RELEVANT, INCLUSIVE, TRUSTED AND OPEN INDICATORS
FOR RESEARCH AND INNOVATION POLICY

# D3.4 Code and Data Repositories

| | | | |
|---|---|---|---|
| **Deliverable No.** | | D3.4 | |
| **Workpackage No.** | 3 | **Workpackage Title** | Data Collection and Analysis |
| **Lead beneficiary** | | DTU | |
| **Dissemination level** | | Public | |
| **Type** | | ORDP: Open Research Data Pilot | |
| **Due Date** | | M23 (30 November 2019) | |
| **Version No.** | | 0.3 | |
| **Submission Date** | | 29 November 2019 | |
| **File Name** | | D3.4 Code and Data Repositories | |
| **Project Duration** | | 36 Months | |

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 770420.

## Version Control

| Version | Date | Author | Notes |
|---|---|---|---|
| 0.1 | 2 April 2018 | Nesta | Template Creation |
| 0.2 | 26 November 2019 | DTU | First draft |
| 0.3 | 28 November 2019 | DTU | Second draft - Incorporated comments from Nesta, Fraunhofer and COTEC |
| 0.4 | | | |

## Reviewers List

| Version | Date | Reviewers | Notes |
|---|---|---|---|
| 0.2.1 | 28 November 2019 | COTEC | |
| 0.2.2 | | | |

## Disclaimer

This document has been produced with the assistance of the European Union. The contents of this publication are the sole responsibility of the author and can in no way be taken to reflect the views of the European Union.

## Executive Summary

The present report overviews code and documentation that was produced during WP3. The purpose of this document is to ensure transparency and reproducibility of the achieved results. First, the overall data infrastructure of EURITO is presented. Second, the explanation of connections between data infrastructure and developed R&I is given. Finally, code and documentation for the EURITO project are included.

## Table of Contents

## List of Figures



## List of Tables

## 1. Introduction

As the current report concludes Work Package 3 “Data Collection and Analysis”, a quick overview of the previous deliverables in the work package is necessary.

In D3.1 “Design of Data Collection Phase”, requirements for the future EURITO infrastructure for each phase of the data collection were defined: data extraction, ingestion and enrichment.

D3.2 “Quantitative Methods” discusses employed data analysis methods and their relation to scaling up activities of pilots that were generated during WP2.

D3.3 “R&I Performance Indicators” describes derived R&I indicators with respect to datasets and quantitative methods.

Thus, the main purpose of this document is to enable transparency and explaining the process of generating innovation indicators to ensure future reuse of the achieved results. First, the overall development infrastructure of the EURITO project is explained. Second, the linkage between the defined indicators and generated outputs is provided to assist the further validation and refinement of indicators. Finally, supporting technical documentation is provided for the purposes of further reproducibility.

The overall infrastructure of the EURITO project has been organised in the three main modules (Figure 1):

1. **Github code repository**. This repository contains the main codebase for data collection, preprocessing and indicator generation functions, as well as project documentation. This codebase is open source and available for everyone to clone and use further.
2. **Amazon Web Services (AWS) EC2 servers**. The main role of these servers is to run the code from Github code repositories and return data and indicators.
3. **AWS S3 repository**. This repository acts as a storage of generated indicators, database backups and configuration files.

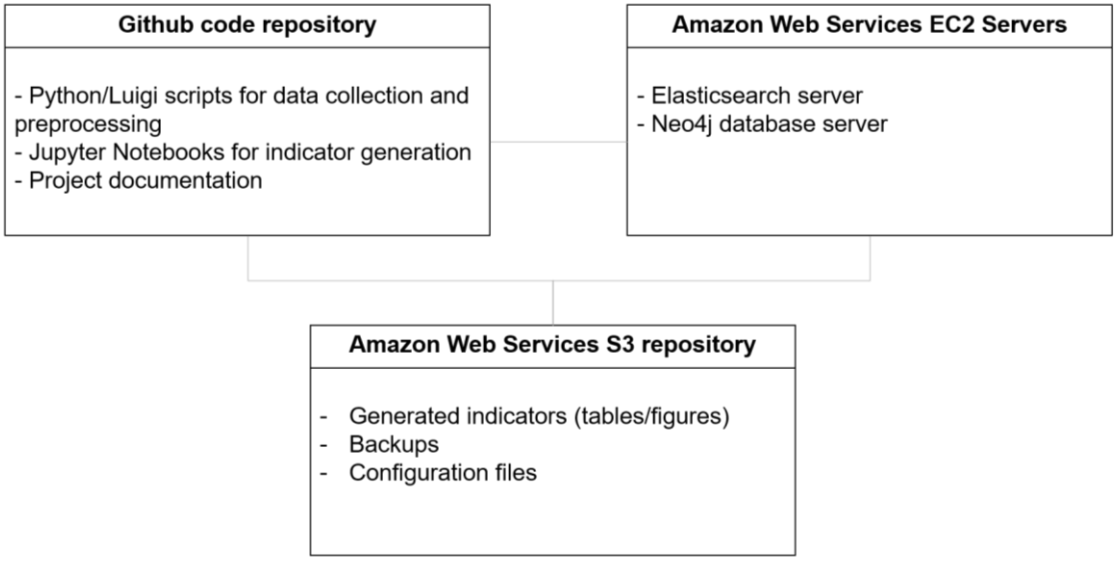


Figure 1. EURITO infrastructure overview

We note that several indicators were redefined due to robustness and infrastructure considerations compared to D3.3. Structural Change indicator due to having a finite number of technological queries. Gender Diversity indicator due to using arXiv and PATSTAT datasets, which do not contain gender information. Theme 4 “Predictive Analytics” will be covered in WP4, as it builds on the results of the current report.

## 2. Data Collection and Preprocessing codebase

### 2.1 Data Collection and Preprocessing setup

Data collection and preprocessing functionality are enabled via a series of Python scripts that can be accessed through the EURITO Github repository at the following link: https://github.com/EURITO/eurito_daps. Execution of the developed Python scripts is managed through Luigi pipelines (https://github.com/spotify/luigi), which allow users to avoid rerunning of time-consuming data collection tasks, when, for instance, only later steps of data preprocessing is needed.

Taking as an example from running a task for collecting and processing OpenAIRE data, the command for collecting software records from the OpenAIRE API looks as follows in a command line:

*luigi --module openaire_to_neo4j_search RootTask --date 2018-04-29 --output-type 'software'*

where *openaire_to_neo4j_search* - name of the Python script
*RootTask* - name of the main function in the script
*date* - date used to label the outputs in the logging database
*output_type*: type of record to be extracted from OpenAIRE API. Accepts "software", "datasets", "publications", "ECProjects"

### 2.2 Datasets

In EURITO we predominantly use the following data sources:

- EU-funded research from CORDIS enriched with OpenAIRE data
- EU-funded research on arXiv
- EU Patents from PATSTAT

**CORDIS and OpenAIRE**

Data from the CORDIS's H2020 API and FP7 API funded projects is extracted using code found in this repository.

In total, 51,250 organisations and 50,640 projects were extracted from the API. There are 1,102 proposal calls, 245,465 publications and 34,507 reports. In total 6,545 are associated with the projects.

Software and dataset outputs are associated with the projects using EC project acronyms, using OpenAIRE API.

All of these entities are then linked together and stored using a neo4j graph database. The code for automatically piping the data in neo4j is provided here.

**arXiv**

All articles from arXiv, which is the world's premier repository of pre-prints of articles in the physical, quantitative and computational sciences, are already automatically collected, geocoded (using GRID) and enriched with topics (using MAG). Articles are assigned to EU NUTS regions (at all levels) using NESTA's nuts-finder python package.
Data is transferred to EURITO's elasticsearch server via NESTA's es2es package. The lolvelty algorithm is then applied to the data in order to generate a novelty metric for each article. This procedure is better described in this blog (see "Defining novelty").
The indicators using this data source are presented in this other EURITO repository.
In total, 1,598,033 articles have been processed, of which 459,371 have authors based in EU nations.

**PATSTAT**

All patents from the PATSTAT service have been collected in NESTA's own database using NESTA's pypatstat library. Since this database is very large, we have selected patents which belong to a patent family with a granted patent first published after the year 2000, with at least one person or organisation (inventor or applicant) based in an EU member state. This leads to 1,552,303 patents in the database.

Data is transferred to EURITO's elasticsearch server via nesta's es2es package. The indicators using this data source are presented in this other EURITO repository.

Full documentation for data collection and preprocessing code is provided in Appendix B and the following link: https://eurito.readthedocs.io/.

## 3. Indicator generation (notebooks)

Once data for indicators is collected and preprocessed, we can generate indicators themselves. The main codebase for indicator generation is provided in the form of reproducible Jupyter Notebooks for each theme (i.e. scale-up) as shown in Appendix A and the following link: https://github.com/EURITO/query_indicators. The generated tables (CSV files) and figures that describe indicators are stored in AWS S3 bucket named "eurito-indicators" and can be accessed using AWS credentials.

Below we explain the indicator generation process for each of the Indicators.

### 3.1 Theme 1 "Emerging Technology and Mapping" Indicators.

This scale-up combines Pilot 1: Emerging Technology Ecosystems (Artificial Intelligence) and Pilot 3: Technological Change Indicators to generate indicators about emerging technology Research and Development (R&D) and its technological innovation system. To achieve this, the level of technological activity is measured by quantifying the relevance of documents in the provided datasets according to a technology query.

#### 1. Level of technological activity (LTA).

Definition. *Level of technological activity (LTA)* is a sum of relevance scores for a set of returned documents for a given technology, where technology is specified as a search query (e.g. "deep learning platforms").

Dataset used: arXiv (PATSTAT may be used as well, which is done by changing the INDEX parameter in the Jupyter Notebook).

The relevance scores are calculated as follows:

Step 1. Feed the technology query Q to ElasticSearch engine, obtain N of seed documents as a result.
Step 2. From the X number of seed documents extract top K keywords
Step 3. Remove seed documents that do not contain these top K keywords
Step 4. For the remaining seed documents, calculate term frequency-inverse document frequency (TF-IDF) centroid.
Step 5. The relevance score of a document is then a TF-IDF similarity from that document to the TF-IDF centroid. This means that being closer to the centroid signifies a higher similarity of that document to the "average" of most relevant documents.

TF-IDF centroid and similarity distance from centroid is calculated as described here.

Parameters Q, N, X and K are specified by the user and explained in the guide to the relevance scoring algorithm of Clio, a search engine that retrieves these relevant

documents: https://github.com/nestauk/clio-lite/blob/master/README.md#a-note-on-relevance-scoring

Due to the exploratory nature of the EURITO project, we decided to have a finite set of technology queries as opposed to a search engine-like functionality. Therefore, in the AWS S3 repository, under the *eurito-indicators/tables/theme_1/ai_activity* folder we generated a set of files with indicator values for predefined technology queries in AI in the respective folders as follows:

*"Natural Language Generation",*
*"Speech recognition",*
*"Virtual Agents",*
*"Machine Learning Platforms",*
*"AI-Optimized Hardware",*
*"Decision Management AI",*
*"Deep Learning Platforms",*
*"Biometrics AI",*
*"Robotic Processes Automation AI",*
*"Natural Language Processing",*
*"Digital Twin AI",*
*"Cyber Defense AI",*
*"Compliance AI",*
*"Knowledge Worker Aid AI",*
*"Content Creation AI",*
*"Peer to Peer Networks AI",*
*"Emotion Recognition AI",*
*"Image Recognition AI",*
*"Marketing Automation AI".*

For each specified technology query, a CSV file with indicator values is generated. An example of such a file containing the total relevance score for the technology query "*AI-Optimized Hardware*" is presented in Table 1:

