In [ ]:
```
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
!pip install ydata-profiling
```

```
Collecting ydata-profiling
  Downloading ydata_profiling-4.18.1-py2.py3-none-any.whl.metadata (22
kB)
Requirement already satisfied: scipy<1.17,>=1.8 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling)
(1.16.3)
Requirement already satisfied: pandas!=1.4.0,<3.0,>1.5 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.2.2)
Requirement already satisfied: matplotlib<=3.10,>=3.5 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling)
(3.10.0)
Requirement already satisfied: pydantic<3,>=2 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling)
(2.12.3)
Requirement already satisfied: PyYAML<6.1,>=6.0.3 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling) (6.0.3)
Requirement already satisfied: jinja2<3.2,>=3.1.6 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling) (3.1.6)
Collecting visions<0.8.2,>=0.7.5 (from visions[type_image_path]
<0.8.2,>=0.7.5->ydata-profiling)
  Downloading visions-0.8.1-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: numpy<2.4,>=1.22 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.0.2)
Collecting minify-html>=0.15.0 (from ydata-profiling)
  Downloading minify_html-0.18.1-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Collecting filetype>=1.0.0 (from ydata-profiling)
  Downloading filetype-1.2.0-py2.py3-none-any.whl.metadata (6.5 kB)
Collecting phik<0.13,>=0.12.5 (from ydata-profiling)
  Downloading phik-0.12.5-cp312-cp312-
manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl.metadata (5.6 kB)
Requirement already satisfied: requests<3,>=2.32.0 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling)
(2.32.4)
Requirement already satisfied: tqdm<5,>=4.66.3 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling)
(4.67.1)
Requirement already satisfied: seaborn<0.14,>=0.10.1 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling)
(0.13.2)
Collecting multimethod<2,>=1.4 (from ydata-profiling)
  Downloading multimethod-1.12-py3-none-any.whl.metadata (9.6 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling)
(0.14.6)
Requirement already satisfied: typeguard<5,>=4 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling) (4.4.4)
```

```
Collecting imagehash==4.3.2 (from ydata-profiling)
  Downloading ImageHash-4.3.2-py2.py3-none-any.whl.metadata (8.4 kB)
Requirement already satisfied: wordcloud>=1.9.4 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling) (1.9.5)
Collecting dacite<2,>=1.9 (from ydata-profiling)
  Downloading dacite-1.9.2-py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: numba<0.63,>=0.60 in
/usr/local/lib/python3.12/dist-packages (from ydata-profiling)
(0.60.0)
Requirement already satisfied: PyWavelets in
/usr/local/lib/python3.12/dist-packages (from imagehash==4.3.2->ydata-
profiling) (1.9.0)
Requirement already satisfied: pillow in
/usr/local/lib/python3.12/dist-packages (from imagehash==4.3.2->ydata-
profiling) (11.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2<3.2,>=3.1.6-
>ydata-profiling) (3.0.3)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5-
>ydata-profiling) (1.3.3)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5-
>ydata-profiling) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5-
>ydata-profiling) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5-
>ydata-profiling) (1.4.9)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5-
>ydata-profiling) (25.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5-
>ydata-profiling) (3.3.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5-
>ydata-profiling) (2.9.0.post0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.12/dist-packages (from numba<0.63,>=0.60-
>ydata-profiling) (0.43.0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.12/dist-packages (from pandas!=1.4.0,<3.0,>1.5-
>ydata-profiling) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.12/dist-packages (from pandas!=1.4.0,<3.0,>1.5-
>ydata-profiling) (2025.3)
Requirement already satisfied: joblib>=0.14.1 in
/usr/local/lib/python3.12/dist-packages (from phik<0.13,>=0.12.5-
>ydata-profiling) (1.5.3)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.12/dist-packages (from pydantic<3,>=2->ydata-
profiling) (0.7.0)
Requirement already satisfied: pydantic-core==2.41.4 in
/usr/local/lib/python3.12/dist-packages (from pydantic<3,>=2->ydata-
profiling) (2.41.4)
```

```
Requirement already satisfied: typing-extensions>=4.14.1 in
/usr/local/lib/python3.12/dist-packages (from pydantic<3,>=2->ydata-
profiling) (4.15.0)
Requirement already satisfied: typing-inspection>=0.4.2 in
/usr/local/lib/python3.12/dist-packages (from pydantic<3,>=2->ydata-
profiling) (0.4.2)
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests<3,>=2.32.0-
>ydata-profiling) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.12/dist-packages (from requests<3,>=2.32.0-
>ydata-profiling) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.12/dist-packages (from requests<3,>=2.32.0-
>ydata-profiling) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.12/dist-packages (from requests<3,>=2.32.0-
>ydata-profiling) (2026.1.4)
Requirement already satisfied: patsy>=0.5.6 in
/usr/local/lib/python3.12/dist-packages (from statsmodels<1,>=0.13.2-
>ydata-profiling) (1.0.2)
Requirement already satisfied: attrs>=19.3.0 in
/usr/local/lib/python3.12/dist-packages (from visions<0.8.2,>=0.7.5-
>visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling) (25.4.0)
Requirement already satisfied: networkx>=2.4 in
/usr/local/lib/python3.12/dist-packages (from visions<0.8.2,>=0.7.5-
>visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling) (3.6.1)
Collecting puremagic (from visions<0.8.2,>=0.7.5-
>visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling)
  Downloading puremagic-1.30-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7-
>matplotlib<=3.10,>=3.5->ydata-profiling) (1.17.0)
Downloading ydata_profiling-4.18.1-py2.py3-none-any.whl (400 kB)
 [2K    [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [0m
 [32m400.4/400.4 kB [0m  [31m9.1 MB/s [0m eta  [36m0:00:00 [0m
 [?25hDownloading ImageHash-4.3.2-py2.py3-none-any.whl (296 kB)
 [2K    [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [0m
 [32m296.7/296.7 kB [0m  [31m20.1 MB/s [0m eta  [36m0:00:00 [0m
 [?25hDownloading dacite-1.9.2-py3-none-any.whl (16 kB)
Downloading filetype-1.2.0-py2.py3-none-any.whl (19 kB)
Downloading minify_html-0.18.1-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.1 MB)
 [2K    [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [0m  [32m3.1/3.1
MB [0m  [31m39.3 MB/s [0m eta  [36m0:00:00 [0m
 [?25hDownloading multimethod-1.12-py3-none-any.whl (10 kB)
Downloading phik-0.12.5-cp312-cp312-
manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (679 kB)
 [2K    [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [0m
 [32m679.7/679.7 kB [0m  [31m41.8 MB/s [0m eta  [36m0:00:00 [0m
 [?25hDownloading visions-0.8.1-py3-none-any.whl (105 kB)
 [2K    [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [0m
 [32m105.4/105.4 kB [0m  [31m8.3 MB/s [0m eta  [36m0:00:00 [0m
 [?25hDownloading puremagic-1.30-py3-none-any.whl (43 kB)
 [2K    [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ [0m  [32m43.3/43.3
kB [0m  [31m3.6 MB/s [0m eta  [36m0:00:00 [0m
 [?25hInstalling collected packages: puremagic, minify-html, filetype,
```

```
multimethod, dacite, imagehash, visions, phik, ydata-profiling
Successfully installed dacite-1.9.2 filetype-1.2.0 imagehash-4.3.2
minify-html-0.18.1 multimethod-1.12 phik-0.12.5 puremagic-1.30
visions-0.8.1 ydata-profiling-4.18.1
```

```
---------------------------------------------------------------------
--------
NameError                                 Traceback (most recent
call last)
/tmp/ipython-input-967197979.py in <cell line: 0>()
      2 from ydata_profiling import ProfileReport
      3
----> 4 profile = ProfileReport(df_scopus, title="Scopus CT
Dataset", explorative=True)
      5 profile

NameError: name 'df_scopus' is not defined
```

In [ ]:
```
import pandas as pd
from ydata_profiling import ProfileReport

BASE_PATH = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/"
```

In [ ]:
```
df_scopus = pd.read_csv(
    BASE_PATH + "scopus_export_CT.csv",
    low_memory=False
)

print(df_scopus.shape)
df_scopus.head()
```

```
(12192, 45)
```

	Authors	Author full names	Author(s) ID	Title	Year	S
0	Wang, Y.	Wang, Yang (57208730125)	57208730125	Effects of troubleshooting robotics learning o...	2026	Thi Ski Cre
1	Lin, Y.; Zhang, Y.; Yang, Y.; Pan, S.; Ren, X....	Lin, Yuru (57281795200); Zhang, Yi (5895719550...	57281795200; 58957195500; 57164390600; 6020965...	Facilitating computational thinking with AI: A...	2026	Thi Ski Cre
2	Hsu, T.-C.; Hsu, T.-P.	Hsu, Tingchia (35173046500);	35173046500; 58366049000	Effects of game-based learning	2026	Thi Ski Cre

	Authors	Author full names	Author(s) ID	Title	Year	S
		Hsu, Taiping (583...		integrated with...		
3	Aksoy, B.D.; Mumcu, F.K.; Cantürk Günhan, B.C.	Aksoy, Behiye Dinçer (60177502400); Mumcu, Fil...	60177502400; 13410584100; 36815607700	Unveiling the nexus: Computational thinking an...	2026	Thi Ski Cre
4	van Bergen, R.; Huebotter, J.; A.; Lanillos, P.	van Bergen, Ruben S. (55502596000); Huebotter,...	55502596000; 57901993200; 60247114700; 2407652...	Object-centric proto-symbolic behavioural reas...	2026	Neu Net

5 rows × 45 columns

In [158]:
```
profile_scopus = ProfileReport(
    df_scopus,
    title="Profiling — Scopus Export CT",
    explorative=True,
    minimal=False
)

profile_scopus

# Define la ruta (asegúrate de que la carpeta 'Analisis' exista o usa la raíz)
BASE_PATH

# Guardar el reporte
profile_scopus.to_file(BASE_PATH+"reporte_scopus.html")

print(f"Reporte guardado exitosamente en: {BASE_PATH}")

profile_scopus.to_file("reporte_scopus.html")
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]


    0%|          | 0/46 [00:00<?, ?it/s] [A
    2%||         | 1/46 [00:01<00:49,  1.09s/it] [A
    4%|▌         | 2/46 [00:01<00:29,  1.51it/s] [A
    7%|▊         | 3/46 [00:02<00:26,  1.64it/s] [A
```

```
  9%|▉         | 4/46 [00:02<00:27,  1.52it/s] [A
 15%|█▍        | 7/46 [00:02<00:11,  3.37it/s] [A
 20%|█▉        | 9/46 [00:03<00:08,  4.59it/s] [A
 22%|██        | 10/46 [00:03<00:07,  5.03it/s] [A
 24%|██▏       | 11/46 [00:03<00:06,  5.25it/s] [A
 26%|██▍       | 12/46 [00:03<00:05,  5.90it/s] [A
 28%|██▋       | 13/46 [00:04<00:08,  3.82it/s] [A
 30%|██▉       | 14/46 [00:04<00:12,  2.57it/s] [A
 33%|███▏      | 15/46 [00:05<00:17,  1.81it/s] [A
 35%|███▍      | 16/46 [00:08<00:33,  1.11s/it] [A
 37%|███▋      | 17/46 [00:14<01:15,  2.60s/it] [A
 54%|█████▍    | 25/46 [00:18<00:20,  1.04it/s] [A
 57%|█████▋    | 26/46 [00:20<00:23,  1.16s/it] [A
 65%|██████▌   | 30/46 [00:21<00:11,  1.43it/s] [A
 70%|██████▉   | 32/46 [00:21<00:07,  1.76it/s] [A
 74%|███████▍  | 34/46 [00:21<00:05,  2.24it/s] [A
 78%|███████▊  | 36/46 [00:21<00:03,  2.80it/s] [A
 85%|████████▍ | 39/46 [00:21<00:01,  3.86it/s] [A
 89%|████████▉ | 41/46 [00:22<00:01,  4.62it/s] [A
 93%|█████████▎| 43/46 [00:22<00:00,  5.20it/s] [A
100%|██████████| 46/46 [00:22<00:00,  2.04it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]

Reporte guardado exitosamente en: /content/drive/My Drive/Colab
Notebooks/PEC3-Visualizacion-UOC/

Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

In [ ]:

```
df_wos = pd.read_csv(
    BASE_PATH + "dataframe_wos.csv",
    low_memory=False
)

print(df_wos.shape)
df_wos.head()
```

```
(5142, 73)
```

	Unnamed: 0	Publication Type	Authors	Book Authors	Book Editors	Book Group Authors
0	0	C	WeiguoZou	NaN	Kim, Y	NaN

	Unnamed: 0	Publication Type	Authors	Book Authors	Book Editors	Book Group Authors	
1	1	B	Repenning, A; Basawapatna, AR; Escherle, NA	NaN	Rich, PJ; Hodges, CB	NaN	
2	2	C	Shih, WC	NaN	NaN	Assoc Comp Machinery	
3	3	C	Wu, SY	NaN	NaN	IEEE	
4	4	C	Lu, CJ; Zhang, S; Chen, XQ	NaN	Chang, T	NaN	

5 rows × 73 columns

In [157]:

```
profile_wos = ProfileReport(
    df_wos,
    title="Profiling – WoS Export CT",
    explorative=True
)

profile_wos
# Define la ruta (asegúrate de que la carpeta 'Analisis' exista o usa la
raíz)
BASE_PATH

# Guardar el reporte
profile_wos.to_file(BASE_PATH+"reporte_wos.html")

print(f"Reporte guardado exitosamente en: {BASE_PATH}")

profile_wos.to_file("reporte_wos.html")
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]


    0%|          | 0/74 [00:00<?, ?it/s] [A
    1%||          | 1/74 [00:00<00:12,  5.63it/s] [A
    4%|▌         | 3/74 [00:01<00:30,  2.33it/s] [A
    9%|▉         | 7/74 [00:02<00:24,  2.73it/s] [A
   14%|█▍        | 10/74 [00:03<00:20,  3.19it/s] [A
   19%|█▉        | 14/74 [00:03<00:11,  5.36it/s] [A
   22%|██▏       | 16/74 [00:03<00:10,  5.50it/s] [A
   24%|██▍       | 18/74 [00:03<00:09,  6.13it/s] [A
   27%|██▋       | 20/74 [00:04<00:08,  6.72it/s] [A
   30%|██▉       | 22/74 [00:08<00:36,  1.44it/s] [A
   31%|███       | 23/74 [00:13<01:08,  1.35s/it] [A
   38%|███▊      | 28/74 [00:13<00:29,  1.57it/s] [A
   41%|████      | 30/74 [00:13<00:22,  1.92it/s] [A
   43%|████▎     | 32/74 [00:13<00:18,  2.30it/s] [A
   54%|█████▍    | 40/74 [00:14<00:06,  5.34it/s] [A
   58%|█████▊    | 43/74 [00:14<00:04,  6.40it/s] [A
   62%|██████▏   | 46/74 [00:14<00:03,  7.46it/s] [A
   66%|██████▌   | 49/74 [00:14<00:02,  8.95it/s] [A
   70%|███████   | 52/74 [00:14<00:01, 11.02it/s] [A
   74%|███████▍  | 55/74 [00:14<00:01, 11.26it/s] [A
   77%|███████▋  | 57/74 [00:15<00:01, 12.03it/s] [A
   80%|███████▉  | 59/74 [00:15<00:01, 12.04it/s] [A
   82%|████████▏ | 61/74 [00:15<00:01, 11.31it/s] [A
   88%|████████▊ | 65/74 [00:15<00:00, 13.76it/s] [A
   92%|█████████▏| 68/74 [00:15<00:00, 12.33it/s] [A
   96%|█████████▌| 71/74 [00:16<00:00, 14.27it/s] [A
  100%|██████████| 74/74 [00:16<00:00,  4.49it/s]


Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]


Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]


Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]


Reporte guardado exitosamente en: /content/drive/My Drive/Colab
Notebooks/PEC3-Visualizacion-UOC/


Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

In [ ]:

```
# ============================================================
# PASO 2 (parte 1): Construir df_master_base.csv (Scopus + WoS)
# - Cargar Scopus + WoS
# - Normalizar DOI
# - Seleccionar columnas (lista blanca)
# - Unir (outer join por DOI)
# - Guardar df_master_base.csv en Drive
# ============================================================

import os
```

```
import re
import pandas as pd
import numpy as np
from google.colab import drive

# --- Montar Drive (si ya está montado, no pasa nada) ---
drive.mount("/content/drive")

BASE_DIR = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/"
SCOPUS_PATH = os.path.join(BASE_DIR, "scopus_export_CT.csv")
WOS_PATH    = os.path.join(BASE_DIR, "dataframe_wos.csv")
OUT_PATH    = os.path.join(BASE_DIR, "df_master_base.csv")

# ------------------------------------------------------------
# 1) Utilidades: normalización de DOI + limpieza básica
# ------------------------------------------------------------
DOI_PREFIX_RE = re.compile(r"^(https?://(dx\.)?doi\.org/)",
flags=re.IGNORECASE)

def normalize_doi(x: object) -> str:
    """Normaliza DOI:
    - str -> lower
    - quita prefijo https://doi.org/
    - trim espacios
    - elimina caracteres raros al final (.,; )
    """
    if pd.isna(x):
        return np.nan
    s = str(x).strip()
    if not s:
        return np.nan
    s = DOI_PREFIX_RE.sub("", s)         # quita prefijo
    s = s.strip().lower()
    s = s.rstrip(" .;,")                 # quita puntuación final común
    # algunos exports traen 'doi:10....'
    if s.startswith("doi:"):
        s = s.replace("doi:", "", 1).strip()
    return s if s else np.nan

def safe_col(df: pd.DataFrame, col: str) -> pd.Series:
    """Devuelve columna si existe; si no, columna NaN."""
    return df[col] if col in df.columns else pd.Series([np.nan]*len(df),
index=df.index)

# ------------------------------------------------------------
# 2) Cargar Scopus + WoS
# ------------------------------------------------------------
df_scopus = pd.read_csv(SCOPUS_PATH, low_memory=False)
df_wos    = pd.read_csv(WOS_PATH, low_memory=False)

print("Scopus shape:", df_scopus.shape)
print("WoS shape:", df_wos.shape)

# ------------------------------------------------------------
# 3) Identificar columna DOI en cada dataset (ajusta si difiere)
#    - Scopus suele traer "DOI"
```

```
#    - WoS suele traer "DOI"
# ------------------------------------------------------------
SCOPUS_DOI_COL = "DOI"
WOS_DOI_COL    = "DOI"

# Si tu export de Scopus usa otra columna, descomenta e indica:
# SCOPUS_DOI_COL = "DOI"  # o "DOI Link" (no recomendado), etc.

# ------------------------------------------------------------
# 4) Lista blanca de columnas a conservar (base bibliográfica)
#    Nota: mantenemos nombres "canónicos" en el master base
# ------------------------------------------------------------
# --- Scopus (ajusta nombres si tu export difiere) ---
SCOPUS_KEEP = {
    "doi": SCOPUS_DOI_COL,
    "title": "Title",
    "year": "Year",
    "journal": "Source title",
    "document_type": "Document Type",
    "authors": "Authors",
    "author_keywords": "Author Keywords",
    "index_keywords": "Index Keywords",
    "abstract": "Abstract",
    "cited_by_scopus": "Cited by",
    "references_count_scopus": "References",
    "publisher_scopus": "Publisher",
    "language_scopus": "Language of Original Document",
    "affiliations_scopus": "Affiliations",
    "country_scopus": "Affiliation Country",
}

# --- WoS (según profiling: 73 vars) ---
WOS_KEEP = {
    "doi": WOS_DOI_COL,
    "title_wos": "Article Title",
    "year_wos": "Publication Year",
    "journal_wos": "Source Title",
    "document_type_wos": "Document Type",
    "publication_type_wos": "Publication Type",
    "authors_wos": "Authors",
    "author_full_names_wos": "Author Full Names",
    "author_keywords_wos": "Author Keywords",
    "keywords_plus_wos": "Keywords Plus",
    "abstract_wos": "Abstract",
    "times_cited_wos_core": "Times Cited, WoS Core",
    "times_cited_wos_all": "Times Cited, All Databases",
    "cited_reference_count_wos": "Cited Reference Count",
    "wos_categories": "WoS Categories",
    "research_areas": "Research Areas",
    "publisher_wos": "Publisher",
    "language_wos": "Language",
    "affiliations_wos": "Affiliations",
    "addresses_wos": "Addresses",
}

# ------------------------------------------------------------
# 5) Construir vistas filtradas + normalizar DOI
```

```
# ------------------------------------------------------------
# Scopus
df_s = pd.DataFrame({k: safe_col(df_scopus, v) for k, v in
SCOPUS_KEEP.items()})
df_s["doi_norm"] = df_s["doi"].apply(normalize_doi)
df_s = df_s.drop(columns=["doi"])
df_s["source_scopus"] = True

# WoS
df_w = pd.DataFrame({k: safe_col(df_wos, v) for k, v in WOS_KEEP.items()})
df_w["doi_norm"] = df_w["doi"].apply(normalize_doi)
df_w = df_w.drop(columns=["doi"])
df_w["source_wos"] = True

# ------------------------------------------------------------
# 6) Deduplicación dentro de cada fuente por doi_norm
#    Regla: quedarse con el registro con más información (menos nulos)
# ------------------------------------------------------------
def dedup_by_doi(df: pd.DataFrame, doi_col: str = "doi_norm") ->
pd.DataFrame:
    df = df.copy()
    df["_nulls"] = df.isna().sum(axis=1)
    df = df.sort_values([doi_col, "_nulls"], ascending=[True, True])
    df = df.drop_duplicates(subset=[doi_col], keep="first")
    return df.drop(columns=["_nulls"])

df_s = dedup_by_doi(df_s)
df_w = dedup_by_doi(df_w)

print("Scopus after dedup:", df_s.shape)
print("WoS after dedup:", df_w.shape)

# ------------------------------------------------------------
# 7) Merge Scopus + WoS (outer) por doi_norm
# ------------------------------------------------------------
df_master_base = df_s.merge(df_w, on="doi_norm", how="outer", suffixes=("",
"_dup"))

# Flags de cobertura (útiles)
df_master_base["has_scopus"] =
df_master_base["source_scopus"].fillna(False)
df_master_base["has_wos"]    = df_master_base["source_wos"].fillna(False)

# ------------------------------------------------------------
# 8) Crear columnas canónicas (prioridad Scopus, fallback WoS)
#    (Esto deja el master base listo para enriquecer después)
# ------------------------------------------------------------
def coalesce(a, b):
    return a.where(~a.isna(), b)

# Año canónico
df_master_base["year"] = coalesce(df_master_base.get("year"),
df_master_base.get("year_wos"))

# Título canónico
df_master_base["title"] = coalesce(df_master_base.get("title"),
df_master_base.get("title_wos"))
```

```
# Journal canónico
df_master_base["journal"] = coalesce(df_master_base.get("journal"),
df_master_base.get("journal_wos"))

# Document type canónico (si Scopus no, usa WoS)
df_master_base["document_type"] =
coalesce(df_master_base.get("document_type"),
df_master_base.get("document_type_wos"))

# Citas canónicas (no definitivas; Dimensions luego puede ser "fuente de
verdad")
df_master_base["cited_by"] =
coalesce(df_master_base.get("cited_by_scopus"),
df_master_base.get("times_cited_wos_all"))

# ------------------------------------------------------------
# 9) Limpieza final + orden columnas
# ------------------------------------------------------------
# Normalizar year a numérico entero donde sea posible
df_master_base["year"] = pd.to_numeric(df_master_base["year"],
errors="coerce").astype("Int64")

# Columna DOI final (sin prefijos)
df_master_base.rename(columns={"doi_norm": "doi"}, inplace=True)

# Orden sugerido (las demás columnas quedan al final)
front_cols = [
    "doi", "title", "year", "journal", "document_type",
    "cited_by", "has_scopus", "has_wos"
]
other_cols = [c for c in df_master_base.columns if c not in front_cols]
df_master_base = df_master_base[front_cols + other_cols]

# ------------------------------------------------------------
# 10) Guardar
# ------------------------------------------------------------
df_master_base.to_csv(OUT_PATH, index=False)
print("✅ Guardado:", OUT_PATH)
print("df_master_base shape:", df_master_base.shape)

# Vista rápida
df_master_base.head(3)
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
Scopus shape: (12192, 45)
WoS shape: (5142, 73)
Scopus after dedup: (10107, 16)
WoS after dedup: (4192, 21)
✅ Guardado: /content/drive/My Drive/Colab Notebooks/PEC3-
Visualizacion-UOC/df_master_base.csv
df_master_base shape: (10831, 39)
```

	doi	title	year	journal	doc
0	10.1002/(sici)1096-9128(199601)8:1<47::aid-cpe...	Benchmarking the computation and communication...	1996	Concurrency Practice and Experience	Arti
1	10.1002/(sici)1097-0193(1999)8:2/3<128::aid-hb...	Computational modeling of high-level cognition...	1999	Human Brain Mapping	Cor pap
2	10.1002/(sici)1097-0363(199706)24:12<1321::aid...	Parallel computation of incompressible flows w...	1997	International Journal for Numerical Methods in...	Arti

