[**Organizations’ documents corpus** 2](#_Toc164345939)

[**Word Bank** 3](#_Toc164345940)

[**UNDP** 6](#_Toc164345941)

[**IFAD** 8](#_Toc164345942)

[**AsDB** 14](#_Toc164345943)

[**Corpus creation** 18](#_Toc164345944)

[**Sentiment Analysis Models** 20](#_Toc164345945)

[**Training Dataset** 20](#_Toc164345946)

[**Test Dataset** 20](#_Toc164345947)

[**Models** 21](#_Toc164345948)

## **Organizations’ documents corpus**

The creation of a corpus is aimed at the later use of this for the identification of the critical factors present on the documents of projects of the domain of this work, i.e. partnered projects between a multilateral organization and one or more IFI (International financial institution); so those projects can be analyzed at an early stage and estimated their chance of success.

The code for this part of the project is available in the repository: jjoao/Nicola-partnerships.

Relevant multilateral organizations:

* World Bank (WB)
* Asian Development Bank (AsDB)
* International Fund for Agricultural Development (IFAD)
* United Nations Development Programme (UNDP)

Relevant IFI:

* African Development Bank (AfDB)
* Inter-American Development Bank (IADB)
* International Development Association (IDA)
* International Bank for Reconstruction and Development (IBRD)
* International Finance Corporation (IFC)
* Islamic Development Bank (IsDB)
* World Bank (WB)
* International Fund for Agricultural Development (IFAD)
* European Investment Bank (EIB)
* European Bank for Reconstruction and Development (EBRD)
* Asian Infrastructure Investment Bank (AIIB)
* Development Bank of Latin America and the Caribbean (CAF)
* Council of Europe Development Bank (CEB)
* Caribbean Development Bank (CDB)
* New Development Bank (NDB)
* United Nations Development Program (UNDP)
* Food and Agricultural Organization of the United Nations (FAO)
* United Nations International Children's Emergency Fund (UNICEF)
* United Nations Population Fund (UNFPA)
* United Nations Relief and Works Agency for Palestine Refugees (UNRWA)
* United Nations Environmental Program (UNEP)
* International Labour Organization (ILO)
* United Nations Entity for Gender Equality and the Empowerment of Women (UNWOMEN)
* United Nations Educational, Scientific and Cultural Organization (UNESCO)
* United Nations Industrial Development Organization (UNIDO)
* United Nations Human Settlements Programme (UNHABITAT)
* United Nations Capital Development Fund (UNCDF)

For each of the relevant multilateral organizations a branch of tools for **project identification**, correspondent **documents gathering**, and **partnership detection** were developed.

Having identified partnered projects, their documents are then processed so as to follow a common naming system and are grouped in the corpus, following some checks (e.g. language verification, file corruption).

Common naming system format:

**<multilateral organization>\_<project id>\_<file type>\_<is partnered>\_<list of partners separated with ‘\_’>\_<country>\_<date>\_<file id>**

The field ‘file id’ may not be filled as it is only needed in case a project has more than 1 document of the same type.

To reverse the names to their original format, it is necessary to use the projects metadata (to know how many partner fields there are).

### **Word Bank**

#### **Project and document gathering**

Relevant files: ‘get\_WB\_reports\_alt.py’, ‘get\_WB\_reports’.

In the case of WB, an API for the access of its projects and respective documents and even some metadata is available.

The main API paths used were:

* <https://search.worldbank.org/api/v2/wds> : used for requesting the documents for a specific project.
* <https://search.worldbank.org/api/v2/projects> : used for requesting the projects for a specific country.

The relevant document types we want for WB projects are:

* Project Appraisal Document
* Implementation Completion Report Review
* Implementation Completion and Results Report
* Project Performance Assessment Report

**Algorithm (pseudo-code):**

1. country\_codes <- load country codes list from file;

2. countries = country\_codes.split()

3. for country in countries:

4. country\_projects = get\_country\_projects(country); # filters projects that are closed and later than 2004

5. for project in country\_projects:

6. # save project metadata (eg. date,country,investment amount)

7. get\_project\_documents(project) # only downloads projects of relevant type

**Notes:**

1. The list of country codes is available on the repository.
2. The script runs in parallel the country loop so as to decrease run time (the countries list is divided equally between a specified number of threads).
3. Another version of this document gathering code is available on the repository (file ‘get\_WB\_reports’), created by professor J. João. This version was document oriented, whilst the above is project oriented, meaning that the relevant projects are first filtered, and the script then only asks for documents relevant projects.

**Output:**

The output is saved in a structured directory as such:

#### **Partnership detection**

Relevant files: ‘identify\_WB\_partnerships.py’.

For WB, the partnership detection of each project had to be done by analyses of their respective downloaded documents.

The types of documents analyzed are:

* Project Appraisal Document
* Implementation Completion and Results Report

**Project Appraisal Document Analysis**

1. partners = []

2. for page in pdf:

3. page\_text = page.get\_text()

4. if not partners and ‘financier’ in page\_text:

5. paragraphs = page\_text.split(‘\n’)

6. for paragraph in paragraphs:

7. found\_partners = text\_partners(paragraph)

8. if found\_partners:

9. # check if monetary reference is in paragraph

10. if 'usd' in paragraph.lower() or 'us$' in paragraph.lower() or '$' in paragraph.lower():

11. partners += found\_partners

12.   if partners:

13. break

With this code we aim to find the potential partners by searching for a page with the word ‘financier’ in it and where partners can be found in a paragraph coupled with a monetary reference.