Table 1. Levels of technological activity for technology query "*AI-Optimized Hardware*"

| | **2014** | **2015** | **2016** | **2017** | **2018** | **2019** |
|---|---|---|---|---|---|---|
| Austria | 0.066 | 0.053 | 0.062 | 0.083 | 0.107 | 0.063 |
| Belgium | 0.068 | 0.104 | 0.163 | 0.188 | 0.244 | 0.145 |
| Bulgaria | 0.000 | 0.001 | 0.000 | 0.006 | 0.000 | 0.000 |
| Croatia | 0.013 | 0.014 | 0.003 | 0.013 | 0.008 | 0.005 |
| Cyprus | 0.005 | 0.001 | 0.003 | 0.001 | 0.008 | 0.007 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Czech Republic | 0.028 | 0.057 | 0.082 | 0.122 | 0.164 | 0.085 |
| Denmark | 0.060 | 0.114 | 0.168 | 0.186 | 0.258 | 0.127 |
| Estonia | 0.001 | 0.003 | 0.003 | 0.002 | 0.000 | 0.004 |
| Finland | 0.048 | 0.068 | 0.105 | 0.112 | 0.130 | 0.063 |
| France | 0.105 | 0.170 | 0.292 | 0.292 | 0.348 | 0.226 |
| Germany | 0.217 | 0.274 | 0.307 | 0.563 | 0.583 | 0.394 |
| Gibraltar | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Greece | 0.021 | 0.038 | 0.027 | 0.025 | 0.033 | 0.037 |
| Hungary | 0.016 | 0.030 | 0.007 | 0.020 | 0.014 | 0.003 |
| Ireland | 0.079 | 0.114 | 0.228 | 0.313 | 0.446 | 0.264 |
| Italy | 0.083 | 0.134 | 0.177 | 0.305 | 0.352 | 0.177 |
| Latvia | 0.000 | 0.004 | 0.002 | 0.000 | 0.000 | 0.001 |
| Lithuania | 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.003 |
| Luxembourg | 0.007 | 0.004 | 0.001 | 0.007 | 0.005 | 0.010 |
| Malta | 0.003 | 0.000 | 0.000 | 0.001 | 0.008 | 0.001 |
| Netherlands | 0.089 | 0.139 | 0.218 | 0.236 | 0.285 | 0.197 |
| Poland | 0.020 | 0.044 | 0.081 | 0.101 | 0.104 | 0.064 |
| Portugal | 0.043 | 0.087 | 0.146 | 0.163 | 0.206 | 0.129 |
| Romania | 0.003 | 0.005 | 0.006 | 0.012 | 0.027 | 0.012 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Slovakia | 0.002 | 0.003 | 0.004 | 0.003 | 0.004 | 0.000 |
| Slovenia | 0.003 | 0.000 | 0.011 | 0.003 | 0.010 | 0.002 |
| Spain | 0.065 | 0.067 | 0.084 | 0.110 | 0.169 | 0.071 |
| Sweden | 0.058 | 0.090 | 0.057 | 0.097 | 0.133 | 0.065 |
| Great Britain | 0.230 | 0.405 | 0.569 | 0.710 | 1.000 | 0.559 |

As seen from the table, UK has the maximum number of highly relevant documents in *AI-Optimized Hardware* compared to the rest of EU countries in the years 2017-2019. In other words, the indicator takes into account not only the sheer number of documents, but also their relevancy to the search query.

Similar files are generated on regional levels, where region names are provided according to NUTS classification available here: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:02003R1059-20180118&from=EN

Similarly to the LTA indicator, files with values for indicators 2-4 for each technological query are located at *eurito-indicators/tables/theme_1/ai_activity /"technological_query_name"/* folder.

2. Average activity by country (AA)

Definition: Average activity by country is a mean relevance score for each country, which is calculated by dividing the total relevance score by the total number of relevant documents. A corrected version of this indicator subtracts a Poisson error from the mean value.æ

3. Trajectory of technological activity (TTA)

Definition: Trajectory of technological activity is defined as a linear coefficient of total relevance score for a specified technological query with respect to previous time periods. A corrected version of this indicator subtracts upper and lower Poisson errors from the linear coefficient values.

4. Concentration of technological activity (CTA)

Definition: Concentration of technological activity (CTA) is a share of total relevance score of a specified country within a sum of total relevance scores for all countries for a given technological query for a given time period.

CTA = LTA(country) / sum of LTAs(all countries)

### 3.2 Theme 2 "New Research Funding Analytics" Indicators.

Datasets: CORDIS, OpenAIRE

First, we have loaded CORDIS dataset into Neo4j graph database. In the graph, objects are stored as a network of nodes and relationships between them. For instance, in

the CORDIS Neo4j database, projects funded by the European Commission are stored as nodes of type “Project” with various meta-attributes (e.g. project acronym, description, funding amount, etc.) and relationships to other nodes (e.g. a project might have a relationship of type “HAS_SOFTWARE” with a node of type “Software”).

Nodes of type "Project" have the following meta-attributes:

*acronym* - project acronym
*betw* - project centrality
*ec_contribution* - funding by EC, mio EUR
*start_date_code, end_date_code* - project start and end date
*framework, funded_under, funding_scheme* - funding framework and program
*grant_num* - 6 digit grant number
*objective* - project objective
*project_description* - project description
*rcn* - project identificator
*status* - project status (e.g. closed, ongoing)
*total_cost* - total budget, mio EUR
*website* - project website

Nodes of type "Organisation" have the following meta-attributes:
*name* - organisation name
*betw* - organisation centrality
*country_code* - 2-letter country code
*country_name* - country name

Second, based on the constructed network, for each node, we have calculated the indicators of project and organisation centrality. Finally, these values were written as a meta-attributes to the respective nodes in the Neo4j database.

Both indicators are based on betweenness centrality, which measures a number of times when the shortest paths between all the nodes in the network pass through a particular node. In essence, betweenness centrality estimates the node’s ability to serve as a “bridge” that connects different network parts. Higher betweenness centrality aims to signify “important” projects and organisations in the R&I ecosystem.

### 5. Project centrality (PC).

Definition: Project centrality is the betweenness centrality of a node in the R&I network that represents a research project, where R&I network refers to a network of research projects, organisations and research outputs linked together.

Table 2 presents a list of top 15 nodes according to project centrality:

Table 2. Top 15 nodes according to project centrality

| **Acronym** | **Project Centrality** | **EC contribution** | **Framework Program** |
|---|---|---|---|
| GrapheneCore1 | 3074818 | EUR 89,000,000 | H2020 |
| GRAPHENE | 2729211 | EUR 54,000,000 | FP7 |
| EGI-InSPIRE | 2452014 | EUR 25,000,000 | FP7 |
| GrapheneCore2 | 2069696 | EUR 88,000,000 | H2020 |

| | | | |
|---|---|---|---|
| HBP | 1782999 | EUR 54,000,000 | FP7 |
| HBP SGA1 | 1755587 | EUR 89,000,000 | H2020 |
| EGEE-III | 1695839 | EUR 32,000,000 | FP7 |
| HBP SGA2 | 1581863 | EUR 88,000,000 | H2020 |
| ELIXIR-EXCELERATE | 1228839 | EUR 19,051,482 | H2020 |
| ECHORD | 1174091 | EUR 18,969,760 | FP7 |
| PAST4FUTURE | 1079601 | EUR 6,647,909 | FP7 |
| AIDA-2020 | 1039285 | EUR 10,000,000 | H2020 |
| EVIMALAR | 1024754 | EUR 12,000,000 | FP7 |
| NANOSIL | 1014953 | EUR 4,300,000 | FP7 |
| PRACE-2IP | 974906.7 | EUR 18,000,000 | FP7 |

As seen from the Table 2, two of the largest research projects in the European Commission, Graphene projects when cover over 150 organisations and 23 countries. In another example, the Human Brain Project (acronym “HBP”) is a ten-year project that started in 2013 and spans more than 100 research organisations (https://www.humanbrainproject.eu/en/about/overview/).

### 6. Organisation centrality (OC)

Definition: Organisation centrality is the betweenness centrality of a node in the R&I network that represents a research organisation, where R&I network refers to a network of research projects, organisations and research outputs linked together.

Table 3 presents a list of top 15 nodes according to organisation centrality:

Table 3. Top 15 nodes according to organisation centrality

| **Organisation centrality** | **Country name** | **Organisation name** |
|---|---|---|
| 114674344.9 | France | CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE CNRS |
| 48472324.52 | Germany | FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V. |
| 27941037.55 | Spain | AGENCIA ESTATAL CONSEJO SUPERIOR DEINVESTIGACIONES CIENTIFICAS |
| 25759598.73 | Italy | CONSIGLIO NAZIONALE DELLE RICERCHE |
| 25258853.9 | France | COMMISSARIAT A L ENERGIE ATOMIQUE ET AUX |

| | | |
|---|---|---|
| | | ENERGIES ALTERNATIVES |
| 23797375.5 | United Kingdom | THE CHANCELLOR MASTERS AND SCHOLARS OF THE UNIVERSITY OF CAMBRIDGE |
| 23357051.39 | United Kingdom | THE CHANCELLOR, MASTERS AND SCHOLARS OF THE UNIVERSITY OF OXFORD |
| 16548974.34 | United Kingdom | IMPERIAL COLLEGE OF SCIENCE TECHNOLOGY AND MEDICINE |
| 15721581.55 | Switzerland | EIDGENOESSISCHE TECHNISCHE HOCHSCHULE ZUERICH |
| 15070285.17 | Belgium | KATHOLIEKE UNIVERSITEIT LEUVEN |
| 13805642.88 | Germany | MAX-PLANCK-GESELLSCHAFT ZUR FORDERUNG DER WISSENSCHAFTEN EV |
| 10625659.35 | Denmark | KOBENHAVNS UNIVERSITET |
| 10435979.53 | Switzerland | ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE |
| 9158880.711 | United Kingdom | THE UNIVERSITY OF EDINBURGH |
| 8711778.707 | Denmark | DANMARKS TEKNISKE UNIVERSITET |

As seen from the Table 3, organisations with the largest centrality correspond to the top research organisations in Europe. For instance, CNRS (France) is the largest fundamental research agency in Europe, while Fraunhofer (Germany) is the largest applied research agency.

### 3.3 Theme 3 “Inclusive and Mission-Oriented R&I” Indicators.

#### 7. Levels of mission activity (LMA)

Definition: Level of activity of a mission is defined as a number of term occurrences in the dataset. Mission terms are extracted from a search query specified by the user.

Granularity: per specified mission, per data source, per country or region (e.g. Europe, Asia).

Calculation of mission activity levels is based on the same mechanisms as in Theme 1 indicators. For the purposes of the scale-up the following mission statements were used as an input mission query:
*“Adaptation to climate change, including societal transformation”*
*“Cancer”*
*“Climate-neutral and smart cities”*
*“Soil health and food”*

As an example, generated levels of mission activity for mission “*Adaptation to climate change, including societal transformation*” are presented in Table 4:

Table 4. Levels of mission activity for mission "*Adaptation to climate change, including societal transformation*"

| | 2014 | 2015 | 2016 | 2017 | 2018 | 2019 |
|---|---|---|---|---|---|---|
| Austria | 0.158 | 0.188 | 0.175 | 0.154 | 0.095 | 0.056 |
| Belgium | 0.137 | 0.187 | 0.192 | 0.227 | 0.158 | 0.123 |
| Bulgaria | 0.011 | 0.012 | 0.014 | 0.023 | 0.013 | 0.005 |
| Croatia | 0.033 | 0.024 | 0.009 | 0.009 | 0.014 | 0.004 |
| Cyprus | 0.007 | 0.010 | 0.004 | 0.002 | 0.010 | 0.000 |
| Czech Republic | 0.067 | 0.105 | 0.089 | 0.143 | 0.092 | 0.072 |
| Denmark | 0.176 | 0.206 | 0.207 | 0.270 | 0.149 | 0.111 |
| Estonia | 0.010 | 0.014 | 0.010 | 0.009 | 0.012 | 0.006 |
| Faroe Islands | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Finland | 0.117 | 0.162 | 0.153 | 0.132 | 0.093 | 0.067 |
| France | 0.586 | 0.561 | 0.569 | 0.516 | 0.341 | 0.195 |
| Germany | 0.937 | 0.951 | 0.901 | 0.860 | 0.607 | 0.347 |
| Gibraltar | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| Greece | 0.061 | 0.051 | 0.049 | 0.074 | 0.032 | 0.039 |
| Hungary | 0.082 | 0.044 | 0.063 | 0.049 | 0.025 | 0.017 |
| Ireland | 0.132 | 0.175 | 0.196 | 0.239 | 0.175 | 0.162 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Italy | 0.409 | 0.386 | 0.415 | 0.427 | 0.461 | 0.229 |
| Latvia | 0.001 | 0.000 | 0.000 | 0.006 | 0.002 | 0.000 |
| Lithuania | 0.002 | 0.003 | 0.002 | 0.009 | 0.003 | 0.000 |
| Luxembourg | 0.008 | 0.008 | 0.015 | 0.009 | 0.010 | 0.008 |
| Malta | 0.002 | 0.000 | 0.000 | 0.011 | 0.004 | 0.000 |
| Netherlands | 0.301 | 0.363 | 0.352 | 0.378 | 0.279 | 0.161 |
| Poland | 0.111 | 0.123 | 0.128 | 0.089 | 0.102 | 0.037 |
| Portugal | 0.115 | 0.135 | 0.167 | 0.190 | 0.118 | 0.099 |
| Romania | 0.014 | 0.014 | 0.016 | 0.016 | 0.016 | 0.009 |
| Slovakia | 0.032 | 0.030 | 0.024 | 0.012 | 0.021 | 0.005 |
| Slovenia | 0.031 | 0.017 | 0.024 | 0.027 | 0.024 | 0.006 |
| Spain | 0.349 | 0.305 | 0.331 | 0.286 | 0.254 | 0.079 |
| Sweden | 0.149 | 0.154 | 0.168 | 0.136 | 0.125 | 0.059 |
| Great Britain | 0.907 | 1.000 | 0.952 | 0.951 | 0.776 | 0.463 |

As seen from Table 4, Great Britain and Germany are countries with the highest mission activity levels with Spain, Netherlands, France, Italy and Denmark after them.