3 rows × 39 columns

In [ ]:
```
profile_df_master_base = ProfileReport(
    df_master_base,
    title="Profiling — df_master_base CT",
    explorative=True
)

profile_df_master_base

# Define la ruta (asegúrate de que la carpeta 'Analisis' exista o usa la raíz)
BASE_PATH

# Guardar el reporte
profile_df_master_base.to_file(BASE_PATH+"profile_df_master_base.html")

print(f"Reporte guardado exitosamente en: {BASE_PATH}")

profile_scopus.to_file("profile_df_master_base.html")
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]


  0%|          | 0/39 [00:00<?, ?it/s] [A
  3%|▎         | 1/39 [00:00<00:15,  2.50it/s] [A
  5%|▌         | 2/39 [00:00<00:11,  3.23it/s] [A
 10%|▉         | 4/39 [00:00<00:06,  5.80it/s] [A
 23%|██▎       | 9/39 [00:01<00:03,  7.76it/s] [A
 28%|██▊       | 11/39 [00:04<00:14,  1.96it/s] [A
 31%|███       | 12/39 [00:11<00:42,  1.56s/it] [A
 36%|███▌      | 14/39 [00:11<00:26,  1.08s/it] [A
 38%|███▊      | 15/39 [00:11<00:21,  1.11it/s] [A
```

```
 44%|████      | 17/39 [00:12<00:15,  1.42it/s] [A
 56%|█████     | 22/39 [00:12<00:05,  3.01it/s] [A
 64%|██████    | 25/39 [00:12<00:03,  3.97it/s] [A
 69%|██████    | 27/39 [00:12<00:02,  4.42it/s] [A
 74%|███████   | 29/39 [00:14<00:03,  2.57it/s] [A
100%|██████████| 39/39 [00:14<00:00,  2.63it/s]


Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]


Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]


Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]


Reporte guardado exitosamente en: /content/drive/My Drive/Colab
Notebooks/PEC3-Visualizacion-UOC/


Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

In [ ]:

```
import os
import re
import pandas as pd
import numpy as np
from google.colab import drive

drive.mount("/content/drive")

BASE_DIR = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/"
IN_PATH  = os.path.join(BASE_DIR, "df_master_base.csv")
OUT_PATH = os.path.join(BASE_DIR, "df_master_base_clean.csv")

df = pd.read_csv(IN_PATH, low_memory=False)
print("Loaded:", df.shape)

# ------------------------------------------------------------
# 1) LISTA BLANCA (conservar)
# ------------------------------------------------------------
KEEP_COLS = [
    # Identificación
    "doi", "has_scopus", "has_wos",

    # Canónicas
    "title", "year", "journal", "document_type",

    # Impacto base
    "cited_by", "cited_by_scopus",
    "times_cited_wos_core", "times_cited_wos_all",
    "cited_reference_count_wos", "references_count_scopus",

    # Clasificación WoS
    "wos_categories", "research_areas",

    # Texto / keywords (para NLP)
    "abstract", "abstract_wos",
```

```
    "author_keywords", "author_keywords_wos",
    "index_keywords", "keywords_plus_wos",

    # Autores / afiliaciones / geo (si lo usarás)
    "authors", "authors_wos", "author_full_names_wos",
    "affiliations_scopus", "affiliations_wos",
    "addresses_wos",

    # Publisher / language
    "publisher_scopus", "publisher_wos",
    "language_scopus", "language_wos",
]

KEEP_COLS_EXIST = [c for c in KEEP_COLS if c in df.columns]

# ------------------------------------------------------------
# 2) LISTA NEGRA POR PATRONES
# ------------------------------------------------------------
DROP_PATTERNS = [
    r"\bbook\b", r"\bseries\b",
    r"\bmeeting\b", r"\bconference\b",
    r"\bproceedings\b", r"\bevent\b",

    r"date of export", r"web of science record", r"\brecord\b",
    r"doi link",
    r"researcher id", r"\borcid\b", r"email",

    r"highly cited status", r"hot paper status",

    r"start page", r"end page", r"article number", r"part number",
]

def matches_any_pattern(colname: str) -> bool:
    s = colname.lower()
    return any(re.search(p, s) for p in DROP_PATTERNS)

drop_by_pattern = [c for c in df.columns if matches_any_pattern(c)]

# ------------------------------------------------------------
# 3) Aplicar keep + drop patterns
# ------------------------------------------------------------
df_clean = df[KEEP_COLS_EXIST].copy()
df_clean = df_clean.drop(columns=[c for c in df_clean.columns if c in
drop_by_pattern], errors="ignore")
print("After keep+pattern drop:", df_clean.shape)

# ------------------------------------------------------------
# 4) Renombrar columnas a estándar
# ------------------------------------------------------------
RENAME_MAP = {
    "cited_by_scopus": "scopus_citations",
    "times_cited_wos_core": "wos_citations_core",
    "times_cited_wos_all": "wos_citations_all",
    "cited_reference_count_wos": "wos_reference_count",
    "references_count_scopus": "scopus_reference_count",

    "research_areas": "wos_research_areas",
```

```
    "author_keywords": "author_keywords_scopus",
    "index_keywords": "index_keywords_scopus",
    "abstract": "abstract_scopus",

    "author_keywords_wos": "author_keywords_wos",
    "keywords_plus_wos": "keywords_plus_wos",
    "abstract_wos": "abstract_wos",
}
df_clean = df_clean.rename(columns={k: v for k, v in RENAME_MAP.items() if
k in df_clean.columns})

# ------------------------------------------------------------
# 5) Tipos
# ------------------------------------------------------------
if "year" in df_clean.columns:
    df_clean["year"] = pd.to_numeric(df_clean["year"],
errors="coerce").astype("Int64")

for c in ["cited_by", "scopus_citations", "wos_citations_core",
"wos_citations_all"]:
    if c in df_clean.columns:
        df_clean[c] = pd.to_numeric(df_clean[c], errors="coerce")

# ------------------------------------------------------------
# 6) Columnas canónicas: publisher y language
# ------------------------------------------------------------
def coalesce(a, b):
    return a.where(~a.isna(), b)

if "publisher_scopus" in df_clean.columns or "publisher_wos" in
df_clean.columns:
    a = df_clean["publisher_scopus"] if "publisher_scopus" in
df_clean.columns else pd.Series([np.nan]*len(df_clean))
    b = df_clean["publisher_wos"]    if "publisher_wos" in df_clean.columns
else pd.Series([np.nan]*len(df_clean))
    df_clean["publisher"] = coalesce(a, b)

if "language_scopus" in df_clean.columns or "language_wos" in
df_clean.columns:
    a = df_clean["language_scopus"] if "language_scopus" in
df_clean.columns else pd.Series([np.nan]*len(df_clean))
    b = df_clean["language_wos"]    if "language_wos" in df_clean.columns
else pd.Series([np.nan]*len(df_clean))
    df_clean["language"] = coalesce(a, b)

# ------------------------------------------------------------
# 7) Ordenar columnas
# ------------------------------------------------------------
ORDER = [
    "doi", "title", "year", "journal", "document_type",
    "publisher", "language",
    "cited_by", "scopus_citations", "wos_citations_core",
"wos_citations_all",
    "scopus_reference_count", "wos_reference_count",
    "wos_categories", "wos_research_areas",
    "has_scopus", "has_wos",
```

```
    "authors", "authors_wos", "author_full_names_wos",
    "affiliations_scopus", "affiliations_wos", "addresses_wos",
    "author_keywords_scopus", "index_keywords_scopus",
"author_keywords_wos", "keywords_plus_wos",
    "abstract_scopus", "abstract_wos",
]
order_exist = [c for c in ORDER if c in df_clean.columns]
rest = [c for c in df_clean.columns if c not in order_exist]
df_clean = df_clean[order_exist + rest]

# ------------------------------------------------------------
# 8) Guardar
# ------------------------------------------------------------
df_clean.to_csv(OUT_PATH, index=False)
print("✅ Saved:", OUT_PATH)
print("Final shape:", df_clean.shape)
df_clean.head(3)
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
Loaded: (10831, 39)
After keep+pattern drop: (10831, 31)
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/df_master_base_clean.csv
Final shape: (10831, 33)
```

	doi	title	year	journal	doc
0	10.1002/(sici)1096-9128(199601)8:1<47::aid-cpe...	Benchmarking the computation and communication...	1996	Concurrency Practice and Experience	Arti
1	10.1002/(sici)1097-0193(1999)8:2/3<128::aid-hb...	Computational modeling of high-level cognition...	1999	Human Brain Mapping	Cor pap
2	10.1002/(sici)1097-0363(199706)24:12<1321::aid...	Parallel computation of incompressible flows w...	1997	International Journal for Numerical Methods in...	Arti

3 rows × 33 columns

In [ ]:

```
profile_df_clean = ProfileReport(
    df_clean,
    title="Profiling — df_clean CT",
    explorative=True
)

profile_df_clean
```

```
# Define la ruta (asegúrate de que la carpeta 'Analisis' exista o usa la
raíz)
BASE_PATH

# Guardar el reporte
profile_df_clean.to_file(BASE_PATH+"profile_df_clean.html")

print(f"Reporte guardado exitosamente en: {BASE_PATH}")

profile_scopus.to_file("profile_df_clean.html")
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]


    0%|          | 0/33 [00:00<?, ?it/s] [A
    3%|▌         | 1/33 [00:01<00:52,  1.63s/it] [A
    6%|▉         | 2/33 [00:01<00:24,  1.28it/s] [A
   12%|█▊        | 4/33 [00:02<00:10,  2.81it/s] [A
   24%|███▋      | 8/33 [00:02<00:03,  6.97it/s] [A
   36%|█████▍    | 12/33 [00:06<00:12,  1.64it/s] [A
   42%|██████▎   | 14/33 [00:06<00:08,  2.12it/s] [A
   64%|█████████▌  | 21/33 [00:06<00:02,  4.61it/s] [A
   73%|██████████▎ | 24/33 [00:07<00:02,  4.10it/s] [A
   82%|███████████▏ | 27/33 [00:09<00:01,  3.21it/s] [A
  100%|██████████████| 33/33 [00:15<00:00,  2.07it/s]


Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]


Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]


Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]

Reporte guardado exitosamente en: /content/drive/My Drive/Colab
Notebooks/PEC3-Visualizacion-UOC/


Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

In [ ]:
```
pd.read_csv(OUT_PATH, nrows=1).columns.tolist()
```

```
['doi',
 'title',
 'year',
 'journal',
 'document_type',
 'publisher',
 'language',
 'cited_by',
 'scopus_citations',
 'wos_citations_core',
 'wos_citations_all',
 'scopus_reference_count',
 'wos_reference_count',
```

```
        'wos_categories',
        'wos_research_areas',
        'has_scopus',
        'has_wos',
        'authors',
        'authors_wos',
        'author_full_names_wos',
        'affiliations_scopus',
        'affiliations_wos',
        'addresses_wos',
        'author_keywords_scopus',
        'index_keywords_scopus',
        'author_keywords_wos',
        'keywords_plus_wos',
        'abstract_scopus',
        'abstract_wos',
        'publisher_scopus',
        'publisher_wos',
        'language_scopus',
        'language_wos']
```

In [ ]:
```
import pandas as pd
import os

BASE_DIR = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/"
PATH = os.path.join(BASE_DIR, "df_master_base_clean.csv")

df = pd.read_csv(PATH, low_memory=False)

print("Rows:", len(df))
print("DOI nulls:", df["doi"].isna().sum())
print("DOI duplicates:", df["doi"].duplicated().sum())

for c in ["abstract_scopus", "abstract_wos"]:
    if c in df.columns:
        print(c, "missing %:", round(df[c].isna().mean()*100, 2))
```

```
Rows: 10831
DOI nulls: 1
DOI duplicates: 0
abstract_scopus missing %: 6.68
abstract_wos missing %: 62.04
```

In [ ]:
```
import json

def inspect_jsonl(path, n=10):
    with open(path) as f:
        for i, line in enumerate(f):
            if i >= n:
                break
            print(json.loads(line).keys())

inspect_jsonl(BASE_PATH + "altmetric_data.jsonl")
inspect_jsonl(BASE_PATH + "altmetrics.jsonl")
inspect_jsonl(BASE_PATH + "crossref_events.jsonl")
```

```
inspect_jsonl(BASE_PATH + "dimensions_data.jsonl")
inspect_jsonl(BASE_PATH + "lens_bulk_data.jsonl")
inspect_jsonl(BASE_PATH + "mendeley_data.jsonl")
inspect_jsonl(BASE_PATH + "OpenAlex.jsonl")
```

```
dict_keys(['query_doi', 'error', 'status'])
dict_keys(['query_doi', 'error', 'status'])
dict_keys(['query_doi', 'error', 'status'])
dict_keys(['title', 'doi', 'isbns', 'altmetric_jid', 'issns',
'journal', 'cohorts', 'context', 'authors', 'type', 'handles',
'pubdate', 'dimensions_publication_id', 'altmetric_id', 'schema',
'is_oa', 'publisher_subjects', 'cited_by_bluesky_count',
'cited_by_posts_count', 'cited_by_accounts_count', 'last_updated',
'score', 'history', 'url', 'added_on', 'published_on',
'scopus_subjects', 'readers', 'readers_count', 'images',
'details_url'])
dict_keys(['query_doi', 'error', 'status'])
dict_keys(['query_doi', 'error', 'status'])
dict_keys(['title', 'doi', 'pmid', 'isbns', 'altmetric_jid', 'issns',
'journal', 'cohorts', 'context', 'authors', 'type', 'handles',
'pubdate', 'epubdate', 'dimensions_publication_id', 'altmetric_id',
'schema', 'is_oa', 'publisher_subjects', 'cited_by_posts_count',
'cited_by_tweeters_count', 'cited_by_accounts_count', 'last_updated',
'score', 'history', 'url', 'added_on', 'published_on',
'scopus_subjects', 'readers', 'readers_count', 'images',
'details_url'])
dict_keys(['query_doi', 'error', 'status'])
dict_keys(['query_doi', 'error', 'status'])
dict_keys(['query_doi', 'error', 'status'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref', 'mendeley'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref', 'mendeley'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref', 'mendeley'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts',
'crossref', 'mendeley'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref', 'mendeley'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
```

```
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref', 'mendeley'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref', 'mendeley'])
dict_keys(['doi', 'timestamp', 'crossref_events'])
dict_keys(['doi', 'timestamp', 'crossref_events'])
dict_keys(['doi', 'timestamp', 'crossref_events'])
dict_keys(['doi', 'timestamp', 'crossref_events'])
dict_keys(['doi', 'timestamp', 'crossref_events'])
dict_keys(['doi', 'timestamp', 'crossref_events'])
dict_keys(['doi', 'timestamp', 'crossref_events'])
dict_keys(['doi', 'timestamp', 'crossref_events'])
dict_keys(['doi', 'timestamp', 'crossref_events'])
dict_keys(['abstract', 'authors', 'category_for', 'concepts', 'doi',
'id', 'journal', 'recent_citations', 'reference_ids',
'referenced_pubs', 'research_orgs', 'times_cited', 'title', 'type',
'year'])
dict_keys(['abstract', 'authors', 'category_for', 'concepts', 'doi',
'id', 'journal', 'recent_citations', 'reference_ids',
'referenced_pubs', 'research_orgs', 'times_cited', 'title', 'type',
'year'])
dict_keys(['abstract', 'authors', 'category_for', 'concepts', 'doi',
'id', 'journal', 'recent_citations', 'reference_ids',
'referenced_pubs', 'research_orgs', 'times_cited', 'title', 'type',
'year'])
dict_keys(['abstract', 'authors', 'category_for', 'concepts', 'doi',
'funders', 'id', 'journal', 'recent_citations', 'reference_ids',
'referenced_pubs', 'research_orgs', 'times_cited', 'title', 'type',
'year'])
dict_keys(['abstract', 'authors', 'category_for', 'concepts', 'doi',
'id', 'journal', 'recent_citations', 'reference_ids',
'referenced_pubs', 'research_orgs', 'times_cited', 'title', 'type',
'year'])
dict_keys(['abstract', 'authors', 'category_for', 'concepts', 'doi',
'id', 'journal', 'recent_citations', 'reference_ids',
'referenced_pubs', 'research_orgs', 'times_cited', 'title', 'type',
'year'])
dict_keys(['abstract', 'authors', 'category_for', 'concepts', 'doi',
'funders', 'id', 'journal', 'recent_citations', 'reference_ids',
'referenced_pubs', 'research_orgs', 'times_cited', 'title', 'type',
'year'])
dict_keys(['abstract', 'altmetric', 'authors', 'category_for',
'concepts', 'doi', 'funders', 'id', 'journal', 'recent_citations',
'reference_ids', 'referenced_pubs', 'research_orgs', 'times_cited',
'title', 'type', 'year'])
dict_keys(['abstract', 'authors', 'category_for', 'concepts', 'doi',
'id', 'journal', 'recent_citations', 'reference_ids',
'referenced_pubs', 'research_orgs', 'times_cited', 'title', 'type',
'year'])
dict_keys(['abstract', 'authors', 'category_for', 'concepts', 'doi',
'funders', 'id', 'journal', 'recent_citations', 'reference_ids',
'referenced_pubs', 'research_orgs', 'times_cited', 'title', 'type',
'year'])
dict_keys(['lens_id', 'title', 'publication_type', 'year_published',
'date_published', 'external_ids', 'open_access', 'authors', 'source',
```

```
'fields_of_study', 'volume', 'issue', 'references', 'abstract',
'start_page', 'end_page'])
dict_keys(['lens_id', 'title', 'publication_type', 'year_published',
'external_ids', 'open_access', 'authors', 'source', 'fields_of_study',
'volume', 'references', 'funding', 'scholarly_citations_count',
'start_page', 'end_page'])
dict_keys(['lens_id', 'title', 'publication_type', 'year_published',
'date_published', 'external_ids', 'open_access', 'authors', 'source',
'fields_of_study', 'volume', 'issue', 'references', 'funding',
'abstract', 'start_page', 'end_page'])
dict_keys(['lens_id', 'title', 'publication_type', 'year_published',
'date_published', 'external_ids', 'authors', 'source',
'fields_of_study', 'volume', 'issue', 'references', 'start_page',
'end_page'])
dict_keys(['lens_id', 'title', 'publication_type', 'year_published',
'date_published', 'external_ids', 'authors', 'source',
'fields_of_study', 'volume', 'issue', 'references', 'abstract',
'start_page', 'end_page'])
dict_keys(['lens_id', 'title', 'publication_type', 'year_published',
'date_published', 'external_ids', 'authors', 'source', 'references',
'funding', 'start_page', 'end_page'])
dict_keys(['lens_id', 'title', 'publication_type', 'year_published',
'date_published', 'external_ids', 'authors', 'source',
'fields_of_study', 'references', 'funding',
'scholarly_citations_count', 'start_page', 'end_page'])
dict_keys(['lens_id', 'title', 'publication_type', 'year_published',
'date_published', 'external_ids', 'open_access', 'authors', 'source',
'fields_of_study', 'volume', 'issue', 'references', 'abstract',
'scholarly_citations_count', 'start_page', 'end_page'])
dict_keys(['lens_id', 'title', 'publication_type', 'year_published',
'external_ids', 'authors', 'source', 'fields_of_study', 'volume',
'references', 'start_page', 'end_page'])
dict_keys(['lens_id', 'title', 'publication_type', 'year_published',
'date_published', 'external_ids', 'authors', 'source',
'fields_of_study', 'volume', 'issue', 'references', 'funding',
'abstract'])
dict_keys(['title', 'type', 'authors', 'year', 'source',
'identifiers', 'keywords', 'volume', 'month', 'publisher', 'day',
'id', 'file_attached', 'imported', 'link', 'reader_count',
'reader_count_by_academic_status', 'reader_count_by_user_role',
'reader_count_by_subject_area', 'reader_count_by_subdiscipline',
'group_count', 'has_pdf', 'open_access', 'crossref_member_id'])
dict_keys(['title', 'type', 'authors', 'year', 'source',
'identifiers', 'keywords', 'volume', 'month', 'publisher', 'day',
'id', 'file_attached', 'imported', 'link', 'reader_count',
'reader_count_by_academic_status', 'reader_count_by_user_role',
'reader_count_by_subject_area', 'reader_count_by_subdiscipline',
'group_count', 'has_pdf', 'open_access', 'crossref_member_id'])
dict_keys(['title', 'type', 'authors', 'year', 'source',
'identifiers', 'keywords', 'volume', 'month', 'publisher', 'day',
'id', 'file_attached', 'imported', 'link', 'reader_count',
'reader_count_by_academic_status', 'reader_count_by_user_role',
'reader_count_by_subject_area', 'reader_count_by_subdiscipline',
'group_count', 'has_pdf', 'open_access', 'crossref_member_id'])
dict_keys(['title', 'type', 'authors', 'year', 'source',
'identifiers', 'keywords', 'volume', 'month', 'publisher', 'day',
'id', 'file_attached', 'imported', 'link', 'reader_count',
```

```
'reader_count_by_academic_status', 'reader_count_by_user_role',
'reader_count_by_subject_area', 'reader_count_by_subdiscipline',
'group_count', 'has_pdf', 'open_access'])
dict_keys(['title', 'type', 'authors', 'year', 'source',
'identifiers', 'keywords', 'volume', 'month', 'publisher', 'day',
'id', 'file_attached', 'imported', 'link', 'reader_count',
'group_count', 'has_pdf', 'open_access'])
dict_keys(['title', 'type', 'authors', 'year', 'source',
'identifiers', 'keywords', 'volume', 'month', 'publisher', 'day',
'id', 'file_attached', 'imported', 'link', 'reader_count',
'group_count', 'has_pdf', 'open_access'])
dict_keys(['title', 'type', 'authors', 'year', 'source',
'identifiers', 'keywords', 'volume', 'month', 'publisher', 'day',
'id', 'file_attached', 'imported', 'link', 'reader_count',
'reader_count_by_academic_status', 'reader_count_by_user_role',
'reader_count_by_subject_area', 'reader_count_by_subdiscipline',
'group_count', 'has_pdf', 'open_access'])
dict_keys(['title', 'type', 'authors', 'year', 'source',
'identifiers', 'keywords', 'volume', 'month', 'publisher', 'day',
'id', 'file_attached', 'imported', 'link', 'reader_count',
'reader_count_by_academic_status', 'reader_count_by_user_role',
'reader_count_by_subject_area', 'reader_count_by_subdiscipline',
'group_count', 'has_pdf', 'open_access', 'crossref_member_id'])
dict_keys(['title', 'type', 'authors', 'year', 'source',
'identifiers', 'keywords', 'pages', 'volume', 'issue', 'month',
'publisher', 'day', 'id', 'file_attached', 'imported', 'link',
'reader_count', 'reader_count_by_academic_status',
'reader_count_by_user_role', 'reader_count_by_subject_area',
'reader_count_by_subdiscipline', 'group_count', 'has_pdf',
'open_access', 'crossref_member_id'])
dict_keys(['title', 'type', 'authors', 'year', 'source',
'identifiers', 'keywords', 'volume', 'month', 'publisher', 'day',
'id', 'file_attached', 'imported', 'link', 'reader_count',
'reader_count_by_academic_status', 'reader_count_by_user_role',
'reader_count_by_subject_area', 'reader_count_by_subdiscipline',
'group_count', 'has_pdf', 'open_access', 'crossref_member_id'])
dict_keys(['doi', 'timestamp', 'openalex', 'altmetric'])
dict_keys(['doi', 'timestamp', 'openalex', 'altmetric'])
dict_keys(['doi', 'timestamp', 'openalex', 'altmetric'])
dict_keys(['doi', 'timestamp', 'openalex', 'altmetric'])
dict_keys(['doi', 'timestamp', 'openalex', 'altmetric'])
dict_keys(['doi', 'timestamp', 'openalex', 'altmetric'])
dict_keys(['doi', 'timestamp', 'openalex', 'altmetric'])
dict_keys(['doi', 'timestamp', 'openalex', 'altmetric'])
dict_keys(['doi', 'timestamp', 'openalex', 'altmetric'])
dict_keys(['doi', 'timestamp', 'openalex', 'altmetric'])
```