**Implementation Completion and Results Report**

1. partners = []

2. for page in pdf:

3. page\_text = page.get\_text()

4. if not partners and ‘financier’ in page\_text and re.search(r'total\s+project\s+cost', page\_text, re.IGNORECASE):

5. paragraphs = page\_text.split(‘\n’)

6. for paragraph in paragraphs:

7. found\_partners = text\_partners(paragraph)

8. if found\_partners:

9. # check if monetary reference is in paragraph

10. if re.search(r'[\d,.]+', paragraph)

11. partners += found\_partners

12.   if partners:

13. break

With this code we aim to find the potential partners by searching for a page with the word ‘financier’ and the pattern 'total\s+project\s+cost', narrowing the search for the financing table (fitz can’t find the table specifically, so this is a workaround). Having found this target page, partners are looked for and are acceptable if they have a number (monetary reference) in the same paragraph (here expected to be same line).

**Notes:**

1. The identification of partners is done by looking for values from a list of IFI acceptable names and acronyms, available on the repository as “IFI\_list.json”.
2. When searching for one of the values in the text, the following regex expression is used: '([^\w\-‐]|^){}([^\w\-‐]|$)'; which in essence looks for the value as an independent word (can’t be found in the middle of other words). The value will be inputted in the “{}” in the expression.
3. Pdf reader used is the module “fitz” in python.
4. The identified partners are filtered so as to have no duplicates and to remove the organization itself, as it most likely is referenced on its own document.

#### **Data management**

Relevant files: ‘manage\_data.py’.

The last available script presents some tools for the analysis and manipulation of the downloaded data.

Some of the available functions are:

* **check\_data\_folder**: analysis the data folder, returning some counters such as the number of different types of documents, total number of projects, total number of partnered projects, total number of projects with documents, etc.
* **rename\_files**: used for renaming the files for the common corpus naming system.
* **inverse\_rename\_files**: used for returning the files from the common corpus naming system to their original names.

### **UNDP**

#### **Project and document gathering**

Relevant files: ‘get\_UNDP\_reports.py’.

In the case of UNDP, an API is available and can be easily explored and tested on <https://api.open.undp.org/api_documentation/api#/api>.

The main API paths used were:

* [https://api.open.undp.org/api/project\_list](https://api.open.undp.org/api) : used for requesting the list of projects. Multiple query parameters allow for an easier filtering of the projects.
* [https://api.open.undp.org/api/projects](https://api.open.undp.org/api) : used for requesting the specific information of a project, as well as its available documents.

The useful query parameters for [https://api.open.undp.org/api/project\_list](https://api.open.undp.org/api) were:

* **year**: allows to filter for projects from 2004 forward.
* **budget\_source**: allows for a request of only partnered projects. A list of ids (donor ids) is used to form this query, which can be found on [https://api.open.undp.org//api/donor-index.json](https://api.open.undp.org/). This list was analyzed for the ids that correspond to the organizations on the IFI list. It’s important to note that for multiple organizations more than 1 id was available, for which further API requests were made to reduce the number of potential ids. The final list that was used is available on the repository under the name “UNDP\_IFI\_funding\_id.json”.

For UNDP, the interesting types of documents are hard to define as their naming system is not as systematic as in the other organizations. Therefore, a different approach was used:

* Document name has “evaluation” in it but does not have “mtr” or “mid term report”.
* Mid term report final.
* Mid term report, if not other documents were found.

**Algorithm (pseudo-code):**

1. ifi\_donors\_ids <- load donors ids list from file;

2. projects = []

3. for donor\_id in ifi\_donors\_ids:

4. params = {‘year’: ‘>2004’ , ‘budget\_source’ : donor\_id}

5. donor\_projects = request(projects\_list\_api,params);

6. projects += donor\_projects

7. for project in projects:

8. project\_info = request(projects\_api)

9. project\_partners <- search ‘donor\_id’ fields in ‘outputs’ field of project\_info

10. # save metadata (dates,partners,country,name,etc.)

11. get\_project\_documents(project\_info) # inside project\_info in field ‘document\_name’ are available the project’s documents. For each, its name is checked according to the aforementioned approach and saved if the conditions are passed

**Notes:**

1. The script runs in parallel the project loop so as to decrease run time (the projects list is divided equally between a specified number of threads).

**Output:**

The output is saved in a structured directory as such:

#### **Partnership detection**

Because of the API’s available parameters, the partnered projects are directly request.

#### **Data management**

Relevant files: ‘manage\_data.py’.

Some of the available functions are:

* **check\_data\_folder**: analysis the data folder, returning some counters such as the number of different types of documents, total number of projects, total number of partnered projects, total number of projects with documents, etc.
* **rename\_files**: used for renaming the files for the common corpus naming system.
* **inverse\_rename\_files**: used for returning the files from the common corpus naming system to their original names.

### **IFAD**

#### **Project and document gathering**

Relevant files: ‘get\_IFAD\_reports.py’.

In the case of IFAD, no API is available and as such, a scraping of their websites was the chosen approach. <https://www.ifad.org/en> for the project and base documents gathering and <https://ioe.ifad.org/en/> for the projects’ evaluation documents.