## Appendix A - Jupyter Notebooks for indicator generation

### Theme 1 Jupyter Notebook

https://github.com/EURITO/query_indicators/blob/master/theme_1/ai_activity/national_ai_activity.ipynb

In [1]:

```
%matplotlib inline
```

In [2]:

```
import os
import sys
sys.path.append(os.path.abspath('../../'))
from query_indicators import generate_save_path
from query_indicators import get_eu_countries
```

In [3]:

```
import boto3
from collections import defaultdict
from clio_lite import clio_search, clio_search_iter
import io
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.collections import PatchCollection
from matplotlib.patches import Rectangle
import numpy as np
import pandas as pd
```

In [4]:

```
# Env variables
mpl.rcParams['hatch.linewidth'] = 0.2
mpl.rcParams['font.size'] = 18
mpl.rcParams['image.cmap'] = 'Pastel1'
os.environ['AWS_SHARED_CREDENTIALS_FILE'] = '/Users/jklinger/EURITO-AWS/.aws/credentials'  # <--- Note: NOT nesta's AWS credentials
```

In [5]:

```
# Some globals
URL = "https://search-eurito-prod-bbyn72q2rhx4ifj6h5dom43uhy.eu-west-1.es.amazonaws.com/"
INDEX = "arxiv_v0"
FIELDS = ['terms_tokens_entity', 'textBody_abstract_article']
EU_COUNTRIES = get_eu_countries()
COLORS = plt.get_cmap('Set2').colors
COLOR_MAP = 'Pastel1'
S3 = boto3.resource('s3')
SAVE_PATH = generate_save_path()  # EURITO collaborators: this is generated assuming you have stuck to the convention 'theme_x/something/something_else.ipynb'
BUCKET = 'eurito-indicators'  # EURITO collaborators: please don't change this
SAVE_RESULTS = True  # Set this to "False" when you want to view figures inline. When "True", results will be saved to S3.
```

```
if SAVE_RESULTS:
    plt.ioff()  # <--- for turning off visible figs
else:
    plt.ion()
```

In [6]:

```
def make_search(query, max_query_terms, yr0=2014, yr1=2019, countries=EU_COUNTRIES,
window=1):
    """
    Retrieve count and score data for a given basic clio search.

    Args:
        query (str): Seed query for clio.
        max_query_terms (list): Triple of max_query_terms (low, middle, high) to use from the initial
query.
        yr0 (int): Start year in range to use in filter.
        yr1 (int): Final year in range to use in filter.
        countries (list): A list of countries to filter (default to all EU).
        window (int): The number of years to consider in between time windows. Note that changing
this will lead to double-counting.
    Returns:
        data (dict): {max_query_terms --> [{year --> sum_score} for each country]}
        all_scores (dict): {max_query_terms --> {country --> [score for doc in docs] } }
    """
    top_doc = None
    _data = defaultdict(lambda: defaultdict(dict))  # {max_query_terms --> {year --> {country -->
score} } }
    all_scores = defaultdict(lambda: defaultdict(list))  # {max_query_terms --> {country --> [score
for doc in docs] } }
    for n in max_query_terms:
        # Set the order of the countries
        for ctry in EU_COUNTRIES:
            _data[n][ctry]
            all_scores[n][ctry]
        # Iterate over years
        for yr in range(yr0, yr1+1):
            # Set default values for countries
            for ctry in EU_COUNTRIES:
                _data[n][ctry][yr] = 0
            # Iterate over docs
            filters = [{"range":{"year_of_article":{"gte":yr, "lt":yr+window}}}]
            for doc in clio_search_iter(url=URL, index=INDEX, query=query, fields=FIELDS,
                                        max_query_terms=n, post_filters=filters, chunksize=5000):
                if '_score' not in doc or doc['terms_countries_article'] is None:
                    continue
                score = doc['_score']
                for ctry in filter(lambda x: x in countries, doc['terms_countries_article']):
                    if top_doc is None:
                        top_doc = doc
                    all_scores[n][ctry].append(score)
                    _data[n][ctry][yr] += score
    # Reformat data as {max_query_terms --> [{year --> score} for each country in order]}
    data = {}
    for n, ctry_data in _data.items():
        data[n] = []
        for ctry, yr_data in ctry_data.items():
```

```
            data[n].append(yr_data)
    return top_doc, data, all_scores
```

# Indicator calculations

Each of these functions is assumed to take the form

```
def _an_indicator_calulation(data, year=None, _max=1):
    """
    A function calculating an indicator.

    Args:
        data (list): Rows of data
        year (int): A year to consider, if applicable.
        _max (int): Divide by this to normalise your results. This is automatically applied in
:obj:`make_activity_plot`
    Returns:
        result (list) A list of indicators to plot. The length of the list is assumed to be equal to the
number of countries.
    """
    # Calculate something
```

In [7]:

```
def _total_activity_by_country(data, year=None, _max=1):
    """
    Indicator: Sum of relevance scores, by year (if specified) or in total.
    """
    if year is None:
        scores = [sum(row.values())/_max for row in data]
    else:
        scores = [row[year]/_max for row in data]
    return scores


def _average_activity_by_country(data, year=None, _max=1):
    """
    Indicator: Mean relevance score. This function is basically a lambda, since it assumes the
average has already been calculated.
    """
    return [row/_max for row in data]


def _corrected_average_activity_by_country(data, year=None, _max=1):
    """
    Indicator: Mean relevance score minus it's (very) approximate Poisson error.
    """
    return [(row - np.sqrt(row))/_max for row in data]


def _linear_coeffs(years, scores, _max):
    """Calculates linear coefficients for scores wrt years"""
    return [np.polyfit(_scores, _years, 1)[0]/_max
            if all(v > 0 for v in _scores) else 0
            for _years, _scores in zip(years, scores)]
```

```
def _trajectory(data, year=None, _max=1):
    """
    Indicator: Linear coefficient of total relevance score wrt year
    """
    years = [list(row.keys()) for row in data]
    scores = [list(row.values()) for row in data]
    return _linear_coeffs(years, scores, _max)


def _corrected_trajectory(data, year=None, _max=1):
    """
    Indicator: Linear coefficient of upper and lower limits of relevance score wrt year
    """
    # Reformulate the data in terms of upper and lower bounds
    years, scores = [], []
    for row in data:
        _years, _scores = [], []
        for k, v in row.items():
            _years += [k,k]
            _scores += [v - np.sqrt(v), v + np.sqrt(v)]  # Estimate upper and lower limits with very
approximate Poisson errors
        years.append(_years)
        scores.append(_scores)
    return _linear_coeffs(years, scores, _max)
```

# Plotting functionality

In [8]:

```
class _Sorter:
    def __init__(self, values, topn=None):
        if topn is None:
            topn = len(values)
        self.indices = list(np.argsort(values))[-topn:]  # Argsort is ascending, so -ve indexing to pick
up topn
    def sort(self, x):
        """Sort list x by indices"""
        return [x[i] for i in self.indices]


def _s3_savefig(query, fig_name, extension='png'):
    """Save the figure to s3. The figure is grabbed from the global scope."""
    if not SAVE_RESULTS:
        return
    outname = (f'figures/{SAVE_PATH}/'
               f'{query.replace(" ","_").lower()}'
               f'/{fig_name.replace(" ","_").lower()}'
               f'.{extension}')
    with io.BytesIO() as f:
        plt.savefig(f, bbox_inches='tight', format=extension, pad_inches=0)
        obj = S3.Object(BUCKET, outname)
        f.seek(0)
        obj.put(Body=f)
```

```
def _s3_savetable(data, key, index, object_path, transformer=lambda x: x):
    """Upload the table to s3"""
    if not SAVE_RESULTS:
        return
    df = pd.DataFrame(transformer(data[key]), index=index)
    if len(df.columns) == 1:
        df.columns = ['value']
    df = df / df.max().max()
    table_data = df.to_csv().encode()
    obj = S3.Object(BUCKET, os.path.join(f'tables/{SAVE_PATH}', object_path))
    obj.put(Body=table_data)


def make_activity_plot(f, data, countries, max_query_terms, query,
                       year=None, label=None, x_padding=0.5, y_padding=0.05, xlabel_fontsize=14):
    """
    Make a query and generate indicators by country, saving the plots to S3 and saving the rawest data
    to tables on S3.


    Args:
        f: An indicator function, as described in the 'Indicator calculations' section.
        data (dict): {max_query_terms --> [{year --> sum_score} for each country]}
        countries (list): A list of EU ISO-2 codes
        max_query_terms (list): Triple of max_query_terms for clio, corresponding to low, middle and high values of
                                max_query_terms to test robustness of the query.
        query (str): query used to generate this data.
        year (int): Year to generate the indicator for (if applicable).
        label (str): label for annotating the plot.
        {x,y}_padding (float): Aesthetic padding around the extreme limits of the {x,y} axis.
        xlabel_fontsize (int): Fontsize of the x labels (country ISO-2 codes).
    """
    # Calculate the indicator for each value of n, then recalculate the normalised indicator
    _, middle, _ = (f(data[n], year=year) for n in max_query_terms)
    low, middle, high = (f(data[n], year=year, _max=max(middle)) for n in max_query_terms)
    indicator = [np.median([a, b, c]) for a, b, c in zip(low, middle, high)]

    # Sort all data by indicator value
    s = _Sorter(indicator)
    countries = s.sort(countries)
    low = s.sort(low)
    middle = s.sort(middle)
    high =  s.sort(high)
    indicator = s.sort(indicator)

    # Make the scatter plot
    fig, ax = plt.subplots(figsize=(15, 6))
    make_error_boxes(ax, low, middle, high)  # Draw the bounding box
    ax.scatter(countries, indicator,  s=0, marker='o', color='black')  # Draw the centre mark
    ax.set_title(f'{label}\nQuery: "{query}"')
    ax.set_ylabel(label)

    # Set limits and formulate
    y0 = min(low+middle+high)
    y1 = max(low+middle+high)
```

```
    if -y1*y_padding < y0:
        y0 = -y1*y_padding
    else:  # In case of negative values
        y0 = y0 - np.abs(y0*y_padding)
    ax.set_ylim(y0, y1*(1+y_padding))
    ax.set_xlim(-x_padding, len(countries)-x_padding)
    for tick in ax.xaxis.get_major_ticks():
        tick.label.set_fontsize(xlabel_fontsize)

    # Save to s3 & return
    _s3_savefig(query, label)
    return ax


def make_error_boxes(ax, low, middle, high, facecolor='r',
                     edgecolor='None', alpha=0.5):
    """
    Generate outer rectangles based on three values, and draw a horizontal line through the
middle of the rectangle.
    No assumption is made on the order of values, so don't worry if they're not properly ordered.

    Args:
        ax (matplotlib.axis): An axis to add patches to.
        {low, middle, high} (list): Three concurrent lists of values from which to calculate the
rectangle limits.
        {facecolor, edgecolor} (str): The {face,edge} colour of the rectangles.
        alpha (float): The alpha of the rectangles.
    """
    # Generate the rectangle
    errorboxes = []
    middlelines = []
    for x, ys in enumerate(zip(low, middle, high)):
        rect = Rectangle((x - 0.45, min(ys)), 0.9, max(ys) - min(ys))
        line = Rectangle((x - 0.45, np.median(ys)), 0.9, 0)
        errorboxes.append(rect)
        middlelines.append(line)

    # Create patch collection with specified colour/alpha
    pc = PatchCollection(errorboxes, facecolor=facecolor, alpha=alpha, edgecolor=edgecolor,
hatch='/')
    lc = PatchCollection(middlelines, facecolor='black', alpha=0.9, edgecolor='black')

    # Add collection to axes
    ax.add_collection(pc)
    ax.add_collection(lc)


def stacked_scores(all_scores, query, topn=8,
                   low_bins=[10**i for i in np.arange(0, 1.1, 0.025)],
                   high_bins=[10**i for i in np.arange(1.1, 2.5, 0.05)],
                   x_scale='log', label='Relevance score breakdown',
                   xlabel='Relevance score', ylabel='Number of relevant documents',
                   legend_fontsize='small', legend_cols=2):
    """
    Create stacked histogram of document scores by country. Two sets of bins are used,
    in order to have a more legible binning scale.
```

```
    Args:
        all_scores (dict): {max_query_terms --> {country --> [score for doc in docs] } }
        query (str): query used to generate this data.
        low_bins (list): List of initial bin edges.
        high_bins (list): List of supplementary bin edges. These could have a different spacing
scheme to the lower bin edges.
        x_scale (str): Argument for `ax.set_xscale`.
        label (str): label for annotating the plot.
        {x,y}_label (str): Argument for `ax.set_{x,y}label`.
        legend_fontsize (str): Argument for legend fontsize.
        legend_cols (str): Argument for legend ncol.
    """

    # Sort countries and scores by the sum of scores by country
    countries = list(all_scores.keys())
    scores = list(all_scores.values())
    s = _Sorter([sum(v) for v in scores], topn=topn)
    scores = s.sort(scores)
    countries = s.sort(countries)

    # Plot the stacked scores
    fig, ax = plt.subplots(figsize=(10, 6))
    plt.set_cmap(COLOR_MAP)
    ax.hist(scores, bins=low_bins+high_bins, stacked=True,
            label=countries, color=COLORS[:len(scores)])

    # Prettify the plot
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.legend(fontsize=legend_fontsize, ncol=legend_cols)
    ax.set_xlim(low_bins[0], None)
    ax.set_xscale(x_scale)
    ax.set_title(f'{label}\nQuery: "{query}"')

    # Save to s3
    _s3_savefig(query, label)
    return ax
```