In [ ]:

```
# =========================
# 0) Setup
# =========================
import os, json, re
import pandas as pd
import numpy as np
from google.colab import drive

drive.mount("/content/drive")
```

```
BASE_PATH = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/"
MASTER_PATH = os.path.join(BASE_PATH, "df_master_base_clean.csv")
ALTMETRIC_PATH = os.path.join(BASE_PATH, "altmetric_data.jsonl")  # <-- usa
este primero
OUT_CSV = os.path.join(BASE_PATH, "altmetric_by_doi.csv")

# -------------------------
# DOI normalizer (misma idea que antes)
# -------------------------
DOI_PREFIX_RE = re.compile(r"^(https?://(dx\.)?doi\.org/)",
flags=re.IGNORECASE)

def normalize_doi(x):
    if x is None:
        return None
    s = str(x).strip().lower()
    if not s or s == "nan":
        return None
    s = DOI_PREFIX_RE.sub("", s)
    if s.startswith("doi:"):
        s = s.replace("doi:", "", 1).strip()
    s = s.rstrip(" .;,")
    return s or None

# =========================
# 1) Cargar master y construir set de DOI
# =========================
df_master = pd.read_csv(MASTER_PATH, low_memory=False)
df_master["doi_norm"] = df_master["doi"].apply(normalize_doi)

master_dois = set(df_master["doi_norm"].dropna().unique())
print("Master rows:", len(df_master))
print("Master DOIs (non-null):", len(master_dois))

# =========================
# 2) Leer JSONL de Altmetric y quedarnos con registros válidos
#    - Filtra logs: {'query_doi','error','status'}
# =========================
rows = []
bad = 0
no_doi = 0
not_in_master = 0

# campos mínimos que queremos extraer (puedes ampliar luego)
def extract_altmetric_fields(obj):
    # doi puede venir en 'doi' o solo como 'query_doi'
    doi = normalize_doi(obj.get("doi") or obj.get("query_doi"))
    if not doi:
        return None

    # si es un log de error y no trae contenido, lo descartamos
    # (ojo: algunos objetos válidos podrían tener status pero también
datos; por eso validamos por 'title' o 'score' o 'altmetric_id')
    is_log_only = ("error" in obj and set(obj.keys()) <=
{"query_doi","error","status"})
    if is_log_only:
```

```
        return None

    # métricas principales
    out = {
        "doi": doi,
        "altmetric_id": obj.get("altmetric_id"),
        "score": obj.get("score"),
        "last_updated": obj.get("last_updated"),
        "published_on": obj.get("published_on"),
        "added_on": obj.get("added_on"),

        # conteos (algunos pueden no existir según versión)
        "cited_by_posts_count": obj.get("cited_by_posts_count"),
        "cited_by_accounts_count": obj.get("cited_by_accounts_count"),
        "cited_by_tweeters_count": obj.get("cited_by_tweeters_count"),
        "cited_by_bluesky_count": obj.get("cited_by_bluesky_count"),

        # Mendeley (viene dentro de Altmetric en tu caso)
        "readers_count": obj.get("readers_count"),
    }

    # title/journal opcional (útil para debugging, luego puedes eliminar)
    out["title_altmetric"] = obj.get("title")
    out["journal_altmetric"] = obj.get("journal")
    out["type_altmetric"] = obj.get("type")

    # history: suele venir como dict por fechas; guardamos resumen si
existe
    hist = obj.get("history")
    if isinstance(hist, dict):
        # algunos historiales incluyen '1d', '1w', '1m', '3m', '6m', '1y' o
fechas
        # aquí guardamos solo keys principales si existieran
        for k in ["1d","1w","1m","3m","6m","1y","at"]:
            if k in hist:
                out[f"history_{k}"] = hist.get(k)

    # context: a veces trae métricas por fuente; guardamos algunos si
existen
    ctx = obj.get("context")
    if isinstance(ctx, dict):
        # ejemplo: ctx puede incluir 'all' -> {'score':..., 'rank':...}
        all_ctx = ctx.get("all")
        if isinstance(all_ctx, dict):
            out["context_all_rank"] = all_ctx.get("rank")
            out["context_all_rank_pct"] = all_ctx.get("rank_pct")

    return out

with open(ALTMETRIC_PATH, "r", encoding="utf-8") as f:
    for line in f:
        line = line.strip()
        if not line:
            continue
        try:
            obj = json.loads(line)
        except Exception:
```

```
            bad += 1
            continue

        rec = extract_altmetric_fields(obj)
        if rec is None:
            # podría ser log-only o sin doi
            if "doi" not in obj and "query_doi" not in obj:
                no_doi += 1
            continue

        if rec["doi"] not in master_dois:
            not_in_master += 1
            # puedes decidir si guardarlo igual; por ahora lo guardamos
aparte? => lo ignoramos en el DF final
            continue

        rows.append(rec)

print("JSONL parse errors:", bad)
print("Records without DOI:", no_doi)
print("Altmetric records not in master:", not_in_master)
print("Altmetric valid records kept:", len(rows))

# =========================
# 3) Crear DataFrame, tipar y deduplicar por DOI
# =========================
alt_df = pd.DataFrame(rows)

# convertir numéricos
num_cols = [
    "score", "cited_by_posts_count", "cited_by_accounts_count",
    "cited_by_tweeters_count", "cited_by_bluesky_count", "readers_count",
    "context_all_rank", "context_all_rank_pct"
]
for c in num_cols:
    if c in alt_df.columns:
        alt_df[c] = pd.to_numeric(alt_df[c], errors="coerce")

# deduplicar: si hubiera más de un registro por DOI, quedarnos con el de
mayor score
if "score" in alt_df.columns:
    alt_df = alt_df.sort_values("score",
ascending=False).drop_duplicates(subset=["doi"], keep="first")
else:
    alt_df = alt_df.drop_duplicates(subset=["doi"], keep="first")

print("Altmetric DF after dedup:", alt_df.shape)

# =========================
# 4) Cobertura sobre master
# =========================
covered = set(alt_df["doi"].dropna().unique())
coverage_pct = len(covered) / len(master_dois) * 100
print(f"Coverage Altmetric over master DOIs: {len(covered)} /
{len(master_dois)} ({coverage_pct:.2f}%)")

# =========================
```

```
# 5) Guardar CSV
# ========================
alt_df.to_csv(OUT_CSV, index=False)
print("✅ Saved:", OUT_CSV)

alt_df.head(5)
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
Master rows: 10831
Master DOIs (non-null): 10830
JSONL parse errors: 0
Records without DOI: 0
Altmetric records not in master: 0
Altmetric valid records kept: 2798
Altmetric DF after dedup: (2798, 23)
Coverage Altmetric over master DOIs: 2798 / 10830 (25.84%)
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/altmetric_by_doi.csv
```

	doi	altmetric_id	score	last_updated	p
1979	10.1371/journal.pbio.2002050	17171592	752.50	1763806017	1.
1193	10.1073/pnas.2022340118	107805630	621.20	1763272862	1.
901	10.1371/journal.pcbi.1010285	132455735	426.65	1677253500	1.
1385	10.1016/j.ssci.2020.104866	82822248	398.85	1739650540	1.
2559	10.1146/annurev.neuro.24.1.167	1381428	386.50	1763916303	9.

5 rows × 23 columns

In [ ]:

```
import os
import pandas as pd
import numpy as np
import re

BASE_PATH = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/"
MASTER_PATH = os.path.join(BASE_PATH, "df_master_base_clean.csv")
ALTM_PATH   = os.path.join(BASE_PATH, "altmetric_by_doi.csv")
OUT_PATH    = os.path.join(BASE_PATH, "df_master_enriched_v1.csv")

# -------------------------
# DOI normalizer (igual que antes)
# -------------------------
DOI_PREFIX_RE = re.compile(r"^(https?://(dx\.)?doi\.org/)",
flags=re.IGNORECASE)

def normalize_doi(x):
    if pd.isna(x):
        return np.nan
    s = str(x).strip().lower()
```

```
    s = DOI_PREFIX_RE.sub("", s)
    if s.startswith("doi:"):
        s = s.replace("doi:", "", 1).strip()
    s = s.rstrip(" .;,")
    return s if s else np.nan

# ------------------------
# Load
# ------------------------
df_master = pd.read_csv(MASTER_PATH, low_memory=False)
alt_df    = pd.read_csv(ALTM_PATH, low_memory=False)

df_master["doi_norm"] = df_master["doi"].apply(normalize_doi)
alt_df["doi_norm"]    = alt_df["doi"].apply(normalize_doi)

# ------------------------
# Merge (left join: conservar universo master)
# ------------------------
df_enriched = df_master.merge(
    alt_df.drop(columns=["doi"], errors="ignore"),
    on="doi_norm",
    how="left",
    suffixes=("", "_alt")
)

# ------------------------
# Flags + cobertura
# ------------------------
df_enriched["has_altmetric"] = df_enriched["altmetric_id"].notna()

coverage = df_enriched["has_altmetric"].mean() * 100
print("Rows:", len(df_enriched))
print(f"Altmetric coverage in enriched DF: {coverage:.2f}%")

# ------------------------
# (Opcional recomendado) convertir epochs a fecha
# ------------------------
for col in ["last_updated", "published_on", "added_on"]:
    if col in df_enriched.columns:
        # epoch en segundos -> datetime
        df_enriched[col + "_dt"] = pd.to_datetime(df_enriched[col],
unit="s", errors="coerce")

# ------------------------
# Guardar
# ------------------------
df_enriched.drop(columns=["doi_norm"], errors="ignore").to_csv(OUT_PATH,
index=False)
print("✅ Saved:", OUT_PATH)

df_enriched.head(3)
```

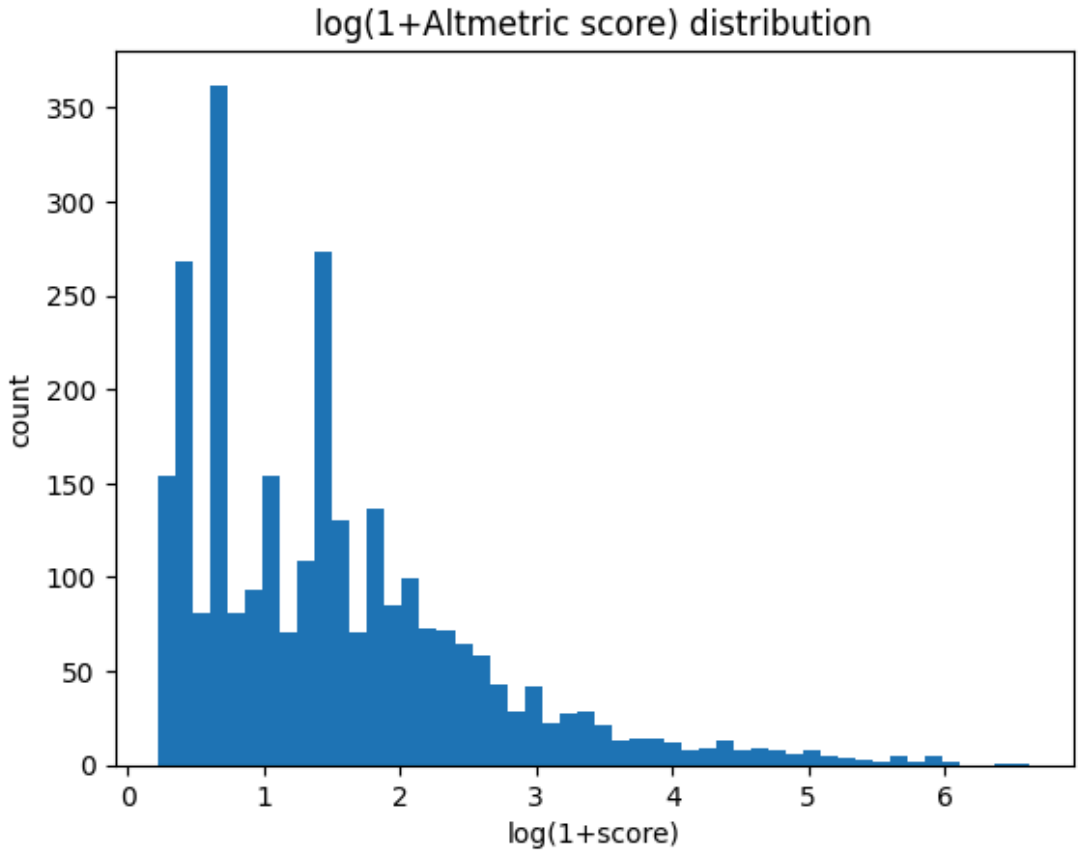
```
Rows: 10831
Altmetric coverage in enriched DF: 25.83%
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/df_master_enriched_v1.csv
```

	doi	title	year	journal	doc
0	10.1002/(sici)1096-9128(199601)8:1<47::aid-cpe...	Benchmarking the computation and communication...	1996	Concurrency Practice and Experience	Arti
1	10.1002/(sici)1097-0193(1999)8:2/3<128::aid-hb...	Computational modeling of high-level cognition...	1999	Human Brain Mapping	Cor pap
2	10.1002/(sici)1097-0363(199706)24:12<1321::aid...	Parallel computation of incompressible flows w...	1997	International Journal for Numerical Methods in...	Arti

3 rows × 60 columns

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import os

BASE_PATH = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"
ALTM_PATH = os.path.join(BASE_PATH, "altmetric_by_doi.csv")
alt_df = pd.read_csv(ALTM_PATH)

print("Rows:", len(alt_df))
print("Score missing %:", round(alt_df["score"].isna().mean()*100, 2))

# distribución score (log para ver cola)
scores = alt_df["score"].dropna().values
plt.figure()
plt.hist(np.log1p(scores), bins=50)
plt.title("log(1+Altmetric score) distribution")
plt.xlabel("log(1+score)")
plt.ylabel("count")
plt.show()

# top 10 por score
alt_df.sort_values("score", ascending=False)
[["doi","score","cited_by_posts_count","readers_count","title_altmetric"]].
head(10)
```

```
Rows: 2798
Score missing %: 0.0
```



	doi	score	cited_by_posts_count	readers_cou
0	10.1371/journal.pbio.2002050	752.50	1365	884
1	10.1073/pnas.2022340118	621.20	337	238
2	10.1371/journal.pcbi.1010285	426.65	111	28
3	10.1016/j.ssci.2020.104866	398.85	646	755
4	10.1146/annurev.neuro.24.1.167	386.50	147	15342
5	10.1038/ncomms13669	377.35	267	269

	doi	score	cited_by_posts_count	readers_cou
6	10.1371/journal.pcbi.1005871	361.10	661	643
7	10.1609/aaai.v38i16.29720	355.05	562	433
8	10.1038/s41586-025-08680-1	353.40	383	59
9	10.1007/s42113-024-00217-5	345.70	716	109

In [ ]:
```
inspect_jsonl(BASE_PATH + "altmetrics.jsonl", n=5)
```

```
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref', 'mendeley'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref', 'mendeley'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref', 'mendeley'])
dict_keys(['doi', 'title', 'year', 'journal', 'authors', 'abstract',
'author_keywords', 'index_keywords', 'openalex', 'concepts', 'lens',
'crossref'])
```

In [ ]:
```
import os, json, re
import pandas as pd
import numpy as np

BASE_PATH = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/"
MASTER_PATH = os.path.join(BASE_PATH, "df_master_base_clean.csv")
DIM_PATH    = os.path.join(BASE_PATH, "dimensions_data.jsonl")

OUT_BY_DOI  = os.path.join(BASE_PATH, "dimensions_by_doi.csv")
OUT_EDGES   = os.path.join(BASE_PATH, "dimensions_citation_edges.csv")
```

```
OUT_NODES   = os.path.join(BASE_PATH, "dimensions_nodes_papers.csv")  #
opcional

# -------------------------
# DOI normalizer
# -------------------------
DOI_PREFIX_RE = re.compile(r"^(https?://(dx\.)?doi\.org/)",
flags=re.IGNORECASE)

def normalize_doi(x):
    if x is None or (isinstance(x, float) and np.isnan(x)):
        return None
    s = str(x).strip().lower()
    if not s or s == "nan":
        return None
    s = DOI_PREFIX_RE.sub("", s)
    if s.startswith("doi:"):
        s = s.replace("doi:", "", 1).strip()
    s = s.rstrip(" .;,")
    return s or None

def safe_len(x):
    return len(x) if isinstance(x, (list, tuple)) else 0

def top_concepts(concepts, k=3):
    if not isinstance(concepts, list) or len(concepts) == 0:
        return None
    if all(isinstance(c, str) for c in concepts):
        return "; ".join(concepts[:k])
    names = []
    for c in concepts:
        if isinstance(c, dict):
            for key in ["display_name", "name", "concept", "label",
"title"]:
                if key in c and c[key]:
                    names.append(str(c[key]))
                    break
    return "; ".join(names[:k]) if names else None

def parse_journal(j):
    """journal puede venir como dict {'id':..., 'title':...} o string."""
    if isinstance(j, dict):
        return j.get("id"), j.get("title")
    return None, None

# -------------------------
# 1) Universo DOI master
# -------------------------
df_master = pd.read_csv(MASTER_PATH, low_memory=False)
df_master["doi_norm"] = df_master["doi"].apply(normalize_doi)
master_dois = set(df_master["doi_norm"].dropna().unique())

# para marcar si la referencia citada está dentro del universo CT
master_dois_lookup = set(master_dois)

print("Master rows:", len(df_master))
print("Master DOIs (non-null):", len(master_dois))
```

```
# ------------------------
# 2) Parse JSONL
# ------------------------
rows_by_doi = []
rows_edges  = []
rows_nodes  = []

bad_json = 0
log_only = 0
no_doi = 0
not_in_master = 0

with open(DIM_PATH, "r", encoding="utf-8") as f:
    for line in f:
        line = line.strip()
        if not line:
            continue
        try:
            obj = json.loads(line)
        except Exception:
            bad_json += 1
            continue

        # logs puros
        if "error" in obj and set(obj.keys()) <= {"query_doi", "error",
"status"}:
            log_only += 1
            continue

        doi = normalize_doi(obj.get("doi") or obj.get("query_doi"))
        if not doi:
            no_doi += 1
            continue
        if doi not in master_dois_lookup:
            not_in_master += 1
            continue

        # --- BY_DOI (métricas) ---
        concepts = obj.get("concepts")
        research_orgs = obj.get("research_orgs")
        funders = obj.get("funders")
        reference_ids = obj.get("reference_ids")
        referenced_pubs = obj.get("referenced_pubs")

        journal_id, journal_title = parse_journal(obj.get("journal"))

        rec = {
            "doi": doi,
            "dimensions_id": obj.get("id"),
            "dimensions_type": obj.get("type"),
            "times_cited": obj.get("times_cited"),
            "recent_citations": obj.get("recent_citations"),

            "n_concepts": safe_len(concepts),
            "top_concepts": top_concepts(concepts, k=3),
```