The main paths used were:

* <https://www.ifad.org/en/web/operations/projects-and-programmes>: from here we gather the list of closed projects after 2004.
* <https://www.ifad.org/en/web/operations/-/project/>: path to project pages.
* <https://ioe.ifad.org/en/>: path to independent evaluations website.

The relevant document types we want for IFAD projects are:

* Project Completion Report
* Final Project Design Report
* Final Design Report
* Project Design Report
* Mid Term Review
* Supervision Report
* President’s Report
* (Mid-term) Review Report
* Project Performance Evaluation
* Project Performance Assessment

Some of these are duplicates of each other to take into account alternative names for the same type of document.

The document types: Project Performance Evaluation (PPE) and Project Performance Assessment (PPA); can only be obtained through the independent evaluations’ website. These documents will be mentioned below as considered the evaluation documents. All others will be mentioned as base documents.

**Algorithm (pseudo-code):**

1. Get projects list

1. projects\_list\_page = request(projects\_list\_url)

2. projects\_table <- obtained by identifying the closed projects table on the html of the page

3. projects = []

4. for project in projects\_list:

5. project\_metadata <- id,date,country,year,name. Obtained by analyzing the project row on the table

6. projects.append(project\_metadata)

2. Get projects’ base documents

In the projects page, we search for all the links whose text/description contains the name of one of the target document types.

1. project <- previously obtained list

2. for project in projects:

3. # create project folder

4. project\_page = request(project)

5. metadata\_table <- obtained by identifying the project info table on the page

6. # save metadata

7. page\_links <- gets all elements in html with ‘href’ parameter

8. for link in page\_links:

9. description = link.text

10. if description matches with target document type:

11. download\_document(link[‘href’])

3. Get projects’ evaluation documents

Since we don’t have a direct link between a project and its evaluation page, we need to use the project’s name to search for potential evaluation candidates. These candidates are filtered by comparing its metadata with the metadata of the project (year,country and name). Some of these comparisons are critical, such as the country and year, while the name is acceptable above a certain threshold of similarity - 0.8 using Jaro-Winkler similarity. Choosing the candidate with the greatest potential, we download the documents of the target types.

1. project <- previously obtained list

2. for project in projects:

3. # create project folder

4. evaluation\_projects\_search\_page = request(evaluation\_projects\_page,project[‘name’])

5. search\_results <- obtained by identifying the results table on the page’s html

6. for result in search\_results:

7. if better\_candidate: # the best candidate is decided by name semantic similarity with the project, comparison with the year of the candidate and its country

8. best\_candidate = result

9. if best\_candidate:

10. project\_evaluation\_page = request(best\_candidate)

11. for link in page\_links:

12. description = link.text

13. if description matches with target document type:

14. download\_document(link[‘href’])

**Notes:**

1. The analysis of the html pages is done using the python module ‘BeautifulSoup’
2. The script runs in parallel the functions that obtain the projects’ documents (base or evaluation).

**Output:**

The output is saved in a structured directory as such:

#### **Partnership detection**

Relevant files: ‘identify\_IFAD\_partnerships.py’.

For IFAD, the partnership detection of each project was done using their metadata, which should contain for partnered projects the field “co-financiers (international)”. For further assurance in the case that this field is not available, the project documents are also analyzed for the presence of partnerships.

The types of documents analyzed are:

* Project Completion Report
* (Final) Project Design Report
* President's Report
* Project Completion Report Digest

**Project Completion Report Analysis**

1. partners = []

2. for page in pdf:

3. page\_text = page.get\_text()

4. tables = page.find\_tables()

5. # actual cost table in page

6. if 'actual costs and financing' in page\_text:

7. for table in tables:

8. partners += partners\_in\_table(table)

9. # (2nd option) financial performance by financier table

10. if not partners and 'financial performance by financiers' in page\_text:

11. for table in tables:

12. partners += partners\_in\_table(table)

13. # (3rd option) projects costs and financing chapter

14. if not partnerships and 'project costs and financing' in page\_text:

15. paragraphs = page\_text.split(‘.\n’)

16. for paragraph in paragraphs:

17. found\_partners = text\_partners(paragraph)

18. # check if ‘financing’ is in paragraph

19. if found\_partners and ‘financing’ in paragraph:

20. partners += found\_partners

21. if partners:

22. break

In this report we have multiple possible locations to try and find partners, although these have different priorities. Firstly we look for the 'actual costs and financing' table and partners included within. If none were found, we try to find the financial performance by financiers table and do the same. Lastly, if we still have no partners, we look for a page with ‘project costs and financing’ (title of a chapter) and look for partners within its text, taking into account that the word ‘financing’ is present in the same paragraph as the partners.