# Bringing it all together

In [9]:

```
def generate_indicator(q, max_query_terms=[7, 10, 13], countries=EU_COUNTRIES, *args,
**kwargs):
    """
    Make a query and generate indicators by country, saving the plots to S3 and saving the rawest
data
    to tables on S3.


    Args:
        q (str): The query to Elasticsearch
        max_query_terms (list): Triple of max_query_terms for clio, corresponding to low, middle
and high values of
                                max_query_terms to test robustness of the query.
        countries (list): A list of EU ISO-2 codes
    Returns:
```

```
        top_doc (dict): The highest ranking document from the search.
        data (dict): {max_query_terms --> [{year --> sum_score} for each country]}
        all_scores (dict): {max_query_terms --> {country --> [score for doc in docs] } }
    """

    # Make the search and retrieve scores by country, and the highest ranking doc
    example_doc, data, all_scores = make_search(q, max_query_terms=max_query_terms,
countries=countries, *args, **kwargs)

    # Reformat the scores to calculate the average
    avg_scores = defaultdict(list)
    for ctry in countries:
        for n, _scores in all_scores.items():
            mean = np.mean(_scores[ctry]) if len(_scores[ctry]) > 0 else 0
            avg_scores[n].append(mean)

    plot_kwargs = dict(countries=countries, max_query_terms=max_query_terms, query=q)
    # Calculate loads of indicators and save the plots
    _ = make_activity_plot(_total_activity_by_country, data, label='Total relevance score',
**plot_kwargs)
    _ = make_activity_plot(_average_activity_by_country, avg_scores, label='Average relevance',
**plot_kwargs)
    _ = make_activity_plot(_corrected_average_activity_by_country, avg_scores, label='Corrected
average relevance',  **plot_kwargs)
    _ = make_activity_plot(_trajectory, data, label='Trajectory', **plot_kwargs)
    _ = make_activity_plot(_corrected_trajectory, data, label='Corrected trajectory', **plot_kwargs)
    _ = stacked_scores(all_scores[max_query_terms[1]], query=q)

    # Save the basic raw data as tables. Note: not as rich as the plotted data.
    _q = q.replace(" ","_").lower()
    _s3_savetable(data, max_query_terms[1], index=countries,
object_path=f'{_q}/total_relevance.csv')
    _s3_savetable(avg_scores, max_query_terms[1], index=countries,
object_path=f'{_q}/avg_relevance.csv')
    _s3_savetable(data, max_query_terms[1], transformer=_trajectory, index=countries,
object_path=f'{_q}/trajectory.csv')

    plt.close('all')  # Clean up the memory cache (unbelievable that matplotlib doesn't do this)
    return example_doc, data, all_scores
```

# Iterate over queries

In [10]:

```
for term in ["Natural Language Generation",
             "Speech recognition",
             "Virtual Agents",
             "Machine Learning Platforms",
             "AI-Optimized Hardware",
             "Decision Management AI",
             "Deep Learning Platforms",
             "Biometrics AI",
             "Robotic Processes Automation AI",
             "Natural Language Processing",
             "Digital Twin AI",
             "Cyber Defense AI",
             "Compliance AI",
```

```
        "Knowledge Worker Aid AI",
        "Content Creation AI",
        "Peer to Peer Networks AI",
        "Emotion Recognition AI",
        "Image Recognition AI",
        "Marketing Automation AI"]:
    print(term)
    print("-"*len(term))
    top_doc, data, all_scores = generate_indicator(term)
    print(top_doc['title_of_article'], ",", top_doc['year_of_article'])
    print(top_doc['terms_countries_article'])
    print(top_doc['textBody_abstract_article'])
    print("\n==============================\n")
```

Natural Language Generation
---
A Deep Architecture for Semantic Parsing , 2014
['CA', 'GB']
Many successful approaches to semantic parsing build on top of the syntactic analysis of text, and make use of distributional representations or statistical models to match parses to ontology-specific queries. This paper presents a novel deep learning architecture which provides a semantic parsing system through the union of two neural models of language semantics. It allows for the generation of ontology-specific queries from natural language statements and questions without the need for parsing, which makes it especially suitable to grammatically malformed or syntactically atypical text, such as tweets, as well as permitting the development of semantic parsers for resource-poor languages.

==========

Speech recognition
---
Spatial Diffuseness Features for DNN-Based Speech Recognition in Noisy and Reverberant Environments , 2014
['DE']
We propose a spatial diffuseness feature for deep neural network (DNN)-based automatic speech recognition to improve recognition accuracy in reverberant and noisy environments. The feature is computed in real-time from multiple microphone signals without requiring knowledge or estimation of the direction of arrival, and represents the relative amount of diffuse noise in each time and frequency bin. It is shown that using the diffuseness feature as an additional input to a DNN-based acoustic model leads to a reduced word error rate for the REVERB challenge corpus, both compared to logmelspec features extracted from noisy signals, and features enhanced by spectral subtraction.

==========

Virtual Agents
---
Expressing social attitudes in virtual agents for social training games , 2014
['FR']
The use of virtual agents in social coaching has increased rapidly in the last decade. In order to train the user in different situations than can occur in real life, the virtual agent should be able to express different social attitudes. In this paper, we propose a model of social attitudes that enables a virtual agent to reason on the appropriate social attitude to express during

the interaction with a user given the course of the interaction, but also the emotions, mood and personality of the agent. Moreover, the model enables the virtual agent to display its social attitude through its non-verbal behaviour.
The proposed model has been developed in the context of job interview simulation. The methodology used to develop such a model combined a theoretical and an empirical approach. Indeed, the model is based both on the literature in Human and Social Sciences on social attitudes but also on the analysis of an audiovisual corpus of job interviews and on post-hoc interviews with the recruiters on their expressed attitudes during the job interview.

==========

Machine Learning Platforms
---
Open science in machine learning , 2014
['NL', 'DE', 'AU']
We present OpenML and mldata, open science platforms that provides easy access to machine learning data, software and results to encourage further study and application. They go beyond the more traditional repositories for data sets and software packages in that they allow researchers to also easily share the results they obtained in experiments and to compare their solutions with those of others.

==========

AI-Optimized Hardware
---
Hands-on experiments on intelligent behavior for mobile robots , 2014
['DE', 'MX']
In recent years, Artificial Intelligence techniques have emerged as useful tools for solving various engineering problems that were not possible or convenient to handle by traditional methods. AI has directly influenced many areas of computer science and becomes an important part of the engineering curriculum. However, determining the important topics for a single semester AI course is a nontrivial task, given the lack of a general methodology. AI concepts commonly overlap with many other disciplines involving a wide range of subjects, including applied approaches to more formal mathematical issues. This paper presents the use of a simple robotic platform to assist the learning of basic AI concepts. The study is guided through some simple experiments using autonomous mobile robots. The central algorithm is the Learning Automata. Using LA, each robot action is applied to an environment to be evaluated by means of a fitness value. The response of the environment is used by the automata to select its next action. This procedure holds until the goal task is reached.
The proposal addresses the AI study by offering in LA a unifying context to draw together several of the topics of AI and motivating the students to learn by building some hands on laboratory exercises. The presented material has been successfully tested as AI teaching aide in the University of Guadalajara robotics group as it motivates students and increases enrolment and retention while educating better computer engineers.

==========

Decision Management AI
---
Hands-on experiments on intelligent behavior for mobile robots , 2014
['DE', 'MX']
In recent years, Artificial Intelligence techniques have emerged as useful

tools for solving various engineering problems that were not possible or convenient to handle by traditional methods. AI has directly influenced many areas of computer science and becomes an important part of the engineering curriculum. However, determining the important topics for a single semester AI course is a nontrivial task, given the lack of a general methodology. AI concepts commonly overlap with many other disciplines involving a wide range of subjects, including applied approaches to more formal mathematical issues. This paper presents the use of a simple robotic platform to assist the learning of basic AI concepts. The study is guided through some simple experiments using autonomous mobile robots. The central algorithm is the Learning Automata. Using LA, each robot action is applied to an environment to be evaluated by means of a fitness value. The response of the environment is used by the automata to select its next action. This procedure holds until the goal task is reached. The proposal addresses the AI study by offering in LA a unifying context to draw together several of the topics of AI and motivating the students to learn by building some hands on laboratory exercises. The presented material has been successfully tested as AI teaching aide in the University of Guadalajara robotics group as it motivates students and increases enrolment and retention while educating better computer engineers.

==========

Deep Learning Platforms
---
Caffe: Convolutional Architecture for Fast Feature Embedding , 2014
['US', 'CH', 'CA', 'IE', 'GB']
Caffe provides multimedia scientists and practitioners with a clean and modifiable framework for state-of-the-art deep learning algorithms and a collection of reference models. The framework is a BSD-licensed C++ library with Python and MATLAB bindings for training and deploying general-purpose convolutional neural networks and other deep models efficiently on commodity architectures. Caffe fits industry and internet-scale media needs by CUDA GPU computation, processing over 40 million images a day on a single K40 or Titan GPU ($\approx$ 2.5 ms per image). By separating model representation from actual implementation, Caffe allows experimentation and seamless switching among platforms for ease of development and deployment from prototyping machines to cloud environments. Caffe is maintained and developed by the Berkeley Vision and Learning Center (BVLC) with the help of an active community of contributors on GitHub. It powers ongoing research projects, large-scale industrial applications, and startup prototypes in vision, speech, and multimedia.

==========

Biometrics AI
---
An Analysis of Random Projections in Cancelable Biometrics , 2014
['PL', 'IL', 'JP', 'KR', 'GB', 'CH', 'CN', 'BR', 'DE', 'IN', 'US']
With increasing concerns about security, the need for highly secure physical biometrics-based authentication systems utilizing \emph{cancelable biometric} technologies is on the rise. Because the problem of cancelable template generation deals with the trade-off between template security and matching performance, many state-of-the-art algorithms successful in generating high quality cancelable biometrics all have random projection as one of their early processing steps. This paper therefore presents a formal analysis of why random projections is an essential step in cancelable biometrics. By formally defining the notion of an \textit{Independent Subspace Structure} for datasets, it can

be shown that random projection preserves the subspace structure of data vectors generated from a union of independent linear subspaces. The bound on the minimum number of random vectors required for this to hold is also derived and is shown to depend logarithmically on the number of data samples, not only in independent subspaces but in disjoint subspace settings as well. The theoretical analysis presented is supported in detail with empirical results on real-world face recognition datasets.

==========

Robotic Processes Automation AI
---
Hands-on experiments on intelligent behavior for mobile robots , 2014
['DE', 'MX']
In recent years, Artificial Intelligence techniques have emerged as useful tools for solving various engineering problems that were not possible or convenient to handle by traditional methods. AI has directly influenced many areas of computer science and becomes an important part of the engineering curriculum. However, determining the important topics for a single semester AI course is a nontrivial task, given the lack of a general methodology. AI concepts commonly overlap with many other disciplines involving a wide range of subjects, including applied approaches to more formal mathematical issues. This paper presents the use of a simple robotic platform to assist the learning of basic AI concepts. The study is guided through some simple experiments using autonomous mobile robots. The central algorithm is the Learning Automata. Using LA, each robot action is applied to an environment to be evaluated by means of a fitness value. The response of the environment is used by the automata to select its next action. This procedure holds until the goal task is reached. The proposal addresses the AI study by offering in LA a unifying context to draw together several of the topics of AI and motivating the students to learn by building some hands on laboratory exercises. The presented material has been successfully tested as AI teaching aide in the University of Guadalajara robotics group as it motivates students and increases enrolment and retention while educating better computer engineers.

==========

Natural Language Processing
---
Linguistic Analysis of Requirements of a Space Project and their Conformity with the Recommendations Proposed by a Controlled Natural Language , 2014
['FR']
The long term aim of the project carried out by the French National Space Agency (CNES) is to design a writing guide based on the real and regular writing of requirements. As a first step in the project, this paper proposes a lin-guistic analysis of requirements written in French by CNES engineers. The aim is to determine to what extent they conform to two rules laid down in INCOSE, a recent guide for writing requirements. Although CNES engineers are not obliged to follow any Controlled Natural Language in their writing of requirements, we believe that language regularities are likely to emerge from this task, mainly due to the writers' experience. The issue is approached using natural language processing tools to identify sentences that do not comply with INCOSE rules. We further review these sentences to understand why the recommendations cannot (or should not) always be applied when specifying large-scale projects.