```
            "n_research_orgs": safe_len(research_orgs),
            "n_funders": safe_len(funders),

            "n_reference_ids": safe_len(reference_ids),
            "n_referenced_pubs": safe_len(referenced_pubs),

            "title_dimensions": obj.get("title"),
            "year_dimensions": obj.get("year"),
            "journal_id_dimensions": journal_id,
            "journal_title_dimensions": journal_title,
        }
        rows_by_doi.append(rec)

        # --- NODES (opcional, útil para red) ---
        rows_nodes.append({
            "doi": doi,
            "title": obj.get("title"),
            "year": obj.get("year"),
            "journal_title": journal_title,
            "times_cited": obj.get("times_cited"),
            "recent_citations": obj.get("recent_citations"),
        })

        # --- EDGES: citación (paper -> referencias) ---
        # Preferimos referenced_pubs si viene, si no reference_ids
        cited_list = referenced_pubs if isinstance(referenced_pubs, list)
and referenced_pubs else reference_ids

        if isinstance(cited_list, list) and cited_list:
            for cited in cited_list:
                # cited puede venir como:
                # - string id "pub.xxxx" o "doi:..." o DOI directo
                # - dict con campos
                cited_id = None
                cited_doi = None

                if isinstance(cited, str):
                    cited_id = cited
                    # si parece DOI, lo guardamos
                    cd = normalize_doi(cited)
                    if cd and "/" in cd and cd.startswith("10."):
                        cited_doi = cd

                elif isinstance(cited, dict):
                    cited_id = cited.get("id") or cited.get("pub_id") or
cited.get("reference_id")
                    cited_doi = normalize_doi(cited.get("doi")) if
cited.get("doi") else None
                    # si no hay doi pero hay external_ids
                    ext = cited.get("external_ids")
                    if cited_doi is None and isinstance(ext, dict) and
"doi" in ext:
                        cited_doi = normalize_doi(ext.get("doi"))

                in_master = (cited_doi in master_dois_lookup) if cited_doi
else False
```

```
                rows_edges.append({
                    "citing_doi": doi,
                    "cited_id": cited_id,
                    "cited_doi": cited_doi,
                    "in_master": in_master,
                    "source": "dimensions"
                })

print("JSON parse errors:", bad_json)
print("Log-only (error/status) skipped:", log_only)
print("Records without DOI:", no_doi)
print("Records not in master:", not_in_master)
print("Valid Dimensions records kept:", len(rows_by_doi))

# -------------------------
# 3) DataFrames + tipado + dedup
# -------------------------
dim_df = pd.DataFrame(rows_by_doi)
edges_df = pd.DataFrame(rows_edges)
nodes_df = pd.DataFrame(rows_nodes)

for c in ["times_cited", "recent_citations", "n_concepts",
"n_research_orgs",
          "n_funders", "n_reference_ids", "n_referenced_pubs",
"year_dimensions"]:
    if c in dim_df.columns:
        dim_df[c] = pd.to_numeric(dim_df[c], errors="coerce")

# dedup por DOI (si llega duplicado, quedarnos con mayor times_cited)
dim_df = dim_df.sort_values("times_cited",
ascending=False).drop_duplicates(subset=["doi"], keep="first")

# nodes dedup por DOI también
nodes_df = nodes_df.sort_values("times_cited",
ascending=False).drop_duplicates(subset=["doi"], keep="first")

print("Dimensions DF after dedup:", dim_df.shape)

covered = set(dim_df["doi"].dropna().unique())
coverage_pct = len(covered) / len(master_dois_lookup) * 100
print(f"Coverage Dimensions over master DOIs: {len(covered)} /
{len(master_dois_lookup)} ({coverage_pct:.2f}%)")

# -------------------------
# 4) Guardar outputs
# -------------------------
dim_df.to_csv(OUT_BY_DOI, index=False)
print("✅ Saved:", OUT_BY_DOI)

# edges puede ser muy grande: guarda igual y luego filtramos por in_master
o por top nodes
edges_df.to_csv(OUT_EDGES, index=False)
print("✅ Saved:", OUT_EDGES, "| edges:", len(edges_df))

nodes_df.to_csv(OUT_NODES, index=False)
print("✅ Saved:", OUT_NODES, "| nodes:", len(nodes_df))
```

```
dim_df.head(3), edges_df.head(3), nodes_df.head(3)
```

```
Master rows: 10831
Master DOIs (non-null): 10830
JSON parse errors: 0
Log-only (error/status) skipped: 0
Records without DOI: 0
Records not in master: 0
Valid Dimensions records kept: 10287
Dimensions DF after dedup: (10283, 15)
Coverage Dimensions over master DOIs: 10283 / 10830 (94.95%)
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/dimensions_by_doi.csv
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/dimensions_citation_edges.csv | edges: 280719
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/dimensions_nodes_papers.csv | nodes: 10283

(                                doi   dimensions_id dimensions_type
\
 9372  10.1146/annurev.neuro.24.1.167  pub.1011665436         article
 9207         10.1145/1118178.1118215  pub.1042177537         article
 9567      10.1037/0033-295x.99.1.122  pub.1027805977         article

      times_cited  recent_citations  n_concepts  \
 9372        11126             1428          35
 9207         6009             1624           5
 9567         3291              176          39

                                          top_concepts
n_research_orgs  \
 9372  cognitive control; prefrontal cortex; prefront...
1
 9207          computer scientists; computer; scientists
1
 9567  working memory capacity; working memory; memor...
1

      n_funders  n_reference_ids  n_referenced_pubs  \
 9372          9              175                175
 9207          0                0                  0
 9567          1               63                 63

                                       title_dimensions
year_dimensions  \
 9372  AN INTEGRATIVE THEORY OF PREFRONTAL CORTEX FUN...
2001
 9207                             Computational thinking
2006
 9567  A Capacity Theory of Comprehension: Individual...
1992

     journal_id_dimensions       journal_title_dimensions
 9372          jour.1087714  Annual Review of Neuroscience
 9207          jour.1079972      Communications of the ACM
 9567          jour.1017903           Psychological Review  ,
```

```
                        citing_doi        cited_id
cited_doi  \
 0  10.1016/j.lindif.2025.102846  pub.1109659555  10.1016/c2009-0-
02249-1
 1  10.1016/j.lindif.2025.102846  pub.1108496834
10.4324/9780415963572
 2  10.1016/j.lindif.2025.102846  pub.1192570487
10.3390/math13172828

    in_master      source
 0      False  dimensions
 1      False  dimensions
 2      False  dimensions  ,
                                    doi  \
 9372  10.1146/annurev.neuro.24.1.167
 9207         10.1145/1118178.1118215
 9567      10.1037/0033-295x.99.1.122

                                                  title  year  \
 9372  AN INTEGRATIVE THEORY OF PREFRONTAL CORTEX FUN...  2001
 9207                               Computational thinking  2006
 9567  A Capacity Theory of Comprehension: Individual...  1992

                       journal_title  times_cited  recent_citations
 9372  Annual Review of Neuroscience        11126              1428
 9207      Communications of the ACM         6009              1624
 9567           Psychological Review         3291               176  )
```

In [ ]:
```
edges_internal = edges_df[edges_df["in_master"] == True]
edges_internal.to_csv(
    os.path.join(BASE_PATH, "dimensions_edges_ct_internal.csv"),
    index=False
)

top_dois = set(nodes_df.sort_values("times_cited",
ascending=False).head(300)["doi"])
edges_top = edges_df[edges_df["citing_doi"].isin(top_dois)]
edges_top.to_csv(
    os.path.join(BASE_PATH, "dimensions_edges_top300.csv"),
    index=False
)
```

In [ ]:
```
import os, json, re
import pandas as pd
import numpy as np

BASE_PATH = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/"
MASTER_PATH = os.path.join(BASE_PATH, "df_master_base_clean.csv")
OA_PATH     = os.path.join(BASE_PATH, "OpenAlex.jsonl")

OUT_OA      = os.path.join(BASE_PATH, "openalex_by_doi.csv")
OUT_OA_EDGES= os.path.join(BASE_PATH, "openalex_citation_edges.csv")  #
opcional

DOI_PREFIX_RE = re.compile(r"^(https?://(dx\.)?doi\.org/)",
```

```
flags=re.IGNORECASE)

def normalize_doi(x):
    if x is None or (isinstance(x, float) and np.isnan(x)):
        return None
    s = str(x).strip().lower()
    if not s or s == "nan":
        return None
    s = DOI_PREFIX_RE.sub("", s)
    if s.startswith("doi:"):
        s = s.replace("doi:", "", 1).strip()
    s = s.rstrip(" .;,")
    return s or None

def safe_get(d, path, default=None):
    cur = d
    for p in path:
        if isinstance(cur, dict) and p in cur:
            cur = cur[p]
        else:
            return default
    return cur

def extract_top_concepts(concepts, k=5):
    """
    OpenAlex concepts suele ser lista de dicts con display_name y
score/level.
    Nos quedamos con top K por score si existe.
    """
    if not isinstance(concepts, list) or not concepts:
        return (0, None)
    # filtrar dicts válidos
    cands = []
    for c in concepts:
        if isinstance(c, dict):
            name = c.get("display_name") or c.get("name")
            score = c.get("score")
            if name:
                cands.append((name, score if isinstance(score,(int,float))
else -1))
    if not cands:
        return (len(concepts), None)
    # ordenar por score desc si existe
    cands.sort(key=lambda x: (x[1] if x[1] is not None else -1),
reverse=True)
    names = [n for n,_ in cands[:k]]
    return (len(concepts), "; ".join(names))

# Universo DOI master
df_master = pd.read_csv(MASTER_PATH, low_memory=False)
df_master["doi_norm"] = df_master["doi"].apply(normalize_doi)
master_dois = set(df_master["doi_norm"].dropna().unique())

print("Master DOIs:", len(master_dois))

rows = []
edges = []
```

```
bad = 0
no_doi = 0
not_in_master = 0
kept = 0
has_refs = 0

with open(OA_PATH, "r", encoding="utf-8") as f:
    for line in f:
        line = line.strip()
        if not line:
            continue
        try:
            obj = json.loads(line)
        except Exception:
            bad += 1
            continue

        doi = normalize_doi(obj.get("doi") or safe_get(obj,
["openalex","doi"]))
        if not doi:
            no_doi += 1
            continue
        if doi not in master_dois:
            not_in_master += 1
            continue

        oa = obj.get("openalex") if isinstance(obj.get("openalex"), dict)
else obj

        concepts = oa.get("concepts")
        n_concepts, top_concepts = extract_top_concepts(concepts, k=5)

        rec = {
            "doi": doi,
            "oa_id": oa.get("id"),
            "oa_type": oa.get("type"),
            "oa_language": oa.get("language"),
            "oa_publication_year": oa.get("publication_year"),
            "oa_publication_date": oa.get("publication_date"),
            "oa_cited_by_count": oa.get("cited_by_count"),
            "oa_referenced_works_count": oa.get("referenced_works_count"),

            "oa_is_oa": safe_get(oa, ["open_access","is_oa"]),
            "oa_status": safe_get(oa, ["open_access","oa_status"]),
            "oa_any_repo_fulltext": safe_get(oa,
["open_access","any_repository_has_fulltext"]),

            "oa_countries_distinct_count":
oa.get("countries_distinct_count"),
            "oa_institutions_distinct_count":
oa.get("institutions_distinct_count"),

            "oa_host_org_name": safe_get(oa,
["primary_location","source","host_organization_name"]),
            "oa_source_name": safe_get(oa,
["primary_location","source","display_name"]),
```

```
            "oa_n_concepts": n_concepts,
            "oa_top_concepts": top_concepts,
        }
        rows.append(rec)
        kept += 1

        # edges de citación (si existe)
        refs = oa.get("referenced_works")
        if isinstance(refs, list) and refs:
            has_refs += 1
            for rid in refs:
                edges.append({
                    "citing_doi": doi,
                    "cited_openalex_work_id": rid,
                    "source": "openalex"
                })

print("JSON parse errors:", bad)
print("Without DOI:", no_doi)
print("Not in master:", not_in_master)
print("Kept:", kept)
print("Records with referenced_works:", has_refs)
print("Edges (OpenAlex):", len(edges))

oa_df = pd.DataFrame(rows)
for c in
["oa_publication_year","oa_cited_by_count","oa_referenced_works_count",

"oa_countries_distinct_count","oa_institutions_distinct_count","oa_n_concep
ts"]:
    if c in oa_df.columns:
        oa_df[c] = pd.to_numeric(oa_df[c], errors="coerce")

# dedup por DOI quedándonos con mayor cited_by_count
if "oa_cited_by_count" in oa_df.columns:
    oa_df = oa_df.sort_values("oa_cited_by_count",
ascending=False).drop_duplicates("doi")
else:
    oa_df = oa_df.drop_duplicates("doi")

coverage = len(set(oa_df["doi"])) / len(master_dois) * 100
print(f"Coverage OpenAlex: {len(set(oa_df['doi']))} / {len(master_dois)}
({coverage:.2f}%)")

oa_df.to_csv(OUT_OA, index=False)
print("✅ Saved:", OUT_OA)

if edges:
    pd.DataFrame(edges).to_csv(OUT_OA_EDGES, index=False)
    print("✅ Saved:", OUT_OA_EDGES)

oa_df.head(5)
```

```
Master DOIs: 10830
JSON parse errors: 0
Without DOI: 0
Not in master: 0
```

```
Kept: 10830
Records with referenced_works: 9312
Edges (OpenAlex): 376622
Coverage OpenAlex: 10830 / 10830 (100.00%)
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/openalex_by_doi.csv
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/openalex_citation_edges.csv
```

	doi	oa_id	o
9872	10.1146/annurev.neuro.24.1.167	https://openalex.org/W2151137320	re
9709	10.1145/1118178.1118215	https://openalex.org/W2339183141	a
10060	10.1037/0033-295x.99.1.122	https://openalex.org/W2072875864	re
10082	10.1080/00031305.1988.10475524	https://openalex.org/W2113952909	a
6906	10.1109/access.2019.2921522	https://openalex.org/W2955897898	a

In [ ]:

```
import os, json, re
import pandas as pd
import numpy as np

# ========================
# 0) Paths
# ========================
BASE_PATH = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/"
MASTER_PATH   = os.path.join(BASE_PATH, "df_master_base_clean.csv")
MENDELEY_PATH = os.path.join(BASE_PATH, "mendeley_data.jsonl")
OUT_CSV       = os.path.join(BASE_PATH, "mendeley_by_doi.csv")

# ========================
# 1) DOI normalizer
# ========================
DOI_PREFIX_RE = re.compile(r"^(https?://(dx\.)?doi\.org/)",
flags=re.IGNORECASE)

def normalize_doi(x):
    if x is None or (isinstance(x, float) and np.isnan(x)):
        return None
```

```
    s = str(x).strip().lower()
    if not s or s == "nan":
        return None
    s = DOI_PREFIX_RE.sub("", s)
    if s.startswith("doi:"):
        s = s.replace("doi:", "", 1).strip()
    s = s.rstrip(" .;,")
    return s or None

# ========================
# 2) Helpers
# ========================
def safe_topk_from_dict(d, k=3):
    """Devuelve 'key1:val1; key2:val2' para top-k por val."""
    if not isinstance(d, dict) or not d:
        return None
    items = []
    for kk, vv in d.items():
        try:
            vnum = float(vv)
        except Exception:
            continue
        items.append((kk, vnum))
    if not items:
        return None
    items.sort(key=lambda x: x[1], reverse=True)
    return "; ".join([f"{kk}:{int(vv) if vv.is_integer() else vv}" for kk,
vv in items[:k]])

def safe_sum_dict(d):
    """Suma valores numéricos de un dict."""
    if not isinstance(d, dict) or not d:
        return np.nan
    total = 0.0
    ok = False
    for _, vv in d.items():
        try:
            total += float(vv)
            ok = True
        except Exception:
            pass
    return total if ok else np.nan

# ========================
# 3) Universo DOI master
# ========================
df_master = pd.read_csv(MASTER_PATH, low_memory=False)
df_master["doi_norm"] = df_master["doi"].apply(normalize_doi)
master_dois = set(df_master["doi_norm"].dropna().unique())

print("Master rows:", len(df_master))
print("Master DOIs (non-null):", len(master_dois))

# ========================
# 4) Parse Mendeley JSONL
# ========================
rows = []
```

```
bad_json = 0
no_doi = 0
not_in_master = 0

with open(MENDELEY_PATH, "r", encoding="utf-8") as f:
    for line in f:
        line = line.strip()
        if not line:
            continue
        try:
            obj = json.loads(line)
        except Exception:
            bad_json += 1
            continue

        # DOI puede venir en varias ubicaciones
        # Tu muestra: obj['identifiers'] suele tener doi
        doi = (
            normalize_doi(obj.get("doi")) or
            normalize_doi(obj.get("query_doi")) or
            normalize_doi((obj.get("identifiers") or {}).get("doi"))
        )

        if not doi:
            no_doi += 1
            continue
        if doi not in master_dois:
            not_in_master += 1
            continue

        # campos típicos por tu estructura
        reader_count = obj.get("reader_count")
        has_pdf = obj.get("has_pdf")
        open_access = obj.get("open_access")

        by_status = obj.get("reader_count_by_academic_status")  # dict
        by_subject = obj.get("reader_count_by_subject_area")    # dict
        by_subdisc = obj.get("reader_count_by_subdiscipline")   # dict
        by_role = obj.get("reader_count_by_user_role")          # dict

        rec = {
            "doi": doi,
            "mendeley_id": obj.get("id"),
            "mendeley_type": obj.get("type"),
            "mendeley_year": obj.get("year"),
            "mendeley_source": obj.get("source"),
            "mendeley_publisher": obj.get("publisher"),

            "mendeley_reader_count": reader_count,
            "mendeley_group_count": obj.get("group_count"),

            "mendeley_has_pdf": has_pdf,
            "mendeley_open_access": open_access,

            # resúmenes top-k (muy útiles para tooltips o filtros)
            "top_academic_status": safe_topk_from_dict(by_status, k=3),
            "top_subject_area": safe_topk_from_dict(by_subject, k=3),
```

```
            "top_subdiscipline": safe_topk_from_dict(by_subdisc, k=3),
            "top_user_role": safe_topk_from_dict(by_role, k=3),

            # sums (por si reader_count viene nulo pero los breakdowns
existen)
            "sum_by_academic_status": safe_sum_dict(by_status),
            "sum_by_subject_area": safe_sum_dict(by_subject),
        }

        rows.append(rec)

print("JSON parse errors:", bad_json)
print("Records without DOI:", no_doi)
print("Records not in master:", not_in_master)
print("Valid Mendeley records kept:", len(rows))

mend_df = pd.DataFrame(rows)

# =========================
# 5) Tipado + dedup por DOI
# =========================
for c in ["mendeley_reader_count", "mendeley_group_count", "mendeley_year",
          "sum_by_academic_status", "sum_by_subject_area"]:
    if c in mend_df.columns:
        mend_df[c] = pd.to_numeric(mend_df[c], errors="coerce")

# dedup: conservar el de mayor reader_count
if "mendeley_reader_count" in mend_df.columns:
    mend_df = mend_df.sort_values("mendeley_reader_count",
ascending=False).drop_duplicates(subset=["doi"], keep="first")
else:
    mend_df = mend_df.drop_duplicates(subset=["doi"], keep="first")

print("Mendeley DF after dedup:", mend_df.shape)

covered = set(mend_df["doi"].dropna().unique())
coverage_pct = len(covered) / len(master_dois) * 100
print(f"Coverage Mendeley over master DOIs: {len(covered)} /
{len(master_dois)} ({coverage_pct:.2f}%)")

# =========================
# 6) Guardar
# =========================
mend_df.to_csv(OUT_CSV, index=False)
print("✅ Saved:", OUT_CSV)

mend_df.head(5)
```

```
Master rows: 10831
Master DOIs (non-null): 10830
JSON parse errors: 0
Records without DOI: 0
Records not in master: 0
Valid Mendeley records kept: 10774
Mendeley DF after dedup: (10774, 16)
Coverage Mendeley over master DOIs: 10774 / 10830 (99.48%)
```

✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/mendeley_by_doi.csv

	doi	mendeley_id	mendeley_type	mende
9859	10.1146/annurev.neuro.24.1.167	b3f2d556-6d25-3231-8ccf-592caf1fc86a	generic	2001
9696	10.1145/1118178.1118215	8e16cc3b-f618-31ea-9b04-6f6f78e6209e	generic	2006
9203	10.1126/science.1192788	6c911538-7146-386a-9601-9e23f99e2a66	generic	2011
8438	10.1007/978-94-017-9762-7_1	2c721a90-d476-3d2b-8aad-282d7fe6ebb1	book_section	2015
4215	10.1016/b978-0-12-818630-5.13078-7	e3c24d84-58ea-34a9-813a-09da9ccd568d	book_section	2022

In [ ]:

```
import os, json, re
import pandas as pd
import numpy as np

# ========================
# Paths
# ========================
BASE_PATH = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/"
MASTER_PATH   = os.path.join(BASE_PATH, "df_master_base_clean.csv")
CROSSREF_PATH = os.path.join(BASE_PATH, "crossref_events.jsonl")
OUT_CSV       = os.path.join(BASE_PATH, "crossref_by_doi.csv")

# ========================
# DOI normalizer
# ========================
DOI_PREFIX_RE = re.compile(r"^(https?://(dx\.)?doi\.org/)",
flags=re.IGNORECASE)

def normalize_doi(x):
    if x is None or (isinstance(x, float) and np.isnan(x)):
        return None
    s = str(x).strip().lower()
    if not s or s == "nan":
```

```
        return None
    s = DOI_PREFIX_RE.sub("", s)
    if s.startswith("doi:"):
        s = s.replace("doi:", "", 1).strip()
    s = s.rstrip(" .;,")
    return s or None

# ========================
# Master DOIs
# ========================
df_master = pd.read_csv(MASTER_PATH, low_memory=False)
df_master["doi_norm"] = df_master["doi"].apply(normalize_doi)
master_dois = set(df_master["doi_norm"].dropna().unique())

print("Master DOIs:", len(master_dois))