**Project Design Report Analysis**

1. partners = []

2. for page in pdf:

3. page\_text = page.get\_text()

4. tables = page.find\_tables()

5. # financing plan table in page

6. if ‘financing plan’ in page\_text:

7. for table in tables:

8. for line in table:

9. found\_partners = text\_partners(line)

10. if found\_partners in line:

11. # check if partner invested (non 0 cell)

12. for cell in line:

13. if cell != 0:

14. partners += found\_partners

15. # (2nd option) financing plan by component table

16. if no partners and ‘financing plan by component’ in page\_text:

17. for table in tables:

18. for line in table:

19. found\_partners = text\_partners(line)

20. if found\_partners in line:

21. # check if partner invested (non 0 cell)

22. for cell in line:

23. if cell != 0:

24. partners += found\_partners

25. # (3rd option) project financing chapter

26. if no partners and ‘project financing’ in page\_text:

27. paragraphs = page\_text.split(‘\n’)

28. for paragraph in paragraphs:

29. found\_partners = text\_partners(paragraph)

30. if found\_partners:

31. # check if monetary reference is in paragraph

32. if ‘usd’ in paragraph or ‘us$’ in paragraph or ‘$’ in paragraph

33. partners += found\_partners

34. if partners:

35. break

Much like with the last document we have different options with different levels of priority. Firstly, the financing plan table is looked for and, within it we look for partners, taking into account that these are only valid if their corresponding cells in the table have at least one nonzero value (meaning that they invested money in the project). Secondly and with the same criteria, we look for the project plan by component table. Lastly, we look for the project financing chapter, where a partner can only be found in a paragraph that as a monetary reference (‘usd’, ’us$’ or ‘$’).

**President’s Report Analysis**

1. partners = []

2. cofinancers\_listing\_pattern = r'Cofinancier(\(s\)|s)?:\d\*\s\*((\w|\s)\*?\(\w+\))+'

3. cofinancers\_pattern = r'(\w|\s)\*?\(\w+\)'

4. cofinancers\_listing = []

5. for page in pdf:

6. page\_text = page.get\_text()

7. if re.search(cofinancers\_listing\_pattern,page\_text):

8. cofinancers\_listing = re.search(cofinancers\_listing\_pattern, page\_text).group(0)

9. break

10. if cofinancers\_listing:

11. partners <- extract partners from cofinancers\_listing through splitting and cleaning of the string and the use of the ‘cofinancers\_pattern’ expression

For the president’s report we look for a pattern (variable ‘cofinancers\_listing\_pattern’) very specific to the initial pages of the document. If that pattern is found, then the cofinanciers can be found on within the matches of the regular expression. From these matches we can then use further cleaning and regular expressions to individually extract each of the partners.

**Project Completion Report Digest Analysis**

1. partners = []

2. cofinancers\_listing\_pattern\_1 = r'Cofinanciers \(if any\)\s\*(([^:]+:[^:;]+\d)+)'

3. cofinancers\_listing\_pattern\_2 = r'Cofinanciers \(if any\)\s\*(([^-]+-[^-;]+\d)+)'

4. pattern\_used = None

5. cofinancers\_listing = []

6. for page in pdf:

7. page\_text = page.get\_text()

8. if re.search(cofinancers\_listing\_pattern\_1, page\_text):

9. cofinancers\_listing = re.search(cofinancers\_listing\_pattern\_1, page\_text).group(1)

10. pattern\_used = 1

11. break

12. if re.search(cofinancers\_listing\_pattern\_2, page\_text):

13. cofinancers\_listing = re.search(cofinancers\_listing\_pattern\_2, page\_text).group(1)

14. pattern\_used = 2

15. break

16. if cofinancers\_listing:

17. partners <- extract partners from cofinancers\_listing through splitting and cleaning of the string (depending on the pattern used above, different processing is needed)

Similar to the last document type, we look for a very specific pattern in which the cofinanciers will be listed. The only big difference to the last analysis is that this type of document may use more than one pattern for the same purpose, and as such we take that into consideration.

**Notes:**

1. The identification of partners is done by looking for values from a list of IFI acceptable names and acronyms, available on the repository as “IFI\_list.json”.
2. When searching for one of the values in the text, the following regex expression is used: '([^\w\-‐]|^){}([^\w\-‐]|$)'; which in essence looks for the value as an independent word (can’t be found in the middle of other words). The value will be inputted in the “{}” in the expression.
3. Pdf reader used is the module “fitz” in python.
4. The identified partners are filtered so as to have no duplicates and to remove the organization itself, as it most likely is referenced on its own document.

#### **Data management**

Relevant files: ‘manage\_data.py’.

The last available script presents some tools for the analysis and manipulation of the downloaded data.

Some of the available functions are:

* **check\_data\_folder**: analysis the data folder, returning some counters such as the number of different types of documents, total number of projects, total number of partnered projects, total number of projects with documents, etc.
* **rename\_files**: used for renaming the files for the common corpus naming system.
* **inverse\_rename\_files**: used for returning the files from the common corpus naming system to their original names.

### **AsDB**

#### **Project and document gathering**

Relevant files: ‘get\_ADB\_reports.py’.

In the case of AsDB, no API is available and as such, a scraping of its website (<https://www.adb.org>) was the chosen approach.

The main paths used were:

* [https://www.adb.org/projects/<project\_id>/main](https://www.adb.org/projects/): path to a specific project page.
* <https://data.adb.org/dataset/adb-sovereign-projects>: path to projects dataset, compiled by AsDB.

The projects list is obtained from the above-mentioned dataset, which lists all AsDB projects since 2005.