==========

Digital Twin AI
---
Hands-on experiments on intelligent behavior for mobile robots , 2014
['DE', 'MX']
In recent years, Artificial Intelligence techniques have emerged as useful tools for solving various engineering problems that were not possible or convenient to handle by traditional methods. AI has directly influenced many areas of computer science and becomes an important part of the engineering curriculum. However, determining the important topics for a single semester AI course is a nontrivial task, given the lack of a general methodology. AI concepts commonly overlap with many other disciplines involving a wide range of subjects, including applied approaches to more formal mathematical issues. This paper presents the use of a simple robotic platform to assist the learning of basic AI concepts. The study is guided through some simple experiments using autonomous mobile robots. The central algorithm is the Learning Automata. Using LA, each robot action is applied to an environment to be evaluated by means of a fitness value. The response of the environment is used by the automata to select its next action. This procedure holds until the goal task is reached. The proposal addresses the AI study by offering in LA a unifying context to draw together several of the topics of AI and motivating the students to learn by building some hands on laboratory exercises. The presented material has been successfully tested as AI teaching aide in the University of Guadalajara robotics group as it motivates students and increases enrolment and retention while educating better computer engineers.

==========

Cyber Defense AI
---
Characterizing the Power of Moving Target Defense via Cyber Epidemic Dynamics , 2014
['US', 'GB']
Moving Target Defense (MTD) can enhance the resilience of cyber systems against attacks. Although there have been many MTD techniques, there is no systematic understanding and {\em quantitative} characterization of the power of MTD. In this paper, we propose to use a cyber epidemic dynamics approach to characterize the power of MTD. We define and investigate two complementary measures that are applicable when the defender aims to deploy MTD to achieve a certain security goal. One measure emphasizes the maximum portion of time during which the system can afford to stay in an undesired configuration (or posture), without considering the cost of deploying MTD. The other measure emphasizes the minimum cost of deploying MTD, while accommodating that the system has to stay in an undesired configuration (or posture) for a given portion of time. Our analytic studies lead to algorithms for optimally deploying MTD.

==========

Compliance AI
---
Compliance for reversible client/server interactions , 2014
['IT']
In the setting of session behaviours, we study an extension of the concept of compliance when a disciplined form of backtracking is present. After adding checkpoints to the syntax of session behaviours, we formalise the operational semantics via a LTS, and define a natural notion of checkpoint compliance. We

then obtain a co-inductive characterisation of such compliance relation, and an axiomatic presentation that is proved to be sound and complete. As a byproduct we get a decision procedure for the new compliance, being the axiomatic system algorithmic.

==========

Knowledge Worker Aid AI
---
Hands-on experiments on intelligent behavior for mobile robots , 2014
['DE', 'MX']
In recent years, Artificial Intelligence techniques have emerged as useful tools for solving various engineering problems that were not possible or convenient to handle by traditional methods. AI has directly influenced many areas of computer science and becomes an important part of the engineering curriculum. However, determining the important topics for a single semester AI course is a nontrivial task, given the lack of a general methodology. AI concepts commonly overlap with many other disciplines involving a wide range of subjects, including applied approaches to more formal mathematical issues. This paper presents the use of a simple robotic platform to assist the learning of basic AI concepts. The study is guided through some simple experiments using autonomous mobile robots. The central algorithm is the Learning Automata. Using LA, each robot action is applied to an environment to be evaluated by means of a fitness value. The response of the environment is used by the automata to select its next action. This procedure holds until the goal task is reached. The proposal addresses the AI study by offering in LA a unifying context to draw together several of the topics of AI and motivating the students to learn by building some hands on laboratory exercises. The presented material has been successfully tested as AI teaching aide in the University of Guadalajara robotics group as it motivates students and increases enrolment and retention while educating better computer engineers.

==========

Content Creation AI
---
Hands-on experiments on intelligent behavior for mobile robots , 2014
['DE', 'MX']
In recent years, Artificial Intelligence techniques have emerged as useful tools for solving various engineering problems that were not possible or convenient to handle by traditional methods. AI has directly influenced many areas of computer science and becomes an important part of the engineering curriculum. However, determining the important topics for a single semester AI course is a nontrivial task, given the lack of a general methodology. AI concepts commonly overlap with many other disciplines involving a wide range of subjects, including applied approaches to more formal mathematical issues. This paper presents the use of a simple robotic platform to assist the learning of basic AI concepts. The study is guided through some simple experiments using autonomous mobile robots. The central algorithm is the Learning Automata. Using LA, each robot action is applied to an environment to be evaluated by means of a fitness value. The response of the environment is used by the automata to select its next action. This procedure holds until the goal task is reached. The proposal addresses the AI study by offering in LA a unifying context to draw together several of the topics of AI and motivating the students to learn by building some hands on laboratory exercises. The presented material has been successfully tested as AI teaching aide in the University of Guadalajara robotics group as it motivates students and increases enrolment and retention

while educating better computer engineers.

==========

Peer to Peer Networks AI
---
Efficient Cooperative Anycasting for AMI Mesh Networks , 2014
['DE', 'FR', 'KR', 'AT', 'JP', 'GB', 'CH', 'US']
We have, in recent years, witnessed an increased interest towards enabling a Smart Grid which will be a corner stone to build sustainable energy efficient communities. An integral part of the future Smart Grid will be the communications infrastructure which will make real time control of the grid components possible. Automated Metering Infrastructure (AMI) is thought to be a key enabler for monitoring and controlling the customer loads. %RPL is a connectivity enabling mechanism for low power and lossy networks currently being standardized by the IETF ROLL working group. RPL is deemed to be a suitable candidate for AMI networks where the meters are connected to a concentrator over multi hop low power and lossy links. This paper proposes an efficient cooperative anycasting approach for wireless mesh networks with the aim of achieving reduced traffic and increased utilisation of the network resources. The proposed cooperative anycasting has been realised as an enhancement on top of the Routing Protocol for Low Power and Lossy Networks (RPL), a connectivity enabling mechanism in wireless AMI mesh networks. In this protocol, smart meter nodes utilise an anycasting approach to facilitate efficient transport of metering data to the concentrator node. Moreover, it takes advantage of a distributed approach ensuring scalability.

==========

Emotion Recognition AI
---
STIMONT: A core ontology for multimedia stimuli description , 2014
['HR']
Affective multimedia documents such as images, sounds or videos elicit emotional responses in exposed human subjects. These stimuli are stored in affective multimedia databases and successfully used for a wide variety of research in psychology and neuroscience in areas related to attention and emotion processing. Although important all affective multimedia databases have numerous deficiencies which impair their applicability. These problems, which are brought forward in the paper, result in low recall and precision of multimedia stimuli retrieval which makes creating emotion elicitation procedures difficult and labor-intensive. To address these issues a new core ontology STIMONT is introduced. The STIMONT is written in OWL-DL formalism and extends W3C EmotionML format with an expressive and formal representation of affective concepts, high-level semantics, stimuli document metadata and the elicited physiology. The advantages of ontology in description of affective multimedia stimuli are demonstrated in a document retrieval experiment and compared against contemporary keyword-based querying methods. Also, a software tool Intelligent Stimulus Generator for retrieval of affective multimedia and construction of stimuli sequences is presented.

==========

Image Recognition AI
---
Hands-on experiments on intelligent behavior for mobile robots , 2014
['DE', 'MX']

In recent years, Artificial Intelligence techniques have emerged as useful tools for solving various engineering problems that were not possible or convenient to handle by traditional methods. AI has directly influenced many areas of computer science and becomes an important part of the engineering curriculum. However, determining the important topics for a single semester AI course is a nontrivial task, given the lack of a general methodology. AI concepts commonly overlap with many other disciplines involving a wide range of subjects, including applied approaches to more formal mathematical issues. This paper presents the use of a simple robotic platform to assist the learning of basic AI concepts. The study is guided through some simple experiments using autonomous mobile robots. The central algorithm is the Learning Automata. Using LA, each robot action is applied to an environment to be evaluated by means of a fitness value. The response of the environment is used by the automata to select its next action. This procedure holds until the goal task is reached. The proposal addresses the AI study by offering in LA a unifying context to draw together several of the topics of AI and motivating the students to learn by building some hands on laboratory exercises. The presented material has been successfully tested as AI teaching aide in the University of Guadalajara robotics group as it motivates students and increases enrolment and retention while educating better computer engineers.

==========

Marketing Automation AI
---
Hands-on experiments on intelligent behavior for mobile robots , 2014
['DE', 'MX']
In recent years, Artificial Intelligence techniques have emerged as useful tools for solving various engineering problems that were not possible or convenient to handle by traditional methods. AI has directly influenced many areas of computer science and becomes an important part of the engineering curriculum. However, determining the important topics for a single semester AI course is a nontrivial task, given the lack of a general methodology. AI concepts commonly overlap with many other disciplines involving a wide range of subjects, including applied approaches to more formal mathematical issues. This paper presents the use of a simple robotic platform to assist the learning of basic AI concepts. The study is guided through some simple experiments using autonomous mobile robots. The central algorithm is the Learning Automata. Using LA, each robot action is applied to an environment to be evaluated by means of a fitness value. The response of the environment is used by the automata to select its next action. This procedure holds until the goal task is reached. The proposal addresses the AI study by offering in LA a unifying context to draw together several of the topics of AI and motivating the students to learn by building some hands on laboratory exercises. The presented material has been successfully tested as AI teaching aide in the University of Guadalajara robotics group as it motivates students and increases enrolment and retention while educating better computer engineers.

==========

**Theme 2 Jupyter Notebook**

# Setting up database connection

Importing required python libraries

In [1]:

```
import os
import sys
sys.path.append(os.path.abspath('../../'))
from query_indicators import generate_save_path

import py2neo
from nesta.core.luigihacks.misctools import get_config
from nesta.core.orms.orm_utils import graph_session
import igraph as ig
import pandas as pd
import boto3
```

In [2]:

```
S3 = boto3.resource('s3')
SAVE_PATH = generate_save_path()  # EURITO collaborators: this is generated assuming you have stuck to the convention 'theme_x/something/something_else.ipynb'
BUCKET = 'eurito-indicators'  # EURITO collaborators: please don't change this
SAVE_RESULTS = True  # Set this to "False" when you want to view figures inline. When "True", results will be saved to S3.
```

Establish connection to the Neo4j database

In [3]:

```
conf = get_config('neo4j.config', 'neo4j')
gkwargs = dict(host=conf['host'], secure=True,
               auth=(conf['user'], conf['password']))
```

In [29]:

```
def _s3_savetable(df, object_path):
    """Upload the table to s3"""
    if not SAVE_RESULTS:
        return
    if len(df.columns) == 1:
        df.columns = ['value']
    #df = df / df.max().max()
    table_data = df.to_csv(sep='|').encode()
    obj = S3.Object(BUCKET, os.path.join(f'tables/{SAVE_PATH}', object_path))
    obj.put(Body=table_data)
```

# Retrieving nodes from Neo4j

Create a graph object which will be used for our queries

In [5]:

```
with graph_session(**gkwargs) as tx:
    graph = tx.graph
```

Set the type of the node that should be retrieved. Available types are: "Project", "Organisation", "Publication", "Topic", "Report", "Datasets", "Software", "Proposal_Call". Simply change the word "Organisation" below to the required node type and re-run the cell.
**Nodes of type "Organisation" have the following fields:**
*name* - organisation name
*betw* - organisation centrality
*country_code* - 2 letter country code
*country_name* - country name

In [9]:

```
node_type = "Organisation"
```

Create a list from the graph nodes.