# ========================
# Parse Crossref JSONL
# ========================
rows = []
bad_json = 0
no_doi = 0
not_in_master = 0

with open(CROSSREF_PATH, "r", encoding="utf-8") as f:
    for line in f:
        line = line.strip()
        if not line:
            continue
        try:
            obj = json.loads(line)
        except Exception:
            bad_json += 1
            continue

        doi = normalize_doi(obj.get("doi") or obj.get("query_doi"))
        if not doi:
            no_doi += 1
            continue
        if doi not in master_dois:
            not_in_master += 1
            continue

        events = obj.get("crossref_events")
        n_events = len(events) if isinstance(events, list) else 0

        rows.append({
            "doi": doi,
            "crossref_events_count": n_events
        })

print("JSON parse errors:", bad_json)
print("Records without DOI:", no_doi)
print("Records not in master:", not_in_master)
print("Valid Crossref records:", len(rows))

cr_df = pd.DataFrame(rows)
```

```
# Dedup (por seguridad)
cr_df = cr_df.drop_duplicates(subset=["doi"], keep="first")

coverage = len(cr_df) / len(master_dois) * 100
print(f"Coverage Crossref over master DOIs: {len(cr_df)} /
{len(master_dois)} ({coverage:.2f}%)")

cr_df.to_csv(OUT_CSV, index=False)
print("✅ Saved:", OUT_CSV)

cr_df.head()
```

```
Master DOIs: 10830
JSON parse errors: 0
Records without DOI: 0
Records not in master: 0
Valid Crossref records: 10830
Coverage Crossref over master DOIs: 10830 / 10830 (100.00%)
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/crossref_by_doi.csv
```

	doi	crossref_events_count
0	10.1016/j.tsc.2025.102068	0
1	10.1016/j.tsc.2025.102070	0
2	10.1016/j.tsc.2025.102056	0
3	10.1016/j.tsc.2025.102049	0
4	10.1016/j.neunet.2025.108407	0

In [ ]:
```
import os, re
import pandas as pd
import numpy as np

BASE_PATH = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/"

MASTER_IN   = os.path.join(BASE_PATH, "df_master_base_clean.csv")
OA_IN       = os.path.join(BASE_PATH, "openalex_by_doi.csv")
DIM_IN      = os.path.join(BASE_PATH, "dimensions_by_doi.csv")
ALTM_IN     = os.path.join(BASE_PATH, "altmetric_by_doi.csv")
MEND_IN     = os.path.join(BASE_PATH, "mendeley_by_doi.csv")

OUT_CSV     = os.path.join(BASE_PATH, "df_master_enriched.csv")
OUT_PARQUET = os.path.join(BASE_PATH, "df_master_enriched.parquet")

# -------------------------
# DOI normalizer (por si acaso)
# -------------------------
DOI_PREFIX_RE = re.compile(r"^(https?://(dx\.)?doi\.org/)",
flags=re.IGNORECASE)

def normalize_doi(x):
```

```
    if x is None or (isinstance(x, float) and np.isnan(x)):
        return None
    s = str(x).strip().lower()
    if not s or s == "nan":
        return None
    s = DOI_PREFIX_RE.sub("", s)
    if s.startswith("doi:"):
        s = s.replace("doi:", "", 1).strip()
    s = s.rstrip(" .;,")
    return s or None

# ------------------------
# 1) Load master base
# ------------------------
df = pd.read_csv(MASTER_IN, low_memory=False)
df["doi"] = df["doi"].apply(normalize_doi)

print("Base master:", df.shape, "DOIs:", df["doi"].nunique(dropna=True))

# ------------------------
# 2) Load enrichment tables
# ------------------------
oa = pd.read_csv(OA_IN, low_memory=False)
oa["doi"] = oa["doi"].apply(normalize_doi)

dim = pd.read_csv(DIM_IN, low_memory=False)
dim["doi"] = dim["doi"].apply(normalize_doi)

altm = pd.read_csv(ALTM_IN, low_memory=False)
altm["doi"] = altm["doi"].apply(normalize_doi)

mend = pd.read_csv(MEND_IN, low_memory=False)
mend["doi"] = mend["doi"].apply(normalize_doi)

# ------------------------
# 3) Rename columns (prefijos consistentes)
# ------------------------
# OpenAlex
oa_rename = {
    "oa_id": "openalex_id",
    "oa_type": "openalex_type",
    "oa_language": "openalex_language",
    "oa_publication_year": "openalex_publication_year",
    "oa_publication_date": "openalex_publication_date",
    "oa_cited_by_count": "openalex_cited_by_count",
    "oa_referenced_works_count": "openalex_referenced_works_count",
    "oa_is_oa": "openalex_is_oa",
    "oa_status": "openalex_oa_status",
    "oa_any_repo_fulltext": "openalex_any_repo_fulltext",
    "oa_countries_distinct_count": "openalex_countries_distinct_count",
    "oa_institutions_distinct_count":
"openalex_institutions_distinct_count",
    "oa_host_org_name": "openalex_host_org_name",
    "oa_source_name": "openalex_source_name",
    "oa_n_concepts": "openalex_n_concepts",
    "oa_top_concepts": "openalex_top_concepts",
}
```

```
oa = oa.rename(columns={k:v for k,v in oa_rename.items() if k in
oa.columns})

# Dimensions
dim_rename = {
    "dimensions_id": "dimensions_id",
    "dimensions_type": "dimensions_type",
    "times_cited": "dimensions_times_cited",
    "recent_citations": "dimensions_recent_citations",
    "n_reference_ids": "dimensions_n_reference_ids",
    "n_referenced_pubs": "dimensions_n_referenced_pubs",
    "n_concepts": "dimensions_n_concepts",
    "top_concepts": "dimensions_top_concepts",
    "journal_title_dimensions": "dimensions_journal_title",
}
dim = dim.rename(columns={k:v for k,v in dim_rename.items() if k in
dim.columns})

# Altmetric
altm_rename = {
    "altmetric_id": "altmetric_id",
    "score": "altmetric_score",
    "last_updated": "altmetric_last_updated",
    "published_on": "altmetric_published_on",
    "added_on": "altmetric_added_on",
    "cited_by_posts_count": "altmetric_posts_count",
    "cited_by_accounts_count": "altmetric_accounts_count",
    "cited_by_tweeters_count": "altmetric_tweeters_count",
    "cited_by_bluesky_count": "altmetric_bluesky_count",
    "type_altmetric": "altmetric_type",
    "context_all_rank": "altmetric_context_all_rank",
    "context_all_rank_pct": "altmetric_context_all_rank_pct",
}
altm = altm.rename(columns={k:v for k,v in altm_rename.items() if k in
altm.columns})

# Mendeley
mend_rename = {
    "mendeley_reader_count": "mendeley_reader_count",
    "mendeley_group_count": "mendeley_group_count",
    "mendeley_has_pdf": "mendeley_has_pdf",
    "mendeley_open_access": "mendeley_open_access",
    "top_academic_status": "mendeley_top_academic_status",
    "top_subject_area": "mendeley_top_subject_area",
}
mend = mend.rename(columns={k:v for k,v in mend_rename.items() if k in
mend.columns})

# -------------------------
# 4) Reduce columns in enrichment tables (evitar inflación)
# -------------------------
oa_keep = ["doi"] + [c for c in oa.columns if c.startswith("openalex_")]
dim_keep = ["doi"] + [c for c in dim.columns if
c.startswith("dimensions_")]
altm_keep = ["doi"] + [c for c in altm.columns if
c.startswith("altmetric_")]
mend_keep = ["doi"] + [
```

```
    "mendeley_reader_count", "mendeley_group_count",
    "mendeley_has_pdf", "mendeley_open_access",
    "mendeley_top_academic_status", "mendeley_top_subject_area"
]

oa = oa[ [c for c in oa_keep if c in oa.columns] ].copy()
dim = dim[ [c for c in dim_keep if c in dim.columns] ].copy()
altm = altm[ [c for c in altm_keep if c in altm.columns] ].copy()
mend = mend[ [c for c in mend_keep if c in mend.columns] ].copy()

# -------------------------
# 5) Merge (left joins por DOI)
# -------------------------
df_enriched = df.merge(oa, on="doi", how="left") \
                .merge(dim, on="doi", how="left") \
                .merge(altm, on="doi", how="left") \
                .merge(mend, on="doi", how="left")

print("Enriched:", df_enriched.shape)

# -------------------------
# 6) Tipado mínimo (para evitar strings raros)
# -------------------------
num_cols = [
    "year",
    "scopus_citations", "wos_citations_core", "wos_citations_all",
    "dimensions_times_cited", "dimensions_recent_citations",
    "openalex_cited_by_count", "openalex_referenced_works_count",
    "altmetric_score", "altmetric_posts_count", "altmetric_accounts_count",
    "altmetric_tweeters_count", "altmetric_bluesky_count",
    "mendeley_reader_count", "mendeley_group_count",
    "openalex_countries_distinct_count",
"openalex_institutions_distinct_count",
    "openalex_n_concepts"
]
for c in num_cols:
    if c in df_enriched.columns:
        df_enriched[c] = pd.to_numeric(df_enriched[c], errors="coerce")

# bools
for c in ["openalex_is_oa", "openalex_any_repo_fulltext",
"mendeley_has_pdf", "mendeley_open_access"]:
    if c in df_enriched.columns:
        df_enriched[c] = df_enriched[c].astype("boolean")

# -------------------------
# 7) Orden de columnas (layout final)
# -------------------------
ORDER = [
    "doi", "title", "year", "journal", "document_type",
    "publisher", "language", "has_scopus", "has_wos",

    # abstracts / keywords
    "abstract_scopus", "abstract_wos",
    "author_keywords_scopus", "index_keywords_scopus",
    "author_keywords_wos", "keywords_plus_wos",
```

```
    # impacto base
    "scopus_citations", "wos_citations_core", "wos_citations_all",
    "dimensions_times_cited", "dimensions_recent_citations",
    "openalex_cited_by_count", "openalex_referenced_works_count",

    # OA / geo / conceptos
    "openalex_publication_date", "openalex_type",
    "openalex_is_oa", "openalex_oa_status",
    "openalex_countries_distinct_count",
"openalex_institutions_distinct_count",
    "openalex_n_concepts", "openalex_top_concepts",
    "openalex_host_org_name", "openalex_source_name",

    # altmetrics
    "altmetric_score", "altmetric_posts_count", "altmetric_accounts_count",
    "altmetric_tweeters_count", "altmetric_bluesky_count",
    "altmetric_published_on", "altmetric_added_on",
"altmetric_last_updated",

    # mendeley
    "mendeley_reader_count", "mendeley_group_count",
    "mendeley_has_pdf", "mendeley_open_access",
    "mendeley_top_academic_status", "mendeley_top_subject_area",
]

order_exist = [c for c in ORDER if c in df_enriched.columns]
rest = [c for c in df_enriched.columns if c not in order_exist]
df_enriched = df_enriched[order_exist + rest]

# -------------------------
# 8) QA rápido
# -------------------------
print("Rows:", len(df_enriched))
print("Unique DOIs:", df_enriched["doi"].nunique(dropna=True))
print("Altmetric coverage:", df_enriched["altmetric_score"].notna().mean())
print("Dimensions coverage:",
df_enriched["dimensions_times_cited"].notna().mean())
print("Mendeley coverage:",
df_enriched["mendeley_reader_count"].notna().mean())
print("OpenAlex coverage:",
df_enriched["openalex_cited_by_count"].notna().mean())

# -------------------------
# 9) Save
# -------------------------
df_enriched.to_csv(OUT_CSV, index=False)
print("✅ Saved:", OUT_CSV)

df_enriched.to_parquet(OUT_PARQUET, index=False)
print("✅ Saved:", OUT_PARQUET)

df_enriched.head(3)
```

```
Base master: (10831, 33) DOIs: 10830
Enriched: (10831, 76)
Rows: 10831
Unique DOIs: 10830
```

```
Altmetric coverage: 0.25833256393684795
Dimensions coverage: 0.9494044871203028
Mendeley coverage: 0.9947373280398855
OpenAlex coverage: 0.9756255193426276
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/df_master_enriched.csv
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/df_master_enriched.parquet
```

	doi	title	year	journal	doc
0	10.1002/(sici)1096-9128(199601)8:1<47::aid-cpe...	Benchmarking the computation and communication...	1996	Concurrency Practice and Experience	Arti
1	10.1002/(sici)1097-0193(1999)8:2/3<128::aid-hb...	Computational modeling of high-level cognition...	1999	Human Brain Mapping	Cor pap
2	10.1002/(sici)1097-0363(199706)24:12<1321::aid...	Parallel computation of incompressible flows w...	1997	International Journal for Numerical Methods in...	Arti

3 rows × 76 columns

```
Warning: Total number of columns (76) exceeds max_columns (20) limiting to first (20) columns.
```

In [ ]:

```
import os, re
import pandas as pd
import numpy as np

BASE_PATH = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"
IN_PATH   = os.path.join(BASE_PATH, "df_master_enriched.csv")

df = pd.read_csv(IN_PATH, low_memory=False)

def norm_text(x):
    if pd.isna(x):
        return ""
    return str(x).lower()

def has_any(text, terms):
    t = norm_text(text)
    return any(term in t for term in terms)

print("Loaded:", df.shape)
```

```
print("Year range:", df["year"].min(), "–", df["year"].max())
print("DOIs:", df["doi"].nunique(dropna=True))
```

```
Loaded: (10831, 76)
Year range: 1970 – 2026
DOIs: 10830
```

In [ ]:

```
import os, json
import pandas as pd
import numpy as np
from collections import Counter

BASE_PATH = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/"
MASTER_IN = os.path.join(BASE_PATH, "df_master_enriched.csv")

# ========================
# 0) Cargar master
# ========================
df = pd.read_csv(MASTER_IN, low_memory=False)
print("Loaded master:", df.shape, "| DOIs:",
df["doi"].nunique(dropna=True))

def norm_text(x):
    if pd.isna(x):
        return ""
    return str(x).lower()

# ========================
# 1) Flags CT como CAPA (no filtro)
# ========================
NEGATIVE_PATTERNS = [
    "computational modeling", "computational model", "computational
simulation",
    "benchmarking the computation", "parallel computation", "incompressible
flows",
    "prefrontal cortex", "neuroscience", "wireless", "6g", "ghz"
]

def build_ct_flags(row):
    title = norm_text(row.get("title"))
    abs1  = norm_text(row.get("abstract_scopus"))
    abs2  = norm_text(row.get("abstract_wos"))
    kw_s  = norm_text(row.get("author_keywords_scopus")) + " " +
norm_text(row.get("index_keywords_scopus"))
    kw_w  = norm_text(row.get("author_keywords_wos")) + " " +
norm_text(row.get("keywords_plus_wos"))
    concepts = norm_text(row.get("openalex_top_concepts"))

    text_all = " ".join([title, abs1, abs2, kw_s, kw_w])

    ct_exact = ("computational thinking" in text_all)

    edu_hit = any(w in text_all for w in [

"education","educational","teacher","teachers","student","students",
```

```
        "school","curriculum","classroom","k-12","primary","secondary"
    ])
    skill_hit = any(w in text_all for w in [
        "programming","coding","robotics","scratch","stem","steam",
        "abstraction","decomposition","algorithm","algorithms",
        "debugging","troubleshooting","problem solving"
    ])
    ct_context = bool(edu_hit and skill_hit)

    ct_concepts = (("education" in concepts) and ("computer science" in
concepts)) or ("computational thinking" in concepts)

    ct_negative = any(p in text_all for p in NEGATIVE_PATTERNS)

    # Score 0-3 (sin filtrar)
    ct_score = int(ct_exact) + int(ct_context) + int(ct_concepts)

    # Label (para tooltips/filtros)
    if ct_negative:
        ct_label = "noise"
    elif ct_score >= 2:
        ct_label = "core"
    elif ct_score == 1:
        ct_label = "broad"
    else:
        ct_label = "none"

    return pd.Series({
        "ct_exact": ct_exact,
        "ct_context": ct_context,
        "ct_concepts": ct_concepts,
        "ct_negative": ct_negative,
        "ct_score": ct_score,
        "ct_label": ct_label
    })

flags = df.apply(build_ct_flags, axis=1)
df = pd.concat([df, flags], axis=1)

print("CT label distribution:\n",
df["ct_label"].value_counts(dropna=False))
print("CT score distribution:\n",
df["ct_score"].value_counts(dropna=False).sort_index())


# ========================
# 2) viz_time_series_all.csv (NO FILTRO)
# ========================
OUT_TIME_ALL = os.path.join(BASE_PATH, "viz_time_series_all.csv")

d = df.copy()
d["year"] = pd.to_numeric(d["year"], errors="coerce").astype("Int64")

# Métrica canónica de impacto académico (robusta con fallback)
d["citations_academic"] = (
    pd.to_numeric(d.get("dimensions_times_cited"), errors="coerce")
      .fillna(pd.to_numeric(d.get("openalex_cited_by_count"),
```

```
errors="coerce"))
      .fillna(pd.to_numeric(d.get("scopus_citations"), errors="coerce"))
      .fillna(pd.to_numeric(d.get("wos_citations_core"), errors="coerce"))
)

d["altmetric_score_filled"] = pd.to_numeric(d.get("altmetric_score"),
errors="coerce").fillna(0)
d["mendeley_readers_filled"] =
pd.to_numeric(d.get("mendeley_reader_count"), errors="coerce").fillna(0)
d["is_oa"] = d.get("openalex_is_oa").fillna(False).astype(bool)

ts_all = (d.dropna(subset=["year"])
          .groupby(["ct_label","year"], dropna=True)
          .agg(
              n_papers=("doi", "count"),
              n_unique_doi=("doi", "nunique"),
              mean_citations=("citations_academic", "mean"),
              median_citations=("citations_academic", "median"),
              total_citations=("citations_academic", "sum"),
              mean_altmetric=("altmetric_score_filled", "mean"),
              total_altmetric=("altmetric_score_filled", "sum"),
              mean_mendeley_readers=("mendeley_readers_filled", "mean"),
              total_mendeley_readers=("mendeley_readers_filled", "sum"),
              share_oa=("is_oa", "mean"),
              mean_ct_score=("ct_score", "mean")
          )
          .reset_index()
          .sort_values(["ct_label","year"]))

ts_all.to_csv(OUT_TIME_ALL, index=False)
print("✅ Saved:", OUT_TIME_ALL, "| rows:", len(ts_all))


# ========================
# 3) viz_impact_scatter_all.csv (NO FILTRO)
# ========================
OUT_SCAT_ALL = os.path.join(BASE_PATH, "viz_impact_scatter_all.csv")

scatter_cols = [
    "doi","title","year","journal","document_type","publisher","language",
    "openalex_is_oa","openalex_oa_status",

"dimensions_times_cited","openalex_cited_by_count","scopus_citations","wos_
citations_core",
    "altmetric_score","altmetric_posts_count","altmetric_accounts_count",
    "altmetric_tweeters_count","altmetric_bluesky_count",
    "mendeley_reader_count","mendeley_group_count",
    "openalex_top_concepts",
    "ct_score","ct_label"
]

scat_all = d[[c for c in scatter_cols if c in d.columns]].copy()

scat_all["citations_academic"] = (
    pd.to_numeric(scat_all.get("dimensions_times_cited"), errors="coerce")
      .fillna(pd.to_numeric(scat_all.get("openalex_cited_by_count"),
errors="coerce"))
```

```
      .fillna(pd.to_numeric(scat_all.get("scopus_citations"),
errors="coerce"))
      .fillna(pd.to_numeric(scat_all.get("wos_citations_core"),
errors="coerce"))
)

scat_all["altmetric_score"] =
pd.to_numeric(scat_all.get("altmetric_score"), errors="coerce")
scat_all["mendeley_reader_count"] =
pd.to_numeric(scat_all.get("mendeley_reader_count"), errors="coerce")

# Columnas rellenas para scatter
scat_all["altmetric_score_filled"] = scat_all["altmetric_score"].fillna(0)
scat_all["mendeley_readers_filled"] =
scat_all["mendeley_reader_count"].fillna(0)
scat_all["is_oa"] =
scat_all.get("openalex_is_oa").fillna(False).astype(bool)

scat_all.to_csv(OUT_SCAT_ALL, index=False)
print("✅ Saved:", OUT_SCAT_ALL, "| rows:", len(scat_all))


# ========================
# 4) viz_geo_all.csv (OpenAlex authorships) (NO FILTRO)
# ========================
# Este paso requiere OpenAlex.jsonl (crudo), porque el summary no trae
países explícitos por paper.
OA_JSONL = os.path.join(BASE_PATH, "OpenAlex.jsonl")
OUT_GEO_ALL = os.path.join(BASE_PATH, "viz_geo_all.csv")

master_dois = set(df["doi"].dropna())
doi_to_meta = df.set_index("doi")
[["year","ct_label","ct_score"]].to_dict(orient="index")

rows = []
parse_errors = 0
kept = 0

with open(OA_JSONL, "r") as f:
    for line in f:
        try:
            r = json.loads(line)
        except Exception:
            parse_errors += 1
            continue

        doi = r.get("doi")
        if doi is None or doi not in master_dois:
            continue

        kept += 1
        meta = doi_to_meta.get(doi, {})
        year = meta.get("year", r.get("year"))
        ct_label = meta.get("ct_label", "none")
        ct_score = meta.get("ct_score", 0)

        authorships = r.get("openalex", {}).get("authorships", [])
```

```
        countries = []

        for a in authorships:
            for inst in a.get("institutions", []):
                c = inst.get("country_code")
                if c:
                    countries.append(c)

        if not countries:
            continue

        cnt = Counter(countries)
        for country, n in cnt.items():
            rows.append({
                "doi": doi,
                "year": year,
                "ct_label": ct_label,
                "ct_score": ct_score,
                "country": country,
                "n_authorships": n
            })

geo_all = pd.DataFrame(rows)
print("OpenAlex parse errors:", parse_errors)
print("OpenAlex records matched master DOIs:", kept)
print("Geo rows (doi-country):", len(geo_all))

# Agregado por país para mapa
geo_agg = (geo_all
           .groupby(["ct_label","country"], dropna=False)
           .agg(
               n_papers=("doi","nunique"),
               total_authorships=("n_authorships","sum"),
               mean_ct_score=("ct_score","mean")
           )
           .reset_index()
           .sort_values("n_papers", ascending=False))

geo_agg.to_csv(OUT_GEO_ALL, index=False)
print("✅ Saved:", OUT_GEO_ALL, "| rows:", len(geo_agg))

geo_agg.head(10)
```

```
Loaded master: (10831, 76) | DOIs: 10830
CT label distribution:
 ct_label
core     6144
none     2464
broad    1422
noise     801
Name: count, dtype: int64
CT score distribution:
 ct_score
0    3022
1    1517
2    3114
3    3178
```