The relevant document types we want for IFAD projects are:

* Project/Program Completion Report
* Reports and Recommendations of the President
* Technical Assistance Completion Report
* Design and Monitoring Framework
* PCR Validation Report
* Validations of Project Completion Reports
* Project Performance Evaluation Report
* TA Performance Evaluation Reports

**Algorithm (pseudo-code):**

1. Get project list

The downloaded projects dataset has some redundant and useless information for our purpose, therefore needing some processing. Firstly, we remove duplicate project mentions by filtering the ‘project\_id’ column. Secondly, we filter the projects to only leave the closed projects approved after 2004. Lastly, some entries had more than one id, so we split these.

1. projects <- previously obtained dataset

2. projects[‘project\_id’].drop\_duplicates()

3. projects = [p for p in projects if p[‘approval data’] > 2004 and p[‘status’] == ‘closed’]

4. projects[‘project\_id’] = [p[‘project\_id’].split(‘/’)[0] for p in projects]

2. Get projects’ documents

A project’s page has its corresponding documents displayed in multiple tables with a similar structure. As such, we first need to filter the tables in the page to only leave the one that refer to documents – these have ‘document type’ in its header. From here, we can check their rows in which the type of the document is displayed in the ‘document type’ column and, if the type matches with one of the relevant types, we download the document. Also, because the dataset from the previous step does not provide the end date of the projects, we search for the row with the column ‘Last Pds Update’ and save its value on the metadata.

1. project <- previously obtained list

2. for project in projects:

3. # create project folder

4. # save metadata

6. project\_page = request(project)

7. page\_tables <- gets all table elements in html

8. filtered\_tables = []

9. # find all document tables

10. for table in page\_tables:

11. if ‘document type’ in table.header:

12. filtered\_tables.append(table)

13. rows <- get all rows from filtered\_tables

14. # get project’s documents

15. for row in rows:

16. document\_type = row[1]

17. if document\_type matches target document type:

18. link = row.find(‘a’,href=True)

19. download\_document(link)

20. # get extra metadata info (end data; not available on the initial dataset)

21. rows <- get all rows in html

22. end\_data = None

23. for row in rows:

24. if ‘Last PDS Update’:

25. end\_data <- process row to get year of closure

26. # save new metadata

**Notes:**

1. The dataset is manipulated using the python module ‘pandas’.
2. The analysis of the html pages is done using the python module ‘BeautifulSoup’ .
3. The script runs in parallel the function that obtain the projects’ documents (base or evaluation).
4. Consecutive requests to the AsDB website can lead to an IP block, so the use of a VPN may be required to run the scripts.

**Output:**

The output is saved in a structured directory as such:

#### **Partnership detection**

Relevant files: ‘identify\_ADB\_partnerships.py’.

For AsDB, the partnership detection of each project had to be done by analyses of their home page on the website and their downloaded documents.

The types of documents analyzed are:

* Report and Recommendation from the President
* Project Completion Report

If both documents are available for the project, a cross-check between the partners found on each document is done to increase the chances of finding true partners.

**Project home page**

In the project page, we search for a row with a column ‘source of funding / amount’ and, if found, we search its value for partners. The value of the row is a table which lists the partners of the project and their provided funding.

**Report and Recommendation from the President Analysis**

1. partners = []

2. for page in pdf:

3. page\_text = page.get\_text()

4. tables = page.find\_tables()

5. # financing plan table

6. if 'financing plan' in page\_text:

7. financing\_plan\_table\_page = False

8. lines = page\_text.split(‘\n’)

9. # confirm that if it is actually the financing plan table

10. for line in lines:

11. if 'table' in line and 'financing plan' in line:

12. financing\_plan\_table\_page = True

13. break

14. if financing\_plan\_table\_page:

15. # find partners

16. paragraphs = page\_text.split('.\n')

17. for paragraph in paragraphs:

18 partnerships += text\_partners(paragraph)

19. if partners:

20. break

In this report we look for the financing plan table. Unlike in previous cases, the pdf reader cannot find the tables in this document type because of their format, requiring a simpler look for the table, done by looking for the strings ‘table’ and ‘financing table’ in a single line. If it is confirmed that the table has been found, we look for partners in the page’s text. This method of search is more lenient as the search for partners is not done only on the table itself, but on the whole page.

**Project Completion Report Analysis**

1. partners = []

2. for page in pdf:

3. page\_text = page.get\_text()

4. if ‘cofin’ in page\_text:

5. paragraphs = page\_text.split(‘\n’)

6. for paragraph in paragraphs:

7. if ‘cofin’ in paragraph or ‘co-fin’ in paragraph:

8. partners += text\_partners(paragraph)

9. # (2nd method) project data table

10. if not partners and ‘project data’ in page\_text:

11. tables = page.find\_tables()

12. for table in tables:

13. partners += partners\_in\_table(table)

14. if partners:

15. break

For this document type we have two methods of finding partnerships. The priority method is to find the string ‘cofin’ in a paragraph in which partners are present. The string ‘cofin’ allows for a match with different variations of the word cofinance (cofinanciers, cofinancing, cofinance, etc.). If no partners were found, then we look for the page with the project data table and search for partners in the page’s tables.

**Notes:**

1. The identification of partners is done by looking for values from a list of IFI acceptable names and acronyms, available on the repository as “IFI\_list.json”.
2. When searching for one of the values in the text, the following regex expression is used: '([^\w\-‐]|^){}([^\w\-‐]|$)'; which in essence looks for the value as an independent word (can’t be found in the middle of other words). The value will be inputted in the “{}” in the expression.
3. Pdf reader used is the module “fitz” in python.
4. The identified partners are filtered so as to have no duplicates and to remove the organization itself, as it most likely is referenced on its own document.