In [ ]:

```
node_list = list(graph.nodes.match(node_type))
```

Convert to dataframe, sort the table according to centrality column and print the top 15

In [14]:

```
node_table = pd.DataFrame(node_list)
node_table = node_table.drop("centrality", axis=1)
top_betw = node_table.sort_values(by=['betw'], ascending=False)
top_nodes = top_betw.head(15)
top_nodes
```

Out[14]:

Save the generated table into S3 bucket

In [15]:

```
_s3_savetable(top_nodes, object_path=f'{node_type}/organisation_centrality_top15.csv')
```

**Nodes of type "Project" have the following fields:**
*acronym* - project acronym
*betw* - project centrality
*ec_contribution* - funding by EC, mio EUR
*start_date_code, end_date_code* - project start and end date
*framework, funded_under, funding_scheme* - funding framework and program
*grant_num* - 6 digit grant number
*objective* - project objective
*project_description* - project description
*rcn* - project identificator
*status* - project status (e.g. closed, ongoing)
*total_cost* - total budget, mio EUR
*website* - project website

In [22]:

```
node_type = "Project"
```

In [23]:

```
node_list = list(graph.nodes.match(node_type))
node_table = pd.DataFrame(node_list)
top_betw = node_table.sort_values(by=['betw'], ascending=False)
#Get first 15 nodes of the specified type
top_nodes = top_betw.head(15)
top_nodes
```

Out[23]:

In [30]:

```
_s3_savetable(top_nodes, object_path=f'{node_type}/project_centrality_top15.csv')
```

**Theme 3 Jupyter Notebook**

In [1]:

```
%matplotlib inline
```

In [2]:

```
import os
import sys
sys.path.append(os.path.abspath('../../'))
from query_indicators import generate_save_path
from query_indicators import get_eu_countries
```

In [3]:

```
import boto3
from collections import defaultdict
from clio_lite import clio_search, clio_search_iter
import io
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.collections import PatchCollection
from matplotlib.patches import Rectangle
import numpy as np
import pandas as pd
```

In [4]:

```
# Env variables
mpl.rcParams['hatch.linewidth'] = 0.2
mpl.rcParams['font.size'] = 18
mpl.rcParams['image.cmap'] = 'Pastel1'
#os.environ['AWS_SHARED_CREDENTIALS_FILE'] = '/home/aidrissov/.aws/credentials'  # <---
Note: NOT nesta's AWS credentials
#from os import path
#print ("File exists:" + str(path.exists('/home/aidrissov/.aws/credentials')))
#print ("directory exists:" + str(path.exists('/home/aidrissov/.aws/')))
#TRY MANUAL CREDENTIALS
```

In [5]:

```
# Some globals
URL = "https://search-eurito-prod-bbyn72q2rhx4ifj6h5dom43uhy.eu-west-1.es.amazonaws.com/"
INDEX = "arxiv_v0"
FIELDS = ['terms_tokens_entity', 'textBody_abstract_article']
EU_COUNTRIES = get_eu_countries()
COLORS = plt.get_cmap('Set2').colors
COLOR_MAP = 'Pastel1'
S3 = boto3.resource('s3')
SAVE_PATH = generate_save_path()  # EURITO collaborators: this is generated assuming you
have stuck to the convention 'theme_x/something/something_else.ipynb'
BUCKET = 'eurito-indicators'  # EURITO collaborators: please don't change this
SAVE_RESULTS = True  # Set this to "False" when you want to view figures inline. When
"True", results will be saved to S3.

if SAVE_RESULTS:
    plt.ioff()  # <--- for turning off visible figs
else:
```

```
plt.ion()
```

In [6]:

```
def make_search(query, max_query_terms, yr0=2014, yr1=2019, countries=EU_COUNTRIES,
window=1):
    """
    Retrieve count and score data for a given basic clio search.

    Args:
        query (str): Seed query for clio.
        max_query_terms (list): Triple of max_query_terms (low, middle, high) to use from the initial
query.
        yr0 (int): Start year in range to use in filter.
        yr1 (int): Final year in range to use in filter.
        countries (list): A list of countries to filter (default to all EU).
        window (int): The number of years to consider in between time windows. Note that changing
this will lead to double-counting.
    Returns:
        data (dict): {max_query_terms --> [{year --> sum_score} for each country]}
        all_scores (dict): {max_query_terms --> {country --> [score for doc in docs] } }
    """
    top_doc = None
    _data = defaultdict(lambda: defaultdict(dict))  # {max_query_terms --> {year --> {country -->
score} } }
    all_scores = defaultdict(lambda: defaultdict(list))  # {max_query_terms --> {country --> [score
for doc in docs] } }
    for n in max_query_terms:
        # Set the order of the countries
        for ctry in EU_COUNTRIES:
            _data[n][ctry]
            all_scores[n][ctry]
        # Iterate over years
        for yr in range(yr0, yr1+1):
            # Set default values for countries
            for ctry in EU_COUNTRIES:
                _data[n][ctry][yr] = 0
            # Iterate over docs
            filters = [{"range":{"year_of_article":{"gte":yr, "lt":yr+window}}}]
            for doc in clio_search_iter(url=URL, index=INDEX, query=query, fields=FIELDS,
                                        max_query_terms=n, post_filters=filters, chunksize=5000):
                if '_score' not in doc or doc['terms_countries_article'] is None:
                    continue
                score = doc['_score']
                for ctry in filter(lambda x: x in countries, doc['terms_countries_article']):
                    if top_doc is None:
                        top_doc = doc
                    all_scores[n][ctry].append(score)
                    _data[n][ctry][yr] += score
    # Reformat data as {max_query_terms --> [{year --> score} for each country in order]}
    data = {}
    for n, ctry_data in _data.items():
        data[n] = []
        for ctry, yr_data in ctry_data.items():
            data[n].append(yr_data)
    return top_doc, data, all_scores
```

# Indicator calculations

Each of these functions is assumed to take the form

```
def _an_indicator_calulation(data, year=None, _max=1):
    """
    A function calculating an indicator.

    Args:
        data (list): Rows of data
        year (int): A year to consider, if applicable.
        _max (int): Divide by this to normalise your results. This is automatically applied in
:obj:`make_activity_plot`
    Returns:
        result (list) A list of indicators to plot. The length of the list is assumed to be equal to the
number of countries.
    """
    # Calculate something
```

In [7]:

```
def _total_activity_by_country(data, year=None, _max=1):
    """
    Indicator: Sum of relevance scores, by year (if specified) or in total.
    """
    if year is None:
        scores = [sum(row.values())/_max for row in data]
    else:
        scores = [row[year]/_max for row in data]
    return scores


def _average_activity_by_country(data, year=None, _max=1):
    """
    Indicator: Mean relevance score. This function is basically a lambda, since it assumes the
average has already been calculated.
    """
    return [row/_max for row in data]


def _corrected_average_activity_by_country(data, year=None, _max=1):
    """
    Indicator: Mean relevance score minus it's (very) approximate Poisson error.
    """
    return [(row - np.sqrt(row))/_max for row in data]


def _linear_coeffs(years, scores, _max):
    """Calculates linear coefficients for scores wrt years"""
    return [np.polyfit(_scores, _years, 1)[0]/_max
            if all(v > 0 for v in _scores) else 0
            for _years, _scores in zip(years, scores)]


def _trajectory(data, year=None, _max=1):
    """
    Indicator: Linear coefficient of total relevance score wrt year
```

```
    """
    years = [list(row.keys()) for row in data]
    scores = [list(row.values()) for row in data]
    return _linear_coeffs(years, scores, _max)


def _corrected_trajectory(data, year=None, _max=1):
    """
    Indicator: Linear coefficient of upper and lower limits of relevance score wrt year
    """
    # Reformulate the data in terms of upper and lower bounds
    years, scores = [], []
    for row in data:
        _years, _scores = [], []
        for k, v in row.items():
            _years += [k,k]
            _scores += [v - np.sqrt(v), v + np.sqrt(v)]  # Estimate upper and lower limits with very
approximate Poisson errors
        years.append(_years)
        scores.append(_scores)
    return _linear_coeffs(years, scores, _max)
```

# Plotting functionality

In [8]:

```
class _Sorter:
    def __init__(self, values, topn=None):
        if topn is None:
            topn = len(values)
        self.indices = list(np.argsort(values))[-topn:]  # Argsort is ascending, so -ve indexing to pick
up topn
    def sort(self, x):
        """Sort list x by indices"""
        return [x[i] for i in self.indices]


def _s3_savefig(query, fig_name, extension='png'):
    """Save the figure to s3. The figure is grabbed from the global scope."""
    if not SAVE_RESULTS:
        return
    outname = (f'figures/{SAVE_PATH}/'
               f'{query.replace(" ","_").lower()}'
               f'/{fig_name.replace(" ","_").lower()}'
               f'.{extension}')
    with io.BytesIO() as f:
        plt.savefig(f, bbox_inches='tight', format=extension, pad_inches=0)
        obj = S3.Object(BUCKET, outname)
        f.seek(0)
        obj.put(Body=f)


def _s3_savetable(data, key, index, object_path, transformer=lambda x: x):
    """Upload the table to s3"""
    if not SAVE_RESULTS:
        return
    df = pd.DataFrame(transformer(data[key]), index=index)
```

```
    if len(df.columns) == 1:
        df.columns = ['value']
    df = df / df.max().max()
    table_data = df.to_csv().encode()
    obj = S3.Object(BUCKET, os.path.join(f'tables/{SAVE_PATH}', object_path))
    obj.put(Body=table_data)


def make_activity_plot(f, data, countries, max_query_terms, query,
                       year=None, label=None, x_padding=0.5, y_padding=0.05, xlabel_fontsize=14):
    """
    Make a query and generate indicators by country, saving the plots to S3 and saving the rawest data
    to tables on S3.


    Args:
        f: An indicator function, as described in the 'Indicator calculations' section.
        data (dict): {max_query_terms --> [{year --> sum_score} for each country]}
        countries (list): A list of EU ISO-2 codes
        max_query_terms (list): Triple of max_query_terms for clio, corresponding to low, middle and high values of
                                max_query_terms to test robustness of the query.
        query (str): query used to generate this data.
        year (int): Year to generate the indicator for (if applicable).
        label (str): label for annotating the plot.
        {x,y}_padding (float): Aesthetic padding around the extreme limits of the {x,y} axis.
        xlabel_fontsize (int): Fontsize of the x labels (country ISO-2 codes).
    """
    # Calculate the indicator for each value of n, then recalculate the normalised indicator
    _, middle, _ = (f(data[n], year=year) for n in max_query_terms)
    low, middle, high = (f(data[n], year=year, _max=max(middle)) for n in max_query_terms)
    indicator = [np.median([a, b, c]) for a, b, c in zip(low, middle, high)]

    # Sort all data by indicator value
    s = _Sorter(indicator)
    countries = s.sort(countries)
    low = s.sort(low)
    middle = s.sort(middle)
    high =  s.sort(high)
    indicator = s.sort(indicator)

    # Make the scatter plot
    fig, ax = plt.subplots(figsize=(15, 6))
    make_error_boxes(ax, low, middle, high)  # Draw the bounding box
    ax.scatter(countries, indicator,  s=0, marker='o', color='black')  # Draw the centre mark
    ax.set_title(f'{label}\nQuery: "{query}"')
    ax.set_ylabel(label)

    # Set limits and formulate
    y0 = min(low+middle+high)
    y1 = max(low+middle+high)
    if -y1*y_padding < y0:
        y0 = -y1*y_padding
    else:  # In case of negative values
        y0 = y0 - np.abs(y0*y_padding)
    ax.set_ylim(y0, y1*(1+y_padding))
```

```
    ax.set_xlim(-x_padding, len(countries)-x_padding)
    for tick in ax.xaxis.get_major_ticks():
        tick.label.set_fontsize(xlabel_fontsize)

    # Save to s3 & return
    _s3_savefig(query, label)
    return ax


def make_error_boxes(ax, low, middle, high, facecolor='r',
                     edgecolor='None', alpha=0.5):
    """
    Generate outer rectangles based on three values, and draw a horizontal line through the
middle of the rectangle.
    No assumption is made on the order of values, so don't worry if they're not properly ordered.

    Args:
        ax (matplotlib.axis): An axis to add patches to.
        {low, middle, high} (list): Three concurrent lists of values from which to calculate the
rectangle limits.
        {facecolor, edgecolor} (str): The {face,edge} colour of the rectangles.
        alpha (float): The alpha of the rectangles.
    """
    # Generate the rectangle
    errorboxes = []
    middlelines = []
    for x, ys in enumerate(zip(low, middle, high)):
        rect = Rectangle((x - 0.45, min(ys)), 0.9, max(ys) - min(ys))
        line = Rectangle((x - 0.45, np.median(ys)), 0.9, 0)
        errorboxes.append(rect)
        middlelines.append(line)

    # Create patch collection with specified colour/alpha
    pc = PatchCollection(errorboxes, facecolor=facecolor, alpha=alpha, edgecolor=edgecolor,
hatch='/')
    lc = PatchCollection(middlelines, facecolor='black', alpha=0.9, edgecolor='black')

    # Add collection to axes
    ax.add_collection(pc)
    ax.add_collection(lc)


def stacked_scores(all_scores, query, topn=8,
                   low_bins=[10**i for i in np.arange(0, 1.1, 0.025)],
                   high_bins=[10**i for i in np.arange(1.1, 2.5, 0.05)],
                   x_scale='log', label='Relevance score breakdown',
                   xlabel='Relevance score', ylabel='Number of relevant documents',
                   legend_fontsize='small', legend_cols=2):
    """
    Create stacked histogram of document scores by country. Two sets of bins are used,
    in order to have a more legible binning scale.

    Args:
        all_scores (dict): {max_query_terms --> {country --> [score for doc in docs] } }
        query (str): query used to generate this data.
        low_bins (list): List of initial bin edges.
```

```
    high_bins (list): List of supplementary bin edges. These could have a different spacing scheme to the lower bin edges.
    x_scale (str): Argument for `ax.set_xscale`.
    label (str): label for annotating the plot.
    {x,y}_label (str): Argument for `ax.set_{x,y}label`.
    legend_fontsize (str): Argument for legend fontsize.
    legend_cols (str): Argument for legend ncol.
  """

  # Sort countries and scores by the sum of scores by country
  countries = list(all_scores.keys())
  scores = list(all_scores.values())
  s = _Sorter([sum(v) for v in scores], topn=topn)
  scores = s.sort(scores)
  countries = s.sort(countries)

  # Plot the stacked scores
  fig, ax = plt.subplots(figsize=(10, 6))
  plt.set_cmap(COLOR_MAP)
  ax.hist(scores, bins=low_bins+high_bins, stacked=True,
      label=countries, color=COLORS[:len(scores)])

  # Prettify the plot
  ax.set_xlabel(xlabel)
  ax.set_ylabel(ylabel)
  ax.legend(fontsize=legend_fontsize, ncol=legend_cols)
  ax.set_xlim(low_bins[0], None)
  ax.set_xscale(x_scale)
  ax.set_title(f'{label}\nQuery: "{query}"')

  # Save to s3
  _s3_savefig(query, label)
  return ax
```