```
Name: count, dtype: int64
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/viz_time_series_all.csv | rows: 149
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/viz_impact_scatter_all.csv | rows: 10831
OpenAlex parse errors: 0
OpenAlex records matched master DOIs: 10830
Geo rows (doi-country): 10973
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/viz_geo_all.csv | rows: 352
```

	ct_label	country	n_papers	total_authorships	mean_ct_score
186	core	US	1528	5240	2.490838
345	none	US	683	2015	0.000000
104	core	CN	476	1501	2.600840
84	broad	US	398	1244	1.000000
117	core	ES	363	1062	2.526171
252	noise	US	315	1060	0.707937
272	none	CN	273	1159	0.000000
286	none	GB	228	505	0.000000
184	core	TW	226	624	2.646018
183	core	TR	203	445	2.596059

In [ ]:
```
geo_agg.head(10)
```

	ct_label	country	n_papers	total_authorships	mean_ct_score
186	core	US	1528	5240	2.490838
345	none	US	683	2015	0.000000
104	core	CN	476	1501	2.600840
84	broad	US	398	1244	1.000000
117	core	ES	363	1062	2.526171
252	noise	US	315	1060	0.707937
272	none	CN	273	1159	0.000000
286	none	GB	228	505	0.000000
184	core	TW	226	624	2.646018
183	core	TR	203	445	2.596059

In [ ]:

```
import os, re
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter

BASE = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"
IN_FILE  = os.path.join(BASE, "df_master_enriched.csv")
OUT_FILE = os.path.join(BASE, "nlp_skill_candidates.csv")

# ========================
# 1) Load data
# ========================
df = pd.read_csv(IN_FILE, low_memory=False)
print("Loaded:", df.shape)

def norm_text(x):
    if pd.isna(x):
        return ""
    x = str(x).lower()
    x = re.sub(r"http\S+", " ", x)
    x = re.sub(r"[^a-z0-9\s\-]", " ", x)
    x = re.sub(r"\s+", " ", x).strip()
    return x

# ========================
# 2) Build CT flags (ct_label, ct_score)
# ========================
NEGATIVE_PATTERNS = [
    "computational modeling", "computational model", "computational
simulation",
    "benchmarking the computation", "parallel computation", "incompressible
flows",
    "prefrontal cortex", "neuroscience", "wireless", "6g", "ghz"
]

def build_ct_flags(row):
    title = norm_text(row.get("title"))
    abs1  = norm_text(row.get("abstract_scopus"))
    abs2  = norm_text(row.get("abstract_wos"))
    kw_s  = norm_text(row.get("author_keywords_scopus")) + " " +
norm_text(row.get("index_keywords_scopus"))
    kw_w  = norm_text(row.get("author_keywords_wos")) + " " +
norm_text(row.get("keywords_plus_wos"))
    concepts = norm_text(row.get("openalex_top_concepts"))

    text_all = " ".join([title, abs1, abs2, kw_s, kw_w])

    ct_exact = ("computational thinking" in text_all)

    edu_hit = any(w in text_all for w in [

"education","educational","teacher","teachers","student","students",
        "school","curriculum","classroom","k-12","primary","secondary"
    ])
    skill_hit = any(w in text_all for w in [
        "programming","coding","robotics","scratch","stem","steam",
```

```
        "abstraction","decomposition","algorithm","algorithms",
        "debugging","troubleshooting","problem solving"
    ])
    ct_context = bool(edu_hit and skill_hit)

    ct_concepts = (("education" in concepts) and ("computer science" in
concepts)) or ("computational thinking" in concepts)
    ct_negative = any(p in text_all for p in NEGATIVE_PATTERNS)

    ct_score = int(ct_exact) + int(ct_context) + int(ct_concepts)

    if ct_negative:
        ct_label = "noise"
    elif ct_score >= 2:
        ct_label = "core"
    elif ct_score == 1:
        ct_label = "broad"
    else:
        ct_label = "none"

    return pd.Series({
        "ct_exact": ct_exact,
        "ct_context": ct_context,
        "ct_concepts": ct_concepts,
        "ct_negative": ct_negative,
        "ct_score": ct_score,
        "ct_label": ct_label
    })

flags = df.apply(build_ct_flags, axis=1)
df = pd.concat([df, flags], axis=1)

print("CT distribution:\n", df["ct_label"].value_counts(dropna=False))

# ========================
# 3) Build text_corpus
# ========================
df["text_corpus"] = (
    df["title"].fillna("").apply(norm_text) + " " +
    df["abstract_scopus"].fillna("").apply(norm_text) + " " +
    df["abstract_wos"].fillna("").apply(norm_text) + " " +
    df["author_keywords_scopus"].fillna("").apply(norm_text) + " " +
    df["index_keywords_scopus"].fillna("").apply(norm_text) + " " +
    df["author_keywords_wos"].fillna("").apply(norm_text) + " " +
    df["keywords_plus_wos"].fillna("").apply(norm_text)
)

df["year"] = pd.to_numeric(df["year"], errors="coerce").astype("Int64")

# ========================
# 4) Split core vs none
# ========================
df_core = df[df["ct_label"] == "core"]
df_none = df[df["ct_label"] == "none"]
print("Core:", len(df_core), "| None:", len(df_none))

# ========================
```

```
# 5) Discriminative terms (TF-IDF core - none)
# ========================
vectorizer = TfidfVectorizer(
    stop_words="english",
    ngram_range=(1,3),
    min_df=10,
    max_df=0.85
)

X_core = vectorizer.fit_transform(df_core["text_corpus"])
terms = np.array(vectorizer.get_feature_names_out())
tfidf_core = np.asarray(X_core.mean(axis=0)).ravel()

X_none = vectorizer.transform(df_none["text_corpus"])
tfidf_none = np.asarray(X_none.mean(axis=0)).ravel()

disc_score = tfidf_core - tfidf_none

disc_df = pd.DataFrame({
    "term": terms,
    "score_discriminative": disc_score,
    "tfidf_core": tfidf_core,
    "tfidf_none": tfidf_none
}).sort_values("score_discriminative", ascending=False)

# ========================
# 6) Emerging terms (growth) using n-grams from recent years
# ========================
recent_year = int(df["year"].dropna().max())
past_year = recent_year - 6

df_recent = df[(df["year"] >= past_year) &
(df["ct_label"].isin(["core","broad"]))].copy()
print("Recent window:", past_year, "-", recent_year, "| rows:",
len(df_recent))

# Simple token counts per year (unigrams only for growth; OK for signal)
rows = []
for _, r in df_recent.iterrows():
    toks = r["text_corpus"].split()
    for t in toks:
        rows.append((t, int(r["year"]) if pd.notna(r["year"]) else None))

tmp = pd.DataFrame(rows, columns=["term","year"]).dropna(subset=["year"])
growth = tmp.groupby(["term","year"]).size().reset_index(name="count")

g_first = growth.groupby("term")["year"].min()
g_last  = growth.groupby("term")["year"].max()
g_total = growth.groupby("term")["count"].sum()
g_span  = (g_last - g_first + 1)

growth_df = pd.DataFrame({
    "term": g_total.index,
    "freq_recent_total": g_total.values,
    "first_year": g_first.values,
    "last_year": g_last.values,
    "active_years": g_span.values
```

```
})

# Growth score (simple + estable)
growth_df["score_growth"] = growth_df["active_years"] *
np.log1p(growth_df["freq_recent_total"])

# ========================
# 7) Merge candidates
# ========================
cand = disc_df.merge(growth_df, on="term", how="left")
cand["score_growth"] = cand["score_growth"].fillna(0)
cand["freq_recent_total"] = cand["freq_recent_total"].fillna(0)

# Keep: positivos y con señal en core
cand = cand[(cand["score_discriminative"] > 0) & (cand["tfidf_core"] > 0)]

# Rank: discriminative primero, luego growth
cand = cand.sort_values(["score_discriminative", "score_growth"],
ascending=False)

# Take top N (mejor que 50)
cand = cand.head(250).reset_index(drop=True)

cand.to_csv(OUT_FILE, index=False)
print("✅ Saved:", OUT_FILE, "| candidates:", len(cand))
cand.head(15)
```

```
Loaded: (10831, 76)
CT distribution:
 ct_label
core     6145
none     2462
broad    1422
noise     802
Name: count, dtype: int64
Core: 6145 | None: 2462
Recent window: 2020 - 2026 | rows: 5345
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/nlp_skill_candidates.csv | candidates: 250
```

	term	score_discriminative	tfidf_core	tfidf_none	freq_recent_
0	programming	0.039362	0.041661	0.002299	13252.0
1	ct	0.039131	0.039238	0.000107	13896.0
2	students	0.038843	0.041607	0.002764	19984.0
3	education	0.031801	0.033392	0.001591	15490.0
4	learning	0.029197	0.040678	0.011481	19000.0
5	skills	0.022892	0.024851	0.001959	10074.0
6	teaching	0.022155	0.023087	0.000932	6761.0
7	teachers	0.021675	0.022167	0.000492	6932.0

	term	score_discriminative	tfidf_core	tfidf_none	freq_recent_
8	school	0.018829	0.019436	0.000607	5604.0
9	educational	0.018561	0.019440	0.000879	5821.0
10	science	0.015884	0.024479	0.008596	7626.0
11	stem	0.015472	0.015904	0.000431	3225.0
12	robotics	0.015438	0.016602	0.001164	3171.0
13	computer science	0.015354	0.016647	0.001293	0.0
14	coding	0.015339	0.016030	0.000691	3292.0

In [ ]:

```
#!pip -q install openai pandas

import os, json
import pandas as pd
from openai import OpenAI

BASE = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"
CAND_PATH = os.path.join(BASE, "nlp_skill_candidates.csv")
LLM_OUT   = os.path.join(BASE, "llm_skills_output.json")

# Setea tu key en Colab (NO la imprimas)
# os.environ["OPENAI_API_KEY"] = "YOUR_KEY"

client = OpenAI()

cand = pd.read_csv(CAND_PATH)
terms = cand["term"].dropna().astype(str).tolist()
input_terms = "\n".join([f"- {t}" for t in terms])

PROMPT = f"""
You are a domain curator. Your task is to organize candidate terms into
Computational Thinking (CT) SKILLS.

IMPORTANT CONSTRAINTS (anti-hallucination):
1) You MUST ONLY use the provided input terms. Do NOT invent new terms or
new skills not supported by input terms.
2) You MUST return strictly valid JSON (no markdown, no commentary).
3) Each skill must be supported by 2–15 evidence terms from the input list.
4) Skills must represent CT capabilities/skills (e.g., abstraction,
decomposition, debugging, algorithmic thinking).
   Terms that describe population/setting (students, education, school,
teachers) MUST NOT be skills.
5) Terms that are research methods/analysis (survey, experiment, analysis,
review, model, dataset) MUST NOT be skills.
6) If a term is ambiguous, put it in "ambiguous_terms" with a reason.

OUTPUT JSON SCHEMA:
{{
  "skills": [
```

```
    {{
      "skill_name": "string (CT skill label, short)",
      "evidence_terms": ["term1","term2", "..."],
      "confidence": 0.0-1.0,
      "notes": "short justification"
    }}
  ],
  "context_terms": [
    {{"term":"...", "reason":"education context / population / setting"}}
  ],
  "method_terms": [
    {{"term":"...", "reason":"research method / analysis technique"}}
  ],
  "discarded_terms": [
    {{"term":"...", "reason":"too generic / unrelated to CT skills"}}
  ],
  "ambiguous_terms": [
    {{"term":"...", "reason":"why ambiguous"}}
  ]
}}

Now process the following INPUT_TERMS:
<INPUT_TERMS>
{input_terms}
</INPUT_TERMS>
"""

resp = client.chat.completions.create(
    model="gpt-5.2",   # ajusta al modelo habilitado en tu cuenta
    messages=[
        {"role":"system","content":"Return only valid JSON."},
        {"role":"user","content":PROMPT}
    ],
    temperature=0.1
)

raw = resp.choices[0].message.content.strip()

with open(LLM_OUT, "w", encoding="utf-8") as f:
    f.write(raw)

print("✅ Saved:", LLM_OUT)
print(raw[:500])
```

```
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/llm_skills_output.json
{
  "skills": [
    {
      "skill_name": "Abstraction",
      "evidence_terms": [
        "abstraction",
        "concepts",
        "programming concepts",
        "computational thinking skills"
      ],
      "confidence": 0.86,
```

```
      "notes": "Directly supported by the explicit term 'abstraction'
    and reinforced by concept-focused terms tied to computational
    thinking."
    },
    {
      "skill_name": "Decomposition",
      "evidence_terms": [
        "decomposition",
        "problem",
```

In [ ]:
```
cand = pd.read_csv(CAND_PATH)
terms = cand["term"].dropna().astype(str).tolist()

# Para evitar prompts gigantes: envía como lista numerada (súper estable)
input_terms = "\n".join([f"- {t}" for t in terms])
print("terms:", len(terms))
```

```
terms: 250
```

In [ ]:
```
import os, json
import pandas as pd

BASE = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"
LLM_OUT   = os.path.join(BASE, "llm_skills_output.json")
DICT_OUT  = os.path.join(BASE, "skill_dictionary_v1.csv")

with open(LLM_OUT, "r", encoding="utf-8") as f:
    data = json.load(f)

rows = []

for s in data.get("skills", []):
    skill = s.get("skill_name","").strip()
    conf  = s.get("confidence", None)
    notes = s.get("notes","")
    for t in s.get("evidence_terms", []):
        rows.append({
            "skill_name": skill,
            "term": str(t).strip(),
            "confidence": conf,
            "notes": notes,
            "match_type": "evidence_term"
        })

for bucket, mtype in [
    ("context_terms","context"),
    ("method_terms","method"),
    ("discarded_terms","discarded"),
    ("ambiguous_terms","ambiguous")
]:
    for obj in data.get(bucket, []):
        rows.append({
            "skill_name": "",
            "term": str(obj.get("term","")).strip(),
            "confidence": "",
```

```
            "notes": str(obj.get("reason","")).strip(),
            "match_type": mtype
        })

df_dict = pd.DataFrame(rows).dropna(subset=["term"])
df_dict = df_dict[df_dict["term"].astype(str).str.len() > 0]

df_dict.to_csv(DICT_OUT, index=False)
print("✅ Saved:", DICT_OUT)
df_dict.head(20)
```

```
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/skill_dictionary_v1.csv
```

	skill_name	term	confidence	notes	match_typ
0	Abstraction	abstraction	0.86	Directly supported by the explicit term 'abstr...	evidence_ter
1	Abstraction	concepts	0.86	Directly supported by the explicit term 'abstr...	evidence_ter
2	Abstraction	programming concepts	0.86	Directly supported by the explicit term 'abstr...	evidence_ter
3	Abstraction	computational thinking skills	0.86	Directly supported by the explicit term 'abstr...	evidence_ter
4	Decomposition	decomposition	0.84	Decomposition is explicitly present and aligns...	evidence_ter
5	Decomposition	problem	0.84	Decomposition is explicitly present and aligns...	evidence_ter
6	Decomposition	problem solving	0.84	Decomposition is explicitly present and aligns...	evidence_ter
7	Decomposition	problem solving skills	0.84	Decomposition is explicitly present and aligns...	evidence_ter

	skill_name	term	confidence	notes	match_typ
8	Decomposition	solving	0.84	Decomposition is explicitly present and aligns...	evidence_ter
9	Decomposition	solving skills	0.84	Decomposition is explicitly present and aligns...	evidence_ter
10	Algorithmic thinking	algorithmic thinking	0.83	Algorithmic thinking is explicitly listed and ...	evidence_ter
11	Algorithmic thinking	algorithmic	0.83	Algorithmic thinking is explicitly listed and ...	evidence_ter
12	Algorithmic thinking	programming	0.83	Algorithmic thinking is explicitly listed and ...	evidence_ter
13	Algorithmic thinking	coding	0.83	Algorithmic thinking is explicitly listed and ...	evidence_ter
14	Algorithmic thinking	code	0.83	Algorithmic thinking is explicitly listed and ...	evidence_ter
15	Debugging	debugging	0.82	Debugging is explicitly present and is a core ...	evidence_ter
16	Debugging	programming	0.82	Debugging is explicitly present and is a core ...	evidence_ter
17	Debugging	computer programming	0.82	Debugging is explicitly present and is a core ...	evidence_ter
18	Debugging	programming skills	0.82	Debugging is explicitly present and is a core ...	evidence_ter

	skill_name	term	confidence	notes	match_typ
19	Programming (coding)	programming	0.78	Multiple direct terms indicate the CT capabili...	evidence_ter

In [ ]:

```
import os, re
import pandas as pd
import numpy as np

BASE = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"
MASTER_IN  = os.path.join(BASE, "df_master_enriched.csv")
DICT_IN    = os.path.join(BASE, "skill_dictionary_v1.csv")
MASTER_OUT = os.path.join(BASE, "df_master_enriched_ctv2.csv")

df = pd.read_csv(MASTER_IN, low_memory=False)
df_dict = pd.read_csv(DICT_IN)

def norm_text(x):
    if pd.isna(x):
        return ""
    x = str(x).lower()
    x = re.sub(r"http\S+", " ", x)
    x = re.sub(r"[^a-z0-9\s\-]", " ", x)
    x = re.sub(r"\s+", " ", x).strip()
    return x

# Texto canónico
df["text_corpus"] = (
    df["title"].fillna("").apply(norm_text) + " " +
    df["abstract_scopus"].fillna("").apply(norm_text) + " " +
    df["abstract_wos"].fillna("").apply(norm_text) + " " +
    df["author_keywords_scopus"].fillna("").apply(norm_text) + " " +
    df["index_keywords_scopus"].fillna("").apply(norm_text) + " " +
    df["author_keywords_wos"].fillna("").apply(norm_text) + " " +
    df["keywords_plus_wos"].fillna("").apply(norm_text)
)

# Conceptos (OpenAlex + Dimensions)
df["concepts_corpus"] = (
    df["openalex_top_concepts"].fillna("").apply(norm_text) + " " +
    df["dimensions_top_concepts"].fillna("").apply(norm_text)
).str.strip()

# --- Skills dictionary
skills_terms = df_dict[df_dict["match_type"]=="evidence_term"].copy()
skills_terms["term_norm"] =
skills_terms["term"].astype(str).str.lower().str.strip()
term2skill = skills_terms.groupby("term_norm")["skill_name"].apply(lambda
x: sorted(set(x))).to_dict()
all_terms = sorted(term2skill.keys(), key=len, reverse=True)

def extract_skills(text):
    hits = set()
```

```
    for t in all_terms:
        if t and t in text:
            for sk in term2skill[t]:
                hits.add(sk)
    return hits

skill_hits = []
n_hits = []
for txt in df["text_corpus"].tolist():
    h = extract_skills(txt)
    skill_hits.append("; ".join(sorted(h)))
    n_hits.append(len(h))

df["n_skills_hit"] = n_hits
df["skills_hit_list"] = skill_hits

# --- Señales base CT (score 0..3 + negativos)
NEGATIVE_PATTERNS = [
    "computational modeling", "computational model", "computational
simulation",
    "benchmarking the computation", "parallel computation", "incompressible
flows",
    "prefrontal cortex", "neuroscience", "wireless", "6g", "ghz"
]

def build_ct_score(row):
    text_all = row["text_corpus"]
    concepts = row["concepts_corpus"]

    ct_exact = ("computational thinking" in text_all)

    edu_hit = any(w in text_all for w in [

"education","educational","teacher","teachers","student","students",
        "school","curriculum","classroom","k-12","primary","secondary"
    ])
    skill_hit = any(w in text_all for w in [
        "programming","coding","robotics","scratch","stem","steam",
        "abstraction","decomposition","algorithm","algorithms",
        "debugging","troubleshooting","problem solving"
    ])
    ct_context = bool(edu_hit and skill_hit)

    ct_concepts = (("education" in concepts) and ("computer science" in
concepts)) or ("computational thinking" in concepts)
    ct_negative = any(p in text_all for p in NEGATIVE_PATTERNS)

    base = int(ct_exact) + int(ct_context) + int(ct_concepts)
    return pd.Series({"ct_score": base, "ct_negative": ct_negative})

tmp = df.apply(build_ct_score, axis=1)
df["ct_score"] = tmp["ct_score"].astype(int)
df["ct_negative"] = tmp["ct_negative"].astype(bool)

# --- Score v2 (0..1)
df["skills_score"] = np.minimum(df["n_skills_hit"], 6) / 6.0
df["ct_score_norm"] = df["ct_score"] / 3.0
```

```
df["ct_membership_score_v2"] = (
    0.55 * df["ct_score_norm"] +
    0.45 * df["skills_score"]
)

# penalizar negativos
df.loc[df["ct_negative"], "ct_membership_score_v2"] *= 0.6

def label_v2(score, neg):
    if neg:
        return "noise"
    if score >= 0.72:
        return "core"
    if score >= 0.45:
        return "broad"
    return "none"

df["ct_label_v2"] = [label_v2(s, n) for s, n in
zip(df["ct_membership_score_v2"], df["ct_negative"])]

print("CT v2 distribution:\n", df["ct_label_v2"].value_counts())

df.to_csv(MASTER_OUT, index=False)
print("✅ Saved:", MASTER_OUT)

df[["doi","year","ct_score","n_skills_hit","ct_membership_score_v2","ct_lab
el_v2"]].head(10)
```

```
CT v2 distribution:
 ct_label_v2
none     4637
core     2955
broad    2437
noise     802
Name: count, dtype: int64
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/df_master_enriched_ctv2.csv
```

	doi	year	ct_score	n_skills_hit	ct_membe
0	10.1002/(sici)1096-9128(199601)8:1<47::aid-cpe...	1996	0	2	0.090000
1	10.1002/(sici)1097-0193(1999)8:2/3<128::aid-hb...	1999	0	1	0.045000
2	10.1002/(sici)1097-0363(199706)24:12<1321::aid...	1997	0	0	0.000000
3	10.1002/(sici)1097-0363(199706)24:12<1353::aid...	1997	0	0	0.000000
4	10.1002/(sici)1097-0363(199706)24:12<1371::aid...	1997	0	0	0.000000

	doi	year	ct_score	n_skills_hit	ct_membe
5	10.1002/(sici)1097-0363(199706)24:12<1417::aid...	1997	0	0	0.000000
6	10.1002/(sici)1097-0363(199706)24:12<1449::aid...	1997	1	4	0.483333
7	10.1002/(sici)1098-111x(199702)12:2<105::aid-i...	1997	0	4	0.300000
8	10.1002/(sici)1098-111x(199911)14:11<1071::aid...	1999	0	1	0.075000
9	10.1002/(sici)1099-1743(200003/04)17:2<149::ai...	2000	0	0	0.000000

In [ ]:

```
PROMPT_V2 = f"""
You are a strict CT-skills curator. Your job is to build a CT skill
dictionary ONLY from the provided terms.

ABSOLUTE RULES (must follow):
A) You MUST ONLY use terms from INPUT_TERMS. Do NOT invent new terms.
B) You MUST output STRICT VALID JSON (no markdown, no extra text).
C) Evidence terms must be SKILL-LIKE and specific. Reject generic context
words.

WHAT COUNTS AS A CT SKILL TERM?
- A term that denotes a CT capability/practice (e.g., abstraction,
decomposition, debugging, algorithm design, pattern recognition, data
representation).
- Prefer specific n-grams (>=2 words) when possible (e.g., "problem
solving" is better than "problem").