#### **Data management**

Relevant files: ‘manage\_data.py’.

The last available script presents some tools for the analysis and manipulation of the downloaded data.

Some of the available functions are:

* **check\_data\_folder**: analysis the data folder, returning some counters such as the number of different types of documents, total number of projects, total number of partnered projects, total number of projects with documents, etc.
* **rename\_files**: used for renaming the files for the common corpus naming system.
* **get\_extra\_metada**: used for gathering extra metadata to the partnered projects.
* **inverse\_rename\_files**: used for returning the files from the common corpus naming system to their original names.

### **Corpus creation**

Relevant files: ‘create\_corpus.py’.

For the creation of the folder, we gather the documents from the partnered projects in the organizations’ data folders.

The corpus creation script has the following methods:

* **create\_corpus:** this function looks for the data folder in each of the organizations folders (AsDB,WB,UNDP and IFAD) and looks for the partnered projects – by checking the ‘partnerships’ field in their metadata – and copys them to a new corpus directory, including their metadata.
* **clean\_corpus:** this function iterates over the documents on the corpus, checking for their language – removing non-English files – and if the pdf files are not corrupted – can be opened by the pdf reader.  
  To check the files language, the python module ‘langdetect’ is used, where the threshold for the files to be valid is for at least 50% of a document’s text to be english.  
  Projects which in the end of cleaning have no available documents, are removed from the corpus.
* **analyze\_corpus:** this function returns an analysis of the corpus. Metrics such as the total number of projects, total number of projects, total number of types of documents of each organization, etc.

**Output:**

The created corpus folder follows the structure:

Where there are two main folders: the projects folder and documents folder. In the projects directory there is a partner folder wherein there is a folder for each organization in which all the partnered projects are saved as a folder with their metadata inside. The documents directory is divided into two folders, one for the design documents and the other for evaluation documents.

The documents categorized as evaluation are:

* AsDB
  + PPER
  + PVR
  + PCR
  + PPAR
* IFAD
  + PPE
  + PPA
  + PCR
  + PCRD
  + MTR
  + SR
* WB
  + ICR
  + ICRR
  + PPAR
* UNDP
  + MTR
  + PE

All other documents are considered as design.

### **Control Corpus**

The main purpose of the corpus is to find from the list of potential project factors compiled by Nicola, those which are critical for a project’s final evaluation and, moreover, those which pertain to projects involving partnerships. As such, the need to create a control corpus of non-partnered projects arises.

This second corpus (although it is stored in the same overall directory of the partner corpus) is created based on the projects of the partner corpus.

For all the organizations, the same approach for creating its portion of the control corpus will apply:

1. Gather from the partner corpus, the projects belonging to the organization and their corresponding metadata.
2. Gather all non-partnered projects from organizations data folder.
3. For each partnered project, it is compared with the still not chosen non-partnered projects according to the characteristics:
   1. Country
   2. Sector
   3. Year the project was finished

The order with which they were listed follows their priority when comparing the projects.

1. If a project is matched for the partnered project, save it in a control corpus directory for the organization.

#### **Matching algorithm**

The matching algorithm for all organizations is similar and can be described by the pseudo code:

1. control\_project = None

2.

3. priority\_evaluation\_doc = True

4. priority\_end\_year <- True if project has end date

5. priority\_country = True

6. priority\_sector <- True if project has sector

7. possible\_projects <- non-partnered projects not yet matched

8.

9. i = 0

10. while i < len(possible\_projects) and not control\_project:

11. valid = True

12. compare\_project = possible\_projects[i]

13.

14. # check priorities

15.

16. ## check if project has same country

17. if priority\_country:

18. if project’s ‘country’ is not the same as compare\_project:

19. valid = False

20.

21. ## check if project has same end year

22. if priority\_end\_year and valid:

23. if project’s ‘end\_year’ is not the same as compare\_project:

24. valid = False

25.

26. if priority\_sector and valid:

27. if project’s ‘sector’ is not the same as compare\_project:

28. valid = False

29.

30. ## check if project has evaluation document

31. if valid and priority\_evaluation\_doc:

32. has\_valid\_evaluation\_doc <- check if compare\_project has any document that is of evaluation type and is valid (not corrupeted and in english)

33. if not has\_evaluation\_doc:

34. valid = False

35.

36. if valid:

37. control\_project <- path to compare project’s directory

38. else:

39. # remove non-partner from projects to compare

40. possible\_projects.pop(compare\_project)

41.

42. i += 1

43. # if none found, uncheck priority and loop through projects again

44. if i == len(possible\_projects) and not control\_project:

45. i = 0

46. if priority\_end\_year:

47. priority\_end\_year = False

48. elif priority\_sector:

49. priority\_sector = False

50. elif priority\_country:

51. priority\_country = False

52. # reset other priorities

53. priority\_end\_year = True

54. priority\_sector = True

55. else:

56. break

57.

58.

59. return control\_project

#### **World Bank, IFAD and AsDB**

In the case of these organizations, when acquiring the projects, both partnered and non-partnered projects where gathered, as such, no further download of projects is needed for gathering the list of non-partnered projects.