# Bringing it all together

In [9]:

```
def generate_indicator(q, max_query_terms=[7, 10, 13], countries=EU_COUNTRIES, *args, **kwargs):
  """
  Make a query and generate indicators by country, saving the plots to S3 and saving the rawest data
  to tables on S3.


  Args:
    q (str): The query to Elasticsearch
    max_query_terms (list): Triple of max_query_terms for clio, corresponding to low, middle and high values of
                max_query_terms to test robustness of the query.
    countries (list): A list of EU ISO-2 codes
  Returns:
    top_doc (dict): The highest ranking document from the search.
    data (dict): {max_query_terms --> [{year --> sum_score} for each country]}
    all_scores (dict): {max_query_terms --> {country --> [score for doc in docs] } }
  """
```

```
    # Make the search and retrieve scores by country, and the highest ranking doc
    example_doc, data, all_scores = make_search(q, max_query_terms=max_query_terms,
countries=countries, *args, **kwargs)

    # Reformat the scores to calculate the average
    avg_scores = defaultdict(list)
    for ctry in countries:
        for n, _scores in all_scores.items():
            mean = np.mean(_scores[ctry]) if len(_scores[ctry]) > 0 else 0
            avg_scores[n].append(mean)

    plot_kwargs = dict(countries=countries, max_query_terms=max_query_terms, query=q)
    # Calculate loads of indicators and save the plots
    _ = make_activity_plot(_total_activity_by_country, data, label='Total relevance score',
**plot_kwargs)
    _ = make_activity_plot(_average_activity_by_country, avg_scores, label='Average relevance',
**plot_kwargs)
    _ = make_activity_plot(_corrected_average_activity_by_country, avg_scores, label='Corrected
average relevance',  **plot_kwargs)
    _ = make_activity_plot(_trajectory, data, label='Trajectory', **plot_kwargs)
    _ = make_activity_plot(_corrected_trajectory, data, label='Corrected trajectory', **plot_kwargs)
    _ = stacked_scores(all_scores[max_query_terms[1]], query=q)

    # Save the basic raw data as tables. Note: not as rich as the plotted data.
    _q = q.replace(" ","_").lower()
    _s3_savetable(data, max_query_terms[1], index=countries, object_path=f'{_q}/LMA.csv')
    _s3_savetable(avg_scores, max_query_terms[1], index=countries,
object_path=f'{_q}/avg_LMA.csv')

    plt.close('all')  # Clean up the memory cache (unbelievable that matplotlib doesn't do this)
    return example_doc, data, all_scores
```

# Iterate over queries

In [12]:

```
for term in ["Adaptation to climate change, including societal transformation",
             "Cancer",
             "Climate-neutral and smart cities",
             "Soil health and food"]:
    print(term)
    print("-"*len(term))
    top_doc, data, all_scores = generate_indicator(term)
    print(top_doc['title_of_article'], ",", top_doc['year_of_article'])
    print(top_doc['terms_countries_article'])
    print(top_doc['textBody_abstract_article'])
    print("\n==============================\n")
```

```
Adaptation to climate change, including societal transformation
---------------------------------------------------------------
Validity of altmetrics data for measuring societal impact: A study using
  data from Altmetric and F1000Prime , 2014
['DE']
Can altmetric data be validly used for the measurement of societal impact?
The current study seeks to answer this question with a comprehensive dataset
```

(about 100,000 records) from very disparate sources (F1000, Altmetric, and an in-house database based on Web of Science). In the F1000 peer review system, experts attach particular tags to scientific papers which indicate whether a paper could be of interest for science or rather for other segments of society. The results show that papers with the tag "good for teaching" do achieve higher altmetric counts than papers without this tag - if the quality of the papers is controlled. At the same time, a higher citation count is shown especially by papers with a tag that is specifically scientifically oriented ("new finding"). The findings indicate that papers tailored for a readership outside the area of research should lead to societal impact. If altmetric data is to be used for the measurement of societal impact, the question arises of its normalization. In bibliometrics, citations are normalized for the papers' subject area and publication year. This study has taken a second analytic step involving a possible normalization of altmetric data. As the results show there are particular scientific topics which are of especial interest for a wide audience. Since these more or less interesting topics are not completely reflected in Thomson Reuters' journal sets, a normalization of altmetric data should not be based on the level of subject categories, but on the level of topics.

==========

Cancer
---
A Statistical Approach to Identifying Significant Transgenerational Methylation Changes , 2014
['GB', 'US', 'CH', 'CA', 'IE']
Epigenetic aberrations have profound effects on phenotypic output. Genome wide methylation alterations are inheritable to pass down the aberrations through multiple generations. We developed a statistical method, Genome-wide Identification of Significant Methylation Alteration, GISAIM, to study the significant transgenerational methylation changes. GISAIM finds the significant methylation aberrations that are inherited through multiple generations. In a concrete biological study, we investigated whether exposing pregnant rats (F0) to a high fat (HF) diet throughout pregnancy or ethinyl estradiol (EE2)-supplemented diet during gestation days 14 20 affects carcinogen-induced mammary cancer risk in daughters (F1), granddaughters (F2) and great-granddaughters (F3). Mammary tumorigenesis was higher in daughters and granddaughters of HF rat dams, and in daughters, granddaughters and great-granddaughters of EE2 rat dams. Outcross experiments showed that increased mammary cancer risk was transmitted to HF granddaughters equally through the female or male germlines, but is only transmitted to EE2 granddaughters through the female germline. Transgenerational effect on mammary cancer risk was associated with increased expression of DNA methyltransferases, and across all three EE2 generations hypo or hyper methylation of the same 375 gene promoter regions in their mammary glands. Our study shows that maternal dietary estrogenic exposures during pregnancy can increase breast cancer risk in multiple generations of offspring, and the increase in risk may be inherited through non-genetic means, possibly involving DNA methylation.

==========

Climate-neutral and smart cities
---
Software-Defined and Virtualized Future Mobile and Wireless Networks: A Survey , 2014
['CN', 'CH', 'GR']

With the proliferation of mobile demands and increasingly multifarious services and applications, mobile Internet has been an irreversible trend. Unfortunately, the current mobile and wireless network (MWN) faces a series of pressing challenges caused by the inherent design. In this paper, we extend two latest and promising innovations of Internet, software-defined networking and network virtualization, to mobile and wireless scenarios. We first describe the challenges and expectations of MWN, and analyze the opportunities provided by the software-defined wireless network (SDWN) and wireless network virtualization (WNV). Then, this paper focuses on SDWN and WNV by presenting the main ideas, advantages, ongoing researches and key technologies, and open issues respectively. Moreover, we interpret that these two technologies highly complement each other, and further investigate efficient joint design between them. This paper confirms that SDWN and WNV may efficiently address the crucial challenges of MWN and significantly benefit the future mobile and wireless network.

==========

Soil health and food
---
GREEND: An Energy Consumption Dataset of Households in Italy and Austria , 2014
['AT']
Home energy management systems can be used to monitor and optimize consumption and local production from renewable energy. To assess solutions before their deployment, researchers and designers of those systems demand for energy consumption datasets. In this paper, we present the GREEND dataset, containing detailed power usage information obtained through a measurement campaign in households in Austria and Italy. We provide a description of consumption scenarios and discuss design choices for the sensing infrastructure. Finally, we benchmark the dataset with state-of-the-art techniques in load disaggregation, occupancy detection and appliance usage mining.

==========

**Appendix B - Data collection and processing code documentation**

# Data sources and core processing¶

In EURITO we predominantly use four data sources:

- EU-funded research from *CORDIS*
- Technical EU research on *arXiv*
- EU Patents from *PATSTAT*
- EU Companies [under license, we can't specify the source publicly. Contact us for more details!]

## CORDIS¶

Data from the CORDIS's H2020 API and FP7 API funded projects is extracted using code found in this repository.

In total, 51250 organisations and 50640 projects were extracted from the API. There are 1102 proposal calls, 245465 publications and 34507 reports. In total 6545 are associated with the projects.

Software outputs are associated with the projects, using OpenAIRE API.

All of these entities are then linked together, and stored using a neo4j graph database. The code for automatically piping the data in neo4j is provided here.

### Cordis to Neo4j¶

Tools for piping data from a SqlAlchemy ORM to Neo4j, to be used in the Luigi pipeline.

orm_to_neo4j(*session*, *transaction*, *orm_instance*, *parent_orm=None*, *rel_name=None*)[source]¶

Pipe a SqlAlchemy ORM instance (a 'row' of data) to neo4j, inserting it as a node or relationship, as appropriate.

**Parameters:**

- **session** (*sqlalchemy.Session*) – SQL DB session.
- **transaction** (*py2neo.Transaction*) – Neo4j transaction
- **orm_instance** (*sqlalchemy.Base*) – Instance of a SqlAlchemy ORM, i.e. a 'row' of data.
- **parent_orm** (*sqlalchemy.Base*) – Parent ORM to build relationship to
- **rel_name** (*str*) – Name of the relationship to be added to Neo4j

build_relationships(*session*, *graph*, *orm*, *data_row*, *rel_name*, *parent_orm=None*)[source]¶

Build a py2neo.Relationship object from SqlAlchemy objects.x

**Parameters:**

- **session** (*sqlalchemy.Session*) – SQL DB session.
- **transaction** (*py2neo.Transaction*) – Neo4j transaction
- **orm** (*sqlalchemy.Base*) – A SqlAlchemy ORM
- **rel_name** (*str*) – Name of the relationship to be added to Neo4j
- **parent_orm** (*sqlalchemy.Base*) – Another ORM to build relationship to. If this is not specified, it implies that `orm` is node, rather than a relationship.

**Returns:** Relationships pointing to the node (inferred from ORM), and one pointing back to it's associated project.

**Return type:** {relationship, back_relationship}

set_constraints(*orm*, *graph_schema*)[source]¶

Set constraints in the neo4j graph schema.

**Parameters:**

- **orm** (*sqlalchemy.Base*) – A SqlAlchemy ORM
- **graph_schema** (*py2neo.Graph.Schema*) – Neo4j graph schema.

prepare_base_entities(*table*)[source]¶

Returns the objects required to generate a graph representation of the ORM.

**Parameters:** **table** (*sqlalchemy.sql.Table*) – SQL alchemy table object from which to extract an graph representation.

**Returns:** Two ORMs and a string describing their relationship

**Return type:** {orm, parent_orm, rel_name}

flatten(*orm_instance*)[source]¶

Convert a SqlAlchemy ORM (i.e. a ‘row’ of data) to flat JSON.