FORBIDDEN AS EVIDENCE (send to context_terms or discarded_terms):
- Generic research words: study, analysis, review, method, experiment,
model, approach, framework.
- Generic education setting/population: education, learning, students,
teachers, school, classroom, curriculum.
- Generic computing terms: programming, coding, code, computer, science,
technology, digital.
- Single-word vague terms: problem, concepts, skills, data (unless it is
part of a specific skill n-gram like "data representation").

EVIDENCE QUALITY REQUIREMENT:
- Each skill MUST have at least 2 and at most 8 evidence terms.
- At least 2 evidence terms must be DIRECT skill terms (not context).
- If you cannot find enough direct evidence terms, DO NOT create the skill;
put candidate evidence in ambiguous_terms.

OUTPUT JSON SCHEMA:
{{
  "skills":[
    {{
      "skill_name":"short label",
      "direct_evidence_terms":["term1","term2"],
```

```
      "supporting_evidence_terms":["term3","term4"],
      "confidence":0.0-1.0,
      "notes":"1 sentence"
    }}
  ],
  "context_terms":[{"term":"...", "reason":"setting/population/general
context"}}],
  "method_terms":[{"term":"...", "reason":"research method/analysis"}}],
  "discarded_terms":[{"term":"...", "reason":"too generic/unrelated"}}],
  "ambiguous_terms":[{"term":"...", "reason":"ambiguous skill vs
context"}}]
}}

INPUT_TERMS:
<INPUT_TERMS>
{input_terms}
</INPUT_TERMS>
"""
```

In [ ]:

```
import os, json
from openai import OpenAI
import pandas as pd

from google.colab import userdata
api_key = userdata.get("OPENAI_API_KEY")
os.environ["OPENAI_API_KEY"] = api_key


BASE = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"
CAND_PATH = os.path.join(BASE, "nlp_skill_candidates.csv")
LLM_OUT_V2 = os.path.join(BASE, "llm_skills_output_v2.json")

client = OpenAI()

cand = pd.read_csv(CAND_PATH)
terms = cand["term"].dropna().astype(str).tolist()
input_terms = "\n".join([f"- {t}" for t in terms])

# --- pega aquí PROMPT_V2 (del bloque anterior) ---
PROMPT_V2 = f"""
You are a strict CT-skills curator. Your job is to build a CT skill
dictionary ONLY from the provided terms.

ABSOLUTE RULES (must follow):
A) You MUST ONLY use terms from INPUT_TERMS. Do NOT invent new terms.
B) You MUST output STRICT VALID JSON (no markdown, no extra text).
C) Evidence terms must be SKILL-LIKE and specific. Reject generic context
words.

WHAT COUNTS AS A CT SKILL TERM?
- A term that denotes a CT capability/practice (e.g., abstraction,
decomposition, debugging, algorithm design, pattern recognition, data
representation).
- Prefer specific n-grams (>=2 words) when possible (e.g., "problem
solving" is better than "problem").

FORBIDDEN AS EVIDENCE (send to context_terms or discarded_terms):
```

```
- Generic research words: study, analysis, review, method, experiment,
model, approach, framework.
- Generic education setting/population: education, learning, students,
teachers, school, classroom, curriculum.
- Generic computing terms: programming, coding, code, computer, science,
technology, digital.
- Single-word vague terms: problem, concepts, skills, data (unless it is
part of a specific skill n-gram like "data representation").

EVIDENCE QUALITY REQUIREMENT:
- Each skill MUST have at least 2 and at most 8 evidence terms.
- At least 2 evidence terms must be DIRECT skill terms (not context).
- If you cannot find enough direct evidence terms, DO NOT create the skill;
put candidate evidence in ambiguous_terms.

OUTPUT JSON SCHEMA:
{{
  "skills":[
    {{
      "skill_name":"short label",
      "direct_evidence_terms":["term1","term2"],
      "supporting_evidence_terms":["term3","term4"],
      "confidence":0.0-1.0,
      "notes":"1 sentence"
    }}
  ],
  "context_terms":[{{"term":"...", "reason":"setting/population/general
context"}}],
  "method_terms":[{{"term":"...", "reason":"research method/analysis"}}],
  "discarded_terms":[{{"term":"...", "reason":"too generic/unrelated"}}],
  "ambiguous_terms":[{{"term":"...", "reason":"ambiguous skill vs
context"}}]
}}

INPUT_TERMS:
<INPUT_TERMS>
{input_terms}
</INPUT_TERMS>
"""


resp = client.chat.completions.create(
    model="gpt-5.2",   # o el modelo que uses
    messages=[
        {"role":"system","content":"Return only valid JSON that matches the
schema."},
        {"role":"user","content":PROMPT_V2}
    ],
    temperature=0.0
)

raw = resp.choices[0].message.content.strip()

# Validación JSON rápida
data = json.loads(raw)

with open(LLM_OUT_V2, "w", encoding="utf-8") as f:
```

```
    json.dump(data, f, ensure_ascii=False, indent=2)

print("✅ Saved:", LLM_OUT_V2)
print("Skills:", len(data.get("skills", [])))
print("Context terms:", len(data.get("context_terms", [])))
print("Discarded terms:", len(data.get("discarded_terms", [])))
```

```
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/llm_skills_output_v2.json
Skills: 8
Context terms: 70
Discarded terms: 77
```

In [ ]:

```
import os, json
import pandas as pd

BASE = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"
LLM_OUT_V2 = os.path.join(BASE, "llm_skills_output_v2.json")
DICT_OUT_V2 = os.path.join(BASE, "skill_dictionary_v2.csv")

with open(LLM_OUT_V2, "r", encoding="utf-8") as f:
    data = json.load(f)

rows = []

for s in data.get("skills", []):
    skill = s.get("skill_name","").strip()
    conf  = s.get("confidence", None)
    notes = s.get("notes","")
    for t in s.get("direct_evidence_terms", []):
        rows.append({"skill_name": skill, "term": t, "confidence": conf,
"notes": notes, "match_type": "direct"})
    for t in s.get("supporting_evidence_terms", []):
        rows.append({"skill_name": skill, "term": t, "confidence": conf,
"notes": notes, "match_type": "support"})

for bucket, mtype in [
    ("context_terms","context"),
    ("method_terms","method"),
    ("discarded_terms","discarded"),
    ("ambiguous_terms","ambiguous")
]:
    for obj in data.get(bucket, []):
        rows.append({
            "skill_name": "",
            "term": obj.get("term",""),
            "confidence": "",
            "notes": obj.get("reason",""),
            "match_type": mtype
        })

df_dict = pd.DataFrame(rows)
df_dict["term"] = df_dict["term"].astype(str).str.strip()
df_dict = df_dict[df_dict["term"].str.len() > 0]

df_dict.to_csv(DICT_OUT_V2, index=False)
```

```
print("✅ Saved:", DICT_OUT_V2)
df_dict.head(20)
```

```
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/skill_dictionary_v2.csv
```

	skill_name	term	confidence	notes	match_type
0	abstraction	abstraction	0.72	Abstraction is explicitly present and is suppo...	direct
1	abstraction	computational thinking skills	0.72	Abstraction is explicitly present and is suppo...	direct
2	abstraction	ct skills	0.72	Abstraction is explicitly present and is suppo...	support
3	abstraction	thinking skills	0.72	Abstraction is explicitly present and is suppo...	support
4	decomposition	decomposition	0.72	Decomposition is explicitly present and aligns...	direct
5	decomposition	computational thinking skills	0.72	Decomposition is explicitly present and aligns...	direct
6	decomposition	ct skills	0.72	Decomposition is explicitly present and aligns...	support
7	decomposition	thinking skills	0.72	Decomposition is explicitly present and aligns...	support
8	debugging	debugging	0.7	Debugging is explicitly present and is commonl...	direct
9	debugging	computational thinking skills	0.7	Debugging is explicitly	direct

	skill_name	term	confidence	notes	match_type
				present and is commonl...	
10	debugging	ct skills	0.7	Debugging is explicitly present and is commonl...	support
11	debugging	thinking skills	0.7	Debugging is explicitly present and is commonl...	support
12	algorithmic thinking	algorithmic thinking	0.78	Algorithmic thinking is directly named and rei...	direct
13	algorithmic thinking	computational thinking skills	0.78	Algorithmic thinking is directly named and rei...	direct
14	algorithmic thinking	algorithmic	0.78	Algorithmic thinking is directly named and rei...	support
15	algorithmic thinking	ct skills	0.78	Algorithmic thinking is directly named and rei...	support
16	problem solving skills	problem solving skills	0.76	Problem solving skills are directly stated wit...	direct
17	problem solving skills	solving skills	0.76	Problem solving skills are directly stated wit...	direct
18	problem solving skills	problem solving	0.76	Problem solving skills are directly stated wit...	support
19	problem solving skills	thinking skills	0.76	Problem solving skills are directly stated wit...	support

In [ ]:

```
# ============================================================
# BLOQUE C (v2): ct_membership_score_v2 usando skill_dictionary_v2.csv
# - pondera hits: direct=1.0, support=0.5
# - conserva señales base (ct_exact / ct_context / ct_concepts) + negativos
# - genera: df_master_enriched_ctv2.csv
# ============================================================

import os, re
import pandas as pd
import numpy as np

BASE = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"
MASTER_IN  = os.path.join(BASE, "df_master_enriched.csv")
DICT_IN    = os.path.join(BASE, "skill_dictionary_v2.csv")   # 👈 usa el v2
MASTER_OUT = os.path.join(BASE, "df_master_enriched_ctv2.csv")

df = pd.read_csv(MASTER_IN, low_memory=False)
df_dict = pd.read_csv(DICT_IN)

def norm_text(x):
    if pd.isna(x):
        return ""
    x = str(x).lower()
    x = re.sub(r"http\S+", " ", x)
    x = re.sub(r"[^a-z0-9\s\-]", " ", x)
    x = re.sub(r"\s+", " ", x).strip()
    return x

# ------------------------------------------------------------
# 1) Texto canónico: título + abstracts + keywords
# ------------------------------------------------------------
for col in ["title","abstract_scopus","abstract_wos",
            "author_keywords_scopus","index_keywords_scopus",
            "author_keywords_wos","keywords_plus_wos"]:
    if col not in df.columns:
        df[col] = np.nan

df["text_corpus"] = (
    df["title"].fillna("").apply(norm_text) + " " +
    df["abstract_scopus"].fillna("").apply(norm_text) + " " +
    df["abstract_wos"].fillna("").apply(norm_text) + " " +
    df["author_keywords_scopus"].fillna("").apply(norm_text) + " " +
    df["index_keywords_scopus"].fillna("").apply(norm_text) + " " +
    df["author_keywords_wos"].fillna("").apply(norm_text) + " " +
    df["keywords_plus_wos"].fillna("").apply(norm_text)
)

# ------------------------------------------------------------
# 2) Conceptos: OpenAlex + Dimensions (si existen)
# ------------------------------------------------------------
if "openalex_top_concepts" not in df.columns:
    df["openalex_top_concepts"] = np.nan
if "dimensions_top_concepts" not in df.columns:
    df["dimensions_top_concepts"] = np.nan

df["concepts_corpus"] = (
    df["openalex_top_concepts"].fillna("").apply(norm_text) + " " +
```

```
    df["dimensions_top_concepts"].fillna("").apply(norm_text)
).str.strip()

# ------------------------------------------------------------
# 3) Construir diccionario de términos -> skills con pesos
#    direct=1.0, support=0.5
# ------------------------------------------------------------
WEIGHT_MAP = {"direct": 1.0, "support": 0.5}

skills_terms =
df_dict[df_dict["match_type"].isin(["direct","support"])].copy()
skills_terms["term_norm"] =
skills_terms["term"].astype(str).str.lower().str.strip()
skills_terms["w"] = skills_terms["match_type"].map(WEIGHT_MAP).fillna(0.5)

# term -> lista de (skill, weight)
term2skill = {}
for _, r in skills_terms.iterrows():
    t = r["term_norm"]
    sk = str(r["skill_name"]).strip()
    w = float(r["w"])
    if not t or not sk:
        continue
    term2skill.setdefault(t, [])
    term2skill[t].append((sk, w))

# ordenar términos largos primero para mejorar matching por substring
all_terms = sorted(term2skill.keys(), key=len, reverse=True)

def extract_skill_hits_with_weights(text):
    """
    Devuelve:
      - hits: dict skill -> max_weight_observed
      - terms_matched: lista de términos que matchearon (para auditoría)
    """
    hits = {}
    matched_terms = []
    for t in all_terms:
        if t and t in text:
            matched_terms.append(t)
            for (sk, w) in term2skill[t]:
                # guardamos el máximo peso observado por skill
                hits[sk] = max(hits.get(sk, 0.0), w)
    return hits, matched_terms

# ejecutar extracción
skills_hit_list = []
skills_hit_terms = []
skills_weight_sum = []
skills_hit_count = []

for txt in df["text_corpus"].tolist():
    hits, matched_terms = extract_skill_hits_with_weights(txt)
    # suma de pesos (capada) + conteo skills
    wsum = float(sum(hits.values()))
    skills_weight_sum.append(wsum)
    skills_hit_count.append(len(hits))
```

```
    skills_hit_list.append("; ".join(sorted(hits.keys())))
    skills_hit_terms.append("; ".join(sorted(set(matched_terms))))

df["skills_hit_list_v2"] = skills_hit_list
df["skills_hit_terms_v2"] = skills_hit_terms
df["n_skills_hit_v2"] = skills_hit_count
df["skills_weight_sum_v2"] = skills_weight_sum

# normalización del score de skills:
# capamos a 6 puntos de peso (equivale a ~6 skills direct)
df["skills_score_v2"] = np.minimum(df["skills_weight_sum_v2"], 3.0) / 3.0

# ------------------------------------------------------------
# 4) Señales base CT (0..3) + negativos
# ------------------------------------------------------------
NEGATIVE_PATTERNS = [
    "computational modeling", "computational model", "computational
simulation",
    "benchmarking the computation", "parallel computation", "incompressible
flows",
    "prefrontal cortex", "neuroscience", "wireless", "6g", "ghz"
]

def build_ct_score(row):
    text_all = row["text_corpus"]
    concepts = row["concepts_corpus"]

    ct_exact = ("computational thinking" in text_all)

    edu_hit = any(w in text_all for w in [

"education","educational","teacher","teachers","student","students",
        "school","curriculum","classroom","k-12","primary","secondary"
    ])
    skill_hit = any(w in text_all for w in [
        "programming","coding","robotics","scratch","stem","steam",
        "abstraction","decomposition","algorithm","algorithms",
        "debugging","troubleshooting","problem solving"
    ])
    ct_context = bool(edu_hit and skill_hit)

    ct_concepts = (("education" in concepts) and ("computer science" in
concepts)) or ("computational thinking" in concepts)

    ct_negative = any(p in text_all for p in NEGATIVE_PATTERNS)

    base = int(ct_exact) + int(ct_context) + int(ct_concepts)
    return pd.Series({"ct_score": base, "ct_negative": ct_negative})

tmp = df.apply(build_ct_score, axis=1)
df["ct_score"] = tmp["ct_score"].astype(int)
df["ct_negative"] = tmp["ct_negative"].astype(bool)

df["ct_score_norm"] = df["ct_score"] / 3.0

# ------------------------------------------------------------
# 5) CT membership v2 (mezcla señales base + skills ponderadas)
```

```
# ------------------------------------------------------------
df["ct_membership_score_v2"] = (
    0.45 * df["ct_score_norm"] +
    0.55 * df["skills_score_v2"]
)

# penalizar negativos fuertes
df.loc[df["ct_negative"], "ct_membership_score_v2"] *= 0.6

def label_v2(score, neg):
    if neg:
        return "noise"
    if score >= 0.80:
        return "core"
    if score >= 0.50:
        return "broad"
    return "none"

df["ct_label_v2"] = [label_v2(s, n) for s, n in
zip(df["ct_membership_score_v2"], df["ct_negative"])]

print("CT v2 distribution:\n", df["ct_label_v2"].value_counts())
print("Skills hit (v2) describe:\n", df["n_skills_hit_v2"].describe())
print("Skills weight sum (v2) describe:\n",
df["skills_weight_sum_v2"].describe())

# ------------------------------------------------------------
# 6) Guardar
# ------------------------------------------------------------
df.to_csv(MASTER_OUT, index=False)
print("✅ Saved:", MASTER_OUT)

# Quick audit: top 20 por skills_weight_sum_v2
df.sort_values("skills_weight_sum_v2", ascending=False)[

["doi","year","title","skills_weight_sum_v2","n_skills_hit_v2","skills_hit_
list_v2",
     "ct_score","ct_negative","ct_membership_score_v2","ct_label_v2"]
].head(20)
```

```
CT v2 distribution:
 ct_label_v2
none     6325
broad    1904
core     1800
noise     802
Name: count, dtype: int64
Skills hit (v2) describe:
 count    10831.000000
mean         1.385745
std          1.893332
min          0.000000
25%          0.000000
50%          1.000000
75%          2.000000
max          8.000000
Name: n_skills_hit_v2, dtype: float64
```

```
Skills weight sum (v2) describe:
 count    10831.000000
mean         1.028991
std          1.520403
min          0.000000
25%          0.000000
50%          0.500000
75%          1.000000
max          7.000000
Name: skills_weight_sum_v2, dtype: float64
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/df_master_enriched_ctv2.csv
```

	doi	year	title	skills_weig
1679	10.1007/s10639-025-13386-y	2025	Co-designing to develop computational thinking...	7.0
7556	10.1145/3724363.3729066	2025	Fostering Computational Thinking in CS1 throug...	7.0
5792	10.1109/vlhcc.2015.7357215	2015	Collaboration and Computational Thinking: A cl...	7.0
1175	10.1007/978-3-642-23456-9_39	2011	Facilitating computational thinking through ga...	7.0
3334	10.1051/e3sconf/202453805034	2024	Retracted:Methods of using STEAM technologies ...	6.5
6334	10.1145/2591708.2591724	2014	Early validation of computational thinking pat...	6.5
6550	10.1145/3130859.3131335	2017	CodeFruits: Teaching computational thinking sk...	6.5
8551	10.17275/per.23.26.10.2	2023	An Evaluation of the Effect of Activity-Based ...	6.5
3703	10.1080/09523987.2023.2324600	2023	Redefining the creative digital	6.5

	doi	year	title	skills_weig
			project for 8t...	
7472	10.1145/3669947.3669951	2024	The Influence of CodeCombat on Computational T...	6.5
2357	10.1016/j.cexr.2025.100102	2025	Group dynamics in collaborative learning: Impa...	6.5
296	10.1007/978-3-030-05270-6_5	2019	ChildProgramming evolution, a method to increa...	6.5
1677	10.1007/s10639-025-13367-1	2025	Scaffolding self-regulation in project-based p...	6.5
9753	10.3390/su16229839	2024	Examining Teachers' Computational Thinking Ski...	6.5
2380	10.1016/j.chb.2019.03.018	2020	Developing young children's computational thin...	6.5
4292	10.1109/access.2018.2877417	2018	Developing Computational Thinking Skills in Ad...	6.5
9440	10.3389/fpsyg.2022.875382	2022	Combined Effects of Block-Based Programming an...	6.5
1635	10.1007/s10639-024-12550-0	2024	Which approach is effective: Comparing problem...	6.5
2518	10.1016/j.compedu.2023.104794	2023	Enhancing student's computational thinking ski...	6.5
5930	10.1111/jcal.12845	2023	Effectiveness of collaboration in	6.5

	doi	year	title	skills_weig
			developing c...	

In [ ]:
```
df[(df["ct_label_v2"]=="core") & (df["n_skills_hit_v2"]==0)][
    ["doi","year","title","ct_score","ct_membership_score_v2"]
].head(30)
```

	doi	year	title	ct_score	ct_membership_score_v2

In [ ]:
```
df[(df["ct_label_v2"]=="none") & (df["skills_weight_sum_v2"]>=2.0)][

["doi","year","title","skills_hit_list_v2","skills_weight_sum_v2","ct_score
","ct_negative"]
].head(30)
```

	doi	year	title	skills_hi
61	10.1002/bmb.20914	2015	Writing throughout the biochemistry curriculum...	abstracti debuggi decompo problem.
62	10.1002/bmb.20975	2016	Development of a structured undergraduate rese...	abstracti debuggi decompo problem.
236	10.1007/11758525_20	2006	Involving undergraduates in computational scie...	abstracti debuggi decompo problem.
1432	10.1007/bf02954535	1993	The computational metaphor and computer criticism	abstracti debuggi decompo problem.
1603	10.1007/s10639-023-11930-2	2024	Promoting students' higher order thinking in v...	abstracti collabora learning; debuggi
2053	10.1007/s12539-016-0170-y	2018	Binding Patterns Associated Aß-HSP60 p458 Conj...	abstracti debuggi decompo problem.

	doi	year	title	skills_hi
3182	10.1021/acs.jchemed.0c01351	2021	Using a Spreadsheet-Based Simulation to Practi...	abstracti debuggi decompo problem.
3187	10.1021/acs.jchemed.3c01352	2024	An Inquiry-Based Computational Chemistry Activ...	collabora learning; solving s
3203	10.1021/bk-2019-1312.ch004	2019	Modeling Reaction Energies and Exploring Noble...	abstracti debuggi decompo problem.
3379	10.1063/1.5041684	2018	Measurement invariance of perceived learning o...	abstracti collabora learning; debuggi
3390	10.1063/5.0051231	2021	Turning undergraduate research assistants into...	abstracti debuggi decompo problem.
3452	10.1063/5.0290369	2025	High school students' critical thinking in sol...	abstracti collabora learning; debuggi
3965	10.1080/18117295.2023.2265241	2024	Teaching Mathematics with Digital Technologies...	abstracti debuggi decompo problem.
4412	10.1109/caibda65784.2025.11183347	2025	A Lightweight Detector: Zero-Shot Detection of...	abstracti debuggi decompo problem.
4432	10.1109/cenim.2018.8711088	2018	Analysis of Brain Tissue and Cerebrospinal Flu...	abstracti debuggi decompo problem.
5142	10.1109/icirca65293.2025.11089784	2025	Early Detection and Stage Prediction of Alzhei...	abstracti debuggi decompo problem.

	doi	year	title	skills_hi
5651	10.1109/tale48000.2019.9225911	2019	Design Thinking for Computational Creativity -...	collabora learning; design
6029	10.1115/fedsm2020-20161	2020	Unified assessment approach for courses with s...	abstracti debuggi decompo problem.
6102	10.1142/13085	2023	Conjuring with Computation: A MANUAL OF MAGIC ...	abstracti decompo
7331	10.1145/3613905.3648110	2024	A Mystery for You: A fact-checking game enhanc...	abstracti debuggi decompo problem.
7387	10.1145/3628096.3628747	2023	Investigating the Efficacy of Large Language M...	abstracti debuggi decompo problem.
7641	10.1155/2024/7914178	2024	Extreme Gradient Boosting Beats In-Silico Iden...	abstracti debuggi decompo problem.
8112	10.1300/j122v17n03_05	1999	New search and navigation techniques in the di...	abstracti debuggi decompo problem.
8116	10.13187/ejced.2017.3.414	2017	Lower-order mathematical thinking skills in fi...	abstracti debuggi decompo problem.
8279	10.1504/ijram.2011.043697	2011	Optimisation-based decision-making for complex...	abstracti decompo
8651	10.18576/isl/120915	2023	Educational Practices Within High Schools and ...	abstracti debuggi decompo problem.
8730	10.18653/v1/2024.findings-emnlp.661	2024	Designing Logic Pattern	abstracti debuggi

	doi	year	title	skills_hi
			Templates for Counter-...	decompo problem.
8756	10.18653/v1/2025.acl-srw.8	2025	Can Multi-turn Self-refined Single Agent LMs w...	algorithn thinking; solving s
8791	10.1890/0012-9658(2003)084[1412:tettwu]2.0.co;2	2003	Training ecologists to think with uncertainty ...	abstracti debuggi decompo problem.
8872	10.2174/1573405618666220823115848	2023	A Methodical and Performance-based Investigati...	abstracti debuggi decompo problem.

In [ ]:
```
def skill_weight_adjust(skill_name):
    s = skill_name.lower()
    ct_markers = [
        "computational", "algorithm", "programming",
        "coding", "robotic", "data"
    ]
    if any(m in s for m in ct_markers):
        return 1.0
    return 0.25
```

In [ ]:
```
skills_terms["w"] = skills_terms["skill_name"].apply(skill_weight_adjust)
```

In [ ]:
```
def label_v2(score, neg, ct_score):
    if neg:
        return "noise"
    if score >= 0.78 and ct_score >= 1:
        return "core"
    if score >= 0.45:
        return "broad"
    return "none"
```

In [ ]:
```
df["ct_label_v2"] = [
    label_v2(s, n, c)
    for s, n, c in zip(
        df["ct_membership_score_v2"],
        df["ct_negative"],
        df["ct_score"]
    )
]
```

In [ ]:
```
PROMPT_V3 = f"""
You are classifying candidate terms extracted from scientific publications
```

```
related to Computational Thinking (CT).
Your job is to produce TWO skill lists:

1) ct_core_skills: CT-specific skills/practices/components. These must be
clearly tied to CT literature and NOT be generic cognitive skills.
2) transversal_skills: general education/cognitive skills that may appear
in CT papers but are NOT CT-defining by themselves.