#### **UNDP**

For UNDP, the download and identification of partnered projects was done simultaneously using their API, and as such its data folder does not contain any non-partnered projects. Therefore, an extra step is needed to list all non-partnered projects, involving requesting these from the API. This is done by, like when gathering its partnered projects, using the list of donors mentioned in the section: [UNDP](#_UNDP); and from the requested projects, remove those which include donors from said list.

#### **Joining Corpus**

After gathering the control projects for each organization, their documents are then renamed in the same manner as the partner projects, where the field <is partnered> will for every document be filled as False.

Afterwards, using the script ‘create\_corpus’ with the flag “-cc” or “—control\_corpus”, for each organization, their control corpus will be included in the overall corpus.

The end corpus directory will have the following directory tree:

The only notable addition is the control directory within the projects folder, wherein the control projects of each organization will be added. These projects’ documents will be included in the documents directory with no further delimitation as the naming system for the files already differentiates control from the partner documents.

### **Projects Ratings**

For the inference of the critical factors, some more information about the project’s is needed, namely, their rating, i.e. the project’s evaluation.

To gather this information, spreadsheets provided by each of the organizations was used to match the project ratings listed with the projects in the corpus, both control and partner projects.

This step, although simple in nature, as it comprises only of the matching of project ids, further decreases the number of usable projects in the corpus, as not all projects have been rated or have their rating disclosed publicly.

## **Sentiment Analysis Models**

For the identification of the critical factors in these kinds of projects, it is necessary to process the created corpus such as to identify positive, negative, and neutral elements of text (likely at sentence level). For that purpose, there surges a need to use sentiment analysis. However, this technology is not usually trained for this domain (economics and project lexicon) and does not provide, by default, very satisfactory results. As such, retraining these models is a task that could prove beneficial for the purpose of the project.

The code for this part of the project is available in the repository: **nicancies/evaluation-sentiment-analysis**.

### **Training Dataset**

The training dataset was created by the aggregation of the available sentiment scores on paragraphs of the AIDA model. These are available on <https://aida.undp.org/landing>.

The corpus was downloaded by searching by each of the SDG (17 in total) on <https://aida.undp.org/sdg>, filtering the results by sentiment value (from positive to negative) and, for each sentiment using the action ‘export all’ in the page.

**Corpus info:**

|  |  |
| --- | --- |
| **Sentiment** | **Total Sentences** |
| **Positive** | 33805 |
| **Mixed Positive** | 33138 |
| **Neutral** | 32930 |
| **Mixed Negative** | 32983 |
| **Negative** | 32660 |
|  | 165516 |

Number of files: 85

Code for the creation of the dataset can be found on the repository in the script: ‘/stanza/train\_test/create\_dataset.py’. The respective function is ‘create\_dataset’. Starting from the raw AIDA files, these can be processed using the function ‘process\_files’ in the same script. Having the processed\_files, ‘create\_dataset’ can be run to build the dataset in json format, divided in 3 files: train, dev and test.

### **Test Dataset**

The test dataset, used for comparison between the trained models, is a subset of the above AIDA dataset. It has a much smaller number of items as the testing of the models could take multiple days if the whole dataset was used (even when using better hardware, such as the GPUs provided by Kaggle). The dataset is built from the processed files.

Requirements:

* Same number of items of each SDG
* At least 5% of the size of the AIDA dataset
* Same number of items of each sentiment
* Items are randomly chosen

Code for the creation of the test dataset can be found in the script ‘/stanza/train\_test/create\_dataset.py’, function ‘create\_test\_dataset’.

### **Models**

The NLP models used for the testing are: Stanza (started), Flair (TODO), Vader (TODO) and TextBlob (TODO).

#### **Stanza**

Stanza allows for retraining of specific pipeline models. In our case, we want to retrain only the sentiment classifier (<https://stanfordnlp.github.io/stanza/new_language_sentiment.html>). A script to turn the raw AIDA dataset into Stanza acceptable is available in the repository in the stanza directory as ‘create\_dataset.py’.

##### **Base model**

**Results:**

* **Confusion Matrix:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **0** | **1** | **2** |
| **0** | 958 | 558 | 104 |
| **1** | 713 | 3716 | 868 |
| **2** | 180 | 814 | 690 |

* **Accuracy:** 0.57

##### **Version 1**

**Model information:**

* **Hyper parameters:**
  + Optimizer: madgrad
  + Seed: 432026036
  + Epochs: 100
* **Data:**
  + Sentiment labels: [‘negative’,’neutral’,’positive’]
    - Mixed sentiment was labeled as neutral
    - Labels from 0-2.
  + Train dataset: 57600 sentences
    - Positive: 20%
    - Neutral: 60%
    - Negative: 20%
    - Text processing:
      * Text cut at first punctuation (‘.’, ’!’, ’?’) before reaching 100 words

**Results:**

* **Confusion Matrix:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **0** | **1** | **2** |
| **0** | 1173 | 431 | 36 |
| **1** | 713 | 3716 | 453 |
| **2** | 34 | 617 | 1056 |