**Parameters:** **orm_instance** (*sqlalchemy.Base*) – Instance of a SqlAlchemy ORM, i.e. a ‘row’ of data.

**Returns:** A flat row of data, inferred from orm_instance

**Return type:** row (dict)

flatten_dict(*row, keys=[('title',), ('street', 'city', 'postalCode')]*)[source]¶

Flatten a dict by concatenating string values of matching keys.

| | |
|---|---|
| **Parameters:** | **row** (*dict*) – Data to be flattened |
| **Returns:** | Concatenated data. |
| **Return type:** | flat (str) |

retrieve_node(*session*, *graph*, *orm*, *parent_orm*, *data_row*)[source]¶

Retrieve an existing node from neo4j, by first retrieving it's id (field name AND value) via SqlAlchemy.

| | |
|---|---|
| **Parameters:** | • **session** (*sqlalchemy.Session*) – SQL DB session.<br>• **transaction** (*py2neo.Transaction*) – Neo4j transaction<br>• **orm** (*sqlalchemy.Base*) – SqlAlchemy ORM describing `data_row`<br>• **parent_orm** (*sqlalchemy.Base*) – Parent ORM to build relationship to<br>• **data_row** (*dict*) – Flat row of data retrieved from orm |
| **Returns:** | Node of data corresponding to data_row |
| **Return type:** | node (py2neo.Node) |

table_from_fk(*fks*)[source]¶

Get the table name of the fk constraint, ignoring the cordis_projects table

| | |
|---|---|
| **Parameters:** | **fks** (`list` of SqlAlchemy.ForeignKey) – All foreign keys for a given table. |
| **Returns:** | The table name corresponding to the non-Project foreign key. |
| **Return type:** | tablename (str) |

get_row(*session*, *parent_orm*, *orm*, *data_row*)[source]¶

Retrieve a flat row of data corresponding to the parent relation, inferred via foreign keys.

| | |
|---|---|
| **Parameters:** | • **session** (*sqlalchemy.Session*) – SQL DB session.<br>• **parent_orm** (*sqlalchemy.Base*) – Parent ORM to build relationship to<br>• **orm** (*sqlalchemy.Base*) – SqlAlchemy ORM describing `data_row`<br>• **data_row** (*dict*) – Flat row of data retrieved from orm |
| **Returns:** | Flat row of data retrieved from parent_orm |
| **Return type:** | _row (dict) |

## Enrich Cordis with OpenAIRE¶

Tools for collecting OpenAIRE data (by Cordis project), and piping to Neo4j.

write_record_to_neo(*record*, *output_type*, *graph*)[source]¶

A utility function, which takes record and writes it to neo4j graph

**Parameters:**

- **record** (*dict*) – a dictionary that contains metadata about a record
- **output_type** (*str*) – type of record to be extracted from OpenAIRE API. Accepts “software”, “datasets”, “publications”, “ECProjects”
- **graph** (*graph_session*) – connection to neo4j database

get_project_soups(*currentUrl*, *reqsession*, *output_type*, *projectID*)[source]¶

Gets a beautiful soup according to output type and projectID

**Parameters:**

- **currentUrl** (*str*) – URL to OpenAIRE API
- **reqsession** (*instance of Requests session*) – currently open HTTP request
- **output_type** (*str*) – type of record to be extracted from OpenAIRE API. Accepts “software”, “datasets”, “publications”, “ECProjects”
- **projectID** (*str*) – EC project identifier

**Returns:** a list of BeautifulSoup objects that contain the results from API call

**Return type:** souplist(list)

get_results_from_soups(*souplist*)[source]¶

Extracts string from all BeautifulSoup objects and merges them into one list

**Parameters:** **souplist** (*list*) – a list of BeautifulSoup objects that contain the results from API call

| | |
|---|---|
| **Returns:** | a list of strings with results metadata |
| **Return type:** | resultlist(list) |

## arXiv¶

All articles from arXiv, which is the world's premier repository of pre-prints of articles in the physical, quantitative and computational sciences, are already automatically collected, geocoded (using GRID) and enriched with topics (using MAG). Articles are assigned to EU NUTS regions (at all levels) using nesta's nuts-finder python package.

Data is transferred to EURITO's elasticsearch server via nesta's es2es package. The lolvelty algorithm is then applied to the data in order to generate a novelty metric for each article. This procedure is bettter described in this blog (see "Defining novelty").

The indicators using this data source are presented in this other EURITO repository.

In total, 1598033 articles have been processed, of which 459371 have authors based in EU nations.

## PATSTAT¶

All patents from the PATSTAT service have been collected in nesta's own database using nesta's pypatstat library. Since this database is very large, we have selected patents which belong to a patent family with a granted patent first published after the year 2000, with at least one person or organisation (inventor or applicant) based in an EU member state. This leads to 1552303 patents in the database.

Data is transferred to EURITO's elasticsearch server via nesta's es2es package. The lolvelty algorithm is then applied to the data in order to generate a novelty metric for each article. This procedure is better described in this blog (see "Defining novelty").

The indicators using this data source are presented in this other EURITO repository.

## Companies¶

We have acquired private-sector company data under license. The dataset contains 550,540 companies, of which 133,641 are based in the EU.

Data is transferred to EURITO's elasticsearch server via nesta's es2es package. The lolvelty algorithm is then applied to the data in order to generate a novelty metric for each article. This procedure is better described in this blog (see "Defining novelty").

The indicators using this data source are presented in this other EURITO repository

# Batchables

## run.py (lolvelty)

Calculates the "lolvelty" novelty score to documents in Elasticsearch, on a document-by-document basis. Note that this is a slow procedure, and the bounds of document "lolvelty" can't be known a priori.

**run**()[source]

# Production pipelines

We use luigi routines to orchestrate our pipelines. The batching procedure relies on batchables as described in batchables. Other than luigihacks.autobatch, which is respectively documented in Nesta's codebase, the routine procedure follows the Luigi documentation well.

## Transfer of Elasticsearch data

This pipeline is responsible for the transfer of Elasticsearch data from a remote origin (in our case, Nesta's Elasticsearch endpoint) to EURITO's endpoint.

*class* **Es2EsTask**(**args*, ***kwargs*)[source]

Bases: **luigi.task.Task**

**date** = *<luigi.parameter.DateParameter object>*

**origin_endpoint** = *<luigi.parameter.Parameter object>*

**origin_index** = *<luigi.parameter.Parameter object>*

**dest_endpoint** = *<luigi.parameter.Parameter object>*

**dest_index** = *<luigi.parameter.Parameter object>*

**test** = *<luigi.parameter.BoolParameter object>*

**chunksize** = *<luigi.parameter.IntParameter object>*

**do_transfer_index** = *<luigi.parameter.BoolParameter object>*

**db_config_path** = *<luigi.parameter.Parameter object>*

**output**()[source]

Points to the output database engine

**run**()[source]

The task run method, to be overridden in a subclass.

See Task.run

*class* **EsLolveltyTask**(**args*, ***kwargs*)[source]

Bases: **nesta.core.luigihacks.estask.LazyElasticsearchTask**

**date** = *<luigi.parameter.DateParameter object>*

**origin_endpoint** = *<luigi.parameter.Parameter object>*

**origin_index** = *<luigi.parameter.Parameter object>*

**test** = *<luigi.parameter.BoolParameter object>*

**process_batch_size** = *<luigi.parameter.IntParameter object>*

**do_transfer_index** = *<luigi.parameter.BoolParameter object>*

**requires**()[source]

The Tasks that this Task depends on.

A Task will only run if all of the Tasks that it requires are completed. If your Task does not require any other Tasks, then you don't need to override this method. Otherwise, a subclass can override this method to return a single Task, a list of Task instances, or a dict whose values are Task instances.

See Task.requires

*class* **RootTask**(**args*, ***kwargs*)[source]

Bases: **luigi.task.WrapperTask**

**production** = *<luigi.parameter.BoolParameter object>*

**date** = *<luigi.parameter.DateParameter object>*

**requires**()[source]

The Tasks that this Task depends on.

A Task will only run if all of the Tasks that it requires are completed. If your Task does not require any other Tasks, then you don't need to override this method. Otherwise, a subclass can override this method to return a single Task, a list of Task instances, or a dict whose values are Task instances.

See Task.requires

## Centrality Pipeline

Takes network from Neo4j database, calculates network centrality measures and updates each node in the database with new centrality attributes

*class* **RootTask**(**args*, ***kwargs*)[source]

Bases: **luigi.task.WrapperTask**

The root task, which collects the supplied parameters and calls the main task.

**Parameters:**

- **date** (*datetime*) – Date used to label the outputs
- **output_type** (*str*) – type of record to be extracted from OpenAIRE API. Accepts “software”, “datasets”, “publications”, “ECProjects”
- **production** (*bool*) – test mode or production mode

**date** = *<luigi.parameter.DateParameter object>*

**output_type** = *<luigi.parameter.Parameter object>*

**production** = *<luigi.parameter.BoolParameter object>*

**requires**()[source]

Call the task to run before this in the pipeline.

*class* **CalcCentralityTask**(**args*, ***kwargs*)[source]

Bases: **luigi.task.Task**

Takes network from Neo4j database, calculates network centrality measures and updates each node in the database with new centrality attributes

**Parameters:**

- **date** (*datetime*) – Date used to label the outputs
- **output_type** (*str*) – type of record to be extracted from OpenAIRE API. Accepts “software”, “datasets”, “publications”, “ECProjects”

- **test** (*bool*) – run a shorter version of the task if in test mode

**date** = *<luigi.parameter.DateParameter object>*

**output_type** = *<luigi.parameter.Parameter object>*

**test** = *<luigi.parameter.BoolParameter object>*

**output**()[source]

Points to the output database engine where the task is marked as done. The luigi_table_updates table exists in test and production databases.

**run**()[source]

The task run method, to be overridden in a subclass.

See Task.run

## Cordis to Neo4j

Task for piping Cordis data from SQL to Neo4j.

*class* **CordisNeo4jTask**(**args*, ***kwargs*)[source]

Bases: **luigi.task.Task**

Task for piping Cordis data to neo4j

**test** = *<luigi.parameter.BoolParameter object>*

**date** = *<luigi.parameter.DateParameter object>*

**output**()[source]

Points to the output database engine where the task is marked as done. The luigi_table_updates table exists in test and production databases.

**run**()[source]

The task run method, to be overridden in a subclass.

See Task.run

*class* **RootTask**(**args*, ***kwargs*)[source]

Bases: **luigi.task.WrapperTask**

**production** = *<luigi.parameter.BoolParameter object>*

**requires**()[source]

The Tasks that this Task depends on.

A Task will only run if all of the Tasks that it requires are completed. If your Task does not require any other Tasks, then you don't need to override this method. Otherwise, a subclass can override this method to return a single Task, a list of Task instances, or a dict whose values are Task instances.

See Task.requires

## OpenAIRE to Neo4j

Pipe data directly from the OpenAIRE API to Neo4j by matching to Cordis projects already in Neo4j.

*class* **RootTask**(**args*, ***kwargs*)[source]

Bases: **luigi.task.WrapperTask**

The root task, which collects the supplied parameters and calls the SimpleTask.

**Parameters:**

- **date** (*datetime*) – Date used to label the outputs
- **output_type** (*str*) – type of record to be extracted from OpenAIRE API. Accepts "software", "datasets", "publications", "ECProjects"
- **production** (*bool*) – test mode or production mode

**date** = *<luigi.parameter.DateParameter object>*

**output_type** = *<luigi.parameter.Parameter object>*

**production** = *<luigi.parameter.BoolParameter object>*

**requires**()[source]

Call the task to run before this in the pipeline.

*class* **OpenAireToNeo4jTask**(**args*, ***kwargs*)[source]

Bases: **luigi.task.Task**

Takes OpenAIRE entities from MySQL database and writes them into Neo4j database

**Parameters:**

- **date** (*datetime*) – Date used to label the outputs
- **output_type** (*str*) – type of record to be extracted from OpenAIRE API. Accepts “software”, “datasets”, “publications”, “ECProjects”
- **test** (*bool*) – run a shorter version of the task if in test mode

**date** = *<luigi.parameter.DateParameter object>*

**output_type** = *<luigi.parameter.Parameter object>*

**test** = *<luigi.parameter.BoolParameter object>*

**output**()[source]

Points to the output database engine where the task is marked as done. The luigi_table_updates table exists in test and production databases.

**run**()[source]

The task run method, to be overridden in a subclass.

See Task.run

# Ontologies and schemas

## Tier 0

Raw data collections (“tier 0”) in the production system do not adhere to a fixed schema or ontology, but instead have a schema which is very close to the raw data. Modifications to field names tend to be quite basic, such as lowercase and removal of whitespace in favour of a single underscore.

## Tier 1

Processed data (“tier 1”) is intended for public consumption, using a common ontology. The convention we use is as follows:

- Field names are composed of up to three terms: a `firstName`, `middleName` and `lastName`
- Each term (e.g. `firstName`) is written in lowerCamelCase.
- `firstName` terms correspond to a restricted set of basic quantities.
- `middleName` terms correspond to a restricted set of modifiers (e.g. adjectives) which add nuance to the `firstName` term. Note, the special `middleName` term `of` is reserved as the default value in case no `middleName` is specified.
- `lastName` terms correspond to a restricted set of entity types.

Valid examples are `date_start_project` and `title_of_project`.

Tier 0 fields are implictly excluded from tier 1 if they are missing from the schema_transformation file. Tier 1 schema field names are applied via nesta.packages.decorator.schema_transform

# Scripts

A set of helper scripts for the batching system.

Note that this directory is required to sit in $PATH. By convention, all executables in this directory start with nesta_ so that our developers know where to find them.

## nesta_prepare_batch

Collect a batchable run.py file, including dependencies and an automaticlly generated requirements file; which is all zipped up and sent to AWS S3 for batching. This script is executed automatically in luigihacks.autobatch.AutoBatchTask.run.

**Parameters**:

- **BATCHABLE_DIRECTORY**: The path to the directory containing the batchable run.py file.
- **ARGS**: Space-separated-list of files or directories to include in the zip file, for example imports.

## nesta_docker_build

Build a docker environment and register it with the AWS ECS container repository.

**Parameters**:

- **DOCKER_RECIPE**: A docker recipe. See docker_recipes/ for a good idea of how to build a new environment.