IMPORTANT RULES (must follow):
- Use ONLY the provided candidate terms as evidence. Do NOT invent new
skills not supported by the candidate terms.
- A skill can be included only if you can cite 1–6 evidence terms from the
list.
- Put generic items such as "problem solving", "critical thinking",
"collaboration", "scaffolding", "higher order thinking"
  into transversal_skills unless they are explicitly framed as
computational/algorithmic/programming/CT.
- If a term is ambiguous, mark it as transversal or exclude it.
- Return valid JSON ONLY (no markdown), exactly matching the schema below.

Candidate terms (top {len(CAND_TERMS)}):
{CAND_TERMS}

JSON schema to return:
{{
  "ct_core_skills": [
    {{
      "skill_name": "string",
      "evidence_terms": ["string", "..."],
      "confidence": 0.0,
      "notes": "short justification referencing CT specificity"
    }}
  ],
  "transversal_skills": [
    {{
      "skill_name": "string",
      "evidence_terms": ["string", "..."],
      "confidence": 0.0,
      "notes": "short justification why it is transversal"
    }}
  ],
  "excluded_terms": [
    {{
      "term": "string",
      "reason": "string (e.g., too generic, domain-specific non-CT,
unclear)"
    }}
  ]
}}
"""
```

In [ ]:

```
import os
import pandas as pd

BASE = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"
CAND_PATH = os.path.join(BASE, "nlp_skill_candidates.csv")
```

```
cand = pd.read_csv(CAND_PATH)

# 👇 coge top 120 (ajústalo: 80–150 suele ir bien)
TOP_N = 120
CAND_TERMS = cand["term"].head(TOP_N).dropna().astype(str).tolist()

# Lo pasamos a texto enumerado para el prompt
CAND_TERMS = "\n".join([f"- {t}" for t in CAND_TERMS])
print(CAND_TERMS.splitlines()[:10])
```

```
['- programming', '- ct', '- students', '- education', '- learning',
'- skills', '- teaching', '- teachers', '- school', '- educational']
```

In [ ]:
```
import os, json
from openai import OpenAI

client = OpenAI(api_key=os.environ.get("OPENAI_API_KEY"))

MODEL = "gpt-5.2"  # ajusta al nombre disponible en tu cuenta

resp = client.chat.completions.create(
    model=MODEL,
    messages=[
        {"role": "system", "content": "You are classifying candidate terms.
Return only valid JSON that matches the schema, with no markdown or extra
text."},
        {"role": "user", "content": PROMPT_V3}
    ],
    response_format={"type": "json_object"},
    temperature=0.2
)

out = resp.choices[0].message.content.strip()
data = json.loads(out)

OUT_JSON = os.path.join(BASE, "llm_skills_output_v3.json")
with open(OUT_JSON, "w", encoding="utf-8") as f:
    json.dump(data, f, ensure_ascii=False, indent=2)

print("✅ Saved:", OUT_JSON)
print("CT-core skills:", len(data.get("ct_core_skills", [])))
print("Transversal skills:", len(data.get("transversal_skills", [])))
print("Excluded terms:", len(data.get("excluded_terms", [])))
```

```
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/llm_skills_output_v3.json
CT-core skills: 4
Transversal skills: 7
Excluded terms: 99
```

In [ ]:
```
import pandas as pd

with open(os.path.join(BASE, "llm_skills_output_v3.json"), "r",
encoding="utf-8") as f:
    data = json.load(f)
```

```
rows = []

def add_rows(group_name, items):
    for item in items:
        skill = item.get("skill_name")
        conf = item.get("confidence", None)
        notes = item.get("notes", "")
        for term in item.get("evidence_terms", []):
            rows.append({
                "skill_group": group_name,          # ct_core_skills |
transversal_skills
                "skill_name": skill,
                "term": term,
                "confidence": conf,
                "notes": notes,
                "match_type": "evidence_term"
            })

add_rows("ct_core_skills", data.get("ct_core_skills", []))
add_rows("transversal_skills", data.get("transversal_skills", []))

df_skill_v3 = pd.DataFrame(rows).drop_duplicates()

OUT_CSV = os.path.join(BASE, "skill_dictionary_v3.csv")
df_skill_v3.to_csv(OUT_CSV, index=False)

print("✅ Saved:", OUT_CSV, "| rows:", len(df_skill_v3))
df_skill_v3.head(10)
```

```
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/skill_dictionary_v3.csv | rows: 20
```

	skill_group	skill_name	term	confidence	not
0	ct_core_skills	Abstraction	abstraction	0.86	Abstraction is a canonical CT component explic.
1	ct_core_skills	Algorithmic thinking	algorithmic thinking	0.90	Algorithmic thinkir is a CT-defining practice...
2	ct_core_skills	Algorithmic thinking	algorithmic	0.90	Algorithmic thinkir is a CT-defining practice...
3	ct_core_skills	Debugging	debugging	0.88	Debugging is a cor CT/programming practice co...
4	ct_core_skills	Programming / Coding	programming	0.78	Programming/codi practices are frequently us...
5	ct_core_skills	Programming / Coding	coding	0.78	Programming/codi practices are frequently us...

	skill_group	skill_name	term	confidence	not
6	ct_core_skills	Programming / Coding	computer programming	0.78	Programming/codi practices are frequently us...
7	ct_core_skills	Programming / Coding	based programming	0.78	Programming/codi practices are frequently us...
8	ct_core_skills	Programming / Coding	visual programming	0.78	Programming/codi practices are frequently us...
9	ct_core_skills	Programming / Coding	block based	0.78	Programming/codi practices are frequently us...

In [ ]:
```
df_skill_v3["base_weight"] = df_skill_v3["skill_group"].map({
    "ct_core_skills": 1.0,
    "transversal_skills": 0.25
}).fillna(0.25)

OUT_CSV_W = os.path.join(BASE, "skill_dictionary_v3_weighted.csv")
df_skill_v3.to_csv(OUT_CSV_W, index=False)
print("✅ Saved:", OUT_CSV_W)
```

```
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/skill_dictionary_v3_weighted.csv
```

In [150]:
```
import os
import pandas as pd
import numpy as np

BASE = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"

MASTER_PATH   = os.path.join(BASE, "df_master_enriched_ctv2.csv")
EDGES_PATH    = os.path.join(BASE, "openalex_citation_edges.csv")
OA_BY_DOI_PATH= os.path.join(BASE, "openalex_by_doi.csv")

OUT_NODES = os.path.join(BASE, "net_paper_nodes.csv")
OUT_EDGES = os.path.join(BASE, "net_citation_edges_filtered.csv")

# ------------------------------
# 1) Load master + edges + oa mapping
# ------------------------------
df = pd.read_csv(MASTER_PATH, low_memory=False)
edges = pd.read_csv(EDGES_PATH, low_memory=False)
oa = pd.read_csv(OA_BY_DOI_PATH, low_memory=False)

print("Loaded master:", df.shape, "| unique DOIs:", df["doi"].nunique())
print("Loaded edges:", edges.shape, "| cols:", list(edges.columns))
print("Loaded openalex_by_doi:", oa.shape, "| cols:", list(oa.columns))
```

```
# ------------------------------
# 2) Build oa_id -> doi map
#    Expect oa_id column in openalex_by_doi.csv
# ------------------------------
if "oa_id" not in oa.columns:
    raise ValueError("No encuentro columna 'oa_id' en openalex_by_doi.csv.
Revisa el archivo.")

oa["oa_id"] = oa["oa_id"].astype(str).str.strip()
oa["doi"]   = oa["doi"].astype(str).str.strip()

oa_map = dict(zip(oa["oa_id"], oa["doi"]))  # oa_id (work id URL) -> doi
print("OA map size:", len(oa_map))

# ------------------------------
# 3) Canonical impact score for ranking nodes
# ------------------------------
def to_num(s): return pd.to_numeric(s, errors="coerce")

for col in [
    "openalex_cited_by_count",
    "dimensions_times_cited",
    "scopus_citations",
    "wos_citations_core",
    "wos_citations_all",
    "altmetric_score",
    "mendeley_reader_count",
]:
    if col in df.columns:
        df[col] = to_num(df[col])

df["citations_rank"] = 0
if "openalex_cited_by_count" in df.columns:
    df["citations_rank"] = df["openalex_cited_by_count"].fillna(0)
elif "dimensions_times_cited" in df.columns:
    df["citations_rank"] = df["dimensions_times_cited"].fillna(0)

# ------------------------------
# 4) CT subset (core+broad) & select top N
# ------------------------------
if "ct_label_v2" not in df.columns:
    raise ValueError("No encuentro 'ct_label_v2' en master.")

df_ct = df[df["ct_label_v2"].isin(["core", "broad"])].copy()
print("CT subset (core+broad):", df_ct.shape)

TOP_N = 1200
df_top = df_ct.sort_values(["citations_rank", "year"], ascending=[False,
False]).head(TOP_N).copy()
node_set = set(df_top["doi"].dropna().astype(str))
print("Selected nodes:", len(node_set))

# ------------------------------
# 5) Convert edges: cited_openalex_work_id -> cited_doi via map
# ------------------------------
if "citing_doi" not in edges.columns or "cited_openalex_work_id" not in
edges.columns:
```

```
    raise ValueError(f"Edges no tienen columnas esperadas. Columnas:
{list(edges.columns)}")

edges["citing_doi"] = edges["citing_doi"].astype(str).str.strip()
edges["cited_openalex_work_id"] =
edges["cited_openalex_work_id"].astype(str).str.strip()

edges["cited_doi"] = edges["cited_openalex_work_id"].map(oa_map)

# Report mapping success
mapped = edges["cited_doi"].notna().mean()
print(f"Mapped cited_openalex_work_id -> DOI: {mapped*100:.2f}%")

# Drop edges where target DOI not known
edges2 = edges.dropna(subset=["cited_doi"]).copy()

# ------------------------------
# 6) Induced subgraph filter (source+target in node_set)
# ------------------------------
edges_f = edges2[edges2["citing_doi"].isin(node_set) &
edges2["cited_doi"].isin(node_set)].copy()
edges_f = edges_f.rename(columns={"citing_doi": "source", "cited_doi":
"target"})
edges_f = edges_f[["source", "target"]].copy()
edges_f["edge_source"] = "openalex"

print("Filtered edges:", edges_f.shape)

# ------------------------------
# 7) Build nodes table (tooltips/styling)
# ------------------------------
NODE_COLS = [
    "doi",
    "title",
    "year",
    "journal",
    "ct_label_v2",
    "ct_membership_score_v2",
    "ct_score",
    "ct_negative",
    "citations_rank",
    "openalex_cited_by_count",
    "dimensions_times_cited",
    "altmetric_score",
    "mendeley_reader_count",
    "openalex_top_concepts",
]
node_cols_exist = [c for c in NODE_COLS if c in df_top.columns]
nodes = df_top[node_cols_exist].copy()

if "year" in nodes.columns:
    nodes["year"] = pd.to_numeric(nodes["year"],
errors="coerce").astype("Int64")

if "title" in nodes.columns:
    nodes["label"] = nodes["title"].astype(str).str.slice(0, 140)
```

```
# ------------------------------
# 8) Save
# ------------------------------
nodes.to_csv(OUT_NODES, index=False)
edges_f.to_csv(OUT_EDGES, index=False)

print("✅ Saved nodes:", OUT_NODES, "| rows:", len(nodes))
print("✅ Saved edges:", OUT_EDGES, "| rows:", len(edges_f))

nodes.head(3), edges_f.head(3)
```

```
Loaded master: (10831, 88) | unique DOIs: 10830
Loaded edges: (376622, 3) | cols: ['citing_doi',
'cited_openalex_work_id', 'source']
Loaded openalex_by_doi: (10830, 17) | cols: ['doi', 'oa_id',
'oa_type', 'oa_language', 'oa_publication_year',
'oa_publication_date', 'oa_cited_by_count',
'oa_referenced_works_count', 'oa_is_oa', 'oa_status',
'oa_any_repo_fulltext', 'oa_countries_distinct_count',
'oa_institutions_distinct_count', 'oa_host_org_name',
'oa_source_name', 'oa_n_concepts', 'oa_top_concepts']
OA map size: 10568
CT subset (core+broad): (3704, 89)
Selected nodes: 1200
Mapped cited_openalex_work_id -> DOI: 15.02%
Filtered edges: (9970, 4)
✅ Saved nodes: /content/drive/My Drive/Colab Notebooks/PEC3-
Visualizacion-UOC/net_paper_nodes.csv | rows: 1200
✅ Saved edges: /content/drive/My Drive/Colab Notebooks/PEC3-
Visualizacion-UOC/net_citation_edges_filtered.csv | rows: 9970
```

Out[150]:

```
(                          doi  \
 6138   10.1145/1118178.1118215
 9218  10.3102/0013189x12463051
 4203    10.1098/rsta.2008.0118

                                                 title  year  \
 6138                             Computational thinking  2006
 9218  Computational Thinking in K-12: A Review of th...  2013
 4203  Computational thinking and thinking about comp...  2008

                                                journal ct_label_v2
\
 6138                          Communications of the ACM       broad
 9218                             Educational Researcher       broad
 4203  Philosophical Transactions of the Royal Societ...       broad

       ct_membership_score_v2  ct_score  ct_negative  citations_rank
\
 6138                0.608333         1        False          6550.0
 9218                0.541667         3        False          2205.0
 4203                0.633333         3        False          1664.0

       openalex_cited_by_count  dimensions_times_cited
altmetric_score  \
 6138                   6550.0                  6009.0
194.20
```

```
 9218                  2205.0                     NaN
69.20
 4203                  1664.0                  1364.0
45.35

      mendeley_reader_count  \
 6138                 3569.0
 9218                 2310.0
 4203                 2251.0

                                openalex_top_concepts  \
 6138                                Computer science
 9218  Computational thinking; State (computer scienc...
 4203  Field (mathematics); Computational thinking; C...

                                                   label
 6138                            Computational thinking
 9218  Computational Thinking in K-12: A Review of th...
 4203  Computational thinking and thinking about comp...  ,
                         source    source
target  \
 19352  10.3102/00346543241241327  openalex
10.1016/j.robot.2015.10.008
 19357  10.3102/00346543241241327  openalex
10.1016/j.compedu.2019.04.013
 19360  10.3102/00346543241241327  openalex
10.3102/0034654317710096

      edge_source
 19352    openalex
 19357    openalex
 19360    openalex  )
```

In [155]:

```
import os
import pandas as pd

BASE = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"
OUT_EDGES_FINAL = os.path.join(BASE, "net_citation_edges_filtered.csv")
OUT_EDGES_FINAL_YEAR = os.path.join(BASE,
"net_citation_edges_filtered_with_year.csv")

# Assume edges_f comes from the previous cell with columns ['edge_source',
'target']
# where 'edge_source' actually holds the citing DOIs.
# This is based on the print output in the error cell.

# Rename 'edge_source' (which contains citing DOIs) to 'source'
edges_f_cleaned = edges_f.rename(columns={'edge_source': 'source'}).copy()

# Add the 'edge_source' column with the literal string "openalex"
# This column was implicitly dropped or confused in the previous step.
edges_f_cleaned['edge_source'] = 'openalex'

print("Columns after cleaning:", list(edges_f_cleaned.columns))

# --- E) Construye el mapa DOI -> year (asegura string) ---
```

```
# 'nodes' DataFrame is available from the previous cell's execution
context.
year_map = dict(zip(nodes["doi"].astype(str), nodes["year"]))

# --- F) Mapear year_source ---
edges_final = edges_f_cleaned.copy()
edges_final["year_source"] =
edges_final["source"].astype(str).map(year_map)

# --- G) Guardar ---
edges_final[["source", "target", "edge_source",
"year_source"]].to_csv(OUT_EDGES_FINAL_YEAR, index=False)
print("✅ Saved:", OUT_EDGES_FINAL_YEAR, "| shape:", edges_final.shape)

edges_f_cleaned[["source", "target",
"edge_source"]].to_csv(OUT_EDGES_FINAL, index=False)
print("✅ Saved:", OUT_EDGES_FINAL, "| shape:", edges_f_cleaned.shape)
```

```
Columns after cleaning: ['source', 'target', 'edge_source']
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/net_citation_edges_filtered_with_year.csv | shape: (9970, 4)
✅ Saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/net_citation_edges_filtered.csv | shape: (9970, 3)
```

In [156]:

```
import os
import pandas as pd

BASE = "/content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-UOC/"

NODES_IN = os.path.join(BASE, "net_paper_nodes.csv")
EDGES_IN = os.path.join(BASE, "net_citation_edges_filtered_with_year.csv")

nodes = pd.read_csv(NODES_IN, low_memory=False)
edges = pd.read_csv(EDGES_IN, low_memory=False)

print("Nodes:", nodes.shape, "| cols:", list(nodes.columns))
print("Edges:", edges.shape, "| cols:", list(edges.columns))

# Asegura tipos
nodes["doi"] = nodes["doi"].astype(str)
nodes["year"] = pd.to_numeric(nodes["year"],
errors="coerce").astype("Int64")

edges["source"] = edges["source"].astype(str)
edges["target"] = edges["target"].astype(str)
edges["year_source"] = pd.to_numeric(edges["year_source"],
errors="coerce").astype("Int64")

# ----------------------------
# Definir periodos
# ----------------------------
periods = {
    "p1_2006_2012": (2006, 2012),
    "p2_2013_2019": (2013, 2019),
    "p3_2020_2026": (2020, 2026),
}
```

```
def export_period(tag, y0, y1):
    # 1) nodos del periodo (por año del paper)
    n = nodes[(nodes["year"] >= y0) & (nodes["year"] <= y1)].copy()
    n_set = set(n["doi"].dropna().astype(str))

    # 2) edges inducidas (source/target dentro del periodo)
    e = edges[edges["source"].isin(n_set) &
edges["target"].isin(n_set)].copy()

    # 3) añade metadato periodo (útil para tooltips/filtros)
    n["period"] = tag
    e["period"] = tag

    # 4) rutas
    out_nodes = os.path.join(BASE, f"net_nodes_{tag}.csv")
    out_edges = os.path.join(BASE, f"net_edges_{tag}.csv")

    # 5) guarda (columnas limpias)
    # Nodos: deja columnas útiles (mantén las que existan)
    keep_nodes = [c for c in [
        "doi","label","title","year","journal","ct_label_v2",
        "ct_membership_score_v2","ct_score","citations_rank",
        "openalex_cited_by_count","dimensions_times_cited",
        "altmetric_score","mendeley_reader_count","openalex_top_concepts",
        "period"
    ] if c in n.columns]
    n[keep_nodes].to_csv(out_nodes, index=False)

    # Edges: deja columnas útiles
    keep_edges = [c for c in
["source","target","edge_source","year_source","period"] if c in e.columns]
    e[keep_edges].to_csv(out_edges, index=False)

    print(f"✅ {tag} | years {y0}–{y1} | nodes: {len(n)} | edges:
{len(e)}")
    print("   saved:", out_nodes)
    print("   saved:", out_edges)

for tag, (y0, y1) in periods.items():
    export_period(tag, y0, y1)
```

```
Nodes: (1200, 15) | cols: ['doi', 'title', 'year', 'journal',
'ct_label_v2', 'ct_membership_score_v2', 'ct_score', 'ct_negative',
'citations_rank', 'openalex_cited_by_count', 'dimensions_times_cited',
'altmetric_score', 'mendeley_reader_count', 'openalex_top_concepts',
'label']
Edges: (9970, 4) | cols: ['source', 'target', 'edge_source',
'year_source']
✅ p1_2006_2012 | years 2006–2012 | nodes: 49 | edges: 67
   saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/net_nodes_p1_2006_2012.csv
   saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/net_edges_p1_2006_2012.csv
✅ p2_2013_2019 | years 2013–2019 | nodes: 450 | edges: 1138
   saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/net_nodes_p2_2013_2019.csv
```

```
   saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/net_edges_p2_2013_2019.csv
✅ p3_2020_2026 | years 2020-2026 | nodes: 699 | edges: 2713
   saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/net_nodes_p3_2020_2026.csv
   saved: /content/drive/My Drive/Colab Notebooks/PEC3-Visualizacion-
UOC/net_edges_p3_2020_2026.csv
```