* **Accuracy:** 0.72

##### **Version 2**

**Model information:**

* **Hyper parameters:**
  + Optimizer: madgrad
  + use\_bert: True
  + Seed: 432026036
  + Epochs: 35
* **Data:**
  + Sentiment labels: [‘negative’,’neutral’,’positive’]
    - Mixed sentiment was labeled as neutral
    - Labels from 0-2.
  + Train dataset: 57600 sentences
    - Positive: 20%
    - Neutral: 60%
    - Negative: 20%
    - Text processing:
      * Text cut at first punctuation (‘.’, ’!’, ’?’) before reaching 100 words
  + Test dataset (training):
    - Text processing:
      * Text cut at first punctuation (‘.’, ’!’, ’?’) before reaching 100 words
  + Validation dataset (training):
    - Text processing:
      * Text cut at first punctuation (‘.’, ’!’, ’?’) before reaching 100 words

**Results:**

* **Confusion Matrix:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **0** | **1** | **2** |
| **0** | 1160 | 416 | 44 |
| **1** | 544 | 3676 | 705 |
| **2** | 17 | 420 | 1247 |

* **Accuracy:** 0.74

##### **Version 3**

**Model information:**

* **Hyper parameters:**
  + Optimizer: madgrad
  + use\_bert: True
  + filter\_channels: 5
  + fc\_shapes: 10
  + Seed: 432026036
  + Epochs: 72
* **Data:**
  + Sentiment labels: [‘negative’,’neutral’,’positive’]
    - Mixed sentiment was labeled as neutral
    - Labels from 0-2.
  + Train dataset: 57600 sentences
    - Positive: 20%
    - Neutral: 60%
    - Negative: 20%
    - Text processing:
      * Text cut at first punctuation (‘.’, ’!’, ’?’) before reaching 100 words
  + Test dataset (training):
    - Text processing:
      * Text cut at first punctuation (‘.’, ’!’, ’?’) before reaching 100 words
  + Validation dataset (training):
    - Text processing:
      * Text cut at first punctuation (‘.’, ’!’, ’?’) before reaching 100 words

**Results:**

* **Confusion Matrix:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **0** | **1** | **2** |
| **0** | 899 | 684 | 37 |
| **1** | 178 | 4283 | 464 |
| **2** | 8 | 515 | 1161 |

* **Accuracy:** 0.77

##### **Version 4**

**Model information:**

* **Hyper parameters:**
  + Optimizer: madgrad
  + filter\_channels: 5
  + fc\_shapes: 10
  + Seed: 432026036
  + Epochs: 72
* **Data:**
  + - Sentiment labels: [‘negative’,’mixed\_negative’,’neutral’,’mixed\_positive’,’positive’]
      * Labels from 0-4.
  + Train dataset: 57600 sentences
    - Positive: 20%
    - Mixed Positive: 20%
    - Neutral: 20%
    - Mixed Negative: 20%
    - Negative: 20%
    - Text processing:
      * Text cut at first punctuation (‘.’, ’!’, ’?’) before reaching 100 words

**Results:**

* **Confusion Matrix:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** |
| **0** | 1400 | 30 | 123 | 7 | 60 |
| **1** | 691 | 127 | 533 | 61 | 228 |
| **2** | 85 | 12 | 1440 | 25 | 75 |
| **3** | 207 | 108 | 503 | 172 | 658 |
| **4** | 46 | 18 | 195 | 56 | 1369 |

* **Accuracy:** 0.55

##### **Version 5**

**Model information:**

* **Hyper parameters:**
  + Optimizer: madgrad
  + use\_bert : True
  + filter\_channels: 5
  + fc\_shapes: 10
  + Seed: 432026036
  + Epochs: 72
* **Data:**
  + - Sentiment labels: [‘negative’,’mixed\_negative’,’neutral’,’mixed\_positive’,’positive’]
      * Labels from 0-4.
  + Train dataset: 57600 sentences
    - Positive: 20%
    - Mixed Positive: 20%
    - Neutral: 20%
    - Mixed Negative: 20%
    - Negative: 20%
    - Text processing:
      * Text cut at first punctuation (‘.’, ’!’, ’?’) before reaching 100 words
  + Test dataset (training):
    - Text processing:
      * Text cut at first punctuation (‘.’, ’!’, ’?’) before reaching 100 words
  + Validation dataset (training):
    - Text processing:
      * Text cut at first punctuation (‘.’, ’!’, ’?’) before reaching 100 words

**Results:**

* **Confusion Matrix:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** |
| **0** | 1325 | 45 | 161 | 3 | 86 |
| **1** | 592 | 148 | 575 | 26 | 299 |
| **2** | 84 | 18 | 1459 | 7 | 69 |
| **3** | 205 | 119 | 632 | 45 | 647 |
| **4** | 43 | 22 | 234 | 18 | 1367 |

* **Accuracy:** 0.53

##### **Usage**

To use one of the trained models for sentiment analysis in python the following code can be used:

1. import stanza

2. stanza\_retrained\_nlp = stanza.Pipeline(

lang='en',

processors='tokenize,sentiment',

sentiment\_model\_path=stanza\_retrained\_model\_dir,

tokenize\_pretokenized=True, # use in the when working with pretokenized text

tokenize\_no\_ssplit=True # use in the when working with pretokenized text

)

3. content <- piece of text to analyze

4. doc = stanza\_retrained\_nlp(content)

5. stanza\_score = doc.sentences[0].sentiment if doc.sentences else -1